Aalto University
School of Science

Master’s Programme in Mathematics and Operations Research

Unsupervised Deep Learning for Semantic Segmentation of Multispectral Forest Point Clouds

Lassi Ruoppa

Master’s Thesis
2025

Aalto University
School of Science

Author	Lassi Ruoppa
Title	Unsupervised Deep Learning for Semantic Segmentation of Multispectral Forest Point Clouds
Degree programme	Mathematics and Operations Research
Major	Systems and Operations Research
Supervisor	Senior University Lecturer Jorma Laaksonen
Advisors	Dr. Matti Lehtomäki, M.Sc. Josef Taher
Collaborative partner	Finnish Geospatial Research Institute
Date 24.02.2025	**Number of pages** 138+4 **Language** English

Abstract

Three-dimensional point clouds acquired with light detection and ranging scanners are used in various applications within forest inventory and plant ecology, such as classification of individual tree species and estimation of tree stem attributes and above-ground biomass. However, a majority of these applications require the semantic segmentation of the data into wood and foliage points, also known as leaf–wood separation. Traditionally, this has been achieved with geometry- and radiometry-based unsupervised algorithms, whose performance is heavily dependent on the density of the point clouds, and tends to be poor on sparser data. By contrast, more recent machine and deep learning-based leaf–wood separation approaches achieve great performance even on sparse point clouds but require manually annotated training data, which is laborious and time-consuming to produce.

To address the limitations of traditional leaf–wood separation algorithms on sparser point clouds while simultaneously not utilizing any manually labeled data, this thesis explores utilizing unsupervised deep learning for the semantic segmentation of multispectral forest point clouds. To this end, the GrowSP-ForMS model is proposed. The model has been adapted from the recent GrowSP architecture by introducing changes specifically designed to improve the accuracy of leaf–wood separation. To assess the accuracy of GrowSP-ForMS, a novel multispectral aerial laser scanning data set captured in boreal forest environment is introduced. A subsection of the data set is manually labeled into foliage and wood and subsequently divided into training and test data. GrowSP-ForMS is compared to two unsupervised algorithms and three fully-supervised neural network architectures. The results demonstrate that GrowSP-ForMS significantly outperforms unsupervised methods and performs comparably to an older supervised model, but is outclassed by more recent supervised approaches. Finally, two ablation studies demonstrate that the proposed changes significantly increase the accuracy of GrowSP-ForMS in comparison to the original GrowSP model and that utilizing multispectral data improves model performance from the monospectral case.

Keywords unsupervised deep learning, point cloud, multispectral LiDAR, semantic segmentation, leaf–wood separation, airborne laser scanning

Aalto-yliopisto
Perustieteiden korkeakoulu

Tekijä Lassi Ruoppa

Työn nimi Valvomaton syväoppiminen multispektristen metsäpistepilvien semanttiseen segmentointiin

Koulutusohjelma Matematiikka ja systeemitieteet

Pääaine Systems and Operations Research

Työn valvoja Vanhempi yliopistonlehtori Jorma Laaksonen

Työn ohjaajat TkT Matti Lehtomäki, DI Josef Taher

Yhteistyötaho Paikkatietokeskus

Päivämäärä 24.02.2025 **Sivumäärä** 138+4 **Kieli** englanti

Tiivistelmä

Laserkeilaimella kerättyjä kolmiulotteisia pistepilviä käytetään erilaisissa metsäinventointiin ja kasviekologiaan liittyvissä sovelluksissa, kuten yksittäisten puulajien luokittelussa, sekä puiden runkoattribuuttien ja maanpäällisen biomassan arvioinnissa. Suurin osa näistä sovelluksista edellyttää datan semanttista segmentointia puu- ja lehvästöpisteisiin, jota kutsutaan myös lehti–puu-erotteluksi. Perinteisesti tämä on toteutettu geometriaan ja radiometriaan perustuvilla valvomattomilla algoritmeilla, joiden suorituskyky riippuu suuresti pistepilvien tiheydestä ja on yleensä heikkolaatuinen harvemmalle datalle. Uudemmat kone- ja syväoppimispohjaiset menetelmät taas ovat erittäin tarkkoja myös harvoilla pistepilvillä, mutta vaativat manuaalisesti luokiteltua koulutusdataa, jonka tuottaminen on työlästä ja aikaa vievää.

Tässä työssä tutkititaan valvomattoman syväoppimisen hyödyntämistä multispektrimetsäpistepilvien semanttiseen segmentointiin. Työn tavoitteena on ratkaista perinteisten lehti–puu-erottelualgoritmien rajoitukset harvemmissa pistepilvissä, ilman että samalla hyödynnetään manuaalisesti luokiteltua koulutusdataa. Tähän tarkoitukseen käytetään GrowSP-ForMS-mallia, joka on muokattu tuoreesta GrowSP-arkkitehtuurista lisäämällä lehvästö- ja puupisteiden erottelun tarkkuuden parantamiseen suunniteltuja parannuksia. GrowSP-ForMS:n tarkkuuden määrittelemiseen hyödynnetään uutta pohjoisessa metsäympäristössä multispektrilaserkeilaimella kerättyä datajoukkoa. Osa datasta luokitellaan manuaalisesti lehvästö- ja puupisteisiin, minkä jälkeen se jaetaan koulutus- ja testijoukkoihin. GrowSP-ForMS:ia verrataan kahteen valvomattomaan algoritmiin ja kolmeen valvottuun neuroverkkoarkkitehtuuriin. Tulokset osoittavat, että GrowSP-ForMS on tarkkuudeltaan huomattavasti valvomattomia menetelmiä parempi ja verrattavissa vanhempaan valvottuun malliin, mutta kaukana uudemmista valvotuista arkkitehtuureista. Lopuksi kaksi ablaatiotutkimusta osoittavat, että ehdotetut muutokset kasvattavat GrowSP-ForMS:n tarkkuutta merkittävästi alkuperäiseen GrowSP-malliin verrattuna ja että multispektridatan hyödyntäminen parantaa mallin suorituskykyä monospektriseen verrattuna.

Avainsanat valvomaton syväoppiminen, pistepilvi, multispektrinen LiDAR, semanttinen segmentointi, lehti–puu-erottelu, ilmalaserkeilaus

Preface

Kiitos perheenjäsenilleni, ystävilleni ja kaikille muillekin, jotka ovat olleet (enemmän tai vähemmän) tukenani tämän diplomityön työstämisessä ja ylipäätään opintojeni aikana. Kiitos myös diplomityöni ohjaajille ja valvojalle hyödyllisestä palautteesta ja neuvoista. Lopuksi vielä kiitos koko FGI:n AMAD tutkimusryhmälle ja sen johdolle. Arvostan sitä, että "luotitte visiooni", vaikka oli alkuun epävarmaa saisiko tästä aiheesta aikaiseksi yhtään mitään julkaisukelpoista. Niin kuin joku joskus sanoi: **"Ei se oo niin vakavaa."**

If you happen to be actually interested in the scientific contents of this thesis, I recommend reading the paper "Unsupervised deep learning for semantic segmentation of multispectral LiDAR forest point clouds" instead. The paper contains the same information presented much more concisely and all experiments were conducted again on better hardware and a larger data set. The preprint is available at: https://doi.org/10.48550/arXiv.2502.06227

Otaniemi, 24.02.2025

Lassi Ruoppa

Contents

Symbols and Abbreviations

Symbols

$\mathcal{B}$	Minibatch
C	Set of classes $\{1, \ldots, c\}$
$\mathbb{Z}$	The set of integers
$\mathbb{R}$	The set of real numbers
$\mathbf{\Theta}$	Network parameters
$\mathbf{W}$	Layer weight matrix
y_i	Ground truth label of point i
$\hat{y}_i$	Predicted class of point i
$\oplus$	Concatenation
$g(\cdot)$	Activation function
$\log(\cdot)$	Natural logarithm
$\tanh(\cdot)$	Hyperbolic tangent function
$\sigma(\cdot)$	Sigmoid function
$\mathbb{1}(\cdot)$	Indicator function
$\lvert \cdot \rvert$	Cardinality of a set
$Q_i(\cdot)$	The ith sample percentile

Abbreviations

2D	Two-dimensional
3D	Three-dimensional
Adam	Adaptive Moment Estimation
AGB	Above-Ground Biomass
ALS	Aerial Laser Scanning
CMD	Cross-Modal Distillation
CNN	Convolutional Neural Network
CRF	Conditional Random Field
CW	Continuous Wave
DBH	Diameter at Breast Height
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DL	Deep Learning
DTM	Digital Terrain Model
EdgeConv	Edge Convolution
FGI	Finnish Geospatial Research Institute
FPN	Feature Pyramid Network
GAFPC	Geometric-Based Automatic Forest Point Classification
GBS	Graph-Based Leaf–Wood Separation
GCNN	Graph Convolutional Neural Network
GMM	Gaussian Mixture Model
GNSS	Global Navigation Satellite System
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
IMU	Inertial Measurement Unit
IoU	Intersection over Union
IQR	Interquartile Difference
ISPRS	International Society for Photogrammetry and Remote Sensing
k-NN	k Nearest Neighbors
KPConv	Kernel Point Convolution
LeakyReLU	Leaky Rectified Linear Unit
LESS	Label-Efficient Semantic Segmentation
LFA	Local Feature Aggregation
LiDAR	Light Detection and Ranging
LocSE	Local Spatial Encoding
mAcc	Mean Accuracy
mIoU	Mean Intersection over Union
ML	Machine Learning
MLP	Multilayer Perceptron
MLS	Mobile Laser Scanning
MS	Multispectral
NN	Neural Networks
oAcc	Overall Accuracy

OE	Orientation Encoding
PCA	Principal Component Analysis
PCT	Point Cloud Transformer
PFH	Point Feature Histogram
pp	Percentage Point
PReLU	Parametric Rectified Linear Unit
PSD	Perturbed Self-Distillation
ReLU	Rectified Linear Unit
ResNet	Residual Network
RNN	Recurrent Neural Network
SCNN	Sparse Convolutional Neural Network
SGD	Stochastic Gradient Descent
SPG	Superpoint Graph
SQN	Semantic Query Network
SSC	Submanifold Sparse Convolution
SVM	Support Vector Machine
TLS	Terrestial Laser Scanning
ToF	Time-of-Flight
UAV	Unmanned Aerial Vehicle
VCCS	Voxel Cloud Connectivity Segmentation
XOR	Exclusive Or

1 Introduction

Three-dimensional (3D) point clouds acquired with light detection and ranging (LiDAR) scanners have a wide variety of applications within forest inventory and plant ecology, including estimation of tree stem attributes, leaf angle distribution and above-ground biomass (AGB) [1]. However, a majority of these applications require accurately classifying each point into either wood or foliage as a preprocessing step. This semantic segmentation task is often referred to as *leaf–wood separation*.

Traditionally leaf–wood separation has been achieved with unsupervised algorithms that utilize either geometric or radiometric properties of the point cloud, but following the recent major advancements in deep learning (DL) the amount of research on DL-based approaches has exploded [2, 3, 4, 5, 6]. Although the DL-based methods achieve extremely high segmentation accuracies, unlike unsupervised leaf–wood separation algorithms they require large amounts of manually generated training data with point-wise semantic annotations, which is extremely time-consuming and challenging to produce. In fact, a recent review of deep learning applications for forest inventory and planning found that acquiring sufficient amounts of representative and labelled training data remains one of the main challenges in the field [7].

Unsupervised leaf–wood separation algorithms are generally designed for point clouds captured with stationary terrestrial laser scanning (TLS) systems. While TLS is able to capture tree-structures with millimeter level details, aerial laser scanning (ALS) allows surveying larger areas spanning multiple square kilometers much more time efficiently at the cost of reduced point density and geometric details. Consequently, conventional unsupervised algorithms often struggle with ALS data due to heavy reliance on point cloud geometry.

Since generating semantic annotations for point cloud data is extremely laborious, weakly-supervised approaches to semantic segmentation that minimize the amount of required ground truth data have long been a research focus. Multiple works have demonstrated that segmentation accuracies comparable to many fully-supervised architectures can be achieved with as little as 0.01% of the ground truth labels [8, 9]. More recently, Zhang et al. [10] introduced the first semantic segmentation model based on unsupervised deep learning, which achieved competitive results on multiple benchmark data sets depicting indoor scenes and urban environments.

LiDAR scanners typically only capture intensity information from one wavelength. However, more recently multispectral (MS) laser scanning systems that operate on multiple distinct wavelengths have emerged as a promising new alternative [11]. MS point clouds have been shown to improve accuracy in tasks such as tree species classification [12] and land cover classification [13]. Li et al. [14] hypothesized that MS data could also improve the accuracy of leaf–wood separation due to the distinctive difference between leaf and wood reflectance on certain wavelengths. As such, the additional intensity information provided by multispectral scanners may help improve the accuracy of unsupervised deep learning models in the task of leaf–wood separation.

1.1 Research Objectives

The objective of this thesis is two answer the following research questions:

1. Can unsupervised DL be utilized to alleviate the issue of reduced point density in ALS data, which causes the performance of conventional leaf–wood separation algorithms designed for TLS point clouds to deteriorate? In other words, does an approach based on unsupervised deep learning outperform traditional algorithms when it comes to ALS data?

2. How does the performance of an unsupervised DL model compare to fully-supervised approaches in the task of leaf–wood separation on multispectral ALS data.

3. Does utilizing multispectral LiDAR with intensity information from multiple wavelengths improve the semantic segmentation performance of unsupervised DL models for forest data?

To answer these questions, we adapt the recent GrowSP [10] model for unsupervised semantic segmentation of point clouds to ALS forest data by introducing modifications specifically designed to improve leaf–wood separation accuracy. The adapted model is then trained on a novel multispectral forest data set and compared to existing leaf–wood separation approaches.

1.2 Contributions

The contributions presented in this thesis are as follows:

1. We propose the world's first multispectral point cloud data set with manual point-level annotations captured with an aerial laser scanning system in a boreal forest environment. The data set is comprised of more than 8 million individual points, each of which has been manually labeled into either wood or foliage.

2. We present a modified version of the recent GrowSP [10] deep learning architecture designed specifically for unsupervised semantic segmentation of multispectral forest point clouds, named **GrowSP-ForMS** (**GrowSP** for **For**est area **m**ulti**s**pectral point clouds).

3. We present an extensive performance comparison between GrowSP-ForMS, two state-of-the-art unsupervised algorithms and three popular fully-supervised deep learning models on the leaf–wood separation task on our proposed multispectral forest environment data set. The comparison demonstrates that GrowSP-ForMS achieves performance that is state-of-the-art in unsupervised leaf–wood separation and comparable to fully-supervised PointNet [15]. All models are compared both qualitatively and quantitatively.

4. We perform an ablation study where the effect of our proposed changes to GrowSP is assessed. By comparing the performance of ablated models we

demonstrate that all of our proposed modifications are beneficial in terms of leaf–wood separation accuracy.

5. The performance of GrowSP-ForMS is assessed on monospectral LiDAR point clouds from multiple distinct wavelengths as well as multispectral data. Our results suggest that using intensity information from multiple wavelengths considerably improves the accuracy of our unsupervised semantic segmentation model when used correctly.

1.3 Structure

This thesis begins with a brief overview of point cloud data and publicly available data sets, basic principles of artificial neural networks, deep learning-based point cloud semantic segmentation and finally various leaf–wood separation methods in Section 2. In Section 3, we describe our manually annotated multispectral forest data set and the data preprocessing pipeline. We begin Section 4 with a detailed description of the GrowSP architecture, our proposed modifications and each of our fully-supervised and unsupervised baseline methods, followed by details of the experimental setup and evaluation metrics, including the procedures used for training the deep learning models and optimizing the hyperparameters of the unsupervised baselines. Subsequently, in Section 5 we present the results of our experiments comprised of a quantitative and qualitative performance comparison between our model and the baseline architectures as well as an ablation study. Finally, Section 6 discusses the results and limitations of this work along with possible directions for future research, followed by the conclusions of the thesis in Section 7.

2 Background

We begin by introducing the concept of point clouds and subsequently showcase some publicly available point cloud data sets in Section 2.1, focusing specifically on the most widely used semantic segmentation benchmark data sets and data sets collected in forest environments. Section 2.2 provides an overview of deep learning, neural networks and their architectural elements, followed by an overview of their training procedures in Section 2.3. In Section 2.4 we give a brief introduction to deep learning-based semantic segmentation of point clouds, highlighting some of the most notable works in the field, grouped by the degree of supervision in training. Finally, in Section 2.5, we discuss the more specific semantic segmentation task of leaf–wood separation in point clouds, including both machine learning (ML) and deep learning-based approaches as well as more traditional geometry and radiometry-based algorithms.

2.1 Point Cloud Data Sets

2.1.1 Point Cloud Data

A three-dimensional point cloud is a collection of points set in a 3D space, which represent the surface of either an individual object or a larger scene [16]. In general the number of points in a point cloud is relatively large, usually at least tens to hundreds of thousands of points, but larger scans depicting e.g. urban environments may contain tens of millions or even billions of points. Each point consists of at least the xyz-coordinates, which define its unique location in the 3D space. In addition, other information such as RGB color or surface reflectance may be stored depending on the method of capture. In contrast to two-dimensional images or voxel-based 3D models, the xyz-coordinates of point clouds can not be represented as a discrete grid-like structure. Rather, they are an inherently unorganized collection of points, which presents an unique set of challenges when it comes to effectively processing point cloud data [16].

A wide variety of point cloud data sets are publicly available for the purposes of benchmarking model performance on various tasks, such as instance segmentation, semantic segmentation, 3D object classification, parts segmentation and 3D reconstruction. Point cloud datasets can broadly be divided into two categories: synthetically created point clouds and point clouds captured in the real world. Synthetic data sets are typically created by sampling points from computer generated 3D models and their main advantage over real world data is that the manual labelling process is considerably less laborious and sometimes even completely unnecessary. Popular examples of synthetic point cloud data sets include ModelNet [17], comprised of 151,128 3D models across 660 common object categories, and ShapeNet [18], which contains roughly 3,000,000 models, 220,000 of which are classified into 3,135 categories. For both ModelNet and ShapeNet, a smaller subset of the full data set is typically used for benchmarking point cloud classification. These data sets are known as ModelNet40 and ShapeNetCore and contain 12,311 models across 40 classes and 51,300 across 55 classes respectively [17, 18].

Point clouds acquired from real world can be further divided into separate classes based on the type of sensor used. Popular methods include photogrammetric point clouds, RGB-D camera based point clouds and finally, point clouds captured using LiDAR sensors. The former two are somewhat similar: a photogrammetric point cloud is constructed by triangulating three-dimensional shape from multiple overlapping images [19]. On the other hand RGB-D cameras capture the depth in images or video using an additional sensor. One such method for capturing depth is time-of-flight (ToF), where the distance of an object from the camera is computed from the travel time of light emitted by a separate illumination unit [20]. We note that ToF based depth measurement is in fact LiDAR based, thus many RGB-D cameras are effectively a fusion between photogrammetry and LiDAR. Both photogrammetric methods and RGB-D cameras initially provide a 3D mesh, which can then be transformed into a point cloud by subsampling points along the surface.

LiDAR Data: LiDAR sensors have two main advantages over most other range sensing solutions. Firstly, they are exceptionally accurate in comparison to sensors such radars or cameras [21], achieving millimeter level accuracy at close ranges and maintaining a centimeter level accuracy even at long distances [22]. Secondly, LiDAR performance is not affected by the ambient lighting conditions, since the range sensing is based on active illumination with a laser. This is in contrast to systems relying on passive illumination, such as cameras, for which change of illumination can have a considerable impact on performance [21].

A LiDAR sensor consists of a transmitter and a receiver. The transmitter contains a laser, which sends a light beam with specific spectral properties toward the target. In addition, the transmitter will often contain a beam expander, which is applied on the beam to reduce its divergence. Within the receiver the backscattered photons are collected using a telescope, followed by an optical analyzing system which selects specific wavelengths from the collected light. Finally, the selected radiation is directed onto a detector and subsequently converted from an optical signal into an electrical signal [23]. The principle setup of a LiDAR system is visualized in Figure 1.

LiDAR systems can be divided into two general categories based on the method of obtaining the location of the target: pulsed LiDAR and continuous wave (CW) LiDAR [24]. Pulse-based LiDARs send a short light pulses with lengths ranging from few to several hundred nanoseconds [23] and estimate the distance of the target based on the time of flight from the sensor to the target and back [24]. On the other hand, CW LiDAR systems emit a constant signal while simultaneously maintaining a reference signal, also known as a local oscillator [24]. The distance of the target is then derived by modulating the amplitude or the frequency of the signal and comparing it to the local oscillator [24].

In addition to the spatial location of the target, LiDAR sensors capture the amplitude information of the returning signal, from which reflectance of the target can be estimated. In fact, some of more advanced LiDAR systems automatically deliver calibrated amplitude and reflectance information. Furthermore, the systems usually collect the return number of each pulse, since there can be several target returns when

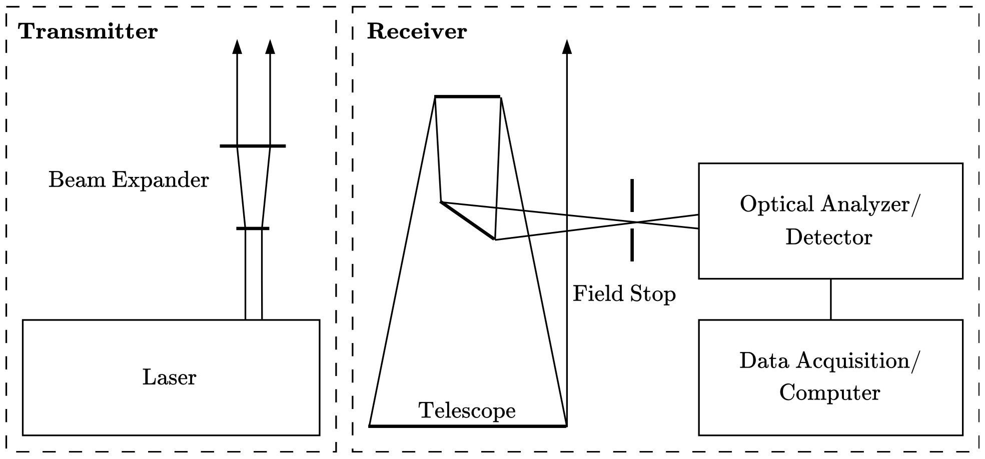


Figure 1: Principle setup of a LiDAR system. The figure has been reproduced based on [23].

a laser beam hits more than one target. This can happen for example in situations when a pulse penetrates vegetation. Finally, some LiDAR sensors are capable of collecting secondary data, such as the echo deviation of the pulse [25].

Although the reflectance values across different wavelengths may contain useful information [25], LiDAR sensors generally only capture information from a single wavelength, typically within the range [250 nm, 11 µm] [23]. A LiDAR system capable of recording data at multiple wavelengths is known as *multispectral* LiDAR [25]. It is important to note that multispectral specifically refers to two or more distinct wavelengths, which are usually considerably separated. This is contrast to *hyperspectral* LiDAR sensors, which cover a certain wavelength range with multitude of, usually equally spaced, channels [25]. Multispectral LiDAR data is currently still somewhat rare, although in recent years sensors, such as the Optech Titan[1], capable of operating at multiple wavelengths have become commercially available.

LiDAR data can be divided into three main categories based on the method of collection, namely aerial laser scanning (ALS), mobile laser scanning (MLS) and terrestrial laser scanning (TLS) [22], each of which comes with its inherent advantages and challenges. ALS is based on conducting LiDAR measurements from an aircraft, where the exact location and orientation of the sensor is collected simultaneously using a navigation system, generally comprised of a global navigation satellite system (GNSS) receiver and an inertial measurement unit (IMU) [22]. Mobile laser scanning utilizes the same method for georeferencing, and while ALS and MLS data are both also collected from a LiDAR sensor on a moving platform, the crucial difference is that mobile laser scanners operate at ground level and are usually attached to a motor vehicle or e.g. a backpack [26]. Conversely, terrestrial laser scanning is quite different from the other two methods, since the sensor remains stationary. Consequently, TLS is considerably more accurate than both ALS and TLS [22].

[1]`https://www.geo3d.hr/3d-laser-scanners/teledyne-optech/optech-titan`

The choice of the collection method naturally depends on the use case of the LiDAR point clouds. ALS is convenient for covering broad regions relatively quickly and is the most cost-effective for areas larger than 50 km^2 [22]. However, due to the distance of the sensor, the point density of ALS data is generally lower, although this can be controlled to a certain degree by altering the flight altitude. Furthermore, due to the distance, the point clouds are less accurate as the footprint of the laser is wider [26]. On the other hand, MLS data provides more accurate and denser point clouds, but the density can suffer from relatively high variation [22] due to e.g. inconsistent movement speed during collection. Furthermore, at the ground level, GNSS coverage can be poor in certain conditions [26], such as under the canopy of a dense forest, which complicates the georeferencing of MLS point clouds at times. Finally, TLS provides the most accurate and dense point clouds, but at the cost of considerably smaller range [22]. If a larger area is to be covered using TLS, the LiDAR system has to be relocated manually.

2.1.2 Semantic Segmentation Benchmark Data Sets

Semantic segmentation refers to the task of separating an object or a scene into semantic classes, which are simply regions that are similar according to some criterion. In the context of point clouds, semantic segmentation entails assigning a class for each individual point. In contrast to *instance segmentation*, which is concerned with separating distinct objects from the point cloud, two points belonging to different entities of the same class should have the same semantic label. Furthermore, an object instance may contain multiple semantic classes. For example, we might classify the roofs and walls of buildings into distinct classes.

In order to compare the performance of semantic segmentation models designed for point clouds as objectively as possible, various benchmark data sets have been proposed over the years. These benchmark data sets are generally either indoor environments collected using some type of RGB-D camera, or outdoor MLS data [27]. The former are typically slightly easier to segment as indoor environments are inherently more structured. Furthermore, the point density in indoor environments is quite consistent, while the varying point density of outdoor LiDAR data presents an additional challenge [28].

While benchmark data sets are excellent for comparing the performance of various semantic segmentation approaches, they have some noteworthy limitations:

1. Most of the popular benchmarks seemingly concentrate on a very restricted set of data sources, specifically indoor scenes and urban environments. Such benchmarks are not suitable for assessing e.g. leaf–wood separation performance.

2. The benchmark data sets are generally captured with generic commercially available systems. As such, if research focus is on investigating the potential of more novel data capturing methods, for example multispectral LiDAR, semantic segmentation benchmarks are often not a feasible choice.

Nevertheless, a basic understanding of widely utilized benchmarks is quite beneficial

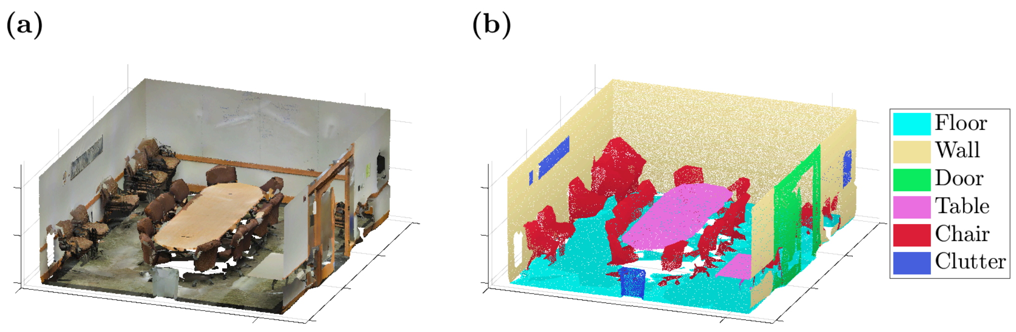


Figure 2: Section of a point cloud from the S3DIS data set [30]. (a) Point cloud of a conference room collected with an RGB-D camera. (b) Manual class annotations superimposed on the point cloud.

e.g. when it comes to choosing which semantic segmentation architecture to utilize for one's own data.

Here we provide a short overview of some of the most popular semantic segmentation benchmark data sets. For a more comprehensive survey of existing data sets, the reader is referred to [27, 28].

Indoor Data Sets: SUN RGB-D was one of the first large-scale indoor 3D data sets to achieve a scale comparable to modern image recognition data sets. Captured using four different RGB-D sensors, SUN RGB-D is comprised of 10,335 manually labeled RGB-D images across 800 object categories. The images contain both 2D and 3D semantic and instance annotations. However, the 3D labels are based on bounding boxes and thus contain some inherent imperfections. Furthermore, the point clouds in SUN RGB-D are based on RGB-D images, that is, they have been captured from one direction only and thus contain occluded regions [29].

At present, S3DIS [30] and ScanNet [31] are by far the two most popular indoor benchmarking data sets. Both data sets are widely utilized not only for comparing the performance of semantic segmentation models, but also other tasks such as instance segmentation and object detection. The former data set consists of high density RGB-D camera based point clouds from 271 rooms across 6 large indoor areas S3DIS [30]. Each point has been manually assigned into one of 13 semantic classes, such as ceiling, floor, window or chair [30]. An example of the semantic annotations is shown in Figure 2.

ScanNet is quite similar to S3DIS, although slightly larger. Consisting of 1,513 RGB-D camera scans across 707 distinct indoor spaces, ScanNet contains manual semantic annotations for 20 classes as well as instance level annotations for each object. In contrast to S3DIS however, the scans in ScanNet are provided as reconstructed 3D surfaces rather than point clouds [31].

Outdoor Data Sets: A large majority of publicly available outdoor data sets are designed for autonomous driving applications. Such data sets are generally captured in urban areas with a car-mounted MLS system, thus the principal differences between distinct data sets come from the utilized LiDAR system and the amount of annotated data. One of the first semantic segmentation benchmarks for autonomous driving was the Oakland 3D data set [32]. Collected with a SICK LMS scanner in Pittsburgh, USA, Oakland 3D is comprised of 1.3 million points that have been hand-labeled into 44 distinct classes, 5 of which are used for benchmarking [32]. Other notable examples of MLS data sets include Sydney Urban Objects [33], Paris-rue-Madame [34] and Paris-Lille-3D [35].

At present, the most widely utilized outdoor semantic segmentation benchmark is the SemanticKITTI [36] data set. Based on the KITTI [37] data set, SemanticKITTI is comprised of 4.5 billion points that have been manually assigned to 28 semantic classes, 25 of which are used for benchmarking purposes. At the time of its release, SemanticKITTI was by far the most extensive MLS benchmarking data set [36], which is likely a contributing factor in its popularity.

Another noteworthy development in the field of autonomous driving is the introduction of multimodal benchmark data sets, which include data from other sensors in addition to LiDAR point clouds. One such data set is nuScenes, which includes RGB images, as well as both LiDAR and radar based point clouds [38]. The scale of nuScenes is similar to SemanticKITTI, as it contains 1.4 billion manually annotated points across 32 classes [38]. In contrast to most benchmarks, nuScenes also features scans from various different weather and illumination conditions [38].

Apart from autonomous driving applications, benchmarking of semantic segmentation models on remote sensing data has a long history within the geospatial data community, especially the International Society for Photogrammetry and Remote Sensing (ISPRS) [39]. The point cloud based ISPRS benchmarking efforts are generally focused on aerial data. Perhaps the most notable example of such data sets is the Vaihingen 3D benchmark (V3D) [40], which was constructed by adding manual 3D annotations to an earlier data set collected in 2010 using a Leica ALS50 system [41]. As the point clouds are relatively sparse, V3D consists of around 780,000 points, manually divided into 7 semantic classes [40].

A more recent example is the Hessingheim 3D benchmark (H3D) [42], which is multimodal data set comprised of LiDAR point clouds and RGB colored 3D meshes representing an urban area. H3D was captured from an unmanned aerial vehicle (UAV) equipped with a RIEGL VUX-1LR scanner and two oblique Sony Alpha 6000 cameras and contains 73.9 million manually annotated points across 10 semantic classes [42].

In addition to aerial data sets, TLS based semantic segmentation benchmarks are also available. A prominent example is the Semantic3D.net data set which is comprised of 30 individual scans of urban scenes captured in Central Europe. In total, Semantic3D.net contains over 4 billion points, each manually assigned to one of 8 classes. An example of the semantic annotations is shown in Figure 3. The high point density and large occlusions inherent to TLS data present an unique challenge, rendering Semantic3D.net an especially interesting benchmark [39].

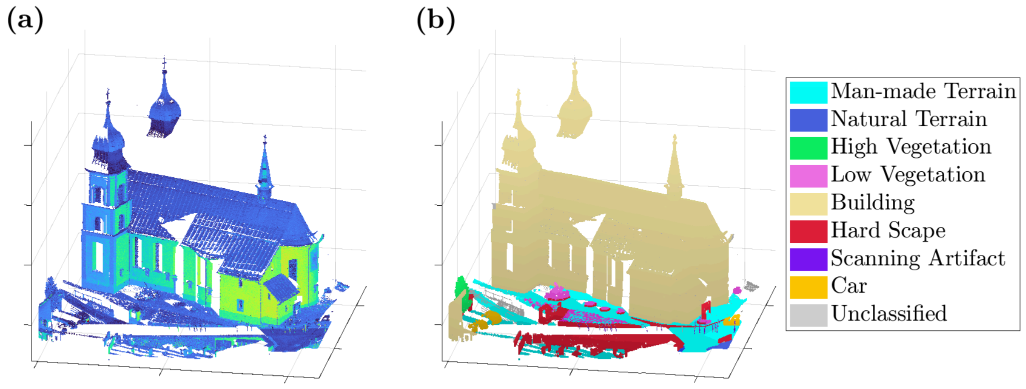


Figure 3: A section from one of the 30 TLS point clouds in the Semantic3D.net data set [39]. (a) TLS point cloud from an outdoor scene. The colors represent reflectance values. (b) Manual class annotations superimposed on the point cloud.

2.1.3 Forest Environment Data Sets

Although LiDAR-based methods have been a staple in research concerning collecting and maintaining forest inventories for years now [6, 22], the number of openly available large-scale data sets with high-quality annotations remains exceedingly low. In fact, most studies opt to use their own small-scale data sets collected using just one type of sensor, which is an issue because methods developed on single-sensor data sets of limited size often generalize poorly for any other type of data [43]. In addition, most of these data sets are proprietary, which further complicates objectively comparing the results of different studies [43]. For example, a recent short review concerning deep learning methods for instance and semantic segmentation of forest point cloud data found the used data is not publicly available in a majority of the cases [6]. Based on these observations, Lines et al. [43] suggested that the lack of openly available curated benchmarking data sets is significantly hindering the development of deep learning methods for forestry applications, such as instance segmentation of individual trees, semantic segmentation of forest point clouds and tree species classification. The most likely explanation for the small number of large-scale forest data sets is that manually labeling forest point clouds is exceedingly time consuming, as noted by e.g. [4, 44]. Even 3D point clouds depicting simpler environments such as indoor scenes are extremely laborious to annotate and the intricate geometry of trees and overlapping of adjacent tree crowns makes manual segmentation of forest data significantly more challenging.

The Cameroon individual tree data set [45, 46] was one of the first openly available TLS forest data sets with semantic annotations, where each point has been manually labeled as either wood or foliage. The data set consists of 61 individual deciduous trees across a wide variety of species and was collected in Eastern Cameroon using a Leica C10 Scanstation [47]. The data set was originally designed for estimating the volumes and above-ground biomass (AGB) of the woody parts of tropical trees [47], but has since been successfully utilized for training a deep semantic segmentation network [48]. Nevertheless, the authors noted that a more diverse data set is required in order to develop a robust model for leaf–wood separation [48].

The data set of Kaijaluoto et al. [4] is another notable publicly available forest environment data set with semantic annotations. Collected with the Finnish Geospatial Research Institute (FGI) Akhka-R3 backpack laser scanning system and designed specifically for DL applications, the data set is comprised of 202 million points from two 32 m by 32 m test sites that have been manually divided into ground, understory vegetation, tree trunk and foliage. In contrast to the Cameroon individual tree data set, [4] opts to classify branches as foliage, while the tree trunk is assigned into its own class.

Publicly available forest point cloud data sets often suffer from either limited sample size or lack of diversity in environments and utilized sensors. In addition, the quality or type of the available manual annotations may be unsuitable for most deep learning applications. For example, the FGI international TLS benchmarking dataset [49] is a high quality and large-scale data set captured using Leica HDS6100 scanner, containing thousands of trees across 24 forest plots in Evo, Finland [49]. However, while manual reference measurements such as the location, height and diameter at breast height (DBH) of individual trees are included, the TLS point clouds have not been manually labeled or segmented. As such, the data set cannot be utilized in training supervised deep learning models for tasks such as instance segmentation of individual trees or semantic segmentation of forests. On the other hand, some national agencies publish large annotated LiDAR data sets [42], which often include forested areas. However the class division and annotations in such data sets are generally quite coarse [42]. As an example, individual points in the NASA G-LiHT data set are simply divided into ground points and other points [50]. Similarly, Finnish national laser scanning data (5 p/m^2) contains slightly more refined class labels, such as ground and low and high vegetation, but the points have been classified with an automated algorithm, rather than manually [51].

In recent years, multiple large scale individual tree data sets have been introduced, namely NeonTreeEvaluation [52], LAUTx [53] and the German individual tree point clouds and measurements [54, 55], which we refer to as GITPCM for the sake of brevity. However, due to the extremely laborious manual segmentation process required for producing high quality individual tree point clouds, many of these data sets utilize some level of automation during the annotation process, which introduces an inherent bias to the data.

NeonTreeEvaluation [52], by far the largest of the three, is a multi-sensor benchmarking data set for detecting individual trees collected by the National Ecological Observation Network, consisting of co-registered RGB and hyperspectral images and LiDAR point clouds. In total, the data set contains 30,975 individual tree annotations across 22 sites and a variety of ecosystems. The point clouds contain labels for each individual tree, and semantic labels for tree points and other points. It is worth noting that the individual tree annotations were initially 2D bounding boxes created for the RGB images, which were then draped over the point cloud. As such, some erroneously labeled points are to be expected, especially near the edges of the tree crowns [52].

Similar to NeonTreeEvaluation, the GITPCM data set [54, 55] consists of co-registered forest data from multiple sensors. More specifically, the data set consists terrestrial, aerial and UAV laser scanning point clouds, collected with RIEGL VZ-400, RIEGL VQ-780i and RIEGL miniVUX-1UAV scanners respectively. In total the data set is comprised of 1,491 individual tree point clouds, along with field measurements, such as DBH, for a majority of the trees. The individual trees were segmented using an automated algorithm, followed by a manual error correction process. Consequently, the authors note that some segmentation errors are present in the data.

Finally, while LAUTx [53] is the smallest of the three data sets comprised of only 515 individual trees, it evidently has the highest quality annotations. The point clouds were collected with a ZEB Horizon handheld laser scanner in Austria and each tree has been manually segmented with no automated steps.

The recently introduced FOR-Instance data set [44] is currently the most promising candidate for becoming the gold standard of benchmarking point cloud based forest inventory methods. FOR-Instance contains a total of 1,130 manually segmented individual trees across five distinct countries and forest environments. High-quality semantic annotations are also provided and each point has been classified into one of five categories: stem, woody branches, live branches (i.e. foliage), low vegetation or terrain. In addition, a small percentage of points have been left unannotated due to significant uncertainty between classes. An example of the semantic annotations in FOR-Instance is shown in Figure 4. In addition to manual point annotations, field measured DBHs are also provided for each individual tree [44, 56].

Although all point clouds in FOR-instance are ALS-based, varying point densities and scanners are also represented, as the flight altitude and LiDAR system differ between the countries present in the data set. In contrast to most forest LiDAR data sets, FOR-Instance provides a split between training and test data along with a benchmarking guide, rendering it machine-learning-ready for a multitude of segmentation, classification and prediction tasks. The authors note that even though the data annotation was carried out extremely meticulously over a period of six months, some errors are still present in the data, since accurately interpreting forest point cloud scenes remains rather challenging. Notable error sources include separation of wood and foliage as well as ground and low vegetation [44].

2.2 Artificial Neural Networks

Artificial neural networks or simply neural networks (NNs) are a broad class of machine learning models that are capable of approximating highly non-linear functions. In recent years, machine learning research has been predominantly focused on deep NN-based models, as they have shown remarkable performance in tasks such as image recognition, natural language processing, autonomous driving and a wide variety of other applications [57].

The performance of traditional ML models is highly dependent on which representation is used for the input data [58]. That is, simply providing the raw input data will usually not yield great results. However, it is often difficult to know what kind

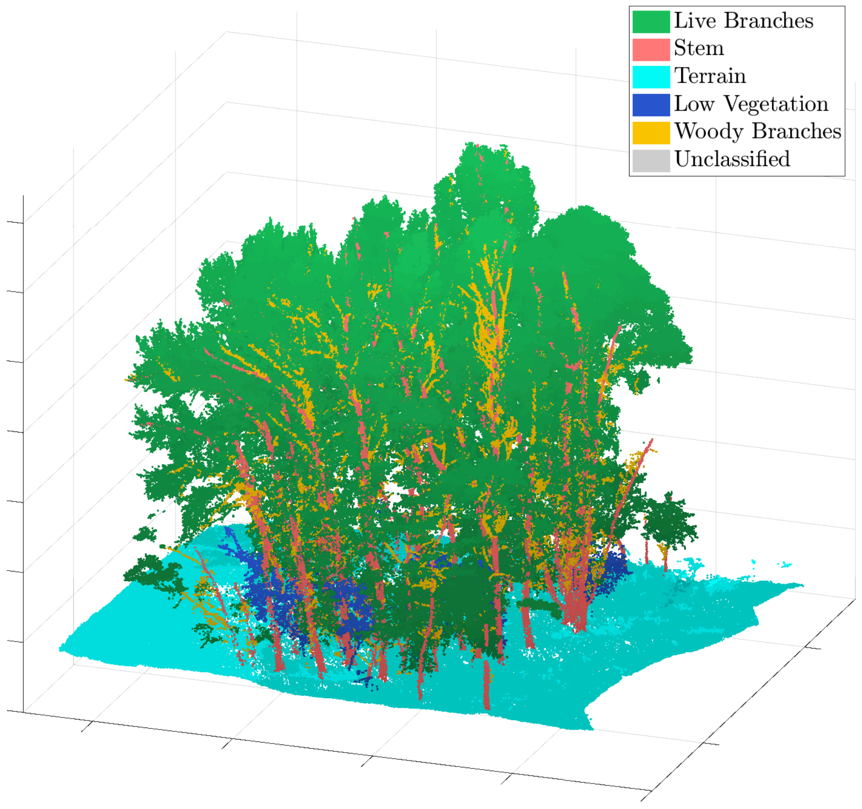


Figure 4: Sample from a plot in the FOR-Instance data set [56] showing the semantic annotations.

of features should be extracted from the data in order to produce the best results. As an example, image descriptors such as SIFT [59] and GLOH [60] are quite intricate and the result of a meticulous design process. On the other hand, conventional ML algorithms can not handle large amounts of high-dimensional data effectively, and their performance will often deteriorate as a result [57]. This issue is especially prominent nowadays, since large datasets data sets with quality labels have become increasingly available to the public [57].

Large enough artificial neural networks can evidently overcome many of the issues of traditional ML mentioned above: firstly, they are able to gradually extract high-level features from raw input data, effectively eliminating the need for complex handcrafted features [57]. Secondly, by utilizing the recent advancements in parallel computing and hardware such as graphics processing units (GPUs), NNs can efficiently handle immense amounts of high-dimensional data in comparison to conventional ML models [57]. In fact, NNs possess the theoretical capabilities to approximate any function mapping: Cybenko [61] showed that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate any well behaved function with any given accuracy.

While multiple distinct types of neural networks exist, *feedforward networks* are perhaps the most integral class of models, encompassing a variety of fundamental architectures, such as multilayer percepetrons (MLPs) and convolutional neural networks (CNNs). The term feedforward refers to how the networks process data: an input $\mathbf{x}$ flows sequentially through the layers that comprise the network. Recurrent neural networks (RNNs) are another type of network, characterized by the inclusion of *feedback connections*, which pass model outputs back into itself at various points. RNN-based long short-term memory architectures [62] have been quite successful in e.g. natural language processing tasks.

Artificial neural networks are named as they are because they were initially loosely inspired by neuroscience [58]. They can be though of as a collection of nodes and edges that connect them, analogous to the neurons and synapses in the brain. The network input then produces an activation signal that travels from node to node, until the output node produces a result corresponding to the function mapping the network approximates. The nodes in neural networks are known as artificial neurons and each one performs a transformation on the input signal using a distinctive set of parameters and some function mapping. The inherent challenge with neural networks is finding a set of parameters that produces the desired output. This process of optimizing the parameters of a neural network is referred to as *training* or *learning*, and involves iteratively updating the model parameters based on the model output and some cost function, generally using a gradient-based optimization method. Section 2.3 covers the process of training neural networks in more detail.

On the other hand, neural networks can be though of as a collection of functions $f_i(\cdot)$ consisting of multiple artificial neurons, also known as layers, that are sequentially applied on the input $\mathbf{x}$. As an example, the output of a network with three layers would then be $f_3(f_2(f_1(\mathbf{x})))$. The final layer in a neural network is referred to as the *output layer*, while the intermediate layers after the input are known as *hidden layers*, aptly named so, since they do not produce the final output and are thus hidden within the network. The *depth* of a network is the number of layers in it, while *width* refers to the number of neurons in a particular layer [58].

Deep learning is a term that is often used synonymously with artificial neural networks. The word deep simply refers to the multiple layers in most present-day artificial neural networks. The exact number of layers that constitutes a deep model is not widely agreed upon, although most researchers consider any model with two or more hidden layers to be deep [57]. While the architecture of a deep learning model certainly plays an important part in its performance, empirical evidence often suggests that deeper models with more layers and parameters tend to produce better results in a variety of tasks in comparison to their shallower counterparts [63]. It is however worth noting this is not always the case after a certain critical number of layers is reached [63]. As an example, Generative Pre-trained Transformer 3 (GPT-3), the predecessor to the GPT-3.5 and GPT-4 models behind the famous ChatGPT chatbot, has around 175 billion trainable parameters across 96 layers [64], while GPT-4 is estimated to contain around a 1,000 times more parameters [65]. Similarly, the top performing

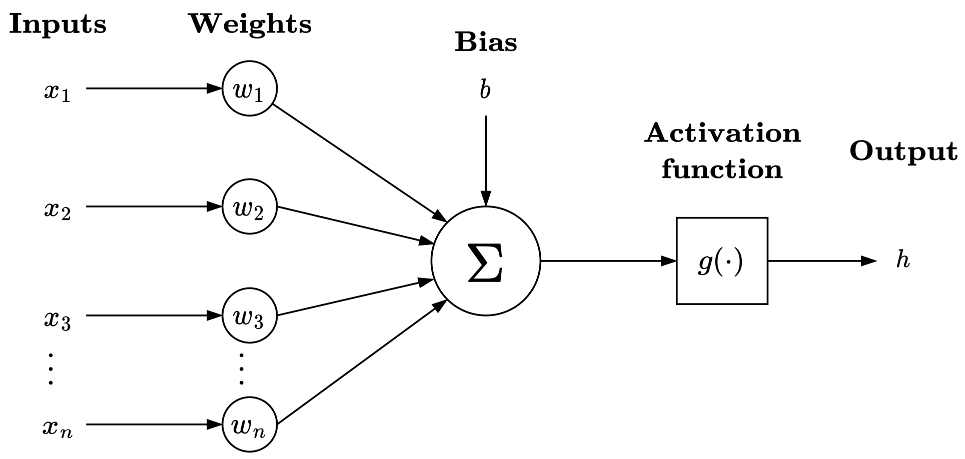


Figure 5: Diagram showing the structure of a typical artificial neuron. The values in the input feature vector x_i are multiplied by their respective trainable weights w_i and summed together with the bias term b. A non-linear activation function $g(\cdot)$ is then applied on the result to produce the final output h.

deep learning architectures [66] for point cloud semantic segmentation benchmark datasets such as S3DIS [30] and SemanticKITTI [36] are consistently both relatively deep and structurally complex [67, 68, 69, 70, 71, 72].

