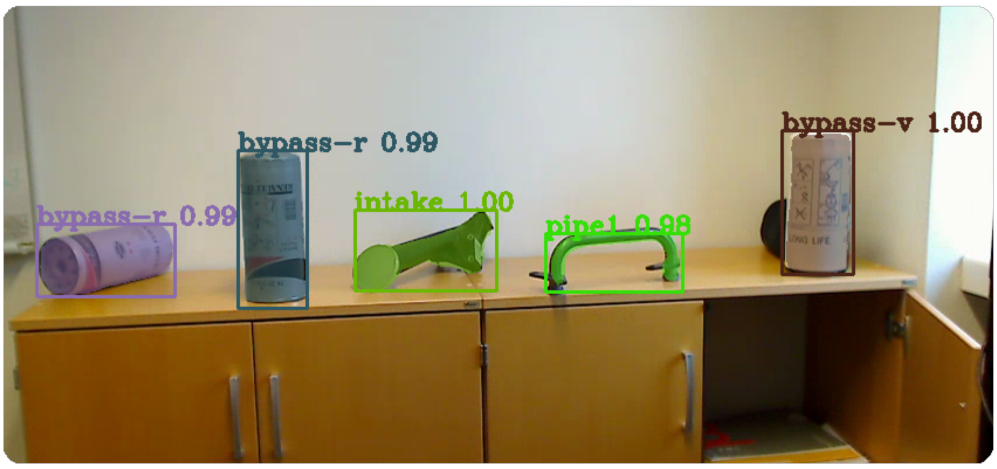
CHALMERS
UNIVERSITY OF TECHNOLOGY



Instance Segmentation and Pose Estimation of Assembly Parts Using Deep Learning and Synthetic Data Generation

Master's thesis in Systems, Control and Mechatronics

TORBJØRN OPHEIM & JONAS LECEROF

The Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master Thesis

Instance segmentation and pose estimation of assembly parts using deep learning and synthetic data generation

Torbjørn Opheim
torbjornop.en@gmail.com

Jonas Lecerof
lejonas@student.chalmers.se

The Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Instance segmentation and pose estimation of assembly parts using deep learning and synthetic data generation
TORBJØRN OPHEIM & JONAS LECEROF

Supervisor: Knut Åkesson, Chalmers
Supervisor: Per-Lage Götvall, Volvo AB

Master's Thesis 2019
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg

Cover: Instance segmentation of assembly parts.

Typeset in LaTeX
Gothenburg, Sweden 2019

Instance segmentation and pose estimation of assembly parts using deep learning and synthetic data generation
Torbjørn Opheim & Jonas Lecerof
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Detecting the exact location of objects enables more sophisticated assistance systems and robots to excel in a multitude of scenarios and industries. At the manufacturing plants of Volvo, there is ongoing research looking into integrating collaborative robotic systems for collaborative assembly, stand-alone bin picking, and quality inspection. Object recognition, instance segmentation, and pose estimation have been identified as crucial parts for solving these problems. The aim is to implement flexible end-to-end solutions, where new assembly parts can be integrated with ease. Because of this, the implementation is limited to the use of synthetic data and RGB images.

In this thesis, two different networks are trained solely on rendered data, and their suitability for different applications is evaluated. We show that for instance segmentation, MaskRCNN generalizes well to real images, but its performance worsens when the scene is subjected to direct light or when the objects are obscured. Furthermore, the performance with different types of data augmentations is evaluated and the problems that arise when optimizing a model for real images using synthetic data are illuminated.

A graphics rendering tool, The Blender Python API, was used to generate datasets for the two different networks. Compared to previous work, on which this thesis is built on, the class list was extended, and randomizing parameters were removed. The rendered objects were then composed with the SUN2012 Pascal collection as backgrounds. Both networks were trained on this data and then evaluated using the metrics: intersection over union (IoU) for semantic labeling and Average Distance (ADD) on both captured and rendered images for pose estimation.

PVNet uses a more shallow network for object segmentation that obtains good results on synthetic validation data but more varying results when validating on real images. On synthetic validation, we obtained similar pose estimation results with assembly parts as we did using a LineMod object with the LineMod settings. This means we can expect similar results as PVNet's author received with "fully" trained models at 200 epochs. On real images, the method starts to perform worse after 50 epochs. The degree of accuracy could be good enough for bin picking, but more accuracy is needed in order to implement it into a quality inspection toolbox.

Keywords: Convolutional Neural Networks, Deep learning, Instance Segmentation, Pose Estimation, Semantic labeling, Synthetic data.

Preface

This thesis is the integral of a 30 ECTS project intended for master students at the master's program in Systems, Control, and Mechatronics at Chalmers and master's program in Industrial Cybernetics at NTNU. The project was worked on during the duration of the spring semester 2019.

We want to thank Volvo and Chalmers for supplying the equipment used during this project. We also want to extend our gratitude towards Jason Li for the previous work related to semantic labeling and data generation and William Falkman for providing photo-realistic textures of the assembly parts.

Acknowledgements

We acknowledge the continued supervision and administrative help from:

Knut Åkesson, Professor and Director of the Master program Systems, control and mechatronics at Chalmers
Per-Lage Götvall, Senior Research Engineer at Volvo AB
Yiannis Karayiannidis, Assistant Professor in Systems, control and mechatronics at Chalmers
Ole Morten Aamo, Professor of Technical Cybernetics at NTNU

We also acknowledge the enlightening dialogues with:

Peng Sida, author of PVNet.
Stan Birchfield, author of DOPE
Jonathan Trembley, author of DOPE.

Torbjørn Opheim & Jonas Lecerof - Gothenburg, June 2019

Contents

List of Figures

List of Tables

s

Nomenclature

3D	Three Dimensional (XYZ Translation)
6D	Six Dimensional XYZ Translation and Rotation
AGV	Automated Guided Vehicle
AI	Artificial Intelligence
API	Application Programming Interface
CAD	Computer Aided Design
CGI	Computer Generated Imagery
CNN	Convolutional Neural Network
COCO	Microsoft Common Objects In Context
CPU	Central Processing Unit
CVFH	Clustered Viewpoint Feature Histogram
DNN	Deep Neural Network
DoF	Degree(s) of Freedom
FAIR	Facebook Artificial Intelligence Research
FoR	Frame of Reference
FOV	Field Of View
FPS	Frames Per Second
GAN	General Adversarial Network
GFLOPS	Giga Floating Point Operations Per Second
GPU	Graphics Processing Unit
ICP	Iterative Closest Point
IoU	Intersection over Union

JPG / JPEG	Joint Photograpic Experts Group (Image Format
NN	Neural Network
ONEIROS	Open-ended Neuro-Electronic Intelligent Robot Operating System
PLY	Polygon File Format
Pose	Rotation and Translation in Six Dimensions
PVNet	Pixel-wise Voting Network
RANSAC	Random Sample Consensus
RCNN	Region Convolutional Neural Network
RGB	Red, Green, Blue (Three Color Channels
RGB-D	Red, Green, Blue (Color and Depth Information
RGBA	Red, Green, Blue, Alpha (Color with Transparency Channel
RNN	Recurrent Neural Network
ROS	Robot Operative System
SIFT	Scale-Invariant Feature Transformation
STL	Standard Triangulation Language (File Format for 3D Geometry
SVM	Support Vector Machine
UV Map	Coordinate mapping between Image Texture and 3D Object

1 Introduction

In this chapter, the general background to the problem, as well as the purpose and scope is explained. A number of questions to be answered throughout this master thesis has been formulated and finally some crude limitations and contributions are stated.

1.1 Background

Throughout the final steps in the production line at Volvo's manufacturing plants, there are lots of detailed instructions regarding assembly that the operator needs to execute in order to ensure the best possible final product. This part of the factory could, therefore possibly be improved with the use of collaborative robots where the production can benefit from the agility of the humans and the repeatability of the robots. In order to enable robots to work in an environment tightly coupled with humans, they would need systems for detecting where the humans are positioned at all times, effectively avoiding collisions and optimizing the resource allocation of shared tools. Incorrect assembly of the engine can become both costly and time-consuming for all parts involved. Therefore the manufacturer, as well as the customer, could benefit from a computer vision system that continuously checks if assembled parts are mounted, and done so correctly.

Quality inspection in manufacturing can be in the form of detecting anomalies and defects, but also verifying if objects are assembled correctly. It can be achieved by estimating the pose of the assembled components and checking if this is any different from a stored ground-truth. To perform this well, a robust pose estimation method with high accuracy is necessary. The pose estimation problem is well known within academia and previous attempts to solve this are mostly either based on template matching [25], keypoint-based methods [26] or dense methods [27]. However, the biggest challenges for template matching methods are truncations, which makes these methods not applicable to this industrial setting. Also, when using keypoint-methods that produce heatmaps, they perform poorly when those are not found, due to object occlusion or truncation. Dense methods are more computationally expensive, but are also more robust. It is most common to find that these use images with depth information (RGB-D) for more straight forward point-to-point matching.

When training a supervised convolutional neural network (CNN), a large annotated dataset for training and verification is needed. Manually annotated datasets require extensive human effort to create, thus making this approach undesirable.

Therefore, it would be beneficial if datasets could be created by utilizing a graphics rendering engine. Although synthetic data has different characteristics than images captured through a real camera. In computer vision, the real world is non-uniform and perceived through a lens with some distortion that might also be covered with some particles of dust or equivalent. The network might, therefore, put too much emphasis on the rendered properties unless precautions are taken to increase generalization.

Figure 1.1 illustrates the part of the assembly line where the possibility of improvements is being investigated. At this specific station, a ladder frame, three filters, two oil pipes, and an air intake are mounted on the engine block with M10 bolts.

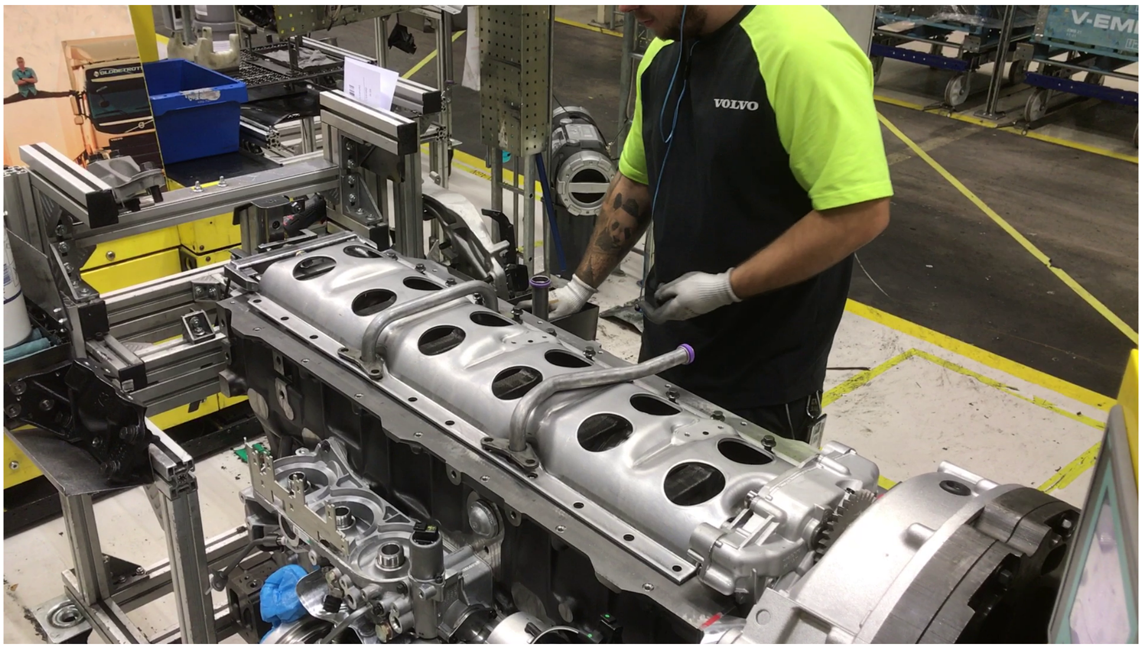


Figure 1.1: Part assembly location that is being investigated.

1.1.1 Objects

For the investigated section of the assembly line, there are 8 kinds of objects mounted on the engine. In Figure 1.2 is a visual representation of them at different scales. Each of the objects has more or less unique visual properties.

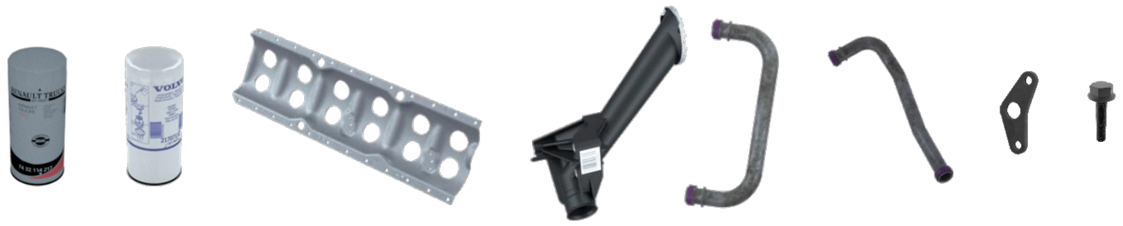


Figure 1.2: From left to right; Renault bypass filter, Volvo bypass filter, ladderframe, oil suction intake, U-shaped coolant pipe, S-shaped coolant pipe, pipe fixture, bolt m10.

- **Bypass filters:** These filters are either gray with the Renault decal or white with the Volvo decal and have a dimension of 27 × 10 cm. The rigid body of these filters is the same, making the texture the only distinguishable feature between them. The filters are threaded downwards into the engine block during assembly, and will usually be seen in an upright position.
- **Ladder frame:** The ladder frame has a dimension of 97 × 25 cm, making it the largest object to be mounted in this part of the assembly line. There are 12 equidistantly spaced large holes, three smaller ones for the different pipes and 24 small holes for the bolts. The texture of the ladder frame is matte metallic. During the assembly, all of the other objects, except the filters, are mounted on top of it.
- **U-shaped coolant pipe:** This pipe is 27 × 10 cm in size with a matte metallic-gray texture. Each of the ends has a purple rubber ring, although if the pipe is mounted correctly, these seals are not visible. Since this pipe is symmetric, it can be installed in two different ways.
- **S-shaped coolant pipe:** Compared to the U-shaped coolant pipe, the S-shaped pipe is slightly larger with a size of 33 × 10 cm. Furthermore, it is not symmetric, making it only possible to mount it on the engine in one way. As the U-shaped counterpart, it has a matte metallic-gray texture with the same kind of purple seals on each end.
- **Oil suction intake:** This part is 33 × 27 cm and is made of black plastic. On the tip of the object, there is a coarse grid metallic filter. There is also a printed bar code on the side of it.
- **Pipe fixture:** The fixtures for the U- and the S-shaped coolant pipes are 10 × 4 cm in size and has a dark matte metallic texture. They have two smaller holes for the bolts and a larger one for the coolant pipe.
- **Bolt m10:** The bolts are flanged, 30 mm long, with an m10 threading thereby the smallest part in this step of the assembly line. The texture is black and metallic, and they are all mounted along the edges of the ladder frame.

1.2 Instance segmentation

Instance segmentation is the act of predicting which pixels correspond to a specific object given an image. What separates it from semantic labeling is that it will also be able to detect different instances of the same object.

1.3 Pose estimation

Pose estimation is the act of estimating the orientation and translation of an object in a coordinate frame. Pose estimation can be done from RGB images, RGB-D images, and point clouds. More rich information usually yields better results, although, in this thesis, the focus is on using solely RGB images.

1.4 Purpose

The purpose of this thesis is to evaluate the performance, applicability, and feasibility of implementation of state-of-the-art instance segmentation and pose estimation with neural networks in the production line, when these methods only have access to synthetically generated data. It would be beneficial for Volvo if the proposed methods could be incorporated into a quality inspection toolbox and bin picking robots that can be implemented in their manufacturing plants. This is one stepping stone toward a more automated factory where humans and robots work jointly together in an open environment.

1.5 Research Questions

This master thesis attempts to evaluate the generalization achieved by training two different neural networks on synthetic data using inference on captured images. Mask R-CNN and PVNet are both methods that execute semantic labeling, while PVNet is also attempting to solve the pose estimation problem.

The question to be answered with respect to instance segmentation is:

- Do we get a better result with Mask R-CNN after using a different data generation approach and using data augmentation?

- What can we expect in accuracy, when running inference on real images?

The evaluation criteria used to determine the accuracy is: precision, recall and Intersection over Union (IoU).

- How does PVNet, trained on synthetic data, perform on synthetic and real validation sets?

The evaluation of this part will be done by analyzing results from IoU and the ADD method.

1.6 Limitations

The main focus is to implement working algorithms for instance segmentation and pose estimation. In order to do this, designing a synthetic data generator for each method is also of high focus. The end goal is to see how well each method performs on synthetic data and real data respectively. Then, combined with a relative pose algorithm, it will be evaluated if it is viable to verify the quality of the assembly based on these methods. The speed of the algorithms developed will not be the primary objective of this work, but rather, the focus is to investigate how feasible this approach is for solving the problem accurately enough with the set limitations.

1.7 Contribution

Throughout the duration of this thesis, an extensive literature study on theory related to different types of machine learning, as well as state-of-the-art algorithms for instance segmentation and pose estimation has been performed. Mask R-CNN has been modified to fit our purpose, then trained and evaluated. Mask R-CNN has also been implemented to run live inference on an Nvidia Tegra Xavier. The previous work related to instance segmentation has been extended, and a comparison of the previous and current results has been performed. PVNET has been implemented, modified, and evaluated. The dataset generation pipeline has been designed in order to be compatible with the training scheme of Mask R-CNN and PVNET, where Mask R-CNN is trained over six different classes, whereas PVNET is trained on four. All classes are present in the relevant section of the engine assembly pipeline at Volvo's factory. The data is constructed from rendered model files and is therefore purely synthesized. Scripts for camera calibration, pose projection and plotting scripts were created with the use of libraries from OpenCV, transforms3d, numpy and Matplotlib. Pre-processing of data involved the use of pythons pickle library, PIL, and OpenCV. Visual debugging tools were created with the Matplotlib, PIL, and OpenCV library. The neural network frameworks PyTorch, Tensorflow, and Keras, were used for modifying the neural networks. The Blender Python API was used to generate datasets for the two different networks. Object key points were selected with the FPS algorithm proposed from PVNet's library.

1.8 Ethical & Sustainability Aspects

Viewpoints regarding ethics and sustainability are needed to be taken into consideration. In this section, two aspects have been identified. The first is directly connected to this project, and the second one is more linked to the long-term goal, which this thesis is a part of.

1.8.1 Personal data privacy

When deploying a quality control system using cameras, workers at the factory will find themselves being recorded, which might compromise their privacy. The data is, however, not stored at this instance of the project, but merely evaluated by the developed algorithms. To evaluate algorithms during the development, the data saved might not be GDPR compliant without individually issued contracts.

One solution to this problem is to have representatives from the workers union at the negotiation table while developing the necessary precautions for setting up the system in the factory. This is done by an assigned team at Volvo and falls outside the scope of this master thesis.

1.8.2 Meaningful workdays

A substantial amount of time in people's life is dedicated to their workdays. For people to cope with their work, in the long run, they need to find meaningfulness in what they do. According to UN Committee on Economic, Social and Cultural Rights, (CESCR) [28], this extends to a part of one of the Sustainable Development Goals set by the UN where it reads as follows:

> "rights also include respect for the physical and mental integrity of the worker in the exercise of his/her employment."

This master thesis is a small piece in the process of further automating tasks throughout the assembly line. This automation might, in the long run, simplify the factory workers' task to an extent where they do not feel enough cognitive satisfaction. Depending on the person, the workers might also feel intrigued by being able to work with robots. Another aspect of this is that collaboration with robots might be a more complex task than it currently is, where humans and robots work separately. This might increase the level of cognitive abilities a worker needs in order to be able to perform the task in a sufficient manner. People who are unable to attain this level of skill would then fall outside the category of employable workers. On the other hand, such a system could also enable people with physical disabilities to be included in the workforce, thus making society more inclusive for them. Since this is not a direct consequence of this master thesis, a solution to this matter will not be taken into consideration in this project.

2 Introduction to machine learning

Since the methods used throughout the thesis utilizes neural networks, theory related to machine learning and neural networks is explained in this chapter. Further, some common frameworks are also introduced.

2.1 Supervised and unsupervised machine learning

Two common categories of learning styles in machine learning are supervised learning and unsupervised learning. In supervised learning, both the target output, i.e., the labeled data, and the input is fed into the network at the training stage. The model is fitted to the relationship between the input and target output in an iterative manner. A validation set is used in order to determine the error on new data, as is an indication on how well the network generalizes. Depending on the type of the target variable, the network is either facing a classification or a regression problem as illustrated in Figure 2.1. All machine learning algorithms in this master thesis belong to the supervised learning class.

In unsupervised learning, there are no labels or ground truth data, as shown in Figure 2.2. The key idea is to learn patterns and similarities on some data set, which then are grouped depending on their different similarities. Depending on which aim the network should achieve, the problem class can be separated into clustering or association problems. Clustering is grouping a set of data in such a manner that all of the objects in the same group are more similar than to those belonging in a different group. The association deals with finding consecutive links between data that does not necessarily make them similar.

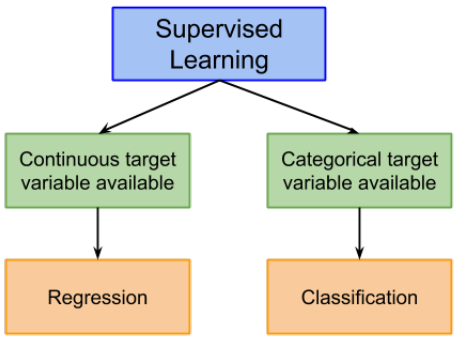

Supervised Learning
Continuous target variable available
Categorical target variable available
Regression
Classification

Figure 2.1: In supervised learning, the training is done with ground truth labels.

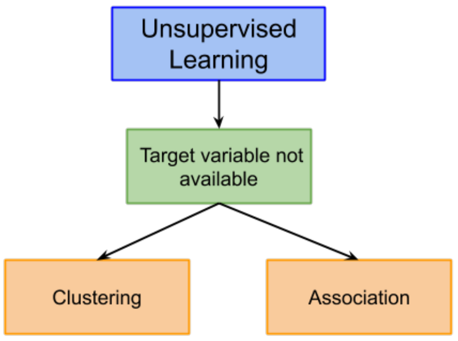


Figure 2.2: In unsupervised learning, the network does not have access to any ground truth.

2.2 Neural networks

Neural networks are comprised of algorithms that recognize trends and patterns in data. They are loosely modeled after the human brain and performs well in generalization tasks. Neural networks are already being used today in an increasing number of applications. An example of a simple neural network is shown in Figure 2.3. The input layer is fitted to the set from which to extract data. The hidden-layers in the middle are the means of extracting the most useful features and trends that exist in the input data. The output layers are the extracted information, typically in the form of a vector with probability values.

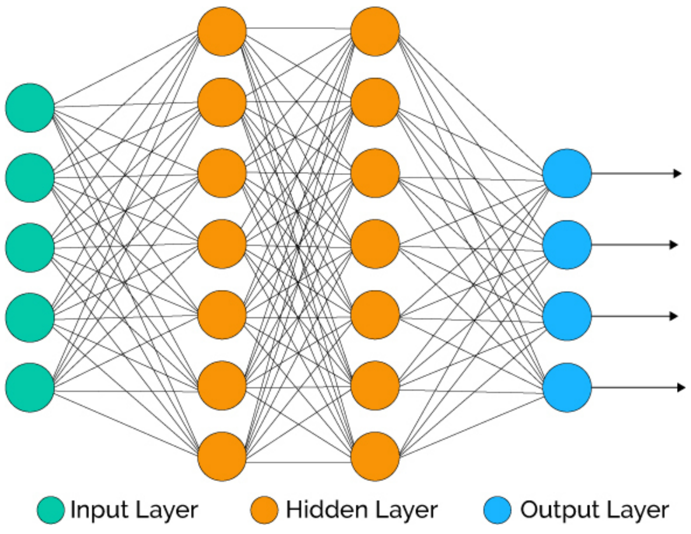


Figure 2.3: Example of the infrastructure of neural networks. [1]

2.2.1 Backpropagation

Backpropagation is used in order to update the weights in a neural networks. Looking at Figure 2.3. Each connection between different nodes inherently holds a weight.

