Tampere University

Jukka Yrjänäinen

PRIVACY CONSCIOUS COMPUTER VISION

Information Technology and Communication
Master of Science Thesis
May 2020

ABSTRACT

Jukka Yrjänäinen: Privacy conscious computer vision
Master of Science Thesis
Tampere University
Electrical Engineering
May 2020

This work describes the design and the implementation of the distributed computer vision solution for person tracking and counting in a public museum. The system consists of an edge device fleet that does send data to a cloud server for further analysis. The key design goal is the protection of the privacy of the persons being monitored. This is achieved by a system that does not send actual images to the server, instead detected persons are represented with a feature vector extracted from the images. Privacy sensitive image data is not stored or transmitted anywhere in the system.

Device design consists of Raspberry Pi single-board computers equipped with a neural network acceleration hardware and a camera module. These devices are used to locate a person from the camera view with the object detection neural network. After object detection a re-identification neural network is applied to the found object to generate a feature vector representation. This vector is sent to the cloud server. Based on the feature vectors it is possible to associate detected people across multiple cameras and moments time. However, it is not possible to reconstruct the original image from the feature vector.

The experiments and performance measurements with edge devices show that the simultaneous use of deep neural networks for object detection and feature generation using relatively low-cost hardware is feasible. The design recommendation based on the experiments is that the use of a dedicated HW accelerator for running all neural networks is preferred. The analysis also show that the variation of the accuracy and computational complexity of used neural networks offers a range of feasible performance trade-offs.

The data analysis method in the server using only feature vector data for tracking and clustering is evaluated. The experiments with publicly available image dataset indicate that with the proposed approach it is possible to approximate person count with reasonable accuracy.

Keywords: computer vision, neural networks, object detection, re-identification, edge computing

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Jukka Yrjänäinen: Yksityisyyttä suojaava konenäköjärjestelmä
Diplomityö
Tampereen yliopisto
Sähkötekniikka
Toukokuu 2020

Tässä työssä on suunniteltu ja toteutettu hajautettu konenäköjärjestelmä henkilöiden seurantaan julkisessa museossa. Järjestelmä koostuu joukosta laitteita, jotka lähettävät tietoa pilvipalvelimelle jatkoanalyysiä varten. Suunnittelun keskeinen lähtökohta on seurattavien henkilöiden yksityisyydensuojan varmistaminen. Tämä on saavutettu ratkaisulla, jossa kuvainformaatiota ei lähetetä palvelimelle. Sen sijaan havaitut henkilöt esitetään erillisillä piirrevektoreilla.

Laitteet on toteutettu Raspberry Pi pienoistietokoneella, johon on liitetty kamera ja ulkoinen neuroverkkokiihdytin. Laite paikantaa henkilön kameran tuottamasta kuvasta syvää neuroverkkoa käyttäen. Tämän jälkeen toinen, *uudelleentunnistus*- (engl. re-indentification) neuroverkko laskee löydetystä kohteessa piirrevektorin, joka lähetetään pilvipalvelimelle. Näiden vektoreiden avulla eri ajankohtina ja eri kameroilla tehdyt havainnot samasta henkilöstä voidaan yhdistää. Piirrevektorista ei kuitenkaan ole mahdollista muodostaa alkuperäistä kuvaa, joten henkilön yksityisyys pysyy suojattuna.

Tehdyt kokeet ja mittaukset osoittavat, että kahden syvän neuroverkon samanaikainen suorittaminen on mahdollista käytetyllä kustannustehokkaalla laskenta-alustalla. Tehdyn analyysin pohjalta voidaan todeta, että kummakin neuroverkon laskenta kannattaa suorittaa ulkoisella kiihdyttimellä. Mittaukset osoittavat myös, että erilaisilla neuroverkkojen konfiguraatioilla saavutetaan laaja kirjo erilaisen suorituskyvyn omaavia toimivia vaihtoehtoja.

Työssä tutkittiin myös palvelimelle tallennettujen piirrevektorien soveltuvuutta henkilöiden kulkureittien muodostamiseen sekä havaittujen henkilöiden kokonaismäärän arviointiin. Kokeet julkisesti saatavalla kuvatietokannalla osoittavat, että henkilöiden määrän laskeminen riittävällä tarkkuudella on mahdollista esitetyillä menetelmillä.

Avainsanat: konenäkö, neuroverkot, uudelleentunnistus, reunalaskenta

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

PREFACE

This work is conducted at the Tampere University in the Machine Learning Group as a part of the Citytrack2 project.

I would like to thank my supervisor Associate Professor Heikki Huttunen for offering this highly interesting project and opportunity to work with talented people in the Machine Learning Group.

I also need to thank and acknowledge the highly valuable work of M.Sc. Xingyang Ni and M.Sc. Bishwo Adhikari for providing re-identification and object detection neural networks utilized in this work. I am thanking also to Mr. Wouter Legiest and Mr. Leevi Raivio who worked with me on the same project, producing valuable results and ideas that impacted also to this thesis.

The people of the Machine Learning Group and other colleagues at the Tampere University have provided a very nice working environment for which I am also grateful.

Tampere, 12th May 2020

Jukka Yrjänäinen

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

AE	Automatic Exposure
AF	Automatic Focus
AI	Artificial Intelligence
API	Application Programming Interface
AWB	Automatic White Balance
BOW	Bag of Words
CCD	Charge Coupled Device
CFAI	Color Filter Array Interpolation
CMOS	Complementary Metal Oxide Semiconductor
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSC	IT Center for Science ltd.
CSI	Camera Serial Interface
DNN	Deep Neural Network
EDPB	Europen Data Protection Board
EU	Europen Data Protection Board
FPGA	Field Programmable Gate Array
fps	frames per second
GDPR	General Data Privacy Regulation
GPU	Graphics Processing Unit
ILSVRC	Imagenet Large Scale Visual Recognition Challenge
IoT	Internet of Things
IR	Intermediate Representation
ISP	Image Signal Processor
JPEG	Joint Picture Expert Group
mAP	mean Average Precision
MIA	Model Inversion Attack
MJPEG	Motion Joint Picture Expert Group
NCS	Neural Compute stick

RCNN	Region-based Convolutional Neural Network
REST	Representational State Transfer
SGD	Stochastic Gradient Descent
SIFT	Scale Invariant Feature Transformation
SoC	System-on-Chipt
SSD	Single Shot Multibox Detector
SSH	Secure Shell
TLV	Time-Lengh-Value
TPU	Tensor Processing Unit
USB	Universal Serial Bus
VOC	Visual Object Classes
WLAN	Wireless Local Area Network
YOLO	You Only Look Once

1 PRIVACY CONSCIOUS PEOPLE MONITORING - A MYTH OR POSSIBILITY?

Detecting the presence of people in a particular location, such as room inside a building or street has multiple different usages. Different security applications do need to monitor the presence of a person in restricted areas and create an alarm if unauthorized persons are detected. People movements in a retail shop can be analyzed in order to optimized shop layout and product placement. Pedestrian traffic can be counted and monitored and movement patterns can be utilized to improve the operation of the traffic lights. While advances in technology are making such applications increasingly more popular, concerns have raised about the privacy of an individual. Is the privacy-conscious computer vision a contradiction in terms - or a possibility for new research and development? In this thesis solutions that do make tracking of a person possible without compromising privacy are investigated.

1.1 People monitoring

There are many different technological solutions for people tracking. Some of them do require active co-operation from a person, such access control implemented via electronic keys or identification cards, mobile applications that are broadcasting the location of the user, or even simply just pressing a button to indicate the presence in a particular spot. Other methods do utilize various sensors to detect the presence of a person. Examples of these include pressure sensors installed in the floor, photovoltaic cells in doorways, microphones monitoring speech or sound of the footsteps, and finally use of cameras.

The use of cameras for person detection has several benefits over many other alternatives. People co-operation is not needed. With a single camera, it is possible to cover a wide area. Data obtained from cameras is rich compared to most other sensor-based solutions allowing analysis of additional information than just the presence of a person. In many cases the images captured by a camera are also directly suitable for viewing by people without further processing, in fact many security applications do utilize human operators that do monitor the live video feed.

Computer vision is widely used for various surveillance tasks. In classical approaches, video streams are transmitted to a centralized location where security authorities monitor them, and video is stored for further reference. Proper management of video in such a way that the privacy of the people is taken into account is an important consideration. Typically only a small number of authorized people are allowed to inspect the video, and the storage duration of the video sequence is also restricted, e.g., for 24 hours. Despite the precautions done for the protect user privacy in such systems, there is always a theoretical possibility that unauthorized persons could obtain access to the video data during the transmission or storage.

Computer vision offers an advantageous and cost-efficient solution for analyzing people's movements also for other purposes than just security surveillance. Counting how many people are present in a particular space or following their movement patterns could provide valuable information for traffic planning, retail-shop layout design, and analysis of the utilization of a particular public service. In these types of scenarios, the privacy of the individuals needs to be considered. Unlike the security surveillance use case where access to the data can be typically restricted, here it would be beneficial to grant wider access to the collected data.

Although a camera-based person monitoring solution has several benefits it is not a problem-free. Issues can be divided into two categories: technical and ethical. Technical challenges relate to the implementation of the computer vision system that is able to perform well enough in a given task in a particular environment with available resources. As an example challenges may arise due to the performance of the algorithms in limited computing equipment or the ability of the camera sensors to capture good enough images in poor illumination conditions. Ethical challenges arise from the fact that camera-based monitoring can be implemented without people co-operation or even consent. The situation is similar when using other sensor-based monitoring system but cameras differ in the sense that the nature of data they do capture is more sensitive than with most other sensors. Specifically it allows the identification of the individual. While identifying a person is a desired feature in many security monitoring cases there are several use cases where this feature is not needed and risk for violation of the privacy of the individual is real. In the European Union (EU) the General Data Protection Regulation (GDPR) has become enforceable at the beginning of 25 May 2018 and it sets strict demands for collecting personal information. Video surveillance data be considered personal information in the spirit of GDPR. Europen Data Protection Board (EDPB) has also issued the first version of guidelines on processing personal data trough video devices for public consultation on 10 July 2019. So, it is seen that when using camera technology for people monitoring privacy protection is also enforced by legislation.

1.2 Protecting the privacy by design

This work studies if it is possible to create a technical solution that serves data analysis needs without compromising the privacy of the individuals. At the high level the proposed solution is based on intelligent camera devices that process the data on the edge, detecting the movement of the people with a neural network-based object detector and using another neural network for extracting abstract features of each individual. The detected objects' locations and corresponding features are sent to the server that collects the data feed from multiple cameras and stores them for further examination. The benefit of the proposed method is that features generated by the neural networks will make it possible to combine information from multiple cameras and time instances to estimate the position where a particular person has been moving. However, it is not feasible to reconstruct the original picture from the feature vector, thus identifying the person as a named individual will be inherently extremely hard if one has access only to feature vector data. Furthermore, as the size of the feature is considerably smaller than the original image, the storage and transmission bandwidth requirements are smaller than when using the video stream.

The approach investigated in this report assumes that there is enough computational power locally in the camera-equipped edge device so that we perform required processing with can an adequate frame-rate in order to provide enough data for analysis. In practice, it should be noted that the performance is not limited only by the computational capacity but also by the thermal behavior of the system. Additionally, the physical size and expense might add extra constraints for hardware that can be employed.

Low-cost edge devices with neural network accelerators have been used for traditional surveillance applications in previous work. For example, Lage *et al.* study the use of object detection neural networks with Raspberry Pi and Neural Compute Stick [1]. Similarly, Dey *et al.* experiment with the implementation of pre-trained classification networks for the application of a robot vehicle navigation, and investigate the partition approaches of the processing between an edge device and a server [2]. These types of studies do not analyze the implementation of both object detection and feature generation in the same edge device. Furthermore, they do not concern the privacy issues involved with the transmission or storage of sensitive image data. Privacy challenges of machine learning-based data analytics have been discussed in [3] by Zhao *et al.*. Authors argue in favor of the approach that the analysis would be done on the edge device, reducing the exposure of the personal data to the cloud servers. This study agrees with the statements, but adds that when analysis requires composing information from multiple edge devices together, sending data to some centralized location cannot be avoided. In order to protect privacy, approaches, as described later in this study, are required. Sophisticated Model Inversions Attack (MIA) techniques as demonstrated by Fredrikson *et al.* in [4] are still a potential threat. However, in our approach as the feature calculation is done in the completely in edge there is no need to expose actual network models to the cloud environment thus making the practical implementation of any MIA strategy very hard.

One might attempt to handle privacy protection issues other means as for example cryptography methods that do protect the data from unauthorized access. While it is possible to limit the access to the data there are still risks related to authentication, with passwords or secure key management. Malicious hacking of a system or deception by social engineering are in case a weak point of such a system where the privacy-sensitive data is stored. Also it should be noted that any system that actually stores privacy-sensitive data needs to be subject to the GDPR regulations.

1.3 Structure of the work

The primary research question in this work is how feasible it is to implement a practical distributed computer vision application for person monitoring use case without compromising the privacy aspects. Key questions that need to be answered are:

- How feasible it is to use low cost small embedded computer as a base for such system, what kind of software and hardware architectures are suitable for running required algorithms and how the edge device fleet management can be arranged?
- What is the performance range of the needed computer vision algorithms in the selected computing environment? Is the performance of the algorithms, specifically, accuracy and speed, sufficient for the use case needs?
- In the cloud server-side, is it possible to fulfill use case requirements for data analysis using only privacy-safe data generated by edge devices, without access to the actual image data.

In this thesis, research questions are approached by designing and implementing the required system and analyzing the performance for a real-life use, visitor monitoring, and counting in a public museum. The scope of the work includes the design and implementation of the edge computing device, development of methodology for data analysis, and doing performance analysis of systems and algorithms. Training of the used neural network models and implementation of the cloud server was not the scope of the work, instead of results provided by researchers of the Machine Learning Group at Tampere University were utilized. The work was part of the CityTrack2 research project.

The thesis is organized as follows: Chapters 2 and 3 are giving an overview of the technologies that are relevant to this work. Chapter 2 is an introduction to the camera and computer vision technology. Chapter 3 starts with an overview of the machine learning followed by an introduction to deep neural networks (DNN). After that the architectures of the DNNs utilized in this work are introduced. The chapter ends with a short review of common frameworks used in the practical design and deployment of the networks. Chapter 4 starts with the analysis of the target use case. HW and SW design of the edge device implemented during the work is covered. At the end neural networks used in the device are discussed. Chapter 5 focuses on edge device performance analysis. The speed of individual networks and throughput of full system is measured and compared to the requirements. At the beginning of the chapter 6 a high level overview of the cloud server architecture utilized in the work is introduced. After that the chapter focuses on analysis methodology that can be applied to the data stored in the server. The performance of the proposed algorithms are measured and analyzed. Finally, chapter 7 discusses the aspect of the privacy-aware design, summarizes the work, and proposes topics for further research.

2 COMPUTER VISION FOR EMBEDDED SYSTEMS

Solutions developed and investigated in this work are using cameras to capture image data and further process it in order to gain an understanding of the presence of people on the premises. Applications like this do belong to the realm of computer vision. It is a collection of the various technologies, algorithms, and methods to process visual information in order to extract some higher-level data from the image data [5]. Recognizing objects, such as faces, persons, or vehicles in the camera view and analyzing their motion are typical examples of computer vision applications.

A practical computer vision system consist of an imaging device, for example a digital camera, that captures the image and a processing system that executes desired algorithms. System can be real-time, where the processing is tightly coupled with the capture to operate with the same rate as images are acquired. It is also possible to completely separate the processing from image acquisition and apply computer vision algorithms for previously captured images or video streams.

Advances in technology, to a large extent driven by the mobile phone industry, have made it possible to include cameras into small embedded devices, making them also a suitable platform for computer vision applications. For example, face detection is almost a standard feature in camera applications present in mobile phones. The same hardware technology base has enabled also production of affordable single-board computing platforms, such as the Raspberry Pi [6] or Jetson Nano [7]. When these or similar platforms are equipped with cameras they are suitable for a large variety of computer vision tasks.

In the following sections an overview of the camera technology for embedded devices is presented, followed by an introduction to some of the computer vision concepts and algorithms that are relevant for this work.

2.1 Camera technology

Image acquisition is the first step in computer vision. In this work the focus is on processing images captured with a visible spectrum camera. A word camera can refer to a full device including, optics, image sensor, processing, storage memory, and display [8]. Sometimes the term is used for a subset, consisting of optics, image sensor and processing system. In the small embedded devices this subset is more often referred to as *a camera module*, a single component that contains optics and a sensor as integrated package.

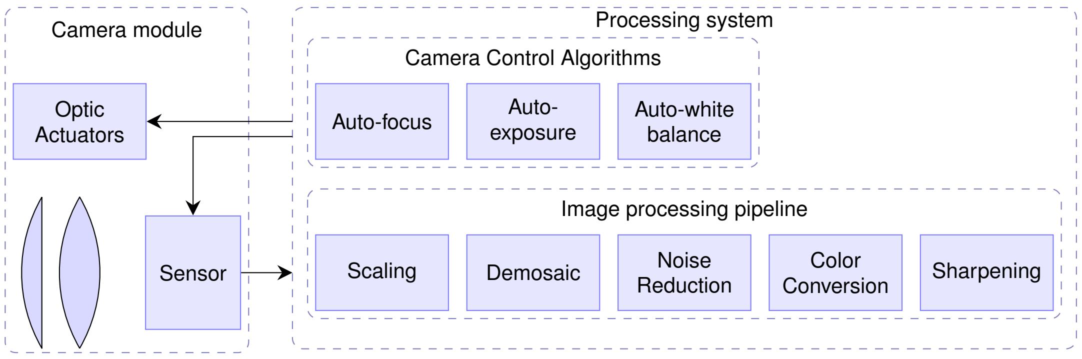


Figure 2.1. *Example of a camera system.*

A processing system for image data can be part of the camera module or it can be located in a separate integrated circuit, such as a system-on-chip (SoC). In embedded devices, such as mobile phones, the data transfer between the camera module and SoC is done with high-speed Camera Serial Interface (CSI). This is a standard specified by Mobile Industry Processor Interface (MIPI) alliance [9]. Many systems-on-chip vendors provide circuitry that has a dedicated processing subsystem, *an image signal processor* (ISP), to handle the data from external camera modules. Especially, modern SoCs developed for mobile phones and tablets do contain advanced ISPs. Examples of these can be found in chipsets produced by Qualcomm [10], Samsung [11], or Intel [12]. It should be noted that most ISP solutions are dedicated only for processing the raw data from the image sensor to provide a visually good quality images. These processing units are typically not exposed directly to the application developers and cannot be used to run arbitrary computer vision algorithms. However, as computer vision is gaining popularity in embedded system, chipset vendors and system providers are making solutions that, besides the ISP, have also computing resources that are suitable for running computer vision algorithms [7].

Figure 2.1 shows a simplified example of a camera system. A camera module consists of a lens arrangement, electro-mechanical actuators, and imaging sensor integrated into a single component. The lenses are moved with actuators in order to get the image focused correctly and with a zoom optics also to change the focal length. In fixed-focus system actuators are omitted and the optical system is optimized for a specific object distance. An example of a camera module is shown in figure 2.2.

Silicon-based solid-state imaging sensor are most commonly used. Two main types of sensors are charge-coupled device (CCD) and complementary metal-oxide-semiconductor (CMOS) sensors. Both operate with the same underlying physical phenomena of converting photons to an electrical charge that is accumulated in the pixel to a capacitance well. The key difference is the mechanism of how the charge is read out from the sensor. In CCD this happens by moving charges from pixel to pixel until they reach the edge of the chip were readout amplifier converts charge to a voltage that is further converted to a digital signal. In CMOS technology the pixel can be connected to readout directly with internal wiring. CCD is the older technology and still widely adopted for example in scien-

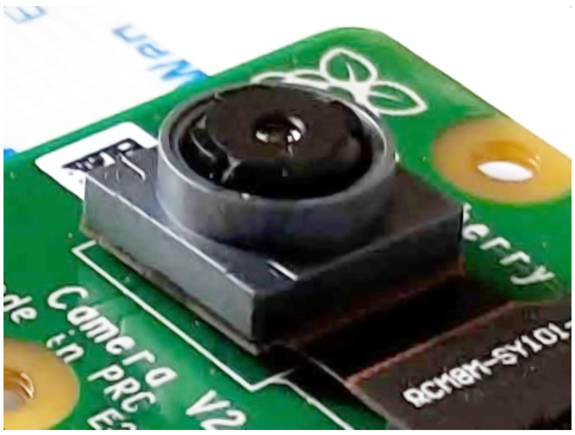


Figure 2.2. Camera module.

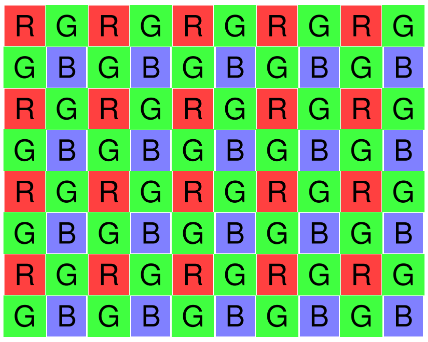


Figure 2.3. *Bayer color filter array.*

tific imaging. Over the last decade CMOS has gain high popularity especially in camera modules utilized in consumer electronics. Typical sensors do not capture all colors at all pixel locations, instead, pixels at the sensor are covered with optical color filters and thus a particular pixel is capturing only a portion of the light spectrum. Processing in the ISP is required to interpolate missing color values. The most common filter arrangement is the so-called Bayer-matrix, illustrated in figure 2.3, where pixels are covered with red, green, and blue filters [8].

The processing system is responsible for controlling the camera module and converting the raw data captured by sensor to the final image. Processing can be further divided into two parts as shown in figure 2.1. In the picture the upper part has camera control algorithms, such as automatic exposure control (AE), automatic focus control (AF), and automatic white balance (AWB). These algorithms analyze the properties of the captured images and adjust the movement of the optics, exposure time, and amplification of the sensor and behavior of other ISP algorithms. The lower part in the figure, the image processing pipeline, typically implemented with dedicated ISP hardware, performs computing-intensive algorithms for image data coming from the sensor. For color imaging the important processing step is the *color filter array interpolation* (CFAI), also referred to as *demosaicing*, where missing color values for pixels are interpolated [13]. Other important processing steps are different noise reduction algorithms and corrections for various distortions caused by the optical system. Typical ISP contains also algorithms for image enhancement and color correction to produce visually pleasing images [14][15].

2.1.1 Image quality for computer vision

The performance of the computer vision can be impacted by the quality of the image data. In general the target is to acquire as much as possible information from the scene. Images should have correct exposure and focus, so a relevant portion of the scene dynamic range is fully captured and spatial information is not lost. Optimal operation of exposure control and focus system is essential as failures in those parts are very challenging to overcome at later processing steps.

