Classifying Retail Store Cabinets with Missing or Misplaced Products Using Verification Learning

S. D. Bonhof

MSc. Thesis
Vehicle Engineering
TU Delft
Student id: 4453107
Classifying Retail Store Cabinets with Missing or Misplaced Products Using Verification Learning

by

S. D. Bonhof

to obtain the degree of Master of Science in Mechanical Engineering at the Delft University of Technology, to be defended publicly on Friday October 30, 2020 at 11:30 AM.

Student number: 4453107
Thesis committee: Dr. Ing. J. Kober, TU Delft, chair
Dr. Ir. C. H. Corbato, TU Delft, supervisor
Ir. C. Pezzato, TU Delft

A fourth committee member will be present, but is not yet known at this time.
An electronic version of this thesis is available at http://repository.tudelft.nl/.
Abstract

Performing tasks in dynamic environments is still an open challenge in robotics. To be able to perform a task reliably in such scenarios, the state of the world has to be continuously monitored. In this context, most state-of-the-art perception methods focus on the recognition and classification of individual objects. However, these methods require extensive data collection and artificial neural network training, especially in complex scenes when the number of unique objects to recognise is large. This is for instance the case of retail stores, where there can be as many as 120,000 different products. Applying the state-of-the-art learning methods in this domain is not only expensive in terms of data gathering, but it will also require models so complex that product recognition would be significantly slow. This research tackles the problem of cabinet classification in a retail store, introducing a method to identify cabinets with missing or misplaced products without individual object recognition. Prior knowledge on the layout of the retail store is used to generate an image of what a cabinet is supposed to look like when it is correctly stocked. Taking an image of the current state of the cabinet and comparing this to the previously created image allows for a verification network to verify whether the cabinet is still fully and correctly stocked or not. This research provides three main results. First, verification learning is demonstrated to transfer well to the retail store cabinet domain, maintaining high speed and accuracy. Second, this work shows that the verification network generalises well to both unseen cabinet configurations as well as unseen products, eliminating the need to include every product in the dataset used to train the network. Lastly, this research shows that verification learning transfers well from simulation to the real world to classify cabinets with missing products. However, this last result does not hold for cabinets with misplaced products, due to the smaller difference between a correctly stocked cabinet image and an incorrectly stocked cabinet image. Furthermore, while the verification network is very fast on the hardware used for this research, it will be significantly slower when applied on the less powerful hardware more commonly found in robots. This thesis represents a starting point for the detection of missing and misplaced products in retail store products, and it serves as a foundation for future research in this domain.
### Nomenclature

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>FC</td>
<td>Fully Connected (layer in a CNN)</td>
</tr>
<tr>
<td>MCC</td>
<td>Matthew’s Correlation Coefficient</td>
</tr>
<tr>
<td>MSP</td>
<td>Misplaced products</td>
</tr>
<tr>
<td>MSPTMN</td>
<td>Dataset for training featuring images of cabinets with misplaced products</td>
</tr>
<tr>
<td>MSPTST</td>
<td>Dataset for testing featuring images of cabinets with misplaced products</td>
</tr>
<tr>
<td>MSPTST#</td>
<td>Dataset for testing featuring images of cabinets with misplaced products where the # represents the fraction of unseen products in the dataset (e.g. MSPTST100 for 100% unseen products)</td>
</tr>
<tr>
<td>MSS</td>
<td>Missing products</td>
</tr>
<tr>
<td>MSSTRN</td>
<td>Dataset for training featuring images of cabinets with missing products</td>
</tr>
<tr>
<td>MSSTST</td>
<td>Dataset for testing featuring images of cabinets with missing products</td>
</tr>
<tr>
<td>MSSTST#</td>
<td>Dataset for testing featuring images of cabinets with missing products where the # represents the fraction of unseen products in the dataset (e.g. MSSTST100 for 100% unseen products)</td>
</tr>
<tr>
<td>RGB</td>
<td>Red Green Blue (values that determine pixel color in an image)</td>
</tr>
<tr>
<td>VN</td>
<td>Verification Network</td>
</tr>
</tbody>
</table>
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>1</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Research Questions</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Goals</td>
<td>3</td>
</tr>
<tr>
<td>1.3.1 Functional Requirements</td>
<td>3</td>
</tr>
<tr>
<td>1.4 Thesis Structure</td>
<td>4</td>
</tr>
<tr>
<td>2 Related Work</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Individual Object Recognition</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1 Template Matching</td>
<td>5</td>
</tr>
<tr>
<td>2.1.2 Feature Matching</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Deep Learning</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Transferring to Cabinet Analysis</td>
<td>7</td>
</tr>
<tr>
<td>3 Verification Learning</td>
<td>9</td>
</tr>
<tr>
<td>3.1 Deep Verification Networks</td>
<td>9</td>
</tr>
<tr>
<td>3.1.1 Input Triplets</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Convolutional Neural Networks</td>
<td>10</td>
</tr>
<tr>
<td>3.2.1 Convolutional Layers</td>
<td>10</td>
</tr>
<tr>
<td>3.2.2 Fully Connected Layers</td>
<td>11</td>
</tr>
<tr>
<td>3.3 Training and Testing</td>
<td>11</td>
</tr>
<tr>
<td>3.3.1 Training Epochs</td>
<td>12</td>
</tr>
<tr>
<td>3.3.2 Model Parameter Initialisation</td>
<td>12</td>
</tr>
<tr>
<td>3.3.3 Network loss &amp; optimisation</td>
<td>12</td>
</tr>
<tr>
<td>3.3.4 Parameter Update Using Backpropagation</td>
<td>13</td>
</tr>
<tr>
<td>3.3.5 Data</td>
<td>13</td>
</tr>
<tr>
<td>3.4 Evaluation</td>
<td>14</td>
</tr>
<tr>
<td>4 Network Design &amp; Data Generation</td>
<td>17</td>
</tr>
<tr>
<td>4.1 Data Generation &amp; Preprocessing</td>
<td>17</td>
</tr>
<tr>
<td>4.1.1 Product Models</td>
<td>17</td>
</tr>
<tr>
<td>4.1.2 Cabinet Generation</td>
<td>18</td>
</tr>
<tr>
<td>4.1.3 Dataset Naming</td>
<td>20</td>
</tr>
<tr>
<td>4.1.4 Triplet Creation</td>
<td>21</td>
</tr>
<tr>
<td>4.1.5 Preprocessing</td>
<td>21</td>
</tr>
<tr>
<td>4.2 Network Design</td>
<td>21</td>
</tr>
<tr>
<td>4.2.1 Single Image Input - CNN</td>
<td>21</td>
</tr>
<tr>
<td>4.2.2 Multiple Image Input - Verification Network</td>
<td>22</td>
</tr>
<tr>
<td>4.3 Hardware &amp; Software.</td>
<td>24</td>
</tr>
<tr>
<td>4.4 Summary</td>
<td>24</td>
</tr>
<tr>
<td>5 Experiments &amp; Results</td>
<td>25</td>
</tr>
<tr>
<td>5.1 Q1: Missing Products</td>
<td>25</td>
</tr>
<tr>
<td>5.1.1 CNN</td>
<td>26</td>
</tr>
<tr>
<td>5.1.2 Verification Network</td>
<td>27</td>
</tr>
<tr>
<td>5.1.3 Comparison</td>
<td>28</td>
</tr>
</tbody>
</table>
5.2 Q2: Misplaced Products .................................................. 29
  5.2.1 CNN ................................................................. 30
  5.2.2 Verification Network .................................................. 31
  5.2.3 Comparison .......................................................... 32
5.3 Q3: Generalisability .......................................................... 33
  5.3.1 Missing Products .................................................... 34
  5.3.2 Misplaced Products ................................................... 35
5.4 Q4: Reality Gap ............................................................... 36
  5.4.1 Missing Products .................................................... 37
  5.4.2 Misplaced Products ................................................... 38
5.5 Q5: Speed ................................................................. 40
5.6 Sensitivity Analysis .......................................................... 41
  5.6.1 Dataset Size .......................................................... 41
  5.6.2 Network Architecture .................................................. 42
  5.6.3 Increasing Cabinet Noise Levels in Simulation ................. 43
5.7 Discussion ................................................................... 44
  5.7.1 Experiments ............................................................ 44
  5.7.2 General Points ........................................................ 45

6 Conclusion & Future Work ......................................................... 47
  6.1 Conclusion ............................................................... 47
  6.2 Future Work .............................................................. 48
    6.2.1 Closing the Reality Gap .............................................. 48
    6.2.2 Network Optimisation ............................................... 48
    6.2.3 Include Non-rectangular Products ................................ 48
    6.2.4 Completing the Pipeline ............................................ 49
    6.2.5 Include Reasoning to Set a Similarity Threshold .......... 49
    6.2.6 Multiple Product Rows ............................................. 49
    6.2.7 Research Product Model Set Size ............................... 49
    6.2.8 Combine Missing and Misplaced Products ................... 49

Acknowledgements ................................................................. 49

Bibliography ........................................................................ 52

A Product Distributions Generalisability Datasets ......................... 57
  A.1 Missing Products ....................................................... 58
    A.1.1 MSPTRN ............................................................. 58
    A.1.2 MSPTST0 ............................................................ 59
    A.1.3 MSPTST20 ........................................................... 60
    A.1.4 MSPTST40 ........................................................... 61
    A.1.5 MSPTST60 ............................................................ 62
    A.1.6 MSPTST80 ............................................................ 63
    A.1.7 MSPTST100 ........................................................... 64
  A.2 Misplaced Products ........................................................ 65
    A.2.1 MSSTRN ............................................................. 65
    A.2.2 MSSTST0 ............................................................ 66
    A.2.3 MSSTST20 ........................................................... 67
    A.2.4 MSSTST40 ........................................................... 68
    A.2.5 MSSTST60 ............................................................ 69
    A.2.6 MSSTST80 ........................................................... 70
    A.2.7 MSSTST100 ........................................................... 71
Introduction

In a supermarket - or any retail store - the products are at the centre of importance. Their state is constantly monitored by employees who make sure that the products are stocked in time and that they are placed in a way that makes it easy for customers to find and recognise them. Deploying robots in retail stores allows for the automation of repetitive tasks. Monitoring the shelves for misplaced or missing products and replacing these products is a good example of such a task.

Placing misplaced products back where they belong might be simple for humans, as it involves skills that are so simple for people to perform that they do not even think about it. A simplified pipeline of an object manipulation task with some of these basic skills can be seen in Figure 1.1. From this pipeline, it can be seen that object recognition is an essential skill for a robot to be able to perform a task, as it is the first step in completing the task. If a product can not be recognised as misplaced then it will not trigger the robot to place it back where it belongs.

**TASK:** Return misplaced products

- Recognise that a product is misplaced
- Object recognition
- Pose estimation
- Grasping
- Moving
- Placing
- Determine the pose of the misplaced product
- Pick up the product using the end-effector
- Move the product using an actuated arm and movable base
- Place the product on the shelf where it belongs

![Figure 1.1: High-level steps to be taken to complete a simple object manipulation task [4].](image)

1.1. Motivation

While recognising misplaced or missing products is trivial for humans to do, it is a rather complex perception task for robotics. Most state-of-the-art perception algorithms focus on individual object recognition, which tends to decrease in speed and performance as the number of objects to be classified grows [19]. This is an issue in retail store environments as large Dutch supermarkets can feature up to unique 30,000 products [1], and the average superstore in the U.S. features about 120,000 different products [5]. However, in a retail store, there is a lot of information available about the environment that is not used in most individual object recognition methods. For example, the location of where each product is supposed to be is easily available, but it is not often used to aid vision directly. Utilising this prior knowledge might improve both the accuracy and speed of the detection of missing or misplaced products. The speed of cabinet analysis is important, especially in retail stores that have a lot of customer traffic through the aisles, since the robot might be occupying space that customers want to be in.

This research introduces a method that uses prior knowledge of the retail store to create a referent image and compare this referent image to the real world, thus eliminating the need for individual object recognition. The hypothesis is that by eliminating the speed and accuracy constraints imposed by object recognition, this new method will be able to find misplaced and missing products faster and with higher accuracy than individual object recognition methods. By comparing an image of what a cabinet
is supposed to look like to an image of what it looks like right now, misplaced products can be found. This is not always possible with a single image network since it uses no prior knowledge of the correct location of products, so it could never ‘know’ that a product is not supposed to be where it is right now. Comparing the current state of a cabinet to this referent image means that the cabinet state and thus the product states only has to be verified (which is a binary classification - the outcome can be ‘correct’ or ‘incorrect’) instead of classified as one of thousands of classes with individual object recognition. While this research will focus only on the verification of whole cabinets, the whole system as can be used to classify individual products as missing or misplaced is laid out in Figure 1.2. First, an image of a whole cabinet is classified as being ‘full’ or ‘missing’/‘misplaced’. The latter means that there are one or more products missing or misplaced on one or more of the shelves in the cabinet. When products are missing or misplaced in a cabinet, the system can then ‘zoom in’ on the individual shelves, and if a shelf has an anomaly, only then will the system compare the real world and the referent at product level. In Figure 1.2, the blue blocks are to be designed during this research, focusing on the ‘Analyse whole cabinet’ block, since this is the first step in the pipeline and also the most complex task (since the network has to compare somewhere between 50-150 products in one image instead of 5-20 products for a single shelf in the second block, or 1 product for the last block).

![Figure 1.2: Diagram of the pipeline for missing or misplaced product detection throughout a retail store. During this research the process of the first blue block ('Analyse whole cabinet', blue outline) will be developed. This process is similar to those in the other blue blocks (with red outline), but simply at another image scale (zoomed in on the cabinet shelves or products).](image)

Comparing two images can be done in several ways. The images can be subtracted, added, multiplied, etc. However, these static methods would be quite sensitive to image noise such as lighting conditions, camera angles, or precise locations of the products on the shelves. Thus, this research compares the images using a verification network. Verification networks have proven effective in facial recognition tasks [20, 42]. This research will show how well verification learning performs on the cabinet classification task.

In this research, the verification learning approach will be compared to a single image input approach in the form of a convolutional neural network with a single image as input and binary output. This will show how adding prior knowledge to the system improves performance. The verification network will not be compared to individual object recognition methods in terms of performance as it is not feasible to test its performance on a very large number of classes (30,000) without relevant data.