Finally, it should be noted that although deep neural networks solve many of the problems in traditional machine learning and can achieve remarkably impressive results in various tasks, they are not without their issues. A prominent issue is the *data hungriness* of deep learning models. That is, training of the best performing models requires massive amounts of, often manually labeled, data [73]. In addition, due to the sheer number of parameters, deep learning models are often uninterpretable to humans. Although some efforts have been made in visualizing the contributions of single neurons, the networks still generally remain black box models [73].

In the following subsections we present the fundamental architectural elements required for constructing artificial neural networks, with a specific focus on components that are most crucial for semantic segmentation of point clouds using deep learning.

2.2.1 Artificial Neuron

Artificial neurons are the most elementary building blocks of neural networks and each hidden layer will generally contain multiple, although the configuration of the neurons varies between different layer types. In modern day applications, an artificial neuron typically consists of two main components: Firstly, trainable parameters, which include both a weight vector $\mathbf{w} \in \mathbb{R}^n$ and a bias term $b \in \mathbb{R}$ and secondly a non-linear activation function, which we denote with $g(\cdot)$. The role and desired properties of the function are discussed in more detail in Section 2.2.2.

For an input feature vector $\mathbf{x} \in \mathbb{R}^n$, where n is the number of input features, the output h of an artificial neuron is given by:

$$h = g(\sum_{i=1}^{n} w_i x_i + b) = g(\mathbf{w}^\top \mathbf{x} + b). \tag{1}$$

More specifically, the neuron first computes an affine transformation of the input vector using the trainable weights $\mathbf{w}$ and the bias b, and subsequently applies a non-linear transformation $g(\cdot)$ on the result, as shown in Figure 5. The intermediate result $z = \mathbf{w}^\top \mathbf{x} + b$ is often referred to as the *linear activation*.

2.2.2 Activation Function

The output of a feedforward network with no non-linear activation functions is always equivalent to simply computing an affine transformation of the input data with appropriate parameters, regardless of the number of hidden layers and artificial neurons (see Section 2.2.3 for a short proof). That is to say, without activation functions, a deep learning model is only able to learn linear functions, which is a significant constraint.

Perhaps most famously, linear models are not able to represent the exclusive or (XOR) function. Given two binary values $x_1, x_2 \in \{0, 1\}$, the XOR function should return 1 if and only if exactly one of the two values is 1 and 0 otherwise. Formally we can define the XOR function for example as follows:

$$\text{XOR}(x_1, x_2) = \mathbb{1}(x_1 = 1) \cdot \mathbb{1}(x_2 = 0) + \mathbb{1}(x_1 = 0) \cdot \mathbb{1}(x_2 = 1), \tag{2}$$

where $\mathbb{1}(\cdot)$ is the indicator function. With an appropriate non-linear activation function, however, we can construct a feedforward network that is capable of learning the XOR function.

The non-linear activation functions within the artificial neurons are the fundamental source behind the flexibility of neural networks, that enables them to approximate highly non-linear functions. In addition, activation functions are also frequently used for scaling the output of a neural network to a specific range. For example, the output may be scaled to a range specific to some real world variable or the interval $[0, 1]$ to denote a probability.

Since a deep learning model will generally contain a large number of artificial neurons, a single neuron is not required to be able to represent highly complicated functions. Instead, the versatility of deep learning models is a result of all the neurons working as collective. Consequently, the choice of activation function is mostly dependent on differentiability and computational efficiency as opposed to expressive power. More specifically, a non-linear activation function should preferably possess at least the following three qualities:

1. The activation function should be at least piecewise differentiable throughout its domain, since the optimization methods used in the learning process are generally based on computing gradients.

2. Evaluating the activation function and the corresponding partial derivatives should be computationally efficient. This becomes increasingly important as the number of artificial neurons and hidden layers increases.

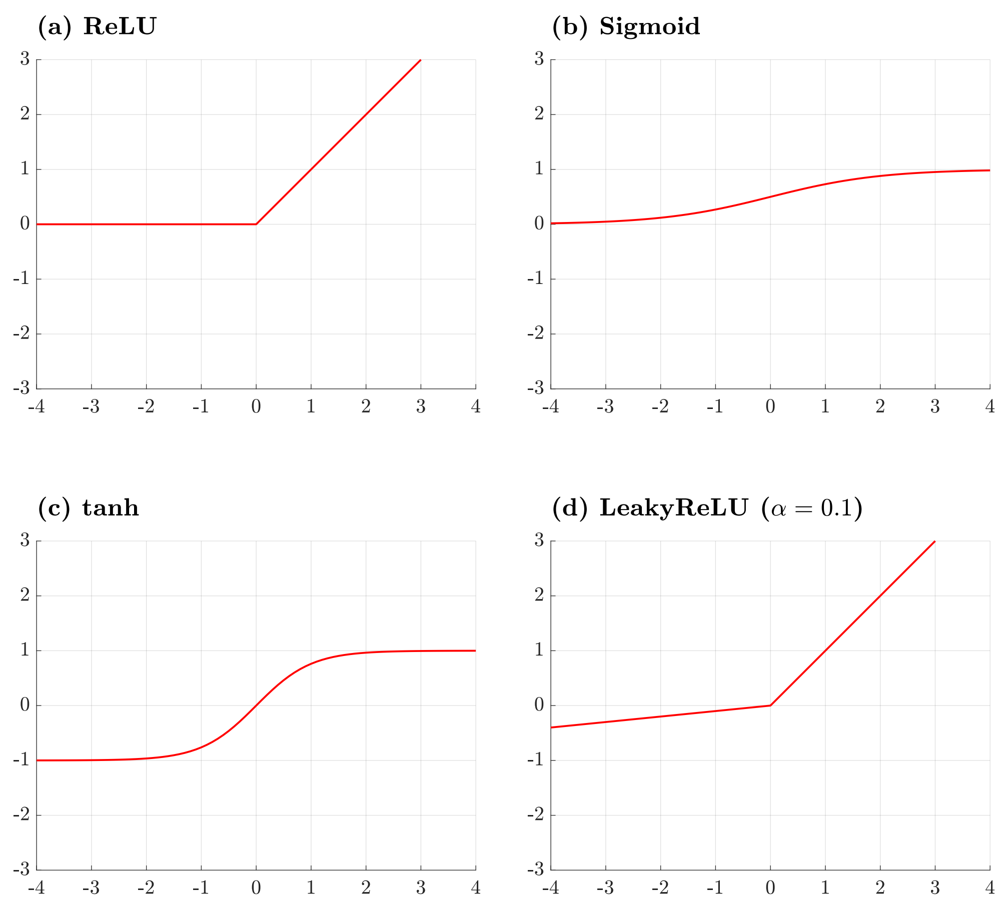


Figure 6: Visualization of some commonly used activation functions for the interval $x \in [-4, 4]$. (a) Rectified linear unit (ReLU). (b) Sigmoid function. (c) Hyperbolic tangent (tanh). (d) Leaky rectified linear unit (LeakyReLU) with the negative slope parameter $\alpha = 0.1$. We note that α has been fixed to an atypically high value in order to better illustrate the difference in comparison to the standard rectified linear unit.

3. The value of the gradient of the activation function should be within a reasonable range across the entire range of possible input values. The phenomenon where the gradients become either excessively small or large, also referred to as vanishing and exploding gradients, may significantly hinder the training process. Exploding gradients can make the learning unstable, as the network parameters may be updated to values far from the optimal range due to the large gradients. Conversely, vanishing gradients slow the training significantly and may even prevent it from converging at all [58].

In the subsequent sections we discuss some of the more popular activation functions that fulfill the above requirements in more detail.

Rectified Linear Unit: The rectified linear unit (ReLU) is currently the most widely used non-linear activation function [74], and the default recommended choice for most feed-forward networks [58]. The activation of rectified linear unit is given by:

$$\begin{aligned} \mathrm{ReLU} &: \mathbb{R} \to \mathbb{R}_0^+ \\ \mathrm{ReLU}(z) &= \max(0, z). \end{aligned} \tag{3}$$

Owing to its piecewise linear structure (see Figure 6 (a)), ReLUs are easy to optimize. In fact, the only difference between a linear unit and a ReLU is that the latter outputs 0 across half its domain. As such, the gradients through a rectified linear unit remain large and consistent whenever the unit is active, that is, the input is positive. Furthermore, the first derivative of the ReLU activation is 1 for all positive inputs and the second derivative is 0 almost everywhere within its domain. Consequently, the gradient is more useful for optimization when compared to activation functions that introduce second-order effects, such as the sigmoid function [58].

The rectified linear unit is differentiable across the entire set of real numbers, except for $z = 0$. While this might suggest that ReLU is incompatible with gradient based learning algorithms, this is not the case in practice, and gradient descent performs at an acceptable level, enabling the use of ReLU in deep learning models. This is largely because the learning algorithms are generally not expected to actually reach a local minima, i.e. a point where the gradient is a zero vector, which permits minima that correspond to points with undefined gradients. Software applications will typically simply use either the left derivative $\mathrm{ReLU}'_-(0) = 0$ or the right derivative $\mathrm{ReLU}'_+(0) = 1$ when $z = 0$ [58].

Sigmoid: In addition to ReLU, another common activation function is the sigmoid function, often denoted $\sigma(\cdot)$ and defined as follows:

$$\begin{aligned} \sigma &: \mathbb{R} \to [0, 1] \\ \sigma(z) &= \frac{1}{1 + e^{-z}}. \end{aligned} \tag{4}$$

The sigmoid function is an increasing function that maps real values to the interval $[0, 1]$, which are properties that make it a popular choice when the output is used for denoting probabilities.

The sigmoid function has a tendency to *saturate* when the linear activations are either very large negative numbers or very large positive numbers, which is relatively common in the context of hidden layers of deep learning models. Here saturation refers to the function output being very close to either one or zero. As a result, the gradient is practically zero when $\sigma(\cdot)$ is in a saturated state, which can significantly complicate or even halt the training process [58]. As can be seen in Figure 6 (b), when the linear activation z is not near zero, $\sigma(\cdot)$ is nearly flat and thus insensitive to small changes in input.

Other Activation Functions: Besides the sigmoid and ReLU, a multitude of other activation functions see frequent use in deep neural networks. Here we briefly discuss two examples, the hyperbolic tangent (tanh) and leaky rectified linear unit

(LeakyReLU), which are effectively variations of the sigmoid function and rectified linear unit respectively.

The hyperbolic tangent is a variation of the sigmoid function. That is, much like the standard $\sigma(\cdot)$, tanh depicts the S-shape that is characteristic of sigmoidal functions, as can be seen in Figure 6 (b)–(c). Unlike the sigmoid function however, the hyperbolic tangent maps inputs from the real axis to the interval $[-1, 1]$:

$$\begin{aligned} \tanh &: \mathbb{R} \to [-1, 1] \\ \tanh(z) &= \frac{e^z - e^{-z}}{e^z + e^{-z}}. \end{aligned} \tag{5}$$

In applications where a sigmoidal activation function is required, tanh typically performs better than $\sigma(\cdot)$. This performance advantage results from tanh resembling the identity function $i(z) = z$ near $z = 0$ more closely than the sigmoid function. Specifically, $i(0) = \tanh(0) = 0$, while $\sigma(0) = \frac{1}{2}$. It follows that training a deep neural network with tanh activation functions is somewhat comparable to training a linear model, assuming that the linear activations of the network remain small, i.e. near zero, during training. Consequently, training a neural network with tanh activation functions tends to be easier when compared to sigmoid activation functions [58].

Much like the sigmoid function, tanh suffers from the problem of saturation of units. As such, with the exception of output units, the use of sigmoidal activation functions is discouraged in the context of feedforward networks. In spite of these drawbacks, sigmoidal activation functions remain an appealing choice when it comes to network architectures that present additional requirements that rule out choices like ReLU. Examples of such architectures include recurrent neural networks and some autoencoders [58].

While the rectified linear unit may seem like an ideal activation function, it is not without its flaws. Perhaps the most notable shortcoming of ReLU is the *dying ReLU problem*, which is a type of vanishing gradient problem. The problem occurs when the linear activation $z = \mathbf{w}^\top \mathbf{x} + b$ of some neuron is nonpositive for all training samples, in which case the output of the neuron is zero regardless of input. Consequently, the gradient of the neuron will also be zero for all training samples, that is, the neuron ”dies” as the parameters can no longer be updated due to the zero gradient. In fact, in the worst case scenario where all neurons die, the network output becomes constant [75].

To solve the dying ReLU problem, there exists a variety of generalizations of the rectified linear unit, that all guarantee a nonzero gradient regardless of the input [58]. One such generalization is the leaky rectified linear unit, where the negative inputs are given a fixed slope:

$$\begin{aligned} \mathrm{LeakyReLU} &: \mathbb{R} \to \mathbb{R} \\ \mathrm{LeakyReLU}(x) &= \max(0, x) + \alpha \cdot \min(0, x), \end{aligned} \tag{6}$$

where $\alpha \in \mathbb{R}^+$ is the slope hyperparameter, typically set close to zero [76]. Choosing an optimal value for α can however be difficult and requires careful hyperparameter optimization. To this end, [77] proposed the parametric ReLU (PReLU), a further generalization of LeakyReLU, where the slope parameter α is made trainable instead of constant. PReLU was shown to significantly increase model performance in the task of image classification with negligible increase in computational cost [77].

2.2.3 Fully Connected Layer

Fully connected layers are perhaps the most common architectural elements of neural networks and the building blocks behind the quintessential deep feedforward network multilayer perceptron [58]. As the name suggests, a fully connected layer is a collection of artificial neurons, each of which has a connection to all neurons in the subsequent layer. Formally, a fully connected layer can be described as follows:

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{7}$$

where $\mathbf{h} \in \mathbb{R}^m$ is the output of the layer, $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the layer weight matrix and $\mathbf{x} \in \mathbb{R}^n$ is the layer input, which is generally the output of the previous layer. Furthermore, $\mathbf{b} \in \mathbb{R}^m$ is the learnable bias vector and $g(\cdot)$ is the non-linear activation function which is typically applied element-wise, that is, individually for each artificial neuron. The dimensionality variables n and m denote the number of artificial neurons in the previous and current layers respectively. Finally, we note that the weight matrix $\mathbf{W}$ is simply composed of the weight vectors of the neurons in the layer stacked together, such that the ith row corresponds to the weight vector $\mathbf{w}$ of the ith neuron.

An MLP can be visualized by displaying all of the artificial neurons within each layer and the connections between them, but it is often more convenient to disregard the individual neurons and connections and only visualize the layers. The aforementioned observation also holds true for most other types of neural network layers. An example of both visualization approaches is shown in Figure 7.

A fully connected layer with no non-linear activation functions is known as a *linear layer*. In order to emphasize the importance of non-linear activation functions in deep learning models, we show that a stack of linear layers has the same representational capabilities as a single linear layer.

Theorem 2.1. *The output of any two linear layers can be represented by one linear layer with appropriately chosen weight matrix and bias vector.*

Proof. Let $\mathbf{h}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$ be the output of the first linear layer. If we then provide it as an input to another linear layer, we get:

$$\begin{aligned}\mathbf{h}_2 &= \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 \\ &= \mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \\ &= (\mathbf{W}_2\mathbf{W}_1)\mathbf{x} + (\mathbf{W}_2\mathbf{b_1} + \mathbf{b}_2) \\ &= \mathbf{W}_3\mathbf{x} + \mathbf{b}_3.\end{aligned} \tag{8}$$

That is, the exact same transformation can be represented with a single hidden layer with the weight matrix $\mathbf{W}_3 = \mathbf{W}_2\mathbf{W}_1$ and bias $\mathbf{b}_3 = \mathbf{W}_2\mathbf{b_1} + \mathbf{b}_2$. The proof can be generalized for an arbitrary number of linear layers using proof by induction, which we omit here for the sake of brevity. □

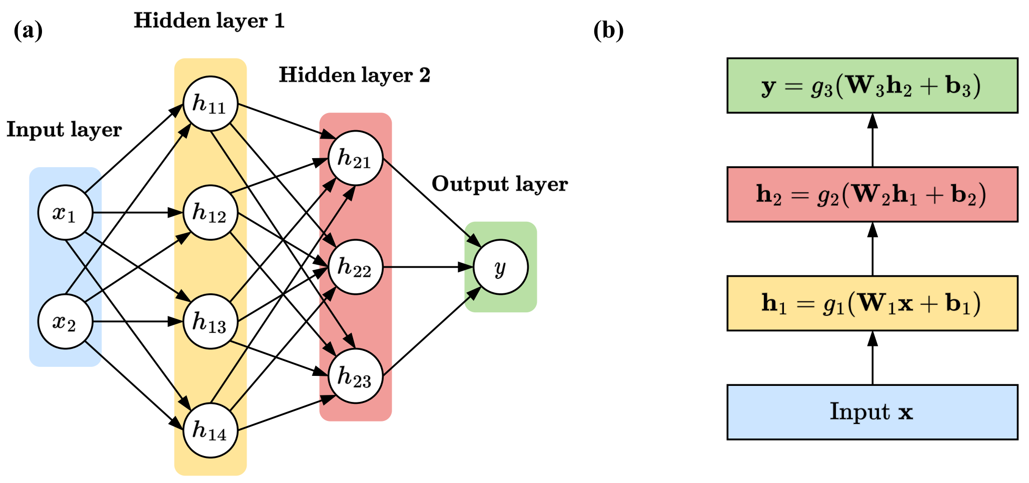

(a)
Hidden layer 1
Hidden layer 2
Input layer
Output layer
x_1
x_2
h_{11}
h_{12}
h_{13}
h_{14}
h_{21}
h_{22}
h_{23}
y
(b)
$\mathbf{y} = g_3(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$
$\mathbf{h}_2 = g_2(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$
$\mathbf{h}_1 = g_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$
Input $\mathbf{x}$

Figure 7: Structure of the same simple feed-forward network with two fully connected hidden layers expressed in two different ways. (a) A schematic showing all neurons within the hidden layers, and the connections between them. (b) A simplified schematic where only each layer and its output is displayed.

An inherent limiting factor of fully connected layers is their scalability. As the input dimensionality or the size and number of the hidden layers increases, so does the number of trainable parameters. In fact, due to each neuron in subsequent layers being connected, the number of parameters can quickly increase to a point where training the network becomes extremely challenging due to the amount of required computational resources, an issue which calls for more intricate layer architectures when it comes to complex input data.

2.2.4 Convolutional Layer

While theoretical results show that a network consisting of fully connected layers is able to approximate any given function, computational resources are often a limiting factor in practical applications. In fact, even processing 2D images quickly becomes computationally infeasible as the number of pixels increases due to the massive number of parameters required for handling each pixel as a separate input. Another shortcoming of fully connected networks is that they disregard the spatial structure of the input.

Convolutional layers, first proposed by [78], are designed to overcome some of the limitations of fully connected networks. Conversely to fully connected layers, convolutional layers utilize shared parameters such that the same set of learnable weights is used for every spatial location. Consequently, the number of trainable parameters and by extension the amount of required computational resources remains manageable. A key assumption made by convolutional layers is that the input features are spatially invariant. That is, similar collections of features within an input should result in similar outputs regardless of their location in the input. Neural networks composed of convolutional layers are referred to as convolutional neural networks.

A noteworthy limitation of convolutional layers is that a clear grid-structured topology is required from the input data [58]. For pixel based 2D images this is already the case, but 3D point clouds pose a problem, as the point coordinates are generally real numbers, especially in the case of real world LiDAR data. As such, point cloud data has to be discretized to enable using conventional convolutional layers. Currently this is often achieved by downsampling the point cloud into sparse voxel representations [79, 80, 81, 82], while earlier implementations have opted for dense voxelizations [83] or 2D projections, such as images [84, 85] or spherical grids [86]. In addition, multiple architectures based on modified convolutions exist, which are discussed in more detail in Section 2.4.

Analogously to fully connected layers, the output tensor $\mathbf{h}$ of a convolutional layer is obtained by applying an appropriate non-linear activation function $g(\cdot)$ on a linear activation $\mathbf{z}$. For convolutional layers, the linear activation is the convolution operation, which comprises a combination of tensor multiplication and summation detailed later in this section. In order to compute the convolution operation, a tensor of trainable weights called *convolution kernel* is slid across the input tensor. The size of the kernel is an user defined hyperparameter.

The number of input channels N_{in} and the number of output channels N_{out} are two other important hyperparameters in the convolutional layer. The former is effectively the dimension of the input vectors, while the latter can be chosen based on model requirements. As an example, each pixel in a 2D image can contain a 3-dimensional color feature vector, where the three values represent the values of the red, green and blue channels. For such input tensors the number of input channels in a convolutional layer would then be $N_{\text{in}} = 3$.

The total number of trainable parameters in the kernel is dependent on both the dimension D of the convolution and the number of input and output channels. Let us consider a 2D convolution with a kernel of height K_h and width K_w. The convolutional layer then contains N_{out} kernels of size $K_h \times K_w \times N_{\text{in}}$, that is, the number of learnable parameters in the layer is

$$K_h \cdot K_w \cdot N_{\text{in}} \cdot N_{\text{out}} + N_{\text{out}}, \tag{9}$$

where the latter term accounts for the bias vector. The size of the kernel is often some small odd integer, such as 3 or 5. Furthermore, while the convolutional kernel can have a different size in every dimension, it is common practice in the field to employ hypercubical kernels and refer to the kernel size with a single parameter K.

Let us now give a formal mathematical definition for the D-dimensional convolution with N_{in} input channels and N_{out} output channels. Let $\mathbf{x_u} \in \mathbb{R}^{N_{\text{in}}}$ be an N_{in}-dimensional input feature vector at index $\mathbf{u} \in \mathbb{Z}^D$ of a tensor in a D-dimensional space. Furthermore, let the trainable weight tensor and bias of the convolutional kernel be $\mathbf{W} \in \mathbb{R}^{K^D \times N_{\text{out}} \times N_{\text{in}}}$ and $\mathbf{b} \in \mathbb{R}^{N_{\text{out}}}$ respectively, where $K^D = \underbrace{K \times K \times \ldots \times K}_{D}$ denotes the dimensions of the kernel. We then denote the $N_{\text{out}} \times N_{\text{in}}$ matrix at index $\mathbf{i} \in \mathbb{Z}^{K^D}$ of the weight tensor by $\mathbf{W_i}$. Using the notation of [80], for an arbitrary output coordinate $\mathbf{u} \in \mathbb{Z}^D$, the

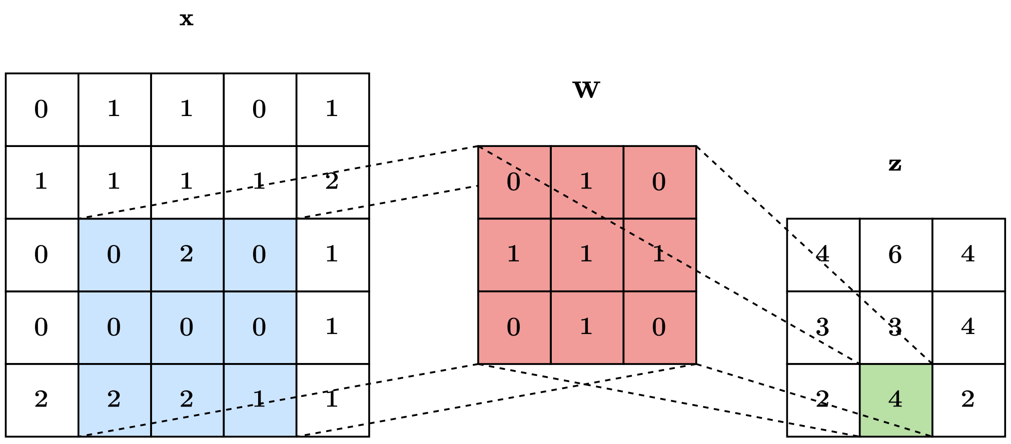


Figure 8: A visualization of a 2-dimensional convolution with a single input and output channel using a 3×3 kernel, stride and dilation one and no padding. The figure highlights how a single output value is formed from the input matrix and learnable kernel weights.

linear activation of the convolutional layer is then given by:

$$\mathbf{z_u} = \sum_{\mathbf{i} \in \mathcal{V}^D(K)} \mathbf{W_i x_{u+i}} + \mathbf{b}, \tag{10}$$

where $\mathcal{V}^D(K)$ is the list of offsets in a D-dimensional hypercube of size K centered at the origin. That is, $\mathcal{V}^D(K)$ denotes the list of offsets that form the kernel. As an example, the offsets corresponding to a 2-dimensional kernel of size $K = 3$ are as follows:

$$\mathcal{V}^2(3) = \{(-1,-1),(0,-1),(1,-1),(-1,0),(0,0),(0,1),(-1,1),(0,1),(1,1)\}. \tag{11}$$

An example of the convolution operation using the same kernel is shown in Figure 8.

In contrast to fully connected layers, the output of a single artificial neuron in a convolutional layer only depends a small area of the input tensor. This area is known as the *receptive field*, and its size is limited by the kernel size K. Since any element outside the receptive field of a neuron has no influence on its output, controlling the size of the receptive field to ensure it covers the entire relevant input region is crucial when it comes to training convolutional neural networks. The size of the theoretical receptive field can be increased by stacking multiple convolutional layers, which many popular deep learning architectures utilize [87].

Besides the kernel size and number of output channels, a convolution is parameterized by its *stride*, *dilation rate* and *padding*. The stride of a convolutional layer determines how many elements are skipped when the kernel is slid across the input tensor. In a standard convolution the default stride is one, which means that the kernel simply moves forward one element at a time. Similarly, for an arbitrary stride s, the kernel skips s elements each time it slides forward. Increasing the stride will decrease the spatial dimensions of the output tensor, which is an effective way of reducing the amount of required computations. In practice adjacent elements of an input tensor often contain similar information and consequently, skipping some elements within the convolutional layer can be a reasonable approach for decreasing the size of the output with no significant loss of information.

Dilation rate is a hyperparameter that controls the distance between the kernel weights, which can be used for increasing the size of the receptive field in a convolutional layer, without increasing the number of trainable parameters. This is an especially useful property when the kernel size K is not sufficiently large, while limited computational resources simultaneously prevent increasing it. With a dilation rate r, the kernel weights are placed in a sparse configuration at every rth element. In practice, this is achieved by adjusting the offsets of the kernel weights $\mathcal{V}_r^D(K)$. For example, in contrast to Equation 11 where $r = 1$, when the dilation rate is increased to $r = 2$, the offsets corresponding to a 2-dimensional kernel of size $K = 3$ are:

$$\mathcal{V}_2^2(3) = \{(-2,-2),(0,-2),(2,-2),(-2,0),(0,0),(0,2),(-2,2),(0,2),(2,2)\}. \tag{12}$$

The resulting kernel now covers a 5×5 area, while crucially the number of learnable parameters remains unchanged. Unlike a 5×5 kernel with $r = 1$, however, the dilated 3×3 kernel has a one element gap between each weight.

For any kernel such that $K > 1$, the convolutional layer will reduce the spatial dimensions of the output tensor in comparison to the input. This is a direct consequence of the fact that the convolution can not be computed for any location where the kernel is not completely within the boundaries of the input tensor. Padding refers to the process of introducing additional elements, most commonly zeros, along the edges of the input tensor, which enables computing the convolution for such locations and by extension decreases the amount of reduction in spatial dimensions. This is especially beneficial in deep networks, where decreasing the output size on every layer would quickly accumulate. As an example, for an arbitrary 2-dimensional input matrix and a square kernel of size K, introducing a padding of $\frac{K-1}{2}$ will result in an output that completely preserves the spatial dimensions of the input tensor, assuming that the stride and dilation rate are both one.

In addition to preserving the spatial dimensions of the input, padding increases the influence elements close to the edges have on the output: assuming $K > 1$, with no padding each element along the edges will have an effect on notably fewer output values in comparison to elements toward the center of the input tensor, which simply fall within the kernel more often.

Sparse Convolutional Layer: Although CNNs are quite efficient for one and two-dimensional (2D) data, increasing the number of dimensions to three will often result in an unreasonably high computational complexity for any slightly larger input [88]. On the other hand, increasing input dimensionality will frequently result in not only a significantly larger number of input elements, but also an increased number of zero-valued, or empty, elements [88]. For example, a majority of the voxels in a voxelized 3D point cloud representing a real world scene will generally contain no points at all. Consequently, increasing the dimensions will increase both the amount of memory used by redundant input elements and the number of redundant computations performed when applying the convolution operation. In order to reduce the memory footprint, such sparsely populated data can be transformed into sparse tensor format, which is a generalization of sparse matrices. A sparse matrix is simply a matrix where most of the elements are zeros [89]. In the context of sparse tensors, conventional tensors are often referred to as *dense*.

Sparse tensors are usually stored in a format that only contains the non-zero elements. This is achieved by representing the tensor as two separate matrices, coordinates $\mathbf{C}$ and features $\mathbf{F}$, which contain the coordinates and values of the non-zero elements respectively:

$$\mathbf{C} = \begin{bmatrix} u_1^1 & u_1^2 & \cdots & u_1^D \\ \vdots & \vdots & \ddots & \vdots \\ u_N^1 & u_N^2 & \cdots & u_N^D \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_N^\top \end{bmatrix}, \tag{13}$$

where $\mathbf{C} \in \mathbb{Z}^{N \times D}$ and $\mathbf{F} \in \mathbb{R}^{N \times N_F}$. The sparse tensor $\mathcal{T}$ is then defined as follows:

$$\mathcal{T}[\mathbf{u}_i] = \begin{cases} \mathbf{F}_i, & \text{if } \mathbf{u}_i \in \mathbf{C} \\ 0, & \text{otherwise,} \end{cases} \tag{14}$$

where $\mathbf{u}_i$ and $\mathbf{F}_i$ denote the ith rows of $\mathbf{C}$ and $\mathbf{F}$ respectively.

While sparse tensors can be significantly more memory efficient in comparison to dense tensors when the data is sparse, they are not functional as input for conventional convolutional layers. To this end [90] proposed sparse convolutional layers for 2D data, which were later generalized to 3D [79, 88, 91] and 4D [80] data. At its core, a sparse convolutional layer performs the exact same convolution operation as a dense one. However, sparse convolutions are able to achieve significant performance gains by exploiting prior knowledge about the structure of the input data, which enables efficiently computing the convolution operation for sparse high dimensional data. Namely, a sparse convolution simply computes the linear activation using only the non-zero elements and disregarding all other input coordinates, which reduces the computational load significantly, as a majority of the coordinates in a sparse input space are empty. Figure 9 shows a comparison between a dense and sparse convolution performed on a sparse matrix.

Employing the same notation as in Equation 10, the linear activation of a sparse convolution for any $\mathbf{u} \in \mathbf{C}_{\text{out}}$ is given by:

$$\mathbf{z}_\mathbf{u} = \sum_{\mathbf{i} \in \mathcal{N}^D(\mathbf{u}, K, \mathbf{C}_{\text{in}})} \mathbf{W}_\mathbf{i} \mathbf{x}_{\mathbf{u}+\mathbf{i}}, \tag{15}$$

where $\mathbf{C}_{\text{in}}$ and $\mathbf{C}_{\text{out}}$ are the coordinate matrices corresponding to the sparse input and output tensors respectively and $\mathcal{N}^D(\mathbf{u}, K, \mathbf{C}_{\text{in}}) = \{\mathbf{i} | \mathbf{u} + \mathbf{i} \in \mathbf{C}_{\text{in}}, \mathbf{i} \in \mathcal{V}^D(K)\}$. That is, $\mathcal{N}^D(\mathbf{u}, K, \mathbf{C}_{\text{in}})$ is a subset of elements in $\mathcal{V}^D(K)$ such that the corresponding coordinates exist in the sparse input tensor.

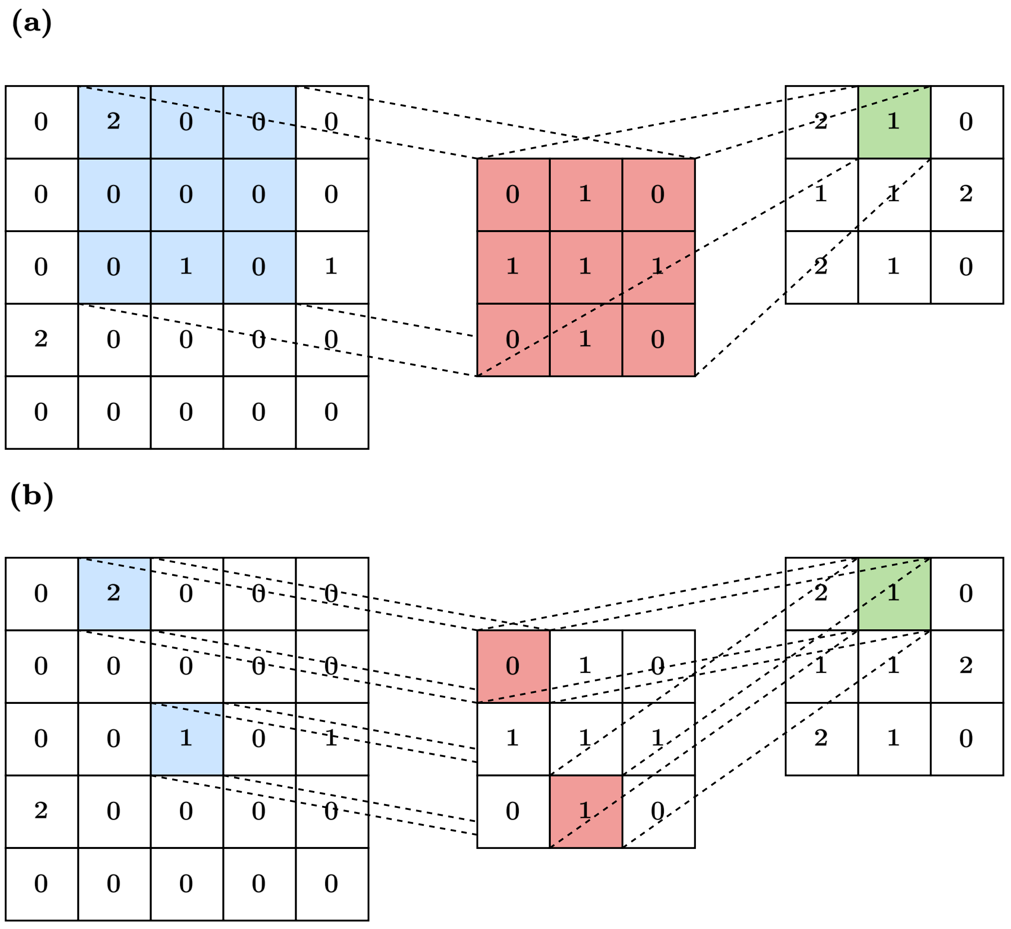


Figure 9: Comparison of a dense convolution and a sparse convolution performed on a sparse matrix using a 3×3 kernel, stride and dilation one and no padding. (a) A dense convolution applied on a sparse matrix. Note how most of the computations are redundant, as a majority of the input features are zerote. (b) A sparse convolution applied on the same sparse matrix. Only non-zero elements in the input matrix are considered, which significantly reduces the number of required computations in comparison to the dense convolution.

In practice, leveraging the computational advantages of sparse tensors and convolutions demands certain additional data structures. Performing operations such as convolution and pooling (see Section 2.2.5) on sparse tensors requires an efficient approach for finding neighbors between the non-zero elements and a method for generating the set of output coordinates. To this end, sparse convolution libraries, such as Minkowski Engine [80], typically implement a *coordinate manager*. Each sparse tensor is then associated with a distinct coordinate manager that handles all coordinate-related operations.

In order to perform a convolution operation on a sparse tensor, the coordinate manager must find which coordinate of the output tensor each element in the input tensor is mapped to. This mapping is known as the *kernel map* and it is comprised of two integer lists: the in map $\mathbf{I} \in \mathbb{Z}$ and the out map $\mathbf{O} \in \mathbb{Z}$. Each element $i \in \mathbf{I}$ indicates a row index of the coordinate matrix $\mathbf{C}$ or the feature matrix $\mathbf{F}$ of a sparse input tensor $\mathcal{T}$, while each $o \in \mathbf{O}$ represents a row index of the coordinate matrix corresponding to the output tensor of the sparse convolution. The in and out maps are ordered such that the kth element of $\mathbf{I}$ and $\mathbf{O}$ correspond to each other. A single kernel map only encompasses the mappings for a single cell in the convolutional kernel, thus the number of kernel maps required for computing the sparse convolution is equal to the number of cells in the convolutional kernel. To further reduce the computational load, both the coordinates and kernel maps are cached by the coordinate manager, as both are reused very frequently.

2.2.5 Pooling Layer

Convolutional neural networks often utilize *pooling layers* between convolutional layers for aggregating output features. In essence, a pooling layer constructs a mapping that downsamples the input in an efficient manner, that is, the dimensionality of the feature map is reduced without introducing additional trainable parameters. In addition, pooling makes the network output approximately invariant to minor spatial transformations of the input [58] and by extension controls overfitting [92]. Invariance to spatial transformations is an especially advantageous property when information about the occurrence of specific features is more important than their exact location [58].

Much like a convolution, a pooling operation is performed by sliding a kernel of size K across the input tensor in increments determined by the stride. A pooling layer with the kernel size set equal to the input dimensions is known as a *global pooling layer* [92]. Global pooling can be useful as an output layer, as it returns just one value for each input channel. In contrast to convolutional layers, pooling layers do not perform a convolution operation on the data and the kernel generally contains no learnable weights. Instead, they utilize some aggregate function to compute a single statistic that represents the input values within the kernel. The motivation behind pooling is similar to that of increasing the stride of a convolutional layer: adjacent input locations often contain similar features, thus aggregating them to reduce input dimensionality is reasonable.

Perhaps the most popular pooling operations are average pooling [93] and max pooling [94]. The former, visualized in Figure 10, outputs the mean of the input elements, while the latter returns the largest value within the kernel. Other popular choices include mixed pooling, which is simply a randomized choice between max and average pooling [95], and L^p pooling, which adopts the form of L^p-norm as the aggregate function [96]. That is, using the previously defined notation, the result of an L^p pooling operation at an arbitrary output coordinate $\mathbf{u}$ is given by:

$$\mathbf{z}_{\mathbf{u}} = \left(\frac{1}{|\mathcal{V}^D(K)|} \sum_{\mathbf{i} \in \mathcal{V}^D(K)} \mathbf{x}_{\mathbf{u}+\mathbf{i}}^{p} \right)^{1/p} . \tag{16}$$

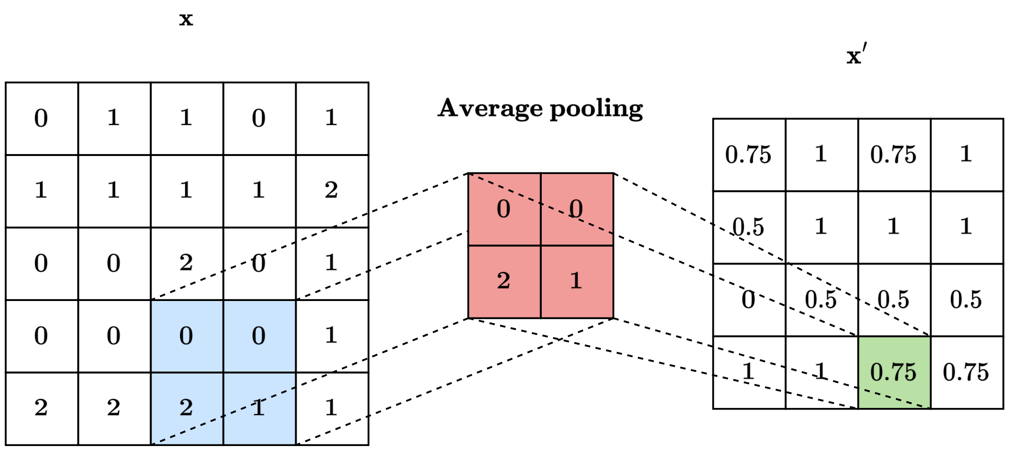


Figure 10: A visualization of average pooling applied on a 2D matrix using a 2×2 kernel, stride one and no padding.

In addition to the methods discussed here, a variety of more novel pooling strategies have been suggested in the literature, some even including learnable parameters [97]. While such methods may offer better performance, the increased complexity will naturally also result in increased computational overhead, which is often not desirable.

2.2.6 Batch Normalization Layer

Modern neural networks are typically trained by dividing the data into small subsets known as *minibatches* (see Section 2.3.2). While normalizing the inputs of a neural network has long been known to be helpful for the training process [98], batch normalization extends this concept by adding intermediate layers that normalize the outputs of layers within deep neural networks [99]. Crucially, batch normalization is applied within minibatches instead of the entire input data set in order to further reduce the computational load. Although a rather recent innovation, batch normalization has been successfully adopted across all fields of deep learning, as it both accelerates the training process and enables using higher learning rates, while also improving the generalization of models [100].

Let $\mathcal{H}$ be the activations of some layer for a minibatch of inputs $\mathcal{B}$ and let m be the number of features in $\mathcal{H}$. A batch normalization layer then normalizes its input $\mathcal{H}$ such that each feature has zero mean and unit variance:

$$\mathcal{H}' = \frac{\mathcal{H} - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}}, \tag{17}$$

where $\mu_{\mathcal{B}}, \sigma_{\mathcal{B}} \in \mathbb{R}^m$ are vectors containing the feature-wise sample mean and sample standard deviation of the current minibatch respectively. Note that the normalization is applied independently for each feature in the input. In order to enable evaluating a trained network for singular inputs instead of entire minibatches, a batch normalization layer will typically cache the running averages of $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ during training [58].

Ioffe and Szegedy [99] noted that simply normalizing the inputs of a layer to have zero mean and unit variance may change the codomain of its outputs and consequently its representative power. To this end, they adopted the additional learnable parameters $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^m$, which scale and shift the normalized value:

$$\widetilde{\mathcal{H}} = \boldsymbol{\gamma}\mathcal{H}' + \boldsymbol{\beta}. \tag{18}$$

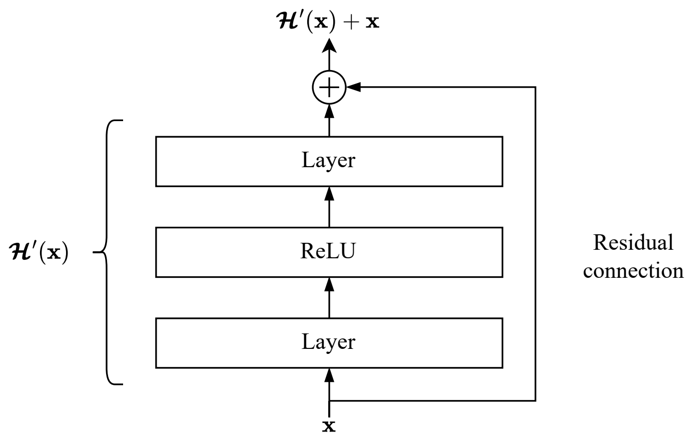
This ensures that the transformation applied by the batch normalization layer can represent the identity transform. Indeed, setting $\boldsymbol{\gamma} = \boldsymbol{\sigma}_{\mathcal{B}}$ and $\boldsymbol{\beta} = \boldsymbol{\mu}_{\mathcal{B}}$ yields the original unnormalized activations $\mathcal{H}$ [99]. While $\mathcal{H}$ and $\widetilde{\mathcal{H}}$ can both represent the same family of functions of the input $\mathcal{B}$, the learning dynamics between the two differ significantly. Namely, the mean of $\mathcal{H}$ is determined by a complex combination of parameters in the previous layers of the network, while the mean of the normalized activations $\widetilde{\mathcal{H}}$ is fully determined by $\boldsymbol{\beta}$. Consequently, the learning process is considerably easier for $\widetilde{\mathcal{H}}$ [58].

A large driving factor in the recent success of deep learning has been the increasing number of layers in the models. Ioffe and Szegedy [99] noted that this also introduces an issue known as *internal covariate shift*, where the distributions of activations in intermediate layers change considerably depending on the input, since small differences in input accumulate into major ones due the large number of layers. This phenomenon then hinders the training process, as the subsequent layers have to learn to accommodate the constants changes in distribution. They further suggested that batch normalization layers effectively eliminate this problem by fixing the input distribution. Later works experimentally showed that the benefits of batch normalization are mainly a result of the larger learning rates it enables, and questioned the role of internal covariate shift. They further argued that these larger learning rates increase the implicit regularization of stochastic gradient descent (see Section 2.3.2) and by extension improve model generalization [100].

2.2.7 Residual Connection

Yet another issue faced by deep learning models is a phenomenon known as *degradation*, where increasing the number of layers causes the model accuracy to first saturate and then start decreasing [101]. Perhaps surprisingly however, degradation is not a result of model overfitting, and the training accuracy of a well performing model may decrease when more layers are added, as experimentally shown by [101, 102, 103]. As an example, consider a well performing deep learning model $\mathcal{M}$. We should then be able to construct a deeper model $\mathcal{M}'$ that has the same training error by simply copying the parameters from $\mathcal{M}$ and having the additional layers perform an identity mapping. It follows that $\mathcal{M}'$ should be able to achieve at least the same training accuracy as $\mathcal{M}$ and the fact that this is not always the case implies that the optimizer simply fails to find an adequate solution [101].

He et al. [101] proposed deep residual networks (ResNets) as a solution to model degradation. ResNets utilize residual connections, which are formally defined as follows: let $\mathcal{H}(\mathbf{x})$ denote the desired non-linear function mapping learned by a stack



$\mathcal{H}'(\mathbf{x}) + \mathbf{x}$
Layer
ReLU
Layer
$\mathcal{H}'(\mathbf{x})$
Residual connection
$\mathbf{x}$

Figure 11: An example of a typical residual connection in a block of two neural network layers. The residual connection performs an identity mapping for the original input $\mathbf{x}$, which is then added to the output of the layers.

of few layers, where $\mathbf{x}$ is the input vector passed to the first layer in the stack. We then let the stack of layers learn another mapping:

$$\mathcal{H}'(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}, \tag{19}$$

from which the desired mapping can be retrieved simply by adding the original input:

$$\mathcal{H}(\mathbf{x}) = \mathcal{H}'(\mathbf{x}) + \mathbf{x}. \tag{20}$$

The residual mapping $\mathcal{H}'(\mathbf{x})$ is a more straightforward problem in terms of optimization when compared to the original mapping with no residuals [101], as evidenced by the undeniable success of deep residual networks in multiple fields, such as object classification, object detection and segmentation [101, 104]. Residual connections are also utilized in transformer-based language models, such as BERT [105] and GPT [106].

In practice, residual connections can be implemented by utilizing *shortcut connections*, which are often also referred to as *skip connections*. A shortcut connection, is a connection that skips one or more layers in a neural network [101]. In a residual connection, the shortcut connection only applies an identity mapping and the input is subsequently added to the output of the skipped stack of layers, as shown in Figure 11. Although using an identity mapping in the residual connection may seem overly simple, especially considering that earlier works utilizing skip connections employed function mappings such as gating [103], this is not the case in practice. In fact, [107] empirically showed that using an identity mapping yields both a more accurate model and a shorter training time in comparison to more complicated mappings, such as scaling or gating.

