UNIVERSITY OF WEST BOHEMIA

Bachelor Thesis

Voice-Interactive Computer Vision on Raspberry Pi

Author:
Martin Adamec

Supervisor:
Ing. Martin Bulín, MSc.

A thesis submitted in fulfillment of the requirements for the degree of Bachelor (Bc.)

in the

Department of Cybernetics

May 22, 2023

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin ADAMEC**
Osobní číslo:	**A19B0336P**
Studijní program:	**B0714A150005 Kybernetika a řídicí technika**
Specializace:	**Umělá inteligence a automatizace**
Téma práce:	**Počítačové vidění s hlasovou interakcí na Raspberry Pi**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1. Nastudujte nejnovější metody v oblasti počítačového vidění, zaměřte se na metody rozpoznávání obličeje a detekce objektů.
2. Vyberte vhodnou metodu rozpoznávání obličeje a upravte ji pro použití na Raspberry Pi za účelem detekce přítomnosti známé tváře.
3. Vyberte vhodný algoritmus pro detekci objektů a upravte jej pro použití na Raspberry Pi.
4. Navrhněte algoritmus pro rozpoznávání nových objektů s využitím hlasového dialogu.
5. Vyvinutou metodu aplikujte na reálnou robotickou entitu.

Rozsah bakalářské práce:	**30-40 stránek A4**
Rozsah grafických prací:	
Forma zpracování bakalářské práce:	**tištěná**

Seznam doporučené literatury:

1. Bishop, C. (1995). Neural networks for pattern recognition. Oxford University Press, USA.
2. Redmon, J., Divvala, S. K., Girshick, R. B., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. CoRR, abs/1506.02640. http://arxiv.org/abs/1506.02640
3. Wang, M., & Deng, W. (2018). Deep Face Recognition: A Survey. CoRR, abs/1804.06655. http://arxiv.org/abs/1804.06655

Vedoucí bakalářské práce:	**Ing. Martin Bulín, M.Sc.** Výzkumný program 1
Datum zadání bakalářské práce:	**17. října 2022**
Termín odevzdání bakalářské práce:	**22. května 2023**

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

L.S.

[illegible] [illegible]

Doc. Ing. Miloš Železný, Ph.D.
děkan

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

Declaration of Authorship

I, Martin ADAMEC, declare that this thesis titled, "Voice-Interactive Computer Vision on Raspberry Pi" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

Date:

"*Remember kids, the only difference between screwing around and science is writing it down.*"

Adam Savage

UNIVERSITY OF WEST BOHEMIA

Abstract

Faculty of Applied Sciences

Department of Cybernetics

Bachelor (Bc.)

Voice-Interactive Computer Vision on Raspberry Pi

by Martin ADAMEC

Artificial neural networks nowadays outperform the "classical" approaches in the area of computer vision by a significant margin, but they come with their own set of problems. Training an artificial neural network is an extremely time-consuming and resource-intensive task (both in terms of necessary hardware and training data), after which the network is able to recognize only a limited collection of classes based on the training data. If a requirement to incorporate a new class into a neural network's recognition capabilities arises, it is necessary to retrain the network, either from scratch, rendering the previously computed weights and biases obsolete, or by using the so-called "transfer learning", an approach based on utilizing the weights and biases obtained from some previous training process, significantly reducing the time and resources needed to achieve the required accuracy of the model. In this bachelor thesis, such concept is utilized in an implementation of a voice dialog system for retraining computer vision models, allowing the user to interactively teach the system to recognize new faces and objects. The effectiveness of the voice dialog system is evaluated through multiple experiments, demonstrating its potential to improve the accuracy and adaptability of computer vision models.

Keywords: Computer vision, Voice interaction, Rapsberry Pi, Human-in-the-loop dialog

ZÁPADOČESKÁ UNIVERZITA

Abstrakt

Fakulta aplikovaných věd

Katedra Kybernetiky

Bakalář (Bc.)

Počítačové vidění s hlasovou interakcí na Raspberry Pi

Autor: Martin ADAMEC

Umělé neuronové sítě dnes v oblasti počítačového vidění výrazně překonávají "klasické" přístupy, ale mají své vlastní problémy. Trénování umělé neuronové sítě je úkol značně náročný na zdroje (jak z hlediska potřebného hardwaru a výpočetního času, tak i z hlediska potřebných trénovacích dat), po kterém je síť schopna na základě trénovacích dat rozpoznat pouze omezený počet tříd. Vznikne-li požadavek na začlenění nové třídy do rozpoznávacích schopností neuronové sítě, je nutné síť přetrénovat, a to buď od začátku, čímž se dříve vypočtené váhy a prahy stanou irelevantními, nebo pomocí takzvaného "transfer learningu", což je přístup založený na využití vah a prahů získaných z nějakého předchozího trénování sítě, čímž se výrazně zredukuje čas a zdroje potřebné k dosažení požadované přesnosti modelu. V této bakalářské práci je takový koncept využit při implementaci hlasového dialogového systému pro přetrénovávání modelů počítačového vidění, který umožňuje uživateli interaktivně učit systém rozpoznávat nové tváře a objekty. Přesnost hlasového dialogového systému je vyhodnocena prostřednictvím několika experimentů, které prokazují jeho potenciál pro zlepšení přesnosti a adaptability modelů počítačového vidění.

Klíčová slova: Počítačové vidění, Hlasová interakce, Rapsberry Pi, Human-in-the-loop dialog

Acknowledgements

I would like to express my gratitude to my supervisor Ing. Martin Bulín, MSc. for his valuable advice, and for being a great mentor throughout the writing of my bachelor thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the area of computer vision today, the state-of-the-art approach is to use artificial neural networks, which have been proven to yield significantly better results than classical methods of computer vision (classical methods refer to computer vision approaches that do not use artificial neural networks). Such methods can still be useful in cases where the size of the available collection of training data is not sufficient (Sertis, 2022), as a large amount of training data is crucial to ensure the correct results when working with artificial neural networks.

The vast majority of those artificial neural networks mentioned above are trained using the so-called *offline learning*, meaning that during the *training phase*, the artificial neural network is trained on a data set of sufficient size (in case of image classification, it is common for the number of images representing each class to range from tens of thousands to millions of individual images). After the training phase of the artificial neural network is finished, the network then enters the *evaluation phase*, in which the coefficients (*weights* and *biases*) of the network are no longer modified.

Generally speaking, this method of training artificial neural networks without any previous parameter initialization (training from scratch) is very resource-intensive, whether in terms of computational time, finances, or human labor. Although nowadays most of the training is done on computers using graphics cards (*GPU*) due to their ability to do many computations in parallel (Towards AI, 2021), the training phase is still an extremely time-consuming process, and the computational time needed to successfully train a deep neural network (artificial neural network with more than one hidden layer) can in some cases reach several weeks. The purchase of a graphics card also means additional costs, as the prices of higher-end graphics cards are in the tens of thousands of crowns (hundreds to thousands of dollars), and their use also increases the overall power consumption of the computer. However, training an artificial neural network is not only costly in terms of the hardware required for the computation itself but also in terms of data preparation (DataSet SHOP, 2023), since to achieve, for example, sufficient accuracy in detecting a human face in an image, the artificial neural network needs to be trained on a very large number of training images in which human faces are photographed from as many different angles as possible, in multiple different resolutions, and with as many kinds of *data augmentation* (adding noise to the image, rotating the image, cropping the image, etc.) as possible.

At the same time, this approach is suitable only for tasks with a static dataset, i.e. for tasks where the neural network is initially trained once on an unchanging number of classes (groups of entities with similar properties) into which it then classifies objects throughout the evaluation phase. As an example of practical use of this approach, we can use the task of automatic detection of bone fracture in an X-ray image (Hardalaç et al., 2022), since these tissue damages generally have specific characteristics by which they can be reliably recognized in the image. In such a case, after the initial training on a substantial training dataset, the network can be easily used in the evaluation phase, because no significant changes in the monitored properties are expected. However, for tasks where the dataset is expected to change repeatedly, it would not be viable to go through the resource-demanding process of training the whole network from scratch, as the dataset could change before the training is even finished. Therefore in such cases, a different, faster, and less resource-intensive approach needs to be used. Possibly the best approach for tasks with dynamically changing datasets is the so-called *online* (or incremental) *learning*, in which the model parameters are not trained from a random initialization, but parameters from an already trained model are used instead, and new data are presented to the model in small batches, which dramatically shortens both the computational time and the size of the dataset needed for the training.

To illustrate the power of online learning and its benefits over the standard offline approach, this thesis presents an implementation of an interactive voice dialog system, which allows the user to interact with the computer vision models (face recognition and object detection) through voice commands and receive real-time feedback. This system provides users with the ability to correct classification errors and to teach the system new to recognize new faces or objects that it has never seen before.

The contribution of this thesis to the field of machine learning is in providing a proof of concept for an interactive system, exploring the possibilities of retraining neural networks, and modifying the performance of computer vision algorithms after they have been trained. The findings of this research could potentially lead to a low-cost handheld device, capable of facilitating object recognition for visually impaired people.

1.1 State of the Art

While the tasks of face recognition and object detection both fall in the area of computer vision, different approaches have proven to achieve the best results for each of those tasks. The topic of human-interactive dialog is also a topic of high interest for research, as new possibilities of human interaction (speech, sign language, facial expression, etc.) are becoming feasible with newly discovered technologies.

1.1.1 Face Recognition

Face recognition is one of the most researched tasks in the field of computer vision, as it can be utilized in security and surveillance applications. At the time of writing, the five best methods listed on the popular machine learning website *Papers With Code* [1] achieve accuracy of over 99% on the *Labeled Faces in the Wild* dataset (Section 4.1), which is arguably the most popular dataset in the field of face recognition. Each of those methods is based on deep artificial neural networks, differing only in the specific architecture of those networks.

1.1.2 Object Detection

The term *object detection* refers to the task of determining the location of an object in an image, and then assigning a label to the localized object. Nowadays, object detection is being dynamically developed in the automotive industry in the field of autonomous vehicles, where an automatic understanding of the vehicle's surroundings is essential to achieve fully autonomous driving. The ultimate main goal in the area of object detection is to find the best trade-off between inference time and accuracy in order to achieve usable results in real time. Currently, the best real-time object detection model is the *YOLOv7* (Wang, Bochkovskiy, and Liao, 2022) from the YOLO (You Only Look Once) family of single-shot object detection models, achieving average precision of 51.2% on the *COCO* dataset (Section 4.3) while achieving a very impressive inference time of only 9 milliseconds on the *NVIDIA Tesla T4*[2] GPU .

1.1.3 Human-in-the-loop methods

Human-in-the-loop refers to a special kind of interaction between humans and machine learning algorithms. Generally speaking, human-in-the-loop methods could be split into two categories depending on which side is in charge of the interaction (Mosqueira-Rey et al., 2022):

- *Active learning* - the system is in control of the interaction, treating the user as a means to annotate unlabeled data for its machine learning algorithms
- *Machine teaching* - the user is in control of the interaction, choosing what kind of data and knowledge they want to teach the machine learning algorithms of the system

[1] `https://paperswithcode.com/sota/face-recognition-on-lfw`
[2] `https://www.nvidia.com/en-us/data-center/tesla-t4/`

The second group of methods is relevant for this bachelor thesis, as the designed interactive system should work by responding to the stimuli from the user. Such system should have a short response time (ideally working in real-time), and the machine learning models should be updated incrementally, building on top of previously obtained knowledge.

1.2 Thesis Objectives

The objectives of this thesis are:

1. to select a face recognition algorithm suitable for Raspberry Pi
2. to select an object recognition algorithm suitable for Raspberry Pi
3. to implement those algorithms on low-cost robotic platform
4. to incorporate those algorithms into a voice-interactive dialog loop

1.3 Thesis Outline

This thesis consists of 8 chapters, following the standard skeleton of scientific publications.

Chapter 2 describes the general concepts of technologies utilized in this bachelor thesis, first focusing on the concept of artificial neural networks and their applications in the area of computer vision (Section 2.1), then moving on the tools used for the tasks of face recognition (Section 2.2) and object detection (Section 2.3) and the concept of *transfer learning* (Section 2.4), which is heavily utilized in the final implementation of the solution. The rest of Chapter 2 then explains the technologies used in the more technical aspects of the final implementation, namely the *communication protocols* used for the communication with the remote server, the *Robot Operating System*, on which the interactive system (and the physical robotic entity) runs, and the *SpeechCloud* platform used for the speech processing in the interactive dialog.

Chapter 3 then details the application of those technologies in the solution presented in this bachelor thesis, while Chapter 4 briefly describes the datasets used either for training or testing the models used in the final implementation.

Chapter 5 presents the results of experiments done to justify the final design choices, and also for the evaluation of the implementation, and the application of the proposed methods on a physical entity called *Robot.v1* is discussed in Chapter 6.

Chapter 7 comes with a discussion of the results, along with suggestions for future work and the alternative approaches tested during the design process. Finally, the thesis is concluded in Chapter 8.

Chapter 2

Tools and technologies

This chapter describes the working principles of technologies used in this bachelor thesis. The technologies are described in such depth that the reader is able to understand how they were utilized in this work.

2.1 Artificial neural network

Artificial neural network (ANN) is a type of machine learning (ML) model inspired by the structure and the operating principle of the human brain. An artificial neural network is comprised of multiple interconnected nodes, called artificial neurons.

An artificial neuron is essentially a simplified model of the biological neuron, having one or more inputs, which are then weighted and summed. The product of this operation is called neuron *activation*, and the formula to calculate its value can be seen in Eq. (2.1).

