Design of a flexible pick-to-light system for assembly operations

Bram Dereeper

Supervisors: Prof. Johannes Cottyn, Kevin Ketelers

Master's dissertation submitted in order to obtain the academic degree of Master of Science in de industriële wetenschappen: elektrotechniek

Department of Industrial Systems Engineering and Product Design
Chair: Prof. dr. El-Houssaine Aghezzaf
Faculty of Engineering and Architecture
Academic year 2017-2018
DESIGN OF A FLEXIBLE PICK-TO-LIGHT SYSTEM FOR ASSEMBLY OPERATIONS

MASTER’S THESIS

Bram Dereeper

Master of Science in Electrical Engineering Technology (Automation)

Campus Kortrijk

Academic year 2017–2018
Preface

While being comfortable with the concept of a pick-to-light system, technologies like object recognition and cloud computing were rather out of my comfort zone as a to-be automation engineer. Although choosing a master’s thesis project that is closer to my field of study probably would have given me less troubled days during the past year, I can look back on a positive experience that widened my field of knowledge. Applying the theoretical insights I had about artificial intelligence aroused my interest for this trending topic even more.

This master’s thesis is of interest to anyone who wants to learn about or implement cloud computing, object recognition or a smart pick-to-light system.

I would like to thank professor Johannes Cottyn (Ghent University, Belgium) to make me go the extra mile to accomplish the desired result. My thanks also go out to professor Michael Packianather (Cardiff University, UK) for the interesting courses about artificial intelligence and following up the project while being abroad.
Abstract

Title: Design of a flexible pick-to-light system for assembly operations
Student: Bram Dereeper
University: Ghent University, campus Kortrijk
Programme: Master of Science in Electrical Engineering Technology (Automation)
Academic year: 2017 – 2018
Keywords: Flexible pick-to-light system, object recognition, cloud computing, convolutional neural network

Short description:

A lot of solutions to aid in the picking process exist. One of these solutions is the pick-to-light system. In this system, a light guides the picker visually towards the product to be picked. Usually this solution is effective, but inflexible. The modules must be connected to a master module and must be configured manually. This means that when a product is added or the location of a product is changed, the system needs to be reconfigured. In the process of refilling the component bins or reconfiguring the bins, mistakes can be made. A smarter solution could make the process easier to use while making less mistakes.

The heart of the smart pick-to-light system is the cloud. Not only all pick-to-light modules will connect to it, but also devices like computers and phones that control the system. When a new picking order is added to the queue in the cloud, the system figures out which components are placed in which bin using sensors with which object recognition is performed. This way, between two picking jobs, products can be freely moved around, new modules can be added or broken modules can be replaced without any reprogramming of the system whatsoever.

In this thesis, background research is done about cloud computing and object detection. The results of the research are used to create a proof of concept using a Meccano case. For this case, a flexible pick-to-light system is needed on an assembly line where two different models of Meccano cars are constructed. The smart pick-to-light system makes sure that switching between constructing these different models goes as smooth as possible.
I. INTRODUCTION

A. Warehouse picking

Warehouse picking is often referred to as the most labour-intensive, expensive and time-consuming operation in manual warehouses [1]. To make this manual picking process more efficient, a lot of solutions have been created. One of these is the pick-to-light system where the user is guided visually towards the correct product. Every module is accompanied by a light and a button. When a certain light is burning, this particular product must be picked. The picking of a product can be acknowledged by pressing the correct button.

Although this guiding is very visual and efficient in use, the system usually lacks flexibility. In a traditional system, all modules are physically connected to the master module. Every module has a fixed address that the master module uses to communicate with it. The user has to suppose that all products are linked to the correct pick-to-light module, although this is never checked by the system.

This is completely different from a smart pick-to-light system, as shown in figure 1. The master communicates with the modules from an external cloud service. If the modules are connected to the internet, they can reach the master in the cloud from any network that has an internet uplink without any reconfiguration. Also, other devices that manage the system and request information from it connect on the same cloud. Smart sensors are implemented to recognise which product is positioned in which module.

Figure 1: Hardware of a smart pick-to-light system

The traditional system has two major disadvantages compared to the smart system. On the one hand, configuration is always needed when a pick-to-light module is replaced or added. Both the address of the new module and the link to the correct product that the module contains must be manually added to the master module’s database. On the other hand, products cannot easily be interchanged between two modules since the master addresses its slaves by address.
II. Goals

The goal of the master’s thesis is to design a user-friendly and smart pick-to-light system. After theoretically researching all required aspects, practical tests are done to eventually create a proof-of-concept. The object recognition is tested using a set of 26 Meccano components.

A. Theoretical research

Firstly, a study is done to get an idea of the current state of the pick-to-light technology. This ensures that the project is new and innovative. To eventually create the described smart pick-to-light system, mainly two subjects need to be researched: object recognition and cloud computing.

The object recognition is an essential component of the system. Several techniques must be researched to eventually find a method that is fast, efficient and accurate. To accomplish this, it is important that the right sensors are chosen to be placed into the pick-to-light module. Preferably, this system does not use a lot of computing power and memory.

Cloud computing is the possibility to use computing resources, applications, storage and computing power over the internet [2]. After finishing the initial research about cloud computing, a practical implementation can be done.

B. Practical realisation

Taking into account the results of the theoretical research, a proof-of-concept of the smart pick-to-light system is built.

III. Results

The master’s thesis contains the results of both the theoretical research and the practical implementation. The theoretical research takes a look at the current status of the pick-to-light technology and searches for a good solution to implement object recognition and cloud computing. Also, the created proof-of-concept is described.

A. Object recognition

Shape, colour and material practically seem the easiest detectable characteristics of the components. Therefore, a camera, to recognize shape and colour, and an inductive sensor, to separate the plastic Meccano parts from the metal ones, are chosen as sensors in the modules.

Several ways to recognize objects using a camera are researched to find the best solution for this application. Amongst these possibilities are creating custom-made feature extractors, the bag-of-features method and recognition using neural networks.

The custom-made feature extractors are programmed with the goal of extracting useful information about the objects that are depicted on the image. The output of these feature extractors can then be calculated for two images. Based on the similarities in the output, the chance that both pictures contain the same object can be calculated. The created feature extractors are: shapes per unit of area, edges per unit of area, the most significant colours and the percentage of SIFT matches [3]. These methods unfortunately are not accurate enough and are too sensitive to changes in exposure.

The bag-of-features method that makes use of SURF descriptors [4], similar to the earlier mentioned SIFT descriptors [3], reaches a recognition percentage of 66% on a dataset of 26 Meccano objects. For each object, 5 pictures are used for training and 12 are used for testing the classifier.

The best results are accomplished using a Convolutional Neural Network (CNN). A CNN is a neural network that searches for the most effective feature extractors itself using training
The resulting features are classified in a fully connected network to calculate the chance for each object that it is present on the image. To get the best results using the smallest dataset, fine-tuning will be used. This is a technique that uses an already trained network and adjusts the weights to the specific dataset. Also, data augmentation is applied to the training images. This is creating new artificial data based on the original data to create a larger dataset.

![Figure 2: 3 pairs of hard to separate products](image)

The master's thesis contains extensive research of all parameters that are required to train a CNN to find out how they affect the accuracy of the model.

Using the best parameters, an accuracy of 87.69% is reached using the dataset of 26 Meccano parts. For both training and testing, 20 images per class are used. In case a perfect metal detector would be used in combination with the camera, the system would guess the correct product 89.65% of the times. Some specific objects are hard to distinguish from one another using this neural network. Figure 2 shows three pairs of products that often get confused. Without these 6 parts, the network reaches an accuracy of 94.5%. Using an ideal inductive sensor increases this percentage to 95.4%.

B. Cloud computing

Although the database and the master program are not implemented on a cloud service, the later implementation is taken into account by programming it cloud-ready. Research has been done to analyse the advantages of using a cloud service in this system.

C. Practical creation of the smart pick-to-light system

Eventually, all researched topics are used to create a proof-of-concept of the smart pick-to-light system.

IV. Conclusion

The necessary theoretical research about existing picking solutions, object recognition and cloud computing has been done. The smart pick-to-light proof-of-concept was created successfully.

There are many opportunities left though to explore. Firstly, a completely wireless version of the modules can be made. This will increase the flexibility of the system even further. Secondly, the feasibility of truly intelligent pick-to-light modules must be researched. The modules can take over the most complex tasks that the master is currently executing, like the object recognition. Thirdly, to complete the project the master should be deployed in the cloud.

V. References


Ontwerp van een flexibele pick-to-light opstelling voor assemblage activiteiten

Student: Bram Dereeper,
Promotor: Johannes Cottyn
Academiejaar 2017-2018

I. INLEIDING

A. Warehouse picking

Het verzamelen van producten in een magazijn wordt vaak vermeld als een van de meest arbeidsintensieve, dure en tijdrovende handelingen in een niet-geautomatiseerd magazijn [1]. Om het handmatige picking proces efficiënter te maken, zijn heel wat oplossingen ontworpen. Eén daarvan is het pick-to-light systeem waarbij de gebruiker visueel naar het product geleid wordt. Ieder product is vergezeld door een module met een lampje en een knop. Wanneer het lampje bij het product brandt, moet dit artikel genomen worden. Het nemen van een product kan bevestigd worden door op de bijhorende knop te drukken.

Hoewel deze begeleiding erg visueel en efficiënt is in gebruik, is er doorgaans een gebrek aan flexibiliteit. Bij een traditionele opstelling zijn alle pick-to-light modules fysiek verbonden aan de master. Elke module heeft een vast adres waarop het aangesproken wordt door de master module. De onderdelen worden verondersteld softwarematig aan de correcte pick-to-light module verbonden te zijn, maar de juistheid van deze informatie wordt nooit geverifieerd door het systeem.

De traditionele opstelling heeft twee belangrijke nadelen ten opzichte van de slimme opstelling. Enerzijds is er steeds herprogrammering nodig wanneer een module vervangen of toegevoegd wordt. Anderzijds kunnen producten niet eenvoudig tussen twee modules verwisseld worden.

Figuur 1: Hardware van smart pick-to-light systeem

Dit staat in contrast met de hardware opstelling van een smart pick-to-light systeem, zoals weergegeven op figuur 1. De centrale master beheert de modules vanuit een externe cloud server. Indien de modules op het internet gekoppeld zijn, kunnen deze zonder herconfiguratie op eender welk netwerk met toegang tot het internet de cloud bereiken. Ook
II. DOELSTELLINGEN
Het doel van de thesis is om een gebruiksvriendelijk en slim pick-to-light systeem te ontwerpen. Na het voeren van een theoretisch vooronderzoek naar alle benodigde aspecten, worden praktische tests gedaan om uiteindelijk tot een proefopstelling te komen. De objectherkenning wordt gedaan aan de hand van 26 Meccano componenten.

A. Studie
Vooreerst wordt een voorstudie gedaan naar de bestaande systemen om zeker te zijn over het innovatief karakter van het project. Om de beschreven smart pick-to-light opstelling te verwezenlijken, moeten hoofdzakelijk twee onderwerpen theoretisch onderzocht worden: objectherkenning en cloud computing.

De objectherkenning is een essentiële component. Verschillende technieken moeten onderzocht worden om uiteindelijk tot een methode te komen die de objectherkenning snel en accuraat uitvoert. Cloud computing is de mogelijkheid om computing resources, applicaties, opslag en rekenkracht, over het internet te gebruiken [2]. Na een inleidend onderzoek naar cloud computing uitgevoerd en neergeschreven te hebben, moet uitgezocht worden hoe een praktische implementatie doorgevoerd kan worden.

B. Praktische implementatie
Rekening houdend met de resultaten van de studie, wordt het slimme pick-to-light systeem praktisch uitgewerkt.

III. RESULTATEN
De scriptie bevat het resultaat van zowel het theoretisch onderzoek als de praktische uitwerking. Het theoretisch onderzoek peilt naar de huidige stand van de pick-to-light technologie en zoekt naar de mogelijkheden tot implementatie van objectherkenning en cloud computing.

A. Objectherkenning
Praktisch blijken vorm, kleur en materiaal de best detecteerbare karakteristieken van de componenten te zijn. Als sensoren worden bijgevolg een camera en een inductieve sensor gekozen.

