

Energy-efficient automatic license plate recognition for edge devices using neural architecture search

Ferenc Fodor
2024.



Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

Energy-efficient automatic license plate recognition for edge devices using neural architecture search

Scientific Students' Association Report

Author:

Ferenc Fodor

Advisor:

Gergő Bendegúz Békési,

Siavash Arjomand Bigdeli,

2024

Contents

Abstract	i
Absztrakt	ii
1 Introduction	1
2 Literature Review	3
2.1 ALPR with Classical Methods	3
2.2 ALPR with Neural Networks	4
2.3 Neural Architecture Search	6
2.3.1 ALPR and NAS	7
3 Methodology	8
3.1 Problem Background and Motivation	8
3.2 Objective and Scope of the Study	9
3.3 Proposed Solution Approach	9
3.4 Benchmarking versus the state-of-the-art	10
3.4.1 Ax: An Adaptive Experimentation Platform	11
4 Experimental Setup	13
4.1 Dataset and Preprocessing	13
4.2 CNN Block Selection	14
4.3 Target Hardware and Latency Table Creation	15
4.4 NAS Pipeline Configuration	16
4.5 Training and Evaluation Setup	16
4.6 Baseline Models for Comparison	17
5 Results	18
5.1 The Ax Experiments	18
5.1.1 Breakdown of Experiment 13	19
5.1.2 Candidate model selection from NAS experiments	20

5.2	Full Training	22
5.2.1	Learning Rate Experiments with the Best Architecture	23
5.2.2	Summary of trained models	23
5.3	Benchmarking	24
5.4	Quantization	25
5.5	Comparisons with LPLite	27
6	Discussions	29
6.1	Significance of this Study	29
6.2	Future work	30
6.2.1	Estimated Latency versus actual Inference Time	30
6.2.2	Exploring other datasets and transfer learning	30
6.2.3	Character recognition	31
	Bibliography	32

Abstract

From AlphaFold to AI assistants, to self-driving cars, artificial intelligence is quickly revolutionizing our world. Yet, these examples are just a few of the real-world applications coming from the research in the field of artificial neural networks (ANN), and every year novel ideas arise, promising an even better tomorrow. Automatic License Plate Recognition (ALPR) is another real-life application of the field, where real-time detection of the license plates has a vast number of use-cases, from automated tolls to parking lots, to contributing to public safety on high-speed roads. However, security cameras do not enjoy the benefits of high-end GPUs for the computational needs of complex models, and as such, finding a reliable solution that can be run on already-existing, low-performance technology is highly sought after.

Neural Architecture Search (NAS) is a technique that automates the design process of neural networks using a search space of architectures, a search strategy that explores the search space, and a performance estimation strategy – based on which the best ANN is selected. NAS-based solutions have shown to be on par or even outperform human-designed models, resulting in simpler and faster architectures, thus making it a promising approach for ALPR.

ALPR can be seen as a three-step process: (1) finding the license plate – an object detection task; (2) character separation – segmentation task; (3) character identification – classification task. It is common within ALPR research that the steps are studied separately, as they represent different types of challenges when it comes to the required neural network type. In this work, I am proposing a NAS-based object detection model for step (1) that improves upon the already-existing solutions found in literature, inspired by the Differentiable Architecture Search approach. To ensure real-time performance in low-power scenarios, the proposed architecture is benchmarked against state-of-the-art You Only Look Once models not only on a GPU, but also on a Raspberry Pi 4. To complete the additional steps of ALPR, the detected license plates are passed through an optical character recognition algorithm to detect the characters. The benchmarking of the full loop is also going to be discussed.

Absztrakt

Az AlphaFoldtól az AI asszisztensekig, az önvezető autókig az intelligens rendszerek gyorsan forradalmasítják világunkat. Ezek a példák azonban csak néhányat képviselnek a mesterséges neurális hálózatok (artificial neural network, ANN) kutatásából származó valós alkalmazások közül, és minden évben új ötletek születnek, amelyek még jobb jövőt ígérnek. Az automatikus rendszámfelismerés (automatic license plate recognition, ALPR) egy másik valós alkalmazása ennek a területnek, ahol a rendszámok valós idejű felismerése számos felhasználási területet érint, az automatizált útdíj-rendszerektől a parkolóig, valamint a közbiztonság növeléséig a gyorsforgalmi utakon. Azonban a biztonsági kamerák nem élvezik a bonyolult modellek számítási igényeihez szükséges csúcskategóriás GPU-k előnyeit, ezért megbízható, már meglévő, alacsony energiafogyasztású technológiákon futtatható megoldások megtalálása igen keresett.

A neurális architektúrák keresése (neural architecture search, NAS) egy olyan technika, amely automatizálja a neurális hálózatok tervezési folyamatát egy architektúra-keresési tér, egy keresési stratégia – amely felfedezi a keresési teret –, és egy teljesítménybecslési stratégia alapján, amelynek segítségével kiválasztják a legjobb ANN-t. A NAS-alapú megoldások bebizonyították, hogy felérnek vagy akár felül is múlják az ember által tervezett modelleket, kevésbé bonyolult és kisebb méretű architektúrákat eredményezve, így ígéretes megközelítést jelentenek az ALPR számára.

Az ALPR háromlépéses folyamatként tekinthető: (1) a rendszám megtalálása – egy objektumfelismerési feladat; (2) az egyes karakterek szétválasztása – szegmentálási feladat; (3) a karakterek azonosítása – osztályozási feladat. Az ALPR-kutatásokban gyakori, hogy a lépéseket külön-külön vizsgálják, mivel eltérő típusú kihívásokat jelentenek a szükséges neurális hálózat típusát illetően. Ebben a munkában egy NAS-alapú objektumfelismerő modellt javaslok az (1) lépéshez, amely továbbfejleszti az irodalomban található meglévő megoldásokat, a Differenciálható Architektúra Keresés megközelítéséből inspirálódva. Az alacsony energiafogyasztást megengedő helyzetekben a garantáltan valós idejű teljesítmény érdekében a javasolt architektúrát a state-of-the-art You Only Look Once modellekkel hasonlítom össze, nemcsak GPU-n, hanem Raspberry Pi-n is. Az ALPR további lépéseinek befejezéséhez a megtalált került rendszámokat optikai karakterfelismerési algoritmuson keresztül futtatom a karakterek felismerése érdekében. A teljes folyamat benchmarkingját is tárgyalni fogom.

Chapter 1

Introduction

Automatic License Plate Recognition (ALPR) has become increasingly critical for applications such as traffic monitoring, law enforcement, and smart city infrastructure. With the growing number of vehicles on the road, especially in urban areas, ALPR systems provide an efficient solution for tracking and managing vehicle flow without manual intervention. ALPR is a contemporary and rapidly evolving field, demanding advanced scientific knowledge and thorough understanding to tackle its complex challenges, particularly for applications on edge devices. Traditional ALPR systems rely on hand-designed models, which can be computationally intensive and not always suited for low-power edge devices, such as those often deployed in real-world scenarios with limited processing capacity. This gap underscores the need for advanced, energy-efficient models capable of delivering real-time performance on constrained hardware. In addressing this challenge, this work not only presents novel industrial solutions but also contributes significant research results relevant to the field.

This study explores the application of Neural Architecture Search (NAS) to develop a high-performance ALPR system optimized for edge devices. By combining NAS with Bayesian Optimization through the Ax platform, the study efficiently explores a diverse architecture search space to identify models that balance accuracy and speed, particularly for the task of license plate detection. Each candidate model is evaluated on both high-power GPUs and resource-constrained platforms like the Raspberry Pi, providing a comprehensive analysis of their performance.

This work focuses on training ALPR models from scratch, allowing for a uniform comparison across architectures without the influence of pre-trained weights or transfer learning biases. Comparing models can be approached from multiple perspectives—such as accuracy, speed, memory footprint, or computational efficiency—each tailored to specific deployment requirements. In this study, the emphasis is on two key metrics: Intersection over Union (IoU) and latency. IoU serves as a measure of detection accuracy, indicating how well the predicted bounding boxes align with ground truth labels, while latency captures the inference time, which is critical for real-time applications on edge devices. By evaluating models through these lenses, this work aims to identify architectures that not only maintain high detection accuracy but also ensure rapid inference, meeting the demands of efficient and effective license plate recognition in real-world scenarios. This comprehensive approach leads to well-founded results that significantly advance the field.

Furthermore, this work compares the best NAS-discovered models with the widely-used YOLOv8 benchmark to showcase how these optimized architectures can achieve comparable or superior performance in detection speed while maintaining high accuracy. This

approach aims to provide a scalable, real-time ALPR solution for deployment on existing infrastructure without the need for extensive hardware upgrades. The results demonstrate that NAS-driven solutions have substantial potential to advance the ALPR field, making them highly suitable for practical, real-world applications in edge environments. Although the field is not closed, and there is room for further innovation, this system already surpasses the state-of-the-art in many respects.

The remainder of this work is structured as follows: Chapter 2 provides a literature review, covering the evolution of Automatic License Plate Recognition (ALPR) systems, from traditional computer vision techniques to modern deep learning approaches and discusses the role of Neural Architecture Search (NAS) in optimizing models for edge devices. Chapter 3 presents the methodology underlying this study, beginning with an overview of the motivation and objectives that drive this work followed by an in-depth description of the selected tools to create and benchmark the proposed solutions. Chapter 4 gives a detailed description of the experimental setup, presenting the dataset, the building blocks of searched architectures and the target hardware. In addition it also introduces the NAS pipeline and discusses the critical parameters for model training. Chapter 5 presents the results of both the NAS experimentation phase as well as the full scale model training. It benchmarks the resulting architectures with the state-of-the-art YOLO and LPLite models before and after quantization. Finally, Chapter 6 concludes this work by highlighting the significance of this study and proposes directions for future work.

Chapter 2

Literature Review

2.1 ALPR with Classical Methods

Automatic License Plate Recognition (ALPR) systems have been integral to numerous applications in areas like traffic management, security, and tolling, attracting considerable research interest long before neural networks became a staple. In [1], the authors provide a detailed review of traditional ALPR methods, highlighting their structured, multi-step approach. They define ALPR as a four-stage process, with the initial two stages being most relevant to this study: (1) vehicle detection, followed by (2) license plate extraction, then (3) segmentation of the license plate into individual characters, and finally (4) character recognition.

Classical approaches to these stages frequently relied on edge detection and boundary analysis. For instance, methods using Sobel filters or vertical edge detection algorithms (VEDA) performed well under varied lighting conditions, effectively identifying the rectangular plate edges. However, they often struggle in scenes with many irrelevant edges, leading to false positives. Another common approach, Connected Component Analysis (CCA), is a binary image processing technique that groups adjacent pixels to isolate objects like the license plate region. CCA is often effective but depends heavily on accurate thresholding to create a binary mask. If thresholding fails, this method can mistakenly fragment the license plate into multiple regions, thereby compromising the accuracy of segmentation.



Figure 2.1: Example of edge detection-based license plate detection [2].

In addition to edge-based techniques, texture- and color-based extraction methods were also explored. These methods are designed to detect changes in texture or color at the license plate boundary, distinguishing it from its surroundings. Color-based techniques have proven especially effective in outdoor scenarios where natural lighting aids in contrast, but they are often sensitive to changes in lighting, making them less reliable in dim or variable light conditions.

Despite these advancements, combining edge, texture, and color features in a single pipeline to improve robustness leads to higher computational demands, limiting the feasibility of these methods on standard, low-power hardware. This trade-off between robustness and computational efficiency has been a persistent challenge, one that has driven the transition toward neural network-based ALPR methods that can learn multi-stage recognition processes with fewer resource-intensive feature engineering steps.

