A Deep Learning Prediction Model for Object Classification

N.E. Sahla

Master of Science Thesis

Delft Center for Systems and Control

A Deep Learning Prediction Model for Object Classification

Master of Science Thesis

For the degree of Master of Science in Mechanical Engineering at Delft University of Technology

N.E. Sahla

February 6, 2018

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by Vanderlande. Their cooperation is hereby gratefully acknowledged.

Copyright © Department of BioMechanical Design (BMD)
All rights reserved.

Abstract

The last decade has marked a rapid and significant growth of the global market of warehouse automation. The biggest challenge lies in the identification and handling of foreign objects. The aim of this research is to investigate whether a usable relation exist between object features such as size or shape, and barcode location, that can be used to robustly identify objects in a bin.

A deep convolutional neural network (CNN) is built in MATLAB and trained on a labeled dataset of thousand product images from various perspectives, to determine on which surface of a product the barcode lies. Training results show that while the training set accuracy reaches 100%, a maximum validation accuracy of only 45% is achieved. A larger dataset is required to reduce overfitting and increase the validation accuracy. When sufficient classification accuracies are reached, smart picking strategies can be implemented to efficiently handle products.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank prof. dr. ir. Martijn Wisse and ir. Jeff van Egmond for their supervision and guidance throughout the thesis study at the Delft University of Technology. At Vanderlande Industries I would like to thank Erik van Dartel for giving me the chance to work on this project, and Marcin Luksa and Djani Burazerovic for their support and guidance at the Vanderlande site. A special shout-out to my friends and family for their continuing support!

Delft, University of Technology N.E. Sahla
February 6, 2018

Chapter 1

Introduction

The last decade has marked a rapid and significant growth of the global market of warehouse automation, driven mainly by the global growth of the e-commerce market. The significant switch from offline shopping to anywhere and anytime online shopping has caused a change in the retail order fulfillment process. This paradigm shift has sparked a widespread growth in warehouse automation systems, with the goal of realizing fully autonomous order fulfillment systems.

1-1 Object Recognition for Warehouse Automation

One of the primary causes for the fast growth of the global market for warehouse automation is the large growth of e-commerce, and the order fulfillment requirements that are created as a result (Fast delivery, track and trace, etc). There is an increased need of systems capable of handling high volumes of multi-item orders.

The biggest challenge in realizing fully autonomous order fulfillment systems lies in the identification and handling of foreign objects. Current image segmentation techniques provide sufficient accuracy to handle foreign objects for pick and place operations. These image segmentation techniques are able to identify and distinguish geometrical shapes, edges, surfaces and other features that are sufficient for robust handling of foreign objects. These features however are insufficient for the identification of foreign objects. This limitation in technology necessitates the presence of a manual worker that provides the identification of foreign objects in an order fulfillment process. For full automation to be realized, the order fulfillment system should be able to autonomously recognize the object that is being handled, so that the object can be transported to its corresponding destination.

1-2 Automation of the Vanderlande Order Picking Process

The current high variability of product demand and saleability do not allow for a traditional ABC zoning strategy[1] in warehouses and thus limit the possibilities of full automation. Instead, picking is performed manually from shelving systems containing randomly placed items. Orders that contain more than one item undergo a two-phase picking process. In the first phase, a manual operator picks products from warehouse shelves, scans and then places the products in a designated tote. Once a tote is full, it is sorted via conveyor belts to a rebin station. The second phase takes place at the rebin station, where individual items are manually scanned by the rebin associate, triggering a light on the rebin wall indicating the chute destination for the item. The rebin associate places the item in the designated chute. Once the chute contains all items for a particular order, the light flashes and the entire order is pushed to the packing side of the chute. A schematic overview of this process is given below in figure 1.

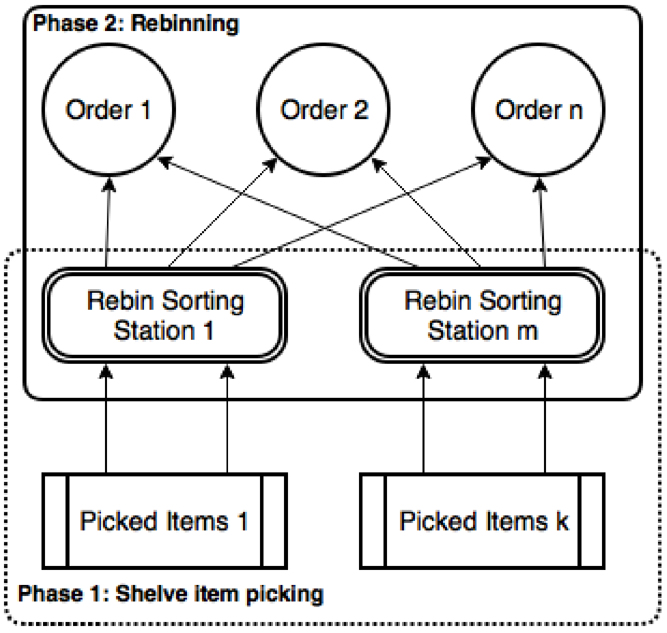


Figure 1-1: Vanderlande order picking process scheme. In phase 1 items are picked from shelves, placed in a bin and transported to a rebin sorting station. In phase two a manual worker scans each item from a bin and places it in its responding order destination.

While Vanderlande Industries does have a variety of machine vision algorithms available for its robotic bin picking solutions, there are no robust solutions available yet for recognition of objects in a bin. The weakness of existing object recognition algorithms lies in the absence of knowledge about the multitude of visual phenomena occurring in the real world, such as occlusion, background clutter and illumination. Unlike humans, computer vision algorithms lack the ability to deal with these visual phenomena without being engineered to deal with them in advance. In order to reach full automation of the Vanderlande order fulfillment cycle,

[1]ABC analysis separates supplies into three categories- "A items" , "B items", and "C items" based on the level of control and record keeping.

phase 2 has to be automated, replacing the manual sorting operator with a robust robotic solution. This robotic system should be able to recognize the objects that are being handled during the Vanderlande order fulfillment cycle.

Designing a robotic system that can recognize and handle foreign objects is difficult due to the multitude of visual phenomena that occur in the real world, such as object occlusion or lighting effects. Making a distinction between two objects that are nearly identical in shape and color, but are completely different items is one of the toughest problems to solve with current computer vision technology. An example of this situation is given in figures 1-2 and 1-3, where two nearly identical looking products still have different contents.

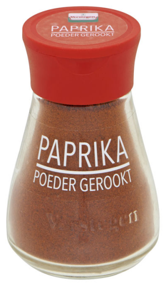


Figure 1-2: Paprika powder 'smoked' version.

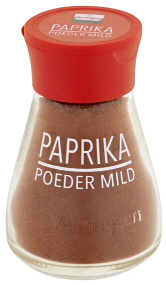


Figure 1-3: Paprika powder 'mild' version. Nearly identical to the 'smoked' version, with a minor difference in texture that is very hard to distinguish with existing computer vision technology.

In order to design a robust order handling system, many of these hard or unusual cases need to be correctly identified. To accomplish this, properties unique to each item should be found and used to classify each item correctly. In some of the most hard cases where almost no visual difference can be found, there still remains a unique property that can be used for correct classification: the barcode. Each item that is processed in Vanderlande warehouses contains a barcode that at some point is scanned by a manual worker to identify its destination during the rebin process. The focus on the new object recognition framework will lie on the use of the barcode as a unique property to correctly identify objects. If the barcode of each object can be located and recognized robustly, each object can be identified and correctly handled. While standard barcode placement guidelines exist, certainly not all items adhere to these regulations. It is very interesting to investigate whether there exists a usable relation between object features, and barcode location. If this relationship exists, it can be used to pick and scan items very efficiently in a warehouse, eliminating the need of human workers for manual object identification. Therefore the main goal of this thesis is defined as follows:

"Does a <u>useful</u> relation exist between object features such as size or shape, and barcode location, that can be used to recognize and handle foreign objects in the Vanderlande order fulfillment cycle."

A machine learning approach is taken to tackle this goal, proposing a novel object recognition framework that should robustly determine the barcode placement pattern.

1-3 Thesis Structure

This thesis report is structured into five chapters. Chapter 2 provides a theoretical explanation of machine learning theory. Chapter three reviews four of the most popular machine learning theories: decision trees, artificial neural networks, support vector machines and k-Nearest-Neighbor classification. Chapter 4 provides a detailed explanation of the proposed object recognition framework. Chapter 5 discusses the implementation phase of the object recognition frameworks, and discusses results of the implementation phase. Lastly, in chapter 6 the conclusion of the project is given and recommendations for further improvements are discussed.

Chapter 2

Machine Learning

In this chapter, an overview is given of the important findings from literature research on machine learning. Many existing machine learning algorithms are described and compared in literature. The global increase of machine learning use and applications raises the question: Which method should be used for a specific application? Exploring these machine learning techniques can give understanding into their usage for specific applications. This can provide the best algorithm for a specific situation or application.

2-1 Machine Learning Definition

Machine learning is a field that is focused on the construction of algorithms that make predictions based on data. A machine learning task aims to identify (to learn) a function $f : X \rightarrow Y$ that maps the input domain X (of data) onto output domain Y (of possible predictions) [1]. Functions f are chosen from different function classes, dependent on the type of learning algorithm that is being used. Mitchell (1997) defines "learning" as follows: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [2]. The performance measure P tells us quantitatively how well a certain machine learning algorithm is performing. For a classification task, the accuracy of the system is usually chosen as the performance measure, where accuracy is defined as the proportion for which the system correctly produces the output. Experience E that machine learning algorithms undergo are datasets. These datasets contain a set of examples that are used to train and test these algorithms.

Classification of Machine Learning Algorithms

Machine learning algorithms can be largely classified into three categories by the type of datasets that are used as experience. These categories are **supervised learning**, **unsupervised learning** and **reinforcement learning**. Supervised learning systems make use of *labeled* datasets $(x, y) \in X \times Y$, where x represents a data point and y the corresponding true prediction for x. This training set of input-output pairs is used to find a deterministic function that maps any input to an output, predicting future input-output observations while minimizing errors as much as possible. Unsupervised learning systems use *unlabeled* datasets to train the system. The objective of unsupervised learning is to derive structure from unlabeled data by investigating the similarity between pairs of objects, and is usually associated with density estimation or data clustering. Reinforcement learning systems do not experience a fixed dataset, but a feedback loop between the system and its experiences [3]. A dynamic environment is considered in which state-action-reward triples are observed as the data. The objective of reinforcement learning is mapping situations to actions with the goal of maximizing rewards. Other learning systems exist that are a combination of two categories, such as **semi-supervised learning** that use both labeled and unlabeled data [4].

Solving Machine Learning Tasks

A wide variety of tasks exist that could be solved with machine learning. Two popular machine learning tasks are regression analysis and classification. In regression analysis, the relationship amongst variables is approximated, for the successful prediction of a value given some input. This task is solved by outputting a function $f : \mathbb{R}^n \to \mathbb{R}$ that fits the data [45]. Regression analysis can be used for example to forecast future stock prices in the trading world. In classification, the machine is asked to determine the category n that a certain input belongs to. The task can be solved by outputting a function $f : \mathbb{R}^n \to \{1, ..., n\}$ [45]. A popular classification problem is object recognition for intelligent systems. Classification can be used for example to classify objects in a warehouse to determine the correct destination of each object, with current state-of-the-art object recognition making use of deep learning algorithms [5].

2-2 Optimal Supervised Learning Algorithm Selection

To determine what machine learning technique suits a specific application, one can analyze the aspects that are necessary for an optimal supervised machine learning pipeline. Kotsiantis et al. [6] describes a pipeline that can be used to create a successful classifier that generalizes well for new data instances. This pipeline is illustrated in figure 2.1.