Verschillende manieren om objecten te herkennen met behulp van een camera worden onderzocht om zo tot de voor deze toepassing beste oplossing te komen. Onder deze mogelijkheden behoren het creëren van eigen feature extractors, de bag-of-features methode en herkenning aan de hand van neurale netten.

De eigen feature extractors werden geprogrammeerd met het doel om nuttige informatie uit afbeeldingen over de afgebeelde objecten. Op basis van de bekomen waarden kunnen dan twee afbeeldingen vergeleken worden om te bepalen of hetzelfde product al dan niet op beide afbeeldingen weergegeven is. Deze technieken bleken echter onvoldoende accuraat en te gevoelig aan variatie in belichting.

De bag-of-features methode, die gebruik maakt van SURF descriptors [3], behaalt een herkenningspercentage van 66% op de set van 26 Meccano objecten.

De beste resultaten worden met een Convolutional Neural Network (CNN) verkregen. Een CNN is een neuraal net dat aan de hand van training data zelf de meest effectieve feature extractors creëert. De bekomen features worden in een volledig
verbonden netwerk geclassificeerd. Om een zo accuraat mogelijk resultaat te bekomen bij een zo klein mogelijke dataset, wordt gebruik gemaakt van finetuning. Dit is een techniek waarbij de gewichten van een voorgetraind netwerk aangepast worden aan de nieuwe herkenningsopdracht. Ook data augmentation wordt toegepast. Dit is het creëren van bijkomende artificiële data op basis van de oorspronkelijke data.

C. Praktische uitwerking van smart pick-to-light systeem

Uiteindelijk is een praktisch werkend smart pick-to-light systeem uitgewerkt. Dit systeem voldoet aan de vereisten van gemakkelijke cloud-implementation en automatische objectherkenning.

IV. BESLUIT

Het nodige theoretische onderzoek rond de bestaande picking oplossingen, objectherkenning en cloud computing werd uitgevoerd. Ook een werkende proof of concept van het smart pick-to-light systeem werd opgebouwd.
Er zijn mogelijkheden om verder te bouwen op het resultaat. Ten eerste zou een draadloze uitwerking van de pick-to-light modules de flexibiliteit nog verhogen. Ten tweede moet de haalbaarheid van hogere intelligentie in de pick-to-light modules onderzocht worden. Deze kunnen taken van de master module overnemen zoals het herkennen van objecten. Ten derde kan het project vervolledigd worden door de centrale applicatie effectief in een cloud onder te brengen.

V. REFERENTIES

### List of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Non-smart and non-wireless pick-to-light hardware configuration</td>
<td>2</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Meccano vehicles that need to be assembled</td>
<td>2</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Smart and wireless pick-to-light hardware configuration</td>
<td>3</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Non-smart pick-to-light network connections</td>
<td>4</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Smart pick-to-light network configuration connections</td>
<td>4</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Approach to the project</td>
<td>5</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Object recognition using bag of features</td>
<td>10</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Bag of features histogram with 500 clusters</td>
<td>10</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Simple representation of relation between artificial intelligence, machine learning and deep learning</td>
<td>11</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Feedforward network [13]</td>
<td>12</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Instant cost function $\epsilon(k)$ plotted in function of the weight $w_i$</td>
<td>13</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Local and global minima in the instant cost function $\epsilon(k)$ plotted in function of the weight $w_i$</td>
<td>14</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Simplified 2D representation of data with decision boundaries</td>
<td>15</td>
</tr>
<tr>
<td>Figure 14</td>
<td>VGG-16 CNN</td>
<td>16</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Graphical explanation of convolutional layer with stride equaling 2 [21]</td>
<td>17</td>
</tr>
<tr>
<td>Figure 16</td>
<td>In- and output of max-pooling layer</td>
<td>17</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Removing background to calculate area taken by objects</td>
<td>22</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Canny edge detection and detected shapes</td>
<td>23</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Test object in front of a green background to calculate the 3 most present colours of</td>
<td>24</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Example of present colours calculation using clustering with 3 and 5 clusters</td>
<td>24</td>
</tr>
<tr>
<td>Figure 21</td>
<td>SIFT-matching between two images containing the same object</td>
<td>24</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Simplifying the process of relation between artificial intelligence, machine learning and deep learning</td>
<td>25</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Simplifying the process of relation between artificial intelligence, machine learning and deep learning</td>
<td>25</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Three images in one of the 26 classes</td>
<td>26</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Accuracy and elapsed time plot for CNN training with varying input image size</td>
<td>29</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Accuracy and elapsed time plot for CNN training with varying learning rate (logarithmic scale)</td>
<td>29</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Progress of accuracy during training using learning rate of 0.0001 (left) and 0.0005 (right)</td>
<td>30</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Progress of accuracy during training using learning rate of 0.1</td>
<td>30</td>
</tr>
<tr>
<td>Figure 29</td>
<td>Accuracy and elapsed time plot for CNN training with varying momentum</td>
<td>31</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Progress of accuracy during training using momentum of 0.0, 0.5 and 1.0</td>
<td>32</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Accuracy and elapsed time plot for CNN training with varying batch size</td>
<td>33</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Progress of accuracy during training using batch size of 32 and 64 images</td>
<td>33</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Accuracy and elapsed time plot for CNN training with varying amount of epochs</td>
<td>34</td>
</tr>
<tr>
<td>Figure 34</td>
<td>Accuracy and elapsed time plot for CNN training with varying amount of trainable layers</td>
<td>35</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Accuracy and elapsed time plot for CNN training with varying size of fully connected layers (logarithmic scale)</td>
<td>35</td>
</tr>
<tr>
<td>Figure 36</td>
<td>Accuracy and elapsed time plot for CNN training with varying dropout rate</td>
<td>36</td>
</tr>
<tr>
<td>Figure 37</td>
<td>Graphical representation of dropout [24]</td>
<td>36</td>
</tr>
<tr>
<td>Figure 38</td>
<td>Progress of accuracy during training using dropout rate of 0.0 and 0.7</td>
<td>37</td>
</tr>
<tr>
<td>Figure 39</td>
<td>Accuracy and elapsed time plot for CNN training with varying rotation range</td>
<td>37</td>
</tr>
<tr>
<td>Figure 40</td>
<td>Accuracy and elapsed time plot for CNN training with varying shift range</td>
<td>38</td>
</tr>
<tr>
<td>Figure 41</td>
<td>Accuracy and elapsed time plot for CNN training with varying shear range</td>
<td>38</td>
</tr>
<tr>
<td>Figure 42</td>
<td>Accuracy and elapsed time plot for CNN training with varying zoom range</td>
<td>39</td>
</tr>
<tr>
<td>Figure 43</td>
<td>Accuracy and elapsed time plot for CNN training with and without flipping images randomly</td>
<td>39</td>
</tr>
<tr>
<td>Figure 44</td>
<td>Accuracy after gradually removing data augmentation</td>
<td>40</td>
</tr>
</tbody>
</table>
Figure 45: Progress of accuracy during training without and with data augmentation ........................................ 41
Figure 46: Hard- and software in proof-of-concept .............................................................................................. 42
Figure 47: Concept of a smart pick-to-light module .............................................................................................. 43
Figure 48: Database schema used for proof of concept ........................................................................................ 45
Figure 49: Using the client software to add a picking list and a component range ........................................... 47
Figure 50: Using the client software to add picking order to queue ................................................................. 48
Figure 51: Performing automatic recognition of all components ....................................................................... 48
Figure 52: Using the client software to link a component to a module ................................................................. 49
Figure 53: Using the client software to start the next order in the queue ........................................................... 50
List of tables

Table 1: Mean Average Precision (mAP %) using different amount of synthetic data using 20 poses ............ 21
Table 2: Layer structure of VGG-16 CNN ........................................................................................................ 27
Table 3: Layer structure of fully connected network added to VGG-16 CNN .................................................. 27
Table 4: Standard network parameters used ...................................................................................................... 28
Table 5: Standard data augmentation parameters used .................................................................................... 28
Table 6: Messages received by master ............................................................................................................. 46
Table 7: Messages sent by master .................................................................................................................. 46
# List of used symbols and abbreviations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Abbreviation/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td></td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td><strong>D</strong></td>
<td></td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Neural Network</td>
</tr>
<tr>
<td><strong>I</strong></td>
<td></td>
</tr>
<tr>
<td>ILSVRC</td>
<td>ImageNet Large Scale Visual Recognition Competition</td>
</tr>
<tr>
<td><strong>R</strong></td>
<td></td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio-Frequency IDentification</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td></td>
</tr>
<tr>
<td>SIFT</td>
<td>Scale Invariant Feature Transform</td>
</tr>
<tr>
<td>SNN</td>
<td>Shallow Neural Network</td>
</tr>
<tr>
<td>SURF</td>
<td>Speeded Up Robust Features</td>
</tr>
<tr>
<td><strong>V</strong></td>
<td></td>
</tr>
<tr>
<td>VB.NET</td>
<td>Visual Basic .NET</td>
</tr>
</tbody>
</table>
1 Introduction

1.1. Introduction to the problem

Warehouse picking is often referred to as the most labour-intensive, expensive and time-consuming operation in manual warehouses [1]. In a lot of applications, items need to be picked out of or put away at a certain location. Components on a workbench need to be put together, products in a warehouse must be collected to dispatch or a delivery of parts must be sorted into the appropriate locations in a warehouse. Also picking components at an assembly station is a time-consuming job when no guidance is provided. When the company does not invest in an efficient picking system, these processes can be both costly and time-consuming.

Naturally a lot of solutions have been developed to make the picking process more efficient, each with their own up- and downsides. The five most popular paperless warehouse picking technologies are systems using a barcode handheld or an RFID tag handheld, voice picking, traditional pick-to-light and RFID pick-to-light [1]. When using a system with a handheld, a tag on the product or product location has to be scanned while picking. If the tag is fixed to the product, this enables the system to verify whether the right product was picked.

In the pick-to-voice system, the picker has a headset with a microphone. A voice guides the picker to the right product. After picking, the operator acknowledges the pick by reading a certain number that is printed on the product as a verification.

For picking products at an assembly station, these methods are cumbersome. A better technology for this specific case would be pick-to-light. Every item location is marked with a light and a button. When the light switches on, this specific product must be picked and the picking must be acknowledged with a push on the button. But the available modules are not smart: they light up when their master requests them to and let their master know when the button has been pressed. The modules have no idea what product they are lighting up for and therefore have to be addressed with a fixed address. As a result, when a new pick-to-light module is added or two products switch places, the system needs to be reconfigured in the master module by adding the address of the new module or reassigning the components. For example, on a workbench, it is possible that all the products are switched when another product has to be made. Or the operator might notice that it would be more efficient if two products were switched places. The reconfiguring in these cases will, in the non-smart case, be time-consuming, complicated and expensive.

In this thesis, the focus will be on the pick-to-light technology on an assembly station in an assembly line. This technology is already widely available, but usually in a non-flexible and non-smart way. Communication and power supply cables need to be provided for each pick-to-light module. This inflexibility is clearly visible on figure 3: Every pick-to-light module needs to be directly connected to the master module to communicate. Consequently, the modules are bound to be relatively close to the master module. This contrasts with the smart system on figure 3: Every module connects to the cloud. If necessary, this cloud can be accessed from anywhere in the world, therefore offering great flexibility. The modules are equipped with sensors to automatically recognise the components they contain. Both adding or removing pick-to-light modules and switching the content of the modules have become a matter of only performing this action. No extra reconfiguration needs to be done upon doing this since everything can be reassigned automatically in the smart system.
1.2. Objectives

The main objective is to create a proof of concept using the specific case of a Meccano assembly line. Two vehicles must be assembled on a line containing four workstations. On each workstation, the needed components for the assembly of a specific Meccano model are positioned in pick-to-light modules to accomplish efficient assembly. When orders for different models are received, the components on the workstation need to be switched to match the components of the other product. In the traditional pick-to-light system without object recognition, this is a tedious task since reconfiguration of the system is needed after switching the components. Using the object recognition in the smart alternative, the switching of these products happens faster and with a smaller error rate.
needs to be picked, each pick-to-light module will try to determine which component it contains based on the list of all possible components, narrowing down the almost infinite amount of parts in this world to just the components available in the warehouse or assembly station. Very important is that enough information is available about each part to help the pick-to-light module to recognise the part it is accompanying using one or multiple sensors.

