OCR of dot peen markings

with deep learning and image analysis

Hannes Edvartsen

Computer Science and Engineering, master's level
2018

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Abstract

A way to follow products through the chain of production is important in the process industry and it is often solved by marking them with serial numbers. In some cases permanent markings such as dot peen marking is required. To ensure profitability in the industry and reduce errors, these markings must be read automatically. Automatic reading of dot peen markings using a camera can be hard since there is low contrast between the background and the numbers, the background can be uneven and different illuminations can affect the visibility. In this work, two different systems are implemented and evaluated to assess the possibility of developing a robust system. One system uses image analysis to segment the numbers before classifying them. The other system uses the recent advances in deep learning for object detection. Both implementations are shown to work in near real-time on a cpu. The deep learning object detection approach was able to classify all numbers correct in a image 60% of the time, while the other approach only succeeded in 20% of the time.
Chapter 1

Introduction

Within the process industry there is a great need for traceability. To follow products through the chain of production, they must be marked [27]. This is often solved by text and bar codes that are stamped directly on the product. In the heavier process industry more resistant markings, that can be read after the product has been grinded or painted, are needed. For this purpose dot peen marking is often used [3].

Since automation is important to ensure profitability in the industry, these markings must be read automatically. Currently, there is no commercial product that can read dot peen markings on uneven and glossy metal.

1.1 Background

Optical character recognition (OCR) is the procedure of a machine scanning an object and retrieving characters from it. Compared to manual entry of data, this can save time and reduce errors.

OCR of any type of font, including handwritten characters, can be done easily today using free software such as Tesseract [6]. The struggles with OCR today is when the characters are hard to distinguish from the background or when they are not clearly visible.

1.2 Motivation

Automating reading of serial numbers can increase the efficiency of handling and tracking different objects, thus saving time and money. Currently these numbers have to be entered to a computer manually, which is both time consuming and prone to error.

1.3 Problem definition

The goal of this project is to create a proof-of-concept system that can localize and classify dot peen marked characters on a metallic surface. The commercial potential of the system will also be evaluated. As a commercial product, the system would be implemented on a hand held device.

The great challenges with this problem in relation to regular scanning of numbers is:

- Identifying single characters on an uneven background.
- Light sources can affect the shadowing and make it harder to read since it is a 3D marking.
• The material can be shiny metal and thus reflect the light in various ways.

• Markings can have different distances between them making an irregular pattern.

• Different distances between markings also make it hard to tell where one character ends and another begins.

See Figure 1.1 for some example images.

![Figure 1.1: A few examples showing different types of difficulties in images (some images have been cropped to exclude irrelevant background).](image)

This is a rather specific problem, which can be simplified by making some assumption on the data. e.g. the rotation and sizes of the characters.

To evaluate the performance of the system the two following parameters must be taken into consideration.

**Accuracy** is a requirement for a robust system. In this case, for a prediction to be correct, all characters in the image must be found and correctly classified. This can be hard since the number of characters is not known.

To minimize the number of images that are falsely predicted to be correct, check-sums and multiple tries can be used to verify the prediction. Nevertheless, the system must be accurate enough to classify all characters on an object correctly within a few tries.

**Time efficiency** is needed for the system to operate in near real-time(within a few seconds). A factor that makes time efficiency even more important is that each object requires multiple scans to verify its correctness. Thus time efficiency depends on accuracy.

### 1.4 Delimitations

Due to time limitations and value for a proof-of-concept, some functionality will be disregarded in this implementation.

For simplicity, all tests will be run on a laptop with pre-generated images, not on a hand held device with a live camera.
The check-sum verification will not be included in the report since it will have no effect on evaluating the accuracy of the system. Neither will multiple tries of same object be considered when calculating accuracy.

1.5 Thesis structure

In Chapter 2, two different methods chosen for solving the problem is presented. In Chapter 3 some work related to object detection and OCR are reviewed. The theory behind a few computer vision operations and object detection using deep learning is described in Chapter 4. Later on, in Chapter 5, the implementation of the two different methods are explained, and in Chapter 6 they are evaluated and compared to each other. In Chapter 7 the solution, result, problems encountered and alternative solutions are discussed.
Chapter 2

Method

In this thesis two methods of solving this problem will be tested and compared. One method will be similar to how classic OCR systems are implemented, i.e. using image analysis to segment out the characters from the image and then classify them one by one using machine learning. The other method will be using a deep learning object detector. A deep learning object detector is a network that is trainable to find multiple objects in an image. A more detailed description of the techniques used is found in Chapter 4. For the rest of the thesis, the two methods will be referred to as the classic version and the deep learning version.

The strategy to solving this problem had two stages. First, a theoretical survey was used to find relevant techniques. Second, experimentation with these techniques was done to find how they worked together in solving this task. The implementations of the system was done iteratively. That is, first a base system was set up. Then each part was enhanced multiple times. The advantage of this method is the possibility to evaluate the system during development to identify existing bottlenecks and address them.

For evaluation of the two methods, they were tested on the same set of images on a Lenovo ideapad 710s laptop with an intel core i5 7th generation cpu(no gpu was used). The metrics for evaluation are:

- Average time required for scanning one image. i.e. the total time it takes to process the whole set of images divided with how many images the set contains.

- Percentage of images that are fully correct classified. i.e. all characters and nothing but the characters have been found.

- A precision score in percentage. How many of all found characters are correctly classified?

- A recall score in percentage. How many of all characters are found and correctly classified?

The results are presented in Section 6.3.
OCR systems for document scanning of the current state-of-the-art such as Tesseract [6] and ABBYY [1] are based on the following methods. Pre-processing, character extraction, feature extraction, classification and post-processing [10]. These methods work very well when identifying characters with high contrast on even background. As the characters becomes harder to distinguish from the background the character extraction becomes more difficult. See Figure 3.1 for an example of document scanning.

