BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

SMART CAMERA FOR TRACKING OBJECTS OF INTEREST

CHYTRÁ KAMERA PRO SLEDOVÁNÍ ZÁJMOVÝCH OBJEKTŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. DAVID MIHOLA**
AUTOR PRÁCE

SUPERVISOR **doc. Dr. Ing. OTTO FUČÍK**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

165000

Institut:	Department of Computer Systems (DCSY)
Student:	**Mihola David, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Embedded Systems
Title:	**Smart camera for tracking objects of interest**
Category:	Computer vision
Academic year:	2024/25

Assignment:

1. Study image processing to enhance images of a scene captured by a camera, as well as appropriate machine image processing algorithms with emphasis on object detection and tracking from the enhanced images.
2. For your chosen sample application, design or select appropriate algorithms that will enhance the captured images of objects of interest and perform real-time object detection and tracking. For example, consider tracking of specific players in sports or tracking useful for security systems.
3. Select a suitable computing platform that has adequate computing power, as low input power as possible, and an accelerator for neural networks.
4. Attach a suitable image sensor with a lens to the computing unit and place everything into a suitable housing to create a prototype smart camera with embedded intelligence.
5. For your chosen application, design and implement a sample software in the smart camera that will enhance the images taken by the camera, detect and trace objects of interest in real time.
6. Test your implemented application on real video sequences.
7. Evaluate the results and discuss possible extensions of the project.

Literature:
According to the supervisor's instructions.

Requirements for the semestral defence:
Points 1-3 of the assignment.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Fučík Otto, doc. Dr. Ing.**
Head of Department:	Sekanina Lukáš, prof. Ing., Ph.D.
Beginning of work:	1.11.2024
Submission deadline:	21.5.2025
Approval date:	31.10.2024

Abstract

Our thesis focuses on low-powered real-time deep learning inference systems utilizing quantized neural networks. Specifically, we implement a multi-stage computer vision application that performs vehicle detection, registration plate detection, and registration plate recognition. The whole application is decomposed between a *Raspberry Pi 5* board combined with *Hailo-8 AI Accelerator* and *Raspberry Pi AI Camera* deep learning accelerators. We enclose the named hardware in a case equipped with lithium-ion rechargeable batteries, enabling the system to be operated hand-held. Furthermore, we focus on the correct image signal processing configuration combined with low-light and super-resolution deep learning image enhancements for improving the detection and reading results. Our application, on average, achieves 0.978 recognition accuracy, 0.965 detection precision, and 0.983 detection recall under multiple use cases and in various lighting conditions, including street lighting at night.

Abstrakt

Naše diplomová práce se zaměřuje na systémy hlubokého učení s nízkou spotřebou pracující v reálném čase, které využívají kvantizované neuronové sítě. Konkrétně implementujeme vícestupňovou aplikaci počítačového vidění, která provádí detekci vozidel, detekci registračních značek a jejich rozpoznávání. Celá aplikace je rozdělena mezi desku *Raspberry Pi 5* kombinovanou s akcelerátory strojového učení *Hailo-8 AI Accelerator* a *Raspberry Pi AI Camera*. Uvedený hardware je umístěn v pouzdře vybaveném dobíjecími lithium-iontovými bateriemi, což umožňuje jeho ruční užívání. Dále se zaměřujeme na správnou konfiguraci zpracování obrazového signálu v kombinaci s vylepšením obrazu pomocí hlubokého učení pro nízké osvětlení a malé rozlišení za účelem zlepšení výsledků detekce a rozpoznání registračních značek. Naše aplikace dosahuje průměrné přesnosti rozpoznání 0.978, přesnosti detekce 0.965 a citlivosti detekce 0.983 v různých scénářích použití a světelných podmínkách, včetně nočního pouličního osvětlení.

Keywords

smart camera, image processing, image enhancement, scene adaptation, machine learning, object detection, object tracking, optical character recognition, neural network quantization, low power processing, real-time processing, Hailo-8, Hailo-15, Raspberry Pi 5, Videology SCAiLX, UP Squared Pro, Raspberry Pi AI Camera

Klíčová slova

chytrá kamera, zpracování obrazu, vylepšení obrazu, adaptace scény, strojové učení, detekce objektů, sledování objektů, optické rozpoznávání znaků, quantizace neuronových sítí, nízkoenergetické zpracování, zpracování v reálném čase, Hailo-8, Hailo-15, Raspberry Pi 5, Videology SCAiLX, UP Squared Pro, Raspberry Pi AI Camera

Reference

MIHOLA, David. *Smart Camera for Tracking Objects of Interest*. Brno, 2025. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Dr. Ing. Otto Fučík

Smart Camera for Tracking Objects of Interest

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of doc. Dr. Ing. Otto Fučík. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
David Mihola
May 21, 2025

Acknowledgements

I thank my supervisor for valuable advice on hardware selection, guidance throughout software development, and help with structuring the thesis. I also acknowledge my mother's support during the testing phase and her encouragement during difficult moments.

Contents

List of Figures

List of Tables

List of abbreviations

2D	two-dimensional. 3, 11, 23, 24, 29
AI	artificial intelligence. 11, 12
API	application programming interface. 42
BCE	Binary Cross Entropy. 31, 64
CCE	Categorical Cross Entropy. 30, 31, 33, 34, 64
CER	Character Error Rate. 34
CISC	complex instruction set computing. 52
CNN	convolutional neural network. 3, 22, 24–26, 28, 29, 31, 32, 34, 36, 57, 80
COCO	Common Objects in Context. 30, 43, 44, 58, 63
CRNN	convolutional recurrent neural networks. 33
CTC	Connectionist Temporal Classification. 33
DCE-Net	Deep Curve Estimation Network. 25, 36, 67, 69, 80
DL	deep learning. 52
DMCNN-VD	Very Deep DeMosaicing Convolutional Neural Network. 26
DSLR	Digital Single-Lens Reflex. 26, 27
ESPCN	Efficient Sub-pixel Convolutional Neural Network. 24, 36, 66, 69, 80
ESRGAN	Enhanced Super Resolution Generative Adversarial Networks. 24
FCN	fully convolutional network. 34
FHD	full high definition. 24, 28, 50
fp32	32-bit floating point number. 37, 38
FPGA	field programmable gate array. 49, 52, 53
FPS	frames per second. 51
GAN	generative adversarial network. 25, 35

GB	gigabytes. 49, 50, 52–54
GFLOPS	giga floating-point operations per second. 49
GHz	gigahertz. 49, 50, 52, 53
GPU	graphical processing unit. 28, 31
HDR	high dynamic range. 3, 21, 69
HEF	Hailo Executable Format. 42
HOG	Histogram of Oriented Gradients. 28
HSI	hue, saturation, intensity. 19
I2C	Inter-Integrated Circuit. 50
ILSVRC	ImageNet Large Scale Visual Recognition Challenge. 31, 32
int4	4-bit integer number. 37, 39
int8	8-bit integer number. 37, 38
IoU	intersection over union. 31, 43, 44, 64
ISP	image signal processing. 11–20, 22, 25–28, 36, 51, 53, 60, 61, 80
JSON	JavaScript Object Notation. 59
k-NN	k-Nearest Neighbors. 28
LDA	Linear Discriminant Analysis. 28
LEC	Light-Enhancement Curve. 25
LPDM	Low-light Post-processing Diffusion Model. 25
MB	megabytes. 27
MIPI	Mobile Industry Processor Interface. 49
MobileViTv2	Mobile Vision Transformer version 2. 34, 58, 71, 81
MOS	Mean Opinion Score. 27
MP	megapixels. 26, 50, 51
MS-SSIM	Multi-Scale Structural Similarity Index Measure. 27
MSE	Mean Square Error. 35, 40
OCR	optical character recognition. 32–34, 57, 58, 61, 63, 65, 66, 70
PCA	Principal Component Analysis. 28
PSNR	Peak Signal-to-Noise Ratio. 22, 27

PTQ post-training quantization. 39, 40, 42, 47, 80

QAT quantization-aware training. 40–42, 44–47, 80

R-CNN Region-Based Convolutional Neural Network. 32
RAM random access memory. 28, 49, 50, 52–54
ReLU Rectified Linear Unit. 29, 38
RGB red, green, blue. 17, 18, 25, 26
RISC reduced instruction set Computer. 52

SD Secure Digital. 76
SIFT Scale-Invariant Feature Transform. 28, 65
SRCNN Super Resolution Convolutional Neural Network. 22
SSD single shot detector. 51
SURF Speeded-Up Robust Features. 28, 65
SVM support vector machine. 28, 32

TOPS tera operations per second. 49, 50

VDSR Very Deep Super Resolution. 22, 36
VGG Visual Geometry Group. 3, 31, 36

YOLO You Only Look Once. 3, 32, 36, 37, 43, 47, 58, 64, 70, 71, 80, 81

Chapter 1

Introduction

Today's world is increasingly driven by smart machines. Most of us have already heard the phrase artificial intelligence and even have tried using it, lately, in the most popular form of chatbots. Not many people, however, are familiar with how the so-called artificial intelligence (AI) is achieved. AI or deep learning, a term sometimes preferred in engineering circles, on the implementation level, most often means linear algebra operations like matrix multiplication or 2D convolution. These computations are highly demanding and require significant processing power, often resulting in high energy consumption.

Despite this, deep learning is becoming viable in a growing range of applications, thanks to advancements in computer chip design and manufacturing. Researchers have been interested in machine image understanding for decades, which has consequently led to the development of smart cameras for, e.g., detection of imperfectly manufactured products or registration plate recognition. Currently, this still often means that the camera transmits captured images to some more powerful computing unit that returns results on which it can capitalize. However, this approach does not scale economically and may not always be possible due to strict security or privacy constraints.

Therefore, there is a need for smart cameras that can operate independently without large data transfers and ideally be powered by batteries or through ambient energy harvesting. Special computer chips designed for low-power deep learning inference are emerging. These AI accelerators can execute state-of-the-art neural networks using techniques like quantization with minimal loss in inference quality compared to processing units, which require orders of magnitude larger input power. Throughout the work on this thesis, we have tested the *Hailo-8 AI Accelerator* and *Hailo-15 AI Vision Processor* from the Israeli firm *Hailo AI* as well as the *i.MX 8M Plus* processor integrated into the *Videology SCAiLX Development-Kit*, and the *Raspberry Pi AI Camera* AI accelerators. We finally selected and integrated together the Hailo-8 AI Accelerator, Raspberry Pi AI Camera, and Raspberry Pi 5 to demonstrate a low-power smart camera prototype powered by a conventional lithium-ion battery pack.

Furthermore, this thesis also focuses on image enhancement for better deep learning inference results. Our focus in this regard lies on real-time processing that achieves high frame rates. Therefore, we cannot use any computationally demanding traditional algorithms and must rely on configuring the available image signal processing (ISP) pipelines in the named devices as well as using the AI accelerators for running image enhancement models. Specifically, we focus on the correct ISP pipeline configuration to produce sharp images, which we couple with low-light and super-resolution deep learning image enhancements.

The rest of this work is structured in the following way. First, we describe basic camera controls and traditional stages of an ISP pipeline. We also look at how the traditional ISP stages can be replaced by deep learning models and common deep learning tasks performed on images. Second, we analyze quantization approaches for compiling full-precision floating point models to run on lower-precision integer-based AI accelerators. Here we also perform an experiment of our own, analyzing differently trained deep learning models compiled for the Hailo AI devices. The third chapter is focused on the assessment of the aforementioned hardware platforms in multiple categories, including image quality, deep learning capabilities, general computing performance, power consumption, implementation difficulty, and purchase price. We use these results to assign each platform a target area of operation. Fourth, we implement a complex computer vision application performing vehicle detection into registration plate detection into registration plate recognition, demonstrating the current capabilities of the Raspberry Pi 5 platform combined with the Hailo-8 AI Accelerator and Raspberry Pi AI Camera. Finally, we thoroughly test and evaluate the implemented application in multiple use cases, including an ablation study, summarize the achieved results, and suggest areas of future research.

Chapter 2

Image Signal Processing and Machine Image Processing

This chapter first briefly presents the most important camera settings. Then, we describe the stages of a traditional ISP pipeline, such as white balancing, demosaicing, or image enhancements, and their impact on the resulting images. Follows a section on how the traditional algorithmic ISP pipeline can be partially or fully replaced by deep learning approaches with a focus on super-resolution, low-light enhancements, joint demosaicking, denoising, and end-to-end sensor-to-image deep learning processing. Then, we present traditional, machine learning, and deep learning tasks performed on images or video and analyze their training data and hardware requirements. We especially focus on the development, training, and deployment of convolutional neural networks. Last, we explain several image-related deep learning tasks, including image classification, object detection, semantic segmentation, optical character recognition, and image generation, with specific examples of well-accepted solutions by the research community.

2.1 Camera Settings

In this thesis, we focus on *edge* cameras. Edge cameras are designed for small, often battery-powered, devices that provide low to medium image quality. These cameras typically offer only a limited set of controls. We describe three staple camera settings consisting of aperture, shutter speed, and analog gain that allow embedded applications to tune the brightness, sharpness, and graininess of captured scenes.

Aperture limits the amount of light passing through a lens onto a camera sensor [18]. It is realized through an adjustable opening of a circular shape placed inside the lens's body. Adjusting the aperture has two effects. Small values, i.e., the opening is fully or almost fully open, let in a lot of light onto the sensor, leading to brighter photos, which might cause overexposure in well-lit environments. It also causes the background not to be in focus by shortening the depth of field, creating a visually pleasing bokeh effect [18]. On the other hand, the opening will let less light onto the sensor with large aperture values, making the photos darker, which might cause washed-out images in dark environments. Lenses with large apertures have a large depth of field and also do not create the bokeh effect, which is welcomed for situations when the whole image must be in focus. Fixed lenses without adjustable apertures are preferred in embedded systems to reduce the number of

moving parts. Therefore, it is important to find the best compromise between focus, i.e., larger depth of field, and amount of light passing onto the sensor beforehand when selecting an embedded camera.

Shutter Speed determines how long the camera sensor is exposed to the incoming light during the image-capturing process [17]. The shutter used to be a physical device that moved in front of and away from the sensor. Most modern cameras, however, no longer have this sliding device, and the shutter is realized electronically. A fast shutter speed, therefore short exposure time, will keep even fast-moving objects in focus but will not let as much light onto the sensor, causing darker images. Conversely, slow shutter speeds, in other words, long exposure times, will allow the sensor to capture more light, resulting in brighter images. On the other hand, long exposure can cause motion blur, when objects that move significantly during the exposure will be blurred.

Analog Gain is performed on the hardware level, where the captured image is amplified in the analog domain before it is converted to a digital image [52]. Small analog gain values mean that there is almost no amplification applied to the captured signal. Thus, the image can appear dark if the captured scene is not well-lit. Larger analog gain values will artificially brighten the image, compensating for the lack of light. The tradeoff is that the captured imperfections, referred to as noise, are also amplified, consequently leading to grainy images [52].

2.2 Traditional Image Signal Processing Pipeline

The purpose of an ISP pipeline is to process acquired data by a sensor (digital camera, scanner, radar, etc.) into an image that is primarily displayed to users. Nowadays, ISP-processed images are increasingly more fed to computer vision algorithms, most often deep neural networks. Although these networks typically still operate on the ISP-processed images, we believe that future development will eventually lead to the removal of the ISP processing for these purposes, as it only transforms and potentially corrupts the captured data by applying algorithms optimized for the human visual system. The scope of this thesis is not to remove the ISP pipeline. Therefore, we still research some traditional ISP algorithms to gain a better understanding of the area. Traditional ISP pipelines often contain, not necessarily in the presented order, most of the following stages:

1. black level subtraction,
2. anti-aliasing and noise filtering,
3. defective pixel correction,
4. demosaicing,
5. gamma correction,
6. white balancing,
7. color correction,
8. lens corrections,

9. luminance and chrominance noise filtering,
10. image enhancements.

Another reason why the listed stages are still kept for neural network image processing is that they normalize raw data from different sensors to images of similar appearance, allowing the reuse of neural networks' designs and training data for various image-capturing hardware. Nevertheless, the only currently necessary stage is demosaicing to preserve the expected input dimensions of most neural networks. On the other hand, noise filtering stages and image enhancements are primarily used to improve the subjective image quality for human consumption. Thus, they might be omitted first, unless they are tuned to implement pre-processing before the deep learning inference.

Moreover, it is important to understand that these stages are almost never performed solely with software algorithms. Typically, the pipeline stages are mapped between the sensor/camera unit, performing, e.g., black level subtraction, specialized hardware tasked with compute-heavy stages like demosaicing, and software algorithms are often only deployed for image enhancements. Therefore, removing the ISP processing should save chip die real estate, decrease latency, and decrease power consumption, even if we consider that current deep neural networks might have to be even deeper in order to compensate for the algorithmic ISP transformations presented below.

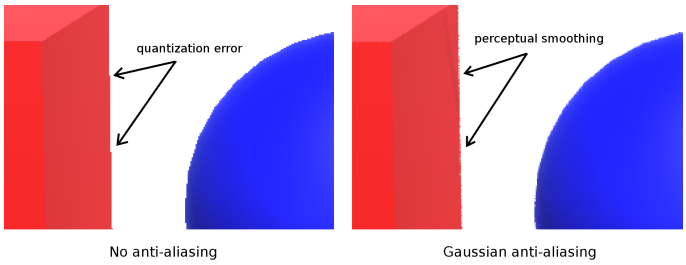
Black Level Subtraction [49] compensates for the baseline signal or residual output of an image sensor, referred to as the black level or dark current, which persists even when no light is present. Various physical and electronic factors, such as sensor design and operating conditions, cause the black level to be non-zero. Its value can also vary across pixels and color channels. Therefore, it is necessary to subtract this black level to represent the captured scene accurately.

The primary benefits of black-level subtraction include improving color accuracy, enhancing contrast, and maximizing dynamic range. Without it, color channels with different black level offsets may result in color tints, leading to inaccurate color representation. Correct black-level subtraction also ensures that true blacks are mapped accurately, preventing a washed-out appearance in darker scenes. Additionally, it helps in reducing fixed pattern noise, which can otherwise become noticeable in images, especially in low-light conditions.

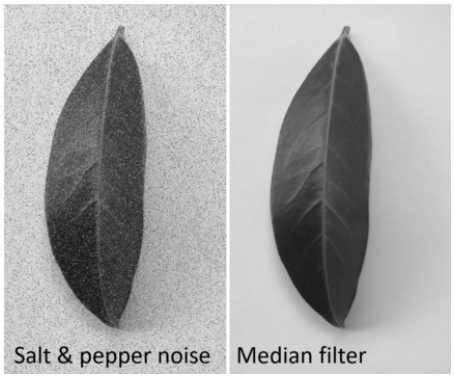
Not performing black-level subtraction can degrade image quality in various ways. It can lead to incorrect color representation, reduced contrast, and limited dynamic range, causing images to appear washed out or foggy. In some cases, fixed pattern noise may become more pronounced, further impacting image clarity. Furthermore, subsequent ISP stages, such as white balance and gamma correction, may suffer from complications, leading to more severe image quality issues.

Anti-Aliasing and Noise Filtering are two crucial techniques for preserving image quality and ensuring accurate signal representation. Anti-aliasing is primarily concerned with preventing visual distortions that arise when high-frequency image content is undersampled[1]. In digital imaging, signals, i.e., pixel values, are sampled at a specific resolution, and when this sampling frequency is insufficient for high-frequency image details, aliasing artifacts such as jagged edges, staircase artifacts near edges, or moiré patterns can appear [102]. Anti-aliasing techniques typically involve applying low-pass filters to smooth

[1]Note that anti-aliasing in this context does not correct violations of the Nyquist–Shannon sampling theorem. This must be done prior in a lens or camera sensor as described in [74].



(a) Example of Gaussian anti-aliasing
Image source:
https://github.com/bburrough/GaussianAntialiasing



(b) Example of noise filtering with median filter
Image source: https://www.futurelearn.com/info/courses/introduction-to-image-analysis-for-plant-phenotyping/0/steps/297750

Figure 2.1: Examples of anti-aliasing and noise filtering

the image by removing or reducing the high-frequency components. Common methods include Gaussian blurring [75], see the left part of Figure 2.1, or Lánczos interpolation [66].

On the other hand, noise filtering focuses on improving subjective image clarity by removing unwanted variations, such as sensor noise, environmental interference, or quantization errors. Various filters are used in noise reduction, such as Gaussian filter [32], box (mean) and median filters[2] [32], see the right part of Figure 2.1, and more advanced techniques like denoising wavelet domain [70] or non-local means [13]. The challenge with noise filtering is to remove as much noise as possible without degrading the essential details of the image. Adaptive noise filters that can adjust their behavior based on local image characteristics are often used to maintain a balance between noise removal and image sharpness.

Defective Pixel Correction must be an early stage in the ISP pipeline that compensates for flawed pixels on the image sensor that may produce incorrect values due to manufacturing imperfections or damage during the sensor's lifetime. Defective pixels, commonly referred to as *hot*, *cold/dead*, or *stuck* pixels, are individual pixels that display unusual brightness or color regardless of the scene's actual lighting. Hot pixels remain bright even in dark conditions, while cold pixels stay dark despite illumination. Stuck pixels tend to display a particular color regardless of actual scene data. These defects, if not corrected, can lead to salt and pepper noise in the final image, degrading overall image quality and visual accuracy, especially in low- or high-light conditions where hot/cold pixels are more prominent. [48]

There are two approaches for defective pixel correction. The first, static, approach corrects only pixels identified as defective during the calibration phase of a sensor after manufacturing. This approach requires fewer computational resources but cannot compensate for newly appearing defective pixels during the sensor's lifetime. The second, dynamic, approach then deals with this issue by periodically scanning the pixel map and marking

[2]Box filters reduce additive Gaussian noise well, while median filters are better for reduction of impulse – salt and pepper noise

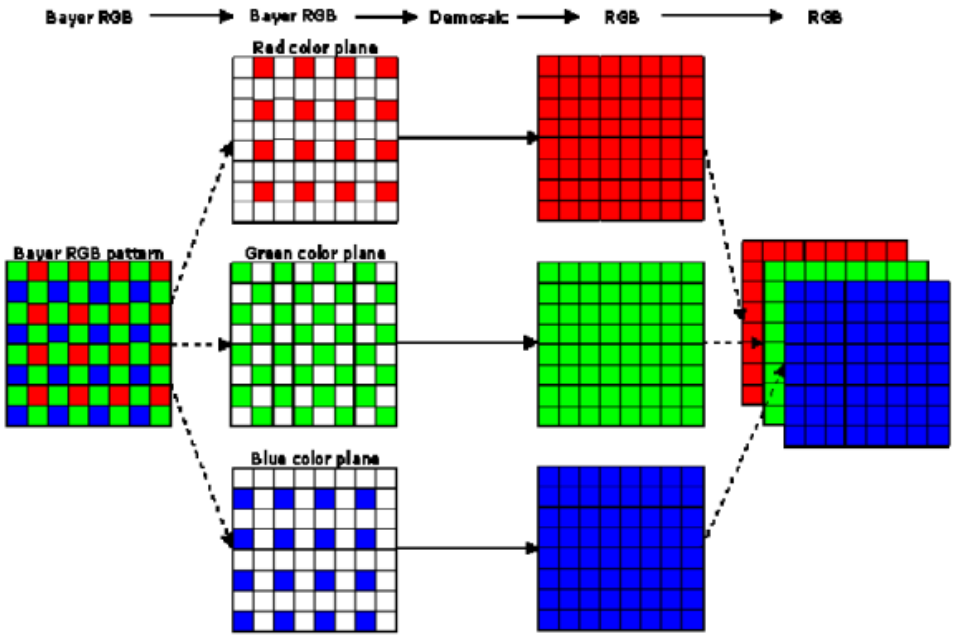


Figure 2.2: Demosaicing
The image was authored by Tsengelidis Savvas in [86].

new pixels as defective if their values consistently differ from the surrounding pixel values. The correction part is the same for both approaches. The defective pixel's value is replaced with an estimate calculated based on the surrounding pixels using some non-trivial robust interpolation technique, which is, e.g., based on the image's feature direction estimates. [25]

Demosaicing [57] is the process in the ISP pipeline that reconstructs full-color images from the incomplete color data captured by a camera sensor. Most modern digital cameras use a color filter array, often a Bayer filter, where each pixel on the sensor captures only one of the three primary colors: red (R), green (G), or blue (B). This arrangement leaves each pixel with only partial color information, meaning that the sensor does not capture full RGB data for each pixel location. Demosaicing algorithms interpolate the missing color information by analyzing the color values of neighboring pixels, producing a complete and visually accurate RGB image, see Figure 2.2.

One of the traditional methods for demosaicing is bilinear interpolation, where missing color values are estimated by averaging neighboring pixels. However, more advanced techniques leverage both intra- and inter-channel correlations to improve the estimation. Sequential demosaicing approaches start by interpolating the green (luminance) channel, which is less aliased, to guide the reconstruction of red and blue (chrominance) channels. These methods include spatial-domain techniques like edge-directed interpolation, where local edge direction adapts the interpolation. Advanced algorithms use adaptive filtering with filters inspired by the human visual system to further refine the luminance value. Chrominance values are then reconstructed using a weighted sum of the neighboring luminance values, with the weights being selected based on the horizontal and vertical gradients provided by edge indicators.

Common artifacts that can appear on not well-demosaicked images are false colors and zipper artifacts. The prior is caused by wrong color interpolation, especially near edges

Figure 2.3: Example of under- and over-gamma-corrected image
The image was authored by Adrian Rosebrock in [84].

or high-contrast transitions. The latter is induced by inaccurate interpolation near sharp edges.

Gamma Correction is a step in the ISP pipeline that adjusts the brightness and contrast of an image to account for the nonlinear response of both display devices and human vision. Images captured by sensors generally represent light intensity in a linear fashion, but human perception of brightness is nonlinear, meaning we are more sensitive to changes in darker tones than in brighter ones [14]. Similarly, most display devices, such as monitors and televisions, are designed with a gamma curve approximating this nonlinear response, leading to a mismatch between the raw sensor data and how the image should be displayed. Gamma correction compensates for this by applying a mathematical transformation to delinearize the image data, ensuring that it appears natural to the human eye when viewed on a standard display [101].

Gamma correction is typically expressed by applying a power-law function to each pixel's intensity value, $I_{out} = I_{in}^{\gamma}$, where γ is a constant typically around 2.2 for most displays and I is normalized intensity of a color channel to the range of $\langle 0, 1 \rangle$ [88]. This transformation compresses the dynamic range of brighter areas while expanding the range in darker regions, aligning the image with how humans perceive light. The correction ensures that mid-tones and shadows are rendered accurately, preventing the image from appearing either too washed out or overly dark. See an example of overly gamma-corrected images in Figure 2.3.

White Balancing is a process in the ISP pipeline that ensures accurate color reproduction by correcting color temperature shifts in images. Different light sources, such as sunlight, shade, tungsten, or fluorescent lighting, emit varying spectra of light, which can cause a color cast, bluish, reddish, or greenish, onto the image [2]. White balancing compensates for this by adjusting the RGB channels so that objects that are perceived as white under neutral lighting conditions are rendered correctly, regardless of the light source [100].

The white balance process typically relies on algorithms that estimate the scene's illuminant, such as the gray-world assumption [8], which assumes that the average color of a scene is neutral gray, or more advanced methods like deep learning-based algorithms that can better adapt to complex lighting environments [3]. Once the light source is identified,

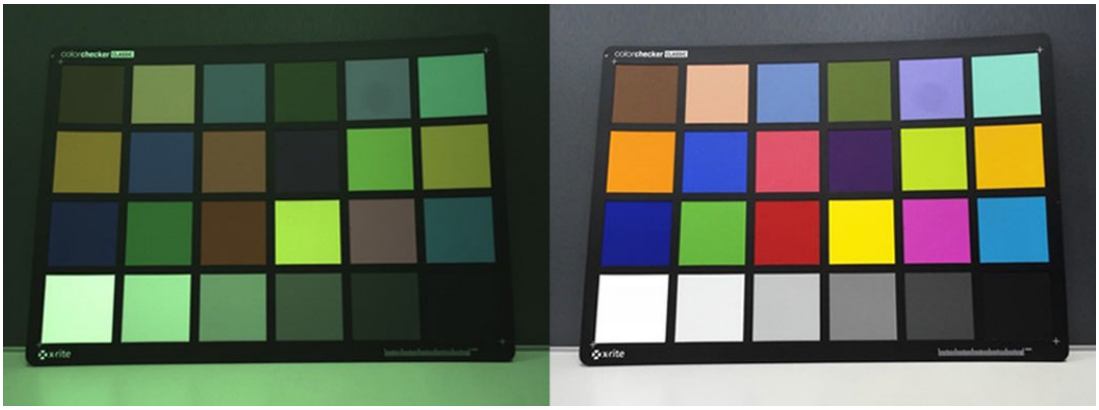


Figure 2.4: Example of color-corrected image
The sub-images are from left to right: unprocessed and color-corrected. Image source:
https://www.technexion.com/resources/what-is-a-camera-isp-what-are-its-functions/

the ISP adjusts the gain of each color channel to neutralize any color bias. This is crucial for ensuring that colors in the image appear as they would to the human eye, preserving natural skin tones and object colors.

Color Correction [76] is an important stage in the ISP pipeline that adjusts both the tonal range and color balance of an image to ensure accuracy and aesthetic quality. Initially, problems involving the tonal range of an image[3] must be addressed before color balancing can take place. Tonal correction ensures that the image's brightness and contrast are optimized, distributing the color intensities evenly between highlights and shadows. This process helps to maximize detail and to avoid issues like over- or under-saturated colors. Tonal adjustments are often carried out interactively, using transformations like S-shaped contrast curves to enhance mid-tones while preserving highlights and shadows.

Once the tonal issues are resolved, color correction focuses on fixing any imbalances in the overall color distribution. Color balancing involves adjusting the proportions of the red, green, and blue channels to ensure accurate representation, particularly in areas that should appear neutral, like white or gray regions. Since every change in one color channel affects the perception of surrounding colors, adjustments must be made carefully. In many cases, skin tones are used as a reference for manual adjustments, as human eyes are particularly sensitive to the natural appearance of skin color.

