HOCHSCHULE DER MEDIEN

Implementing Deep Learning Object Recognition on NAO

Bachelor's Thesis in the Degree Course
Medieninformatik
Submitted by:
Yann Philippczyk
Matr.-No.:
25692
at **Hochschule der Medien Stuttgart**
on January 11, 2016

First Examiner: **Prof. Dr.-Ing. Johannes Maucher**
Second Examiner: **Prof. Walter Kriha**

Implementing Deep Learning Object Recognition on NAO

by:

Yann Philippczyk

Computer Science and Media
Stuttgart Media University

January 11, 2016

Statutory Declaration/ Eidesstattliche Erklärung (German)

Hiermit versichere ich, Yann Philippczyk, an Eides statt, dass ich die vorliegende Bachelorarbeit mit dem Titel "Implementing Deep Learning Object Recognition on NAO" selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der eidesstattlichen Versicherung und prüfungsrechtlichen Folgen (§ 26 Abs. 2 Bachelor-SPO der Hochschule der Medien Stuttgart) sowie die strafrechtlichen Folgen (gem. § 156 StGB) einer unrichtigen oder unvollständigen eidesstattlichen Versicherung zur Kenntnis genommen.

Ort, Datum, Unterschrift

Abstract (German)

"Deep Learning" Ansätze haben sich für die Objekterkennung als sehr effektiv erwiesen, insbesondere in der Form künstlicher neuronaler Netze. Im Rahmen der vorliegenden Bachelorthesis wird aufgezeigt, wie eine einsatzbereite Objekterkennung auf NAO Robotern mit Convolutional Neural Networks implementiert werden kann, basierend auf vortrainierten Modellen. Die gleichzeitige Erkennung mehrerer Objekte wird mit dem Multibox-Algorithmus realisiert. Die Erkennungsraten der Implementierung werden in Tests evaluiert und analysiert.

Außerdem stellt die Implementierung eine grafische Benutzeroberfläche zur Verfügung, die Möglichkeiten zur Anpassung des Objekterkennungsprozess und zur Steuerung des Roboterkopfes bietet, um Objekte leichter im Blickfeld erfassen zu können. Zuzüglich wird ein Dialogsystem zur Abfrage der Erkennungsergebnisse vorgestellt.

Abstract

Deep learning methods have proven highly effective for object recognition tasks, especially in the form of artificial neural networks. In this bachelor's thesis, a way is shown to implement a ready-to-use object recognition implementation on the NAO robotic platform using Convolutional Neural Networks based on pretrained models. Recognition of multiple objects at once is realized with the help of the Multibox algorithm. The implementation's object recognition rates are evaluated and analyzed in several tests.

Furthermore, the implementation offers a graphical user interface with several options to adjust the recognition process and for controlling movements of the robot's head in order to easier acquire objects in the field of view. Additionally, a dialogue system for querying further results is presented.

Contents

1 Introduction

Sight is one of the most important, if not the single most important sense humans possess. The simple act of looking at a scene, its entities, their relations towards each other and a multitude of other circumstances, relays a vast amount of information which can be used among other things for interaction with entities, orientation in a location, or learning from what has been seen. With current rapid progress in the development of autonomous systems like cars, drones and robots for lots of possible applications in all fields of activity, it is only natural that intelligent systems need effective ways for not only perceiving, but also understanding their surroundings through the means of robotic or, more general, computer vision.

While the field of computer vision has been around for a long time, recent years have seen great advances and a renewed focus on applying deep learning techniques in the form of neural networks for object recognition tasks. Especially Convolutional Neural Networks (CNNs) have shown promising results in recognizing objects with unprecedented accuracy [39]. In respect of this background, it was of interest to apply the object recognition capabilities offered by CNNs to a practical, ready-to-use example.

1.1 Motivation and Goal

Being able to accurately recognize objects would pave the way for further research and projects, therefore a great incentive was given to actually develop an implementation for an usable object recognition system capable of detecting and classifying objects on the NAO robot platforms [23] available to us.

Since the NAO robot already offers ways of retrieving images from its cameras, and several frameworks supporting deep learning techniques for object recognition are readily available, the goal of this work is to combine these technologies to a working implementation. As for the actual neural network executing the recognition, the pretrained models accessible through the Caffe deep learning framework [3] appeared to be a suitable choice. They represent a solution perfectly tailored to our needs, relatively easy to implement and without the need for further training of the underlying CNN. Thus, it was possible to treat the exact internal workings of the object recognition more or less like a black box and concentrate on the combination of the mentioned technologies.

In order to be usable together with other projects, desirable features for the implementation are a clean architecture and modularity of the code, so that reusing it is possible in parts or as a whole, as well as the resulting software to be usable for demonstrations without much special knowledge or training regarding object recognition or the robot. Furthermore, due to a CNN having to be trained on a certain set of object categories resulting in a model for recognition, it would be interesting to make the models in use by the software interchangeable, so that potentially a fitting, specialized model could be used for each task at hand.

1.2 Content

The present work is segmented into the following parts, where the first four provide an overview over the primary technologies and frameworks used, and the latter three deal with the actual implementation of the object recognition on NAO, as well as evaluations and discussions:

- **Convolutional Neural Networks:** Describes the concepts of Convolutional Neural Networks and their typical architecture.
- **The Caffe-Framework:** Provides an introduction to the Caffe-framework, which is used as the concrete implementation of CNNs for object recognition in this bachelor's thesis.
- **NAO:** A description regarding aspects of the NAO robot like the underlying NAOqi-framework and hardware used for the implementation.
- **Related Work:** Presents similar work and distinctions compared to the current work.
- **Implementation:** In-depth description of the developed implementation's architecture and functionality.
- **Tests and Experiments:** The experiments conducted to verify different aspects of the implementation.
- **Conclusion:** Summarizes and discusses the current work, and gives an outlook over future usage and possible improvements.

1.3 Term Definitions

This section defines the terms used within the present work, whose meaning might be ambiguous.

- **Object**: Any physical living or non-living entity, be it an organism, an item, a vehicle or anything else, will be referred to as an **object**. Due to misrecognitions, structures formed e.g. by shadows or resulting from artifacts of low image qualities might be falsely identified as **objects**.
- **Object Recognition**: In the context of the present work, **object recognition** means both detecting that and where object instances are present in a given scene, and identifying to which object class (for example "car", "fruit", "electronic equipment", etc) they belong exactly, thus classifying the object.
- **Implementation**: The entire software developed in the current work for realizing the object recognition, communicating with the NAO robot, as well as the user interface will be referred to as the **implementation** in this work.
- **Convolutional Neural Networks**: A **Convolutional Neural Network**, abbreviated **CNN**, is a special type of artificial neural network with a distinct architecture especially suitable for object recognition. While CNNs will mainly be treated as a black box in the current work, an overview and description of their fundamental concepts is given in 2.
- **Local/Remote Classification**: The process of using one of the two classification methods described in detail in 6.2, referring to executing the classification process either **locally** on the system used for controlling NAO, or with a **remote** (web) service.
- **Bounding Box**: In the current work, a **bounding box** is a rectangular box enclosing a specific detected object on a given image. **Bounding boxes** are defined by their coordinates on the image representing the width and length as well as the position of the rectangle, and are typically visualized as coloured boxes around the object in question.

2 Convolutional Neural Networks

2.1 Overview

Currently, Convolutional Neural Networks represent the state of the art in terms of computer vision and especially object recognition and identification. This chapter will present an overview over CNNs and their most important functionalities and components.

While the foundations and principles of CNNs can be traced back to the work of Kunihiko Fukushima [29] and Yann LeCun et al. [33] in the 1980s, recent years have seen a renewed research interest in the technology. A current example of CNNs gaining the attention of a wider public outside of the immediate research community is the Deep Dream engine developed by Google [10].

The effectiveness of CNNs can be seen in the winning entries of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [39]. The ILSVRC represents a benchmark for object recognition tasks. Contenders compete on a subset of the complete ImageNet data divided into 1000 object categories. The provided training data encompasses 1.2 million images divided between these categories [11]. The challenge itself consists of two tasks, one being object detection, the other classification and localization.

Most of the top contenders in the ILSVRC used CNNs, with remarkable results compared to earlier object recognition implementations, which used hand-crafted components. During ILSVRC 2014, the winning GoogleNet achieved a top-5 error of only 6.67% [41].

Apart from good recognition rates, another point important for practical applications is the CNNs' robustness toward highly variable image parameters like lighting, distance, alignment and partial occlusion of the object to be identified, demonstrated for example in research toward robust place [40] and face [27] recognition.

2.2 Architecture

This section provides an introduction to CNNs and focuses on their differences to regular neural networks. While the exact operating principles of CNNs are treated as a black box in other parts of the current work, the most important aspects will be concisely explained here. All the information in this section was extracted and summarized from the sources indicated in the text.

As regular artificial neural networks are inspired by biological neural networks in the brain, CNNs are additionally inspired by the visual cortex found in mammals [9] [25]. While CNNs naturally share many similarities with other deep neural networks, e.g. neurons with weighted parameters, calculating an output and trained through adapted gradient descent and back-propagation to learn certain characteristics, as well as using input-, output- and hidden layers, there are major differences in the exact architecture [25]. CNNs' distinct architecture makes them uniquely suitable for object recognition in images and video.

The primary, eponymous component are the convolutional layers. Regular neural networks are not suitable for images due to the bad scaling of fully connected neurons in the first hidden layer, each having weights equal to the width $\times$ height $\times$ depth of the image, were the depth is typically the amount of colour channels, 3 in case of RGB images. For an image resized to a resolution of 200 $\times$ 200, this results in 120000 parameters for each single neuron in the first hidden, fully connected layer. Consequences of such a large amount of parameters are very long training times and a high probability for overfitting [34] [8].

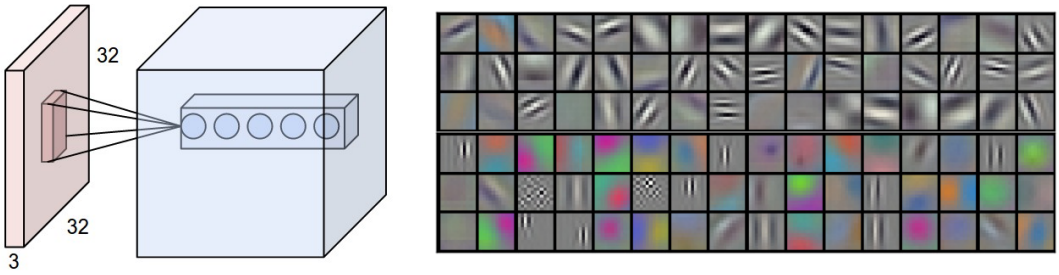

Figure 1: **Left:** A single column with a depth of 5 neurons and its corresponding receptive field (dark red). Source:[8] **Right:** 96 Filters learned by the first convolutional layer of an CNN trained by Krizhevsky et al. Source:[32]

In contrast, in convolutional layers, neurons are locally connected and form so called columns. Each column does not see the entire input at once, but only a local region, called the receptive field. A single column with a depth of 5 and its receptive field can be seen on the left in Figure 1. Considering width and height, the receptive field is much smaller than the original input, but extends to the full depth of the image (e.g. $3 \times 3 \times 3$). The columns thus work as filters, reacting to local input patterns [9]. Each filter is moved in a predefined stride over the image, activating and learning when certain image characteristics are seen. Convolving the inputs leads to an activation map for each filter, as shown on the right in Figure 1. These activation maps are in turn stacked to form the output volume. An activation map's filters share parameters, so that a learned feature can be detected by all filters in the map [9]. This also helps to reduce the overall amount of parameters and therefore training time, with slight adaptations needed for the backpropagation algorithm [32].

