Reading barcodes with neural networks

Fredrik Fridborn
Master of Science Thesis in Electrical Engineering

Reading barcodes with neural networks
Fredrik Fridborn
LiTH-ISY-EX--17/5102--SE

Supervisor: Doktorand Felix Järemo-Lawin
ISY, Linköpings universitet
Doktor Erik Ringaby
SICK IVP

Examiner: Docent Per-Erik Forssén
ISY, Linköpings universitet

Computer Vision Laboratory
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Copyright © 2017 Fredrik Fridborn
Abstract

Barcodes are ubiquitous in modern society and they have had industrial application for decades. However, for noisy images modern methods can underperform. Poor lighting conditions, occlusions and low resolution can be problematic in decoding. This thesis aims to solve this problem by using neural networks, which have enjoyed great success in many computer vision competitions the last years. We investigate how three different networks perform on data sets with noisy images. The first network is a single classifier, the second network is an ensemble classifier and the third is based on a pre-trained feature extractor. For comparison, we also test two baseline methods that are used in industry today. We generate training data using software and modify it to ensure proper generalization. Testing data is created by photographing barcodes in different settings, creating six image classes - normal, dark, white, rotated, occluded and wrinkled. The proposed single classifier and ensemble classifier outperform the baseline as well as the pre-trained feature extractor by a large margin. The thesis work was performed at SICK IVP, a machine vision company in Linköping in 2017.
Acknowledgments

I would like to thank my supervisors Felix and Erik for discussions regarding choice of methods and practicalities regarding implementation. In addition, I want to thank Felix for feedback on the thesis work.

I would also like to thank Ola Friman at SICK IVP for productive discussions.

Linköping, Oktober 2017
Fredrik Fridborn
Contents

Notation .................................................................................... ix

1 Introduction ........................................................................ 1
  1.1 Motivation ....................................................................... 1
  1.2 Purpose ........................................................................... 2
  1.3 Problem formulation ......................................................... 2
  1.4 Limitations ..................................................................... 2

2 Theory .................................................................................. 3
  2.1 Barcodes .......................................................................... 3
    2.1.1 Encoding .................................................................. 3
  2.2 Artificial neural networks .................................................. 4
    2.2.1 The neuron .................................................................. 4
    2.2.2 Activations .................................................................. 5
    2.2.3 Loss functions ............................................................ 6
    2.2.4 Optimization .............................................................. 6
    2.2.5 Backpropagation ........................................................ 7
    2.2.6 Data ........................................................................... 7
    2.2.7 Training ..................................................................... 9
    2.2.8 Ensemble methods ...................................................... 12
  2.3 Convolutional neural networks .......................................... 12
    2.3.1 Convolutions ........................................................... 12
    2.3.2 Pooling ..................................................................... 13
    2.3.3 Fully connected layers ................................................ 13
    2.3.4 Spatial dropout ........................................................ 14
    2.3.5 Transfer learning ....................................................... 14

3 Method .................................................................................. 15
  3.1 Data .............................................................................. 15
    3.1.1 Training data ............................................................. 15
    3.1.2 Validation data ........................................................ 16
    3.1.3 Testing data ............................................................. 17
  3.2 Implementation of CNN .................................................... 18
## Notation

### Some sets

<table>
<thead>
<tr>
<th>Notation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Set of non-negative integers</td>
</tr>
</tbody>
</table>

### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>Artificial neural network</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional neural network</td>
</tr>
<tr>
<td>FC</td>
<td>Fully connected</td>
</tr>
<tr>
<td>BN</td>
<td>Batch normalization</td>
</tr>
<tr>
<td>NOR</td>
<td>Normal</td>
</tr>
<tr>
<td>NDA</td>
<td>Dark</td>
</tr>
<tr>
<td>NWH</td>
<td>Light</td>
</tr>
<tr>
<td>UND</td>
<td>Upside-down</td>
</tr>
<tr>
<td>OCC</td>
<td>Occluded</td>
</tr>
<tr>
<td>WRI</td>
<td>Wrinkled</td>
</tr>
<tr>
<td>ALL</td>
<td>All</td>
</tr>
<tr>
<td>TSC</td>
<td>Tuned single classifier</td>
</tr>
<tr>
<td>TENS</td>
<td>Tuned ensemble classifier</td>
</tr>
<tr>
<td>VGG</td>
<td>VGG16-based classifier</td>
</tr>
<tr>
<td>HAL</td>
<td>HALCON decoder</td>
</tr>
<tr>
<td>INH</td>
<td>Inhouse decoder</td>
</tr>
</tbody>
</table>
1

Introduction

1.1 Motivation

Barcodes are representations of data that can be understood by a computer. The barcode is typically a combination of black and white bars usually representing a series of decimal digits, see figure 1.1. Barcodes have become ubiquitous in modern society, and there are a variety of types for different applications. There are several ways for a computer to read them, most commonly using a laser sheet and some trivial image processing. However, many methods perform poorly when the codes are not perfectly visible. This can be due to poor resolution, scene illumination, image noise, physical damage to the code, curvature and camera position. In the case of the laser scanner in the supermarket checkout, this problem can be manually handled. However, in the case of an automated reading system along a conveyor belt, this could mean downtime and require attention from an operator. This problem will be addressed in this thesis.

Figure 1.1: EAN-13 barcode. The outer two bars are edge markers. The decimals below represent the 13 digit number encoded in the barcode. Image source: Wikipedia [24]
1.2 Purpose

The purpose of this thesis is to produce a barcode recognition system that can interpret the encoded number in a picture of a physical barcode - an image recognition task. Many state of the art image recognition systems today make use of data driven machine learning techniques. One of the most successful methods used in image recognition today is artificial neural networks (ANN), in particular the convolutional neural network (CNN). [3] This thesis will examine how well CNN perform in decoding barcodes in terms of accuracy, computational efficiency and speed. Three different networks will be compared against themselves and two baseline decoders. These are an inhouse decoder from SICK and a decoder from the HALCON image processing library. [5] The three networks are a single CNN, an ensemble classifier combining several CNNs and a partly pretrained CNN.

In the information recognition pipeline, the recognition will be preceded by a pixelwise barcode detection system that finds a barcode in an image of a scene. This way, the system proposed in the thesis will be part of a pipeline that can decode the barcode information provided an image of an object labeled with a barcode.

Depending on the application, ANNs may require copious amounts of data if they are to generalize properly [18]. This will not be a problem in this case, as there is software available to generate images of barcodes. This effectively eliminates the problem of annotating data, provided that the system generalizes.

With proper generalization, the process of scanning barcodes becomes more automated, relieving the operator of this burden. In the case of the conveyor belt, this reduces downtime and allows the operator to focus on other tasks.

1.3 Problem formulation

In the development of the system, there are three questions that must be answered.

- Can a CNN be used for barcode decoding in images?
- Can a CNN be trained to satisfactory performance using synthetic images?
- Which of the three CNNs perform best in comparison?

