Master's thesis

NTNU
Norwegian University of Science and Technology

Christian Peter Bech Aschehoug

Segmentation and pose estimation of objects in RGB-D data using deep learning

Master's thesis in Cybernetics and Robotics
Supervisor: Jan Tommy Gravdahl, Øystein Skotheim
June 2019

NTNU
Kunnskap for en bedre verden

Christian Peter Bech Aschehoug

Segmentation and pose estimation of objects in RGB-D data using deep learning

Master’s thesis in Cybernetics and Robotics
Supervisor: Jan Tommy Gravdahl, Øystein Skotheim
June 2019

Norwegian University of Science and Technology

NTNU
Kunnskap for en bedre verden

Abstract

Machine vision has become an increasingly important part of modern industry. The overall machine vision process often includes imaging, analysis of images, and information extraction. Deep learning techniques, especially convolutional neural networks (CNNs), is the state-of-the-art in image analysis and scene understanding. They have shown impressive results in tasks such as object classification, detection, and segmentation.

This thesis combines machine learning with 3D computer vision by evaluating how deep neural networks can segment and estimate the pose of 3D data. Extracted RGB images from 3D data captured by Zivid camera, is used as input to two deep neural networks, namely LinkNet34 and Mask R-CNN, for segmentation. LinkNet34 performs binary segmentation, and Mask R-CNN computes instance segmentation. The outputted mask from these two networks is used to segment the 3D data. Segmented 3D data from instance segmentation is used in pose estimation of objects in the scene.

Experiment results prove that segmentation and pose estimation on data captured by Zivid camera is achievable. Binary segmentation was only able to partially segment data, while Mask R-CNN successfully segmented the different objects in the input image. Training data and synthetic data gives numerically and visually verification of performance. This is used to substantiate results from data captured by Zivid camera. Pose estimation is done using fast global registration. Without optimizing the registration method, the results are surprisingly good, with small errors in translation and rotation for most cases.

Sammendrag

Maskinsyn er en stadig viktigere del av moderne industri og . Den overordnede maskinvisjonsprosessen består av bildetakning, analyse av bilder og informasjonshenting. Dype læringsteknikker, spesielt konvolusjonelle nevrale nettverk, er toppmoderne metoder i bildeanalyse og sceneforståelse. De har vist imponerende resultater i oppgaver som objektklassifisering, objektdeteksjon og bildesegmentering.

Denne oppgaven kombinerer maskinlæring med 3D-datasyn for å evaluere hvordan dype nevrale nettverk kan segmentere og estimere posisjon og orientering i 3D-data. RGB-bilder henter fra 3D-data tatt med Zivid-kamera, brukes som input til to dype nevrale nettverk, LinkNet og Mask R-CNN, for segmentering. LinkNet utfører binær segmentering, mens Mask R-CNN segmenterer hver instanse for seg. De genererte maskene fra de to nevrale nettverkene brukes til å segmentere 3D-dataene. Segmenterte 3D-data fra Mask R-CNN brukes videre til å estimere posisjon og orientasjon for objekter i scenen.

Resultater fra ekpserimenter viser at segmentering og posisjonsestimering på data tatt med Zivid kamera er oppnåelig. Binær segmentering var bare i stand til delvis segmentere dataen, mens Mask R-CNN klarte å segmenterte de ulike objektene i dataen. Treningsdata og syntetiske data gir numeriske og visuelle bekreftelse på gode segmenteringer, som brukes til å underbygge resultater fra data tatt med Zivid-kamera. Pose estimering er gjort ved hjelp av fast global registrering. Uten å optimalisere registreringsmetoden er resultatene overraskende gode, med mindre feil i translasjon og orientering for de fleste tilfellene.

Preface

This thesis concludes my Master of Science degree in Cybernetics and Robotics at Norges teknisk-naturvitenskapelige universitet (NTNU). The work has been carried out during the spring of 2019, and the study has been completed in collaboration with Zivid.

The project was carried out with minimal prerequisites. A lab spot with a Zivid camera and a powerful computer was provided. Many hours of self-education have been spent to properly understand 3D data, point cloud processing, neural network, and PyTorch framework since my insight in these fields was limited. The choices made during the project are based on my knowledge. Most of the software implementations are developed independently, but with inspiration from other implementations.

I would like to thank my supervisor Jan Tommy Gravdahl, and co-supervisor from Zivid, Øystein Skotheim, for their guidance throughout the project. Allowing me to work on this project have been very educational, and my interest in machine learning and computer vision is further increased. Furthermore, I would like to thank Aleksandar Babic from Zivid, for his many helpful advise regarding machine learning.

Lastly, a special thank to my fellow students, Øystein Brox and Edvard Rauø Vasdal, for excellent collaboration and social working environment at the office.

Trondheim June 6, 2019
Christian Peter Bech Aschehoug

Contents

List of Tables

List of Figures

Abbreviations

CAD	Computer-Aided Design
CNN	Convolutional Neural Network
FCN	Fully Convolutional Neural Network
FPFH	Fast Point Feature Histogram
ICP	Iterative Closests Point
IoU	Intersection over Union
MPC	model predictive control
RANSAC	RANdom SAmple Consensus
R-CNN	Region-based Convolutional Neural Network
ReLu	Rectified Linear Unit
RGB-D	Red, Green, Blue - Depth
RoI	Region of Interest
SGD	Stochastic gradient descent
SVM	Support Vector Machine
RPN	Region Proposal Network
ToF	Time-of-Flight

Chapter 1

Introduction

1.1 Motivation

The industry is rapidly advancing towards industry 4.0. Two of the leading technologies responsible for this progress is robotic machine vision and machine learning. Robotic operations often rely on visual data for decision-making. Such operations can be pose estimation, object detection/localization, and data segmentation. When Microsoft launched Microsoft Kinect in 2009, 3D cameras were made affordable and commercialized. This contributed to accelerating the transition of computer vision from 2D to 3D.

Present-day, 3D sensors that acquire 3D data are readily available, and 3D machine vision algorithms play an increasingly important part in smart manufacturing, autonomous driving, medical image processing, and robotics.

The main challenge with visual-based robotic applications is to recognize and localize objects. Scenes vary in complexity, shape, and sizes of objects differ, and objects can be occluded. Overcoming these difficulties is an active field of research in computer vision.

Machine learning became prominent within the modern computer vision field after the introduction of convolutional neural networks (CNNs) and its success at image classification tasks. These methods have been further developed and adapted to new fields, e.g., object detection and scene segmentation.

Today, neural network, especially deep neural networks, can gain a high-level understanding of complex scenes. These approaches are considered state-of-the-art, and the best models are as good as and even better than humans to recognize and localize objects in an image.

1.2 Problem description

Zivid delivers industry 3D cameras and software to increase productivity in robotic applications. Example of such tasks is bin-picking, pick-and-place, and quality control. A challenge with these tasks is to detect the object in the 3D scene, and accurately calculate its position. Zivid wants to investigate how machine learning methods can be used for object recognition and pose estimation in 3D images.

The problem can be formally stated as

Problem statement

1. Use a neural network to do object detection and segmentation in 3D images.
2. Use 3D algorithm to estimate the pose of segmented objects from (1).

This thesis focuses mostly on (1), but pose is also estimated.

The problem statement is divided into the following steps:

- Research of model architectures suitable for segmenting objects from the background in a typical pick & place scenario.
- Investigate existing datasets commonly used for training a neural network for segmentation.
- Develop and train a model using existing dataset and test the model with self-generated data from Zivid camera.
- Investigate methods for fine alignment and test their applicability for accurate pose estimation.

1.3 Limitations

Even though the task at hand is open, are there still some limitations worth mentioning.

- **Computational power**: The models should be able to run on most computers. This excludes usage of some methods, especially very deep neural networks.
- **Pre-trained model**: To ease the training process is transfer learning used. The new model is initialized using weights from a pre-trained model. This sets some limitations on what model to be used.
- **Object types**: Using pre-trained weights and existing datasets restrains what type of objects that can be used when generating new data with Zivid camera. For evaluation pose estimation, these objects must have a corresponding point cloud or mesh available as this is needed for detecting pose.

1.4 Contributions

To solve the task described in section 1.2, following contributions, have been made by the author.

- **Synthetic data generation**: To obtain data with ground truth pose of every instance, was a simulator created by fellow student Edvard Rauø Vasdal as part of his master thesis, used. The simulator was modified from generating boxes to take in a point cloud sampled from a CAD model. The model creates a 3D scene and generates, RGBD-image, ground truth masks, and pose for every instance in the scene. Fifty different scenes were generated using this simulator.
- **Training pipeline**: A script for training neural network has been implemented, where the user only needs to specify hyperparameters. Training can be done using 3-channel input or 4-channel input, depending on the neural network. The method builds a data loading function that reads images from the WISDOM-Real training dataset from Danielczuk et al. (2018). Pre- and post-processing of data is also implemented. Pre-processing includes resizing, normalization, and augmentation of input data. Post-processing consists of denormalization and masks extraction. Batch loss validation loss and evaluation metrics are saved during training, along with the model weights with the lowest loss.
- **Binary segmentation**: A neural network named LinkNet34 have been developed to do binary segmentation. This network was selected after extensive research on segmentation methods and model architectures concerning the limitations in section 1.3. The model architecture is adapted from GitHub Shvets et al. (2018b) and modified to best solve the tasks at hand. It is trained using the implemented training pipeline. Both training data and data taken by Zivid camera, are evaluated on LinkNet34. Two versions of this network have been created, one 3-channel for RGB input and one 4-channel for RGBD input.
- **Instance segmentation**: Another neural network named Mask R-CNN (He et al. (2017)) is used for instance segmentation. This model is already trained, and the pre-trained network is extracted from GitHub Abdulla (2017). Both synthetic data from the simulator and real data is tested on this model. Intersection over Union is implemented and used as the evaluation metric. Every segmented image from the synthetic dataset can be seen in the appendix.
- **Segmentation of 3D data**: Methods for segmenting 3D data using the output masks of both LinkNet34 and Mask R-CNN have been developed. The pinhole camera model is used to generate 3D scenes for the WISDOM-Real training data, while for data captured by Zivid camera is the 3D data segmented directly. Binary segmentation segments 3D data into background or objects. Instance, segmentation segments 3D data for every detected object.
- **Pose estimation**: An implementation for estimating pose for every instance using segmented 3D data from Mask R-CNN is created. The algorithm used fast global registration (Zhou et al. (2016)) to align the segmented point cloud with a CAD point

cloud. The CAD point cloud is generated by sampling points on CAD models found on the web. The same point clouds are used in the simulator. Implementation of fast global registration is extracted from Open3D, an open source 3D library (Zhou et al. (2018)). The centroid of every instance is calculated and used as initialization of the registration method.

1.5 Outline

This report is organized as follows.

Chapter 2 introduces the necessary background theory to understand the methods that have been implemented properly. A more in-depth theory of neural network and some architectures is presented as this is a the core of this thesis. Then, different types of segmentation techniques are presented.

Chapter 3 gives a brief literature review of related work.

Chapter 4 explains the experimental setup, which software and hardware have been used to implement and train the neural network, and what kind of data has been used for training.

Chapter 5 presents an overview of the two neural networks used, their architecture, and how they have been implemented. Details around training pipeline, such as pre- and post-processing, validation metrics and pose estimation is then explained.

Chapter 6 presents the results of the experiments, in particular, binary segmentation, instance segmentation, and pose estimation. After each topic, a discussion of the specific results is given.

Chapter 7 includes the conclusion of this thesis and suggestions for future work.

Chapter 2

Background Theory

2.1 RGBD data methods

The challenge of collecting 3D geometric data of a scene is to capture the depth. There are several different ways of estimating/calculating this depth, but they can be divided into three main categories, namely stereo vision, structured light, and time-of-flight (ToF).

These three methods use either multiple cameras, combine cameras with a light source or both to generate 2D data. Accuracy and complexity of the setups differ for each setup, as well as the range. Hence, which method to use depends on the specific application.

2.1.1 Stereo Vision

One way of computing 3D geometry from a scene is by using two cameras. By comparing scene information from two (often slightly) different viewpoints, 3D information is extracted by inspecting the relative position of objects in the two images. Using two cameras is called passive stereo vision.

The two cameras are often aligned parallel to each other. This simplifies geometric calculations because the camera center are at same height, the focal length is equal for both cameras, and the epipolar lines falls along the horizontal lines of the images. If cameras are not in parallel, image rectification must be applied, before 3D information about the scene can be obtained.

Disparity, the relative difference of the pixel coordinate of an object, is calculated using triangulation and the depth is calculated from this disparity using

$$Z = f\frac{b_x}{d}$$

Here, f is the focal length, d is the disparity and b_x is the distance between camera centers.

Visual representation of stereo vision setup and the disparity, $d = X_L - X_R$, can be seen in Figure 2.1.

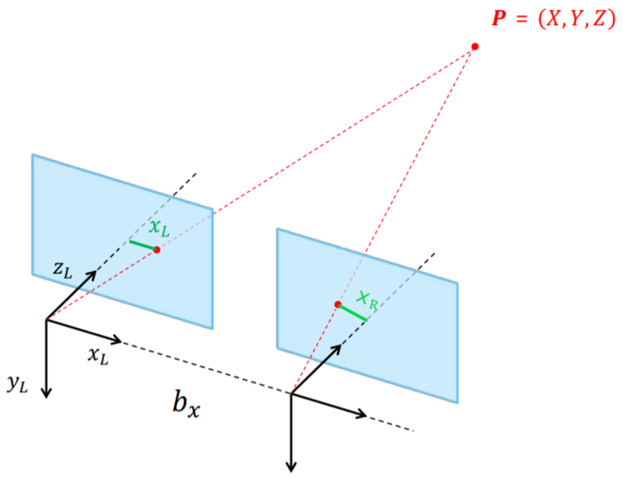


Figure 2.1: Parallel stereo vision setup with two cameras. Source: lecture slides from TTK21 Introduction to Visual Simultaneous Localization and Mapping.

2.1.2 Structured Light

Instead of using two cameras, one of the cameras are replaced with a projector. The principle of structured light is simple. A light source or a projector illuminates a scene with structured light. An imaging sensor, e.g camera, captures an image of the scene under this structured light illumination. Because of the object surfaces will the pattern be distorted.

The geometric relationship between a structured light projector, an object surface point P, and a camera can be seen in Figure 2.2.. Using triangulation, this can be expressed as

$$Z = B\frac{\sin(\theta)}{\sin(\alpha + \theta)}$$

where Z is the depth estimate, θ is the angles between the projected light ray, B is the baseline between camera and projector , and α is the corresponding angle between the light ray and the camera.

The structured light can be single patterned or multiple patterned, depending on how accurate the depth estimate must be. The light can either be visible or infrared, containing colors or be gray-scale.

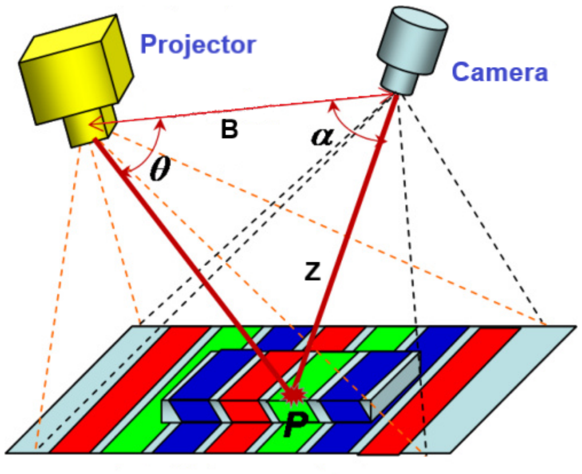


Figure 2.2: Illustration of structured light set up. Source: Geng (2011).

2.1.3 Time of Flight Camera

A third way of gathering 3D information about a scene is by measuring the time-of-flight (ToF) it takes for a camera to register a light pulse traveling from a light source to the scene and back again after reflection. If continuous wave are used, the phase shift between the reflected light and the original light is used to calculate depth.

For pulse wave, the depth is expressed as

$$Z_{TOF} = \frac{c}{2}\Delta T$$

while for the continuous wave, the depth is

$$Z_{TOF} = \frac{c}{2}\frac{\Delta\varphi}{2\pi f}$$

Here, c is the speed-of-light constant, and ΔT is the time it takes to detect the reflected light, $\Delta\varphi$ is the phase shift, and f is the frequency of the light. Figure 2.3 shows the principle for measuring the depth Z_{TOF}

In contrast to the two previous techniques are ToF cameras able to estimate a depth map directly without the use of traditional computer vision algorithms. Therefore, ToF cameras are often able to run real time with low software complexity. But, it is not as accurate as structured light cameras, and it works bad in heavy illuminated environment.

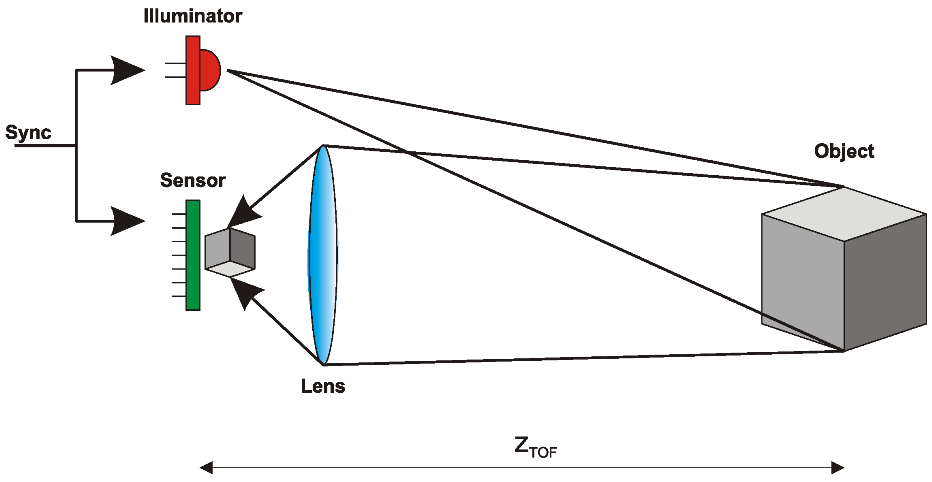


Figure 2.3: Illustration of ToF camera. Source: Perenzoni et al. (2011).

2.2 Segmentation

One of the key problems in modern computer vision is to get a complete scene understanding by recognizing the content of an image. Neural networks can be divided into three levels, high, mid and low, depending on what they are able to detect.

High-level is *image classification* networks that outputs if objects are present in the image or not. Traditional CNNs with fully connected layers belongs here. Mid-level is *object detection* networks that both classify and locate objects by drawing bounding boxes around them. R-CNN by Girshick et al. (2014) is an example of this type of network. Low-level is *segmentation* networks where the network classifies each pixel to one of the predetermined classes. These networks classifies, locates and creates segmentation masks for every object in the scene. Mask R-CNN by He et al. (2017) is an example of this type of network.

For segmentation networks it is common to distinguish between semantic segmentation and instance segmentation. The latter delineate between different instances of the same class, while the first only segments between classes. Instance segmentation is more complicated as it combines object detection and semantic segmentation. Instance segmentation network requires correct detection of all objects in an image as well as accurately segmenting each instance.

Figure 2.4 shows what different types of networks can output for the different levels.

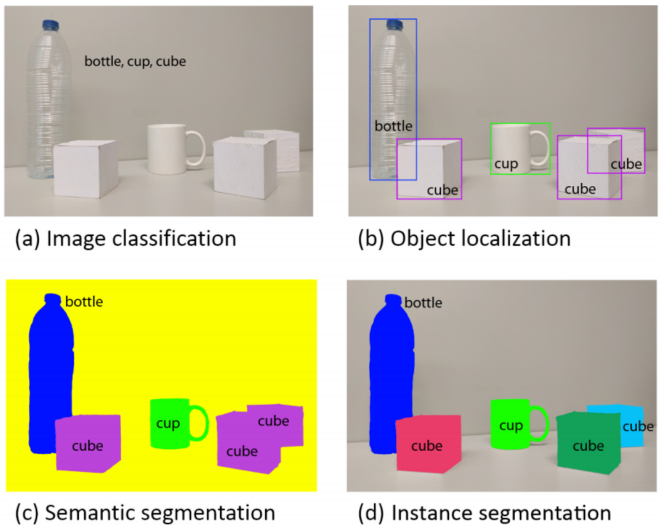


Figure 2.4: Different types of segmentation. High-level (top left) are able to classify objects in a scene, mid-level (top right) can also draw bounding boxes around every object, and low-level (bottom) is divided into semantic and instance segmentation. Source: Garcia-Garcia et al. (2017)

Semantic segmentation used to be solved using conditional random field, a probabilistic framework which model relationships between pixels. However, with the development of neural networks, semantic segmentation problems is now solved using deep learning methods.

A typical semantic segmentation architecture follows the FCN structure introduced by Long et al. (2015). The encoder network is often a pre-trained image classification network, and it detects discriminative features in the image. Feature extraction performance depends on the backbone model, which depends on the task. The decoder project the detected features onto the pixel space for a dense classification.

2.3 Neural Networks

Neural networks have made a significant impact in computer and machine vision. Its applications are used in everything ranging from industry bin picking and autonomous vehicles to medical image processing and consumer product, e.g., mobile phones.

2.3.1 Supervised & Unsupervised learning

Machine learning can be divided into supervised and unsupervised learning, depending on the way the model is trained.

Supervised learning is when prior knowledge about the output of the data, a ground truth, exists. Every feature in the dataset is associated with a label or target. Since the target y is provided, it shows the machine learning system what to do, hence supervised learning Goodfellow et al. (2016). The goal is to learn a mapping function between the input and output, mathematically expressed as

$$y = \mathcal{H}(\mathbf{x})$$

where y is the known output, x is the input and $\mathcal{H}$ is a mapping function. A model of the mapping function is generated by training on the known dataset.

Unsupervised learning is when x is the only known parameter. The model learns the underlying structure of the data, and there are no right or wrong outputs.

2.3.2 Basics of Neural Network

A neural network consists of multiple neurons. A neuron consists of four parts; input values, weights and biases, weighted sum, and an activation function, see Figure 2.5.

Multiple neurons together create a layer, and multiple layers form a neural network. The input to a neuron is either the input to the network (if it is the input layer), or the output of the previous layer. Figure 2.6 presents a multilayer neural network. The input to a layer x is multiplied with the layer corresponding weights w and biases b, such that the weighted sum for all j neurons in the layer is

$$\text{weighted sum} = w \cdot x = \sum_j w_j x_j + b_j \tag{2.1}$$

An activation function, σ, maps this weighted sum to a desired interval, often (0,1) or (-1,1). The mapping depends on the type of activation function, which can be linear or nonlinear. The goal of training a neural network is to generate a mapping function that approximates the output, Y, given input X. A loss function quantifies how well this mapping is done. The design of the loss function depends on the specific problem.