The first two steps of this pipeline are the most important and largely define the performance of the classifier. Identifying required data consists of determining and choosing the most relevant features. By excluding irrelevant or redundant features, data dimensionality can be reduced. Too much irrelevant or redundant information can prevent a learning algorithm from finding patterns in data or even result in false results. The pre-processing step is used to deal with this information redundancy, and is often used to deal with noise or missing values. The end result of the pre-processing step is the input of the training dataset.

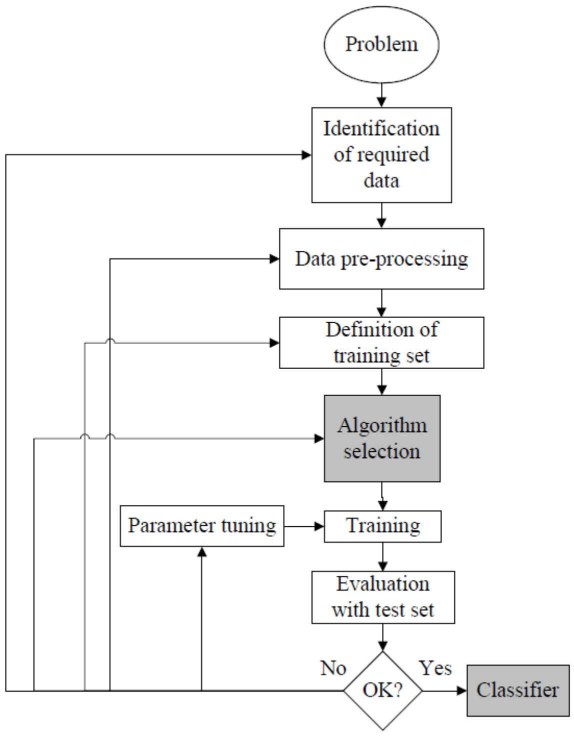


Figure 2-1: Supervised machine learning pipeline that is used to create a successful classifier [6].

Describing and Processing Datasets

A method to describe the datasets that are used to train and test machine learning algorithms is with a design matrix [3]. The design matrix is a matrix that consists of all datapoints, where each column can correspond to a certain feature. For example, if one uses a set of 10 photos taken of an object with a resolution of 1600x1200 with three features for each photo, the dataset can be represented with a design matrix $X \in \mathbb{R}^{10\times1600\times1200\times3}$. Datasets that are used to train and test machine learning algorithms vary from relatively simple, to large and complex datasets. An example of a relatively simple dataset is the Iris dataset [7]. This dataset contains 150 samples, each with four data instances and can be described by a design matrix $X \in \mathbb{R}^{150\times4}$. On the other hand, when dealing with photographs datasets can become very large due to high image resolutions. An image with a resolution of 1900×1080 pixels, with each pixel representing a datapoint with an x, y and z value can easily produce an amount of 6156000 data points per image. Due to the large amount of datapoints per example, a lot of processing power is necessary to train and test the machine learning algorithm. As the amount of data produced keeps growing each day, more and more complex data can be used to train machine learning algorithms. In the field of computer vision, datasets can consist of up to thousands of images that can each again contain dozen of features.

2-3 Reasons for Scaling Up Machine Learning

The amount of data and events that are saved and logged every day keeps increasing, especially in the world of Internet and finance. Chains such as Walmart are able to collect many Petabytes of data each hour from its customers, and globally billions of financial transactions take place each day. As the amount of sensors being used in these industries keeps expanding, the amount of observation that could be used as training data also keeps increasing. The problem becomes even bigger when each of these data points contains multiple features. When billions of datapoints each day are accompanied by thousands of features per datapoint, the dataset can easily result in 10^{12} datapoint-feature pairs per day. Storing this data means that these datasets easily become hundreds of Petabytes large.

When the scale of a machine learning task is too large for a single-machine processing unit, parallelization techniques can be used to speed up the process. Bekkerman et al. [1] describes four settings for which large-scale machine learning are necessary:

1. Dataset consists of a high quantity of datapoints.
2. Datapoints have high input dimensionality.
3. The model and algorithm are complex.
4. When dealing with inference time constraints.

Firstly, in the case of data collection by chains such as Walmart, large number of datapoints are collected. Secondly, high input dimensionality plays a role when dealing with machine learning tasks that involve the use of video or image data. For such cases, the input dimensionality can easily extend to a scale of 10^6. Thirdly, highly complex non-linear models, or very computational intensive processes can be used to outperform simpler algorithms and reach higher accuracies [1]. Two examples of such algorithms are multilayer deep networks and decision trees [8]. Lastly, inference time constraints can be described with applications that make use of sensors and real time prediction, such as a walking humanoid robot, or an autonomous driving vehicle. In these settings, the inference time is subject to certain constraints for the successful execution of a task (such as walking, or driving). Bekkerman et al. argued that for these four settings, the utilization of highly parallelized hardware architectures such as GPUs, or the implementation of efficient computational frameworks such as DryadLINQ is necessary [1].

No Free Lunch Theorems of Optimization

Numerous articles have been published that compare the performance of various machine learning algorithms [6][9][10][11]. However, no clear solution has been found to the question which algorithm is best for a particular application. Mooney [12] argues that no inductive algorithm is universally better than any particular other. Wolpert [13] confirms this argument and shows theoretically that (with certain assumptions), no classifier can always reach a better performance than any particular other. Wolpert and Macready [14] formed the "No Free Lunch Theorems of Optimizations" in which they stipulate that there is no existing universal optimization technique, however a particular algorithm can outperform others if it is specifically designed for the structure of a particular problem [15].

With this in mind, a comparative study still provides sufficient insight in the performance and usability of algorithms for specific applications. Additionally, by performing a scalability study insights can be gained in how algorithm computational time increases with scale. This information can prove useful for large-scale machine learning implementations.

Chapter 3

Review of Machine Learning Algorithms

In this chapter, the findings of the literature survey are presented. Four of the most popular and influential machine learning algorithms in data mining [10] are presented and reviewed: Decision Trees, Neural Networks, Support Vector Machines and k-Nearest-Neighbor. Neural networks are interesting in particular, as they are the most typical approach when working with highly dimensional data, such as 3D object classification. The theory behind each algorithm is explained, advantages and drawbacks are analyzed, and the scalability potential of each algorithm is discussed.

3-1 Large Scale Machine Learning Implementation

The last decade has marked a rapid and significant growth of large-scale machine learning applications, attributed by two main reasons: (1) The increase of large datasets over many modern applications, and (2) The growth and evolution of programming frameworks and hardware architectures that can be used by many learning algorithms [42]. Large datasets are often collected and saved on large distributed storage platforms, motivating the development of machine learning algorithms that can utilize this efficiently stored data. Improved programming frameworks and hardware architectures allow for efficient and simultaneous processing of these large sums of stored data. Furthermore, the increased use of sensors that make real-time decisions based on large sums of high dimensional and complex features causes the growing demand in utilization of hardware architectures that allow efficient parallel computations in learning applications. Two examples of technology that have affinity with such data are visual object detection for autonomous systems [43] and speech recognition [44].

The goal of investigating machine learning algorithms is to provide an overview of the most effective algorithms and learn how well they can be implemented in large-scale machine learning applications. In order to conduct a thorough but efficient study, the following literature research questions were formulated:

1. **What are the most successfully performing supervised machine learning algorithms?**
   This is investigated by analyzing published articles that compare the performance of these algorithms on different various datasets. The performance is evaluated based on three main criteria:

   (a) Classification speed: The amount of time needed to classify a query instance.

   (b) Accuracy: The amount of correct predictions made divided by the total amount of predictions.

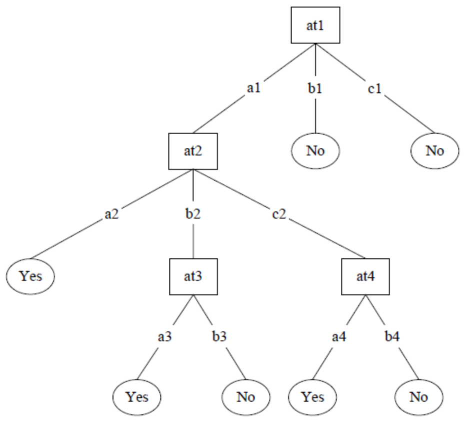
   (c) Noise Tolerance: How well the machine learning algorithm can cope with noise such as irrelevant or redundant features.

   Most importantly, the largest bottlenecks and issues are investigated, and model comprehensibility is taken into consideration.

2. **What are the scaling possibilities for these successful machine learning algorithms?**
   Theoretical complexity analysis is performed on the most successful algorithms to determine their scalability potential. Furthermore, the practical viability of these scaled algorithms is analyzed.

3-2 Decision Trees

Decision tree algorithms comprises of trees that categorize data by looking at feature values. Every node in a decision tree depicts a feature that has to be classified, and the branches depict values that are considered by such a node. A survey of existing work on decision tree construction is done by Murthy [16], who describes the use of decision trees in multiple disciplines such a statistics, pattern recognition and machine learning. One of the most popular decision tree algorithms in literature is the C4.5 algorithm by Quinlan [17], an improved version of the earlier decision tree algorithm by Quinlan, the ID3 [18]. C4.5 uses a divide and conquer strategy to grow a tree, selecting one feature with a minimum of two outcomes that divides the set of samples most effectively. When all instances in a certain set belong to one class, or the set size is too small, a label is assigned to a decision tree leaf equal to the most popular class label [10]. A decision tree is illustrated in Figure 3.1. The highest decision node is called the root node. The feature that can sort the data most effectively is chosen as the root node. This strategy is duplicated for each sub-division of the training data, until all data is divided into specific class batches. Determining which feature sorts the training data most effectively can be done through many techniques such as the "Gini Index" [19], "Information Gain" [20], or the "ReliefF Algorithm" [21]. Even though each of these techniques differs from the other, a survey of decision tree algorithms by Murthy [16] showed that no single best method exist for determining the best sorting feature. However, for a particular dataset, comparing individual techniques may provide useful insights for the choice of sorting feature.



at1	at2	at3	at4	Class
a1	a2	a3	a4	Yes
a1	a2	a3	b4	Yes
a1	b2	a3	a4	Yes
a1	b2	b3	b4	No
a1	c2	a3	a4	Yes
a1	c2	a3	b4	No
b1	b2	b3	b4	No
c1	b2	b3	b4	No

Figure 3-1: Decision tree for the training set with classes Yes and No. The instance [at1=a1, at2=b2, at3=a3, at4=a4] sorts to the nodes at1, at2 and at3, classifying the instance as positive represented by the value "Yes" [6].

In order to obtain right sized trees Breiman et al. [19] proposed "pruning", a technique that is used to reduce the complexity of decision trees by removing sections of a tree that provide little information for the classification of instances. To prevent overfitting training data, two common approaches can be used: (1) stopping the training algorithm before a perfect fit is reached; and (2) pruning the decision tree. A survey of popular pruning methods is presented by Elomaa, who concludes that no single best pruning method exists. Breslow and Aha [46] provide a good overview of decision tree simplification methods for comprehensibility improvements.

Advantages and Challenges

The main advantage of decision tree algorithms is that they are easily understood. The reason why a particular decision tree classifies data to a certain class can be easily interpreted, even by people that are not an expert on the area. Tjen-Siem Lim et al. performed a comparative study of decision trees against other machine learning algorithms, and concluded that the C4.5 algorithm possesses a very good combination of classification speed and error rate [29]. Ruggieri [30] has done a research where he examined the runtime performance of the C4.5 algorithm. Based on this study he created an improved version of the C4.5, the EC4.5, that calculated identical decision trees as in C4.5 with an increased performance gain of up to 500% [30].