At each node, all values from each connection are summed and an external bias is also added. These sets of weights and biases make up a function that predicts an output based on the input. If the network is well-trained it can handle new cases it has not seen before with a very good accuracy.

$$C_0 = \sum_{j=0}^{n_L - 1} (a_j^{(L)} - y_j)^2 \tag{2.1}$$

$$a_j^L = f(\sum_{k=0}^{n} \omega_{jk}^L a_k^{L-1} + b_j^L) = f(z_j^{(L)}) \tag{2.2}$$

The activation, denoted a_j^L, at the j-th neuron in the L-th layer are described with equation (2.2). The function wrapping the whole expression represents a type of activation function that normalizes the number between zero and one. The sigmoid, or hyperbolic tangents and rectifiers shown in Figure 2.4 are commonly used in the activation layers.

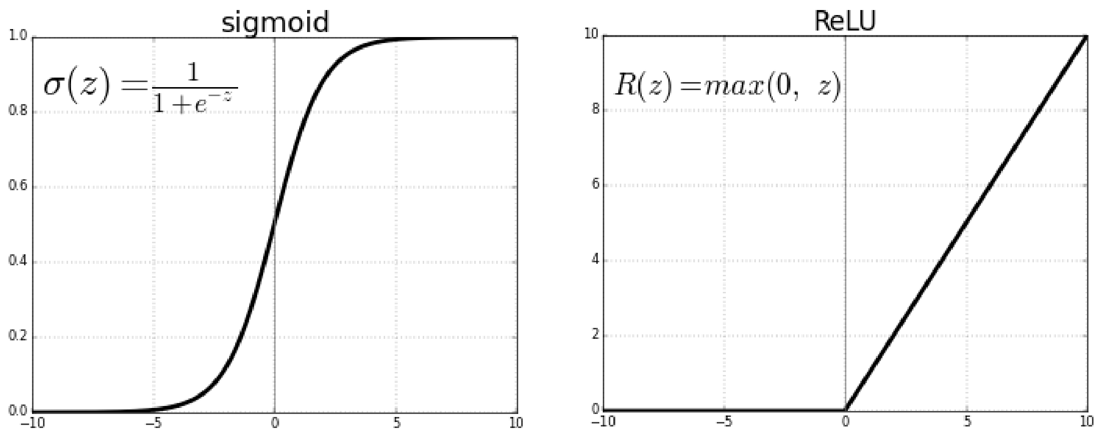


Figure 2.4: Activation functions: sigmoid and rectified linear unit.

Using the chain rule the sensitivity of the cost function with respect to the weight at the last layer is calculated by identifying the respective derivatives.

$$\frac{\partial C_0}{\partial \omega_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial \omega_{jk}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0^{(L)}}{\partial a_j^{(L)}} \tag{2.3}$$

Equation (2.3) describes how the chain rule can be used to describe the change in the cost function for a fully connected node in layer "L", with respect to the change in the weights between a single node "k" in layer "L-1" to another single node "j" in layer "L". Each term is the results calculated in equations (2.4), (2.5) and (2.6).

$$\frac{\partial C_0}{\partial \omega_{jk}^{(L)}} = 2(a_j^{(L)} - y) \tag{2.4}$$

$$\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = f'(z_j^{(L)}) \tag{2.5}$$

$$\frac{\partial z_j^{(L)}}{\partial \omega_{jk}^{(L)}} = a_k^{(L-1)} \tag{2.6}$$

L is the number of layers in the network, y is the desired output, b is the bias, j represents the number of the node in that layer. In in a classification setting, y in equation (2.2.1), would always either be a 1.0 or a 0.0, depending on what the ground truth actually is during training.

$$\frac{\partial C_0}{\partial \omega_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0^{(L)}}{\partial a_j^{(L)}} \tag{2.7}$$

Average over all the batches in one training set is calculated as such:

$$\frac{\partial C}{\partial \omega^{(L)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial \omega_{jk}^{(L)}} \tag{2.8}$$

As shown in equation (2.9), each term represents the cost functions sensitivity with respect to weights or biases. Each term will be calculated with respect to the weights between the last node and the second-to-last node, which again is affected by the nodes wired with that one.

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial \omega^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial \omega_{jk}^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix} \tag{2.9}$$

2.2.2 Tensors

Tensors are the primary data structure used in neural networks. A tensor can be formulated as a mathematical generalization of other more specific instances of other datatypes. A scalar, vector, matrix, and an N-dimensional array can all be generalized as a tensor with 0, 1, 2 and N-dimensions respectively. There are three dimensions present in an RGB image, height, width, and a color channel. This information is then stored as a tensor and passed into the network. In order for the neural networks to work with images of different resolution and aspect ratios, it is practical to generalize the functions handling the input tensors. One way to do this is to flatten the tensor. This is done by reshaping it to a one-dimensional array and then squeezing it, so it is considered as a tensor with only one element inside.

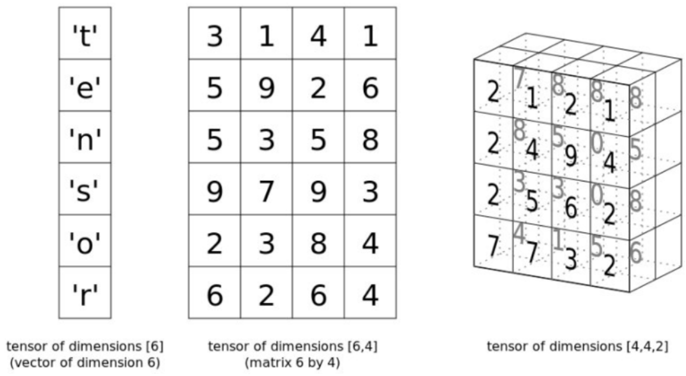


Figure 2.5: Illustration of different tensors. [2]

2.2.3 Deep neural networks

Deep neural networks differ from more shallow ones in the sense that they consist of more hidden layers. Hyperbolic tangents or sigmoids may result to vanishing gradients. In particular, these activation functions cluster the weights around zero and one, and thus the gradients will be almost zero in many cases.

The other extreme is exploding gradients, which happens when the solution surface is rapidly changing. A remedy to this problem is to implement a gradient checker that reduces the length of the vector if it is above a certain threshold. This method is called clipping and is a simple way of not overextending in search for the minimum.

2.2.4 Convolutional neural networks

Convolutional neural networks are biologically inspired models that work in a similar fashion to the mammalian visual cortex. The mathematical architecture has distinct local connections, layers and utilizes spatial invariance of visual information. Each layer is a mathematical kernel operation applied to the previous state of the image.

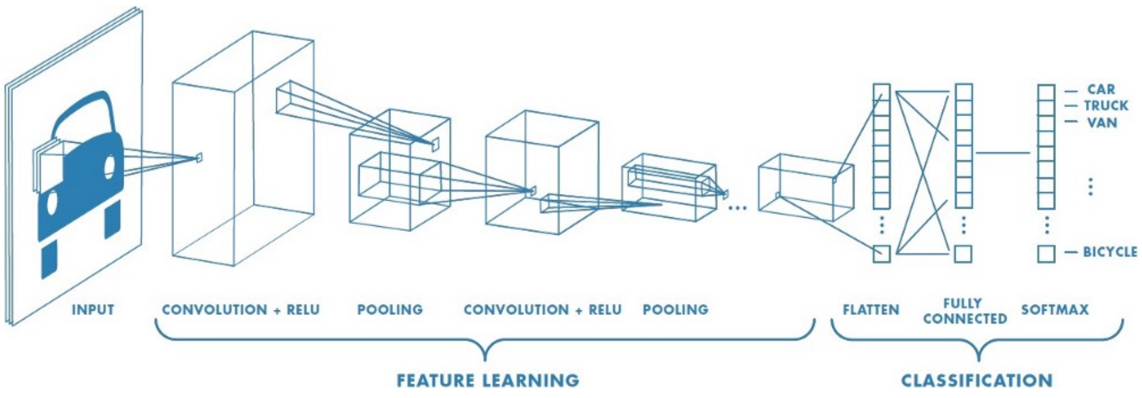


Figure 2.6: Illustration of a convolutional neural network. [3]

For object recognition networks, the first step is to load an image as the input. Then the size of a receptive field is chosen, and a convolution operation is performed in a sliding window manner. The result of this operation is the dot products of the kernel and the respective receptive field of the original image. The kernel size can vary in the first and second dimension, but will always need to match the depth of the input images. Secondly, a suitable activation function, like a rectified linear unit (ReLU), is used on the inputs to each node. Thirdly, a pooling operation is applied in order to extract essential features, while reducing the number of dimensions. These three operations are then repeated in the same sequence throughout the feature learning part of the network. The feature learning part will usually consist of more than three layers in CNN's, which corresponds to at least three rounds of convolution, activation and pooling. After the feature learning phase, comes the classification phase. The last layer is flattened and connected to a fully connected layer, which corresponds to the same number of classes that are to be detected. After the fully connected layer, the feature findings are converted to a probability distribution, softmax or similar, over all the classes. This enables the network to propose the class with the highest probability to the user of the CNN.

This architecture can further be altered to output the bounding box and mask of an object. The main difference is that gradual bilinear upsampling of the features is performed until the image has reached its original dimension. As illustrated in Figure 2.7, an image of a dog is classified and masked through a deconvolutional network.

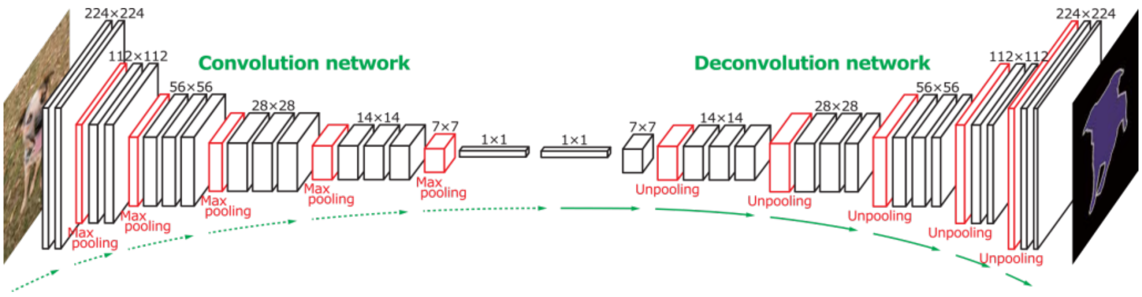


Figure 2.7: Convolution and deconvolution of a dog image. [4]

While a majority of all the parameters in a network consists of weights and constants, there are also some hyper-parameters needed to be determined before the training is initiated. Some of them are described in the list below.

Number of Classes is a measure of how many classes that the network is supposed to classify.

Kernel Size determines the size of the convolutional kernel / filter.

Padding is done by adding values, usually of zero, outside the borders and is done in order to retain the information that exists at the edges as well as hinder shrinking of the convolutional output. The size of the padding needed depends on the size of the convolutional kernel.

Stride describes the step size of the kernel operations.

Pooling is a way to reduce the complexity of the model while retaining most of the most important info after the convolution and activation operations are performed. Max pooling is popular because it creates sharp and relevant features. Average pooling and sum pooling are other methods. In addition to the pooling method, the size of the pooling is also important to define. Adapting the pooling size to match the input dimensions is also possible.

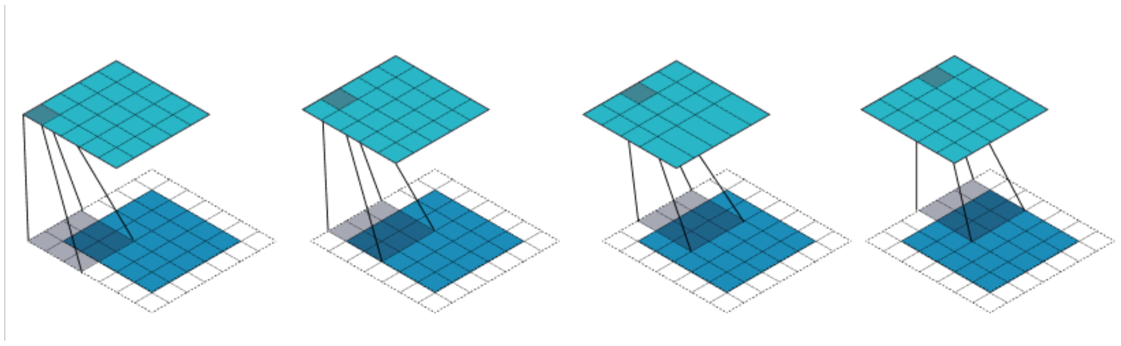


Figure 2.8: Convolution with a 3×3 kernel, stride of 1 and padding around the border. [5]

2.3 Training Neural networks

In order to train the network the data, optimizers, network architecture, batch size, learning rate and tensor handling framework etc. needs to be specified.

2.3.1 Optimizers

Optimizers are algorithms used for minimizing the loss function. The solution manifold is non-convex for all neural networks in supervised learning, because the activation functions are non-linear. This is why the complexity and shape of the solution manifold changes with each layer in the neural network. For deeper networks, the optimization has a higher global minimum convergence rate. In the sections below, some of the most popular optimizers are described.

Stochastic Gradient Decent and batch gradient descent are the most common and straight-forward optimizers. Stochastic gradient descent is an iterative optimizer which calculates the steepest gradient direction and then steps in that direction in order to find the global minimum.

Adagrad is an optimizer that adapts different learning rates for individual features. This is especially good in scenarios where many input examples are missing, such as sparse datasets.

Adam stands for adaptive moment estimation, and is utilizing knowledge of past gradients to calculate the current ones. It adds a fraction of the previous gradients to the current update vector, effectively creating momentum.

L-BFGS is a limited memory modification of the well known Quasi-Newton method Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS), which works by approximating the differences of past gradients. The BFGS' Hessian approximation is calculated over the whole history of gradients, whereas the L-BFGS is calculated over a limited history of past iterations. The latter method is used in the context of neural networks because the number of parameters in the problem is often in the millions. The method is preferably implemented with trust regions or online search for ensuring some convergence.

2.3.2 Batch size

Combining all the gradient suggestions from the optimizer into a batch enables less frequent update of the weights as the vectors are combined into one over the entire batch. Although small batch size gradient descent gives empirically better generalization, a higher minibatch trains the network faster and allows for batch normalization. According to Masters and Luschi [29], Wilson and Martinez [30] a batch size should not exceed 32 when using batch normalization and without batch normalization, there are performance gains even at a size as small as two.

If there is high cooperation between the local and global gradient, online training is faster. The previous steps do then cancels out with respect to progressing towards the global minimum since they are all pointing towards the same direction. If the cooperation is low or unknown, however, the solution manifold is not very smooth, and the solver will be more robust and have a higher chance of convergence if batches are used.

2.3.3 Training set size

A general rule of thumb is to have at least ten times as many data points as degrees of freedom in the model. This is, however, heavily dependent on the complexity of the model and if regularization is used. With the use of data augmentation, such as re-sizing, flipping, rotating of other pixel-wise operations, the size of the data set can be increased. This is also a strategy that can be used in order to make the neural network more confident and robust in its generalization.

2.3.4 Regularization

Memorization is not the same as learning, so considerations need to be made with regards to training a model. In the sections below, there are two ways of regularization, which will ensure that one will end up with a model that should be aptly fit for the task at hand.

Early stoppage can facilitate the process of ensuring that the network is not overfitting, a portion of the data set can be excluded from the training as a validation set. This data will not be trained on, but rather fed forward through the network within fixed intervals while training. If the validation error diverges, it is a sign that the model has crossed the optimum between generalization and specification.

Dropout is a method proposed by [31] that enables and disables nodes in a layer. This is done in order to force the network to discover new paths of logical flow, thus increases generalization and prevents over-fitting. The probability of dropout is generally below 50 percent, and most newer deep learning networks are utilizing one dropout layer.

2.3.5 Normalization

Normalization is a technique often applied as part of data preparation at the input level of the model. The goal is to change the values of numeric columns in the dataset to a common scale, while keeping the differences in the ranges of values. Two types of normalization techniques commonly used are listed below.

- **Batch Normalization** is based on the mean and standard deviation of the datapoints in the batch.

- **Batch Renormalization** tries to find a global normalization based on both the training data and the inference data. This is achieved by keeping a moving average asymptotically approaching a global normalization.

2.3.6 Transfer learning

Transfer learning is the act of using a network that has been trained to perform a similar task and adapt and retrain the last layers for the new problem. Successfully doing this reduces the computational effort of training the network for the new task, but it also has the potential to yield better performance when, for instance, the amount of training data is limited or restricted.

The model used as the starting point for transfer training is called a backbone network. When choosing a backbone network, the main tradeoff is usually the computational complexity of the network during inference, the size of the last layers when retraining and how well the network performs on a similar task. In Figure 2.9 different models with their respective accuracy on the image classification task can be observed. The size of the circle is the number of parameters in the form of weights and biases. This number is directly linked to the computational complexity of retraining it, and, as can be seen in the figure, increasing the numbers of parameters does not necessarily mean that the network has better performance on a specific task.

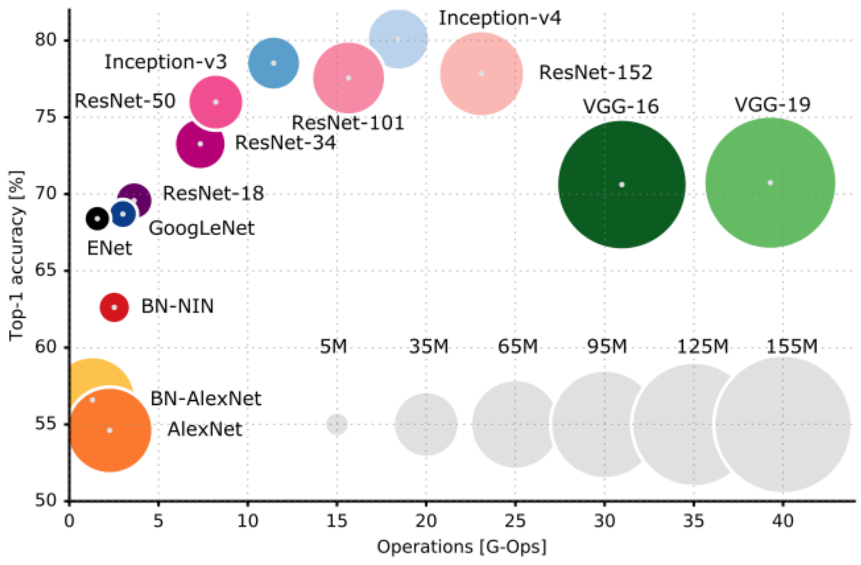


Figure 2.9: Overview of existing models for image classification. [6]

2.3.7 Loss functions

It is essential to use appropriate loss functions in order to evaluate how well the network performs, and that it is learning from the training set. Commonly through stochastic gradient descent, the network uses this function for optimization when searching for a local minimum. The most widely used loss functions in machine learning can be categorized as either a classification loss or a regression loss. Both of these categories contain a broad set of widely used functions. Illustrated in Figure 2.10 are some of the most common classification loss functions. Likewise, in Figure 2.11 are some of the most common regression loss functions. Each of them is further explained in the sections below.

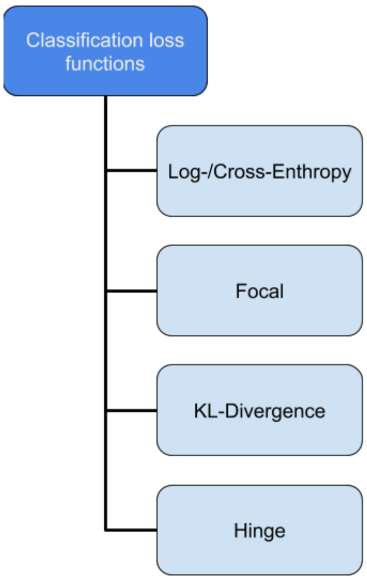


Figure 2.10: Some of the most common classification loss functions.

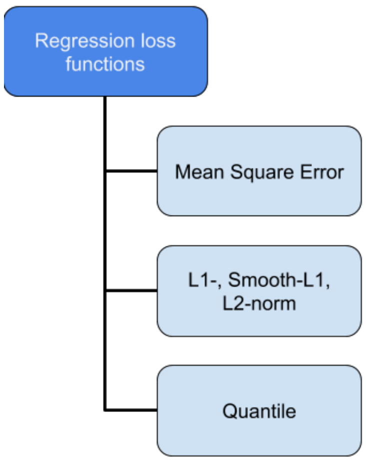


Figure 2.11: Some of the most common regression loss functions.

Classification loss functions are usually applied to machine learning problems where the predictor is applied to classification problems. A classification problem is a problem where the input is mapped to a discrete set of categorical output variables. A simple example would be: given an image as input, is the image of a cat or a dog? Here the categories is a vector of length two containing the classes "cat" and "dog", but the categorical vector can be of any size.

Log/Cross-Entropy loss is a logarithmic loss function that measures the performance of a particular classification. It is defined in equation (2.10), where $y_{o,c}$ is the ground-truth of a given class i.e. a one-hot vector, and $y^p_{o,c}$ is a prediction given this class. Given a prediction and some ground truth, the cross-entropy loss heavily penalizes false but certain predictions, which is further illustrated in Figure 2.12 below.

$$L = \sum_{c=1}^{M} y_{o,c} \log(y^p_{o,c}) \tag{2.10}$$

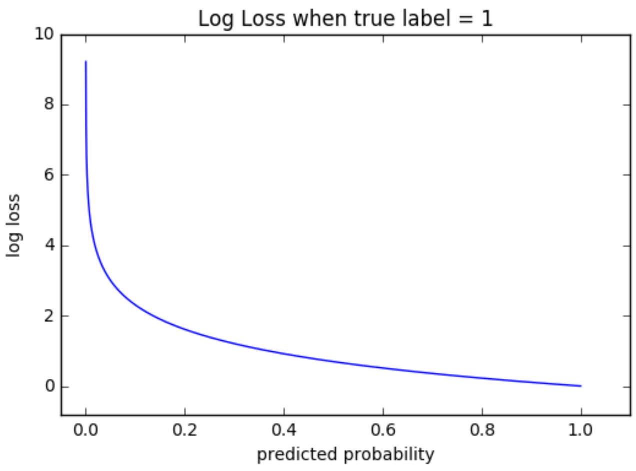


Figure 2.12: The Log loss function heavily penalizes the network if it predicts the class with a low probability. [7]

Focal loss was derived by Facebook AI Research in order to improve the performance of the Cross-Entropy loss [32]. In contrast to the Cross-Entropy loss, the Focal loss penalizes the network depending on how hard the example is to classify. Given a binary classifier for simplicity, the Focal loss is defined in equation (2.11). As before in the Cross-Entropy, the $y_{o,c}$ is the ground-truth of a given class and $y^p_{o,c}$ is a prediction given this class. The γ is a tuning parameter, obtained through experiments. Typically, the classification of the foreground is harder than the classification of background. For applications like instance segmentation, the γ is usually set to a value below one.

$$L = -(1-p_t)^{\gamma}\log(p_t) \text{ where } p_t = \begin{cases} y^p_{o,c} \text{ if } y = c \\ 1 - y^p_{o,c} \text{ otherwise} \end{cases} \tag{2.11}$$

KL Divergence or Kullback–Leibler Divergence *KL Divergence* is an estimate of how well samples drawn from one distribution matches a query distribution or a model. The exact definition is defined in equation (2.12), where $p(x)$ is the sampled distribution and $q(x)$ is the model. If the distribution of the model matches one of the sampled distribution, then the KL Divergence becomes zero. Analogous to this is that, if the mismatch between the distributions is large, the KL divergence will approach one.

$$L = \frac{1}{N}\sum_{i=1}^{N} y_i \log(\frac{y_i}{y^p_i}) \tag{2.12}$$

Hinge loss mainly penalizes incorrectly but certain classified outputs. It is defined as equation (2.13), where the α term is the boundary for where the loss should become zero. This means that if the output is correctly classified with a probability over the threshold *alpha* the loss is zero, otherwise it is linear. An illustration of the hinge loss can be seen in Figure 2.13.

$$L = \frac{1}{N}\sum_{i=1}^{N} max(0, \alpha - y_i y_i^p) \tag{2.13}$$

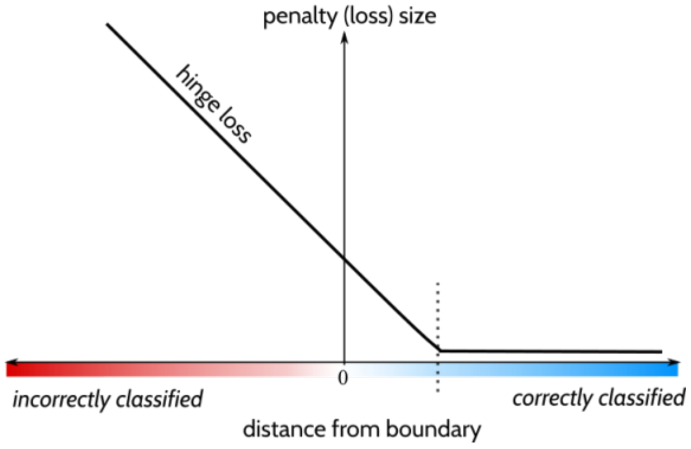


Figure 2.13: The hinge loss increases linearly when the classifier is incorrectly certain about a predicted output. [8]

Regression loss is different from the classification loss in the sense that yields a prediction distribution, whereas the regression losses are quantitative. Since most of them are continuous rather than discrete and are differentiable, their derivative can usually be used for backpropagation. Some of the more common ones are explained further in the sections below.