1.4 Limitations

The scope of this thesis is limited to the recognition of barcodes. This means that the output from the system preceding the proposed system will be regarded as given. The reason for this limitation is twofold. Firstly, researching both systems would be infeasible given the duration of the thesis. Secondly, the detector is already under investigation by other parties at Linköping University.

Moreover, this thesis will only look into a limited number of barcode types, namely the EAN-13 barcode. The reason for this is the limited time frame.
2.1 Barcodes

Barcodes are representations of numbers that can be easily understood by a computer. There are a variety of types of barcodes for different applications, for example different appearances and variations in the resolution of the digit representation and different sizes, see figure 2.1 and 2.2. [1]

![UPC barcode](source: Wikipedia [24])  
![RM barcode](source: Wikipedia [24])

2.1.1 Encoding

A barcode typically consists of a start and stop pattern on the edges. In between these patterns each digit has a unique representation of vertical lines. The last digit of many barcodes is usually a check digit that is only used by the barcode reader to validate the reading; it adds no information. In the case of the EAN-13 barcode in image 2.3 it represents 12 decimal digits. Each digit is represented by 7 modules of either black or white lines. In total it has 95 modules. 84 modules make up the digits and 11 modules make up the start and end patterns as well as a middle pattern, denoted by C1, C2, C3. The representation of each digit is displayed above the bars. The encoding is typically different on the left and right hand side of the middle pattern, see e.g. the digit 4. This way, the system knows
in which order it is reading the code. There might also be a variations on odd and even digits, e.g. the two digits 0 on the left hand side. [1]

![Figure 2.3: Decoding of EAN-13 barcode. Image source: Wikipedia](image)

### 2.2 Artificial neural networks

Predecessors of deep learning and ANNs date back to the 1940’s. The name plays on the goal of emulating the human brain. However, during the course of history, the goal of the discipline has developed into not creating a digital brain per se but rather achieving statistical generalization - solving new problems properly given information from solved ones. [3]

The objective of the ANN is to approximate some function $f$ via a set of parameters $\phi$. This can be for example transforming some input $x$ into some space $S$. The parameters are typically unknown beforehand and are decided via an iterative approach to minimizing each parameter’s contribution to the function approximation error. This optimization process is referred to as training and uses training samples and their ground truth - the desired output. [3]

#### 2.2.1 The neuron

The smallest component of an ANN is the neuron or node. In figure 2.4, each circle corresponds to a neuron. Each neuron collects the outputs of previous neurons, much like the dendrites in the human brain. Neurons on the same level form a layer. The number of layers is known as the depth of the network. [3]

![Figure 2.4: Simple ANN](image)

![Figure 2.5: Neuron model. $x_i$ could be a pixel value, $w_i$ are weights, $y$ is output. $f$ activates the sum of the inputs. Image adapted from [10]](image)
connected to all nodes in the next layer. During training, we only present the desired output to the network - we do not care about the intermediary layers as long as the output layer corresponds to the ground truth. This is why they are referred to as hidden layers. In deep learning, we have several hidden layers in succession. [3]

The parameters $\phi$ map the input tensor to the output tensor. Tensors are multidimensional arrays and are a generalization of scalars, vectors and matrices [19]. $\phi$ are typically weights $w$ and biases $b$. Weights can be seen as the arcs that each neuron collects, scaling the output $x$ from a previous node as seen in figure 2.5. The bias is added to the sum of all arcs and the result is passed to an activation function $f$ yielding an output $y$. A mathematical expression is provided in equation 2.1. In the brain, the activation function represents the threshold that the sum must reach before the neuron fires. [3]

$$y = f\left(\sum_i w_i x_i + b\right)$$ (2.1)

### 2.2.2 Activations

The outcome of the network is decided by an ensemble of nodes. Thus it makes sense to control whether or not they should make a contribution, i.e. whether or not they should fire. For example, it could be wise to normalize the output, squashing it to some interval, otherwise smaller contributions could be drowned. This can be done with e.g. the Sigmoid function $f(x) = \frac{1}{1+e^{-x}}$, seen in figure 2.6. Historically, the Sigmoid has been a common choice of activation function. [10] However, one drawback with the Sigmoidal shape is the vanishing gradient problem. The difference between two values on the edges of the Sigmoid is basically zero, effectively eliminating the gradient which leads to poor optimization. [7]

The default recommended activation function for modern ANNs is the rectified linear unit, defined by $f(x) = \max(0, x)$, seen in figure 2.7. As the name implies, the function is nearly linear - we see that the function consists of two linear parts separated by a nonlinearity in $x = 0$. The linear parts have many good properties in the optimization of the parameters. [3]

For multi-class classification problems, like the one in this thesis, the last layer should output some probability of which class the input belongs to. In this case, we want to classify a decimal digit encoded into an image. We will thus have ten nodes, each node representing the probability of each digit 0-9. This is done by allowing each node to represent a class and using a different activation function on the last layer. In this thesis, the softmax classifier has been used, which utilizes the softmax function:

$$g_j(y) = \frac{e^{y_j}}{\sum_k e^{y_k}}$$ (2.2)

In this equation, we take all $j$ elements in the output vector $y$ and normalize over the sum of all elements. This squashes the values in $g_j$ between 0 and 1 in a way that the elements in $g$ sum to 1. This representation has a nice probabilistic
interpretation - the vector now contains probabilities in percentage of each class. Moreover, the exponentiation makes the smaller quantities much smaller so that the more probable classes receive higher score in relation. [9]

2.2.3 Loss functions

We want to express the performance of the network in some way that enables us to optimize the parameters to achieve optimal performance, i.e. minimal loss. The softmax classifier uses the cross-entropy loss function to do this:

\[ L_x = -\log \left( \frac{e^{y_j}}{\sum_k e^{y_k}} \right) \]  

In this equation, we have a loss value \( L \) given an input \( x \) that yields the output \( y \). \( j \) is the index of the correct element in \( y \), recovered from the ground truth in the training dataset. The denominator normalizes over all elements like before in equation 2.2. A probabilistic interpretation of the equation is that we want to minimize the negative log likelihood of the correct answer, i.e. maximizing the positive log likelihood of the correct answer. [9]

2.2.4 Optimization

The loss function is the metric that says how well the network performs with which we will optimize the parameters of our network. But how do we go about minimizing this function? The simplest approach would be to guess all parameters in the system until we find a combination that yields minimal loss. This approach is obviously infeasible given a large system, so we look for more sophisticated methods. [3]

Another approach would be to initialize the parameters of the system, observe the gradient of the loss function and move in the gradient’s direction towards the minimum with some step size \( \alpha \), also referred to as learning rate. Given a large
enough number of iterations, we should theoretically converge to some local or global minimum. Using the gradient to minimize a problem is referred to as gradient descent. [3]