2.3 Training Neural Networks

Deep neural networks are highly complex and non-convex functions, with the purpose of approximating the true distribution of some underlying data set. In order to achieve the desired function mapping, the network parameters $\mathbf{\Theta}$ must be optimized by fitting the model to a training data set that adequately represents the distribution of the data. This process is often referred to as training or learning.

Training a neural network constitutes optimizing a loss function, which usually measures how well the model outputs match some target, by updating the model parameters according to an optimization scheme. The loss function should be chosen such that it represents the objective adequately while simultaneously being relatively straightforward to optimize. We provide examples of loss functions designed for classification tasks in Section 2.3.1.

Optimization methods used in training neural networks are generally gradient-based and work by iteratively updating the model parameters $\mathbf{\Theta}$ based on the gradient of the loss function $\nabla_{\mathbf{\Theta}}\mathcal{L}(\cdot)$ with respect to the parameters themselves. Gradient-based optimization strategies, which are essentially variants of gradient descent, are discussed further in Section 2.3.2. Computing the gradients is a non-trivial task in itself, which is handled by the back-propagation algorithm in current deep learning applications. The principles behind back-propagation are briefly explained in Section 2.3.3.

There are a few key differences between optimization in the context of training neural networks and traditional optimization. Firstly, the employed loss function is often a proxy for the actual objective function, which may be infeasible to optimize efficiently. As such, minimizing the loss does not necessarily minimize the true objective. Secondly, training algorithms generally do not halt at a local or global minimum, but rather based on some other criterion that is designed to prevent overfitting. Consequently, the derivatives of the loss may still be relatively large when training stops, which is in contrast to traditional optimization where the algorithm is considered to have converged when the gradient becomes very small.

A persisting problem within the field of machine learning is generalizing the performance of a model to new input data outside of the training set. Strategies explicitly designed for preventing models from overfitting and by extension improving generalization, often at the expense of increased training error, are collectively known as regularization [58]. Now, since present day deep neural networks may contain millions or even billions of learnable parameters, they can be quite prone to overfitting. Consequently, development of more effective regularization strategies has been a central research topic in the field [58]. Section 2.3.4 discusses various regularization strategies proposed for preventing the network parameters from overfitting during training.

2.3.1 Loss Function

This discussion focuses specifically on loss functions utilized for classification tasks, as regression is not relevant to semantic segmentation. In this context, the loss function acts a proxy for the objective of maximizing the number of correctly classified data points across the training set [58]. The loss, denoted $\mathcal{L}(\cdot)$, is a measure of how well the neural network has classified the inputs in comparison to some target, which in the supervised learning setting is simply the ground truth class.

One of the most simple loss functions is computing the mean squared error between the ground truth vector $\mathbf{y}$ and the predicted labels $\hat{\mathbf{y}}$:

$$\mathcal{L}_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2. \tag{21}$$

At present, mean squared error has largely been replaced by functions within the cross-entropy family of loss functions [58] and most current DL classification models rely on minimizing the cross-entropy loss [108]. Consequently, cross-entropy loss has been widely utilized by DL methods designed for point cloud classification tasks, including point-wise classification, i.e. semantic segmentation.

In order to define the conventional cross-entropy loss, let $\mathbf{x}_i \in \mathbb{R}^{|C|}$ denote the unnormalized output vector of a model corresponding to the ith input point, often also referred to as logits, and let $\mathbf{x}_i^c$ denote the value of $\mathbf{x}_i$ corresponding to class $c \in C$. Further, let $y_i \in C$ be the corresponding target. The cross-entropy loss can then be computed using the following formula:

$$\mathcal{L}_{\mathrm{CE}}(\mathbf{x}_i, y_i) = -\log \frac{\exp(\mathbf{x}_i^{y_i})}{\sum_{c \in C} \exp(\mathbf{x}_i^c)}, \tag{22}$$

where $\log(\cdot)$ is the natural logarithm. Note that Equation 22 is for computing the loss of a singular data point, while in practice the model input will be a minibatch containing multiple data points. The set of losses computed for the individual data points can be reduced into a single value for example by computing the sum or mean of the losses. Let $\mathbf{X}_j \in \mathbb{R}^{N \times |C|}$ be a matrix of output vectors corresponding to a minibatch $\mathcal{B}_j$ with N data points and let $\mathbf{y}_j \in \mathbb{R}^N$ be the targets of each data point. Using mean as the reduction operation, the cross-entropy loss of the entire minibatch can then be computed as follows:

$$\mathcal{L}_{\mathrm{CE}}(\mathbf{X}_j, \mathbf{y}_j) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_{\mathrm{CE}}(\mathbf{x}_n, y_n), \tag{23}$$

where $\mathbf{x}_n$ and y_n denote the nth element of $\mathbf{X}_j$ and $\mathbf{y}_j$ respectively.

Considerable imbalance between ground truth classes is a common problem in DL data sets including those based on real-world LiDAR data. Focal loss [109] attempts to address this problem by modifying the standard cross-entropy loss such that effect of well-classified samples is weighted down. Here well-classified refers to data points that have a high probability of belonging to a certain class. Consequently, optimizing the focal loss will focus training on the sparse set of hard to classify data points, preventing the more prevalent classes from overwhelming the network parameters during training. Formally, focal loss is defined as follows:

$$\begin{aligned} \mathcal{L}_{\mathrm{F}}(\mathbf{x}_i, y_i) &= (1 - \exp(-\mathcal{L}_{\mathrm{CE}}(\mathbf{x}_i, y_i)))^{\gamma} \cdot \mathcal{L}_{\mathrm{CE}}(\mathbf{x}_i, y_i) && (24) \\ &= -(1 - \mathbf{p}_i^{y_i})^{\gamma} \cdot \log(\mathbf{p}_i^{y_i}), && (25) \end{aligned}$$

where $\gamma \geq 0$ is the user defined focusing parameter. Equation 25 takes advantage of the following equality:

$$\exp(-\mathcal{L}_{\mathrm{CE}}(\mathbf{x}_i, y_i)) = \mathbf{p}_i^{y_i}, \tag{26}$$

where $\mathbf{p}_i^{y_i}$ denotes the predicted probability that the data point $\mathbf{x}_i$ corresponds to is part of class $y_i \in C$. Analogously to Equation 23, the focal loss of minibatch $\mathcal{B}_j$ can then be computed by taking the mean of the individual losses:

$$\mathcal{L}_{\mathrm{F}}(\mathbf{X}_j, \mathbf{y}_j) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_{\mathrm{F}}(\mathbf{x}_n, y_n). \tag{27}$$

Besides cross-entropy loss and focal loss, a wide variety of other loss functions have been proposed in the literature. In fact, essentially any differentiable function can be used as a loss function. Many loss functions are significantly more complex, although simultaneously more descriptive, while some include multiple objectives or are highly specialized for some task. As an example, the PointInfoNCE [82] loss function designed for contrastive pretraining on point cloud data is computed by comparing the values of extracted point-level features across a set of matched points from two views. Increasing the complexity of the loss function will naturally also increase the complexity of computing the gradients and by extension slow down the training. Furthermore, a more complex loss function will not necessarily always improve model performance in any way. As such, choosing the loss function effectively involves choosing the correct balance between descriptive power and computational simplicity.

2.3.2 Optimization

Gradient descent is an algorithm for minimizing an objective function and the basis of most modern optimizers utilized in DL. The algorithm works by iteratively updating the parameters in the opposite direction of the gradient. For an arbitrary neural network, the update step is formally defined as follows:

$$\mathbf{\Theta}_{t+1} = \mathbf{\Theta}_t - \eta \nabla_{\mathbf{\Theta}} \mathcal{L}(\mathbf{\Theta}_t), \tag{28}$$

where $\mathbf{\Theta}_t$ are the model parameters at step t and η is the learning rate, which determines the size of the step taken along the gradient. In conventional gradient descent, the gradient is computed for the entire data set and the algorithm is guaranteed to converge to the global minimum for convex surfaces and to a local minimum for non-convex surfaces [110].

In practice the training data sets used by modern DL models are generally so large that computing the gradient across the entire data set is infeasible. To overcome this limitation, stochastic gradient descent (SGD) only utilizes a minibatch of inputs $\mathcal{B}_j, |\mathcal{B}_j| > 1$ for each update step. Using the notation from the previous sections, the update step of SGD is:

$$\mathbf{\Theta}_{t+1} = \mathbf{\Theta}_t - \eta \nabla_{\mathbf{\Theta}} \mathcal{L}(\mathbf{\Theta}_t; \mathbf{X}_j, \mathbf{y}_j), \tag{29}$$

where the gradient is computed as follows:

$$\nabla_{\boldsymbol{\Theta}}\mathcal{L}(\boldsymbol{\Theta}_t;\mathbf{X}_j,\mathbf{y}_j) = \frac{1}{|\mathcal{B}_j|}\sum_{n\in\mathcal{B}_j}\frac{\partial}{\partial\boldsymbol{\Theta}}\mathcal{L}(\boldsymbol{\Theta}_t;\mathbf{x}_n,y_n). \tag{30}$$

Although convergence is not guaranteed, it has been shown that when the learning rate is slowly decreased, SGD shows the same convergence behaviour as conventional gradient descent. The process of adjusting the learning rate during training is known as *scheduling* [110].

While the standard SGD algorithm is quite effective, it is not without its problems. Challenges to overcome include choosing a proper learning rate and schedule or getting trapped in suboptimal local minima [110]. To this end, a wide variety of more complex optimization schemes have been proposed for training DL models. Below, we present a few examples.

Areas where the surface curves significantly more steeply in one dimension compared to others cause SGD to oscillate and decelerate its convergence. Momentum is a method designed to alleviate this problem. This is achieved by adding a fraction of the previous update step to the current update:

$$\mathbf{v}_{t+1} = \gamma\mathbf{v}_t + \eta\nabla_{\boldsymbol{\Theta}}\mathcal{L}(\boldsymbol{\Theta}_t) \tag{31}$$

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_{\boldsymbol{t}} - \mathbf{v}_{t+1}, \tag{32}$$

where γ is the momentum term, usually set close to 0.9. In essence, the momentum term increases the updates for dimensions whose gradients point in the same direction during consecutive steps, while simultaneously decreasing the updates for dimensions whose gradients change direction [110].

Adaptive moment estimation (Adam) [111] is another extremely popular optimizer, that can be seen as an extension of SGD. Adam computes individual adaptive learning rates for each parameter using estimates of first and second moments of the gradients, denoted $\mathbf{m}_t$ and $\mathbf{v}_t$ respectively:

$$\mathbf{m}_t = \beta_1\mathbf{m}_{t-1} + (1-\beta_1)\nabla_{\boldsymbol{\Theta}}\mathcal{L}(\boldsymbol{\Theta}_t) \tag{33}$$

$$\mathbf{v}_t = \beta_2\mathbf{v}_{t-1} + (1-\beta_2)\nabla_{\boldsymbol{\Theta}}\mathcal{L}(\boldsymbol{\Theta}_t)^2, \tag{34}$$

where β_1 and β_2 are user defined decay rates. Both $\mathbf{m}_t$ and $\mathbf{v}_t$ are initialized with zeros, resulting in them being biased towards zero. As such, Adam applies a bias correction:

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t} \tag{35}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t}. \tag{36}$$

The Adam update rule then utilizes the bias-corrected moment estimates:

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_t}+\epsilon}\hat{\mathbf{m}}_t, \tag{37}$$

where ϵ is a small term commonly set to 10^{-8}.

By combining the advantages of previously proposed optimizers, Adam is able to deal with both sparse gradients and non-stationary objectives, which make it a versatile optimizer that is robust and well-suited to a wide range of non-convex optimization problems [111].

2.3.3 Back-propagation

In order to update the network parameters, the optimization methods presented in Section 2.3.2 require computing the gradient of the loss function with respect to the model parameters $\nabla_{\boldsymbol{\Theta}}\mathcal{L}(\cdot)$. In modern neural networks this is achieved with the back-propagation algorithm, which is effectively an efficient application of the chain rule of derivatives. Starting from the output of the neural network, the gradient is computed iteratively for each layer by propagating a signal backwards through the network. First published in 1970 as a tool for estimating the effects of arithmetic rounding errors on the results of complex expressions [112] and discovered even earlier in the context of certain process models in chemical engineering [113], back-propagation was first applied for training MLPs in the manner that is currently standard by Werbos [114] and later popularized by Rumelhart et al. [115].

In order to illustrate the main principle behind back-propagation, let us consider a neural network with l hidden layers that outputs a N-dimensional vector $\mathbf{y} \in \mathbb{R}^N$. Let $\mathbf{h}^l \in \mathbb{R}^{K_l}$ then denote the input of the lth hidden layer (i.e. the output of the $(l-1)$th hidden layer) and $\boldsymbol{\theta}^l$ the corresponding parameters. Layer l is represented by the function $\mathbf{y} = f_l(\mathbf{h}^l, \boldsymbol{\theta}^l)$, while layer $l-1$ is equivalent to $\mathbf{h}^l = f_{l-1}(\mathbf{h}^{l-1}, \boldsymbol{\theta}^{l-1})$ and so on.

Starting from the network output, we first compute $\frac{\partial \mathcal{L}}{\partial y_n} \forall n \in \{1, \ldots, N\}$. Using these values and by applying the chain rule, the gradients of the loss function w.r.t. to the parameters and inputs of layer l can be computed as follows:

$$\frac{\partial \mathcal{L}}{\partial \theta_j^l} = \sum_{n=1}^{N} \frac{\partial \mathcal{L}}{\partial y_n} \frac{\partial y_n}{\partial \theta_j^l} \tag{38}$$

$$\frac{\partial \mathcal{L}}{\partial h_k^l} = \sum_{n=1}^{N} \frac{\partial \mathcal{L}}{\partial y_n} \frac{\partial y_n}{\partial h_k^l} \tag{39}$$

As the functions representing the layers $f_l(\cdot)$ are all differentiable, the partial derivatives w.r.t. the layer parameters and inputs that appear in Equation 38 and Equation 39 ($\frac{\partial y_n}{\partial \theta_j^l}$ and $\frac{\partial y_n}{\partial h_k^l}$) are quite straightforward to compute. The signals from layer l can then be used to compute the derivatives required for updating the parameters of the previous layer:

$$\frac{\partial \mathcal{L}}{\partial \theta_j^{l-1}} = \sum_{k=1}^{K_l} \frac{\partial \mathcal{L}}{\partial h_k^l} \frac{\partial h_k^l}{\partial \theta_j^{l-1}}. \tag{40}$$

In fact, from Equation 40 it follows that the gradients required for updating the parameters of an arbitrary layer i can be computed by utilizing the signals $\frac{\partial \mathcal{L}}{\partial h_k^{i+1}}$

from the subsequent layer. This is exactly where the efficiency of back-propagation lies: computing the gradients in terms of signal from the subsequent layer avoids a considerable amount of computations in the chain rule that would be required otherwise. These savings in computational overhead are made possible by iterating through the network backward, rather than forward.

Deep learning libraries such as PyTorch [116] and TensorFlow [117] implement back-propagation by constructing a computational graph alongside the neural network. The nodes in the graph contain the necessary computations for both the forward and backward-passes [58].

2.3.4 Regularization

Although deep neural networks can approximate highly complicated and non-linear functions, the chosen family of neural networks does not necessarily include the true distribution of the data, that is, the data generating process. In fact, since the domains DL models are applied to are generally very complex, the true distribution is effectively guaranteed to be outside the model family in most cases. Consequently, controlling model complexity is not simply a matter of finding an appropriately sized model with the correct number of parameters. In practice, the model that minimizes the generalization error is almost always a large model with appropriate regularization [58]. In other words, regularization is a crucial part of training DL models. Below, we provide a few examples of popular regularization strategies utilized in deep learning.

Data augmentation: One of the most effective ways of preventing overfitting and improving the generalization of deep learning models is to increase the amount of training data [58]. However, this is often not possible and the amount of available data always has some limit. One way to alleviate this problem is constructing additional artificial inputs based on the available data, a process also known as *data augmentation* [58]. In practice, creating additional training data is easiest for classification tasks, where a new data point can be created by applying some transformation on any input [58]. In fact, most DL-based point cloud semantic segmentation models utilize data augmentation as well (see e.g. [15, 82]). Essentially the goal of data augmentation is to make the model invariant to a wide variety of transformations, which is an advantageous property when if comes to model generalization [58].

Early stopping: When a neural network begins to overfit, one can generally observe that the training error continues to decrease, while the validation error begins to increase over time. Consequently, in such situations returning the model from the training epoch where the validation error was the lowest will yield a model with a lower validation error than the model parameters at the end of the training. Often the model that minimizes the validation error will also perform better on the test set [58].

This regularization strategy known as *early stopping* is extremely simple, yet effective. During training, the model parameters are stored whenever the validation error improves. Once the validation error has not improved for some user-defined number of iterations, the training terminates. At this stage, rather returning the parameter values at the end of the training, we return model with the best performance on the validation set. In essence, early stopping acts as a regularizer by restricting the optimization procedure to a comparatively small parameter space in the neighborhood of the initial parameter values [58].

In addition to being very simple to implement, early stopping does not incur a significant computational cost, as it only requires storing the parameters of the best model. A more significant drawback is the requirement for an additional validation data set, which can be a limitation if the training set is small to begin with [58].

Dropout: Assuming an unlimited amount of computational resources, the most effective way to regularize a fixed-sized model is computing an average of the predictions across all possible parameter combinations, such that each combination is weighted by its posterior probability given the training data [118]. More generally, the process of training several models and averaging their predictions in an effort to reduce the generalization error is referred to as *bagging*. The reason bagging works is that distinct models generally do not make the exact same mistakes on the test set [58].

In the context of neural networks, training multiple models is often not practical, especially given that combining models is most effective when they have either differing architectures or training data. Optimizing the hyperparameters and training each model would be extremely laborious and require significant computational resources. Moreover, the amount of available training data is often quite limited. Dropout is a regularization strategy designed to address both of the aforementioned issues. In practice, dropout consists of randomly dropping out units and all associated connections by temporarily removing them from the network [118].

Given a neural network with n units, dropout is comparable to sampling thinned network from the 2^n networks that can be formed by dropping one or more units. Here thinned network refers to the network that is formed by the retained units after applying dropout. As all of the thinned networks share weights, the total number of parameters remains the same. During training, a new thinned network is sampled for each input. As such, training a neural network with dropout regularizaton is similar to training a collection of 2^n distinct networks with extensive weight sharing [118].

Since explicitly computing the average of all thinned networks is not feasible at test time, dropout computes an approximation by scaling the trained parameter of each unit by the corresponding dropout probability p. This way, the 2^n networks with shared weights can efficiently be combined into a single model. Dropout has been shown to significantly reduce overfitting and improve generalization of neural networks in tasks such as classification [118].

The most straightforward practical implementation of dropout retains each unit with probability p independently of other units during training. At test time, the effect of averaging the predictions over multiple thinned networks is then approximated by using a single unthinned network with smaller weights [118].

[118].

2.4 Point Cloud Semantic Segmentation

In the following subsections, we provide an overview of deep learning-based approaches for semantic segmentation of point clouds. The methods have been grouped by the level of supervision used in the training into fully-supervised, weakly supervised and unsupervised approaches. In the case of unsupervised methods, traditional algorithms are briefly discussed as well due to the limited amount of research on DL-based models. For a more comprehensive overview of deep learning-based semantic segmentation of point clouds, the interested reader is referred to [27, 28, 119, 120] in the case of supervised methods and [121] for weakly supervised methods.

2.4.1 Supervised Methods

Point cloud semantic segmentation methods can be roughly divided into two categories based on how they handle the input cloud: *rule-based* and *point-based* methods. Methods in the former category first transform the point cloud into some structured data format, such as 2D image or voxel image, while point-based methods process the input clouds directly [119]. Below we provide a brief introduction to some of the most notable neural network architectures in both categories.

Rule-Based Methods: Some of the first rule-based DL methods for semantic segmentation of point clouds were based on generating two-dimensional representations of the 3D points and applying existing neural network architectures designed for image data. Multiple projection strategies have been proposed, all of which have their own drawbacks caused by the information loss inherent to projecting from three to two dimensions [27].

Multi-view projections, which are comprised of multiple images generated by projecting the point cloud onto a plane from several virtual camera perspectives, are one method for generating two-dimensional representations of point clouds. Multi-view projections were first used for semantic segmentation by MVCNN [122], which first extracts features from each projection using a conventional 2D CNN and subsequently aggregates them in a pooling layer. Finally, the obtained 2D labels are projected back to the original point cloud [122]. Similar to MVCNN, SnapNet [123] generates both RGB and depth composite images of the input point cloud from multiple random views. In order to leverage the complementarity of the depth and color information, the images are utilized for training two parallel CNNs after which the predictions are fused and projected back on the point cloud [123]. Later on, SnapNet-R [124] improved upon the performance of SnapNet by introducing 3D-consistent view generation and a CNN capable of handling the generated RGB and depth images simultaneously.

Since any multi-view projection is an approximate abstraction of the 3D scene, semantic segmentation methods utilizing multi-view projections are prone to occlusions and defects in the point cloud [119] and their performance can be quite sensitive to the chosen virtual camera perspectives [120]. Furthermore, the projection itself leads to a significant loss of geometric information, which further affects the segmentation accuracy [120].

Spherical projections, where the points are projected onto the surface of a sphere and then rasterized, present another popular alternative for 3D to 2D transformation. Spherical projections were first utilized for semantic segmentation by SqueezeSeg [86], where a 2D CNN is used for producing a point-wise label map of the transformed input cloud, which is further refined by a conditional random field (CRF). Later, SqueezeSegV2 [125] introduced a novel context aggregation module which improved SqueezeSeg's previous results by reducing its sensitivity to dropout noise, that is, missing points in the input LiDAR point cloud. Furthermore, issues caused by class imbalance were addressed by opting for focal loss instead of cross entropy loss during training [125].

To solve problems such as discretization errors and blurry 2D CNN outputs caused by the use of spherical projections as intermediate representations of the 3D point cloud, RangeNet++ [126] introduced a post-processing algorithm based on k-nearest neighbors (k-NN). Another issue presented by spherical projections of LiDAR data is that in comparison to regular RGB images, the feature distribution of LiDAR images varies drastically based on location, decreasing the segmentation performance of standard convolutional layers. To address this, SqueezeSegV3 [127] proposed using spatially-adaptive convolutions, which alter the filter based on location within the LiDAR image.

While DL methods utilizing spherical projections have proved more effective for semantic segmentation than multi-view projection-based methods, the issue of occlusions caused by the projection remains. Furthermore, the projection step may cause distortions at the edges of the image [27].

Moving away from two-dimensional projections, voxelization is an effective way of transforming point clouds into structured format that does not cause occlusions. In a regular voxelization, the point cloud is first divided into a 3D grid, after which a voxel-level representations are derived from the points within each grid cell [27]. Voxelized point clouds can be processed using conventional convolutional neural networks that have been generalized to three dimensions. An early example of this is VoxNet [83], which used 3D CNNs for object detection from point clouds transformed into occupancy grids. A 3D CNN designed specifically for semantic segmentation was first proposed by Huang and You [128]. However, their implementation does not produce a dense labeling of the point cloud, as the predicted class of each voxel is only assigned to points near its center during label inference [128].

Obtaining dense labelings presents a problem in the context of applying standard 3D CNNs to voxelized point clouds. If the voxels are large, assigning the predicted label to each point within the voxel produces excessively coarse results, while decreasing the size of voxels increases the memory footprint of the convolutions cubically, quickly rendering the use of 3D convolutional layers computationally infeasible. In order to obtain dense labeling while keeping the computational requirements reasonable, SegCloud introduced a pipeline, where coarse voxel predictions from a 3D CNN are transferred back to the point cloud via trilinear interpolation after which a fully-connected CRF is applied to enforce spatial consistency of the labels [129]. Similarly, to better preserve local geometry of the point cloud, PointGrid [130] proposed using 3D CNNs in combination with a novel 3D grid structure, where a constant number of points $K > 1$ is sampled from within each cell.

When a point cloud is voxelized, a significant portion of the voxels contain no points at all. As such, a sparse tensor that only contains information about the occupied grid cells is generally a more efficient data structure for processing voxelized point clouds. To this end, Graham [88] adapted sparse convolutional layers for point cloud data, achieving significant performance gains in 3D classification. Later on, sparse convolutional neural networks (SCNNs) were also utilized for efficient object detection from LiDAR point clouds by Engelcke et al. [91]. A noteworthy drawback of sparse convolutions is that they dilate the sparse inputs by applying conventional full convolutional operations. That is, the number of non-empty voxels increases substantially with each consecutive convolutional layer. Consequently, as the feature sparsity decreases, so do the computational advantages provided by sparse convolutions, which limits the number of layers in a sparse convolutional neural network in practice. To circumvent this, Graham et al. [79] proposed submanifold sparse convolutions (SSCs), which restrict the output of a convolution to the set of active input points. The computational cost of SSCs is around 50% less than that of standard sparse convolutions, which enabled implementing and training deeper convolutional networks such as ResNets [101] and DenseNets [131] for point cloud data [79]. As a result, SSC-based methods achieved at the time state-of-the-art performance on semantic segmentation of point clouds [79]. Choy et al. [80] later generalized submanifold sparse convolutions for generic input and output coordinates as well as arbitrary kernel shapes and succesfully utilized the generalized SSCs for semantic segmentation of spatio-temporal point cloud data.

Hybrid methods which combine rule-based and point-based architectures present another interesting research direction. A recent example of a hybrid approach intended for semantic segmentation is PCSCNet [132], which extracts point-wise features from the raw point cloud using an MLP and then aggregates them into voxel-level features by applying a point convolution within each voxel. The extracted voxel-wise features are subsequently provided as input for a 3D SCNN, which outputs global voxel-level features. Finally, the local point-level features and global voxel-level features are concatenated and passed into an MLP, which outputs a classification for each point [132].

Besides voxelization, a variety of alternative discretization methods that exploit the inherent sparsity of point clouds have been proposed in the context of DL-based semantic segmentation. OctNet [133] hierarchically partitions the voxelized point cloud into a set of unbalanced octrees, each of which divides the 3D space in accordance to the density of the data. The authors then present efficient implementations of standard CNN operations for their octree-based data structure, which dynamically focus computational resources [133]. OctNet achieves a significant decrease in required computational resources, which enables using higher resolutions than conventional voxels [133]. Similarly, O-CNN [134] utilizes a novel octree structure, where the average normal vectors of the point cloud sampled in the finest leaf octants are used as input features for a octree compatible 3D CNN. Unlike OctNet, O-CNN limits the convolutional operations to the octants of the 3D shape boundaries, which further reduces the computational load [134].

SPLATNet [135] proposed discretizing point clouds into high-dimensional lattices. The network utilizes sparse bilateral convolutional layers, which interpolate input features onto a permutohedral lattice, then apply a convolutional operation and finally map the output back to the input points via barycentric interpolation [135]. Later on, Rosu et al. [136] proposed LatticeNet, a hybrid method which combines sparse permutohedral lattices with the point-based network PointNet [15]. LatticeNet distributes the input point features onto a lattice, extracts local features with a PointNet and subsequently uses them as input for a 3D CNN. Finally, the model applies a novel learned data-dependent interpolation, DeformSlice, for projecting lattice features back onto the point cloud. LatticeNet was shown to be especially efficient at semantic segmentation tasks [136].

Point-Based Methods: While rule-based semantic segmentation methods achieve great performance and are often more computationally efficient, the process of transforming the input point cloud into a structured data format inherently incurs a loss of geometric information. To circumvent this, point-based semantic segmentation neural networks process point clouds directly, distinguishing them from rule-based methods that discretize the input [27].

PointNet [15] was the first neural network designed to process raw point clouds directly. A pioneering work in the field of point-based deep learning methods, PointNet first extracts point level features using shared MLPs, where the same MLP is applied to each individual point. The learned features are then aggregated into a global feature vector using a symmetrical pooling layer, specifically global max pooling. Since semantic segmentation cannot be performed based on global features alone, the segmentation network of PointNet combines the global features with local point-wise features [15]. The architecture of PointNet is discussed in more detail in Section 4.2.1.

A notable shortcoming of PointNet is its inability to capture local geometric information. This significantly limits PointNet's ability to learn fine-grained patterns, which is especially a problem in the context of semantic segmentation. To solve this problem, the authors proposed an improved hierarchical architecture called PointNet++, which first partitions the input point cloud into overlapping local regions. In a process reminiscent of CNNs, it then uses PointNets for extracting local features from the small neighborhoods, which are subsequently grouped to larger units and used for extracting higher level features. The process is repeated until the network has obtained features for the entire input cloud. In addition, PointNet++ introduces novel set abstraction layers which adaptively aggregate features according to local point densities [137].

Most early point-based semantic segmentation DL models relied on expensive sampling techniques or computationally heavy pre- and postprocessing steps, which limited their use to small-scale data. In order to achieve efficient semantic segmentation of large-scale point clouds, Hu et al. [138] proposed RandLA-Net, which uses shared MLPs in combination with random downsampling. Since random sampling alone would result in crucial features being dropped, RandLA-Net introduced a local feature aggregation module that progressively increases the receptive field of each point [138]. A more thorough description of the RandLA-Net architecture is provided in Section 4.2.1.

Motivated by the success of convolutional neural networks in rule-based semantic segmentation, various strategies for point-wise convolutions have been proposed. PointCNN [139] generalizes CNNs to leverage spatially-local correlation from unstructured point cloud data. The main innovation of PointCNN is the χ-convolution operation, which first weights and permutes the input points and features using a shared MLP and subsequently applies a conventional convolution on the output. Unlike a regular convolution that uses a $K \times K$ kernel on a grid, a χ-convolution considers the K neighboring points of each point [139].

While the representative power of PointCNN is considerable, using shared MLPs within the χ-convolutions makes the operator quite complex and convergence of the network more difficult. As such, Thomas et al. [140] proposed kernel point convolutions (KPConv), which use a set of kernel points to determine where each kernel weight is applied. That is, both the kernel weights and features are carried by points and their area of influence is defined by a correlation function. Rather then using a fixed number of nearest neighboring points, KPConv utilizes spherical neighborhoods with a user defined radius. The kernel points are chosen by maximizing the distance between them within a sphere and their number can selected based on the desired resolution [140].

PointSIFT [141] augmented the architecture of PointNet++ with novel PointSIFT modules, which significantly improved its performance. A PointSIFT module first passes the input through a stack of orientation-encoding (OE) units, which are point-wise local feature descriptors that encode information of eight orientations. Subsequently, the module concatenates the outputs from each OE unit and applies a point-wise convolution, producing a d-dimensional multi-scale feature for each individual point [141].

Graph convolutional neural networks (GCNN), which operate directly on graph structures consisting of edges and nodes, are frequently employed in the context of computer vision [27] as well as knowledge graphs and recommendation systems [119]. Since an unstructured point cloud can easily be transformed into a graph representation where nodes correspond to individual points and edges capture local geometric information and relationships between adjacent points, several GCNN architectures have been proposed for point-based semantic segmentation of point clouds.

Since GCNNs can be computationally very demanding, Landrieu and Simonovsky [142] proposed superpoint graph (SPG), which first partitions the input point cloud into superpoints using an unsupervised algorithm. The superpoints are then used for constructing a graph representation of the point cloud, which is orders of magnitude smaller than any graph built using the original points. Such a representation enables applying DL models on large-scale point clouds without major sacrifice in fine details. As the superpoints are not necessarily of the same size, SPG first embeds each node in the superpoint graph into a fixed size feature vector with a PointNet. The embeddings are then used as input for training a GCNN to classify the superpoints [142].

Wang et al. [143] introduced DGCNN, which uses novel edge convolutions (EdgeConv) that capture local geometric structures while maintaining permutation invariance. An EdgeConv operation consists of extracting edge features from the

k-nearest neighbor graph of a point cloud with a shared MLP, and subsequently applying a channel-wise symmetric aggregation function, e.g. max. In contrast to conventional GCNNs, DGCNN dynamically updates the graph by recomputing the k-nearest neighbors after each layer in the output feature-space. The authors found this improved performance of the network as with dynamic graphs the receptive field is as large as the diameter of the point cloud, while simultaneously remaining sparse [143].

Lei et al. [144] proposed another novel convolutional operator, spherical convolution (SPH3D), and combined it with an encoder-decoder GCNN similar to U-Net [145]. In a SPH3D, the convolution is applied on all points within a certain radius of the kernel center. The kernel itself is spherical and divided into grid cells that are uniform along the azimuth and elevation dimensions and non-uniform along the radial dimension. As a result, the grid cells located close to the center of the kernel are more fine-grained and can encode detailed local geometric information of the points. By contrast, the grid cells in a conventional convolutional kernel are of uniform size. As such, traditional convolutional layers rely on increased resolution to capture finer details, which limits their scalability in comparison to SPH3Ds [144].

Inspired by the recent success of transformer-based architectures in natural language processing (e.g. BERT [105] and GPT [106]) and computer vision (e.g. ViT [146]), a number of works have proposed transformers adapted for processing point cloud data [119]. A transformer is a decoder–encoder network comprised of three main modules: input embedding, positional encoding, and self-attention, all of which contain only order-independent operations. Consequently, transformers are inherently order invariant, rendering them suitable for processing unstructured point cloud data [147]

Point cloud transformer (PCT) [147] was among the first works to successfully adapt the transformer-architecture for point cloud data. PCT introduced several adjustments to transformers designed specifically for point clouds, including coordinate-based input embedding, optimized offset-attention and neighbor embedding. The offset-attention replaces the attention feature with the offset between the input of self-attention module and attention feature, which they found was more suitable for point cloud learning [147].

While PCT achieved at the time state-of-the-art results across a variety of point cloud understanding tasks, its performance was limited by memory consumption and computational complexity due to its use of global attention applied directly on the point cloud [148]. On the other hand, Point Transformer v1 (PTv1) [149], another pioneering work in transformer-based point cloud understanding networks, is comprised of novel point transformer layers, which are based on vector self-attention. The vector attention only applies self-attention in the local neighborhood of the query point, which alleviates the memory problems with global self-attention [149].

In order to increase the size of the receptive field in comparison to previous transformer-based models, Stratified Transformer [150] proposed using a window-based self-attention in combination with a stratified key-sampling strategy, where distant points are sampled as keys in a sparser way. This strategy enabled capturing long-range contexts effectively, while incurring negligible extra computations.

A notable shortcoming of the vector attention used by PTv1 is that a deeper network increases the number of parameters drastically, which restricts the efficiency and generalization ability of the model. PTv2 [148] addressed these limitations by replacing the vector attention with grouped vector attention, where the input channels are grouped evenly into g groups with shared vector attention weights. In addition, PTv2 introduced improved positional encoding and partition-based pooling, achieving state-of-the-art performance on various semantic segmentation benchmarks [148]. Later, focusing on simplicity and performance, PTv3 [67] shifted to using serialized neighborhoods and a simpler attention mechanism tailored for serialized point clouds. Furthermore, PTv3 eliminated the reliance on relative positional encoding by introducing a novel conditional positional encoding based on sparse convolutions. These improvements enabled significant scaling, expanding the receptive field from 16 to 1024 points, while simultaneously providing a $3\times$ increase in processing speed and a $10\times$ improvement in memory efficiency in comparison to PTv2 [67].

Similarly to PTv3, Superpoint Transformer (SPT) [151] was designed specifically for efficient semantic segmentation of large point clouds. Using a novel algorithm, the input clouds are first transformed into hierarchical superpoint partitions comprised of superpoint-graphs of increasing coarseness. The hierarchical partitions are then used as input for the SPT network, which utilizes a self-attention scheme based on graph-attention networks to learn relationships between superpoints at multiple scales. SPT leverages the hierarchical graph structure effectively, achieving close to state-of-the-art performance while remaining very compact and resource efficient [151].

2.4.2 Weakly Supervised Methods

While fully-supervised deep learning can provide excellent performance, the large amounts of manually annotated data required is extremely time consuming and often also expensive to produce. On the other hand, although unsupervised DL requires no ground truth data at all, the performance of unsupervised models is often far below that of supervised methods. Weakly supervised deep learning attempts to bridge the gap between the two ends of the supervision spectrum by training models with a very limited amount of ground truth data. The main principle is simple: a small amount of data can often be labeled in a reasonable amount of time, yet it provides helpful supervision that improves model performance and makes the training process more straightforward than in the unsupervised case.

Inspired by the success of weakly supervised methods in the field of semantic segmentation of 2D images, a growing number of works have applied weak supervision to point cloud data. Weakly supervised point cloud semantic segmentation methods can be further divided into three categories based the type of ground truth labels they utilize: *2D labels*, *limited labels* and *pseudo labels* [121].

2D Labels: Manually annotating 3D point clouds is significantly more laborious than 2D images, partly because the amount of data in a 3D representation of a scene or an object is substantially higher than any corresponding 2D representation. As such, utilizing only 2D ground truth labels for 3D deep learning applications is a form of weak supervision. Wang et al. [152] proposed an architecture where a graph-based pyramid feature network is trained for 3D semantic segmentation with only 2D labels utilizing a combination of a novel 2D reprojection method and 2D-3D joint optimizer. Their method achieved performance close to that of PointNet++ on the S3DIS benchmark data set with just 16.7% of the labels [152]. Similarly, Kweon and Yoon [153] introduced a joint 2D-3D framework, which simultaneously extracts 3D features from a point cloud and 2D features from a corresponding image. The extracted features are then used as input for a multi-objective loss, comprised of a 3D-to-2D loss and 2D-to-3D loss, computed using a correlation matrix and projected class activation map respectively. Using only 1% of all available frames, their approach achieved an mIoU of 47.4 on the test set of ScanNetV2 [153].

Limited Labels: Moving on from 2D label-based approaches, a more commonly utilized method of weak supervision is using a limited amount of ground truth labels, such that only a small minority of all available data is labeled. The principle behind this strategy is that training such a model would only require manually annotating a few data points. For benchmark data sets, weak supervision with limited labels is usually achieved by randomly sampling a certain percentage of all ground truth labels for training.

Xu and Lee [154] introduced a framework that combines a DGCNN [143] backbone with a 4-part loss function and label propagation at inference stage. In addition to standard segmentation loss, their loss incorporates inexact supervision, self-supervision and spatial and color smoothness constraints. A pioneering work in the field of weakly supervised semantic segmentation of point clouds, their method achieved a performance comparable to that of supervised baselines with only 10% of the labels.

Zhang et al. [155] introduced perturbed self-distillation (PSD) for weakly supervised semantic segmentation of large-scale point clouds. PSD introduces a novel perturbed self-distillation framework, which passes the original point cloud and a randomly transformed point cloud through the same backbone network and a GCNN and subsequently utilizes a self-distillation loss to ensure consistency among the extracted features. To further refine the results, an optional context-aware module can be added to introduce a constraint for point-wise feature affinity. The performance of PSD is comparable to the fully-supervised RandLA-Net with 1% of the point labels [155].

Focusing similarly on large-scale point clouds, semantic query network (SQN) [156] managed to further reduce the required level of supervision. SQN uses the encoder network of RandLA-Net [138] to extract hierarchical point-wise features from the input cloud and subsequently uses them as input for a point feature query network. Given a single query point, the point feature query network aggregates features from its neighboring points into a single vector, which is used for infering the points semantic class. This effectively enables back-propagating training signals from a few sparsely annotated points to a much wider spatial context. Using just 0.1% of ground truth labels, the performance of SQN is close to supervised baselines on multiple benchmark data sets [156].

Owing to their recent popularity, transformer architectures have been proposed for weakly supervised semantic segmentation as well. The MIL-Transformer of Yang et al. [157] adopts the multiple instance learning paradigm and combines it with a simple transformer network. In practice, given an input cloud $\mathcal{P}$ and an additional anchor cloud $\mathcal{P}'$, for each semantic class c in $\mathcal{P}'$ the transformer outputs a positive bag if the class if also present in $\mathcal{P}$ and a negative bag otherwise. The bag values are then used as input for a multiple instance learning loss function [157].

Finally, contrastive scene contexts [158] introduces a slightly different approach: the model is first pretrained using an unsupervised contrastive learning framework that accounts for both point-level correspondences and spatial contexts in a scene. The learned weights are then transferred to a semantic segmentation network that can be trained in a weakly supervised manner. Additionally, the authors propose a novel active labeling strategy to further improve model performance: for each scene, a human annotator is given k points to label based on the features extracted by the pretrained network. The points are chosen as the centroids of clusters created by applying k-means on the features [158].

Pseudo Labels: Pseudo label-based weakly supervised methods and approaches based on limited labels generally utilize the exact same kind of ground truth labels. The key difference between the two is that methods in the former class also assign an artificial label to some or all unlabeled data points during training, while the latter do not.

Following SPG [142], SSPC-Net [8] divides the input cloud into superpoints using an unsupervised algorithm and subsequently constructs a superpoint graph to be used as input for a GCNN. To facilitate weak supervision, SSPC-net introduces a dynamic superpoint label propagation method which accurately infers pseudo labels of superpoints and a superpoint dropout strategy to filter out pseudo labels of low quality [8]. By combining this pseudo labeling strategy with a novel coupled attention module, SSPC-Net delivers competitive results on multiple benchmark data sets using only 0.01% of all labels.

Li et al. [159] proposed HybridCR, a framework that leverages both point consistency and contrastive regularization with pseudo labeling based on the backbone network predictions. The network implements a multi-objective loss, comprised of segmentation loss for the labeled points, consistency loss for the extracted features from the original and augmented point cloud as well as local and global contrastive losses. The contrastive losses regularize the extracted features to be more consistent within point neighborhoods and pseudo label groups [159].

Label-efficient semantic segmentation (LESS) [9] presented a straightforward preprocessing pipeline to create sparsely annotated data for weakly supervised semantic segmentation of point clouds:

1. Detect ground points with a RANSAC-based [160] algorithm.

2. Construct a graph from the remaining points and divide it into connected components.

3. For each component, a human annotator manually labels one point from every class present in the component. The labels are then propagated to all unlabeled points in the component, creating pseudo labels.

LESS divides the labels created during the preprocessing into three distinct classes, sparse, weak and propagated, corresponding to human labeled single points, pseudo labels of points from components that contain multiple classes and pseudo labels of points from pure components that only contain one class respectively. The labels are then used for training a Cylinder3D [161] backbone network with a novel loss function comprised of three separate contrastive losses, one for each of the label types [9].

2.4.3 Unsupervised Methods

As the name implies, unsupervised learning is the exact opposite of fully-supervised learning. That is, the main principle of unsupervised methods is to learn using raw data that contains no manual annotations. Eliminating the need for ground truth labels is undoubtedly extremely advantageous, especially for data that is highly laborious to annotate manually such as point clouds. This is of course provided that the unsupervised approach can produce results that are at least comparable to the supervised methods, which is often not the case.

The application of unsupervised deep learning for semantic segmentation of point clouds remains relatively unexplored to this date. In fact, to the best of our knowledge, the first deep learning-based models for unsupervised semantic segmentation of point clouds were proposed just last year. Conversely, traditional algorithms for unsupervised semantic segmentation of point clouds are an extensively researched topic and high-performance algorithms have been proposed for a wide variety of applications, including power lines [162] as well as railway [163, 164], forest [165, 166] and indoor environments [167]. Moreover, such algorithms have frequently achieved accuracies that are comparable to some fully-supervised approaches. As an example, the voxel and geometry-based algorithm of Poux and Billen [167] achieved performance approaching that of PointNet on the S3DIS benchmark data set, while Tian and Li [166] reported a higher accuracy than a fully-supervised random-forest classifier with their graph-based algorithm in the task of leaf–wood separation.

However, while traditional unsupervised algorithms can achieve very impressive results in the task they are designed for, they generally suffer from two major drawbacks:

1. The unsupervised semantic segmentation algorithms are extremely task specific. That is, an algorithm designed for certain type of point cloud data does not work for data from any other environment, since the number of classes and their attributes are usually hardcoded into the method. For example, an algorithm designed for semantic segmentation of power lines cannot be utilized for semantic segmentation of indoor environments.

2. In order to achieve adequate performance, the hyperparameters have to be optimized separately for each point cloud data set. Traditional unsupervised semantic segmentation algorithms often rely on a set of fixed thresholds, which can vary significantly depending on e.g. point density. Furthermore, the number of hyperparameters generally increases as a function of the number of ground truth classes. For example, the algorithm of Lamas et al. [164] for unsupervised

semantic segmentation of railway environments into 10 classes has more than 30 user defined parameters.

By contrast, an unsupervised deep learning model designed for semantic segmentation of point clouds can theoretically be applied to any type of point cloud data and the number of hyperparameters to optimize generally remains reasonable regardless of the number of classes in the ground truth.

Recent developments in fully-supervised and semi-supervised deep learning for point clouds, including the introduction of powerful yet lightweight DL architectures suitable for point cloud feature extraction, and the introduction of large-scale benchmark data sets, have paved the way for unsupervised DL approaches for point cloud data. Consequently, first steps toward DL-based fully-unsupervised semantic segmentation of point clouds were made very recently, with the introduction of GrowSP [10], U3DS^3 [168] and PointDC [169] architectures.

To the best of our knowledge, GrowSP [10] was the first unsupervised deep learning-based model for semantic segmentation of point clouds. Comprised of three main modules, a feature extractor, a superpoint constructor and a semantic primitive clustering module, GrowSP models the task of unsupervised point cloud semantic segmentation as a joint feature learning and clustering task. As a preprocessing step, GrowSP utilizes the superpoint constructor to divide each input cloud into superpoints using a combination of voxel cloud connectivity segmentation (VCCS) [170] and region growing [171]. The model then passes each input point cloud through a SCNN feature extractor and subsequently constructs superpoint level features by computing the mean of point-wise features within each superpoint. The superpoint representations are then clustered into $S > |C|$ categories with k-means by the semantic primitive clustering module, where C denotes the set of ground truth classes and $| \cdot |$ its cardinality. The cluster assignments are used as pseudo labels for optimizing the feature extractor, while cluster centroids are utilized as a classifier for obtaining semantic primitive labels corresponding to extracted point-level features during training. Similar superpoints are iteratively merged with k-means as the training progresses, while simultaneously updating the semantic primitives [10]. A more in depth description of the operating principles and architecture of GrowSP is given in Section 4.1.1.

U3DS^3 [168] is another recently proposed architecture for unsupervised semantic segmentation. Proposed shortly after GrowSP, the training process, especially the formation of pseudo labels, relies on many of the same principles. U3DS^3 processes point clouds in blocks of constant size, which are first voxelized and subsequently passed as input to a neural network feature extractor, which is a simple yet effective 8-layer 3D CNN. Similar to GrowSP, U3DS^3 then utilizes k-means for clustering the extracted features with VCCS-based superpoints of the input point cloud as guidance. The cluster assignments are used as pseudo labels to train the neural network backbone, while the cluster centroids form a classifier for obtaining label predictions for any extracted feature [168].

In order to improve the robustness of the extracted features, U3DS^3 features two separate pathways which produce two distinct feature representations for the same input block. The first pathway applies both a color transformation and a geometric transformation on the input prior to extracting features, while the second pathway only performs a color transformation. The two different feature representations are then clustered separately, producing two groups of cluster centroids and pseudo labels. During training, U3DS^3 optimizes a two-part loss function which encourages the extracted features to match both the pseudo labels from the same pathway and the labels produced by the different pathway [168].

The third and final unsupervised DL model for semantic segmentation of point clouds, PointDC [169], utilizes cross-modal distillation (CMD) as an initialization step prior to clustering extracted point-level features. In CMD, an input point cloud is first converted into images from multiple viewpoints. PointDC then utilizes a pretrained 2D feature extractor, specifically STEGO [172], for obtaining initial feature representations for the images. The 2D-features are projected back into the input cloud and used as supervision for a sparse 3D U-Net [79] feature extractor.

