A comparison of Automatic License Plate Recognition approaches with different hardware setups

TM Buchler



orcid.org/0000-0002-8170-2578

Dissertation accepted in fulfillment of the requirements for the degree *Master of Science in Computer Science* at the North-West University

Supervisor: Prof JJ Greeff

Co-supervisor Prof G Drevin

Graduation: October 2026

Acknowledgements

I wish to express my sincere gratitude to the following:

My supervisors, **Prof Japie Greeff** and **Prof Gunther Drevin**, for their guidance, patience and invaluable feedback throughout this study.

My family, for their unwavering support and encouragement during the course of my studies.

My employer, **Sapconet**, for supporting this study both financially and through the practical guidance that inspired the research.

The **North-West University**, for giving me the opportunity to pursue this degree and for their patience throughout what has been a lengthy journey since 2021.

ABSTRACT

Automatic license plate recognition (ALPR) systems are widely used for traffic management, security and logistics, but most implementations rely on expensive specialised hardware or are evaluated on a single system configuration. There is limited research comparing different ALPR approaches across varying levels of hardware, and no established methodology for performing such comparisons consistently.

This study presents a reproducible methodology for comparing software-based ALPR approaches across different hardware platforms. The methodology is demonstrated by empirically comparing three ALPR pipelines — YOLO detection with Tesseract OCR, OpenALPR detection and recognition, and YOLO detection with YOLO recognition — across three hardware configurations: a low-level Raspberry Pi 4 B, an intermediate-level consumer desktop with an NVIDIA GTX 1080 Ti, and a high-level desktop with an NVIDIA RTX 3080 Ti. Accuracy was measured using multiple edit distance metrics, including Levenshtein, Damerau-Levenshtein, Hamming, Jaro-Winkler, Kendall Tau and longest common subsequence. Processing time per image was also captured across all configurations.

The results demonstrate that the YOLO-based detection and recognition pipeline significantly outperforms both the Tesseract and OpenALPR approaches, achieving up to 98.75% accuracy on a region-specific South African dataset across all hardware tiers. Traditional preprocessing techniques such as grayscaling and binarization provided minimal benefit when using the YOLO pipeline. The small YOLO model variants offered the best balance of accuracy and speed on intermediate and high-level hardware, while the nano or tiny variants were required for the Raspberry Pi, reducing processing time from approximately 3700ms to under 2000ms per image at the cost of approximately 4% accuracy.

The primary contribution of this study is the reproducible comparison methodology itself, which can be applied to any combination of ALPR software and hardware. Additionally, the study provides practical recommendations for deploying ALPR systems using free software on consumer-level hardware, with particular relevance to cost-sensitive environments such as small businesses, municipalities and researchers in South Africa.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION 1

1.1 Background 1

1.2 Problem statement 2

1.3 Scope of the study 3

1.3.1 Data 3

1.3.2 Software 3

1.4 Research paradigms and key concepts 4

1.4.1 Key concepts 4

1.4.2 Classifications 4

1.5 Research questions 5

1.5.1 Primary research question 5

1.5.2 Secondary research questions 5

1.6 Objectives of the study 6

1.6.1 Primary objective 6

1.6.2 Secondary objectives 6

1.7 Ethics 6

1.8 Dissertation overview 7

CHAPTER 2: LITERATURE REVIEW 8

2.1 Preprocessing 8

2.1.1 Enhancement and sharpening 8

2.1.2 Grayscaling 11

2.1.3 Binarization 12
2.1.4 Skeletonization 14
2.1.5 OpenCV 15
2.2 Artificial Intelligence 16
2.3 Machine learning 18
2.3.1 Artificial neural networks 18
2.3.2 Convolutional neural networks 20
2.3.3 Annotation 21
2.3.4 Neural network frameworks 22
2.3.4.1 TensorFlow 22
2.3.4.2 YOLO 24
2.4 ALPR 28
2.4.1 OpenALPR 29
2.4.2 Comparison of ALPR approaches 29
2.4.3 ALPR in South Africa 30
2.5 Character segmentation 31
2.6 Optical character recognition 32
2.6.1 Machine learning approaches to OCR 32
2.6.2 Tesseract 33
2.6.3 Comparison of OCR approaches 34
2.7 Edit distance 34
2.7.1 Hamming distance 35
2.7.2 Jaro-Winkler distance 35

2.7.3 Kendall Tau distance 36
2.7.4 Levenshtein distance 36
2.7.5 Damerau–Levenshtein distance 37
2.7.6 Longest common subsequence 37
2.7.7 Comparison of edit distance approaches 37
2.8 Image augmentation 38
2.9 Summary 40

CHAPTER 3: METHODOLOGY 41

3.1 Research methodology and paradigm 41
3.2 Experimental setup 41
3.2.1 Hardware setup 41
3.2.1.1 Low-level machine 42
3.2.1.2 Intermediate-level hardware 43
3.2.1.3 High-level hardware 43
3.2.1.4 Excluded hardware 43
3.2.2 Software setup 43
3.3 Experimental procedure 45
3.3.1 Data preparation 46
3.3.2 Training and validation 46
3.3.3 Performing the experiments 47
3.3.4 Data collection 47
3.3.5 Evaluation 48
3.4 Risks to validity 48

3.5 Summary 48

CHAPTER 4: RESULTS 50

4.1 Universal results 50

4.2 South African results 64

4.3 Summary 74

CHAPTER 5: CONCLUSIONS 75

5.1 Summary of findings 75

5.2 Research questions 76

5.2.1 Primary research question 76

5.2.2 Secondary research questions 76

5.3 Limitations 77

5.4 Future work 77

5.5 Contributions 78

REFERENCES 79

LIST OF TABLES

Table 2-1 Key OpenCV modules used in the study 16

Table 2-2 Comparison of edit distance algorithms 34

Table 3-1 Hardware configuration for the systems used in the study 42

Table 3-2 Installation instructions for the software used in the study 44

Table 4-1 Description of methods used for the initial tests 50

Table 4-2 Calibration with different YOLO configurations on high-level hardware 55

Table 4-3 Calibration with different YOLO configurations on low-level hardware 56

Table 4-4 Calibration with different YOLO configurations on intermediate-level hardware 57

Table 4-5 Non-ML OpenALPR results with preprocessing 58

Table 4-6 YOLO calibration results with 200, 400 and 600 images 61

Table 4-7 Results of tests run with YOLOv8s on augmented images 69

Table 4-8 Combined results for the low-level hardware configuration 70

Table 4-9 Combined results for the intermediate-level hardware configuration 70

Table 4-10 Combined results for the high-level hardware configuration 70

Table 4-11 Combined results for the smallest models on low-level hardware configuration 72

Table 4-12 Combined results for the smallest models on intermediate-level hardware configuration 72

Table 4-13 Combined results for the smallest models on high-level hardware configuration 73

LIST OF FIGURES

Figure 2-1 Original Image before image processing has occurred 9

Figure 2-2 Spatial domain sharpened image 10

Figure 2-3 Frequency domain sharpened image 10

Figure 2-4 Grayscale image with no additional preprocessing 12

Figure 2-5 Image with binarization applied 14

Figure 2-6 Skeletonized Image with most detail removed 15

Figure 2-7 Skeletonized image with additional preprocessing 15

Figure 2-8 Components, subfields and types of AI (adapted from Abioye et al., 2021) 17

Figure 2-9 ANN diagram showing input, hidden and output neurons 19

Figure 2-10 A typical CNN with one convolutional layer and one pooling layer 21

Figure 2-11 An image annotated in CVAT. Both the license plate and the characters on the license plate are annotated. 22

Figure 2-12 Illustration of a scalar vs a vector vs a matrix vs a third-order tensor 23

Figure 2-13 YOLO architecture, with 24 convolutional layers followed by two fully connected layers 25

Figure 2-14 Consistent dual assignments for NMS-free training 27

Figure 2-15 OpenALPR processing steps 29

Figure 2-16 Base Image without any augmentations 38

Figure 2-17 Augmented Image, with added skew to change the angle of the license plate 38

Figure 2-18 Augmented image, skewed with added noise and a random background 39

Figure 2-19 Various Albumentation augmentations on an image of a parrot 39

Figure 3-1 Overview of the experimental pipeline from data preparation to result storage 45

Figure 4-1 Initial Image without any preprocessing 51

Figure 4-2 Extracted license plate image (extracted with YOLOv8s) 51

Figure 4-3 Confusing image where multiple different images have been merged 52

Figure 4-4 Extracted license plate image (extracted with YOLOv8s) from image with multiple license plates 53

Figure 4-5 License plate is visible on image, but could not be extracted 53

Figure 4-6 License plate is visible on image, but could not be extracted 54

Figure 4-7 Clearly visible license plate image 54

Figure 4-8 Calibration results on high-level hardware 55

Figure 4-9 Calibration on low-level hardware 56

Figure 4-10 Calibration on intermediate-level hardware 57

Figure 4-11 Combined calibration results for all three systems 58

Figure 4-12 Non-ML OpenALPR results with preprocessing 59

Figure 4-13 Annotated characters on license plate image 60

Figure 4-14 Training data progression, from 200 to 600 images 61

Figure 4-15 Unreliable overlap suppression 62

Figure 4-16 Potential character overlaps on license plate image 62

Figure 4-17 License plate with extraneous text 63

Figure 4-18 Confused model, incorrect license plate prediction 64

Figure 4-19 Artificially generated dummy license plate 64

Figure 4-20 South African license plate backgrounds for all nine provinces 65

Figure 4-21 Accurate license plate prediction 65

Figure 4-22 Skew augmented artificial dummy license plate 66

Figure 4-23 Standard Image with incorrect predictions 67

Figure 4-24 Dummy Image with random background 68

Figure 4-25 License plate with random background 68

Figure 4-26 Augmented license plate with random background 69

Figure 4-27 Processing time by YOLO version (small models) 71

Figure 4-28 Accuracy comparison of small vs nano/tiny YOLO models 73

Figure 4-29 Processing time comparison of small vs nano/tiny YOLO models 73

LIST OF ABBREVIATIONS

Abbreviation / Term	Definition
AI	Artificial Intelligence
ALPR	Automatic License Plate Recognition
ANN	Artificial Neural Network
ANPR	Automatic Number Plate Recognition
ASTER	Attentional Scene Text Recognizer with Flexible Rectification
BSD	Berkeley Software Distribution (license)
CCA	Connected Component Analysis
CmBN	Cross mini-Batch Normalization
CNN	Convolutional Neural Network
COCO	Common Objects in Context (dataset)
CUDA	Compute Unified Device Architecture
CVAT	Computer Vision Annotation Tool
DCT	Discrete Cosine Transform
DNN	Deep Neural Network
DPM	Deformable Parts Model
DWT	Discrete Wavelet Transform
GELAN	Generalized Efficient Layer Aggregation Network
GLD	Generalized Levenshtein Distance
LCS	Longest Common Subsequence
LPCS	License Plate Character Segmentation
LPR	License Plate Recognition
MORAN	Multi-Object Rectified Attention Network
NMS	Non-Maximum Suppression
OCR	Optical Character Recognition
OpenALPR	Open-source Automatic License Plate Recognition (library)
OpenCV	Open-source Computer Vision (library)
PAN	Path Aggregation Network
PGI	Programmable Gradient Information
PSM	Page Segmentation Method
R-CNN	Region-based Convolutional Neural Network
ReLU	Rectified Linear Unit
SAM	Spatial Attention Module
SAT	Self-Adversarial Training
YOLO	You Only Look Once (object detection framework)

Chapter 1: Introduction

Automatic license plate recognition (ALPR) systems have become increasingly important in the modern world. There are many different software approaches to ALPR, but there is limited guidance on how to systematically compare their performance across different hardware configurations. This dissertation presents a reproducible methodology for performing such comparisons and demonstrates it by evaluating multiple ALPR approaches across three hardware setups, each representing a different level of performance and price.

For example, a company may wish to use their existing cameras to validate that their cargo trucks are in the correct locations, by checking the license plates of said trucks when they stop to unload. These cameras might not be optimally positioned to obtain a clear view of the license plate; may have poor night vision; or could be poorly shielded from inclement weather. Therefore, by providing a methodology to evaluate the performance of various ALPR approaches across different hardware configurations, this study aims to assist organisations in making informed decisions regarding the most suitable and cost-effective ALPR set-ups for their operational needs.

1.1 Background

The above example actually reflects the real-world scenario that inspired this study, as a project was implemented to read the license plates of trucks to determine if they were unloading their cargo in the correct locations. The company involved in the project had recently installed an expensive camera system with good night vision capabilities, but they did not install LPR cameras and did not have the budget to reinstall all their cameras, so a software-based approach using existing cameras was required.

There are many different methods available that can be used to perform ALPR (Agbemenu *et al.*, 2018; Du *et al.*, 2013; Laroca *et al.*, 2018). ALPR can be performed with specialised hardware, in the form of licence plate recognition cameras that have the license plate recognition software built into the cameras themselves (Yang *et al.*, 2022) or it can be performed using images or video feeds from generic cameras (Garain *et al.*, 2008).

Recognising license plates from images or videos requires specific software to extract (Palekar *et al.*, 2017) and interpret the license plate details (Mittal & Garg, 2020). This dissertation focuses on comparing the performance of different approaches to software-based license plate recognition, including machine learning and algorithmic methods. These approaches will be tested on systems with varying degrees of computing power, from a relatively cheap low-level single board computer to expensive high-performance consumer-level hardware.

Extracting a license plate from an image can be accomplished with computer vision software (Szeliski, 2022). OpenCV is a widely used computer vision software library, which is both free and Open Source, allowing anyone to use it without being constrained by licensing concerns (Culjak *et al.*, 2012). This software can be used to determine the location of a license plate in an image. Once located, the license plate can be extracted and processed using optical character recognition to obtain the text thereon.

Deep neural networks (DNNs) are another method that can be used to perform license plate extraction. Neural networks function by first being trained to recognize certain objects in an image, be it an animal, vehicle, person or, for the purposes of this research, a license plate (Goel *et al.*, 2020). Neural networks are trained using a set of images called a training set, where the target is annotated manually, then verified using a smaller subset of the image set called the testing set, before finally being validated using another subset of the image set called the validation set (Chauhan *et al.*, 2018; Li *et al.*, 2021). The number of images needed to train these models is usually very large.

Optical character recognition (OCR) is a method whereby a computer reads text in an image (Patel *et al.*, 2012). Tesseract is a free, open source tool that can be used to perform OCR (Smith, 2013). This is the final step in the process which only occurs once a license plate has been successfully extracted from an image. If another part of the image has been extracted instead of the license plate, for example a line of text on the side of a commercial vehicle, it can cause the character recognition algorithm to provide an incorrect result for the image.

The accuracy and speed of both the extraction and recognition of license plates might be increased by using image preprocessing, which is a set of methods used to modify an image (Jeong *et al.*, 2018). Image preprocessing techniques include grayscaling, binarization, enhancement and many more.

1.2 Problem statement

There are many different approaches to ALPR, ranging from an image processing approach with OCR (Agbemenu *et al.*, 2018), to using machine learning and neural network (Chauhan *et al.*, 2018) frameworks such as TensorFlow (Abadi *et al.*, 2016) or YOLO (You Only Look Once) (Laroca *et al.*, 2018). Each approach has its own strengths and weaknesses, but it is still unclear exactly how well each approach works with different computer hardware setups; a computer with no dedicated graphical processing unit might perform better using certain methods than a computer with a weak dedicated graphical processing unit. Therefore, this study seeks to provide a reproducible methodology for comparing the performance of different ALPR approaches across

different hardware platforms (Goel *et al.*, 2020; Shafiee *et al.*, 2017). Ideally, a simple comparison of public hardware benchmarks to the ones used in this study should give any future researchers valuable insights into the performance of their own hardware.

1.3 Scope of the study

Given the problem statement, this study will have restrictions on the data and software used. This ensures that the experimental procedures employed in this study are consistent and reproducible and simplifies the process of setting up new experiments.

1.3.1 Data

There is a significant difference between processing images that are captured during the daytime and images captured during the nighttime. This study will be restricted to daytime images captured under clear weather conditions and to South African license plates. Images captured at night or during adverse weather conditions, such as rain, will be filtered out to maintain experimental consistency.

This study is further restricted to South African license plates because there can be significant variations between different countries' license plates, which can severely impact the accuracy of the readings (Srinivas *et al.*, 2008), so a dataset with multiple countries' data would need to be tested to see if a viable result can be produced. Although International license plate datasets are more widely available and may be used to calibrate systems, where necessary, the primary evaluation dataset remains South African. Future research could expand the scope of this study to include multiple international plate formats, as well as nighttime and adverse-weather image conditions.

Images will all be resized to 640x640, so that they are, on average, 100KB in size. All images will be converted to jpeg for consistency, since mixing image formats can cause unexpected behaviour.

1.3.2 Software

A single programming language (Python) will be used for all processing. Python is an interpreted, rather than compiled, programming language that is cross-platform and supports all the tools necessary for this study. Future studies can be performed with optimised code that is compiled for each platform to see how performance is impacted. OpenCV will be the primary image processing library used, since it is compatible with Python and free to use. OpenALPR will be the algorithmic library used to perform the ALPR operations, since it is also free to use, even though

it has not been updated since 2020. YOLO will be used to test the machine learning side of ALPR.

These are not the only tools available that are able to perform the functions necessary for the experiments in this study. They are a representation of free tools that anyone can access that will work on most computers; similar alternative software can be used instead. For example, there is a Python library called scikit-image[1] that can also be used for image manipulation, and OpenCV is itself not bound to Python; the software that inspired this paper was written using OpenCV with Java.

1.4 Research paradigms and key concepts

This section introduces the key concepts that form part of research paradigms (Section 1.4.1) and then classifies the paradigm that is used in this study (Section 1.4.2).

1.4.1 Key concepts

A research paradigm is a basic belief system and theoretical framework (Abdul Rehman & Alharthi, 2016). This is the lens that researchers use to view the world and conduct their studies. The key aspects of research paradigms are ontology, epistemology, methodology and methods. Some researchers also include axiology, i.e. ethics, as part of this framework (Pretorius, 2024).

Ontology is the science of what is, of applying a classification of anything in any sphere of being (Smith, 2012). Epistemology, also known as the theory of knowledge, is a field that seeks to know how we know things (Audi, 2010). Methodology refers to the overarching plan and reasoning that permeates the entire research process (Mills, 2014). Methods are the specific techniques and procedures employed to collect and analyse data within the research framework (Mills, 2014).

1.4.2 Classifications

There are many research paradigms, but these can be grouped into four main classifications: (Kivunja & Kuyini, 2017):

Critical Theory guides research from the perspective of power structures and social justice (Devetak, 2013). In this case, the results of this study can provide anyone with even basic computer resources access to powerful tools that they can use without having to spend any money.

[1] https://github.com/scikit-image/scikit-image

Interpretivism guides research by focusing on understanding human behaviour, experiences and actions through qualitative methods (Alharahsheh & Pius, 2020; Potrac *et al.*, 2014). Since this study focuses strictly on acquiring and interpreting data from experiments, a quantitative method is required.

Pragmatism emphasises practical problem solving by selecting methods that yield or have yielded actionable insights (Legg & Hookway, 2008). It can use both qualitative and quantitative methods, however, while it could be suitable for use in this study, it is more difficult to use for a straightforward experimental setup than positivism.

Positivism guides research by adopting a systematic, objective, and quantifiable approach to investigating phenomena, relying heavily on empirical evidence and experimentation (Park *et al.*, 2020). Positivism is used as the basis for the study, since it is best used when you wish to objectively observe the relationship between different hardware and software setups when extracting and reading license plates (Park *et al.*, 2020). This study is a quantitative study where we accurately measure the time it takes to perform license plate recognition on each system, down to the closest millisecond, and measure the accuracy of each prediction given to us by the system. The relationship between the computer hardware and each software implementation is key to the study.

1.5 Research questions

This study has both a primary and secondary research questions, where the secondary questions are used to help answer the primary research question.

1.5.1 Primary research question

The research question that is addressed in this dissertation is: How can one compare the performance of software for license plate recognition on different hardware platforms?

1.5.2 Secondary research questions

To support the primary research question, we can ask the following secondary questions:

1. What metrics can be used to compare the accuracy and speed of license plate recognition implementations?
2. How do different software-based approaches to license plate recognition compare in recognition accuracy?
3. How do different hardware configurations effect the processing speed of license plate recognition software?

1.6 Objectives of the study

The objectives of the study will seek to address the research questions and are also divided into a primary objective (to answer the primary research question) and secondary objectives to answer the secondary research questions.

1.6.1 Primary objective

The primary objective of the study is to provide a reproducible methodology that can be used to compare the performance of software-based license plate recognition across different hardware platforms.

1.6.2 Secondary objectives

The secondary objectives will address the secondary research questions as follows:

1. To identify and apply metrics that can effectively compare the accuracy and speed of ALPR implementations.
2. To evaluate different software-based approaches to ALPR and compare their recognition accuracy.
3. To measure and compare the processing speed of ALPR software across different hardware configurations.

1.7 Ethics

This study does not involve humans or interaction with humans, but there are several important ethical principles that are applied at every step of the research. This study adheres to the ethics policy of the North-West University (2018), and to its principles; the key principles of this policy being:

- Beneficence and non-maleficence: maximize the benefit and minimize the harm,
- Distributive justice: a fair balance of risk and benefits among all stakeholders,
- Respect: respecting the choices and autonomy of all research participants.

The study has complied with all ethical considerations and seeks to create benefits without causing any harm. An application has been made to, and approved by, the Faculty of Natural and Agricultural Science Research Ethics Committee. The risk category is minimal, and the ethics number assigned to the study is NWU-01276-24-A9. The ethics certificate itself is also attached to this dissertation.

As this study employed only free and publicly available data[2], there was no need to obtain permission for its use.

Even though the data is publicly available, this study uses artificially generated license plate images to showcase the various preprocessing and processing steps wherever possible. Simulating license plates in this way reduces the number of real license plate images that must be shown in the study, thus significantly reducing the potential for breaching someone's privacy.

1.8 Dissertation overview

This dissertation is structured as follows: **Chapter 2** provides a review of the present literature as it relates to the study and identifies the metrics that can be used to compare ALPR implementations, addressing the first secondary research question. **Chapter 3** details the methodology used in the study, including the experimental setup and procedure, providing the framework for answering the second and third secondary research questions. **Chapter 4** presents the results of applying this methodology across the three hardware configurations. **Chapter 5** closes with the analysis of the data gathered and the conclusions reached.

[2] https://universe.roboflow.com/haeun-kim-ri91b/license-plate-detection-wienp

Chapter 2: Literature review

This literature review is organised into several key areas that underpin the development of ALPR systems. Each area directly supports the evaluation of various ALPR methods across different hardware configurations, which is central to the research question. It begins by examining preprocessing techniques aimed at enhancing image quality and system robustness. The review then explores both machine learning and non-machine learning methodologies for ALPR. Subsequent sections delve into character segmentation and OCR, essential for accurate license plate extraction, and further analyse the roles of edit distance calculations to measure the accuracy of the ALPR implementations and image augmentation to increase the size of the training and validation data sets.

This study will investigate several concepts and methods, including preprocessing (Section 2.1), artificial intelligence (Section 2.2), machine learning types and frameworks (Section 2.3), ALPR (Section 2.4), character segmentation (Section 2.5), optical character recognition (Section 2.6), edit distance methodologies (Section 2.7) and image augmentation (Section 2.8).

2.1 Preprocessing

Preprocessing is an important part of OCR as it helps the system make more accurate predictions (Mittal & Garg, 2020) by helping to increase the image quality, especially for the components that are needed, such as the character bodies and the contrast between the characters and the background (Zhang *et al.*, 2011). The number and order of preprocessing steps depends on the type of image being presented (Srinivas *et al.*, 2008). Image preprocessing extracts valuable information from an image (Chaki & Dey, 2018). It can be used to increase the quality of the image while also reducing the amount of unwanted information within an image (Chaudhuri *et al.*, 2017).