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.1}$$

where:

- z: neuron activation value
- x_i: i^{th} input value of the neuron
- w_i: i^{th} input weight of the neuron
- b: bias (threshold) of the neuron

Because the activation value is arbitrary and unbounded, using this value without any kind of normalization could result in numerical instability, as the numerical values could raise above the maximum range that computers are able to store in memory (overflow errors) during computation. To achieve such normalization, an *activation function* has to be used. By applying an activation function as in Eq. (2.2), the value of the neuron activation is mapped into a finite interval whose limits depend on the activation function used. The value obtained from the activation function is called neuron *activity*.

$$a = f(z) \tag{2.2}$$

where:

- a: neuron activity
- z: neuron activation value
- f: neuron activation function

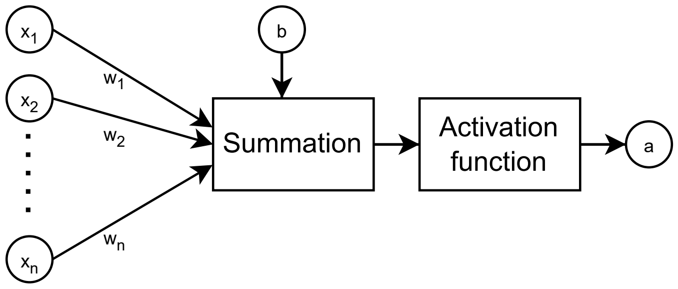


FIGURE 2.1: Artificial neuron (perceptron) working principle

Artificial neurons are then organized into layers, which are essentially groups of neurons with the same designated function. The first layer of any artificial neural network is called an *input layer*, which serves as an entry point for the network, through which it receives the data to process. The input layer is usually a *fully-connected layer*, meaning it passes the inputs to each neuron of the next layer. The last layer of an artificial neural network is called the *output layer*, representing the result of the network processing. In the context of classification, when the task involves distinguishing between only two classes, the output layer may comprise of only a singular neuron with a binary activation function. The activity of the neuron would then indicate whether the processed input belongs to the first class (activation function has a value of 1) or the second class (activation function has a value of 0). For the case of multi-label classification, the number of neurons in the output layer corresponds to the number of classes in the classification problem. In these cases, so-called *one-hot vector* encoding could be used, where the neuron representing the target class has a value of one, whereas the value of every other neuron is zero. Another commonly used type of encoding is obtained by using a special activation function in the output layer, called the *softmax* function. This function converts all of the N output values into a probability distribution with N possible outcomes, which normalizes the output of the network, as the sum of all the probabilities is equal to one. The softmax probability of the i^{th} neuron can be obtained by using the formula in Eq. (2.3)

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{2.3}$$

Any layer between the first (input) and last (output) layer is called a *hidden layer*. If the network has more than one hidden layer, it is referred to as a *deep neural network* (*DNN*). An example of a deep neural network is shown in Fig. 2.2. Convolutional neural networks also fall into the category of deep neural networks, and they are further described in Section 2.1.1.

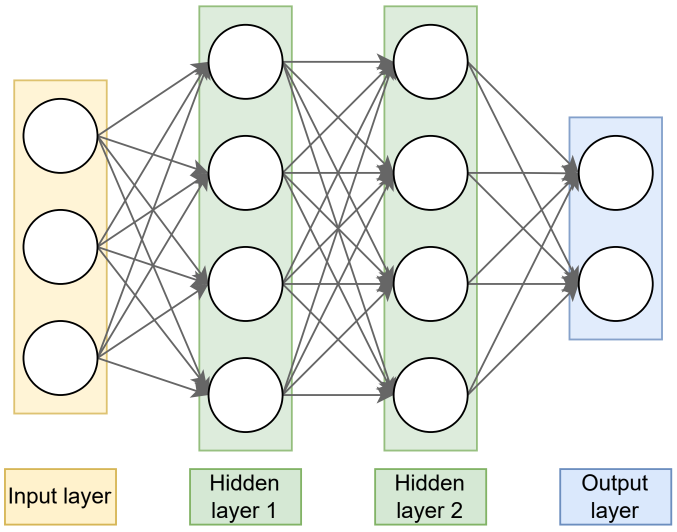


FIGURE 2.2: Block diagram of a fully-connected deep neural network with two hidden layers

The hidden layers of a neural network allow the network to learn dependencies and patterns present in the dataset, which allows the network to solve complex tasks without the need for an explicit algorithm programmed directly for the task.

As mentioned in Chapter 1, a neural network can either be in a *training phase*, or in a *evaluation phase*. In the training phase, the network is presented with labeled training data, meaning that for each input value, the desired output value is known (this type of training is called *supervised learning*). The network takes the input value and calculates the output based on its current parameters (weights and biases), after which it calculates the error between the predicted and the correct output from the labeled training data. The error is calculated according to the so-called *loss function*, which is determined based on the nature of the problem, whether it is a *classification* or *regression* task (a simple comparison is shown in Fig. 2.3. These two fundamental machine learning tasks differ in their objectives. While the goal of regression is to predict a continuous numerical value based on input features (e.g. predicting the temperature for the next day), the goal of classification is to assign the input to a given category based on its features (e.g. labeling an email as either spam or not).

One of the most commonly used loss functions for regression problems is the *mean squared error* (*MSE*), which is calculated by the formula in Eq. (2.4).

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.4}$$

where

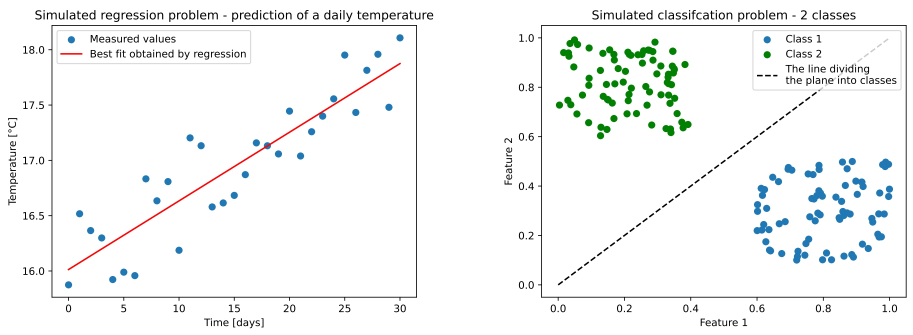
- N = number of samples in the output y
- y_i = correct value of the i^{th} element of the output
- $\hat{y}_i$ = predicted value of the i^{th} element of the output
- i = index of the output element

For the problem of classification, the most commonly used loss function is *cross-entropy*, which is more suitable for outputs with a probabilistic distribution. The formula to calculate cross-entropy is shown in Eq. (2.5).

$$H(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{2.5}$$

where

- N = number of samples
- y_i = correct value of the i^{th} element of the output
- $\hat{y}_i$ = predicted value of the i^{th} element of the output
- i = index of the output element



(A) Artificial regression task (B) Artificial classification task

FIGURE 2.3: Demonstration of regression and classification problems on artificial data

The goal of the training phase is to find a set of weights and biases that minimizes the loss function. Due to the high dimension of the loss function, especially in deep neural networks, it is essentially not possible to find the minimum of the loss function analytically. Because of that, the network iteratively adjusts its trainable parameters, *weights* and *biases* (see Eq. (2.2) for their meaning in artificial neuron activation). The magnitude of those adjustments is determined by the negative gradient of the loss function. The gradient is a vector of all partial derivatives, and the value of a gradient at any given point of the functions the gradient shows the direction of the steepest increase (the negative gradient therefore shows the direction of the steepest decrease). This method of finding the minimum of the loss function is called the *gradient descent*. One can imagine the working principle of this algorithm as starting at a random point on the loss function (random initialization of weights and biases), calculating the negative gradient at that point, and finally taking a small step along the loss function in the direction of the negative gradient. The simplification of the gradient descent algorithm is displayed in Fig. 2.4. However, because the gradient descent always takes a step in the direction of the steepest decrease, it can end up prematurely when it encounters a local minimum, and so finding the optimal solution is not always guaranteed. Such a case is displayed in Fig. 2.5, where w_L denotes the local minima that the gradient descent algorithm could end in before reaching the optimal weight value of w^*.

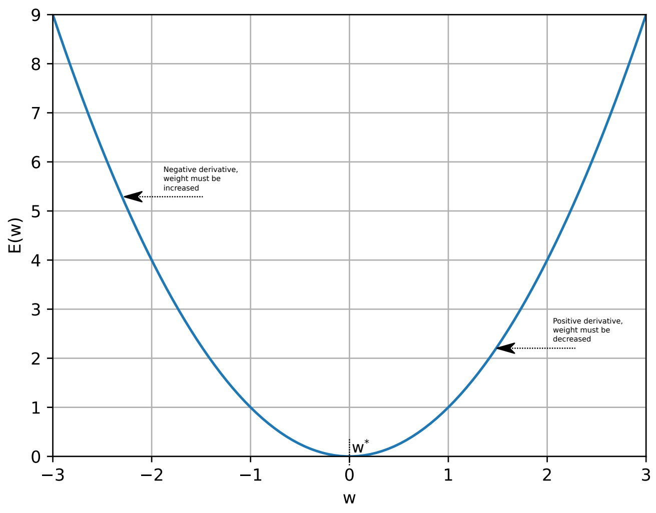


FIGURE 2.4: Illustration of the dependency of the loss function ($L(w)$) on a single weight w - single global minimum, w^* denotes the optimal value of the weight w

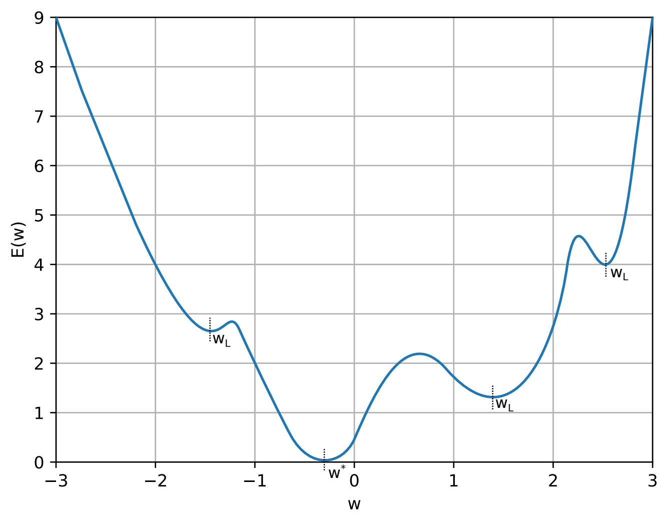


FIGURE 2.5: Illustration of the dependency of the loss function ($L(w)$) on a single weight w - multiple local minima, w^* denotes the optimal value of the weight w, while each w_L denotes a local minimum which the gradient descent algorithm could end up in

In general, neural networks should be used in tasks where an explicit solution is either not known, or where it would be too complicated to produce an exact algorithm. However, in cases where the explicit solution of the problem is known, such as linear regression problems, it is generally better to use the explicit solution due to the extensive computational cost of training a neural network.

2.1.1 Convolutional neural network

Convolutional neural networks (CNN) are artificial neural networks that contain at least one *convolutional layer*, which is a special kind of layer with a trainable convolutional filter, called a *kernel.* Because of the advantages listed below, convolutional neural networks have become state-of-the-art in digital image processing and computer vision applications since their first introduction in 1998, when they were used to recognize handwritten digits on checks in US banks (Lecun et al., 1998), to today, when transformer architecture networks (Section 2.1.2) are beginning to gain traction in the area of computer vision. In order to fully explain the convolutional neural network, it is first necessary to explain the operation of *convolution* itself. Since only discrete convolution is relevant for this thesis, the continuous version of the operation will not be explained in detail, and the term "convolution" will hereafter refer to the discrete version when used. Convolution is a mathematical operation on two number sequences, denoted $f(n)$ and $h(n)$, which produces another number sequence, denoted $g(n)$, where each element is a sum of the element-wise product of both original sequences, as shown in Eq. (2.6).

$$g(n) = f(n) * h(n) = \sum_{k=-\infty}^{\infty} f(k) \cdot h(n-k) \tag{2.6}$$

The intuition behind the process of convolving two sequences could be described as:

1. Having two number sequences ($f(n)$ and $h(n)$), reverse one in time ($f(n-k)$)
2. Slide one of the sequences (h) over the other (f)
3. For each slide (k), compute the element-wise product of the overlapping values
4. Add up the product ($\sum$)
5. Store the sum as an element of a new sequence on the corresponding index (k)

For better demonstration, the following example is provided in Fig. 2.6, in which two short arrays of numbers, f and h are convolved according to the formula in Eq. (2.6). For the parts of the arrays that do not overlap, one can imagine that the array simply has a value of zero at those indices, as illustrated by the red zeros in the image.

