HTBLuVA Wiener Neustadt
Höhere Lehranstalt für Informatik

DIPLOMARBEIT

Lego Recognition Tool

Ausgeführt im Schuljahr 2017/18 von:

Christine ZEH 5CHIF-25

Simon BABOVIC 5CHIF-3

Betreuer / Betreuerin:

Dr. Michael Stifter

Wiener Neustadt, am 31 March 2018

Abgabevermerk:

Übernommen von:

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Wiener Neustadt, am 31 March 2018

Verfasser / Verfasserinnen:

Christine ZEH

Simon BABOVIC

Contents

Abstract

This diploma thesis' goal was, to develop a machine, which is capable of identifying Lego bricks and sorting them into boxes. At first, there had to be thought of ways to successfully identify a piece of Lego. The idea of using a scale arose.

The analysis of the weight information, which is stored in the Bricklink database, brought up, that with the weight, a part can be identified with an accuracy of at least 90%. As it turned out the weights in the database are not accurate, the focus shifted to identifying Lego parts with cameras and the scale from then on has been used as assisting function. The next idea was to work with neural networks and image classification.

As for a neural network capable of distinguishing between all Lego parts huge computing power and loads of training data is needed, the choice was to limit the number of distinguishable parts to five. Three cameras are used in different angles, to achieve a good all-around view of the part. As it turned out, that the amount of training data needed even for such a small neural network is still enormous, it was decided to again shift the focus and to concentrate on reducing the amount of the required training data. This was done by a process called image preprocessing. In this process the pictures taken from the part get reduced to only necessary information, by for example removing the background of the image.

To make this process easier, a prototype camera box was constructed, which fits well for image preprocessing. Additionally a transporting system for the Lego parts should have been developed, but as this turned out to be too complex for an IT diploma thesis, it got replaced by developing an user interface. It's task is to visualise the process of learning and image classification to the user. For this, a server was developed, which, because of JavaScript being limited to Websockets, acts as a bridge between Websockets and TCP streams.

Furthermore, a method was developed to automatically generate training data for the neural network, as the manual creation of it is enormously time consuming. It is called the virtual learning method. With this method, training data is created by virtually making pictures of three-dimensional Lego models. It requires further development as it is in an early stage, but it is most likely to bring the project to production, as it removes the need of creating training data manually.

Although the main goal was not achieved due to the underestimation of mechanical complexity and lack of time and budget, yet this diploma thesis provides a firm basis for further development.

Kurzfassung

Das Ziel dieser Diplomarbeit war es, eine Maschine zu entwickeln, die Legosteine identifizieren und in Boxen sortieren kann. Zu aller erst musste überlegt werden, wie eine Maschine so einen Stein erfolgreich erkennen kann. Die Idee eine Waage dafür zu verwenden entstand.

Die Analyse der Gewichtsinformationen in der Bricklink-Datenbank ergab, dass man mithilfe des Gewichts eines Legosteins diesen mit einer Genauigkeit von mindestens 90% identifizieren kann. Als jedoch erkannt wurde, dass die Gewichtsinformationen in der Datenbank nicht exakt sind, wurde der Fokus darauf gerichtet, Legosteine mithilfe von Kameras zu erkennen. Die Waage wird weiterhin in assistierender Funktion verwendet. Die nächste Idee war, ein neuronales Netz zur Bilderkennung zu verwenden.

Weil ein neuronales Netz, dass zwischen allen Legoteilen unterscheiden kann, sehr viel Rechenleistung und riesige Mengen an Trainingsdaten benötigt, wurde entschieden die Zahl der erkennbaren Teile auf fünf zu limitieren. Es stellte sich heraus, dass selbst für so ein kleines Netz extrem viele Trainingsdaten benötigt werden. Der Fokus wurde nun darauf gelegt, die Menge der gebrauchten Trainingsdaten zu reduzieren. Das wurde durch Bildvorbearbeitung erreicht. Dabei werden vor der Bilderkennung Störfaktoren wie zum Beispiel der Hintergrund entfernt und das Bild auf das wesentliche reduziert.

Um diesen Prozess zu vereinfachen, wurde ein Prototyp einer Kamerabox entwickelt, der sich gut zur Bildvorbearbeitung eignen soll. Zusätzlich sollte noch ein Transportsystem für Legosteine entwickelt werden, jedoch stellte sich schnell heraus, dass dies zu komplex für eine Informatik Diplomarbeit ist. Also wurde dieser Teil mit der Entwicklung einer Benutzeroberfläche ersetzt. Die Aufgabe dieser Oberfläche ist es, dem Benutzer die Abläufe des Lernens und der Bilderkennung zu visualisieren. Für diesen Zweck wurde ein Server entwickelt, der, da JavaScript auf Websockets limitiert ist, als Brücke zwischen Websockets und TCP Streams agiert.

Des Weiteren wurde eine Methode, um Trainingsdaten automatisch generieren zu lassen, entwickelt, da das manuelle erstellen von Trainingsdaten extrem viel Zeit in Anspruch nimmt. Die Methode wurde "Virtual learning method" genannt. Mit ihr werden Trainingsdaten erzeugt, in dem virtuell Fotos von dreidimensionalen Modellen von Legoteilen gemacht werden. Sie benötigt noch Entwicklungsarbeit, ist aber am wahrscheinlichsten der Schlüssel zum Erfolg dieses Projekts, da sie die Notwendigkeit von manueller Erstellung von Trainingsdaten beseitigt.

Auch wenn das Hauptziel dieser Diplomarbeit durch das Unterschätzen der mechanischen Komplexität und durch Zeit- und Budgetmangel nicht erreicht wurde, so bietet sie trotzdem eine solide Basis für zukünftige Entwicklungsarbeit.

Chapter 1

Introduction

Author: Christine Zeh

Everyone has already played with Lego bricks before. It is the largest toy company in the world and has a yearly revenue of 5.1 billion Euro.

Figure 1.1: The Lego logo after the redesign in 1998.[1]

1.1 History

In 1932 Ole Kirk Kristiansen establishes his company to build household supplies and wooden toys. The companies name is Lego, from the Danish words "Leg Godt" which means "play well". 1949 Lego rolled out a precursor of the bricks known nowadays, called "Automatic Binding Bricks". They were available in two different sizes and four colours. Over the next few years an increasing number of different brick moulds were invented and in 1955 the "System of Play" was released with it's first set (see figure 1.2).[2]

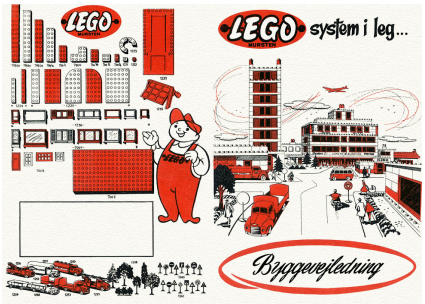


Figure 1.2: The Lego catalogue with the System of Play in 1955.[3]

[1] *Lego logo.*
[2] *Lego.*

In the year 1966 706 million bricks are produced and sold in 42 different countries all over the world. In the 1980's 70 percent of all families, with children under 14 years old, possess some sort of Lego bricks.[4] And this remains true up to the present day. With Lego Films, Lego Worlds, Lego Video games and Lego sets of nearly every movie, city or scene one can think off, there's something for everyone. This is why adult people are still fascinated by the possibilities when creating with Lego.
Nathan Sawaya was the first artist to use Lego bricks to express himself. Since 2001 he has been building his sculptures containing between 100.000 and up to multiple million bricks. In figure 1.3 one of his creations is pictured.

Figure 1.3: A sculpture named "Red" from Nathan Sawaya built out of Lego bricks.[5]

1.2 Bricklink

At some time around 2000, buying Lego sets in the store wasn't enough for a programmer from Hawaii and he created the website Bricklink. It enabled people to sell their old sets, bricks and anything linked to Lego. Approximately 830.000 member can shop from 11.000 shops selling on this website.[6] Some of them are individuals, but for most of them it's their main profession.

A completely new work area has emerged, where old bricks are bought, sorted and resold. This professional field could emerge because of three main reasons.

- It is not important when the brick was made, because Lego parts are downward compatible.

[3] *Brickfetish.*
[4] *Lego.*
[5] *Nathan Sawaya.*
[6] *Bricklink.*

- 485 billion bricks have been printed since the company started and there's a high chance of discovering old bricks at home
- A higher price is paid if the bricks are sorted

That's why online and backyard flea markets are looted to find good maintained and cheap Lego. After enough Lego parts are gathered up they need to be differentiated per part an colour. But to earn valuable profit thousands of them have to be sold. The sorting of tons of bricks is therefore necessary.

Assuming a human needs three seconds to sort one Lego piece, 1000 parts take 50 minutes. Further assuming each workforce gets paid ten Euro per hour, the sorting of one Lego piece costs 0.8 cents. This amount is fairly high, considering the average selling price for one part is about three cents. To reduce the cost per part the process of sorting should be automatised.

1.3 Task

During this diploma thesis a prototype for the autonomous recognition and semi automatic separation is developed. The operation is administered through a user interface. With the interface multiple part groups can get managed. Each group can contain up to five parts to classify simultaneous. Parts not associated with another part are presented accordingly.

The software is designed to allow for autonomous sorting in the future. Therefore to simplify the construction of a compatible mechanical system, concepts to realise the sorting mechanism are developed. These concepts are introduced in the conclusion in chapter 6.

1.4 Technologies

The idea of building a sorting machine is not new. In fact in the the last few years many different Lego sorting systems have been developed. They range from performing simple tasks to very complex ones. The two most developed ones on the internet were realised by Richard Chow[7] and Jacques Mattheij.[8]

Sorting with the machine built by Richard Chow is done entirely mechanically. The parts pass gaps on a conveyor belt. If the parts are small enough they fall into the hole onto the below conveyor belt or in a bucket. Another differentiation can be achieved by passing under a obstacle. If the brick is to high, it slides down to get to the correct path to its associated bucket.

The machine from Jacques Mattheij is a more software based construction. Multiple months he has been working and improving his software. Since last year his machine can sort similar groups of bricks or if wanted colour groups. His program identifies the bricks through a neural network.[9] This approach is only made possible a few years ago due to an achievement made by Geoffrey Hinton (in chapter 4 the subject is dealt with).

[7] *Sort Your Legos Like an Engineer.*
[8] *Jacques Mattheij.*
[9] Mattheij, "How I Built an AI to Sort 2 Tons of Lego Pieces".

Although these developments fulfil their task well and precise, these implementations are not necessarily suitable for the mass market.
To name some reasons:

- To perform mechanical sorting enough space has to be available. The sorting machine has measurements of 1.5 x 3 meters and if more than standard bricks need to be sorted, multiple machines have to be set up.
- With mechanical sorting the expansion is time consuming, because new sorting mechanism need to be developed.
- The implementation created by Jacques Mattheij is expensive. To perform fast enough classification a GTX1080 Ti Nvidia video card[10] at a price of 700 Euro is needed.
- At this point the second solution sorts parts in similar groups. With this solution a further sorting is indispensably.

Our intention is the development of a low budget system to perform the identification task only on a few parts. This low price allows to put multiple machines in succession and therefore enables the sorting of a very diverse repertoire of brick moulds. For this reason the costs are an essential factor throughout this diploma thesis.
Further on a user interface to administer the processes simplifies the handling. This enables the expanding and controlling of systems even for untrained operators.

[10] *GTX1080 Ti Nvidia card.*

Chapter 2

Assisting functions

To be able to identify parts, data has to be gathered and transmitted to the classifying function. This is the purpose of the assisting functions. This area covers the database with the detectable parts and example images, the scale and it's associated interface and the information extraction from the images.

2.1 Database

Author: Christine Zeh

A database is accumulation of data. It is managed by a Database-Management-System (DBMS), that standardises every form of adaptation and access. It provides languages to store, manipulate and request the information. Relations between information can be represented in a more efficient way. This is achieved through tables, which represent relational elements with shared characteristics. Each table can be related to multiple other tables. To guarantee uniqueness of an element, a primary key can be used. Furthermore, with this key the relation between two elements is identified.[1]

2.1.1 Database structure

The task of the database is to provide all possible detectable parts and the saved neural networks with its associated categories and parts. This is achieved by three interconnected tables (illustrated in figure 2.1). The biggest one contains all possible parts, which have the correct size to be detected. The table *Saved Networks* contains the path to the save networks. It is referenced by the table *Detectable Parts*, which consists of the parts that can be classified by the network.
The website Bricklink manages the biggest database of existing Lego parts.[2] Based on this database the table containing all parts is constructed. Most of the bricks get an assigned part number from Lego which is moulded on the brick. If no part number was assigned or additional information is needed to obtain uniqueness Bricklink assigns or extends the part number. The assigned part number can be a constant or a descriptive name. In illustration 2.2 the different ways of naming parts is outlined.[3]

[1] Hillebrand, *Datenbanken und Informationssysteme.*
[2] *Bricklink.*
[3] *Bricklink Naming Scheme.*

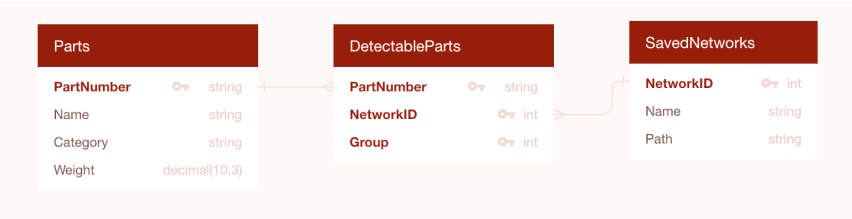


Figure 2.1: The tables with their descriptive attributes and the relations between them.

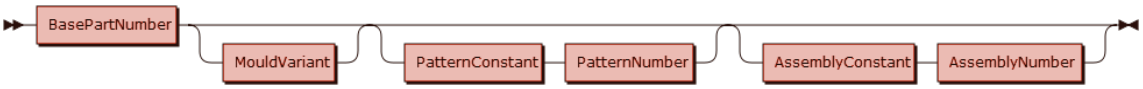


Figure 2.2: EBNF diagram of the part numbering scheme

Base Part The base part number is 4-5 digits long and moulded on the part or a constant or descriptive name assigned by Bricklink.

Mould If different types of moulds exists, different moulds are followed by a letter assigned in consecutive alphabetical order.

Pattern If different pattern types exists, each gets an assigned number, preceded by a constant.

Assembly If the part consists of multiple components, the prefix 'c' is appended, followed by a sequential number

2.1.2 Reducing the number of parts

The initial table from Bricklink consists of 47347 elements from 199 categories. Only distinctive bricks should remain in the final table.

First of all, categories containing special parts are located. The removing of this categories included around 10000 elements.
In the next step around 1000 parts without weight are deleted.
The biggest deletion consisted of all parts with special patterns or moulds. If a part had a associated base part it is deleted. This removed again around 28000 pieces.
In the end around 8000 parts are left. This reduction ensured a more simplified search, although with a closer inspection even more parts could be excludes.

2.2 Scale

Author: Simon Babovic

As Lego parts come in a huge variety of sizes and shapes, it makes perfect sense to use every property of a piece to identify it. One such property is weight. To measure the weight of a Lego part, a scale had to be bought. But at first, some research has been made to determine the type of scale to buy.

2.2.1 Weight Groups

To save time and resources (e.g. CPU time), the first idea was to sort out Lego parts by weight before they run through image-classification. The image-classifier is dealt with in chapter 4. All existing Lego parts can be grouped in 1829 weight-groups and the ones which are represented with an undefined weight in the parts database. These undefined weight parts are mostly stickers, which are not part of the project so they got removed from the database. The average size of each group is 24, with a maximum size of 2771 pieces. There are 756 Lego parts which can be sorted out without image-classification in theory, due to their unique weight in the database. To get an overview of the parts and their weight, a graphical comparison of weight-groups and the amount of parts within this weight-group was created (See 2.3).

