Independent project (degree project), 15 credits, for the degree of Master of Science in Computer Science with Specialization in Embedded Systems, Spring Semester 2019

Department of Computer Science, Faculty of Natural Sciences, Kristianstad University

Automated Supply-Chain Quality Inspection Using Image Analysis and Machine Learning

Yuehan Zhu
Abstract
An image processing method for automatic quality assurance of Ericsson products is developed. The method consists of taking an image of the product, extract the product labels from the image, OCR the product numbers and make a database lookup to match the mounted product with the customer specification. The engineering innovation of the method developed in this report is that the OCR is performed using machine learning techniques. It is shown that machine learning can produce results that are on par or better than baseline OCR methods. The advantage with a machine learning based approach is that the associated neural network can be trained for the specific input images from the Ericsson factory. Imperfections in the image quality and varying type fonts etc. can be handled by properly training the net, a task that would have been very difficult with legacy OCR algorithms where poor OCR results typically need to be mitigated by improving the input image quality rather than changing the algorithm.

Keywords
Image Analysis, Computer vision, OCR, Machine Learning, Neural Networks, LSTM networks
## Contents

1  Introduction ........................................................................................................... 8  
   1.1  Background .................................................................................................. 8  
1.2  Statement of Problem ...................................................................................... 9  
   1.2.1  Research Questions .............................................................................. 10  
   1.2.2  Methodology ......................................................................................... 10  
1.3  Limitations ..................................................................................................... 11  
2  Literature Study .................................................................................................. 12  
   2.1  Binarization Algorithm ............................................................................... 12  
      2.1.1  General edge detection ..................................................................... 12  
      2.1.2  Threshold detection ........................................................................... 12  
   2.2  Contour detection ....................................................................................... 13  
   2.3  OCR Algorithm .......................................................................................... 13  
      2.3.1  Legacy OCR for reference ................................................................. 13  
      2.3.2  OCR using machine learning ............................................................... 14  
3  Theoretical background ...................................................................................... 14  
   3.1  Edge detection ............................................................................................. 14  
      3.1.1  Canny Edge detector ......................................................................... 14  
      3.1.2  Image Thresholding .......................................................................... 15  
      3.1.3  Contour detection .............................................................................. 17  
   3.2  Tesseract legacy OCR engine ....................................................................... 20  
      3.2.1  Baseline (row) detection .................................................................. 21  
      3.2.2  Character detection .......................................................................... 22  
      3.2.3  Character classification .................................................................... 22  
   3.3  Neural Networks and machine learning ...................................................... 22  
      3.3.1  Supervised and unsupervised machine learning ............................... 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.2</td>
<td>Neural networks</td>
<td>24</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Convolutional neural networks</td>
<td>24</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Recurrent Neural Networks</td>
<td>25</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Long Short-Term Memory networks</td>
<td>25</td>
</tr>
<tr>
<td>3.3.6</td>
<td>Discussion</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>Implementation</td>
<td>28</td>
</tr>
<tr>
<td>4.1</td>
<td>Input data</td>
<td>28</td>
</tr>
<tr>
<td>4.2</td>
<td>Label Extraction Algorithm</td>
<td>29</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Binarization</td>
<td>29</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Contour detection</td>
<td>30</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Contour classification</td>
<td>31</td>
</tr>
<tr>
<td>4.3</td>
<td>OCR</td>
<td>32</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Baseline OCR algorithm</td>
<td>33</td>
</tr>
<tr>
<td>4.3.2</td>
<td>LSTM algorithm</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>Results</td>
<td>36</td>
</tr>
<tr>
<td>5.1</td>
<td>Label extraction</td>
<td>36</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Canny edge detector</td>
<td>36</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Threshold edge detection</td>
<td>41</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Discussion</td>
<td>47</td>
</tr>
<tr>
<td>5.2</td>
<td>OCR</td>
<td>47</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Legacy OCR</td>
<td>48</td>
</tr>
<tr>
<td>5.2.2</td>
<td>LSTM OCR</td>
<td>48</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion and Future Work</td>
<td>49</td>
</tr>
<tr>
<td>6.1</td>
<td>Conclusion</td>
<td>49</td>
</tr>
<tr>
<td>6.2</td>
<td>Future Work</td>
<td>50</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Expand usage to other products</td>
<td>50</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Deploy in real-time environment</td>
<td>50</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Train LSTM OCR with font definition instead of product labels</td>
<td>51</td>
</tr>
<tr>
<td>7</td>
<td>References</td>
<td>52</td>
</tr>
<tr>
<td>8</td>
<td>Appendix A, source code implementation</td>
<td>54</td>
</tr>
<tr>
<td>8.1</td>
<td>Label extraction</td>
<td>54</td>
</tr>
<tr>
<td>8.2</td>
<td>OCR</td>
<td>56</td>
</tr>
<tr>
<td>8.2.1</td>
<td>OCR network specification</td>
<td>58</td>
</tr>
</tbody>
</table>
Table of Figures

Figure 1 Structure of cabinet image ................................................................. 9
Figure 2 Overview of algorithm ..................................................................... 10
Figure 3 Product label extraction .................................................................. 10
Figure 4 Example of 0- and 1-components .................................................... 17
Figure 5 4-connected case .............................................................................. 18
Figure 6 8-connected case .............................................................................. 18
Figure 7 Surroundness among connected components and among borders .... 20
Figure 8 History of tesseract development .................................................... 21
Figure 9 General structure of a neural network .............................................. 24
Figure 10 A recurrent neural network. Unrolled loop to the right. ................. 25
Figure 11 The repeating module in a standard RNN contains a single layer .... 26
Figure 12 The repeating module in an LSTM contains four interacting layers 26
Figure 13 The cell state in an LSTM module .................................................. 27
Figure 14 Example of a sigmoid gate .............................................................. 27
Figure 15 LSTM network used for OCR step ................................................. 28
Figure 16 All shapes found by border following algorithm ......................... 31
Figure 17 Remaining shapes after classification (example) ......................... 32
Figure 18 Example of extracted product label (text blurred for confidentiality reasons) .................................................................................................................. 35
Figure 19 Raw output from the Canny edge detector (example) ................. 37
Figure 20 Output of border following algorithm on the Canny edge image (blue shapes) ........................................................................................................ 38
Figure 21 Dilated edge image ....................................................................... 39
Figure 22 Output of border following algorithm on the dilated Canny edge image (blue shapes) .................................................................................................................. 40
Figure 23 Missed label detections for the Canny edge detection ................. 41
Figure 24 Missed label detections for binary thresholding ............................ 42
Figure 25 Example of binarized cabinet image using Kittler-Illingworth threshold...... 43
Figure 26 Detected shapes with Kittler-Illingworth threshold (example) ....... 44
Figure 27 Example of binarized cabinet image using Otsu threshold ............ 45
Figure 28 Detected shapes with Otsu threshold (example) ............................ 45
Figure 29 Example of binarized cabinet image using tuned optimal simulated threshold (0.8) .......................................................................................................................... 46
Figure 30 Detected shapes with optimal simulated threshold (example) ................... 47
Figure 31 Matlab source code listing for label detection (double click for full listing). 56
Figure 32 Ubuntu 18.04 command line reference for tesseract OCR (double click for full listing) ............................................................................................................. 58