2.2 ALPR with Neural Networks

With the emergence of Convolutional Neural Networks (CNNs), ALPR systems significantly improved in accuracy, especially in unconstrained environments involving sloped angles, variable lighting, and partial occlusions. CNNs address these challenges by learning complex patterns directly from data, allowing for robust feature extraction across diverse conditions. In neural network-based ALPR, two main approaches have emerged: (1) end-to-end models that integrate both license plate detection and character recognition into a single pipeline, and (2) the more traditional, widely adopted approach that separates vehicle and license plate detection from character segmentation, employing specialized models for each stage.

In [3], the authors introduce a unified deep learning model that performs both license plate detection and character recognition in a single forward pass. The architecture combines convolutional layers for feature extraction, a region proposal network for plate detection, and recurrent neural networks (RNNs) with Connectionist Temporal Classification (CTC) for sequence recognition. This model achieves 96.13% recognition accuracy on datasets like CarFlag-Large and processes images in just 300 ms on a Titan X GPU, significantly outperforming the traditional two-stage approach, which requires 450 ms per image.

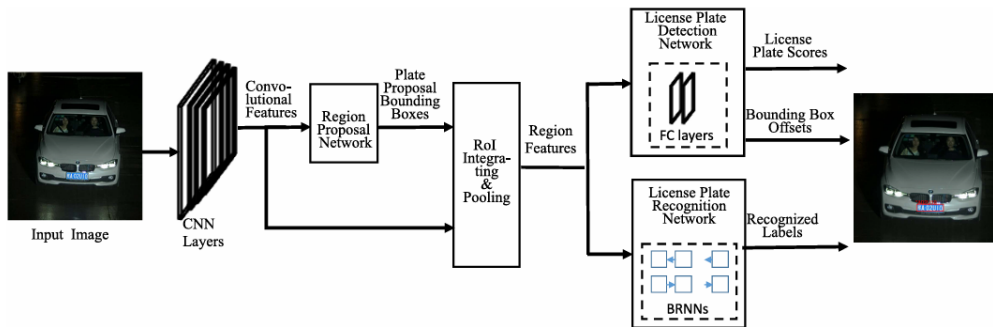


Figure 2.2: Combined architecture for end-to-end ALPR from [3].

The paper by Zhenbo Xu et al. [4] introduces CCPD, the largest and most diverse publicly available license plate (LP) dataset with over 250,000 images annotated with bounding boxes and vertex locations. To enhance ALPR, the authors propose a novel architecture, Roadside Parking Net (RPnet), which integrates detection and recognition into a

single forward pass using shared convolutional features. RPnet is designed to be efficient and accurate, operating at over 60 frames per second with 98.5% recognition accuracy, significantly outperforming existing methods. The study emphasizes the importance of large, diverse datasets and unified models for robust ALPR in real-world, unconstrained environments.

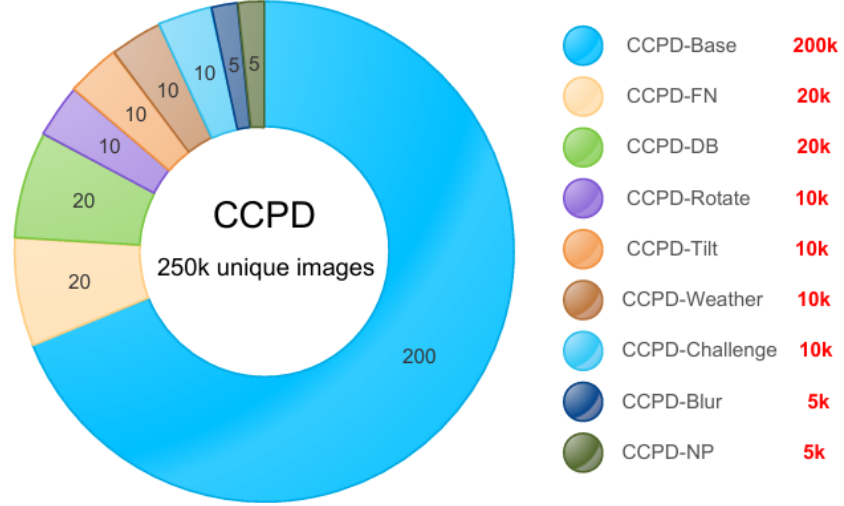


Figure 2.3: Distribution of the CCPD dataset from [4].

In [5], the authors present a highly efficient ALPR system that leverages the YOLO object detection framework in a multi-model approach. The system begins with two YOLO-based models for vehicle and license plate detection, followed by a segmentation model to locate individual characters, and concludes with two separate classification models for letters and digits. To further enhance performance, the authors applied data augmentation techniques, including inverted and flipped license plates. The system achieved an impressive recognition rate of 93.53% at 47 frames per second (FPS) on the SSIG dataset, outperforming commercial systems like Sighthound and OpenALPR.

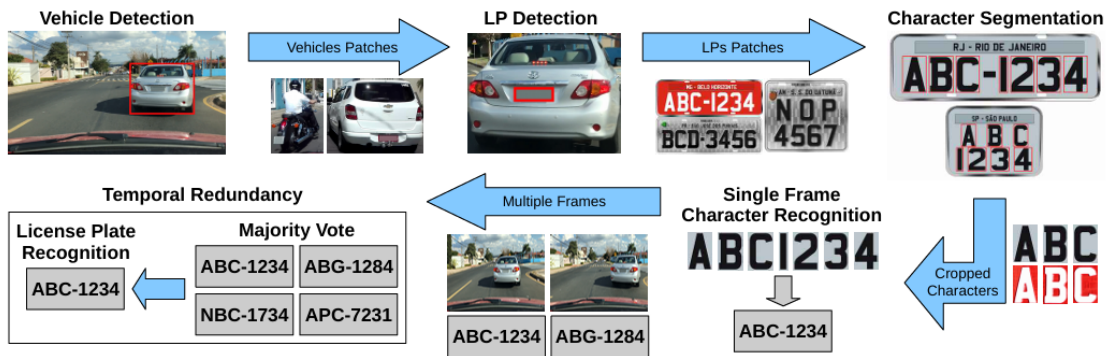


Figure 2.4: Architecture of the YOLO-based solution from [5].

In [6], a two-step algorithm for license plate identification uses a modified Hourglass network based on CenterNet. This approach involves initial plate detection followed by precise corner detection, enabling perspective correction for improved character recognition. The refined model, 40% smaller than standard, achieves 405 frames per second and an average accuracy of 96.19%, excelling in varied conditions like challenging angles and lighting.

While these ALPR models are cautiously hand-designed to achieve high accuracy and speed, Neural Architecture Search (NAS) offers a promising framework to automate and optimize the architecture selection process, potentially surpassing manual designs in both efficiency and performance.

2.3 Neural Architecture Search

Neural Architecture Search (NAS) automates the design of neural network architectures, shifting away from labor-intensive, manual architecture engineering. As outlined in [7], NAS seeks to optimize neural architectures for tasks like image classification and object detection, often surpassing human-designed networks in performance. NAS operates within three core components: (1) the search space, which defines potential architectures, (2) the search strategy, which explores these architectures, balancing efficiency and coverage, and (3) performance estimation, which evaluates architectures’ effectiveness while minimizing computational costs. This framework enables NAS to discover high-performing architectures with minimal human intervention, paving the way for enhanced adaptability and efficiency in deep learning models.

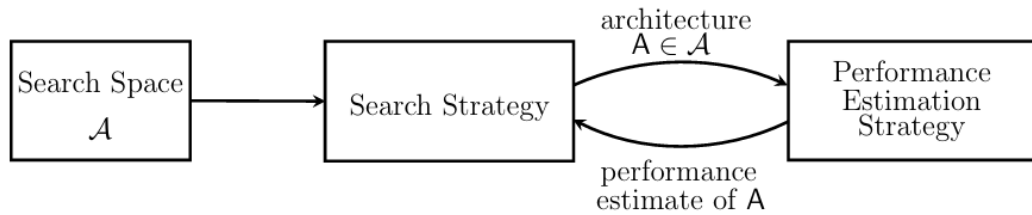


Figure 2.5: Illustration of a general Neural Architecture Search pipeline [7].

NAS employs various search strategies to explore the vast space of possible network architectures efficiently. Random Search offers a baseline by randomly sampling architectures, though it’s often computationally expensive and lacks direction. Reinforcement Learning (RL)-based NAS uses an agent that iteratively learns to design better architectures by receiving rewards based on model performance, guiding it to increasingly optimal configurations over time. Evolutionary Algorithms simulate biological evolution, where a population of architectures evolves through mutations and crossovers, selecting only the fittest models after each generation. Bayesian Optimization (BO) uses probabilistic models to estimate the performance of architectures, narrowing down the search space by focusing on high-potential areas; BO is particularly useful when evaluating architectures is costly.

In NAS, proxy datasets — smaller or simplified versions of the target dataset — are often used to reduce the computational cost and time associated with evaluating different architectures. This approach, while efficient, can lead to suboptimal architectures that don’t generalize well on the full dataset. ProxylessNAS [8] overcomes this by directly training and evaluating candidate architectures on the target dataset and hardware without relying on proxies. It does so by using a differentiable search process, allowing efficient exploration without compromising on real-world performance. Similarly, One-for-All (OFA) [9] mitigates the need for proxies by training a supernet that encompasses multiple sub-networks of varying sizes and complexities, which can then be selected or fine-tuned based on specific resource constraints. However, these methods are highly resource-intensive,

requiring significant computational power and memory to evaluate and store large-scale architectures directly on the target dataset and within supernetworks.

2.3.1 ALPR and NAS

In [10], the authors present a method to develop efficient ALPR systems specifically tailored for resource-constrained edge devices. By employing NAS, they design lightweight neural networks optimized for both accuracy and computational efficiency, with a focus on real-time performance and energy efficiency. Inspired by [11], they build a latency table of the selected building blocks of future architectures and benchmark them on the selected target device. Then, they use this latency table to tune the loss function of NAS during model search. They also explore an alternative method, called partially-connected differentiable architecture search (PC-DARTS) [12], to create an additional set of models that are hardware agnostic. These approaches yield models that achieve competitive accuracy while functioning effectively on low-power platforms. In a subsequent study, [13] the same research group advances this work by deploying a hardware solution using the previously optimized models.

To conclude, it is evident that NAS-driven ALPR solutions are in their nascent stages, yet they hold substantial promise for further advancements. With ongoing research and innovation, these methods have the potential to redefine the capabilities of ALPR on resource-constrained devices, making real-time, high-accuracy recognition feasible on affordable hardware. This work aims to contribute to this pioneering field, advancing the state-of-the-art in energy-efficient ALPR systems that can adapt to real-world constraints. The continuous development of NAS methodologies will only expand the possibilities for practical and scalable ALPR applications in the future.

Chapter 3

Methodology

3.1 Problem Background and Motivation

The number of vehicles on the road has been steadily increasing worldwide, contributing to rising congestion, environmental concerns, and demands on infrastructure. According to the International Organization of Motor Vehicle Manufacturers, global car ownership exceeded 1.4 billion in recent years [14], with emerging economies experiencing rapid growth in vehicle usage as access to affordable transportation expands. This growth is particularly pronounced in emerging economies, where expanding middle classes and increased access to affordable transportation have led to rapid increases in vehicle ownership. For instance, between 2015 and 2020, Vietnam experienced an annual growth rate of 17% in its motorization rate, while China and India saw annual increases of 14% and 10%, respectively [15]. This upward trend is expected to continue, especially in urban areas. The United Nations projects that by 2050, approximately 68% of the global population will reside in cities, up from 55% in 2018, underscoring the need for advanced solutions to manage traffic flow, enhance safety, and improve mobility in increasingly congested urban environments [16].

Automatic License Plate Recognition (ALPR) systems play a pivotal role in supporting this need, offering efficient, scalable solutions for vehicle monitoring and management. ALPR systems enable a range of applications, from traffic law enforcement and toll collection to parking management and urban planning. In addition to supporting traffic efficiency, ALPR can contribute to enhanced security through automated tracking and monitoring capabilities. By automating vehicle identification and reducing the manual oversight required, ALPR provides a practical tool for cities and organizations to address the challenges posed by growing vehicle numbers, making it a critical asset in modern smart city infrastructures.