Scalability Analysis

Decision trees containing only a single variable (univariate) are very prominent in the data mining world and differ in their division method. Building an univariate decision tree has a time complexity of $O(dfNlog(N))$, where N represents the total sample size, f the number of features and d the number of tree nodes [31]. Building univariate decision trees in reasonable time is very beneficial in data mining applications that contain large datasets, and can be achieved by parallelizing these algorithms. Olcay and Onur [31] propose side-by-side applications of the C4.5 algorithm through three different methods: (1) feature based parallelization; (2) node based parallelization; and (3) data based parallelization. They concluded that theoretically, feature based parallelization would reach high speedups on datasets with high dimensionality, data based parallelization on large datasets and node based parallelization on datasets that contain trees with a high number of nodes when discriminated by the serial decision tree. However in all of these cases, perfect load balancing is required. In their simulations, they observed that which parallelization method works best or how much speedup is reached depends on how well the load is distributed amongst processors.

3-3 Artificial Neural Networks

Neural networks have gained widespread recognition as an effective machine learning algorithm by outperforming many algorithms such as Support Vector Machines in various relevant applications such as pattern recognition [22][23]. A neural network is an architecture that comprises of units named neurons. These architectures usually consist of three different layers: the input layer which contains the input feature vector; the output layer that consists of the neural network response; and the layer in between that contains the neurons that connect to both the input and output. An example of a neural network is illustrated in figure 3.2. This artificial neural network, called a Feed-forward neural network, only allows signals to travel from input to output.

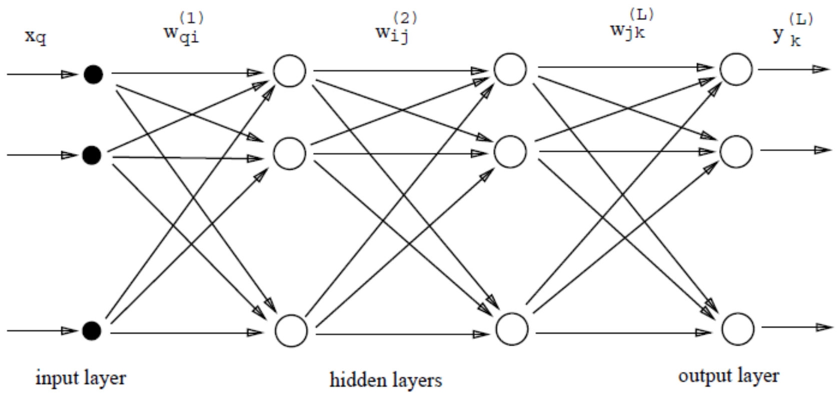


Figure 3-2: Feed-forward Artificial Neural Network, allows signals only to travel forward from input to output [32].

Artificial neural networks consist of three fundamental characteristics: the network architecture; input and activation functions; and the weight of input connections. The network architecture and functions are chosen at the initial stage and remain the same during training. The performance of the neural network is reliant on the value of the weights. The weights are tuned during training so that a certain output is achieved. ANN can be trained using a multitude of different training programs [24]. A very prominent training method is the Back Propagation algorithm [25]. Other techniques include the Weight-elimination algorithm which automatically infers the network topology, and Genetic algorithms that try to derive the network architecture and train its weights [26][27].

Advantages and Challenges

The main issue of NN implementation is deriving the correct hidden layer size. When the amount of neurons is not determined properly, the derived system does not generalize well to unseen instances. On the other hand when too much nodes are used, overfitting may occur and the desired optimum may not be found at all. Deriving the right quantity of neurons is discussed in a study by Kon and Plaskota [48]. The main advantage of using artificial neural networks is its capability to process data with high dimensional features such as images. Drawbacks of artificial neural networks are high computing costs that consume large amounts of processing power and physical memory usage, and difficult comprehensibility for average machine learning users [6][9].

Scalability Analysis

To discuss scalability of neural networks, the widely used back propagation algorithm is considered. The Back Propagation algorithm, as described by Kotsiantis et al. [6], generally includes the next six steps:

1. An instance is given as input to the network.
2. The system output is correlated to the wished result from a specimen and the error is determined for each neuron.
3. Compute the local error for each neuron.
4. Weight alteration to minimize the local error.
5. For the error a punishment is accredited to previous level neurons, where more importance is added to the neurons that are stronger weighted.
6. A reiteration is done where each neuron punishment is used as its error.

The optimal weight arrangement is gained through multiple adjustments of the weights by the algorithm. With n instances and W weights, the training stage computation time is equal to O(nW) [6]. No exact formula exists to determine the amount of training required for a neural network. This number generally depends on the problem and chosen neural network architecture. A study by Barron [33] showed that the error in a feedforward neural network could be equal to O(1/N). Furthermore, a rule of thumb exists for the size of training set T [34]:

$$T = O(N/\epsilon) \tag{3-1}$$

where ϵ represents the part of classification errors that is allowed. Because the feedforward operation is O(N), large parallel computers can be used to train these neural networks and derive the weights, and thereafter copy them to a serial computer for implementation.

3-4 Support Vector Machines

Support Vector Machines (SVM) is a discriminative classification technique that derives from the Structural Risk Minimization principle from computational learning theory. The aim with SVM is to find the most optimal classification function that differentiates between units of classes in training data. With a linearly separable dataset, the most optimal classification function can be decided by constructing a hyperplane which maximizes the margin between two datasets and thus creates the largest possible distance between datasets [23]. A visualization of this strategy is illustrated in figure 3.3.

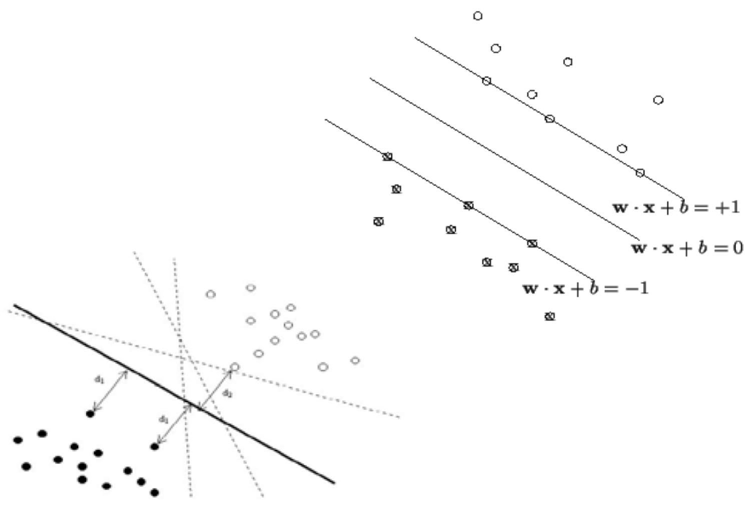

$\mathbf{w}\cdot\mathbf{x}+b=+1$
$\mathbf{w}\cdot\mathbf{x}+b=0$
$\mathbf{w}\cdot\mathbf{x}+b=-1$
d_1
d_2
d_1

Figure 3-3: Creating the optimal separating hyperplane with the use of support vectors [9].

The idea behind SVMs is that by finding the maximum margin and thus the most optimal hyperplane, the best generalization ability is reached. This results in the best classification performance for both the training data as well as future data. In order to find maximum margin hyperplanes, SVM aims to maximize function 3.1 in relation to $\vec{\mathbf{w}}$ and b:

$$L_P = \tfrac{1}{2}\|\vec{\mathbf{w}}\| - \sum_{i=1}^{t} \alpha_i \gamma_i (\vec{\mathbf{w}} \cdot \vec{\mathbf{x}_i} + b) + \sum_{i=1}^{t} \alpha_i \tag{3-2}$$

Here, t represents training point quantity, α_i stands for Lagrangian multipliers and L_P exemplifies the Lagrangian. Vector $\vec{\mathbf{w}}$ and constant b characterize the hyperplane. When the optimal separating hyperplane is found, the datapoints that lie on its margin are defined as the support vector points. The solution is a linear combination of these support vector points, and all other datapoints are ignored. This means that the the complexity of a SVM is not affected by the number of features present in the training data, making SVMs very suitable for learning problems that contain a large number of features in comparison to the amount of training instances. An issue with the implementation of SVMs is that for many datasets, no separating hyperplane can be found when the data contains misclassified instances. Real world problems however often involve nonlinearly separable data, where no hyperplane can be found that correctly divide the cases in a training set. An answer to this inseparability issue is outlining the data onto a altered feature space. By choosing a suitable altered feature space of more than necessary dimensionality, it is possible to make any regular training set divisible [6].

Advantages and Challenges

The use of SVMs provides many advantages. The algorithm is based on an established theory, desires only tens of training specimens, and is unresponsive to the number of dimensions [6]. However, the learning technique has relatively complex training and categorization algorithms, and high memory and time utilization during training and classification phases [9].

Scalability Analysis

Training support vector machines is done by solving an N^{th} dimensional Quadratic Programming problem (QP), with N representing the number of training instances. Implementing standard QP methods to solve this problem involves large matrix operations and time consuming numerical calculations. This method is very inefficient and not practical for large scale applications, and is a well known drawback of the SVM method. More efficient methods exist that can solve the QP problem relatively quickly, such as *Sequential Minimal Optimization* (SMO) [35]. This method fragments the QP problem into QP sub-problems and solves the SVM quadratic programming problem without the use of numerical QP optimization steps or extra matrix storage. A novel SVM approach finds an estimation of a least surrounding sphere of a group of items [10]. SVM methods are binary, and when a multi-class problem has to be solved, the problem is divided into a group of various classification challenges.

3-5 *k*-Nearest-Neighbor

k-Nearest-Neighbor classification, or kNN, is a machine learning algorithm that localizes a group of k objects in a training case that has the closest proximity to the test object, and then assigns a label derived from the prevalence of a class in the closest proximity. Three important components are needed for this algorithm: a group of labeled objects; a proximity metric; and the number k of nearest neighbors [10]. A popular proximity metric that is used for kNN classification is "Euclidian Distance", explained by formula [6]:

$$D(x,y) = (\sum_{i=1}^{m} |x_i - y_i|^2)^{1/2} \tag{3-3}$$

Other metrics exist for defining the distance between instances of a dataset. Examples include the Minkowsky, Camberra or Chebychev metrics [11], although often weighing strategies are used that alter the voting influence for more accurate results.

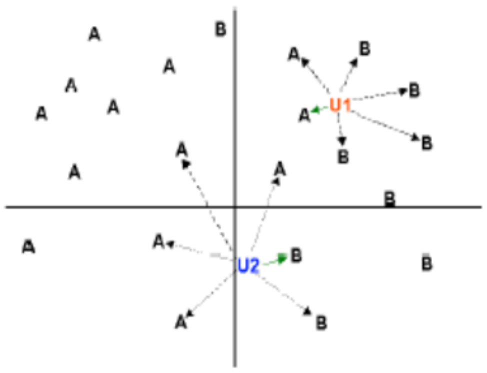


Figure 3-4: Representation of a k-Nearest-Neighbor graph.

When the k-nearest-neighbor list is acquired the test objects are categorized according to the majority class, given in formula 3.4:

$$Majority\ Voting : Y^{'} = argmax \sum_{(x_i,y_i)\ \in\ D_z} I(v = y_i) \tag{3-4}$$

Here, v represents the class label, y_i represents the ith nearest neighbor class label, and $I(\cdot)$ is the indicator functions that returns value one for a valid argument or zero for an invalid argument [10].

Advantages and Challenges

The main advantage of k-Nearest-Neighbor algorithms is that they are easily understood and implemented. Despite being simple the algorithm still performs well for many cases, especially for multi-modal classes. However, essential problems influence the algorithm behavior. Kotsiantis et al. [6] describe three main reasons: (1) large storage requirements; (2) oversensitive to the choice of similarity function; and (3) a key method for selecting k is lacking.