Table of Equations

Equation 1 Intensity Mean Value (foreground).............................................................. 16
Equation 2 Intensity Mean Value (background) ............................................................ 16
Equation 3 Intensity Variance (foreground) ............................................................... 16
Equation 4 Intensity Variance (background) .............................................................. 16
Equation 5 Kittler-Illingworth threshold .................................................................... 17
Equation 6 Otsu threshold ....................................................................................... 17
Equation 7 Neighborhood matrix for image dilation .................................................. 38
Equation 8 VGSL network specification for LSTM based OCR .............................. 58

Table of Tables

Table 1 Legacy OCR algorithm results ........................................................................ 48
Table 2 LSTM OCR algorithm results ....................................................................... 48
1 Introduction

1.1 Background

The problem addressed in this project stems from Ericsson supply chain. For a certain product family, circuit boards and cabinets are being produced in the factory. Each cabinet consists of several slots that can hold one circuit board each.

Final mounting of the product before shipping to customer consists of mounting circuit boards in the slots of the cabinet according to a detailed customer specification. Which boards need to be mounted in which slots varies from customer to customer and from order to order. The desired mounting is stored in a customer order database.

When final mounting of the circuit boards is finished, a quality assurance step is carried out. Here, the staff is scanning each board with a bar code scanner and the result is compared to the customer order database. There are several problems with this method. Firstly, it is error prone, since the boards must be scanned in the correct order and no board can be omitted from scanning. Secondly, it is time consuming and labor intensive.

Therefore, an automated image processing system is proposed to replace the manual quality assurance. In Figure 1, the structure of the cabinet is shown. The basic idea with the automated system is to capture an image of the whole cabinet, extract the white product labels and detect the text printed on the label using OCR. The extracted text, that contains the unique product identifier, can subsequently be compared with the expected value in a customer order data base.
1.2 Statement of Problem

Industrial AI is gaining more and more traction as hardware gets more capable and computational power less expensive. In systems where AI is not yet deployed, heuristic or parametric algorithms with a priori knowledge about the structure of input data are commonly used. These can be effective, but rarely robust to changes of input data characteristics.

In this project, a prototype system for image recognition will be built of which one part will be implemented using machine learning. Specifically, the OCR part where changes in input data is expected will be using ML techniques. The expected advantage is that the algorithm itself will not have to be updated as input data changes. Instead, the same neural network simply needs to be re-trained using new input samples. The expected advantage is that training a neural network with a new set of sample images can be done by a factory worker, whereas a new OCR algorithm for new input data would require engineering effort.
The goal of the project is to achieve an image recognition system based on ML, that can perform equally well as a system built on legacy OCR techniques but with the added advantages that ML can provide. An overview of the proposed algorithm is shown in Figure 2

![Figure 2 Overview of algorithm](image)

The data base contains a finite, and quite small, set of products. This means that the OCR does not necessarily need to be error free. The data base search will be performed as selecting the product in the database with the least hamming distance to the OCR result. It is enough if the resulting data base search produces an error free result.

The extraction of product labels is a high-level task specific to this project and must be further broken down to be analyzed. The idea is to first detect foreground objects in the image by binarization and thereafter make a structural analysis of the resulting edge image to find contours that represent objects. The detected object must as a last step be classified so that only rectangle or square shaped objects are used as objects of interest. In Figure 3, the proposed steps for product label extraction are shown.

![Figure 3 Product label extraction](image)

### 1.2.1 Research Questions

This project needs to answer the following research questions:

1. Image analysis methods like binarization and shape detection are wide adopted in literature. Can such algorithms perform well enough to detect the product labels in this application? What accuracy is possible to obtain?
2. Can ML be used to perform OCR on the product labels? Which character error rate is possible to obtain?

### 1.2.2 Methodology

The following steps are proposed to accomplish the project
1. Apply literature research to find suitable algorithms for product label extraction. Please refer to chapter 2.1.

2. Apply literature research to find a suitable ML method as well as a baseline (traditional) method for OCR for performance comparison. Please refer to chapter 2.3.

3. Obtain an input image set from factory with enough image quality. Please refer to chapter 4.1

4. Implement the chosen algorithms in a prototype system and evaluate the label extraction as well as OCR reading results between ML and traditional OCR method for the obtained input images. Please refer to chapter 4.2 and 4.3.

1.3 Limitations

1. In this project, we assume a high quality of input image. It will not be explored how the algorithm responds to lower image quality, due to e.g. varying light conditions. This can be an interesting area for future work.

2. The characteristics of the product labels is not expected to change during this project. Therefore, it will not be possible to test the robustness of the algorithm to changes in input characteristics.

3. This project aims at finding a proof of concept for the complete system as well as the underlying algorithms using a normal desktop computer. It out of the scope to deploy the solution on dedicated hardware in a real-time environment.
2 Literature Study

The methodology for the literature study has been to define a set of keywords for each area of study and explore the generated search results. The search has been refined based on number of hits and number of citations. Google Scholar [22] has been used as search engine.

2.1 Binarization Algorithm

The binarization of an image aims at converting a grayscale image to binary where objects of interest in the foreground are white and background is black. For image binarization, two approaches have been researched. The first approach is to search for standard, general edge detection techniques, that can be applied to any image. The second approach is to use image thresholding to detect objects of interest.

2.1.1 General edge detection

The edge detection problem has been thoroughly studied in literature for many years. Since the problem of edge detection is well known, the focus of this literature research has been to find reliable sources for comparison of edge detection techniques. The age of the articles has not been judged important in this case, but rather the number of citations is indicative of the quality of the work.

Keywords: image AND edge AND detection AND techniques

The search gave around 100 results with these keywords in the title.

Reference [1] from 2008 is the most referenced with over 900 citations and has been used in this project. The paper is comprehensive and compares gradient based techniques (Sobel, Robert and Prewitt operators) as well as Laplacian of Gaussian operator. These techniques are compared to the well-known Canny edge detection algorithm.

2.1.2 Threshold detection

A separate search has been done on image thresholding techniques.

Keywords used: image AND thresholding AND performance
The search gives approximately 200000 results and needs to be refined. When examining the number of cites, it was quickly found that one article is much more cited than others. Reference [3] has almost 5000 cites which is significantly higher that other search hits which average on about 100-200 cites. We decided to use the results in this article as basis for implementation and test.

2.2 Contour detection

In this project we strive at finding specific contours or shapes rather than edges alone. The edge detection step is the first step to detect the shapes. The following step consists of finding contours defined by the detected edges. Contour detection is basically a topological and structural analysis of an image.

Keywords: image AND topological AND structural AND analysis

The search results are abundant. Most results are highly specialized, poorly cited, and only applicable in a very narrow application domain. The most cited, and most promising result from this search is an algorithm proposed by Suzuki in 1985 and referenced in [4]. It is a general-purpose algorithm that can transform edge images into a topological hierarchy of objects, or contours.