Mean Square Error *(MSE)* is one of the most commonly used regression loss functions, not only in machine learning but it is also widely used in System Identification other areas where you want to fit some mathematical model to a situation. It is defined as (2.14), where y_i is the ground truth and y_i^p is what the model predicts.

$$L_{mse} = \frac{1}{N}\sum_{i=1}^{N}(y_i - y_i^p)^2 \tag{2.14}$$

A visual toy example of this is illustrated in Figure 2.14. A model is fitted, in this case, a straight line in blue to a set of points in green. The straight red dashed lines between the points and the model are the errors which are to be minimized. Squaring the errors negates the possibility of cancellation from a difference in signs in the summation. The Mean Absolute Error *MAE* achieves this by using the modulus of the errors. One feature with the MSE that can be somewhat problematic is that if just some of the points are relatively far away, this metric gets heavily penalized.

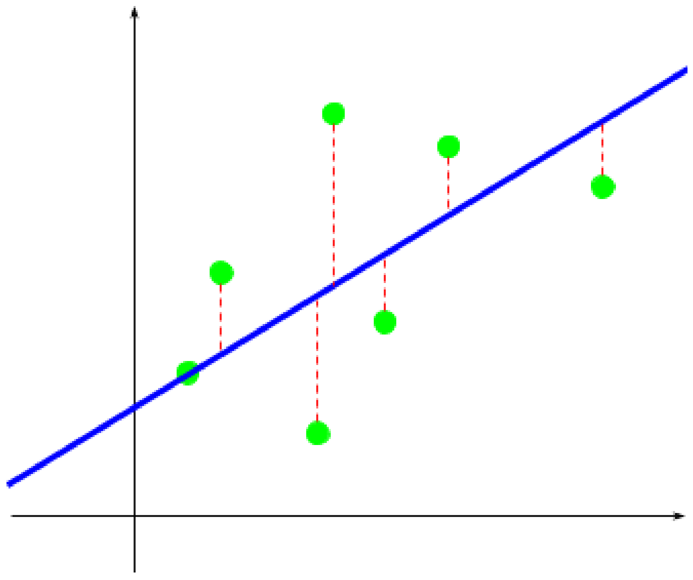


Figure 2.14: An illustration of the MSE.

Least Square Regression and Least Absolute Deviation differs from MSE, the Least Square Regression *LS* and Least Absolute Deviation *LAD* because they are not divided with the number of errors in the sum, but in other aspects, the idea is the same. Three of the most commonly used loss functions in machine learning are the L1-norm, the Smooth L1-norm, and the L2-norm. The L1-norm is defined in equation (2.15) and the Smooth L1-norm is defined in (2.16). Compared to the L2-loss function, defined in (2.17), the loss is, for the most part, linear and thus the L1-loss functions do not penalize large errors significantly more than smaller ones. The difference between the L1-norm and the Smooth L1-norm is that as the errors approach zero, the Smooth loss becomes similar to the L2-norm. Doing this removes the discontinuous part, thus enabling the gradient to be calculated in the entire domain of the function. Since stochastic gradient descent is so commonly used for backpropagation in the training of a neural network, the Smooth L1-norm is preferred over the L1-norm.

$$L = \sum_{i=0}^{N} \mid y_i - y_i^p \mid \tag{2.15}$$

$$L = \sum_{i=0}^{N} L_i \text{ where } L_i = \begin{cases} |y_i - y_i^p|, & \text{if } x > \alpha \\ \frac{1}{|\alpha|}(y_i - y_i^p)^2, & \text{if } x \leq \alpha \end{cases} \tag{2.16}$$

$$L = \sum_{i=0}^{N} (y_i - y_i^p)^2 \tag{2.17}$$

In Figure 2.15, the differences between the LS L2-norm and the two L1-norms are illustrated further. As can be seen, outside the bounds of α, the two L1-norms are the same while inside the bounds of α the general trajectory of the Smooth L1-norm, and the L2-norm are equivalent.

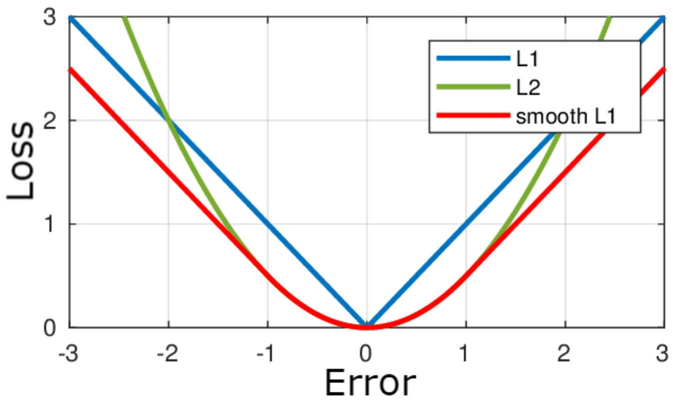


Figure 2.15: The L1-norm in blue, the smooth-L1-norm in red and L2-norm in green. [9]

Quantile loss. The regression loss functions mentioned above assume that the residuals have a constant variance, but that is not always the case. The Quantile loss function is defined in (2.18) and can adjust how the model should handle positive respectively negative errors with a scaling factor γ, thus adjust the effect of non-normally distributed errors. In Figure 2.16, an experiment using a small and a large γ with a linear regressor are illustrated. In the figure, the error covariance is not normally distributed, and thus the slope of the regressor is adjusted by tuning the γ term in equation (2.18).

$$L = \sum_{i=y_i<y_i^p} (\gamma - 1) \mid y_i - y_i^p \mid + \sum_{i=y_i \geq y_i^p} \gamma \mid y_i - y_i^p \mid \tag{2.18}$$

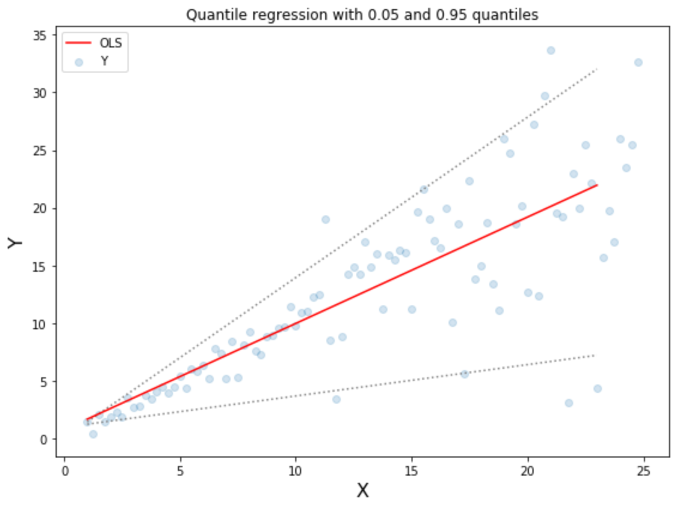


Figure 2.16: Example of a linear model fitted to a distribution where the error covariance is not normally distributed.

2.3.8 Overfitting and cross-validation

Since a neural network essentially is a multivariate function with many thousands, if not millions, of degrees of freedom that tries to fit a mathematical model to data, there is a significant risk of overfitting. Illustrated below in Figure 2.17 is an example where one can see how increasing the degrees of freedom does not necessarily mean that the model fits the true trajectory better. As can be seen, a 4'th degree polynomial fits the true trajectory, whereas a 15'th degree polynomial does not. If the validation loss acts as a stochastic random variable, then there are some erroneous parts, either the network itself or the training data.

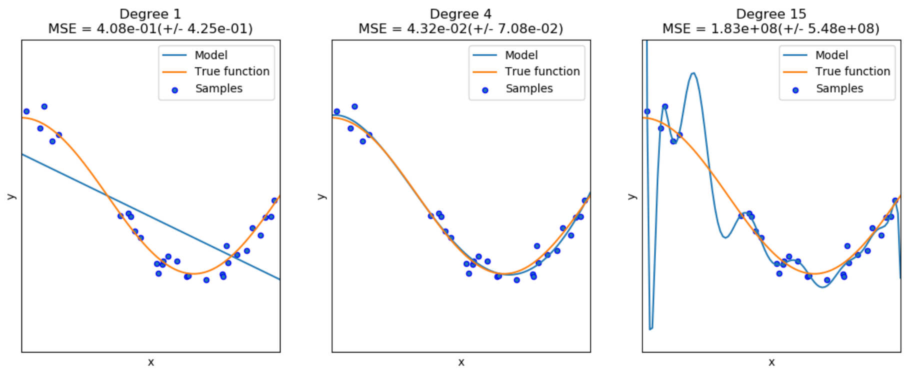


Figure 2.17: Example of the overfitting problem where a blue line is fitted to an orange line through some measurements. [10]

When training a network, there is a substantial risk of overfitting, even when precautionary efforts are made to prevent it. The loss of the training set might show a good trend, but the validation set is showing the opposite trend. In Figure 2.18, an example of overfitting is illustrated. Even though the accuracy over the training set increases i.e., the loss decreases and the network performs worse on the test set. The best performance is achieved after about 800 epochs in this case, as the test set accuracy decreases (the loss increases) from that point.

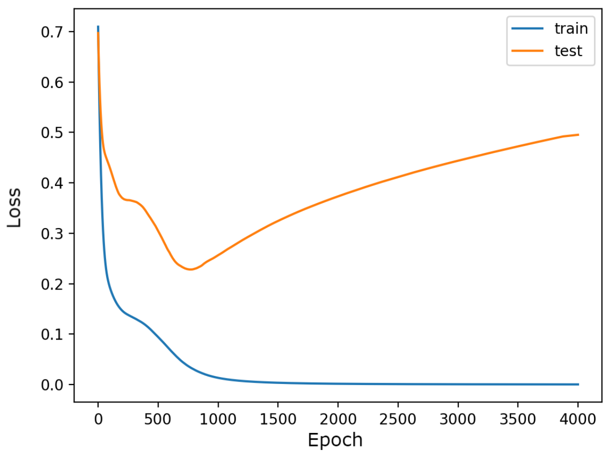


Figure 2.18: Example of divergence of the validation loss. [11]

2.4 Frameworks

There are many frameworks in machine learning, some of which are described in the followin chapters.

2.4.1 TensorFlow

TensorFlow is a machine learning application program interface (*API*) developed by Google Inc. As the name implies, the framework revolves around computations with vector and matrix generalization, i.e., tensors. It was built with scalability in mind and is, since 2017, both free and open source under the Apache License 2.0. Using a language with low abstraction capabilities, namely C++, the developers have been able to make the API optimized to a variety of different hardware architectures. The API also has support for many other languages, where one of the most used is Python. The factors above make TensorFlow extremely popular for solving many tasks related to machine learning using neural networks. Since neural networks use linear algebra extensively, there are significant speedup gains to be made by making the computations on one or multiple graphics processing units (*GPU*). By default, TensorFlow prioritizes the resource allocation of the GPU given that the operation has support to be executed both on the GPU and the central processing unit (*CPU*).

There are two key components of the TensorFlow API. The first one being a library that is used to build functions as computational graphs, whereas the second one executes said graph. A computational graph is simply an abstract way of describing a directed graph where each node represents a function. In TensorFlow such graphs are called *data flow graphs*.

2.4.2 TensorBoard

TensorFlow also comes with a toolkit for visualization of data, namely TensorBoard, and is automatically included when installing TensorFlow. Once a network is set up, essential variables can be tracked either offline or online and be visualized through a web-browser. Commonly the training loss and validation loss is the data of most interest, but since any data can be tracked with minor coding effort, it is also a powerful tool for troubleshooting. TensorBoard also has the capabilities of visualizing the entire execution as a connected graph, where the nodes are the computations, and the edges are the data, similar to how the execution carried out in TensorFlow. This feature is also a useful tool for troubleshooting why a network might behave unexpectedly.

2.4.3 Keras

Using TensorFlow at its core, the high-level application programming interface (API) Keras was developed in Python. Initially, Keras was developed as a part of the research effort of project *ONEIROS*, which stands for open-ended Neuro-Electronic Intelligent Robot Operating System. The framework focuses on being modular, easy to use, and free; thus, it has become widely popular both within research and non-professional users. The downside, however, is that the Keras library is not as customizable as the libraries on which it is built.

2.4.4 PyTorch

PyTorch is a machine learning library for Python based on the Torch library. It was developed by FAIR in 2016 and is both free and open-source under the Berkeley Software Distribution (BSD) license. In PyTorch, the programming paradigm is imperative rather than sequential and thus not as intuitive. Imperative programming describes how the state of a program should change, and no variables exist unless a function calls them. This is beneficial during back-propagation since this utilizes a dynamically built graph; hence, the model can edit itself on the fly with the aim to improve accuracy.

3

Introduction to computer vision

In this chapter, additional theory used, that isn't explicitly related to machine learning is presented.

3.1 The camera model

A camera provides a projective mapping of the world and is an essential tool in computer vision. To be able to dissect an image and extract useful information about how pixels on the image relates to points in the 3D space, a model needs to be made. The most common model to use it the Camera Obscura- or the Pinhole Camera Model, which provides a one to one mapping of the rays on to an image plane. The conceptual illustration of this model is shown in Figure 3.1. The more formal representation is illustrated in Figure 3.2. Here, the point P is projected to the point P' located on the image plane Π. For this simple model all projections are linear mappings through the pinhole, or center of the camera O. The image plane is located with its center C' aligned with the k-axis of the pinhole at a distance f. This distance is also known as the focal length of the camera. The mathematical relation to this is explained further in Section 3.2 below.

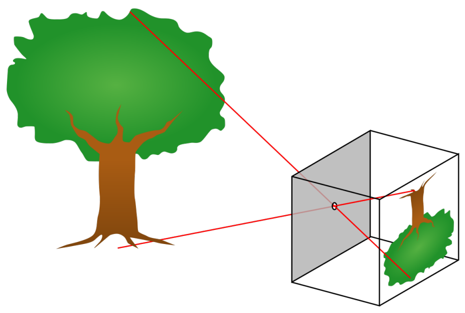


Figure 3.1: The pinhole camera model, the tree in the real world are vertically flipped in the image plane. [12]

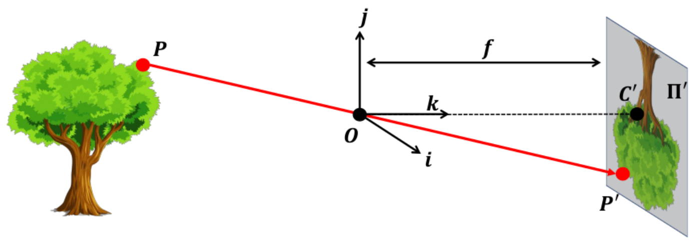


Figure 3.2: The point P in the 3D world is projected on to the image plane Π through the pinhole O. [13]

3.2 The camera matrix

The camera matrix gives a mathematical relation to how a pixel in the image relates to the world. It can be decomposed into two separate categories, the intrinsic matrix, and the extrinsic matrix. All parameters contained within the camera are the intrinsic parameters and are usually different for different cameras, whereas the extrinsic matrix is independent of the camera model. One way to think about it is to imagine that all of the information to the right-hand side of the center of the camera center in Figure 3.2 is contained within the intrinsic matrix, whereas the extrinsic matrix captures where O is and how it is oriented. Formally, the camera matrix is defined in equation (3.1), and as can be seen, it is composed of the product of the intrinsic multiplied with the extrinsic matrix.

$$P = \underbrace{\begin{bmatrix} K \end{bmatrix}}_{\text{Intrinsic Matrix}} \times \underbrace{\begin{bmatrix} R \mid t \end{bmatrix}}_{\text{Extrinsic Matrix}} \tag{3.1}$$

The intrinsic matrix captures the properties of the camera; that is how the properties of each pixel relate to the real world. There are three properties related to this, shearing, scaling, and translation.

- **2D Shear:** This is usually zero, but if the pixels in the sensor are not perfectly squared due to some filter or manufacturing error, this value will be something else than zero.
- **2D Scaling:** The scaling tells how each of the individual pixels scales to the real world via the focal plane.
- **2D Translation:** The translation of the axis in the camera frame perpendicular to the image plane. This axis is usually set to be centered in the image.

The mathematical representation of these properties is shown in equation (3.2) below where the composition of the intrinsic K matrix is shown in its entirety.

$$K = \underbrace{\begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{2D Translation}} \times \underbrace{\begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{2D Scaling}} \times \underbrace{\begin{bmatrix} 1 & \frac{s}{f_x} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{2D Shear}} = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

The extrinsic matrix, on the other hand, captures the location and orientation of the camera in the world frame. It is defined as a rotation matrix in the world frame made 4×4 by adding a row and a column multiplied with a translation matrix in the world frame made 4×4 as illustrated in equation (3.3).

$$\left[\begin{array}{c|c} R & t \end{array}\right] = \underbrace{\left[\begin{array}{c|c} I & t \\ \hline 0 & 1 \end{array}\right]}_{\text{3D translation}} \times \underbrace{\left[\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array}\right]}_{\text{3D Rotation}} = \left[\begin{array}{ccc|c} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{array}\right] \tag{3.3}$$

When combining all of the matrices explained above, the final camera matrix P is composed. To further demonstrate how the intrinsic and extrinsic parameters relate to each other, the image and the world, an illustrative figure is shown in Figure 3.3. As the image shows, the coordinate frames are not the same, and it is the camera matrix that provides the information of how they are related.

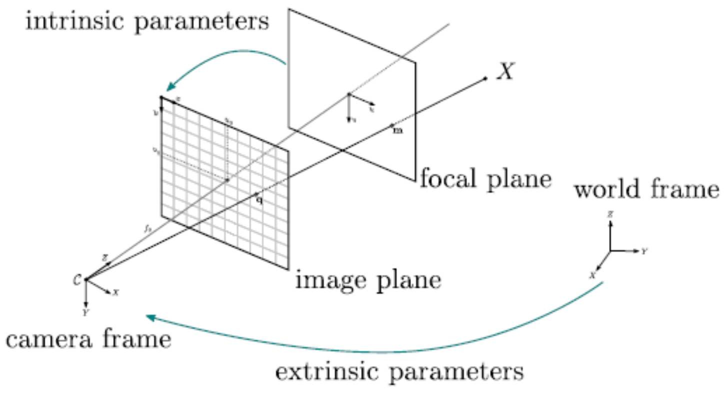


Figure 3.3: A visual representation of how the intrinsic and extrinsic parameters related to the worlds frame coordinate system. [14]

3.3 PnP

Given a set of 3D points and their corresponding location in an image, the PnP algorithm tries to solve the relative 6D pose between the calibrated camera and that object. This algorithm is utilizing the perspective projection matrix shown already in equation (3.1) to project the 3D points into 2D or vice versa, as shown in equation (3.4).

$$s \underbrace{\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}}_{p_c} = \underbrace{\begin{bmatrix} f_x & \gamma & c_w \\ 0 & f_y & c_h \\ 0 & 0 & 1 \end{bmatrix}}_{\text{K}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & r_3 \end{bmatrix}}_{[R|t]} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{p_w} \tag{3.4}$$

Where p_c is the image coordinate of homogeneous image point, K is the matrix of intrinsic parameters, $[R|t]$ is the matrix of external parameters and p_w represents the homogeneous world or object point. The majority of the solvers assume that the camera is already calibrated, the points correspondences can not be colinear and that the results might yield ambiguities which might need post-processing. Due to it's many use-cases for computer vision, XR and robotics, this is a field that has been extensively studied and therefore many variations of the solver have emerged. The implementations range from the most minimal form of **P3P** to **EP*n*P** and P*n*P using **RANSAC**.

3.4 Hough Voting

The Hough transform is a feature extraction technique commonly used in image analysis and computer vision. The purpose of the technique is to find imperfect instances of objects within a specific class of shapes by a voting procedure. The object candidates are obtained as local maxima in the space of the Hough transform, and the goal of doing this is to capture as much information as possible with the minimum descriptive effort.

3.5 Mahalanobis distance

The Mahalanobis distance is a measurement of how close a point is with respect to a probabilistic distribution. Commonly used for detecting outliers it can also be used for multidimensional probabilistic regression, making them suitable for neural networks as well as pose regression. The Mahalanobis distance between an observation $\vec{x} = \{x_0, x_1, \ldots, x_n\}^T$ to an existing cluster of data points with mean values $\vec{\mu} = \{\mu_0, \mu_1, \ldots, \mu_n\}^T$ is calculated as shown in (3.5), where **C** is the covariance matrix.

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^T C^{-1} (\vec{x} - \vec{\mu})} \tag{3.5}$$

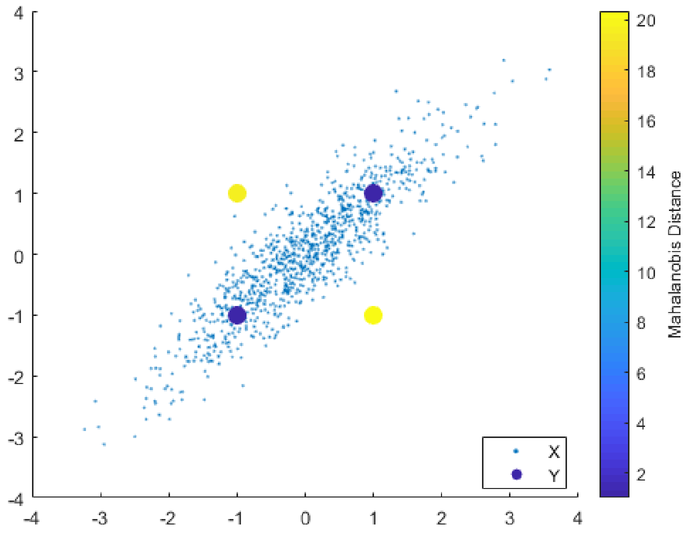


Figure 3.4: An example of four points with same euclidean distance to (0.0 , 0.0), but very different Mahalanobis distance

4
Recognition, pose estimation and synthetic data

In this chapter, the recognition problem and the pose estimation problem is stated. The algorithms Mask R-CNN and PVNet, as well as methods for manipulating synthetic data, are explained.

