Deep Learning Based Image Segmentation of RGB-D Data in Warehouse Automation

Isa El Doori

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Deep Learning Based Image Segmentation of RGB-D Data in Warehouse Automation

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Isa El Doori

June 20, 2019

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

Abstract

The ability to locate specific objects within images is an essential step in various computer vision based engineering applications. Image segmentation is the task of dividing an image into "segments" that are uniform as well as homogeneous with respect to some characteristics, for example grey tone or texture as in Haralick *et al.*

This thesis seeks to perform image segmentation using a Deep Learning (DL) approach in the area of warehouse automation, specifically focusing on an order picking use case of Vanderlande Industries (VI). Generally in literature, DL algorithms for image segmentation are split into two main classes: algorithms for RGB images and algorithms for RGB-D images. RGB stands for the Red, Green, and Blue values of a pixel. RGB-D stands for the Red, Green, Blue, and Depth values of a pixel. The depth value in this case differs from the RGB values in that it does not give a value for a colour intensity, but rather it gives a value for physical distance between the camera and the object it is capturing.

The challenge addressed by this thesis focuses on whether the introduction of depth data results in a substantially better performance than using RGB-only images, based on a dataset provided by VI. Also, this thesis looks into the maximum allowed deviation along the X-axis in the registration of the depth data to the RGB images.

Two networks from literature were investigated and implemented in MATLAB for this purpose: the SegNet architecture proposed by Badrinarayanan *et al.* and the FuseNet architecture proposed by Hazirbas *et al.* Through experiments we have found that, for this use case, the introduction of complementary depth data leads to an improvement over the use of RGB-only images. We also find that, for this use case, the maximum allowed deviation along the X-axis in the registration of the depth data to the RGB images is approximately equal to 1.67 millimetres.

The results in this thesis seem to indicate that investing in acquiring an additional depth band does have a positive effect on the accuracy of image segmentation for order picking in warehouse automation.

Table of Contents

List of Figures

List of Tables

Preface

This thesis project came to be after speaking to both Vanderlande Industries and the Delft University of Technology. A Machine Learning topic is not a topic often pursued by students from DCSC. However, my will to carry out a project within the field of Artificial Intelligence, and with the help from Vanderlande Industries, have made this project possible.

Acknowledgements

I would like to thank my supervisor dr. ing. Raf Van de Plas for all of his assistance during the entirety of this thesis project. Also, many thanks to all the people from Vanderlande Industries for giving me the opportunity to work on a project that entails state-of-the-art research and development within the industry. Special thanks to Mariana Goldak, Caglar Saneras, and Martin Plantinga from Vanderlande Industries, who have supervised me on a daily basis. I would also like to thank the many students from Vanderlande Industries and the university which whom I have had many discussions on how to successfully conduct this thesis project.

Delft, University of Technology Isa El Doori
June 20, 2019

"A computer would deserve to be called intelligent if it could deceive a human into believing that it was human."

— *Alan Turing*

Chapter 1

Introduction

Warehouse automation has become one of the major focus points of automation companies, such as Vanderlande Industries (VI). The rise of e-commerce models has sparked an uprise in warehouse automation. The demand from consumers for products and fast delivery times are at an all time high. According to an article from the Dutch Central Bureau of Statistics (CBS), the online share of Dutch consumers has risen from 30% in 2007 to 79% in 2017 [11]. The same article also states that products not arriving on time is the most common problem consumers face, with 24% of Dutch online consumers claiming they experienced a late delivery of their ordered products. This has sparked a need of automatisation in warehouse processes. These processes consist of every step from the moment an order arrives at a warehouse to the actual shipping of the order to the consumer. In recent years, the use of cameras has been significantly incorporated in warehouse automation processes. This thesis will address one specific application of computer vision to the order picking process of the warehouse automation process.

1-1 Order picking process

Whenever an order comes in at a warehouse, different processes are triggered. Depending on how the warehouse has structured its products, the order will be processed and shipped to the consumer. When considering a small warehouse containing only three products, products A, B, and C, one can see how the processes unfold. Consider all of the products are placed in three different trays, where each tray is filled with the same type of product and an order comes in for products A and C. A human controller is available to physically grab one item from tray A and one item from tray C and places them in a box ready for shipment. One of the methods to automate this process is called order picking, where a robot recognises the order, knows the products and their location within a tray, and picks-and-places the items in a box ready for shipment. The main focus of this thesis is the recognition of the product shapes and their locations within a tray by solving the image segmentation task using a Deep Learning (DL) approach. In the next section, we will briefly discuss what image segmentation and DL are.

1-2 Image segmentation

For various kinds of applications, it would be useful to know the location of specific objects within images. Also, the type of object, and the primitive shape of the object can be important information for these applications. As discussed in the previous section, an example of such an application would be order picking at a warehouse, where one would like to know what the location is of a specific product that needs to be picked. Also, autonomous driving applications would be an example, where it is vital to know what the location of pedestrians walking in front of the vehicles is. This task is often called the segmentation of images. Image segmentation has been a computer vision problem for several decades. The term segmentation was first used in Rosenfeld *et al.* [12]. However, others had been working on solving the segmentation problem without explicitly calling it segmentation. For example, Roberts *et al.* [13] developed an algorithm that detects edges of a three-dimensional solid in a two-dimensional image. This means that everything inside these edges, or boundaries, belong to a certain object. Over subsequent years, researchers had looked into the possibilities to solve the image segmentation task automatically and in a supervised way. This means that for a certain amount of example data, the desired output has to be known a priori. These techniques to solve the task automatically are also known within the area of Artificial Intelligence (AI) [14, 15]. According to Coleman *et al.* [15], the main issue of these AI techniques is that the algorithms require several hours of computing power in the training phase, as the algorithms are computationally expensive.

However, at the time of [15], it was just the beginning of the information age, with relatively slow CPUs and a relative shortage of data. Both of these aspects are, in most cases, necessary for these AI techniques to work properly. Digital advances since the time of [15] have been made in terms of faster CPUs and GPU computing, where the calculations can be done in parallel. Also, the internet has become a source of available data, with people uploading millions of images every day. As discussed by Sakai *et al.* [14], supervised AI techniques need a certain amount of example data to be known a priori. With the availability of datasets like the ImageNet dataset [16], most researchers now have the tools and data to propose new AI solutions to the Image Segmentation task. Specifically, in computer vision, Deep Learning (DL) has gained popularity in recent years. Since [14] and [15], it has become generally accepted that AI techniques can solve the image segmentation task with similar results as other solutions. Therefore, it should not come as a surprise that DL would be able to solve the image segmentation task as well. Furthermore, according to Sermanet *et al.* [17], DL techniques are capable of solving other computer vision tasks with surprising results.

Even though image segmentation is an interesting task, the field of computer vision also entails several other tasks that are related. One can look at most computer vision tasks as a single task with several sub-steps [17]. One of the first steps or challenges within computer vision is object recognition. Recognition is, as the name implies, the task of recognising the object that is portrayed in the image. For example, if the algorithm receives an input image of a cat, the algorithm needs to recognise it as a cat and classify the image as an image that contains a cat. This task is often called image classification. The next step or challenge within computer vision is object localisation. As the name implies, localisation is the task of localising the object that is portrayed within the image. Continuing our example of the cat: after the algorithm has classified the image as an image that contains a cat, it would then declare boundaries to specify the approximate location of the cat in the image.

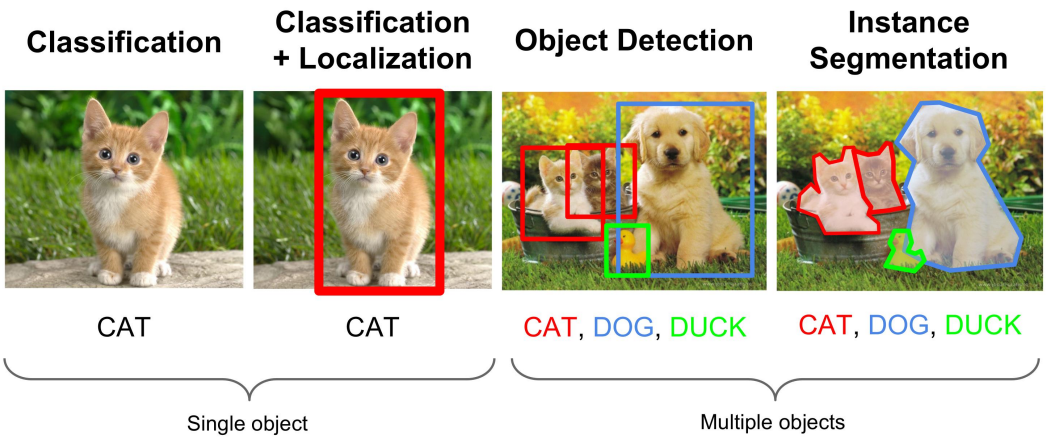


Figure 1-1: Example of different types of computer vision tasks. From left to right this image shows the different computer vision problems, i.e. Classification, Classification and Localization, Object Detection, and Instance Segmentation. The first two tasks can mostly be applied to single object problems, the second two tasks can mostly be applied to multiple objects. Source: van Loon *et al.* [3].

Here, we can see that for object localisation, one usually needs to solve the image classification problem first. The third step or challenge of the computer vision is object detection. Detection is the task of detecting if there is an object portrayed in the image or not. Object detection is basically the same as object localisation, with the additional step of localising all objects present in an image and classifying them accordingly to their respective class, and defining a background class when no object is present. Continuing our cat example: the detection algorithm has to be able to tell us whenever nothing is portrayed in the image, and when a cat is portrayed in the image, which usually means that it needs to localise the cat in the image. Furthermore, whenever multiple object classes are present in the image, the algorithm needs to be able to detect them as well [17]. Continuing to the next step brings us to image segmentation. This time, it is not as obvious from the name as the previous three problems. What image segmentation really is will be discussed in chapter 2 of this thesis. See Figure 1-1 for an example of these different kinds of computer vision problems.

1-3 Problem statement

As discussed in the previous section, it would be useful for the order picking process to implement automated image segmentation. Generally in literature, the DL algorithms for image segmentation are split into two classes: algorithms for RGB images and algorithms for RGB-D images, where RGB stands for the Red, Green, and Blue values of a pixel, and RGB-D stands for the Red, Green, Blue, and Depth values of a pixel. The depth value in this case differs from the RGB values in that it does not give a value for a colour intensity, rather it gives a value of a physical distance between the camera and the object it is capturing.

Together with Vanderlande Industries (VI), the following questions have been researched during this thesis work:

- **How does the use of RGB-D data affect the performance compared to the usage of RGB-only data, applied to the VI dataset?**
- **How much deviation is allowed in the registration of the depth data to the RGB data?**

To answer these questions, this thesis will implement two different Deep Learning (DL) architectures obtained from literature. These two networks are trained on data provided by VI.

1-4 Outline

This thesis consists of five chapters. Chapter 2 will provide a theoretical background on image segmentation, DL, and the DL architectures that are implemented during the thesis work. In Chapter 3, this thesis shall explain the experimental setup, the data provided by VI, and how the DL architectures were implemented. Chapter 4 will explain all of the different experiments conducted during the thesis work. In Chapter 5, this thesis will discuss the results obtained from the experiments conducted in Chapter 4. Finally, this thesis will have a conclusion in Chapter 6.

Chapter 2

Background

We have to establish a few items of general information before we can discuss the DL-specific algorithms implemented in this thesis. This chapter will first discuss the definition of image segmentation in section 2-1. Thereafter, we will discuss the definition and use of DL algorithms in section 2-2. Finally, we will discuss the architectures of the implemented networks in section 2-3.

2-1 What is image segmentation?

In literature, there is no one single definition of image segmentation. For example, Rosenfeld *et al.* [12] defines image segmentation to be the operation of singling out the appropriate picture subsets, thus dividing the image into "segments". Coleman *et al.* [15] gives a more strict definition, these subsets also have to be homogeneous. A homogeneous segment is defined by Coleman *et al.* [15] as a segment that is similar within itself. When a segment mostly consists of a cat, for example, and partly consists of a dog, the segment would not homogeneous according to Coleman *et al.* [15]. Haralick *et al.* [18] is even stricter, stating that regions of an image segmentation should be uniform as well as homogeneous with respect to some characteristics such as grey tone or texture. A uniform segment is defined by Haralick *et al.* [18] as a segment that is identical within itself. When a segment consists of a cat, but contains grey tones, for example, that are not similar to the grey tones of the segment in general, the segment would not be uniform according to Haralick *et al.* [18]. A general observation from the literature seems to be that the more research is done on image segmentation, the stricter the definitions tend to become. The objective for Shi *et al.* [19], for example, is to use the low-level knowledge attributes, such as consistency of brightness, colour, texture, or motion, to sequentially come up with hierarchical partitions. Also, according to Rosenfeld *et al.* [12], there is no one approach to the image segmentation task. More recent research, however, primarily seems to be concentrated on a pixel-wise approach, e.g. [20, 21, 22, 23]. An approach is considered a pixel-wise approach when the proposed algorithm uses the pixel attributes of the segments, e.g. grey scale, texture, brightness, colour, etc.

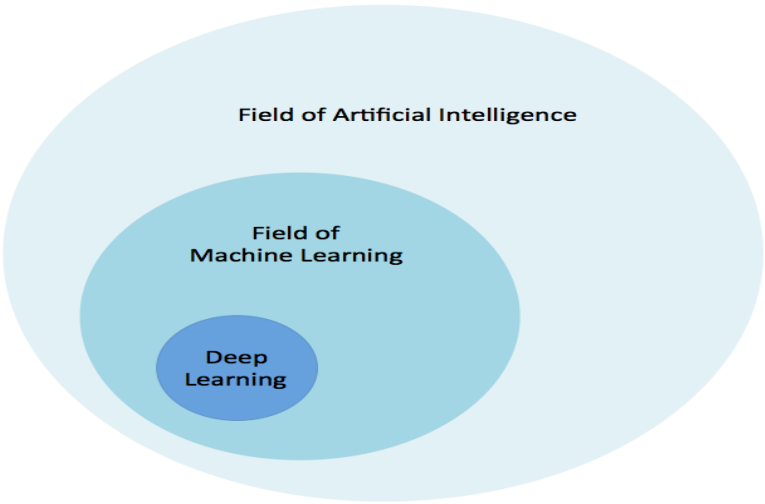


Figure 2-1: Where Deep Learning fits in the spectrum of Artificial Intelligence. According to Patterson *et al.* [4], Deep Learning is a subset of the field of Machine Learning, which is a subset of the field of Artificial Intelligence. This image gives us an idea on how broad the field of Artificial Intelligence actually is. Source: Patterson *et al.* [4].

In this thesis, we will consider the definition of image segmentation provided by Haralick *et al.* [18], since most of the solutions proposed by literature follow this definition. Also, for the same reason, we will consider pixel-wise approaches to the image segmentation task.

2-1-1 Semantic image segmentation

The idea of semantics is, in most cases, to link a class label to the segmented object. Generally, semantics were introduced to the image segmentation task since DL gained popularity, see chapter 1 and Figure 1-1. In general, there are very little classical image segmentation algorithms available in literature that also define the semantic label of the segmented object. This thesis will only consider semantic image segmentation when discussing the DL approaches to the image segmentation task.

2-2 What is Deep Learning?

To put Deep Learning (DL) in context, Figure 2-1 illustrates the relationship between Artificial Intelligence (AI), Machine Learning (ML), and DL. As can be seen from Figure 2-1), DL is a subset of the field of ML, which is in itself a subset of AI [4]. The field of AI is broad and has a substantial history. This thesis will provide a short introduction here. To have an idea of how machines can learn, we first need to take a look to what we mean by "learning". Different definitions have been given over the years. Mitchell *et al.* [24] defines a learning algorithm as: "a computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.". A different definition, but still somewhat similar, is given by Bekkerman *et al.* [25] as:

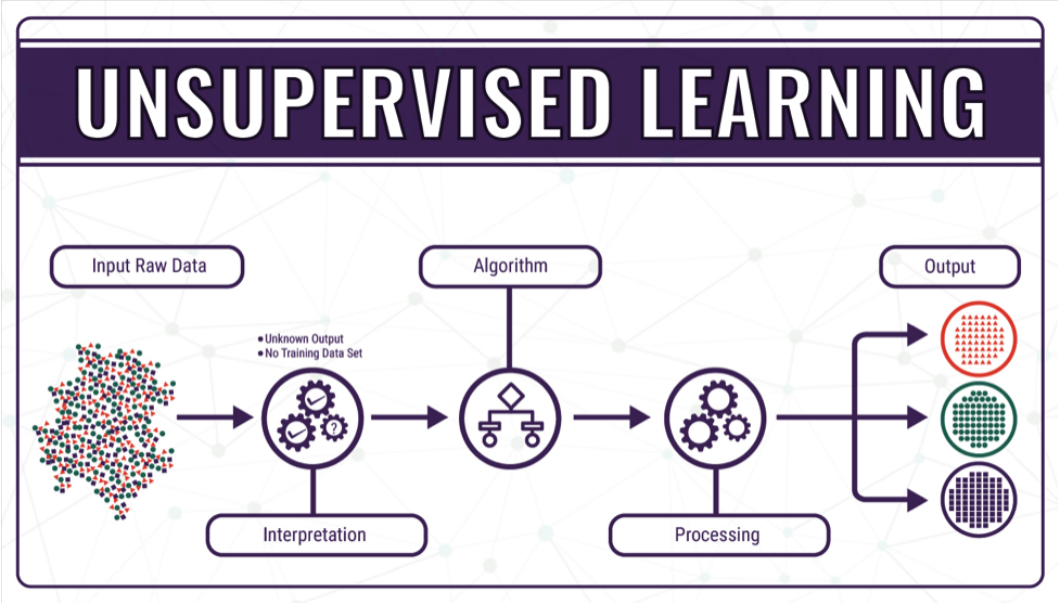


Figure 2-2: Unsupervised Learning algorithm. From the image we can see that raw data is sent to be interpreted first before being sent to the algorithm. The interpretation in this case will try to find relevant information or patterns. After some processing, the unsupervised learning algorithm will product its output. Source: Smolyansky *et al.* [5].

"a machine learning task (function) aims to identify (to *learn*) a function $f : \mathcal{X} \to \mathcal{Y}$ that maps input domain $\mathcal{X}$ (of data) onto output domain $\mathcal{Y}$ (of possible predictions)." ML algorithms can be broadly categorised into three main categories, **Unsupervised**, **Supervised**, or other types of algorithms, on the basis of what kind of experience E they are allowed to use during the learning process [26]. **Unsupervised Learning** algorithms mostly try to predict a certain output using only input information. The algorithm learns what it perceives as underlying properties of the input data-set. Different algorithms can be used to perform different tasks, for example a k-means Iterative Fisher algorithm can be used for clustering tasks, trying to divide the input data into different clusters that are "similar" [27]. **Supervised Learning** algorithms mostly try to predict a certain output using input information, as well as a **label** or desired **target** [26]. In contrast to unsupervised learning algorithms, which learn properties the algorithm itself perceives as useful, a supervised learning algorithm attempts to learn the specified target. It does this by minimising a certain error that it computes by comparing the network's output prediction to the desired target. The algorithm then adjusts its internal model accordingly. A simple example would be a black-box system identification task, where only the input and output of the system are known. Another example that can be solved by using a DL approach, would be certain classification tasks. For example, Lee *et al.* [28] make use of convolutional deep belief networks for audio classification, i.e. speech recognition. To predict future inputs, a supervised DL algorithm can be used to replicate the system's behaviour. However, some ML algorithms do not just predict their output using a fixed data-set. For example, **Reinforcement Learning**, interacts with an environment, so there is a feedback loop between the learning system and its output [26]. See Figures 2-3, 2-2, and 2-4 for the visual representations of the supervised, unsupervised, and reinforcement Learning algorithms respectively.

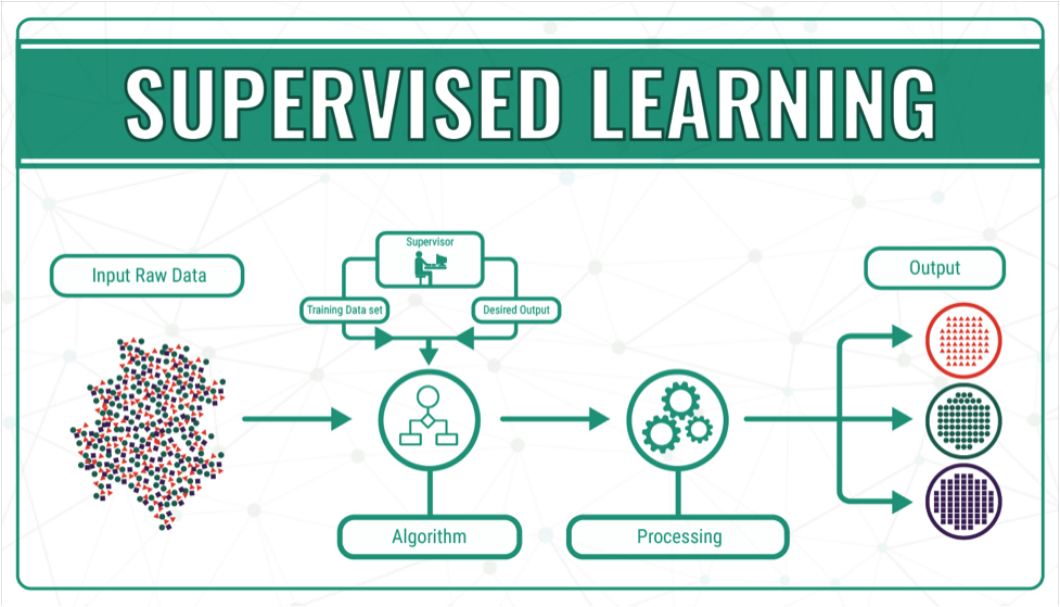


Figure 2-3: Supervised Learning algorithm. From the image we can see that raw data is sent to the algorithm. After the algorithm is done with it, a "supervisor" will compare the output generated from the algorithm to the desired output. After some processing, the supervised learning algorithm will produce its output. Source: Smolyansky *et al.* [5].