Using a deep learning approach means that a lot of data is required. Since gathering real data is very expensive, a cabinet generator is designed and implemented that takes 3D product models and creates RGB images of cabinets in different states and configurations. To limit the time needed to 3D model products only rectangular products are considered. Since the classification involves whole cabinets, only a relatively small product model set is needed to generate a large number of cabinets in different states and configurations. So, generating enough data through simulation is feasible.

### 1.2. Research Questions

The research question that will be addressed in this thesis is:

*How can prior knowledge of the correct location of products in a cabinet help to detect missing or misplaced products in a retail store?*

This question raises some subquestions which will be used as a guideline to eventually answer the main research question:

1. Can the **absence** of products be detected more reliably and quickly by comparing current cabinet state images with a referent image than with classification using a single image?
2. Can the misplacement of products be detected more reliably and quickly by comparing current cabinet state images with a referent image than with classification using a single image?

3. How generalisable is this approach to new products as opposed to classification without prior knowledge?

4. How generalisable is this approach to real-world data when it is trained purely on simulated data?

5. How does the verification network improve the overall speed with respect to individual object classification?

1.3. Goals

This research has several goals. The first goal is to improve cabinet state classification using verification learning over using a single image input method by adding prior knowledge. Achieving high performance using the verification learning method would also mean that implementing and testing an individual object recognition method might not be necessary, saving a lot of effort that would - in that case - not necessarily have resulted in better performance.

The second goal is to achieve high-speed cabinet classification. Improving classification speed means that the robot will be able to perform its tasks quicker since it doesn’t have to stand still and wait until it has identified the missing or misplaced products for too long.

Another goal is to make the reality gap as small as possible. The reality gap - which occurs when an artificial neural network is trained on simulated data and tested on real data - will prevent the network to transfer from the simulation to the real world. The goal is to close this gap by varying external variables in the simulation such as light, product displacements and shelf heights as much as possible.

The last goal is to make the system generalisable to new cabinet configurations and new products. If this goal is achieved then the system will be able to be used in new environments where it might encounter products and cabinet configurations that it has not been trained on.

1.3.1. Functional Requirements

In general, there are a few requirements that the system has to meet to be useful. The following requirements are set for the system:

1. The system should work on a large number of different products;
   - Retail stores can feature up to 30,000 different products.

2. The system should be able to analyse a cabinet within seconds;
   - Taking a lot of time to analyse a cabinet means a lot of downtime for a robot to be standing still in a busy retail store aisle.

3. The system should be able to analyse a cabinet regardless of camera orientation, lighting direction and colour, and to a certain extent, intensity;
   - Retail stores are always well-lit, but the light sources are not consistent. Light can come from different directions and can have a slightly different colour. Furthermore, photographs of cabinets can be taken from slightly different angles, resulting in slightly different images.

4. The system should be able to generalise to cabinets containing products it has not seen before without the need for retraining;
   - Being able to use the system without retuning or retraining in a new environment or on new products means that the system is much easier to deploy in a different retail store. Furthermore, if new products are introduced in a store, the system will be able to handle them.

To limit the scope of this research, the following assumptions have been made:

1. The robot has an RGB camera;
2. The robot can point the RGB camera so that it fully captures a cabinet;
3. The cabinets are well-lit;
4. The correct product for each location in each cabinet is known OR; A referent image is available in the knowledge base;
5. The number of products missing or misplaced in a cabinet can vary. Cabinets with few missing/misplaced products will occur less often than cabinets where more products are missing/misplaced, since the missing and misplaced products are constantly corrected during a day in a store;
6. Products in a cabinet are positioned right-side-up;
7. Products in a cabinet are facing the front.

1.4. Thesis Structure
This thesis starts by introducing some related work in Chapter 2. Then, some background knowledge on deep learning and verification learning is given in Chapter 3, followed by the network design and data generation in Chapter 4. To answer the research questions, Chapter 5 lays out the designed experiments, followed by the results of those experiments. Finally, the thesis is concluded and recommendations for future work are given in Chapter 6.
Related Work

Object recognition is a very broad topic in the field of computer vision. To narrow down the related literature this research will focus on the recognition of objects on a very large scale, in terms of both the number of classes and number of objects to recognise in a single task.

2.1. Individual Object Recognition

Within individual object recognition, template matching and feature matching are popular methods that store information of object classes to later use this information to recognise an object by respectively overlaying templates or matching feature keypoints for each class separately. Template and feature matching are methods of individual object recognition.

2.1.1. Template Matching

Template matching relies on the creating of templates of the objects that have to be recognised. A template is often simply a 2D image of an object from a certain angle. From this image corners and edges can be extracted which are then matched to the object to be recognised by shifting, rotating and scaling the template over the unidentified object. The best fit produces a distance measure. Once all stored templates are matched to the object, the object is identified as the object belonging to the template that achieved the lowest distance measure. An example of template matching can be seen in Figure 2.1. This method requires templates to be stored of every object to be recognised, and distance measures to be calculated over those templates. This need for matching a large number of templates on different rotations, scales and positions in an image results in a method with a very large overhead in terms of computing speed and memory. Even template matching methods highly optimised for speed (such as the one proposed by Wu and Toet [48]) take between 0.94 ms and 1.59 ms to match a template to part of an image. Considering the number of templates needed to be checked before an object might have been found this method may not be fast enough for object recognition in a retail store. Furthermore, due to its global approach, the whole template should match the object as well as possible, meaning that this approach is not well suited for products that are deformable, such as bags of chips.
2.1.2. Feature Matching

2.1.2. Feature Matching

Feature matching is based on keypoint features which are extracted from known objects and matched to unknown objects. A keypoint is a point in an image which is easy to recognise in a new image. For example in a white image with a black square, a keypoint in the centre of the black square is useless since those pixels are equal everywhere in the square, while a keypoint on the corner is useful since there is only one of those corners in the image. The keypoints have a description that acts as a fingerprint. This description (or ‘descriptor’) is the driving power of feature matching. If a keypoint has a clear description, it will be easier to match to another keypoint than if the description was not clear or unique to that keypoint. One of the most popular descriptors is SIFT (Scale Invariant Feature Transform) [31]. This descriptor takes a window of pixels around a keypoint and describes the keypoint using the gradient magnitudes and orientations of the surrounding pixels. The gradients are then accumulated, resulting in a description vector. This description vector can then be matched to other description vectors. To be able to detect and recognise objects, many keypoints from those objects have to be stored. More keypoints to match means a higher chance of recognising an object, since there is no guarantee that all the same keypoints will be found on an unidentified object when trying to recognise is as when that object was introduced in the set of saved objects with keypoints. Different keypoint descriptors have been developed to increase both speed and descriptiveness. The ORB (Oriented FAST and Rotated BRIEF) descriptor proposed by Rublee et al. [34] is orders of magnitude faster than its counter descriptors (SIFT [31] and SURF [3]) but still takes on average 15.3 ms to detect and compute $2 \times 10^6$ features in a $640 \times 480$ image. After detection, matching these features takes another 17ms, making for a total of over 30 ms. While the authors mention that these times can be sped up considerably by not matching all the query descriptors to the training data, it does give us a good idea on what to expect in terms of speed when considering a large object class set size and a large number of objects to recognise. Due to the local approach of this method (i.e. only small, local parts of the image are being matched), this method is applicable to deformable objects as well, although it is less reliable since keypoint outlier removal methods such as RANSAC (Random Sample Consensus) [38] cannot be applied when the spatial characteristics of the keypoints are not preserved.
2.2. Deep Learning

Recently, deep learning approaches have outperformed many other object recognition methods. Some deep neural networks even outperform humans on certain databases [18]. It has to be noted that the tested dataset is criticised to feature objects that are too far away from everyday human life to get a relatable human performance measure [32]. Nevertheless, their performance is very impressive and very useful in many scenarios such as Autonomous vehicles [24], Facial recognition [21, 42], detecting fake news [35], and many others.

Most deep learning architectures designed for the processing of images make use of convolutional layers to form a Convolutional Neural Network (CNN). Starting in 1998 with LeNet-5 [30] featuring a total of 5 layers. Since then, CNNs have become more and more complex featuring more layers and more model parameters, to be able to approximate more and more complex functions. He et al. [19] allowed the creation of ‘ResNet’ networks up to 1001 layers by adding skip connections to their network, preventing overfitting and common gradient problems. With the same idea in mind, Iandola et al. [25] created an architecture that propagates layer outputs even further into the network, preserving information from the earlier layer outputs to be used in the latter layers. This allowed for the creation of complex networks that feature fewer parameters than ResNet while including more layers and achieving better performance.

For computer vision, many deep learning approaches focus on the recognition of a single object in an image, or the detection and subsequent recognition of multiple objects in one image, both without prior knowledge. Another approach would be to take multiple images and compare these to produce a measure of their similarity. This is called Deep Verification Learning and it is most often used for face verification [22, 40, 41, 43, 51]. This method depends on training a network to place objects of the same class closely together in the feature space and placing objects from different classes as far away as possible. It has been shown that increasing intra-class variability while decreasing the inter-class variability can improve the generalisation of a trained model [47]. An example of a verification network can be seen in Figure 2.3. Instead of simply comparing individual objects, deep learning could also be used to compare larger images, comprising of multiple objects.

2.3. Transferring to Cabinet Analysis

To achieve the goal of this research, a system will be created utilising prior knowledge of the store layout to generate images, that will then be used in a verification network to compare with real photos of retail store cabinets. This means that the method used for facial verification will be transferred to the domain of retail store cabinets. Analysing two cabinets, the same parts of a cabinet must be compared, i.e. the product in the top left corner of the cabinet with an unknown state should be compared to the product in

Figure 2.2: An example of feature matching in a retail store. Here, keypoints are identified in the image on the right and matched with identified keypoints in the image on the left. Keypoints are matched when their descriptions are very similar.
2. Related Work

Figure 2.3: Example of a verification network used for facial recognition. Two images of faces are put through several convolutional layers resulting in a feature vector. This feature vector describes the most important characteristics of the images needed to compare them. The feature vectors are subtracted from one another and put through a fully connected layer that outputs the chance for two classes; genuine pairs and imposter pairs. The chance for a genuine pair is taken as similarity value, where a value of 1 means that the two input images are the same person, and a value of 0 means that the two input images are not the same person [20].

The top left corner of the cabinet it is being compared to. The same spatial importance is present in facial recognition. Sun et al. [40] first align faces that will be compared by recognising the eyes and mouth and similarity transform the images based on those three points. They also feature several convolutional channels in their deep neural architecture to compare different parts of the face separately. These parts are also compared when flipped to account for gaze direction of the person in the image. This indicates that comparing smaller regions of a larger image results in better comparisons. Comparing cabinets is easier in the sense that the products will always face approximately the same direction when placed correctly, and, when the cabinet itself is extracted from an image and perspective transformed to fit the image, each part of the cabinet always occupies the same space in the image as the cabinet it is being compared to. So, while flipping images of cabinets is not necessary (in fact, it is undesirable), comparing different, smaller regions of a cabinet might help the comparison due to lower complexity in the image.
Verification Learning

Now that the related work is introduced, this chapter will go more in-depth on deep verification learning. It is important to understand how a model is found which compares two images best, as this will highlight which parts of the system can be tuned or changed to improve the performance. This chapter will start by illustrating a verification network, and continue by explaining the different parts of the network and some of the mathematics that power it.

3.1. Deep Verification Networks

Deep learning is used to extract feature vectors that describe the most prominent features of an input that allow for the highest intra-class variability while keeping the inter-class variability low. This makes it easy to distinguish between inputs with the same characteristics and inputs with different characteristics.

For image classification using a single image, this feature vector would be passed to an additional fully-connected layer (as explained in Section 3.2.2) and a softmax layer which outputs a vector with the length of the number of classes in which every value is in the range \([0,1]\) and all values sum up to 1.

For classification using multiple images, the feature vectors coming from the network can be used to get a quantitative measure of similarity between the inputs. The feature vectors can be compared in different ways. They can be compared directly using, for example, their Euclidean or Mahalanobis distance \([10]\), but they can also be compared using a learned metric which takes the feature vector of two images and learns a metric which compares both feature vectors to produce a single output value. However, this could cause the output not to be properly normalised, exceeding the \([0,1]\) range. To mitigate this, Chen et al. \([6]\) proposed a metric learning layer which produces two outputs, which are normalised using a softmax layer to be within the range \([0,1]\). From these two outputs, the output value of the inputs belonging to the class ‘similar pairs’ is taken and used to compare to the value resulting from the comparison of the other two images. These approaches are visualised from left to right in Figure 3.1. Chen et al. \([6]\) show a rather large improvement in the performance of the improved metric learning approach over the other two when performing a person re-identification task. They also show that using a quadruplet loss (using four image inputs instead of three) increases the performance even further, but this improvement is only very small. Furthermore, this approach allows for the usage of the triplet network with only two images, using only two of the input channels. Since the output of two of the channels is a similarity representing whether the images from the two channels are similar or not, no third channel is needed for additional comparison during runtime. The output is a value ranging from 0 to 1, so, a threshold has to be set to determine which similarities are to be expected from cabinets that are the same and which are not. This threshold can either be set to a static value, or it can be determined at runtime by an external knowledge reasoner which takes into account external factors which might influence the output values.
3.1.1. Input Triplets

Since each triplet is a combination of three images, a lot of different combinations can be made. Some of these combinations feature images that are more similar to each other than other combinations. Combinations in which the images are very similar yet belong to a different class (or in this case; cabinet type) are more difficult to separate, thus belong to a set of ‘hard triplets’. Training the network on these hard triplets will cause the network to improve faster. This is due to the fact that easy triplets naturally have a larger distance between them, and the loss function used in verification learning penalises small distances between inputs that are of a different class. The triplets that naturally have a larger distance produce gradients that are close to zero, and thus do not improve the network. To ensure fast convergence, hard triplet mining [8] (or ‘smart mining’ [17]) creates triplets that produce larger gradients so that each triplet contributes to the improvement of the network.

3.2. Convolutional Neural Networks

The backbone of any neural network that processes images is their Convolutional Neural Network, which is what the images pass through to obtain meaningful representations in a feature vector.