Advancements in machine learning, particularly in deep learning, have transformed the field of computer vision, enabling highly accurate object detection and recognition systems that were previously challenging or impractical to implement. Convolutional Neural Networks introduced robust feature extraction capabilities, allowing models to identify complex patterns and structures in images, such as vehicle shapes and license plates, even in varying conditions. With the rise of high-performance GPUs and optimized frameworks like TensorFlow and PyTorch, training large, accurate models has become more accessible and efficient, reducing the barriers to deploying sophisticated computer vision solutions in real time. As a result, the development of ALPR systems has become more feasible, as modern machine learning models can now handle challenges like varied lighting,

angles, and image resolutions that are common in real-world scenarios. These advancements enable ALPR solutions to operate reliably and efficiently, paving the way for broad application in areas such as traffic monitoring, security, and smart city infrastructure.

Most modern neural networks rely on intensive computations and are often developed without consideration for the hardware limitations in road management infrastructure. However, techniques like Neural Architecture Search enable the discovery of models that are not only lightweight but also compatible with the sometimes outdated systems commonly used in road management. By automating architecture design to meet specific constraints such as low power consumption and minimal latency, NAS facilitates the deployment of efficient ALPR models that can operate reliably within existing infrastructure, extending the capabilities of these systems without requiring extensive hardware upgrades.

3.2 Objective and Scope of the Study

The primary objective of this study is to use Neural Architecture Search to develop an efficient model for license plate (LP) detection as part of my broader MSc thesis project focused on end-to-end ALPR. Due to time constraints prior to the student conference, this paper presents only the LP detection phase, where NAS is leveraged to find a high-performance, resource-efficient model capable of operating on real-world road management infrastructure. NAS enables the automated search for architectures tailored to specific requirements like speed and power efficiency, allowing the selection of lightweight models that can run on hardware-constrained environments without sacrificing accuracy.

The study aims to set up a NAS pipeline that can identify suitable models for LP detection and benchmark these models against widely used YOLO-based solutions. YOLO models are renowned for their high accuracy and speed in object detection tasks but are often computationally intensive, which can be a limitation in resource-limited systems. By focusing on three main aspects—(1) establishing an effective NAS pipeline, (2) identifying models optimized for LP detection, and (3) comparing performance with out-of-the-box YOLO models—this work seeks to demonstrate that NAS can yield competitive results for LP detection with the added benefit of efficiency. The findings will showcase NAS’s potential to enhance ALPR by providing lightweight, hardware-friendly models suitable for deployment in existing infrastructure.

3.3 Proposed Solution Approach

In developing this solution, I draw inspiration from the work presented in License Plate Recognition Using Neural Architecture Search for Edge Devices, which demonstrates how NAS can be applied to create efficient, edge-compatible ALPR models. Similar to their approach, I chose to use *CNN blocks* rather than single *CNN cells* as the primary building units in my NAS search space. By using larger, pre-configured blocks from architectures like MobileNet and Residual blocks, this design allows for richer feature extraction and more effective modeling of complex visual patterns in license plates while keeping computational costs manageable. This strategy is essential for LP detection tasks, as it balances efficiency with the capacity to handle variations in real-world conditions such as lighting and angle.

In addition, I adopted the idea of constructing a latency table to optimize the model for deployment on the target hardware, which in this case is a Raspberry Pi 4B. Measuring

the latency of each CNN block on the Raspberry Pi enables the NAS to prioritize blocks that maintain a balance between accuracy and speed, ensuring real-time performance. However, while the original study utilized FBNet and PC-DARTS for the NAS process, I diverged by implementing Ax, an adaptive experimentation platform designed for scalable optimization. Ax’s flexibility allows me to define customized constraints and objectives, making it well-suited to this project’s specific goals of lightweight, efficient LP detection. This tailored approach enables the exploration of architectures that are uniquely optimized for both hardware constraints and the requirements of LP detection, with the potential to outperform standard solutions like YOLO in both efficiency and compatibility with resource-limited hardware.

3.4 Benchmarking versus the state-of-the-art

The YOLO (You Only Look Once) family of models has set the standard for real-time object detection, combining high-speed performance with robust accuracy, making it an appealing choice for Automatic License Plate Recognition (ALPR) systems. YOLO models achieve real-time detection by processing images in a single forward pass, dividing the image into a grid, and predicting bounding boxes and class probabilities for each cell in parallel. This single-shot approach relies on several key techniques and optimizations, especially in its smaller, edge-friendly variants, such as YOLOv5 Nano and YOLOv8 Small, which focus on delivering both accuracy and efficiency.

YOLO models utilize anchor boxes, which are predefined bounding box shapes and sizes, to detect objects of varying sizes in a single image. By using these predefined anchor boxes, YOLO models can predict multiple potential bounding boxes for each cell in the grid, effectively capturing objects of different scales, which is critical for accurately detecting license plates of varying dimensions in ALPR tasks. However, selecting appropriate anchor box sizes and configurations requires careful tuning, as it directly impacts both accuracy and computational efficiency. Another essential component of YOLO models is Non-Maximum Suppression (NMS). After the model generates numerous bounding box proposals, NMS filters out overlapping boxes by retaining only the boxes with the highest confidence scores, ensuring a single prediction for each detected object. This is particularly useful in scenarios where overlapping detections are common, such as dense traffic where multiple vehicles and license plates are present.

In recent YOLO versions, including YOLOv8, optimizations known as Bag of Freebies (BoF) and Bag of Specials (BoS) have been introduced to further improve model performance without significantly increasing inference time. Bag of Freebies (BoF) encompasses techniques applied during the training phase to boost accuracy without impacting inference speed. Examples include label smoothing, data augmentation, and class balance adjustment. Label smoothing, for instance, prevents the model from becoming overly confident in its predictions, which can improve generalization. Similarly, advanced data augmentation techniques help the model learn to recognize license plates in a wider range of conditions, such as varying angles, lighting, and occlusions. Bag of Specials (BoS), on the other hand, refers to a set of architectural modifications specifically designed to enhance detection performance, particularly for small objects like license plates. Examples of BoS techniques include the addition of Path Aggregation Network (PAN) layers, which enhance feature fusion across different scales, and the use of Squeeze-and-Excitation layers, which enable the model to emphasize important features while suppressing irrelevant information. These BoS techniques are incorporated selectively, as they can increase model

complexity and inference time, making them more suited to scenarios where slightly more computational resources are available.

The latest YOLO version, YOLOv8 [17], integrates these advancements along with adaptive anchor generation, which automates the anchor box selection process based on the target dataset, and improved activation functions for faster convergence during training. YOLOv8 Small specifically targets edge devices, offering a compact model with reduced latency suitable for deployment in low-power scenarios. Despite these improvements, even the lightweight versions of YOLO can present limitations when deployed on extremely constrained hardware, such as basic edge devices or older security cameras, due to the relatively high computational cost associated with anchor box prediction and NMS.

The NAS approach in this study aims to leverage automated architecture search to develop models that meet the specific demands of license plate detection on resource-constrained devices, potentially outperforming even optimized YOLO variants. By exploring alternative architectures tailored to the precise needs of ALPR, NAS can generate efficient solutions that minimize resource consumption while maintaining accuracy. Through benchmarking against YOLO models, this study intends to showcase how NAS can yield competitive, hardware-friendly models that push the boundaries of what lightweight YOLO architectures can achieve in terms of speed and energy efficiency, thereby demonstrating a viable alternative for ALPR applications in real-world, low-power scenarios.

3.4.1 Ax: An Adaptive Experimentation Platform

Ax, developed by Facebook AI, is a flexible platform for adaptive experimentation designed to optimize complex systems and machine learning models. Originally created to facilitate large-scale A/B testing and hyperparameter tuning, Ax provides a structured approach to experiment design and evaluation, making it well-suited for Neural Architecture Search (NAS) tasks. By supporting custom constraints, multi-objective optimization, and integration with hardware-specific performance data, Ax enables a highly customizable experimental process. At its core, Ax systematically sets up experiments by defining objectives, constraints, and parameters within a specified search space. The platform then orchestrates trials through its Scheduler, shown in Figure 3.1, logging performance and iteratively refining its approach based on observed outcomes, which allows for efficient navigation toward optimal configurations.

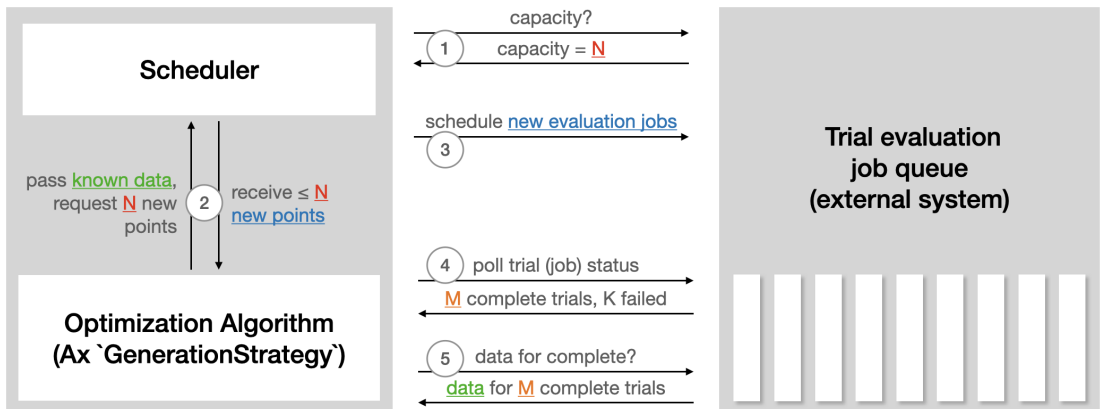


Figure 3.1: The Scheduler functions as the loop controller for optimization, coordinating with the backend to initiate trials, monitor their progress, and collect results [18].

For its generation strategy, Ax employs a combination of SOBOL sequences and BOTORCH, a library for Bayesian Optimization built on PyTorch. SOBOL sequences initialize the search with a quasi-random sampling method, effectively covering the search space without bias, which provides a strong baseline for exploration. BOTORCH then builds on this foundation by guiding further exploration through Bayesian Optimization, using a probabilistic model to prioritize high-potential areas of the search space based on past results. This dual approach of SOBOL initialization followed by BOTORCH refinement allows Ax to balance broad exploration with focused optimization, ensuring efficient and directed searches that yield robust, high-performance models.

Bayesian Optimization (BO) improves parameter tuning by using a probabilistic model to predict outcomes, unlike traditional methods like grid search, which are inefficient in high-dimensional spaces. BO constructs a surrogate model with Gaussian Processes (GPs) based on prior observations, allowing it to estimate performance and uncertainties for unseen parameter values. This surrogate model, alongside an acquisition function, identifies new, high-value parameter configurations to test, balancing exploration of unobserved regions with exploitation of known promising areas. This approach enables BO to achieve better results with fewer evaluations than other optimization methods.

To further increase efficiency, BO can be executed in a staggered parallel fashion, allowing several trials to run concurrently without fully sacrificing the benefits of sequential optimization. Ax’s optimization engine, BoTorch, supports popular acquisition functions such as Expected Improvement (EI), which encourages exploration by favoring configurations with high uncertainty. In noisy environments, BoTorch uses an adapted form of EI to continue optimizing while managing uncertainty. GPs within BoTorch use a kernel function to model relationships among observed data, refining predictions and uncertainty as new data points are evaluated, making BO particularly suitable for complex, resource-intensive parameter tuning tasks.

