Tampere University

Janne Siltanen

DETECTING OBJECTS IN CARGO HANDLING OPERATIONS FROM 3D POINT CLOUD DATA

Master of Science Thesis
Faculty of Engineering and
Natural Sciences
Examiners: Prof. Matti Vilkko
and Ass.prof David Hästbacka
April 2022

ABSTRACT

Janne Siltanen: Detecting objects in cargo handling operations from 3D point cloud data
Master of Science Thesis
Tampere University
Automation Engineering
April 2022

The objective of this master thesis is to research existing 3D point cloud deep learning methods and their ability to detect keypoints in automatic cargo handling operations. Point clouds contain spatial information, which is one reason why they have gained popularity in automated operations in many industries. Deep learning can be utilized to detect points of interest from all kinds of data and research on detection from point clouds has increased recently.

This master thesis researches the usage of convolutional and transformer-based deep neural networks for detecting key points from point cloud data. Six different point cloud deep learning methods will be covered and four different methods will be selected and utilized for testing with the data from automatic cargo handling operations. The methods will be tested for inference time and accuracy and compared to the method currently used in the production system. Inference time will be tested on edge environments and training server environment.

The thesis first presents the background of the problem in the introduction along with the research questions. Theoretical background of the related topics will be presented. Deep learning with point clouds has its own characteristics and problems when compared to 2D images so those will be covered in the theoretical part as well. This thesis will also cover how the training is executed, what kind of tests are used and how the used scripts operate.

The results show that in terms of accuracy convolutional-based method performed the best out of the researched methods and it outperformed the currently used method. Good results were also obtained with transformer-based models. In terms of inference time, not all methods can be utilized in an edge machine learning environment unless GPU computation is available.

Keywords: keypoint detection, deep learning, point clouds, convolutional neural networks, transformer neural networks

TIIVISTELMÄ

Janne Siltanen: Kohteiden havaitseminen lastinkäsittelyoperaatioissa 3D pistepilvidatan avulla
Diplomityö
Tampereen yliopisto
Automaatiotekniikka
Huhtikuu 2022

Tämän diplomityön tavoitteena on tutkia olemassa olevia 3D-pistepilven syväoppimismenetelmiä ja niiden kykyä havaita avainpisteitä automaattisissa lastinkäsittelyoperaatioissa. Pistepilvet sisältävät tilallista informaatiota mikä on yksi syy ne ovat saavuttaneet suosiota automaattisisssa operaatioissa eri teollisuuden aloilla. Syväoppimista voidaan hyödyntää erilaisten kiinnostavien kohteiden löytämiseen erilaisista datatyypeista ja nimenomaan pistepilvistä havaitsemisen tutkimus on ollut kasvussa viime aikoina.

Tässä diplomityössä tutkitaan konvoluutioneuroverkkoihin ja transformer-neuroverkkoihin perustuvien mallien käyttöä avainpisteiden havaitsemisesta pistepilvidatasta. Kuusi erilaista pistepilvisyväoppimismallia esitellään ja neljä näistä valitaan testaukseen automaattisen lastinkäsittelyoperaation datalla. Menetelmien inferenssiaika ja tarkkuus testataan ja tuloksia verrataan tuotanjärjestelmässä tällä hetkellä käytössä olevaan menetelmään. Inferenssiaika testataan tuotantoympäristöissä sekä koulutuspalvelinympäristössä.

Diplomityö alkaa johdannolla, jossa esitellään ensin ongelman tausta sekä tutkimuskysymykset. Asiaan liittyvien aiheiden teoreettinen tausta esitellään teorialuvussa. Pistepilvien syväoppimisessa on omat piirteensä ja ongelmansa verrattuna 2D-kuvien syväoppimiseen, joten niitä käsitellään myös teoriaosuudessa. Tässä työssä käydään läpi myös kuinka koulutus suoritetaan, millaisia testejä käytetään ja miten käytetyt skriptit toimivat.

Tulokset osoittavat, että tarkkuudeltaan konvoluutioneuroverkkopohjainen menetelmä suoriutui parhaiten tutkituista menetelmistä ja sen tarkkuus oli parempi kuin 2D intensiteettikuviin perustuva menetelmä. Hyviä tuloksia saatiin myös transformer-pohjaisilla menetelmillä. Inferenssiajan testauksessa huomattiin ettei kaikki menetelmiä voida hyödyntää tuotantoympäristössä, ellei GPU-laskentaa ole saatavilla.

Avainsanat: avainpistetunnistus, syväoppiminen, pistepilvet, konvoluutioneuroverkot, transformer-neuroverkot

PREFACE

This thesis was carried out in the fall of 2021 and spring 2022 for the Measurement Systems and Machine Learning team in Kalmar.

This thesis allowed me to study a whole new field of technology in top of my other studies. I haven't completed single machine learning course which meant I had to start learning from a scratch. It gave rise to its own challenges, but when I finished the job I am happy that I took on the challenge.

I would like to thank Lari Melander for providing me with the subject and supporting me during the whole thesis and Topi Hintikka for his expertise and support on the topic. I would also like to thank the supervisors Matti Vilkko and David Hästbacka for giving feedback throughout this project. Additionally I am grateful for my family and friends who have been supporting me during my studies.

Tampere, 26 April 2022

Janne Siltanen

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture. Parallel computing platform and programming model created by Nvidia
CV	Computer Vision
FN	False negative
FP	False positive
FP16	Floating point precision of 16 bits
FP32	Floating point precision of 32 bits
Lidar	Light detection and ranging
LSTM	Long short-term memory
ML	Machine Learning
MLP	Multi Layer Perceptron, a class of feedforward artificial neural network with one or more hidden layers
NLP	Natural Language Processing
NN	Neural network
PCA	Principal Component Analysis
SLP	Single Layer Perceptron, a class of feedforward artificial neural network without hidden layers
SVM	Support vector machine
T-Net	Transformation network
Tensor	Dimensional data structure
TN	True negative
ToF camera	Time-of-flight camera
TP	True positive
XML	Extensible Markup Language

1. INTRODUCTION

1.1 Motivation

Automation in various industry fields has grown massively during recent years. The use of software-based solutions has provided the users with safe operations and precise results on the tasks. Automation exists in nearly all fields of industry, but it has gained popularity in, for example, heavy machinery. It is being used to replace humans in dangerous working conditions and to gain consistent and accurate measuring results. Working with large machines in rough conditions such as container terminals are risky for humans and errors can be extremely costly.

Container terminals are crucial points from the view of cargo flow. They are complex logistical systems, which during the last few years have gradually begun to adopt automation. This is due to efficiency, cost reduction as well as safety concerns. Containers are moved and stacked with automatic cranes or straddle carriers. In day-to day operation these cranes move the containers from the ship transfer area to the container field which is used as a storage area. From there they are either moved to another ship or to the truck transfer area where trucks arrive to pick them up. In automated container terminals these transfer areas are also the boundaries of the automated area. The crane places the container on the truck chassis, which is a platform attached to a truck head. In each corner of the chassis is a connector called a twist lock, which are standardized size metal bumps where each corner casting of the container will be placed and secured. Another way to secure is with so-called gooseneck trailers, which have a narrow structure at the front of the chassis. The containers have a tunnel on the bottom frame of the container which will fit directly on the narrow part of the chassis.

If the automated straddle carriers handle all the container moves themselves the environment stays rather constant. The container locations are fixed within the field and the machines handle the positioning of the containers with little-to-no variance. When human operation is included in the process like in the truck transfer area more attention is required due to always changing target. The driver can drive the truck chassis at various angles and offsets. Additionally the amount of different chassis models and types makes it difficult to do consistent standardized automatic moves. A system which can detect the accurate location of the twist locks on the chassis on every container move is required.

Machine vision is a powerful method which can be utilized for the problem. With the use of 3D-cameras a point cloud of the chassis and the container can be captured. With the use of deep neural networks, it is possible to identify and measure the key points from the point cloud. Deep neural networks are a growing field of technology and are commonly used with images and sound, but they can also be applied to point cloud data.

Point clouds are usually noisy and contain objects which are not essential for the system, like truck cabins, concrete blocks or other parts of the field. This means the data must be processed accordingly before applying the machine learning algorithm.

1.2 State of art

Deep neural networks for processing traditional 2D-images are being used in many applications, but some practical applications require more information about the environment. Information provided by 2D-images sometimes is not enough and more spatial information is necessary. 3D point clouds are a way to capture dimensional information about the surrounding environment and deep learning methods for 3D point cloud data are being researched constantly at the moment. At the time of writing this thesis many of them are very recently published. These methods are more thoroughly introduced in Subchapter 3.2.

PointNet++ is a deep hierarchical feature learning method for point clouds created by Qi et al. at Stanford University. It is one of the most popular methods and is being used as a base for many other point cloud methods. It is based on the team's earlier paper PointNet and the improvements in PointNet++ pertain to the improved ability to generalize complex scenes and recognize fine-grained patterns. [1]

Dynamic Graph CNN (DGCNN) is a deep learning neural network architecture for point clouds created by Wang et al. at MIT in USA. The team created a novel neural network module called EdgeConv, which has ability to incorporate local neighborhood information from the point cloud. The team propose that EdgeConv based architectures are suitable for point cloud tasks like segmentation and classification. [2]

Point Cloud Transformer (PCT) is a point cloud learning framework based on the transformer network model. It was created by Guo et al. at Tsinghua University in Beijing, China. Due to its success in natural language processing (NLP) and image processing, the creators chose to apply the method to point clouds. Guo et al. state that transformer is inherently permutation invariant for processing a sequence of points, which makes it applicable for point cloud learning. [3]

1.3 Objectives and research questions

My task in this thesis is to test four predetermined methods for 3D point cloud data to detect key points from the truck chassis and container. The key points mentioned are the twist lock or the corner of the chassis and the corner casting of the container. The current way these points are detected is a combination of 2D intensity image and 3D point cloud. The researched methods are based on convolutional and transformer neural networks and will be compared to the currently used method. Interesting attributes to analyze are inference time, accuracy and how the model behaves in edge machine learning environment. The goal is to find how well convolutional and transformer neural network models can be utilized to find the key points straight from the point cloud and is it preferable to use in the end-user application instead of the currently used method. With these boundaries we can form the following research questions:

1. In terms of inference time are transformer and convolutional based deep learning inference models suitable for edge machine learning environment?
2. Is convolutional or transformer based deep learning inference model for point clouds better than current 2D intensity image based method in terms of accuracy?

Suitability in question one is referring to the algorithm's ability to do detection by utilizing the hardware available in the edge machine. If compared to manual operation a skillful manual straddle carrier driver can usually do this task within a few seconds. From this we can get a boundary since we don't want the automatic move to extend the operation excessively.

The first objective in the solution is to reshape the training data to be used in training and train the model. The training data has been gathered and labelled by the team. Dataset contains around 1000 point clouds collected from the automated customer terminal site. After the model has been trained it will be tested with some portion of the dataset. These results will be compared to the ones from the current method used in the application.

Theoretical background will be presented in the beginning of this thesis. The theoretical part will cover the relevant topics around the research questions and the functionality of used neural network structures. Principles of machine learning and characteristics of point clouds will also be presented.

2. MACHINE LEARNING

This chapter will go through the basics of machine learning and introduce the main principles and terminology which will be used later in the thesis. Subchapter 2.1 will review the principles of machine learning and what are the steps in a machine learning solution. This subchapter also introduces metrics used to evaluate models in machine learning. Neural networks and deep learning have gained massive popularity in machine learning solutions lately. These are covered in Subchapter 2.2. Convolutional neural networks are a way to utilize deep learning to multiple kinds of data and have gained popularity in image detection. Convolutional neural networks are covered in Subchapter 2.3. Transformer neural networks are a novel way to utilize deep learning and have gained multiple state-of-art solutions in for example natural language processing. Transformers are covered in Subchapter 2.4.

2.1 Principles of machine learning

Peter Flasch in his book “Machine Learning: The Art and Science of Algorithms that Make Sense of Data” summarizes machine learning as using the right features to build the right models that achieve the right tasks. [4, p. 13] Machine learning is a subset of the wider concept called artificial intelligence (AI), which is the art of making computers mimic human behavior. [5, p. 8]

Machine learning is used to tackle problems which cannot be solved with traditional algorithms. Commonly used example is detecting spam email. No algorithm can detect which email is spam and which is not but given enough examples the system can learn and do accurate approximations. A more technical approach is a computer that is programmed to optimize a performance criterion by using past experience or example data. A model is created with arbitrary parameters and the execution of this program optimizes the parameters with the each cycle the program runs. This continuous looping of the program is called training. [6, p. 3]

Data has gained a lot of importance during the growth of machine learning. It is used in teaching the models. In order to train the model most efficiently, the most important features must be extracted from the data. Features are properties or relevant parts which the system should be focusing on inside the data. They can be called the building blocks of the datasets. Features are captured from the raw data in a form which is feasible for the learning algorithm. In order to learn from the features, they need to have a label. Labelling means that each data sample needs to have a meaningful and descriptive label so that the model is able to learn from it. An example

could be a photo containing a car and having a label “car”. The strings are usually transformed into number representations since the algorithms handle only numbers.

Machine learning can roughly be divided into supervised learning and unsupervised learning. The difference is which setting the model is using and if the data is labelled. For example supervised learning is commonly used to create predictive models and its typical tasks are classification and regression. [4, p. 17] The goal of the supervised learning algorithm is to learn a mapping formula from the input data to an output. The correct values for the training process are provided by a supervisor, which can be considered as a teacher for the algorithm. Supervised learning is more or less divided into classification and regression problems. In the traditional cases when the output needs to be a binary value such as true or false, the learning problem is called classification. In the case that it needs to be a numerical value the problem is called regression. [7, p. 694]

Clustering data with the intention of using the clusters to assign class labels to new data can be called unsupervised learning. Unlike in supervised learning, unsupervised learning methods don’t have a supervisor, which means the data is unlabeled. The goal is to seek for characteristics or regularities from the data. [6, p. 11] Unsupervised learning algorithm is utilized to learn patterns from the input even though no explicit feedback is supplied. [7, p. 694] An example of unsupervised learning is density estimation, which is commonly used in statistical analysis. The algorithm attempts to find if certain patterns appear in the data more often than others. [6, p. 11] Unsupervised learning problems can be utilized into three main tasks; clustering, association and dimensionality reduction.

Reinforcement learning is third approach which can be classified as basic machine learning paradigm. The idea of reinforcement learning is that an agent have to learn behavior by trial and error interactions with its changing environment. The algorithm learns by doing mistakes and by getting information from a human. [8, p. 1] Reinforcement learning is commonly used with robotics and recommendation systems.