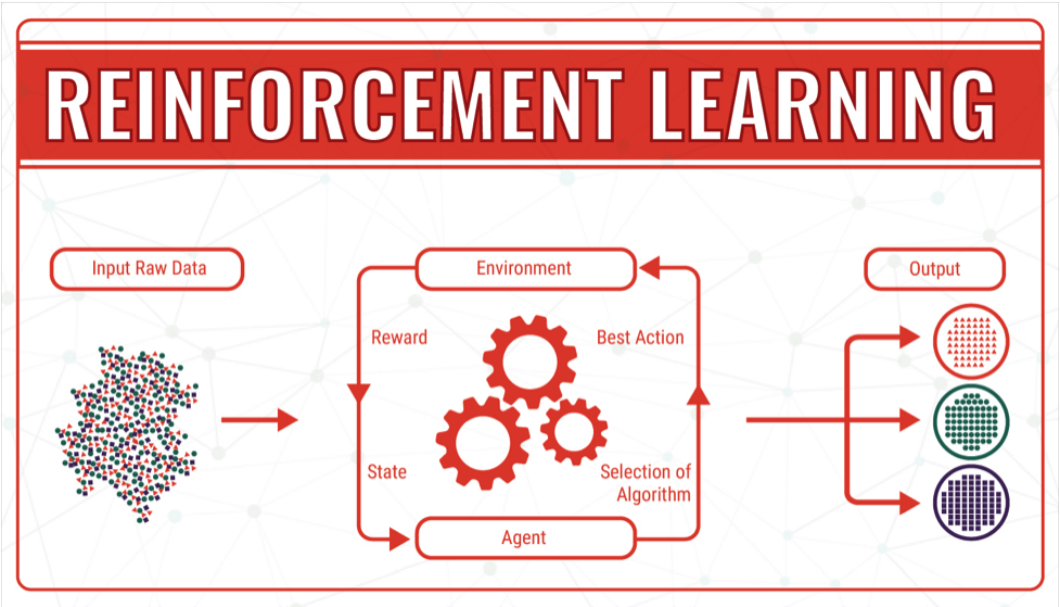


Figure 2-4: Reinforcement Learning algorithm. From the image we can see that raw data is being sent to a feedback loop containing an agent and an environment. The agent selects a certain algorithm that yields in the calculation of the best action that could be taken. The action will be compared to the environment, which yields a certain reward and a certain state that the system is in after applying the action. Whenever the reward, and thus the state, are not desired, the agent chooses a different approach. After the desired state has been reached, the reinforcement learning algorithm produces its output. Source: Smolyansky *et al.* [5].

DL can be used in supervised, unsupervised, and reinforcement learning algorithms respectively [29]. DL algorithms consists of so-called Deep Artificial Neural Networks, commonly called Neural Networks (NNs). A standard Neural Network (NN) consists of many simple, connected units called neurons, each producing a sequence of real-valued activations [26]. Input neurons get activated through sensors perceiving the environment, other neurons get activated through weighted connections from previously active neurons [29]. Learning occurs when the network has found certain weights that produce a desirable output. Generally, when one talks about NNs, one is talking about Feedforward NNs. Feedforward is used here in the sense that there is no closed loop between the layers. In the case where there are closed loops between the layers, we generally speak about Recurrent Artificial Neural Networks (RNNs). See Figures 2-5 and 2-6 for the visual representations of a Feedforward NN and a RNN respectively. The goal of NNs is thus to find these weights that produce a desirable output [29]. The problem of finding these weights can mathematically be put as an optimisation problem of minimising an objective function. For example, the "XOR" problem discussed in [26], would be minimising the objective function $J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\boldsymbol{x} \in \mathbb{X}} (f^*(\boldsymbol{x}) - f(\boldsymbol{x}; \boldsymbol{\theta}))^2$, where $\boldsymbol{x}$ represent the input values, and $\boldsymbol{\theta}$ represent the parameters of the model. Goodfellow *et al.* [26] further states that $f(\boldsymbol{x}; \boldsymbol{\theta})$ could be a linear model with θ consisting of the weights $\boldsymbol{w}$ and b. Thus, their model is defined by $f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^\top \boldsymbol{w} + b$. By minimising the objective function $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{w}$ and b, Goodfellow *et al.* [26] is able to find the weights of the NN corresponding to the "XOR" problem. There are different kinds of objective functions for different kinds of tasks. Therefore, there are also different kinds of optimisation algorithms one can use to solve these optimisation problems. The most used algorithms are the gradient-based algorithms, such as the back-propagation algorithm [26].

There are different kinds of NNs for different kinds of tasks. The type of NNs mostly used in image segmentation are Convolutional Neural Networks (CNNs), introduced by Fukushima *et al.* [30, 31, 32] and put into practice by LeCun *et al.* [33]. Convolution is the act of applying a filter, better known as a "kernel", on a matrix for various types of feature extractions. Different types of kernels extract different types of features. Therefore, the outputs of these convolutional layers are called feature maps [26]. In addition to the convolutional layer, the CNN architecture also entails a few other layers. The "Rectified Linear Unit" (ReLU) layer replaces all negative values in the feature map by zero. This layer is mostly applied directly after the convolutional layer, and the output of this layer is called the rectified feature map. The "pooling" layer reduces the dimension of a feature map by only considering the most important information and leaving out the rest. For example, a max-pooling layer only considers the maximum value of a spatial window of the original feature map. A "fully connected" layer is a traditional NN where all neurons of the previous layer are connected to all neurons of the following layer. This operation is done in order to classify an input to a specified output. In the case of image segmentation it is used to classify an input pixel to an object class [26].

When all of these layers are combined, one can start training the network using the back-propagation algorithm, for example [33]. However, the number of data used will mostly determine the performance of the network. The first version of the GoogLenet network [34], for example, uses 1.2 million images to train their network. Since available data can be scarce, other training methods have become available, which do not require millions of images. The method mostly used in image segmentation is Transfer Learning (TL).

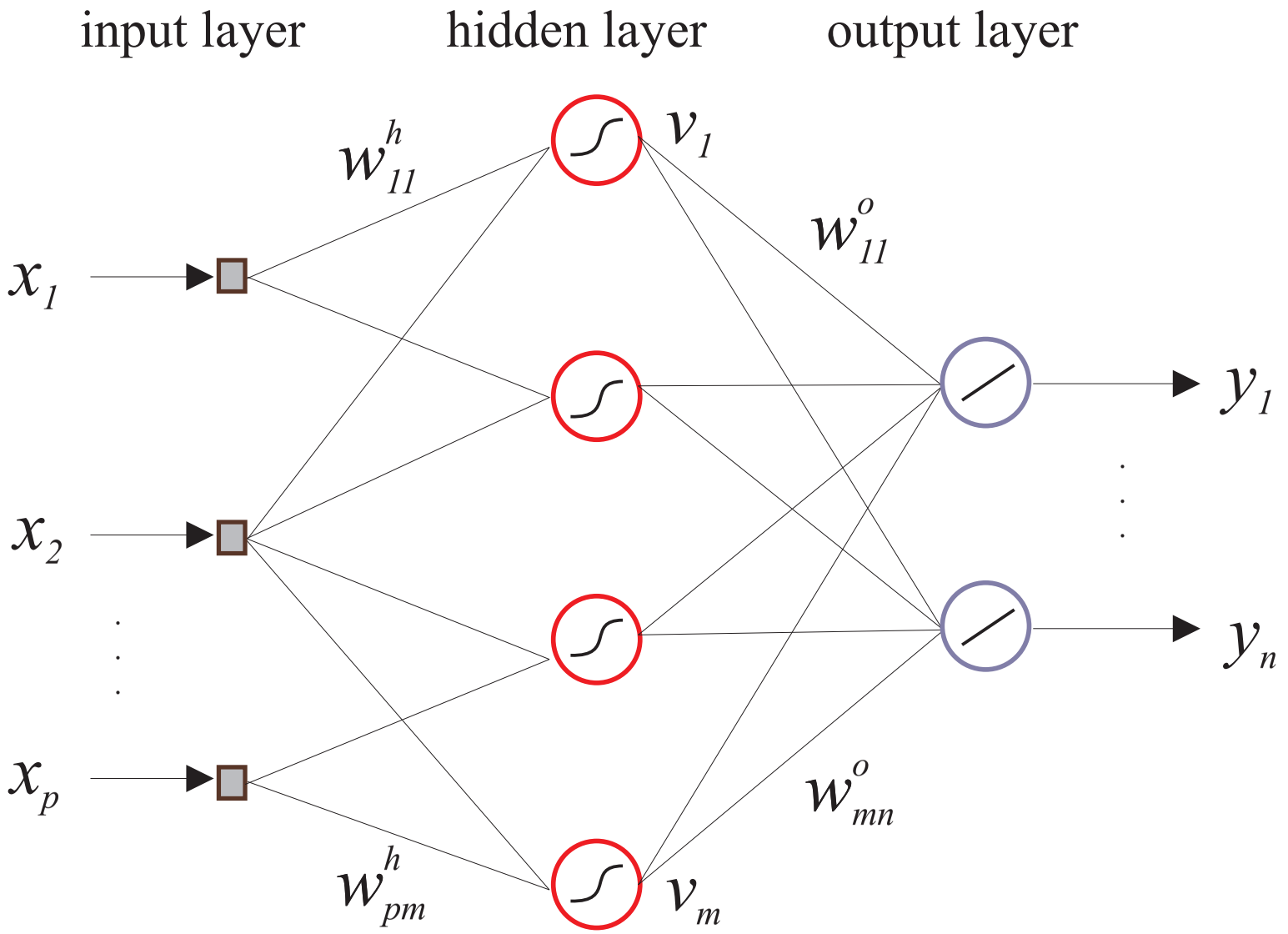


Figure 2-5: A Feedforward NN with a single hidden layer. x_1, x_2, and x_p denote the inputs given to the network. The connections from the input layer neurons to the hidden layers neurons, v_1 to v_m all have certain weights, w^h_{11}, to w^h_{pm}. Also, the connections from the hidden layer neurons to the output layer neurons y_1 to y_n all have certain weights w^o_{11} to w^o_{mn}. The goal of the NN is to find the optimal weights where the error between the output y_1 to y_n and the desired output y^d_1 to y^d_n is minimal. Source: Babuska *et al.* [6]

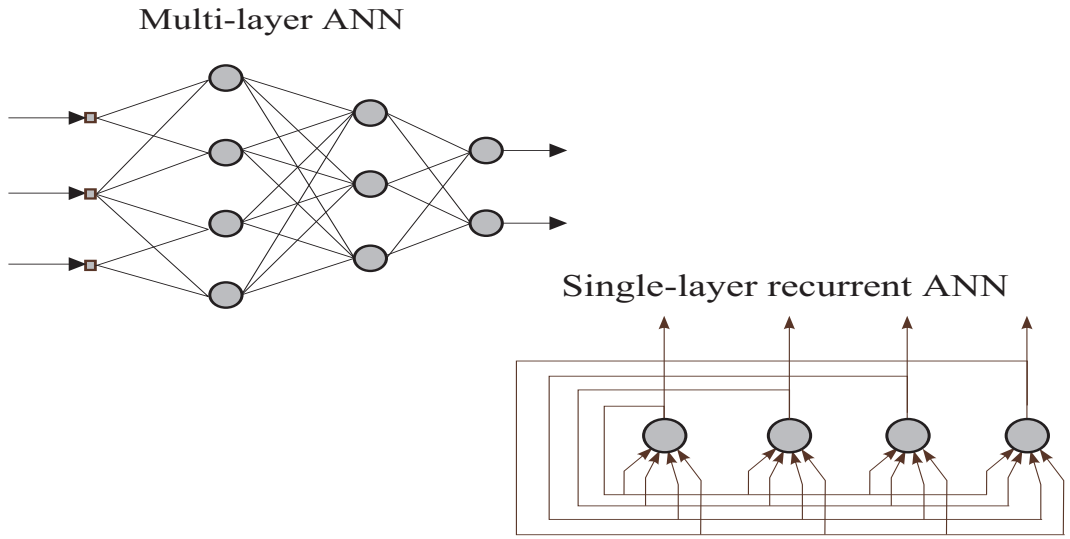


Figure 2-6: A multi-layer ANN and a single-layer recurrent ANN. The multi-layer ANN has the same structure as a Feedforward NN, only with multiple hidden layers. The recurrent ANN also has the same structure, with the main difference of it having a feedback loop incorporated after each layer. This means that the output weights of the neurons are also taken into account. The main objective remains the same for both types, to find the optimal weights. Source: Babuska *et al.* [6]

TL is the act of reusing knowledge attained from past, often related, tasks to make it easier to learn a new task [35]. Following the cat example set in chapter 1 and Figure 1-1, TL would be useful if one has a network available for the classification of the cat object class, but would like to extend the network to also classify dogs. Since the new task is related to the old task, i.e. the classification of animals, one does not need to train the network from scratch for both classes. Also, using TL to retrain the existing network to classify the new class would in most cases require less data [35].

2-3 Implemented algorithms

Now that we have established a background for image segmentation in general, we can start exploring several DL image segmentation algorithms. As stated in section 2-2, DL can be used in supervised, unsupervised, and reinforcement learning approaches [29]. This thesis will only consider supervised learning approaches since unsupervised and reinforcement learning approaches are generally not common in image segmentation. Generally in literature, the approach of combining RGB and depth information into a CNN architecture is done by two methods. “Early fusion” is the method of introducing a fourth channel at the beginning of the architecture and modifying the architecture accordingly. Modifying the architecture in this sense means that all of the layers of the existing architecture should be modified for a four-channel input. How this is done differs from architecture to architecture. “Late fusion” or some form of “late fusion” is the method of introducing a separate “branch” just for the depth information and fusing the RGB and depth “branches” at some point, either by performing concatenation, element-wise summation, or transformation [36, 37, 38, 2].

This thesis will also experiment with the idea of early vs. late fusion to determine what fusion method will, in this use case, yield in a better performance. This is why the RGB network, which is chosen to be implemented, is the SegNet architecture proposed by Badrinarayanan *et al.* [1], and the RGB-D network that is chosen to be implemented is the FuseNet architecture proposed by Hazirbas *et al.* [2]. FuseNet, in this case, very much resembles the SegNet architecture, which is why these two are chosen.

According to Badrinarayanan *et al.* [1], most DL solutions to the image segmentation problem, including the ones mentioned earlier, make use of somewhat similar architectures, namely architectures based on the VGG 16-layer network [7]. This thesis will first discuss the VGG 16-layer network architecture in section 2-3-1. Afterwards, this thesis will discuss the SegNet and the FuseNet architectures in sections 2-3-2 and 2-3-3 respectively. Finally, in section 2-3-4 we will discuss the results of these two networks on the SUNRGBD data-set.

2-3-1 The VGG-16 network

The VGG 16-layer network proposed by [7] is a network that is trained to classify 1000 different objects classes. The VGG 16-layer network was also the runner-up in image classification and the winner in object localisation of the 2014 ILSVRC competition [39]. Since then, a lot of research has been done by others using the VGG 16-layer network as their core CNN architecture. By making use of TL, which in this case consists of using the trained weights of the VGG 16-layer network, researchers could train on different data-sets more easily and in a less time consuming way than training their networks from scratch. Also, by extending this existing pipeline, more complex tasks, such as the image segmentation task, could be solved. For this reason, both networks that are going to be discussed in sections 2-3-2 and 2-3-3 have the VGG 16-layer network as their core architecture. The network consists of thirteen convolutional and pooling layers, and three fully connected layers. The final layer acts as a softmax layer that outputs a probability map of all the 1000 classes on which the network is trained. The output of the network is the class of which the probability of it being contained in the image is the largest. Figure 2-7 shows the detailed architecture of the VGG 16-layer network.

2-3-2 SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation [1]

Badrinarayanan *et al.* [1] propose a semantic segmentation method by means of a decoder network, which they call SegNet. This method differs from the deconvolution network proposed by Noh *et al.* [40]. Also, the terminology used by Badrinarayanan *et al.* [1] differs in that they do not call the first part of their network a convolution network, but they call it an encoder network. Figure 2-9 shows the detailed network architecture of SegNet. If we compare Figures 2-9 and 2-7, we can see that the encoder part of SegNet is nearly identical to the VGG 16-layer network with the exception of the final three fully connected layers. Badrinarayanan *et al.* [1] claim that this approach makes the SegNet network relatively smaller and easier to train than other architectures that keep the fully connected layers, like the convolution network proposed by [40].

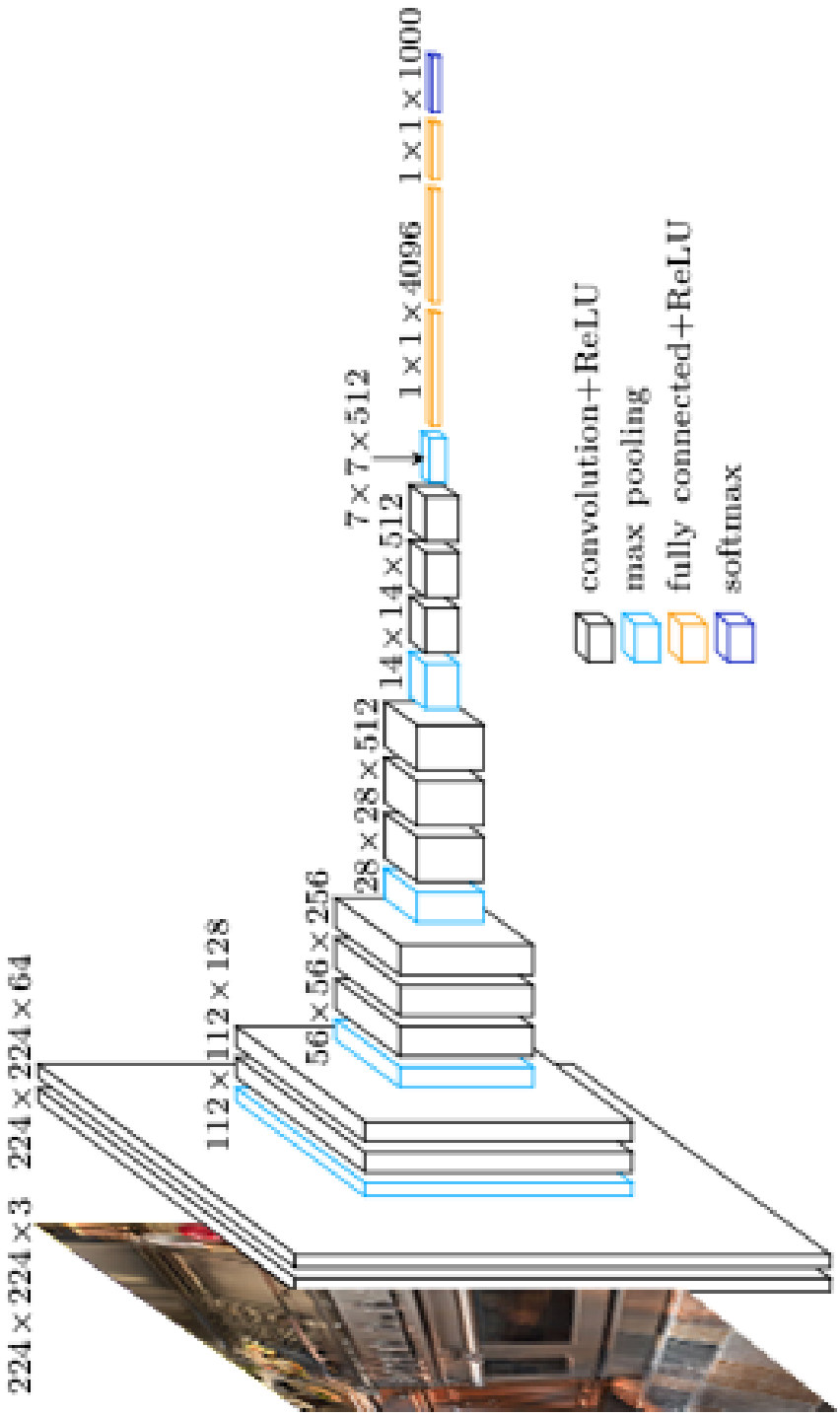


Figure 2-7: The VGG 16-layer network proposed by Simonyan *et al.* [7]. The network consists of six blocks of layers. The network begins with an input layer. The first two blocks consist of two convolutional layers followed by a max pooling operation. The third, fourth, and fifth blocks consist of three convolutional layers followed by a max pooling operation. Each convolutional layer is followed by a ReLU operation. The final block consists of three fully connected layers, which are followed by a softmax operation yielding the output of the network. Source [8]

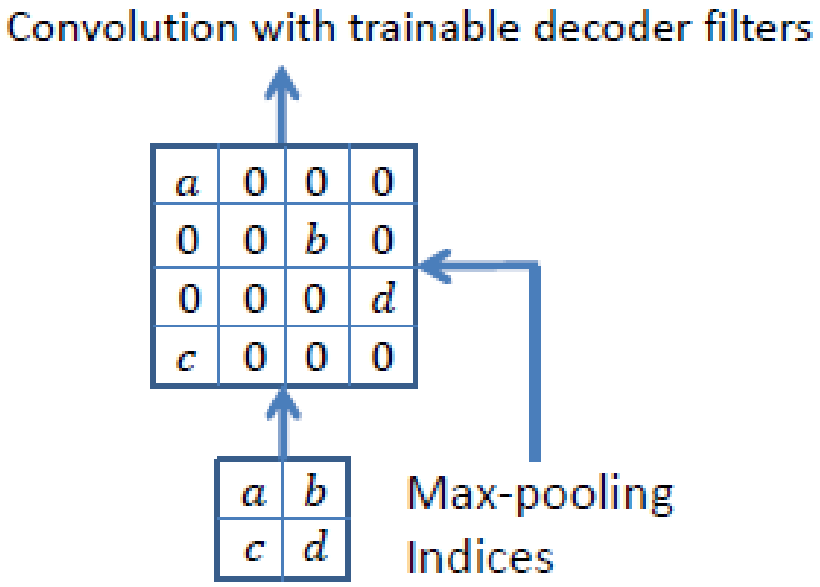


Figure 2-8: An illustration of the SegNet decoder. The labels "a", "b", "c", and "d" correspond to values in a feature map. SegNet uses the max pooling indices to upsample (without learning) the feature map(s) and convolves them with a trainable decoder filter bank [1].