2.1.1 Enhancement and sharpening

The first preprocessing techniques we will examine are enhancement and sharpening. These methods may be used to efficiently improve the quality of images, allowing for enhanced accuracy of the license plate extractor and OCR reader.

Image enhancement is a process where the pixel intensity of the image is changed by mathematically adjusting the value that represents brightness in the image, so that the image looks subjectively better, while also providing an improved input image for image processing steps (Ibrahim & Pik Kong, 2009; Maini & Aggarwal, 2010). Image enhancement modifies the properties and attributes of an image, such as the brightness, contrast or sharpness of the image, to make

it more suitable for a given task (Maini & Aggarwal, 2010), for example making it easier to recognise the characters on an image with OCR (Ibrahim & Pik Kong, 2009). Image sharpening is a process where the image edge information is enhanced and is a subset of image enhancement (Zhang *et al.*, 2011).

Image enhancement methods can be divided into Spatial and Frequency Domain methods (Guo *et al.*, 2022). In the case of Spatial Domain methods, the image's pixels (i.e. the picture's elements) are dealt with directly, as shown in Figure 2-1 and Figure 2-2 (Rahman *et al.*, 2014). However, in the case of Frequency Domain methods a mathematical function, such as a Fourier transform or a Discrete wavelet transform (DWT) or a Discrete cosine transform (DCT), is first applied to the image and then the frequency domain is used to perform operations. This is followed by another mathematical function, such as an Inverse Fourier transform, being employed to restore the resultant image, as shown in Figure 2-1 and Figure 2-3 (Aghagolzadeh & Ersoy, 1992). Spatial Domain methods are relatively simple to perform and have a lower complexity compared to Frequency Domain methods (Maini & Aggarwal, 2010; Singh & Mittal, 2014). Images for license plate recognition can be enhanced by removing unneeded lines on the image with an algorithm, such as the unwanted-line elimination algorithm (Al-Ghaili *et al.*, 2013).



Figure 2-1 Original Image before image processing has occurred



Figure 2-2 Spatial domain sharpened image



Figure 2-3 Frequency domain sharpened image

2.1.2 Grayscaling

Image recognition systems, including Machine Learning applications such as R-CNNs (Gilbert & Rusli, 2020), commonly use grayscale, which is a technique that removes colour from an image as an initial step for further preprocessing (Kanan & Cottrell, 2012). Grayscale images are often employed because their use simplifies the algorithms that need to be used and reduces the overall computation requirements (Kanan & Cottrell, 2012; Zhang *et al.*, 2011), since many image processing steps can be completed with only image intensity (Papamarkou & Papamarkos, 2013). Colour images might also increase the amount of training data required to train a machine learning model or program model (Kanan & Cottrell, 2012).

Converting an image from colour to grayscale is a lossy procedure, where the representation changes from 3 channels or dimensions to 1 channel or dimension (Macêdo *et al.*, 2015; Papamarkou & Papamarkos, 2013). Pixels in colour images can be represented with an RGB (Red, Green and Blue) value (the three pure channels (Macêdo *et al.*, 2015)), which can be further represented in three dimensions, XYZ, and illustrated with the attributes lightness, chroma and hue, where the lightness is the brightness of an area judged relative to the brightness of a similarly illuminated area (CIE, 2022a); chroma is the colourfulness of an area compared to a similarly illuminated area that appears grey, white or highly transmitting (CIE, 2022b); and hue describes a pure colour (Shi *et al.*, 2009), such as red, green, blue or yellow (CIE, 2022c).

Each component has 256 possible values, if each RGB value occupies 8 bits, with a total of 256 * 256 * 256 = 16 777 216 possible values (Zhang *et al.*, 2011). A grayscale image consists only of the intensity or luminosity value (Kekre & Thepade, 2008), with 256 possible values (Zhang *et al.*, 2011). The Y channel (luminance component) is usually used as the grayscale image (Papamarkou & Papamarkos, 2013).

There are several methods of performing image grayscale conversion (Kanan & Cottrell, 2012). The most common method uses the weighted means of the RGB image channels. Common terms for these means are the Intensity and Luminosity, with issues often occurring when there is a disparity in the luminance and the chromaticity (Kanan & Cottrell, 2012). Other methods include Luma and Lightness, which generate a more perceptually accurate image in terms of brightness (Kanan & Cottrell, 2012; Macêdo *et al.*, 2015). Figure 2-1 shows the original image before any processing occurred, while Figure 2-4 shows the image after it has been grayscaled.

Common software such as Adobe Photoshop require users to set the parameters for the grayscale operation with trial and error (Saravanan, 2010).



Figure 2-4 Grayscale image with no additional preprocessing

2.1.3 Binarization

The third method we will examine is image binarization, which makes use of grayscaled images.

Binarization is a technique where images in grayscale format are converted into a binary format (Srinivas *et al.*, 2008), where the foreground information is formed from black pixels and the background information is formed from white pixels, or vice versa (Singh, Sinam, *et al.*, 2012; Singh *et al.*, 2011; Stathis *et al.*, 2008). The product of this is called a binary image or a monotonic image (Wu *et al.*, 2015). One of the simpler methods of binarization is called thresholding (Nina *et al.*, 2011; Singh *et al.*, 2011).

The goal of binarization is to reduce the amount of information present in the image (Chaudhuri *et al.*, 2017) and to separate objects from the background (Eyupoglu, 2017). It is often necessary for the image to be first converted into grayscale for binarization to take place (Puneet & Garg, 2013), but some binarization methods can work with colour images (Saini, 2015).

Binarization can be applied to the whole image (Srinivas *et al.*, 2008) in a process called global binarization (Stathis *et al.*, 2008), or it can be applied to only parts of the image in a process called local binarization (Singh *et al.*, 2011), or both can be applied simultaneously in a hybrid approach (Stathis *et al.*, 2008).

Global binarization is usually much faster to use (Singh *et al.*, 2011) but can produce noise along the image borders (Singh, Sinam, *et al.*, 2012). One example of global binarization is the Otsu method (Huang *et al.*, 2012). It requires a single threshold value that is applied to the whole image (Nina *et al.*, 2011). Otsu's method of binarization assumes there are only two components of an image (Yousefi, 2015), the background and the foreground (Oliver et al., 2011). This method is similar to that employed by many other binarization techniques (Singh, Sinam, *et al.*, 2012). It does not work well in images where there are shadows or uneven illumination, however, (Yoon, 2013) and can produce noise in the image along the borders (Singh *et al.*, 2011). Other global binarization techniques include the Fixed Thresholding Method (Puneet & Garg, 2013), the Kapur method (Fung & Chamchong, 2010) and the Kittler and Illingworth Method (Fung & Chamchong, 2010; Milyaev *et al.*, 2013; Puneet & Garg, 2013).

Local binarization techniques, which are sometimes called locally adaptive binarization techniques (Eyupoglu, 2017), evaluate the neighbouring pixels' data for each pixel or each region to determine its threshold and is then classified into an appropriate category (Fung & Chamchong, 2010; Singh, Sinam, *et al.*, 2012). Local binarization techniques are frequently sensitive to noise in images (Oliver et al., 2011) and have much larger computational requirements than global binarization since each individual pixel has to be computed (Singh *et al.*, 2011).

There are methods of increasing the speed of local binarization by, for example, using the integral sum image technique to find the mean of the neighbouring pixels before performing image processing (Singh *et al.*, 2011). Other examples include the Niblack Method, where a rectangular window is shifted across the image to calculate the threshold using the mean and the variance of the intensity values within the rectangle; the Sauvola Method (inspired by the Niblack Method), which uses the standard deviation value of the pixels within the window; and the Bernsen Method, which uses contrast within the window to set the threshold (Eyupoglu, 2017; Puneet & Garg, 2013; Saini, 2015; Singh *et al.*, 2011).

It is possible to use multiple images that have different binarization methods applied to them (Yoon, 2013). Hybrid approaches, such as the Improved IGT method, can apply global binarization to whole images and then use local binarization methods only on some parts of said images (Stathis *et al.*, 2008). Examples of hybrid approaches include O'Gorman and Liu and Li (Singh *et al.*, 2011).

Binarization is often the first preprocessing method OCR engines use (Srinivas *et al.*, 2008), often for applications in license plate recognition (Du *et al.*, 2013) or document image analysis (Fung & Chamchong, 2010). Figure 2-5 shows an example of an image where binarization has been applied.

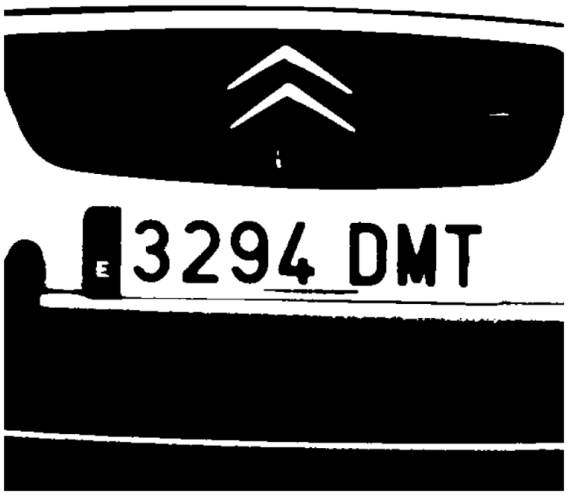

Figure 2-5 Image with binarization applied

Taken together, these methods involve a trade-off between speed and robustness, rather than one method being best in all cases. Global methods such as Otsu's are fast and simple, as a single threshold is applied to the whole image (Nina et al., 2011). For that same reason, however, they tend to fail when the lighting is uneven or shadows are present (Yoon, 2013), which is common when license plates are captured outdoors. Local methods such as Niblack, Sauvola and Bernsen adjust the threshold for each region and therefore cope better with uneven lighting, but they are slower, as every pixel must be evaluated (Singh et al., 2011), and are more sensitive to noise (Oliver et al., 2011). Hybrid methods try to combine the two by applying global thresholding to the whole image and using local thresholding only on the more difficult regions (Stathis et al., 2008). For license plate recognition, where images are usually captured outdoors under changing lighting, this trade-off tends to favour local or hybrid binarization over a single global threshold.

2.1.4 Skeletonization

The final method we will examine is image skeletonization/thinning.

Image thinning, also called image skeletonization (Saeed *et al.*, 2010), is a process where a binary image is reduced to lines (Kaur & Sharma, 2013) that are only one pixel in width (Abu-Ain *et al.*, 2013). It can be done in two ways: iterative and non-iterative. When skeletonization is done iteratively, it can be done either sequentially, where the unwanted pixels are removed after identifying the desired pixels in each iteration; or in parallel, where the pixels are all erased after identifying the entire image's unwanted pixels.

For non-iterative skeletonization, the skeleton is extracted without examining each pixel individually (Abu-Ain *et al.*, 2013). Figure 2-6 shows an extreme example of image skeletonization, where most of the image details were removed, whereas Figure 2-7 shows skeletonization after preprocessing has been completed.

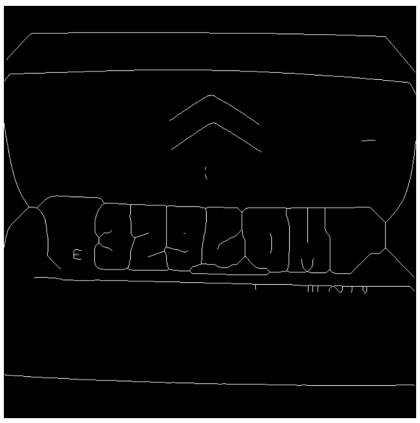

Figure 2-6 Skeletonized Image with most detail removed

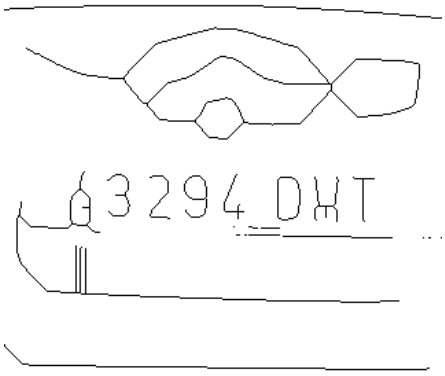

Figure 2-7 Skeletonized image with additional preprocessing

2.1.5 OpenCV

OpenCV (Open-Source Computer Vision) is a software library used for image and video analysis and manipulation, and is the library used to perform preprocessing for this study (Culjak *et al.*, 2012). The library began as a research project at Intel in 1998, and has been available since 2000 on the BSD license (Pulli *et al.*, 2012).

The Computer Vision part of OpenCV refers to the science of programming a computer to process and understand images and video, including performing preprocessing on images (Culjak *et al.*, 2012). It contains many modules that deal with image or video-related tasks, a mix of low-level image processing functions, and high-level functions, such as facial detection (Pulli *et al.*, 2012),

with the main modules found at [3]. Some examples of OpenCV modules that might be used for this study can be found in Table 2-1.

Table 2-1 Key OpenCV modules used in the study

Module Name	Purpose
Calib3d	Camera calibration and 3D reconstruction
Core	Core OpenCV functions, always used
DNN	Deep neural networking module
Imgcodecs	Image file reading/writing
Imgproc	Image processing
ML	Machine Learning
Objdetect	Object detection
Stitching	Image stitching

The OpenCV authors have also written another library, called OpenCV_contrib[4], that adds extra modules to OpenCV. These modules include support for additional operations, such as image skeletonization, image quality analysis, MatLab interfacing, video stabilization and many more. OpenCV can be installed and used without adding the OpenCV_contrib modules.

2.2 Artificial Intelligence

Alan Mathison Turing, a British computer scientist, mathematician and military officer, is considered the father of computer science and artificial intelligence (Grzybowski *et al.*, 2024). The term Artificial intelligence (AI) was first introduced during the Dartmouth Conference in 1956 (Krauss, 2024), and is a broad term with many different types and subfields, such as machine learning and computer vision, among others. AI itself has undergone periods of low enthusiasm and support, typically referred to as AI winters, with the two most prominent being between 1974-1980 and 1987-1993 (Sarikaya, 2024; Toosi *et al.*, 2021).

Figure 2-8 shows a breakdown of the types and subfields of AI, as well as the different components that make it up (Abioye et al., 2021). AI can be divided between narrow AI, which is limited in scope but excels in the specific task it was created to perform, and general AI which seeks to replicate human intelligence and perform any task that a human can (Babu & Banana, 2024).

[3] https://docs.opencv.org/4.x/
[4] https://github.com/opencv/opencv_contrib

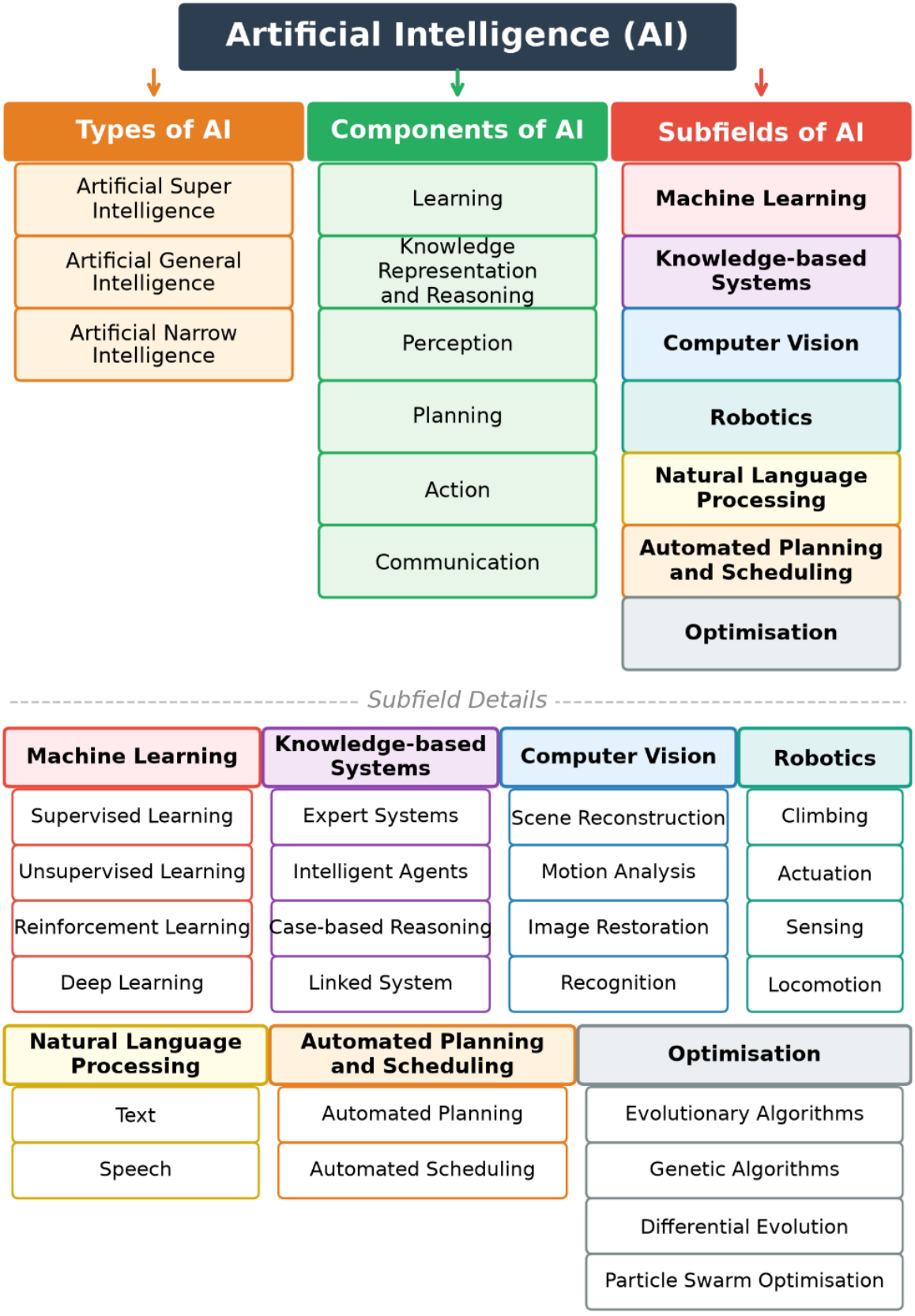

Figure 2-8 Components, subfields and types of AI (adapted from Abioye et al., 2021)

This study uses narrow AI to perform ALPR, as discussed in the next section.

2.3 Machine learning

Machine learning is a subfield of AI and a discipline that seeks to build machines that learn from experience (Jordan & Mitchell, 2015), or from past data that can be used to model, control or predict without being explicitly programmed (Abioye *et al.*, 2021; Hu *et al.*, 2021). Machine learning systems are commonly used for object detection, speech transcription, advertisement targeting, and searching, among other applications (Goldsborough, 2016). Machine learning has subfields, such as Deep Learning, Supervised Learning, Unsupervised Learning, and Reinforcement Learning (Abioye *et al.*, 2021).

Supervised learning concerns machines making decisions using labelled data sets, with the most common examples being classification and regression problems. Unsupervised learning concerns machines learning without labelled datasets to learn the relevant patterns or underlying structure of the data. The most common examples of unsupervised learning are dimension reduction, clustering and anomaly detection. Reinforcement learning is concerned with the computing environment where the learning is focused on the outcome of interactions within the environment. Reinforcement learning has been used in applications such as personalised medicine and in Google's AlphaGo AI (Abioye *et al.*, 2021; Hu *et al.*, 2021; Li, 2018).

Deep learning is a form of representation learning with multiple levels of representation, where a machine is fed raw data and is expected to automatically discover the representations needed for classification and detection. It employs complex neural network architectures with many hidden layers composed of non-linear input data transformations. It can be expressed with mathematical notation, but for real-world usage it must be transformed into a computer program. Software such as the Torch framework and Caffe library can be used to develop and train deep learning neural networks (Goldsborough, 2016).

2.3.1 Artificial neural networks

The first neural networks were presented as biological models and as components for circuits. They were introduced in the form of simple neurons by McCulloch and Pitts in 1943 (Anderson, 1995). Artificial neural networks (ANNs) are a form of machine learning (Abiodun *et al.*, 2018).

ANNs consist of neurons, which are in effect processing units (Wu & Feng, 2018). These neurons are divided into three categories: input, output and hidden. The input neurons accept data from the outside world, the output neurons realise the processing result, and the hidden neurons are located between the input and output and cannot be observed from outside the system (Wu & Feng, 2018). Figure 2-9 shows the typical layers found in an ANN, with the weights represented as lines between the neurons, according to (Wang, 2003).

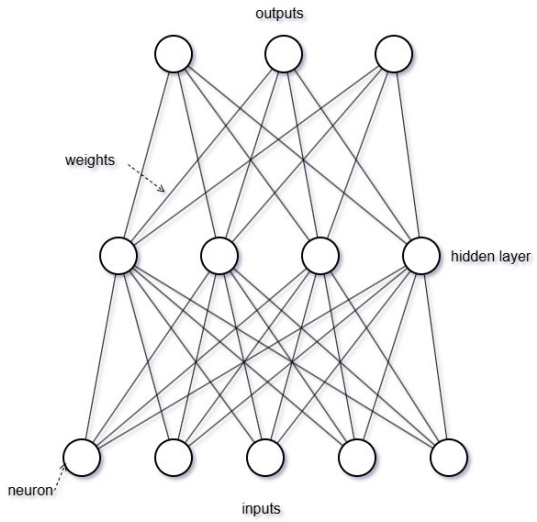

Figure 2-9 ANN diagram showing input, hidden and output neurons

Mathematically, an ANN neuron can be described with the following equation (Krenker *et al.*, 2011):

$$y(k) = F\left(\sum_{i=0}^{m} w_i(k) \cdot x_i(k) + b\right) \tag{2.1}$$

Where:

- $x_i(k)$ is the input value in discrete time *k* where i goes from 0 to *m,*
- $w_i(k)$ is the weight value in discrete time *k* where i goes from 0 to *m,*
- *b* is bias,
- *F* is a transfer function,
- $y(k)$ is the output value in discrete time *k.*

ANNs are used for clustering, image recognition, natural language processing, pattern recognition and prediction across many different disciplines (Abiodun *et al.*, 2018), including license plate recognition (Erdinc Kocer & Kursat Cevik, 2011). ANNs can be implemented in the form of DNNs, with convolutional neural networks (CNNs) (Li, Yang, *et al.*, 2020) being effective at recognising objects in images (Jeong *et al.*, 2018).

CNNs also have several sub-categories, including Region-based Convolution neural networks (R-CNNs) (Gilbert & Rusli, 2020), which are used by frameworks such as YOLO (Shafiee *et al.*, 2017). R-CNNs have also evolved, from R-CNN to Fast R-CNN to Faster R-CNN, which is better for object detection (Gilbert & Rusli, 2020).

2.3.2 Convolutional neural networks

Convolutional neural networks (CNNs) are similar to ANNs, in the sense that they are comprised of neurons that self-optimise through learning, with the only notable difference between CNNs and ANNs being that CNNs are primarily used in the field of pattern recognition in images (O'Shea & Nash, 2015). CNNs consist of three types of layers: convolutional layers, pooling layers, and fully-connected layers (Yamashita *et al.*, 2018).

The convolutional layer is a set of feature maps with neurons attached to it where most of the computation is involved (Aloysius & Geetha, 2017). The rectified linear unit (ReLu) is an activation function commonly used in the convolutional layer that introduces non-linearity by outputting the input directly if positive, allowing faster training compared to sigmoids (O'Shea & Nash, 2015).