Object detection is a research area that has started to flourish in recent years as convolutional neural networks has risen in poularity for computer vision[5, 4, 23, 14, 15]. The first popular object detector came in 2013 [5], which was slow but accurate. In the following years the process has become much more efficient and accurate. Currently some systems can run object detection algorithms in over 100 frames per seconds on GPU’s [22]. Other systems can classify each pixel in an image to get perfectly segmented objects [7]. See Figure 3.2 for two examples of object detectors.
Figure 3.2: Figure (a) is the yolo object detector which can run in over 100 frames per second. Figure (b) shows the mask r-cnn prediction, which label each pixel to a distinct class. Mask r-cnn runs in about 5 frames per second.
In this chapter a few different digital image analysis and machine learning techniques will be explained. The techniques explained here have been used in the implementation of the final system or during testing different approaches.

4.1 Image analysis

4.1.1 Segmentation

Segmentation of an image is to separate the different objects from each other and the background, and thus easily found by a computer in the image. A fast and popular segmentation method will be explained in this section. Other methods exist, but were considered not appropriate for this problem.

**Thresholding** is an efficient way to segment an image. That is, following a threshold every pixel becomes a predefined fixed value. This way, objects with different pixel intensities can be separated. Binary thresholding is when the output only has two different values and can be represented with the equation

\[
    v' = \begin{cases} 
    0 & \text{if } v < T \\
    1 & \text{if } v \geq T 
    \end{cases} \tag{4.1}
\]

Where

- \( T \) = threshold value
- \( v \) = pixel value from image
- \( v' \) = pixel value after thresholding.

The two main types of thresholding are global and local. In global thresholding the threshold value \( T \) from Equation 4.1 is the same for the whole image. In local thresholding, different values for \( T \) are calculated for different regions in the image. A local threshold method is often more resistant to shadows and light reflections in the image [11].

There exists numerous functions to determine the value of \( T \). Here Otsu, Sauvola and Phansalkar will be described.
Otsu is based on finding $T$ using the pixel intensity histogram [20] of an image. Namely calculating where the peaks in the histogram are and setting $T$ to divide the image as appropriate as possible. Otsu works best when there exists distinct peaks in the histogram [18].

Sauvola is often used for document binarization. $T$ is determined by

$$T = m \cdot \left[ 1 + k \cdot \left( \frac{s}{R} - 1 \right) \right],$$

(4.2)

where $m$ and $s$ are the mean pixel intensity and standard deviation of a region in the image. $R$ and $k$ are constants which are often set to 128 and 0.5 respectively [24].

Phansalkar is an extended version of sauvola making it more sensitive. Sauvola algorithm assumes the text in documents are rather thin, which can result in holes in objects that are wider. This can happen if the current window of consideration mainly contains an object that is dark, then $T$ becomes smaller than the pixel values for the object such that the binarization of the area becomes white even though there is an object there. Phansalkar extends Equation 4.2 as

$$T = m \cdot \left[ 1 + p \cdot e^{-q \cdot m} + k \cdot \left( \frac{s}{R} - 1 \right) \right],$$

(4.3)

where $p$ and $q$ are two new constants often set to 3 and 0.4 respectively [19]. When $p = 0$, phansalkar is the same as sauvola.

Examples of three different threshold functions are seen in Figure 4.1.

Figure 4.1: Effects of different threshold functions on an example image. As seen in (b) the global method does not work well when there is shadows in the image. Figure (c) is the best in this case where phansalkar is used. Sauvola used in (d) is not enough sensitive to find anything.
4.1.2 Filtering

Filtering can be used to enhance certain features or alter an image. A filter operation slides a kernel over each pixel in an image and performs an operation based on nearby pixels. The kernel is a matrix and its size determines which pixels are considered nearby. The values in the kernel can also affect what the output image will become. In this section a few useful types of filters will be presented.

**Convolution** filters use a weighted kernel to modify the image. For each pixel in the image the kernel passes by, a new value will be calculated based on the weights in the kernel and the pixels it overlaps with. Each weight in the kernel will be multiplied with the pixel it overlaps with, and the sum of all products will be the new value for the current pixel. Figure 4.2 presents an example of a convolution filter.

Figure 4.2: An example of a convolutional filter. The filter is moved across each pixel calculating what the feature map will be. The calculations taking place is a matrix multiplication of the filter and the current overlapping part of the features. Then the sum of the calculated matrix becomes the new value for that position.

**Min and max** filters can be used to enhance the darker or lighter parts of an image. These types of filters sets each pixel to the same value as the minimum or maximum value of nearby pixels. The nearby pixels are defined by those within the kernel.

The pixel in the center of the sliding kernel will take the same value as the one with the highest or lowest pixel within the kernel. Thus if a min filter is used, darker regions in the image will be enlarged. An example of min and max filters can be seen in Figure 4.3.

![Figure 4.3](image)

Figure 4.3: Effects of a min and max filter with kernel size of 3x3.

**Morphological transformations** is another type of filter which can be useful in object detection when modifying binary images. Some widely used morphological transformations are erosion, dilation, closing, opening, white top hat and black top hat. All of these are based on erosion and dilation. The effect of the transformation can be adjusted with different shapes and sizes of the kernel.
Erosion is a transformation where objects are reduced in size, and likewise dilation is when objects are expanded. A closing transformation is an dilation followed by an erosion and opening transformation is an erosion followed by an dilation. The effect of an closing is that the objects will first expand in all directions, removing small patches of background, and then return to its original size. Opening will have the same effect except that objects are first reduced in size before re-sized again. Thus the smallest objects will disappear. Closing and opening is therefore often used to bind objects together or remove small objects. White top hat is the difference between the original and the opening of the image. Black top hat is the difference between the original and the closing of the image. Top hat operations can be used to extract objects of different sizes. Examples of these transformations can be seen in Figure 4.4.