SegNet has an encoder network and a decoder network, followed by a classification layer, i.e. a softmax layer, that performs a pixel-wise classification action. The encoder network consists of the thirteen convolutional layers of the VGG 16-layer network. Badrinarayanan *et al.* [1] chose to leave out the fully connected layers of the VGG 16-layer network in favour of keeping higher resolution feature maps. This reduces the number of parameters of the network, according to Badrinarayanan *et al.* [1]. Each encoder layer has an analogous decoder layer. Hence, their decoder network also has 13 layers. For their decoder network, Badrinarayanan *et al.* [1] propose to store the indices that were pooled from the encoder network to use in the upsampling step of their decoder. Furthermore, Badrinarayanan *et al.* [1] state that the appropriate decoder in the decoder network upsamples its input feature map by using those memorized pooling locations, which produces a sparse feature map. That feature map is then convolved with a trainable decoder filter to produce a dense feature map. See Figure 2-8 for a visual representation provided by Badrinarayanan *et al.* [1]. Note that this method of the upsampling decoder proposed by [1] is very similar to the upsampling, unpooling, and deconvolution method proposed by Noh *et al.* [40] a year earlier. Badrinarayanan *et al.* [1] also acknowledges this. However, they claim that the DeconvNet architecture proposed by [40] has a lot more parameterisation, needs more computational resources and is harder to train end-to-end. According to [1], this is due to the fact that [40] do not disregard the fully convolutional layers of the VGG 16-layer network. The results of SegNet on the SUNRGBD dataset will be evaluated in section 2-3-4.

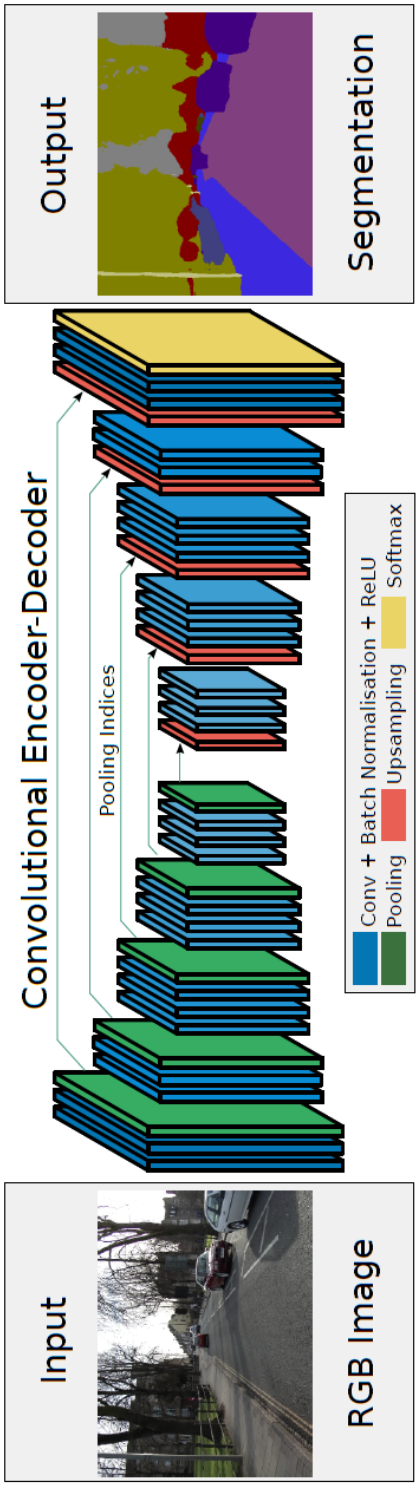
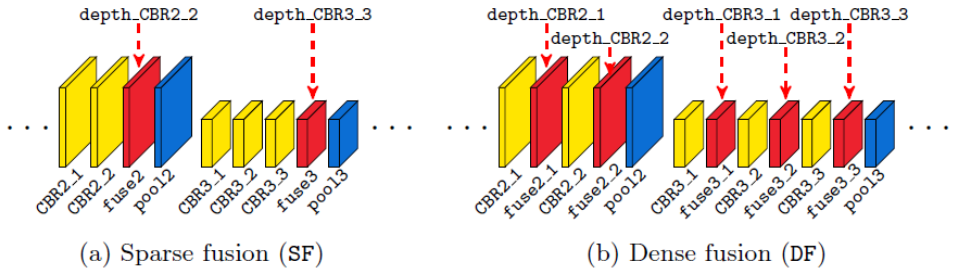


Figure 2-9: The SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification [1].



(a) Sparse fusion (SF) (b) Dense fusion (DF)

Figure 2-10: Illustration of different fusion strategies at the second and third convolution blocks of VGG 16-layer net. (a) The fusion layer is only inserted before each pooling layer. (b) The fusion layer is inserted after each CBR block. [2].

2-3-3 FuseNet: Incorporating Depth into Semantic Segmentation via a Fusion-based CNN Architecture [2]

Hazirbas *et al.* [2] propose an encoder-decoder type network, much like the DeconvNet proposed by [40] and the SegNet proposed by [1], which they call FuseNet. The proposed network by [2] differs from the networks by [40] and [1] in the sense that they consider having two encoder sections, i.e. a RGB encoder branch and a depth encoder branch. Figure 2-11 shows the detailed architecture of FuseNet. When comparing Figures 2-11 and 2-9, one can see that these networks are identical except for two particular aspects. One aspect is the additional depth encoder section. Furthermore, Hazirbas *et al.* [2] also makes use of dropout layers, which they derive from a network called Bayesian SegNet, proposed by [9]. Bayesian SegNet is actually an extension of SegNet because of these dropout layers.

Kendall *et al.* [9] use this dropout layer to induce a Bayesian network, which will in turn, so they claim, boost performance. However, since Hazirbas *et al.* [2] do not use the dropout layers during test time, we will also disregard them for this review. The depth encoder branch proposed by Hazirbas *et al.* [2] is basically the same type of encoder branch as the RGB encoder branch. However, instead of a three-channel input, it expects a one-channel input. Hazirbas *et al.* [2] thus only consider the physical distance between the camera and the objects captured by the camera. Furthermore, Hazirbas *et al.* [2] normalise the depth data to have its values lie between 0 and 255. This results in consistency of the data, since the RGB values of an image also lie between 0 and 255. The RGB encoder branch and the depth encoder branch are "fused" together by fusion layers that perform element-wise summation. In order to enhance the RGB feature maps, making use of this kind of fusion means that the discontinuities of the feature maps computed on the depth image are thus added into the RGB feature maps. Hazirbas *et al.* [2] further propose two different implementations of FuseNet; a dense fusion and a sparse fusion. A dense fusion is the network where there are multiple fusion layers per convolution block, and a sparse fusion is the network with just a single fusion layer convolution per block. These two network architectures are illustrated in Figure 2-10. The results of the FuseNet network on the SUNRGBD dataset will be evaluated in section 2-3-4.

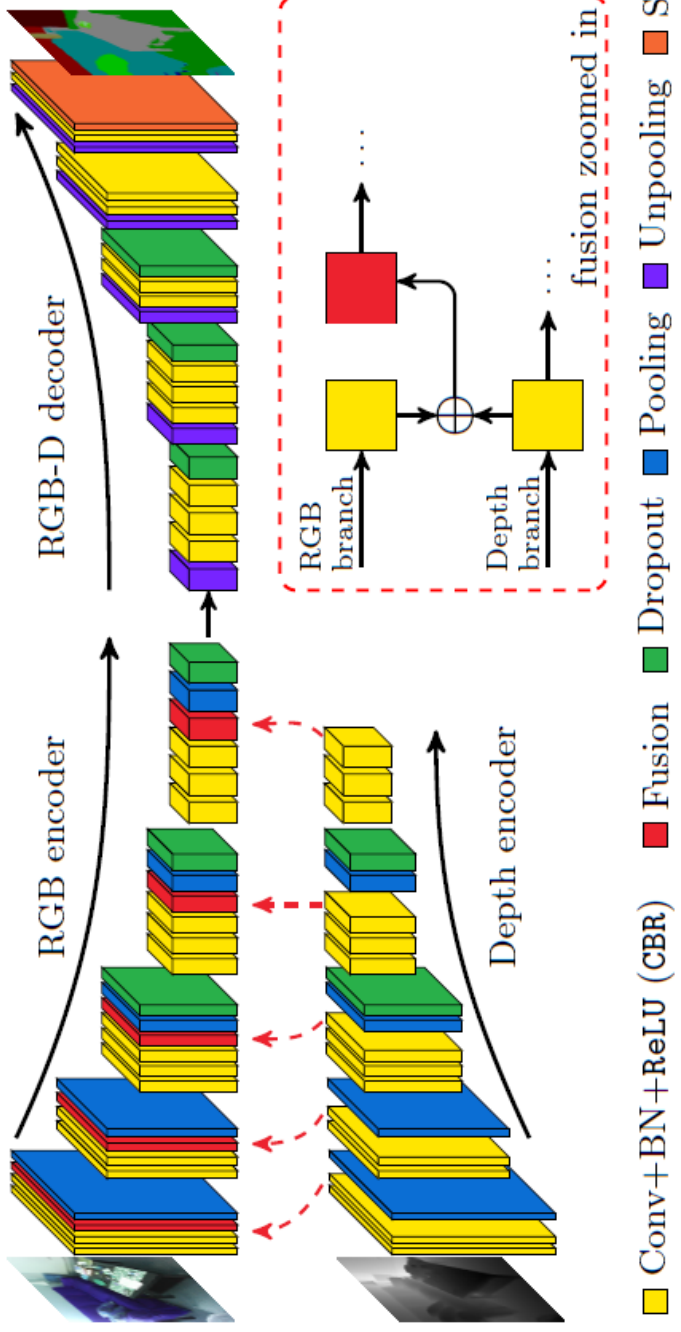


Figure 2-11: The FuseNet architecture proposed by Hazirbas *et al.* [2]. Colors indicate the layer type. The network contains two branches to extract features from RGB and depth images, and the feature maps from depth are constantly fused into the RGB branch, denoted with the red arrows. In this architecture, the fusion layer is implemented as an element-wise summation, demonstrated in the dashed box [2].

	Global	Mean	IoU
SegNet [1]	72.63%	44.76%	31.84%
FuseNet [2]	**76.27%**	**48.30%**	**37.29%**

Table 2-1: The results of the proposed networks in terms of global accuracy, mean class accuracy and the Intersect over Union (IoU) value. We can see that the FuseNet architecture proposed by Hazerbas *et al.* [2] performs better in all three evaluation metrics. An increase of 3.64% is observed in the global accuracy metric, an increase of 3.54% is observed in the mean class accuracy metric, and an increase of 5.45% is observed in the IoU metric.

2-3-4 Discussion

The two network architectures discussed in the previous sections are evaluated on the SUN-RGBD dataset. This data-set has been chosen as the benchmark for this review since the results of some of these networks on the SUNRGBD dataset have been made available by Badrinarayanan *et al.* [1] for its own architecture. The same is true for the FuseNet architecture proposed by Hazirbas *et al.* [2]. The results are grouped by three metrics: the global accuracy, the mean class accuracy, and the intersect over union value. The global accuracy is the percentage of correctly classified pixels, the mean class accuracy is the average of class-wise accuracy, and the Intersect over Union (IoU) value is the average value of the intersection of the prediction and the true regions over the union of them. These evaluation metrics are defined as:

$$Global = \frac{1}{N}\sum_{c} TP_c, \quad c \in 1...K$$

$$Mean = \frac{1}{K}\sum_{c} \frac{TP_c}{TP_c + FP_c}, \quad c \in 1...K$$

$$IoU = \frac{1}{K}\sum_{c} \frac{TP_c}{TP_c + FP_c + FN_c}, \quad c \in 1...K$$

In these formulas, K denotes the total number of classes, c denotes the class number, N denotes the total number of annotated pixels, TP denotes the pixels that are labeled as positive for the class and truly are positive for the class, FP denotes the pixels that are labeled as positive for the class, but actually are not being part of that class, and FN denotes the pixels that are labeled as negative for the class, but actually are part of that class.

The results of the network architectures are presented in Table 2-1. From this table, we can clearly see that the addition of the depth data does indeed improve the performance of the network. The global accuracy improves by 3.64%, the mean class accuracy improves by 3.54%, and the IoU improves by 5.45%. We can also see that the FuseNet architecture proposed by Hazirbas *et al.* [2] performs better in general. It is important to note, however, that the results of the SegNet architecture displayed in Table 2-1 are based on RGB-only data. The architecture has not been trained on RGB-D data and thus it would not (yet) be possible to conclude whether late fusion in this case would perform better than early fusion. This thesis will thus conduct its own experiment in chapter 4.

Chapter 3

Implementation

This thesis will further cover the implementation of the proposed architectures in sections 2-3-2 and 2-3-3 respectively. We will first cover the camera setup used to capture all of the images in section 3-1. Afterwards, in section 3-2, we will cover the structure of the data used to train the networks. Finally, we will explain how the networks were implemented in MATLAB, in sections 3-3 and 3-4 respectively.

3-1 Camera setup

The camera setup used to capture the data can be seen in Figure 3-1. It consists of a dual camera setup, one camera to capture the RGB images, and one camera to capture the depth data. The camera used to capture the RGB images is an RGB camera, made by the iDS-imaging company, that captures the images at a resolution of 1600×1200 pixels and outputs a regular RGB image. The camera used to capture the depth data is an Ensenso stereo camera that captures the data at a resolution of 1280×1024 points. The Ensenso camera does not output a regular image, its output consists of a point cloud containing the spatial X-, Y-, and Z-coordinates of what it is capturing. See Figures 3-2 and 3-3 for an example of an RGB image captured by the iDS camera and a point cloud captured by the Ensenso stereo camera respectively.

Looking at Figure 3-1, we can see how the cameras are set up. The two cameras are located at point one in Figure 3-1 and are pointed downwards to point two in Figure 3-1. The cameras are set up next to each other, with the iDS RGB camera hanging a little lower than the Ensenso stereo camera. Point two in Figure 3-1 denotes the location of the objects.

The way these cameras are set up introduces the issue that the images are not captured within the same reference frame. Also the fact that the two cameras do not capture the images within the same aspect ratio, i.e. 3:4 for the iDS camera and 4:5 for the Ensenso stereo camera, makes that a spatial registration of the data is needed. Luckily, the software of the Ensenso camera already has a function to calibrate multiple camera setups.

Figure 3-1: The camera setup used to capture the data. 1: the dual camera setup, the blue camera being the Ensenso camera for depth images, the black camera being the iDS camera for RGB images. 2: the box with the products inside.

Figure 3-2: An RGB image captured by the iDS camera. We can see the box with products inside. This is a raw image that has not been pre-processed.

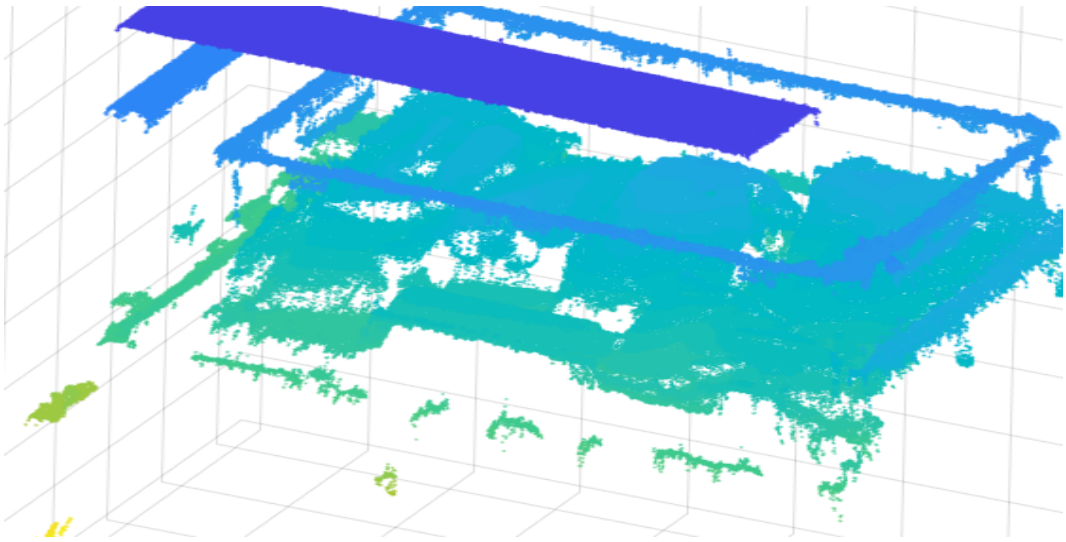


Figure 3-3: A point cloud captured by the Ensenso stereo camera. The dots represent the points captured by the camera. The intensity of the colour tells us more about the physical distance between the camera and the objects. The darker the colour intensity, the closer the object is to the Ensenso stereo camera.

3-2 Vanderlande Industries (VI) RGB-D data

The data that is captured by the camera setup specified in section 3-1 needs to be pre-processed before it can be used in the experiments. Before the pre-processing begins, this thesis will make the following assumptions:

- **The RGB images and the depth data are registered to each other.**
- **The data has been labeled accordingly.**

Furthermore, the code to each pre-processing step can be found in Appendix A. This thesis will first cover the pre-processing of the depth data in section 3-2-1. Afterwards, we will cover the pre-processing of the RGB images and the labels in section 3-2-2. Finally, in section 3-2-3 we will summarize how the data has been structured.

3-2-1 Pre-processing the depth information

As stated in section 3-1, the Ensenso stereo camera outputs point cloud data containing the spatial X-, Y-, and Z-coordinates of the objects it is capturing. Since only the depth information is relevant for this thesis, we disregard the spatial X- and Y-coordinates. When considering only the spatial Z-coordinates of the point cloud data, one can plot it as an image as can be seen in Figure 3-4. It is important to note that the X- and Y-coordinates in Figure 3-4 are not the same as the spatial X- and Y-coordinates captured by the Ensenso stereo camera. Rather, they are pixel-coordinates that correspond to the points the camera has captured. Figure 3-4 is a gray scale image, where the intensity reports the physical distance between the object and the camera. The darker a pixel is, the closer it is to the camera. One observation that can be noticed from Figure 3-4 is that there are some dark spots in the figure where one does not expects to see them. These locations were originally filled with Not-a-Number (NaN) values. These NaN values exist because the Ensenso camera cannot find a depth value for every pixel. However, since NNs are not designed to handle NaN inputs, we fill these NaN positions with alternative values. There are several methods to accomplish this. In this thesis, we choose to fill these NaN positions by means of linear interpolation between the non-NaN values.