In addition to manual transformations, automated techniques like histogram equalization can be applied to color images to distribute intensity values uniformly across an image. This is often done in the HSI color space, where only the intensity component is adjusted to improve brightness without altering hue or saturation. While this process can significantly improve overall detail and brightness, it may also affect color vibrancy. Therefore, further adjustments to the saturation component might be needed to maintain the original appearance of the image, particularly when working with vibrant content. An example of an unprocessed and color-corrected image is shown in Figure 2.4.

Lens Corrections are a stage in the ISP pipeline aimed at compensating for the optical imperfections of camera lenses, ensuring more geometrically accurate and visually appealing

[3]Tonal range defines levels between the darkest and lightest points in the image

images. As light passes through the lens elements, it undergoes distortion, especially near the edges of the frame. Common types of distortion include barrel distortion, where straight lines bow outward, and pincushion distortion, where lines bend inward [16]. ISP algorithms, such as described in [16, 85, 64], apply geometric transformations to the image to counteract these effects by mapping pixels to corrected positions that restore straight lines to their true geometry.

In addition to geometric distortion, other optical aberrations, such as vignetting and chromatic aberration, are also corrected in this stage. Vignetting occurs when the periphery of the image appears darker than the center due to the lens's inability to transmit uniform light across the sensor [47]. The ISP pipeline compensates by brightening the darker regions, making the image illumination uniform. Chromatic aberration, caused by the failure of a lens to focus all colors at the same point, can manifest as color fringing near the edges of objects [60]. By adjusting the alignment of different color channels based on the lens profile, this artifact can be reduced, resulting in sharper and more color-accurate images.

Luminance and Chrominance Noise Filtering are techniques within the ISP pipeline aimed at selectively reducing noise in the luminance (brightness) and chrominance (color) components of an image [7, 96]. Luminance noise, often perceived as graininess, affects the brightness values of pixels and is typically more noticeable in darker areas of an image. Since the human eye is highly sensitive to changes in brightness [73], it is crucial to filter luminance noise carefully to maintain image detail. Various methods, such as adaptive filters, are employed to remove this noise while preserving fine structures like edges. These filters analyze local image characteristics, applying stronger noise reduction in flatter, textureless areas and softer filtering in regions with intricate details, ensuring minimal loss of sharpness [36].

In contrast, chrominance noise, which affects the color information in an image, is often less perceptible to the human eye due to our lower sensitivity to color variations [73]. This makes applying more aggressive noise filtering to the chrominance components (Cb and Cr) possible without visibly degrading image quality. Methods like wavelet-based filtering [30] or non-local means filtering [65] can be applied to chrominance and luminance, leveraging spatial correlation to enhance noise suppression. The challenge lies in balancing the level of filtering to ensure that color noise is reduced effectively while maintaining natural and vivid color transitions in the final image.

Image Enhancements substantially impact the subjective visual quality of images processed through the ISP pipeline. Enhancing sharpness and related acutance[4], see Figure 2.5, improves the clarity and detail of an image by emphasizing edges and fine textures. This is typically achieved through fixed-neighborhood methods like unsharp masking, which enhances high-frequency details of an image by subtracting its blurred version, or by adaptive image sharpening methods that use edge-detection algorithms to focus primarily on sharpening edges. While enhancing sharpness can significantly improve subjective image clarity, excessive sharpening can introduce artifacts, such as halos around objects, so it must be applied judiciously to maintain a natural appearance. [21]

Contrast adjustment is another key enhancement that affects the separation between the lightest and darkest parts of an image, directly influencing its overall dynamic range [44]. Increasing contrast creates more noticeable differences between highlights and shadows,

[4] Acutance is a subjective perception of sharpness that is related to the edge contrast of an image

Figure 2.5: Example of acutance variation
The sub-images are from left to right: unprocessed, with slightly increased acutance, with strongly increased acutance. The image was authored by Dorothea Lange and is available at: https://en.wikipedia.org/wiki/Acutance#/media/File:Accutance_example.png

Figure 2.6: Combination of images taken with different exposure times into an HDR image
The sub-images are taken from left to right with increasing shutter speed, i.e., with less light captured by the sensor. The image was authored by Peter Thoeny and is available at: https://www.flickr.com/photos/peterthoeny/13345987993

adding depth and dimension to the image. However, too much contrast can result in lost detail in bright or dark areas, as these are going to be clamped into the highest or lowest possible pixel values [9], so careful balancing is necessary to preserve information across the tonal range. Techniques like high dynamic range (HDR) imaging can be employed to overcome this limitation by combining multiple exposures, see Figure 2.6, to capture a wider range of brightness levels [9]. HDR ensures that both shadows and highlights are preserved, enhancing image realism, especially in scenes with significant lighting variation.

Saturation adjustment, which controls the intensity of colors in an image, is often used to make colors more vivid or to achieve specific artistic effects [103]. Enhancing saturation can make an image appear more vibrant, but oversaturation may lead to unnatural colors and a loss of subtle color variations, especially in areas like skin tones or natural landscapes.

To avoid this, vibrance adjustment is frequently employed alongside saturation. Vibrance selectively increases the intensity of less-saturated colors while protecting skin tones and other already vibrant areas, creating a more balanced and natural enhancement [11].

2.3 Image Signal Processing Pipeline with Deep Learning

In recent years, the traditional ISP pipeline has been slowly replaced by deep learning algorithms, more precisely by convolutional neural networks. The deep learning ISP ranges from specific post-processing tasks, such as resolution enhancements or low-light enhancements, to replacement of some ISP stages like demosaicing and denoising to replacing the whole traditional ISP pipeline[5]. All of these approaches are mostly oriented to improve the visual quality of the produced images. Nevertheless, subsequent computer vision algorithms can also benefit, e.g., from low-light-enhanced images, if they are designed for well-lit conditions.

The following sections describe each of the named deep learning ISP approaches with examples of neural networks specialized for the presented tasks.

2.3.1 Image Enhancements with Deep Learning Post-Processing

This section explains resolution enhancements (super-resolution) and low-light enhancements, two common deep learning image post-processing tasks.

Super-Resolution is a task where low-resolution images are scaled up into higher-resolution images of the best possible quality. The super-resolution can be divided into two approaches.

First, the approach, where the low-resolution image is initially scaled up to the desired size algorithmically, e.g., with the bicubic upscale algorithm, and then this image is passed through a convolutional neural network (CNN) which is expected to improve it.

The Super-Resolution Convolutional Neural Network (SRCNN), one of the first successful super-resolution deep learning methods, presented by Dong et al. in [23], does exactly the described. They use only a three-layer CNN. Dong et al. also tried deeper CNN architectures but reported difficulties with convergence during training and did not see any improvement in the measured peak signal-to-noise ratio (PSNR) metric. The issue with convergence has been solved since then with the introduction of residual connections by He et al. in [34] and will be briefly touched on later.

Another example of the upscale first approach is the Very Deep Super Resolution (VDSR) CNN developed by Kim et al. in [45]. They overcome the training instability seen in the SRCNN by introducing a residual connection of the input image with the output of the deep CNN and by performing adjustable gradient clipping. Their approach enables them to construct a 20-layer network while significantly reducing the training time and outperforming the SRCNN in the PSNR metric. Kim et al. also argue that their network is able to perform arbitrary upscaling given by the selected size of the input image, which is an advantage of the approach, where images are first scaled algorithmically.

Second, the approach, where the low-resolution image is passed through a CNN as is, and the CNN contains special upscaling layers to produce a higher resolution output. The

[5]The stages that are not performed by the sensor/camera itself.

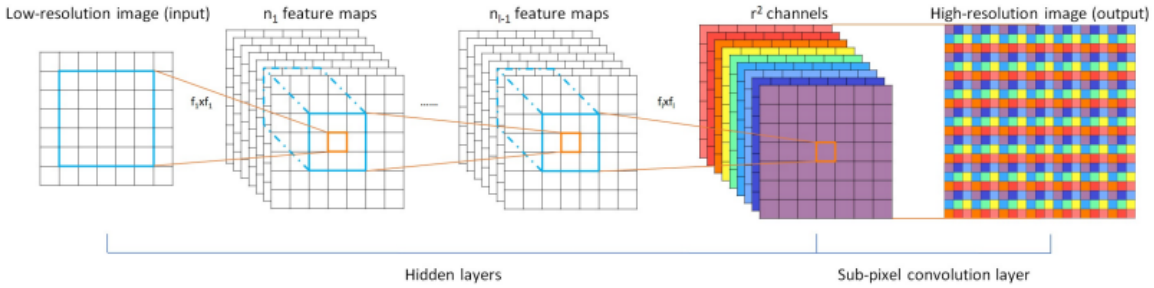
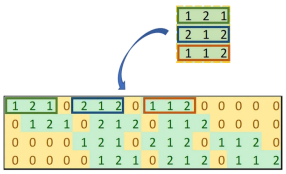
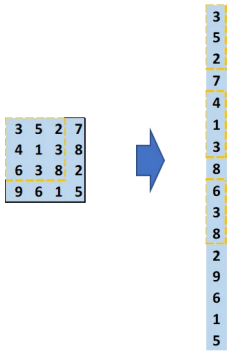


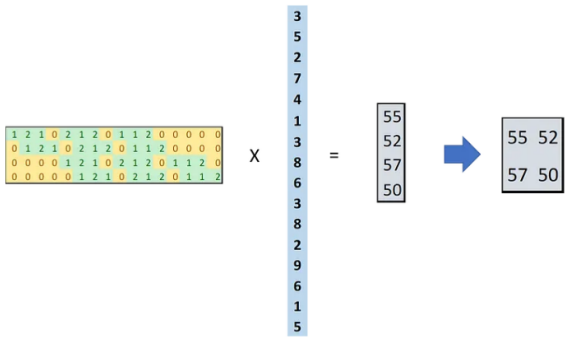
Figure 2.7: Neural network using a sub-pixel convolution layer
The upscaling factor of this network is r (3), which requires r^2 (9) input channels to the sub-pixel convolution layer. The source of the image is [90], where Shi et al. introduced this layer.



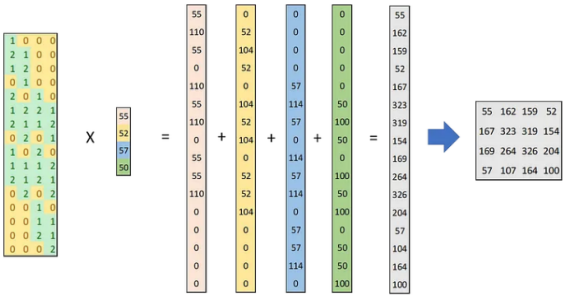
(a) 3x3 convolution kernel converted to a zero-padded convolution matrix



(b) 4x4 image patch converted to a vector



(c) Zero-padded convolution matrix by image patch vector multiplication realizing 2D convolution



(d) Transposed zero-padded convolution matrix by vectorized 2x2 image patch matrix multiplication realizing 2D transposed convolution

Figure 2.8: Sequence of images explaining 2D convolution realized by vector-matrix multiplication and the progression to 2D transpose convolution and its realization
The images were authored by Kuan Wei in [99].

main advantage of this approach is better computational efficiency since the input size does not scale quadratically with the scaling factor, as it does with the upscale first approach.

The specialized layers are a sub-pixel convolution layer, which rearranges a higher number of lower-resolution feature maps into a lower number of higher-resolution maps by spatially shuffling and stacking the channels [90], see Figure 2.7, and a transpose convolution layer, which performs a matrix by vector multiplication, where the matrix is a transposed matrix of a zero-padded convolution matrix for 2D convolution [99], see Figure 2.8. An ex-

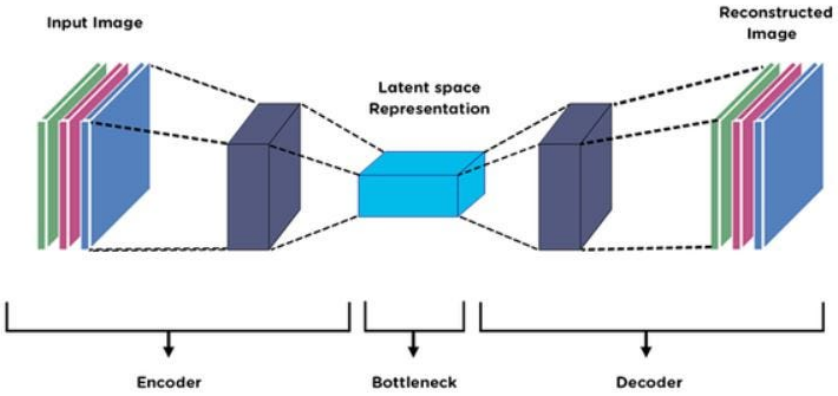


Figure 2.9: Autoencoder architecture
The image was authored by Ahmadsabry in [4].

ample CNN using the prior upscaling layer is the Efficient Sub-pixel Convolutional Neural Network (ESPCN) presented by Shi et al. in [90]. They were the first to develop a CNN allowing to up-scale FHD video by a factor of 4 in real time with state-of-the-art performance at the time of publishing. The later upscaling layer is then used, e.g., in the WaveMixSR neural network introduced by Jeevan et al. in [41]. Their network takes images converted to the YCbCr color space and leverages 2D-discrete wavelet transform to achieve state-of-the-art results with an architecture that is more computationally efficient and requires less training data than equally- or worse-performing transformer-based counterparts.

There are, however, also other approaches to upscale images during inference. For example, the Enhanced Super-Resolution Generative Adversarial Networks (ESRGAN) developed by Wang et al. in [98] uses algorithmic upscaling layers interpolating nearest pixels followed by a traditional 2D convolutional layer.

Low-Light Enhancement is a task that improves the visual effects of images captured in poor lighting conditions for subsequent processing. Its main goal is to enhance the low brightness, low contrast, and color distortions while suppressing noise in such images [95].

The default idea of how to denoise and improve image quality, even to this day, is to use autoencoder-based networks [95]. Autoencoder is a neural network that can be trained in an unsupervised manner thanks to its clever architecture, composed of the encoder and decoder components, see Figure 2.9. These components can be separated and used, e.g., for compression, generation of embeddings, etc. The encoder and decoder components are, however, kept connected for the task of image enhancements while the training process changes. It is no longer an unsupervised task. The inputs to the autoencoder must be noisy images, i.e., images captured in low-light conditions, and the ground truths for training must be enhanced versions of the input images. The pioneering work on this topic that leverages the autoencoder architecture was published by Lore et al. in [62].

Taking images in pairs of low-lit and well-lit scenes is challenging. Such training datasets are usually created artificially by applying low-light effects to the well-lit images. This means that the training data do not perfectly reflect reality. Consequently, models trained on this data cannot learn all the intricacies of real low-lit images. Therefore, they will struggle to generalize well. As a result, Jiang et al. in [43] came up with a solution based

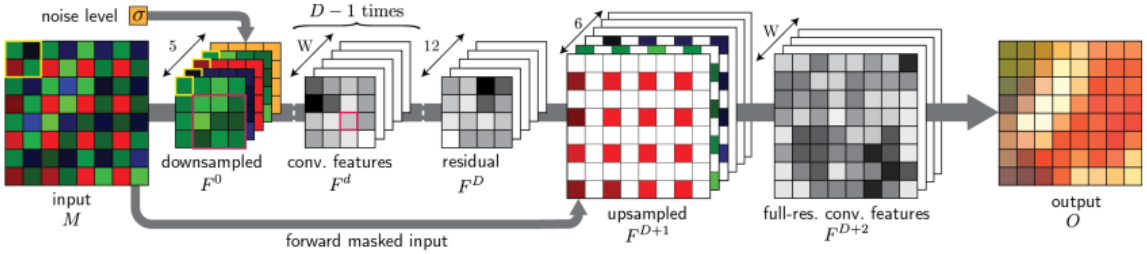


Figure 2.10: Architecture of joint demosaicking and denoising CNN
The image was authored by Gharbi et al. in [29].

on a different neural network architecture, the Generative Adversarial Network (GAN). They named their network EnlightenGAN and were able to reach state-of-the-art performance without using paired training data. GANs are generative neural networks. Their architecture is based on two networks, a generator and a discriminator. In simple terms, the two networks compete against each other. The generator is trained to generate outputs, enhanced low-light images in this case, while the discriminator is trained to classify whether an image comes from the generator or it is a real image taken under well-lit conditions. Another approach that does not require paired training data was presented by Guo et al. in [33]. Their Deep Curve Estimation Network (DCE-Net) is trained to predict parameters of Light-Enhancement curves (LECs). The LEC is a quadratic curve with a single parameter α. The DCE-Net outputs eight such parameters for each pixel of the input image. The image ($\mathbf{I}$) is then enhanced by an iterative application of the LEC using Equation (2.1), where all of the operands are pixel-wise.

$$\mathbf{I_{n+1}} = \mathbf{I_n} + (\mathbf{I_n^2} - \mathbf{I_n}) \cdot \alpha_\mathbf{n} \tag{2.1}$$

A different interesting approach was developed by Panagiotou et al. in [71]. They use a diffusion model named the Low-light Post-processing Diffusion Model (LPDM) to further improve images already brightened by a low-light enhancement network, such as the EnlightenGAN. Diffusion models are, similarly to GANs, generative models mostly known for generating images given a text prompt. They are based on starting with Gaussian noise and iteratively removing it until a clear image is reached. This process can be guided by some other information, e.g., an embedding of a text prompt. Since images enhanced by low-light enhancement networks can still be noisy, diffusion models like the LPDM are great candidates to further improve the quality of these enhancements. The downside of the diffusion models is the iterative, computationally expensive, denoising process. Panagiotou et al., however, claim that they can estimate and remove any noise in a single pass through their LPDM.

2.3.2 Image Signal Processing Pipeline with Deep Learning

It seems that the most popular stages of the traditional ISP pipeline to be deep learning accelerated are denoising (noise filtering) and demosaicing. As discussed above, denoising aims to remove unwanted artifacts, and demosaicing tries to infer full RGB channels from incomplete data. Removing anything not exactly specified and generating something based on incomplete information is difficult to achieve algorithmically, while neural networks typically excel at such tasks.

Gharbi et al. were among the first to introduce joint demosaicking and denoising deep learning CNN in [29], see Figure 2.10. The inputs to their network are a Bayer image

(single channel of combined red, green, and blue pixels) and a noise estimate. The noise estimate is especially important as the amount of noise in images changes given lighting conditions or ISO setting of a camera [29]. The final RGB output is an affine combination of a set of feature maps produced by their CNN. Moreover, their contribution lies in gathering a dataset of difficult patches from online photo collections. They use this dataset to train their CNN model and achieve state-of-the-art performance, beating all previous approaches in numerous metrics.

The joint demosaicking and denoising problem was also addressed by Kokkinos et al. in [46]. They use an iterative approach with a residual CNN named ResDNet, whose main inputs are a color filter array and an estimated noise. Counterintuitively, their iterative approach that performs multiple passes through the ResDNet is significantly faster than other solutions available at that time. This is because the iterative approach allows their network to have many fewer parameters and, therefore, be much faster. Another advantage of their smaller network is that it can be trained on less data and still achieve state-of-the-art performance at the time of publishing.

Last, we look at deep learning demosaicking performed on edge devices. Ramakrishnan et al. focused on searching for the best viable neural network for edge computing using a Pareto front[6] between negative color peak signal-to-noise ratio as the loss and the number of parameters as the neural network complexity in [78]. They used the Very Deep DeMosaicing Convolutional Neural Network DMCNN-VD state-of-the-art neural network architecture at the time of their experimentation, published by Syu et al. in [94] as their reference. They applied several modifications to this network, e.g., changing the number of filters, changing the number of blocks, and replacing traditional convolutional layers with depthwise separable convolutions. Their search was successful in finding numerous architectures with fewer trainable parameters that outperformed the reference neural network.

2.3.3 End-to-End Deep Learning Image Signal Processing Pipeline

End-to-end deep learning ISP pipeline in this context means the replacement of all traditional ISP stages presented in Section 2.2 with deep learning processing, if they are not performed by the sensor/camera itself. This is becoming a popular trend[7] since deep learning algorithms can produce not only better results but, thanks to their specific computation requirements, are also faster and easier to hardware-accelerate.

Ignatov et al. were probably the first to show that the end-to-end deep learning ISP pipeline is a viable solution in [38]. They trained an inverted pyramid CNN named pyNET to process Bayer images into high-quality RGB ones. Their network architecture is based on multiple scales, i.e., parallel paths in the graph of the CNN, see Figure 2.11. Each scale has a different receptive field of the input image and, therefore, processes different characteristics of it. Scales with smaller receptive fields are supposed to process local elements, e.g., focusing on texture enhancements or on removing noise, while scales with large receptive fields should evaluate and improve global information, such as the global color and overall brightness of the image. The training was performed on Bayer images taken by a Huawei P20 phone with a 12.3 MP Sony Exmor IMX380 sensor as inputs and RGB images taken and processed by a Canon 5D Mark IV DSLR professional camera

[6] Pareto front is a set of solutions in a multi-objective optimization problem where no objective can be improved without worsening at least one other objective, representing the trade-offs between competing goals.

[7] All major phone manufacturers are increasingly adding deep learning into their ISP pipelines.

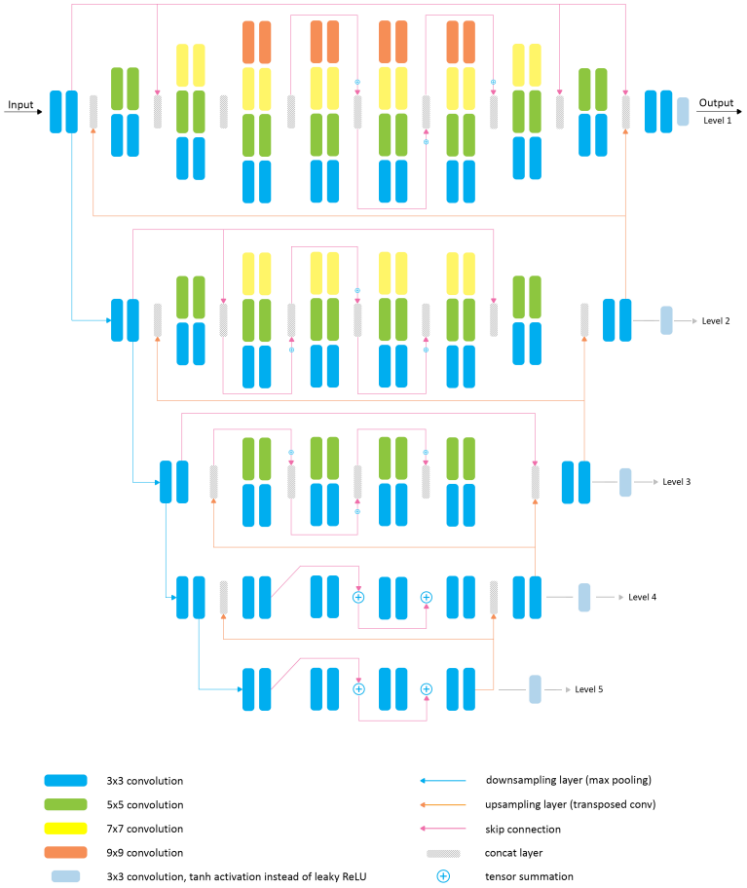


Figure 2.11: PyNET architecture
The image was authored by Ignatov et al. in [38].

as the ground truths. Ignatov et al. conducted both a quantitative evaluation and a user study. The quantitative evaluation study concluded that their model outperformed all other approaches that they compared their model against in the PSNR and MS-SSIM metrics. More importantly, the user study showed that people prefer the images processed by their model over the built-in ISP pipeline of the Huawei P20 phone in the MOS metric. The built-in ISP pipeline got a score of 2.56 against a score of 2.77 achieved by their end-to-end deep learning approach, where a score of 2 means clearly worse quality and a score of 3 comparable image quality to the images from the Canon 5D Mark IV DSLR.

The same authors, Ignatov et al., also presented a mobile version of the pyNET called PyNET-V2 Mobile in [39]. Their goal was to make a similar end-to-end deep learning-driven ISP pipeline that would still perform well in terms of image quality, but be memory efficient, have only operations and layers that can be accelerated on mobile devices, and have acceptable latency. They adjusted the original neural network, e.g., by reducing the number of processed scales from 5 to 3 or by halving the number of convolutional filters and reducing their size to a maximum of 3x3. To regain some of the performance lost by these changes, the network was equipped with additional processing blocks named channel attention module block, or, in short, a CAM block, and spatial attention module block, shortened to SAM block. Refer to their work in [39] for a description of these processing blocks. These modifications mean that the PyNET-V2 Mobile is only 3.6 MB in size and

runs with considerably low RAM usage when processing FHD images. Its performance is on par with the pyNET neural network, while its latency is 274 ms, which results in about 47 times faster inference when running on a MediaTek Dimensity 1000+ GPU.

In conclusion, Ignatov et al. prove that end-to-end deep learning ISP pipelines are possible on both desktop processing environments with the pyNET neural network and also on mobile devices with the PyNET-V2 Mobile neural network. In both cases, not only is the inference performance acceptable, but more importantly, the visual results are comparable to or surpass traditional approaches.

2.4 Computer Vision Algorithms

Computer vision algorithms can be divided into traditional (algorithmic), machine learning, and deep learning approaches. We will mainly focus on deep learning image processing, i.e., CNN-driven processing. Some traditional and machine learning algorithms are only listed below for completeness.

2.4.1 Traditional Computer Vision Algorithms

- Scale-Invariant Feature Transform (SIFT) is a computer vision algorithm for the detection and matching of local features invariant to image scale developed by David G. Lowe in [63].
- Speeded-Up Robust Features (SURF) rotation-invariant descriptor and detector based on Haar-like features, partially inspired by SIFT, on which it improves in both computational efficiency and robustness. Bay et al. published this algorithm in [10].
- Histogram of Oriented Gradients (HOG) is an algorithm used to extract features from an image that can be further processed by machine learning algorithms, like Support Vector Machines (SVMs), typically for human detection. The algorithm was developed by Navneet Dalal and Bill Triggs in [20].

2.4.2 Machine Learning Computer Vision Algorithms

- Linear Discriminant Analysis (LDA) for feature extraction combined with machine learning classifiers like SVM or k-Nearest Neighbors (K-NN) for image classification.
- Principal Component Analysis (PCA) for feature extraction combined with cosine distance or K-NN for retrieval of similar images.
- AdaBoost is an algorithm that combines multiple weak learners based on their performance into an ensemble that can be used, e.g., for image classification. The algorithm was published by Freund et al. in [28].

2.4.3 Deep Learning Computer Vision Algorithms

Deep learning computer vision tasks can be divided among others include the following categories: image classification, object detection, optical character recognition, semantic segmentation, and image generation. These five named categories are described in the sections below. But first, we look at how neural networks work and how they are trained in general.

The basic architecture of any neural network typically includes three types of layers. First, layers with trainable parameters. These are most often 2D convolutional layers for image processing. Second, an activation function that follows each trainable layer. Activation functions allow the networks to learn nonlinear patterns and dependencies. Nowadays, the most popular activation function for hidden layers is the Rectified Linear Unit (ReLU). Activation functions of the output layers are chosen based on the performed task and will be mentioned below. Third, layers that improve the training of neural networks, such as the Batch Normalization layer, and layers that are supposed to reduce the issue of overfitting to the training data, which is, e.g., the Dropout layer. Overall, neural networks can be described as functions, although graph-like visualization is preferred for complex networks. Such function for the simplest CNN with a single 2D convolutional layer followed by a fully connected layer is expressed as shown in Equation (2.2), where $\mathbf{W_1}$ are learned weights of the convolutional filter, $\mathbf{W_2}$ are learned weights of the fully connected layer, $\mathbf{b_1}$ and $\mathbf{b_2}$ are learned biases for the respective layers, ϕ_1 and ϕ_2 are activation functions, $\mathbf{X}$ is the input image, and $\mathbf{y}$ is the output of the network.

$$\mathbf{y} = \phi_2(\mathbf{W_2} \times \phi_1(Conv2D(\mathbf{W_1}, \mathbf{X}) + \mathbf{b_1}) + \mathbf{b_2}) \tag{2.2}$$

Typically, a model of some neural network is created by randomly initializing the weights[8] and biases. The training of the model is based on minimizing the difference between its predictions and the ground truth[9]. It is realized by adjustments of the model's weights and biases through an algorithm called back-propagation. This algorithm was first proposed for the purpose of training neural networks by Yann LeCun in [54]. The difference between the model's output and the ground truth is defined by a loss function that is specific to a given problem solved by the model[10]. This means that the goal is to minimize the value of the loss function with respect to the model's output. Loss functions are, in general, convex, and models represent differentiable functions. Therefore, finding their minimum is equivalent to finding parameters for which their first derivative with respect to the parameters equates to zero[11]. This cannot be solved analytically and must be done iteratively by computing the derivative for a given output. The sign of the computed derivative indicates the slope direction of the loss function, and the magnitude of the derivative indicates its steepness. Since the goal is to find a zero slope, the model's output must be adjusted in the opposite direction of the slope. Adjusting the model's output can only be done by changing its weights and biases. Consequently, the goal is to minimize the loss function with respect to the weights and biases. Neural networks typically have multi-dimensional outputs. Hence, partial derivatives must be computed to form a gradient. As denoted above, neural networks are basically composite functions. This allows the chain rule of differentiation to be applied. Equation (2.3) denotes how to derive the loss function with respect to the $\mathbf{W_2}$ weights of the exemplar model denoted by Equation (2.2), where $L(\cdot)$ is a loss function, gt is the ground truth labels, $\mathbf{o_2} = \mathbf{W_2} \times \mathbf{o_1} + \mathbf{b_2}$, and $\mathbf{o_1} = \phi_1(Conv2D(\mathbf{W_1}, \mathbf{X}) + \mathbf{b_1})$. This example illustrates how the gradient of the loss is propagated back through the network.

$$\begin{aligned} \frac{\partial L(\mathbf{y}, gt)}{\partial \mathbf{W_2}} &= \frac{\partial L(\mathbf{y}, gt)}{\partial \mathbf{y}} \cdot \frac{\partial \phi_2(\mathbf{o_2})}{\partial \mathbf{W_2}} \\ &= \frac{\partial L(\mathbf{y}, gt)}{\partial \mathbf{y}} \cdot \frac{\partial \phi_2(\mathbf{o_2})}{\partial \mathbf{o_2}} \cdot \frac{\partial (\mathbf{W_2} \times \mathbf{o_1} + \mathbf{b_2})}{\partial \mathbf{W_2}} \end{aligned} \tag{2.3}$$

[8] Xavier (Glorot) and He (Kaiming) weight initialization techniques are the most popular.
[9] This type of training is called supervised learning.
[10] Some models can even be trained using multiple loss functions.
[11] This method does not guarantee to find a global minimum.