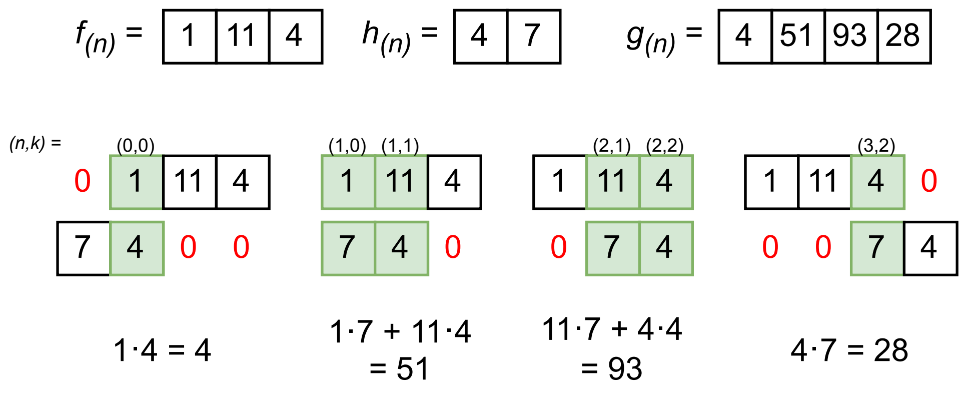


FIGURE 2.6: Demonstration of how a one-dimensional convolution of two short arrays works

However, in computer vision, two-dimensional convolution is used for most applications, as it is best suited for single-image processing. Images are basically matrices of pixel values - for grayscale images, pixel values are stored directly in a single two-dimensional array (Fig. 2.7), for colored images, the pixel values are stored in a three-dimensional array, where a value for each of the color channels, *R*ed, *G*reen and *B*lue (*RGB* for short), is stored in a separate page of the matrix (Fig. 2.8). An example of how a 3D matrix is structured as shown in Fig. 2.9. In some special applications, such as analysis of volumetric (three-dimensional) images or video analysis, where it can be used to analyze the motion of an object over time, three-dimensional convolution is used. However, since only the two-dimensional version is utilized in this thesis, it is explained below.

The general formula of discrete two-dimensional convolution is given in Eq. (2.7),

$$g(i,j) = f * h = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(i-m, j-n) \cdot h(m,n) \tag{2.7}$$

where

- g = product of the convolution
- f = original two-dimensional matrix
- h = convolution kernel
- i, j, m, n = matrix indices, $f(i,j)$ represents the value of the pixel at row i and column j in the matrix f

but since digital images have a finite size and are represented as arrays in computers (indexing from zero is assumed in this case), the equation can be rewritten to the form given in Eq. (2.8)

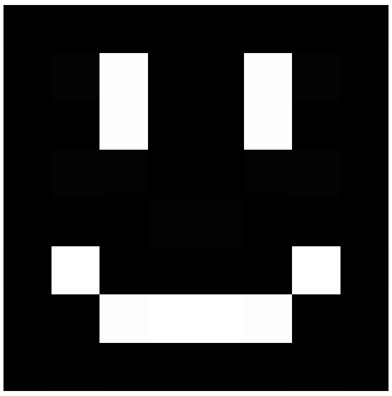
$$g(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(i-m, j-n) \cdot h(m,n) \tag{2.8}$$

where

- g = convolved image
- f = original image
- h = convolution kernel
- i, j, m, n = array indices
- M, N = size (height and width) of the convolution kernel h

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 255 & 0 & 0 & 0 & 0 & 255 & 0 \\ 0 & 0 & 255 & 255 & 255 & 255 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(A) The two-dimensional array representing the image in a computer



(B) The same array when the values are represented as pixel values (intensity)

FIGURE 2.7: An illustration of an 8x8 grayscale image representation in a computer (values are represented by unsigned 8-bit integers)

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 255 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 255 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

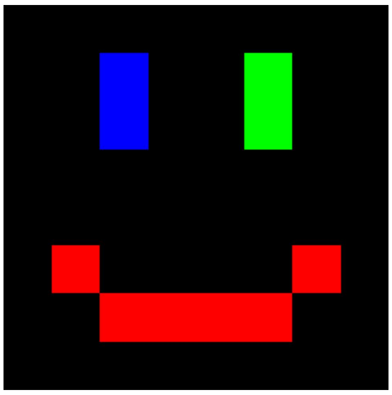
(A) The three-dimensional array representing the image in a computer - Blue channel

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 255 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 255 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

(B) The three-dimensional array representing the image in a computer - Green channel

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 255 & 0 & 0 & 0 & 0 & 255 & 0 \\
0 & 0 & 255 & 255 & 255 & 255 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

(C) The three-dimensional array representing the image in a computer - Red channel



(D) The same array when the values are represented as pixel values

FIGURE 2.8: An illustration of an RGB image representation in a computer (pixel values are represented by unsigned 8-bit integers)

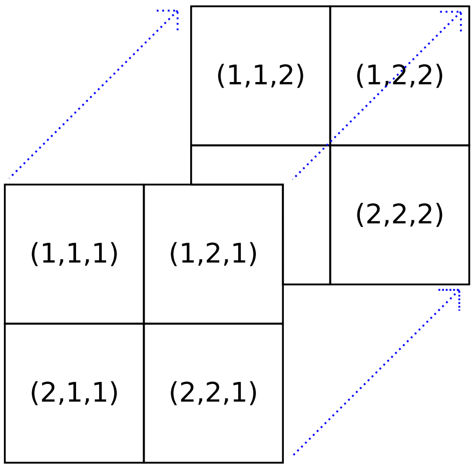
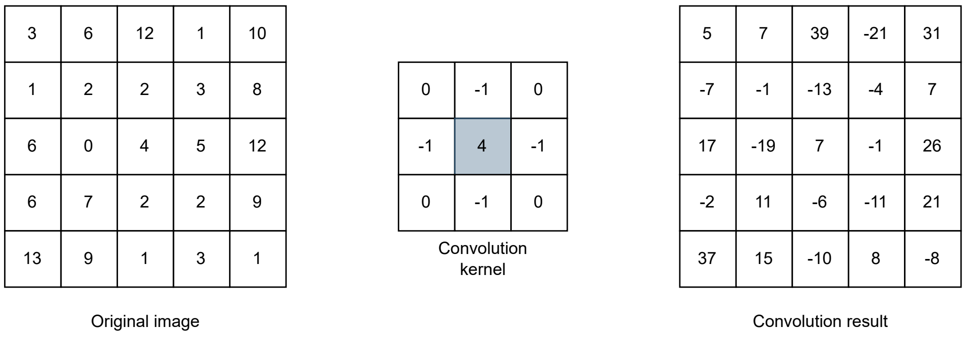


FIGURE 2.9: Illustration of a three-dimensional matrix with indexing in the format (row, column, page)

To better understand the working principle described by the formula, the illustration in Fig. 2.10 is provided. One can imagine the algorithm of the two-dimensional discrete convolution as taking the convolution kernel and gradually shifting the kernel (colored square with value "4" in the image) center over each pixel in the original image. Doing so, for each pixel in the original image, a region of overlapping points is formed. In this region, each value from the original image is multiplied with a corresponding value from the convolution kernel, and all of those values get summed up. The resulting sum is then stored in a copy of the original image, at the same position that the pixel had in the original image. Because the convolution kernel would reach out of bounds on the edge pixels of the original pixels, any element (pixel) beyond the borders of the original image is considered to have a zero value, which cancels out its effect on the resulting sum.



3	6	12	1	10
1	2	2	3	8
6	0	4	5	12
6	7	2	2	9
13	9	1	3	1

Original image

0	-1	0
-1	4	-1
0	-1	0

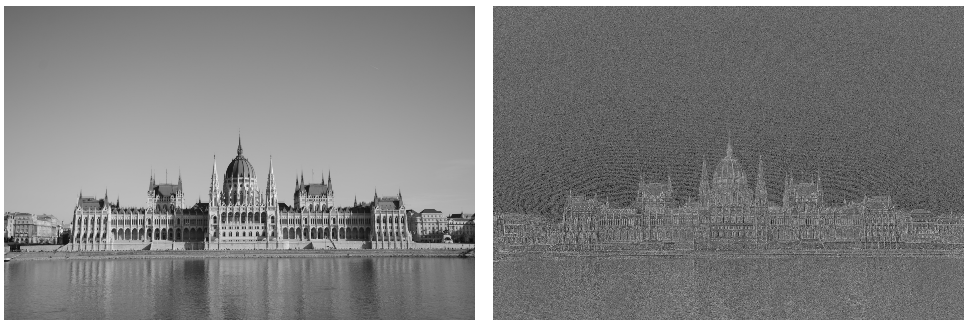
Convolution kernel

5	7	39	-21	31
-7	-1	-13	-4	7
17	-19	7	-1	26
-2	11	-6	-11	21
37	15	-10	8	-8

Convolution result

FIGURE 2.10: Demonstration of how a two-dimensional convolution with a Laplace kernel works

As can be seen in Fig. 6.1, convolution is used to extract important features in the processed image (edges in the case of the Laplace kernel, as it is a gradient approximation).



(A) A photo of the Hungarian Parliament Building (Országház) in Budapest

(B) The same photo after applying a Laplace kernel convolution to it

FIGURE 2.11: Demonstration of the usage of two-dimensional convolution on a real image (Laplace kernel)

In convolutional neural networks, the values of the convolution kernels are not given as a part of the network architecture, but rather are obtained as a result of the network training, where the network modifies the parameters in such a way that the convolutional layers extract the features that the network evaluates as relevant for classification, such as edges (extracted by layers closer to the input layer), curves and other, more complex patterns (extracted by the layers deeper in the network architecture). The weights (values for each pixel) are typically shared among the whole convolutional layer of the network, which results in a reduction of the network's trainable parameters, which significantly lowers the computational intensity of the network training, and it can also help prevent overfitting. The parameter reduction can be illustrated by a comparison of a classical feed-forward network and a convolutional network on the task of classifying 32x32 images. Given that each image is comprised of 1024 pixels, the input layer would have $1024 \cdot \psi$ trainable parameters, with ψ being the number of neurons in the first hidden layer. Contrary to that, a convolutional layer with a 5x5 convolution kernel would only have 26 parameters (25 weights and 1 bias), as the weights of the kernel are shared among the whole layer.

Convolutional networks also usually make use of one or more *pooling* layers. Pooling layers are used to reduce the dimension of the feature maps produced by the convolutional layers. This is achieved by dividing the original input (feature map) into non-overlapping square regions, and by only retaining the most significant value from each area in the new, reduced feature map. The most commonly used type of pooling is the *Max-Pooling*, where the retained value is the largest value from the area, as is shown in Fig. 2.12. The other commonly used type of pooling is the *Average-Pooling*, where the retained value is the average value from the whole area. The dimension reduction achieved by pooling can help to reduce the number of parameters needed to train the network, which speeds up the training process, as fewer operations need to be computed during the training phase, and it can also help to make the network more robust to small changes in the input values and to overfitting, as only the most significant value from each area is used during the training. Unlike the convolution kernel values, the pooling layers usually have fixed parameters set by the author of the network as a part of the network architecture.

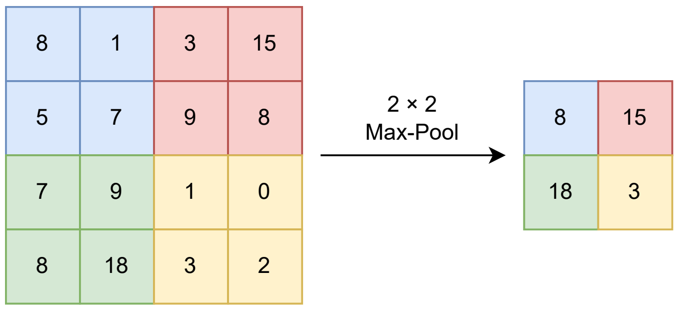


FIGURE 2.12: Demonstration of how a 2x2 Max-Pooling works

2.1.2 Transformer architecture

Transformer architecture of artificial neural networks was first proposed in the paper "Attention Is All You Need" (Vaswani et al., 2017) from the Google Brain team, and was originally designed for machine translation. In general, transformers are well-suited for processing sequential data, making them the current state-of-the-art approach in the area of natural language processing (NLP), as can be seen in the (at the time of writing) arguably most popular example of *ChatGPT*, which is based on the GPT (Generative Pre-trained Transformer) 3.5 (Brown et al., 2020). A huge benefit of using the transformer architecture is its ability to process input data in parallel with the ability to process a value infinitely far back in the input sequence.

The key concept of the transformer architecture is its so-called *self-attention* mechanism, a *sequence-to-sequence* (meaning the input, as well as the output, is a sequence of vectors, also called *tokens*) algorithm that allows the network to dynamically weigh different parts of the input based on their importance for the current task, such as the most important words (or phrases) in a sentence in natural language processing. The working principle of a self-attention layer can be described in the following steps:

1. Get the embedding vector for each element of the input
2. Compute a *query*, *key* and *value* vectors by applying linear transformations (multiplying with the respective weight matrices) to the input token, as shown in Eq. (2.9).
3. Compute the similarity score for the token (dot product of the query vector with the key vector), as shown in Eq. (2.10)
4. Normalize the scores using a softmax function (Eq. (2.3)), which assigns a weight to each element based on its relevance to the other elements, as shown in Eq. (2.11)
5. Based on the obtained weights, compute a weighted sum of the value vectors (Eq. (2.12)), where the weights are given by the normalized scores, yielding the output of the self-attention layer, which can be further utilized, usually as an input to the next layer of the neural network

$$\begin{aligned} q_i &= W_q \cdot x_i \\ k_i &= W_k \cdot x_i \\ v_i &= W_v \cdot x_i \end{aligned} \tag{2.9}$$

where

- q_i, k_i, v_i = i^{th} *query*, *key* and *value* vector, respectively
- $x_i = i^{th}$ embedding vector of the input sequence
- W_q, W_k, W_v = weight matrices for *query*, *key* and *value* vector transformations, respectively
- i = index

$$w'_{ij} = q_i^T \cdot k_j \tag{2.10}$$

where

- w'_{ij} = element of a weight matrix before normalization
- q_i = i^{th} query vector
- k_j = j^{th} key vector
- i, j = vector indices

$$w_{ij} = \text{softmax}(w'_{ij}) \tag{2.11}$$

$$y_i = \sum_j w_{ij} \cdot v_j \tag{2.12}$$

where

- w_{ij} = element of a normalized weight matrix
- y_i = i^{th} vector of the output sequence
- v_j = j^{th} value vector
- i, j = vector indices

For better understanding of the meaning of the key, query and value vectors, an illustration is provided in Fig. 2.13.

