LiDAR attribute based point cloud labeling using CNNs with 3D Convolution Layers

Miguel Díaz-Medina[a], José M. Fuertes, Rafael J. Segura-Sánchez, Manuel Lucena and Carlos J. Ogayar-Anguita

Department of Computer Science, University of Jaén, 23071, Spain. E-mails: {mdmedina, jmf, mlucena, rsegura, cogayar}@ujaen.es

ARTICLE INFO

Keywords:
CNN
Edgeconv
Point Cloud
LiDAR

ABSTRACT

The goal of this work is to study how semantic segmentation techniques can be adapted to use LiDAR point clouds in their original format as input. Until now, almost all the methods that have been proposed to apply deep learning models on point clouds keep their focus on generic point clouds, making use only of geometric or color information.The deep learning architecture proposed in this work acts as an alternative to classical convolution models with other convolutional operators, applied to 3D data, such as EdgeConv, which use their nearest neighbors to extract local features. We show the results of an experimentation that reveals the influence of additional LiDAR information channels on the performance of the neural network. Unlike other related works, this one is based in the study of semantic segmentation applied to outdoor environments.

CRediT authorship contribution statement

Miguel Díaz-Medina: Research, coding, experiments, data analysis, paper writing . **José M. Fuertes:** Research design, data analysis, paper writing . **Rafael J. Segura-Sánchez:** Research design, experiments, data analysis, paper writing . **Manuel Lucena:** Research design, experiments, data analysis, paper writing . **Carlos J. Ogayar-Anguita:** Research design, data analysis, paper writing .

ORCID(s): 0000-0003-2577-323X (M. Díaz-Medina); 0000-0001-6624-4102 (J.M. Fuertes); 0000-0002-3075-6963 (R.J. Segura-Sánchez); 0000-0002-5546-3745 (M. Lucena); 0000-0003-0958-990X (C.J. Ogayar-Anguita)

1. Introduction

In recent years, point clouds have become the default format for representing three-dimensional data captured from real environments. It is an apparently simple representation format, unlike others such as triangle or quad meshes that require advanced 3D modeling software for editing. In short, a point cloud is a collection of points or elements of information that relate positions in the plane or space with properties of those locations, such as color, surface reflectivity (intensity), etc. There are various techniques for generating and/or capturing point clouds. One of the most widespread is photogrammetry, which consists of taking multiple images over a three-dimensional environment and extracting correspondence points based on the position of the camera. However, the most widely used technology for point cloud acquisition is LiDAR (Light Detection And Ranging). It is a very specialized technology and much more powerful than traditional scanners. In addition to capturing geometric information efficiently and precisely, it also obtains additional information about each laser bounce, such as the color of the surface, intensity, number of returns, radiometric information, etc. LiDAR data follows the specification of the same name, which has become a de facto standard. These types of scanners generate much denser point clouds with more information than traditional scanners.

With the appearance of the first methods based on deep learning for the intelligent processing of geometric information, numerous research works have faced the issue of transferring the classic problem of semantic segmentation in images to the world of point clouds Qi et al. (2016, 2017); Thomas et al. (2019); Wang et al. (2018). However, this extrapolation is not immediate since there are significant differences between both data models. Unlike images, a point cloud is not represented by a regular format that is directly accepted by a deep learning model.

On the other hand, while in an image the connection between pixels is inherent to the format, since a pixel is related to its 8 closest neighbors, in a point cloud we do not have any type of connection and/or relationship between points, neither intrinsically nor extrinsically. In addition, the amount of information that accompanies each piece of information is different. The first approaches that emerged, such as Qi et al. (2016, 2017), carry out the processing using only the geometric information of each point, or sometimes accompanied by information that can be captured by any type of scanner, such as RGB color. However, the LiDAR scanner is capable of capturing much more information from the environment than a conventional scanner, up to 18 different attributes in its traditional version such as, for example, the intensity channel, which measures the reflectivity of the surface of materials, something crucial to distinguish a shiny material from a matte material, among other uses.

From the point of view of pattern recognition, much of the power of LiDAR data has been wasted, since today we do not have fine-tuned methods for the full use of this source of information. Traditionally, the performance of deep learning models has been measured solely by metrics obtained using geometric channels. On the other hand, there is abundant literature on different techniques for the intelligent processing of point clouds Hu et al. (2020); Liu et al. (2019); Qi et al. (2016, 2017); Wang et al. (2018); Wu et al. (2019). Initially, in first works such as Qi et al. (2016) it

was assumed that the points in the cloud were independent, and thus they were processed in the forward propagation of the network, obtaining a single feature vector that described the set of the cloud globally. However, this has proven to be inadequate, since each point in the cloud is also described by its local environment, made up of its surrounding neighbors. Meanwhile, it is assumed that there is a natural connection between the points, and it is necessary (unlike images) to define a neighbor query operator that relates each point to its environment. This way, local descriptors are obtained through successive convolutions and pooling to finally obtain a feature vector that describes the cloud, and thus have higher quality information before propagating the matrices to the classification layers of the network.

Taking all of the above into consideration, the main contribution of this work is to prove the influence of the incorporation of additional information channels (e.g. color and intensity) to the geometric information commonly used in the different semantic segmentation models on a deep network architecture. We use the following methodology:

- A neighbor query method based on kNN (k Nearest Neighbors) is defined.
- On the basis of a well-known 3D data processing method, An adaptation LiDAR data segmentation is proposed. In addition to using geometric information, it also incorporates other information channels provided by the LiDAR scanner.
- Taking the Semantic 3D dataset Hackel et al. (2017) as the starting data source, it is shown that the use of additional information channels (such as intensity) significantly improves the segmentation results compared to using only the geometric positions of points (see Figure 1).