This smart system will have multiple advantages relative to the currently widely available pick-to-light systems. Firstly, the configuration can be changed and expanded without any reconfiguring. Secondly, the addressing of the modules will now be more intuitive: the pick-to-light system will for example not be asked to light up module 104, but to light up the module that accompanies the small black tires. Thirdly, a pick-to-light module can connect to the cloud from anywhere and is not bound to be directly connected to the master module. Figure 3 shows that connecting to the cloud server is possible over any network that is uplinked to the internet. It is visible that not only the pick-to-light modules connect to this cloud server. The other devices will connect to the cloud as well with the goal of receiving information about the picking process, adding a new picking job to the queue or adjusting picking parameters.

By comparing figure 1 and figure 3, one might get the idea that connecting devices and pick-to-light modules in the smart version is a lot more difficult than in the standard configuration. From a software point-of-view, this is most definitely not the case. When a cloud server is used, like in figure 5, every module will write to the same IP-address. When a device or module is moved from one network to another, it will still use the same information to access the cloud and will still be able to function without any reconfiguration. In figure 4, the addressing will differ when a device is moved from one network to another. Accessing the master module from another network might even require making new configurations in the router of the network with the master module. This proves that the smart configuration makes sure that the final system is hustle-free.
The only drawback of the smart system is that some information about the parts of the product has to be known to perform automatic object recognition. Depending on how much information will be needed from each component, gathering this information might be time-consuming. Therefore, a solution must be found to recognise objects with as few required information as possible.

1.3. Approach to the project

Figure 6 shows the taken approach to the project in a flowchart. In the first semester, mainly research will be done. The research starts off by analysing the current state of the pick-to-light technology to determine the originality of the proposed smart pick-to-light system. Once the conclusion is made that a similar system does not exist yet, the problem has to be broken apart into areas of needed research. Mainly knowledge about the topics of cloud computing and object recognition will have to be gained. Then the exploration starts: a good and thorough theoretical knowledge is useful to make sure the best decisions are made during the practical realisation.
When enough knowledge is gained, a theoretical system can be created. This system should already take the final application of the proof of concept into account. The practical system will be used in the assembly of Meccano vehicles. With this knowledge, the parts of the used Meccano vehicles can be used to test the performance of the object recognition system. Based on this theoretical model, the practical choices can be made. The cloud service provider and object recognition system can be chosen. Eventually, all tested components must be combined to create a working proof of concept to apply on the described Meccano case.

Figure 6: Approach to the project
2 Background research

2.1. Literature research

The goal of this master’s thesis is to design a working proof of concept of a smart pick-to-light system. To accomplish this, the problem is broken down into its two most challenging parts: cloud computing and object recognition.

2.1.1. Cloud computing

Cloud computing is the ability to use computing resources - applications, storage and processing power - over the internet [2]. The actual cloud is a big data centre and its low-level software that runs the cloud [3].

In traditional computing, where a local server is installed in the company, the computing infrastructure is owned by the company that uses it. This is also called on-premise computing [2]. This model has multiple problems. Firstly, the initial investment is big [2]. When a web developer wants to publish his first website, he will have to buy his own server without knowing whether the website will be a success or not. Therefore, the website developer is taking a financial risk. Secondly, the amount of available server space is limited. Therefore, always one of the following problems will occur: the server is either underused (underutilisation) or the server will not be able to handle all requests (saturation) [2] [3]. It is hard to predict how much server space will be needed: a website might suddenly become very popular. The initially provided server will not be able to handle these requests and will fail to answer all requests. As a reaction, the web developer will order more servers. But it might take weeks before the new servers are operational. In an ideal world, the extra computing resources would be immediately available. The fixed available computing resources induce a waste in resources and thus in money. Because servers in traditional computing are purchased based on the peak demand, on average only 5 to 20% of the server computing resources in the world is actually used [3]. Thirdly, installing and maintaining a server room is an expensive and difficult task. Software updates, maintenance, troubleshooting and replacing broken parts can only be done by specialised staff. Usually, neither the main objective nor the core competence of these companies is managing data centres [2]. It would be better if a company specialised in server rooms would do the design and maintenance. That way, the company using the servers can focus more on their actual main objective.

Cloud computing solves these three problems. When a company uses the cloud to provide its services, there is no initial investment. The company only must pay the owner of the cloud for the actual usage of resources. This is called metered payment or pay as you go [2]. In the cloud, software automatically scales the provided resources in data and processing power for a certain service. When a peak in requests comes for this service, the cloud administrator will allocate more resources to this app, enabling it to answer all requests. In this way, the server can provide enough resources for the peak requests without underutilisation on times when there are less requests. When the service gradually becomes more popular, no investment in new servers will have to be done.

The available computing resources seem infinite to the user of the cloud [3]. Another advantage of using cloud computing is that the user does not have to install and maintain the servers. The administrator of the cloud will take care of the proper maintenance of both the hardware and software and will use the vast amount of knowledge they have to create a secure and stable system. In cloud computing it is also possible to spread the requests over multiple servers in the same or even in different datacentres. This way, the uptime, the time the service is available, is maximal.
2.1.2. Object recognition

2.1.2.1. Introduction to object recognition

An important objective of the final pick-to-light system, is that no reprogramming is necessary when components are added, removed or switched places or when modules are removed or added. Some ‘quite smart’ picking solutions have already been developed. A good example is the inBin (Intelligent Bin) [4], developed at the Fraunhofer Institute. It is a low energy wireless smart picking system. Each smart module, or smart bin in this case, stores all the necessary data about the product it contains. No master module orders the bin to light up. When a list of picks is transmitted through the web interface, all modules can access this data. Based on the information each bin knows about its own product, all modules will communicate with each other to agree on a picking order. The decisions can be made on position, but also on other important data like expiration date. It is obvious that the inBin is a smarter solution than the case where the modules just wait for a signal of the master to light up.

But although being smart, the inBin is still not as flexible as the end product of this thesis aims to be. When the components in the bins are switched around, the bin will not notice. Each time the components are switched, they need to be reassigned to the correct bin.

The inBin would be a good starting point to create the ‘truly smart’ pick-to-light system that recognises the products it contains. The inBin has a lot of nifty features like low energy usage, power generation out of light and motion, adding picking orders... The component recognition is a big challenge in this project and might only be possible up to a certain point: even for a human it is not easy to say whether a certain bolt has a diameter of 5mm or 6mm.

The biggest part of the object recognition will have to be done using a camera. Without a camera, it is impossible to tell apart a sufficient amount of the components. A lot of objects would have to be recognised with the help of a human operator, making the system inconvenient and useless. Because a camera can detect all visible features like colour, shape..., the other sensors are only useful if they detect object properties that are invisible to the eye (and therefore to the camera). Examples of these properties are weight, magnetic properties, conductivity, capacitance, elasticity, toughness, melting point... A lot of material properties are of course useless in object recognition: to detect the melting point or toughness, the object will have to be damaged. Detecting the elasticity is too complex. The detected capacitance might change because of the number of objects that are currently in stock. Also, the conditions might influence it. The weight cannot be used to identify the object because only the combined weight of all the components in one box can be weighted. And since the contents are still unidentified, it is impossible to know the weight per component and it is therefore impossible to link the weight measurement to a certain component. When the module has detected which component is present in the box using the camera and the magnetic sensor, a weight sensor could be used, in combination with the specified weight per component, to detect the number of available components. In the end only one property that cannot be detected by a camera seems to be useful in this application: whether the object is (partly) magnetic. For this purpose, an inductive sensor will be installed in every module. If it is known whether the object is (partly) magnetic, a first selection can already be made based on the measurement. This will reduce the needed processing time of the computer because it does not have to check a part of the components list for a match.
2.1.2.2. Vision in object recognition

2.1.2.2.1. Template matching

To make the system work, a robust method for recognising objects on a picture must be found. One way to do this, is by template matching. Template matching is a technique in digital image processing for finding small parts in a picture which match a template image. It can be used in manufacturing as a part of quality control, a way to navigate a mobile robot, or as a way to detect edges in images [5]. Template matching can be performed in OpenCV. The template image will then simply be slid over the query image. Several comparison methods are implemented to detect how much the template image differs from the specific part of the query image [6]. While very effective for certain engineered environments, where object pose and illumination are tightly controlled, template matching becomes computationally infeasible when object rotation, scale, illumination, and 3D pose may vary [7]. Since the components will not be especially carefully placed in a way that the camera can easily detect them and the lighting conditions might vary, template matching is not an option.

2.1.2.2.2. Feature extraction

It would seem very logical to extract features of an object on an image that make sense to a human like: amount of squares, roundness, number of holes, shininess... Although this is possible, and was also explored in this thesis, it is not easy to program a useful feature extractor based on these features. Instead of human-friendly features, computer-friendly features can more efficiently be extracted. These features might be less obvious to the human eye. This is not a problem: a lot of algorithms have been created to extract object features from a picture. Once these features have been extracted and stored, they can be used to efficiently search a new image for these features. If enough features show a match, it can be assumed that this object is portrayed on the new image.

Own feature extractors

The method for feature extraction does not always need to be complex when the set of objects that needs to be recognised is known. A selection of distinctive simple parameters can be detected and this way the objects can be recognised. Examples of these features are the most dominant colours and density of circular shapes in the objects.

Rapid Object Detection using a Boosted Cascade of Simple Features

Rapid object detection using a boosted cascade of simple features is a popular algorithm for object recognition. A big advantage of this algorithm is that the detection is fast. It is able to run on low-power devices. Using an old 700MHz Intel Pentium 3 processor from the year 2000, faces on a 384 by 288-pixel camera feed can still be detected at 15 frames per second [8]. The images can be processed that fast because a feature-based system is used instead of a pixel-based system. The algorithm makes use of three kinds of features: the two-, three- and four-rectangle features. The value of a two-rectangle feature is the difference between the sum of the pixels within two rectangular regions. The regions have the same size and shape and are horizontally or vertically adjacent. A three-rectangle feature computes the sum within two outside rectangles subtracted from the sum in a centre rectangle. Finally, a four-rectangle feature computes the difference between diagonal pairs of rectangles [8].

The best features are selected by an algorithm by providing a big set of pictures of the item to be recognised. In the paper, a face detector is created by providing the algorithm with 4916 pictures of faces. These selected features may seem like an abstract concept, but some of them make sense to a human as well. The first feature that is selected for the face detection focuses on the property that the region of the eyes is usually darker than the region of the nose and cheeks.
To calculate these rectangle features in an efficient way, the paper proposes to create the ‘integral image’ before calculating the rectangle features. The integral image is calculated by giving a value to each pixel that equals the cumulative sum of the values of all the pixels above and to the left of this pixel. In this way, any rectangle feature can easily be calculated with a sum of only four digits, as shown in the paper. This speeds up the object recognition. To even further increase the recognition speed, a cascade system is introduced to quickly discard background regions. Thus, more computing can be spent on the most promising areas.

Once the distinguishing features have been selected, these are saved into a text file. This file can later be used to do fast object recognition without having to recalculate the rectangle features.

Although this automatic feature detection system seems promising for our purpose of detecting objects, it has a couple of problems. The first problem is that 4916 images were used in the paper to select the rectangle features. It is obviously not feasible to require thousands of pictures of each component that is used in the warehouse. The second problem is that the square features have a certain orientation. The two-square feature that describes that the region of the eyes is darker than the region of the cheeks, assumes that the darker area will always be positioned above the light area. Therefore, the object will only be detected when it is positioned under the right angle, with a small degree of freedom. Although the detection is fast and both the used processing power and power usage are low, the big amount of input images to select the features and the lack of freedom in orientation make this algorithm unusable in our application.