A CNN does not only consist of convolutional layers. There are several additional components needed to form a fully working CNN, as detailed in [8]:

- **Input Layer**: Holds the complete input for the CNN, e.g. the image to be classified.
- **Pooling Layer**: Inserted between convolutional layers to reduce the spatial dimensions of input volumes. Downsampling is performed with a MAX operation using filters striding over the input, each time taking the maximum number as output, therefore reducing the dimensions. For example, a 2×2 MAX pooling filter would take the maximum value of 4 adjacent values, reducing the input dimensions by 75%.
- **Fully-connected Layer (FC)**: Similar to regular neural networks, each neuron in this layer is fully connected to the others in the previous layer. FC layers are usually inserted at the end of the CNN, and one FC layer always forms the final output layer, mapping its input to the object classes used in the current training.
- **Rectified linear unit (ReLU)**: The ReLU is not a layer, but an activation function used in convolutional and FC layers. It computes the output $f(x) = max(0, x)$ and optimizes certain calculations during training and usage while leaving the spatial dimensions unchanged [32].

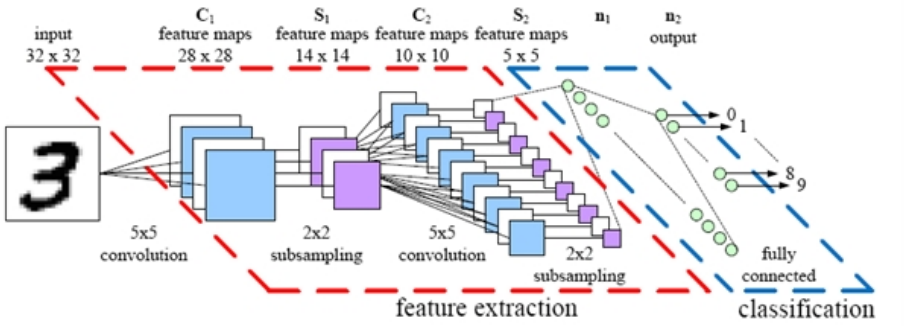

Figure 2: Architectural overview of a CNN, used for OCR in this example. Image source: `http://parse.ele.tue.nl/education/cluster2`

The architecture of a CNN determines how these layers are combined and how sections are repeated. A simple architecture example would be `INPUT -> [[CONV/RELU] -> POOL]` $\times$ `2 -> FC` where the $\times$ 2 means a repetition of the layers in the brackets. Generally, the earlier convolutional layers learn low-level image features like recognizing single edges or colours, while later ones learn increasingly higher level features like the structures or textures of objects. Another architecture example is visualized in Figure 2.

There are many variations of architectures, and lots of the exact mechanisms and details had to be omitted in this summary, like the drop-out method during training, fractional pooling and zero-padding [32][8]. Describing all of them is out of scope for the current work. Nevertheless, this section should provide a basic understanding of the core components and functionalities of CNNs.

3 The Caffe-Framework

3.1 Overview

Caffe [3] [30] is an open-source, deep-learning framework specialized in image classification with CNNs, developed by the Berkeley Vision and Learning Center. It is a toolbox for all important machine learning operations like training, testing, and deploying trained models [30]. In this work, it represents the main tool for tasks related to the object recognition itself. Caffe is implemented in C++, with Python and Matlab bindings and supports CPU as well as GPU calculations, the latter enabled by CUDA support.

Caffe was chosen for the implementation of object recognition in the current work due to its accessibility, documentation and the available pretrained models, allowing a quick development and deployment on the NAO robot. The Caffe web demo [5] proved to be useful as well, mainly for initial performance tests and usage in 6.2.2.

Alternative machine-learning frameworks are Torch [22], or the wide-spread Theano [21]. However, these frameworks were less accessible and offered much fewer readily available pretrained models.

3.2 Architecture

The architecture used by Caffe is described in detail in [30] and [4], from where the information presented in this section is taken. A CNN model in the Caffe framework is composed of individual layers, which together form a net. The data, which is forwarded through the net, is wrapped in data structures called blobs in Caffe.

A blob is a customizable, multidimensional array, which holds the actual data and provides a unified memory interface hiding the mechanisms necessary for synchronizing mixed operations on the CPU and GPU.

Nets organize the composition of layers and the data flows for forward and backward passes between them, usually starting with a data layer for loading input data, and ending with a loss layer calculating the loss functions.

Layers compute the actual operations on the data, for example convolutions and pooling. Caffe layers are equivalent to the general layers described in 2.2. Input data is taken through bottom connections, output is accordingly returned through top connections. This process

is called forward computation. Other ones are setup computations, which initialize the layer, and backward computations, which calculate and pass the gradients during training. Caffe's layer catalogue is organized in types. At the bottom of a net are data layers, which handle the raw input, for example from databases or directly from memory. Vision layers handle images and encompass types like convolution and pooling layers. Activation and neuron layers perform element-wise operations, for example with ReLUs, and loss layers handle the aforementioned learning process, e.g. by calculating a softmax loss. Of practical importance is the fact that all layers have both CPU and GPU routines, which produce the same output, thus enabling easy switching between whether the CNN's calculations should be carried out on the CPU or GPU.

3.3 Models

The pretrained models used for object recognition in 6.2 can be found in the Caffe model zoo at [3]. Each model realizes a specific CNN architecture. The models used throughout this work are BVLC CaffeNet, AlexNet and GoogleNet, all trained on the ILSVRC 2012 data set and offering similar performance. These models represent the core component enabling object recognition in the implementation.

The architecture of a specific model is defined in plaintext protocol buffer schema files (`*.prototxt`), while the corresponding trained models are available as binary protocol buffer files (`*.caffemodel`) as noted in [4].

3.4 Maximally Accurate/Specific Results

As explained in [6], Caffe is able to not only naively calculate highly specific classifications for an object detected in an image, but also the more abstract, higher-level class to which the object belongs by using a process, which is colloquially termed "hedging your bets", introduced in [26]. This is possible because ImageNet data is organized in a tree-like structure with more abstract entities higher up towards the root, while the leafs are concrete object classes. A good example for part of this tree is shown in Figure 3, from the paper linked above: The root of the subtree is the abstract class **Animal** with a child node **Mammal**, which in turn has the concrete classes **Zebra** and **Kangaroo** as its children.

This system is extremely useful for robust object recognition. It alleviates the problem of potentially too specific and therefore incorrect classifications. For example, given an image

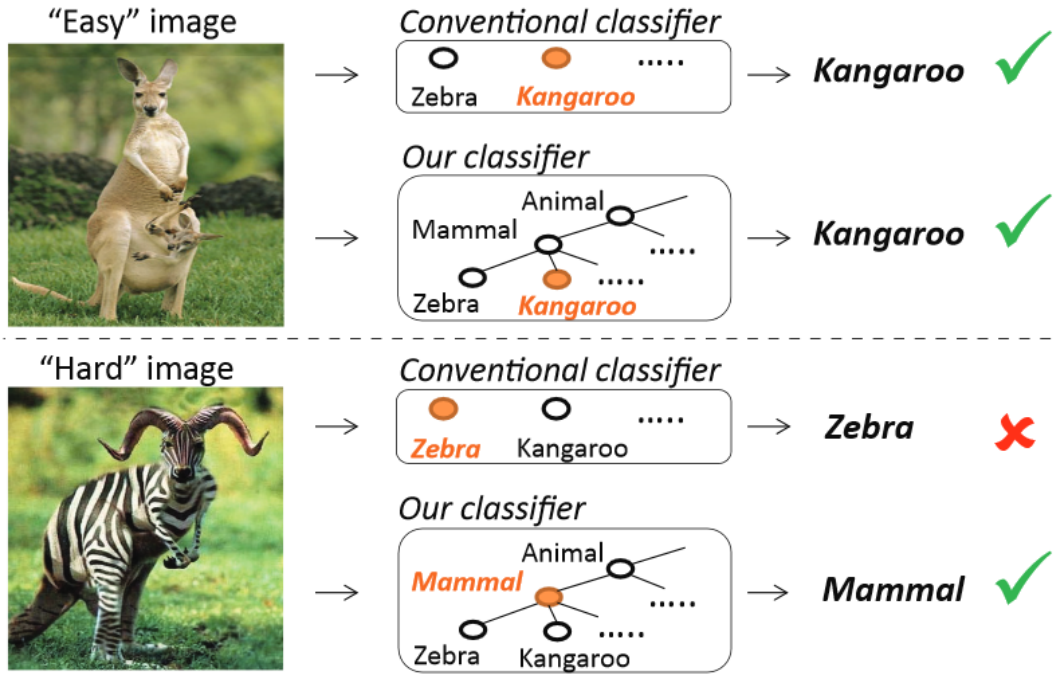

Figure 3: Example for "hedging the bets" when classifying unknown objects. ("Our classifier" denotes the classifier used in the source paper.) Image source: [26]

of a Golden Retriever with overall low class probabilities calculated by the CNN, instead of going for a maximally specific class and returning the false result **Chihuahua**, by "hedging its bets" the algorithm backs off higher into the class tree and returns the result **Dog**. This is less specific, but correct in this case.

Additionally, this allows the object recognition to even classify objects, which are not within the trained categories, but close enough to one to have a more abstract classification assigned correctly. As an example, the training set might not have contained quads, which do not exactly fit into the trained categories **Motorbike** or **Car** and even less their child nodes' categories. However, by backing off, the algorithm is able to correctly identify the quad as a **Vehicle**, which is a parent node of both **Motorbike** and **Car**. Another example for such a classification, in this case of a horned zebra-kangaroo hybrid, which does not even exist in reality, is given in Figure 3.

This mechanism is directly used in the current work to calculate the maximally accurate and specific results, in 6.2. While not correct in all cases, it usually provides additional useful results.

4 NAO

4.1 The Robot

The NAO robot, which is used in the current work, is a humanoid robotic platform manufactured by Aldebaran Robotics [14]. It is mainly used for education and research. The robot itself is about 58 cm high, capable of movement and speech, and possesses a range of sensor types and on-board computational capabilities (1.6 GHz CPU, 1 GB RAM, 2 GB flash memory, 8 GB micro SDHC). Connecting to the robot is possible by Ethernet and Wi-Fi.

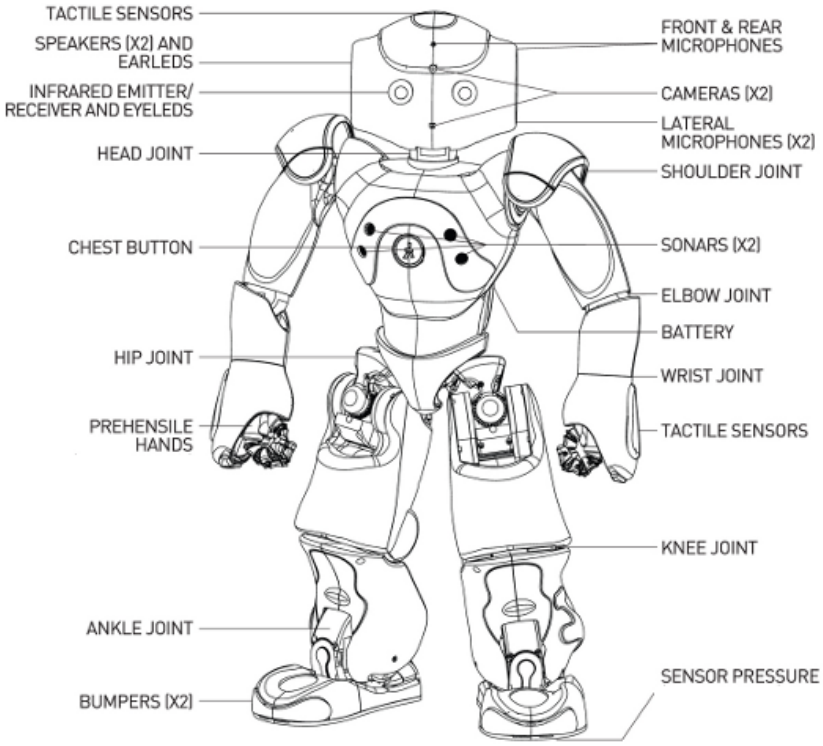

Figure 4: Arrangement of sensors and actuators. Image source: `http://doc.aldebaran.com/2-1/family/nao_h25/index_h25.html`

The most important sensors for the current work are the two front cameras able to take pictures in a resolution range from 160 x 120 up to 1280 x 960. It is important to note that the cameras are not actually placed in the "eyes" of the robot, but one on the forehead and the other in position of the "mouth", as can be seen in Figure 4, enabling vision downwards without moving the head. While the cameras are able to produce a continuous image stream, only single, static images are usable for object recognition in the current implementation.