2. Previous work

Point cloud segmentation is defined as the process of classifying point clouds into multiple homogeneous regions, where points in the same region share the same properties. In the case of semantic segmentation, the property shared by all the points in a sub-region is of a semantic nature which makes sense to humans, such as belonging to the same category (for example: building). Point cloud segmentation is challenging due to the high point redundancy, uneven density, and lack of natural structure present in the data. Due to those issues, current segmentation systems achieve acceptable accuracy using only geometric information that is explicitly present in the point cloud. On the other hand, with the emergence of LiDAR technology as the preferred device for capturing point clouds, additional information is available in the datasets. That information could be incorporated into segmentation systems as meta-information for geometry, in order to improve the precision of current systems.

In the literature, we can find several types of techniques for the segmentation of point clouds. There are mainly two groups: classical techniques that use only geometric methods and attributes to deal with input point clouds Grilli et al. (2017); Nguyen and Le (2013), and techniques that make use of deep learning to work with geometric models Liu et al.

(2019); Qi et al. (2017); Thomas et al. (2019). As for the first, these techniques only make use of geometric attributes and do not apply any type of knowledge in their application. Mainly, four techniques are distinguished in this group: techniques based on edge Sappa and Devy (2001), techniques based on region growth (top-down, bottom-up) Besl and Jain (1988), techniques based on adjustment of models Bosché (2012); Fischler and Bolles (1981), or techniques based on unsupervised clustering Grilli et al. (2017). The interest regarding the applicability of these techniques has been diminished, since today new approaches based on deep learning are available. Unlike classical techniques, the second group of techniques is built on top of algorithms that extract knowledge from data and exchange the expert-driven approach for a data-driven variant. Classification decisions are made based on patterns extracted from the data and not on the knowledge of the expert. This is an advantage since in many work scenarios it is difficult to define a knowledge representation that fits this data format.

As stated before, there are some techniques based on deep learning that are capable of configuring their own knowledge representation without any type of intervention by the expert, beyond the provision of labeled data on which the neural network can find a correspondence that match each input element to an output tag Liu et al. (2019). In this sense, proposals from Qi et al. (2016, 2017) stand out as first works in learning in the context of geometric deep learning. They use three-dimensional data in its original format without the need to perform a transformation on the data model and thus keeping all its features. The format of the data on which these models will be applied is specific, since they are three-dimensional point clouds. Unlike other regular data formats, such as an image, there is no regular finite structure that fits over the points of a cloud since 3 properties are fulfilled: high redundancy, lack of natural structure and lack of order. In Wang et al. (2018) a specific line of action is established to apply deep neural approaches on data on which there is no natural structure, so that convolution operations can be applied for feature extraction.

The usual approach with deep learning defines a convolution operator that takes into account the nearest neighbors of each point Mahdaoui and Sbai (2020); Wang et al. (2018). In this way, each element of the kernel covers a finite set of points that are related by their spatial proximity. The point on which the neighbor query is performed will be the point that will add local features of its neighbors through the convolution operator. This point serves as the kernel centroid, so that each point can obtain a local descriptor based on its nearest neighbors. Related to this, the work from Wang et al. (2018) defines different neighbor query approaches and a convolutional network whose basic element is known as EdgeConv. The basic principle of this work is that points belonging to the same semantic category may be far apart in Euclidean space, but the distance between them will be minimized in feature space. Therefore, features are extracted convolutionally and hierarchically, so that the deeper the network, the more complex the features extracted and also the information that accompanies each point. In this way, a kernel is defined to regularize the input data based on the neighbor query. In Li et al. (2018) a similar procedure is used, except that it infers a transformation matrix that

is applied to the input data to save the invariance to geometric transformations of that input data. However, both works omit all the additional information that can provide the LiDAR technology.

Also noteworthy is the work of Thomas et al. (2019), which proposes two versions of a fixed-size kernel to process a full point cloud using a sliding-window approach. Those kernels consist of a rigid version, where the location of the points within the kernel does not change, and a deformable version, where the points in the kernel are able to adapt their positions to fit the processed surface. This increases the accuracy of the model by having a kernel that takes the form described by the points from which local features are extracted. Similarly, this model is applied to basic point clouds, and it does not take into account any additional features of LiDAR data.

The work from Hu et al. (2020) proposes a learning model designed to work with massive point clouds, by using a method that is quite similar to previous ones. However it introduces a random sampling method that simplifies the processing of massive point clouds (several tens of millions of points). Different variants of algorithms are studied to perform a sampling on the original point cloud. After analyzing supervised variants based on learning versus purely algorithmic unsupervised variants, it is concluded that random selection covers all the desired aspects for the sampling algorithm. Starting from the already sampled point cloud, an approach similar to that used in Wang et al. (2018) is applied in Feng et al. (2019); Dos Santos et al. (2016). They propose to add a supervised component in feature selection trough the use of Attentive Pooling. However, once again, no additional LiDAR data is used for improving the precision of segmentation systems.