4.1 Recognition

There are four important recognition algorithms that are used extensively in this master thesis. These are classification, localization, semantic- and instance segmentation. In the subsections below they are further explained.

4.1.1 Classification

The classification problem is a central topic in supervised machine learning where the data is separated through predictions on belonging to a specific predefined class. The unsupervised version of this problem is called clustering, as explained above. A classifier is simply an algorithm or a mathematical function that maps some input data to a category of classes. A toy example of the classification problem is; given an input image, is this an image of a cat? Here there is only one class, namely "cat" and the output of the network is binary value. If the classification is extended to span multiple classes, this problem is a multiclass classification, and the output is a binary vector. Another important instance of the classification algorithm is the multilabel classification. This means that the classifier is able to separate different instances of the same class. The output is then a vector of a number corresponding to how many instances of a particular class exists in the image rather than a binary vector.

4.1.2 Localization and classification

In addition to classification, a localization algorithm also localizes where the target is in an image. It is always bundled with classification, and if the network is able to detect and localize multiple classes of objects in the same image, this is called object detection. In Figure 4.1 below, three sheep are both classified and localized.

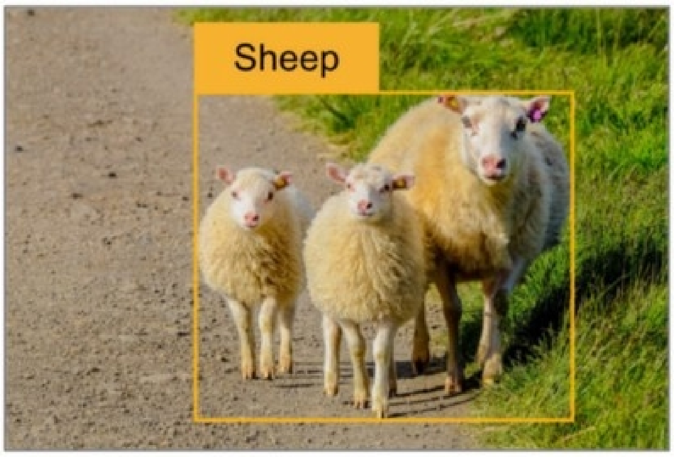


Figure 4.1: Example of classification and localization of sheeps in an image. [15]

4.1.3 Semantic segmentation

In semantic segmentation, each pixel in an image is associated with a distinct class in a set. Individual objects of the same class are not separated from each other but instead joined together in the same set of pixels. In Figure 4.2 below, all the pixels belonging to the sheep class are in the same set of classified pixels. The same goes for the grass and the road; thus, it is impossible to say if the pixels belong to different instances of the same class or not.

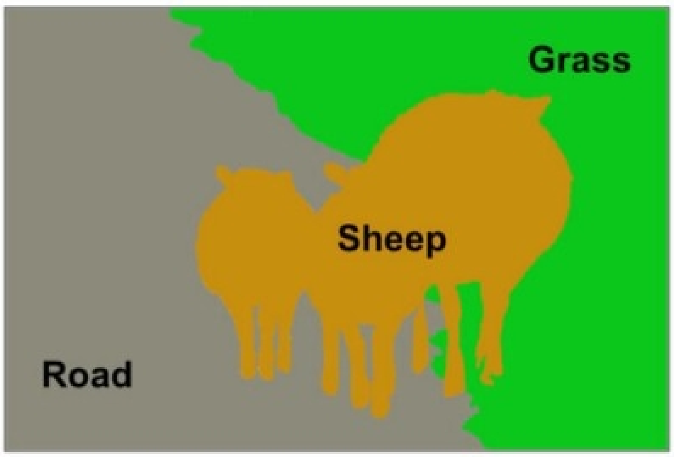


Figure 4.2: Example of semantic segmentation of sheeps in an image. [15]

4.1.4 Instance segmentation

In instance segmentation, each pixel is labeled with a specific instance of the different classes. This algorithm is thus able to make a distinction between two or more objects of the same class. In Figure 4.3, this is illustrated as the different sheep are assigned different colors and numbers.

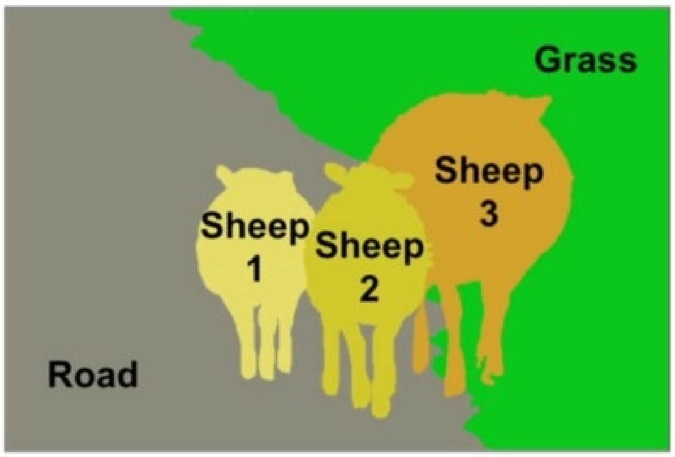


Figure 4.3: Example of instance segmentation of sheeps in an image. [15]

4.1.5 Related work

Before neural networks became the standard of solving recognition tasks in images, many computer vision approaches have been tried with varying success. Thresholding-based segmentation, for instance, partitions the image histogram into two or more classes based on the characteristics of the histogram [33]. The threshold can either be set manually, by running numerous controlled tests or automatically by histogram clustering. Another method is Conditional Random Fields (CRF), where a statistical model is used to map each pixel to a class, taking neighboring pixels into account[34].

The standard computer vision approach, although usually computationally fast, never yielded a similar level of accuracy as neural networks. Segmentation based methods yield a segmentation mask before, or without region detection. In [35] the authors combine two different modules, or subnetworks, namely an initial semantic segmentation module together with a CRF to perform instance segmentation. This makes the segmentations compete with each other, so a more common approach in recent years is to create a mask and then classify it, as done in Mask R-CNN described below.

4.1.6 Mask R-CNN

In recent years the Facebook AI Research (*FAIR*) has been contributing a lot to the field of machine learning, and especially neural networks. One significant contribution is the network architecture Mask R-CNN, which is an extension of the Faster R-CNN developed in 2016. The Faster R-CNN is relatively fast, but it only proposes bounding boxes and thus are not made for instance segmentation. Mask R-CNN solves this issue by first using the Region Proposal Network from Faster R-CNN and then adding a branch for predicting an object mask in parallel with the bounding box prediction. The mask is generated for each region of interest and is not class specific, meaning that the segmentation mask is generated first and then labeled after the bounding box has been classified. This method speeds up both the training process and the interference because the pixel-wise mask prediction does not compete against other classes. The network can utilize many different backbone architectures, but the ResNet101 showed promising performance and is the one used in this thesis.

The Region proposal Convolutional Neural Network (R-CNN) solves the recognition task of localization and classification. It does so by using a process called selective search, where the network extracts regions of different sizes and from the image and tries to predict if they contain objects of interest. The regions are then re-sized into squares and passed forward in the network, where they are classified using an SVM. The final bounding box is generated through linear regression of the proposed regions. An illustration of this pipeline is presented in Figure 4.4 below.

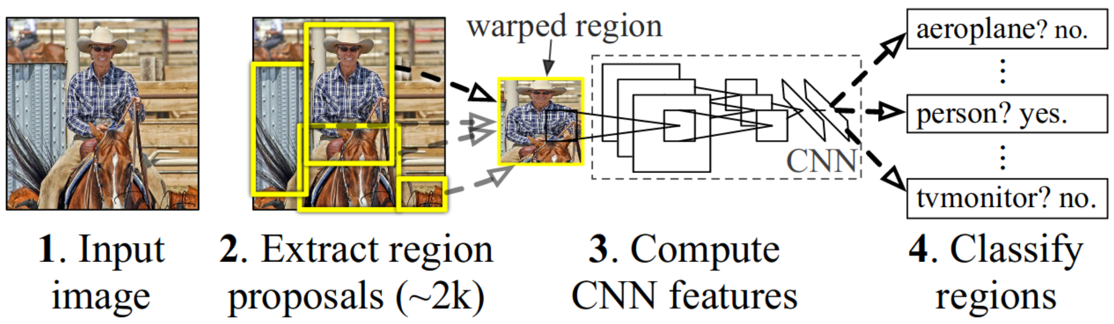


Figure 4.4: The pipeline from an input image, through the region proposal step and the classification step of the R-CNN using an SVM. [16]

In the R-CNN network, both the interference and the training are slow and tedious. For each image, about 2000 proposed regions were fed through an eight-layer network. It also utilized three different networks for different tasks, thus making the pipeline challenging to train. To decrease the computational complexity, the network Fast R-CNN was released in 2015 [17]. This uses a technique called RoI pooling, where the image is fed through a network that both propose RoIs and a feature map in one forward pass. Overlapping regions share the same computations, enabling a reducement in computational complexity by storing the feature map and extracting features from the saved values. In Figure 4.5 is a visual illustration of how the RoI and feature map are jointly created through a CNN. Furthermore, the different networks for feature extraction, classification, and regression were combined into a single network making training less of a tedious process.

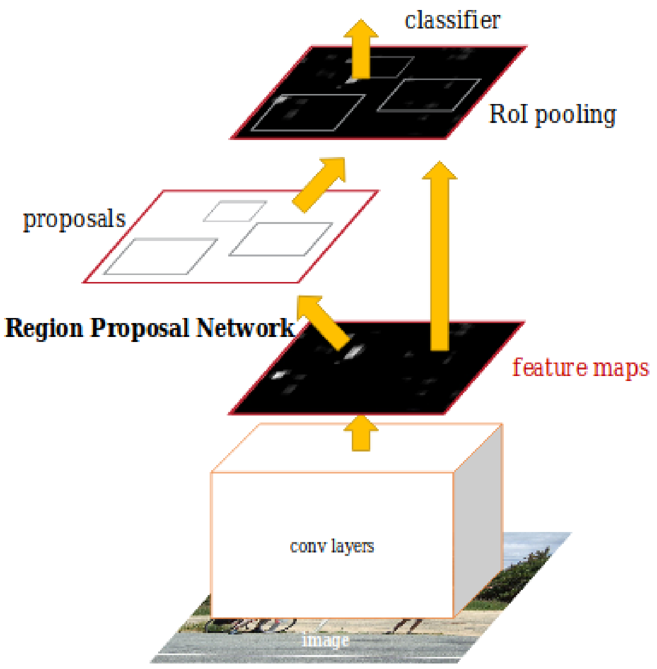


Figure 4.5: The convolutional layers in Fast R-CNN output both the region proposals and the feature maps. [17]

Mask R-CNN has two main features in addition to the Faster R-CNN. First of all, it adds a fully convolutional network that generates binary masks in parallel to the bounding box and classification from Faster R-CNN, as illustrated in Figure 4.6. Since the mask is not classified in this step, the pixels will not compete over different classes, thus simplifies the pipeline significantly. Secondly, it introduces a method called Region of Interest Align (RoIAlign). The RoIPooling layer from Faster R-CNN induces inaccuracies in the alignment of the features in the feature map. By removing the quantization of the coordinates connected from striding, the alignment was made more exact.

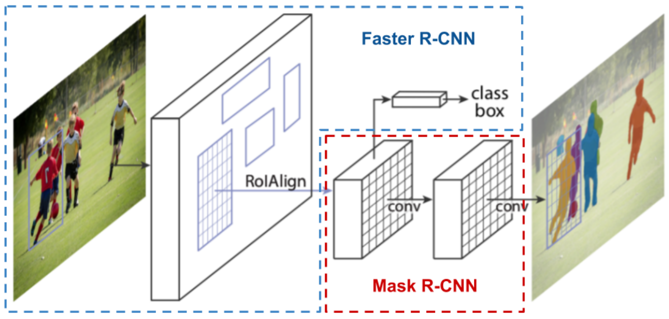


Figure 4.6: The Mask R-CNN network simply adds two convolutional layers in addition to the Faster R-CNN architecture. [18]

Loss functions

Mask R-CNN comes with five distinctive loss functions, with some of them inherited from the network architecture it is built. Depending on the application at hand, all of these are not equally important, and it is up to the designer to weight them appropriately when retraining the network. In this thesis, however, they have the same weights. All of these functions are explained further below.

Smooth-L1 loss Mask R-CNN, RPN, and Fast R-CNN all use the Huber loss, which is a special version of the smooth-L1 loss function explained in 2.3.7. In the case of these networks, they are defined as equation 4.1. Here x is the residuals of taking $y_i - y_i^p$.

$$L_{smooth} = \begin{cases} \frac{1}{2}x^2, & \text{if } |x| > 1 \\ |x| - \frac{1}{2}, & \text{otherwise} \end{cases} \tag{4.1}$$

Classifier loss The classifier loss classifies a set of defined regions as either background or foreground. Each of these regions is called anchors, and each anchor is compared to the ground truth bounding boxes in the data set. An anchor is classified as foreground, thus positive, if the intersection over union *(IOU)* of an object and the anchor point is above 70%. If the IOU is below 30%, it is deemed to be a negative anchor point, and if the IOU is anything in between it is considered to be a neutral anchor point and are discarded in the metric. The anchor point sizes and ratios are defined prior to training, where the usual aspect ratios are 0.5, 1, and 2.

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{cls}(t_i, t_i^*) \tag{4.2}$$

RPN bounding box loss The RPN bounding box loss uses the smooth-L1 loss function over predicted anchor point boxes together with the ground truth bounding box for a class. The class is specified as positive foreground, negative background, and neutral. Only the positive anchors contribute to the loss. It is defined as in (4.3) where y is the midpoint of the bounding box, together with the logarithmic size of the bounding box.

$$L_{RPNbbox} = L_{smooth-L1}(y_i - y_i^p), \text{ where } y = \begin{bmatrix} dx & dy & log(height) & log(width) \end{bmatrix}^T \tag{4.3}$$

Mask R-CNN bounding box loss

The Mask R-CNN bounding box loss is the loss after the bounding box refinement step and is defined as in (4.4). In this case, y is a vector containing the same parameters as in the RPN bounding box loss explained above. The total loss is then the mean of the smooth-L1 loss over the ground truth subtracted with the prediction.

$$L_{mrcnnbbox} = mean(L_{smooth-L1}(y_i - y_i^p)), \text{ where } y = \begin{bmatrix} x & y & log(h) & log(w) \end{bmatrix}^T \tag{4.4}$$

Mask loss The multitask loss is a binary cross-entropy loss, thus the level of uncertainty given a distribution y_{ij}. It is defined according to equation (4.5) where y_{ij} is the label of the cell in the true mask for the region of size $m \times m$, $\hat{y}^k_{ij}$ is the predicted value given the ground truth class k. The mask loss L_{mask} is defined only on positive RoIs.

$$L_{mask} = -\frac{1}{m^2} \sum_{1 \leq i,j \leq m} (y_{ij} log(\hat{y}^k_{ij}) + (1 - y_{ij}) log(1 - \hat{y}^k_{ij})) \tag{4.5}$$

Multitask loss The multitask loss of Mask R-CNN is the combined loss of the classification, localization, and segmentation mask on each sampled RoI. The respective loss is summed together as in equation (4.6).

$$L = L_{class} + L_{mrcnnbbox} + L_{mask} \tag{4.6}$$

4.2 Pose Estimation

Pose estimation through computer vision methods is possible because it is feasible to extract features from the data in numerous ways. The pose estimation problem generally gets easier if one has an asymmetric object with distinct features, distinct shapes, and unique texture, which are not the case in most industrial settings.

4.2.1 Related work

Given only an RGB image, some methods [36] aim for single shot instance segmentation which later leads to 6D pose estimation whereas more traditional methods are often based on template matching techniques [25]. The downfall of these methods is that they perform poorly when the object is subjected to occlusions. In this section, related pose estimation techniques are being addressed.

PoseNet [37] was one of the earliest holistic methods in pose estimation, using a CNN for direct pose regression without graph optimization. This is still very difficult due to a large search space and a lack of depth information, using RGB-images. As a way to limit the search area, PoseCNN [38] and similar methods [26] starts with segmenting the scene and later predicting the depth of the object in order to estimate the pose in the remaining three dimensions. 3D pose estimation still inherits non-linearities, which does not make it out-of-the-box compliant to the natural scheme of how classification works in CNN's. Recent end-to-end deep learning methods [39] are still not thoroughly tested with regards to sufficient generalization. Other articles have tackled this by discretizing the rotation space and tackling the 3D rotation as a classification task from there. However, the pose prediction will then only be as good as the resolution of the discretization. Some interpolation in post-refinement is, therefore, necessary to truly achieve the correct pose.

A keypoint-based method would entail detecting some key points of the object and after that utilizing a PnP algorithm or other geometrical constraints for figuring out the 2D-3D correspondences and then predicting the pose. For objects with distinct textures, existing methods perform well under normal conditions. The challenge then lies in detecting texture-less objects, which are primarily defined by their silhouette and depth. Newer methods using only RGB-images [40] have obtained a result of 54% passing rate of all inferences run with respect to the "Pose 6D criterion" on sequences from the T-LESS dataset [41]. Using RGB-D data will give the most significant increase in performance when dealing with texture-less objects because the shape of the object is more descriptive than the texture. Furthermore, methods creating heat maps are highly susceptible to failure when these are occluded.

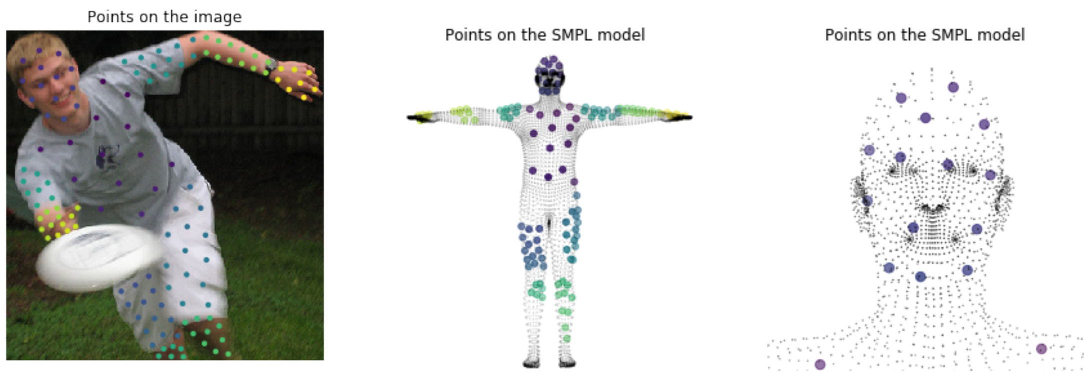


Figure 4.7: Illustration of how the dense representation of the SMPL-model is mapped to the human. [19]

Dense methods are methods that make use of dense, structured information for deterministic purposes. These methods usually require more computing power as the amount of data in the solution space will be higher than most other methods. Hough voting could further be used for pixel-wise or patch-wise prediction of the desired output. Random forest algorithms can be used for pixel-wise 3D object prediction and then use geometrical constraints on top of that to get a more robust pose estimation. Although dense methods might be more difficult to converge, they are also more robust with respect to occlusions.

Point pair features and local histograms are two other ways of determining the pose. Point pair features(PPF's) and extensions of this technique have shown great promise in terms of accuracy according to [42]. PPF's are antisymmetric 4D descriptors of a pair of oriented 3D points m_1 and m_2, constructed as:

$$F(m_1, m_2) = (||d||_2, \angle(n_1, d), \angle(n_2, d), \angle(n_1, n_2)) \tag{4.7}$$

where d is the difference vector, n_1 and n_2 are the normals at m_1 and m_2 respectively and $|| \cdot ||$ represents the Euclidean distance. Local histograms are reliant on color information, but have also yielded good results and has been used for monocular tracking [43].

4.2.2 PVNet

PVNet [20] is an abbreviation for pixel-wise voting network and is has been able to obtain state-of-the-art results, especially on truncated and occluded RGB images.

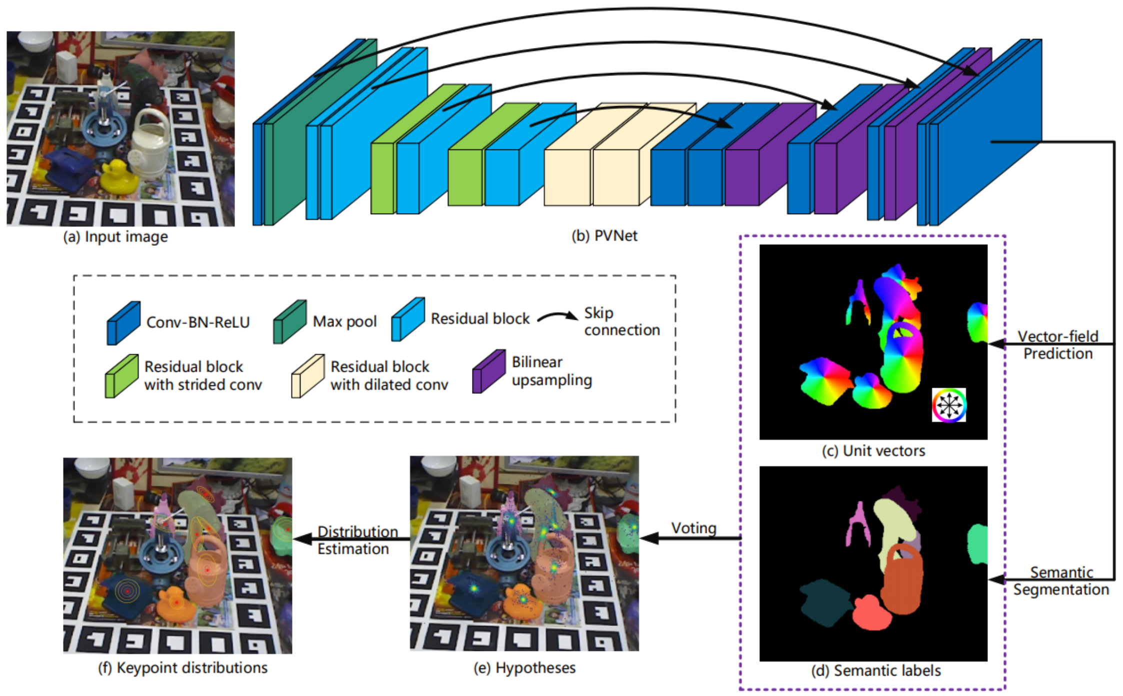


Figure 4.8: Architecture of PVNet. [20]

PVNet is comprised of three main parts. The first one is the semantic labeling, where one uses a ResNet architecture pre-trained at ImageNet as the initialization of weights and as the general backbone in the new network. The network outputs both semantic labeling, as well as a vector field prediction of where the object is. Then a hough voting is performed in order to predict a hypothesis for where the key points are located. After that, an uncertainty-driven PnP algorithm is performed to match the best inliers with the same points in 3D and hence receive the relative pose $[R, t]$ relative to the camera. The output of the voting-based methods is a spatial probability distribution for each of the key points.

In the case of PVNet, the keypoints are the pixels with the highest number of votes after every pixel vote in which general direction each keypoint is to be located based on nine degrees of information it already has knowledge about.

4.3 Synthetic data

There are many advantages to using synthetic data. In an industrial setting, virtual models of the different parts already exist from the design phase of the product. From that, a dataset can be created by applying photo-realistic textures and using advanced rendering engines.