The field of view for each camera is 47.64° vertically and 60.97° horizontally [15]. Downwards inclination is 1.2° for the top camera, 39.7° for the bottom one.

Other hardware components of immediate importance are the microphones and the loud speakers, used for speech recognition in the dialogue system 6.5 and returning the object recognition's results acoustically in addition to console output. The head joint relevant for the movement controls in 6.4.1 offers two degrees of freedom.

4.2 The NAOqi Framework

NAOqi, as described in [16] and [17], is a framework developed by Aldebaran and deployed alongside the NAO robot. It is a Linux distribution based on Gentoo and serves as operating system for the robot, enabling cross-platform development of programs for NAO. Main programming languages are C++ and Python. In the current work, version 2.1.4.13 was used. The NAOqi API is separated into several parts, each allowing access to distinct functionalities and systems of the robot:

- `NAOqi Core` contains the central methods for interacting with the robot's settings, resources, network connections, etc, and of special importance for the current work, the data transferred to and from every robot module through `ALMemory`.
- `NAOqi Vision` is another important part of the API, as it is used to take pictures with `ALVideoDevice`.
- Other APIs used are `NAOqi Motion` and `NAOqi Audio`. `NAOqi Motion` enables access to all functions controlling movement and navigation. `ALMotion` is used in the developed user interface for moving the robot's head. Likewise `NAOqi Audio` makes it possible to utilize speech and speech recognition for allowing human-machine-interaction with `ALTextToSpeech` and `ALSpeechRecognition`, respectively. This API enabled the development of the dialogue system in 6.5.

Further APIs, which were however not directly used in the current work, are `NAOqi PeoplePerception`, `Sensors`, `Trackers`, `Diagnosis` and `DCM`, each of them providing access to a multitude of functionalities like tracking objects, recognizing faces, using sonar and infrared sensors, diagnosis systems and many more.

Additional software provided by Aldebaran for NAO are the Choregraphe and Monitor tools. Choregraphe is a tool for creating behaviours for NAO by using a graphical user interface.

However, the current work makes no use of Choregraphe itself. On the other hand, Monitor, which is deployed together with Choregraphe, was used for development, testing and is highly suitable for usage alongside the object recognition implementation. Monitor provides access to NAO's image streams produced by the robot's cameras, and therefore allows the user to see from the robot's viewpoint. Different settings concerning the camera parameters like resolution, frame rate, contrast, etc are available.

Note that the part of the implementation described in 6.3.1, which actually retrieves images for classification, uses parameters independent from those set in Monitor. Another useful feature of Monitor is the ability to switch between the top and bottom cameras.

While the integration of a camera stream into the user interface in 6.4 would have been possible, this was found to be redundant in the presence of Monitor, which is readily available for deployment with NAO.

4.2.1 Using NAOqi Modules

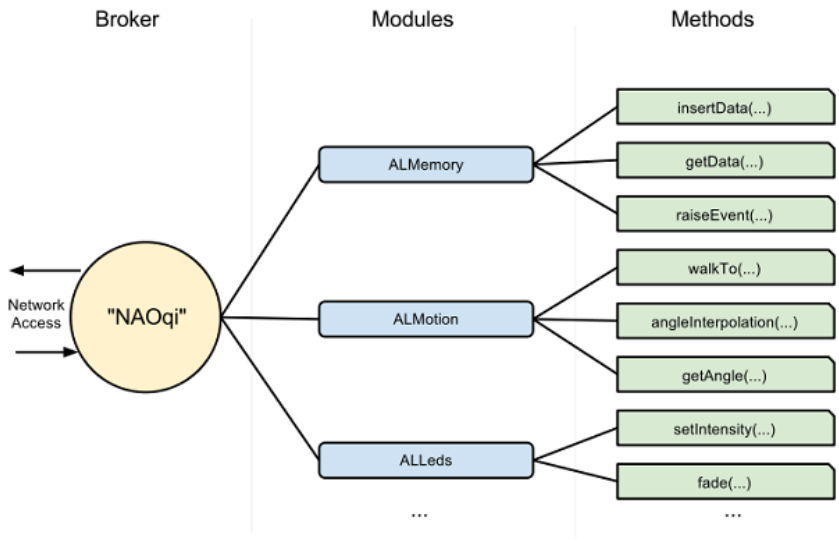

Figure 5: Relation between brokers, modules and their respective methods. Image source: [17]

NAOqi modules offer specific behaviours like moving a joint a certain amount of degrees in a direction. Elaborate details on using NAOqi modules are provided in the documentations cited in the following, however this section will summarize the key information extracted from these sources.

Accessing NAOqi modules like `ALMemory` requires a broker [17] loading libraries containing those modules and providing directory services and network access for calling them, as shown in Figure 5. Brokers are typically transparent. The primary tool for working with modules during development are proxies acting like the corresponding modules. In order to react to defined circumstances like a user talking to the robot, modules can subscribe to events, for example a proxy subscribing to `'WordRecognized'`. The reaction itself is then specified in an appropriate callback function, in this case `onWordRecognized(...)`. An alternative to the proxy concept is provided by the experimental `qimessaging` API [19] through sessions and services, which is used in 6.5.

5 Related Work

5.1 Overview

In this section, work similar to the current implementation as well as the differences are described, namely the `ALVisionRecognition` module, research on the iCub robotic platform, and other projects on NAO robots. While there is a vast amount of research conducted in the field of (robotic) object recognition, much more than can be mentioned here, to the best of our knowledge, there is no other work trying to realize object recognition using pretrained CNN models on NAO in a mostly similar way to the one presented in the current work.

5.2 Work on iCub

The iCub robot [35] is a humanoid platform functionally similar to NAO. In [36] iCub is taught to recognize different objects, using the BLVC Reference CaffeNet for extracting visual representations. While the actual process described in the paper is more elaborate, it can be summarized as follows: A set of different objects is shown to iCub and labeled through speech recognition when showing the object to iCub. After the labeling is completed, the robot is given multiple recognition tasks to test the effectiveness of the system.

The approach in the current work differs, apart from using a different robotic platform, in so far, as there is no further training involved before recognizing objects: The Caffe models are used without changes directly for recognition of arbitrary objects selected from the pretrained categories. The focus lies on the application of existing frameworks and technologies.

5.3 Work on NAO

The only other project incorporating the NAO platform and CNNs which could be found is described in [24]. The research goal is to recognize hand gestures with NAO's camera using a Multichannel Convolutional Neural Network developed and trained by Barros et al. Verification was done on a reference data set, and a data set recorded with the robot.

The project apparently does not use pretrained models, and focuses solely on hand gestures, unlike the current implementation, which is meant to recognize a large set of different object classes, as mentioned before.

5.4 ALVisionRecognition

The `ALVisionRecognition` module [1] is part of the NAOqi framework. It provides functionalities for learning to identify specific objects, which have to be labeled by hand. However, the capabilities offered by this module can be seen as the opposite of what the current work tries to implement.

While `ALVisionRecognition` enables the user to teach the robot specific object instances by using visual key points. Visually different objects can not be identified, if they have not been specifically learned beforehand, as described in [1]. Additionally, `ALVisionRecognition`, as provided by NAOqi, has no concept of object classes at all.

In contrast, by using CNNs for object recognition as in the current work, it is possible to recognize objects never seen before, if the object category they belong to is part of the training set underlying the model in use. On the other hand, in its current form, the implementation does not have the ability to recognize whether a specific object has been encountered before.

For example, given a specific car **A**, after **A** has been learned and labeled, `ALVisionRecognition` would be able to identify **A** if encountered again, but not a visually different looking car **B**, which could have been produced by another manufacturer. `ALVisionRecognition` can identify **A** only with the label given to it during training. However, by using the object recognition with CNNs implemented in the current work and an appropriately trained model, it is possible to recognize both **A** and **B** as belonging to the object category "car" without special training on any of the two instances. Depending on the instances, even a finer recognition like "sports car" or "minivan" is carried out.

6 Implementation

6.1 Overview

This chapter describes in detail the development process of the implementation for realizing object recognition with CNNs on NAO. All the developed modules will be explained, how they interact, as well as the general architecture of the solution. Where appropriate, use cases are described for clarification. The chapter's subsections are arranged chronologically in the order in which the respective parts of the implementation were developed.

The entire implementation's source code was written in Python 2.7, or uses Python wrappings in case of most of the code parts directly relying on other frameworks like Caffe or NAOqi. Development and testing was carried out on Ubuntu 14.04.

Besides developing an implementation for object recognition itself on NAO, a goal for this work is to provide code in a highly modular fashion, so that it can easily be reused as a whole or in parts in other projects or solutions for object recognition.

The implemented graphical user interface, which is described later on in 6.4, is meant to be operated by any user without requiring a lot of knowledge or training regarding the software, the NAO robot or object recognition.

6.2 Initial Design

The initial design contained the prototypes for the core functionality, namely the forwarding of images to the pretrained Caffe models for object recognition. Two use cases or modes of operation were identified from the start:

- **Local classification:** The image recognition is run directly on the device used for executing the scripts. This requires the Caffe framework and the needed pretrained CNN models to be installed on the device.
- **Remote classification:** The image, which is to be classified, is sent to a remote service running the Caffe web demo. In case of the present work, the Berkeley server at `demo.caffe.berkeleyvision.org/` is used, as it offers good performance and availability. For running the web demo on other systems, Berkeley provides code at `https://github.com/BVLC/caffe/tree/master/examples/web_demo`

Whenever local or remote classification are mentioned in the present work, these two modes are meant. Both are implemented in separate Python scripts. Due to the two mode's exact internal functionalities, there are differences in the expected parameters and the formatting of the return values, however the usage for calculating the recognition results is the same, in principle. The scripts described in the following are used for all tasks regarding the object recognition itself in the current work.

6.2.1 Local Classification

The script `localClassification.py` provides the method `classify_image_local(caffe_root, model_path, prototxt_path, image_path)`, which is based on tutorial code found at [12] for the CNN's basic usage and [5] for calculating the maximally accurate results. The method expects the absolute system path to the directory in which the Caffe framework is installed, the relative paths to the pretrained model to be used, its prototxt file and the absolute path to the image, which shall be classified.

This signature allows to switch between installed models on the fly. However, if other image categories than those defined in the ILSVRC 2012 are used, several modifications to the script's source code have to be performed. Since the categories have to remain the same throughout the present work to retain comparability between implementation performances and to keep the method's signature somewhat concise, this case is disregarded in the following.

The script further sets internal parameters for the object recognition and forwards the image to the CNN, which identifies the object on the image. Additionally to the five top labels found, which form the maximally specific result set, five more labels are calculated as mentioned before by "hedging the bets" [26] for the maximally accurate results.

6.2.2 Remote Classification

Implemented in `remoteClassification.py`, the function `classify_image_remote(image_path, url)` realizes the remote classification. Aside from the mandatory path to the image, a URL can be specified, which links to a service for identifying the objects. By default, the Caffe Web Demo at `demo.caffe.berkeleyvision.org/` is used. The web demo runs the CaffeNet model for identification.

The image is sent using a cURL command, and the response is parsed for the two sets containing maximally specific and accurate classifications.

6.3 Implementation on NAO

Using the prototypes from the initial design phase, the object recognition was implemented on the NAO robot with the help of the NAOqi framework provided by Aldebaran.

All the procedures mentioned in the following are executed not on the NAO robot itself, but on a system connected to it, using the methods offered by the frameworks to communicate with the robot and send commands. This was done mainly because object recognition with CNNs is relatively demanding in terms of computational performance and while it is possible to use only the CPU, recognition works fastest when a graphics card is available for forwarding images through the neural network, which is not the case on NAO robots.