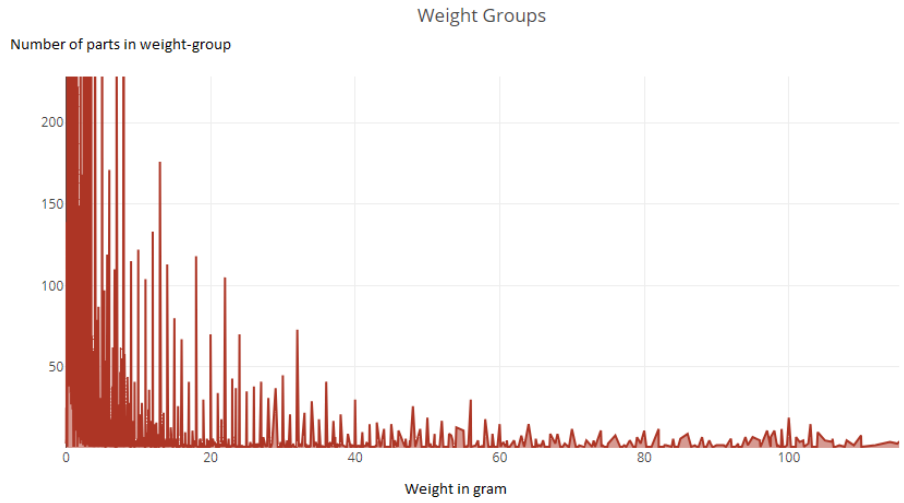


Figure 2.3: All groups zoomed into origin, because the plot would be to big to show the entire one.

As figure 2.3 shows, there are much bigger weight groups at the the very small weights. This implies that most of the parts are very lightweight. This has to be taken adequately into account when selecting a scale. As figure 2.3 makes almost no statement for the majority of the parts, a new plot was created to take a look at the very tiniest parts of the database (See figure 2.4). This graph clearly shows where most of the parts cluster. In the tiny spectrum of 0.5 gram to 1.5 gram there are at least 9000 parts. This greatly depicts the incredible amount of variety in Lego parts.
Now a question comes up. When there are so many parts with almost the same weight, does it even make sense to pre-sort them with scaling? Yes.

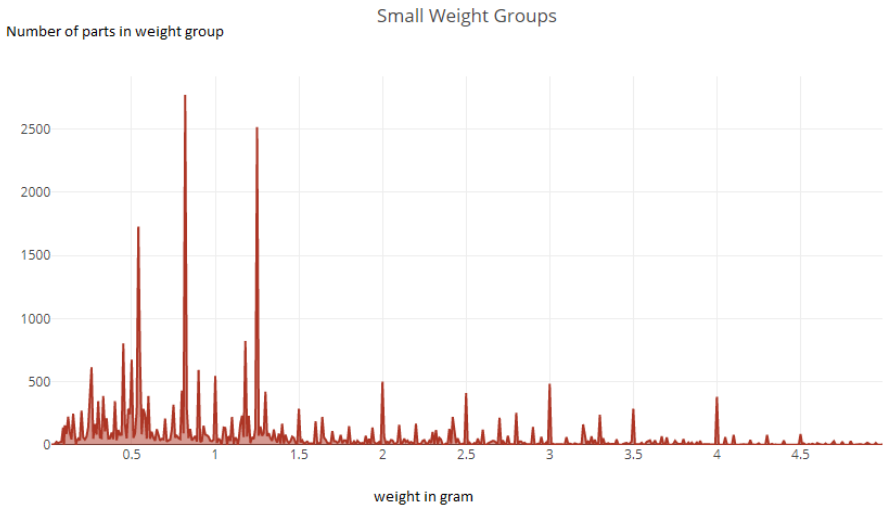


Figure 2.4: Weight-groups with a weight of 0 to 5 gram.

2.2.2 The Advantage of pre-sorting

To illustrate the big advantage of the scale, relative numbers will be used instead of absolute numbers. As mentioned above, the average size of a weight-group is 24. With a total of about 30000 parts in the database, this means the the complete dataset is reduced to 0.08% of it's size. Even in the worst case with the largest weight-group, the reduction lies at 9.2%, which saves massive amounts of CPU-time.

2.2.3 The Scale used to identify the weight group of a part

Each weight-group differs in a minimum of 0.01 gram. To obtain the most accurate and stable results, a precision scale was chosen (see figure 2.5). It provides an Ethernet Adapter to stream the data. To determine when the data is transmitted, different options are available.

Stability Sends data automatically upon stability

Accepted Range Automatically outputs data if it's in a specific range

Continuously Repeatedly sends data as fast as possible

Periodic Sends data after indicated time

To achieve the most exact results, the Stability option is used.

Figure 2.5: Ohaus Scout skx622 precision scale

2.2.4 The Problem with pre-sorting

After a few tests were made with the new scale, some new problems emerged. It came out, that the weights in the database are not precise. Furthermore problems with very old Lego parts got detected. It seems Lego parts lose a perceivable amount of weight after a while of use through deterioration. This led to the decision not to pre-sort Lego parts and instead provide the weight to the part classifier as additional information. The more input data a classifier gets, the better.

2.2.5 Software

Process

A Python program opens a connection to the scale and awaits the transmission of strings. The string contains useless characters, which need to be removed. Afterwards with a database the weight-group with the specific weight is returned and presented in a table (see figure 2.6).

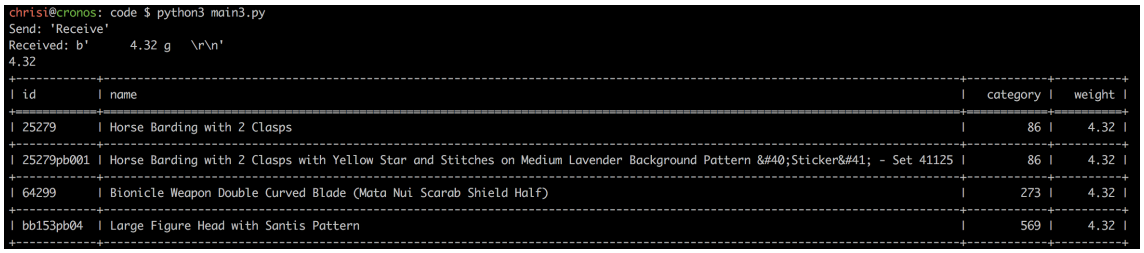


Figure 2.6: Output of the program.

Code

With the library async.io a connection to the scale is opened. To start receiving data, a message has to be send firstly. Then, in an endless loop, the program waits for a responds. If it receives a response, the data is striped to filter out the number. This number is then used to create the table.

```
1     reader, writer = yield from asyncio.open_connection('169.254.1.1', 9761, loop=loop
      )
2
3     writer.write(message.encode())
4
5     while(True):
6         data = yield from reader.read(100)
7         print('Received: %r' % data)
8
9         try:
10            data = data.strip(' \t\n\r g')
11            num = round(float(data), 2)
12            print(num)
13            getTable(num)
14        except ValueError:
15            pass
```

2.3 Image Processing

Author: Christine Zeh

To improve the speed and correctness of the classification unnecessary information has to be removed. Multiple libraries provide functionality to perform this task. With these libraries different algorithms get applied to the image to improve the legibility of structural information.
The image preprocessing works through six to seven steps

- Changing the Colorspace
- Smoothing the Image
- Image Thresholding
- Morphological Transformation
- Removing static disturbing objects
- Removing white borders
- Reducing the Size

2.3.1 OpenCV - Open Source Computer Vision Library

This open source library has the aim at real-time computer vision. The library was officially launched in 1999 by Intel[4] with the intention to offer a joint infrastructure and disseminate vision knowledge by making their project accessible for everyone. It provides prebuilt methods for all kind of types of working with graphics. 3D reconstruction, video analysis, image processing and machine learning are only a few sectors they provide Libraries to.

2.3.2 Changing the Colourspace

"Colour space is a three-dimensional geometric space with axes appropriately defined so that symbols for all possible colour perceptions of humans or other animals fit into it in an order corresponding to the psychological order.[5]"
Colours get represented by distinct numbers, which vary for each colourmodel. Through changing the colour space the discrimination of colours can be improved. To remove the information of color for different Lego-pieces the image is converted to a greyscale image. In the greyscale colourspace intensity substitutes the colour information. The intensity per colour can be calculated with the brightness of red, green and blue, the primary colours in the RGB Model (see fig 2.7).[6]

[4] *Intel Corporation.*
[5] Kuehni, *Color Space and Its Divisions: Color Order from Antiquity to the Present.*
[6] Ibid.

(a) RGB colourspace **(b)** greyscale colourspace

Figure 2.7

2.3.3 Smoothing the Image

Smoothing is used to reduce noise or pixelation of an image. Most smoothing methods use a low-pass filter as a basis. It reduces high-frequency information, which decreases the disparity between pixels. Through this process, the next steps will have a stronger effect. For the implementation of the lower disparity, a kernel is convoluted to the image. Through changing kernel size the smoothing effect intensity is adapted.

The convolution is done between the kernel, which is flipped by row and column, and an image piece.

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\right)[2,2] = (i{\cdot}1)+(h{\cdot}2)+(g{\cdot}3)+(f{\cdot}4)+(e{\cdot}5)+(d{\cdot}6)+(c{\cdot}7)+(b{\cdot}8)+(a{\cdot}9) \tag{1}$$

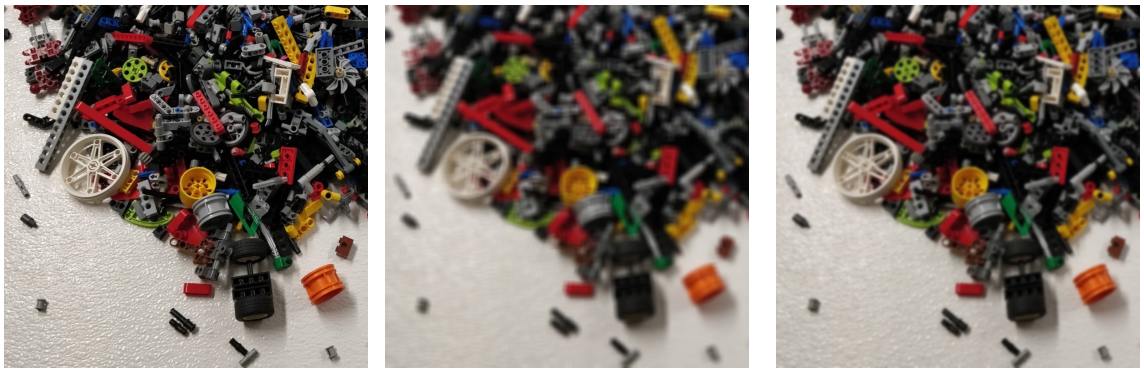
Each values on the equivalent position in the matrices get multiplied and the outcome is added up. The value of the centred pixel of this section gets the resulting number.

The weights of a Gaussian kernel are arranged to be equivalent to calculating a weighted average of the corresponding image pixels. Using box- or disk-shaped kernel strong ringing and truncating effects are created, therefore are not well suited[7] (see illustration 2.8). Typical structures for the three kernel mentioned can be

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

3x3 Box-shaped kernel 5x5 Gaussian kernel 9x9 Disk-shaped kernel

[7] Burger and Burge, *Principles of Digital Image Processing: Advanced Methods*.



(a) Original Image **(b)** box-shaped kernel **(c)** Gaussian kernel

Figure 2.8: Comparison of different kernels with (a) as original image, (b) outlines the truncating effects of a box-shaped kernel. At image (c) a Gaussian kernel is applied.

2.3.4 Image Thresholding

"Thresholding is a process of converting a grayscale input image to a bi-level image by using an optimal threshold.[8]"
The aim is to extract objects from the background. Thresholding algorithms classify in two groups. The different types are demonstrated in figure 2.9.

Global Thresholding Algorithms

Global thresholding algorithms use a single threshold for the whole image. This is only applicable if object and background values are fairly consistent.

Adaptive Thresholding Algorithms

Better results with variable lighting or areas are ensured through the adaptation of the threshold value for different areas in the image. The local threshold can be calculated by either the average values of the neighbours or the accumulated values weighted with a Gaussian window.[9]

By using adaptive thresholding only the outlines of the Lego Piece are maintained to represent the structure of the part.

2.3.5 Morphological Transformation

To enhance recognizability and minimise disruption a kernel gets applied to the binary image. It is similarly structured to convolution. (see 2.3.3).

Erosion

Each element under the kernel is set to zero if not all pixels hold the value one. Pixels near boundaries get distracted, this allows the removing of small grain, also called white noise.

[8] Devi, "Thresholding: A Pixel-Level Image Processing Methodology Preprocessing Technique for an OCR System for the Brahmi Script".
[9] Ibid.

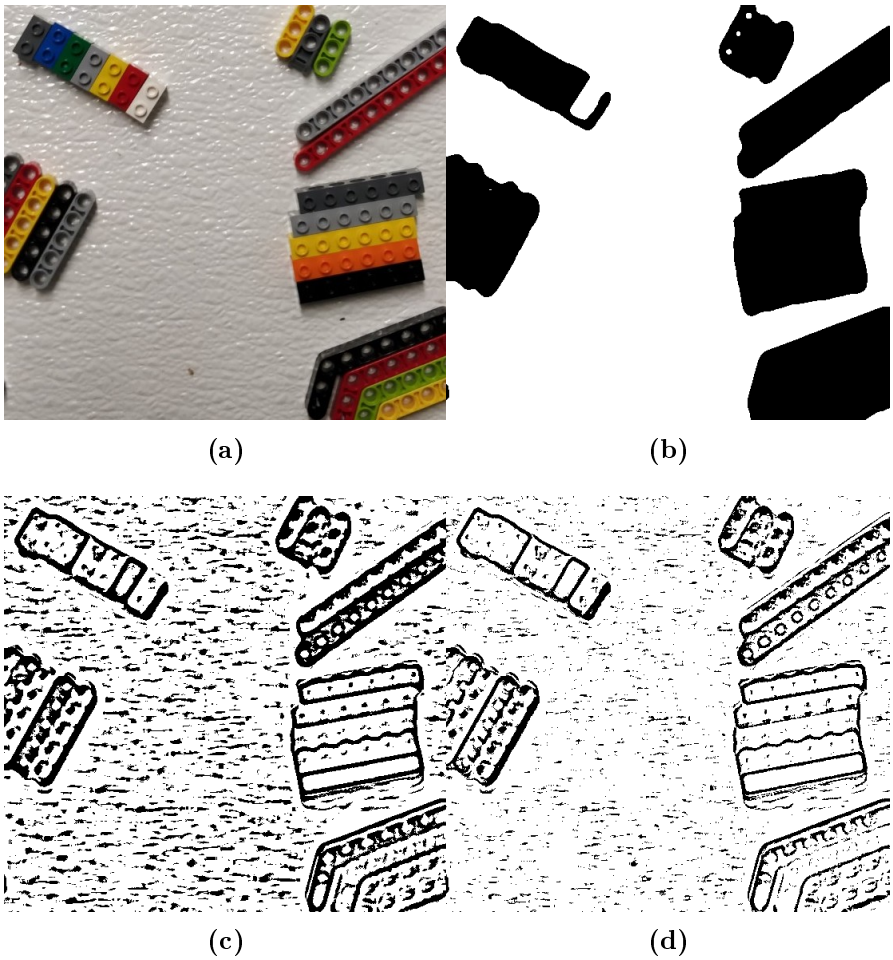


Figure 2.9: (a) is the original image with non-uniform illumination. Global thresholding with a value of 127 was applied to (b). An adaptive threshold with an average sum of the neighbours is used for (c). The Gaussian weighting of the neighbours for the threshold achieves the best result (d).

Dilation

Each element under the kernel receives the value one if a minimum of one pixel is one. The outcome is the increasing of foreground, which eliminates small black holes in white areas.

The Combination

Operation of the methods in succession can prevent altering the width and heights of objects (see 2.10). To remove small black spots from the image, after it was altered to binary colours by adaptive thresholding (see 2.3.4), Erosion was applied as a first step and afterwards Dilation was used.

2.3.6 Removing static disturbing objects

This step is easy to accomplish but has a major impact on the later identification of the part. It is possible to skip this process by removing the gaps in the measurement environment. This gap however, appears currently disruptive on the processed image.

(a) (b) (c)

Figure 2.10: Image (b) displays the removing of white noises from original image (a). Erosion and dilation consecutively lead to this result. Black points in the foreground (c) get removed by dilation followed by erosion (d).

As the object is static, pixel wise removing of the lines is possible. Through trial and error the coordinated of the start and end point has to be determined. When they are known, a white line can cover the disruption. As the camera angle can change slightly, the line is wide enough to balance the offset. In figure 2.11 a descriptive line is laid over the disruptive gap.

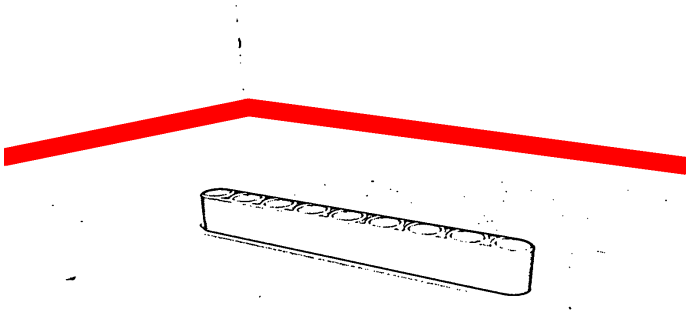


Figure 2.11: The removing of static lines can minimise the disruption and improve the result. For better recognisability the covering line is coloured red.