Adam [16] is an optimization method that utilizes a moving average of the gradient and its square to minimize the loss function:

\[
\begin{align*}
m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla \phi_t \\
v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot \nabla^2 \phi_t \\
\hat{m}_t &= m_t / (1 - \beta_1^t) \\
\hat{v}_t &= v_t / (1 - \beta_2^t) \\
\phi_t &= \phi_{t-1} - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{align*}
\]

In this equation, \( \beta_1, \beta_2 \in [0, 1) \) are hyperparameters that determine the decay rate of the moving averages \( m \) and \( v \). The gradient is given by \( \nabla \), \( \alpha \) is the step size, \( \phi \) corresponds to the parameters in the network and \( \epsilon \) is simply a small number that prevents division by zero. The moving average in equations 2.4 and 2.5 is initially biased towards 0, as \( m \) and \( v \) are initialized as vectors of zeroes. This bias is particularly prominent during the first few iterations. To combat this, the authors introduce the bias corrected terms \( \hat{m} \) and \( \hat{v} \):

\[
\begin{align*}
\hat{m}_t &= \frac{m_t}{1 - \beta_1^t}, \\
\hat{v}_t &= \frac{v_t}{1 - \beta_2^t}, \\
\phi_t &= \phi_{t-1} - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{align*}
\]

In equation 2.7 we have introduced the variable \( t \), which simply is the current iteration. After a sufficient amount of iterations, \( \hat{m} = m \) and \( \hat{v} = v \), as \( \beta_1, \beta_2 \in [0, 1) \). It should be noted that Adam does not have an explicit decay rate of the learning rate. However, Adam has a form of automatic step size annealing that comes from the quotient \( \frac{\alpha \cdot m}{\sqrt{v} + \epsilon} \to 0 \) as we approach optima. [16]

### 2.2.5 Backpropagation

As mentioned earlier, all network parameters \( \phi \) together decide the outcome of the network and thus the value of the loss function \( L \). To find each parameter's contribution to the loss function, the chain rule is used recursively starting at \( L \), moving backwards through the network towards the input layer. This process is known as backpropagation, illustrated in figure 2.8. Here, we are looking at an operation somewhere in the network that has two input activations \( x, y \) that form an output \( z \) via some differentiable function, in this case \( f = x \cdot y \). The derivative of the loss value \( L \) with respect to \( z \) is calculated and equations 2.8-2.11 illustrates how the chain rule is applied, backpropagating the loss function's gradient via the local gradients of \( x \) and \( y \). [13]

### 2.2.6 Data

To properly approximate some function \( f \) with parameters \( \phi \), the quality and quantity of the data is paramount. If the quality of your data is poor, for example if you want to classify images of dogs and cats but only have dogs in your
training data, you are bound to generalize poorly. To match human performance, contemporary machine learning algorithms require millions of labeled examples. [3]

Typically, the dataset is divided into distinct subsets used for training, validation and testing. The ratio between training and validation data is typically 4:1, as seen in figure 2.9. The training data is used to train the parameters and the validation data is used to tune the hyperparameters of the network, e.g. learning rate and decay rates. After training and tuning, the performance of the model is evaluated on the test set. [3]

Typically the training data is divided into subsets referred to as mini-batches, seen in the lower part of figure 2.9. The size of the subset is referred to as the batch size. We use the samples in each mini-batch and calculate their gradients to form an average over the entire subset. This means three things. Firstly, we approximate the gradient of the entire training set with the mini-batch gradient. Thus, if we increase the batch size, we will have a more accurate approximation of the entire training set. Secondly, we eliminate the problem of noisy gradients
from noisy samples. Thirdly, modern processors can crunch several samples in the same time as a single sample due to hardware accelerated parallelism thus shortening training time. [7]

2.2.7 Training

At training time the network is presented with the training data and attempts to minimize the loss function. There are two distinct phases during training: the forward- and backward pass. In the forward pass, a sample is presented to the network and each node calculates its output as well as the local gradient. The backward pass performs backpropagation, recursively applying the chain rule. This eventually results in a Jacobian matrix informing the network of how each parameter should be updated in order to minimize the loss. After updating, the forward pass starts again with a new sample. [13] When all mini-batches have been presented to the network, an *epoch* has passed and the network typically trains for a set number of epochs or until a metric has plateaued. [12]

Overfitting

During training, there are two metrics that are typically monitored - the loss value and accuracy. The loss value is defined in equation 2.3 and the accuracy is simply how many correct answers the model has. These two metrics are continually evaluated for the training data and calculated after each epoch for the validation data. The relation between training and validation accuracy is important to track. If the training accuracy is much higher than the validation accuracy, the network is *overfitting* the data. This is because the model has learned parameters that closely matches the training data but not the validation data, preventing proper generalization. This is shown in figure 2.10, where the blue line corresponds to the training accuracy and the dashed lines correspond to the validation accuracy. The model that yields the green line performs well whereas the other model overfits the training data. To reduce overfitting we must regularize the model. [11] Regularization is defined as anything we do to reduce the generalization error but not the training error. [3]

Parameter initialization

Before training, the parameters $\phi$ must be carefully initialized, otherwise the optimization process might never converge. We want an initialization method that is both good and fast, e.g. drawing the weights from random distributions. [3] The method used in this thesis is Glorot initialization, which initializes weights using the uniform distribution:

$$W \sim U \left[ -\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}} \right]$$

In this equation, $n_j$ is the size of layer $j$ and $U$ is the uniform distribution. Biases $b$ are initialized as 0. [2]
Figure 2.10: An illustration of overfitting. The green validation accuracy is close to the blue training accuracy whereas the red validation accuracy has a larger offset. This implies that the red curve is overfitting on training data.

**Dropout**

Dropout is a regularization method that prevents overfitting by randomly disabling nodes in a layer during training by a chance $p$. The nodes are disabled momentarily, effectively removing their input and output arcs from the model. This prevents co-adaption and forces the nodes in the network to extract very general features. By randomly removing nodes, we train several smaller models which together forms a more powerful classifier. Figures 2.11-2.12 show dropout applied to a simple ANN. [21]

![Figure 2.11: Nodes in an ANN without dropout.](image1)

![Figure 2.12: Nodes in an ANN with dropout.](image2)

**One-hot encoding**

The softmax classifier outputs a vector of probabilities for all the classes. The training data must match this and is therefore encoded into a one-hot represen-
tation, replacing the class label by a vector. The vector is of equal length to the number of classes. The element with the index that corresponds to the class label is replaced with a one while the rest of the elements is set to zero. In this paper, the term one-hot representation is defined as a matrix where the rows are one-hot encoded vectors, essentially a set of vectors stacked on top of each other, see figure 2.13.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{example.png}
\caption{Example of one-hot encoding of a digit vector.}
\end{figure}

**Batch normalization**

Batch normalization (BN) is a method that normalizes the inputs to layers which leads to faster training and allows higher learning rates. It also makes the importance of a good initialization less important as well as having a regularizing effect. The BN algorithm has four steps. First, we calculate batch mean and variance:

\begin{align}
\mu &= \frac{1}{m} \sum_{i=1}^{m} x_i \\
\sigma^2 &= \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)^2
\end{align}