If the computer vision algorithm is utilizing color information, the reconstruction of a correct object color under different illumination is needed. A correct operation of the automatic white balance algorithm is critical in a such situation. Most algorithms perform generally quite well in typical scenes [16]. A challenging situation might arise if the scene has mixed illumination with very different color temperatures. For example, when a room is getting light both from sunlight coming through windows and fluorescent tubes [17].

Noise can also affect the performance of the computer vision. There are multiple noise sources present in an imaging system. Some of them are due to the manufacturing process of the sensor causing non-uniformity in the pixel responses. Others caused by thermal noise and other mechanisms in sensor electronics. A photon shot noise is due to the quantum nature of the light itself [8]. High illumination level typically gives a better signal-to-noise ratio (SNR),resulting in better performance of computer vision. Low light scenes are thus more challenging, in these cases the increase of the exposure time can improve SNR, at the possible expense of lower frame-rate and blurring of moving objects.

The processing of raw data from the camera sensor has also an impact on the performance of the computer vision system. Noise reduction algorithms in ISP are important for improving the SNR. The goodness of color filter array interpolation with various other scaling functionality is critical for how much spatial resolution there are left in the image after processing pipeline. It should be noted that sometimes the processing might also limit the computer vision accuracy. For example, when ISP performs operations that are optimized to produce a subjectively visually pleasing images, some details might have been removed from the image or some spatial structures are overemphasized due to the sharpness and contrast enhancement. For some computer vision applications it might be useful to bypass part of the camera processing or even just to operate with the raw sensor data.

2.2 Computer vision algorithms

Many computer vision solutions do require basic image processing methods such as scaling, cropping, rotation, and filtering of the image data. Also transforming the image to another representation, such as different color space is often needed, before more complex computer vision algorithms are applied. The range of computer vision applications is very wide, resulting to a vast amount of different algorithms and methods. There are still common concepts that serve as building blocks for solutions required by different use cases. Many of these are implemented as part of dedicated SW libraries such as OpenCV [18]. This section discusses about two computer vision concepts, object detection and object tracking.

2.2.1 Object detection

Object detection aims to find the location of a particular object in a picture. Object could be a real physical world object as a car, a person, or a table. Alternatively search targets could be a part of the physical object such as the human face. There are numerous approaches for object detection problem. Some of those have been developed to search a particular type of object, such as face and do relay explicitly properties known to belong to that object type, for example a particular geometric relation between eyes and mouth. Many others, all-though developed for a specific use case, are based on more generic features and can be applied to detect different objects. A good feature set is considered to be invariant to some of transformations in the image, for example, to the change of the global brightness. Examples of such features are histograms of oriented gradients (HoG) that was introduced for person detection [19] and Scale Invariant Feature Transform (SIFT) [20].

Having a means for extracting a relevant feature vector from a particular location at the image the task is to determine if features are representing the searched object or not. This is can be done by using a *classifier* to see if a feature vector is part of a wanted object class, such as the face, or not [21, p. 97]. Classifiers are normally created by using training data sets, representing examples of objects to be detected and negative examples that should not be detected. A simple classifier is a linear classifier that creates hyperplanes between samples belonging to different classes. To cope better with classes that cannot be separated with linear surfaces more advanced approaches as Support Vector Machines or neural networks are widely used [21].

For fast classification especially on general-purpose computer a cascade classifier can be used [22]. In this approach the classification happens with series of different classifiers. At each step a simple but fast classifier rejects an immediate large number of non-objects and passes only accepted potential detection candidates to the next classifier. The final result is then obtained at the last step. As the iteration stops when the candidate is rejected to be a part of the class algorithm executes fast on the typical images from which large portions can be immediately rejected.

To handle a size variation of the objects, a typical approach is to scale the image to different sizes, also called image pyramid-representation and the search is conducted in all different resolutions.

The PASCAL Visual Object Classes (VOC) [23] was a project that provided a dataset and yearly challenges between 2005-2012 for object classification, detection, segmentation, action classification, and person layout. The state of the art detector at the 2012, final year of the competition, were based on SVM classifiers working with bag-of-words (BOW) feature set. It is stated that performance in the last challenge year plateaued with classical approaches. The emergence of the neural network-based methods for detection brought significant performance increase, outperforming the state of art with more than 50% [24].

A with convolutional neural networks (CNN), not only classifier, but also features used for detection can be trained with the example data set [21]. Furthermore, object detection neural networks can cover the whole detection process, including the search of various image locations and scales into one execution of the networks that outputs the object position coordinates directly [25]. Neural networks for object detection are discussed more in chapter 3.

When considering embedded devices the complexity of the used object detector is important. Typical use case requires real-time processing in the environment that is limited not only by computational capacity, but also power consumption and heat dissipation are setting constraints [26]. When selecting object detection methodology for a specific use case, these implementation related limitations need to be considered parallel with the detection accuracy of the available algorithms.

2.2.2 Tracking

When object detection is done on the video signal it is often desired to combined information from successive video frames in order to track the object movement over time. One approach is to associate objects that coordinates are closest to each other from the current and previous frames to the same trajectory. In practice the object detection accuracy is varying and in some frames the object might be temporarily lost due to for example occlusion. To handle these situations the position data needs to be filtered in order to produce more reliable location information.

A well-known framework that is utilized to filter motion in such situations is a Kalman filter [27]. It is a recursive filter that maintains a system state vector $\mathbf{x}_{k-1}$ and corresponding uncertainty covariance matrix $\mathbf{P}_{k-1}$. The state vector contains for example, the current position and velocity of the object. Filter operates in two, typically alternating, steps:

1. **Predict the future state** $\mathbf{x}_k$ and covariance $\mathbf{P}_k$, based on model of the object motion. Model parameters can be based on the physical properties of the object and knowledge about how the environment impacts to it.
2. **Refine the state with new measurement** $\mathbf{z}_k$ and known measurement error given by covariance $\mathbf{R}_k$. The difference between predicted and observed state $\mathbf{y}_k$ is called *an innovation*, as it is the amount of new information gained. Kalman filter calculates from covariances $\mathbf{P}_{k-1}$ and $\mathbf{R}_k$ matrix $\mathbf{K}$, referred to as *Kalman gain*, that defines how much the prediction can be corrected with new measured information. The update for $\mathbf{x}_k$ and covariance $\mathbf{P}_k$ is calculated by adding to the predicted values the innovation multiplied by $\mathbf{K}$.

A tracking by detection, is a different approach for connected objects between frames. Unlike using object position coordinates and motion filtering, in this method objects from different frames are matched by utilizing their similarity [28][29][30]. The benefit of the method is the conceptual simplicity. Approach works also with low and varying frame rates when the object motion is not easy to predict. Also, all-though useful, the exact time information when frames are captured is not mandatory, as long as the order of the frames in time is known. Finally, it is noted that as the method does not require information about the camera position it could be also utilized in cases where camera is non-stationary.

The requirement for this method is that we have a means to compare the similarity between objects detected in consecutive frames. A similarity measure can be created by obtaining some features vectors from detected objects and measuring distances in the feature vector space. Small distance would correspond to high similarity. Sometimes features utilized in object tracking can also be adopted for this purpose, but they might not be discriminant enough with objects belonging to the same class.

The basic *tracking by detection* algorithm can be outlined in the following way:

1. For all detected objects c_i in the frame F_n find corresponding object p_i in previous frame F_{n-1} in such way that sum of distances between all paired objects is minimized.
2. If a distance between a pair of objects in consecutive frames is smaller or same than predefined threshold t_1 add object c_i into an existing trajectory. If the distance is above the threshold start a new trajectory.

At the presence of more than one object in successive frames, the target is to find the best combination for all possible pairs. This can be achieved by solving a *linear sum assignment problem* between feature vector distances. The solution of the problem represents the minimum sum of all distances within all possible combinations between objects in these frames. Method published by Harold W. Kuhn [31] and James Munkres [32], called a *Hungarian algorithm* or *Kuhn-Munkres algorithm* can be used to solve this problem in polynomial time. There exists several variations of the implementation of the algorithm, one of those is presented in the appendix A.

For object tracking it is also possible to combine both coordinate-based and detection by tracking approaches. This can be done by using one of the methods as a primary means for creating trajectories and using the other for complementing it. For example, when tracking with position-based method object trajectories that overlap in the view might get mixed. In a such situation comparing the similarity of objects might help to resolve into a better solution. Likewise the similarity measure in tracking by detection can be include also position information. This would prevent potentially erroneous matching when, for example, two very similar objects are at the same time present in the picture.

3 NEURAL NETWORKS

This chapter will give a general introduction to machine learning and will then focus more specifically on neural networks. Principles of the networks are described in early sections followed by an introduction to some network structures used in visual information processing tasks. Use cases of object detection and re-identification are covered more in detail. Finally we discuss about implementation of the networks with relevant HW and SW frameworks.

3.1 Overview

Artificial Intelligence (AI) is a general term used to describe information processing systems that aim to solve problems that do require similar pattern recognition and deduction capabilities as humans are able to do. Detection of objects from the scene, converting speech to text, or processing of natural language are examples of such problem domains. Also cases, where the computer system has been able to beat best humans in complex strategy games as in *chess,* [33] and in *go* [34] can be noted as significant milestones in the development of the AI.

Machine learning is a subset of AI methods where the logic required to solve the problem is not coded directly. Instead the system will try to analyze example data and learn to provide the desired response also when new unseen data is set as an input. Learning can be further divided into supervised and unsupervised (self-supervised). In supervised learning a system will be trained by giving examples of input and desired output. For example, a training input could be a picture of a dog and the desired output is the 100% probability that it belongs to a class of dog-images. During the training phase a system will try to adapt itself in such a manner that differences between input and given output are minimized. After successful training the system should be able to provide meaningful output for novel input data. In the dog picture example this would mean that output for a new dog image, not used at the training phase, would also result in a high confidence

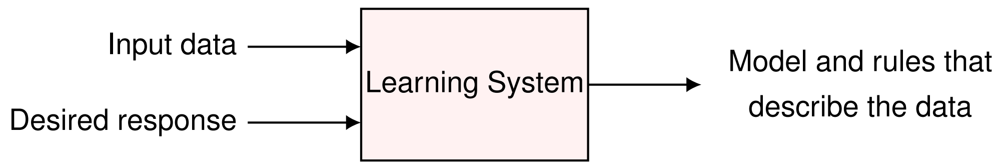


Figure 3.1. *Principle of the Machine Learning*

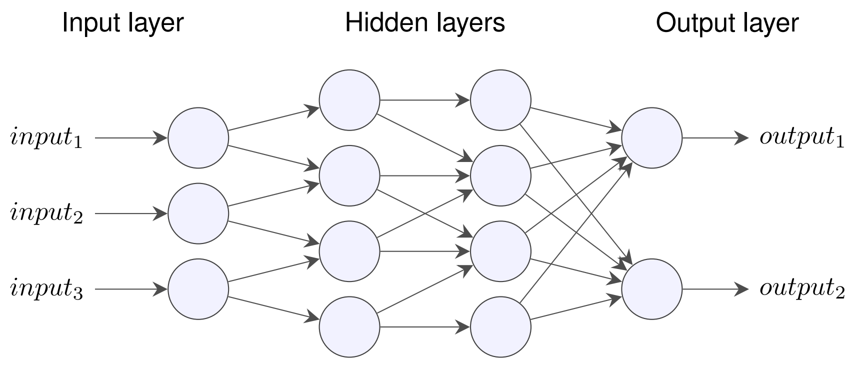


Figure 3.2. *Simple Neural Network*

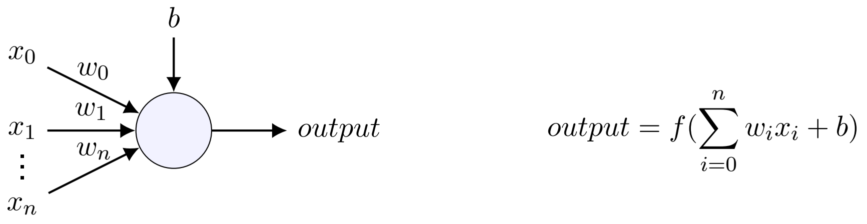


Figure 3.3. *Single Neuron*

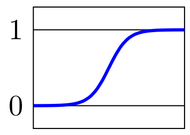
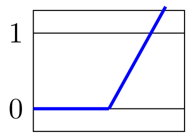
for dog-image class and a low confidence for e.g. cat-images. Figure 3.1 shows the principle of supervised machine learning. In unsupervised learning system will try to create models of the data without explicit examples of the desired output. These methods are typically used to find structures and similarities from the data. A well-known example of unsupervised learning is a SOM (Self Organizing Map) [35].

Neural networks are a family of are widely used machine learning methodologies. Early neural networks have been inspired by information processing in biological brains [36] [37]. But one should be careful not to make too strong links between these models and the real functionality of the brain. Studies in neuroscience indicate that information processing already in neuron level is typically more complicated than operation in the artificial neural models [38].

A typical neural network can be seen as a directed graph where each node will do an operation to the input and will forward its output to the next *layer* of nodes that are called *neurons*. Figure 3.2 illustrates simple neural network. The input data moves trough 4 neuron layers and output is defined by the operations that each neuron will perform to the data. First layer is called as *input layer* and last layer *output layer*. Layers in between are called *hidden layers*. If all neurons in a layer are connected to all neurons from the previous layer, the layer is called *fully connected.* Each layer can be seen as a transformation of the data to a new representation. Operations performed by each neuron and the topology of the network, how neurons are connected and combined as layers, is decided according to what kind of data processing tasks network should perform.

Operation of the single neuron is illustrated in figure 3.3. Input data x_i is multiplied by corresponding weights w_i and result is summed together with additional bias term b and

Table 3.1. *Examples of neuron activation functions*

Type of neuron	Activation function	Example Graph
Logistic sigmoid	$f(x) = \frac{1}{1 + e^{-\sum_i w_w x_i + b}}$	
Restricted Linear Unit (RLU)	$f(x) = max\{0, \sum_i w_w x_i + b\}$	

fed through non-linear *activation function* $f()$. In vector form this can be expressed as,

$$y = f(\mathbf{w}^\mathsf{T}\mathbf{x} + b). \tag{3.1}$$

Where y denotes the scalar output of the neuron with input vector $\mathbf{x}$ and weight vector $\mathbf{w}$. Activation functions have a significant impact on how the neuron and the whole network operates. Also the non-linearity is important, if all activation functions would be linear, the successive operation could be combined to a single neuron. Typical activation functions are shown in table 3.1.

The final layer of the network is chosen according to what kind of output is needed. This is defined by the purpose of the network. Typically final layers are *fully connected*, all neurons do receive input from all neuron previous layer. Table 3.2 shows some examples of network output units and typical situations for what they could be used.

Table 3.2. *Examples of network output units. Vector* $\mathbf{h}$ *represents the output of the last hidden layer while* $\hat{y}$ *and* $\hat{\mathbf{y}}$ *are the final output of the network.*

Output unit		Description
Linear	$\hat{y} = \mathbf{w}^\mathsf{T}\mathbf{h} + \mathbf{b}$	Scalar output as linear combination of the hidden units, suitable for various task where a continuous output is required.
Logistic sigmoid	$\hat{y} = \sigma(\mathbf{w}^\mathsf{T}\mathbf{h} + \mathbf{b})$	Scalar output limited between 0 and 1 with sigmoid function. This is suitable for two class classification problems.
Softmax	$\hat{y}_i = \frac{e^{\mathbf{w_i}^\mathsf{T}\mathbf{h}+\mathbf{b_i}}}{\sum_j^n e^{\mathbf{w_j}^\mathsf{T}\mathbf{h}+\mathbf{b_j}}}$	Vector output $\hat{\mathbf{y}} = [\hat{y}_0, \hat{y}_1, ..., \hat{y}_n]$ as normalized probability distribution, for example for predicting the probability of input belonging to each of i output classes.

3.2 Training of neural networks

In the machine learning aim is to capture the statistical properties of the training data in a model output distribution $M_\theta(x)$, parameterized by θ, in such way that the systems ability to predict correct output $y \in Y$ from input $x \in X$ is maximized. The correctness of output is given by a *loss function*, that measures the difference between network output $M_\theta(x)$ and the desired output y defined in the training data. Loss functions are task-dependent and various criteria can be used to derive suitable loss function [21].

In *maximum likelihood* framework the optimization task can be expressed as maximizing a conditional distribution in respect to the model parameters [21, p. 129],

$$\theta_{ML} = \arg\max_{\theta} P(Y|X : \theta). \tag{3.2}$$

Where X represents all inputs and Y all outputs. The task is to find model parameters θ_{ML} that do maximize the likelihood of producing correct y given x. This can be done by minimizing a dissimilarity measure between the training distribution and the model distribution. The *cross-entropy* between two distributions P, Q can be used as a measure of dissimilarity [21, p. 73] and is defined as

$$H(P, Q) = -\mathbb{E}_{xP}[\log Q] \tag{3.3}$$

In case of discrete probability distributions, this can be expressed

$$H(P, Q) = -\sum_{i} P(x_i) \log Q(x_i). \tag{3.4}$$

Based on this, as an example, for a data set $T = \{(\mathbf{x_1}, \mathbf{y_1}), (\mathbf{x_2}, \mathbf{y_2}), ..., (\mathbf{x_n}, \mathbf{y_n})\}$, where each element $(\mathbf{x_i}, \mathbf{y_i})$ is a pair of vectors of length m, $\mathbf{x_i} \in X$ defining the input and $\mathbf{y_i} \in Y$ giving corresponding desired response and $\mathbf{\hat{y}_i} = M_\theta(\mathbf{x_i})$ is the output of the network for input $\mathbf{x_i}$, the loss over the whole set can be defined as,

$$L(T, \theta) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m} y_{ij} \log \hat{y}_{ij}. \tag{3.5}$$

By minimizing the $L(T, \theta)$ is respect to θ the likelihood in equation 3.2 is maximized. This example can be adopted for a classification task where the network aims to predict the probability of input belonging to a particular class of set size m, each element of $\mathbf{\hat{y}_i}$ representing likelihood for a single class.

The use case of the network defines the requirements for the loss function. The choice of the loss function is also closely related to the selection of the output unit. In some use cases also multiple loss functions can be used together simultaneously to optimize different aspects of the network output.

3.2.1 Optimization with gradient descent

In practice the training of the network is an optimization process where the weight values of the neurons are changed with iterative optimization techniques until that network performance defined by a loss function is reaching an acceptable level. Gradient descent is an iterative algorithm that tries to find a local minimum of the function by calculating the gradient and updating each of the weights w, to the direction pointed by corresponding partial derivative as shown in equation 3.6

$$w_{updated} = w - e\frac{\partial L}{\partial w} \tag{3.6}$$

parameter e is *learning rate* and defines how much weights are updated at each iteration step.

The direct minimization of function like 3.5 is often unpractical, especially if the training set is very large. To overcome this issue *stochastic gradient* descent (SGD) algorithms are used [21, p. 286]. SGD combines the result of individual gradient descents calculated with a subset of training samples, called *minibatches*, to joint updates of the weights that statistically drive the cost function towards the optimization target.

Stochastic gradient descent algorithms have many variations and extensions that aim to improve the convergence speed and the accuracy of the optimization. A method of momentum [39] aims to speed the convergence using a weighted moving average of past gradients at the update step, keeping up the gained "momentum" of the decent toward to the minimum. Many SGD algorithms also focus on adaptive optimization of the learning rate, as it has a significant impact on the optimization process and hence also to the final network performance [21, p. 286]. Examples of these are AdaGrad [40], RMSProp [41] and Adam [42].

Besides the actual optimization algorithm the learning process of the network can be improved by adding additional layers to the network structure during the training phase. An example of these is the *batch normalization* [43] layers that can be placed in-between of any layer of the network. Batch normalization calculates the mean and variance of the input values for a layer over the current batch of training data. After that it normalizes values by subtracting the mean from the input and divides the result by the variance. This creates a new normalized input for the layer. As the input data ranges for layers are more close to each other after the normalization, the operation of the SGD algorithm is more stable. Without normalization, gradients are more likely to vanish for the small weights, effectively prevent the weight updated, or the opposite, having very large gradient values for large weights could cause optimization to diverge [21, p. 309].

When a network is trained it is important to ensure that the *overfitting* of the model is

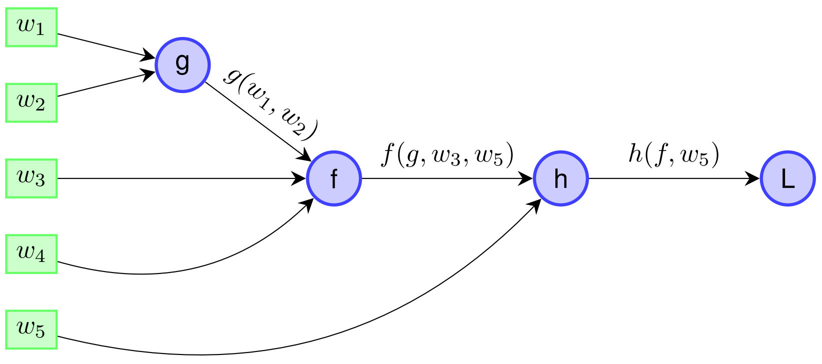


Figure 3.4. *Computational graph for* $L(\mathbf{w}) = h(f(w_3, w_4, g(w_1, w_2)), w_5)$

avoided. If the training data set is too small with respect to the network complexity it is possible that the network learns to perform well on that training data but has lost the ability to generalize, resulting in a poor performance with novel input. It is advisable to divide available data to at least 2 parts. A training set, used during the optimization and a validation set used for testing network performance. If the performance of the validation set is significantly worse than with training set it is an indication of possible overfit.

3.2.2 Backpropagation

A gradient descent optimization method requires a way to calculate the gradient of the loss function with respect to each parameter of the network. A key tool in this process is the *back-propagation algorithm* [44] that is an efficient method for calculating loss function gradient over the whole network using a chain rule of calculus and propagating cost from the end of the network backward.

To illustrate the principle of the backpropagation algorithm, let's consider a function $L(\mathbf{w})$ that can be written as combined function of 3 other functions f, g, h:

$$L(\mathbf{w}) = h(f(w_3, w_4, g(w_1, w_2)), w_5). \tag{3.7}$$

Figure 3.4 show computational graph equivalent to equation 3.7. Output of the function can be obtained with *forward* pass by calculating first inner function g followed by f and h progressing from left to right in the graph. For example, when wanting to calculate the partial derivative

$$\frac{\partial L}{\partial w_1},$$

the forward calculations can be also utilized. According to chain rule in calculus we can write

$$\frac{\partial L}{\partial w_1} = \frac{\partial h}{\partial w_1} = \frac{\partial h}{\partial f}\frac{\partial f}{\partial w_1} = \frac{\partial h}{\partial f}\frac{\partial f}{\partial g}\frac{\partial g}{\partial w_1}. \tag{3.8}$$

This implicates that we can calculate the derivative proceeding backward from right to left. If the input to a particular node in the graph is already calculated and stored during the forward pass it is available for derivative calculation of the following node. In this way there needs to be explicit formulas for calculation of the derivatives only each of the nodes instead of the need to reconstruct derivative function for the whole graph. A loss function of a neural network can be seen as a similar computational graph where neurons are functions of their input weights. As the whole network is a compound function of the individual neurons, a chain rule principle can be utilized to calculate partial derivatives with respect to each weight.

