Active and incremental learning for semantic ALS point cloud segmentation

Yaping Lin\textsuperscript{a}, George Vosselman\textsuperscript{a}, Yanpeng Cao\textsuperscript{b}, Michael Ying Yang\textsuperscript{a,}\textsuperscript{*}

\textsuperscript{a} Faculty of Geo-Information Science and Earth Observation (ITC), University of Twente, Enschede, the Netherlands
\textsuperscript{b} State Key Laboratory of Fluid Power and Mechatronic Systems, School of Mechanical Engineering, Zhejiang University, Hangzhou, China

\textbf{ARTICLE INFO}

Keywords: Point clouds, Semantic segmentation, Active learning, Incremental learning, Deep learning

\textbf{ABSTRACT}

Supervised training of a deep neural network for semantic segmentation of point clouds requires a large amount of labelled data. Nowadays, it is easy to acquire a huge number of points with high density in large-scale areas using current LiDAR and photogrammetric techniques. However, it is extremely time-consuming to manually label point clouds for model training. In this paper, we propose an active and incremental learning strategy to iteratively query informative point cloud data for manual annotation and the model is continuously trained to adapt to the newly labelled samples in each iteration. We evaluate the data informativeness step by step and effectively and incrementally enrich the model knowledge. The data informativeness is estimated by two data dependent uncertainty metrics (point entropy and segment entropy) and one model dependent metric (mutual information). The proposed methods are tested on two datasets. The results indicate the proposed uncertainty metrics can enrich current model knowledge by selecting informative samples, such as considering points with difficult class labels and choosing target objects with various geometries in the labelled training pool. Compared to random selection, our metrics provide valuable information to significantly reduce the labelled training samples. In contrast with training from scratch, the incremental fine-tuning strategy significantly save the training time.

1. Introduction

Point clouds, collections of points in 3D space, are characterized by their powerful abilities to represent position, size, shape and orientation of objects. Interpretation of point clouds captured by airborne laser scanning (ALS) systems is an essential step for many applications, such as 3D city modelling and urban land administration. Manually identifying urban objects like buildings, trees, and bridges requires a huge amount of human effort. To reduce this time-consuming and tedious work, researchers put their efforts on investigating the potential of machine learning techniques to deal with point cloud semantic understanding automatically.

Supervised machine learning is the most commonly used technique in point cloud interpretation. It relies on labelled data to train statistical models. A lot of models have been researched for the task of semantic point cloud segmentation, like random Forests (RF) (Chehata et al., 2009), Supported Vector Machine (SVM) (Lodha et al., 2006), Gaussian Mixture Model (GMM) (Weinmann et al., 2014), AdaBoost (Lodha et al., 2007) and Artificial Neural Networks (ANN) (Xu et al., 2014). Recently, deep neural networks made significant breakthroughs in point cloud classification and segmentation tasks, such as PointNet (Qi et al., 2017a) and PointCNN (Li et al., 2018). Although the deep learning paradigm shows its power in complicated feature representation, it requires a massive amount of ground truth data to avoid overfitting during the training. Unfortunately, the ground truth for the semantic point cloud segmentation requires pointwise labelling, which is very time-consuming when done manually. To label ALS point clouds covering 2 km\(^2\) in Dublin center into 8 categories, over 2500 h were spent with an appropriate tutorial, supervision, and careful cross-checks to minimize the error (Zolanvari et al., 2019). Therefore, strategies should be proposed to alleviate such manual annotation effort.

To reduce manual annotation efforts, one possible strategy is to bring labels from other data sources. For example, Yang et al. (2020) directly bring 2D labels from topographic maps to train the model for semantic segmentation of point clouds. Another solution is to effectively train models with only a small set of labelled data. Semi-supervised learning is one of the techniques proposed to train models with limited labelled data and a large amount of unlabelled data (Zhu and Goldberg, 2009). It takes advantages of unlabelled data in order to facilitate supervised learning tasks in the lack of labelled data. This has been applied to many
applications like image classification (Kingma et al., 2014), object tracking (Doulamis and Doulamis, 2014) and anomaly detection (Rav-anbakhsh et al., 2016). Tensor-based learning also shows considerable potentials in training models with small numbers of labelled samples (Makantasis et al., 2018a) as it significantly reduces the number of weight parameters required in model training (Makantasis et al., 2018b).