The next step to pre-process the depth data is to consider the resolution of the newly obtained depth image. The chosen resolution for this thesis is 480×360 pixels, which corresponds to the same resolution used by Badrinarayanan *et al.* [1] during the training of their proposed SegNet architecture. One can input a higher resolution image in a NN. However, it would make the system computationally expensive, it would increase training time, and it would require hardware that was not available to us. Originally, the resolution of the registered gray scale depth image is at 1600×1200 pixels. Thus, after filling all the NaN positions, we want to reduce the resolution of the gray scale image to the desired resolution of 480×360 pixels. The re-sizing of the image has been done by means of a nearest-neighbours interpolation approach instead of a bi-cubic interpolation approach mostly used in the re-sizing of RGB images. Linear of bi-cubic interpolation approaches tend to adjust the values of the depth image during the re-sizing operation.

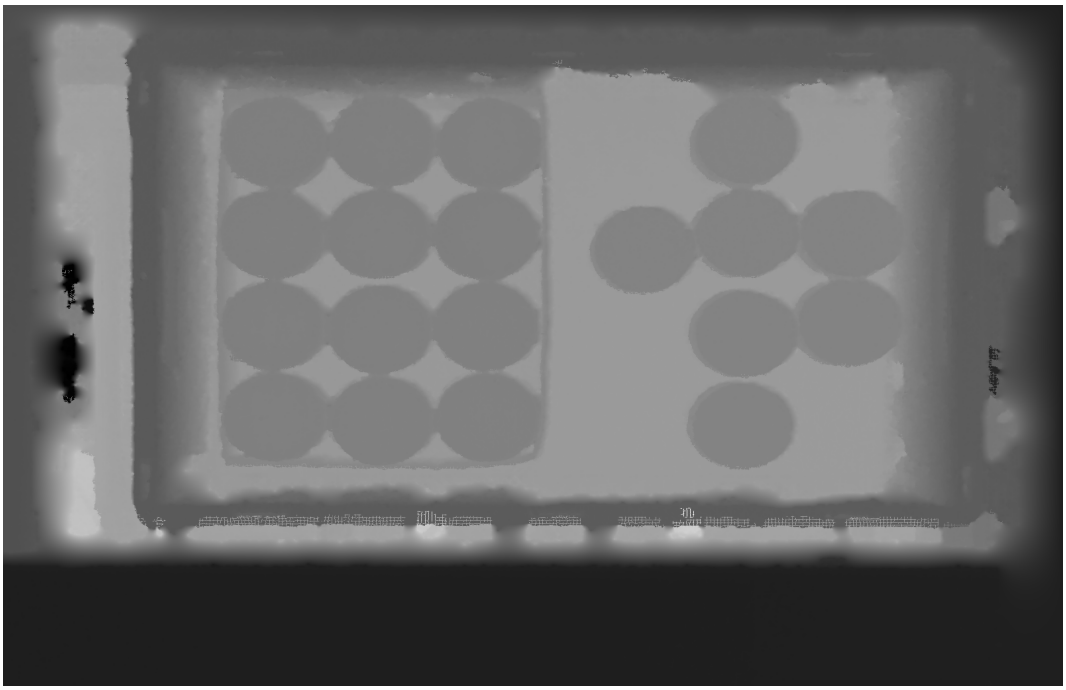


Figure 3-4: The spatial Z-coordinates of a point cloud captured by the Ensenso stereo camera. This is a gray scale image, the darker the pixel, the closer it is to the camera. However, we can see certain dark spots within the image in locations where we do not expect them. These locations are filled with Not a Number (NaN) values.

However, since we only want to re-size while preserving the depth values, the nearest-neighbours interpolation approach is implemented in order to preserve as much information as possible.

Finally, we wish to extract only the most important information of the image. In Figure 3-2 we can see that the image does not only contain the area we wish to segment, which is the bottom of the box, but also a lot of unnecessary information such as the table, the boundaries of the box the objects are in, etc. To make it "easier" for the system, this thesis has chosen to disregard all of the unnecessary areas and focus only on the objects that lie on the bottom of the box. Thus, the depth images have been cropped to a resolution of 335×230 pixels. Note that this operation is not the same as re-sizing the images. Figure 3-5 shows the final result of a depth image after pre-processing.

3-2-2 Pre-processing the RGB images and the labels

The RGB images and their corresponding labels require less pre-processing than the depth information, since the the iDS RGB camera produces a satisfactory RGB image. The RGB images and their corresponding labels just need to be re-sized and cropped in the same manner as we did with the depth information.

The approach mostly used in the re-sizing of RGB images is the bi-cubic interpolation approach and thus we will not deviate from the default. Therefore, the RGB images have been re-sized from a resolution of 1600×1200 pixels to 480×360 pixels by means of a bi-cubic interpolation approach.

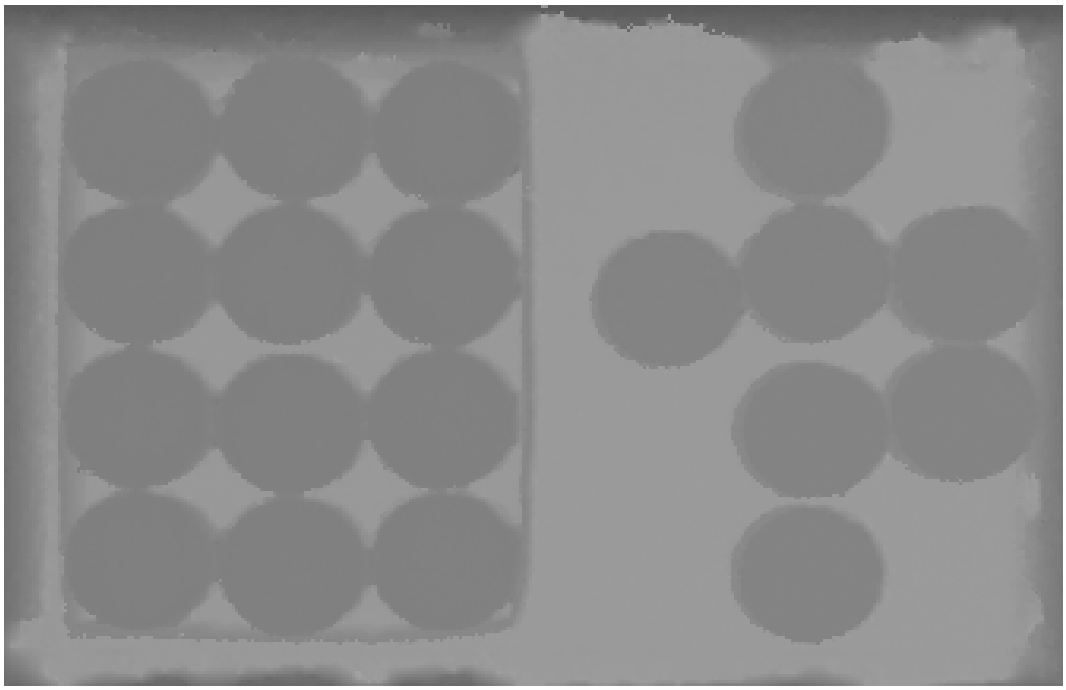


Figure 3-5: After pre-processing: the spatial Z-coordinates of a point cloud captured by the Ensenso stereo camera. This is a gray scale image, the darker the pixel, the closer it is to the camera. This image has been resized to a resolution of 480×360 and has been further cropped to only consider the center of the box.

The corresponding labels, however, have been re-sized from a resolution of 1600×1200 pixels to 480×360 pixels by means of a nearest-neighbours interpolation approach. In contrast to a nearest-neighbours interpolation approach, a bi-cubic interpolation approach can yield to wrong labels in the re-sized image. Therefore, this approach has been chosen for the same reason as for the depth information, namely to preserve as much information as possible.

Both the RGB images and their corresponding labels have been cropped to a resolution of 335×230 pixels in the same manner as we did with the depth information. Figures 3-6 and 3-7 show the final results of an RGB image and its corresponding label after pre-processing.

3-2-3 General structure of the data

Now that we have pre-processed all of the data, we can assign the data to their corresponding classes. The data consists of 790 RGB-D images, which are stored as 790 different matrices of dimensions $[H \times W \times C]$, where H denotes the height of the image, W denotes the width of the image, and C denotes the number of channels. In our case the matrices are of dimensions $[230 \times 335 \times 4]$. The first three channels denote the RGB image and the final channel denotes the gray scale depth image.

The data is split between a set that is used for training and a set that is used for validation with a ratio of 75% and 25% for training and validation respectively.

Figure 3-6: After pre-processing: an RGB image captured by the iDS camera. This image has been resized to a resolution of 480×360 and has been further cropped to only consider the center of the box.

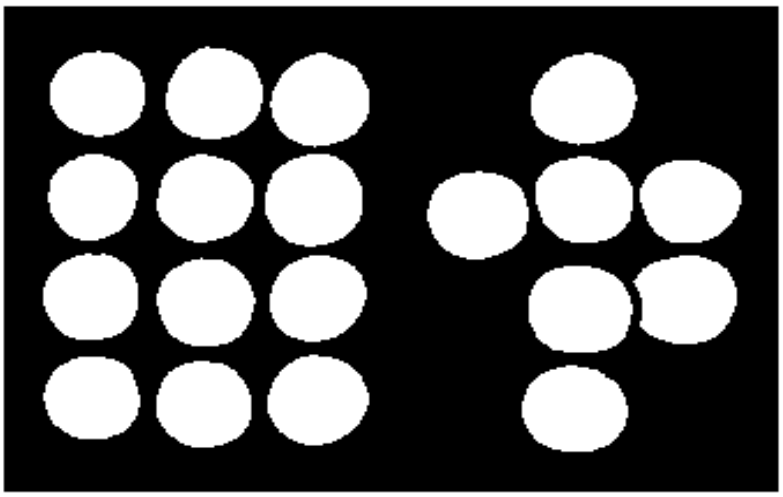


Figure 3-7: After pre-processing: visual representation of a label image. This image has been resized to a resolution of 480×360 and has been further cropped to only consider the center of the box.

The classes, and thus their corresponding labels, represent the primary shape of the objects. In total, the objects have seven different classes. However, some of the classes are grouped together as they can be perceived as similar. The seven classes are:

- **Environment**
- **Cuboid**
- **Planar**
- **Sphere**
- **Cylindrical or flask**
- **Complex shape**
- **Undefined**

The new classes become:

- **Environment, e.g. the bottom of the box.**
- **Cuboid or planar, e.g. a box of cereal or a pack of batteries**
- **Sphere or cylindrical or flask, e.g. a ball, a can of soda, or a bottle of soda.**
- **Complex shape or undefined, e.g. a bag of chips or a stuffed animal.**

These classes have labels 0,1,2, and 3 for the "Environment", the "Cuboid or planar", the "Sphere or cylindrical or flask", and the "Complex shape or undefined" classes respectively. See Figure 3-8 for an example of a labeled image and the data overlayed on each other.

Furthermore, in order to answer one of the problem statements explained in section 1-3, we want to unregister the depth images and the RGB images manually. This way we can assess the difference between registered data and unregistered data. The way this is done is by shifting the depth image along the X-axis. The percentage of shift is explained in more detail in Chapter 4.

Finally, a simple data-augmentation is applied to the images due to our lack of training samples. Data-augmentation is the act of transforming the data in order to increase the number of training samples. It is necessary to apply data-augmentation when one does not have enough data to train a network. We chose to apply a random translation and rotation along the X- and Y-axes and a random reflection along the X-axis. The randomness of the translations, rotations, and the reflection makes each new image unique and ready for use during training.

The data is now ready to be trained on the NN architectures.

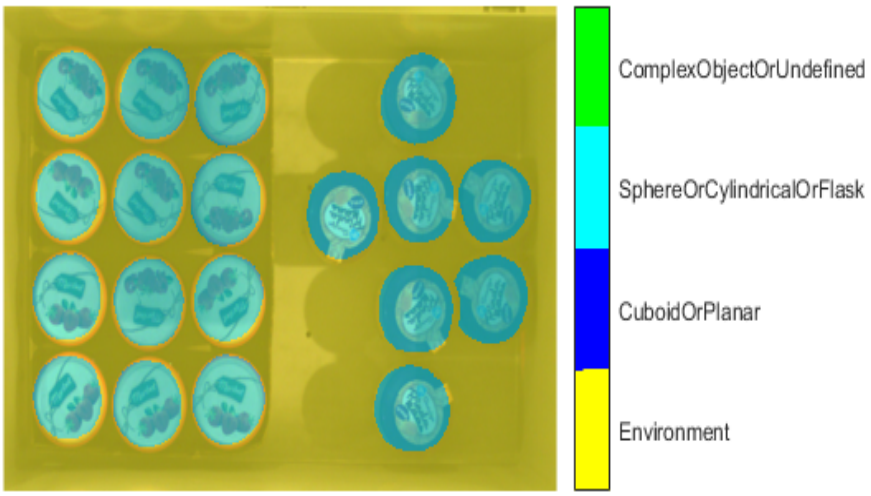


Figure 3-8: A label image overlayed on an RGB image. The colours represent the class. Green stands for the "complex object or undefined" class, light blue stands for the "sphere, cylindrical, or flask" class, dark blue stands for the "cuboid or planar" class, and yellow stands for the "environment" class.

3-3 SegNet RGB-D implementation in MATLAB

As discussed in Chapter 1, the networks are implemented using MATLAB. The code can be found in Appendix B-1. It is relatively straigtforward to implement a SegNet architecture in MATLAB, since MATLAB has a version of SegNet built in its source code. To implement SegNet as proposed by Badrinarayanan *et al.* [1], one could use the

$$segnetLayers(imageSize, numClasses, model)$$

function of MATLAB. This function expects mainly three input variables. The first input variable is the size of the data, or the image size. The second input variable is the number of different classes, or labels. The third input variable is what model the SegNet network should be initialised to. When setting the image size as having three channels, i.e. $[H, W, 3]$, where H denotes the height of the image, W denotes the width of the image, and 3 denotes the Red, Blue, and Green channels respectively. Also, when setting the model as the "$vgg16''$ model, one would obtain the network as proposed by Badrinarayanan *et al.* [1].

However, our focus is on implementing the SegNet architecture with a four channel input based on early fusion. This means that the input layer and the first convolutional layer of the SegNet network have to change. The input layer as it originally is expects the data to be of shape $[H \times W \times 3]$. In our new input layer, the expected structure of the data would be of shape $[H \times W \times 4]$, the fourth channel denoting the depth data.

As for the first convolutional layer, the parameters that have to be changed are: the expected channel size and the weights matrix. The current expected channel size is set at three, this is because of the previous input layer channelling a three-channel input to the convolutional layer. Since the new input layer channels a four-channel input to the convolutional layer, this value should be set at four. Furthermore, the current weights matrix is structured as a $[3 \times 3 \times 3 \times 64]$ matrix. One should read this structure as 64 different 3×3 kernels that are applied to all of the three channels. Since we have an additional fourth channel, the new weights matrix should be structured as a $[3 \times 3 \times 4 \times 64]$ matrix. In other words, 64 different 3×3 kernels that are applied to all of the four channels. We have filled the weights matrices appropriately. As for all of the kernels that are going to be applied to the first three channels, doing nothing to those values would be the best choice as to not lose the information learned by the pre-trained VGG-16 layer network. In other words, for those channels we would like to retain their Transfer Learning (TL) property. As for the fourth channel kernels, many values can be filled in for those weights. However, since we wished to retain TL for the first three channels, we have chosen the same for the fourth channel as well. This is done by averaging the weights of the first three kernels. Suppose matrices W_1^1, W_2^1, and W_3^1 are the first out of 64 3×3 kernels that are applied to the first three channels. Taking the average of these three weight matrices yields in the average of the pre-trained VGG-16 weights for the first 3x3 kernels. The idea is that this way, one could end up with 64 different 3×3 kernels that can be applied to the depth data, while retaining the TL property.

This network could, however, be prone to over-fitting. This is because of the fact that the data that is available to our experiments is scarce, while there are a lot of trainable parameters within this network. This network architecture already contains Batch Normalization and ReLU operations after each convolutional layer to tackle the over-fitting problem. Our concern is that perhaps this would not be enough due to the size of the implemented network compared to the relatively small data-set we possess. That is why for this thesis, we have chosen to add dropout layers to the architecture. According to Srivastava *et al.* [41], dropout operations offer an easy way to prevent over-fitting. The idea of dropout operations is to randomly "drop", or disregard, certain neurons and their connections during training [41]. This will in turn restrict neurons to accustom too much to each other. The authors of Badrinarayanan *et al.* [1] have also made this choice in a different paper proposed by Kendall *et al.* [9], in the network called Bayesian SegNet. This network applies dropout layers after the third, fourth, and fifth max pooling operations and before the first, second, and third up-sampling operations. This network architecture can be seen in Figure 3-9. In this thesis, we have chosen to follow this architecture of dropout layer placement.

3-4 FuseNet implementation in MATLAB

As can be seen in Figure 2-11, the FuseNet architecture proposed by Hazirbar *et al.* [2] has two encoder branches and a single decoder branch. The first encoder branch handles the RGB images and the second encoder branch handles the depth data. The code can be found in Appendix B-2. The first aspect of the implementation of this network within MATLAB is how to handle the input. The RGB encoder branch expects a $[H \times W \times 3]$ input, where H denotes the height of the RGB image, W denotes the width of the RGB image, and 3 denotes the Red, Blue, and Green channels of the image.

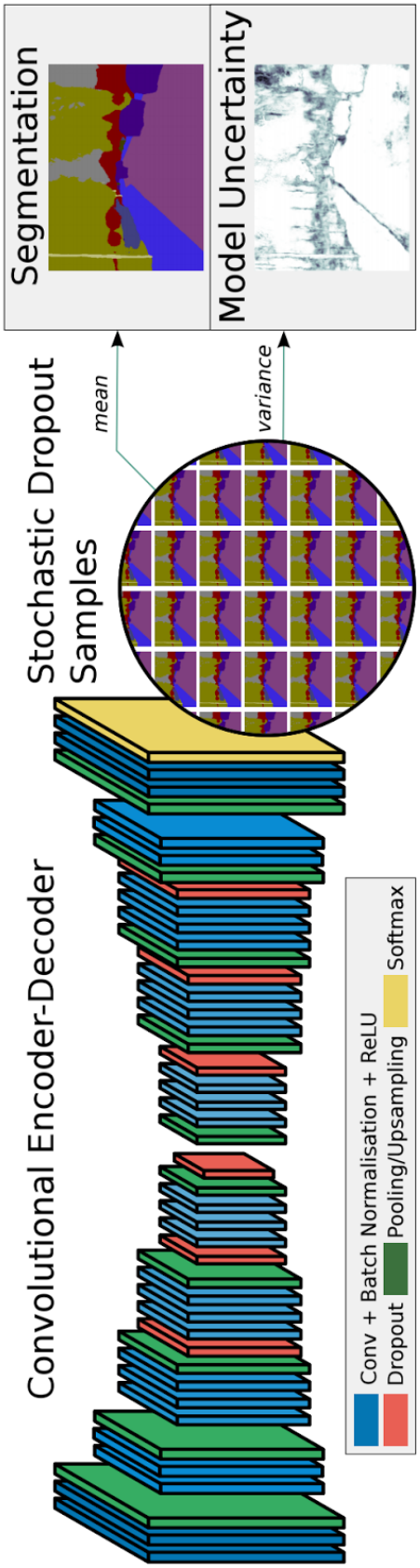


Figure 3-9: The Bayesian SegNet architecture [9]. This figure shows the entire network architecture. The network encoder is based on the VGG-16 layer network proposed by Simonyan *et al.* [7], with the decoder placing them in reverse order. The probabilistic output is obtained from Monte Carlo samples of the model with dropout at test time. Kendall *et al.* [9] take the variance of these softmax samples as the model uncertainty for each class. This thesis will not concentrate on the probabilistic output of this model, we are only interested in the network architecture.

Whereas the depth encoder branch expects a $[H \times W \times 1]$ input, where H denotes the height of the depth image, W denotes the width of the depth image, and 1 denotes the depth channel. Unfortunately, one cannot have two different input layers within MATLAB. Therefore, another method to handle the input data had to be found in order to implement this network in MATLAB. One such method is to have a single input layer, where the input is of shape $[H \times W \times 4]$, where H denotes the height of the RGB-D data, W denotes the width of the RGB-D Data, and 4 denotes the Red, Blue, Green, and depth channels of the data. This input is directed to two different convolutional layers, where the weights of both of these layers are frozen. "Frozen" in this sense means that the weights cannot change when training the network. The parameters of the first convolutional layer are chosen as: three different 1×1 kernels with a stride of one, the number of expected channels is four, and the weights matrix is defined as

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

The output of this convolutional layer would be the first three channels of the input layer, which in this case is the entire RGB image.

The parameters of the second convolutional layer are chosen as: one 1×1 kernels with a stride of one, the number of expected channels is four, and the weights matrix is defined as

$$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}.$$

The output of this convolutional layer would be the final channel of the input layer, which in this case is the entire depth image.