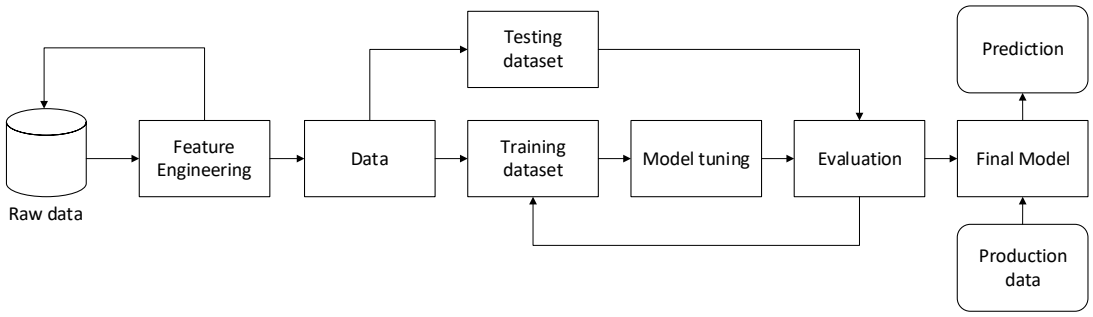

Figure 2.1 *Steps of machine learning solution. Figure adapted from* ***[9]****.*

Machine learning solution can be divided to a sequence of steps which are shown in the Figure 2.1. First step includes collecting data and making sure the quality is inherent. It is essential to fix the missing values, remove clearly unsuitable data points and harmonize the set. This operation is called feature engineering. It is essential to make the data as clear and simple as possible. This consists of selecting the important features and formatting the data to a format usable by the algorithm. Training and testing datasets are separated from the full dataset. Common ratio suggested in several literatures is 30% for validation and 70% for training. Occasionally the validation set is also divided equally for testing set and validation set. Next step is the selection of used algorithm and what kind of neural network architecture to use if using deep learning. If the model is not performing as desired the parameters for training can be adjusted. The model is tuned until the outcome is suitable. The final model can now be used to predict from the production data. [10, Ch. 4]

2.1.1 Evaluation metrics

When evaluating models ability to do predictions it is necessary to analyze what is the task the model is created for and what are the consequences if the model does not work as desired. To verify how correct the model predicts on different cases a tool called confusion matrix can be utilized.

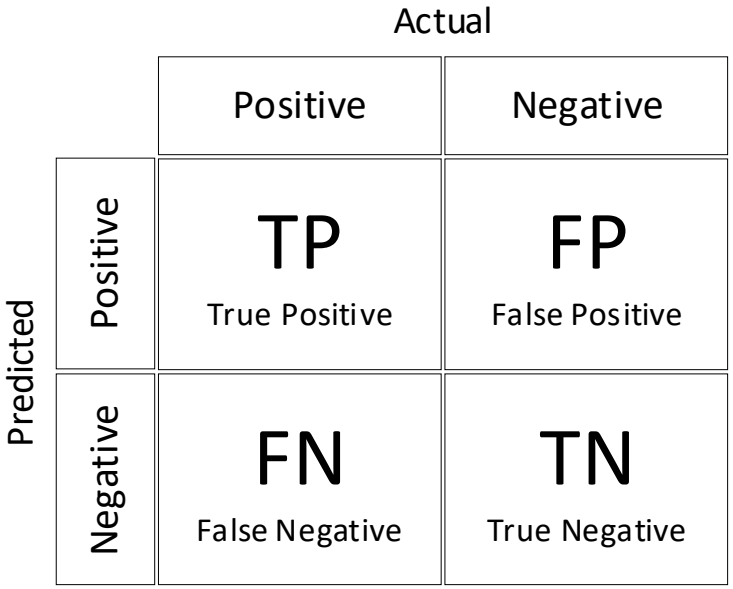

Figure 2.2 Confusion matrix. Figure adapted from ***[11, p. 79]***

Confusion matrix is a matrix where the actual ground truth values are compared to the models predicted values. The matrix is introduced in Figure 2.2. The base values shown in the middle of the matrix are used to analyze the results of the model.

- True positive (TP) is a result where the model has correctly predicted a positive value
- False positive (FP) is a result where the model has predicted the value to be positive, while the actual ground truth value is negative
- False negative (FN) is a result where the model has predicted a negative value while the actual ground truth value is positive
- True Negative (TN) is a result where the model has correctly predicted a negative value

These four different result types can be utilized in calculating different evaluation metrics. Two useful and frequently used metrics are precision and recall.

Precision is a metric which can be used to determine what amount of the positive predictions were correct. It can be calculated with following equation:

$$Precision = \frac{TP}{TP+FP}, \tag{2.1}$$

where TP is the amount of true positive predictions and FP is the amount of false positive predictions. With precision it is possible to analyze how correctly the model is able to predict positive outcomes. If the model is able to function without any false positives the precision has a value of 1.0. [12]

Recall is a metric which also takes negative predictions into consideration. It can be used to determine what amount of actual positive results were predicted correctly. Recall can be calculated with following equation:

$$Recall = \frac{TP}{TP+FN}, \tag{2.2}$$

where TP is the amount of true positive predictions and FN is the amount of false negative predictions. If the model is able to function without predicting any false negatives the recall value has a value of 1.0. [12]

2.2 Neural Networks and Deep Learning

Neural networks are a subset of machine learning, which are based on neural network models. They are a class of algorithms which are loosely inspired by the behavior of human brain and are utilized in both supervised and unsupervised learning tasks. One common definition for NN's is *a learning function from one space to another using data*. They can be used in all kinds of data formats such as videos, text, sound images and point clouds. [13, Ch. 2]

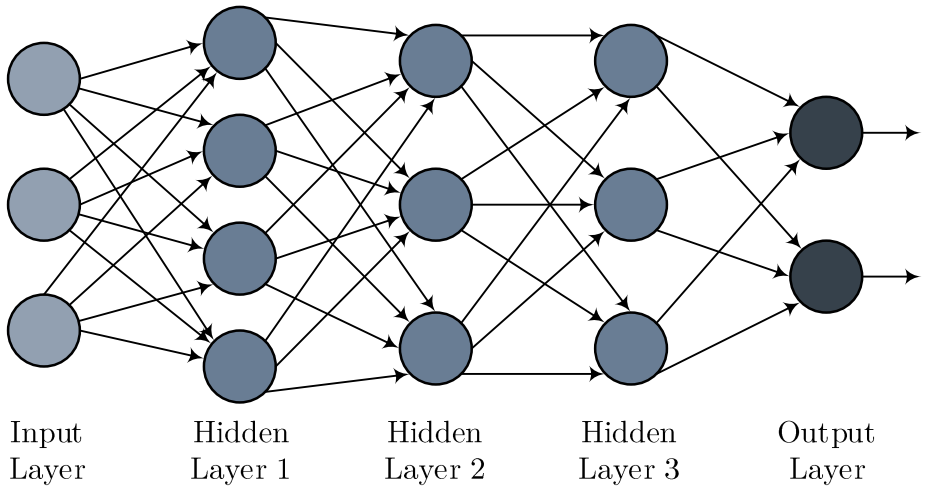

Figure 2.3 Feed-forward neural network with three hidden layers. Figure adapted from ***[14, p. 66]***

Neural networks are represented in node-like pattern to simplify the representation of a complex function. Figure 2.3 represents a feed-forward neural network. In feed-forward network the data flows only in one direction unlike in recurrent neural network where the information can flow in cycles.[15, Ch. 21.1] The feed-forward network consists of three types of layers: input layers, hidden layers and output layers. Input layers takes in the data in a data structure called input vector. The size of input layer is defined by the dimensions of the input data. In traditional neural networks the input layer is 1-dimensional so the input data needs to be flattened to feasible format. For example 64x64 pixel monochrome image would be flattened to a 1x4096 pixel

format before feeding it to the input layer. Input layer does not do any operations to the data, just forward it to hidden layers.

Hidden layers are the layers between input and output layers. Hidden layers handle the mathematical functionalities so they are the one where actual calculations occur. [5, p. 61] The calculations of basic feed-forward network are introduced later in this chapter. The amount of hidden layers define the type of the network. If the network does not contain any hidden layers, it is called Single Layer Perceptron (SLP). If there are one or more hidden layers like in the Figure 2.3, it is defined as Multi Layer Perceptron (MLP). [16, Ch. 2]

The results of the network come from the output layer. It can be set up to a format feasible for the problem setting. For classification problems it is common to see each class having its own output neuron representing the probability of the class in the input data. For binary classification the output can be two neurons representing the probability of True and False. [5, p. 61]

The building blocks of neural networks are called neurons. They are entities which can compute a simple function of its input to an output. [17] Each neuron takes in a group of numbers and calculates an output value with two-step process. First step is to calculate the weighted sum of the inputs and add bias to the result. The second step is to pass the result to an activation function. [14, p. 69] These operations are introduced in Figure 2.4.

Weights are each neurons parameters which represent how the neuron transfers information and how each neuron responds. It can be considered as how activated the particular neuron is and is generally presented as a value from 0 to 1. The neural output can e.g. present an edge or feature in an image or single note in a sound file. [5, p. 10]

Bias is a value which is added to the weighted sum. It can be considered as a constant from linear algebra. It is used to eliminate the need to get the decision boundary to go through origin. Decision boundary is a boundary the neural network uses to categorize the input items from each other. [14, p. 83]

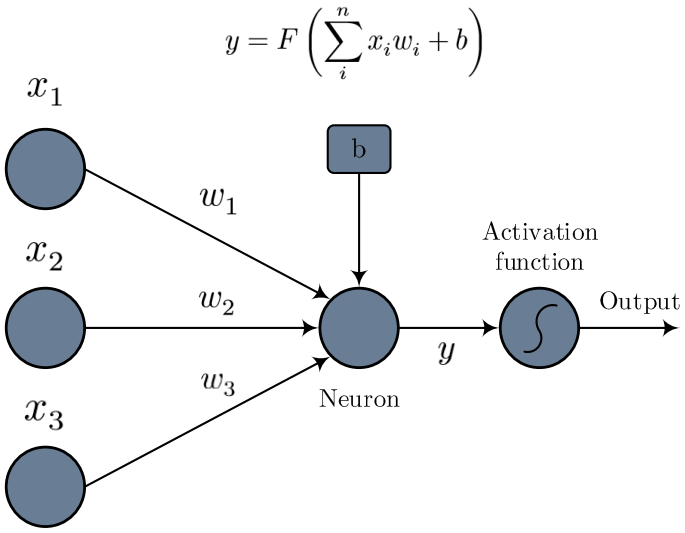

Figure 2.4 Representation of a neuron. Figure adapted from ***[5, p. 10]***

After the weighted sum is calculated and bias added, the result is fed to an activation function. As seen from the calculations so far, they are all linear operations. With plainly linear operations it is not possible to represent real-life complex nonlinear relationships. Therefore a nonlinear operation is required in order to get the best possible result from the network. Activation functions are used to convert the results to a nonlinear format. In principle, the use of any nonlinear function as an activation function grants the neural network an ability to learn a nonlinear mapping between inputs and outputs. Additionally many activation functions have good mathematical attributes, which are useful in the learning process. [14, p. 75] Three commonly used functions are introduced in Figure 2.5.

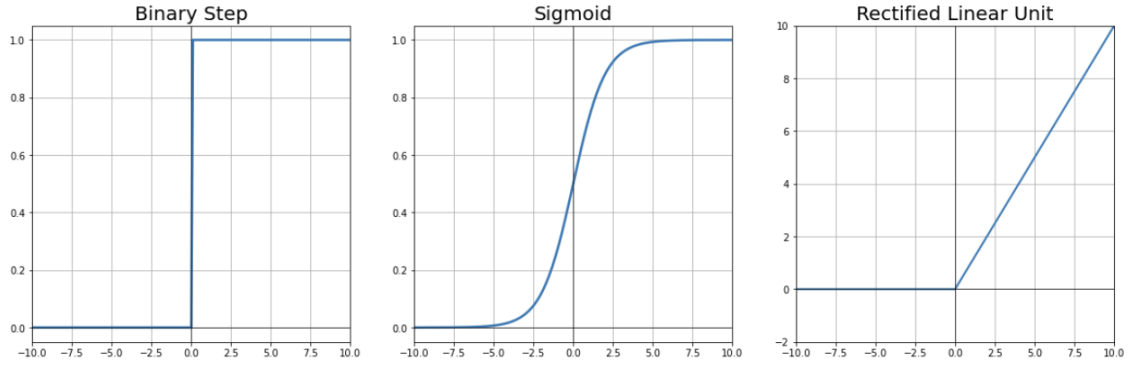

Figure 2.5 Activation functions

Binary step function is commonly used in binary classification problems. It will activate to a constant value once a threshold is reached. It is simple but cannot handle multi-class problems. Binary step function is shown in Figure 2.5 on the left side.

Sigmoid function is an S shaped curve, which was one of the initial and often used activation functions. It transforms the model into a logistic form and fits the input values between 0 and

1. Its setbacks are the complex logistical computation and that the curve is not zero-centered. Sigmoid function is shown in Figure 2.5 in the middle.

Rectifier Linear Unit (ReLU) has become one of the most used activation functions in neural networks during last years. ReLU's value is 0 until it hits the origin, after which it turns into linear curve. It is simple and fast to compute but not perfect.[18, pp. 15–19] ReLU's issues are that it does not differentiate at value 0 and that in some cases it can make weights update in a way that the corresponding neuron will never activate in any data point. [19] Rectified Linear Unit function is introduced in Figure 2.5 on the right.

The learning of the neural network in a supervised manner happens with updating the weights and biases of the network to best suit the outcome required. In the beginning the weights of the network are either initialized randomly or with some predetermined values. A data sample is sent through the network and the network makes a prediction of the correct answer. This process of sending data through the network as shown in the Figure 2.3 and it is called Forward Propagation. Since the learning is supervised, the data sample has a label defining the correct value. Correct value is compared to the one predicted by the network and a loss value is calculated. A loss is a value which determines on how wrong or right the networks prediction was. It can be calculated with different formulas which are called loss functions. One commonly used loss function is mean square error which applies a square operation to the error value. This increases the weight of the error.

After the loss value is calculated the parameters of the network can be adjusted on the basis of the loss function value. This process of updating the weights is called Backward Propagation. The whole process is repeated through the whole training data pool with minimizing the loss as main goal.[20] Minimizing the loss is handled by an optimizer. It is a function which is responsible of minimizing the loss and providing the most accurate results possible for the network. There are several different optimizers with different attributes and calculation times. Some of the commonly used optimizers are Gradient Descent (GD), Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam). [5, p. 70]

These principles shown are the basic building blocks of all neural networks. Additionally there is a definition to a type of neural network whose goal is to learn multiple levels of representation in order to model complex data relationships. These kind of neural networks are called deep neural networks or deep learning. [21] Deep learning is a subset of machine learning which utilizes the use of neural networks. With deep learning it is possible to learn abstractions and hierarchical formations to understand the patterns the input data contains. It makes it possible to

make decisions by identifying and highlighting the characteristics from large datasets. The *deep* in the name comes from the fact that it has more than one state of nonlinear transformation. Nonlinear transformation is a function which transforms features from one representation to another. A neural network is considered deep if it consists of multiple hidden layers, typically more than three. These layers are able to learn different levels of features from the data samples e.g. the first layer learning edges or colours, the second layer learning corners and third layer learning small textures. The layers learn in an unsupervised manner and discover features on their own. The final layer then transforms the task into a supervised task like classification or regression on which the network can make supervised decisions. [5, pp. 8–9]

2.3 Convolutional Neural Networks

Basic operations like classifying and regressing simple data is an easy task for feed-forward neural networks but when the amount and the complexity of data increases the tasks turn very difficult. Convolutional neural networks (CNN) excel with large and complex datasets like images, videos and sound. [10, Ch. 18] They were designed for image recognition tasks and the first applications was the recognition of handwritten numbers. The initial idea of CNN's was to create a network where neurons in the first parts of network are able to isolate local characteristics from dataset. The later part of the network can combine these properties to higher level features and get a more holistic understanding of the data. Common example is face recognition where early part of the network can distinguish simple shapes like lines and curves and the later part of the network can combine these lower level features to facial features like nose or eye. This extraction of features is called feature mapping. [14, p. 145]

Unlike in basic neural networks, CNN's have special layers which make the method very powerful for large data samples like high quality images. These layers are convolutional layer, pooling layer and activation layer. Activation layer is a layer which executes activation function to each neuron it is connected to. ReLU is commonly used activation function in CNN's. [5, pp. 71–72] Activation functions are introduced in Subchapter 2.2.