Finally, the $\mathbf{W_2}$ weights are updated according to the Equation (2.4), where α is a learning rate, a hyper-parameter of the training process, which influences how much the weights are updated in each iteration and, therefore, the speed of convergence.

$$\mathbf{W_2'} = \mathbf{W_2} - \alpha \cdot \frac{\partial L(\mathbf{y}, gt)}{\partial \mathbf{W_2}} \tag{2.4}$$

Conversely, it cannot be too large, otherwise the model's weights will start to oscillate. Finally, we need to clarify that the weights of a model that is trained are not typically updated with each individual input, although it is also possible. The mathematically correct way is to use the average gradient for all available training inputs. However, this is not possible and practical for large datasets, so the most common approach is to compute and back-propagate an average gradient for a batch of inputs.

A common development of applications capitalizing on the use of neural networks has, in recent years, been driven by pre-trained models. Those are well-validated and tested models of neural networks, examples given below, developed by large groups of scientists and trained on extensive datasets that capture most of the intricacies of the real world, such as ImageNet [22] or COCO [58]. These models are often made publicly downloadable with supplementary computer code for fine-tuning or transfer learning. Fine-tuning is a process where an unchanged model is further trained on a target data set, while transfer learning requires small changes to the model, usually to its output layers, and then training it with some restrictions. The restrictions are typically a small learning rate and freezing[12] of some layers. In both cases, the target dataset does not have to be very large, although the model's performance tends to improve with more specific data in the target domain. This enables the use of deep learning in situations where it would otherwise be impossible due to not having enough training data, and with better results than traditional machine learning methods could provide.

The main upside of deep learning is superior performance compared to machine learning or hand-crafted approaches. On the other hand, there are also some downsides. The first is the need for excessive amounts of training data. The second downside is a much higher need for memory and computational resources than with traditional methods. The first issue can be solved to a certain degree by the aforementioned transfer learning and fine-tuning techniques or improvements in the collection of training data. The second issue is much more burning, especially when training of deep learning models is considered. One of the most prominent solutions in this regard seems to be advanced analog chips, e.g., Rasch et al. propose a robust analog in-memory training algorithm, and A. P. James addresses the needs for strong analog neural chips in [80] and [40]. Memory and computational inefficiencies during inference are typically solved by decreasing the bit widths of the learned weights in a process called quantization, which we look at in depth in Chapter 3. Quantization can allow large deep learning models to run even on embedded devices like smartphones or smart cameras.

Image Classification is a fundamental tasks in deep learning. The goal is to assign labels to images from a defined set of categories. Hence, the Categorical Cross Entropy (CCE) Loss is used with the Softmax activation of the output layer. Typically, each image is assigned to one category, but multi-class classification is also possible. Deep learning

[12]The gradient stops being propagated at some layer of the network, and only the layers in the output direction are trained. The layer at which the training stops can be moved repeatedly backward during the transfer learning process.

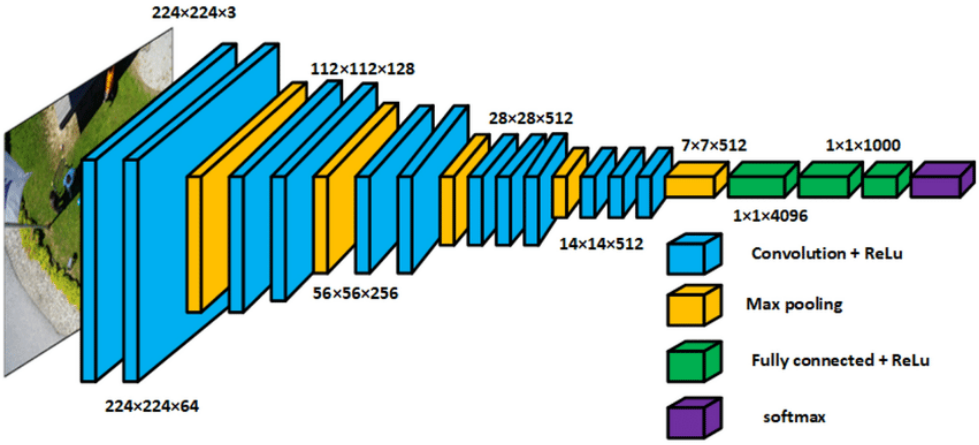


Figure 2.12: Architecture of 16-layer VGGNet
The image was authored by Ahmed et al. in [5].

neural networks, mainly CNNs, have significantly advanced image classification by enabling automated feature extraction, which replaces traditional hand-crafted methods listed above.

CNNs started to appear as superior to traditional approaches in 2012 when AlexNet, developed by Krizhevsky et al. in [51], won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 by a large margin with an error rate of 15.3 %, compared to 26.2 % achieved by the second-best entry. Truly deep learning CNN, named VGGNet, was then developed two years later by Karen Simonyan and Andrew Zisserman in [91]. The largest of the VGGNet versions has 19 trainable layers with 144 million parameters. The architecture of the most often used version of the VGGNet with 16 layers is depicted in Figure 2.12. Karen Simonyan and Andrew Zisserman were among the first to utilize multiple GPUs to accelerate the training. Many new CNN architectures followed since then, most notably the ResNet, which introduced residual connections solving the vanishing gradient problem[13] and enabling even deeper architectures, published by He et al. in [34].

In recent years, the most innovative solution has been to use vision transformers. Dosovitskiy et al. were among the first to develop a vision transformer for image classification in [24]. They split the input image into patches, which are converted into embeddings using a learnable linear projection. Resulting in that they are able to leverage Multi-Head Self-Attention layers for improved classification performance.

Object Detection extends beyond image classification by identifying and localizing multiple objects within an image. Unlike classification, which assigns a single label to an entire image, object detection predicts both the category of objects and the coordinates of their bounding boxes. Modern object detection networks typically combine three loss functions. First, Intersection over Union (IoU) Loss for maximizing the overlap between the predicted and ground truth boxes. Second, Binary Cross Entropy (BCE) Loss is used to train prediction confidence. Third, CCE Loss minimizes the difference between the predicted and ground truth class of the detected objects. Consequently, the Sigmoid activation function is used for outputs evaluated by the IoU and BCE losses, while the CCE loss is computed on outputs passed through the Softmax activation function.

[13]Gradients decrease or increase exponentially as they are back-propagated through layers using the gain rule, making it difficult for layers close to the input image to update effectively during training.

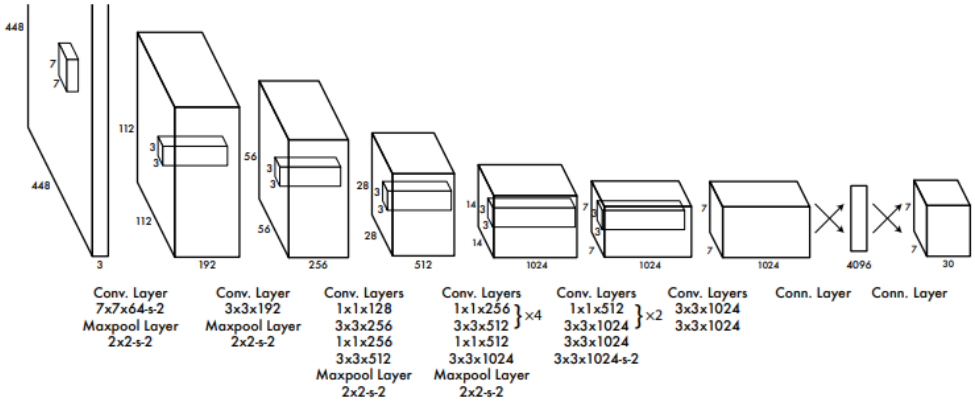


Figure 2.13: YOLO CNN architecture
The image was published by the authors of the YOLO CNN, Redmon et al., in [81].

Sermanet et al., as they point out, were the first to introduce a deep learning CNN named OverFeat for object detection in [87]. Their network uses a sliding window approach to scan the image at multiple scales and positions, applying the CNN to overlapping regions to generate predictions. These predictions are then refined and combined via a greedy merge strategy. They won the localization and detection parts of the ILSVRC 2013 with this approach. Girshick et al., also in 2013, proposed a Region-Based CNN (R-CNN) in [31] that combines the Selective Search algorithm with machine and deep learning. Their approach is based on generating region proposals using the Selective Search algorithm, followed by the extraction of features from each region with a CNN and the use of a machine learning classifier, such as SVM, to label the objects. This idea was soon improved by Ren et al. in [82] with a Faster R-CNN architecture. They were able to replace the algorithmic region proposal computation with a Region Proposal Network as well as remove the need for a machine learning classifier, thus making their network end-to-end deep learning. Another big leap forward in object detection was made by Redmon et al. with their You Only Look Once (YOLO) CNN in [81]. Their architecture divides the input image into a grid and predicts bounding boxes with confidence for each cell in a single forward pass of the network, treating object detection as a regression problem. See the visualization of the YOLO CNN architecture in Figure 2.13. This approach enables real-time object detection by avoiding region proposals and directly predicting objects' bounding boxes and classes. There were many iterations of the YOLO-type CNNs over the years, with many authors adding new improvements. At the time of writing this thesis, the latest published YOLO CNN was the YOLOv11 CNN developed by the Ultralytics[14] group.

The current trend in object detection is to combine CNNs with transformers. As with classification, the goal here is to build upon the advantages offered by the attention mechanism. Carion et al. were the first in 2020 to combine a CNN backbone based on the ResNet architecture with a traditional transformer encoder-decoder architecture in [15].

Optical Character Recognition (OCR) is a computer vision task that automates the conversion of text present in scanned documents or photographed images to a computer-encoded representation. Tesseract is the most popular and very successful algorithmic rule-based OCR engine, well described by Ray Smith in [92]. However, modern OCR systems

[14] https://www.ultralytics.com

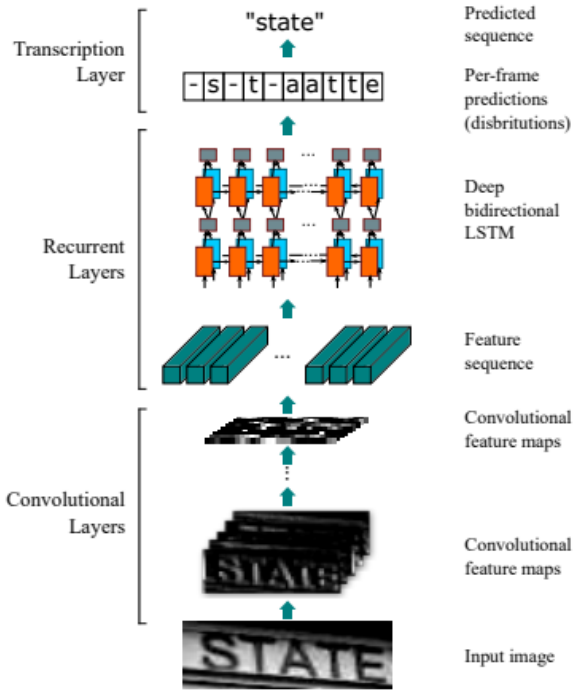


Figure 2.14: Convolutional recurrent neural network architecture for optical character recognition

The image was authored by Shi et al. in [89].

leverage deep learning and often treat the task as a combination of detection and sequence recognition problems. Connectionist Temporal Classification (CTC) Loss and CCE Loss are the most common loss functions used for training models of such networks. The CTC Loss is typically used for recurrent neural networks when there are no character-level annotations, only text labels. On the other hand, CCE Loss is used in attention-based models where the goal is to learn the alignment between image features and output tokens, i.e., characters, through a self-attention mechanism. The Softmax output activation function is typically used in both cases.

A baseline approach to OCR is to first detect individual symbols in an image with an object detection model and then recognize them with a classification model. Although naive, this approach can be successfully used in cases when the input images are constrained, e.g., for recognizing postcode numbers that have a specific number of digits. A significant improvement in general OCR came with the introduction of deep convolutional recurrent neural networks (CRNN), which combine convolutional layers for feature extraction with recurrent layers for sequence modeling and classification. Shi et. al. are the pioneers of CRNNs for OCR. They present a neural network composed of convolutional layers for image feature extraction, recurrent layers for the prediction of label distribution, and a transcription layer, which predicts the final character sequence in [89]. Their architecture is visualized in Figure 2.14. They achieved the best accuracy results on the majority of tested datasets at the time of evaluation in 2015, in several instances beating models of much larger neural networks. The introduction of vision transformers and their combination with language models brought another significant improvement to OCR. Li et. al. present an architecture that uses a pre-trained vision transformer for image feature extraction combined with a pre-trained transformer-based language model in [56]. Another improvement that they

bring is the prediction of word-piece tokens instead of separate characters. This approach brings better recognition performance on sentence-like texts while retaining good computational complexity. Their best model beats other state-of-the-art approaches that do not use external language models in the CER metric, with a score of 2.89 at the time of the evaluation in 2021. This approach cannot, however, be used when the read text does not have any a priori probability, like registration plate numbers. In the case of registration plate recognition, vision transformers can be trained to directly predict, e.g., seven to ten symbols from a limited number of classes, i.e., most often just capital letters and digits. Andrés Aranda utilizes Mobile Vision Transformer version 2 (MobileViTv2) architecture to do exactly the aforementioned. He offers multiple pre-trained registration plate OCR models in his fast-plate-ocr[15] GitHub repository.

Semantic segmentation is a deep learning task that involves partitioning an image into distinct regions, where each pixel is assigned a label corresponding to a specific object or category. Unlike object detection, which provides bounding boxes around objects, image segmentation delivers a finer level of granularity, outlining object boundaries with pixel-level precision. This makes it fundamental for applications where detail matters, such as medical imaging, autonomous vehicles, or augmented reality. The commonly used loss functions are the CCE Loss combined with the Dice Loss, which maximizes overlap between predicted and ground truth masks. Both of these loss functions are tied to the Softmax output activation function.

One of the first end-to-end deep learning CNN for image pixels-to-pixels level semantic segmentation was developed by Long et al. in [61]. They use a type of neural network called a fully convolutional network (FCN). The overall architecture is similar to one of the autoencoders described above in Section 2.3.1. Their FCN incorporates only convolutional and transposed convolutional layers[16]. This approach has two major advantages over CNNs with fully connected layers. First, fully connected layers have much higher trainable parameter counts, which leads to higher memory requirements, and they generally require more computational resources on modern hardware. Second, having only convolutional layers enables the trained models to process images of various sizes since the weights of convolutional filters are shared. This FCN architecture significantly improves upon the state-of-the-art on the PASCAL VOC 2011 and 2012 test sets while also reducing the inference time in orders of magnitude. Another popular solution is the HRNetV2 architecture published by Wang et al. in [97]. This network is also an FCN, but improves upon the architecture proposed by Long et al. by maintaining high-resolution representations throughout the whole network in parallel with lower resolutions. The parallel streams of different resolutions are repeatedly fused in the network's architecture, see Figure 2.15, which allows it to capture both global and local dependencies very well. The HRNetV2 achieved state-of-the-art performance at the time of its publication.

Of course, semantic segmentation is also a task that can leverage the attention mechanism provided by transformers. Swin Transformer is one such architecture capable of not only semantic segmentation introduced by Liu et al. in [59]. They introduce a shifted window mechanism for self-attention, which operates on non-overlapping windows, reducing computational complexity to linear with respect to image size.

[15] https://github.com/ankandrew/fast-plate-ocr

[16] Transposed convolutional layers are called deconvolution layers in [61].

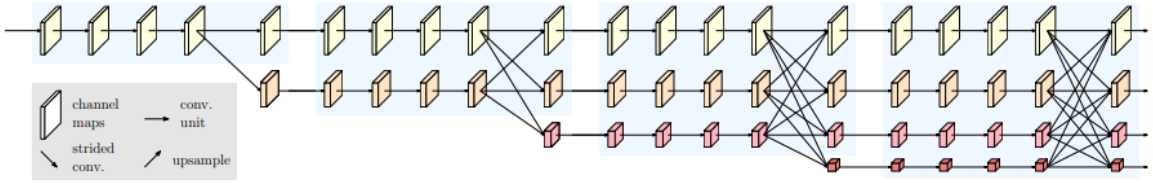


Figure 2.15: Visualization of parallel streams of different resolutions in the HRNetV2 architecture

The image was authored by Wang et al. in [97].

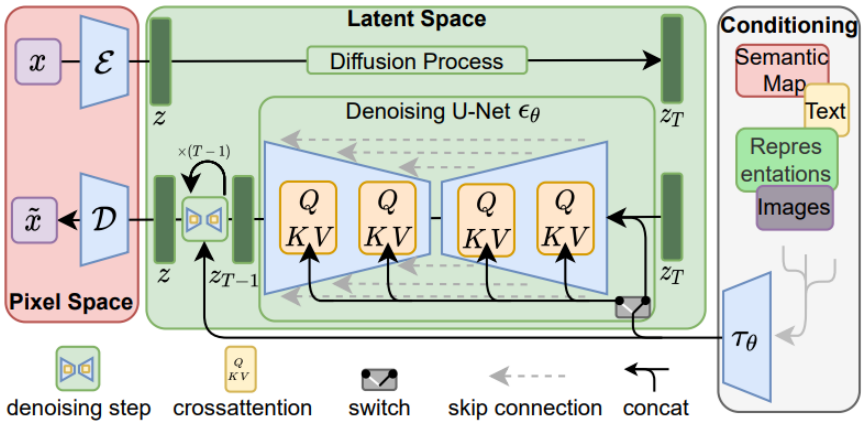


Figure 2.16: Neural network architecture for latent diffusion image generation

The image was authored by Rombach et al. in [83].

Image Generation involves creating new, realistic images from scratch or modifying existing ones by using models of deep learning neural networks. Recent advancements in this field have led to its increased popularity due to various applications, including content creation, data augmentation, image post-processing, and artistic design. Several neural network types, such as autoencoders, GANs, and diffusion models, also used for tasks other than image generation, have already been named in Section 2.3.1. Typically, the Mean Square Error (MSE) Loss is used to minimize the difference between predicted and ground truth pixel colors. The MSE Loss can operate with several activation functions. The Sigmoid activation function is the most popular in this context. Hyperbolic Tangent activation function or no output activation function is, however, also possible.

Autoencoders and GANs are already quite obsolete for the task of image generation. Therefore, they will not be discussed further. Diffusion models have also been improved upon with latent diffusion models. The former operates in a pixel space, which makes them computationally inefficient, while the latter exploits a lower-dimensional latent representation of an image compressed by a pre-trained encoder part of an autoencoder network. Once the diffusion process is completed in the latent space, the compressed information is transformed back into pixel space with the decoder part of the autoencoder. One of the pioneers of latent diffusion models are Rombach et al., who published their latent diffusion model based on a time-conditional UNet neural backbone in [83]. Their architecture is depicted in Figure 2.16. Furthermore, they augment the UNet backbone with a cross-attention mechanism, which enables them to train models of the network to generate images based on a text prompt or object layout description. Please, view their paper for examples of the generated images. Moreover, the network can be trained for the task of super-resolution

or object removal from photographs. Their models achieved or surpassed state-of-the-art performance in all of these tasks at the time of publishing.

The current state-of-the-art image generation is led by the DALL-E series of networks developed by the OpenAI company, which is publicly known mostly for its chatbots. The DALL-E networks are based on a 12-billion parameter autoregressive transformer trained on 250 million image-text pairs by Ramesh et al. in [79]. Unfortunately, from the research standpoint, although the DALL-E networks have been extensively described in academic papers, blog posts, and public discussions by OpenAI, their implementation details, including architecture, hyperparameters, and training processes, are not fully disclosed.

2.5 Chapter Summary

In this chapter, we present the main three camera controls including aperture, shutter speed, and analog gain, followed by a broad traditional ISP pipeline describing the following stages: black level subtraction, anti-aliasing and noise filtering, defective pixel correction, demosaicing, gamma correction, white balancing, color correction, lens corrections, luminance and chrominance noise filtering, and image enhancements. Next, we describe deep learning ISP approaches incorporating the VDSR [45] or ESPCN [90] CNNs for super-resolution, EnlightenGAN [43] or DCE-Net [33] for low-light enhancements, joint demosaicking and denoising CNN [29] or ResDNet [46] for joint demosaicking and denoising, and PyNET [38] or PyNET-V2 [39] for end-to-end sensor-to-image deep learning ISP. Then, we focus on the inner workings of neural networks, including an intuitive presentation of the back-propagation algorithm for their training, which is followed by their upsides, i.e., superior performance to any other method, and downsides, i.e., need for large datasets and a lot of computational resources. Last, we research common computer vision tasks in the style of a review paper. We describe past and current approaches by presenting selected neural networks, e.g., VGGNet [91] for image classification, YOLO CNN [81] for object detection, TrOCR [56] for optical character recognition, HRNetV2 [97] for semantic segmentation, and a diffusion model based on a time-conditional UNet neural backbone [83] for image generation.

We have now gained a broad understanding of ISP, image enhancements, neural networks, their training, and computer vision tasks. Object detection with the YOLO architecture is especially important for the rest of our thesis. Therefore, in the next chapter, we present quantization approaches and a quantization experiment with a specific model of this architecture to better utilize it on edge devices.

Chapter 3

Quantization for Efficient Inference on the Edge

In this chapter, we present the advantages of quantization as a tool for making neural networks' inference computationally possible with low-power devices, such as mobile phones, wearable electronics, or smart sensors, collectively referred to as devices *on the edge* or *edge devices*. First, we briefly describe the motivations behind quantization from 32-bit floating point numbers (Fp32) to 8- or even 4-bit integers (Int8 and Int4, respectively), the quantization process itself, and the required changes during inference with such quantized deep learning models. Then, we look at the two most common approaches: post-training quantization and quantization-aware training. This is followed by a section dedicated to a compromise between the just-named approaches, which are quantization-aware fine-tuning and quantization-aware transfer learning. Last, we methodically compare the performance of these approaches with an experiment on the YOLOv8 convolutional neural network for detection and present the results.

3.1 Quantization and Quantized Inference

Quantization was, until recently, mostly used in the context of converting analog signals with an infinite number of values to a finite set of values, i.e., quanta, representable in computers on an available number of bits [19]. Lately, the term is most often used in regard to the mapping of neural networks' weights from Fp32 to Int8 or even Int4 integers. The latter usage of the term can be seen as a lossy compression with a pre-determined compression ratio[1]. The goal of any lossy compression with a given compression ratio is to retain as much quality, e.g., visual appeal of compressed images. In the case of neural networks, the quality can be understood as the performance of the quantized network or, rather, the performance difference between the full-precision network and its quantized version[2].

Compression of neural networks, i.e., less required memory and disk space, is one of the two main motivations for quantization. The second motivation is linked to computation speed and power efficiency. The neural network inference is predominantly based

[1] In case of quantization from Fp32 to Int8 the ratio is 4.

[2] In some cases, quantized networks can outperform their full-precision original versions. One explanation can be that the full-precision network might be over-fitted on the training data, which the quantization process counters, allowing for better generalization.

on an extensive series of multiplications and additions[3]. Both floating-point multiplication and addition are multi-step processes. During multiplication, only the mantissae of the two operands are multiplied, the sign bits must be xor-ed, and the exponents added. Furthermore, the resulting product must be normalized and the final exponent adjusted accordingly [1]. Addition is not simpler. The two operands must first be aligned to the same exponent[4], converted to their second complement if negative, added, the sign bit must be determined, the resulting sum must be converted back from the second complement if negative, normalized, and, finally, the exponent must be correctly adjusted [1]. On the other hand, multiplication and addition of Int8 operands require just an 8-bit multiplier/adder, which is yet much simpler than the 24-bit hardware components required for the operations on Fp32 mantissae that are stored normalized on 23-bits with the 24th bit being implicitly set to 1[5]. Clearly, the hardware requirements for Int8 computation are much lower than for the Fp32 one, which should result in both much faster and more power-efficient designs. Baalen et al. in [6] come to a similar conclusion when comparing computations in much simpler 8-bit floating point numbers to the 8-bit integers.

The most common type of quantization is Uniform Affine Quantization because it enables efficient implementation of fixed-point arithmetic [68]. We will refer to this type of quantization simply as quantization for the rest of this thesis if not explicitly specified otherwise. The quantization is defined by three parameters: scale, zero-point, and bit-width [68]. The bit-width, scale, and zero-point are denoted in the equations below with b, s, and z, respectively.

The scale is a floating-point number determining the interval between two values that can be quantized without an error. The simplest approach to obtaining the scale is to subtract the minimum and the maximum recorded values and divide the difference by the number of representable values on the given bit-width subtracted by 1, i.e., $2^b - 1$. Consider $\mathbf{W}$ as the learned weights of a neural network. Then the scale can be, in the simplest way, computed as shown in Equation (3.1).

$$s = \frac{max(\mathbf{W}) - min(\mathbf{W})}{2^b - 1} \tag{3.1}$$

This is, however, a very naive approach. Typically, the quantization is performed on a per-layer or per-channel basis. Outliers are also addressed to reduce the quantized range and, therefore, decrease the average quantization error.

The zero-point is an integer, and its purpose is to represent quantized zeros without any errors, which is important since many operations, like zero-padding or the results of ReLU activation, require precise zeros not to introduce errors [68]. The zero-point can be implicit in the middle of the quantized range, then we are talking about symmetric quantization, or explicit, which results in an asymmetric quantization. Again, the simplest way to compute the explicit zero-point is expressed in Equation (3.2), where $round(\cdot)$ is the round-to-nearest operator.

$$z = round(\frac{0 - min(\mathbf{W})}{s}) \tag{3.2}$$

[3]Multiplications and additions are the basics of more complex algebraic operations like matrix multiplication or convolution.

[4]The smaller number is shifted to the exponent of the larger number. This is the reason why adding a very large and a very small floating point number will be imprecise or have no effect at all.

[5]That is in most cases, unless the number is denormalized.

Finally, the learned 32-bit floating point weights $\mathbf{W}$ of a neural network are quantized into b-bit integer weights $\mathbf{W_{int(b)}}$ as shown in Equation (3.3), where $clamp(\cdot)$ operator saturates values smaller than 0 to 0 and values larger than $2^b - 1$ to $2^b - 1$.

$$\mathbf{W_{int(b)}} = clamp(round(\frac{\mathbf{W}}{s}) + z) \tag{3.3}$$

The inverse operation, de-quantization, denoted in Equation (3.4), obtains the original weights with a quantization error $\hat{\mathbf{W}}$.

$$\hat{\mathbf{W}} = (\mathbf{W_{int(b)}} - z) \cdot s \tag{3.4}$$

There are two types of quantization errors. First, the errors caused by the $round(\cdot)$ operator, i.e., the rounding error, that lies in the $\langle -\frac{1}{2}s, \frac{1}{2}s \rangle$ interval [68]. Second, the errors inflicted by choosing the scale and/or zero-point differently than the examples above, such that the operands of the $clamp(\cdot)$ operator can be out of the $\langle 0, 2^b - 1 \rangle$ interval and must be saturated if that is the case. There are several methods that are trying to minimize the quantization errors, but most importantly, that perform the quantization with the goal of achieving the best inference performance. Some of these methods offer to make a trade-off between performance and computational efficiency, where users can typically set one of the parameters that must be achieved. In some cases, like convolutional filters, even Int4 weights can lead to acceptable results, as reducing the bit widths is becoming a trend on modern neural network accelerators[6]. We can see a future where special low bit-width and low power consumption neural network models with higher false positive detection rates are developed to run as a pre-stage to more complex and more power-demanding models that, in turn, will not be utilized as often, hence, potentially improving both the power efficiency and accuracy of such systems.

3.2 Post-Training Quantization

Post-training quantization (PTQ) is a quantization method of neural networks after they have been fully trained. Unlike quantization-aware training, PTQ requires no retraining or modification of the original training process, making it a popular choice when computational resources are limited or the original training data is unavailable. In both cases, this is typical for large pre-trained models or models trained on sensitive data like medical records. PTQ aims to convert floating-point values to lower-bit representations, most often 8-bit integers, while striving to minimize the resulting degradation in model accuracy caused by the mapping of weights to a lower precision. [37]

To mitigate this, PTQ employs techniques such as calibration, in which a calibration dataset is passed through the trained model to obtain statistics about the distribution of layer outputs and their activations. Calibration enables the selection of optimal scale and zero-point values, which minimize quantization errors in a way that is more representative of the actual calibration data and the learned weights. Additionally, per-channel or per-tensor quantization schemes can be employed, depending on the precision requirements of individual layers. In per-channel quantization, each layer's channel (or filter) is quantized with its unique scale and zero-point, offering better accuracy but at the cost of slightly increased storage and computation complexity. Furthermore, after the whole model is quantized,

[6]The Hailo-8 AI Accelerator supports 4-, 8-, and 16-bit weights, see https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/OPTIMIZATION.rst.

another forward pass of the calibration dataset can be performed on both the quantized and full-precision models to observe the quantization errors accumulating throughout the quantized model. These errors can then be further reduced by iteratively adjusting the quantization parameters of each layer so that the final outputs of the quantized model are as close to the full-precision model as possible. Typically, the MSE is minimized. One such algorithm, the AdaQuant, was developed by Hubara et al. in [37]. They also reduce the degradation of the quantized model's accuracy by dynamically adjusting the bit-widths of weights in different parts of the quantized model, i.e., layers that cause worse accuracy degradation are quantized using more bits per weight, e.g., 16 bits, while other layers may not affect the performance significantly when quantized even to sub-8-bit values. This approach is, however, not always applicable, as some accelerators may support only the 8-bit integer data type.