Scalability Analysis

The training phase of kNN classification does not exist, instead all feature values are stored in memory. Unlike other machine learning algorithms like SVMs or decision trees, kNN classifiers are considered slow learners. Creating the model is computationally inexpensive, whereas the classification phase is computationally expensive because each k-nearest neighbor needs a label. This requires calculating the distance metric of the unlabeled instance to all instances in the labeled set, which can be extremely expensive in particular for large datasets. There are a number of existing methods that deal with this problem, such as the inverted index join algorithm [36], one of the fastest algorithms for producing exact kNN graphs. However, this algorithm requires $O(n^2)$ asymptotic time complexity, and has an exponentially growing execution time. More recent algorithms are being developed that guarantee a time complexity of O(n), such as the greedy filtering algorithm by Park et al. [37].

3-6 Discussion and Conclusion

Several empirical comparison studies exist that review and discuss the latest and most popular machine learning algorithms. Kotsiantis et al. show that C4.5 contains a good mixture of error rate and calculation time, with easily interpreted decision trees. However, when compared to SVM, C4.5 has a lower performance when there are multi-dimensions and continuous features [6]. Furthermore, Kotsiantis and Miyamoto et al. show that when comparing a subset of the C4.5 and kNN (among other algorithms) with SVM, SVM was shown to have a significant higher performance than C4.5 and kNN [6][38]. The difficulties that arise when working with Neural Networks, come from choosing the right size of the hidden layer. An underestimation of this layer can lead to poor generalization, and an overestimation can lead to overfitting [10]. Royas-Bello et al. and Aruna et al. compared SVM with Neural Networks using multiple datasets and both concluded that SVM has a higher accuracy than Neural Networks [39][40]. The kNN algorithm stores all feature values and lacks a training phase. However, there are multiple issues that can affect the performance of this algorithm. Kotsiantis et al. and Wu et al. describe that kNN requires large storage, is sensitive to noise and irrelevant features, is sensitive of the distance measure used and the choice of k is not principled [6][10]. Similarly, the presence of irrelevant features in a neural network can make network training very inefficient.

SVMs and Neural networks achieve better performances working with multi-dimensions or continuous features, as opposed to logic-based systems working better with categorical or discrete features. Furthermore NNs and SVMs achieve better performances when working with multicollinearity and an existing input-output relation. Decision trees on the other hand do not cope well when diagonal partitioning is present. kNN classifiers can be very affected by noise, a characteristic that is opposed by decision trees which are very tolerant of noise. Univariate decision trees are outperformed (speedwise) by NNs by multiple orders of magnitude.

Reflecting back on Wolpert and Macready's "No Free Lunch Theorems of Optimizations", no single machine learning algorithm can outperform another one over all datasets. However, by performing a comparative study, insights can be learned in the performance and usability of algorithms for specific applications. Important features of the four machine learning techniques decision trees, artificial neural networks, kNN and SVM are summarized and compared in table 4.1, based on the evidence of reviewed empirical and theoretical studies.

	Classification Speed	Accuracy	Noise Tolerance	Model Comprehensability	Scalability Potential
Decision Trees	++	+-	+-	++	+-
ANN	++	++	+-	- -	+
kNN	- -	+-	- -	+-	+
SVM	++	++	+-	- -	+-

Table 3-1: Comparison of machine learning algorithms (++ signs represent the best situation and – signs represent the worst situation). ANN scores the highest points and is chosen as the best candidate algorithm.

Concluding this study, when asking the question which algorithm performs well on a particular classification problem, the best approach to answering this question is by performing a benchmark of the best algorithm contenders and selecting the most promising algorithm for a specific application.

The last decade of progress on visual recognition tasks primarily involved the use of SIFT and HOG features. When looking at the particular application of visual recognition, the best candidate algorithms from table 3-1 are ANNs and SVMs. Artificial Neural Networks, in particular Convolutional Neural Networks saw considerable use in the 1990s, but fell out of fashion in the computer vision area due to the rise of SVMs. However, Krizhevsky et al. revived interest in CNNs by presenting a significant increase in image classification accuracy on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This result stemmed from training a large CNN on 1.2 million labeled images. With the presently available abundance of visual data, and empirical evidence of their high achieved accuracies on various datasets, CNN is chosen as the best candidate algorithm.

Chapter 4

Object Identification Framework

In this chapter, a new object identification framework is discussed. Vanderlande system requirements and constraints, and barcode placement guidelines are used to define the desired objective, and input, data and outputs of the framework.

4-1 Barcode Placement Guidelines

Many applications exist that can find and read barcodes from complex scenes with a camera, and use it to identify objects. Differences in illumination, poses and perspective distortion can make barcode identification quite difficult, but algorithms exist that provide adapted solutions for these complex scenes. The case that is most interesting is the one where a barcode is concealed as a result of the product pose, and cannot be identified as a result hereof without altering the position of the object. Figure 4-1 provides an example of a warehouse container consisting of items where the barcode is visible as well as other items with barcodes concealed behind other surfaces.

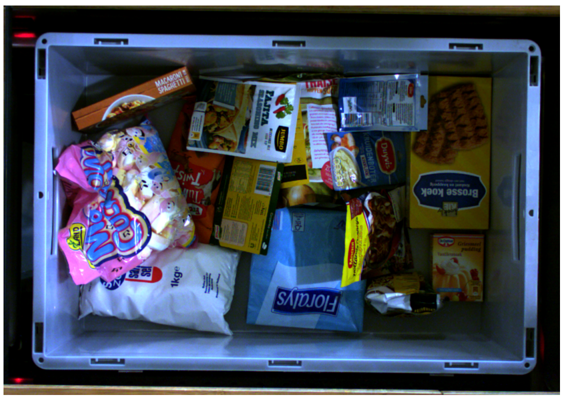


Figure 4-1: A typical arrangement of products in a container where not all barcodes are visible. Revisiting the main research question: can a barcode placement pattern be derived to localize barcodes on products with concealed barcodes.

It is known that a certain pattern exists for the barcode placement on objects of different sizes and shapes. Most of the items that are being handled in Vanderlande warehouses follow the GS1US guidelines for barcode symbol placement. These guidelines provide the general principles that apply, mandatory rules and recommendations for barcode symbol placement on specific packaging types, as is illustrated in figure 4-2 for bottles and jars.

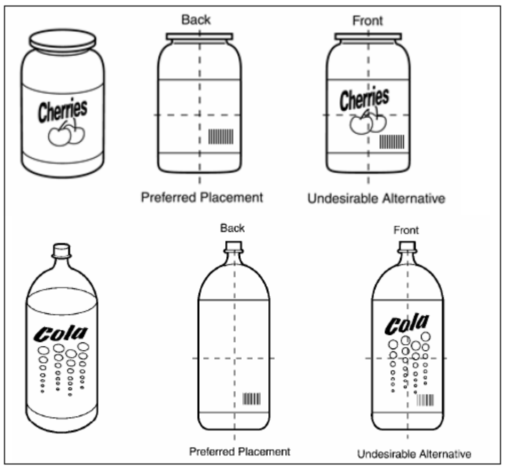


Figure 4-2: Barcode symbol placement on bottles and jars using the GS1US standard guidelines. With not all items adhering to the guidelines, the question raised is: can a usable barcode placement pattern be uncovered.

These guidelines can prove very handy in predicting the correct barcode location when the barcode is concealed. However in many cases, barcode placement on items can derogate from the GS1US guidelines. While not all items adhere to these barcode placement regulations, it is very interesting to investigate whether there exists a usable relation between object features such as size or shape, and barcode location. If this relationship exists, it can be used to pick and scan items very efficiently in a warehouse, eliminating the need of a human worker for object identification. Machine learning will be implemented to derive these patterns of barcode placement on objects of different shapes and sizes. By analyzing shape, texture and a set of other possible features, a new object recognition framework is proposed that should robustly determine the barcode location of objects.

4-2 Barcode Localization Framework

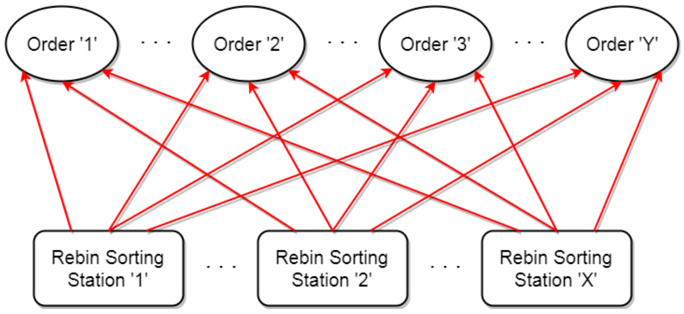
The framework of the complete object recognition framework is defined. As decided in chapter 3, the framework will be built with the use of a convolutional neural network. The goal of this barcode localization method is defined as follows:

> Find the pattern of barcode placement on products with the use of a convolutional neural network to determine on which surface of an item the barcode lies.

In order to automate the rebin phase of the Vanderlande order fulfillment system as depicted in figure 4-3, two important requirements have to be satisfied by the newly developed object recognition framework:

1. The system has to reach 40% barcode localization prediction accuracy: With Vanderlande order fulfillment systems processing vast amounts of products, the newly designed vision systems should strive for at least 40% prediction accuracy. Hereafter improvements can be made by introducing more data to learn from, or implementation of more optimal learning techniques.

2. The system should be scalable to very large product sets: The designed vision system should be able to process current Vanderlande warehouse product sets, which lie in the order of 10^7 different products.



Order '1'
Order '2'
Order '3'
Order 'Y'
Rebin Sorting Station '1'
Rebin Sorting Station '2'
Rebin Sorting Station 'X'

Figure 4-3: Rebin phase of the Vanderlande order fulfillment system. Orders are created with items from different rebin sorting stations.

When designing the object recognition framework, one important system constraint has to be taken in consideration:

1. Data cannot be collected easily: Creating and annotating data for large product sets to use in a learning pipeline can be very time-consuming and expensive. Furthermore, in most cases Vanderlande does not have access to these product sets, e.g. the complete product set of retailer "Bol.com".

4-3 Defining Framework Parameters

In order to build the barcode localization framework, the input, output and class definition are first defined.

4-3-1 Vanderlande Product Categories

Vanderlande makes automation solutions for different markets: parcel, warehouse and airports, which means that objects that these solutions handle generally exhibit a large variety in shape, color, weight and other physical properties. It is important to realize that for this project, the focus lies on the warehousing segment. Furthermore, products are chosen from standardized categories, because this avoids ambiguities regarding the class definition and interpretation. One such standard is GS1-US, which provides categorization that is suitable for testing various types of item categories, see figure 4-6. The scope of this project is restricted to "Box" category items, for which a proof of concept is built. Once a working framework is realized, other product categories can be added to the framework.

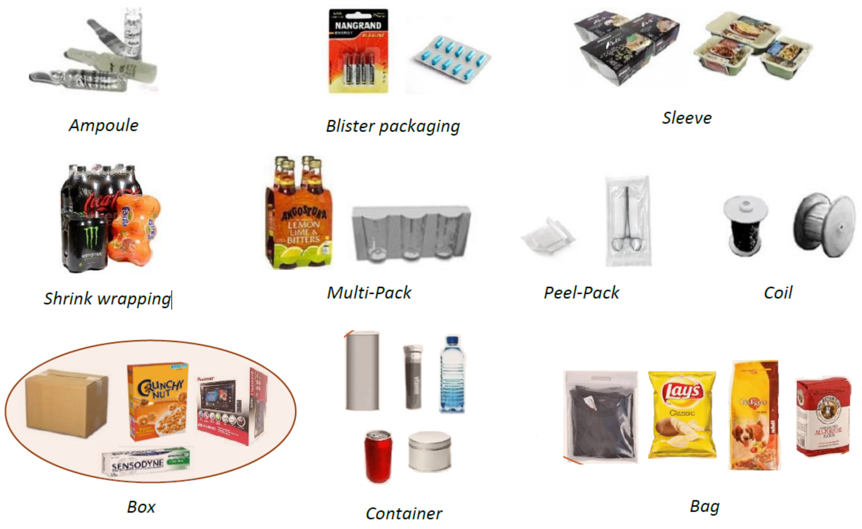


Figure 4-4: GS1-US product categories. This thesis study will only focus on 'box' shaped objects.