Following contour detection, a contour classification step is needed to separate the wanted contours from unwanted. This step is a straightforward heuristical step to determine if the shape is rectangular with the expected size and position. It is further described in 4.2.3.

2.3 OCR Algorithm

2.3.1 Legacy OCR for reference

The focus of this search is to find open source implementations of OCR engines.

Keywords: open AND source AND ocr

Returns around 74000 results. The article in [14] is the most cited reference. It is an overview of an open source implementation, tesseract.
2.3.2 OCR using machine learning

Using neural networks for OCR is a relatively new field of study. It is difficult to get a comprehensive and cohesive set of references by a single search. The number of citations is also significantly lower than more studied fields like edge detection.

Keywords: neural AND networks AND image AND classification


A narrower search for finding suitable neural networks for ocr was tried

Keywords: neural AND networks AND ocr

The search yields reference [6] and [7] suggesting that LSTM neural networks is a promising approach due to its ability of keeping both long- and short-term memory for a sequence of patterns with inter-dependency. The result is not surprising since OCR line recognition is by nature about detecting a sequence of objects in an image.

To investigate the combination of the two approached, the following search was used

Keywords: convolutional AND “long short-term memory” AND network


3 Theoretical background

3.1 Edge detection

3.1.1 Canny Edge detector

The conclusion from [1] is that the Canny edge detector [2] performs better than all other methods in all studied scenarios, except when the image is noisy. Since we have control
over the image quality in this project, the Canny detector is chosen as candidate for the general edge detection technique.

The Canny detector operates on a grayscale image and uses thresholding of intensity gradients according to the following steps

1. Find the intensity gradient of the image by Sobel operator.
2. Remove thick edges by Non-maximum suppression. The gradient typically increases gradually around an edge. In this step only the maximum gradient pixel is kept as an edge candidate. Surrounding pixels with lower gradients are discarded.
3. Perform canny edge thresholding using low and high gradient threshold
   a. Gradients stronger than high threshold are considered edges
   b. Gradients weaker than low threshold are not considered edges
   c. Gradients between the thresholds are considered edges if the corresponding pixel is directly connected to a strong edge

For a more detailed explanation of the algorithm, please refer to [2].

3.1.2 Image Thresholding

Image thresholding is a general technique aiming at classifying a pixel as either “dark” or “bright”. As opposed to the Canny edge detector, thresholding is applied directly on the grayscale intensity value, rather than the gradient.

In an image with distinct dark or bright objects, this can be an attractive first step to object recognition. Visually, the product labels in our images appear much brighter than the background cabinet, which makes this approach promising.

Image thresholding is expected to have two advantages over the Canny detector. Firstly, if the hypothesis that labels can be detected using thresholding is correct, they may prove to be better performing for our purpose. Secondly, thresholding techniques are computationally less expensive which makes them attractive for deployment in a real-time environment.
Reference [3] contains a ranked list of thresholds based on results in general images. Common for all thresholds is that they are calculated based on statistical properties in the image. The best performing threshold in this evaluation is the Kittler-Illingworth threshold defined in Equation 5. As a comparison, the Otsu threshold will also be evaluated as it has widespread use. This threshold is defined in Equation 6.

Let \( p(g) \) denote the probability distribution function of a grayscale image for all possible gray intensity values \( g \), \( 0 \leq g \leq G \) where \( G \) is defined by the quantization scheme for the image. E.g. \( G = 255 \) for an 8-bit quantized image. Furthermore, let \( P(g) \) denote the cumulative distribution function of \( p(g) \). Let threshold value \( T \) denote the binary threshold \( 0 \leq T \leq G \). The following definitions apply.

\[
m_f(T) = \sum_{g=0}^{T} g \cdot p(g)
\]

*Equation 1 Intensity Mean Value (foreground)*

\[
m_b(T) = \sum_{g=T+1}^{G} g \cdot p(g)
\]

*Equation 2 Intensity Mean Value (background)*

\[
\sigma_f^2(T) = \sum_{g=0}^{T} [g - m_f(T)]^2 \cdot p(g)
\]

*Equation 3 Intensity Variance (foreground)*

\[
\sigma_b^2(T) = \sum_{g=T+1}^{G} [g - m_b(T)]^2 \cdot p(g)
\]

*Equation 4 Intensity Variance (background)*

Given the definitions in Equation 1 through Equation 4, the optimum thresholds are defined below.
\[ T_{K-I} = \arg\min_T \{P(T)\log \sigma_f(T) + [1-P(T)]\log \sigma_b(T) - P(T)\log P(T) - [1-P(T)]\log[1-P(T)] \} \]

*Equation 5 Kittler-Illingworth threshold*

\[ T_{otsu} = \arg\max_T \left\{ \frac{P(T)[1-P(T)][m_f(T)-m_b(T)]^2}{P(T)\sigma_f^2(T) + [1-P(T)]\sigma_b^2(T)} \right\} \]

*Equation 6 Otsu threshold*

### 3.1.3 Contour detection

The task of contour detection consists of detecting object contours from a binary edge image. Reference [4] proposes an algorithm using border following technique. Border following is one of the fundamental techniques in the processing of digitized binary images.

Let “connected component” denote an area in a binary image with pixels of same value (0 or 1). All pixels in a connected component shall have at least one neighboring pixel belonging to the same connected component. A 1-component is a connected component of pixels with value 1, similar definition for 0-component. A 0-component whose border is the frame of the image is called “background”. A 0-component whose border is not the frame of the image is called “hole”.

![Example of 0- and 1-components](image)

*Figure 4 Example of 0- and 1-components*

Let ”4-connected case” denote the procedure of, for each pixel, examine 4 surrounding pixels according to Figure 5. In Figure 5, p denotes the currently considered pixel and x denotes the examined surrounding pixels.
A similar definition for "8-connected case" applies according to Figure 6.

It is well known that in order to avoid a topological contradiction 0-pixels must be regarded as 8- (4-) connected if 1-pixels are dealt with as 4- (8-) connected.

The border following algorithm derives a sequence of coordinates, or the chain codes, from the border between a 1-component and a 0-component (background or hole). An attractive feature with this algorithm is its ability to do structural analysis of the identified objects and thus find parent objects and holes in identified contours. It does so using a number of definitions and properties when processing the image.

**Definition 1** Border point. In the 4- (8-) connected case, a 1-pixel \((i, j)\) having a 0-pixel \((p, q)\) in its 8- (4-) neighborhood is called a border point.

**Definition 2** Surroundness among connected components. For given two connected components \(S_1\) and \(S_2\) in a binary picture, if there exists a pixel belonging to \(S_2\) for any 4-path from a pixel in \(S_1\) to a pixel on the frame, we say that \(S_2\) surrounds \(S_1\).

**Definition 3** Outer border and hole border. An outer border is defined as the set of the border points between an arbitrary 1-component and the 0-component which surrounds it directly. Similarly, the set of the border points between a hole and the 1-component which surrounds it directly as a hole border.
**Definition 4 Surroundness among borders.** For two given borders $B_0$ and $B_n$ of a binary picture, we say that $B_n$ surrounds $B_0$ if there exists a sequence of borders $B_0, B_1, \ldots, B_n$ such that $B_k$ if the parent border of $B_{k-1}$ for all $k$ ($1 \leq k \leq n$).