![Figure 4.4: Examples of different morphological transformations using a kernel of size 2x2. Here objects are considered to be black.](image)

4.1.3 Contrast enhancement

Contrast enhancement can make different objects seem more separated. A popular method to adjust contrast is contrast limited adaptive histogram equalization (CLAHE). CLAHE will calculate new pixel values based on the histogram of nearby pixels to get better contrasts. Since it only looks at nearby pixels, even the smallest changes in intensity can be found and made visible. An example of CLAHE is shown in Figure 4.5.
4.2 Object classification

Once a character has successfully been segmented out from an image it needs to be classified as a specific character. Two common ways of doing this is by using feature based machine learning or convolutional neural networks.

4.2.1 Feature based machine learning

In feature based machine learning some features in the image must be extracted before classification. A popular feature extraction method for OCR is histogram of oriented gradients (HOG). HOG extract features from patches in the image such that closeness to other pixels are taken into consideration when features are generated. For a deeper explanation of HOG, see [8].

K nearest neighbour saves all the features for an image in a vector. This vector is then placed in a vector space. When enough vectors are added, clusters will start to take form for each different class. To classify a new vector, calculate the distance (often euclidean distance is used) to the clusters. The $k$ nearest neighbours will then vote for what class it belongs to [2]. See Figure 4.6 for a simple example of how $k$ nearest neighbours work.

![Figure 4.6: A simple example of $k$ nearest neighbours. Yellow dots belong to one class and the purple dots belong to another. The star is of unknown class. When predicting the star, simply calculate the distances to the other dots. Then the $k$ nearest neighbours will vote on which class it belongs to. In this case when $k = 3$ the star is predicted to be of class B, and when $k = 5$ it is predicted to belong to class A.]
**Support Vector Machines** are similar to k nearest neighbours in the sense that the training feature vectors are placed in a vector space of n dimensions. The difference is that when all training data is inserted, for each class a hyper-plane is calculated such that it divides itself from the rest of the data. If the data is undivisible in such a fashion, more features can be added by just adding together the square of previous features until the data is divisible (i.e. creating more complex features). When a new vector is to be classified, simply just check which side of the hyperplanes it belongs to [21]. Figure 4.7 visualizes how support vector machines work in a two-dimensional space with two classes.

![Figure 4.7: An example of a support vector machine with two dimensional data. The blue and red squares are two different classes. When predicting a new vector, calculate which side of the line it resides in.](image)

### 4.2.2 Deep learning

The phrase deep learning originates from *neural networks*. Neural networks consists of several neurons that are structured in layers. Neural networks can learn advanced patterns from features. Here, the basic functionality of a neural network, how features can be found automatically and how a network can learn will be covered.

**Neural networks**

A neural network can be seen as a function

$$f(X) = Y,$$ (4.4)

where \( X \) is some features and \( Y \) is one of a set of predefined classes. Each class has a label to represent it. The basic idea with a neural network is to train structured neurons to become the function \( f(X) \) by feeding it a lot of different \( X \) and corresponding \( Y \). The data that is used for training is known as the *ground truth*.

As seen in Figure 4.8, each layer of neurons provide their output as input to each neuron in the next layer. The layers are called *fully connected*, since all nodes are connected to each other between the neighbouring layers.
Figure 4.8: A neural network with 4 layers. One input layer, two hidden layers, one output layer. All layers between input and output are called hidden layers.

**Neurons** are the building blocks of a neural network and can be given as the function

\[
output = f(bias + \sum_{i=0}^{k} w_i \cdot input_i). \tag{4.5}
\]

Where \(w_i\) is a weight associated with \(input_i\), \(f\) is an arbitrary function, \(bias\) is a static input used as a regulator for the amplitude of the output and \(k\) is the amount of neurons in the previous layer. That is, each neuron takes a number of inputs, multiplies each input by a weight, sums them up, add the bias, applies a function to the sum, and produces an output. The weights in an neuron are trainable and the reason it can learn.

**Activation functions** are the functions that are applied to the summed up inputs in a neuron. Often the sigmoid function defined as \(s(x) = \frac{1}{1+e^{-x}}\) is used in the last layer to get an \(output \in (0,1)\). Activation functions in the rest of the network are used to introduce non-linearity and the most common choice is Rectified Linear Unit (ReLU), which is defined as \(f(x) = max(0,x)\). ReLU is often used because of its simplicity and effectiveness.

**Feature extraction**

For two dimensional data, such as images, a trainable feature extractor which consists of different filters can be used. These are also structured into multiple layers such that more advanced features can be extracted deeper into the network.

**Convolution filters** as explained in Section 4.1.2, are used to extract features. A neural network which uses convolutional layers for feature extraction is called a CNN. Just like the weights in a neuron, the weights in the kernel of a convolution filter can be trained to find the features important for the task at hand. The new image obtained after applying the convolution filters is called a feature map.

**Depthwise and pointwise convolution filters** are time optimized convolution filters. The convolution is split into two parts to reduce the amount of operations that are necessary. For an explanation of how they work and how efficient they are, see [9].
Pooling is often used to downsample the data, i.e. reduce the number of features, but keep the most important. This is done to reduce computation time and to reduce the risk of focusing on non-relevant features. The operation is similar to a filter, except that instead of moving just one pixel from previous pixel, two or more is moved. The number of positions the filter is moved is called stride and the most common is 2, i.e. when the stride is 2, the image will be halved in width and height. The most popular pooling strategy is max pooling, which extracts the max value that the filter is passing. See Figure 4.9 for a simple example of pooling.

![Pooling Example](image)

Figure 4.9: An example of max and average pooling. A kernel of size 2x2 and stride 2 is used in this example.