2.3.7 Removing white borders

In figure 2.11 an image of a brick captured in the measurement environment is illustrated. It can be seen that only a small section keeps the information. To minimise the uninformative space white borders are removed. This is achieved by counting black pixels in each row and column. If the count is less than a specific number the line is cut.
The possibility of cutting parts of a brick is accepted in favour to guarantee image noise, too heavy to be removed by the morphological transformation, is also cut away.

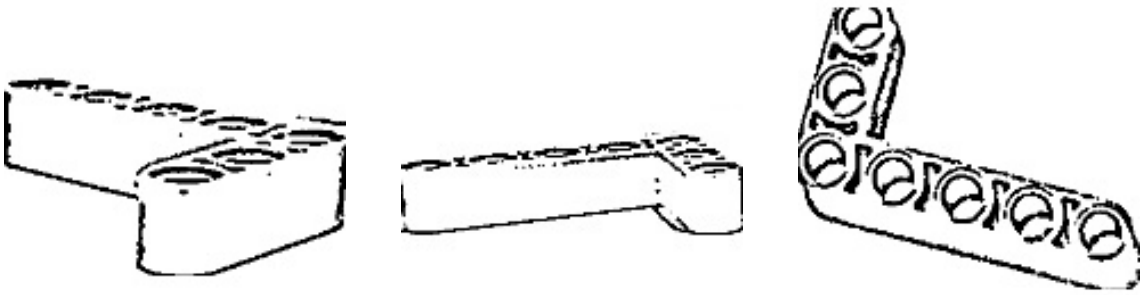
2.3.8 Reducing the Size

To ensure faster computation the size of the images is reduced to the minimal possible size. The height and width experienced as appropriate to store enough information is 150 pixels each. The image has the same height and width measurements as the placement of the part is random. Therefore it can't be anticipated which side has to be wider.

Although the received image doesn't has the measurements of a square. To prevent tearing or straining of the image, white borders are again added until the the desired shape is achieved. Afterwards it is shrunk.

2.3.9 Result

Through the steps of preprocessing the image is reduced to retain only relevant information. Only identification of the shape should be possible with a minimal amount of other information left (see 2.12).



(a) **(b)** **(c)**

Figure 2.12: The necessary steps to maintain only relevant information applied to all three used perspectives.

Chapter 3

Physical Structure

Author: Simon Babovic

As the project progressed, some answers to various questions had to be worked out, such as:

- What system should the part classification process run on?
- How many cameras are needed?
- How does a properly designed surrounding for image processing look like?
  - Shape
  - Colour
  - Illumination
- Where to place such a setting?
- If multiple cameras are used, how would they fit in the surrounding without interfering with it?

The big challenge is, to design the surrounding in a way that the software has no need for consuming big amounts of ressources, and at the same time to find common ground with simplicity.

3.1 System

The main aim for the project is to develop a well working part classification process with minimal financial effort. As Microsoft requires charged licences, the decision on the operating system fell on Linux based operating systems. For experimental purposes a Mac Book Pro 13" Late 2014 was used. A Mac Book or similar systems are clearly no low budget solution, so an alternative had to be worked out.

3.1.1 RaspberryPi 3

The RaspberryPi 3 is a microcomputer with an ARM chipset. It runs Raspbian as operating system, which is based on Debian. It has a processor with a speed of 1.2GHz and an onboard memory of 1GB. It is available online for 35€, so it definetly is the best low budget solution. In comparison with the Mac Book Pro 13" Late 2014 the RaspberryPi 3, in the speed's view, is clearly a poorer solution than the Mac. However, the microcomputer is a cheap solution for users who do not need the part classifier to operate at high speeds, so all the hardware used for the process is going to be compatible with the RaspberryPi 3.

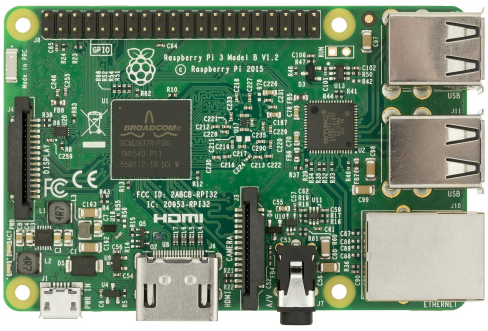


Figure 3.1: RaspberryPi 3

3.2 Surrounding Components

3.2.1 Number of cameras

As the cameras had to be bought to keep the project running, the most important decision which had to be made, was how many cameras should be used for image processing. It soon became clear, that there need to be two cameras to properly detect the shape of a Lego piece. As there is going to be no mechanical mechanism to place a Lego brick into a predefined standard position, a third camera was added to provide additional information to the part classifier.

3.2.2 Illumination

The most important property of a light to buy for the illumination of the setting is, that the lamp fills the complete setting with almost the same amount of light. Most commercial available light bulbs illuminate unequally in space. The easiest solution is, to use a LED spotlight, which gets connected to the setting at the top. Therefore the light also minimises the shadow casting of Lego parts.

3.2.3 Placement

At first, the plan was to build an surrounding around the scale, because this seemed to be the easiest solution. But after some attempts to build a provisional frame around the scale, it soon became clear that this idea was the easiest to think of, but not the easiest to implement. One of the reasons for this is, that the scale's top is too small to put three cameras around and preventing them to look at each other. This, the fact that the scale has colored parts and the highly reflective property of the scale's top, brings up the followinig solution.

Solution

The image processing setting and it's parts need to be well geared to each other. The scale is not a component of this setting, so it should not play a role in it. The goal is to construct a surrounding, which is homogenous of color and prevents light reflection, which can be a problem to the cameras.

The best way to achieve this is to construct a camera-box, to place it near the scale and to build a mechanism, which after scaling brings the Lego part to the camera-box.

3.3 Angle of view

"In photography, angle of view (AOV) describes the angular extent of a given scene that is imaged by a camera."[1]
The basis of the design is the angle of view of the cameras. So before making a first design attempt, the angle of view had to be measured. This was accomplished by pointing the camera to a flat surface a 90 degree angle, taping the borders of view and measuring the distance from the flat surface to the camera. Now, to find out the angle, a true to scale sketch got produced (see figure 3.2), and the angles were measured from the sketch.

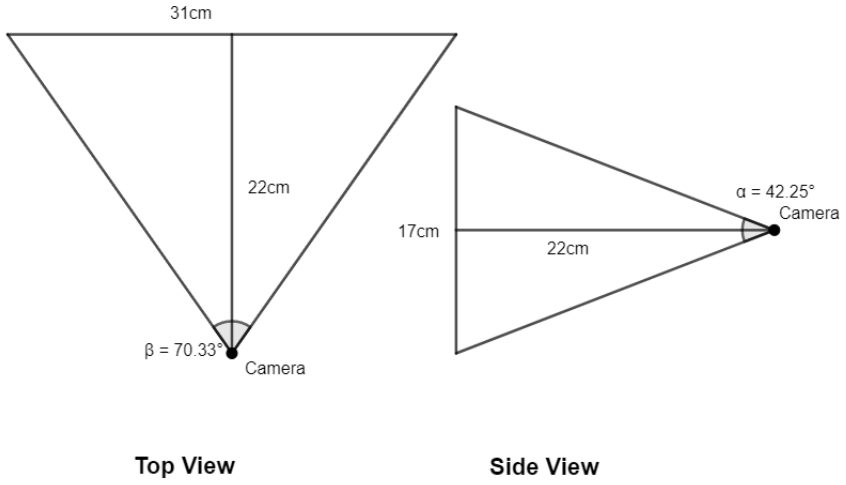


Figure 3.2: A true to scale sketch of the angle of view of the used cameras

After gaining knowledge about the angle of view of the cameras, everything was on place to design a camera-box which fits all the requirements listed above.

[1] *Angle of View.*

3.4 Camera

Author: Simon Babovic

As seen in chapter Scale (figure 2.3), there are some Lego-parts, which share the same weight-class with other bricks. To specify one piece of Lego out of all the other ones, a software should identify the form of it using cameras. While testing various cameras which were lying around in the robotics lab, it became clear that choosing a camera would be very challenging.

3.4.1 Problems

Illumination

The first experiment showed, that most of the cameras' frame frequencies did not match with the rate of the light emitted from a lamp with an alternating source. This caused a flickering in the transmitted images. For this problem three solutions had been drafted.

Finding a camera with adjustable frame frequency: This would be the easiest solution, but at this time there was no guarantee that there is a camera which meets all the other criteria and allows to change it's frame frequency at the same time.

Changing the source: In this solution, the alternating source would be replaced with a direct current like a car battery. This would solve the problem of a flickering light bulb and the software could work as it is supposed to.

Compensate with software: Another idea was, to take multiple pictures in short periods and to lay these pictures on top of each other to compensate the flickering effect which caused the software to not working properly.

Solution As it later transpired, the first solution is an option. As this one is the easier to implement than the software solution and changing the source would require further testing with a direct current, the first solution was chosen.

Image resolution

While testing with different pieces of Lego, it became clear, that low-resolution cameras have problems with little holes and nubs. The picture taken seems to be clear enough for image processing, but the shadow of the brick endowed uncertanty for the software about where - in a hole - the part ends and the background starts. After testing with higher resolutions (FullHD iPhone camera), this problem was solved. The ressource-expensive image processing in mind, further testing was initiated to see, if same results can be achieved with lower resolutions. But it seems to be the minimum requirement to have a FullHD (1080x1920) resolution for detecting holes in Lego bricks.

Integration

It must not be forgotten, that the whole system must be runnable on a RaspberryPi 3, so the cameras need to be compatible with it. As the RaspberryPi 3 has a completely different chipset than customary computers, there has to be a driver for the camera to enable it to run on this chipset. This hugely constraints the selection.

Price

Because the project runs on low budget, there is always an eye on the price of a component too. The cheapest solution would be the Raspberry Camera Module V2, but as always, there is a problem with the cheapest solution. This camera has a special connector, which makes it impossible to connect multiple cameras to the RaspberryPi at once, without buying a special splitter, which is sold on demand, which means, it has a delivery time of over a month. So other solutions had to be searched.

3.4.2 Chosen Product

The best camera to choose turned out to be the Logitech C922 (see figure 3.3). It carrys a driver which is compatible with the ARM chipset of the RaspberryPi, it has the option to programmatically andjust the image frame rate and with a price of 119€, it is a little bit expensive, but stays within the budget.

Figure 3.3: Logitech C922[2]

3.5 Camera-Box

Author: Simon Babovic

At first, there was only one idea on how to build a camera-box, namely to design every part of it and stamping the parts out of aluminium. There was just one problem: aluminium is strongly reflective. Light reflection is a big problem when it comes to image classification. The easiest solution was, to buy a non-reflective adhesive white foil and to face it to the parts of the camera-box.

3.5.1 Building the Box from scratch

As the baseplate is the most important part in the camera box, it became the first part to be designed.

The Baseplate

The first try was to minimize the cameras' dead angles and making the baseplate big enough to prevent the cameras looking at each other. As the angle of view of the cameras is very small, there are no 90 degree angles at the corners of the baseplate. This ensures minimal dead spots. (See figure 3.4).

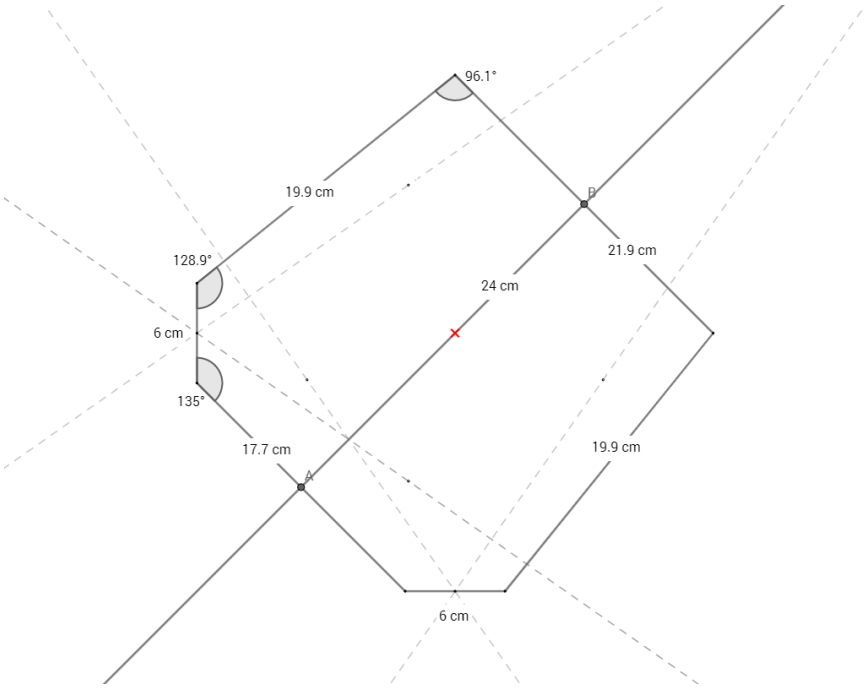


Figure 3.4: Baseplate draft: Solid lines represent the shape of the baseplate, dashed lines represent the angle of view of the cameras and the red dot in the middle represents the center, resp. the point at which the third camera is looking at.

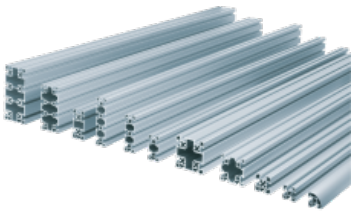
For the purpose of testing, before designing further parts, the baseplate got stamped out and adhered with the white non-reflective foil. The test results were great. The foil worked perfectly, the plate was big enough to prevent cameras looking at each other and dead angles were minimal. But after some time, a task based problem came up.

Scalability Problem

In order to maximize efficiency, the customer wants to add more and more machines to work simultaneously. This is a problem, because when the machine is not built of standardized parts, it becomes very hard to scale, causing the path from prototype to production to extend significantly. Because of this, and the fact that this project focuses on software, not construction, the idea of building the box from scratch got discarded. The new idea was to use BOSCH Profiles.

3.5.2 BOSCH Profiles

BOSCH Profiles are standardized parts, which come in various sizes and shapes (See figure 3.5a). With them, it is possible to build almost everything someone can think of. As shown in figure 3.5b, there are easy ways to construct such simple things like boxes. Even entire machines can easily be built of BOSCH Profiles only (See figure 3.5c). There is just one problem.



(a) basic BOSCH Profile parts

(b) small construction

(c) machine built of BOSCH Profiles only

Figure 3.5: Examples of BOSCH Profiles and their use.

The price of the parts needed to construct a camera-box with all the requirements would overrun the project's budget with a factor of three, so the idea had to be discarded too. But it was not a complete waste of effort scouring through catalogues searching for suitable parts.

From Prototype to Production

The advantages of BOSCH Profiles provide a facility to bring the prototype to production. The next idea was, to construct a camera-box from non-standardized parts, but keeping it simple, so that the prototype can easily be recreated with BOSCH Profiles, bringing the prototype to production.

3.5.3 Chipboard Box

The idea this time was, to buy white chipboard plates, cut them to size and bolt them together. The inner size of the box has to be about 20x25x25cm. The outer size is of no relevance as the image classification only takes place in the inner section of the box. Chipboard plates with a depth of 19mm and a white surface were chosen. After cutting them to size, the plates were bolted together. The baseplate attaches with hinges to the rest of the box as there needs to be an emptying mechanism.

(a) Raw camera-box without cameras attached to it.

(b) Camera-box with cameras and light attached to it.

Figure 3.6: The way of an empty box to a camera-box.

Chapter 4

Part Classification

Author: Christine Zeh

4.1 Recap

The measurement environment consists of the scale (see 2.2), the cameras (see 3.4), the right lighting (see 3.2.2) and the basic software, discussed prior.The basic software implies the provision of the weight by the scale and the preprocessing of the images (see 2.3). With this environment set up, the focus lies on an adequate classification method, which allow a precise and quick computation of the pictures data.

4.2 Measurement Procedure

In Figure 4.1 the process, to receive an estimated part number, is illustrated. Starting the next step with a key press is necessary to ensure the user had enough time to place the Lego part.