(2.13)  
(2.14)

After this, we normalize all inputs with the batch mean and variance. $\epsilon$ is a small number that prevents division by zero.

\[ \hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \]  
(2.15)

Finally, we multiply with a scaling factor $\gamma$ and add a translation factor $\beta$.

\[ y_i = \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i) \]  
(2.16)

The BN algorithm adds two new parameters $\gamma$ and $\beta$ that are to be trained and backpropagated through. By doing so, the model can learn to undo the normalization, if that means the loss function will be minimized. [7]
Max norm regularization

Another way to regularize the model is via max norm regularization. The norm of the weights of the hidden nodes are clamped to a maximum value, preventing them to become too large. [21]

2.2.8 Ensemble methods

Ensemble methods are a way of decreasing the generalization error of your model. The concept is simple - train several models independently and during inference, present all models with the sample and average their output for a more accurate prediction. [3] In the case of classification, using an ensemble classifier can boost the accuracy of the ANN by a few percent at the cost of more compute as all models must predict an outcome thus leading to slower inference time. [12]

2.3 Convolutional neural networks

For the last couple of years, CNNs have been very successful in solving machine learning problems on images. Neuroscience is credited with the inspiration for the network architecture, providing insights into how the brain processes visual input. Images are processed in steps, with simple cells detecting basic shapes and more complex cells detecting patterns. [3] For machine vision, CNNs has removed the need to engineer features ad-hoc, greatly facilitating development of machine learning models. By letting a CNN extract relevant features with its special architecture, practitioners can address other problems. [14]

2.3.1 Convolutions

Like the name implies, CNNs utilize the discrete two-dimensional convolution:

$$ (F * X)(i, j) = \sum_{w} \sum_{h} X(i - w, j - h)F(w, h) \quad (2.17) $$

In this equation, $X$ is an image with pixel coordinates $i$ and $j$. $F$ is a filter kernel with coordinates $w$ and $h$. The output of the convolution is referred to as a feature map. Figures 2.14 - 2.15 illustrate a convolution of an image in green with a filter in blue, yielding a yellow feature map. The size of the filter is 2x2 and is referred to as the kernel size or filter size. The kernel size defines how much of the input that the nodes will see. This is referred to as the receptive field. For example, a kernel with size 5x5 on the first layer means that the nodes in the first layer will have seen more of the input image than a kernel with size 3x3. It has a larger receptive field. The filter moves across the image one step at a time. This is referred to as filter stride. [3]

In this convolutional layer, the learnable parameters $\phi$ are the filter parameters $t, u, v, w$. The nodes in the convolutional layer can be seen as each entry in the feature map, multiplying some local region in the input by the filter kernel. The
number of parameters needed are reduced not only by this local connectivity but also because they are shared over the entire feature map. [3]

For a classification task, we train our network to extract information that is useful for our specific purpose, typically edges. By having several filter kernels, we can learn to extract different types of information from the image. Each filter kernel yields a feature map that are stacked along the depth. After the convolution, the feature maps are passed through a nonlinear activation function, e.g. the rectified linear unit. [3]

2.3.2 Pooling

Following the nonlinearity, the feature maps are subject to pooling, which downsamples the feature maps. A small region in each feature map is replaced by some statistical representation, e.g. the maximum value of the region. This is referred to as max pooling. The size of the region is referred to as the pool size. By utilizing pooling operations we make the model more invariant to spatial translation - the position of some feature we want to detect is less important than the feature itself. [3]

In this thesis, a convolutional block is a combination of convolutions, activations and pooling operations.

2.3.3 Fully connected layers

After a series of convolutional blocks, we have a tensor containing the extracted features. By flattening this tensor, we end up with a feature vector containing information about the extracted features in the image. We then use standard fully connected layers and transform the feature vector to the output vector. Dropout may be applied to these layers.
2.3.4 Spatial dropout

In the convolutional block, standard dropout will cancel out random entries in the feature map. However, the very similar neighbouring entries might not be dropped out, which counters the purpose of reducing co-adaptation of neurons. To remedy this, we employ spatial dropout. Instead of dropping individual neurons we now drop out entire feature maps, properly regularizing the model. [22]

2.3.5 Transfer learning

An CNN can have several million parameters and training these usually require large datasets. Transfer learning is a way to work around this. By utilizing a network pretrained on a large dataset, e.g. ImageNet [6], it is possible to reduce problems related to data deficiency. One can either use the weights as a better initialization than random values or simply use the convolutional blocks to extract features. [15] There are a variety of pretrained networks available online and VGG16 from Visual Geometry Group at Oxford is one of these. It has shown state-of-the-art results in computer vision competitions. [20]
3.1 Data

The CNN was trained using synthetic data that was produced for this thesis. Validation and testing was done on both synthetic images and real world images of barcodes under various conditions.

3.1.1 Training data

The data was generated using the open source library pyBarcode [23]. A random 12 digit number, leading zeros allowed, was created and used to generate a barcode, the last digit being a checkdigit provided by the library. The barcode is seen in image 3.1a. To reduce size, the barcodes were converted to a grayscale format stored as 32 bit floats. The final dimensionality is $100 \times 196 \times 1$. The corresponding 12 digit number was one-hot encoded. The digits of the barcodes is thus defined by the random seed that yields the 12 digit number. There are no duplicates.

Data augmentation

As we want our synthetic data to generalize properly in our target domain, data augmentation is paramount. We must handle both small perturbations in spatial location, rotation as well as shifting light conditions and occlusions. In addition to this, we must handle fully rotated barcodes. To achieve this, we augment the training data by applying translations and rotations. Occlusions are simulated by randomly dropping out segments of the barcode and we also add noise and Gaussian blur.

As figure 3.1a-3.1d shows, the digits below the barcode is removed. If the digits are not removed, we notice that the model learns to read the characters
3.1 Method

Figure 3.1: Excerpt from the training dataset. (a) shows a raw barcode, (b) shows a standard modified barcode, (c) shows a lighter and partially occluded barcode, (d) shows a darker barcode with low contrast.

instead of the barcode, which was undesirable. It could be of interest to fuse the results from the model and simple optical character recognition, creating a stronger model.

We translate the image $\pm 5$ pixels and rotate $\pm 2^\circ$ by sampling normal distributions with the corresponding variances. We also perform full rotations on 50% of the barcodes, turning the barcode upside down to simulate reading an object upside down. A random portion of the barcode is removed to simulate occlusions. 30% of the barcodes have their overall brightness increased and 30% have their overall contrast reduced. In total, the training dataset consists of 200000 samples.

3.1.2 Validation data

Validation data is acquired by photographing barcodes printed on paper. The camera used is a Point Grey Grasshopper3 provided by SICK IVP. After collection, the images are cropped and scaled to $100 \times 196 \times 1$. We simulate poor lighting conditions by changing aperture settings and simulate occlusion by covering pieces of the barcodes. Some images are fully rotated and some papers are wrinkled to simulate uneven surfaces. Table 3.1 specifies the details of the dataset. Figure 3.2a-3.2f shows an excerpt from the validation dataset.
Table 3.1: Specification of the validation dataset.