The depth encoder branch of the FuseNet architecture differs from the encoder branch proposed by Kendall *et al.* [9] in two locations. The first convolutional layer of the depth encoder branch differs and the depth encoder branch does not have a max-pooling and dropout operations after the last convolutional layers. The first convolutional layer of the depth encoder branch expects a one-channel input. This thesis will make use of the same arguments and methodology as discussed in section 3-3 to produce this convolutional layer. However, instead of a four-channel convolutional layer proposed in section 3-3, an averaging operation is done and only these weights are used for the layer. In other words, the first convolutional layer of the depth encoder branch would be the same as the first convolutional layer as proposed in section 3-3, where the first three channels are disregarded.

The RGB encoder branch of the FuseNet architecture differs from the encoder branch proposed by Kendall *et al.* [9] in five locations. As discussed in section 2-3-3, the branches are "fused" together by means of element-wise summation of the feature maps. To implement this in MATLAB, before each max-pooling operation an addition layer is introduced. The inputs to these addition layers are the final ReLU layers before each max-pooling operation in both branches. The output of these addition layers is the element-wise summation of both inputs, which is directed to the max-pooling operations of the RGB encoder branch.

As for the decoder branch, this branch does not differ from the decoder branch proposed by Kendall *et al.* [9]. Therefore, the implementation of this branch is the same as discussed in section 3-3.

Chapter 4

Experiments

We now define the different experiments that were conducted within this thesis project. The experiments conducted can be divided into two groups. The first group of experiments is the training of the RGB-D data where the depth data and the RGB images are registered to each other. The second group of experiments is the training of the RGB-D data where the depth data and the RGB images are not registered to each other. With the first group of experiments, this thesis will try to tackle the first problem statement as stated in section 1-3, where we would like to know what the benefits are of using RGB-D as compared to using RGB-only data. With the second group of experiments, this thesis will try to tackle the second problem statement as stated in section 1-3, where we would like to know how much deviation in the registration of the depth data and the RGB images will still yield a result better than using RGB-only data. This chapter will discuss the first group of experiments in section 4-1, and the second group of experiments in section 4-2.

4-1 Experiments of registered RGB-D data

This thesis has conducted a total of four experiments with regard to the first problem statement as stated in section 1-3. These experiments are straight-forward and are conducted by training the two different networks on the available data. We will evaluate the results in chapter 5 of this thesis.

Experiment one: Training SegNet RGB-D network using RGB-only data In this first experiment, we want to train the SegNet RGB-D network implemented in section 3-3 using RGB-only data. As specified in section 3-3, this network expects a four-channel input. Since RGB-only data contains three channels, we have chosen to add a grey-scale image of the data as its fourth channel. A grey-scale image contains the same information as an RGB image, only averaged to one channel. This way, we do not "give" the network more information, we only "give" it the same information twice. See Figure 4-1 for an example of a grey-scale image. In other words, we create a "fake" depth image to feed to the network.

Figure 4-1: A grey-scale image of an RGB image captured by the iDS RGB camera after pre-processing. This will create a "fake" depth channel to feed to the network.

Experiment two: Training SegNet RGB-D network using registered RGB-D data In this second experiment, we want to train the SegNet RGB-D network implemented in section 3-3 using registered RGB-D data. This experiment is straight-forward in the sense that nothing should be done additionally to conduct this experiment.

Experiment three: Training FuseNet RGB-D network using RGB-only data In this third experiment, we want to train the FuseNet RGB-D network implemented in section 3-4 using RGB-only data. As specified in section 3-4, this network expects a four-channel input. The same method as the first experiment is applied, i.e. the fourth channel of the data contains an averaged grey-scale image of the data.

Experiment four: Training FuseNet RGB-D network using RGB-D data In this fourth experiment, we want to train the FuseNet RGB-D network implemented in section 3-4 using registered RGB-D data. This experiment is straight-forward in the sense that nothing should be done additionally to conduct this experiment.

4-2 Experiments of unregistered RGB-D data

This thesis has conducted a total of two experiments with regard to the second problem statement as stated in section 1-3. These experiments are also straight-forward and are conducted by training only the FuseNet network on the available data. In Chapter 5, it will become clear why we intentionally left out the SegNet architecture for this experiment. We will evaluate the results in Chapter 5 of this thesis.

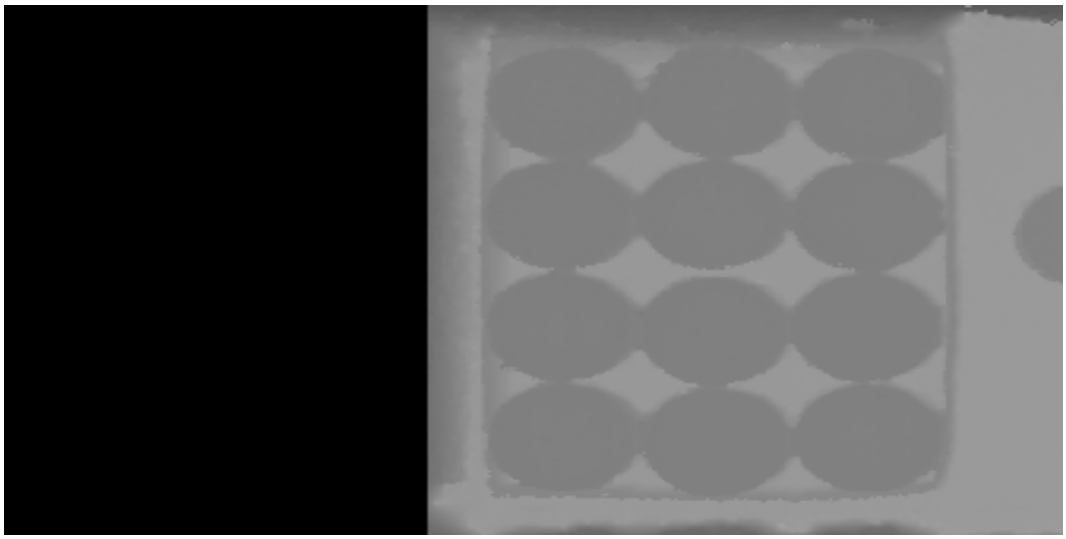


Figure 4-2: Visual representation of a depth image that has been shifted by 40% along the X-axis, using MATLAB, without replacing the zero-positions.

Experiment one: Shifting the depth image along the X-axis from 0% and 100% with a step size of 10% In this first experiment, we want to train the FuseNet RGB-D network implemented in section 3-4 using a variety of unregistered depth data. This thesis has chosen to first look at the big picture by shifting the depth data along the X-axis from 0% shift to 100% shift with a step size of 10%. The shift 1is done using the MATLAB function

$$imtranslate.$$

This function will translate the image with a specified value. The main issue with using this MATLAB function is that the values in the shifted locations are replaced by zero. An unregistered image tends to look similar to a registered image. However, as can be seen in Figure Figure 4-2, this operation does not simulate a true unregistered depth image. Therefore, these zero-locations are replaced by the mean of the maximum values of the depth data. The MATLAB command of this value would be

$$mean(max(Depth\_Image)).$$

Figure 4-3 shows that this operation yields in a better simulation of an unregistered image. Remember that the higher the value of a pixel within the depth image, the higher the physical distance between the object and the camera. Taking the mean of the maximum values of the depth data would thus correspond to replacing the zero-locations by the average value of the bottom of the box the objects are located in. Figure 4-2 shows an example of a depth image that is shifted 40% along the X-axis without replacing the zero-positions, and Figure 4-3 shows an example of a depth image that is shifted 40% along the X-axis with the zero-positions replaced by the mean of the maximum values of the depth image.

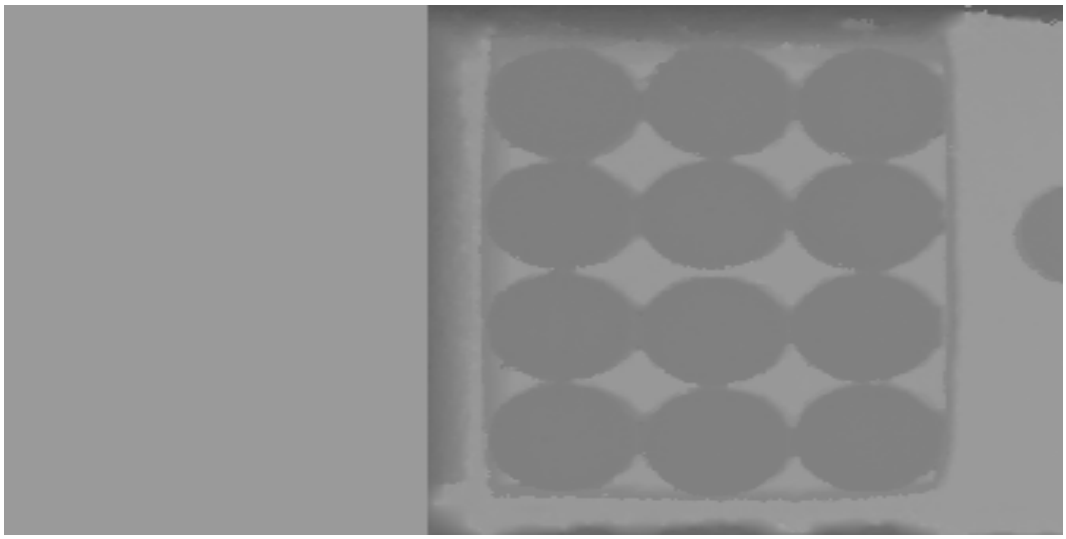


Figure 4-3: Visual representation of a depth image that has been shifted by 40% along the X-axis, using MATLAB, with the zero-positions replaced by the mean of the maximum values of the depth image.

Experiment two: Shifting the depth image along the X-axis from 0% and 10% by systematically reducing the shift along the X-axis In this second experiment, we want to train the FuseNet RGB-D network implemented in section 3-4 using a different variety of unregistered depth data. This time, we will focus on the maximum allowed shift along the X-axis where the performance would still be better than using RGB-only data. We chose to evaluate the network by systematically decreasing the shift along the X-axis, between 0% and 10% shift, until we found the maximum allowed deviation.

Chapter 5

Results and Discussion

This chapter will evaluate all of the results obtained from the different experiments as stated in Chapter 4. This chapter will first discuss the evaluation metrics used to evaluate the results in section 5-1. The results of the first group of experiments conducted in section 4-1 concerning the first problem statement as stated in section 1-3 will be evaluated and discussed in section 5-2. Finally, the results of the second group of experiments conducted in section 4-2 concerning the second problem statement as stated in section 1-3 will be evaluated and discussed in section 5-3.

5-1 Evaluation metrics

This thesis has chosen three evaluation metrics to evaluate the results of the different experiments. Before we can specify these evaluation metrics, we want to look at a general structure of a confusion matrix. A confusion matrix is a matrix that specifies all of the correctly classified and incorrectly classified values, its general structure looks like:

$$\left[\begin{array}{lll|lll} True & Positives & (TP) & False & Positives & (FP) \\ \hline False & Negatives & (FN) & True & Negatives & (TN) \end{array}\right]$$

The matrix can be read as: *TP* denotes all of the pixels that are classified to a class, say cuboid, and that truly belong to that class. *FP* denotes all of the pixels that are classified to a class, say cuboid, but that actually belong to a different class, say cylinder. *FN* denotes all of the pixels that are classified to a different class, say cylinder, but that actually belong to the real class, say cuboid. *TN* denotes all of the pixels that are classified to a different class, say cylinder, and that actually belong to that class.

Using the values extracted from the confusion matrix, we can now look at the three different evaluation metrics.

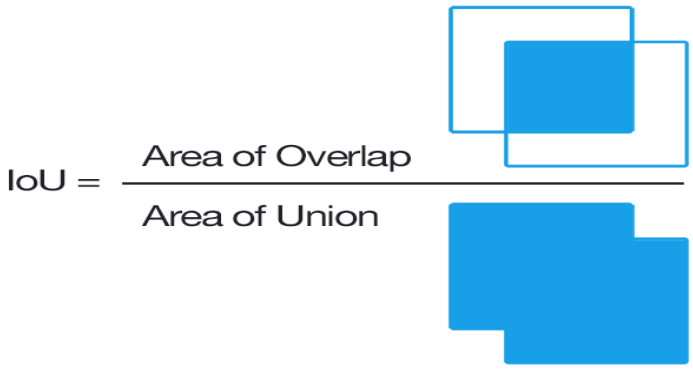


Figure 5-1: The Intersect over Union (IoU) metric can be described as the area of overlap between the true label and the predicted label divided by the total area of union of the two labels. Source: [10]

All of the evaluation metrics used in this thesis can be derived from this confusion matrix. The first evaluation metric used in this thesis is the global accuracy. It can be calculated from the confusion matrix as

$$Global \quad accuracy = \frac{1}{N}\sum_{c} TP_c, \quad c \in 1...K, \tag{5-1}$$

where N denotes the total number of pixels within an entire data-set, K denotes the total number of classes, c denotes the class number, and TP denotes the True Positive values.

The second evaluation metric used in this thesis is the mean class accuracy. It can be calculated from the confusion matrix as

$$Mean \quad class \quad accuracy = \frac{1}{K}\sum_{c} \frac{TP_c}{TP_c + FN_c}, \quad c \in 1...K, \tag{5-2}$$

where K denotes the total number of classes, c denotes the class number, TP denotes the True Positive values, and FN denotes the False Negative values. Note that this formula is the same as calculating the recall.

The final evaluation metric used in this thesis is the mean Intersect over Union (mIoU). It can be calculated from the confusion matrix as

$$mIoU = \frac{1}{K}\sum_{c} \frac{TP_c}{TP_c + FP_c + FN_c}, \quad c \in 1...K, \tag{5-3}$$

where K denotes the total number of classes, c denotes the class number, TP denotes the True Positive values, FP denotes the False Positive values, and FN denotes the False Negative values. Of these three evaluation metrics, the IoU metric can be visualised as can be seen in Figure 5-1. The IoU metric can also be calculated as the total area of overlap between the true label and the predicted label divided by the total area of union of the two labels.

	Global accuracy	Mean class accuracy	mIoU
SegNet RGB + "fake" depth	73.99%	73.57%	48.69%
SegNet RGB-D	76.00%	73.63%	50.50%
FuseNet RGB + "fake" depth	85.54%	80.75%	64.13%
FuseNet RGB-D	**87.04%**	**83.96%**	**66.56%**

Table 5-1: The results of four different experiments in terms of global accuracy, mean class accuracy and the intersect over union value. We can see that the FuseNet architecture trained on RGB-D data performs best in all three evaluation metrics. We can also see that, in both RGB-only and RGB-D cases, the FuseNet architecture performs better than the SegNet architecture in all three evaluation metrics. This strengthens the notion that a late fusion method performs better than an early fusion method. Finally, we can see that using RGB-D data in both the SegNet and FuseNet architectures, the performance increases in all three evaluation metrics.

5-2 Results of the experiments of registered RGB-D data

As stated in section 3-2-3, the available data has been split in 75% training data and 25% validation data. The split is done such that the data has been divided randomly, this way we avoid most biases the network can learn. The results of the four experiments can be found in Table 5-1.

From these values we can see that when we used the real RGB-D data in both the SegNet and the FuseNet architectures, we achieve better results than using RGB-only data (Fusenet: global accuracy of 87.04% when training with RGB-D data as compared to 85.54% when training with RGB-only data, mean class accuracy of 83.96% when training with RGB-D data as compared to 80.75% when training with RGB-only data, and mIoU of 66.56% when training with RGB-D data as compared to 64.13% when training with RGB-only data. SegNet: global accuracy of 76.00% when training with RGB-D data as compared to 73.99% when training with RGB-only data, mean class accuracy of 73.63% when training with RGB-D data as compared to 73.57% when training with RGB-only data, and mIoU of 50.50% when training with RGB-D data as compared to 48.69% when training with RGB-only data). Although the performance of the networks is only a couple of percent better in all three evaluation metrics, the improvement seems consistent. Just as Hazirbas *et al.* [2] has shown, the addition of depth data does seem to bring improvement to the performance of the network (Fusenet: global accuracy increase of 1.5% when training with RGB-D data, mean class accuracy increase of 3.21% when training with RGB-D data, and mIoU increase of 2.43% when training with RGB-D data. SegNet: global accuracy increase of 2.01% when training with RGB-D data, mean class accuracy increase of 0.06% when training with RGB-D data, and mIoU increase of 1.81% when training with RGB-D data).

Also, when comparing the result of both SegNet experiments to both FuseNet RGB-D experiments, we can see that the FuseNet architecture performs considerably better in all evaluation metrics (RGB + "fake" depth: global accuracy increase of 11.55% when training with the FuseNet architecture, mean class accuracy increase of 7.18% when training with the FuseNet architecture, and mIoU increase of 15.44% when training with the FuseNet architecture. RGB-D: global accuracy increase of 11.04% when training with the FuseNet architecture, mean class accuracy increase of 10.33% when training with the FuseNet architecture, and mIoU increase of 16.06% when training with the FuseNet architecture).

This seems to indicate that late fusion could be a wiser choice than early fusion. This result has shown us that the intuition of [36, 37, 38, 2] was correct in the sense that late fusion can be a better way to incorporate the depth data to the network architecture.

However, another way that we can evaluate the results of these networks is by means of a visual approach. Here, we will show a couple of examples of images that were fed to these differently trained networks and discuss what we see goes right or wrong. In image segmentation there is generally a problem that is caused by the fact that there is no knowledge of the contents of an image a priori. This means that the systems cannot determine beforehand how many segments are required in a particular image. The problem gets divided into two cases: "under-segmentation", which happens when parts of the image that actually belong to different objects, or to an object and the background, are assigned to the same segment; and "over-segmentation", which occurs when parts of the image belonging to a single object are split apart by the algorithm [42].

Figures 5-2, 5-3, 5-4, and 5-5 show the results of one particular image when fed through the four different networks. Each figure is split into the ground truth image (left) and the predicted image (right). Also, each label is colour-coded into four different colours. Yellow corresponds to the "Environment" class, dark blue corresponds to the "Cuboid or Planar" class, light blue corresponds to the "Sphere or Cylinder or Flask" class, and green corresponds to the "Complex Object or Undefined" class. Figures 5-2, 5-3, 5-4, and 5-5 demonstrate that under-segmentation occurs heavily in all cases. However, we can also see that the FuseNet architecture (Figures 5-4 and 5-5) has much less under-segmentation than the SegNet architecture (Figures 5-2, and 5-3) in both data-sets.

When comparing the SegNet architecture trained on RGB-only data and trained on RGB-D data (when comparing Figures 5-2 and 5-3), we can see that in the case of RGB-D data (Figure 5-3) the network suffers from over-segmentation on the bottom right corner. However, the under-segmentation is still reduced. This means that using RGB-D data can help the network to perceive the objects somewhat better. In the case of FuseNet trained on RGB-D data compared to the FuseNet trained on RGB-only data (Figure 5-4 compared to Figure 5-5), we can see that only the under-segmentation has been reduced. This also helps in our statement that using RGB-D data can improve the network. Also, we would like to point out that the manner of under-segmentation in the SegNet networks, as can be seen by Figures 5-2 and 5-3, can probably be explained due to the early fusion method.

Looking at a different example, Figures 5-6, 5-7, 5-8, and 5-9 show an image, containing a single object, that has been fed to the networks. This example shows the difference between the SegNet and the FuseNet architectures at its core. We can see from Figures 5-6 and 5-7 that the SegNet architecture suffers highly from under-segmentation, even when adding depth information. We can also see from Figures 5-8 and 5-9 that the FuseNet architecture suffers less from under-segmentation. That is why the experiments conducted in section 5-3 were only conducted by training the FuseNet architecture. This result further supports the notion that a late fusion method might perform better than an early fusion method.

As a final example, Figures 5-10 and 5-11 show yet another image that has been fed to the networks. However, this example highlights a difference between using RGB-only images and RGB-D data while training the FuseNet architecture. Figure 5-10 shows that the FuseNet network trained on RGB-only images slightly suffers from over-segmentation in some cases.

	Global accuracy	Mean class accuracy	mIoU
FuseNet RGB-D 0% shift	***87.04%***	***83.96%***	***66.56%***
FuseNet RGB-D 10% shift	85.05%	79.41%	62.76%
FuseNet RGB-D 20% shift	85.19%	77.12%	61.85%
FuseNet RGB-D 30% shift	83.22%	72.49%	56.34%
FuseNet RGB-D 40% shift	84.32%	78.84%	61.55%
FuseNet RGB-D 50% shift	84.70%	79.06%	61.01%
FuseNet RGB-D 60% shift	81.73%	71.31%	55.66%
FuseNet RGB-D 70% shift	85.13%	74.67%	60.08%
FuseNet RGB-D 80% shift	83.35%	78.34%	58.36%
FuseNet RGB-D 90% shift	83.45%	72.37%	58.06%
FuseNet RGB-D 100% shift	84.99%	76.57%	61.47%
FuseNet RGB + "fake" depth	85.54%	80.75%	64.13%

Table 5-2: The results of the FuseNet RGB-D network trained on the different shifts of the depth data along the X-axis. The shifts occur from 0% to 100% shift with a step-size of 10%. The results are evaluated in terms of global accuracy, mean class accuracy and the intersect over union value. This table shows that only a 0% shift along the X-axis performs better than using RGB-only data in the subsequent evaluation metrics. This seems to indicate that the maximum shift we are looking for lies between 0% and 10%.