Despite these methods, PTQ can lead to significant accuracy drops, particularly in networks with high sensitivity to quantization errors. For such models, other advanced techniques, such as bias correction or cross-layer equalization, may be implemented to enhance accuracy post-quantization. Importantly, these techniques do not require any calibration data nor running back-propagation. [67]

Nagel et al. in [68] recommend using the following pipeline in order to achieve competitive post-training quantization performance with many computer vision and natural language processing models. The first step, a pre-processing step, should be the cross-layer equalization that makes the full-precision model more quantization-friendly. Models of neural networks with depth-wise separable layers and per-tensor quantization can benefit the most, though it often enhances other layers and quantization options as well. Next, they recommend selecting quantization parameters and adding quantization operations to the network. Symmetric quantization is preferred for weights, while asymmetric quantization might be the better choice for activations. Most importantly, the choice of quantization type should be guided by the capabilities of the target hardware. Then, they advise setting the quantization parameters of weight tensors using layer-wise MSE criteria. On the other hand, the min-max method can sometimes be advantageous for per-channel quantization. If a small calibration dataset is available, AdaRound can be applied to optimize weight rounding. In the absence of a calibration dataset or if the network uses batch normalization, analytical bias correction can be used instead. Finally, they determine the quantization ranges for all data-dependent tensors (activations) in the network using the MSE-based criteria for most layers, which requires a small calibration dataset to minimize the MSE loss. Alternatively, batch normalization-based range setting can be used to create a fully data-free pipeline.

In summary, PTQ is a practical choice for deploying large models where retraining is not feasible, offering a fast and resource-efficient path to quantization. While calibration and optimization techniques help preserve accuracy, PTQ can still struggle with models sensitive to precision loss. By contrast, quantization-aware training typically delivers better resilience to quantization errors but at the cost of extra training resources. Therefore, PTQ is ideal when simplicity and deployment speed are prioritized over inference accuracy.

3.3 Quantization-Aware Training

Quantization-aware training (QAT) is an approach that integrates the quantization process directly into the training phase of a neural network model, enabling the model to learn how to operate effectively under low-precision constraints. Unlike post-training quantization,

where quantization occurs after training and can introduce substantial accuracy losses, QAT minimizes this impact by exposing the model to quantization effects during training. This technique enables the model to adapt its weights to the quantization-induced noise and other artifacts, thereby achieving better performance when deployed with lower precision weights. This does not, however, come for free. Simulating the reduced precision causes an overhead in training time as well as used memory, which might not be acceptable when training very large models. Moreover, this method cannot be used when there is not enough properly labeled data available or the neural network is not well defined to be instrumented with the necessary additional operators. [68]

In QAT, the network weights and activations are simulated at reduced precision during forward propagation, while the back-propagation process continues to operate with full precision. This allows the model to approximate the effects of lower bit widths, such as 8-bit integers, during training. The training process, thus, incorporates quantization noise directly into the weight updates, encouraging the network to adjust parameters to improve resilience to quantization-induced errors. Baalen et al. describe this well in [6].

A key aspect of QAT is the inclusion of fake quantization nodes, which simulate quantization by converting floating-point values to lower-precision values during training. These nodes approximate the quantization behavior by introducing the aforementioned rounding and clamping operators without altering the original floating-point precision of the network parameters during training. This approach allows the model to converge to weights that are better suited for quantized inference. The issue here is the computation of the back-propagation because the round and clamp operators do not have a gradient. Fortunately, estimations exist, one of the most popular being the Straight-Through Estimator proposed by Geoffrey E. Hinton and formulated by Bengio et al. in [12]. The estimation enables the model to learn the most effective quantization parameters. Another problem is weight oscillation. It occurs when a full-precision weight is close to the decision threshold between two quantization bins and, in subsequent training iterations, keeps falling repeatedly in the first and then the second bins. This problem has also been mitigated, e.g., by Nagel et al. in [69].

The quantization operators are typically applied for both weights and activations on a per-layer basis, i.e., each layer has a set of the quantization parameters (scale and zero-point), which provide a relatively fine-grained quantization scheme that balances accuracy and computational efficiency [50]. In addition to scale and zero-point adjustments, QAT can also incorporate gradient clipping, where gradients are constrained within a specific range to mitigate the instability introduced by quantization noise [72]. This helps maintain training stability and allows the network to reach optimal performance under the quantized settings. Furthermore, special learning rates, considering that the quantized weights only change their discrete values when transitioning between the quantization bins, can be used. Otherwise, the training may plateau as traditional learning rates might not propagate gradients strongly enough to induce changes. One such learning rate, the Transition-Adaptive Learning Rate, was developed by Lee et al. in [55].

Overall, QAT is highly effective for models that need high accuracy in quantized form, such as those deployed in mobile applications or edge devices with stringent computational and memory limitations. However, QAT requires greater computational resources during training compared to standard training, as it introduces additional overhead for simulating quantized behavior. The trade-off is often worthwhile, as QAT-trained models typically exhibit higher resilience to quantization errors than their post-training quantized counterparts.

3.4 Quantization-Aware Fine-Tuning and Transfer Learning

As discussed in Section 2.4, the current state-of-the-art approach is to leverage pre-trained models for various applications. These models are, as a rule, not developed using QAT nor are they already quantized because various target hardware architectures can require different quantization. Therefore, the quantization for a specific device remains with the developer of the underlying application. This means that such a developer has three options: quantize a downloaded pre-trained model as it is using PTQ, perform transfer learning or fine-tuning in full precision, i.e., training the model on the target dataset without changes to the training process, followed by PTQ, or conduct quantization-aware transfer learning/fine-tuning, i.e., modifying the training process to simulate quantization, combined with PTQ for the final result.

He et al. in [35] and Jeon et al. in [42] successfully used quantization-aware fine-tuning of diffusion and large language models, respectively. Finkelstein et al. then describe quantization-aware fine-tuning of practically any model using the full precision network as a teacher for the quantized student in a knowledge-distillation setting in [27]. This method is, among others, used for the quantization of full precision models for Hailo AI devices by their Dataflow Compiler.

To our knowledge, there has not been any work published on quantization-aware transfer learning yet. A method where a pre-trained model would be trained in a quantization-aware setting after all necessary changes had been made to achieve the new end goal. Consequently, we have conducted an experiment to determine what could be the go-to approach for object detection models sized for edge deep learning. The experimental setup, the performed experiment, and its results are presented in the next section.

3.5 Experimental Comparison of Quantization Approaches

This section aims to provide guidance for quantizing relatively small deep learning models capable of running on edge devices equipped with accelerator units working in 8-bit or lower integer precision. We have asked ourselves the following questions:

1. *Does quantization-aware training bring any performance benefits, or are the currently used post-training quantization methods advanced enough to maintain on-par performance?*

2. *Can quantization-aware training only at the end of a learning phase, i.e., quantization-aware fine-tuning, equate or even surpass the model's performance as if it was trained in a quantization-aware manner from the beginning?*

3. *Can quantization-aware transfer learning be leveraged when adjusting the specialization of a conventionally pre-trained model?*

Our experimental setup consists of PyTorch Quantization API[7] for quantization-aware training and Hailo Dataflow Compiler[8] for post-training optimizations, quantization and compilation[9] of models into HEF file format supported by Hailo AI accelerators. The Hailo

[7] https://pytorch.org/docs/stable/quantization-support.html

[8] https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/GETTING_STARTED.rst

[9] The compiled models also contain Non-Maximum Suppression algorithm for deduplication of multiple predictions predicting the same object and its bounding box made by the model.

Dataflow Compiler can also perform quantization-aware fine-tuning if a provided dataset contains more than 1024 samples, while only a calibration is performed for smaller datasets. Thus, we can always compare four scenarios. Conventionally trained model (1) compiled with a small dataset[10] and (2) compiled with a large dataset[11] against a model trained in quantization-aware setting (3) compiled with a small dataset and (4) compiled with a large dataset. To answer our posed question, we have analyzed models of the YOLOv8m convolutional neural network for detection, a mid-sized network offering a great trade-off between detection performance, number of parameters, and computational performance. We use the following metrics for the evaluation of the trained YOLOv8m models:

- Intersection over Union (IoU) is the ratio between the intersection of the area of a predicted bounding box (P) with the area of a ground truth bounding box (GT) and their union.
$$IoU = \frac{area_{GT} \cap area_P}{area_{GT} \cup area_P}$$
- Precision is a measure of quality, and it is calculated as a ratio between relevant retrieved instances (true positives) and all retrieved instances (true positives and false positives), i.e., how often a model predicts correctly when it makes a prediction.
$$Precision = \frac{true\ positives}{true\ positives + false\ positives}$$
- Recall is a measure of quantity and it is calculated as a ratio between relevant retrieved instances (true positives) and all relevant instances (true positives and false negatives), i.e., how often a model makes a correct prediction when there is something to be predicted.
$$Recall = \frac{true\ positives}{true\ positives + false\ negatives}$$
- F1 score is the harmonic mean of precision and recall.
$$F1 = 2 \cdot \frac{Precition \cdot Recall}{Precition + Recall}$$

A common property holds for all the listed metrics. The higher their value, the better the given model performs in that regard.

The first planned experiment focuses on training models from scratch in a conventional and quantization-aware setting. Its goal is to determine the achievable performance gain of the latter method. In both scenarios, the training is performed on a newly initialized model with the same random seed. Both training runs use the same hyperparameters, e.g., the batch size (32 samples), optimizer (Stochastic Gradient Descent), learning rate schedule (0.01 to 0.001), etc. We have trained both models for 401 epochs on the COCO [58] dataset and compared their performance after compilation for the Hailo-8 AI Accelerator every 25 epochs. The COCO dataset contains labels (bounding boxes and classes) for detecting objects from 80 diverse classes. This dataset is often used for object detection models pre-training before they are published online.

[10]The small dataset contains 100 samples of the dataset that the model was originally trained on.

[11]The large dataset contains 1250 samples of the dataset that the model was originally trained on, but only 1024 samples used by the Hailo Dataflow Compiler.

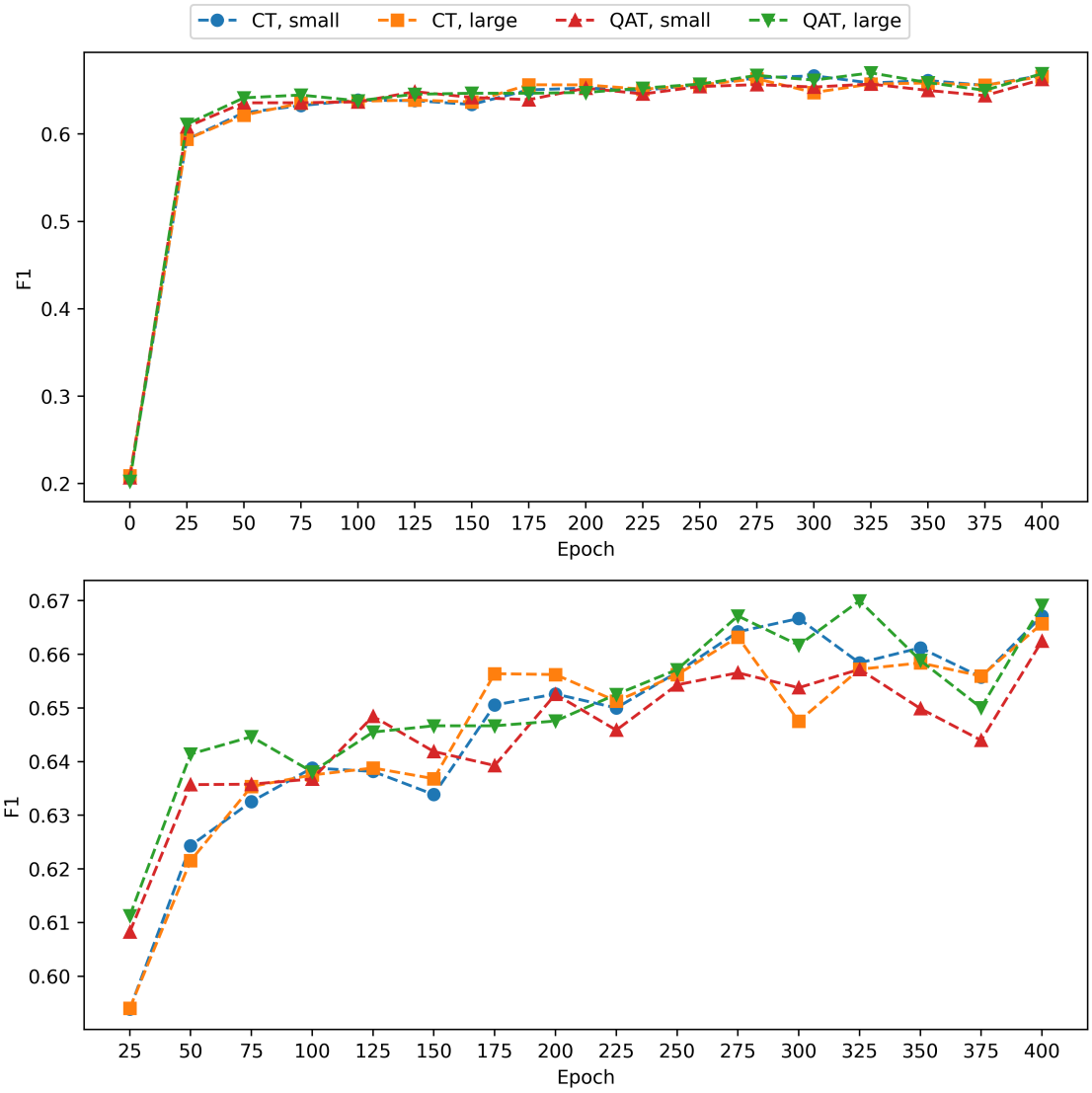


Figure 3.1: Best F1 scores achieved by compiled models for the Hailo-8 AI Accelerator during training

CT and QAT stand for conventional training and quantization-aware training, respectively. Small and large indicate the dataset size used for the compilation of the models.

The models' performance on the evaluation part of the COCO dataset during training is depicted using the F1 score metric in Figure 3.1. We have evaluated each model at various confidence levels and picked the largest value for each tested epoch. Confidence is a value provided by the model that indicates how strongly it believes that its predictions are correct. Higher confidence values will decrease the number of false positives but also increase the number of false negatives, i.e., increase Precision and decrease Recall. IoU also tends to increase when the model is more confident with its predictions. This is depicted, for better clarity, only for models compiled with the large dataset in Figure 3.2. As we can see in the first referenced figure, all models reach acceptable performance after around 50 to 75 epochs. Although we cannot claim that the models are fully trained after the 401 epochs, further performance increases in the following epochs are only marginal.

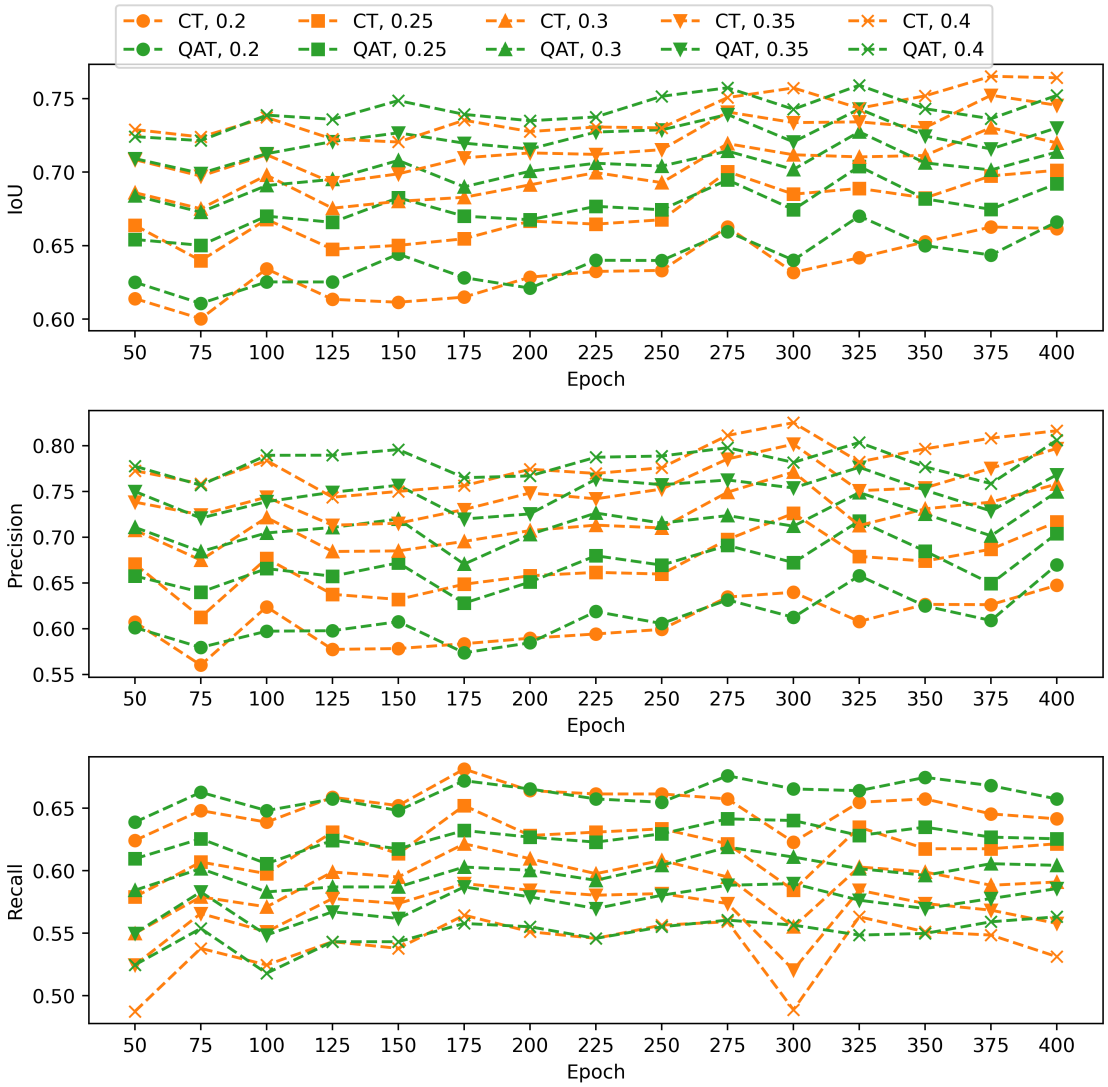


Figure 3.2: Iou, Precision and Recall metrics at various confidence levels achieved by compiled models for the Hailo-8 AI Accelerator during training

CT and QAT stand for conventional training and quantization-aware training, respectively. Decimal values indicate the confidence level used.

The more important aspect of our experiment, quite unexpectedly given the research conducted above, is that the quantization-aware training does not seem to provide any significant benefit. The performance differences are within the range that can easily be explained by the randomness of the compilation process that we cannot control. Nevertheless, there might be an explanation for the unexpected results. The quantization-aware training with its quantized forward pass might not enable the model to learn as well as the conventionally trained model can. The learning deficiency might then be recovered during the quantization process, ultimately leading to the same results for both training approaches.

Therefore, we have also performed the same evaluation on both models before compilation with full floating-point precision. The evaluation results using the same approach as

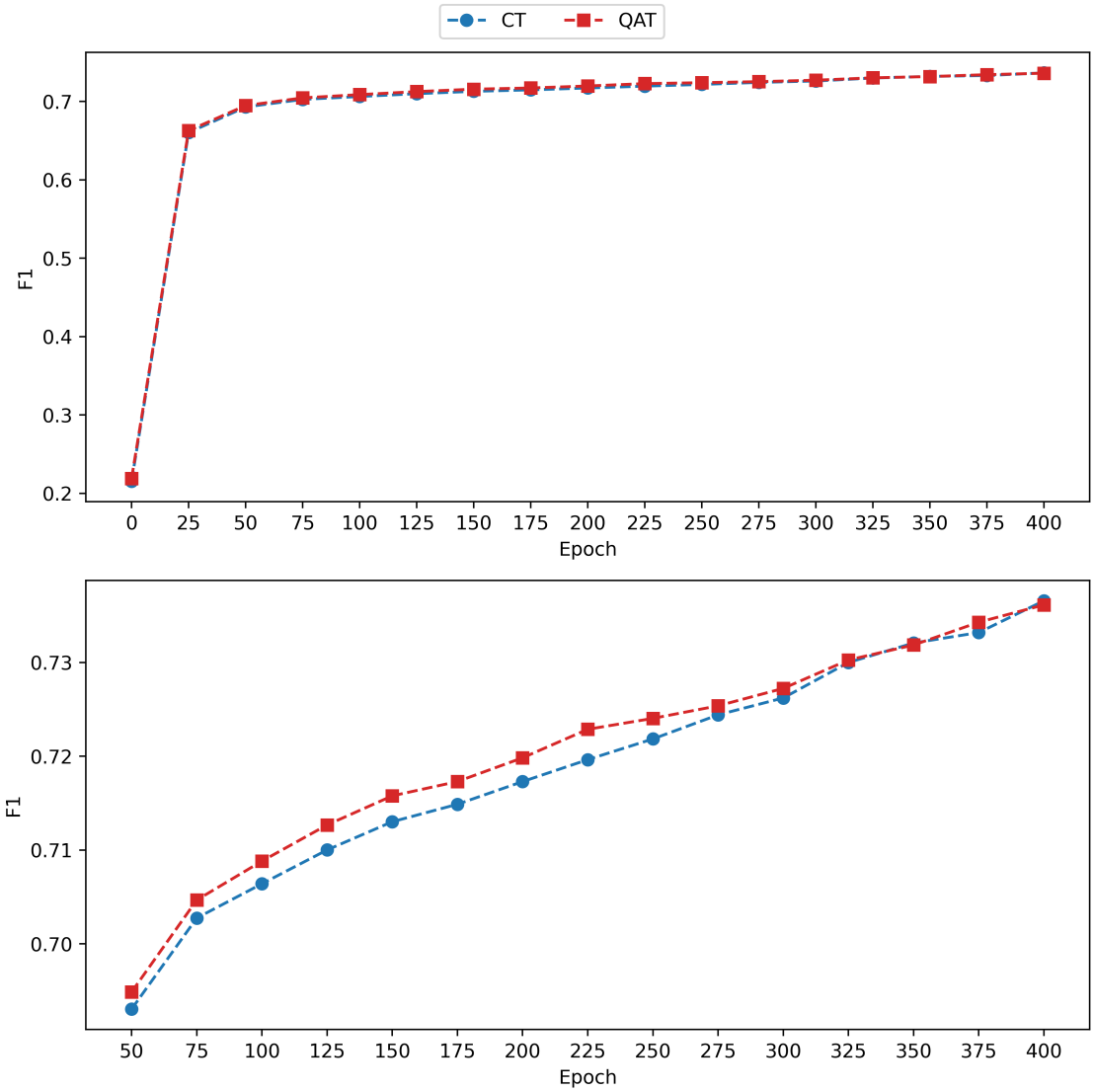


Figure 3.3: Best F1 scores achieved by full floating point precision models during training
CT and QAT stand for conventional training and quantization-aware training, respectively.

for the compiled models with the F1 score metric are shown in Figure 3.3. We can clearly see that the performance of the quantization-aware trained model matches its conventionally trained counterpart. Despite the fact that the models are not fully converged yet, as evidenced by the ever-increasing F1 score values, we can disprove our proposed explanation for the lack of performance difference between the compiled models. We should, however, mention that the quantization is not perfect. When comparing the figures 3.3 and 3.1, we can see that the full floating point precision models achieve an F1 score of around 0.735 and are still improving, while the quantized models only reach values around 0.67. This is a non-negligible difference.

Although we have planned further experiments, including quantization-aware fine-tuning and quantization-aware transfer learning of a conventionally trained model to increase its quantized performance on a target dataset and when changing the model's specialization, respectively, the first experiment answers all of our posed questions. Quantization-aware

training does not provide any performance benefit for models of the YOLOv8m convolutional neural network compiled with the Hailo Dataflow Compiler in our experiments. Therefore, neither quantization-aware fine-tuning nor quantization-aware transfer learning would. We also do not see a significant difference between using the small and large datasets for compilation. This is, on one hand, unexpected, but on the other hand, encouraging. Encouraging in the way that it is not necessary to perform quantization-aware training, which in our case took 58 minutes per epoch compared to 33 minutes per epoch needed for conventional training on an NVIDIA RTX 6000 Ada graphics card, while post-training quantization can achieve practically the same results. Even though we should not generalize these results to other neural networks or to other compilers, we can be reasonably confident that models of other neural networks will behave similarly when post-training quantized and compiled with the Hailo Dataflow Compiler. Nevertheless, comparison of PTQ and QAT approaches on other neural networks should be investigated in future research.

3.6 Chapter Summary

This chapter focuses on neural network quantization, a process where learned floating point weights of neural networks are converted to integers in order to save memory and improve performance in terms of speed and power efficiency with a minimal impact on the deep learning inference results. We present the mathematical framework for Uniform Affine Quantization [68]. Follows a description of the two main quantization approaches, PTQ and QAT. We mention calibration and algorithms like AdaQuant [37] or AdaRound [68] for PTQ as well as the Straight-Through Estimator [12], gradient clipping [72] or Transition-Adaptive Learning Rate [55] for QAT. Then, we briefly look at quantization-aware fine-tuning and transfer learning, where we describe, e.g., a knowledge-distillation approach for fine-tuning [27]. Last but not least, we conduct an experiment of training a model of the YOLOv8m convolutional neural network in both conventional and quantization-aware settings and compile it with the Hailo Dataflow Compiler. We come to the unexpected conclusion that the PTQ matches the performance of the QAT for this model and compiler, while the literature suggests that QAT should outperform PTQ.

The understanding of quantization with the presented results in this chapter, combined with the theoretical knowledge gained in the previous chapter, gives us a strong basis to implement a deep-learning-based computer vision application on edge integer-based deep learning accelerators. Therefore, the next chapter focuses on the selection of the most suitable platform for our application.

Chapter 4

Target Platform Assessment and Selection

The goal of this chapter is to describe and defend the selection of a target device for our final application. Throughout the work on this thesis, we have tested and compared four platforms with 8- or 4-bit integer acceleration of neural network inference. The main questions that we want to answer in this chapter are:

- *Are there any blocking issues with any of the platforms that make the platform not suitable for our purpose?*
- *How do the platforms stand against each other in the following categories: image quality, deep learning capabilities, general computing power, power consumption, implementation difficulty, and purchase price?*
- *What can be another purpose of each platform if it is not selected as our target platform?*

We start the chapter with a short description of each platform that includes both objective metrics and subjective impressions. This is followed by a section where we compare the platforms against each other. Finally, we select one platform to proceed with and lay out some potential use cases for the remaining platforms.

4.1 Target Platforms

This section describes the following tested platforms: Hailo-15 AI Vision Processor[1], Up Squared Pro board[2] with Hailo-8 AI Accelerator[3], Videology SCAiLX Development-Kit[4], and Raspberry Pi 5[5] equipped with Raspberry Pi AI HAT+[6] containing the Hailo-8 AI Accelerator and Raspberry Pi AI Camera[7].

[1] https://hailo.ai/files/hailo-15-product-brief-en/
[2] https://up-board.org/up-squared-pro/
[3] https://hailo.ai/products/ai-accelerators/hailo-8-ai-accelerator/
[4] https://www.videologyinc.com/product-guide?store-page=SCAiLX-Development-Kit-p685628811
[5] https://rpishop.cz/raspberry-pi-5/6498-raspberry-pi-5-8gb-ram.html
[6] https://rpishop.cz/554037/raspberry-pi-ai-hat-26-tops/
[7] https://rpishop.cz/535854/raspberry-pi-ai-camera/

4.1.1 Hailo-15 AI Vision Processor

The Hailo-15 AI Vision Processor is a promising development board. It offers a system-on-chip solution combining a Hailo neural network core, vision subsystem, digital signal processor, and application processor. This is coupled with 4 GB of random access memory (RAM) and many peripherals, including a MIPI in two versions: a MIPI Camera Serial Interface for the transmission of image frames from an image sensor to the application processor and a MIPI Display Serial Interface providing the connection between the application processor and a display.

The Hailo neural network core is capable of delivering up to 20 TOPS, which enables running multiple state-of-the-art deep learning models in parallel. The vision subsystem can process up to 600 megapixels per second with noise reduction algorithms, high dynamic range digital overlapping, electronic/digital image stabilization, lens shading correction, distortion correction, digital zoom, flipping, rotation, and video compression into H.264 or H.265 standards. The digital signal processor offers 256 multiply-accumulate units running at 700 MHz, supporting up to 350 GFLOPS per second. Lastly, the application processor is a quad-core Arm Cortex-A53 unit running at up to 1.3 GHz.

Despite these deep learning and processing capabilities, we were not able to execute our test applications on this platform. We have encountered issues when compiling the software, which we could not resolve. Consequently, we had to look for alternatives in the form of the Videology SCAiLX Development-Kit in Section 4.1.3 and a Raspberry Pi setup in Section 4.1.4. However, we hope that once the technical support for this board improves, we will be able to utilize its capabilities.

4.1.2 Up Squared Pro Board with Hailo-8 AI Accelerator

The Up Squared Pro board, in our case, is equipped with a low-power Intel Atom x7-E3950 quad-core processor for edge computing running at 1.60 GHz or 2.0 GHz in burst mode, 4 GB RAM, and an Intel Altera MAX 10 field programmable gate array (FPGA). Since the processor runs the x86-64 instruction set, a full Windows or Linux operating system can be loaded up and installed on the board using a boot-up flash drive. We have opted for Ubuntu 20.04 LTS. Moreover, the board can be equipped with peripherals and used as a standalone computer.

The Atom processor is capable enough to perform relatively computationally difficult pre- and post-processing tasks, but not for deep learning inference. Consequently, we have equipped the board with the Hailo-8 AI Accelerator, capable of delivering 26 TOPS for neural network inference on paper. Having a full operating system with a compiler makes testing applications leveraging the Hailo-8 AI Accelerator reasonably straightforward. It can also run a full Python interpreter with most of the common libraries[8]. Unfortunately, the Up Squared Pro board does not have a MIPI interface for connecting cameras, nor are there any specific cameras available for this board. Initially, our main goal with this board was to learn the Hailo AI toolchain while waiting for the delivery of the Hailo-15 AI Vision Processor. Since the experiments with the Hailo-15 AI Vision Processor were not successful, as mentioned above, we still tested the board on pre-recorded videos, as there are similar boards, e.g., the Up Squared Pro 7000 board, that can be equipped with a MIPI camera. Therefore, a combination of an Intel low-power processor, such as the Atom x7 series or N50 processors, with the Hailo-8 AI Accelerator can be a viable option.