Recent reports have presented promising progress using either texture-less[40] data or using domain randomization where a random texture from a set of different textures are applied. This forces the neural network to learn structural features of the object, rather than colors and surface properties. Doing this makes the network more robust to changes in the environment, such as occlusions and varying light conditions.

4.3.1 Frameworks

The main frameworks used before training the networks are blender and ImgAug. Blender is used for dataset generation, while ImgAug is used by the data loader of the neural network in the training process.

Blender

Blender is a free and open source 3D creation suite licensed under GNU General Public License v2. It is written in C, C++, and Python and supports a wide variety of features, such as modeling, simulation, animation, and rendering. It comes with a flexible Python controlled interface, which opens the possibilities for a fully automated rendering pipeline through Python scripts. The Cycles render engine is Blender's built-in path-tracer engine. Compared to ray-tracing, the path-tracer gives a more visually realistic rendered image. In Blender, the Cycles render also has support for CUDA, which speeds up the rendering process by a significant amount when an Nvidia graphics card is available. For AMD graphics cards the equivalent rendering mode is through OpenCL.

ImgAug

The act of cropping, flipping, scaling, rotating, translating, and applying noise to the original training data set both increases the data set and also helps the neural networks to generalize. A commonly used library in the machine learning community is ImgAug, which is a part of the Python Package Index, and that allows for data augmentation of images. It has support for many kinds of augmentations which can be used in the data loader when training. They can be categorized as either geometric or arithmetic augmentations. The geometric ones are changes where pixels are moved around spatially, thus changes the structure of the body in the image. Below is a list of geometric augmentations used in this thesis and their respective effects on an example image are further illustrated in Figure 4.9.

- **Crop:** Crops the image around a specified or random location.
- **Flip:** Flips the image over the x- or y axis respectively.
- **Scale:** Changes the scale of the image using bi-linear interpolation.
- **Translate:** Translation of the image over the x- and/or y axis.
- **Rotate:** Clockwise rotation around the center point of the image
- **Shear:** Linear mapping of the pixels in fixed direction by an amount proportional to its signed distance from the line parallel to that direction and goes through the origin.

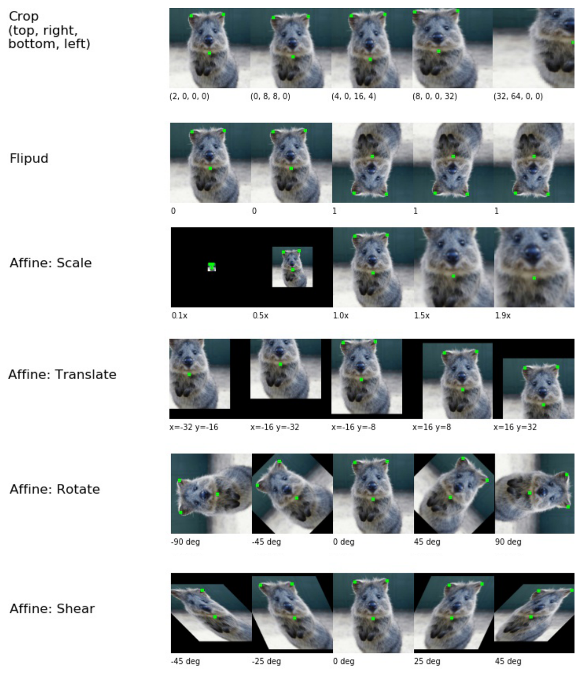


Figure 4.9: Each of the geometric augmentations with different input parameters used in the training set illustrated on an image. [21]

The arithmetic augmentations are pixel-wise modifications done to the image where the body structure of the object remains the same, but where each channels respective pixel's value necessarily does not. Below is a list of such augmentations used to augment the training data set, and in Figure 4.10 their respective effect on an example image.

- **Gaussian blur:** Filters the entire image with a Gaussian filter.

- **Coarse dropout:** Sets random square areas of the image to zero, using a miniature pixel map that is then up sampled.

- **Edge detect:** Highlights the edges and makes the rest of the image darker

- **Emboss:** Each pixel is replaced by either a highlight or a shadow, depending of its gradient with respect to the neighboring pixels. The filtered image will represent the rate of color change at each location.

- **Sharpen:** A combination of blurring, edge enhancement and contrast increment to increase how sharp the image appears to the human eye.

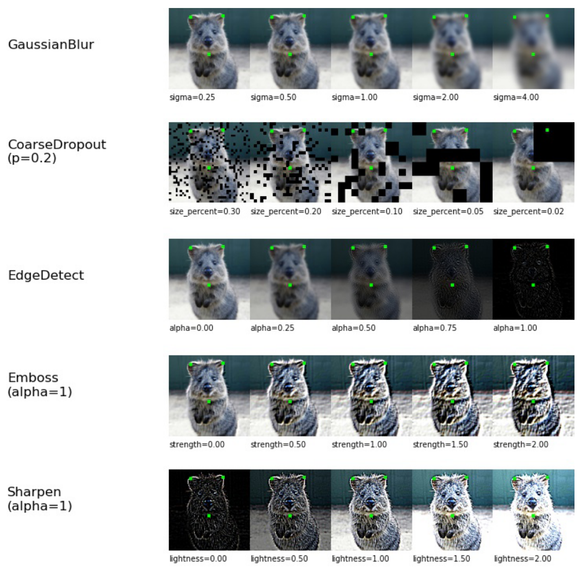


Figure 4.10: Each of the arithmetic augmentations with different input parameters used in the training set illustrated on an image. [21]

In order to keep a correct mapping of the segmentation map, they need to be augmented with the same parameters as its corresponding image for the geometric augmentations. The segmentation maps should not, however, be affected by the arithmetic augmentations. The augmentations needed for a project depends on the application at hand, and there are more different kinds of augmentations available in the library than listed here.

4.3.2 Equidistantly spaced points on a sphere

When sampling poses of an object, precautions to prevent the samples from clustering around a particular view should be made. A common problem in both computer graphics and other STEM fields is to generate equidistantly spaced points on a sphere. There is no mathematical formula that does this correctly except in some special cases. When solving this problem, it is a matter of how approximate the solution is allowed to be.

The naive approach is to generate the points on the sphere using equal longitudinal and latitudinal angles, but as can be seen in the upper parts of Figure 4.11, the points will get clustered around one of the axles.

A more suitable approach, in the sense of correctness, is viewing the points as magnets and let them repel each other until an equilibrium has been reached. This is however computationally heavy, so another way is to use a Fibonacci lattice, also known as the Golden Spiral method. This way of generating equidistantly spaced points on a sphere has a computational complexity of $\mathcal{O}(N)$, but it is also less exact than the recursive solution. The MSE of the distances are dependent on how many points that are generated, but to the human eye, the points will look equidistantly spaced, and for this project, it is accurate enough. The Golden Spiral method is implemented as follows:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} r \cdot \cos\theta_i \\ \frac{2i+N-1}{N} \\ r \cdot \sin\theta_i \end{bmatrix}, \text{ where } \begin{cases} \theta_i = ((i+1) \mod N) \cdot (3-\sqrt{5})\pi \\ r = \sqrt{1-y_i^2} \end{cases} \tag{4.8}$$

x_i, y_i and z_i are a sample point in Cartesian coordinates and N is the number of sample points to be generated.

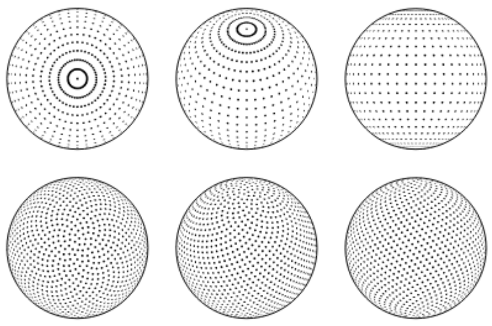


Figure 4.11: In the top sphere, points are separated by equal longitudinal and latitudinal angles. In the bottom sphere, the points are separated using the Fibonacci lattice. [22]

4.3.3 Point clouds and mesh sampling

PVNet uses a point cloud of the different objects in order to extract suitable key points to be trained in the network. Experiments have shown that using an optimization technique to extract the key points from the actual surface rather than using the bounding box yields better results. Thus, for this purpose, the point cloud needs to be generated out of the different model files. There are several ways to do this, and the first attempts were to use a free software called MeshLab and export each respective model using their built-in export function. The point cloud generated is, however, not uniformly distributed, as can be seen in Figure 4.12 but instead sampled at the vertex of each polygon in Figure 4.13.

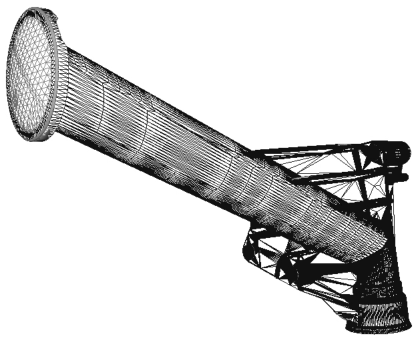


Figure 4.12: The mesh grid of the object, where the surface consists of polygons

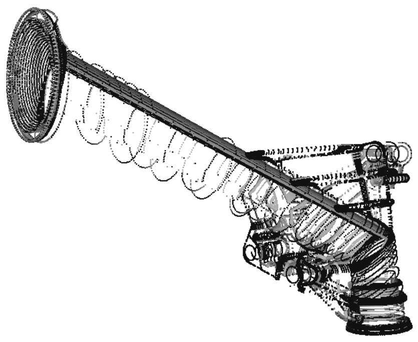


Figure 4.13: A point cloud representation of the intake. The points are sampled in the vertexes of the polygon representation.

The key point extraction algorithm should extract points as far as possible from each other, and to do so, the points in the point cloud must represent the entire surface of the object. One strategy is to sample from each of the triangles randomly. Because of the different areas of the triangles, this sampling technique will result in a non-uniformly distributed point cloud, as illustrated in Figure 4.14. The upper circles show a uniformly sampled mesh, whereas the lower circles illustrate how the distribution will be if sampled according to the technique described above.

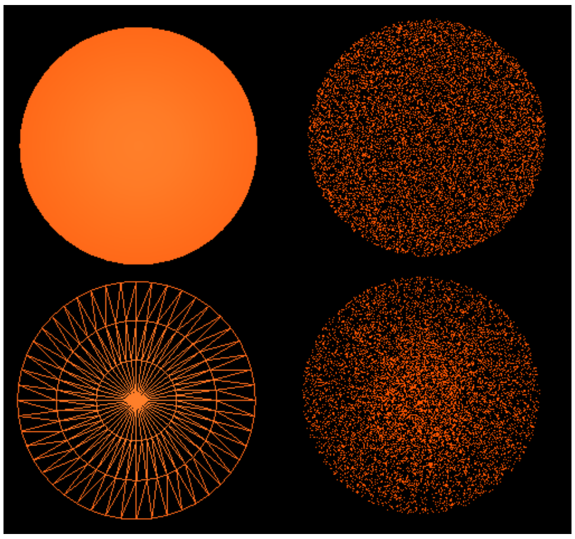


Figure 4.14: Uniformly and non uniformly sampled mesh of a dome. [23]

To remedy this phenomenon, the probability of taking a random sample inside a triangle should be proportional to how large area that triangle has in relation to the surface area of the mesh. Smaller triangles will then be less likely to be sampled from, and will therefore not contribute to unevenly distributed clusters. In Figure 4.15, the intake is sampled through this technique. As the figure illustrates, the samples are evenly distributed over the entire mesh of the intake.

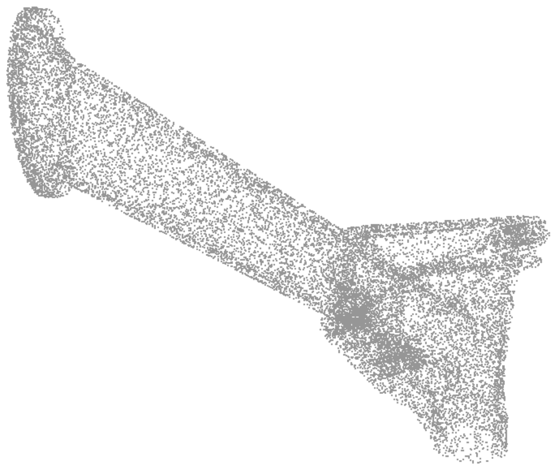


Figure 4.15: Properly sampled mesh of the intake with 20 000 samples.

5
Methods

In this chapter, the methods, and parameters used as well as some motivation of them is stated.

5.1 Implementation

The implementation revolves around creating an end-to-end pipeline of generating synthetic data, training a network on this data, and lastly running inference on real data. It is assumed that the solution manifold gradient looks quite similar at the early stages of training, but due to lens distortion, natural light, shadows, it is likely that the global minimum differs in the later phases of training. This is taken into consideration while developing the algorithms and determining parameters.

5.1.1 Evaluation methods

In this thesis, there are two methods used for evaluation of the performance. The first is the intersection over union (IoU) for the instance segmentation, and the second one is the Average Distance (ADD) for pose estimation.

Intersection over Union

Intersection over Union, also known as the Jaccard index is a metric used to determine the similarities between two sample sets. Mathematically it is defined as in equation (5.1) and a visual representation is shown in Figure 5.1. The figure illustrates the respective areas as highlighted, with the overlapping regions in the nominator and the combined regions in the denominator. In machine learning, it is an evaluation metric commonly used to measure the accuracy of an object detector. If there are no overlapping areas, this metric yields a score of zero whereas two fully overlapping areas yields a score of one.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{5.1}$$

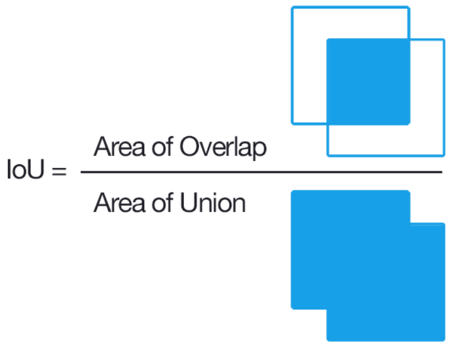


Figure 5.1: The IoU metrics for two overlapping sets. [24]

ADD metric

The IoU metric only captures 3 degrees of freedom, so for 6D pose estimation, another evaluation method is necessary. Average Distance of Model Points is a method proposed by Hinterstoisser et al. [44] and are defined in equation (5.2). The error of the estimated pose $\hat{P}$ with respect to the ground truth pose $\bar{P}$ of an object model M is calculated as the average distance to the corresponding model point. If the error is under 10% of the max diameter of the object the method is considered as a pass.

$$e_{ADD}(\hat{P}, \bar{P}; M) = \underset{x \in M}{\text{avg}} ||\hat{P}x - \bar{P}x||_2 \tag{5.2}$$

The evaluation images seen in Figure 5.2 and 5.3, are randomly selected from an evaluation video sequence. As seen from the histogram values in Figure 5.4, evaluation image 1 has a more balanced overall color space, but is also affected by an automatic light balancing feature in the Logitech camera, resulting in some noise on high intensities. Figure 5.3 has less noise but decent gradients, as seen in Figure 5.5.

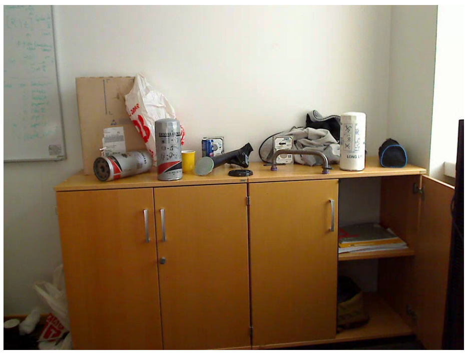


Figure 5.2: Evaluation image 1, sunlight from window, no roof lighting

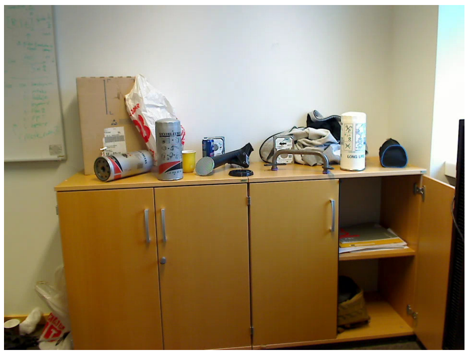


Figure 5.3: Evaluation image 2, sunlight from window and roof lighting

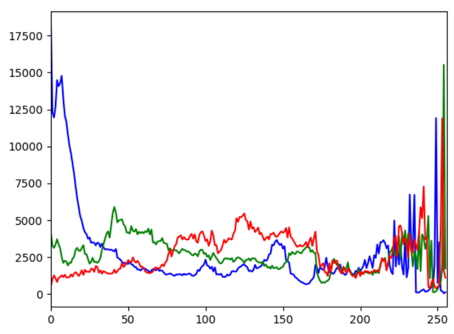


Figure 5.4: Histogram values of image 1

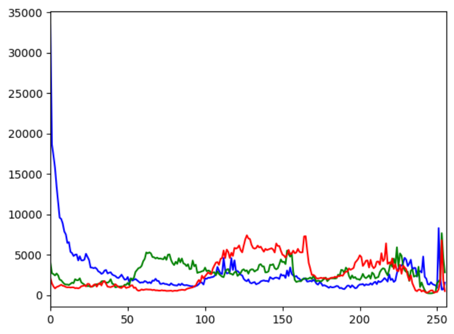


Figure 5.5: Histogram values of image 2

5.1.2 Mask R-CNN parameters

Data set generation

Data set for the training and validation sets of Mask R-CNN consisted of objects rendered with an equal distance from the camera and lights. The light settings were set to a fixed intensity, and the objects were rotated with fixed intervals around two of their local Z and Y axis. Two thousand images for each class were rendered. Each individual image was rotated 15 times yielding a total of 30 000 different poses for each class. The generated images were then composed into a data set with a different number of classes in each image. A total of 60 000 images were produced for the training, and 6 000 of them were used for the validation set.

Training parameters

The training parameters for Mask R-CNN were extracted from the previous work upon which this thesis is built and initially set as in Table 5.1 below. The network was trained for 120 epochs, with 80 of them only training the heads and 40 of them the last four layers. The above parameters were tweaked during the training phase to the ones presented in Table 5.2 to yield a smoother convergence on the training and validation loss. In the augmented settings, the network was trained over 150 epochs over the last four layers.

Table 5.1: Untuned training parameters

Parameter	**Value**
Steps per epoch	1000
Initial learning rate	0.001
Learning momentum	0.9
Weight decay	0.0001

Table 5.2: Tuned training parameters

Parameter	**Value**
Steps per epoch	1000
Initial learning rate	0.0001
Learning momentum	0.8
Weight decay	0.00007

Four models with different amounts of augmentations implemented in the data loader where made. The first model was made without any augmentation whereas the last three have their respective parameters are listed in Table 5.3, 5.4, 5.5 and 5.5. Each augmentation has a specific probability of occurring, and the augmentations are either applied to each channel with the same parameters or with different ones for each channel. The channel-wise probability is only applied to the fraction of the augmentations specified. The ranges in the two rightmost columns are the bounds within the augmentations applied. For instance, if the augmentation **Add** are applied and the bounds are -20 to $+20$, a pixel value between these bounds are added to all pixel values in the image.

Table 5.3: Parameters for the first augmentation settings. Augmentations on rows with the same colors are mutually exclusive.

Augmentation	**Probability of occurrence**	**Probability channelwise**	**Range from**	**Range to**
Shear	0.2	-	-16°	+16°
Add	1.0	-	-50	+50
Add	0.2	0.5	-20	+20
Multiply	0.2	0.2	0.8	1.2

Table 5.4: Parameters for the second augmentation settings. Augmentations on rows with the same colors are mutually exclusive.

Augmentation	**Probability of occurrence**	**Probability channelwise**	**Range from**	**Range to**
Horizontal flip	0.3	-	-	-
Shear	0.6	-	-16°	+16°
Add	0.8	-	-50	+50
Add	0.07	0.4	-20	+20
Multiply	0.1	0.2	0.8	1.2
Gaussian noise	0.1	0.2	0	5.1

Table 5.5: Parameters for the third augmentation settings. Augmentations on rows with the same colors are mutually exclusive.

Augmentation	**Probability of occurrence**	**Probability channelwise**	**Range from**	**Range to**
Horizontal flip	0.3	-	-	-
Scale x	0.6	-	-20%	+20%
Scale y	0.6	-	-20%	+20%
Shear	0.6	-	-16°	+16°
Add	0.35	-	-70	+70
Add	0.15	0.4	-30	+30
Contrast normalization	0.2	-	0.5	1.5
Gaussian noise	0.1	0.2	0	5.1

Table 5.6: Parameters for the fourth augmentation settings. Augmentations on rows with the same colors are mutually exclusive.

Augmentation	**Probability of occurrence**	**Probability channelwise**	**Range from**	**Range to**
Horizontal flip	0.3	-	-	-
Vertical flip	0.2	-	-	-
Crop & fad	0.5	-	-0.05%	0.1%
Shear	0.6	-	-16°	+16°
Translate	0.6	-	-0.2%	+0.2%
Rotate	0.6	-	-45°	45°
Sharpen	0.5	-	0.75	1.5
Emboss	0.5	-	0	2.0
Contrast normalization	0.5	0.5	0.5	2.0
Coarse dropout	0.5	-	0	0.1
Edge detect	0.5	-	0	0.1
Directed edge detect	0.5	-	0	0.1

5.1.3 PVNet

Rendering

The rendering setup in PVNet uses textured PLY files as objects and samples the poses of each object to be rendered in Linemod from a set of annotated real images. This is done partly in order to render images in poses that are more likely to occur, but also to make sure that the objects are always rendered within the field of view. Then the camera is placed on a sphere in a pseudo equidistantly manner, as explained previously in 4.3.2. Since a similar annotated dataset does not exist for the set of objects in this thesis, the pose was set to a uniformly random rotation in all axis with the boundaries $[0, 360)$ for all objects. The source of light in the setup is two point-light sources placed in the scenery at a uniformly random distance from the object between $[1, 2]$ with a light intensity of between $[0, 5, 2]$. This is the standard light setup that comes with PVNet and was not changed.

The different classes of objects differ significantly in size from each other and are therefore rendered with different limits. The idea is that the network should train on data that contains the entirety of the objects in the beginning and then trained on a data set containing occlusions. However, the camera is supposed to be static, and the objects are mounted relatively close to each other. Because of this, in the context of quality inspection, it is important that all of the object classes have an overlapping distance interval in the training set. Linear regression with different scaling factors was made for each of the classes where the regressand is the distance to the object, and the regressor is the maximum translation distance in x and y from the center for that distance. The maximum allowed distance from the camera to the object was set so that the object would be contained within a bounding box with an area of 2500 pixels. The individual settings for each of the respective classes are listed in Table 5.7 below.

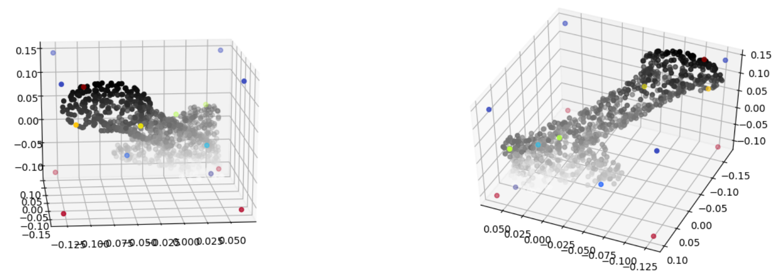
Table 5.7: Translational boundries per class

Class	**Min. distance**	**Max. distance**	**Coefficient x**	**Coefficient y**
Intake	1.0	2.0	0.31	0.24
Ladderframe	1.5	3.0	0.15	0.12
Bypass-v	1.0	2.0	0.27	0.20
Pipe2	1.0	2.0	0.26	0.20

The boundaries for each object class, the maximum and minimum distance are the distance between the camera and the object in meters. The allowed translational movement of the objects is dependent on the actual distance with the coefficient factors in x and y.