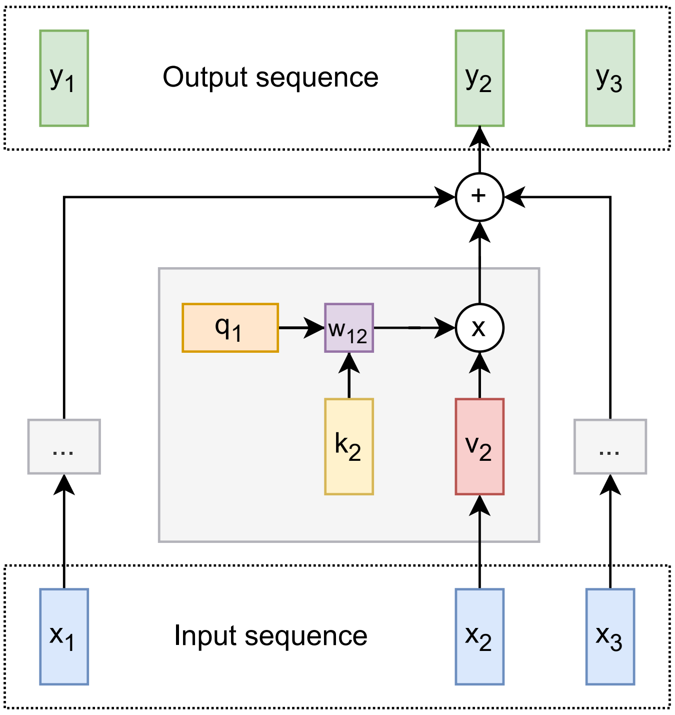


FIGURE 2.13: Illustration of how the *key*, *query* and *value* vectors are utilized in the *attention* computation

A more detailed explanation of the transformer architecture can be found at (Bloem, 2019).

Because transformer architecture has proven to be very powerful in the area of NLP, the possibility of applying transformer architecture to computer vision tasks is also being explored nowadays. The result of the research in this are the so-called *Vision Transformers* (*ViT*), originally introduced in (Dosovitskiy et al., 2021).

Vision Transformers split the input image into patches of fixed size (16 by 16 pixels in the original paper). Each of those patches is then squeezed into a one-dimensional array, resulting in a vector from which the embedding is computed. This embedding is then passed into a transformer structure as described earlier in this section. Because the transformer architecture can work with very long (theoretically infinite) input sequence, the network is able to capture global dependencies in the input image, which can improve the network's ability to understand the scene and objects in the scene (if there is a hand near a face in the image, it is more probable that there will also be a bottle or a cup in the image). This is an improvement over convolutional neural networks, as those can capture local dependencies very well, but can struggle with capturing dependencies over large regions of the input image. Vision Transformers also utilize *positional encodings*, which help the transformer architecture retain the important information about the position of the input vector in the input sequence (transformers normally process the whole sequence in parallel regardless of the original order, hence why the positional information has to be added explicitly).

2.2 Face recognition framework

For the task of face recognition, an open-source Python library called *face_recognition* [1] was chosen for its simple API, which makes it extremely easy to use in the project. The face_recognition library is built on top of the *dlib* [2] library, which is a machine learning library written in C++. Pretrained models from the dlib library are used for computationally expensive tasks such as the extraction of facial features from images, as C++ is by design faster (and therefore more suitable for such tasks) than Python, and the face_recognition library is essentially a high-level Python API for the high-performance algorithms.

Because the goal of this bachelor thesis was to implement the face recognition algorithm in a human-interactive dialog loop on the robotic entity, it was necessary for the user to be able to add a new face to the list of known faces interactively. The face_recognition library allows this level of interaction, because each of the known faces is stored in a form of a *face encoding*, which is a vector whose each component represents a location of one of the facial features.

[1]`https://github.com/ageitgey/face_recognition`
[2]`https://github.com/davisking/dlib`

The size of this vector varies based on the model used for the feature extraction:

- model "large" - vector of 68 points (see Fig. 2.14)
- model "small" - vector of 5 points (labeled 36, 39, 42, 45 and 33 in Fig. 2.14)

The smaller model reduces the computation time, but at the cost of reduced accuracy of the face recognition itself. The larger model is used in the implementation in this bachelor thesis.

For checking if any face is present in the source image, two face-detection algorithms from the library can be utilized:

1. convolutional neural network (Section 2.1.1)
2. classical-approach called *Histogram of oriented gradients* (*HOG*)

Although the method using a convolutional neural network achieves a better recognition accuracy and is more robust to changes in the face angle and rotation, the *HOG* method has a lower inference time when running solely on CPU (without GPU acceleration), and it was therefore chosen for the implementation in this bachelor thesis, as it allows a real-time inference on the limited hardware of the robotic entity. Histogram of oriented gradients works by dividing the image into multiple smaller regions (usually 8 by 8 pixels) and computing intensity gradients and orientations in each of those regions (essentially a magnitude and direction of edges in the region). After this computation, the values are normalized and histograms are generated for each of the regions. By concatenating all the histograms, the feature vector is obtained.

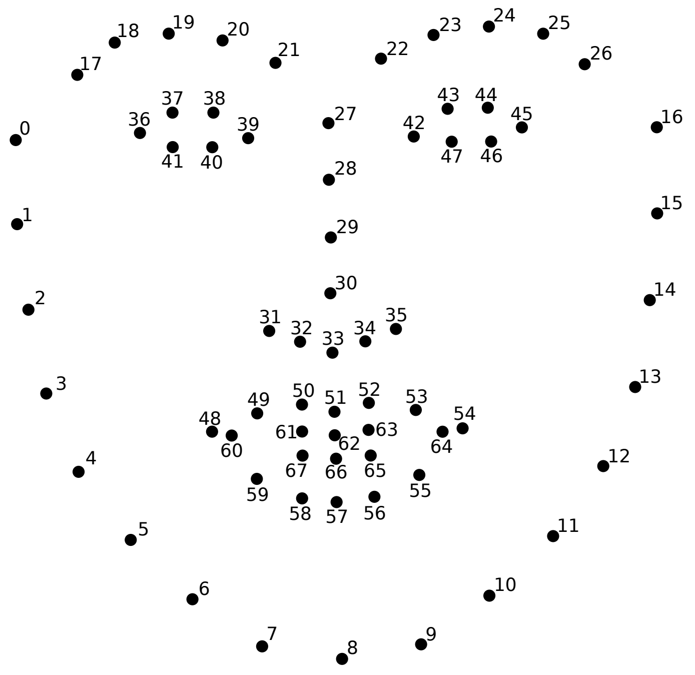


FIGURE 2.14: The 68 points representing the face landmarks locations

The simplified process of getting a face recognition result from the source image is shown in a flowchart in Fig. 2.15.

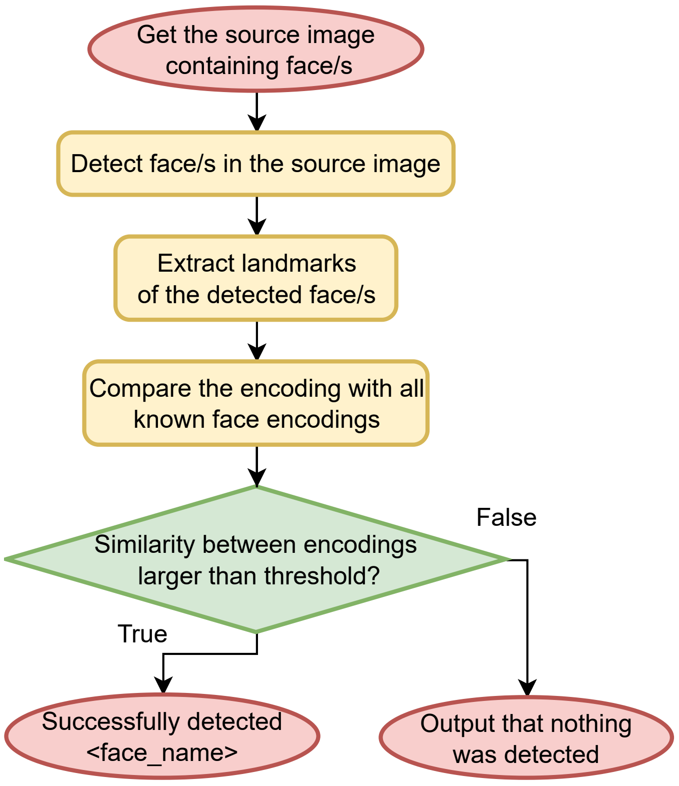

Get the source image containing face/s
Detect face/s in the source image
Extract landmarks of the detected face/s
Compare the encoding with all known face encodings
Similarity between encodings larger than threshold?
False
True
Successfully detected <face_name>
Output that nothing was detected

FIGURE 2.15: A flowchart representing the process of getting the face recognition result from the source image

2.3 Object detection

First, it should be explained what the term *object detection* means. Object detection is a computer vision task that aims to find an object within a source image and classify it into one of the n known classes. Thus, it could be said that the object detection task consists of two subtasks:

1. *object localization* = finding the position of the object in the source image
2. *object classification* = assigning the object into one of the n known classes

Nowadays, this is commonly achieved by using artificial neural networks, which are trained to find a so-called *bounding box* around the object, which is a rectangle that delimits the area containing the object in the image. Object detection can then simply be seen as taking the input image, using the object localization algorithm to find the bounding box around the object, and then passing the region within the bounding box to the object recognition algorithm, which determines which class the object belongs to.

MobileNetV3

As the main implementation of the object detection algorithm in this bachelor thesis, the latest network architecture from the *MobileNet* family of efficient convolutional neural networks was chosen, namely the *MobileNetV3 Large* version (Howard et al., 2019). The whole family of MobileNet networks is designed to run efficiently on embedded devices, such as smartphones or single-board computers, like Raspberry Pi. MobileNetV3 architecture improves the accuracy in comparison to the previous version by using a *Squeeze-and-Excitation* (*SE*) blocks (Hu et al., 2017) in the design of the network, which allow the network to better capture the interdependencies between the image channels. SE blocks work by first *squeezing* each channel into a single numerical value by using global average pooling (explained in Section 2.1.1). This numerical value is then passed to an auxiliary two-layer network, which produces the weight assigned to each channel of the image when processed by the main neural network.

To reduce the computational cost of inference, MobileNetV3 uses a specialized neuron activation function, which was specifically designed with the limited processing power of mobile devices in comparison to desktop computers. This activation function is called *hard-swish* (shown in Fig. 2.16), and is based on the ReLU (*R*ectified *L*inear *U*nit) activation function. The formula for the hard-swish function is as follows:

$$f(z) = \text{h-swish}(z) = z \cdot \frac{\text{ReLU6}(z+3)}{6} \tag{2.13}$$

The ReLU6 activation function is a modified version of the widely used ReLU activation function, the formulas for both of those functions are as follows:

$$f(z) = \text{ReLU}(z) = \max(z, 0) \tag{2.14}$$

$$f(z) = \text{ReLU6}(z) = \min(\max(z, 0), 6) \tag{2.15}$$

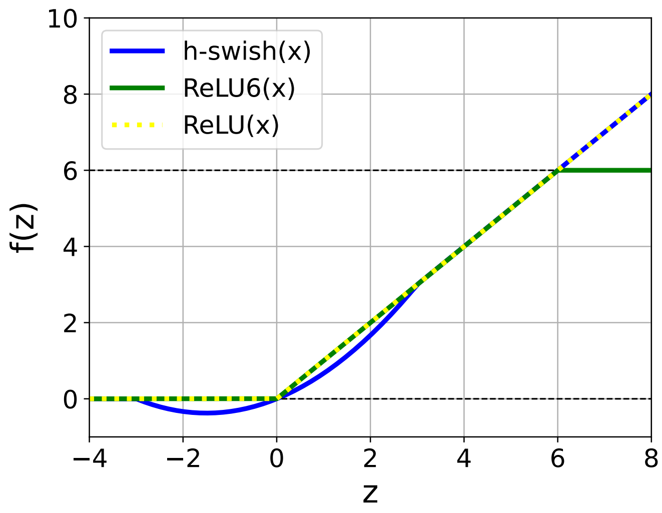


FIGURE 2.16: Neuron activation functions: *ReLU*, *ReLU6* and *Hard-swish*

YOLOS tiny

The YOLOS (You Only Look at One Sequence) network architecture, first introduced in (Fang et al., 2021), is a modification of the original Vision Transformer architecture, in which it replaces the classification output token by a hundred randomly initialized output tokens for object detection, which allows the network to predict bounding boxes of the object in the image as well as the class of the object.

YOLOS Tiny is the smallest version of the model proposed in the original paper. It has the lowest prediction accuracy of all the introduced versions, but it is also the least computationally expensive one, making it the best choice for using it in the implementation on the physical robotic entity with its limited computational resources.

Image embedding extractor + Classifier

During the design phase, the possibility of using the output of the pre-trained neural network as an embedding of the input image and then using a classifier on the embeddings was considered. This approach is similar to the *frozen feature extractor* solution discussed in (Section 2.4), but the final architecture could be seen as taking the concept a step further - while the frozen feature extractor approach only replaces the output layer of the whole network, the second concept splits the whole model into two separate networks, with the first one being the pre-trained network without the last layer, which produces an embedding from the input image, and the second being any type of classifier, which assigns labels to the embeddings produced by the first network.