[8] Some must be compiled specifically for the Atom processor.

4.1.3 Videology SCAiLX Development-Kit

The Videology SCAiLX Development-Kit combines an i.MX 8M Plus processors with a SCAiLX 2 MP Color Global Shutter Camera. The processor is a quad-core Arm Cortex A53 unit running at 1.6 GHz with a neural processing unit operating at up to 2.3 TOPS coupled with 2 GB of RAM. The 2.3-megapixel camera produces up to a FHD video. Moreover, it offers a configurable image signal processing pipeline over an I2C bus.

We received the development kit already assembled with a pre-installed operating system containing all necessary libraries and programs, such as a Python interpreter. A secure shell server was also already running when powering up the device for the first time. Initiating a connection with the kit was, therefore, straightforward. On top of that, Videology provides a detailed user manual containing a description of the board, instructions on how to assemble it, and, most importantly, examples of Python code for running deep learning models on the video feed from the camera. The manual also describes how to configure the image signal processing pipeline and how to stream the processed video over Ethernet.

4.1.4 Raspberry Pi 5 with Hailo-8 AI Accelerator and Raspberry Pi AI Camera

The Raspberry Pi 5 board is mounted with a quad-core Arm Cortex A76 processor running at 2.4 GHz. Our unit has 8 GB of RAM from the maximum of 16 GB that can be purchased. We were able to quickly set up the board with an operating system provided by Raspberry Pi. We use the option to connect to it over a secure shell. However, the board also supports connecting up to two monitors and other necessary peripherals to work as a standalone computer. We have additionally equipped the board with a Raspberry Pi AI HAT+ containing the Hailo-8 AI Accelerator, delivering 26 TOPS for neural network inference. Moreover, we have decided to use a Raspberry Pi AI Camera with an IMX500 Intelligent Vision Sensor[9]. The sensor contains a pipeline to run feed-forward neural networks directly on it and transmit the video feed with the inference results to the Raspberry Pi 5 for post-processing.

The provided operating system already comes with a pre-installed Python interpreter. Raspberry Pi provides Python libraries for both the camera and the accelerator, which are easy to install. Their software stack is also accompanied by detailed documentation. Furthermore, they offer a plethora of self-explaining examples on how to command the camera's image signal processing pipeline, how to pass data to the accelerator for inference, how to parse the inference results and apply an overlay on received video frames, and how to encode, display, store or stream the video over Ethernet.

4.2 Platform Comparison

We have designed three test applications. First, a simple video streaming over Ethernet. Second, an application that leverages a deep learning model for detection to draw bounding boxes around detected objects in a video streamed over Ethernet. Third, an application that detects a human in a video tracks it and crops the part of the video where the human is located, which is then streamed over Ethernet. Although we were not able to run the applications fully on all platforms and the applications do not test extensively possible

[9] https://developer.sony.com/imx500/imx500-key-specifications

scenarios, we believe that the results are representative enough to compare the tested platforms against each other in the six categories presented in the sections below.

4.2.1 Image Quality

Image quality is largely dependent on the camera sensor, lens, ISP pipeline, and the correct setting of the whole camera system. We have evaluated the platforms in four image-quality domains: resolution, frame rate, ISP configuration, and visual appeal.

The Hailo-15 AI Vision Processor is the best of the tested camera systems. We have opted for the LI-IMX678-MIPI-HL-118H[10] camera that can record up to 12-bit 3856x2176 video at 60 FPS. The ISP pipeline of this platform offers a wide range of manual controls like noise reduction, wide dynamic range, or dead pixel correction. Moreover, it contains automatic modes for the best setting of exposure, focus, and white balance. Although we were not able to test the configuration of the ISP pipeline due to difficulties with the compilation of software for this platform, we still liked the video product by example applications provided by Hailo AI the most. We can also confirm that this platform can provide the 3856x2176 video at the claimed 60 FPS.

The second-best tested camera system is the Raspberry Pi 5 with the Raspberry Pi AI Camera[11]. This system can capture a 10-bit 4056×3040 video at 10 FPS or a 10-bit 2028×1520 video at 30 FPS. The Raspberry Pi's ISP pipeline also offers automatic control of exposure, focus, and white balance. Other controls for color adjustments, noise reduction, or high dynamic range are available as well. Furthermore, their ISP pipeline is very well documented and available as open source for user modification. We have tested and verified both resolutions at their respective frame rates and tried to adjust the parameters of the recorded video using the ISP controls. The video produced by this camera system is visually almost on par with the one produced by the Hailo-15 camera system. The Raspberry Pi AI Camera equipped with the IMX500 1/2.3 sensor only lacks in low-light conditions in comparison to the LI-IMX678-MIPI-HL-118H camera that houses a larger IMX678[12] 1/1.8 sensor.

The third-ranked camera system is the SCAiLX 2 MP Color Global Shutter Camera[13] integrated in the SCAiLX Development-Kit. The data sheet claims that the camera can produce a 1920x1200 video at up to 120 FPS. Although this camera system's ISP pipeline also provides automatic exposure and white balance, as well as many other configurable controls, the video produced does not visually match the quality of the previous two systems. This is partly caused by the smaller available resolution and partly by an even smaller 1/2.6 AR0234CS[14] image sensor. Moreover, the maximum number of FPS that we achieved with this camera system was 60 when not processing the video at all. Simple H.264 compressed video streaming over Ethernet achieved only 38 FPS, while an inference of an SSD MobileNetV1 model, followed by image cropping, and H.264 compression ran only at 16 FPS.

That leaves the Up Squared Pro board in fourth place. The version of our Up Squared Pro board does not support any specific camera, nor have we tested any unofficial ones. We also have not been able to find any documentation regarding ISP on this board. However,

[10] https://leopardimaging.com/wp-content/uploads/2024/05/LI-IMX678-MIPI-HL-118H_Datasheet.pdf
[11] https://datasheets.raspberrypi.com/camera/ai-camera-product-brief.pdf
[12] https://www.sony-semicon.com/files/62/flyer_security/IMX678-AAQR_AAQR1_Flyer.pdf
[13] https://www.videologyinc.com/product-guide?store-page=SCAiLX-2MP-Global-Shutter-Camera-p657537518
[14] https://www.onsemi.com/parametrics/AR0234CS/create-overview-pdf

this board is equipped with an FPGA, which can be well utilized for custom video frame processing at large frame rates.

4.2.2 Deep Learning Capabilities

The Raspberry Pi 5 system clearly has the most capable hardware for deep learning (DL) inference. Having two DL inference accelerators coupled with a quad-core application processor enables running a three-stage DL pipeline, e.g., detection of objects on the Raspberry Pi AI Camera, their classification on the Hailo-8 AI Accelerator, and possibly further classification or other DL tasks with a full floating-point model executed on the processor. Furthermore, each DL inference stage can be interleaved by a processor stage, which can implement any necessary pre- and post-processing tasks, like image crops and re-scaling, required between the DL stages. Therefore, we rank this system first in this category.

It is difficult to determine the second-ranked system in the DL capabilities category. The Up Squared Pro board, combined with the Hailo-8 AI Accelerator, can deliver better on-paper performance. On the other hand, the tight integration of the network core, vision subsystem, digital signal processor, and application processor on a single chip in the Hailo-15 AI Vision Processor allows the implementation of similar pipelines as described above, while the Hailo-8 AI Accelerator can run only a single neural network model at a time. To be precise, multiple models of neural networks can be merged together before compilation and then run simultaneously on the accelerator, or multiple separately compiled models can run in a round-robin schema on the accelerator alternately. However, the option to run DL stages directly interleaved with some algorithmic processing is, to our best knowledge, not possible[15]. Consequently, given that Hailo-15 AI Vision Processor is targeted at DL video processing, which we are primarily interested in, we rank it second and the Up Squared Pro system third. Nevertheless, for other DL tasks, e.g., text or sensor data processing, the Hailo-8 AI Accelerator could be superior.

That leaves the Videology SCAiLX Development-Kit fourth. Although this system can run the same DL models as the aforementioned ones, the much lower on-paper DL performance is noticeable. The inference times are significantly longer, which reduces the maximum throughput of the system, as already touched upon in the previous section. Any pipelining of DL tasks would also be hardly possible with the limited DL processing power.

4.2.3 General Computing Power

We have not performed any elaborate testing of the computing power of each device. It would be difficult to design a representative benchmark because some of the devices have additional hardware accelerators other than the general processing unit. Therefore, we have based the following ordering on the available datasheet information and our judgment.

We rank the Raspberry Pi 5 with its quad-core processor running at 2.4 GHz[16] and 8 GB of RAM first, we value general computing power the most in this category. Consequently, we place the Up Squared Pro board second, having a quad-core processor running at 2.0 GHz in

[15]Of course, pipelining in a sense that the output of one of the models is processed on an application processor and later used as an input to another of the models might be possible. This approach is utilized for image enhancements in Section 5.2.

[16]Processor clock frequency is a good indicator of how fast it generally performs a certain algorithm. Although we are comparing an Intel processor against Arm processors with CISC and RISC, respectively, the embedded processors do not differ that significantly, e.g., the SSE (Intel) and Neon (Arm) vector instruction sets are very similar.

burst mode coupled with 4 GB of RAM and an FPGA that can deliver additional general computing performance. Next, that is the third, is the Hailo-15 AI Vision Processor. Although its quad-core processor only runs at 1.3 GHz with 4 GB of RAM, the additional digital signal processor offering 256 multiply-accumulate units will be able to accelerate many data processing algorithms. The Videology SCAiLX Development-Kit has a quad-core processor operating at 1.6 GHz, which is more than the Hailo-15 AI Vision Processor can deliver, but it does not have any additional acceleration. Its RAM is the smallest of the tested systems at 2 GB. Therefore, we rank it as fourth in this category.

4.2.4 Power Consumption

Power consumption is an important aspect of a device, mainly when its primary operation is battery-powered. We summarize the power requirements at high compute loads based on datasheet information, information found online, and our limited testing in Table 4.1.

Device	**Component**	**Component Power Consumption [W]**	**Overall Power Consumption [W]**
Hailo-15 AI Vision Processor	board	7.50	8.25
	LI-IMX678-MIPI-HL-118H camera	0.75	
Up Squared Pro	board	20.00	24.00
	Hailo-8 M.2 AI Accelerator Module	4.00	
Videology SCAiLX	Development-Kit including camera	5.50	5.50
Raspberry Pi	Raspberry Pi 5 board	6.50	11.75
	Raspberry Pi AI HAT+[17]	4.00	
	Raspberry Pi AI Camera	1.25	

Table 4.1: Power consumption of tested systems and their components

4.2.5 Implementation Difficulty

We must place the Raspberry Pi ecosystem first in terms of implementation difficulty. They provide extensive documentation on how to use various cameras and accelerators with their boards, as well as on how to configure their ISP pipeline. Their software libraries for control of the devices and ISP are open-source, which allows user modifications if necessary. Furthermore, they offer a wide variety of software examples that can be modified for a specific use case.

Second-placed is the Videology SCAiLX Development-Kit. Although their development kit and ISP are documented on the Raspberry Pi's level, they do not provide as many software examples and do not have as large a user community that could help with eventual issues. Nonetheless, we were able to test this platform without encountering any difficulties.

We rank the Up Squared Pro board third. Any specific documentation for this board was not needed, because it behaves like a desktop computer. We used code examples for the Hailo-8 AI Accelerator and Linux libraries provided by Hailo AI to implement applications leveraging the accelerator for deep learning inference. We were able to successfully run these applications after overcoming some initial issues with compilation and linking.

[17]The Raspberry Pi AI HAT+ contains the Hailo-8 AI Accelerator chip.

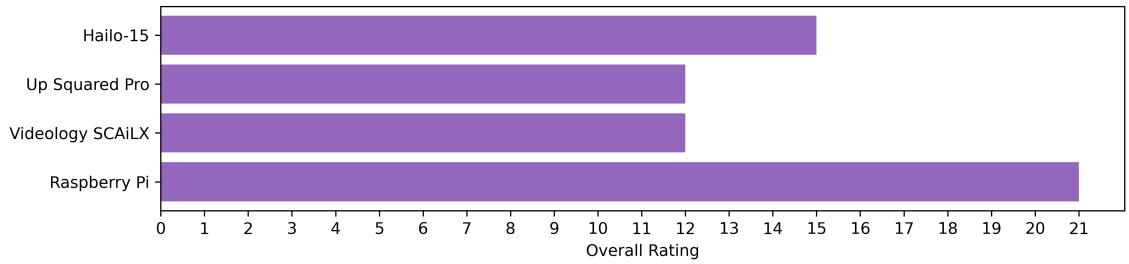


Figure 4.1: Overall ranking of selected platforms – higher means better

The Hailo-15 AI Vision Processor must be ranked fourth in this category. The available documentation for this platform is not of the highest quality, and the supporting software for the compilation of test applications did not work. We have encountered issues with missing files and outdated links to external files, which we have not yet been able to resolve fully. Furthermore, neither our posts on the Hailo Community forum nor an official ticket created on the Hailo AI website were answered.

4.2.6 Purchase Price

The purchase price, although often of development parts, is a good indicator of the price for the final system when sold in smaller quantities. We summarize the purchase prices of individual components and the prices of the final systems in Table 4.2.

Device	Component	Component Price [$]	Overall Price [$]
Hailo-15 AI Vision Processor	4 GB RAM board	402.00	701.00
	LI-IMX678-MIPI-HL-118H camera	299.00	
Up Squared Pro	4 GB RAM board	319.00	518.00
	Hailo-8 M.2 AI Accelerator Module	199.00	
Videology SCAiLX	2 GB RAM Development-Kit	1299.00	1299.00
Raspberry Pi	Raspberry Pi 5 8 GB RAM board	80.00	283.75
	Raspberry Pi AI HAT+[18]	133.75	
	Raspberry Pi AI Camera	70.00	

Table 4.2: Purchase prices of tested systems and their components

4.3 Chapter Summary

In this chapter, we described and tested four promising platforms: Hailo-15 AI Vision Processor, Up Squared Pro board with Hailo-8 AI Accelerator, Videology SCAiLX Development-Kit, and Raspberry Pi 5 with Hailo-8 AI Accelerator and Raspberry Pi AI Camera; in six categories: image quality, deep learning capabilities, general computing power, power consumption, implementation difficulty, and purchase price. The overall ratings computed as sums of scores in the individual categories are depicted in Figure 4.1. To better understand the multi-objective optimization problem and to determine dominated solutions as well as

[18]The Raspberry Pi AI HAT+ contains the Hailo-8 AI Accelerator chip.

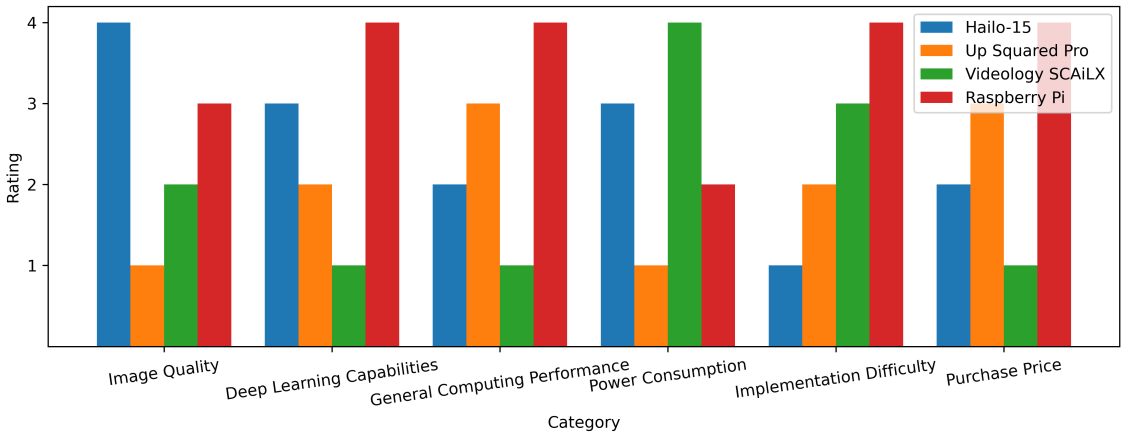


Figure 4.2: Ranking of selected platforms in tested categories – higher means better

Figure 4.3: Head-to-head comparison of selected platforms in tested categories (dots in each head-to-head comparison represent the following categories from left to right: image quality, deep learning capabilities, general computing power, power consumption, implementation difficulty, and purchase price)

Pareto-optimal solutions, we also provide ratings of the devices for each tested category in Figure 4.2, while Figure 4.3 depicts head-to-head per-category comparisons of the devices. The Up Squared Pro board solution is the only dominated one, while the others are on the Pareto front. Despite the fact that we did not analyze, e.g., temperature ratings of the devices for their potential use outside or in other harsh conditions, we are now confident

to answer our posed questions from the beginning of this chapter. Unfortunately, we must conclude that the Hailo-15 AI Vision Processor and the Up Squared Pro board both have blocking issues that prevent us from using them in further development. In the first case, it is caused by the outdated supplementary software, while the latter platform does not have a suitable camera. The Videology SCAiLX Development-Kit is a well-integrated, all-in-one system, but it mainly lacks deep learning capabilities for more complex applications. Consequently, supported by its overall rating, we chose the combination of Raspberry Pi 5 with Hailo-8 AI Accelerator and Raspberry Pi AI Camera as the platform to proceed with for developing a complex deep-learning-based application. In regard to other use cases for the platforms that we will not further use, we conclude the following. The Hailo-15 AI Vision Processor seems to have the capabilities to run similar applications as described in Chapter 5. The Up Squared Pro is a great platform for testing and debugging applications during development, but it is not the best platform to be used as a smart camera system. Last, the Videology SCAiLX Development-Kit in its non-development version can be used to implement a smart camera that does not require high frame rates and is placed in a well-lit environment, e.g., to detect anomalies on a production line in a factory.

Finally, we now have a theoretical understanding of image processing and computer vision tasks from the first chapter, we have analyzed quantization for deep learning models on edge devices in the second chapter, and we have selected an edge platform to implement a deep-learning-driven computer vision application on in this chapter. Consequently, the next chapter focuses on the implementation of such an application.

Chapter 5

Smart Camera Application

The initial intention was to build a computer vision system to detect and track people, simulating a cameraman by tracking sportsmen, or for security applications. Since we have designed and selected a powerful embedded system, we have extended the goal not only to detect and track objects, but also to recognize attributes of the tracked objects. Our final application will detect and track vehicles, detect their registration plates, and recognize the plate numbers. We believe this application can better demonstrate all the capabilities of the selected hardware. To prove that and beyond, we again ask ourselves several research questions:

- *Are there any first-hand limitations that could be resolved by changing some of the selected hardware components?*
- *Is the use of state-of-the-art pre-trained neural network models sufficient, or could the application benefit from designing custom deep learning models?*
- *To what extent can deep-learning-based image enhancement improve subsequent tasks on the enhanced images?*

The following sections not only try to answer the posed questions but also present two deep learning pipelines. First, a pipeline primarily focused on the best quality of the registration plate recognition in a reasonably well-lit environment, with image signal processing-focused enhancements. Second, a pipeline that can operate in dark conditions and leverage some of its deep learning capacity for both image enhancement and registration plate recognition.

5.1 Image Signal Processing Focused Deep Learning Pipeline

This section describes a deep learning application focused on the best possible registration plate reading results, a task called optical character recognition (OCR), and their association with specific vehicles under daylight conditions.

We first present the models of the deep learning CNNs used. This is followed by a description of algorithms required for pre- and post-processing between the deep learning stages. The third subsection depicts the assignment of the deep learning and algorithmic tasks on the available hardware components. Then, we talk about image enhancements mediated via the configuration of the image signal processing pipeline that aim to improve registration plate recognition. Last, we describe improvements that have already been

implemented and suggest other possible improvements that can be addressed in future research.

5.1.1 Deep Learning Models

Our detection and OCR pipeline contains three deep learning models of the following convolutional neural networks:

1. **You Only Look Once version 8 size N (YOLOv8n)**: trained for detection of vehicles on 320x320 pixels large images. We have used a subset of the COCO [58] dataset, extracting images that contain at least one occurrence of the car, motorcycle, bus, or truck classes. Furthermore, we have extended this dataset with images from the Front and Rear Images of Car [53] dataset to ensure better generalization.

2. **You Only Look Once version 8 size S (YOLOv8s)**: trained for registration plate detection on 640x640 pixels large images. The training data were composed of images from the Large License Plate Detection Dataset [26] and the Vehicle Registration Plates Computer Vision Project [93] dataset.

3. **Mobile Vision Transformer version 2 (MobileViTv2)**: downloaded pre-trained for OCR of European registration plates from the fast-plate-ocr[1] GitHub repository.

5.1.2 Pre- and Post-processing Algorithms

Each of the deep learning inference stages must be interleaved by the processing of results from the previous stage and the preparation of the inputs for the next stage. The first deep learning stage is performed on the whole captured frame, rescaled to the 320x320 pixels input size of the vehicle detection model. The processing of the results and preparation of the inputs for the registration plate detection model are performed in the following steps. First, we select only the detection results with higher confidence than a preset threshold. Second, the bounding box coordinates are converted to have the same aspect ratio by enlarging the shorter detected side of each bounding box. Third, the bounding boxes are passed to a simple intersection over union tracker, i.e., we are trying to associate the same vehicle across multiple frames. Fourth, we crop and rescale the detected patches with vehicles to the 640x640 pixels input size of the registration plate detection model.

The results of the registration plate detection model are again bounding box coordinates and their confidence scores. Here, we simplify and for each vehicle select a detection with the highest confidence score or no detection if none surpasses a preset threshold. This simplification is based on the assumption that each vehicle has, at most, a single visible registration plate. Although this is a wrong assumption, as we will discuss and improve later in this chapter. The selected bounding box coordinates are rescaled and shifted to the full-sized frame, which enables us to obtain the registration plate crops in full resolution. Each registration plate crop is then converted to a grayscale image before being passed to the OCR model.

The OCR model outputs a list of characters and a list of confidence scores associated with each character. Since we are tracking the vehicles across multiple frames, we can combine more OCR results of a registration plate belonging to the same vehicle to increase the chance of its accurate recognition. We score symbols at each position of a registration

[1] https://github.com/ankandrew/fast-plate-ocr

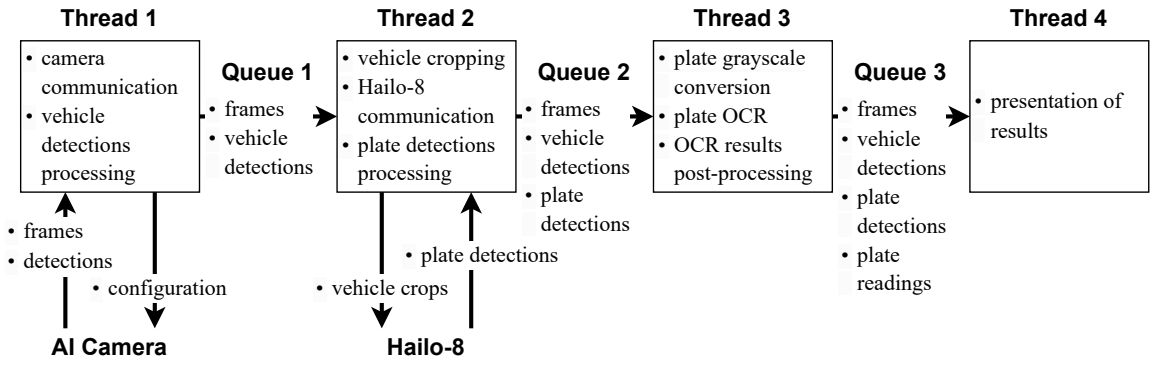


Figure 5.1: Multi-threaded pipeline software architecture with a limited-sized queue communication between the compute threads

plate with the number of occurrences at that position weighted by the occurrence confidences, and finally select the symbols that have the highest weighted occurrence scores at each position. In other words, we sum the confidence scores of symbols at each position across multiple frames and select the symbols with the highest accumulated confidence at each symbol position once the tracked vehicle no longer appears in the captured frames. Furthermore, we are monitoring a drop in the accumulated confidence for symbols near the end of the registration plate number. If the confidence between two consecutive symbols drops significantly, the recognized plate number is truncated.

The last algorithmic processing consists of a variety of options for storing, transmitting, or displaying the captured video and the semantic information obtained by the deep learning models. We primarily store the video frames as separate images and serialize the semantic data into labels stored in the JSON format. This enables us to perform post-mortem analysis of the collected data and also to generate annotated videos for visualization if necessary. Another option is to annotate the video in real time, stream it over Ethernet or Wi-Fi, and display it on a host machine. However, the streaming requires H.264 compression, which is computationally demanding, leading to smaller achievable frame rates. Moreover, this approach also requires carrying a host machine when using the camera system. Therefore, we have also tried equipping the camera system with a small 640x480 pixel display. Displaying data on a built-in display does not require any video compression, which leads to better frame rates. On the other hand, the display consumes a non-negligible amount of energy and does not provide the best visual quality under bright sunlight.

5.1.3 Software-to-Hardware Assignment

It is important to not only select appropriate hardware but also to utilize it with well-written software correctly. Our application is of a pipeline nature, i.e., we collect images, process them in multiple steps, and store, display, or stream them. If any stage of the application cannot operate on the required frame rate, it creates a bottleneck, ultimately slowing the other stages. Therefore, we have tested various assignments of the deep learning and algorithmic stages described in the two previous sections to come to a conclusion depicted by diagrams in Figures 5.1 and 5.2 from the software and hardware points of view, respectively.

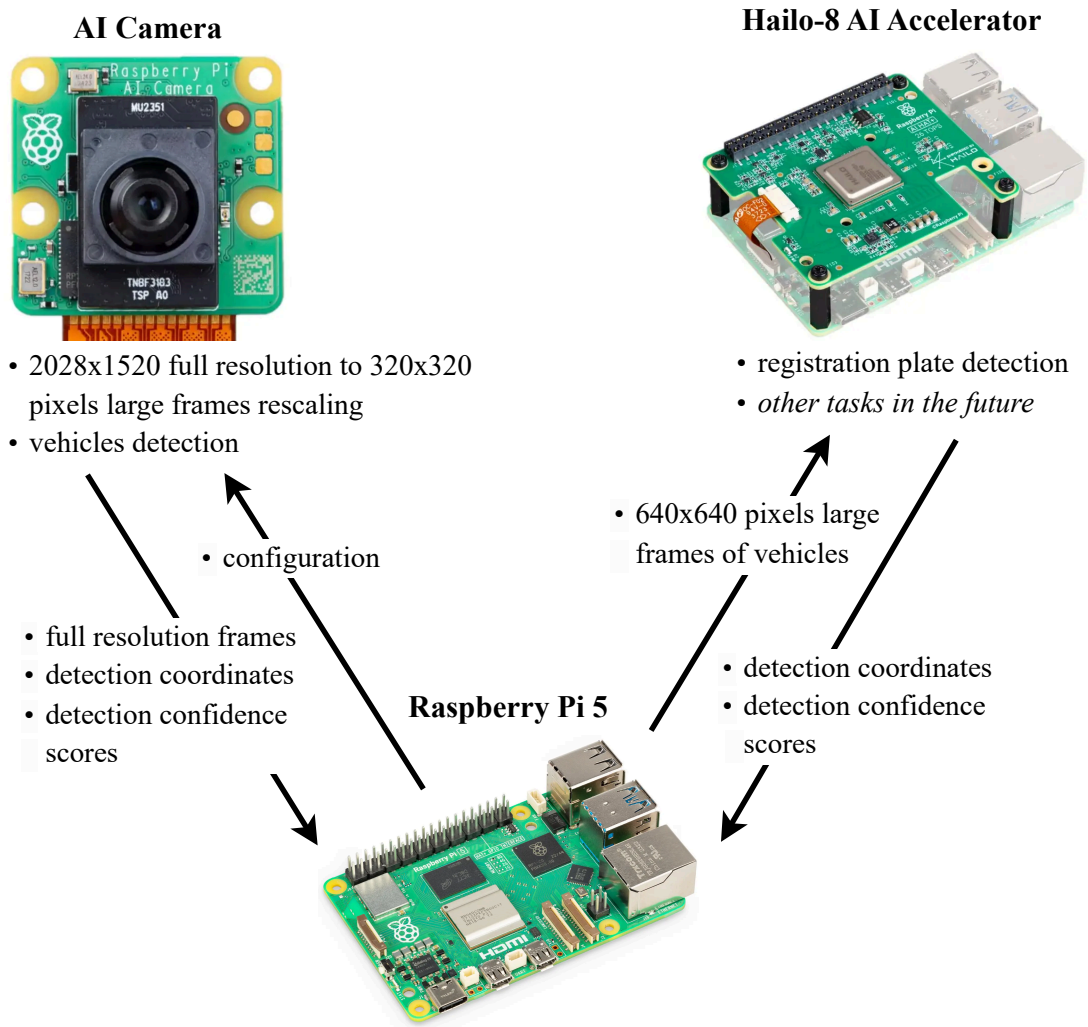


Figure 5.2: Assignment of software tasks to hardware components

5.1.4 Image Enhancements

When selecting our hardware platform, we have carefully looked for a configurable ISP pipeline because it is the best to tune images for a specific need. In our case, we prioritize images that will result in high accuracy of registration plate recognition over visual appeal to people. Already early on during our experiments, we have noticed that the registration plate recognition quality is strongly related to the shutter speed of the camera. When the shutter speed is slow, i.e., the sensor is exposed to the incoming light for a longer time, approximately 25 ms, the images become blurry enough to affect the recognition quality while

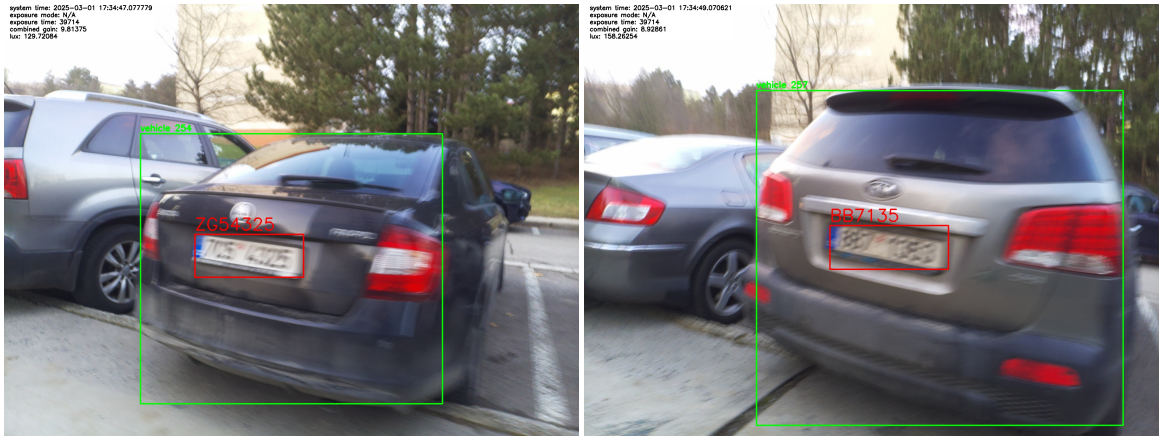


Figure 5.3: Failed registration plate readings due to unnecessarily long exposure times in hand-held footage captured at walking speed, see Figure A.1 for more examples

shooting a hand-held video at a walking speed. Selected frames of two failed registration plate readings due to unnecessarily long exposure times are captured in Figure 5.3.