Learning

A CNN can only function well when all the weights in both the convolutional and hidden layers are adjusted to produce the wanted output. Adjusting the weights is called training. This is done by first making the network predict on some data, then calculating how wrong it was, and adjusting each weight such that the error is minimized. This is done iteratively until the network doesn’t get any better.

Loss functions are used to calculate the error of the network. That is done by comparing the predicted output $y$ and the real label $\hat{y}$. For classification, the labels are often transformed to the form 1 or 0 (object or not object), then the prediction can be everything between 0 and 1. For multi-class classification each label has its own prediction value.

A common loss function for classification is cross entropy loss which can be described as

$$\text{cross entropy loss} = \begin{cases} -\log(y) & \text{if } \hat{y} = 1 \\ -\log(1 - y) & \text{if } \hat{y} = 0 \end{cases}.$$  \hspace{1cm} (4.6)

In cases where there exists few hard training examples and many easy training examples, one could use focal loss which minimizes penalty when the predicted probability is high. Focal loss is defined as

$$\text{Focal loss} = \begin{cases} -\alpha(1 - y)^\gamma \log(y) & \text{if } \hat{y} = 1 \\ -\alpha(y)^\gamma \log(1 - y) & \text{if } \hat{y} = 0 \end{cases}.$$  \hspace{1cm} (4.7)

In Equation 4.7, $\gamma$ and $\alpha$ are two parameters that adjusts the balance in the function. Focal loss has been proven to work well when these are set to 2 and 0.25 respectively [15]. When $\gamma = 0$ and $\alpha = 1$, focal loss is the same as cross entropy.
For regression where the values are not always between 0 and 1, $L_1$ or $L_2$ loss functions are often used. They are defined as

\[
L_1 = \sum_{i=0}^{n} |\hat{y}_i - y_i|, \tag{4.8}
\]

\[
L_2 = \sum_{i=0}^{n} (\hat{y}_i - y_i)^2. \tag{4.9}
\]

Where

$n = \text{The amount of different values.}$

**Back propagation** is the process of going back through the network after a prediction is made and adjusting the weights based on what loss has been calculated. The impact on the error is calculated individually for each weight in the network, starting with the layer closest to the output layer. When the impact has been calculated for a weight, it is adjusted to decrease the error. How much the weight is adjusted is also affected by a value called *learning rate*. Learning rate is a parameter used to change how fast the network will be trained. Lower learning rate results in longer training time and vice versa.

**Overfitting** is when the network becomes really good at predicting on the training data, but bad at predicting never seen data. That is, the system does not generalize well. This can be prevented using different methods such as pooling, data augmentation and drop out. Data augmentation is when the training data is slightly altered each time it is passed through the network for training. Drop out is when a number of randomly chosen weights are frozen during training such that they are not affected by the current training iteration. Pooling is described in Section 4.2.2.

### 4.3 Object detection using deep learning

Object detection is the task of recognizing where (possibly multiple) objects are in an image and also label them as a distinct class. In 2013 deep learning was proven useful in object detection with the creation of region-based CNN[5]. Since then, several different deep learning object detectors have been developed [4, 23, 16, 22]. The main improvement has been in time efficiency.

**Re-usage of feature maps** is the main reason for current state-of-the-art object detectors time efficiency. That is, the image to be predicted is only once passed through feature extraction layers, as opposed to passing each region of interest through feature extraction. Thus the same feature maps are used for region proposals and all predictions.

**Classification** methods such as neural networks always requires the same number of inputs. This can cause some problems for object detection since the regions in an image to be classified can have different sizes. Solving this problem is often done using a special type of pooling. What is done is basically just dividing the region into the amount of pre-defined amount of cells regardless of what the real size of the region is. Then the largest value in each cell is chosen as the representative for it. See Figure 4.10 for an example of how this can work.
Region proposing is an important stage for object detectors and there exists two different types of methods. One-stage and two-stage detectors. One-stage detectors generate regions using the sliding window approach. Each region the window passes by is classified as background or one of the possible classes. To decrease the number of regions, the window can be moved multiple steps at the time. Two-stage detectors generate regions in a separate algorithm such as region proposal network (as explained in [23]) which uses the feature map to localize interesting areas.

What is commonly used in all region proposal methods is anchors. Anchors are a fixed set of sizes and height/width ratios that will be used as ground boxes. e.g. in a sliding window approach, the different anchors will be used as the window, and for each anchor a prediction will be made (is it background or one of the available classes). The sizes and ratios of anchors are adjusted for the specific objects that is to be found. See Figure 4.11 for an example.
Figure 4.11: An example of how anchors can be used in a sliding window approach. Here the blue and red boxes are anchors, the green square is the current position of the window. Since the window is moved a fixed set of steps for each prediction, it can be seen as the image is divided into a grid where each cell is predicted upon. Each cell in the grid gets a prediction for each anchor specified. In this case there is two. Different aspect ratios for anchors are used since objects may have different aspect ratios.

Generally speaking, one-stage detectors are faster but less accurate than two-stage.

Bounding box regression is often used to get more precise location predictions. Since the regions proposed rarely fit the object exactly, bounding box regression can be used to predict the mid-point, width and height of the object. This uses same features as classification and works in a similar way. The difference is that there are four values to be predicted, and they are not only between 0 and 1. Loss functions such as those in Equations 4.8 and 4.9 are instead used to calculate the error. To check whether a bounding box is predicted correctly, intersection over union (IoU) is used. IoU is a measurement for how much two boxes overlap and is defined as

$$ \text{IoU} = \frac{bb_{GT} \cap bb_{pred}}{bb_{GT} \cup bb_{pred}} $$

(4.10)

Where

- $bb_{GT} = \text{Bounding box for the ground truth}$
- $bb_{pred} = \text{Predicted bounding box}$

Two boxes overlap perfectly when $\text{IoU} = 1$ and nothing when $\text{IoU} = 0$. An overlap is considered approved when $\text{IoU}$ is larger than a specified threshold.