A first approximation is made in Biasutti et al. (2019). However, it only projects the point cloud onto a plane instead of applying convolution approaches directly to the points in their original format. This brings the segmentation problem to the image domain instead of studying it from a Geometric Deep Learning perspective. Despite the fact that previous works propose promising approaches to overcome the irregularity and lack of natural structure of point clouds, all of them carry out learning using only the position of each point in space, that is, the coordinates (x, y, z) of each point. The advancement of capture technology in recent years has led to the appearance and subsequent evolution of scanners such as LiDAR, which are capable of capturing much more information from the environment than a traditional scanner would. LiDAR data continues to be used in the same way (generated with scanners or photogrammetry), and so the additional information contained in it continues to be wasted. Therefore, it would be convenient to study the performance improvement that can be achieved by incorporating some of the information channels that can be obtained using LiDAR. As an analogy, it would be the equivalent of classifying images using only grayscale snapshots, given the quality of current cameras.

3. Description of the method

In our proposal, the processing pipeline consists of starting from an unstructured data source, the 3D point cloud, and finally obtaining a label associated with each point as a result of the semantic segmentation. As discussed above, point clouds are irregular and unstructured data models, as opposed to the input data format required by a neural model. Therefore, our first step towards semantic segmentation consists of performing a preprocessing on the input dataset to prepare it for the subsequent training and validation phases in the neural models. Since a neural network always needs to receive data of the same nature, we need to regularize the point cloud. Thus, we will divide the cloud into blocks with the same number of points. Each block serves as an input item to the subsequent neural model. At the end, each point will be labeled in the corresponding semantic category.

The neural network used is inspired by the work of Wang et al. Wang et al. (2018). It starts from the basic idea of building a feature graph. The premise is that points belonging to the same semantic category also share features. A (dynamic) feature graph will be built several times throughout the propagation in the network and as features are extracted. In this way, the points belonging to the same category would be geometrically distant in the initial Euclidean space $\mathbb{R}^3$, but in a feature space of, for example, dimension $\mathbb{R}^{64}$, the points would tend to be close since their features are very similar.

Finally, once the points have been grouped in the feature space by their similarity, a classification layer is applied. It will divide the high-dimensional space into the different classes defined by the problem, obtaining as a result the semantic segmentation associated with the input block.

3.1. Data preprocessing

The point clouds belonging to the dataset Semantic3D Hackel et al. (2017) correspond to really dense scans of outdoors. In order to be able to manipulate them efficiently, it is necessary to carry out a sampling process that selects a sufficiently significant and representative sample of the input cloud, to preserve the greatest amount of information in the smallest possible volume. In this case, random sampling is chosen, since it is the simplest and most efficient at execution time, based on the work of Hu et al. (2020). The RGB channels are normalized to the interval [0, 1] using a max-min normalization. This will facilitate the learning of the network Goodfellow et al. (2016), and will improve the speed of convergence of the model.

3.1.1. Block generation

In each forward propagation of the network, a given set of blocks (batches) will be processed, each of them composed of a pre-established number of points. Unlike voxelization-based approaches, the points are contained in their original format and no discretization has been performed on them. To generate the blocks, it is necessary to divide each cloud into blocks of fixed size considering the horizontal plane of the cloud, and extract points from each block

following a sliding window approach. That is, to define a grid of fixed size and move it along and across the horizontal plane of the point cloud, extracting subsets of points of the same dimensions. If the number of points contained in a block is less than desired, it will be necessary to carry out a sampling and replacement process to replicate the existing points as many times as necessary to achieve the desired number. This will not affect network performance.

It must be noted that the sliding window approach is applied on the X and Y axes, considering the Z axis as the vertical axis or perpendicular to the surface of the earth. Therefore, there will be blocks with a high variation in the Z axis, and blocks with a minimum variation, although the study of this variation is not a matter of interest Hackel et al. (2017); Hu et al. (2020); Wang et al. (2018); Thomas et al. (2019). The blocks must have a fixed area (e.g. $1m^2$ or $4m^2$). As blocks are generated, they have to be stored in a structure common to all processed point clouds, along with the class label associated with each point. If the number of points contained in a block is higher than desired, a sampling without replacement will be applied to select a sample of size equal to the desired number. If this number is less, what is indicated immediately before will be carried out.

3.1.2. Generation of training batches

Once all point clouds have been processed, it is necessary to store the blocks taking into account a cross-validation approach that allows evaluating effectively the model accuracy. The experimentation scenarios will be defined based on a k-fold Cross-Validation strategy. In this way, the list of already processed blocks will be divided into k parts, and $2k$ different files will be generated, numbered by $[0, k-1]$ and in training and test versions. The training files will contain $k-1$ parts of the total, and the test file corresponding to this part will contain a single part that will be used for model validation in each epoch.

3.2. Neural network architecture

This section details the design of the neural network to be used (see Figures 2 and 3). The basic idea is the dynamic construction of the nearest neighbor graph after each processing layer in the network. It is a convolutional neural network, because in each convolution step it analyzes and adds information from the closest neighbors to a point. It is done so that a point has no information about itself in the segmentation step, and at the same time knows peculiarities about how its environment is distributed.

In Figure 2 we can see a complete neural architecture where the input is a matrix (n, F) with a total of n points and F features for each point (so, each point is described by a feature vector of dimension F), and the output is a matrix (n, p), where p is the possible number of classes in the dataset. The neural network is divided into several modular blocks, EdgeConv Wang et al. (2018), that constitutes the core component of this architecture. This block is responsible for extracting the local characteristics of the neighborhood of each point and will be described in the next section.