Object Recognition from Local Scale-Invariant Features

David G. Lowe introduced in his paper ‘Object Recognition from Local Scale-Invariant Features’ a way to overcome the problems that the previously mentioned algorithm had. His approach makes use of local image features, making sure that an object can be detected even if a part of it is hidden on the input image. These features are also invariant to image scaling, translation, rotation and partially invariant to illumination changes. Planar shapes will also be recognized at an angle up to 60° away from the camera. A rotation of up to 20° of a 3D object is not a problem either [7]. We can conclude that the features selected by the algorithm of Lowe are way more invariant to any kind of changes than in the algorithm as described by Viola and Jones. The paper of Lowe presents a method for image feature generation called the Scale Invariant Feature Transform (SIFT). This approach transforms an image into a large collection of local feature vectors, each of which is invariant to image translation, scaling, and rotation, and partially invariant to illumination changes and 3D projection. The local feature extraction can just be done based on one picture of the item.

With the solution defined by Lowe, two problems still occur. Firstly, only one side of an object is shown on the picture from which the features are extracted. Therefore, only this side of the object will be recognised with an allowed variation on the angle of 60° for 2D objects, for example a credit card, and 20° for a 3D object. Taking pictures of all sides of the object and search each of these pictures for features, can be a solution to this problem. The second problem is the immunity to scale differences. We need this because it is very inconvenient if the item on the picture needs to have a certain pixel width to be recognised. But the disadvantage is that when for example a big and a small bolt are put next to each other on a picture, they will both be recognised as a bolt. Therefore, it is impossible to see the difference between different sizes of a certain shape.

Bag of features

The bag of features method is used for image classification. To train a bag of features, a set of training images must be provided for each class. All descriptors of each of these images are calculated. These descriptors can be extracted from the dataset with training images using SIFT, SURF or another algorithm. SURF descriptors are similar to the earlier described SIFT descriptors [7], but are more robust and can be calculated faster [9]. Once all descriptors are calculated, the most distinctive ones per category are kept. Then the features are balanced. This means that weaker features will be discarded of the classes that have more features than the class with the
least number of features until all classes have an equal number of features. This prevents that the resulting classifier will be biased towards a certain class. Now K-means clustering is applied to the features. This machine learning method sorts each feature into one of the $k$ clusters. Then the histogram is created for each class, as seen on Figure 7 depicting the number of descriptors that got sorted into each individual cluster. The shown histograms were assigned randomly to a class and are only for illustration purposes. Figure 8 shows a real example of the resulting histogram after clustering all features into 500 clusters.

To classify an image into one of the classes, the descriptors of the query image are calculated. These get sorted into the $k$ number of clusters to eventually create the histogram that depicts how many descriptors were classified into each cluster. Figure 7 shows an example histogram for a certain query image. The image gets classified into the class that has the closest matching histogram [10]. In Figure 7 it is clear that the histogram of the query image has the closest match with the histogram of the plane class. All calculated descriptors were divided into one out of 500 clusters.

Figure 7: Object recognition using bag of features

Figure 8: Bag of features histogram with 500 clusters [10]
2.1.2.2.3. Deep learning

**Introduction to artificial intelligence, machine learning and deep learning**

Artificial intelligence denotes the behaviour of a machine which, if a human behaves in the same way, is considered intelligent [11]. Humans are exposed to an uncountable amount of sensory data every second of the day and are somehow able to capture the critical part of these data [12]. This selected critical input is processed by the brain for future use and decision-making. This forms the basis for machine learning, a subset of artificial intelligence: a machine gets exposed to big sets of data and will use the useful part of it to train itself to later on classify unseen data. It usually makes use of a deep neural network (DNN), which is inspired by the actual functioning of the human brain. Recent neuroscience found out that the sensory signals do not get explicitly pre-processed by the neocortex of the brain, but rather get propagated through a complex hierarchy of modules that, over time, learn to represent observations based on the regularities they find [12]. The relation between artificial intelligence, machine learning and deep learning is graphically shown in [Figure 9](#).

![Figure 9: Simple representation of relation between artificial intelligence, machine learning and deep learning](#)

**Neural Network: basic operation**

This simplified theory of neural networks is based on the course Artificial Intelligence, taught at Cardiff University by professor Michael S. Packianather [13].

A neural network is, like the name suggests, a network that consists of connected neurons. It is stacked in multiple layers: an input layer, one or multiple hidden layers and an output layer. This can be seen on [Figure 10](#) where each circle represents a neuron and an arrow represents a connection between two neurons. Data are fed to the network through the input neurons. Each neuron accepts an input value. Some only accept binary inputs, others can handle integers or real numbers. In a simple image classification task, the two input neurons in [Figure 10](#) could for example be fed with data like the amount of green in the picture and the amount of blue. This data flows propagates along the connections through the hidden layers where calculations are done. Eventually the network produces an output at the output layer. The first output neuron in [Figure 10](#) can produce the probability that the input picture contains a boat. The second output neuron can give the probability that the input picture contains a dog. Because boats are usually piloted in water, this percentage will be high when a lot of blue is present in the input picture. Because a dog usually plays in a meadow, that particular output neuron will produce a high percentage when a lot of green is detected. The patterns and correlations do not need to be programmed into the network. The network trains itself. For training the simple network that distinguishes boats from dogs, supervised learning can be used. This means that the output is known. The network can for example
be trained with 250 images of dogs and 250 images of boats. In training mode, this will cause the network to adapt in such a way that the output gets closer to the expected output for a certain input. When the input contains information about the picture of a boat, the expected output is 100% for the first output neuron and 0% for the second output neuron. When information about the picture of a dog is fed into the network, the expected output is 0% for the first output neuron and 100% for the second one.

Figure 10: Feedforward network [13]

Figure 11 shows one of the neurons in a neural network. The inputs $x_1$ to $x_n$ are each multiplied by their corresponding weight $w_1$ to $w_n$. This weight depicts the strength of the connection. The net input $\text{net}(k)$ of this neuron is calculated by adding all resulting values: $\text{net}(k) = \sum_{i=1}^{n} x_i \times w_i$, with $k$ being the iteration number. This means that when the first item of the dataset is fed into the network, $k$ is equal to one. For the second item, $k$ is equal to two and so on. This is a linear calculation. To be able to represent non-linear problems, a non-linearity is introduced into the network by feeding the $\text{net}(k)$ through a non-linear activation function. This way the output of the neuron is calculated: $y(k) = f(\text{net}(k))$. If this is a neuron in the output layer and the network is in training mode using supervised learning, the error is calculated by subtracting the network output from the desired output: $e(k) = y_d(k) - y(k)$.

Figure 11: Neuron

Based on the error $e(k)$, the instant cost function is defined as $\varepsilon(k) = \frac{1}{2} e^2(k)$. The square makes sure that negative and positive errors do not cancel each other out. This cost function needs to be minimized to create an effective network. Based on the cost function, decisions are made about how much the weights have to be
changed. The weight for the next iteration \( k + 1 \) is calculated according to the delta rule:

\[
\Delta w_j(k + 1) = w_j(k) + \Delta w_j(k)
\]

The weight change can be calculated with the gradient descend algorithm. The goal of the gradient descend algorithm is to adjust the weights in a way that the instant cost function reaches its minimum. A minimum in the instant cost function is reached when the derivative of the instant cost function \( \varepsilon(k) \) to the weight \( w_j(k) \) equals zero. The formula for the weight change according to the gradient descent algorithm can therefore be written as in equation 1.

\[
\Delta w_j(k + 1) = -\eta \frac{\delta \varepsilon(k)}{\delta w_j(k)} \quad \text{(Equation 1)}
\]

with \( \eta \) being the learning rate. The learning rate is a multiplier that influences how big the performed weight changes are. The minus sign before the derivative makes sure that the algorithm searches for minima and not for maxima. Figure 12 shows an example of a calculated weight change using the given algorithm. When making use of a momentum \( \beta \), the formula for the gradient descent algorithm can then be written as in equation 2.

\[
\Delta w_j(k + 1) = -\eta \frac{\delta \varepsilon(k)}{\delta w_j(k)} + \beta \Delta w_j(k) \quad \text{(Equation 2)}
\]

The larger the learning rate \( \eta \), the bigger the calculated weight changes are. Assuming that the learning rate is chosen reasonably small, equation 1 will only take steps that lead to a smaller instant cost function. When a local minimum is reached, the weights will not change anymore. This is no problem if the instant cost function in function of the weight has only one minimum. Then the weight will converge to the global minimum. But as depicted in Figure 13, the instant cost function usually has multiple minima. Only one of these minima is the optimum and leads to the lowest error rate. This is the global minimum. The other minima are local minima and do not lead to the best solution. To reduce the chance of getting stuck in a local minimum, a momentum term is introduced into the gradient descend algorithm. This term depends on the previous weight change. This makes it possible to climb out of a local minimum, possibly reaching a better minimum. The larger \( \beta \) is chosen, the more climbing can be done on the instant cost function curve.

To adjust the weights in a whole network instead of in one neuron, the backpropagation algorithm is used. During training the data are fed into the network and the actual output of the network is compared to the known desired output. The difference between these two values is used to adjust the weights in a backwards motion. When the output of the network is a single value, the error \( e(k) \) is the difference between the desired network output \( y_d(k) \) and the actual network output \( y(k) \).

\[
e(k) = y_d(k) - y(k) \quad \text{(Equation 3)}
\]

The new value of the weights can then be calculated with backpropagation using equation 1. Neural networks usually have multiple outputs, each representing a class. These classes can be objects like a dog, a cat or a boat. When an image is fed through the network, the output of each class will produce the probability that this object is present in the image. There are multiple values now and another formula than equation 3 must be used to calculate the error \( e(k) \). Different loss functions can be used to calculate the error. A popular loss function is the Euclidean distance.

To illustrate the use of a loss function, a simple example is given. Imagine a neural network with three classes: dog, cat and boat. During training, the image of a dog is propagated through the network. The neural network has an output of 0.75 for dog, 0.15 for cat and 0.1 for boat. The desired value is 1 for dog and 0 for both cat and boat. Using the Euclidean distance as loss function, the error \( e(k) \) is then calculated as

\[
e(k) = \sqrt{(0.75 - 1)^2 + (0.15 - 0)^2 + (0.1 - 0)^2} = 0.31.
\]
The weights get adjusted every time a batch has been fed through the network. Since the datasets are often too large to load into the working memory of the computer, the data gets split up in batches. A typical batch size for images is 32. After calculating the output of the network for every element in the batch, the weights are adjusted according to the loss of the total batch. An epoch is one forward and backward pass of all training examples. Multiple batches are collected in an epoch and multiple epochs are performed during the training of the network.

It can be concluded that to create a network that accomplishes its task well, one does not have to be an expert in the particular field of the application. In the example of classifying images with dogs and boats, it is not necessary to have the knowledge that boats usually are photographed in a blue environment and dogs in a green one. The network can figure out this correlation by itself based on the provided training data. But the network needs useful input, a good structure and good training parameters to converge to an optimal solution. Therefore, knowledge in the area of building and training neural networks is required.

![Figure 12: Instant cost function $\varepsilon(k)$ plotted in function of the weight $w_j$ [13]](image1)

![Figure 13: Local and global minima in the instant cost function $\varepsilon(k)$ plotted in function of the weight $w_j$ [13]](image2)
Deep Neural Network (DNN)

DNNs have recently shown outstanding performance on image classification tasks. On the yearly ImageNet Large Scale Visual Recognition Competition (ILSVRC), error rates dropped drastically thanks to the development of DNNs [14]. DNNs are deep, contrary to Shallow Neural Networks (SNNs), because they contain a lot of hidden layers. This deep architecture makes it possible for the network to learn more complex models than SNNs [15]. Figure 14 shows the same dataset, containing two classes, twice. The red data points belong to class x and the blue data points to class y. After training, the neural network has created decision boundaries represented by the black line. The decision boundary on the left picture was established with a very shallow network: it cannot handle any non-linear complexity. Because the data is non-linear, some of the data points got classified wrongly. The decision boundary on the right was established through the training of a deeper network. The network can handle more complexity and can therefore classify all data points correctly. When an unseen data point is fed into the network, it will be plotted in this space. Based on the position of the data point compared to the decision boundary, the classifier predicts the correct class.