3.3 Convolutional neural networks

The neural network is considered to be a deep, when it contains several hidden layers. There is no strict definition of how many hidden layers are required to qualify as a deep neural network (DNN), but network with only one hidden layer is considered to be "shallow". Typically modern deep networks do consist of tens or even hundreds of layers. Theoretically a single hidden layer network can approximate any practical multilayer network that is using typical activation functions [45]. However, empirical data shows that deeper networks are able to generalize better [21, p. 198]. Some problems like visual pattern recognition do fit intuitively very well to the layered structure where each of the processing layers changes the representation of the previous step towards the desired output.

An important sub-category of DNNs are convolution neural networks (CNN). These are architectures having layers that perform convolution operations to the input data. A discrete convolution is an operation where a *convolution kernel* is sliding over the input data. At each point kernel values are multiplied with data values and results of the multiplication are summed together to provide convolution output corresponding to each input sample point. Formally a discrete convolution of two finite sequences f, g can be expressed as

$$f[x] * g[x] = \sum_{m=0}^{M} f[m]g[x-m]. \tag{3.9}$$

In image processing the use of a 2-dimensional kernel, also called a mask or filter window, is typical. Convolution can be extended to higher dimensions in a straightforward manner:

$$f(x,y) * g(x,y) = \sum_{m=0}^{M} \sum_{n=0}^{N} f(m,n)g(x-m,y-n). \tag{3.10}$$

In CNN convolution layer consists of a set of the kernels which weights can be learned during the training. A convolution layer has *input channels*, corresponding to the outputs of previous layers and *output channels*. Inside a layer a single convolution kernel takes a vector-valued input, combining several input channels and provides output channel of

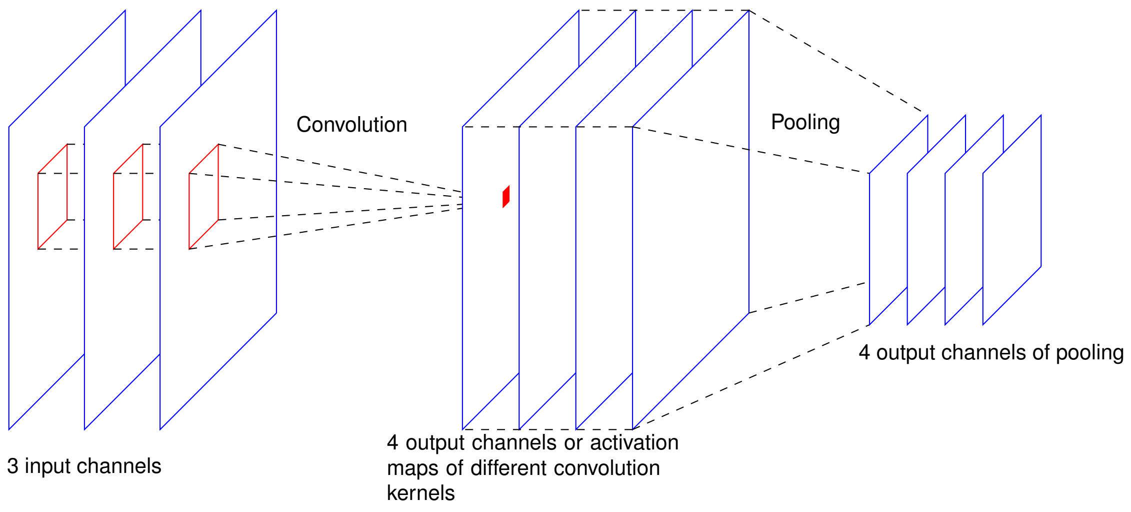

Convolution
Pooling
4 output channels of pooling
3 input channels
4 output channels or activation maps of different convolution kernels

Figure 3.5. *Main operations, convolution and pooling of Convolutional Neural Network (CNN).*

that kernel, also called as a feature or activation map. In typical CNN kernel weights for each output channel are shared between different spatial locations. One convolution layer can have several kernels, each providing their own output channel. Stacking the output channels of the all kernels in the layer provides the full output of that convolution layer.

Typically CNN consists of successive convolution layers where after each layer there is sub-sampling performed by *pooling layer*. Pooling layers do combine data that is near to each other spatially. The most common pooling is *max pooling* where layer output is the maximum value inside a defined input window. Figure 3.5 gives an example of convolution and pooling operations. A kernel slides over 3 input channels and does convolution with the data on the regions marked with a red outline. The same is repeated with 4 different kernels in the layer, creating 4 output channels. These are then sub-sampled to a lower resolution by pooling operation. In CNN architectures convolution layers are typically followed by layers of fully connected neurons with non-linear activation functions to provide the final output of the network.

3.3.1 Image classification

Image classification task aims to provide information to which class an image belongs. Classification can be based on the objects that are in the view, for example, pictures containing humans, animals, vehicles can be classified into separate groups. These classes can also have a sub-classes such as images with dogs or cats, both part of the animal class. Classification can be also based on more generic properties of the picture, such as images taken in the morning or evening, outdoors, or indoors. In many case classes at the same hierarchy level can be considered mutually exclusive, a picture cannot be the same time taken in the evening or morning. Classification task where image belongs to multiple classes is also possible and also is referred to as image labeling.

Image classification research has advanced greatly over the last decade. ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [46] was an online competition for image classification, carried out between 2010 and 2017. In 2012 convolution neural network-based solution named as Supervision, later better known as AlexNet [47] won the challenge. Since that all the winners of the competition were based on deep convolution networks, achieving better and better results.

When a CNN is trained for classification tasks, convolution layers do work as *feature extractors*. Typically first layers create output that matches the small scale structures of the images, such as flat areas, edges, or textures. Kernels in the following layers learn activation to the high-level features such as eyes, mouth, nose, and layers after that abstract it further to respond for example to a human face. The final layer in a typical classification task is a softmax activation that produces a vector $\hat{\mathbf{y}}$ that gives probabilities of the input belonging to each of the classes. For example, the presence of the face would trigger a high response to the output unit that gives the probability of an image belonging to a "person"-class.

Training data consists of input images and desired output vector $\mathbf{y}$ of length C, corresponding to the number of classes. Each element y_j gives a probability of image belonging to one of the classes. In case of mutually exclusive classes only one of the elements has value of 1, rest being 0. Loss function in maximum likelihood framework used for classification is called a categorical cross-entropy defined as,

$$L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C} y_{ij}\log \hat{y}_{ij}, \tag{3.11}$$

Image classification has been an inspiration to several neural network designs, examples of these are Inception [48], MobileNets [49], VGG [50], and ResNet [51]. The trend has been for deeper networks and many advances in the network design have been related to how to improve the training of these deep structures. Successful examples of such design are *Resnet*-architectures. Besides advances in the classification accuracy there has been also a focus on network architectures that can do require less computational resources. As an example *Mobilenet*-architectures can be implemented with real-time performance in devices such a mobile phones.

CNN architectures for classification are in many cases used as substructures for network designs targeted for other tasks as well. For example, networks designed for *object detection* and *re-identification* can utilizing classification architecture as a *backbone network*.

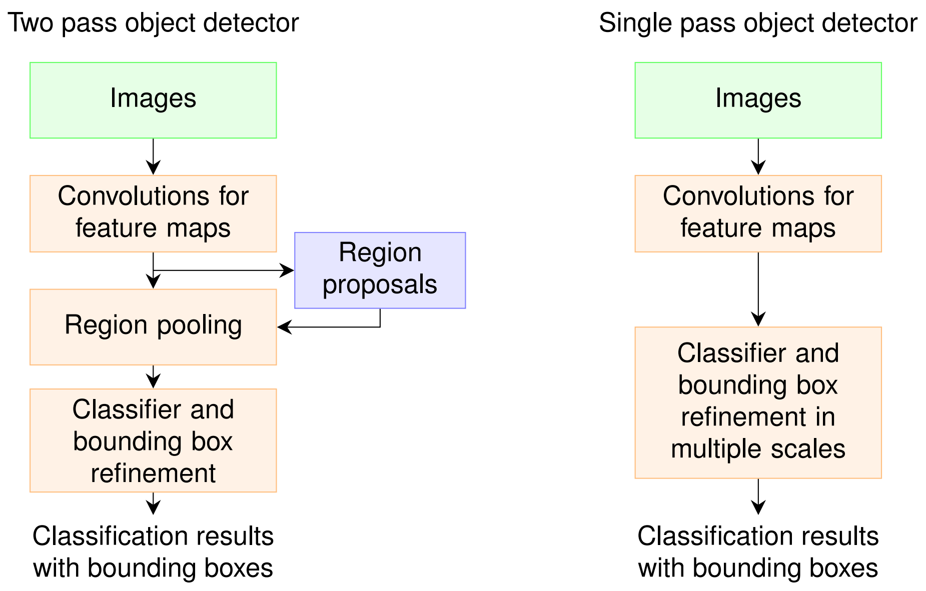

Two pass object detector
Images
Convolutions for feature maps
Region proposals
Region pooling
Classifier and bounding box refinement
Classification results with bounding boxes
Single pass object detector
Images
Convolutions for feature maps
Classifier and bounding box refinement in multiple scales
Classification results with bounding boxes

Figure 3.6. *Comparision of one and two stage object detector architectures.*

3.3.2 Object detection

Overview of the object detection problem and some solution approaches is discussed in 2. As stated the development of the performance of classical approaches was somewhat stagnated and the introduction of novel methods based on DNNs brought significant performance improvements over the previous state of the art [24]. Research in the field is very active and several solutions have been created to improve both detection accuracy and speed. There are two primary architectural approaches of object detection with CNN illustrated in figure 3.6.

Two-pass object detection networks do create first a set of proposals for potential object locations. The presence of the object is then inspected more closely only at these regions, typically running a classifier to recognize the object type and regression stage to refine the object bounding box. For high accuracy the regressor could be a class-specific. Several high-performance networks have been designed with different flavors of two-pass architecture. Examples of these are the Region-based Convolutional Neural Network (RCNN) family, RCNN [24], Fast-RCNN [52], and Faster-RCNN [53]. While achieving high accuracy the speed of two-pass networks is not as high as with single-pass architectures [54].

Single-pass architectures do not have a separate proposal phase, network output is produced as the result of the single forward pass. Examples of this type of architectures are You Only Look Once (YOLO) [54] and Single Shot Multibox Detector (SSD) [25]. Both aiming to predict both class probabilities and object locations.

The principle of the SSD structure is shown in figure 3.7. The design consist of a backbone network, followed by additional convolution layers providing detection results to the final non-maximum suppression layer. The base network is a truncated version of a CNN

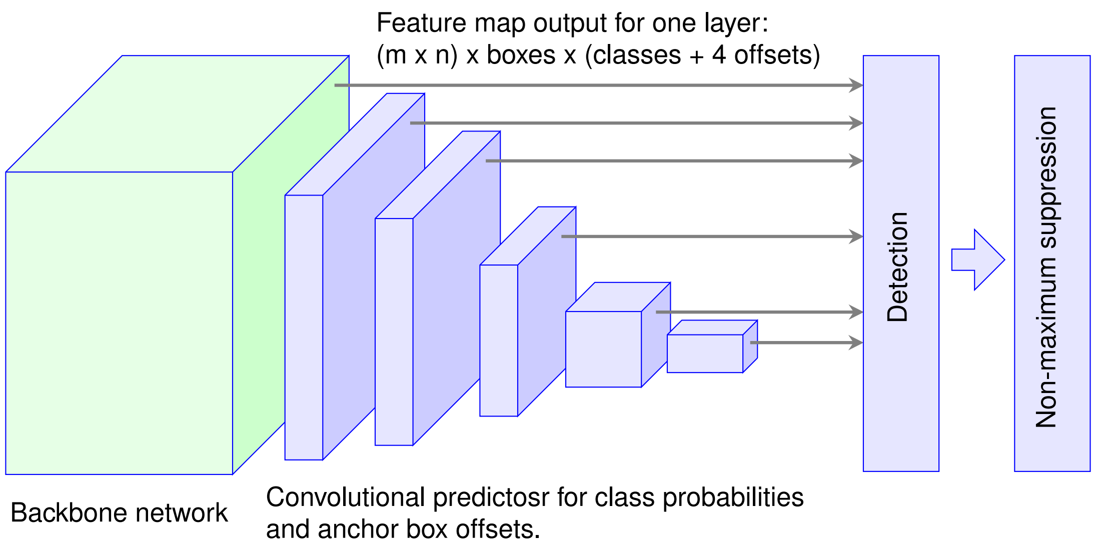

Feature map output for one layer:
(m x n) x boxes x (classes + 4 offsets)
Detection
Non-maximum suppression
Backbone network
Convolutional predictosr for class probabilities
and anchor box offsets.

Figure 3.7. *Principle of SSD [25] architecture.*

image classifier, without fully connected classification layers. In the original version this base network was VGG16, but other networks can be used as well. Popular choices for base network are ResNet, when high detection accuracy is desired or Mobilenets, when processing speed is more important.

The operating principle of SDD is the following: After a feature map is created by the base network, a series of convolutional layers with decreasing spatial size are applied in order to provide additional feature maps and predictions at different scales. Position proposals for interesting regions are not searched separately, instead, a set of predefined anchor boxes with fixed positions and aspect ratios are considered. In each location and aspect ratio of the anchors, a small convolution kernel calculates the confidence value for each class being present in the region. Similarly the object location is also predicted with convolution providing 4 offset vectors that refine the original position of the box. The location predictor is not class-specific, but the choice of different anchor box aspect ratios is done based on the analysis of various real-life objects to give coverage for different object shapes. Predictions from all layers are collected and non-maximum suppression operation is performed to provide output only the highest confidence classes.

Training loss for SSD has two parts, *a localization loss* L_{loc} and *a confidence loss* L_{conf} and total loss is defined as,

$$L = \frac{1}{N}(L_{conf} + \alpha L_{loc}), \tag{3.12}$$

where α is weight term set to 1 by cross validation [25].

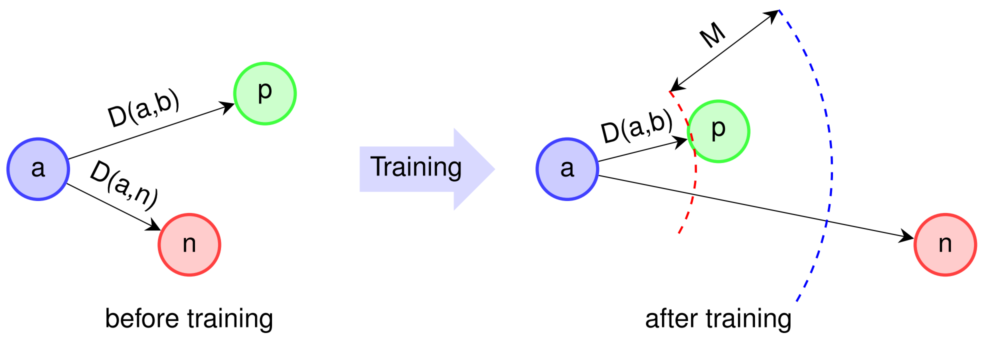


Figure 3.8. *Principle of triplet loss [55]. Training drives samples belonging to same class closer and saples from different classes apart of each other. If negative sample is further than $D(a,p) + M$ indicated by blue arc, the loss is 0.*

3.3.3 Re-identification

Re-identification is a task where the target is to find objects that are similar to a presented example. A typical practical scenario is a search where an image of an object is compared to the images in the database and the target is to retrieve a picture containing the same object. Another example use case is people tracking in a surveillance application, where the target is to find a person that has been in the view of a particular camera and moves then to another view.

The design target for the re-identification algorithm is to convert an image containing the object of interest to a new representation in a new feature embedding space. Objects similar to each other, for example, images representing the same person, should be near to each other, with respect to some metric or similarity measure, in this feature space. Likewise dissimilar objects should have a large distance between them. Additionally, the feature representation should be invariant to different scales, illumination conditions, camera angles, or different backgrounds of the objects. Furthermore, also the transformations of the object itself, for example, a different pose of the person should not impact to the distance significantly.

One possible approach for re-identification is to use CNN. It can be noted that when a network is trained for the classification task, the output of the layers before the actual classifier provide a feature representation of the incoming picture. This observation can serve as a base for the re-identification network design. The target is to train network to classify images of the same object, such as a person, to a single class. In addition to that other loss functions that reward from the close distance between feature vectors mapped to the same class and far distances between classes can be used.

The key aspect in the training of a network to produce for re-identification features is the definition of a suitable loss function. Widely used example of such function is a *triplet loss* [55] that was developed originally for face recognition tasks and has been also utilized in re-identification. In triplet loss 3 samples from the training set are chosen randomly: *anchor*, *positive* and *negative*. *Positive* is picked randomly from the same class as the

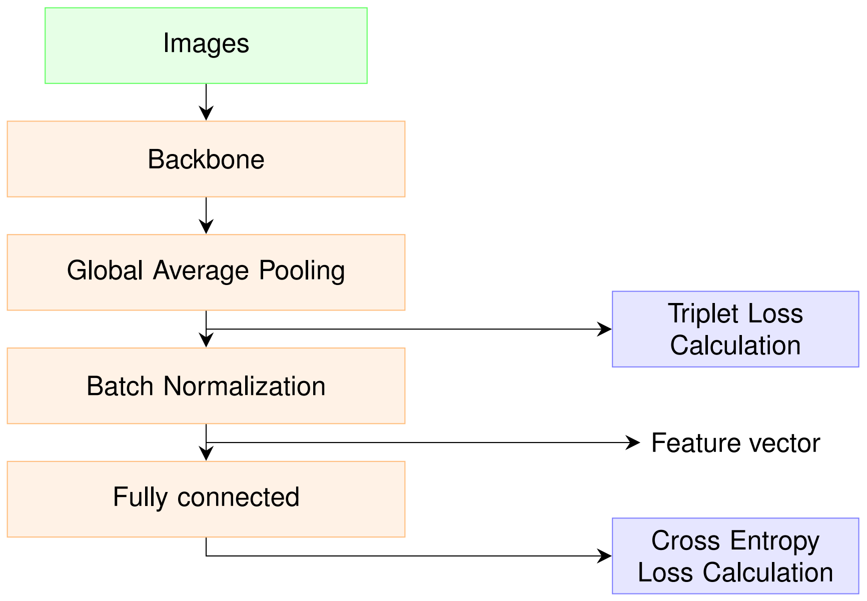


Figure 3.9. *An example of a person re-identification network structure, adopted from [59].*

anchor and *negative* picked randomly from a different class from the *anchor*. All of these samples are fed through the CNN that outputs a corresponding feature vectors $\mathbf{a}$, $\mathbf{p}$, $\mathbf{n}$. The loss is defined as

$$L = max(D(\mathbf{a}, \mathbf{p}) - D(\mathbf{a}, \mathbf{n}) + M, 0), \tag{3.13}$$

where D is a function giving a distance between vectors, and M, refering to *margin* is a tunable parameter defining the threshold where condition $D(\mathbf{a}, \mathbf{p}) + M < D(\mathbf{a}, \mathbf{n})$ is met. In this condition the output of the loss is 0. In other situations loss is positive. Figure 3.8 illustrates the principle of the triplet loss.

The research on different loss functions is active and several different approaches have been proposed, these include *lifted structured loss* [56], *archface loss* [57], and *ranked list loss* [58].

A representative example of re-identification with neural networks is presented in the work of Luo *et. al.* [59]. The work proposes a strong baseline implementation of DNN architecture for person re-identification. Figure 3.9 illustrates the key components of the proposed architecture. At first a backbone model such as MobileNet [49] or ResNet50 [60] computes the feature maps of the input images. This is followed by the global average pooling layer, which calculates the mean value of each channel. Results move through a batch normalization layer to a fully connected classification layer.

In training phase the triplet loss $L_{Triplet}$ and the center loss L_C are calculated from the output of the global averaged pooling layer. This is combined with the categorical cross-entropy loss L_{ID} calculated from the output of the final fully connected layer. Total training loss is the sum of these 3 losses,

$$L = L_{ID} + L_{Triplet} + \beta L_C, \tag{3.14}$$

where the weight value β is used to adjust the impact of the center loss.

In the inference stage, the output of the batch normalization layer represents the embedded features of the images, and the cosine distance metric is employed to calculate the distance between the feature vectors of two images.

3.4 Frameworks

Deep neural networks can be seen as a series of successive matrix or tensor operations. This formulation offers a practical way to express different layers as a combination of standardized building blocks. There exist several software frameworks that can be used to represent networks in this manner. Tensor algebra, where a large number of similar operations are performed on the regular data structures, is also suitable for efficient mapping of implementations on parallel computing platforms like GPUs (Graphics Processing Unit) or specialized TPUs (Tensor Processing Unit). Most of the popular SW frameworks do offer optimized GPU.

In addition to the above-mentioned solutions, there are SW libraries that are not targeted for constructing and training networks but do focus on the efficient execution of already trained networks. OpenCV computer vision library has a dedicated DNN-module that offers functionality to run networks created with different frameworks. Also OpenVino™ framework, promoted by Intel™ offers tools for optimized network execution especially on HW produced by Intel.

The key concept in OpenVino is Intermediate Representation (IR), a format for storing DNNs. OpenVino includes a tool, called *model optimizer*, for converting networks from popular frameworks to IR format. This IR can be then used as an input for different *Inference Engines*, implementation optimized for running network on a particular HW platform. Currently OpenVino offers an inference engine for certain field-programmable Gate Array (FPGA) architectures and processors like Visual Processing Units (VPU), CPUs, and GPUs. Concept of OpenVino is shown in figure 3.10. ONNX [61] is an open network format targeting similar interoperability between various frameworks and implementation platforms.