4-3-2 Input Data

The dataset that is created consists of images of box category item, taken from a top side view. Examples of these images with single items from various poses can be found in figure 4-7.

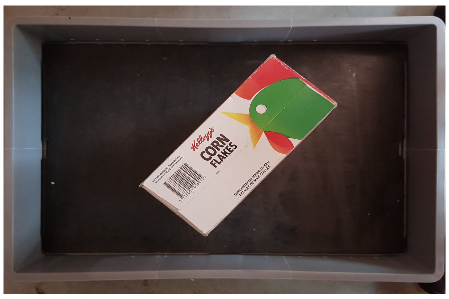


Figure 4-5: Box of cereal with a visible barcode on the top plane.

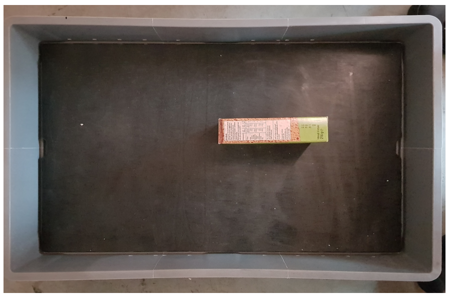


Figure 4-6: Box of cookies with a concealed barcode. For both the cereal box as well as the box of cookies, the barcode localization framework should be able to determine the barcode location.

The images will be imported in the framework as RGB data, with a resolution of 400x300. Thus input data will consist of a design matrix $X \in \mathbb{R}^{N\times400\times300\times3}$, where 'N' represents the number of samples in the dataset.

4-3-3 Class Definition

As convolutional neural networks are a type of supervised learning, labeled data is needed for the training phase of the framework. In order to label all data efficiently, the following class definition is chosen:

$$c_n = \{1, 2, 3, 4, 5, 6\}$$

In this class definition each of the numbers represent one of the six faces of a box sized object. This class definition is always applied from the perspective of the camera. For all cases, the face that is closest to the camera is defined as the top face (c=1), from there on the other faces are defined. The face opposite of the top face is the bottom face (c=3). The faces adjacent to the top and bottom faces are the front (c=2), back (c=4), right (c=5) and left face (c=6). With the camera perspective in mind, the definition is interpreted and illustrated as follows:

$$c_n = \{1, 2, 3, 4, 5, 6\} = \{\text{top, front, bottom, back, right, left}\}$$

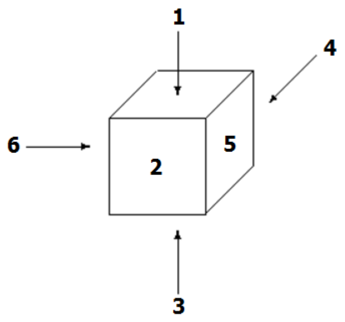


Figure 4-7: Proposed object recognition framework with class definition $c_n = \{1, 2, 3, 4, 5, 6\} = \{\text{top, front, bottom, back, right, left}\}$.

4-3-4 Output Vector

The output of the framework consists of a categorical probability distribution vector, that provides the probabilities of a barcode being present on a specific face. A possible outcome can be given by:

$$c_n = \{1, 2, 3, 4, 5, 6\} = \{0.80,\ 0.10,\ 0.05,\ 0.02,\ 0.02,\ 0.01\}$$

This outcome means that an 80% chance exists that the barcode will lie on the top face, 10% that it will lie on the front face, and so on. The complete framework is illustrated in figure 4-10.

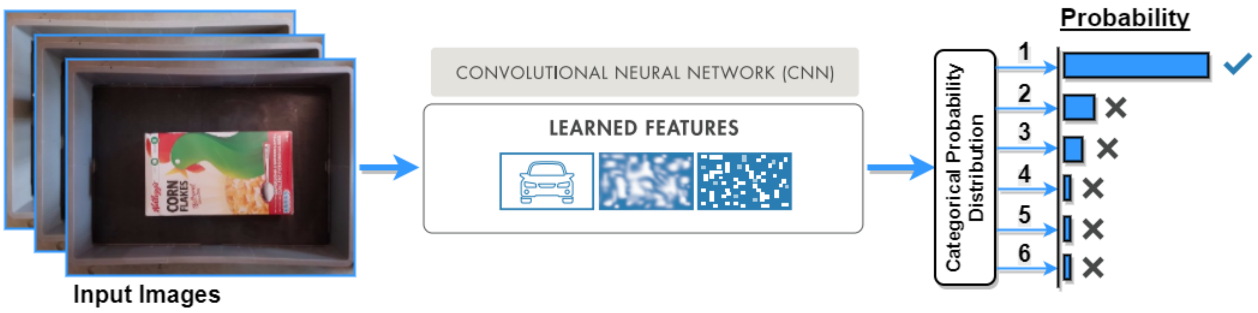

Input Images
CONVOLUTIONAL NEURAL NETWORK (CNN)
LEARNED FEATURES
Categorical Probability Distribution
Probability
1
2
3
4
5
6

Figure 4-8: Class definition for a box shaped object. For a particular input image with the portrayed categorical probability distribution, the resulting barcode location would be class 1, the top side.

Chapter 5

Deep Learning Implementation

In this chapter, first an introduction is given to Convolutional Neural Networks. Hereafter the proposed object recognition framework is implemented in Matlab, by building a CNN to train a system to recognize the position of barcodes on products.

5-1 Convolutional Neural Network Theory

Convolutional neural networks is currently one of the most prominent algorithms for deep learning with image data. Whereas for traditional machine learning relevant features have to be extracted manually, deep learning uses raw images as input to learn certain features. CNNs consist of an input- and output layer, and several hidden layers between the input and output. Examples of in between layers are convolutional layers, max-pooling layers and fully connected layers.

CNN architectures vary in the number and type of layers implemented for its specific application. For continuous responses, the network should include a regression layer at the end of a network, whereas for categorical responses the system must include a classification function and layer. Neurons in each CNN layer are arranged in a 3D arrangement, and transform a three dimensional output from a three dimensional input. For our particular application, the input layer holds the images as 3D inputs, with the height, width and RGB values as dimensions. Hereafter, in the convolutional layer neurons are attached to the regions of the image and transformed into a three dimensional output, see figure 5-1.

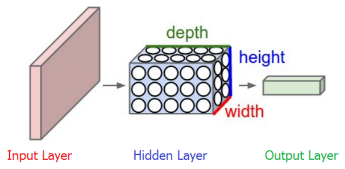


Figure 5-1: A CNN where the red input layer consisting of an image is transformed into a 3D arrangement. The height and width of the hidden layer are the dimensions of the image, and the depth consists of the three RGB channels [42].

CNN configurations comprise of a multitude of hidden layers. In each layer, activation volumes are altered with the use of differentiable functions. Four principle layer types exist that are used to build CNN configurations, an example is illustrated in figure 5-2.

1. **Convolutional Layer (CONV)**: Convolutional filters are used to derive an activation map from the input data.
2. **Rectified Linear Unit Layer (ReLU)**: Filters negative values to provide only positive values for a much faster training time.
3. **Pooling Layer (POOL)**: Performs nonlinear down-sampling and cuts down the amount of parameters for a simpler output.
4. **Fully Connected Layer (FC)**: Computes the class probability scores by outputting a vector of C dimensions, with C being the number of classes. All neurons are connected to this layer.

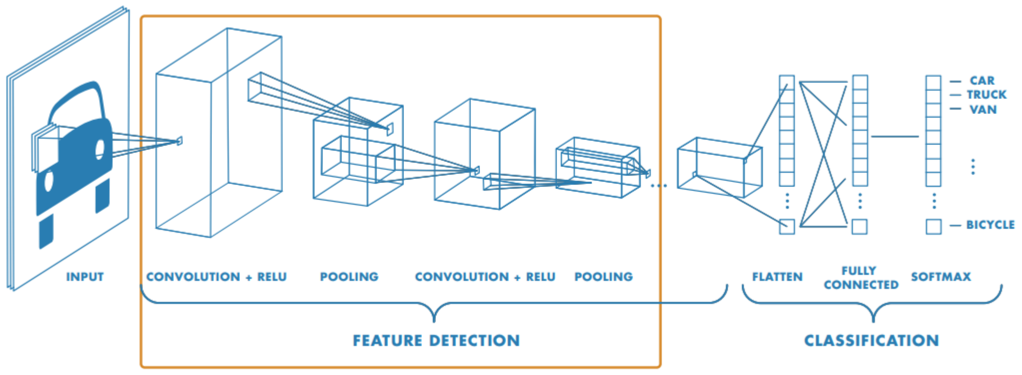


Figure 5-2: Convolutional neural network architecture that classifies input images as belonging to a number of categories including cars, trucks, vans and bicycles [41].

5-2 Building a CNN in Matlab

The framework described in this thesis is developed in Matlab. The decision to use Matlab was made because Vanderlande uses the program internally for all its applications. The framework itself is meant to run on a desktop computer. Two Matlab toolboxes are used: The Parallel Computing, and Neural Network Toolboxes. These toolboxes provide convolutional training algorithms, pre-programmed layers and convolutional networks such as AlexNet, and parallel computing capabilities to significantly decrease computing time.

CNN Network Layer Configuration

When defining the network layer configuration it is important to note that there does not exist an exact formula for optimal layer configuration. Instead, the best approach is to trial and error, where a few layer configurations are explored and benchmarked to see how well they work. Pretrained networks such as AlexNet can also be used as the initial layer configuration. However, in this case, while inspiration is drawn from the AlexNet layer configuration, own implementations of layer configuration are pursued. The chosen layer configuration is illustrated in figure 5-3:

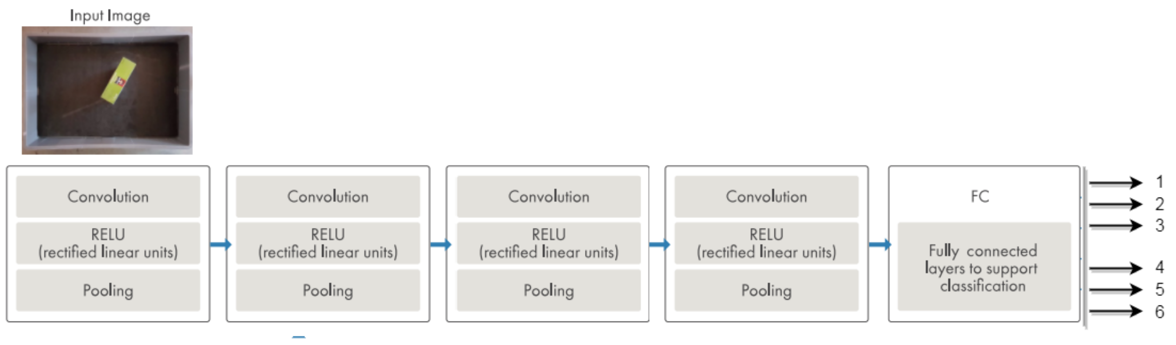

Input Image
Convolution
RELU (rectified linear units)
Pooling
Convolution
RELU (rectified linear units)
Pooling
Convolution
RELU (rectified linear units)
Pooling
Convolution
RELU (rectified linear units)
Pooling
FC
Fully connected layers to support classification
1
2
3
4
5
6

Figure 5-3: Convolutional neural network layer configuration. The designed network consists of 13 layers, with varying filter sizes tweaked through trial and error iterations.