2.4 Transfer learning

First of all, it should be specified that in the scientific community, there are sometimes slightly different definitions of this term. In the context of this bachelor thesis, the term transfer learning represents a machine learning technique where a model (in our case an artificial neural network) trained for one task is reused (either as is or in a modified form) for a different but principally similar task. In transfer learning, instead of building a new network from scratch with randomly initialized parameters, an already trained network is used as the baseline for the new model. Such a pre-trained network is typically trained on a large dataset, such as ImageNet for image classification, from which it has already learned the general features of entities present in the training dataset. The network's ability to recognize those general features can then be used as a starting point for learning the more specific features of the new task. Transfer learning generally reduces the computational time needed for the training process, and the size of the dataset needed for the training of the neural network is also significantly smaller (the computational time needed for the training can be reduced from days to minutes, and the number of images needed to represent each class can be reduced from tens of thousands to hundreds of images).

In transfer learning, two main approaches to handling the pre-trained model exist:

- *fine-tuning* the whole neural network
- using the pre-trained model as a *frozen feature extractor*

Fine-tuning the network means that the parameters of the pre-trained model are used as a starting point for the training phase, instead of the random initialization used when training from scratch. The network is then trained as normal, with all the network parameters being updated during the training phase, but the convergence to the correct parameter values is faster thanks to the previous initialization. Using the pre-trained model as a frozen feature extractor means that the parameters of all of the layers are fixed to the values obtained from the pre-trained model, and only the parameters of the last, fully connected layer of the network are modified during the training phase (in some cases, n-last layers can be modified, which should theoretically improve the performance, but at the cost of additional computational time needed to retrain the network). This means that the new model is essentially a combination of two separate neural networks, the first one being the original pre-trained model with its last layer removed, and the second one being a single-layer network. The first network then takes the image to be classified as its input, and based on the trained parameters, the network outputs a vector of the image features (image embedding). This image embedding is then used as an input for the second neural network, which then outputs the score for each of the possible classes.

In this bachelor thesis, both of the above-mentioned approaches were tested, and it was experimentally determined that using the pre-trained model as a frozen feature extractor is a more viable strategy than fine-tuning the whole network, because of the shorter time needed to train the new neural network. The exact results of those experiments can be seen in Chapter 5.

2.5 Communication protocols

Since the Raspberry Pi single-board computer is not powerful enough to handle the computationally demanding task of retraining a neural network, an external server had to be used (see Chapter 3). The communication protocols used for communication with the server are described in this chapter.

2.5.1 HTTP

HTTP (Hypertext Transfer Protocol) is a communication protocol used to transfer web pages, images, and other data over the internet, and it is therefore the foundation of data communication for the World Wide Web. HTTP is a stateless protocol, which means that each request is independent of any previous requests, and it operates on a *client-server* model, where the client sends a *request* to the server, and the server then sends a *response* with the requested data (HTTP, 2023).

HTTP request consists of a method, a URL, and headers that provide additional information about the request, such as the type of data being requested (HTML, XML, JSON, binary, etc...), sender description or the preferred language of the sender. Probably the two most commonly used methods are the *GET* method, which is used for transferring data from the server to the client, and the *POST* method, which is used for transferring data from the client to the server. The HTTP response then contains a status code, which indicates whether the request was successful (status codes 200-299) or not successful (status codes 400-499 for client-side errors, status codes 500-599 for server-side errors) along with headers that provide information about the response, such as its length or content type (Mozilla, 2023).

Nowadays, over 80 % of websites use the secure extension of the HTTP protocol (W3 Techs, 2023), called HTTPS (Hypertext Transfer Protocol Secure), which provides encryption using *TLS* (Transport Layer Security) on top of all the standard HTTP functions. Overall, HTTP is a fundamental protocol of the internet and plays a crucial role in enabling communication and data transfer between clients and servers.

2.5.2 WebSocket

WebSocket is a communication protocol that enables two-way (full-duplex) communication between a client and a server over a single TCP (Transmission Control Protocol) connection. Unlike *HTTP*, which only allows one-way requests from client to server, WebSocket protocol allows for real-time communication between the client and server.

WebSocket protocol establishes a connection between the client and server using a special handshake process, where the client sends an HTTP request to upgrade the communication to the server. If the server supports WebSocket communication, it responds with an HTTP response which confirms that the protocol has been switched. Once the connection is upgraded to a WebSocket connection, both the client and server can send data to each other in real-time until the connection is closed. Messages used in the WebSocket protocol consist of a *header*, which contains information about the message, such as its length and type, and a payload, which contains the actual data being sent (WHATWG, 2023).

2.5.3 MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight, TCP/IP-based messaging protocol designed for high-latency (or otherwise unreliable networks), where small amounts of data (low-bandwidth networks) are being transmitted between devices (machine-to-machine, M2M communication). Thanks to these properties, MQTT is often used in IoT (Internet of Things) applications, where resources are often limited.

Because messaging in the MQTT protocol is based on the publish-subscribe model, there are two types of entities in MQTT - *client* and *broker*. The client can then work either as a *publisher* or as a *subscriber*. The publisher client sends a message within the specified *topic* to the broker, which then forwards the message to all clients subscribed to the topic, as is illustrated in Fig. 2.17.

A single client can work as a publisher or as a subscriber in multiple different topics. A single client can work as a publisher in one topic and as a subscriber in another topic at the same time, but it can work only as a subscriber or a publisher within one topic (MQTT, 2023).

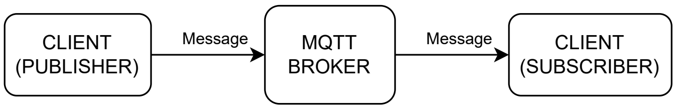


FIGURE 2.17: Block diagram of the MQTT messaging model

MQTT supports three levels of reliability and guarantee of delivery of the message, called *QoS* (Quality of Service):

- *QoS 0*: The message is delivered once or not at all
- *QoS 1*: The message is delivered at least once, but duplicates may occur
- *QoS 2*: The message is delivered exactly once

To demonstrate how MQTT communication works, consider the entities shown in Fig. 2.18.

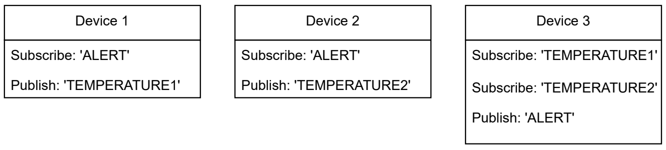


FIGURE 2.18: Model example of MQTT communication between three clients

Entities named *Device 1* and *Device 2* represent simple IoT devices measuring some arbitrary temperature values. Each of these devices publishes the measured temperature to its respective MQTT topic, and both of these devices subscribe to the 'ALERT' topic, which can be seen as a system-wide warning from the main device. This main device is the entity named *Device 3*, and it represents a smart home control center, which subscribes to both of the 'TEMPERATUREx' and publishes to the 'ALERT' topic. When *Device 1* publishes the measured value to the 'TEMPERATURE1' topic, only *Device 3* receives the message, because *Device 2* is not subscribed to the topic. On the other hand, when *Device 3* publishes a message to the 'ALERT' topic, both *Device 1* and *Device 2* receive the message, and can further process it according to their source code.

2.6 ROS - Robot Operating System

Robot operating system or simply ROS is an open-source software development kit maintained by the Open Robotics organization, which offers a collection of software libraries, tools, and drivers that helps simplify the developing and operating robot software. ROS is based on a decentralized architecture, where each component (called node) of the robot system is an individual executable program. These nodes are connected to each other via a message-passing system called ROS topics (ROS Wiki, 2023). Some of the ROS elements used in this bachelor thesis are listed below along with their short description:

- *Package* - ROS nodes and other relevant software, such as configuration or external libraries, are bundled into *packages* for release. These bundles should have enough functionality to be useful, but not so much that they become too difficult to use.
- *Node* - Node is a process responsible for running the executable code. Nodes should be written to only process a single specific task, and complex tasks should then be solved by multiple nodes communicating via messages.
- *Topic* - Topic is a named channel over which individual nodes send messages to each other. Topics work on an anonymous publish/subscribe model, where nodes do not know with what nodes they are communicating, and are only interested in the messages published to their subscribed topics.
- *Message* - Message is a simple data structure containing the data being sent to a topic. Standard primitive types (integers, floating point numbers, boolean values, etc.) are supported, as well as arrays of those types. Messages can also include custom, user-defined structures and arrays of such structures.
- *Service* - Service is analogous to a *topic*, with the difference being that services work on a request/reply model, and it is used in situations corresponding to a remote procedure call, where using a *topic* would not be sufficient. A client calls the service by sending the request message, and then it awaits the reply from the service.
- *Action* - Action works on the same principle as *service*, but it is more suitable for long-running tasks, as it can be canceled by the client if the execution time exceeds some threshold, and it can also send periodic feedback on the execution status.

According to the official website (ROS, 2023), the main goal of the ROS framework is to make code reusability as easy as possible in robotics research and development. As each node of the robot system is essentially a standalone executable program, such a node, or even a collection of multiple nodes (called Packages), can then be easily distributed among the research community.

The version of ROS used in this bachelor thesis is ROS Noetic Ninjemys, which is a distribution of ROS 1 released in May 2020.

2.7 SpeechCloud

SpeechCloud is a collection of tools for solving tasks associated with voice dialog systems, which was developed at SpeechTech, s.r.o. in cooperation with the University of West Bohemia in Pilsen. The tasks relevant for this bachelor thesis are automatic speech recognition (*ASR*), which means automatic computer recognition and transcription of human speech into text, and text-to-speech (*TTS*), which is a computer analysis and conversion of written text into a synthetic speech that sounds like a human voice (Švec, Neduchal, and Hrúz, 2022).

As SpeechCloud is a proprietary software, its implementation details are not accessible to the general public. However, because SpeechCloud is only used as a service for the ASR and TTS tasks in this bachelor thesis, an explanation of the client API is provided along with the simplified block diagram of the SpeechCloud architecture, which is shown in Fig. 2.19.

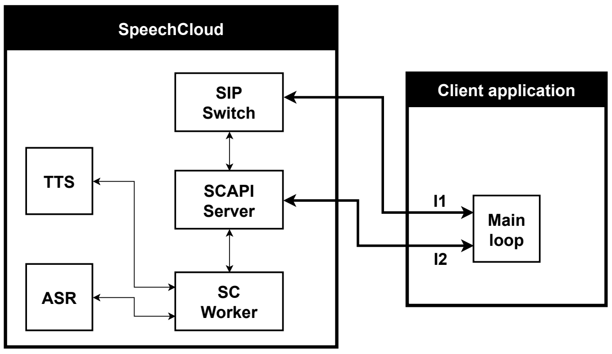


FIGURE 2.19: Simplified block diagram of the SpeechCloud architecture

The components in the block diagram shown in Fig. 2.19 are explained below:

- *SC API Server* - primary access point for the client application, issues configuration for the client application, provides logging of incoming and outgoing messages, mediates communication between the client application and SC Worker, and provides authentication for the SIP Switch
- *SIP Switch* - provides the transport of audio data between SC Worker and client application
- *SC Worker* - mediates communication between the client application and the ASR and TTS kernels, sends and receives the audio data via the SIP Switch
- *ASR* - speech recognition kernel
- *TTS* - speech synthesis kernel
- *I1* - communication interface for the standard VoIP (Voice over IP) audio data transfer between SpeechCloud and the client application
- *I2* - communication interface for the control communication, client configuration and logging messages between SpeechCloud and the client application based on either HTTP or WebSockets protocol

Chapter 3

Interactive learning dialog

The actual goal of this bachelor thesis was to implement the object detection and face recognition algorithms in an interactive dialog loop, where the system could dynamically learn to recognize new objects and faces based on the interaction with the user. The goal was for the system to be able to start "blank" (in a sense that the models were pre-trained, but not yet retrained accordingly to the task given by the user), where the user would first introduce themselves, and then teach the entity to recognize various objects.

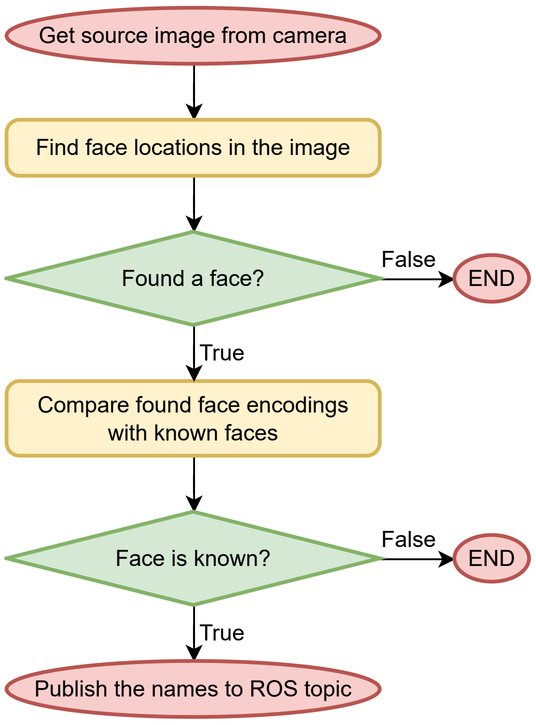
It is first necessary to explain how the detection and recognition modules are implemented on their own, and then explain how they interact with each other and how they are integrated into the interactive dialog loop.

3.1 Face recognition module

The face recognition module must be able to handle two tasks – recognizing faces on the camera and learning to recognize a new face. The recognition task is handled by the Python library described in (Section 2.2). The module itself is implemented in such a way that each iteration of the algorithm takes the image from the system's camera as the input, extracts encodings of the faces in the input image, computes the similarity between the faces in the input image and the known faces, and produces a list of all the known faces detected in the input image as its output, as shown in (Fig. 3.1).