With these definitions, the following properties can be defined (see reference [4] for detailed proof)

**Property 1 Uniqueness.** For an arbitrary l-component of a binary picture its outer border is one and unique. For any hole its hole border (the border between that hole and the l-component which surrounds it directly) is also unique.

**Property 2 Surroundness mappings.** For any binary picture, the surroundness relations in Definition 4 are isomorphic with the following mappings:

- a l-component ↔ its outer border;
- a hole ↔ its hole border between the hole and the l-component surrounding it directly
- the background ↔ the frame

An example from [4] of a topological structure following these definitions is given in Figure 7.
The border following algorithm uses these properties to not only find contours but also classify them in a hierarchical manner according to Figure 7 which will facilitate downstream contour classification.

### 3.2 Tesseract legacy OCR engine

Tesseract [14] is an open source OCR engine developed by HP's Bristol laboratories from 1984 to 1994. At the beginning, it was functional as a word-recognition engine for HP's flatbed scanners. In the 1995 UNLV OCR character recognition accuracy test, it won the top rank and caught a lot of wide and global attention. Development of the Tesseract ceased after 1994 after HP abandoned the OCR market.

In 2005, HP contributed Tesseract to the open source community. The source code was obtained by Nevada institute of information technology. Meanwhile, Google began to expand and optimize Tesseract. At present, Tesseract is released on Google Project as an open source Project, which has gained new life. The latest version of Tesseract is 4.01. It supports more than 60 languages, provides an engine and a command-line tool.
Tesseract is using its own page layout analysis since HP has independently developed a page layout analysis technology, and the input is a binary image with optional polygonal test regions defined. Character segmentation and recognition is the design goal of the entire Tesseract.

### 3.2.1 Baseline (row) detection

The connected component analysis is the preparatory work of character recognition. It is distinguishing the contents of images, such as text, picture and tables. For brevity, those connected components will be referred to as Blobs. The line finding algorithm is designed so that a skewed page can be recognized without having to de-skew, thus saving loss of image quality [16]. It plays the key role of blob filtering and line construction process. Once the text lined have been found, the baselines are fitting with curved baselines[16], which is a more common scanning. Blobs takes apart by fitted baselines with an original
straight baseline. The quadratic spline is fitted to the most populated partition (assuming baseline) by least squares fitting.

### 3.2.2 Character detection

The next step is to extract each character from words. Tesseract provides one method that called fixed pitch detection which crops immediately by character calls from the fixed pitch text. However, when the text is non-fixed-pitch or proportional text, tesseract will apply the calculation of the vertical range of the gap between the baseline and mean line. The proportional text would be broken into words by calculate result of definite spaces and fuzzy paces accordingly.

### 3.2.3 Character classification

Classification proceeds as a two-step process. In the first step, a class pruner creates a shortlist of character classes that the unknown might match. Each feature fetches, from a coarsely quantized 3-dimensional lookup table, a bit-vector of classes that it might match, and the bit-vectors are summed over all the features. The classes with the highest counts (after correcting for expected number of features) become the short-list for the next step.

Each feature of the unknown looks up a bit vector of prototypes of the given class that it might match, and then the actual similarity between them is computed. Each prototype character class is represented by a logical sum-of-product expression with each term called a configuration, so the distance calculation process keeps a record of the total similarity evidence of each feature in each configuration, as well as of each prototype, The best combined distance, which is calculated from the summed feature and prototype evidences, is the best over all the stored configurations of the class.

### 3.3 Neural Networks and machine learning

Pictures can present a lot of information, such the location and the association between each object. Each object we can see in the image is the portray of spatial information. Humans can easily decode the message from the picture with their eyes. Under normal conditions, the speed of human eyes in recognizing coherent images is 24 images/second,
which corresponds to about 40ms (milliseconds) per image. This short time is enough for
the human to catch the visual of the image, transfer it to the brain and make the analysis.

Machine learning, ML, and neural networks, NN, is an attempt to emulate the way the
human brain processes information, in this case an image.

There exist a wide variety of ML variants in literature, each one optimized for some
specific characteristics in input data and/or problem class.

### 3.3.1 Supervised and unsupervised machine learning

A fundamental problem classification relates to the concept of supervised and
unsupervised learning.

Unsupervised machine learning is used for problems with continuous output data such as
clustering and anomaly detection. In short, different data mining problems. Unsupervised
techniques do not require any labels to be defined for the input data, and thus training is
not necessary.

Supervised machine learning is the machine learning task of learning a function that can
map an input to an output based on training input-output pairs. The task of this type of
learning is to infer a transfer function based on this training data. Generally, the more
training data applied, the more accurate the inferred function will be when operating on
new, unknown input data.

Supervised machine learning is relevant for classification problems. In a classification
problem, an arbitrary input data shall be mapped to one member of a finite set of output
possibilities, S. OCR detection is a classic example of a classification problem. The input
data is an image of a character, in some format, e.g. jpeg, and the set S is the alphabet that
the transfer function shall classify the character to. The alphabet will of course vary from
language to language and so the training of the system will be done for a specific
language.

In this literature study, only supervised machine learning techniques will be explored.
3.3.2 Neural networks

A neural network is a representation of a transfer function $f$ that can map an input $x$ to an output $y$. The transfer function $f$ can be decomposed into sub-functions that can be further decomposed. Such a decomposition is often referred to as a “layer” in a neural network. If the transfer function $f$ can be decomposed in three different steps, then the network is said to have three layers. The first layer, representing the input data, is called the “input layer”. Any intermediate layer, before the output is produced, is called “hidden layer”. Finally, the output of the network is called “output layer”. The structure is shown in Figure 9.

![Figure 9 General structure of a neural network](image)

A neural network can be visually represented using nodes (“neurons”) connected with arrows thus forming a network. Each neuron can take one or more input data, apply weights to them and feed through its primitive transfer function to produce the output data.

The concept of training a neural network relates to the process of adjusting the applied weights of all neurons to produce an output result with as high accuracy as possible.

3.3.3 Convolutional neural networks

Convolutional neural networks, CNN, are widely used in image classification problems [1]. CNNs are hierarchical neural networks whose convolutional layers alternate with subsampling layers. CNNs vary in how convolutional and subsampling layers are realized and how the nets are trained.
For OCR, this type of image classification reduces to classifying an image with one character or a fixed number of characters with a priori known structure such as a US zip code [9].

### 3.3.4 Recurrent Neural Networks

Recurrent neural networks are a type of neural network designed for the task of making a sequence of classifications. One classification is based on the history of previous classifications; thus, such a network has built-in memory, realized by loops, that can further improve the classification result. This is shown in Figure 10.

![Figure 10: A recurrent neural network. Unrolled loop to the right.](image)

Each classification maps the input data $x$ to the output data $h$. The result is fed into the next classification.