Figure 5-11, on the other hand, shows that the FuseNet network trained on RGB-D data does not suffer from over-segmentation in this case. This does not mean that the FuseNet network trained on RGB-D data does not suffer from over-segmentation. This example is just to show that in some cases, the use of RGB-D data can improve the network in contrast to RGB-only images.

5-3 Results of the experiments of unregistered RGB-D data

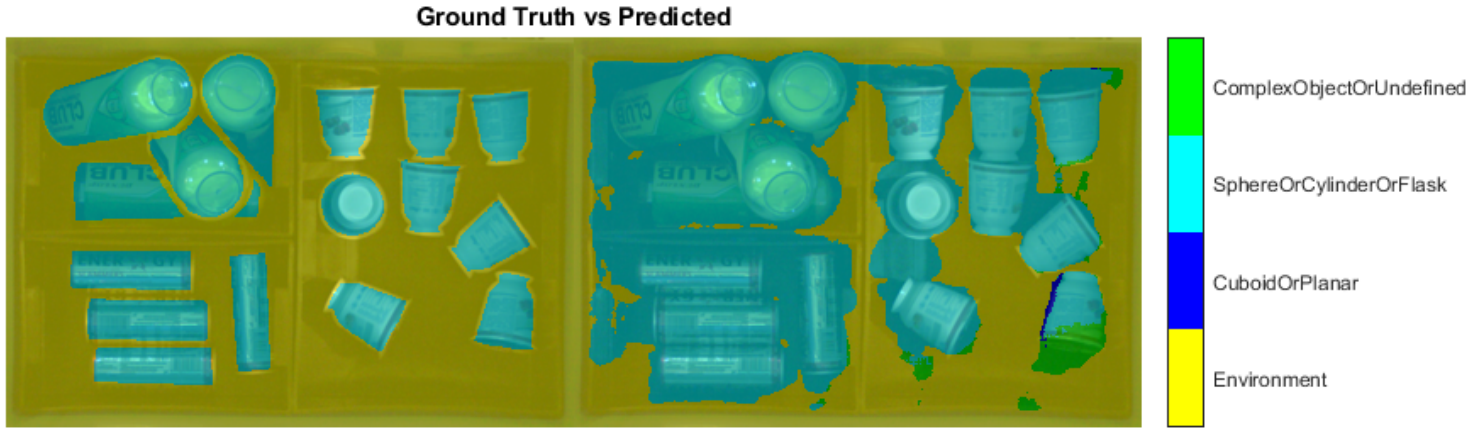
We have seen in section 5-2 that the FuseNet architecture performs significantly better in all three performance metrics than the SegNet architecture. For this reason, as stated in section 4-2, we will only evaluate the FuseNet network for the following experiments. Also, as stated in section 4-2, the experiment is done in two phases. The first phase is shifting the depth image along the X-axis from 0% to 100% shift with a step-size of 10%. The result of this phase can be found in Table 5-2. As we can see from Table 5-2, the FuseNet RGB-D network trained with 0% shift performs best of all of the experiments. However, we also notice that none of the networks trained on shifted depth data performs better than the RGB-only case. Figures 5-12, 5-13, and 5-14 visualise these results. Looking at these figures, while none of the networks trained on shifted data performs better than the RGB-only case, there is a large fluctuation between the results that differs from shift-to-shift. We do not have a clear explanation for this observation. However, when looking at a curve fit, we can see that there is still a downward trend in all three of the evaluation metrics the larger the shift along the X-axis gets.

Now that we have seen that, in general, there is a downward trend the larger the shift along the X-axis gets, we can zoom in between 0% and 10% shift to truly determine the maximum allowed deviation in the un-registration of the depth data.

(a) Ground truth labels vs. the predicted labels overlayed on the original RGB image

(b) Original RGB image

Figure 5-2: SegNet trained on RGB-only data. We can see that the network does not perform as desired and under-segmentation occurs heavily.



(a) Ground truth labels vs. the predicted labels overlayed on the original RGB image

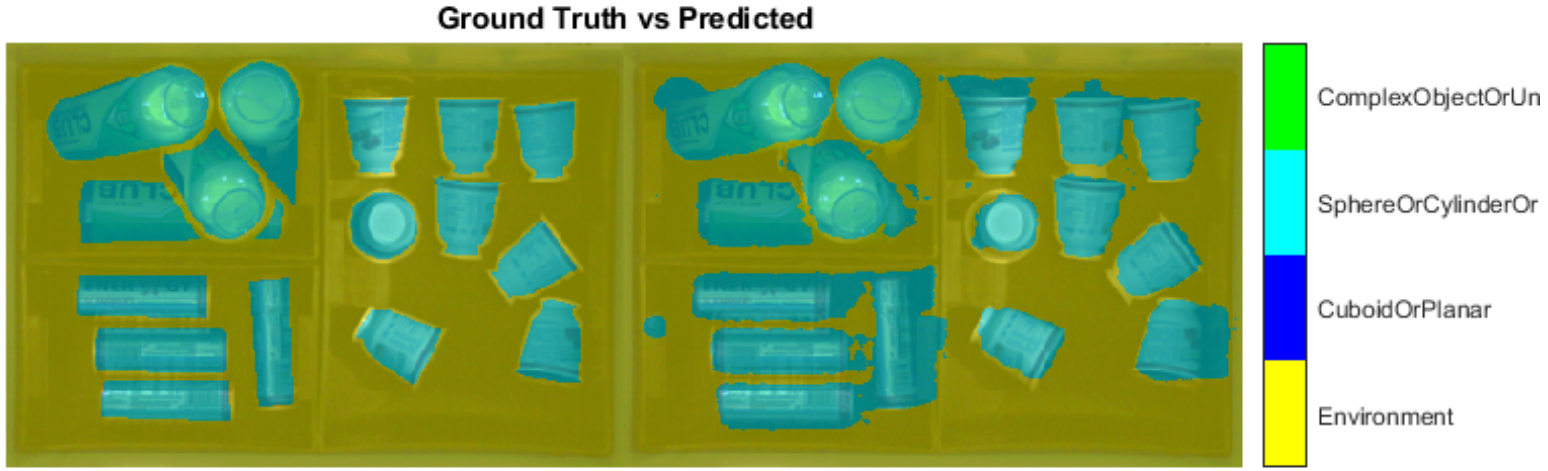
(b) Original RGB image

Figure 5-3: SegNet network trained on RGB-D data. We can see that the network does not perform as desired and under-segmentation occurs heavily. We can also see that the under-segmentation that occurs in this case is less severe than the RBD-only case. However, we can also see a little over-segmentation occurring in the bottom right corner.

(a) Ground truth labels vs. the predicted labels overlayed on the original RGB image

(b) Original RGB image

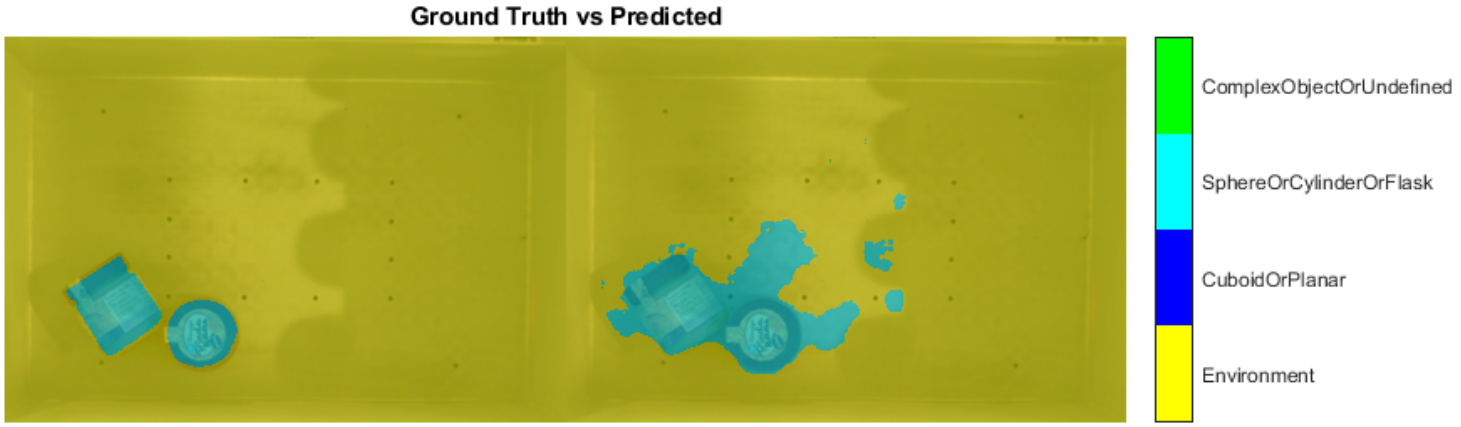
Figure 5-4: FuseNet network trained on RGB-only data. We can see that the network performs almost as desired, with less under-segmentation occurring when comparing it to the SegNet RGB-only, and SegNet RGB-D networks. Also, we can see that there is no over-segmentation as in the case of the SegNet RGB-D network.



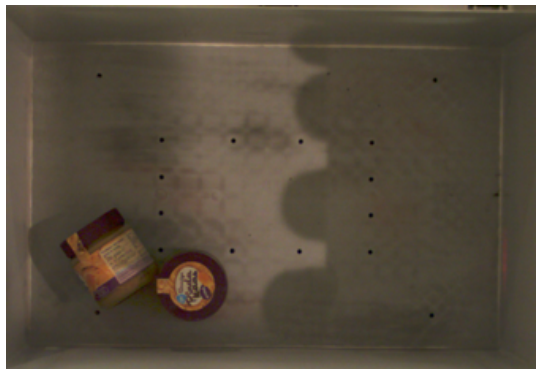
(a) Ground truth labels vs. the predicted labels overlayed on the original RGB image

(b) Original RGB image

Figure 5-5: FuseNet network trained on RGB-D data. We can see that the network performs almost as desired, with less under-segmentation occurring when comparing it to the SegNet RGB-only, SegNet RGB-D. Under-segmentation is further reduced compared to the FuseNet RGB-only networks. Also, we can see that there is no over-segmentation as in the case of the SegNet RGB-D network.



(a) Ground truth labels vs. the predicted labels overlayed on the original RGB image



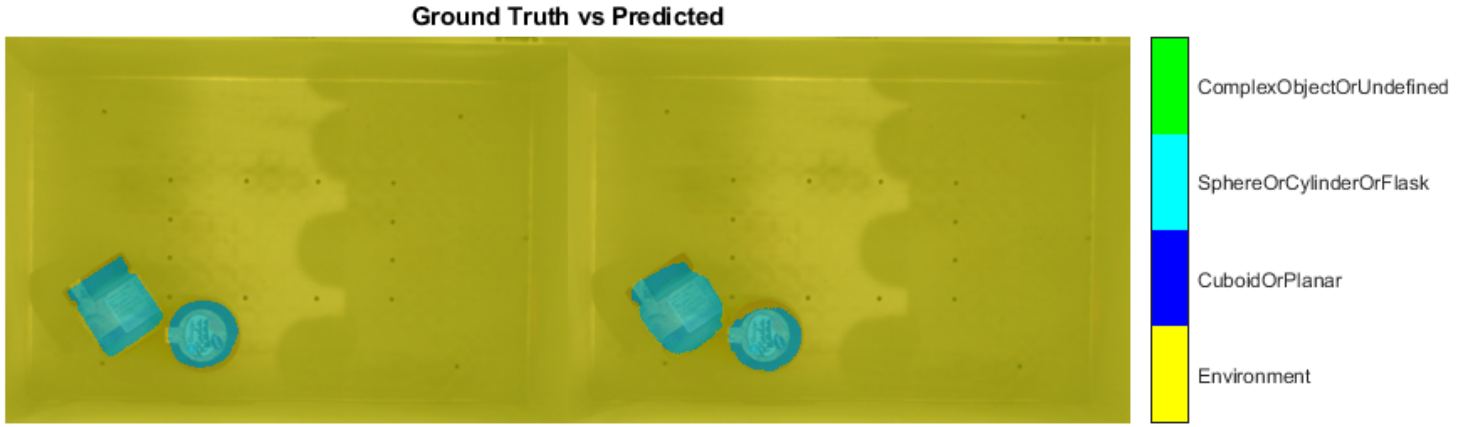
(b) Original RGB image

Figure 5-6: SegNet network trained on RGB-only data. We can see that the network does not perform as desired and under-segmentation occurs heavily.

(a) Ground truth labels vs. the predicted labels overlayed on the original RGB image

(b) Original RGB image

Figure 5-7: SegNet network trained on RGB-D data. We can see that the network does not perform as desired and under-segmentation occurs heavily. We can also see that the under-segmentation that occurs in this case is less severe than the RGB-only case.



(a) Ground truth labels vs. the predicted labels overlayed on the original RGB image

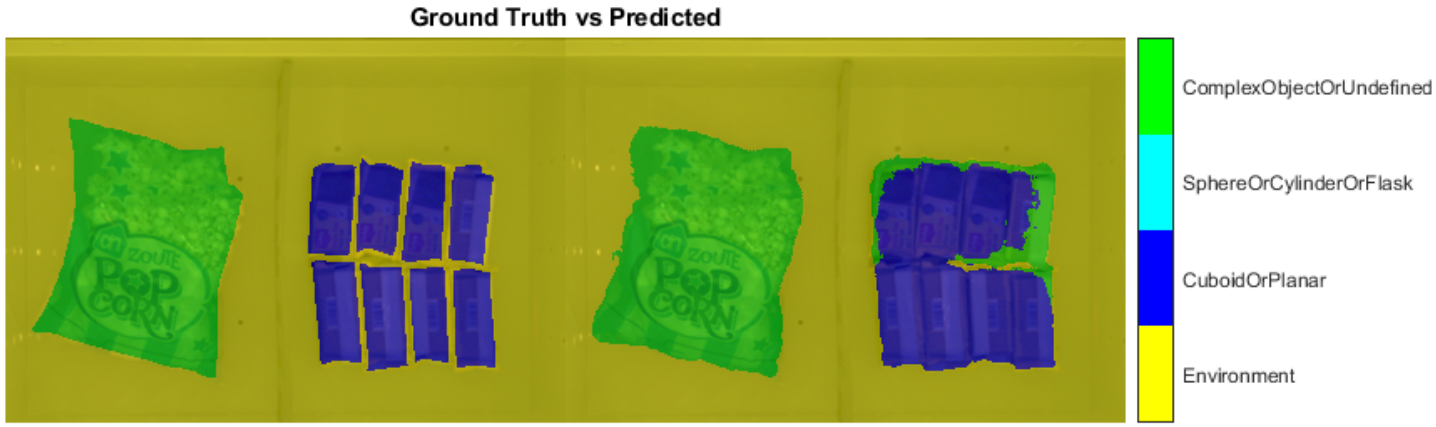
(b) Original RGB image

Figure 5-8: FuseNet network trained on RGB-only data. We can see that the network performs almost as desired, with less under-segmentation occurring when comparing it to the SegNet RGB-only, and SegNet RGB-D networks.

(a) Ground truth labels vs. the predicted labels overlayed on the original RGB image

(b) Original RGB image

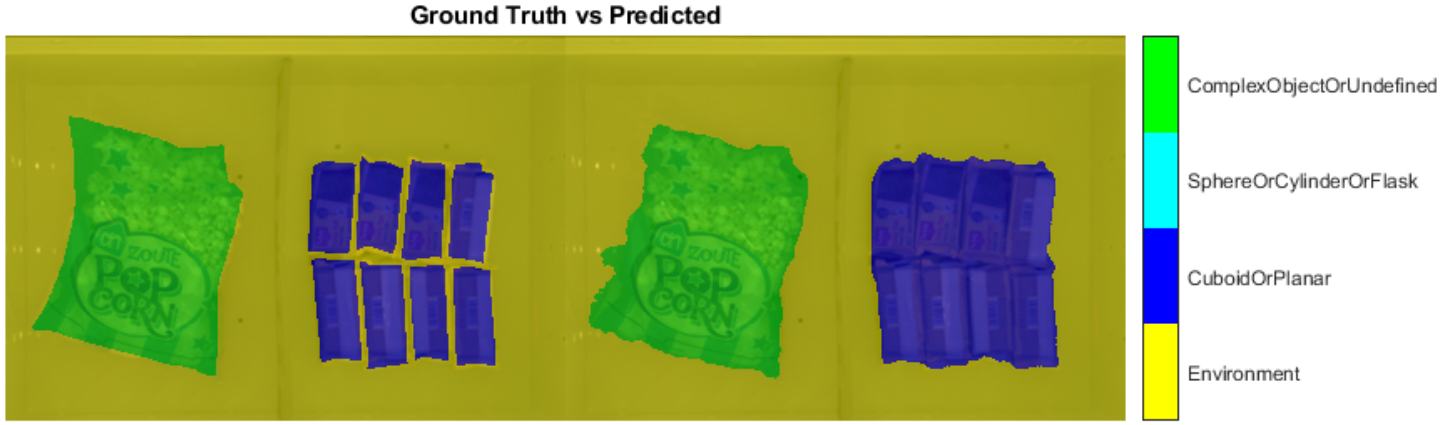
Figure 5-9: FuseNet network trained on RGB-D data. We can see that the network performs almost as desired, with less under-segmentation occurring when comparing it to the SegNet RGB-only, SegNet RGB-D, and the FuseNet RGB-only networks.



(a) Ground truth labels vs. the predicted labels overlayed on the original RGB image

(b) Original RGB image

Figure 5-10: FuseNet network trained on RGB-only data. We can see that the network performs almost as desired. However, over-segmentation does occur in some areas.



(a) Ground truth labels vs. the predicted labels overlayed on the original RGB image

(b) Original RGB image

Figure 5-11: FuseNet network trained on RGB-D data. We can see that the network performs almost as desired, with less under-segmentation occurring when comparing it to the and the FuseNet RGB-only network. Also, we can see that there is no over-segmentation as in the case of the FuseNet RGB-only network.

	Global accuracy	Mean class accuracy	mIoU
FuseNet RGB-D 0% shift	***87.04%***	***83.96%***	***66.56%***
FuseNet RGB-D 0.5% shift	**85.95%**	**82.85%**	**64.81%**
FuseNet RGB-D 1% shift	85.39%	83.12%	64.19%
FuseNet RGB-D 2% shift	85.14%	78.78%	63.38%
FuseNet RGB-D 5% shift	84.16%	74.76%	58.90%
FuseNet RGB-D 10% shift	85.05%	79.41%	62.76%
FuseNet RGB + "fake" depth	85.54%	80.75%	64.13%

Table 5-3: The results of the FuseNet RGB-D network trained on the different shifts of the depth data along the X-axis. The shifts occur from 0% to 10% shift by means of systematically reducing the shift along the X-axis. This thesis has chosen to evaluate the shifts 0%, 0.5%, 1%, 2%, 5%, and 10%. The results are evaluated in terms of global accuracy, mean class accuracy and the intersect over union value. We can see that a shift of 0% along the X-axis performs best in all three evaluation metrics. However, a shift of 0.5% performs better than using RGB-only data in all three evaluation metrics.

The method used in this second phase of the experiment to determine this maximum allowed deviation is to systematically reduce the shift along the X-axis until we found the maximum allowed deviation. The results of this second phase can be found in Table 5-3. Also, Figures 5-15, 5-15, and 5-15 visualise the results of Table 5-3. This thesis has chosen to first look at a 5% shift along the X-axis. From Table 5-3 and the Figures 5-15, 5-16, and 5-17 we can see that this is too much shift. Afterwards, we looked at 2% shift along the X-axis. Again, from Table 5-3 and the Figures 5-15, 5-16, and 5-17 we see that is still too much shift. We chose to than look at a shift of 1% along the X-axis. Looking at Table 5-3 and the Figures 5-15, 5-16, and 5-17, we can see that we are getting closer. The final shift this thesis has looked at is a shift of 0.5% along the X-axis.

From Table 5-3 and Figures 5-15, 5-16, and 5-17 we can say that the maximum allowed shift along the X-axis of the depth data would be 0.5%. This value corresponds to a shift of one pixel of the depth image. When scaling back the image to its original size it would correspond to a shift of three pixels along the X-axis. As stated in section 3-1, we have the spatial X-coordinates of the objects available.

Thus, we can translate these three pixels to be correspondent to 1.67 millimetres of shift along the X-axis. In other words, the maximum allowed shift along the X-axis where the results are still better than the RGB-only case would be 1.67 millimetres.