Chapter 4

Experimental Setup

4.1 Dataset and Preprocessing

The CCPD (Chinese City Parking Dataset) [4] was selected for this study because it is the largest and most comprehensive publicly available dataset for license plate (LP) detection, containing a diverse range of images collected in real-world conditions. The CCPD dataset includes over 250,000 images of vehicles with annotated license plates, captured under various conditions such as different lighting, angles, and occlusions. Each image has a high resolution, typically around 720x1160 pixels, providing clear details of the license plates as well as vehicle surroundings, which is advantageous for training robust models.



Figure 4.1: Resized images from CCPD showcasing the variety of the dataset. (a)-(d) frontal, rear, side, (e)-(f) acute angle, (g)-(h) nighttime views. Red bounding boxes highlight the license plates.

To ensure compatibility with edge devices and optimize computational efficiency, all images were resized to 224x224 pixels. This size is a common choice in object detection tasks on resource-constrained devices, balancing model accuracy and processing speed. Figure 4.1

presents a variety of sample images from the resized dataset, illustrating the diverse angles, lighting, and backgrounds present in the CCPD dataset.

Notably, the license plates in the CCPD images are typically centralized within the frame, which may limit the model’s ability to generalize to off-centered plates in real-world scenarios. To address this, I applied data augmentation using RandomAffine transformations with a translation range of (0.2, 0.2), which shifts the license plate position within the image. This augmentation introduces variability in plate location, enhancing the model’s robustness to position shifts. The distribution of dimensions, original, and augmented license plate positions can be seen in Figure 4.2, illustrating a more diverse placement achieved through this transformation.

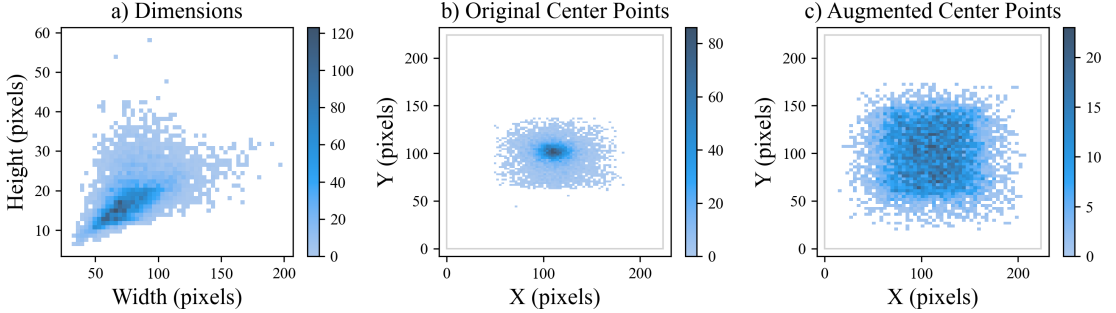


Figure 4.2: Distribution of license plate (a) dimensions, (b) center points without, (c) with augmentation in the training set. Gray boxes indicate the image borders.

Due to computational constraints, the original CCPD dataset was reduced to a subset of 10,000 randomly sampled images for training and 4,000 images for validation during full model training for both the viable NAS-found models as well as the baseline. For the NAS search, a proxy dataset was created with 2500 training and 1000 validation images, all randomly sampled from the original CCPD dataset. This selection provides a manageable yet representative sample of the full dataset, allowing for efficient model training both for NAS and the subsequent full training while maintaining diversity in vehicle types, lighting conditions, and angles.

4.2 CNN Block Selection

In configuring the Neural Architecture Search for license plate detection, I selected a diverse set of CNN blocks to explore various architectural possibilities. Firstly, I incorporated blocks inspired by the RPNets architecture, known for its efficiency in license plate recognition tasks [4]. These blocks are designed to effectively capture spatial features pertinent to license plate detection. Additionally, I included MobileNetV2 blocks, which utilize depthwise separable convolutions to reduce computational complexity while maintaining performance, making them suitable for deployment on resource-constrained devices [19]. Furthermore, I also included MobileNetV3 blocks, which feature squeeze-and-excitation (SE) modules that adaptively recalibrate channel-wise feature responses, enhancing the network’s ability to focus on informative features and further boosting efficiency [20]. To enhance the model’s receptive field without increasing the number of parameters, I integrated a Dilated Convolution Block with a dilation rate of 2. Dilated convolutions expand the kernel by inserting spaces between elements, allowing the network to capture multi-scale contextual information, which is beneficial for detecting license plates at various

scales and orientations [21]. An overview of the block configurations is shown in Figure 4.3.

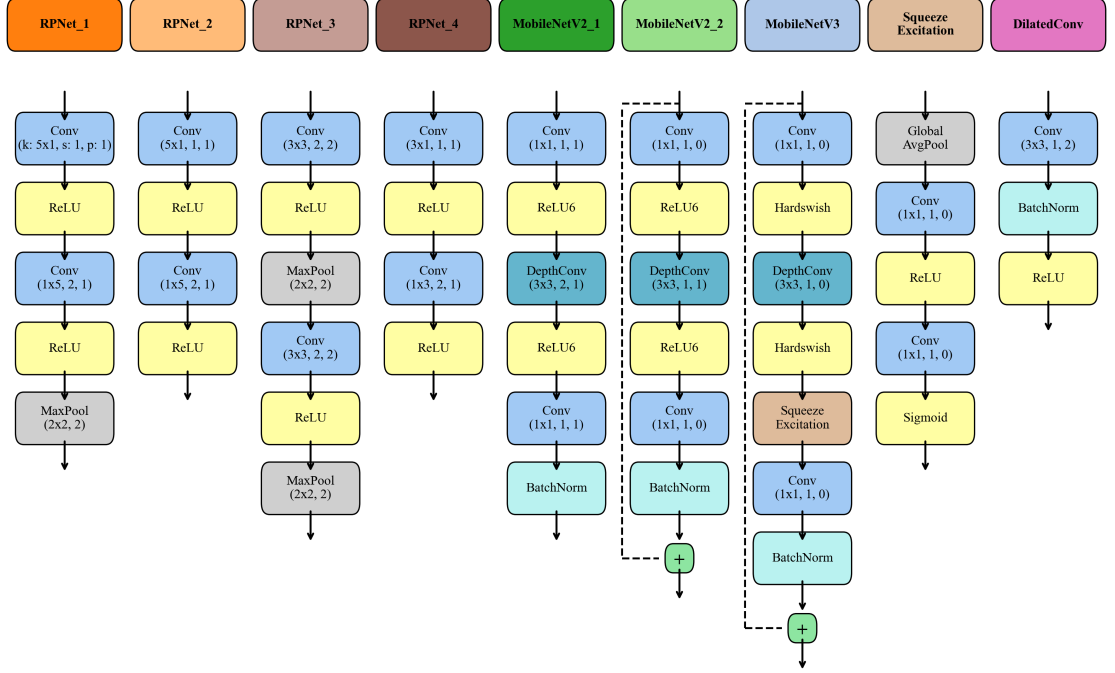


Figure 4.3: Schematics of blocks used in NAS for ALPR. Values in brackets: k=kernel size, s=stride, p=padding.

Furthermore, individual MaxPooling and AveragePooling cells were also included to down-sample feature maps, aiding in translation invariance and reducing spatial dimensions. An Identity cell was also added to allow the network to retain information across layers without modification, facilitating gradient flow and enabling the NAS to explore architectures with varying depths and complexities.

4.3 Target Hardware and Latency Table Creation

The Raspberry Pi 4B, 4GB model was selected as the target hardware platform for testing the final license plate detection models, primarily due to its cost-effectiveness and suitability for real-world deployment in low-resource settings. As an affordable and energy-efficient device, the Raspberry Pi 4B offers a practical solution for edge-based applications like ALPR, where budget and power constraints are often critical factors. With its quad-core ARM Cortex-A72 processor and 4GB of RAM, the Raspberry Pi 4B provides enough computational power to handle lightweight deep learning models while remaining accessible to a wide range of users and organizations.

To ensure that the NAS-selected models perform efficiently on this hardware, a latency table was created by measuring the inference speeds of each CNN block discussed previously, which is shown in Figure 4.4.

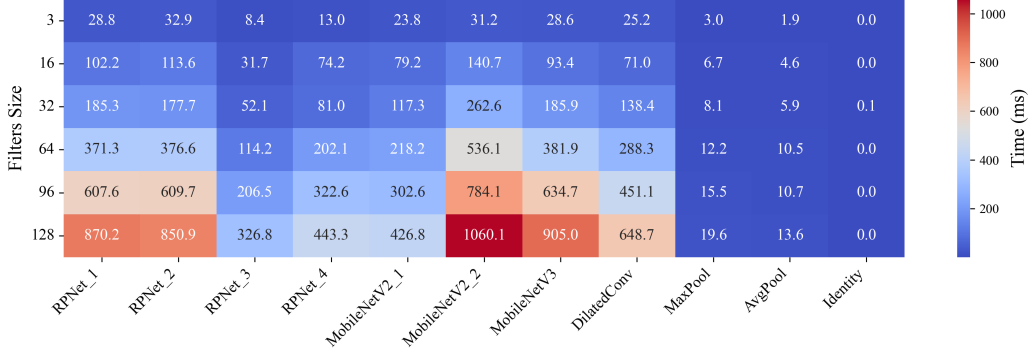


Figure 4.4: Inference latency measurements on Raspberry-Pi 4B of the blocks used for NAS.

4.4 NAS Pipeline Configuration

The NAS pipeline was configured using the Ax framework to systematically explore and optimize the architecture space for license plate detection. An Ax client was set up with a generation strategy for 60 generations, where the first 35 generations used a SOBOl sequence for broad exploration of the search space, followed by 25 generations utilizing BOTORCH strategies for refined optimization. This staggered approach allowed for efficient exploration in the early stages and focused refinement in the later stages, leveraging the strengths of both random and Bayesian optimization techniques. Each experiment was repeated over multiple days to accumulate a comprehensive set of architectural variations and performance data.

The Ax client was configured for multi-objective optimization with two primary objectives to minimize: (1) the IoU loss on the validation set, representing the model’s accuracy in detecting license plates, and (2) the latency of the architecture. The latency metric was calculated as the sum of the latencies of each block within the architecture, using the latency measurements obtained on the Raspberry Pi 4B.

The parameter configuration included key design elements, with depth set as a variable parameter, allowing the model’s depth to range from 2 to 12 layers. The possible layers were composed of the CNN blocks described earlier, including RpNet-inspired blocks, MobileNetV2 and MobileNetV3 blocks, a DilatedConvBlock, MaxPooling, AveragePooling, and an IdentityBlock, with filter sizes of 16, 32, 64, 94, and 128 channels, where applicable. Since certain CNN blocks reduce the data dimensions, some cell combinations led to infeasible architectures. In these cases, the evaluation function assigned a high penalty to steer the optimization process away from such incompatible combinations.

4.5 Training and Evaluation Setup

For the NAS optimization, initially, training was set at 5 epochs per model, but it became evident that some architectures required more time while others trained quickly. Consequently, I adjusted the training duration to a time limit of up to 5 minutes per model, ensuring efficient use of resources while allowing each model an equal time to reach a meaningful performance. As a result, most models did not fully converge during the Ax optimization phase; however, those that achieved strong results within the allocated

training time generally maintained their effectiveness during full training. Full training was limited to 300 epochs, but most models did converge earlier.

The Adam optimizer was used consistently across all training phases in this work. For NAS optimization, a fixed learning rate of 0.001 was initially applied; however, this rate proved too low for deeper models, limiting their ability to achieve reasonable performance within the time constraints. Raising the learning rate to 0.02 enabled significantly more models to progress effectively during the allocated training time. For full model training, various learning rate schedulers were evaluated. Although the CosineAnnealingLR scheduler with a starting rate of 0.001 yielded good results, the best performance was achieved with CosineAnnealing with warm restarts and weight decay [22]. A comparison between the various schedulers used in this work is shown in Figure 4.5.