6.3.1 Basic Implementation

The first step for realizing object recognition with NAO was getting a picture from the robot's camera. This was implemented in the function `identify_image_nao(robotIp, port, local_or_remote)`. Apart from the information needed to connect to the robot, it must be specified whether classification should be run locally or with the remote service.

The code for retrieving the image from NAO is based on [20]. The exact image parameters like resolution, colour mode etc are specified in the code. The image itself is accessed by using a camera proxy provided by `ALVideoDevice` and, depending on the parameter set, passed on to local or remote identification using `classify_image_local` or `classify_image_remote`, respectively.

The script then returns the results calculated by the identification methods to the calling method. Additionally, the pictures taken are saved to a designated directory for possible use in tests and evaluations.

When using the option for multiple object recognition, the image is first passed to the Multibox detector described in 6.6.5. The image segments produced are then treated as single images and forwarded as described above.

Apart from retrieving the pictures, architecture-wise this script serves as the central interface between the parts of the implementation controlling the robot's behaviour and those using

Caffe to perform identification. It is not dependent on the user interface described in the next section and can be called from any other module, which needs to perform object recognition.

6.4 User Interface

The graphical user interface developed for this work provides several functions apart from activating the object recognition. As stated earlier, it is designed to be functional and easily usable without special knowledge about the implementation.

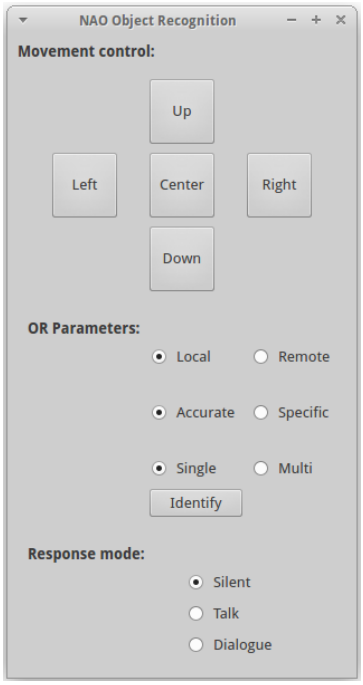

Figure 6: Screenshot of the implemented user interface. (Multibox version)

The script `objRecognitionGui.py` contains the entire code for the user interface, which is written in `PyQt4`. In the following, the functionalities provided by the GUI are described in detail.

6.4.1 Movement Control

While planning the development of the implementation, it became obvious that direct control over the robot's head movement would prove valuable, not only for later use, but also for taking test images easier without having to move the robot manually or using other scripts or tools like Choregraphe.

The movement buttons in the user interface allow control over the head joint's yaw and pitch angles using a motion proxy to `ALMotion`. The head is moved by keeping the buttons pressed, thus continuously sending signals to the handler, which in turn uses the method `setAngles` provided by the `ALMotion` API. The "Center" button sets both yaw and pitch to 0, so that the robot's head returns to the initial position and looks straight forward.

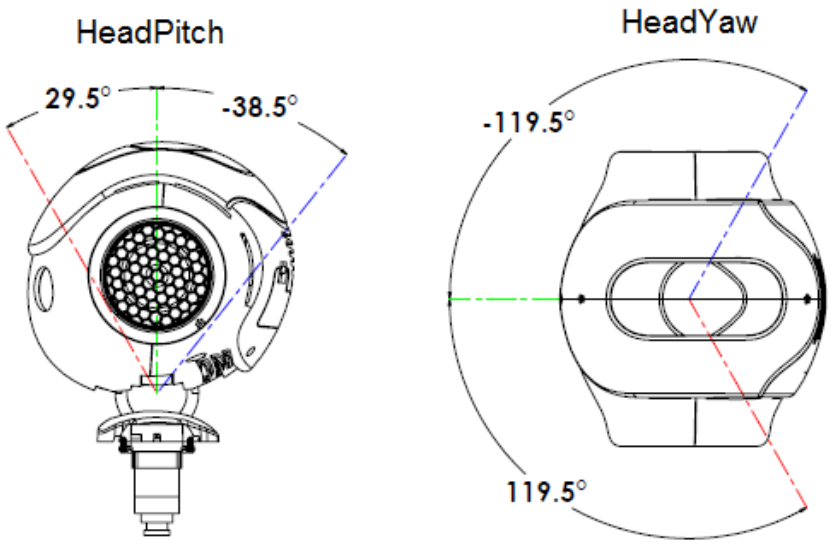

Figure 7: Maximum angles for the head joint. Image source: `http://doc.aldebaran.com/1-14/_images/hardware_headjoint_3.3.png`

As shown in Figure 7, maximum angles for pitch are 29.5° (forward) and -38.5° (backwards), the maximum range for yaw is ± 119.5°. However, with the current implementation of the head controls, the head movement starts bucking when approaching these maximum physical values, especially during pitching. As such, movement is slightly restrained to -34°, 28° and ± 110°, which results in noticeably smoother movement.

A detail related to responsive head controls is that once the head joint reaches a maximum angle, no further `setAngles` commands for the same direction should be sent. While this will not cause the head motor to try moving the joint beyond maximum values or damage the robot, additional `setAngles` calls towards direction **A** cause movement commands towards the opposite direction **B** to be delayed when sent.

For example, if not stopping to sent `setAngles` commands when the user keeps pressing the "Left" button, while the head joint is already turned all the way left, the next few presses on "Right" will appear to have no effect.

Stiffening the joint is another important detail concerning head movement. Before being able to control their movement, joints must be stiffened with the `setStiffnesses` method from the API, meaning that the motors for the joint in question are activated. The head joint is automatically stiffened when starting the GUI, and remains so while it is running. When stiffened, joints must not be moved by hand, as this could damage the motors.

The `atexit` Python module is used to ensure that the head joint is unstiffened when the GUI is closed. However, as explained in the `atexit` documentation [2], registered functions are not called when the program exits due to an external signal, a fatal internal error, or a call of `os._exit()`. In this case, the GUI should be restarted (and possibly closed) to correctly stiffen/unstiffen the joint.

6.4.2 Parameter Controls

The first part of the parameters, which can be chosen in the user interface, controls the object recognition itself, whether it should be run locally or remote, and whether accurate or specific results should be returned. This does not affect console output in silent mode. These settings are further elaborated in section 6.2. The second part is used to set the response mode of the robot after the recognition results are available. "Silent" only prints out the results in the console, and no acoustic feedback at all is given by the robot. "Talk" will cause NAO to provide acoustic feedback when starting the identification and tell the top result for either maximally accurate or specific results, as specified by the user. "Dialogue" will instead start the dialogue mode, which is further described in 6.5.

When using the Multibox-version of the implementation, there are additional options for switching between "Single" and "Multi" identification for recognizing multiple objects in a given image. This option is further explained in 6.6.5. Choosing "Multi" will set the response mode automatically to silent and disable the other response modes, because the bigger amount of results returned by Multibox is too unwieldy for acoustic responses, especially in dialogue form.

Clicking the "Identify" button will take a picture with the camera and forward it to the object recognition processes, as explained in 6.3.1. In the implementation's current form, only one identification process can run at a time, due to demanding computations when using only the CPU, and to prevent interferences in the robot's responses especially during the dialogue mode.

6.5 Dialogue System

The dialogue system is meant to offer the user a more interactive way of obtaining the object recognition results from the NAO robot. In the current implementation, the dialogue system is activated by selecting the response mode "Dialogue" in the user interface before starting the identification.

Once the CNN is done with the identification calculations, it returns the two lists with the five top results for maximally accurate and specific labels, as in the other response modes. Depending on the user's choice, the first entry of the selected result list is now read out by the robot, formulated as a question whether the result is correct. The robot's speech recognition is now activated, and it is waiting for an answer.

Valid answers the user can give are "Yes", "No" and "Exit". On a "Yes", the robot acknowledges the answer, and the dialogue, as well as the identification process, terminates and the user can carry on with the next recognition or other tasks on the robot. "Exit" works functionally the same way, providing a way to immediately terminate the dialogue without going through further results. "No" will cause the robot to proceed with asking the user whether the next result in the list is correct, again waiting for an answer. If no further results are available in the list, the robot will say so, and the dialogue and identification will terminate.

6.5.1 Implementation with NAOqi

The dialogue system's initial implementation in `identificationDialogue.py` used the NAOqi API with a speech broker and proxies to `ALMemory`, `ALSpeechRecognition` and `ALTextToSpeech` from NAOqi [16]. The entry point for the dialogue is the function `speechRecog (robotIp, port, results)`, where the last parameter holds the result list, which shall be used for the dialogue.

The script loops through the result list as explained in the prior section, using the callback function `onWordRecognized` for handling recognized commands from the user. A detail worth mentioning is that the speech recognition should only be active as long as NAO is waiting for a response and switched off before and after that, in order to prevent interference by NAO's own voice.

Implemented this way, the dialogue system was found to perform well when used on its own without the user interface running. Unfortunately, starting it from the GUI prevented the

system from working correctly, due to the way the NAOqi framework and its brokers work by using global variables for uniquely identifying event subscriptions.

This resulted in problems in combination with the GUI, which is running in its own thread with separate global variables, causing the dialogue not to be found under its name given in `identificationDialogue.py`.

A possible solution is to implement the dialogue system in the same `*.py` file as the user interface, thus sharing global variables. However, doing this would have broken up the modularity of the code, especially the clean separation between the user interface and the dialogue system, thus deteriorating the overall architecture.

6.5.2 Implementation with qimessaging

Using the experimental `qimessaging` API [19] proved to be a better solution. The overall functionality and procedure remains the same, however instead of brokers and proxies for subscriptions, `qimessaging` works by creating sessions and using services, as well as a special `qi.Application` for the event loop. The exact functionality is explained in the API.

Due to the service based architecture not needing global variables in the same way as NAOqi does, the dialogue system could be implemented separate from the GUI, retaining code modularity and functionality.

6.6 Locating Multiple Objects

Using the classification modes described in 6.2 will calculate results for one object in a given image, which was forwarded to the CNN. However, if there are multiple viable objects on the image, it is not possible to directly control, which object is identified, or to get classifications for all objects at once. Multiple passes of the same image through the same CNN will always return identical results. Moreover, no information is returned about the identified object's location in the image.

As such, a way is needed to potentially detect additional objects in the image, and pass those objects, or the image regions that contain them, individually to the CNN. There are many well documented approaches and algorithms for detecting multiple objects and their locations in images, and discussing all of them would be out of scope for the current work. Therefore,

the focus will largely remain on concrete implementations of those algorithms, which were more closely considered for incorporation into the current object recognition pipeline.

6.6.1 Sliding Window

One of the more naive approaches is to use a simple sliding window implementation, that is, to move a rectangular window (with a size smaller than the target image) in a certain pattern over the image as shown in Figure 8 and applying one of the implemented image classifiers to each of the segments obtained this way.

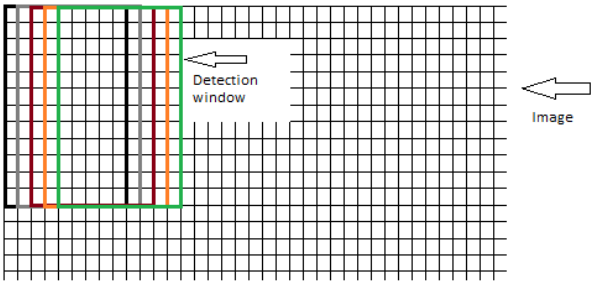

Figure 8: The detection window is moved over the target image in strides of one. Image source: `http://stackoverflow.com/questions/26607418/improving-accuracy-opencv-hog-people-detector`

While this approach is relatively simple to implement as described, it has numerous disadvantages. Obviously, depending on the size of the window and the exact pattern and stride used to slide it over the target image, there might be lots of segments, which have to be classified by the CNN. Apart from being prohibitively wasteful with regard to the performance, the same object could be detected multiple times on different segments, or not at all, depending on how it was fitted into the segments. This is especially the case if one single large object covers most of the image.