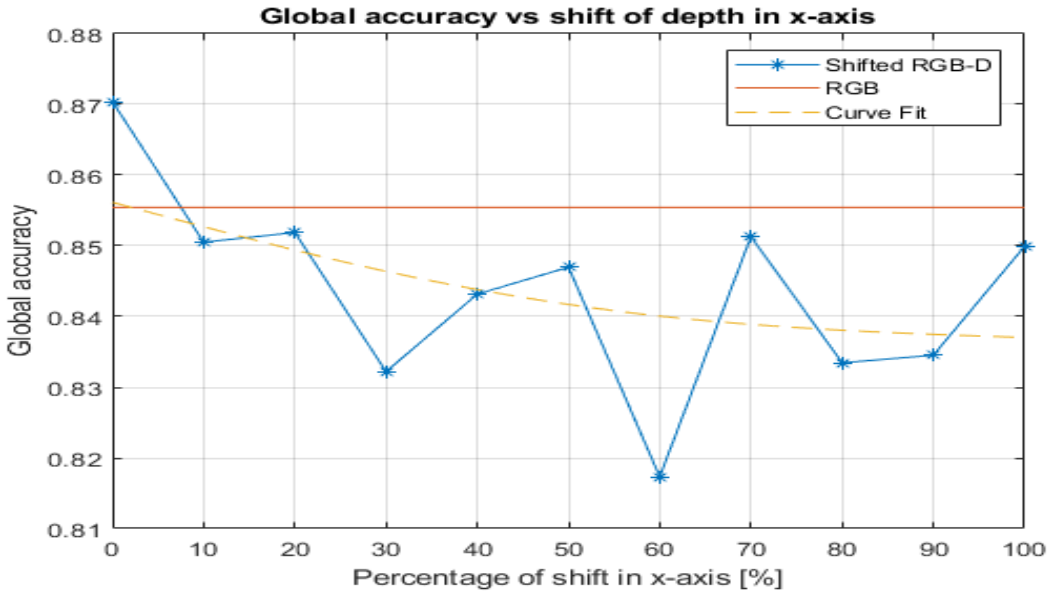


Figure 5-12: Visual representation of the global accuracy's of the different shifts of the depth data along the X-axis. These shifts occur from 0% to 100% shift along the X-axis with a step-size of 10%. We can see that only a shift of 0% along the X-axis performs better that using RGB-only data. Also, the results fluctuate in a manner that it may seem that a bigger shift can result in a better performance. However, we can also see from the curve fit that there is a downward trend in the performance.

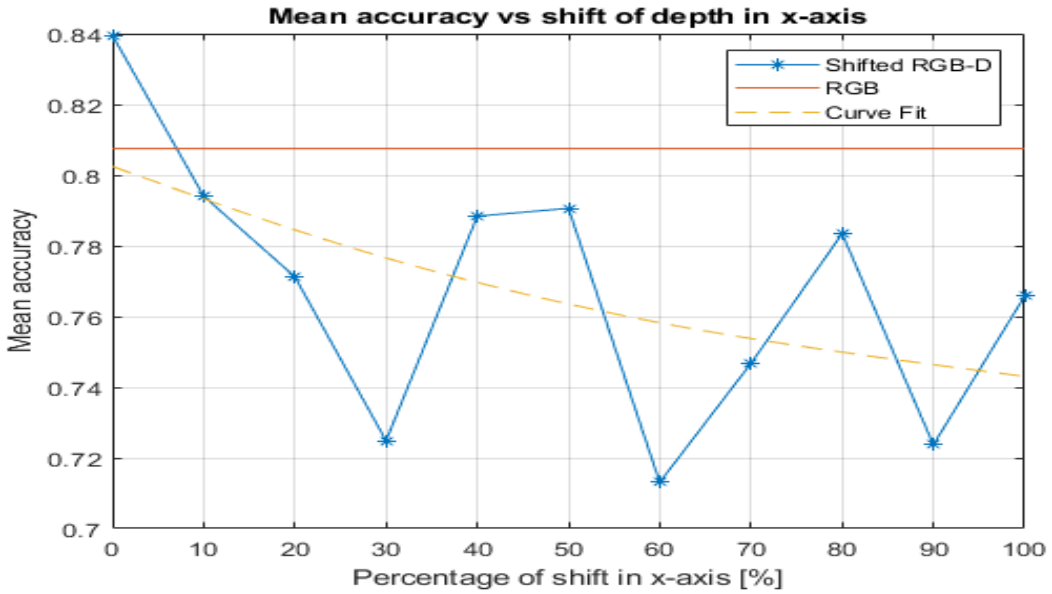


Figure 5-13: Visual representation of the mean class accuracy's of the different shifts of the depth data along the X-axis. These shifts occur from 0% to 100% shift along the X-axis with a step-size of 10%. We can see that only a shift of 0% along the X-axis performs better that using RGB-only data. Also, the results fluctuate in a manner that it may seem that a bigger shift can result in a better performance. However, we can also see from the curve fit that there is a downward trend in the performance.

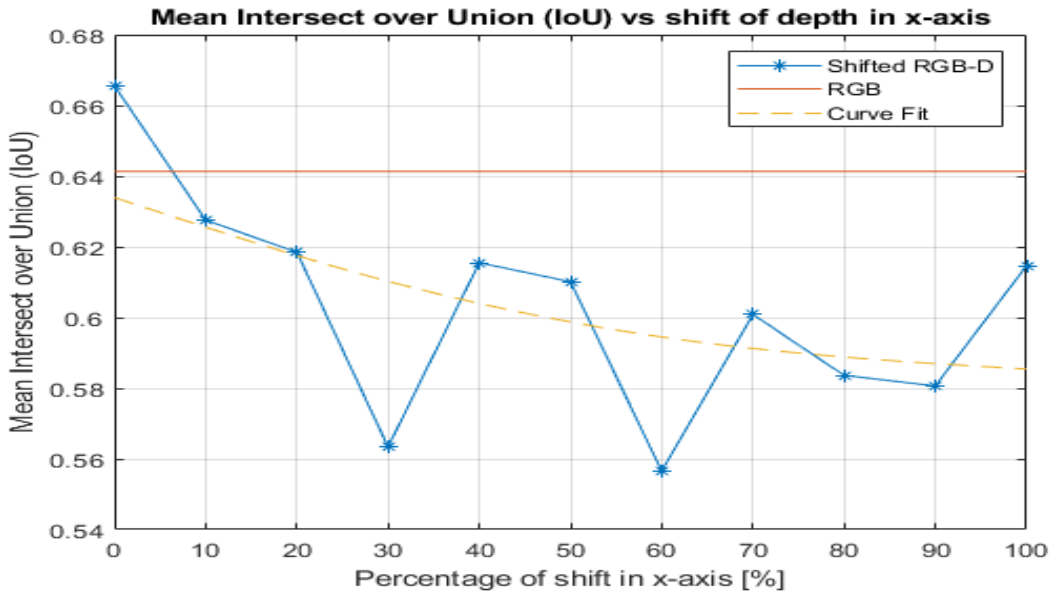


Figure 5-14: Visual representation of the mean Intersect over Union (mIoU) values of the different shifts of the depth data along the X-axis. These shifts occur from 0% to 100% shift along the X-axis with a step-size of 10%. We can see that only a shift of 0% along the X-axis performs better than using RGB-only data. Also, the results fluctuate in a manner that it may seem that a bigger shift can result in a better performance. However, we can also see from the curve fit that there is a downward trend in the performance.

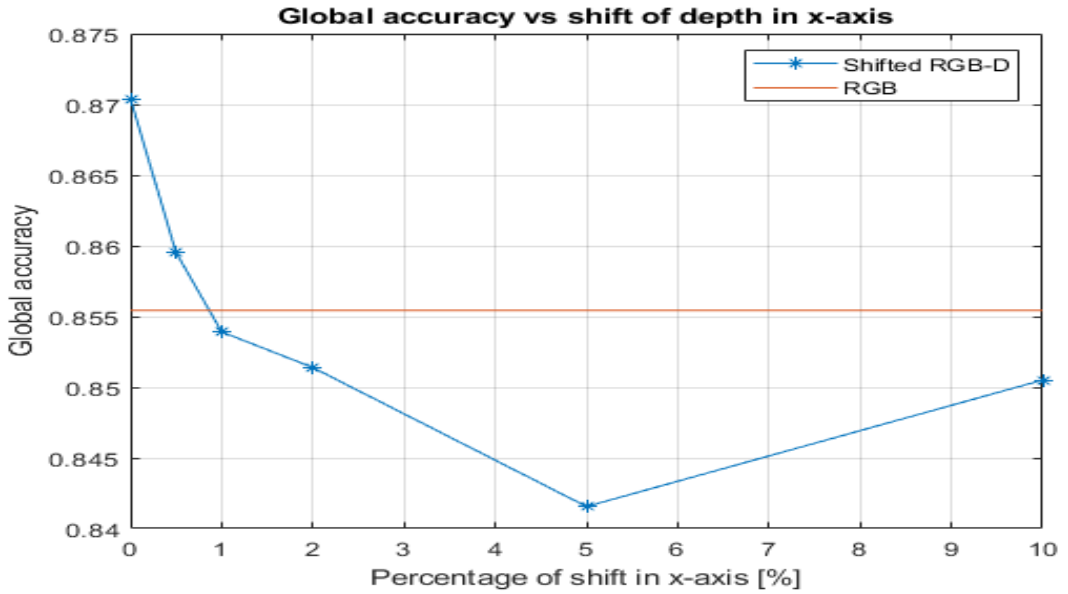


Figure 5-15: Visual representation of the global accuracy's of the different shifts of the depth data along the X-axis. These shifts occur from 0% to 10% shift along the X-axis. We can see that shifts of 0% and 0.5% perform better than using RGB-only data.

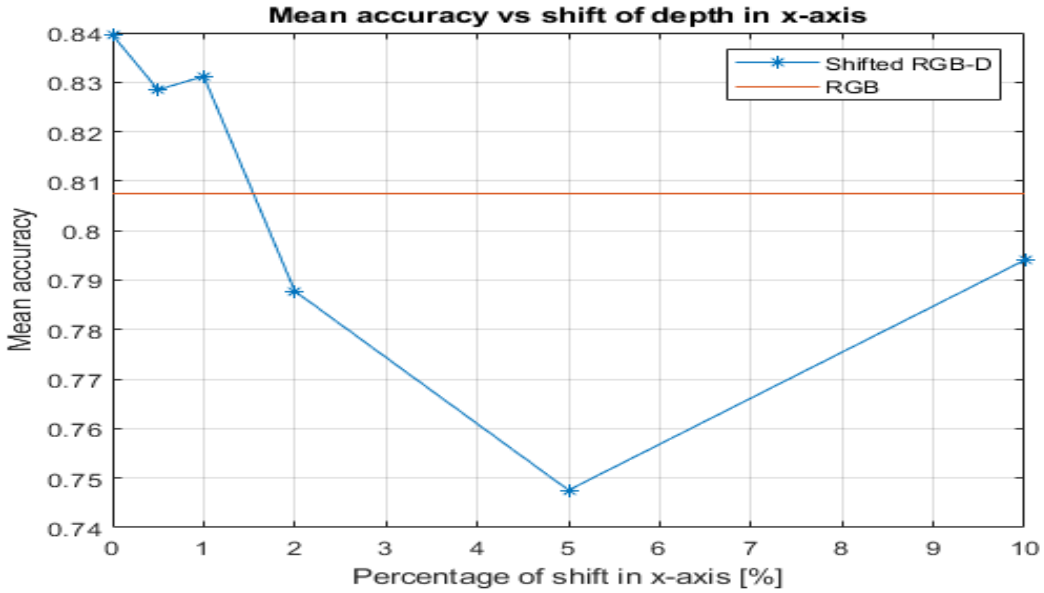


Figure 5-16: Visual representation of the mean class accuracy's of the different shifts of the depth data along the X-axis. These shifts occur from 0% to 10% shift along the X-axis. We can see that shifts of 0%, 0.5%, and 1% perform better than using RGB-only data.

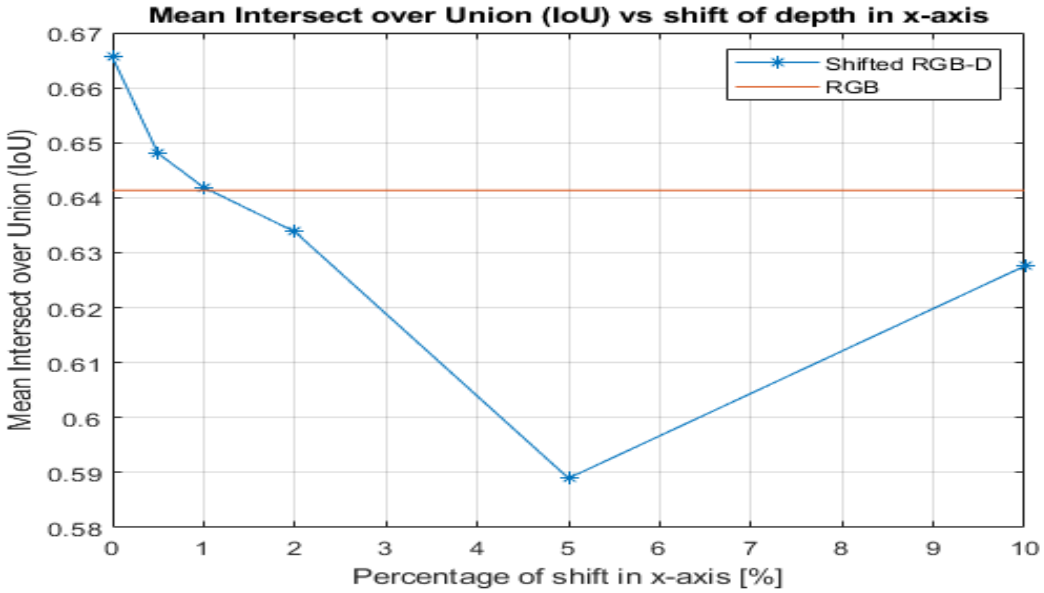


Figure 5-17: Visual representation of the mean Intersect over Union (mIoU) values of the different shifts of the depth data along the X-axis. These shifts occur from 0% to 10% shift along the X-axis. We can see that shifts of 0% and 0.5% perform better than using RGB-only data. We can also see that a shift of 1% performs just as well as using RGB-only data.

Chapter 6

Conclusions and Recommendations

6-1 Conclusions

This thesis started with two problem statements, they are:

- **How does the use of RGB-D data affect the performance compared to the usage of RGB-only data, applied to the VI dataset?**
- **How much deviation is allowed in the registration of the depth data to the RGB data?**

We have given a general background on what image segmentation is. We have also given a general background on DL and the use of DL-specific algorithms in image segmentation. We eventually chose two DL specific algorithms from literature to implement, i.e. SegNet proposed by Badrinarayanan *et al.* [1], and FuseNet proposed by Hazirbas *et al.* [2]. Afterwards, this thesis has gone through its implementation pipeline, i.e. how the data was structured given raw data, and how the algorithms were implemented. Through experiments conducted during this thesis project, we can now answer these two questions.

How does the use of RGB-D data affect the performance compared to the usage of RGB-only data, applied to the VI dataset? We have seen from the four experiments conducted during this thesis work that the introduction of depth data affects the performance in a positive manner. This thesis has chosen to evaluate the performance of the algorithms using thee different evaluation metrics, i.e. global accuracy, mean class accuracy, and mean Intersect over Union (mIoU). Table 6-1 shows the results of the two networks implemented with the two different data-sets. From Table 6-1 we can see that the introduction of depth data increases the performance in all three evaluation metrics in both network architectures.

	Global accuracy	Mean class accuracy	mIoU
SegNet RGB + "fake" depth	73.99%	73.57%	48.69%
SegNet RGB-D	76.00%	73.63%	50.50%
FuseNet RGB + "fake" depth	85.54%	80.75%	64.13%
FuseNet RGB-D	**87.04%**	**83.96%**	**66.56%**

Table 6-1: The results of four different experiments in terms of global accuracy, mean class accuracy and the intersect over union value. We can see that the FuseNet architecture trained on RGB-D data performs best in all three evaluation metrics. We can also see that, in both RGB-only and RGB-D cases, the FuseNet architecture performs better than the SegNet architecture in all three evaluation metrics. This strengthens the notion that a late fusion method performs better than an early fusion method. Finally, we can see that using RGB-D data in both the SegNet and FuseNet architectures, the performance increases in all three evaluation metrics.

However, we can see that the FuseNet architecture, using RGB-only data or using RGB-D data, performs significantly better than the SegNet architecture. This implies that the addition of depth data by performing late fusion might perform better than by performing early fusion. This proposed idea is further encouraged when looking at visual examples. We saw that, in contrast to the FuseNet architecture, the SegNet architecture highly suffers from under-segmentation in a lot of cases. The SegNet architecture suffered from under-segmentation even in simpler cases. However, this does not mean that the FuseNet architecture does not suffer from under-segmentation. It suffers from it in a lighter manner. Also, when looking at visual examples, we see that the FuseNet architecture trained on RGB-only data suffered from over-segmentation in some cases, whilst the FuseNet architecture trained on RGB-D images did not. This also strengthens our results.

How much deviation is allowed in the registration of the depth data to the RGB data? We have seen from the experiments conducted during this thesis work on shifting the depth data along the X-axis that the performance of the network starts to deviate from its unaltered performance. Again, this thesis has chosen to evaluate the performance of the algorithms using thee different evaluation metrics, i.e. global accuracy, mean class accuracy, and mean Intersect over Union (mIoU). Table 6-2 shows the results of the different shifts along the X-axis. These shifts were established by systematically reducing the shift along the X-axis. From Table 6-2 we can see that the network trained on the unaltered depth data performs best in all three evaluation metrics. However, we wanted to know how much deviation is allowed in the registration such that the performance, in all three evaluation metrics, would still be considered better than the performance of using RGB-only data. Table 6-2 shows that a deviation of 0.5% along the X-axis accomplishes this result, while a deviation of one% does come close. We can thus conclude that the maximum allowed deviation along the X-axis would be 0.5%, which corresponds to a shift of 1 pixel of the depth data. This shift of one pixel, when translated back into its original image size, corresponds to a shift of 1.67 millimetres in the real world.

	Global accuracy	Mean class accuracy	mIoU
FuseNet RGB-D 0% shift	***87.04%***	***83.96%***	***66.56%***
FuseNet RGB-D 0.5% shift	**85.95%**	**82.85%**	**64.81%**
FuseNet RGB-D 1% shift	85.39%	83.12%	64.19%
FuseNet RGB-D 2% shift	85.14%	78.78%	63.38%
FuseNet RGB-D 5% shift	84.16%	74.76%	58.90%
FuseNet RGB-D 10% shift	85.05%	79.41%	62.76%
FuseNet RGB + "fake" depth	85.54%	80.75%	64.13%

Table 6-2: The results of the FuseNet RGB-D network trained on the different shifts of the depth data along the X-axis. The shifts occur from 0% to 10% shift by means of a random tree search. This thesis has chosen to evaluate the shifts 0%, 0.5%, 1%, 2%, 5%, and 10%. The results are evaluated in terms of global accuracy, mean class accuracy and the intersect over union value. We can see that a shift of 0% along the X-axis performs best in all three evaluation metrics. However, a shift of 0.5% performs better than using RGB-only data in all three evaluation metrics.

6-2 Recommendations for Future Work

Even though during the entirety of this thesis project we tried to work as complete as possible, due to time constraints, a couple of aspects were disregarded.

The first issue we would like to recommend for further investigation is the tuning of the hyper-parameters of the network. Currently, these parameters, such as the learning rate, the batch size, the number of training iterations, etc., were chosen on the basis of simple experiments. We would recommend a more extensive hyper-parameter tuning for the future.

Also, we have tried to reduce the possibility of the networks being over-fitted by designing the network using Batch Normalisation, Rectified Linear Unit (ReLU), and dropout operations to reduce over-fitting. However, no experiment to validate that the networks were over-fitted or not was conducted. This was not done mainly because of the fact that there was not enough data for us to work with. We would recommend a validation of the over-fitting problem for the future by gathering more data.

Finally, as for the shifts in the depth data, we have only shifted the depth data along the X-axis in one direction. For future work, we would recommend experimenting with the shift along the X-axis in both directions. Also, shifts along the Y-axis and rotations along the Z-axis should be experimented with to fully assess the robustness of the registration problem.

Appendix A

Code to pre-process the data

This Appendix contains all of the code to pre-process the raw data captured by the Ensenso stereo camera and the iDS RGB camera.