<table>
<thead>
<tr>
<th>Dataset conditions</th>
<th>Abbreviation</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>NOR</td>
<td>57</td>
</tr>
<tr>
<td>Dark</td>
<td>NDA</td>
<td>34</td>
</tr>
<tr>
<td>Light</td>
<td>NWH</td>
<td>35</td>
</tr>
<tr>
<td>Upside-down</td>
<td>UND</td>
<td>35</td>
</tr>
<tr>
<td>Occluded</td>
<td>OCC</td>
<td>31</td>
</tr>
<tr>
<td>Wrinkled</td>
<td>WRI</td>
<td>32</td>
</tr>
<tr>
<td>All</td>
<td>ALL</td>
<td>224</td>
</tr>
</tbody>
</table>

Table 3.2: Specification of the testing dataset.

<table>
<thead>
<tr>
<th>Dataset conditions</th>
<th>Abbreviation</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>NOR</td>
<td>46</td>
</tr>
<tr>
<td>Dark</td>
<td>NDA</td>
<td>33</td>
</tr>
<tr>
<td>Light</td>
<td>NWH</td>
<td>35</td>
</tr>
<tr>
<td>Upside-down</td>
<td>UND</td>
<td>25</td>
</tr>
<tr>
<td>Occluded</td>
<td>OCC</td>
<td>32</td>
</tr>
<tr>
<td>Wrinkled</td>
<td>WRI</td>
<td>32</td>
</tr>
<tr>
<td>All</td>
<td>ALL</td>
<td>203</td>
</tr>
</tbody>
</table>

Figure 3.2: Excerpt from the validation dataset. We have: (a) NOR, (b) NDA, (c) NWH, (d) UND, (e) OCC, (f) WRI.

3.1.3 Testing data

Testing data is collected in the same way as the validation data, creating two disjoint subsets consisting of barcodes that completely represent all cases. Table 3.2 specifies the details of the testing dataset. Figure 3.3a-3.3f shows an excerpt from the testing dataset.
3.2 Implementation of CNN

The CNN proposed in this thesis is inspired by work by Goodfellow et al. [4] and Jaderberg et al. [8]. [4] perform digit recognition on street numbers and [8] perform natural text recognition with a similar architecture but using synthetic data. Both achieve excellent results on their datasets. Figure 3.4-3.5 illustrate the similarities between the architecture from [4] and the one proposed in this thesis.

Given the nature of the problem formulation in [4], they must take account for a varying number of digits in the street numbers. Because of this, they must post-process to find the most probable house number based on the digit predictions of each classifier together $H_2 - H_6$ with a prediction of how many digits the number contains - $H_1$. Similarly to [8], we have prior information about the problem. In our case, we know the number of encoded digits which relaxes the problem and eliminates the need of post-processing. Moreover, [8] achieve very good results with synthetic training data, which is reassuring as our training set is synthetic.

Feature extraction

The CNN consists of four subsequent blocks that extract deep features, each block consisting of convolutional, activation and pooling layers. The convolutional layers consists of 32, 64, 128, 256 filters respectively and are activated by rectified linear units. The kernel stride is 1. The activation map from the last convolution in the block is passed into a max pooling layer with a pool size of $2 \times 2$. Each activation is preceded by a BN layer. The kernel size is $3 \times 3$ or $5 \times 5$, specified in chapter 4.

Top classifier

The output of the last block is flattened and passed into FC layers with 4096 and 130 nodes respectively, activated by rectified linear units. The output of the last FC layer is reshaped to a $13 \times 10$ matrix and activated by a softmax function. The
output can be interpreted as 13 classifiers giving class scores for 10 classes, with each classifier handling one decimal digit encoding in the image. The spatial location of the digits is thus encoded into the FC layers.

**Figure 3.4:** Architecture in [4]. 
$H_2 - H_6$ predicts digits and $H_1$ predicts sequence length. Input image source: Goodfellow [4]

**Figure 3.5:** Proposed architecture in thesis. $H_1$ contains the first 10 element of $H$, $H_2$ the next 10 etc.

### 3.2.1 Training

A variety of models were tested, consisting of different number of convolutional layers per block, different kernel sizes for the first two blocks and different number of FC layers. The tests were performed on the ALL dataset. The model with the highest validation accuracy was selected.

After the model architecture had been chosen, test with different dropout settings were conducted to combat overfitting. Different combinations of spatial and standard dropout were tested for all datasets. The best architecture for each dataset was selected. Finally, each model was subject to hyperparameter tuning, validated on its corresponding dataset. The hyperparameters were the optimizer parameters and the maximum norm as defined in sections 2.2.4 and 2.2.7. The hyperparameters were found via random grid search. The architecture with the highest validation accuracy was selected. [12]
3.2.2 Testing

The models are evaluated using the test dataset specified in section 3.1.3. For each model, four metrics were computed - digit and barcode accuracy and inference time on GPU and CPU. The digit accuracy is the percentage of digits that are correct. The barcode accuracy is the percentage of barcodes with all information carrying digits correct - the check digit is allowed to be incorrectly classified as it holds no information. The baseline methods only provided barcode accuracies and are thus not included in the digit accuracy tests. For all three ANNs, inference time was the prediction time after the model had been loaded on GPU or CPU memory, i.e. only the matrix multiplications were timed. For the last two methods, inference time was the time spent decoding after relevant parameters were set. Neither of these had software that were GPU accelerated therefore leaving out that metric.

3.3 Comparisons

In this thesis, four different methods of reading barcodes have been tested. The comparison was done on the ALL testing set, see table 3.2.

3.3.1 Single classifier

The single classifier consisted simply of the model trained in section 3.2.1 that had the highest validation accuracy on ALL.

3.3.2 Ensemble classifier

Similarly to the single classifier, the ensemble classifier was built with the models with highest validation accuracy on each category except for ALL. The classifiers performed voting on which class that was most likely. For each digit, the mean of all predictions decided which digit was selected. Using the maximum prediction was also tested with the identical outcome of the voting.

3.3.3 VGG classifier

To see whether or not training on synthetic images of barcodes actually improved performance, a comparison was made with a network trained on ImageNet. VGG16 [20] was used for this purpose. The top classifier was removed, leaving only the convolutional layers that extract deep features. VGG16 takes 3-channel data, so the single channel of the training dataset was tripled to a final dimensionality of $100 \times 196 \times 3$. After feature extraction, the activation map was flattened and fed into the top classifier specified in section 3.2.
3.3.4 HALCON decoder

To compare with an established method of reading barcodes, the EAN13 barcode reader from the HALCON image processing library was used, a decoder that has industrial applications today [17]. Several measures were taken to ensure optimal performance of the barcode reader. Firstly, the input image was normalized and was preprocessed to enhance the contrast. Secondly, if the barcode was not properly decoded by the standard decoder, a closer examination was performed by attempting to find a bounding box before decoding. Finally, the threshold of the barcode reader was tuned by testing a large number of possible values, keeping the best threshold. The tuning was done on the validation dataset and the performance was evaluated on the test set using the best threshold for that set.