For example, each segment could only contain a small part of the object, resulting in the CNN being unable to classify it at all, due to having not enough information provided at once by each segment seen separately.

On the other hand, if the detection window or the strides are too big, there might be several objects in one segment, leading back to the initial problem. While there are refinements for the sliding window approach alleviating these points, a more computationally efficient method was sought.

6.6.2 R-CNN

A widespread solution for finding image regions containing objects is R-CNN, which combines region proposals (using Selective Search) and identification. Especially Fast R-CNN as described in [38] would have been an interesting possibility for usage with the NAO robot due to its higher speed.

Unfortunately, all R-CNN implementations, which were considered, rely on Matlab at runtime. Apart from not being easily available, Matlab would have been a rather demanding dependency restricting the overall applicability of the implemented solution. For those reasons, R-CNN was ruled out as an option.

6.6.3 Geodesic Object Proposals

Geodesic Object Proposals (GOP) [31], developed by Krähenbühl and Koltun, work in a different way compared to the other algorithms presented in this section. Instead of producing bounding boxes, GOP extracts the objects on a given picture themselves, as shown in Figure 9. According to [31], GOP offers precise results combined with computational efficiency.

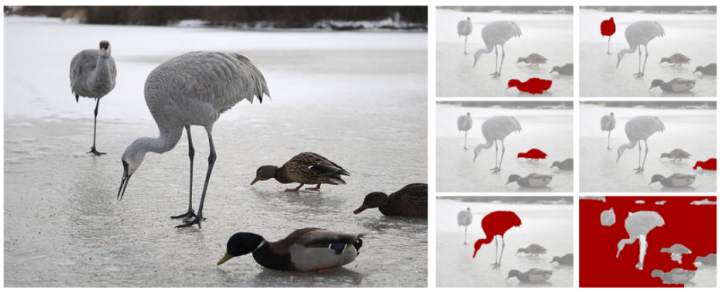

Figure 9: Object proposals produced by GOP. Image source: [31]

In practice, GOP proved difficult to install, requiring several other data sets to build. Additionally, due to a lack of documentation covering actual usage, GOP could not be correctly incorporated into the implementation, even after several attempts, and was therefore abandoned as possible solution.

6.6.4 Darknet Framework

The YOLO ("You Only Look Once") system [37], which is based on the Darknet framework, was another interesting option. Similar to R-CNN, it combines region proposals and identification into one single detection pipeline, shown in Figure 10. Additionally to offering good performance, the framework is easy to install and to run, which is important for practical considerations.

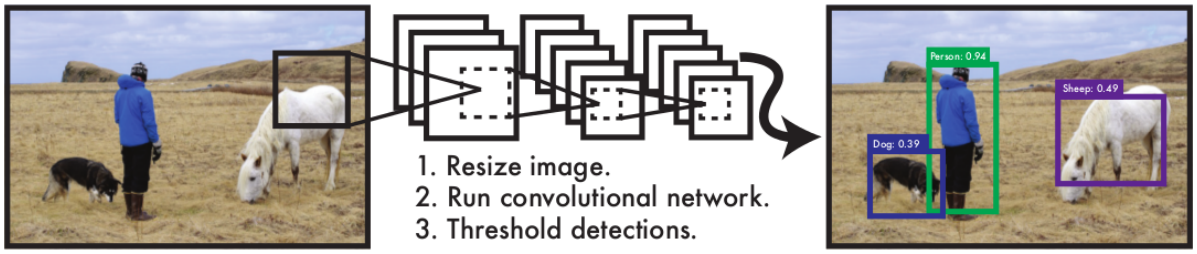

Figure 10: The detection pipeline realized by YOLO. Image source: [37]

However, although Darknet itself is capable of using similar pretrained models for identification as Caffe, pretrained weights for YOLO are only available for the 20 object classes of the VOC 2012 data set, a rather large setback from the 1000 classes offered by the Caffe modules trained on the ILSVRC 2012 data. Smaller tests indicated that the detection of object classes not covered by the VOC 2012 data set is unsurprisingly highly unreliable, depending on the difference between an unknown object class and the closest known one.

Training YOLO weights on ILSVRC 2012 categories would have been theoretically possible, but due to hardware and time requirements this is out of scope for the current work.

6.6.5 Multibox

The "DeepMultiBox" detector, developed by Erhan et al. [28] turned out to be a good solution for incorporation in the implementation, for considerations regarding both performance and architecture. The main advantage Multibox offers for the present work is that it works class agnostic. Thus, even if models with different trained categories are used or if completely unknown objects are present, Multibox is able to detect objects viable for identification on a given image. Examples for object detection with Multibox are given in 7.4.

Disadvantages are the high computational requirements for object detection with Multibox, as well as the chance of detecting the same object on a given image multiple times with

slightly different bounding boxes. This results in further drawbacks: Classifying the same object multiple times unnecessarily wastes a lot of performance. Additionally, if an object label occurs more than once in a given result set, it is not immediately clear whether the same object was recognized several times, or if there are indeed multiple instances of an object class present in the image. These disadvantages are however shared by lots of other object detection algorithms.

While Multibox depends on the experimental Caffe2-Framework [7], this dependency is preferable to MatLab for practical considerations, especially regarding accessibility. Most dependencies of Caffe2 are already satisfied after installing Caffe.

As for the actual usage of Multibox in the current work, if the user selected "Multi" as parameter in the user interface, the method `identify_image_nao` from `classifyImage_multi.py` will pass the image taken by NAO's camera to `create_img_segments(img_path)` in the file `imageSegments.py`. A slightly modified version of the Multibox code found in the IPython Notebook at [13] is then executed with `generate_boxes(img_path)`, which is contained in the script `multibox.py`.

In contrast to the original code, the modified one will not display the image with the bounding boxes found, but will instead return the boxes' coordinates to `create_img_segments`. The coordinates are then used to create segments containing the objects found by cropping the original image according to the coordinates. The segments are saved to a designated directory and forwarded to the rest of the object recognition pipeline as single images for identification. Older segments from previous identification runs will be overwritten to prevent cluttering the system with segment files.

Using multiple object recognition will return a larger result set, as the five top detections are used as segments, each yielding five identification results both for accurate and specific classifications, bringing the total results returned to 50 per image.

6.7 Configuration

In order to collect those parameter values, which are commonly used throughout the implementation, in a central point, they were added to `objRecogConfig.py`. The file contains the robot's ip, which might have to be changed to the current one manually before starting the implementation, as well as the paths to the NAOqi and Caffe modules, the pretrained models and other files required for the local classification and URL for remote classification.

The Multibox version additionally contains the path to the Caffe2 directory. If the model in use should be exchanged for another one, only the paths to the prototxt and caffemodel files have to be adjusted, as long as the model uses the ILSVRC 2012 object categories.

Working with other categories altogether would require providing the respective files, appropriate changes to the path entries, and probably modifications to parts of `localClassification.py` depending on the differences between Caffe-provided models and the new model, as the system was developed specifically with the usage of the Caffe framework in mind.

6.8 Final Architecture

This section describes how all of the aforementioned components from chapter 6 work together and summarizes the data flows between them. The implementation's final architecture assembles all functionalities and algorithms, and is meant to shape and support the collaboration between the three actors involved in the object recognition process: The user, the robot and the computer system, as depicted in Figure 11.

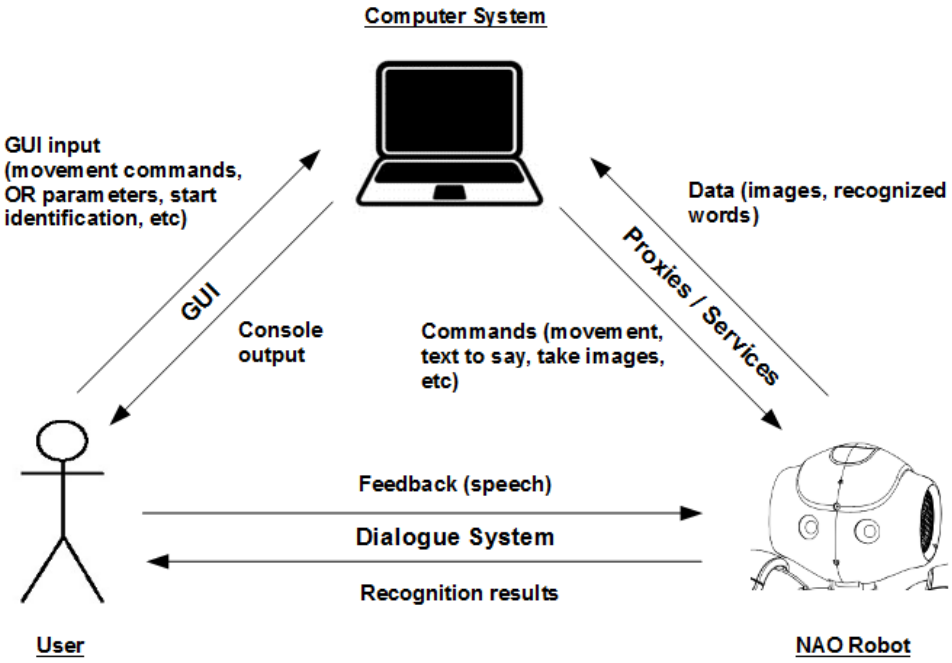

Figure 11: The three actors (underlined) collaborating by using the respective interfaces or communication channels (enclosed by the arrows).

The computer system serves as command interface between user and robot, and carries out the actual object recognition calculations using the CNN with the pretrained models.

The robot takes images with its cameras, reports the results using speech and enters the dialogue mode, depending on the response mode selected. The user operates and controls the entire system, and is able to provide feedback using the robot's speech recognition. It should be noted that while it might be theoretically possible to perform the object recognition calculations on the NAO robot itself, its computational capabilities are insufficient for yielding results fast enough, due to the requirements of the Caffe framework and especially Multibox.

As for the actual software architecture, the data flows and interoperability between the developed modules described in this chapter are shown in Figure 12.

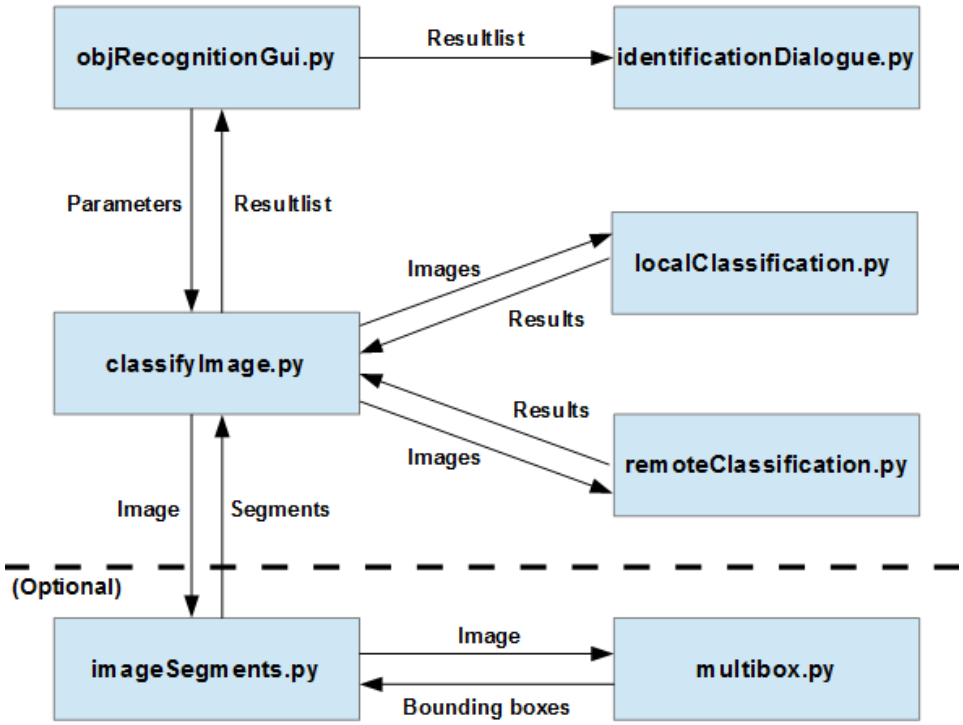

Figure 12: Software composition and data flows of the implementation. Components beneath the dashed line are only used when Multibox is required. All corresponding scripts for the implementation's Multibox version are marked with a `_multi` appendix in their file names.