Convolutional Neural Network (CNN)

One deep learning approach that has achieved high accuracies in classification and detection of objects on images is the Convolutional Neural Network (CNN) [16]. With the use of CNNs, machine learning algorithms implement a similar system to the one in the neocortex of the human brain [12]. Convolutional Neural Networks are particularly designed for use on two-dimensional data, such as images or videos [12]. Images could be thought of as two-dimensional arrays of numbers or matrices. This is a very crude representation though since these numbers have a large spatial coherence [17]. For images, each two-dimensional array is called a channel. A black-and-white image consists of one channel. A full-colour image consists of three channels and is therefore represented by a three-dimensional array with size $m \times n \times 3$ with $m$ being the width of the image and $n$ the height.

In a traditional neural network used for object and pattern recognition, a hand-designed feature extractor gathers the relevant information and eliminates the irrelevant features. A fully-connected network is trained as a classifier to categorize the resulting feature vectors into classes. More interesting would be to eliminate the hand-designed feature extractor and feed the images directly to the network. Using backpropagation, the first layers can be trained to create appropriate feature extractors [18]. Since all weights are learned using backpropagation, the CNN can be considered creating its own feature extractors [18].
Figure 15 shows the layer structure of the VGG-16 CNN. It consists of convolutional layers, each accompanied by a Rectified Linear Unit (ReLU) and max-pooling layers. These layers form the part of the network that perform the feature extracting on the images. The last three layers form the fully connected network that is added behind the feature extraction layers to classify the image based on the output of these extractors. In contrast to the fully connected network that is present in many kinds of neural networks, the combination of convolutional, ReLU and max-pooling layers is very specific to CNNs.

The first layer in a CNN is always an input layer that accepts an image of a fixed size with a fixed number of channels. The next layer is a convolutional layer. This layer is defined by 3 parameters. The first parameter is the kernel size or filter size. This is the size of the filter matrix. The second important parameter is the weights. The weights are the values in the filter matrix. The third parameter is the stride. Together these parameters result in an unambiguous result.

The filter matrix will be multiplied with specific subparts of the layer input. The start of this multiplication process can be imagined as a flashlight shining over the input of the convolutional layer, only showing the left top part with a size equal to the filter matrix size \[19\]. Figure 16(a) represents an image of seven by seven pixels with only one channel. The imaginary flashlight shines in a way that only the pixels within the red square can be seen. This submatrix is multiplied with a filter of size 3x3x1. To multiply this part of the image and the filter matrix, this filter must have the same depth as the input \[19\]. The elements in the output of this multiplication are summed to get one value as a result. This value is then added to the top left of the output matrix. This value is represented by the red square in figure 16(b). The multiplication with the filter matrix will be done with other parts of the input as well. The imaginary flashlight will scan over the input with a certain step size to isolate different parts. This step size is called the stride. In figure 16 the stride is two since the next extracted coloured rectangle is always two steps further. With a stride of two and a 3x3x1 filter, nine values can be calculated by convolving in the 7x7x1 picture. These values are all calculated and added to the output matrix in their correct position. This output matrix is also called a feature map. One convolutional layer can have multiple filters. Each filter creates its own feature map. These are all stacked on top of each other in the output like channels in an image \[20\].

Convolutional filters are not a new concept. They are not only used in CNNs, but also in image editing programs. The effect of some filter matrices is widely known. Examples are blur, edge detect and emboss filters. Of course, a lot more possibilities are possible, like filters than filter out curves, colours or even parts of objects like a nose.
These elementwise multiplications and additions are all linear operations. To introduce non-linearity into the network, in order to be able to solve non-linear problems, it is conventional to add a non-linear layer behind each convolutional layer. Because of its computational efficiency, usually a ReLU layer is used to accomplish this [21]. Each individual matrix element at the input of the ReLU layer is subjected to the ReLU function, which can mathematically be written as:

\[
\begin{cases} 
    x < 0: y = 0 \\
    x \geq 0: y = x 
\end{cases}
\]

This function changes all negative values to zero and keeps all zeros and positive values. The output dimensions are the same as the input dimensions.

Figure 15 shows that after every second or third convolution-ReLU pair, a max-pooling layer is used in the VGG-16 CNN. A pooling layer is used to downsample matrices and decrease their dimensionality. This is accomplished by passing a filter of size \(n \times n\) over the input with stride \(n\). The filter will therefore only see each element once. Instead of doing a multiplication, the output of the filter is equal to the largest value in the selected submatrix. This process is graphically represented in Figure 17.

Convolutional-ReLU-pairs and max-pooling layers can be stacked in any order with the goal of creating effective feature extractors. When the training is started, all values of the filters are randomly chosen. After propagating
the first batch of images through the network, the loss function is calculated to determine the performance of the network. Based on this information, the weights of the filters are adjusted. If the network is trained correctly, these filters will extract information that is useful to classify the images using the fully connected network.

**Transfer learning and fine tuning**

The filters in the CNNs need to be trained with very large datasets of labelled images to accomplish good results. In the ILSVRC-2012 competition 1.2 million labelled training images divided into 1000 classes were provided and used for training [14]. These images were collected from the web and labelled by human labellers using Amazon’s Mechanical Turk crowd-sourcing tool [14]. If the network to recognise parts in a pick-to-light system would be trained from scratch, a lot of effort would have to be made to gather 1200 images per product.

Two techniques exist that make creating an accurate classifier with a small dataset possible: transfer learning and finetuning. These methods make use of a pretrained network, for example one of the networks that was trained for the ILSVRC competition, to classify another dataset than it was originally trained for. The output classes do not need to correspond. Many deep networks trained on natural images share a common phenomenon. The filters in the first layers extract general features like colour blobs and textures. These first layers will therefore also perform well for other datasets with images containing completely different objects than those it was trained with. However, the filters in the last layers tend to extract features that are more specific to the training data. These layers are specialised to a certain set of images or classes. These might therefore produce more insignificant data when pictures with other objects are fed through the network [22]. The fully connected network, positioned after the last convolutional layers, is completely specialised in the classes that were used for training the original network.

To overcome the issues of specialization by the higher layer neurons to their original task at the expense of performance on the target task [22], some adjustments to the pretrained networks have to be done. The original fully connected network is replaced by a new fully connected network that is initialised with random weights. The number of neurons in the last layer of this added network is equal to the number of classes. Each output will produce the recognition probability for one class. When transfer learning is used, all weights of the convolutional layers are maintained. When fine tuning is applied, some of the feature extractor layers, usually the more specialised ones, are trained with the new dataset as well.

When the target task of the network resembles the original task, then transfer learning is usually applied. Otherwise finetuning will be used during training.

**Data augmentation**

Deep learning methods, such as not-pretrained CNNs, require approximately 5000 pictures per class to have a good performance [23]. Data augmentation is a technique where linear and nonlinear transforms are done on the training data to create synthetic new training images [16]. The first group of these transforms are the geometric transformations. This includes performing rotations and shifts on the images. Also, a random zoom and homographic projection can be applied. The rotation, the zoom and the homographic projection make the trained network more rotation, scale and perspective invariant. Colour transformations are also often applied to represent different lighting conditions of the object. Also blurring and noise addition are popular distortions. These techniques can be used to produce accurate recognition rate with only as much as 20 training images per class [16]. The results, displayed in table 1, prove that data augmentation can be useful to create a more precise classifier.

Examples of how images are affected by the various data augmentation methods are given below.
- Rotation range
The image is rotated over a random angle within the given interval.

- Shift range
The image is shifted over a random distance within the given interval.

- Zoom range
The image is zoomed with a random zoom rate within the given interval. The horizontal and vertical zoom are not per definition equal, possibly causing distortion of the image.
- Shear range
The image is skewed over a random angle within the given interval.

- Flip
The image is randomly flipped or not flipped. This can be done both horizontally and vertically.

- Channel shift
The colour channels are shifted with a random amount within the given interval, causing a distortion in the colour scheme of the image.
- **Gaussian**
A random amount of blur within the given interval is applied to the image.

- **Noise**
An effective way of adding noise as a data augmentation technique, is randomly choosing one out of multiple kinds of noise and applying it to the input image [16]. These include gaussian, salt and pepper, Poisson and speckle noise.

Table 1: Mean Average Precision (mAP %) using different amount of synthetic data using 20 poses

<table>
<thead>
<tr>
<th>Model</th>
<th>6250 images</th>
<th>12500 images</th>
<th>25000 images</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16</td>
<td>94.32%</td>
<td>97.42%</td>
<td>96.36%</td>
</tr>
</tbody>
</table>
2.2. Experimental research

2.2.1. Object recognition

By doing a thorough literature research, the most promising methods for object recognition were selected. Since there are a lot of popular object recognition methods, one can assume that the best object recognition method does not exist. Each of these methods has advantages and disadvantages that make it less or more suitable for a certain application. That is why an experimental research needs to be done as well, keeping the application of the proof of concept in mind. Therefore, the performance of the object recognition will be tested on the set of 26 components that are needed for the Meccano case.

2.2.1.1. Object specific recognition methods

2.2.1.1.1. Description of the recognition methods

Using an open source library for image processing called OpenCV [6], simple features can be extracted from the images. A well-chosen feature extractor gathers information in the image that has a relation to the photographed object. When the feature extractor is applied to an unseen image, the output feature can give an indication whether a certain object is displayed in this image. Four application-specific feature extractors were created.

Shapes per unit area

After applying an edge detector to the image, OpenCV [6] can be used to recognise certain shapes in these edges. Then the number of triangles, squares, rectangles, pentagons and circles on an image can be calculated. When this is related to the surface taken by the objects on the image, then the density of these shapes per object area unit can be calculated. This way the same result should be returned no matter how many objects there are displayed on the image, given that they are not overlapping.

To calculate the area of the objects, the background is removed from the image. This can be seen in figure 18 where image (b) is the result of replacing the background of image (a) with plain black colour.

![Figure 18: Removing background to calculate area taken by objects](image)

Removing the background also makes it possible to perform a clean edge detection on the picture. Figure 19(a) shows the result of performing canny edge detection on figure 18(b). Based on the edges, a search algorithm to detect shapes can be executed. The detected shapes are marked on figure 19(b). Rectangles are marked in green, circles are marked in red and pentagons are marked in blue.
Figure 19: Canny edge detection and detected shapes

The similarity in shape distribution per unit of object area between two images is calculated as an Euclidean distance in n-dimensional space, with n being the amount of different kinds of shapes to detect. A small Euclidean distance infers that the same object might be depicted on both images.

**Edge per unit area**

An edge filter can easily be applied to a picture using OpenCV. A result using the popular Canny edge detector is shown in figure 19(a). Since the area of all objects on the image is known, it is possible to calculate the length of edges per unit of object area. Because of the fine thread, the amount of edge per area should be a large number for screws. Other objects with big flat surfaces will have a small edge per area value.

**Object colour**

When the objects that need to be recognised have a range of different colours, this might be a very interesting feature to extract. Although the average colour of an object does only give valuable information when the object mainly consists of one colour, the k most present colours always contain useful information. This can be done using k-means clustering. All pixels are represented in 3D RGB space. The unsupervised artificial intelligence technique called k-means clustering then divides the pixels into k clusters, each containing pixels with similar colours. The averages of each cluster then represent one of the k most present colours in the object. Figure 20 shows an example of an object. After subtracting the background, the most present colours can be calculated. The result is graphically depicted in figure 21. The colour of each part in the bar represents the average of the cluster. The width represents the percentage of pixels that were sorted into this cluster. Figure (a) shows the result for $k = 3$ and figure (b) for $k = 5$.

**Percentage of SIFT matches**

OpenCV also has the built-in functionality to calculate SIFT descriptors of an image. When two images are provided, these descriptors can be calculated from every image and then matched to each other. Figure 22 depicts the SIFT descriptors that were calculated from two images containing the same kind of screw. These descriptors are represented by a circle. If the circle is green, there is a match with the other picture. If it is blue, the descriptor does not have a match in the other picture. Matches are represented by a drawn line.

Figure 22 shows that a lot of matches are found between the two images. This is a desired result since the two images contain the same object. Figure 23 shows that also a lot of matches are found between two different
objects as well though. If a high percentage of the descriptors in the query image can be matched to the train images, it can be assumed that both images contain the same object.