Identifying and only labelling the most informative samples is another promising alternative. Settles (2009) show that the informativeness of training samples differs. Some samples are informative and therefore improve the model performance, while some bring less information and others are even outliers to models. Only a part of the annotated data determines the models’ parameters. Therefore, an efficient learning strategy should be investigated to select the most informative samples for model optimization. Then manual annotation can be reduced because only selected informative samples need to be manually annotated. Active learning is an efficient learning strategy proposed to solve the problem.

The aim of active learning is to create a small training subset from a larger unlabelled data set. The strategy is to assess the sample informativeness in the unlabelled pool using the current model state. Then informative samples are manually annotated and added to the current training data for the next training. This minimizes manually labelling efforts during ground truth preparation while keeping the model performance in a supervised learning process. Active learning has been studied in many tasks like natural language processing (Wang et al., 2019a), object detection (Kellenberger et al., 2019), image classification (Wang et al., 2017), image semantic segmentation (Vežnjevets et al., 2012) and remote sensing (Tuia et al., 2011). Nevertheless, how to perform point cloud labelling tasks by active learning is rarely researched (Feng et al., 2019; Lin et al., 2020; Luo et al., 2018).

In addition to the active learning that iteratively selects informative samples for training, incremental learning is a type of machine learning technique where the learning process occurs whenever new data emerge and the current learned knowledge is adjusted according to the new data. In incremental learning, the model knowledge is continuously enlarged by the continually added samples. Incremental learning has been investigated in many computer vision applications like object recognition (Bai et al., 2015), image classification (Ristin et al., 2016) and segmentation (Tasar et al., 2019), visual tracking (Dou et al., 2015) and surveillance (Shin et al., 2018). Some researches involve active selection processes in incremental learning for image related tasks (Brust et al., 2020; Zhou et al., 2017). However, to our knowledge, there is no research to integrate active learning with incremental learning for semantic segmentation of ALS point clouds.

In this paper, we propose an effective framework for semantic segmentation of large-scale ALS point clouds in urban areas based on both active and incremental learning techniques. The objective of the paper is to effectively query informative samples that can improve the performance of deep learning models meanwhile minimizing the annotation efforts required for training data preparation. Also, we alleviate the training efforts by implementing incremental learning. We assess the informativeness of point clouds by three uncertainty metrics, point entropy, segment entropy and mutual information. The major contributions of this paper are as follows:

1) We introduce an active and incremental learning framework to effectively reduce the number of training samples required by deep neural networks for semantic segmentation of large ALS point clouds.

2) To identify the most informative parts of a point cloud for model training, we quantitatively assess both data dependent and model dependent uncertainties using three independent metrics. The data dependent uncertainty is estimated by point entropy and segment entropy. The segment entropy considers interactions among neighbouring points. Model dependent uncertainty is estimated by mutual information which analyses the disagreements produced by different model parameters.

3) To make use of the knowledge obtained from previous training and reduce training efforts in the task of semantic segmentation of large ALS point clouds, we allow the model to incrementally learn from the point clouds by fine-tuning the model obtained from the previous stage instead of training the model from scratch for each active learning iteration.

The rest of the paper is structured as the following. Section 2 reviews previously related work. Section 3 introduces the proposed active and incremental learning framework and describes the network structure as well as three query functions. Experimental results are presented and analysed in Section 4. Section 5 concludes the main observations in the paper and provides some suggestions for the future work.

2. Related work

2.1. Deep learning approaches

Recently, deep learning algorithms have been improving accuracy on semantic segmentation of point clouds. There are two groups of methods, 2D and 3D. For 2D methods, point clouds are firstly projected onto 2D image planes and then passes into Convolutional Neural Networks (CNNs). For example, Kalogerakis et al. (2017) and Boulch et al. (2018) capture point cloud images from different views and take these images as the input of image based CNNs. The output pixelwise image labels are then projected back to 3D points. However, the work of Kalogerakis et al. (2017) only deals with partial segmentation of objects instead of assigning pointwise labels for either indoor or outdoor scenes which are much more complex. Although SnapNet (Boulch et al., 2018) is designed for semantic segmentation of point clouds in complex scenes, during the projection, self-occlusion is inevitable, especially for complicated scenes. To deal with ALS data covering large areas, some researchers project point clouds onto 2D grids from the top view consisting of some simple attributes per grid cell like mean, minimum and maximum height (Hu and Yuan, 2016). Yang et al. (2017) also use 2D grids but improve the accuracy of ISPRS benchmark dataset by adding more geometric and full-waveform features to 2D grids. However, their method requires large memory to process the data.