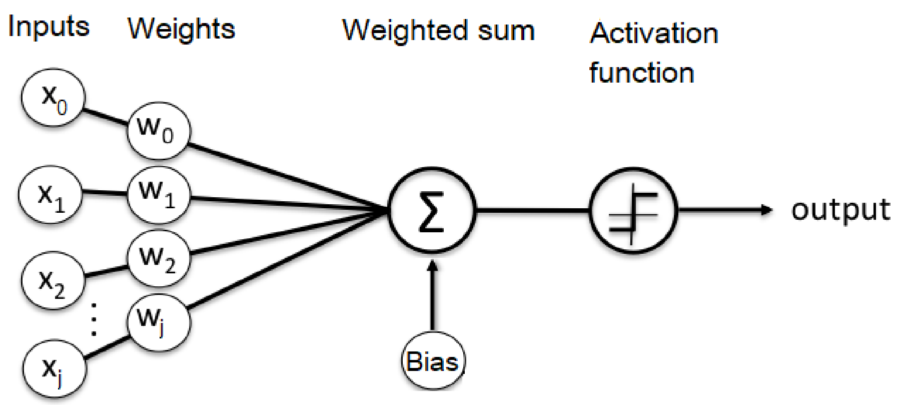


Figure 2.5: Visualization of a single neuron with multiple inputs.

Adjusting neural networks weights and biases change the output value of the loss function. How to update weights and biases, and hence minimizing the loss function, is calculated using gradient descent techniques.

Calculating gradients of the neural network is done in the forward pass. The input signal flows through the network, and the output of the network is compared against the ground truth. In the backward pass, backpropagation calculates the partial derivative, which is used to update the weights and biases of the network.

The gradient of a cost function C consisting of m variables, $v_1, \ldots, v_m$, is defined as

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m} \right)^T \tag{2.2}$$

A small change in $\Delta v = (\Delta v_1, \ldots, \Delta v_m)^T$, results in a small change ΔC in C.

$$\Delta C \approx \nabla C \cdot \Delta v \tag{2.3}$$

To assure that (2.3) holds, a learning rate η is chosen small enough such that small changes in variables can be expressed as.

$$\Delta v = -\eta \nabla C \tag{2.4}$$

(2.4) helps the neural network converge. It also guarantees that (2.3) is negative, and thus converge to a minimum following the update rule.

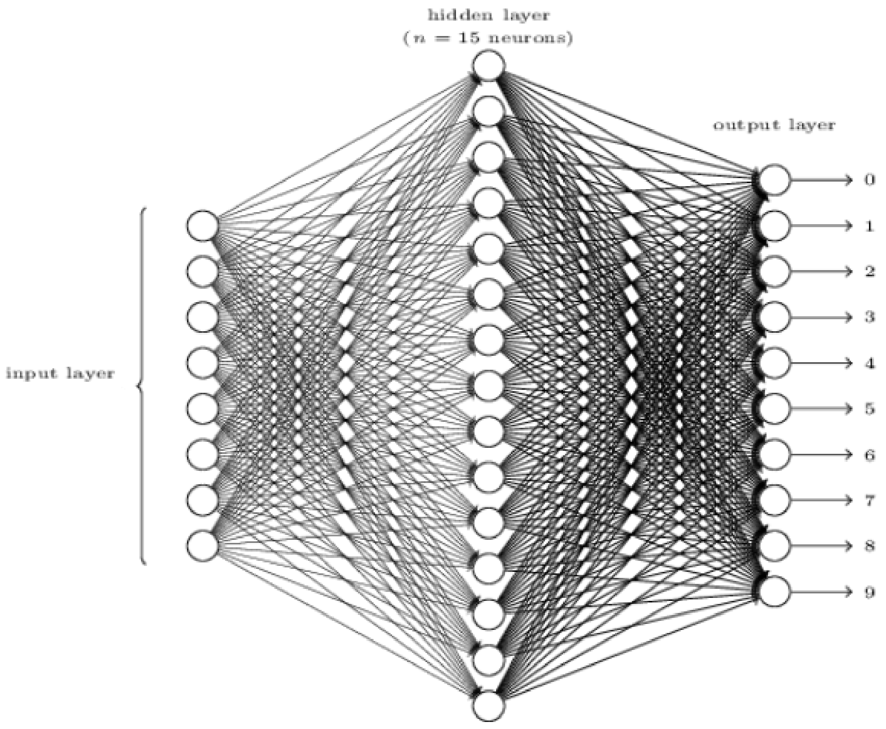


Figure 2.6: Visualization of an artificial neural network: Multiple neurons connected together forms a neural network. This network have 7 input neurons, one hidden layer with 15 neurons and 10 output neurons in output layer. Source Nielsen (2015).

$$v \rightarrow v' = v - \eta \nabla C \tag{2.5}$$

A given weight w_k and bias b_l are updates using the update rule. This is what makes a neural network learn.

$$w_k \rightarrow w_k' = w_k - \eta \frac{\partial C}{\partial w_k} \tag{2.6}$$

$$b_l \rightarrow b_l' = b_l - \eta \frac{\partial C}{\partial b_l} \tag{2.7}$$

2.3.3 Convolutional Neural Network

As mentioned above, many neurons connected forms a layer. A fully connected layer is when each neuron in layer $l-1$ is connected to every neuron in layer l. This is inefficient when doing image analyses because the network does not use spatial information.

Instead, convolutional neural network (CNN) is inspired by the biological process in the brain, where neurons only respond to its receptive field, a restricted region of interest. CNN uses local receptive fields to connect neurons from one layer to the next. Only a specific area of neurons are connected to neurons in the next layer. This is called a convolutional layer. One or more convolutional layers form a conventional network, From Goodfellow et al. (2016); *Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers*.

Convolutional layer shares weights and bias for each hidden layer. This reduces the number of parameters considerably, yet more importantly, it makes every neuron in a layer detect the same feature. For this reason, such mapping is often called feature mapping. The shared weights and bias defines the filer/kernel. The size of a feature map is determined by the number of filters, stride, and zero-padding. Different filters detect different features and hence generates different feature maps.

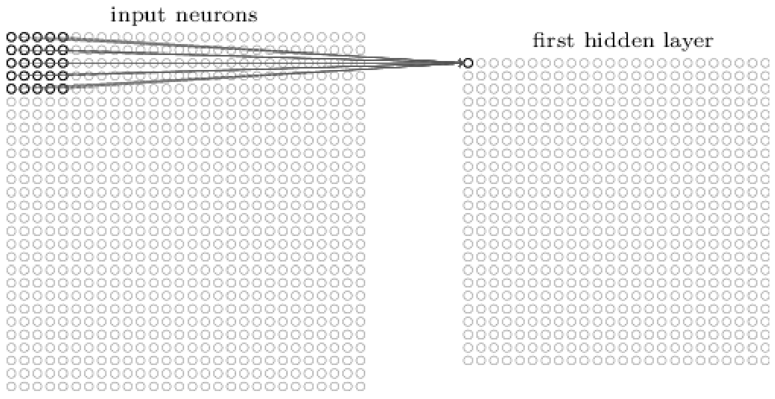


Figure 2.7: Visualization of a convolutional layer with 28x28 input size, 5x5 filter size and stride 1. Source: Nielsen (2015).

Right after a convolutional layer comes a pooling layer typically. This layer simplifies the information from the convolutional layer by downsampling the feature map, e.g., max-pooling. Similar to the convolutional layer are pooling layers, also depending on the filter size and stride. This layer reduced the dimension and therefore, the number of parameters needed in the CNN.

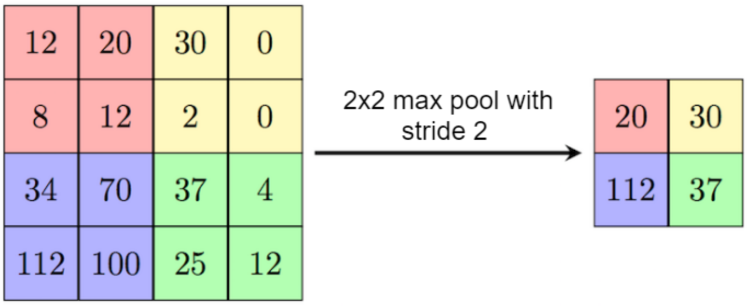


Figure 2.8: Visualization of a simple max-pooling layer with 4×4 input size, 2×2 filter size and stride 2.

The last layer(s) in CNN are usually fully-connected layers. These layers connect every neuron in the previous layer to every neuron in the next layer. The final layer of the network classifies the input data based on the features detected from the earlier layers. A complete CNN can be seen in Figure 2.9.

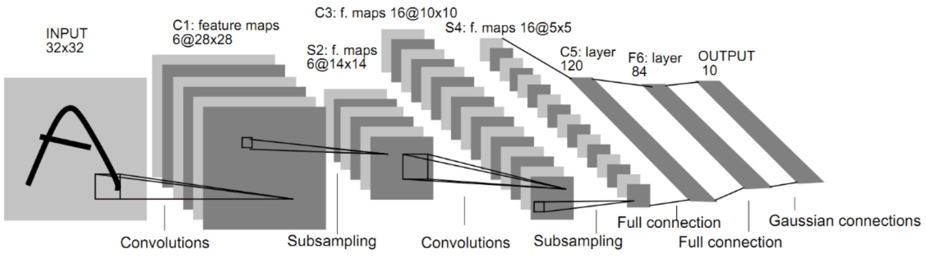


Figure 2.9: Representation of a convolutional neural network. Source: Lecun et al. (1998).

2.3.4 Residual Neural Network

Detecting more features in CNN is done by adding more layers. This makes the network deeper but can also lead to two problems, namely the degradation problem and vanishing gradient problem.

The vanishing gradient problem is a problem regarding the gradient in later layers in a CNN. These converge either to zero or very high values. When the gradients tend to zero will not the weights in the network update. This results that a network will not learn. This problem can be partially solved by normalizing the input data and use normalization layers between convolutional layers.

The degradation problem was first discussed in He et al. (2015). Despite adding more layers, and thus more parameters, the researchers observed a lower accuracy in training.

It is easier for the network to learn a zero mapping, $\mathcal{H}(\mathbf{x}) = 0$, than an identity function, $\mathcal{H}(\mathbf{x}) = x$. This leads to optimization problems.

Therefore, they presented a residual framework for deep neural network, called Residual Neural Network. Instead of only straight lined connections between layers, was shortcut connection added in the architecture. This efficiently solves both the vanishing gradient and degradation problem and enables networks to be very deep with, e.g., ResNet152, which consists of 152 layers.

Instead of doing the desired original mapping $\mathcal{H}(\mathbf{x})$, a new mapping is defined as

$$\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x} \tag{2.8}$$

This makes it possible to rewrite the original mapping to

$$\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x} \tag{2.9}$$

where x is the shortcut connection, in this case, a simple identity mapping. This connection does not add more parameters or complexity to a model. Hence, residual neural networks can be very deep.

Residual networks are easier to optimize than regular CNN. In an extreme case, it is easier to push the residual ($\mathcal{F}(\mathbf{x})$) to zero rather than fit an identity mapping.

When implementing a residual network, it is coming to use ResNet blocks. There are two different blocks. The first block is two layers deep and is used in smaller networks such as ResNet18, ResNet34. For more extensive networks, e.g., ResNet50 and ResNet101, a three-layer block is used. These blocks represent the calculations above and can be seen in Figure 2.10.

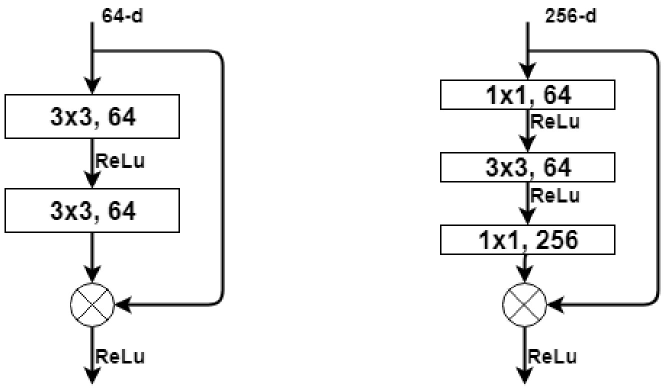


Figure 2.10: Block diagram of residual blocks of depth 2 and depth 3. After every convolutional layer follows Rectified Linear Units (RelU) activation function.

2.3.5 Fully Convolutional Networks

CNN is designed for image classification because of the fully connected layers at the end of the network. Instead, Long et al. (2015) replaced the fully-connected layers in an CNN with convolutional layers. This resulted in a Fully Convolutional Network (FCN). This type of network can train end-to-end to do pixel-wise prediction and is therefore widely used in various image segmentation tasks.

FCN can be divided into two architectures, encoder-decoder and dilated convolution. The encoder-decoder architectures extract semantic information from images by downsampling in the encoder and then upsample in the decoder to recover a full-resolution semantic segmentation map. Contra, the dilated convolution architectures uses dilated convolution that expands receptive fields exponentially such that there is no need for downsampling. Hence, spatial information is kept throughout the network. However, dilated convolution usually have higher memory consumption because of the size of the activation map. This makes it challenging to have a deep neural network with many hidden layers.

FCN combines what and where in an image by using skip connections from different layers of the feature hierarchy. In this way, course high layer features are combined with fine, low layer features. The downsampling process loses some spatial information, but the effect of this loss is reduced with this skip-architecture. This results in performance improvement on metric scores and sharper segmentation. Examples of this type of network architecture is U-net by Ronneberger et al. (2015) and LinkNet by Shvets et al. (2018a).

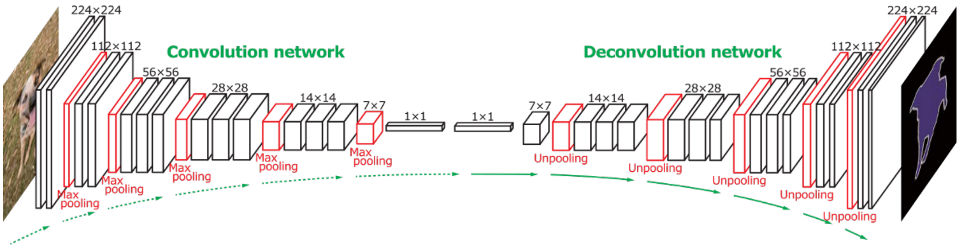


Figure 2.11: FCN network. The convolutional network s based on VGG16-layer net from Simonyan and Zisserman (2014), with a multilayer deconvolution network for pixel-wise class segmentation. Source: Noh et al. (2015).

2.3.6 Region-based CNN

A CNN finds the class of objects in an image, but not the localization. Region-based Convolutional Neural Network (R-CNN) can do both. This type of task is called object detection.

R-CNN was introduced by Girshick et al. (2014). Instead of feeding the entire image to a CNN, they use ”recognition using regions” paradigm. Gu et al. (2009). Object detection using R-CNN consists of tree modules. One module generates Regions of Interests (RoI), one module is a CNN that extracts features from all the generated RoIs, and the last module does object detection using a set of class-specific linear Support Vector Machines (SVM). R-CNN outperformed every other method at that time, gaining a mean average precision (mAP) of 53,3% on VOC 2012, a benchmark dataset created by Everingham et al. (2012).

Figure 2.12. illustrates the R-CNN pipeline. (2) A selective search algorithm generates around 2000 RoIs from the input image. The details of how this algorithm detects RoIs are not presented as it is regarded out of the scope of this report. (3) Each of the 2000 RoIs is warped into a square of size 227×227 pixels, then fed into a CNN. (4) The CNN outputs a 4096-dimensional feature vector, and an SVM of size $4096 \times N$, where N is the number of classes, scores each extracted feature. The SVM also computes four offset values to increase bounding box precision, e.g., when the object is cut in half from the proposed region. For each class, a greedy non-maximum suppression removes RoIs based on their score, so every object have one corresponding bounding box.

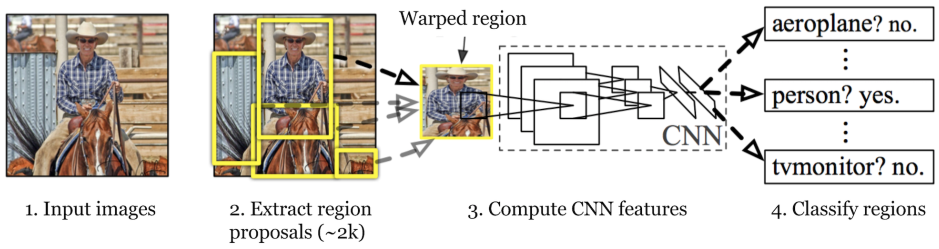


Figure 2.12: R-CNN network architecture. (1) is the input image, (2) RPN extracts RoIs (3) features are computed using a CNN, and (4) classifies features in the image. Source: Girshick et al. (2014).

Two properties improve the efficiency of detection. First, the feature vectors are low-dimensional with only 4096 dimensions. Second, all the CNN parameters are shared across all categories, because the only class-specific computations are dot products between features and SVM weights, and non-maximum suppression. This makes R-CNN scalable for adding more classes, computationally efficient and memory efficient, compared to other algorithms at that time.

Even though R-CNN object detection is efficient for prediction, it has several drawbacks. Training is multi-stage. First, CNN is trained, and then it fits SVMs to the model features. Training R-CNN is time-consuming since, for each input image, 2000 region proposals must be classified. This also demands a significant amount of computer memory. Lastly, object detection is flow. Detection time using VGG16, developed by Simonyan and Zisserman (2014), with GPU is 47s/image.

2.3.7 Fast R-CNN

Due to R-CNN constraints, Girshick (2015) proposed an improved version by introducing a new training algorithm and model architecture called Fast R-CNN.

Instead of feeding every RoI through a CNN, the entire image along with RoI proposals are fed through a CNN. RoI proposals are still generated using selective search.

Figure 2.13 shows the Fast R-CNN pipeline. For every RoI; A RoI pooling layer converts features inside the RoI into a small feature map. This convertion is done by downscaling the RoI to a grid of sub-windows of fixed dimension $H \times W$. The feature map is generated by max-pooling each of the sub-window and extract the max values for each cells. Each feature map is fed into a sequence of fully connected layers that flattens the map into a feature vector. This RoI feature vector is branches into two sibling output layers, one layer predicts the class and one layer generates the offset values for the bounding box.

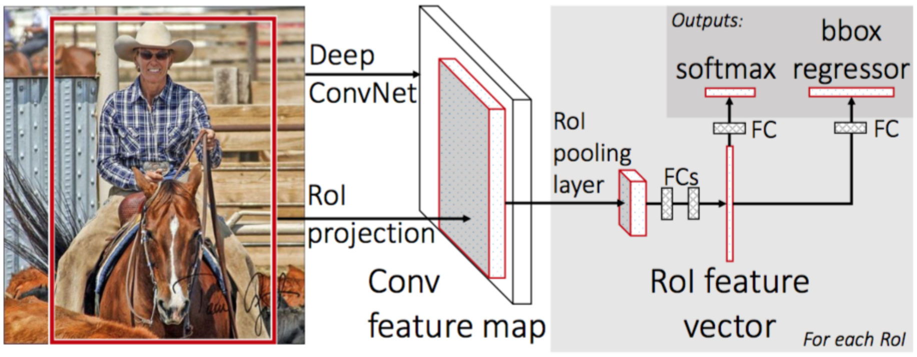


Figure 2.13: Fast R-CNN network architecture. Source: Girshick (2015)

Training Fast R-CNN is done in a single-stage. more efficient than R-CNN because it takes advantages of feature sharing during the training phase. A hierarchical scheme samples stochastic gradient descent mini-batches. RoIs from the same image shares computation and memory in the forward and backward passes. A streamlined training process makes Fast R-CNN training single-stage, with one stage that optimizes both a classifier and bounding box regressor.

A multi-task loss function L is adapted to train both classifier and bounding-box regressor of the network.

$$L\left(p, u, t^{u}, v\right) = L_{\text{cls}}(p, u) + \lambda[u \geq 1] L_{\text{reg}}\left(t^{u}, v\right) \tag{2.10}$$

where

$$L_{\text{cls}}(p, u) = -\log p_{u} \tag{2.11}$$

is the classifier loss, $p = (p_0, \ldots, p_K)$ is the discrete probability distribution for a RoI and u is the true class. L_{reg} is the bounding-box regressor loss, where $t^u = \left(t^u_{\mathrm{x}}, t^u_{\mathrm{y}}, t^u_{\mathrm{w}}, t^u_{\mathrm{h}}\right)$ is the predicted offset values and $u, v = (v_{\mathrm{x}}, v_{\mathrm{y}}, v_{\mathrm{w}}, v_{\mathrm{h}})$ is the true bounding-box offset values. $[u \geq 1]$ is the Iverson bracket.

L_{reg} is expressed as

$$L_{\mathrm{reg}}\left(t^u, v\right) = \sum_{i \in \{\mathrm{x}, \mathrm{y}, \mathrm{w}, \mathrm{h}\}} \mathrm{smooth}_{L_1}\left(t^u_i - v_i\right) \tag{2.12}$$

where

$$\mathrm{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{2.13}$$

Initially, R-CNN uses a L_2 loss function to calculate bounding box errors, but this is more sensitive to outliers than L_1. Therefore, L_1 is used instead.

As a result of these improvements, training Fast R-CNN using the deep VGG16 network is $9\times$ faster, test-time is $213\times$ faster than R-CNN as well as achieving higher mAP on VOC 2012 benchmark dataset by Everingham et al. (2012).

2.3.8 Faster R-CNN

The bottleneck of Fast R-CNN is the region proposals method. The selective search uses at best 2 seconds per image. This makes Fast R-CNN unable to run real time. Ren et al. (2017) further adapted Fast R-CNN to use a Region Proposal Network (RPN) instead of the selective search algorithm to find RoIs. This drastically increased the run-time.

Faster R-CNN is a single, unified network composed of two modules. One module is a deep RPN which propose RoIs. The other module is a Fast R-CNN detector that uses the proposed regions for object detection. Both modules share convolutional layers and features. This makes training easier and speeds up the run-time. Figure 2.14 illustrates the network.

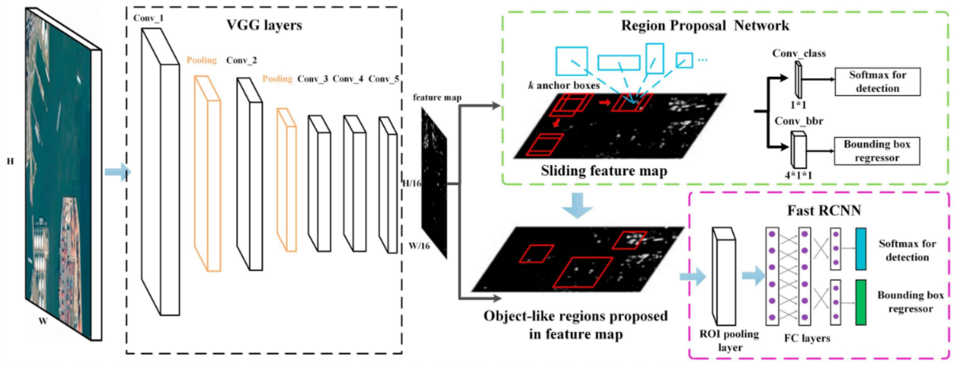


Figure 2.14: Faster R-CNN network architecture. This example uses VGG16 as backbone to extract features. The two modules can be seen in the right of the image. Source: Deng et al. (2018).