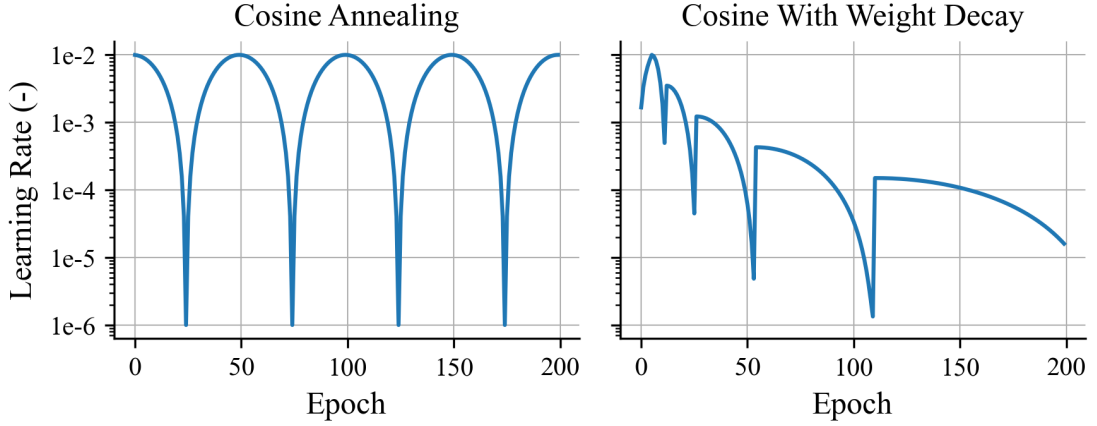


Figure 4.5: Comparison of CosineAnnealingLR and CosineAnnealing with warm restarts and weight decay.

Finally, all model training was conducted on a GTX 1060 6GB CUDA-enabled GPU. For comprehensive evaluation, model performance was tested across multiple platforms: on the GTX 1060 GPU, on an Intel i7-7700HQ processor, and on the Raspberry Pi 4 (4GB) with an ARM Cortex-A72.

4.6 Baseline Models for Comparison

For the baseline model YOLOv8n was used, the smallest model in what was the latest YOLO version at the time this project began. YOLOv8n was selected specifically for its balance of accuracy and computational efficiency, making it highly suitable for deployment on edge devices like the Raspberry Pi 4B. While YOLOv11 has since been released by Ultralytics, its launch came too late to be included in this study, so YOLOv8n serves as the primary benchmark for comparison in terms of both accuracy and latency.

The batch size was set to 32, ensuring efficient data processing without overwhelming memory resources, while the input image size was set to 224x224 pixels. The Adam optimizer was selected for gradient optimization and the cosine learning rate scheduling was enabled, starting with an initial learning rate of 0.001. Pretrained weights were disabled to allow for custom training specific to the license plate dataset. This setup aimed to provide a fair benchmark, leveraging the efficiency and accuracy capabilities of YOLOv8n in comparison to the NAS-optimized models.

Chapter 5

Results

5.1 The Ax Experiments

A total of 13 experiments were conducted to evaluate different model configurations for license plate detection. Each experiment had slight modifications of the training parameters and architecture settings with the goal to optimize performance. The key settings and results of these experiments are summarized in the Table 5.1, providing an overview of the model configurations and their corresponding validation outcomes. Each row in the table represents a unique experiment, identified by a Model ID and associated with a distinct Seed for reproducibility. The T/V Size column denotes the number of training and validation samples used. The Learning Rate (LR) column shows the flat learning rate applied to each model. The #M column indicates the number of individual model architectures explored within each experiment, and %Viable represents the percentage of viable models found during each run. Additionally, the MDepth column shows the maximum depth allowed for each trial and the ADepth is the average depth of all models found in each experiment. Max Obj provides the highest objective score obtained during training. The horizontal separations indicate major changes in the experimental setup, and the bold values represent the main talking point of each experiment block.

ID	Seed	T/V Size	LR	#M	%Viable	ADepth	MDepth	MaxObj
1	687	2.5k / 1k	0.001	60	70.00	2.89	5	0.59
2	774	2.5k / 1k	0.001	22	54.55	3.54	5	0.70
3	127	2.5k / 1k	0.001	19	36.84	4.43	8	0.67
4	948	2.5k / 1k	0.001	136	45.59	4.48	8	0.69
5	53	2.5k / 1k	0.001	34	11.76	8.50	12	0.57
6	394	2.5k / 1k	0.001	16	31.25	6.97	12	0.65
7	467	2.5k / 1k	0.001	69	42.03	5.36	10	0.63
8	573	2.5k / 1k	0.001	28	42.86	3.52	5	0.69
9	348	1.5k / 1k	0.001	50	58.00	3.88	5	0.47
10	215	1.5k / 1k	0.001	50	26.00	6.23	12	0.43
11	284	2.5k / 1k	0.01	60	78.33	4.14	8	0.65
12	949	2.5k / 1k	0.01	60	83.33	3.98	8	0.67
13	263	4k / 1k	0.02	49	71.43	6.10	8	0.67

Table 5.1: Summary of NAS training configurations.

Each experiment consists of two stages: (1) SOBOL exploration for the unbiased discovery of potential models, followed by (2) BOTORCH exploitation using insights from the exploratory phase to conduct a more targeted search. Experiment 1 served as a baseline, primarily for pipeline setup, and focused on shallow models as indicated by its lowest average depth. In Experiment 2, exploration depth was increased while reducing the number of trials, leading to the best candidate architecture discovered across all experiments. Experiment 3 aimed to explore even deeper models, but with a low percentage of viable outcomes, it was halted before reaching the exploitation stage. Experiment 4, building on previous learning, was the longest experiment, and it did result in several candidates for full training.

Experiments 5-10 introduced a more aggressive depth exploration. Starting with Experiment 5, the maximum model depth was set to 12 and the minimum to 6, in an attempt to target deeper architectures. However, the low viability of models indicated that this depth might not be optimal, as evidenced by a low maximum objective score. Consequently, Experiments 6, 7, and 8 revisited shallower model exploration, with minimum depths set at 5, 4, and 3, respectively. This approach yielded a higher percentage of viable models and a steady increase in maximum objective scores.

Deeper models, however, bring higher computational demands. Thus, in Experiments 9 and 10, the training set was reduced by nearly half, with Experiment 9 focusing on shallow models and Experiment 10 on deeper ones. Unfortunately, this reduction negatively impacted both viable model percentages and maximum objectives, showing that a smaller training set hindered performance.

Finally, in Experiments 11-12, the training size was restored to 2500 images, and a balanced approach with a maximum depth of 8 was adopted, turning the focus to learning rate adjustments. The ideal trial size was solidified at 60 runs (25 SOBOL + 35 BOTORCH). These experiments achieved high viable model percentages due to faster model training from the increased learning rate, giving deeper models a chance to perform well even in the time-constrained training implemented for NAS. By Experiment 13, the training size and learning rate were further increased, and most importantly, the MaxPool, AvgPool and Identity blocks were added into the search space.

5.1.1 Breakdown of Experiment 13

Figure 5.1 below provides a comprehensive view of the results from Experiment 13, showcasing key metrics across each trial, including objective scores, latency, training time, and the composition of CNN blocks in each model architecture.

In the Validation IoU plot (top row), we see a general upward trend in the objective scores as the experiment progresses. This indicates that the BOTORCH phase effectively refines the model architectures, leading to higher-performing configurations. The initial SOBOL exploration phase produces diverse objective values, with a gradual improvement as the experiment transitions into the BOTORCH exploitation phase, where targeted exploration results in more consistently high objective scores.

The Latency plot (second row) shows the estimated latency for each model configuration on the target hardware. Latency varies significantly in the early trials but becomes more consistent and generally lower as the experiment advances, indicating a preference for efficient models that meet both performance and computational constraints. This trend suggests that the BOTORCH phase is successful in identifying architectures that strike

a balance between high performance and low latency, optimizing for models suitable for edge deployment.

In the Time per Image plot (third row), we observe the actual measured training time per image on the GTX 1060 GPU. The training time shows some variability across trials, with occasional spikes indicating more complex architectures that likely contain additional or more computationally intensive blocks. Overall, however, training times stabilize as the experiment progresses, indicating that the selected models become more consistent in terms of computational requirements.

The Block Count plot (bottom row) provides insight into the composition of each model in terms of the building blocks used. The colors represent different types of blocks, such as RPNNet, MobileNet, Dilated Conv, Pooling, and Identity blocks. In the early trials, there is a diverse mix of block types, which reflects the broad exploration of the search space during the SOBOL phase. As the BOTORCH phase begins, we see a more balanced and targeted combination of blocks, with RPNNet and MobileNet blocks frequently used, indicating their suitability for achieving the experiment’s objectives.

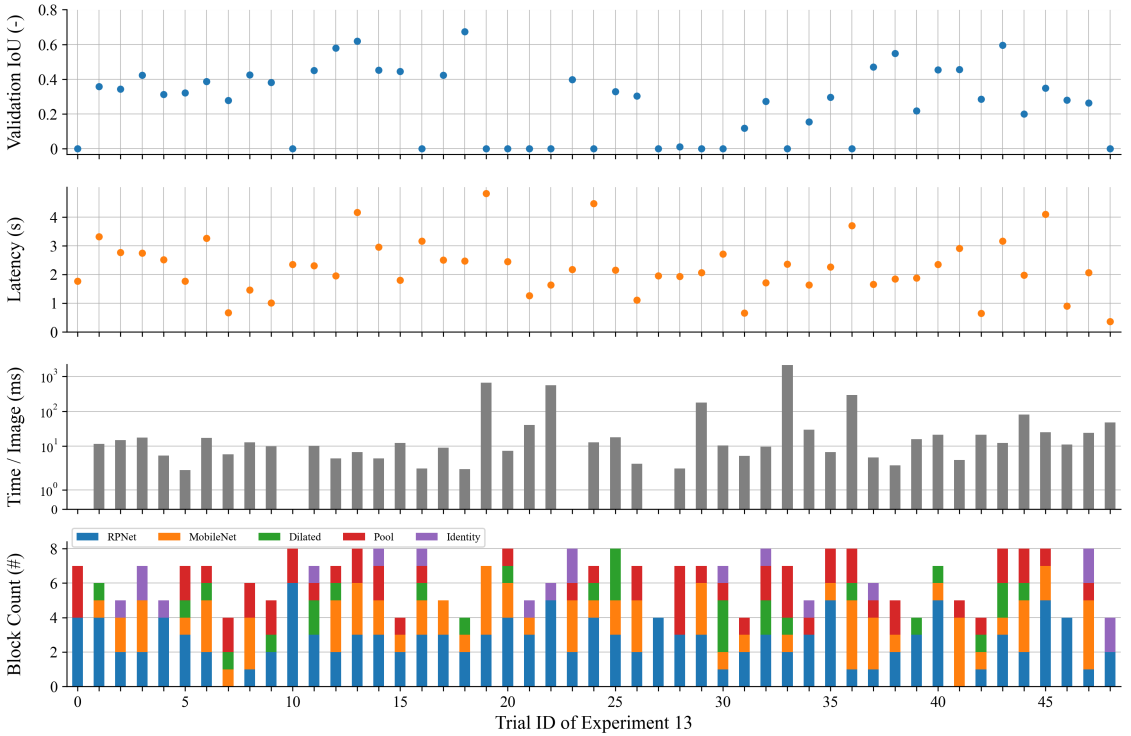


Figure 5.1: Overview of Experiments 13.

5.1.2 Candidate model selection from NAS experiments

Across the 13 NAS trials, a total of 653 architectures were evaluated, with a viability rate of 53.14%, resulting in 347 candidates suitable for full training. The performance of these models is summarized in Figure 5.2-(a), comparing their objective scores and estimated latency—both key optimization parameters. Since fully training all models was impractical, only those within the red box were selected for extended training. Figure 5.2-(b) provides a closer look at the performance and internal structure of these selected architectures, as identified during the NAS process. The latency values in the figure represent estimations based on the previously discussed latency table created on the target

device, while the Time per Image indicates the actual training time per image, as measured on the GPU used for training.