3.2.1. Convolutional Layers

For the classification of images, Convolutional Neural Networks (CNNs) are the state-of-the-art. Where normal neural networks used weights and biases between nodes to create an output, CNNs use convolutions and pooling, which reduces the number of parameters to update, while still using all the input information as well as spatial information of the pixel values in the image. The input of a convolution can be described as $i(x, y)$, a pixel intensity for each $x, y$ position in the image. The convolution is done by sliding a kernel (or filter) $k(x, y)$ over the image pixel values. While sliding the kernel over the image, the pixel values are multiplied by the values in the kernel, summed to produce a single value, and then passed through a nonlinear activation function. This process can be seen in Figure 3.2. The nonlinear activation function allows for a nonlinear combination of weights and biases, resulting in nonlinear classification boundaries between the classes. Since the convolution kernel calculates one output value from a pixel and its neighbouring pixels, the filtered output image is smaller than the input image. This can be prevented by deploying different padding methods, but this is not usually done. Pooling combines the values of multiple pixels to create a single pixel value using an averaging or maximum operation. This decreases the image size and lowers the resolution of the image that will be convoluted with the next kernel. Pooling aims to increase invariance to lighting conditions, positional changes of an object in the image, robustness to clutter, and lastly, increasing the representation compactness [50].
3.2.2. Fully Connected Layers

While it is possible to achieve good performance on image recognition tasks with CNNs without fully connected layers [39], most architectures do use them. **Fully connected layers** consist of multiple perceptron nodes connected in layers. Each perceptron node takes in values adjusted with weights and adds a bias, according to

$$y_j = \phi \left( \sum_{i=1}^{n} w_{ij} x_i + b_j \right),$$  \hspace{1cm} (3.1)

where $y_j$ is the output of perceptron $j$, $w_{ij}$ is the weight between input $x_i$ and perceptron $j$, $b_j$ is the bias of perceptron $j$, and $\phi$ is the nonlinear activation function. Taking the output of a perceptron node and using that as input for every perceptron node in a next layer creates a fully connected layer, as seen in Figure 3.3. This is also sometimes referred to as a multilayer perceptron network. The weights and biases between all the nodes are the parameters that are updated when the network is trained.

![Diagram of a fully connected layer](image)

Figure 3.3: General structure of Multilayer Perceptron networks. The number of hidden layers and perceptrons per hidden layer are hyperparameters which can be chosen during the design of the network.

3.3. Training and Testing

Neural networks take an input and produce an output during the feedforward state. Getting the desired output given an input is done by updating the parameters of each perceptron (their weights and biases) and each convolutional kernel individually until the desired outcome is produced. Passing inputs to the network and updating its parameters to produce the desired outcomes is called **training**. After training, the network is tested using separate testing data. Since the network is trained to produce specific outputs given specific images (from the training set), it could overfit on the training data, meaning that it will not work well when given data that it has not been trained on. If a network works on test data just as well as on the training data, this means that its **generalisation** is very good.
3.3.1. Training Epochs

Neural networks are trained in epochs. One epoch equals one training cycle where the whole training dataset has been used once fully to train the network. Once the network has been trained on the training set once, it is tested on the test dataset. During testing, the network parameters are not changed. Networks are often trained for tens to hundreds of epochs. To determine how many epochs are needed the performance change over the epochs should be analysed. As soon as the accuracy of the network plateaus it is not useful to train the network any further with that particular dataset.

3.3.2. Model Parameter Initialisation

The parameter values in a network have to be initialised. These initial values form the model from which the network will start training to find the model parameters resulting in the lowest loss. If the model is initialised close to a local minimum the network will be able to optimise more quickly than when the model is initialised far from a local minimum. While random parameter initialisation already achieves quite good results [37], pretraining networks on general image databases like ImageNet has proven to improve training speed and generalisation performance [49]. A pretrained network has already been trained on the classification of images, this means that the network already has convolutional kernels trained to recognise certain (basic or more specific) patterns in images.

3.3.3. Network loss & optimisation

When training a network, the input images are passed through the network, producing an outcome. This outcome is compared to the desired outcome to assess the correctness of the model using a loss function. Once several inputs have been processed (number of inputs before updating the model parameters depends on the batch size hyperparameter), the network parameters are updated and the loss is calculated over a batch of new inputs. This creates a multidimensional field with losses at several positions in a grid (each dimension of the grid is a model parameter of the network). The goal of optimising the network is to find the position in the grid where the loss is lowest. Since there can be millions of parameters in a network, trying every value for every parameter is practically impossible. Instead, optimisation techniques such as Stochastic Gradient Descent (or variations such as ADAM [28], or Adagrad [14]) aim to find the point of minimum loss efficiently. The optimiser takes steps in the direction of the model gradient to achieve a minimal loss value. The size of the steps that are taken is very important. If the steps are too small, it takes a very long time to converge, while too large steps might cause the network to become stuck in a situation where it is constantly oscillating around the local minimum. The step size is also referred to as the learning rate. One way to handle the learning rate step size is to use a learning rate schedule. This starts the training with a certain learning rate and decreases it as training progresses. Since the input data can vary a lot, the gradient calculated for a given input batch also varies. This might result in a gradient descent which is sensitive to the specific inputs given at any time, resulting in oscillations in directions other than the true gradient [36]. To decrease this sensitivity, momentum is added to the gradient descent. Momentum takes the parameter update from the previous step and determines the update for the current step using a linear combination of the previous parameter update and the current gradient. Its effect can be seen in Figure 3.4.
3.3.4. Parameter Update Using Backpropagation

Once the loss has been calculated, this is propagated backwards into the network to update each network model parameter. The direction in which each parameter has to be updated is determined by its gradient (i.e. its derivative). The gradient of each parameter is efficiently calculated using the multivariate chain rule. This allows for reuse of the gradients of parameters later in the network (closer to the output) when calculating the gradient of earlier parameters (closer to the input), since for each node in the network only its derivative with respect to its arguments (incoming values from earlier nodes) has to be calculated. This is visualised in Figure 3.5, where \( \frac{\partial z}{\partial w} \) denotes the derivative of the network loss to that node (through all latter nodes). The backpropagation would continue by deriving \( \frac{\partial z}{\partial w} \) and \( \frac{\partial z}{\partial h} \) to their respective arguments to get the gradient of each parameter, allowing their update into a direction which would lead to a lower loss for the next input batch.

![Efficient backpropagation using the chain rule](image)

3.3.5. Data

To be able to train and test a deep learning model, a lot of data is needed. How much data and what kind of data differs from case to case, depending on the complexity of the problem and the level of performance that is required. For some applications neural networks are trained using millions of images [11].

Collecting and labelling good quality data is expensive. Imagine standing in a supermarket with the task to take photographs of cabinets, some of which are fully stocked, and others which are not. Now realise that you need to take thousands of these images. This is why data is so limited.

Since deep neural networks approximate a function that fits the given data, it is important to test them using data that they have not been trained on. This is to avoid overfitting. Overfitting will cause a network to have a very good performance on the training data, but have a very bad generalisation to unseen data. So the network is trained in a way that it will only be able to classify images it has seen before. But since data is so limited, it is often split unequally, to allow the network to use much more data to train on than to test on. To ensure that the smaller test set doesn’t by chance contain only datapoints that are easy to classify (and thus mistakenly rewarding the network with a high performance) k-fold crossvalidation is applied. K-fold crossvalidation splits the whole dataset in K smaller sets. The network is then trained on the first K-1 of these sets, and tested on the one set that is left. Then, the network is again trained but now using a different test to test. This is repeated until each small dataset is used for testing once. This is visualised in Figure 3.6. The performances over all the folds are averaged and the standard deviation is reported to show stability of the network training and it’s generalisation.

![Visualised example of 3-fold crossvalidation](image)
While large datasets such as ImageNet [11] exist and are open for anyone to use, not every possible application of deep learning has such a dataset available. In the absence of data, and when there are not enough resources to create data from the real world, simulation is the alternative. Within a simulation it is possible to simulate the real world and create virtually as much data as needed. The only restriction being the available hardware and time available to render the data. These restrictions, however, are far less restrictive than the ones for the generation of real data.

Training networks on simulated datasets can allow the networks to be trained to such an extent that they generalise well to unseen, generated data. However, these models often do not generalise well to real data. This is called the reality gap. Simulations generally are not 100% the same in every aspect as the real world. So, the network might be trained to detect certain features that are found in simulated data but that are not present in real data. Or, the network might ignore important features of the real world since they were not present in the simulation. There is a lot of research going on to bridge the reality gap with approaches such as hybrid datasets that augment real datasets with simulated datasets [27], or domain randomisation that tries to include as much randomness into the simulation so that the real world could be considered to be simply one of those randomised simulations [45].

3.4. Evaluation
There are different methods to compare the performance of deep neural networks with binary classification. Binary classification can have four different outcomes:

- **True Positive (TP):** Input belonging to class 1 is classified as class 1
- **True Negative (TN):** Input belonging to class 0 is classified as class 0
- **False Positive (FP):** Input belonging to class 0 is classified as class 1
- **False Negative (FN):** Input belonging to class 1 is classified as class 0

From these outcomes, TP and TN are desired to be high, and FN and FP are desired to be low. These can also be used in a confusion matrix:

<table>
<thead>
<tr>
<th>Target</th>
<th>Prediction</th>
<th>Class 1</th>
<th>Class 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 1</td>
<td>True Positive</td>
<td>False Negative</td>
<td></td>
</tr>
<tr>
<td>Class 0</td>
<td>False Positive</td>
<td>True Negative</td>
<td></td>
</tr>
</tbody>
</table>

Using these outcomes, different metrics can be used to compare different solutions. The most intuitive metric is **accuracy**, according to

\[
\text{Accuracy} = \frac{TN + TP}{TN + TP + FN + FP}.
\]  

(3.2)

Having an accuracy of 1 means that all inputs are correctly classified, while an accuracy of 0 means that all inputs are misclassified. This metric is very easy to understand, but it also can be very deceiving. For example, the number of inputs belonging to each class might not be balanced. E.g. if there are 1000 samples of class 1 and 10 of class 0, then classifying every input as class 1 yields an accuracy of \( \frac{0+1000}{0+1000+0+10} = \frac{1000}{1010} = 99\% \), while the network is in effect doing nothing useful. Furthermore, the accuracy metric assumes that each outcome has the same weight. That is, misclassifying class 1 carries the same consequences as misclassifying class 0, which is not always the case. Another popular metric is the **F1-score**. The F1 score is calculated using **precision** and **recall**, according to:

\[
F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}},
\]  

(3.3)

where

\[
\text{precision} = \frac{TP}{TP + FP},
\]  

(3.4)

\[
\text{recall} = \frac{TP}{TP + FN}.
\]  

(3.5)
The F1 score is more robust against class imbalance, but it changes depending on which class is deemed positive and which class is deemed negative. Hand and Christen [16] have also criticised this method and claim that alternative measures should be used.

Recently, Chicco and Jurman [7] have shown using different use cases how both accuracy and the F1 score can be misleading. Instead, they argue that the Matthews correlation coefficient (MCC) gives a much more honest representation of the performance of a model, regardless of class balance. The MCC is defined as:

\[
MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}
\]

The MCC can range between -1 and 1, where 0 means that the solution is as good as random guessing, 1 is a perfect solution, and -1 is the opposite of a perfect solution. Using the example with 1010 classes, the network model would have achieved an MCC of 0, which is a much better indication of its performance. This method, however, still regards FN and FP as equally bad.

Since the datasets in this research are balanced and the costs for FP and FN classifications are equal, accuracy can be a valid performance metric. However, to make comparisons to datasets which might be imbalanced in the future easier, the MCC is included in some of the results as well.
Network Design & Data Generation

Using the background knowledge on convolutional neural networks (CNN) and verification networks (VN) from the previous chapter, this chapter will discuss the design choices of the CNN and VN that will be used to detect whether products are missing or misplaced in a retail store cabinet. This chapter will also discuss the need for data generation and the chosen method to do so. Regarding the generated data, this chapter will also discuss the necessary preprocessing of the images and the creation of triplet images that are needed to train the VN. The next chapter will compare the methods discussed in this chapter with experiments designed to answer the research questions stated in Chapter 1.

4.1. Data Generation & Preprocessing

As discussed in Section 3.3.5, a lot of images are needed to be able to train both methods to detect cabinets that have missing or misplaced products. Since this dataset is not available, one has to be made. This is done by simulating retail store cabinets using crude product models, since gathering real data is not feasible for this thesis. One thing that makes this domain very difficult and confusing has to do with the seen and unseen data points. The networks have to be trained on certain data points that are from there on ‘seen’, and tested on data points that it was not trained on, and are therefore ‘unseen’. The problem in this domain is that there are two levels of seen and unseen data. The first level - which is what most deep learning approaches deal with - is that cabinets can have different configurations. Products can be on different shelves, there can be different numbers of shelves per cabinet, different numbers of products on the whole cabinet and so forth. During this research, networks will always be tested on unseen cabinet configurations. However, there is a deeper level of seen and unseen data, namely the products themselves. As there are up to 30,000 products at one Dutch retail store alone, it is not feasible to create product models for all these products and use these to generate the cabinets. So instead, a set of products is created. Now, since the trained models have to perform well in real supermarkets that do feature those 30,000 products, the generalisability towards unseen products has to be tested as well.

Training a Neural Network to distinguish ‘full’ cabinets from ‘unfull’ or ‘misplaced’ cabinets requires data that covers a lot of different cabinet configurations. Going to a retail store and physically take photos of cabinets is not feasible due to the needed size of the dataset. Instead, a random cabinet generator has been made that takes 3D product models and uses these to fill a cabinet. This has been done using Blender, which is open-source 3D rendering software running on Python [9]. This section will briefly discuss the creation of the product models as well as the method of creating cabinets with random configurations.

4.1.1. Product Models

3D models of the supermarket products have been created by making a simple cube and sticking a front-view photo of a product downloaded from AH.nl on it. Since the products are mostly seen from the front-view, product sides other than the front are not that important. The product set consists of 204 products varying from ground coffee beans to milk. The product set has been split into a train and test set, ensuring that cabinets used to test the system feature only products that the network has not
been trained on. This will illustrate the ability of the algorithm to extend to products that it has not been trained on. While new products would still require referent images, it is still much easier than taking multiple photos of each individual product, possibly from different angles with different light conditions, to do object recognition itself. A few examples of product models can be seen in Figure 4.1.

4.1.2. Cabinet Generation

This subsection will discuss the way cabinets are generated. First, the generator starts with an empty cabinet, featuring a first shelf that is the base of the cabinet (placed on the ground), as well as the back of the cabinet. Then, a random product is taken from the product set that is used to generate the cabinets. The height of this randomly placed product determines the height between the previous shelf (the base if no shelves are placed yet) and the next shelf. Four centimetres are added to this height to ensure that the randomly picked product fits the shelf nicely. This process is repeated until the space that is left between that last added shelf and the top of the cabinet is smaller than the height of the tallest product plus those 4 centimetres. When this rest height is smaller than the height of the tallest product plus those 4 centimetres, no more shelves are added. This ensures that there are still products that fit on the top shelf without extending beyond the height of the cabinet. Note that at this point no actual products are added to the cabinet yet.