With the settings described above, 4000 images of each class were rendered over a planar surface with the Pascal 2012 dataset as texture to the background plane. The orientation of the objects was saved as a serialized python object in a separate file. The segmentation mask of the object was saved as an image containing binary integers. From the different objects, a 9D dense point representation of the object was generated through mesh sampling with 70 000 points. This representation contains the spacial information, the surface normals, and the RGB data of all the sampled points. By using the point cloud model of the object, the eight points with the maximum summed distance from each other were extracted. Each pixel in a prediction mask votes towards one of these key points, and they are needed in order to regress the pose and bounding box of an object.

Because performing matrix multiplication and repeated tasks are much faster when using CUDA, it is beneficial to do this as much as possible. This is why it is needed to convert everything that will go into the networks to tensors. For further manipulating the data one does therefore need to move the respective variables back from CUDA memory to CPU memory.



(a) Pose 1 (b) Pose 2

Figure 5.6: Dense points(gray), keypoints(various colors) and 3D bounding box points(blue-red)

The segmentation loss is calculated as a cross entropy loss between the prediction and the ground truth mask, averaged over the mini-batch. The vertex loss is calculated with a smooth L1-loss, based on the discrepancy between the vertex-predictions, ground truth projected vertices and vertex weights. Precision and recall is calculated as in equation (5.3) and (5.4) respectively.

$$Precision = \frac{true\,positives}{true\,positives + false\,positives} \tag{5.3}$$

$$Recall = \frac{true\,positives}{true\,positives + false\,negatives} \tag{5.4}$$

Backbone network, batch size, training rate

ResNet18 was chosen as the backbone in PVNet because of its high performance/size ratio in accordance with Figure 2.9. The smaller network facilitates the task of obtaining a faster training process when using transfer learning and retraining the deeper layers. The weights have been pre-trained on ImageNet and are used as a starting point when retraining the whole network. Batch normalization was also used as it has been shown to improve both training convergence and boost performance significantly [29]. There are some conflicting statements concerning the size of the mini-batches. One the one hand, progressively larger mini-batches can utilize higher efficiency processors as well as distributed computing over multiple nodes. Based on these reasons and the results from articles like [45] this seems reasonable. On the other hand, small mini-batch sizes take less memory, which increases the possibility of storing these variables closer to the CPU. Therefore a smaller batch size can utilize the higher cache speeds in addition to that the gradient updates are more recent and will have a higher probability of faster convergence.

According to [29], a learning rate of 2^{-10} and a batch size of 8 with batch normalization obtained the best results. It also always performed a few percents better when using data augmentation. For the training of PVNet, a normalized batch approach with mini-batches of 8 was implemented. The base learning rate was set to 0.001, and the decay rate was 0.5 every 20 epochs, and Adam was used as the optimizer. The batch size and learning rate were increased when it seemed like the weights were stagnating in order to investigate if the optimization was stuck in a local minimum.

5.2 Camera Calibration

Multiple scripts were written to record sharp images of the calibration checkerboard. The initialization was done by sampling many frames and storing the Laplacian values of the first 20 images and then choose the 5th best value as a threshold for the algorithm. Moving the camera induces motion blur, which decreases the laplacian value of the picture and effectively culling it from the sample set. When calibrating the camera, 100 images of a checkerboard in different position across the receptive field was taken. The provided e-Con cameras had substantial higher order radial distortion when diverging from the center of the camera sensor. Due to various software and hardware issues, it was decided to use a Logitech QuickCam 9000 instead throughout the entire project.

6
Evaluation of suitability for industry applications

In this chapter, the results of semantic labeling and pose estimation on synthetic and real data are presented and observations of importance are formulated.

6.1 Segmentation results

In order to evaluate the suitability for industry applications, Mask R-CNN and PVNet are evaluated with respect to segmentation results. These results are shown below.

6.1.1 MaskRCNN

Synthetic data

Training the last layer of MRCNN yielded the loss illustrated in Figure 6.1 and 6.2. Even though the training loss is high, it is decreasing. What is especially noticeable is the variance of the validation loss. It seemingly acts like a random variable rather than a smooth graph with a negative slope, as usually is intended. A possible reason for this is that the initial learning rate is too high coupled with a decay rate that is too low, so the stochastic gradient descent overshoots the minimum by a large margin. Tweaking these parameters, however, did not improve the resulting validation losses in any significant way. By studying the individual contributions to the losses, the main contributors to the stochastic behavior came from the RPN classification loss and the RPN bounding box proposal loss. As can be seen in Figure 6.3, the RPN network falsely proposes and classifies the background as foreground and vice versa in about 20% of the cases. This implies that there are objects which are very hard to classify. Furthermore, the behavior of the bounding box proposal loss, seen in Figure 6.4, suggests that some of the objects in the set are hard to fit a bounding box around. Given that the bolt and the pipe fixture in the classes are extremely small from the set distance, they were excluded from the data set.

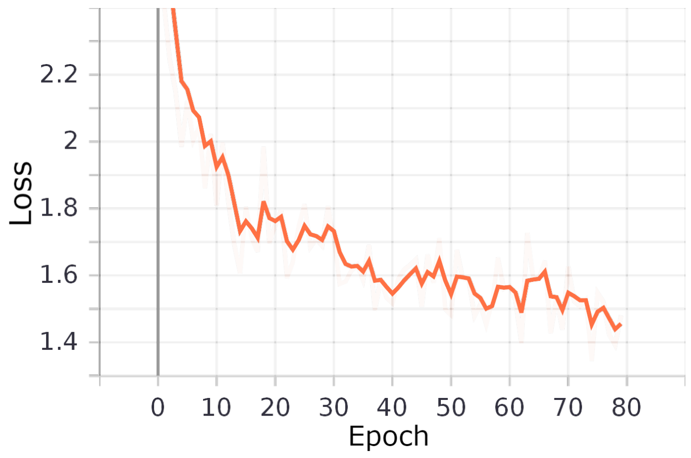


Figure 6.1: Total error training with 8 different classes over 80 epochs.

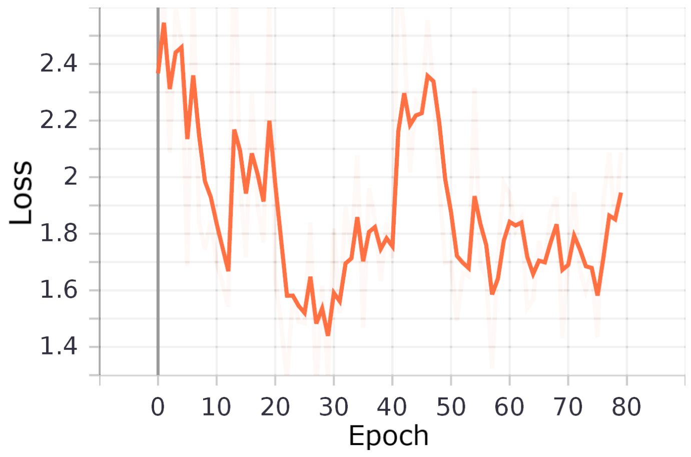


Figure 6.2: Total validation loss behaves as a seemingly stochastic random variable.

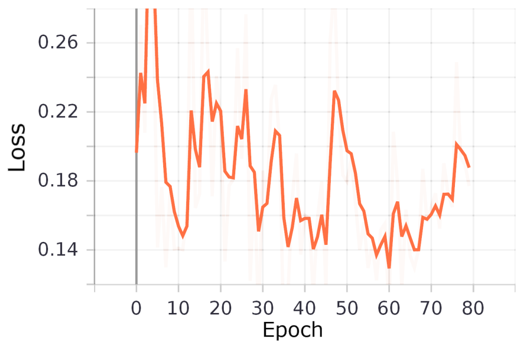


Figure 6.3: Validation loss for the RPN classifier and the main contributor to the behaviour of the validation loss

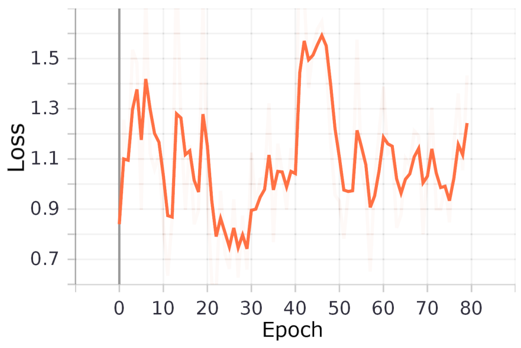


Figure 6.4: Validation loss for the RPN bounding box proposals.

After this change to the data set, the resulting validation loss can be seen in Figure 6.5. The trend is moving downwards and has a steep decline after about 80 epochs, where the network is trained more deeply for 40 epochs. The sharp trend downwards can be due to two reasons. Firstly, since the network, when only training the heads has not yet reached the global minimum, resetting the learning rate facilitates the process of moving towards it. Secondly, the last four layers might have a significant impact on the task of object recognition. Given that the decline flattens out after just five epochs, the first reason is the more probable one, but it is not possible to say that the last four layers have no impact at all.

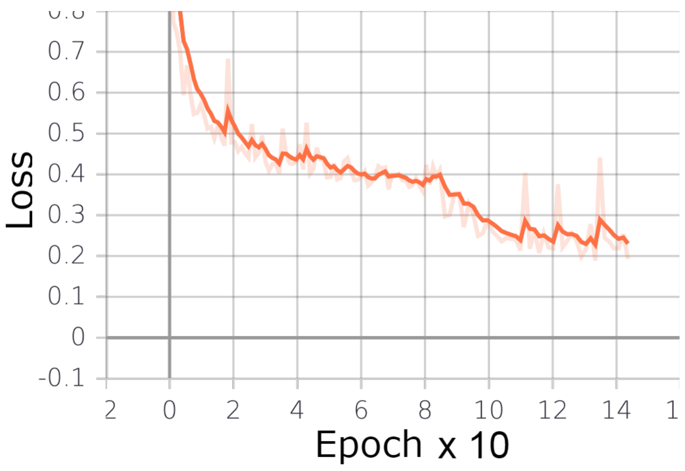


Figure 6.5: Bounding box loss training only the heads over 80 epochs and then training the last 4 layers for 40 epochs with new training parameters.

Four models were trained over 150 epochs on the four last layers with augmentation activated. The idea was to make the model more robust against occlusions, variations in light settings, and color distortions in the camera sensor. In Figure 6.6, the validation loss for this model throughout the training is illustrated. Even though the data is intentionally altered, it has a loss comparable to the unaugmented model. Since the validation data set is not augmented, this implies that the model is general enough to recognize the objects even if there are more significant changes to the environment.

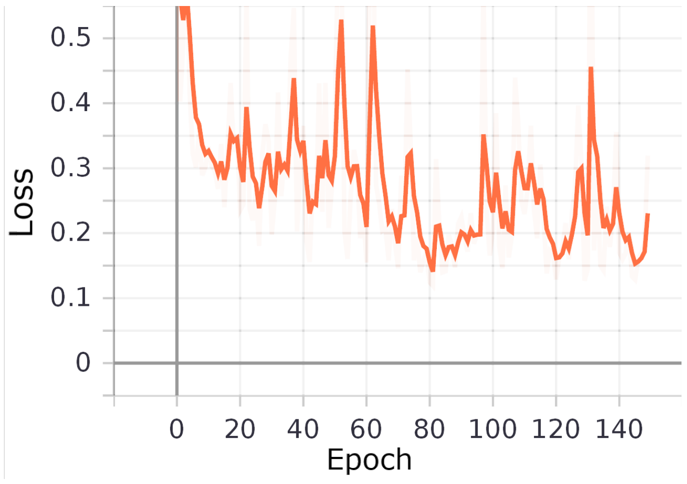


Figure 6.6: Validation loss with augmented training data.

Real data

Even though the loss using synthetic data acts seemingly fine, there is a substantial risk of overfitting. In Figure 6.7 and Figure 6.8, the behaviour of the loss over a synthetic and a real hand-annotated data set can be observed. After about 50 epochs, the network starts to overfit to the synthetic data, and as is seen in Figure 6.8 the loss for the real set starts to increase. This means that the pre-trained generalization of the weights used in the transfer learning starts to lose its meaning. Because of this, it was decided to evaluate the IoU for the different networks after 45 epochs and 70 epochs.

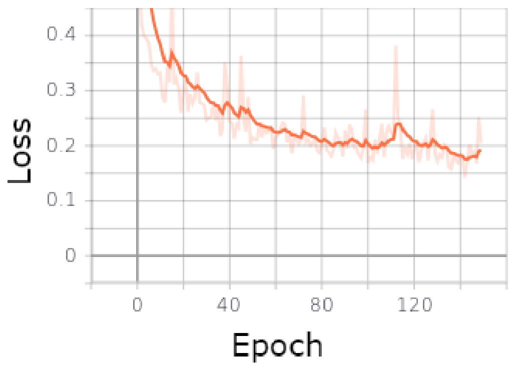


Figure 6.7: Total training loss

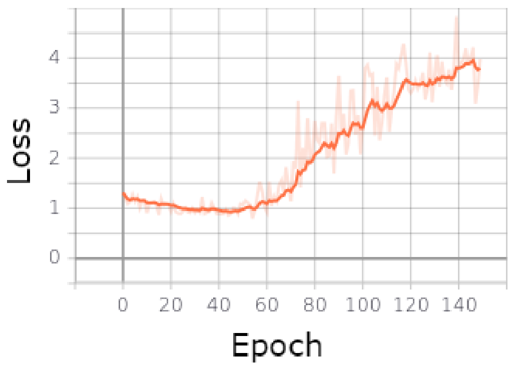


Figure 6.8: Total validation loss from validation on real images

In Table 6.1 and 6.2 are the IoU metrics for a set of two hand annotated images in different light settings. In Table 6.1 the light settings are uniform with little directed light as shown in Figure 5.3. The columns are the IoU for the different objects in the scene, and the rightmost column shows the number of false positives. The best overall performance was obtained by the model without augmentations, trained to 120 epochs. The models with augmentations have in general problems with false positives and false negatives. In Table 6.2, the light settings are more directed as shown in Figure 5.2. In this case, the augmentations help with the IoU, but there are still issues with false positives and false negatives for the models with augmentation. That means the model sometimes either identifies the background as an object for false positives, or vice versa for the false negatives.

Table 6.1: Validation on a real image over some of the objects in a cluttered scene and very little directed illumination. Metrics are in IoU and number of false positives in the rightmost column. The rows in white are models trained over 45 epochs, the rows in light-cyan are models trained over 70 epochs, and the rows in dark-cyan are models trained over 120 epochs

Model	bypass-r	bypass-r	intake	pipe1	bypass-v	FP
Previous	-	-	0.709	0.142	0.837	2
No augmentation	0.869	0.867	0.836	0.410	0.925	2
No augmentation	**0.880**	0.921	0.837	0.658	**0.926**	1
Augmentation 1	0.847	0	0.821	**0.686**	0.897	1
Augmentation 1	0	**0.938**	0.840	0.587	0.879	**0**
Augmentation 2	0.801	0.897	0.823	0.548	0.897	1
Augmentation 2	0	0.899	**0.868**	0	0.897	0
Augmentation 3	0.783	0	0.860	0.543	0.904	1
Augmentation 3	0	0.886	0.836	0.526	0	1
Augmentation 4	0.811	0	0.822	0.665	0.906	1
Augmentation 4	0.631	0	0.830	0.675	0.895	1

Table 6.2: Validation on a real image over some of the objects in a cluttered scene and more directed illumination. Metrics are in IoU and number of false positives in the rightmost column. The rows in white are models trained over 45 epochs, the rows in light-cyan are models trained over 70 epochs, and the rows in dark-cyan are models trained over 120 epochs

Model	bypass-r	bypass-r	intake	pipe1	bypass-v	FP
Previous	-	-	0.793	0.175	0.862	4
No augmentation	0.914	0.848	0.897	0	0.913	1
No augmentation	**0.920**	0.877	0.894	0.509	0.940	**0**
Augmentation 1	0.907	0.906	0.892	**0.608**	0.943	2
Augmentation 1	0.879	**0.912**	**0.922**	0.498	0.944	1
Augmentation 2	0.853	0	0.890	0	0.920	2
Augmentation 2	0.898	0.857	0.904	0	0.904	1
Augmentation 3	0.898	0	0.901	0.494	0.931	2
Augmentation 3	0.907	0.865	0.881	0.471	0.915	1
Augmentation 4	0.891	0	0.840	0.548	0.905	2
Augmentation 4	0.878	0	0.886	0.508	**0.948**	2

An example of the inference using the settings with the best results is shown in Figure 6.9. As can be seen, even though the metrics in Table 6.2 yields a poor metric for pipe1, the bounding box for all of the objects are seemingly correctly sized and correctly aligned.

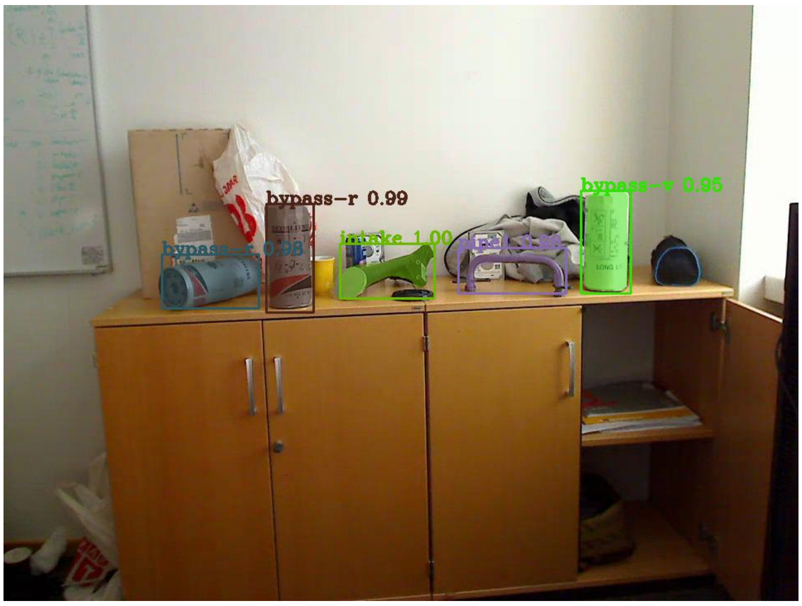


Figure 6.9: Inference on a real cluttered scene with the model trained without augmentations.

Observations

Doing the inference on the full evaluation video sequence some general observations are made. The model in all cases is good at predicting the different classes but puts too much emphasis on the color of the different items. Because of this, it struggles to differentiate the Renault and the Volvo filters in different light settings or when a shadow is cast upon the Volvo filters. This yields false predictions with high confidence, which is usually undesirable. Furthermore, the trained models are highly sensitive to occlusions, even though the model with heavy augmentations has pixel-wise dropout.

6.1.2 PVNet

Synthetic data

ResNet18 achieves average masking errors on the synthetic validation set of ca 5%, and recall and precision are usually around 90%, with a model trained to epoch 45. In extremely cluttered scenes it typically also produces some false positives, which in itself might be okay, but in some cases it is enough to disturb the voting layer, making the resulting pose quite distorted. Examples of mask predictions on synthetic data is illustrated in Figures 6.16 and 6.17.

Real data

Below are examples of mask predictions.

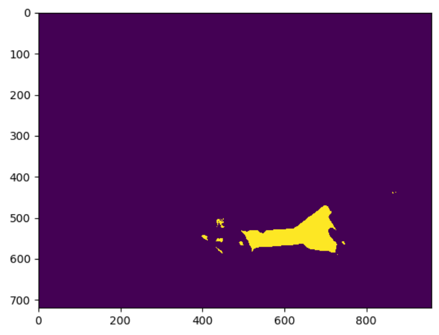


Figure 6.10: Mask prediction - muted light.

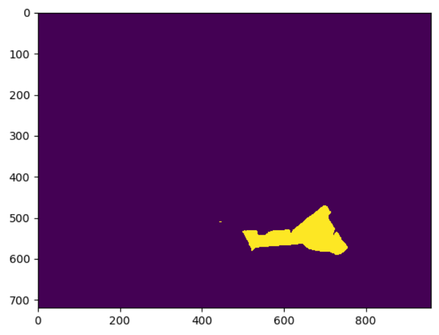


Figure 6.11: Mask prediction - direct sunlight.

Table 6.3: Real inference - muted light / direct sunlight. Resnet model at epoch 95. Hand-annotated ground truth

Metric	Muted light	Direct Sunlight from the side
LOSS SEG	0.0128	0.0116
PRECISION(IoU)	0.9423	0.9849
RECALL	0.7960	0.8454

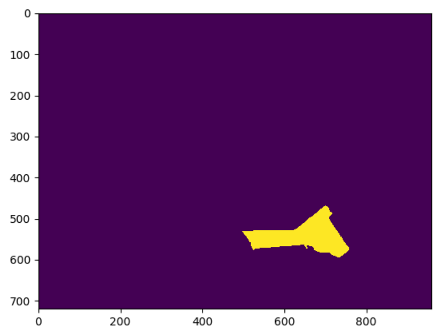


Figure 6.12: Mask prediction - muted light.

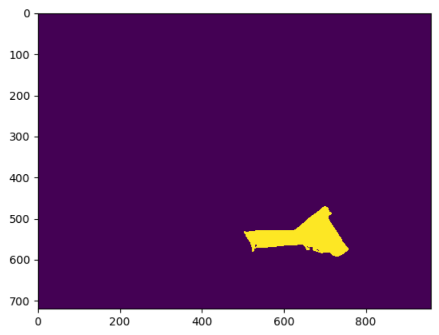


Figure 6.13: Mask prediction - direct sunlight.

Table 6.4: Real inference - muted light / direct sunlight. Resnet model at epoch 139. Hand-annotated ground truth

Metric	Muted light	Direct Sunlight from the side
LOSS SEG	0.0083	0.0153
PRECISION(IoU)	0.9701	0.978
RECALL	0.8773	0.8607

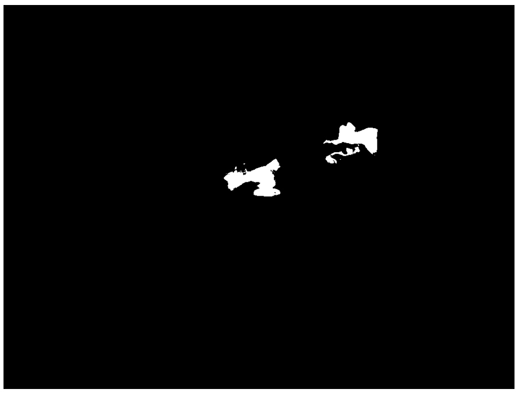


Figure 6.14: Image 1 for segmentation comparison, IoU = 39.43% (epoch 95)

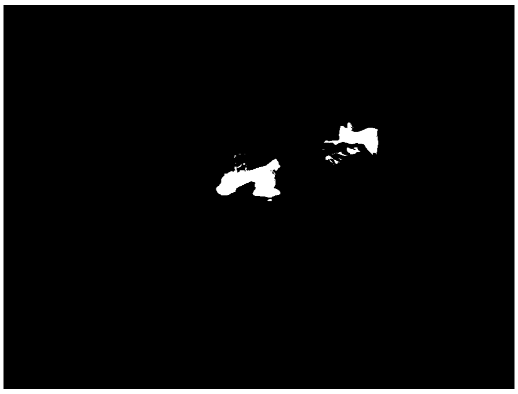


Figure 6.15: Image 2 for segmentation comparison, IoU = 45.41% (epoch 95)

Observations

Figure 6.10, shows that the network predicts some false positives due to the darkness of the scene, while in a scene with more direct sunlight, the object is more clearly identified. In Figure 6.14 the IoU turns out to be lower than in the brighter picture as seen in Figure 6.15, although both images are predicting false positives on the dark textile in the background.