An intrinsic problem with recurrent neural networks is that they typically don’t handle long-term memory well [6]. A classification step $t$ may provide input to the step $t+1$, but in some situations the output is not useful until in step $t+n$, where $n$ may vary. In such situations, the recurrent neural network does not perform well. This effect is commonly referred to as the “vanishing gradient problem”, and has been studied in previous works [6].

### 3.3.5 Long Short-Term Memory networks

Long Short-Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced in [7]. They work tremendously well on a large variety of problems and are now widely used.
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn.

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer as shown in Figure 11.

![Figure 11 The repeating module in a standard RNN contains a single layer.](image)

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way as shown in Figure 12.

![Figure 12 The repeating module in an LSTM contains four interacting layers](image)

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
The cell state can be compared to a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged. The cell state is highlighted in Figure 13.

![Figure 13 The cell state in an LSTM module](image)

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. Figure 14 is an example of a sigmoid gate. The sigmoid is a number between 0 and 1, and the value is calculated during training of the network.

![Figure 14 Example of a sigmoid gate](image)

The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means it does not let any information through,” while a value of one means it lets all information through.

An LSTM has three of these gates, to protect and control the cell state.
3.3.6 Discussion

As shown, several different neural network architectures have been proposed in literature, all having advantages and disadvantages. Reference [8] proposes a neural network architecture combining the advantages of convolutional and LSTM networks. Tesseract [14][23] implements an easy way of combining neural network layers using a network specification language [21]. The pre-trained network shipped with tesseract is using the network architecture proposed in [8]. It has been trained with the network specification in Figure 15, with English alphabet text for many different fonts.

![Figure 15 LSTM network used for OCR step](image)

As can be seen, both forward and reverse LSTM are used which will provide the network with the ability of using both information forward and backwards in time.

4 Implementation

To avoid re-implementation of algorithms already known in literature, Matlab™ [18] has been used due to its extensive built-in support for image processing. Remember that the scope of this report is to propose a prototype implementation of the complete method outlined in 1.2. It is out of the scope to propose efficient implementations of specific signal processing algorithms.

4.1 Input data

The input data to the algorithms consist of digital images of the cabinets. A sample set of cabinets for both training and evaluation has been obtained in the factory using an Iphone8 digital camera. The camera has a resolution of 12 megapixel. The light conditions in the factory are good, which produced a relatively sharp image, easily readable for the human eye.

Already mounted cabinets where manually photographed after quality inspection to make sure no erroneously mounted cabinets were part of the input data.
For a production deployment of the system, an automatic image capturing scheme will of course be necessary, but is out of the scope for this report.

An input data set of 5 cabinets with around 100 product labels was used for evaluation and tuning of the algorithms.

### 4.2 Label Extraction Algorithm

The extraction of the labels consists of three steps

1. Binarization
2. Contour detection
3. Contour classification

The purpose of the binarization detection is to convert the grayscale (intensity) image to a binary edge image with black and white pixels.

The purpose of the contour detection is to find closed contours, or shapes, in the binary image and describe them as simple collections of coordinates.

The final stage, contour classification, will filter out only contours of interests, i.e. the product labels. This is done using a priori knowledge of the size and shape of the labels.

#### 4.2.1 Binarization

As described in 2.1, the Canny detector and a simple binary threshold algorithm will be compared. Matlab provides an implementation of the Canny algorithm in the image processing toolbox. Example of usage:

```matlab
$ rgb = imread(filename);
$ I = rgb2gray(rgb);
$ Ifilt = imgaussfilt(I, 'FilterSize', [5 5]);
$ bw = edge(Ifilt,'Canny',[th th*3]);
```

According to [1], the Canny detector is sensitive to noise. To mitigate this a standard gaussian filter is applied before edge detection.
The second binarization method examined, thresholding, is straightforward and does not require any specific algorithm. Matlab provides a convenience function for thresholding.

\[
\text{bw=imbinarize(I,th);}
\]

Where “I” is the input intensity (grayscale) image and “th” is a threshold in the interval [0,1]. Matlab also provides implementations for calculating optimum Otsu threshold according to [3].

\[
\text{t_otsu=graythresh(I); %Otsu threshold}
\]

There is no built-in function for calculating the Kittler-Illingworth threshold, instead the following Matlab implementation is used

\[
\text{function [T, V] = kittler(I, h, g)}
\]

\[
\text{if ~isempty(I)}
\]

\[
\text{h = histc(I(:), 0:ceil(max(I(:))));}
\]

\[
\text{end}
\]

\[
\text{if nargin < 2}
\]

\[
\text{g = (0:(length(h)-1))'};
\]

\[
\text{else}
\]

\[
\text{g = reshape(g, [], 1);}
\]

\[
\text{end}
\]

\[
\text{C = cumsum(h);}
\]

\[
\text{M = cumsum(h .* g);}
\]

\[
\text{S = cumsum(h .* g.^2);}
\]

\[
\text{sigma_f = sqrt(S ./ C - (M./C).^2);}
\]

\[
\text{Cb = C(end) - C;}
\]

\[
\text{Mb = M(end) - M;}
\]

\[
\text{Sb = S(end) - S;}
\]

\[
\text{sigma_b = sqrt(Sb ./ Cb - (Mb./Cb).^2);}
\]

\[
\text{P = C ./ C(end);}
\]

\[
\text{V = P .* log(sigma_f) + (1-P).*log(sigma_b) - P.*log(P) - (1-P).*log(1-P);}
\]

\[
\text{V(~isfinite(V)) = Inf;}
\]

\[
\text{[~, idx] = min(V);}
\]

\[
\text{T = g(idx);}
\]

\[
\text{end}
\]

4.2.2 Contour detection

Matlab also implements border following algorithm for contour detection. The second argument defines connectivity (4- or 8-) neighbor connectivity.

\[
\text{[B,L] = bwboundaries(bw,4,'noholes');}
\]
Figure 16 shows the output of the border following algorithm discussed in 2.2. The binarized image from the previous step is used as input, but the resulting borders are shown in the original image for clarity.

The border following algorithm will find all shapes in the image, including many that are not of interest.

4.2.3 Contour classification

The purpose of the contour classification is to filter out only objects of interest from the total collection of objects from the previous step.

The contour classification will use the a-priori knowledge that the product labels are rectangle shaped, and that they all are hierarchically organized directly inside the image frame, c.f. Figure 7.

Firstly, all objects classified as holes by the border following algorithms are removed. Secondly, each shape is approximated as a polygon with a maximum approximation error. All 4-sided polygons are subsequently classified as objects of interest. Thirdly, after polygon approximation, a second filtering is designed to remove diamond shaped polygons. This step is carried out by calculating cosine for each angle in the polygons. Polygons with cosine close to 0 (90° angle) will be classified as object of interest.
Figure 17 Remaining shapes after classification (example)

For polygon approximation, the Matlab function “reducem” is used

$[yout, xout] = \text{reducem} \left( \text{boundary}(:,1), \text{boundary}(:,2), \text{tol} \right)$;

Where “boundary” is the output of the “bwboundaries” function. The rest of the algorithm is standard logic and can be found in 8.1.

