TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

OLLI ELORANTA

DEEP LEARNING-BASED OBJECT DETECTION WITH POINT CLOUD DATA

Bachelor of Science Thesis

Examiner: Joni Kämäräinen
Submitted: September 27, 2018

ABSTRACT

OLLI ELORANTA: Deep learning based object detection with point cloud data
Tampere University of Technology
Bachelor of Science Thesis, 27 pages
September 2018
Bachelor's degree in Information Technology
Major: Machine Learning
Examiner: Joni Kämäräinen

Keyword: Lidar, Point cloud, Deep Convolutional Neural Networks, Machine Learning

Deep convolutional neural networks (CNNs) are used in various tasks, especially in classification and object detection in two-dimensional images. In this work, two deep convolutional neural networks were experimented for detecting objects from three-dimensional point cloud data. Neural network models can utilize the additional depth information of point clouds to learn spatial features based on the locations of the data points.

Using deep convolutional neural networks for object detection has promising results but creating a point cloud dataset from scratch requires time. The aim of the experiments was to create an own dataset that fits the pre-defined models. The dataset had only several examples for experimenting the models, but good datasets should be as large as possible. The point clouds need complex processing to ensure effectiveness and precise accuracy for the model.

TIIVISTELMÄ

OLLI ELORANTA: Deep learning based object detection with point cloud data
Tampereen teknillinen yliopisto
Kandidaatintyö, 27 sivua
Syyskuu 2018
Tietotekniikan kandidaatin tutkinto-ohjelma
Pääaine: Machine Learning
Tarkastaja: Joni Kämäräinen
Avainsanat: Pistepilvi, Lidar, Syvä konvoluutioneuroverkko, Koneoppiminen

Syviä konvoluutioneuroverkkoja käytetään monissa tehtävissä. Tyypillisiä esimerkkejä ovat luokittelu ja kappaleiden tunnistus valokuvista. Tässä työssä kokeiltiin kahta eri neuroverkkoarkkitehtuuria kappaleiden tunnistukseen kolmiulotteisista pistepilvistä. Neuroverkot voivat hyödyntää pistepilvissä olevaa syvyystietoa ja käyttää sitä avaruudellisten ominaisuuksien oppimiseen.

Syvien konvoluutioneuroverkkojen käytöllä kappaleiden tunnistuksessa on lupaavia tuloksia, mutta mallin opetusesimerkkien luomiseen menee aikaa. Neuroverkkomalleihin pyrittiin luomaan opetusesimerkkejä ja muokkaamaan toteutusta omiin esimerkkeihin sopivaksi. Neuroverkkojen opetusesimerkkejä luotiin vain muutamia, mutta todellisuudessa niitä pitäisi olla mahdollisimman paljon. Pistepilvet vaativat monimutkaista käsittelyä, jotta neuroverkosta saadaan tehokas ja tarkka.

PREFACE

This thesis was written during the summer of 2018. I would like to thank Jussi Poikonen for giving me the opportunity of writing the thesis during my work as a summer trainee at Rolls-Royce in Turku and for helping me with writing the thesis. The possibility to do the thesis at work allowed me to test problems in the thesis further than I could have done by myself with the access to data and advices in programming. I am also grateful for the feedback given by Jussi and my family about the thesis.

September 27, 2018

Olli Eloranta

Tampere, Finland

CONTENTS

LIST OF ABBREVIATIONS

2D	*Two-dimensional*
3D	*Three-dimensional*
CNN	*Convolutional neural network*
Lidar	*Light detection and ranging, laser remote sensing technology*
MLP	*Multi-layer perceptron*
MSE	*Mean squared error*
Radar	*Radio detection and ranging, radio wave remote sensing technology*
ReLU	*Rectified linear unit*

1. INTRODUCTION

Self-driving cars have been thought to change road transportation and to improve the shipping of goods on land [1]. With autonomous trucks, shipping could be done without taking the driver's working hours or sleep into consideration and it could be done without pauses. Autonomous cars allow the disabled and underaged to travel without a driver and driving while intoxicated would not be possible. By removing human error from vehicles, they would become safer. Efficient autonomous parking would give space to more buildings in cities now taken by parking lots.

Similar to cars, ships can have large benefits by becoming autonomous [2]. The use of unmanned ships would mean removing dozens of crew members. Rooms, such as accommodations and the bridge, could be removed as well. This saves money in salary costs and gives more space for cargo. Cargo ships are often the most cost-efficient way to carry a large cargo and being unmanned would be even less costly. In addition to the lower costs, unmanned vessels could be safer without a crew. After all, human error accounts to most of the accidents at sea.

Both autonomous cars and ships have the need of precise technology, such as cameras and laser scanners, to process their surroundings. Cars need to be able to quickly process its immediate surroundings and the traffic on the road. Ships have more time to react to other objects, but there are still a lot of other vessels and sea markers that need to be taken into consideration. Detecting these objects well in advance can help in plotting a correct course.

The aim of this thesis is to study and test the detection of objects in a maritime environment. Detection from images is widely studied and implemented so this work will be done with three-dimensional data of laser scanned points. Two different neural network architectures from other studies that fit the problem, PointNet [3] and VoxelNet [4], are used to test the object detection. The major problem is the creation and pre-processing of a dataset for the neural network models. Data visualization and manipulation is more complex with 3D data than with 2D image data that neural networks more commonly use.

The work consists of a theoretical background of the methods used and an implementation that tests the detection of objects in 3D. Laser imaging and the resulting point cloud data are presented in Section 2. In Section 3, the architecture of basic neural networks and deep convolutional networks are presented, followed by Section 4 on how to teach them. Section 5 presents the experiments for object detection with two different architectures, PointNet and VoxelNet. Finally, in Section 6, the most meaningful results are presented as a conclusion. Conclusions include the discussion of the success of the work and possible future research.

2. POINT CLOUD

Lidar (light detection and ranging) is a type of remote sensing technology that uses rapid light pulses to measure ranges to its surroundings [5]. The technology is similar to the more commonly used radar (radio detection and ranging). The lidar sensor emits intense beams of light that reflect from surfaces around the sensor. The device measures two characteristics from the beams of light emitted:

1. The time difference between the emitted and reflected pulses
2. The angle of emitting the pulse

With these measurements the device can produce accurate three-dimensional coordinates for a large set of points. This set of points is often called a point cloud which has up to several million points. Each point in the point cloud usually contains its X, Y and Z co-ordinates along with the intensity of the returned beam and the time of measurement.

Three-dimensional mapping with lidar is used in various applications where three-dimensional information is needed of different surfaces. Lidar can be used for modelling streets [6] and studying aerosols in the atmosphere [7]. Laser imaging is also used in applications such as the navigation systems of self-driving cars [8] and in robot vacuum cleaners [9].

Elevation maps of the earth are commonly created with air-based lidar systems [5]. Flying over the area of measurement while scanning the area below the aircraft produces elevation data of huge areas, as shown in Figure *1*.

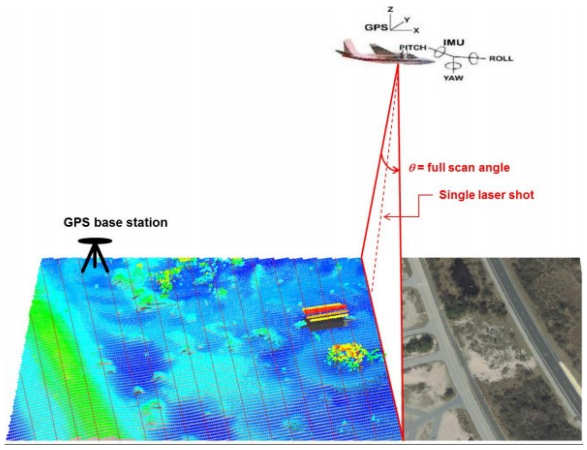

Figure 1. ***Airborne lidar performing line scanning [5]***

However, modern lidar scanners are not limited to line-scanning or measurements on a single plane, which are common in aerial measurements [10]. With the advances in laser technology, it is possible to mount lidar devices to mobile platforms to accurately measure the surroundings of moving vehicles [5]. The lidar scanners utilize multiple coordinated emitter-detector pairs that rotate 360 degrees multiple times a second. This allows a wider field-of-view of the scanner's surroundings with the capability to map the sensor's entire surroundings, for example a room. These 360 lidars can collect hundreds of thousands of points in a second.

Three-dimensional point clouds can be projected to a two-dimensional plane to view them on a computer screen [11]. This can be done by projecting the points to a camera's image coordinates. A 3x4 camera projection matrix which transforms homogenous 3D coordinates to homogenous 2D coordinates is needed for projecting the points. The equation for an approximation of moving from 3D to 2D camera coordinates is

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \cong \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{1}$$

where the first vector contains the projected homogenous 2D points, the 3x4 array is the camera projection matrix and the last vector contains the 3D points to be projected [12]. Estimating the values of the camera matrix can be done with points from the 2D camera plane and 3D world plane that correspond each other. The correspondence points can be found for example by picking corresponding points from a camera's image and a lidar scanner's point cloud.

3. NEURAL NETWORKS

Artificial neural networks are mathematical models which aim to emulate the neural architecture of human brains [13]. The human mind is excellent at learning and generalizing data, which has led to the interest of replicating it mathematically. The earliest models of neural networks have been used in the 1950's [14]. Because of the limited computational power available at the time, neural networks were not effective enough to reach the amount of usage as today. For a visual recognition competition in 2012, Alex Krizhevsky, Ilya Sutskeyer and Geoff Hinton created a deep neural network that reached low error rates [15]. After this advance, the development of artificial intelligence and neural networks has sky-rocketed.

3.1 Basic structure

In the structure of a basic neural network, neurons are connected to each other in layers [13]. The neural network consists of three types of layers: an input layer, several hidden layers and an output layer. The basic structure is shown in Figure *2*.

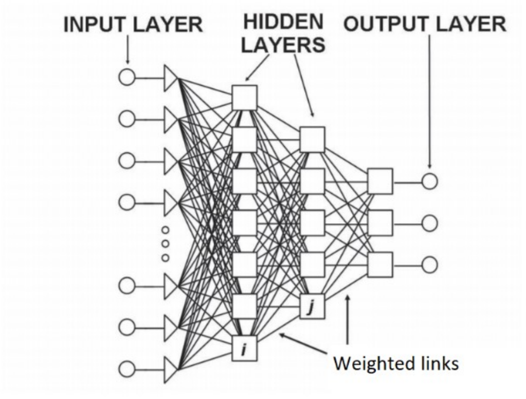

Figure 2. ***The basic structure of a neural network. Adapted from [13].***

The input layer receives the input data from the user and transfer it to the hidden layers. The data is transferred through weighted links between the neurons. The number of hidden layers can vary depending on the problem. The neurons calculate outputs from the given data. This is done with their activation function based on each neuron's weights and with an added bias. Finally, the processed data is given from the hidden layers to the output layer.

Neural networks with the structure described above are called multi-layer perceptrons, or MLPs [16]. The layers are fully-connected, meaning that a neuron has a connection to each of the neurons on the next and previous layers. The MLP is trained to work as a non-linear mapping between the input and output vector.

Neurons are traditionally the smallest parts of neural networks, based on the neurons in human brains [17]. They can be thought as a method for weighing evidence to make decisions. The inputs are the evidences, and the output tells if the decision is made or not. Perceptrons are the simplest implementation of a neuron with binary inputs and outputs. Each input has a weight which represents its importance. The output is calculated by the weighted sum of the inputs. The calculation which is used in a neuron is called its activation function. Each neuron also has a bias, which is a value added to the calculation that affects the output.

Instead of basic perceptrons, neural networks' layers usually consist of non-binary activation functions [17]. These types of neurons can have inputs and an output that range from 0 to 1. For example, instead of calculating the weighted sum, sigmoid neurons use a sigmoid function as its activation function. This smoothes out the output. Thus, the resulting values can be more precise.

Rectified linear units (ReLU) have been proved to be more effective in some applications of neural networks than previously used neurons with a sigmoid function [18]. ReLUs have an activation function of

$$f(x) = max(0, x)$$

where the unit is deactivated below 0 and gives a linear output after it. This gives sparse representations to the neural network. Many of the hidden units in the network produce zeros and are not activated. Networks with ReLU are faster to train than previous ones because of the simple activation function.

3.2 Deep convolutional network

Networks with fully-connected layers, as introduced above, are good at some simple classification tasks with small data [17]. However, lately the most popular way of using neural networks has been with deep convolutional networks. This allows more complex data to be used for teaching with faster performance. A network with multiple convolutional layers and generalizing pooling layers has been proved to perform very well.

Convolution is a formal mathematical operation like multiplication and addition [19]. Convolution has its own star operator

$$y[n] = h[n] * x[n] \tag{2}$$

which can be confused with the multiplication operator. Convolution takes two signals, the input signal and a filter, to produce an output signal.

The mathematical form for convolution is

$$y[i] = \sum_{j=0}^{M-1} h[j]x[i-j] \tag{3}$$

where h is the filter with M points, x the input signal and i the index of the point that is being calculated [19]. As j runs through 0 to M-1, each point of *h[j]* is multiplied with the sample from the input signal, *x[i-j]*. The products are added together to get the output value of the point. One dimensional convolution is presented in **Virhe. Viitteen lähdettä e i löytynyt.**.

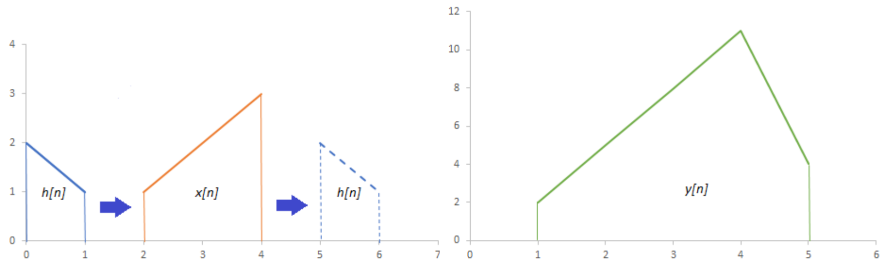

Figure 3. Example of a one-dimensional convolution.

The input signal *h[n]* has the values (2, 1) and *x[n]* has the values (1, 2, 3). The signal *h[n]* can be thought to slide over each of the points of *x[n]* to calculate the output *y[n]*. For example, the equation (3) can be used to calculate *y[i]* at *i=1* to be 2*2+1*1=5 as the convolved point. This corresponds to the point at n = 2 in the output signal.

Convolution can also be done with 2D signals, such as images [20]. Instead of having an impulse response, the convolution is done with a 2D filter kernel. Each point in the output is influenced by a group of points that are inside the filter. Different filters produce different results. An example convolved image of a baggage line x-ray scan is presented in Figure 4.

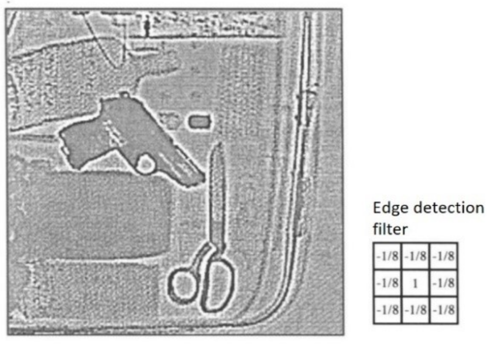

Figure 4. Convolution of an image with a 2D filter [20].

The edge detection filter is moved from left to right and top to down on each of the input pixels. This creates a convolved output with highlighted edges. Each pixel in the resulting convolution is calculated so, that the pixel values of the image are multiplied with the corresponding values in the edge detection filter. Finally, the values are added together.

Convolutional neural networks have convolutional layers in them [21]. The convolutional layers consist of feature maps that are produced with convolutional filtering. Each neuron in the network receives a small area of the previous layer that has gone through a filter. This area is called the neuron's local receptive field. Each of the neurons on the same feature map share the same filters but receive different inputs from the previous layer. The filter parameters of a neuron in a convolutional neural are adjusted when training the network. The input and feature maps of a convolutional neural network are presented in Figure 5. The image input and the feature maps of a convolutional networkFigure 5.

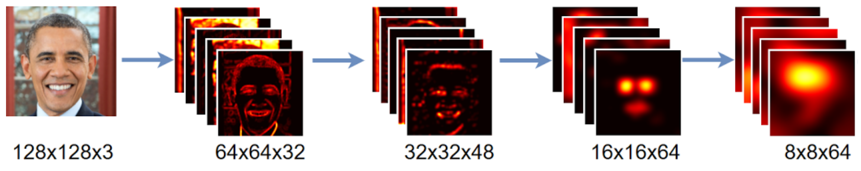

Figure 5. The image input and the feature maps of a convolutional network [22].

The input image is given to the first layer, that produces 32 different feature maps. This is done with each layer, and the network learns to characterize important features from images given to it.

Convolutional networks use pooling layers to simplify the outputs from convolutional layers [17]. In deep neural networks, the number of neurons would be immensely high without some type of simplification. This would lead to slow performance of the network. A pooling layer summarizes small regions of the convolutional layer's output and produces a smaller layer. Pooling allows the network to generalize data better. Max-pooling is a common procedure that outputs only the maximum activation of the region. By having a 2×2 pooling region, you can make the layers a fourth of the size of the previous one.

A simple, fully-connected network with a couple of hidden layers can be considered as a deep neural network [17]. However, a convolutional neural network with convolutional and pooling layers for an output can be much more effective. In tasks where a vector output is needed, at least one fully-connected layer is used at the end of the network. This is done to produce the final output from the convolutional layers' multi-dimensional output. An example structure of a convolutional neural network for image classification is presented in Figure 6.

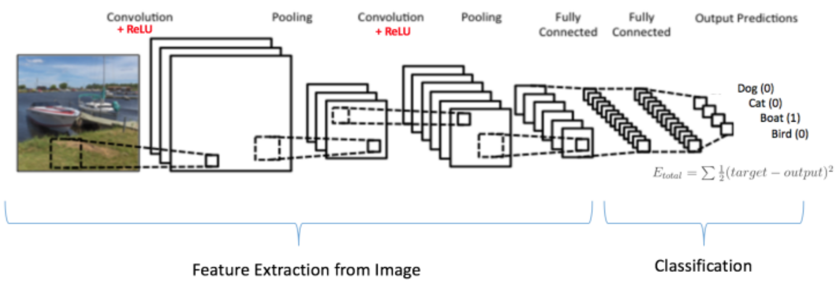

Figure 6. Example architecture of a convolutional neural network [23].

Convolutional neural networks have smaller sized layers due to pooling and they learn faster than fully-connected networks. This allows the network to be even dozens of layers deep and can learn the characteristics of more complicated data, such as in image recognition.

To further improve the generalization and thus the performance of a convolutional neural network, batch normalization can be added to its layers [24]. This normalizes the activations belonging to the input's feature map of a layer by subtracting the channel's mean and dividing by its standard deviation. This leads to having zero mean and unit variance in the layer's activation. With batch normalization the training of the network is faster, and it enables higher learning rates and improved accuracy.

4. TRAINING A DEEP CONVOLUTIONAL NETWORK

A convolutional network can be taught to do multiple different tasks [17]. The network learns the training data given to it by trying to minimize the error of its predictions. The user needs to define what architecture to use and with what parameters. The aim for the network is to learn to depict the data given to it. Then again, learning training examples too well results in overfitting and the network's performance is reduced. Creating a good dataset is extremely important to get good performance.

4.1 Typical tasks

The applications of neural networks can vary widely [17]. Common tasks include classification, object detection and semantic segmentation. These tasks can be solved with other machine learning methods as well, but neural networks have been proved to be effective. Examples of the three common tasks are presented in Figure 7.

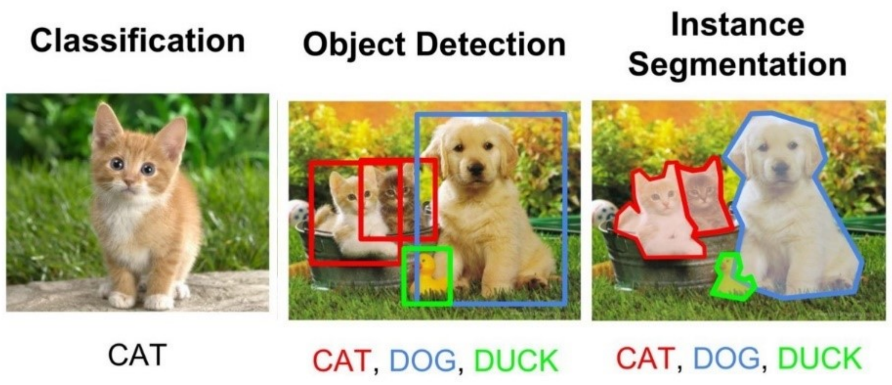

Figure 7. A comparison between common machine learning tasks. Adapted from [25].

In instance segmentation, only the semantically segmented objects that are searched for are displayed as segmented.

A typical problem for neural networks is data classification [17]. The aim in classification is to correctly classify given inputs to corresponding labels. For example, pictures of cats and dogs can be classified by training the neural network to distinguish these from one another. Trying to express the features of an animal or any other object algorithmically is extremely difficult, which is why neural networks are often used in this type of a task.

A dataset for a classification task consists of the images and corresponding labels [26]. Each label has the information what class the image belongs in. Of the three different applications that are presented here, classification has the simplest datasets. For example, pictures of images with cats and dogs can be stored in two folders for distinction.

Object detection aims to tell if an image contains an object or not [27]. If an object is found, its location is also presented. Multiple objects from different classes can be searched from the image. Object detection might require more pre-processing than classification. The network might need cropped objects or bounding boxes in the training data to learn detection.

Another type of machine learning task is semantic segmentation [28]. Each point of data is labelled to its corresponding class. With images, semantic segmentation aims to give a label to each pixel. The output can be immediately interpreted as the labels of pixels in the image when transformed into the same dimensions as the input image. Semantic segmentation can also be used as a way of detecting objects. Concentrations of points where pixels are labelled as the same class can be detected as objects [27].

4.2 Training process

The learning process of neural network can be thought as a problem of updating the weights and biases of the neurons in the network [29]. Neural networks learn from examples automatically and their performance is improved by iteratively updating the weights. Thus, the data of the examples given to the network is the focus for the users who want effective networks. The user must know what information is available to the neural network and choose a learning paradigm.

There are three learning paradigms used in neural networks: supervised, unsupervised and hybrid [29]. In supervised learning, the network is given correct labels to all the training data it is given. Weights are updated to produce answers as close as possible to the correct ones. On the contrary, unsupervised learning does not require the correct answers. An unsupervised neural network searches patterns and correlations between the data examples. The correlations are organized to form categories based on what the network has noticed. Hybrid learning combines both supervised and unsupervised learning where the correct labels of only parts of the data are given to the network.

A cost function is declared to find the correct weights and biases for the neurons [17]. The cost function is smaller the closer the neural network's output is to the real answer. An example cost function could be the mean squared error, or MSE. Minimizing this is done with a method called gradient descent. Gradient descent recognizes what changes in the weights and biases make the cost function smaller. This is done by repeatedly calculating the gradient of the cost function. A simple one-dimensional visualization of gradient descent is shown in Figure 8.

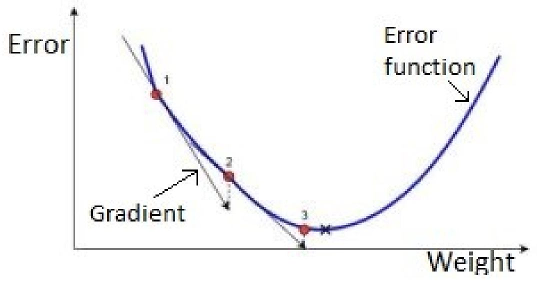

Figure 8. A one-dimensional visualization of gradient descent. Adapted from [30].

The above visualization shows how gradient descent minimizes the cost function based on the gradients. At the cross on the bottom, the weight *w* has the optimal value of giving the lowest error. By following the gradient, the error is reduced. With some iterations a good approximation of the correct value can be found.

The stochastic gradient descent is a simplification of calculating the gradient [31]. Instead of calculating the gradient by testing all the data, stochastic gradient descent estimates the gradient based on randomly picked examples. This is a much faster way of calculating the gradient and allows it to be done while training the model. Stochastic gradient descent optimization methods, such as AdaGrad, RMSProp and Adam are used for optimizing machine learning models [32]. Adagrad is used with sparse gradients and RMSProp with on-line and non-stationary settings. Adam is a combination of these two that aims to have both the advantages. The Adam method computes individual learning rates for parameters from the calculated estimates of the first and seconds moments of the gradients.

An algorithm called backpropagation is used for the calculations in gradient descent [17]. Backpropagation computes the partial derivatives of the cost function with respect to any weight or bias in the network. The neural network tries to model the data, which produces some errors in the output. Backpropagation goes backward through the layers and tries to adjust the network's parameters. The adjustments are done according to the gradients to minimize the error given by the cost function.

A learning rate is added to the backpropagation for the gradient descent to work correctly [17]. It is a small value that limits the changes done by the gradient descent. This is done to prevent the gradient descent on doing too large changes in the model. The optimal learning rate depends on the architecture of the model and the training data it has. It is usually found by trial and error by the user by observing the performance of the model.

4.3 Datasets and training

A dataset is needed for a neural network for training [17]. Evaluating the final performance of the model on training data from results in overfitting [33]. A separate test set is required apart from the training set. This is used for getting an un-biased result from the neural network. The test set is not used directly in the training of the neural network.

To start training, a neural network takes the training dataset as input to the model [17]. When using stochastic gradient descent, the model is taught with batches produced from the training set. Averaging the gradients of the batches gives an estimate of the true gradient. This speeds up the learning of the model. Going through all the data, or all the batches, means the completion of one epoch of training. The number of epochs needed to train a model depends on the amount of training data and the complexity of the model.

Many values in a neural network model need to be changed to find the combination of parameters with the best results [17]. The number of layers or the parameters of the layers differ in different situations. By following the model's performance on the test set, the user can see how much the changes have helped. Tweaking hyperparameters, such as the size of batches, is a way to make the model increasingly accurate after the user has found approximately good values for them.

4.4 Preventing overfitting

If a model includes more terms than necessary or uses too complicated approaches to model the data given to it, it might overfit on the training data given to it [35]. This means that the model focuses too much on irrelevant details of the data. Neural networks are meant to be taught to generalize the data, not to learn the specific characteristics of each training example given to it. [36] A visualization of model preciseness on a set of data points is presented in Figure 9.

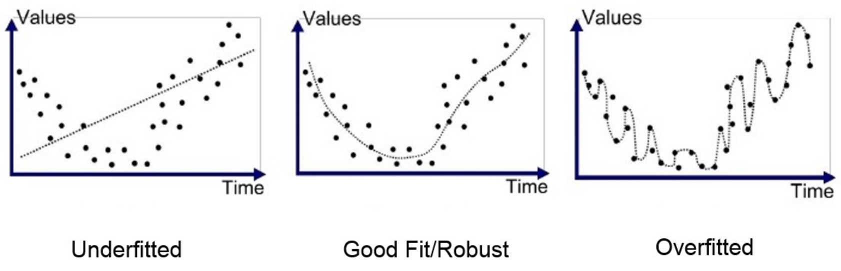

Figure 9. Fitting a model on points [37].

An underfitted model is inaccurate and too simple. On the other hand, an overfitted model is too accurate. The overfitted model is overly complex and is not accurate if new data points are given to it. The complexity of the architecture of a network should depend on the amount of available data to prevent overfitting [36]. Keeping the network relatively simple helps in generalizing the data.

The most common way to avoid overfitting during training is to train the model the right amount of time. [36] Training a network for too long teaches the training data too well and the neural network loses accuracy. Conversely, the network does not learn to model the data well enough if it is trained for too little time. Injecting noise to the training data also helps in the generalization and in preventing overfitting.

Combining multiple neural network models and averaging their outputs nearly always improves the overall performance [38]. However, training multiple deep neural networks is time-consuming. A way to reach similar performance without the need of additional models is to add dropout to the used network. With dropout, units are dropped out from the network, meaning that they are temporarily removed during training. For example, half of the units can be randomly removed from the network. This leads to having a possible collection of 2^n different thinned networks with shared weights during training, where n is the number of neurons in the layer with dropout. Each of these thinned networks gets trained rarely, or never, but they can be combined to work as a single network during test time.

5. DEEP ARCHITECTURES FOR POINT CLOUD OBJECT DETECTION

Two different methods are used to test the detection of objects in point cloud data gathered with a lidar device. PointNet is a deep learning architecture that is used in 3D object classification and segmentation. It takes point clouds directly as inputs and outputs [3]. VoxelNet is a 3D detection framework that transforms the point cloud into voxels that each contain a small amount of points. It produces bounding boxes based on the features of the voxels [4]. These two approaches require different labelling and point cloud pre-processing. The creation of a dataset is the main problem in these experiments.

5.1 PointNet

The PointNet architecture has the capability of either object classification or semantic segmentation in 3D [24]. For object classification, an input is directly sampled as a point cloud or pre-segmented from a scene. This outputs the probability scores for all the classes. The network has capabilities of the semantic segmenting parts of objects or objects in scenes. The latter semantic segmentation method is used in this work because the object segmenting can be used as a way for detecting interesting objects in large scenes.

5.1.1 Architecture

The PointNet architecture has three key modules that tackle the complexity of point cloud data [3]. A max pooling layer works as a symmetric function to combine information from all the points. This is done to make the model invariant to input permutation, which is a problem with the unordered nature of point clouds. A local and global information combination structure manages to keep data of the local neighbourhood intact after the max pooling. The max-pooling has previously formed a vector from its multi-dimensional input. Thirdly, two joint alignment networks align the input points and point features. These give improved robustness in simple geometric transformations.

The segmentation network uses the classification network as its base [3]. The architecture of PointNet is presented in Figure 10.

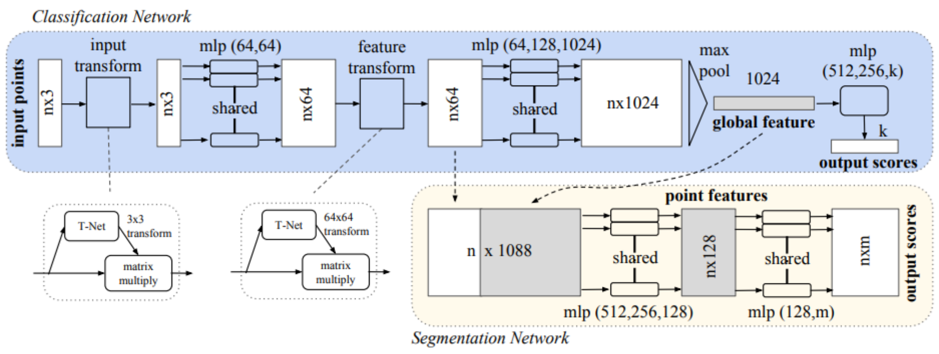

Figure 10. The PointNet architecture [3].

The network takes *n* points as input and applies the input and feature transformations. Both the transformations are done with a mini-network. Called “T-net” in the image, the mini-networks are similar to the structure of the whole PointNet network. “mlp” stands for multi-layer perceptron, meaning a fully-connected neural network. The numbers in brackets are the layers’ sizes.

The transformed point features are aggragated by max pooling and the classification scores are produced for the classification network [3]. The semantic segmentation network concatenates the global and local features. The concatenated features run through the segmentation layers to produce the output as scores for each point. All layers in the model, except the last one, include ReLU and batch normalization.

5.1.2 Creating a dataset

The semantic scene segmentation is experimented with the Stanford 3D semantic parsing dataset in the PointNet paper [3]. The dataset contains 3D scans from six different areas including 271 rooms. Each point is annotated with one of the classes from thirteen categories, such as chair, table, wall and floor. The PointNet implementation has pre-processing steps that are created for the dataset.

The points are split by room and each room is sampled into blocks of an area of a square metre [3]. The semantic segmentation network predicts per point class in each block. At training time each block has 4096 randomly sampled points. At test time, all the points are used. To get the results in the PointNet paper, six different models were taught. Each model has a different area from the six areas of the Stanford 3D dataset as a test set. The predictions of all the six models are combined to calculate more precise predictions than a single model.

The creation of an own dataset for PointNet is researched for this work. The aims were to correctly segment scenes from a riverside environment. This allows to recognise the surroundings of a lidar scanner attached to a boat. The boat operated along the river Aura in Turku, Finland. The riverside has ships, buildings and the wharf along the riverbank as interesting classes which the model tries to distinguish from one another.

The training data is gathered from a lidar scan converted into a single text file. Each line of the file contains the XYZ-coordinates, the timestamp and the intensity of the measured point. The point cloud file is in sensor frame, which means that the base coordinate (0, 0, 0) is always on the base of the lidar scanner. This means that small parts of the point cloud need to be processed at a time, because the movement and rotation of the boat makes the points overlap.

A Python program first reads the point cloud file. It takes sections of a couple of seconds of points based on the timestamps. These sections are shown to the user for the cropping of individual objects. An example section of the point cloud paired with an image taken from the same area is presented in Figure 11.

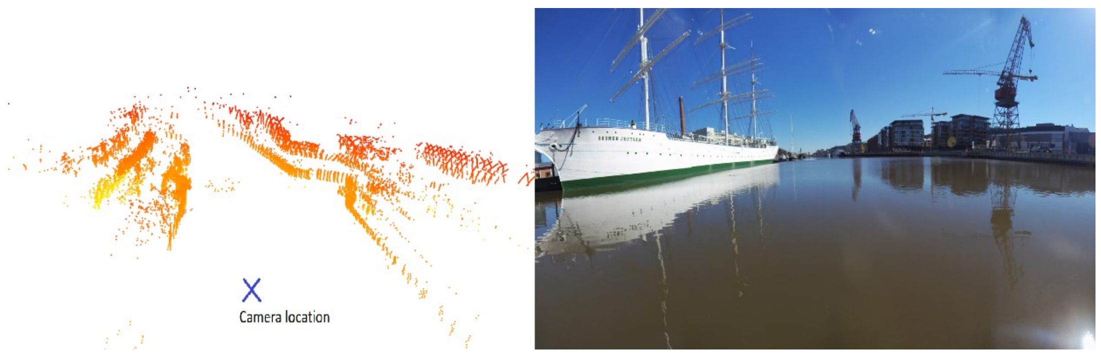

Figure 11. An example point cloud and corresponding photo of the Aura dataset

Each of the cropped objects is saved as an own file named after the class it represents. After all the objects are extracted, they are combined back into a single point cloud. The classes are added as labels to it as an extra column based on the object’s name. The coordinate values are divided by 20 to make training and testing faster, because PointNet groups points in a one square metre area. The points are also shifted to all be positive numbers because of the computations in PointNet. The Python library Numpy [39] is used for the computations, saving and loading with arrays. The library Open3D [40] is used for the visualization and cropping of point clouds.

After the dataset is created, the h5py Python package is used to transform it into a binary HDF5 format. HDF5 is used to store large amounts of data effectively [41]. A Python program released along the PointNet model’s code in GitHub is used as a base to convert the files into a format that PointNet uses as inputs. Small adjustments are made to convert most of the previously made point cloud scenes into a training set and the rest of them into a test set. The program stores a thousand cubes of 4096 points into each HDF5 file. Each point in the blocks has a vector with XYZ-coordinates, colour of the point and normalized location as to the scene. The recorded data has not measured the colour of the scenes, so the colour is added as zeros.

5.1.3 Training and results

A dataset created in the manner described in the previous subsection is used in the training and evaluation of the PointNet network. The created dataset is rather small with ten point cloud scenes of the river environment. Only ten scenes are created because of the small amount of available data. Each scene has approximately fifteen to thirty thousand points. Seven scenes are used as a training set and three as a test set for the model. The scenery along Aura is quite similar in different parts, so having a small dataset is thought to learn the area well enough for experimenting with the model.

Small adjustments were needed in the available code to train the PointNet model. The test data is the HDF5 test data created from the riverside scenery instead of one of the six areas in the Stanford 3D dataset. The number of classes is also changed in different parts of the PointNet code. After these adjustments a PointNet model is ready to be trained with our dataset.

The model is trained for 20 epochs. Stochastic gradient descent with the Adam optimizer is used with 0.001 as the initial learning rate. Training with a Nvidia GTX 1080 GPU took around 20 minutes. The semantic segmentation accuracy reached approximately 85% while evaluating with the test set of three scenes. Larger datasets, such as the Stanford 3D dataset would take even days to train on a similar setup. One of the point clouds on the test set is shown on top of a corresponding image in Figure 12.

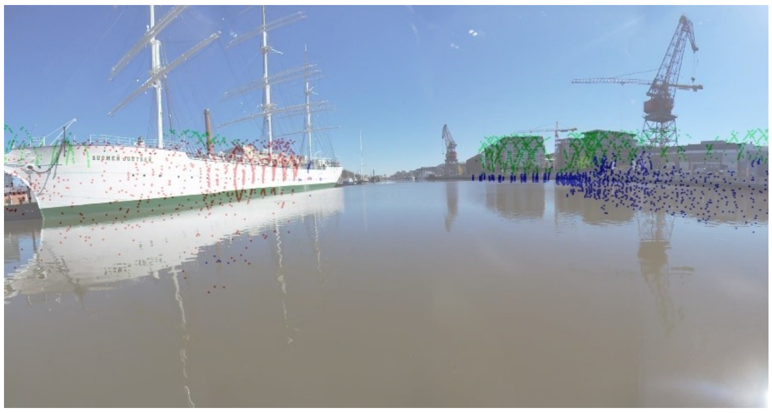

Figure 12. A semantic segmentation result of PointNet illustrated on top of the corresponding image.

Points coloured as red, green and blue correspond to the classes ship, building and wharf respectively. The projected points do not fit the image perfectly because of the wide angle of the camera. The lidar scanner also picks points from some of the reflections, so predicted points can be seen in the water as well.

Training with such a small dataset is prone to overfitting which can be seen from the testing results. A relatively high accuracy of approximately 70% was achieved already after two epochs. Trying to segment different data with the model might prove to be inaccurate. However, the results prove that the model is capable of learning to segment data from hand-labelled scenes, which was the aim of the experiment. Training the model with an accurately labelled and large dataset might allow the model to work in various environments and to recognize many more classes.

To follow up on the object detection, the semantically segmented points would need to be interpreted as separate objects. A method of detecting objects from the scene would be by finding large enough clusters of points with the same labels [3]. For example, breadth-first search could be used to search nearby points with the same label from a small area starting from a randomly picked point. If the cluster of points with same labels has enough points to be interpreted as a separate object, the bounding box is marked around that area. The detection score is computed as an average from the scores of each point in the category.

5.2 VoxelNet

VoxelNet is a 3D detection framework that predicts bounding boxes around the detected objects in point clouds [4]. The model learns feature representations from raw point clouds to predict the bounding boxes. An unofficial implementation of VoxelNet is experimented in this work [42]. The unofficial implementation is a modified version of the original paper's implementation with fixed bugs, support for data creation and more elaborate instructions for use.

5.2.1 Architecture

The VoxelNet has three functional blocks that form its architecture [4]. The model utilizes a feature learning network, convolutional middle layers and a region proposal network. With these blocks working together, VoxelNet can efficiently detect objects in sparse point clouds.

The input point clouds are partitioned into equally spaced voxels to form a voxel grid of the data [4]. The points are grouped to voxels based on the location they are in. This results in voxels with differing amounts of points. Each voxel is randomly sampled to have a same amount of points. This is done to save computational power and to decrease the imbalance between voxels with a different amount of points. The voxels' each dimension is 0.2m. Each voxel has a maximum of 35 points. If a voxel has less than 20 points, it is ignored. Because most of the voxels tend to be empty, removing them is a critical step for efficiency.

The feature learning network encodes the features of each voxel to use as inputs for the convolutional layers that learn the features [4]. The network calculates the input feature set for each point and transforms them into a feature space with a fully-connected network. The network has a linear layer, a batch normalization layer and a ReLU layer. Max pooling is used for all point-wise features. The transformed points are concatenated to form a feature representation of the voxel. An illustration of the feature learning network is presented in Figure 13.

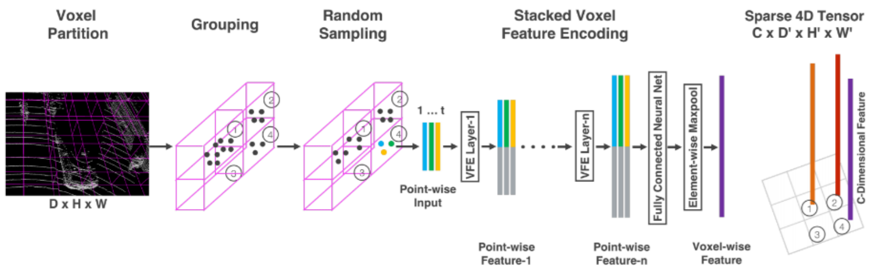

Figure 13. The VoxelNet feature learning network [4].

Modifying the points from raw point cloud to this representation helps in learning descriptive shape information. The modification is done with the help of point-wise features and locally aggregated features.

After encoding the features of voxels, the data is forwarded to the convolutional middle layers [4]. Each middle layer applies 3D convolution, batch normalization and ReLU. Voxel-wise features are aggregated to an expanding receptive field and the final output is given to the region proposal network. An illustration of the region proposal network is presented in Figure 14.

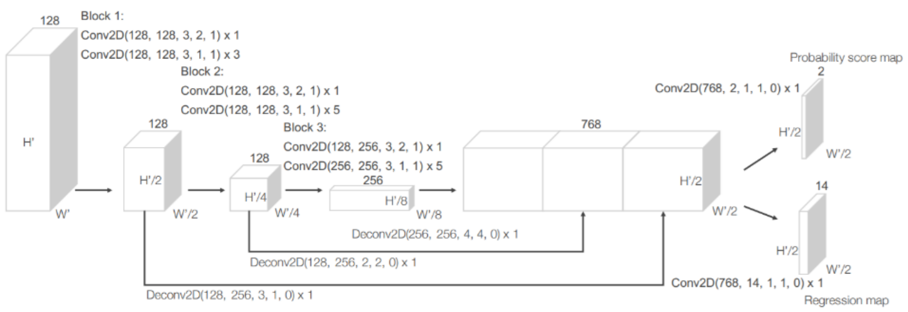
128
Block 1:
Conv2D(128, 128, 3, 2, 1) x 1
Conv2D(128, 128, 3, 1, 1) x 3
Block 2:
Conv2D(128, 128, 3, 2, 1) x 1
Conv2D(128, 128, 3, 1, 1) x 5
Block 3:
Conv2D(128, 256, 3, 2, 1) x 1
Conv2D(256, 256, 3, 1, 1) x 5
H'
W'
128
H'/2
W'/2
128
H'/4
W'/4
256
H'/8
W'/8
768
H'/2
W'/2
Deconv2D(256, 256, 4, 4, 0) x 1
Deconv2D(128, 256, 2, 2, 0) x 1
Deconv2D(128, 256, 3, 1, 0) x 1
Conv2D(768, 2, 1, 1, 0) x 1
Probability score map
2
H'/2
W'/2
14
H'/2
W'/2
Conv2D(768, 14, 1, 1, 0) x 1
Regression map

Figure 14. The VoxelNet region proposal network [4].

The numbers in brackets are the layers' sizes. *H'* and *W'* are the height and width of the voxel grid in number of voxels. The region proposal network has convolutional layers that downsample the feature maps also with added batch normalization and ReLu. The output is upsampled and the features are added together. Finally, the last feature map is mapped to the probability score map and a regression map. These can be interpreted as the detected objects in the scene.

5.2.2 Dataset and experiments

The implementation is evaluated on the KITTI 3D object detection benchmark in VoxelNet's paper [4]. The implementation's data processing is done with the characteristics of this dataset. With this in mind, a small experiment is done to create a dataset for use with the VoxelNet model for this work. The aim is to create data similar to the KITTI 3D dataset. To do this, a video and lidar scan recorded at the same time were combined to allow the labelling of data by hand. The data is from a recording of a ship leaving the port. The aim for the model is to detect the ship. A Python program is created for the labelling of data by hand. The libraries Numpy [39] and Open3D [40] are used for the array computations and working with point clouds.

First off, the camera and lidar data need to be calibrated for projecting the bounding box of the ship also to the image. This is done by finding ten correspondence points and solving the projection matrix. The point where the ship starts moving away from the port in both the video and point cloud is found by observing. From this point forward, a window of approximately fifteen seconds of point cloud points is extracted for labelling. The resulting point clouds have approximately fifteen to twenty thousand points each. Fifteen point clouds are extracted and labelled for the dataset.

The extracted smaller point clouds are shown to the user labelling the data. Points from around the ship are picked with the graphical interface of Open3D. An example pair of a point cloud and image of the dataset are presented in Figure 15.

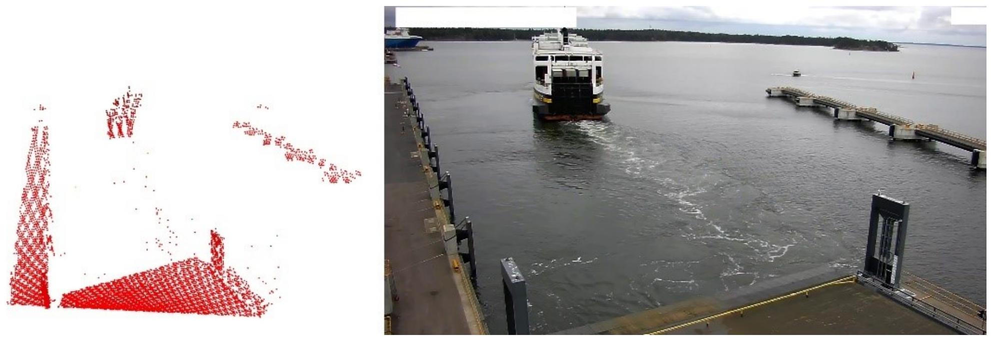

Figure 15. Example data from the dataset for VoxelNet

The lowest and highest valued points in each of the three axes are picked to get the bounding box of the ship. The program calculates the centre's location of the ship in addition to the ship's length, height and width in the 3D plane. The picked points are also projected to the 2D plane and the ship's bounding box is calculated. Lastly, the user is asked of the approximate rotation of the ship based on the camera frame.

The position and rotation data from the program are saved as a label. The corresponding label, image and extracted point cloud are named accordingly and saved in their own folders. This process is done multiple times to create a small dataset similar to the KITTI

3D dataset. However, correct labelling of the data is not enough. Changes in the implementation are needed because of the differences to the KITTI 3D dataset's car detection in comparison to this dataset's ship detection.

Multiple parts of the program of the unofficial implementation are altered to attempt to get the model working with a self-made dataset. The program has a large amount of set values that define the characteristics of the detected objects and voxels. The model is done to detect cars in streets and many assumptions are made in the program regarding that.

In the end, the model could not learn to detect the ship. Multiple values were changed to try to fit the ship's characteristics and the dataset was altered by for example rescaling the data to be smaller. Extra features, such as augmenting the data with noise and jittering, were also removed. After the feature encoding the data gets corrupted, and it is not completely sure if it is because of errors in the labelling or the difference of the dataset compared to the KITTI 3D dataset. The latter one seems more likely, as the labels and data seem very similar when comparing the two datasets.

5.3 Comparison results

The PointNet and VoxelNet architectures seem to be effective models for detecting objects in point clouds. Both architectures are deep convolutional neural networks that have multiple smaller blocks inside them. PointNet semantically segments point clouds. These segmented point clouds can be used for detecting objects by clustering the points. VoxelNet predicts the 3D bounding boxes of objects it detects.

The PointNet architecture has three important characteristics that allows it to semantically segment point clouds. A max pooling layer helps in working with the unordered point clouds. Local and global information of the points are combined to keep spatial information intact. Two joint alignment networks align the processed points. The point clouds are split into blocks with an area of 1x1m, each having 4096 points.

VoxelNet processes the point clouds to voxels. The voxels are boxes with sides of 0.2m and with 20 to 35 points each. The VoxelNet architecture also has three important characteristics. A fully-connected feature learning network learns the features of the voxels. Convolutional middle layers transform the feature learning network's coded voxels. The region proposal network outputs the 3D bounding boxes of the detected objects.

The two architectures had different datasets from maritime environments. Both datasets were created from similar point clouds. The PointNet dataset was created by cropping objects from the point clouds and labelling the separated objects. The separated objects are added back together to form labelled point cloud scenes. The final dataset had ten point cloud scenes. With the VoxelNet dataset the point clouds were not altered. Points around the detectable ship were picked to get the location and dimensions of the ship. The

VoxelNet dataset had fifteen labelled point cloud scenes. Both datasets have point clouds with twenty thousand points in average.

The PointNet model learned to semantically segment point cloud scenes with an 85% accuracy. However, the testing was done with highly similar examples as in the training set. With new data the network has not seen before the results will be significantly lower. The VoxelNet model did not learn to detect objects from the point cloud scenes. The model would need more adjustments to fit the problem given to it.

6. CONCLUSIONS

Using lidar-generated point clouds for detecting objects requires complicated deep learning architectures and processing of data. Creating training data for a detection model takes time and requires specific tools for visualization. The model for learning point cloud features can have dozens of layers. The point cloud is drastically altered for efficiency and to keep the spatial information intact.

The PointNet and VoxelNet architectures demonstrated two ways of detecting objects in a scene. PointNet attempts to semantically segment each point in the data by learning the local and global features of the points and classifying each of them. Clusters of points with the same labels can be detected as objects. VoxelNet similarly learns the features of the points using voxels, 3D boxes of points, and learning their properties. Combining a convolutional network and region proposal network, VoxelNet can predict 3D bounding boxes of the detected objects.

Experiments with the two models demonstrated some challenges of using neural networks in general. Data for an own dataset is hard to come by and creating the dataset take time. Models need precisely the right type of data and their parameters need tuning for them to work. The PointNet model reached around 85% accuracy on its test set, but it had a small amount of training data of seven 3D scenes. This means the model would be bad at generalizing what it has learned, and it would not be accurate with other data given to it. The VoxelNet model did not work in the end, but creating a dataset to it was also an important challenge typical to neural networks.

To get the best results for the problems that were experimented in the work, the models and their parameters would need to be altered to fit the data and aims. The training dataset would need thousands or more examples instead of only a few. Using a different model with a highly different architecture might give more accurate results in detecting objects at sea. One problem with using convolutional neural networks for the problem is the sparseness of the point cloud data. Vessels or other objects that are hundreds or thousands of meters away are hardly detectable with lidar scanners. Then again, near the coast and ports, there are many objects to be detected in close range.

Deep learning with 3D data has not been researched nearly as much as with images. The one additional dimension can tell a lot more about the observed area. It can give more accurate results in scenarios where the depth of the scene is essential. Traffic in land and sea can benefit from knowing the distance to objects the vehicle has detected. Future advances in 3D imaging and neural networks can make methods more efficient and easier to use. This could introduce 3D detection and classification to different fields, from industrial manufacturing to medical examinations.

REFERENCES

[1] R. Zakharenko, Self-driving cars will change cities, Regional Science and Urban Economics, Vol. 61, 2016, pp. 26-37. https://www-sciencedirect-com.libproxy.tut.fi/science/article/pii/S016604621630182X.

[2] J. Walker, Autonomous Ships Timeline - Comparing Rolls-Royce, Kongsberg, Yara and More, TechEmergence, 2018.

[3] C.R. Qi, H. Su, K. Mo, L.J. Guibas, PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation, arXiv:1612.00593, 2016. https://arxiv.org/abs/1612.00593.

[4] Y. Zhou, O. Tuzel, VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection, arXiv:1711.06396, 2017. http://arxiv.org/abs/1711.06396.

[5] Lidar 101: An Introduction to Lidar Technology, Data, and Applications, in: Revised, NOAA Coastal Services Center, Charleston, SC, 2012.

[6] P. Babahajiani, L. Fan, J. Kämäräinen, M. Gabbouj, Urban 3D segmentation and modelling from street view images and LiDAR point clouds, Machine Vision and Applications, Vol. 28, Iss. 7, 2017, pp. 679-694.

[7] S. Veerabuthiran, Exploring the atmosphere with lidars, Resonance, Vol. 8, Iss. 4, 2003, pp. 33-43. https://link-springer-com.libproxy.tut.fi/article/10.1007/BF02883530.

[8] J. Li, H. Bao, X. Han, F. Pan, W. Pan, F. Zhang, D. Wang, Real-time self-driving car navigation and obstacle avoidance using mobile 3D laser scanner and GNSS, Multimedia Tools and Applications, Vol. 76, Iss. 21, 2017, pp. 23017-23039.

[9] R. Amadeo, Neato Botvac Connected review: A LIDAR-powered robot vacuum is my maid now, Ars Technica, 2018. https://arstechnica.com/gadgets/2016/01/neato-botvac-connected-review-a-lidar-powered-robot-vacuum-is-my-maid-now/.

[10] J. Shackleton, B. VanVoorst, J. Hesch, Tracking People with a 360-Degree Lidar, Advanced Video and Signal Based Surveillance, 29 Aug.-1 Sep. 2010, IEEE, pp. 420-426.

[11] K. Simek, The Perspective Camera - An Interactive Tour, GitHub, 2012.

[12] H. Hu, G. Williams & J. Hays, Project 3: Camera Calibration and Fundamental Matrix Estimation with RANSAC, 2018. University assignment, Georgia Institute of Technology.

[13] F. Amato, A. López, E.M. Peña-Méndez, P. Vaňhara, A. Hampl, J. Havel, Artificial neural networks in medical diagnosis, Journal of Applied Biomedicine, Vol. 11, Iss. 2, 2013, pp. 47-58.

[14] E. Roberts, Neural Networks, 2000. https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/index.html.

[15] A. Beam, Deep Learning 101 - Part 1: History and Background, 2017. https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html.

[16] M.W. Gardner, S.R. Dorling, Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences, Atmospheric Environment, Vol. 32, Iss. 14-15, 1998, pp. 2627-2636.

[17] M. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

[18] X. Glotor, A. Border, Y. Bengio, Deep Sparce Rectifier Neural Networks, Journal of Machine Learning Research, 2011, Florida, USA, pp. 315-323.

[19] S.W. Smith, Digital Signal Processing, A Practical Guide for Engineers and Scientists, Chapter 6, Elsevier Inc., 2002, 107-122 p.

[20] S.W. Smith, Digital Signal Processing, A Practical Guide for Engineers and Scientists, Chapter 24, Elsevier Inc., 2002, 397-422 p.

[21] O. Abdel-Hamid, L. Deng, D. Yu, Exploring convolutional neural network structures and optimization techniques for speech recognition, Interspeech, Vol. 2013, 2013, pp. 3366-3370.

[22] Y. Bai, S.S. Bhattacharyya, A.P. Happonen, H. Huttunen, Elastic Neural Networks: A Scalable Framework for Embedded Computer Vision, arXiv:1807.00453, 2018. https://arxiv.org/abs/1807.00453.

[23] D. Barber, J. Mills, S. Sarah, Geometric validation of a ground-based mobile laser scanning system, ISPRS Journal of Photogrammetry and Remote Sensing, Vol. 63, Iss. 1, 2008, pp. 128-141.

[24] J. Bjorck, C. Gomes, B. Selman, Understanding Batch Normalization, arXiv:1806.02375, 2018. http://arxiv.org/abs/1806.02375.

[25] A. Ouaknine, Review of Deep Learning Algorithms for Object Detection, Medium, 2018.

[26] A. Krizhevsky, I. Sutskever, G. Hinton, ImageNet classification with deep convolutional neural networks, Communications of the ACM, Vol. 60, Iss. 6, 2017, pp. 84-90.

[27] B. Cyganek, Object Detection and Recognition in Digital Images: Theory and Practice, Wiley, Somerset, UNITED KINGDOM, 2013.

[28] E. Shelhamer, J. Long, T. Darrell, Fully Convolutional Networks for Semantic Segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 39, Iss. 4, 2017, pp. 640-651.

[29] A.K. Jain, J. Mao, K.M. Mohiuddin, Artificial neural networks: a tutorial, Computer, Vol. 29, Iss. 3, 1996, pp. 31-44.

[30] Stochastic Gradient Descent - Mini-batch and more, Adventures in Machine Learning, 2017.

[31] L. Bottou, Large-Scale Machine Learning with Stochastic Gradient Descent, in: Anonymous (ed.), Proceedings of COMPSTAT'2010, Physica-Verlag HD, Heidelberg, 2010, pp. 177-186.

[32] D.P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization, Conference paper at ICLR 2015.

[33] J. Brownlee, What is the Difference Between Test and Validation Datasets?, Machine Learning Mastery, 2017.

[34] R. Kohavi, A Study of Cross-Validation and Bootstrap
for Accuracy Estimation and Model Selection, Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, Canada, pp. 1137-1143.

[35] D.M. Hawkins, The Problem of Overfitting, Journal of Chemical Information and Computer Sciences, Vol. 44, Iss. 1, 2004, pp. 1-12.

[36] A.P. Piotrowski, J.J. Napiorkowski, A comparison of methods to avoid overfitting in neural networks training in the case of catchment runoff modelling, Journal of Hydrology, Vol. 476, 2013, pp. 97-111.

[37] A. Bhandle, What is underfitting and overfitting in machine learning and how to deal with it, Medium, 2018.

[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from
Overfitting, Journal of Machine Learning Research, Iss. 15, 2014, pp. 1929-1958.

[39] NumPy, Python package, 2018. http://www.numpy.org/

[40] Q. Zhou, J. Park, V. Koltun, Open3D: A Modern Library for 3D Data Processing, arXiv:1801.09847, 2018. http://arxiv.org/abs/1801.09847.

[41] A. Collette, HDF5 for Python, Python package, 2018. http://www.h5py.org/

[42] Q. Huang, voxelnet, GitHub project, 2018. http://github.com/qianguih/voxelnet