Some of the popular frameworks are listed below:

- **Tensorflow™** [62] is an open-source machine learning platform, supporting Python, JavaScript and Swift programming languages. It offers a framework to create and train networks and deploy them to CPU or GPU environments. Pre-trained models are also offered.
- **Keras** [63] is a higher level Python abstraction layer build on top of Tensorflow. It

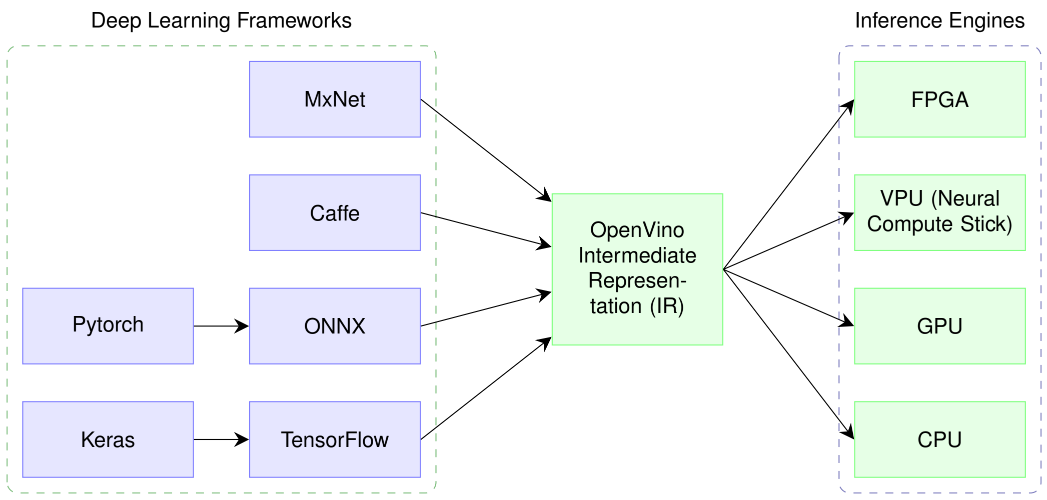


Figure 3.10. *Deep learning frameworks and conversion to OpenVino format*

provides user-friendlier API than Tensorflow and enables fast development of the model from idea to implementation.

- **PyTorch** [64] offers Python API similar to Keras for model development. It supports network acceleration on GPU and is also supported by many cloud computing platforms,
- **Caffe** [65] Caffe is one of the first deep learning framework that have gained high popularity. It offers Python API and GPU acceleration. Pre-trained models are also available.
- **MXNet** [66] is a similar framework than Tensorflow offering tools to build, train and deploy networks. It offers bindings to 8 different programming languages.
- **ONNX** [61] is not a framework for model development, but it is an open format built to represent machine learning models. Target is that models build with some framework could be converted to ONNX format and further deployed for execution on the platforms that support ONNX.
- **OpenVino™** [67] framework, promoted by Intel™ offers tools for optimized network execution especially on HW produced by Intel.

The field is rapidly evolving and new features are continuously added to the frameworks. There exist also SW libraries that are built on top of previously mentioned frameworks. These are typically targeted for a specif use case. For example, a Tensor Flow Object Detection API [68] and Detectron2 [69] do offer both code and pre-trained network model for object detection.

4 EDGE DEVICE FOR PEOPLE COUNTING

This chapter describes the implementation of an edge device solution designed for visitor counting of public museum center. Edge computing is a distributed computing paradigm, in which part of the computing or data storage does happen near where the data is gathered, rather fully taking place is remote servers. This is to improve response time, reduce transmission bandwidth, utilize distributed computing resources to balance load or, like in this case, ensure that privacy-sensitive data is processed locally and not sent to the network [70].

Aim is to design and implement a system of connected computing devices that can transfer data over a network without requiring human control. These are also referred to as an Internet-of-Things (IoT) devices. Solution space is constrained by cost, processing power requirements, physical size, power consumption, and availability of suitable hardware. The described implementation is based economical solution of using a single-board computer platform with an additional neural computation accelerator.

The next sections start with the use case analysis and derive requirements for the implementation, followed by discussions on high-level design choices. After that implemented HW and SW solutions are covered in more detail. Finally, neural networks and their deployment for the use case are covered.

4.1 Use case description

The museum center has different exhibitions and there is a need to understand how many people are visiting a particular exhibition during a day. In addition to counting people, information such as the duration of the visit and statistics of most regularly used entry and exit ways are interesting. Demographic data from visitors such as gender and age would be also useful.

The privacy of the visitor should be respected. Premises do contain already a separate security surveillance camera system and appropriate notifications informing visitors that video surveillance is ongoing are in place. However, the system used for people counting should not be used for similar purposes. The data that makes the identification of individual possible should not be recorded. Specifically when the data is stored to the server it should be ensured that this data does not contain personal information and does not require handling required by GDPR legislation.

Museum environment sets restrictions to the camera placement. The set-up should be discreet in order not to disturb the actual exhibition. For the same reason devices should be silent. Some objects in the museum might block a view. Also as some of the object might be statues or pictures of the persons that sets a requirement for the person detection system to be able to ignore these.

Based on the above requirements the primary design targets for the system can be stated:

- Camera units that are able to detect a person in real-time and calculate a unique feature vector that is sent to an external server. Images should not be stored anywhere, not even locally to the devices.
- A cloud server that stores that information sent by camera units to a database.
- Analyses SW that reads the data from the database and provides a visitor count report.

4.2 Hardware design

Hardware choices for this project are based on earlier work [26], where the feasibility of the single-board computer system for computer vision application was analyzed. Single-board computers do offer a practical and economical way to build product prototypes or even actual commercial products, when the volumes are relatively small and design and production costs for a dedicated computer would be too high.

Earlier work indicates also that for advanced computer vision application that uses neural networks, the CPU processing power of a single-board computer might not be sufficient. For example, although it is possible to run SSD based object detection on CPU the frame-rate would be limited to less than 1 fps (frames per second). In this case CPU load would be near 100%, leaving limited capacity for running any other SW. When the processor load is in a high level for several minutes it can result in processor core temperature raise to the level where performance throttling would start to lower the clock frequency. This could finally lead to the thermal shutdown of the whole system. Fortunately these issues can be greatly reduced or even avoided by using additional accelerators for neural computation that would offload the heaviest load from the mainboard CPU [26].

Following key hardware components are chosen for the device:

- **Raspberry Pi 3B+**

  A single-board computer that is built around Broadcom BCM2837B0 system-on-chip. The CPU is 64bit quad-core ARM Cortex-A53 with 1.4 GHz clock frequency. The chip contains also a graphics processing unit (GPU) called VideoCore® IV that is responsible for 3D rendering, processing of the camera data, and video encoding and decoding. Board contains also dual-band WLAN chip and physical Ethernet and USB ports. Power is fed through a micro-USB connector [6].

- **Raspberry Camera Module V2**

  A small circuit board housing a camera module based on the Sony IMX219 image sensor. The sensor board is connected to the Raspberry motherboard with flexible cable and data is transmitted over a camera serial interface (CSI) bus. The native resolution of the sensor is 3280×2464 pixels. Video frame-rate with 1240x1080 resolution is 30 frames per second (fps) and with 640x480 it is possible to achieve 90 fps. The optical system is fixed focus with manual focus adjustment and offers 62.2° horizontal and 48.8° vertical field of view [71]. Camera module is also shown in figure 2.2.

- **Intel© Neural Compute Stick 2 (NCS2)**

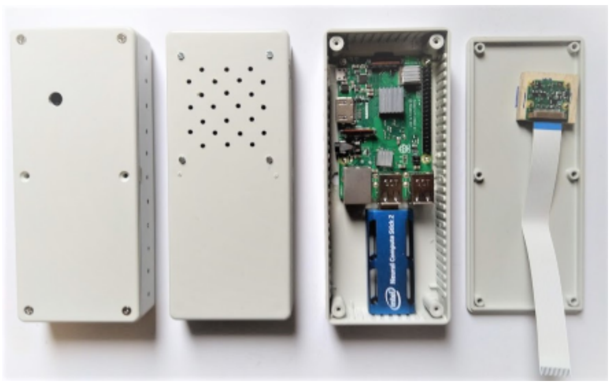
  An external accelerator for neural networks. This is a passively cooled device that is attached to the host computing system via the USB interface. Both power and data are transmitted over the USB. The heart of the NCS2 is Movidius® Myriad X VPU MA2485 chip that contains 16 processing cores dedicated to running neural networks [12]. The chip contains also additional functionalities, such as ISP, but those are not available for the users of NCS [72].

All hardware is inserted into a plastic enclosure of the size $165\,\text{mm} \times 70\,\text{mm} \times 28\,\text{mm}$ that was customized to have a camera mount and ventilation. The weight of the unit is about 175 g, thus making installation at location easy with for example adhesive tape or magnets. Before the motherboard is placed into the enclosure, Raspberry PI's main chips are equipped with aluminum heat sinks to improve cooling. No active cooling system, like motorized fans, is used in order to ensure silent operation in the museum environment. At installation location a system is powered from the wall electricity with separate USB power adapter, but other USB compatible power sources are possible. For a short term installation the use of battery power is also feasible. Figure 4.1a shows the hardware in its enclosure.

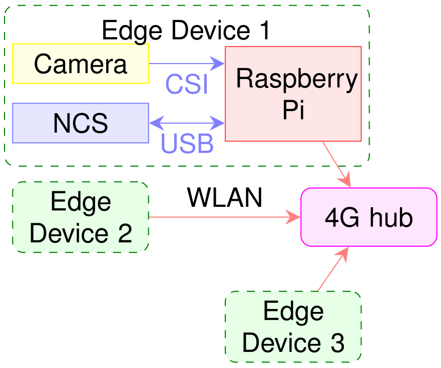
A separate WLAN hub with a 4G cellular modem is placed at the installation location. This provides a local wireless network (WLAN) and internet connectivity. All devices are connecting to this WLAN and will send data to the cloud server via a shared modem. It is also possible to utilize existing wireless network available at the premises if such is available. However, a separate modem makes installation independent of local network access control and allows easy and fast installation of the system at any location. Main system components are shown in figure 4.1b

4.3 Software design

Software running on the edge device can be divided into two main parts, a management layer and an application layer. The management layer is responsible for monitoring and controlling device state and providing mechanisms for updating device software remotely. The application layer will run the software required by the actual use case of the



(a) Hardware with an enclosure.



(b) System Components

Figure 4.1. *Hardware and system*

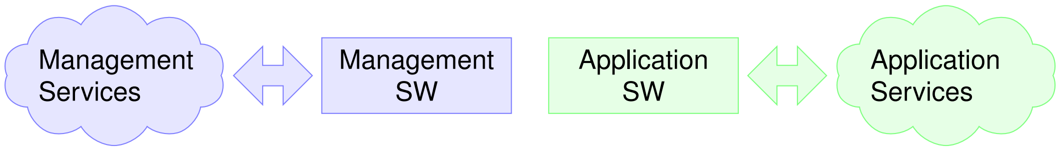


Figure 4.2. *Two part SW architecture for the edge device.*

device. Although these functionalities can be implemented as software components and processes running under the same operating system, it is advisable to separate them into their own isolated operating system instances. In this manner, for example erroneous application layer software update has a lower risk to break management functionality and prevent remotely fixing problematic code. Also as requirements for the management layer are rather independent of the application there exist generic solutions that can be used for this. In this respect there exists several also several commercial SW frameworks for IoT device management [73][74][75].

High level architecture of the software for the device is illustrated in figure 4.2. Application and management layers are separated into their own OS instances and are also communicating with respective cloud services independently. In this set-up application-related cloud functionality, its usage rights, administration and development are also completely separated from the device management cloud. In the following sections both SW layer for current implementation are described more details.

Finally, it should be noted that it is also possible to deploy device SW without the management layer. This is suitable for small scale testing with one or very few devices in the environment where local physical access to devices is easy. For large production level deployment with large device fleet the management SW layer is highly recommended.

4.3.1 Device management solution

As stated in the previous section it is desired to have SW architecture where the application layer lives in an isolated environment with its own OS. For a full device management solution also other requirements need to be full-filled. Some of these need also dedicate management cloud services. For a desired solution following requirements are defined:

1. **Isolated application environment.**

   An architecture that enables running edge application software in its own OS, isolated from the management SW running on the device.

2. **Well defined application environment.**

   Mechanisms that support easy definition and replication of the whole application environment, including OS, all SW libraries, and the application itself.

3. **Provisioning support.**

   Provisioning is a process where a device is assigned to be a part of the system. Only devices with proper identification and credentials should be allowed to register to be a part of the system. In order to establish secure communication between the device and a network, the network needs to be able to authenticate device in a reliable and secure manner.

4. **Remote SW updates.**

   The primary requirement is to update the application or part of it remotely. The second level requirement is to be able to update also application layer operating system.

5. **Remote terminal access.**

   A Secure Shell (SSH) connection to application is needed, for application control and debugging purposes. Connection should be possible thru-out various network configurations and firewalls.

Solutions

Container technologies [76] offer solutions for isolation of the application environments and do give also a mechanism to define the whole application layer stack, including OS, all libraries, and the application content. Containers belong to the family of *virtualization* technologies. There are two main flavors of these techniques, *virtual machines* that are SW layers emulating physical computing environment and *containers* that do rely on the services provided by an underlying host operating system.

Containers are considered to be more lightweight than full virtual machines. As a container does not bring a full operating system but relays on the functionality offered by the host operating system the size and start-up time for a containerized application could be smaller than when using virtual machines. Containers are widely adopted when implementing web services, but they can be also utilized in local computing environments [77][78].

For current design container solution based on the Docker [76] platform is chosen. This fulfills requirement 1 by design, when we place application with its OS inside the container

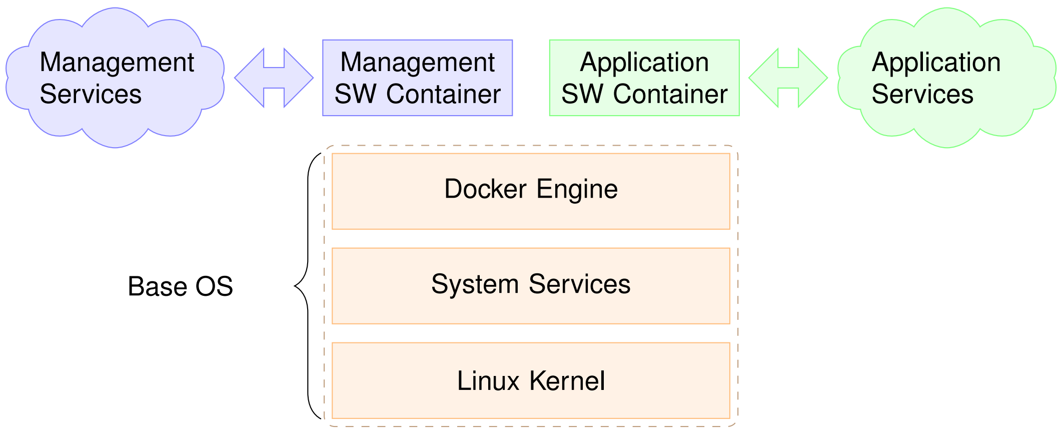


***Figure 4.3.** Container managed Edge SW solution*

and management SW solution to a parallel container as shown in figure 4.3. Requirement 2 is also covered as Docker offers a convenient way to define the building process for a container with a single *dockerfile*.

It is possible to develop a proprietary solution that covers the rest of the requirements, but it is beyond the scope of this thesis. Instead a fleet management solution provided by Balena [75] is taken into use. Balena offering consist of similar Docker container compatible architecture that can be deployed on the device as defined earlier. In addition it offers two possibilities to implement cloud functionality that would cover the rest of the requirements. Commercial solution offers a full web-based device management dashboard hosted on their serves and at the moment of writing it is free for fleets having at most 10 devices. There exist also OpenBalena, a free solution for an unlimited number of devices. OpenBalena offers most of the required features with a command-line interface instead of the graphical UI and it needs to be hosted on the own server. For initial experiments the commercial Balena platform is chosen.

4.3.2 Application software

The key requirements for the application SW are derived from the use case. Also device state monitoring capability and features supporting development work are needed. Main requirements impacting the design are listed below:

1. **Person detection.**

   Detect person from the camera image. Calculate a feature vector for each detected person. Send feature vector to the cloud server.

2. **Status message.**

   Deliver heartbeat status message to the server at predefined intervals. Include detailed status information such as CPU utilization, amount of the available memory, temperature of the system. At device start and per request deliver a thumbnail image from the camera for testing

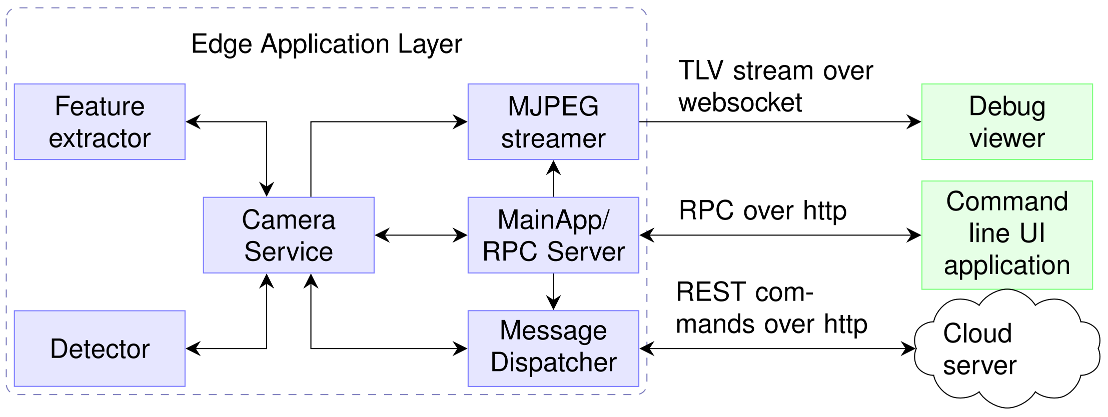


Figure 4.4. *Application SW Architecture*

3. **Control over the network.**

   Offer remote interface over a network for basic control of application features, such as start, stop, and sending additional debug information to a server.

4. **Development and testing support.**

   Enable local image and detection result view for testing and debugging. Enable video feed with detection results to a test server.

Desing

High-level SW architecture for the solution is described in figure 4.4. The design is modular consisting of 7 primary components in the device SW side. There are also 2 additional components "Debug Viewer" and "Command line UI application" that are intended to support the development and testing of network-related functionalities. The whole solution is implemented with Python programming language.

Solution is launched by starting the entry point of the application "MainApp". This initializes the system and starts the camera and network-related functionalities. When the initialization is complete, the system starts sending a regular status message "Heartbeat", to the defined network server addresses. Application start also a remote procedure call (RPC) server process that can receive commands over the Http connection to control functionalities of the application. For testing and development needs an additional command-line application is provided for sending RPC messages.

The main functional component of the solution is a "Camera Service" that controls the camera HW and triggers the object detection thread for incoming image data. Object detection is performed by executing an SSD neural network on the compute stick. OpenCV DNN API is utilized for controlling the execution of the network. Prior to sending the image data to the NCS, it is scaled and normalized in the CPU. After detection is done, portions of the image that contain detected objects are fed into a re-identification neural network, running either on CPU or NCS, for generating the corresponding feature vector for each object. The object detection data, coordinates, and feature vectors provided by

Camera Service are then delivered to "Message Dispatcher" threads that do send data to the cloud server via representational state transfer (REST) application programming interface (API) for storage and further analysis.

"MJPEG streamer" is an additional functionality intended to work mainly as a development aid. It provides full image data from the camera as a stream of JPEG-compressed images. Images and the detection data are combined as a joint data stream with simple proprietary time-length-value (TLV) protocol. This stream is then transmitted over a WebSocket protocol to a receiving server. "A Debug viewer" is a simple web server, that can handle WebSocket protocol, decode the TLV data and show them in simple graphical UI. This enables a real-time view of the received data. The same server can also receive and visualize detection data and heartbeat REST API messages. It should be noted that the MJPEG streamer can be removed from the SW package for the devices that are deployed for production use. This would ensure that privacy-sensitive image data cannot be received from the installation location.

Execution of neural networks

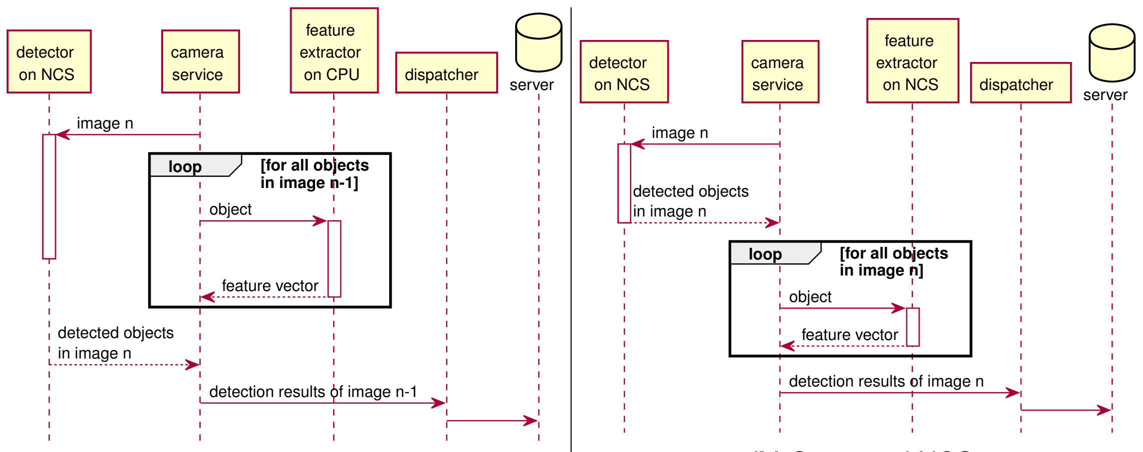
During the work two alternative architectures for running the neural networks were tested. The first approach is to execute the object detection network on the neural compute stick and feature extraction network on the CPU. In this solution both networks are running in parallel, object detector finding objects from the incoming frame number n while feature extraction is calculating feature vectors for objects in previous frame $n-1$. The sequence diagram of this is shown in figure 4.5a.

In the second approach both object detection and feature extraction are running on the compute stick. In this solution the processing is sequential, the first object is detected from incoming frame n and immediately after that features are generated for found objects. The corresponding sequence diagram for this is shown in figure 4.5b.

While the first approach allows parallelism and utilization of all available computing resources in the system, a potential drawback is the high utilization of the CPU that might cause thermal issues. Parallel approach is also having a latency of 1 frame compared to the sequential architecture. The final performance of the system depends on the running time of the networks on each computing platform. Also, the number of the object detected in the frame is impacting to the performance. The comparison of the performance is covered in Chapter 5.

4.4 Neural networks used in the implementation

The solution uses a neural network for both object detection and feature extraction. Due to the privacy issues it was not possible to collect training data from the actual installation location. Instead networks were trained with publicly available databases. The actual training process is not in the scope of this work. Several different variants of the networks



***(a)** Parallel CPU and NCS*

***(b)** Sequental NCS*

***Figure 4.5.** Processing partitioning options of feature extraction and object detection*

were tested, especially to understand the processing speed differences of different design choices. Performance in this regard is analyzed in chapter 5.

The design choices of the used networks are briefly discussed below. After that we cover the process of deploying the models in the device.

Object detection

The system needs to be able to find bounding boxes associated with all persons in the camera view with a reasonable frame-rate. In order to limit the design options, the baseline requirement for object detector speed is set to be above 1 fps. Based on earlier analysis [26] this implicates that for object detection only networks that would be running on NCS accelerator are considered.

For object detection, there are two commonly used neural network solutions. The Single-Stage Detection (SSD) [25] and the Regions-CNN (RCNN) [79]. These two structures represent two widely architecture families. The two-stage R-CNN is traditionally perceived as more accurate, especially with small targets. On the other hand, the single-stage SSD type networks are simpler, reach faster execution time, and still reach a reasonable accuracy when the targets are not exceptionally small. When choosing which object detection architecture to use, initial tests for comparing SSD and Faster-RCNN networks on the target architecture was done. The Faster-RCNN structures tested did not reach above the set minimum speed target, thus SSD framework was chosen as a design baseline.