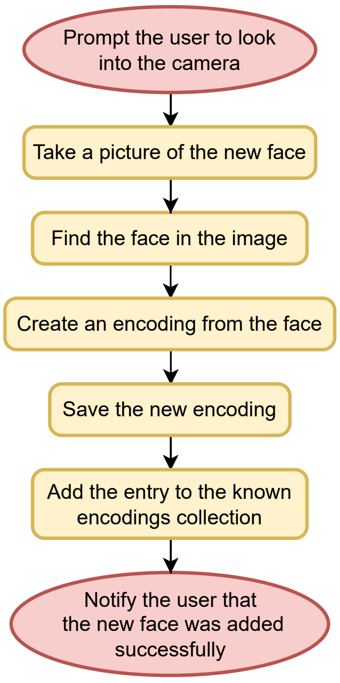
The task of adding a new face is similar to the task of recognizing faces. After the entity is prompted to take a picture of the new face via the voice dialog, it then starts a countdown of five seconds, after which the picture is taken. The entity then prompts the user to say his name, and pairs the name with the encoding, storing it in its collection of known faces. The collection of known faces is stored in a dedicated directory in the internal storage of the Raspberry Pi. To ensure that each encoding would be stored under a valid filename (as names can generally be comprised of multiple words, e.g. "Martin Adamec"), each encoding file is stored in a default binary format of the Python NumPy library under a name following the pattern "fXX.npy", where XX is a number of the encoding zero-padded to two digits (the first encoding would therefore be saved as "f01.npy", and the tenth encoding would be saved as "f10.npy"). The name provided by the user is then used as a key in the dictionary, while the ID (the name without the ".npy" extension) of the encoding file is used as a value corresponding to the key, which ensures that a user can easily rewrite the encoding used to recognize them. This dictionary is then stored in a JSON file, which is loaded for loading the known faces (see example below).

```
{
  "Martin Adamec": "f01",
  "Aragorn II Elessar, King of Gondor and Arnor": "f02",
  "Robert": "f03"
}
```



Get source image from camera
Find face locations in the image
Found a face?
False
END
True
Compare found face encodings with known faces
Face is known?
False
END
True
Publish the names to ROS topic

(A) Flowchart of the process of recognizing faces in a single frame



Prompt the user to look into the camera
Take a picture of the new face
Find the face in the image
Create an encoding from the face
Save the new encoding
Add the entry to the known encodings collection
Notify the user that the new face was added successfully

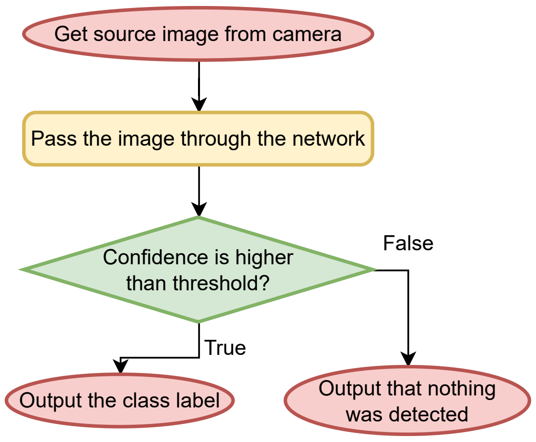
(B) Flowchart of the process of adding a new known face for recognition

FIGURE 3.1: Flowcharts of the two sub-processes of the face recognition module

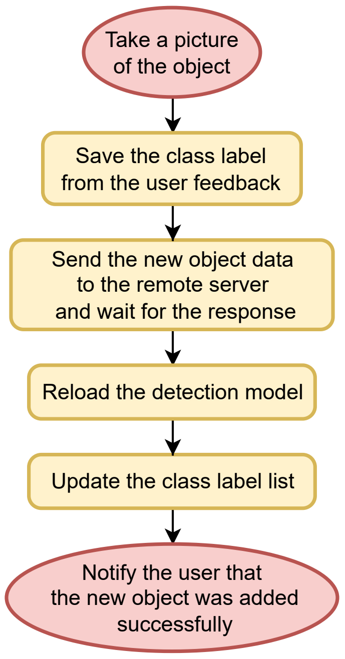
3.2 Object detection module

Similarly, the object detection module also has to handle two tasks – recognizing the object shown on camera, and learning to recognize a new object. The model used for the first task is the combination of an image embedding extractor with a classifier (Section 2.3), in this case, a fully-connected feed-forward neural network with one hidden layer comprised of 60 neurons. This network is retrained during each iteration of the learning dialog, with the weights being modified based on feedback from the user. The network used for the embedding extraction is the MobileNetV3 Large model, described in (Section 2.3).

Despite only modifying the weights of the small fully-connected network, the process of retraining the network on the custom dataset is still very computationally expensive, and the training therefore has to be done on a remote server with a GPU (the exact time difference can be seen in (Section 5.2)) to keep the delay between showing the new object to the system and the system being able to recognize it as short as possible (by utilizing the remote servers computational power, this delay is reduced to hundreds of milliseconds as opposed to minutes when training directly on Raspberry Pi). After the system is prompted by the user to enter the learning loop, the system tries to recognize the object shown on the camera based on the current configuration of the classifier. The predicted class label (name of the object) is then presented to the user, who gives the system feedback on whether the prediction is correct or not. If the prediction is correct, nothing is modified and the system continues to another iteration. If the prediction is evaluated as incorrect by the user, the system retrains the classifier in order to be able to assign a correct class label to the object. The retraining itself is performed on the remote server, where the retraining of the small fully-connected layer is done in under a second, as shown in (Chapter 5). Another benefit of the architecture of the utilized detection model is that only small amounts of data are transferred between the system and the remote server - a single image with a class label from the system to the server, and the weights of the small neural network from the remote server back to the system (because the embedding extractor is not modified at any point, it can be stored on both the remote server and the Raspberry Pi). As shown in (Fig. 3.2), the detection task itself is implemented in a way similar to the face recognition task - the image taken by the camera is passed to the neural network retrained for the current custom dataset. If the confidence of the predicted class passes a threshold of 40%, the predicted class is outputted as a result. Otherwise, the user is notified that no object was recognized by the system.


Get source image from camera
Pass the image through the network
Confidence is higher than threshold?
False
True
Output the class label
Output that nothing was detected

(A) Flowchart of the process of detecting an object in an image


Take a picture of the object
Save the class label from the user feedback
Send the new object data to the remote server and wait for the response
Reload the detection model
Update the class label list
Notify the user that the new object was added successfully

(B) Flowchart of the process of retraining the network to recognize a new object

FIGURE 3.2: Flowcharts of the two sub-processes of the object detection module

3.3 Integration into the dialog loop

This section describes the approach chosen for the integration of the four above-mentioned procedures into an interactive dialog loop (Fig. 3.3). When the system is activated, it starts a main infinite loop, called "watch around", during which it tries to find known faces (Fig. 3.1a) in the video stream from the camera. If a known face is detected, the entity enters a dialog loop (Fig. 3.4) during which the user can give a command to recognize an object shown to the camera (Fig. 3.2a), to enter a learning loop in which the object detection model can be retrained (Fig. 3.2b), or to add a new face to its collection of known faces (Fig. 3.1b). The process of "watching around" until a known face is detected could be compared to the *wake word detection*, commonly used with voice assistants, referring to a specific phrase used to activate the assistant (e.g. "Hey Google" for the Google Assistant[1]), just modified to react to video input (face) instead of audio input (phrase).

[1]`https://assistant.google.com/`

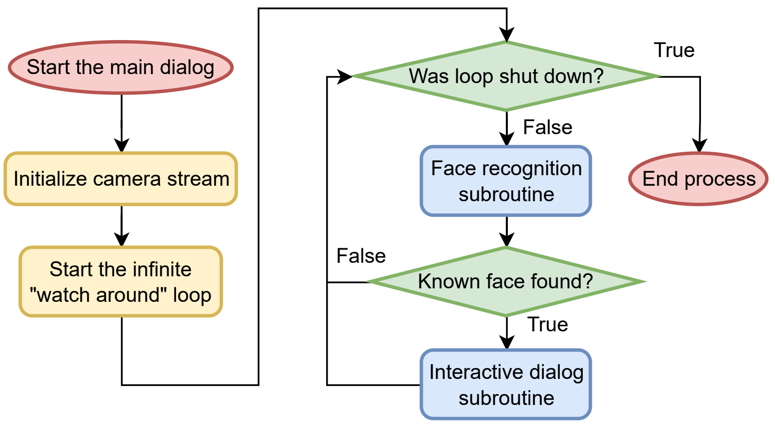


FIGURE 3.3: Flowchart of the main loop of the interactive system

In Fig. 3.3, it can be seen that the system scans the camera input for faces in an infinite loop, waiting until a known face is detected before allowing the user to interact with the computer vision models. This was chosen for simplicity reasons, as the "watch around" operation is not time intensive when no known face is detected, allowing for multiple loop iterations per second. This approach theoretically makes it impossible for unknown users to add themselves to the collection of known faces, creating a primitive form of authorization. However, because the goal of this thesis was not to create a security system, but rather an interactive dialog system implemented on a low-cost robotic entity (Chapter 6), an unknown user can manually prompt the system to enter the "New face" subroutine via the robotic entity GUI.

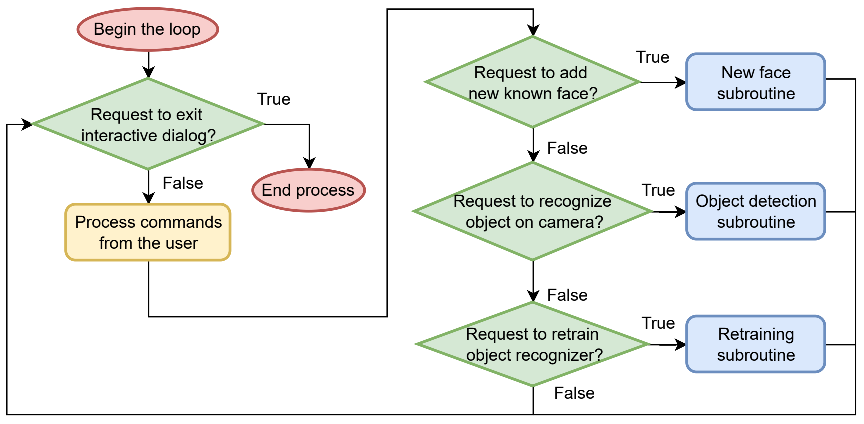

Begin the loop
Request to exit interactive dialog?
True
End process
False
Process commands from the user
Request to add new known face?
True
New face subroutine
False
Request to recognize object on camera?
True
Object detection subroutine
False
Request to retrain object recognizer?
True
Retraining subroutine
False

FIGURE 3.4: Flowchart of the interactive dialog loop (computer vision tasks)

As shown in Fig. 3.4, when the interactive dialog loop is entered, the system is ready to recognize voice commands via the ASR module (alternatively, the commands can be given via the GUI of the system). If a command to start any of the three previously described tasks (recognize object, retrain detector, add new known face) is recognized by the system, the respective subroutine is entered. After this task is completed, a new iteration of the loop starts from the beginning, meaning that only one task can be done per loop iteration. This is done for simplicity reasons, such as that the loop would always start with the user command recognition.

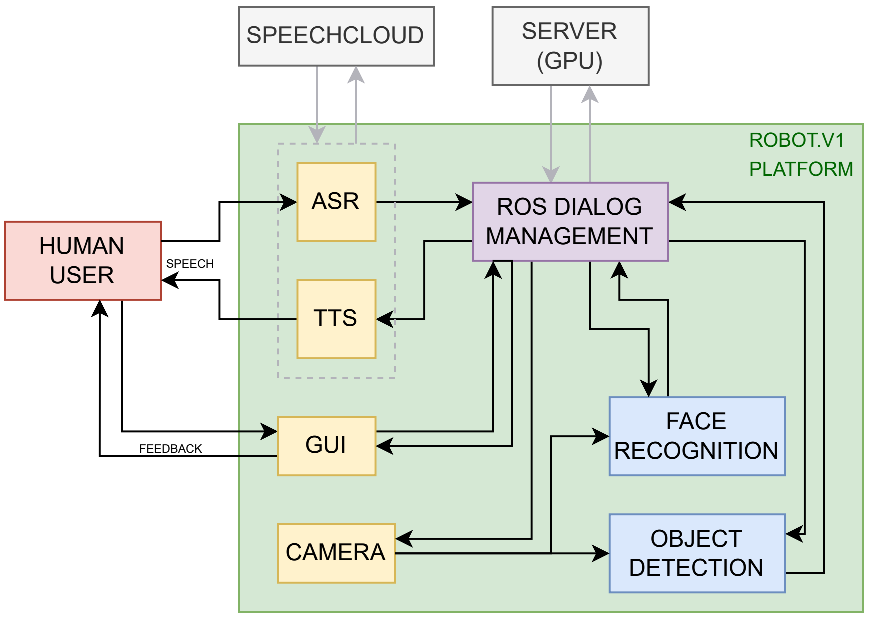


FIGURE 3.5: Block diagram of the interactions between the robotic entity and the human user

The block diagram depicted in Fig. 3.5 illustrates the details of the interactions between individual modules of the robotic platform *Robot.v1* (Chapter 6). The block labeled "ROS DIALOG MANAGEMENT" represents the algorithm that controls the whole robotic entity, which can simply be seen as the entry point to the "watch around" infinite loop for the purpose of this thesis (in reality, this block could serve as an entry point multiple different processes at once, but those other processes are not the subject of this bachelor thesis). The blocks labeled "GUI" and "CAMERA" represent the peripherals of the robotic entity, while the blocks labeled "ASR" and "TTS" represent the speech recognition and synthesis blocks described in Section 2.7. Finally, the blocks labeled "FACE RECOGNITION" and "OBJECT DETECTION" represent the modules described in Section 3.1 and Section 3.2 respectively.