On input, the network receives points in their raw format, with the features defined in the training dataset. Local

features are extracted at three different levels of abstraction, the EdgeConv blocks (see Figure 3). The deeper we dive into the neural network architecture, the more abstract the point features become. And furthermore, the perceptive field of the EdgeConv operator grows as we propagate through the network. In the first block, the features of the closest points are added with the features of each center point. In the second block, the features of the point already contain information about its neighbors in the previous application of EdgeConv, so in the second application of that operator, we indirectly query features about the neighbors of the neighbors of the center point.

Finally, once we obtain information at three different levels, we have 3 matrices of $(n, 64)$, we concatenate them and we get an single matrix of $(n, 192)$, where each point in the initial matrix is described by local features of three different levels of abstraction. We apply a MLP on this matrix to get a global feature vector, which describes all local neighborhoods in the input block. After that, this global vector is concatenated to the features of each point, obtaining a matrix of features $(n, 1216)$. To obtain the segmentation scores of each point, a shared MLP is applied on this matrix, formed by 4 layers of 256, 256, 128 and p neurons respectively, finally obtaining a matrix (n, p) where points are already semantically segmented, being p is the number of classes in the dataset.

3.2.1. EdgeConv

It constitutes the core on which this network architecture is built, and consists of a few key steps (see Figures 3 and 4):

- **Input**. As stated before, the network takes as input a matrix of points (n, F), with a total of n points and F features for each point, so each point is described by a feature vector of dimension F. The number of points does not change at any layer of the network, however the features that describe them do. In the most basic approach, this matrix will be composed of the n input points of the cloud, and for each of them we will have the three coordinates (x, y, z), the intensity, the color, etc. When this input matrix is received, the feature graph is computed using the k nearest neighbors of each point.

- **Features graph**. Let $P = \{p_i, ..., p_n\}$ be a fixed-size input block consisting of a subset of the original point cloud. For each input point, its nearest neighbors are calculated using the Euclidean distance and considering the points embedded in $\mathbf{R}^F$, being F the number of features for each point $p_i \in P$. The Nearest Neighbors algorithm is applied to find the nearest points to an initial point p_i. Let $kNN(p_i) = \{p_i^1, ..., p_i^k\} \subseteq P$ a set of the nearest neighbors for p_i. The features of the input point p_i are added to the features of each neighbor, $\{p_i^1, ..., p_i^k\}$. For each neighbor, the number of features will be doubled, obtaining a concatenation of the features of the original point and the result of the difference between the features of the original point and the features of the neighbor, like:

$$p_i^{j'} = (p_i, p_i - p_i^j) \forall j \in [1, k]$$

The above operation will compute the edges of the feature graph as the difference between the features or locations of each pair of points in a feature space. We are not interested in the absolute position of the neighbors, but rather the relative position of the neighbors with respect to the original point.

As this process is repeated for all points in a block, each point will change from being described as a single set of features, to being described by a set of aggregated neighbor features. So the input matrix (n, F) will be resized to $(n, k, 2F)$.

- **MLP (Multilayer Perceptron)**. Once the nearest neighbor graph has been calculated, it is necessary to apply the intelligent logic for the identification and search of patterns. It is a question of inferring new features from the combined features of the origin point and neighbors, that is, the feature graph. In order to maintain the philosophy of Qi et al. (2016), and achieve the shared multilayer perceptron behavior, this phenomenon is implemented using convolution layers, regardless of the deep learning framework to be used. The feature that distinguishes convolution layers from traditional MLPs is the fact that the kernel can be shared over the input data. Thus, the kernel size will be 1 so that only one input sample is considered at a time. In this way, the number of output points will be equal to the input. By adjusting the number of filters (kernels used), the number of output features is achieved. Consequently, for a graph of input features of dimensions $(n, k, 2f)$, an output matrix of dimensions (n, k, a_n) is obtained, where the last term is the number of features that the MLP will infer for each point. That term also contains local neighborhood information for each point in the input block. In this work, the number of output features chosen will usually be 64. In some blocks, two MLPs will be used, one followed by another, while the last EdgeConv block will only use one MLP.

- **Pooling**. The last step within an EdgeConv block is the application of an aggregation or reduction layer. This layer will select the most relevant or frequent features from the k neighbors of each point, and will output a single feature vector of size a_n for each point. Thus, the input array will be (n, k, a_n), and the output a matrix of dimension (n, a_n), where a_n is a feature vector for the original input point that summarizes local information of all its neighbors. For each of the a_n features present in all neighbors of a point, it will select the value according to an aggregation mode. In this work max-pooling is used.

In short, the EdgeConv Wang et al. (2018) module receives a dot matrix with its different features, and generates another similar output dot matrix, varying the number and type of features. The input features will be low-level, while the output features will be higher-level abstractions over the first ones, as established by the classical convolution philosophy. The output features will be formed by an abstract aggregation of the features of each input point with the features of its neighbors. The EdgeConv blocks are applied a total of three times in the network, and in this way they provide all the points with local information about their environment, as well as global information from the rest of

the local environments present in the input cloud.

After applying each of the three EdgeConv layers, the output of each layer is concatenated. That is, the output of each layer will be a matrix $(n, 64)$, composed of the points of the input cloud, and 64 features. Meanwhile, after the concatenation along the axis of the features, a new matrix of dimensions $(n, 192)$ is obtained. In this new matrix, each point will contain features at three different levels of abstraction. As the level of abstraction is higher, it contains information about a more global environment of the cloud. After the concatenation of the three matrices, a Multilayer Perceptron-type network is again applied, represented by a 1D convolution layer, and a new matrix of dimensions $(n, 1024)$ is obtained. Each of the points is described, at this moment, by a vector of 1024 features.