3.3.5 Inhouse decoder

To compare with another method, the SICK algorithm for reading EAN13 barcodes was also tested. An application expert was given the test sets and returned the accuracies for each dataset.

It should be noted that both the HALCON decoder and the inhouse decoder do not only attempt to decode the barcode, it also attempts to find it in the image. Both approaches thus solve a more difficult problem. Moreover, the inhouse decoder is not adapted to these kinds of images - it is used on images taken by the decoder itself. The decoder API does not allow any parameter tuning of the decoding process, only for detection of the barcodes.

3.4 Tools

The system was developed using the programming language Python. The API Keras and the deep learning library TensorFlow were used to facilitate implementation of ANNs. The library pyBarcode [23] was used to create the barcodes that were used in training. The HALCON library was version 13.0.

The system was implemented in Windows 10 on a PC with a 3.60 GHz Intel core Xeon E5-1620, 16 GB RAM and NVIDIA GeForce GTX 1080 Ti. All tensor operations were accelerated by the GPU.
4

Results

4.1 Implementation of CNN

As mentioned in section 3.2, a variety of models were tested. The models are presented in table 4.1. The filter size of the first layers were varied to investigate how a larger receptive field impacts the model accuracy. The number of convolutional layers were changed to see how deeper features (models 1-4) would compare to more shallow features (models 5-8). Each convolutional layer were preceded by a BN layer and succeeded by a pooling layer. This means that the blocks in models 1-4 downsample the inputs more than models 5-8, effectively reducing the number of parameters. The number of FC layers were varied to examine if a deeper top classifier improved accuracy.

<table>
<thead>
<tr>
<th>Model name</th>
<th>Conv layers per block</th>
<th>Kernel sizes per block</th>
<th>FC layers</th>
<th>Parameters $[10^6]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN1</td>
<td>2</td>
<td>5x5x4, 3x3x4</td>
<td>4096x2</td>
<td>25.934</td>
</tr>
<tr>
<td>CNN2</td>
<td>2</td>
<td>5x5x4, 3x3x4</td>
<td>4096x1</td>
<td>9.140</td>
</tr>
<tr>
<td>CNN3</td>
<td>2</td>
<td>3x3x8</td>
<td>4096x2</td>
<td>35.281</td>
</tr>
<tr>
<td>CNN4</td>
<td>2</td>
<td>3x3x8</td>
<td>4096x1</td>
<td>18.501</td>
</tr>
<tr>
<td>CNN5</td>
<td>1</td>
<td>5x5x2, 3x3x2</td>
<td>4096x2</td>
<td>59.709</td>
</tr>
<tr>
<td>CNN6</td>
<td>1</td>
<td>5x5x2, 3x3x2</td>
<td>4096x1</td>
<td>42.915</td>
</tr>
<tr>
<td>CNN7</td>
<td>1</td>
<td>3x3x4</td>
<td>4096x2</td>
<td>59.675</td>
</tr>
<tr>
<td>CNN8</td>
<td>1</td>
<td>3x3x4</td>
<td>4096x1</td>
<td>42.882</td>
</tr>
</tbody>
</table>

Table 4.1: Specification of the tested CNN models. Where there are two kernel sizes specified, the first kernel sizes refers to the first blocks and the second refers to the subsequent blocks.
The CNN is trained end-to-end using the Adam optimizer from chapter 2.2.4 with $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$. Validation was done on ALL. If the validation loss did not decrease in 3 epochs, the training was aborted and the parameters were saved. The results are presented in figure 4.1. The training time is approximately 3 hours.

![Figure 4.1: Validation digit accuracy for CNN1-8. CNN5 has the highest validation accuracy.](image1)

![Figure 4.2: Model CNN5. The gap between validation and training accuracy implies overfitting.](image2)

4.1.1 Dropout tests

As the figure 4.1 shows, the architecture CNN5 yields the highest validation accuracy. However, figure 4.2 shows that overfitting is a problem. We employ dropout from chapter 2.2.7 and test some different dropout factors, specified in table 4.2. We apply spatial dropout from chapter 2.3.4 in blocks 1-4 and standard dropout in FC layers 1-2.

<table>
<thead>
<tr>
<th>Model name</th>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</th>
<th>Block 4</th>
<th>FC 1</th>
<th>FC 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dropout 1</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>Dropout 2</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>Dropout 3</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Dropout 4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>Dropout 5</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*Table 4.2: Specification of the tested CNN models with listed dropout rates.*

The CNN is trained end-to-end using the Adam optimizer from chapter 2.2.4 with $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$. Validation was done on each dataset. The results are shown below with the best results in bold. The best model on ALL was chosen as the single classifier and the best models for each other category formed the ensemble classifier.
4.1 Implementation of CNN

<table>
<thead>
<tr>
<th>Model</th>
<th>NOR</th>
<th>NDA</th>
<th>NWH</th>
<th>UND</th>
<th>OCC</th>
<th>WRI</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dropout 1</td>
<td>0.960</td>
<td>0.887</td>
<td>0.820</td>
<td>0.954</td>
<td>0.928</td>
<td>0.906</td>
<td>0.926</td>
</tr>
<tr>
<td>Dropout 2</td>
<td>0.953</td>
<td>0.921</td>
<td>0.840</td>
<td>0.949</td>
<td>0.955</td>
<td>0.921</td>
<td></td>
</tr>
<tr>
<td>Dropout 3</td>
<td>0.955</td>
<td>0.878</td>
<td>0.829</td>
<td>0.947</td>
<td>0.958</td>
<td>0.897</td>
<td>0.916</td>
</tr>
<tr>
<td>Dropout 4</td>
<td>0.939</td>
<td>0.887</td>
<td>0.782</td>
<td>0.958</td>
<td>0.938</td>
<td>0.885</td>
<td>0.895</td>
</tr>
<tr>
<td>Dropout 5</td>
<td>0.953</td>
<td>0.885</td>
<td>0.842</td>
<td>0.947</td>
<td>0.955</td>
<td>0.916</td>
<td>0.920</td>
</tr>
</tbody>
</table>

Table 4.3: Specification of digit accuracies for each validation dataset. All models employ dropout specified by table 4.2.

When we plot the accuracy and validation accuracy for the single classifier, we see that we still have a problem with overfitting but the differences between the accuracies are much smaller in figure 4.3 than figure 4.2.