The pooling layer performs down-sampling on feature maps (i.e. 2D matrices or tensors) from previous layers and produces new feature maps with condensed resolutions, thereby reducing the spatial dimension of the input to reduce the number of parameters or weights, which reduces the computing cost (Gholamalinezhad & Khosravi, 2020). This also controls the overfitting of the network, where the model becomes too specific to training data instead of being generalized (Gholamalinezhad & Khosravi, 2020; Zafar *et al.*, 2022). Some popular pooling methods are max pooling, that optimises the spatial size (e.g. width or height) of the feature map, defined as

$$f_{max}(X) = \max x_i x_j \tag{2.2}$$

Where *J*-related filters are used for the composition of the *m*-th max pooling band

$$pm = [p1, m, \ldots., pj, m, \ldots., pJ, m] \in R^J \tag{2.3}$$

Other popular pooling methods include average pooling, mixed pooling, stochastic pooling, L_p pooling, spatial pyramid pooling, and region of interest pooling (Gholamalinezhad & Khosravi, 2020).

The fully-connected layer has neurons that are connected to all neurons in the previous layer; the high-level reasoning is done in this layer (Aloysius & Geetha, 2017). This layer is utilised as the CNN's classifier, with the output being the final CNN output (Alzubaidi *et al.*, 2021).

Figure 2-10 shows a typical CNN with only one convolutional layer and one pooling layer, and multiple fully connected layers.

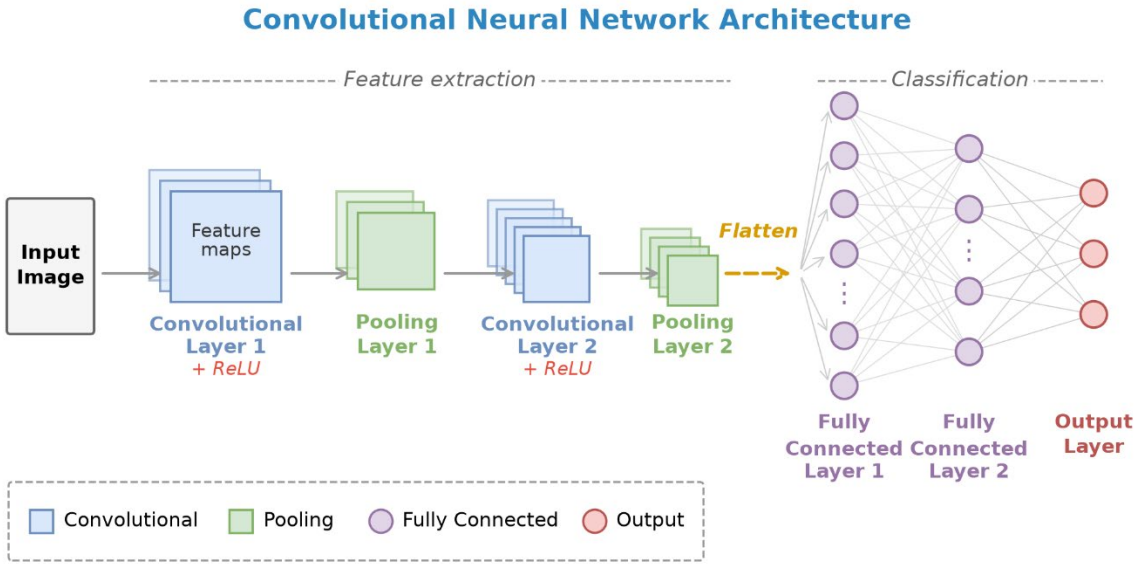

Figure 2-10 A typical CNN with one convolutional layer and one pooling layer

2.3.3 Annotation

Image annotation is essentially the process of adding metadata to an image to simplify access to it, thus it can allow for more effective searching of images (Hanbury, 2008).

Training data can be prepared by labelling datasets containing images with tools such as CVAT[5] (Hossain *et al.*, 2021). This tool is free to use and can be used to label both the license plate and the characters on the license plate, as shown in Figure 2-11.

It is also possible to perform automatic image annotation using tools such as single labelling annotation using binary classification, annotation using ANN, and annotation using a decision tree (Zhang *et al.*, 2012).

[5] https://www.cvat.ai/

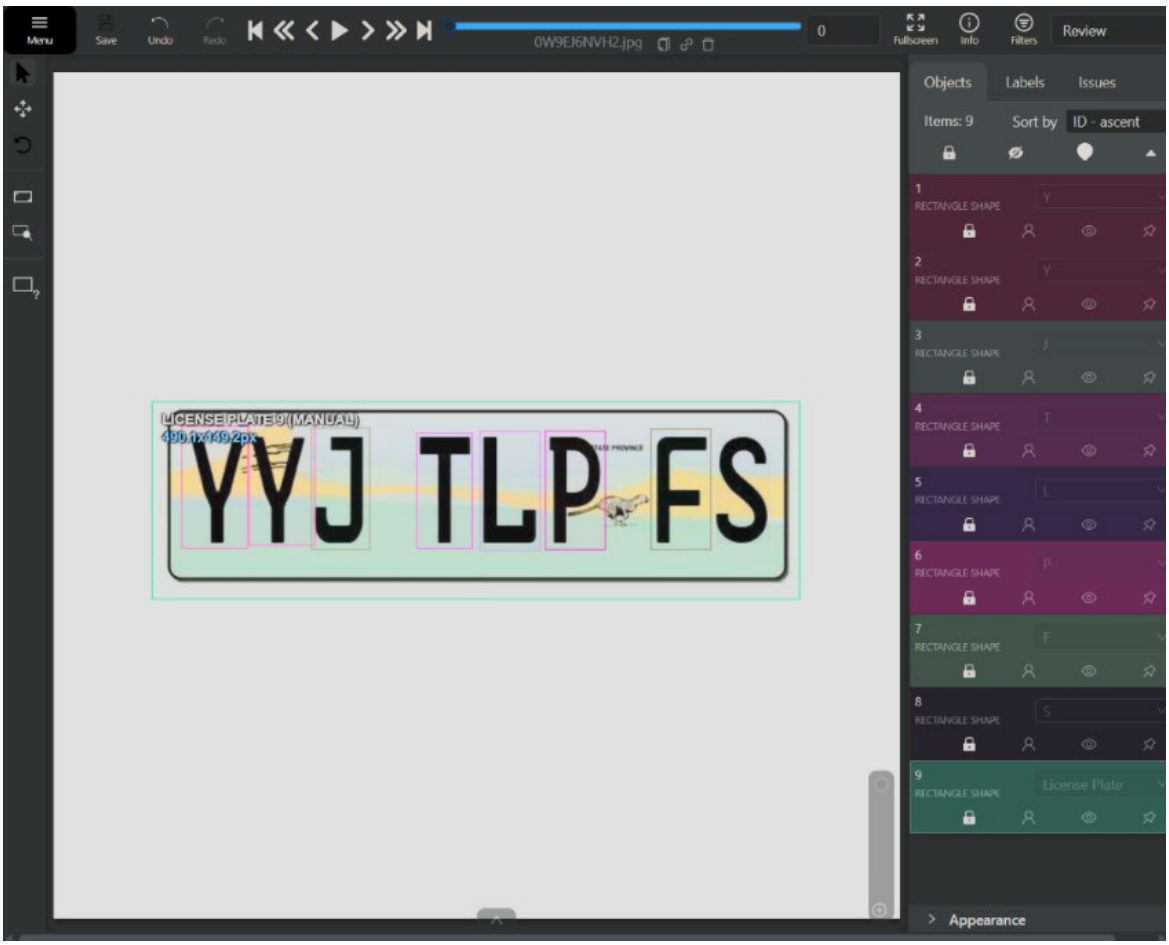

Figure 2-11 An image annotated in CVAT. Both the license plate and the characters on the license plate are annotated.

2.3.4 Neural network frameworks

Building neural networks from scratch is extremely time-consuming, so neural network frameworks that can contain pretrained models and have existing integrations into python are used.

2.3.4.1 TensorFlow

TensorFlow is an open-source deep learning software library that is used for defining, training and deploying machine learning models (Goldsborough, 2016). The core algorithms used by TensorFlow are written in C++ and CUDA (Compute Unified Device Architecture). It was originally created by Google (Pang *et al.*, 2020).

Machine learning algorithms in TensorFlow are represented as computational graphs or dataflow graphs (Pang *et al.*, 2020), which are a form of directed graphs where operations are described

by vertices and nodes and data flowing between them are described as edges (Abadi *et al.*, 2016; Goldsborough, 2016).

The TensorFlow model can be trained to better comprehend the objects that will be fed into it to be recognised (Meng *et al.*, 2017), but it should be done carefully to ensure that the training does not cause over-fitting (Peker, 2019), which is caused by the presence of noise, the limited size of the training data set, and the complexity of classifiers (Ying, 2019). In mathematics, a tensor is a higher order matrix, e.g.

$$x \in R^{H\,x\,W\,x\,D} \tag{2.4}$$

is a third order tensor where H is the height of the image, W is the width of the image, and D is the depth of an image, also commonly referred to as the number of channels. A colour image is an example of a third order tensor that we use daily (Wu, 2017). Figure 2-12 graphically illustrates the difference between the types of tensors, where *d* is the number of dimensions.

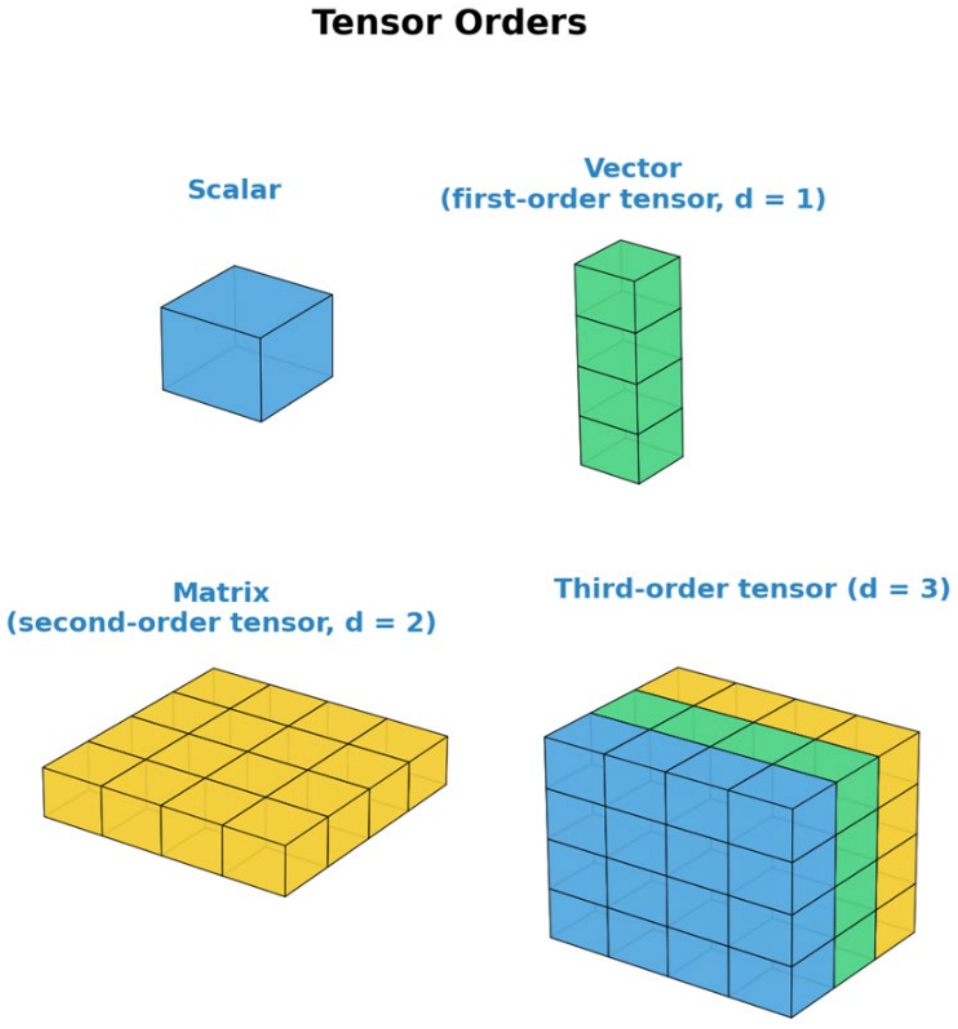

Figure 2-12 Illustration of a scalar vs a vector vs a matrix vs a third-order tensor

2.3.4.2 YOLO

YOLO (You Only Look Once) is an object detection framework that uses neural networks to quickly detect objects in an image (Jeong *et al.*, 2018; Jiang *et al.*, 2022; Shafiee *et al.*, 2017). It was first introduced in a research paper in 2016 (Redmon *et al.*, 2016). When this paper was first conceived, there were five different YOLO versions released: YOLO, YOLOv2, YOLOv3, YOLOv4 and YOLOv5, with some limited versions for specific applications such as YOLO Lite (Jiang *et al.*, 2022) or Fast YOLO (Shafiee *et al.*, 2017). YOLOv1 was introduced in (Redmon et al., 2016), YOLOv2 in (Redmon & Farhadi, 2016), YOLOv3 in (Redmon & Farhadi, 2018), YOLOv4 in (Bochkovskiy *et al.*, 2020). YOLOv5 was not introduced in a paper but was released on GitHub as open-source code[6].

YOLO recognizes objects using R-CNN (region-based convolutional neural network) in a bounded box on the image, called an anchor, where the input image is divided into an S x S sized grid (Redmon et al., 2016). The original YOLO has 24 convolutional network layers which are followed by two fully connected network layers, and is based on the GoogLeNet network (Jiang *et al.*, 2022; Redmon *et al.*, 2016). The structure of the original YOLO is simple to understand, the model it uses is small and fast (Jiang et al., 2022), and it only makes use of a single neural network (Redmon et al., 2016). YOLO reduces the input image's size to 448x448, then feeds it into the model, and then applies thresholding based on the confidence of the results (Redmon et al., 2016). YOLO also makes use of the whole image, instead of using a sliding window that many other object detection methods such as R-CNN or DPM (Deformable Parts Model) use (Shafiee *et al.*, 2017), which greatly reduces the rate in which background objects are identified as relevant objects, being less than half, compared to Fast R-CNN (Redmon et al., 2016). It produces a result in the form of a 7 x 7 x 30 tensor of predictions (Redmon et al., 2016).

[6] https://github.com/ultralytics/yolov5

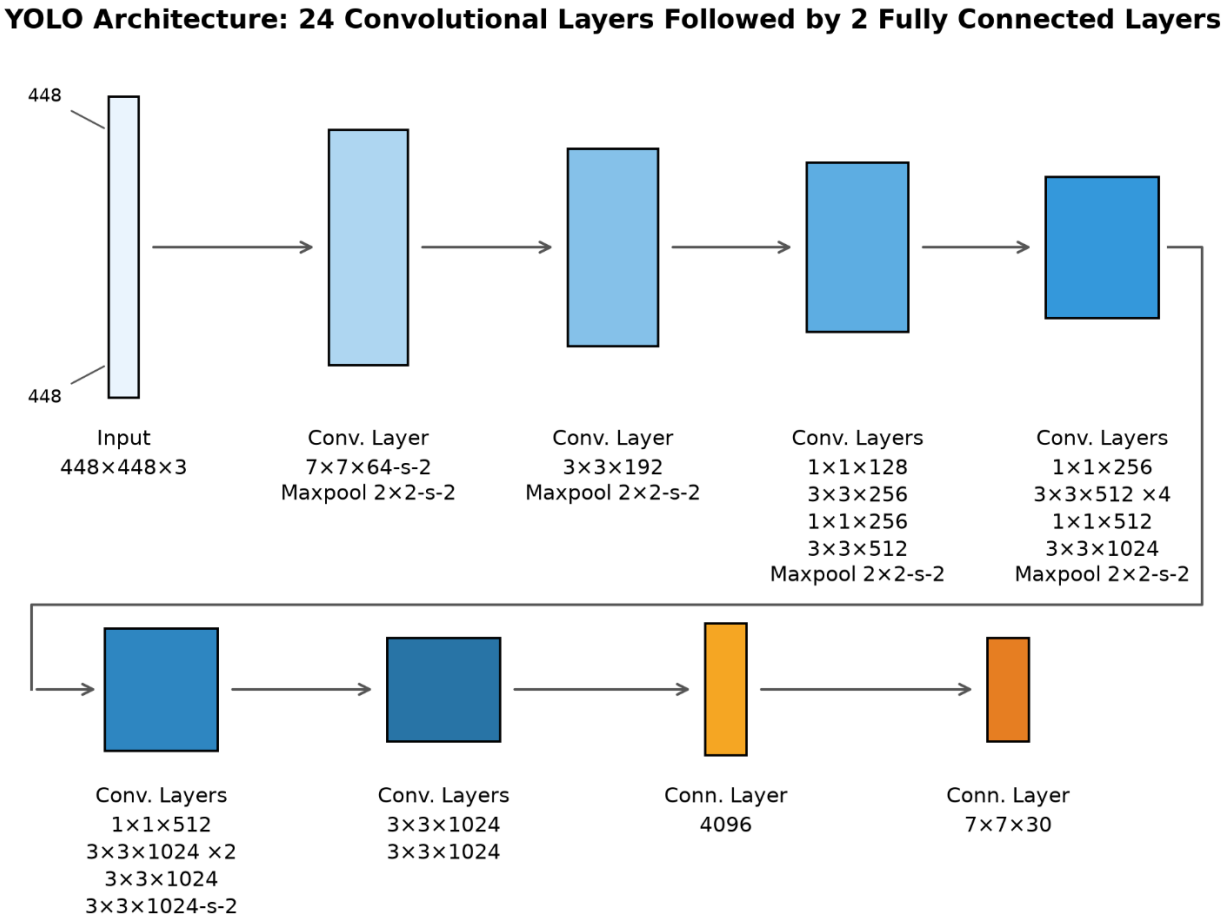

Figure 2-13 YOLO architecture, with 24 convolutional layers followed by two fully connected layers

YOLOv2, which has a system called YOLO9000 in the paper that introduced it, because it can detect over 9000 object categories (Redmon & Farhadi, 2016), uses a model with 19 convolutional layers and five maximum pooling layers and is based on the Darknet-19 model (Jiang *et al.*, 2022; Laroca *et al.*, 2018; Redmon & Farhadi, 2016). It simplifies the YOLO network, rather than expanding on it, with the goal of increasing the accuracy and the speed of the framework (Jiang et al., 2022).

Fast YOLO is based on YOLOv2 and seeks to accelerate the framework to be able to perform real-time object detection on embedded devices (Shafiee *et al.*, 2017). It also introduces a joint training algorithm to train object detectors on both detection data (data meant to find objects) and classification data (data meant to label objects in an image) (Redmon & Farhadi, 2016). Fast YOLO was developed to perform ALPR (Laroca *et al.*, 2018). YOLO Lite is also based on the

YOLOv2 framework, but it was developed instead for computers without GPUs (Graphics Processing Units) (Huang *et al.*, 2018).

YOLOv3 does not include any major design changes compared to YOLOv2; instead, it includes many incremental changes to the overall design of the framework (Redmon & Farhadi, 2018). YOLOv3 uses a model that contains 53 convolutional layers, called the Darknet-53 model, named after the Darknet-19 model (Laroca *et al.*, 2018). It uses the k-means clustering algorithm, which is an iterative, numerical, non-deterministic and unsupervised method of classifying raw data and searching for patterns hidden inside the raw data, to predict the initial width and height of the prediction bounding boxes (Zhao & Li, 2020). It also includes a new classifier and is considered to be an incremental improvement over YOLOv2 (Redmon & Farhadi, 2018). YOLOv3 also has subtypes, such as mini-YOLOv3, which was created to be a smaller version of YOLOv3, only having a feature extraction network backbone 16 percent the size of Darknet-53 (Mao *et al.*, 2019).

YOLOv4 is more focused on comparing data than the previous YOLO versions (Laroca *et al.*, 2018). YOLOv4 has been optimized to work better on a single GPU, with features such as selecting optimal hyper-parameters while applying genetic training, modifying SAM, PAN and Cross mini-Batch Normalization (CmBN). A new data augmentation method, Mosaic (a method of mixing four training images) is used, in addition to introducing a feature called Self-Adversarial Training (SAT), where the model changes the image to make it appear as if there is no desired object in the image, which is followed by the model being trained to find this obscured object (Bochkovskiy *et al.*, 2020).

YOLOv5[7] describes a family of object detection architectures and models based on the COCO dataset (Zhao & Li, 2020). It was released on 18 May 2020 by Glenn Jocher, and it is also an open-source framework, specifically made to be used with the Python 3.8 programming language and pip using the PyTorch framework. YOLOv5 uses the COCO128 dataset, which consists of the first 128 images in the COCO train2017 dataset, to both train and validate the training pipeline.

YOLOv6 was developed by a Chinese technology company called Meituan in 2022 (Wang, Li, *et al.*, 2023). It introduced anchor-aided training, which combines anchor-based and anchor-free paradigms during the training of the model, while remaining free of anchors at inference (Li *et al.*, 2023).

[7] https://github.com/ultralytics/yolov5

YOLOv7 focused on the training process, rather than the architecture, by introducing a “trainable bag-of-freebies” (Wang, Bochkovskiy, *et al.*, 2023). It introduces model re-parameterisation (which means the model has a different structure during training than inference), which allows it to reduce the number of parameters by about 40% (Wang, Bochkovskiy, *et al.*, 2023).

YOLOv8 was released by Ultralytics on January 10, 2023[8]. It uses an anchor-free split head (output layer), which means that one branch of the network predicts bounding box location and another predicts the class, rather than having both operations being performed by the same layer.

YOLOv9[9] was released in 2024 and introduced features such as Programmable Gradient Information (PGI), which helps to combat information loss in deep networks, and Generalized Efficient Layer Aggregation Network (GELAN), which helps to reduce parameters required (Wang *et al.*, 2025). It was built on the Ultralytics YOLOv5 codebase.

YOLOv10 was also released in 2024, built by Tsinghua University researchers, and is the first YOLO model to eliminate Non-Maximum Suppression (NMS) at inference by using dual assignments consistently (Wang *et al.*, 2024). Figure 2-14, obtained from the ultralytics website, demonstrates how this architecture is used.

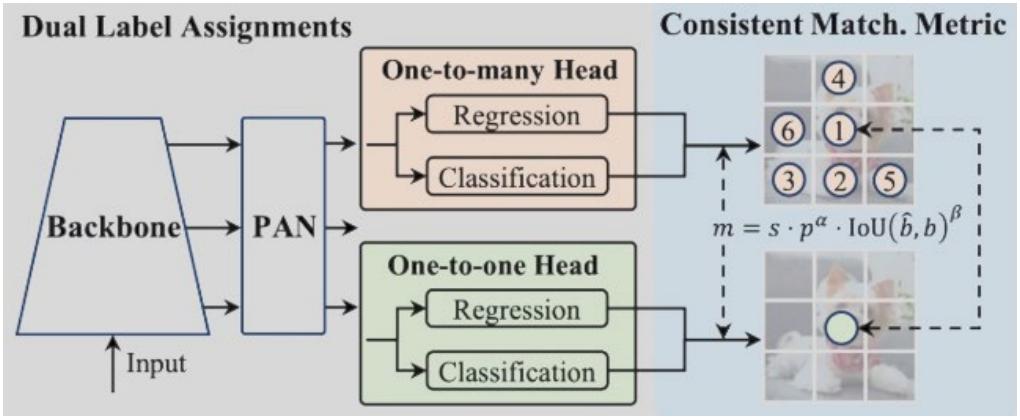

Figure 2-14 Consistent dual assignments for NMS-free training[10]

YOLO11 through to YOLO26 (the naming convention was changed from YOLOv to YOLO at YOLO11) have been released by the time of writing, and new versions are constantly being developed and released[11]. YOLO models can be used by serving them with TensorFlow and

[8] https://docs.ultralytics.com/models/yolov8/
[9] https://docs.ultralytics.com/models/yolov9/
[10] https://docs.ultralytics.com/models/yolov10/
[11] https://docs.ultralytics.com/models

other software, such as PyTorch (Li, Zhao, *et al.*, 2020), which is a Python library that is used for deep learning (Imambi *et al.*, 2021).