The Region Proposal Network is a small FCN. It takes an input of any size and outputs a set of rectangular object proposals as well as a score for each proposal. RoIs is generated by sliding a small network over the convolutional feature map output. Input to this small network is a $n \times n$ size sliding window of the convolutional feature map. Every sliding window is mapped into a lower-dimensional feature before its fed into two 1×1 convolutional layer, a box-classification layer (cls) and a box-regressor layer (reg).

For each sliding window, k anchor boxes are generated. The anchor boxes are centered at the sliding window center and are linked to a scale and aspect ratio. By default, three scales and three aspect ratios are used, resulting in $k = 9$ anchors. This gives $4k$ coordinates as output in the reg layer and $2k$ scores in the cls layer, see Figure 2.15. In general, for a given feature map of size $H \times W$, there are $W \times H \times k$ anchors. One significant property is that the anchors are translation invariant.

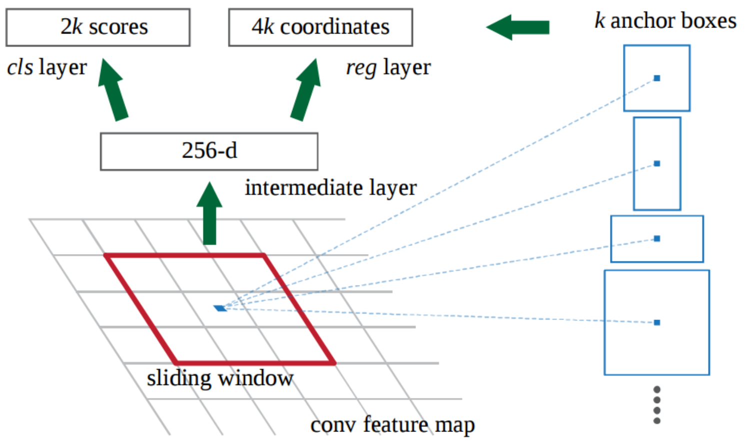


Figure 2.15: RPN at a specific position on the convolutional feature map. Source: Ren et al. (2017).

Anchor boxes are labeled as positive if the anchor have has an IoU overlap higher than 0.7 with any ground truth box., or if it is the anchor with the highest IoU overlap with a ground truth box. Otherwise it is labeled as negative. A single ground truth box can assign positive labels to several anchors.

An objective function similar to the multi-loss function in Fast R-CNN explained in section 2.3.7 is used.

$$L\left(p_i, t_i\right) = \frac{1}{N_{cls}} \sum_i L_{cls}\left(p_i, p_i^*\right) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}\left(t_i, t_i^*\right) \tag{2.14}$$

i represents the index of an anchor in a mini-batch, p_i is predicted probability for the anchor *i* being an object. Ground truth label p_i^* is either 1 or 0 depending if the anchors is labeled labeled as positive or not. L_{reg} is the same as in Fast R-CNN, where t_i is the vector representing the predicted bounding box, and t_i^* is the ground truth box coordinates. $p_i^* L_{reg}$ means that the regressor loss is only activated for positive anchors. Both terms are normalized by N_{cls} and N_{reg} and as in in Fast R-CNN weighted by λ.

A 4-step alternating optimization is used to train Faster R-CNN. First, the RPN is trained using a pre-trained model from ImageNet and fine tune it for region proposals. Second, the output proposals from RPN are used to train a Fast R-CNN detection network. This network is initialized by ImageNet-pre-trained model. Third, the detector network is used to initialize RPN training. The shared convolutional layers are now fixed, and the unique RPN layers are fine-tuned. Last, keeping the shaped convolutional layers fixed, the Fast R-CNN layers are fine-tuned. This process is iterated.

Overall, Faster R-CNN can do object detection in real time, even with large neural networks such as VGG16 as a backbone (5fps). This is because the Regional Proposal Network simultaneously predicts object bounding boxes and objectness scores. It achieves state-of-the-art accuracy on VOC 2012 from Everingham et al. (2012) and MS COCO datasets by Lin et al. (2014b).

2.3.9 Mask R-CNN

He et al. (2017) at Facebook AI Research developed a network for doing instance segmentation called Mask R-CNN. It consits of two stages. The first stage of Mask R-CNN is identical to Faster R-CNN, where a RPN generates RoIs. In the second stage, Mask R-CNN extends Faster R-CNN by including a small FCN to predict binary segmentation masks for each RoI. This runs in parallel with the existing classification and bounding-box prediction. Figure 2.16 shows the network structure. Mask R-CNN outputs class label, box offset, and a binary mask for each RoI.

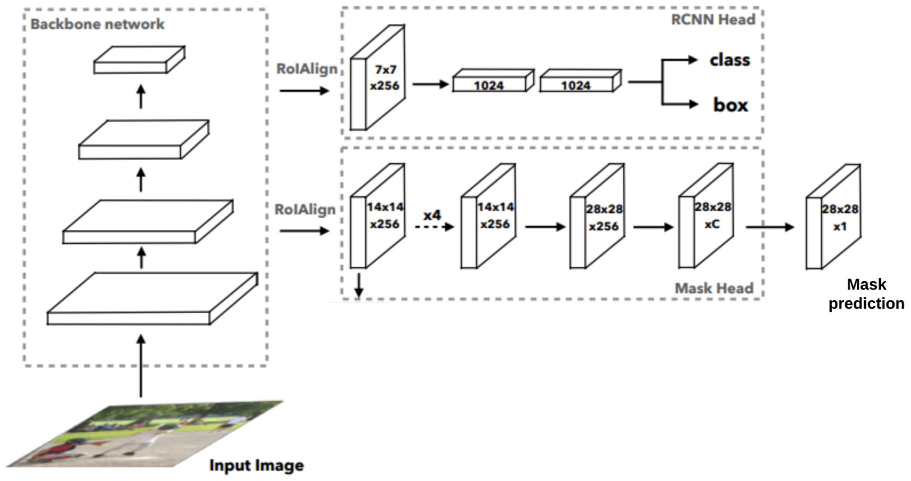


Figure 2.16: Mask R-CNN framework. Image adapted from Huang et al. (2019).

The small FCN predicts an $m \times m$ pixel-to-pixel segmentation mask for each RoI. This mask encodes the object's spatial layout. A property of FCN is that it keeps the explicit $m \times m$ layout without collapsing it into a vector, as is done with class labels and box offsets.

Since Faster R-CNN was not designed for pixel-to-pixel alignment between input and output, a quantization-free layer was developed. This layer is called RoIAlign, and it preserves spatial correspondences between pixels. RoIAlign plays a crucial role in mask prediction.

RoIPool layer, used in Faster R-CNN, introduces a misalignment between RoI and the extracted feature because of quantization. Differently, RoIAlign avoids quantization of RoI boundaries and hence align extracted features with the input. Bilinear interpolation from Jaderberg et al. (2015) computes exact values of the input feature for four sampling points in each RoI bin. The result is then aggregated using max or average. Illustration of RoIalign on an feature man is seen in Figure 2.17.

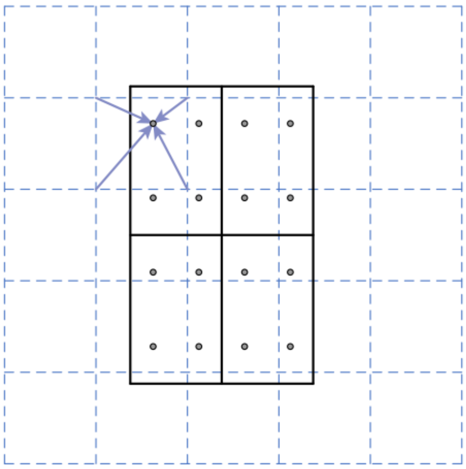


Figure 2.17: RoIAlign illustration on a feature map (dashed grid). An RoI (solid lines) wit 2×2 bins, and 4 sampling points in each bin. By bilinear interpolation from nearby grid points is the value for each sampling point computed. Source: He et al. (2017).

A multi-task loss function on each sampled RoI is minimized during training.

$$L = L_{cls} + L_{box} + L_{mask} \tag{2.15}$$

L_{cls} is the classifier head loss for classification, L_{box} is bounding box refinement loss, and L_{mask} is the mask head loss. L_{cls} and L_{box} is equivalent to Fast R-CNNs loss terms, equation (2.11) and equation (2.12).

L_{mask} is mask head loss. It is defined as the average binary cross-entropy loss.

$$L_{mask} = -\frac{1}{N}\sum_{i}^{N}\left[p(y_i)\log(y_i) + (1 - p(y_i))\log\left(1 - y_i\right)\right] \tag{2.16}$$

where $p(y_i)$ is predicted probability for the pixel being the right class, y_i is the ground truth label and N is all points. Only positive RoIs and the class-specific mask of each RoI contributes to the loss. If there are no positive RoIs, this loss is 0.

2.4 Pose estimation

Zhou et al. (2016) presented a registration method called fast global registration for pose estimation. Fast global registration does not involve iterative sampling, model fitting or local refinement, and is accordingly, able to estimate pose considerably faster than traditional algorithms such as Raguram et al. (2008) RANSAC based algorithm. Zhou et al. (2016) state that fast global registration " *provides the accuracy achieved by well-initialized local refinement algorithms, without requiring an initialization and at lower computational cost.*"

Fast global registration estimates pose T between two point clouds P and Q by optimizing the given cost function

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \rho(\|\mathbf{p} - \mathbf{T}\mathbf{q}\|) \tag{2.17}$$

where $\mathcal{K} = [p, q]$ is the set of pair-to-pair correspondences, This $\mathcal{K}$ is generated using Fast Point Feature Histogram (FPFH) feature. This feature was chosen by the authors because it takes a fraction of a millisecond to compute, and it yields good matching accuracy.

$\rho(.)$ is a robust penalty estimated using a scaled Geman-McClure estimator

$$\rho(x) = \frac{\mu x^2}{\mu + x^2} \tag{2.18}$$

This penalty function is essential for avoiding to sample, validate, prune or recompute erroneous correspondences. Disabling erroneous correspondences from the optimization enables the algorithm to achieve high computational efficiency. An example of good and erroneous correspondences is seen in Figure 2.18.

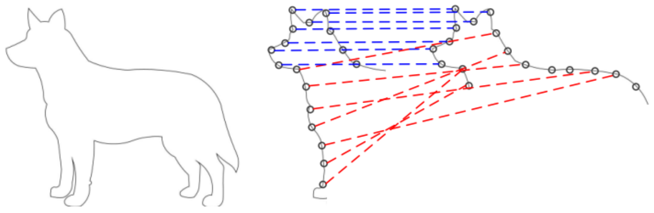


Figure 2.18: Illustration of point-to-point correspondences for a 2D shape. blue correspondences are genuine, red correspondences are erroneous. These are disabled Source: Zhou et al. (2016).

Optimising the cost function 2.17 directly is difficult. Therefore, it is rewritten using the Black-Rangarajan duality. The new joint objective function over pose T and line process $\mathbb{L} = \{l_{\mathbf{p},\mathbf{q}}\}$ is expressed as

$$E(\mathbf{T}, \mathbb{L}) = \sum_{(\mathbf{p},\mathbf{q}) \in \mathcal{K}} l_{\mathbf{p},\mathbf{q}} \|\mathbf{p} - \mathbf{T}\mathbf{q}\|^2 + \sum_{(\mathbf{p},\mathbf{q}) \in \mathcal{K}} \Psi\left(l_{\mathbf{p},\mathbf{q}}\right) \tag{2.19}$$

where

$$\Psi\left(l_{\mathbf{p},\mathbf{q}}\right) = \mu\left(\sqrt{l_{\mathbf{p},\mathbf{q}}} - 1\right)^2$$

The new objective function 2.19 is non-convex. μ weights the prior term against the alignment term. Large μ smooths out the objective function and hence grants more correspondences in the optimizing step. Reducing μ sharpens the shape of the objective function and makes the registration more precise. Graduated non-convexity is used to set μ. It initialized μ with a large value and decreases it during optimization until a threshold value is reached. Figure 2.19 portray this process.

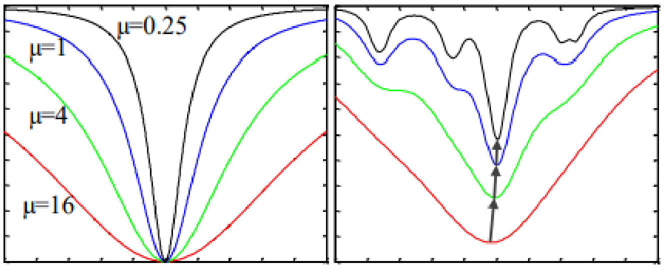


Figure 2.19: Illustration of graduated non-convexity for different values of μ. Right: Geman-McClure penalty estimator, Left: objective function. Source: Zhou et al. (2016).

Optimizing the new objective function 2.19 is done by alternating between optimizing pose T and line process $\mathbb{L}$. This is equivalent to optimizing equation 2.17. When optimizing for T, $\mathbb{L}$ is fixed and vice versa. This is more efficient than optimizing 2.17 directly.

Chapter 3

Related work

3.1 Related Work

After the success of CNNs, have extensive research and development been made with object detection and segmentation in 2D. As a result of cheap and accurate 3D sensors, have reasearcher now lifted neural networks research to 3D space. Converting traditional 2D CNNs to volumetric 3D CNNs is relatively simple, as long as the data is structured. Structuring 3D data can be accomplished by sample it as voxels or using RGB-d images.

Zhou and Tuzel (2017) structures 3D data by dividing it into equally spaced 3D voxels. Each voxel is then transformed to 4D tensors, which are processed by convolutional layers. This end-to-end 3D network detects objects and predicts bounding boxes. Zeng et al. (2016) generates $30 \times 30 \times 30$ voxels from local patches in a point cloud. Bounding boxes and 6D pose is estimated by finding point-to-point correspondences between these voxels from two partially point clouds.

Using 3D CNN is computational heavy as the number of parameters in the neural network is drastically increased by the new dimension. Furthermore, voxel representation does not take the density of a point cloud into account as data is rendered unnecessarily spacious. On the other hand, RGB-D representation utilizes 2D CNN methods that are optimized for speed and performance. More importantly, RGB-D representation makes it possible to take advantages of the massive amount of training data available on the web.

Plenty of methods, frameworks, and neural networks architectures have been developed for segmenting RGB images. Long et al. (2015) revolutionized semantic segmentation with the introduction of fully convolutional network (FCN). Any classification network could easily be adapted to an FCN by replacing every fully connected layer with fully convolutional layers. Badrinarayanan et al. (2017) and Ronneberger et al. (2015) extended the FCN design by adding skip architecture between the encoder and decoder. Feature layer outputs in the decoder are gradually upsampled. Every upsampled output is combined with the corresponding feature output in the encoder. This improves accuracy, reduces number of parameters needed and reduced processing time. Chaurasia and Culurciello (2018)s LinkNet uses this method to bypass spatial information directly from the encoder to the decoder. Hence, no information is lost during the downsampling phase and better segmentation is achieved.

He et al. (2017) extended Faster R-CNN developed by Ren et al. (2017) with a segmentation branch named Mask R-CNN. This network outperformed all existing single-model networks on object detection, bounding box regression and instance segmentation. Many other researchers have later used this as a baseline for their work. Hu et al. (2017) proposed a new partially supervised training paradigm to ease instance segmentation model training. This enables Mask R-CNN to detect thousands of different categories without the need of complete instance segmentation annotations for all categories in the training data.

Danielczuk et al. (2018) trains a Mask R-CNN exclusively on synthetic depth images. Synthetic Depth (SD) Mask R-CNN achieves similar performance to a Mask R-CNN trained on a massive, hand-labeled RGB dataset, then fine-tuned on real images from the experimental setup. This shows that synthetic data, which is much easier and cheaper to generate, can be used as training data, without loosing any performance.

Depth information has gradually been included in segmentation networks as 3D sensors have become more affordable. Depth data provide new information about the scene, which can boost the performance of neural networks. Schwarz et al. (2018) developed a depth fusion method that does object detection and semantic segmentation. Jiang et al. (2018) employs the FCN encoder-decoder architecture, but passe RGB and depth images to two separate network. Then, a fusion structure fuses RGB and depth features over several layers to produce accurate semantic segmentation outputs.

An essential part of many real-world robotic applications involves pose estimation. Neural networks have shown good performance in this field of study as well. Xiang et al. (2017) estimates 3D translation and 3D rotation of objects from RGB images using the objects segmentation mask. Chu et al. (2017) combines RGB with depth images to estimate 6D pose. Two separate deep neural networks extract information from RGB and depth images, and then an image matching framework generates 6D pose used for grasping. They won the Amazon Picking Challenge stowing task in 2017. Wang et al. (2019) proposes a generic framework for 6D pose estimation from RGBD images, without separating RGB and depth images. This increases the performance in highly cluttered scenes.

Chapter 4

Experimental Setup

4.1 Hardware and Software

A test rig was set up to test the two neural networks on a small dataset generated with a Zivid camera. A requirement for taking pictures with Zivid camera is that the computer has a GPU. Zivids own GUI software, Zivid Studio, is used to capture images and change light settings. Figure 4.1 shows the lab setup.

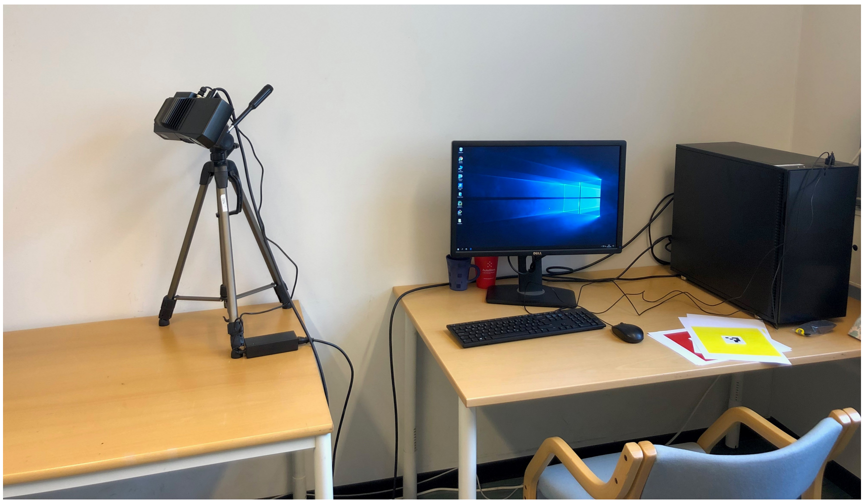


Figure 4.1: Image of the lab setup with Zivid camera on the left and the computer on the right.

4.1.1 Zivid One camera

As mentioned above, is Zivid One camera used for generating test data. Zivid One is an RGB-D camera developed by Zivid. It captures images of size 1920×1200 (2.3M pixels) at 10Hz frame rate with precision down to 0.1mm. The camera uses structured light, where 13 different light patterns are used to generate a structured point cloud of the scene. The optimal range is between 0.6 and 1.1m, with a maximum range of 1.5m.

Zivid camera generates an organized point cloud. For each point, the 3D coordinates (XYZ), color (RGB), and a confidence factor (Q) are captured. The outputted 3D data is a matrix of shape $[1200 \times 1920 \times 7]$, see Figure 4.2.

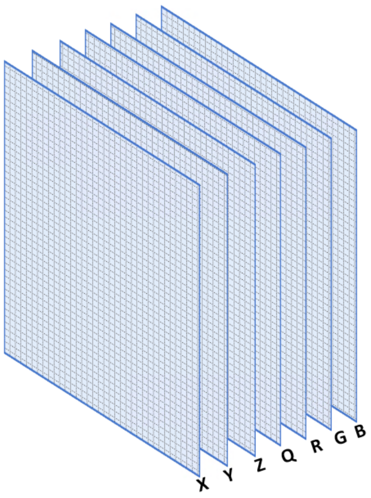


Figure 4.2: Mathematical visualization of the matrix generated by a Zivid camera.

Because the point cloud is organized, evaluating RGB-D images or point cloud is equivalent. The RGB and depth images can be extracted directly from the 3D data by only take the last three layers of the matrix. A pixel at a given location in the RGB-D images corresponds to the point at the same location in the point cloud. This is an important property that enables using 2D machine learning techniques to segment the RGB image, and then use the output masks to segment the 3D data.

4.1.2 Machine Learning Frameworks

Two different deep learning frameworks have been used to build the two neural networks. PyTorch was used for LinkNet34, and TensorFlow was used for Mask R-CNN.

PyTorch is an open-source framework based on tensors developed by Paszke et al. (2017). It has a Python frontend with Torch engine backend, but it is also compatible with C++. It runs on both CPU and GPU, with its own cuda implementation. This makes it especially easy to run on NVIDIA GPUs. PyTorch is dynamic, define-by-run framework, and there is no need for a compiler step. It is dynamic because the computational graph constructed to represent mathematical expressions is dynamic. It is this computational graph that is used to compute gradients. Hence, the user can directly compute the gradient of a particular expression. This also makes debugging more manageable because there is no need to go through two abstractions, the Python code, and the actual computational graph, to detect the bug. This is different from TensorFlow.

TensorFlow was created by Xu et al. (2016) at Google Brain and is another deep learning framework. It has a massive ecosystem of tools, community resources, and libraries, that makes it easy to create and train deep learning models. TensorFlow follows a define-compile-run paradigm. It generates a static computational graph for mathematical expressions, compiles it, and then gradients can be computed. Since the computational graph is static, the gradient cannot be directly computed for a distinct expression. Debugging is more difficult because of this static nature, but in return, it gives more space to optimize the underlying computation.

4.2 Datasets

The data used for training LinkNet34 and evaluate and test the performance of both networks is split up in two types of data; training and synthetic.

Training data

Training LinkNet34 is done using parts of *Warehouse Instance Segmentation Dataset for Object Manipulation* (WISDOM) created by Danielczuk et al. (2018). This dataset was chosen because it contained households objects that were easily accessible for generating real data in the lab. Further, if there were time was the idea of the author to include depth information in the network by adding a new input channel. Thus, this would make the neural network unable to use pre-trained weights.

WISDOM dataset is split up in WISDOM-Sim and WISDOM-Real. Wisdom-Sim contains 50.000 synthetic depth images. WISDOM-Real consists of 800 RGB-D images containing 3849 instances of 50 real objects. All of these objects are commonly found in a household. Only the WISDOM-Real dataset has been used for training and evaluation since pre-trained weights are preferred to use.

The WISDOM-Real dataset is split into a high resolution subset and a low resolution subset, each with 400 images. There is an average of 4.8 object instances per image. For both resolutions, color images, depth images, and segmentation masks are provided, as well as intrinsic camera parameters. Figure 4.3 shows an example of an RGB image, depth image, and corresponding segmentation mask. Every image is of size 772×1032 pixels.