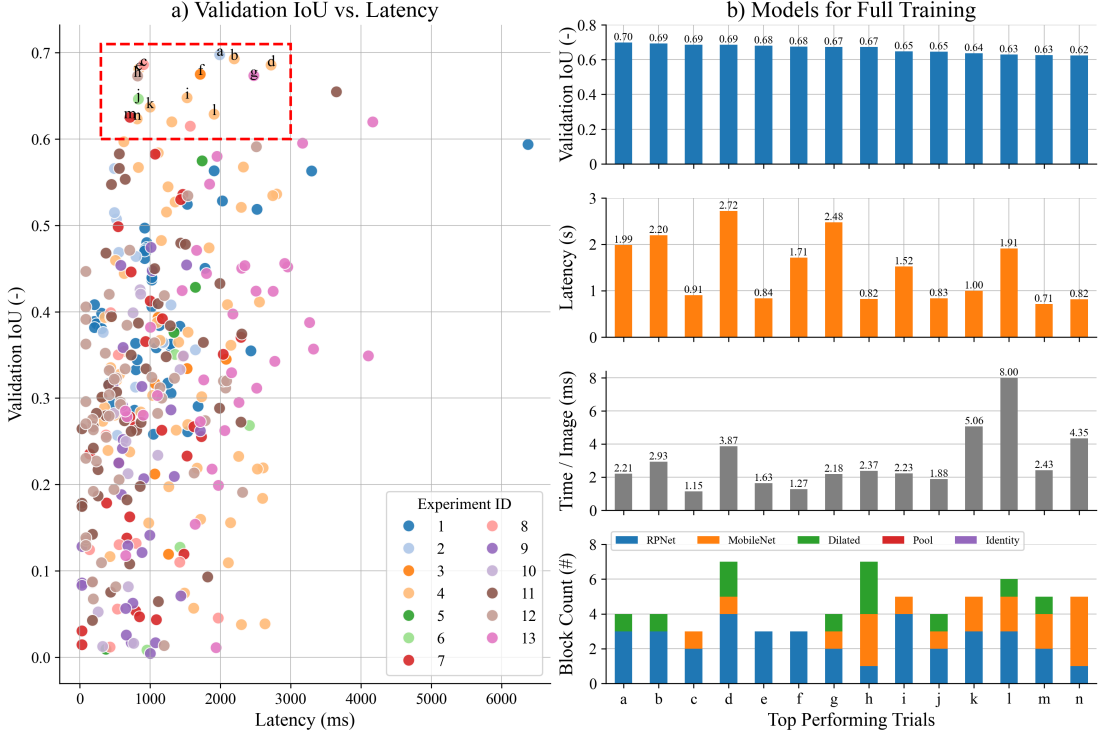


Figure 5.2: Best candidate architectures found during NAS. (a) Best Objective vs. Latency of all viable models found during NAS. (b) Breakdown of model performance of selected architectures for full training.

It is notable, though unsurprising, to observe a strong preference for RPNets blocks across all top-performing architectures. These blocks are specifically optimized for detection tasks and have demonstrated high effectiveness in ALPR applications [4]. MobileNet blocks, while well-suited for mobile and embedded devices, offer only moderate improvements with GPU acceleration, with relatively smaller speedups even during training [23], [19] - a phenomena that is also visible in the more MobileNet-heavy architectures, such as Models K and L.

To better understand the distinctions among the selected architectures, Figure 5.3 provides a detailed look at the internal parameters chosen for each model. Each architecture was augmented with a GlobalAveragePool layer and a final Linear layer with 4 outputs during training. These layers were not part of the NAS exploration but were necessary to ensure a consistent output format: a sequence of 4 values representing the bounding box corners. In Figure 5.3, the numbers shown beneath each block name represent the number of output channels generated by that layer. The input channels for each block correspond to the output channels of the preceding block, with the exception of the first block in each model, which has an input of 3 channels to match the RGB image input format.

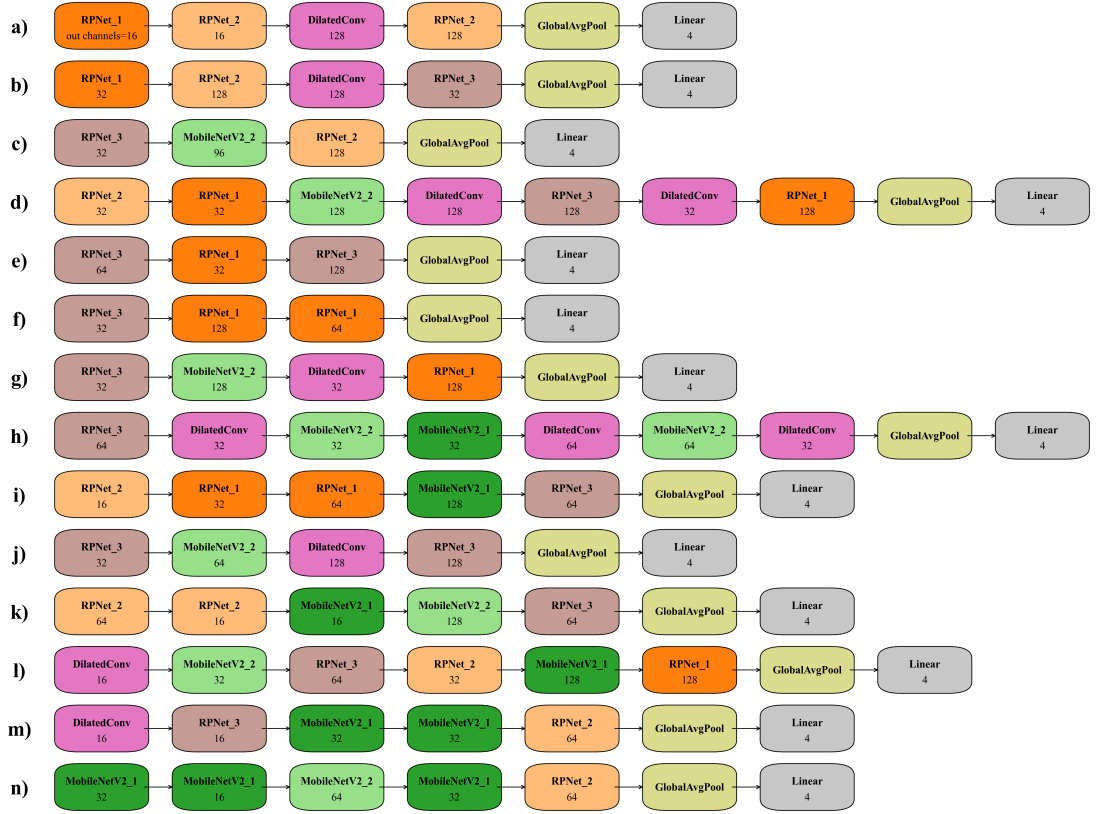


Figure 5.3: A look inside all the 14 candidate architectures. For each model, an average pooling layer is added before the fully connected layer with four outputs for the bounding box. Numbers in the boxes indicate the size of the output channel of each block.

5.2 Full Training

Once all the candidate architectures were established, the models were trained for 200 to 300 epochs. For the loss function, the initial thinking was to use IoULoss (which is equal to $1 - \text{IoU}$). However, it became quickly apparent that without any prior knowledge, the models struggled to progress in the right direction. As such, two alternative solutions were considered: MSELoss and SmoothL1Loss. MSELoss is commonly used for bounding box regression because it ensures that predictions are closely aligned with ground truth coordinates, though it may be sensitive to outliers, leading to higher penalties for large deviations. Smooth L1 Loss, also known as Huber Loss, combines aspects of L1 and L2 losses, applying a less steep penalty for larger errors and reducing sensitivity to outliers. It is particularly useful in object detection tasks because it provides a balance, penalizing small errors effectively while being robust to occasional large deviations in bounding box predictions, making it more stable for localization tasks than MSELoss. While both options were explored separately, the final loss function, as shown in 5.1 was ultimately defined as a weighted sum of the three aforementioned losses. This combination provided steady progress across epochs and facilitated effective convergence of the models.

$$\text{loss} = 0.4 \times \text{IoULoss} + 0.3 \times \text{SLLoss} + 0.3 \times \text{MSELoss} \quad (5.1)$$

5.2.1 Learning Rate Experiments with the Best Architecture

In the beginning, a flat learning rate was applied, but it became apparent that it was not the right path. As such, CosineAnnealingLR was chosen as the scheduler as it allows for repeated cycles of gradual reduction and warm restarts, helping the models escape potential local minima and explore more of the loss landscape. Figure 5.4 shows the performance of the best architecture under various learning rate schedulers. From top to bottom, the figure illustrates the evolution of the Objective - which is the Validation IoU, the Training Loss and the Learning Rate. Despite the benefits of the scheduler, the training was still noisy, as shown in Fig 5.4-(a), this the decision was made to change to a chained scheduler solution that utilizes CosineAnnealing, but with weight decay. This adjustment provided smoother convergence by regularizing the model and further stabilizing the training, resulting in more consistent performance improvements across epochs. Further fine-tuning followed, where both the frequency of restarts, 5.4-(c), and the initial and final learning rates were tuned, 5.4-(d), until a suitable learning rate was found. This learning rate was then used for all other architectures in their subsequent trainings.

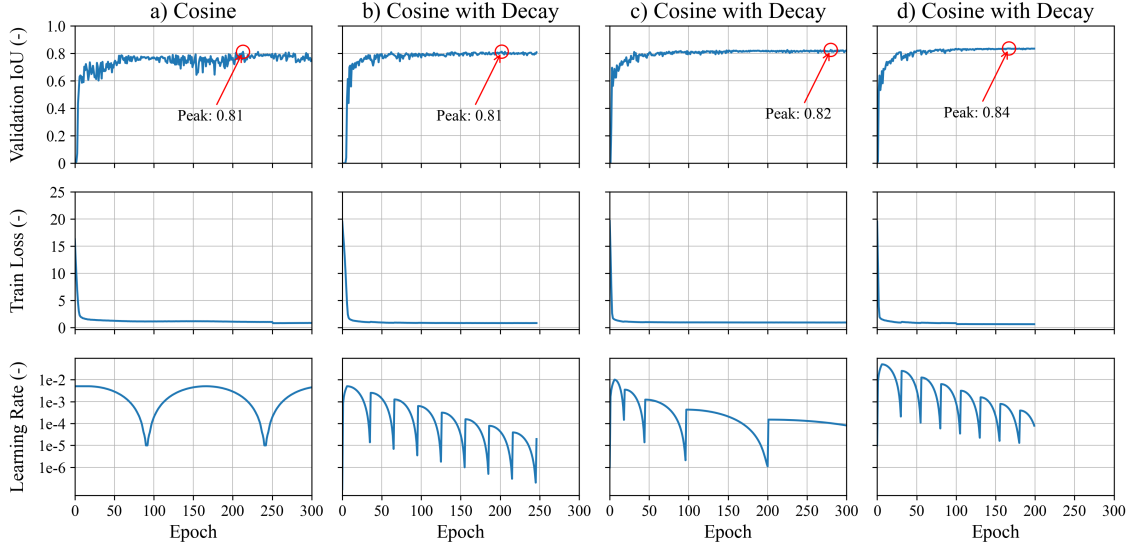


Figure 5.4: Learning Rates trials with Model H. (a) CosineAnnealingLR, (b)-(d) various flavors of CosineAnnealingLR with warm restarts and weight decay.