Looking at the YOLO versions together, two broad trends can be seen rather than a simple list of improvements. The first is architectural. Over successive versions, YOLO has gradually moved away from manually set components towards ones that are either learned or removed entirely, from the anchor boxes and Non-Maximum Suppression used by the early versions, to the anchor-aided training of YOLOv6 (Li et al., 2023), the anchor-free detection head of YOLOv8, and the NMS-free inference of YOLOv10 (Wang et al., 2024). The second trend is a shift in focus from redesigning the network to improving the training process, as seen in the "trainable bag-of-freebies" introduced by YOLOv7 (Wang, Bochkovskiy, et al., 2023).

For large, general-purpose datasets, these changes often make only small differences to accuracy, as newer versions tend to be incremental improvements over older ones (Redmon & Farhadi, 2018). They matter more for deployment, where lighter versions such as Fast YOLO and YOLO Lite were developed specifically for embedded devices or computers without GPUs (Huang et al., 2018; Shafiee et al., 2017). This suggests that the choice of YOLO version should be based on the available hardware and dataset rather than simply using the newest version, which is something this study examines by testing whether smaller and older models remain competitive when hardware is limited.

2.4 ALPR

ALPR, also called Automatic Number Plate Recognition (ANPR) (Agbemenu *et al.*, 2018) or Automatic Automobile License Plate Recognition (Wang *et al.*, 2010), is a process where vehicle license plates are extracted from images and the characters converted into computer readable text (Du *et al.*, 2013). The ALPR process can be performed using Deep Learning implementations, such as the YOLO framework; or with computer vision-based approaches, such as OpenALPR (Laroca *et al.*, 2018) .

The process is typically done by either first finding the license plate on the image (license plate detection) (Agbemenu *et al.*, 2018), which can, for example, be done by using a sliding window across the whole image (Prates *et al.*, 2013), or by first finding the vehicle and then the license plate (Gonçalves *et al.*, 2016), which is the most important step of the process, since the following steps are redundant if not applied to the correctly detected license plate (Laroca *et al.*, 2018). Enhancements can then be applied to the license plate image to make the following steps easier (Arulmozhi *et al.*, 2012).

The next step is isolating the characters on the license plate (character and line segmentation, called plate segmentation or license plate character segmentation), which is crucially important for the character recognition step (Agbemenu *et al.*, 2018; Gonçalves *et al.*, 2016). The final step is usually character recognition, which can be done with libraries such as Tesseract-OCR.

Use cases for ALPR include traffic control, traffic law enforcement, toll collection and security (Du *et al.*, 2013).

2.4.1 OpenALPR

OpenALPR is an open-source library written in C++, with bindings (software which allows a software developer to use languages other than the one the software was written in to interact with the software (Li *et al.*, 2024)) in other languages, such as C#, Java, NodeJS, Go and Python[12]. The OpenALPR library uses computer vision algorithms to detect license plates on an image and recognises the characters on the license plate using Tesseract OCR.

Figure 2-15 shows the processing steps used by OpenALPR. It is adapted from (Wan Ismail, 2021).

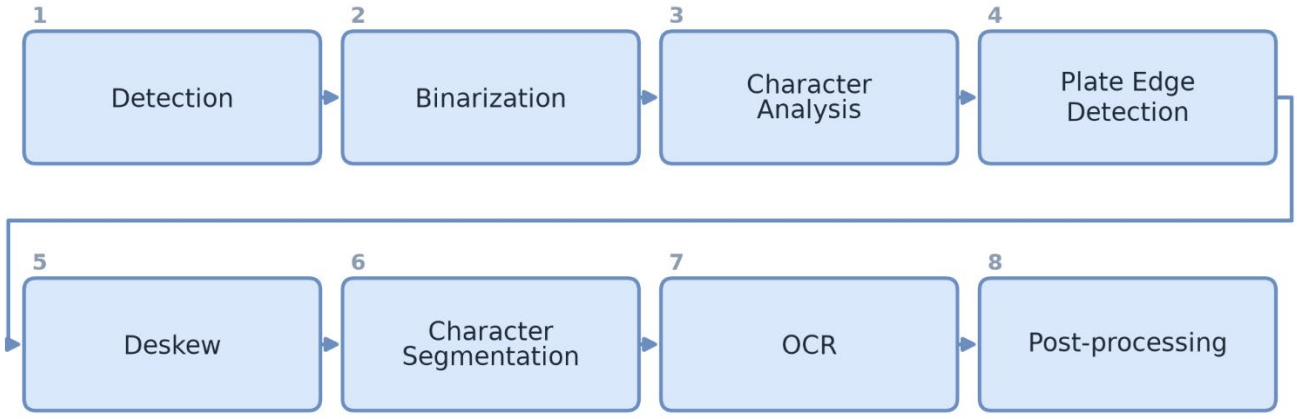

Figure 2-15 OpenALPR processing steps

2.4.2 Comparison of ALPR approaches

The two approaches described above, computer vision-based pipelines such as OpenALPR and deep learning-based detectors such as YOLO, each have advantages and disadvantages that are worth comparing directly. Computer vision approaches break the recognition process into

[12] https://github.com/openalpr/openalpr

separate steps, such as edge detection, binarization and Tesseract OCR. This makes them easier to interpret and removes the need for training data. However, because each step is designed for expected conditions, these approaches can struggle when license plates are skewed, unevenly lit or distorted (Shi et al., 2019; Yoon, 2013), and often require manual adjustment to restore accuracy.

Deep learning approaches work differently, as they learn features directly from labelled data. This allows them to handle the angles and lighting variations that computer vision pipelines struggle with (Laroca et al., 2018), but it also requires large, labelled datasets, which can be expensive and time-consuming to gather (Xu et al., 2023), as well as more computing power for both training and inference. As labelled data and GPUs have become more available, research has increasingly moved from computer vision approaches towards deep learning approaches (Laroca et al., 2018). However, there is limited research comparing how well this improved accuracy holds up when the required hardware is not available, which is the focus of this study.

2.4.3 ALPR in South Africa

Most of the literature reviewed so far has a global focus, but South African license plates have specific characteristics that make local research worthwhile. Vehicle registration in South Africa is administered at a provincial level under the National Road Traffic Act 93 of 1996 (South Africa, 1996), and each province issues plates with its own identifying codes and, in several cases, its own layout and colour scheme, so a single national format cannot be assumed. These systems also change over time; KwaZulu-Natal, for example, introduced a new provincial number plate system in December 2023 (SAnews, 2023). These provincial differences mean that a model trained on plates from one region, or from another country, will not necessarily perform well on South African plates without local data.

Despite these differences, ALPR is already deployed at scale in South Africa. The largest example is the South African National Roads Agency (SANRAL) Open Road Tolling system on the Gauteng freeway network, which uses ALPR to identify vehicles for tolling. Robinson and Venter (2019) analysed more than 79 million records from this system and reported a plate-matching rate of over 95%, concluding that the ALPR data was at least as reliable as conventional traffic counting methods. This shows that ALPR is practical under South African conditions on well-controlled, fixed-camera installations, although such installations are very different from the lower-cost, general-purpose hardware considered in this study.

In contrast to this industrial deployment, localised academic research on South African ALPR is relatively limited. Early South African work on plate segmentation and recognition was presented

at a South African pattern recognition workshop in the 1990s (van Heerden, 1996), but comparatively little research has focused specifically on South African plates since then, particularly research aimed at low-cost or developing-context deployments. This is the gap that this study addresses: rather than assuming access to dedicated tolling infrastructure, it evaluates how accurately and how quickly South African plates can be recognised using affordable, general-purpose hardware, which is more representative of what a typical municipality or small operator in a developing context could deploy.

2.5 Character segmentation

Character segmentation is the process where the text characters are located (Tian *et al.*, 2015) and the background of the image is separated (Guo & Liu, 2008). Character segmentation that focuses on segmenting characters from license plates is called License Plate Character Segmentation (LPCS) (Gonçalves *et al.*, 2016; Wang *et al.*, 2010). Character Segmentation approaches for documents often differ from LPCS, since documents do not usually have the same contextual information license plates have (Gonçalves *et al.*, 2016). Documents are, however, affected by some of the problems license plates face, such as text skew (AL-Khatatneh *et al.*, 2015; Razak *et al.*, 2008).

LPCS involves dividing the license plate into the smallest rectangles that contain only a single character (Wang *et al.*, 2010). License plates that have been extracted can be tilted and have differing levels of brightness, which can adversely affect character recognition. The license plate tilt can be removed with a bilinear transformation which maps the tilted license plate onto a straight rectangle (Du *et al.*, 2013), or the horizontal and vertical tilt can be removed using a least-squares method (Wang *et al.*, 2010).

Character segmentation methods include vertical projection based methods, where a vertical histogram is used to find the valleys separating the characters; connected component analysis (CCA) based methods, where techniques such as edge detection, thresholding, resampling and filtering are used to locate and isolate the license plate and characters by finding connected components on the image with the same aspect ratio as the characters; Hough transform based methods; Vector quantization based methods; and Hidden Markov Chains based methods (Jagannathan *et al.*, 2013; Tian *et al.*, 2015).

Line segmentation is a process where the OCR engine attempts to detect the lines of text and segment them (Razak *et al.*, 2008) and is used during Document Image analysis (Romero *et al.*, 2015). It is an important step because inaccurate line segmentation will cause problems in the

recognition stage of OCR (Razak *et al.*, 2008). Line segmentation can be performed using techniques such as vertical projection profiles, and connected components analysis (Romero *et al.*, 2015).

2.6 Optical character recognition

OCR is a large field of computer science with many different applications (Singh, Bacchuwar, *et al.*, 2012). It is the electronic translation of non-machine written text into machine text (Singh, Bacchuwar, *et al.*, 2012) and is a common method of digitizing text so that it can be edited on a computer (Asif *et al.*, 2014). This study will focus primarily on using Tesseract as well as various machine learning approaches for performing OCR.

2.6.1 Machine learning approaches to OCR

Machine learning can be used to perform OCR. This approach is similar to the process for extracting license plates, where it uses a model trained to read the text rather than using algorithms like OpenCV, which can be difficult to properly implement, as license plate images can be captured from many angles with different distortions, as opposed to scanned documents where the text is usually more uniform (Shi *et al.*, 2016). Distorted text can be difficult for algorithms to handle (Shi *et al.*, 2019) and often requires specific preprocessing to remove the distortion.

Some examples of these approaches are ASTER and MORAN. ASTER stands for attentional scene text recognizer with flexible rectification. It works by first running the image through a text rectification network which flattens the text based on a morphological analysis, essentially making it more similar to text in a scanned document (Shi *et al.*, 2019). This processed image is then pushed through a text recognition network which recognises the characters on the image itself. MORAN[13] is an acronym for multi-object rectified attention network (Luo *et al.*, 2019) and can be compared to ASTER, since it performs rectification on the text before performing text recognition. These approaches are designed for scene text recognition (not license plates specifically), so they are not explicitly used in the experiments, but were included in this review to demonstrate the viability of completely ML-based OCR as an alternative to mixed-and-algorithmic approaches such as Tesseract.

[13] https://github.com/Canjie-Luo/MORAN_v2

2.6.2 Tesseract

Tesseract is an implementation of OCR that was developed by the software company HP between 1984 and 1994 (Smith, 2007), although some sources put the development date between 1985 and 1995 (Smith, 2013). It was released as Open-Source software[14] in late 2005 or 2006 and is now mostly developed by Google (Badla, 2014; Smith, 2013). It is a portable character recognition engine (Agbemenu *et al.*, 2018) and can run on many operating systems (Badla, 2014). Tesseract started as a PhD project and eventually became a candidate hardware and software add-on for HP's flatbed scanners. The research was motivated by the poor OCR engines that could not read poor quality print (Smith, 2007).

Initial Tesseract development focused more on increasing its efficiency in relation to rejecting incorrect results, rather than increasing its ability to produce accurate results (Smith, 2007). Tesseract is an example-based system where the system uses a set of example rules that can be changed or modified, depending on the requirements (Badla, 2014). Tesseract can detect if the text on an image is mono-spaced or proportional. It has Unicode (UTF-8) support and can recognise words in over 100 languages without any additional setup (Palekar *et al.*, 2017).

The first step tesseract performs is a CCA in which the various components of an image are identified and stored (Smith, 2007). By inspecting the nesting of outlines and the number of child/grandchildren outlines, it is simple to detect inverse text as well as recognising it as black and white text (Smith, 2007). The outlines are gathered via nesting into data structures called Blobs, which, in turn, are organized into text lines, which then are analysed for fixed pitch or proportional text (Smith, 2007).

Tesseract can be further trained to increase the accuracy of the OCR it performs for specific use cases (Sun & Zhao, 2020), for example recognising another language, such as Ancient Greek (White, 2012). The first step involves teaching Tesseract the shape of the characters by feeding it images with text, and then specifying the coordinates and UTF-8 code of each character, which are then saved as TIF files which contain images of the characters, and BOX files which contain the UTF8-code and coordinates of the characters (Nayak & Nayak, 2014).

After the files have been generated, Tesseract must be run in training mode and with the TIF and BOX files as parameters, which will then generate a TR file (Nayak & Nayak, 2014). If another character set is being used, the *unicharset_extractor* program can be fed the BOX file to generate a *unicharset* file that contains the font and character properties, such as *isalpha*, which determines

[14] https://github.com/tesseract-ocr/tesseract

if the characters are alpha characters (Nayak & Nayak, 2014). These properties can be manually edited and should then be saved as a *font_properties* file which. together with the TR file. will be used to run the *mftraining*, which generates the *inntemp* and *pffmtable* (Sun *et al.*, 2019), and *cntraining* programs, which generate the normproto files (Sun *et al.*, 2019). The *pffmtable*, *inntemp*, *normproto* and *unicharset* files are the minimum number of files that must be generated[15].

While this study uses the default English-language Tesseract model, the training process documented here can be used in future work if the OpenALPR component ever needs extending.

2.6.3 Comparison of OCR approaches

The two OCR approaches described above reflect a similar trade-off to the one seen in license plate detection. Tesseract works best on uniform, document-like text, which is what its connected-component process was originally designed for (Smith, 2007). It tends to struggle with distorted or angled text, however, as its fixed processing steps do not handle these variations well (Shi et al., 2016). The machine learning recognisers ASTER and MORAN address this weakness by rectifying the text before recognition (Luo et al., 2019; Shi et al., 2019), but, like other machine learning approaches, they require training data and more computing power. The choice between them therefore depends on the situation. This study uses Tesseract, as it does not require training and integrates easily with the OpenALPR pipeline, even though it can struggle with distorted plates, while fully machine learning-based recognition offers a more robust but more demanding alternative for systems with more resources available.

2.7 Edit distance

When referring to strings, edit distance is a metric used to determine string similarity, i.e. the minimum number of character insertions (adding a character to a string), deletions (removing a character from a string), substitutions (removing a character from a string and then adding another character in the exact same place) (Ristad & Yianilos, 1998) or transposition (the moving of a character from one position in a string to another) (Ukkonen, 1983). The subsections below explore methods of determining the edit distance briefly, with Table 2-2 showing a comparison of the features of various edit distance algorithms.

Table 2-2 Comparison of edit distance algorithms

[15] https://tesseract-ocr.github.io/tessdoc/tess3/Training-Tesseract-3.00–3.02.html

Algorithm	Operations					Different Lengths?
	Insertion	Deletion	Substitution	Transposition	Other	
Hamming			X			No
Jaro-Winkler				X		Yes
Kendall-Tau					Uses rank correlation	No
Levenshtein	X	X	X			Yes
Damerau-Levenshtein	X	X	X	X		Yes
LCS	X	X				Yes

2.7.1 Hamming distance

The original application for Hamming was determining the error introduced by noise over a channel when a message is sent between its source and destination (Bookstein *et al.*, 2002; Hamming, 1950). The errors in these cases refer to substitutions, and Hamming measures the number of substitutions that transforms one string into another. Since Hamming measures substitution and not Insertion or Deletion, the strings that are compared must be of the same length (Metcalf & Casey, 2016). The Hamming distance between $x, y \in \{0,1\}^n$ is defined as

$$d_H^n(x,y) = \sum (i=1)^n 1(x_i \neq y_i), \tag{2.5}$$

where $1(\cdot)$ is the indicator function (Pang & El Gamal, 1986).

2.7.2 Jaro-Winkler distance

Jaro-Winkler measures the distance between two strings by first measuring the length of the strings, then finding similar characters between the strings, followed by finding the number of transposed characters (Friendly, 2019). It only measures transpositions, not additions, deletions or substitutions (Cheatham *et al.*, 2017; Jaro, 1989). Jaro-Winkler is based on the Jaro similarity metric, defined as

$$d_j = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) \tag{2.6}$$

where *m* is the number of matching characters, *t* is half the number of transpositions and s_1 and s_2 are the lengths of strings (Cheatham *et al.*, 2017). Jaro-Winkler is an extension of the Jaro distance based on Winkler's observation that typing errors occur most commonly in the middle or

end of the word, with emphasis put on matching prefixes if the Jaro distance exceeds a certain threshold, defined as

$$d_w = d_j + \ell p(1 - d_j), \tag{2.7}$$

where ℓ defines the length of the common prefix and p is a weighting factor (Cheatham *et al.*, 2017).

2.7.3 Kendall Tau distance

Kendall Tau is not traditionally used to compare strings character by character (Brossart *et al.*, 2018). It works by measuring the minimum number of adjacent swaps (i.e. transpositions) to convert one element into another, often by comparing the relative ordering of elements in two lists or sequences, rather than string characters (Cicirello, 2019). The Kendall Tau rank correlation coefficient is defined as

$$T(\sigma_1, \sigma_2) = \frac{2}{n * (n - 1)} \sum_{i,j \in S \wedge i<j} sign(\sigma_1(i) - \sigma_1(j)) * sign(\sigma_2(i) - \sigma_2(j)) \tag{2.8}$$

,where σ_1 and σ_2 are permutations (Cicirello, 2020). Kendall-Tau distance is similar, with the simplest definition being

$$K(\sigma_1, \sigma_2) = |D| \tag{2.9}$$

where *D* is a set of discordant pairs (Cicirello, 2020).

2.7.4 Levenshtein distance

Levenshtein distance is an algorithm used to measure the distance between two strings (Haldar & Mukhopadhyay, 2011). Using Levenshtein, the similarity between two strings can be quantified and represented as a numerical value. The Generalized Levenshtein Distance (GLD) is a common method of using Levenshtein where it compares deletion (one string is missing a character that another contains), insertion (one string contains a character that the other does not), and substitution (one string has a character that is different in the other string) where the sum of these differences is called the edit distance (Yujian & Bo, 2007). Given $X, Y \in \Sigma^*$, GLD is defined as

$$GLD(X, Y) = \min \gamma(T_{X,Y}) \tag{2.10}$$

where $T_{X,Y} = T_1 T_2 \ldots T_i$ is used to denote an edit operation (Yujian & Bo, 2007).

2.7.5 Damerau–Levenshtein distance

Damerau-Levenshtein is an extension of Levenshtein that also considers transpositions, rather than just additions, deletions and substitutions, defined as (Zhao & Sahni, 2019)

$$d_{DL}(i,j) = min \begin{cases} d_{DL}(i-1,j)+1 & (deletion) \\ d_{DL}(i,j-1)+1 & (insertion) \\ d_{DL}(i-1,j-1)+c(s_1[i],s_2[j]) & (substitution) \\ d_{DL}(i-2,j-2)+1 & if\ s_1[i]=s_2[j-1] \wedge s_1[i-1]=s_2[j] \end{cases} \quad (2.11)$$

2.7.6 Longest common subsequence

Longest common subsequence (LCS) measures the number of insertions and deletions to convert one string into another one (Bergroth *et al.*, 2000). It starts by finding the longest subsequence of characters common to both strings and then measuring how many insertions and deletions are needed to change one string to another (Apostolico & Guerra, 1987; Paterson & Dančík, 1994). The traditional technique is to determine the LCS for all possible prefix combinations of the input strings, defined by (Bergroth *et al.*, 2000):

$$R[i,j] = \begin{cases} 0 & if\ i=0\ or\ j=0 \\ R(i-1,j-1)\ +\ 1 & if\ s_1[i]\ =\ s_2[j] \\ max(R(i-1,j),R(i,j-1)) & if\ s_1[i]\ \neq\ s_2[j] \end{cases} \quad (2.12)$$

2.7.7 Comparison of edit distance approaches

Although all six metrics are reported in the results as a cross-check, they are not equally suited to measuring license plate recognition accuracy, and it is worth explaining why Levenshtein distance is used as the main measure. OCR errors on license plates generally take one of three forms, as a character may be inserted, deleted or substituted, and Levenshtein is the only one of these metrics that accounts for all three within a single value (Yujian & Bo, 2007). The others each capture only part of this. Hamming distance measures only substitutions and requires the two strings to be the same length (Metcalf & Casey, 2016), which does not work when a character is missing or added. Jaro-Winkler and Kendall Tau focus on transposition and ordering rather than the full set of edits, and Kendall Tau is not normally intended for comparing strings character by character (Brossart et al., 2018). Damerau-Levenshtein adds transposition to the three operations that Levenshtein uses (Zhao & Sahni, 2019), but transposition is fairly rare in single-line license plate OCR, so it offers little more than Levenshtein in this case.

2.8 Image augmentation

A substantial number of images are required to adequately train a machine learning model. Gathering these images can be challenging and expensive (Xu *et al.*, 2023). Image augmentation involves applying image processing techniques to generate artificial data (Bloice *et al.*, 2017). This can increase the size of the training set and reduce the risk of overfitting, a phenomenon that occurs when a model matches the training data to such a high degree that it cannot match testing data anymore (Shorten & Khoshgoftaar, 2019). The image library Albumentations[16] provides many functions to augment an image, such as blurring, rotation, grayscaling, etc. (Buslaev *et al.*, 2020). A few examples of image augmentation can be found in Figure 2-16, Figure 2-17, Figure 2-18 and Figure 2-19.



Figure 2-16 Base Image without any augmentations



Figure 2-17 Augmented Image, with added skew to change the angle of the license plate

[16] https://albumentations.ai/docs/3-basic-usage/choosing-augmentations/

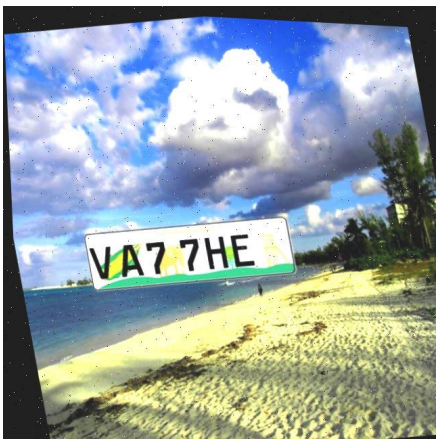

Figure 2-18 Augmented image, skewed with added noise and a random background



Figure 2-19 Various Albumentation augmentations on an image of a parrot[17]

[17] https://albumentations.ai/

2.9 Summary

The literature review considered the key techniques underpinning ALPR systems, from preprocessing and machine learning frameworks to optical character recognition and edit distance metrics. Section 2.7 identified several edit distance algorithms that can be used to measure the accuracy of ALPR implementations, addressing the first of the secondary research questions. However, there is limited research comparing different ALPR approaches across different hardware configurations using a consistent methodology. This study addresses that gap by providing a reproducible methodology for evaluating ALPR performance across hardware platforms.