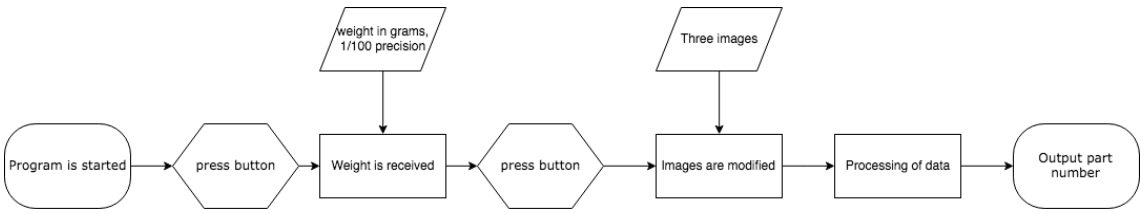


Figure 4.1: The Process to receive an estimated part number. A more detailed description to the modification of the images can be found in section 2.3. How the data is processed is going to be explained in this chapter.

4.3 Choosing the right approach

Depending on the weight and images from three angles, the part number of a Lego part should be identified. Different approaches can lead to this goal. Two types were taken into consideration.

4.3.1 Feature Detection

The difference or resemblance between two images can be determined by comparing their features. Features are sections or characteristics of an image which are distinguishable. Features can be either global or local. Colour and texture are examples for global ones. Interesting regions are described with local features like shape and orientation. For feature detection the same results have to be obtained with different perspectives, scaling or other discrepancies.[1] The features of the taken images are compared with an image library featuring all detectable parts. A reduction of the number of parts to compare can be achieved by checking the associated weight.

The error rate of this approach would be hard to minimise, because the shape of Lego pieces is not diverse enough to achieve good results. Through the weight this problem could be reduced, but the weight in the database is to inaccurate to delimit exact enough. With a huge effort a fairly satisfactory solution could be reached. High computational demand can still be an issue with weight ranges corresponding a lot of parts.

4.3.2 Learning Models

The concept of learning models is fairly easy. The program 'learns' from training data to predict prospective inputs. Therefore the specifications of the parts are detected automatically. The challenge of developing and implementing the right model to fit the task best can be a hurdle in the short amount of time available. But with the right model the number of wrong classified parts could drop to a few percent. This can only be accomplished with a lot of training data, but gathering this data is a rather big additional effort.

If more parts are needed learning models can be adapted for different pieces by training the model with the new data. The same implementation with feature detection can be more complex, because unique features have to be defined. Another favourable point, regarding learning models, is the continuous improvement through learning. With consideration of the expandability and necessary work required, for the two approaches, the decision was made for a learning model.

4.4 Machine Learning System

"A computer program is said to learn from experience with respect to some task and some performance measure, if its performance on the tasks, as measured by the performance measure, improves with experience[2]"

This means, the computer improves on a task, by identifying correlations and adapting certain values using the training data.
Machine learning can complete different tasks. For each of them a different set-up is most suitable.

Library

The biggest library to implement Machine Learning is Tensorflow, developed by Google.[3] It offers basic functions to build and use neural networks. The library used for this project is

[1] M. Hassaballah, "Description and Matching".
[2] Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*.
[3] *Tensorflow*.

called TensorLayer.[4] It is build on top of Tensorflow and provides modules for reinforcement and deep learning.

4.4.1 Different Machine Learning Systems

Supervised Learning This is the most used type. The system learns with training data already containing it's desired output, called the label.

Semi supervised Learning In the first step the system is trained without labels and a few samples containing labels determine how to classify this data. This can be used to name persons in a lot of pictures.

Unsupervised Learning Without the desired output accessible, the principal task of this learning method is to find correlations and anomalies.

Reinforcement Learning An agent observes an environment and chooses actions, on the basis of a policy. For every action it receives a reward or penalty. The policy gets adapted to minimise penalties. The desired output, can only be achieved with multiple actions in succession. If needed each can depend on prior actions, a possible scenario is a game like chess.

Semi supervised is a possible way to train the model, considering no labelled data is gathered. However, depending on someone placing the Lego parts, the data can get labelled simultaneously. This allows the use of supervised training, which can speed up training success.

Offline or Online Learning With a big amount of training data necessary to achieve satisfactory results, learning can take multiple days or weeks. By using online learning new data can be integrated in the system without relearning the whole model.
The Lego sorting system uses online learning, but not automatically. The model can be further trained with new data manually, if the correct identified parts drop.

Instance- or Model-Based Learning Instance-based learning measures the resemblance of data. Based on the training data, model-based learning systems make predictions. Both variants can be suitable for this project, but model-based learning seemed more promising.

4.4.2 Fitting the task

Most model-based systems use a mathematical function that fits the features of the input best. Features are the attributes the model has to set in correlation to compute the right output. Linear Regression is the simplest model to use on a task. It is in general a linear function,

$$y(x) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \cdots + \theta_n \cdot x_n \tag{3}$$

whereby the values of the parameters θ get changed to reach the smallest mean error. For more complex tasks a polynomial function can be used.

[4] *TensorLayer*.

In 2006 Geoffrey Hinton introduced an, until this time, impossible approach.[5] Using artificial neural networks solving more complex tasks is possible. These networks are used in a technique called deep learning.

Deep Learning

An artificial neural network consists of interconnected neurons. Each neuron receives input patterns and evaluates the output value with the aid of firing rules (illustrated in figure 4.2).[6] Each input gets multiplied depending on its weight. Then all inputs get added up with a bias value. The output value is computed with an activation function applied to this value.

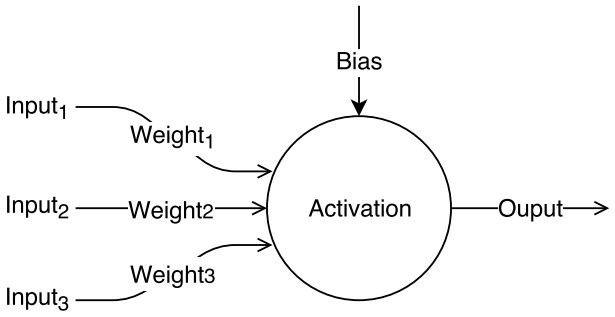


Figure 4.2: An artificial neuron receives input signals and evaluates the output value with the aid of firing rules

The most used activation function is the *rectified linear unit* (ReLU). The function receives the input and if it's positive, outputs the value of the input. If the input is negative it is set to zero. By forwarding a value of zero the weights can't be effective and the neuron dies. A circumvention is called Leaky ReLu, if the result is negative the output will be $y = 0.01 * x$.
In 2015 a new activation function was introduced called *exponential linear unit* (ELU).[7] It can reduce training time and enhance performance of the classification.

$$ELU_\alpha(x) = \begin{cases} \alpha(exp(x) - 1) & if x < 0 \\ x & if x > 0 \end{cases} \tag{4}$$

Like the ReLU activation, ELU outputs the input value if its positive. If x is negative it approaches the value of α, which is usually set to -1. This ensures the output is never zero. A drawback is the higher computation expense because of the exponential function, but this is compensated by the better convergence.

Neurons are arranged in layers, which are set in succession. The input layer receives raw information. It is connected to hidden layers, which receive the computed data one after another. Layers are called hidden if they are connected to neurons on both sides, that means they are not directly accessible. Multiple hidden layers try to identify structures and determine probabilities for the different output options (see figure 4.3).

[5] Hinton and Salakhutdinov, "Reducing the dimensionality of data with neural networks."
[6] *Neural Networks.*
[7] Djork-Arne Clevert, *Fast and accurate Deep Network Learning by Exponential Linear Units (ELUs).*

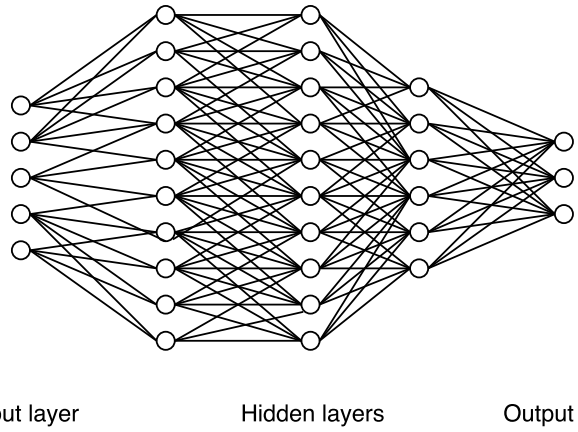


Figure 4.3: An artificial neural network consists of multiple layers. It tries to identify structures and determine probabilities.

Neural networks can differ in their structure, depending on their assigned task. To compute a simple number a small network can be sufficient but if the classification is computed on the basis of an image the network can be fairly deep and wide. Neural networks for image classification or natural language processing are called Convolutional Neural Networks (CNN).

4.4.3 Convolutional Neural Networks

According to a paper published by Kunihiko Fukushima,[8] the visual understanding in a human brain is done layer upon layer. Neurons of the lowest layer can only react to a small field of the visual input. Neurons of the second layer can only react to a small field of the reactions of the first layer. With each layer simple patterns can be combined to more complex ones (see figure 4.4).
This is adapted for CNNs using the convolutional layer. To obtain different perspectives of the receptive field, filters, kernels with either the value zero or one, get multiplied to the pixel. The output of all neurons using the same filter is called a feature map. It highlights the sections similar to the filter. A convolutional layer can apply multiple filters on its input and therefore has multiple feature maps to detect structures.
To reduce the size and the needed computational demand, pooling layers are put between the convolutional layer. This layer minimises the neurons by summarising a section. This is achieved with an aggregate function applied on a kernel.
With multiple layers in succession, the network is capable of classifying even very similar objects correctly. A big competition named ILSVRC ImageNet challenge ranks different neural network architectures based on their error rates on difficult image recognition tasks[9].[10]

4.4.4 Designing a test network

To speed up the development of the network and its components a test model is designed and is going to be expanded later. It performs a simpler task by classifying on basis of

[8] Fukushima, "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position".
[9] Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*.
[10] *ILSVRC ImageNet Challenge*.

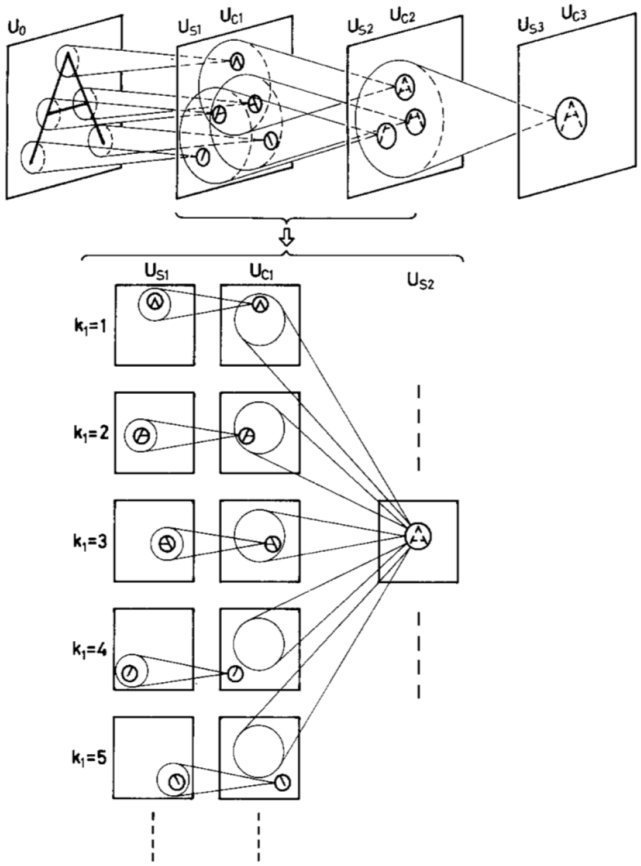


Figure 4.4: An example of the visual understanding of a human brain. A neuron can only react to a small field of visual inputs. The second layer can only react to a field of the first layer, but multiple layers in succession can identify complex structures. The picture was published in a paper written by Kunihiko Fukushima[11]

one perspective. Further on the testing takes only two parts with one image each into consideration. (The two parts are illustrated in figure 4.5). This allows the testing of more possibilities.

As beginner in this field the use of an already tested architecture seemed more reasonable, as designing a CNN architecture can be time-consuming and difficult. Therefore different existing networks were tested to find the best suitable.
The first architecture was the LeNet-5. It was created in 1988 by Yann LeCun and is the most known architecture for CNNs.[12] It consists of three convolution layers with two pooling layers. The network is efficient in categorising small and simple images (32x32 pixels), but is too shallow to detect complex structures. The differences of the sample parts could not be distinguished.
Last year's winner of the ILSVRC challenge is too complex to continue using it in the expanded version, this was the reason why the next tested architecture was AlexNet. It is the winner of the ILSVRC challenge in 2012. It consists of five convolutional layers and

[11] Fukushima, "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position".
[12] Y. LeCun and Haffner, "Gradient-based learning applied to document recognition".

(a) (b)

Figure 4.5: Lego parts used to test the different architectures.

only two pooling layer, which means it demands a lot of computational performance. But regarding the similarity of the Lego parts, AlexNet was the only possibility. After 840 training images a 97.5% accuracy was achieved (plotted in figure 4.6).[13]

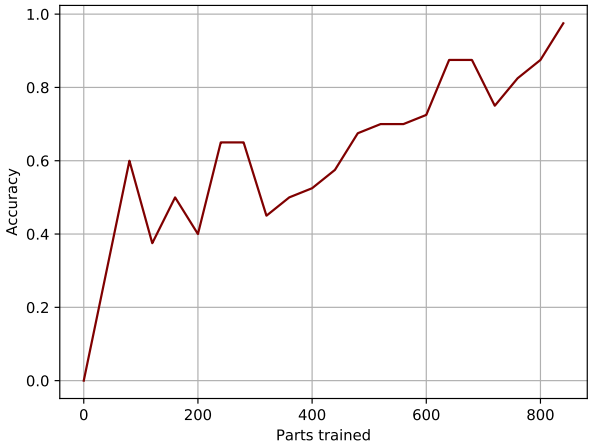


Figure 4.6: Training improvement of AlexNet tested with two static images. The accuracy is determined through 40 predictions.

Implementation

In table 4.1 the exact AlexNet architecture is described. The column *maps* declares the feature maps for each layer. The *size* specifies the amount of neurons and the *kernel size* is the receptive field of a convolutional layer. A slight change is made for the implementation in our project, by using the activation function ELU, instead of RELU.

[13] Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*.

Table 4.1: AlexNet Architecture[14]

Type	Maps	Size	Kernel size	Stride	Padding	Activation
Fully Connected	-	1000	-	-	-	Softmax
Fully Connected	-	4096	-	-	-	ReLU
Fully Connected	-	4096	-	-	-	ReLU
Convolution	256	13x13	3x3	1	same	ReLU
Convolution	384	13x13	3x3	1	same	ReLU
Convolution	384	13x13	3x3	1	same	ReLU
Max Pooling	256	13x13	3x3	2	valid	-
Convolution	256	27x27	5x5	1	same	ReLU
Max Pooling	96	27x27	3x3	2	valid	-
Convolution	96	55x55	11x11	4	same	ReLU
Input	1	224x224	-	-	-	-

Predefined functions from Tensorlayer are used to implement the network. For example a convolutional layer can be created with the function *tl.layers.Conv2dLayer* and the parameters have to be set as stated in the table 4.1.

```
#Tensorlayer function to implement a Convolutional Layer
network = tl.layers.Conv2dLayer(
    network,                    #prior layer
    act = tf.nn.relu,           #activation
    shape = [3, 3, 256, 384]    #kernel size, prior map size, current map size
    strides=[1, 1, 1, 1],
    padding='SAME',
    name = 'cnn_layer')         #name for identification
```

4.4.5 Optimiser

Optimiser are responsible for the improvement of the result. If the wrong optimiser is used with a not suitable learning rate, the network can not succeed.
On the basis of a cost function an optimiser updates values to improve the output correctness. These values include the weights of the connections between the neurons and their biases. Calculating the probability error, the cost function has to be minimised to improve the accuracy.
The most used cost function is the cross entropy function. The equation for a single neuron

$$C(y,a) = -\frac{1}{m}\sum_{i=1}^{m}(y * log(a)) \tag{5}$$

outlines the functionality of a cost function. If the prediction a is near the correct output y, $(y * log(a))$ is equal to 1. If all computed outputs are nearly correct the cost function will be 1.[15]
With the cross entropy cost function a vector can be computed containing the cost for each class. The equation for one class, k, is stated below.

$$\Delta_{\theta_k} J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(p_k^{(i)} - y_k^{(i)})x^{(i)} \tag{6}$$

[14] Alex Krizhevsky, *ImageNet Classification with Deep Convolutional Neural Networks.*
[15] Nielsen, *Neural Networks and Deep Learning.*

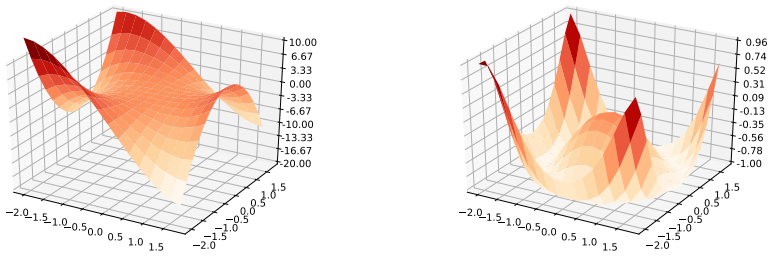
The possibility that the ith instance is in class k, is indicated by the probability p. y is the wanted output and x is the instance. With these vectors the optimiser finds a parameter matrix θ to minimise the cost.
The learning rate η helps to police the updating of the value. By multiplying it with the resulting value deterioration is diminished, this implies the value needs to be be smaller than one.
At the beginning a gradient descent optimiser was used, but led to no noticeable improvement of the accuracy. In the following sections the different optimiser tested and it's problems are outlined.