SSD can use different backbone networks for feature extraction, and the choice of backbone is an important design parameter balancing the trade-off between accuracy and speed. For this work 3 different SSD architecture variants are chosen: SSD with MobileNetV1 [49] and MobileNetV2 backbones and lighter version of SSD, i.e., SSDLite, with MobileNetV2 backbone [80].

In addition to the choice of the backbone, a network can be configured with different

Table 4.1. *Different Object detection network configurations and their mean average precision (mAP) [23] in validation set when training was finished.*

		Mobilenet Lite	Mobilenet V1	Mobilenet V2
Resolution	Depth	mAP	mAP	mAP
100x100	0.25	0.44	0.40	0.49
	0.50	0.43	0.47	0.50
	0.75	0.49	0.47	0.47
	1.00	0.54	0.49	0.54
200x200	0.25	0.53	0.46	0.53
	0.50	0.52	0.51	0.51
	0.75	0.56	0.54	0.58
	1.00	0.62	0.60	0.61
300x300	0.25	0.44	0.49	0.54
	0.50	0.56	0.53	0.56
	0.75	0.60	0.54	0.58
	1.00	0.65	0.62	0.65

parameters that also impact the speed and accuracy trade-off. SDDs with 3 input sizes of 300x300, 200x200 and 100x100 are constructed. Further variation is done with 4 different values for depth multiplier parameter (α = 1, 0.75, 0.5 and 0.25). This impacts the number of channels used in layers of SSD. Network configuration with the mean average precision (mAP) are listed in table 4.4.

As a starting point, initial models were obtained from the TensorFlow model zoo [81]. These models had a pre-trained weights trained on MS COCO dataset [82]. Fine-training of networks was done with person class images from OpenImages dataset [83]. The training and validation set was downloaded 76k(76561) images from the *train* and 1.8k (1874) images from the *validation* repository of OpenImages dataset person class, filtering occluded, truncated, depicted, and images taken from inside. The aim is to create a representative network capable of detecting persons in a wide range of different scenarios.

Re-identification

As target is to associate detected people across multiple cameras and moments time, a *person re-identification* neural network is chosen as a means to provide features that are sent to the server. For the use case a network should be able to create features vectors where the distances between observations of the same person are small, while the distance between the vectors representing persons with different identities should be large.

Speed requirements are dictated by the assumed number of people simultaneously on the camera view and system frame-rate. For parallel implementation on the CPU the

Table 4.2. *Different re-identification network configurations and their mean average precision (mAP) in validation set when training was finished.*

		Mobilenet V2
Resolution	Depth	mAP
96x96	0.35	0.49
	0.50	0.59
	0.75	0.64
	1.00	0.66
	1.40	0.67
128x128	0.35	0.58
	0.50	0.65
	0.75	0.71
	1.00	0.70
	1.40	0.70
160x160	0.35	0.62
	0.50	0.67
	0.75	0.72
	1.00	0.72
	1.40	0.74

calculation of all re-identification feature vectors should happen inside the same time envelope as the object detection. For example 4 persons in a view at frame-rate of 3 fps, would result in a requirement to create 12 vectors per second, corresponding 83 ms processing time. In sequential NCS accelerated architecture the speed requirement is much faster. Referring to the previous example, the 3 fps system allows about 333 ms of total execution time. If 300 ms is consumed by object detection, remaining 33 ms are left to process 4 detections. This gives about 8.3 ms speed target for accelerated detection.

The networks used in this implementation are adopted from the person re-identification algorithm [59]. Overview of the method is presented in chapter 3, section 3.3.3.

MobileNetV2 [80] backbone is chosen for the re-identification network, as its computational complexity is suitable for real-time implementation in the edge device. Like in the case of object detection, several variants of the network with different configuration parameter options are chosen. Networks with 4 different input sizes (160x160, 128x128, 96x96) and 5 different depth multipliers (1.4, 1.0, 0.75, 0.5, 0.35) are constructed. Network configurations with the mean average precision (mAP) are listed in table 4.4.

The backbone model is initialized with pre-trained weights on the ImageNet [84] dataset. The actual training is done on the Market-1501 [85] dataset. Several strategies are implemented to prevent the overfitting issue, namely, random erasing data augmentation [86], label-smoothing regularization [87] and ℓ_2 regularization.

Model deployment

As mentioned in the previous section the actual execution of the networks for inference is done with OpenCV DNN and OpenVino frameworks. Before networks can be used with these SW libraries they need to be prepared and converted to a proper format. The first step is a "freezing" of the model. With this process all training related network layers are removed or their parameters are fixed. An example of such layer is the batch normalization that during the training time calculates mean and variance of the input batch, but in the frozen in inference model reduces only as normalization operation with fixed parameters. Tensor flow framework provides a convenience python script for model freezing. For execution on CPU the frozen model can be used directly without any further conversion steps.

For the execution of the model on NCS additional steps are needed. In SW stack the NCS is represented as an OpenVino inference engine that will execute networks provided in IR (Intermediate Representation) - format. The frozen TensorFlow model need to be converted to this format with the *model optimizer* tool. The tool is a python script that reads the input network and maps its layer and operations to the corresponding ones in IR that are supported by the inference engine. If the input network is consisting of common building blocks this mapping is a straightforward process. This is a case with re-identification networks used in this work. Object detection networks based on Tensorflow Object Detection API, do require some additional information for conversion. This is provided for the conversion script with a separate file that needs to correctly match to the version of the TensorFlow and Object detection API [67].

Also in general it needs to be ensured that version of the SW libraries are aligned. Specifically versions of Tensorflow, Tensorflow Object detection API, OpenVino on the device, and OpenVino on the system were conversion are done, need to be compatible. In this project a separate verified development environment for model conversions was composed as a Docker container. This ensures an easy set-up of the system to a new development computer with a single command and guarantees that all required libraries are in place.

5 DEVICE PERFORMANCE

Different aspects of system performance are analyzed in this chapter. In order to provide good quality data the camera device needs to have high enough frame-rate so that there is enough data for trajectory estimation and clustering. On the other hand the system power consumption and heat generation should be in acceptable limits to ensure consistent and interrupted operation. In the following sections analysis reports on frame-rate, processor load, and thermal behavior, with respect to different partitioning of various neural networks between CPU and NCS, are provided.

5.1 Execution time of neural networks

The upper limits of the system speed are investigated by doing measurements of the neural network execution times on the device. The same "Feature Extractor" and "Detector" SW components from actual application solution, illustrated in figure 4.4, are used for running networks also in the measurements. Time consumed in image pre-processing (scaling, cropping, normalization) is part of the execution. Also network outputs are parsed and stored to the data structures during the measurement. However, in the test set-up the other processing load of the system is minimized. The camera system is disabled and input data for the neural networks is fed from the memory, into which images are loaded prior to the start of the test run. Also the network output is not delivered to servers outside the device, but is discarded after the output data structure is formed.

Execution time is evaluated in two operating points of the CPU: 1400 MHz "performance" and 600 MHz "power-save" mode. All results are an average of 10 consecutive executions of the network. Prior to starting averaging, 5 warm-up runs of the network are done to ensure the exclusion of any abnormal time measurement. Typically the first run of the network, especially on NCS, will take significantly more time as the network model is loaded and prepared for execution. Measurements were done in the room temperature of about 20 °C. Measurement sequences were kept short and had one minute delay in-between, in order to guarantee that CPU temperature of the Raspberry would remain under 70 °C preventing any possibility for automatic performance throttling altering the clock frequency.

Tests are performed separately for object detection and re-identification. After execution time measurements, graphs illustrating trade-offs between speed and accuracy are presented. The joint performance of both networks in various combinations is also analyzed.

Table 5.1. *Object detector frame-rate in frames per second (fps) of different networks configuration running on NCS, measured with 2 different CPU frequencies.*

		Mobilenet Lite		Mobilenet V1		Mobilenet V2	
		fps @		fps @		fps @	
Resolution	Depth	600 MHz	1400 MHz	600 MHz	1400 MHz	600 MHz	1400 MHz
100x100	0.25	29.22	48.37	32.74	54.78	29.43	46.60
	0.50	27.29	40.53	30.72	49.54	28.27	43.14
	0.75	24.91	35.38	29.26	45.09	26.38	38.29
	1.00	23.51	32.74	27.04	40.49	24.64	35.55
200x200	0.25	19.45	27.88	23.49	35.69	19.66	27.95
	0.50	18.14	24.96	21.70	32.35	18.84	26.06
	0.75	16.25	21.53	19.76	27.83	17.18	22.85
	1.00	15.48	20.12	17.86	24.70	16.20	21.45
300x300	0.25	13.40	17.77	16.60	23.38	13.57	17.94
	0.50	12.61	16.31	15.03	20.65	12.97	16.90
	0.75	11.11	13.69	13.57	17.77	11.50	14.36
	1.00	10.30	12.60	12.15	15.54	10.73	13.30

5.1.1 Object detection

In all tests input image size for the detector was 640×480 that was scaled during the pre-processing to network input size as a part the each detector execution. 36 different network configurations were tested with two CPU frequencies, resulting in 72 individual measurements. For each configuration a detector frame-rate, in frames per second (fps), is measured. The measurement results are presented in table 5.1.1.

Fastest processing rate, with CPU clock frequency of 1400 MHz is 54.78 fps, corresponding to 18 ms execution time. It is achieved with MobilNet V1 backbone, a 100×100 resolution and a depth multiplier value of 0.25. The slowest configuration is with MobileNet Lite backbone, a 300×300 input resolution and 1.00 depth multiplier, at 600 MHz, resulting 10.30 fps and 97 ms processing time.

Within fixed CPU frequency the fps ranges are 10.32 to 32.74 and 12.60 to 54.78, for 600 MHz and 1400 MHz, respectively. The relative difference between a particular configuration running at two different frequencies is larger with smaller resolution and lower depth multiplier values. This is expected, as the running time on the accelerator is shorter for a lower complexity network, the relative contribution of CPU side pre-processing and USB transfer is more significant to the total execution time.

It is observed that the choice of input resolution and depth multiplier are giving a large set of variations in speed. Differences between the chosen backbone architectures are smaller, but it is noted that networks with MobileNet V1 backbone are fastest and those with MobileNet Lite are slowest. Without further insight on the details of the internal processing architecture in the Myriax X chip the root cause for this cannot be confirmed. Speculatively results can be seen as an indication that optimization done in Mobilenet

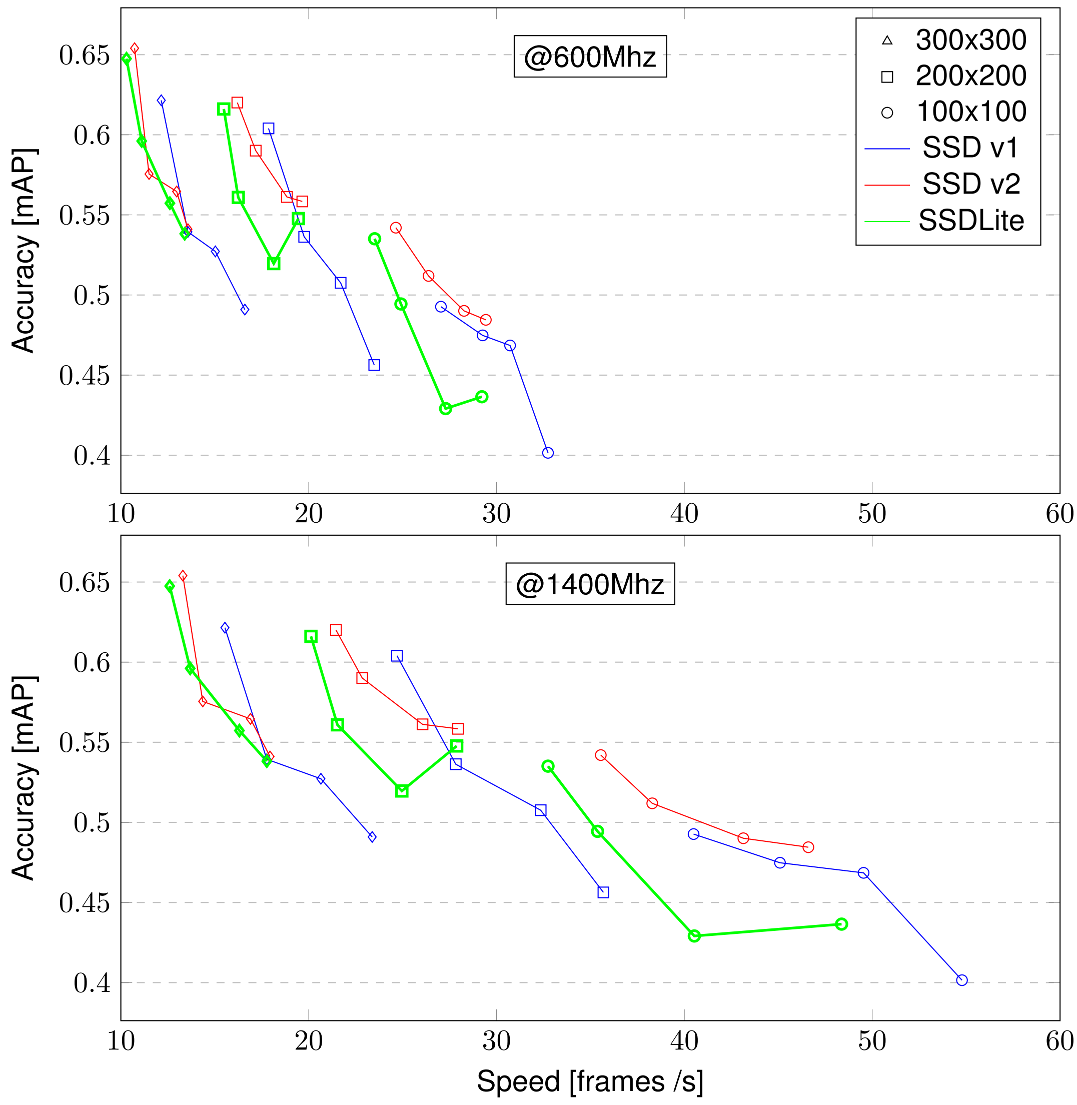


Figure 5.1. *Comparison of different SSD implementations running on NCS. Each solid line shows the effect of different of depth multiplier values (1.0, 0.75, 0.5, 0.25), while input size and network architecture do remain constant. The impact of CPU clock frequency can be seen when comparing upper and lower graphs.*

Lite architecture does not best possible fit to the processing architecture inside the accelerator.

Figure 5.1 shows the trade-offs between processing rate and accuracy when using different network configurations. In general it can be observed that smaller input resolution and smaller depth multiplier values, while offering faster execution, do result to lower detection accuracy. It can be also noticed from the graphs with some lower values of the depth multiplier this behavior is not consistent and in two cases depth multiplier values of 0.25 are resulting to higher precision than the same resolution configuration with 0.5 depth value. The probable explanation for this is the stochastic nature of the training process, with a larger number of the training steps it is likely that results would be more consistent. It can be observed that even Mobilenet V1 is fast the accuracy is lower compared to the other two networks. It can be stated that configurations based on the Mobilenet V2 backbone

Table 5.2. *Execution time of different re-identification network configurations, running on CPU or NCS with two possible CPU clock frequency operating points.*

		Execution time (ms) on CPU		Execution time (ms) on NCS	
Resolution	Depth	600 MHz	1400 MHz	600 MHz	1400 MHz
96x96	0.35	47	24	15	13
	0.50	62	32	16	14
	0.75	101	52	19	17
	1.00	131	67	21	19
	1.40	222	117	27	24
128x128	0.35	63	33	20	17
	0.50	83	44	22	19
	0.75	136	72	26	23
	1.00	173	92	28	25
	1.40	282	151	34	31
160x160	0.35	90	47	27	23
	0.50	119	63	29	24
	0.75	197	104	35	30
	1.00	250	130	37	32
	1.40	406	213	45	40

are presenting the best trade-off.

5.1.2 Re-identification

The speed of the object re-identification network is tested with an input image of the size 640×480. This is to emulate the worst-case scenario for the CPU side pre-possessing step where the detected person would occupy the whole camera view. In typical use case this is somewhat unlikely, but still possible scenario. 12 different network configurations are tested, running on either CPU or Neural Compute Stick, with two CPU operating frequencies. The total number of measurements is 48. Results are shown in the table 5.1.2.

The variation in the execution speed is relatively high. The slowest configuration with 406 ms processing time has 160×160 input resolution, 1.40 depth multiplier and it is running on CPU with 600 MHz. The fastest 13 ms time is achieved by running network of 96×96 input resolution and 0.35 depth multiplier on NCS while CPU frequency is 1400 MHz. Thus the fastest measurement is 31 times faster than the slowest.

Similar correlations between resolution, depth multiplier, and execution time are observed as with the object detection networks. In general, lower complexity networks are faster. When the network is executed in the CPU with the higher clock frequency it is on average about 1.9 times faster than with lower frequency operating point. In NCS execution the impact of a CPU frequency to execution time smaller. It can be observed that when running a network in NCS the execution time difference within the same resolution is

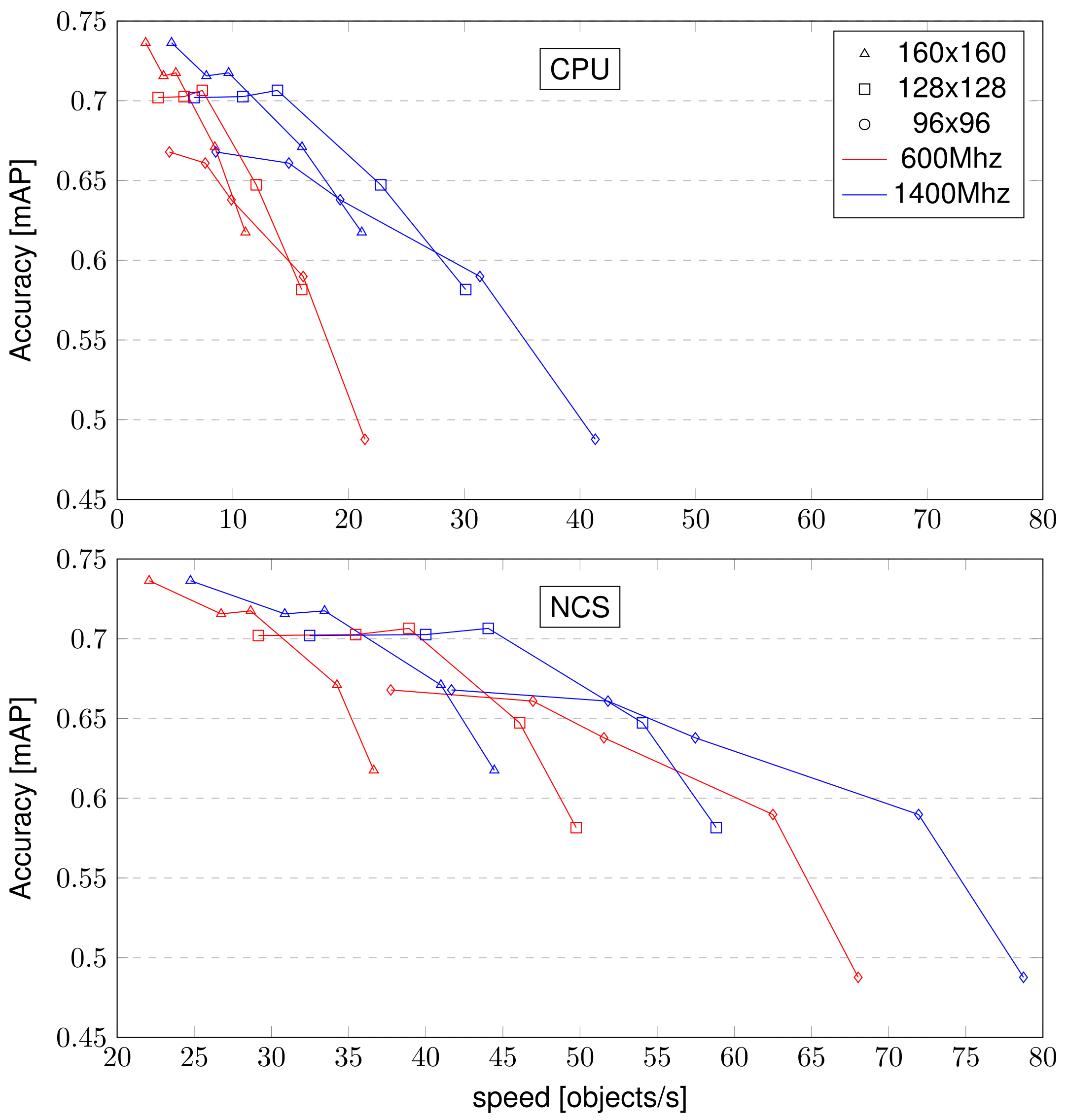


Figure 5.2. *Comparison between the implementations of the re-identification. Each solid line shows how different depth multipliers (1.4, 1.0, 0.75, 0.5, 0.35) do impact to the performance, while resolution remains fixed. Upper graph shows performance on CPU and lower on NCS.*

near-constant. At 96 × 96 resolution this difference is about 2.1 ms, at 128 × 128 it is 3.2 ms and at at 160 × 160 it is 4.9 ms. This can be explained by the fact that a clock frequency has a direct impact on the resolution-dependent pre-processing and USB data transfer while actual network running time in NCS is independent of the host CPU speed.