The goal is to obtain a global feature vector, as Qi et al. (2016) does in its architecture. To do this, a pooling layer is applied that will select the 1024 most frequent features at all points. That is, given the features matrix $(n, 1024)$, where each column represents a different feature and each row a different point, it will select the most representative value (max-pooling) of each feature, obtaining a new global feature vector of size 1024. Next, for each point, this feature vector is added to the three previously concatenated matrices. In this way, a new matrix $(n, 192 + 1024) = (n, 1216)$ is obtained, and thus, each point now has both local information about its neighbors, and also global information about the whole point cloud.

3.2.2. Classification layers

Once the feature extraction step is complete, it is necessary to complete the semantic segmentation process. It consists of assigning a semantic label to each of the entry points. For this, MLPs will be applied, encoded as 1D convolution layers, in order to apply a shared MLP to the whole entry point set. A total of three layers are applied, successively reducing the dimensionality of the feature space, migrating from a feature matrix $(n, 1216)$ to a label matrix (n, p), where p represents the number of classes of the dataset, or the number of distinguishable semantic categories in the segmentation problem.

3.2.3. Permutation invariance

It is usual to use some data augmentation technique that allows increasing the generalization capacity of the network. This process consists of slightly modifying the input data, so that the data maintains the probability distribution of the complete dataset, but varies enough to teach the network different perspectives on the same item of information. This increase the generalization capacity of the network.

In this work, the data augmentation consists of randomly altering the order of the points contained in each of the clouds that the network will have to process. In this way, the network will not memorize the order of the points to assign a label, but will instead obtain permutation invariance, something essential for the treatment of any point cloud. That is, the segmentation result should be the same regardless of the order in which the network is fed with the different

Table 1
Parameters for the generation of the training dataset.

Parameter	value
Partitions K Fold	5
Block size	1.0 m
Points per block	4096
Minimum of points per block	1024
Stride	1.0

points contained in the cloud.

4. Experiments and results

A series of experiments are carried out to test the influence of the different information channels obtained by LiDAR with regard to the exclusive usage of the geometric data. The goal is to test the hypothesis that the additional attributes obtained by LiDAR improve the accuracy of traditional systems that only make use of the geometric information. For this, the Semantic 3D dataset Hackel et al. (2017) is used, constituted in its training partition by 15 massive point clouds whose points are labeled in 8 semantic categories (see Figure 1).

4.1. Dataset

The Semantic3D dataset Hackel et al. (2017) is used to test the architecture, as well as the influence of the different information channels. A training and validation dataset is generated following the configuration described in Table 1.

Each point has 10 features; the coordinates (x, y, z), the intensity i, the color channels (r, g, b) normalized to the interval $[0, 1]$, as well as the coordinates (x_n, y_n, z_n) normalized according to the block to which each point belongs. In addition, the class to which it belongs is stored next to each point. It is an integer $[0, 7]$ that describes the 8 classes available in this data set. The list of classes considered is shown in the Table 2. As can be seen, it does not exactly match the description indicated in Hackel et al. (2017), since class 0 (unclassified points) has been discarded, and a new numbering has been defined by subtracting the value 1 from the identifiers of the rest of the classes to normalize them to the interval $[0, 7]$.

4.2. Definition of parameters

All experiments share some general parameters. These common parameters are described in Table 3. The main difference between all the experiments is the features contained in the input dataset, that is, the usage of different combinations of LiDAR information channels to serve as input to the network. A total of 6 experiments are proposed. The initial hypothesis implies that the influence of some channels provided by the LiDAR scanner improves the semantic segmentation process. Therefore, the experiments are performed adding data channels incrementally.

Table 2
List of classes for semantic segmentation.

Class	Identifier
Artificial terrain	0
Natural terrain	1
High Vegetation	2
Low Vegetation	3
Building	4
Hard scape	5
Scanning artifacts	6
Vehicles	7

Table 3
Overall training parameters.

Parameter	Value
Learning rate	0.001
Epochs	100
Nearest neighbors	20
Scheduler	True
Optimizer	Stochastic Gradient Descent (γ=0.9)
Batch size	2
Word vector size	1024

Table 4
Overall training combinations of information channels.

Experiments	Features
1	geom (x, y, z)
2	geom (x, y, z) + intensity
3	geom (x, y, z) + color (r, g, b)
4	geom (x, y, z) + intensity + color (r, g, b)
5	geom (x, y, z) + norm. geom
6	All data

Table 5
Intersection over Union obtained for each class in all the experiments.

Experiments	Avg	IoU_0	IoU_1	IoU_2	IoU_3	IoU_4	IoU_5	IoU_6	IoU_7
1 (xyz)	0.69	0.95	0.00	0.24	0.00	0.64	0.02	0.00	0.37
2 (xyz+int)	**0.83**	0.93	**0.50**	**0.80**	0.28	**0.86**	0.19	0.05	**0.66**
3 (xyz+rgb)	0.74	**0.98**	0.08	0.45	0.14	0.61	0.07	0.01	**0.68**
4 (xyz+int+rgb)	0.81	0.91	0.41	0.80	0.38	0.83	0.18	0.07	0.63
5 (xyz+xyz_{norm})	0.33	0.59	0.00	0.00	0.00	0.08	**0.47**	0.00	0.00
6 (all)	**0.85**	0.95	**0.50**	**0.86**	**0.40**	0.85	0.10	**0.30**	0.34

For the validation of the experiments, a cross-validation scheme with $K = 5$ is used. The proposed experiments are described in Table 4.