Gradient Descent Optimiser

The most popular optimiser is gradient descent,

$$\theta = \theta - \eta \cdot \Delta_\theta J(\theta) \tag{7}$$

which updates the parameters by subtracting the result of the cost function, reduced by the learning rate. It is difficult to achieve a good convergence with this optimiser. Using a too small learning rate, improvements will take too long and with a too large learning rate there's a possibility the optimum will never be reached. Also the optimiser can't react to characteristics of the data set. Applying gradient descend to an neural network, other problems can be local minima and saddle points.[16]



(a) **(b)**

Figure 4.7: Gradient descent can get stuck in local minima or saddle points

Momentum Optimisation

The enhancement of gradient descent, momentum optimisation, can pass over local optima

$$\begin{aligned} m &\leftarrow \beta \cdot m + \eta \cdot \Delta_\theta J(\theta) \\ \theta &\leftarrow \theta - m \end{aligned} \tag{8}$$

by adding the prior momentum m to the local gradient, this simulates acceleration. Although due to this acceleration, oscillating at the global optimum can happen.[17]

[16] Ruder, "An overview of gradient descent optimization algorithms".
[17] Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*.

RMSProp

The RMSProp algorithm

$$\begin{aligned} s &\leftarrow \beta \cdot s + (1-\beta) \cdot \Delta_\theta J(\theta) \otimes \Delta_\theta J(\theta) \\ \theta &\leftarrow \theta - \eta \cdot \Delta_\theta J(\theta) \oslash \sqrt{s+\epsilon} \end{aligned} \tag{9}$$

reduces the vector along the steepest dimension to correct the direction towards the global optimum. First the squared gradients get accumulated, where the symbol $\otimes$ represents the element-wise multiplication. This sum is than added to gradients of the most recent iterations, achieved by exponential decay. The resulting s is then used to reduce the scaling vector by element-wise division, represented by $\oslash$. To avoid a division through zero ϵ, usually a number like 10^{-10}, is added. With this calculation the learning rate decays over time and more intense if the dimension is steep.[18]
Before Adam Optimiser was developed, it was the most used optimiser.

Adam Optimiser

In 2015 a paper was published, introducing a newly invented optimiser called Adam.[19] Optimisation with Adam is a combination of the Momentum Optimiser and RMSProp.

$$\begin{aligned} m &\leftarrow \beta_1 \cdot m + (1-\beta_1) \cdot \Delta_\theta J(\theta) \\ s &\leftarrow \beta_2 \cdot s + (1-\beta_2) \cdot \Delta_\theta J(\theta) \otimes \Delta_\theta J(\theta) \\ m &\leftarrow \frac{m}{1-\beta_1^T} \\ s &\leftarrow \frac{s}{1-\beta_2^T} \\ \theta &\leftarrow \theta - \eta \cdot m \oslash \sqrt{s+\epsilon} \end{aligned} \tag{10}$$

The second and fifth step keeps the track to the global optimum. The first step calculates the momentum. In step 3 and 4 faster acceleration at the beginning is achieved by increasing m and s, where T is the iteration number. At the moment the Adam algorithm is the fastest optimiser and used for the project.[20]

4.4.6 Prevent overfitting

A neural network is trained on training data. It's accuracy is tested on data never seen in the period of training. The prevention of overfitting means to prevent poor adaptability and therefore bad accuracy with the test data. This achieved through different techniques, which have been implemented.

Early stopping

The easiest one to implement is early stopping. Periodical the accuracy of the network is measured and the best state of the network is saved. If the accuracy drops over time, the saved state is loaded.

[18] Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*.
[19] Kingma and Ba, "Adam: A Method for Stochastic Optimization".
[20] Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*.

Dropout

While the network is trained, every step some neurons are dropped, this means they get ignored, with a 50 percent chance. This implies that a neuron can't rely on its predecessor and its successor and therefore needs to generalise more and be less sensitive if the input changes slightly. A benefit of one to two percent accuracy is possible with this method.

4.5 Expanding the Network

After the components to be used are determined the network is expanded to perform the complete task.
To classify with the aid of all images and the weight, four small networks are concatenated. The small networks to compute the images have the structure of the test model architecture. The concatenation of the networks is done horizontal. This means the output of each neuron of the networks is transmitted to one layer. To guarantee every value is regarded the width of the layer is as wide as all four networks combined.

4.5.1 Testing the correct operation

At this point the correct operation needs to be tested. Over the course of one week the network was trained with 375 parts per hand. With the experience gained in the process two realisations were obtained.

As shown illustration 4.8 the improvement of the accuracy was hardly present. With 50 parts tested to achieve the accuracy, the improvement consisted of two parts instead of one part correctly classified.

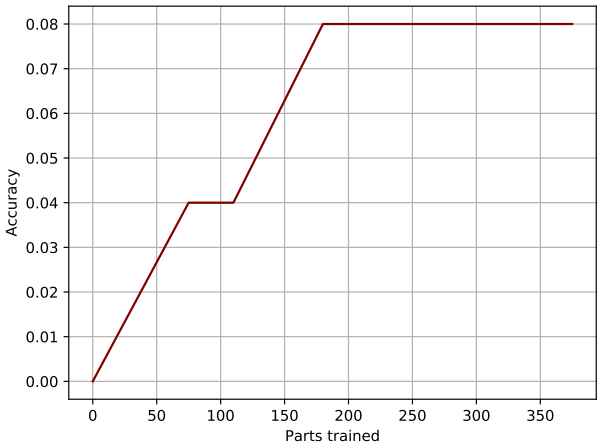


Figure 4.8: Accuracy improvement over time

Although the correctly classified parts were rare, the test run was a success. The network is able to recognise general structures of the parts, this can be concluded through the identification of parts as different ones but with similar structure. Two bricks with it's classified counterpart can be seen in figure 4.9. This proofs the ability of the enhanced network layer architecture to still identify complex patterns, irrespective of the concatenation.

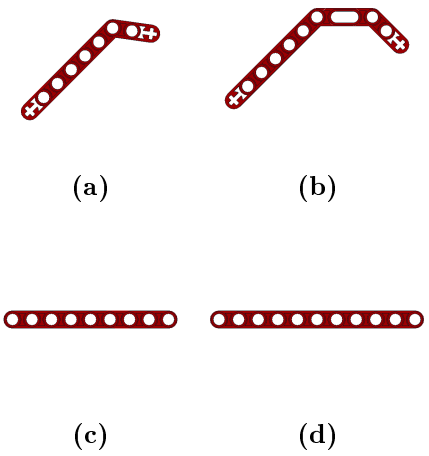


Figure 4.9: The neural network classifies part (a) as part (b) and (c) as (d). This proofs the ability to identify general structures.

To increase the correct predictions more training data is needed, nevertheless. With 200 inputs the accuracy of 8% has not improved therefore an estimated amount of 400 units of training data is set for an improvement of 4%. For an accuracy of 80% this would result to 8000 inputs. Although with higher accuracy training gets slower. Therefore an approximately training data in the range of 10000 - 15000 is needed. Training the network, to achieve a satisfying rate of correct classified parts per hand would be a time consuming task. To accelerate the learning process another possibility has to be found.

4.5.2 Wrong Assumptions

In the beginning of designing the machine learning system an assumption was made, but never verified. It was assumed that the concatenation will align the individual results of the networks.
Testing the correct operation it was thought wrong that the network will improve over time. With virtual training (discussed in 4.6.1 on page 38) the mistake was realised. The individual sections distract each other and a maximal accuracy of 15% can't be exceeded. The progress during training is illustrated in figure 4.10

4.5.3 Correction of the Mistake

To correct the made mistake a new classification process has to be worked out. The error was the concatenation, therefore it has to me avoided. The circumvention was the returning to the test model.
Each image is classified by the CNN network. If two of the three classifications are equal the part is considered the corresponding label. This procedure also implies advantages. As all images pass through the same network, three times as much training data is provided. The reason is, that the parts can be situated in any position, therefore some perspectives can be recorded from multiple cameras. With this amendment, the weight will be utilised to review the identification of the network only.
To test this approach three parts with training data in different perspectives is used. Reaching over 90% accuracy with 2000 units of training data, the test was a success. The training progress can be spotted in figure 4.11 later in this chapter.

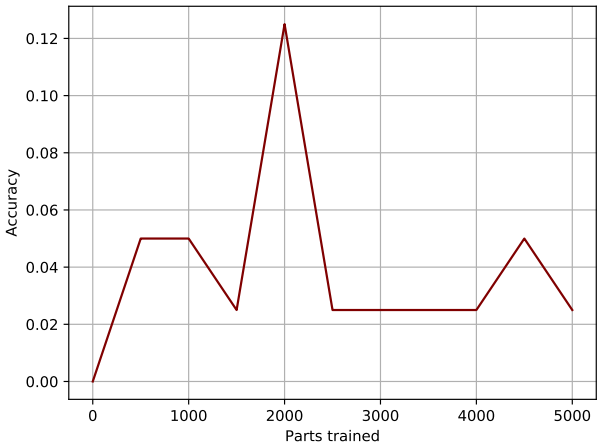


Figure 4.10: Accuracy improvement over time of the enhanced model consisting of four concatenated small networks.

4.5.4 Number of outputs

Deciding how many parts to classify simultaneously is not as simple as it seems in the first moment. The question is more complex because of the variables associated with the network. During training it became clear that the learning rate has to be adapted, depending on the number of outputs.
This is because the learning rate slows down the adaptation of the weights. If it's to high the weights cannot be adjusted accurately enough. This leads to the problem that the network skips some outputs, therefore never classifies them correctly.

Trying to solve this problem by adjusting the learning rate more precisely or dropping the learning rate while training were not successful. In figure 4.11 the achieved accuracies for different numbers of outputs are illustrated. Each testing was executed with a learning rate of 0.0001. The set up of the test runs is declared later in section 4.6.2 on page 39
The dropping in accuracy if the number of outputs is greater than 5 can clearly be seen. This leads to the conclusion to set the upper boundary to five parts. With further studies the boundary can be raised later.
In the graph it can be seen however that learning pace decreases with the number of outputs. If the learning rate has to be decreased also, the speed can drop further. It is to be presumed that the training time will increase exponential with the number of parts classified simultaneously.
With this in mind using five parts can lead to better results even if more would be possible. The number of training data and time needed to increase to a sufficient accuracy would be too high.

4.6 The learning process

The neural network can be used with three different programs. The first one, to train the network the general knowledge of the structures on the basis of virtual models. This step is not yet developed completely and can be omitted. The second program is needed to optimise the output on real data and the last one for the automatic identification of Lego

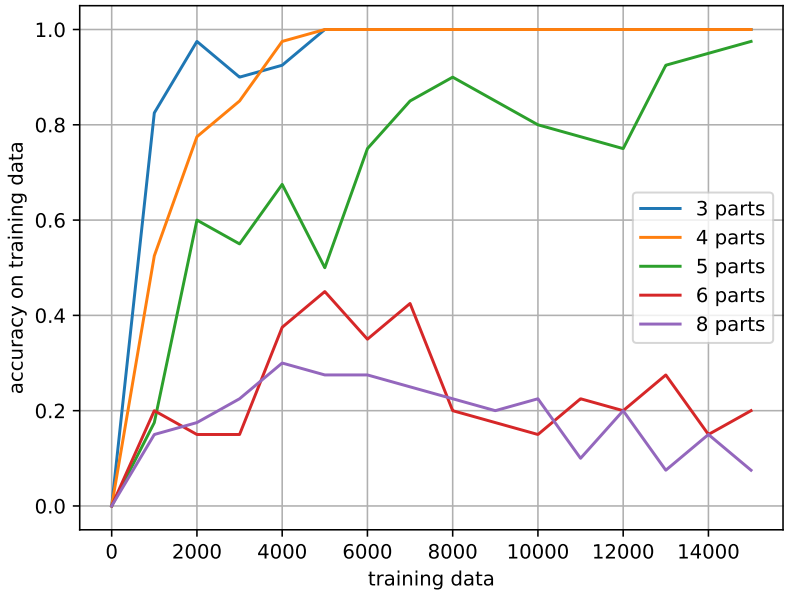


Figure 4.11: The training process while training illustrated with different numbers of parts with 360 units of virtual training data for each part. The accuracy was tested on 50 units of the training data, therefore needs to be tempered with caution

parts.

4.6.1 Virtual generation of training data

To train the network sufficiently, a large amount of training data is needed. Using 3D-Models of LEGO parts, an almost realistic record should be generated. LDView is a program to generate pictures of LEGO designs with realistic lighting and structure.[21]

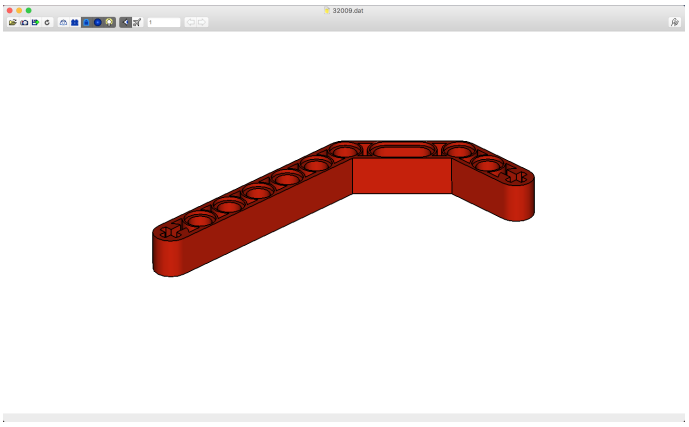


Figure 4.12: LDView is a program to display LEGO 3D-Models

It provides a command line tool to capture pictures of the model. With a script each time three images from the parts are recorded, positioned as in the measurement environment.

[21] *LDView*.

The script captures image pairs in each of the 360 degrees. In figure 4.13 a direct comparison between training data generated in the measurement environment and with LDView is illustrated.

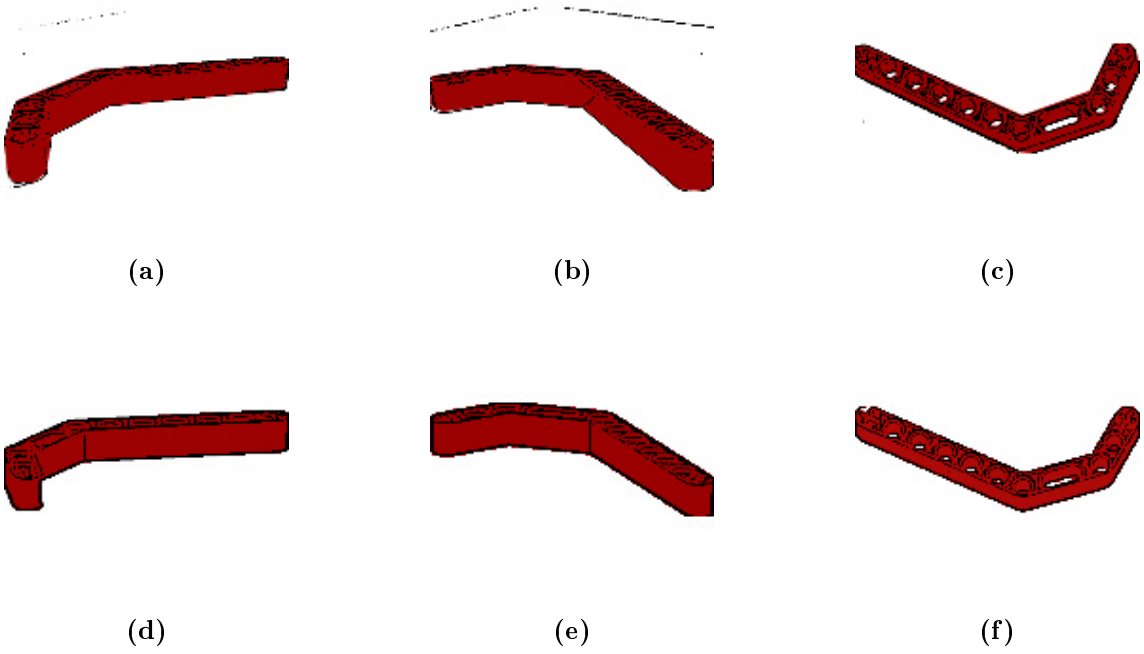


Figure 4.13: The comparison of training data generated in the measurement environment (a), (b), (c) and virtually with LDView (d), (e), (f). The parts are coloured to achieve better perceptibility.