4.3 OCR

The product numbers need to be extracted from the labels for further processing. This is accomplished by OCR.

OCR (optical character recognition) technology, refers to electronic devices (e.g. scanner, digital camera) that examine characters which are printed out on paper, detect shapes by observing dark and bright patterns, and then use character recognition to translate the shapes into computer text.

In this project, machine learning techniques will be explored for this task. This approach is expected to be more robust to changes in the image characteristics, like fonts and image quality etc. If the characteristics of the product label changes, the neural network used for machine learning does not need to be changed, but only re-trained with new input data.
This task is easier and does not require algorithm or programming skills and can therefore be carried out by operators in the supply chain.

To evaluate the performance of the machine learning approach, it will be compared to a high performing baseline OCR algorithm that does not use machine learning.

The goal is to achieve at least comparable results between legacy OCR and ML based OCR.

An existing implementation of OCR algorithms, Tesseract, defined in [14], is used for evaluation of both legacy and LSTM OCR.

### 4.3.1 Baseline OCR algorithm

The tesseract OCR engine can be found in [23]. It is distributed as open source and can be most easily accessed by cloning the corresponding git repository. Also, several Linux distributions, notably Debian distributions like Ubuntu, have tesseract already packaged and ready to install. The drawback with pre-packaged installations is that the version tends to be quite old. In order to enjoy the latest feature set and bug corrections, we have built tesseract from source in this project. The installation on Debian based systems is a straightforward process. Firstly, prerequisites shall be installed.

```bash
$ sudo apt-get install automake ca-certificates g++ git libtool libleptonica-dev make pkg-config
$ sudo apt-get install libicu-dev
$ sudo apt-get install libpango1.0-dev
$ sudo apt-get install libcairo2-dev
$ cd tesseract
```

Secondly, tesseract shall be built from source (requires an installed C++ compiler)

```bash
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
$ sudo ldconfig
$ make training
$ sudo make training-install
```

This installs both the OCR engine and the training tools for LSTM.
To use the tesseract OCR, simply invoke the tesseract command. In its simplest form it can be used like this

```bash
$ tesseract imagename -l LANG -oem OEM
```

Where LANG is the language and OEM is the OCR engine mode. OEM=0 shall be used for legacy OCR mode. The first argument “imagename” is the name of an image containing the text to be OCR, e.g. a scanned document.

### 4.3.2 LSTM algorithm

The legacy OCR algorithm is compared to ML based algorithm using an LSTM network. It is important to notice that the LSTM network is designed as a text line recognizer that must be used together with a page analysis algorithm. For example, the page analysis included in the legacy tesseract, described in 3.2.1.

To perform OCR using this pre-trained network in Figure 15, tesseract shall be invoked with OEM set to LSTM and language specified to English.

```bash
$ tesseract imagename -l eng -oem 1
```

#### 4.3.2.1 LSTM training

The font used on the product label is Ericsson specific and not part of the training set shipped with tesseract, which will reduce the OCR result for this font. To mitigate that problem, the network is fine-tuned with a sample set of actual product labels to tune the convolutional filters and LSTM weights towards our specific application. The fine-tuning is achieved by using the existing network weights as starting point and just feed the network with new training data. Two of the cabinets mentioned in section 4.1 with 40 product labels were used as sample set for training.

The input to the LSTM training is images with lines of texts together with box definitions specifying the coordinates for each line of text in the training image. In this manner, the training step can be executed without the need for page analysis and the inherent error probability of this algorithm.
The training images and box files need to be put in a specific directory, in this example ~/tesstutorial/myboxfiles. The files need to follow naming conventions to be recognized by the tesseract training tools:

```
${LANG}.${FONTNAME}.${EXPOSURE}.box
${LANG}.${FONTNAME}.${EXPOSURE}.tif.
```

The tif-file can contain several pages, one per image in the training set. After the training data has been prepared, the training is done as a sequence of commands invoking the training tools built in 4.3.1. After each iteration, tesseract outputs the residual word and character error rate. It was found by manual testing that after 4000 iterations, the error rate stopped to decrease. It took around 2 hours on a normal desktop computer to complete the 4000 iterations for this particular set of training images.

```bash
$ src/training/tesstrain.sh --fonts_dir /usr/share/fonts --lang eng --linedata_only --noextract_font_properties --langdata_dir ../langdata --tessdata_dir ./tessdata --fontlist "Impact Condensed" --output_dir ~/tesstutorial/ericssonval --my_boxtiff_dir ../myboxfiles
$ mkdir -p ~/tesstutorial/ericsson_from_full
$ src/training/combine_tessdata -e tessdata/best/eng.traineddata ~/tesstutorial/ericsson_from_full/eng.lstm
$ src/training/lstmtraining --model_output ~/tesstutorial/ericsson_from_full/ericsson --continue_from ~/tesstutorial/ericsson_from_full/eng.lstm --traineddata tessdata/best/eng.traineddata --train_listfile ~/tesstutorial/ericssonval/eng.training_files.txt --max_iterations 4000
$ src/training/lstmtraining --stop_training --continue_from ~/tesstutorial/ericsson_from_full/ericsson_checkpoint --traineddata tessdata/best/eng.traineddata --model_output ~/tesstutorial/ericsson_from_full/eng.traineddata
```
This command sequence will produce a fine-tuned network with weights stored in
~/tesstutorial/ericsson_from_full/eng.traineddata file.

Once the network is trained, the image that is subject to OCR can be input without box
definitions. The page analysis algorithm is run firstly to extract the text lines from the
input image. Each line of text is subsequently fed into the LSTM network for OCR
detection. The OCR procedure using the fine-tuned LSTM network took around 2
seconds to complete for one product label on a desktop computer.

$ tesseract --tessdata-dir
~/tesstutorial/ericsson_from_full ../testlabels/cabinet04_label2.png
stdout -l eng --oem 1

For comparison, the OCR result is shown for both the original and the fine-tuned network.

5 Results

5.1 Label extraction

The label extraction is defined as the sequence of all proposed algorithms

The output from the sequence of these algorithm is a set of rectangles that should match
the desired product labels.

To compare and tune the different algorithms, “missed detection” is used as quality
measure. Let $N_{found}$ be the number of found product labels by the algorithm and $N_{true}$
be the true number of product labels in the input image. The missed detection, $P_{miss}$, is
then defined as 

$$ P_{miss} = 1 - \frac{N_{found}}{N_{true}}. $$

5.1.1 Canny edge detector

A sample output of the Canny detector is shown in Figure 19.
As can be seen, the edges are broken and do not form continuous shapes. To give an idea of the performance, the border following algorithm was run on the edge image without any shape classification. The result is shown in Figure 20. It is clear from this example that no valuable shapes can be found with this edge detection result.
Figure 20 Output of border following algorithm on the Canny edge image (blue shapes)

To restore broken edges, picture dilation is a common method. The standard neighborhood matrix in Equation 7 is used.