The segmentation results on the cross-method validation images are slightly better when the objects are more illuminated, although both scenes in the test image detect a lot of false positives regarding the cloth.

6.2 Pose estimation results

In order to evaluate the applicability of industry applications, PVNet is evaluated with respect to pose estimation results. These results are shown below.

Synthetic data

The first column in Figure 6.16 illustrates the image used as input as well as the predicted pose in terms of the projected bounding box as well as the ground truth bounding box. The second column illustrates the mask-prediction on synthetic data after 86 epochs of training. Column three illustrates the ground truth mask of the object. The corresponding Table 6.5 shows that the calculated ADD distance at 1.5-2.5 meters is from 5.25 cm to 14.21 cm, and the rotation is from 5.0 degrees to 10.1. The results will improve when training longer, predicting to reach at least a 60% ADD passing score at epoch 200 [20]. For figure 6.17 the layout is the same as Figure 6.16. As seen in Table 6.6, the ADD distance ranges from 7.1 cm to 19.1 cm and the rotation is from 3.1 to 15.4. An interesting detail to notice when investigating the masking predictions is in the first row and second column in Figure 6.17. There exists a prediction island patch, north of the actual object. This patch is also allowed to do pixel voting with seemingly high confidence. This creates a skewed total hypothesis, and the pose estimation is slightly worse because of it.

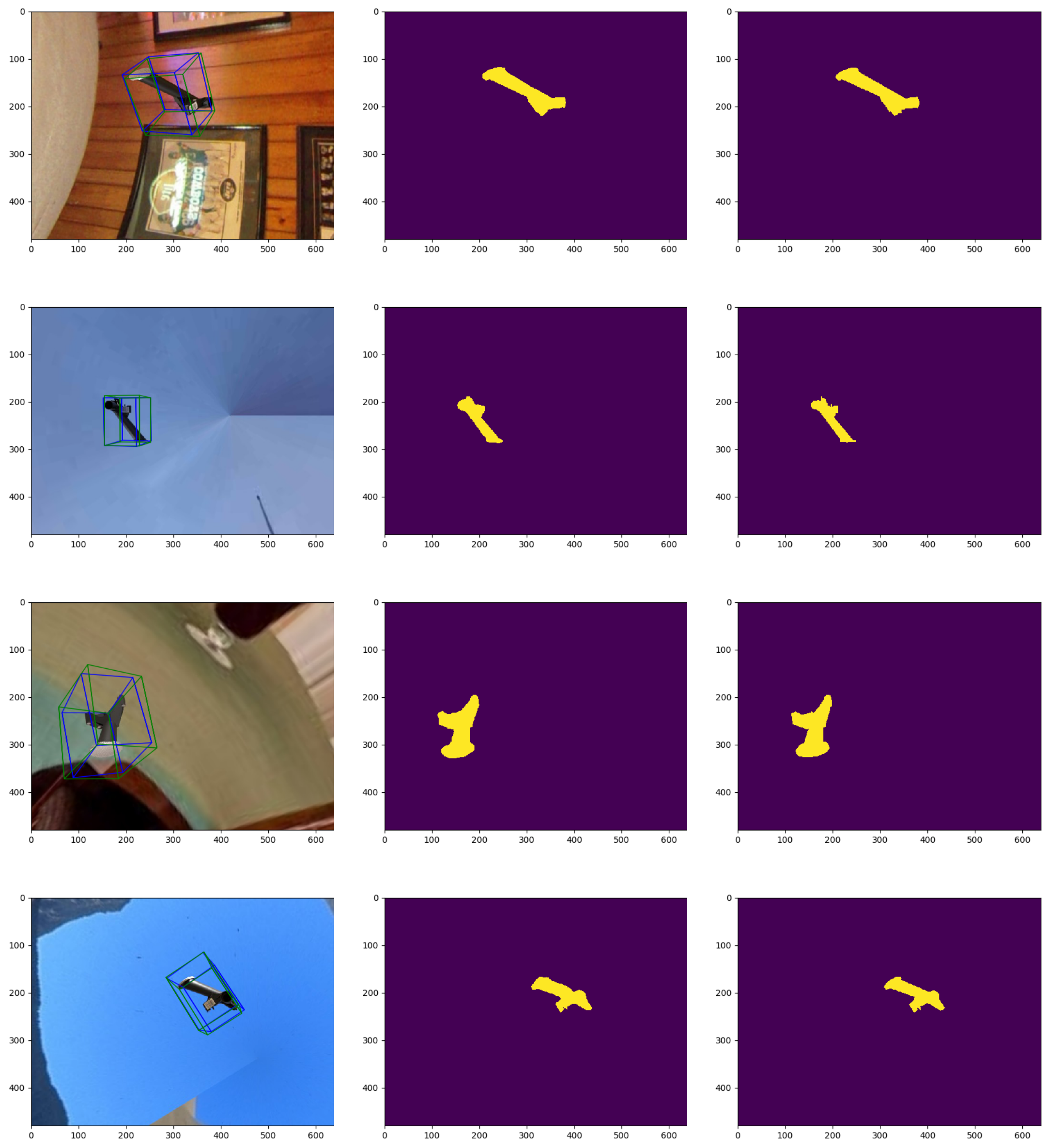


Figure 6.16: Pose estimation results on synthetic validation set, part 1.

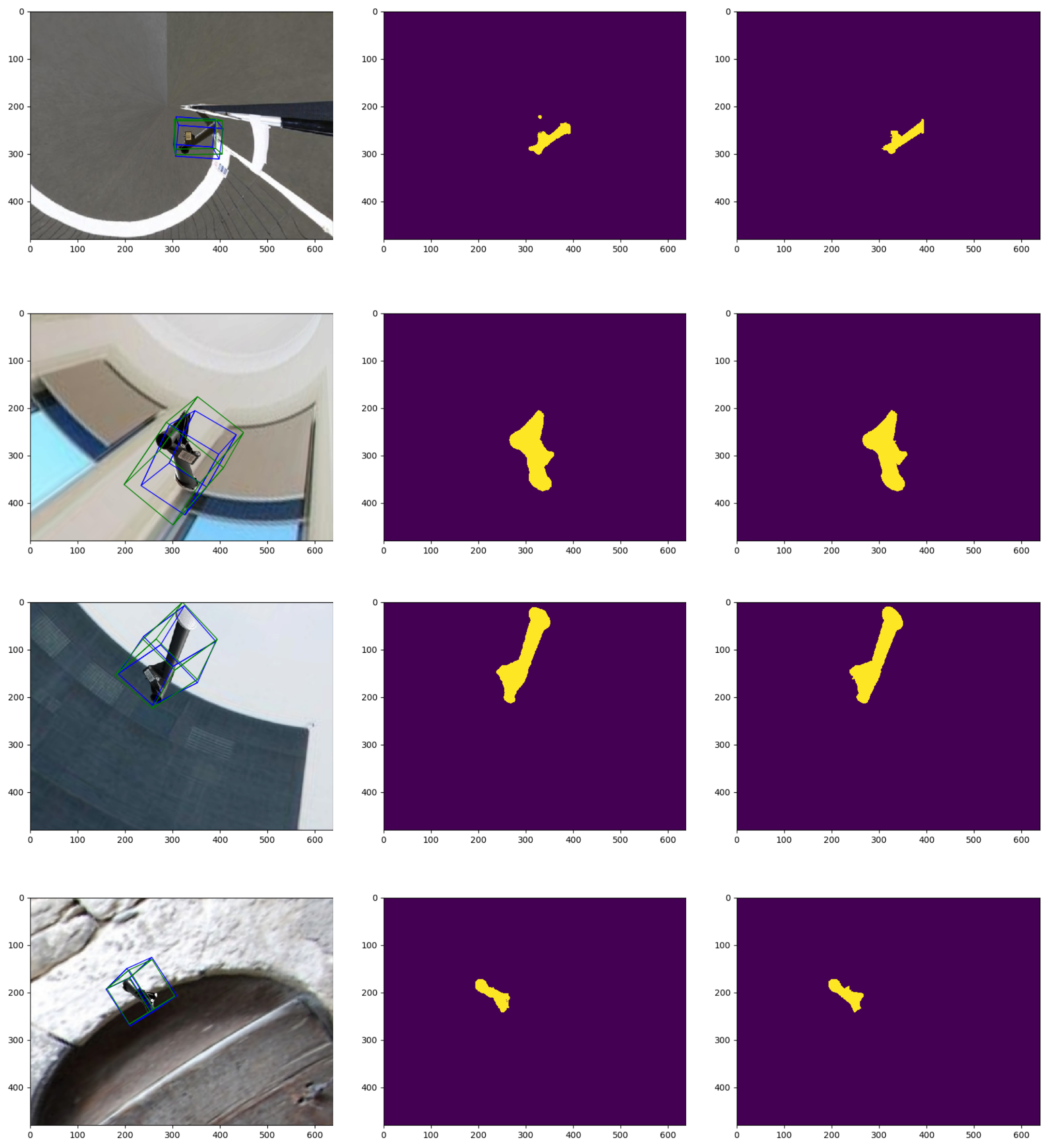


Figure 6.17: Pose estimation results on synthetic validation, part 2.

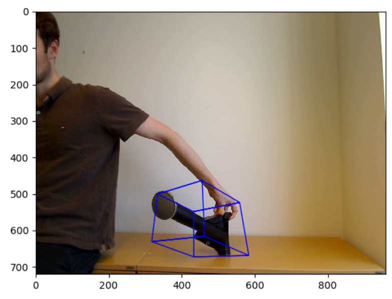
Table 6.5: Metrics that correspond to each instance in figure 6.16

image	seg	ver	precision	recall	dist[cm]	rot[°]
1	0.0022	0.0209	0.9633	0.09873	5.3	10.1
2	0.002860	0.02590	0.8860	0997	7.2	5.5
3	0.004	0.0299	0.94	0.980	10.9	6.9
4	0.0032	0.0245	0.912	0.983	14.2	5.0

Table 6.6: Metrics that correspond to each instance in figure 6.17

image	seg	ver	precision	recall	dist[cm]	rot[°]
1	0.00429	0.2619	0.890	0.9036	10.9	11.0
2	0.00239	0.0346	0.9766	0.9854	19.1	15.4
3	0.00365	0.0284	0.962	0.976	7.1	5.4
4	0.0023	0.0611	0.9384	0.9211	9.9	3.1

Real data


(a) Pose prediction on a real image (1).

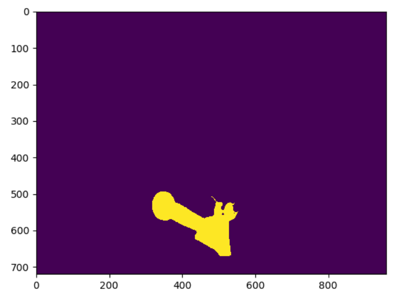
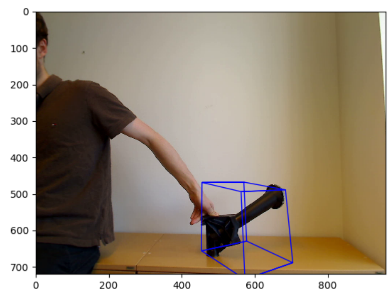

(b) Mask prediction on a real image (1).

Figure 6.18: These images are taken from a verification sequence performed with a model at epoch 109.


(a) Pose prediction inference on a real image (2).

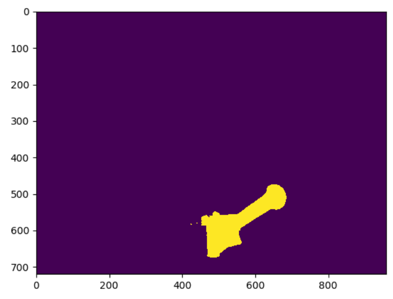

(b) Mask prediction on a real image (2).

Figure 6.19: These images illustrate a better pose estimation sequence performed with a model at epoch 45.

Occlusion results

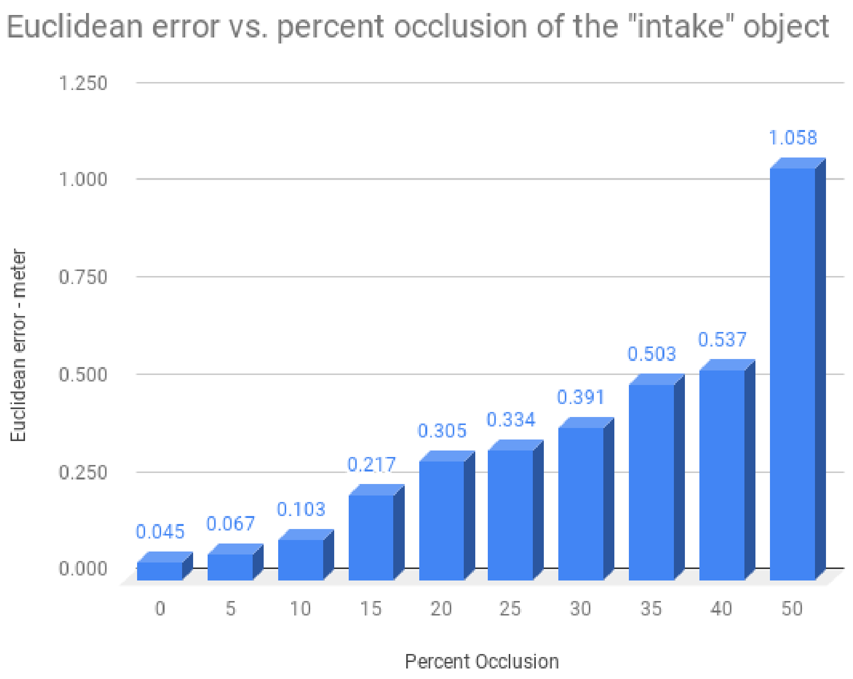


Figure 6.20: Graph of successful pose estimation vs percentage of truncation.

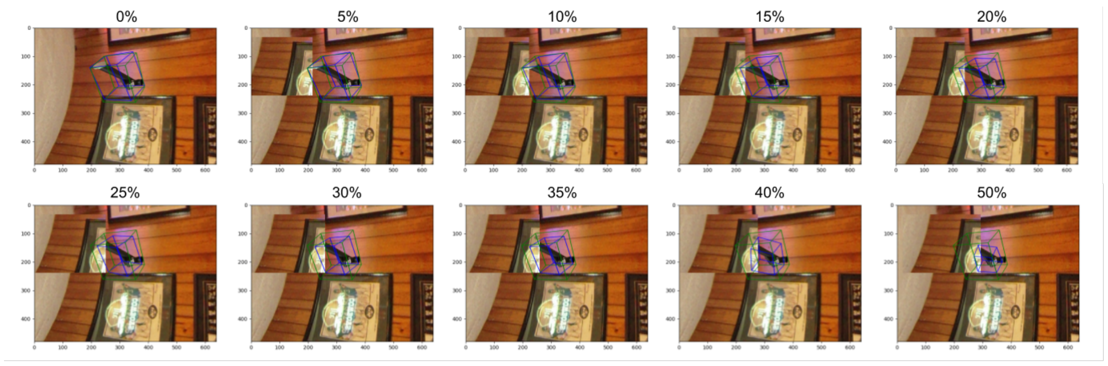


Figure 6.21: Images of predicted pose while the object is gradually truncated.

Deducing from graph 6.20 and figure 6.21 we see that the algorithm is interpreting the truncated mask as an object observed farther away. It is therefore clear that an occluded dataset is needed to be included in the training data in order to obtain a more accurate pose estimation on occluded objects.

Observations

From the verification sequence performed on real video, the images in Figures 6.19 and 6.18 were randomly selected. When running inference on the video with a model trained to epoch 45 and 109 respectively, the model that is trained shorter performs better. Occasionally, the algorithm also interprets some of the points as if they were located only a few centimeters from the camera. However, the object is always rendered at a distance greater than one meter from the camera. Even though this is quite a substantial discrepancy, the vertex loss is between 20% and 3%, with a mode of 5% throughout the verification set. This is because the vertices are regressed in the two-dimensional image and the RANSAC algorithm decides that this is the most likely pose, given the spatial probability distribution of 3D points.

It is important to acknowledge that this neural network has not seen a single real image and has only built its knowledge from synthetic images. The drop in performance is, therefore expected.

6.3 Evaluation summary

Some general takeaways from the results are:

- Segmentation is mostly sensitive to lighting.
- In early stages of synthetic training, the network often has lower confidence and is often predicting wrong brand when classifying the oil filters.
- In later stages of synthetic training, the network usually specializes more and more on smaller domain specific details, which yields worse performance on real images.
- The pose estimation is very dependent on a good mask.
- The pixel wise voting specialized after epoch 50, best generalization is therefore achieved in the interval 30-50.

Direct lighting, especially on metallic and shiny objects, creates high gloss patches due to the reflection of light rays. This creates in the best cases an oversaturated mask, and in the worst cases a hole in the mask. On the one hand, a type of mask refinement step [46] could fix this in post-processing. On the other hand, it is possible to simulate these high brilliance occurrences during training or have the method run in a muted light environment.

The second observation are derived from extensive empirical testing on real videos. The problem was solved using uniform light randomization in the augmentations, but the concept could be taken further. However, training the network for too long, it loses its generalization features and starts to become specialized on rendered objects.

Since it is beneficial to have a great mask, this could be refined in a post-processing step before the pixel-wise voting takes place. The pixel voting would benefit from being trained in terms of accuracy, but shorter in terms of generalization to real images. This means that the domain gap between the synthetic data and the real data needs to be decreased in order to train over more epochs without losing performance.

6.4 Research questions

In the introduction of this thesis, the questions below were raised.

- Do we see a better result with Mask R-CNN after using a bigger dataset and using data augmentation?
- What can we expect in accuracy, when running inference on real images?
- How does the generalization from synthetic to real images behave during training?
- How does PVNet, trained on synthetic data, perform on synthetic and real validation sets?

With regards to the first research question, yes, we do see a better result with the new dataset. Training the model for more than 40 epochs is crucial to obtain a lower class loss when running inference on real images. Training the model less than 100 epochs is beneficial to retain good recall of the objects. Regarding the second research question in the bullet list, using the MaskRCNN method, we can expect accuracies shown in Tables 6.1 and 6.2. Using the PVNet method, we can expect an IoU ranging from 39% to 97%, depending on how long the model is trained, how many similar objects are in the scene and how cluttered the environment is in general. Regarding the third research question, even at this stage of using photo-realistic renders with principle shading, randomized lighting, noise components, and more the approximation the synthetic data optimization still diverges at one stage from real images. Therefore the models need to be selected at an epoch where the loss is low, and generalization across domains is still good. Regarding the last research question, the pose estimation varies in accuracy. It is reliant on a correct mask prediction, and the masking accuracies vary. As a consequence, the accuracy of the pose estimation varies as well. This is because the mask is used for generating keypoint predictions. On synthetic data, it performs slightly worse than the cat from the LineMod dataset and is therefore predicted to perform slightly under 70% in terms of the ADD passing rate on synthetic validation data. This means that the class is slightly harder to estimate based on its appearance and shape. On real images, the estimates are varying a lot in performance. Shorter trained models perform better overall, as they retain more of the generalization from the transfer learning. The estimations can be used as good initial guesses for further pose refinement or used, as is, in systems with feedback control.

7 Discussion

In this chapter, we discuss the results presented in the previous chapter and application specific advice is given.

7.1 Quality inspection

In order to let a system make decisions on the manufacturing line, very high precision and repeatability are needed. For this, the current state of the available pose estimation algorithms is not sufficiently refined with the set limitations. However, the topic is being heavily researched, and many of the papers investigated prior to choosing the PVNet algorithm has been published in the last year. Thus, it is likely that there will be more accurate algorithms for this problem in the near future. The error of the pose estimation, however, would most likely need to be less than one centimeter and one degree in order to be reliable. For the investigated part of the assembly line, the cameras are supposedly mounted approximately three meters from where the parts are assembled. This requires that the cameras have a high resolution or narrow field of view, which in turn increases the computational complexity and makes the implementation expensive. As a standalone system, the problem is currently intractable, but the quality inspection could be assisted by incorporating information sharing between the other robotic systems working in the same section of the assembly line.

The oil filter has a decal that is applied each time differently in the production, making a rotational pose prediction based on features found in the decal ambiguous. The filters also have threads with multiple entry points and mechanical inaccuracies, making the pose estimation quite hard, insufficiently accurate, and even maybe unnecessary. The metric of most interest is the fact that it is mounted and that the normal direction is orthogonal to some other plane of the motor. It might be enough to determine this from an accurate mask prediction, but introducing depth cameras could also assist.

We observed that without augmentation, the Volvo bypass filter was predicted accurately and with high confidence when it was sufficiently illuminated. However, it was, with a high confidence, predicted as a Renault filter under other illumination circumstances. In the training data, the light intensity and direction were constant in a fashion that is similar to how the lighting is at Volvo's factories. To make the model more robust on a broader range of scenarios, we suggest that one or two point light sources with random location and intensity are introduced in the rendering step in addition to additive augmentations.

In terms of occlusions, the network did not perform very well while undergoing simple tests. A rough estimate is that any more significant occlusion than 20% will start to affect the recall.

Although the network performed better than the previous work, it was still somewhat prone to false positives, although the confidences were usually under 75%. A threshold of 85% or similar would block these from being interpreted as predictions. The occurrence of false positives varied a lot with respect to the level of illumination in the scene. Some false positives can be justified to a certain degree since they only inspect the region proposals in the image and are not using causal reasoning. For example, the metallic circular part of the oil intake sort of looks like a Renault oil filter seen from above. Similarly, a black textile formed in a certain way might make a vague resemblance to the oil suction intake.

7.2 Future work

For bin picking, there is usually a lot of occluded and truncated parts lying in different orientations. If the remaining challenges with the pose estimation in this thesis are solved, it could be valuable to implement this method in bin picking systems. Evaluating the method when it also has been trained on an occluded dataset, is the first step towards this.

For these algorithms to be used in a collaborative setting, it would be crucial to also know where the human is at all times. These human pose detection networks [27, 47] could worth investigating more in that case.

Incorporating occlusion and truncations in the dataset for training is deemed necessary in order to achieve overall better results.

Measures could be taken in order to obtain a more overlapping solution manifold of synthetic and real images. Training a general adversarial network (GAN) to create more realistic images from synthetic images could be a feasible way to do this.

For the pose detection scenarios yield inaccurate results, the method could be extended with a pose refinement approach, by utilizing the loss function proposed in [48] or similar. Although it is worth noticing that [48] had some problems when the target objects were partially occluded by other objects that were similar in color or shape, so in some cases, it might make the pose worse.

For real-time pose estimation in high FPS, resolution or on slower hardware, it might be beneficial to make use of temporal consistency between frames with regards to pose estimation and shrinking the search space.

Nvidia has recently released a framework for generating automatically annotated data sets, with their pose and depth in a artificial 3D environment through Unreal Engine 4. This framework is also able to utilize domain randomization, which in turn makes a network more generalized to the objects textures and thus increasingly focused on the structural properties of the object. Given that the CAD data is available for the object of interest, there are great opportunities to put up an automatic pipeline for data set generation and training of new objects.

8
Conclusion

In this thesis, instance segmentation and pose estimation have been considered as problems that need to be solved in order to increase the efficiency of production and quality of goods produced at manufacturing sites. For the sake of scalability and feasible implementation, a limitation of only using synthetically generated images of the assembly parts has been set for these problems.

Instance segmentation, with the use of neural networks, shows real promise for a multitude of industrial and consumer applications. We approached the instance segmentation problem by generating new datasets of rendered objects and retrained the last four layers of Mask R-CNN. We were able to achieve better results on a set of hand annotated real images by creating a larger dataset as well as testing different augmentation methods. By manipulating the light intensity, the class loss decreased, but the recall suffered at a confidence threshold of 60%. The accuracy of the semantic labeling is dependent on cluttering, occlusions, and light conditions, but in general, the results obtained were good when considering the limitations.