For a more detailed description of how bounding box regression works, see [5].

Feature pyramid networks have been proven useful as feature extractor for object detectors [14, 15]. A feature pyramid network follows a bottom-up top-down pathway, i.e. First from the original image, convolution layers, pooling, etc. are applied to gather rich features and decrease the resolution. Then from the highest level of feature maps, up-sampling is done by increasing the resolution back to previous size and combining the feature maps from both pathways. The up-sampled feature maps then contain rich features with high resolution at different scales, giving more scale independent predictions. The layers in the bottom-up pyramid are called the backbone and are independent from the top-down pathway. Figure 4.12 gives an illustration of how it works.
Figure 4.12: An illustration of how feature pyramid networks work. The pyramid to the left is a number of convolution and pooling layers which extracts features and reduces resolution. The pyramid to the right is up-sampled from the levels of the feature extraction. The combination is done by adding the corresponding feature map from the bottom-up pathway to the top-down feature maps. Predictions can then take place on feature maps of different scales with rich features.

**Non maximum suppression** is a widely used algorithm to post-process the detections. The algorithm repetitively selects the prediction with highest confidence and delete's all other predictions with the same label and an $IoU > a$. When finished, no predictions with $IoU > a$ of same class will exist. This is used since predictions that overlap too much are often duplicates.
Chapter 5

Implementation

In this chapter the implementation of the two different methods will be described in detail. The dataset used will also be covered.

The main libraries and tools used for implementation are OpenCV [17] and scikit-image [25] for image filtering and segmentation operations, Tensorflow [26] and keras [12] for deep learning tasks and LabelImg [13] was used to annotate the data.

5.1 Assumptions

This is a specific problem to solve. To make the final solution better, some assumptions on the data were made.

The sizes of the images are 610x350 pixels. The text will always be horizontal and have a max tilt of 25 degrees. The aspect ratio for a character can be assumed to be between 1:1 and 1:2 (accounting for tilted characters). The maximum height of a character is 120 pixels and minimum is 20 pixels.

5.2 Classic OCR version

This section describes how characters are extracted and classified from an image. The system flow can easily be divided into four stages. Pre-processing, region of interest (RoI) extraction, RoI classification and post-processing. The input of the system is a grayscale image, and the output is boundingboxes with classifications. See Figure 5.1 for an overview of the system architecture.

![Figure 5.1: An overview of the system architecture.](image)

In the following subsections the different stages are explained in more detail.

5.2.1 Pre-processing

The purpose of the pre-processing stage is to segment the image into background and foreground. Namely convert it into a binary image where characters are black and background is white.
The images provided can be very hard to segment and require a few steps. First an adaptive histogram equalization with contrast limiting (CLAHE) is applied to the image according to [28]. This will increase the contrast in the image making characters more distinct which is shown in Figure 5.2 (b). Since the characters are visible due to shadows they tend to be darker than the background. That is why a minimum filter is then applied to the image to enhance the darker regions as seen in Figure 5.2 (c). After that, the image is converted to binary using a local threshold function. A local threshold is used due to shadows and lighting variations in the images. The images can be of different difficulty (e.g. shadows, rugged background and depth of marks), through experiments it was shown that different threshold functions worked best for different types of difficulties. Due to its modifiability, phansalkar was chosen and the parameters were set based on mean value of pixel intensities. A result can be seen in Figure 5.2 (d).

Since the background can be so rugged, a lot of noise can remain after the thresholding. The noise is often slightly smaller than the characters and most of it can therefore be removed using opening and top-hat filters. Every image is different and thus require different parameters for noise removal. It was shown that the results could be improved by basing the parameters on the mean value of the pixel intensities of the binary image and reversing the noise removal if too much of the objects was removed. See Figure 5.2 (d) and (e) for result.

Lastly an closing filter is applied to the image. This is due to characters are often split after the thresholding and noise removal. See Figure 5.2 (f) for effect.

![Figure 5.2: The different stages of image pre-processing.](image)

### 5.2.2 RoI extraction

From the binary image provided from the pre-processing, this stage will generate regions that likely contain a character. This is done in a few steps.

1. Due to many split characters, closing morphism with a horizontal kernel is applied to merge characters from each row. See Figure 5.3 (c) for example.

2. Cut out each found row from the image before horizontal opening was applied. e.g. figure 5.3 (d) is cut out from top left part of Figure 5.3 (b) using the region found in Figure 5.3 (c).

3. For each row, run vertical dilation to merge split characters but keep them separated from other characters. Figure 5.3 (e) is Figure 5.3 (d) after vertical dilation.
4. Save all black areas as regions of interest.

5. Since there still can exist split and merged characters, each region found will be either split in width or widened if it is considered to be too wide or thin.

![Figure 5.3: The different stages of RoI extraction.](image)

Since assumptions on character sizes can be made, regions that are too small will be deleted to increase efficiency of the system.

The final result of Figure 5.3 (a) can be seen in Figure 5.3 (f).

### 5.2.3 Classification

In this stage each region generated will be classified to one of the available characters or as background. As seen in Figure 5.1, the input image is also directed to the classification stage. That is because the pre-processed image is too modified to recognize the regions as characters. So the regions will be classified from the original.

A small CNN was chosen as the classifier. The network layout consisted of 2 convolution layers, each followed by max pooling with stride 2. After the feature extraction a neural network with three layers operates as classifier. Relu was used in all layers except for the output layer which uses softmax to get a prediction \( \in (0, 1) \) for each class. See Figure 5.4 for a descriptive illustration of the architecture.

![Figure 5.4: The architecture for the classifier of the classic version.](image)
5.2.4 Post-processing

Since it is known that there is no overlapping characters in the image, the results can be improved by filtering some of the found regions. This is done using non maximum suppression, i.e. deleting all found characters that overlap by a certain amount except for the one with highest confidence.