Outside of CMD, the training process of PointDC is based on the same principles as GrowSP and U3DS^3. Extracted features are clustered with k-means and cluster centroids are used as a classifier for obtaining pseudo labels, which are then iteratively updated as the training progresses. During the clustering, PointDC handles the input clouds as supervoxels, rather than superpoints like GrowSP and U3DS^3. That is, the input point clouds are first voxelized and then segmented into semantically consistent groups of voxels. PointDC simply utilizes the mean of the voxel-wise features to construct supervoxel-level feature representations [169].

GrowSP and U3DS^3 perform similarly on the popular benchmark data sets S3DIS, ScanNet and SemanticKITTI, with segmentation accuracies on par with the fully-supervised PointNet, while PointDC yields slightly worse, albeit competitive, results. However, all existing unsupervised DL models are naturally far away from achieving performance comparable to state-of-the-art fully-supervised and semi-supervised approaches, which is to be expected considering the limited amount of research on the field [10, 168, 169].

2.5 Leaf–wood Separation

The task of leaf–wood separation involves separating the hard wooden components and foliage from each other in point clouds depicting both individual trees and broader forest environments. The problem has been widely studied in the context of remote sensing of forests, since accurate leaf–wood separation has a wide variety of applications in both forest inventory and plant ecology [1]. In forest inventory, the separated wooden components can be used for estimating attributes of tree stems, such as position, DBH and stem curve [1], which yield information about the maturity of wood and the share of structural timber [173]. Conversely, the applications of leaf–wood information in plant ecology are concerned with estimating physiological properties of trees, including leaf angle distribution, gap fraction, canopy radiation and above-ground biomass [1].

This section provides a brief overview of the wide variety of methods proposed for leaf–wood separation. Section 2.5.1 discusses heuristic unsupervised algorithms that require no labeled training data, while approaches based on supervised machine learning and deep learning are covered in Section 2.5.2 and Section 2.5.3 respectively.

2.5.1 Unsupervised Algorithms

Traditional leaf–wood separation algorithms can broadly be divided into two classes, radiometric and geometric, based on how the classification is performed [165]. The former class of algorithms bases the classification on radiometric properties of the points, such as reflectance, while the latter focuses on local and global geometry of the point cloud. Unlike supervised ML or DL methods, traditional leaf–wood separation approaches do not require any prior knowledge about the classes of individual points. Rather, the algorithms detect wood points based on heuristic assumptions about their geometric or radiometric properties as well as the overall structure of trees.

Essentially all existing geometry and/or radiometry based algorithms are specifically designed for TLS point clouds. While the performance for TLS data can be extremely good, the algorithm may fail almost catastrophically when applied to ALS data (as later observed in Section 5.2.1). This is likely due to the fact that the TLS aimed methods rely heavily on either the density of the point cloud or accuracy of the intensity characteristics. On significantly sparser data, e.g. ALS point clouds, the geometric descriptors become less pronounced and the reliability of radiometric properties is simultaneously reduced.

Another shortcoming that many traditional leaf–wood separation algorithms suffer from is that they rely on the input clouds being individual trees (see e.g. [166, 174, 175]). As such, if the point cloud contains an entire forest scene, heavy preprocessing may be required prior to performing leaf–wood separation. Furthermore, producing sufficiently accurate individual tree segments may even require time-consuming manual segmentation, especially in the case of more complex forests.

Radiometry-based Algorithms: Most leaf–wood separation methods that are based on radiometric characteristics have been proposed in the context of 3D reconstruction of trees and forest from point clouds [174, 176, 177] or in research where the main concern is removing the wooden components from the point cloud [178]. In such works the focus is often on some application for which the leaf–wood separation is simply a preprocessing step. Consequently, the point-level the accuracy of the leaf–wood separation is rarely addressed and classifications can be quite coarse.

Côté et al. [174] extracted wood points from TLS scans based on two manually chosen threshold values $t_w > t_f$ such that points with an intensity greater than t_w were assigned to the wood class and points with an intensity below t_f were considered foliage. Points with an intensity between the two threshold values were simply removed. Similarly, Béland et al. [178] performed the leaf–wood classification using two fixed thresholds, which were determined through a careful analysis of different reflectance distributions, including scans from both leaf-on and leaf-off conditions as well as scans containing multiple returns per pulse.

Radiometry-based leaf–wood separation methods have also been proposed for more novel LiDAR data. Wu et al. [176] extracted wood points from simulated full waveform LiDAR using a single intensity threshold. They determined the optimal threshold by applying the density-based spatial clustering of applications with noise (DBSCAN) [179] algorithm to leaf-off point clouds for a range of different thresholds and chose the value that maximized the number of clusters [176]. On the other hand, Yang et al. [177] detected wood points from full waveform TLS scans with a single intensity threshold, which was determined based on the distribution of the ratios between reflectance values and pulse widths.

Li et al. [14] merged data from near-infrared shortwave-infrared scanners and demonstrated that multispectral LiDAR may improve radiometry-based classification of wood and foliage points. In their work they show that there is a distinctive difference between leaf and wood reflectance in the two wavelengths, which enables effective discrimination between two classes [14].

Although most early leaf–wood separation methods were radiometry-based, the approach has fallen out of favor in more recent research due to a multitude of reasons. Firstly, the methods operate on the assumption that the optical properties of leaf and wood points differ significantly at the operating wavelength of the laser scanner. However, factors such as distance, a partial laser hit and its incidence angle can heavily influence such properties [165]. Secondly, the wavelength and power vary between different LiDAR systems, which necessitates calibrating the intensity threshold separately for each distinct instrument. Furthermore, the reflectance characteristics also vary between different tree species even to the degree that the intensity threshold approach is not applicable at all for some species [175].

Geometry-based Algorithms: Much like most radiometry-based algorithms, predecessors to modern geometry-based leaf–wood separation algorithms were mostly designed for 3D reconstruction of trees. Xu et al. [180] introduced an algorithm which first forms a graph by connecting neighboring points and subsequently extracts the wood points starting from a pre-defined root points using Djikstra's shortest path algorithm [181]. Notably, since the algorithm was designed for generating meshes from tree point clouds, its accuracy was not assessed and it featured a phase where additional branches are synthesized. Livny et al. [182] proposed a similar approach, where initial branch structure graphs are extracted using multi-root Dijkstra's algorithm.

On the other hand, Raumonen et al. [183] introduced a local approach for constructing precision tree models from TLS scans, in which the point cloud is first covered with small connected surface patches referred to as cover sets. Utilizing the geometrical properties and neighborhood relations of the cover sets, their method iteratively reconstructs the wooden components of the entire tree. The point cloud and the cover sets are segmented into the trunk and branches, which are subsequently modeled as collections of cylinders [183].

As a part of their framework for volumetric analysis, Belton et al. [184] performed leaf–wood separation on TLS scans of individual trees by applying Gaussian mixture model (GMM) [185] clustering on geometric point cloud descriptors. The geometric features were computed for multiple neighborhood radii sizes after which the size of the feature space was reduced with principal component analysis (PCA) [186]. The points were initially clustered into six distinct classes, which was followed by a manual examination to separate the clusters representing trunk and branches from the foliage

[184].

Tao et al. [175] presented one of the first geometry-based algorithms designed exclusively for leaf–wood separation from TLS point clouds depicting individual trees. Their approach first slices the point cloud horizontally into multiple bins and subsequently detects circles and lines in each bin. Wood components are then detected by thresholding and a skeleton of wood points is formed with Djikstra's shortest path algorithm. Having formed the wood point skeleton, the algorithm classifies the remaining points with a range search [175].

Wang et al. [187] developed an algorithm suitable for entire forest scenes, where the point cloud is first divided into superpoints based on similarity of the z-components of normal vectors within spherical neighborhoods. They then apply dynamic segment merging similar to region growing, where the geometric superpoint descriptors are updated after each merge. Finally, segments comprised of wood points are identified based on a linearity threshold.

A significant number of modern leaf–wood separation algorithms are based on graph-representations of the point cloud. LeWoS [165] first oversegments the point cloud graph into connected components based on density and verticality and subsequently estimates the wood probability of each component based on linearity and size thresholds. Class regularization is then applied to extract the final leaf–wood split. Later, Wang [1] extended the method with additional geometric features for the connected component-based superpoints and superpoint-graph optimization after the initial wood probability estimation.

Somewhat similarly, given a point cloud depicting a single tree, graph-based leaf–wood separation (GBS) [166] first constructs a graph and subsequently finds the shortest paths from all points to the root point. The point cloud is then segmented into homogeneous clusters at multiple scales and initial wood points are identified based on cylindrical and linear characteristics of said clusters. The final leaf–wood classification is formed by applying region growing with the initial wood clusters as seed points [166].

Vicari et al. [188] presented an approach that combines clustering and graph-based leaf–wood separation methods. Their algorithm utilizes four separate classification branches: two based on GMM clustering of geometric features and two based on shortest path analysis applied on a neighborhood graph representation of the tree. Each branch is followed by an optional filtering step after which the predictions are combined into a single leaf–wood classification [188].

Outside of graph-based approaches, a wide variety of other strategies have been proposed for geometric leaf–wood separation. Wan et al. [189] introduced a method where a point cloud is first decomposed into three parts based on local curvature. The two parts with smaller curvature are then voxelized and segmented into connected components, each of which is classified as either wood or foliage based on estimated significance of difference between geometric features. Finally, detected wood segments are merged and all remaining points are classified as foliage [189]. On the other hand, Shcherbcheva et al. [190] proposed a statistical approach, which first fits a GMM with two components to each of six geometric point cloud descriptors and subsequently detects infliction points between the centroids of the resulting clusters. The infliction points are then used as flexible thresholds for discerning between leaf and wood points followed by a postprocessing and denoising step [190].

2.5.2 Machine Learning-based Methods

Various machine learning-based approaches have been proposed for leaf–wood separation, utilizing both geometric and radiometric point cloud information. The main advantage of ML-based methods is that they usually require considerably less hyperparameter optimization in comparison to unsupervised algorithms, where the number of parameters that require careful tuning can be quite high.

However, the drawback of utilizing supervised machine learning is that the training process requires manually generated point-wise ground truth labels, which are generally very time-consuming to produce. On the other hand, in contrast to many traditional algorithms, ML-based leaf–wood separation methods do not necessarily require the input clouds to be represent individual trees, provided that the separation is based on performing a point-wise classification. Considering that manually producing individual tree segments can be quite laborious in itself, this can be a notable advantage.

Given manually labeled training data of ground, foliage and wood points, geometric-based automatic forest point classification (GAFPC) [191] trains a separate GMM for each of the three classes using the point-wise geometric features. The GMMs are then utilized for predicting the probability of a point belonging in each class and classification is realized by assigning each point the class with the highest probability. The initial classification is followed by a postprocessing steps comprised of six consecutive filters designed to correct misclassified points and remove noise [191].

Later research on ML for leaf–wood separation generally opts for using existing ML classifiers and focuses on feature engineering and post-processing. Furthermore, in an effort to develop approaches that generalize better to data from multiple sensors, such approaches often disregard radiometric features choosing instead to only utilize geometric features. Much like in the case of unsupervised algorithms, research on ML-based leaf–wood separation appears to mainly focus on TLS data, most likely due to the heavy reliance on point cloud geometric features.

Yun et al. [192] used geometric features and estimated surface normals to train a support vector machine (SVM) [193] to separate wood and foliage in TLS scans of individual deciduous trees, achieving an overall accuracy over 90% for most species. On the other hand, to asses the feasibility of ML-based leaf–wood separation, Wang et al. [194] compared the performance of four machine learning methods, SVM, naive Bayes classifier [195], GMM [185] and random forest [196] on TLS scans of individual trees from two different species. Utilizing a total of 26 geometric features the best performance was achieved with random forest, although most approaches reached an overall accuracy of at least 95%.

Zhou et al. [197] proposed a multi-scale strategy, where geometric features are computed for several neighborhood sizes. They found that the multi-scale features improved the performance of random forest classifier on TLS scans of individual trees in comparison to utilizing only one optimal neighborhood size. Using similar multi-scale features, Krishna Moorthy et al. [198] compared the performance of random forest, XGBoost [199] and LightGBM [200] on leaf–wood separation from full forest point clouds. Similar to earlier work, they found that random forest yielded the best performance, although the other tested models performed comparably [198].

In contrast to the ML methods based solely on point cloud geometry, Zhu et al. [201] used adaptive radius near-neighbor search for obtaining optimal geometric and radiometric features and subsequently trained a random forest model for classifying points into foliage, wood and ground from RGB colorized TLS forest point clouds. They found that the addition of radiometric features significantly improved the models performance in comparison to only using geometric features [201].

2.5.3 Deep Learning-based Methods

Following the explosive growth of deep learning, recent research in leaf–wood separation has mainly centered on DL-based methods, often focusing on the more general task of semantic segmentation in forest environments, which usually includes additional classes such as ground or understory vegetation. While the difficulty of acquiring ground truth labels remains a problem, DL-based approaches appear to achieve a better performance in comparison to traditional machine learning and unsupervised algorithms (see e.g. [48, 202]). Consequently, the expensive manual labeling step may be considered justified, provided that the performance improvements are significant enough.

Deep learning approaches for leaf–wood separation are generally based on some generic architecture for semantic segmentation of point clouds with additional modifications designed to improve performance on forest data. To the best of our knowledge all published methods utilize fully-supervised learning. In other words, neither weakly supervised learning nor unsupervised learning has been applied to the task of leaf–wood separation.

In one of the first works to apply deep learning to the task of leaf–wood separation, Xi et al. [2] trained a 3D fully convolutional neural network to classify each point in TLS point clouds as either foliage, stem or branch. Their network processed the input clouds in $128 \times 128 \times 128$ blocks of uniform 5 cm voxels, with occupancy, average intensity and average z-coordinate used as input feature representations of each voxel [2].

Windrim and Bryson [203] proposed a pipeline where individual trees are first extracted from raster images of an ALS point cloud using Faster R-CNN [204] and subsequently segmented into three classes, foliage, upper stem and lower stem, using a VoxNet [83] inspired encoder-decoder 3D CNN. In a later work [205], the authors improved the performance of their pipeline by including LiDAR intensity as one of the input features for the semantic segmentation network, now opting to classify stem points into just one class. In addition, they tested a PointNet [15] based semantic segmentation network, which performed slightly worse then the 3D CNN [205].

Morel et al. [202] combined geometric leaf–wood separation algorithms with deep learning. Their approach first partitions the point clouds into overlapping batches of uniform size, which are then used as input for a modified PointNet++ [137]. Both point coordinates and geometric descriptors are used as input features, and final classification of points in overlapping regions of two batches is decided such that the overlap is classified as wood if either batch is classified as wood. The model was trained using simulated TLS point clouds of individual trees, but demonstrated great performance on real TLS data as well [202].

Somewhat similarly, Shen et al. [206] introduced a pipeline where TLS point clouds are first partitioned into geometrically consistent regions using energy segmentation. Their novel geometric feature balance module then balances the amount of partitions representing different dominant geometric features by creating additional samples of underrepresented features through data augmentation. Finally, the balanced data is used for training a PointCNN [139] model to classify points into one of four classes, ground, foliage, stem and others, representing points not in any of the former three classes [206].

Kim et al. [207] and Wielgosz et al. [5] both utilized PointNet++ for semantic segmentation of TLS forest point clouds, with the former assigning each point to one of foliage, stem and branch, while the latter opted to leave out the branch class in favor of ground. Additionally, Wielgosz et al. [5] investigated using the semantic segmentation results for delineating individual trees with graph-based algorithm and proposed a Bayesian strategy for hyperparameter optimization.

A persistent problem with leaf–wood separation methods is that they are often designed with an individual scanning method in mind, e.g. TLS or ALS, which limits their transferability to data captured with other sensors [3]. This is especially a problem in the context of deep learning, where training a model may take multiple days and requires manually labeled data which is laborious to produce. To this end, Krisanski et al. [3] proposed a sensor agnostic semantic segmentation model designed to simultaneously handle point clouds from multiple sources. This was achieved by training a modified PointNet++ model on a diverse data set comprised of TLS, MLS, ALS and UAV ALS point clouds [3].

Kaijaluoto et al. [4] semantically segmented MLS forest point clouds using raw (non-georeferenced) 2D laser scanner measurements in 2D raster format. With their 2D CNN based on U-Net [145] using only range, reflectance and echo deviation as input features, they achieved performance on par with RandLA-Net [138]. Their novel approach to LiDAR semantic segmentation requires much less contextual information for classifying each point and is fast enough for real-time applications [4].

Although most DL-based leaf–wood separation methods opt for using modified versions of existing point cloud semantic segmentation architectures and focus most research efforts on data preprocessing and feature engineering, some more novel semantic segmentation networks have also been proposed.

LWSNet [48] augmented a standard 3D U-Net [145] with local contextual feature enhancement via a rearrangement attention mechanism and residual connection optimization during the upsampling stage, both designed specifically for improving leaf–wood separation performance. On the other hand, MDC-Net [208] introduced a novel multi-directional collaborative convolutional module designed for extracting discriminative features for leaf–wood separation. Based on their proposed module, they designed a multiscale CNN which utilizes the down- and up-sampling modules of PointNet++. When trained on TLS point clouds depicting individual trees, both

LWSNet and MDC-Net outperformed multiple popular semantic segmentation methods in the task of leaf–wood separation [208].

While many works have proposed forest point cloud processing frameworks comprised of both semantic and instance segmentation (see e.g. [5, 203, 205]), all of them have performed the process sequentially with one segmentation before the other. However, Xiang et al. [6] recently introduced ForAINet for panoptic segmentation, i.e. simultaneous semantic and instance segmentation, of forest data. ForAINet utilizes a 3D SCNN based on U-Net for feature extraction followed by three separate branches for semantic prediction, center-offset prediction and feature embedding. The input features include both LiDAR information such as intensity and return number as well as geometric descriptors. ForAINet achieves state-of-the-art performance on the FOR-Instance [44] benchmark data set comprised of high definition ALS point clouds.

3 Research Material

Since the objective of our research is investigating unsupervised semantic segmentation of multispectral ALS forest data, no existing benchmark data set is suitable for our purposes. To this end, we create a novel point cloud data set depicting boreal forest environment with more than eight million points that have been manually labeled as either wood or foliage. This section describes the acquisition, preprocessing and annotating of our proposed multispectral data set.

3.1 Data Acquisition

Essentially all of the models designed for unsupervised semantic segmentation of point clouds presented in Section 2.4.3 utilize RGB color information whenever it is available [10, 168, 169], and while LiDAR sensors are not capable of capturing color information, they can capture reflectance values. Studies have shown that the reflectance characteristics of objects can vary significantly depending on both the material of the object and the wavelength of the LiDAR [25]. As such, reflectance values can be utilized in manner similar to color information.

Multispectral LiDAR data has been shown to improve performance across a variety of remote sensing tasks. Hakula et al. [12], Yu et al. [209] and Kukkonen et al. [210] all found that using MS point clouds improved the species classification accuracy of individual trees in boreal forests in comparison to monospectral data. Similarly, utilizing intensity information from multiple wavelengths improved the accuracy of land cover classification, a type of semantic segmentation task, for both machine learning [211, 212], and deep learning [13] based classifiers. As such, utilizing LiDAR reflectance values from multiple different wavelengths should, in theory, also lead to performance improvements when it comes to the task of leaf–wood separation.

Our dataset was captured with FGI's in-house developed laser scanner system HeliALS-TW, which consists of a total of three different Riegl LiDAR scanners (Riegl GmbH, Austria), specifically VUX-1HA, miniVUX-3UAV, and VQ-840-G, which we refer to as scanners 1, 2 and 3 respectively. The respective wavelengths of the three scanners are 1,550 nm, 905 nm, and 532 nm. Other scanner specific characteristics are summarized in Table 1. In order to accurately track its position and orientation, HeliALS-TW is also equipped with a positioning system consisting of a NovAtel ISA-100C IMU, a NovAtel PwrPak7 GNSS receiver, and a NovAtel GNSS-850 antenna [12].

The HeliALS-TW system was mounted to a helicopter and MS point clouds were collected in the summer of 2021 from multiple test sites, five of which were used in this study. The helicopter was flown over each test site from two perpendicular directions at an approximate altitude of 80 m above ground level. In order to increase the number of trunk returns, the scan planes were set at an angle of 15° forwards with respect to the xz-plane [12].

Subsequent to data collection, the trajectory of the HeliALS-TW system was computed using both Waypoint Intertial Explorer [213] and a single virtual GNSS base station from Trimnet service (RINEX 3.04), which was positioned approximately

Table 1: The specifications of the three individual LiDAR scanners in the multispectral HeliALS-TW system. Since the beam of scanner 2 is elliptical, the beam divergence and diameter are expressed as two values. The maximum scanning angle of scanner 3 is expressed as *angle in flight direction* $\times$ *the angle perpendicular to flight direction*. Note that the approximate point density is higher than the number of sent pulses as there are often multiple returns from a single pulse in forested areas. The table is reproduced based on [12], where the same system was used.

Scanner	**1**	**2**	**3**
Model	VUX-1HA	miniVUX-3UAV	VQ-840-G
Wavelength (nm)	1,550	905	532
Flight altitude (m)	80	80	80
Approximate point density (points/m^2)	1,400	500	1,600
Number of returns	12	5	15
Maximum scanning angle (°)	360	120	28×40
Laser beam divergence (mrad)	0.5	0.5×1.6	1
Laser beam diameter at ground level (cm)	4	4×12.8	8
Pulse repetition rate (kHz)	1,017	300	200

at the center of each test site. The raw individual scans were then georeferenced utilizing the GNSS and IMU data in the Riegl RiPROCESS software, resulting in dense georeferenced point clouds from all three scanners [12].

The georeferenced monospectral data from scanners 1, 2 and 3 were combined into a multispectral point cloud using k-nearest neighbors interpolation with $k = 1$. For each point, the nearest neighbors were retrieved from each of the three point clouds corresponding to the three different scanners. Subsequently, the reflectances of the nearest neighbors were concatenated, resulting in a multispectral cloud with three reflectance values for each point. For points that had no neighbors from one or more scanners within a threshold distance of 0.25 m, the fields corresponding to the missing scanners were left empty. Finally, the predefined circular test sites with a diameter of 55 m were cut from the preprocessed MS point cloud. The final MS point cloud of one such test site is visualized in Figure 14 (a) on page 72.

3.2 Point Cloud Normalization and Ground Removal

Having constructed the MS point cloud, we normalized it by subtracting the ground elevation from the z-coordinates. The ground elevation was estimated using a digital terrain model (DTM). In order to construct the DTM, the point cloud was first divided into a rectangular grid along the xy-plane. Subsequently, we applied the method of Hyyppä et al. [214] without Gaussian smoothing to obtain the lowest point within each cell in the grid. The lowest points were then utilized for extracting smooth surfaces from the point cloud with the ground extraction algorithm of Lehtomäki et al. [215] and the DTM was reconstructed from the obtained surface patches. Finally, the detected ground points were removed from the normalized point cloud. A comparison of one original test site point cloud and the normalized point cloud with the ground removed can be seen in Figures 14 (a)–(b).

3.3 Manual Annotation of Data

Although training an unsupervised model requires no manually annotated data, incorporating some labeled data is nevertheless essential for quantitatively evaluating model performance and objectively comparing various configurations. Furthermore, while the entire training data set does not need to have class annotations, the labeled portion should be representative of the overall data set in order for the performance measures to be accurate. It follows that for our forest data, the annotated sections should contain forest of different densities and all major tree species present in the data, namely Scots pine (*Pinus sylvestris* L.), Norway spruce (*Picea abies* (L.) H. Karst), Downy birch (*Betula pubescens* Ehrh.) and Silver birch (*Betula pendula* Roth). To this end, we chose to add class annotations for two of the five circular plots with 55 m diameter. The remaining three plots are left unlabeled and can only be used as training data by our unsupervised model. The first of the two plots is relatively sparse and consists of almost exclusively pine trees. Conversely, the second plot is quite dense, consisting primarily of birches and spruces.

The two plots were manually annotated using the point cloud processing tool CloudCompare [216] and a workflow similar to [5]. That is, the annotation consisted of two separate steps: during the first step, we performed instance segmentation of individual trees by separating each section of the point cloud that could reasonably be identified as a tree. Only trees with approximately 50% or more of their points located within the test plots were segmented. The minimum height for a tree was set to be around three meters and any object lower than that was simply considered understory vegetation and ignored during the instance segmentation. The branches of adjacent trees with intertwined crowns were separated to the degree that it was practically possible. In the second step, we carried out semantic annotation on each tree instance segmented during the first step. The individual points were classified into one of two classes, wood, which comprises the stem and branches, and foliage, which consists of everything else, such as leaves and needles. All points that were not considered part of a tree instance in the first step, i.e. points that were part of either understory vegetation or partial trees with less than 50% visible, were classified as foliage, unless they were clearly part of a standalone branch or stem, in which case they were classified as wood.

The choice of ground truth classes in the context of leaf–wood separation and semantic segmentation of forest data is somewhat ambiguous. In fact, based on our review of the literature, three distinct approaches for defining the ground truth classes see consistent use, the most popular of which appears to be splitting trees into wooden components and foliage. The aforementioned split has been widely used in the context of machine learning [194, 197], deep learning [3, 48, 202, 206, 208] as well as geometry and radiometry based algorithms [1, 165, 166, 187]. Some more recent works have opted to further separate the wood component into two classes, stem and branch [44, 207]. The third ground truth definition divides the point cloud

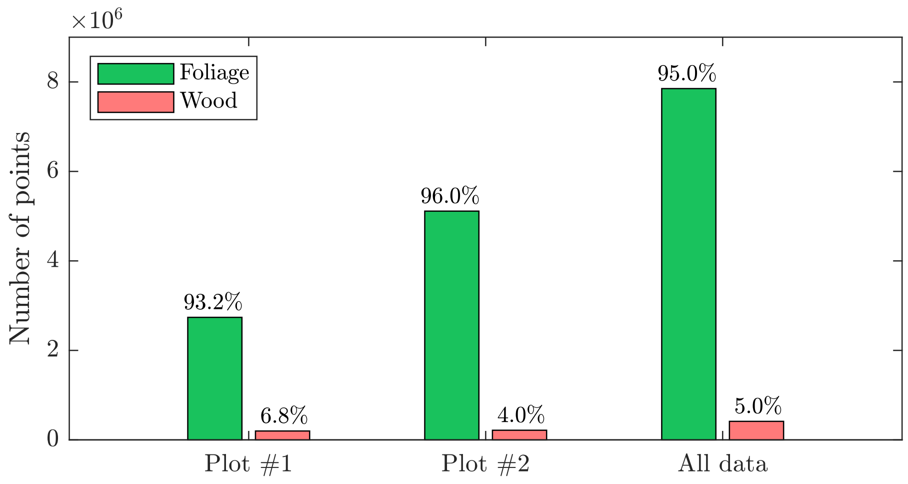


Figure 12: Class distribution of the two manually annotated plots separately and as a combined data set. The notable imbalance between classes displayed in the distribution is a problem that is commonly experienced with real world data.

into stem and foliage, that is, branches are considered a part of the foliage. This type of ground truth has mostly seen use in deep learning models [4, 5, 205].

Motivated by the popularity of the wood and foliage split for ground truth, we elected to use it as well. It is worth noting that dividing the points into stem and foliage appears to mostly occur in the context of deep learning applications. The split appears slightly artificial, since branches are intuitively considerably more similar to stems than leaves or needles. In fact, Kaijaluoto et al. [4] noted a tendency for deep learning based methods to misclassify some branches, especially larger ones, as stem when branches were considered foliage. They conjectured this was a result of the surface texture and reflective properties of branches being quite similar to the stem. The aforementioned observation is further evidence that considering the stem and branches as part of the same class is the more reasonable choice.

In total, the entire annotation process lasted around 270 hours and was carried out by a single annotator. As is often the case when manually annotating point clouds, some number of erroneously labeled points are to be expected (see e.g. [4, 42, 44]). In other words, the manual annotations are by no means perfect, since discerning between classes is practically impossible in some cases even for a human annotator. This is especially true when it comes to forest data, since it is often impossible to say where a tree trunk or a branch ends and foliage starts as there's no clear difference between the two. Kaijaluoto et al. [4] noted similar problems in the context of manually annotating MLS forest data, and the considerably sparser ALS data that we use may further amplify the issue.

Altogether, 356 individual trees were segmented from the two plots, 89 and 267 from plots #1 and #2 respectively. The constructed data set, containing both the individual trees and the remaining points from both plots, comprised over 8 million manually annotated points, with around 400,000 wood points and over 7.6 million foliage points. The exact class distributions are shown in Figure 12, both for the entire

(a) (b) (c)

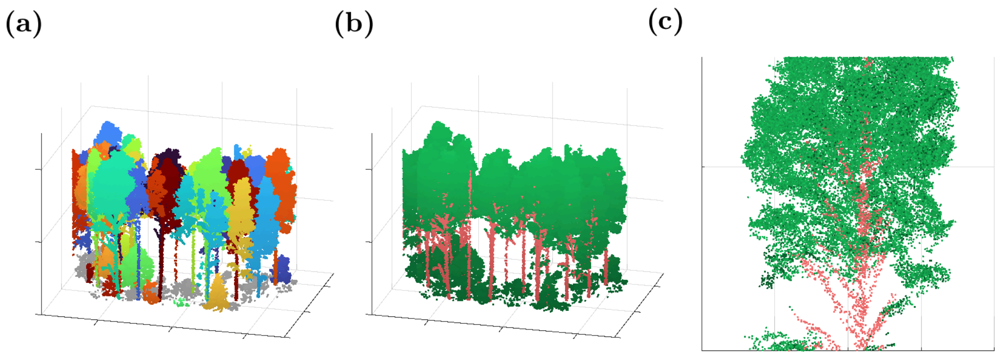


Figure 13: Sample of the manually annotated point cloud of plot #2. (a) Instance segmentation of trees. Each individual tree has been colored with a distinct color. Points colored gray are not part of any segmented tree instance. Note that some of the visualized trees are missing parts, since only a section of the full point cloud is displayed. (b) Semantic annotations for each individual point. The colors red and green represent wood and foliage points respectively. (c) Part of an individual tree showing the finer details of the semantic annotation.

data set and the two annotated plots separately. Figure 12 effectively demonstrates the imbalanced number of points between the classes in our data, a problem frequently experienced with data sets captured in natural environments, including many popular point cloud data sets [36]. An example of the instance and semantic annotations for a section of plot #2 is shown in Figure 13.

3.4 Training and Test Data

The cylindrical test plots of diameter 55 m contain around 2–10 million points on average. Storing the test plot point clouds on the graphics processing unit (GPU) would require a very large amount of memory, which prevents us form using the full test plots as input data for training neural networks on our hardware. As such, the raw data was further divided into smaller cylindrical neighborhoods. The number of points in each neighborhood varied between 50 and 410,000 points across the entire data set, with 135,000 points being the average. Figure 14 summarizes the preprocessing steps conducted to produce the training data.

There are a multitude ways of defining the local neighborhood of points when dividing a point cloud into smaller sections, for example cubic boxes, spheres and cylinders are popular choices in the literature [217]. We opted to use cylindrical neighborhoods following Xiang et al. [217], who utilized them for the task of panoptic segmentation of point clouds representing urban street areas and forests. Since panoptic segmentation is simply simultaneous instance and semantic segmentation, cylindrical neighborhoods are a reasonable choice for our present semantic segmentation problem.

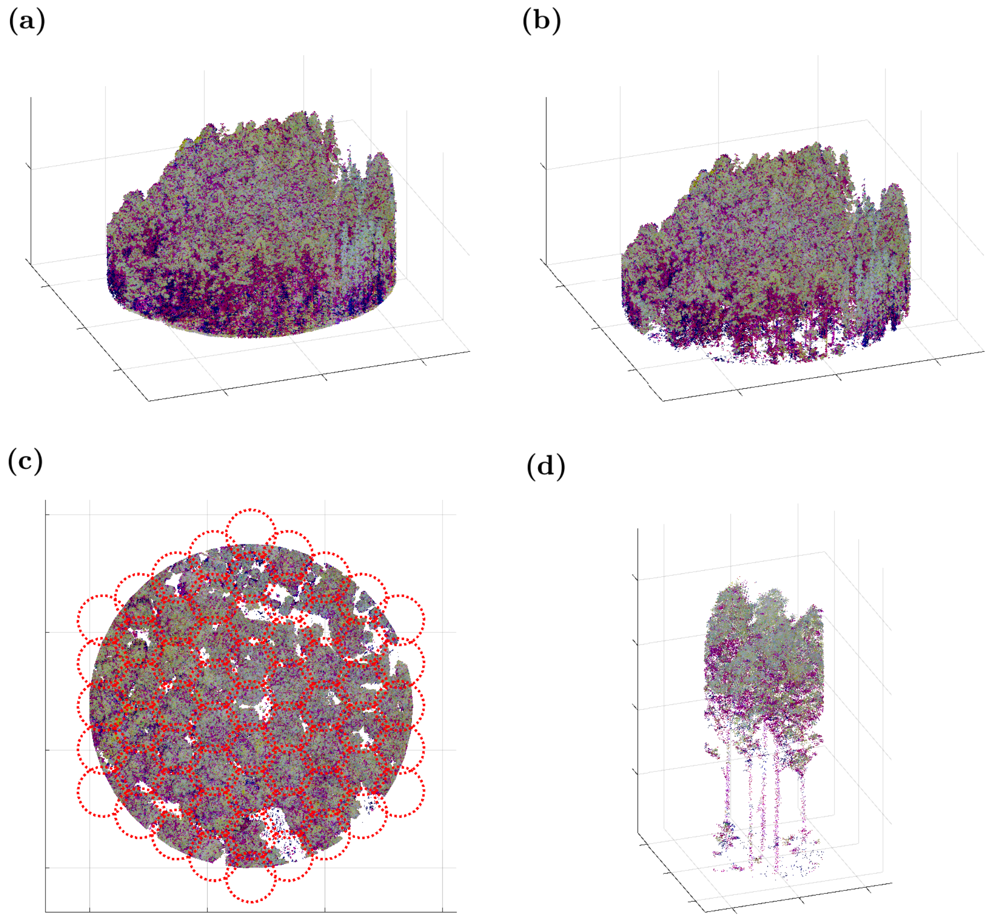


Figure 14: Visualization of the data preprocessing steps for plot #2. Note that the colors used for points in this image are pseudo colors created by using scaled reflectance values from scanners 1, 2 and 3 for the red, green and blue channels respectively. (a) Original multispectral point cloud of the forest plot. (b) Normalized point cloud from which the ground points and elevation have been removed. (c) Outlines of the smaller cylindrical neighborhoods of radius $r_c = 4.2$ m used as training data superimposed on the normalized point cloud in the xy-plane. (d) Example of a smaller cylindrical neighborhood that has been cut out from the normalized point cloud.

[217] found that cylindrical neighborhoods have two advantages over other methods: firstly, they avoid cutting objects, including trees, along the vertical axis. Secondly, they ensure computational efficiency, since cylindrical neighborhoods are compliant with many efficient neighborhood search algorithms, including fast radius search.

The problem of dividing the cylindrical test sites into smaller cylindrical neighborhoods can be simplified to covering a circle with multiple smaller circles in the xy-plane. However, excluding trivial solutions, a circle can not be covered by circles without a degree of overlap between some of them. This overlap, however, should be minimized. Firstly, because large overlaps can adversely affect how well the model generalizes to test data, since the model is trained on the same data points multiple times. Secondly, minimizing the overlap simultaneously minimizes the amount of computational resources that are wasted on predicting labels for the same data points on multiple instances at evaluation time. In order to optimize the overlap, we choose the distance d_c between the centers of adjacent overlapping circles according to the following equation:

$$d_c = \sqrt{3} r_c, \tag{41}$$

where r_c is the radius of the circle. This will result in a hexagonal lattice arrangement (see Figure 14 (c)), which has been shown to be the most optimal method of covering a plane with circles in terms of minimizing the area of overlap [218].

The radius of the cylinders was set to $r_c = 4.2$ m in order to achieve a number of points within cylinders that can feasibly be processed during training, while maintaining a large enough radius that the cylinders contain primarily complete trees. Furthermore, the chosen radius practically minimizes the amount of empty space in the cylinders around the perimeter of the test sites.

Finally, we center the coordinates within each cylinder by subtracting the corresponding means from the x- and y-coordinates and the minimum value from the z-coordinates. In addition, we normalize the reflectance features from each of the three laser scanners. The normalization is conducted separately for each channel in our multispectral data. As the reflectance values may contains outliers, for example when a point is missing the reflectance data from one or more channels, we choose to apply a normalization scheme that is robust to outliers. We begin by computing the interquartile difference (IQR), that is, the difference between the 75th and 25th percentile:

$$\mathrm{IQR}^j = Q_{75}(\mathbf{x}^j_{\text{reflectance}}) - Q_{25}(\mathbf{x}^j_{\text{reflectance}}), \tag{42}$$

where $Q_i(\cdot)$ is the ith sample percentile and $\mathbf{x}^j_{\text{reflectance}}$ denotes the reflectance values of channel $j \in \{1, 2, 3\}$. Subsequently, we subtract the median and divide the result by IQR^j:

$$\tilde{\mathbf{x}}^j_{\text{reflectance}} = \frac{\mathbf{x}^j_{\text{reflectance}} - Q_{50}(\mathbf{x}^j_{\text{reflectance}})}{\mathrm{IQR}}.$$

The final normalized reflectance vector $\bar{\mathbf{x}}^j_{\text{reflectance}}$ is then computed by subtracting the minimum:

$$\bar{\mathbf{x}}^j_{\text{reflectance}} = \tilde{\mathbf{x}}^j_{\text{reflectance}} - \min(\tilde{\mathbf{x}}^j_{\text{reflectance}}). \tag{43}$$

From each of the two annotated plots, we selected a continuous section of 12 cylinders as a test set, which is around 20% of the full data set. The other ~ 80% of the data was used for training. Although unsupervised DL does not necessarily require any labeled training data, an annotated training split was nevertheless required for training our fully-supervised baselines. Due to the relatively small amount of labeled data, we opted to not further split the training data into separate validation and training sets. The exact split is shown in Figure 15.

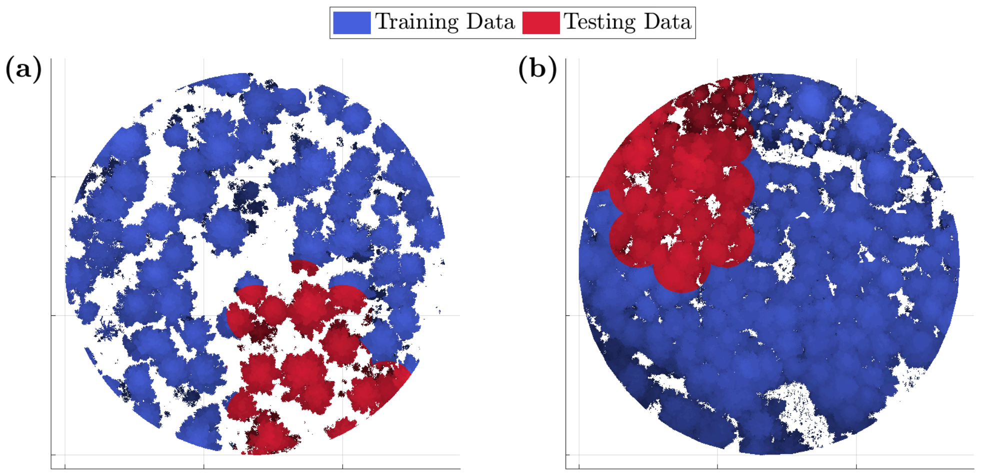


Figure 15: Visualization of the split between training and testing data for our manually labeled data set. (a) Train-test split for plot #1. (b) Train-test split for plot #2.

3.5 Data Augmentation

We adopt the three data augmentation techniques utilized by the original GrowSP model [10], namely *rotation*, *scaling* and *translation*, all of which are standard data augmentation methods in the field [82]. Furthermore, we opt to use the same hyperparameters as GrowSP for each of the augmentation methods. All augmentations are applied randomly on-the-fly during training. The employed augmentation methods and their hyperparameters are described in more detail below.

Rotation: As the name suggests, this augmentation method entails rotating the point cloud around the coordinate axes and is often parameterized by the maximum permitted amount of rotation around each of the axes. In our case, the maximum rotation in radians is bound to the interval $[-\frac{\pi}{32}, \frac{\pi}{32}]$ for the x- and y-axes, while the full range $[-\pi, \pi]$ is allowed around the z-axis. The amount of rotation is sampled separately for each axis from a uniform distribution parameterized by the corresponding rotation bounds.

Scaling: The point cloud xyz-coordinates are randomly scaled by multiplying them with a scaling coefficient sampled from the uniform distribution $\mathcal{U}(0.9, 1.1)$. The same scaling coefficient is used for all axes.

Translation: In the context of point clouds, translation refers to a simple additive shift applied on the xyz-coordinates. By default, the translation is randomly sampled from the uniform distribution $\mathcal{U}(0, 50)$ (meters) separately for each of the three coordinate axes.

3.6 Populating Missing Reflectance Values

As mentioned in Section 3.1, all points in the multispectral point cloud do not contain reflectance values from all three scanners. This poses a problem, especially in the unsupervised learning context, since a missing reflectance value is theoretically quite a notable semantic characteristic. However, in practice two points that are missing a reflectance value are not necessarily semantically similar at all, as the missing values are most often due to occlusions. As such, there is a risk that the model simply groups all points with missing reflectance values into one class, regardless of whether the point is wood or foliage.

In an attempt to minimize the effect of missing reflectance values, we propose a simple strategy for populating the empty reflectance fields based on superpoints. Once the initial superpoints have been constructed using the algorithm detailed in Section 4.1.2, we simply replace all missing reflectance values with the mean of the available values within each superpoint. In the rare case that none of the points within a superpoint contain data for some reflectance channel, the mean reflectance value of the closest superpoint with reflectance data available is used instead. The populating is handled separately for each of the three reflectance channels.

4 Research Methods

This section presents the methodology utilized in this thesis. First, Section 4.1 details the original GrowSP architecture and our modified version, GrowSP-ForMS. Next, Section 4.2 describes three supervised and two unsupervised leaf–wood separation methods, which are used as performance baselines for GrowSP-ForMS, followed by a list of the experiments and evaluation metrics used for assessing the performance of GrowSP-ForMS in Section 4.3. Finally, Section 4.4 describes the process of training the neural networks and optimizing the hyperparameters of the unsupervised baselines.

4.1 Neural Network Architecture

In this section we present our proposed neural network architecture for unsupervised semantic segmentation of multispectral forest point clouds, which has been adapted from the GrowSP architecture [10]. We first give a detailed description of the original GrowSP architecture in Section 4.1.1 and then present our proposed modifications to the base architecture in Section 4.1.2.

4.1.1 GrowSP Architecture

We choose to utilize GrowSP [10] over the two other existing neural network architectures desgined for unsupervised semantic segmentation of point clouds, PointDC [169] and U3DS^3 [168], for two reasons. Firstly, as mentioned in Section 2.4.3, GrowSP and U3DS^3 perform similarly on popular benchmark datasets in terms of segmentation accuracy, while both clearly outperform PointDC. Secondly, the source code of GrowSP has been made publicly available by the original authors[2], which is not the case for U3DS^3. Utilizing an official implementation of the base architecture ensures that performance is not hindered by an incorrectly implemented model, which is a considerable risk if we were to program the entire model from scratch.

GrowSP approaches unsupervised semantic segmentation of point clouds as a joint problem of point cloud feature extraction and clustering. A majority of the architecture consists of three main modules, a *superpoint constructor*, a *neural network feature extractor* and a *semantic primitive clustering module*.

Let us have a training data set composed of $H \in \mathbb{Z}^+$ three-dimensional point clouds $\{\mathcal{P}_1, \ldots, \mathcal{P}_H\}$, where each point cloud contains the x, y and z coordinates for each point, and optionally color. Each point cloud in the data set represents an individual training input. Then, given an individual point cloud $\mathcal{P}_i$ with N points, the feature extractor returns a feature vector for each individual point $\mathcal{F}_i \in \mathbb{R}^{N \times K}$, where $K \in \mathbb{Z}^+$ is a user defined embedding length, e.g. $K = 128$. The extracted features $\mathcal{F}_i$ and the input point cloud $\mathcal{P}_i$ are then given as input for the superpoint constructor, which will progressively generate larger and fewer superpoints as the training epoch increases. Finally, the superpoints are passed as input for the semantic primitive clustering module, which generates a pseudo label for each superpoint. We

[2]https://github.com/vLAR-group/GrowSP

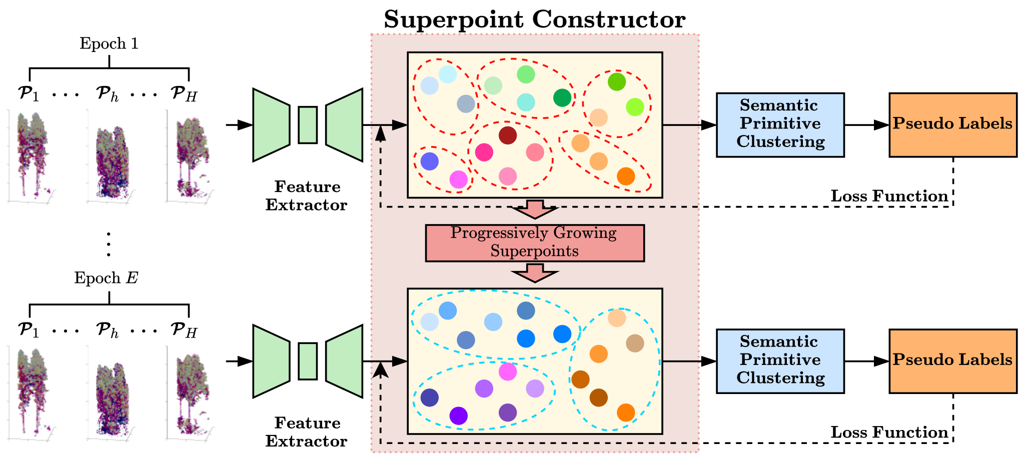

Superpoint Constructor
Epoch 1
$\mathcal{P}_1 \cdots \mathcal{P}_h \cdots \mathcal{P}_H$
Feature Extractor
Semantic Primitive Clustering
Pseudo Labels
Loss Function
Progressively Growing Superpoints
Epoch E
$\mathcal{P}_1 \cdots \mathcal{P}_h \cdots \mathcal{P}_H$
Feature Extractor
Semantic Primitive Clustering
Pseudo Labels
Loss Function

Figure 16: Overview of the generic GrowSP model architecture. The learning framework is comprised of three main modules: a superpoint constructor, a neural network feature extractor and a semantic primitive clustering module.

then use these pseudo labels in place of the ground truth labels in the loss function to optimize the parameters of the feature extractor network. The architecture of the generic GrowSP network is summarized in Figure 16 and below we discuss the most fundamental parts in more detail.

Initial Superpoints: Superpoints are larger sections of the original point cloud, such that each point within the same superpoint is ideally a part of the same semantic category. The authors of GrowSP argue that a superpoint is more likely to have geometric meaning in comparison to a single point, which simplifies the process of extracting high-level semantic information.

During early stages of training GrowSP, the features extracted from the point cloud are effectively random. As such, leveraging some existing algorithm is a more reliable method for constructing the initial superpoints. GrowSP adopts a combination of VCCS [170] and region growing [171]. VCCS first divides the input point cloud into a voxel grid with a user defined resolution R_{vox}. Subsequently, it distributes a set of seed points evenly on the voxel grid and incrementally expands them into superpoints. To further reduce the number of superpoints, a region growing algorithm is then applied on the original point cloud. If a majority of points within a VCCS superpoint are part of the same region, the entire superpoint is merged to the region in question. VCCS and region growing both account for spatial distances as well as distances between surface normals and normalized RGB values. We denote the set of initial superpoints of an input point cloud $\mathcal{P}_i$ by $\{\mathbf{s}_i^1, \dots, \mathbf{s}_i^{M^0}\}$, where M^0 is the number of superpoints in $\mathcal{P}_i$. It is worth noting that the number of initial superpoints M^0 generally differs between distinct point clouds.