The pose estimation problem is very challenging, but one can reduce the solution space by semantically labeling the object first. We approached it similarly as we did with instance segmentation, but for the ResNet18, we retrained the whole network from ImageNet weights. This was done in order to reference it with PVNet's results more easily. Due to our limitations, we observed that the performance on real images worsened after training for more extended amounts of time, which supports the theory that the two domains inherently have different global minima. After running live inference on real video evaluation sequences, it was clear that the pose estimation is not very robust, so improvements need to be made before it is applicable in an industrial environment.

In a quality inspection setting, the instance segmentation results shown are indicating great promise. The methods work best in a well-illuminated scene, although direct shine can sometimes saturate the mask border or create holes in the mask predictions, especially for more shallow networks. In dark scenes, PVNet predicts scattered false positives, and the MaskRCNN classifies the filters wrong without augmentation of the training set. With intensity augmentation, it classified more accurately but performed worse in terms of recall.

We obtained slightly worse results using the ADD metric on our evaluation images with assembly parts than for LineMod objects. This was mostly due to rendering the objects further away from the camera. Since the method is very reliant on a good mask, the instance segmentation produced with our setup would need to perform better in order to be implemented on the production line. Without training on a truncated dataset, the pose estimation performs poorly. The method is overall better suited for bin picking applications as a slight inaccuracy can be tolerated.

Overall, it seems like this is a promising area to investigate further. At this stage, what we are left with is a trade-off between high accuracy and the ability to generalize. Instance segmentation generalizes well, but pose estimation is still reliant on additionally being trained on real images or more intricate augmentations in order to achieve higher accuracy on real images.

Bibliography

[1] Artificial neural network and neural network applications - xenonstack. https://www.xenonstack.com/blog/artificial-neural-network-applications/. (Accessed on 05/24/2019).

[2] Pytorch tensor basics. https://www.kdnuggets.com/2018/05/pytorch-tensor-basics.html. (Accessed on 05/23/2019).

[3] Interns explain cnn – data wow. https://blog.datawow.io/interns-explain-cnn-8a669d053f8b. (Accessed on 06/15/2019).

[4] Fully convolutional neural networks for object detection. https://www.azoft.com/blog/fully-convolutional-neural-networks/. (Accessed on 05/28/2019).

[5] Tensorflow conv2d layers: A practical guide - :. https://missinglink.ai/guides/deep-learning-frameworks/tensorflow-conv2d-layers-practical-guide/. (Accessed on 06/15/2019).

[6] Chris Kawatsu, Frank Koss, Andy Gillies, Aaron Zhao, Jacob Crossman, Benjamin Purman, Dave Stone, and Dawn Dahn. Gesture recognition for robotic control using deep learning. 08 2017.

[7] Loss functions — ml cheatsheet documentation. https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html. (Accessed on 06/15/2019).

[8] functions - how do you minimize "hinge-loss"? - mathematics stack exchange. https://math.stackexchange.com/questions/782586/how-do-you-minimize-hinge-loss. (Accessed on 06/15/2019).

[9] Plots of the l1, l2 and smooth l1 loss functions. | download scientific diagram. https://www.researchgate.net/figure/Plots-of-the-L1-L2-and-smooth-L1-loss-functions_fig4_321180616. (Accessed on 06/15/2019).

[10] Sklearn model selection. https://sklearn.org/auto_examples/model_selection/plot_underfitting_overfitting.html#sphx-glr-auto-examples-model-selection-plot-underfitting-overfitting-py. (Accessed on 06/15/2019).

[11] Jason Brownlee. How to stop training deep neural networks at the right time using early stopping. https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stoppi l. (Accessed on 05/23/2019).

[12] https://upload.wikimedia.org/wikipedia/commons/3/3b/Pinhole-camera.svg. (Accessed on 06/15/2019).

[13] Kenji Hata and Silvio Savarese. Cs231a course notes 1: Camera models. https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf. (Accessed on 06/15/2019).

[14] cameras — openmvg library. https://openmvg.readthedocs.io/en/latest/openMVG/cameras/cameras/. (Accessed on 06/15/2019).

[15] Aurélien Géron. https://www.oreilly.com/ideas/introducing-capsule-networks. (Accessed on 06/15/2019).

[16] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL http://arxiv.org/abs/1311.2524.

[17] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL http://arxiv.org/abs/1506.01497.

[18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL http://arxiv.org/abs/1703.06870.

[19] Github - facebookresearch/densepose: A real-time approach for mapping all human pixels of 2d rgb images to a 3d surface-based model of the body. https://github.com/facebookresearch/DensePose. (Accessed on 05/24/2019).

[20] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. Pvnet: Pixel-wise voting network for 6dof pose estimation. In *CVPR*, 2019.

[21] Overview of augmenters — imgaug 0.2.9 documentation. https://imgaug.readthedocs.io/en/latest/source/augmenters.html. (Accessed on 06/15/2019).

[22] Alvaro Gonz´alez. Measurement of areas on a sphere using fibonacci and latitude–longitude lattices. *CoRR*, 2009. URL https://arxiv.org/abs/0912.4540.

[23] David de la Iglesia. https://medium.com/@daviddelaiglesiacastro/3d-point-cloud-generation-from-3d-triangular-mesh-bbb602ecf238. (Accessed on 06/16/2019).

[24] Intersection over union (iou) for object detection - pyimagesearch. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. (Accessed on 06/15/2019).

[25] Roberto Opromolla, Giancarmine Fasano, Giancarlo Rufino, and Michele Grassi. A model-based 3d template matching technique for pose acquisition of an uncooperative space object. *Sensors (Basel, Switzerland)*, 15:6360–82, 03 2015. doi: 10.3390/s150306360.

[26] Georgios Pavlakos, Xiaowei Zhou, Aaron Chan, Konstantinos G. Derpanis, and Kostas Daniilidis. 6-dof object pose from semantic keypoints. *CoRR*, abs/1703.04670, 2017. URL http://arxiv.org/abs/1703.04670.

[27] Riza Alp Güler, Natalia Neverova, and Iasonas Kokkinos. Densepose: Dense human pose estimation in the wild. *CoRR*, abs/1802.00434, 2018. URL http://arxiv.org/abs/1802.00434.

[28] UN Committee on Economic, Social and Cultural Rights, (CESCR). General comment no. 18: The right to work (art. 6 of the covenant). 2006. URL https://www.refworld.org/docid/4415453b4.html.

[29] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *CoRR*, abs/1804.07612, 2018. URL http://arxiv.org/abs/1804.07612.

[30] D. Randall Wilson and Tony R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Netw.*, 16(10):1429–1451, December 2003. ISSN 0893-6080. doi: 10.1016/S0893-6080(03)00138-2. URL http://dx.doi.org/10.1016/S0893-6080(03)00138-2.

[31] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313.

[32] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017. URL http://arxiv.org/abs/1708.02002.

[33] Da-Wen Sun. *Image Segmentation Techniques*, chapter 3, pages 41–52. Elsevier, 2011. ISBN 0080556248.

[34] Michael D. Grossberg Yu-Chi J. Hu and Gikas S. Mageras. Semi-automatic medical image segmentation with adaptive local statistics in conditional random fields framework. 2008. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4649859.

[35] Anurag Arnab and Philip H. S. Torr. Pixelwise instance segmentation with a dynamically instantiated network. *CoRR*, abs/1704.02386, 2017. URL `http://arxiv.org/abs/1704.02386`.

[36] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: making rgb-based 3d detection and 6d pose estimation great again. *CoRR*, abs/1711.10006, 2017. URL `http://arxiv.org/abs/1711.10006`.

[37] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 2938–2946, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.336. URL `http://dx.doi.org/10.1109/ICCV.2015.336`.

[38] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *CoRR*, abs/1711.00199, 2017. URL `http://arxiv.org/abs/1711.00199`.

[39] Mai Bui, Sergey Zakharov, Shadi Albarqouni, Slobodan Ilic, and Nassir Navab. When regression meets manifold learning for object recognition and pose estimation. 05 2018.

[40] Mahdi Rad and Vincent Lepetit. BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. *CoRR*, abs/1703.10896, 2017. URL `http://arxiv.org/abs/1703.10896`.

[41] Tomáš Hodaň, Pavel Haluza, Štěpán Obdržálek, Jiří Matas, Manolis Lourakis, and Xenophon Zabulis. T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.

[42] Lilita Kiforenko, Bertram Drost, Federico Tombari, Norbert Krüger, and Anders Glent Buch. A performance evaluation of point pair features. *Computer Vision and Image Understanding*, 166:66 – 80, 2018. ISSN 1077-3142. doi: https://doi.org/10.1016/j.cviu.2017.09.004. URL `http://www.sciencedirect.com/science/article/pii/S1077314217301601`.

[43] Henning Tjaden, Ulrich Schwanecke, and Elmar Schömer. Real-time monocular pose estimation of 3d objects using temporally consistent local color histograms. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 124–132, 2017.

[44] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In Kyoung Mu

Lee, Yasuyuki Matsushita, James M. Rehg, and Zhanyi Hu, editors, *Computer Vision – ACCV 2012*, pages 548–562, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-37331-2.

[45] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. *CoRR*, abs/1711.00489, 2017. URL `http://arxiv.org/abs/1711.00489`.

[46] Jungeun Park, Chaewon Shin, and Chulyun Kim. PESSN: precision enhancement method for semantic segmentation network. In *IEEE International Conference on Big Data and Smart Computing, BigComp 2019, Kyoto, Japan, February 27 - March 2, 2019*, pages 1–4. IEEE, 2019. ISBN 978-1-5386-7789-6. doi: 10.1109/BIGCOMP.2019.8679313. URL `https://doi.org/10.1109/BIGCOMP.2019.8679313`.

[47] Daniel Castro, Steven Hickson, Patsorn Sangkloy, Bhavishya Mittal, Sean Dai, James Hays, and Irfan A. Essa. Let's dance: Learning from online dance videos. *CoRR*, abs/1801.07388, 2018. URL `http://arxiv.org/abs/1801.07388`.

[48] Fabian Manhardt, Wadim Kehl, Nassir Navab, and Federico Tombari. Deep model-based 6d pose refinement in RGB. *CoRR*, abs/1810.03065, 2018. URL `http://arxiv.org/abs/1810.03065`.

A
Appendix

A.1 ResNet18s

```
ResNet(
    (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (layer2): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (layer3): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2),
        bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2),
        bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2),
        bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2), dilation=(2, 2),
        bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(4, 4), dilation=(4, 4),
        bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(4, 4), dilation=(4, 4),
        bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(4, 4), dilation=(4, 4),
        bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(4, 4), dilation=(4, 4),
        bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=3)
    (fc): Sequential(
      (0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace)
    )
  )
  (conv8s): Sequential(
    (0): Conv2d(384, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.1, inplace)
  )
  (up8sto4s): UpsamplingBilinear2d(scale_factor=2, mode=bilinear)
  (conv4s): Sequential(
    (0): Conv2d(192, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.1, inplace)
  )
  (up4sto2s): UpsamplingBilinear2d(scale_factor=2, mode=bilinear)
  (conv2s): Sequential(
    (0): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.1, inplace)
  )
  (up2storaw): UpsamplingBilinear2d(scale_factor=2, mode=bilinear)
  (convraw): Sequential(
    (0): Conv2d(35, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.1, inplace)
    (3): Conv2d(32, 20, kernel_size=(1, 1), stride=(1, 1))
  )
)
```

A.1.1 Forward when forward-passing a 640×480 image

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 320, 240]           9,408
       BatchNorm2d-2         [-1, 64, 320, 240]             128
              ReLU-3         [-1, 64, 320, 240]               0
         MaxPool2d-4         [-1, 64, 160, 120]               0
            Conv2d-5         [-1, 64, 160, 120]          36,864
       BatchNorm2d-6         [-1, 64, 160, 120]             128
              ReLU-7         [-1, 64, 160, 120]               0
            Conv2d-8         [-1, 64, 160, 120]          36,864
       BatchNorm2d-9         [-1, 64, 160, 120]             128
             ReLU-10         [-1, 64, 160, 120]               0
       BasicBlock-11         [-1, 64, 160, 120]               0
           Conv2d-12         [-1, 64, 160, 120]          36,864
      BatchNorm2d-13         [-1, 64, 160, 120]             128
             ReLU-14         [-1, 64, 160, 120]               0
           Conv2d-15         [-1, 64, 160, 120]          36,864
      BatchNorm2d-16         [-1, 64, 160, 120]             128
             ReLU-17         [-1, 64, 160, 120]               0
```

```
    BasicBlock-18         [-1, 64, 160, 120]               0
        Conv2d-19          [-1, 128, 80, 60]          73,728
   BatchNorm2d-20          [-1, 128, 80, 60]             256
          ReLU-21          [-1, 128, 80, 60]               0
        Conv2d-22          [-1, 128, 80, 60]         147,456
   BatchNorm2d-23          [-1, 128, 80, 60]             256
        Conv2d-24          [-1, 128, 80, 60]           8,192
   BatchNorm2d-25          [-1, 128, 80, 60]             256
          ReLU-26          [-1, 128, 80, 60]               0
    BasicBlock-27          [-1, 128, 80, 60]               0
        Conv2d-28          [-1, 128, 80, 60]         147,456
   BatchNorm2d-29          [-1, 128, 80, 60]             256
          ReLU-30          [-1, 128, 80, 60]               0
        Conv2d-31          [-1, 128, 80, 60]         147,456
   BatchNorm2d-32          [-1, 128, 80, 60]             256
          ReLU-33          [-1, 128, 80, 60]               0
    BasicBlock-34          [-1, 128, 80, 60]               0
        Conv2d-35          [-1, 256, 80, 60]         294,912
   BatchNorm2d-36          [-1, 256, 80, 60]             512
          ReLU-37          [-1, 256, 80, 60]               0
        Conv2d-38          [-1, 256, 80, 60]         589,824
   BatchNorm2d-39          [-1, 256, 80, 60]             512
        Conv2d-40          [-1, 256, 80, 60]          32,768
   BatchNorm2d-41          [-1, 256, 80, 60]             512
          ReLU-42          [-1, 256, 80, 60]               0
    BasicBlock-43          [-1, 256, 80, 60]               0
        Conv2d-44          [-1, 256, 80, 60]         589,824
   BatchNorm2d-45          [-1, 256, 80, 60]             512
          ReLU-46          [-1, 256, 80, 60]               0
        Conv2d-47          [-1, 256, 80, 60]         589,824
   BatchNorm2d-48          [-1, 256, 80, 60]             512
          ReLU-49          [-1, 256, 80, 60]               0
    BasicBlock-50          [-1, 256, 80, 60]               0
        Conv2d-51          [-1, 512, 80, 60]       1,179,648
   BatchNorm2d-52          [-1, 512, 80, 60]           1,024
          ReLU-53          [-1, 512, 80, 60]               0
        Conv2d-54          [-1, 512, 80, 60]       2,359,296
   BatchNorm2d-55          [-1, 512, 80, 60]           1,024
        Conv2d-56          [-1, 512, 80, 60]         131,072
   BatchNorm2d-57          [-1, 512, 80, 60]           1,024
          ReLU-58          [-1, 512, 80, 60]               0
    BasicBlock-59          [-1, 512, 80, 60]               0
        Conv2d-60          [-1, 512, 80, 60]       2,359,296
   BatchNorm2d-61          [-1, 512, 80, 60]           1,024
          ReLU-62          [-1, 512, 80, 60]               0
        Conv2d-63          [-1, 512, 80, 60]       2,359,296
   BatchNorm2d-64          [-1, 512, 80, 60]           1,024
          ReLU-65          [-1, 512, 80, 60]               0
    BasicBlock-66          [-1, 512, 80, 60]               0
```

```
           Conv2d-67          [-1, 256, 80, 60]       1,179,648
      BatchNorm2d-68          [-1, 256, 80, 60]             512
             ReLU-69          [-1, 256, 80, 60]               0
           ResNet-70        [[-1, 64, 320, 240],
                             [-1, 64, 160, 120],
                              [-1, 128, 80, 60],
                              [-1, 256, 80, 60],
                              [-1, 512, 80, 60],
                              [-1, 256, 80, 60]]              0
           Conv2d-71          [-1, 128, 80, 60]         442,368
      BatchNorm2d-72          [-1, 128, 80, 60]             256
        LeakyReLU-73          [-1, 128, 80, 60]               0
UpsamplingBilinear2d-74     [-1, 128, 160, 120]               0
           Conv2d-75         [-1, 64, 160, 120]         110,592
      BatchNorm2d-76         [-1, 64, 160, 120]             128
        LeakyReLU-77         [-1, 64, 160, 120]               0
UpsamplingBilinear2d-78      [-1, 64, 320, 240]               0
           Conv2d-79         [-1, 32, 320, 240]          36,864
      BatchNorm2d-80         [-1, 32, 320, 240]              64
        LeakyReLU-81         [-1, 32, 320, 240]               0
UpsamplingBilinear2d-82      [-1, 32, 640, 480]               0
           Conv2d-83         [-1, 32, 640, 480]          10,080
      BatchNorm2d-84         [-1, 32, 640, 480]              64
        LeakyReLU-85         [-1, 32, 640, 480]               0
           Conv2d-86         [-1, 20, 640, 480]             660
      Resnet18_8s-87         [[-1, 2, 640, 480],
                             [-1, 18, 640, 480]]              0
================================================================
Total params: 12,957,748
Trainable params: 12,957,748
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 3.52
Forward/backward pass size (MB): 25,918,692.19
Params size (MB): 49.43
Estimated Total Size (MB): 25,918,745.13
----------------------------------------------------------------
```

A.1.2 Forward when forward-passing a 1280×960 image

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 640, 480]           9,408
       BatchNorm2d-2         [-1, 64, 640, 480]             128
              ReLU-3         [-1, 64, 640, 480]               0
         MaxPool2d-4         [-1, 64, 320, 240]               0
            Conv2d-5         [-1, 64, 320, 240]          36,864
       BatchNorm2d-6         [-1, 64, 320, 240]             128
```

```
             ReLU-7          [-1, 64, 320, 240]               0
           Conv2d-8          [-1, 64, 320, 240]          36,864
      BatchNorm2d-9          [-1, 64, 320, 240]             128
            ReLU-10          [-1, 64, 320, 240]               0
      BasicBlock-11          [-1, 64, 320, 240]               0
          Conv2d-12          [-1, 64, 320, 240]          36,864
     BatchNorm2d-13          [-1, 64, 320, 240]             128
            ReLU-14          [-1, 64, 320, 240]               0
          Conv2d-15          [-1, 64, 320, 240]          36,864
     BatchNorm2d-16          [-1, 64, 320, 240]             128
            ReLU-17          [-1, 64, 320, 240]               0
      BasicBlock-18          [-1, 64, 320, 240]               0
          Conv2d-19         [-1, 128, 160, 120]          73,728
     BatchNorm2d-20         [-1, 128, 160, 120]             256
            ReLU-21         [-1, 128, 160, 120]               0
          Conv2d-22         [-1, 128, 160, 120]         147,456
     BatchNorm2d-23         [-1, 128, 160, 120]             256
          Conv2d-24         [-1, 128, 160, 120]           8,192
     BatchNorm2d-25         [-1, 128, 160, 120]             256
            ReLU-26         [-1, 128, 160, 120]               0
      BasicBlock-27         [-1, 128, 160, 120]               0
          Conv2d-28         [-1, 128, 160, 120]         147,456
     BatchNorm2d-29         [-1, 128, 160, 120]             256
            ReLU-30         [-1, 128, 160, 120]               0
          Conv2d-31         [-1, 128, 160, 120]         147,456
     BatchNorm2d-32         [-1, 128, 160, 120]             256
            ReLU-33         [-1, 128, 160, 120]               0
      BasicBlock-34         [-1, 128, 160, 120]               0
          Conv2d-35         [-1, 256, 160, 120]         294,912
     BatchNorm2d-36         [-1, 256, 160, 120]             512
            ReLU-37         [-1, 256, 160, 120]               0
          Conv2d-38         [-1, 256, 160, 120]         589,824
     BatchNorm2d-39         [-1, 256, 160, 120]             512
          Conv2d-40         [-1, 256, 160, 120]          32,768
     BatchNorm2d-41         [-1, 256, 160, 120]             512
            ReLU-42         [-1, 256, 160, 120]               0
      BasicBlock-43         [-1, 256, 160, 120]               0
          Conv2d-44         [-1, 256, 160, 120]         589,824
     BatchNorm2d-45         [-1, 256, 160, 120]             512
            ReLU-46         [-1, 256, 160, 120]               0
          Conv2d-47         [-1, 256, 160, 120]         589,824
     BatchNorm2d-48         [-1, 256, 160, 120]             512
            ReLU-49         [-1, 256, 160, 120]               0
      BasicBlock-50         [-1, 256, 160, 120]               0
          Conv2d-51         [-1, 512, 160, 120]       1,179,648
     BatchNorm2d-52         [-1, 512, 160, 120]           1,024
            ReLU-53         [-1, 512, 160, 120]               0
          Conv2d-54         [-1, 512, 160, 120]       2,359,296
     BatchNorm2d-55         [-1, 512, 160, 120]           1,024
```

```
          Conv2d-56       [-1, 512, 160, 120]         131,072
     BatchNorm2d-57       [-1, 512, 160, 120]           1,024
            ReLU-58       [-1, 512, 160, 120]               0
      BasicBlock-59       [-1, 512, 160, 120]               0
          Conv2d-60       [-1, 512, 160, 120]       2,359,296
     BatchNorm2d-61       [-1, 512, 160, 120]           1,024
            ReLU-62       [-1, 512, 160, 120]               0
          Conv2d-63       [-1, 512, 160, 120]       2,359,296
     BatchNorm2d-64       [-1, 512, 160, 120]           1,024
            ReLU-65       [-1, 512, 160, 120]               0
      BasicBlock-66       [-1, 512, 160, 120]               0
          Conv2d-67       [-1, 256, 160, 120]       1,179,648
     BatchNorm2d-68       [-1, 256, 160, 120]             512
            ReLU-69       [-1, 256, 160, 120]               0
          ResNet-70       [[-1, 64, 640, 480],
                           [-1, 64, 320, 240],
                          [-1, 128, 160, 120],
                          [-1, 256, 160, 120],
                          [-1, 512, 160, 120],
                          [-1, 256, 160, 120]]              0
          Conv2d-71       [-1, 128, 160, 120]         442,368
     BatchNorm2d-72       [-1, 128, 160, 120]             256
       LeakyReLU-73       [-1, 128, 160, 120]               0
UpsamplingBilinear2d-74   [-1, 128, 320, 240]               0
          Conv2d-75        [-1, 64, 320, 240]         110,592
     BatchNorm2d-76        [-1, 64, 320, 240]             128
       LeakyReLU-77        [-1, 64, 320, 240]               0
UpsamplingBilinear2d-78    [-1, 64, 640, 480]               0
          Conv2d-79        [-1, 32, 640, 480]          36,864
     BatchNorm2d-80        [-1, 32, 640, 480]              64
       LeakyReLU-81        [-1, 32, 640, 480]               0
UpsamplingBilinear2d-82   [-1, 32, 1280, 960]               0
          Conv2d-83       [-1, 32, 1280, 960]          10,080
     BatchNorm2d-84       [-1, 32, 1280, 960]              64
       LeakyReLU-85       [-1, 32, 1280, 960]               0
          Conv2d-86       [-1, 20, 1280, 960]             660
     Resnet18_8s-87      [[-1,  2, 1280, 960],
                          [-1, 18, 1280, 960]]              0
================================================================
Total params: 12,957,748
Trainable params: 12,957,748
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 14.06
Forward/backward pass size (MB): 414,714,768.75
Params size (MB): 49.43
Estimated Total Size (MB): 414,714,832.24
----------------------------------------------------------------
```