Implementation options are giving a wide range of possible trade-offs between speed and accuracy. This is illustrated figure 5.2. Both depth multiplier and input resolution have a significant impact on both variables. Graphs indicate that if there is a need to improve the speed, at first it would be advisable to choose a higher resolution network with possible lower depth multiplayer value. Moving the lower resolution can happen after depth value is below 0.75. It should be also noted that even if the speed with 96 × 96 resolution with 0.35 depth value can be high, the performance drop is also significant. Before production-

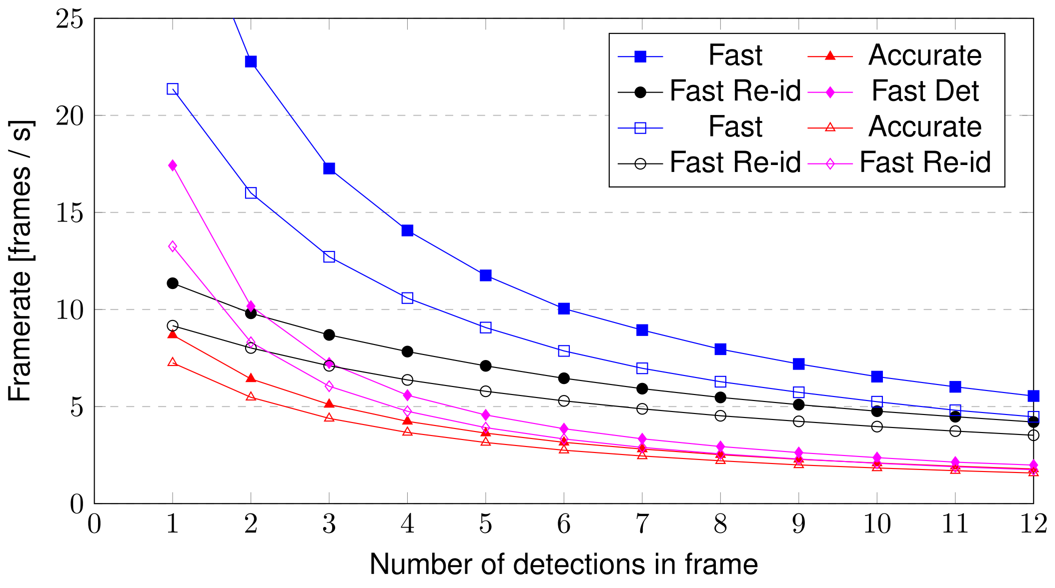


Figure 5.3. *Examples of the total system performance with different implementations of object detection and re-identification networks. All networks are running on NCS. Solid marker indicates 1400Mhz CPU frequency and open marker 600Mhz.*

level deployment a careful use case related performance testing is recommended to see the network of this low accuracy is sufficient for the use case.

5.1.3 Combined execution of the networks

The performance of the system depends on the combined execution time of object detection and re-identification. As seen in previous sections, choices with hyper-parameters are giving different trade-offs between accuracy and speed. The final choice is dictated by the use case. For example, when tracking people over a large number of cameras, high accuracy re-identification network is needed. On the other hand, in the set-up where there is a need to perform a single-camera tracking-by-detection scheme, a simpler feature generation network is might be sufficient, but a higher frame-rate for object detection is preferred. Likewise, for example, the object detection accuracy requirements are impacted by the assumed distance from person to the camera. At small distances, where object occupies a large area from the image, lower input resolution might be sufficient.

Figure 5.3 illustrates estimated joint performance in conditions in which all networks are running on NCS. The graph shows various example configurations: *"Accurate"*, where the SSD and re-identification networks with the highest mAP value are running together, *"Fast"*, in which fastest networks are executed and finally two trade-off configurations combining fast detector and with accurate re-identification and vice versa. It can be seen that the object count dominates the total processing time and restricts the available maximum frame-rate. Between the example combinations shown on the graph, there are more options, especially when the total system-frame target is below 10 fps.

When choosing the optimal partitioning between CPU and NCS, measurements indicate that the option where all networks are running on NCS would offer more better-performing combinations. Only when assumed object number in the camera view is small, so that re-identification network can process all detected objects parallel to object detector, the theoretical total system frame-rate with CPU based re-identification can be higher compared to running all networks on NCS. For the networks consider in this study the practical person count for a such scenario would be limited to 1 or 2 person simultaneously present in the view.

5.2 System level performance

Performance testing is done also in the whole system level to gain an understanding of how the data flow from the camera and the rest of the SW functionality would impact the total frame-rate. The aim is to test total device performance with varying input image conditions.

In system-level testing a real camera signal is needed with repeatable input. If a suitable HW is available camera data can be emulated by feeding stored image data to the CSI interface. In this project such a HW was not at hand. An alternative is to create a physical set-up, in which illumination and scene content can be fully controlled and repetition is possible. As the testing is primarily focused on operation speed the test set-up can by a computer display that produces input to the camera. The refresh rate of the display is set to be high enough compared to the exposure time of the camera. Also stationary images are used to prevent possible movement artifacts due to the inference of display refresh and camera image capture rate. For this work, a simple test images with different number of persons are used. Test scenes are constructed in such manner that all presented objects were always detected by all used object detection networks. It should be noted that this set-up cannot be used to measure the accuracy of actual object detection or feature extraction. Test set-up is shown in figure 5.4

5.2.1 Measurements

For full system-level performance measurements a subset of the combinations of the networks, considered relevant for the use case, are chosen. For re-identification the most accurate network with 160×160 resolution and 1.4 depth is selected. In addition to that network having 128×128 resolution and 0.75 depth, seen as a good trade-off between accuracy and performance, is included into the test set. These networks are then combined to a full solution with 3 variants of object detection, all having depth value of 1.0, but covering all 3 resolution: 300×300, 200×200 and 100×100.

Measurements are done at the 1400 MHz CPU clock frequency. Both parallel and sequential SW architectures are tested to see the performance difference when re-identification is running on Neural Compute Stick or CPU. For each configuration a total system frame-rate and CPU load percentage with 5 different image inputs are measured. Each input

Table 5.3. *Total system frame-rate for 6 different combinations of object detection and re-identification measured at 1400 MHz operating point for different person counts (1 to 5) in the view. In all cases object detector is running on the neural compute stick. For re-identification two cases are compared: NCS denoting execution on the accelerator and CPU on the main processor.*

Re-identification	Object detection	NCS			CPU		
(resolution, depth)	(resolution, depth)	persons	fps	load %	persons	fps	load %
160x160, 1.4	300x300, 1.0	1	6.2	22	1	1.9	79
		2	5.0	20	2	1.2	83
		3	4.2	19	3	0.8	86
		4	3.5	17	4	0.6	87
		5	3.0	16	5	0.5	87
	200x200, 1.0	1	6.5	23	1	1.9	80
		2	5.3	21	2	1.1	84
		3	4.3	19	3	0.8	85
		4	3.6	17	4	0.6	87
		5	3.1	15	5	0.5	87
	100x100, 1.0	1	7.2	24	1	2.0	80
		2	5.4	23	2	1.1	84
		3	4.4	18	3	0.7	85
		4	3.7	17	4	0.6	87
		5	3.2	16	5	0.5	87
128x128, 0.75	300x300, 1.0	1	6.5	23	1	4.1	67
		2	5.8	21	2	2.6	75
		3	4.8	20	3	1.9	78
		4	4.5	20	4	1.5	80
		5	4.2	19	5	1.2	83
	200x200, 1.0	1	7.3	24	1	4.0	67
		2	6.3	23	2	2.6	76
		3	5.6	23	3	1.9	79
		4	4.9	21	4	1.5	81
		5	4.3	20	5	1.2	83
	100x100, 1.0	1	8.2	25	1	4.2	67
		2	6.7	23	2	2.6	73
		3	5.6	21	3	1.9	79
		4	4.8	19	4	1.5	82
		5	4.4	20	5	1.2	83

image has different number of persons, from 1 to 5, in order to see the impact of person count to the performance. The total number of measured test cases is 60. Results are listed in table 5.2.1. All numbers are the average of individual measurements over 30 s period. Before each measurement round, a stable state is ensured by running 30 s stabilization period during which no measurements were recorded.

In general it is observed that person count is the most dominating factor for the overall performance. The results show also that when re-identification is running on NCS the

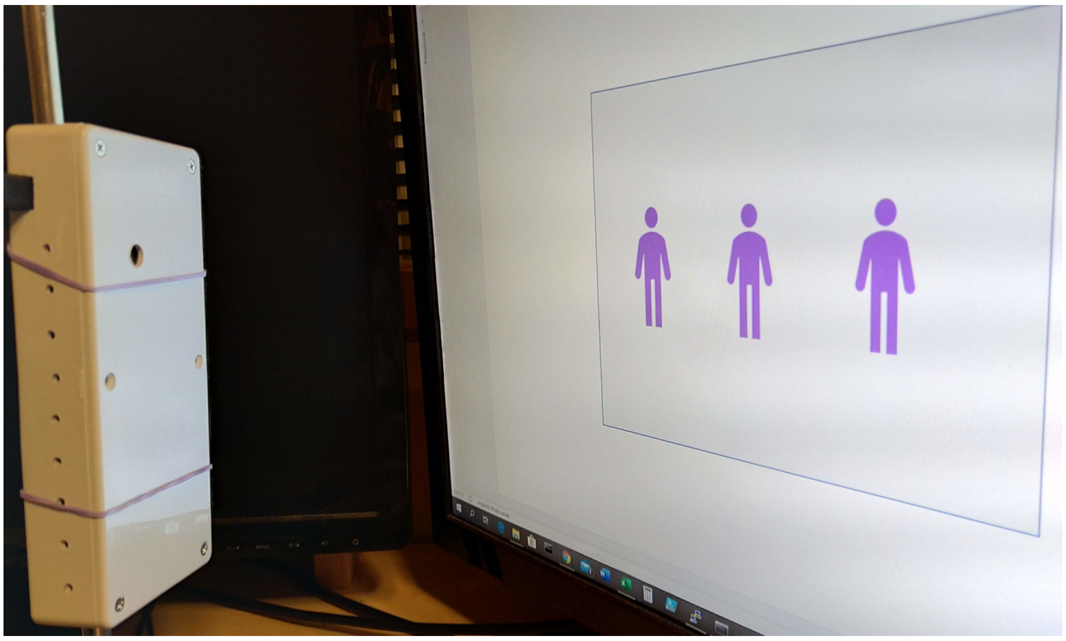


Figure 5.4. *Test set-up for system level measurements.*

frame-rate is in higher than with CPU execution. The difference is emphasized when the object count increases, with 5 people in the view and using more complex 160 × 160 resolution re-identification the difference between full NCS implementation is about 6 times faster compared to CPU. Only in cases were simpler re-identification network is used and there is a single person in the view CPU execution can reach to comparable level.

The choice of the networks have a larger relative impact in CPU+NCS implementation than with NCS. This is expected as in CPU implementation the re-identification will start to dominate the total execution time faster than in NCS implementation.

Processing partitioning has a significant impact to the CPU load. Executing the re-identification in CPU results in most cases over 80% load. As the measurement period is purposefully set to 30 s, to prevent system overheating, the measured temperature did not exceed 80 °C. However, earlier studies [26] from the similar system indicate that maintaining such high load level more than about 60 s would cause the system to start reducing the clock frequency in order to protect the processor from overheating. Due to this, over a longer execution period frame-rate would be reduced. When running re-identification in NCS the CPU load is significantly smaller and system temperature would stay below 70 °C even with continuous execution.

CPU load increase as object count increases with CPU re-identification. With NCS implementation the effect is the opposite. This is due to the reason that as the system frame-rate is getting smaller also the total amount of the CPU operations related to handling of detected objects is reduced. All-though both implementation options are impacted by this phenomenon it is completely hidden inside the high load caused by re-identification calculation in CPU implementation.

The measured system performance is lower than the theoretical limits presented in 5.3. This understandable as the full system needs to perform other tasks such as processing and delivering detection results to the server. Also receiving of the data from the real camera, can reduce the frame-rate compared to the test case in which images are read from the memory. However, as the CPU load in full NCS implementation is relatively low it is possible that synchronization between threads in the application is causing additional latency. With more detailed performance profiling of the software it could be possible to find bottlenecks and improve the frame-rate. This is a topic for further study.

5.3 Discussion

One of the primary research questions for this thesis was to evaluate if the combination of object detection and feature extraction could be running on the edge compute platform. Measurements reported in this chapter do show that it is possible to perform such computation on the hardware that is implemented for this study. It can be also seen that all architectural options may not be feasible to support use case requirements.

By analyzing the use case, high-level requirements for speed can be estimated. As the aim is to track moving pedestrians we want at the minimum 2, but preferably much more observations from the person walking in the camera view. Time t, how long a person will be observable when walking across the camera view, perpendicular to the image plane, can be calculated with the following formula:

$$t = \frac{d \tan \frac{1}{2}\alpha}{v} \tag{5.1}$$

where d is the distance to the camera, α is the field of view and v is the objects speed. Lower limit for system frame-rate can be approximated in the following manner: If assuming person walking with a speed of 5 km h^{-1}, corresponding 1.4 m s^{-1}, walking across the camera view from 3 m distance to the camera, the person will stay in the view about 1.3 s. In order to capture the minimum 2 observations the camera frame-rate needs to be above 1.5 fps.

Performance requirement depends also on the assumed number of persons that are present in the view at the same time. Target for this depends on the camera placement, field of the view and assumptions about how persons are moving in the view. If we are assuming 5 person in the view simultaneously the above estimated 1.5 fps level can be exceeded with all full NCS based solutions. On the other hand CPU re-identification cannot reach this minimum target with any of the measured system configuration. If the person count requirement is lower, there are CPU based configuration that will reach the required level. However, if the person flow is continuous, even with a small number of people simultaneously in the view, the CPU based solution needs to maintain a high processing load. This would eventually result to high system temperature and lower clock frequency resulting in lower frame-rate. In extreme cases, for example with higher room temperature, even the system thermal shutdown could be possible.

Although simple analysis of minimum frame rate with equation 5.1 is setting a target as 1.5 fps, it is likely the higher system frame-rate would result better performance in the actual use case. It is expected that if object detection frame-rate is higher the further analysis of the data, namely, object tracking and clustering of the detected person would perform better. Based on this it can be stated that architecture where both object detection and re-identification are executed in the Neural Compute Stick, is the best option.

6 DATA ANALYSIS IN THE CLOUD

The data from camera devices are received on the server-side and stored in the database. Depending on the needs, analysis of the data can be triggered for each new incoming message, or processing can be done for a batch of data, collected during a longer period. For the use case considered in this thesis there are no real-time requirements for processing the data. It is sufficient that an analysis report is provided for example for each day.

This chapter presents a data analysis procedure for counting the number of persons observed during a predefined period, in order to verify the feasibility of the proposed solution for the full end-to-end use case The performance of the described method is tested with a publicly available image database. Developed and testing of the more advanced analysis methodology is beyond the scope of this thesis work and will be subject to further study. However, at the end of the chapter some ideas for more advanced methods and approaches are outlined.

6.1 Server architecture

The design of the data storage and processing cloud server used in this project is described more in detail in the thesis work of Wouter Legiest [88]. However, as the data analysis is designed to take place on the server-side, the main components and functions of the web server system are utilized in this project are discussed here briefly.

A modern cloud server consists of modular architecture where each of the components have a specific role [89]. The server solution used in this project consists of following main components, that are also illustrated in figure 6.1:

- **Web server**, receives the incoming network traffic. It handles invalid requests and can server static content like simple web pages [90]. Other requests are passed to the application server. Web server SW used in this project in **NGINX** [91].
- **Application server**, provides the core functionality of the server. It processes requests coming from the web server, for example calls to the REST services and creates responses with the dynamically generated content. Application server for this project is based on **Django framework** [92].
- **Database**, provides the storage for the received detection and heartbeat data. In order to prepare of a large amount of delivered data, **TimescaleDB** [93] that is

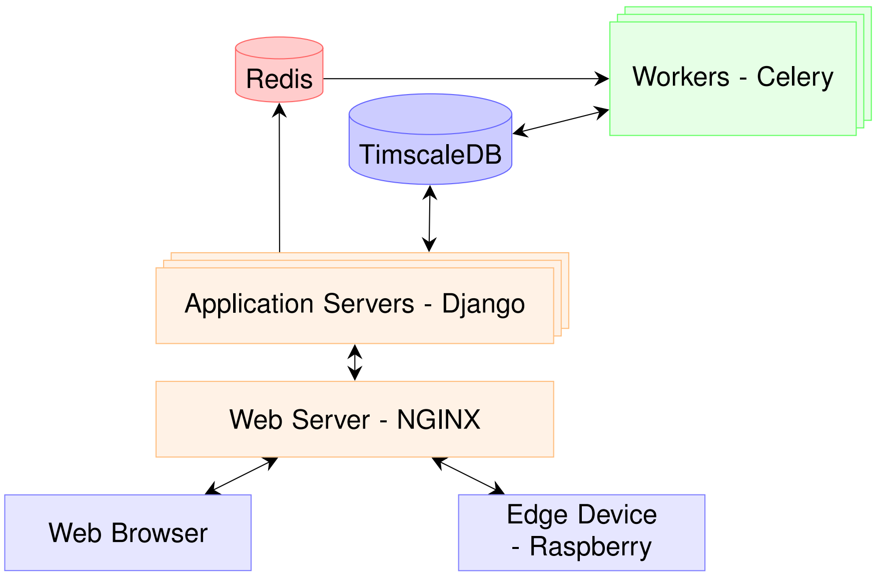


Figure 6.1. *High level architecture of the cloud implementation [88]. From the client-side, the cloud can be accessed with a web browser, for example for viewing the stored data or requesting a report based on the data. Data from edge devices is collected via REST API and stored into the database.*

optimized for handling time-series data is used.

- **Message broker, caching service** Component that works as a fast memory-based database and dispatcher of the messages to a distributed worker. **Redis** [94] is used as a provider of these functions.
- **Distributed worker** takes care of requests that need a dedicated and possible time consuming processing that is not practical to implement inside the application server. It reads message for message broker and start required processing. For example each request for visitor report could trigger a new dedicated data analysis worker. Worker framework called **Celery** [95] was chosen for this application.

Each of the components is implemented as a separate Docker [76] container and running of the system is orchestrated with DockerCompose application. This kind of architecture makes cloning of the solution to another server platform straightforward. The system is deployed on the cPouta [96] cloud computing service, offered by CSC, IT Center for Science Ltd.

6.2 Data analysis for people counting

The problem of the person counting can be formulated as a task for combining all detected people to clusters, where each cluster would represent a unique individual. In an ideal situation the number of the clusters would match the number of observed people. A possible approach for clustering would be to use to operate directly to the feature vectors representing each observation. However, this kind of approach for a large number of detected persons could be computationally intensive. In addition it fails to utilize

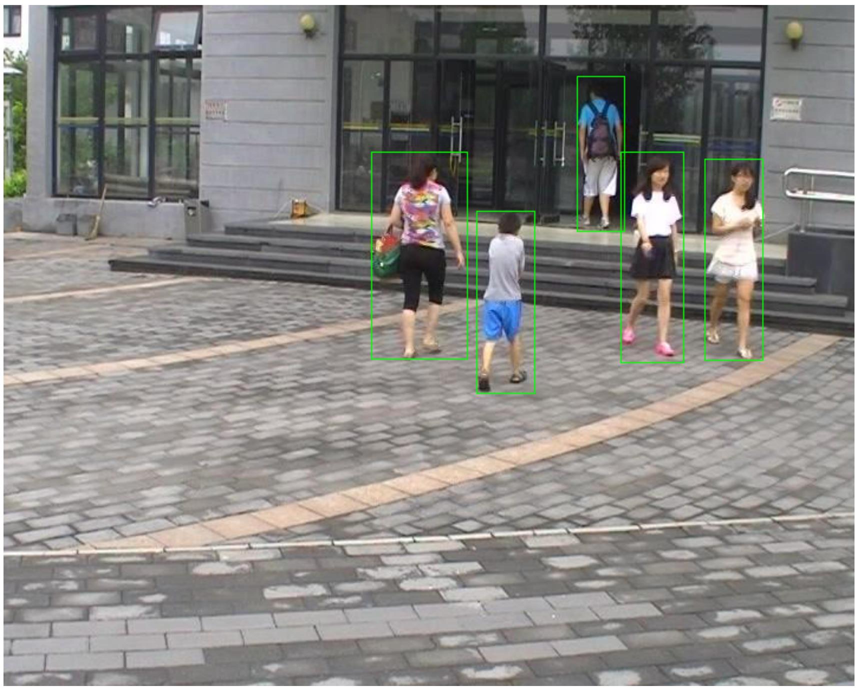


Figure 6.2. *Example image from the dataset.*

implicit spatial and temporal information present in the stored data. This shortcoming could be avoided by first combining individual detections in consecutive frames to *trackelets*, sequences of at least 2 observations that do represent a movement trajectory of a single person. Additional clustering methods can then operate on tracklets, rather than individual feature vectors.

Creating a movement trajectories do also server as a base for other analysis tasks, where for example movement directions and location of the persons are subject of the interest. It should be noted that in some cases only trajectory information without further clustering is sufficient for approximating the number of persons in a particular space. This is the case when entries and exits to a space can be observed. The difference between observed entries and exit represents the number of people present in a given time.

Dataset for experiments

Due to the privacy concerns it is not feasible to collect large image data in actual deployment location. Small scale dataset was collected with consent persons that were participating to the project, but this is used only for testing of the system, like verifying that camera positioning at the installation site or as an input in functional software tests. In order to evaluate the performance of data analysis publicly available image database is used. PRW (Person Re-Identification in the Wild) dataset [97] is chosen for this purpose. It has 11 816 video frames captured with 6 different cameras, 5 with 1920×1080 resolution and 1 with 720×576 resolution. Full video-frames with annotations of bounding box coordinates for each person in the image are provided. When capturing a camera framerate was 25 fps, but the annotation is done only for every 25th frame, so the effective frame-rate of the images in the dataset is at a maximum of 1 fps. Only those images that have a person in them are stored in the set, so in some cases the time between images is larger than 1 s.

Dataset has 932 annotated person identities. A subset of the data is chosen for experiments. Scene, where 720×576 resolution camera was located, is considered having similarities with the target use case. The chosen subset had 1261 image frames and 228 person identities, 8 of those do appear only in a single frame.

6.2.1 Movement trajectories

In this study *a tracking by detection* approach is utilized to combining of objects from the previous frame to the current frame. The benefit of the method is simplicity and robustness. As object coordinate data is not used, camera calibration is not needed. Approach works also with low and varying frame rates when the object motion is not easy to predict. This is the case especially with pedestrians moving around with varying pace, occasionally stopping and changing direction. Also, all-though useful, the exact time information when frames are captured is not mandatory, as long as the order of the frames in time is known.

The basic *tracking by detection* algorithm described in chapter 2 forms a base for the tracklet forming method shown here. The method is extended to search suitable match not only from the just previous frame but it looks p number of frames in the past and compares to the ends of tracklets that have already stopped.

1. For all detected objects c_i in the frame F_n find corresponding object p_i in previous frames $F_{n-k}, 1 \leq k \leq p$ in such way that sum of distances between all paired objects is minimized. A *Hungarian algorithm* [31] [32] described in chapter 2 is utilized for finding the optimal mathes in step 1.
2. If a distance between a pair of objects in consecutive frames is smaller or same than predefined threshold t_1 add object c_i into an existing tracklet. If distance is above the threshold start a new tracklet.

After this process of forming tracklets, there might be individual objects in some frames that are not combined with any other object. We refer to the objects as *singles*. As we defined earlier a tracklet needs to have at least 2 objects, an additional processing step might be required.

3. For each detected object in a frame F_n that does not belong to any tracklet, do calculate a distance to a tracklets that do end in frames $F_{past}, past < n$ and do start $F_{future}, future > n$. Combine a detected object to a tracklet with a smallest distance, if distance is above the threshold t_2.

In step 3 the threshold t_2 is chosen to be larger than in step 2. If after step 3 there are still objects that do not belong to any tracklets, these can be treated as outliers and will be ignored in the future processing.