![Figure 4.3: Overfitting for the single classifier. The differences between the accuracies is lower than figure 4.2.](image)

4.1.2 Hypermeter tuning

After the dropout tests, hyperparameter tuning was performed on the best models in accordance with table 4.4. The recommended optimizer settings as defined in [16] were slightly varied. The maximum norm was varied as recommended in [21].
<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Lower bound</th>
<th>Upper bound</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>0.00075</td>
<td>0.00125</td>
<td>Uniform</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>0.85</td>
<td>0.95</td>
<td>Uniform</td>
</tr>
<tr>
<td>$\beta_2$</td>
<td>0.99800</td>
<td>0.99990</td>
<td>Uniform</td>
</tr>
<tr>
<td>Max norm</td>
<td>3</td>
<td>4</td>
<td>Uniform</td>
</tr>
</tbody>
</table>

*Table 4.4: Specification of how hyperparameters were tested.*

Similarly as before, the best model on ALL is labeled the tuned single classifier (TSC) and the best models for each other category formed the tuned ensemble classifier (TENS). The best model parameters are visualized in table 4.5.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$\alpha$</th>
<th>$\beta_1$</th>
<th>$\beta_2$</th>
<th>Max norm</th>
<th>Validation accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOR</td>
<td>0.00080</td>
<td>0.88961</td>
<td>0.99829</td>
<td>3.2360</td>
<td>0.965</td>
</tr>
<tr>
<td>NDA</td>
<td>0.00103</td>
<td>0.94242</td>
<td>0.99889</td>
<td>3.4524</td>
<td>0.937</td>
</tr>
<tr>
<td>NWH</td>
<td>0.00080</td>
<td>0.88961</td>
<td>0.99829</td>
<td>3.2360</td>
<td>0.868</td>
</tr>
<tr>
<td>UND</td>
<td>0.00080</td>
<td>0.88961</td>
<td>0.99829</td>
<td>3.2360</td>
<td>0.974</td>
</tr>
<tr>
<td>OCC</td>
<td>0.00083</td>
<td>0.91509</td>
<td>0.99814</td>
<td>3.3238</td>
<td>0.973</td>
</tr>
<tr>
<td>WRI</td>
<td>0.00116</td>
<td>0.89850</td>
<td>0.99850</td>
<td>3.7933</td>
<td>0.914</td>
</tr>
<tr>
<td>ALL</td>
<td>0.00094</td>
<td>0.86385</td>
<td>0.99965</td>
<td>3.3630</td>
<td>0.932</td>
</tr>
</tbody>
</table>

*Table 4.5: Specification of digit accuracies for each validation dataset.*

### 4.1.3 VGG classifier

The fully connected configuration in SC was used as the top classifier for the features extracted from VGG16 from chapter 2.3.5. The CNN is trained using the Adam optimizer from chapter 2.2.4 with $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. Validation was done on ALL. The resulting model (VGG) was then saved and compared with the others. The model had 55 million trainable parameters.

### 4.2 HALCON decoder

The HALCON barcode reader initially performed very poorly on the images as can be seen in the first row of table 4.6. To make a fair comparison, the edges of the images were extended, as these were found to be the root cause. The new edge values were calculated in two ways - by copying the edge value or by a constant value - in this case the maximum value of the image. Three different barcodes were created - no padding and padding of 10 and 20 pixels. The modified images are shown in images 4.4 - 4.5.

The results are shown in the table below. The digit accuracy of the barcode reader is the same as the barcode accuracy, as there is no way of finding individual digits in the HALCON library. The best results are marked in bold.
Figure 4.4: Constant edge padding.

Figure 4.5: Same edge padding.

<table>
<thead>
<tr>
<th>Padding</th>
<th>NOR</th>
<th>NDA</th>
<th>NWH</th>
<th>UND</th>
<th>OCC</th>
<th>WRI</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0.4</td>
<td>0.061</td>
<td>0.000</td>
<td>0.265</td>
<td>0.300</td>
<td>0.258</td>
<td>0.229</td>
</tr>
<tr>
<td>Constant,10</td>
<td>0.768</td>
<td>0.242</td>
<td>0.206</td>
<td>0.765</td>
<td>0.633</td>
<td>0.548</td>
<td>0.516</td>
</tr>
<tr>
<td>Constant,20</td>
<td>0.786</td>
<td>0.242</td>
<td>0.206</td>
<td>0.794</td>
<td>0.667</td>
<td>0.484</td>
<td>0.511</td>
</tr>
<tr>
<td>Edge,10</td>
<td>0.750</td>
<td>0.242</td>
<td>0.206</td>
<td>0.765</td>
<td>0.633</td>
<td>0.484</td>
<td>0.502</td>
</tr>
<tr>
<td>Edge,20</td>
<td>0.804</td>
<td>0.242</td>
<td>0.176</td>
<td>0.765</td>
<td>0.667</td>
<td>0.516</td>
<td>0.516</td>
</tr>
</tbody>
</table>

Table 4.6: Specification of validation accuracy for the HALCON decoder.

The threshold and padding that yielded the highest validation accuracy (HAL) was then used on the test set. For setups that had the same validation accuracy, only the best was kept. The results are presented in table 4.8.

4.3 Inhouse decoder

The inhouse decoder (INH) provided by the application expert at SICK was applied directly on each test set. The results are presented in table 4.8.

4.4 Testing

The two metrics - digit and barcode accuracy, was calculated for each model and each test set. The baseline methods only evaluate barcode accuracies. The results are shown below with the best results in bold. Inference time is shown in table 4.10. To make a fair comparison, the tensor operations are performed both on CPU and GPU, as neither HAL nor INH utilizes the GPU. To gain some insight, table 4.9 show how many incorrect digits that were behind each incorrectly classified barcode.

<table>
<thead>
<tr>
<th>Model</th>
<th>NOR</th>
<th>NDA</th>
<th>NWH</th>
<th>UND</th>
<th>OCC</th>
<th>WRI</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSC</td>
<td>0.968</td>
<td>0.904</td>
<td>0.870</td>
<td>0.944</td>
<td>0.925</td>
<td>0.950</td>
<td>0.928</td>
</tr>
<tr>
<td>TENS</td>
<td>0.957</td>
<td>0.855</td>
<td>0.848</td>
<td>0.935</td>
<td>0.933</td>
<td>0.955</td>
<td>0.914</td>
</tr>
<tr>
<td>VGG</td>
<td>0.318</td>
<td>0.207</td>
<td>0.224</td>
<td>0.178</td>
<td>0.144</td>
<td>0.178</td>
<td>0.230</td>
</tr>
</tbody>
</table>

Table 4.7: Comparison of models by digit accuracy for each test set.
### Table 4.8: Comparison of models by barcode accuracy for each test set. Models marked with an asterisk attempt to solve a more complex problem - detection and decoding. Moreover, INH is not adapted to these types of images.