Chapter 3: Methodology

This chapter discusses the methodology used to design, implement and evaluate the ALPR approaches developed in this study. Section 3.1 details the methodology used to perform the experiments, while Section 3.2 discusses the setup of the hardware and software components that are required to replicate the experiments. Section 3.3 shows the procedure used to perform the experiments.

3.1 Research methodology and paradigm

This study follows a quantitative experimental methodology that falls within the positivist paradigm, as discussed in Section 1.4. Experiments are used to measure and compare the accuracy and processing speeds of different ALPR methods across various hardware configurations. By using these quantitative methods, the results can be determined objectively through analysis.

The independent variables in this study are the hardware configurations and the ALPR methods. The dependent variables are the accuracy (measured via edit distance, as discussed in Section 2.7) and processing time per image. Controlled variables include the image resolution (640x640), image format (JPEG) and the software versions shown in Table 3-2.

3.2 Experimental setup

This section will detail the hardware setup, software setup and experimental procedure required to perform and replicate the experiments that will be detailed in Chapter 4.

3.2.1 Hardware setup

This study will use three levels of hardware to perform the experiments (low-level, intermediate level and high-level). If an experiment cannot be performed on one of the setups (possibly due to incompatibility with the software, or insufficient hardware to run a certain model), it will be reported in the results.

The experiment will still be performed on all of the setups that are compatible; for example, larger machine learning models will likely not work on the low-level machine (even medium-sized models are large, with the largest YOLOv10x model being over 23 times larger than the YOLOv10n model[18]), but they might work on the intermediate- or high-level machines.

[18] https://docs.ultralytics.com/models/yolov10/#performance

There might also be compatibility issues with some of the software, since the low-level machine uses a different CPU architecture (ARM compared to x86) than the intermediate-and high-level machines. Table 3-1 provides a detailed description of the hardware used to perform the experiments for this study.

Table 3-1 Hardware configuration for the systems used in the study

Component	Low-level	Intermediate-level	High-level
Device	Raspberry Pi Model 4 B	Custom-built (second-hand)	Custom-built (new)
CPU	Broadcom BCM2711 Quad-core Cortex-A72 @ 1.8 GHz	AMD Ryzen 5 3600X, 6 cores / 12 threads, 3.8 GHz base / 4.4 GHz boost, 7nm, AM4	AMD Ryzen 7 7800X3D, 8 cores / 16 threads, 4.2 GHz base / 5.0 GHz boost, 5nm, AM5
Memory	4 GB LPDDR4-3200	16 GB DDR4-3600	64 GB DDR5-6000
GPU	Integrated (OpenGL ES 3.1, Vulkan 1.0)	NVIDIA GeForce GTX 1080 Ti	NVIDIA GeForce RTX 3080 Ti
Storage	Micro-SD card	SSD	SSD
Networking	Gigabit Ethernet	Gigabit Ethernet	Gigabit Ethernet
Power supply	5V DC via USB-C (3A)	800W consumer PSU	1000W consumer PSU
Operating System	Raspbian x64 (2024-11-19)	Windows 11 23H2	Windows 11 23H2

The tests were all performed on freshly booted machines; however, the machines did not have a clean install of Windows. The intermediate-level computer previously had Windows 10 installed, and was upgraded to Windows 11, while the high-level computer came with Windows 11 pre-installed. Both versions of Windows were upgraded to 23H2 and the tests were all run on this version of Windows 11.

3.2.1.1 Low-level machine

The Raspberry Pi Model 4 B was selected for this study due to its affordability, availability, and the fact that it uses a Linux-based OS. Additionally, it is relatively easy to set up and operate, making it an ideal choice for the testing requirements of this research.

The Raspberry Pi came equipped with a 32-bit operating system (Raspbian). However, this was not suitable for some of the software that was used, like the Ultralytics library. To resolve this problem, the 64-bit version of the OS (Raspbian x64, version 2024-11-19)[19] was downloaded and

[19] https://www.raspberrypi.com/software/operating-systems/

flashed onto the removable storage the Raspberry Pi came equipped with and the installation was completed after booting with the flashed storage (Hollingworth, 2022).

3.2.1.2 Intermediate-level hardware

A second-hand computer was purchased to simulate an average level of hardware that might be available for an end-user. It uses an older generation of CPU and GPU, but it does not use the cheapest possible parts. This machine is supposed to represent what a typical consumer would have access to, without having to resort to buying the most expensive components. While still an intermediate-level machine in 2026, it can be considered on the lower-end of intermediate-level hardware.

3.2.1.3 High-level hardware

A new computer was built using high-level consumer-level hardware. Higher-end and more expensive hardware that is even better optimised for Machine Learning than this setup is available. At the time when the experiments were performed, this machine was a high-end computer, but it would be classified at the higher levels of intermediate-level hardware in 2026, with the GPU already being two generations behind the latest offerings from Nvidia.

3.2.1.4 Excluded hardware

The following hardware is not tested as part of this study:

- Cameras with built-in ALPR
- Server-level hardware
- AI-specific hardware
- Data centres, i.e. cloud computing

The main factor preventing the testing of these devices is cost; the experiment can be replicated by others if they have access to the hardware, since the trained models used are freely available.

3.2.2 Software setup

Python is the language used to build the software pipelines used in this study. It comes bundled with a package manager called pip that is used to install and manage packages.

The core software components that can be used for both the machine learning approach and the non-machine learning approach are OpenCV and Tesseract. OpenCV is a general library that contains many useful image processing and manipulation functions, such as, for example, grayscaling. Tesseract forms the basis for the OCR performed on the extracted license plates.

OpenALPR will be primarily used for the non-machine learning approach, but is not exclusive to it. YOLO is the primary machine-learning approach to performing ALPR in this experiment. CVAT is used to annotate the images, and Albumentations is used to augment the images. The versions and installation methods for these software platforms and libraries can be found in Table 3-2.

Table 3-2 Installation instructions for the software used in the study

Software	Version	Install Method	Notes
Albumentations	2.0.6	pip install albumentations	Will install the latest version and all dependencies
CVAT	N/A	CVAT.ai website	Account might be required to use
OpenALPR	2.3.0	Compiled from source[20]	This software is extremely outdated
OpenCV	4.9.0	pip install opencv-python	Can also be installed from https://opencv.org/releases/
Python	3.14	Downloaded from website	See https://www.python.org/downloads/
Tesseract	4.1.2	Depends on OS	See https://tesseract-ocr.github.io/tessdoc/Installation.html
YOLO	All	pip install ultralytics	Ultralytics hosts all the YOLO versions used for this study

OpenALPR is noted to be outdated; the paid version is actively being developed, but the free version was last updated in 2020. The software does have Python bindings available on pip, but it is not enough to get a functioning version of the program. OpenALPR needs to be compiled and installed first, by downloading the source code and using the instructions found in its source code repository, before running the pip install. The Python bindings might also need to be changed to the installed OpenALPR directory, since the default installation directory might differ from the directory the Python bindings point to.

The YOLO weights are simple to download from Ultralytics. After installing Ultralytics via pip, the weights can be downloaded at[21]. Thereafter run the following command in the directory where the training output needs to be created:

```
yolo task=detect mode=train model=<model name, e.g. yolov8n>.pt
device=<device number of GPU, typically 0 in most machines with a single
GPU> data=<data.yaml directory on file system> epochs=<number of epochs
(complete passes through the training data set), default 100> imgsz=640
```

[20] https://github.com/openalpr/openalpr
[21] https://docs.ultralytics.com/models/yolov5/#usage-examples

The instructions within the “<>” brackets must be replaced for the command to be run successfully.

3.3 Experimental procedure

This section will detail the procedure followed to design, run and evaluate the results of the experiments. Figure 3-1 is a flowchart that details the procedure from start to finish.

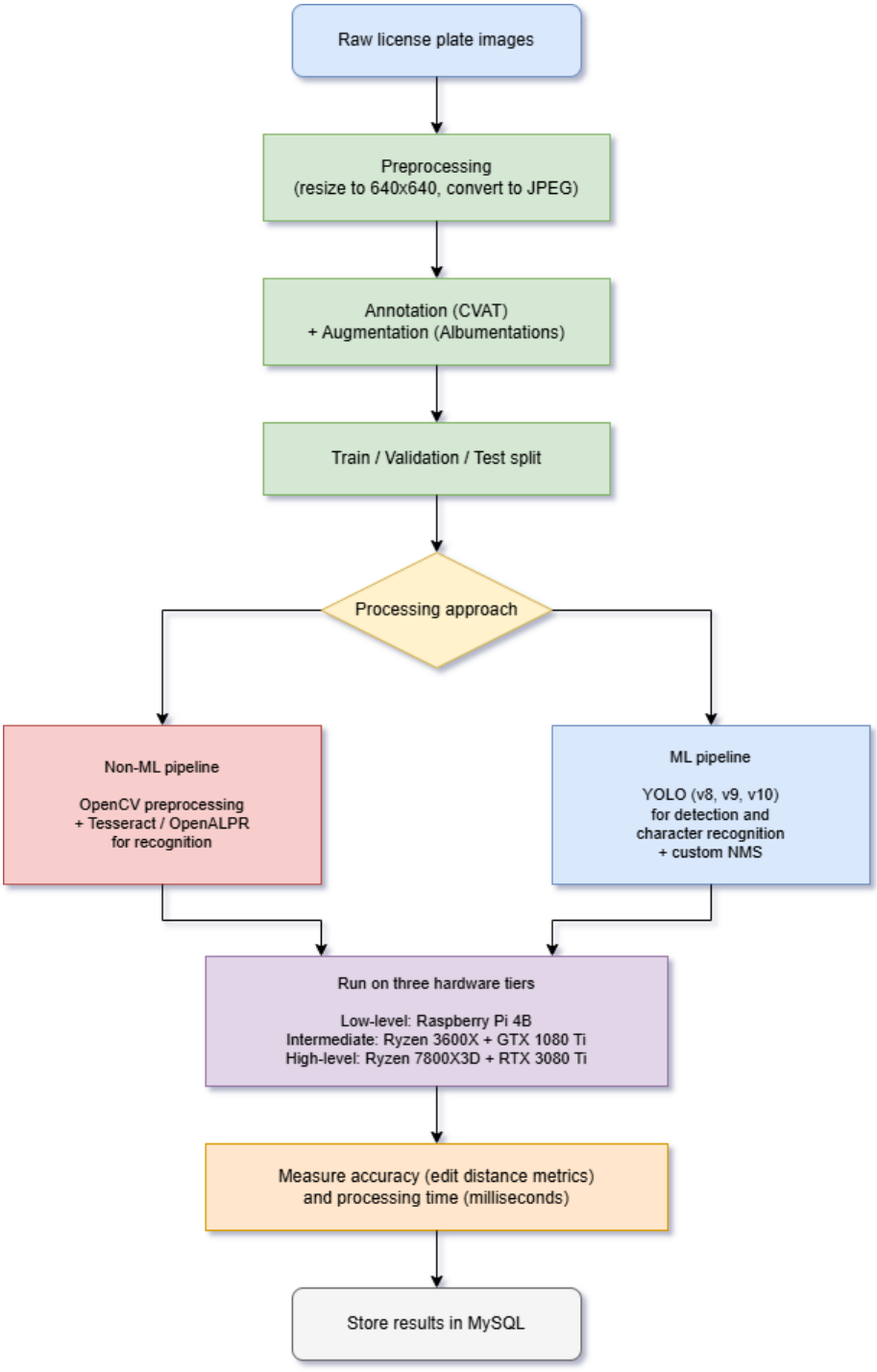

Figure 3-1 Overview of the experimental pipeline from data preparation to result storage

3.3.1 Data preparation

Images were sourced from publicly available license plate[22] datasets[23], with the initial set containing a mixture of South African and international license plates. Images were resized to 640x640 where possible to keep the files under 100KB, and all images were converted to JPEG to ensure the format is consistent for the models, as described in Section 1.3.1.

Many of the publicly available license plates were already annotated and where possible this annotation was used, however very few were annotated on a character level – they were only annotated on the license plate level, so CVAT was used to annotate the characters on these images. This is an extremely time-consuming process, since each image usually has between 5 and 10 small characters that need to be annotated.

Upon moving to local specific images, there was a need to generate additional images due to the limited availability in public datasets. This also helped ensure no one's privacy was violated. These generated license plates were also annotated with CVAT.

3.3.2 Training and validation

The final dataset consisted of 3500 training images and 360 validation images for the locally focused experiments. A dataset of this size is relatively small for training a deep neural network from scratch, as deep learning models generally need a large amount of labelled data to learn effectively (Xu et al., 2023). To offset this, the models were not trained from scratch but were instead fine-tuned from pretrained weights, an approach known as transfer learning. Rather than starting from random values, the YOLO weights downloaded from Ultralytics (the .pt files used in the training command in Section 3.2.2) are already trained on a large, general-purpose dataset, namely COCO (Zhao & Li, 2020).

These weights already contain general visual features such as edges and shapes, so training on the smaller license plate dataset only needs to adapt them to this specific task rather than learn everything from the beginning. This reduces the amount of data required to reach usable accuracy.

The effective size of the training set was further increased using image augmentation (discussed in Section 2.8), which generates additional variations of the existing images and helps reduce the risk of overfitting (Shorten & Khoshgoftaar, 2019). Together, transfer learning and augmentation make a dataset of this size sufficient for the fine-tuning performed in this study. Gathering a

[22] https://universe.roboflow.com/licence-plate-recognition
[23] https://universe.roboflow.com/roboflow-universe-projects/license-plate-recognition-rxg4e

substantially larger set of locally focused, character-level annotated images would also have been very time-consuming (Xu et al., 2023) and, for real plates, would have raised the privacy concerns noted in Section 3.3.1.

Small subsets of these, usually 200, 400, etc. were used to calibrate the software to make sure the training could run successfully, and the generated models could give at least rough results. Every model was trained using 100 epochs.

Multiple YOLO versions were trained and evaluated, with a special emphasis on YOLOv8-YOLOv10, since they were the latest available at the time the experiments were being conducted. For each version, both the small and tiny model variants (or whichever naming convention was used for the equivalent[24]) were trained.

3.3.3 Performing the experiments

Each experiment was calibrated with ten images, first to verify that the model was giving results (some failed training runs had caused hundreds of detections on a single image; this was caused by having mislabelled annotation files). After calibration the experiment was performed on the mixed dataset, and finally it was performed on the local, South African only dataset.

3.3.4 Data collection

Data was automatically collected into a MySQL data table with the following fields:

- Unique ID: A unique identifier for each entry, used to make entries unique in a relational database in cases where all other fields might be duplicated;
- System: The hardware the test was run on;
- Method: The predefined set of preprocessing and processing methods used;
- Prediction: The license plate prediction result obtained from the software;
- Result: The correct, human-identified license plate prediction result; and
- Difference: The difference between the result and prediction fields expressed as a percentage. This is calculated using several edit distance algorithms.

[24] YOLO does not have standard nomenclature for model sizes between different versions

3.3.5 Evaluation

Accuracy was measured using the edit distance metrics discussed in Section 2.7, where possible. Processing time for each run was also captured at the image level, by capturing the time right before and right after each image was processed.

Each experiment was run three times to ensure the consistency of results. The accuracy results were identical across all three runs for each configuration, since the models are deterministic when given the same input (i.e. they always give the same output given the same input). Processing times varied slightly between runs, likely due to environmental factors such as the amount of heat in the system and currently running background processes. The mean processing time across the three runs is reported in Chapter 4.

3.4 Risks to validity

Any aspect of the experiment that might raise doubt as to whether the experiment was accurate or meaningful risks the validity of the research (Rogers & Révész, 2019). The types of validity that will be explored below are internal, external, and construct validity (Rogers & Révész, 2019; Smith, 2005).

In terms of internal validity, the intermediate-and high-level machines were not clean installs and could have background processes that affect the timing. As a validity measure, the experiments were run after a clean boot, and all experiments were run three times with the mean processing time being used.

Regarding external validity, the results are specific to South African license plates, and daytime conditions under clear weather. Further work might explore nighttime conditions and inclement weather, as well as training models for other regions.

For construct validity, edit distance metrics measure character-level accuracy, but don't fully consider how valuable certain partial reads might be. This is mitigated by using multiple different edit distance formulas, not relying on any single one for a source of truth.

3.5 Summary

In this chapter the research methodology and paradigm, experimental setup and procedure used in this study were detailed. The experimental setup section discussed the hardware and software setup, with the three hardware configurations being established, ranging from the low-level Raspberry Pi to the high-level consumer computer.

The software components required to perform the experiments were described, with detailed installation instructions given where required. The experimental procedure showed the data preparation, training and validation, performance of the experiments, data collection and the final evaluation of the data.

The methodology described in this chapter provides the framework for answering the second and third of the secondary research questions: comparing the recognition accuracy of different software-based approaches and measuring how different hardware configurations affect processing speed. It also provides the tools required to answer the primary research question on how one can compare the performance of various combinations of hardware and software for ALPR purposes.

The results of the experiments will be discussed in Chapter 4.

Chapter 4: Results

This chapter will collate and describe the results and detail how they were gathered from the experiments, as well as explain the choices that were made.

4.1 Universal results

There are many permutations for tests that can be performed. The tests are all divided into different methods, each with its own set of parameters used. The preliminary tests on the high-level system were conducted using Methods 2 through 7, with Method 1 reserved for calibrating the software for each system.

The methods for the initial tests are shown in Table 4-1. The “psm” flag shown in Table 4-1 refers to the page segmentation mode, which controls how Tesseract controls the layout of the input image.

Table 4-1 Description of methods used for the initial tests

Method	Detection	Preprocessing	Recognition	Notes
2	YOLOv8s	Grayscale	Tesseract (PSM 8)	Single word mode
3	YOLOv5n	Grayscale	Tesseract (PSM 8)	Single word mode
4	YOLOv3t	Grayscale	Tesseract (PSM 8)	Single word mode
5	YOLOv8s	Grayscale	Tesseract (PSM 6)	Uniform text block mode
6	YOLOv5n	Grayscale	Tesseract (PSM 6)	Uniform text block mode
7	YOLOv3t	Grayscale	Tesseract (PSM 6)	Uniform text block mode
11	OpenALPR	Grayscale	OpenALPR (confidence 1.0)	All high contrast
12	OpenALPR	Grayscale	OpenALPR (confidence 0.3)	Some high contrast
13	OpenALPR	Grayscale + bilateral filter	OpenALPR (confidence 1.0)	All high contrast
14	OpenALPR	Grayscale + bilateral filter	OpenALPR (confidence 0.3)	Some high contrast
15	OpenALPR	Grayscale + bilateral filter + global threshold	OpenALPR (confidence 1.0)	All high contrast
16	OpenALPR	Grayscale + bilateral filter + global threshold	OpenALPR (confidence 0.3)	Some high contrast

Images were first sent through the detecting network, which produced coordinates on the image that contained the detected license plate. See below the initial image, and the extracted image (in Grayscale):



Figure 4-1 Initial Image without any preprocessing



Figure 4-2 Extracted license plate image (extracted with YOLOv8s)

Each method was set up by first testing with a smaller sample size of 10 images and monitoring the results of the test. A full test with 226 images followed, running on the same parameters. Each method was saved in its own file and duplicated across the three testing machines.

Of the 226 images that were fed into the network, an average of 222 were extracted, with an overall accuracy of roughly 98.23% for the high-level system. The low-level system had a lower accuracy, at 210 license plates extracted with an accuracy of 92.92%. The midrange system had an accuracy of 98.23%, the same as the high-level system. This is likely due to both the mid-level and high-level systems using the same operating system (Microsoft Windows 11), unlike the low-level system which uses Raspbian x64.

Even in confusing images the systems were able to find license plates, e.g. Figure 4-3 contains many candidate license plates in different positions on the image, however Figure 4-4 shows that license plates can still be extracted from the image.



Figure 4-3 Confusing image where multiple different images have been merged

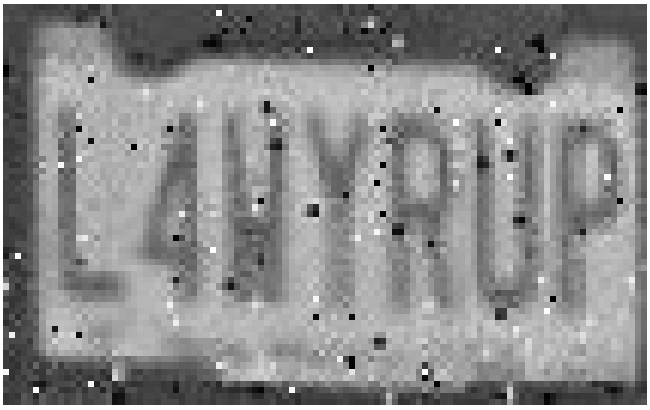

Figure 4-4 Extracted license plate image (extracted with YOLOv8s) from image with multiple license plates

However, there were instances where the license plate was clearly visible but not extracted. This may be because its position on the vehicle differs from what the model anticipates.



Figure 4-5 License plate is visible on image, but could not be extracted



Figure 4-6 License plate is visible on image, but could not be extracted

During the initial runs on the high-level system, minimal preprocessing was applied—grayscale conversion at most—which led to suboptimal results. While the detection component achieved moderately high accuracy, the text recognition component yielded very low accuracy. For example, in Figure 4-7 we have a clearly visible license plate image, but no characters could be detected or read.



Figure 4-7 Clearly visible license plate image

Table 4-2 Calibration with different YOLO configurations on high-level hardware

Method	Levenshtein	LCS	Hamming	D-Levenshtein	Jaro-Winkler	Kendall Tau	Time (ms)
2	7.01	6.50	3.43	7.01	25.40	-4.54	299.49
3	7.54	7.36	4.76	7.54	26.22	2.16	379.13
4	6.24	5.47	1.90	6.24	22.06	-4.35	903.57
5	3.15	2.40	0.00	3.15	9.21	0.07	384.40
6	4.56	3.80	0.78	4.56	11.01	-0.89	357.73
7	2.81	2.55	0.10	2.81	8.47	-0.80	883.59

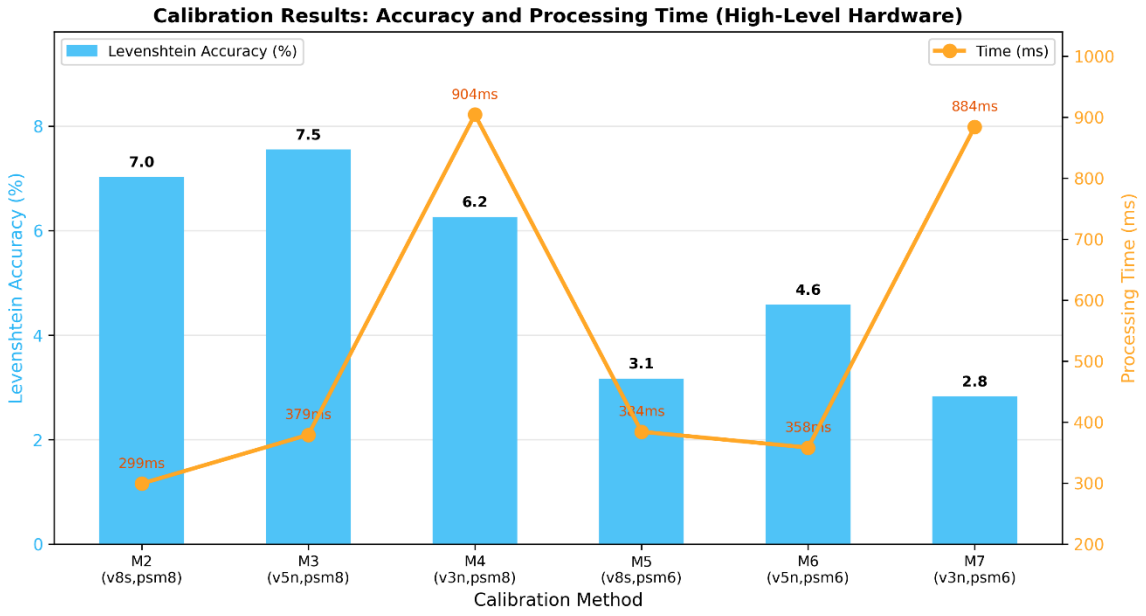

Figure 4-8 Calibration results on high-level hardware

In Table 4-2 and Figure 4-8, the values shown are the measure of accuracy or similarity to the correct, human-verified values of the data. A value of 100 would mean that the predicted results contained no errors, while a value of 0 would mean that all the data compared to the ground truth was incorrect. Do note that Kendall-Tau has negative values – this is due to Kendall Tau not

being suited for edit distance calculations. It is used to assess the correlation between two ranked lists, and while it can be used to generate a value for edit distance, the results are often unreliable for this purpose, as evidenced by the negative values in the table. For this reason, Kendall-Tau will be excluded from future results.