5.3 Deep learning version

The deep learning version is a modified implementation of the architecture called retinanet described in [15]. Instead of implementing it from scratch, the github repository https://github.com/fizyr/keras-retinanet was used as a base for this implementation.

Only one network is needed for the object detection, but it consists of different parts with different tasks. The network takes a grayscale image as input and outputs bounding boxes with corresponding label and confidence. An overview of the network can be seen in Figure 5.5.

![Network Architecture Diagram](image)

**Figure 5.5:** An overview of the network architecture. First 3 feature maps of different scales are obtained in the feature extraction. These are then used to predict bounding boxes and labels for the different objects. Lastly some post-processing is done to enhance the results.
The different parts of the network will be further described below.

### 5.3.1 Feature extraction

The feature extraction is a feature pyramid network as described in Section 4.3. The backbone used was mobilenet [9] (without the fully connected layers) which is a CNN designed to be fast and easily adjustable with the hyper-parameter $\alpha \in (0, 1)$ setting the width of the network. Mobilenet uses depthwise and pointwise convolutional filters to increase computation speed. $\alpha$ is set to 0.25 for a small and fast version. Layer 5, 11 and 13 were extracted to the feature pyramid network from the backbone. The architecture of the backbone can be seen in Figure 5.6.

![Diagram of the backbone architecture](image)

Figure 5.6: An overview of the architecture for the backbone used. Each block also contains batch normalization, and ReLU activation function.

### 5.3.2 Classification

Only the two pyramid levels with the highest resolution are provided to the classification stage. This is due to the sizes of characters are never large enough to be classified on the low resolution feature maps. The classification network is applied to both provided pyramid level to get different scales. This implementation is a one-stage detector, i.e. it uses sliding window with anchors as region proposal method (see Section 4.3 for more detailed description). The anchors used have two different ratios, 1:1 and 2:1. Only those two are necessary since the ratio of the characters are similar.

A region will be approved if the prediction confidence for a character is larger than 50%. The loss function used for classification is focal loss as described by Equation 4.7 due to high class imbalance in the training set. e.g. much more data for background.

### 5.3.3 Bounding box regression

To get a better estimation of where the character is located, bounding box regression is used. The loss function used is $L1$ as defined in Equation 4.8.
A lightweight regression structure was used to improve computation speed. This was chosen since perfect bounding boxes are not that important for the result.

5.3.4 Post-processing

The post processing consists of two different operations. Removing overlapping characters found, and applying the bounding box regression to the anchors. Removing overlaps is done using a modified version of the non maximum suppression algorithm explained in Section 4.3. Since it is known that no characters will overlap, not only the characters of the same label will be deleted if they overlap too much. The \( IoU \) required for an overlap to be deleted is set to 0.3.

To get the predicted bounding boxes, the regression values predicted in the bounding box regression stage are applied to the anchors. That is, the center, width and height of the anchor is adjusted to fit the character better.

5.4 Dataset

The dataset consisted of 180 different images with 20 different classes (background excluded). Each image had on average roughly 22 different characters in them.

The classic version had a classifier for a region containing only one character, therefore the classifier was trained on cropped characters. The deep learning version on the other hand was trained on the whole images.

As seen in Table 5.1, the classes were rather imbalanced which can cause problems when training a neural network. To prevent the system for being over trained on the classes with more data, some actions were made. The deep learning version used the focal loss function to calculate the classification error, which penalizes more on difficult classes (e.g. classes with less examples). For the classic version, it worked to just balance the training set since each character was cropped out from the image.

Since the regions found with the classic version rarely are perfect, the training data was also cropped with random padding ranging from 30% to -15% increase in width and height. Two different background classes were also used for the classic version. One for arbitrary background, and one for when parts of characters were visible, but not a distinct class could be decided.

Variance in the images was the top priority when collecting data. That is, different angles, shadows, lighting reflections and types of metal was preferred. The serial numbers available for data collection were limited, which resulted in imbalanced classes.

| Class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | c | d | i | n | p | s | t | z | / |
| Num   | 336 | 383 | 217 | 410 | 259 | 477 | 127 | 69 | 152 | 336 | 452 | 78 | 222 | 78 | 78 | 133 | 125 | 42 | 42 | 42 |
Chapter 6

Evaluation

In this chapter the results of the two different implementations from Chapter 5 will be presented. First the different evaluation metrics will be discussed, then the result will be presented.

6.1 Metrics

As mentioned in Section 2 four different metrics will be considered for evaluation. Average time consumed, percentage of fully correct images, precision and recall. Here they will be explained in more detail.

First we define a correct prediction of a single character such that the predicted bounding box and the ground truth bounding box has an \( \text{IoU} \geq 0.3 \) and the label is correct. \( \text{IoU} \) is defined as in Equation 4.10.

To understand the pros and cons of the system we use the metrics precision and recall. Precision is a measurement of how many of the found characters are predicted correctly, and recall is a measurement of how good the system is at finding characters. Precision and recall is calculated using the following equations.

\[
\text{precision} = \frac{TP}{TP + FP} \quad (6.1)
\]

\[
\text{recall} = \frac{TP}{TP + FN} \quad (6.2)
\]

Where

\( TP = \) True Positives(Amount of correctly classified characters),
\( FP = \) False Positives(Amount of wrongly classified characters),
\( FN = \) False Negatives(Amount of characters not found).

For a fully correct predicted image, all ground truth characters and nothing else must be found and classified correctly, i.e. precision and recall must both be 1 for the image in question. For evaluation, the percentage of fully correct predicted images is calculated. We call this metric accuracy and define it as

\[
\text{accuracy} = \frac{\text{fully correct predictions}}{\text{Total amount of images}} \quad (6.3)
\]

6.2 Test setup

To get as honest results as possible, the images were divided into three different sets. Train, validation and test. The train set was used as training data. The validation set was used to evaluate how good the training proceeded
and to know when no more improvements occurred. The test set was used as a final independent set to evaluate the system. Both implementations were tested on the same set of images.