\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & 1 & 1 \\
0 & 1 & 0
\end{bmatrix}
\]

*Equation 7 Neighborhood matrix for image dilation*

The dilation is done using Matlab function “imdilate”

\[
\text{bw} = \text{imdilate(bw, nhood)};
\]

Figure 21 shows an example of a dilated binary edge image after 4 dilation iterations
As can be seen in Figure 21, the dilation operation does improve the quality by closing many broken edges. The border following algorithm was used on the edge image in Figure 21, the result is shown in Figure 22. In Figure 22, also shape classification was used to filter out objects that are not of interest. It is clear from this example that the dilation can significantly improve the accuracy. This example also suggests that missed labels are likely not due to the border following or classification algorithm, but due to unsatisfactory performance of the edge detection.
To evaluate the performance of the Canny detector, more than one example is needed. Instead, the threshold of the Canny detector is swept over the set of input images obtained in section 4.1 to evaluate the performance. After each Canny edge detection operation, for each threshold, the image is dilated as described above. Finally, the border following algorithm is used to detect the shapes and the missed detection is calculated. The result is shown in Figure 23.
The Canny detector cannot produce a missed detection lower than ~40%. This is far from the requirement of 0%.

### 5.1.2 Threshold edge detection

To properly evaluate the optimum threshold, a sweep simulation has been done for sample images of the cabinet. The threshold range for a binary threshold is in the interval [0,1], and the sweep has been done for the following set of thresholds [0.1, 0.2, ..., 0.9]. For each threshold value, $P_{\text{miss}}$ is simulated and the results are compared to the suggested optimum thresholds in section 2.1.2. The result of this simulation is shown in Figure 24.
Figure 24 Missed label detections for binary thresholding

The blue line in Figure 24 represents the simulated missed detection as a function of threshold value. The dashed lines depict the Kittler-Illingworth, Otsu and best performing thresholds respectively.

As can be seen in Figure 24, the best threshold for this type of image differs substantially from the general optimum thresholds evaluated from literature. The general optimum thresholds are general purpose and designed to find as many objects as possible in an image. By operating in this way, they will find many objects that are not of interest and miss some objects that are of interest in this specific type of image.

The reason is that the labels in the cabinets are the only objects with almost white pixels. Therefore, it is possible to set a high threshold without missing any of the labels. On the contrary, a low threshold gets too many false detections and consequently misinterprets the labels and fails to identify them as single objects. When the threshold is tuned using real sample images it is possible to achieve the requirement of 0% missed detection, as can be seen in the figure.
To illustrate how the different threshold values affect the binary image, three examples are given below. Figure 25 shows the result for the Kittler-Illingworth threshold, Figure 27 for the Otsu threshold and Figure 29 for the simulated optimum threshold (0.8).

Figure 25 Example of binarized cabinet image using Kittler-Illingworth threshold
Figure 26 Detected shapes with Kittler-Illingworth threshold (example)
Figure 27 Example of binarized cabinet image using Otsu threshold

Figure 28 Detected shapes with Otsu threshold (example)
Figure 29 Example of binarized cabinet image using tuned optimal simulated threshold (0.8)
5.1.3 Discussion
With proper tuning of the threshold it is possible to achieve better results with thresholding than with the canny detector. This is probably due to the general nature of the Canny detector. The thresholding performs better with this type of input image but may perform significantly worse with other types of images. For this project an optimum binary threshold of 0.8 is used.

5.2 OCR
In this chapter the results from the OCR algorithms, outlined in 4.3 and implemented as described in 4.3, are presented. A sample set of 4 cabinets with a total of 87 product labels have been used for this performance evaluation.
5.2.1 Legacy OCR

The legacy OCR implementation described in 4.3.1 is used for comparison.

<table>
<thead>
<tr>
<th>Untrained character error rate</th>
<th>Trained character error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>51.17%</td>
<td>36.64%</td>
</tr>
</tbody>
</table>

*Table 1 Legacy OCR algorithm results*

The legacy OCR implementation performs poorly on the input data provided by the label extraction algorithm. The label extraction does not perfectly fit to the label, it often leaves a dark edge around the label due to imperfect detection. It is a known weakness in traditional OCR that distortions in the image edges often are misinterpreted as characters. This is a probable reason for the poor result.

5.2.2 LSTM OCR

<table>
<thead>
<tr>
<th>Basic character error rate</th>
<th>Fine-tuned character error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.18%</td>
<td>0.3%</td>
</tr>
</tbody>
</table>

*Table 2 LSTM OCR algorithm results*

The LSTM algorithm performs better than traditional OCR. The trained network does not mis-detect dark edges as characters. However, if the network is not trained with the font used in the input images the performance is still relatively poor. Since LSTM network do not generalize well, the input data need to be similar to the trained data. The fine-tuned network performs order of magnitude better than the network that was only trained with standard fonts.

This result is particularly promising considering the intended use-case where the font and structure of the product labels may vary over time.
6 Conclusion and Future Work

6.1 Conclusion

An image processing method for automatic detection of products mounted in a cabinet has been developed in this project. Standard algorithms from research and literature have been applied and modified when necessary. The requirement on the algorithm is to produce an error free product detection that will be applied as an automated quality assurance step in the factory. It is shown that using the proposed algorithms it is possible to achieve this goal.

Let us reiterate the research questions and try to give an answer based on the results obtained.

Image analysis methods like binarization and shape detection are widely adopted in literature. Can such algorithms perform well enough to detect the product labels in this application? What accuracy is possible to obtain?

In section 4.2, the proposed label extraction is explained, and the results are shown in section 5.1. It can be concluded that general methods from literature do not perform well. The canny detection fails to detect all labels and optimum thresholds perform relatively poor. However, a visual analysis of the image suggests that the product labels are significantly brighter than other objects in the image. Raising the threshold well above optimum thresholds will therefore be able to generate the desired error-free result. Based on this result, the answer to the research question is “No”. Heuristic adaptions to existing algorithms are necessary.

Can ML be used to perform OCR on the product labels? Which character error rate is possible to obtain?

In section 4.3.2 the proposed ML based algorithm is explained. Section 5.2.2 shows the results of the algorithm. The conclusion is that an LSTM based network, properly trained, can meet, and significantly outperform, legacy OCR methods. It can indeed produce low enough error rate for error free data base look-ups. The big difference in performance
between legacy OCR and LSTM is however not representative. It is a function of the quality of the input labels. This project suggests a whole set, or chain, of algorithms that are designed to perform a specific task. The performance that can be obtained in one step is dependent on the input quality from the previous step. Label extraction has been shown to be a relatively difficult task that requires tuning of existing methods. And still, with the methods suggested, the quality of the labels is not perfect which degrades the performance of the downstream OCR algorithm. However, it is clearly shown that an LSTM based OCR is much more robust to imperfection in the input images than a traditional OCR algorithm. The answer to the second research question is therefore “yes”, it is possible to achieve the goal with an ML based OCR algorithm.

6.2 Future Work

Based on the limitations in 1.3, several improvements and refinements can be proposed.

6.2.1 Expand usage to other products

The prototype developed in this project is fine-tuned towards a specific product family where the quality assurance step was most urgently needed. Ericsson supply chain manufactures products for a whole range of product families, and they all share common characteristics. Cabinets, boards and product labels can be found in every product line. It should be possible to fine tune the label extraction algorithm to also work for other products. There might be differences in label size that may require adaptations, but the label placement in the picture has not been used as feature in the algorithm and should not matter.