When tested on the low-level system, the results were similar, however the time significantly increased and certain tests could not be run successfully (e.g. any tests using Yolov3n yielded only 3-4 results before freezing), as shown in Table 4-3 and Figure 4-9.

Table 4-3 Calibration with different YOLO configurations on low-level hardware

Method	Levenshtein	LCS	Hamming	D-Levenshtein	Jaro-Winkler	Time (ms)
2	7.01	6.50	3.43	7.01	25.40	5125.32
3	7.54	7.36	4.76	7.54	26.22	2519.97
4	6.24	5.47	1.90	6.24	22.06	14473.31

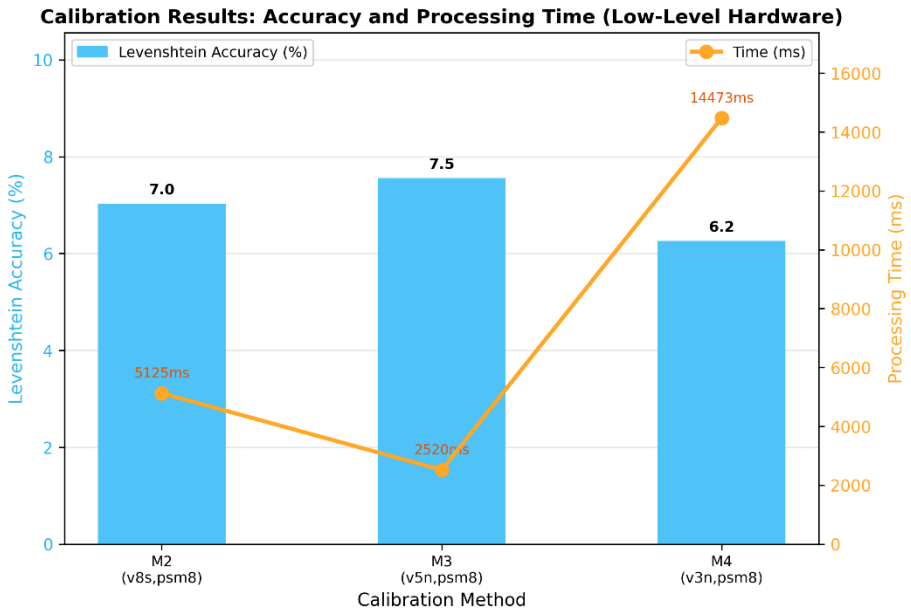

Figure 4-9 Calibration on low-level hardware

On the intermediate-level system, the results were more similar to the high-level system than the low-level system, with none of the experiments failing due to inadequate hardware or incompatibility, as shown in Table 4-4 and Figure 4-10.

Table 4-4 Calibration with different YOLO configurations on intermediate-level hardware

Method	Levenshtein	LCS	Hamming	D-Levenshtein	Jaro-Winkler	Time (ms)
2	7.01	6.50	3.43	7.01	25.40	353.38
3	7.54	7.36	4.76	7.54	26.22	435.85
4	6.24	5.47	1.90	6.24	22.06	1029.42
5	3.15	2.40	0.00	3.15	9.21	445.44
6	4.56	3.80	0.78	4.56	11.01	413.02
7	2.81	2.55	0.10	2.81	8.47	1011.65

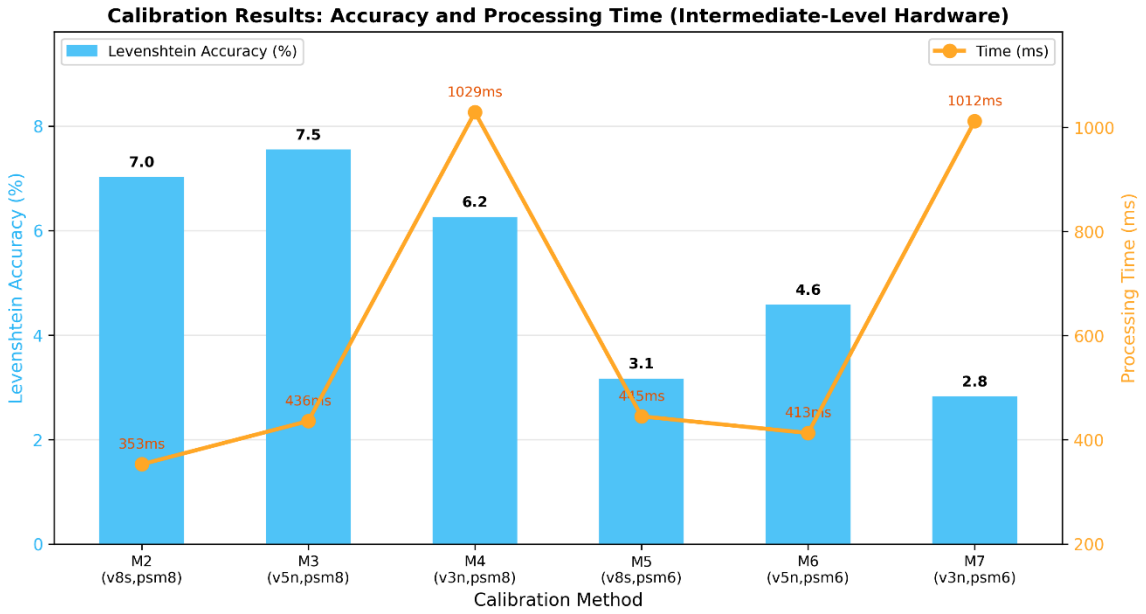

Figure 4-10 Calibration on intermediate-level hardware

The low-level system is approximately (could not capture all metrics due to incomplete data) 1600% slower than the high-level system, whereas the intermediate-level system was only 15.45% slower. However, the accuracy of the results in all 3 systems is identical, with the combined results shown on Figure 4-11.

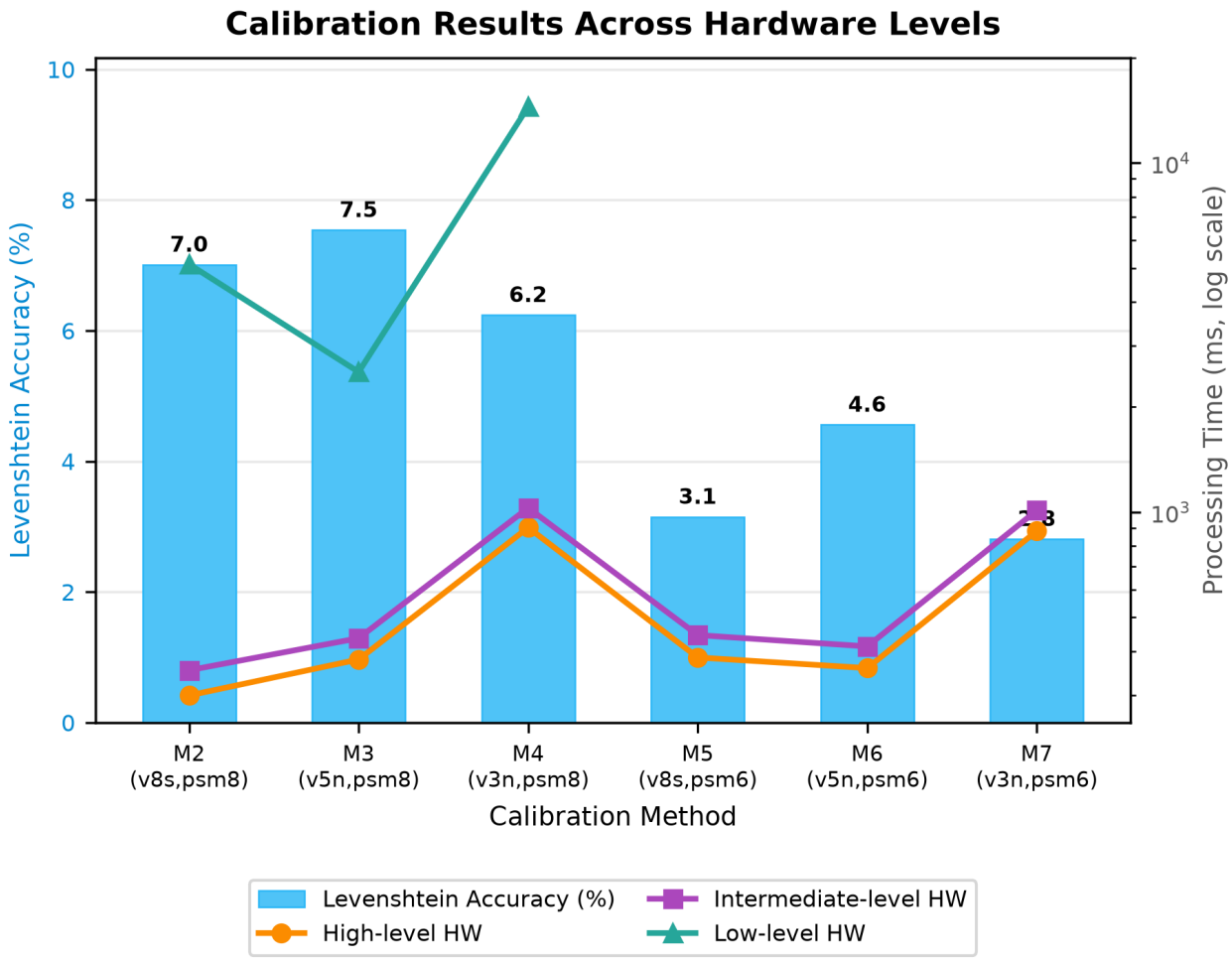

Figure 4-11 Combined calibration results for all three systems

Further tests were conducted using the same minimal preprocessing, but using the OpenALPR library, with the license plate detection component enabled. The results were significantly more promising using this, as shown here in Table 4-5 and Figure 4-12.

Table 4-5 Non-ML OpenALPR results with preprocessing

Method	Levenshtein	LCS	Hamming	D-Levenshtein	Jaro-Winkler	Time (ms)
11	37.65	32.48	37.06	37.65	46.07	729.84
12	37.56	32.36	37.31	37.56	46.13	707.18
13	35.70	30.33	33.88	35.70	43.13	708.48
14	35.74	30.30	33.88	35.74	43.48	717.37

15	19.67	16.39	16.02	19.67	24.32	525.95
16	17.19	14.00	12.11	17.19	22.32	528.83

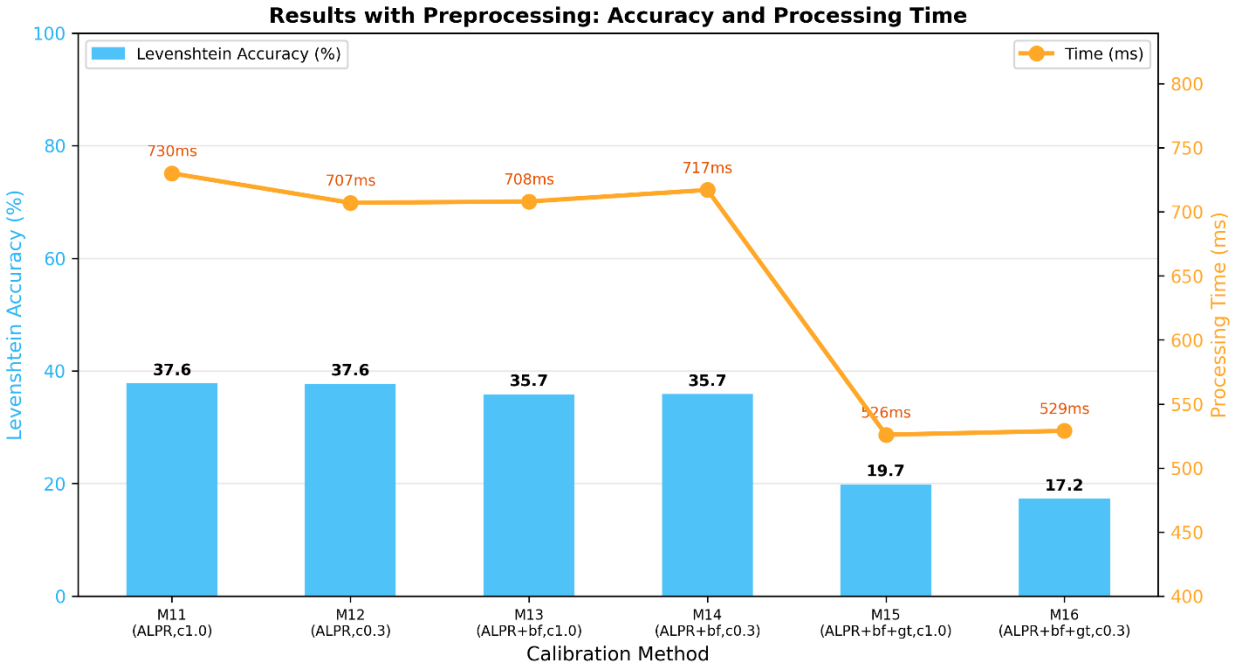

Figure 4-12 Non-ML OpenALPR results with preprocessing

OpenALPR is optimized for license plate recognition, so it performs many more calculations to get a better result. However, it takes twice as long as simply using OpenCV and Tesseract. On clear license plates, such as Figure 4-7, it gets good results: “YAANKESB”. Many of these errors can also be corrected using post recognition correction, especially if you are limiting your data set to a single country.

Because these results were still highly inaccurate, a new experiment was set up where the YOLO framework is used for both detection and recognition. The characters on the license plates were annotated, and the models were trained using the updated data set:



Figure 4-13 Annotated characters on license plate image

Given the low accuracy of the previous experiments, the test was first calibrated on the high-level machine until approximately 50% accuracy according to the Levenshtein distance metric had been reached.

To assess the viability of the experiment, a small subset of the data with only 200 training images and 20 validation images was first annotated, trained and then tested. The results of this assessment are shown in Table 4-6 with YOLOv9s performing the best and YOLOv8s performing the worst. YOLOv10s only performed slightly better than YOLOv8s, even though it is newer than YOLOv9s.

Table 4-6 YOLO calibration results with 200, 400 and 600 images

YOLO version	200 Training Images Levenshtein	400 Training Images Levenshtein	600 Training Images Levenshtein
YOLOv8s	24.43	40.30	45.51
YOLOv9s	33.68	44.52	48.92
YOLOv10s	26.27	41.43	47.96

These initial results were promising, justifying the effort it would take to annotate significantly more images (it is much more time consuming to annotate individual characters on an image compared to just annotating the license plate on the image).

The same test was then initially performed using 400 images, followed by another test with 600 images with the results also shown in Table 4-6.

Figure 4-14 shows the progression in accuracy from 200 images to 400 images to 600 images.

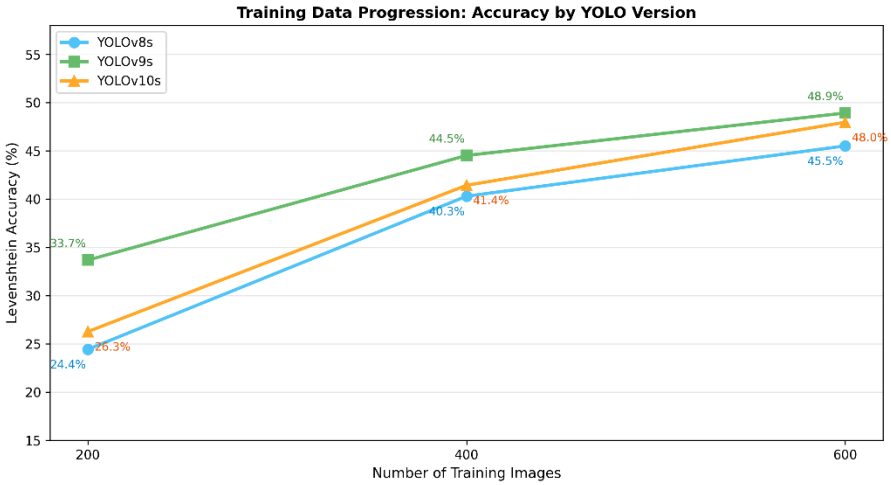

Figure 4-14 Training data progression, from 200 to 600 images

One issue with using this approach is that the YOLO NMS (Non-Maximum Suppression) that is used to remove license plates that occupy the same space can be unreliable. In Figure 4-15, the predicted license plate was 856G626. The G and the second 6 are completely overlapping, but the lowest confidence prediction was not removed.

To address the issue, a custom NMS solution was implemented in Python, designed to eliminate all predictions with over 90% overlap. On average, this adjustment resolved 4–5 overlapping

license plates per run, reducing errors by 2–3% while maintaining a negligible impact on runtime, adding only 5ms at most on a high-performance system. We can see the unreliable overlap suppression here in Figure 4-15.

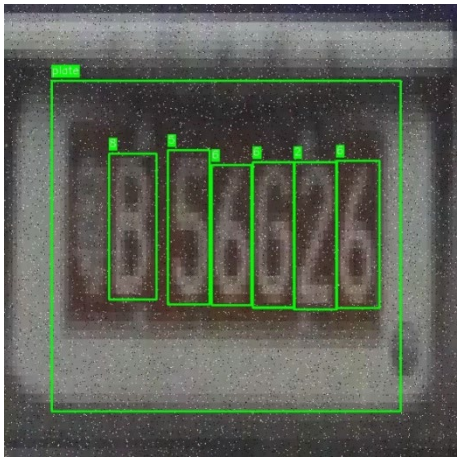

Figure 4-15 Unreliable overlap suppression

An example of this working is shown in Figure 4-16.



Figure 4-16 Potential character overlaps on license plate image

- SD0G63GXWM before NMS
- S063GXW after NMS

Even though the prediction is still incorrect after NMS (due to the similarity the character D and the characters 0 and O have in certain fonts), it is still closer to the correct value.

Given that the accuracy of the model on a completely random dataset with license plates from multiple countries has risen to almost 50%, we can use the lessons learnt to apply this to a local dataset.

In conclusion, incorporating a diverse range of license plates from different regions introduces numerous challenges that the model must address. One significant issue is the ambiguity of certain characters across different fonts. For example, the letter "D" in one font may closely resemble the number "0" in another. Similarly, distinguishing between the letter "O" and the number "0" can be challenging, even for humans. Additionally, license plates from various regions often include extraneous text, such as the name of the region, which can interfere with the model's accuracy. For instance, training a model to recognize Kansas license plates requires it to disregard the word "Kansas" printed on the license plate. Variations in colour and other license plate-specific features further add to the complexity, potentially confusing the model during recognition.

In Figure 4-17 and Figure 4-18, we can see a license plate with a lot of extraneous text that you would not find on a South African license plate. The model gets confused, since it is difficult to determine which part is license plate and which part is not.



Figure 4-17 License plate with extraneous text



Figure 4-18 Confused model, incorrect license plate prediction

4.2 South African results

To achieve high accuracy, the model requires a substantial number of license plates from each region in both the training and validation datasets. Considering that the United States alone has over 50 different license plate designs, and factoring in license plates from other countries, the dataset size quickly becomes immense.

To prepare for this, a number of dummy license plates were generated using the different provincial backgrounds and the *FE Schrift* font as seen in Figure 4-19 and Figure 4-20. The test consisted of 200 images split between testing, training and validation.



Figure 4-19 Artificially generated dummy license plate

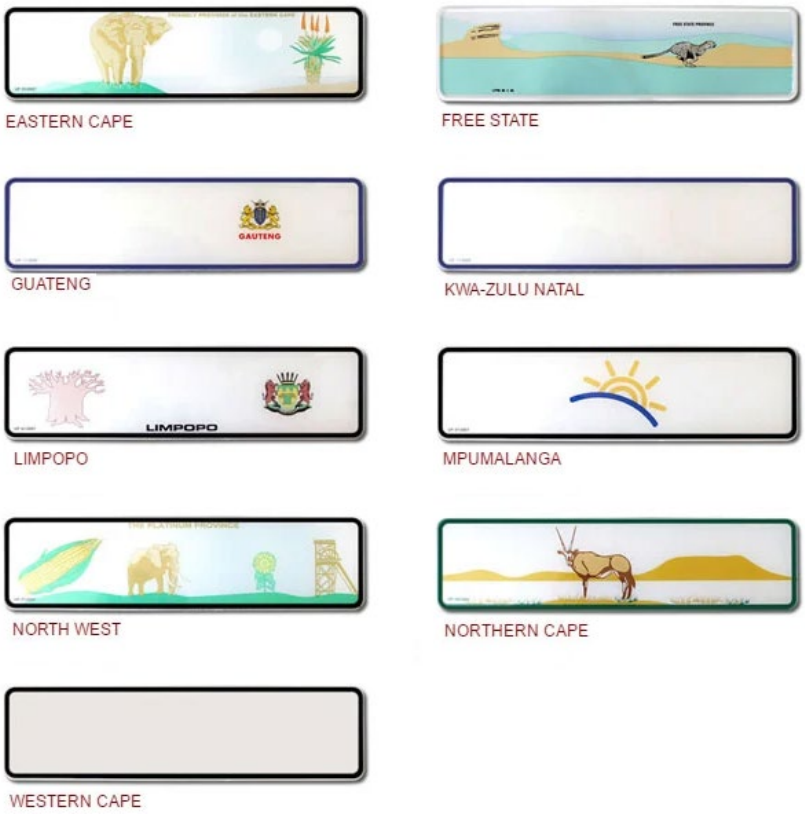

Figure 4-20 South African license plate backgrounds for all nine provinces

The resulting model for this reached an accuracy of 100% on all devices, as seen on Figure 4-21.



Figure 4-21 Accurate license plate prediction

These dummy license plates are not a perfect representation of real-world license plates, however, since they have uniform backgrounds, the license plate is in the exact same position across each image, there are no variations in license plate size and there are also no shadows to contend with. It does, however, illustrate that the model is viable for these kinds of license plates, especially given the high accuracy with the small training set.

For the next set of tests, dummy license plates were created with various backgrounds added, as along with several augmentations like added noise, skewing and moving the license plate, as shown on Figure 4-22, where there is noise added and the license plate itself is skewed, with noticeable black borders where the image was pulled away from the background.



Figure 4-22 Skew augmented artificial dummy license plate

However, the white background on the image causes issues when trying to push non-generated images through the model. For example, Figure 4-23 shows a standard vehicle with a license plate clearly visible (censored to ensure privacy), but the model detected characters in strange places.



Figure 4-23 Standard Image with incorrect predictions

The model can be further refined by using more real-life license plates, and by modifying the dummy license plates to have a variety of random backgrounds (instead of the white background the model seems to require, based on the incorrect prediction in the previous Figure).

There are a few ways to achieve this. The simplest approach is to simply add the license plate to another image, as shown in Figure 4-24 where a blank license plate was added to a background depicting a motor vehicle.