Chapter 4

Data

This chapter provides a brief description of the various datasets used in this bachelor thesis.

4.1 Dataset D1 - Labeled Faces in the Wild

The *Labeled Faces in the Wild* (*LFW*) dataset[1] is a large collection of images of faces. This dataset consists of over 13 000 face images scraped from the internet, and is frequently used as a benchmark for facial recognition tasks, and was also used to pre-train the models in the *dlib* library, which was used in (Section 2.2).

4.2 Dataset D2 - ImageNet

The *ImageNet* dataset[2] is a large-scale collection of images used in object recognition tasks. The dataset consists of a thousand classes ranging from animals to food to everyday objects, with each class represented by thousands to millions of labeled images in various resolutions. The ImageNet dataset is often used as a benchmark for image classification tasks, and many pre-trained networks, including the MobileNetV3 model (Section 2.3) used in this bachelor thesis, have been trained on this very dataset.

4.3 Dataset D3 - COCO

The *COCO* (Common Objects in Context) dataset[3] is another popular large-scale dataset used in the area of computer vision for the tasks of object detection, segmentation, and captioning. The object detection part of this dataset contains hundreds of thousands of labeled images in various resolutions, forming a total of 80 object classes. The COCO dataset is also commonly used to benchmark models in computer vision.

[1] `http://vis-www.cs.umass.edu/lfw/`
[2] `https://image-net.org/index.php`
[3] `https://cocodataset.org/#home`

4.4 Dataset D4 - Intel Image Classification

The *Intel Image Classification* dataset[4] is a medium-sized dataset consisting of 6 classes, with most of the images having a resolution of 150 by 150 pixels, with some of the images being smaller. Despite this dataset commonly being used for the task of scene understanding (the classes represent physically larger objects such as a mountain or a building), it was determined that using this dataset would nicely illustrate the limits of using a pre-trained model for the task of object recognition.

4.5 Dataset D5 - Custom dataset

Due to the nature of the problem addressed in this bachelor thesis, it is expected that a user will create a custom dataset comprised of the objects that they want to teach to the entity. Because the objects in the dataset are going to change based on the dialog interactions with the user, only the general structure of the custom dataset is described - when prompted by the user, the robotic entity takes a picture of what the user is currently showing to the camera. Each picture is taken in the resolution of 640 by 480 pixels, as that is the default resolution of the OpenCV library used to handle the camera hardware. The images representing the object are then stored under the label provided by the user in the interactive dialog.

[4]https://www.kaggle.com/datasets/puneet6060/intel-image-classification

Chapter 5

Experiments and results

Because the purpose of the robotic entity is to run a dialog loop with the shortest possible time delay between the user input and the response from the robotic entity (ideally in real-time), the computational speed was the most important parameter in the design of the solution, and the accuracy of the algorithm was the second most important parameter.

5.1 Evaluation of face recognition

This section describes the results of the multiple experiments performed in order to evaluate the task of face recognition.

5.1.1 Face localization algorithm - HOG or CNN

When implementing the algorithm described in (Section 3.1), it was necessary to determine what settings should be used in the final implementation to meet the above-mentioned criteria. Both of the algorithms for face localization that the *face_recognition* library offers were compared. Based on the results shown in Table 5.1, the *HOG* algorithm was chosen for the implementation, as it is noticeably faster than the *CNN* based localization method.

Face localization model	**Average inference time per frame**
CNN	3.64 s
HOG	0.58 s

Table 5.1: Comparison of computational time for the two localization algorithms available in the *face_recognition* library - average over 100 iterations

The same comparison was made for the choice between the 68-point model and the 5-point model for the extraction of facial features, but the computational time difference has proven to be negligible, so the 68-point model was chosen for the implementation.

5.1.2 Influence of the number of known faces

Because the comparison of the encodings of the face detected on camera and the known faces is done in a sequential manner, comparing the detected face encoding with every known face encoding, it was expected that the inference time would increase with a higher number of known faces. However, this has not proven to be the case, as the time difference was not observed to increase with the growing size of the known faces collection, as can be seen in Table 5.2.

Number of known faces	**Average inference time per frame**
1	0.69 s
10	0.69 s
100	0.69 s

TABLE 5.2: Comparison of inference times per frame for different number of encoding in the known face collection - average over 100 iterations

5.2 Evaluation of object detection

This section describes the results of the multiple experiments performed in order to evaluate the task of object detection.

5.2.1 Ready-to-use object detection methods

To illustrate how well the used object detection models would work if used without any modifications to the pre-trained weights, an experiment to determine the accuracy was done on 100 random images from two datasets - *D3* (COCO) and *D4* (Intel Image Classification).

Model	**Mean accuracy (D3)**	**Mean accuracy (D4)**
MobileNetV3 Large	0.97	0.27
YOLOS Tiny	0.96	0.43

TABLE 5.3: Comparison of average prediction accuracy on datasets D3 (Section 4.3) and D4 (Section 4.4) over 100 random samples

As expected, both models achieved high average accuracy on the *COCO* dataset, as both networks are pre-trained on the *ImageNet* dataset, which has a significant overlap of classes with the COCO dataset. On the other hand, on the *Intel Image Classification*, both models achieved significantly lower average accuracy, which is to be expected, since there is no overlap of class labels between this dataset and the training datasets of both networks (theoretically speaking, the accuracy would be 0 due to the mutually exclusive class labels, but the detection was counted towards the accuracy if any object was detected correctly, regardless of the actual label in the dataset D4).

5.2.2 Influence of transfer learning

Multiple experiments were done in order to determine the more suitable approach between the two alternatives described in Section 2.4 - *fine-tuning* and *frozen feature extractor*. The results (time needed for retraining the network on the robotic entity) for both approaches are shown in (Table 5.4). The network was trained for 15 epochs.

Dataset	**Fine-tuning**	**Frozen feature extractor**
3 classes, 3 train, 2 val	5 m 5 s	3 m 11 s
5 classes, 3 train, 2 val	6 m 28 s	3 m 5 s
2 classes, 100 train, 70 val	94 m 48 s	46 m 33 s

Table 5.4: Comparison of computational time needed for retraining the MobileNetV3 Large network on Raspberry Pi

This experiment was also performed to demonstrate the high time requirements, justifying the utilization of the remote server for this task.

5.2.3 Image embedding + FFNN

To illustrate the efficiency of this approach (described in more detail in (Section 2.3)), the experiment was repeated multiple times with different backbones for the image embedding extraction. This experiment was done on the *Intel Image Classification* dataset.

Embed. model	**Mean embedding time**	**Train time**	**Accuracy**
Resnet-18	0.0365 ± 0.0033 s	0.42 s	0.882
Alexnet	0.0246 ± 0.0039 s	0.37 s	0.874
Vgg-11	0.1258 ± 0.0143 s	1.09 s	0.896
Densenet	0.1194 ± 0.0216 s	4.98 s	0.884
efficientnet_b0	0.0460 ± 0.0090 s	3.61 s	0.895
efficientnet_b1	0.0591 ± 0.0063 s	6.63 s	0.889
efficientnet_b2	0.0605 ± 0.0105 s	7.84 s	0.898
efficientnet_b3	0.0846 ± 0.0112 s	9.56 s	0.883
mobilenet_v3	**0.0358 ± 0.0092 s**	**0.8 s**	**0.915**

Table 5.5: Performance comparison of different image embedding extractors on dataset D4 (Section 4.4)

As can be seen from the results in Table 5.5, the MobileNetV3 had the best performance of all tested backbones. A significant improvement in accuracy when compared with the experiment performed on the pre-trained, unmodified version of the MobileNetV3 network can also be seen, which further demonstrates the power of transfer learning.

5.2.4 Influence of ONNX optimization

To improve the performance of the object detection models on the limited hardware of the Raspberry Pi platform, the *ONNX* framework mentioned in Section 3.2 can be utilized. The inference time comparison was performed on randomly selected images from the D3 dataset (Section 4.4), and the result of this comparison for the *MobileNetV3 Large* model is shown in (Table 5.6), and the result for the *YOLOS Tiny* model is shown in (Table 5.7).

Hardware	**Normal inference time**	**ONNX inference time**
Laptop	0.105 ± 0.033 s	0.078 ± 0.037 s
RaspberryPi 4	1.524 ± 0.239 s	0.637 ± 0.193 s

TABLE 5.6: Comparison of average inference time of the baseline model and the compiled ONNX model over 1000 iterations - MobileNetV3 Large

Hardware	**Normal inference time**	**ONNX inference time**
Laptop	0.051 ± 0.009 s	0.034 ± 0.007 s
RaspberryPi 4	0.493 ± 0.111 s	0.373 ± 0.112 s

TABLE 5.7: Comparison of average inference time of the normal model and the compiled ONNX model over 1000 iterations - YOLOS Tiny

From the results of both experiments, it is clearly visible that the *ONNX* optimization improves the performance of the model, noticeably reducing the inference time for a single frame. Even for the chosen image resolution of 150 by 150 pixels, the time difference is significant, and it allows for a real-time response with a delay of under 1 second, even for the slower model based on MobileNetV3.

The "Laptop" mentioned in both Table 5.6 and Table 5.7 had the following specifications:

- CPU - Intel Core i3-7130U, 2.70 GHz, 2 cores, 4 threads
- RAM - 16 GB
- GPU - The laptop used for the experiments has no external GPU

The experiments were originally performed on a laptop to determine if utilizing the ONNX optimization would be of any relevance, and the experiments were repeated on Raspberry Pi after the ONNX optimization was proven to achieve significantly better results.

5.2.5 Learning dialog evaluation

As a final experiment for the evaluation of the object detection module, an example of running the learning loop is presented. In this examples, the system starts with a custom dataset consisting of three known class labels: *ball*, *mug* and *pen*.

During the runtime of the learning loop, more samples are added to the dataset representing each class, and a completely new class "face" is added during the training loop. The number of samples in each class over time can be seen in the lower graph in (Fig. 5.1), and the accuracy for each class along with the mean accuracy for all classes combined can be seen in the upper graph in (Fig. 5.1). It can be seen that even with a relatively small number of samples in each class, the system has a very high recognition accuracy for all classes, demonstrating the power of using the pre-trained feature extractor, and how drastically the computational time and size of the dataset needed to achieve satisfactory results are reduced thanks to the utilization of transfer learning (Section 2.4).

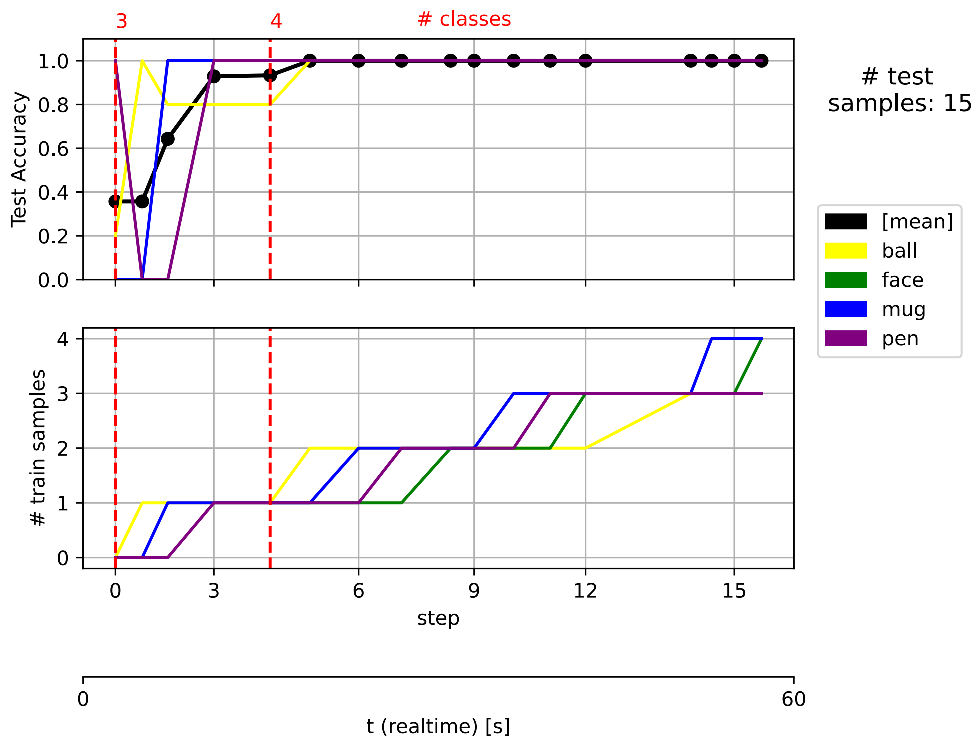


FIGURE 5.1: Example of the learning loop for adding new objects

Chapter 6

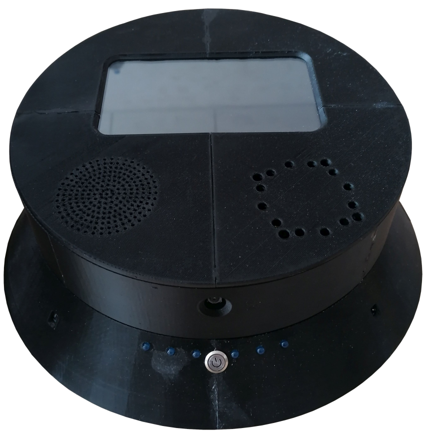
Application

As a final step in this bachelor thesis, the algorithms described in Chapter 3 were implemented on a physical entity, called *Robot.v1*.