Another script uses the above mentioned capture script to generate these images of the part associated with the submitted part number. In the last step a python program joins the images with the associated labels. The images also get preprocessed to improve the learning performance. The resulting data is stored in arrays to transfer it to the model more easily.
With this technique a sufficient amount of training data can be generated. This becomes possible by creating a 360 degree view in training data. With a model possible of detecting five different parts, training data in the size of 1800 units is generated.

4.6.2 Learning of the general structures

With enough training data generated the network is ready to be trained. In this step the weight is not taken into account.
The expanded network architecture tests were executed with this program. The accuracy was tested on training data, therefore it is higher as with data never seen before. Although because all perspectives should be covered, this was the only way.

Currently the virtual generated training data is not diverse enough to use the trained network to identify parts in the real set-up. In figure 4.14 the progress of training a network with virtual data and testing it's accuracy with real test data is illustrated. It can be seen that the accuracy tested with real data remains parallel to the accuracy tested with virtual

data. This can lead to the conclusion that by improving the generation of training data, the real training can be minimised. Nevertheless, to accomplish better results the existing network has to be further trained.

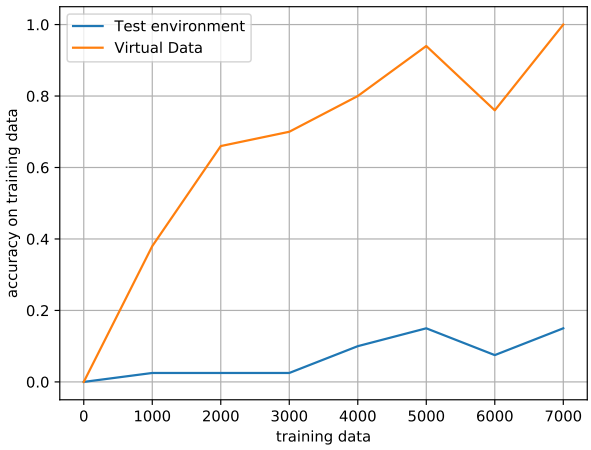


Figure 4.14: The accuracy while the network is trained on virtually generated training data (orange) and the accuracy of this network tested with the measurement environment (blue)

4.6.3 Learning in the measurement environment

As the virtual generation is not developed well enough, the network can be trained in the measurement environment only. On basis of the gained data around 13000 parts need to be scanned to reach an accuracy over 90%. During testing with the measurement environment an average time of one minute per part has emerged. Summed up to the estimated parts a total time of 216 hours would be necessary.

4.6.4 Predicting the part number

After having the network trained to a sufficient accuracy the network can classify parts with this program without updating the parameters of the network further. This means no user has to check or clarify the output. If the accuracy occurs to not be satisfying, the network can be further trained with the second step.
If parts are encountered with three different predictions the part is declared as not associated. To decrease the probability of sorting bricks wrong, the prediction is checked for plausibility by comparing the measured weight with the weight from the database. If it is outside the range of 0.5 grams the part is also declared as not associated.

Chapter 5

User Interface

Author: Simon Babovic

To improve usability and regulation of the part classifier, the idea was to create an user interface. A plan had to be worked out on how to pack the complex process of image classification into a simple, human readable form. The goals of the user interface are to visualise, what the part classifier does, and to provide an easy way of communication between the software and the user, when human input is needed. From planning phase, three core features emerged.

- A live preview for production, which visualises the process of image classification.
- An interface to control the process of reinforcement learning.
- An interface for experimental virtual learning.

The user also shuold be able to combine virtual learning and reinforcement learning for experimental purposes. Because of this, and the fact that the production software uses the data generated from virtual and reinforcement learning, there needs to exist a shared ressource. This ressource is the neural network itself. A connection had to be made for the features to use this shared ressource. So a fourth feature arose: **The administration of neural networks**.

After the features bacame clear, an implentation variant for the user interface had to be chosen. As a web interface qualifies best for visualising data, due to the strict distinction between data and design, which makes it easy to implement, this option was chosen. There is just one problem. The communication of a web application and a python process. The client side of a web interface is implemented in JavaScript. The only technology supported for real time communication is the Websocket protocol, but python is best to communicate with TCP sockets. The easiest solution would be to implement the Websocket protocol beside the part classifier to allow communication. This is considered as very heavy-weight, so an attempt was started to find another solution.

5.1 The Bridge

The idea was, to implement a server, which is capable of converting Websocket messages to TCP streams. Over this server it would be easy for two processes, of which one implements the Websocket Protocol and the other one TCP sockets, to communicate. This is because of the server creating an abstraction layer. This abstraction denotes itself through two endpoints. One on which the TCP client connects and the other one, where the Websocket

client connects. JSON got chosen as message format, as it is widely used in the web sector and almost every programming language has a way of supporting it.

5.1.1 Libraries used to establish communication

Node.js

Node.js is a runtime-environment, which allows to run javaScript on the server side of the application. It's architecture is event-driven and it provides a lot of different modules for every purpose, like Socket.IO or the Net-module, which is used to establish tcp connections with Node.js. It's latest stable version is 9.5.0 from Jannuary 31st 2018.

WebSockets

WebSocket is a protocol to provide a full-duplex communication over a TCP connection. It works on HTTP ports and unlike normal TCP streams, which process raw bytes, the WebSocket protocol provides a concept of messages. As JavaScript is limited to Websockets and is not able to open TCP streams, a server had to be implemented to bridge WebSocket messages to a set of bytes (text) for the TCP stream. For the client side of the user interface, Socket.IO was used as WebSocket library.

Socket.IO

Socket.IO real-time web application library. It consists of two parts. The client part which runs in the browser and the server part which is a Node.js library. It uses the Websocket protocol. Socket.IO provides an API to easily build websocket appllications. It's latest stable release is 2.0.3 from June 13th 2017.

5.1.2 Message Types

To ensure a well working communication, a few message types got defined. They are grouped in two sets.

- **Emitting Messages** are messages which have their origin at the user interface. They are sent to the server, which then forwards the message to the part classifier.
- **Receiving Messages** are the opposite of emitting messages. They come from the part classifier and are sended to the user interface. They are most of the time responses to emitting messages.

Every message can hold JSON data if it needs to. An example message looks like this:

```
1    {
2        "message_type": "bridge",
3        "type": "stop"
4        "data": { "force": "false" }
5    }
```

This message would be an *Emitting Message*. The value of the attribute *message_type* indicates to the server, that the message should be forwarded to the part classifier. This attribute is the only one which gets read by the server. The rest of the attributes indicate to the part classifier to gently shut down and to save before exit. For emitting messages, this JSON object gets stringified and sended to the tcp socket. For receiving messages, this JSON string gets parsed and verified. Once validated, a new websocket message with given name and data will be sent to the user interface.

Following message types got defined for the communication between the two processes. To distinguish the message type in JSON, the name of the message is put in the type attribute as shown above.

Emitting Messages		
Name	**Description**	**Response**
startup	starts the part classifier and loads a neuronal network	startup success or startup fail
stop	saves the current neuronal net and stops the classifier	stop success or stop fail
save	saves the current neuronal net	save success or save fail
learning response	response to learning request	-

Receiving Messages		
Name	**Description**	**Response**
learning request	made a prediction and need human response	learning response
startup success	confirms startup complete	-
startup fail	indicates error on startup	-
stop success	confirms shutdown complete	-
save success	indicates save complete	-
save fail	indicates error on save	-

While implementing the bridge, it soon became clear, that the bridge is not everything needed for a well working user interface. On both sides of the communication is a need for information from the database, so the idea of a database API arose. For the TCP side, a new message type named database got defined, which allows the part classifier to make database queries over the TCP socket.

5.2 Database API

The idea was to create an API over http, which abstracts the access to the database. As http provides different methods, it was a matter of interest to implement the functions with their respective http methods.

- **GET** for getting a ressource
- **POST** for creating a ressource
- **DELETE** for deleting a ressource

5.2.1 Used libraries to implement http methods

Node Express

Express is a module of NodeJS (See 5.1.1). It provides functions for implementing a http server. With this it is possible to create custom routes. For example can the route */networks* point to the file */html/nets.html*. Also, express allows to define route parameters, so a route like */networks/1* can for example point to */html/networks?id=1*. This is not limited to files, meaning the response does not have to be HTML. It also is possible to send plain text or JSON. This makes the express module perfect for implementing a database API.

Node mysql

NodeJS (see 5.1.1) provides a module named mysql, which is capable of establishing a connection to a database and running queries on it. This library was used, because it is very easy to run database queries due to the event driven process of JavaScript.

5.2.2 Mode of operation

The goual was to implement the API invocations as intuitive as possible, so for example *GET /network/1* to get information about the network with the id 1, or *DELETE /network/1* to delete the network with the id 1. The only problem is, that the express server also defines the routes for the frontend (see 5.4), so the database api routes had to be encapsulated from the frontend routes. This is done by prefixing all database API routes. The example invocation from above now looks like this: *GET /db_ api/network/1*.

5.3 Architecture

When combining the bridge and the database application programming interface, it results in the following communication architecture (see 5.1).

5.3.1 States

The server can have different states which determine if the server is waiting for a response or not. More than one state can be active at the same time. These states are as follows (also see 5.2):

- **Running**
  The part classifier is running.
- **Save requested**
  The user requested to save the neuronal net and the server is waiting for confirmation.
- **Startup requested**
  The user requested to start the part classifier and the server is waiting for confirmation.
- **Stop requested**
  The user requested a shutdown of the part classifier and the server is waiting for confirmation.
- **Waiting for learning response**
  The part classifier has made a prediction and waits for a human interaction.

These states can be visualised as deterministic finite state machine (see 5.2).

5.4 Frontend

The frontend is written in HTML5, CSS3 and javaScript. As the interface is installed locally and will never be listed on a search engine, it does not need to load super fast. Therefore, without compunction, third-party libraries, such as Bootstrap and jQuery were used to most of all improve code readability. To save time, the dashboard template from Bootstrap was copied and pared down to the bone.

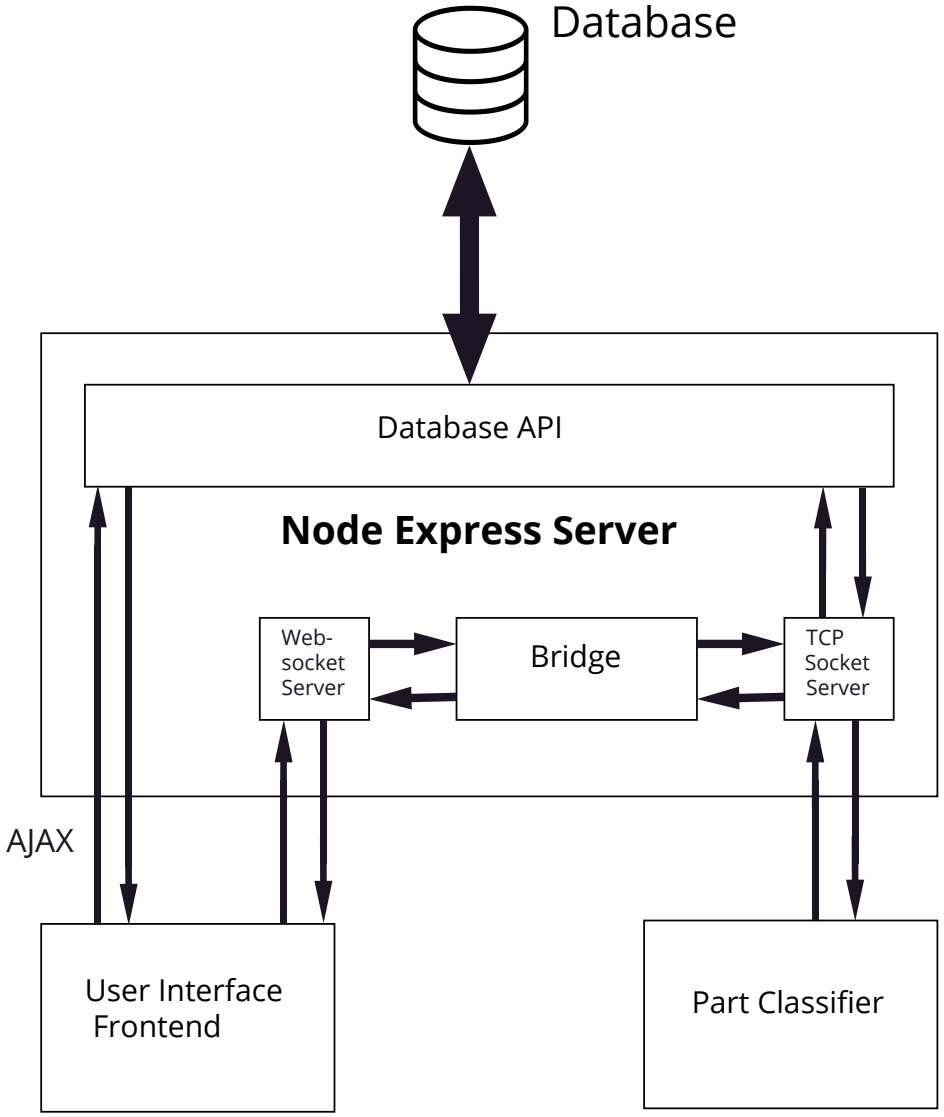


Figure 5.1: This graphic depicts the architecture of the server, its communication interfaces and the two processes using them. The architecture consists of two parts: the database API, which is invokable through http reqests for the web app and through TCP messages for the part classifier, and the bridge, which forwards Websocket messages as strings through TCP sockets and vice versa.

5.4.1 Used Libraries to create a good user experience

Bootstrap

Bootstrap is a framework for web-development which aims to make the process of developing a frontend easier and faster. It provides design templates for almost every frontend component, such as buttons, forms, tables, etc. It also provides JavaScript plugins for more functionality. It was developed by Twitter as open source project on GitHub. It's first release was in August 2011 and in June 2014 it was the number one project on GitHub. It's current stable release is 4.0.0 from Jannuary 18th 2018. In this project the advantages of bootstrap are used to build up a frontend fast and easy.