5.2.2 Summary of trained models

Table 5.2 provides a summary of the results from individual model training sessions. It's worth noting that the time-constrained training on the proxy dataset during NAS acted as an initial performance indicator, with full training not necessarily yielding significantly improved outcomes. This is especially evident in shallower models, where gains remain minimal. However, in cases like Models H, D, and K, extended training led to notable performance improvements, underscoring that deeper models are generally better suited for capturing finer details in license plate recognition, but they also need to be trained for longer.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Depth	4	4	3	7	3	3	4	7	5	4	5	6	5	5
NAS	0.70	0.69	0.69	0.69	0.68	0.67	0.67	0.67	0.65	0.65	0.64	0.63	0.63	0.62
Train	0.75	0.70	0.80	0.83	0.74	0.69	0.80	0.84	0.70	0.74	0.69	0.77	0.75	0.64
Gain	0.06	0.01	0.11	0.14	0.06	0.02	0.13	0.16	0.05	0.09	0.05	0.14	0.12	0.02

Table 5.2: Comparison of model Depth, Peak IoU values between NAS and Training for models A to N, including Gained performance.

5.3 Benchmarking

Comparing models involves assessing multiple aspects, as different models can excel in distinct areas. For instance, one model might deliver higher accuracy, making it more reliable for tasks where precision is critical, while another might perform faster, which is advantageous for real-time applications or when deployed on resource-constrained devices. As a result, the comparison of models is often based on specific metrics that align with the intended application goals, such as accuracy, speed, memory usage, or energy efficiency. In this work, we define a *better* model by evaluating both the Intersection over Union (IoU) scores, to measure accuracy, and the model latencies, to assess speed. This dual-criterion approach allows us to identify models that strike an effective balance between high detection accuracy and efficient processing, catering to the requirements of real-time applications.

To establish a thorough benchmarking process, all models were evaluated on an unseen portion of the CCPD dataset across three different devices: a GPU, a CPU, and a Raspberry Pi. This multi-platform testing provides insight into each model’s performance under various computational constraints. The results are presented in Figure 5.5. The top row of the figure displays the IoU distribution across the test set for each device, illustrating the consistency and accuracy of each model’s predictions. Meanwhile, the bottom row highlights the relationship between the mean IoU and measured inference time, offering a direct comparison of model efficiency. For reference, performance of the YOLOv8n model on each device is also included, serving as a benchmark for evaluating the relative strengths of the different architectures.

Across all three testing scenarios, the YOLOv8n model demonstrates the sharpest IoU distribution, with an average IoU score around 0.86, outperforming all NAS-based models in accuracy. However, in terms of inference speed, all NAS-derived models surpass YOLO on the GPU, achieving nearly double the inference speed. The top-performing models, H and D, reach mean IoU scores of 0.84 and 0.83, respectively, while maintaining a considerable speed advantage.

On the CPU, YOLO remains faster than some NAS models and achieves speeds comparable to model D. Nonetheless, model H exhibits a tighter IoU distribution peak, while still benefiting from the speed advantage observed in the GPU tests.

On the Raspberry Pi, the NAS-based models form two performance clusters: models B, D, K, and L show inference times close to YOLO’s, while the remaining models are nearly four times faster. Examining these four architectures in Figure 5.3, reveals that they incorporate more DilatedConv and MobileNet blocks with 128 channels, suggesting that these larger layers may contribute to the performance slowdown on lower-end hardware like the Raspberry Pi.

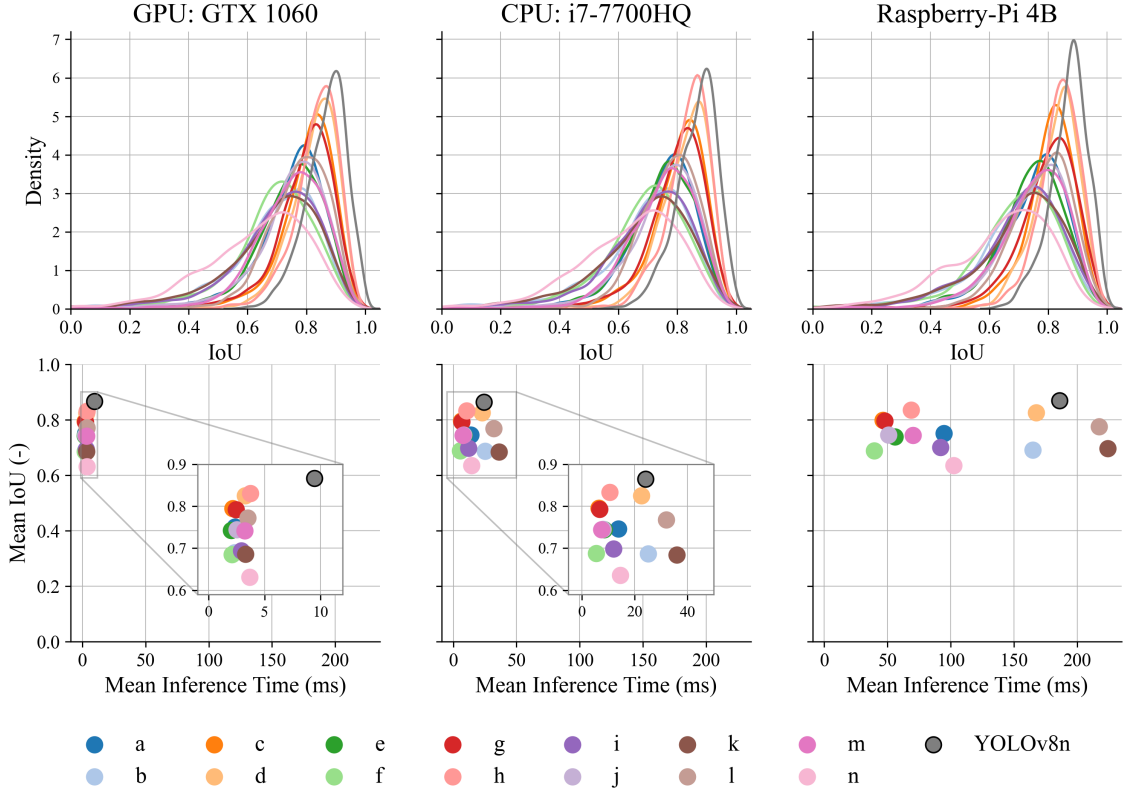


Figure 5.5: Performance comparison of the various models on GPU, CPU, and Raspberry Pi. Top: distribution of IoU values across the test set. Bottom: Mean IoU versus Mean Inference Time - measured.

In summary, while Model H exhibits a slight drop in IoU compared to YOLOv8n, the significant speed gains it achieves across different hardware platforms make it a strong competitor. Model H's efficient architecture enables it to maintain comparable accuracy while offering faster inference, particularly on the GPU and Raspberry Pi, where its performance is almost double that of YOLO. These advantages highlight Model H as a viable alternative for real-time license plate detection, especially in resource-constrained environments where speed is critical.

5.4 Quantization

Quantization is a technique used in deep learning to reduce the size and computational requirements of neural networks by approximating floating-point weights and activations with lower-precision values, such as 8-bit integers. This process is particularly useful for deploying models on resource-constrained devices like mobile phones and edge hardware, where memory and processing power are limited. Quantization can significantly improve inference speed, enabling faster and more efficient real-time applications. However, the primary drawback is a potential reduction in model accuracy, as the lower precision can lead to a loss of detail in the representations.

To enable the quantization of all models, the ONNX framework was selected due to its native support for YOLO models and straightforward pipeline setup for custom models.

[24]. The quantization targets chosen included Float16 (known in YOLO as half) and INT8 for compatibility across devices. Additionally, custom models were quantized to INT16, which is not supported by YOLO. Quantized models were benchmarked solely on the CPU and Raspberry Pi, with the results detailed in Figures 5.6 and 5.7.

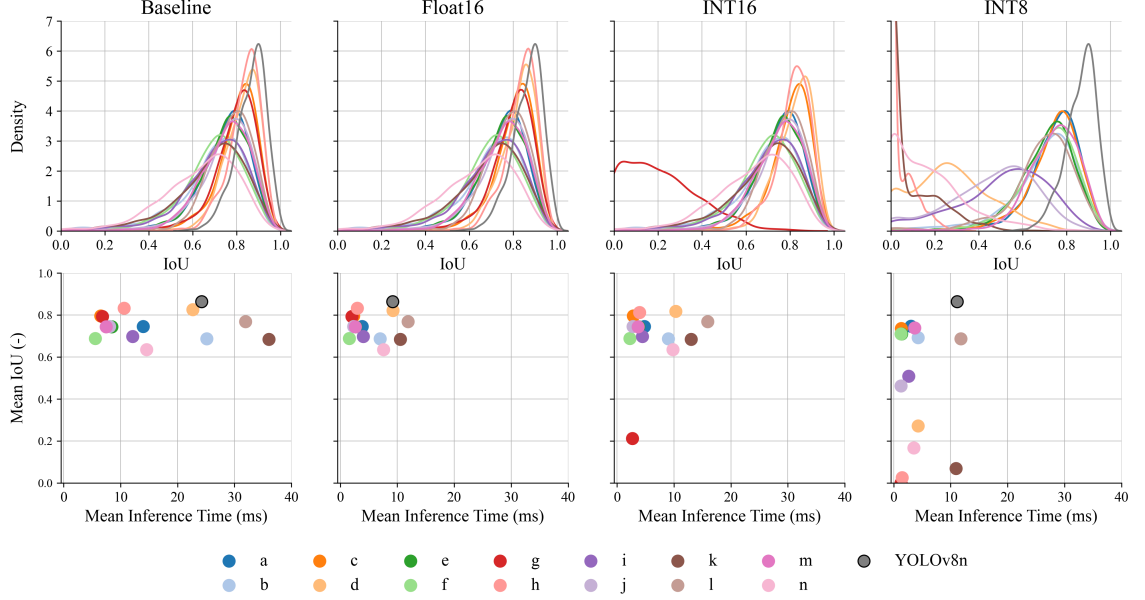


Figure 5.6: Performance evaluation of the various quantization measured on CPU.

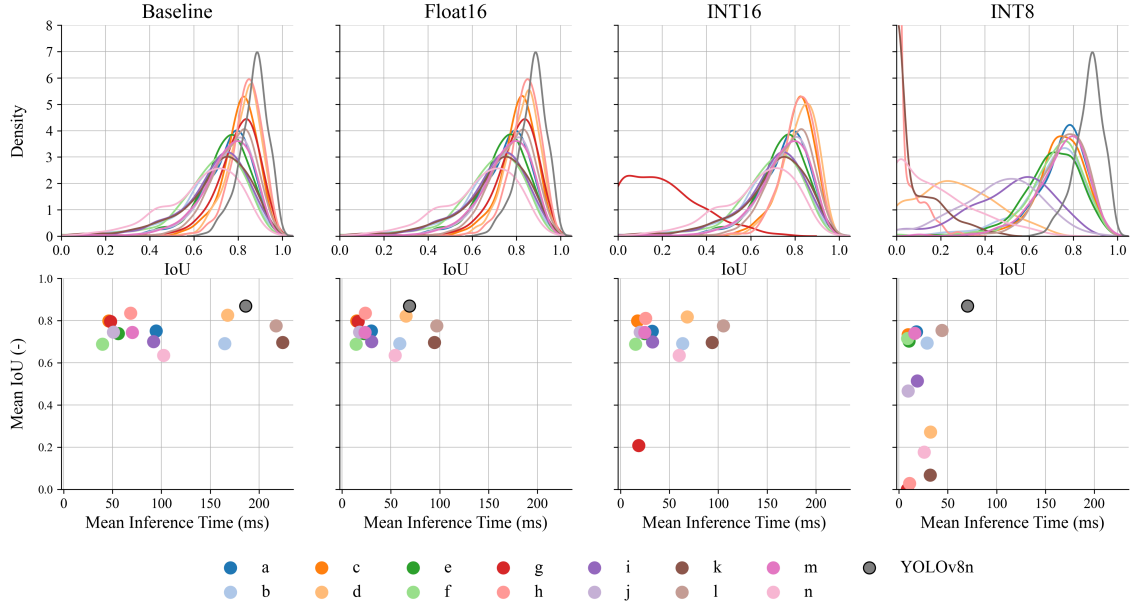


Figure 5.7: Performance evaluation of the various quantization measured on Raspberry Pi.