4.3. Results

The Intersection over Union (IoU) results are collected in the 8 classes defined in the training set in Table 5. As can be seen, the experiment that uses all the available information channels dominates the rest of the experiments. It achieves 2% more precision than the second best result (using only intensity). Therefore, this shows us that the intensity channel is essential to perform a good semantic segmentation of the point cloud. While properties such as color (r, g, b) can be obtained using conventional scanners, or photogrammetric techniques such as Sfm (Structure from motion), the intensity channel is unique to LiDAR scanners. On the other hand, the small difference in the accuracy associated with experiment 2 and experiment 6 could be due to the random initialization of the trainable parameters of the neural network. That is why a 2% higher result is achieved in the second case. Whatever the case, all the configurations are capable of identifying artificial terrain with a success rate of over 90%, since they are the simplest geometric structures.

Table 6
Precision vs. loss comparison of the experiments.

	Training		Validation	
Experiment	Loss	Precision	Loss	Precision
1 (xyz)	1.39	67.90%	1.55	55.80%
2 (xyz+int)	1.13	**87.67**%	**1.29**	77.40%
3 (xyz+rgb)	1.25	77.89%	1.50	62.36%
4 (xyz+int+rgb)	1.13	87.66%	1.33	75.84%
5 (xyz+xyz_{norm})	1.26	76.44%	3.42	45.32%
6 (all)	**1.08**	**89.37**%	**1.29**	**77.90**%

To conclude the comparative analysis of the different experiments, Table 6 reflects the average precision obtained in each experiment. The result is the average of the average IoUs in each partition. It is noticeable that the results associated with experiment 6, which uses all available information channels, dominate the rest in the 4 assessment parameters. In addition, the exclusive use of geometric channels significantly worsens the performance of the trained models, whether they are raw geometric channels or normalized geometric channels according to the coordinates of each block. However, it is appropriate to take into account that the difference between the experiments that only uses the geometric channel with respect to the use of all the channels is minimal. It is necessary to find a trade-off between maximizing the accuracy obtained and minimizing the resources required to train and test the models. The amount of resources (memory, processing) needed to train a model using only the intensity channel is significantly lower than the resources needed to train a model using all the information.

Assuming that the experiment that provides the best results is the one that uses all the information channels, the confusion matrices related to the last training epoch in each of the 5 validation partitions are shown in Tables 7 to 11. The training results are ignored, since they do not measure the generalization capability of the neural model.

5. Conclusion

The goal of this work is to study how semantic segmentation techniques can be adapted to use LiDAR point clouds in their original format as input. One of the most successful techniques for semantic segmentation of point clouds Wang et al. (2018); Qi et al. (2017); Thomas et al. (2019); Hu et al. (2020) has been chosen. The main problem is that convolutional models do not adapt naturally to this type of data. Meanwhile, it is necessary to establish a neighbor query operator that allows local characteristics to be extracted from each point. Thus, the chosen proposal makes use of EdgeConv, a convolutional operator that uses the nearest neighbors to extract local features.

A set of experiments is proposed to evaluate the influence of additional LiDAR information channels. Also, the use of both intensity and color are studied for assessing their combined influence on the quality of the results obtained. It is observed that the exclusive use of geometric channels produces worse results compared to the incorporation of additional channels. Specifically, the intensity channel represents a remarkable improvement with respect to the rest of the channels, including the color.

Finally, it is observed that the most remarkable result is obtained using all available information channels: intensity, color and normalized coordinates. However, the improvement is negligible (2% more, 89.37% in training) compared to using only intensity (87.67% in training). In particular, we achieved up to 86% accuracy on average for the high vegetation class using only the color, geometry and intensity channels. Accuracy barely reaches 40% using only geometry. This parameter is of great interest for numerous tasks related to remote sensing, since vegetation constitutes one of the most important elements that can be captured. The exclusive use of intensity results in an average accuracy of 77.40% in validation, while the incorporation of all channels increases this percentage to 77.90%. Thus, it is concluded that the intensity channel is promising to start a study of its influence on other datasets of a different nature.

6. Code availability

The data and source code with the hyperparameter setup and different approaches analyzed in this study are openly available at https://github.com/mdiazm/lidar-labeling-using-3d-cnns for extension and replication purposes.

References

Besl, P.J., Jain, R.C., 1988. Segmentation Through Variable-Order Surface Fitting. IEEE Trans. Pattern Anal. Mach. Intell. 10, 167–192.

Biasutti, P., Bugeau, A., Aujol, J.F., Brédif, M., 2019. RIU-Net: Embarrassingly simple semantic segmentation of 3D LiDAR point cloud. CoRR abs/1905.0. arXiv:1905.08748.

Bosché, F., 2012. Plane-based registration of construction laser scans with 3D/4D building models. Advanced Engineering Informatics 26, 90–102.

Dos Santos, C., Tan, M., Xiang, B., Zhou, B., 2016. Attentive Pooling Networks. CoRR abs/1602.0.

Feng, M., Zhang, L., Lin, X., Gilani, S.Z., Mian, A., 2019. Point Attention Network for Semantic Segmentation of 3D Point Clouds. arXiv:1909.12663.

Fischler, M.A., Bolles, R.C., 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Commun. ACM 24, 381–395.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press.

Grilli, E., Menna, F., Remondino, F., 2017. A review of Point Clouds segmentation and classification algorithms. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLII-2/W3, 339–344.