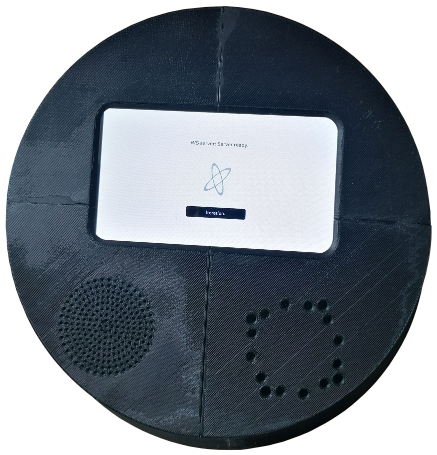
Robot.v1 is a multimodal low-cost physical entity developed in cooperation of Faculty Of Applied Sciences and Ladislav Sutnar Faculty Of Design And Art at University Of West Bohemia in Pilsen (Fakulta Aplikovaných věd a Fakulta designu a umění Ladislava Sutnara Západočeské univerzity v Plzni). The budget for the whole entity was set at 10 000 crowns, and the goal of the whole project is to create a functional prototype as a proof of concept. Robot.v1 has a rotund shape and is comprised of two parts, the stationary base and the rotational part. As the entity was designed to be multimodal, it needs to have adequate peripherals to be able to handle both audio and video input and output. The list of all the peripherals is listed below:

- Microphone array
- Camera (Raspberry Pi Camera Module 2)
- Speaker
- LCD touchscreen
- Power button
- Light sensors

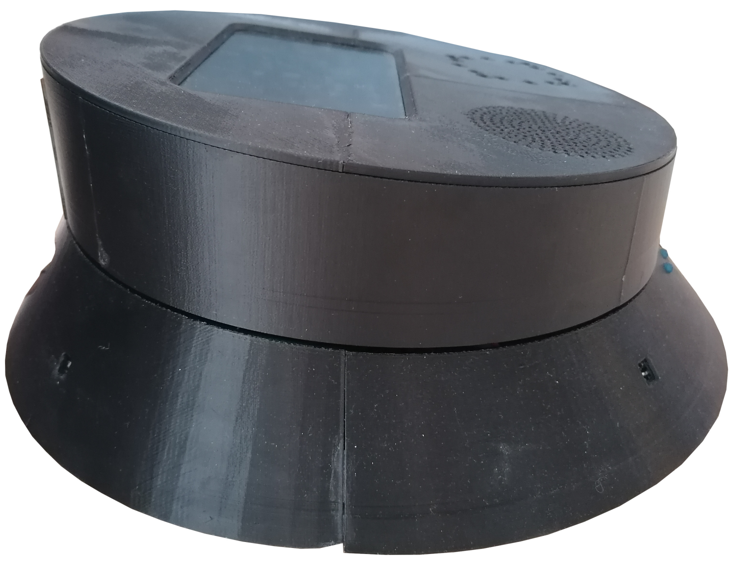
The computational part of the entity is based on Raspberry Pi 4B, which is the latest version in the line of low-cost single-board computers developed by the Raspberry Pi Foundation, running the Raspbian operating systems (a Debian-based GNU-Linux distribution designed specifically for the Raspberry Pi). This board can be purchased for a price of about 1000 to 2500 crowns (35 to 75 dollars) depending on the RAM size, and it is able to operate all the necessary peripherals. The physical case of Robot.v1 was designed and modeled by a design student and then printed on a 3D printer, which was chosen to ensure short iteration times between different stages of prototyping.



(A) A photo of the Robot.v1 entity from the front



(B) A photo of the Robot.v1 entity from the top, showing the LCD touchscreen



(C) A photo of the Robot.v1 entity from the side

Figure 6.1: Photos of the physical realization of the Robot.v1 entity

Chapter 7

Discussion

The overall objective of this bachelor thesis consisted of four subtasks:

1. Determine the most suitable algorithms for the tasks of face recognition and object detection on Raspberry Pi
2. Incorporate those algorithms into a voice-interactive system
3. Design a method for teaching the system to recognize new faces and objects
4. Implement the voice-interactive system on a physical robotic entity

7.1 Recapitulation of Methods

At first, a detailed explanation of the algorithms chosen for the implementation was given in Chapter 2, so that the design choices could be understood completely.

The implementation of the face recognition module was described in (Section 3.1), explaining how the recognition of the user's face is incorporated into the interactive voice dialog, how the information about known faces is stored in the system, and how the system learns to recognize a new face.

In Section 3.2, the implementation of the object detection module is described, explaining how the user can teach the system to recognize new objects, detailing how the model is split into two smaller models, one of which acts as an extractor of the feature vectors (embeddings) from the image, and the other then acts as a classifier, assigning an individual class (label) to each of those embeddings.

Finally, the incorporation of both of the modules into the interactive voice dialog is explained in Section 3.3, describing how the user can interact with each module within the main dialog loop.

Alternative approach to object detection

As was previously stated in this thesis, *MobileNetV3 Large* (Section 2.3) network was originally chosen as the go-to model for the object detection task. However, during the experiment phase, multiple new approaches were discovered, leading to the addition of the *YOLOS Tiny* network, which utilizes the transformer architecture explained in (Section 2.1). The original idea for the retraining procedure for the object detection task was to prompt the user to show the new object on camera and rotate it around, while the robotic entity would take 100 images of the object, ensuring that the dataset would contain images taken from multiple different angles. However, this approach has been rejected during the implementation in favor of the approach described in (Section 3.2) in the final implementation, as it was determined that it better illustrates the effects of retraining the entity on new objects.

7.2 Summary of Results

The three main tasks described in Chapter 3 along with their implementation details, and the experiments performed to verify the solutions were described in Chapter 5. The summary of the results achieved in each of the three tasks respectively is provided below.

7.2.1 Face recognition

The implementation presented in Section 3.1 has been proven to be able to keep the response time below one second, allowing the module to run in real time. The system is able to learn to recognize a new face when prompted, and it is able to do so from just a single picture, from which the feature vector is extracted and stored in the collection of known faces.

7.2.2 Object detection

The solution presented in Section 3.2 has shown the system's ability to learn to recognize a new object from just a few sample images shown to the system, demonstrating the interactivity of the implementation. When the system is prompted by the user, it is able to recognize the object shown on camera, which is done entirely on the Raspberry Pi hardware, while still retaining a real-time response. When the system is prompted to enter the learning loop, it takes a picture of the object shown on camera, tries to assign a label to it with its current classifier settings, and then it takes feedback from the user. If the feedback tells the system that there was an error in the classification, the system sends the picture and the correct label to the remote server to retrain the classifier there. After receiving the updated model back from the server, the system is ready for the next iteration of the training loop.

7.2.3 Application on physical entity

Although at the time of writing this bachelor thesis the physical prototype of the Robot.v1 entity is not yet complete due to unforeseen complications during construction, the multiple experiments described in Chapter 5 are sufficient to demonstrate the validity of the entire Robot.v1 project concept and the idea of creating a low-cost multimodal robotic entity in general. It has been proven that even with such limited computing power it is possible to create an interactive system that can process both audio stimuli (voice control) and visual stimuli (object and face recognition), while at the same time handling all the peripherals described in Chapter 6.

Chapter 8

Conclusion

The purpose of this bachelor thesis was to verify whether it is possible to implement an interactive system on the computationally limited hardware of the Raspberry Pi 4 single board computer so that the system is able to process audio and visual stimuli from the user with sufficient accuracy.

The solution chosen and implemented was presented for both the task of face recognition and for the task of object detection, along with the integration of both of these solutions into the main dialog loop of the interactive system. Experiments showing that the system is able to recognize new users by learning their faces, and to recognize new object by interactively getting feedback from the user, and updating its current knowledge based on this feedback were also presented.

To sum up this thesis, it works well as a proof of concept, showing that it is indeed possible to achieve quite impressive results in the tasks of face recognition, which works entirely on the Raspberry Pi hardware, and even in the task of object detection, where the remote server is utilized to speed up the computationally intensive process of retraining a neural network, all while maintaining interactivity and the ability to teach the system to recognize new objects and faces. However, there is still a lot of work that could be done to further improve the solutions presented.

8.1 Future Work

8.1.1 Robot.v1

As implied by the "v1" in the name "Robot.v1", the form of the physical entity at the time of writing the bachelor thesis is only a first prototype, serving as a proof of concept for the whole idea of a low-cost robotic entity. Therefore, the addition of new features is expected in the future. Some of the future features are already planned, namely automatic rotation of the top of the robotic entity so that the solar panel is directed towards the strongest light source, or automatic rotation of the entity by the microphone towards the sound source, and it is possible that they will become the subject of bachelor theses in the upcoming years.

8.1.2 The general concepts utilized in this thesis

During the March of 2023, the group of students working on various parts of the Robot.v1 entity as part of their bachelor theses participated in the "Aimtec Hackathon 2023" event, weekend whose central theme was "when code helps" ("Když kód pomáhá"), focusing on new ways in which technology can make life easier for people with different kinds of disabilities. During this event, the idea arose to apply the knowledge gained during the writing of this bachelor thesis to a small handheld device that would be able to help visually impaired people recognize small objects that might be difficult for them to distinguish. The concepts and methods introduced in this bachelor thesis were mainly developed as proof of concept, but if there is any potential of a real-world application, this idea was determined as the best choice by the author of this thesis.

Bibliography

[1] Y. Lecun et al. *Gradient-based learning applied to document recognition.* 1998. DOI: 10.1109/5.726791.

[2] Jie Hu et al. "Squeeze-and-Excitation Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42 (8 Sept. 2017), pp. 2011–2023. ISSN: 19393539. DOI: 10.1109/TPAMI.2019.2913372. URL: https://arxiv.org/abs/1709.01507v4.

[3] Ashish Vaswani et al. *Attention Is All You Need.* 2017. arXiv: 1706.03762 [cs.CL].

[4] Peter Bloem. *Transformers from scratch.* [Accessed: 11.04.2023]. 2019. URL: https://peterbloem.nl/blog/transformers.

[5] Andrew Howard et al. *Searching for MobileNetV3.* 2019. arXiv: 1905.02244 [cs.CV].

[6] Tom B. Brown et al. *Language Models are Few-Shot Learners.* 2020. arXiv: 2005.14165 [cs.CL].

[7] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.* 2021. arXiv: 2010.11929 [cs.CV].

[8] Yuxin Fang et al. *You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detection.* 2021. arXiv: 2106.00666 [cs.CV].

[9] Team Towards AI. *How do GPUs Improve Neural Network Training?* [Accessed: 27.03.2023]. 2021. URL: https://towardsai.net/p/l/how-do-gpus-improve-neural-network-training.

[10] Fırat Hardalaç et al. "Fracture Detection in Wrist X-ray Images Using Deep Learning-Based Object Detection Models". In: *Sensors 2022, Vol. 22, Page 1285* 22 (3 Feb. 2022), p. 1285. ISSN: 1424-8220. DOI: 10.3390/S22031285. URL: https://www.mdpi.com/1424-8220/22/3/1285/htmhttps://www.mdpi.com/1424-8220/22/3/1285.

[11] Eduardo Mosqueira-Rey et al. "Human-in-the-loop machine learning: a state of the art". In: *Artificial Intelligence Review 2022 56:4* 56 (4 Aug. 2022), pp. 3005–3054. ISSN: 1573-7462. DOI: 10.1007/S10462-022-10246-W. URL: https://link.springer.com/article/10.1007/s10462-022-10246-w.

[12] Sertis. *Why you should care about classical approaches to computer vision.* [Accessed: 25.03.2023]. 2022. URL: https://sertiscorp.medium.com/why-you-should-care-about-classical-approaches-to-computer-vision-847c2ba27e05.

[13] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.* 2022. arXiv: 2207.02696 [cs.CV].

[14] Jan Švec, Petr Neduchal, and Marek Hrúz. "Multi-modal communication system for mobile robot". In: *IFAC-PapersOnLine* 55.4 (2022).

17th IFAC Conference on Programmable Devices and Embedded Systems PDES 2022 — Sarajevo, Bosnia and Herzegovina, 17-19 May 2022, pp. 133–138. ISSN: 2405-8963. DOI: `https://doi.org/10.1016/j.ifacol.2022.06.022`. URL: `https://www.sciencedirect.com/science/article/pii/S2405896322003378`.

[15] Prices DataSet SHOP. *Pricing for legally clean datasets for AI training.* [Accessed: 27.03.2023]. 2023. URL: `https://www.datasetshop.com/pricing`.

[16] HTTP. *HTTP Documentation.* [Accessed: 11.04.2023]. 2023. URL: `https://httpwg.org/specs/`.

[17] Mozilla. *HTTP.* [Accessed: 11.04.2023]. 2023. URL: `https://developer.mozilla.org/en-US/docs/Web/HTTP`.

[18] MQTT. *MQTT.* [Accessed: 11.04.2023]. 2023. URL: `https://mqtt.org/`.

[19] ROS. *ROS.org.* [Accessed: 17.04.2023]. 2023. URL: `https://www.ros.org/`.

[20] Introduction ROS Wiki. *ROS Wiki - Introduction.* [Accessed: 17.04.2023]. 2023. URL: `https://wiki.ros.org/ROS/Introduction`.

[21] Technologies W3 Techs. *Usage statistics of Default protocol https for websites.* [Accessed: 11.04.2023]. 2023. URL: `https://w3techs.com/technologies/details/ce-httpsdefault`.

[22] WHATWG. *WebSockets.* [Accessed: 11.04.2023]. 2023. URL: `https://websockets.spec.whatwg.org/`.