The two-phase building the tracklets is motivated with an assumption that for all interesting persons there should be at least 2 detections in the frames that near in time. This assumption excludes objects that do appear in the frames only one time. For example, people that are near the image borders, or appear briefly on doorway or window could cause such situations. Also person moving a very fast with respect to the frame-rate across the camera view might be observed only one time. Finally occlusion caused by other moving objects in the scene could limit the number of successful detections. If after tracklet creation there are still a large number of single objects, it is a possible indication that detector device frame-rate, object detector performance, or camera placement is not optimal for the use case.

Distance measure and choice of the thresholds

In *tracking by detection* objects with high similarity are combined. Feature vectors generated by the re-identification neural network are designed to use a cosine similarity to measure how close two objects are from each other. This similarity is defined as a cosine of an angle between vectors. It can be easily calculated by utilizing the Euclidian dot product formula. Given to feature vectors A and B cosine similarity can be calculated as,

$$similarity = cos\theta = \frac{A \cdot B}{||A|| ||B||} \tag{6.1}$$

An equivalent distance measure can be defined as,

$$distance = 1 - cos\theta \tag{6.2}$$

Figure 6.3 shows two examples of tracklets formed with the described process and what are the measured cosine distances between feature vectors of each detected object.

The choice of the threshold when a new tracklet is started is important for the algorithm. Too small threshold will result in too short tracklets and with too high value the risk of joining different people to the same track is increasing. A suitable value for threshold can be estimated by looking at the distribution of the distances between feature vectors observed in consecutive frames. As feature vectors are designed to have a short distance to similar objects and large distances to distinct ones, this distribution can be assumed to have two peaks. A suitable threshold can be selected from the "valley" between these peaks.

Examples of this are shown in figure 6.4, where histograms of distances between consecutive frames are presented. Histograms represent cases, where the feature vectors are calculated with different re-identification networks. Distances corresponding to the lowest value in each valley are, from *a* to *d*, 0.34, 0.42, 0.39, 0,40. It is seen that a more complex network is having a more clear difference between two peaks of the distribution. This supports the assumption that a more complex and more accurate network is able to

Table 6.1. *The number of tracks created in the test set with feature vectors created with different re-identification networks and with different threshold values.*

Re-identification	First phase				Merging Singles			
(resolution, depth)	t_1	Tracks	Singles	Errors	t_2	Tracks	Singles	Errors
	0.38	538	137	7	0.6	461	47	10
160x160, 1.4	0.40	510	116	8	0.6	448	44	10
	0.42	489	101	10	0.6	437	40	11
	0.38	542	147	7	0.6	457	44	13
128x128, 0.75	0.40	513	118	8	0.6	447	43	12
	0.42	489	99	8	0.6	437	39	13
	0.38	582	170	8	0.6	485	56	14
96x96, 1.0	0.40	533	151	13	0.6	469	53	20
	0.42	526	131	14	0.6	459	54	20
	0.38	650	211	7	0.6	535	66	21
96x96, 0.35	0.40	601	170	9	0.6	508	56	20
	0.42	576	153	9	0.6	493	53	19

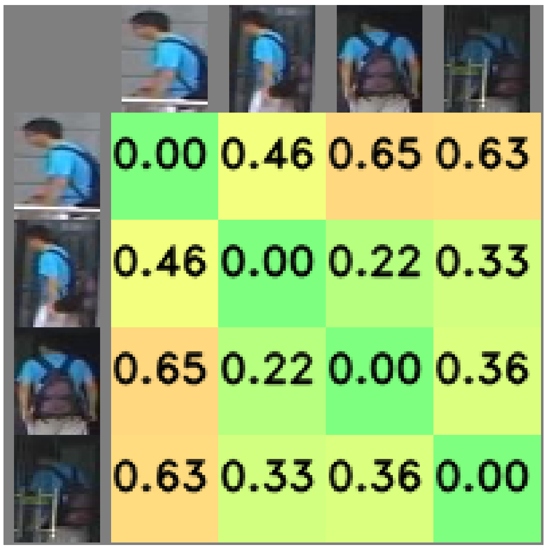
separate matching and non-matching objects better than lower accuracy networks.

Experiments

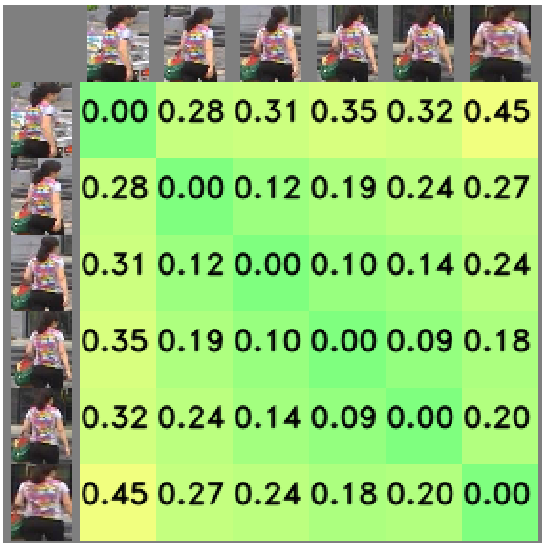
Four different re-identification network configurations are tested for the tracklet creation. Configurations do represent a set of various trade-offs between accuracy and speed, further analyzed in chapter 5. In the experiment networks generate feature vectors for objects in the selected test set. These are then combined into tracklets with two-phase method presented earlier. The initial threshold for all cases in the first phase is 0.4, chosen as a median of the values that give the lowest point between peaks in all histograms shown in figure 6.4. Value was varied within $\pm 5\%$ interval, to see how sensitive the results are for threshold selection. Value for second threshold t_2 was chosen more heuristically to be 50% larger than t_1. Value is still in the edge of the "valley" in all histograms and was verified with additional experiments to produce good compromise between correct and false merging of singles.

Result are shown in table 6.2.1. Table includes: total number of tracklets, how many of those are singles and number of errors, i.e. cases were two different identities defined in ground truth annotation are combined erroneously to the same tracklet. Results are shown after the initial phase and after the merging of singles at the second phase.

It can be observed that the first phase results are relatively sensitive to the choice of the threshold, but the effect is smaller after the second phase. As expected, more accurate networks produce features that are better for forming tracklets, the total number is smaller with fewer singles and errors. When considering the next clustering phase, the small error amount is considered more important than a low tracklet count. Forming of clusters can fix discontinued trajectories, but the method used in this work will not break existing

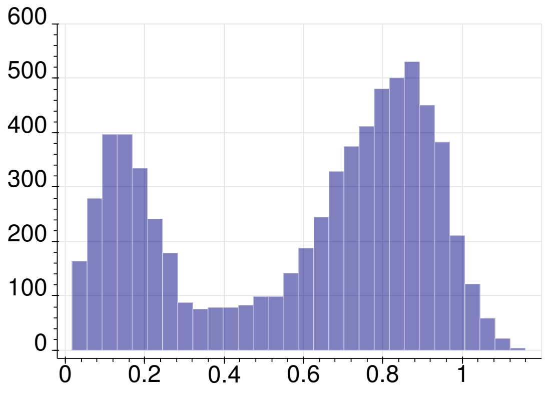


0.00	0.46	0.65	0.63
0.46	0.00	0.22	0.33
0.65	0.22	0.00	0.36
0.63	0.33	0.36	0.00

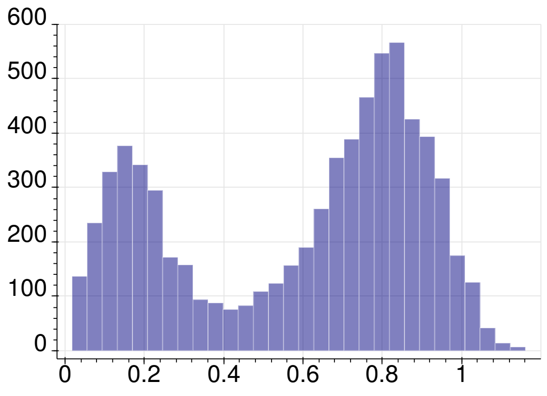


0.00	0.28	0.31	0.35	0.32	0.45
0.28	0.00	0.12	0.19	0.24	0.27
0.31	0.12	0.00	0.10	0.14	0.24
0.35	0.19	0.10	0.00	0.09	0.18
0.32	0.24	0.14	0.09	0.00	0.20
0.45	0.27	0.24	0.18	0.20	0.00

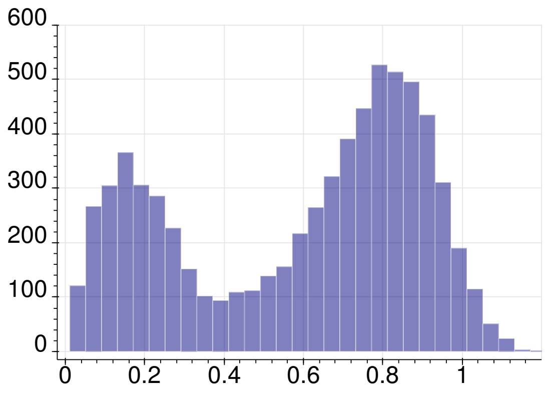
Figure 6.3. *Examples of distances between objects belonging in to a tracklet. Feature vectors are generated with 160x160, depth 1.4 network configuration.*



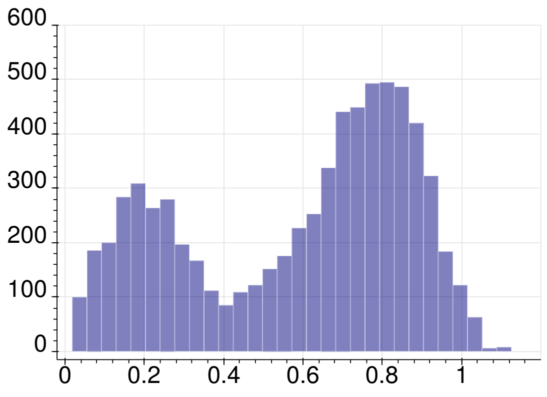
(a) *Resolution 160 × 160 depth 1.4*



(b) *Resolution 128 × 128 depth 0.75*



(c) *Resolution 96 × 96 depth 1.0*



(d) *Resolution 96 × 96 depth 0.35*

Figure 6.4. *Examples of measured distances between feature vectors. On top a distance matrix between person belonging to a the same tracklet. Below of that histograms of distances observed between vectors in the consecutive frames. Vectors for histograms are generated with different network parameters, shown below of each graph.*

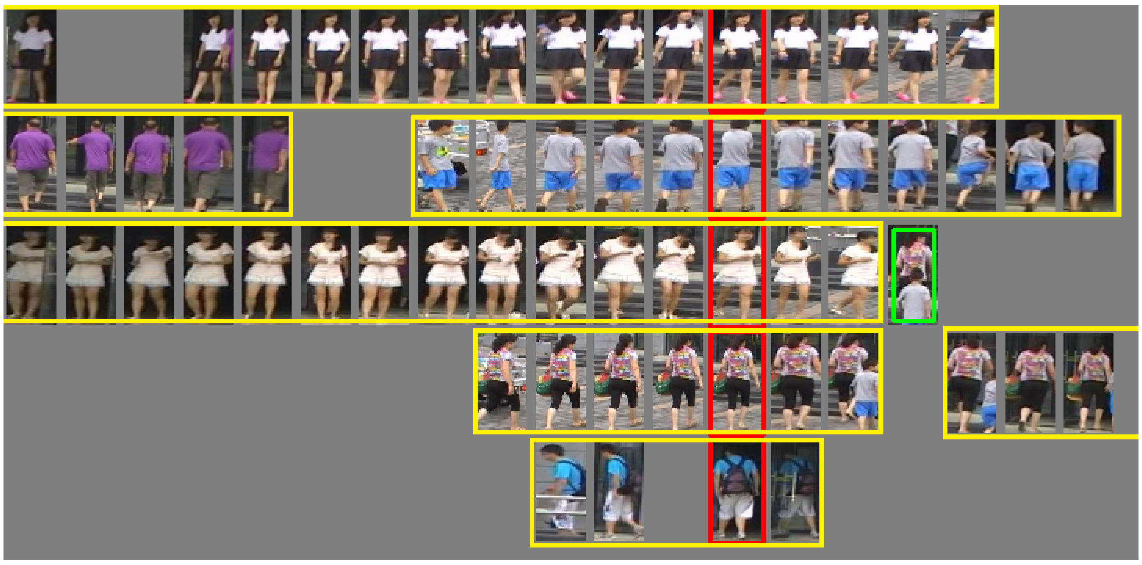


Figure 6.5. *Examples of tracks created. Images outlined with yellow border belong to the same tracklet. Each vertical column shows all detected objects in a single frame. Objects outlined with red color are from image shown in fig.6.2. Image marked with green border is a single detection.*

tracklets, so error in this phase will remain and can possibly impact also to the accuracy of the clustering process.

After two-stage tracklet creation process using threshold value $t_1 = 0.4$ the re-identification configuration with 160 $\times$ 160 resolution and 1.4 depth is giving overall best performance. Configuration with 128 $\times$ 128 resolution and 0.75 depth is producing very similar results in terms of the total number of tracklets but with a higher amount or errors. With the other two tested configurations results are worse, the total number of tracklets is larger, and still there more erroneously combined detections.

An example of created tracklets are shown in figure 6.5. Each row in the figure shows small images of persons belonging to the same tracklet. Columns indicate the video frame from which persons are detected. For example, images outlined with red border are from the same video frame, shown in dataset example figure 6.2.

In the figure there are 5 successfully created tracklets. However, the one in the second row from below that is split into two parts. The image marked with a green outline is a single detection that is not combined with any tracklet. It can be easily seen that it should have been part of the divided track below. Closer analysis shows that person in that image is heavily occluded by another person, resulting in a feature vector with a distance greater than 0.6 to any other nearby tracklet.

The short tracklet at the bottom row is an example of the successful merging of single detection in the second phase. The distance between features of this tracklet can be seen in the figure 6.3, left side. It is observed that not all distances are below 0.4 threshold value. The second stage merging with threshold 0.6 has joined otherwise isolated detection to this track.

6.2.2 Clustering

The target is to create clusters from data stored in the server database so that all observations from the same person are associated with a single cluster. There are extensive amounts of different algorithms for clustering and choosing the best approach for a particular task and data is not straightforward [98].

As for the person counting use case, the interesting output is the number of clusters, rather than clusters itself, the clustering algorithm should work without the target cluster count as input parameter. Instead, the algorithm should stop when some other criteria are reached. This rules out of some approaches, like basic K-means clustering [99]. Also, algorithms such as K-means that do rely on the assumption that cluster are convex, is most likely not optimal for the current data. It is hypothesized that feature vectors from a single person would form a more complex manifold in high dimensional embedding space.

Algorithms like DBSCAN [100] and OPTICS [101] are fulfilling the criterion of not requiring the number of the cluster as an input parameter. Also as they are based on clustering regions of high density separated by low-density areas, there is no prior assumption made from the actual shape of the cluster. However, a hypothesis that clustering of tracklets would be performing better than the clustering of individual feature vectors impacts to the choice of the algorithm. Initial clustering experiments of individual feature vector samples with DBSCAN and OPTICS indicate lower performance than with agglomerate clustering based on combining tracklets. But it should be noted that this hypothesis is not verified in the current work and will be subject to a further study, beyond the scope of this thesis.

The final criterion, considering the scope of the work, is implementation complexity. A method that is conceptually simple, relatively fast for running on the server system, and could be implemented with python as a Celery worker process in the cloud server. Based on the analysis above a variant of agglomerative clustering algorithm was chosen as a means for clustering. The method is outlined as follows:

1. Initialize a set of clusters, so that all clusters do contain a single tracklet.
2. Find a pair of clusters having the smallest distance.
3. If the distance of the pair is smaller than a threshold t_c and any of the tracklets in clusters to be combined do not overlap in time, merge them into a new cluster and delete originals. If no more suitable pairs for merging can be found, stop the algorithm.
4. Go back to step 2.

Operation of the algorithm requires a distance measure between clusters. There exists several different alternatives for distances between two sets of vectors $A = \mathbf{a_0}, \mathbf{a_1}, ..., \mathbf{a_n}$ and $B = \mathbf{a_0}, \mathbf{a_1}, ..., \mathbf{a_n}$. Some examples of these are:

- *Single (minimum) link*: Calculate all distances between vectors in set A and set B.

Choose the smallest of those as distance between sets.

- *Complete (maximum) link*: Calculate all distances between vectors in set A and set B. Choose the smallest of those as distance between sets.
- *Average link*, Calculate all distances between vectors in set A and set B. Choose the average of those as distance between sets.

For algorithm presented in we choose a distance measure combining properties from both average and minimum distances. A distance between two tracklets is measured by calculating an *average* of cosine distances between each individual feature vector. Distance between two clusters is then obtained as a *minimum* distances between tracklets belonging to clusters.

The clustering method is impacted also by the choice of the threshold, too small value will not combine all tracklets and too high value will create erroneous clusters that do represent more than one individual. As distance between clusters is fundamentally based on distances between feature vectors, the same value that was used when forming tracklets can be utilized as a starting point. This can be further optimized for a particular scenario by experimentation.

Experiments

Tracklets formed with the methodology described in section 6.2.1 are the starting point for the clustering. The same 4 re-identification network configurations used in previous tracklet forming experiments are compared. Thresholds values $t_1 = 0.4$ and $t_2 = 0.6$ are applied for all configurations. The impact of removing single detections, prior to clustering is also investigated. Clustering is performed with two alternative scenarios, first operating on the direct output of the two-phase tracklet creation process and secondly when tracks containing only one single detection are removed prior to clustering. Threshold $t_c = 0.4$ was used. Results of the experiments are shown in table 6.2

Table 6.2. *The number of cluster created in the test set with feature vectors created with different re-identification networks. Upper section shows results with singles included to the clustering and lower section when they are ignored from the clustering.*

Re-identification		Start stage			After merging		
(resolution, depth)	t_c	Clusters	Singles	Errors	Clusters	Singles	Errors
160x160, 1.4	0.4	448	44	10	232	24	29
128x128, 0.75	0.4	447	43	12	285	28	33
96x96, 1.0	0.4	469	53	20	319	37	38
96x96, 0.35	0.4	508	56	20	355	37	37
160x160, 1.4	0.4	404	0	10	211	0	27
128x128, 0.75	0.4	404	0	12	259	0	30
96x96, 1.0	0.4	416	0	20	283	0	35
96x96, 0.35	0.4	452	0	20	321	0	34

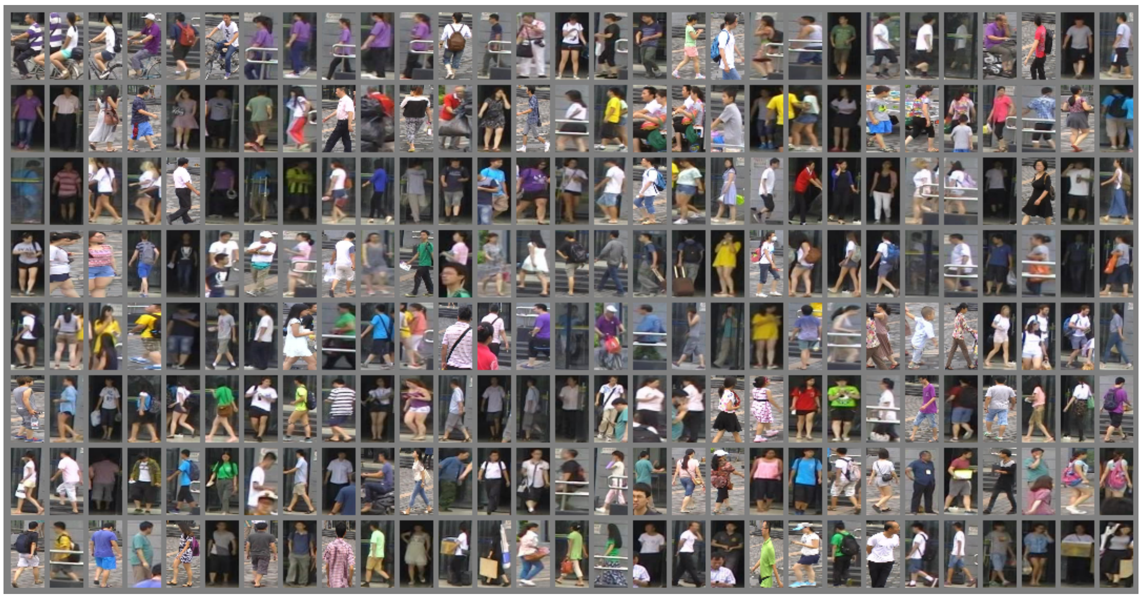


Figure 6.6. *Example of clustering. First image of each 232 cluster created based on features from 160 $\times$ 160, depth 1.4 re-identification network.*

The same subset of the PRW dataset [97], used in the previous experiment, is also utilized here. As mentioned earlier, the chosen subset had 1261 image frames and 228 person identities, from which 8 persons do appear only in a single frame.

From the experimental results it can be seen that network configuration with 160 $\times$ 160 resolution and 1.4 depth value is performing with the best accuracy in approximating the number of persons present in the dataset. With this configuration, without removing singles, the cluster count is 232, a value that is very close to the ground truth value of 230. The clusters formed with this set-up are shown in figure 6.6. When singles are removed before clustering the cluster count is 211. In both cases there are still about 13% of erroneously created clusters containing more than one person identity. This indicates also that there clusters of the same identity that have failed to merge. With other configuration the cluster count and respective error count are higher.

Results are promising and indicate that clustering of the tracklets, based only on the feature vector data generated with a re-identification neural network, can approximate the person count in this use case. However, the final verdict of the person counting accuracy remains cannot be stated with the limited dataset used in this experiment. This is seen as a topic for further experiments and analysis.

6.3 Discussion

In this chapter an analysis method that can be applied for the data stored in a cloud service is developed. The analysis is based only on the feature vector data provided by edge devices. Conducted experiments indicate that creation of motion trajectories of persons and clustering them for approximation of the person count is feasible. Based on this it can be concluded that the realization full end-to-end use case described in chapter 4 is possible.

Experimental results shown here are based on a publicly available image database as the large scale data collection at the actual use case location was not possible due to data privacy reasons. Although the dataset shares similarities with the scenes in the target usage environment there are also issues that might lower the analysis accuracy when compared to the situation if the data would have been collected from the installation location with the actual edge devices. More importantly as the PWR dataset is primarily intended for re-identification and object detection evaluation, the frame rate is limited at the best to 1 fps. With this low frame-rate the feature vector differences between persons in successive frames are more likely to be larger than if observations would have been done with shorter intervals. As a consequence of this, the thresholds used in forming tracklets and merging clusters are relatively high, producing errors observed in the results. Higher frame-rate would also allow tracklets to have more frames, resulting in better re-presentation of the person and lower probability for single detection. It is likely that this would also improve the clustering performance. For further experiments a higher frame-rate dataset should be obtained.