Semantic point cloud segmentation can also be solved by 3D deep learning networks. To adapt unstructured point clouds to 3D convolutional filters, a group of methods partition 3D space into small grid voxels (Maturana and Scherer, 2015; Tchapmi et al., 2017; Wu et al., 2015). However, comparing to initial point clouds, the voxelization not only causes loss in data representation but also introduces artifacts. These drawbacks hinder the learning of 3D features. Also, voxel structures store unoccupied grids in 3D space and this leads to high memory requirements. To avoid the disadvantages of the voxelization, networks that can directly consume unstructured point clouds are designed.

PointNet (Qi et al., 2017b) directly takes unstructured points as inputs and learns pointwise feature through a sequence of Multilayer Perceptron (MLP) layers. With the spatial transformer, the network is robust to variance in geometric transformation. Since PointNet only allows pointwise features to be learned independently, PointNet++ with a hierarchical structure is proposed to capture the geometric relationships among points in different scales (Qi et al., 2017b).

To exploit the contextual information among neighbouring points like 2D convolutional kernels, 3D convolutional networks are also introduced to directly consume irregularly structured point clouds. Unlike Voxnet (Maturana and Scherer, 2015) applying 3D convolutions on a discrete space, some methods define 3D convolutional operators over continuous space. For each point, weights of the neighbouring points depend on spatial distribution around the central points. Thomas et al. (2019) define both fixed and deformable Kernel Point Convolutions (KPCov) on continuous space. Linear correlation, assessing the
Fig. 1. The proposed framework for active and incremental learning strategy for the semantic segmentation of point clouds. First of all, point clouds in the training area are split into tiles and separated into two groups: labelled (minority) and unlabelled (majority). If no previous model is available, the network is trained from scratch. Otherwise, the model is incrementally fine-tuned according to the previous model. The model is validated on the validation tiles to avoid the overfitting to labelled data during the training. Then, the trained network selects unlabelled tiles by one of three queries depending on the metric to assess the informativeness of the tile. Query 1 directly consumes unlabelled tiles and query 2 relies on the unsupervised segmentation. Query 3 directly takes unlabelled points and evaluates the disagreement caused by model parameters. Selected tiles are labelled before the next training. The trained network is evaluated on testing tiles in each iteration.
distances between neighbouring points and kernel points, is applied to assign different weights to different areas inside the domain of convolutional kernels. Positions of kernel points in deformable convolutions are learnable and can adapt to local geometry. Some other methods also define 3D operators in continuous space like SpiderCNN (Xu et al., 2018), PointConv (Wu et al., 2019) and Flex-Convolution (Groh et al., 2019).

Some studies introduce graph convolutions to semantic point cloud segmentation tasks, where each point is taken as a graph vertex and edges are defined by relations among neighbouring points. For example, EdgeConv (Wang et al., 2019b) dynamically computes graphs not only on 3D spatial space but also on higher dimensional feature space, in order to capture the topological information in point clouds. In addition to construct graphs over single points, Landrieu and Simonovsky (2018) define graphs on superpoints which are geometrically homogeneous point sets, aiming to efficiently deal with large scale point clouds. Edges in SuperPointGraphs represent the adjacency relationships between superpoints. Graph Convolutional Networks (GCNs) are applied to exploit the contextual information among shapes and object and this makes the GCN to consider a wider range of point clouds compared to point based GCNs.