Consequently, we have configured the ISP pipeline aggressively, always first to maximize the analog and digital gains over increasing the exposure times, which are always capped at a maximum of 20 ms. Furthermore, we use three exposure modes: Short, Normal, and Long, in which the pipeline can operate. To restrict the ISP pipeline in increasing exposure times even further, we switch between the modes based on a running average of lighting conditions measured in lux[2] by the camera sensor over 512 frames. Table 5.1 summarizes the allowed exposure time and the combined gain ranges of the three ISP modes with the illuminance that restricts their usage. Moreover, as can be seen from the illuminance running average ranges, we also implement hysteresis to prevent oscillation between the modes on the range boundaries. Selected frames of the same scene as above, now with correct registration plate readings, are shown in Figure 5.4.

Mode	**Exposure Time Range [ms]**	**Combined Gain Range [dB[3]]**	**Illuminance Running Average Range [lx]**
Short	$\langle 0.001; 1.0 \rangle$	$\langle 0; 27.6 \rangle$	$\langle 500; \text{inf} \rangle$
Normal	$\langle 1.0; 4.0 \rangle$	$\langle 27.6; 27.6 \rangle$	$\langle 60; 525 \rangle$
Long	$\langle 4.0; 20.0 \rangle$	$\langle 27.6; 27.6 \rangle$	$\langle 0; 75 \rangle$

Table 5.1: Image signal processing pipeline exposure times and analog gains configuration given lighting conditions

We must, however, disclose that the registration plate readings, even with the default ISP configuration, are typically correct. The above examples are rather rare occasions. This only proves that the confidence-based post-processing of the OCR results provides a major benefit. Some of the eventually correct registration plate readings are displayed in Figure 5.5. Notice how the readings gradually improve.

[2]Lux (lx) is the unit of illuminance, measuring the amount of light (lumens) falling on a surface per square meter.

[3]Decibel (dB) is a logarithmic unit expressing the ratio of two values computed as $20 \cdot log_{10}(\frac{output}{input})$.

Figure 5.4: Correct registration plate readings with configured image signal processing pipeline for shorter exposure times in hand-held footage captured at walking speed, see Figure A.2 for more examples

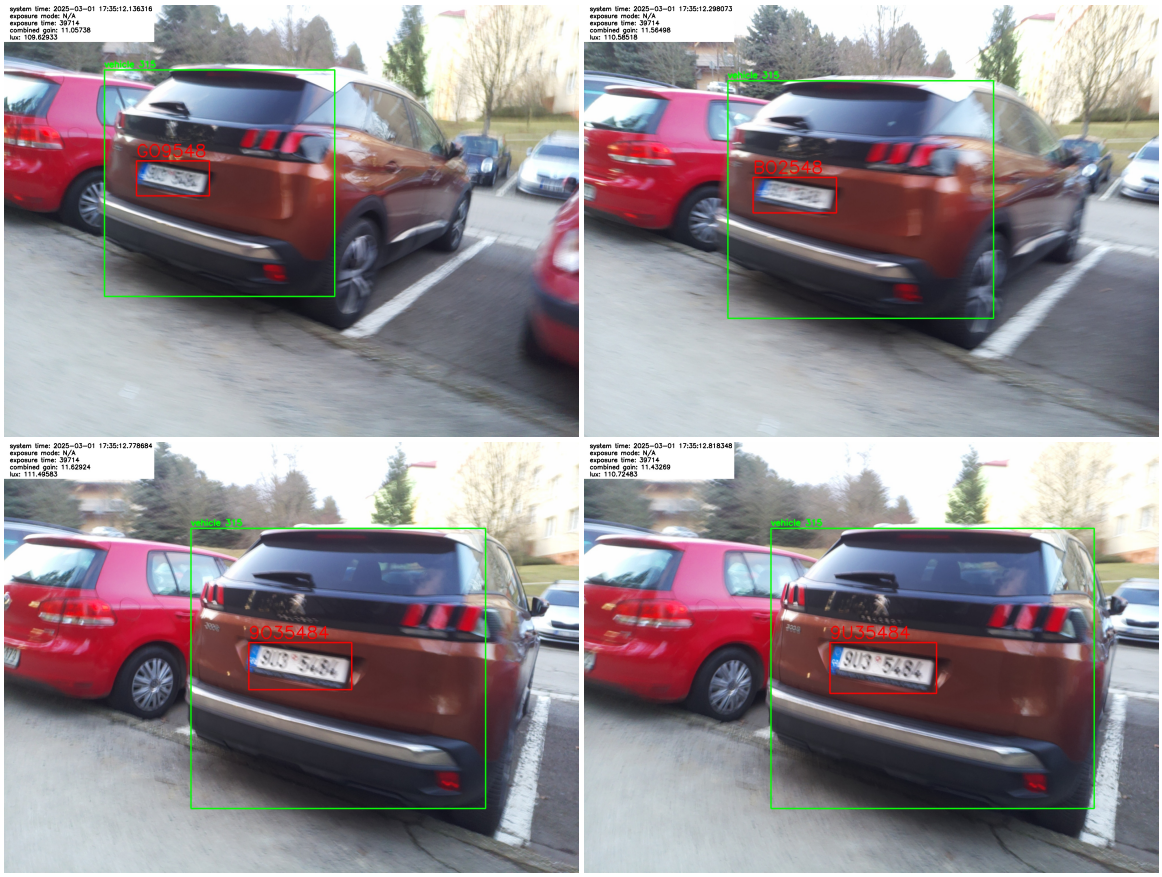


Figure 5.5: Eventually correct registration plate readings with default image signal processing pipeline settings in hand-held footage captured at walking speed, see Figure A.3 for more examples

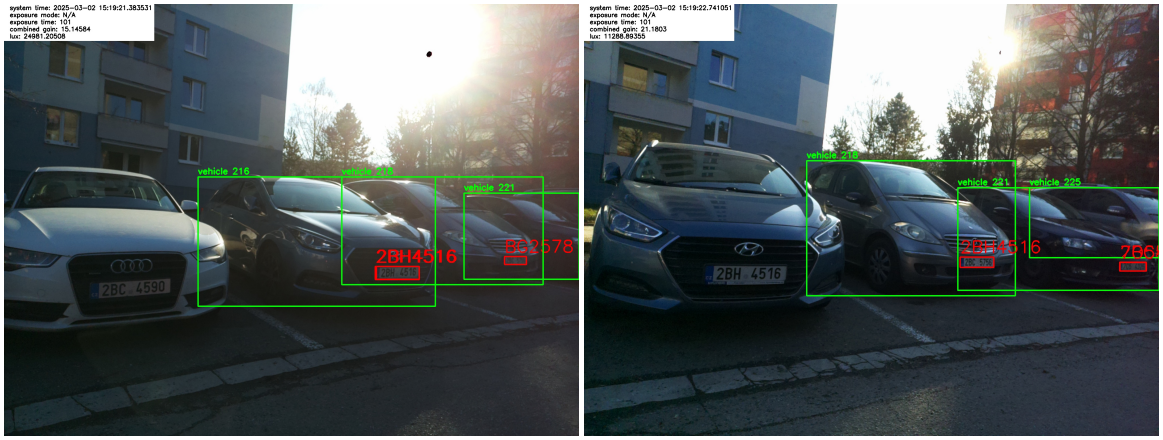


Figure 5.6: General vehicle detection model causing overlaps leading to incorrect registration plate assignment, thus their incorrect readings in hand-held footage captured at walking speed, see Figure A.4 for more examples

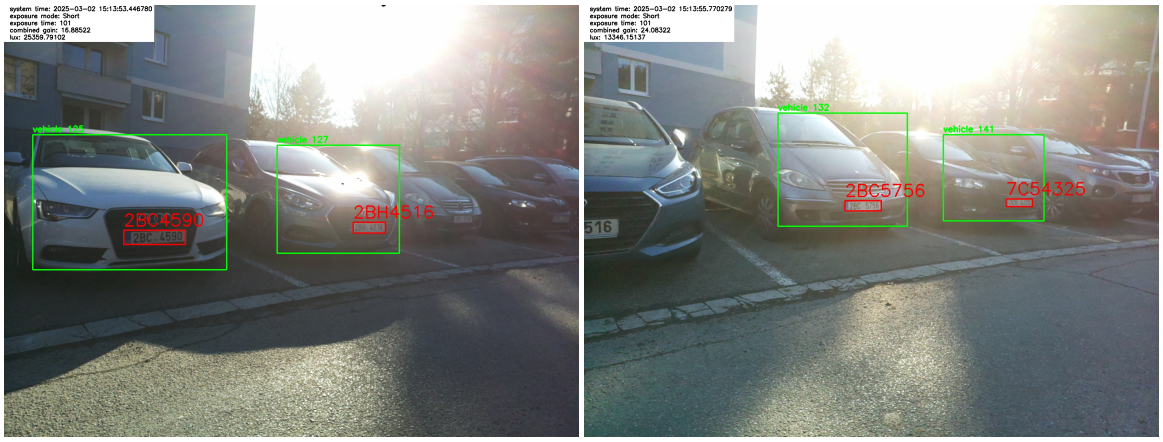


Figure 5.7: Fine-tuned vehicle detection model detecting only vehicle fronts, leading to correct registration plate assignment and readings in hand-held footage captured at walking speed, see Figure A.5 for more examples

5.1.5 Already Addressed and Future Improvements

The most noticeable issue when reviewing the collected footage during daylight conditions comes from the vehicle detection part of the application. As mentioned above, we have trained the vehicle detection model on two general datasets, where partially occluded vehicles are also annotated. This leads to overlapping vehicle detections, thus breaking our aforementioned assumption of one registration plate per vehicle, which, in turn, causes registration plates not belonging to a given vehicle to be detected, as depicted in Figure 5.6. Consequently, our confidence-based OCR results post-processing algorithm cannot work properly. Therefore, we have opted to annotate our own vehicles-front-and-rears dataset and further fine-tune the vehicle detection model. The dataset contains images collected by us and re-annotated images from the COCO [58] and UC3M-LP[77] datasets to increase its variability and decrease the chance of overfitting. Finally, the improvement achieved by the fine-tuning can be seen in Figure 5.7, capturing the same scene in close time succession.

The second refinement of our application addresses hardware utilization. Although the application currently does not suffer from any bottlenecks, we find the Hailo-8 AI Accelerator underutilized. That is, the current registration plate detection model runs on unnecessarily large vehicle crops and can detect an arbitrary number of registration plates. Since we have now shown with the updated vehicle detection model that we can assume a single registration plate per detected vehicle, the current model could be replaced by a different one. This model can run on 320x320 pixels large crops instead of the current 640x640 size, therefore be deeper with more parameters, and perform, e.g., the following tasks:

- single registration plate localization with a confidence score,
- manufacturer and model classification,
- vehicle type classification,
- color classification,
- orientation (front/rear) classification.

Designing such a capable neural network and training its model is a complex task worthy of future research. We do, nevertheless, suggest a baseline architecture and its training process. Our proposed neural network is based on a shared backbone[4] taken from a proven neural network such as one of the ResNet or YOLO families discussed in Section 2.4.3. The backbone is followed by a regression head[5] for the localization task, and classification heads for the remaining listed tasks. All of the heads can be very similar, again taking inspiration from the aforementioned proven neural networks. They will only differ in the number of outputs, output layer activation functions, and loss functions used during training. The latter two differences are likely to be the following:

- detection (registration plate localization) – Sigmoid activation function for all outputs with the IoU Loss for maximizing the overlap between the predicted and ground truth boxes and the BCE Loss for the confidence training,
- binary classification (vehicle rear/front) – Sigmoid output activation function followed by the BCE Loss,
- general classification (remaining tasks) – Softmax activation function for all outputs combined with the CCE Loss.

The training must be performed in two phases. All of the heads must be trained separately in the first phase while the backbone is frozen[6], effectively used only to generate two-dimensional embeddings containing important information about the input samples. An efficient approach would be to first generate the embeddings for all samples in a used dataset and then not use the backbone while training the heads at all. Once all the heads

[4]Backbone contains early layers of a complex neural network. It typically extracts general information about the input. In the case of computer vision, those are two-dimensional convolutional layers producing feature maps that are activated, e.g., in areas with edges or corners close to the input and with more complex features, like faces, in deeper layers.

[5]Head consists of layers near the end of a neural network and of its output, e.g., in case of classification, an array of probabilities predicting the presence of the learned classes.

[6]Gradients are not back-propagated through the backbone. Therefore, the weights are not updated.

are converged, the second training phase can begin. First, the backbone and the heads must be assembled into one large neural network. Then, the training can continue, now with the combined gradients from all the heads also back-propagated through the backbone.

The third improvement lies in the use of other cameras. Although the Raspberry Pi AI Camera brings high-quality images and additional deep learning performance, it does not perform the best in low-light conditions, having only 1.55 µm large pixels, nor can it be used for recognition of registration plates on distant vehicles with its small 4.74 mm focal length lens. The current camera could be replaced or supplemented, e.g., by the Raspberry Pi Global Shutter Camera[7], which has much larger 3.45 µm pixel size for better low-light performance and can be equipped with a variety of lenses to focus objects at a specific distance. The global shutter further helps when capturing moving objects. We believe that the best approach is to use the combination of the Raspberry Pi AI Camera for detection, even of distant vehicles, and the Raspberry Pi Global Shutter Camera for registration plate detection and recognition. However, this will require software for the alignment of the two cameras. The software does not need to run constantly. Calibration during startup could be sufficient if the cameras are mounted immovably during operation. The traditional computer vision SIFT or SURF algorithms briefly presented in Section 2.4.1 are great candidates to start with. These algorithms locate specific features in an image and assign them descriptors. Using the descriptors to match features from two different images could enable their near-perfect alignment. Consequently, the detection and OCR pipeline could be adjusted to use two cameras, switching between them for the registration plate detection and recognition based on the position of a detected vehicle. Another approach, purely software-based, that can improve the application in low-light conditions and for distant detections is to recover some of the missing information using deep learning models. We try that with a modified deep learning pipeline in the next section.

5.2 Image Enhancement Focused Deep Learning Pipeline

The second deep learning application aims to further improve the above-presented one. The task remains the same, i.e., detect registration plates with a detection model and recognize them with an OCR model, but the circumstances differ. Now, the application pipeline is redesigned to also support non-ideal conditions.

We further divide this section into three experiments. First, a super-resolution experiment in which we are testing whether an increased resolution of distant registration plates can improve their recognition results. Second, a low-light experiment in which we are trying to determine whether running low-light enhancements prior to the registration plate detection and recognition can improve the results. Third, an experiment comparing an updated registration plate detection and OCR pipeline for low-light conditions with the original deep learning pipeline.

5.2.1 Super-Resolution Experiment

The goal of this experiment is to increase the resolution of detected registration plates with a deep learning model and determine whether it leads to better recognition results. We have modified the main application pipeline for this experiment in the following ways:

1. We do not perform vehicle detection to simplify the pipeline.

[7] https://www.raspberrypi.com/products/raspberry-pi-global-shutter-camera/

2. We configure the camera and image signal processing pipeline to provide two video streams of 640x640 pixels and 1280x1280 pixels resolution, respectively.

3. We only detect and recognize plates that are between 35 and 105 pixels wide in the lower-resolution stream.

4. The OCR is applied on four variants of the same registration plate: directly on the lower-resolution crops, on two times enlarged crops by a super-resolution model, on two times enlarged crops by a bicubic algorithm, and on crops from the higher-resolution video stream.

5. We do not track the detected plates and do not improve their readings by combining results from multiple frames.

We are using the ESPCN [90] presented in Section 2.3.1, specifically trained for enlarging registration plate crops collected by ourselves in previous experiments. A comparison of the registration plate recognition of the four OCR variants is displayed in Figure 5.8. We have selected a scenario in which the deep-learning-based super-resolution improves the recognition results, i.e., at a specific distance when approaching or moving away from vehicles. However, the registration plate recognition performed on the higher-resolution video stream provides the same or better results. Therefore, we must conclude that the deep-learning-based super-resolution approach can provide benefits only in scenarios where a higher-resolution video stream is not available. The idea of similarly enlarging registration plate crops that are 35 to 105 pixels wide in higher resolution emerges to be logical in improving the maximum operating distance of the system. Unfortunately, this is not, for the most part, possible since the registration plates are already out of focus at such distances when using the Raspberry Pi AI Camera.

5.2.2 Low-Light Enhancement Experiment

This experiment aims to determine whether the use of a low-light enhancement deep learning model leads to improvements in the detection and recognition of registration plates in dark environments. We have modified the main application pipeline to perform this experiment as follows:

- We do not perform vehicle detection to simplify the pipeline.
- We configure the camera and image signal processing pipeline to operate in a high dynamic range mode, providing alternately short and long exposure frames[8]
- The short exposure frames are enhanced by a low-light enhancement model. Thus, we effectively generate a third frame.
- We decrease the confidence of a successful registration plate detection to a smaller value to improve recall in the not ideal conditions and perform the detection and OCR on the three frames separately.
- We do not track the detected plates and do not improve their readings by combining results from multiple frames.

[8]Note that the short and long exposure frames are not the same frames. They are taken right after each other, but still can significantly differ, mainly in sharpness, due to the movement of the camera.

Figure 5.8: Readings of low resolution (left upper), 2x super-resolution (right upper), 2x bicubic interpolation (left lower), and 2x actual crops (right lower) of registration plates across multiple frames in hand-held footage, see Figure A.6 for more examples

We are using a pre-trained model of the DCE-Net [33] discussed in Section 2.3.1 adjusted to process 640x640 pixels large frames. The results of this experiment are much more promising for further incorporation into the main application pipeline. Consequently, we have performed a deeper analysis comparing the recognition quality of the three frame variants. We counted 313 perfectly correct registration plate readings in the short exposure, low-light enhanced frames. The registration plate recognition of long exposure frames achieved 204 perfectly correct results, while there were no, or very close to zero, correct readings in the short exposure frames. In total, there were 441 perfectly correct registration plate readings in at least one frame of the triplets. However, we must disclose that the results

Figure 5.9: Correct registration plate readings in enhanced short exposure frames (middle) during the civil twilight period in hand-held footage captured at slow walking speed, see Figure A.7 for more examples

Figure 5.10: Correct registration plate readings in long exposure frames (right) during the civil twilight period in hand-held footage captured at slow walking speed, see Figure A.8 for more examples

were collected from 4585 frame triplets[9] that had at least one registration plate detection, although often false positives or completely unrecognizable due to blurriness. Nevertheless, we conclude the experiment as successful, i.e., the enhancement of the short exposure frames enables correct registration plate recognition, which, on top of that, significantly outperforms the recognition on the long exposure frames. Examples of perfectly correct registration plate readings in the enhanced frames, long exposure frames, and in both are shown in Figures 5.9, 5.10, and 5.11, respectively. Note that the examples were collected towards the end of the civil twilight[10] period, which was from 17:44 to 18:16 on the day of conducting the experiment.

5.2.3 Low-Light Deep Learning Pipeline

Encouraged by the results of the above low-light enhancement experiment, we improved the main deep learning pipeline application to work also during nighttime. Initially, we updated

[9] That is, 9170 recorded frames by the camera.

[10] Civil twilight is the period just after sunset when the Sun is below the horizon but still illuminates the sky enough for most outdoor activities.

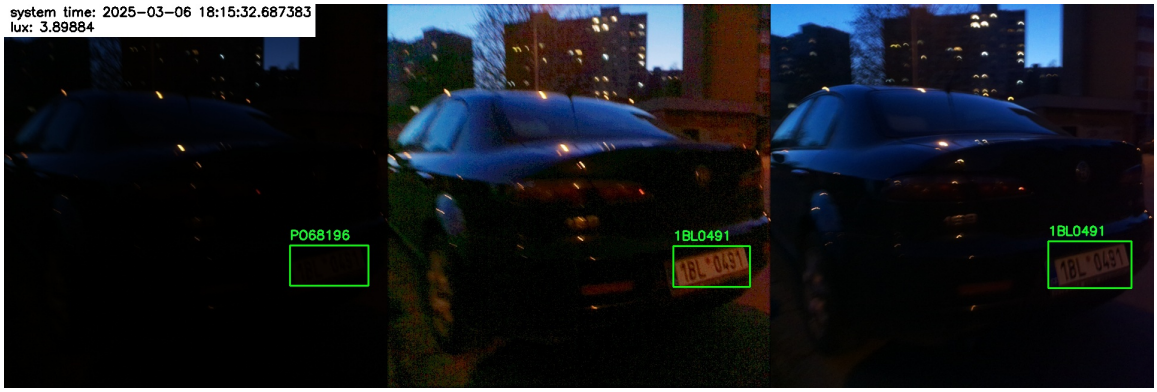


Figure 5.11: Correct registration plate readings in both enhanced short exposure (middle) and long exposure frames (right) during the civil twilight period in hand-held footage captured at slow walking speed, see Figure A.9 for more examples

the application to use the HDR long and short exposure frames with low-light enhancement prior to the registration plate detection and recognition. Later, we also upgraded the hardware setup with artificial lights and, by conducting follow-up experiments, further improved the application.

The first update to the main application resides in the following:

1. We do not perform vehicle detection to avoid false negative detections at the beginning of the pipeline.
2. We configure the camera and image signal processing pipeline to operate in an HDR mode, providing alternately short and long exposure frames in 2028x1520 pixels full resolution as well as 640x640 pixels low resolution for the subsequent deep learning tasks.
3. We enhance the low-resolution short exposure frames with a low-light enhancement model.
4. Registration plates are detected in both the low-resolution long exposure and enhanced frames.
5. We track the detected registration plates across subsequent frames instead of tracking detected vehicles.
6. The registration plate recognition is performed on the full-resolution long exposure frame and on the enhanced frame. Since the enhanced frame is of low resolution, we also perform a super-resolution of the detected registration plates if they are smaller in width than 105 pixels.
7. We keep the confidence-based post-processing of the registration plate readings from the main application.

The deep-learning-based low-light and super-resolution image enhancements are performed by the previously used DCE-Net [33] and ESPCN [90] networks, respectively. We have collected footage of both the updated and original applications during the civil twilight period of the day to determine if there are any improvements. However, we have not performed

Figure 5.12: Correct night mode registration plate detections and recognition in hand-held footage captured at slow walking speed, see Figure A.10 for more examples

any objective analysis of the collected data since each application tends to have a different cause of incorrect registration plate readings. The original application typically fails to detect vehicles and, therefore, also registration plates, while the modified application is less susceptible to false negative detections but does not recognize the detected registration plates as well due to their blurriness caused by the lower resolution of the short exposure frames and prolonged shutter times of the long exposure frames. Overall, we observe that the updated application performs slightly better, especially when the illuminance drops below 3 lux, mainly thanks to the artificial brightening of the captured frames by the low-light enhancement model.

Although the updated application brings a noticeable improvement, it still does not perform well enough in darker conditions, like a street illuminated by street lights at night. Hence, we have also opted for a hardware upgrade, equipping our system with two cheap bicycle lights. The lights do not seem to illuminate the scene significantly to the human eye, but the measured average illuminance rises from below 1 to around 3 lux, depending on the captured scene. More importantly, the reflective coding of the registration plates lights up on the captured video, which makes their detection and recognition perfectly possible, as is visualized in Figure 5.12. Vehicle detection, nevertheless, does not work properly. Consequently, we have modified the original application to switch to a night mode when the average illuminance drops below 10 lux, in which the vehicle detections are not used, and registration plates are detected on the whole captured frame instead. The tracking algorithm is kept in place, but now tracks registration plates. Repeated experiments with this modification show that the application can correctly detect and recognize the vast majority of near registration plates, see evaluation results in Section 6.2. Moreover, the low-light enhancement is no longer needed, which simplifies the deep learning pipeline, thus allowing higher frame rates or more power-efficient operation.

5.3 Chapter Summary

We start this chapter by presenting a baseline deep learning pipeline that detects vehicles and registration plates that are recognized by an OCR model. The pipeline does the following: detects vehicles on the Raspberry Pi AI Camera using a YOLOv8n model, crops the

Figure 5.13: Battery-powered smart camera prototype

detections, passes them to the Hailo-8 AI Accelerator for registration plate detection with a YOLOv8s model, recognizes the cropped detected registration plates with a MobileViTv2 model running on the Raspberry Pi 5, post-processes the readings with a confidence-based algorithm, and finally stores or presents the results. We follow this with an analysis of the initial results, determine areas of the highest potential improvements, and realize them. The first update lies in the correct configuration of the image signal processing pipeline, i.e., compensating fast shutter speeds with large analog and digital gains, to provide sharp yet still reasonably bright images. The second important improvement is achieved by training a vehicle's front and rear detection model on a custom dataset. Next, we perform super-resolution and low-light experiments, which lead us to answers to our first and third posed questions at the beginning of the chapter. Although we can achieve better results with deep-learning-enhanced images, we come to the conclusion that the system can be more significantly improved by adding artificial lights. The final prototype of your camera system with artificial lights included is shown in Figure 5.13. Moreover, another potential improvement could be achieved by supplementing or replacing the Raspberry Pi AI Camera with a camera that can be equipped with a zoom lens and has better low-light characteristics. The answer to the second question is that the application could benefit from a custom deep learning model based on our analysis of the current hardware utilization and a proposal for a registration plate localization and vehicle classification neural network.

Overall, we have successfully utilized the acquired knowledge, findings, and selected hardware from the previous chapters to implement a real-time deep-learning-based computer vision application on a battery-powered device. The last remaining step is to test and evaluate the application, which we carry out under multiple use cases in the next chapter.

Chapter 6

Testing and Evaluation

The final chapter is dedicated to the testing and evaluation of the selected system and the presentation of final solutions. We again use the chapter's introduction to pose questions that will lead us throughout the research as follows:

- *How well does the system perform in terms of the precision, recall, and accuracy metrics?*
- *Can the system perform well under conditions that it was not primarily designed for?*
- *How can we recommend using the system and what could be the best human interaction?*

We start the chapter with an objective analysis where we collect hand-held annotated footage at various daylight conditions to compare it against human annotations. Next, we repeat the same analysis under dim and dark conditions during the twilight and nighttime of the day. This is followed by an experiment where we use the same system to detect and recognize registration plates of moving vehicles and vice versa, where we collect the footage of standing vehicles from a moving car. Then, we perform a brief ablation study in which we remove some parts of the application and observe the effects on the objective measures. Last, we present three possible levels of human interaction with the system and the required interfaces.

6.1 Daytime Precision, Recall, and Accuracy Analysis

We have collected hand-held footage of vehicles at a reasonably fast walking speed exceeding 5 km/h at various times of the day across multiple days and weather conditions. We store all the footage during these tests on the device with the detected coordinates and registration plate numbers in the form of labels for later human re-labeling. We look for three kinds of errors during the re-labeling: undetected vehicles, incorrectly detected vehicles, and incorrectly recognized registration plates. We do not perform the analysis on a per-frame basis but consider only misdetections and misreadings when a vehicle is not detected, or its registration plate is not correctly recognized across the video section in which it appears. Tables 6.1 and 6.2 contain the test results in numbers and metrics, respectively.

The small inconsistencies are induced by varying test locations, e.g., larger incorrectly detected vehicle counts are caused by perpendicularly parked vehicles along a narrow road when, at times, vehicles across the road are detected from the opposite sidewalk, while

Date and Time	Median Illuminance [lx]	Weather	Correctly Detected Vehicles	Incorrectly Detected Vehicles	Undetected Vehicles	Correct Readings
2025-03-22 13:15	15027.78	sunny	138	13	3	138
2025-03-22 16:00	4785.77	cloudy	140	5	1	136
2025-03-23 07:30	1307.70	overcast	302	2	8	301
2025-03-25 18:05	279.94	cloudy	224	12	9	222

Table 6.1: Vehicle detection and registration plate recognition evaluation in numbers

Date and Time	Precision	Recall	Accuracy
2025-03-22 13:15	0.914	0.979	1.000
2025-03-22 16:00	0.966	0.993	0.971
2025-03-23 07:30	0.993	0.974	0.997
2025-03-25 18:05	0.949	0.961	0.991

Table 6.2: Vehicle detection and registration plate recognition evaluation in metrics

larger undetected vehicle counts typically happen when capturing parallel parked vehicles due to a too sharp angle of the camera relative to the vehicle front or rear, especially when they are parked very densely. The registration plate recognition accuracy is close to perfect, not dropping below 0.971. Incorrect readings most often have just one misread symbol, which allows correction against a database of registration plates using string edit distance[1] to find probable matches. The number of incorrectly detected vehicles could be decreased by finding an appropriate minimum vehicle size relative to the frame width and height in pixels. Finally, the number of undetected vehicles should be reduced by adding new training samples of the problematic vehicles to our dataset presented in Section 5.1.5.