Since a limited amount of annotated data was used, the sets were divided manually to ensure that many different types (tilt, shadows, light reflections, type of metal, etc.) of images were included in each set.

### 6.3 Results

In this section the results for the two different implementations are presented.

In Table 6.1 and 6.2 the precision, recall, accuracy and seconds per image are presented for each version. Figure 6.1 shows a graph with comparisons of the two versions result on the test set. Figure 6.2 shows a graph of the individual characters found with both versions. Lastly a graph in Figure 6.3 shows the accuracy of the systems if some errors were allowed.

#### Table 6.1: The results for each data set using the deep learning version.

<table>
<thead>
<tr>
<th>Deep learning version</th>
<th>precision</th>
<th>recall</th>
<th>accuracy</th>
<th>seconds/image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train set</td>
<td>0.989</td>
<td>0.967</td>
<td>0.738</td>
<td>0.31</td>
</tr>
<tr>
<td>Validation set</td>
<td>0.984</td>
<td>0.956</td>
<td>0.640</td>
<td>0.30</td>
</tr>
<tr>
<td>Test set</td>
<td>0.980</td>
<td>0.932</td>
<td>0.600</td>
<td>0.30</td>
</tr>
</tbody>
</table>

#### Table 6.2: The results for each data set using the classic version.

<table>
<thead>
<tr>
<th>Classic version</th>
<th>precision</th>
<th>recall</th>
<th>accuracy</th>
<th>seconds/image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train set</td>
<td>0.940</td>
<td>0.746</td>
<td>0.292</td>
<td>0.172</td>
</tr>
<tr>
<td>Validation set</td>
<td>0.892</td>
<td>0.652</td>
<td>0.200</td>
<td>0.129</td>
</tr>
<tr>
<td>Test set</td>
<td>0.946</td>
<td>0.745</td>
<td>0.240</td>
<td>0.164</td>
</tr>
</tbody>
</table>
Figure 6.1: In this graph the precision, recall, accuracy and time per image for both versions are displayed. Precision, recall and accuracy are shown in percent, while the time required per image is associated with the right y-axis which represent seconds. The deep learning version is represented by the blue bars, and the classic version is represented by the orange bars.

Figure 6.2: This graph shows for each class how many of all possible characters were found by the two different versions. The Blue bars represent the ground truth (how many characters really existed), the deep learning version is represented by the green bars, and the classic version is represented by the orange bars.
Figure 6.3: In this graph the accuracy for both versions are shown if some characters are allowed to be faulty classified. The deep learning version is represented by the blue bars, and the classic version is represented by the orange bars.
In this chapter some implementation choices, problems encountered and the result will be discussed.

7.1 Implementation

In this section a few implementation decisions will be discussed. Why they were chosen and if it could work better with something else.

7.1.1 Classic version

Two different background classes were used since just randomly choosing an area that is background as training data resulted in too many false positives when the region to classify contained half of two characters.

Both K-NN, SVM and a small neural network were tested as classifiers. SVM and K-NN used HOG features, and the neural network used two convolutional filters as feature extractor. Both SVM and K-NN was faster than the neural network. The average time required to process a whole image using K-NN was 0.13 seconds, and 0.11 seconds for SVM. Using the neural network required 0.16 seconds. The big difference between the classifiers was in precision, recall and accuracy. K-NN achieved an precision and recall of 48%, SVM was a little better with precision and recall of 56%, while a neural network reached 94% precision and 74% recall. Both K-NN and SVM did not classify a single image fully correct. Thus a neural network was chosen as classifier for the classic version. SVM and K-NN could be further experimented with to increase their accuracy, but since the neural network performed well both in accuracy and time, no more time was spent on other classifiers in this project. The main problem was still to segment the characters correctly.

There exists numerous amount of alterations for implementing the character segmentation(other filters, segmentation algorithms, combinations of methods, parameter adjusting etc.), and of course not all could be tested. Thus, the final implementation in this project is possibly not the most optimal approach.

7.1.2 Deep learning version

The backbone used in the feature pyramid network was the smallest version of mobilenet [9]. Any smaller version was never tried due to reasonably good results and time limitations. Designing a new backbone architecture is something that should be tested if lower computation time is required.

The reason a one-stage detector was chosen was due to its time efficiency, and the retinanet that was used as base had shown comparable results to two-stage detectors [15].
The bounding box regression is something that has made the system easier to evaluate since the predicted boxes are more similar to the ground truth. But in a final product, perfect boxes are not necessary as long as their position relative to the other characters can be calculated. Removing this could decrease computation speed even more.

### 7.1.3 Data set

The data set used was rather small since no pre-annotated data existed. This project was not about annotating as much data as possible to get the best results, but to annotate enough data to see if this approach could work as a real product. i.e. a proof-of-concept. The available images of dot peen markings often had similar characters on them(sometimes only one character differed). When choosing what data to annotate, this was taken into consideration to get the most variations possible.

Since images in the data set included similar characters, some images in the validation and test set became very much alike the train set(in the sense of what characters existed and in what order, not shadows, angle, etc.). Could this have lead to overfitting the model? For example if a "d" is often located to the right of an "a", will the model learn that it is more likely to be an "a" if a part of the "d" is visible in the region too? This is possible, and the best way to find an answer is to acquire more data where the characters are randomly positioned.

### 7.2 Problems

Here the main problems encountered are discussed.

#### 7.2.1 Classic version

A very time consuming problem with the classic version was to balance the parameters used in different operations. For example, since all steps in pre-processing depended on each other, it became very difficult to find the best option. Since "fixing" one could "destroy" another. The same problem occurred when choosing which operations to use. One could say that the segmentation of the characters was hard to generalize.