Feature Extractor: In principle, any neural network capable of extracting K-dimensional features for each point of a point cloud $\mathcal{P}_i$ could be used as a feature

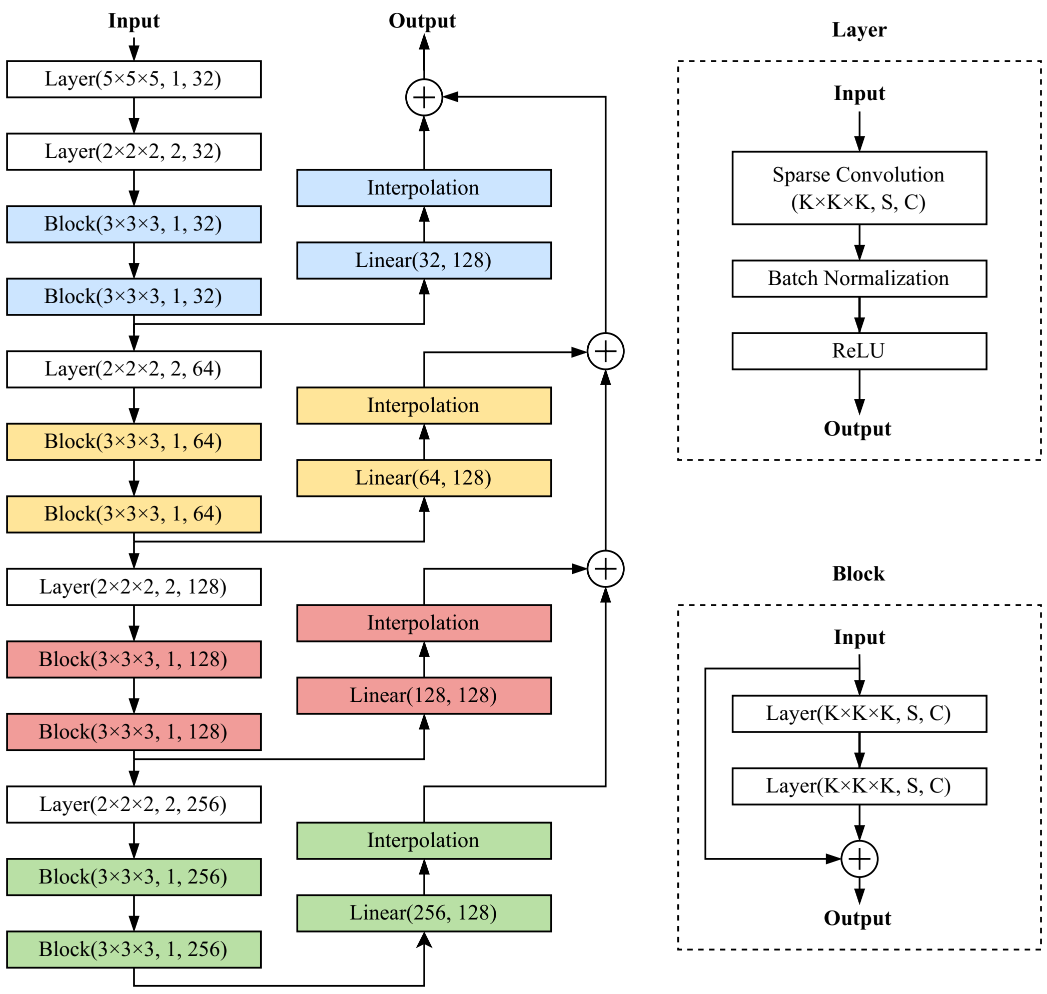


Figure 17: Architecture of the ResNet16FPN neural network used as a feature extractor in GrowSP. The hyperparameters K, S and C are kernel size, stride and number of output channels respectively.

extractor in GrowSP. The authors opt to use Sparse Convolutional Neural Networks, specifically Minkowski convolutional neural networks [80], which generalize the 3D sparse convolutions proposed by [79] for generic input and output coordinates in addition to arbitrary kernel shapes.

The architecture of the chosen feature extractor consists of an encoder and a decoder. The former is a residual neural network [101], particularly ResNet16, while the latter is a feature pyramid network (FPN) -type [219] architecture, where outputs from each stage of the ResNet are utilized. To be exact, the decoder consists of 4 MLPs which produce 128-dimensional features from the output of the ResNet16 at different levels. The outputs of the MLPs are then interpolated and added together to produce the final per-point 128-dimensional features. The entire architecture of the feature extractor, which we refer to as ResNet16FPN, is illustrated in Figure 17.

Progressively Growing Superpoints: Once the feature extractor has been trained

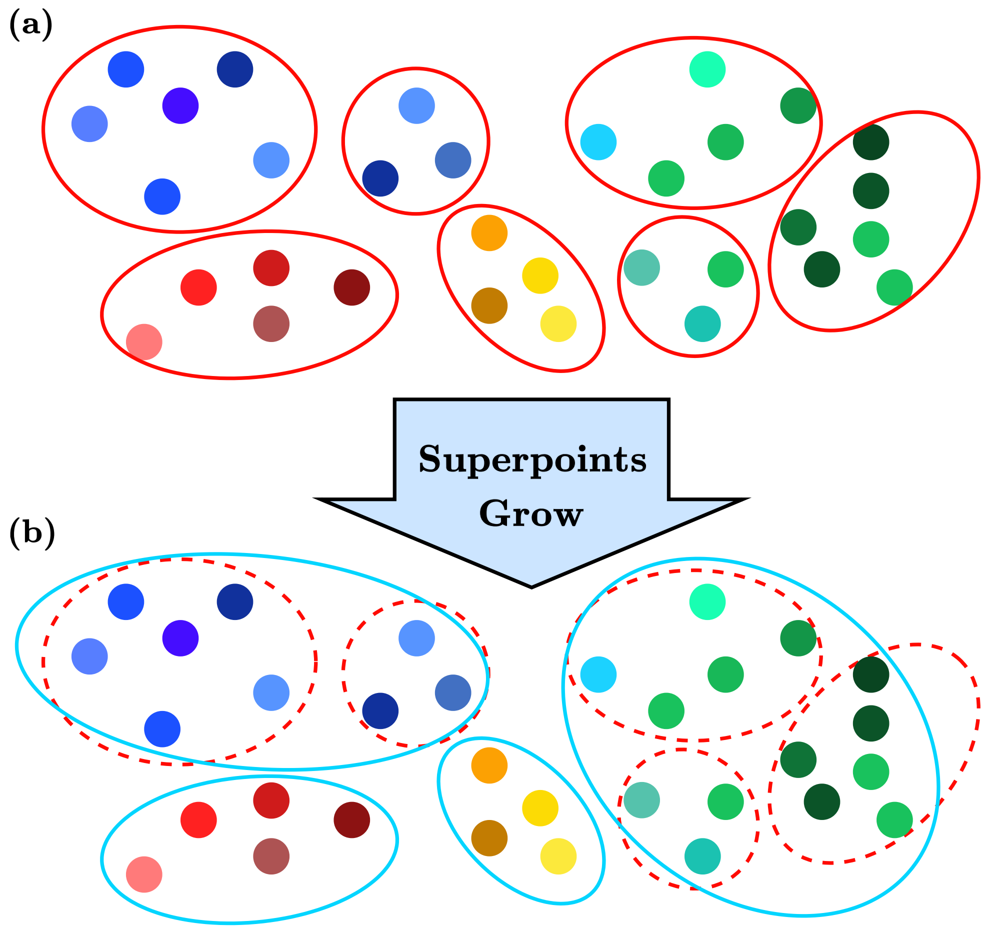


Figure 18: Illustration of progressively growing superpoints. Each dot represents a neural embedding of a single three-dimensional point. (a) Initial superpoints. Each red ellipse represents one superpoint. (b) Larger superpoints after initial superpoints are merged based on the extracted features. The turquoise ellipses represent the new superpoints. Note how all superpoints are not necessarily merged in every iteration.

for a sufficient number of epochs, the extracted features are expected to become increasingly more informative. As such, once the pretraining phase has concluded, GrowSP begins merging the initial superpoints into larger ones based primarily on the extracted neural features. Figure 18 illustrates the principle of growing the superpoints iteratively, where spatially neighboring superpoints with similar feature representations are merged into larger superpoints.

In order to merge the initial superpoints, GrowSP first constructs superpoint level representations based on the point level neural features. For each input point cloud $\mathcal{P}_i$, we have the features extracted by the backbone neural network $\mathcal{F}_i \in \mathbb{R}^{N \times K}$ and the set of initial superpoints $\{\mathbf{s}_i^1 \dots \mathbf{s}_i^{M^0}\}$. GrowSP then constructs a feature representation for each superpoint $\mathbf{s}_i^j \subset \mathcal{P}_i,\ j \in \{1, \dots, M^0\}$ by computing the mean of the individual features corresponding to points in $\mathbf{s}_i^j$:

$$\bar{\mathbf{f}}_i^j = \frac{1}{|\mathbf{s}_i^j|} \sum_{p \in \mathbf{s}_i^j} \mathbf{f}_i^p, \tag{44}$$

where $\mathbf{f}_i^p \in \mathbb{R}^{1 \times K}$ is the feature vector retrieved from $\mathcal{F}_i$ corresponding to the individual point $p \in \mathbf{s}_i^j$.

In addition to the neural features, GrowSP utilizes superpoint coordinates $\bar{\mathbf{p}}_i^j \in \mathbb{R}^{1\times 3}$, colors $\bar{\mathbf{c}}_i^j \in \mathbb{R}^{1\times 3}$ and estimated surface normals $\bar{\mathbf{n}}_i^j \in \mathbb{R}^{1\times 3}$ when merging superpoints. The superpoint level representations are computed in the exact same manner as the neural feature representations $\bar{\mathbf{f}}_i^j$, that is, by averaging over the relevant features within the superpoint. All superpoint level features are then concatenated into a single vector:

$$\tilde{\mathbf{f}}_i^j = \bar{\mathbf{f}}_i^j \oplus w_{\text{xyz}} \cdot \bar{\mathbf{p}}_i^j \oplus w_{\text{rgb}} \cdot \bar{\mathbf{c}}_i^j \oplus w_{\text{norm}} \cdot \bar{\mathbf{n}}_i^j, \tag{45}$$

where $w_{\text{xyz}}, w_{\text{rgb}}, w_{\text{xyz}} \in \mathbb{R}^+$ are scalar weights for the coordinates, colors and surface normals respectively and $\oplus$ denotes concatenation.

Having these initial superpoint features $\{\tilde{\mathbf{f}}_i^0, \ldots \tilde{\mathbf{f}}_i^{M^0}\}$, GrowSP merges the M^0 initial superpoints into $M^1 < M^0$ larger superpoints with the k-means clustering algorithm [220]. The superpoint growing is conducted independently for each point cloud $\mathcal{P}_i$ and every time $\widehat{E} \in \mathbb{Z}^+$ epochs have passed, where $\widehat{E}$ is some predefined number. Assuming the superpoints grow a total of T times, the final number of superpoints M^T is also a predefined value such that $M^0 < M^1 < \ldots < M^T$. Subsequent to growing, the superpoints are passed as input for the semantic primitive clustering module.

Semantic Primitive Clustering: Each of the H point clouds $\mathcal{P}_i$ in the data set contain some number M_i of superpoints, where the number of superpoint varies by epoch. In essence, all of these superpoints together can be viewed as a large set of primitive semantic elements, which should somehow be grouped together in order to discover the actual semantic classes in the data. Zhang et al. [10] empirically find that simply clustering all superpoints into $|C|$ categories is excessively aggressive and greatly hinders the model performance. This is a result of the fact that superpoints belonging to different classes are often quite similar. If they are then erroneously assigned into the same category in early stages of training, GrowSP generally fails to correct this error over time. As such, the authors opt to group the superpoints into a relatively large number of semantic primitives at all stages of training the network.

The method used for grouping superpoints into semantic primitives is very similar to that which was used for merging the initial superpoints and GrowSP utilizes the exact same strategy for constructing the superpoint level neural embeddings $\bar{\mathbf{f}}_i^j$. However, since pseudo labels are required from the beginning of the training, the feature extractor cannot be pretrained prior to clustering the superpoints for the first time, which was the case with growing superpoints. A problem arises: during early stages of training the model, the extracted features are basically meaningless. As such, clustering the superpoints into semantic primitives based on only the extracted neural features is not a reliable option. To this end, GrowSP utilizes point feature histograms (PFH) [221], which are widely used geometric features based on estimated surface normals of the point cloud. PFHs can be computed for multiple dimensions and thus the authors opt to use 10-dimensional PFHs in GrowSP, which we denote by $\mathbf{f}_i^j$. In addition, superpoint level color information $\bar{\mathbf{c}}_i^j$ is also utilized if available. As was the case with progressively growing superpoints, the neural embeddings are simply concatenated with the other feature vectors:

$$\hat{\mathbf{f}}_i^j = \bar{\mathbf{f}}_i^j \oplus w_{\text{pfh}} \cdot \mathbf{f}_i^j \oplus w_{\text{rgb}} \cdot \bar{\mathbf{c}}_i^j, \tag{46}$$

where $w_{\text{pfh}}, w_{\text{rgb}} \in \mathbb{R}^+$ are scalar weights for the corresponding feature vectors. The resulting features $\{\hat{\mathbf{f}}_i^0, \dots, \hat{\mathbf{f}}_i^{M^0}\}$ from each point cloud $\mathcal{P}_i$ are then clustered into S semantic primitives using the k-means algorithm. In contrast to the superpoint constructor, all superpoints across all point clouds in the data set $\{\mathcal{P}_1, \dots, \mathcal{P}_H\}$ are clustered simultaneously, instead of processing each point cloud independently.

Having formed the semantic primitives, the semantic primitive labels of each superpoint are saved as S-dimensional one-hot pseudo labels. Subsequently, we construct a linear classifier from the S estimated centroids from the k-means algorithm. The centroid of each primitive $i \in \{1, \dots, S\}$ is computed by averaging over the neural features of the superpoints that were assigned to it. The PFHs and colors are discarded prior to the centroid computation. The classifier can then be used to obtain S-dimensional logits for each individual point in a given neural embedding $\mathcal{F}_i$. Identically to the progressive growing of superpoints, semantic primitive clustering is conducted once every $\widehat{E}$ epochs.

Training and Evaluating: As discussed in Section 2.4.3, GrowSP is based on progressively growing superpoints. However, in actuality the training consists of two phases: a pretraining phase, during which only the initial superpoints are used, and a growing phase, during which the size of superpoints is increased iteratively.

During training, we first pass a batch of point clouds $\mathcal{B}$ through the feature extractor, and subsequently pass the extracted features $\mathcal{F}_i$ to the semantic primitive classifier to obtain S-dimensional logits for each point. Standard cross-entropy loss is then applied between the logits and the previously saved pseudo labels, which we utilize as a replacement for ground truth labels. GrowSP uses a SGD optimizer with a polynomial learning rate scheduler to optimize the network parameters. The pretraining and growing phases use a separate optimizer and scheduler, although the hyperparameters are exactly the same for both.

To evaluate the network, that is, to obtain actual semantic class predictions, the S centroids of the semantic primitive classifier are simply clustered into $|C|$ classes using the k-means algorithm, where C is the set of ground truth classes. The obtained $|C|$ centroids are then used as a classifier to make predictions. Notably, the classifier only requires the per-point neural embeddings from the feature extractor network to predict the semantic class, that is, initial superpoints are no longer needed once GrowSP has been trained. Another advantage of GrowSP is that since only semantic primitives are learnt during training, the number of classes $|C|$ does not necessarily need to be defined prior to training the model.

Matching Predicted Labels to Ground Truth: The only information about the ground truth labels provided to GrowSP during training is the number of classes $|C|$. As such, assuming labeled data for assessing the predictive performance of the model is available, the numeric values of the predicted labels do not necessarily match the ground truth. For example, the predicted label signifying class i can be "class 5", while the corresponding class in the ground truth is "class 0".

In order to match predicted class labels to the ground truth classes when evaluating model accuracy with labeled data, GrowSP adopts the same approach as Cho et al. [222], where the problem is modeled as a linear assignment problem: Let us denote the unmatched and matched predicted labels of point n by $\tilde{y}_n$ and $\hat{y}_n$ respectively. Similarly, let y_n denote the ground truth label of the nth point. Finally, let us denote the set of predicted classes by $\widetilde{C}$ and the ith element in a set by C_i. The goal is to find a bijective mapping from $\widetilde{C}$ to C, such that the overlaps between the labels in each class $i \in \widetilde{C}$ and the corresponding class $j \in C$ are maximized.

We begin by constructing a confusion matrix $\mathbf{A}$, in which the element on the ith row and jth column, $\mathbf{A}_{ij}$, is the number of existing pairs $(\tilde{y}_n, y_n)$ such that $\tilde{y}_n = \widetilde{C}_i$ and $y_n = C_j$. Let $\mathbf{A}_{\max}$ then be the maximum element in $\mathbf{A}$ and let $\mathbf{X}$ denote the mapping matrix. The linear assignment problem for finding a mapping from $\widetilde{C}$ to C is formulated as follows:

$$\min_{\mathbf{X}}. \sum_{i=1}^{|\widetilde{C}|} \sum_{j=1}^{|C|} \left(\mathbf{A}_{\max} - \mathbf{A}_{ij}\right) \mathbf{X}_{ij} \tag{47}$$

$$\text{s.t.:} \sum_{i=1}^{|\widetilde{C}|} \mathbf{X}_{ij} = 1, \ \forall j \in \{1, \ldots, |C|\} \tag{48}$$

$$\sum_{j=1}^{|C|} \mathbf{X}_{ij} = 1, \ \forall i \in \{1, \ldots, |\widetilde{C}|\} \tag{49}$$

$$\mathbf{X}_{ij} \in \{0, 1\}, \ \forall (i, j) \in \{1, \ldots, |\widetilde{C}|\} \times \{1, \ldots, |C|\}. \tag{50}$$

The optimal result $\mathbf{X}^*$ is a mapping from the predicted class labels to the ground truth labels such that if $\mathbf{X}^*{}_{ij} = 1$, then $\widetilde{C}_i$ is mapped to C_j. GrowSP then utilizes the Hungarian method [223] to solve the linear assignment problem and forms the actual predicted labels $\hat{y}_n$ based on the result.

4.1.2 Proposed Modifications to GrowSP

While the generic GrowSP model can be utilized for semantic segmentation of multispectral forest point clouds to some success based on our experiments, we found that its performance can be drastically improved with a few relatively straightforward changes to the model architecture. In this section we present our proposed modifications which constitute *GrowSP for forest area multispectral point clouds*, referred to as GrowSP-ForMS from here on out.

Geometric Point Cloud Descriptors: The generic GrowSP model utilizes 10-dimensional point feature histograms as geometric point cloud descriptors to improve semantic primitive clustering in the early stages of training, where the extracted neural features are essentially random. The PFHs are computed at superpoint level based on estimated surface normals of the point cloud. However, we found that the surface normals within complex forest canopy structures can be quite heterogeneous, that is, the normals frequently vary highly even within small geometrically consistent areas. As such, while PFHs are a reasonable choice when it comes to segmenting point clouds with simpler geometry, such as indoor scenes, they are not necessarily suitable for forest environments. To this end, we adopt another set of geometric point cloud descriptors as specified by Hackel et al. [224], which are based on the eigenvalues of point neighborhood covariance tensors.

Given a point cloud $\mathcal{P}_j \in \mathbb{R}^{N\times 3}$, the covariance tensor of a point $p_i \in \mathbb{R}^3$ is computed as follows:

$$\Sigma_i = \frac{1}{K} \sum_{p_k \in \mathbf{P}_i} (p_k - \bar{p})(p_k - \bar{p})^\top, \tag{51}$$

where $\mathbf{P}_i = \{p_i^1, \dots, p_i^K\}$ is a set comprised of the K nearest neighbors of p_i and $\bar{p} = \text{med}_{p_k \in \mathbf{P}_i}(p_k)$ is its medoid. Multiple distinct geometric descriptors can then computed based on the eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ and eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ of the covariance tensor Σ_i [224]. We choose to use linearity, planarity, sphericity, verticality and the first principal component (PCA1), which are formally defined as follows for each point $p_i \in \mathcal{P}_j$:

$$\text{Linearity}(p_i) = \frac{\lambda_1 - \lambda_2}{\lambda_1} \tag{52}$$

$$\text{Planarity}(p_i) = \frac{\lambda_2 - \lambda_3}{\lambda_1} \tag{53}$$

$$\text{Sphericity}(p_i) = \frac{\lambda_3}{\lambda_1} \tag{54}$$

$$\text{Verticality}(p_i) = 1 - |\hat{\mathbf{z}}^\top \mathbf{e}_3| \tag{55}$$

$$\text{PCA1}(p_i) = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3}, \tag{56}$$

where $\hat{\mathbf{z}} = [0, 0, 1]$ [224]. The linearity, planarity and shpericity features provide information about the presence of linear 1D structures, planar 2D structures and volumetric 3D structures respectively [225]. On the other hand, verticality describes the prevalence of linear structures specifically along the z-axis. Notably, the values of the presented eigenvalue-based geometric descriptors are always within the interval $[0, 1]$, which is a convenient property in the context of deep learning, as the features will generally not require further normalization.

In addition to the features described above, various other geometric descriptors can be computed using the eigenvalues of the covariance matrix, including features such as anisotropy and eigenentropy [225], which aim to describe the point distribution within a given neighborhood in a more general sense. The five features described above were chosen specifically with leaf–wood separation in mind, and in fact most have been previously utilized in the task by both deep learning models and traditional algorithms. As an example, Shen et al. [206] proposed a geometric feature balance model utilizing linearity, planarity, sphericity and verticality, which enhanced the performance of PointCNN [139] in semantic segmentation of TLS forest data. Similarly, multiple unsupervised algorithms have successfully utilized linearity and verticality [1, 165, 166] as well as sphericity and PCA1 [190] for leaf–wood separation. Finally, in addition to the previous success of the chosen features, extensive visual analysis on our proposed multispectral data set revealed all of the chosen descriptors distinguish certain subsets of the wood points quite effectively, while the disregarded features seemingly exhibit no such properties.

Figure 19 shows a comparison between our chosen geometric features and the first point feature histogram descriptor from 10-dimensional PFHs. As can be seen in the figure, the geometric features appear much more effective when it comes to distinguishing wood points from foliage. For example, linearity (a) highlights small branches, while the verticality (d) of the stem is notably higher in comparison to the surrounding points. By contrast, the first PFH (f) does not appear to highlight any specific region of the tree, rather, the feature values are seemingly random. The situation is similar for the other nine PFHs, which were left out from the visualization for the sake of brevity.

The definition of the neighborhood $\mathbf{P}_i$ in Equation 51 is an important factor when it comes to computing the covariance tensor Σ_i, as it can significantly effect the feature relevance in the context point cloud semantic segmentation [226]. Weinmann et al. [226] found that cylindrical neighborhoods are less suitable for the task, while spherical neighborhoods parameterized by either a radius or the number of nearest neighbors proved more applicable. We adopt a combination of the two strategies, where all points within a spherical neighborhood of radius r_n up to the K nearest neighbors are used for computing the covariance tensor. This way the distance to the furthest points remains reasonable in low density regions of the point cloud, while simultaneously the number of neighbors is kept reasonable when point density is high. Following Shcherbcheva et al. [190], the radius was set to $r_n = 0.35$ m.

Inspired by the eigenentropy-based adaptive neighborhood selection used by Weinmann et al. [227], we adopt a multiscale strategy for computing our geometric descriptors. First, the geometric features were computed for each $K \in \mathcal{K} = \{20, 50, 100, 150\}$. Subsequently, for each of our five geometric descriptors, the four vectors corresponding to the different neighborhood sizes were compiled into a single feature by computing the point-wise mean:

$$\mathrm{F}_{\text{mean}}(\mathcal{P}_i) = \frac{1}{|\mathcal{K}|} \sum_{K \in \mathcal{K}} \mathrm{F}_K(\mathcal{P}_i), \tag{57}$$

where $\mathrm{F}_K(\cdot)$ is some geometric feature computed for point cloud $\mathcal{P}_i$ using a maximum of K nearest neighbors within the spherical neighborhood. The goal of our approach is to minimize the variance of feature values within geometrically consistent regions caused by varying point densities. While simple, our method appears quite effective at

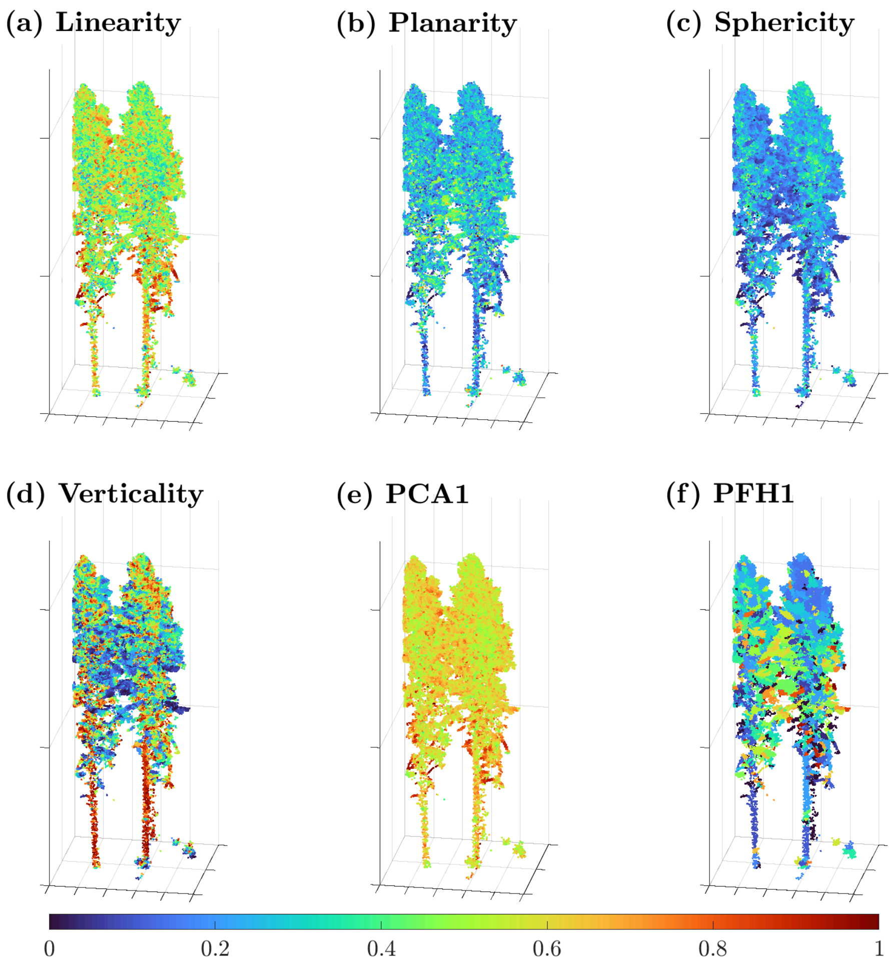


Figure 19: Comparison between geometric features and point feature histogram descriptors for one cylinder from plot #1. The geometric features have been computed for the entire plot prior to dividing it into smaller cylinders. (a) Linearity. (b) Planarity. (c) Sphericity. (d) Verticality. (e) First principal component. (d) First point feature histogram descriptor from 10-dimensional PFHs. Note that the values have been scaled to the range [0,1] for visualization purposes.

smoothing out the geometric descriptors of forest data based on our experiments.

Graph-based Superpoint Constructor: In an ideal situation each initial superpoint would contain labels from only one class, but this is generally never the case in practice. As such, assessing the accuracy of the utilized superpoint constructor is important, since superpoints comprised of multiple semantic classes will generally have an adverse effect on the performance of GrowSP. The accuracy of a superpoint constructor can be assessed based on a simple majority vote, such that for each superpoint, all of its points are assigned the most common ground truth label within that particular superpoint. Subsequently, the accuracy is computed by comparing the superpoint-based labels to the ground truth.

Based on our experiments, while the VCCS and region-growing-based superpoint constructor utilized by the generic GrowSP model functions well for geometrically simple indoor and outdoor areas, it is unsuitable for the complex structures present within forest canopy. That is, when applied to our MS forest data, a significant number of wood points are assigned into superpoints where foliage points are the majority. This effectively means that it is extremely difficult for the model to classify most wood points correctly, since the majority class within each superpoint will generally dominate the mean of the semantic information extracted from the superpoint. To this end, we propose an improved superpoint constructor designed specifically for point clouds depicting forest environments. Much like the original VCCS-based superpoint constructor, ours is comprised of two steps: we first construct initial superpoints using the graph-based method of Landrieu and Simonovsky [142] and subsequently merge small superpoints into similar larger ones based on distance and geometric point cloud descriptors.

The superpoint constructor of Landrieu and Simonovsky [142] groups point neighborhoods into superpoints based on geometric descriptors, rather than color and surface normals, which makes it more suitable for forest LiDAR data. In fact, Shen et al. [206] used the same method for segmenting artificially planted forest point clouds into geometrically similar regions in their PointCNN-based semantic segmentation model. Given a point cloud $\mathcal{P}_i$, the algorithm first computes a set of d_g geometric descriptors $f_j \in \mathbb{R}^{d_g}$ for each point $p_j \in \mathcal{P}_i$ [142]. Subsequently, the algorithm constructs a 10-nearest neighbor adjacency graph $G = (V, E)$ of $\mathcal{P}_i$, where the neighbors are determined with respect to the Euclidean distance between xyz-coordinates [142]. The set of initial superpoints is then defined as the constant connected components of the solution to the following optimization problem:

$$\underset{\{g_i\}\in\mathbb{R}^{d_g}}{\operatorname{argmin}} \sum_{i\in V} ||g_i - f_i||^2 + \lambda \sum_{(i,j)\in E} w_{ij}[g_i - g_j \neq 0], \tag{58}$$

where w_{ij} denotes the edge weight, defined as the inverse of the distance between points p_i and p_j, λ is the user defined regularization strength and $[\cdot]$ is the Iverson bracket. We then utilize the ℓ_0-cut pursuit algorithm of Landrieu and Obozinski [228] to find an approximate solution for Equation 58 and subsequently extract the superpoints by finding the constant connected components of the solution [142].

Landrieu and Simonovsky [142] utilized linearity, verticality, planarity, sphericity and elevation, defined as the z-coordinate of each point normalized over the entire input point cloud, as the geometric neighborhood descriptors f_j. On the other hand, Shen et al. [206] opted for the same geometric descriptors, but dropped the elevation feature. For our data set, we found that while linearity and verticality were clearly the most crucial features, the performance of the superpoint constructor was quite insensitive to the choice of any additional geometric descriptors. Ultimately we opted to use the same features as Shen et al. [206], as the combination yielded slightly better performance than other tested feature combinations. The geometric features f_j where further weighted by $w_{f_j} = 10$ and the regularization strength was set to $\lambda = 0.1$. We note that since the additional weighing by $w_{f_j} = 10$ cannot be formulated concisely, it was omitted from Equation 58.

Although the superpoint constructor of Landrieu and Simonovsky [142] yields a high accuracy on forest data, the number of superpoints per each input point cloud is notably higher than when using VCCS and region growing. Furthermore, many of the superpoints are comprised of just 2 to 5 points, or even only one point in some cases. This poses a problem, since if the number of superpoints in each input point cloud grows too large, both computing the superpoint level features and clustering the superpoints into semantic primitives becomes computationally extremely demanding to the point of being infeasible. Furthermore, very small superpoints are unlikely to contain a sufficient amount of semantic information. As such, some method for further merging small superpoints to similar adjacent ones is required. To this end, we propose a method based on nearest neighbors and geometric descriptors that utilizes an adaptive threshold for the minimum superpoint size. For a point cloud $\mathcal{P}_i$ and the corresponding initial ℓ_0-cut pursuit-based superpoints, the merging process is carried out as follows:

1. Define the maximum number of superpoints permitted within one input point cloud $\mathrm{SP}_{\mathrm{max}}$. For our data set we opt for $\mathrm{SP}_{\mathrm{max}} = 2,200$, which offers a great middle ground between accuracy and performance.

2. Determine the minimum number of points each superpoint must contain $\mathrm{PTS}_{\mathrm{min}}$. The threshold is adaptive, such that starting from an initial value of $\mathrm{PTS}^0_{\mathrm{min}} = 5$, the value is increased iteratively by one, until removing all superpoints with less than $\mathrm{PTS}_{\mathrm{min}}$ points yields at most $\mathrm{SP}_{\mathrm{max}}$ superpoints. All superpoints with at least $\mathrm{PTS}_{\mathrm{min}}$ points are then considered *admissible*.

3. Identify *singular superpoints* within the group of non-admissible superpoints. Here singular refers to superpoints comprised of two or less points.

4. Cluster singular superpoints into larger ones with DBSCAN [179]. For our data, the neighborhood radius and core point threshold were set to $\epsilon = 0.2$ and min_samples = 5 respectively. Besides point xyz-coordinates, the input matrix is augmented with the geometric descriptors that are seemingly the most effective at distinguishing stems and branches, namely verticality, linearity and PCA1. All resulting clusters with more than one point are then considered superpoints. We note that the VCCS-based superpoint constructor in the original GrowSP handles singular superpoints in a similar manner.

5. Merge each of the remaining non-admissible superpoints and the clustered singular superpoints into the closest admissible superpoint. Given a small superpoint, for each of its points we find the nearest point in $\mathcal{P}_i$ that is part of an admissible superpoint. The closest admissible superpoint is then the superpoint that most commonly occurs among the nearest points.

The initial superpoints generated by our graph-based superpoint constructor were also used for populating missing reflectance values when preprocessing the data (see Section 3.6).

Oversegmentation of Predicted Labels: We empirically observe that oversegmenting the semantic primitives at prediction time significantly improves model accuracy. Here oversegmentation refers to setting the number of clusters k to a value that is higher than the number of ground truth classes $|C|$, which yields a result where foliage and wood points have been further divided into multiple classes. Once the wooden components have been separated from the foliage, the oversegmented classes can easily be assigned to either wood or foliage based geometric properties of the point cloud, since a subset of a point cloud consisting of mostly wood points differs quite significantly from a subset comprised of foliage points in terms of descriptors such as linearity and verticality.

In practice, prediciting the classes using oversegmentation is implemented as follows: instead of using the actual set of ground truth classes C, we define a new set C_{over} with additional classes, such that $|C_{\text{over}}| > |C|$, which is used for oversegmenting the point cloud. Subsequently, we compute the average linearity of each oversegmented class and assign them to either wood or foliage based on a user defined linearity threshold L_{min}. In theory the oversegmentation could separate stem and branches into different classes as well, in which case introducing an additional verticality threshold may be necessary. However, in practice we find that the number of wood classes is always only one for our data set. For our multispectral data set, utilizing $|C_{\text{over}}| = 6$ classes and a linearity threshold of $L_{\text{min}} = 0.55$ yields the best performance.

Notably, since the training of GrowSP is entirely based on the semantic primitive labels, which are then clustered into $|C|$ classes at prediction time, neither the choice of $|C_{\text{over}}|$ nor L_{min} has any effect on the training. Consequently, both hyperparameters can be optimized once the model has already been trained. That is, having trained the model, the user can simply iterate over the set of reasonable values for $|C_{\text{over}}|$ and L_{min} and pick the combination that yields the best overall performance. This is a notable advantage, as our oversegmentation strategy effectively provides a significant increase in segmentation accuracy with no additional computational overhead during training.

We conjecture that this phenomenon stems from certain sections of the foliage being semantically more similar to the wooden components than other foliage in an unsupervised setting. Consequently, clustering the semantic primitives into two classes directly yields a worse result than performing an oversegmentation followed by detecting the wood components based on geometric properties. It is also intuitively reasonable that lower sections of the foliage that are constantly in shade would be semantically different from e.g. the tree crown. On the other hand, one might expect the foliage of a deciduous tree to be in a different class from trees with needles, as the two differ from each other quite significantly. At least the former hypothesis is seemingly confirmed by our experiments, where lower sections of the trees and tree tops are frequently assigned into a different class.

Oinonen et al. [229] observed a similar phenomenon while utilizing GrowSP for unsupervised semantic segmentation of multispectral LiDAR data depicting built environments, where oversegmenting significantly improved the semantic segmentation accuracy of numerous classes. However, in the absence of ground truth labels, automatically matching each oversegmented class into one of the classes in C is extremely difficult, if not impossible, as the cityscape data contains a large number of classes, many of which are geometrically quite similar. By contrast, in the case of leaf–wood separation, effectively leveraging this property of GrowSP is remarkably straightforward, since the only two existing classes are easily distinguished from each other by geometric descriptors once separated.

Decaying Clustering Weights: Since extracted neural features are initially meaningless, GrowSP leverages traditional features, such as color, intensity information and geometric descriptors, as supervision for the semantic primitive clustering. As the training progresses, the extracted features are expected to become more meaningful, while simultaneously the traditional features become less important for the clustering. In fact, since the extracted features are initially essentially random, the traditional features should be weighted quite heavily in early stages of training. On the other hand, if the traditional features are favored excessively, the extracted features do not influence the semantic primitive clustering at all and the model fails to learn any meaningful features. To this end, we propose utilizing decaying weights for the traditional features during semantic primitive clustering in the pretraining stage, such that the longer the model has been trained for, the less weight such features are given.

We adopt a simple strategy, where the clustering weights decay linearly as a function of the current epoch, until a certain minimum weight $w^*_{\text{coef}} \geq 0$ is reached. We empirically found that 20% of the full clustering weight, i.e. $w^*_{\text{coef}} = 0.2$, is a reasonable value for the minimum. The linear weight decay is then achieved by multiplying the constant clustering weights with a weight coefficient:

$$w_{\text{coef}} = \max\left(1 - \frac{E}{E_{\max}}, w^*_{\text{coef}}\right), \tag{59}$$

where E is the current epoch and $E_{\max}$ is the maximum epoch where decay is allowed. In our case, we set $E_{\max} = 100$. This way the pretraining stage has a stabilization period during which the weights remain constant prior to the phase of the training where the superpoints begin growing. Formally, the augmented feature vectors used for clustering semantic primitives are then defined as follows:

$$\hat{\mathbf{f}}_i^j = \bar{\mathbf{f}}_i^j \oplus w_{\text{coef}} \cdot w_{\text{geof}} \cdot \mathbf{f}_i^j \oplus w_{\text{coef}} \cdot w_{\text{rgb}} \cdot \bar{\mathbf{c}}_i^j, \tag{60}$$

where $\mathbf{f}$ now denotes the geometric descriptors instead of the 10-dimensional PFHs and $w_{\text{geof}} \in \mathbb{R}^+$ is the corresponding weight.

Other Changes: Since the scale of the extracted features and input features such as coordinates and color can differ quite significantly, GrowSP applies a point-wise L_2 normalization on each augmented feature tensor both while merging superpoints and when clustering semantic primitives. However, we empirically found that applying the normalization feature-wise yielded a more stable training procedure and better performance when using LiDAR-based forest data. It should be noted that the L_2 normalization is not explicitly mentioned in the original article, but is present in the source code of the official GrowSP implementation.

While merging superpoints, we opt to replace the L_2 normalization with feature-wise min-max normalization, which we found to be slightly more effective. In addition, the surface normals are replaced with the geometric descriptors, and none of the features are weighted.

4.2 Baseline Methods

In an effort to assess the performance of our proposed model in comparison to existing leaf–wood separation methods, we test three popular fully-supervised deep learning methods and two unsupervised geometry-based algorithms on our multispectral data set. While all of these methods were mentioned in our literature review (see Section 2.4 and Section 2.5), this section provides a more detailed description of each of our baseline architectures.

4.2.1 Supervised Baselines

Rather than opting for supervised DL models that are specifically designed for leaf–wood separation, we choose three classic point cloud semantic segmentation architectures as our supervised baselines, namely PointNet [15], RandLA-Net [138] and ResNet16FPN [79]. There are two fundamental reasons behind this choice: Firstly, most state-of-the-art leaf–wood separation models have no publicly available implementations and implementing the architecture from scratch poses a considerable risk of programming mistakes. Additionally, the model architecture descriptions frequently omit crucial details, which further complicates the process. On the other hand, supervised leaf–wood separation models are generally optimized for some specific data set, thus there is no guarantee the model would work for our data without significant modifications. As such, opting for popular point cloud semantic segmentation models with open source implementations available that contain no data specific optimizations is the more reasonable choice in our case.

Secondly, considering that our proposed model is the first unsupervised deep learning based leaf–wood separation model, the performance is expected to be well below the state-of-the-art fully-supervised architectures. By contrast, comparing the performance to more straightforward supervised models provides interesting insight into how close unsupervised models are to achieving performance that is comparable to some of the most popular fully-supervised architectures.

Below we give an overview of the architectures of our three chosen supervised baselines.

PointNet: PointNet [15] was the first deep learning model to take unordered point clouds directly as input and remains quite popular to this day. In fact, many DL based leaf–wood separation models are heavily inspired by or based on PointNet and its successor PointNet++ (see e.g. [3, 202]). As such, PointNet is a natural choice as one of our fully-supervised baselines and we adopt an open source PyTorch implementation[3] for our experiments. The architecture of PointNet is summarized in Figure 20 and below we describe its most notable features in more detail.

[3]https://github.com/yanx27/Pointnet_Pointnet2_pytorch

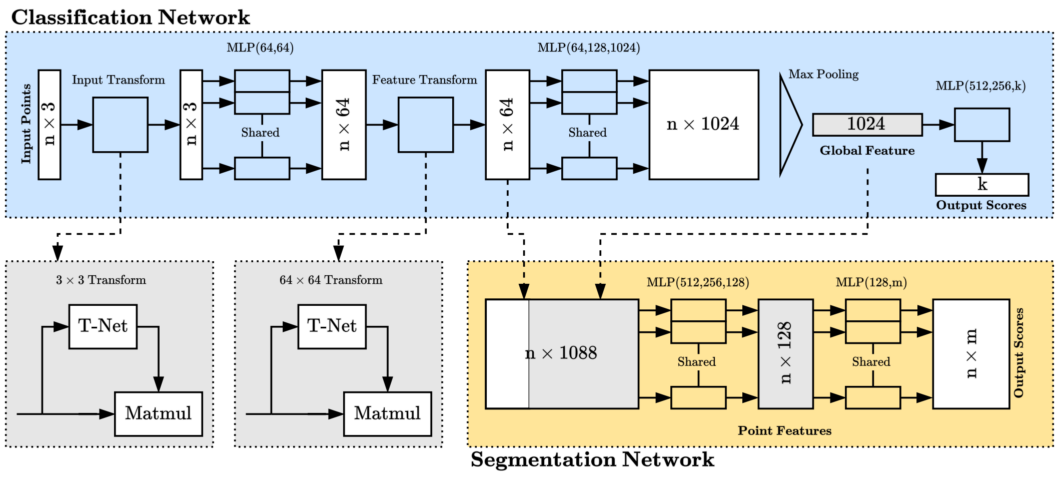


Figure 20: Schematic of the PointNet architecture, where n is the number of points in the input point cloud, k is the number of classes in the classification network and m is the number of classes in the segmentation network. Furthermore, *Matmul* denotes the standard matrix multiplication, while *MLP* represents a shared multilayer perceptron, with the subsequent numbers in brackets specifying the size of its layers. Batch normalization is applied on all layers that use the rectified linear unit as an activation and the last MLP in the classification network includes dropout layers. The figure is reproduced based on [15].

At its core, PointNet is a multilayer perceptron, with certain clever additions inspired by the following structural characteristics of point clouds:

1. Point clouds are unordered. In contrast to pixel based images or voxel based 3D objects, the individual points in a point cloud have no specified ordering. That is, the a set of N points can represent the exact same input point cloud in a total of $N!$ distinct orders, which the network needs to be invariant to.

2. Adjacent points are not isolated, rather, neighboring points frequently form meaningful subsets. As such, the network should be able to capture local structures as well as the interactions between these structures.

3. Point clouds are invariant under certain transformations. For example rotating or translating a point cloud does not alter what it represents, which the network should account for.

PointNet first extracts point level features using a collection of shared MLPs. That is, the same MLP is applied to each individual point in the input. Subsequently, the features are aggregated into a 1024-dimensional global feature vector using a global max pooling layer, which simultaneously makes the output permutation invariant.

While the global feature vector is well suited for tasks such as classification, point level segmentation requires both global and local information. To this end, the classification network is augmented with a segmentation network. The 1024-dimensional global feature vector of the input point cloud is concatenated with the

64-dimensional point-wise features, forming a 1088-dimensional feature for each individual point, comprised of both local and global information. Subsequently, the combined features are passed as input for the segmentation network, which extracts new per point features based on the aggregated information. This allows the segmentation network to predict point-wise characteristics that are reliant on both local geometry and global semantics, such as semantic classes [15].

PointNet achieves transformation invariance by predicting an affine transformation matrix using a small NN referred to as T-Net and subsequently applying the transformation on the input point cloud. Moreover, PointNet extends the transformation invariance to the feature space as well by applying the same process to the local point-wise features. However, in this case the transformation matrix is restricted to being close to orthogonal due to its high dimension. The architecture of T-Net closely resembles that of the classification network in Figure 20, being comprised of point-wise feature extraction, max pooling and fully connected layers.

RandLA-Net: Much like PointNet, RandLA-Net [138] is also a point based DL model that does not require discretizing the input point cloud. However, RandLA-Net includes some architectural choices designed specifically for efficient yet effective semantic segmentation of large-scale 3D point clouds, making it a great option for processing ALS forest environment data. Furthermore, RandLA-Net has previously been successfully utilized for semantic segmentation of MLS forest data [4]. In our experiments, we opt to use an open source PyTorch implementation of RandLA-Net[4], which is based on the official TensorFlow implementation.

In order to effectively reduce point density, RandLA-Net progressively downsamples the input point cloud on each neural layer. However, while the random sampling used by RandLA-Net is extremely efficient in comparison to other downsampling strategies, it can randomly discard crucial features. To prevent this, RandLA-Net introduces the local feature aggregation (LFA) module, which is applied in tandem with the downsampling. An LFA module, shown in Figure 21, retains prominent features by aggregating information within point neighborhoods. Each LFA module is comprised of *local spatial encoding* (LocSE) units and *attentive pooling* layers within a *dilated residual block* and is applied on every point in the input cloud in parallel [138].