Figure 20: Test object in front of a green background to calculate the 3 most present colours of

(a)

(b)

Figure 21: Example of most present colours calculation using clustering with 3 and 5 clusters

Figure 22: SIFT-matching between two images containing the same object
2.2.1.1.2. Results

All object specific recognition methods are tested using 19 Meccano components. Twenty pictures of each component are analysed using the programmed feature extractors. In each picture, the probability that a certain component is displayed on it is calculated for every single one of the components. Therefore, a list of 19 recognition percentages is created for every picture. For a group of 20 pictures of a certain component, the average of all percentages in the 20 resulting lists is calculated. This results in one list of 19 percentages again. This is done for each of the 19 groups containing 20 pictures, each representing one of the 19 components. The list of the 19 parts then each time gets sorted according to the recognition percentage. The average position of the actual component that was displayed on the picture is used as a measure of accuracy.

**Shapes per unit area**

The average position of the correct object was 8.1 in the list of 19 objects. The programmed descriptor therefore does not seem to be useful. The main problem is that it is hard to detect shapes when lighting conditions change and when there is a lot of overlap between the objects. Shapes are also not always detected correctly.

**Edge per unit area**

The average position of the correct object was 3.6. Although it is definitely not enough to guess which product is depicted on the picture, it seems like useful information is extracted that can be used together with other information in the object recognition process.

**Object colour**

This feature extractor seems promising since the colour was correctly detected every single time. The best matching object to a picture was every time either the correct object or another object that has the same colour.
Percentage of SIFT matches
Since some objects with very simple shapes did not have any SIFT descriptors, the results are very poor and no useful result could be obtained from the test.

2.2.1.2. Bag of features
The bag of features method for object recognition was implemented using MATLAB. For training, 17 images were provided of each of the 26 Meccano parts. MATLAB used 5 of these images for training and 12 for testing. The overall accuracy of the created bag of features with SURF-descriptors was 66%.

2.2.1.3. Convolutional Neural Network

2.2.1.3.1. Description of the tests
In part 2.1.2.2.3., some parameters used in the context of image recognition using CNNs were introduced. In this part, the effect of these parameters on the recognition rate is researched using a dataset of 520, to the network unseen, images, 20 in each of the 26 classes. Each class represents one Meccano part. Figure 11 displays 3 out of the 20 images in one of the 26 classes.

All tests are done using finetuning, based on the VGG-16 model. Finetuning is described in the part 2.1.2.2.3. This model was trained based on a subset of the ImageNet database for the yearly ImageNet Large Scale Visual Recognition Competition (ILSVRC). Table 2 shows the layer structure of the VGG-16 CNN that is used for the tests. Also, all pre-trained weights are used for each layer. Only the fully connected classification layers are omitted when loading the network since these are completely dependent on the specific classes that were used during the training. This fully connected network is replaced by the newly created network displayed in table 3. The weights of this network are randomly initialised. The first two layers in this network flatten the output of the last layer of the feature extraction part of the CNN. Flattening means that the multidimensional features are converted to a 1D structure. The following dense layers form the actual fully connected neural network used for classification based on the detected features. A dropout layer is included to prevent overfitting.

The fully connected network always needs training based on the available dataset since the weights were initialised randomly. The filter layers copied from the VGG-16 network do not have to be trained since they were.
already trained with the subset of the ImageNet dataset. Finetuning these weights can possibly be beneficial though.

**Table 2: Layer structure of VGG-16 CNN**

<table>
<thead>
<tr>
<th>Block</th>
<th>Kind of layer</th>
<th>Output size (for input image height and width = 64 pixels, number of channels = 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Input</td>
<td>(64,64,3)</td>
</tr>
<tr>
<td></td>
<td>Convolutional (ReLU)</td>
<td>(64,64,64)</td>
</tr>
<tr>
<td></td>
<td>Convolutional (ReLU)</td>
<td>(64,64,64)</td>
</tr>
<tr>
<td></td>
<td>Max pooling</td>
<td>(32,32,64)</td>
</tr>
<tr>
<td>2</td>
<td>Convolutional (ReLU)</td>
<td>(32,32,128)</td>
</tr>
<tr>
<td></td>
<td>Convolutional (ReLU)</td>
<td>(32,32,128)</td>
</tr>
<tr>
<td></td>
<td>Max pooling</td>
<td>(16,16,128)</td>
</tr>
<tr>
<td>3</td>
<td>Convolutional (ReLU)</td>
<td>(16,16,256)</td>
</tr>
<tr>
<td></td>
<td>Convolutional (ReLU)</td>
<td>(16,16,256)</td>
</tr>
<tr>
<td></td>
<td>Convolutional (ReLU)</td>
<td>(16,16,256)</td>
</tr>
<tr>
<td></td>
<td>Max pooling</td>
<td>(8,8,256)</td>
</tr>
<tr>
<td>4</td>
<td>Convolutional (ReLU)</td>
<td>(8,8,512)</td>
</tr>
<tr>
<td></td>
<td>Convolutional (ReLU)</td>
<td>(8,8,512)</td>
</tr>
<tr>
<td></td>
<td>Convolutional (ReLU)</td>
<td>(8,8,512)</td>
</tr>
<tr>
<td></td>
<td>Max pooling</td>
<td>(4,4,512)</td>
</tr>
<tr>
<td>5</td>
<td>Convolutional (ReLU)</td>
<td>(4,4,512)</td>
</tr>
<tr>
<td></td>
<td>Convolutional (ReLU)</td>
<td>(4,4,512)</td>
</tr>
<tr>
<td></td>
<td>Convolutional (ReLU)</td>
<td>(4,4,512)</td>
</tr>
<tr>
<td></td>
<td>Max pooling</td>
<td>(2,2,512)</td>
</tr>
</tbody>
</table>

**Table 3: Layer structure of fully connected network added to VGG-16 CNN**

<table>
<thead>
<tr>
<th>Kind of layer</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Flatten</td>
<td></td>
</tr>
<tr>
<td>Flatten</td>
<td></td>
</tr>
<tr>
<td>Dense (ReLU)</td>
<td></td>
</tr>
<tr>
<td>Dropout</td>
<td></td>
</tr>
<tr>
<td>Dense (ReLU)</td>
<td></td>
</tr>
<tr>
<td>Dense (softmax)</td>
<td></td>
</tr>
</tbody>
</table>

Except for the varying parameter in each particular test, the parameters in table 4 and table 5 are used during the training of the CNNs.
Table 4: Standard network parameters used

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used pre-trained CNN</td>
<td>VGG-16</td>
</tr>
<tr>
<td>Image width</td>
<td>64 pixels</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.001</td>
</tr>
<tr>
<td>Momentum</td>
<td>0.9</td>
</tr>
<tr>
<td>Batch size</td>
<td>32 images</td>
</tr>
<tr>
<td>Trainable layers</td>
<td>Last 10 layers</td>
</tr>
<tr>
<td>Size of layers in fully connected network</td>
<td>2048 neurons</td>
</tr>
<tr>
<td>Dropout rate</td>
<td>50%</td>
</tr>
</tbody>
</table>

Table 5: Standard data augmentation parameters used

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotation range</td>
<td>40°</td>
</tr>
<tr>
<td>Shift range (vertical and horizontal)</td>
<td>10%</td>
</tr>
<tr>
<td>Shear range</td>
<td>1°</td>
</tr>
<tr>
<td>Zoom range</td>
<td>20%</td>
</tr>
<tr>
<td>Flip (vertical and horizontal)</td>
<td>True</td>
</tr>
<tr>
<td>Number of epochs</td>
<td>75 epochs</td>
</tr>
</tbody>
</table>

2.2.1.3.2. Results

For each of the varied parameters, an accuracy and elapsed time plot have been created. The accuracy depicts the percentage of right guesses on the dataset of 520 unseen labelled images. All of these images are fed through the network that has been trained with the particular parameters. For each of the images, the class with the highest match percentage is compared to the label. If these two match, then it is considered a correct guess. Otherwise the guess is wrong.

Sometimes the progress of the accuracy during training is also shown, for example in figure 27. The blue curve shows each time the accuracy when classifying images that were used while training the network. The orange curve shows the accuracy when classifying images that are unseen by the network.

Since training these networks requires a lot of time, only one network was trained for each parameter value. Therefore, only general trends will be analysed and not a change in accuracy for a specific value of a parameter since this can be a coincidence.

All results are obtained using the Python library Keras running on a Google Tensorflow background. The CPU used for the training is an Intel® Core™ i7-4712MQ.

Network and training parameters

- Image size

The first layer of the CNN has a fixed size. Before being fed into the network, the image must be resized to match the input size of the network. Usually square dimensions are chosen between 32x32 and 256x256 pixels. While a 32 by 32 pixels image is enough to recognise simple shapes like letters and digits, a higher quality image must be used to recognise more complex objects like specific animals.

In this test, four networks were trained to accept square images with a size of 48, 64, 128 and 264 pixels. Figure 11 shows that for our purpose, the extra detail obtained by increasing the quality of the image does help to obtain a higher accuracy. While the improvement in accuracy obtained by increasing the quality of the image
gets smaller for larger images, the training time knows a quadratic growth. Therefore, an optimum between accuracy and training time must be chosen.

![Graph showing accuracy and elapsed time plot for CNN training with varying input image size](image)

**Figure 25: Accuracy and elapsed time plot for CNN training with varying input image size**

Since the accuracy of 87.7%, that was accomplished with the standard parameters as shown in table 4 and an image size of 256 pixels, was the best accuracy over all tests, this resulting network is used in the proof-of-concept. With an ideal inductive sensor, this recognition percentage would even rise to 89.7%.

- **Learning rate**

The parameter learning rate is explained in part 2.1.2.2.3. The learning rate must be tweaked to make the network converge in an optimal way. The training was done with a learning rate of 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001. The accuracy graph shows that a learning rate of 0.0001 or 0.0005 delivers the best results. Values of 0.005 or higher make it impossible for the network to converge.

![Graph showing accuracy and elapsed time plot for CNN training with varying learning rate](image)

**Figure 26: Accuracy and elapsed time plot for CNN training with varying learning rate (logarithmic scale)**

According to the accuracy plot in **figure 26(a)**, training with a learning rate of 0.0001 accomplishes the same result as training with a learning rate of 0.0005. There can be found a difference in the way the result is accomplished though. In **figure 27**, it is visible that a faster conversion is achieved with a higher learning rate. This is normal since a higher learning rate allows bigger changes of the weights in the network. This way the weights converge faster to the optimal result. Although **figure 26(b)** shows that learning rate only has a minimal
effect on the time to train one epoch, the number of epochs needed to reach a certain accuracy is affected by the learning rate.

![Model accuracy (learning rate = 0.0001)](image)

![Model accuracy (learning rate = 0.0005)](image)

Figure 27: Progress of accuracy during training using learning rate of 0.0001 (left) and 0.0005 (right)

A learning rate that is too small causes the network to converge slower. Even worse is choosing a learning rate that is too big since then no convergence is reached. This is caused by changes in weights that are too big. When one step an optimum is reached, the change in the next step is so big that the weight is far away again from being optimal. This way the network endlessly keeps adjusting the weights without any improvements in accuracy. This can be seen in figure 28.

![Model accuracy (learning rate = 0.1)](image)

Figure 28: Progress of accuracy during training using learning rate of 0.1

- **Momentum**
  The parameter momentum is explained in part 2.1.2.2.3. Momentum is needed in order to converge to the global minimum instead of only to a local minimum. Without any momentum, all weights will converge to a nearby minimum. Using a momentum term, the learning will be more consistent and usually towards a better optimum. Adding a momentum improves the accuracy without increasing the training time per epoch. Therefore, adding a momentum term is recommended.
Figure 29: Accuracy and elapsed time plot for CNN training with varying momentum

Figure 30 shows the progress of the accuracy during training using three different values for the momentum. Plot (a) shows the accuracy progress when training is performed without any momentum. All weights converge rather fast to a nearby local minimum. After the initial converging the accuracy still fluctuates, but shows no upwards trend anymore. Plot (b) shows the progress of the accuracy during training when making use of a momentum of 0.5. The training is more consistently converging towards an optimum and the accuracy increases towards the optimal solution. Plot (c) shows the accuracy for a momentum of 1.0. Because so much momentum is added, the weights can ‘escape’ from every minimum and keep changing without converging to a certain optimum. The network is not converging to a solution.