`objRecognitionGui.py` serves as starting point, passing user selected parameters and commands to `classifyImage.py`, which serves as central hub for the object recognition pipeline. Depending on whether multiple objects should be recognized, it forwards images or image segments calculated by Multibox to the classification functions provided by `localClassification.py` and `remoteClassification.py`. The identified labels for the objects are collected in a list and returned to `objRecognitionGui.py`, which activates the dialogue mode if selected by the user.

Due to Multibox requiring additional computational capacities and dependencies (especially Caffe2), the decision was made to provide one version of the implementation without and another one with Multibox.

Regarding modularity, it is entirely possible to use the implemented object recognition pipeline starting with `classifyImage.py` without the provided user interface and incorporating it into other systems for use with NAO. Interpreting the diagram in Figure 12 as a tree with `objRecognitionGui.py` as its root, all children of `classifyImage.py` could be used entirely without NAO or exchanged for other object recognition algorithms, as long as they implement the same interfaces.

An element not shown in Figure 12 is the file `objRecogConfig.py`, containing the configurations described in 6.7. While all modules need certain information in this file, this should pose no hindrance to modularity, as it is simply a collection of parameter values.

6.9 Example Use Case

After booting the NAO robot and connecting to it via Ethernet or Wifi, the user will first have to make sure that the configuration file `objRecogConfig.py` contains the correct robot IP address and that the service entered for the remote classification is reachable. It is recommended to use the Monitor tool alongside the object recognition and load the camera viewer plugin in order to be able to see from the robot's viewpoint.

The implementation can then be started by launching the script `objRecognitionGui.py`, which will automatically connect to the robot and initialize the user interface. The user is then able to move the robot's head so that the object to be identified is within its field of view, which can be checked with Monitor. Afterwards, parameters for adjusting the object recognition can be set. Clicking the "Identify" button will start the object recognition process, which might take some time, depending on whether the calculations are carried out on the GPU or the CPU, and whether multiple objects have to be identified. The robot will announce the start of the recognition process acoustically, if "Silent" was not chosen as response mode.

After the object recognition is completed, the results are written to the terminal and depending on the response mode, the robot will additionally tell the user the class label recognized with the highest confidence for the object acoustically, or start a dialogue as described in 6.5. This completes the recognition process, and another object can be identified.

Shutting down the implementation should be done by clicking the "x" in the GUI window, so that the head joint can be unstiffened correctly, in order to prevent damage to the robot.

7 Tests and Experiments

7.1 Overview

This chapter describes the tests and experiments conducted to evaluate different characteristics of the implementation, the most important one being recognition rates. A point to note about the evaluation of the recognition rates is a certain subjectivity regarding when exactly a recognition result is precise enough to be regarded as correct. An example would be whether the recognition result set `[('electronic equipment', '0.76110'), ('modem', '0.65957'), ('equipment', '0.63811'), ('personal computer', '0.54178'), ('machine', '0.53449')]` calculated for a modem counts as positive result concerning Top-1 scores, or whether `'vessel'` is not specific enough to count as correct classification of a water bottle.

Therefore, a notion for the required precision of a more abstract or high-level result to count as correct has to be defined. Given a CNN C, a specific object O_0 and a corresponding object label L_0, where L_0 identifies O_0 with the highest amount of precision possible, i.e. L_0 represents the ground truth. Consequently, L_0 enables us to meaningfully interact with O_0, or draw conclusions from its presence in the observed scene. All of these possible interactions or conclusions are part of the set S_0. S_0 can be imagined as a knowledge base specifically matching L_0.

The label L_{rec} is the result of forwarding an image of O_0 to C. L_{rec} may potentially be less precise than L_0. L_{rec} is defined as being precise enough to count as true positive recognition, if L_{rec} enables most of the interactions and conclusions in S_0.

To continue the example above, let O_0 equal a water bottle, L_0 accordingly equal `'water bottle'` (which is within the trained categories of C) and L_{rec} equal `'vessel'`. While L_{rec} is less precise than L_0, it is still possible to conclude from L_{rec} that O_0 might contain a liquid or to handle O_0 in a certain way without spilling the content, etc. Thus, `'vessel'` usually is a sufficiently precise label for O_0 and can be counted as positive recognition.

In contrast, with O_0 and L_0 as before, but L_{rec} equal 'item', L_{rec} would not be precise enough, as 'item' does not give us any information specifically linked to a water bottle.

This notion is still somewhat subjective, because it is hardly possible to completely enumerate every S_n for each corresponding object O_n used in the tests. The required precision also depends on the task at hand. For example, if the robot simply has to differentiate between

animals and man-made objects, identifying a water bottle as `'vessel'` is precise enough. Yet it is not, if the robot has to fetch the water bottle among an assortment of soda, wine and other bottles. Nonetheless, the notion defined here should be adequate for practical considerations, and help to keep the evaluations consistent.

However, it will only be applied to more abstract recognitions. Recognition results, which are within the same object class, but incorrect due to being too specific, will not be counted as positive recognitions. While the label L_0 equal `'Golden Retriever'` might have a nearly identical S_0 compared to an L_{rec} equal `'Dalmatian'`, it is still incorrect and the maximally accurate result type is supposed to handle this event appropriately.

7.2 Preprocessing-Test

7.2.1 Purpose

The purpose of this test was to examine the effects on the recognition rate when applying different preprocessing techniques to images prior to categorization by the CNN, both for the maximally accurate and specific results. A technique found to be effective would have been incorporated into the existing object recognition implementation for NAO. Especially enhancing overall image quality with a Gaussian blur or stronger contrasts was expected to improve recognition rates.

7.2.2 Data

The original data used for the test consists of a wide variety of manually labeled images divided into three categories by their sources.

The primary data set comprises 59 images taken with NAO's camera. Photographed were different available objects, for example electronic equipment (modems, computer mice, mobile phones etc), pens and fruit. Images were taken from different angles, distances and varying lighting situations. Additionally, differing amounts of background clutter are present in the images, as the implementation on NAO is expected to work outside of laboratory conditions. The images have a resolution of 640 $\times$ 480.

For further comparisons of the preprocessing techniques, two secondary data sets are used in this test. The first consisted of 34 random images downloaded from the Internet, again

picturing some everyday objects, but also vehicles and animals, with resolutions ranging from 224 × 203 to 2560 × 1920 and a good image quality.

The second set contains 34 pictures taken with a mobile phone camera and a resolution of 2448 × 3268. This data set is similar to the primary one, again with everyday objects, background clutter etc. Pictures in this set were deliberately taken with a low quality both to test the effectiveness of the preprocessing techniques and the robustness of the object recognition.

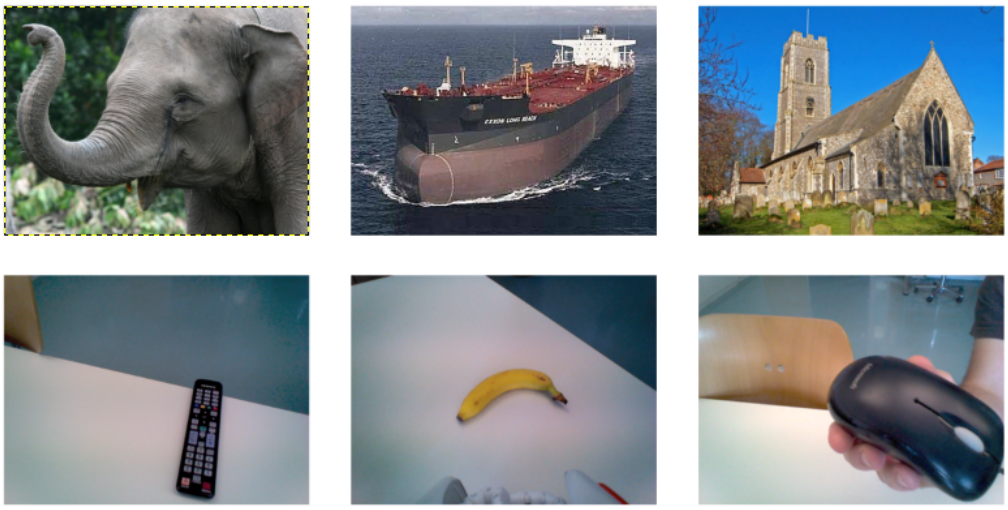

Figure 13: Example images from the download (upper row) and NAO (lower row) data sets.

As NAO's camera is able to take images in varying resolutions, using different ones throughout the data sets was intended in order to observe possible differences in recognition rates.

All three data sets contain a small percentage of objects, which are not within the categories trained by the CNN, to evaluate how these would be categorized.

7.2.3 Experiment

The entire test procedure was conducted with scripts written in Python. The first step was to apply the preprocessing techniques separately to each image, resulting in a new image for each technique. For this, the `Pillow` module, a fork of the Python Imaging Library (PIL), was used [18].

Selected methods were: Enhancing the contrast (`PIL.ImageEnhance.Contrast(image)`)

with a parameter value of 3, enhancing the sharpness (`PIL.ImageEnhance.Sharpness(image)`) again with a value of 3, applying a Gaussian blur (`PIL.ImageFilter.GaussianBlur()`) and converting the image to greyscale (`PIL.ImageOps.grayscale(image)`). All three data sets now contain the original images and each of their respective four converted variants. An example is given in Figure 14.

Figure 14: Original image (from the NAO data set) and its contrast enhanced, sharpness enhanced, greyscaled and blurred conversions.

Next, the data sets were passed to the object recognition implementation and all images including the originals were locally categorized. The CNN used for the test procedure was GoogleNet, as it is the neural network used by default for local identification in this work. Results (accurate as well as specific) of the categorization for each image were written to files in human readable format and evaluated by viewing the output for every image. An example for the raw recognition output, in this case for the original image of the remote control in Figure 14, is shown in Figure 15. For each image, both maximally accurate and specific result sets were calculated by the CNN. The total number of result sets is 590 for the NAO images, 340 for the downloaded and 310 for the phone images. Each result set contains the five most confident classifications for the given image and result type.

7.2.4 Results

The results obtained in this test indicate that none of the selected preprocessing techniques improved the categorization noticeably and consistently across all data sets. Result quality

```
Results for remote-2015_10_06-12_56_05.png:

Accurate:
[('remote control', '1.72425'), ('device', '0.56717'), ('electronic equipment', '0.45337'),
('equipment', '0.39847'), ('machine', '0.30824')]
Specific:
['n04074963 remote control, remote' 'n02666196 abacus'
 'n03075370 combination lock'
 'n04372370 switch, electric switch, electrical switch' 'n04125021 safe']
```

Figure 15: Raw recognition output for a remote control, which evaluated to a true positive result. The number values for the accurate results indicate the confidence value (defined by Caffe), the number values for the specific results are the class index of the respective label.

was highly dependent on the exact input image. The exact recognition results are shown in Table 1, Table 2 and Table 3.

Table columns labeled with "Original" show the recognition results for the unmodified images, "Contrast", "Gaussian Blur" etc the results for images preprocessed with the corresponding technique. The line label "Top-5 positive" means that a correct classification was among the five top results returned by the CNN. "Better" means the number of images, which were classified incorrectly in the original version, but correctly in the preprocessed one. For example, this would be the case for an image with the ground truth L_0 equal `'lemon'`, which was falsely recognized with L_{orig} equal `'potato'` in the original version, but correctly with $L_{contrast}$ equal `'citrus fruit'` in the contrast enhanced version. "Worse" accordingly denotes the number of images, which were incorrectly classified after being preprocessed, but correctly in the original version.