Let $\mathcal{P}_j$ be an input point cloud with d-dimensional input features. Furthermore, let us denote the xyz-coordinates of the ith point in $\mathcal{P}_j$ by $p_i \in \mathbb{R}^3$ and the corresponding feature vector by $\mathbf{f}_i \in \mathbb{R}^d$. The LocSE block then first gathers the H nearest neighboring points of p_i using a simple k-NN search and subsequently applies a relative point position encoding on each of the H points $\{p_i^1, \dots, p_i^k, \dots, p_i^H\}$ as follows:

$$\mathbf{r}_i^k = \mathrm{MLP}\left(p_i \oplus p_i^k \oplus (p_i - p_i^k) \oplus ||p_i - p_i^k||\right), \tag{61}$$

where $||\cdot||$ is the Euclidean norm. Finally, the encoded relative point positions $\mathbf{r}_i^k$ are concatenated with the corresponding feature vectors $\mathbf{f}_i^k$. The output of a LocSE unit for

[4] https://github.com/matthiasverstraete/3d_recognizer/tree/main/randlanet

an arbitrary point p_i is then a set of H augmented feature vectors $\hat{\mathbf{F}}_i = \{\hat{\mathbf{f}}_i^1, \ldots, \hat{\mathbf{f}}_i^H\}$, such that [138]:

$$\hat{\mathbf{f}}_i^k = \mathbf{r}_i^k \oplus \mathbf{f}_i^k. \tag{62}$$

The purpose of the attentive pooling layer is to aggregate the set of neighboring point features $\hat{\mathbf{F}}_i$ by automatically learning the most important local features. The layer first computes attention scores for each feature using a shared function $g(\cdot)$, which is comprised of a shared MLP and a softmax function:

$$\mathbf{s}_i^k = g(\hat{\mathbf{f}}_i^k, \mathbf{W}), \tag{63}$$

where $\mathbf{W}$ is the learnable weight tensor of the shared MLP. The learned attention scores $\mathbf{s}_i^k$ are then utilized as a soft mask for weighting feature importance:

$$\tilde{\mathbf{f}}_i = \sum_{k=1}^{K} \hat{\mathbf{f}}_i^k \cdot \mathbf{s}_i^k. \tag{64}$$

In summary, given a point $p_i \in \mathcal{P}_j$, the combination of LocSE and attentive pooling units learn a corresponding feature vector $\tilde{\mathbf{f}}_i$ that aggregates geometric information and feature vectors of its H nearest neighboring points [138].

In order to minimize the loss of geometric details caused by the substantial downsampling of the input point cloud, the RandLA-Net architecture significantly increases the receptive field of each individual point. This is achieved by stacking multiple LocSE blocks and attentive pooling layers combined with a residual connection into a dilated residual block. As shown in Figure 21, the dilated residual blocks in the local feature aggregation module are comprised of two stacks of LocSE and attentive pooling units, which offers a great balance between computational efficiency and effectiveness [138]. We note that similar to standard convolutional layers, receptive field refers to the area of the input that affects a particular value within the output vector. In contrast to convolutional layers, however, the receptive field is comprised of individual points within some neighborhood rather than spatial locations within an tensor.

ResNet16FPN: Following the original GrowSP article [10], we opt to use the feature extractor of GrowSP as one of our supervised baselines. The only required modification is adjusting the neural embedding dimension $K = 128$ to the number of classes in the ground truth, which in our case is $K = 2$. For a more detailed overview of the network architecture, the reader is referred to the feature extractor description in Section 4.1.1.

In contrast to PointNet and RandLA-Net, ResNet16FPN is not a point based DL model, rather, it requires a sparse voxel representation of the point cloud as an input. As such, each point in the input cloud is assigned to a voxel within a uniform voxel grid. Following [10], we set the size of the voxels to 0.05 m. At prediction time each point is then assigned the label of the corresponding voxel.

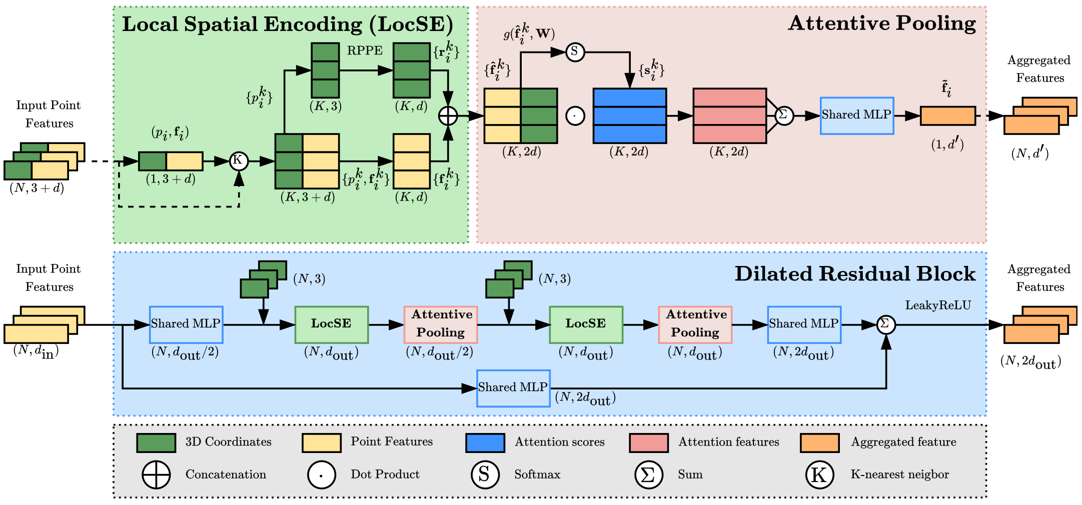


Figure 21: Schematic of the local feature aggregation module in the RandLA-Net neural network architecture. The two top panels show the local spatial encoding block, which extracts features from the input data, and the attentive pooling layer that assigns importance weights to the features based on local context and geometry. *RPPE* in the LocSE block denotes relative point position encoding. The middle panel illustrates how LocSE blocks and attentive pooling layers are stacked within a residual block in order to increase the size of the receptive field, while the bottom panel includes descriptions of the various elements in the schematic. The figure is reproduced based on [138].

4.2.2 Unsupervised Baselines

LeWoS: LeWoS [165] is a rather recent unsupervised algorithm for leaf–wood separation, based on point cloud connectivity and geometric descriptors. While the algorithm was originally designed for TLS data of individual tropical trees, based on their experiments the authors established it is applicable for complete point clouds from various forest environments [165]. What makes LeWoS an especially attractive choice for leaf–wood separation is that it only requires one user defined parameter, namely the feature similarity threshold N_{z_thres}. Furthermore, the authors found that the performance of LeWoS is in fact quite insensitive to the value of N_{z_thres} [165], which simplifies the process of parameter optimization. For the aforementioned reasons and the fact that the official MATLAB implementation of the algorithm is openly available[5], LeWoS was chosen as our first unsupervised baseline for leaf–wood separation.

The semantic segmentation process of LeWoS is comprised of three fundamental steps: recursive graph segmentation, class probability estimation and class regularization. In the recursive segmentation steps, LeWoS constructs an undirected graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ is a set of nodes, each corresponding to a point in the input point cloud, and E is a set of edges, where each element $e_{ij} \in E$ represents

[5] https://github.com/dwang520/LeWoS

an edge between neighboring points. The graph construction is based on point cloud density as well as geometric features, specifically verticality, which we denote by N_z [165].

The recursive graph segmentation begins by constructing an initial graph with an edge between each point p_i and its 10 nearest neighbors $\mathbf{P}_i = \{p_i^1, \ldots, p_i^{10}\}$. Subsequently, each edge e_{ij} between p_i and its neighbor p_i^j is pruned as follows:

$$e_{ij} = \begin{cases} \text{Keep}, & \text{if } \begin{cases} |N_{\mathrm{z}i} - N_{\mathrm{z}j}| < N_{\mathrm{z\_thres}}, \\ d_{ij} < \overline{d}_{ij} + \sigma_{d_{ij}}, \\ d_{ij} < \overline{\max(d_{ij})} + \sigma_{\max(d_{ij})} \end{cases} \\ \text{Prune}, & \text{otherwise}, \end{cases} \tag{65}$$

where $N_{\mathrm{z\_thres}}$ is the user defined feature similarity threshold, $d_{ij} = ||p_i - p_i^j||$ and $\overline{d}_{ij}$ and $\sigma_{d_{ij}}$ are the mean and standard deviation of d_{ij} within the neighborhood $\mathbf{P}_i$ respectively. Similarly, $\max(d_{ij})$ denotes the maximum value of d_{ij} within $\mathbf{P}_i$. As such, the first condition ensures that the feature values N_z within a neighborhood are similar, while the second condition controls local point cloud density connectivity. The third condition, while similar to the second, controls global density connectivity, as the mean and standard deviation are computed for the maximum neighborhood distances across the entire point cloud [165].

The final pruned graph can then be segmented into clusters of spatially adjacent points with similar features by finding the connected components. LeWoS recursively applies the graph segmentation method on the resulting clusters, such that geometric point features and neighborhoods are updated after each iteration [165]. The graph segmentation is applied recursively until no more clusters can be further segmented but for a maximum of 10 iterations [165]. Finally, in order to increase the prominence of the geometric linearity feature in branch clusters, the small cover set-based method of [183] is applied for splitting large branch clusters.

Subsequent to performing the recursive point cloud segmentation, LeWoS estimates a class probability for each cluster. The class of each segment is predicted based on both size, i.e. number of points, and the linearity of the segment, such that the segment is considered wood if it meets certain thresholds for both conditions. However, instead of directly assigning a class for each cluster based on some constant threshold, LeWoS tests a wide range of possible threshold values. More specifically, the threshold values used by LeWoS are $\{0.7, \ldots + 0.02 \ldots, 0.94\}$ and $\{10, \ldots + 2 \ldots, 50\}$ for linearity and size respectively, where $+d$ denotes multiple consecutive elements with a repeated difference of d. The class probabilities are then given by the relative frequency of each class prediction across the set of all predictions [165].

The estimated class probabilities form a so called *soft labeling* set S, where each point is represented by two probability values denoting the confidence that the point belongs in the wood and foliage classes respectively. Now, for reasons such as heavy data occlusion or abnormal growth angles, the wood probability of some branches may at times be insignificant. Consequently, directly classifying each point to the class with the highest probability is not necessarily the optimal choice, as it can yield a labeling with high variance even within small consistent neighborhoods, where a spatially smooth result would be preferable. To this end, LeWoS applies a class regularization on S, which aims to find a more spatially consistent labeling set S', while simultanously remaining as close as possible to the original labeling S. Formally, the class regularization problem can be modeled as follows:

$$S' = \underset{\mathbf{0} \in \Omega^V}{\text{argmin}} \left(\Phi(S, S') + \gamma \Psi(S') \right), \tag{66}$$

where $\Phi(\cdot)$ is a linear fidelity term, $\Psi(\cdot)$ is a regularizer that favours spatially smooth solutions, $\gamma > 0$ is the user defined regularization strength and Ω is the solution search space. Equation 66 is then solved using the α-expansion algorithm, which yields the set of improved and spatially smoothed leaf–wood labels S', that is, the final semantic classification [165]. S' is a vector of zeros and ones, corresponding to foliage and wood classes respectively.

GBS: We chose the graph-based leaf–wood separation (GBS) of Tian and Li [166] as our other unsupervised baseline. Similar to LeWoS, GBS only utilizes point cloud geometry, that is xyz-coordinates, in the segmentation process, disregarding any radiometric information. However, in contrast to LeWoS, GBS fully utilizes shortest path-based graph features in the task of leaf–wood separation. This makes GBS compact, yet efficient and the authors show its performance is invariant to tree species and size as well as sufficiently robust to data subsampling. When comparing the performance of GBS to existing methods, the authors found that GBS outperforms LeWoS for most tree species and is able to detect small branches more consistently. As such, GBS makes for an interesting unsupervised baseline alongside LeWoS. In our experiments we utilize an official implementation of the GBS algorithm, which the authors provide as an open source Python package [230].

Much like LeWoS, GBS begins by constructing an initial graph $G = (V, E)$ from the input point cloud, where the nodes are individual points p_i. However, instead of simply connecting the k nearest neighbors, GBS adopts a novel graph construction algorithm from the open-source Python library TLSeparation [231], which enables constructing a connected graph with fewer edges under the condition of occlusion [166]. Subsequent to creating the initial graph, GBS finds the shortest paths from all points to the root point using Djikstra's shortest path algorithm [166]. We denote the shortest path from the root point p_r to p_i by $\text{sp}(p_i) = (p_r, \ldots, p_i)$ and its length by $|\text{sp}(p_i)| = l_i$.

In the second step, GBS performs a multiscale point cloud segmentation based on the initial graph G. Firstly, the initial graph is pruned in order to remove falsely connected long edges and part edges between leaf and wood points. Formally, each edge $e_{ij} \in E$ is subjected to the following conditions

$$e_{ij} = \begin{cases} \text{Keep}, & \text{if } \begin{cases} d_{ij} \leq 2 \cdot \min\left(l_i - l_{\text{pre}(i)}, l_j - l_{\text{pre}(j)}\right) \\ \varphi(p_{\text{pre}(i)} p_i, p_{\text{pre}(j)} p_j) \leq \theta \end{cases} \\ \text{Prune}, & \text{otherwise}, \end{cases} \tag{67}$$

where $d_{ij} = ||p_i - p_i^j||$, $\text{pre}(i)$ denotes the index of the precursor of point p_i on $\text{sp}(p_i)$, $\varphi(\cdot)$ is the angle between two spatial vectors and θ is a user defined direction difference threshold. Secondly, the point cloud is segmented into equal interval layer bins based on the shortest path length. Each bin is then clustered by finding the connected components of the pruned graph G'. In an effort to improve the generalization of GBS across different tree species, it utilizes a multiscale strategy for segmenting. That is, the segmentation is performed using multiple distinct interval sizes, each of which corresponds to a single-scale segmentation result. The set of intervals I is a user defined parameter [166].

Following the multiscale point cloud segmentation, GBS performs an initial wood point extraction separately for each single-scale segmentation result. The wood point extraction is based on both the geometry and size of the clusters within the single-scale segmentation. Firstly, clusters that are either too large or too small are classified as foliage using a threshold that depends on the current interval size. Secondly, GBS utilizes cylindrical fitting to detect trunks and large branches. A cluster is labeled wood if the fitting error is under a user defined threshold E. Similarly, principal component analysis in combination with a linearity threshold L is used for recognizing small branches that exhibit linear properties. Finally, any remaining clusters are considered foliage [166].

Having extracted the initial wood points, GBS applies a label correction procedure that modifies the labels of misclassified wood clusters into foliage. The correction is heuristic in nature, being based on the observation that as the order of a branch increases, so does its linearity, while simultaneously the radius decreases. In order to improve the effects of the correction, it is generally applied for a total of three times [166].

Lastly, the final wood points are extracted by applying a region growth algorithm [171] on the pruned graph G', where the initial wood points from the previous step are used as seed points. The purpose of the final step is identifying wood points within irregular clusters, for example broken, curved or bifurcated branches, that could not be extracted based on cylindricity or linearity conditions [166].

4.3 Experiments and Evaluation Metrics

In order to gain a versatile understanding of the accuracy and capabilities of GrowSP-ForMS, we designed a collection of experiments that sought to answer the following questions:

1. Does multispectral LiDAR data provide an advantage over monospectral data when it comes to unsupervised deep learning-based leaf–wood separation?

2. What effect does each of our suggested changes have on the performance of the original GrowSP architecture individually?

3. Is our proposed graph-based superpoint constructor better suited for forest point cloud data than the VCCS-based superpoint constructor used by the generic GrowSP model?

4. How does the deep learning based GrowSP-ForMS architecture perform in the task of leaf–wood separation in comparison to traditional non-learning unsupervised algorithms LeWoS [165] and GBS [166]?

5. How well does the unsupervised GrowSP-ForMS architecture perform in comparison to popular fully-supervised architectures PointNet [15], RandLA-Net [138] and ResNet16FPN [79] in the task of leaf–wood separation?

These questions serve as an extension of the research objectives and questions presented in Section 1.1.

To conclusively answer question 1, we trained GrowSP-ForMS separately for each available intensity channel, all two-channel combinations and finally all three channels. Similarly, in order to answer question 2, we performed an ablation study, where starting with the generic GrowSP model we iteratively added each of our proposed improvements one by one, assessing the effects of each addition in the process. In the case of question 3, we simply generated the initial superpoints for our multispectral forest data set using both the original VCCS-based superpoint constructor and our new graph-based superpoint constructor. Their performance was then compared in terms of both accuracy and number of superpoints. Finally, for questions 4 and 5, we simply trained and evaluated our unsupervised and fully-supervised baselines on the same multispectral data set that GrowSP-ForMS was trained on and subsequently compared their performance to our best model.

The task of comparing the quantitative performance of different semantic segmentation models and assessing the effects of suggested changes to GrowSP calls for an objective measure of performance. To this end, we adopt the evaluation metrics most frequently used in literature concerning point cloud semantic segmentation: overall accuracy (oAcc), mean accuracy (mAcc), intersection over union (IoU) and mean intersection over union (mIoU).

Overall accuracy is defined as simply the proportion of correctly classified points across the set of all points. Using the previously introduced notation, overall accuracy can be calculated as follows:

$$\text{oAcc} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i). \tag{68}$$

However, overall accuracy alone is not an adequate metric for measuring model performance, as it tends to give an overly optimistic score in comparison to the true model performance, especially when there is a notable class imbalance. As an example, classifying each point in our EvoMS data set as foliage would yield an overall accuracy of nearly 95%.

Mean accuracy attempts to alleviate the problem of class imbalances by measuring how accurately distinct classes are identified on average. To compute the metric we first calculate the proportion of correctly classified points in each class $c \in C$ and then calculate the mean of accuracies across all classes:

$$\text{mAcc} = \frac{1}{|C|}\sum_{c\in C}\left(\frac{1}{N_c}\sum_{i=1}^{N} \mathbb{1}(y_i = c)\cdot\mathbb{1}(y_i = \hat{y}_i)\right), \tag{69}$$

where N_c is the number of points in class $c \in C$.

An even more powerful measure of model performance is intersection over union, often also referred to as the Jaccard index [232]. For class $c \in C$, IoU is defined as the ratio between the number of points correctly classified as class c and the number of points either classified or actually belonging to class c:

$$\mathrm{IoU}_c = \frac{\sum_{i=1}^{N} \mathbb{1}(y_i = c) \cdot \mathbb{1}(y_i = \hat{y}_i)}{\sum_{i=1}^{N} \mathbb{1}(y_i = c) + \mathbb{1}(\hat{y}_i = c) - \mathbb{1}(y_i = c) \cdot \mathbb{1}(y_i = \hat{y}_i)}. \tag{70}$$

The advantage intersection over union has in comparison to mean accuracy is that IoU penalizes both false positives and false negatives, while mAcc explicitly only accounts for the latter.

In order to combine the class-wise IoUs into a single measure of model performance, we can calculate the mean intersection over union by averaging the IoUs across all classes as follows:

$$\mathrm{mIoU} = \frac{1}{|C|} \sum_{c \in C} \mathrm{IoU}_c. \tag{71}$$

Since quantitative performance metrics can occasionally be misleading, regardless of how robust they are designed to be, we additionally evaluate the results qualitatively. This is carried out by simple visual inspection of the predicted classes of each semantic segmentation model and a manual comparison to the ground truth.

4.4 Training and Parameter Optimization

For all baselines we utilized open source implementations available online. All experimentation, parameter optimization and training of DL models was carried out on the same system. Specifications of the most essential hardware and software are listed in Table 2.

4.4.1 Supervised Deep Learning Models

In order to achieve comparable results for all fully-supervised baselines, we aimed to keep the training procedure as similar as possible between the distinct models. In practice this means that we used the cylinders of radius $r_c = 4.2$ m as input data and utilized the same data augmentation techniques (see Section 3.5) and optimization backend for all models. Besides xyz-coordinates, the three available intensity channels were used as input features for all three fully-supervised baselines. Similar to GrowSP-ForMS, we used the point cloud data where missing reflectance values had been populated using superpoints as described in Section 3.6. It should be noted that the original data with missing reflectance values was also experimented with, but initial results suggested that the populated data provided a slightly better performance. As such, all further testing was restricted to the data with populated reflectance values.

Following [4], who used the same configuration for semantic segmentation of MLS forest data, we utilize the Adam optimizer with the cyclical learning rate scheduler of [233]. Furthermore, we adopted the same scheduling policy, namely *triangular2*,

Table 2: Hardware and software specifications of the system used for training the deep learning models and all other experiments.

Device	**Specifications**
CPU	Intel® Core™ i7-10750H 5.0 GHz
GPU	NVIDIA® Quadro RTX™ 4000 Mobile 8 GB GDDR6
Memory	2×32 GB DDR4 3200 MHz
Software	
Ubuntu (OS)	22.04.3 LTS
GPU Driver	535.154.05
CUDA	11.3.1
Python	3.8.12
Conda	23.5.2
pip	21.2.4
PyTorch	1.10.2
Torchvision	0.11.3
Minkowski Engine	0.5.4
MATLAB	9.14.0.2337262 (R2023a) Update 5

Table 3: Base and maximum learning rates used for the supervised deep learning baselines. The values were determined using the learning rate test proposed by [233].

Model	**Base learning rate**	**Maximum learning rate**
PointNet [15]	0.001	0.0011
RandLA-Net [138]	0.002	0.0060
ResNet16FPN [79]	0.006	0.0090

where the learning rate alternates linearly between a set base and a maximum value, such that the maximum is halved each time it is reached. The length of one cycle was set to 10 epochs in all experiments. The values of the base and maximum learning rates are listed in Table 3 and were determined separately for each model using the learning rate test proposed by [233]. Finally, in contrast to [4], due to the considerable class imbalance in our training data, we minimize the focal loss with $\gamma = 2$ instead of the standard cross-entropy loss.

Both the different model architectures and available computational resources presented certain limitations on the input data format. Therefore the cylinders of radius $r_c = 4.2$ m had to be further preprocessed to enable training some of our supervised baselines. In addition, GPU memory limitations necessitated tuning the batch size and number of training epochs separately for each model. On the other hand, since we opted to not have separate validation data due to relatively small amount of manually annotated points, we could not follow the common practice of using validation loss and accuracy to asses when training should be concluded. As such, each model was trained until the training loss and mIoU had seemingly stabilized. The model specific additional data preprocessing steps and other hyperparameters are presented below.

PointNet: Due to the comparatively high memory usage of PointNet and limited computational resources, we could not train the model using the cylinders of radius $r_c = 4.2$ m as input. Inspired by the semantic segmentation pipeline in the original PointNet article [15], we divided the cylinders into 2.8 m $\times$ 2.8 m blocks. To enable using a batch sizes larger than $|\mathcal{B}| = 1$, PointNet requires the input clouds to have an uniform number of points. As such, we randomly sampled 10,000 points from each block during training, while at prediction time all points were provided as an input. The number of points during training was set to 10,000 as it was close to the maximum number of points across all blocks. This way any significant loss of geometric information due to downsampling was prevented.

We found the simple centering and IQR based normalization used for xyz-coordinates and reflectance respectively was not an adequate normalization strategy for training PointNet, as the loss did not converge. Therefore, following the semantic segmentation configuration in the original PointNet article [15], we normalized all input vectors to have values within the range $[0, 1]$. Consequently, in order to keep the coordinate values reasonable following data augmentation, the distribution used for sampling the random translations was changed to $\mathcal{U}(0, 0.1)$.

The batch size used during training PointNet was $|\mathcal{B}| = 8$. Multiple different batch sizes were experimented with and $|\mathcal{B}| = 8$ provided the most accurate results and improved training stability. It should however be noted that the maximum batch size permitted by available GPU memory was $|\mathcal{B}| = 12$. The model was trained for a total of 320 epochs.

RandLA-Net: While the efficiency of RandLA-Net enables using the original cylinders as input without further preprocessing, much like PointNet, RandLA-Net requires an uniform number of points in each input during training. To this end, we randomly sampled 80,000 points from each input cloud while training the model. The reasoning for setting the number of points to 80,000 was effectively the same as with PointNet. Moreover, following the same reasoning as with PointNet, we chose to utilize $[0, 1]$ normalization for the RandLA-Net input data.

In accordance with the original article [138], the number of neighbors used in the k-NN search performed by the LocSE units was set to $k = 16$. The batch size was set to $|\mathcal{B}| = 2$, which, due to GPU memory limitations, was also the maximum feasible batch size and the model was trained for 165 epochs.

ResNet16FPN: The batch size was set to $|\mathcal{B}| = 8$ empirically and the model was trained for 520 epochs. No further data preprocessing or downsampling was required and unlike PointNet and RandLA-Net, we utilized the same data normalization procedure as with GrowSP-ForMS for training ResNet16FPN.

4.4.2 Unsupervised Deep Learning Models

For training GrowSP-ForMS, we mostly adopted the configuration utilized in the original article [10]: the voxel size of the SCNN feature extractor was set to 0.05 m and cross-entropy loss was used as the cost function. Furthermore, the loss was minimized using an SGD optimizer and a polynomial learning rate scheduler with an initial learning rate and momentum of 0.1 and 0.9 respectively. The batch size during training was set to $|\mathcal{B}| = 4$, as it was the maximum permitted by the available GPU memory. For the same reason, the number of semantic primitives and extracted features were set to $S = 150$ and $K = 64$ respectively, as opposed to the values $S = 300$ and $K = 128$ utilized in the original article [10] (see Section 4.1.1). Finally, the weights given to reflectance features and geometric descriptors during semantic primitive clustering were $w_{\text{rgb}} = 1$ and $w_{\text{geof}} = 2$ respectively (see Equation 60).

The model was trained for $E_{\text{pretrain}} = 150$ epochs during the pretraining stage and $E_{\text{grow}} = 60$ epochs during the growth stage, decreasing the number of superpoints from $M^1 = 1500$ to $M^T = 1200$. It should be noted that although this is notably lower than the number of epochs GrowSP was trained for ($E_{\text{pretrain}} + E_{\text{grow}} \approx 1000$), we observed that training GrowSP-ForMS for a larger number of epochs yielded no additional performance gains and the training loss stabilizes quite early on our MS forest data set.

The number of superpoints in the initial merging $M^1 = 1500$ was proportional to that used by GrowSP ($M^1 = 80$) when considering that the number of initial superpoints is considerably higher for our forest environment data than benchmark data sets such as S3DIS and ScanNet. On the other hand, while the original GrowSP decreased the number of superpoints by around 75% ($M^1 = 80 \longrightarrow M^T = 20$), the corresponding ratio for GrowSP-ForMS is only around 20%. However, through extensive testing we established that proceeding with the merging until a smaller number of superpoints consistently reduced model accuracy. As such, we opted for a significantly shorter and less aggressive growth stage during training.

In an effort to save computational resources during training, the geometric point cloud descriptors and surface normals were precomputed for the input data. Furthermore, the features were computed prior to dividing the test site point clouds into cylinders of radius $r_c = 4.2$ m as described in Section 3.4, in order to avoid the loss of geometric information that comes with dividing the point cloud into smaller segments.

Finally, since the unsupervised GrowSP-ForMS does not require ground truth labels for training, we augmented the labeled training split with the three available unlabeled test plots in an effort to improve model generalization.

4.4.3 Traditional Algorithms

A significant limitation many of the traditional geometry or radiometry-based leaf–wood separation algorithms have is that they require the input point clouds to represent individual trees. This is also the case for GBS, one of our two unsupervised baselines. As such, the training and test data sets were altered slightly, since the cylinders of radius $r_c = 4.2$ m generally contain both multiple trees and understory vegetation.

Furthermore, the cylinders often contain partial trees, as the individual trees are frequently located at the intersection of multiple cylinders. To form the train-test split for individual trees, each tree was simply assigned to the training or test set based on the cylinder that contained the majority of its points while points that were not part of any individual tree were simply discarded.

Although LeWoS was originally tested on individual tropical trees, it is noted that the method is fully applicable to plot-wide data of various forest environments and tests have shown equal effectiveness on such data [165]. Nevertheless, we chose to evaluate LeWoS for both individual trees and the full cylinders in an effort to provide a fair assessment of its effectiveness.

In addition to requiring slightly different input data format in comparison to the DL models, both LeWoS and GBS have parameters that should be adjusted for the data at hand in order to achieve the best possible results. Below we detail how the parameter optimization was conducted for each of the two algorithms.

LeWoS: Algorithms designed for unsupervised leaf–wood separation often require the user to specify multiple parameter values (see e.g. [166, 188, 189]). By contrast, LeWoS only requires one user defined parameter, namely the feature difference threshold $N_{\text{z\_thres}}$, which is quite a significant advantage. Furthermore, the parameter was shown to be rather insensitive, that is, the performance of LeWoS is unaffected by small changes in $N_{\text{z\_thres}}$ [165].

Wang et al. [165] noted that generally $N_{\text{z\_thres}}$ is within the range [0.1,0.2] and recommended the value 0.15. However, since the value of $N_{\text{z\_thres}}$ was optimized using TLS data in contrast to our ALS data, we opted for testing all values $N_{\text{z\_thres}} \in \{0.1, \ldots + 0.025 \ldots, 0.4\}$, where $+d$ indicates several elements with a repeated difference of d. Subsequently, the $N_{\text{z\_thres}}$ that generated the highest mIoU on the training set was chosen. Since LeWoS was evaluated for both the cylindrical point clouds and individual trees, the parameter optimization was conducted separately for the two cases.

GBS: In order to achieve optimal results on our data set, we defined a set of reasonable parameter values based on the suggestions in the article where GBS was proposed [166] and subsequently performed a grid search on the training set. The parameter combination that yielded the highest mIoU on the training split was then used for comparing performance on the test set.

The GBS algorithm has four essential adjustable parameters, precursor direction difference threshold θ, interval length set I and linearity and cylindricity thresholds L and E [166] (see Equation 67). The authors suggest using the set $I = \{0.1, 0.2, 0.3, 0.5, 1\}$ for trees with a height in the range [2 m, 60 m], which we opt to use since all individual trees in our data set are within the range. Similarly, the authors propose using $\theta \in [0.1\pi, 0.2\pi]$ in cases where the woody components are surrounded by leaves or when leave groups exhibit linear shapes, while $\theta \in [0.2\pi, 0.3\pi]$ is recommended if leaves are distributed at the ends of branches [166]. As such, we defined the set of reasonable parameter values as $\theta \in \{0.1\pi, 0.2\pi, 0.3\pi\}$.

In the original article, the values of L and E were empirically optimized to 0.9

and 0.2 respectively [166]. Since the density of our data differs quite significantly from the TLS data used for determining the parameter values, we chose to inspect the effects of altering the parameter values slightly. The sets of reasonable values were chosen as $L \in \{0.85, 0.9, 0.95\}$ and $E \in \{0.15, 0.2, 0.25\}$. In total the set of reasonable parameter values is comprised of 27 distinct parameter combinations, which, while laborious, is still a feasible number to process.

5 Results

In this section we report the results of our experiments on the multispectral forest environment data set detailed in Section 3. Section 5.1 discusses the performance of our proposed graph-based superpoint constructor and compares its performance to the VCCS-based constructor used by standard GrowSP. Similarly, the performance of GrowSP-ForMS in the task of leaf–wood separation is discussed in Section 5.2. The results feature an extensive performance comparison between existing methods and GrowSP-ForMS as well as an ablation study that assesses the effects of both the proposed changes to GrowSP and the usage of multispectral LiDAR.

5.1 Graph-Based Superpoint Constructor

Table 4 shows a comparison of our proposed graph-based superpoint constructor and the VCCS-based superpoint constructor used by the generic GrowSP model. The accuracy metrics have been computed using the majority vote strategy described in Section 4.1.2. In addition to the overall accuracy, mean accuracy and mean intersection over union, we list the intersection over union for the foliage and wood classes separately. A visual example of the superpoints constructed by each method is shown in Figure 22. VCCS was tested both with the suggested voxel size of 0.05 meters and with an increased voxel size of 0.1 meters. It should be noted that although the original GrowSP article states the voxel size used for superpoint construction is 0.5 m, the official source code reveals the size is in fact 0.05 m.

Besides the accuracy of the initial superpoints, the number of superpoints per input point cloud is another crucial metric for measuring superpoint constructor performance. Since the semantic primitive clustering module of GrowSP clusters all superpoint level representations, a lower number of superpoints is always preferable

Table 4: Comparison of the default VCCS-based superpoint constructor used in the generic GrowSP model and our proposed graph-based constructor. Accuracy metrics were computed by assigning each point the majority ground truth label within the respective superpoint. $\overline{M^0}$ denotes the mean number of superpoints across all input point clouds. For the VCCS-based superpoint constructors, the number in parentheses denotes the voxel size in meters.

Constructor	Data	$\overline{M^0}$	OA (%)	mAcc (%)	mIoU (%)	Foliage (%)	Wood (%)
VCCS (0.05)	Training	18 641	98.6	96.2	85.4	98.5	72.2
	Test	22 299	98.6	95.7	86.5	98.6	74.5
	All	19 361	98.6	96.1	85.6	98.5	72.8
VCCS (0.1)	Training	1441	97.5	91.3	76.0	97.5	54.5
	Test	1182	97.3	91.5	75.0	97.3	52.6
	All	1390	97.5	91.3	75.7	97.4	54.0
Graph-based	Training	1347	98.1	92.4	81.3	98.0	64.6
	Test	1438	98.0	92.1	81.1	97.9	64.3
	All	1365	98.0	92.4	81.3	98.0	64.5

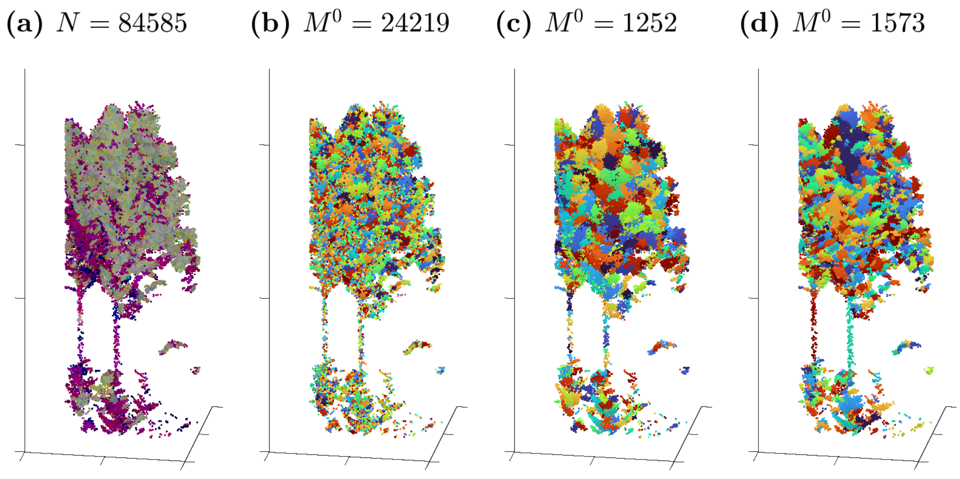


Figure 22: Qualitative comparison of the different superpoint constructors for a sample cylinder from plot #2 of our multispectral data set. (a) Input point cloud. N denotes the number of points in the cloud. (b) Superpoints from the VCCS-based superpoint constructor when voxel size is 0.05 meters. (c) Superpoints from the VCCS-based superpoint constructor when voxel size is 0.1 meters. (d) Superpoints from our proposed graph-based superpoint constructor. For figures (b)–(d), M^0 denotes the number of superpoints in the result.

considering the computational complexity of the clustering. In fact, increasing the number of superpoints beyond a certain threshold will essentially make training GrowSP infeasible, as simply the semantic primitive clustering would take multiple hours or even days. For reference, with around 1,400 initial superpoints per input point cloud the clustering process takes around 10 minutes on our hardware.

With the original voxel size of 0.05 meters, the VCCS-based superpoint constructor yields almost 20,000 superpoints per input point cloud on average, which effectively makes training GrowSP infeasible. Increasing the voxel size to 0.1 meters brings the average number of superpoints per input cloud down to around 1,400, which, based on our experiments, is around the upper bound of reasonable number of initial superpoints. However, as can bee seen from Table 4, this also results in a significant decrease in the mIoU (–9.9 percentage points (pp)) and IoU of wood points (–18.8 pp). As previously mentioned, a reasonably high accuracy is requirement for achieving great performance with GrowSP, since discerning wood points from foliage becomes increasingly difficult if a significant number of wood points are within superpoints that are comprised of mostly foliage points.

The graph-based superpoint constructor produces an average of around 1,400 superpoints per input cloud, which is identical to the corresponding metric of the VCCS-based constructor, while also providing a significant increase in mIoU (+5.6 pp) and IoU of wood points (+10.5 pp) across the entire data set. These results suggest that the proposed graph-based superpoint constructor is more suitable for processing forest data than the VCCS-based constructor. The effects the more accurate initial superpoints have on the performance of GrowSP are assessed in Section 5.2.2.

5.2 Leaf–wood Separation

5.2.1 Model Performance and Comparison

Quantitative performance metrics of fully-supervised and unsupervised baselines as well as the best performing configuration of GrowSP-ForMS for the training and test split of our MS data set are listed in Tables 5 and 6 respectively. We note that the performance metrics of models that required individual trees as input (denoted by *) are not strictly comparable to the models that accepted entire forest scenes as input. This is due to the fact that the training and test splits differ slightly between the two (see Section 4.4.3). For the class-wise segmentation accuracies, the interested reader is referred to Appendix B.

For all fully-supervised deep learning models, the reported performance was achieved by training the corresponding model on the training split using the configuration described in Section 4.4.1. Similarly, GrowSP-ForMS was trained using the pipeline presented in Section 4.4.2. However, instead of only training GrowSP-ForMS with one predefined set of input features, we experimented with a wide variety of feature combinations and chose the best one as our final model. Perhaps interestingly, by far the best performing setup only utilized reflectance from scanner 1 as the input feature but all three channels for the semantic primitive clustering and superpoint merging. Finally, the reported accuracies for the unsupervised traditional algorithms are from the best performing hyperparameters based on a grid search across the training split.

For LeWoS, the variation in segmentation accuracy between the tested parameter values was minor, as the mIoU was within the range 50±2% for all configurations. This serves as further empirical evidence of LeWoS being quite insensitive to the value of $N_{\text{z\_thres}}$, as asserted in the original paper [165]. The best performance in terms

Table 5: Quantitative results of different approaches on the **training split** of our proposed multispectral data set. The best results in each category have been highlighted. (*) Results when the inputs are individual trees. These accuracy metrics are not strictly comparable to others due to the different data format.

Model	OA (%)	mAcc (%)	mIoU (%)	Foliage (%)	Wood (%)
Supervised methods					
PointNet [15]	96.0	79.6	67.2	95.9	38.4
RandLA-Net [138]	97.4	89.1	75.1	97.3	53.0
ResNet16FPN [79]	**97.9**	**89.7**	**80.4**	**97.8**	**63.0**
Unsupervised methods					
LeWoS [165]	93.8	60.6	51.8	93.8	9.8
LeWoS* [165]	94.6	68.7	53.4	94.6	12.1
GBS* [166]	94.0	67.9	57.9	93.9	22.0
GrowSP-ForMS (Ours)	**96.2**	**81.2**	**67.4**	**96.1**	**38.8**

of training set mIoU was achieved with $N_{\text{z\_thres}} = 0.2$ when inputs were entire forest scenes and $N_{\text{z\_thres}} = 0.35$ when individual trees were provided as input. On the other hand, the segmentation performance of GBS varied quite significantly between the tested parameter combinations. The highest accuracy was achieved when precursor direction difference threshold was set to $\theta = 0.1\pi$, and the linearity and cylindricity thresholds were set to $L = 0.9$ and $E = 0.15$ respectively.

Based on the quantitative accuracy metrics, LeWoS fails almost completely at leaf–wood separation both on the training and test splits, with the mIoU around 51% for both configurations. Although LeWoS performed slightly better when individual trees were used as input, the accuracy improvement was essentially negligible, especially considering that the subset of points that do not belong to any individual tree are not accounted for in the accuracy computations. GBS performs notably better, reaching an mIoU of 60.2% on the test set. Nevertheless, GrowSP-ForMS outperforms GBS by a large margin on the test split, both in terms of mAcc (+11.8 pp) and mIoU (+8.4 pp). Consequently, the performance of GrowSP-ForMS is state-of-the-art in the task of unsupervised leaf–wood separation of multispectral ALS point clouds.

In case of the supervised baselines, the models ranked as expected, with more recent architectures performing better. The best model, ResNet16FPN achieved test set mAcc and mIoU of 89.4% and 78.4% respectively, being clearly more accurate than older PointNet (–14,1 pp and –15,2 pp in mAcc and mIoU respectively). Notably, all supervised baselines outperformed the best unsupervised baseline GBS in terms of all quantitative performance metrics.

When compared to the fully-supervised baseline architectures, the performance of GrowSP-ForMS is comparable to PointNet. On the other hand GrowSP-ForMS is severely outmatched by both RandLA-Net and ResNet16FPN in terms of test split mIoU (–9.8 pp and –5.8 pp respectively). Such performance differences however were expected, as similar metrics were reported on multiple benchmark data sets in the

Table 6: Quantitative results of different approaches on the **test split** of our proposed multispectral data set. The best results in each category have been highlighted. (*) Results when the inputs are individual trees. These accuracy metrics are not strictly comparable to others due to the different data format.

Model	OA (%)	mAcc (%)	mIoU (%)	Foliage (%)	Wood (%)
Supervised methods					
PointNet [15]	95.2	75.3	63.2	95.1	31.3
RandLA-Net [138]	97.2	89.3	74.4	97.1	51.7
ResNet16FPN [79]	**97.6**	**89.4**	**78.4**	**97.5**	**59.2**
Unsupervised methods					
LeWoS [165]	93.3	58.3	50.7	93.3	8.0
LeWoS* [165]	93.7	63.4	51.7	93.7	9.7
GBS* [166]	94.4	72.4	60.2	94.3	26.1
GrowSP-ForMS (Ours)	**96.4**	**84.2**	**68.6**	**96.3**	**40.8**

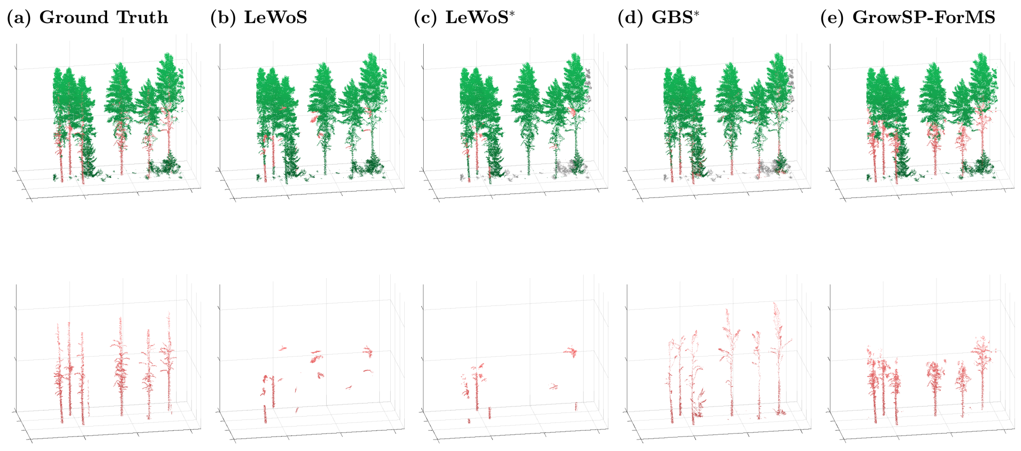


Figure 23: Qualitative results of the unsupervised baseline methods and GrowSP-ForMS on a subsection of the **test split** of plot #1 in our multispectral data set. The top row shows the full segmentation results and the row below contains only points classified as wood. (*) Results when the inputs are individual trees. Points colored grey are not part of any individual tree and were thus not classified by the algorithm.

original GrowSP paper [10].

Figures 23 and 24 show a visual comparison of segmentation results on a subsection of the test split of plot #1. Equivalent visual comparisons for a subsection of plot #2 are shown in Figures 25 and 26. A visual inspection of the results seemingly confirms what the quantitative performance metrics suggested: LeWoS fails quite badly for our data set, as most points are classified foliage. It appears that only highly linear sections of the trunk and branches with minimal occlusions are successfully detected. Conversely, while the segmentation results of GBS appear qualitatively excellent at first glance, the quantitative accuracy metrics do not reflect this at all. A closer inspection and comparison the ground truth labels reveal this is because GBS has a tendency of detecting non-existent branches within the tree crown, while simultaneously often failing to detect clearly visible branches in lower sections of the tree. Furthermore, GBS appears to detect only the very center section of the trunk and usually misclassifies around half of the trunk as foliage. We conjecture that these are consequences of the varying point density, which GBS is heavily reliant on: lower sections with a low point density cause issues even if the branch is clearly visible, while some dense continuous sections within the tree crown are mistaken for branches.

In comparison to the unsupervised baselines, the segmentation results of GrowSP-ForMS are visually more consistent. Majority of the trunk as well as lower branches are usually correctly detected, although especially within branches the predictions are often overly smooth in the sense that they are quite noisy since many nearby foliage points are erroneously classified as wood. The other main source of error appears to be upper sections of the trunk, which GrowSP-ForMS consistently fails to detect. However, considering that even the best supervised baselines RandLA-Net and ResNet16FPN seemingly struggle with the same sections, we conclude that the upper

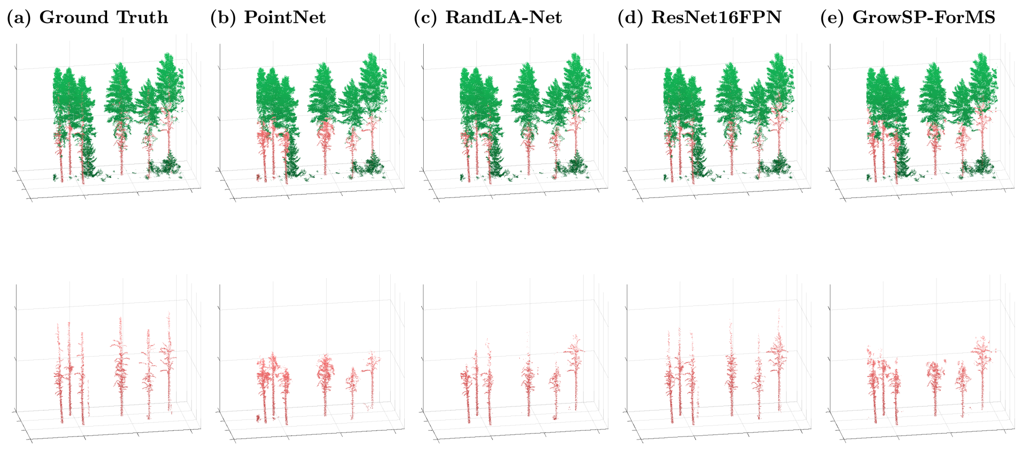


Figure 24: Qualitative results of the supervised baseline methods and GrowSP-ForMS on a subsection of the **test split** of plot #1 in our multispectral data set. The top row shows the full segmentation results and the row below contains only points classified as wood.

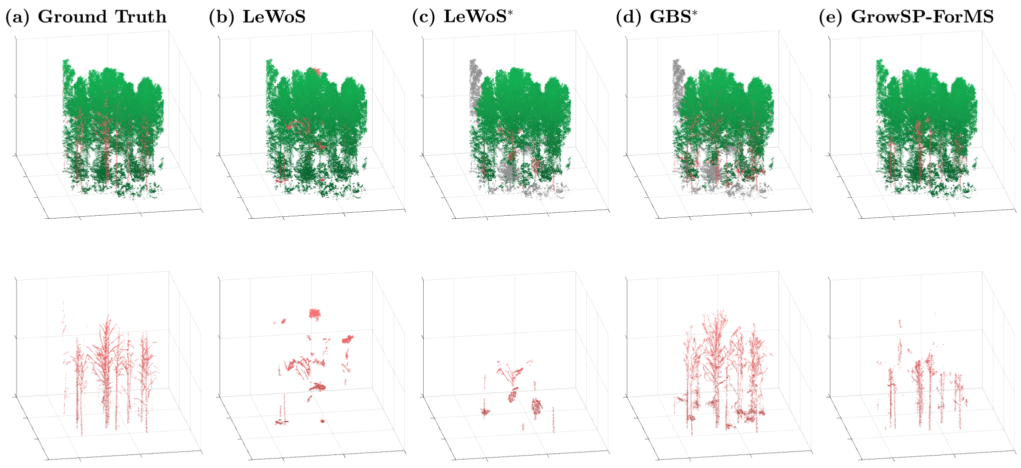


Figure 25: Qualitative results of the unsupervised baseline methods and GrowSP-ForMS on a subsection of the **test split** of plot #2 in our multispectral data set. The top row shows the full segmentation results and the row below contains only points classified as wood. (*) Results when the inputs are individual trees. Points colored grey are not part of any individual tree and were thus not classified by the algorithm.

trunk is overall rather difficult to classify correctly.