Convolution is a mathematical operation between two functions which outputs a new function to show how the shape of the first function is modified by the second one. [22, p. 157] If the functions are continuous, convolution can be calculated with integrating the product of the functions when other one is rotated 180 degrees. However images, sound and other data formats are not continuous so the convolution can be calculated with the use of dot product. [10, Ch. 18] Dot product formula is shown in Equation 2.1. Variables a and b are the input matrices and n represents the dimensions.

$$a \cdot b = \sum_{i=1}^{n} a_i b_i \quad (2.1)$$

The targets of convolutional layers is to reduce commonly large data samples to more manageable format without losing too much relevant information. High resolution images can have millions of pixels so traditional neural network operations would consume lots of processing power and time. Figure 2.6 introduces convolutional operation done to a pixel matrix of a monochrome image. Convolutional layers don't have weights as in Figure 2.4 but a weight matrix called filter or kernel. This is presented in Figure 2.6 in the middle. Similarly to weights in traditional neural networks filters are updated and optimized during training process to reduce the loss function's value but instead of single value, the whole matrix of values is updated. [14, p. 148]

In Figure 2.6 each value in the 5x5 input image matrix represents single pixel's grayscale value. The 3x3 filter is applied to each part of the input image and a dot product is calculated and added to the output matrix. After the backpropagation the filter values are updated the same way as described in Subchapter 2.2. Value called stride indicates how the convolved matrix moves on the image. As in the example figure, the stride is 1 so the convolve matrix (marked with blue and yellow) moves 1 pixel at the time. After the first row is calculated, the matrix switches the stride amount of steps down and continues proceeding to the right side again.

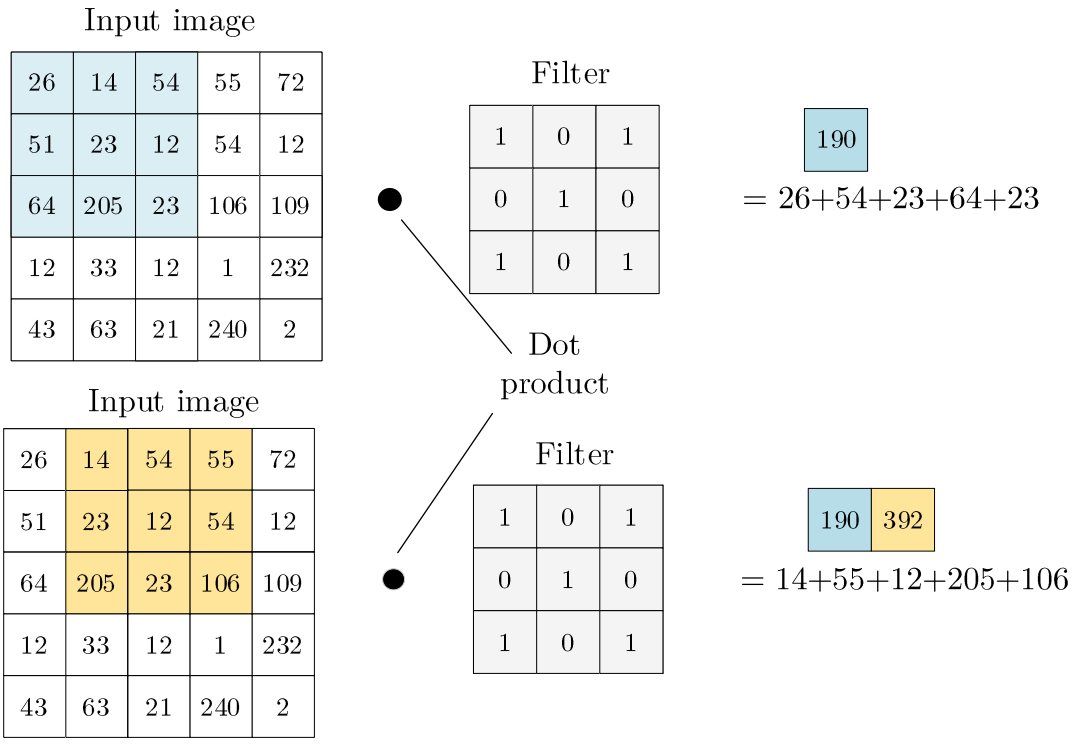

Figure 2.6 Convolution operation with a stride of 1. Figure adapted from ***[5, p. 73]***.

The size of the filter and the stride are both hyperparameters. Hyperparameters are parameters which the affect the networks learning process. They must be chosen by the person running the

learning algorithm. For example with a large stride value the output matrix shrinks rapidly to a small size which might not be optimal for the end result. On the other hand a small stride value grants more precise results but makes the training process more time consuming.

The output matrix's dimensions depends on the size of input image (W), filter (F), stride(S) and zero padding (P). It can be calculated with the Equation 2.2. In the Figure 2.6 the output matrix dimensions would be 3x3.

$$\frac{W-F+2P}{S}+1 \tag{2.2}$$

Occasionally it is necessary to modify the input matrix so that the dimensions of the output matrix is an integer. This is done with the zero padding operation, where units of zeroes is added on the borders of the matrix. Practically this operation does not modify the input value excessively, but it grants the ability to control the size of the output matrix spatially. [23] Zero padding operation is introduced in Figure 2.7 on the left side.

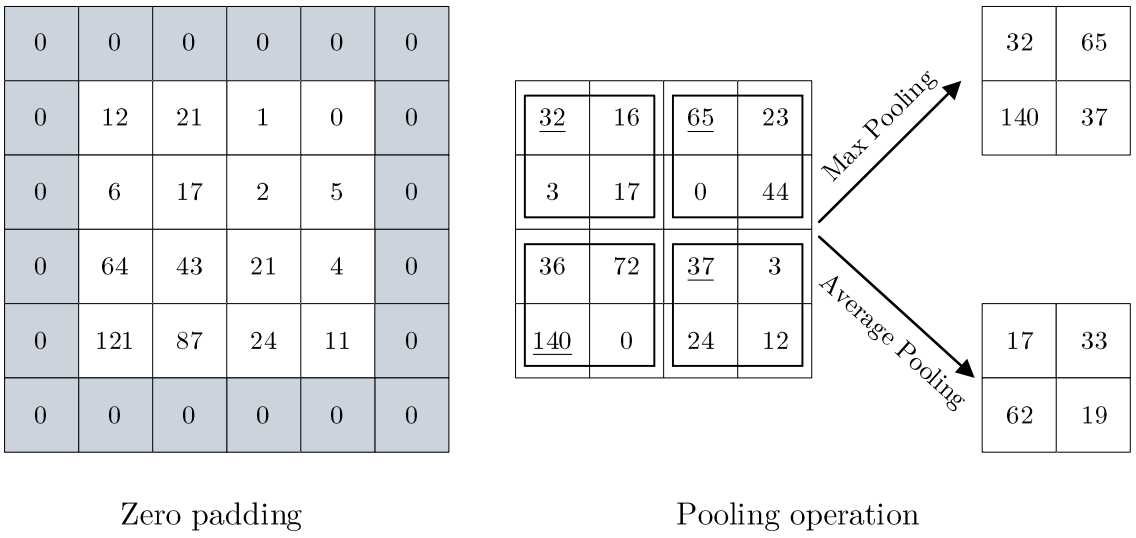

Figure 2.7 Zero padding and pooling operations. Figure adapted from ***[10, Ch. 18]***

Even though convolutional layer shrinks the input matrix, more size reduction is usually required. This is where pooling operation comes handy. The purpose of pooling is to remove noise and keep only the most significant and meaningful features in the data. Similarly to convolutional operation, certain size window scans the input matrix and applies mathematical operation to that part of the matrix. The output value of the pooling operation is shifted to a new matrix. Figure 2.7 introduces max pooling and average pooling operations. In upper part of Figure 2.7 the 2x2 max pooling operation scans the 4x4 input matrix and picks maximum value from each quarter of the matrix. Average pooling functions the same way but takes an average value of the window. [10, Ch. 18]

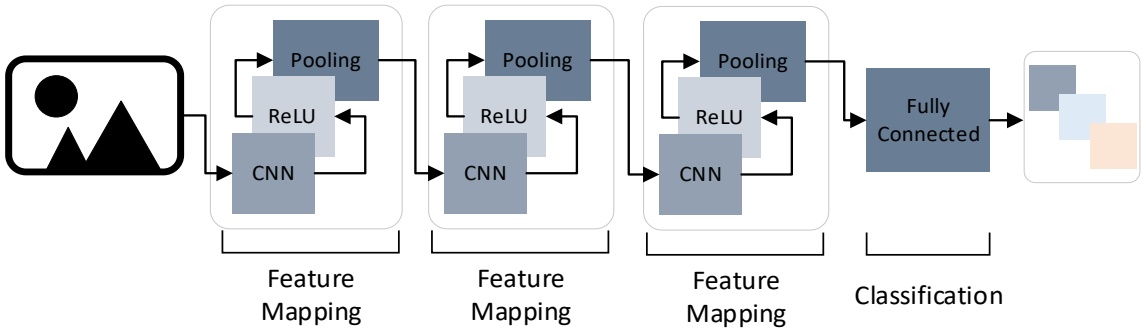

Figure 2.8 Structure of deep convolutional neural network for images. Figure adapted from ***[10, Ch. 18]***

These introduced layers are the building blocks of most common convolutional neural network structures. Figure 2.8 introduces commonly used CNN structure for images. It contains multiple sets of convolutional, ReLU and pooling layers in that exact order. Amount of these sets depend on the image attributes. The result of these layers is fed into a fully-connected layer. This layer is responsible of gathering all the feature mappings from the previous layers and forming a structure to classify the input image. [10, Ch. 18]

2.4 Transformer Neural Networks

Transformer neural network is a method of deep learning that uses self-attention mechanism to weight importance of each element of the input data differently. In time of writing this thesis transformers are quite recent and being used mostly in NLP and computer vision tasks (CV). It was invented to fix the flaws of the methods like recurrent neural networks (RNN) and long short-term memory (LSTM). Transformers were invented by Vaswani et al. at Google in their research paper “Attention is all you need” and this subchapter is highly focused on that. [24]

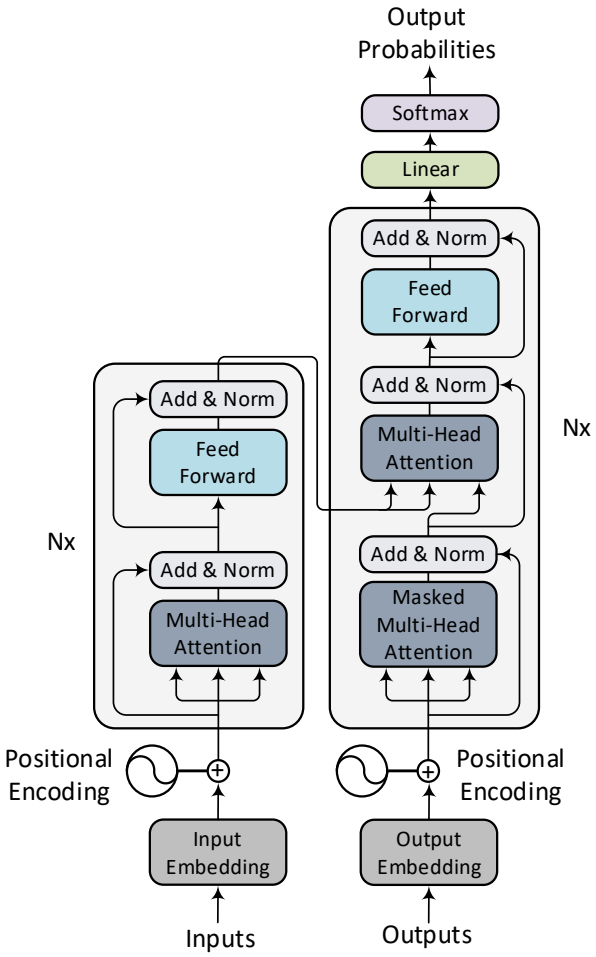

Figure 2.9 Transformer model architecture. Figure adapted from ***[24]***.

Transformer networks are based on encoder-decoder architecture which is introduced in the Figure 2.9. The architecture consists of encoder block, which is introduced on the left side and decoder block, introduced on the right side. In the original research paper Vaswani et al. used N=6 stack of these identical layers on both encoder and decoder.

In the transformer model, input data needs two operations before it is fed into the encoder or decoder stack. Embedding operation arranges similar samples in the input together in the context of the input space. Unlike in CNN's transformer networks do not contain positional data. Positional encoding takes into account the order and position of each element in the input data. This is done with sine and cosine functions with the following equations:

$$PE_{pos,2i} = \sin(pos/1000^{2i/d_{model}}) \tag{2.3}$$

$$PE_{pos,2i+1} = \cos(pos/1000^{2i/d_{model}}).$$

In these equations *pos* is the position of the input data sample inside the whole input data, *d* is the dimension of that sample and *i* represent the indices of each the embedding position dimensions. With the embedding and positional operations the model does not lose crucial information about the structure of the input data. [24]

The encoder block consists of two separate sub-layers called Multi Head Attention layer and feed-forward layer. The feed-forward layer transforms the output of the block to a format the next encoder or decoder block is able to handle. After each sub layer there is an addition and normalization layer which adds the data from before the sub-layer and does a normalization operation to the result. These are called residual connections and are used to reduce the loss of important information within the data flow inside the block. [24] Residual connections also help with the vanishing gradient problem when during the training phase of a feedforward network the error signal shrinks exponentially as a function of the distance from the last layer. [25] The decoder block is similar to encoder block but it additionally contains a Masked Multi-Head Attention block and connections from the encoder block.

Important element of transformer neural network is self-attention. It means how the architecture relates each item in input data to each other item. It produces a table containing information on each item in relation to other items. It is done by mapping a set of key-value pairs and a query to an output. The output is calculated as weighted sum of the values where the weight given to each value is determined by the query's compatibility function with the relevant key. [24] The usage of attention is utilized with a scaled dot-product attention layer operation introduced in Figure 2.10 on the left side. Using multiple attention layers in parallel produces a Multi-Head Attention layer introduced in Figure 2.10 on the right side.

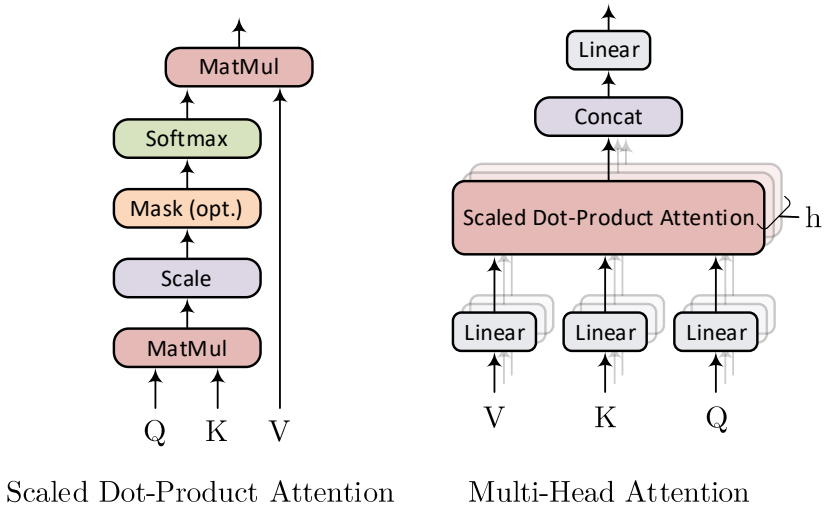

Figure 2.10 Attention layers. Figure adapted from ***[24]***.

The output of the Scaled Dot-Product Attention layer is calculated as:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.4}$$

where Q, K and V are the query, key and value vectors and $\sqrt{d_k}$ represents the dimension of the key vector. [24] With the combination of several attention operations in parallel the transformer is able to gain greater power of discrimination from the data pool. [26]

Vaswani et al. at stated that transformer neural networks outperform convolutional layers and recurrent layers in terms of training time. Their team were able to achieve new state of the art results in NLP related tasks and transformers are currently being tested widely in many fields to replace older methods. [26]

3. POINT CLOUDS IN KEY POINT DETECTION

This chapter begins with a review about point clouds and how they are produced. When working with point clouds it is necessary to take into consideration what technology produces the clouds and that they are usually exposed to noise. Additionally this chapter will review methods and challenges of utilizing point clouds in key point detection using deep learning methods. These methods are chosen by their popularity and accuracy metrics. Six different point cloud neural network architectures will be reviewed in Subchapter 3.2 and in the Subchapter 3.3 four methods are chosen for testing with the point cloud data used in cargo handling operations.

3.1 Point clouds and their acquisition

Point cloud is a representation of 3D object or a space. In its most simple format it is a set of various amount of points each having its individual x, y and z coordinate. Point cloud sizes vary from thousands to millions depending on the accuracy requirement of the application. Point clouds can additionally have RGB color data and intensity information. They have become one of the most used data types for representing 3D models and environments. The success is due to increased availability of devices to produce them as well as growth in areas like augmented reality, autonomous driving and robotics. [27] Compared to other 3D representations of data like depth images, volumetric grids and meshes, point clouds are usually considered the most accurate one due to the representation of original geometric information without discretization.

There are two common technologies to capture point clouds: Light Detection and Ranging (LIDAR) and time-of-flight (ToF) cameras. LIDAR determines the distance and direction of surrounding environment by launching laser pulses in certain directions and measuring the time-of-flight of each pulse. This time can be used to calculate physical distance to the surface. In order to get spatial information the directions and distances of the results can be converted to point clouds. [28] Advantages of using LIDAR to produce point clouds are good accuracy and that it functions well in dusty and foggy setting. Disadvantages are a high cost and that the devices function by rolling a shutter which might make it fragile under use. [29]

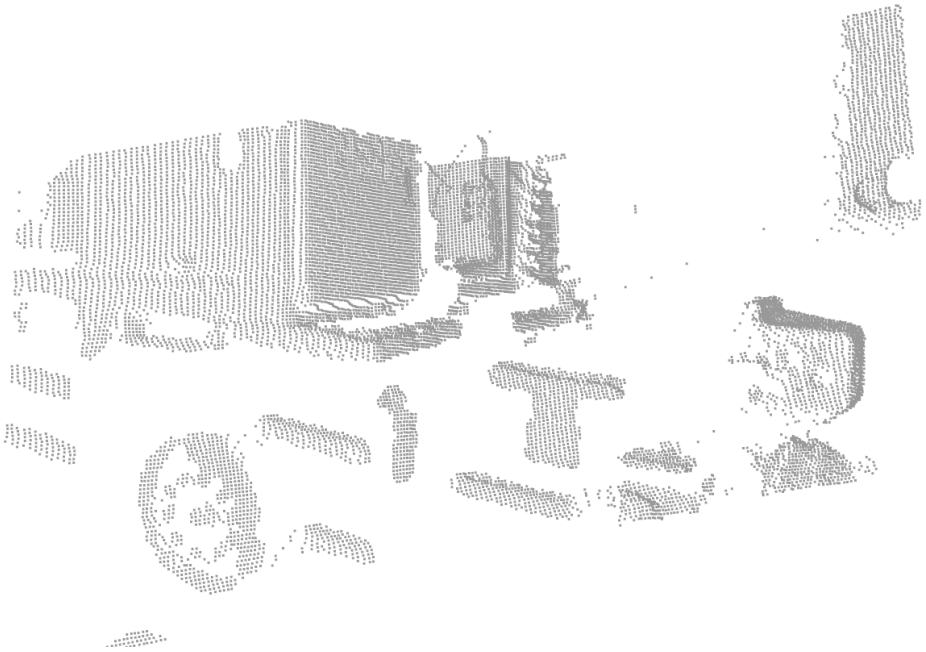

Figure 3.1 2D projection of a point cloud generated by time-of-flight camera

Figure 3.1 introduces a point cloud of a truck cabin created by time-of-flight camera. As a difference to LIDAR technology, ToF-cameras don't use single pulses for measuring the distance but instead a continuous modulated light from LED or laser to illuminate the whole scene in the camera's scope. The distance is calculated from the emission signal's phase change on the receiver side. The phase change of the signal determines the time-of-flight and consequently the distance. These distance valued can be used to generate the 3D point cloud. The advantages of ToF-cameras are a low price, small size and mechanical robustness. The accuracy is not as good as in LIDAR systems and it doesn't work well in dusty and foggy environments. Additionally reflections from e.g. the sun can cause unwanted noise to the cloud. [29]

3.2 Recent deep learning research with point clouds

Due to growth of point cloud using applications, researchers have started to study methods to utilize deep learning on them. Bello et al. in their paper "Review: Deep Learning on 3D Point Clouds" reviewed deep learning methods and challenges on point clouds and discovered several key factors. They noticed that deep learning on point clouds is a challenging task due to the latent nature of point clouds. One particularly common issue with point clouds is noise and outliers. It can be caused by messy scenes and that the point cloud leaves part of the scene blind-sided. This causes unwanted and misaligned points to the scene. Noise is common with all data types and usually with the right type of filtering and data preparation it can be mostly removed. [27]

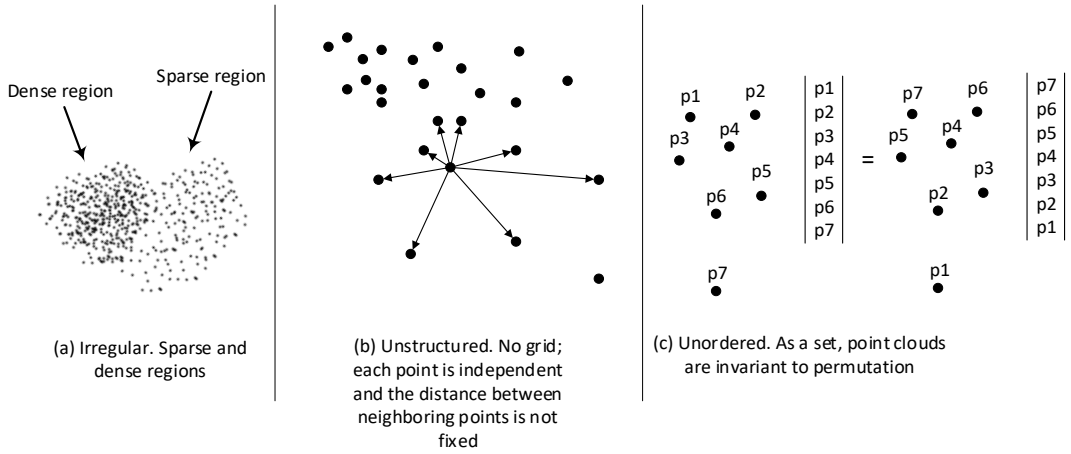

Figure 3.2 Challenges of point cloud data. Figure adapted from ***[27]***

Additionally Bello et al. state that the challenges of point cloud data are that they are naturally irregular, unstructured and unordered. Irregularity, represented in Figure 3.2a means the points are not equally sampled across the sections of a scene. This results some areas having sparse point densities and others having more dense areas. Subsampling techniques can help to reduce irregularity, but cannot totally eradicate it. Point cloud data are also unstructured which means the data are not organized into a regular grid. Each point within the cloud is scanned separately and the distances between them is not constant as seen in the Figure 3.2b. The third challenge Bello et al. state is that the points in point clouds do not have order. The point cloud is commonly a list of coordinates saved in a file and the order they are saved does not affect the scene represented. In other words a point cloud is invariant to permutation. This is shown in Figure 3.2c. [27]

Bello et al. show several methods used to overcome these challenges. For example traditional convolutional neural networks are not effective on raw point clouds due to the challenges mentioned. The convolution operation relies on the data being on a structured grid, regular and ordered, all of which are missing in the point clouds. Researchers have attempted to solve these challenges by transforming point clouds to a structured grid with Structured Grid-Based Learning. It includes voxel-based and multi-view-based methods. Voxel-based methods transform the point into a fixed size volumetric occupancy grid, for which the CNN methods can then be utilized. However, because of the sparse density of the voxels, the memory consumption of them is high. Another approach is multi-view-based methods where the method takes several 2D projections of the point clouds and then apply traditional methods to them. Multi-view-based methods have scored better results than voxel-based due to 2D-methods have been researched more throughout. [27]

Since the creation of PointNet [30] by Qi et al. there have been multiple methods created on raw point clouds. These methods attempt to utilize deep learning despite the challenges mentioned earlier in this subchapter. Many of them have achieved state-of-the-art performance on different benchmark metrics. [27] Six of these raw point cloud methods are introduced in this subchapter.

3.2.1 PointNet++

PointNet++ is a deep hierarchical feature learning method for point clouds created by Qi et al. at Stanford University. It is a bettered version of the group's earlier method called PointNet, which they call the pioneer of deep learning on point sets at the time created. The improvements in PointNet++ pertain to the improved ability to generalize complex scenes and recognize fine-grained patterns. The new structure is a hierarchical neural network method that utilizes the principles of PointNet on a recursive way on a nested partitioning of the input point set. The researchers discovered that point sets are frequently sampled with different densities, which causes neural networks trained on uniform densities to perform poorly. Their experiments show that PointNet++ is able to learning from point set features in an efficient and robust way. [1]

The idea of PointNet++ is to partition the point sets to overlapping local areas determined by the distance metric of the underlying space. It captures geometric structures from small portions of the point set to extract local features. These captured features are then grouped into larger sets and processed in order to form higher level features. This process is repeated in order to get the features of the whole point set. This kind of principle is very similar to those in traditional CNN's. [1]

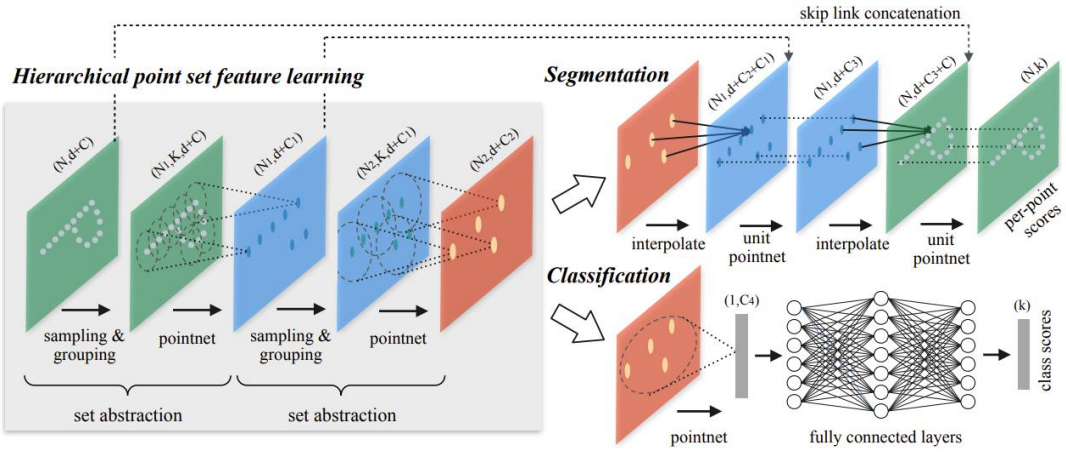

Figure 3.3 PointNet++ hierarchical feature learning architecture. Figure adapted from ***[1]***

Main component of PointNet++ is the hierarchical feature learning architecture introduced in Figure 3.3. It consists of several set of abstraction levels. In each level the point set is handled and abstracted in order to get a new set with less elements. Each of these “set abstraction” layers introduced in Figure 3.3 on the left side is made of three key layers: sampling layer, grouping layer and PointNet layer. Sampling layer chooses a set from the input points, which determines the weights of the local areas. Grouping layer forms local region sets by seeking neighboring points from around the centroids. PointNet layer is used to encode local region patters into feature vectors. The result of the hierarchical point set feature learning block is then fed into a segmentation or classification block whichever the application requires. [1]

3.2.2 Dynamic Graph CNN

Dynamic Graph CNN (DGCNN) is a deep learning neural network architecture for point clouds created by Wang et al. in MIT in USA. This architecture contains a novel module for neural networks called EdgeConv which they state is suitable for CNN-based high-level segmentation and classification tasks on point clouds. EdgeConv is an operation which preserves the permutation variance while capturing local geometrical structure. It produces edges that characterize the connections between a point and its feature neighbors instead of producing point features from their embedding. EdgeConv can group points in Euclidian and semantic space, because it explicitly creates a local graph and learns the edge embeddings. [2]

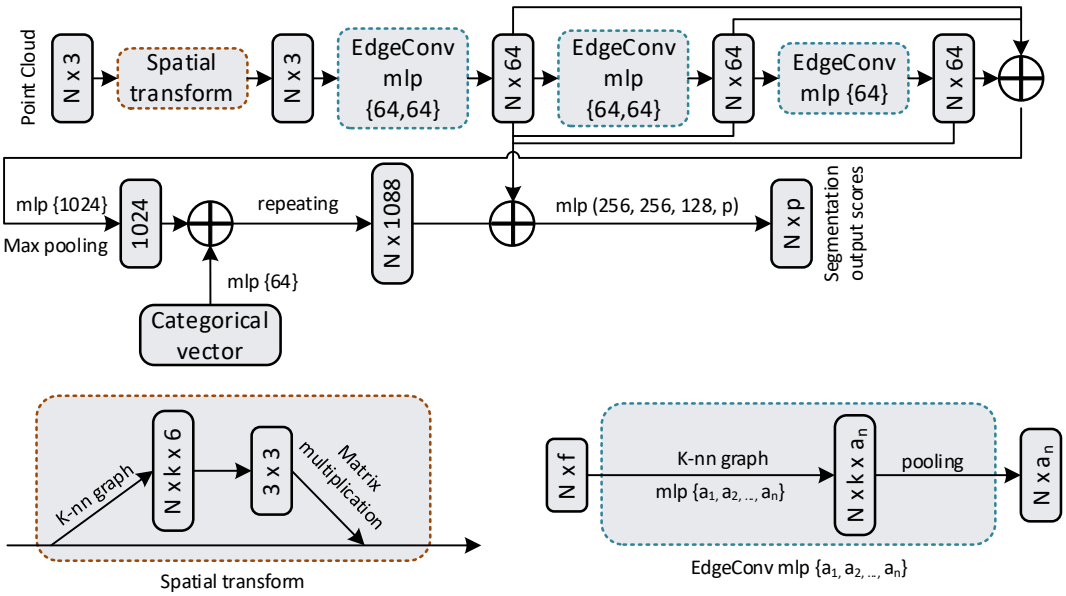

Figure 3.4 Dynamic Graph CNN architecture for segmentation. Figure adapted from ***[2]***

Wang et al. proposed an architecture using their EdgeConv module, introduced in the Figure 3.4. The architecture takes in input of *n* points and does a spatial transform operation shown in the left down corner of the figure. The spatial transform uses an approximated 3x3 matrix to align an input point set to a canonical space. A tensor concatenating each point's coordinates and coordinate differences between its *k* surrounding points is used to estimate the 3x3 matrix. This result is fed to an EdgeConv layer which calculates the features of the edges for each point. This is done by applying a multi-layer perceptron (mlp) with the amount of layer neurons defined as $\{a_1, a_2, \ldots, a_n\}$, and creates a tensor of a shape $n \times a_n$ after pooling among neighboring edge features. EdgeConv layer is presented on the lower right of Figure 3.4. In the segmentation model in Figure 3.4 all EdgeConv outputs are concatenated along with the 1-dimensional global descriptor and an output per-point classification scores is produced. [2]

3.2.3 PointConv

PointConv is convolution operation for 3D point clouds used to build deep neural networks. It is created by Wu et al. at Oregon State University in the US. It extends the basic convolutional operation introduced in Subchapter 2.3 in a way that it can be utilized for 3D point clouds. The team proposed a new approach in which they apply convolution on 3D point clouds with non-uniform sampling. Wu et al. noted that a convolutional operation can be viewed as a discrete approximation of a continuous convolution operator. When applied to 3D space the weights of this operator can be thought as a continuous function of the 3D point coordinates with regard to a reference 3D point in 3D space. MLP can be used to approximate this continuous function, but the downside is these algorithms do not take non-uniform sampling into account. The teams proposed method PointConv takes point cloud coordinates as input and learns an MLP to approximate a weight function. PointConv then applies an inverse density scale to the learnt weights to adjust for non-uniform sampling. [31]

The team state that their PointConv method have also gotten good results in memory efficiency. This is reached by the use a change of summation order strategy to build a implementation of PointConv that allows it to scale up to modern convolutional neural network levels. [31]

Addition to PointConv the team extended it to create another method called PointDeconv. With PointDeconv it is possible to utilize information in coarse layers and transmit it to finer layers. The team states that this achieves good results on segmentation tasks. [31]

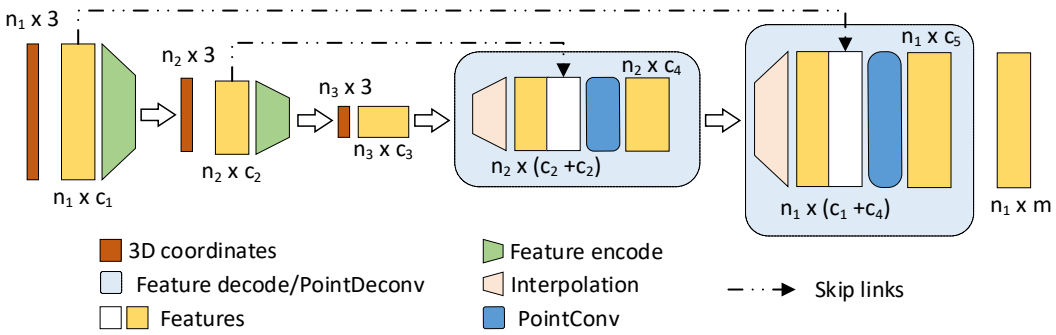

Figure 3.5 Feature encoding and propagation with PointConv. Figure adapted from ***[31]***

Figure 3.5 introduces feature encoding and propagation with PointConv, which can be used for segmentation tasks with 3D point clouds. It is made up of two parts which are interpolation section and PointConv section. To begin, the coarse features from the previous layer are propagated via interpolation layer. The interpolation is achieved by using linear interpolation operation to the features from the three closest points. The result from the operation is then concatenated with features from the convolutional layers of the same resolution using skip links. PointConv operation is applied to these concatenated features to get the final deconvolution output. This process is repeated until all of the features of the input points have been propagated back to the original resolution and the final output scores can be calculated. [31]

3.2.4 KPConv

Kernel Point Convolution or KPConv is a novel design of point convolution created by Thomas et al. in Mines ParisTech academic institution in France. It is inspired by traditional image-based convolution introduced in Subchapter 2.3. Instead of pixels KPConv uses kernel point sets to determine the region where each kernel weight is applied. The kernel weighths are conveyed by the points and their area of effect is determined by a correlation function. Thomas et al. state that this design is flexible due to number of kernel points not being constrained. Their experiments show that KPConv can be used to create very deep architectures for deep learning tasks like classification and segmentation, while maintaining rapid inference and training times. [32]

Figure 3.6 introduces the basic principle of KPConv on 2D points. The idea is similar to the convolution for images show in in Figure 2.6. In the case of images each of the pixel feature vectors are multiplied by a filter matrix. With KPConv input point amount can vary and they aren't aligned with kernel points. Hence each point feature matrix is multiplied with all the kernel filter

matrices and a correlation coefficient is calculated based on its relative location to kernel points. [32]

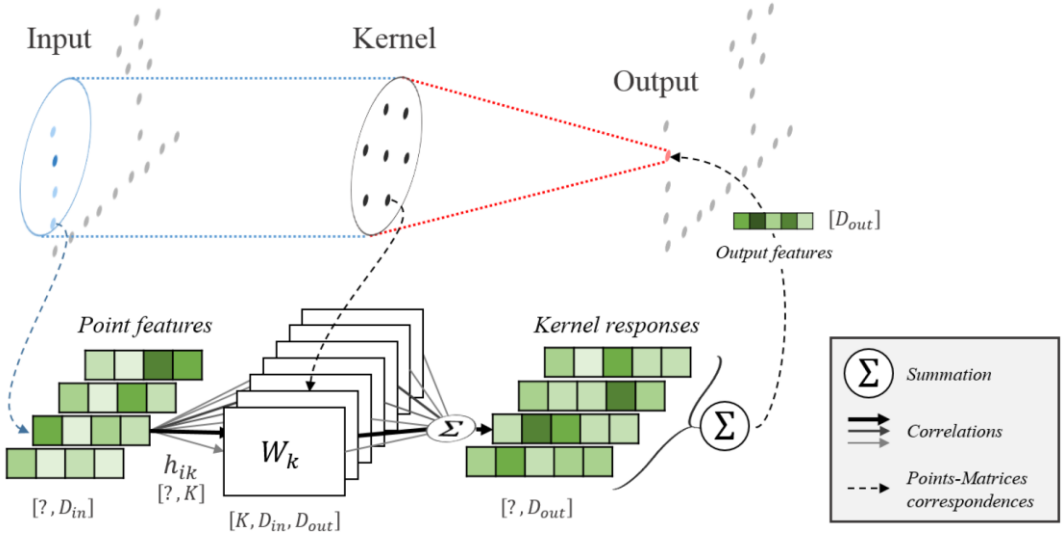

Figure 3.6 KPConv illustration on 2D points. Figure adapted from ***[32]***

Thomas et al. designed two different neural network architectures to handle classification and segmentation tasks on point clouds. The segmentation network called KP-FCNN has an encoder/decoder structure where the encoder has 5-layers each containing two convolutional blocks. The results from the last layer are aggregated by a global average pooling and handled by fully connected and softmax layers. The decoder block utilizes nearest upsampling to get final pointwise features. These features are delivered between intermediate levels of the encoder and the decoder by the use of skip links. The upsampled pointwise features and the features from the skip links are concatenated and then processed by unary convolution to get the final results. [32]

3.2.5 Point Cloud Transformer

Point Cloud Transformer (PCT) is a point cloud learning framework based on the transformer network model introduced in Subchapter 2.4. It was created by Guo et al. at Tsinghua University in Beijing, China. Due to its success in NLP and image processing, the creators chose to apply the method to point clouds. Guo et al. state that transformer is inherently permutation invariant for processing a sequence of points, which makes it applicable for point cloud learning. They improved the basic transformer model by adding support for further point sampling and closest neighbor search to the input embedding. [3]

The idea in PCT is to use inherent order invariance. The advantage of this is to avoid the definition of the order of the points in the cloud and direct the feature learning process through the attention mechanism. [3]

Even though PCT is based on the original NLP transformer model, some adjustments are needed to for it to function well in point cloud application. These adjustments have been gathered in to three different modules: coordinate-based input embedding module, optimized offset-attention module and neighbor embedding module. Coordinate-Based input embedding module is used to generate distinguishable features from the point cloud, since each point has unique coordinates representing its spatial position. Optimized offset attention module is an upgrade to the original self-attention mechanism. Lastly the neighbor embedding module is used to improve point embedding. Attention is powerful in capturing global features from the dataset, but the downside is that it might ignore local geometric information, which is crucial for point cloud learning. [3]

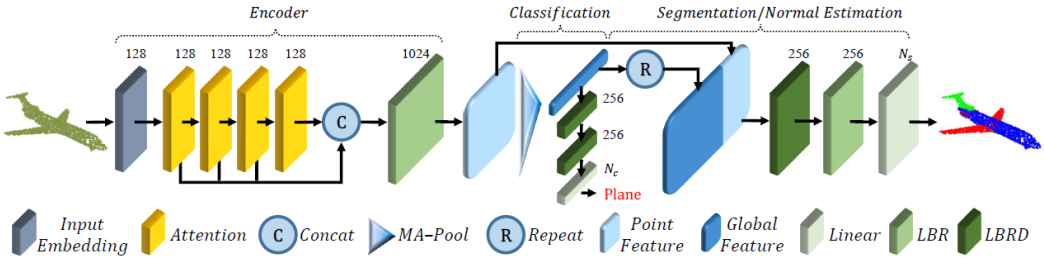

Figure 3.7 Point Cloud Transformer architecture ***[3]***

Figure 3.7 represents the architecture model of the Point Cloud Transformer. The goal of the PCT is to convert input points to a new higher-dimensional feature space, which can determine the semantic affinities between points. Additionally the high-dimensional feature space can serve as a base for other point cloud operations. The input points are embedded into a new feature space using the PCT encoder component. These embedded features are then fed into four stacked attention layers, each of which learns a semantically wide and distinguishable representation of the each point. The output of the attention layers is input to a linear layer generating the final output of the block. This encoder block follows similar principle to the one in original transformer introduced in Subchapter 2.4. The exception is the lack of positional embedding, since each point's coordinate already contain the positional information. [3]

3.2.6 Point Transformer

Point Transformer network is deep learning architecture for point clouds used for variety of 3D understanding tasks created by Zhao et al. It is also the name of the novel deep learning layer

the team created used as an important building block of the architecture. Alongside PCT, Point Transformer is based on the original transformer model introduced in Subchapter 2.4. The team state that transformer based models are well-suited for point cloud tasks due to the self-attention operation. It's invariance to permutation, cardinality of the input elements and inclusive embedding make it a great method for point cloud tasks. Based on these attributes the team constructed an architecture which is entirely based on pointwise operations and self-attention. [33]

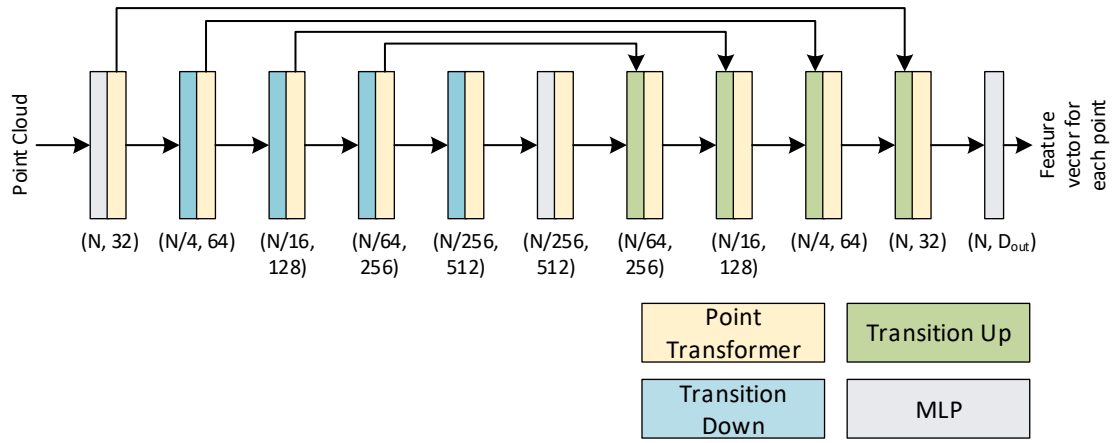

Figure 3.8 Point transformer network for semantic segmentation. Figure adapted from ***[38]***

Figure 3.8 introduces Point transformer network for semantic segmentation. The backbone structure of the architecture consists of feature encoding and feature decoding. Each of which have five stages that progressively downsample and then upsample the point sets. The sampling rates on this network are [1, 4, 4, 4, 4] and their corresponding cardinalities can be seen from Figure 3.8 under each block. Transition modules presented with blue (down) and green (up) can be seen in Figure 3.8 in the lower right corner. They are used to connect the consecutive stages. Transition down block is used to tune down the cardinality. Transition up block is used to tune up the cardinality and to map features from the downsampled input point set. [33]

The primary feature aggregation operator on the network is the Point Transformer block presented with tan color. It includes a self-attention layer, residual connection and linear projections for dimensionality reduction. Input for the Point Transformer block is a set of feature vectors along with 3D coordinates. As an output the block facilitates information interchange between these feature vectors. It is done by providing new feature vectors for each data point. The final decoder block constructs a feature vector for each point as output. It uses a MLP-layer to create mapping for the final result logits for each point in the input set. [33]

3.3 Selected methods

All of the researched methods state that their approach can reach state-of-the-art results on 3D point clouds tasks like segmentation and classification. None of the researched methods is directly usable for the application of this thesis, but segmentation task is rather close and the models can be usually modified to our needs. Hence the methods which can function fairly well on segmentation tasks were appealing to be chosen for testing.

The research question of this thesis is about using transformer and convolutional neural network so it was necessary to research somewhat equal amount of methods from each category. Since only two viable options for point clouds were found, both *Point Transformer* and *Point Cloud Transformer* are chosen for testing.

On the methods with CNN's many viable options were found but not all of them turned out to be usable for this problem. Initial tests revealed that some methods don't function at all with the data tested with or the training scripts provided were not suitable for the environment used. *PointNet++* and *Dynamic Graph CNN* were chosen to be analyzed due to their large amount of training scripts available and somewhat usable results on the initial testing.

4. TRAINING AND TESTING SETUPS

This chapter will go through what new code is written, what kind of testing setup is used and how the models are modified, trained and tested. Subchapter 4.1 will introduce the hardware and software environments used. The data used to train models consists of raw point clouds and labels which are then combined into a single file. This is explained in Subchapter 4.2.

Subchapter 4.3 will go through how the training script is put together and what steps are involved in the training process. Subchapters 4.4 and 4.5 will explain the basic functionality of the inference testing and the accuracy testing scripts. The chapter ends with a test plan, where all the different test variations are introduced.

4.1 Environments

Models trained in this thesis will be tested on three different hardware environments called Training, Edge IPC and Edge Xavier and they will be called with these names for the rest of this thesis.

Table 4.1 Hardware environments

Environment	GPU	CPU	RAM	Operating System
Training	Nvidia GeForce RTX 3090	AMD Ryzen 9 5900X 12-Core	32GB	Ubuntu 20.04.3 LTS
Edge IPC	No GPU	Intel Core i3-4010U 2-core	8 GB	Ubuntu 20.04.2 LTS
Edge Xavier	Nvidia Volta 384-core	NVIDIA Carmel ARM 6-core	8 GB	Ubuntu 18.04.6 LTS

Table 4.1 introduces the hardware specifications of each of the environments. Training environment is as the name suggest an environment where all the training will happen. Fast training process requires a powerful GPU and the data and model architecture will determine how much GPU memory is required. Large point clouds contain a lot of data which means the GPU memory requirement for the model will be high.

Edge IPC environment is the environment where the models will be utilized in the end user application mentioned in the introduction of this thesis. This environment does not contain a

CUDA-supported GPU, which means the model inference has to be run on a CPU. This will most likely cause longer inference time since the parallel computing utility of a GPU cannot be utilized.

Edge Xavier environment is Nvidia's Jetson Xavier NX Developer Kit. It is a compact computer with GPU computing power designed to handle AI tasks in edge environments. Xavier is tested to see how a small form factor AI computer can handle the large point clouds models. Xavier board contains a CUDA-supported GPU.

Neural networks used in this thesis use PyTorch as base software framework for deep learning. PyTorch is a deep neural network programming Python package for GPU-accelerated learning developed by Facebook AI team. It is based on a library called Torch which was originally built in C language. PyTorch contains libraries for tensor manipulation on CPU and GPU. It has built-in neural network libraries, tools for model training and a multiprocessing library which is able to utilize shared memory. [34]

4.2 Data preprocessing

The dataset used to train and test the models is 959 raw point clouds captured from real-life operations. Each of them contains a scene where a 40 foot container is hovering over a truck chassis. In real life operation the application would need to operate several sizes of containers, but due to lack of sufficient training data from those scenarios this thesis only concentrates on point clouds captured from 40ft scenes. Figure 4.1 introduces the scenery of a twist lock chassis and a container in a real life image.

Figure 4.1 Scene of container placing on truck chassis with twist locks

The time-of-flight camera used to capture the point clouds has a maximum amount of 23232 points, however the actual amount varies a lot depending on the scenery captured. This means the point amount of the raw point cloud file is not constant. Deep learning models are commonly trained with a constant data sample size which means the raw point clouds point amount need to be fixed. This raised a concern on what the point amount should be. With some analyzing of the raw point clouds it was noticed that the point amount varies greatly anywhere from 3000 points to 20000 points. Lower amount of points would benefit fast computing but it would risk

losing relevant features from the point cloud. However too large point amount would make the model substantially slower and increase the memory requirement. Point amounts chosen for testing are shown in Subchapter 4.6.

Figure 4.2 Scene of container placing on truck chassis with a gooseneck

These point clouds are annotated with a specific labelling tool. Annotations are made manually from 2D intensity images of the captured point clouds. Each point cloud can be annotated with a maximum of two of the classes shown in the Table 4.2. The scenes can have a class 0 label and either class 1 or class 2 label. Class 1 is a scene where the truck chassis has twist locks in it. This is shown in Figure 4.1. Class 2 is a scene where a gooseneck truck chassis is present in the scene. A real life image of this scene is presented in Figure 4.2. Additionally single class label is possible if the scene does not have anything else in it. The labels are saved in separate xml-file with entries containing point cloud file name and 2-dimensional coordinates for each label.

Table 4.2 Label classes

Class 0	Container corner
Class 1	Twistlock
Class 2	Gooseneck

In order to get the labels and raw point cloud data files to a feasible format for the deep learning algorithms, they will be combined into a single file using a script. This script takes as parameters

the directory where the raw point clouds are located as well as the location of the xml-file containing the labels in x and y format. Additionally it is given annotation radius, point number and parameters about the ToF-camera setup so that the same script can be used with different equipment.

This script attempts to remove unwanted and unnecessary objects from the scene. These include the truck cabin and concrete blocks, which are both common in a terminal scene. A similar process is used in the real-life application. To achieve this a method called furthest point sampling is used. This method attempts to resample the point cloud by choosing n-number of points in a way that they are as far as each other as possible. It attempts to keep the point cloud as original as possible, while removing unnecessary points. The goal is to maintain the major features while removing the small unneeded details.

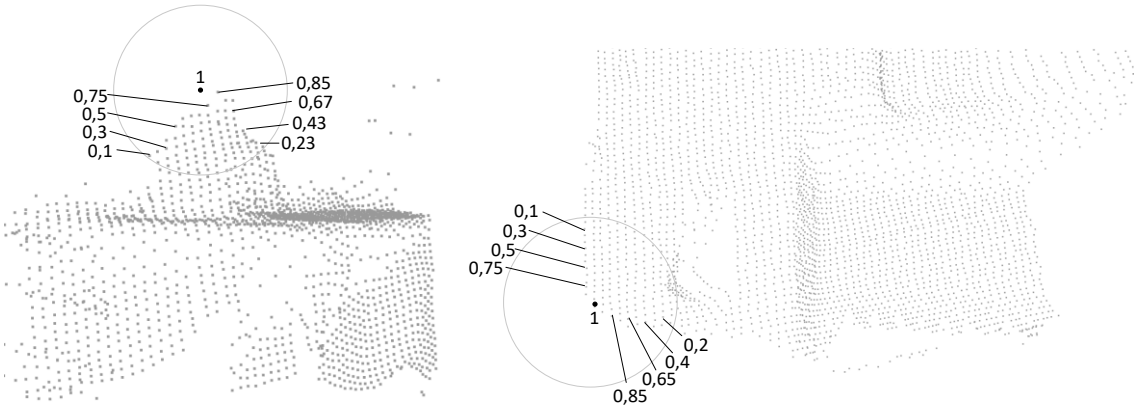

Figure 4.3 Twistlock (left) and container corner (right) key point label probabilities

Figure 4.3 illustrates the end result from the data preparation script. As mentioned before, the script takes a value called annotation radius as a parameter. This is the radius of the illustrated circle in the figure. In reality it is a 3D ball covering the whole area but for illustration purposes it is shown as a circle. Every point inside this ball is considered as the label, with each having its own probability to be the exact key point. The point chosen in the labelling phase will be the point with the probability of one and each nearby point inside the radius will be given a Gaussian probability depending on the distance from the actual label. The points near the center will have higher probability while the ones near the circumference will have probabilities approaching zero. This kind of probability-based approach is done so that the machine learning algorithm is given more options from where to learn. In the real life application the goal is to find with some approximation where the key point is. It is not required to get exactly the one in the corner, the ones nearby it are also valid options.

The whole data pool will be split to an 80/20 ratio for training and testing set. This is done with a Python script which will take the raw data folder, split ratio and target test and train folders as parameters. The script will calculate the correct amount of point clouds and will copy a randomized set to train and test sets.

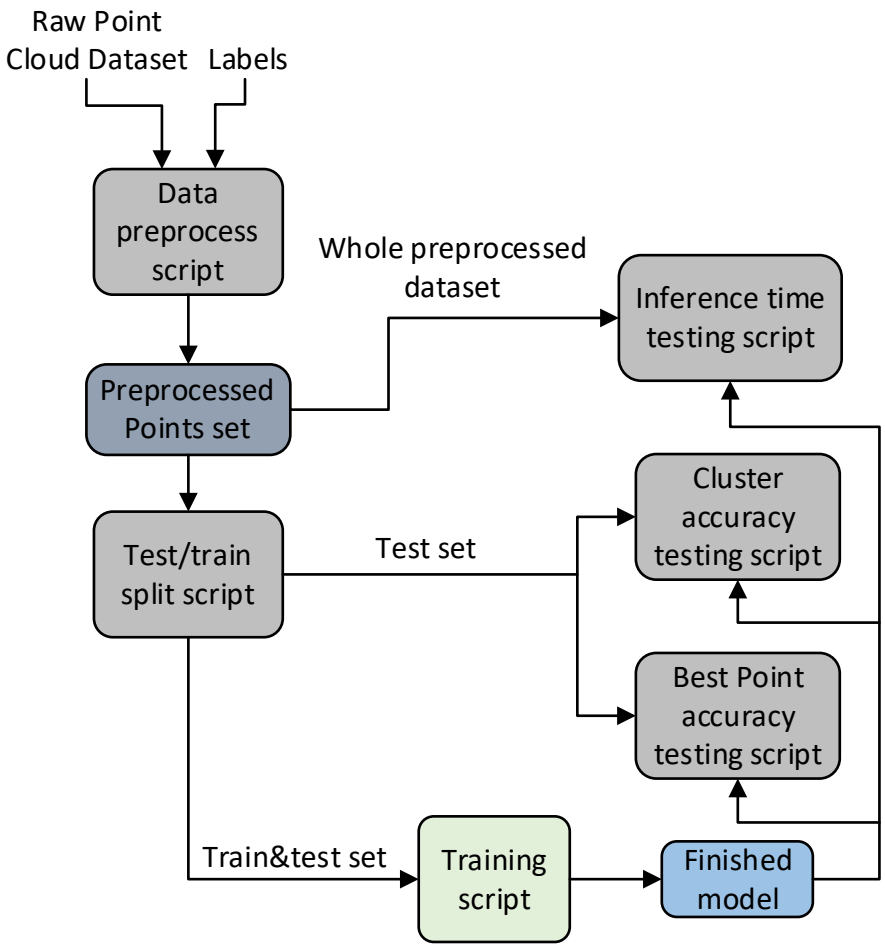

Figure 4.4 Data flow diagram

Figure 4.4 shows what data each script is using. The preprocessed data are created from the raw point cloud dataset and the labels. The outcome will be a preprocessed dataset which is then used directly for inference testing as a whole. This whole dataset is also divided into test and train sets which will be used by the training script and testing scripts.

4.3 Training and model adjustments

The training script will take as parameters the training and testing data set folders, model type used, epoch amount, batch size and logging directory. The script starts by initializing the given parameters to variables and creating a structure to a folder where all training related files will

be placed. After this the script creates a logger to log training related information to a text-file. Training set and testing set are loaded and then used to create train loader and test loader objects. This will help to optimize memory usage and batch sizing during training. The training script will load the model type from the separate model file based on the given model type parameter. The script is able to continue from previous training so if it finds existing weight models from the logging directory, it will initialize those as starting points. In case there is no existing model, the script will initialize the weights with the starting values. After this the optimizer will be initialized and assigned to the model.

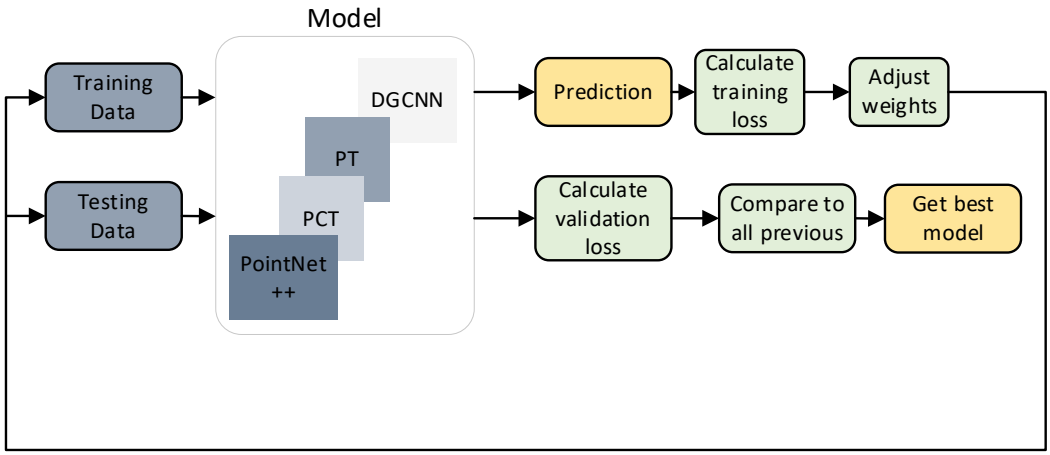

Figure 4.5 Training loop diagram

Figure 4.5 introduces the training loop diagram. The loop will run as many epochs as determined in the parameters. On each epoch a batch will be taken from the training set and fed to the model. After the model has done its prediction, training loss is calculated and weights are adjusted accordingly. This principle is explained more thoroughly in Subchapter 2.2.

Additionally a validation loss is calculated for each epoch. Data from the test set is fed to the model to see how it functions with data it hasn't used for adjusting weights. With analyzing validation loss it is possible to see how the predictions develop. The script will automatically save the version of the model which has the best validation loss value. It is done by saving the validation loss value from each epoch to a list and doing a comparison to find if the new validation loss is less than all others in the list. Additionally the script will save the model every 20 epochs to make sure the progress is saved in case of an unexpected halt of the training.

Each neural network code is downloaded from an open source repository and modified to fit the needs of our own data. For PointNet++, [35] an implementation for part segmentation is used. The modifications include cleaning the code and simplifying the structure for our own needs. This architecture is able to utilize the data straight and there was no need to modify the input

or output tensors. For Point Cloud Transformer and Point Transformer [36] the same repository was used. For PCT only a classification network code was available so it was necessary to build the network based on the PCT architecture for segmentation. From the open source repository the module for attention mechanism was utilized. For Point transformer the code for part segmentation was utilized and only the size of input and output tensors were modified. DGCNN code for segmentation was taken from [37]. In also this network structure the size of input and output tensors were modified to fit our data. Additionally the embedding dimensions were adjusted to 16384. This neural network architecture utilizes a method called k-nearest neighbors so the k-value was changed from 20 to 40 to fit better our dataset.