<table>
<thead>
<tr>
<th>Model</th>
<th>NOR</th>
<th>NDA</th>
<th>NWH</th>
<th>UND</th>
<th>OCC</th>
<th>WRI</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>0.870</td>
<td>0.606</td>
<td>0.371</td>
<td>0.800</td>
<td>0.625</td>
<td>0.750</td>
<td>0.675</td>
</tr>
<tr>
<td>Ensemble</td>
<td>0.783</td>
<td>0.515</td>
<td>0.400</td>
<td>0.680</td>
<td>0.469</td>
<td>0.689</td>
<td>0.596</td>
</tr>
<tr>
<td>VGG</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.040</td>
<td>0</td>
<td>0</td>
<td>0.005</td>
</tr>
<tr>
<td>Halcon*</td>
<td>0.756</td>
<td>0.406</td>
<td>0.206</td>
<td>0.708</td>
<td>0.645</td>
<td>0.484</td>
<td>0.500i</td>
</tr>
<tr>
<td>SICK*</td>
<td>0.348</td>
<td>0.0</td>
<td>0.057</td>
<td>0.600</td>
<td>0.281</td>
<td>0.219</td>
<td>0.241</td>
</tr>
</tbody>
</table>

### Table 4.9: Number of incorrect digits per incorrect barcode for ALL.

<table>
<thead>
<tr>
<th>Model</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSC</td>
<td>21</td>
<td>14</td>
<td>13</td>
<td>7</td>
<td>9</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TENS</td>
<td>34</td>
<td>14</td>
<td>14</td>
<td>15</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 4.10: Specification of inference time per barcode for all models.

<table>
<thead>
<tr>
<th>Model</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSC</td>
<td>0.073</td>
<td><strong>0.001</strong></td>
</tr>
<tr>
<td>TENS</td>
<td>0.209</td>
<td>0.004</td>
</tr>
<tr>
<td>VGG</td>
<td>0.177</td>
<td>0.002</td>
</tr>
<tr>
<td>HAL</td>
<td>0.020</td>
<td>-</td>
</tr>
<tr>
<td>INH</td>
<td><strong>0.003</strong></td>
<td>-</td>
</tr>
</tbody>
</table>

*Table 4.10: Specification of inference time per barcode for all models.*
5

Discussion

5.1 Results

In these tests, the CNN approach seems to have outdone the traditional barcode readers if we look at tables 4.7 - 4.8. It should also be noted that if the unpadded images were to be used for HAL, the differences would have been even greater.

TENS and TSC have similar digit accuracies - TSC is ahead by a small margin for some test sets and TENS is ahead for others. However, the barcode accuracy was dominated by the TSC, where TENS underperformed. This can also be seen in table 4.9 where we see that TENS had a much larger portion of single digit errors whereas one of the TSC predictions misclassify 8 digits out of 13. The classifiers in the ensemble are unanimous for all digits except one and the voting procedure in this case actually interferes with the overall outcome.

VGG performs extremely poorly in all cases, on par with the unpadded HAL and INH. A possible explanation for this is that the feature extractor was trained on ImageNet, a dataset containing millions of mostly natural images, see figure 5.1. Barcodes have a very synthetic appearance - black and white, straight edges and a repetitive pattern of rectangles. It is possible that the set of features describing these patterns are not well represented in the set of features found by filter kernels trained on natural images. The very same top classifier performs well given the features from filters that are taught on the synthetic images, which supports this explanation.

In the NWH category, TSC has roughly twice the barcode accuracy compared to HAL and almost ten times more compared to INH, suggesting that the network has learned some more advanced normalization steps than the standard contrast enhancement that was performed as a preprocessing step for HAL and INH.

We noticed that HAL beat TENS and INH by several percentage points and perform similarly to TSC in the OCC category, implying that the model failed in
Figure 5.1: (a)-(b) are generic ImageNet images. Image source: ImageNet [6]. (c) is a barcode from the training set and (d) is from the validation set.

Learning the 1D representation of the code. Initial tests implied that this could be a problem and measures were taken by removing larger strips of the barcode but to no avail. However, inspecting the images from the OCC category we notice that they generally are sharper compared to other categories, see figures 5.2-5.3. The reason for this is most likely different conditions during data acquisition.

Figure 5.2: Sample from the Nor validation set.  

Figure 5.3: Sample from the OCC validation set.

Regarding inference times in table 4.10, INH and HAL greatly outperformed all CNN approaches. Large matrix operations are notoriously slow on CPU and inference with a 55 million parameter CNN is no exception. Had this been implemented on an embedded device, however, the matrix calculations would most likely be hardware accelerated by e.g. FPGA.

5.2 Method

Why did HAL and INH do poorly? The results in tables 4.7 - 4.8 implies that the CNN approach greatly outperforms the baseline methods. Time spent honing
the CNNs overshadow the time spent tuning the baseline by orders of magnitude, leading to a somewhat uneven comparison. In addition, both methods solve a harder problem - both detection and decoding.

For HAL, low resolution seems to have been problematic. We realize this by using identical test settings but upsampling the NOR test set by a factor of two. The barcode accuracy increases from 0.756 to 0.826, implying that the CNN approach is more resilient to lower resolutions.

For INH, the decoder is simply not adapted for these kinds of images - it is used on images that the decoder captures. In production, a misread barcode can for example result in a parcel ending up in the wrong end of the world which leads to considerable economical repercussions. Therefore if the proposed image is of such poor quality that the decoder cannot provide an accurate assessment, it is better to disregard the barcode as incorrect and alert an operator to ensure proper handling. The version of the decoder used in this thesis is different from the one used in production and has fewer tuning parameters. For example, there are no parameters that adjust the decoding - only the detection. For this reason, the comparison of INH and HAL is suffering - the HAL has parameters that allow for this kind of tuning.

For this thesis, all datasets were constructed from scratch. Having constructed both validation and test set, one must show restraint and vigilance to avoid using prior information about the test set to make decisions regarding the architecture of the network. Ideally, test sets should be kept secret from the practitioner until the final architecture has been decided rather than gaining insights about the test set during construction of the datasets. For this thesis, the only interaction with the test set was during construction and final evaluation.
6 Conclusions

6.1 Research questions

In the introduction, three questions were posed:

- Can a CNN be used for barcode decoding in images?
- Can a CNN be trained to satisfactory performance using synthetic images?
- Which of the three CNNs perform best in comparison?

The results show that a CNN performs on par or better than a standard barcode reader in decoding so the answer to that question is yes. It was done so by training on a dataset consisting of only synthetic images, so the answer to the second question is yes. Regarding the third question, a single classifier validated on all of the datasets outperforms the others in both inference time and accuracy. The standard decoding methods proved to perform poorly given the datasets but still made a fair baseline and did not undermine the conclusion.

6.2 Future work

For the EAN-13 classifier, we have 13 classifiers that create a 13 digit number. Using the same architecture and just changing the number of classifiers, it would be of interest to decode other barcodes and see how well the method performs. Initial tests on EAN-8 and Code39 barcodes show promising results.

Today, the ensemble classifiers perform votes after the softmax function. One method to achieve higher accuracy would be to average their values before the softmax, otherwise the smaller probabilities are squashed to zero.

It would also be of interest to utilize the digits beneath the barcode and fuse the predictions together forming an even stronger classifier in the process.