Training the deep convolutional network depicted in figure 5-3 can take a variable amount of time, dependent on the dataset size and available processing power. Three computation options are available to train CNNs, and choosing the most optimal one is crucial depending on the amount of available time to solve a particular task:

1. CPU based computation
2. GPU based computation
3. Cloud based GPU computation

The CPU based computation options is the most simple and readily available. However since a CPU computes task in serial configuration, training the network takes the longest time using this method. The use of a Graphical Processing Unit cuts down the training time significantly and can be done with Matlab without further programming. However, a CUDA based NVidia GPU is necessary with atleast 3.0 compute capability for parallel computation. There is also the possibility to use multiple GPUs, this decreases processing time even further. Lastly, cloud based GPU computation considers the employment of cloud resources for the processing power. Written matlab code can be enhanced for cloud computing purposes. In this thesis framework, both CPU and GPU computation is considered to investigate the effect of the use of GPUs on decreasing the computation time.

5-3 CNN Training Phase

Collecting and Labeling Data

The testing configuration in which data is created and collected is depicted in figure 5-4. The RGB camera takes images from a top perspective, and saves these images in a resolution of 400x300, with a .jpg extension.

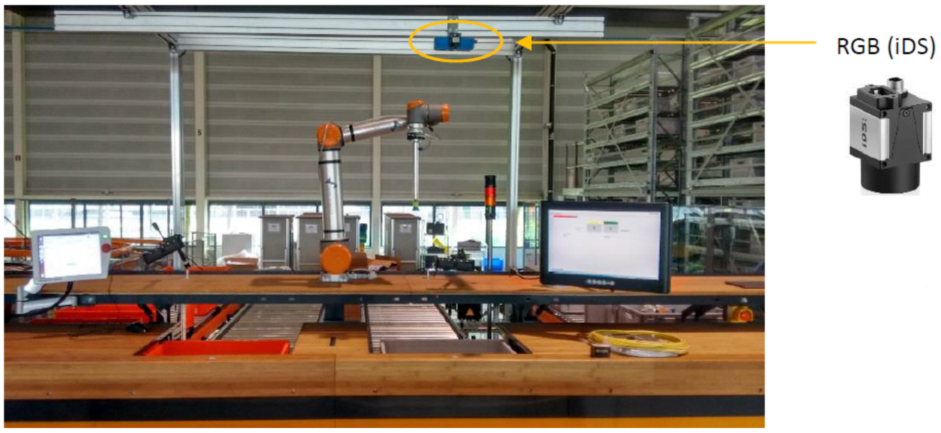


Figure 5-4: Data collection configuration. Container images are taken from the top side with an RGB camera. Items in the containers are placed and repositioned manually.

A dataset is made of **1000 images**, with each images from each item being taken from multiple views, with atleast one image of each face. The labeling strategy is defined as follows:

> **Using the class definition that is defined in chapter 4, label each sample with the number of face where the barcode is located.**

Labeling all samples produces the following file structure (in a Windows enviroment):

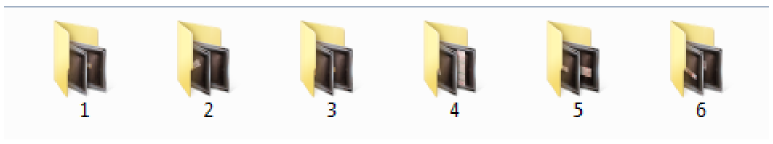


Figure 5-5: Labeled data, with each folder containing numerous data instances belonging to its specific class.

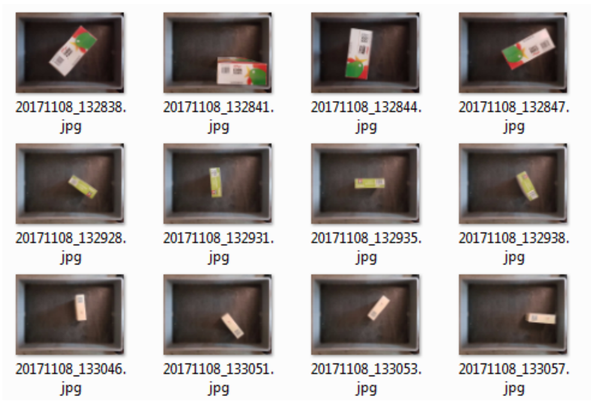
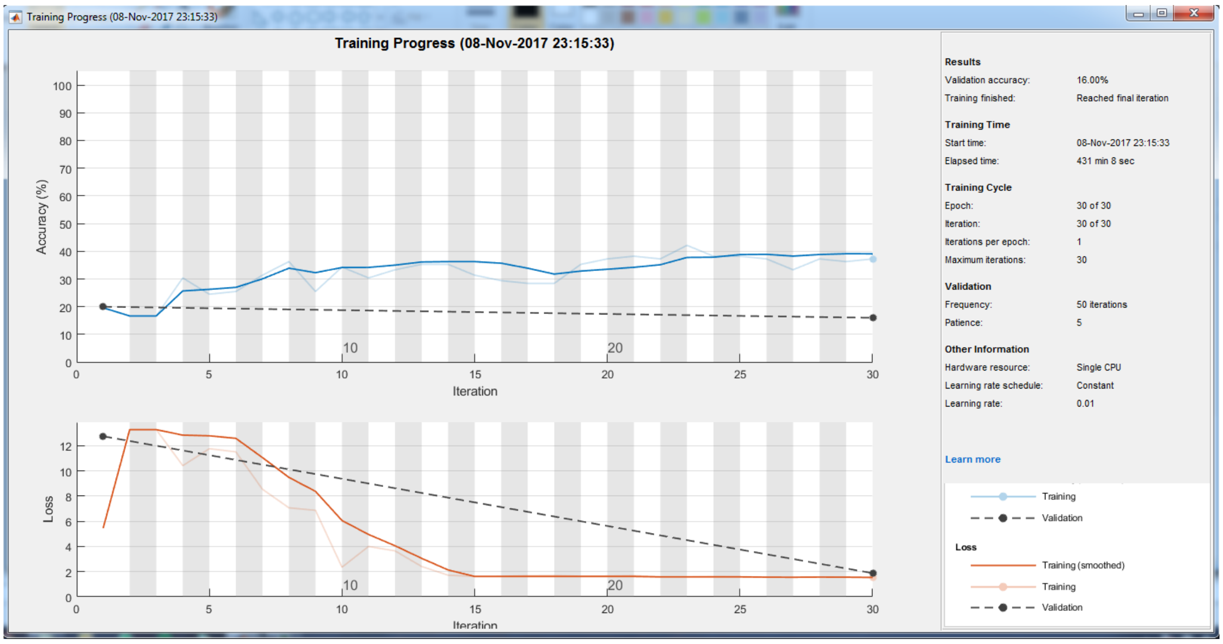


Figure 5-6: Data instances in folder '1'. All instances in this folder are labeled '1', meaning the barcode is on the top plane, as is visible in the photos.

Training the CNN

When importing the labeled dataset, a division of 75%/25% is made between the training data and validation data. This means that 75% of the data is used to train the network, and 25% of the data is used to validate the network. After a sufficient validation accuracy is achieved, an additional test set can be used to see how well the network performs.

The first configuration is ran on a CPU based computation model. The CPU used is an **Intel Core i7 720QM processor**, with four physical and four virtual cores. The training progress and result are shown in figure 5-7:



Results

Validation accuracy:	16.00%
Training finished:	Reached final iteration

Training Time

Start time:	08-Nov-2017 23:15:33
Elapsed time:	431 min 8 sec

Training Cycle

Epoch:	30 of 30
Iteration:	30 of 30
Iterations per epoch:	1
Maximum iterations:	30

Validation

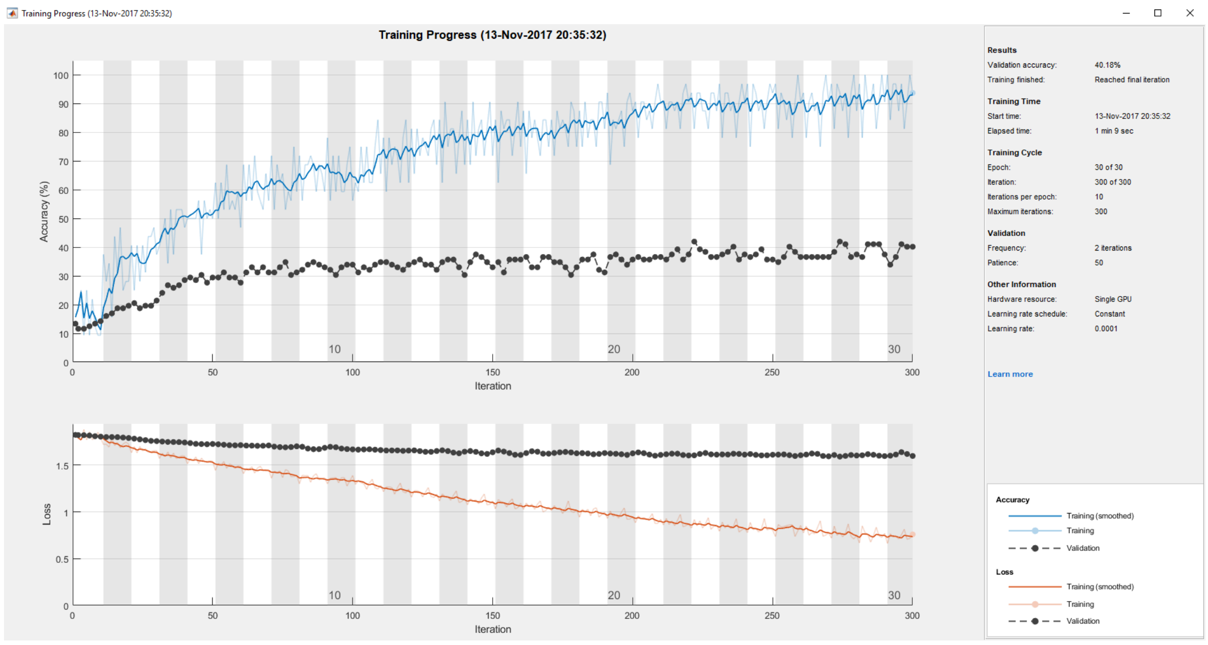
Frequency:	50 iterations
Patience:	5

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.01

Figure 5-7: CNN training run 1, with the use of a CPU. A validation accuracy of 16% is reached with a training time of roughly 431 minutes.

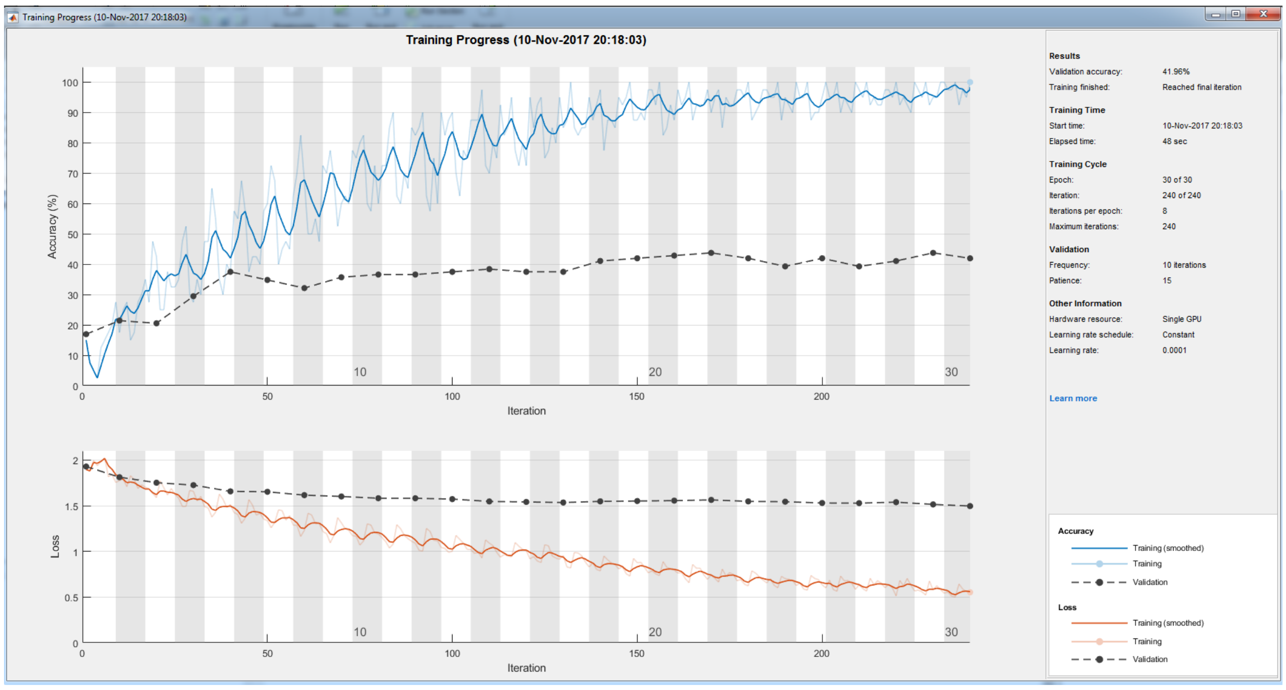
Even though the CPU used has four physical cores, it can be immediately noted by the elapsed time, that training with a CPU still takes an extreme amount of time and is not really feasible for fine-tuning in the scope of this project. Even when a CPU consists of multiple cores and is top of the line, high processing times still remain an issue. Looking at the test results, the validation accuracy of 16% is very insufficient and it looks like the CNN has not learned any valuable features that can be used for correct classification. By altering the amount of features that the convolutional layers calculate, and the filter size that the convolutional layer uses, the system is fine-tuned further to increase validation accuracy. However, from the second run on a GPU based computation is taken. For the following runs, a **NVidia GTX1060** GPU is used and an **Intel i7-6700HQ** CPU is used. The results of the second run are shown in figure 5-8:



Results	
Validation accuracy:	40.18%
Training finished:	Reached final iteration
Training Time	
Start time:	13-Nov-2017 20:35:32
Elapsed time:	1 min 9 sec
Training Cycle	
Epoch:	30 of 30
Iteration:	300 of 300
Iterations per epoch:	10
Maximum iterations:	300
Validation	
Frequency:	2 iterations
Patience:	50
Other Information	
Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Figure 5-8: CNN training run 2, with the use of a GPU. System parameters are already tweaked a bit, by decreasing filter size and number of filters a validation accuracy of 40.18% is reached with a dazzling training time of only 1 minute and 9 seconds.

What can be noted immediately is that the computation time decreased from 431 minutes to about 1 minute!. This means that the computation time decreased with a factor of around 400. This at the very least allows for efficient fine-tuning of the CNN. Looking at the validation results, an accuracy of 40.16% is achieved. While the training set accuracy does increase to around 100%, the validation accuracy keeps oscillating around 40%. Comparing the training loss with the validation loss it can be seen that although the training set does converge to zero, the validation set stays around 1.75. Further finetuning is done to the CNN to investigate whether altering the learning rate, minibatch or epoch size, or layer configurations and settings can improve the CNN even more. This is done numerous times consecutively, showing some of the interesting results in figures 5-9 to 5-11:



Training Progress (10-Nov-2017 20:18:03)

Results

Validation accuracy:	41.96%
Training finished:	Reached final iteration

Training Time

Start time:	10-Nov-2017 20:18:03
Elapsed time:	48 sec

Training Cycle

Epoch:	30 of 30
Iteration:	240 of 240
Iterations per epoch:	8
Maximum iterations:	240

Validation

Frequency:	10 iterations
Patience:	15

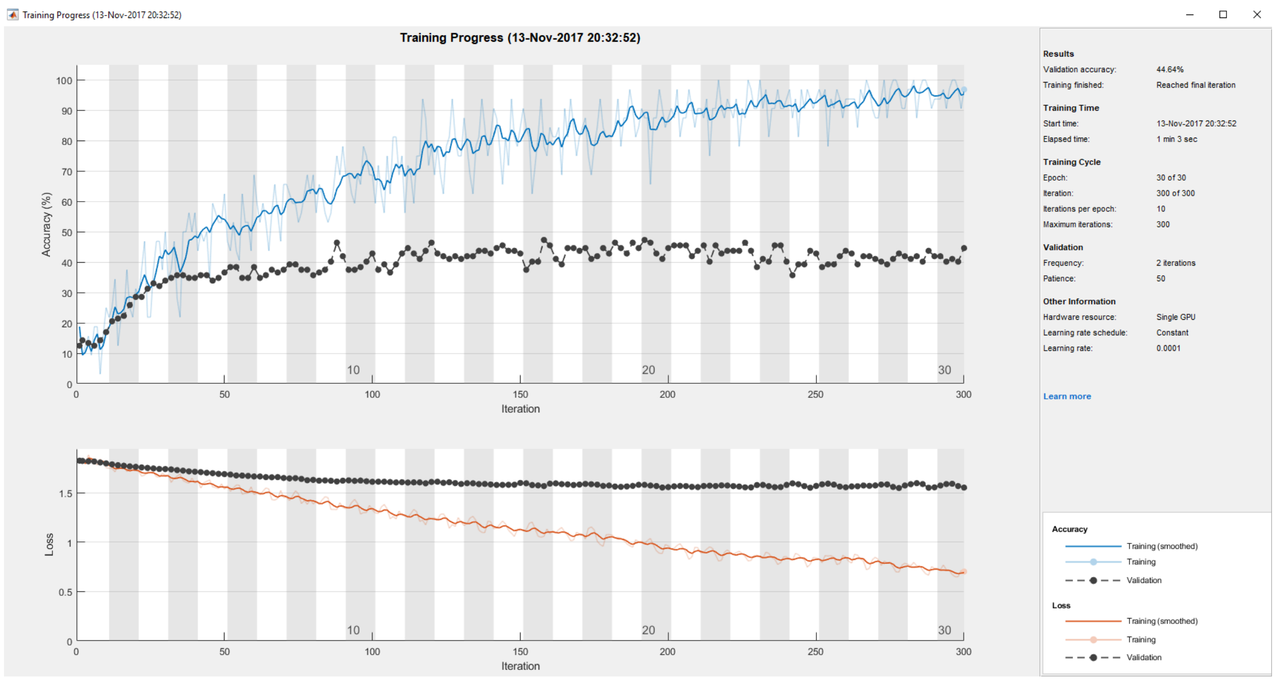
Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Accuracy
Training (smoothed)
Training
Validation

Loss
Training (smoothed)
Training
Validation

Figure 5-9: CNN training run (with a GPU), with altered learning rates and layer filter sizes. A validation accuracy of 41.96% is reached with an training time of 48 seconds.



Training Progress (13-Nov-2017 20:32:52)

Results

Validation accuracy:	44.64%
Training finished:	Reached final iteration

Training Time

Start time:	13-Nov-2017 20:32:52
Elapsed time:	1 min 3 sec

Training Cycle

Epoch:	30 of 30
Iteration:	300 of 300
Iterations per epoch:	10
Maximum iterations:	300

Validation

Frequency:	2 iterations
Patience:	50

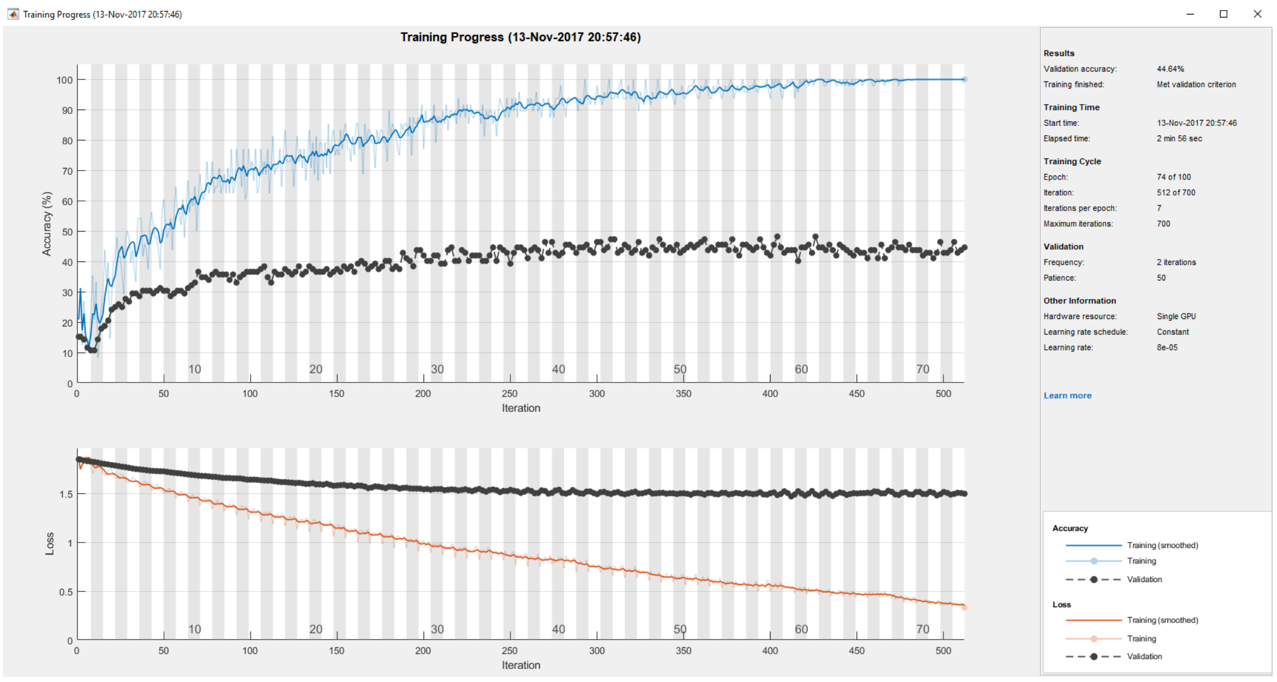
Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Accuracy
Training (smoothed)
Training
Validation

Loss
Training (smoothed)
Training
Validation

Figure 5-10: CNN training run (with a GPU), with altered learning rates and layer filter sizes. A validation accuracy of 44.64% is reached with an training time of 63 seconds.



Results

Validation accuracy:	44.64%
Training finished:	Met validation criterion

Training Time

Start time:	13-Nov-2017 20:57:46
Elapsed time:	2 min 56 sec

Training Cycle

Epoch:	74 of 100
Iteration:	512 of 700
Iterations per epoch:	7
Maximum iterations:	700

Validation

Frequency:	2 iterations
Patience:	50

Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	8e-05

Learn more

Figure 5-11: CNN training run (with a GPU), with altered learning rates and layer filter sizes. A validation accuracy of 44.64% is reached with an training time of 2 minutes and 56 seconds.

After running a couple of dozen tests, it is concluded that the validation accuracy of the data set lies around 45% when the system is fine-tuned. The training set accuracy rises to 100%, but the validation set keeps hovering around 45%. From these results it can be concluded that with the current size of the dataset being around 1000 images, the CNN cannot distinguish enough between the data instances. These results are a perfect illustration of the case where the network does not generalize well enough to not seen data instances. The CNN has overfit the data, as can be seen from the 100% accuracy on the training set, and tweaking the learning rate and batch sizes do not seem to have any additional effect after being finetuned to reach these accuracies of around 45%. The rapid conclusion that can be made is: a larger dataset is needed to train the CNN, so that overfitting is reduced and higher accuracies can be reached.

Chapter 6

Conclusion

Reflecting back on the proposed object recognition framework in light of the defined requirements and constraints, it is concluded that a sufficient accuracy is achieved of 44.64%, higher than the requirement of 40%. In order to reach a framework that robustly automates the rebin phase of the order fulfillment process within Vanderlande, the classification accuracy of the CNN has to be improved. The logical conclusion that could be made from the training results, is that the dataset has to be drastically increased to prevent overfitting of the data. Furthermore datasets of all other GS1US product categories have to be added to the CNN for the framework to be implemented in a Vanderlande warehouse. Although generating and collecting this data might take a lot of time, from the results that have been achieved up until now it can be seen that even with small datasets, promising results are shown.