The best way found to dynamically alter some parameters was to use the mean value of the pixel intensities as an indicator of how the image was structured. But in some images where other factors such as curves or dents in the metal existed, the mean value got altered too much. On average the results improved, thus dynamically setting parameters were kept.

Achieving good results with the classic method is possible, but there exists so many choices for the segmentation problem, such that finding a good enough solution can be very time consuming. This gives an indication that using deep learning could be used for finding a solution to the problem faster.

#### 7.2.2 Deep learning version

The hardest part with the deep learning version was to understand all parts of it to be able to modify the base to fit this problem better. The first successfully trained model showed good results in precision and recall, but high computation time. Thus, most time was spent on reducing the computation time without decreasing the other evaluation metrics. It is often the case that using deep learning for image analysis can be computationally expensive.
As mentioned for the classic version, it can be time consuming to find a good solution for the segmentation of the characters. For the deep learning version, the network will find the solution automatically. Still, it can be very time consuming to annotate enough data for the system to be accurate enough.

### 7.3 Result

Figure 6.1 gives the result of both final versions. As it can be seen there, the deep learning version is better in all metrics except for computation time. The main disadvantage of the classic version is recall, which in its turn reduces accuracy significantly. The low recall value is a sign that the character segmentation is not good enough. This was expected behavior as low contrasts and uneven background was mentioned in Section 1.3, and that is why a deep learning version also was tested. The average time required per image is twice as much for the deep learning version.

In Table 6.1 and 6.2 some comparisons between test, validation and training set are put together. As expected, the deep learning version performs best for training data, second best for validation data and worst for test data. This is due to both the training and validation sets have been involved in the training procedure. The test set is not significantly worse than the validation set, and that is a sign that the system is not over-fitted. For the classic version the results differ more between the sets. This could be because the data used for training the classifier is manually cropped out characters, but the final evaluation was made on the full images with regions generated by the segmentation algorithm. Thus the final evaluation is dependent on good region proposals. As seen in Table 6.2, the recall for the validation set is about 10% lower than train and test. This is probably due to it containing more images that it could not segment well. Another effect of its poor segmentation ability in the validation set can be seen in the computation time required. Validation data is on average faster than train and test data. This is most likely due to it not requiring to classify as many regions as the other two. Thus, if the classic version was to have the same recall as the deep learning version, its computation time would also increase.

In Figure 6.2, a comparison of how many of each character is found in the test set is presented. From this it can be seen that even though the classes are unbalanced, there is not much difference in percentage found. The exception is "z" and "/" for the classic version, but since there are such few examples in the test set, the struggle to find these may not be a general concern (missing only one, results in a big drop in percentage).

In a real serial number scanning system it usually exists a finite number of possible outputs which are known. Thus some database matching could be done to verify the correctness of an output. One could even know where and what characters must be missing or has been wrongly classified. That is why a test was made to see how accuracy increased when one or more characters was allowed to be missing. In Figure 6.3, a graph shows the accuracy of the two versions when 0, 1, 2, 3 and 4 characters are allowed to be wrong. From this it can be seen that even if we let four characters be wrong, 20% is still not correct for the deep learning version and over 40% for the classic version. This is an indication that some images are much harder to predict than others. From manually studying the results, it could be seen that the hardest characters to find and classify were the ones with dark shadows on them.

As stated in Section 1.3, the goal of this project was to evaluate if any of these two solutions could be turned into an accurate enough system to work in the industry. From the results given in Section 6.3, we can draw the conclusion that the deep learning version is the better alternative of the two. The classic version is faster, but the deep learning version is still efficient enough to process multiple images in a few seconds on a cpu. For the easier images the system works well, but harder images reduce accuracy significantly. For an increased accuracy, the system must perform better on the harder images. This could possibly be solved by annotating more data of
the types that the system struggles with or setting restraints on the user such as not rotating the camera too much or sometimes having to process for a longer time (i.e. scan the object longer while changing the angle slightly). If an image would be impossible to read, the worst case scenario would be to manually enter the characters which is acceptable if it does not happen too often.
Chapter 8

Conclusions and future work

In this chapter the conclusion of the project is presented and also some things that can be done to further improve the results.

8.1 Conclusion

The goal of this project was to create two proof-of-concept systems for OCR of dot peen markings and evaluating them to assess if it is possible to develop a functional system from. The first version was developed using the classic OCR approach of segmenting the characters in the image and then classifying the found regions. The other one was to use new object detection algorithms using deep learning. This project has shown that better precision, recall and accuracy was achieved when using a deep learning object detector compared to the classic approach. The average time required to process one image with the deep learning version was 0.30 seconds and the classic version required 0.16 seconds per image. Due to the significantly better precision, recall and accuracy achieved in the deep learning version, this is considered to be the best alternative.

The system struggles with the more difficult images. This could be improved by annotating more data of the difficult kind, and setting restraints on usage of the system (e.g. camera rotation) such that difficult images are minimized.

According to the results presented in Section 6.3 and with more training data, a conclusion could be drawn that one should be able to create a good enough system to work in the industry using the deep learning approach.

8.2 Future work

The main improvement to be made on the classic version of the system is to generate more good region proposals. This can be seen since the biggest drawback is in recall which is a measurement of how many characters are found. For the deep learning version, more training data should be added to increase accuracy and a smaller backbone for the feature extraction part of the network should be tested to improve computation time.

As mentioned in Chapter 7, an improvement for both versions would be the ability to check a database for possible outputs. This way some characters do not need to be found in order to know what the final output should be.
Appendix A

Predicted images

A few images with predictions drawn on them is presented below. The images in the left column is from the classic version and the images in the right column is from the deep learning version. Both versions seem to struggle most with the type of images as in Figure A.7, A.8 and A.9.

Some images have been cropped to exclude information not suitable for the report.

Figure A.1

Figure A.2
Bibliography