Figure 4-24 Dummy Image with random background

More randomness can be created by adding the license plate image to a random background. The background does not even have to contain a vehicle, since we are ultimately not training the model to look for vehicles; we are training it to look for license plates and read the text on them. This is shown here in Figure 4-25, where a generated license plate is shown on a nature-themed background. In Figure 4-26, the license plate is added in a different part of the image, to simulate how vehicles can enter an image in different positions, depending on where the camera is located. It is also skewed and grayscaled, with another random background image added.



Figure 4-25 License plate with random background

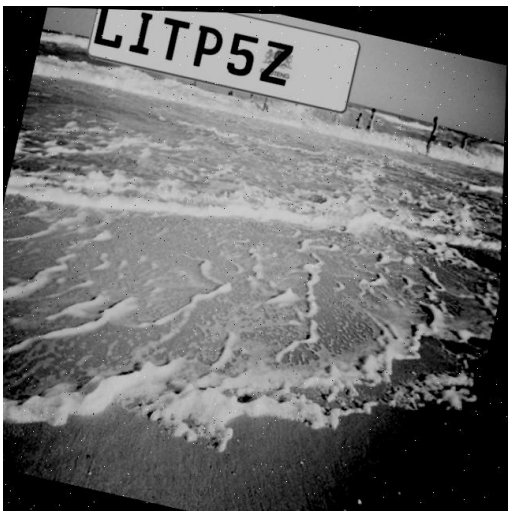

Figure 4-26 Augmented license plate with random background

These generated images, combined with real-life license plates can then be used to train the model for any license plate position and any text that might occur on the license plates (e.g. using the same license plate multiple times teaches the model to only look for said license plate).

Using this model, experiments were performed on all three systems with the newly trained model based on the YOLOv8s (small) variant using 200 images, composed of a combination of artificially generated and real-world license plates. For the first test, only artificially generated license plates were used to establish a baseline, and perfect results were obtained, as shown in Table 4-7.

Table 4-7 Results of tests run with YOLOv8s on augmented images

Hardware	Levenshtein	LCS	Hamming	D-Levenshtein	Jaro-Winkler	Time (ms)
High-level	100	100	100	100	100	196.29
Intermediate level	100	100	100	100	100	283.31
Low-level	100	100	100	100	100	3984.8

The next set of tests were conducted on the combined test set (both generated and real-world license plates) using YOLO versions 8 to 10 using 200 test images, with Table 4-8 showing the low-level hardware configuration results, Table 4-9 the intermediate-level hardware configuration results and Table 4-10 showing the high-level hardware configuration results. It was interesting

to note that YOLOv8s and YOLOv9s gave identical results across the three hardware systems for this test. Figure 4-27 compares the processing time for the small YOLO models across the three hardware systems.

Table 4-8 Combined results for the low-level hardware configuration

YOLO version	Levenshtein	LCS	Hamming	D-Levenshtein	Jaro-Winkler	Time (ms)
YOLOv8s	98.75	97.5	98.75	98.75	99.25	3727.70
YOLOv9s	98.75	97.5	98.75	98.75	99.25	5446.90
YOLOv10s	97.41	92.86	98.75	97.41	98.95	3727.70

Table 4-9 Combined results for the intermediate-level hardware configuration

YOLO version	Levenshtein	LCS	Hamming	D-Levenshtein	Jaro-Winkler	Time (ms)
YOLOv8s	98.75	97.5	98.75	98.75	99.25	299.30
YOLOv9s	98.75	97.5	98.75	98.75	99.25	602.81
YOLOv10s	97.41	92.86	98.75	97.41	98.95	354.97

Table 4-10 Combined results for the high-level hardware configuration

YOLO version	Levenshtein	LCS	Hamming	D-Levenshtein	Jaro-Winkler	Time (ms)
YOLOv8s	98.75	97.5	98.75	98.75	99.25	265.01
YOLOv9s	98.75	97.5	98.75	98.75	99.25	484.71
YOLOv10s	97.41	92.86	98.75	97.41	98.95	287.50

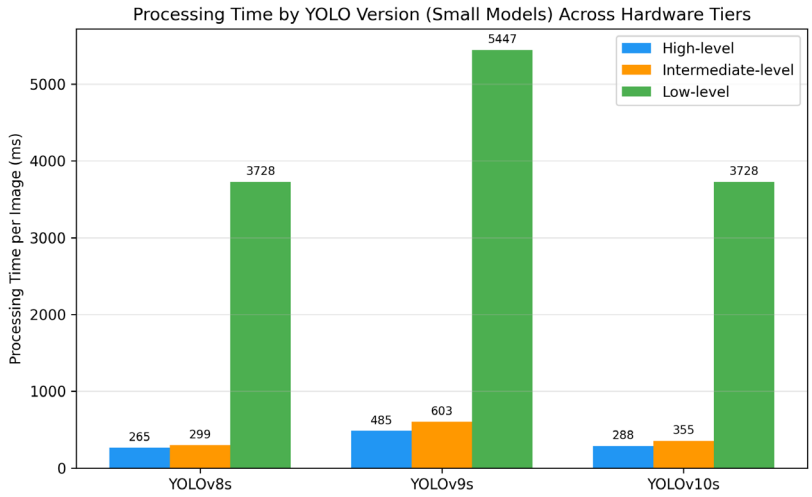

Figure 4-27 Processing time by YOLO version (small models)

It is worth examining why accuracy dropped from a perfect 100% on the artificially generated plates (see Table 4-7) to 98.75% on the combined dataset (see Tables 4-8 to 4-10, for YOLOv8s and YOLOv9s). The main reason is the difference between the two datasets. As noted earlier, the artificial plates are idealised; they use a single, consistent font (*FE-Schrift*), a uniform background, a fixed plate position and size, and no shadows. The combined dataset adds real-world plates, which reintroduce the variation the artificial set deliberately excludes, such as shadows, uneven lighting, differing camera angles and plate positions, and a wider range of fonts and backgrounds. The model therefore had to cope with conditions it did not face during the perfect baseline run.

The type of error is also informative. Across the combined results, the Hamming and Levenshtein scores are identical (both 98.75 for YOLOv8s and YOLOv9s), while the LCS score is slightly lower (97.5). Because Hamming only counts substitutions and requires the two strings to be the same length (see Section 2.7), this pattern indicates that the remaining errors were character substitutions on plates that were detected and segmented into the correct number of characters, rather than missing or additional characters caused by detection failures. In other words, the plates were found and read to the correct length, but the occasional individual character was misread. This is consistent with the character-ambiguity problem discussed earlier, where visually similar characters such as "D", "0" and "O" are easily confused in real-world fonts (see Figure 4-16), a problem the dummy plates avoid by using a single uniform font. YOLOv10s shows the same pattern more strongly, with a noticeably lower LCS score (92.86), which fits the earlier observation that the newer version did not outperform YOLOv8s or YOLOv9s on this task.

This final test was conducted using the best possible models trained with as many South African images as possible, with 3500 training images and 360 validation images. The prediction results

on all three systems were identical. This is likely due to essentially only using the model and not relying on many different libraries that can differ between operating systems (the Raspberry Pi runs on Linux).

For the mid-range and high-level systems, the results are very good with high accuracy and a relatively low processing time. The low-level system has the same accuracy, but it takes much longer to process the images, making it less likely to be a viable solution. To mitigate the shortcomings of the small model, we can try the nano model (or tiny model in the case of YOLOv9) to process the same dataset. We can see Table 4-11 showing the low-level hardware configuration results, Table 4-12 the v9 intermediate-level hardware configuration results and Table 4-13 the high-level hardware configuration results.

Table 4-11 Combined results for the smallest models on low-level hardware configuration

YOLO version	Levenshtein	LCS	Hamming	D-Levenshtein	Jaro-Winkler	Time (ms)
YOLOv8n	94.32	89.83	98.75	94.32	97.45	1943.10
YOLOv9t	95.32	92.26	98.75	95.32	98.35	3648.66
YOLOv10n	94.34	89.47	100	94.34	97.55	2003.56

Table 4-12 Combined results for the smallest models on intermediate-level hardware configuration

YOLO version	Levenshtein	LCS	Hamming	D-Levenshtein	Jaro-Winkler	Time (ms)
YOLOv8n	94.32	89.83	98.75	94.32	97.45	237.95
YOLOv9t	95.32	92.26	98.75	95.32	98.35	526.02
YOLOv10n	94.34	89.47	100	94.34	97.55	361.70

Table 4-13 Combined results for the smallest models on high-level hardware configuration

YOLO version	Levenshtein	LCS	Hamming	D-Levenshtein	Jaro-Winkler	Time (ms)
YOLOv8n	94.32	89.83	98.75	94.32	97.45	196.08
YOLOv9t	95.32	92.26	98.75	95.32	98.35	434.79
YOLOv10n	94.34	89.47	100	94.34	97.55	300.03

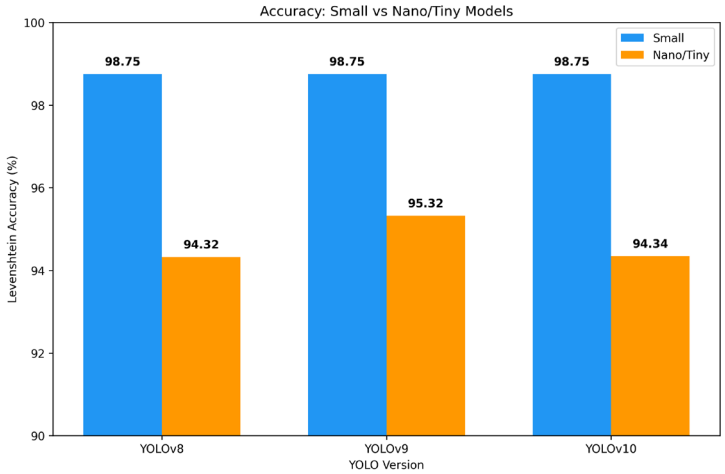

Figure 4-28 Accuracy comparison of small vs nano/tiny YOLO models

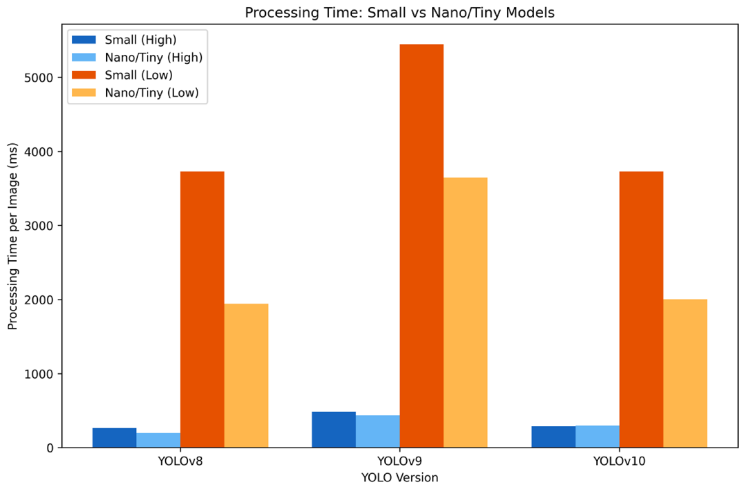

Figure 4-29 Processing time comparison of small vs nano/tiny YOLO models

Figure 4-28 and Figure 4-29 show clearly the difference in accuracy and processing time between the small and tiny/nano YOLO models.

4.3 Summary

This chapter highlights the results informing the secondary questions two and three using the metrics identified in Chapter 2 and the methodology defined in Chapter 3, by comparing different combinations of ALPR software across three hardware configurations. The experiments in Sections 4.1 and 4.2 showed that YOLO significantly outperformed OpenALPR and OpenCV with Tesseract approaches across all hardware tiers. The small YOLO variants offered the best balance between accuracy and speed on intermediate-and high-level hardware, while the tiny or nano variants (depending on YOLO version) were required for the Raspberry Pi. The results are analysed further in Chapter 5.

Chapter 5: Conclusions

5.1 Summary of findings

This dissertation developed and applied a reproducible methodology for comparing software-based ALPR approaches across different hardware platforms. Using this methodology, legacy algorithmic methods such as OpenALPR were compared with modern machine learning models across three hardware configurations. The findings clearly demonstrate that ML-based methods, specifically YOLO, far surpass traditional approaches in both accuracy and efficiency, while legacy methods rely on time-intensive preprocessing pipelines that still produce inferior results.

One key insight from this study is the significant advantage of training models for specific regions, rather than attempting a global approach. Regional training ensures that the model focuses on distinct font and license plate patterns without the confusion introduced by overlapping or inconsistent styles across different regions.

By using YOLOv8's small version, which balances performance and computational efficiency, it is possible to achieve an impressive accuracy of up to 98% for a given region, making it a practical solution for real-world deployment. This model can be trained with as little as 1000 real-world images, if image augmentation is used and generated images are used to pad out characters that do not often occur.

When using intermediate-level or high-level hardware, the small YOLO models are recommended. The tiny/nano models are less accurate and not significantly faster than the small models. For lower-end systems, an ideal solution would be to get a device with a GPU like a Jetson Nano. However, a Raspberry Pi 4B becomes viable if the tiny model is used, although the accuracy is somewhat lower and it still takes 2 seconds per frame to process, so the incoming framerate will need to be limited.

The recommendation for each system is as follows:

1. Low-level: use the tiny or nano version of YOLOv8 or YOLOv10. Avoid YOLOv9, since it is consistently slower.
2. Intermediate level: use the small version of YOLOv8+. Even YOLOv9 is viable, because even though it is slightly slower it is also slightly more accurate.
3. High-level: the small version of YOLOv8+ is sufficient. Higher accuracy might be obtained with larger models, but the small version is fast and accurate enough for use.

Additionally, if a low-level system is to be used, it is recommended that one with some sort of GPU be used instead of a Raspberry Pi 4B. The Raspberry Pi 4B is suitable for general tasks, but the lack of GPU processing does slow it down significantly. One recommendation would be Nvidia's Jetson Nano, which is in a similar price range but is specialised for this kind of processing.

5.2 Research questions

In Section 1.5 of Chapter 1, a primary research question and three secondary research questions were identified. The primary research question asked: **How can one compare the performance of software for license plate recognition on different hardware platforms?**

The secondary research questions asked:

1. **What metrics can be used to compare the accuracy and speed of license plate recognition implementations?**
2. **How do different software-based approaches to license plate recognition compare in recognition accuracy?**
3. **How do different hardware configurations effect the processing speed of license plate recognition software?**

This section provides the answers to these research questions.

5.2.1 Primary research question

The performance of ALPR software across different hardware platforms can be compared using the methodology developed in this study. By combining multiple edit distance metrics to measure accuracy and per-image timing to measure speed, the methodology provides a consistent framework for evaluating any combination of ALPR software and hardware. This was demonstrated by applying it to three hardware configurations and multiple software approaches, with the results presented in Chapter 4.

5.2.2 Secondary research questions

The secondary research questions can be answered as follows:

1. Multiple edit distance algorithms were evaluated, with Levenshtein, LCS, Hamming, Damerau-Levenshtein and Jaro-Winkler all providing useful measures of accuracy. Kendall Tau was found to be unsuitable for this purpose and was excluded (see Table 4-2). Processing speed was effectively measured using per-image timing averaged across three runs.

2. The YOLO-based approach significantly outperformed both OpenALPR and the YOLO with Tesseract combination across all metrics. The OpenALPR approach achieved a maximum Jaro-Winkler accuracy of 46.13% (see Table 4-5), while the fully YOLO-based approach achieved up to 98.75% (see Table 4-10). The remaining 1.25% error rate is caused by issues such as poor quality images, difficult camera angles, etc. Traditional preprocessing techniques did not meaningfully improve accuracy for the YOLO-based approach.
3. The low-level system was between 10 and 14 times slower than the high-level system, with the intermediate-level system being only marginally slower. Processing times for the small YOLO models ranged from 265ms on the high-level system, to 3728ms on the low-level system (see Table 4-8, Table 4-9 and Table 4-10). The nano/tiny model was approximately 4% less accurate but reduced the processing time on the low-level system to below 2000ms (see Table 4-11, Table 4-12 and Table 4-13). This means that it is reduced from about one frame every four seconds to one every two seconds.

5.3 Limitations

This study focuses on a single region (South Africa), with very specific requirements (clear weather and daytime). There are also a limited number of real-world plates available, so the experiments were supplemented using artificial license plates.

The experiments were also performed only on consumer-level hardware, as intended by the scope of the study. No specialised hardware was tested, even though hardware such as LPR cameras exists specifically to perform LPR. The systems that were used as the intermediate-and high-level systems did not use a clean install of Windows, which might have also skewed some results based on what else was installed on those systems.

5.4 Future work

Several areas for future research have been identified based on the limitations and findings of this study. The experiments in this study focused on images taken during the day in clear weather, however images taken during the night or during inclement weather would likely show different results.

This study can also be performed using license plates from another region or attempt to create a model that works for multiple regions (despite the issues shown in Section 4.1). A universal model would have a much higher value than a purely local model. Future researchers can also apply this methodology to non-consumer hardware such as servers.

The experiments can also be rerun on compiled programs using languages such as C++ or Rust, instead of Python. The overhead of using an interpreter might have increased the processing times for the images, so it would be worthwhile to test how this influences the processing time.

Newer YOLO versions should also be tested. They are constantly being released, and each one has improvements that might completely change the recommendations provided in Section 5.1, however the same methodology can be used as outlined in Chapter 3.

5.5 Contributions

The primary contribution of this study is a reproducible methodology for evaluating and comparing ALPR performance across different hardware platforms, enabling future researchers to benchmark their own configurations and software choices against the results presented here.

Applying this methodology, the study provides an empirical evaluation of ALPR methods across three consumer-level hardware systems using South African license plates, a combination of parameters that was not previously studied. The study demonstrates that modern YOLO models achieve near perfect accuracy when trained on region-specific data, even on less expensive and less powerful hardware, and provides practical guidance for selecting the appropriate model based on available resources. This means that in many situations dedicated ALPR hardware may not be required.

These findings have positive implications for small businesses, municipalities and researchers. By providing a low-cost solution using existing hardware, small businesses can use license plate recognition to validate deliveries and as an additional form of security. For municipalities, it enables vehicle monitoring and access control without significant capital investment.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … Zheng, X. 2016. TensorFlow: A System for Large-Scale Machine Learning. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA. pp. 265–283.

Abdul Rehman, A. & Alharthi, K. 2016. An introduction to research paradigms. *International Journal of Educational Investigations*. 3(8):51–59.

Abiodun, O.I., Jantan, A., Omolara, A.E., Dada, K.V., Mohamed, N.A. & Arshad, H. 2018. State-of-the-art in artificial neural network applications: A survey. *Heliyon*. 4(11):e00938.

Abioye, S.O., Oyedele, L.O., Akanbi, L., Ajayi, A., Davila Delgado, J.M., Bilal, M., … Ahmed, A. 2021. Artificial intelligence in the construction industry: A review of present status, opportunities and future challenges. *Journal of Building Engineering*. 44:103299.

Abu-Ain, W., Abdullah, S.N.H.S., Bataineh, B., Abu-Ain, T. & Omar, K. 2013. Skeletonization Algorithm for Binary Images. *Procedia Technology*. 11:704–709.

Agbemenu, A., Yankey, J. & Addo, E.O. 2018. An Automatic Number Plate Recognition System using OpenCV and Tesseract OCR Engine. *International Journal of Computer Applications*. 180:1–5.

Aghagolzadeh, S. & Ersoy, O.K. 1992. Transform image enhancement. *Optical Engineering*. 31(3):614–626.

Al-Ghaili, A.M., Mashohor, S., Ramli, A.R. & Ismail, A. 2013. Vertical-Edge-Based Car-License-Plate Detection Method. *IEEE Transactions on Vehicular Technology*. 62(1):26–38.

Alharahsheh, H.H. & Pius, A. 2020. A Review of key paradigms: positivism VS interpretivism. *Global Academic Journal of Humanities and Social Sciences*. 2(3):39–43.

AL-Khatatneh, A., Ali Pitchay, S. & Al-qudah, M. 2015. A Review of Skew Detection Techniques for Document. In: *2015 17th UKSim-AMSS International Conference on Modelling and Simulation (UKSim)*. IEEE. pp. 316–321.

Aloysius, N. & Geetha, M. 2017. A review on deep convolutional neural networks. In: *2017 International Conference on Communication and Signal Processing (ICCSP)*. IEEE. pp. 588–592.

Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., … Farhan, L. 2021. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*. 8(1):53.

Anderson, J.A. 1995. *An Introduction to Neural Networks*. Cambridge, MA: MIT Press.

Apostolico, A. & Guerra, C. 1987. The longest common subsequence problem revisited. *Algorithmica*. 2(1–4):315–336.

Arulmozhi, K., Perumal, S.A., Mohan, M.V. & Nallaperumal, K. 2012. Skew detection and correction of Indian vehicle license plate using polar Hough Transform research. In: *2012 IEEE International Conference on Computational Intelligence and Computing Research*. IEEE. pp. 1–4.

Asif, A., Hannan, S.A., Perwej, Y. & Vithalrao, M.A. 2014. An overview and applications of optical character recognition. *International Journal of Engineering Science and Advanced Technology*. 3(7):261–274.

Audi, R. 2010. *Epistemology: A Contemporary Introduction to the Theory of Knowledge*. 3rd edn. New York: Routledge.

Babu, M.V.S. & Banana, K. 2024. A study on narrow artificial intelligence-An overview. *International Journal of Engineering Science and Advanced Technology*. 24(4):210–219.

Badla, S. 2014. Improving the efficiency of Tesseract OCR engine. San Jose, CA, USA: San Jose State University. (Master's Project).

Bergroth, L., Hakonen, H. & Raita, T. 2000. A survey of longest common subsequence algorithms. In: *Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000*. IEEE. pp. 39–48.

Bloice, M.D., Stocker, C. & Holzinger, A. 2017. *Augmentor: An Image Augmentation Library for Machine Learning*. (arXiv:1708.04680). arXiv.

Bochkovskiy, A., Wang, C.-Y. & Liao, H.-Y.M. 2020. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. (arXiv:2004.10934). arXiv.

Bookstein, A., Kulyukin, V.A. & Raita, T. 2002. Generalized Hamming Distance. *Information Retrieval*. 5(4):353–375.

Brossart, D.F., Laird, V.C. & Armstrong, T.W. 2018. Interpreting Kendall's Tau and Tau-U for single-case experimental designs. *Cogent Psychology*. 5(1).

Buslaev, A., Iglovikov, V.I., Khvedchenya, E., Parinov, A., Druzhinin, M. & Kalinin, A.A. 2020. Albumentations: Fast and Flexible Image Augmentations. *Information*. 11(2):125.

Chaki, J. & Dey, N. 2018. *A Beginner's Guide to Image Preprocessing Techniques*. Boca Raton, FL: CRC Press.

Chaudhuri, A., Mandaviya, K., Badelia, P. & Ghosh, S. 2017. Optical Character Recognition Systems. In: *Optical character recognition systems for different languages with soft computing*. Springer. pp. 9–41.

Chauhan, R., Ghanshala, K.K. & Joshi, R.C. 2018. Convolutional Neural Network (CNN) for Image Detection and Recognition. In: *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*. IEEE. pp. 278–282.

Cheatham, M., Cruz, I.F., Euzenat, J., Pesquita, C., Dreßler, K. & Ngonga Ngomo, A.-C. 2017. On the efficient execution of bounded Jaro-Winkler distances. *Semantic Web*. 8(2):185–196.

Cicirello, V.A. 2019. *Kendall Tau Sequence Distance: Extending Kendall Tau from Ranks to Sequences*. (arXiv:1905.02752). arXiv.

Cicirello, V.A. 2020. Kendall Tau Sequence Distance: Extending Kendall Tau from Ranks to Sequences. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*. 7(23):163925.

CIE. 2022a. *Lightness*. *e-ILV: International Lighting Vocabulary*. http://cie.co.at/eilv/680 Date of access: 2 July 2022.