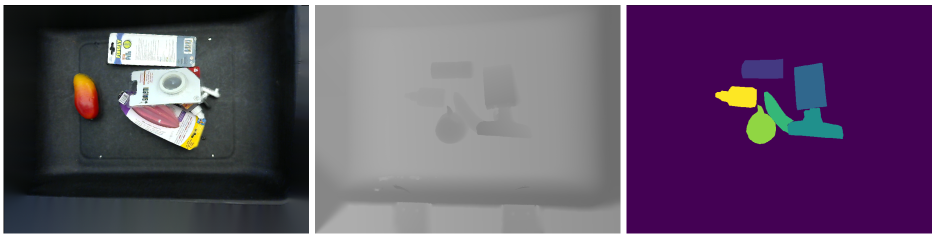


Figure 4.3: Example of WISDOM-Real dataset. Right: RGB image, middle: depth image, left: segmentation mask.

The segmentation masks are ordered such that every object have its own pixel value, e.g., $background = 0, object_1 = 1, object_2 = 2....$ The pixel value are not unique for an instance, it varies for a given object in different masks. Because of this, it is not possible to train a network to do instance segmentation, as the network is not able to understand that an object is the same in several images.

To do instance segmentation, the existing WISDOM-Real dataset is modified such that every object have a fixed pixel value, e.g., the $banana = 3, hammer = 6...$ for every image. As explained in section 5.1 was not LinkNet34 able to do instance segmentation. Hence is the modified WISDOM-Real dataset never used.

Synthetic data

To evaluate pose estimation for instance segmentation is the ground truth pose needed. To the author knowledge, there was no datasets available that had the same instances as WISDOM-Real (Danielczuk et al. (2018)) and COCO dataset (Lin et al. (2014b)) and also offered ground truth poses. Accordingly was a 3D data generator used to create synthetic data. The simulator was originally implemented by Edvard Rauø Vasdal as a part of his master thesis for generating 3D data of boxes. Small modifications enabled the simulator to use point clouds generated from CAD models as inputs. It outputs a point cloud with a corresponding RGB image, depth image, ground truth masks, and ground truth poses (transformation matrices).

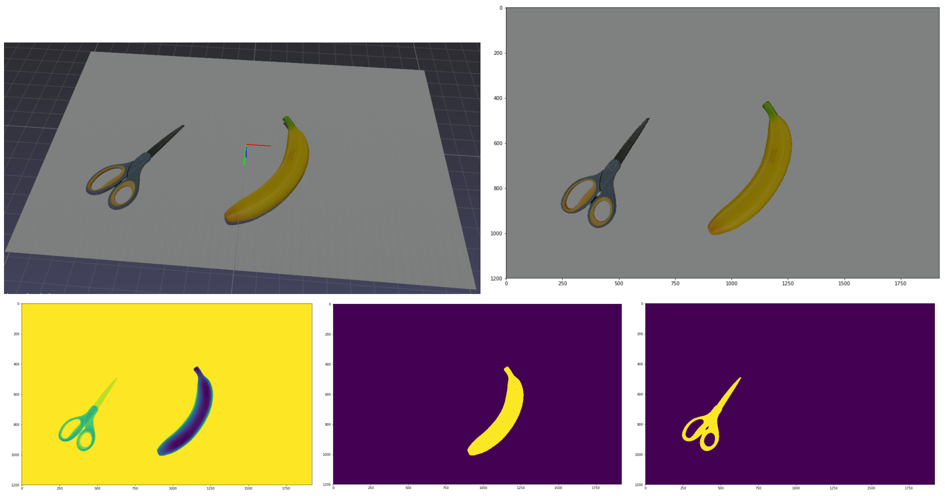


Figure 4.4: Example of output from the simulator. Top left: Point cloud. Top right: RGB image. Bottom left: depth image, Bottom middle: binary mask of first instance, Bottom right: binary mask of second instance.

The synthetic dataset consists of 50 point clouds. Every point cloud has two instances (scissor and/or banana), which are randomly placed in the scene. The scene was set to be 1500 mm away from the camera coordinate system. Each instance is randomly translated between $[-600, 600]$ mm along the x-axis and $[-400, 400]$ mm along the y-axis. A simple rotation around z-axis randomly rotates the objects between $[0, 360]$ degrees. If the second object in a scene overlaps the first object, it is removed, and another object is placed instead. Consequently, there is no occlusion in the dataset. RGB and depth images are created by projecting the scene into an image located at the camera origin using the pinhole camera model. Transformation matrices are generated from translation and rotation. Figure 4.4 shows an example of a set of point clouds, RGB-D images, and masks generated by the simulator.

Chapter 5

Method and implementation

5.1 Overall approach

Much effort has been made in the development of building and training a residual network called LinkNet in PyTorch. The author's idea was that LinkNet should be trained using the WISDOM-Real dataset from Danielczuk et al. (2018) to do both semantic segmentation and instance segmentation.

LinkNet34 was first trained to do binary segmentation, semantic segmentation with two classes. Architecture, implementation and training details of LinkNet34 is presented in section 5.2 and 5.4. Figure 5.1 shows an overview of the binary segmentation process. 3D data is captured by Zivid camera. The RGB image is extracted from this data and used as input to LinkNet34, which outputs a binary segmentation mask. The masks are then used to segment the 3D data into background and object point clouds. The results of this segmentation are presented in chapter 6.

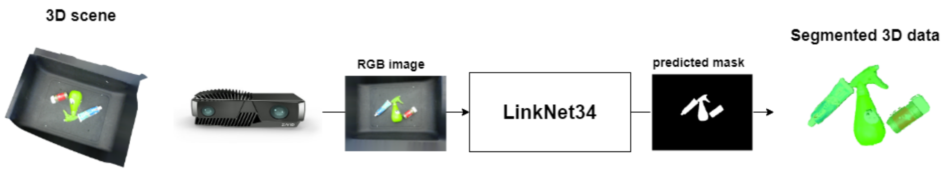


Figure 5.1: Overview of binary segmentation using LinkNet34. 3D scene and RGB image is taken from the WISDOM-Real dataset. Predicted mask and segmented 3D data is the results of LinkNet34s prediction in this particular case.

When LinkNet34 later was trained for instance segmentation on the modified WISDOM-Real dataset, it failed to produce any results. The limited dataset was too small to even partially detect a single object. Therefore was decided the author that this network should only be used for binary segmentation.

Instance segmentation is necessary to be able to estimate pose and hence complete the second goal of problem statement presented in 1.2. Therefore is a pre-trained Mask R-CNN implemented on TensorFlow by Abdulla (2017) is used for instance, segmentation. Architecture and implementation of this network are presented later in this chapter. Figure 5.2 gives an overview of instance segmentation pipeline. Note that instead of predicting a single segmentation mask, outputs the network a segmentation mask for each instance.

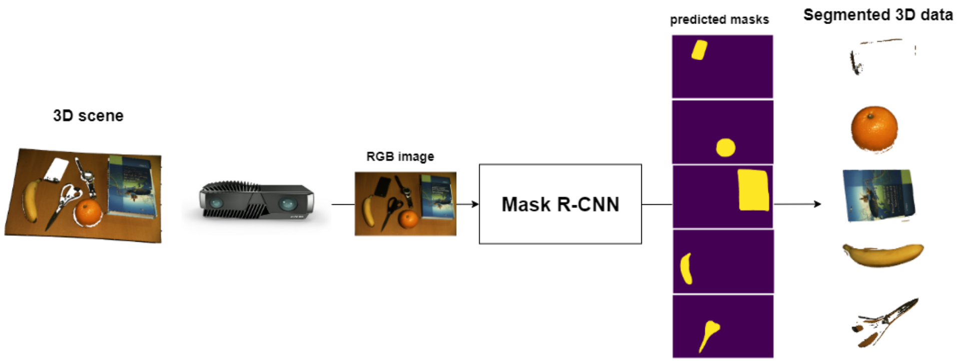


Figure 5.2: Overview of instance segmentation. Predicted masks and segmented 3D data are actual results from the given 3D scene.

Segmented 3D data outputted from Mask R-CNN is combined with matching point clouds generated from CAD models to estimate pose. Mesh models can also be used for generating the corresponding point coud. Figure 5.3 illustrates this process. The registration method used is fast global registration by Zhou et al. (2018). The implementation of this is explained in section 5.4.

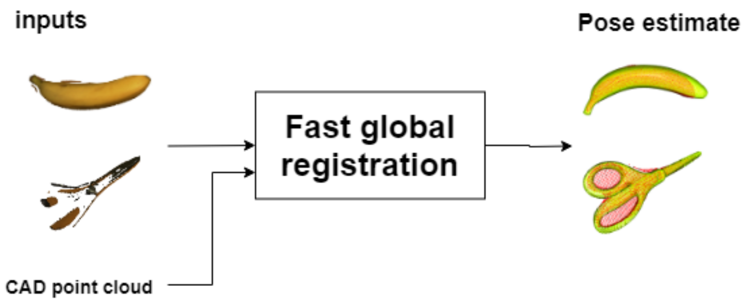


Figure 5.3: Overview of pose estimation approach. Input to the registration method is a segmented point cloud and a corresponding point cloud extracted from a CAD or mesh model.

5.2 LinkNet34

The LinkNet34 was chosen as segmentation model architecture, as it utilizes network parameters efficiently to learn features. It is a small network with 21.3 million parameters. Nevertheless, it achieves state-of-the-art performance at low computational cost and run-time. Several teams have used the same architecture in semantic segmentation competitions on Kaggle such as in medical instrumental segmentation by Allan et al. (2019).

LinkNet34 is a modified version of the original LinkNet developed by Chaurasia and Culurciello (2018). The only modification is that ResNet34 is used as backbone instead of the lighter network ResNet18.

5.2.1 Architecture

LinkNet34 follows a FCN architecture with a downsampling phase using an encoder and an upsampling phase using a decoder. By using skip connections between the encoder and the decoder is spatial information, that can be lost during the downsampling phase, recovered. Besides, the decoder can use fewer parameters since the encoder is sharing knowledge at every layer. This results in an overall more efficient network compared to existing many state-of-the-art segmentation networks.

The encoder is a residual network with 34 layers, explained in section 2.3.4. ResNet34 was chosen as backbone because it is a lighter network compared to contemporary networks, e.g., VGG16 and ResNet101. Nevertheless, in "An analysis of deep neural network models for practical applications" by Canziani et al. (2016), it gives satisfactory performance and run-time, and demands little computational power.

The decoder consists of 4 decoder blocks. Each decoder block starts with a 1×1 convolutional filter, followed by batch normalization. This reduces the number of filters by 4. Then 4×4 transpose convolution upsamples the feature map before another batch normalization layer is applied. Lastly, come another 1×1 convolutional layer and batch normalization. Between every decoder block is the output summed with a skip connection from the encoder.

After the last decoder block, two layers of regular convolution is applied, which reduces the number of channels/filters from 64 to the desired number of classes.

The whole architecture can be seen in Figure 5.4. The entire downsampling part is ResNet34. This architecture is shown in Figure 5.5. The first layer is a 7×7 convolutional layer followed up with a max pooling layer. Both these layers have a stride of 2. Then, 16 Residual blocks with depth 2, see Figure 2.10 for illustration, follows with ship connections after residual block 3, 7 and 13. Usually is the last layer of a segmentation neural network a softmax layer that turns the class prediction for every pixel into probabilities that sums to one. Because a softmax function is included in the loss function, there is no need for it in the network.

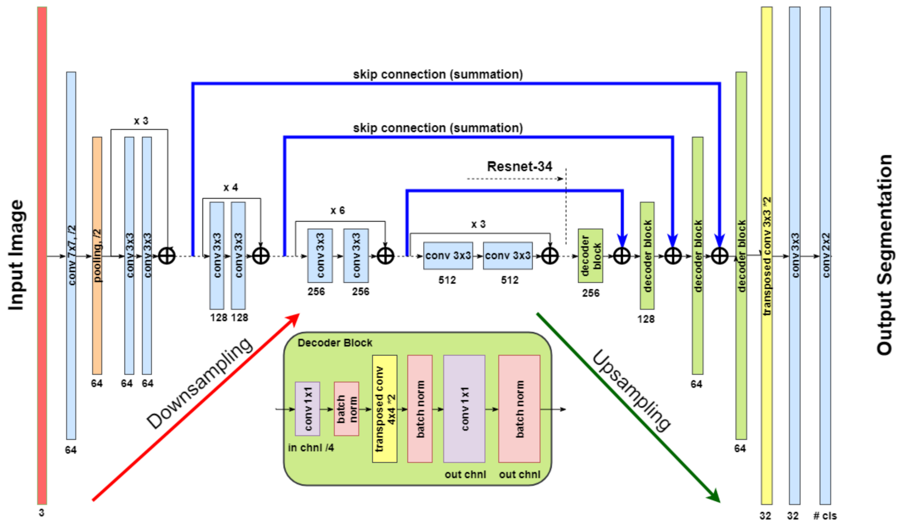


Figure 5.4: Model architecture for LinkNet34 with ResNet34 as encoder. Source: modified from Shvets et al. (2018b)

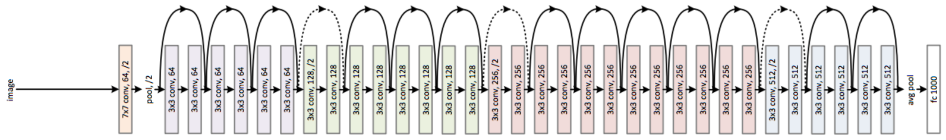


Figure 5.5: Model architecture ResNet34. In LinkNet34 is the last fully connected layer and average pooling removed. Source: He et al. (2015).

PyTorch offers a library with basic layers for neural networks, such as convolutional layers, ReLU, and batch normalization layers. From these are the decoder blocks, shown in Figure 5.4 build. From PyTorchs model zoo is ResNet34 extracted with its pre-trained weights. The different encoder layers in LinkNet34 is set to corresponding layers in ResNet34. Pre-trained weights are used to initialization encoder weights before training the network.

5.2.2 Loss function

PyTorch offers multiple built-in loss functions ready to use. A common function to use for multi-class segmentation problems is cross entropy loss. Cross-entropy loss combines logarithmic softmax function with negative likelihood loss to train a classifier for every class. Input parameters are an input tensor, target tensor, and the function outputs an output tensor with the same size as the target. The input tensor is raw unnormalized scores for each class. Its shape is $(minibatch, C, d_1, d_2)$, where C is the number of classes and d_1 and d_2 are the size of the image. For each of the C classes, the tensor contains scores for each pixel represent the probability of the specific class. The target input is of shape $(minibatch, d_1, d_2)$, where each value is $0 \leq \text{targets}[i] \leq C - 1$. Here, i is the index of the minibatch.

For a given tensor x and a class, the cross-entropy loss function is described as

$$\text{Loss}(x, \text{class}) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right) \quad (5.1)$$

The loss function calculates the loss for every RGB image in a minibatch and outputs the sum of loss for the entire batch. The reason for summing the losses for every image is due to the fact that a typical image in WISDOM-Real dataset is mostly covered with background pixels. Therefore, if the mean output were to be used, would a predicted mask labeled as background for every pixel results in a low loss. This can potentially slow down the training.

5.2.3 Optimizer

Several different optimization algorithms are available in PyTorch. All of them are based on gradient descent, and they are split up in two categories depending on if the learning rate is adaptive or not.

The challenge with vanilla gradient descent, methods with constant learning rate, is that it is difficult to choose a proper learning rate. If the learning rate is too low, convergence is slow, while too large can stop convergence and in the worst case lead to divergence. Adaptive methods solve this problem by adjusting the updates for each parameter depending on their importance.

In the article ”An overview of gradient descent optimization algorithms” by Ruder (2016), the author suggest using an adaptive learning-rate method because there is no need to tune the learning rate manually and it achieves good results fast. Of all the adaptive learning-rate methods, the article concludes that Adam optimizer algorithm is the best overall choice.

Adam or Adaptive Moment Estimation was presented by Kingma and Ba (2015). It combines the advantages of two other stochastic gradient descent methods, namely Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). Adam computes adaptive learning rates for different parameters from the estimate of the first (the mean) and second moments (the variance) of the gradient.

Mathematically this can be expressed as,

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\, g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2 \end{aligned} \tag{5.2}$$

where m_t is the mean and v_{t-1} is the variance.

Due to initialization of β_1 and β_2, the estimates above are biased. This is counteracted by computing by calculating the bias-corrected estimates

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{5.3}$$

These bias-corrected estimates updates the model parameter, θ_{t+1},

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{5.4}$$

Here, β_1 and β_2 are the decay rates and η is the learning rate.

The recommended parameters from the Adam paper are $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. This is the same parameters that are used as default in PyTorch and hence in the experiment. The Adam optimizer algorithm in PyTorch performs a single optimization step based on the loss.

5.3 Mask R-CNN

An existing Keras and TensorFlow implementation of Mask R-CNN by Abdulla (2017) have been used as network for instance segmentation. Mask R-CNN is a more complex model than LinkNet34 and therefore, able to detect more features in an image.

Their implementation deviates from the original implementation by He et al. (2017) in three cases to make the code more general.

- Input image size must be dividable with 2 six times to avoid fractions during the downsampling and upsampling phase. All images are resized to 1024×1024 pixels by padding the image with zeros. The aspect ratio is preserved. This enables the use of multiple images per batch. The paper resizes every image to 800×1000 pixels.
- Ground truth bounding boxes are computed from ground truth masks instead of using bounding box coordinates given in some dataset. The smallest box that encapsulates the entire mask of an object is the ground truth bounding box for that mask. This makes the model able to handle dataset regardless if it provides bounding boxes or not. It also simplifies the implementation of rotation, resizing, and

cropping bounding boxes. New bounding boxes are calculated from the transformed mask instead of computing a bounding box transformation and then apply it to the original bounding boxes.

- The learning rate is changed from 0.02 to 0.001 because it caused weights to explode.

The input to the network is an RGB image of any size. The output is a dictionary with RoIs coordinates (x1, y1, x2, y2), class id, scores and mask for each RoI, as well as computational time.

5.3.1 Architecture

As explained in 2.3.9, Mask R-CNN architecture is divided into two architectures, backbone, and head.

The backbone architecture consists of a convolutional network and a Feature Pyramid Network (FPN). These extracts feature and anchors from the entire image. Since LinkNet34 was not able to segment instances with a ResNet34 network as backbone, is ResNet101 used as backbone for Mask R-CNN.

ResNet101 starts with the same architecture as ResNet34, with a convolutional and pooling layer, both with stride 2. For every convolutional layer comes a batch normalization layer. Then, five stages of downsampling follow. The identity block is the three layers deep block explained in 2.3.4. The convolutional block is a modified version of the identity block, where the shortcut connection has a convolution and batch normalization layer instead of identity mapping.

1. Stage 1: 7×7 convolutional layer, batch normalization, Relu activation layer, and a 3×3 max pool layer
2. Stage 2: one convolutional block, $2\times$ identity blocks
3. Stage 3: one convolutional block, $3\times$ identity blocks
4. Stage 4: one convolutional block, $22\times$ identity blocks
5. Stage 5: one convolutional block, $2\times$ identity blocks

Feature Pyramid Network is build up with five pyramid levels, where each level is matched with a feature map for the five stages in ResNet. A convolutional layer is added to all pyramid levels to get the final feature map. The last pyramid level is only used in Region Proposal Network, but not in the classifier heads. FPN generates square or rectangular anchors that cover the full image at different scales. For an input image of 1024×1024, it generates around 200k anchors. The order of anchors are important, and therefore are they sorted by the pyramid levels, and for each level, the anchors are sorted by feature map processing sequence, e.g., from pixel $(0, 0)$ to pixel (x_{max}, y_{max}).

The head architecture consists of an RPN, classifier head, bounding box head, and mask head architecture. RPN is a small fully convolutional network with two convolutional layers. It divided into two stages. Stage 1 loops through the pyramid levels, classifies every anchor as a positive, neutral or negative anchor, and returns an object/no-object score. Only the 1000 (2000 for training) best anchors are accepted as to stage 2.

Stage 2, a proposal layer, classifies the accepted anchors with a class ID, a confidence score, and generates bounding boxes. Positive RoIs are regions of interests that do not have a class id set to background. It filters anchors based on anchor scores and applied non-maximum suppression to remove anchors which overlap. Bounding boxes are shifted based on calculated offset values, and clip bounding boxes that are outside the image boundary. This layer returns proposals in normalized coordinates as RoIs.

A PyramidRoIAlign layer extracts RoIs outputted from RPN at every pyramid level and sends it to a detection layer for refinement. For an RoI to be a detection, it must have a confidence score $>= 0.7$. Low confidence detections are filtered away based on a min confidence threshold. Per-class non-maximum suppression removes detections of same classes that overlay each other in an image. Bounding box transformation is applied to RoIs to shift it based on a bounding box standard deviation. Only the top 100 detections in an image are used for mask segmentation.

The mask head is a small FCN consisting of a PyramidRoIAlign layer, four convolutional layers with corresponding batch normalization and Relu activation layers, a transpose convolution, and another convolutional layer. For every instance in the image with a bounding box and class ID, it generates a binary mask. To save memory and computational time are only the masks pixel that is inside the bounding box stored instead of the full image with masks.

5.3.2 Loss Function

Mask R-CNN has a general network architecture. Several different models can be used as *backbone* for feature extraction and as *head* for bounding-box recognition. The paper uses 50 and 101 layer ResNet by He et al. (2015) and ResNeXt by Xie et al. (2017), and a Feature Pyramid Network (FPN) by Lin et al. (2017) as *backbone*. The same architecture as in Faster R-CNN is used as *head*, but the head is extended to add a mask branch.

The same loss function that was presented in the paper He et al. (2017) and explained in section 2.3.9 are used, and has been adapted to use TensorFlow libraries.

For calculating L_{cls} is a sparse softmax cross-entropy with logits used. This loss function is designed for single-class labeling, which fits nicely because each RoI is linked to one class. Only active classes, classes that are presented in the image, contributes to the loss. Equation 5.5 shows how the loss equation for K categories is designed in the implementation, where u is ground truth and p is the network prediction.

$$L_{cls} = -\sum_{K} u_K \log p_K \tag{5.5}$$

Bounding box refinement loss, L_{box}, is calculated using the smooth L1 loss function from section 2.3.7, equation 2.12. Only positive RoIs, and right class IDs for each RoI contributes to this loss.

L_{mask} is implemented using binary cross-entropy loss function. This is implemented in the same way as it is explained in equation 2.16 in section 2.3.9. Only positive RoIs and the class-specific mask of each RoI contributes to the loss. Same functions as If there are no positive RoIs, this loss returns 0.

The Regional Proposal Layer also has two loss functions. The first loss function is an anchor classifier loss. It uses the cross entropy loss in equation (5.5). Only negative and positive anchors contribute to the loss. The second loss function is a bounding box loss. The same loss function used for bounding box refinement, equation (2.12) is used for this. Only positive anchors contribute to the loss.