Another notable observation when comparing the results between Figures 23 and 25 is that GrowSP-ForMS performs notably worse on test plot #2. There are at least two possible explanations for this: firstly, test plot #2 is denser and therefore more challenging to segment. Secondly, the trees on plot #2 are mostly birches and spruce, while plot #1 is predominantly pine, which may be easier to segment semantically.

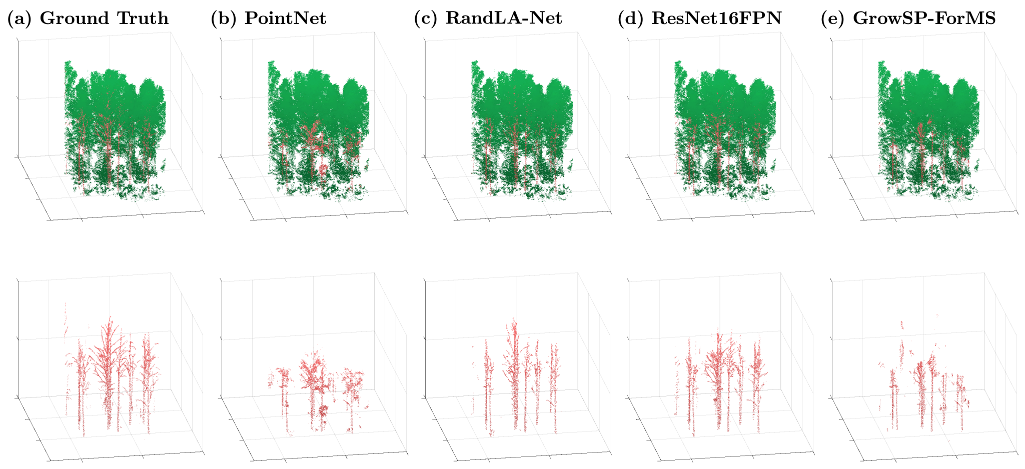


Figure 26: Qualitative results of the supervised baseline methods and GrowSP-ForMS on a subsection of the **test split** of plot #2 in our multispectral data set. The top row shows the full segmentation results and the row below contains only points classified as wood.

Shifting the visual comparison to the supervised baselines, the results of PointNet are visually strikingly similar to GrowSP-ForMS, espcially on plot #1. The segmentation results of PointNet are also somewhat noisy and upper sections of the trunk are seldom successfully detected. On the other hand, RandLA-Net and ResNet16FPN outperform GrowSP-ForMS quite clearly, as was also the case in the quantitative performance. ResNet16FPN yields visually slightly better results than RandLA-Net, mainly thanks to detecting a few more branches and some of the upper regions of the trunks. Overall both models yield comparable results which appear to be quite close to the ground truth.

Additional visual comparisons of semantic segmentation results for previously unseen unlabeled data that was not part of the training or test set can be found in Appendix A.1.

5.2.2 Ablation Study

In order to accurately assess the usefulness of our proposed changes to GrowSP and the benefits of using multispectral data for unsupervised leaf–wood separation, we conducted two separate ablation studies. In the first study, starting with the generic GrowSP architecture, we iteratively augmented the model with our improvements one by one, training and evaluating again each time. Similarly, in the second study we trained the full GrowSP-ForMS model for each intensity channel separately, as well as all two-channel combinations. The performance metrics of the ablated models on the test split of our data set are listed in Tables 7 and 8 for ablation studies 1 and 2 respectively.

For the ablation study on proposed modifications to GrowSP, we utilized the input feature combination that yielded the best segmentation performance for the full model.

Table 7: Ablation study on our proposed modifications to the original GrowSP architecture. We note that in experiments where geometric features were not utilized, 10-dimensional point feature histograms were used instead. The reported accuracies were achieved on the **test split** of our multispectral data set. The overall highest accuracy metrics have been highlighted.

Oversegmentation of Predicted Labels	New Superpoints	Geometric Features	Decaying Clustering Weights	oAcc (%)	mAcc (%)	mIoU (%)	Foliage (%)	Wood (%)
				70.6	51.7	38.3	69.9	6.8
✓				91.8	58.6	52.0	91.7	12.3
✓	✓			95.7	82.4	60.5	95.6	25.3
✓	✓	✓		93.1	68.8	62.9	92.9	32.9
✓	✓		✓	88.6	55.2	49.4	88.5	10.2
✓	✓	✓	✓	**96.4**	**84.2**	**68.6**	**96.3**	**40.8**

Table 8: Ablation study on the effects of using mono- and multispectral data. The reported accuracies were achieved on the **test split** of our multispectral data set. The overall highest accuracy metrics have been highlighted. (*) Verticality threshold $V_{\text{min}} = 0.55$ used for identifying the wood classes instead of the linearity threshold.

Scanner 1	Scanner 2	Scanner 3	oAcc (%)	mAcc (%)	mIoU (%)	Foliage (%)	Wood (%)
✓			94.7	72.4	**62.0**	94.6	**29.3**
	✓		78.3*	55.3*	44.9*	77.6*	12.1*
		✓	91.6*	63.2*	56.6*	91.4*	21.8*
✓	✓		93.4	66.6	58.5	93.2	23.8
✓		✓	93.5	68.4	61.1	93.3	28.9
	✓	✓	89.9*	55.6*	49.9*	89.8*	10.0*
✓	✓	✓	**95.4**	**77.9**	61.4	**95.4**	27.5

That is, only reflectance from scanner 1 was used as input feature, while all channels were exploited in semantic primitive clustering and superpoint merging. Conversely, when assessing the benefits of multispectral data, the listed channels were used both as input features and for clustering and merging of superpoints. In ablation study 1, experiments without the new initial superpoints utilized superpoints generated by the VCCS-based constructor with voxel size 0.1 m. Similarly, experiments without geometric features used the 10-dimensional PFHs following the original GrowSP architecture.

The first ablation study emperically proves that all of our proposed modifications to GrowSP are beneficial in the context of leaf–wood separation from MS ALS point clouds. That is, each change individually improves model performance in terms of test set mIoU in comparison to the baseline. The most significant improvements in segmentation accuracy were provided by oversegmentation of predicted labels (+13.7 pp in mIoU), which is admittedly more of a postprocessing heuristic, and the new graph-based superpoint constructor (+8.5 pp in mIoU). Interestingly, the decaying clustering weights cause a significant decrease in performance when PFHs are used (–13.5 pp in mIoU), but work exceedingly well when using geometric features (+5.7 pp in mIoU).

The results of ablation study 2 were not as straightforward. The two best performing models were the one that only utilizes scanner 1 and the model that utilizes all three scanners, with the former achieving a slightly higher mIoU and the latter a higher oAcc and mAcc. Although it might seem that these results suggest there is no notable difference between mono- and multispectral data, this is not the case in practice, since our best performing GrowSP-ForMS configuration utilizes multispectral information for semantic primitive clustering and superpoint merging and outperforms the best monospectral model by a considerable margin (+6.6 pp in test set mIoU).

Overall, the performance of the two best ablated models was essentially comparable. This is especially interesting considering that none of the one- or two-channel setups managed to outperform the scanner 1 single-channel model in terms of any performance metrics. The results suggest that the 1550 nm wavelength is by far the most useful for leaf–wood separation, although the 905 and 532 nm wavelengths are also demonstrably beneficial, especially for improving the accuracy of the foliage class.

In some experiments of the second ablation study where the wood class was not separated adequately, the mean linearity was not high enough to distinguish the wood class based on the linearity threshold $L_{\text{min}} = 0.55$. In such cases (denoted by * in Table 8), the wood classes were detected based on mean verticality instead. The threshold value for mean verticality was set to $V_{\text{min}} = 0.55$. This simultaneously highlights a shortcoming and an advantage of the oversegmentation heuristic. If segmentation accuracy is poor, the wood class may not be automatically detected. However, changing the either threshold or metric such that the wood detection functions correctly is fast and does not require training the model again. Nevertheless, manual calibration is needed from the user in such cases, which is obviously a disadvantage.

A visual comparison of the segmentation results for ablated models can be found in Appendix A.2.

6 Discussion

Through achieving state-of-the-art performance on our data set, we demonstrated that unsupervised deep learning is a viable approach for leaf–wood separation from multispectral ALS point clouds. In fact, the accuracy of GrowSP-ForMS was comparable to the slightly older fully-supervised semantic segmentation model PointNet even though unsupervised DL has previously not been applied to leaf–wood separation. Consequently, considering that the size of the data set was also fairly limited, we believe future work will improve on the baseline set by us.

In addition to achieving state-of-the-art performance, GrowSP-ForMS is relatively lightweight in terms of computational requirements. On a GPU with just 8 GB of VRAM, a circular forest plot with a 55 m diameter can be processed in around 10 seconds when divided into cylinders with a radius of 4.2 m. This is comparable to the tested unsupervised algorithms and significantly faster than PointNet.

Our results indicate that multispectral LiDAR data can improve the accuracy of unsupervised leaf–wood separation. However, utilizing the multispectral information effectively was not a matter of simply providing the reflectance information of all channels as input features for the feature extractor. Rather, it demanded careful experimentation with different input feature combinations and training setups. Somewhat counter-intuitively, when reflectance features from all three channels were used as input, the performance of the model was comparable to a model that only utilizes reflectance information from one scanner operating at a wavelength of 1550 nm. We conjecture that while reflectance information in MS point clouds is beneficial for semantic segmentation tasks, utilizing the data in a manner that improves model performance is not always as straightforward as e.g. with RGB color, at least within our framework. While comparing leaf–wood separation accuracy of monospectral and multispectral data under a fully-supervised setting was outside the scope of this thesis, it would be interesting to investigate whether a similar phenomenon occurs. At least in the context of land-cover classification using MS point clouds, Zhang et al. [234] did not observe such behavior.

The initial superpoint constructor is one of the most crucial parts of GrowSP, since superpoint accuracy is equivalent to a soft upper bound for general model accuracy. Although predicted labels are not assigned on a superpoint level at test time, the backbone will generally learn similar features for all points within the same superpoint. As such, it is rare to correctly classify a wood point that is within a superpoint comprised mainly foliage points or vice versa. Our ablation study empirically demonstrated that the improved superpoint constructor was one of the most beneficial individual changes to GrowSP in terms of test set mIoU, which further highlights the importance of accurate initial superpoints.

It is crucial to note that unsupervised leaf–wood separation algorithms are generally designed for TLS data. As such, unsatisfactory performance on ALS data is to be expected due to heavy reliance on geometric point cloud characteristics that are significantly weaker in sparser data. In fact, many works have noted that the accuracy of unsupervised algorithms quickly begins to deteriorate when the point density decreases. For example, Tian and Li [166] observed a significantly lower segmentation accuracy for both LeWoS and GBS when a subsampled version of the same point cloud was used. Vicari et al. [188] reported a similar phenomenon for their leaf–wood separation algorithm. Consequently, while GrowSP-ForMS achieves state-of-the-art performance in unsupervised leaf–wood separation for ALS data, no conclusions should be drawn about accuracy on point clouds captured with other types of LiDAR systems. Nevertheless, our results demonstrate that when it comes to ALS data, unsupervised DL is the more viable option for leaf–wood separation in comparison to geometry-based algorithms.

Based on the qualitative performance comparison GrowSP-ForMS performs distinctly worse on test plot #2, which is mainly comprised of deciduous birches. This may be due to the inherently more complex structure of deciduous trees, which results in them being more difficult to segment accurately, as observed in Xiang et al. [6] where notably lower panoptic segmentation accuracy was reported for plots with more complex branch structures. Another possible explanation is the imbalanced training data set. As previously mentioned, since GrowSP-ForMS does not require ground truth labels during training, the training split was augmented with three unlabeled plots in an attempt to improve model generalization. However, due to lack of data from forest plots dominated by birch or other deciduous species, a large majority of trees within the extra training data were either pine or spruce. As such, GrowSP-ForMS may have learned to distinguish the wooden components within pine and spruce more effectively due to the imbalance.

Our study has several noteworthy limitations, which are further discussed in Section 6.1. Furthermore, while the segmentation accuracy of GrowSP-ForMS is state-of-the-art in unsupervised leaf–wood separation, the performance is far below modern fully-supervised architectures. As such, even if not requiring any manually annotated data is a considerable advantage, further performance improvements are required in order to reach an accuracy that is adequate for operative purposes. Thankfully, since this study is the first published leaf–wood separation model based on unsupervised DL, there are multiple promising directions for future research which we summarize briefly in Section 6.2.

6.1 Limitations

The limited amount of computational resources, specifically GPU VRAM, set certain restrictions on training the DL models. Firstly, test plots had to be divided into smaller cylinders in order to fit the data into GPU memory. This may have an impact on performance, since e.g. Xiang et al. [217] observed that a larger context within forest scenes improved the accuracy of panoptic segmentation. Secondly, minibatches had to be restricted to relatively few data points due to memory limitations. Similarly, lack of GPU VRAM prevented testing certain hyperparameter combinations and e.g. the number of semantic primitives and extracted features in GrowSP had to be halved.

In comparison to benchmark data sets such as FOR-Instance [44], our proposed multispectral data set is relatively limited in size for deep learning purposes. As such, the risk of overfitting is relatively high and the reported quantitative performance metrics on the test split may not reflect model generalization adequately. Furthermore, due to the limited amount of manually annotated data available, we chose to not have a separate validation split. Consequently, most hyperparameter values were chosen either based on the literature or empirically and are not necessarily perfectly optimal for our use case.

It should be acknowledged that the procedure used for training the fully-supervised deep learning baselines left some things to be desired. Firstly, the training set was relatively limited in size, which is known to hinder the process of training deep learning models and can increase the risk of overfitting. Secondly, due to very limited computational resources, the input data format, batch size and number of training epochs was not necessarily optimal for every model. However, this was only a limiting factor for PointNet and RandLA-Net, as the memory efficiency of ResNet16FPN allowed choosing the hyperparameters relatively freely. For PointNet, the input cylinders had to be divided into smaller sections due to memory limitations. Consequently, a certain amount of local and global geometric information was lost, which undoubtedly hindered the model's performance. Similarly, the performance of RandLA-Net would most likely have benefited from a larger batch size and a higher number of neighbors in training.

Finally, the lack of a validation set made it slightly more difficult to accurately asses when training should be concluded. As such, the accuracy metrics on the test set are lower than the corresponding values on the training set for all three supervised baselines, which is generally a sign that the models may be overfitting slightly. However, considering that the performance on the test split was nevertheless quite close to that achieved on the training split, we conclude that any possible overfitting is not very severe. This is further supported by visual assessment of predicted labels for unlabeled data that was not used in training, where the quality of predictions remains qualitatively great and similar to that observed on the training and test sets. If the models were overfitting egregiously, we would expect the prediction quality to decrease significantly on such data, which is not the case.

Considering the factors above, we stress that the presented accuracy metrics should not be considered the best possible performance that can be attained using the respective models. Rather, the performance serves as a baseline of what is achievable with a relatively simple training arrangement and a limited amount of training data and computational resources. For this study such metrics are sufficient, as the objective of the supervised baselines was to asses the performance of simple semantic segmentation DL models in comparison to the first ever unsupervised DL based leaf–wood separation model, which was expected to perform far below the state-of-the-art supervised models.

Finally, it is worth noting that a sparse convolution-based supervised panoptic segmentation network presented in a recent study correctly detected 55.3% and 68.4% of stem and wooden branch points respectively on the test split of the FOR-Instance benchmark data set [6]. This is notable for the following reasons:

1. Our SCNN supervised baseline ResNet16FPN succesfully detected 80.5% of all wood points (see Appendix B), which is comparable to the metrics presented above achieved by the state-of-the-art model of [6].

2. FOR-Instance is very similar to our data set, as both are comprised of ALS forest

point clouds with comparable point densities. Furthermore, the data in FOR-Instance was captured using Riegl VUX-1UAV and MiniVUX-1UAV scanners, which are relatively similar to the scanners included in HeliALS-TW. The only notable differences between FOR-Instance and our data are that the former is monospectral and contains a wider variety of distinct forest environments. As such, we would expect a well trained SCNN to achieve a similar or slightly better performance on our data set in comparison to [6], which ResNet16FPN does.

3. Earlier studies comparing the performance of PointNet, RandLA-Net and SCNNs on popular semantic segmentation benchmarks have observed performance differences between the three models that are comparable those observed by us (see e.g. [10, 80, 138]).

Based on these observations, we conclude that although the presented performance of the supervised baselines is not necessarily optimal, it is most likely relatively close.

Besides being limited in size, our proposed data set only contains scans from boreal forests. As such, the reported results do not reflect on the performance of GrowSP-ForMS across forests in general since e.g. temperate and tropical forests are structurally very different. Further, since the training and test splits are from the same forest plots, albeit different areas, the test set is not necessarily optimal for measuring the generalization ability of the models. Nevertheless, we attempted to select the test data set such that it has representations of all the different types of forest environments present in the data.

6.2 Possible Directions for Future Research

While the results of our study are promising, the difference in leaf–wood separation accuracy between GrowSP-ForMS and state-of-the-art supervised methods remains large. Fortunately, since unsupervised DL for semantic segmentation of point clouds remains relatively unexplored, especially in the context of forest data, there are numerous prospects for future research that could help bridge the gap. In this section we suggest some possible directions future research could take.

Firstly, our experiments should be reconducted on a system with significantly more computational power, specifically a GPU with more VRAM or even multiple GPUs. This would alleviate many of the limitations discussed in Section 6.1 by enabling e.g. larger batch sizes during training and utilizing the original number of neural features and semantic primitives in GrowSP.

Secondly, more MS forest point clouds should be manually labelled for a validation set to make the training process more reliable. Even if additional labeled data is not available, significantly increasing the amount of unlabeled training data could improve model performance and generalization, as is often the case with unsupervised DL. In fact, in our case more MS point clouds depicting circular forest test plots would have been available, but hardware limitations prevented us from utilizing them during training. It should be noted that since increasing the amount of training data may result in prohibitively long computation times during semantic primitive clustering, adopting a more efficient clustering algorithm such as minibatch k-means [235] could be necessary.

Aside from hardware and data related improvements, future research should consider utilizing other neural networks as the feature extractor. We opted to use the same backbone network as the generic GrowSP, but there is no guarantee that ResNet16FPN is the optimal choice for forest data. A more recent transformer-based architecture such as PTv3 [67] may provide more descriptive features and by extension improve model performance. Similarly, different optimizers, learning rate schedulers and loss functions should also be experimented with. As an example, due to the considerable class imbalance in forest data, a contrastive loss function such as focal loss could improve the accuracy of leaf–wood separation. The addition of geometric input features for the feature extractor should also be considered, since multiple previous studies have found them useful for DL-based leaf–wood separation [6, 48, 208]. It is worth noting that many of the aforementioned changes are dependent on more computational resources being available.

As demonstrated in Section 5.2.2, improving the accuracy of superpoints provided one of the most significant increases in segmentation performance. As such, we believe that further improving the superpoint constructor should be one of the main focuses of future work. Increasing the accuracy of the initial superpoints and decreasing their number would both be beneficial. The former from the perspective of model accuracy and the latter when it comes to preserving computational resources. Furthermore, decreasing the number of superpoints inherently results in them being larger, which could also help improve model accuracy as larger superpoints may contain more semantic information. For example, a more deliberate strategy for merging the initial graph-based superpoints should be researched. Another possibility is classifying the point clouds using some existing unsupervised leaf–wood separation algorithm or DL model trained on another data set and subsequently utilizing the predictions for constructing initial superpoints. The latter would be possible, as pretrained weights for various forest semantic segmentation models, such as ForAINet [6], have been made publicly available[6].

On the other hand, one of the significant shortcomings of GrowSP-ForMS is that the superpoint growing phase has to be halted quite early as model performance quickly starts to deteriorate if too many superpoints are merged. Some other clustering algorithm, possibly one that considers connectivity of the superpoints, could be more suitable and enable growing the superpoints further without decreasing model accuracy.

Finally, utilizing unsupervised contrastive pretraining as proposed by e.g. [82, 158] should also be investigated. The generic GrowSP architecture utilizes geometric features for semantic primitive clustering, since extracted neural features are effectively random during early stages of training [10]. The addition of a contrastive pretraining stage prior to training GrowSP-ForMS might eliminate the need for geometric point cloud features entirely, provided that the learned features are descriptive enough. This would help make the model more data-driven and less reliant on heuristic geometric

[6]https://github.com/bxiang233/ForAINet

descriptors.

In addition to the possible improvements to GrowSP-ForMS discussed above, we strongly recommend experimenting with different types of LiDAR forest data in future work. Although GrowSP-ForMS was specifically designed for semantic segmentation of multispectral forest point clouds, it can also be trained using monospectral data, albeit this does hinder its accuracy as evidenced by the ablation study in Section 5.2.2. Since multispectral LiDAR systems remain relatively rare to this day, efforts should be made to improve the performance of GrowSP-ForMS on monospectral point clouds. Simply optimizing the model hyperparameters with monospectral training data may yield some improvements. Such changes would also enable assessing the performance of our method on monospectral benchmark forest data sets, such as FOR-Instance [44].

Similarly, while GrowSP-ForMS achieves state-of-the-art performance on unsupervised semantic segmentation of ALS forest point clouds, the comparison to existing leaf–wood separation algorithms is not completely fair, considering that such methods are typically designed for TLS data which is both considerably denser and contains less occlusions. As such, in the future, GrowSP-ForMS should also be trained using TLS data. We conjecture that when it comes to TLS point clouds, the difference in performance between GrowSP-ForMS and existing geometric leaf–wood separation algorithms may be negligible and many may even outperform GrowSP-ForMS. However, due to the lack of existing MS TLS systems, the previously mentioned performance improvements for monospectral data are crucial.

7 Conclusions

In this thesis, we created a multispectral point cloud data set depicting a boreal forest environment with more than eight million individual points, each of which has been manually labeled into either wood or foliage. Captured with FGI's in-house developed aerial laser scanner system HeliALS-TW, our data set is the world's first MS LiDAR-based data set for semantic segmentation of forests.

Furthermore, we presented and implemented a modified version of the GrowSP architecture, GrowSP-ForMS, designed specifically for unsupervised semantic segmentation of forest data. Subsquently, we trained GrowSP-ForMS on our multispectral ALS forest data set and compared its performance to two unsupervised leaf–wood separation algorithms and three fully-supervised neural networks. Based on this comparison, we concluded that GrowSP-ForMS achieves state-of-the-art performance in unsupervised leaf–wood separation of multispectral ALS point clouds. Furthermore, while the two more recent supervised models outperformed GrowSP-ForMS by a considerable margin, its performance was comparable to the slightly older PointNet.

In order to assess the effectiveness of our proposed modifications to GrowSP, we performed an ablation study where the generic GrowSP model was iteratively augmented with each of our improvements one by one. The results indicated that all of our proposed changes improved model performance in terms of test set mIoU. However, somewhat counter-intuitively decaying clustering weights only increased segmentation accuracy when used in combination with the other modifications.

Finally, by comparing the performance of GrowSP-ForMS on mono- and multi-spectral versions of our data set, we demonstrated that using multispectral LiDAR improves leaf–wood separation accuracy in an unsupervised DL setting. However, since our best performing model only utilized multispectral information in certain parts of the segmentation pipeline, we concluded that utilizing MS information effectively requires carefully choosing the optimal combination of input features, at least within the GrowSP framework.

The results of this thesis conclusively answer all of the research questions. Firstly, by comparing the performance of GrowSP-ForMS to two conventional unsupervised leaf–wood separation algorithms, we demonstrated that utilizing unsupervised deep learning can alleviate the problems caused by reduced point density in ALS data. Secondly, GrowSP-ForMS was compared to three fully-supervised semantic segmentation architectures, which showed that its leaf–wood separation accuracy is on par with the older PointNet but far from more recent architectures. Lastly, multispectral data was shown to improve the semantic segmentation performance of our unsupervised DL-based leaf–wood separation model.

Although the performance of GrowSP-ForMS is not on par with the best fully-supervised architectures and further improvements are needed to reach accuracies required at operational level, it nevertheless demonstrates that unsupervised deep learning is a viable option for semantic segmentation of forest point clouds. The fact that our model performs comparably to an earlier fully-supervised architecture is very promising and there are various prospects for future research that could further improve segmentation accuracy.

References

[1] Di Wang. Unsupervised semantic and instance segmentation of forest point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 165:86–97, 2020. doi:10.1016/j.isprsjprs.2020.04.020.

[2] Zhouxin Xi, Chris Hopkinson, and Laura Chasmer. Filtering Stems and Branches from Terrestrial Laser Scanning Point Clouds Using Deep 3-D Fully Convolutional Networks. *Remote Sensing*, 10(8), 2018. doi:10.3390/rs10081215.

[3] Sean Krisanski, Mohammad Sadegh Taskhiri, Susana Gonzalez Aracil, David Herries, and Paul Turner. Sensor Agnostic Semantic Segmentation of Structurally Diverse and Complex Forest Point Clouds Using Deep Learning. *Remote Sensing*, 13(8), 2021. doi:10.3390/rs13081413.

[4] Risto Kaijaluoto, Antero Kukko, Aimad El Issaoui, Juha Hyyppä, and Harri Kaartinen. Semantic segmentation of point cloud data using raw laser scanner measurements and deep neural networks. *ISPRS Open Journal of Photogrammetry and Remote Sensing*, 3:100011, 2022. ISSN 2667-3932. doi:10.1016/j.ophoto.2021.100011.

[5] Maciej Wielgosz, Stefano Puliti, Phil Wilkes, and Rasmus Astrup. Point2Tree(P2T)—Framework for Parameter Tuning of Semantic and Instance Segmentation Used with Mobile Laser Scanning Data in Coniferous Forest. *Remote Sensing*, 15(15), 2023. doi:10.3390/rs15153737.

[6] Binbin Xiang, Maciej Wielgosz, Theodora Kontogianni, Torben Peters, Stefano Puliti, Rasmus Astrup, and Konrad Schindler. Automated forest inventory: Analysis of high-density airborne LiDAR point clouds with 3D deep learning. *Remote Sensing of Environment*, 305:114078, 2024. doi:10.1016/j.rse.2024.114078.

[7] Alireza Hamedianfar, Cheikh Mohamedou, Annika Kangas, and Jari Vauhkonen. Deep learning for forest inventory and planning: a critical review on the remote sensing approaches so far and prospects for further applications. *Forestry: An International Journal of Forest Research*, 95(4): 451–465, 02 2022. doi:10.1093/forestry/cpac002.

[8] Mingmei Cheng, Le Hui, Jin Xie, and Jian Yang. SSPC-Net: Semi-supervised Semantic 3D Point Cloud Segmentation Network. *The AAAI Conference on Artificial Intelligence*, 35(2): 1140–1147, 2021. doi:10.1609/aaai.v35i2.16200.

[9] Minghua Liu, Yin Zhou, Charles R. Qi, Boqing Gong, Hao Su, and Dragomir Anguelov. LESS: Label-Efficient Semantic Segmentation for LiDAR Point Clouds. In *Computer Vision – ECCV 2022*, pages 70–89. Springer Nature Switzerland, 2022. doi:10.1007/978-3-031-19842-7_5.

[10] Zihui Zhang, Bo Yang, Bing Wang, and Bo Li. GrowSP: Unsupervised Semantic Segmentation of 3D Point Clouds. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17619–17629, 2023. doi:10.1109/CVPR52729.2023.01690.

[11] Narges Takhtkeshha, Gottfried Mandlburger, Fabio Remondino, and Juha Hyyppä. Multispectral Light Detection and Ranging Technology and Applications: A Review. *Sensors*, 24(5), 2024. doi:10.3390/s24051669.

[12] Aada Hakula, Lassi Ruoppa, Matti Lehtomäki, Xiaowei Yu, Antero Kukko, Harri Kaartinen, Josef Taher, Leena Matikainen, Eric Hyyppä, Ville Luoma, Markus Holopainen, Ville Kankare, and Juha Hyyppä. Individual tree segmentation and species classification using high-density close-range multispectral laser scanning data. *ISPRS Open Journal of Photogrammetry and Remote Sensing*, 9:100039, 2023. doi:10.1016/j.ophoto.2023.100039.

[13] Dilong Li, Xin Shen, Haiyan Guan, Yongtao Yu, Hanyun Wang, Guo Zhang, Jonathan Li, and Deren Li. AGFP-Net: Attentive geometric feature pyramid network for land cover classification using airborne multispectral LiDAR data. *International Journal of Applied Earth Observation and Geoinformation*, 108:102723, 2022. doi:10.1016/j.jag.2022.102723.

[14] Zhan Li, Ewan Douglas, Alan Strahler, Crystal Schaaf, Xiaoyuan Yang, Zhuosen Wang, Tian Yao, Feng Zhao, Edward J. Saenz, Ian Paynter, Curtis E. Woodcock, Supriya Chakrabarti, Timothy Cook, Jason Martel, Glenn Howe, David L. B. Jupp, Darius S. Culvenor, Glenn J. Newnham, and Jenny L. Lovell. Separating leaves from trunks and branches with dual-wavelength terrestrial lidar scanning. In *2013 IEEE International Geoscience and Remote Sensing Symposium - IGARSS*, pages 3383–3386, 2013. doi:10.1109/IGARSS.2013.6723554.

[15] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, 2017. doi:10.1109/CVPR.2017.16.

[16] Shan Liu, Min Zhang, Pranav Kadam, and C.-C. Jay Kuo. *Explainable Machine Learning Methods for Point Cloud Analysis*, pages 87–140. Springer International Publishing, 2021. doi:10.1007/978-3-030-89180-0_4.

[17] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015. doi:10.1109/CVPR.2015.7298801.

[18] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al.. ShapeNet: An Information-Rich 3D Model Repository. *arXiv preprint arXiv:1512.03012*, 2015. doi:10.48550/arXiv.1512.03012.

[19] Franz Leberl, Arnold Irschara, Thomas Pock, Philipp Meixner, Michael Gruber, Set Scholz, and Alexander Wiechert. Point Clouds. *Photogrammetric Engineering & Remote Sensing*, 76(10): 1123–1134, 2010. doi:10.14358/PERS.76.10.1123.

[20] Jianwei Li, Wei Gao, Yihong Wu, Yangdong Liu, and Yanfei Shen. High-quality indoor scene 3D reconstruction with RGB-D cameras: A brief review. *Computational Visual Media*, 8(3): 369–393, 2022. doi:10.1007/s41095-021-0250-8.

[21] Peide Wang. Research on Comparison of LiDAR and Camera in Autonomous Driving. *Journal of Physics: Conference Series*, 2093(1):012032, 2021. doi:10.1088/1742-6596/2093/1/012032.

[22] Jie Shan and Charles K Toth. *Topographic Laser Ranging and Scanning: Principles and Processing, Second Edition*. CRC press, 2018.

[23] Ulla Wandinger. *Introduction to Lidar*, pages 1–18. Springer New York, 2005. doi:10.1007/0-387-25101-4_1.

[24] Daniel J. Lum, Samuel H. Knarr, and John C. Howell. Frequency-modulated continuous-wave LiDAR compressive depth-mapping. *Optics Express*, 26(12):15420–15435, 2018. doi:10.1364/OE.26.015420.

[25] Martin Pfennigbauer and Andreas Ullrich. Multi-Wavelength Airborne Laser Scanning. In *The International Lidar Mapping Forum, ILMF, New Orleans*, 2011.

[26] Keith Williams, Michael J. Olsen, Gene V. Roe, and Craig Glennie. Synthesis of Transportation Applications of Mobile LIDAR. *Remote Sensing*, 5(9):4652–4692, 2013. doi:10.3390/rs5094652.

[27] Dheerendra Pratap Singh and Manohar Yadav. Deep learning-based semantic segmentation of three-dimensional point cloud: a comprehensive review. *International Journal of Remote Sensing*, 45(2):532–586, 2024. doi:10.1080/01431161.2023.2297177.

[28] Yuxing Xie, Jiaojiao Tian, and Xiao Xiang Zhu. Linking Points With Labels in 3D: A Review of Point Cloud Semantic Segmentation. *IEEE Geoscience and Remote Sensing Magazine*, 8(4): 38–59, 2020. doi:10.1109/MGRS.2019.2937630.

[29] Shuran Song, Samuel P. Lichtenberg, and Jianxiong Xiao. SUN RGB-D: A RGB-D scene understanding benchmark suite. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 567–576, 2015. doi:10.1109/CVPR.2015.7298655.

[30] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3D Semantic Parsing of Large-Scale Indoor Spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1534–1543, 2016. doi:10.1109/CVPR.2016.170.

[31] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, pages 2432–2443, 2017. doi:10.1109/CVPR.2017.261.

[32] Daniel Munoz, J. Andrew Bagnell, Nicolas Vandapel, and Martial Hebert. Contextual classification with functional Max-Margin Markov Networks. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 975–982, 2009. doi:10.1109/CVPR.2009.5206590.

[33] Mark De Deuge, Alastair Quadros, Calvin Hung, and Bertrand Douillard. Unsupervised Feature Learning for Classification of Outdoor 3D Scans. In *Australasian conference on robitics and automation*, volume 2. University of New South Wales Kensington, Australia, 2013.

[34] Andrés Serna, Beatriz Marcotegui, François Goulette, and Jean-Emmanuel Deschaud. Paris-rue-Madame database: a 3D mobile laser scanner dataset for benchmarking urban detection, segmentation and classification methods. In *4th International Conference on Pattern Recognition, Applications and Methods ICPRAM 2014*, 2014.

[35] Xavier Roynard, Jean-Emmanuel Deschaud, and François Goulette. Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *The International Journal of Robotics Research*, 37(6):545–557, 2018. doi:10.1177/0278364918767506.

[36] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jürgen Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9296–9306, 2019. doi:10.1109/ICCV.2019.00939.

[37] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012. doi:10.1109/CVPR.2012.6248074.

[38] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A Multimodal Dataset for Autonomous Driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11618–11628, 2020. doi:10.1109/CVPR42600.2020.01164.

[39] Timo Hackel, Nikolay Savinov, Lubor Ladicky, Jan D Wegner, Konrad Schindler, and Marc Pollefeys. Semantic3D.net: A new Large-scale Point Cloud Classification Benchmark. *arXiv preprint arXiv:1704.03847*, 2017. doi:10.48550/arXiv.1704.03847.

[40] Joachim Niemeyer, Franz Rottensteiner, and Uwe Soergel. Contextual classification of lidar data and building object detection in urban areas. *ISPRS Journal of Photogrammetry and Remote Sensing*, 87:152–165, 2014. doi:10.1016/j.isprsjprs.2013.11.001.

[41] Michael Cramer. The dgpf-test on digital airborne camera evaluation overview and test design. *Photogrammetrie-Fernerkundung-Geoinformation*, 2:73–82, 2010. doi:10.1127/1432?8364/2010/0041.

[42] Michael Kölle, Dominik Laupheimer, Stefan Schmohl, Norbert Haala, Franz Rottensteiner, Jan Dirk Wegner, and Hugo Ledoux. The Hessigheim 3D (H3D) benchmark on semantic segmentation of high-resolution 3D point clouds and textured meshes from UAV LiDAR and Multi-View-Stereo. *ISPRS Open Journal of Photogrammetry and Remote Sensing*, 1:100001, 2021. doi:10.1016/j.ophoto.2021.100001.

[43] Emily R. Lines, Matt Allen, Carlos Cabo, Kim Calders, Amandine Debus, Stuart W. D. Grieve, Milto Miltiadou, Adam Noach, Harry J. F. Owen, and Stefano Puliti. AI applications in forest monitoring need remote sensing benchmark datasets. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 4528–4533, 2022. doi:10.1109/BigData55660.2022.10020772.

[44] Stefano Puliti, Grant Pearse, Peter Surový, Luke Wallace, Markus Hollaus, Maciej Wielgosz, and Rasmus Astrup. FOR-instance: a UAV laser scanning benchmark dataset for semantic and instance segmentation of individual trees. *arXiv preprint arXiv:2309.01279*, 2023. doi:10.48550/arXiv.2309.01279.

[45] Stéphane Momo Takoudjou, Pierre Ploton, Bonaventure Sonké, Jan Hackenberg, Sébastien Griffon, Francois de Coligny, Narcisse Guy Kamdem, Moses Libalah, Gislain II Mofack, Gilles Le Moguédec, Raphaël Pélissier, and Nicolas Barbier. Data from: Using terrestrial laser scanning data to estimate large tropical trees biomass and calibrate allometric models: a comparison with traditional destructive approach [Dataset], 2018. URL `https://doi.org/10.5061/dryad.10hq7`.

[46] Di Wang, Stéphane Momo Takoudjou, and Eric Casella. LeWoS: A universal leaf-wood classification method to facilitate the 3D modelling of large tropical trees using terrestrial LiDAR [Dataset], 2020. URL `https://doi.org/10.5061/dryad.np5hqbzp6`.

[47] Stéphane Momo Takoudjou, Pierre Ploton, Bonaventure Sonké, Jan Hackenberg, Sébastien Griffon, Francois de Coligny, Narcisse Guy Kamdem, Moses Libalah, Gislain II Mofack, Gilles Le Moguédec, Raphaël Pélissier, and Nicolas Barbier. Using terrestrial laser scanning data to estimate large tropical trees biomass and calibrate allometric models: A comparison with traditional destructive approach. *Methods in Ecology and Evolution*, 9(4):905–916, 2018. doi:10.1111/2041-210X.12933.

[48] Tengping Jiang, Qinyu Zhang, Shan Liu, Chong Liang, Lei Dai, Zequn Zhang, Jian Sun, and Yongjun Wang. LWSNet: A Point-Based Segmentation Network for Leaf-Wood Separation of Individual Trees. *Forests*, 14(7), 2023. doi:10.3390/f14071303.

[49] Xinlian Liang, Juha Hyyppä, Harri Kaartinen, Matti Lehtomäki, Jiri Pyörälä, Norbert Pfeifer, Markus Holopainen, Gábor Brolly, Pirotti Francesco, Jan Hackenberg, Huabing Huang, Hyun-Woo Jo, Masato Katoh, Luxia Liu, Martin Mokroš, Jules Morel, Kenneth Olofsson, Jose Poveda-Lopez, Jan Trochta, Di Wang, Jinhu Wang, Zhouxi Xi, Bisheng Yang, Guang Zheng, Ville Kankare, Ville Luoma, Xiaowei Yu, Liang Chen, Mikko Vastaranta, Ninni Saarinen, and Yunsheng Wang. International benchmarking of terrestrial laser scanning approaches for forest inventories. *ISPRS Journal of Photogrammetry and Remote Sensing*, 144:137–179, 2018. doi:10.1016/j.isprsjprs.2018.06.021.

[50] Bruce D. Cook, Lawrence A. Corp, Ross F. Nelson, Elizabeth M. Middleton, Douglas C. Morton, Joel T. McCorkel, Jeffrey G. Masek, Kenneth J. Ranson, Vuong Ly, and Paul M. Montesano. NASA Goddard's LiDAR, Hyperspectral and Thermal (G-LiHT) Airborne Imager. *Remote Sensing*, 5(8):4045–4066, 2013. doi:10.3390/rs5084045.

[51] National Land Survey of Finland. Laser scanning data 5 p, 2024. URL `https://www.maanmittauslaitos.fi/en/maps-and-spatial-data/datasets-and-interfaces/product-descriptions/laser-scanning-data-5-p`. Accessed: 25.1.2024.

[52] Ben. G. Weinstein, Sarah J. Graves, Sergio Marconi, Aditya Singh, Alina Zare, Dylan Stewart, Stephanie A. Bohlman, and Ethan P. White. A benchmark dataset for individual tree crown delineation in co-registered airborne RGB, LiDAR and hyperspectral imagery from the National Ecological Observation Network. *bioRxiv*, 2020. doi:10.1101/2020.11.16.385088.

[53] Tockner Andreas, Gollob Christoph, Ritter Tim, and Nothdurft Arne. LAUTx - Individual Tree Point Clouds from Austrian forest Inventory plots, 2022. URL `https://doi.org/10.5281/zenodo.6560112`.

[54] Hannah Weiser, Jannika Schäfer, Lukas Winiwarter, Nina Krašovec, Fabian Ewald Fassnacht, and Bernhard Höfle. Individual tree point clouds and tree measurements from multi-platform laser scanning in German forests. *Earth System Science Data*, 14(7):2989–3012, 2022. doi:10.5194/essd-14-2989-2022.

[55] Hannah Weiser, Jannika Schäfer, Lukas Winiwarter, Nina Krašovec, Christian Seitz, Marian Schimka, Katharina Anders, Daria Baete, Andressa Soarez Braz, Johannes Brand, Denis Debroize, Paula Kuss, Lioba Lucia Martin, Angelo Mayer, Torben Schrempp, Lisa-Maricia Schwarz, Veit Ulrich, Fabian E. Fassnacht, and Bernhard Höfle. Terrestrial, UAV-borne, and airborne laser scanning point clouds of central European forest plots, Germany, with extracted individual trees and manual forest inventory measurements, 2022. URL `https://doi.org/10.1594/PANGAEA.942856`.

[56] Stefano Puliti, Grant Pearse, Peter Surový, Luke Wallace, Markus Hollaus, Maciej Wielgosz, and Rasmus Astrup. FOR-instance: a UAV laser scanning benchmark dataset for semantic and instance segmentation of individual trees, 2023. URL `https://doi.org/10.5281/zenodo.8287792`.

[57] Shams Forruque Ahmed, Md Sakib Bin Alam, Maruf Hassan, Mahtabin Rodela Rozbu, Taoseef Ishtiak, Nazifa Rafa, M Mofijur, ABM Shawkat Ali, and Amir H Gandomi. Deep learning modelling techniques: current progress, applications, advantages, and challenges. *Artificial Intelligence Review*, pages 1–97, 2023. doi:10.1007/s10462-023-10466-8.

[58] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[59] David G. Lowe. Object recognition from local scale-invariant features. In *The Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999. doi:10.1109/ICCV.1999.790410.

[60] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005. doi:10.1109/TPAMI.2005.188.

[61] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. doi:10.1007/BF02551274.

[62] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9: 1735–80, 12 1997. doi:10.1162/neco.1997.9.8.1735.

[63] Eshaan Nichani, Adityanarayanan Radhakrishnan, and Caroline Uhler. Do deeper convolutional networks perform better? In *International Conference on Machine Learning*, 2021.

[64] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[65] Anis Koubaa. GPT-4 vs. GPT-3.5: A Concise Showdown. *Preprints*, 2023. doi:10.20944/preprints202303.0422.v1.

[66] Meta AI. Papers with Code. `https://paperswithcode.com/`, 2024. Accessed: 22.1.2024.

[67] Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. Point Transformer V3: Simpler, Faster, Stronger. *arXiv preprint arXiv:2312.10035*, 2023. doi:10.48550/arXiv.2312.10035.

[68] Haoyi Zhu, Honghui Yang, Xiaoyang Wu, Di Huang, Sha Zhang, Xianglong He, Tong He, Hengshuang Zhao, Chunhua Shen, Yu Qiao, et al.. PonderV2: Pave the Way for 3D Foundation Model with A Universal Pre-training Paradigm. *arXiv preprint arXiv:2310.08586*, 2023. doi:10.48550/arXiv.2310.08586.

[69] Yu-Qi Yang, Yu-Xiao Guo, Jian-Yu Xiong, Yang Liu, Hao Pan, Peng-Shuai Wang, Xin Tong, and Baining Guo. Swin3D: A Pretrained Transformer Backbone for 3D Indoor Scene Understanding. *arXiv preprint arXiv:2304.06906*, 2023. doi:10.48550/arXiv.2304.06906.

[70] Xin Deng, WenYu Zhang, Qing Ding, and XinMing Zhang. PointVector: A Vector Representation in Point Cloud Analysis. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9455–9465, June 2023.

[71] Xin Lai, Yukang Chen, Fanbin Lu, Jianhui Liu, and Jiaya Jia. Spherical Transformer for LiDAR-Based 3D Recognition. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17545–17555, 2023.

[72] Lingdong Kong, Youquan Liu, Runnan Chen, Yuexin Ma, Xinge Zhu, Yikang Li, Yuenan Hou, Yu Qiao, and Ziwei Liu. Rethinking Range View Representation for LiDAR Segmentation. In *The IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 228–240, 2023.

[73] Gary Marcus. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018. doi:10.48550/arXiv.1801.00631.

[74] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017. doi:10.48550/arXiv.1710.05941.

[75] Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying ReLU and Initialization: Theory and Numerical Examples. *Communications in Computational Physics*, 28(5):1671–1706, 2020. doi:10.4208/cicp.oa-2020-0165.

[76] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al.. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing.*, volume 30, page 3, 2013.

[77] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015. doi:10.1109/ICCV.2015.123.

[78] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980. doi:10.1007/BF00344251.

[79] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9224–9232, 2018. doi:10.1109/CVPR.2018.00961.

[80] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3070–3079, 2019. doi:10.1109/CVPR.2019.00319.

[81] Haotian Tang, Zhijian Liu, Xiuyu Li, Yujun Lin, and Song Han. TorchSparse: Efficient Point Cloud Inference Engine. In D. Marculescu, Y. Chi, and C. Wu, editors, *Machine Learning and Systems*, volume 4, pages 302–315, 2022.

[82] Saining Xie, Jiatao Gu, Demi Guo, Charles R. Qi, Leonidas Guibas, and Or Litany. PointContrast: Unsupervised Pre-training for 3D Point Cloud Understanding. In *Computer Vision – ECCV 2020*, pages 574–591. Springer International Publishing, 2020. doi:10.1007/978-3-030-58580-8_34.

[83] Daniel Maturana and Sebastian Scherer. VoxNet: A 3D Convolutional Neural Network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, 2015. doi:10.1109/IROS.2015.7353481.

[84] Luca Caltagirone, Samuel Scheidegger, Lennart Svensson, and Mattias Wahde. Fast LIDAR-based road detection using fully convolutional neural networks. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1019–1024, 2017. doi:10.1109/IVS.2017.7995848.

[85] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3D Object Detection Network for Autonomous Driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6526–6534, 2017. doi:10.1109/CVPR.2017.691.

[86] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1887–1893, 2018. doi:10.1109/ICRA.2018.8462926.

[87] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the Effective Receptive Field in Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[88] Benjamin Graham. Sparse 3d convolutional neural networks. *arXiv preprint arXiv:1505.02890*, 2015. doi:10.48550/arXiv.1505.02890.

[89] Sergio Pissanetzky. *Sparse matrix technology-electronic edition*. Academic Press, 1984.

[90] Benjamin Graham. Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*, 2014. doi:10.48550/arXiv.1409.6070.

[91] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3Deep: Fast object detection in 3D point clouds using efficient convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1355–1361, 2017. doi:10.1109/ICRA.2017.7989161.

[92] Afia Zafar, Muhammad Aamir, Nazri Mohd Nawi, Ali Arshad, Saman Riaz, Abdulrahman Alruban, Ashit Kumar Dutta, and Sultan Almotairi. A Comparison of Pooling Methods for Convolutional Neural Networks. *Applied Sciences*, 12(17), 2022. doi:10.3390/app12178643.

[93] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

[94] Marc'Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007. doi:10.1109/CVPR.2007.383157.

[95] Dingjun Yu, Hanli Wang, Peiqiu Chen, and Zhihua Wei. Mixed Pooling for Convolutional Neural Networks. In *Rough Sets and Knowledge Technology*, pages 364–375. Springer International Publishing, 2014. doi:10.1007/978-3-319-11740-9_34.