Now, the generator iterates over every shelf in the cabinet to fill the shelves. For each shelf, a random product is picked from the product set until a product is picked that fits the shelf in terms of height and width (determined by the space that is still left on the shelf). This product is then added to the shelf and a new random product is chosen to be placed next to the previous product. To increase randomness in the cabinets, this new product cannot be the same as the previous product. This then continues until the rest space on the cabinet is smaller than the width of the smallest product in the product set combined with a minimum space that has to be between every product. The leftover space is then divided between every product to ensure even spacing between all the products on each shelf.

After doing this for every shelf a simulated photograph is made by rendering the cabinet model using Blender’s Eevee renderer using a camera that fully captures the cabinet.

For every cabinet that is generated, the generator places shelves at appropriate heights based on random products in the product set. This results in cabinets with varying shelf heights and number of shelves. Each product has a horizontal gap (between products) and a vertical gap (between the product and the shelf above it). The generator randomly picks products until it has found a product that fits the shelf height without leaving a gap between the products that is larger than 2 cm or smaller than 1 cm. The method to place shelves is aimed to allow every product to be used in a cabinet approximately as many times as every other product to ensure product variability in the data, as well as to represent the compactness of products in real retail store cabinets.

Figure 4.1: A couple of examples of product models as used in Blender to generate cabinets. Note that the front face of the product is correct while the sides are not.
4.1. Data Generation & Preprocessing

<table>
<thead>
<tr>
<th>Cabinet State</th>
<th>Fully Stocked</th>
<th>Correctly Stocked</th>
<th>Product Location Deviations¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referent</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Unfull</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Misplaced</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Different cabinet states and their configurations as generated using the cabinet generator.

As can be seen in Table 4.1 there are four different types of cabinet configurations to create a render of: the referent, featuring a full cabinet with products that are neatly placed; a full cabinet from the real world, with slight product deviations (rotations and x, y shifts); an unfull cabinet from the real world, with slight product deviations and one or more missing products, and a ‘misplaced’ cabinet from the real world, with slight product deviations and one or more misplaced products. All types are generated for every cabinet that is created, and multiple renders are taken with different product deviations and lighting conditions (lighting direction and colour temperature). Figures 4.2a and 4.2b show a small part of the same cabinet with and without the product deviations. **Note that there is only one row of products. If a product is missing then there is no product behind the missing product that is revealed.** It is chosen to create the unfull cabinets this way since products only need to be marked as ‘missing’ when there are no more products left.

The number of products to remove or misplace in a cabinet is determined using an exponential probability distribution as seen in Figure 4.3. Whenever this function outputs a 0, 1 product will be removed, essentially adding the probability of removing/misplacing 0 products to removing/misplacing 1 product, and setting the probability of 0 products to 0.0. The choice of using an exponential probability distribution is based on the idea that neural networks learn faster on data that is more difficult, as well as the assumption that cabinets wherein few products are missing or misplaced are more difficult for the networks to classify or verify. However, training the network solely with cabinets that have only a few products missing or misplaced might result in bad generalisation to cabinets where more products are missing, so it is preferred to include at least a few of these cabinets to the data.

¹Products deviations refer to a small change in x, y, and yaw of the product. See also Figures 4.2a and 4.2b.
4.1.3. Dataset Naming

Since most experiments performed during this research require custom datasets designed to investigate particular characteristics of either the datasets or the networks trained with them, a naming convention has been set. This naming convention is explained in Figure 4.5. The first three letters of the name denote the type of cabinet to be detected (either MSS for ‘missing’ or MSP for ‘misplaced’). The next three letters show the usage of the data (either TRN for training or TST for testing). Lastly, the datasets used for testing include a number at the end varying from 0 to 100. This denotes the fraction of unseen products used to generate the cabinets used to create the dataset images. This is not present for the training datasets as the products in that dataset are all seen by definition. While datasets with the same name are the exact same datasets within an experiment (i.e. the dataset is generated once and used multiple times), they can be generated again with the same characteristics if they are used in a different experiment.

The misplaced product is on the sixth shelf from the top, second product from the right.
4.2. Network Design

This section will discuss the choice of CNN that will be used to perform cabinet classification among the classes ‘full’, ‘unfull’ and ‘misplaced’, as well as the verification network that was designed to achieve the goal as stated in Chapter 1. The CNN is a method of classifying a cabinet as ‘correct’ or ‘incorrect’ based on a single image, while the verification network compares an image of a cabinet with an unknown state to an image of a cabinet from the referent. The CNN is an integral part of the verification network as it is used to extract feature vectors from the input images, which are then used for comparison, as explained in Chapter 3.

4.2.1. Single Image Input - CNN

To classify the cabinet images using a single image using deep learning, convolutional neural networks (CNN) are state-of-the-art. CNN’s use convolutional layers to extract features from images which can then be classified using non-linear classification boundaries created by the fully connected layers. A CNN designed for image classification that transfers well to the retail store cabinet domain has to be found. Two very promising architectures are ResNet [19] and DenseNet [23]. These architectures propagate the outputs of earlier layers to later layers to promote gradient flow (networks that do not propagate these outputs often suffer from a vanishing or exploding optimisation gradient as more layers are added [15, 23]). Whereas ResNet sums the outputs of one layer with the outputs of the single next layer, DenseNet concatenates the output of each layer with all the next layers. This results in an architecture in which each layer receives a collection of all the information of all the preceding layers (in other words, in a network with $L$ layers, the $i^{th}$ layer has $i$ inputs, resulting in a total of $\frac{L(L+1)}{2}$ connections instead of just $L$ in traditional networks), which has shown to perform better than ResNet, while having fewer parameters [23]. However, due to backpropagation needing the gradient to every layer output, this becomes a very memory-hungry algorithm, and thus is less speed-efficient. Due to limited time and hardware resources for network training, the ResNet architecture was tested first on usability. Testing the ResNet18 CNN on cabinets with missing products yielded very good results, which indicates that it might be a very suitable architecture for the cabinet verification task.
Another advantage is that it is possible to interchange the ResNet18 network with their deeper variants, ResNet34, ResNet50, or even deeper if necessary, without altering the rest of the code. These deeper networks might achieve higher performance at the cost of runtime speed. The ResNet18 network as imported using PyTorch features an image input of 224x224 RGB images, which are processed through 18 layers, and has an output vector of size [1000x1] to be able to classify 1000 classes as pretrained on ImageNet. To classify between correct (full or referent) and incorrect (unfull or misplaced) cabinets, the fully connected layer transferring the [512x1] feature vector to the [1000x1] class vector is replaced by one that transforms the feature vector to [2x1] to allow for binary classification. This last layer is passed through a softmax layer to normalize the output. The input is classified as the class corresponding to the index of the highest value in the output vector. The network can be seen in Figure 4.6. The pipeline can be notated mathematically as:

\[ \begin{bmatrix} y \\ z \end{bmatrix} = f(x), \]  

(4.1)

where \( y \in [0,1] \) and \( z \in [0,1] \) are the normalised output values \((y + z = 1)\) resulting from input image \( x \) belonging to class 0 or 1, respectively.

To train the single image input method a cross entropy loss is used according to:

\[ L(y, \hat{y}) = -\sum_{i=1}^{n} y_i \cdot \log(\hat{y}_i), \]  

(4.2)

where \( y_i \) is the class label of input \( i \), and \( \hat{y}_i \) is the network output scalar value for input \( i \). The cross-entropy loss is useful in multidimensional classification tasks such as this, as its value increases together with the disparity between the prediction of the correct class and the actual class label belonging to the input. This loss penalises or rewards the output value of the correct class only. The distribution of the output values along the other classes does not affect the loss. Furthermore, this loss function has been shown to be effective in the field of facial verification using a similar network [41].

The rest of this report will refer to the Single Image Input method simply as 'CNN'.

4.2.2. Multiple Image Input - Verification Network

To compare two input images, verification networks have been chosen due to their success in face verification tasks. These tasks have a set of facial images and compare a new image to all stored images to find the closest match to verify the identity of the person whose face is in the image. Transferring this method to the cabinet verification domain requires the network to have a specific input to compare the unknown image to, instead of comparing it to a number of different comparison images. Furthermore, instead of determining which comparison image is closest to the unknown input image, the verification network has to produce a similarity score quantifying the similarity between the two images that have been compared.

The verification network uses part of the CNN to extract feature vectors which are then used for comparison between different inputs. Instead of passing the [512x1] feature vector to a fully connected layer which maps the features to the classes, the feature vectors extracted from different images are subtracted from each other. This ‘difference’ vector is then passed to a fully connected layer which maps this new [512x1] vector to a [2x1] vector as described in Section 3.1. This transformation from the feature vector to the output vector can be any combination of FC layer(s). The choice has been made to have a single FC layer as this is most similar to the original ResNet18 architecture, which
both shows that one FC layer is sufficient and makes that the comparison between the CNN and the verification network is focused on the addition of prior knowledge, instead of the changes in architecture design. Furthermore, the main source of performance comes from the convolutional layers, as these are the layers that extract features. The FC layers simply pick and combine the features that make for easy classification. Lastly, since most of the model parameters in a network come from the FC layers, adding more of these will result in slower training and lower memory-efficiency, while not necessarily increasing the performance. A different configuration has been tested as well, where the feature vector is first transformed to a \([1024\times1]\) vector before transforming it to the final output vector. While there might be configurations that can achieve better performance, adding more fully connected layers will only be helpful if this added complexity is needed to actually classify the features extracted by the convolutional layers, otherwise the model will start to overfit. Adding the additional layer yielded very poor results in this case, validating the choice for a single FC layer transforming the feature vector to the output vector. From the final output vector one of the two values is used as a measure of similarity. This pipeline can be notated mathematically with:

\[
    z = f(x, y),
\]

where \(z\) is the similarity value between 0 and 1 resulting from the comparison of images \(x\) and \(y\). The desired outputs of the system are:

\[
    z = f(x, y) = \begin{cases} 
    0, & \text{if } s_u = s_{\text{ref}} \\
    1, & \text{if } s_u \neq s_{\text{ref}}
    \end{cases},
\]

where \(s_u\) is the state of the cabinet that has an unknown state, and \(s_{\text{ref}}\) is the referent state of the cabinet, resulting from input images \(x\) and \(y\).

The network as used during training can be seen in Figure 4.7, where and anchor image (in this case the referent image) is compared to an image of the same class (full cabinets) and an image of a different class (either unfull or 'misplaced' depending on the experiment). During runtime a cabinet with unknown state is compared to the referent cabinet. Since only two images are used, the network is used with only two inputs, as seen in Figure 4.8. The fully connected (FC) layer transforming the \([512\times1]\) feature vector to a \([2\times1]\) output vector is chosen such that it closely relates to the original ResNet18 architecture, as the last FC layer in ResNet18 transforms the \([512\times1]\) feature vector to a vector with length equal to the number of classes. It is possible to add more FC layers, for example the feature vector could be transformed to a \([1024\times1]\) vector before transforming that to a \([2\times1]\) output vector, but preliminary experiments showed no advantage of adding extra FC layers over staying close to the original ResNet18 architecture.

![Figure 4.7: Verification network used to train the CNN and fully connected layers used to get a similarity score between images. The anchor image corresponds to the referent image, the positive image is the image of a correctly filled cabinet, and the negative image is an image of an unfull cabinet or a cabinet with misplaced products.](image-url)
Before training the verification network, the CNN layers that are used in the verification network are loaded with the model parameters that those layers converged to when training the whole CNN network separately (for cabinet analysis using a single image). This is to give the network better initialisation parameters which improves the training speed and generalisation of the network [49]. The network is then retrained using a triplet network, as visualised in Figure 4.7. The triplet loss can simultaneously lower the intra-class variability and increase the inter-class variability. Since the distances between the referent and full cabinet \(d(\text{ref}, p)\) and the distance between the referent and the unfull or ‘misplaced’ cabinet \(d(\text{ref}, n)\) range from 0 to 1, the loss function is defined as

\[
L = d(\text{ref}, p) + (1 - d(\text{ref}, n)),
\]

(4.5)

where \(d(\text{ref}, p)\) is the distance between images within a class, and \(d(\text{ref}, n)\) is the distance between images that do not belong to the same class. This is slightly different from more traditional triplet loss functions where a margin is introduced, since those applications do not limit the distances to range 0 to 1. In this case, the margin is 1.

4.3. Hardware & Software

The network is built using Python 3.8 and the PyTorch 1.4.0 library [33] for Deep Learning. Network training and testing is done on an Alienware Aurora i9 running on Ubuntu 18.04. This PC features an Intel Core i9-9900K @ 4.7 GHz on 8 cores, two NVidia RTX 2080 Super GPUs which are used with CUDA 10.2, and 32 GB of DDR4 RAM.

4.4. Summary

This chapter has discussed the choice to use the ResNet18 architecture for both methods. While the single image input method (CNN) uses the ResNet18 architecture by only changing the output vector length to accommodate for two different classes (‘full’ and ‘unfull’ or ‘full’ and ‘misplaced’), the verification network passes two images through the CNN separately, takes their extracted feature vectors to then subtract those and pass them to a last fully connected layer to result in a similarity value between 0 and 1. The CNN is trained with a cross-entropy loss, while the VN is trained using a version of a triplet loss.

This chapter also discussed how data is generated by creating simple 3D models of retail store products and using those to randomly generate cabinets. These cabinets are then used to create images that will be used to train and test both methods.
Experiments & Results

Following the research questions laid out in Chapter 1, this chapter will introduce for each question an experiment that is designed to investigate the issues they raise. The information resulting from these experiments will be discussed regarding the respective research question as well as some general points. The first five sections in this chapter each correspond to a question, which will be repeated and for which an appropriate experiment is laid out subsequently. Section 5.6 will discuss some more experiments designed to gain insight in how the methods respond to changes in dataset size, chosen CNN network architecture, and the level of noise introduced in the simulation. Note that for the first two experiments both accuracy and the MCC are used to measure the performance. Since the results for these experiments show a high correlation between the MCC and the accuracy, the subsequent experiments regard only the accuracy, as this is a more intuitive measure of performance.