6.2 Low-Light Precision, Recall, and Accuracy Analysis

We have collected similar hand-held footage as in the previous section at a slightly slower walking speed during the twilight and nighttime hours of the day, with artificial lights turned on in both cases. The human re-labeling, as well as the assessment, is conducted identically to the daylight evaluation. The vehicle detection into registration plate detection into registration plate recognition pipeline was used in the test at the twilight period of the day, while the pipeline that we have iterated to as the best for low-light conditions in Section 5.2.3, i.e., the vehicle detection is turned off, was used for the tests at night. The results in numbers and in metrics are presented in Tables 6.3 and 6.4, respectively. There was a mild fog causing more reflection on the 2025-03-26 test day, hence the higher median illuminance value. Moreover, we used just one of the two available lights during the test on 2025-03-27, while both lights were used on 2025-03-29, which explains the difference in the median illuminance for these tests.

Expectedly, the results in unfavorable lighting conditions are slightly worse than those in the daytime. Crucially, the performance difference is only marginal. The worse recognition accuracy is caused by the prolonged shutter times, which make the recorded video at times blurry. The very slight improvement in recall can be explained by the slower walking speed, i.e., there are more frames available for each vehicle, but mainly by removing one

[1]String edit distance specifies the minimum number of character edits, deletions, or additions such that two character strings become the same.

Date and Time	Median Illuminance [lx]	Day Period	Correctly Detected Vehicles	Incorrectly Detected Vehicles	Undetected Vehicles	Correct Readings
2025-03-25 18:30	24.17	twilight	283	9	3	266
2025-03-26 20:50	6.22	night	300	50	1	285
2025-03-27 21:15	2.31	night	267	13	0	254
2025-03-29 21:45	3.33	night	275	7	8	265

Table 6.3: Low-light vehicle detection and registration plate recognition evaluation in numbers

Date and Time	Precision	Recall	Accuracy
2025-03-25 18:30	0.969	0.990	0.940
2025-03-26 20:50	0.857	0.997	0.950
2025-03-27 21:15	0.954	1.000	0.951
2025-03-29 21:45	0.975	0.972	0.963

Table 6.4: Low-light vehicle detection and registration plate recognition evaluation in metrics

detection step that can fail. The noticeable drop in precision during the test on 2025-03-26 was caused by a very small confidence threshold and no size constraints for a correct registration plate detection. In reaction to these results, we have increased the threshold and, more importantly, the minimum registration plate size relative to the frame width in pixels for the following tests. As can be seen, these changes lead to a large improvement in precision while almost not affecting recall.

6.3 High-Speed Precision, Recall, and Accuracy Analysis

This section tests and evaluates two use cases that our system was not primarily designed for: the detection and recognition of registration plates of moving vehicles from a stationary position and the detection and recognition of registration plates of stationary vehicles from a moving car. We have collected footage with the camera being stationary in the first case, and hand-held in a moving car in the latter case. The evaluation of the collected footage was conducted in the same way as described in Section 6.1.

First, we present the results of the tests where moving vehicles were filmed from a stationary position in Table 6.5 using numbers and in Table 6.6 using metrics. The footage was collected during rush hours, but without the vehicle speeds affected by traffic congestion. The system was configured only to detect vehicles in the near road lane, i.e., each of the tests was performed on the same road but from a different location, capturing the more frequent traffic direction. The camera was positioned approximately for half of the test duration to capture approaching vehicles and for the other half to capture departing vehicles on both test days.

The larger number of undetected vehicles during the test on 2025-03-28 was caused by a too-shallow angle of the camera relative to the approaching/departing vehicles, i.e., the camera was capturing close vehicles at a too-sharp angle, which prevented their correct detection. We have decreased the rotation range of the camera, i.e., the camera was more perpendicular to the road, in the test on 2025-04-02, which eliminated most undetected vehicles.

Date and Time	Median Illuminance [lx]	Speed Limit [km/h]	Correctly Detected Vehicles	Incorrectly Detected Vehicles	Undetected Vehicles	Correct Readings
2025-03-28 16:20	11987.60	50	180	0	6	178
2025-04-02 07:45	7405.01	50	199	0	2	199

Table 6.5: Moving vehicle detection and registration plate recognition evaluation in numbers

Date and Time	Precision	Recall	Accuracy
2025-03-28 16:20	1.000	0.968	0.989
2025-04-02 07:45	1.000	0.990	1.000

Table 6.6: Moving vehicle detection and registration plate recognition from stationary position evaluation in metrics

Second, we show the results of the test where stationary vehicles were filmed from a moving car. Table 6.7 contains the results in numbers while Table 6.8 in metrics. We used the car's cruise control to keep the specified speeds in Table 6.7 when possible. All the tests were performed on the same day, with the weather conditions changing between overcast to partially cloudy. Note that at speeds exceeding 30 km/h, it becomes nearly impossible for humans to recognize all registration plates, not to mention record the plate numbers.

Date and Time	Median Illuminance [lx]	Driving Speed [km/h]	Correctly Detected Vehicles	Incorrectly Detected Vehicles	Undetected Vehicles	Correct Readings
2025-03-30 10:15	23091.75	25	267	1	4	264
2025-03-30 12:35	4900.56	20	323	0	4	323
2025-03-30 15:25	24122.94	35	305	3	5	303

Table 6.7: Stationary vehicle detection and registration plate recognition from moving car evaluation in numbers

Date and Time	Precision	Recall	Accuracy
2025-03-30 10:15	0.996	0.985	0.989
2025-03-30 12:35	1.000	0.988	1.000
2025-03-30 15:25	0.990	0.984	0.993

Table 6.8: Stationary vehicle detection and registration plate recognition from a moving car evaluation in metrics

The results are again near perfect in all aspects, not dropping below 0.984 in any of the used metrics. Some of the undetected vehicles and misread registration plates are caused by overexposure when the lighting conditions suddenly change, e.g., by leaving a shaded area. We have also encountered one issue that was not immediately apparent in any of the previous tests. We are collecting gigabytes of data during each test, which is stored on an SD card, causing constant stress on the file system. Sometimes, this leads to longer storage times, which, in turn, cause dropped frames and, consequently, undetected vehicles. We do not count such undetected vehicles in the evaluation because the intended final system will use a different method of results reporting, see Section 6.5.

Overall, we are confident that the system could be used without any major modifications in both situations. That is, installed in selected locations besides frequently used roads or mounted to a vehicle designed for data collection.

6.4 Ablation Study

The purpose of an ablation study in the context of deep learning is to determine how much each component contributes to the overall results. We have analyzed two components: the vehicle detection prior to the registration plate detection discussed in Section 5.1.1 and the confidence-based post-processing algorithm presented in Section 5.1.2 that combines multiple registration plate readings into a final prediction.

The first experiment was already conducted during the tests that are described in Section 6.2. Vehicle detection was not used during the tests at night. Initially, this led to a drop in precision due to more false positive registration plate detections, but we have countered that by setting constraints on what a valid detection is. We have also, yet unmentioned above, slightly modified the tracking algorithm. Instead of tracking just the detected registration plates, we track an area around the plates, effectively approximating vehicles. This small modification considerably improves the intersection-over-union-based tracking because registration plates can move significantly between two frames when the camera moves rapidly, which is the case when collecting hand-held footage while walking. In conclusion, the results point to that when the primary goal of the system is registration plate recognition without the necessity for precise vehicle detection, removing the vehicle detection stage is possible. In fact, it might even be the preferred variant if the goal is to maximize recall.

For the second experiment, we have conveniently extended labels in some of the tests performed in the above sections to include the confidence values of each registration plate reading. We now reuse the collected data to compare the performance of the confidence-based algorithm against a greedy algorithm that takes the registration plate reading with the highest confidence value per detection track. Table 6.9 shows the comparison of the two approaches when the minimum number of symbols per registration plate is not constrained for the greedy algorithm. When we consider only the registration plate readings that have at minimum seven symbols, which is the most common number of symbols in the Czech Republic, the performance of the greedy algorithm improves, mainly in the test on 2025-03-30, as can be seen in Table 6.10. Overall, we conclude that the developed confidence-based algorithm provides a significant benefit, especially in more difficult conditions such as low light at night.

Date and Time	Detected Registration Plates	Correct Readings Before	Correct Readings After	Correct Recognition Accuracy Before	Correct Recognition Accuracy After
2025-03-29 21:45	273	263	222	0.963	0.813
2025-03-30 15:25	305	303	262	0.993	0.859

Table 6.9: Comparison of registration plate recognition before and after the replacement of our confidence-based algorithm with a greedy algorithm

Date and Time	Detected Registration Plates	Correct Readings Before	Correct Readings After	Correct Recognition Accuracy Before	Correct Recognition Accuracy After
2025-03-29 21:45	273	263	227	0.963	0.832
2025-03-30 15:25	305	303	300	0.993	0.984

Table 6.10: Comparison of registration plate recognition before and after the replacement of our confidence-based algorithm with a greedy algorithm constrained to a minimum of seven symbols

6.5 Human Machine Interface

Most software, in order to be functional and helpful, requires a well-designed interface for human interactions. Based on the results of the testing performed in previous sections, we think that there are three user interface directions that can turn the system into a final product.

First, an autonomous solution that is basically without any user interface that facilitates communication with the system directly. Such a system can be used for traffic monitoring, e.g., counting how many distinct vehicles travel on a specific road per week or how many do a daily two-way trip. All of this could be done on the device, matching the vehicles by registration plate numbers and only sending the aggregated data to some collection point, thus eliminating potential issues with surveillance legal restrictions. Such applications might grow in popularity with the development of smart cities or for better traffic understanding, enabling more efficient road designs specific to a given area that minimize traffic congestion.

Second, a human-aware solution, which gives the operator feedback but does not expect any input. The use case might be the following. The operator is on patrol in a designated area and points the camera of the system at parked vehicles. The system then gives a haptic or sound notification that a vehicle was detected, its registration plate successfully recognized, and verified against a database. For the occasions when the recognized registration plate cannot be found in the database, the system will give different feedback, e.g., instructing the operator to slightly change the position of the camera.

Third, a human-in-the-loop solution that gives the operator more control. The idea here is to use Bluetooth communication with a mobile phone running a custom application. The application will store images of detected vehicles into three categories of registration plates: successfully recognized (found in a database), misread (not found in a database), and suspicious (found in a database, but, e.g., marked as expired parking). The user will be able to use the application to correct the misread plate numbers and issue parking tickets after verifying that a suspicious registration plate number is correctly recognized and belongs to the expected vehicle. We have designed and implemented a prototype of such an application for the Android smartphone operating system, whose user interface is shown in Figure 6.1.

6.6 Chapter Summary

We have thoroughly tested our vehicle detection and registration plate recognition system in various lighting conditions throughout the waking hours of the day. Overall, we have detected 3,207 vehicles, of which the system correctly recognized 3,138 registration plates,

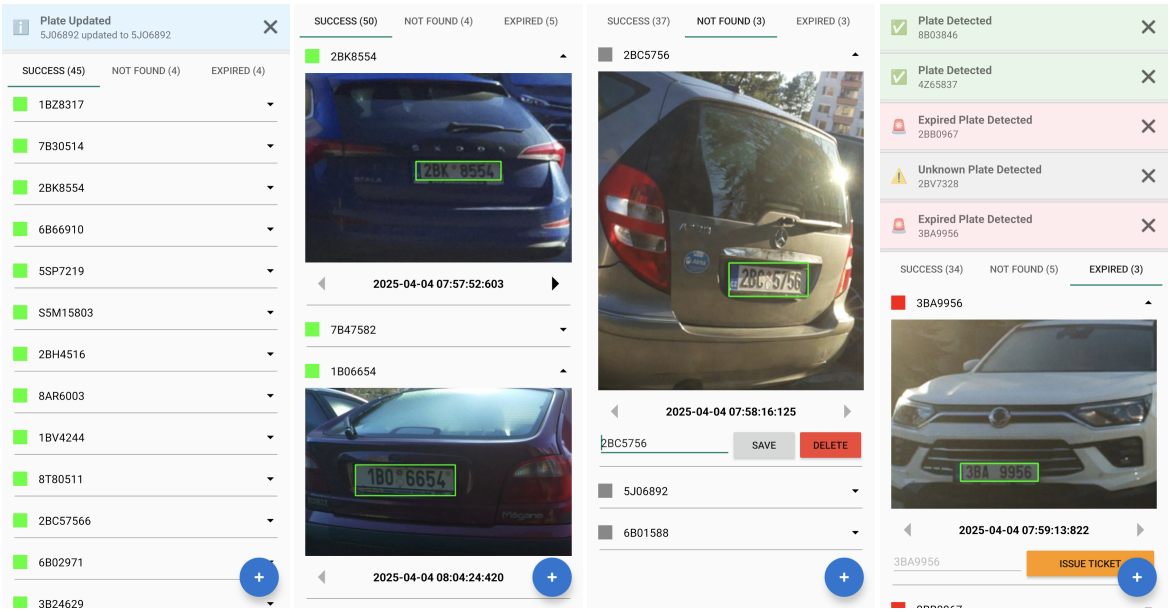


Figure 6.1: Screenshots of Android smartphone application facilitating user interface to our vehicle detection and registration plate recognition system
(vehicles are randomly assigned to the Success, Not Found, and Expired categories in a ratio of 8:1:1, respectively)

which is 0.978 recognition accuracy. The total numbers of incorrectly detected and undetected vehicles are 115 and 54, respectively, which results in 0.965 precision and 0.983 recall. If we remove the 2025-03-26 20:50 low-light test, which had an identified and later fixed issue, the overall precision, recall, and accuracy would be 0.978, 0.982, and 0.976, respectively. This answers our first question from the beginning of this chapter very positively. Moreover, the results clearly show, answering our second question, that the developed system is also very capable outside its primary use case, which is a hand-held operation, by successfully detecting moving vehicles and recognizing their registration plates at 50 km/h speeds as well as detecting and recognizing registration plates of stationary vehicles while moving at speeds exceeding 30 km/h. Next, we perform an ablation study in which we find that the confidence-based algorithm for post-processing of the registration plate readings brings a significant benefit, while we conclude that removing the vehicle detection stage might slightly improve the registration plate detection recall. Last, we suggest three possible modes of operation with different levels of human involvement and implement an Android mobile application demonstrating one of them. However, we do not have a definitive answer to our last posed question. The best human interaction with the system must be determined by the end goal, i.e., when the goal is to detect all vehicles and recognize their registration plates perfectly, humans will have to be still involved, while when occasional errors are acceptable, the system can operate autonomously.

Chapter 7

Conclusion

Our thesis focuses on low-powered deep learning inference systems utilizing quantized neural networks. Specifically, we implement a multi-stage computer vision application that performs vehicle detection, registration plate detection, and registration plate recognition in real time, extending our thesis's initial goal only to detect and track objects. The whole application is decomposed between the Raspberry Pi 5 board, combined with the Hailo-8 AI Accelerator, and the Raspberry Pi AI Camera deep learning accelerators. We enclose the named hardware in a case equipped with lithium-ion rechargeable batteries, enabling hand-held operation of the system. Furthermore, we focus on the correct image signal processing configuration combined with low-light and super-resolution deep learning image enhancements for improving the detection and recognition results.

We start the thesis with a chapter, where we present the main camera controls and a multi-stage traditional image signal processing pipeline. Next, we describe deep learning ISP approaches incorporating super-resolution, low-light enhancements, joint demosaicking and denoising, and end-to-end sensor-to-image deep learning ISP, from which we later use the ESPCN [90] and DCE-Net [33] CNNs for super-resolution and low-light enhancement, respectively. Then, we focus on the inner workings of neural networks, including an intuitive presentation of the back-propagation algorithm for their training. Last, we research common computer vision tasks in the style of a review paper. We describe and review past and current approaches to image classification, object detection, optical character recognition, semantic segmentation, and image generation, where we present the YOLO [81] CNN for object detection and pre-trained OCR models from the fast-plate-ocr[1] GitHub repository that we base our application on.

The second chapter focuses on neural network quantization, a process where learned floating-point weights of neural networks are converted to integers in order to save memory as well as improve performance in terms of speed and power efficiency with a minimal impact on the deep learning inference results. We present the mathematical framework for Uniform Affine Quantization [68] and the three main quantization approaches: post-training quantization, quantization-aware training, and quantization-aware fine-tuning or transfer learning. We conclude the chapter with an experiment of our own, comparing models of the YOLOv8m CNN trained conventionally and in a quantization-aware setting compiled with the Hailo Dataflow Compiler. We come to the unexpected conclusion that the PTQ matches the performance of the QAT for this model and compiler, while the literature suggests that QAT should outperform PTQ.

[1] https://github.com/ankandrew/fast-plate-ocr

The third chapter describes four platforms that we have tested during the work on this thesis: Hailo-15 AI Vision Processor, Up Squared Pro board with Hailo-8 AI Accelerator, Videology SCAiLX Development-Kit, and Raspberry Pi 5 with Hailo-8 AI Accelerator and Raspberry Pi AI Camera; and we evaluate them head-to-head in six categories: image quality, deep learning capabilities, general computing power, power consumption, implementation difficulty, and purchase price. We have concluded that the Hailo-15 AI Vision Processor and the Up Squared Pro board both have blocking issues that prevent us from using them in further development. In the first case, it was caused by the outdated supplementary software, while the latter platform did not have a suitable camera. The Videology SCAiLX Development-Kit is a well-integrated, all-in-one system, but it mainly lacks deep learning capabilities for more complex applications. Consequently, supported by its overall rating, we have chosen the Raspberry Pi 5 platform to proceed with.

The fourth chapter is the most important part of our thesis. We utilize the knowledge acquired in the previous chapters to implement a baseline, predominantly 8-bit integer-based, deep learning pipeline. The pipeline performs the following: detects vehicles on the Raspberry Pi AI Camera using a YOLOv8n model, crops the detections, passes them to the Hailo-8 AI Accelerator for registration plate detection with a YOLOv8s model, recognizes the cropped detected registration plates with a MobileViTv2 model running on the Raspberry Pi 5, post-processes the readings with a confidence based algorithm, and finally stores or presents the results. We then extensively experimented to tune and improve the baseline implementation. The first major update lies in the correct configuration of the image signal processing pipeline, i.e., by compensating fast shutter speeds with large analog and digital gains to provide sharp yet bright images. The second important improvement is achieved by training a vehicle's front and rear detection model on a custom dataset. Next, we examine the impact of super-resolution and low-light image enhancements. Although we can achieve better results with deep-learning-enhanced images, we come to the conclusion that the system can be more significantly improved by adding artificial lights and potentially improved by supplementing or replacing the Raspberry Pi AI Camera with a camera that can be equipped with a zoom lens and has better low-light characteristics.

The last chapter describes the testing of the finalized system in various lighting conditions throughout the waking hours of the day in three use cases. The primary use case is a handheld operation of the camera. The other two use cases include the detection and recognition of registration plates on moving vehicles at 50 km/h and vice versa, the detection and recognition of registration plates on stationary vehicles from a moving car at speeds exceeding 30 km/h. Overall, we have detected 3,207 vehicles, of which the system correctly recognized 3,138 registration plates, which is 0.978 recognition accuracy. The total numbers of incorrectly detected and undetected vehicles are 115 and 54, respectively, which results in 0.965 precision and 0.983 recall. Then, we perform an ablation study in which we find that the confidence-based algorithm for post-processing of the registration plate readings brings a significant benefit, while we conclude that removing the vehicle detection stage might slightly improve the registration plate detection recall. Last, we suggest three possible modes of operation of the system with different levels of human involvement and implement an Android mobile application demonstrating one of them.

In conclusion, we have demonstrated that it is possible to implement complex deep learning applications running in real time on small battery-powered devices and achieve results of high quality. Furthermore, beyond the scope of this thesis, we have performed an experiment comparing quantization approaches, collected and annotated a custom dataset for vehicle detection, and suggested a custom architecture and training process of a neural

network for registration plate localization and vehicle brand, model, orientation, and color classification. Moreover, we identify several areas for future research, including testing other available Raspberry Pi cameras that can be equipped with zoom lenses for registration plate recognition of distant vehicles and in automatic alignment of the new camera with the Raspberry Pi AI Camera using image features.

Bibliography

[1] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, 2019, p. 1–84. Available at: https://ieeexplore.ieee.org/document/8766229.

[2] Afifi, M. and Brown, M. S. *Interactive White Balancing for Camera-Rendered Images*. 2020. Available at: https://arxiv.org/abs/2009.12632.

[3] Afifi, M.; Brubaker, M. A. and Brown, M. S. *Auto White-Balance Correction for Mixed-Illuminant Scenes*. 2021. Available at: https://arxiv.org/abs/2109.08750.

[4] Ahmadsabry. *A Perfect guide to Understand Encoder Decoders in Depth with Visuals*. 2023. Available at: https://medium.com/@ahmadsabry678/a-perfect-guide-to-understand-encoder-decoders-in-depth-with-visuals-30805c23659b.

[5] Ahmed, I.; Ahmad, M. and Jeon, G. A real-time efficient object segmentation system based on U-Net using aerial drone images. *Journal of Real-Time Image Processing*, october 2021, vol. 18. Available at: https://link.springer.com/article/10.1007/s11554-021-01166-z.

[6] Baalen, M. van; Kuzmin, A.; Nair, S. S.; Ren, Y.; Mahurin, E. et al. *FP8 versus INT8 for efficient deep learning inference*. 2023. Available at: https://arxiv.org/abs/2303.17951.

[7] Baqai, F. A.; Riccardi, F.; Pflughaupt, R. A.; Molgaard, C. and Varghese, G. *Advanced multi-band noise reduction*. September 2015. Available at: https://patents.google.com/patent/US9641820B2/en.

[8] Barnard, K. *Practical Colour Constancy*. Burnabay, CA, 1999. PhD thesis. Simon Fraser University. Available at: http://kobus.ca/research/publications/99/PHD-99/index.html.

[9] Battiato, S.; Castorina, A. and Mancuso, M. High dynamic range imaging for digital still camera: an overview. *Journal of Electronic Imaging*. SPIE, 2003, vol. 12, no. 3, p. 459 – 469. Available at: https://doi.org/10.1117/1.1580829.

[10] Bay, H.; Ess, A.; Tuytelaars, T. and Van Gool, L. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 2008, vol. 110, no. 3, p. 346–359. ISSN 1077-3142. Available at: https://www.sciencedirect.com/science/article/pii/S1077314207001555. Similarity Matching in Computer Vision and Multimedia.

[11] Becton, H. *The crucial difference between vibrance and saturation.* February 2017. Available at: https://medium.com/@HunterBecton/the-crucial-difference-between-vibrance-and-saturation-c5bd4b8e5bf6.

[12] Bengio, Y. *Estimating or Propagating Gradients Through Stochastic Neurons.* 2013. Available at: https://arxiv.org/abs/1305.2982.

[13] Buades, A.; Coll, B. and Morel, J.-M. Non-Local Means Denoising. *Image Process. Line*, 2011, vol. 1. Available at: https://api.semanticscholar.org/CorpusID:34599104.

[14] Bull, D. R. Chapter 4 - Digital Picture Formats and Representations. In: Bull, D. R., ed. *Communicating Pictures.* Oxford: Academic Press, 2014, p. 99–132. ISBN 978-0-12-405906-1. Available at: https://www.sciencedirect.com/science/article/pii/B9780124059061000040.

[15] Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A. et al. End-to-End Object Detection with Transformers. *CoRR*, 2020, abs/2005.12872. Available at: https://arxiv.org/abs/2005.12872.

[16] Chari, V. and Veeraraghavan, A. Lens distortion, radial distortion. In: *Computer Vision: A Reference Guide.* Springer, 2021, p. 739–741. Available at: https://link.springer.com/referenceworkentry/10.1007/978-0-387-31439-6_479.

[17] Cox, S. *Introduction to Shutter Speed in Photography.* 2017. Available at: https://photographylife.com/what-is-shutter-speed-in-photography.

[18] Cox, S. *Understanding Aperture in Photography.* 2017. Available at: https://photographylife.com/what-is-aperture-in-photography.

[19] Crecraft, D. and Gergely, S. 7 - Analog-to-digital and digital-to-analog conversion. In: Crecraft, D. and Gergely, S., ed. *Analog Electronics.* Oxford: Butterworth-Heinemann, 2002, p. 156–179. ISBN 978-0-7506-5095-3. Available at: https://www.sciencedirect.com/science/article/pii/B9780750650953500076.

[20] Dalal, N. and Triggs, B. Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05).* 2005, vol. 1, p. 886–893 vol. 1. Available at: https://ieeexplore.ieee.org/document/1467360.

[21] Das, A. and Rangayyan, R. M. Enhancement of image edge sharpness and acutance. In: Dougherty, E. R. and Astola, J. T., ed. *Nonlinear Image Processing VIII.* SPIE, 1997, vol. 3026, p. 133 – 142. Available at: https://doi.org/10.1117/12.271116.

[22] Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K. et al. ImageNet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition.* 2009, p. 248–255. Available at: https://ieeexplore.ieee.org/document/5206848.

[23] Dong, C.; Loy, C. C.; He, K. and Tang, X. Image Super-Resolution Using Deep Convolutional Networks. *CoRR*, 2015, abs/1501.00092. Available at: http://arxiv.org/abs/1501.00092.

[24] Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X. et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *CoRR*, 2020, abs/2010.11929. Available at: https://arxiv.org/abs/2010.11929.

[25] El Yamany, N. A. Robust Defect Pixel Detection and Correction for Bayer Imaging Systems. In: *Digital Photography and Mobile Imaging*. 2017. Available at: https://api.semanticscholar.org/CorpusID:63047311.

[26] Elmenshawii, F. *Large-License-Plate-Detection-Dataset*. 2024. Available at: https://www.kaggle.com/datasets/fareselmenshawii/large-license-plate-dataset.

[27] Finkelstein, A.; Fuchs, E.; Tal, I.; Grobman, M.; Vosco, N. et al. QFT: Post-training quantization via fast joint finetuning of all degrees of freedom. In: *ECCV*. 2022. Available at: https://arxiv.org/abs/2212.02634.

[28] Freund, Y. and Schapire, R. E. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 1997, vol. 55, no. 1, p. 119–139. ISSN 0022-0000. Available at: https://www.sciencedirect.com/science/article/pii/S002200009791504X.

[29] Gharbi, M.; Chaurasia, G.; Paris, S. and Durand, F. Deep joint demosaicking and denoising. *ACM Trans. Graph.* New York, NY, USA: Association for Computing Machinery, december 2016, vol. 35, no. 6. ISSN 0730-0301. Available at: https://doi.org/10.1145/2980179.2982399.

[30] Gheorghe, R. V.; Goma, S. R. and Aleksic, M. An image-noise filter with emphasis on low-frequency chrominance noise. In: Rodricks, B. G. and Süsstrunk, S. E., ed. *Digital Photography V*. SPIE, 2009, vol. 7250, p. 72500B. Available at: https://doi.org/10.1117/12.805751.

[31] Girshick, R. B.; Donahue, J.; Darrell, T. and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, 2013, abs/1311.2524. Available at: http://arxiv.org/abs/1311.2524.

[32] Goyal, B.; Dogra, A.; Agrawal, S.; Sohi, B. and Sharma, A. Image denoising review: From classical to state-of-the-art approaches. *Information Fusion*, 2020, vol. 55, p. 220–244. ISSN 1566-2535. Available at: https://www.sciencedirect.com/science/article/pii/S1566253519301861.

[33] Guo, C.; Li, C.; Guo, J.; Loy, C. C.; Hou, J. et al. Zero-Reference Deep Curve Estimation for Low-Light Image Enhancement. *CoRR*, 2020, abs/2001.06826. Available at: https://arxiv.org/abs/2001.06826.

[34] He, K.; Zhang, X.; Ren, S. and Sun, J. Deep Residual Learning for Image Recognition. *CoRR*, 2015, abs/1512.03385. Available at: http://arxiv.org/abs/1512.03385.

[35] He, Y.; Liu, J.; Wu, W.; Zhou, H. and Zhuang, B. *EfficientDM: Efficient Quantization-Aware Fine-Tuning of Low-Bit Diffusion Models*. 2024. Available at: https://arxiv.org/abs/2310.03270.

[36] Horiuchi, T.; Watanabe, K. and Tominaga, S. Adaptive Filtering for Color Image Sharpening and Denoising. In: *14th International Conference of Image Analysis and Processing - Workshops (ICIAPW 2007)*. 2007, p. 196–201. Available at: https://ieeexplore.ieee.org/document/4427500.

[37] Hubara, I.; Nahshan, Y.; Hanani, Y.; Banner, R. and Soudry, D. Accurate Post Training Quantization With Small Calibration Sets. In: Meila, M. and Zhang, T., ed. *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 18–24 Jul 2021, vol. 139, p. 4466–4475. Proceedings of Machine Learning Research. Available at: https://proceedings.mlr.press/v139/hubara21a.html.

[38] Ignatov, A.; Gool, L. V. and Timofte, R. Replacing Mobile Camera ISP with a Single Deep Learning Model. *CoRR*, 2020, abs/2002.05509. Available at: https://arxiv.org/abs/2002.05509.

[39] Ignatov, A.; Malivenko, G.; Timofte, R.; Tseng, Y.; Xu, Y.-S. et al. *PyNet-V2 Mobile: Efficient On-Device Photo Processing With Neural Networks*. 2022. Available at: https://arxiv.org/abs/2211.06263.

[40] James, A. P. Towards Strong AI with Analog Neural Chips. In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2020, p. 1–5. Available at: https://ieeexplore.ieee.org/document/9180545.

[41] Jeevan, P.; Srinidhi, A.; Prathiba, P. and Sethi, A. *WaveMixSR: A Resource-efficient Neural Network for Image Super-resolution*. 2023. Available at: https://arxiv.org/abs/2307.00430.

[42] Jeon, H.; Kim, Y. and Kim, J. joon. *L4Q: Parameter Efficient Quantization-Aware Fine-Tuning on Large Language Models*. 2024. Available at: https://arxiv.org/abs/2402.04902.

[43] Jiang, Y.; Gong, X.; Liu, D.; Cheng, Y.; Fang, C. et al. EnlightenGAN: Deep Light Enhancement without Paired Supervision. *CoRR*, 2019, abs/1906.06972. Available at: http://arxiv.org/abs/1906.06972.

[44] Jirsa, P. *The Ultimate Guide to Contrast in Photography*. August 2022. Available at: https://www.adorama.com/alc/the-ultimate-guide-to-contrast-in-photography/.