NAO data set	Original	Contrast	G. Blur	Greyscale	Sharpness
Total image count:	59	59	59	59	59
Top-5 positive (accurate):	28	29	26	20	27
Recognition rate (accurate):	47%	49%	44%	34%	46%
Better (accurate):	N/A	6	0	0	0
Worse (accurate):	N/A	5	2	8	1
Top-5 positive (specific):	30	32	26	23	31
Recognition rate (specific):	51%	54%	44%	39%	53%
Better (specific):	N/A	6	0	1	2
Worse (specific):	N/A	4	4	8	1

Table 1: Results for the NAO data set. Total image count: 295

Recognition rates for the images originally taken with NAO's camera, shown in Table 1, indicate that none of the preprocessing techniques lead to a significant overall improvement in

recognition rates for the data set. Applying Gaussian blur or enhancing the sharpness had little effect at all on the recognition. Gaussian blur led to a decline of 7% for maximally accurate and 13% for maximally specific results, compared to the original images. Sharpness had a similar effect, with the two improved specific results offset by two incorrect classifications, each one for specific and accurate values.

Higher blur rates than those used in this test lead to nearly unrecognizable images, while high sharpness leads to the unwanted creation of structures on the image, which caused some images to be recognized e.g. as `'jigsaw puzzles'`.

Using a greyscale unsurprisingly led to clearly inferior results with recognition rates of only 34% and 39%, as colour is an important factor for identification of some objects.

Enhancing the contrast resulted in significant changes, compared to other techniques. However, the positive and negative changes offset each other with 21% improvement versus 18% deterioration for accurate results (i.e. 6 better recognitions versus 5 worse, compared to the 28 positive recognitions for the original data set) and 20% versus 13% for specific results. While those numbers add up to a slight improvement at the bottom line, the higher unpredictability regarding the outcome of the object recognition is not desirable. Furthermore, the gain is inconsistent, as can be seen in the results for the secondary data sets.

Higher contrast values are even more detrimental, because similar to high sharpness values, they can lead to the creation of new structures inexistent in reality, or to an overemphasis of background clutter, resulting in unusable classifications.

The recognition rates for the downloaded data set in Table 2 show the smallest changes after preprocessing compared to the other data sets, with less than 5% differences (0 or 1 divergent recognitions) for all values except the greyscale ones. Applying the greyscale again led to an overall decline, except for two improved maximally accurate results. Noteworthy is that no preprocessing technique led to a single improvement in specific results for the downloaded data set. This indicates that the CNN works best on unmodified images with good quality, similar to those it was trained on.

Classifying the data set originating from images taken with the mobile phone camera yielded similar results to those of the NAO data set, as can be seen in Table 3. Greyscale and enhanced sharpness led to small changes both to the better and the worse within 17%. The Gaussian Blur had no effect at all on the outcome, probably due to the combination of a much higher resolution of this data set's pictures and the subsequent downsizing by the CNN, from 2448×3264 pixels to 224×224 pixels in case of GoogleNet. Again, enhancing

the contrast had the strongest effect, this time however deteriorating the recognition rates with 9% better versus 30% worse recognitions for accurate results (again compared to the number of images originally classified correctly), respectively 13% versus 17% for specific results.

Downloaded data set	Original	Contrast	G. Blur	Greyscale	Sharpness
Total image count:	34	34	34	34	34
Top-5 positive (accurate):	26	26	25	27	25
Recognition rate (accurate):	76%	76%	74%	79%	74%
Better (accurate):	N/A	1	0	2	0
Worse (accurate):	N/A	1	1	1	1
Top-5 positive (specific):	25	24	24	21	24
Recognition rate (specific):	74%	71%	71%	62%	71%
Better (specific):	N/A	0	0	0	0
Worse (specific):	N/A	1	1	4	1

Table 2: Results for the downloaded data set. Total image count: 170

Mobile data set	Original	Contrast	G. Blur	Greyscale	Sharpness
Total image count:	34	34	34	34	34
Top-5 positive (accurate):	23	18	23	19	22
Recognition rate (accurate):	68%	53%	68%	56%	65%
Better (accurate):	N/A	2	0	0	0
Worse (accurate):	N/A	7	0	4	1
Top-5 positive (specific):	12	11	12	11	12
Recognition rate (specific):	35%	32%	35%	32%	35%
Better (specific):	N/A	3	0	1	1
Worse (specific):	N/A	4	0	2	1

Table 3: Results for the mobile phone data set. Total image count: 155

The lower recognition rates for the NAO and mobile phone data sets in comparison to results reported for GoogleNet in [41] are partly a result of the lower image quality (most pronounced with the mobile phone data set) and the distance to the objects, which will be further elaborated in 7.3. This was intended, as using easily classifiable images would not have allowed to observe whether the applied preprocessing effectively improves recognition rates. Additionally, some objects, which were not within the training set of the CNN, were purposefully included in the test.

In conclusion, due to the declines occurring throughout the recognition rates, no preprocessing steps for the images were incorporated in the current implementation. This experiment

further confirms that CNNs need no additional preprocessing of input images, as intended by their design.

7.3 Robustness-Test

7.3.1 Purpose

This test was conducted to verify the effectiveness of the implemented object recognition solution on the NAO robot under circumstances like varying distances, lighting and image resolution. In contrast to the preprocessing-test in 7.2, this test focuses on these controlled environment parameters. The overall recognition rates were expected to be similar or slightly inferior to those presented for example in [41] and to deteriorate significantly because of declining image quality (due to lower light) and increasing distance to the object.

7.3.2 Data

All images used for this test were taken with the robot's camera. Altogether, there are 66 different images, partitioned into four data sets by environment parameters. The pictures contain electronic equipment, fruit, and other everyday objects. Again, some objects not contained within the CNN's trained classes were included. This simulates that the robot will sometimes have to identify such untrained object classes in a non-laboratory environment. Background clutter was reduced, though not completely omitted in order to retain a certain non-laboratory environment.

The primary data set consists of 42 images of 14 objects, each taken from close (approximately 15 cm - 20 cm), medium (approximately 75 cm) and far distances (approximately 150 cm). The images were taken with good lighting and a resolution of 640 $\times$ 480.

Three secondary data sets were created to analyze the effect of the varying environment parameters. For these sets, the same seven objects were used over close to far distances.

The second data set contains 21 images taken in lower light compared to the primary data set. This had a noticeable negative effect on the image quality through higher Gaussian noise, though all objects remain clearly visible. Pictures were taken of seven objects over close to far distances and a resolution of 640 $\times$ 480.

A similar data set with 21 pictures in a darker environment was used as well, with the same

objects, distances and resolution used in the low light data set. This time, the images taken by NAO's camera appear to be in greyscale due to the darkness, although the objects are still visible. Though practical use of NAO under those lighting conditions is unlikely, the experiment was meant to test the limits of the object recognition.

The last data set, again with 21 pictures, was taken in good lighting and with a resolution of 1280 $\times$ 960, which is the highest available for NAO's camera.

Figure 16: Example images from the data sets: Pen in close distance (primary set), water bottle in far distance (low light set), modem in medium distance(dark set).

7.3.3 Experiment

The images were created by placing the objects in the defined distances in front of the robot's camera. The images in low light and darkness were taken by darkening the room accordingly.

As in 7.2 images were forwarded to the implemented local classification using GoogleNet. The accurate and specific identifications were again manually evaluated.

7.3.4 Results

Further images of a wider range of test objects would have been desirable, however the possible object selection was restricted by the CNN's trained categories. Additionally, a large amount of these categories are vehicles and animals, which were not available in the laboratory for obvious reasons. Therefore, the selection was further restricted to objects at hand. Nevertheless, objects of this kind will probably be relevant in future object recognition tasks for NAO.

The combined results for accurate and specific classification are shown in Table 4. For each object, three pictures were taken, each of which yielded four result values: Two top-5 and

two top-1 results for both accurate and specific classifications. Consequentially, there are 168 results for the primary data set, 84 each for the secondary data sets, altogether 504 results. In Table 4, "Top-5 positive" means that a correct classification was among the five top results returned and "Top-1 positive" means that the first result was correct.

Data set:	Primary	Low Light	Dark	High Res.
Object count:	14	7	7	7
Top-5 results:	84	42	42	42
Top-5 positive:	38	23	9	21
Top-5 negative:	46	19	33	21
Top-5 error:	55%	45%	79%	50%
Top-1 results:	84	42	42	42
Top-1 positive:	28	13	3	14
Top-1 negative:	56	29	39	28
Top-1 error:	67%	69%	93%	67%
Total result count:	168	84	84	84
Total positive:	66	36	12	35
Total negative:	102	48	72	49

Table 4: Results for accurate and specific classifications.

Although the error rates with respective top-5 errors of 55%, 45%, 78% and 50% in Table 4 appear high, the breakdown for top-5 results for close, medium and far distances in Table 5 shows that recognition rates are much better when the object of which an image is taken is in close proximity to the robot, except for the dark image set. Top-5 error rates for close distances lie at 25%, 7%, 79% and 21%. While the overall recognition rates are worse than those reported in other works like [41], there is a high probability that this is due to the much lower image quality offered by NAO's camera compared to the images provided in the ImageNet database. Additionally, the error rates would have been lower, if the object selection would have been strictly restricted to trained object classes.

Considering these points, the recognition rates in close distances are satisfactory. As expected, objects with colours contrasting the background clutter or with distinct shapes like a remote control are recognized well, while rather monochromatic, smooth objects like a computer mouse or reflective ones like a smart phone pose more of a difficulty for the object recognition.

The results obtained from the low light and dark image set were better than expected, considering the low quality of the images. An added challenge when identifying images taken in very low light or dark conditions are objects for which the colour is an important factor

Data set:	Primary	Low Light	Dark	High Res.
Object count:	14	7	7	7
Results per distance:	28	14	14	14
Close positive:	21	13	3	11
Close negative:	7	1	11	3
Close error:	25%	7%	78%	21%
Medium positive:	16	10	5	7
Medium negative:	12	4	9	7
Medium error:	43%	29%	64%	50%
Far positive:	1	0	0	3
Far negative:	27	14	14	11
Far error:	96%	100%	100%	79%

Table 5: Breakdown of top-5 results for the different distances.

during recognition, e.g. fruits like oranges and lemons. Objects with a distinct form, such as bottles, were recognized better.

A comparison between the primary and the high resolution sets' results is inconclusive. While the recognition seems to benefit from higher resolution on longer distances, the limited data sets available can not reliably confirm this assumption.

An interesting detail is that the angle, at which an object is observed, potentially has a big effect on the outcome of the recognition, especially on really close distances, where the object fills a major part of the image. For example a remote control, which is usually easily recognized by the CNN, was falsely classified as keyboard when in close distance. This case happened for example in the low light data set and explains the otherwise unexpected increase in accuracy from a top-5 error of 78% to 64% when comparing close and medium distance results.

Overall, the results for the recognition rates show that the implemented solution operates well, given good environmental parameters.

7.4 Multibox-Test

7.4.1 Purpose

To evaluate how well Multibox [28] is able to detect objects on images taken with NAO, another experiment was performed. Of interest were the number of objects found, and

whether certain object categories were noticeably easier or more difficult to detect. This test was kept rather concise, because the overall effectiveness of Multibox itself is already well-documented in the source given above.

7.4.2 Data

The data set is composed of 20 pictures taken with NAO's camera. Similar objects to the ones in the other tests' data sets were used, however this time obviously multiple objects are present in each picture. The exact number varies between 1 and 5 objects. Additionally, different backgrounds and partial occlusion of objects were used on some images.

Distances to the objects are between close to medium range, similar to 7.3. All the pictures have a resolution of 640 x 480.