The methods have been used with the settings shown in Table 5.1.

### Table 5.1: Hyperparameter settings used to get the results discussed in this chapter

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>CNN</th>
<th>Verification Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Rate</td>
<td>0.001</td>
<td>0.01</td>
</tr>
<tr>
<td>Momentum</td>
<td>0.9</td>
<td>0.5</td>
</tr>
<tr>
<td>Decay rate</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Decay step</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Batch size</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

#### 5.1. Q1: Missing Products

This experiment will answer the research question “Can the absence of products be detected more reliably and quickly by comparing current cabinet state images with a referent image than with classification using a single image?”

To analyse the performance of the verification network as compared to the CNN, two datasets ‘MSSTRN’ and ‘MSSTST0’ are created consisting of three different types of cabinets; full cabinets without product deviations (referent image), full cabinets with product deviations, and unfull cabinets with product deviations. The cabinets are randomly generated using a set of 203 retail store product models. The CNN is tested on only two of the cabinet types; ‘full’ and ‘unfull’. The referent images are not used to train the CNN since this is considered knowledge stored in the knowledge base which the method with the single image input does not use.

Each image dataset is created from 200 cabinets, featuring 5 referent images, 10 ‘unfull’ images, and 10 ‘full’ images, making for a total of 5000 images for training and 5000 images for testing. The CNN only uses 4000 images for training and 400 for testing, since it does not use the referent images. To increase the reliability of the results, both models are trained 5 times on MSSTRN and tested 5 times on MSSTST0, and vice-versa, to create two-fold cross-validation. For each fold, the trained model parameters of the CNN with the best performance are saved and used in the corresponding CNN layers of the VN as weight initialisation.
The data generation process can be seen in Figure 5.1, and the training and testing process for a single fold can be seen in Figure 5.2.

![MSSTRN DATASET GENERATION](image)

**Figure 5.1:** Dataset generation process for a dataset featuring missing products. This flowchart shows the process for the generation of images belonging to a single cabinet. The different colours visualise the products used to generate the cabinet. There is a test product set (blue), a train product set (red) and combined product sets (purple).

![VN CNN CNN (pretrained on ImageNet) CNN (trained on MSPTRN) 1. Verification Network (with trained CNN model param.) 2. Performance](image)

**Figure 5.2:** Training and testing procedure for both methods on datasets with missing products. This image shows only the first fold. The purple colour visualises that the data is created using all the product models.

The next subsections will show the performance of both the CNN and the VN on missing products using the MSSTRN and MSSTST0 datasets. The first subsection will show the results on the CNN, followed by a subsection on the VN. Finally, both methods are compared.

### 5.1.1. CNN

The classification of cabinets with missing products with a single CNN works very well. The CNN achieved an average accuracy of $99.8 \pm 0.071\%$, and an MCC of $0.997 \pm 0.0014$ on the test sets. The model that achieved the highest performance in terms of both accuracy and MCC achieved an accuracy of $99.9\%$ and an MCC of $0.998$. In Figure 5.3 the learning curve of the model that achieved the best performance can be seen. It is observed that the model reaches its peak performance at the second epoch, after which it plateaus. From the confusion matrix in Table 5.2, it can be seen that three-quarters of the misclassified cabinets were cabinets that are actually missing a product but that were classified as full cabinets. Having more unfull cabinets classified as full as opposed to the other way around is a trend which is observed to occur at almost all epochs for every trained model.
5.1. Q1: Missing Products

CNN training with 10 epochs (train size:4000, test size:4000)

Figure 5.3: Learning curves when training a CNN for classification between full and unfull cabinets. The train and test sizes refer to the number of generated images (200 cabinets x (10 full images + 10 unfull images) = 4000), as shown in Figure 5.1. From Epoch 2 onward there is no significant improvement.

Table 5.2: Confusion matrix showing the targets and predictions for cabinets with missing product using a CNN.

<table>
<thead>
<tr>
<th>Target</th>
<th>Prediction</th>
<th>Full</th>
<th>Unfull</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>1999</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Unfull</td>
<td>3</td>
<td>1997</td>
<td></td>
</tr>
</tbody>
</table>

5.1.2. Verification Network

The classification of missing products using the VN also works very well. Training the VN a total of 10 times over two folds (using the model parameters of the CNN with the best performance on each fold as weight initialisation) yielded an average accuracy of $97.3 \pm 2.6\%$ and an MCC of $0.949 \pm 0.0486$. The model achieving the highest performance in terms of both accuracy and MCC achieved an accuracy of $99.7\%$ and an MCC of $0.995$. In Figure 5.4 the learning curve can be seen for the trained model that performed best on the test set. Figure 5.5 shows how the VN improves over the epochs by moving the distances between referent-full pairs to 0 and those for referent-unfull pairs to 1. From the confusion matrix in table 5.3 it can be seen that most of the mistakes were full cabinets that were classified as unfull. Another observation is that the verification network is sensitive to the initial model parameters, obtained by the trained CNN for each fold. The VNs that were initialised with the model parameters from the first fold achieved a significantly lower performance than the verification networks initialised with the model parameters from the second fold. The best CNN for the first fold achieved an accuracy of $99.88\%$, and the verification networks trained with those model parameters as initialisation achieved an average accuracy of $95.28 \pm 2.1\%$. Meanwhile, the best CNN for the second fold achieved an accuracy of $99.9\%$, and the VNs trained using those initial model parameters achieved an average accuracy of $99.41 \pm 0.39\%$. So, a good model parameter initialisation from training the CNN helps the performance of the VN by a large margin.
5. Experiments & Results

Verification training with 10 epochs and 50 triplets per cabinet

![Learning curves](image)

Figure 5.4: Learning curves when training a verification network for classification between full and unfull cabinets featuring seen products in unseen cabinet configurations.

Table 5.3: Confusion matrix showing the targets and predictions for cabinets with missing products using a verification network.

<table>
<thead>
<tr>
<th>Target</th>
<th>Full</th>
<th>Unfull</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>9958</td>
<td>42</td>
</tr>
<tr>
<td>Unfull</td>
<td>9</td>
<td>9991</td>
</tr>
</tbody>
</table>

5.1.3. Comparison

In Figure 5.6 the performance of both methods are plotted. From this figure, it can be seen that the CNN performs slightly better than the VN. The CNN median has an accuracy which is on average 0.4 percentage points higher than the verification network when considering only the VNs trained with the model parameters corresponding to the overall highest performing trained CNN as weight initialisation.
5.2. Q2: Misplaced Products

This experiment will answer the subquestion "Can the misplacement of products be detected more reliably and quickly by comparing current cabinet state images with a referent image than with classification using a single image?".

Analysing the performance of the VN on misplaced products is done in the same way as for the previous experiment. However, instead of images of cabinets where products are missing, images are generated where products might be misplaced. The same methods will be compared as in the previous experiment.

Two datasets are generated; 'MSPTRN' and 'MSPTST0'. Both datasets contain images created from 200 randomly generated cabinets featuring 203 product models. From each cabinet 25 images are generated; 5 referent images in which the cabinet is fully and correctly stocked, 10 'full' images in which the cabinet is fully stocked with products that have small positional deviations, and 10 'misplaced' images in which products have small deviations and a number of products might be replaced with a different product. The referent images are not used to train the CNN since this is considered knowledge stored in the knowledge base which the method with the single image input does not use.

As with the previous experiment, the train and test sets are also switched to accommodate two-fold cross-validation. The data generation and the experiment process can be seen in Figures 5.7 and 5.8, respectively.

![Figure 5.6: CNN vs Verification network in terms of Accuracy and MCC on seen products in unseen cabinet configurations.](image)

![Figure 5.7: Dataset generation process for a dataset featuring missing products. This flowchart show the process for the generation of images belonging to a single cabinet. The different colours visualise the products used to generate the cabinet. There is a test product set (blue), a train product set (red) and combined product sets (purple).](image)
5. Experiments & Results

The following subsections will show the performance of both the CNN and the VN on misplaced products using simulated data. The first subsection will show the results on the CNN network, followed by a subsection on the VN. Finally, both methods are compared.

### 5.2.1. CNN

The single CNN achieves an average accuracy of $73.14 \pm 0.61\%$ and an average MCC of $0.523 \pm 0.012$. The model with the best performance reached an accuracy of $74.25\%$ and an MCC of $0.550$. Its corresponding learning curve can be seen in Figure 5.9 and its confusion matrix is shown in Table 5.4.

![Learning curve for the CNN training on misplaced cabinet data featuring seen products in unseen cabinet configurations.](image)

---

**Figure 5.8:** Training and testing procedure for both methods on datasets with missing process. During the second fold of cross-validation, MSPTRN and MSPTST0 are switched in this image. The purple colour visualises that the data is created using all the product models.

---

**Figure 5.9:** Learning curve for the CNN training on misplaced cabinet data featuring seen products in unseen cabinet configurations.
5.2. Q2: Misplaced Products

Table 5.4: Confusion matrix showing the targets and predictions for cabinets with misplaced products using a CNN.

<table>
<thead>
<tr>
<th>Target</th>
<th>Prediction</th>
<th>Full</th>
<th>Misplaced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>1910</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>Misplaced</td>
<td>940</td>
<td>1060</td>
<td></td>
</tr>
</tbody>
</table>

5.2.2. Verification Network

The VN achieves an average accuracy of $82.54 \pm 4.32\%$ and an average MCC of $0.670 \pm 0.076$. On average, this is an increase of almost 8.5% in accuracy and 0.15 in MCC over the CNN. The highest achieved accuracy is $89.63\%$ which coincides with the highest MCC of 0.800. It is observed that for misplaced products, there is not a large difference between the two folds as was seen for missing products in the previous experiment. The learning curves for the model achieving the best performance can be seen in Figure 5.10. Figure 5.11 shows how the network performs over the epochs in terms of distances between the referent and the full cabinets and the referent and misplaced cabinets. The corresponding confusion matrix is seen in Table 5.5. While it might look as if the network is still improving at 10 epochs, training the network 10 times for 20 epochs did not improve the performance over training for 10 epochs.

Figure 5.10: Learning curve for the verification network training on misplaced product data featuring seen products in unseen cabinet configurations.
Distances between referent-full (green) and referent-misplaced (red) at different epochs.

Figure 5.11: Visualisation of how the VN improves during training. The distances between referent-full (green bars) should be 0 while the distances between referent-misplaced (red bars) should be 1. These images show how the network moves these distances to where they should be during training.

Table 5.5: Confusion matrix showing the targets and predictions for cabinets with misplaced products using a verification network.

<table>
<thead>
<tr>
<th>Target</th>
<th>Prediction</th>
<th>Full</th>
<th>Misplaced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>8291</td>
<td>1709</td>
<td></td>
</tr>
<tr>
<td>Misplaced</td>
<td>366</td>
<td>9634</td>
<td></td>
</tr>
</tbody>
</table>

5.2.3. Comparison

Comparing both methods on the detection of cabinets with misplaced products, it can be seen that the VN performs significantly better than the CNN. In Figure 5.12 the performances for both methods can be seen. The VN has an average accuracy that is almost 8.5% higher than the CNN average accuracy.

Figure 5.12: MCC scores and accuracies of the CNN and verification network when trained and tested 10 times on both of the folds featuring seen products in unseen cabinet configurations.
5.3. Q3: Generalisability

This experiment will answer the subquestion: “How generalisable is this approach to new products as opposed to classification without prior knowledge?”.

Generalisability in this case is defined as the capability of a network to be deployed in environments where there are products the network has not seen during training. This is tested by using separate sets of products for the generation of cabinet images used in the training and testing datasets.

To test the generalisability, both methods are trained on a dataset generated using only 100 product models, and tested on a number of different datasets generated using a different fraction of seen and unseen products. The two training datasets, ‘MSSTRN’ and ‘MSPTRN’, feature 100 products in 200 different cabinet configurations, used to generate a total of 5000 images for each dataset. These datasets are used to train both methods for missing and misplaced products, respectively and separately. Both methods are trained five times and each time the model parameters are stored. To test generalisability, 6 different test datasets have been created; each with a different number of seen (present in training data) and unseen product models. The generation of these datasets is visualised in Figure 5.13. Each dataset is used to test both methods 5 times. For the VN, the model will the trained once for every model parameter set that is saved from the CNN after training (5 in total). The fraction of seen models in the datasets increases from 0% to 100% in steps of 20%. This experiment is visualised in Figure 5.14. For example, dataset ‘MSSTST40’ features 60 seen products and 40 unseen products to generate 200 cabinets with missing products. Since the cabinets are randomly generated using the available products, the fraction of unseen products that are actually used in the cabinets is not guaranteed to coincide with the fraction of unseen products in the product model set. To account for this, the products used to generate the cabinets are counted, and the actual fraction of unseen products used in the cabinets in the datasets is saved and shown in Table 5.7, along with the dataset names. The occurrences of each product in each dataset can be seen in the respective section in Appendix A, along with the distributions of product height, product width, number of shelves in a cabinet, and number of products in a cabinet.

![Dataset generation](image)

Figure 5.13: Dataset generation to test generalisability of both methods. Displayed are the generation of MSPTRN (0% unseen products), MSPTST100 (100% unseen products) and MSPTST20 (20% unseen products). The same method is used to generate datasets with 40, 60, and 80% unseen products as well as for cabinets with missing products (MSST...). The different colours visualise the products used to generate the cabinet. There is a test product set (blue), a train product set (red) and combined product sets (purple).
Since the accuracy and MCC correlated perfectly during the previous experiments, this experiment will consider only the accuracy to maintain readability of the results.

## 5.3.1. Missing Products

With regards to cabinets with missing products, neither method is expected to have a lot of trouble transferring to unseen products, since the empty spaces that have to be detected still look the same. Looking at Figure 5.15, it is clear that the CNN is not significantly affected by the fraction of unseen products, maintaining a very high performance. Meanwhile, the VN seems to maintain performance as well, but with a very high standard deviation. Looking more closely at the individual trained VN models as shown in Figure 5.16, it can be observed that the high variability is not caused by variability within a trained model but rather between trained models. This shows that weight initialisation is very important to get a good performance from the VN. Once a model is trained well, however, it will perform well regardless of whether the products are seen during training or not. Training the model multiple times using the same initialisation shows that the training of the model is stable as long as the initialisation is good.

Overall, the results show that both methods generalise really well to unseen products for the classification of cabinets with missing products.