Similarly, product labels from other product families may have a different layout and different text font. However, this should be possible to handle by re-training the LSTM network using the steps outlined in 8.2.

6.2.2 Deploy in real-time environment

A natural next step would be to deploy the prototype solution in a real environment. In the factory, cabinets are being produced and assembled at a high pace. In order to cope with this pace, the system must be subject to real-time constraints. A response time of in
the order of 1 second is reasonable from the picture of the cabinet is taken until a quality verdict, with an optional list of erroneous mounts, is produced.

The first step in a deployment would be to evaluate hardware requirements. Can a desktop computer produce fast enough response or is custom hardware needed to accomplish the task?

When determining hardware requirements, a deeper analysis into computational efficiency of the algorithms must be performed. Is it possible to optimize any of the algorithms to achieve lower hardware requirements?

6.2.3 Train LSTM OCR with font definition instead of product labels

If the font definition for the product labels is available, it can be used instead for training. The procedure involves creating a sample text image using the font. The sample text should contain as large portion as possible of the language used. In this case, the language is limited and consists of all possible product numbers in the data base. Training with such images could improve the OCR result.
7 References

[17] Minakshi Kumar, Digital Image Processing, URL
[18] Matlab™, URL
[19] OpenCV, URL
[21] VGSL specification, Google, URL
[22] Google Scholar search engine, URL
[23] Tesseract OCR, URL
8 Appendix A, source code implementation

8.1 Label extraction

The label extraction was implemented and analyzed in matlab. Refer to the following source code to reproduce the results presented in this report.
clear all
close all
warning off
filename='cabinet01.jpg';
method='canny';
rgb = imread(filename);
I = rgb2gray(rgb);
t_otsu=graythresh(I); %Otsu threshold
t_kittler=kittler(I)/100; %Kittler-Illingworth threshold
if strcmp(method, 'canny')
    th_list = 0.01:0.02:0.21;
else
    th_list = 0.1:0.1:0.9;
end
for th = th_list
    figure(1);
    num_detect = 0;
imshow(I,'InitialMagnification',30);
    if strcmp(method, 'canny')
        Ifilt = imgaussfilt(I, 'FilterSize', [5 5]);
        bw = edge(Ifilt,'Canny',[th th*3]);
        nhood = [0 1 0;1 1 1;0 1 0];
        for dil=1:4
            bw = imdilate(bw, nhood);
        end
    else
        bw=imbinarize(I,th);
    end
    bw=removesmallobjectsandholes(bw, 1000);
    [B,L] = bwboundaries(bw,4,'noholes');
    figure(1)
    hold on
    length(B)
    for k = 1:length(B)
        boundary = B{k};
        tol=40;
        [yout, xout] = reducem(boundary(:,1), boundary(:,2), tol); %Polygon approximation
        if length(yout)>=3
            poly=polyshape([xout yout]);
            numcorners=length(poly.Vertices);
            a=area(poly);
            h=max(poly.Vertices(:,2))-min(poly.Vertices(:,2));
            w=max(poly.Vertices(:,1))-min(poly.Vertices(:,1));
            ratio=h/w;
            %text(poly.Vertices(1,1),poly.Vertices(1,2),[num2str(numcorners) ', '
            ' num2str(ratio,3)],'Color','red')
            if numcorners >= 4 && ratio>=1.3 && ratio<=2.2
                num_detect=num_detect+1;
                plot(boundary(:,2), boundary(:,1), 'b', 'LineWidth', 2);
                plot(poly,'FaceAlpha',0,'EdgeColor','green')
            end
        end
    end
    hold off
    title(['Original grayscale ' filename ' Detected contours: ' num2str(num_detect) ' Threshold: ' num2str(th)])
    keyboard
end
8.2 OCR

The OCR is done with the open source software package Tesseract. It is available under GPL license. In this project it was used in an Ubuntu 18.04 system. This chapter lists the command line references used to produce the results.
Install missing fonts
>sudo apt-get update
>sudo apt-get install ttf-mscorefonts-installer
>sudo apt-get install fonts-dejavu
>sudo apt-get install gfonts
>fc-cache -vf

If ttf-mscorefonts-installer doesn't work properly:
>sudo apt-get --purge remove ttf-mscorefonts-installer
>cd
>wget http://ftp.br.debian.org/debian/pool/contrib/m/msttcorefonts/ttf-mscorefonts-installer_3.6_all.deb
>dpkg -i ttf-mscorefonts-installer_3.6_all.deb
>fc-cache -vf

Build and install tesseract
Commands are taken from these websites:
https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00
https://github.com/tesseract-ocr/tesseract/wiki/Compiling-%E2%80%93-GitInstallation

%Configure directory structures for the tutorial
sudo apt-get install git
mkdir ~/tesstutorial
cd ~/tesstutorial
mkdir langdata
cd langdata
wget https://raw.githubusercontent.com/tesseract-ocr/langdata_lstm/master/radical-stroke.txt
wget https://raw.githubusercontent.com/tesseract-ocr/langdata_lstm/master/common.punc
wget https://raw.githubusercontent.com/tesseract-ocr/langdata_lstm/master/font_properties
wget https://raw.githubusercontent.com/tesseract-ocr/langdata_lstm/master/Latin.unicharset
wget https://raw.githubusercontent.com/tesseract-ocr/langdata_lstm/master/Latin.xheights
mkdir eng
cd eng
wget https://raw.githubusercontent.com/tesseract-ocr/langdata/master/eng/eng.training_text
wget https://raw.githubusercontent.com/tesseract-ocr/langdata/master/eng/eng.punc
wget https://raw.githubusercontent.com/tesseract-ocr/langdata/master/eng/eng.numbers
wget https://raw.githubusercontent.com/tesseract-ocr/langdata/master/eng/eng.wordlist
cd ~/tesstutorial
git clone --depth 1 https://github.com/tesseract-ocr/tesseract.git
cd tesseract/tessdata
wget https://github.com/tesseract-ocr/tessdata/raw/master/eng/eng.traineddata
wget https://github.com/tesseract-ocr/tessdata/raw/master/osd.traineddata
mkdir best
cd best
cd ~/tesstutorial/tesseract
%Build tesseract and training tools
sudo apt-get install automake ca-certificates g++ libtool libleptonica-dev make pkg-config libpango1.0-dev
./autogen.sh
/configure
% Build and install tesseract
Make
sudo make install
sudo ldconfig
% Build and install training tools
make training
8.2.1 OCR network specification

The LSTM network used in tesseract is defined using VGSL (variable graph specification language) developed by Google as part of its tensorflow machine learning package. A description of the language can be found in [21]. The complete network specification used in this project is shown in Equation 8.

\[
[1,36,0,1 \text{ct}3,3,16 \text{mp}3,3 \text{ly}64 \text{fx}96 \text{rx}96 \text{fx}512 \text{c}1]
\]

*Equation 8 VGSL network specification for LSTM based OCR*