5.3.3 Optimizer

Stochastic Gradient Descent (SGD) with momentum is used to optimize weights and train the network. The momentum is set to 0.9 and the learning rate to 0.001. Including a momentum accelerates SGD in the appropriate direction and dampens oscillations.

5.4 Implementation

5.4.1 Data loading

Images taken with Zivid camera are stored in NetCDF4 format. As mentioned in section 4.1.1, outputs the Zivid camera a matrix of size $1200 \times 1920 \times 7$. Zivid data is loaded using dataset function from NETCDF4 library. The entire point cloud (XYZQRGB) and RGB image is extracted as arrays.

WISDOM-Real data is loaded by overwriting an abstract class dataset available in PyTorch. The overwritten class is called *WisdomLoader*. It takes in the root folder to where the training color images, depth images, and masks are located, and generates a dataset. This dataset is split up in a training dataset and a validation dataset, with a train/validation ratio of 0.8. A PyTorch iterator provides both training and validation datasets to load images in parallel, batching the data and shuffling it. It is this iterator that passes a batch of corresponding RGB-D images and masks to the network for training and validation.

5.4.2 Pre-processing of data

Point cloud pre-processing

After the point cloud is loaded, is redundant points removed. The contrast value Q are removed as they are not needed, and hence is the shape of the point cloud reduced to $1200 \times 1920 \times 6$. Some points in the point clouds do not have depth value because they was occluded in the 3D scene when the data was generated. These points are filled with NaN, not a number, by Zivid software. Some third library functions are not able to understand NaN values, and they are set to 0 to keep the orderness of the point cloud.

Training data augmentation

Data augmentation is applied to data loaded by *WisdomLoader*. Data augmentation makes the network more robust to different translations and illuminations. As augmented data also "increases" the dataset since it seen as new data from a neural network point of view.

Rotation, horizontal flip, and brightness changes are randomly applied to data in *Wisdom-Loader* before it is passed to LinkNet34. For each of the three augmentations, there is a 0.3 chance of modifying a color image, depth image, and mask with the given augmentation. Multiple changes can be applied to the same data, but it is important that the same set of augmentations are applied to both color image and mask. Otherwise, will not a binary mask correspond to its equivalent image and the network will not be able to learn. Figure 5.6 illustrates the different augmentations.

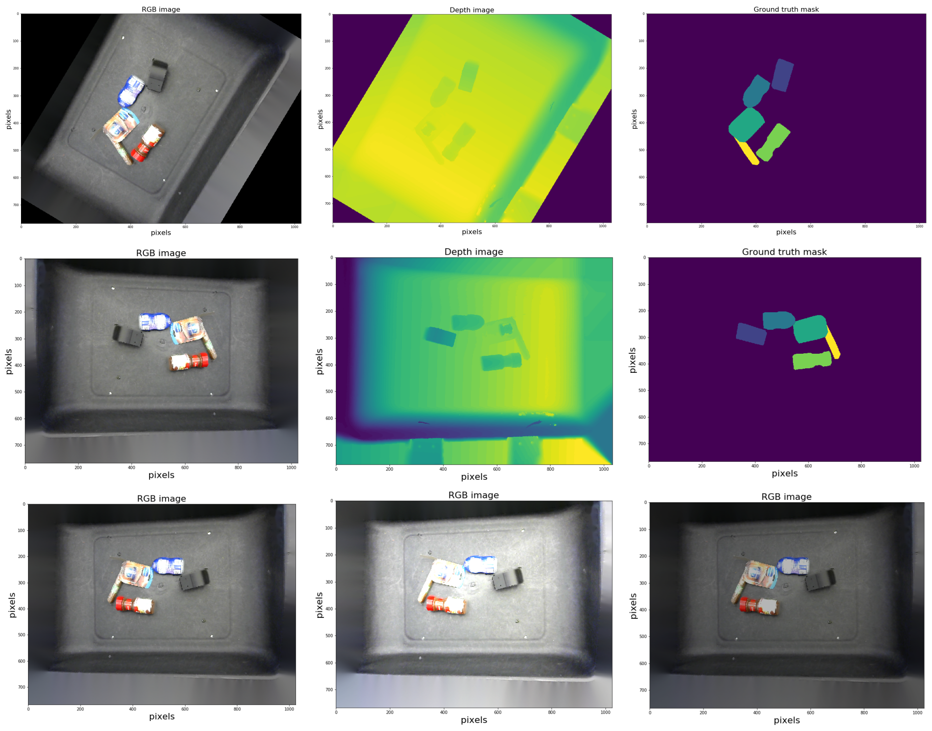


Figure 5.6: Example of rotation (top row), flip (middle row) and brightness changes (bottom row) to data from WISDOM-Real dataset. Brightness changes are only applied to the RGB image.

Image crop and normalization

LinkNet34 demands an input size that is dividable by 64, because of the downsampling phase in the encoder. Therefore are the input images center cropped to size 768×1024 before it is passed to the network.

Normalizing RGB image makes the training less sensitive to feature scales and makes the network learn faster. Every RGB image is normalized such that the mean value for each input channel, $R_{mean} = B_{mean} = G_{mean} = 0$ and standard deviation, $R_{std} = B_{std} = G_{std} = 1$. Normalizing a channel c with mean μ and standard deviation σ is done using

$$c_{new} = \frac{c_i - \mu}{\sigma} \tag{5.6}$$

5.4.3 Training details

The training was done mostly during the nighttime with different model parameter settings. Changes were made to the number of epochs, batch size, and learning rate. Input images are resized from 768×1024 to 480×640 for faster training during turning of parameters. When evaluating with Zivid data, the ideal image resizes would be 1216×1920 (only losing 4 pixels in height). This demands the 20GB of allocated memory. Because of this is Zivid data center cropped to 768×1024.

Model hyperparameters were chosen to be

Parameters	**Description**	**Value**
$\mathcal{L}$	loss function	cross entropy with sum
Solver	solver	Adam
a	learning rate	$1e-4$
m	momentum	0.9
b	batch size	8
Epochs	Number of trained epochs	20

Table 5.1: Hyperparameter setup for LinkNet34

Validation and losses

The loss for every sample is collected to see that the network is training properly.

During training is cross-entropy loss, explained in 5.2.2, used to calculate sum loss for every minibatch. The loss value is between $[0, inf)$. For every second minibatch is the sum loss saved to a log-file along with the corresponding epoch and batch.

Validation loss is calculated for every image in every batch during the validation phase. Both cross-entropy loss and IoU is calculated and saved to the same log-file as training loss as well as their corresponding batch and epoch. The IoU is between $[0, 1]$, where 1 means complete overlap between predicted mask and ground truth mask.

Training pipeline

Training a neural network follows the step explained in 2.3. PyTorch have two modes for neural networks model, namely *train* and *eval* mode. The model mode is set to *train* for every new epoch. This is important because it tells the model to learn new features and activate layers such as dropout and batch normalization.

The training data is loaded with *WisdomLoader*, augmented, and passed to the network. The model predicts a binary mask based on its weights. Loss is calculated between the predicted mask and ground truth using the loss function from 5.2.2.

Gradients are calculated though backward pass. PyTorch accumulate gradients, so before it is calculated for every weight, it is set to 0.

Last, Adam optimizer, explained in 5.2.3, updates the weights using the calculated gradients. All loss values from the loss function are extracted and used for calculating the average loss for every batch and total training loss.

After each epoch is an evaluation phase. This estimates how well the model is performing and calculates validation loss and IoU scores. The model is set to *eval* mode. This tells the model that no new features are to be learned. The evaluation pipeline is similar to training pipeline, except for the optimizer step, no weights are updated in this mode. The evaluation runs on a validation dataset. This dataset is unseen to the model and has not been used for training. Estimating model performance is done by calculating IoU score for every batch. To check for overfitting are the average loss in training compared to the average loss in evaluation.

Model weights, training losses, and IoU losses are saved after each finished epoch as a backup. This makes it possible to pause and resume training without losing progress. Logging losses also gives insight into training performance and efficiency.

5.4.4 Post processing of data

Both neural networks output segmentation masks that are used to segment a point cloud. LinkNet34 outputs a single binary mask for all the objects in an image, while Mask R-CNN outputs a binary mask, score, and label for every instance. Hence, LinkNet34 are only able to separate between foreground and background, whereas Mask R-CNN is able to segment background from objects and even instances of the same object. A binary mask has the same size as the RGB and depth images, but all the values are either 0 or 1.

3D point cloud generation from RGBD images

Pinhole camera model is used to construct a point cloud from the color and depth images in the WISDOM-Real dataset. A 3D point xyz is created from image coordinates (u, v) and a depth value d using,

$$x = \frac{(u - c_x)z}{f_x} \qquad y = \frac{(v - c_y)z}{f_y} \qquad z = \frac{d}{\text{depth scale}} \tag{5.7}$$

f_x, f_y and c_x, c_y are the focal lengths and principal point coordinates. These are found in the camera intrinsic matrix.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1105.0 & 0 & 511.5 \\ 0 & 1105.0 & 384.0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.8}$$

Equation, 5.7 shows that modifying the depth image is equal to segment the point cloud. Pixels with a value of 0 will not generate a 3D point. Applying the binary mask on top of depth image, and only extract pixels where the segmented mask has values > 0, results in a segmented point cloud. The same mask must also be applied to the color image since color image, and depth image must have the same shape for 3D reconstruction. Figure **??** presents an example of 3D reconstruction using the equations above.

For test data taken with a Zivid camera is there no need to generate it through a camera model and intrinsic parameters, as the point cloud is already available. Changing the RGB or depth layer in the matrix, see Figure 4.2, directly affects the point cloud. Therefore, using the outputted masks from the neural networks to segment the RGB images directly segments the point cloud.

Point cloud segmentation

Mask R-CNN outputs N instance binary masks. Each mask can segment the original point cloud to acquire only the points of the desired object. This segmentation is done by iterating over every dimension (X,Y,Z,R,G,B), see Figure 4.2, and extract the values where the segmented mask is > 0. This would not be doable if the NaN points were removed in the pre-processing as the point cloud would not have been organized, and the size of the binary masks and point cloud wound not correspond. After segmentation are the shape of the segmented point cloud changed to # of points $\times$ 6, an unordered structure.

After extracting the segmented point cloud, points that have 0 depth is removed. This process is similar to the masking process, where a function iterates over X, Y, Z, R, G, B columns and removes points where $depth = 0$.

Instead of having N different binary masks, are they collected in one mask with a unique pixel value. The background is defined as 0, while the objects have specific values given from the COCO dataset from Lin et al. (2014b), e.g., book=74, scissor=77, and cup = 42. For visualizing the full instances mask are a random color generator used to generate N different RGB values, one for each instance.

Centroid calculation

To determine where the to initial the point cloud alignment is the centroid calculated.

The mean $\overline{x}, \overline{y}, \overline{z}$ is calculated from

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{5.9}$$

$$\overline{y} = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{5.10}$$

$$\overline{z} = \frac{1}{N}\sum_{i=1}^{N} z_i \tag{5.11}$$

where N is the total number of points in the 3D case.

Finding the centroid in a mask in 3D are the x, y, and z values extracted directly from the segmented point cloud. The mean is calculated from these points using equation 5.11. The same pixel points and 3D points are used in pose estimation using Principal Component Analysis.

5.4.5 Pose estimation

The pose of segmented objects is estimated by aligning the segmented point cloud with a point cloud generated from a corresponding CAD or mesh model. This is done using fast global registration.

Typically, pose is estimated through a two-step pipeline consisting of a coarse alignment,e.g., as RANSAC method from Raguram et al. (2008), followed up by an iterative local refinement, like ICP created by Pomerleau et al. (2013). This pipeline has significant drawbacks. Both alignments perform computationally expensive iterative tasks in their loops. Most of the computations are done on alignment candidates that are later on discarded.

Fast global registration method presented by Zhou et al. (2016), is a single stage algorithm for pose estimation that does not involve iterative sampling, model fitting, or local refinement. It does not require a proper initialization, only that the 3D surfaces are overlapping. This is both computationally and time-efficient. Tested on synthetic data, RANSAC used 2.54 seconds on a coarse registration for one pose estimation. The equivalent using fast global registration took 0.182 seconds. For these reasons is fast global registration used as registration method for pose estimation. The theory behind this algorithm is presented in 2.4.

Fast global registration aligns a point cloud generated from a CAD model or mesh with the segmented point cloud. This is why instance segmentation needs to be done before pose can be estimated. 3D data software Open3D have fast global registration included as part of its library. This is used for implementing the pose estimation method.

Both the segmented point cloud and the CAD point cloud is downsampled as a pre-processing step using voxel downsampling. Each voxel has a size of 4 mm. From the downsampled point clouds are vertex normals estimated and Fast Point Feature Histograms (FPFH) feature calculated for each point. This is a 33-dimensional vector describing the local geometric property of a point. The CAD point cloud is then translated to the centroid of the segmented point cloud. This is done because fast global registration requires partially overlapping 3D surfaces.

Fast global registration computes the transformation between the two downsampled point clouds to best align them. The pose estimate is the dot product between the initial transformation and the alignment transformation.

5.5 Evaluation metrics

5.5.1 Segmentation metrics

For semantic and instance segmentation, it is common to measure the pixel-level classification accuracy of the segmentation. Every pixel in an image is assigned a class label and compared to the ground truth class label. The pixels are stored in a confusion matrix C. Element C_{ij} is the number of pixels with ground truth class i, but have been labeled class j. The confusion matrix is used to label predicted mask as true positive (TP), false positive (FP) or false negatives (FN), depending on an overlap threshold between ground truth and segmented mask.

Intersection over Union

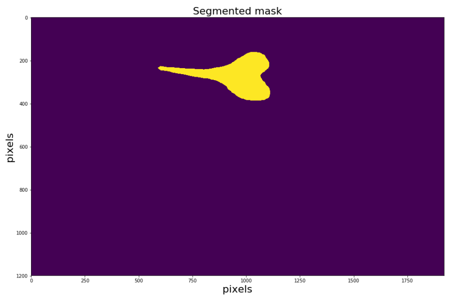
To evaluate model performance is the evaluation metric Intersection over Union (IoU), also called the Jaccard index, used. IoU measure the accuracy of a predicted binary mask versus ground truth mask. It is one of the defacto evaluation metrics for semantic image segmentation according to Csurka et al. (2013). This evaluation metric was chosen because it is unlikely that a prediction is going to match the ground truth at pixel-to-pixel level exactly. IoU does not measure exact pixel-to-pixel correspondences. Instead it measures the overlap between the predicted binary mask and ground truth mask.

IoU is expressed as

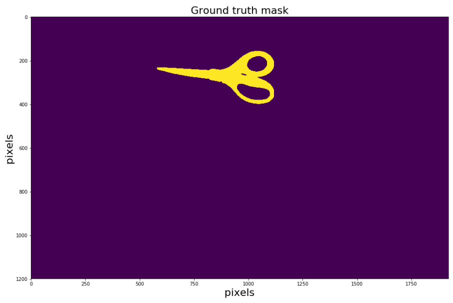
$$IoU_i = \frac{|\text{target} \cap \text{prediction}|}{|\text{target} \cup \text{prediction}|} = \frac{\mathrm{TP}_i}{\mathrm{TP}_i + \mathrm{FP}_i + \mathrm{FN}_i} \tag{5.12}$$

Here, $|\text{target} \cap \text{prediction}|$ is the intersection and $|\text{target} \cup \text{prediction}|$ is the union.

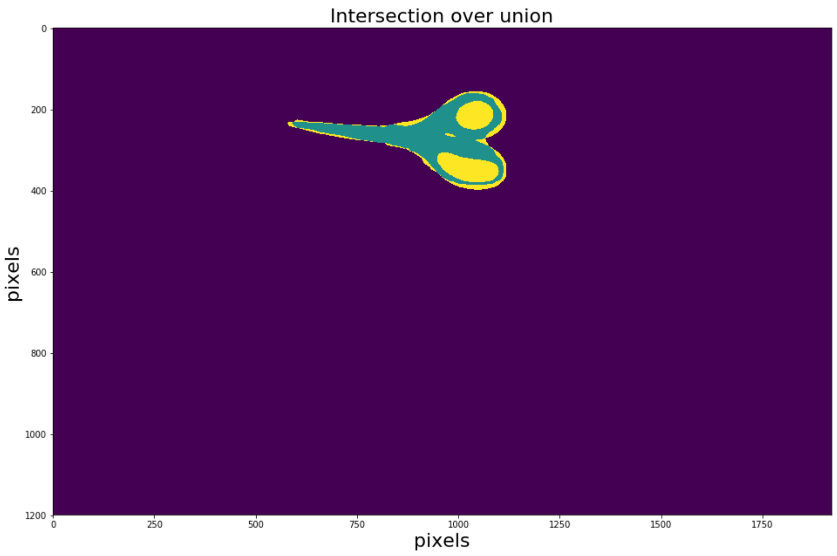
The range of IoU is between 0 and 1, where 1 is complete overlap between ground truth and estimated mask. Values above 0.5 are considered a good match Lin et al. (2014a). Most results are reported in terms of IoU. Figure 5.7 shows an example of ground truth mask and segmented mask outputted from Mask R-CNN.



(a) predicted mask from Mask R-CNN



(b) Ground truth mask from synthetic data



(c) Intersection over Union.Green pixels shows overlapping regions (intersection) and yellow pixels show the union between the two masks.

Figure 5.7: Visualization of predicted instance mask (left), the corresponding ground truth mask (right) and the IoU between them (bottom).

5.5.2 Pose estimation metrics

To easier evaluate pose estimation is it divided into translation and rotation with their own evaluation metric.

Euclidean distance

Measuring error in translation is done using Euclidean distance, also called L_2 norm. This is the straight-line distance between two points in Euclidean space.

The euclidean distance d between estimated translation vector $\hat{T}$ and ground truth translation vector T is

$$d = \|\hat{\mathbf{T}} - \mathbf{T}\|_2 \tag{5.13}$$

The vector d represents the error in x, y, and z-direction.

Quaternion distance

Several different metrics can be used as a metric for 3D orientation. Huynh (2009) concludes that is "*both spatially and computationally more efficient to use the unit quaternions for 3D rotations*". Also, quaternions are ambiguity-invariant. Therefore are quaternion distance used as validation metric for orientation.

For an estimated rotation converted to unit quaternions $\hat{q}$ and a ground truth quaternion q is the rotation angle θ between them expressed as

$$\theta = 2 \arccos\left(|\langle \hat{\mathbf{q}}, \mathbf{q} \rangle|\right) \in [0, 180°] \tag{5.14}$$

where $\langle . \rangle$ is the inner product between the two quaternions.

θ denotes the angle which q_1 needs to rotate to be aligned with q_2. The smaller the angle, the better are the pose estimation in terms of orientation.

Chapter 6

Results and discussion

6.1 Evaluation

Evaluation is split up in three parts. First, evaluation of LinkNet34 with binary segmentation. This section covers training of the network, evaluation of training data, and evaluation of real data. Second, evaluation of Mask R-CNN ability to do instance segmentation. This section covers evaluation on synthetic data and real data and segmentation of point clouds. Last, the evaluation of pose estimation using segmented point clouds from Mask R-CNN is done. This is only evaluated on synthetic data, as the ground truth pose and a CAD model of the objects are needed.

6.2 Binary segmentation

Explained in section 2.2, the goal of binary segmentation is to separate the background from foreground (objects). IoU, explained in section 5.5, measures the performance of LinkNet34s binary segmentation.

The encoder in LinkNet34 is pre-trained on ImageNet from Russakovsky et al. (2015), and only fine-tuning of the network was therefore needed.

The wisdom-real dataset was used for training. This dataset was randomly split up in one training set and one validation set, see table 6.1. The validation set was only used after each training epoch. Data augmentation ”enlarged” both datasets by randomly flipping, rotation, and changing the brightness of images.

WISDOM-Real dataset	**Training set**	**Validation set**
400	320	80

Table 6.1: Distribution of images between training and validation dataset.

6.2.1 Training results

Training loss from training on the WISDOM-Real dataset is shown in Figure 6.1. The plot compares the loss, respectively, between training and validation. The blue graph represents the training loss, while the orange line is the validation loss. Training loss is sampled for every second batch, and validation loss is sampled after each epoch for every batch. One epoch consists of 40 batches. Looking at the training loss, one can see where one epoch ends and another starts by inspection where a new line segment begins and ends. This plot is used to verify that the network is training properly and not overfitting.

The shape of the two graphs is as expected, with an initial phase of high loss that rapidly decreases, followed by a slow, steady decrease, and ending with a close to constant loss. Validation loss nicely follows the training loss indicates that the network does not seem to overfit. Training curve seems to be noisy, but this is most likely due to the small batch size as outliers affect the loss stronger.

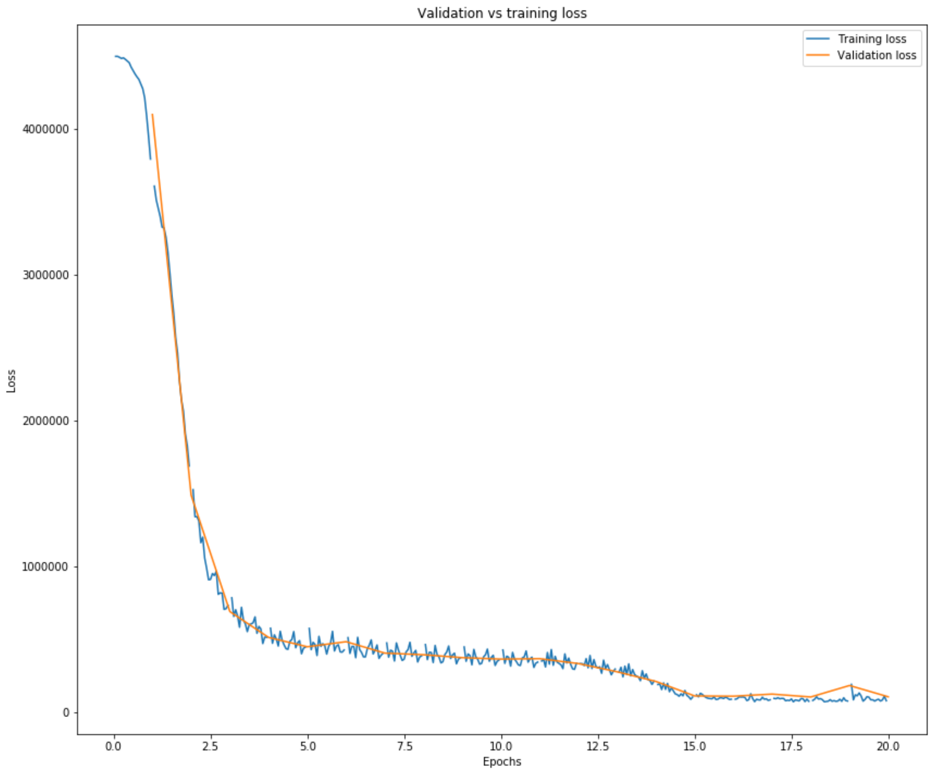


Figure 6.1: Training loss and validation loss over epochs

Cross-entropy loss with sum used as loss function, as stated in table 5.1. The reason for using the sum is because the number of pixels labeled as foreground/objects is outnumbered compared to pixels labeled as background. This is discussed further in section 6.2.5. The sum sums all the loss outputs together. Loss is initialized to $4,496,106$, and decreases to $101,814$ for training loss and $109,160$ for validation loss after 20 epochs.