CIE. 2022b. *Chroma*. *e-ILV: International Lighting Vocabulary*. http://cie.co.at/eilvterm/17-22-074 Date of access: 2 July 2022.

CIE. 2022c. *Hue*. *e-ILV: International Lighting Vocabulary*. https://cie.co.at/eilvterm/17-22-067 Date of access: 2 July 2022.

Culjak, I., Abram, D., Pribanic, T., Dzapo, H. & Cifrek, M. 2012. A brief introduction to OpenCV. In: *2012 Proceedings of the 35th International Convention MIPRO*. IEEE. pp. 1725–1730.

Devetak, R. 2013. Critical theory. In: *Theories of international relations*. Basingstoke: Palgrave Macmillan. pp. 162–186.

Du, S., Ibrahim, M., Shehata, M. & Badawy, W. 2013. Automatic License Plate Recognition (ALPR): A State-of-the-Art Review. *IEEE Transactions on Circuits and Systems for Video Technology*. 23(2):311–325.

Erdinc Kocer, H. & Kursat Cevik, K. 2011. Artificial neural networks based vehicle license plate recognition. *Procedia Computer Science*. 3:1033–1037.

Eyupoglu, C. 2017. Implementation of Bernsen's locally adaptive binarization method for gray scale images. *The Online Journal of Science and Technology*. 7(2):68–72.

Friendly, F. 2019. Jaro–Winkler Distance Improvement For Approximate String Search Using Indexing Data For Multiuser Application. *Journal of Physics: Conference Series*. 1361(1):012080.

Fung, C.C. & Chamchong, R. 2010. A Review of Evaluation of Optimal Binarization Technique for Character Segmentation in Historical Manuscripts. In: *2010 Third International Conference on Knowledge Discovery and Data Mining*. IEEE. pp. 236–240.

Garain, U., Jain, A., Maity, A. & Chanda, B. 2008. Machine reading of camera-held low quality text images: An ICA-based image enhancement approach for improving OCR accuracy. In: *2008 19th International Conference on Pattern Recognition*. IEEE. pp. 1–4.

Gholamalinezhad, H. & Khosravi, H. 2020. *Pooling Methods in Deep Neural Networks, a Review*. (arXiv:2009.07485). arXiv.

Gilbert, N. & Rusli, A. 2020. Single object detection to support requirements modeling using faster R-CNN. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*. 18(2):830–838.

Goel, A., Tung, C., Lu, Y.-H. & Thiruvathukal, G.K. 2020. A Survey of Methods for Low-Power Deep Learning and Computer Vision. In: *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. IEEE. pp. 1–6.

Goldsborough, P. 2016. *A Tour of TensorFlow*. (arXiv:1610.01178). arXiv.

Gonçalves, G.R., da Silva, S.P.G., Menotti, D. & Schwartz, W.R. 2016. Benchmark for License Plate Character Segmentation. *Journal of Electronic Imaging*. 25(5):053034.

Grzybowski, A., Pawlikowska–Łagód, K. & Lambert, W.C. 2024. A History of Artificial Intelligence. *Clinics in Dermatology*. 42(3):221–229.

Guo, J.-M. & Liu, Y.-F. 2008. License Plate Localization and Character Segmentation With Feedback Self-Learning and Hybrid Binarization Techniques. *IEEE Transactions on Vehicular Technology*. 57:1417–1424.

Guo, Y., Zhu, Y., Liu, L. & Huang, Q. 2022. Research Review of Space-Frequency Domain Image Enhancement Methods. *Journal of Computer Engineering & Applications*. 58(11).

Haldar, R. & Mukhopadhyay, D. 2011. *Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach*. (arXiv:1101.1232). arXiv.

Hamming, R.W. 1950. Error detecting and error correcting codes. *The Bell System Technical Journal*. 29(2):147–160.

Hanbury, A. 2008. A survey of methods for image annotation. *Journal of Visual Languages & Computing*. 19(5):617–627.

Hollingworth, G. 2022. Raspberry Pi OS (64-bit). *Raspberry Pi*. https://www.raspberrypi.com/news/raspberry-pi-os-64-bit/ Date of access: 18 Nov. 2024.

Hossain, Md.Y., Nijhum, I.R., Sadi, A.A., Shad, Md.T.M. & Rahman, R.M. 2021. Visual Pollution Detection Using Google Street View and YOLO. In: *2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE. pp. 433–440.

Hu, L., Chen, J., Vaughan, J., Aramideh, S., Yang, H., Wang, K., … Nair, V.N. 2021. Supervised Machine Learning Techniques: An Overview with Applications to Banking. *International Statistical Review*. 89(3):573–604.

Huang, M., Yu, W. & Zhu, D. 2012. An Improved Image Segmentation Algorithm Based on the Otsu Method. In: *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. IEEE. pp. 135–139.

Huang, R., Pedoeem, J. & Chen, C. 2018. YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers. In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE. pp. 2503–2510.

Ibrahim, H. & Pik Kong, N.S. 2009. Image sharpening using sub-regions histogram equalization. *IEEE Transactions on Consumer Electronics*. 55(2):891–895.

Imambi, S., Prakash, K.B. & Kanagachidambaresan, G.R. 2021. PyTorch. In: K.B. Prakash & G.R. Kanagachidambaresan, eds. *Programming with TensorFlow: Solution for Edge Computing Applications*. Cham: Springer International Publishing. pp. 87–104.

Jagannathan, J., Sherajdheen, A., Vijay Deepak, R.M. & Krishnan, N. 2013. License plate Character Segmentation using horizontal and vertical projection with dynamic thresholding. In: *2013 IEEE International Conference on Emerging Trends in Computing, Communication and Nanotechnology (ICECCN)*. IEEE. pp. 700–705.

Jaro, M.A. 1989. Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. *Journal of the American Statistical Association*. 84(406):414–420.

Jeong, H.-J., Park, K.-S. & Ha, Y.-G. 2018. Image Preprocessing for Efficient Training of YOLO Deep Learning Networks. In: *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE. pp. 635–637.

Jiang, P., Ergu, D., Liu, F., Cai, Y. & Ma, B. 2022. A Review of Yolo Algorithm Developments. *Procedia Computer Science*. 199:1066–1073.

Jordan, M.I. & Mitchell, T.M. 2015. Machine learning: Trends, perspectives, and prospects. *Science*. 349(6245):255–260.

Kanan, C. & Cottrell, G.W. 2012. Color-to-Grayscale: Does the Method Matter in Image Recognition? *PLOS ONE*. 7(1):e29740.

Kaur, K. & Sharma, M. 2013. A Method for Binary Image Thinning using Gradient and Watershed Algorithm. *International Journal of Advanced Research in Computer Science and Software Engineering*. 3(1).

Kekre, H.B. & Thepade, S.D. 2008. Color Traits Transfer to Grayscale Images. In: *2008 First International Conference on Emerging Trends in Engineering and Technology*. IEEE. pp. 82–85.

Kivunja, C. & Kuyini, A.B. 2017. Understanding and Applying Research Paradigms in Educational Contexts. *International Journal of Higher Education*. 6(5):26–41.

Krauss, P. 2024. What is Artificial Intelligence? In: P. Krauss, ed. *Artificial Intelligence and Brain Research: Neural Networks, Deep Learning and the Future of Cognition*. Berlin, Heidelberg: Springer. pp. 107–112.

Krenker, A., Bešter, J. & Kos, A. 2011. Introduction to the artificial neural networks. In: *Artificial Neural Networks: Methodological Advances and Biomedical Applications*. Rijeka: InTech. pp. 1–18.

Laroca, R., Severo, E., Zanlorensi, L.A., Oliveira, L.S., Gonçalves, G.R., Schwartz, W.R. & Menotti, D. 2018. A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. pp. 1–10.

Legg, C. & Hookway, C. 2008. Pragmatism. In: *The Stanford Encyclopedia of Philosophy*. https://plato.sydney.edu.au/entries/pragmatism/ Date of access: 13 Apr. 2025.

Li, Y. 2018. *Deep Reinforcement Learning: An Overview*. (arXiv:1701.07274). arXiv.

Li, C., Li, L., Geng, Y., Jiang, H., Cheng, M., Zhang, B., … Chu, X. 2023. *YOLOv6 v3.0: A Full-Scale Reloading*. (arXiv:2301.05586). arXiv.

Li, G., Müller, M., Ghanem, B. & Koltun, V. 2021. Training Graph Neural Networks with 1000 Layers. In: *Proceedings of the 38th International Conference on Machine Learning*. PMLR. pp. 6437–6449. https://proceedings.mlr.press/v139/li21o.html Date of access: 26 Mar. 2024.

Li, H., Rajbahadur, G.K. & Bezemer, C.-P. 2024. Studying the Impact of TensorFlow and PyTorch Bindings on Machine Learning Software Quality. *ACM Transactions on Software Engineering and Methodology*. 34(1):1–31.

Li, Y., Zhao, Z., Luo, Y. & Qiu, Z. 2020. Real-Time Pattern-Recognition of GPR Images with YOLO v3 Implemented by Tensorflow. *Sensors*. 20(22):6476.

Li, Z., Yang, W., Peng, S. & Liu, F. 2020. *A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects*. (arXiv:2004.02806). arXiv.

Luo, C., Jin, L. & Sun, Z. 2019. MORAN: A Multi-Object Rectified Attention Network for Scene Text Recognition. *Pattern Recognition*. 90:109–118.

Macêdo, S., Melo, G. & Kelner, J. 2015. A comparative study of grayscale conversion techniques applied to SIFT descriptors. *Journal on Interactive Systems*. 6(2):1.

Maini, R. & Aggarwal, H. 2010. *A Comprehensive Review of Image Enhancement Techniques*. (arXiv:1003.4053). arXiv.

Mao, Q.-C., Sun, H.-M., Liu, Y.-B. & Jia, R.-S. 2019. Mini-YOLOv3: Real-Time Object Detector for Embedded Applications. *IEEE Access*. 7:133529–133538.

Meng, C., Sun, M., Yang, J., Qiu, M. & Gu, Y. 2017. Training Deeper Models by GPU Memory Optimization on TensorFlow. In: *Proc. of ML Systems Workshop in NIPS*. pp. 26.

Metcalf, L. & Casey, W. 2016. Chapter 2 - Metrics, similarity, and sets. In: L. Metcalf & W. Casey, eds. *Cybersecurity and Applied Mathematics*. Boston: Syngress. pp. 3–22.

Mills, J. 2014. Methodology and methods. In: J. Mills & M. Birks, eds. *Qualitative methodology: a practical guide*. London, United Kingdom: Sage. pp. 31–47.

Milyaev, S., Barinova, O., Novikova, T., Kohli, P. & Lempitsky, V. 2013. Image Binarization for End-to-End Text Understanding in Natural Images. In: *2013 12th International Conference on Document Analysis and Recognition*. IEEE. pp. 128–132.

Mittal, R. & Garg, A. 2020. Text extraction using OCR: A Systematic Review. In: *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*. Coimbatore, India: IEEE. pp. 357–362.

Nayak, M. & Nayak, A.K. 2014. Odia characters recognition by training tesseract OCR engine. *International Journal of Computer Applications*. 25–30.

Nina, O., Morse, B. & Barrett, W. 2011. A recursive Otsu thresholding method for scanned document binarization. In: *2011 IEEE Workshop on Applications of Computer Vision (WACV)*. IEEE. pp. 307–314.

O'Shea, K. & Nash, R. 2015. *An Introduction to Convolutional Neural Networks*. (arXiv:1511.08458). arXiv.

Palekar, R.R., Parab, S.U., Parikh, D.P. & Kamble, V.N. 2017. Real time license plate detection using openCV and tesseract. In: *2017 International Conference on Communication and Signal Processing (ICCSP)*. IEEE. pp. 2111–2115.

Pang, K.F. & El Gamal, A. 1986. Communication Complexity of Computing the Hamming Distance. *SIAM Journal on Computing*. 15(4):932–947.

Pang, B., Nijkamp, E. & Wu, Y.N. 2020. Deep Learning With TensorFlow: A Review. *Journal of Educational and Behavioral Statistics*. 45(2):227–248.

Papamarkou, I. & Papamarkos, N. 2013. Conversion of color documents to grayscale. In: *21st Mediterranean Conference on Control and Automation*. IEEE. pp. 1609–1614.

Park, Y.S., Konge, L. & Artino, A.R. 2020. The Positivism Paradigm of Research. *Academic Medicine*. 95(5):690–694.

Patel, C., Patel, A. & Patel, D. 2012. Optical character recognition by open source OCR tool tesseract: A case study. *International Journal of Computer Applications*. 55(10):50–56.

Paterson, M. & Dančík, V. 1994. Longest common subsequences. In: I. Prívara, B. Rovan, & P. Ruzička, eds. *Mathematical Foundations of Computer Science 1994*. Berlin, Heidelberg: Springer. pp. 127–142.

Peker, M. 2019. Comparison of Tensorflow Object Detection Networks for Licence Plate Localization. In: *2019 1st Global Power, Energy and Communication Conference (GPECOM)*. IEEE. pp. 101–105.

Potrac, P., Jones, R.L. & Nelson, L. 2014. Interpretivism. In: *Research Methods in Sports Coaching*. London: Routledge. pp. 31–41.

Prates, R.F., Cámara-Chávez, G., Schwartz, W.R. & Menotti, D. 2013. Brazilian License Plate Detection Using Histogram of Oriented Gradients and Sliding Windows. *International Journal of Computer Science and Information Technology*. 5(6):39–52.

Pretorius, L. 2024. Demystifying Research Paradigms: Navigating Ontology, Epistemology, and Axiology in Research. *The Qualitative Report*. 29(10):2698–2715.

Pulli, K., Baksheev, A., Kornyakov, K. & Eruhimov, V. 2012. Real-time computer vision with OpenCV. *Communications of the ACM*. 55(6):61–69.

Puneet, P. & Garg, N. 2013. Binarization Techniques used for Grey Scale Images. *International Journal of Computer Applications*. 71:8–11.

Rahman, S., Rahman, Md.M., Hussain, K., Khaled, S.M. & Shoyaib, M. 2014. Image enhancement in spatial domain: A comprehensive study. In: *2014 17th International Conference on Computer and Information Technology (ICCIT)*. IEEE. pp. 368–373.

Razak, Z., Zulkiflee, K., Idris, M.Y.I., Tamil, E.M., Mohamed Noor, M.N., Salleh, R., … Yaacob, M. 2008. Off-line Handwriting Text Line Segmentation : A Review. *International Journal of Computer Science and Network Security*. 8(7):12–20.

Redmon, J. & Farhadi, A. 2016. *YOLO9000: Better, Faster, Stronger*. (arXiv:1612.08242). arXiv.

Redmon, J. & Farhadi, A. 2018. *YOLOv3: An Incremental Improvement*. (arXiv:1804.02767). arXiv.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. 2016. You Only Look Once: Unified, Real-Time Object Detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. pp. 779–788.

Ristad, E.S. & Yianilos, P.N. 1998. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 20(5):522–532.

Robinson, A. & Venter, C. 2019. Validating traffic models using large-scale Automatic Number Plate Recognition (ANPR) data. *Journal of the South African Institution of Civil Engineering.*

Rogers, J. & Révész, A. 2019. Experimental and quasi-experimental designs. In: *The Routledge Handbook of Research Methods in Applied Linguistics*. London: Routledge. pp. 133–143.

Romero, V., Sánchez, J.A., Bosch, V., Depuydt, K. & de Does, J. 2015. Influence of text line segmentation in Handwritten Text Recognition. In: *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE. pp. 536–540.

Saeed, K., Tabędzki, M., Rybnik, M. & Adamski, M. 2010. K3M: A universal algorithm for image skeletonization and a review of thinning techniques. *International Journal of Applied Mathematics and Computer Science*. 20(2):317–335.

Saini, R. 2015. Document image binarization techniques, developments and related issues: a review. *International Journal of Computer Applications*. 116(7):41–44.

Saravanan, C. 2010. Color Image to Grayscale Image Conversion. In: *2010 Second International Conference on Computer Engineering and Applications*. Bali Island, Indonesia: IEEE. pp. 196–199.

Sarikaya, F. 2024. *The Cycles of AI Winters: A Historical Analysis and Modern Perspective*. Zenodo.

Shafiee, M.J., Chywl, B., Li, F. & Wong, A. 2017. *Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video*. (arXiv:1709.05943). arXiv.

Shi, B., Wang, X., Lyu, P., Yao, C. & Bai, X. 2016. Robust Scene Text Recognition with Automatic Rectification. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. pp. 4168–4176.

Shi, B., Yang, M., Wang, X., Lyu, P., Yao, C. & Bai, X. 2019. ASTER: An Attentional Scene Text Recognizer with Flexible Rectification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 41(9):2035–2048.

Shi, Y., Ding, Y., Zhang, R. & Li, J. 2009. Structure and Hue Similarity for Color Image Quality Assessment. In: *2009 International Conference on Electronic Computer Technology*. IEEE. pp. 329–333.

Shorten, C. & Khoshgoftaar, T.M. 2019. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*. 6(1):60.

Singh, G. & Mittal, A. 2014. Various Image Enhancement Techniques - A Critical Review. *International Journal of Innovation and Scientific Research*. 10(2):267–274.

Singh, A., Bacchuwar, K. & Bhasin, A. 2012. A Survey of OCR Applications. *International Journal of Machine Learning and Computing*. 2(3):314–318.

Singh, O.I., Sinam, T., James, O. & Singh, T.R. 2012. Local Contrast and Mean based Thresholding Technique in Image Binarization. *International Journal of Computer Applications*. 51(6):5–10.

Singh, T.R., Roy, S., Singh, O.I., Sinam, T. & Singh, K.M. 2011. A New Local Adaptive Thresholding Technique in Binarization. *IJCSI International Journal of Computer Science Issues*. 8(6):271–277.

Smith, B. 2012. Ontology. In: G. Hurtado & O. Nudler, eds. *The Furniture of the World: Essays in Ontology and Metaphysics*. Amsterdam: Editions Rodopi. pp. 47–68.

Smith, G.T. 2005. On Construct Validity: Issues of Method and Measurement. *Psychological Assessment*. 17(4):396–408.

Smith, R. 2007. An Overview of the Tesseract OCR Engine. In: Vol. 2. *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. IEEE. pp. 629–633.

Smith, R.W. 2013. History of the Tesseract OCR engine: what worked and what didn't. In: R. Zanibbi & B. Coüasnon, eds. *Document Recognition and Retrieval XX*. Burlingame, California, USA: SPIE. pp. 865802.

South Africa. 1996. National Road Traffic Act 93 of 1996. Pretoria: Government Printer.

Srinivas, B.A., Agarwal, A. & Rao, C.R. 2008. An Overview of OCR Research in Indian Scripts. *International Journal of Computer Science and Engineering Systems*. 2(2):141–153.

Stathis, P., Kavallieratou, E. & Papamarkos, N. 2008. An Evaluation Technique for Binarization Algorithms. *Journal of Universal Computer Science*. 14(18):3011–3030.

Sun, Y. & Zhao, X. 2020. Research and implementation of license plate recognition based on android platform. *MATEC Web of Conferences*. 309:03034.

Sun, P., Xie, Q., Wu, Z., Feng, X., Cai, J. & Jiang, Y. 2019. Yi Characters Recognition Based on Tesseract-OCR. In: *2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. IEEE. pp. 102–106.

Szeliski, R. 2022. *Computer Vision: Algorithms and Applications*. Springer Nature.

Tian, J., Wang, R., Wang, G., Liu, J. & Xia, Y. 2015. A two-stage character segmentation method for Chinese license plate. *Computers & Electrical Engineering*. 46:539–553.

Toosi, A., Bottino, A.G., Saboury, B., Siegel, E. & Rahmim, A. 2021. A brief history of AI: how to prevent another winter (a critical review). *PET clinics*. 16(4):449–469.

Ukkonen, E. 1983. On approximate string matching. In: M. Karpinski, ed. *Foundations of Computation Theory*. Berlin, Heidelberg: Springer. pp. 487–495.

van Heerden, Renier. Automatic Number Plate Segmentation and Recognition. November 1996.

Wan Ismail, W.Z. 2021. An Illegal Dumping Detection System Based on Image Processing in OpenALPR. *ASM Science Journal*. 15:1–7.

Wang, S.-C. 2003. Artificial Neural Network. In: S.-C. Wang, ed. *Interdisciplinary Computing in Java Programming*. Boston, MA: Springer US. pp. 81–100.

Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J. & Ding, G. 2024. YOLOv10: Real-Time End-to-End Object Detection. *Advances in Neural Information Processing Systems*. 37:107984–108011.

Wang, C.-Y., Bochkovskiy, A. & Liao, H.-Y.M. 2023. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE. pp. 7464–7475.

Wang, C.-Y., Yeh, I.-H. & Mark Liao, H.-Y. 2025. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. In: A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, & G. Varol, eds. *Computer Vision – ECCV 2024*. Cham: Springer Nature Switzerland. pp. 1–21.

Wang, J., Li, Q., Fang, Z., Zhou, X., Tang, Z., Han, Y. & Ma, Z. 2023. YOLOv6-ESG: A Lightweight Seafood Detection Method. *Journal of Marine Science and Engineering*. 11(8):1623.

Wang, J.-X., Zhou, W.-Z., Xue, J.-F. & Liu, X.-X. 2010. The research and realization of vehicle license plate character segmentation and recognition technology. In: *2010 International Conference on Wavelet Analysis and Pattern Recognition*. IEEE. pp. 101–104.

White, N. 2012. Training Tesseract for Ancient Greek OCR. *Eutypon*. (28–29).

Wu, J. 2017. *Introduction to convolutional neural networks*. Nanjing University.

Wu, Y. & Feng, J. 2018. Development and Application of Artificial Neural Network. *Wireless Personal Communications*. 102(2):1645–1656.

Wu, Y., Rawls, S., AbdAlmageed, W. & Natarajan, P. 2015. *Learning Document Image Binarization from Data*. (arXiv:1505.00529). arXiv.

Xu, M., Yoon, S., Fuentes, A. & Park, D.S. 2023. A Comprehensive Survey of Image Augmentation Techniques for Deep Learning. *Pattern Recognition*. 137:109347.

Yamashita, R., Nishio, M., Do, R.K.G. & Togashi, K. 2018. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*. 9(4):611–629.

Yang, G., Coble, D., Vaughan, C., Peele, C., Morsali, A., List, G.F. & Findley, D.J. 2022. Waiting Time Estimation at Ferry Terminals Based on License Plate Recognition. *Journal of Transportation Engineering, Part A: Systems*. 148(9):04022064.

Ying, X. 2019. An Overview of Overfitting and its Solutions. *Journal of Physics: Conference Series*. 1168:022022.

Yoon, Y. 2013. Best Combination of Binarization Methods for License Plate Character Segmentation. *ETRI Journal*. 35(3):491–500.

Yujian, L. & Bo, L. 2007. A Normalized Levenshtein Distance Metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 29(6):1091–1095.

Zafar, A., Aamir, M., Mohd Nawi, N., Arshad, A., Riaz, S., Alruban, A., … Almotairi, S. 2022. A Comparison of Pooling Methods for Convolutional Neural Networks. *Applied Sciences*. 12(17):8643.

Zhang, D., Islam, Md.M. & Lu, G. 2012. A review on automatic image annotation techniques. *Pattern Recognition*. 45(1):346–362.

Zhang, X., Xu, F. & Su, Y. 2011. Research on the License Plate Recognition based on MATLAB. *Procedia Engineering*. 15:1330–1334.

Zhao, C. & Sahni, S. 2019. String correction using the Damerau-Levenshtein distance. *BMC Bioinformatics*. 20(11):277.

Zhao, L. & Li, S. 2020. Object Detection Algorithm Based on Improved YOLOv3. *Electronics*. 9(3):537.