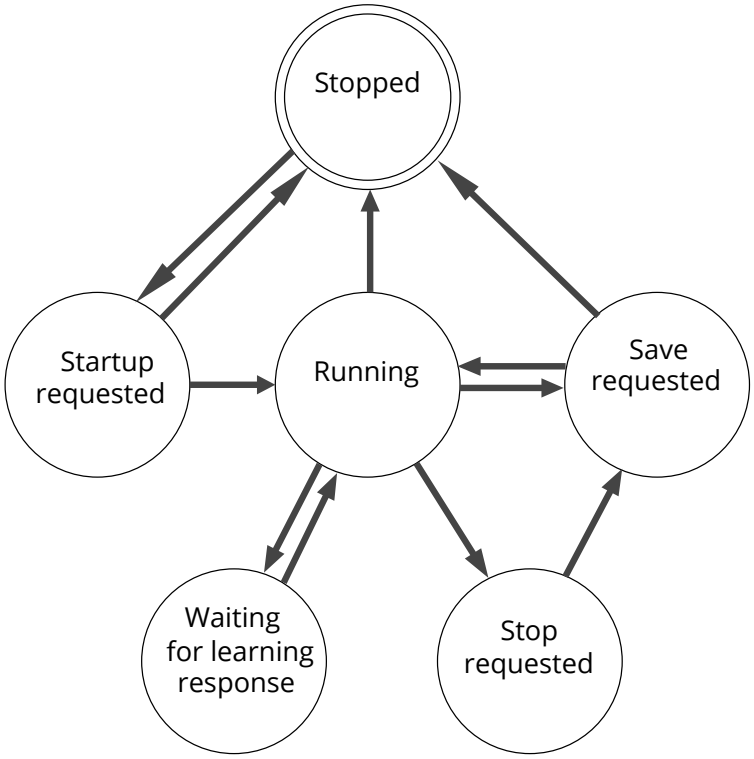


Figure 5.2: This graphic shows the states the express server can take and how they change. The stopped state is both start and final state. When the start is requested and an error occurs, the machine goes back to stopped, otherwise it changes to running. The *waiting for learning response* state is taken when human interaction is needed. When a stop is requested, there will always be a save request before shutting down, except the shutdown is forced. In this case *SIGKILL* is sended to the process and the machine immediatly changes to stopped.

jQuery

jQuery is a pure JavaScript library to improve usability and code readability of JavaScript. It provides functions which make it very easy to select and manipulate HTML elements in very few lines of code. For element selection, it uses the same syntax as CSS, which especialy makes the code much simpler to read. It was developed by John Resig and it's first release was in August 2006. It's latest stable release is 3.3.1 from Jannuary 20th 2018.

5.4.2 User Interface Client

The user interface client provides it's core functionalities, such as administrating, training and monitoring of the part classifier, in four main pages.

1. Live Preview
2. Reinforcement Learning
3. Virtual Learning
4. Networks

These pages are integrated in one main frame for both design and functionality purposes. The design of the main frame is on all pages the same, so it only needs to be loaded once.

The connection with the server is implemented in this main frame, thereby messages do not get lost if the page needed is not loaded at the time the message arrives.

5.4.3 Networks

The networks page is dealt with first, because this is the page where the administration of the networks happens (see 5.3) and every other page uses such a network. The main function of this page is to create networks. A network is created by defining a name for it and by adding Lego parts to it. In addition it is possible to view and delete networks created in the past by clicking on the name of it in the top table. Also there is a feature to virtualise a network. After a network got virtualised, it can be used with virtual learning. When clicking the button, the server creates all files necessary for the virtual learinging process (see 4.6.1).

One major problem came up before implementing the page, which had to be solved first: *How can the about 10000 parts in the database be visualised to the user and what is the easiest way to effectively search through a dataset this large?*

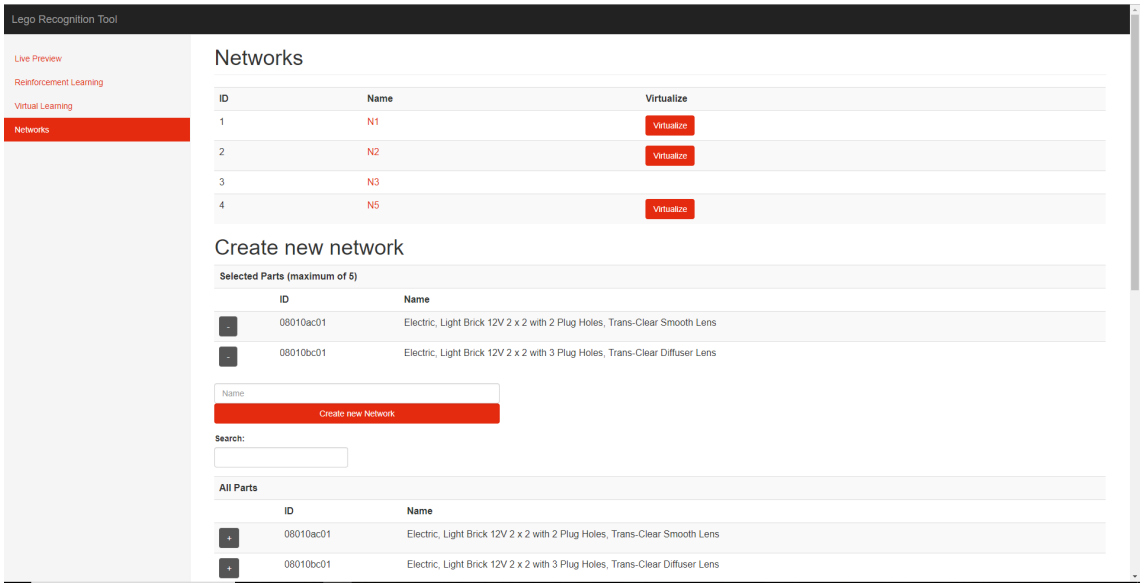


Figure 5.3: Graphical network administration page

The search function

The idea was to show 20 parts from the database in a table, while all other parts are loaded in the background. It then is possible to page through sets of 20 parts. Additionaly the idea of a live search function, which is capable of searching through all parts by name and partnumber, arose. A search algorithm would be used everytime the user types a character into the search bar. This resulted in unresponsive behaviour, due to the huge amount of unsorted data which has to be searched through. The best solution for this problem was, to wait until the user is ready to look at the search results, resp. has finished typing or stops typing to look through the interim results. An interval of 600 millisecounds was chosen, as this is the time the backspace key needs to initiate channeling.

5.4.4 Live Preview

The live preview is the interface for the production process, in which parts get classified. The only options for controlling it are starting and stopping. Before starting the process, a network, created in the networks administration page and trained in one of the learning pages, has to be selected. After start, the classifier sends result messages periodically to the interface. The main goal of this page was to provide an easy way for the user to monitor the part classification process. This is achieved by a big container on the top of the page showing all results the classifier got from classifying one part (see 5.4). The page also provides a table of a short history of identified parts.

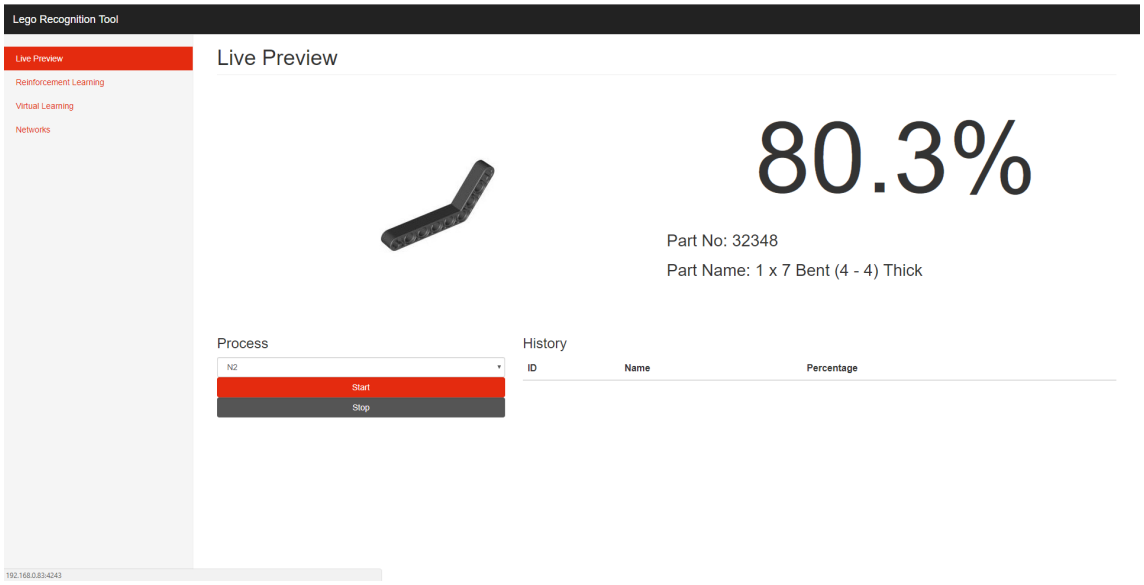


Figure 5.4: Production live preview page

The main problem while implementing this page was, that the mechanical structure has not been developed far enough to automatise the process of delivering a part to the machine. As the live preview page should require as little human input as possible, a program was written to simulate periodical identification of parts by sending dummy results in an interval of five secounds.

The only feature not implemented is the image view of the classified part. As this page is experimental and only usable after the mechanical work has been done, the image of the lego part in figure 5.4 only is a dummy picture. Later on, it should show the photos made from the cameras to provide a better overview of which part has been identified.

5.4.5 Reinforcement Learning

The reinforcement learning page is the main page of the user interface. Its goal is to provide an easy way for the user to train a network by hand. This process is executed in six steps.

1. Start the process after a trained network has been selected and wait for the program to start
2. Place a Lego part into the machine and tell the process that the machine is ready by clicking the Ready-button (see 5.5)
3. Wait for the classifier until it makes a prediction

4. Tell the classifier the correct part number of the part in the machine
5. Repeat from step 2
6. Save the network and stop the process by clicking the Stop-button

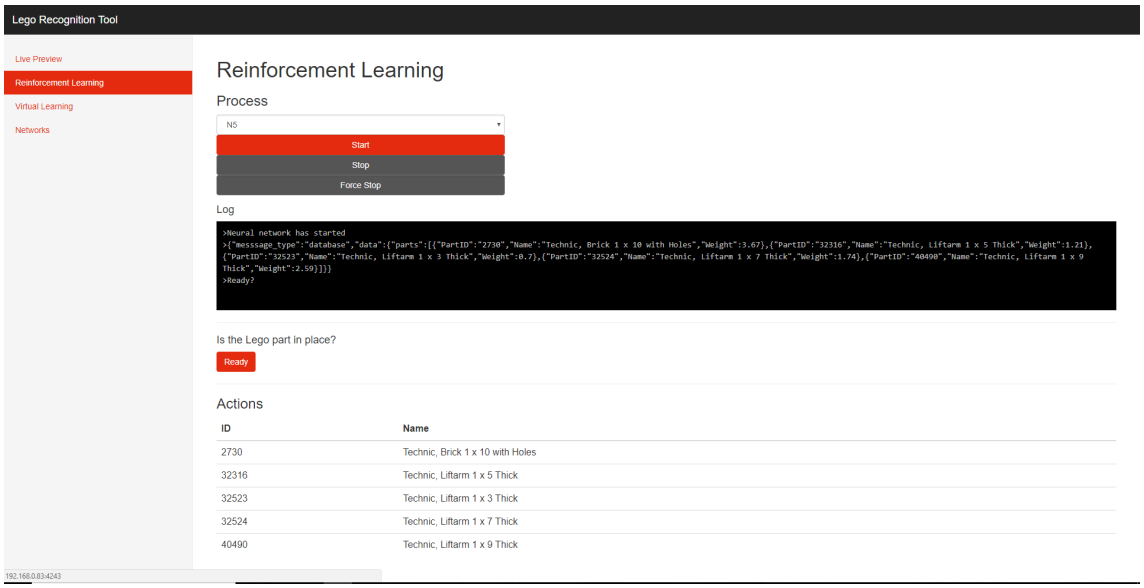


Figure 5.5: Reinforcement learning interface

The main problem again was how to visualise the whole process to the user. The first idea was to tell the classifier the correct answer by entering the number of the part sitting in the machine. But this idea got drafted because part numbers are very long and hard to remember. The other idea was to instead showing the possible part numbers in a table, hieghligthing the guess of the part classifier and allow the user with a click on the part number to pick the correct answer.

The Log container

The log container serves as information display. It is connected with the standard output stream of the classifier and designed to look like a shell. In this container, information, like when the process has started or when it is waiting for human input, gets displayed. As the complete project output is a prototype, the log container also is connected to the standard error stream of the classifier, so debugging for the developers gets easier.

5.4.6 Virtual Learning

The virtual learning interface was created for experimental purposes, as the virtual learning process is experimental too (see 4.6). It provides a larger log container than the reinforcement learning interface (for the purpose of the log container see 5.4.5).

Before starting the virtual learning process, again a network has to be selected. These networks are limited to virtualised networks (see 5.4.3). This means it is only possible to train the network virtually, when virtual training data has been created.

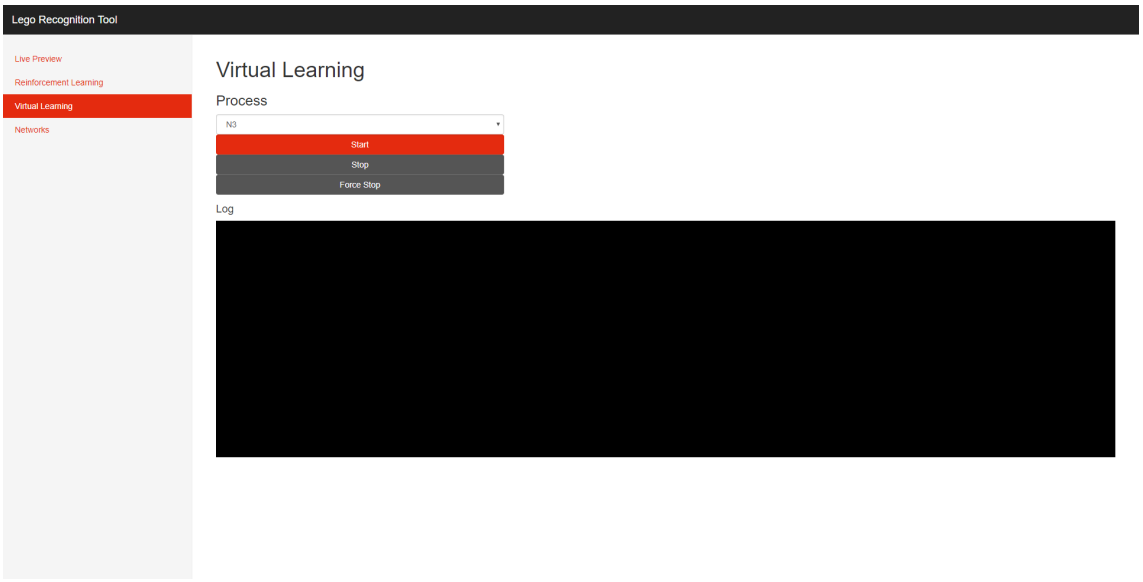


Figure 5.6: Virtual learning interface

5.5 Installation

As the project has many dependencies to existing software and libraries, the idea of creating an install script arose. For this, three files were created:

- **package.json**
  This file stores the information about required NodeJS modules and their version (For more information about NodeJS, see 5.1.1). It allows the user to automatically install all requirements by running the command *npm install*.
- **requirements.txt**
  This file stores the information about required python packages and libraries. With the command *pip install -r requirements.txt* all packages required for the image classification process get installed automatically.
- **install.sh**
  The goal of this file is to automate the installation process with no need of human input. At first it installs the mysql server and imports the parts database along with the tables needed by the user interface. It combines the two commands mentioned above, right after NodeJS, Python, NPM and PIP got installed. In the last step LDView gets installed and all files necessary for the experimental virtual learning method get created.

5.5.1 Preconditions

The install script presupposes to run on a fresh Debian 8 machine because of two reasons. Firstly, it uses the apt-package-manager and secondly, to successfully install the mysql server without human input, debconf commands are used by the script, which are only available on Debian operating systems.

Chapter 6

Conclusion

Author: Simon Babovic

The Lego Recognition Tool project provides a firm basis for identifying Lego bricks, while using only low budged gadgets in an image classification process. The project was built up from scratch completely. The task was to develop a machine capable of identifying Lego pieces, while having a budget for materials and gadgets of 1000€. This project is intended to be the first step to an autonomous Lego-sorting machine. At first, there had to be thought of ways to identify one piece, then solutions to many problems had to be worked out.

6.1 Recap

In the beginning, ways to fulfil the task had to be found. This was done by doing research on what already is available in the Lego sector. One of the outcomes of this research was the Bricklink database, which stores information of about 40000 Lego parts. It also includes information about the weight of each piece in the database, which led to the idea of using a scale as an easy way of distinguishing between groups of parts.

An analysis of the information in the database resulted in an outcome, which states, that by measuring the weight of a part, 90% of all parts can be excluded in a prediction in the worst case scenario. In the best case scenario, a part can be identified by its weight only. As a precision scale was bought and tested, the first problems arose. It turned out, that the weight information in the database is not precise, instead, the values are rounded up. Also, as the database is made from a community, the deviation of weights is variable, meaning it is exclusive for single parts. Additionally the abrasion of secondhand Lego pieces have not been considered. So it is possible that an older part has less weight than a newer one. However, the approach with the scale was not a complete waste of time, as it still plays a role in the project later on.