6.2.2 Evaluation of training data

Evaluation is done using the randomly sampled evaluation dataset. These images are not present during the training. IoU is calculated for every input image and the average IoU for every batch along with the batch is shown in Figure 6.2.

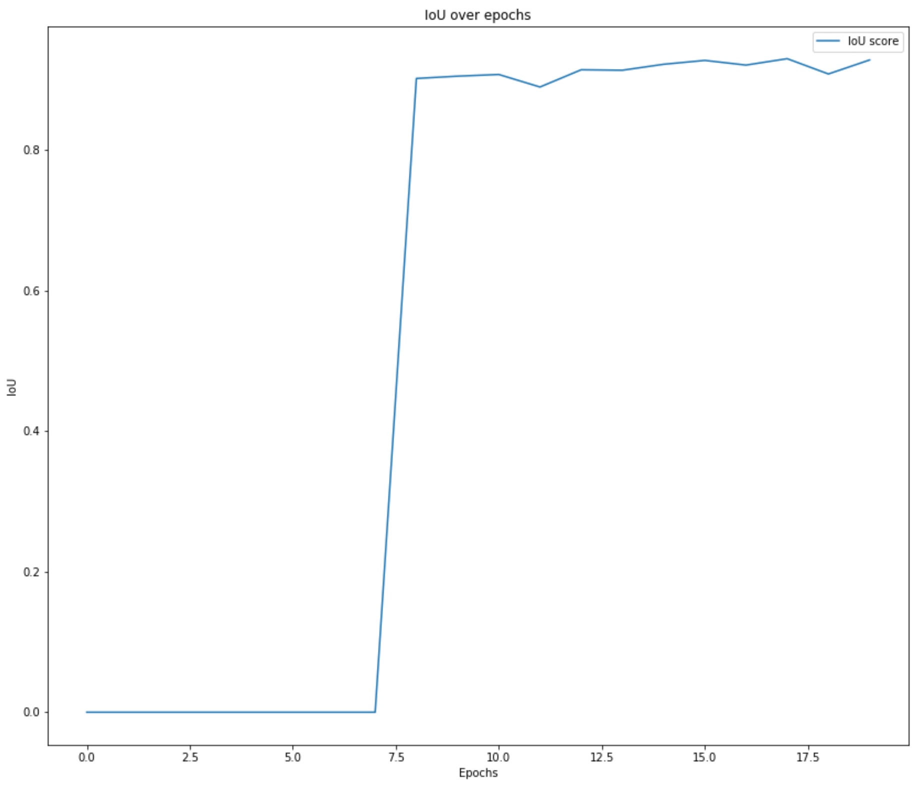
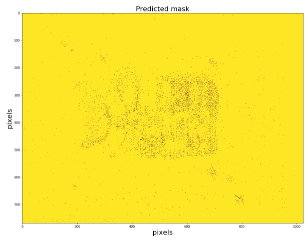


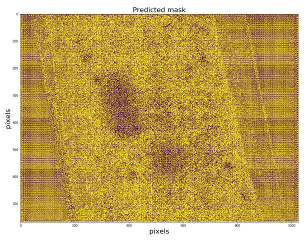
Figure 6.2: IoU over epochs

The average IoU is 0 for the first six epochs before it drastically spikes up to 0.9016. The reason for this behavior is the way the network learns to separate objects from the background. LinkNet outputs two masks, one background mask, and one object mask. Every pixel in a mask is assigned a class score for that respective mask. A prediction mask extracts the class of highest predicted class score at every pixel. This prediction mask is compared to the ground truth mask to calculate IoU.

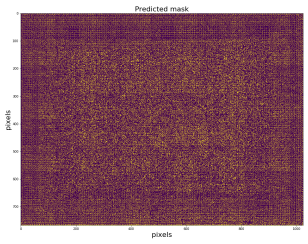
Figure 6.3 shows predicted masks outputted from the network at different epochs. Yellow pixels are objects prediction, and dark pixels are background prediction. First predicted mask is almost inverted from the ground truth (Figure 6.3a), and thus the loss in Figure 6.1 is initiated so high. During training are more and more pixels labeled as background (Figure 6.3b, 6.3c, 6.3d). After 24 batches (Figure 6.3e), is almost every pixel labeled as background. Since the number of pixels belonging to the background is so much higher than pixels belonging to the object, is the loss reduced by increasing the score (negative score is allowed) for the background labeled pixels (Figure 6.3f). In epoch 8 is the network more confident that some pixels belong to the object class, and gives these higher score values (Figure 6.3g). This is the turning point seen in Figure 6.2. After this, the network learns to better separate object mask from the background mask (Figure 6.3h, 6.3i).



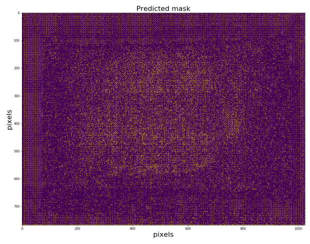
(a) Epoch:0, Batch:2



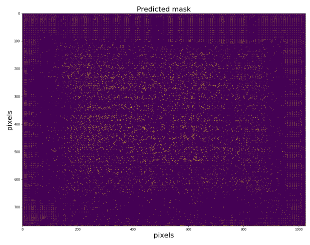
(b) Epoch:0, Batch:8



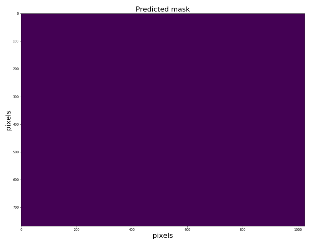
(c) Epoch:0, Batch:2



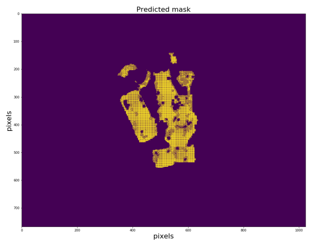
(d) Epoch:0, Batch:20



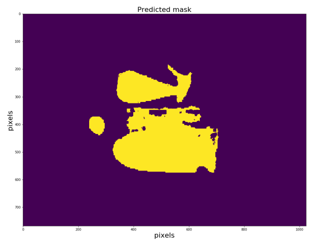
(e) Epoch:0, Batch:24



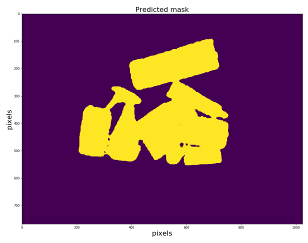
(f) Epoch:8, Batch:14



(g) Epoch:7, Batch:16



(h) Epoch:8, Batch:20



(i) Epoch:19, Batch:16

Figure 6.3: Predicted label mask extracted from different parts of training.

The mean IoU for the last validation set during training was 0.92178. This is good results regarding that a $IoU > 0.5$ is commonly known as a satisfying overlap. From Figure 6.4 it is seen that the predicted mask is not able to properly separate two objects. The borderline between objects and the background is the most difficult part to segment, accordingly is this result expected.

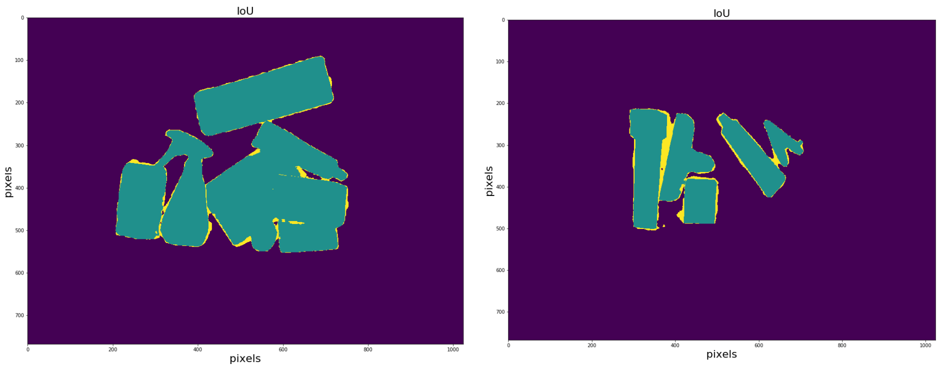


Figure 6.4: Two examples of IoU. Green pixels are intersection between predicted and ground truth mask. Yellow pixels are union between the same masks.

Figure 6.5 illustrates how accurate the binary mask overlaps the actual objects in an RGB image. This aligns well with the high IoU scores.

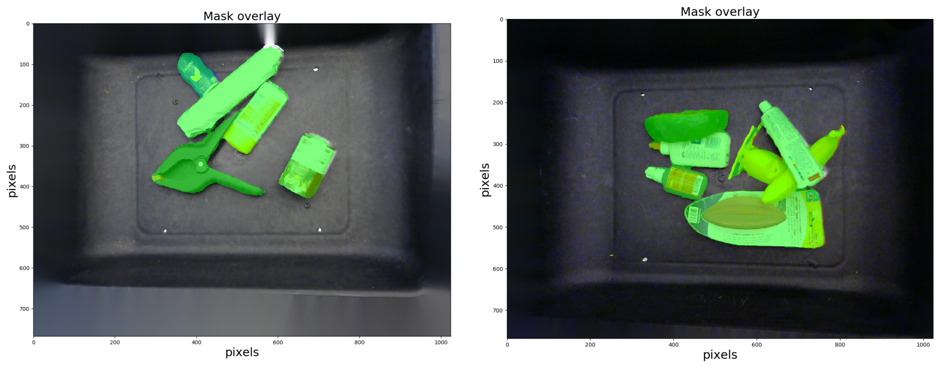


Figure 6.5: Binary masks laid over their corresponding RGB images.

6.2.3 Segmentation of training data

Point clouds are constructed from RGB-D images using the camera intrinsic matrix and camera pinhole model. The predicted binary mask is applied to the depth images to segment the point cloud. The modified depth image only have values greater than zero at pixels where the binary mask is > 0. When the point cloud is constructed using a modified depth image is only the segmented part of the point cloud projected onto 3D space. Thus, the original point cloud is segmented. Figure 6.6 shows example of point clouds segmentation. The segmentation results look good. The background labeled points are removed, and the object labels are untouched. As the network has difficulties segmenting edges, part of objects may be removed during the segmentation. This is seen in second row in Figure 6.6.

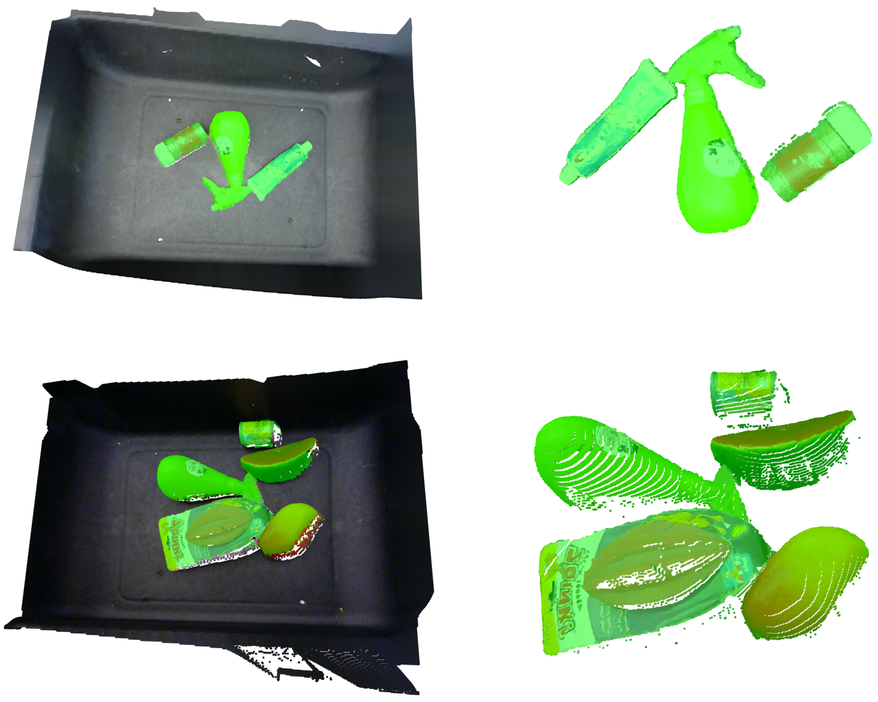


Figure 6.6: Original point cloud and segmented point cloud.

6.2.4 Segmentation on real data

Only visual results can be used as evaluation for data taken with Zivid camera. The outputted mask from LinkNet34 with the corresponding input can be seen in Figure 6.7. These results are not as expected, especially since the training data gave such good results. This indicates that the network is not able to learn general features across different type of images displaying similar scenes.

From the images its seen that the network is able to partially segment objects from the background. Which objects the network segments better and which one it segments worse seems arbitrary, as the color does not seem to be the reason. Overall, the scissors shaft and the back of a toothbrush package is segmented best. Why these objects are better segmented than, e.g., the box, is difficult to explain. For large objects, such as the book and box, is edges typically segmented, but not pixels between them. This is discussed more in 6.2.5

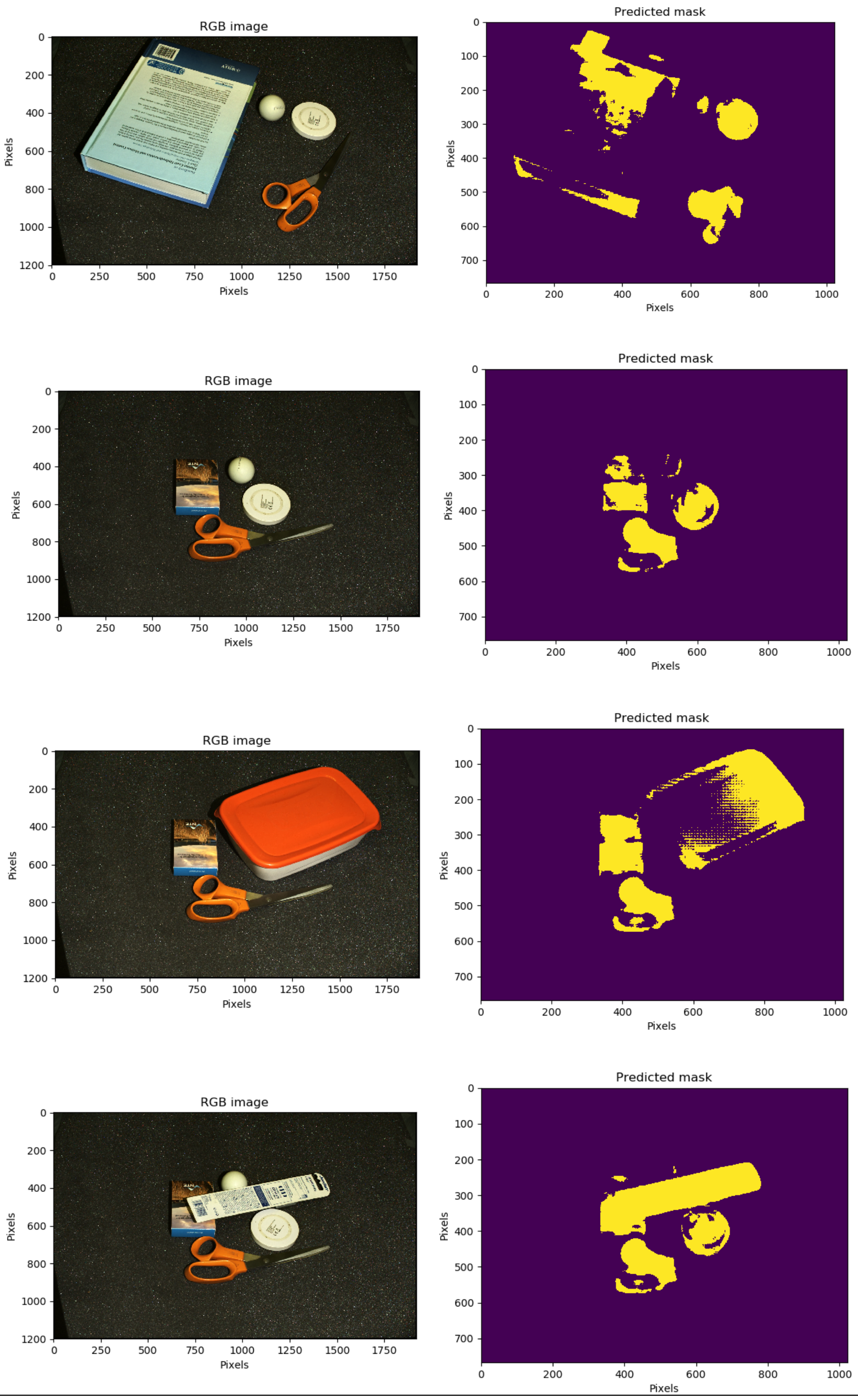


Figure 6.7: Network outputs on data taken with Zivid camera. Left: RGB image. Right: predicted mask.

6.2.5 Discussion

The networks are definitely able to learn to label a pixel as either object or background. Still, it creates one mask that merges all the different object together even when the objects are apart. Properly labeling of edges is where the network can still improve. Using different loss function that weights edges will force the network to prioritize these. Regardless, the network has an average IoU of 0.9138 for every validation after epoch 8, with a maximum $IoU = 0.9297$ at epoch 18.

In the beginning, training was done using cross entropy loss with default mean output. Since most of the pixels in the training data belong to the background class, the network immediately thinks it does a decent prediction. This initialized the loss to values around 0.73. After 15 epochs the loss was down to 0.0572, a good improvement. Nevertheless, such small numbers made it difficult for the network to update its weights efficiently.

Changing to output the sum resulted in faster training and better segmentation results. These values are also more intuitive, and hence, it is easier to observe and check for overfitting.

As Figure 6.2 shows, it takes 8 epochs to even get some results. The network set pixel values for each mask depending on how certain it is of the class. Large positive values mean that a given pixel probably belongs to the class, negative values mean that it probably does not belong to the class, and values around zero mean that the network is unsure. Observing the mean and max value of both background and object mask during these first eight epochs revealed something interesting. The network label pixels in the background mask as positive values while the object mask has negative labels. At batch 16 in epoch 8, the minimum background pixel value is -0.039, whereas the maximum object pixel value is 0.0048. Which pixels that are labeled as object class is visualized in Figure 6.3g. Increasing the learning rate might produce results faster, but tuning it too high makes the network unstable.

In contrast to the good results on training data, is LinkNet34 not able to accurately segment data from Zivid. This result was not expected. A downside with neural networks is that its difficult to properly understand why its working or why it is not working.

One can argue that LinkNet34 is not deep enough to capture the information in a high-resolution image produced by Zivid. Contrary, LinkNet34 proved good results by Shvets et al. (2018a) on Allan et al. (2019)s MICCAI 2017 Robotic Instrument Segmentation Sub-Challenge, where image size was 1920×1080. This suggests that the fault lies in the datasets or the training pipeline and not the model itself.

Even though the training dataset is small, is the model pre-trained and augmentation is applied. Pre-trained network removes the need for extensive training dataset, and data augmentation ”increases” the size of the data. Without enough augmentation is it difficult for the network to learn general features across different images. High validation loss means that the variance of the network is high, which indicates that more data is needed. From Figure 6.1, the shape of the training loss is decaying nicely. This contradicts that the training dataset is too small for LinkNet34 to learn binary segmentation.

LinkNet34 might have been trained too much and overfitted. This would explain why it performs poorly on the Zivid data. However, a typical sign for overfitting is that training loss decreases while validation loss increases. Figure 6.1 shows no sign of this.

Difference in illumination between training data and Zivid data, might be another reason for the bad performance. Zivid data have sharp contrasts and a higher spectre of intensity in colors. Training data is more uniformly, where the whole scene is brighter. More data augmentation might help the model be more robust to brightness changes and produce better segmentation results on data from Zivid.

6.3 Instance segmentation

Instance segmentation is done using Mask R-CNN, and it is tested on synthetic data and real data. With the synthetic data, it is possible to calculate IoU, since the ground truth instance masks are accessible. For real data, only a visual inspection can be done to evaluate network performance.

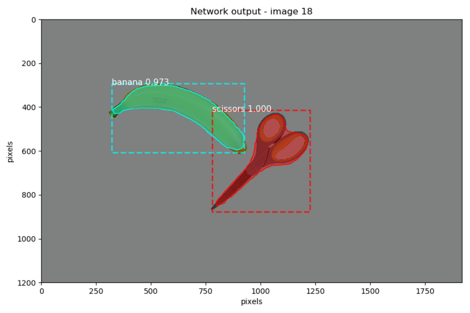
6.3.1 Segmentation on synthetic data

As explained in section 4.2 consists of the synthetic datasets of 50 images with a total of 100 instances. Table 6.2 gives an overview of the number of instances in the dataset and how well Mask R-CNN managed to segment them.

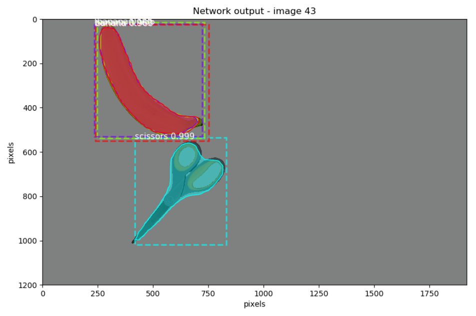
	Total	**Scissor**	**Banana**
Number of instances	100	47	53
Detected instances	95	47	48
False instances	2	0	2
Missed instances	7	0	7

Table 6.2: 100 instances from the synthetic data tested on Mask R-CNN. False instances means instances where the network labels wrong or generates multiple labels and scores for a single instance. Missed instances are instances that the network were not able to detect.

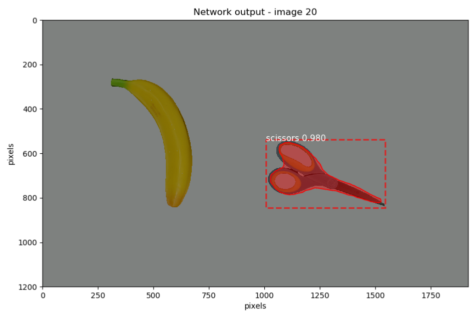
Of the 100 instances, all the 47 scissor objects were successfully detected, but only 48 of the 53 banana objects detected. Seven objects were labeled as background, and one banana got labeled three times. Because of this, false labeling did the network detect 95 instances. Figure 6.8 visualizes four examples of the network output, one example of a successful segmentation, one example of multiple instance labeling and one example of a missed detection of an instance and one example of bad segmentation.