[45] Kim, J.; Lee, J. K. and Lee, K. M. Accurate Image Super-Resolution Using Very Deep Convolutional Networks. *CoRR*, 2015, abs/1511.04587. Available at: http://arxiv.org/abs/1511.04587.

[46] Kokkinos, F. and Lefkimmiatis, S. Iterative Joint Image Demosaicking and Denoising Using a Residual Denoising Network. *IEEE Transactions on Image Processing*, 2019, vol. 28, no. 8, p. 4177–4188. Available at: https://ieeexplore.ieee.org/document/8668795.

[47] Kordecki, A.; Palus, H. and Bal, A. Practical vignetting correction method for digital camera with measurement of surface luminance distribution. *Signal, Image and Video Processing*. Springer, 2016, vol. 10, p. 1417–1424. Available at: https://link.springer.com/article/10.1007/s11760-016-0941-2.

[48] Koul, N. *Defective Pixel Correction: An Insight into Enhancing Image Quality.* 2023. Available at: https://www.linkedin.com/pulse/defective-pixel-correction-insight-enhancing-image-naveen/.

[49] Koul, N. *Understanding Black Level Subtraction: Its Importance, Implementation, and Impact on Image Quality.* 2023. Available at: https://www.linkedin.com/pulse/understanding-black-level-subtraction-its-importance-naveen/.

[50] Krishnamoorthi, R. Quantizing deep convolutional networks for efficient inference: A whitepaper. *CoRR*, 2018, abs/1806.08342. Available at: http://arxiv.org/abs/1806.08342.

[51] Krizhevsky, A.; Sutskever, I. and Hinton, G. E. ImageNet classification with deep convolutional neural networks. *Commun. ACM*. New York, NY, USA: Association for Computing Machinery, may 2017, vol. 60, no. 6, p. 84–90. ISSN 0001-0782. Available at: https://doi.org/10.1145/3065386.

[52] Kumar, P. *How Are Gain and Image Quality Related?* 2025. Available at: https://www.e-consystems.com/blog/camera/technology/sensor-and-isp/how-are-gain-and-image-quality-related/.

[53] Kunal, K. *Front and rear images of car.* 2021. Available at: https://www.kaggle.com/datasets/kushkunal/front-and-rear-images-of-car.

[54] LeCun, Y. A theoretical framework for Back-Propagation. In: Touretzky, D.; Hinton, G. and Sejnowski, T., ed. *Proceedings of the 1988 Connectionist Models Summer School.* CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988, p. 21–28. Available at: https://yann.lecun.com/exdb/publis/pdf/lecun-88.pdf.

[55] Lee, J.; Kim, D.; Jeon, J. and Ham, B. *Transition Rate Scheduling for Quantization-Aware Training.* 2024. Available at: https://arxiv.org/abs/2404.19248.

[56] Li, M.; Lv, T.; Cui, L.; Lu, Y.; Florêncio, D. A. F. et al. TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models. *CoRR*, 2021, abs/2109.10282. Available at: https://arxiv.org/abs/2109.10282.

[57] Li, X.; Gunturk, B. and Zhang, L. Image demosaicing: a systematic survey. In: Pearlman, W. A.; Woods, J. W. and Lu, L., ed. *Visual Communications and Image Processing 2008.* SPIE, 2008, vol. 6822, p. 68221J. Available at: https://doi.org/10.1117/12.766768.

[58] Lin, T.; Maire, M.; Belongie, S. J.; Bourdev, L. D.; Girshick, R. B. et al. Microsoft COCO: Common Objects in Context. *CoRR*, 2014, abs/1405.0312. Available at: http://arxiv.org/abs/1405.0312.

[59] Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y. et al. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. *CoRR*, 2021, abs/2103.14030. Available at: https://arxiv.org/abs/2103.14030.

[60] Lluis Gomez, A. and Edirisinghe, E. A. Chromatic aberration correction in RAW domain for image quality enhancement in image sensor processors. In: *2012*

IEEE 8th International Conference on Intelligent Computer Communication and Processing. 2012, p. 241–244. Available at: https://ieeexplore.ieee.org/document/6356192.

[61] Long, J.; Shelhamer, E. and Darrell, T. Fully Convolutional Networks for Semantic Segmentation. *CoRR*, 2014, abs/1411.4038. Available at: http://arxiv.org/abs/1411.4038.

[62] Lore, K. G.; Akintayo, A. and Sarkar, S. LLNet: A Deep Autoencoder Approach to Natural Low-light Image Enhancement. *CoRR*, 2015, abs/1511.03995. Available at: http://arxiv.org/abs/1511.03995.

[63] Lowe, D. G. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, Nov 2004, vol. 60, no. 2, p. 91–110. ISSN 1573-1405. Available at: https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[64] Lv, S.; Ding, Y. and Wei, H. An improved image distortion correction algorithm. In: CRC Press. *Multimedia, Communication and Computing Application: Proceedings of the 2014 International Conference on Multimedia, Communication and Computing Application (MCCA 2014), Xiamen, China, October 16-17, 2014*. 2015, p. 319. Available at: https://books.google.cz/books?hl=cs&lr=&id=TMfECQAAQBAJ&oi=fnd&pg=PA319&dq=Lv,+S.%3B+Ding,+Y.+and+Wei,+H.+An+improved+image+distortion+correction+algorithm.&ots=8SBB6kz_Kx&sig=gYBTD3bcFCF-6RZXXGfDKaatASY&redir_esc=y#v=onepage&q&f=false.

[65] Matsumura, M.; Takamura, S. and Shimizu, A. Largest coding unit based framework for non-local means filter. In: *Proceedings of The 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference*. 2012, p. 1–4. Available at: https://ieeexplore.ieee.org/abstract/document/6411796.

[66] Mazzoli, F. *Lánczos interpolation explained*. October 2010. Available at: https://mazzo.li/posts/lanczos.html.

[67] Nagel, M.; Baalen, M. van; Blankevoort, T. and Welling, M. Data-Free Quantization through Weight Equalization and Bias Correction. *CoRR*, 2019, abs/1906.04721. Available at: http://arxiv.org/abs/1906.04721.

[68] Nagel, M.; Fournarakis, M.; Amjad, R. A.; Bondarenko, Y.; Baalen, M. van et al. A White Paper on Neural Network Quantization. *CoRR*, 2021, abs/2106.08295. Available at: https://arxiv.org/abs/2106.08295.

[69] Nagel, M.; Fournarakis, M.; Bondarenko, Y. and Blankevoort, T. *Overcoming Oscillations in Quantization-Aware Training*. 2022. Available at: https://arxiv.org/abs/2203.11086.

[70] Norouzzadeh, Y. and Rashidi, M. Image denoising in wavelet domain using a new thresholding function. In: *International Conference on Information Science and Technology*. 2011, p. 721–724. Available at: https://ieeexplore.ieee.org/document/5765347.

[71] Panagiotou, S. and Bosman, A. S. Denoising diffusion post-processing for low-light image enhancement. *Pattern Recognition*. Elsevier BV, december 2024,

vol. 156, p. 110799. ISSN 0031-3203. Available at: http://dx.doi.org/10.1016/j.patcog.2024.110799.

[72] Park, J.; Lee, S. and Song, B.-C. Stable Quantization-Aware Training with Adaptive Gradient Clipping. In: *2023 International Conference on Electronics, Information, and Communication (ICEIC)*. 2023, p. 1–3. Available at: https://ieeexplore.ieee.org/document/10049939.

[73] Prangnell, L. *Visible Light-Based Human Visual System Conceptual Model*. 2019. Available at: https://arxiv.org/abs/1609.04830.

[74] Rafael C. Gonzalez, R. E. W. Digital Image Processing Global Edition. In:. PEARSON Education Limited, 2017, p. 236. Available at: https://dl.icdst.org/pdfs/files4/01c56e081202b62bd7d3b4f8545775fb.pdf.

[75] Rafael C. Gonzalez, R. E. W. Digital Image Processing Global Edition. In:. PEARSON Education Limited, 2017, p. 165. Available at: https://dl.icdst.org/pdfs/files4/01c56e081202b62bd7d3b4f8545775fb.pdf.

[76] Rafael C. Gonzalez, R. E. W. Digital Image Processing Global Edition. In:. PEARSON Education Limited, 2017, p. 437–443. Available at: https://dl.icdst.org/pdfs/files4/01c56e081202b62bd7d3b4f8545775fb.pdf.

[77] Ramajo Ballester Álvaro; Armingol Moreno, J. M. and de la Escalera Hueso, A. Dual license plate recognition and visual features encoding for vehicle identification. *Robotics and Autonomous Systems*, 2024, vol. 172, p. 104608. ISSN 0921-8890. Available at: https://www.sciencedirect.com/science/article/pii/S0921889023002476.

[78] Ramakrishnan, R. K.; Jui, S. and Nia, V. P. Deep Demosaicing for Edge Implementation. *CoRR*, 2019, abs/1904.00775. Available at: http://arxiv.org/abs/1904.00775.

[79] Ramesh, A.; Pavlov, M.; Goh, G.; Gray, S.; Voss, C. et al. Zero-Shot Text-to-Image Generation. *CoRR*, 2021, abs/2102.12092. Available at: https://arxiv.org/abs/2102.12092.

[80] Rasch, M. J.; Carta, F.; Fagbohungbe, O. and Gokmen, T. Fast and robust analog in-memory deep neural network training. *Nature Communications*, Aug 2024, vol. 15, no. 1, p. 7133. ISSN 2041-1723. Available at: https://doi.org/10.1038/s41467-024-51221-z.

[81] Redmon, J.; Divvala, S. K.; Girshick, R. B. and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*, 2015, abs/1506.02640. Available at: http://arxiv.org/abs/1506.02640.

[82] Ren, S.; He, K.; Girshick, R. B. and Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*, 2015, abs/1506.01497. Available at: http://arxiv.org/abs/1506.01497.

[83] Rombach, R.; Blattmann, A.; Lorenz, D.; Esser, P. and Ommer, B. High-Resolution Image Synthesis with Latent Diffusion Models. *CoRR*, 2021, abs/2112.10752. Available at: https://arxiv.org/abs/2112.10752.

[84] Rosebrock, A. *OpenCV Gamma Correction*. 2015. Available at: https://pyimagesearch.com/2015/10/05/opencv-gamma-correction/.

[85] Santana Cedrés, D.; Gómez, L.; Alemán Flores, M.; Salgado, A.; Esclarín, J. et al. An Iterative Optimization Algorithm for Lens Distortion Correction Using Two-Parameter Models. *Image Processing On Line*, 2016, vol. 6, p. 326–364. https://doi.org/10.5201/ipol.2016.130.

[86] Savvas, T. *Algebraic Modeling of Transformations from Bayer to RGB Images*. Crete, GR, 2006. Diploma thesis. Technical University of Crete. Available at: https://doi.org/10.26233/heallink.tuc.10983.

[87] Sermanet, P.; Eigen, D.; Zhang, X.; Mathieu, M.; Fergus, R. et al. *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*. 2014. Available at: https://arxiv.org/abs/1312.6229.

[88] Shao, V. *Implementing the Gamma Correction Algorithm Using the TMS320C2xx DSP*. SPRA361. Texas Instruments Taiwan Limited Technical Staff, september 1997. Available at: https://www.ti.com/lit/an/spra361/spra361.pdf.

[89] Shi, B.; Bai, X. and Yao, C. An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition. *CoRR*, 2015, abs/1507.05717. Available at: http://arxiv.org/abs/1507.05717.

[90] Shi, W.; Caballero, J.; Huszár, F.; Totz, J.; Aitken, A. P. et al. Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network. *CoRR*, 2016, abs/1609.05158. Available at: http://arxiv.org/abs/1609.05158.

[91] Simonyan, K. and Zisserman, A. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. Available at: https://arxiv.org/abs/1409.1556.

[92] Smith, R. An Overview of the Tesseract OCR Engine. In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. 2007, vol. 2, p. 629–633. Available at: https://ieeexplore.ieee.org/document/4376991.

[93] Startups, A. *Vehicle Registration Plates Computer Vision Project*. 2021. Available at: https://universe.roboflow.com/augmented-startups/vehicle-registration-plates-trudk.

[94] Syu, N.; Chen, Y. and Chuang, Y. Learning Deep Convolutional Networks for Demosaicing. *CoRR*, 2018, abs/1802.03769. Available at: http://arxiv.org/abs/1802.03769.

[95] Tian, Z.; Qu, P.; Li, J.; Sun, Y.; Li, G. et al. A Survey of Deep Learning-Based Low-Light Image Enhancement. *Sensors*, 2023, vol. 23, no. 18. ISSN 1424-8220. Available at: https://www.mdpi.com/1424-8220/23/18/7763.

[96] Tomaselli, V.; Guarnera, M. and Roffet, G. *Image chroma noise reduction*. September 2012. Available at: https://patents.google.com/patent/US9135681B2/en.

[97] Wang, J.; Sun, K.; Cheng, T.; Jiang, B.; Deng, C. et al. Deep High-Resolution Representation Learning for Visual Recognition. *CoRR*, 2019, abs/1908.07919. Available at: http://arxiv.org/abs/1908.07919.

[98] Wang, X.; Xie, L.; Dong, C. and Shan, Y. *Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data*. 2021. Available at: https://arxiv.org/abs/2107.10833.

[99] Wei, K. *Understand Transposed Convolutions*. 2020. Available at: https://towardsdatascience.com/understand-transposed-convolutions-and-build-your-own-transposed-convolution-layer-from-scratch-4f5d97b2967.

[100] Xiao, F.; Farrell, J. E.; DiCarlo, J. M. and Wandell, B. A. Preferred color spaces for white balancing. In: Sampat, N.; Motta, R. J.; Blouke, M. M.; Sampat, N. and Motta, R. J., ed. *Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications IV*. SPIE, 2003, vol. 5017, p. 342 – 350. Available at: https://doi.org/10.1117/12.478482.

[101] Xiao, K.; Fu, C.; Karatzas, D. and Wuerger, S. Visual gamma correction for LCD displays. *Displays*, 2011, vol. 32, no. 1, p. 17–23. ISSN 0141-9382. Available at: https://www.sciencedirect.com/science/article/pii/S0141938210000740.

[102] Yang, L.; Sander, P.; Lawrence, J. and Hoppe, H. Antialiasing Recovery. *ACM Trans. Graph.*, may 2011, vol. 30, p. 22. Available at: https://hhoppe.com/aarecovery.pdf.

[103] Yurlovska, M. *The meaning of Saturation in photography. A guide for beginners.* December 2022. Available at: https://skylum.com/cs/blog/saturation-in-photography-a-guide-for-beginners.

Appendix A

Extended Figures

Figures in Chapter 5 contain only a limited examples of each described phenomenon in order to show large enough images for the printed version of this thesis. We encourage readers of the printed version as well as the compressed electronic version to download the 76 MB uncompressed version with full-resolution images from:
https://nextcloud.fit.vutbr.cz/s/k7WQKXEwqZGMtcz.
Please, inspect the images in the Figures shown below by zooming the downloaded document in order to see the necessary detail.

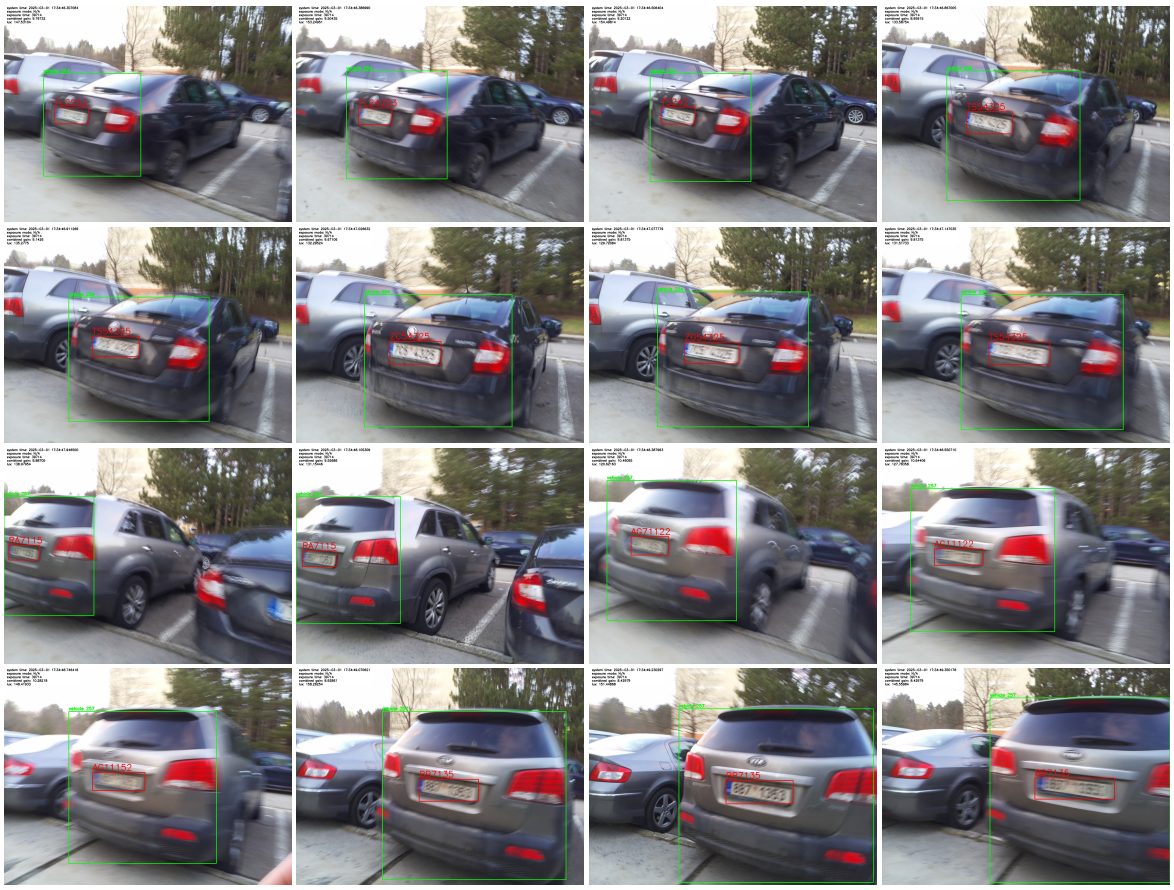


Figure A.1: Failed registration plate readings due to unnecessarily long exposure times in hand-held footage captured at walking speed

Figure A.2: Correct registration plate readings with configured image signal processing pipeline for shorter exposure times in hand-held footage captured at walking speed

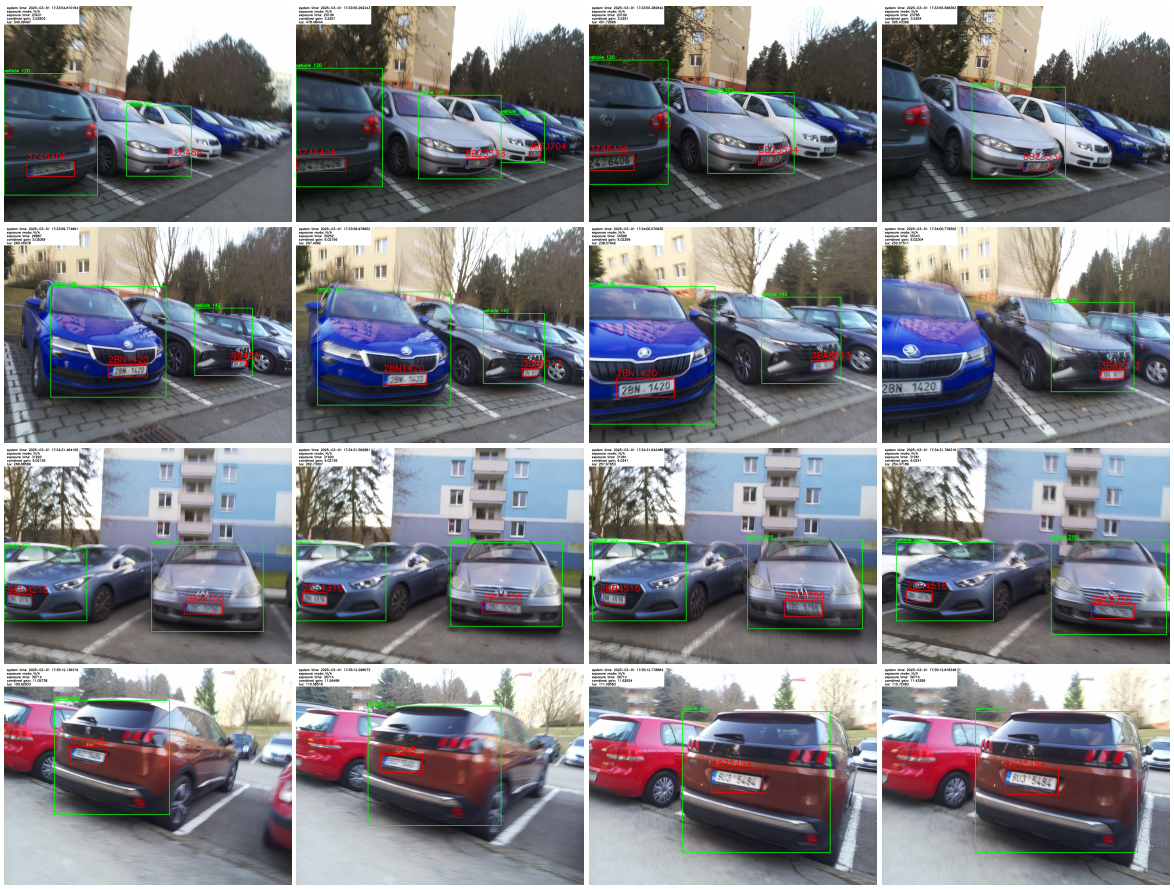


Figure A.3: Eventually correct registration plate readings with default image signal processing pipeline settings in hand-held footage captured at walking speed

Figure A.4: General vehicle detection model causing overlaps leading to incorrect registration plate assignment, thus their incorrect readings in hand-held footage captured at walking speed

Figure A.5: Fine-tuned vehicle detection model detecting only vehicle fronts, leading to correct registration plate assignment and readings in hand-held footage captured at walking speed

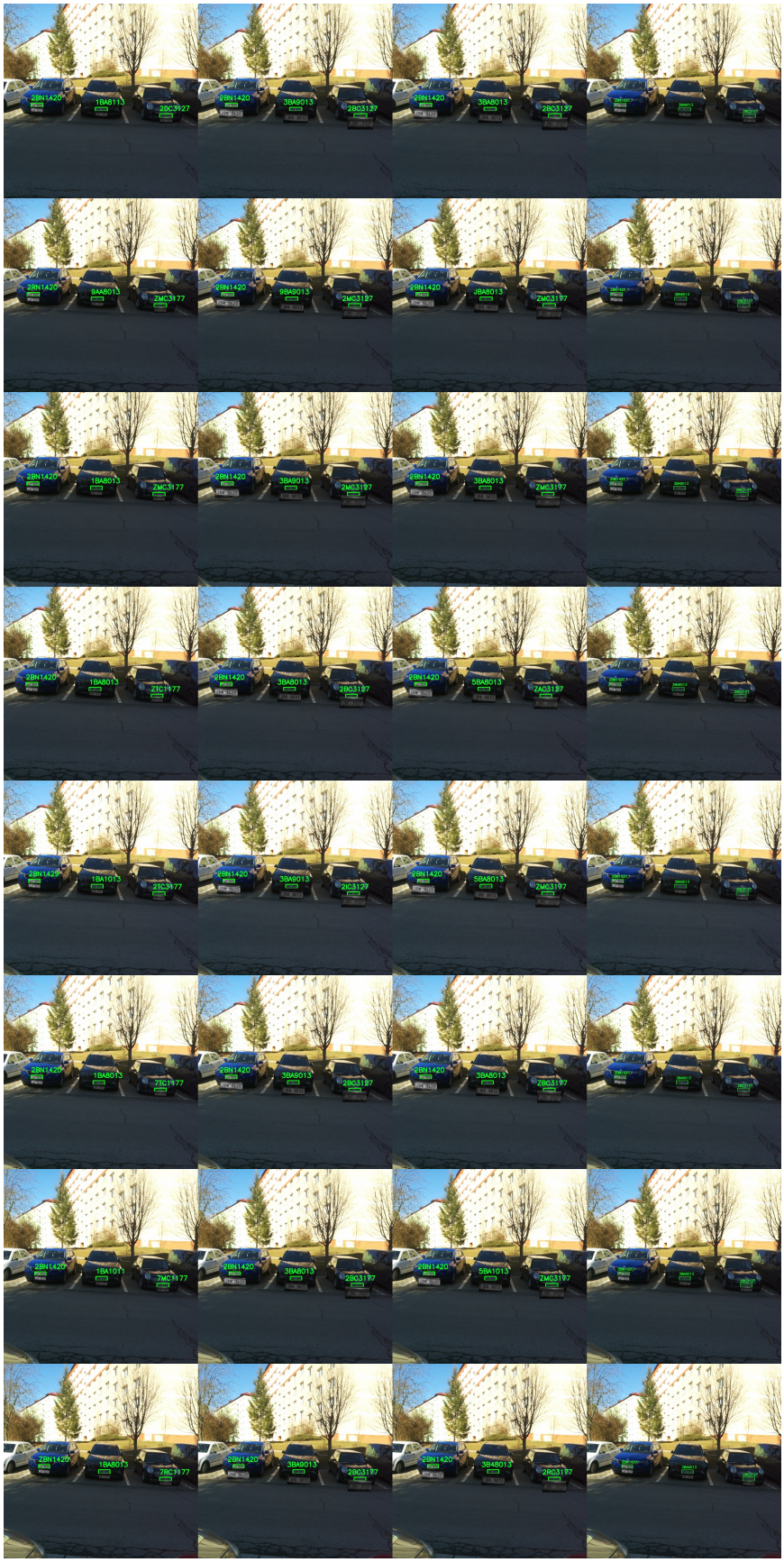


Figure A.6: Readings of low resolution (leftmost), 2x super-resolution, 2x bicubic interpolation, and 2x actual crops of registration plates across multiple frames in hand-held footage

Figure A.7: Correct registration plate readings in enhanced short exposure frames (middle) during the civil twilight period in hand-held footage captured at slow walking speed

Figure A.8: Correct registration plate readings in long exposure frames (right) during the civil twilight period in hand-held footage captured at slow walking speed

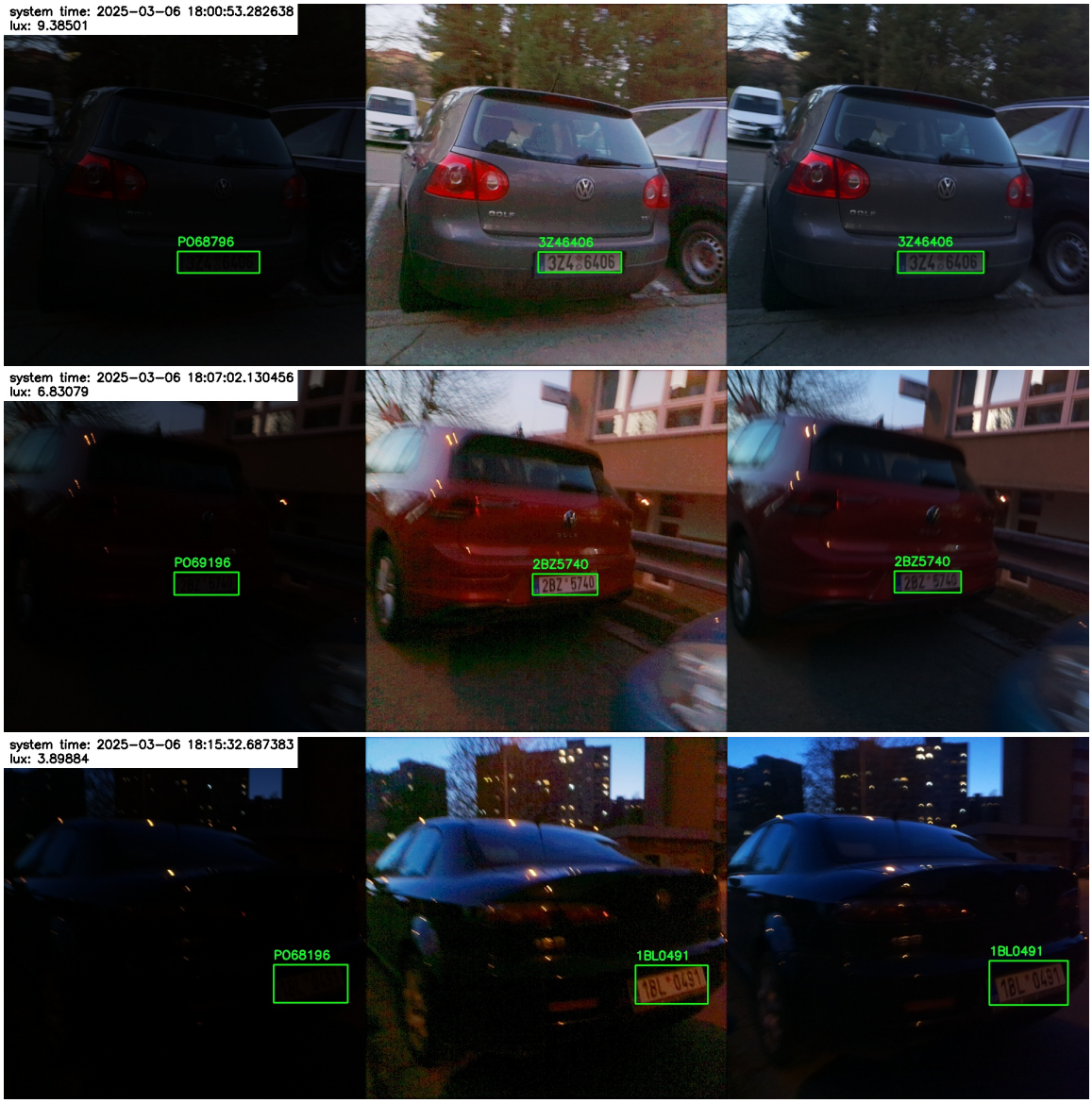


Figure A.9: Correct registration plate readings in both enhanced short exposure (middle) and long exposure frames (right) during the civil twilight period in hand-held footage captured at slow walking speed

Figure A.10: Correct night mode registration plate detections and recognition in hand-held footage captured at slow walking speed