[96] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A Theoretical Analysis of Feature Pooling in Visual Recognition. In *The 27th International Conference on International Conference on Machine Learning*, page 111–118. Omnipress, 2010. doi:10.5555/3104322.3104338.

[97] Hossein Gholamalinezhad and Hossein Khosravi. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*, 2020. doi:10.48550/arXiv.2009.07485.

[98] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, 1998. doi:10.1007/3-540-49430-8_2.

[99] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *The 32nd International Conference on Machine Learning*, volume 37, pages 448–456, 2015.

[100] Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding Batch Normalization. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[101] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. doi:10.1109/CVPR.2016.90.

[102] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5353–5360, 2015. doi:10.1109/CVPR.2015.7299173.

[103] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015. doi:10.48550/arXiv.1505.00387.

[104] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. *arXiv preprint arXiv:1605.07146*, 2016. doi:10.48550/arXiv.1605.07146.

[105] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018. doi:10.48550/arXiv.1810.04805.

[106] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. 2018.

[107] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. In *Computer Vision – ECCV 2016*, pages 630–645. Springer International Publishing, 2016. doi:10.1007/978-3-319-46493-0_38.

[108] Anqi Mao, Mehryar Mohri, and Yutao Zhong. Cross-Entropy Loss Functions: Theoretical Analysis and Applications. In *The 40th International Conference on Machine Learning*, volume 202, pages 23803–23828. PMLR, 2023.

[109] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2): 318–327, 2020. doi:10.1109/TPAMI.2018.2858826.

[110] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. doi:10.48550/arXiv.1609.04747.

[111] Diederik P. Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014. doi:10.48550/arXiv.1412.6980.

[112] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's thesis, (in Finnish), Univ. Helsinki, 1970.

[113] Andreas Griewank. Who invented the reverse mode of differentiation? *Documenta Mathematica, Extra Volume ISMP*, pages 389–400, 2012.

[114] Paul J. Werbos. Applications of advances in nonlinear sensitivity analysis. In *System Modeling and Optimization*, pages 762–770. Springer Berlin Heidelberg, 1982. doi:10.1007/BFb0006203.

[115] David E Rumelhart, Geoffrey E. Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi:10.1038/323533a0.

[116] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, 2024. doi:10.1145/3620665.3640366.

[117] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow, Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[118] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. doi:10.5555/2627435.2670313.

[119] Rui Zhang, Yichao Wu, Wei Jin, and Xiaoman Meng. Deep-Learning-Based Point Cloud Semantic Segmentation: A Survey. *Electronics*, 12(17), 2023. doi:10.3390/electronics12173642.

[120] Jiaying Zhang, Xiaoli Zhao, Zheng Chen, and Zhejun Lu. A Review of Deep Learning-Based Semantic Segmentation for Point Cloud. *IEEE Access*, 7:179118–179133, 2019. doi:10.1109/ACCESS.2019.2958671.

[121] Jingyi Wang, Yu Liu, Hanlin Tan, and Maojun Zhang. A survey on weakly supervised 3D point cloud semantic segmentation. *IET Computer Vision*, 1(14), 2023. doi:10.1049/cvi2.12250.

[122] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view Convolutional Neural Networks for 3D Shape Recognition. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 945–953, 2015. doi:10.1109/ICCV.2015.114.

[123] Alexandre Boulch, Joris Guerry, Bertrand Le Saux, and Nicolas Audebert. SnapNet: 3D point cloud semantic labeling with 2D deep segmentation networks. *Computers & Graphics*, 71: 189–198, 2018. doi:10.1016/j.cag.2017.11.010.

[124] Joris Guerry, Alexandre Boulch, Bertrand Le Saux, Julien Moras, Aurélien Plyer, and David Filliat. SnapNet-R: Consistent 3D Multi-view Semantic Labeling for Robotics. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 669–678, 2017. doi:10.1109/ICCVW.2017.85.

[125] Bichen Wu, Xuanyu Zhou, Sicheng Zhao, Xiangyu Yue, and Kurt Keutzer. SqueezeSegV2: Improved Model Structure and Unsupervised Domain Adaptation for Road-Object Segmentation from a LiDAR Point Cloud. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4376–4382, 2019. doi:10.1109/ICRA.2019.8793495.

[126] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. RangeNet++: Fast and Accurate LiDAR Semantic Segmentation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4213–4220, 2019. doi:10.1109/IROS40897.2019.8967762.

[127] Chenfeng Xu, Bichen Wu, Zining Wang, Wei Zhan, Peter Vajda, Kurt Keutzer, and Masayoshi Tomizuka. SqueezeSegV3: Spatially-Adaptive Convolution for Efficient Point-Cloud Segmentation. In *Computer Vision – ECCV 2020*, pages 1–19. Springer International Publishing, 2020. doi:10.1007/978-3-030-58604-1_1.

[128] Jing Huang and Suya You. Point cloud labeling using 3D Convolutional Neural Network. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2670–2675, 2016. doi:10.1109/ICPR.2016.7900038.

[129] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. SEGCloud: Semantic Segmentation of 3D Point Clouds. In *2017 International Conference on 3D Vision (3DV)*, pages 537–547, 2017. doi:10.1109/3DV.2017.00067.

[130] Truc Le and Ye Duan. PointGrid: A Deep Network for 3D Shape Understanding. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9204–9214, 2018. doi:10.1109/CVPR.2018.00959.

[131] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017. doi:10.1109/CVPR.2017.243.

[132] Jaehyun Park, Chansoo Kim, Soyeong Kim, and Kichun Jo. PCSCNet: Fast 3D semantic segmentation of LiDAR point cloud for autonomous car using point convolution and sparse convolution network. *Expert Systems with Applications*, 212:118815, 2023. doi:10.1016/j.eswa.2022.118815.

[133] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. OctNet: Learning Deep 3D Representations at High Resolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6620–6629, 2017. doi:10.1109/CVPR.2017.701.

[134] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN: octree-based convolutional neural networks for 3D shape analysis. *ACM Transactions on Graphics*, 36(4), 2017. doi:10.1145/3072959.3073608.

[135] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. SPLATNet: Sparse Lattice Networks for Point Cloud Processing. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2530–2539, 2018. doi:10.1109/CVPR.2018.00268.

[136] Radu Alexandru Rosu, Peer Schütt, Jan Quenzel, and Sven Behnke. LatticeNet: Fast point cloud segmentation using permutohedral lattices. *arXiv preprint arXiv:1912.05905*, 2019. doi:10.48550/arXiv.1912.05905.

[137] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[138] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11105–11114, June 2020. doi:10.1109/CVPR42600.2020.01112.

[139] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution On X-Transformed Points. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[140] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6410–6419, 2019. doi:10.1109/ICCV.2019.00651.

[141] Mingyang Jiang, Yiran Wu, Tianqi Zhao, Zelin Zhao, and Cewu Lu. PointSIFT: A SIFT-like Network Module for 3D Point Cloud Semantic Segmentation. *arXiv preprint arXiv:1807.00652*, 2018. doi:10.48550/arXiv.1807.00652.

[142] Loic Landrieu and Martin Simonovsky. Large-Scale Point Cloud Semantic Segmentation with Superpoint Graphs. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018. doi:10.1109/CVPR.2018.00479.

[143] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics*, 38(5), 2019. doi:10.1145/3326362.

[144] Huan Lei, Naveed Akhtar, and Ajmal Mian. Spherical kernel for efficient graph convolution on 3d point clouds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(10): 3664–3680, 2021. doi:10.1109/TPAMI.2020.2983410.

[145] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241. Springer International Publishing, 2015. doi:10.1007/978-3-319-24574-4_28.

[146] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al.. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv preprint arXiv:2010.11929*, 2020. doi:10.48550/arXiv.2010.11929.

[147] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. PCT: Point cloud transformer. *Computational Visual Media*, 7:187–199, 2021. doi:10.1007/s41095-021-0229-5.

[148] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point Transformer V2: Grouped Vector Attention and Partition-based Pooling. In *Advances in Neural Information Processing Systems*, volume 35, pages 33330–33342. Curran Associates, Inc., 2022.

[149] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point Transformer. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16239–16248, 2021. doi:10.1109/ICCV48922.2021.01595.

[150] Xin Lai, Jianhui Liu, Li Jiang, Liwei Wang, Hengshuang Zhao, Shu Liu, Xiaojuan Qi, and Jiaya Jia. Stratified Transformer for 3D Point Cloud Segmentation. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8490–8499, 2022. doi:10.1109/CVPR52688.2022.00831.

[151] Damien Robert, Hugo Raguet, and Loic Landrieu. Efficient 3D Semantic Segmentation with Superpoint Transformer. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 17149–17158, 2023. doi:10.1109/ICCV51070.2023.01577.

[152] Haiyan Wang, Xuejian Rong, Liang Yang, Shuihua Wang, and Yingli Tian. Towards Weakly Supervised Semantic Segmentation in 3D Graph-Structured Point Clouds of Wild Scenes. In *BMVC*, page 284, 2019.

[153] Hyeokjun Kweon and Kuk-Jin Yoon. Joint Learning of 2D-3D Weakly Supervised Semantic Segmentation. In *Advances in Neural Information Processing Systems*, volume 35, pages 30499–30511. Curran Associates, Inc., 2022.

[154] Xun Xu and Gim Hee Lee. Weakly Supervised Semantic Point Cloud Segmentation: Towards 10× Fewer Labels. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13703–13712, 2020. doi:10.1109/CVPR42600.2020.01372.

[155] Yachao Zhang, Yanyun Qu, Yuan Xie, Zonghao Li, Shanshan Zheng, and Cuihua Li. Perturbed Self-Distillation: Weakly Supervised Large-Scale Point Cloud Semantic Segmentation. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15500–15508, 2021. doi:10.1109/ICCV48922.2021.01523.

[156] Qingyong Hu, Bo Yang, Guangchi Fang, Yulan Guo, Aleš Leonardis, Niki Trigoni, and Andrew Markham. SQN: Weakly-Supervised Semantic Segmentation of Large-Scale 3D Point Clouds. In *Computer Vision – ECCV 2022*, pages 600–619. Springer Nature Switzerland, 2022. doi:10.1007/978-3-031-19812-0_35.

[157] Cheng-Kun Yang, Ji-Jia Wu, Kai-Syun Chen, Yung-Yu Chuang, and Yen-Yu Lin. An MIL-Derived Transformer for Weakly Supervised Point Cloud Segmentation. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11820–11829, 2022. doi:10.1109/CVPR52688.2022.01153.

[158] Ji Hou, Benjamin Graham, Matthias Nießner, and Saining Xie. Exploring Data-Efficient 3D Scene Understanding with Contrastive Scene Contexts. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15582–15592, 2021. doi:10.1109/CVPR46437.2021.01533.

[159] Mengtian Li, Yuan Xie, Yunhang Shen, Bo Ke, Ruizhi Qiao, Bo Ren, Shaohui Lin, and Lizhuang Ma. HybridCR: Weakly-Supervised 3D Point Cloud Semantic Segmentation via Hybrid Contrastive Regularization. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14910–14919, 2022. doi:10.1109/CVPR52688.2022.01451.

[160] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. doi:10.1145/358669.358692.

[161] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR Segmentation. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9934–9943, 2021. doi:10.1109/CVPR46437.2021.00981.

[162] Matti Lehtomäki, Antero Kukko, Leena Matikainen, Juha Hyyppä, Harri Kaartinen, and Anttoni Jaakkola. Power line mapping technique using all-terrain mobile laser scanning. *Automation in Construction*, 105:102802, 2019. doi:10.1016/j.autcon.2019.03.023.

[163] Mostafa Arastounia. Automated Recognition of Railroad Infrastructure in Rural Areas from LIDAR Data. *Remote Sensing*, 7(11):14916–14938, 2015. doi:10.3390/rs71114916.

[164] Daniel Lamas, Mario Soilán, Javier Grandío, and Belén Riveiro. Automatic Point Cloud Semantic Segmentation of Complex Railway Environments. *Remote Sensing*, 13(12), 2021. doi:10.3390/rs13122332.

[165] Di Wang, Stéphane Momo Takoudjou, and Eric Casella. LeWoS: A universal leaf-wood classification method to facilitate the 3D modelling of large tropical trees using terrestrial LiDAR. *Methods in Ecology and Evolution*, 11(3):376–389, 2020. doi:10.1111/2041-210X.13342.

[166] Zhilin Tian and Shihua Li. Graph-Based Leaf–Wood Separation Method for Individual Trees Using Terrestrial Lidar Point Clouds. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–11, 2022. doi:10.1109/TGRS.2022.3218603.

[167] Florent Poux and Roland Billen. Voxel-based 3D Point Cloud Semantic Segmentation: Unsupervised Geometric and Relationship Featuring vs Deep Learning Methods. *ISPRS International Journal of Geo-Information*, 8(5), 2019. doi:10.3390/ijgi8050213.

[168] Jiaxu Liu, Zhengdi Yu, Toby P Breckon, and Hubert PH Shum. U3DS^3: Unsupervised 3D Semantic Scene Segmentation. *arXiv preprint arXiv:2311.06018*, 2023. doi:10.48550/arXiv.2311.06018.

[169] Zisheng Chen, Hongbin Xu, Weitao Chen, Zhipeng Zhou, Haihong Xiao, Baigui Sun, Xuansong Xie, and Wenxiong Kang. PointDC: Unsupervised Semantic Segmentation of 3D Point Clouds via Cross-modal Distillation and Super-Voxel Clustering. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14244–14253, 2023. doi:10.1109/ICCV51070.2023.01314.

[170] Jeremie Papon, Alexey Abramov, Markus Schoeler, and Florentin Worgotter. Voxel cloud connectivity segmentation-supervoxels for point clouds. In *The IEEE conference on computer vision and pattern recognition*, pages 2027–2034, 2013. doi:10.1109/CVPR.2013.264.

[171] Rolf Adams and Leanne Bischof. Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647, 1994. doi:10.1109/34.295913.

[172] Mark Hamilton, Zhoutong Zhang, Bharath Hariharan, Noah Snavely, and William T. Freeman. Unsupervised Semantic Segmentation by Distilling Feature Correspondences. In *International Conference on Learning Representations*, 2022.

[173] Eric Hyyppä, Juha Hyyppä, Teemu Hakala, Antero Kukko, Michael A. Wulder, Joanne C. White, Jiri Pyörälä, Xiaowei Yu, Yunsheng Wang, Juho-Pekka Virtanen, Onni Pohjavirta, Xinlian Liang, Markus Holopainen, and Harri Kaartinen. Under-canopy UAV laser scanning for accurate forest field measurements. *ISPRS Journal of Photogrammetry and Remote Sensing*, 164:41–60, 2020. doi:10.1016/j.isprsjprs.2020.03.021.

[174] Jean-François Côté, Jean-Luc Widlowski, Richard A. Fournier, and Michel M. Verstraete. The structural and radiative consistency of three-dimensional tree reconstructions from terrestrial lidar. *Remote Sensing of Environment*, 113(5):1067–1081, 2009. doi:10.1016/j.rse.2009.01.017.

[175] Shengli Tao, Qinghua Guo, Yanjun Su, Shiwu Xu, Yumei Li, and Fangfang Wu. A Geometric Method for Wood-Leaf Separation Using Terrestrial and Simulated Lidar Data. *Photogrammetric Engineering & Remote Sensing*, 81(10):767–776, 2015. doi:10.14358/PERS.81.10.767.

[176] Jiaying Wu, Kerry Cawse-Nicholson, and Jan van Aardt. 3D Tree reconstruction from simulated small footprint waveform lidar. *Photogrammetric Engineering & Remote Sensing*, 79(12): 1147–1157, 2013. doi:10.14358/PERS.79.12.1147.

[177] Xiaoyuan Yang, Alan H. Strahler, Crystal B. Schaaf, David L.B. Jupp, Tian Yao, Feng Zhao, Zhuosen Wang, Darius S. Culvenor, Glenn J. Newnham, Jenny L. Lovell, Ralph O. Dubayah, Curtis E. Woodcock, and Wenge Ni-Meister. Three-dimensional forest reconstruction and structural parameter retrievals using a terrestrial full-waveform lidar instrument (Echidna®). *Remote Sensing of Environment*, 135:36–51, 2013. doi:10.1016/j.rse.2013.03.020.

[178] Martin Béland, Dennis D. Baldocchi, Jean-Luc Widlowski, Richard A. Fournier, and Michel M. Verstraete. On seeing the wood from the leaves and the role of voxel size in determining leaf area distribution of forests with terrestrial LiDAR. *Agricultural and Forest Meteorology*, 184: 82–97, 2014. doi:10.1016/j.agrformet.2013.09.005.

[179] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *The Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, volume 96, pages 226–231, 1996.

[180] Hui Xu, Nathan Gossett, and Baoquan Chen. Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Transactions on Graphics (TOG)*, 26(4):19–es, 2007. doi:10.1145/1289603.1289610.

[181] Edsger Wybe Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:doi.org/10.1007/BF01386390.

[182] Yotam Livny, Feilong Yan, Matt Olson, Baoquan Chen, Hao Zhang, and Jihad El-Sana. Automatic reconstruction of tree skeletal structures from point clouds. In *ACM SIGGRAPH Asia 2010 Papers*. Association for Computing Machinery, 2010. doi:10.1145/1866158.1866177.

[183] Pasi Raumonen, Mikko Kaasalainen, Markku Åkerblom, Sanna Kaasalainen, Harri Kaartinen, Mikko Vastaranta, Markus Holopainen, Mathias Disney, and Philip Lewis. Fast Automatic Precision Tree Models from Terrestrial Laser Scanner Data. *Remote Sensing*, 5(2):491–520, 2013. doi:10.3390/rs5020491.

[184] D. Belton, S. Moncrieff, and J. Chapman. Processing tree point clouds using Gaussian Mixture Models. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-5/W2:43–48, 2013. doi:10.5194/isprsannals-II-5-W2-43-2013.

[185] Geoffrey J. McLachlan and David Peel. *Finite mixture models*, volume 299. Wiley, 2000.

[186] Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space . *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. doi:10.1080/14786440109462720.

[187] Di Wang, Jasmin Brunner, Zhenyu Ma, Hao Lu, Markus Hollaus, Yong Pang, and Norbert Pfeifer. Separating Tree Photosynthetic and Non-Photosynthetic Components from Point Cloud Data Using Dynamic Segment Merging. *Forests*, 9(5), 2018. doi:10.3390/f9050252.

[188] Matheus B. Vicari, Mathias Disney, Phil Wilkes, Andrew Burt, Kim Calders, and William Woodgate. Leaf and wood classification framework for terrestrial LiDAR point clouds. *Methods in Ecology and Evolution*, 10(5):680–694, 2019. doi:10.1111/2041-210X.13144.

[189] Peng Wan, Jie Shao, Shuangna Jin, Tiejun Wang, Shengmei Yang, Guangjian Yan, and Wuming Zhang. A novel and efficient method for wood–leaf separation from terrestrial laser scanning point clouds at the forest plot level. *Methods in Ecology and Evolution*, 12(12):2473–2486, 2021. doi:10.1111/2041-210X.13715.

[190] A. Shcherbcheva, M. B. Campos, X. Liang, E. Puttonen, and Y. Wang. Unsupervised Statistical Approach for Tree-Level Separation of Foliage and Non-Leaf Components from Point Clouds. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLVIII-1/W2-2023:1787–1794, 2023. doi:10.5194/isprs-archives-XLVIII-1-W2-2023-1787-2023.

[191] Lixia Ma, Guang Zheng, Jan U. H. Eitel, L. Monika Moskal, Wei He, and Huabing Huang. Improved Salient Feature-Based Approach for Automatically Separating Photosynthetic and Nonphotosynthetic Components Within Terrestrial Lidar Point Cloud Data of Forest Canopies. *IEEE Transactions on Geoscience and Remote Sensing*, 54(2):679–696, 2016. doi:10.1109/TGRS.2015.2459716.

[192] Ting Yun, Feng An, Weizheng Li, Yuan Sun, Lin Cao, and Lianfeng Xue. A Novel Approach for Retrieving Tree Leaf Area from Ground-Based LiDAR. *Remote Sensing*, 8(11), 2016. doi:10.3390/rs8110942.

[193] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. doi:10.1007/BF00994018.

[194] D. Wang, M. Hollaus, and N. Pfeifer. Feasibility of machine learning methods for separating wood and leaf points from terrestrial laser scanning data. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W4:157–164, 2017. doi:10.5194/isprs-annals-IV-2-W4-157-2017.

[195] Igor Kononenko. Comparison of inductive and naive Bayesian learning approaches to automatic knowledge acquisition. *Current trends in knowledge acquisition*, 8:190, 1990.

[196] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001. doi:10.1023/A:1010933404324.

[197] Junjie Zhou, Hongqiang Wei, Guiyun Zhou, and Lihui Song. Separating Leaf and Wood Points in Terrestrial Laser Scanning Data Using Multiple Optimal Scales. *Sensors*, 19(8), 2019. doi:10.3390/s19081852.

[198] Sruthi M. Krishna Moorthy, Kim Calders, Matheus B. Vicari, and Hans Verbeeck. Improved Supervised Learning-Based Approach for Leaf and Wood Classification From LiDAR Point Clouds of Forests. *IEEE Transactions on Geoscience and Remote Sensing*, 58(5):3057–3070, 2020. doi:10.1109/TGRS.2019.2947198.

[199] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. Association for Computing Machinery, 2016. doi:10.1145/2939672.2939785.

[200] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: a highly efficient gradient boosting decision tree. In *The 31st International Conference on Neural Information Processing Systems*, pages 3149–3157. Curran Associates Inc., 2017.

[201] Xi Zhu, Andrew K. Skidmore, Roshanak Darvishzadeh, K. Olaf Niemann, Jing Liu, Yifang Shi, and Tiejun Wang. Foliar and woody materials discriminated using terrestrial LiDAR in a mixed natural forest. *International Journal of Applied Earth Observation and Geoinformation*, 64: 43–50, 2018. doi:10.1016/j.jag.2017.09.004.

[202] Jules Morel, Alexandra Bac, and Takashi Kanai. Segmentation of unbalanced and in-homogeneous point clouds and its application to 3D scanned trees. *The Visual Computer*, 36 (10-12):2419–2431, 2020. doi:10.1007/s00371-020-01966-7.

[203] Lloyd Windrim and Mitch Bryson. Forest Tree Detection and Segmentation using High Resolution Airborne LiDAR. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3898–3904, 2019. doi:10.1109/IROS40897.2019.8967885.

[204] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[205] Lloyd Windrim and Mitch Bryson. Detection, Segmentation, and Model Fitting of Individual Tree Stems from Airborne Laser Scanning of Forests Using Deep Learning. *Remote Sensing*, 12 (9), 2020. doi:10.3390/rs12091469.

[206] Xingyu Shen, Qingqing Huang, Xin Wang, Jiang Li, and Benye Xi. A Deep Learning-Based Method for Extracting Standing Wood Feature Parameters from Terrestrial Laser Scanning Point Clouds of Artificially Planted Forest. *Remote Sensing*, 14(15), 2022. doi:10.3390/rs14153842.

[207] Dong-Hyeon Kim, Chi-Ung Ko, Dong-Geun Kim, Jin-Taek Kang, Jeong-Mook Park, and Hyung-Ju Cho. Automated Segmentation of Individual Tree Structures Using Deep Learning over LiDAR Point Cloud Data. *Forests*, 14(6), 2023. doi:10.3390/f14061159.

[208] Wenxia Dai, Yiheng Jiang, Wen Zeng, Ruibo Chen, Yongyang Xu, Ningning Zhu, Wen Xiao, Zhen Dong, and Qingfeng Guan. MDC-Net: a multi-directional constrained and prior assisted neural network for wood and leaf separation from terrestrial laser scanning. *International Journal of Digital Earth*, 16(1):1224–1245, 2023. doi:10.1080/17538947.2023.2198261.

[209] Xiaowei Yu, Juha Hyyppä, Paula Litkey, Harri Kaartinen, Mikko Vastaranta, and Markus Holopainen. Single-Sensor Solution to Tree Species Classification Using Multispectral Airborne Laser Scanning. *Remote Sensing*, 9(2), 2017. doi:10.3390/rs9020108.

[210] Mikko Kukkonen, Matti Maltamo, Lauri Korhonen, and Petteri Packalen. Multispectral Airborne LiDAR Data in the Prediction of Boreal Tree Species Composition. *IEEE Transactions on Geoscience and Remote Sensing*, 57(6):3462–3471, 2019. doi:10.1109/TGRS.2018.2885057.

[211] Leena Matikainen, Kirsi Karila, Juha Hyyppä, Paula Litkey, Eetu Puttonen, and Eero Ahokas. Object-based analysis of multispectral airborne laser scanner data for land cover classification and map updating. *ISPRS Journal of Photogrammetry and Remote Sensing*, 128:298–313, 2017. doi:10.1016/j.isprsjprs.2017.04.005.

[212] Tee-Ann Teo and Hsien-Ming Wu. Analysis of land cover classification using multi-wavelength lidar system. *Applied Sciences*, 7(7), 2017. doi:10.3390/app7070663.

[213] NovAtel Inc., Canada. Waypoint Inertial Explorer (version 8.9), 2022. URL `https://novatel.com/products/waypoint-post-processing-software/inertial-explorer`.

[214] Eric Hyyppä, Antero Kukko, Risto Kaijaluoto, Joanne C. White, Michael A. Wulder, Jiri Pyörälä, Xinlian Liang, Xiaowei Yu, Yunsheng Wang, Harri Kaartinen, Juho-Pekka Virtanen, and Juha Hyyppä. Accurate derivation of stem curve and volume using backpack mobile laser scanning. *ISPRS Journal of Photogrammetry and Remote Sensing*, 161:246–262, 2020. doi:10.1016/j.isprsjprs.2020.01.018.

[215] Matti Lehtomäki, Anttoni Jaakkola, Juha Hyyppä, Jouko Lampinen, Harri Kaartinen, Antero Kukko, Eetu Puttonen, and Hannu Hyyppä. Object Classification and Recognition From Mobile Laser Scanning Point Clouds in a Road Environment. *IEEE Transactions on Geoscience and Remote Sensing*, 54(2):1226–1239, 2016. doi:10.1109/TGRS.2015.2476502.

[216] Daniel Girardeau-Montaut. CloudCompare (version 2.12.4), 2022. URL `https://www.cloudcompare.org`.

[217] Binbin Xiang, Torben Peters, Theodora Kontogianni, Frawa Vetterli, Stefano Puliti, Rasmus Astrup, and Konrad Schindler. Towards accurate instance segmentation in large-scale LiDAR point clouds. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, X-1/W1-2023:605–612, 2023. doi:10.5194/isprs-annals-X-1-W1-2023-605-2023.

[218] Richard Kershner. The Number of Circles Covering a Set. *American Journal of Mathematics*, 61(3):665–671, 1939. doi:10.2307/2371320.

[219] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017. doi:10.1109/CVPR.2017.106.

[220] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28 (2):129–137, 1982. doi:10.1109/TIT.1982.1056489.

[221] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3384–3391, 2008. doi:10.1109/IROS.2008.4650967.

[222] Jang Hyun Cho, Utkarsh Mall, Kavita Bala, and Bharath Hariharan. PiCIE: Unsupervised Semantic Segmentation Using Invariance and Equivariance in Clustering. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16794–16804, June 2021. doi:10.1109/CVPR46437.2021.01652.

[223] Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi:10.1002/nav.3800020109.

[224] Timo Hackel, Jan D. Wegner, and Konrad Schindler. Contour Detection in Unstructured 3D Point Clouds. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1610–1618, 2016. doi:10.1109/CVPR.2016.178.

[225] M. Weinmann, B. Jutzi, and C. Mallet. Feature relevance assessment for the semantic interpretation of 3D point cloud data. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-5/W2:313–318, 2013. doi:10.5194/isprsannals-II-5-W2-313-2013.

[226] Martin Weinmann, Boris Jutzi, and Clément Mallet. Geometric features and their relevance for 3D point cloud classification. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 4:157–164, 2017. doi:10.5194/isprs-annals-IV-1-W1-157-2017.

[227] Martin Weinmann, Michael Weinmann, Clément Mallet, and Mathieu Brédif. A Classification-Segmentation Framework for the Detection of Individual Trees in Dense MMS Point Cloud Data Acquired in Urban Areas. *Remote Sensing*, 9(3), 2017. doi:10.3390/rs9030277.

[228] Loic Landrieu and Guillaume Obozinski. Cut Pursuit: Fast Algorithms to Learn Piecewise Constant Functions on General Weighted Graphs. *SIAM Journal on Imaging Sciences*, 10(4): 1724–1766, 2017. doi:10.1137/17M1113436.

[229] Oona Oinonen, Lassi Ruoppa, Josef Taher, Matti Lehtomäki, Leena Matikainen, Kirsi Karila, Teemu Hakala, Antero Kukko, Harri Kaartinen, and Juha Hyyppä. Unsupervised semantic segmentation of urban high-density multispectral point clouds. *arXiv preprint arXiv:2410.18520*, 2024. doi:10.48550/arXiv.2410.18520.

[230] Zhilin Tian and Shihua Li. Graph-based Leaf–Wood Separation Method for Individual Trees Using Terrestrial Lidar Point Clouds: GBSeparation—a python package, 2022. URL `https://doi.org/10.5281/zenodo.6837613`.

[231] Matheus B. Vicari. TLSeparation – A Python library for material separation from tree/forest 3D point clouds, 2017. URL `https://github.com/TLSeparation/source`.

[232] Paul Jaccard. The distribution of the flora in the alpine zone. *New phytologist*, 11(2):37–50, 1912. doi:10.1111/j.1469-8137.1912.tb05611.x.

[233] Leslie N. Smith. Cyclical Learning Rates for Training Neural Networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017. doi:10.1109/WACV.2017.58.

[234] Zhiwen Zhang, Teng Li, Xuebin Tang, Xiangda Lei, and Yuanxi Peng. Introducing Improved Transformer to Land Cover Classification Using Multispectral LiDAR Point Clouds. *Remote Sensing*, 14(15), 2022. doi:10.3390/rs14153808.

[235] David Sculley. Web-scale k-means clustering. In *The 19th International Conference on World Wide Web*. Association for Computing Machinery, 2010. doi:10.1145/1772690.1772862.

A Additional Qualitative Performance Comparisons

This section contains additional visual comparisons of segmentation results.

A.1 Qualitative Performance Comparison on Unlabeled Data

Figure A1 shows a performance comparison between GrowSP-ForMS and our fully-supervised baselines on a subsection of an unlabeled forest plot that was not part of the training or test set. Furthermore, the data in question was not used as auxiliary unlabeled data when training GrowSP-ForMS. That is, the input data is previously unseen for all models presented in the figure.

Conversely, the PointNet model performs notably worse on the unlabeled data from a previously unseen plot, which suggest that the model does not generalize well. This is most likely a result of the limited data set used during training, which appears to affect the performance of PointNet more severely than RandLA-Net or ResNet16FPN.

We note that although the segmentation results displayed in the figure appear to be of high quality, the lack of ground truth labels makes assessing the true quality of the predictions challenging. Finally, it should be noted that since no individual tree segments were available for the unlabeled data, the comparison could not be performed for the unsupervised baseline methods, since some of the algorithms require individual tree point clouds as input.

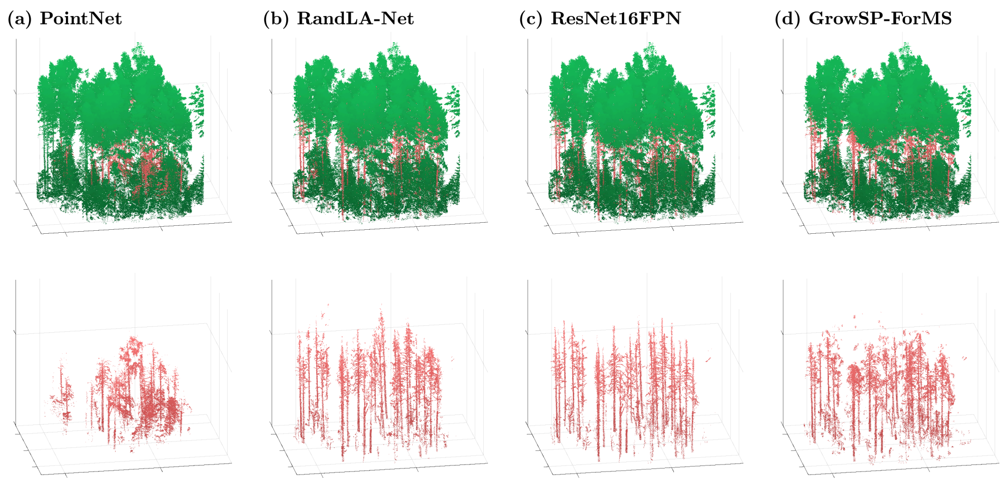


Figure A1: Qualitative results of GrowSP-ForMS and our supervised baseline methods on a subsection of an unlabeled plot that was not used when training any of the models. The top row shows the full segmentation results and the row below contains only points classified as wood. Note the absence of ground truth data, which makes assessing the true quality of the predictions challenging.

A.2 Qualitative Performance Comparison of Ablated Models

Figure A2 shows a visual comparison of the segmentation results from each model in the ablation study discussed in Section 5.2.2. The different ablation setups in the figure are described in Table A1. The qualitative results are consistent with quantitative results listed in Table 7 (see Section 5.2.2) in the sense that the segmentation results appear visually better for setups that achieved a higher accuracy. Furthermore, the visualization shows that our proposed changes to the GrowSP model also improve its qualitative performance.

The segmentation results of GrowSP-ForMS for each reflectance channel separately, as well as all two- and three-channel combinations are displayed in Figure A3. In the figures, scanners 1, 2 and 3 operate on the wavelengths 1,550 nm, 905 nm and 532 nm respectively. A more detailed description of the multispectral HeliALS-TW system can be found in Table 1 in Section 3.1. Similarly to the first ablation study, the qualitative results reflect the quantitative results listed in Table 8 (see Section 5.2.2). That is, scanner combinations that achieved a higher quantitative segmentation accuracy also provide visually better results. Additionally, the best performing GrowSP-ForMS

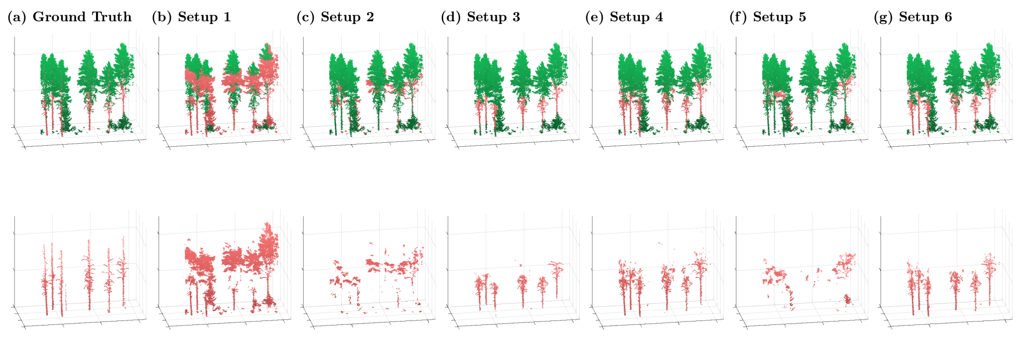


Figure A2: Qualitative results of each ablated GrowSP-ForMS model on a subsection of the **test split** of plot #1 in our multispectral data set. The top row shows the full segmentation results and the row below contains only points classified as wood. The different setups are described in Table A1.

Table A1: Description of each setup in the visual comparison of ablated models. We note that in experiments where geometric features were not utilized, 10-dimensional point feature histograms were used instead.

Setup	Oversegmentation of Predicted Labels	New Superpoints	Geometric Features	Decaying Clustering Weights
1				
2	✓			
3	✓	✓		
4	✓	✓	✓	
5	✓	✓		✓
6	✓	✓	✓	✓

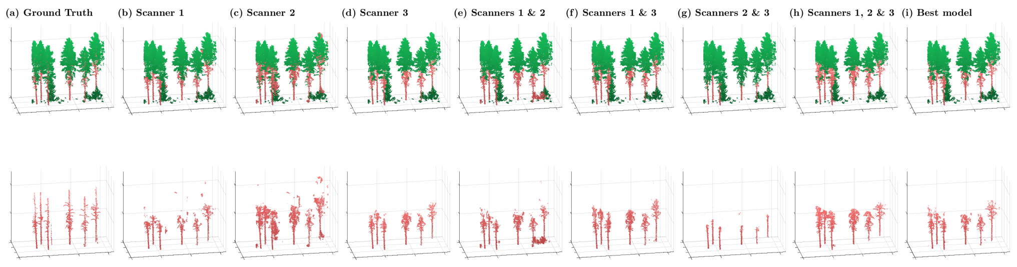


Figure A3: Qualitative results of GrowSP-ForMS for each scanner and all possible 2 and 3 scanner combinations on a subsection of the **test split** of plot #1 in our multispectral data set. The top row shows the full segmentation results and the row below contains only points classified as wood. In (b)–(h) the titles list which scanners were used to achieve the segmentation results. (i) Segmentation results for the best parameter combination, which uses reflectance features from scanner 1 as input for the feature extractor and all three scanners for semantic primitive clustering and superpoint merging.

model in terms of test set mIoU also generates visually the highest quality segmentation results when compared to other scanner combinations.

B Additional Quantitative Performance Comparisons

In this section we list the class specific accuracies of each of the best leaf–wood separation models presented in Section 5.2.1 in the form of confusion matrices. Figures B1 and B2 show the confusion matrices of the supervised baselines on the training and test split of our proposed multispectral data set respectively. Similarly, Figures B3 and B4 show the corresponding metrics for our unsupervised baselines and GrowSP-ForMS on the two splits. The class-level accuracies are consistent with the IoUs listed in Tables 5 and 6. That is, the confusion matrices do not provide much additional information, since lower class-level accuracies correspond with lower class-level intersection over union values and vice versa. Nevertheless, the metrics effectively demonstrate that the high amount of wood points misclassified as foliage is the most significant contributor to the low IoU values each model achieved with the wood class.

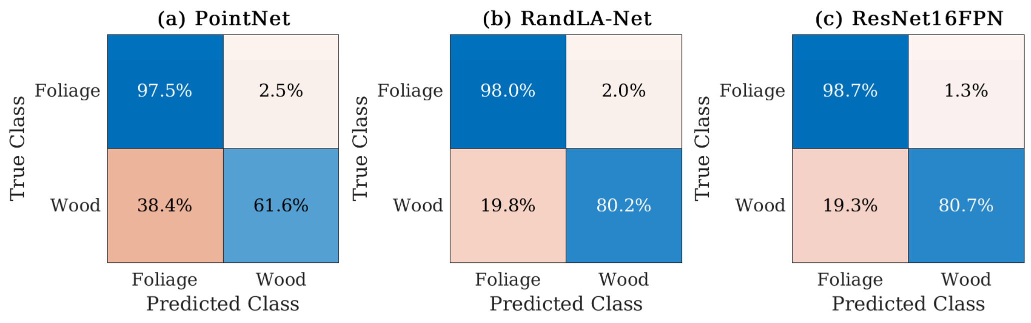


Figure B1: Confusion matrices of the best supervised models on the **training split** of our proposed multispectral data set. (a) PointNet [15]. (b) RandLA-Net [138]. (c) ResNet16FPN [79].

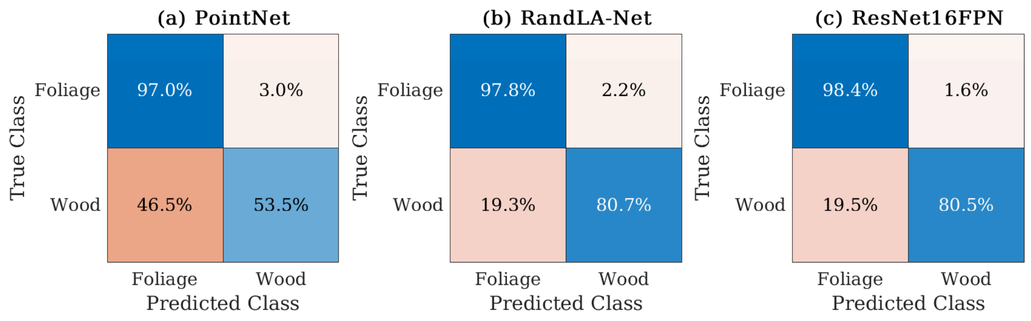


Figure B2: Confusion matrices of the best supervised models on the **test split** of our proposed multispectral data set. (a) PointNet [15]. (b) RandLA-Net [138]. (c) ResNet16FPN [79].

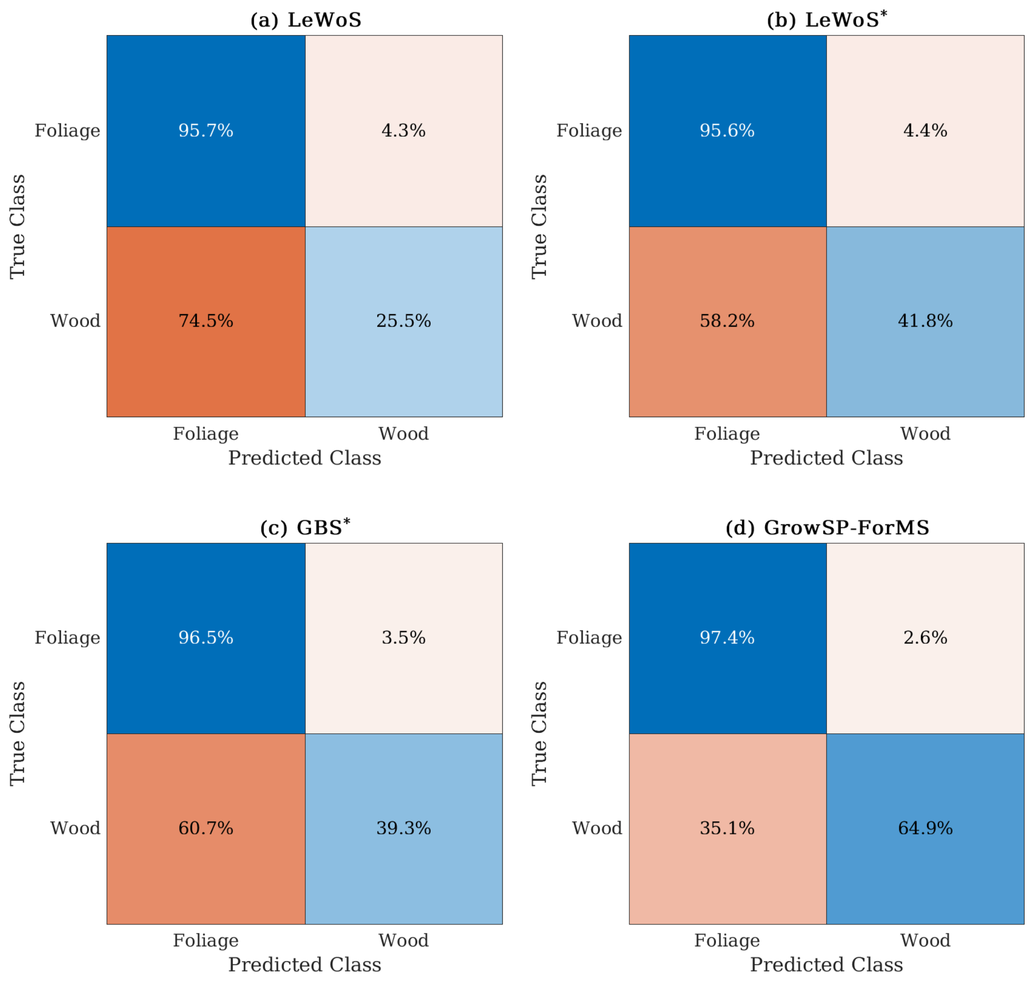


Figure B3: Confusion matrices of the best unsupervised models on the **training split** of our proposed multispectral data set. (a) LeWoS. (b) LeWoS when individual trees are used as input. (c) GBS. (d) GrowSP-ForMS. (*) Results when the inputs are individual trees. These accuracy metrics are not strictly comparable to others due to the different data format.

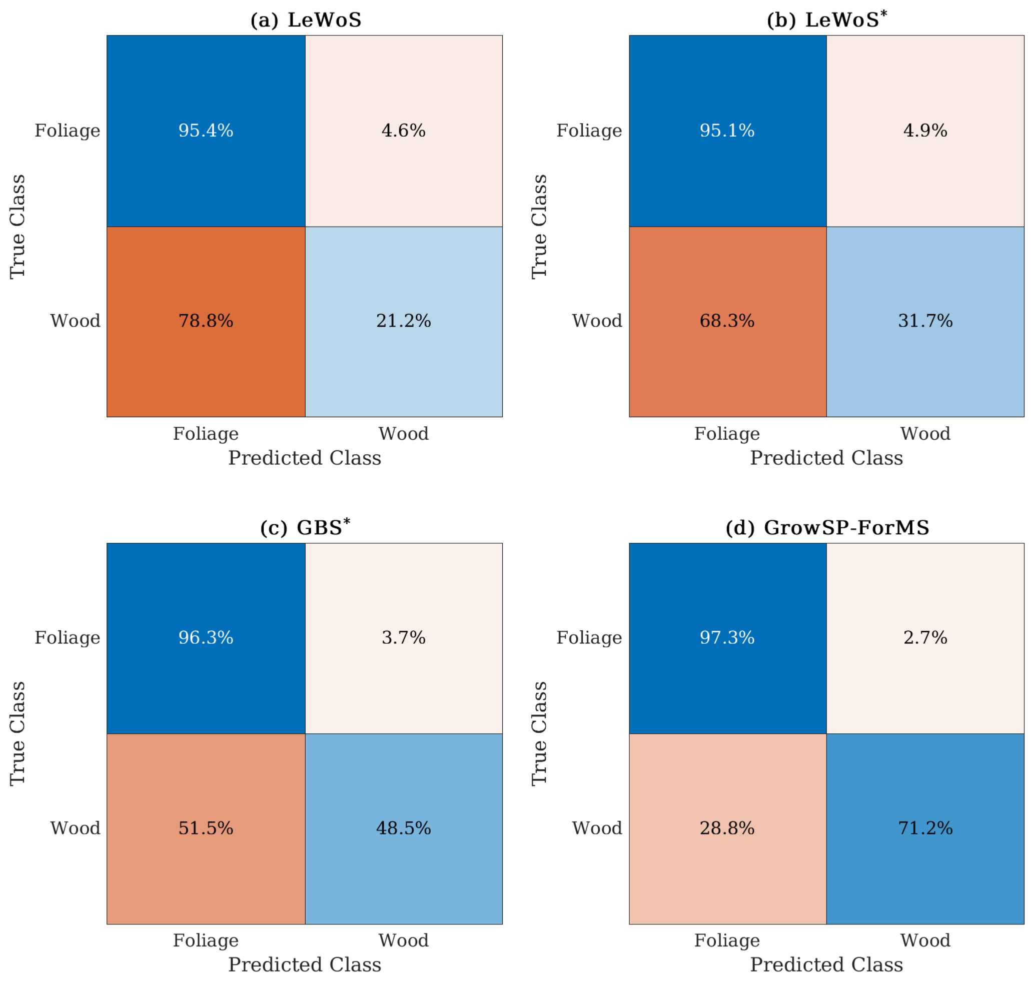


Figure B4: Confusion matrices of the best unsupervised models on the **test split** of our proposed multispectral data set. (a) LeWoS. (b) LeWoS when individual trees are used as input. (c) GBS. (d) GrowSP-ForMS. (*) Results when the inputs are individual trees. These accuracy metrics are not strictly comparable to others due to the different data format.