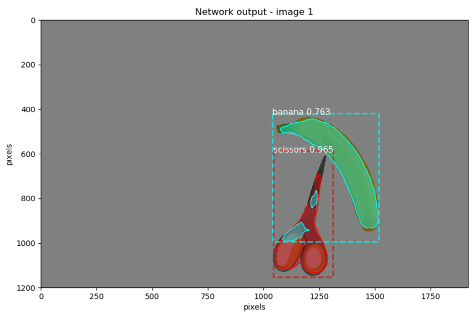
(a) Example of successfully detection



(b) Example of false detection of instances



(c) Example of missed detection of instances



(d) Example of bad segmentation

Figure 6.8: Examples of Mask R-CNN network output. Top left: Successful detection of both instances. Top right: false detection of the banana object. Bottom left: Missed detection of instance. Bottom right: Bad segmentation and low label score.

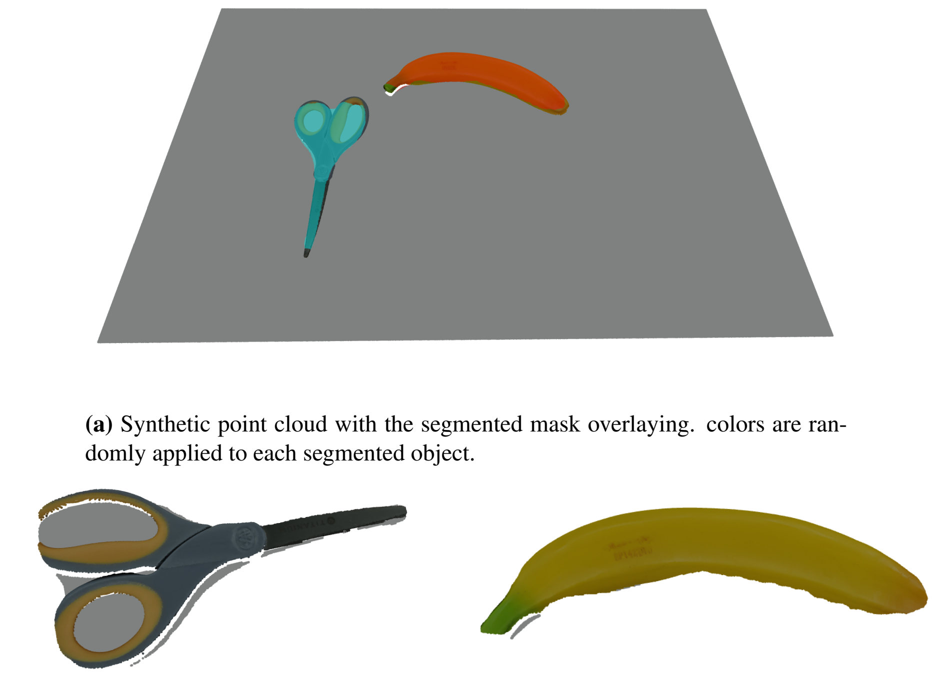
	Total	**Scissor**	**Banana**
Mean labeling score	0.96280	0.99187	0.93435
Average IoU	0.74558	0.61752	0.87098

Table 6.3: Mean label score and IoU for the synthetic dataset. Both values are in the range $[0, 1]$, where 1 is best.

The mean label score, average IoU can be seen in 6.3. When Mask R-CNN first detects an object, it is in average 96.3% confident that it has given it the right label. Scissor object has, in general, a higher labeling score. This corresponds well to the fact that every scissor instance was detected, while some banana instances were not. The average IoU is significantly better for the bananas than the scissors. The reason for this is that the scissor mask includes the holes for the fingers, while the ground truth mask does not.

These results are discussed more thoroughly in the discussion section 6.3.3.

The output masks are applied to the point cloud to segment the 3D data. An example of this is shown in Figure 6.9. The first image illustrates two instance masks overlaying the point cloud. The next two images show the segmented point clouds. The original point cloud is build up with 2304000 points. The segmented scissor contains 48959 points, while the banana contains 67205 points.


(a) Synthetic point cloud with the segmented mask overlaying. colors are randomly applied to each segmented object.

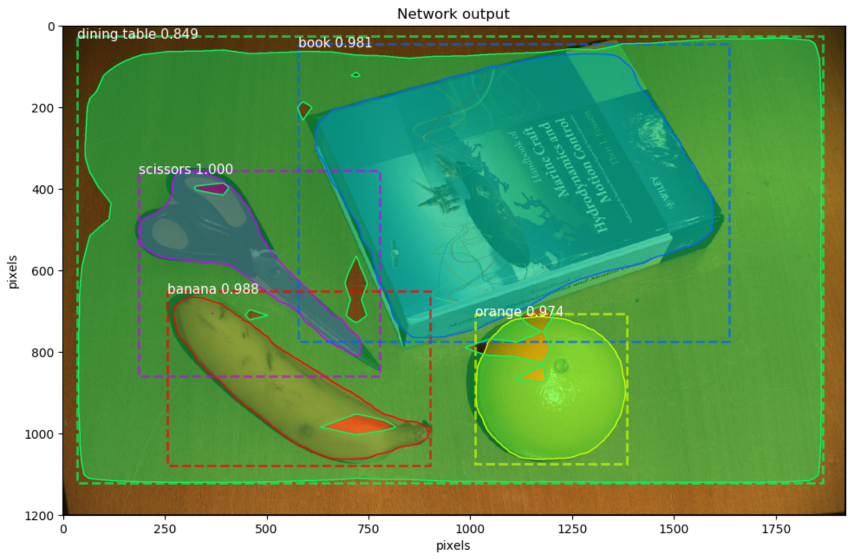
(b) Segmented point cloud of the scissor object **(c)** Segmented point cloud of the banana object

Figure 6.9: Example of segmentation mask applied to a point cloud, and the segmented object point clouds.

The segmented point clouds are used with point cloud generated from CAD models to estimate the pose of every instance. These results are presented in 6.4.

6.3.2 Segmentation on real data

Mask R-CNN is tested on data taken with a Zivid camera at the lab. Some result of this is seen in Figure 6.10 and Figure 6.11. The large image shows the output of the network on top of the original RGB image. The smaller images show the segmented point clouds of every instance detected.

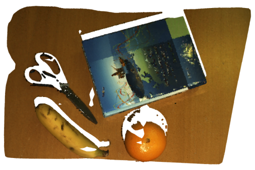


(a) Network output projected on top of RGB image.

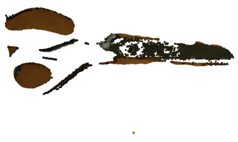
(b) Segmented banana

(c) Segmented book

(d) Segmented orange

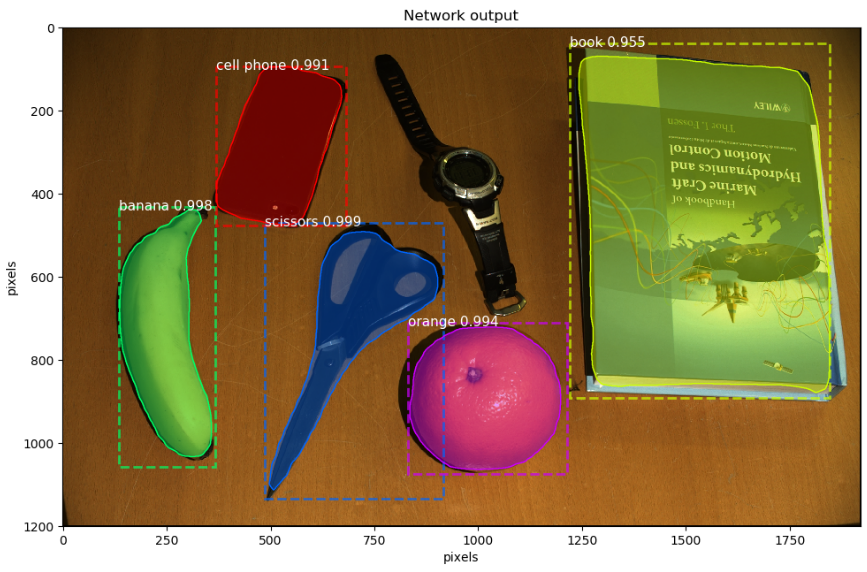


(e) Segmented dining table



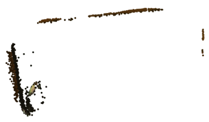
(f) Segmented scissor

Figure 6.10: Network output and segmented point clouds on real data taken with Zivid camera.



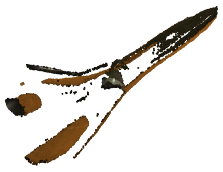
(a) Network output projected on top of RGB image.

(b) Segmented orange

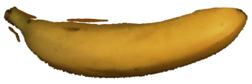


(c) Segmented cell phone

(d) Segmented book



(e) Segmented scissor



(f) Segmented banana

Figure 6.11: Network output and segmented point clouds on real data taken with Zivid camera.

Both figures illustrate that Mask R-CNN is able to predict, label, draw bounding box, and generate segmentation masks for most instances. Except from the dining table prediction in Figure 6.11, are the results decent. The class score is high, with the lowest score being the dining table with 0.849. Every other score is above 0.9, where most lies around 0.98. Bounding boxes nicely bound every instance, and it is able to capture the entire object in most cases. Segmentation mask, on the other hand, often misses edges of objects, as in binary segmentation. They are still able to capture the main shape of every object shown in the segmented point clouds.

The segmented point clouds of scissors (Figure 6.10f, 6.11e) and a cell phone (Figure 6.11c), shows strange results as almost every point in the point clouds are missing. These are not the result of bad segmentation by Mask R-CNN, but comes from the fact that Zivid camera is not able to generate points on surfaces that are close to black.

6.3.3 Discussion

Mask R-CNN have shown great results on multiple benchmark datasets such as MS COCO from Lin et al. (2014b) and PASCAL VOC 2012 by Everingham et al. (2012). Only using the pre-trained model without fine-tuning, it still shows good results.

As explained in section 2.3.9, label prediction, bounding box, and mask generation is done in parallel and independently of each other. An object that gets detected by the network will be labeled, and a bounding box and segmentation mask are generated. If the same object is detected multiple times, accordingly, it will receive multiple labels with scores, bounding boxes, and segmentation masks. This can be observed in Figure 4.4.

A bounding box does not necessarily tightly cover the segmentation mask and vice versa. This illustrates that bounding boxes and segmentation masks are created independently of each other. Even though a label can be wrong, can both bounding box and segmentation mask be accurate. If the label score was dependent on, e.g., segmentation mask, it is expected that an object labeled ”scissor” should affect the segmentation mask branch to create a scissor-shaped mask. Contrarily, a segmentation mask shaped like a banana influence the labeling branch to label it like a banana. In Figure 6.10 is the segmentation mask of a dining table clearly bad. Still, the label score is 0.849. Nevertheless, one can argue that if segmentation, labeling, and bounding box branches were dependent, better outputs could be generated. This would increase the complexity and reduce the flexibility of mask R-CNN.

Mask R-CNN shows good results on the synthetic dataset. In manages to detect, correctly label, create bounding box and segment the object for almost every instance. Nevertheless, the synthetic data is ideal in the way that there are no occlusions, no noise, and perfect illumination. A more realistic scenario would be to modify the simulator so it can add noise to the point clouds and objects can be placed on top of each other.

The average IoU score for both objects is satisfactory since it is common to classify a $IoU > 0.5$ as a good match. Banana class average IoU of 0.870 is considerably higher than the scissors 0.617, see table 6.3 for a full overview. The reason for this is that the segmentation mask outputted from Mask R-CNN always have a convex shape. Part of the scene background is therefore segmented with the scissors, as seen in Figure 6.9b. When IoU is calculated, these points will be classified as false positives (FP), and result in an overall lower IoU. The IoU for a scissor is seen in Figure 5.7c, where green pixels are the intersection and yellow pixels are the union between segmented and ground truth masks.

The label score is good for both classes, and the differences between them are too small to draw any definite conclusions on why one is better than the other. Exactly how the network sometimes misses or detects multiple instances of bananas is challenging to explain, especially when the mean label score for detected bananas is 0.934. A possible reason might be that the synthetic data is to differ from the training data. Mask R-CNN is trained on COCO dataset from Lin et al. (2014b). This dataset contains real-world images commonly found on the internet and not close up images of few objects.

Segmented point clouds are extracted from the original point cloud using the segmentation masks outputted from Mask R-CNN. Even though the segmentation is done in 2D, is the masks able to nicely segment 3D data, as seen in Figure 6.9. Parts of both classes, especially edges, are not always segmented properly. As in binary segmentation, using another loss function that weights edges of objects might be able to capture the borderline between an object and background better.

It is difficult to numerically establish how well real data is segmented as no ground truth exists. Visually it seems that segmentation mask from synthetic data and real data are equally good, e.g comparing synthetic banana (Figure 6.9c) with real banana (Figure 6.10b, 6.11f). Both segmented point clouds include a small part of the background floor, and both have some rough edges caused by the segmentation masks. Class scores and bound boxes are also similar for both data types. Though mask R-CNN sometimes misses objects, if it first detects one, it is confident that it has labeled it to the right class. If the network was fine-tuned on either WISDOM-Real dataset or a larger synthetic dataset, would it probably detect more instances. Considering the similarities between the results of synthetic and real data, could one generalize that a neural network trained on synthetic data will perform well on real data.

6.4 Pose estimation

The pose estimation is tested on the segmented point clouds generated from the segmentation mask from Mask R-CNN. Only the synthetic dataset generated from the simulator has been used. The reason for only testing on synthetic data is that the ground truth pose is available and CAD models of the objects are exists. There were no CAD models for the real objects taken with Zivid camera.

Synthetic data is ideal with no noise, uniformly distributed points, right light conditions, and no occlusions. This creates an optimal scene and optimal data. Since the pose estimation task is to investigate if it is possible to obtain pose by directly using an output mask from Mask R-CNN, is the use of only synthetic data reasonable. Therefore has there been no effort in optimizing the pose estimation algorithm

The results of pose estimation are shown in table 6.4, with example visualization in Figure 6.12. For every plot are the segmented point cloud displayed in red and the CAD point cloud in green, and they have been downsampled for better visualization.

	x *[mm]*	y *[mm]*	z *[mm]*	*angle* [deg]
Mean error	2.625	2.787	17.686	25.516°
Absolute mean error	12.063	0.282	17.791	25.516°
Standard deviation	8.592	9.7	19.735	42.428°

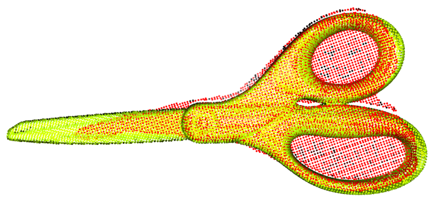
Table 6.4: Table over error in translation and rotation

Error in translation is calculated using euclidean distance and rotational error with quaternion distance. These validation metrics are explained in section 5.5. Details about deviation in translation and rotation is presented in section 6.4.1 and 6.4.2, and further discussed in section 6.4.3.

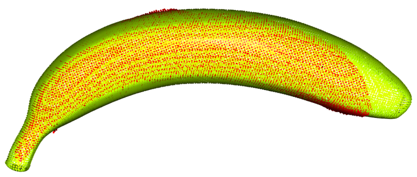
An example of ground truth matrix with corresponding estimated matrix is presented below in equation 6.1. and 6.2. Translations are in *mm* and rotation is in *rad*. Ground truth transformation is a translation and a simple rotation around z-axis.

$$T_{\text{ground truth}} = \begin{bmatrix} -0.4695 & -0.8829 & 0.0000 & -560.0000 \\ 0.8829 & -0.4694 & 0.0000 & -90.0000 \\ 0.0000 & 0.0000 & 1.0000 & 1500.0000 \\ 0.0000 & 0.0000 & 0.0000 & 1.0000 \end{bmatrix} \tag{6.1}$$

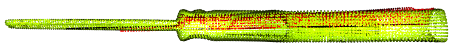
$$T_{est} = \begin{bmatrix} -0.5014 & -0.8610 & -0.085 & -547.8328 \\ 0.8652 & -0.4993 & -0.0459 & -95.4137 \\ -0.0029 & -0.0966 & 0.9953 & 1496.9327 \\ 0.0000 & 0.0000 & 0.0000 & 1.0000 \end{bmatrix} \tag{6.2}$$



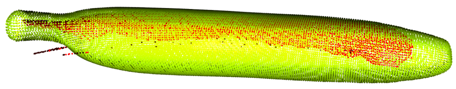
(a) Top view of scissor object



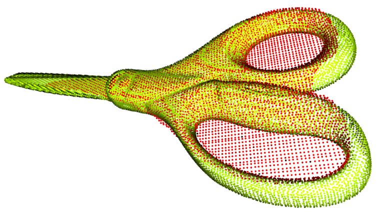
(b) Top view of banana object



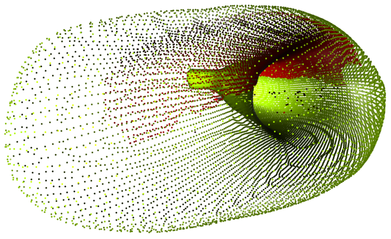
(c) side view of scissor object



(d) Side view of banana object



(e) skew view of scissor object



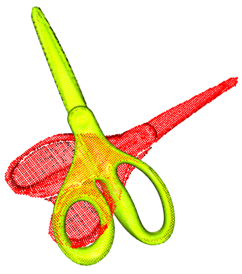
(f) inside view of banana

Figure 6.12: Different results of pose estimation showing alignment of the two point clouds. Red point cloud: segmented point cloud. Green point cloud: Point cloud generated from a CAD model

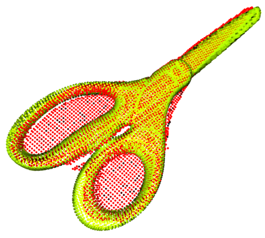
Two transformations are used to generate a pose estimation. The first transformation is a simple translation that moves a CAD point cloud to the centroid of a segmented point cloud. The centroid is calculated using the point coordinates from the segmented point cloud that have been generated using an instance mask outputted by Mask R-CNN. This transformation ensures that segmented point cloud and the CAD point cloud have overlapping 3D surfaces, as required by fast global registration by Zhou et al. (2016).

Fast global registration, explained in 5.4.4, calculates the second transformation by aligning the two point clouds. The full transformation is the dot product of these two smaller transformations. Figure 6.13a and Figure 6.13b visualizes the two transformations for the matrices stated in equations 6.2. It also shows the alignment of the two point clouds. The errors in this specific example is $x = 10.814\ mm, y = -6.015\ mm, z = 3.709\ mm$ in translation and $\theta = 3.883°$ in rotation.

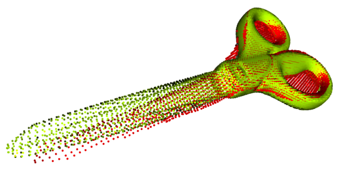
(a) The two point clouds before pose estimation is used.



(b) The two point clouds after first transformation



(c) The two point clouds after fast global registration Zhou et al. (2016).



(d) Different view of the two point clouds after fast global registration

Figure 6.13: The two transformations steps. Red point cloud: Segmented point cloud. Green point cloud: Point cloud generated from a CAD model.

Because the segmented mask from Mask R-CNN is one single mask without any holes, are part of the table present in the segmented point cloud. These points do not have a corresponding point in the CAD point cloud, and can therefore be seen as noise during the pose estimation.

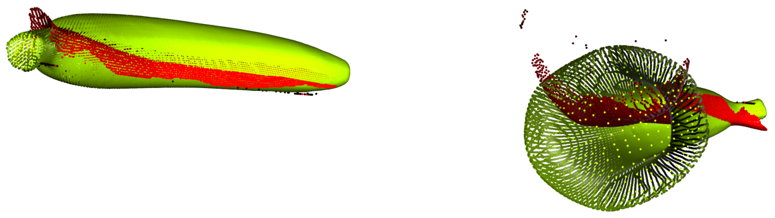


Figure 6.14: Transformation error from different views. Red point cloud: Segmented point cloud. Green point cloud: Point cloud generated from a CAD model.

6.4.1 Error in rotation

The error in rotations can be seen in 6.5. Measuring the rotational error was done using the quaternion distance explained in section 5.5. The error value is the angle in degrees needed to rotate the ground truth quaternion to the estimated quaternion. The ground truth rotation for every instance is a simple rotation around the z-axis. The number of degrees that an object rotates is randomly chosen by the simulator.

	angle
Mean error [*deg*]	$25.516°$
Standard deviation [*deg*]	$42.428°$

Table 6.5: Table over error in rotations in degree.

The numbers themselves in table 6.5 is not so good as the mean error is too high. Plotting the histogram of errors reveal that few instances that drastically increases the mean error and standard deviation. Figure 6.15 shows this histogram of instances with their errors. Most of the instances are centered below the mean. There are five instances that are rotated more than $160°$.

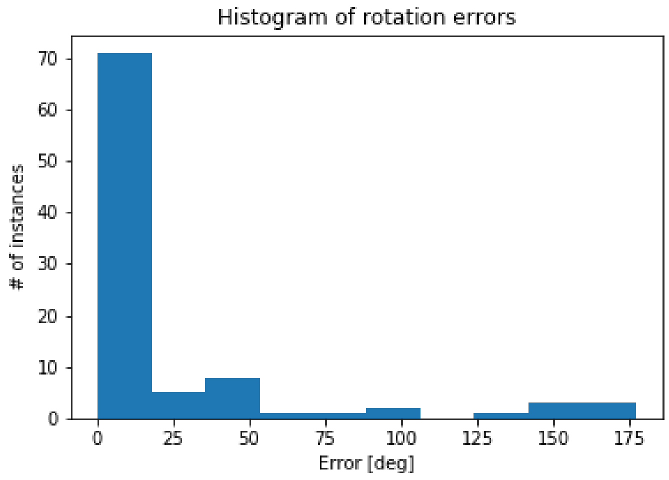


Figure 6.15: Histogram of error in rotations from the 95 detected instances.

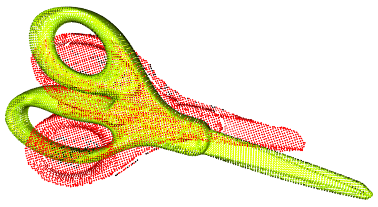
If those five instances are removed, the new mean and standard deviation is

Error in rotation	**angle**
Mean error[*deg*]	15.88°
Standard deviation [*deg*]	21.183°

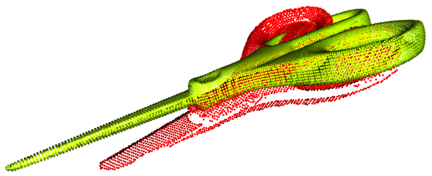
Table 6.6: Table over error in rotations in degree over a sub-set of the synthetic data. The subset contains all instances which has less than 150 degree deviation.

This is more satisfying, but not good enough for real life applications such as robotic picking.