7.4.3 Experiment

The images were forwarded to a modified version of the Multibox script described in 6.6.5. However, the version of the algorithm used for this test does not return the bounding boxes' coordinates, but draws the boxes directly onto the images and saves them to a directory for later evaluation, which was again done manually. Only the detections with the five highest confidence ratings will have bounding boxes assigned, and multiple boxes around the same object are possible.

During this test, the images were not passed to the CNN for identification, as the focus lay solely on the Multibox algorithm's ability to detect objects present on the images taken by NAO's camera.

7.4.4 Results

For evaluation purposes, only bounding boxes, which enclosed the target objects reasonably well were counted as positive detections. Neither boxes enclosing large regions of the image with multiple objects in them, nor detections of "clutter" objects were counted, an important example being the robot's arms. Both of these cases can be seen in the second picture of Figure 17.

Figure 17: Images from the data set with the calculated bounding boxes.

The results show good detection rates for the data set, depicted in Table 6. Except for two cases, at least half or all of the target objects present in the pictures were found by Multibox. Small objects (in relation to image size) like pens or smooth and reflective objects like smart phones proved more difficult to detect or may have been assigned a detection confidence to low to be among the five top results. While Multibox is able to return more than the five top results, the larger result sets would be to unwieldy for practical use in this work.

As mentioned in 6.6.5, Multibox is computationally demanding and requires a GPU for good performance. During this test, the algorithm was run using only the CPU, with the detection taking several minutes per image, which is far too long for practical usage. However, as with Caffe, running Multibox on a CPU or a GPU makes no difference in the calculated results.

Image ID:	08_00	33_24	37_02	54_15	11_04	33_56	34_48	36_46	37_13	37_55
Object count:	2	4	2	3	3	4	4	2	1	2
Objects detected:	1	3	2	3	2	1	2	1	1	1
Detection Rate:	50%	75%	100%	100%	67%	25%	50%	50%	100%	50%
Image ID:	38_08	50_28	51_43	52_01	52_21	53_31	54_34	54_51	55_13	56_50
Object count:	2	3	3	3	3	5	3	4	4	3
Objects detected:	1	1	3	3	2	3	2	3	2	3
Detection Rate:	50%	34%	100%	100%	67%	60%	67%	75%	50%	100%

Table 6: Results for the Multibox-test.

Overall, Multibox is an effective addition to the implemented object recognition pipeline when run on appropriate hardware. It not only allows to recognize multiple objects at once, but also to gain information about where those objects are located in the picture. This could be useful for future applications.

8 Conclusion

8.1 Summary

In this work, a way for implementing a robust, ready-to-use object recognition on the NAO robotic platform has been shown, using CNNs based on pretrained models provided by the Caffe framework. The CNNs are treated as black boxes, with the input being the images taken by NAO, and the output being the result sets containing the classifications for maximally accurate and maximally specific labels. The implementation has a modular architecture with cleanly defined interfaces, enabling reuse, and the pretrained models can be exchanged against others if needed.

For identifying several objects on a given image, the Multibox algorithm developed by Google researchers was added to the image recognition pipeline, marking the image regions with the highest probabilities of containing a viable object with bounding boxes. From those bounding boxes, the coordinates for image segments are derived and forwarded to the actual classification methods.

The implementation is made accessible to the user by means of a graphical user interface, offering several options to adjust the recognition process and controlling movement of the robot's head in order to easier acquire objects in the field of view. The interactivity between human and robot is further enhanced with a dialogue system for querying additional results.

To evaluate the resulting recognition rates and analyze possible improvements through image preprocessing, several tests and experiments with data sets mainly taken with NAO's camera were conducted.

8.2 Discussion

While the installation process of Caffe and Caffe2 has its tricky parts, the object recognition with pretrained models could be seamlessly integrated into a solution working together with the well-documented NAO robotic platform.

In order to ensure good performance considering the time needed for identification, the object recognition should be run on hardware suited to the task, preferably with a decent GPU, so that the slow CPU-mode can be avoided. Especially Multibox proved to be very demanding regarding the hardware.

The user interface and the dialogue system turned out to be effective improvements for interactivity and ease of use of the system, both during the development and testing phases, as well as for a new user.

As for the actual recognition rates, testing yielded good results over reasonable distances depending on the object in question and the lighting, although larger test data sets would have been desirable. While results were inferior to those officially reported for example in [39], the decline in recognition rates can be attributed to the lower image quality offered by NAO's cameras, compared to the images available in the ImageNet database. Multibox, while having high computational demands as mentioned before, proved to be an effective addition in the respective test. An issue were the multiple bounding boxes assigned to the same object, resulting in duplicate recognition runs and corresponding decline in performance.

Regarding the enhancement of image quality, none of the tested preprocessing methods achieved a clear improvement, with the resulting recognitions being highly unpredictable and dependent on the exact preprocessed picture forwarded to the CNN.

With that said, the implementation is still expected to perform satisfactory in terms of recognition rates and as a potential supporting part for future projects using object recognition.

8.3 Outlook

Based on the current work, a multitude of improvements and further additions to the implementation are imaginable.

First and foremost, the implementation would benefit from a way to improve recognition rates. A part of this undertaking would be improving the image quality itself. While acquiring better cameras is probably not feasible, unless a version of the NAO head with superior ones is released by Aldebaran, further research could be conducted towards preprocessing. While the results found in this work seem to indicate that none of the techniques, which were applied yielded overall better results, it is of course not excluded that certain procedures, or a combination thereof, can be found to achieve this goal.

Another way of generally improving the implementation regarding recognition rates could be to use models trained on larger data sets with more categories, or even using models specifically trained on images taken with NAO. This might be rather difficult however, considering that the pretrained models used in this work were already trained on a data base of more than one million images.

The detection of objects could also be enhanced through some additions to Multibox, like suppressing duplicate detections, or using a leaner and faster solution for region proposals altogether, if one exists.

As for future work, object recognition opens up a lot of possibilities. It enables the robot to better analyze its environment and interact with it. An interesting addition would be a system capable of remembering classifications of objects and their position in the world. Together with the tree-like structuring of the object classes, this allows semantic understanding and meaningful interaction if a data base of actions possible with specific objects or their characteristics would be available, without having to set these for every single object class. The data base could be similar to ontologies used in artificial intelligence. For example, actions possible with the object class "bottle", like lifting one without spilling the content and pouring it into a glass, are probably applicable to most kinds of bottles, regardless of them being wine or water bottles, and could be carried out after successful recognition.

While this might seem far-fetched, it only goes to show the possibilities offered by effective and robust object recognition in combination with robotic platforms.

References

[1] Alvisionrecognition. http://doc.aldebaran.com/2-1/naoqi/vision/alvisionrecognition.html#alvisionrecognition. Last accessed: 2015-12-22.

[2] atexit documentation. https://docs.python.org/2/library/atexit.html. Last accessed: 2015-12-22.

[3] Caffe. http://caffe.berkeleyvision.org/. Last accessed: 2015-12-22.

[4] Caffe tutorial. http://caffe.berkeleyvision.org/tutorial/. Last accessed: 2015-12-22.

[5] Caffe web demo. https://github.com/BVLC/caffe/blob/master/examples/web_demo/app.py. Last accessed: 2015-12-22.

[6] Caffe web demo tutorial. http://caffe.berkeleyvision.org/gathered/examples/web_demo.html. Last accessed: 2015-12-22.

[7] Caffe2 repository. https://github.com/Yangqing/caffe2. Last accessed: 2015-12-22.

[8] Convolutional neural networks for visual recognition. http://cs231n.github.io/convolutional-networks/. Last accessed: 2015-12-22.

[9] Convolutional neural networks (lenet). http://deeplearning.net/tutorial/lenet.html. Last accessed: 2015-12-22.

[10] Deepdream - a code example for visualizing neural networks. http://googleresearch.blogspot.de/2015/07/deepdream-code-example-for-visualizing.html. Last accessed: 2015-12-22.

[11] Imagenet large scale visual recognition challenge. http://image-net.org/challenges/LSVRC/2014/index#data. Last accessed: 2015-12-22.

[12] Instant recognition with caffe. http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/00-classification.ipynb. Last accessed: 2015-12-22.

[13] Multibox repository. https://github.com/google/multibox. Last accessed: 2015-12-22.

[14] Nao humanoid robot platform. http://www.istec.org/wp-content/uploads/2013/11/Datasheet_H25_NAO_Next_Gen_EN.pdf. Last accessed: 2015-12-22.

[15] Nao video camera. http://doc.aldebaran.com/2-1/family/robots/video_robot.html. Last accessed: 2015-12-22.

[16] Naoqi api. http://doc.aldebaran.com/2-1/naoqi/. Last accessed: 2015-12-22.

[17] Naoqi concepts. http://doc.aldebaran.com/2-1/dev/naoqi/index.html. Last accessed: 2015-12-22.

[18] Pillow (pil fork). http://pillow.readthedocs.org/en/3.0.x/. Last accessed: 2015-12-22.

[19] qimessaging api. http://doc.aldebaran.com/2-1/dev/python/qimessaging.html. Last accessed: 2015-12-22.

[20] Retrieving images. http://doc.aldebaran.com/1-14/dev/python/examples/vision/get_image.html#python-example-vision-getimage. Last accessed: 2015-12-22.

[21] Theano. http://deeplearning.net/software/theano/. Last accessed: 2015-12-22.

[22] Torch. http://torch.ch/. Last accessed: 2015-12-22.

[23] Who is nao? https://www.aldebaran.com/en/humanoid-robot/nao-robot. Last accessed: 2015-12-22.

[24] Pablo Barros, Sven Magg, Cornelius Weber, and Stefan Wermter. A multichannel convolutional neural network for hand posture recognition. In Stefan Wermter, Cornelius Weber, Włodzisław Duch, Timo Honkela, Petia Koprinkova-Hristova, Sven Magg, Günther Palm, and AlessandroE.P. Villa, editors, *Artificial Neural Networks and Machine Learning – ICANN 2014*, volume 8681 of *Lecture Notes in Computer Science*, pages 403–410. Springer International Publishing, 2014.

[25] Dan C. Cireșan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification.

In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1237–1242. AAAI Press, 2011.

[26] Jia Deng, Jonathan Krause, Alex Berg, and Li Fei-Fei. Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3450–3457, Providence, RI, USA, June 2012.

[27] Stefan Duffner and Christophe Garcia. Robust Face Alignment Using Convolutional Neural Networks. In *International Conference on Computer Vision Theory and Applications (VISAPP 2008)*, pages 30–37, January 2008.

[28] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks. *CoRR*, abs/1312.2249, 2013.

[29] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[30] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014.

[31] Philipp Krähenbühl and Vladlen Koltun. Geodesic object proposals. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, pages 725–739, 2014.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[33] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989.

[34] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):276–279, 1995.

[35] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, and Francesco Nori. The icub humanoid robot: An open platform for research in embodied cognition. In *Pro-*

ceedings of the 8th Workshop on Performance Metrics for Intelligent Systems, PerMIS '08, pages 50–56, New York, NY, USA, 2008. ACM.

[36] Giulia Pasquale, Carlo Ciliberto, Francesca Odone, Lorenzo Rosasco, and Lorenzo Natale. Real-world object recognition with off-the-shelf deep conv nets: How many objects can icub learn? *CoRR*, abs/1504.03154, 2015.

[37] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[38] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[39] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, pages 1–42, April 2015.

[40] Niko Sunderhauf, Sareh Shirazi, Adam Jacobson, Feras Dayoub, Edward Pepperell, Ben Upcroft, and Michael Milford. Place recognition with convnet landmarks: Viewpoint-robust, condition-robust, training-free. In *Robotics: Science and Systems*, Auditorium Antonianum, Rome, July 2015.

[41] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

Appendix

Content of accompanying CD

The current work is documented on an accompanying CD with the following content:

- The present bachelor's thesis as PDF file.
- The user manual for the developed implementation as PDF file.
- The Python project containing the files necessary for running the implementation. Does not contain Caffe/Caffe2 files.
- Snapshots of the referenced web sources at the time of writing as PDF files.

List of Tables

List of Figures