Table 5.6: Datasets consisting of 200 cabinets featuring a number of seen products combined with a number of unseen products. Since cabinets are randomly generated, the percentage of unseen products present in each cabinet might differ from the percentage of unseen product models in the product models set. The third column shows the percentage of unseen products present in the generated cabinets. The product distributions can be seen in Appendix A.1 for every dataset.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Seen Products</th>
<th>Unseen Products</th>
<th>Actual Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSSTRN</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MSSTST0</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MSSTST20</td>
<td>80</td>
<td>20</td>
<td>22.8</td>
</tr>
<tr>
<td>MSSTST40</td>
<td>60</td>
<td>40</td>
<td>46.7</td>
</tr>
<tr>
<td>MSSTST60</td>
<td>40</td>
<td>60</td>
<td>63.5</td>
</tr>
<tr>
<td>MSSTST80</td>
<td>20</td>
<td>80</td>
<td>78.3</td>
</tr>
<tr>
<td>MSSTST100</td>
<td>0</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
5.3. Q3: Generalisability

5.3.2. Misplaced Products
Training both methods on MSPTRN and testing them on the MSPTST datasets results in the accuracies as shown in Figure 5.17. From this figure, it can be seen that both methods suffer from products that are not seen during training. A higher fraction of unseen products results in a lower performance. While the performance of the CNN drops down to 57.0%, the verification network still performs rather well at 73.5%. So, while the performance of the CNN drops to an accuracy that is not much better than random guessing, the verification network still performs relatively well.

Another interesting observation is that the CNN when trained on a dataset generated with 100 products and testing it on a test dataset with those same 100 products, performs significantly worse than when it was trained and tested on a dataset created with cabinets featuring 200 products in Experiment 2. On the other hand, the Verification Network performs just as well on both datasets. This suggests that the verification network is less dependent on the number of unique product models used to generate the training dataset than the CNN. This is very important considering that the creation of product models
for training is expensive, so being able to classify cabinets that can feature any of the up to unique 120,000 products using only a fraction of that number for training is very beneficial.

Table 5.7: Datasets consisting of 200 cabinets featuring a number of seen products combined with a number of unseen products. Since cabinets are randomly generated, the percentage of unseen products present in each cabinet might differ from the percentage of unseen product models in the product models set. The third column shows the percentage of unseen products present in the generated cabinets. The product distributions can be seen in Appendix A.2 for every dataset.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Seen Products</th>
<th>Unseen Products</th>
<th>Actual Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSPTRN</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MSPTST0</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MSPTST20</td>
<td>80</td>
<td>20</td>
<td>22.5</td>
</tr>
<tr>
<td>MSPTST40</td>
<td>60</td>
<td>40</td>
<td>47.1</td>
</tr>
<tr>
<td>MSPTST60</td>
<td>40</td>
<td>60</td>
<td>63.1</td>
</tr>
<tr>
<td>MSPTST80</td>
<td>20</td>
<td>80</td>
<td>78.6</td>
</tr>
<tr>
<td>MSPTST100</td>
<td>0</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 5.17: Accuracy of both methods when trained on MSPTRN and tested on datasets with increasing fraction of unseen products in the cabinet generation.

5.4. Q4: Reality Gap

This experiment will answer the subquestion: “How generalisable is this approach to real-world data when it is trained purely on simulated data?”

So far, both methods have only been trained and tested using images from cabinets in simulation. While simulations are becoming more and more advanced, there is still a difference between simulated and real-life data. This disparity is called the reality gap. The reality gap is a difficult one to bridge for deep learning approaches, in different domains from computer vision [46] to evolutionary robotics [29]. The reality gap causes systems that perform well in simulated environments to lose that performance when deployed in a real-life environment. This experiment will investigate to what extent the method proposed in this research suffers from this decrease in performance.

The reality gap is investigated only in low sample size, as it is very time-consuming to manually take images from cabinets in their different states and prepare them to be used by the network. To test the reality gap, 42 photos are taken of real retail store cabinets that are either stocked fully and correctly, or that contain products that are randomly removed or misplaced. The network is trained on the simulated data and tested on the real images. The verification network will be tested by using a real image of a fully stocked cabinet as referent image. Note that the real cabinets feature many products that have not been modelled and used for training.
Also important to note is that the images are taken of three different cabinets. One of these cabinets features only square products as is the case in the simulated data, cabinet 2 features some square products and some non-square products, and cabinet 3 features mostly non-square products such as bags of chips. The three different cabinets can be seen in Figures 5.18a, 5.18b, and 5.18c.

This section will discuss the results first for the cabinets with missing products, followed by the cabinets featuring misplaced products.

![Real Cabinets](image)

(a) Real Cabinet 1  (b) Real Cabinet 2  (c) Real Cabinet 3

Figure 5.18

### 5.4.1. Missing Products

To test both methods on real cabinets with missing products, the models are trained on MSSTRN created using 200 product models. The generation of the real data is visualised in Figure 5.19. The experimental setup is shown in Figure 5.20.

![MSSTST_RL Dataset Generation](image)

Figure 5.19: Dataset generation for the real data. The real data has been created by taking photographs of a real cabinet in full, referent, and unfull states.
Experiments & Results

Verification Network (with trained CNN model param.)

1. MSSTRN
2. MSSTST_RL

Performance

MSSTRN
MSSTST_RL

CNN (pretrained on ImageNet)

CNN (trained on MSPTRN)

Performance

1. CNN
2. VN

Figure 5.20: Experiment setup for the real data. Both methods are trained on simulated data and subsequently tested on real data.

Testing both methods trained on 200 cabinets featuring missing products and testing these trained models on real data yielded an accuracy of 64.2% for the CNN, and an accuracy of 84.5% for the VN. The distances between the image pairs can be seen in Figure 5.21. Looking at the individual image pairs, it is observed that the performance differs between cabinets. The VN achieves an accuracy of 100% on cabinet 1, which, as discussed earlier, features products that are most like the products used in the simulation. On cabinets 2 and 3, the VN achieves accuracies of 82.1% and 71.4%, respectively.

This experiment shows that the verification method transfers significantly better to real data than the CNN.

Figure 5.21: Input pairs distances for real data featuring missing products. The anchor - positive (green) pairs are referent - full pairs, while the anchor - negative (red) pairs are referent - unfull pairs. The green pairs should be as close to 0 as possible, while the red pairs should be as close to 1 as possible.

5.4.2. Misplaced Products

The same method is applied to real-world data featuring cabinets with misplaced products. The generation of the real data is visualised in Figure 5.22. The experimental setup is shown in Figure 5.23.
MSPTST_RL DATASET GENERATION

![Diagram of dataset generation for real data](image)

**REALITY GAP**

**Q4: Reality Gap**

Figure 5.22: Dataset generation for the real data. The real data has been created by taking photographs of a real cabinet in full, referent, and misplaced states.

![Experiment setup for real data](image)

**Figure 5.23: Experiment setup for the real data. Both methods are trained on simulated data and subsequently tested on real data.**

On misplaced data both methods achieve a mean accuracy of about 50%, meaning that neither method transfers well to real data with misplaced products. The distances between the image pairs in the verification network can be seen in Figure 5.24. From this graph it can be seen that not only are the distances all the pairs overlaying, they are also all very large. This might be caused by higher noise levels in the real data. The network has learned to filter out the noises introduced in the simulation, increasing the signal to noise ratio (differences caused by misplaced products vs differences caused by noise). In the real data, this signal to noise ratio is even lower due to the larger differences between the different real-world images, caused by different camera angles, differences between the manually cropped images, etc. This causes the network to be unable to filter out the noise successfully, resulting in all image pairs receiving a high distance score.
Experiments & Results

Figure 5.24: Input pairs distances for real data featuring misplaced products. The anchor - positive (green) pairs are referent - full pairs, while the anchor - negative (red) pairs are referent - misplaced pairs. The green pairs should be as close to 0 as possible, while the red pairs should be as close to 1 as possible.

5.5. Q5: Speed

This section will answer the subquestion: “How does the verification network improve the overall speed with respect to individual object classification?”

To investigate the time needed to analyse a cabinet, images are passed to both the CNN and the verification network and an average of their processing times is measured. It is not possible to directly compare these times to the times needed for individual object recognition, as there is no network directly available that is known to be able to classify 30,000 to 120,000 retail store products with comparable accuracy to the verification network readily available. Instead, the verification network speed is compared to a more complex network, which might be able to classify the individual objects with an accuracy approaching that of the VN (even though this is very difficult to estimate [12]).

Comparing the speed of the CNN classifying the whole cabinet from one image with the speed of the verification network it is expected that the verification network will take approximately twice as long since it passes two images through the whole network instead of just one. Passing multiple classification tasks to both methods yields the speeds as seen in Table 5.8. The VN takes slightly less than twice the time of the CNN since it subtracts two 512 value feature vectors to then pass it to a single fully connected layer instead of two final fully connected layers, which would be the case if two images are passed through the CNN separately. While the VN is slower than the CNN at this stage, its speed can be increased up to the speed of the CNN by storing the referent as extracted feature vectors instead of RGB images, eliminating the need of processing these images at runtime.

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Time [ms]</th>
<th>Standard Deviation [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>4.70</td>
<td>0.818</td>
</tr>
<tr>
<td>VN</td>
<td>8.81</td>
<td>0.560</td>
</tr>
</tbody>
</table>

Table 5.8: Time needed to analyse one cabinet using one image taken from the real world store.

The cabinets could also be analysed using individual object recognition, although this requires some significant assumptions. Both methods require some operations before they can pass the images through the network. Since these steps are similar in both methods, they will not be considered when comparing their speed. Thus, calculating the total time needed to fully analyse a cabinet given any method is simplified to:

\[ T[s] = N \times t[s], \]  

(5.1)
where $T$ is the total time needed in seconds, $N$ is the total number of images needed to be analysed, and $t$ is the time needed to analyse a single image. This equation can be used to compare the verification network method to a method using individual object recognition. Assuming a cabinet with 6 shelves featuring a total of 100 products, the verification network would need to analyse $N = 10^7$ images if all shelves contain missing or misplaced products, and $N = 1$ image if the shelf is fully and correctly stocked. Assuming that a ResNet152 network is sufficiently complex to accurately classify 30,000 images, the individual object recognition method needs almost 7 times more time than ResNet18 to process an input image ($33.0\text{ ms}$ for ResNet152 vs $4.7\text{ ms}$ for ResNet18). So, the individual object recognition method takes approximately 3 times as long as the VN to process one image. Inserting this into equation 5.1, the time needed to process all images needed to analyse a cabinet can be obtained, as seen in Table 5.9. From these results, it can be seen that the VN is estimated to be approximately 3.5 to 350 times faster than the individual object recognition method. Cabinets with more products would increase this ratio even further, cabinets with fewer products would decrease this speedup, but not lower than 3.5 (a cabinet with one product would require one image processing from both methods, wherein the VN is approximately 3.5 times faster than the much larger classification network needed for individual object recognition). Optimising the VN by using feature vectors as referent instead of RGB images means that the VN is approximately 7 to 700 times faster than individual object recognition when considering just the processing of the images through the networks. In practice, the speedup ratio will be lower due to other operations that both methods require and will take approximately the same time for both methods. For example, scaling the images to 224x224 as required by the network takes about 8 milliseconds on the hardware stated in Section 4.3.

Using the system on less powerful hardware will slow down the processing by the networks significantly. The preprocessing, however, won’t be impacted as much since that is primarily performed on the CPU while the networks are processed by two powerful GPU’s. So, while the numbers shown here would fit the requirements well, it is difficult to say whether the requirements will be met on slower hardware that is more commonly found on robots.

<table>
<thead>
<tr>
<th>Method</th>
<th>Best Case Scenario</th>
<th>Worst Case Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual Object Recognition</td>
<td>$700 \times t$</td>
<td>$700 \times t$</td>
</tr>
<tr>
<td>Verification Network</td>
<td>$t$</td>
<td>$10^7 \times t$</td>
</tr>
</tbody>
</table>

Table 5.9: Times needed to fully analyse a cabinet for the best case scenario (fully correctly stocked cabinet) and the worst case scenario (at least one missing or misplaced product on every shelf). $t$ is a variable depending on the system hardware that runs the network. The times considered are the times needed to pass an image through the network.

Using $t = 5\text{ ms}$, the VN method needs 5 to 535 ms to process all the images through the network, whereas the individual object recognition would need 3745.0 ms for the same processing. Note that these are the time estimates as performed on a PC featuring two high-end GPUs, any PC with slower hardware will take longer to process each image which will increase the time disparity between these methods. Using a PC with only CPU could potentially decrease the forward speed through the network by a factor of 10.

5.6. Sensitivity Analysis

This section will discuss the way the system responds to changes in different design choices. This will include dataset size, CNN depth, and noise levels in the cabinet generation.

5.6.1. Dataset Size

Initially, 200 cabinets were generated for training and 200 for testing, each cabinet with a total of 25 images (10 of the ‘full’ state, 10 of the ‘misplaced’ or ‘unfull’ state, and 5 ‘referent’ images). However, increasing the dataset size often increases the performance of the trained model, since the dataset then becomes a better representation of the real data distribution. A perfectly representative dataset has the potential of perfectly training a model. Adding more data after this will not increase the performance

---

1 Based on its performance on ImageNet, which has ‘only’ a 1000 classes but features much more intra-class variation. This means that the classification task per class is much harder on ImageNet than for retail store products, since retail store products usually look exactly the same, while, for example, a cat (a possible ImageNet class) can look very different from another cat but both belong to the same class.
even further. Thus, to find how the performance of the network changes with a change in dataset size, both methods have been trained using 20, 40, 80, 120, 160, 200, 300, 400, 500 and 600 cabinets. The methods have been trained 5 times for each cabinet size. The results can be seen in Figure 5.25. This figure shows an optimum performance for the VN at around 120 - 200 cabinets. Adding more cabinets decreases the performance slightly. The performance of the CNN increases significantly up to 160-200 cabinets. After this, it is rather unstable but seems to increase a few percentage points at 500 cabinets. From this figure, it can be concluded that using 200 cabinets creates a balanced trade-off between performance for both methods and time needed to create datasets and train both methods.

![Accuracy of both methods on different dataset sizes](image)

Figure 5.25: Mean accuracy of both methods when increasing the dataset size for cabinets with misplaced products. It can be seen that the performance peaks between 160 and 200 cabinets, meaning that increasing the dataset size even further won’t improve performance further. The error bars show the standard deviation. Each method is trained 5 times on 200 cabinets and tested on 200 cabinets with unseen products.