Because of the problems with the weights of Lego parts, the main focus of the project got shifted to identification with cameras. This method consists of two important parts: The software and the mechanical structure, which got, due to complexity, partly replaced with an user interface. To accelerate the process of development, it got parallelised. That means, that one member of the team worked on the software, and the other one worked on the mechanical setup, and later on the user interface.

6.1.1 Image classification software

The first idea was to create a neural network, which is capable of learning differences between shapes of all existing lego pieces. This turned out very complex and not achievable, due to lack of time and budget, as such a complex neural network requires huge computing power to accomplish training. Furthermore, there is absolutely no training data available, so it would have had to be created from scratch. The huge size of a dataset required to train such a neural network would widely go beyond the scope of this project, so other solutions had to be worked out.

One of these solutions was, to create a small straightforward neural network, which is capable of distinguishing between a maximum of five parts. This size turned out to work with available ressources, although five out of 40000 parts are a huge limitation. As it became clear, that even the amount of training data required for a small neural network like this is still huge, the focus shifted to both proving the neural network is adaptive, and reducing the amount of required training data.

The learning ability of the network got proven by a method called virtual learning. This method was developed to automate the process of creating training data by making pictures virtually from 3D-Objects and was originally meant to replace the reinforcement learning method, where human input is needed. To be able to detect errors in predictions, the scale is now used. It checks if the predicted part is in a range of its weight in the database. The amount of required training data got reduced by preprocessing the images of the Lego bricks. This means, that all irellevant information like background and color gets irradicated from the picture. This also implicated some hardware requirements.

6.1.2 Mechanical setup

The goal of the mechanical setup was primarily to create a setting, where image preprocessing is more easily done. This was accomplished by creating a camera box, which is homogeneous in color on the inside, so the background can be easily distinguished from the foreground. Three cameras were chosen to take pictures from the Lego part in the box from different angles, to create a good all-round view on the piece.

Also one goal was to implement a mechanical system, which is capable of bringing single Lego pieces from an unsorted box to the camera box, and from the camera box to a sorted box. While constructing the camera box, the complexity of constructing such a mechanism became clearer, so this goal got considered as unachievable for IT students, especially in the time of a school year. Furthermore, it makes no sense for an IT diploma thesis to focus on mechanical components, so the goal of creating a transporting system has been replaced with another one. As the whole project is experimental and as the part classification process is hardly imaginable, the idea of creating an user interface arose.

6.1.3 User interface

The user interface was created to visualise the process of part classification to the user and to provide administration options of a learning process. As the idea was to implement the user interface as a web app, the biggest problem was the communication between it, the part classification process and the database. Because JavaScript is limited to Websockets, and implementing the Websocket protocol in the part classifier is considered very heavy-weight, a NodeJS server was implemented to bridge Websocket messages from the user

interface to TCP streams for the part classifier and vice versa. It also provides a database API which is reachable through TCP sockets, so the part classifier has not to deal with http requests.

The frontend is designed to be self-explaining and easy to understand, while at the same time to be suitable for further development of the part classifier by providing debugging options.

6.1.4 Summary

In summary, it can be stated that both the huge variety in Lego parts and the complexity of the mechanical setup got underestimated. However, it was still possible to get good results with little budget, which make the project, as it is, a firm basis for further development.

6.2 Outlook

As the project could not be completed due to lack of time and budget, there is still much to do to bring the project to success. There is need for further development in three categories.

- **Mechanical structure**
  The sector, where the most development work has still to be done, is the mechanical one. Ways to bring a Lego part to classification and away from it have to be worked out.
- **Software improvement**
  In future development, the software of the part classifier has to be adapted to bring the machine to production.
- **System**
  The key of this project is to keep the budget required for such a machine low, so solutions have to be brought up to ensure this goal will be achieved too.

6.2.1 Mechanical concepts

Here some concepts on how a mechanical structure could look like get introduced.

Bringing the Lego part to classification

To bring a piece of Lego to the classification setting, a good way would be to lead them over conveyors. But as the parts hugely differ in size, this is not the only thing to do. A good solution would be, to pre-sort Lego bricks by size. This could be achieved with a vibration sieve. It could lead very tiny and very big parts to different classification settings, which are fitted best for their size. Concepts on how such a setting could look like are not available yet. Furthermore would it make sense to use multiple conveyor belts, which run beside one another, to split up groups of Lego pieces, so they can get to the classification setting as single piece.

Bringing the Lego part to the box

One concept to bring the a piece of Lego from classification to the correct box, is to let gravity handle the transportation. For this, the classification process would take place at the very top of the machine. A construction below would handle the navigation to the correct box. The concept is built on hatches, which open the path to the box, after a Lego piece has been identified. For more information, see 6.1.

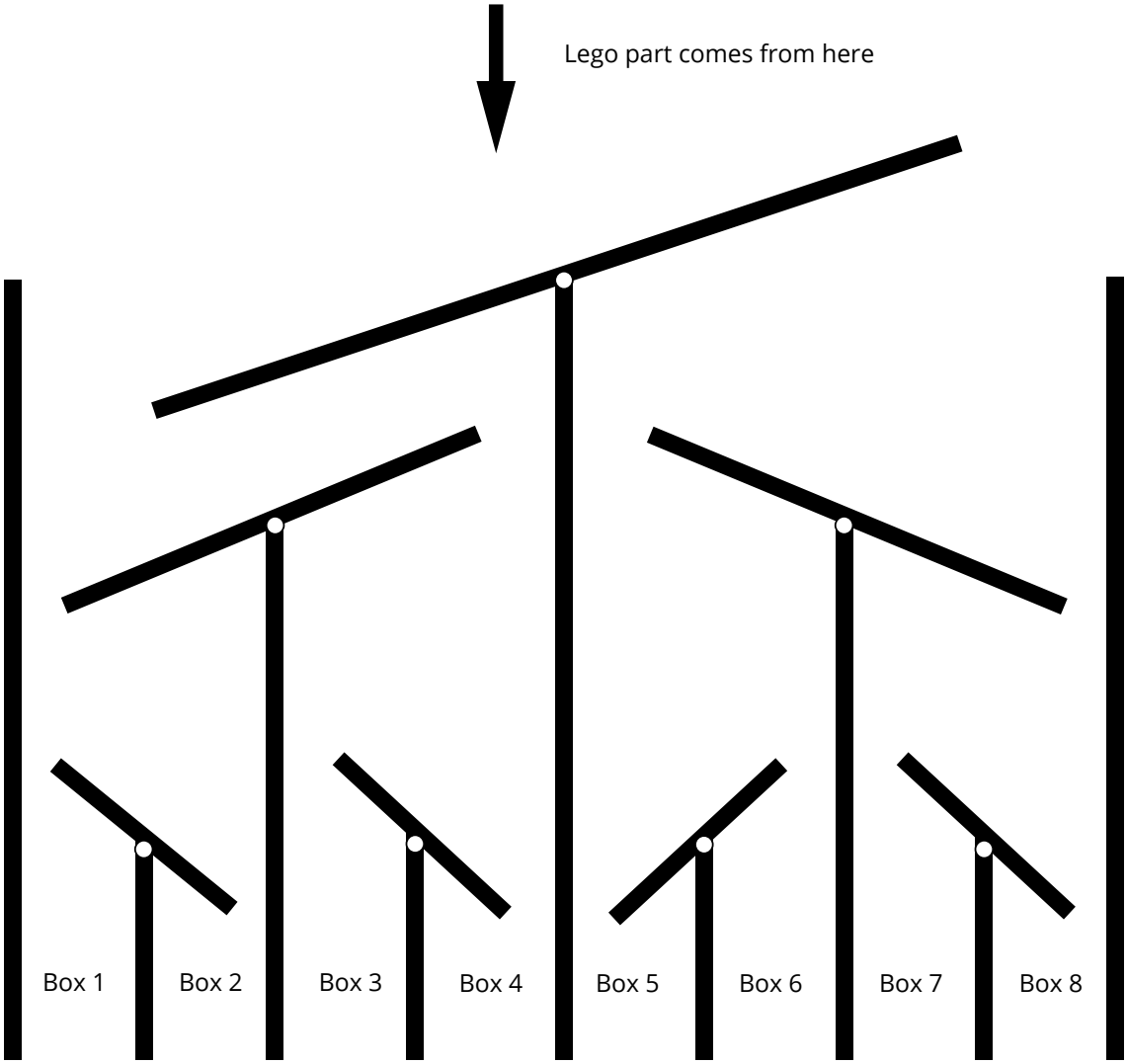


Figure 6.1: This figure is a draft of a transporting system from the image classification setting, to the boxes. The idea is, that after classification, hatches open the way to the correct box. Then the Lego part gets released and falls through the hatches into the box. When bringing this concept from two dimensions to three dimension it gets even more efficient.

6.2.2 Software improvement

To improve usability of the classification process, the virtual learning method is key, as this method can bring the learning and the classification process to production. There will have to be found solutions to the problems this method still has. One of this problems is, that the photos generated from the software are too clean, so pictures of an actual part are too different from the generated ones. This could be solved, either by using better cameras and a cleaner setting, or by adding deliberate noise to the generated pictures.
Another thing to improve would be the neural network itself. As the virtual learning method could eradicate the effort of creating training data for the neural network, a much bigger one would be able to be created. This indeed would raise the required computing power, but for this problem a solution had been worked out too.

6.2.3 System improvement

The problem is, that when using a bigger neural network, more computing power is needed to train it. More computing power can get expensive, and if using more than one machine, this would miss the goal of the project to run on low budget. One suggestion is, to create a central learning unit, which handles the training of neural networks. This unit would be the most expensive in the complete project, but the classification units would not need to have such a computing power. They might also run on a RaspberryPi, which comes at a cost of about 40€. So instead of using the same system to train and to classify parts, the training part of the process should be transfered to another location (See 6.2).

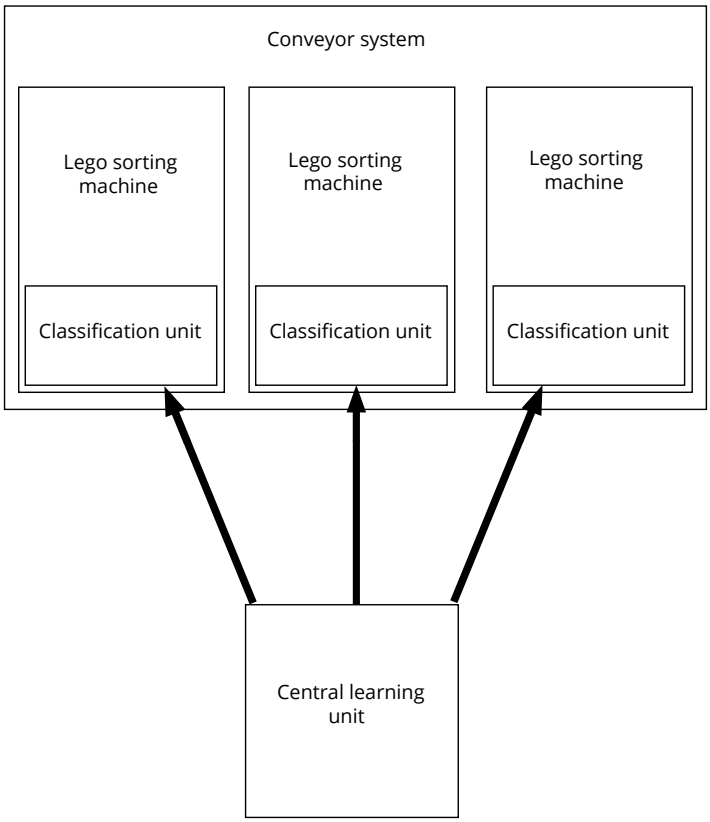


Figure 6.2: When using more than one machine, which all can distinguish between different Lego parts, it would make sense to transfer the learning process away from the classifaction process. This is because learning requires more computing power, than classifying does. So while the machines work together to identify a Lego part through image classification and, for example a conveyor system, the trained nets are stored in the central learning unit. The classification units can load the neural networks to use them in production.

Bibliography

Alex Krizhevsky Ilya Sutskever, Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. Tech. rep. 2012, p. 1097.

Angle of View. https://en.wikipedia.org/wiki/Angle_of_view. Oct. 2017.

Brickfetish. http://brickfetish.com/catalogs/dk/dk_1955.html. Feb. 2018.

Bricklink. https://www.bricklink.com/. Mar. 2018.

Bricklink Naming Scheme. https://www.bricklink.com/help.asp?helpID=168. Sept. 2017.

Burger, Wilhelm and Mark J. Burge. *Principles of Digital Image Processing: Advanced Methods*. Springer, 2013. ISBN: 978-1848829183.

Devi, H. K. Anasuya. "Thresholding: A Pixel-Level Image Processing Methodology Preprocessing Technique for an OCR System for the Brahmi Script". In: *Ancient Asia* (2006). DOI: http://doi.org/10.5334/aa.06113, 161–165.

Djork-Arne Clevert Thomas Unterthiner, Sepp Hochreiter. *Fast and accurate Deep Network Learning by Exponential Linear Units (ELUs)*. Tech. rep. Johannes Kepler University, Linz, Austria, Feb. 2016.

Fukushima, Kunihiko. "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: (1980).

Geron, Aurelien. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly, 2017. Chap. The Machine Learning Landscape. ISBN: 978-1491962299.

— *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly, 2017. Chap. Convolutional Neural Networks. ISBN: 978-1491962299.

— *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly, 2017. Chap. Training Deep Neural Nets. ISBN: 978-1491962299.

GTX1080 Ti Nvidia card. https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti/. Mar. 2018.

Hillebrand, Kurt. *Datenbanken und Informationssysteme*.

Hinton, G. E. and R. R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." In: *Science* 313 (July 2006).

ILSVRC ImageNet Challenge. http://image-net.org. Jan. 2018.

Intel Corporation. https://www.intel.com/content/www/us/en/homepage.html. Sept. 2017.

Jacques Mattheij. https://jacquesmattheij.com/. Mar. 2018.

Kingma, Diederik P. and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

Kuehni, Rolf G. *Color Space and Its Divisions: Color Order from Antiquity to the Present*. Wiley-Interscience, 2003. ISBN: 978-0471326700.

LDView. http://ldview.sourceforge.net/. Mar. 2018.

Lego. https://www.lego.com/en-us/aboutus/lego-group/the_lego_history. Mar. 2018.

Lego logo. http://lego.wikia.com/wiki/LEGO_logo?file=LEGO_logo.jpg. Mar. 2018.

Logitech C922. https://www.logitech.com/de-at/product/c922-pro-stream-webcam. Oct. 2017.

M. Hassaballah Aly Amin Abdelmgeid, Hammam A. Alshazly. "Description and Matching". In: *Image Feature Detectors and Descriptors*. Springer, 2016.

Mattheij, Jacques. "How I Built an AI to Sort 2 Tons of Lego Pieces". In: *IEEE Spectrum* (June 2017). DOI: https://spectrum.ieee.org/geek-life/hands-on/how-i-built-an-ai-to-sort-2-tons-of-lego-pieces.

Nathan Sawaya. http://www.nathansawaya.com/. Mar. 2018.

Neural Networks. https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html. Jan. 2018.

Nielsen, Michael A. *Neural Networks and Deep Learning*. Determination Press, 2015.

Ruder, Sebastian. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

Sort Your Legos Like an Engineer. https://makezine.com/2015/07/20/sort-legos-like-engineer/. Mar. 2018.

Tensorflow. https://www.tensorflow.org/. Jan. 2018.

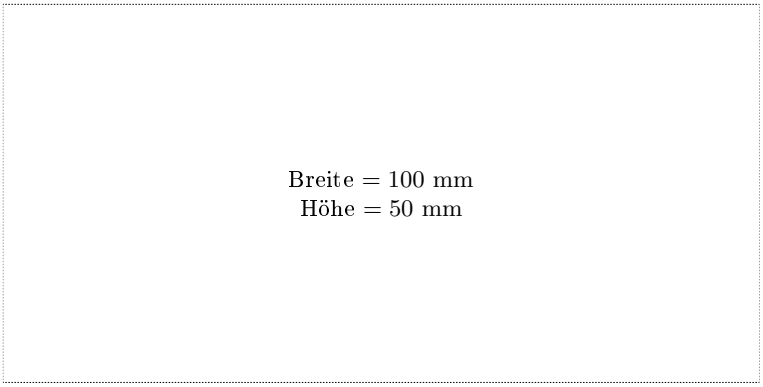
TensorLayer. https://tensorlayer.readthedocs.io. Jan. 2018.

Y. LeCun L. Bottou, Y. Bengio and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* (Nov. 1998). URL: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf.

z

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —