TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

Liine Kasak 179629IAIB

USING CONVOLUTIONAL NEURAL NETWORKS FOR OBJECT DETECTION AND CLASSIFICATION IN POINT CLOUDS

Bachelor’s thesis

Supervisor:	René Pihlak
	MSc
Co-Supervisor:	Andri Riid
	PhD

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Liine Kasak 179629IAIB

PUNKTIPILVE OBJEKTIDE TUVASTAMINE JA KLASSIFITSEERIMINE KONVOLUTSIOONILISTE NÄRVIVÕRKUDEGA

Bakalaureusetöö

Juhendaja:	René Pihlak
	MSc
Kaasjuhendaja:	Andri Riid
	PhD

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Liine Kasak

17.05.2020

Abstract

The spatial data from 3-dimensional point clouds have great potential for use in various tasks, such as autonomous vehicles or virtual maps of cities, once the objects in these clouds are classified.

In this thesis, convolutional neural networks are constructed to classify point cloud objects of classes posts, vegetation, buildings and cars. Two scene classification methods are also constructed: a sliding window method, which classifies the scene by segment, and a method which clusters the scene into objects before classifying the objects. The sliding window method lays a grid over the initial point cloud, slides over this grid with a cube with certain dimensions and assigns the underlying points the class of the predicted value of the cube. The clustering method involves usage of existing algorithms for point cloud clustering. Segments are generated from the clustered objects one by one, and the points of each object are assigned based on a majority vote over all segments of an object.

Two methods for analysing the networks' performances are proposed. One method is segment-wise classification. A list of all segments from all objects is generated. These segments are classified one by one. The second method involves object-wise classification. This method iterates over all objects, generates segments for the object and applies all segments a class based on the majority vote over all segments of the object.

The proposed network achieved an accuracy of 83.7% in segment-wise classification and 99.2% in object-wise classification.

The second method achieved higher accuracy and thus is the proposed method.

This thesis is written in English and is 36 pages long, including 6 chapters, 28 figures and 7 tables.

Annotatsioon

PUNKTIPILVE OBJEKTIDE TUVASTAMINE JA KLASSIFITSEERIMINE KONVOLUTSIOONILISTE NÄRVIVÕRKUDEGA

Kolme-dimensioonilistel punktipilvede ruumilistel andmetel on potentsiaalseid kasutusvõimalusi mitmetes valdkondades, nagu isesõitvad sõidukid või linna digitaliseerimine, eeldusel, et punktipilvede objektid on klassifitseeritud.

Antud töös kasutatakse konvolutsioonilisi närvivõrke, et klassifitseerida kolme-dimensioonilisi objekte postideks, vegetatsiooniks, majadeks ja autodeks. Konstrueeriti kolm erinevat mudelit. Esialgselt lahendati probleem vaid kolme klassi leidmiseks: postid, vegetatsioon ja majad. Probleemi laiendamisel lisati ka autode klass juurde ning koostati nelja klassi klassifitseerija – sellest sai teine mudel. Kolmas mudel koostati neljast erinevast binaarsest klassifitseerijast. Iga klassi jaoks treeniti üks närvivõrk ning konstrueeriti meetod, mis klassifitseerib objekti nelja klassi kasutased binaarseid klassifitseerijaid. Võrreldes teist ja kolmandat mudelit, mis on nelja klassi klassifitseerijaid, olid paremad tulemused neist esimesel.

Stseeni klassifitseerimist teostati kahe meetodiga: libiseva akna meetod ja klasterdatud objektide klassifitseerimise meetod. Libiseva akna meetod hõlmab kindla suurusega kuubiku libistamist üle punktipilve ruudustiku. Iga kuubiku hinnang määrab allolevate punktipilve punktide väärtuse. Klasterdamise meetodis kasutatakse olemasolevaid punktipilve klasterdamise algoritme. Klasterdamise tulemusel tekkinud objektide järjendist itereeriti üle ning iga objekti punktidele määrati klass, mis on kõige populaarsem objekti segmentide hinnangute seas.

Klassifitseerijate hindamiseks katsetati kahte meetodit. Esimese meetodi puhul hinnati närvivõrkude täpsust segmentide põhjal. See tähendab, et esmalt koostati järjend kõikide objektide segmentides ning seejärel neid segmente hinnati ühekaupa. Teise meetodi puhul hinnati närvivõrkude täpsust objektide põhiselt. Objektidest itereeriti üle, genereerides iga objekti jaoks segmendid. Kõikidele segmentidele määrati ühene klass vastavalt enamushääletusele. Närvivõrk saavutas täpsuse 83,7% segmendipõhise klassifitseerimisega, 99,2% objektipõhise klassifitseerimisega.

Visuaalselt olid paremad tulemused klasterdamisel ning seda toetab ka kõrgem täpsus objektipõhisel klassifitseerimisel. Seega pakutud lahendus nelja klassi probleemile on teine närvivõrgu mudel koos klasterdamise meetodiga.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 36 leheküljel, 6 peatükki, 28 joonist, 7 tabelit.

List of abbreviations and terms

2D	2-dimensional
3D	3-dimensional
API	Application Programming Interface
CNN	Convolutional Neural Network
FN	False Negatives
FP	False Positives
GIS	Geographic Information System
GPU	Graphics Processing Unit
I/O	Input/output
Lidar	Light Detection and Ranging
PCD	Point Cloud Data
PCL	Point Cloud Library
ResNet	Residual Network
TN	True Negatives
TP	True Positives
VGG Net	Visual Geometry Group Network
voxel	Volumetric pixel

Table of contents

List of Figures

List of Tables

1 Introduction

Light Detection and Ranging (Lidar) technology is widely used for acquiring point cloud data due to its high precision, long range and invariance to lighting conditions [1]. This data has great potential to be used for autonomous vehicles and for mapping cities [2]. Several point clouds have been obtained from scenes in Tallinn using Lidar technology for the purpose of 3-dimensional (3D) object detection and classification.

Manual point by point classification of 3D point clouds is labour expensive and time consuming as each scene can contain millions of points. Thus, machine learning can be applied to the problem of object classification in large point cloud scenes. Convolutional Neural Networks (CNNs) have successfully been applied to the problem of classifying 2-dimensional (2D) images. This motivates the use of CNNs for the 3D classification problem as well.

The goal of this work is to construct a method to classify posts, buildings, vegetation and cars in point cloud scenes. This goal is divided into the following subtasks: 1) generate datasets annotated by class, 2) to build, train and analyse the CNN model, 3) to construct different approaches to classify objects in point cloud scenes, and 4) to validate the results on point cloud scenes. As a result of this work, a new method of object classification for point cloud scenes has been developed.

This paper consists of six sections. The introduction composes the first part. In the second part, the theoretical background is introduced. An overview of literature of related works is given in the third part. The fourth part is the core of the paper: the process of data generation and preparation is explained, the designed CNN models are introduced, and methods for scene classification are described. Results of the trained network models and scene classification methods are analysed in the fifth part. Lastly, a summary is presented in the sixth part.

2 Background

In this section the definitions, methods and technologies used in the thesis are explained to aid comprehension of the proposed method.

2.1 Dataset type

This section discusses the technology behind the dataset type and the required augmentation of data for 3D machine learning.

2.1.1 Lidar

Lidar is a 3D scanner which measures the distance to a target by sending out laser light and measuring the reflected light with a sensor. 3D representations of scenes are constructed using the information of wavelength and differences in laser return times. [2]

Lidar technology has many applications: agricultural purpose to determine farmland yield results to determine where to apply fertilizer, uses in archaeology to produce high-resolution datasets quickly and cheaply and easily integrate into a Geographic Information System (GIS), usage in autonomous vehicles for obstacle detection and avoidance, and so on. [2]

2.1.2 Point cloud data

A point cloud is a set of data points usually in 3D space. These datasets are generally produced by 3D scanners. [3]

The Point Cloud Library (PCL), a library for 2D/3D image and point cloud processing, commonly uses the Point Cloud Data (PCD) file format for input/output (I/O) operations when manipulating point cloud data. The possible fields to contain in the file for each point are XYZ coordinates, RGB values, XYZ normals. Other commonly used file formats include PLY (a polygon file format), STL (a file format native to CAD software), OBJ (a geometry definition file format), X3D (the ISO standard XML-based file format used for 3D computer graphics data), and many others. The PCD file format has many advantages: the ability to store and process organized point cloud datasets, the usage of binary data types to speed up file saving and loading, efficient storage as different data types are allowed, the support of n-D histograms for feature descriptors. [4]

2.1.3 Data representation for machine learning

For 2D machine learning, images are usually converted into arrays depicting a grid where each pixel is represented by their RGB colour model. Similarly, point clouds are usually converted into 3D grids for machine learning.

In this approach, the point cloud is contained within a bounding box, which is thereafter cut into voxels (volumetric pixels) of certain size. Each voxel is represented by a number depending on the points contained in the voxel. The simpler notion is to represent the voxel with a zero or one depending on if any points are inside the voxel. This data format is normally called a voxel, occupancy or density grid, referencing the grid-like structure, or a segment, referencing the fact that it could be a part of a larger point cloud scene.

2.2 Classification

In machine learning, **binary classification** is the problem on classifying instances into one of two classes accordingly. The problem of classifying instances into one of three or more classes is called **multiclass or multinomial classification**. [5]

Classification is considered a type of supervised learning where the labels are provided with the input data. Classification is the process of predicting the class of given data points. [6] The algorithm which implements classification is called a **classifier**. The performance of a classifier depends on the characteristics of the dataset. To determine a suitable classifier, classifiers are evaluated using different metrics: precision, recall, and many others. [7]

Confusion matrix is a table of predicted values where each row represents a class X, each column represents class Y, and the table cell values shows how many elements of class X where predicted to be in class Y. A confusion matrix is helpful for easily calculating values for precision, accuracy, recall and F-1 Score. [8]

An example of a confusion matrix is given in Figure 1. The number of instances from class one which are predicted as class one is called True Positives (TP). The number of instances from class one which are predicted as class zero is called False Negatives (FN). The number of instances from class zero which are predicted as class one is called False Positives (FP). The number of instances from class zero which are predicted as class zero is called True Negatives (TN). These abbreviations are used to describe precision (12), recall (2), accuracy (3) and F1 score (4) calculations for a single class.

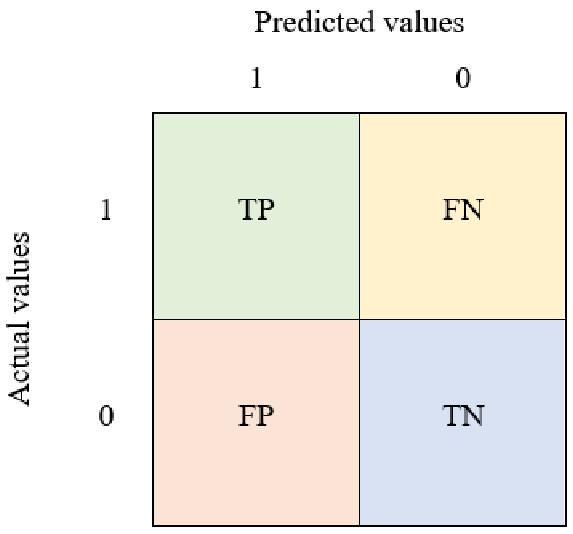

Figure 1. Example of a confusion matrix applied to a binary classification problem.

For multi-class classification, the confusion matrix to determine TP, FP, FN and TN for class one is as presented in Figure 2.

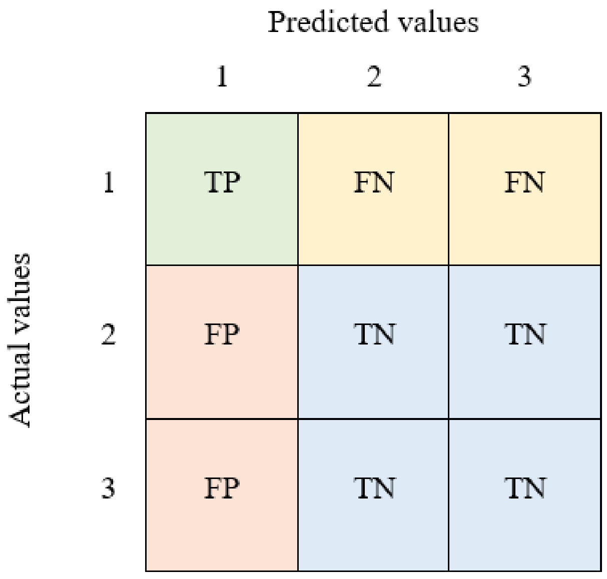

Figure 2. Example of a confusion matrix for a three-class problem, where TP, FP, FN and TN are described for class one.

Precision (12) describes the proportion of positive identifications which were correct. [8]

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Recall (2) is also known as sensitivity and describes the proportion of correctly guessed actual positives identified correctly. [8]

$$Recall\ = \frac{TP}{TP + TN} \tag{2}$$

Accuracy (3) determines how many elements were classified correctly out of all classifications. [8]

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

The **F1 Score** (4) is the harmonic mean of precision and recall [9]. It determines how many instances were classified correctly without missing a significant number of instances. [8]

$$F1\ score = 2\frac{Precision\ \cdot Recall}{Precision + Recall} \tag{4}$$

When generalising these metrics for all classes there are two approaches: average or weighted metrics. Average metrics calculations involve summing the metrics amongst the classes and dividing the result by the number of classes. In weighted metrics, the metrics are multiplied by the number of instances in that class and then the result is divided by the size of the dataset. In the equations for average and weighted precision (5)(8), recall (6)(9) and F1-Score (7)(10), *n* is the number of classes, *size*$_c$ is the number of instances in class *c* and $size = \sum_c^n size_c$.

$$Precision_{average} = \left(\sum_c^n Precision_c\right)/n \tag{5}$$

$$Recall_{average} = \left(\sum_c^n Recall_c\right)/n \tag{6}$$

$$F1score_{average} = \left(\sum_c^n F1score_c\right)/n \tag{7}$$

$$Precision_{weighted} = \left(\sum_c^n Precision_c \cdot size_c\right)/size \tag{8}$$

$$Recall_{weighted} = \left(\sum_c^n Recall_c \cdot size_c\right)/size \tag{9}$$

$$F1score_{weighted} = \left(\sum_{c}^{n} F1score_c \cdot size_c \right) / size \qquad (10)$$

2.3 Neural Networks

Neural networks are one of the most commonly used machine learning methods where the user defines the input and output structure, and the network model. Neural networks are multi-layer networks of neurons used to classify data and make predictions. [10]

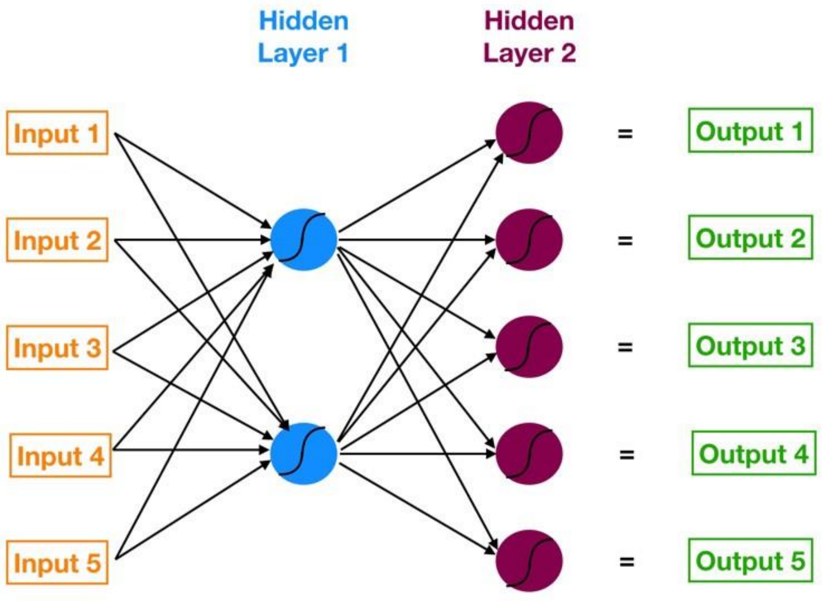

Figure 3. Example neural network with two hidden layers. [10]

Deep neural networks contain multiple hidden layers, where the output of each neuron is expressed by the equation (11) where *W* denotes weight and *In* denotes input. The output of a neuron is calculated using an activation function which is usually nonlinear [10].

$$Z1 = activation(W_1 \cdot In_1 + W_2 \cdot In_2 + W_3 \cdot In_3 + W_4 \cdot In_4 + W_5 \cdot In_5 + Bias_{Neuron1}) \qquad (11)$$

An example of an activation function is ReLU (rectified linear unit) $f(x) = \max(0, x)$ [11]. Another activation function used in this work is Softmax (12), where $x \in \{x_1, x_2, x_3, \dots, x_n\}$ and $i \in \{1, 2, 3 \dots n\}$.

$$f(x) = \frac{e^x}{\sum_i e^{x_i}} \tag{12}$$

Generalising the output of one neuron, the general formula for the output of one layer, where the prior layer is m elements deep and the current layer is n elements deep, is presented in equation (13), where *[W]* is the n by m matrix of weights, *[X]* is the m by one matrix of inputs, *[Bias]* is the n by one matrix of neutron biases, *[Z]* is the n by one matrix of outputs, and @ denotes matrix multiplication. [10]

$$[W]@[X] + [Bias] = [Z] \tag{13}$$

For the training process, the user must define a cost function and use gradient descent optimisation to minimise it. [10]

2.3.1 Convolutional Neural Networks

A CNN is a Deep Learning algorithm which can assign importance to various aspects of the input data and be able to differentiate one from the other. CNN requires less pre-processing than other classification algorithms as CNNs can learn filters or characteristics that in primitive methods would be hand-engineered. [12]

Compared to a feed-forward architecture, a CNN can capture spatial and temporal dependencies in images or other multidimensional grids through the application of relevant filters. CNNs are better fitting for this type of data due to the reduction in the number of parameters involved and reusability of weights. [12]

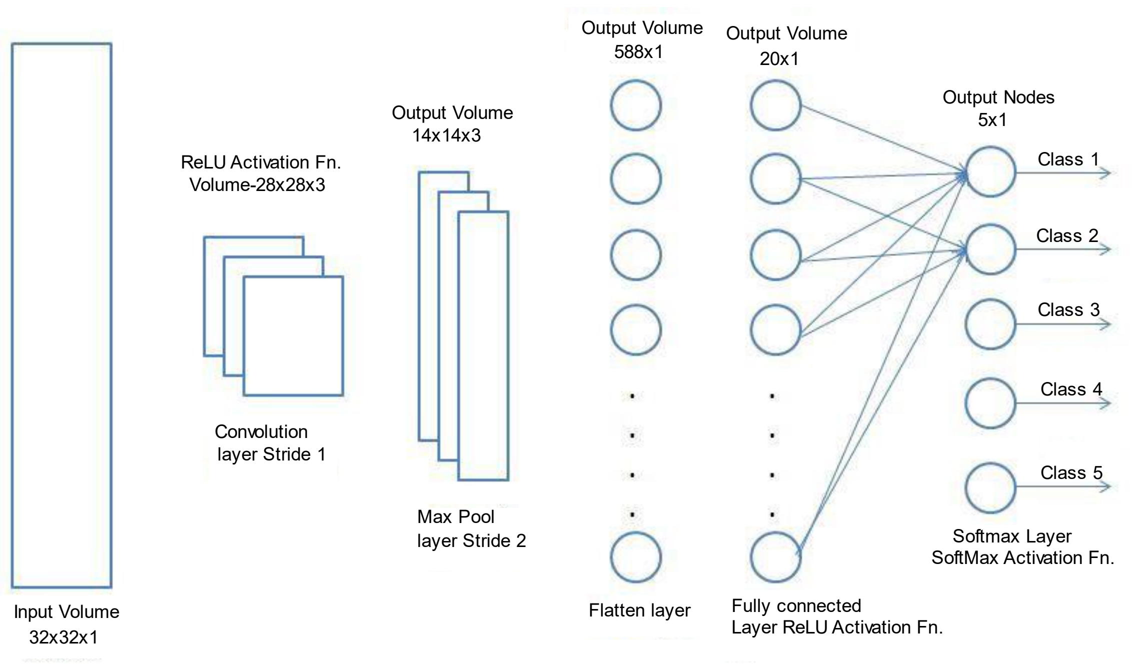

Figure 4. Example CNN architecture.

The element carrying out the convolutional operation in the first part of a **convolutional layer** is called a kernel or filter which has a dimensionality of $d - 1$, where d is the dimensionality of the input data. The kernel “slides” over the input with a certain stride value and applies the filter on the underlying values. The objective of the convolution operation is to extract high-level feature maps from the grid. [12]

Like the convolutional layer, the **pooling layer** reduces the spatial size of the convolved feature to decrease the computational power required to process the data. The pooling layer extracts dominant features which are rotational and positional invariant. **Max pooling** applies a calculation which returns the maximum value of the portion of the grid covered. **Average pooling** returns the average value of the portion. [12]

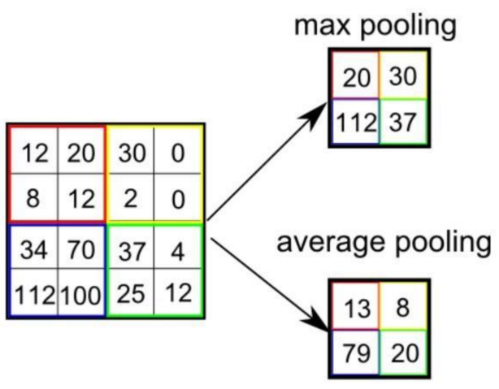

Figure 5. Max and average pooling. [12]

A **fully connected layer** learns non-linear combinations of the high-level features as represented by previous layers. To classify input data, usually a **softmax layer** is used at the end of the model.

2.3.2 Training process

The training of a neural network is a process during which the parameters are modified to improve the results of classification. Training is done using an iterative training algorithm where the input data and output shape are given.

To estimate the error of the network and measure how good or bad the results are, a **loss function** is used. If the model succeeds in learning to classify data, the loss function value should decrease during the training process as the weights of neurons are adjusted. Usually, a model is trained until the loss function does not significantly change anymore.

In classification tasks, the **cross-entropy** loss is commonly used when a probabilistic interpretation of the scores is desired. [13] The loss function evaluates as (14), where *M* is the number of classes, $y_{o,c}$ is a binary indicator if the class label *c* is the current observation *o*, and $p_{o,c}$ is the predicted probability that *o* is of class *c*. [14] The dissimilarity between the true label distribution *y* and the predicted label distribution *p* is measured. [13]

$$loss_{cross\_entropy}(y,p) = -\sum_{c=1}^{M} y_{o,c} \ln\left(p_{o,c}\right), \tag{14}$$

The purpose of training a model is that it would also be able to classify data which it had not previously seen yet. This is achieved if the network succeeds at generalising features. To test the networks ability to generalise, its performance on the test dataset is analysed.

The input data can be presented to the network during its training iterations either in samples, batches or the whole dataset if the memory allows it. A sample is a single row of data. The **batch** size is a parameter which defines the number of samples to process before updating model parameters. During one **epoch** all instances in the dataset are processed.

2.3.3 Popular convolutional neural network architectures

Visual Geometry Group Network (VGG Net) is a CNN architecture which may look simple but outperforms many complex architectures. The idea is that every succeeding layer has more filters than the preceding layer, and only 3x3 kernels are used as 5x5 kernels are more expensive and two 3x3 kernels can almost cover what one 5x5 kernel covers. The most popular VGG Nets are VGG-16 and VGG-19, where the number of layers with weights are accordingly 16 and 19. [15] [16]

VGG Nets are not suitable for deeper networks as they are prone to the vanishing gradient problem. [15] This means that in some cases the gradient is vanishingly small and prevents the weight from changing its value. [17] VGG Networks are useful for small classification tasks and transfer learning. [15]

Residual Network (ResNet) is a deep network which has residual connections. Unlike VGG Nets where every layer is connected to the previous layer, every layer input is the results of concatenation of the previous layer and the one before the previous layer. This way each layer can see more features than just from the previous layer. This architecture is possible by using batch normalisation layers after every convolutional layer. [15] Batch normalization is the process of normalising the output of a layer by subtracting the batch mean and dividing by the batch standard deviation. [18] These layers will enable the usage of higher learning rates while training and hence help train faster and minimise the vanishing gradient problem. [15]

In **DenseNet** every layer is connected to the all previous layers in the same block. This means that less filters are required. This also helps towards the vanishing gradient problem. [15]

The “inception” in **Inception Net** suggests going “deeper” in the network. While ResNets are also deep networks, the idea of Inception Net is to go wider as well. This is done by concatenating several parallel layers having different filters in every inception block. Inception Nets have less parameters to train compared to other architectures. [15]

Xception Net means Extreme Inception. The Xception architecture aims to be computationally more efficient than Inception Nets. This is achieved by replacing convolutional operations with depth wise separable convolutional operations. A normal convolution operates several filters over each feature map of the input and sums the corresponding values. In depth-wise separable convolutions the computational complexity is reduced as every kernel is of two dimensions only and convolves only over one feature map. [15]

2.4 Implemented technologies

In this work, programming language Python 3 and C++ are used. The TensorFlow library is used for training the network. PCL tools are used to perform various operations on point cloud files. The Cloth Simulation Filter (CSF) library [19] is used as an alternative method of ground removal. The Open3D library is used for I/O operations on PCD files in Python.

2.4.1 Point Cloud Library

PCL is a project for 2D/3D image and point cloud processing. The library contains algorithms for feature estimation, surface reconstruction, model fitting and segmentation.

The PCL **Progressive Morphological Filter** is a filter used for detecting nonground LIDAR measurements such as buildings, vehicles and vegetation. The filter works by gradually increasing the window size of the filter and using elevation difference threshold to remove nonground data and preserve the ground data points. [20] Using the PCL library, it is possible to write a script in C++ to use the filter and extract non-ground points as well.

The PCL **Euclidean Cluster Extraction** is a clustering method which divides an unorganised point cloud model into smaller parts. [21]

The PCL **Visualizer** is used in this thesis to view the point cloud data and evaluate classification.

2.4.2 Cloth Simulation Filter

An alternative method of ground removal is the CSF method. This approach separates the ground and non-ground objects with the following steps: 1) inverts the point cloud over the *xy*-plane, 2) drops a cloth onto the terrain, 3) estimates ground points using the cloths shape. This work stands out with its unique approach and the fact that it does not require too many parameters, as do other solutions. However, this method yields high errors when the terrain is disconnected. [19] Our version of the cloth simulation filter, where it is possible to read a PCD file and after applying the CSF save ground and non-ground PCD files, is presented in [22].

2.4.3 Open3D

Open3D is a library which supports 3D data structures and algorithms in C++ and Python. [23] In this work, the library is used in Python code for point cloud read and write operations and modifying PCD files to contain colour values according to classification.

2.4.4 TensorFlow and Keras

TensorFlow is and open source library for developing and training machine learning models. [24] TensorFlow implements the Keras Application Programming Interface (API) specification. Keras is a high-level neural networks API. Keras allows easy and fast prototyping of neural networks, so the user does not have to focus on mathematical details. [25] The user must define a model, add layers, such as a convolution, pooling or dense layer, and define the input and output data shape. In contrast to the high-level Keras API, the low-level TensorFlow Core API requires working with TensorFlow computational graphs, tensors, operations, and sessions. This can be a lot harder to understand. [26]

In the context of TensorFlow, tensors are multidimensional arrays. The library does differentiable operations across the tensors. It also allows computations on Graphics Processing Units (GPUs) for computers with GPUs which support CUDA. CUDA is a parallel computing platform and API model which allows the usage of GPUs for general purpose data processing. [27]

3 Related works

This section discusses works related to classifying objects in 3D point cloud scenes. These works differ in used data types, CNN input shape, output shape and the scene classification method. For an overview of related works see Table 1.

3.1 Convolutional neural network input type

Usually solutions to convert point clouds into a suitable CNN input involve either generating images of the point cloud under different angles [28], [29], or transforming the point cloud into a voxel or occupancy grid [30], [31], [32], [33].

In the **multi-view** approach, the 3D problem is reduced to multiple 2D problems, where the images are in grayscale according to the number of points in a pixel. This is demonstrated in the work of Huang and You [28] where they compute three views of each object and classify them in a trained CNN with orthogonal view projections as input. The orthogonal view projection consists of three images: the XY-plane, XZ-plane and YZ-plane. Pang *et al* [29] also use multiple 2D images of an object as the CNN input. These images are evenly chosen on a sphere. This method is chosen since CNNs synergise well with it as a lot of training data will be accumulated with the projection.

The more popular approach is to represent point clouds as **voxel grids**. This is preferred, as important data, such as structural information, is lost when applying 2D techniques on 3D [30]. Unfortunately using voxel grids are largely redundant due to the sparsity of point clouds. Xiang *et al* [31] implement an octree-based CNN to address this issue. The construction of the octree-based structure starts by enclosing the point cloud in a bounding box, known as the root node of the tree. Then, the octree nodes are recursively subdivided into eight equal-sized child nodes until reaching the finest resolution of the tree.

3.2 Scene classification

Classification on a larger point cloud scene is done using the sliding window method with voting or pre-classification segmentation of the point cloud via clustering. Both methods have their advantages and disadvantages.

The first approach, the **sliding window method**, consists of multiple steps. First, the scene is transformed into a voxel grid. Then sub-grids with CNN input shape are generated by sliding over the initial voxel grid. Lastly, the network is used to predict the results of the grids and the class of each point in the original point cloud scene is determined by voting. The advantage to this is that objects near each other can be distinguished. Unfortunately, as a result of this method, there are objects where most points are correctly classified, but some points inside it are misclassified. The edges of objects are usually also tougher to classify.

The second approach consists of point cloud scene segmentation with **clustering**. Every clustered object is classified by a majority vote. The determined class is applied to every point in the clustered object. The benefit of this is that each object only contains points of one class. This avoids the misclassification of object edges. Depending on the minimal distance between clusters, two objects of different classes may be clustered together, and thus classified as one object.

As an exception, Xiang *et al* [31] do scene classification in a way where the segmentation is done by the CNN. This means that the output has the same shape as the input, where voxels are annotated with various classes. Maturana and Scherer [33] and Qi *et al* [34] do not describe a scene classification method.

Table 1. Comparison of related works.

Approach	**Data type**	**CNN input**	**CNN output**	**Scene classification method**
[35]	RGBD	2D+3D	Label	Segmentation
[28]	3D point cloud	2D	Label	Segmentation
[30]	3D point cloud	Voxel grid	Label	Sliding window
[31]	3D point cloud	Voxel grid	Voxel grid	Segmentation by CNN
[29]	3D point cloud	2D	Label	Segmentation
[32]	3D point cloud	Voxel grid	Label	Sliding window
[33]	3D point cloud	Voxel grid	Label	-
[34]	3D point cloud	Set of points	Label	-

4 Proposed method

This section discusses the voxelization method, the constructed CNN and scene classification methods. According to the drawbacks of the first constructed neural network, new models were created.

4.1 Dataset

This study was partially supported by the Archimedes Foundation and Reach-U Ltd. in the scope of the smart specialization research and development project #LEP19022: “Applied research for creating a cost-effective interchangeable 3D spatial data infrastructure with survey-grade accuracy”. In the scope of this project, a dataset of point clouds was acquired using the Lidar technology. Specifically, the Velodyne Puck Lidar was used. This product has a range of 100 m [36]. In October 2018 and June 2019 approximately 250 files of point cloud data were obtained with the Velodyne Lidar, of which 25 files were annotated to create point cloud objects to be used in this work.

To construct the dataset, the point cloud data was processed in several steps. The ground was removed from the original point clouds using PCL Progressive Morphological Filter. After clustering with PCL Euclidean Cluster Extraction, the resulting objects were separated into the desired classes. The initial dataset consists of objects of three classes: posts, vegetation, building. During the research, the dataset was augmented by adding new data and creating a separate class for cars.

When converting point clouds to voxel grids the size of one voxel is $10 \times 10 \times 15$ cm, where the length 15 cm is along the z axis. The chosen voxel grid shape is (32, 32, 32), thus the size is $3.2 \times 3.2 \times 4.8$ m. The size of the dataset and its distribution between classes is shown in Table 2. Since the size of objects differ, the number of voxel grids generated for class is also given. The dataset is divided into train, validation and test datasets. The class balance was kept in mind when creating the train and validation datasets from the initial dataset. For the second dataset, class weights are initialised, so the balancing of the dataset was not required. Both datasets are used in the constructed CNNs.

Table 2. Initial dataset description.

Class	**Objects**	**Voxel grids**			
		Total	**Training**	**Validation**	**Test**
Posts	1 011	2 289	1 791	190	308
Vegetation	605	3 655	1 839	198	1 618
Buildings	113	5 105	1 849	179	3 077
Total	**1 729**	**11 049**	**5 479**	**567**	**5 003**

Table 3. Second dataset description

Class	**Objects**	**Voxel grids**			
		Total	**Training**	**Validation**	**Test**
Posts	1 011	2 289	1 791	190	308
Vegetation	4 920	14 331	7 423	3 253	3 655
Buildings	270	8 384	2 442	457	5 485
Cars	333	682	262	71	349
Total	**6 534**	**25 686**	**11 918**	**3 971**	**9 797**

Each point cloud object is converted into voxel grids with shape (32, 32, 32) and step $s = 16$. This means that when an object is converted into a voxel grid of shape (x, y, z), a voxel grid with shape (32, 32, 32) “slides” over the initial grid with step $s = 16$ along each axis until the input shape is covered. To avoid mostly empty voxel grids, a minimum of 20 points is required in each cubed point cloud to be converted into a voxel grid.

Figures Figure 6 and Figure 7 show examples of voxelizing objects.

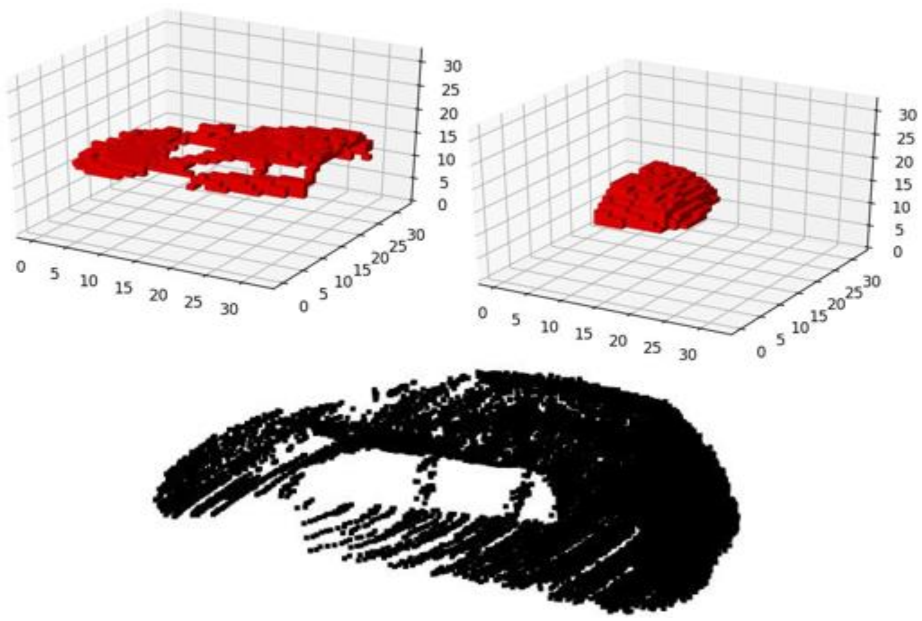

Figure 6. Point cloud and voxel grid representations of a car.

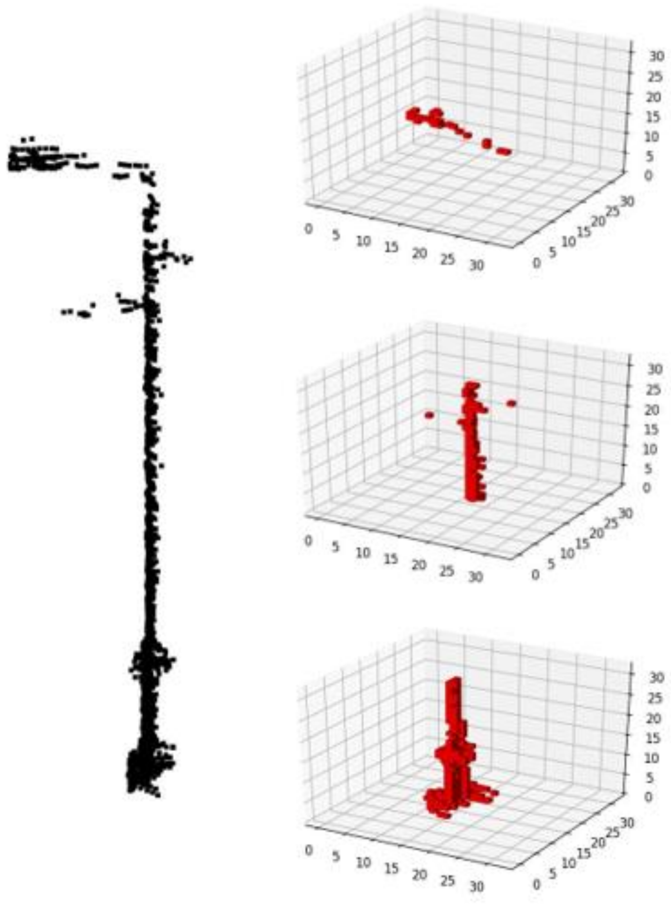

Figure 7. Point cloud and voxel grid representations of a post.

In the beginning of each training epoch, input voxel grids are shuffled and randomly rotated by $90 \cdot k|k \in \{0...3\}$ degrees around the *z* axis.

4.2 The proposed Convolutional Neural Network

For this task the DenseNet model was chosen due to feature map reuse and the small number of parameters required. The layer shapes were adjusted to handle 3D data, and less filters were used for computational reasons.

The constructed model consists of four dense blocks, each followed by a transitional block. Within the dense block are $n \in \{6, 12, 24, 16\}$ dense layers, of which each consists of one convolutional layer with a $(1 \times 1 \times 1)$ kernel and two convolutional layers with a $(3 \times 3 \times 3)$ kernel. In each dense layer the feature maps from the last block are concatenated with the feature maps of the new block. The transitional block consists of a convolutional layer with $(1 \times 1 \times 1)$ kernel and an average pooling layer. After the four dense blocks, global average pooling is performed to minimize overfitting by reducing the total number of parameters in the model. Lastly, there is a fully connected layer resulting in *n* outputs.

In the constructed networks, the $growth\_rate = 4$. This means that in every dense layer four new feature maps are added to the feature maps. This network architecture can be reused in different networks as the only parameter which requires changing is the number of outputs *n*.

A full summary of the network is presented in [37]. The description and visualisation of the basic architecture can be seen in

Table 4 and Figure 8. In the figure, the smallest blocks represent applying convolution to the previous layer.

Table 4. Architectural overview of 3D DenseNet.

Layers	**Output size**	**3D DenseNet**
Convolution	$18 \times 18 \times 18$	$4 \times 4 \times 4$ convolution, stride 2
Pooling	$20 \times 20 \times 20$	$3 \times 3 \times 3$ max pooling, stride 1
Dense block (1)	$20 \times 20 \times 20$	$\begin{bmatrix} 1 \times 1 \text{ convolution} \\ 3 \times 3 \text{ convolution} \\ 3 \times 3 \text{ convolution} \end{bmatrix} \times 6$
Transition block (1)	$20 \times 20 \times 20$	$1 \times 1 \times 1$ convolution
	$19 \times 19 \times 19$	$2 \times 2 \times 2$ average pooling, stride 1
Dense block (2)	$19 \times 19 \times 19$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition block (2)	$19 \times 19 \times 19$	$1 \times 1 \times 1$ convolution
	$18 \times 18 \times 18$	$2 \times 2 \times 2$ average pooling, stride 1
Dense block (3)	$18 \times 18 \times 18$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Transition block (3)	$18 \times 18 \times 18$	$1 \times 1 \times 1$ convolution
	$17 \times 17 \times 17$	$2 \times 2 \times 2$ average pooling, stride 1
Dense block (4)	$17 \times 17 \times 17$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$
Classification layer		Global average pooling
		Fully connected softmax layer

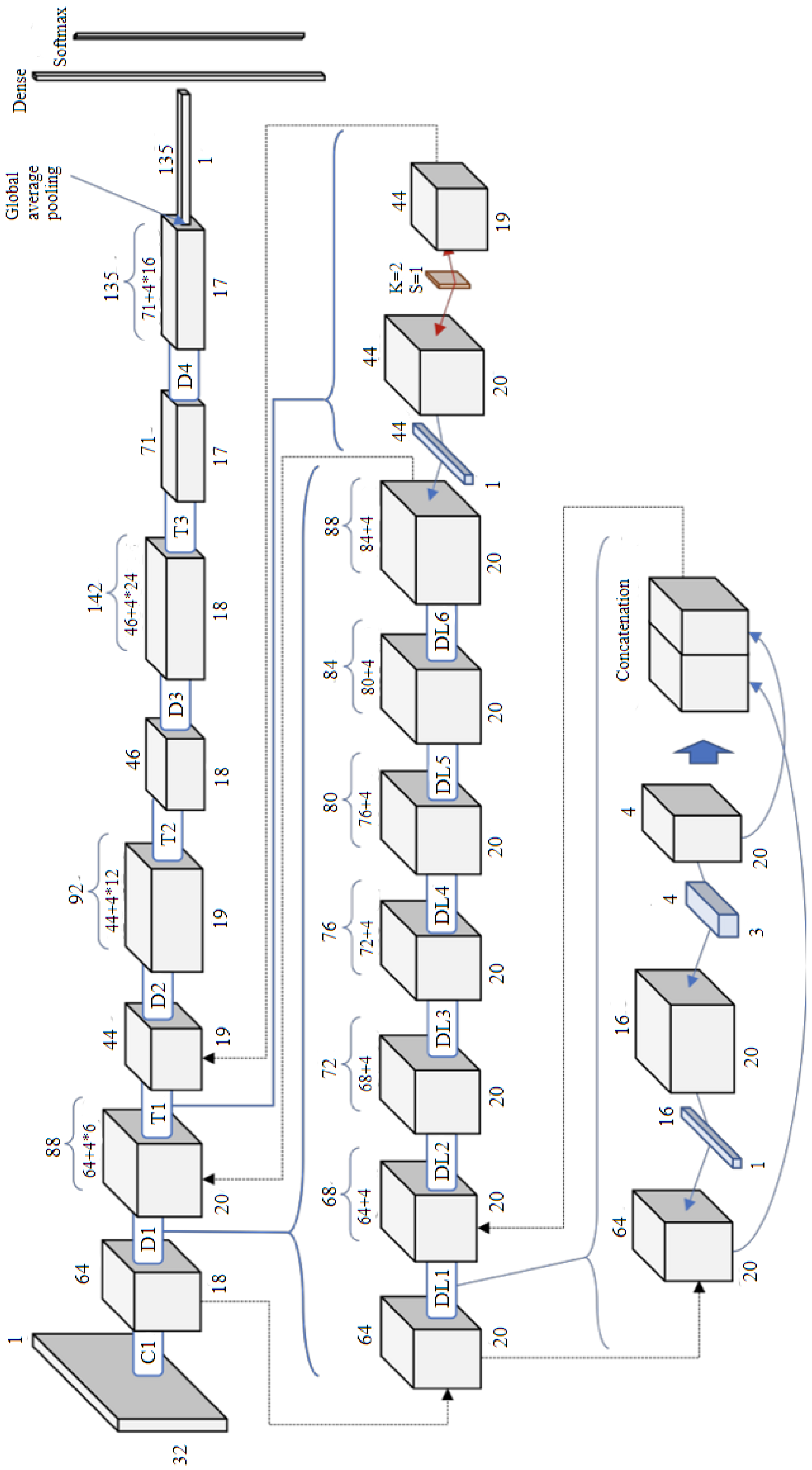

Figure 8. 3D DenseNet architecture. Numbers on the top describe the number of feature maps in the current block or layer. Numbers on the bottom represent the input shape (e.g. number x refers to a shape of $(x \times x \times x)$). Dx = dense block number x; T1 = transition layer number x; DLx = dense layer number x; K = kernel; S = stride. [38] (modified)

Multiple CNNs with this architecture were trained:

- **DenseNet-3C** - a CNN with an output for three classes: post, vegetation, building (the number of outputs $n = 3$);
- **DenseNet-4C** - a CNN with an output for four classes: post, vegetation, building, car (the number of outputs $n = 4$);
- **DenseNet-4CB** – a method which uses four CNNs with an output for two classes, where each is a binary classifier for the classes post, vegetation, building and car (each classifier has $n = 2$ outputs).

Initially, the 3-class network was constructed. With this network cars were often misclassified, and thus the 4-class network was made. Realising that each added class meant training a new network, binary classifiers were made for each class. This way the networks already trained are still usable if there is a wish to add a new class.

The models are trained with a batch size of 32, Adadelta as the optimizer, and categorical cross-entropy as the loss function.

For the training, a computer with the graphics card Nvidia GeForce RTX 2080 is used.

4.3 Scene classification

To classify objects in a larger point cloud scene, both approaches are implemented – the sliding window and clustering methods. Before each method, ground removal is done using PCL Progressive Morphological Filter. This method was chosen because with the correct parameters it is possible to aggressively remove all ground points, even if this means losing the bottom part of some objects. This is important to the clustering method, because if there was ground remaining under a few close objects, they would be clustered into one object.

4.3.1 Sliding window method implementation

The input point cloud is read from a PCD file. As the first step, the point cloud in transformed into a voxel grid representation where occupied voxels are represented with one and empty voxels with zero. This means every connected set of non-empty voxels depict detected objects.

As the second step, the grid is padded so that the furthermost points would also get multiple predictions. Voxel grids of shape (32, 32, 32) are generated by sliding over the grid with a step s. All these grids are predicted, and the probabilities are appended to the list of probabilities for all voxels in the centre of the voxel grid with c side length. As the last step, the class of each voxel is voted. The coloured by class point cloud is saved to a PCD file.

4.3.2 Clustering method implementation

In the clustering approach, the point cloud scene is segmented into objects by applying the PCL Euclidean Cluster Extraction. The minimal distance between clusters is d_{min} and the minimum number of points in cluster is 100. The initial ground removal height is 0.4 m to aggressively remove the ground so that no objects are connected to each other. The set of clustered objects are the detected objects in this method.

This method involves classifying points by object. For each object, voxel grids are generated with step $s = 16$. All these grids are then predicted using the network. The object class derived from the mean of predictions is applied to the all points included in the object. If predictions for all classes are under probability p, the object is classified as "other".

5 Analysis

This section analyses the performances of the proposed CNN architecture and the two different scene classification methods. According to the analysis, ideas on how to improve the solution are given.

5.1 Performance of the Convolutional Neural Networks

In order to evaluate the networks ability to learn, the accuracy on the train and validation dataset should increase whilst the loss decreases.

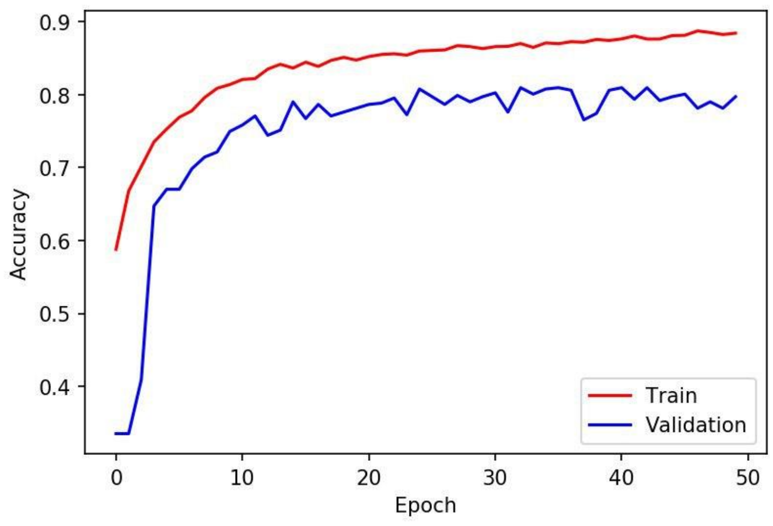

Figure 9. Accuracy on train and validation datasets for DenseNet-3C.

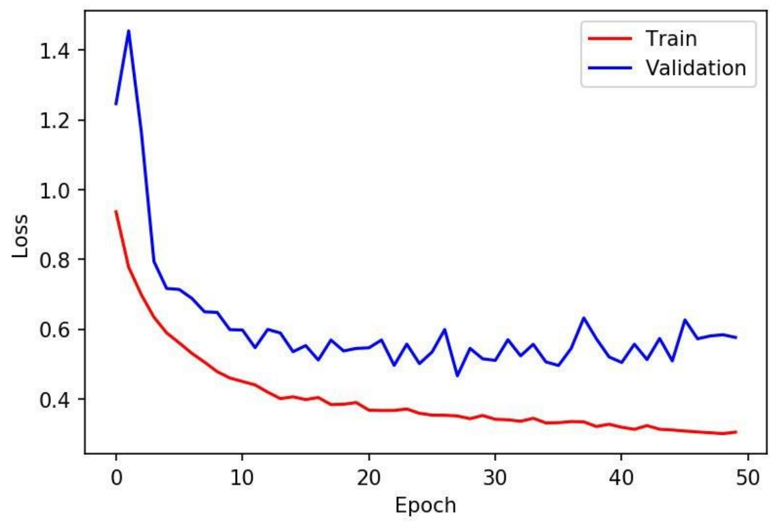

Figure 10. Loss on train and validation datasets for DenseNet-3C.

According to Figures Figure 9 and Figure 10, the DenseNet-3C network is suitable to generalize the features of different objects. Final accuracy on the training dataset lies around 88% while the accuracy on the validation set is 80%.

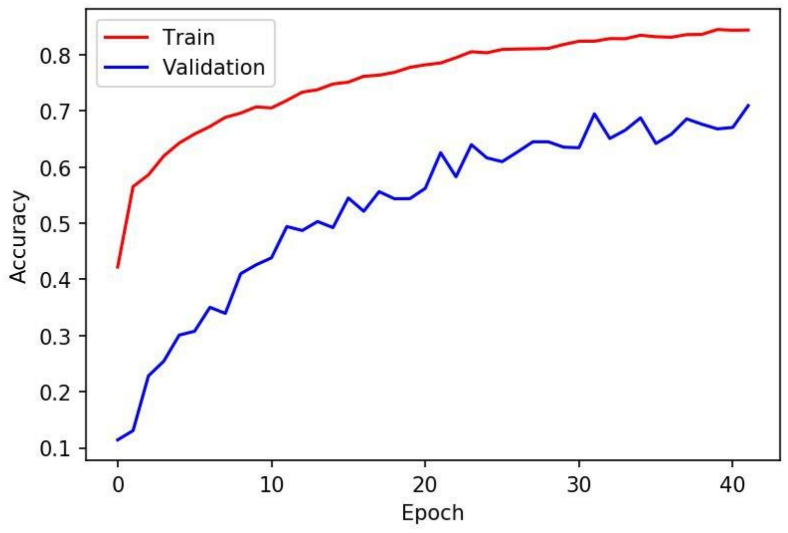

Figure 11. Accuracy on train and validation datasets for DenseNet-4C.

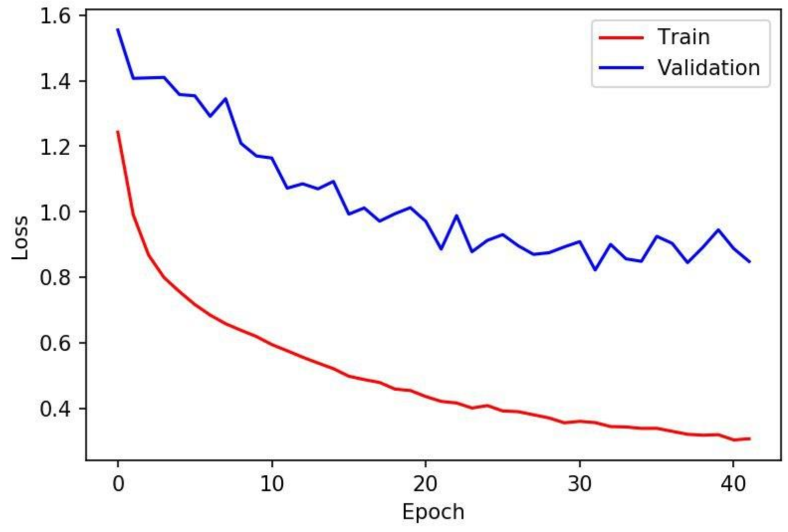

Figure 12. Loss on train and validation datasets for DenseNet-4C.

Figures Figure 11 and Figure 12 show the accuracy and loss of the training and validation dataset for DenseNet-4C. Final accuracy on the training dataset lies around 84% while the accuracy on the validation set is 71%. The results are slightly worse due to a more complicated problem – instead of three classes, four classes are taught to the network.

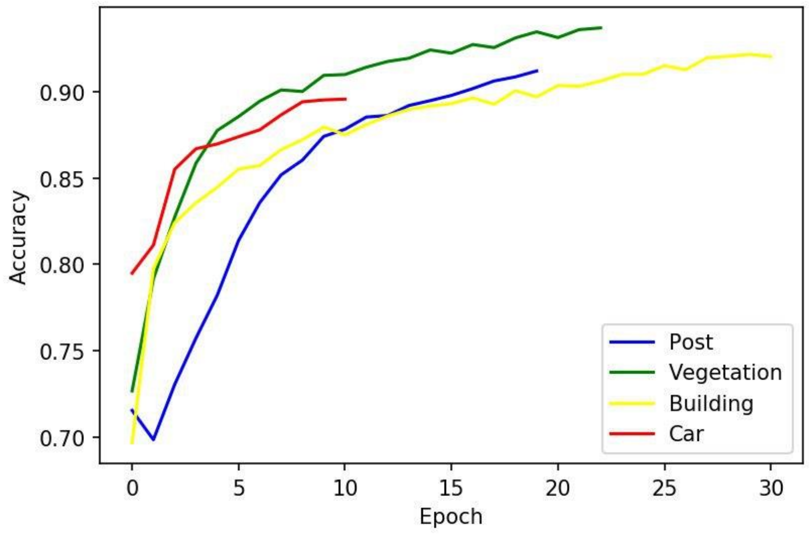

Figure 13. Accuracy on train dataset for networks of DenseNet-4CB.

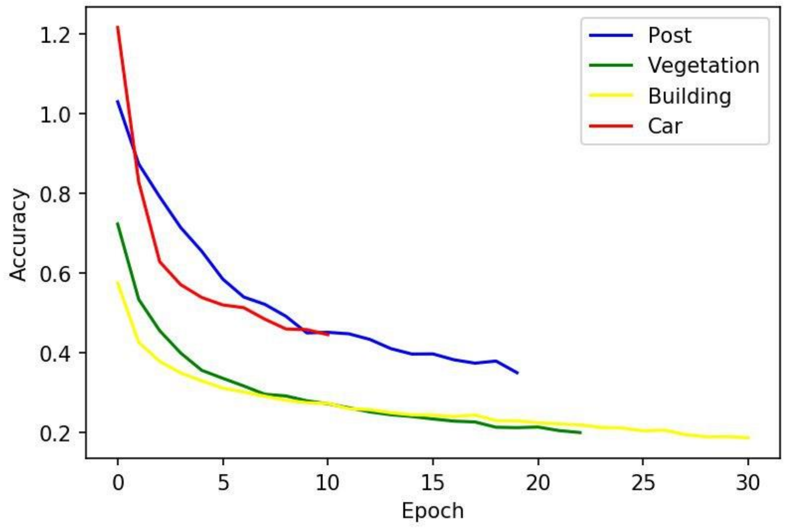

Figure 14. Loss on train dataset for networks on DenseNet-4CB.

Figures Figure 13 and Figure 14 display the training dataset accuracy and validation curves for all the binary classifiers of DenseNet-4CB. The number of epochs trained differ amongst the classifiers as early stopping was used. The graphs show that the traits of vegetation and buildings were the easiest to learn, possibly since these classes have more data to support training than the other classes. The final accuracies on the training dataset are for posts, vegetation, buildings and cars respectively 91%, 93%, 92% and 90%. Thus, the average accuracy on the training dataset is higher for the DenseNet-4CB when compared to DenseNet-4C.

5.2 Classification analysis

To assess the shortcomings of the classification task, a confusion matrix is generated on the test dataset. In this confusion matrix, each segment or voxel grid of an object is considered as one instance. Columns and rows represent classes. Each cell with column *y* and row *x* shows how many instances of class *x* is predicted to belong to class *y*.

As per Figure 15, for the DenseNet-3C the traits of posts are the easiest to learn. The biggest confusion occurs when classifying segments of buildings. This might be since the edge voxel grids of buildings may be confused with other classes.

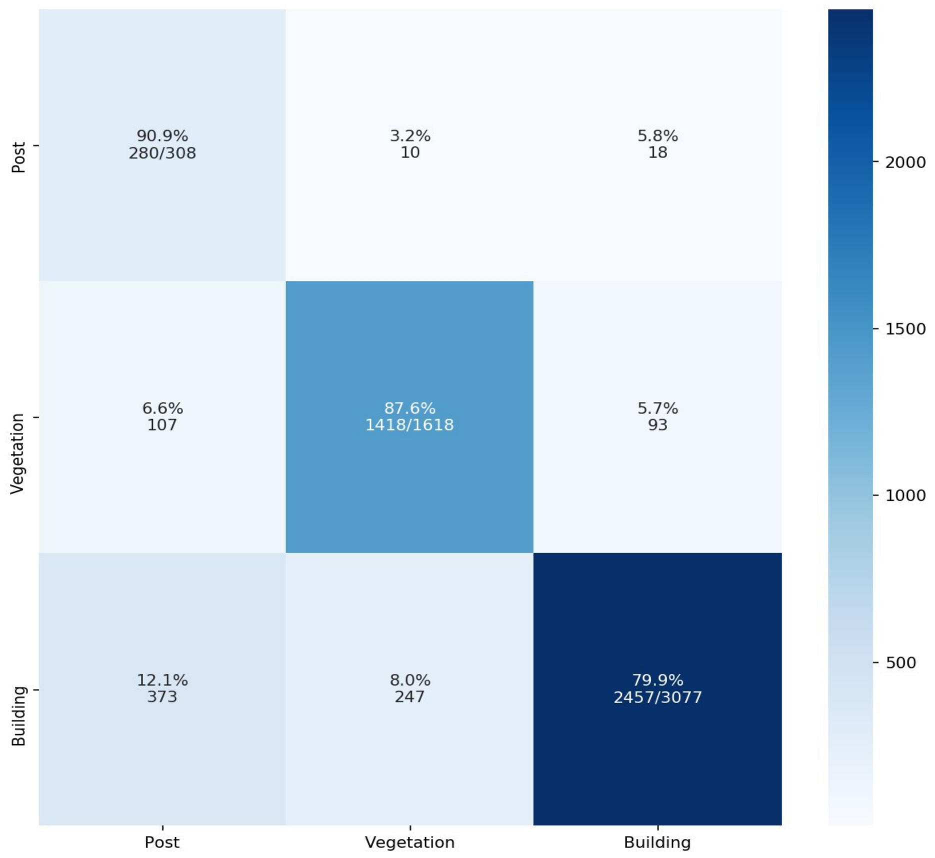

Figure 15. Confusion matrix on test dataset for DenseNet-3C (evaluation by segment).

To improve results, another confusion matrix is generated with object-wise predictions. Object-wise classification means that each voxel grid of an object produces a list of predictions for each class (e.g. $[0.1, 0.3, 0.6]$ means that there is a 0.1 probability that this segment is a post, and so on). These predictions are then summed by class amongst all voxel grids of the object and divided by the number of voxel grids. The class with the highest probability in this result is the determined class of the object and represents the class of every segment generated from the object.

According to Figure 16, the results of DenseNet-3C have drastically improved. This validates the assumption that edges of objects are more difficult to classify. The highest remaining is in classifying buildings as vegetation. The cause of this might be that some pieces of buildings are of irregular shape, as is vegetation.

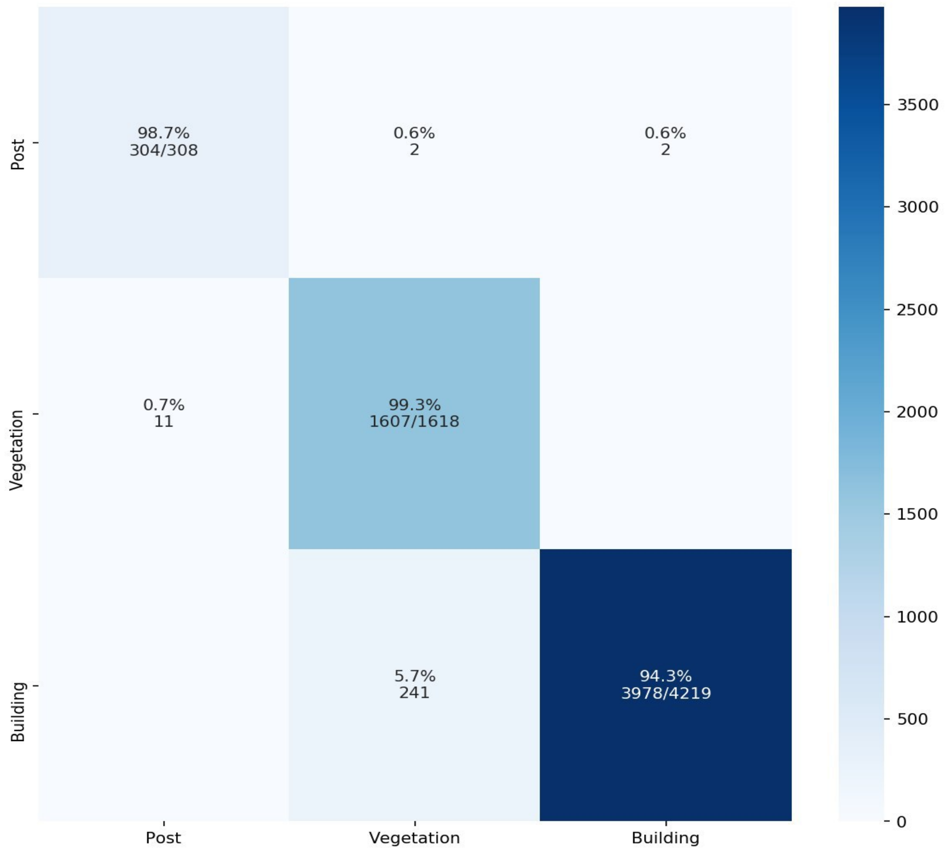

Figure 16. Confusion matrix on test dataset for DenseNet-3C (evaluation by object).

Comparing the confusion matrices of evaluation by segment and evaluation by object, this also sheds light onto which scene classification method will perform better. Since the evaluation by object provides better results, this method will be used to analyse the performances of the four class models as well.

The object-wise confusion matrices for DenseNet-4C and DenseNet-4CB are shown respectively in Figures Figure 17 and Figure 18. Mostly these models have similar results, however regarding the classification of cars, the DenseNet-4C performs better. This might be due to insufficient training of the classifier for cars of DenseNet-4CB.

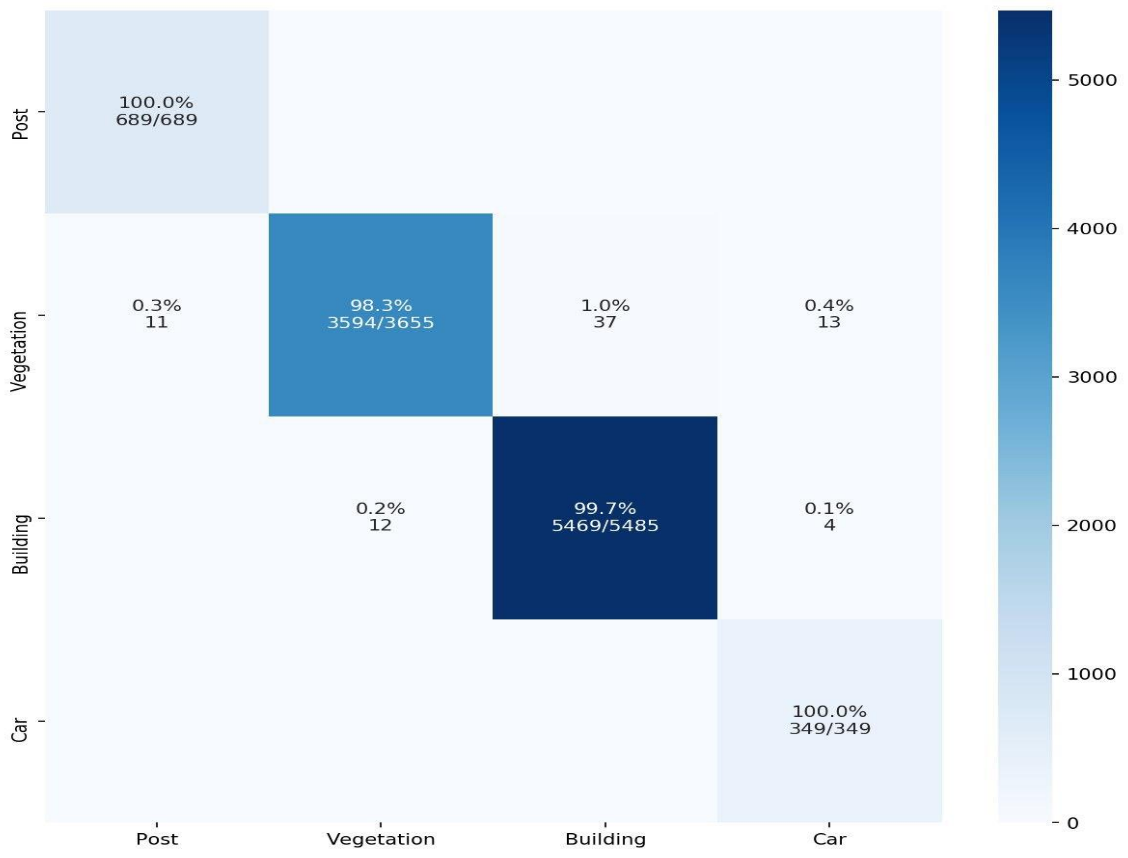

Figure 17. Confusion matrix on test dataset for DenseNet-4C (evaluation by object).

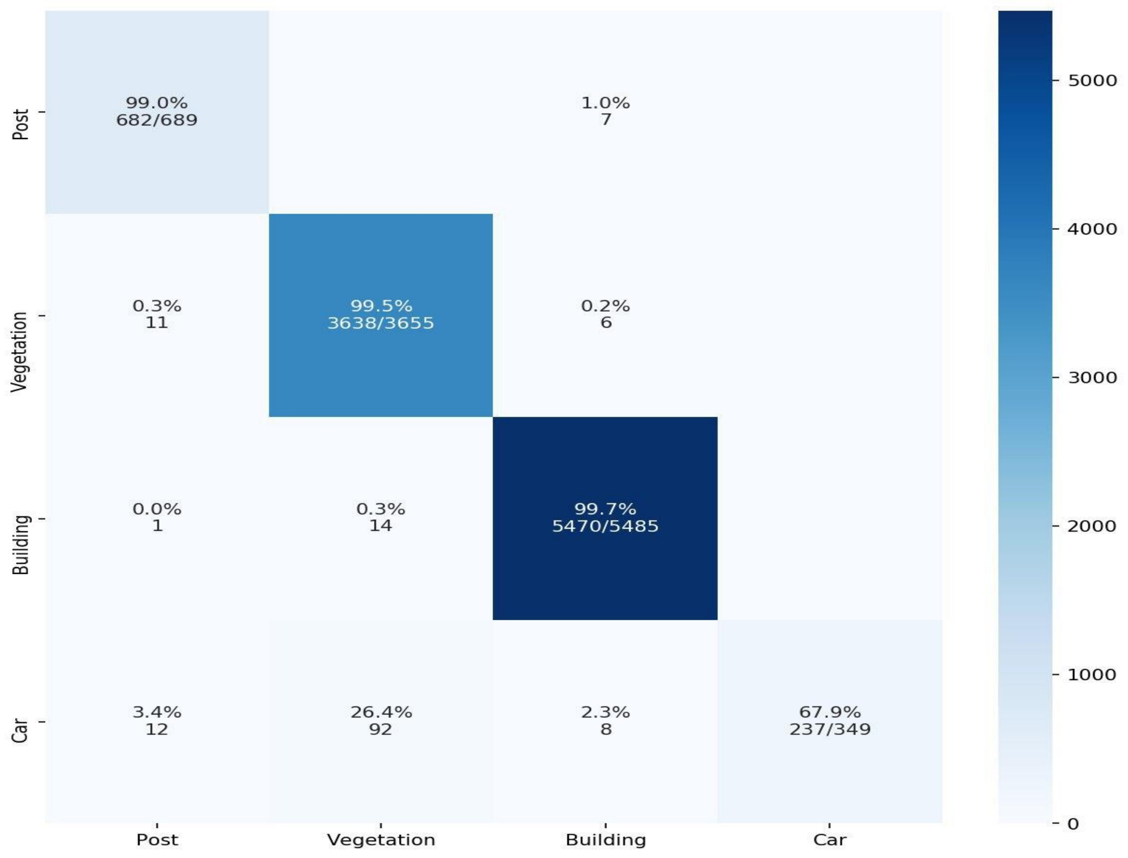

Figure 18. Confusion matrix on test dataset for DenseNet-4CB (evaluation by object).

Table 5 gives a better overview of the performances of three models on the test datasets. The results of DenseNet-4CB are worse due to the class of cars having recall of only 68%. Surprisingly, while adding a fourth class, the evaluation metrics have not decreased much for DenseNet-4C when compared to the three-class DenseNet-3C.

Table 5. Comparison of DenseNet-3C, DenseNet-4C and DenseNet-4CB.

Model	**Accuracy**	**Average**			**Weighted**		
		Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
DenseNet-3C	**99.5%**	**98.5%**	99.2%	**98.9%**	**99.5%**	**99.5%**	**99.5%**
DenseNet-4C	99.2%	98.2%	**99.5%**	98.8%	99.3%	99.2%	99.2%
DenseNet-4CB	98.5%	98.4%	91.5%	94.2%	98.6%	98.5%	98.4%

5.3 Scene classification performance

This section analyses the performance of both scene classification methods by varying the parameters.

The example point cloud contains 1 875 232 points. The calculated voxel grid size is $1924 \times 1814 \times 182$ voxels (x, y, z side lengths). Since the size of voxels is $0.1 \times 0.1 \times 0.15$ m, the size of the voxel grid of this point cloud scene is $192.4 \times 181.4 \times 27.3$ m. The point cloud is coloured according to determined classes of points: posts are blue, vegetation is green, and buildings are yellow.

5.3.1 Sliding window method analysis

First the step **parameter *(s)*** is analysed with $c = 32$ used as default, where s is the step and c represents the side length of the cube in the centre of the predicted voxel grid, where the voxels inside the cube are appended the probabilities. As seen the figures below, this parameter does not increase performance by much, yet smaller steps take exponentially more time. The processing took two minutes and 46 seconds for Figure 19 with step $s = 16$, 15 minutes and 35 seconds for Figure 20 with $s = 8$, and one hour and 56 minutes for Figure 21 with $s = 4$.

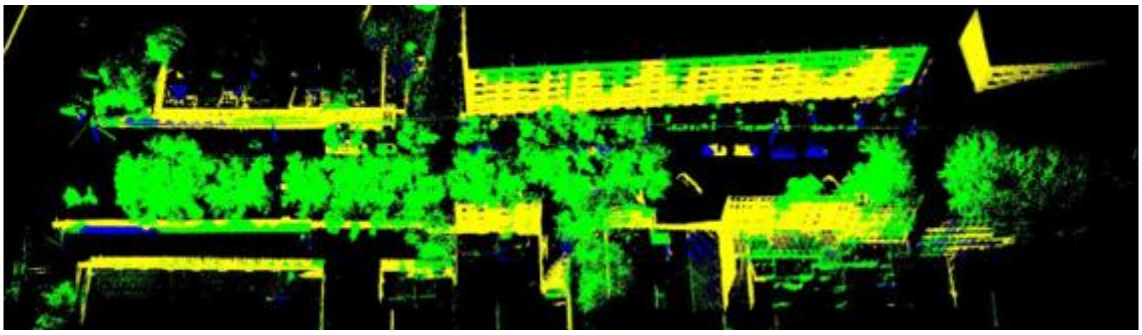

Figure 19. Sliding window method with $s = 16$ and $c = 32$ for DenseNet-3C, where s is the step and c is the side length of the cube where predictions are applied.

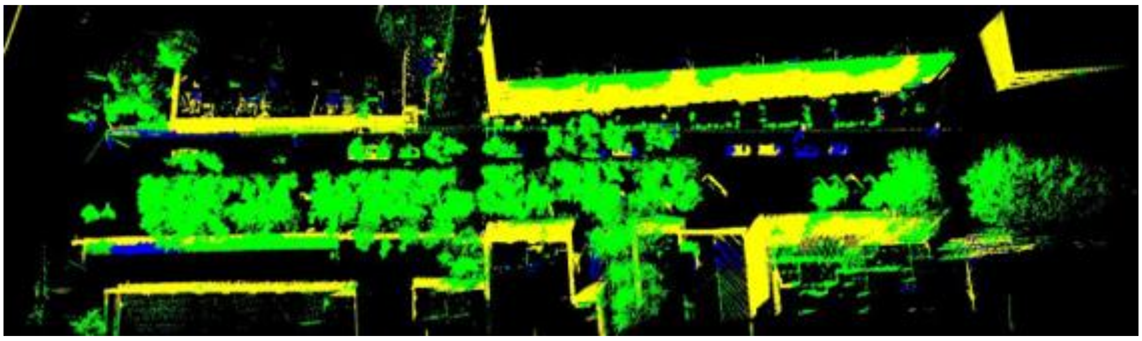

Figure 20. Sliding window method with $s = 8$ and $c = 32$ for DenseNet-3C, where s is the step and c is the side length of the cube where predictions are applied.

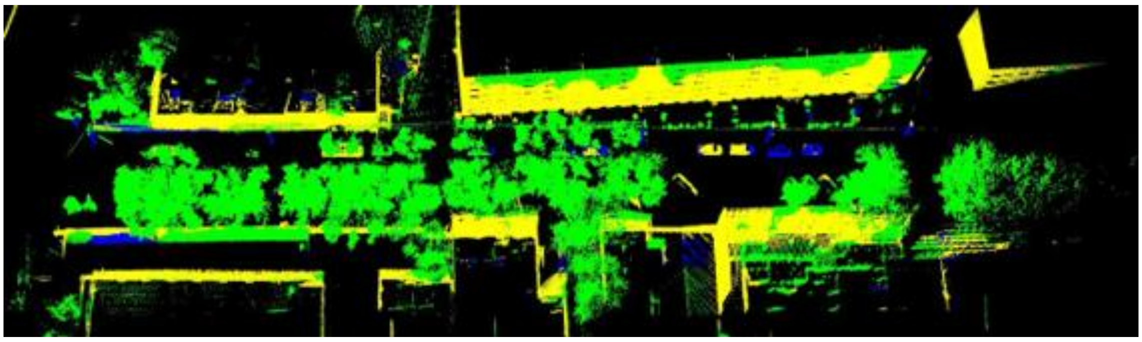

Figure 21. Sliding window method with $s = 8$ and $c = 32$ for DenseNet-3C, where s is the step and c is the side length of the cube where predictions are applied.

Next, the **parameter *(c)*** is analysed. Step $s = 16$ is used in these calculations. Parameter c cannot be smaller than s as this would result in some voxels not being classified. Figure 22 displays the results when $c = 16$.

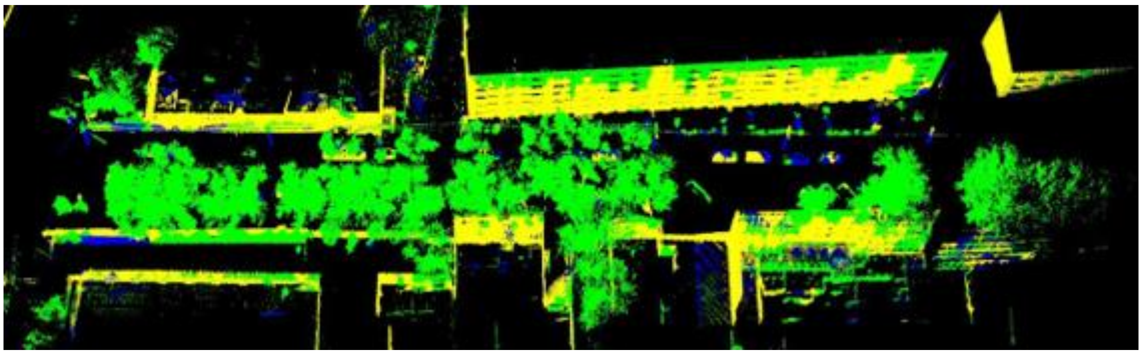

Figure 22. Sliding window method with $s = 16$ and $c = 16$ for DenseNet-3C, where s is the step and c is the side length of the cube where predictions are applied.

Unfortunately, the adjustment of the parameters s and c have not provided better results. Thus, the most time efficient parameters are proposed: $s = 16$ and $c = 32$.

5.3.2 Scene clustering method analysis

In the scene clustering method, the parameters analysed are d, the minimum distance between clusters, and p, the minimum required probability of the predicted class. If the probability lies under p for each class, the object is said to be of class "other" and is coloured grey. The colour codes for the other three classes remain the same.

The problem with **parameter *d*** is that if it is too low, there will be many small clusters which are hard to predict. If d is too high, different objects will be clustered as one. If **parameter *p*** is too low, objects that do not belong to the three classes, for example cars, will be classified wrongly. If it is too high, an object may belong to a class, but since its probability is not high enough it will be classified as "other".

Figures Figure 23 and Figure 24 show the effect of $d = 0.2$ and $d = 0.3$ respectively with a constant $p = 0.6$. Figures Figure 25 and Figure 26 depict the results of $d = 0.2$ and $d = 0.3$ respectively with a constant $p = 0.5$. According to the results, the proposed parameters are $d = 0.3$ and $p = 0.5$. The average processing time is one minute and 30 seconds.

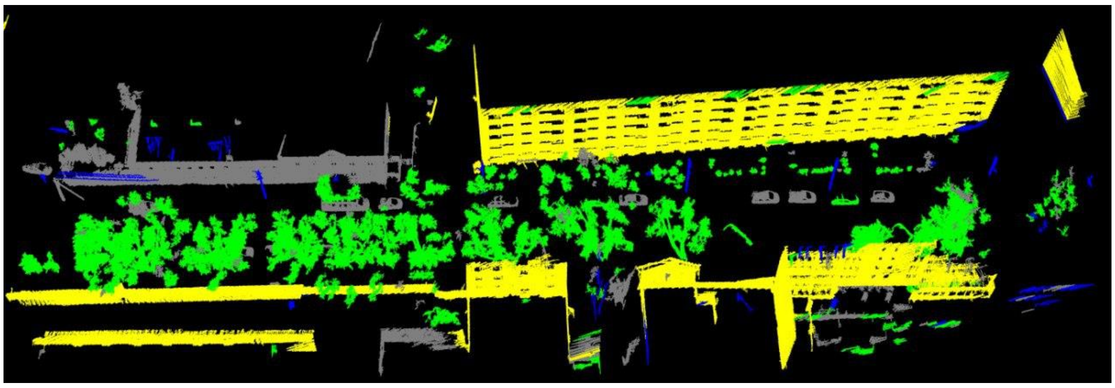

Figure 23. Scene clustering method with $p = 0.6$ and $d = 0.2$ for DenseNet-3C, where d is the minimum distance between clusters and p is the minimum required probability of the predicted class.

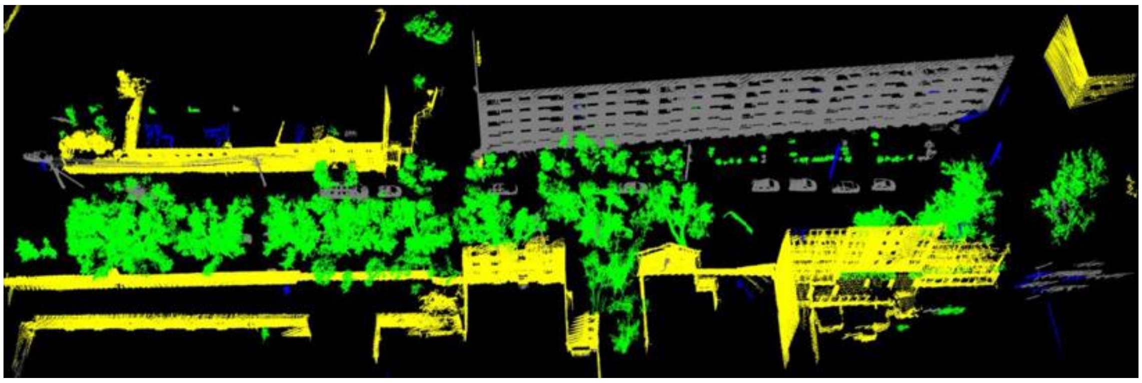

Figure 24. Scene clustering method with $p = 0.6$ and $d = 0.3$ for DenseNet-3C, where d is the minimum distance between clusters and p is the minimum required probability of the predicted class.

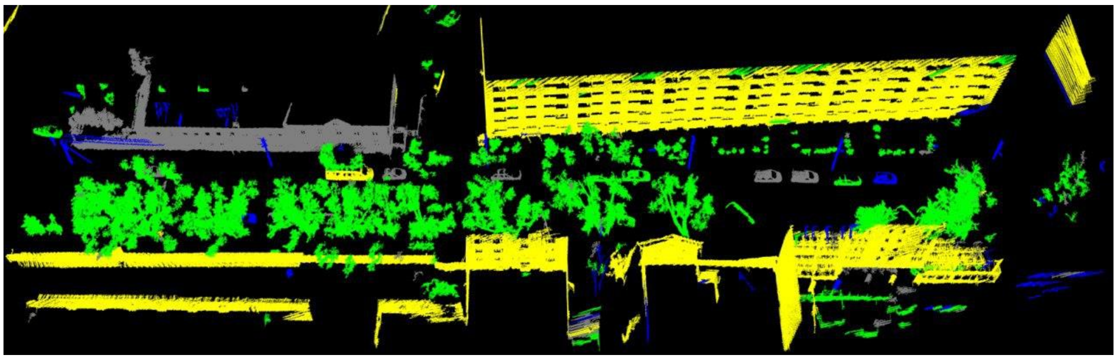

Figure 25. Scene clustering method with $p = 0.5$ and $d = 0.2$ for DenseNet-3C, where d is the minimum distance between clusters and p is the minimum required probability of the predicted class.

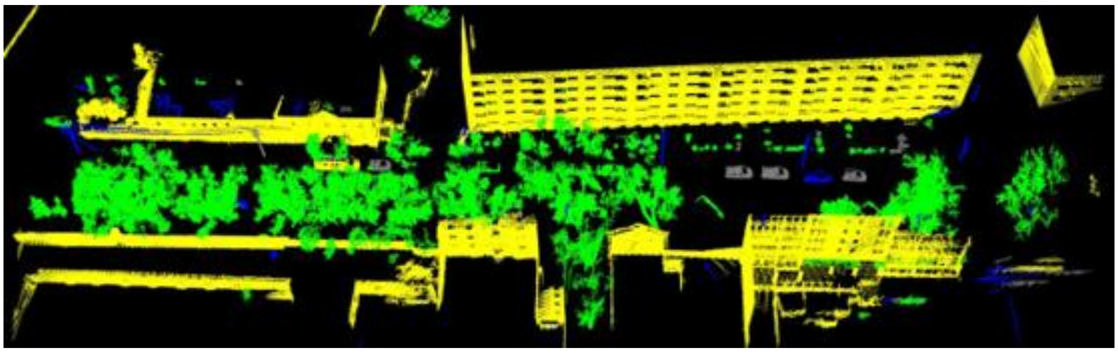

Figure 26. Scene clustering method with $p = 0.5$ and $d = 0.3$ for DenseNet-3C, where d is the minimum distance between clusters and p is the minimum required probability of the predicted class.

Since the scene clustering method gives better results visually and is also more time efficient, the other networks are analysed with this method. The probability parameter *p* is set to 0.2 as there are not a lot of objects to be classified as "other" when the class for cars exists. The distance parameter *d* remains the same for clustering.

Comparing Figures Figure 27 and Figure 28, the DenseNet-4C provides better results. While DenseNet-4CB classifies a few cars as vegetation, the DenseNet-4C classifies them correctly. Contrarily, the DenseNet-4C classifies some vegetation as cars. Regarding posts, the DenseNet-4C identifies a post with wires on the left side while the other model does not. The processing time for this scene is one minute and 15 seconds for DenseNet-4C and three minutes for DenseNet-4CB.

Figures Figure 29 and Figure 30 demonstrate the models' performance on a larger point cloud scene. The most significant shortcoming for both models is classifying some buildings as vegetation. As regards moving and stationary cars, the DenseNet-4C performs better again. The processing time for this scene is nine minutes for DenseNet-4C and 13 minutes for DenseNet-4CB.

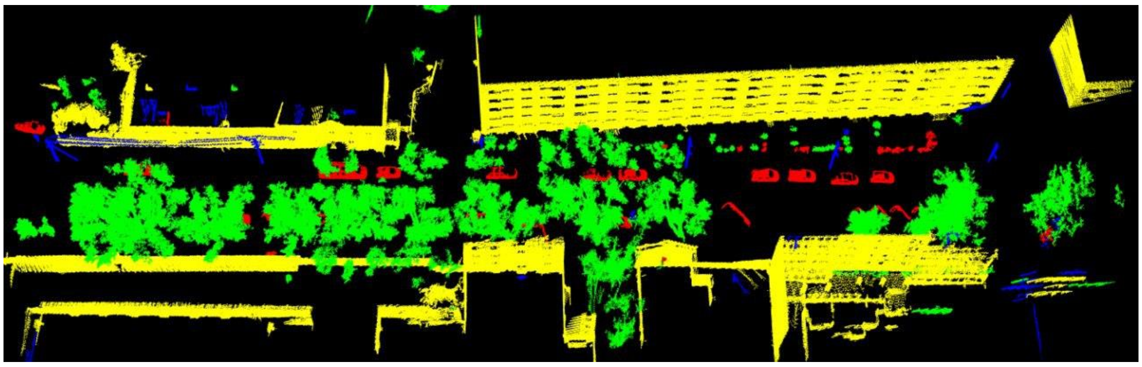

Figure 27. Scene clustering method with $p = 0.2$ and $d = 0.3$ for DenseNet-4C, where *d* is the minimum distance between clusters and *p* is the minimum required probability of the predicted class.

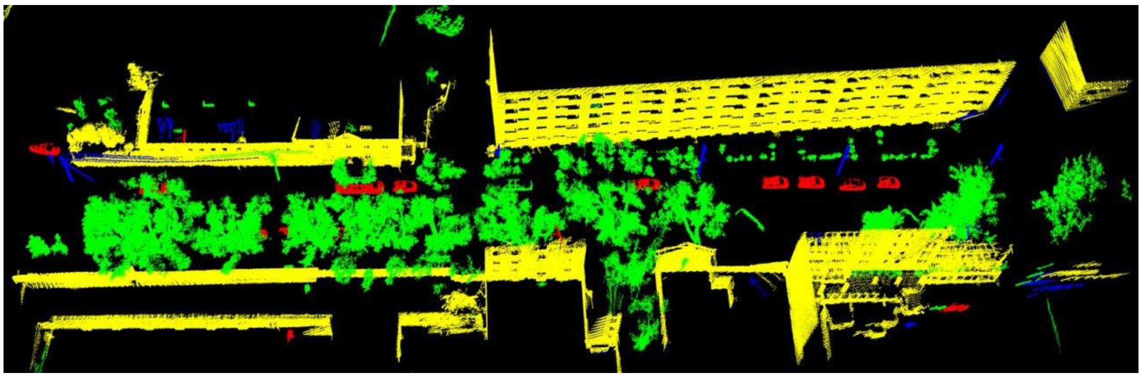
Figure 28. Scene clustering method with $p = 0.2$ and $d = 0.3$ for DenseNet-4CB, where d is the minimum distance between clusters and p is the minimum required probability of the predicted class.

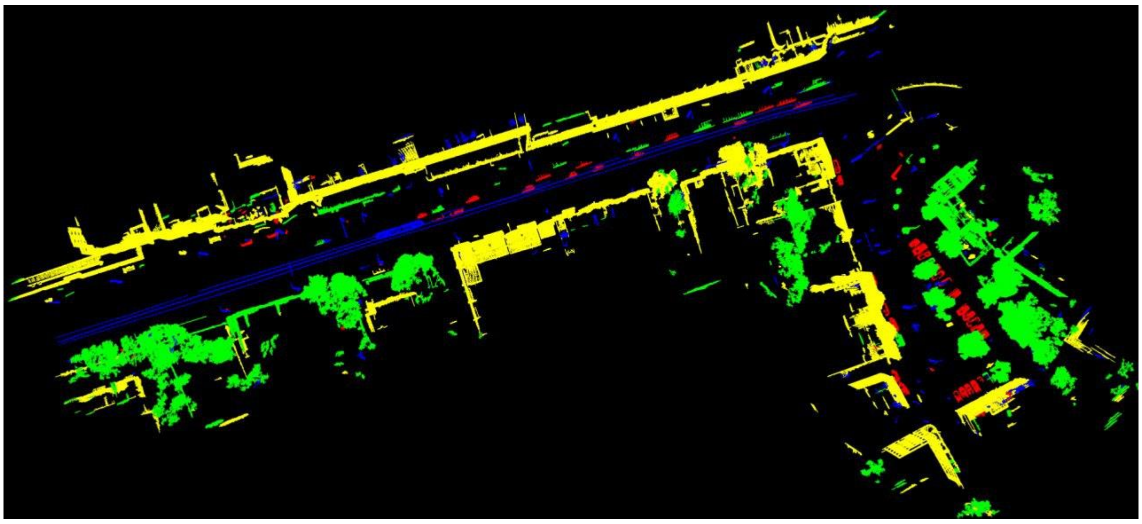
Figure 29. Scene clustering method with $p = 0.2$ and $d = 0.3$ for DenseNet-4C, where d is the minimum distance between clusters and p is the minimum required probability of the predicted class.

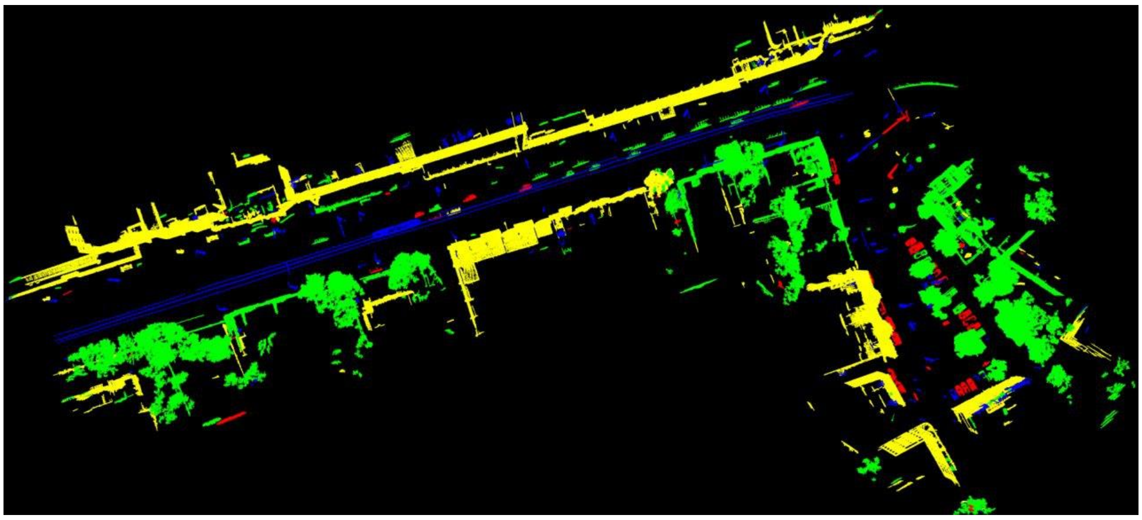
Figure 30. Scene clustering method with $p = 0.2$ and $d = 0.3$ for DenseNet-4CB, where d is the minimum distance between clusters and p is the minimum required probability of the predicted class.

Considering all experiments, the proposed method for scene classification is the scene clustering method. Regarding the model, DenseNet-4C is proposed if no more than these four classes are required in scene classification. If it is desirable to add new classes in the future, DenseNet-4CB is suggested.

5.4 Proposed future improvements

The suggested solution could be improved in many ways. Firstly, the dataset size is quite small at the moment as only 25 point cloud scenes were used to generate the dataset used in this work. A larger dataset would help the network generalise the classes even better. Secondly, the classes could be more specific – a hierarchical structure could be made, i.e. the current class of posts could be further classified as street lighting, traffic light or traffic sign posts. Lastly, the most significant issue concerns scene classification. Further research should be conducted on how to perform the clustering of objects with less error.

5.4.1 Proposed scene classification improvements

Even though the results of scene classification are quite satisfactory, they obviously are not as accurate as the performance on the test dataset would suggest. The task of scene classification is a lot more difficult than just classifying “clean” objects.

The test dataset consists of well clustered and verified objects. During the dataset generation many unfit objects, such as small clusters which could not be identified or buildings with connected trees, were disregarded.

The objects generated from the scenes are not as cleanly clustered as it is not always possible to separate objects. The parameter for the minimal distance between the clusters is applied to the whole scene and while in some parts a smaller parameter would be more suitable, in other parts a larger parameter would yield better results. There is no perfect variable which would result in perfect object clustering. Thus, further research on the topic of scene segmentation is suggested.

6 Summary

The classification of point cloud scenes is a relatively new research. Point cloud data takes much more time to process than 2D data as adding a third dimension with length L increases the data size L times. However, 3D data has a great potential in various fields due to the benefits of spatial data. The purpose of this work was to construct a point cloud data classifier to detect posts, vegetation, buildings and cars in point cloud scenes. To fulfil this objective, first, a new dataset was constructed, second, a custom voxelization of point clouds was programmed, third, CNN models were built, and lastly two scene classification methods were implemented.

This work has verified that CNNs, more specifically DenseNets in this case, are suitable to solve the problem of 3D object classification. The proposed model achieved an accuracy of 99.2% on the test dataset.

The scene classification problem was solved with two different methods: the sliding window method and the scene clustering method. The proposed approach of scene clustering achieved visually better results. This method was also validated by comparing segment-wise and object-wise classification results. While the chosen network only achieved an accuracy of 83.7% with segment-wise classification, the object-wise classification produced an accuracy of 99.2%.

The proposed solution could be improved in several ways. First, the training dataset should be expanded in size and by adding new and more specific classes. Second, further research should also be conducted regarding the scene clustering method, as the simple approach of clustering objects with a minimal distance between objects does not always produce perfectly clustered objects.

References

[1] “Why LiDAR,” LeddarTech, [Online]. Available: https://leddartech.com/why-lidar/. [Accessed 15 May 2020].

[2] “Lidar,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Lidar. [Accessed 22 April 2020].

[3] “Point cloud,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Point_cloud. [Accessed 22 April 2020].

[4] “The PCD (Point Cloud Data) file format,” Point Cloud Library, [Online]. Available: http://pointclouds.org/documentation/tutorials/pcd_file_format.php. [Accessed 22 April 2020].

[5] “Multiclass classification,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Multiclass_classification. [Accessed 22 April 2020].

[6] S. Asiri, “Machine Learning Classifiers,” [Online]. Available: https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623. [Accessed 10 May 2020].

[7] “Statistical classification,” Wikipedia, [Online]. Available: https://en.wikipedia.org/w/index.php?title=Statistical_classification. [Accessed 22 April 2020].

[8] “Evaluation metrics for classification problems in machine learning,” Toward Data Science, [Online]. Available: https://towardsdatascience.com/evaluation-metrics-for-classification-problems-in-machine-learning-d9f9c7313190. [Accessed 22 April 2020].

[9] “F1 score,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/F1_score. [Accessed 16 May 2020].

[10] T. Yiu, “Understanding neural networks,” Towards Data Science, [Online]. Available: https://towardsdatascience.com/understanding-neural-networks-19020b758230. [Accessed 22 April 2020].

[11] D. Liu, “A Practical Guide to ReLU,” Medium, 30 Nov 2017. [Online]. Available: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7. [Accessed 16 May 2020].

[12] “A Comprehensive Guide to Convolutional Neural Networks,” Towards Data Science, [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed 22 April 2020].

[13] K. Koidl, “Loss Functions in Classification Tasks,” [Online]. Available: https://www.scss.tcd.ie/~koidlk/cs4062/Loss-Functions.pdf. [Accessed 10 May 2020].

[14] “Loss Functions,” [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html. [Accessed 24 April 2020].

[15] S. K. Basaveswara, “CNN Architectures, a Deep-dive,” Towards Data Science, [Online]. Available: https://towardsdatascience.com/cnn-architectures-a-deep-dive-a99441d18049. [Accessed 24 April 2020].

[16] E. Ma, “What is VGG neural network?,” 9 Dec 2019. [Online]. Available: https://becominghuman.ai/what-is-the-vgg-neural-network-a590caa72643. [Accessed 10 May 2020].

[17] “Vanishing gradient problem,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Vanishing_gradient_problem. [Accessed 24 April 2020].

[18] “Batch Normalization in Neural Networks,” Towards Data Science, [Online]. Available: https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c. [Accessed 24 April 2020].

[19] W. Zhang, J. Qi, W. Peng, H. Wang, D. Xie, X. Wang and G. Yan, “An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation,” June 2016. [Online]. Available: https://www.researchgate.net/publication/303975654_An_Easy-to-Use_Airborne_LiDAR_Data_Filtering_Method_Based_on_Cloth_Simulation. [Accessed 10 May 2020].

[20] K. Zhang, S.-C. Chen, D. Whitmanl, M.-L. Shyu, J. Yan and C. Zhang, “A Progressive Morphological Filter for Removing,” *IEEE Transactions on Geoscience and Remote Sensing,* April 2003.

[21] “Euclidean Cluster Extraction,” Point Cloud Library, [Online]. Available: http://www.pointclouds.org/documentation/tutorials/cluster_extraction.php. [Accessed 24 April 2020].

[22] L. Kasak, “CSF,” [Online]. Available: https://github.com/LiineKasak/CSF. [Accessed 15 May 2020].

[23] “Introduction,” Open3D, [Online]. Available: http://www.open3d.org/. [Accessed 24 April 2020].

[24] “TensorFlow,” TensorFlow, [Online]. Available: https://www.tensorflow.org/. [Accessed 24 April 2020].

[25] “Keras: The Python Deep Learning Library,” Keras, [Online]. Available: https://keras.io/. [Accessed 24 April 2020].

[26] M. Heller, “What is Keras? The deep neural network API explained,” 28 Jan 2019. [Online]. Available: https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html. [Accessed 10 May 2020].

[27] “CUDA,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/CUDA. [Accessed 24 April 2020].

[28] J. Huang and S. You, “Vehicle detection in urban point clouds with orthogonal-view convolutional neural network,” *IEEE International Conference on Image Processing (ICIP),* pp. 2593-2597, 2016.

[29] G. Pang and U. Neumann, “3D point cloud object detection with multi-view convolutional neural network,” *23rd International Conference on Pattern Recognition (ICPR),* pp. 585-590, 2016.

[30] Y. Huang and S. You, “Point cloud labeling using 3D Convolutional Neural Network,” *23rd International Conference on Pattern Recognition (ICPR),* pp. 2670-2675, 2016.

[31] B. Xiang, J. Tu, J. Yao and L. Li, “A Novel Octree-Based 3-D Fully Convolutional Neural Network for Point Cloud Classification in Road Environment,” *IEEE Transactions on Geoscience and Remote Sensing,* vol. 57, pp. 7799-7818, 2019.

[32] X. Roynard, J.-E. Deschaud and F. Goulette, “Classification of Point Cloud Scenes with Multiscale Voxel Deep Network,” 10 April 2018. [Online]. Available: https://arxiv.org/pdf/1804.03583.pdf. [Accessed 22 April 2020].

[33] D. Maturana and S. Scherer, “VoxNet: A 3D Convolutional Neural Network for real-time object recognition,” *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* pp. 922-928, 2015.

[34] C. R. Qi, L. Yi, H. Su and L. J. Guibas, “PointNet++: Deep Hierarchical Feature Learning on Point Sets in Metric Space,” 2017 June 2017. [Online]. Available: https://arxiv.org/pdf/1706.02413.pdf. [Accessed 26 April 2020].

[35] M. B. Soares and S. Wermter, “Point Cloud Object Recognition using 3D Convolutional Neural Networks,” *2018 International Joint Conference on Neural Networks (IJCNN),* pp. 1-8, 2018.

[36] “Puck,” Velodyne Lidar, [Online]. Available: https://velodynelidar.com/products/puck/. [Accessed 15 May 2020].

[37] L. Kasak, “Model Summary,” 11 May 2020. [Online]. Available: https://github.com/LiineKasak/3d-recognition/blob/master/model_summary.txt. [Accessed 11 May 2020].

[38] P. Ruiz, “DenseNets,” Aug 2018. [Online]. Available: http://www.pabloruizruiz10.com/resources/CNNs/DenseNets.pdf. [Accessed 15 May 2020].