- Batch Size
As explained in part 2.1.2.2.3., batch size is the number of pictures that are propagated through the network at once before calculating the loss and updating the weights. A network was trained with a batch size of 1, 2, 4, 8, 16, 32 and 64 images. The number of pictures used per epoch is the same in each test. The accuracy plot in Figure 31(a) shows that training with a batch size of 32 images resulted in the best accuracy.

A very small batch size of one or two images results in a very low accuracy. The reason is that the loss calculation and weight adjustment are each time based on the information of one image passing through the network. This image is not a good representation for the whole dataset, which causes a bad representation of the gradient in equation 2. When a wrong gradient is used, the weights get adjusted in a non-ideal way.

It can also be noticed that the accuracy using 64 images per batch is lower than when using 32 images per batch. The cause is that in case of a batch size of 64 images, the weights only get adjusted half as many times as in the case of a batch size of 32 images. This causes the network to converge slower. Figure 32 shows that after 75 epochs, the accuracy for the model that was trained with a batch size of 64 is not flattening out as much yet as the accuracy for the model that was trained with a batch size of 32. The model trained using a batch size of 64 images has more potential left to improve than the model trained with a batch size of 32 images.
Figure 30: Progress of accuracy during training using momentum of 0.0, 0.5 and 1.0

The second plot in Figure 31 shows that the elapsed time in function of the batch size is hyperbolic. It makes sense that the training time is larger for smaller batch sizes. Every time a batch is propagated through the network, the loss gets calculated and the weights are adjusted. The smaller the batch size, the more times the weight changes need to be calculated and executed per epoch. Two actions are done during the elapsed time represented in the second plot. Firstly, the images get propagated through the network. Secondly, the loss function gets calculated and the weights are adjusted after finishing each batch. Propagating the images through the network must be done the same number of times, no matter what the batch size is. This time does therefore not depend on the batch size. The time needed for calculating the loss and adjusting the weights does depend on the batch size and is inversely proportional to it. When the batch size is twice as large, the amount of times that the second action has to be performed is halved. Therefore, the elapsed time is expected to be expressed by

\[
\text{elapsed time} = \frac{1}{\text{time to calculate loss and adjust weights}} + \text{time to classify images}.
\]
• **Amount of epochs**

In one epoch, every training image is propagated through the network. To acquire a well-trained network, multiple epochs are necessary. In this test, a network will be trained during 10, 30, 50, 75, 100, 125, 150 and 200 epochs.

The first plot in figure 33 shows that the amount of performed training epochs has a significant effect on the accuracy. The general trend is that an increase of training epochs causes an increase in accuracy. Figure 33(b) shows that this is at the cost of a linearly rising training time. Therefore, an optimum between accuracy and training time has to be found.
Figure 33: Accuracy and elapsed time plot for CNN training with varying amount of epochs

- Amount of trained layers

The used CNN consists of pre-trained filters copied from the VGG-116 network and a randomly initialised fully connected network to classify the images into classes based on the output of the filters. The filter layers copied from the VGG-16 network have already been trained, but using another dataset with different classes than the Meccano dataset. If the filters satisfy the needs and provide an output that enables the fully connected network to classify the Meccano parts, then no retraining of the filter layers is necessary. This is called transfer learning. The last filter layers can represent high-level features though like paws, beaks, wheels, etc. When the objects to detect are not matching the ones used for training, then finetuning some of the last layers can be beneficial for the accuracy.

The influence of the number of trained layers was tested by training none of the filter layers and by training the last 4, 8, 12, 15 and 19 filter layers of the VGG-16 network. These specific numbers of layers were chosen to match the blocks of the CNN displayed in table 2. The resulting accuracies and training times are graphically displayed in figure 34.

Figure 34(a) shows that finetuning the last 8 filter layers gives the best results. This means that the last two filter blocks contained high level features that were too specifically adjusted to the subset of the ImageNet dataset. Some finetuning based on the Meccano dataset of these layers tweaks these layers to create filters that extract high-level features that match the Meccano components. The first three blocks apparently contain filters that are general enough. These possibly detect colour blobs, edges, curves...

The second plot in figure 34 shows that finetuning more layers increases the training time. Therefore, it is important to not finetune too many layers since this decreases the accuracy and increases the training time.
Figure 34: Accuracy and elapsed time plot for CNN training with varying amount of trainable layers

- **Size of fully connected layers**

  The fully connected network is stacked behind the filter layers from the VGG-16 model. It classifies the images based on the output of the filters. The last layer of this subnetwork is the output layer and contains as many neurons as there are classes. Each neuron represents a class and will produce the probability that the input image contains the object represented by this class. The size of the other layers is free to choose. In this test, a network with fully connected layer sizes of 32, 64, 128, 256, 1024, 2048, 4096 and 8192 neurons will be trained. The plots displaying the resulting accuracy and training time for each of these sizes are displayed in Figure 35. Plot (a) shows that the accuracy is best for a size of 2048 neurons per fully connected layer. For higher sizes, the accuracy decreases and the training time increases.

- **Dropout rate**

  Since the Meccano dataset is very small and data augmentation is implemented, the same image, or a data augmented variety of it, is fed multiple times into the network per epoch. This may encourage overfitting of the CNN. Overfitting means that the network scores a lot better on classifying seen images than on classifying unseen data. The network is adapting too much to the specific training images, causing a lack of generalisation. Dropout is a technique for addressing this problem. During the training, neurons and their connections are randomly

![Figure 35: Accuracy and elapsed time plot for CNN training with varying size of fully connected layers (logarithmic scale)](image)
removed from the network [24]. This is graphically depicted in Figure 37. This prevents these neurons from adapting too much to the dataset.

Figure 36 shows the accuracy and training time for a network trained with a dropout rate of 0.0 to 0.9 in steps of 0.1. The first plot shows that adding dropout can only slightly improve the accuracy of the network. The first reason that the improvement is only small, is that the difference between the train and the validation accuracy is not big. This is shown on Figure 38(a), which depicts training without any dropout. The second plot, with a dropout of 0.7, shows that the difference between the training and validation accuracy has indeed decreased. But since about 70% of the neurons in the dropout layer are deactivated during training, the network will converge slower. Since both networks are trained with the same number of epochs, the network that makes use of dropout will not reach the same accuracy. It is indeed clear in Figure 38 that there is more improvement opportunity left for the network with a dropout rate of 0.7 than for the network without any dropout. This is the second reason that adding a dropout layer did not show significant improvements.

The graph in Figure 36(b) shows that the training time per epoch is equal for any dropout rate, but as stated before higher dropout rates require more epochs to converge and will therefore have to train longer.

Figure 36: Accuracy and elapsed time plot for CNN training with varying dropout rate

Figure 37: Graphical representation of dropout [24]
Figure 38: Progress of accuracy during training using dropout rate of 0.0 and 0.7

Data augmentation parameters

- Rotation range

Before propagating the image through the network, the image is rotated over a random angle in the range of $-\alpha^\circ$ to $+\alpha^\circ$. The value $\alpha$ is a data augmentation parameter that can freely be chosen between 0° and 360°. Figure 39 shows the accuracy and training time for networks trained with different rotation ranges applied to the input images. The used rotation ranges during the test are 0, 10, 20, 40, 60, 100, 140 and 180 degrees.

The accuracy plot in figure 39(a) shows that although performing rotation can improve the results, the improvement is neither big nor guaranteed. Rotating the input images does not or only slightly affect the training time.

Figure 39: Accuracy and elapsed time plot for CNN training with varying rotation range
• Shift range
This data augmentation methods shifts the image before feeding it through the network. The shift range can be freely chosen between 0 and 100%. Figure 40 shows the accuracy and training time for an applied shift range of 0, 10, 20, 30, 50, 80 and 100%. The vertical and horizontal shift range was always chosen to be equal. The accuracy plot in Figure 40 does not show very significant improvements when shift is applied to the input images. For some ranges, the accuracy decreases. The training time plot shows that shifting the images does increase the training time significantly.

Figure 40: Accuracy and elapsed time plot for CNN training with varying shift range

• Shear range
Shear skews an image at a random angle within the given range. The shear range can be chosen between 0° and 360°. Figure 41 shows the accuracy and training time for an applied shear within the ranges of 0°, 30°, 60°, 90°, 180°, 270° and 360°. The accuracy does not show a certain trend and a conclusion about the effect on accuracy by applying a shear cannot be made. Shear does not affect the training time of the network.

Figure 41: Accuracy and elapsed time plot for CNN training with varying shear range

• Zoom range
Zoom extracts a certain part of the image with a random zoom within the given zoom range. The horizontal and vertical zoom are not always chosen equally. This way the object in the data augmented object might look smaller
or wider than in the original image. The accuracy and training time of the neural network were measured for zoom ranges of 0, 0.05, 0.1, 0.15, 0.2, 0.3, 0.5, 0.8 and 1. Figure 42 (a) shows that zoom range has a positive effect on the accuracy of the trained CNN up to a value of 0.3. When a bigger value for the zoom range is chosen, the accuracy decreases. Apart from an unexplainable spike in training time for a zoom range of 0.05, applying zoom as data augmentation does not influence training time a lot.

Figure 42: Accuracy and elapsed time plot for CNN training with varying zoom range

- **Flip**

When both horizontal and vertical flip are activated, this data augmentation method mirrors the image at a random rate around the horizontal and vertical axis. The accuracy of the neural network was once tested without any flips and once with both horizontal and vertical flips. Figure 43 shows that performing random flips on the input data improved the accuracy significantly with no effect on the training time.

Figure 43: Accuracy and elapsed time plot for CNN training with and without flipping images randomly
• General overview on effect of data augmentation parameters

The accuracy plot in [figure 39(a)] up to [figure 43(a)] shows that performing a data augmentation technique on the training images does not guarantee an improvement in accuracy. To further test the effect of data augmentation, six more networks are trained. The first network is trained according to the parameters as shown in table 4 and table 5 and therefore all data augmentation techniques are applied. The second network uses the same parameters, but no rotation is applied to the input images. The third network does neither apply rotation nor shift. The fourth network does not apply shear as well. The fifth network leaves the use of zoom behind. The sixth network does not flip its images on top of all this and does therefore not apply any data augmentation anymore. [Figure 44] shows how the accuracy evolves when removing the data augmentation methods step by step in the same order as described earlier in this paragraph. Mainly not applying a shift to the input images results in a significantly smaller accuracy. Overall the data augmentation seems to improve the accuracy, but this graph also shows that not every single data augmentation method always introduces an improvement in accuracy.

![Accuracy (gradually removing data augmentation)](image)

[Figure 44: Accuracy after gradually removing data augmentation]

[Figure 45] shows the progress of accuracy during training without (a) and with (b) applying data augmentation to the input images. Although the accomplished accuracy on the test dataset is similar in both cases, this is not the case for the training data accuracy. When no data augmentation is used, the gap between the training data accuracy and the test data accuracy is a lot bigger. This means that the network is overfitting: although the network recognises the objects on training images well, this is not the case for unseen images. Since recognising objects on unseen images is the goal, this is not a good situation.

It can be concluded that, because data augmentation makes sure that every input image is unique, data augmentation prevents the weights in the network from adapting too much to the training data. Therefore, the effect of overfitting can be avoided or diminished. Data augmentation is not always beneficial for the accuracy of the network.
Figure 45: Progress of accuracy during training without and with data augmentation.
3   Practical realisation

As a proof of concept, the researched theoretical concepts are implemented into a basic smart pick-to-light system. Although only being a concept model, it is fully functional. It is built around the earlier described Meccano assembly case.

Figure 46 shows the implementation of both the hardware and software in the configuration of the proof of concept. Instead of implementing the master software and the database in a cloud service, it is running on a computer. The master software does not need any user input or graphical user interface. The master software controls the whole system. It contains the neural networks to perform image recognition, manages the database and makes sure that the light on the correct module is switched on. Either another or the same pc is used as a device from which the picking process can be controlled by running the client software on it. This software connects to the master software. Using this software, messages can be sent to the master software to let it perform actions like requesting pictures from the modules, adding products to the database and adding a picking order to the queue. Each pick-to-light module runs the module software. Every module connects to the master software as well. The main purpose of this software is taking pictures and sending these to the master when required. It also controls the light on the module and informs the master when the button is pressed.