2.2. Active learning

The objective of active learning is to sample data based on a calculated informativeness metric and maximize model performance with fewer labelled samples. How to evaluate the informativeness of samples is the main research question in active learning and this has been studied in the machine learning community for a long time. There are many ways to evaluate the sample informativeness for active learning. Uncertainty-based active learning criteria is probably the simplest and most commonly used (Settles, 2009). It selects samples the ‘model’ is least certain about, like again sampling and least confident sampling. This is a simple and direct method for probabilistic learning models (Settles, 2009). Density weighted methods query samples by assessing the intrinsic distribution and structure of the data and select samples are representative to the whole dataset, like Gaussian similarity (Zhu, 2005), divergence similarity (McCallumzy and Nigamy, 1998) and clustering (Xu et al., 2007). Expected change based methods estimate the influence of unlabelled samples on the current model. For example, Settles and Craven (2008) choose samples that make the largest change in the model by calculating the expected length of the gradient.

Recently, active learning has been incorporated with deep architectures in many studies. Gal et al. (2017) propose several uncertainty-based metrics based on Bayesian CNNs for image classification. Monte-Carlo dropout technique is applied to approximate the Bayesian process for image classification. In the experiments, Beluch et al. (2018) propose a framework to integrate higher order Markov Random Field (MRF) with active learning in order to efficiently assign pointwise labels to mobile LiDAR point clouds with limited labelling efforts. Assuming two nearby points are likely to share the same label, Luo et al. (2018) evaluate the neighbour-consistency during the sampling. That means, for a certain supervoxel, it is taken as a wrongly labelled sample if its predicted label is not the same as the label of its nearby manually annotated supervoxel. In this case, the ‘incorrectly labelled’ samples will be queried and manually annotated and then used to improve model performance in the next iteration. Although this work takes the advantage of interactions among neighbouring supervoxels and saves manual labelling by selecting optimal training supervoxels, taking MRF as the classifier still requires hand-crafted features which is less representative compared to deep learning features. Feng et al. (2019) propose a framework to integrate a state-of-the-art deep learning method with uncertainty-based active learning queries for 3D object detection in point clouds. They evaluate both aleatoric (data dependent) and epistemic (model dependent) uncertainty through Monte-Carlo dropout and deep ensembles techniques. With their active learning strategies, the model only needs 40% of the labelled data to achieve comparable accuracy when using all labelled data. Lin et al. (2020) integrate the deep learning network PointNet++ with two data dependent metrics for the semantic segmentation of ALS point clouds. However, the method does not take advantages of the knowledge learned from the previous stage, which makes the training process very time-consuming.

2.3. Incremental learning

An important step to make the active learning training more efficient is to retain the knowledge from previous tasks, i.e. training steps. Incremental learning is a method where new data are continuously added to existing training data in order to extend the knowledge of the current model. In the deep learning paradigm, classifiers and task-specific features are jointly learned. New samples cannot be simply added to update parameters as can be done in models like least-squares regression, because neural networks are non-convex and highly non-linear. To update model parameters in non-convex and highly non-linear networks, optimization techniques, such as gradient descent, are implemented to gradually refine model parameters to achieve global optima. In this scenario, simply updating models incrementally by only using new data is likely to make large changes in previously learned weights, forcing the model to adapt to new data. Therefore, its performance on the old data dramatically degrades (Kirkpatrick et al., 2017).

To maintain the performance on the old task, Castro et al. (2018) propose an end-to-end incremental learning strategy by combining distillation loss with cross entropy loss for image classification tasks. Distillation loss which is used to transfer information between different networks is adapted to maintain knowledge obtained from previous tasks and a cross entropy loss is used to learn from new data. Russo et al. (2016) propose progressive neural networks where features acquired from old tasks are blocked to retain previous knowledge and new sub-networks are created to learning information from new data. In addition to enlarging the network, Kirkpatrick et al., (2017) propose the elastic weight consolidation (EWC) which penalizes on the difference between the new and old tasks. Brust et al. (2020) integrate incremental learning strategies with active learning criteria for object detection. The incremental learning is achieved by simple yet effective fine-tuning. After selecting new data by active learning criteria, newly labelled samples and old samples are assigned with different weights and are mixed. The model trained by the old data is updated to acquire information from new data by fine-tuning with those weighted samples. Similarly, Zhou et al. (2017) propose a framework to actively and incrementally fine tune CNNs for biomedical images. To our knowledge, there is no research that combines active and incremental learning with deep learning for semantic segmentation of point clouds.