Figure 5.6 illustrate the impact of quantization on model speed when running on a CPU. Transitioning from the Baseline (Float32) to Float16 shows that some models achieve nearly double the inference speed without any noticeable loss in IoU performance. Notably, the previously slower models, along with YOLO, benefit most from this Float16 quantiza-

tion, highlighting how half-precision can significantly enhance performance. However, this improvement does not extend as effectively to INT16 and especially INT8, where custom models begin to experience heavy degradation in IoU. Interestingly, YOLO maintains its IoU robustness under INT8 quantization but gains no substantial speed improvement over its Float16 counterpart.

A very similar conclusion can be drawn from the test done on the edge device shown in Figure 5.7. This outcome is unsurprising, given that the CPU and Raspberry Pi had already exhibited similar patterns in Baseline model performance. The consistent response to quantization across these devices suggests that Float16 quantization provides a robust and effective approach for enhancing model speed on resource-constrained hardware without sacrificing accuracy.

5.5 Comparisons with LPLite

Finally, the performance of the top models identified by Ax NAS, alongside the YOLOv8n benchmark, is compared to the proposed LPLite family from [10]. It is essential to note that LPLite’s reported metrics were achieved under different conditions than those in this study. The LPLite family comprises NAS-optimized models trained on 480x480 pixel images across the complete CCPD-core dataset, using differentiable architecture search techniques. In contrast, the models in this paper were trained on a reduced dataset of 10,000 images with an input size of 224x224 pixels. Performances reported in [10] are measured on a Raspberry-Pi 3B with an Intel Neural Compute Stick 2 - which in principle should improve performance, and the results on GPU are measured on an NVIDIA T4 GPU, which significantly outperforms the GTX 1060 used for this paper. Despite these differences, examining these results remains valuable as LPLite represents the closest comparable study. The results are detailed in Table 5.3.

Model	Mean IoU	FPS (RPi)	FPS (GPU)
H Baseline	0.836	14	268
D Baseline	0.826	6	308
C Baseline	0.799	21	463
G Baseline	0.795	20	408
H Float16	0.836	42	-
D Float16	0.822	15	-
C Float16	0.799	66	-
G Float16	0.795	61	-
YOLOv8n	0.869	5	105
YOLOv8n Float16	0.869	14	-
LPLite_h	0.823	84	11
LPLite_hh	0.816	87	6250
LPLite_m	0.810	6	69
LPLite_mh	0.811	239	2857
LPLite_l	0.788	<1	71
LPLite_lh	0.784	<1	1075

Table 5.3: Comparison of top performing models with the YOLO benchmark and the LPLite family proposed in [10].

It’s no surprise that YOLO models have become so widely adopted—their detection performance is hard to surpass. However, even the *nano* variants come with complex architectures, making them relatively slow on edge devices and opening opportunities for alternative models that can rival their effectiveness. The top models in this study demonstrate promising potential in this regard, particularly when compared to both YOLO and LPLite. Notably, Baseline Model H achieves the best overall performance, surpassing all versions of the LPLite detectors in mean IoU. However, in terms of speed, it remains slower than much of the competition, though it still outperforms YOLO. Encouragingly, the quantized version of Model H reaches a processing speed of 42 FPS, making it well-suited for real-time applications.

Chapter 6

Discussions

6.1 Significance of this Study

This study provides a comprehensive overview of Automatic License Plate Recognition (ALPR) as a research field, examining recent advancements and utilizing them to develop new models specifically for license plate detection. The topic of this thesis is modern and not fully explored, requiring high-level scientific and in-depth knowledge. Leveraging Neural Architecture Search combined with Bayesian Optimization through the Ax Experimentation platform, the study explores a broad search space of potential architectures, aiming to identify models that are both accurate and capable of real-time performance on edge devices, which for this work was a Raspberry-Pi4B.

All models presented in this work are trained from scratch, unless stated otherwise, using the same datasets to ensure fair and consistent comparisons. By enforcing a strict train-test split, the robustness and validity of the results are ensured. Alongside the newly discovered architectures, a state-of-the-art YOLOv8 model is used as a benchmark, trained under identical conditions. This work successfully identifies several architectures that not only match but sometimes surpass the benchmark model in inference speed, delivering comparable prediction accuracy on both high-performance GPUs and resource-limited edge devices.

Finally, the architectures identified in this work are compared to the LPLite family, which serves as the most closely related benchmark available in the literature. Although this comparison is not entirely "apples to apples"—since the LPLite models were trained under different conditions, including larger image sizes and a complete dataset—it still provides valuable context. In fact, the use of a larger dataset and higher-resolution images likely gave LPLite an advantage, so achieving comparable or superior results under more constrained conditions highlights the effectiveness of the proposed approach. The comparison demonstrates that the approach discussed here successfully produces efficient, high-performing models that hold their own against established methods, reinforcing the viability of the proposed NAS-based strategy for real-time license plate detection on edge devices. Thus, the system is robust and scalable, as evidenced by the extensive test results, and represents a novel, yet non-existent solution in the field.

6.2 Future work

While this study has made significant advancements in developing efficient ALPR systems using Neural Architecture Search, there remain several opportunities to further enhance the models and address unresolved challenges. Future work can focus on refining latency estimations, exploring additional datasets and transfer learning techniques, and extending the system to include character recognition capabilities. The following sub-sections outline these potential directions.

6.2.1 Estimated Latency versus actual Inference Time

This work adopts the concept of a latency table-based approach for estimating architecture latency, where the inference speed of each individual building block is measured on the target device and then used as a reference to predict the combined latency of future models. However, keen observers may notice that the actual inference times recorded during final testing are notably lower than the initial estimates. Figure 6.1 presents a comparison of the estimated latency and model parameter count against the measured inference times. This discrepancy remains unexplained at the time of writing and highlights an area for further investigation.

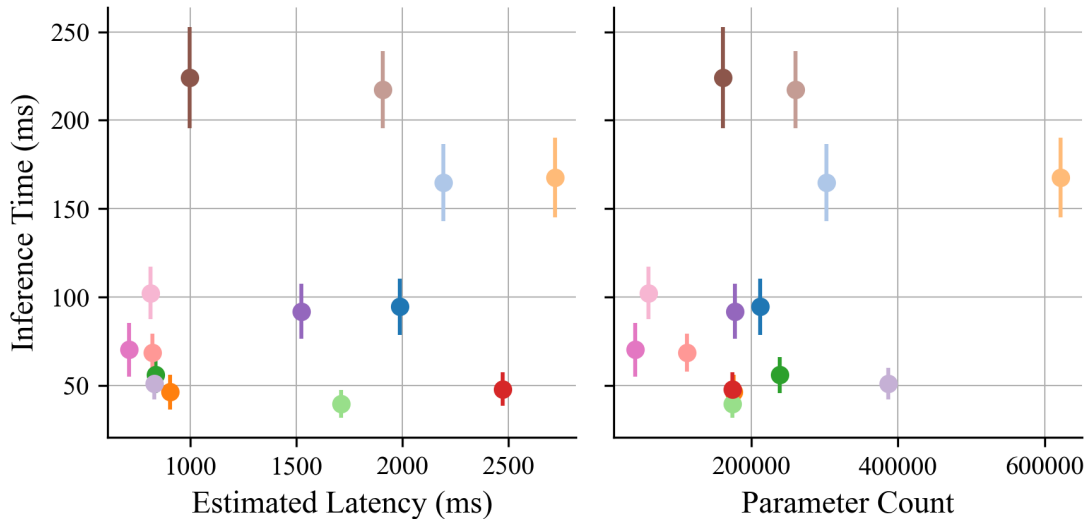


Figure 6.1: Comparison of Inference Time measured on the Raspberry-Pi versus the Expected Latencies and Parameter Counts.

6.2.2 Exploring other datasets and transfer learning

As a direction for future work, exploring additional datasets and incorporating transfer learning could further enhance the robustness and adaptability of ALPR models. Training on diverse datasets, encompassing a variety of lighting conditions, angles, and license plate formats, would likely improve model generalization to real-world scenarios. Leveraging transfer learning from large-scale datasets or related tasks could provide a valuable starting point by utilizing pre-trained weights that capture essential visual features, reducing the need for extensive training on limited data. This approach could lead to faster convergence,

improved accuracy, and optimized performance, particularly on resource-constrained edge devices.

6.2.3 Character recognition

This work represents a significant milestone in my MSc thesis, which aims to develop a complete, end-to-end license plate recognition system using two specialized models: (1) a license plate detection model—this current study—and (2) a character recognition model, which will form the second half of my thesis. Throughout this project, I gained substantial insights into detection models, experimented with a wide range of architectures, and refined training and benchmarking loops. The extensive experimentation and rigorous testing underscore the well-founded nature of the results, demonstrating both the robustness and scalability of the system. These learnings and optimizations will be valuable as I progress with the character recognition component and future research.

Bibliography

- [1] S. Du, M. Ibrahim, M. Shehata, and W. Badawy, “Automatic license plate recognition (alpr): A state-of-the-art review,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, no. 2, pp. 311–325, 2013.
- [2] D. Amusan, O. Arulogun, and A. Falohun, “Nigerian vehicle license plate recognition system using artificial neural network,” *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 11, pp. 1–5, 2015.
- [3] H. Li, P. Wang, and C. Shen, “Towards end-to-end car license plates detection and recognition with deep neural networks,” 2017.
- [4] Z. Xu, W. Yang, A. Meng, N. Lu, and H. Huang, “Towards end-to-end license plate detection and recognition: A large dataset and baseline,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 255–271, 2018.
- [5] R. Laroca, E. Severo, L. A. Zanlorensi, L. S. Oliveira, G. R. Gonçalves, W. R. Schwartz, and D. Menotti, “A robust real-time automatic license plate recognition based on the YOLO detector,” in *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–10, July 2018.
- [6] M. Kundrotas, J. Janutėnaitė-Bogdanienė, and D. Šešok, “Two-step algorithm for license plate identification using deep neural networks,” *Applied Sciences*, vol. 13, no. 8, 2023.
- [7] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey,” 2019.
- [8] H. Cai, L. Zhu, and S. Han, “Proxylessnas: Direct neural architecture search on target task and hardware,” 2019.
- [9] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, “Once-for-all: Train one network and specialize it for efficient deployment,” 2020.
- [10] J. Shashirangana, H. Padmasiri, D. Meedeniya, C. Perera, S. Nayak, J. Nayak, V. Shanmuganathan, and S. Kadry, “License plate recognition using neural architecture search for edge devices,” *International Journal of Intelligent Systems*, vol. 37, 05 2021.
- [11] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search,” 2019.
- [12] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong, “Pc-darts: Partial channel connections for memory-efficient architecture search,” 2020.

- [13] H. Padmasiri, J. Shashirangana, D. Meedeniya, O. Rana, and C. Perera, “Automated license plate recognition for resource-constrained environments,” *Sensors*, vol. 22, no. 4, 2022.
- [14] World Metrics, “Global car ownership statistics,” 2023.
- [15] Visual Capitalist, “Vehicles per capita by country,” 2023.
- [16] United Nations, “2018 revision of world urbanization prospects,” 2018.
- [17] Ultralytics, *YOLOv8 Documentation*, 2023.
- [18] PyTorch, *Multi-objective Neural Architecture Search with Ax*, 2023.
- [19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” 2019.
- [20] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, “Searching for mobilenetv3,” 2019.
- [21] F. Yu and V. Koltun, “Multi-scale context aggregation by dilated convolutions,” 2016.
- [22] S. Naeem, “Cosine annealing warm restarts with initial linear warmup followed by weight decay for pytorch.”
- [23] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [24] Microsoft, *ONNX Runtime Quantization Example for Image Classification on CPU*, 2023.