5.6.2. Network Architecture

More complex deep learning problems require more complex neural networks to solve. One way to increase the complexity of a network is to add more convolutional layers. A network is sufficiently complex when the performance peaks at a certain level of complexity, adding more complexity after this peak will lower the performance due to overfitting. The effect of adding layers to the network is investigated by training the models using the deeper ResNet variants ResNet34 and ResNet50. Furthermore, the DenseNet121 and DenseNet169 architecture is tested as well, since this architecture has outperformed ResNet in the object recognition domain before [23]. The results can be seen in Figure 5.26. In this graph, it can be seen that adding more layers to the ResNet CNN lowers the performance a bit, likely due to overfitting, while the deeper DenseNet architectures have better performance over the ResNet18 architecture, with the DenseNet121 (DenseNet architecture with 121 layers) achieving the highest accuracy of over 80% when trained 5 times on 200 cabinets. While this is an improvement over the ResNet18 architecture, it comes at the cost of lower speed. The times needed for each network to process an input can be seen in Figure 5.27. Here, it can be seen that increasing the complexity of the network also increases the time needed to process an input and generate an output. The Verification network always takes about twice as long as the CNN, due to the processing of two inputs instead of one. As stated before, this time can be reduced greatly by processing the stored referent images and turning those into their feature vectors beforehand. Doing this allows the network to perform while only needing to process one of the two inputs at runtime, bringing the needed time down to the needed time required by the single input method. From this figure, it can also be seen that the DenseNet121 architecture takes almost 6 times as long as the ResNet18 architecture.
5.6. Sensitivity Analysis

### 5.6.3. Increasing Cabinet Noise Levels in Simulation

Deep neural networks have the potential to be robust to noise by learning to extract only the important features from the inputs. While the generated data already featured noise to some degree in terms of lighting conditions and product positions, the levels of these noise factors can be increased to challenge the network even more. Training and testing both methods with increased randomisation for the lighting colours (anywhere on the RGB spectrum instead of color temperatures limited between 3500 and 5000 Kelvin) in the simulation yielded an accuracy of 50% on the CNN, meaning that this dataset is not well-suited to train the network. Different levels of noise could be explored but is not done here due to time constraints.
5.7. Discussion

For convenience, Table 5.10 shows a summary of all the results. This section will start by discussing the results of the individual experiments in Subsection 5.7.1. Subsequently, some general points worth discussing regarding this research are discussed in subsection 5.7.2.

Table 5.10: Summary table comparing the results gained from each experiment. Speed is estimated in terms of baseline time \( t \) which is the time needed to process a single image using a ResNet18 network, which is dependent on system hardware.

<table>
<thead>
<tr>
<th>Method</th>
<th>Missing (Accuracy)</th>
<th>Misplaced (Accuracy)</th>
<th>Generalisability (Accuracy on 100% unseen products)</th>
<th>Reality Gap</th>
<th>Speed [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>99.8 ± 0.07%</td>
<td>73.14 ± 0.61%</td>
<td>MSP: ≈ 57%</td>
<td>MSS: 64.2%</td>
<td>4.70</td>
</tr>
<tr>
<td>Verification Network</td>
<td>99.41 ± 0.39%</td>
<td>82.54 ± 4.32%</td>
<td>MSP: ≈ 74%</td>
<td>MSS: 84.5%</td>
<td>8.81²</td>
</tr>
</tbody>
</table>

5.7.1. Experiments

This subsection will discuss the results of the individual experiments.

Missing Products

From the results in the previous sections, it can be seen that the VN almost always outperforms the single input image method (referred to as CNN). For missing products, the CNN performs just a little bit better than the VN, but their performance is very comparable. This makes sense because the absence of a product can be detected without needing the knowledge of which product is supposed to be in that location in the cabinet.

Misplaced products

For misplaced products, it is clear that the VN outperforms the CNN significantly. Surprisingly, the CNN still performs quite well. Even though the CNN cannot determine whether a product is misplaced by looking at which product is supposed to be there, it can still deduce the misplacement, probably by looking at the gaps between products. During cabinet generation, an attempt is made to replace products with products that fit that spot best, so that very large gaps can be avoided. This creates a slightly more difficult dataset which should improve the learning rate of the networks, but still allows for the CNN to classify some cabinet images correctly.

Generalisability

When exploring the generalisability for both methods, it is clear that while for missing products neither of the methods suffer from unseen products in the test datasets. However, there is a clear distinction between the CNN and the VN for misplaced products. While both methods decrease in performance as the number of unseen products increases, the VN consistently outperforms the CNN by almost 20 percentage points. Another interesting point that arose during this experiment is that the VN requires fewer different product models in the cabinets used to create the image datasets to get good performance. This is suggested by the fact that during training and testing the methods on cabinets featuring 100 (Experiment 3) or 200 (Experiment 2) product models the VN achieves approximately the same performance (around 85%), while the CNN decreases in accuracy from almost 75% when using 200 product models to less than 62% when using 100 product models. This is a very important conclusion since scarcity in available product models is shown to be less of a problem for the VN than for the CNN, making training of the VN much cheaper than the CNN.

²Note that if the referent images used in the verification network are converted to feature vectors offline, the verification network will be almost as fast as the CNN.
5.7. Discussion

Reality Gap
When testing both methods trained on simulated data on real data the results show that while for cabinets with misplaced products neither method works at all, they do work for cabinets with missing products. Here, the VN outperforms the CNN by a large margin (84.5% vs 64.2%, respectively). So, while the CNN drops in accuracy by about 35 percentage points, the VN only decreases by about 15 percentage points. This shows that by comparing images instead of classifying them without reference improves the transferability from simulation to real data. Furthermore, it is noted that the VN achieves a 100% accuracy on the cabinet featuring products that have rectangular shapes. This suggests that including products with different shapes in the simulation might improve the VN’s performance on the cabinets with differently shaped products as well.

Speed
In terms of speed, the CNN is, as expected, almost twice as fast as the verification network. However, the speed of the VN can be brought up to the speed of the CNN by storing the referent images as extracted feature vectors, eliminating the need for the extraction of features from the image at runtime. Comparing the VN with an individual object recognition approach shows that the VN is much faster than the individual object recognition approach, regardless of the state of the cabinet.

5.7.2. General Points
While running experiments, a few things happened that went against the initial expectations or that have to be kept in mind when concluding this research. The following parts discuss these issues.

Network Retraining
After training the CNN, its model parameter values from the model with the best performance are stored in a local file. When using the VN, these parameters are imported into the network as weight initialisation. If this is not done, the VN does not improve over training. Furthermore, it is important that the network is retrained using the same datasets for training and testing as used for the training of the CNN. This way, the network performance is still measured on a dataset featuring cabinets and products it has not seen during either the retraining of the single CNN or during training of the VN. Secondly, it is important to train the VN with a good weight initialisation. Small differences in weight initialisations can change the performance of the VN significantly. That said, once the VN is properly trained, its performance has shown to be very stable over the variety of datasets that it has been tested on.

Need for Referent Image
The need for a referent image showing all the products in a cabinet can be time-consuming. This referent image can be provided either by stocking the store fully and taking photographs of every cabinet to then save these images in a knowledge base, or to create 3D models of all products and simulate the cabinets as they are supposed to be in the store. The latter is the less preferred option as the products will take a lot of time and effort to model and add to the simulation, while the former method is more accessible since retail stores are preferably fully and correctly stocked most of the time. Furthermore, since retraining is not needed in new retail stores, it is not necessary to create a large dataset using all the products featured in the new store to train the network.

Data Generation
The product model sets have been split into two sets beforehand. These were set up in such a way that both sets can be used individually to generate cabinets. During several experiments, it became clear that the cabinets generated using the 100 products from one product set were not equal to the ones generated using the 100 products from the other set. When training the methods using the arbitrarily called ‘train product set’ and testing them on the ‘test product set’ the performance was consistently better than the methods trained using the test product set and tested using the train product set. Optimally, the product sets wouldn’t be split between two static product sets, but rather during generation the generator randomly picks 100 products from the complete set of 200 products for training, and uses the other 100 for testing. However, due to the way the cabinets are generated this is currently not possible. Since the generation of cabinets with misplaced products takes (a) random product(s) and attempts to replace it with a different product from the set that fits at that location in the cabinet, there
is a high chance that there is no product that meets these requirements. This will cause the generator to get stuck looking for products that do fit.

This problem can be solved by creating more product models; more product models means that there is a larger variety of product sizes and the number of products that can replace other products, decreasing the chance of getting stuck searching for replacement products.

Another solution would be to recognise when a product has no replacements and exclude that product from being replaced from that point on. This, however, is not a very good one as every product should be able to be replaced.
Conclusion & Future Work

This chapter will provide a summary of the conducted research and the answers to the initial research questions as stated in Chapter 1, as well as directions for future work.

6.1. Conclusion

The answer to the question

*How can prior knowledge of the correct location of products in a cabinet help detecting missing or misplaced products in a retail store?*

is that a verification network which produces a similarity score between the cabinets and a referent can help with the detection of misplaced products in retail store cabinets, but at this stage is not ready yet to be deployed in retail stores.

The functional requirements set in Section 1.3.1 can be revisited now that the experiments have been discussed. Each requirement will be repeated and its fulfilment will be stated based on the characteristics the system has shown during the experiments. The requirements have been fulfilled to the following extent:

1. The system should work on a large number of different products;
   - The system is trained and tested on a total of 200 different products. While this is far short of 30,000 or 120,000, the experiments testing the system on unseen products show that even if products are not seen during training, the system still performs well. This is a promising indication that the system can work on a very large product set, too, since it does not have to be trained on all the products to be able to classify the cabinets they are in.

2. The system should be able to analyse a cabinet within seconds;
   - The proposed method can analyse a single image in just a few milliseconds. In the worst-case-scenario (every shelf has a missing or misplaced product), the processing of all the necessary images through the VN alone would take less than a second, which would fit the requirement well. However, this speed is highly dependent on system hardware. This speed is achieved on a workstation with two high-end GPU’s, which is not commonly found on robots due to price, size, and power consumption. Furthermore, there are more operations needed to complete the whole pipeline as shown in Figure 1.2, such as cabinet/shelf/product extraction, image cropping and image normalisation. While the system might be able to meet the requirement on the hardware used for this research, and while the system is significantly faster than the individual object recognition counterpart, it is unlikely that the system meets this requirement on hardware more commonly found in robots.

3. The system should be able to analyse a cabinet regardless of lighting direction and colour, and to a certain extent, intensity;
• The system has been tested using simulated data featuring varying lighting conditions, and has been shown to handle the introduced noise levels well. Due to time constraints, the system has not been tested on real data with a focus on lighting conditions. However, the images taken from the real cabinets did feature different lighting conditions. The system has not been trained on different camera angles, but it was able to handle the slightly different camera angles that occurred while manually taking real photographs of cabinets.

4. The system should be able to generalise to cabinets containing products it has not seen before without the need for retraining.

• The system has been tested on an increasing fraction of unseen products. While the performance degraded slightly, it still performed very well on cabinets with missing products and relatively well on cabinets with misplaced products. Training the system on a larger initial set of products might improve these performances even further.

The main contribution of this research is a method which can analyse whether or not cabinets are fully and correctly stocked with high speed and improved performance, which can be used in environments where the product set might be unseen by the network, and which is transferable from simulation to real data for certain tasks. It is important to note that because the verification network is able to generalise well to unseen products, this method can be used in new retail stores or in a known retail store with a new product set, without the need to retrain the network.

There are some significant issues that prevent this system to be deployed. The main issues are speed and transferability to real data including non-rectangular products. While the latter might be improved by additional tuning and training of the networks with additional product models, the speed of the system is dependent on the available hardware. One solution to this could be to run the system on a server that does feature the necessary hardware capabilities, but this would create the need for such a server to be bought and set up additionally to the robot. Another drawback that is an intrinsic characteristic on the system is that a referent image of every cabinet is needed to analyse that cabinet.

6.2. Future Work
During this research, several issues arose, as well as ideas to improve the system even further. This section will briefly discuss the topics for future work.

6.2.1. Closing the Reality Gap
Currently, the verification network does not generalise to real-world data for cabinets with misplaced products. Transferring from simulated data to real data is a common issue with deep learning trained on simulated data, and there are several techniques to decrease this reality gap. For example, adding real data to the training set. Future work could investigate which technique (or assembly of techniques) works best for this application and show how well the network performs on real data after optimisation.

6.2.2. Network Optimisation
The experiments that are done in this research are aimed to compare a single image network to a network which takes in both an image to be classified and an image containing prior knowledge. While both networks are optimised to a certain extent with regards to the training hyperparameters as stated in Table 5.1, neither is optimised fully and tuning the networks might increase their performance even further.

6.2.3. Include Non-rectangular Products
Currently, the image datasets are generated using cabinets featuring only rectangular products. Testing the VN on a real cabinet featuring only rectangular products where products might be missing achieves a perfect accuracy. Testing it on real cabinets with non-rectangular products causes the performance to drop. Adding non-rectangular products to the data might improve the VN’s performance on cabinets featuring non-rectangular products.
6.2. Future Work

6.2.4. Completing the Pipeline
As mentioned in Chapters 1 and 2, once a cabinet has been verified as being ‘unfull’ or ‘misplaced’, the system should be able to ‘zoom in’ to shelf-level to verify the state of each shelf in the cabinet. If a shelf has missing or misplaced products, then the system should zoom in to verify individual products. This allows the system to determine which products are missing or misplaced in a cabinet, but it might also improve the performance of the system due to the comparison of smaller parts of the cabinet when analysing the individual shelves or products.

6.2.5. Include Reasoning to Set a Similarity Threshold
Currently, this system produces a scalar similarity measure to show how similar or dissimilar two images are in the range \([0, 1]\). However, there is no clear boundary between inputs that are the same and inputs that aren’t. Instead, a threshold has been set at 0.5, meaning that a similarity measure of lower than this threshold results in that the inputs are considered the same, and a similarity measure over the threshold means that the inputs are not the same. While this static threshold might work for cabinets with rectangular products as these are not very different between different images, apart from possibly their position and orientation, while comparing cabinets with more deformable objects might need a different threshold, since comparing two correctly filled cabinets might yield a high similarity value (indicating that the difference is high) due to the different shapes the products might have at that time. Instead of setting the threshold to a static value, an external reasoner might be able to take into account the type of products in a cabinet to set a threshold depending on which cabinet is being verified, and to decide to zoom in on shelf and product level even if images might have a score that suggests they are very similar.