3. Method

The proposed workflow is presented in Fig. 1. The red dash line box illustrates the active and incremental learning strategy introduced in this paper. There are three major steps, namely; training, point cloud
query, and annotation. The following sections first describe the active learning framework. Next, the details of the network used in our paper are explained. Then, point entropy, segment entropy and mutual information are introduced to select informative point cloud tiles. Finally, how the incremental learning strategy is implemented in our research is explained.

3.1. Active learning

Let us consider a set of unlabelled point cloud tiles $S$ which are generated by splitting the training area. To initialize the active learning framework, we first select and annotate a subset $S_0$ from $S$. We ensure $S_0$ contains at least one instance of each class. Then $S_0$ are excluded from $S$ and we define the reduced unlabelled pool as $S_0$. The initial network $M_0$ is trained on $S_0$. The active learning loop starts with the trained model $M_0$ estimating the informativeness of all point cloud tiles in the unlabelled pool $S_0$. Then we select $K$ samples that are most informative and annotate them to form a set of labelled tiles $S_1$. We update current labelled pool $S_0$ with $S_1$ to form a new labelled set $S_1$, while excluding $S_1$ from the current unlabelled pool $S_0$ to form a new unlabelled set $S_0$. Instead of training from scratch, we obtained a new model $M_1$ by using all labelled tiles $S_1$ to incrementally update the model $M_0$ obtained from the previous step. For the $n^{th}$ iteration, selected tiles, the labelled pool, the unlabelled pool, and the trained model are defined by $S_n, S_n, S_n$ and $M_n$ respectively. This querying and training loop is repeated until the stopping criterion is met, such as no significant improvement in network performance for several iterations or sufficient network performance has been achieved. Algorithm 1 summarizes the active and incremental learning strategy step by step.

### Algorithm 1: Active Learning Algorithm

**Input**: a pool of unlabelled point cloud tiles $S$, and a trained neural network $M_0$.

**Output**: the manually labelled point cloud tiles $S_n$, and a trained neural network $M_n$.

**Initialization:**

1. $S_0 = \text{SampleAnnotation}(S)$ \text{Manually annotate a small subset.}
2. $S_0 = S_0 \setminus S_0$ \text{Update set of unlabelled point cloud tiles.}
3. $M_0 = \text{Train}(S_0)$ \text{Train $M_0$ from scratch.}

**Active selection and incremental model updates:**

1. $n = 1$
2. while the stopping condition is not met, do:
   1. $S_n = \text{ALquery}(M_{n-1}, S_{n-1})$ \text{Select $K$ tiles for labelling.}
   2. $S_n = S_{n-1} \cup S_n$ \text{Update labelled tiles.}
   3. $S_n = S_{n-1} \setminus S_n$ \text{Reduce unlabelled pool.}
   4. $M_n = \text{Train}(S_n, S_n)$ \text{Use all labelled data to update the model.}
   5. $n = n + 1$
3. Return $S_n$ and $M_n$

Unlike picking points or super voxels proposed by Luo et al. (2018), point cloud tiles ($S_a$) are queried by functions introduced in Section 3.2. Luo et al. (2018) calculate pre-defined pointwise geometrical features based on neighbouring points, like planarity and linearity. Then an MRF classifier is trained for assigning a label to each point according to those pre-defined features. However, deep learning based methods learn geometrical features from data. Thus means that the input to the network should be a group of points that can preserve geometrical information, instead of a single point with a set of pre-defined features. Therefore, point cloud tiles are taken as the input of the forward pass and network weights are updated through back-propagation according to the loss function. If only a part of the tile is annotated, a complete tile is still required as the network needs the input to have geometrical representative characteristics. The computational cost will not change with the proportion of labelled points. The only thing that will be changed is the loss where unlabelled points give no contribution. If we query points from all unlabelled points, all point cloud tiles are required to be put into the network for each training. The training time for one epoch is the same as that of using the fully labelled whole training data. However, if we select point clouds by tiles, querying fewer tiles results in less computation time to complete a single training epoch.

4.2.1. Point entropy

Shannon Entropy (SE) is an information metric indicating how much information is required to ‘encode’ a distribution.

$$E = - \sum_{i=1}^{c} p(y = c|x) \log(p(y = c|x))$$

where $p(y = c|x)$ is the predictive probability for class $c$ coming after the softmax function at the last layer of the network.