Hackel, T., Savinov, N., Ladicky, L., Wegner, J.D., Schindler, K., Pollefeys, M., 2017. SEMANTIC3D.NET: A new large-scale point cloud classification benchmark, in: ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, pp. 91–98.

Hu, Q., Yang, B., Xie, L., Rosa, S., Guo, Y., Wang, Z., Trigoni, N., Markham, A., 2020. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition .

Li, J., Chen, B.M., Lee, G.H., 2018. SO-Net: Self-Organizing Network for Point Cloud Analysis.

Liu, W., Sun, J., Li, W., Hu, T., Wang, P., 2019. Deep Learning on Point Clouds and Its Application: A Survey. Sensors 19.

Mahdaoui, A., Sbai, E.H., 2020. 3D Point Cloud Simplification Based on *k*-Nearest Neighbor and Clustering. Advances in Multimedia 2020, 8825205.

Nguyen, A.T.L., Le, H.B., 2013. 3D point cloud segmentation: A survey. 2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM) , 225–230.

Qi, C.R., Su, H., Mo, K., Guibas, L.J., 2016. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. arXiv preprint arXiv:1612.00593 .

Qi, C.R., Yi, L., Su, H., Guibas, L.J., 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space, in: Advances in Neural Information Processing Systems 30. Curran Associates, Inc., pp. 5099–5108.

Sappa, A.D., Devy, M., 2001. Fast range image segmentation by an edge detection strategy, in: Proceedings Third International Conference on 3-D Digital Imaging and Modeling, pp. 292–299.

Thomas, H., Qi, C.R., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J., 2019. Kpconv: Flexible and deformable convolution for point clouds, in: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).

Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M., 2018. Dynamic Graph {CNN} for Learning on Point Clouds. CoRR abs/1801.0. arXiv:1801.07829.

Wu, W., Qi, Z., Fuxin, L., 2019. PointConv: Deep Convolutional Networks on 3D Point Clouds, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Table 7
Confusion matrix for the first partition.

Ground truth \ Prediction	Artificial terrain	Natural terrain	High vegetation	Low vegetation	Buildings	Hard scape	Scanning artifacts	Cars
Artificial terrain	91.92	0.63	0.10	0.73	6.17	0.24	0.09	0.12
Natural terrain	75.40	8.29	1.44	0.31	14.26	0.29	0.00	0.01
High vegetation	0.04	0.16	93.49	0.84	5.35	0.11	0.00	0.00
Low vegetation	10.72	8.54	42.83	13.14	21.34	2.53	0.49	0.40
Buildings	2.66	0.35	1.58	0.18	93.76	1.36	0.11	0.00
Hard scape	46.09	6.62	5.22	2.18	29.40	7.74	0.08	2.67
Scanning artifacts	39.77	0.28	1.35	7.96	38.33	2.58	8.57	1.16
Cars	36.59	0.23	0.00	3.05	40.78	1.17	0.05	18.14

Table 8
Confusion matrix for the second partition.

Ground truth \ Prediction	Artificial terrain	Natural terrain	High vegetation	Low vegetation	Buildings	Hard scape	Scanning artifacts	Cars
Artificial terrain	95.32	3.29	0.56	0.10	0.29	0.28	0.01	0.16
Natural terrain	21.28	77.20	0.57	0.22	0.54	0.19	0.00	0.00
High vegetation	3.44	0.73	86.71	3.25	4.79	1.08	0.00	0.00
Low vegetation	32.18	2.41	16.73	42.94	5.68	0.06	0.00	0.00
Buildings	1.50	0.21	1.74	0.62	95.31	0.60	0.00	0.01
Hard scape	17.13	12.12	9.88	6.40	45.79	7.75	0.31	0.62
Scanning artifacts	53.73	2.28	0.29	0.68	19.99	0.90	7.43	14.70
Cars	40.89	1.77	0.10	0.02	1.49	3.80	4.68	47.25

Table 9
Confusion matrix for the third partition.

Ground truth \ Prediction	Artificial terrain	Natural terrain	High vegetation	Low vegetation	Buildings	Hard scape	Scanning artifacts	Cars
Artificial terrain	67.20	28.46	1.52	0.09	2.01	0.50	0.00	0.24
Natural terrain	15.70	68.45	4.32	1.06	7.73	2.12	0.00	0.61
High vegetation	0.87	2.47	86.66	0.79	7.54	1.67	0.00	0.00
Low vegetation	0.04	8.73	23.75	36.59	24.37	6.52	0.00	0.00
Buildings	1.54	5.21	1.83	0.72	88.20	0.91	0.87	0.73
Hard scape	6.85	7.66	5.92	1.47	47.11	28.28	2.17	0.52
Scanning artifacts	32.72	3.93	1.58	0.00	35.64	11.09	2.12	12.91
Cars	11.13	0.37	0.00	0.00	26.92	0.09	0.58	60.90

Table 10
Confusion matrix for the fourth partition.

		Prediction							
		Artificial terrain	Natural terrain	High vegetation	Low vegetation	Buildings	Hard scape	Scanning artifacts	Cars
Ground truth	Artificial terrain	97.21	0.65	0.11	0.02	0.25	0.45	0.01	1.31
	Natural terrain	25.62	72.44	0.64	0.12	0.45	0.66	0.00	0.08
	High vegetation	1.30	1.85	88.92	2.45	5.02	0.46	0.01	0.00
	Low vegetation	1.03	0.79	29.62	38.96	22.87	6.66	0.00	0.08
	Buildings	0.32	1.22	0.33	1.45	96.27	0.30	0.00	0.10
	Hard scape	8.57	4.94	6.34	5.54	33.92	31.28	0.02	9.40
	Scanning artifacts	28.04	1.73	0.16	0.00	10.96	6.97	34.66	17.48
	Cars	5.78	0.02	0.01	0.00	6.81	1.35	0.08	85.94

Table 11
Confusion matrix for the fifth partition.