3.1. Hardware

Mainly the theoretical aspect of object recognition is researched in this master’s thesis. The hardware was chosen initially for the ease of use. It is not the most suitable hardware for this application, but it made the creation of a proof of concept model easier. In a later stage, research can be done to find the most efficient and suitable hardware.
The master software and the MySQL database are running on a pc. During programming, the later implementation of the software into a cloud service was taken into account.

The module software was implemented on a Raspberry Pi 3 Model B. It is equipped with a Pi Camera v2, a button and an LED. All these components were fixed to a picking bin to create a concept model of a pick-to-light module within the smart system [Figure 47]. The inside of the bin is covered in green paper to give an optimal contrast with the products that it contains and allow easier object recognition.

![Figure 47: Concept of a smart pick-to-light module](image)

The client software, where the picking process can be controlled, is also running on a pc. Since this is a simple piece of software that only sends TCP/IP messages, the process can be controlled by devices ranging from smartphones to servers. All communication is done over WIFI.

### 3.2. Software

For the master, the modules and the client, VB.NET is chosen as the programming language. The module is also partly programmed in Python since the libraries Keras, to perform object recognition with neural networks, RPi.GPIO, to access the GPIO of the Raspberry Pi and PiCamera, to take pictures with the Raspberry Pi, are unavailable for VB.NET. These Python programs get called in a command prompt by the VB.NET software.
3.2.1. Database

All necessary data for controlling the smart pick-to-light system are stored in a MySQL database. Also, extra information about previous performed picks is stored. These data can for example be used to analyse the efficiency of the system or for troubleshooting later on.

The database consists of 10 tables, which are displayed in the database model depicted on figure 48. The table Products contains all necessary information to recognise the products in the pick-to-light modules. The field prName contains a recognisable name for the product in the client software. The path given to a picture of the product in prPicturePath is used to visually show a certain product. This way an operator can easily decide based on the taken picture whether the recognition software made the right decision. The path to a folder with pictures of this product stored in prProductFolder is used for training the neural network. Pictures can be added in this folder to later on retrain a network with these added pictures. This way the system continuously learns from the input.

PickingList is a junction table between PickingOrders and Products. It contains links to products that have to be picked in a certain picking order. The order in which the products are picked in a product order is determined by prVolgorde. The field prQuantity shows the quantity of the product that has to be picked in a certain step. A picking list can be considered as a general recipe for the picks that need to happen to gather everything for a certain picking order.

The PickingOrders table contains all different picking sequences that can be performed. These can be added to the Queue table. Picking orders that are added to the queue will be executed one by one.

The PickingListQueue refers to an entry in the PickingList. The reason is that the entry in the PickingListQueue table is unique. It contains information about when the pick started and whether it is busy or not.

The PickingLog table stores information about every executed pick. Apart from the picking order and the picked product, it also contains information about the duration of the pick and the identity of the picker.

When a module connects to the master, it is added to the ActiveModules table. A connected module sends a message to the master every five seconds to confirm it is still online and connected. The master then sets the appropriate boolean in the column amActive to true. If a module has not sent anything in the last five seconds, this Boolean is set to false. Another important entry is the foreign key that links to a product. This is the product that the specific module contains. This link is adjusted after the automatic product recognition process.

The tables RecognitionResult and ProductRecognition are used to store the output of the neural network after performing object recognition on all received pictures from the modules. Thanks to this output, assigning the correct products to the correct modules happens easy and fast.
Figure 48: Database schema used for proof of concept
3.2.2. Master

The master is the heart of this proof of concept since both the modules and the clients connect to it. It handles the received messages as displayed in table 6. The master also sends the messages displayed in table 7 to the appropriate modules.

<table>
<thead>
<tr>
<th>Source of message</th>
<th>Message</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>Heartbeat</td>
<td>This message is sent by each module every 5 seconds. The master sets the sending module to active in the database. The message also contains the current battery level of the module. This is also adjusted in the database. When no heartbeat message has been received from a module for more than 5 seconds, then this module will be considered as not being active anymore until another message is received.</td>
</tr>
<tr>
<td></td>
<td>Pick acknowledge</td>
<td>When a required product is picked, this is acknowledged by pressing the button on the module. When this happens, the module sends an acknowledge to the master. This way the master knows that the product was successfully picked. The master will request this module to switch off its light and will request another active module to switch on its light.</td>
</tr>
<tr>
<td></td>
<td>Image</td>
<td>This message contains a picture taken by a module. The picture is used to determine which components are sorted into which modules.</td>
</tr>
<tr>
<td>Client</td>
<td>Link product to module</td>
<td>After the recognition percentages are calculated for each module, these are written to the database by the master. Based on this information, the operator can easily link products to the correct modules. When he does, the actual link in the database is made by the master upon receiving this message that contains the IP-address of the module and the name of the product to link.</td>
</tr>
<tr>
<td></td>
<td>Request pictures</td>
<td>Before starting an order, a request can be made by the client to let all modules send a picture to the master. The client only sends this request to the master. As a reaction the master passes on this request to all active modules. When all images are received, the object recognition will be executed.</td>
</tr>
<tr>
<td></td>
<td>Configuration accepted</td>
<td>Once all products are linked to the correct modules, this message can be sent from the client. The master will start the next order in the queue.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light on request</td>
<td>Request a module to switch its light on</td>
</tr>
<tr>
<td>Light off request</td>
<td>Request a module to switch its light off</td>
</tr>
<tr>
<td>Request picture</td>
<td>Request a module to take and send a picture to the master</td>
</tr>
</tbody>
</table>
3.2.3. Modules

In this proof of concept, the modules are slaves and perform actions upon request. Only the heartbeat messages are sent on own initiative. A module only communicates with the master. Therefore, all messages sent and received by the modules can be found in table 6 and table 7. The modules have no graphical user interface since its IO, an LED and a button, are available as hardware on the module.

3.2.4. Client

The client is a program that is used by the operator. Although the picking process can be controlled using this program, all functions are actually executed by the master. The client merely requests the master to do them using TCP/IP messages. Since the software is this simple, an implementation for other devices like tablets can easily be made. Since the client only sends messages to the master, all messages sent can be found in table 6.

To explain the functions that are built-in into the client software, an example case where a new product needs to be assembled is presented. Firstly, an Excel sheet must be created containing the order in which the components need to be picked and the amount needed with each pick. Figure 49 shows how Excel sheets can be imported in the client software. The client software writes all products and picking lists to the database.

![Figure 49: Using the client software to add a picking list and a component range](image)

After adding the picking order and all its components to the database, the picking order can be added to the queue [Figure 50]. The queue is a table in the database that contains all orders that need to be picked. They will be executed one by one.
If nothing was changed in the system, the picking order can be started now. But if some components were added, removed or switched between bins, then automatic recognition should be performed before starting the order. To do this, the button "Request pictures" must be clicked (Figure 51). This will notify the master that automatic recognition is required. The master will handle this by requesting all pick-to-light modules to take a picture and send it to the master. When all pictures have arrived, the neural network is loaded and the object recognition is performed. The results of the object recognition are written to the database.
These results are fetched from the database and displayed in the client software [Figure 52]. In the left list, the ip address of each module is shown. When a certain module is selected, also the picture that was sent by this module is displayed on the left. On the right, the recognition percentage is shown for every product. The operator can select the correct product, guided by both the percentages and the displayed pictures. When an actual implementation would be done, it would be more useful to automatically assign the products that have high recognition percentages. This software does not do that since it was written for demonstration purposes.

Figure 52: Using the client software to link a component to a module

Once all needed products have correctly been assigned to a pick-to-light module, the next order in the queue can be started [Figure 53]. The picking sequence is started and the light on the pick-to-light module that contains the first needed product will switch on.

3.3. Cloud computing in the smart pick-to-light system

Based on the described advantages and disadvantages of cloud computing, it is possible to make a choice for the smart pick-to-light application on either implementing this or not implementing it. The first advantage of cloud computing is that no decisions must be made on how much resources the servers have to contain. In traditional computing, the servers must be bought before the development of the service starts to be able to do testing during the development. But before the start of the development, it is still unsure how much data will have to be transmitted through this server. For example, it is not exactly sure yet how much information will be needed for the object recognition. It is also possible that the warehouse will expand later on, resulting in a significant increase of requests because of the increase in pick-to-light modules. Because the computing resources obtained from the cloud are freely scalable, using the cloud will be more economical and convenient than using a traditional local server.
The second advantage of cloud computing is that the installation and maintenance of the cloud is done by external experts. This saves a lot of development time, although none of this extra time spent on getting the servers up and running would add any value to the system.

Figure 53: Using the client software to start the next order in the queue
4 Conclusion

Although great efforts have been made already over time to improve the efficiency of picking processes, a lot of progress is still needed to create an extremely flexible and efficient system. This master’s thesis takes a first leap to create this smart pick-to-light solution.

A traditional pick-to-light system contains a master that addresses its pick-to-light modules with a fixed address. Usually these are all physically connected to the master. When a module is added or removed, or when products are moved between the pick-to-light modules, then reconfiguration needs to be done. The address of a module needs to be added, removed or linked to the correct product.

In the case of a smart system, no reconfiguration is needed. The smart modules can recognise the products using their built-in sensors. To create an extremely flexible system, the master is implemented in an external cloud service and the modules are connected wirelessly.

To implement this smart system, recognising objects is one of the most important prerequisites. Tests on a set of 26 Meccano parts show that the combination of a camera and an inductive sensor is effective for object recognition. Using a convolutional neural network with 20 training images per class, an object recognition accuracy of 87.7% was reached purely by analysing the taken images. When an ideal inductive sensor is included, an accuracy of 89.7% is reached.

To get results of this magnitude with only 20 training images per class using a convolutional neural network, some specific techniques need be applied during the training of the network. The first important technique is finetuning: instead of initialising the network’s weights to a random value, pre-trained values were used. These values are adjusted using the small training dataset. Finetuning is absolutely necessary to create an accurate network using a small dataset. Secondly, artificial data were created using data augmentation. Although this technique does not always prove to have a positive effect on the accuracy, it does prevent overfitting.

Also, research was done to determine whether cloud computing is suitable for this application. It shows that cloud computing can add to the flexibility of the system by letting any kind of device connect on it from anywhere. Apart from offering flexibility, the cloud also offers the advantage that no initial investment has to be done in a server. Because its resources are scalable, no extra investments have to be done when the needs of the application grow.

Using the knowledge gained from the research, a proof of concept is created. The resulting system does an accurate recognition of the objects. Modules can easily be added or removed without any reconfiguration. Although the master program and the database were not implemented in the cloud, the later implementation was taken into account during programming. The client software allows easily adding new jobs to the picking queue and reading information about the ongoing process. New picking orders and products can be easily imported from Microsoft Excel.

Although the created system does have many features that comply with the described smart pick-to-light system, there is still work required to create the wished ultimate flexibility and user-friendliness. Firstly, research needs to be done to select the best battery and wireless technology to make all modules completely wireless. Secondly, the sensors need improvement. The camera should have auto-focus to adjust its focus according to the filling grade of the bins. The inductive sensor needs to be implemented. Thirdly, both the master software and the MySQL database need to be moved to the cloud to make use of its advantages. Fourthly, more intelligence should be available in the modules instead of in the master. Although the current system acts smart, the modules are only slaves to the master. Ideally, the master in the cloud only gathers information and acts as an access point for all clients to read this information and to control the system. These commands can then be passed on to the modules, that effectively execute them. Lastly, the client software needs to be expanded. Currently, the object recognition only slightly helps to assign the products to the correct modules by displaying the match percentages.
in a list and showing the picture that was taken. Algorithms must be developed that automatically assign the products that the CNN recognised with large certainty. Only in case of doubt, an operator should be needed.

Overall, first steps towards an industrially usable smart pick-to-light system have successfully been taken. The most important problem of object recognition has been researched and tested into detail and a successful solution has been found. Although the created proof of concept has shortcomings that prevent it from being used in an industrial environment already, it does show the most important basic concepts of the described smart pick-to-light system.
5 Bibliography