4.4 Accuracy testing

With some initial analysis of the model's output tensors it was noticed that some models output large amounts of points as predictions. It means the models are suggesting a lot more points as key points as is necessary. This is not automatically a bad thing, but it creates a concern on how to test the models accuracy properly. If the model outputs large probabilities on points near the actual annotation, but does a single miss prediction with the highest probability on a point which is far away from the actual label. If in case the model is tested with single best point accuracy method it will lead to bad result even though the model predicted rather well on the points near the actual label. For this reason two different approaches are taken to measure accuracy: cluster accuracy and best point accuracy.

Cluster-based accuracy testing script takes as parameters the method used, the weights file, the point cloud files to be tested with, the directory where the model files are located in and whether to use CPU or GPU for running the tests. The script starts by initializing the testing dataset to a dataset class object for convenient data handling. The model weights are loaded from the file provided and it is saved to the device chosen. Testing thresholds are initialized and the test loop is started. The test loop loads the data from the dataset class one point cloud and label at a time. The prediction is done with the model weights and the result along with the point cloud is sent to a clustering function in order to get the keypoints.

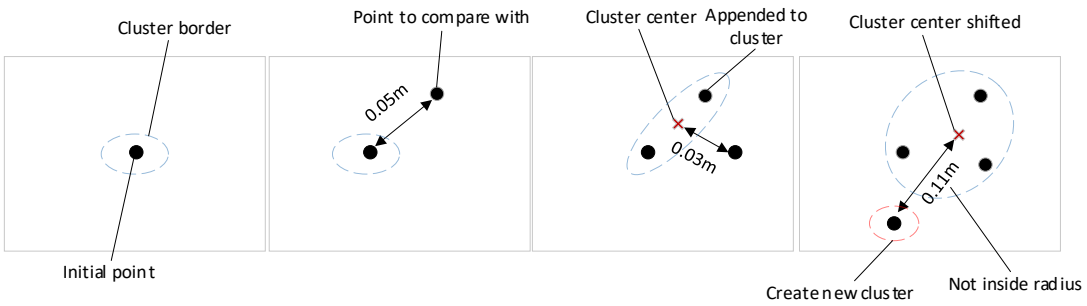

Figure 4.6 Clustering for accuracy testing illustrated

The clustering function picks each point with a probability higher than 0.3 to be clustered. Figure 4.6 visualizes the principle of the function. The clustering radius is set to be 0.1 meters. The function takes each prediction point one by one and creates a python class object called Group for them which corresponds to the cluster in the figure. Then each consequent point is checked if it is located inside the 0.1m radius of some existing cluster center point. If this is true, the point is added to that cluster and a new center point is calculated since the new point will shift the center point. If the point is not within this radius a new cluster group is created and the point is appended there. This loop continues until all prediction points are clustered. It returns a list of each clusters center points which represent the key points to where the labels are compared.

The comparison of the results is done with a commonly used tool for evaluation called confusion matrix introduced in Subchapter 2.1.1. Recall and precision values are calculated from the results and compared to three different thresholds. The cluster accuracy testing script outputs precision and recall values for each class with each different thresholds. The result output will be nine different values in total.

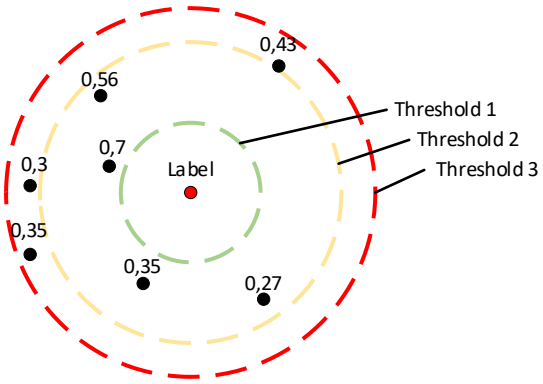

Figure 4.7 Best point accuracy testing with thresholds

The second method used to evaluate accuracy is best point accuracy testing visualized in Figure 4.7. It is a separate script which takes in all the same parameters as the cluster-based testing. It does the similar initializations and the testing loop functions the same way. The difference is that accuracy is calculated from only a single point. After the prediction the script finds the point with the highest probability of each class and compares it straight to the labeled point with a probability of 1. The distance between the prediction and the label of the specific class is compared and a threshold is chosen accordingly. The thresholds are the same as used in the clustering method. After the whole testing dataset is evaluated the total percentage of the predictions accuracy on thresholds will be calculated and outputted.

4.5 Inference testing

Inference time testing will be done on each model with a separate script. This script will take as parameters the model name, model file, and the whole data pool directory. Additionally it can be given a parameter to choose if CPU or GPU will be used for the test.

The script will load the whole dataset to a Python dataset object for easy data usage. After the script will initialize the model from the file and load it to the correct device. The testing loop will iterate over the whole data pool and on each item it will do a prediction. The prediction is not used on anything since we are only interested in the time it took to make it. Python timer will start before the prediction and finish after it is done. Each of the prediction times will be saved to a Python list and after the whole data pool is iterated an average of all the prediction times will be calculated.

4.6 Test plan

The goal of the test plan is to find out how point amount affect model accuracy and inference time. As discussed in Subchapter 4.2 the point amount varies a lot within the point clouds. To get a good understanding of its effects two point cloud amounts were chosen: 8192 points and 16384 points. The reasoning for the selections was the fact that with the larger point amount most of the point clouds features are preserved and the model can learn from nearly the original point cloud. The smaller point amount was a natural choice since it’s half the size of the larger one, which means some point clouds features will become less accurate. Each of the methods will be trained for 600 epochs and the weights-model which has the lowest evaluation loss will be selected. The number of epochs needed for diverging for each method will be revealed in Chapter 5.

To be able to compare the new point cloud methods to the old intensity image based method, it is necessary to train the intensity-based models with the same dataset as the point clouds. The point clouds are first converted to png-format image files and the models are trained with the images.

Inference time tests are introduced in Table 4.3. Each model will be tested with the whole dataset containing 959 point clouds each with two point amounts. These tests will be ran in each of the environments introduced in Subchapter 4.1.

Table 4.3 Inference time tests

Model	Point Amount	Test type	Test data	Data amount	Notes
DGCNN	8192	Inference time	8192 Points set	959	Will be tested in all environments
DGCNN	16384	Inference time	16384 Points set	959	Will be tested in all environments
PointNet++	8192	Inference time	8192 Points set	959	Will be tested in all environments
PointNet++	16384	Inference time	16384 Points set	959	Will be tested in all environments
Point Transformer	8192	Inference time	8192 Points set	959	Will be tested in all environments
Point Transformer	16384	Inference time	16384 Points set	959	Will be tested in all environments
Point Cloud Transformer	8192	Inference time	8192 Points set	959	Will be tested in all environments
Point Cloud Transformer	16384	Inference time	16384 Points set	959	Will be tested in all environments
2D Intensity Method		Inference time	2D Intensity images created from point clouds	959	Will be tested in all environments

Cluster accuracy and best point accuracy tests are presented in Table 4.4. Three weight-models of each method will be trained with three different train/test splits and each will be tested with

the corresponding test set. Average results of these three tests will be presented in Chapter 5. The thresholds used for evaluation will be 0.01m, 0.05m and 0.09m. The 2D Intensity method with a similar principle.

Table 4.4 Cluster and best point accuracy tests

Model	Point Amount	Test type	Test data	Data amount	Notes
DGCNN	8192	Cluster/ best point accuracy	8192 Points test sets 1,2,3	192	Average result of 3 test sets will be presented
DGCNN	16384	Cluster/ best point accuracy	16384 Points test sets 1,2,3	192	Average result of 3 test sets will be presented
PointNet++	8192	Cluster/ best point accuracy	8192 Points test sets 1,2,3	192	Average result of 3 test sets will be presented
PointNet++	16384	Cluster/ best point accuracy	16384 Points test sets 1,2,3	192	Average result of 3 test sets will be presented
Point Cloud Transformer	8192	Cluster/ best point accuracy	8192 Points test sets 1,2,3	192	Average result of 3 test sets will be presented
Point Cloud Transformer	16384	Cluster/ best point accuracy	16384 Points test sets 1,2,3	192	Average result of 3 test sets will be presented
Point Transformer	8192	Cluster/ best point accuracy	8192 Points test sets 1,2,3	192	Average result of 3 test sets will be presented
Point Transformer	16384	Cluster/ best point accuracy	16384 Points test sets 1,2,3	192	Average result of 3 test sets will be presented
2D Intensity Method		Cluster/ best point accuracy	2D Intensity image test sets 1,2,3	192	Average result of 3 test sets will be presented

5. OBSERVED COMPUTATION TIMES AND ACCURACIES

This chapter will show the results from the tests introduced in Subchapter 4.6. These results will be compared to the currently used 2D intensity image based method. There will also be discussion about how the point amount affects the overall accuracy and the inference time of the model. This chapter will begin with the introduction of the inference time test results in the Subchapter 5.1. The results from the cluster accuracy will be introduced each method at a time in the Subchapter 5.2. The chapter will conclude with the results from best point accuracy testing in the Subchapter 5.3.

5.1 Inference time results

Inference time results from testing are presented in Table 5.1. When the inference time testing were performed in Edge Xavier environment it was noticed that not all models were able to run inference in that environment. PointNet++ models had in issue were some libraries required by the neural network were not supported in Edge Xavier's Ubuntu environment. Due to this reason PointNet++ inference time could not be tested in Edge Xavier environment.

Table 5.1 Inference time results

	Edge Xavier	**Edge IPC**	**Training**
DGCNN 8k	592.9 ms	10477.5 ms	46 ms
DGCNN 16k	2053.3* ms	>100 s	161.6 ms
PointNet++ 8k	Not supported	1951.1 ms	137.1 ms
PointNet++ 16k	Not supported	3003 ms	153.8 ms
Point Cloud Transformers 8k	331.8 ms	4131.7 ms	17 ms
Point Cloud Transformers 16k	1203.3 ms	>100 s	59.1 ms
Point Transformer 8k	No sufficient VRAM	>100 s	745.5 ms
Point Transformer 16k	No sufficient VRAM	>100 s	1298.3 ms
2D Intensity Method	408.4 ms	353.1 ms	44.9 ms

**FP16 accuracy used for inference to fit the model into Xavier RAM*

Another issue faced in the Edge Xavier environment was the lack of GPU memory. This caused problems when testing inference time on Point transformers method. The method requires more GPU memory than is available which meant inference time tests couldn't be performed in Edge Xavier environment. Additionally DGCNN 16k method was required to run in FP16 mode to be able to fit the model into the GPU memory. FP16 means the model inference is ran with using 16-bit weights instead of 32-bit weights. Using the FP16 results in slightly lower accuracy but at the same time it reduces the memory requirement of the model. In this case it was done to see approximation inference time for DGCNN method.

DGCNN and PCT methods had both usable inference times in Edge Xavier environment. PCT with 8192 points was even able to outperform the 2D intensity method by approximately 75 milliseconds. DGCNN and PCT methods with 16k points were under three seconds so both can be considered usable in real life operation. It was also noticed that when comparing the 8k version to 16k version the inference time approximately quadrupled in both PCT and DGCNN.

The results from Edge IPC immediately reveal the power of the GPU when handling complex point cloud neural network models. Both Point transformers models along with the PCT 16k and DGCNN 16k models got over 100 seconds as inference time in this environment. This amount can be considered unusable since it would slow down the real life operation dramatically. Apart from PointNet++ all methods have the inference time increasing over tenfold when doubling the point amount from 8k to 16k. Both PointNet++ methods along with PCT with 8192 points were able to perform decently in this environment with the result of few seconds. However none of the methods comes to even close to 2D Intensity image method. It is interesting to see that the 2D method also performs better in CPU only environment than in the Edge Xavier environment.

Each method performs well in the training environment and would be considered usable in the real life operation. Even the highest inference time from Point transformers with 16k points got a little bit over one second as a result. When comparing the 8k results to the 16k results overall no clear pattern can be found. In all cases the inference time increased in 16k points which was to be expected. In PointNet++ the inference time increased only 20ms when going from 8k to 16k. However no consistency can be seen in the results when comparing transformer-based methods to CNN-based methods. When comparing the point cloud methods to the 2D intensity image method DGCNN with 8k points gets almost the same result. PCT with 8k points got a lot faster result and with 16k the result exceeds the 2D method with only 20ms. Other methods got much higher results when compared to 2D method but as said, all of the results would be usable in the real life-operation.

5.2 Cluster accuracy results

Results from the cluster accuracy testing can be found in the tables of this subchapter. Three different models were trained from three different splits and the average of those three results are shown in each table. A similar principle is used in all of the accuracy tests in this subchapter and Subchapter 5.3.

Table 5.2 2D Intensity image method cluster accuracy results

Class 0	**Precision**	**Recall**
Threshold 0.01	0.629	0.520
Threshold 0.05	0.908	0.743
Threshold 0.09	0.928	0.754
Class 1		
Threshold 0.01	0.567	0.555
Threshold 0.05	0.961	0.946
Threshold 0.09	0.970	0.953
Class 2		
Threshold 0.01	0.338	0.367
Threshold 0.05	0.875	0.945
Threshold 0.09	0.896	0.962

Table 5.2 introduces the results from the 2D Intensity image method. Results show that the intensity image method in 2nd and 3rd thresholds has high results of around 0.9 in recall and precision across all classes except class 0. This means the models functionality on class 0 is different and lacks the ability to find the key points in quarter of the cases in this class. Also the 1st threshold across all the classes is high which means the model is able to do the predictions very accurately.

Table 5.3 introduces the results from DGCNN methods 8k and 16k cluster accuracy tests. The 8k version started to diverge after 350 epochs and only minor improvements after that were made. The 16k version required around 300 epochs to diverge. The training for the 8k version with 600 epochs lasted for 14 hours and for the 16k version it took 35 hours.

When compared between the two point amounts it can be seen that the 16k precision and recall is higher in each threshold and in each class which was to be expected due to higher amounts of information in the point cloud. When comparing the DGCNN 16k version to the 2D intensity

image method the 1st threshold is better in each class in the 2D intensity image method. However the recall of the 2nd and 3rd are better in DGCNN 16k and while precision is being rather the same for both. Higher recall means the model is more accurate on the selection of points it has predicted. DGCNN with 16k does get better overall results and is more consistent across all classes.

Table 5.3 DGCNN cluster accuracy results

Point amount	8192		16384	
Class 0	**Precision**	**Recall**	**Precision**	**Recall**
Threshold 0.01	0.284	0.365	0.443	0.469
Threshold 0.05	0.725	0.927	0.939	0.993
Threshold 0.09	0.742	0.941	0.943	0.996
Class 1				
Threshold 0.01	0.184	0.215	0.375	0.392
Threshold 0.05	0.679	0.796	0.925	0.969
Threshold 0.09	0.691	0.810	0.934	0.978
Class 2				
Threshold 0.01	0.154	0.146	0.207	0.232
Threshold 0.05	0.754	0.712	0.878	0.983
Threshold 0.09	0.792	0.746	0.889	0.994

Table 5.4 introduces the results from PointNet++ 8k and 16k cluster accuracy tests. Both versions diverged on approximately 300 epochs. The training for the 8k version with 600 epochs lasted for 13 hours and for the 16k version it took 20 hours. As seen from the table the precision is low across the whole table. Low precision indicates that the model predicts almost all of the points as key points and hence would not be usable in real life application. It is necessary to review how the model performs in the best point accuracy tests to see does it give higher probabilities to the key points near the label point. In terms of recall the method performs decently and is able to achieve higher results in cluster accuracy tests. The difference between the two point amounts is not very different.