A-1 Pre-process depth information

```
% This code is to preprocess the PCD files to only get the depth map
% Also, this code gives every depth map a unique name
% This is necessary since all of the PCD files received from Fizyr have
% the exact same name.

% Add Inpaint_nans folder to workspace
addpath('..\Include\Inpaint_nans\Inpaint_nans')

% Gather all of the folders
AllFolders = dir(fullfile('C:\Git\students-assignments\
    SemanticSegmentation\MATLAB\Images\Fizyr_example\data*'));

% Initialize FolderNames variable
FolderNames = cell(length(AllFolders),1);
for i = 1:length(AllFolders)
    % Fill in the FolderNames variable to contain all of the folders
    % we are going to look into
    FolderNames(i) = {fullfile(AllFolders(i).folder,AllFolders(i).name)};
end

for j = 1:length(FolderNames)
    % Read current folder
    FolderName = FolderNames{j};
    % Navigate to 'resources' folder where the image is located in
    fileFolder = fullfile(FolderName, 'resources');
    % Find all .png images
```

```
26      dirOutput = dir(fullfile(fileFolder, '*.png'));
27      % Initialize the fileNames variable
28      fileNames = cell(length(dirOutput),1);
29      for m = 1:length(dirOutput)
30         % Fill in the fileNames variable to contain the path to all of the
31         % images we want to rename
32         fileNames(m) = {fullfile(dirOutput(m).folder,dirOutput(m).name)};
33      end
34      for k=1:length(fileNames)
35          % Read current filename
36          fname = fileNames{k};
37          % Read current PCD file
38          ptCloud = pcread(fname);
39          % Consider only the depth information
40          Z_value=double(ptCloud.Location(:,:,3));
41          % Use the inpain_nans function to fill in all of the NaN values
42          Z_value = inpaint_nans(Z_value, 2);
43          % Normalize the depth map to be between 0 and 1
44          normZ_value = Z_value - min(Z_value(:));
45          normZ_value = normZ_value ./ max(normZ_value(:));
46          % Convert depth map to uint8, this will make it between 0 and 255
47          normZ_value = im2uint8(normZ_value);
48          % Change the name to have a unique name
49          name = strcat('181204_', AllFolders(j).name,'_depth.png');
50          % Save the depth map in the 'FolderName' folder
51          imwrite(normZ_value, fullfile(FolderName, name))
52
53      end
54  end
```

A-2 Resize and crop all of the data

```
1  %% Resize and crop the data acoordingly
2  % (Can be incoporated in previous for loop), for testing purposes did
       them
3  % seperately.
4
5  fileFolders = fullfile('C:\Git\students-assignments\SemanticSegmentation\
       MATLAB\Images\Containers&Bags_PCD_example');
6  Outputdirectory = dir(fullfile(fileFolders,'*.png'));
7  fileNames = {Outputdirectory.name};
8  % For every depth map: resize to [360 480]
9
10 for l=1:length(fileNames)
11      ImageName = fileNames{l};
12      Image = imread(ImageName);
13      ResizedImage = imresize(Image,[360 480], 'nearest');
14      CroppedImage = ResizedImage(31:260,81:415,:);
15      imwrite(ResizedImage,fullfile('C:\Git\students-assignments\
            SemanticSegmentation\MATLAB\Images\Containers&Bags_PCD_example',
            ImageName))
16 end
```

A-3 Generate RGB-D data format

```
% This code generates RGB-D .mat files

%% Store RGB images in 4 channel matrix

% Locate all the RGB files from the specified folder
addpath('..\Images\generate_4channel_example\RGB')
fileFolderRGB = fullfile('C:\Git\students-assignments\
    SemanticSegmentation\MATLAB\Images\generate_4channel_example\RGB');
dirOutputRGB = dir(fullfile(fileFolderRGB,'*.png'));
fileNamesRGB = {dirOutputRGB.name};

TestImage = imread(fileNamesRGB{1});

[Height,Width,~] = size(TestImage);

numberFiles = length(fileNamesRGB);

NumberChannels = 4;

ImageTrainData = uint8(zeros(Height,Width,NumberChannels,numberFiles));

for k=1:length(fileNamesRGB)
    % Read RGB image
    CurrentImage=fileNamesRGB{k};
    Image=imread(CurrentImage);
    % Store RGB image in first 3 channels of the 4-channel matrix
    ImageTrainData(:,:,1,k) = Image(:,:,1);
    ImageTrainData(:,:,2,k) = Image(:,:,2);
    ImageTrainData(:,:,3,k) = Image(:,:,3);
end

%% Store Depth images in 4 channel matrix

% Locate all the depth files from the specified folder
addpath('..\Images\generate_4channel_example\depth')
fileFolderDepth = fullfile('C:\Git\students-assignments\
    SemanticSegmentation\MATLAB\Images\generate_4channel_example\depth');
dirOutputDepth = dir(fullfile(fileFolderDepth,'*.png'));
fileNamesDepth = {dirOutputDepth.name};

for i=1:length(fileNamesDepth)
    % Read depth image
    CurrentDepth=fileNamesDepth{i};
    Depth=imread(CurrentDepth);
    % Store depth image in the 4th channel of the 4-channel matrix
    ImageTrainData(:,:,4,i) = Depth(:,:);
end

%% Save every 4 channel image in a mat format

```

```
49 for l = 1:length(fileNamesRGB)
50     FileName=fileNamesRGB{l};
51     img = ImageTrainData(:,:,:,l);
52     name = strcat(FileName(1:end-3), 'mat');
53     save(fullfile('C:\Git\students-assignments\SemanticSegmentation\
           MATLAB\Images\generate_4channel_example\mat',name), 'img')
54 end
```

Appendix B

Code of the proposed networks

This Appendix contains the implemented networks. Section B-1 contains the code of the implemented SegNet RGB-D network, and section B-2 contains the code of the implemented FuseNet network.

B-1 SegNet RGB-D MATLAB implementation

```
imageSize = [230,335,4]; numClasses = 4;
SegNetRGBD_Network = segnetLayers([imageSize(1,1),imageSize(1,2),3],
    numClasses,'vgg16');

averageWeights = single(zeros(3,3,64));

for i = 1:64
    averageWeights(:,:,i) = (SegNetRGBD_Network.Layers(2,1).Weights
        (:,:,1,i) + SegNetRGBD_Network.Layers(2,1).Weights(:,:,2,i) +
        SegNetRGBD_Network.Layers(2,1).Weights(:,:,3,i))/3;
end

newWeights = single(zeros(3,3,4,64));

for i = 1:64
    newWeights(:,:,1:3,i) = SegNetRGBD_Network.Layers(2,1).Weights(:,:,:,
        i);
    newWeights(:,:,4,i) = averageWeights(:,:,i);
end

newBias = double(zeros(1,1,64));
conv4dlayer = convolution2dLayer(3,64,'Name','conv4d','Padding',[1  1],'
    BiasLearnRateFactor',0,'WeightL2Factor',0);
conv4dlayer.Weights = newWeights;
conv4dlayer.Bias = newBias;
```

```
21
22 newInput = imageInputLayer([230,335,4],'Name','newinputImage');
23
24 dropout1 = dropoutLayer(0.5,'Name','dropout1');
25 dropout2 = dropoutLayer(0.5,'Name','dropout2');
26 dropout3 = dropoutLayer(0.5,'Name','dropout3');
27 dropout4 = dropoutLayer(0.5,'Name','dropout4');
28 dropout5 = dropoutLayer(0.5,'Name','dropout5');
29 dropout6 = dropoutLayer(0.5,'Name','dropout6');
30
31 SegNetRGBD_Network = disconnectLayers(SegNetRGBD_Network,'inputImage','
       conv1_1');
32 SegNetRGBD_Network = disconnectLayers(SegNetRGBD_Network,'conv1_1','
       bn_conv1_1');
33
34 SegNetRGBD_Network = removeLayers(SegNetRGBD_Network,'inputImage');
35 SegNetRGBD_Network = removeLayers(SegNetRGBD_Network,'conv1_1');
36
37 SegNetRGBD_Network = addLayers(SegNetRGBD_Network,newInput);
38 SegNetRGBD_Network = addLayers(SegNetRGBD_Network,conv4dlayer);
39 SegNetRGBD_Network = addLayers(SegNetRGBD_Network,dropout1);
40 SegNetRGBD_Network = addLayers(SegNetRGBD_Network,dropout2);
41 SegNetRGBD_Network = addLayers(SegNetRGBD_Network,dropout3);
42 SegNetRGBD_Network = addLayers(SegNetRGBD_Network,dropout4);
43 SegNetRGBD_Network = addLayers(SegNetRGBD_Network,dropout5);
44 SegNetRGBD_Network = addLayers(SegNetRGBD_Network,dropout6);
45
46 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'newinputImage','
       conv4d');
47 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'conv4d','
       bn_conv1_1');
48
49 SegNetRGBD_Network = disconnectLayers(SegNetRGBD_Network,'pool3/out','
       conv4_1');
50 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'pool3/out','
       dropout1');
51 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'dropout1','conv4_1
       ');
52
53 SegNetRGBD_Network = disconnectLayers(SegNetRGBD_Network,'pool4/out','
       conv5_1');
54 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'pool4/out','
       dropout2');
55 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'dropout2','conv5_1
       ');
56
57 SegNetRGBD_Network = disconnectLayers(SegNetRGBD_Network,'pool5/out','
       decoder5_unpool/in');
58 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'pool5/out','
       dropout3');
59 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'dropout3','
       decoder5_unpool/in');
60
```

```
61 SegNetRGBD_Network = disconnectLayers(SegNetRGBD_Network,'decoder5_relu_1
       ','decoder4_unpool/in');
62 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'decoder5_relu_1','
       dropout4');
63 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'dropout4','
       decoder4_unpool/in');
64
65 SegNetRGBD_Network = disconnectLayers(SegNetRGBD_Network,'decoder4_relu_1
       ','decoder3_unpool/in');
66 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'decoder4_relu_1','
       dropout5');
67 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'dropout5','
       decoder3_unpool/in');
68
69 SegNetRGBD_Network = disconnectLayers(SegNetRGBD_Network,'decoder3_relu_1
       ','decoder2_unpool/in');
70 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'decoder3_relu_1','
       dropout6');
71 SegNetRGBD_Network = connectLayers(SegNetRGBD_Network,'dropout6','
       decoder2_unpool/in');
```

B-2 FuseNet MATLAB implementation

```
1  imageSize = [230,335,4]; numClasses = 4;
2  FuseNet_Matlab_Network = segnetLayers([imageSize(1,1),imageSize(1,2),3],
       numClasses,'vgg16');
3
4  averageWeights = single(zeros(3,3,64));
5
6  for i = 1:64
7      averageWeights(:,:,i) = (FuseNet_Matlab_Network.Layers(2,1).Weights
           (:,:,1,i) + FuseNet_Matlab_Network.Layers(2,1).Weights(:,:,2,i) +
           FuseNet_Matlab_Network.Layers(2,1).Weights(:,:,3,i))/3;
8  end
9
10 newWeights = single(zeros(3,3,1,64));
11
12 for i = 1:64
13     newWeights(:,:,1,i) = averageWeights(:,:,i);
14 end
15
16 newBias = double(zeros(1,1,64));
17 conv1dlayer = convolution2dLayer(3,64,'Name','conv1d','Padding',[1  1],'
       BiasLearnRateFactor',0,'WeightL2Factor',0);
18 conv1dlayer.Weights = newWeights;
19 conv1dlayer.Bias = newBias;
20
21 channel1Weights = single(zeros(1,1,4));
22 channel1Weights(1,1,4) = 1;
23 conv1channellayer = convolution2dLayer(1,1,'Name','conv1channel','
       WeightLearnRateFactor',0,'BiasLearnRateFactor',0,'WeightL2Factor',0);
24 conv1channellayer.Weights = channel1Weights;
```

```
25
26 channel3Weights = single(zeros(1,1,4,3));
27 channel3Weights(1,1,1,1) = 1; channel3Weights(1,1,2,2) = 1;
       channel3Weights(1,1,3,3) = 1;
28 conv3channellayer = convolution2dLayer(1,3,'Name','conv3channel','
       WeightLearnRateFactor',0,'BiasLearnRateFactor',0,'WeightL2Factor',0);
29 conv3channellayer.Weights = channel3Weights;
30
31 newInput = imageInputLayer(imageSize,'Name','newinputImage');
32
33 dropout1 = dropoutLayer(0.5,'Name','dropout1');
34 dropout2 = dropoutLayer(0.5,'Name','dropout2');
35 dropout3 = dropoutLayer(0.5,'Name','dropout3');
36 dropout4 = dropoutLayer(0.5,'Name','dropout4');
37 dropout5 = dropoutLayer(0.5,'Name','dropout5');
38 dropout6 = dropoutLayer(0.5,'Name','dropout6');
39
40 depthLayers = FuseNet_Matlab_Network.Layers(3:44,1);
41
42 for i = 1:length(depthLayers)
43    Name = depthLayers(i,1).Name;
44    Name = strcat(Name,'_depth');
45    depthLayers(i,1).Name = Name;
46 end
47
48 dropoutdepth1 = dropoutLayer(0.5,'Name','dropoutdepth1');
49 dropoutdepth2 = dropoutLayer(0.5,'Name','dropoutdepth2');
50
51 addition1 = additionLayer(2,'Name','addition1');
52 addition2 = additionLayer(2,'Name','addition2');
53 addition3 = additionLayer(2,'Name','addition3');
54 addition4 = additionLayer(2,'Name','addition4');
55 addition5 = additionLayer(2,'Name','addition5');
56
57 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'
       inputImage','conv1_1');
58 FuseNet_Matlab_Network = removeLayers(FuseNet_Matlab_Network,'inputImage'
       );
59
60 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,newInput);
61 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,conv1dlayer);
62 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,
       conv1channellayer);
63 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,
       conv3channellayer);
64 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,dropout1);
65 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,dropout2);
66 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,dropout3);
67 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,dropout4);
68 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,dropout5);
69 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,dropout6);
70 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,depthLayers);
71 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,dropoutdepth1);
```

```
72 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,dropoutdepth2);
73 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,addition1);
74 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,addition2);
75 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,addition3);
76 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,addition4);
77 FuseNet_Matlab_Network = addLayers(FuseNet_Matlab_Network,addition5);
78
79 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    newinputImage','conv3channel');
80 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    conv3channel','conv1_1');
81
82 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    newinputImage','conv1channel');
83 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    conv1channel','conv1d');
84 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'conv1d','
    bn_conv1_1_depth');
85
86 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'pool3/
    out','conv4_1');
87 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'pool3/out'
    ,'dropout1');
88 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'dropout1',
    'conv4_1');
89
90 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'pool4/
    out','conv5_1');
91 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'pool4/out'
    ,'dropout2');
92 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'dropout2',
    'conv5_1');
93
94 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'pool5/
    out','decoder5_unpool/in');
95 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'pool5/out'
    ,'dropout3');
96 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'dropout3',
    'decoder5_unpool/in');
97
98 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'
    decoder5_relu_1','decoder4_unpool/in');
99 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    decoder5_relu_1','dropout4');
100 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'dropout4',
    'decoder4_unpool/in');
101
102 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'
    decoder4_relu_1','decoder3_unpool/in');
103 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    decoder4_relu_1','dropout5');
104 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'dropout5',
    'decoder3_unpool/in');
```

```
105
106 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'
    decoder3_relu_1','decoder2_unpool/in');
107 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    decoder3_relu_1','dropout6');
108 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'dropout6',
    'decoder2_unpool/in');
109
110 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'
    pool3_depth/out','conv4_1_depth');
111 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    pool3_depth/out','dropoutdepth1');
112 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    dropoutdepth1','conv4_1_depth');
113
114 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'
    pool4_depth/out','conv5_1_depth');
115 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    pool4_depth/out','dropoutdepth2');
116 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    dropoutdepth2','conv5_1_depth');
117
118 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'relu1_2
    ','pool1');
119 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'relu1_2','
    addition1/in1');
120 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    relu1_2_depth','addition1/in2');
121 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'addition1'
    ,'pool1');
122
123 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'relu2_2
    ','pool2');
124 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'relu2_2','
    addition2/in1');
125 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    relu2_2_depth','addition2/in2');
126 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'addition2'
    ,'pool2');
127
128 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'relu3_3
    ','pool3');
129 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'relu3_3','
    addition3/in1');
130 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    relu3_3_depth','addition3/in2');
131 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'addition3'
    ,'pool3');
132
133 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'relu4_3
    ','pool4');
134 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'relu4_3','
    addition4/in1');
```

```
135 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    relu4_3_depth','addition4/in2');
136 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'addition4'
    ,'pool4');
137
138 FuseNet_Matlab_Network = disconnectLayers(FuseNet_Matlab_Network,'relu5_3
    ','pool5');
139 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'relu5_3','
    addition5/in1');
140 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'
    relu5_3_depth','addition5/in2');
141 FuseNet_Matlab_Network = connectLayers(FuseNet_Matlab_Network,'addition5'
    ,'pool5');
```

Bibliography

[1] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," in *https://arxiv.org/abs/1511.00561v3*, eprint arXiv:1511.00561, 2016.

[2] C. Hazirbas, M. Ma, C. Domokos, and D. Cremers, "Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture," in *ACCV*, 2016.

[3] R. van Loon, "Machine learning explained: Understanding supervised, unsupervised, and reinforcement learning," in *http://bigdata-madesimple.com/machine-learning-explained-understanding-supervised-unsupervised-and-reinforcement-learning/*, 2018.

[4] J. Patterson and A. Gibson, *Deep Learning: a practitioner's approach*. O'Reilly Media, Inc., 2017.

[5] E. Smolyanski, "The basics of video object segmentation," in *https://techburst.io/video-object-segmentation-the-basics-758e77321914/*, 2017.

[6] R. Babuska, "Knowledge based control systems lecture notes," in *TU Delft DCSC Lecture Notes*, 2018.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *https://arxiv.org/abs/1409.1556v6*, eprint arXiv:1409.1556, 2014.

[8] M. Loukadis, J. Cano, and M. O'Boyle, "Accelerating deep neural networks on low power heterogeneous architectures," in *11th International Workshop on Programmability and Architectures for Heterogeneous Multicores*, 2018.

[9] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," in *https://arxiv.org/abs/1511.02680v2*, eprint arXiv:1511.02680, 2016.

[10] A. Rosebrock, "Intersection over union (iou) for object detection," in *https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/*, 2016.

[11] N. van Toorn, "Netherlands in eu top 5 online shopping," in *https://www.cbs.nl/en-gb/news/2018/38/netherlands-in-eu-top-5-online-shopping*, 2018.

[12] A. Rosenfeld, "Picture processing by computer," *ACM Computing Surveys (CSUR)*, vol. 1, no. 3, pp. 147–176, 1969.

[13] L. G. Roberts, "Machine perception of three-dimensional solids," *MIT Thesis*, 1963.

[14] T. Sakai, M. Nagao, and T. Kanade, "Computer analysis and classification of photographs of human faces," in *Proc. First USA-JAPAN Computer Conference*, pp. 55–62, 1972.

[15] G. B. Coleman, "Image segmentation by clustering," *USCIPI report 750*, 1977.

[16] O. Russakovsky, J. Denh, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, ch. All. eprint arXiv:1409.0575, 2014.

[17] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *CoRR*, vol. abs/1312.6229, 2013.

[18] R. M. Haralick and L. G. Shapiro, "Image segmentation techniques," *Computer Vision, Graphics, and Image Processing 29, 100-132(1985)*, 1985.

[19] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 888–905, Aug. 2000.

[20] R. Adams and L. Bischof, "Seeded region growing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, pp. 641–647, June 1994.

[21] S. A. Hojjatoleslami and J. Kittler, "Region growing: A new approach," *IEEE Transactions on Image Processing*, vol. 7, pp. 1079–1084, July 1998.

[22] A. Vedaldi and S. Soatto, "Quick shift and kernel methods for mode seeking," *Computer Vission - ECCV 2008*, vol. 5305, 2008.

[23] S. S. Al-amri, N. V. Kalyankar, and S. D. Khamitkar, "Image segmentation by using threshold techniques," *Journal Of Computing*, vol. 2, no. 5, pp. 83–86, 2010.

[24] T. M. Mitchel, *Machine Learning*. McGraw-Hill Companies, Inc., 1997.

[25] R. Bekkerman, M. Bilenko, and J. Langford, *Scaling up Machine Learning*. Cambridge University Press, 2012.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[27] D. Clausi, "K-means iterative fisher (kif) unsupervised clustering algorithm applied to image texture segmentation," *Pattern Recognition*, vol. 35, pp. 1959–1972, 2002.

[28] H. Lee, Y. Largman, P. Pham, and A. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Advances in Neural Information Systems*, vol. 22, NIPS, 2009.

[29] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[30] K. Fukushima, "Neural network model for a mechanism of pattern recognition unaffected by shift in position - neocognitron," *Trans. IECE*, vol. J62-A(10), pp. 658–665, 1979.

[31] K. Fukushima, "Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[32] K. Fukushima, "Artificial vision by multi-layered neural networks: Neocognitron and its advances," *Neural Networks*, vol. 37, pp. 103–119, 2013.

[33] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, 1989.

[34] C. Szegedy, W. Lie, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabbinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pp. 1–9, 2015.

[35] L. Yang, S. Hanneke, and J. Carbonell, "A theory of transfer learning with applications to active learning," *Machine Learning*, vol. 90, pp. 161–189, 2012.

[36] S.-J. Park, K.-S. Hong, and S. Lee, "Rdfnet: Rgb-d multi-level residual feature fusion for indoor semantic segmentation," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[37] S. Gupta, R. Girshick, P. Arbelaez, and J. Malik, "Learning rich features from rgb-d images for object detection and segmentation," *Computer Vision âĂŞ ECCV 2014*, vol. 8695, 2014.

[38] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, IEEE, 2015.

[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[40] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *https://arxiv.org/abs/1505.04366v1*, eprint arXiv:1505.04366, 2015.

[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, 2014.

[42] F. Estrada, *Advances in Computational Image Segmentation and Perceptual Grouping*. Department of Computer Science, University of Toronto, 2005.