Also the described approaches for creating tracklets and clustering them can be further improved. For example, the position information of detected objects can be utilized in connection with detection by tracking scheme. The current method is also sensitive for the selection of the thresholds applied in tracklet formation. More advanced ideas, such as dynamically adapting threshold based on the distance distribution of the tracklets are topics to investigate further.

Testing of the more advanced clustering algorithms is a relevant topic for further study. Also the chosen distance measure between tracklets and clusters can have a potentially significant impact on the accuracy of the algorithm. The preliminary analysis of clusters indicate that feature vectors in the embedding space can form complex manifolds for which simple distance-based metric perform poorly. The experimenting with subspace and manifold distances, for example as presented in [102], could be a promising direction.

7 CONCLUSION

The main research question for this thesis is the feasibility of a computer vision system that while performing people monitoring tasks, would not violate the privacy of the individual. Furthermore, the target is that privacy is protected *implicitly* by the design. This means that it is not sufficient only to protect privacy-sensitive information by traditional *explicit* means, such as, cryptography and restring access to the data with security keys or passwords. The design needs to ensure that privacy-sensitive data is not be stored or transmitted at all anywhere in the system.

The concept of implicit privacy has several benefits over more traditional methods. There is no need to separately manage access to sensitive personal information beyond the normal access control of the system. Also unauthorized access to the data stored in the cloud server, due to SW failures, human errors, or malicious hacking would cause significantly less damage as the data itself does not contain privacy critical information. From a legal perspective, a system that does not store personal information, has also less strict requirements. For example, the general data protection regulation (GDPR) is less concerned if it is possible to demonstrate that already the system design prevents the existence of personal data. Also as technology development has made video surveillance systems accessible for a wider range of users, regulators, and policymakers in the public and the private sectors are setting stricter limitations for the use of such solutions. However, if it can be shown that although cameras are used as a part of the monitoring solution, the images are not transmitted anywhere, they are potentially accepted easier for installation.

It should be noted that the implicit privacy of data processing is not replacing any normal security requirements to the system. Any access to any compute or data resource needs to be in any case protected by standard best practices, encrypted communication, secure keys, and strong passwords. If these security measures are not followed it would be possible, for example, by hacking to get access to the critical system component and alter the implementation in such a way that data privacy is compromised.

The important viewpoint, as part of the research question, is how feasible in practice a privacy-aware system is to implement. Real-life use case sets restrictions for costs, physical size, and power consumption of the hardware, to mention a few. These limitations are then imposing further constraints on the software and algorithms utilized in the system. This study approaches the research question with practical design and implementation of a distributed computer vision system for a real-life use case, a visitor counting in a public museum. The solution consists of a fleet of camera-equipped edge devices that can be remotely managed. Cameras detect persons moving in the view and for each detection create a feature vector that is sent to the cloud server. Algorithms running in the server analyzing the data and provide a report about the number of the visitor.

The experiments and performance measurements done with implemented edge devices show that the simultaneous use of deep neural networks for object detection and feature generation using low-cost off-the-shelf hardware is feasible. The design recommendation as a result of the study is that use of a dedicated HW accelerator for running all neural networks is preferred. The analysis also show that the variation of the accuracy and computational complexity of used neural networks offers a range of feasible trade-off choices between performance and accuracy. Also a design of the data analysis algorithm for the server-side implementation is done. The study evaluates the feasibility of utilizing only feature vector data for tracking and clustering. The experiments indicate that with this approach it is possible to approximate visitor count as defined by the requirements of the use case.

The analysis of the data and experiments with implemented algorithms do show that there is potential to improve the solution further. Some future study items concerning improvements for tracking and clustering are discussed at the end of chapter 6. In addition to those also improvements in the performance, in both accuracy and speed, of utilized neural networks would be beneficial.

When considering the proposed design with implicit privacy protection, there are still open questions if the system can be fully proven to be secure in this respect in all conditions. A critical component in the system is the model used for the creation the feature vector. In normal conditions these models are part of the internal SW of the edge device and not accessible from outside. But if someone gains access to a model, the open question is that would it be possible to invert the model to the level that person could be recognized from the feature vector data.

As a conclusion it can be stated that practical design and implementation of computer vision systems that do implicitly protect the privacy of individuals is possible. In the future this type of design might be seen even as mandatory for a larger range of different use cases. As there remain interesting open topics in this field, further studies around the subject are invited.

REFERENCES

[1] Lage, E. S., Santos, R. L., Junior, S. M. T. and Andreotti, F. Low-Cost IoT Surveillance System Using Hardware-Acceleration and Convolutional Neural Networks. *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. Apr. 2019, 931–936. DOI: 10.1109/WF-IoT.2019.8767325.

[2] Dey, S., Mondal, J. and Mukherjee, A. Offloaded Execution of Deep Learning Inference at Edge: Challenges and Insights. *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. Mar. 2019, 855–861. DOI: 10.1109/PERCOMW.2019.8730817.

[3] Zhao, J., Mortier, R., Crowcroft, J. and Wang, L. Privacy-Preserving Machine Learning Based Data Analytics on Edge Devices. *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*. AIES '18. New Orleans, LA, USA: ACM, 2018, 341–346. ISBN: 978-1-4503-6012-8. DOI: 10.1145/3278721.3278778. URL: http://doi.acm.org/10.1145/3278721.3278778.

[4] Fredrikson, M., Jha, S. and Ristenpart, T. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. Denver, Colorado, USA: ACM, 2015, 1322–1333. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813677. URL: http://doi.acm.org/10.1145/2810103.2813677.

[5] Szeliski, R. .-.-. A. and Applications. *Computer Vision - Algorithms and Applications*. Springer, London, 2011. ISBN: 978-1-84882-934-3.

[6] *Raspberry Pi Products*. [Online; accessed 21-April-2020]. URL: https://www.raspberrypi.org/products.

[7] *Jetson Nano*. [Online; accessed 21-April-2020]. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/.

[8] Nakamura, J. *Image Sensors and Signal Processing for Digital Still Cameras*. CRC Press, Apr. 2016. ISBN: 978-0849335457.

[9] *mipi alliance - Camera and Imaging*. [Online; accessed 21-April-2020]. URL: https://mipi.org/specifications/camera-and-imaging.

[10] *Snapdragon System-in-Package*. [Online; accessed 21-April-2020]. URL: https://www.qualcomm.com/products/snapdragon-system-package.

[11] *Samsung Exynos processors*. [Online; accessed 21-April-2020]. URL: https://www.samsung.com/semiconductor/minisite/exynos/products/all-processors/.

[12] Intel. *Intel Movidius Myriad X VPU*. https://www.movidius.com/myriadX. [Online; accessed 21-October-2019].

[13] Ramanath, R., Snyder, W. E., Bilbro, G. L. and Sander, W. A. Demosaicking methods for Bayer color arrays. *J. Electronic Imaging* 11 (2002), 306–315.

[14] Ramanath, R., Snyder, W. E., Yoo, Y. and Drew, M. S. Color image processing pipeline. *IEEE Signal Processing Magazine* 22.1 (2005), 34–43.

[15] Kao, W., Wang, S., Chen, L. and Lin, S. Design considerations of color image processing pipeline for digital cameras. *IEEE Transactions on Consumer Electronics* 52.4 (2006), 1144–1152.

[16] Barnard, K., Martin, L., Coath, A. and Funt, B. A comparison of computational color constancy Algorithms. II. Experiments with image data. *IEEE Transactions on Image Processing* 11.9 (2002), 985–996.

[17] Nikkanen, J., Gerasimow, T. and Kong, L. Subjective effect of white-balancing errors in digital photography. *Optical Engineering - OPT ENG* 47 (Nov. 2008). DOI: 10.1117/1.3013232.

[18] Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[19] Dalal, N. and Triggs, B. Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886–893 vol. 1.

[20] Nguyen, T., Park, E.-A., Han, J., Park, D.-C. and Min, S.-Y. Object Detection Using Scale Invariant Feature Transform. *Genetic and Evolutionary Computing*. Ed. by J.-S. Pan, P. Krömer and V. Snášel. Cham: Springer International Publishing, 2014, 65–72. ISBN: 978-3-319-01796-9.

[21] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[22] Viola, P. and Jones, M. Robust Real-Time Face Detection. *International Journal of Computer Vision* 57 (May 2004), 137–154. DOI: 10.1023/B:VISI.0000013087.49260.fb.

[23] Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J. and Zisserman, A. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision* 111.1 (Jan. 2015), 98–136.

[24] Girshick, R., Donahue, J., Darrell, T. and Malik, J. Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.1 (2016), 142–158.

[25] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. and Berg, A. C. SSD: Single Shot MultiBox Detector. *Computer Vision – ECCV 2016*. Ed. by B. Leibe, J. Matas, N. Sebe and M. Welling. Cham: Springer International Publishing, 2016, 21–37.

[26] Yrjänäinen, J. *Neuroverkkokiihdytystä käyttävän konenäkösovelluksen toteutus*. Mar. 2019. URL: http://urn.fi/URN:NBN:fi:tty-201902251262.

[27] Kalman, R. E. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering* 82 (1960), 35–45. DOI: doi:10.1115/1.3662552.

[28] Zhang, L., Li, Y. and Nevatia, R. Global data association for multi-object tracking using network flows. June 2008. DOI: 10.1109/CVPR.2008.4587584.

[29] Pirsiavash, H., Ramanan, D. and Fowlkes, C. C. Globally-optimal greedy algorithms for tracking a variable number of objects. *CVPR 2011*. 2011, 1201–1208.

[30] Berclaz, J., Fleuret, F., Turetken, E. and Fua, P. Multiple Object Tracking Using K-Shortest Paths Optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.9 (2011), 1806–1819.

[31] Kuhn, H. The Hungarian method for the assignment problem. (1955), 83–97.

[32] Munkres, J. Algorithms for the Assignment and Transportation Problems. (1957).

[33] MurrayCampbella, Jr., J. H. and Hsu, F.-h. Deep Blue. *Artificial Intelligence* 134 (Jan. 2002), 57–83. ISSN: 0004-3702. URL: `https://doi.org/10.1016/S0004-3702(01)00129-1`.

[34] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Driessche, G. van den, Graepel, T. and Hassabis, D. Mastering the game of Go without human knowledge. *Nature* 550 (Oct. 2017). Article, 354–359. URL: `https://doi.org/10.1038/nature24270`.

[35] Kohonen, T. *Self-Organizing Maps*. Springer Series in Information Sciences). Springer-Verlag Berlin Heidelberg, 2001. ISBN: 978-3-540-67921-9.

[36] McCulloch, W. S. and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* 5 (1943), 115–133. ISSN: 0007-4985. URL: `https://doi.org/10.1007/BF02478259`.

[37] Rosenblatt, F. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books, 1962. URL: `https://books.google.ca/books?id=7FhRAAAAMAAJ`.

[38] London, M. and Häusser, M. DENDRITIC COMPUTATION. *Annual Review of Neuroscience* 28.1 (2005). PMID: 16033324, 503–532. DOI: `10.1146/annurev.neuro.28.061604.135703`. URL: `https://doi.org/10.1146/annurev.neuro.28.061604.135703`.

[39] Polyak, B. Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics* 4 (Dec. 1964), 1–17. DOI: `10.1016/0041-5553(64)90137-5`.

[40] Duchi, J., Hazan, E. and Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* 12.61 (2011), 2121–2159. URL: `http://jmlr.org/papers/v12/duchi11a.html`.

[41] Hinton, G. *Neural Networks for Machine Learing*.

[42] Kingma, D. P. and Ba, J. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: `1412.6980 [cs.LG]`.

[43] Ioffe, S. and Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by F. Bach and D. Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, 448–456. URL: `http://proceedings.mlr.press/v37/ioffe15.html`.

[44] Rumelhart, D. E., Hinton, G. E. and Williams, R. J. Learning representations by back-propagating errors. *Nature* 323.6088 (1986), 533–536. ISSN: 1476-4687. DOI: `10.1038/323533a0`. URL: `https://doi.org/10.1038/323533a0`.

[45] Leshno, M., Lin, V. Y., Pinkus, A. and Schocken, S. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks* 6.6 (1993), 861–867. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80131-5. URL: http://www.sciencedirect.com/science/article/pii/S0893608005801315.

[46] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115.3 (2015), 211–252. DOI: 10.1007/s11263-015-0816-y.

[47] Krizhevsky, A., Sutskever, I. and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger. Curran Associates, Inc., 2012, 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[48] Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[49] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[50] Simonyan, K. and Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556* (Sept. 2014).

[51] He, K., Zhang, X., Ren, S. and Sun, J. Deep Residual Learning for Image Recognition. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[52] Girshick, R. Fast R-CNN. *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, 1440–1448.

[53] Ren, S., He, K., Girshick, R. and Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett. Curran Associates, Inc., 2015, 91–99. URL: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf.

[54] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, 779–788.

[55] Hermans, A., Beyer, L. and Leibe, B. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737* (2017).

[56] Oh Song, H., Xiang, Y., Jegelka, S. and Savarese, S. Deep Metric Learning via Lifted Structured Feature Embedding. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2016.

[57] Deng, J., Guo, J., Xue, N. and Zafeiriou, S. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2019.

[58] Wang, X., Hua, Y., Kodirov, E., Hu, G., Garnier, R. and Robertson, N. M. Ranked List Loss for Deep Metric Learning. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2019.

[59] Luo, H., Gu, Y., Liao, X., Lai, S. and Jiang, W. Bag of tricks and a strong baseline for deep person re-identification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2019.

[60] He, K., Zhang, X., Ren, S. and Sun, J. Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, 770–778.

[61] *Open Neural Network Exchange*. `https://onnx.ai/`. [Online; accessed 10-May-2019].

[62] *Tensorflow, An end-to-end open source machine learning platform*. `https://www.tensorflow.org/`. [Online; accessed 10-May-2019].

[63] *Keras Documentation*. `https://keras.io/`. [Online; accessed 10-May-2019].

[64] *PyTorch*. `https://pytorch.org/`. [Online; accessed 10-May-2019].

[65] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* (2014).

[66] *MXNet A Flexible and effiencit library for deep learning*. `https://mxnet.apache.org/`. [Online; accessed 10-May-2019].

[67] Intel. *Intel Distribution of OpenVINO toolkit*. `https://software.intel.com/en-us/openvino-toolkit`. [Online; accessed 10-May-2020].

[68] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S. and Murphy, K. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, 3296–3297.

[69] Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y. and Girshick, R. *Detectron2*. `https://github.com/facebookresearch/detectron2`. 2019.

[70] Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3.5 (2016), 637–646.

[71] *Raspberry Pi Camra Module*. [Online; accessed 21-April-2020]. URL: `https://www.raspberrypi.org/documentation/hardware/camera/`.

[72] *Intel® Neural Compute Stick 2 (Intel® NCS2)*. [Online; accessed 21-April-2020]. URL: `https://software.intel.com/content/www/us/en/develop/hardware/neural-compute-stick.html`.

[73] *https://azure.microsoft.com/en-us/services/iot-edge/*. [Online; accessed 21-April-2020]. URL: `https://www.balena.io/`.

[74] *https://aws.amazon.com/iot/solutions/iot-edge/*. [Online; accessed 21-April-2020]. URL: `https://www.balena.io/`.

[75] *balena - The complete IoT fleet management platform*. [Online; accessed 21-April-2020]. URL: https://www.balena.io/.

[76] *Docker documentation*. [Online; accessed 21-April-2020]. URL: https://docs.docker.com/.

[77] Morabito, R. Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation. *IEEE Access* 5 (2017), 8835–8850.

[78] Morabito, R., Cozzolino, V., Ding, A. Y., Beijar, N. and Ott, J. Consolidate IoT Edge Computing with Lightweight Virtualization. *IEEE Network* 32.1 (2018), 102–111.

[79] Ren, S., He, K., Girshick, R. and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*. 2015, 91–99.

[80] Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A. and Chen, L. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. *CoRR* abs/1801.04381 (2018). arXiv: 1801.04381. URL: http://arxiv.org/abs/1801.04381.

[81] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S. and Murphy, K. *Speed/accuracy trade-offs for modern convolutional object detectors*. 2016. arXiv: 1611.10012 [cs.CV].

[82] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L. and Dollár, P. *Microsoft COCO: Common Objects in Context*. 2014. arXiv: 1405.0312 [cs.CV].

[83] Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Duerig, T. and Ferrari, V. The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv:1811.00982* (2018).

[84] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition*. Ieee. 2009, 248–255.

[85] Zheng, L., Shen, L., Tian, L., Wang, S., Wang, J. and Tian, Q. Scalable person re-identification: A benchmark. *Proceedings of the IEEE International Conference on Computer Vision*. 2015, 1116–1124.

[86] Zhong, Z., Zheng, L., Kang, G., Li, S. and Yang, Y. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896* (2017).

[87] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. Rethinking the inception architecture for computer vision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, 2818–2826.

[88] Legiest, W. *Design of a back-end for a camera based person detection system*. Aug. 2019. URL: http://urn.fi/URN:NBN:fi:tuni-201908092857.

[89] Fulton, J. *Web Architecture 101*. [Online; accessed 21-April-2020]. Nov. 2017. URL: https://engineering.videoblocks.com/web-architecture-101-a3224e126947.

[90] Butler, T. *NGINX Cookbook*. Packt Publishing, 2017. ISBN: 9781786466174.

[91] *NGINX - homepage*. [Online; accessed 21-April-2020]. URL: https://www.nginx.com/.

[92] *Django - homepage*. [Online; accessed 21-April-2020]. URL: https://www.djangoproject.com/.

[93] *TimescaleDB Overview*. [Online; accessed 21-April-2020]. URL: https://docs.timescale.com/latest/introduction.

[94] *Redis - homepage*. [Online; accessed 21-April-2020]. URL: https://redis.io/.

[95] *Celery - homepage*. [Online; accessed 21-April-2020]. URL: http://www.celeryproject.org/.

[96] *cPouta Community Cloud*. [Online; accessed 21-April-2020]. URL: https://research.csc.fi/cpouta.

[97] Zheng, L., Zhang, H., Sun, S., Chandraker, M., Yang, Y. and Tian, Q. Person Re-Identification in the Wild. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[98] Hämäläinen, W., Kumpulainen, V. and Mozgovoy, M. Evaluation of clustering methods for adaptive learning systems. Jan. 2014, 237–260. DOI: 10.4018/978-1-4666-6276-6.ch014.

[99] Lloyd, S. P. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* 28 (1982), 129–136.

[100] Ester, M., Kriegel, H.-P., Sander, J. and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. *KDD-96 Proceedings*. AAAI Press, 1996, 226–231.

[101] Ankerst, M., Breunig, M., Kriegel, H.-P. and Sander, J. OPTICS: Ordering Points to Identify the Clustering Structure. Vol. 28. June 1999, 49–60. DOI: 10.1145/304182.304187.

[102] Wang, R., Shan, S., Chen, X., Dai, Q. and Gao, W. Manifold–Manifold Distance and its Application to Face Recognition With Image Sets. *IEEE Transactions on Image Processing* 21.10 (2012), 4466–4479.

[103] Pilgrim, R. *Tutorial on Implementation of Munkres' Assignment Algorithm*. [Online; accessed 21-April-2020]. Aug. 1995. DOI: 10.13140/RG.2.1.3572.3287. URL: http://csclab.murraystate.edu/~bob.pilgrim/445/munkres.html.

A HUNGARIAN ALGORITHM

Method published by Harold W. Kuhn [31] and James Munkres [32], called a *Hungarian algorithm* or *Kuhn-Munkres algorithm* can be used to solve *linear sum assignment problem* also known as *minimum weight matching in bipartite graphs* in polynomial time.

A high-level operation of the one version of the algorithm can be outlined in the following steps, adopted from [103]:

0. Create a $n \times m$ cost matrix C where each element of the matrix represent the distance between objects in previous and current frames.
1. For each row of the matrix, find the smallest element, and subtract it from every element in that row.
2. Find a zero in the resulting matrix. If there is no marked zero in its row or column mark it (red color in figure A.1). Repeat for all elements.
3. Cover (gray color in fig.) each column containing a marked zero. If $min(x, y)$ columns are covered, the marked zeros are showing the complete assignment go to final step 7, otherwise continue to step 4.
4. Find a noncovered zero and prime (color blue in fig.) it. If there are no marked zero in the row containing this primed zero go to step 6. otherwise cover this row and uncover the column containing the marked zero. Continue in this manner until there are no uncovered zeros left. Save the smallest uncovered value (color red in fig.) and go to step 6.
5. Construct a series of alternating primed and marked zeros as follows (red outline in fig.). Let Z0 represent the uncovered primed zero found in Step 4. Let Z1 denote

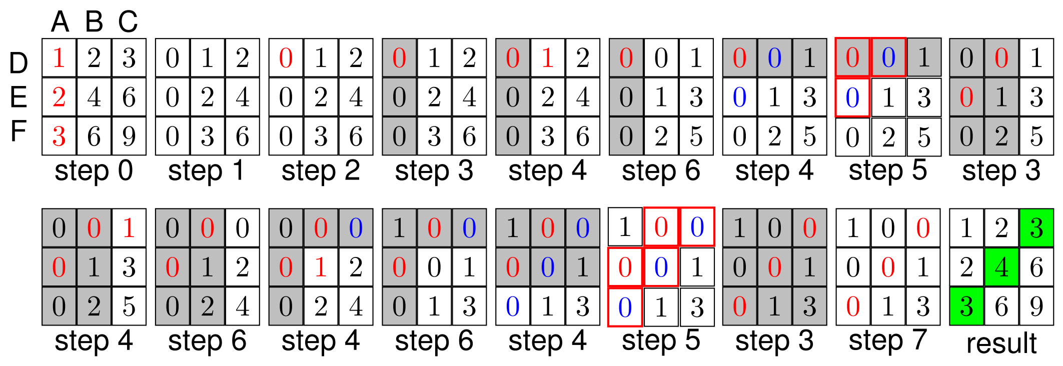


Figure A.1. *Example of Hungarian algorithm progress to find optimal assignment costs.*

the marked zero in the column of Z0 (if any). Let Z2 denote the primed zero in the row of Z1 (there will always be one). Continue until the series terminates at a primed zero that has no marked zero in its column. Unmark each marked zero of the series, mark each primed zero of the series, erase all primes and uncover every line in the matrix. Return to Step 3

6. Add the value found in Step 4 to every element of each covered row, and subtract it from every element of each uncovered column. Return to Step 4 without altering any marks, primes, or covered lines.
7. Assignment pairs are indicated by the positions of the marked zeros in the cost matrix. If $C(i, j)$ is a starred zero, then the element associated with row i is assigned to the element associated with column j.

An example of how these steps are iterated in practice is illustrated in figure A.1. Figure show 3 $\times$ 3 matrix that gives a cost between all combination of items from set A,B,C and C,E,F. Starting from an initial cost matrix in the top left corner, iteration progresses until the optimal assignment is found. It is seen that combining A with F, B with E, and C with D would result smallest total cost. It should be noted that although an example is done with a square matrix algorithm works also with a rectangular matrix. The only requirement for this algorithm version is that there are at least as many columns as rows, a condition that can be easily obtained by rotating the matrix.