		Prediction							
		Artificial terrain	Natural terrain	High vegetation	Low vegetation	Buildings	Hard scape	Scanning artifacts	Cars
Ground truth	Artificial terrain	92.07	3.64	0.02	0.11	2.92	0.12	0.03	1.09
	Natural terrain	51.00	31.92	4.13	2.15	10.20	0.60	0.00	0.00
	High vegetation	1.57	0.45	83.20	10.53	4.05	0.17	0.03	0.00
	Low vegetation	5.41	13.13	3.12	37.85	33.25	6.98	0.04	0.22
	Buildings	2.27	0.20	0.75	0.53	89.83	2.58	0.18	3.66
	Hard scape	16.86	10.30	5.53	2.28	39.20	11.84	2.79	11.20
	Scanning artifacts	29.28	3.65	0.46	6.98	31.66	0.78	19.27	7.93
	Cars	9.78	4.64	0.56	0.03	11.22	4.13	6.53	63.10

List of Figures

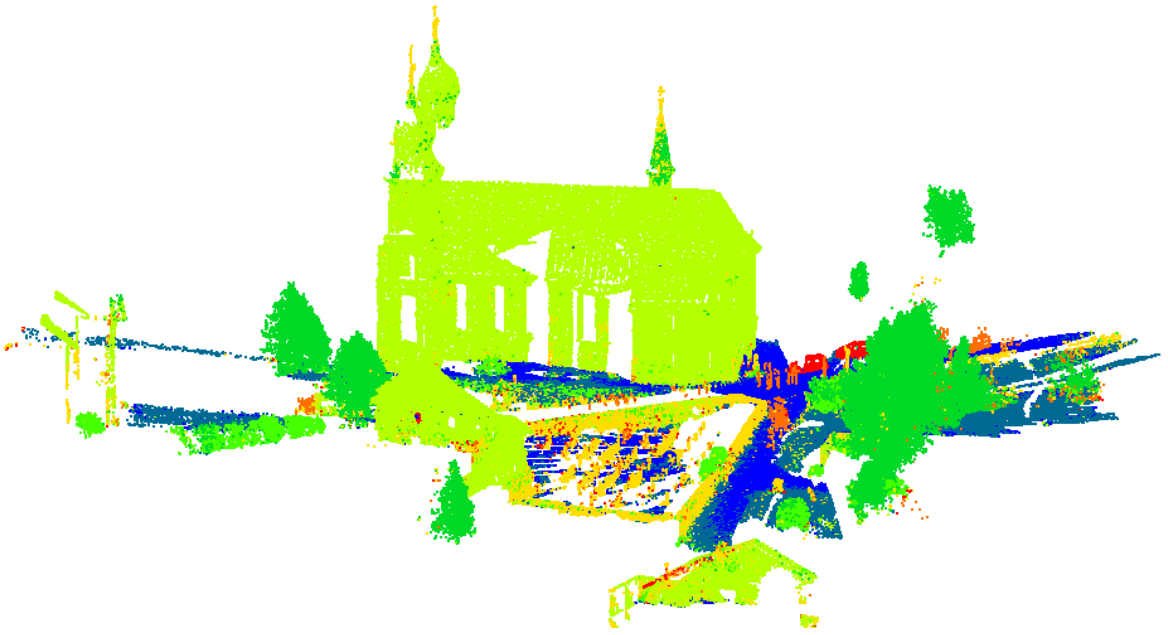

Figure 1: The Semantic 3D dataset.

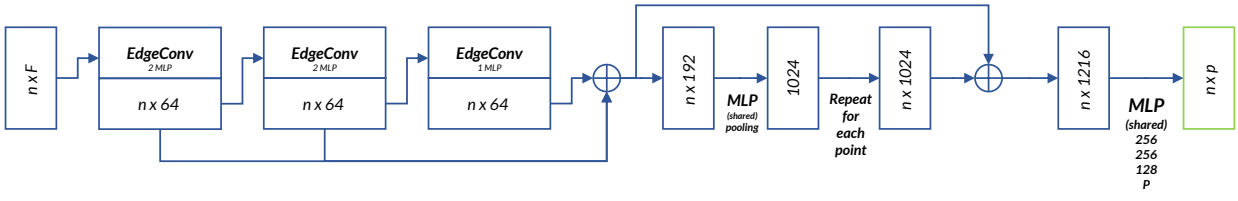

Figure 2: Neural network architecture used.

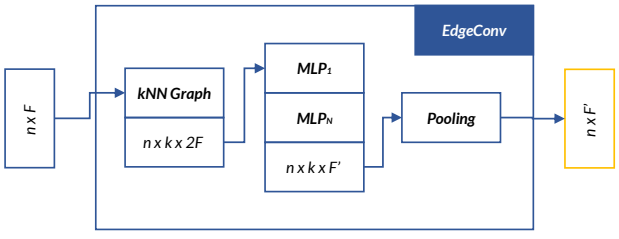

Figure 3: Detail of the EdgeConv layer.

Figure 4: Dynamic graph building using kNN in EdgeConv.