Table 5.4 PointNet++ cluster accuracy results

Point amount	8192		16384	
Class 0	**Precision**	**Recall**	**Precision**	**Recall**
Threshold 0.01	0.003	0.057	0.012	0.077
Threshold 0.05	0.026	0.556	0.096	0.594
Threshold 0.09	0.039	0.805	0.131	0.781
Class 1				
Threshold 0.01	0.001	0.073	0.002	0.048
Threshold 0.05	0.011	0.776	0.027	0.698
Threshold 0.09	0.013	0.856	0.031	0.763
Class 2				
Threshold 0.01	0.000	0.009	0.002	0.026
Threshold 0.05	0.005	0.074	0.035	0.449
Threshold 0.09	0.008	0.191	0.047	0.574

Table 5.5 introduces the results from PCT 8k and 16k cluster accuracy tests. The version with 8k points diverged after 500 epochs, while the 16k version required around 300 epochs. The training for the 8k version with 600 epochs lasted for 8 hours and for the 16k version it took 29 hours. Interestingly the versions don't have too much difference when doing comparisons. In some classes the 8k version was even able to outperform the higher point amount version. This is counter-intuitive since the larger point cloud has double the information to the smaller one. The 16k version has more consistent recall and precision along all the classes and their thresholds. When comparing to the 2D intensity image method the latter outperforms both versions by a lot. Only the class 0 recalls are higher in PCT than 2D intensity image method.

Table 5.5 Point Cloud Transformer cluster accuracy results

Point amount	8192		16384	
Class 0	**Precision**	**Recall**	**Precision**	**Recall**
Threshold 0.01	0.212	0.193	0.219	0.223
Threshold 0.05	0.870	0.795	0.802	0.820
Threshold 0.09	0.897	0.812	0.852	0.853
Class 1				
Threshold 0.01	0.171	0.136	0.119	0.114
Threshold 0.05	0.826	0.629	0.705	0.669
Threshold 0.09	0.863	0.651	0.818	0.743
Class 2				
Threshold 0.01	0.112	0.087	0.068	0.064
Threshold 0.05	0.666	0.468	0.769	0.704
Threshold 0.09	0.802	0.543	0.814	0.727

Table 5.6 introduces the results from Point transformer 8k and 16k cluster accuracy tests. The version with 8k points diverged after 300 epochs and the 16k version required approximately 350 epochs. The training for the 8k version with 600 epochs lasted for 80 hours and for the 16k version it took 144 hours. Class 0 and class 1 results from the two different point amounts are rather similar and 8k point amount can even outperform the higher point amount in some thresholds. However the 16k point amount achieves more consistent results across all classes. Like other point cloud methods the threshold 0.01 results are lower than the 2D intensity image method which means the point cloud methods are not performing as well in extremely accurate predictions. Surprisingly the Point transformer with 8k points is able to outperform the 16k version at the lowest threshold across all classes. This indicates the larger point amount does not necessarily mean more accurate predictions at small thresholds.

Table 5.6 Point Transformer cluster accuracy results

Point amount	8192		16384	
Class 0	**Precision**	**Recall**	**Precision**	**Recall**
Threshold 0.01	0.291	0.286	0.278	0.264
Threshold 0.05	0.935	0.920	0.922	0.876
Threshold 0.09	0.955	0.932	0.943	0.888
Class 1				
Threshold 0.01	0.298	0.301	0.236	0.213
Threshold 0.05	0.864	0.874	0.876	0.789
Threshold 0.09	0.881	0.887	0.891	0.803
Class 2				
Threshold 0.01	0.248	0.220	0.155	0.145
Threshold 0.05	0.843	0.746	0.921	0.861
Threshold 0.09	0.878	0.777	0.939	0.878

5.3 Best point accuracy results

Table 5.7 introduces the best point accuracy results from all methods. The color coding is done in a way that each row's darkest tone indicates the highest results in that threshold and the lightest indicates the lowest. Hence the darker the column of the method is the better the results are in comparison to other methods.

Table 5.7 Best point accuracy results

	DGCNN 8k	DGCNN 16k	PointNet++ 8k	PointNet++ 16k	PCT 8k	PCT 16k	Pt 8k	Pt 16k	2D Intensity image method
Class 0									
Threshold 0.01	0.402	0.455	0.047	0.088	0.183	0.211	0.307	0.245	0.198
Threshold 0.05	0.947	1.000	0.263	0.531	0.829	0.845	0.973	0.890	0.695
Threshold 0.09	0.994	1.000	0.656	0.854	0.978	0.976	0.996	0.991	0.757
Class 1									
Threshold 0.01	0.236	0.344	0.000	0.013	0.125	0.110	0.235	0.214	0.194
Threshold 0.05	0.870	0.978	0.200	0.187	0.708	0.746	0.946	0.842	0.934
Threshold 0.09	0.990	1.000	0.870	0.902	0.981	0.987	0.995	0.970	0.967
Class 2									
Threshold 0.01	0.326	0.391	0.007	0.072	0.177	0.246	0.261	0.197	0.050
Threshold 0.05	0.916	0.985	0.064	0.490	0.745	0.806	0.949	0.913	0.818
Threshold 0.09	0.964	0.997	0.567	0.740	0.965	0.988	0.978	0.988	0.964

The color coding reveals right away that DGCNN methods with both point amounts clearly stand out in terms of accuracy. The 16k version is performing extremely well across all classes in 2nd and 3rd thresholds. When comparing DGCNN to 2D intensity image method DGCNN outperforms it with both 8k and 16k versions.

Another well-performer in best point accuracy is Point Transformer. Interestingly the 8k version achieves better results than the 16k version. Point Transformer also manages to perform better than the 2D intensity image method across all classes and thresholds. Another transformer-based method PCT performed slightly worse than Point Transformer. No substantial difference between PCT 8k and PCT 16k can be seen either. Comparison between PCT and 2D intensity image method reveals that PCT has more constant accuracies across all classes and thresholds. 2D intensity method had worse results in class 0 than other classes which was also shown in the cluster accuracy testing of the method.

Similarly to cluster accuracy tests the worst performer in best point accuracy is PointNet++ with both point amounts. Somewhat decent accuracies were acquired from the 3rd threshold but in comparison to other methods in other thresholds the results are clearly worse.

6. CONCLUSIONS

The target of this thesis was to research existing deep neural network methods for point clouds and analyze their ability to detect key points in cargo handling operations. Specifically, the goal was to find out how well convolutional and transformer-based neural network models can be utilized to find key points directly from keypoints. Comparison was made to the currently used method in the end-user application which utilizes 2D intensity image from the point cloud. The attributes of interest which were researched were the accuracy and the inference time in edge machine learning environment. With these boundaries the following research questions were formed:

1. In terms of inference time are transformer and convolutional based deep learning inference models suitable for edge machine learning environment?
2. Is convolutional or transformer based deep learning inference model for point clouds better than current 2D intensity image based method in terms of accuracy?

Answering the first research question required inference time testing of the models in different environments. In the introduction suitability was defined as a comparison to manual cargo handling operation, which can usually be done in a few seconds. The edge machine learning environment with GPU shows that the models are suitable for cargo handling operations in terms of inference time. However, the edge machine learning environment without GPU utilization could not perform the detection with most methods within a reasonable time which means they are not suitable.

The second research question was answered by performing cluster accuracy tests and best point accuracy tests on all methods and comparing them to the 2D intensity image method. Results show that no conclusion can be drawn as to the superiority of the convolutional or transformer-based method in terms of accuracy. Convolutional-based DGCNN with 16384 points performed the best out of all methods and it outperformed the 2D intensity image method. Additionally transformer-based Point Transformer with 16384 points performed better than the 2D intensity image method in terms of overall accuracy from both accuracy tests.

For future research on this topic, more data should be gathered. The total data amount of 1000 point clouds is not enough to train the models well enough for them to be used in real life operation. This dataset is too small and the actual capability of the methods cannot be studied well enough.

Additionally it was researched how the point amount of the point cloud affects inference time and accuracy. Higher point amount caused the inference time to grow substantially in CPU operations and in GPU operations the increase could be seen as well. However the improvement in accuracy was not as unambiguous as the improvement in inference time. For example in DGCNN and Point Transformer methods the difference between accuracies between 8192 points and 16384 points was very small. From this it can be concluded that a higher point amount does not always mean more accurate results. This should be further researched to find out which point amount works best for which method.

In addition to these the hyperparameters of well-performing methods should be researched further. In this work, only DGCNN hyperparameters were slightly adjusted which resulted in a significant improvement in results. Additionally, it would be important to analyze the effect of hyperparameter adjustment on the stability and overall performance of the models. The accuracy of transformer-based models could potentially be improved by hyperparameter adjustment and a closer look at the neural network architectures. The poor performance of PointNet++ should also be researched further. It is one of the most widely used methods for point cloud neural networks, which would suggest the model would perform better than the results of this thesis suggest.

REFERENCES

[1] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” *ArXiv Prepr. ArXiv170602413*, 2017.

[2] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, “Dynamic Graph CNN for Learning on Point Clouds,” *ACM Trans. Graph. TOG*, vol. 38, pp. 1–12, 2019.

[3] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu, “PCT: Point cloud transformer,” *Comput. Vis. Media*, vol. 7, no. 2, pp. 187–199, Apr. 2021, doi: 10.1007/s41095-021-0229-5.

[4] Peter. Flach, *Machine learning : the art and science of algorithms that make sense of data*. Cambridge: Cambridge University Press, 2012.

[5] Anurag Bhardwaj, Wei Di, and Jianing Wei, *Deep Learning Essentials : Your Hands-on Guide to the Fundamentals of Deep Learning and Neural Network Modeling*. Birmingham, UK: Packt Publishing, 2018.

[6] E. Alpaydin and F. Bach, *Introduction to Machine Learning*. Cambridge, UNITED STATES: MIT Press, 2014. [Online]. Available: http://ebookcentral.proquest.com/lib/tampere/detail.action?docID=3339851

[7] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, Global Edition*. NOIDA, INDIA: Pearson Education Limited, 2016.

[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement Learning: A Survey,” *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, May 1996, doi: 10.1613/jair.301.

[9] A. Lopez, “Machine Learning Project Steps -,” Jun. 05, 2019. https://t2client.com/blog/aamlpsal/ (accessed Oct. 06, 2021).

[10] D. Esposito, *Introducing machine learning*, 1st edition. Place of publication not identified: Published with the authorization of Microsoft Corporation by Pearson Education, 2020.

[11] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms : A Classification Perspective*. Cambridge: Cambridge University Press, 2011.

[12] “Classification: Precision and Recall | Machine Learning Crash Course | Google Developers.” https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall (accessed Mar. 11, 2022).

[13] A. Trask, *Grokking Deep Learning*, 1st edition. Manning Publications, 2019.

[14] J. D. Kelleher, *Syväoppiminen*. Helsinki: Terra Cognita, 2020.

[15] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*, 4th ed. Pearson, 2020.

[16] M. W. Gardner and S. R. Dorling, “Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences,” *Atmos. Environ.*, vol. 32, no. 14, pp. 2627–2636, Aug. 1998, doi: 10.1016/S1352-2310(97)00447-0.

[17] Inc. O’Reilly Media, *Deep Learning*, (2015).

[18] G. Ciaburro and B. Venkateswaran, *Neural Networks with R*. Birmingham, UK: Packt Publishing, 2017.

[19] "CS231n Convolutional Neural Networks for Visual Recognition - Neural Networks 1," Stanford University. Accessed: Nov. 05, 2021. [Online]. Available: https://cs231n.github.io/neural-networks-1/

[20] G. E. Hinton, "How Neural Networks Learn from Experience," *Sci. Am.*, vol. 267, no. 3, pp. 144–151, 1992.

[21] L. Deng and D. Yu, "Deep Learning: Methods and Applications," *Found Trends Signal Process*, vol. 7, no. 3–4, pp. 197–387, Jun. 2014, doi: 10.1561/2000000039.

[22] R. J. Lopez, *Advanced engineering mathematics*. Boston: Addison-Wesley, 2000.

[23] "CS231n Convolutional Neural Networks for Visual Recognition - Convolutional Neural Networks," Stanford University. Accessed: Nov. 05, 2021. [Online]. Available: https://cs231n.github.io/convolutional-networks/

[24] A. Vaswani *et al.*, "Attention is All you Need," *ArXiv*, vol. abs/1706.03762, 2017.

[25] D. Sussillo and L. F. Abbott, "Random Walk Initialization for Training Very Deep Feedforward Networks," *ArXiv Neural Evol. Comput.*, 2014.

[26] K. Doshi, "Transformers Explained Visually (Part 3): Multi-head Attention, deep dive | by Ketan Doshi | Towards Data Science." https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853 (accessed Dec. 08, 2021).

[27] S. A. Bello, S. Yu, C. Wang, J. M. Adam, and J. Li, "Review: Deep Learning on 3D Point Clouds," *Remote Sens.*, vol. 12, no. 11, 2020, doi: 10.3390/rs12111729.

[28] K. Jo, S. Lee, C. Kim, and M. Sunwoo, "Rapid Motion Segmentation of LiDAR Point Cloud Based on a Combination of Probabilistic and Evidential Approaches for Intelligent Vehicles," *Sensors*, vol. 19, p. 4116, Sep. 2019, doi: 10.3390/s19194116.

[29] "LiDAR and ToF Cameras – Technologies explained – ToF-Insights." https://tof-insights.com/time-of-flight-lidar-and-scanners-technologies-explained/ (accessed Jan. 20, 2022).

[30] C. Qi, H. Su, K. Mo, and L. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," *2017 IEEE Conf. Comput. Vis. Pattern Recognit. CVPR*, pp. 77–85, 2017.

[31] W. Wu, Z. Qi, and L. Fuxin, "Pointconv: Deep convolutional networks on 3d point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9621–9630.

[32] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6411–6420.

[33] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 16259–16268.

[34] "Facebook brings GPU-powered machine learning to Python | InfoWorld." https://www.infoworld.com/article/3159120/facebook-brings-gpu-powered-machine-learning-to-python.html (accessed Jan. 31, 2022).

[35] Benny, *Pytorch Implementation of PointNet and PointNet++*. 2022. Accessed: Feb. 24, 2022. [Online]. Available: https://github.com/yanx27/Pointnet_Pointnet2_pytorch

[36] Y. You, *Pytorch Implementation of Various Point Transformers*. 2022. Accessed: Feb. 24, 2022. [Online]. Available: https://github.com/qq456cvb/Point-Transformers

[37] V. SaRoDe, *Learning3D: A Modern Library for Deep Learning on 3D Point Clouds Data.* 2022. Accessed: Feb. 24, 2022. [Online]. Available: https://github.com/vinits5/learning3d