When training the CNN, GPU implementation is necessary. Training the CNN with merely a CPU implementation can take months to complete for the complete GS1US product category dataset. It is logical to conclude that fine-tuning of the system during this process is not feasible in that time setting. Scalability does not seem to be an issue with the implementation of GPU computation, and furthermore cloud-based computation can be implemented to decrease training time even more, when training very large datasets.

When sufficient CNN classification accuracies are reached, smart picking strategies can be implemented where a picking robot tries to classify products by showing the faces of a product along the classification probability vector. This method ensures immediate picking when the product is classified correctly, and secondary picking in the case of misclassification, where the robot tries to scan the faces according to the classification probability vector until the barcode is found. This ensures minimal to no error costs, since the product is always scanned. In this case, the CNN classification accuracy correlates to how fast products are classified by the barcode scanner in the first try, where improvements in accuracy means faster picking.

Appendix A

MATLAB Code

A-1 Convolutional Neural Network Program

```
%% Loading and Exploring Image Data

Datasetpath = fullfile('C:\Users\EliteBook\Desktop\Labeled Data800x600');
productData = imageDatastore(Datasetpath,...
    'IncludeSubfolders',true,'LabelSource','foldernames');

%% Image and Label Count

labelCount = countEachLabel(productData)

%% Defining Image Size

img = readimage(productData,1);
size(img)

%% Specify Training and Validation Sets

trainNumFiles = 56;       % 65 = 86/14% Train/Val Split, 56 = 75%/25%
[trainDigitData,valDigitData] = splitEachLabel(productData,trainNumFiles,
    'randomize');

% %% Define Network Architecture
% % Define the convolutional neural network architecture.
% layers = [
%     imageInputLayer([150 200 3])
%
%     convolution2dLayer(5,24,'Padding',1)
%     batchNormalizationLayer
%     reluLayer
%     maxPooling2dLayer(2,'Stride',2)
%      convolution2dLayer(5,24,'Padding',1)
%      batchNormalizationLayer
%      reluLayer
%      maxPooling2dLayer(2,'Stride',2)
%
%      convolution2dLayer(3,16,'Padding',1)
%      batchNormalizationLayer
%      reluLayer
%      maxPooling2dLayer(2,'Stride',2)
%      convolution2dLayer(3,8,'Padding',1)
%      batchNormalizationLayer
%      reluLayer
%      %maxPooling2dLayer(2,'Stride',2)
%
% %      convolution2dLayer(7,16,'Padding',1)
% %      batchNormalizationLayer
% %      reluLayer
% %      maxPooling2dLayer(2,'Stride',2)
% %      convolution2dLayer(9,16,'Padding',1)
% %      batchNormalizationLayer
% %      reluLayer
% %      maxPooling2dLayer(2,'Stride',2)
%
% %      convolution2dLayer(9,16,'Padding',1)
% %      batchNormalizationLayer
% %      reluLayer
% %      convolution2dLayer(9,16,'Padding',1)
% %      batchNormalizationLayer
% %      reluLayer
% %      maxPooling2dLayer(2,'Stride',2)
%
%      fullyConnectedLayer(6)
%      softmaxLayer
%      classificationLayer];


%% Defining Neural Network Architecture

layers = [
    imageInputLayer([108 144 3])

    convolution2dLayer(7,32,'Padding',1)
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)
    %%%%%
    convolution2dLayer(5,16,'Padding',1)
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)
    %%%%%
    convolution2dLayer(3,16,'Padding',1)
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,8,'Padding',1)
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(6)
    softmaxLayer
    classificationLayer];

%% Specify Training Options

options = trainingOptions('sgdm',...
    'InitialLearnRate',0.00015, ...          % Default 0.01
    'MaxEpochs',30, ...
    'ValidationFrequency',2,...
    'L2Regularization',0.0001,...            % Default: 0.0001
    'ValidationData',valDigitData,...
    'MiniBatchSize',40,...                   % Default: 128
    'ValidationPatience',50,...              % Default: 5
    'Verbose',false,...
    'ExecutionEnvironment', 'parallel',...
    'Plots','training-progress');
%     'LearnRateSchedule','piecewise',...
%     'LearnRateDropFactor',0.1,...            % Default 0.1
%     'LearnRateDropPeriod',10,...             % Default 10

%% Train Network Using Training Data

net = trainNetwork(trainDigitData,layers,options);


%% Classify Validation Images and Compute Accuracy

predictedLabels = classify(net,valDigitData);
valLabels = valDigitData.Labels;

accuracy = sum(predictedLabels == valLabels)/numel(valLabels)
```

Bibliography

[1] R. Bekkerman, M. Bilenko, J. Langford, *Scaling Up Machine Learning*, Cambridge University Press, January 2012.

[2] Mitchell, T. (1997). *Machine Learning.* McGraw Hill.

[3] Goodfellow, Ian, Yoshua Begnio, and Aaron Courville. 2016. *Deep Learning.* Cambridge: MIT Press.

[4] Hady M.F.A., Schwenker F. (2013). Semi-supervised Learning. In: Bianchini M., Maggini M., Jain L. (eds) *Handbook on Neural Information Processing.* Intelligent Systems Reference Library, vol 49. Springer, Berlin, Heidelberg

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1* (NIPS'12), F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 1. Curran Associates Inc., USA, 1097-1105.

[6] Kotsiantis, S.B. (2007). Supervised Machine Learning: A Review of Classification Techniques. Informatica 31:249-268.

[7] R. A. Fisher (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics.* 7 (2): 179-188

[8] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard (2015). Multimodal Deep Learning for Robust RGB-D Object Recognition. In: *Proceedings of 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 681âĂŞ687.

[9] B. Baharudin, L.H. Lee, K. Khan (2010). A review of machine learning algorithms for textdocuments classification, In: *Journal of Advances in Information Technology* 1:4âĂŞ20.

[10] Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P.S., Zhou, Z.H.,Steinbach, M., Hand, D. J., and Steinberg, D.

2007. Top 10 Algorithms in Data Mining, *Knowledge and Information Systems* (14:1), pp. 1-37.

[11] Wettschereck, D., Aha, D. W. & Mohri, T. (1997). A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms. In: *Artificial Intelligence Review* 10:1âĂŞ37.

[12] Mooney RJ (1996) Comparative experiments on disambiguating word senses: an illustration of the role of bias in machine learning. In: *Proceedings of the conference on Empire methods in national language processing*, pp 82âĂŞ91.

[13] Wolpert DH (1992) On the connection between in-sample testing and generalization error. *Complex Systems* 6:47âĂŞ94.

[14] D. H. Wolpert and W. G. Macready. (1997). No free lunch theorems for optimization. *Trans. Evol. Comp* 1, 1, 67-82.

[15] Ho YC, Pepyne DL (2002) Simple explanation of the no free lunch theorem of optimization. *Cybern Syst Anal* 38(2):4409âĂŞ4414.

[16] Murthy, (1998), Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey, *Data Mining and Knowledge Discovery* 2:345âĂŞ389.

[17] Quinlan, J.R. (1993). C4.5: Programs for machine learning. Morgan Kaufmann, San Francisco

[18] Quinlan, J.R. (1979), "Discovering rules by induction from large collections of examples", D. Michie ed., *Expert Systems in the Microelectronic age*, pp. 168-201.

[19] Breiman L., Friedman J.H., Olshen R.A., Stone C.J. (1984) Classification and Regression Trees, Wadsforth International Group.

[20] Hunt E., Martin J & Stone P. (1966), Experiments in Induction, New York, Academic Press.

[21] Kononenko I. (1994) Estimating attributes: Analysis and extensions of RELIEF. In: Bergadano F., De Raedt L. (eds) *Machine Learning*: ECML-94. ECML 1994.

[22] Scholkopf, C., Burges, J. C. & Smola, A. J.(1999). *Advances in Kernel Methods.* MIT Press.

[23] Vapnik, V. (1995). The Nature of Statistical Learning Theory. New York: Springer.

[24] Neocleous, C. & Schizas, C., (2002), Artificial Neural Network Learning: A Comparative Review, LNAI 2308, pp. 300âĂŞ313, Springer.

[25] JÃijrgen Schmidhuber, Deep learning in neural networks: An overview, In: *Neural Net-*

works, Volume 61, 2015, Pages 85-117.

[26] Weigend, A. S., Rumelhart, D. E., & Huberman, B. A. (1991). Generalization by weight-elimination with application to forecasting. In: R. P. Lippmann, J. Moody, & D. S. Touretzky (eds.), *Advances in Neural Information Processing Systems 3*, San Mateo, CA: Morgan Kaufmann.

[27] Siddique, M. N. H. and Tokhi, M. O. (2001), Training Neural Networks: Backpropagation vs. Genetic Algorithms, *IEEE International Joint Conference on Neural Networks*, Vol. 4, pp. 2673âĂŞ2678.

[28] Kuramochi M, Karypis G (2005), Gene Classification using Expression Profiles: A Feasability Study. *Int J Artif Intell Tools* 14(4):641-660.

[29] Tjen-Sien, L., Wei-Yin, L., Yu-Shan, S. (2000). A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms. *Machine Learning 40*: 203âĂŞ228.

[30] Ruggieri, S. (2001). Efficient C4.5. *IEEE Transactions on Knowledge and Data Engineering* 14 (2): 438-444.

[31] Olcay Taner YÄśldÄśz, Onur Dikmen (2007), Parallel univariate decision trees, Pattern Recognition Letters, Volume 28 , Issue 7 (May 2007), Pages: 825-832.

[32] Anil K. Jain, Jianchang Mao, and K.M Mohiuddin (1996), Artificial Neural Networks: A Tutorial, *IEEE Computer Society, Volume 29, Number 3*

[33] Barron, A.R. (1993), âĂIJUniversal Approximation Bounds for Superpositions of Sigmoidal Functions,âĂİ *IEEE Transactions on Information Theory*, Vol. 39.

[34] Haykin, S. (1999), Neural Networks: A Comprehensive Foundation, 2nd Edition, Prentice-Hall.

[35] Platt, J. (1999). Using sparseness and analytic QP to speed training of support vector machines. In Kearns, M., Solla, S. & Cohn, D. (ed.), *Advances in neural information processing systems.* MIT Press.

[36] Lee, D., Park, J., Shim, J., Lee, S.-g. (2010): An efficient similarity join algorithm with cosine similarity predicate. In: *Bringas, P.G., Hameurlain, A., Quirchmayr, G. (eds.) DEXA 2010, Part II. LNCS, vol. 6262*, pp. 422âĂŞ436. Springer, Heidelberg

[37] Park, Y., Park, S., Lee, S.-g., & Jung, W. (2014). Greedy filtering: A scalable algorithm for k-nearest neighbor graph construction. In: *Database systems for advanced applications* (pp. 327âĂŞ341). Springer.

[38] Miyamoto D, Hazeyama H, Kadobayashi Y (2008) An evaluation of machine learning-based methods for detection of phishing sites. *Aus J Intell Inf Process Syst* 10(2):54âĂŞ63

[39] Rojas-Bello RN, Lago-FernÃąndez LF, MartÃŋnez-MuÃśoz G, SÃąnchez-MontaÃśÃĪs MA (2011) A comparison of techniques for robust gender recognition. *IEEE Int Conf Image Process* 569âĂŞ572

[40] Aruna S, Rajagopalan SP, Nandakishore LV (2011). An empirical comparison of supervised learning algorithms in disease detection. *Int J Inf Technol Converg Serv* 1:81âĂŞ92

[41] Mathworks (2017). Introducing Deep Learning with MATLAB. *80789v00*

[42] Li, Fei-Fei, and Justin Johnson. 'CS231n: Convolutional Neural Networks for Visual Recognition.' CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University, cs231n.github.io/.