6.2.6. Multiple Product Rows
In the generation of the cabinets, there is only one row of products. This causes the back of the cabinet to be seen when products are missing, and products to be almost flush with the front of the cabinet when they are not missing. However, it is entirely possible that products are not missing but simply one or more rows further back (when only the product on the first row has been taken). Future research should test whether the system is able to classify a product as present even if it is not on the first row from the front. This should be added to the data generation and network testing as well.

6.2.7. Research Product Model Set Size
One of the goals of this research was to develop a method of detecting missing or misplaced products without needing to retrain the networks to be used in new environments, where the products might be different from the products the network has been trained on. This means that the network has to generalise well to unseen products. While this has been explored in Section 5.3, and while the results in that section compared to the first experiments suggests that the VN doesn’t need a lot of product models to perform well on unseen products, what has not been explored is exactly how the number of products that the model has been trained on affects the performance on unseen products. It is possible that the models generalise better to unseen products when they are trained on a larger initial set of products. This can be investigated by creating 100 more product models, then training both models on an increasing product set size (from 100 to the 200 products used during this research) and then test these models on the exact same dataset generated using the new 100 product models.

6.2.8. Combine Missing and Misplaced Products
This research has considered cabinets with missing and cabinets with misplaced products separately. However, it is very much possible that a cabinet has products that are missing as well as products that are misplaced. Future work should investigate how well the verification network performs when it can encounter both missing and misplaced products in a single cabinet.
Acknowledgements

I would like to thank Dr. Ir. Carlos Hernandez Corbato for supervising this thesis. Your feedback has always been very thorough and helped me to guide my work and get it to where it is now.

Furthermore, I would like to thank Ir. Corrado Pezzato. Even though you were not assigned to be my daily supervisor, I could always count on your thoughts on any issues I had, and your feedback on any of the pieces of text I sent you.

Lastly, I would like to thank AIRlab Delft and Ahold Delhaize for providing a place to work and a small test supermarket where I could gather data in peace.

Throughout the COVID-19 pandemic, it was often difficult to stay motivated and to keep on working when I was not able to meet with my peers. Thank you to everyone I reached out to for listening to my ramblings and giving me advice on how to carry on!


Product Distributions Generalisability
Datasets
A.1. Missing Products

A.1.1. MSPTRN

Figure A.1: Product occurrences in dataset MSPTRN

(a) Height occurrences in dataset MSPTRN
(b) Width occurrences in dataset MSPTRN
(c) Shelf number occurrences in dataset MSPTRN
(d) Product number occurrences in dataset MSPTRN

Figure A.2
A.1.2. MSPTST0

Figure A.3: Product occurrences in dataset MSPTST0

(a) Height occurrences in dataset MSPTST0

(b) Width occurrences in dataset MSPTST0

(c) Shelf number occurrences in dataset MSPTST0

(d) Product number occurrences in dataset MSPTST0
### A.1.3. MSPTST20

#### Figure A.5: Product occurrences in dataset MSPTST20

<table>
<thead>
<tr>
<th>Product</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeRuijter muisjes_gestamp</td>
<td></td>
</tr>
<tr>
<td>Duryea maizen</td>
<td></td>
</tr>
<tr>
<td>Unox cupasoup_pittige_tomaa</td>
<td></td>
</tr>
<tr>
<td>Unox cupasoup_tomaa</td>
<td></td>
</tr>
<tr>
<td>Unox cupasoup_tomaat_crem</td>
<td></td>
</tr>
<tr>
<td>Wasa volkore</td>
<td></td>
</tr>
<tr>
<td>Wasa zuurdesem_rogg</td>
<td></td>
</tr>
<tr>
<td>LU cracotte_crackers_nature</td>
<td></td>
</tr>
<tr>
<td>LU cracotte_crackers_volkore</td>
<td></td>
</tr>
<tr>
<td>Unox cupasoup_prei_crem</td>
<td></td>
</tr>
<tr>
<td>Wasa sesa</td>
<td></td>
</tr>
<tr>
<td>Wasa vezelrij</td>
<td></td>
</tr>
<tr>
<td>Wasa zuurdesem_glutenvri</td>
<td></td>
</tr>
<tr>
<td>Wasa zuurdesem_meergrane</td>
<td></td>
</tr>
<tr>
<td>AH hagelslag_mel</td>
<td></td>
</tr>
<tr>
<td>AH hagelslag_puu</td>
<td></td>
</tr>
<tr>
<td>DeRuijter chocoladehagel_puu</td>
<td></td>
</tr>
<tr>
<td>DeRuijter vlokken_puu</td>
<td></td>
</tr>
<tr>
<td>DeRuijter vlokken_wi</td>
<td></td>
</tr>
<tr>
<td>L OR koffiecups_espresso_estrem</td>
<td></td>
</tr>
<tr>
<td>L OR koffiecups_espresso_forz</td>
<td></td>
</tr>
<tr>
<td>L OR koffiecups_espresso_guatemal</td>
<td></td>
</tr>
<tr>
<td>Nescafe gold_cappucin</td>
<td></td>
</tr>
<tr>
<td>Perla koffie_aroma_decaf</td>
<td></td>
</tr>
<tr>
<td>Perla koffie_aroma_roo</td>
<td></td>
</tr>
<tr>
<td>Perla koffiecups_espresso_dar</td>
<td></td>
</tr>
<tr>
<td>Perla koffiecups_lungo_dar</td>
<td></td>
</tr>
<tr>
<td>Unox cupasoup_champignon_crem</td>
<td></td>
</tr>
<tr>
<td>Unox cupasoup_kerri</td>
<td></td>
</tr>
<tr>
<td>DeRuijter chocoladehagel_mel</td>
<td></td>
</tr>
<tr>
<td>AH hagelslag_meer</td>
<td></td>
</tr>
<tr>
<td>AH hagelslag_wi</td>
<td></td>
</tr>
<tr>
<td>Dorset muesli_cranberry_cherry_almond</td>
<td></td>
</tr>
<tr>
<td>Dorset muesli_fibr</td>
<td></td>
</tr>
<tr>
<td>Heinz_tomato_gezeef</td>
<td></td>
</tr>
<tr>
<td>L OR koffiecups_espresso_decafeinat</td>
<td></td>
</tr>
<tr>
<td>L OR koffiecups_espresso_india</td>
<td></td>
</tr>
<tr>
<td>L OR koffiecups_espresso_Colombia</td>
<td></td>
</tr>
<tr>
<td>L OR koffiecups_espresso_Colombia</td>
<td></td>
</tr>
<tr>
<td>MORA oven_kipkor</td>
<td></td>
</tr>
<tr>
<td>MORA oven_kipnugget</td>
<td></td>
</tr>
<tr>
<td>MORA oven_rundvlees_bitterball</td>
<td></td>
</tr>
<tr>
<td>MORA oven_bamischijve</td>
<td></td>
</tr>
<tr>
<td>MORA oven_groente_bitterballe</td>
<td></td>
</tr>
<tr>
<td>ARLA bio_karnemel</td>
<td></td>
</tr>
<tr>
<td>ARLA bio_mel</td>
<td></td>
</tr>
<tr>
<td>ARLA bio_mel_halfvo</td>
<td></td>
</tr>
<tr>
<td>AH krokante_muesli_appel_rozij</td>
<td></td>
</tr>
<tr>
<td>AH thee_zwart_sinasapex</td>
<td></td>
</tr>
<tr>
<td>AH thee_zwart_winter</td>
<td></td>
</tr>
<tr>
<td>CarribeanGold_rietsuikerklontje</td>
<td></td>
</tr>
<tr>
<td>Crystal Clear_lemon_passionfruit</td>
<td></td>
</tr>
<tr>
<td>Crystal Clear_orange_mandarinx</td>
<td></td>
</tr>
<tr>
<td>Crystal Clear_pear_coconut</td>
<td></td>
</tr>
<tr>
<td>DrOetker_brownie</td>
<td></td>
</tr>
<tr>
<td>DrOetker_cupcake</td>
<td></td>
</tr>
<tr>
<td>Quaker cruesli_choclate</td>
<td></td>
</tr>
<tr>
<td>Quaker cruesli_cookies_cream</td>
<td></td>
</tr>
<tr>
<td>Quaker cruesli_red_fruit</td>
<td></td>
</tr>
<tr>
<td>Quaker cruesli_luchti</td>
<td></td>
</tr>
<tr>
<td>Wasa goudbrand</td>
<td></td>
</tr>
<tr>
<td>Wasa leichtgewicht</td>
<td></td>
</tr>
<tr>
<td>Zaanse_hoeve_volle_yoghurt</td>
<td></td>
</tr>
<tr>
<td>Zaanse_hoeve_karnemel</td>
<td></td>
</tr>
<tr>
<td>Zaanse_hoeve_vanille_yoghurt</td>
<td></td>
</tr>
<tr>
<td>AH krokante_muesli_choco</td>
<td></td>
</tr>
<tr>
<td>AH krokante_muesli_note</td>
<td></td>
</tr>
<tr>
<td>AH thee_zwart_winter</td>
<td></td>
</tr>
<tr>
<td>ARLA bio_yoghurt_vanil</td>
<td></td>
</tr>
<tr>
<td>ARLA bio_yoghurt_vanil</td>
<td></td>
</tr>
<tr>
<td>Eat Natural_ancient_grain</td>
<td></td>
</tr>
<tr>
<td>Eat Natural_boekweit</td>
<td></td>
</tr>
<tr>
<td>Eat Natural_noten_zade</td>
<td></td>
</tr>
<tr>
<td>Heinz_tomato_frit</td>
<td></td>
</tr>
<tr>
<td>Crystal Clear_cranberry_lime</td>
<td></td>
</tr>
<tr>
<td>Crystal Clear_ginko_apple</td>
<td></td>
</tr>
<tr>
<td>Crystal Clear_ginko_grape</td>
<td></td>
</tr>
<tr>
<td>Crystal Clear_apple_sapex</td>
<td></td>
</tr>
<tr>
<td>Crystal Clear_apple_vanil</td>
<td></td>
</tr>
<tr>
<td>DrOetker_brownie</td>
<td></td>
</tr>
<tr>
<td>DrOetker_cupcake</td>
<td></td>
</tr>
<tr>
<td>Quaker cruesli_hazelnut_honey</td>
<td></td>
</tr>
<tr>
<td>Quaker cruesli_luchti</td>
<td></td>
</tr>
<tr>
<td>Wasa_goudbrand</td>
<td></td>
</tr>
<tr>
<td>Wasa_lightweight</td>
<td></td>
</tr>
<tr>
<td>Zaanse_hoeve_karnemel</td>
<td></td>
</tr>
<tr>
<td>Zaanse_hoeve_vanille_yoghurt</td>
<td></td>
</tr>
</tbody>
</table>

#### Figure A.6

- (a) Height occurrences in dataset MSPTST20
- (b) Width occurrences in dataset MSPTST20
- (c) Shelf number occurrences in dataset MSPTST20
- (d) Product number occurrences in dataset MSPTST20
A.1. Missing Products

A.1.4. MSPTST40

Figure A.7: Product occurrences in dataset MSPTST40

(a) Height occurrences in dataset MSPTST40

(b) Width occurrences in dataset MSPTST40

(c) Shelf number occurrences in dataset MSPTST40

(d) Product number occurrences in dataset MSPTST40

Figure A.8
A.1.5. MSPTST60

Figure A.9: Product occurrences in dataset MSPTST60

(a) Height occurrences in dataset MSPTST60

(b) Width occurrences in dataset MSPTST60

(c) Shelf number occurrences in dataset MSPTST60

(d) Product number occurrences in dataset MSPTST60
A.1.6. MSPTST80

Figure A.11: Product occurrences in dataset MSPTST80

(a) Height occurrences in dataset MSPTST80
(b) Width occurrences in dataset MSPTST80
(c) Shelf number occurrences in dataset MSPTST80
(d) Product number occurrences in dataset MSPTST80
A.1.7. MSPTST100

Figure A.13: Product occurrences in dataset MSPTST100

(a) Height occurrences in dataset MSPTST100
(b) Width occurrences in dataset MSPTST100
(c) Shelf number occurrences in dataset MSPTST100
(d) Product number occurrences in dataset MSPTST100
A.2. Misplaced Products

A.2.1. MSSTRN

Figure A.15: Product occurrences in dataset MSSTRN

(a) Height occurrences in dataset MSSTRN
(b) Width occurrences in dataset MSSTRN
(c) Shelf number occurrences in dataset MSSTRN
(d) Product number occurrences in dataset MSSTRN
A.2.2. MSSTST0

Number of occurrences of each product over all cabinets in dataset MSSTST0

Figure A.17: Product occurrences in dataset MSSTST0

(a) Height occurrences in dataset MSSTST0

(b) Width occurrences in dataset MSSTST0

(c) Shelf number occurrences in dataset MSSTST0

(d) Product number occurrences in dataset MSSTST0

Figure A.18
A.2. Misplaced Products

A.2.3. MSSTST20

Figure A.19: Product occurrences in dataset MSSTST20

(a) Height occurrences in dataset MSSTST20

(b) Width occurrences in dataset MSSTST20

(c) Shelf number occurrences in dataset MSSTST20

(d) Product number occurrences in dataset MSSTST20
A.2.4. MSSTST40

Figure A.21: Product occurrences in dataset MSSTST40

(a) Height occurrences in dataset MSSTST40
(b) Width occurrences in dataset MSSTST40
(c) Shelf number occurrences in dataset MSSTST40
(d) Product number occurrences in dataset MSSTST40
A.2.5. MSSTST60
A.2.6. MSSTST80

Figure A.25: Product occurrences in dataset MSSTST80

(a) Height occurrences in dataset MSSTST80
(b) Width occurrences in dataset MSSTST80
(c) Shelf number occurrences in dataset MSSTST80
(d) Product number occurrences in dataset MSSTST80

Figure A.26
A.2. Misplaced Products

A.2.7. MSSTST100

![Number of occurrences of each product over all cabinets in dataset MSSTST100](image)

Figure A.27: Product occurrences in dataset MSSTST100

![Product height distribution](image)

(a) Height occurrences in dataset MSSTST100

![Product width distribution](image)

(b) Width occurrences in dataset MSSTST100

![Number of shelves occurrences](image)

(c) Shelf number occurrences in dataset MSSTST100

![Product count distribution](image)

(d) Product number occurrences in dataset MSSTST100

Figure A.28