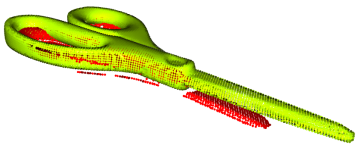
All the rotated objects are scissors. The reason for this might be because it is a relatively flat point cloud, with symmetry in XY-plane. Since the segmented point cloud is only captured from a specific angle, matching with a full 360-degree point cloud can result that the CAD model gets rotated 180 degrees off around the x-axis. This error can be seen in Figure 6.16c and Figure 6.16d. One part of the scissor has a small bump right after the two scissor blades connects with the handle. This bump is faced upwards in every scene in the synthetic dataset. When matching between the CAD point cloud and segmented point cloud, fast global registration algorithm sometimes switches the rotation around x-axis. Visualizing examples of large rotation errors are shown in Figure 6.16. This type of misalignment only occurred to scissor objects.



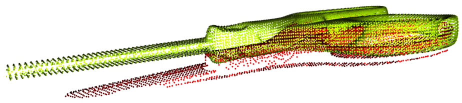
(a) Rotation error, the rotation error is off by 167.562°.



(b) Rotation error, the rotation error is off by 167.562°.



(c) Rotation error, the rotation error is off by 176.787°



(d) Rotation error, the rotation error is off by 176.787°

Figure 6.16: Two rotations errors from two views. Red point cloud: Segmented point cloud. Green point cloud: Point cloud generated from a CAD model.

Even though some rotations have a high error, is the alignment of the two point clouds pretty accurate. Because of the symmetry of the objects, a large rotation error can produce a good alignment. This is discussed more in section 6.4.3.

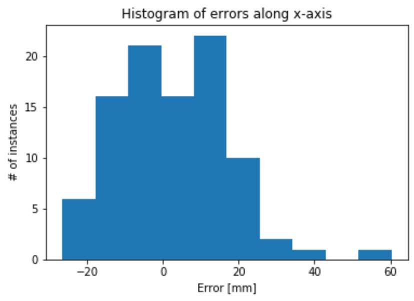
6.4.2 Error in translation

Euclidean distance is used to calculate translation error, as described in section 5.5. Every instance in the synthetic dataset is moved 1500 mm away from the camera origin and translated randomly between $[-600, 600]$ mm along x-axis and $[-400, 400]$ mm along y-axis. The mean error in the different directions can be seen in table 6.7 along with corresponding histograms in Figure 6.17.

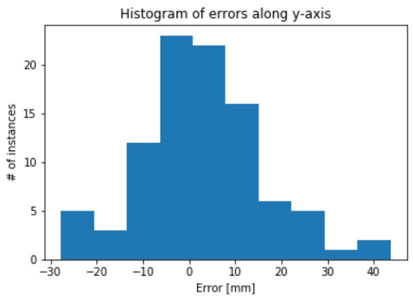
Error in translation	x-axis	y-axis	z-axis
Mean error $[mm]$	2.625	2.787	17.686
Absolute mean error $[mm]$	12.063	10.282	17.791
Absolute standard deviation $[mm]$	8.845	8.846	17.002

Table 6.7: Table over error in translation in mm. x,y,z are the axis over where the error is calculated.

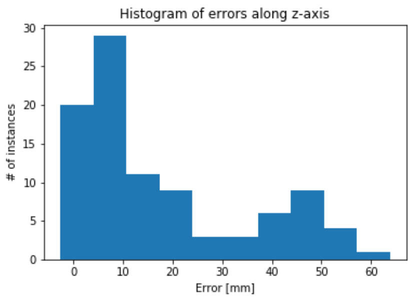
These results are quite satisfying. Absolute mean error in x- and y-direction is little under 12 mm, while error in z-direction is below 18 mm. Absolute mean error and standard deviation are computed because normal mean can cancel out values if they are equal, but with opposite sign. For 95 instances are the mean error under 3 mm in x- and y-direction and 18 mm in z-direction. For many industrial application, this is too much, but as a *proof of concept* for investigating pose estimation from segmented masks, are the results promising.



(a) Histogram of error in x-direction



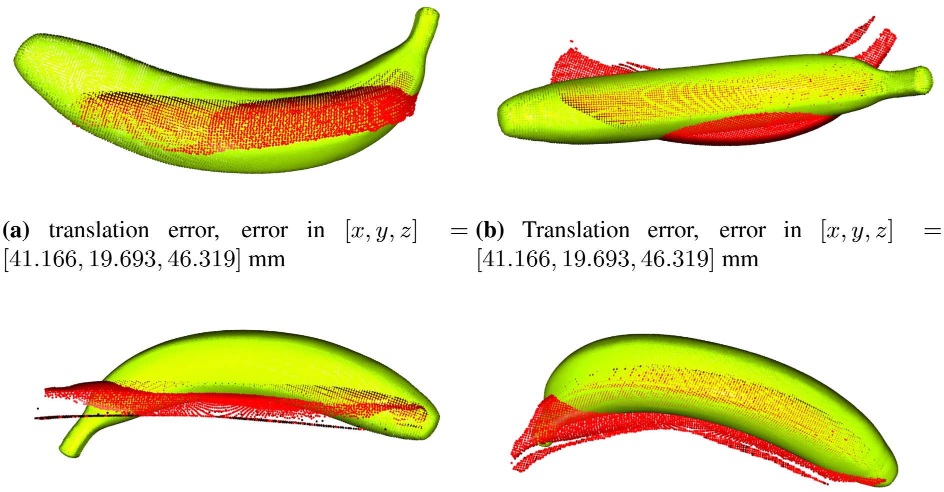
(b) Histogram of error in y-direction



(c) Histogram of error in z-direction

Figure 6.17: Histograms of errors in translation for pose estimation.

The plots in Figure 6.17 displays the magnitude of errors combined with the number of instances. One can see that the error along x- and y-axis are close to normally distributed, while the magnitude of errors along the z-axis is less than 2 cm. Examples of alignments with significant translation error can be seen in Figure 6.18 along with its translational error.



(a) translation error, error in $[x, y, z] = [41.166, 19.693, 46.319]$ mm

(b) Translation error, error in $[x, y, z] = [41.166, 19.693, 46.319]$ mm

(c) Translation error, error in $[x, y, z] = [22.023, 39.946, 46.29]$ mm

(d) Translation error, error in $[x, y, z] = [22.023, 39.946, 46.29]$ mm

Figure 6.18: Two translation errors from two point of views. Red point cloud: Segmented point cloud. Green point cloud: Point cloud generated from a CAD model.

6.4.3 Discussion

The overall performance of pose estimation is seen in the last row in table 6.4. There are mainly two reasons that explains the error in translation and rotation.

First, the shape of the two point clouds makes it difficult to retrieve a good pose estimation. Point clouds from CAD models are created by sample points on the CAD surfaces. This generates a multi-view point cloud with evenly distributed points. The segmented point cloud is generated from a single view, and its points are most likely not uniformly distributed. Also, segmentation mask outputted from Mask R-CNN can contain points that belong to other objects such as the background. These points are not present in the CAD point cloud and work as disturbances in point cloud matching.

Secondly, the geometry of the objects makes it difficult to accurately estimate pose, especially the rotation. Both scissor and banana are symmetric in one or more axis. The scissor is symmetric in XY-plane, which makes it difficult to separate between top and bottom. Even though the banana is symmetric in XY-plane will this not cause problems, since the curve of the banana makes it simple to separate a top from bottom. The fact that both objects are texture-less and have relatively smooth surfaces makes it more difficult to match point clouds properly. According to Hodan et al. (2017), this has been a problem for traditional algorithms for a long time.

When fast global registration Zhou et al. (2016) tries to align two corresponding point clouds, arises the two problems explained above. The segmented point cloud can maximum cover parts of the CAD point cloud. Combined with challenging geometric of the objects, This leads to wrong point-to-point correspondences and hence wrong pose estimation.

Estimating rotation is more challenging than estimating translation, and thus are the error in rotation larger. From Figure 6.15 its clear that most of the 95 instances have an rotational error $< 20^\circ$. Of every instance that has large rotation error, $150^\circ >$, are all scissors. This is caused by the problematic symmetry of the scissor CAD model. Visualizing these outliers shows that most of these alignments are not that bad as the error suggests. Instead, they overlap each other quite well, see Figure 6.16. The CAD point cloud is aligned to the segmented point cloud, but it is flipped upside down. This leads to a rotation error of almost 180° off.

Error in translation is surprisingly small, see table 6.7. The reason for this is because pose estimation is initialized with a translation transformation that is not far away from the ground truth translation.

Figure 6.17 shows the histogram of error for each dimension. Mean, absolute mean and the standard deviation is almost the same in x and y-direction, while these values are larger in z. Hodaň et al. (2016) state that *"...accuracy of the object position in the X and Y axis of the camera coordinate system is more important than accuracy in the Z axis, which represents the viewing direction of the camera"*. Hence, a larger error along the z-axis is acceptable.

Figure 6.18 shows examples of misalignment in translation. What is interesting is that all the objects that have a large error in the translation are bananas. The reason for this that the scissor object is thinner than the banana. The banana is approximate 5.5 cm high, while the scissor is 2.5 cm. Fast global registration aligns the point clouds by minimizing the objective function 2.17. In cases where the error is large, finds the algorithm that that optimal translation is between the top and bottom of the CAD point cloud. This can be seen in Figure 6.18 and Figure 6.14. In cases where the error is small, is the algorithm able to find the top of the banana and scissor and hence align the point clouds properly. 6.12 shows examples of this. If the objects had not been so symmetric, alignment would be less challenging.

Fast global registration is initialized using calculated 3D centroid for a given segmented point cloud. This centroid will deviate from the actual geometric center simply because the segmented point cloud is single-viewed and not uniform. This shifts the estimated centroid away from the actual centroid.

6.5 Comment on the results

Starting with an idea of how machine learning could be suited for Zivid applications, implementing and training a deep neural network to archiving results presented in 6, have been an invaluable experience for the author.

Working with machine learning and 3D computer vision have lifted the authors' knowledge from basic to more in-depth insight into both fields. Throughout the thesis have important decision been made, such as type of network architecture and type of training data. These decisions have been taken based on the best knowledge of the author at that time.

In hindsight, some of these decisions were not the best. If a similar task were to be conducted again, would another training dataset be used instead of WISDOM-Real as it is quite small. Then LinkNet34 could be used for instance segmentation as well.

A simulator would also have been implemented entirely by the author as it can generate useful training data with ground truth labels, masks, and poses.

Chapter 7

Conclusion and future work

7.1 Conclusion

The goal of this project is to investigate how machine learning can be used in combination with Zivid camera. From this was two problems defined, object segmentation of RGB-D data, and pose estimation of the segmented objects. Both problems have been solved in this thesis.

Two types of segmentation have been implemented with two different neural networks. LinkNet34 was used for binary segmentation. This network was implemented and trained on the WISDOM-Real dataset. The results show that LinkNet performed exceptionally well on the evaluation dataset, both numerically with high IoU score and visually by using the segmented masks to segmenting 3D data. Results of binary segmentation on data captured by Zivid camera is discussable. At best LinkNet managed to partly segment objects in an image. The reason for this cannot be stated for sure, other than it lies in either the training data or the training pipeline.

Instance, segmentation is done using Mask R-CNN. Mask R-CNN shows impressive results for both synthetic and Zivid data, especially since no training or fine-tuning has been done to the network. It detects and segments most instances with good labeling scores, bounding boxes, and prediction masks. The average IoU calculated for the synthetic data proves numerically that Mask R-CNN produces accurate segmentation masks that can be used to segment objects in 3D data. Visual inspection between segmented objects from synthetic and real data substantiate that the network has an equivalent IoU score on real data as well.

The results of the pose estimation were surprisingly good. The findings confirm that fast global registration computes a decent estimate of a pose. This is without any focus on optimizing the method in any way. Error in translation is small due to proper initialization, while the error in rotation sometimes deviates a lot. This is because of the simple geometry and symmetry of the object instances. The numerical findings are consistent with the visual results. As a *proof of concept*, the findings offer strong evidence that pose estimation can achieve good performance if the global registration algorithm is optimized to the specific data.

In conclusion, the overall findings of this study are satisfying. By utilizing both RGB image and 3D data captured by Zivid camera, the deep neural network estimates segmentation masks. These are used to segment the 3D data. The segmented 3D data shows sufficient quality to be used in pose estimation of the objects in the scene.

7.2 Future work

As the results showed, neural networks were able to segment 3D data from Zivid and from this pose could be estimated. This motivates for further experiments.

The poorly binary segmentation suggests that further research is needed. As the model has proven to work on other segmentation tasks, is a good start to train the network on a larger dataset to see if this improves the results.

As Zivid produces both RGB and depth images, is a natural next step to extend the networks to incorporate depth information. Depth images add new information and should increase the performance of networks. There are two possible strategies for doing this. One way is passing RGB and depth images through two separate networks and fuse the outputted feature maps together. Another way is adding a new input channel to the network such that both RGB and depth images are passed through the same network. This was implemented by the author but was not used due to time constraints.

A neural network is only as good as the training data. Developing high-quality datasets which can be used for training is often time-consuming. Danielczuk et al. (2018) proves that training on synthetic data gives good performance on real data. Therefore, the simulator should be further developed. The simulator generates 3D data for any type of objects since it only needs a point cloud representation of the instances to create data. Options for adding noise and occluding objects would give more realistic datasets. Same datasets can be used for binary segmentation, instance segmentation and pose estimation as the ground truth masks and poses are generated for every object in a scene. Good datasets facilitate further research in both segmentation and pose estimation domain.

Further investigation is needed to accurately determine how well-suited mask R-CNN is for pose estimation. The registration method used for point cloud alignment can be tuned to minimize pose error. Weather an optimized, fast global registration produces good enough estimates by itself, or if it should be used as initializing for a local registration method such as ICP, should be tested. Other registration methods such as RANSAC and generic algorithm should also be explored as fast global registration was the only method tested in the experiments.

Bibliography

Abdulla, W., 2017. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN.

Allan, M., Shvets, A., Kurmann, T., Zhang, Z., Duggal, R., Su, Y., Rieke, N., Laina, I., Kalavakonda, N., Bodenstedt, S., García-Peraza, L. C., Li, W., Iglovikov, V., Luo, H., Yang, J., Stoyanov, D., Maier-Hein, L., Speidel, S., Azizian, M., 2019. 2017 robotic instrument segmentation challenge. CoRR abs/1902.06426.
URL http://arxiv.org/abs/1902.06426

Badrinarayanan, V., Kendall, A., Cipolla, R., Dec 2017. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (12), 2481–2495.

Canziani, A., Paszke, A., Culurciello, E., 2016. An Analysis of Deep Neural Network Models for Practical Applications, 1–7.
URL http://arxiv.org/abs/1605.07678

Chaurasia, A., Culurciello, E., 2018. LinkNet: Exploiting encoder representations for efficient semantic segmentation. 2017 IEEE Visual Communications and Image Processing, VCIP 2017 2018-Janua, 1–4.

Chu, K., Zhang, Q., Han, H., Xu, C., Pang, W., Ma, Y., Sun, N., Li, W., 2017. A systematic review and meta-analysis of nonpharmacological adjuvant interventions for patients undergoing assisted reproductive technology treatment. International Journal of Gynecology and Obstetrics 139 (3), 268–277.

Csurka, G., Larlus, D., Perronnin, F., 01 2013. What is a good evaluation measure for semantic segmentation? Vol. 26.

Danielczuk, M., Matl, M., Gupta, S., Li, A., Lee, A., Mahler, J., Goldberg, K., 2018. Segmenting Unknown 3D Objects from Real Depth Images using Mask R-CNN Trained on Synthetic Data.
URL http://arxiv.org/abs/1809.05825

Deng, Z., Sun, H., Zhou, S., Zhao, J., Lei, L., Zou, H., 05 2018. Multi-scale object detection in remote sensing imagery with convolutional neural networks. ISPRS Journal of Photogrammetry and Remote Sensing.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., Zisserman, A., 2012. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

Garcia-Garcia, A., Orts-Escolano, S., Oprea, S., Villena-Martinez, V., Garcia-Rodriguez, J., 2017. A Review on Deep Learning Techniques Applied to Semantic Segmentation, 1–23.
URL `http://arxiv.org/abs/1704.06857`

Geng, J., Jun 2011. Structured-light 3d surface imaging: a tutorial. Adv. Opt. Photon. 3 (2), 128–160.
URL `http://aop.osa.org/abstract.cfm?URI=aop-3-2-128`

Girshick, R., 2015. Fast R-CNN. Proceedings of the IEEE International Conference on Computer Vision 2015 International Conference on Computer Vision, ICCV 2015, 1440–1448.

Girshick, R., Donahue, J., Darrell, T., Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 580–587.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press, `http://www.deeplearningbook.org`.

Gu, C., Lim, J. J., Arbeláez, P., Malik, J., 2009. Recognition using regions. 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1030–1037.

He, K., Gkioxari, G., Dollar, P., Girshick, R., 2017. Mask R-CNN. Proceedings of the IEEE International Conference on Computer Vision 2017-October, 2980–2988.

He, K., Zhang, X., Ren, S., Sun, J., 2015. Deep Residual Learning for Image Recognition.
URL `http://arxiv.org/abs/1512.03385`

Hodan, T., Haluza, P., Obdrzálek, S., Matas, J., Lourakis, M. I. A., Zabulis, X., 2017. T-LESS: an RGB-D dataset for 6d pose estimation of texture-less objects. CoRR abs/1701.05498.
URL `http://arxiv.org/abs/1701.05498`

Hodaň, T., Matas, J., Obdržálek, Š., 2016. On evaluation of 6d object pose estimation. In: Hua, G., Jégou, H. (Eds.), Computer Vision – ECCV 2016 Workshops. Springer International Publishing, Cham, pp. 606–619.

Hu, R., Dollár, P., He, K., Darrell, T., Girshick, R., 2017. Learning to Segment Every Thing.
URL `http://arxiv.org/abs/1711.10370`

Huang, Z., Huang, L., Gong, Y., Huang, C., Wang, X., 2019. Mask Scoring R-CNN. In: CVPR.

Huynh, D. Q., Oct 2009. Metrics for 3d rotations: Comparison and analysis. Journal of Mathematical Imaging and Vision 35 (2), 155–164.
URL `https://doi.org/10.1007/s10851-009-0161-2`

Jaderberg, M., Simonyan, K., Zisserman, A., Kavukcuoglu, K., 2015. Spatial Transformer Networks, 1–15.
URL `http://arxiv.org/abs/1506.02025`

Jiang, J., Zheng, L., Luo, F., Zhang, Z., 2018. Rednet: Residual encoder-decoder network for indoor RGB-D semantic segmentation. CoRR abs/1806.01054.
URL `http://arxiv.org/abs/1806.01054`

Kingma, D. P., Ba, J., 2015. Adam: A method for stochastic optimization. CoRR abs/1412.6980.

Lecun, Y., Bottou, L., Bengio, Y., Haffner, P., Nov 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86 (11), 2278–2324.

Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C. L., 2014a. COCO common objects in context detection evaluation. `http://cocodataset.org/#detection-eval`.

Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C. L., 2014b. Microsoft COCO: common objects in context. CoRR abs/1405.0312.
URL `http://arxiv.org/abs/1405.0312`

Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S., 2017. Feature pyramid networks for object detection. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017 2017-January, 936–944.

Long, J., Shelhamer, E., Darrell, T., June 2015. Fully convolutional networks for semantic segmentation.

Nielsen, M. A., 2015. Neural Networks and Deep Learning. Determination Press, `http://neuralnetworksanddeeplearning.com/chap1.html`.

Noh, H., Hong, S., Han, B., 2015. Learning deconvolution network for semantic segmentation. In: Computer Vision (ICCV), 2015 IEEE International Conference on.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A., 2017. Automatic differentiation in PyTorch. In: NIPS Autodiff Workshop.

Perenzoni, M., Stoppa, D., Perenzoni, M., Stoppa, D., nov 2011. Figures of Merit for Indirect Time-of-Flight 3D Cameras: Definition and Experimental Evaluation. Remote Sensing 3 (11), 2461–2472.
URL `http://www.mdpi.com/2072-4292/3/11/2461`

Pomerleau, F., Colas, F., Siegwart, R., Magnenat, S., 04 2013. Comparing icp variants on real-world data sets. Autonomous Robots.

Raguram, R., Frahm, J.-M., Pollefeys, M., 10 2008. A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. Vol. 5303. pp. 500–513.

Ren, S., He, K., Girshick, R., Sun, J., 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE transactions on pattern analysis and machine intelligence 39 (6), 1137–1149.
URL `http://www.ncbi.nlm.nih.gov/pubmed/27295650`

Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 9351, 234–241.

Ruder, S., 2016. An overview of gradient descent optimization algorithms, 1–14.
URL `http://arxiv.org/abs/1609.04747`

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., Fei-Fei, L., 2015. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115 (3), 211–252.

Schwarz, M., Milan, A., Periyasamy, A. S., Behnke, S., 2018. RGB-D object detection and semantic segmentation for autonomous manipulation in clutter. International Journal of Robotics Research 37 (4-5), 437–451.

Shvets, A. A., Rakhlin, A., Kalinin, A. A., Iglovikov, V. I., 2018a. Automatic instrument segmentation in robot-assisted surgery using deep learning. In: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA). pp. 624–628.

Shvets, A. A., Rakhlin, A., Kalinin, A. A., Iglovikov, V. I., 2018b. Linknet implementation on pytorch. `https://github.com/ternaus/robot-surgery-segmentation`.

Simonyan, K., Zisserman, A., 09 2014. Very deep convolutional networks for large-scale image recognition. arXiv 1409.1556.

Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Fei-Fei, L., Savarese, S., 2019. Densefusion: 6d object pose estimation by iterative dense fusion. CoRR abs/1901.04780.
URL `http://arxiv.org/abs/1901.04780`

Xiang, Y., Schmidt, T., Narayanan, V., Fox, D., 2017. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. CoRR abs/1711.00199.
URL `http://arxiv.org/abs/1711.00199`

Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K., 2017. Aggregated residual transformations for deep neural networks. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017 2017-Janua, 5987–5995.

Xu, T., Jin, X., Huang, P., Zhou, Y., Lu, S., Jin, L., Pasupathy, S., 2016. TensorFlow: A System for Large-Scale Machine Learning. TensorFlow: A System for Large-Scale Machine Learning, 619.

Zeng, A., Song, S., Nießner, M., Fisher, M., Xiao, J., 2016. 3dmatch: Learning the matching of local 3d geometry in range scans. CoRR abs/1603.08182.
URL `http://arxiv.org/abs/1603.08182`

Zhou, Q.-Y., Park, J., Koltun, V., 10 2016. Fast global registration. Vol. 9906.

Zhou, Q.-Y., Park, J., Koltun, V., 2018. Open3D: A modern library for 3D data processing. arXiv:1801.09847.

Zhou, Y., Tuzel, O., 2017. Voxelnet: End-to-end learning for point cloud based 3d object detection. CoRR abs/1711.06396.
URL `http://arxiv.org/abs/1711.06396`