When the model is quite certain about a class label, it will assign a very high predictive probability to that class and giving low values to other classes. In this case, the entropy value is low. On the contrary, high entropy value occurs when similar predictive probabilities are given to multiple classes and this suggests the model is not confident in the prediction. Here, we select samples that the model is most uncertain about and therefore those with high entropy values will be queried. In this research, point clouds are selected by tiles. To estimate the informativeness of the tile, we calculate the mean of pointwise entropy within each unlabelled tile. The $K$ tiles with the highest average pointwise entropy will be queried, annotated, and added to labelled data pool for the next training.

3.2.2. Segment entropy

Apart from assessing pointwise uncertainty, the informativeness of point cloud tiles can also be evaluated at the segment level. The objective of point cloud segmentation is to partition point clouds into geometrically homogenous units. In this paper, we use the unsupervised
segmentation method proposed by Vosselman et al. (2017). This method combines both planar surface extraction algorithms and feature based segmentation methods. Firstly, a Hough transformation and surface growing algorithms are implemented to extract planar objects but they produce unnecessary small fragments in non-planar objects like trees. As a result, in the second step, only large segments are kept as planar objects and the remaining points are re-segmented by the feature based segment growing algorithm. The algorithm considers normal vector directions and planarity to group points on non-planar objects like vegetation, chimneys and cars. Next, to overcome the over-segmentation on imperfect planar ground points, large adjacent segments are merged if their normal vectors are nearly parallel and points in one segment are also able to fit the plane of the other segment and vice versa. Finally, unsegmented points are given segment labels by majority voting in their neighbourhood. Isolated points still without a segment label are excluded from segment entropy calculation.

Here we assume that points within a geometrical homogenous unit share the same semantic label. Hence, if a model gives different labels to points within a segment, this model is likely to generate wrong predictions on this segment and thus those uncertain samples should be selected for the next training. The percentage of different predicted class labels within a segment is used to assess the informativeness of point cloud tiles. Suppose we have a segment consisting of N points and predicted pointwise labels are represented by \( \tilde{y}_1, \ldots, \tilde{y}_n, \ldots, \tilde{y}_N \). The following shows how we calculated segment entropy:

\[
\tilde{y}_n = \text{argmax}_c P(y_n|x)
\]

\[
f(\tilde{y}_n, c) = \begin{cases} 
1, & \text{if } \tilde{y}_n = c \\
0, & \text{else}
\end{cases}
\]

\[
q(c) = \frac{\sum_{n=1}^{N} f(\tilde{y}_n, c)}{N}
\]

\[
E_{seg} = -\sum_{c=1}^{C} q(c) \log(q(c))
\]

where \( E_{seg} \) represents the entropy within a segment and \( q(c) \) is the proportion of class \( c \) among the predicted labels, computed in equation (4). \( \tilde{y}_n \) is the predicted class label which has the highest predictive probability. In order to avoid the underestimation of informativeness on large segments, segment entropy is given to their point members and then the mean of pointwise segment entropies is calculated to represent the informativeness of unlabelled tiles. Fig. 2 illustrates the predicted labelled on the roof segment produced by models. The middle figure shows the variance of predicted labels on the roof and this variance leads to a high segment entropy. Tiles that comprise segments with high entropies are likely to be picked for the training in the next iteration.

### 3.2.3. Mutual information

The above two metrics evaluate the data dependent (aleatoric) uncertainties. The following section explains how to estimate model dependent uncertainty by mutual information based on Bayesian Neural Networks. Bayesian Neural Networks are neural networks where prior probability distributions, like standard Gaussian priors, are placed over model parameters (Gal et al., 2017). However, direct inference from Bayesian networks is computationally expensive. Therefore, as a stochastic regularization technique, dropout which randomly ignores some of the neurons during the training, is used to approximate inference in Bayesian networks (Gal and Ghahramani, 2016). To extract uncertainty in prediction induced by the uncertainty in weights, multiple forward passes are performed with activated dropout during the testing, which samples from the approximate posterior.

In PointNet++, before the final prediction, a fully connected layer is inserted to integrate all features and then give logits to every class. Dropout is often set in this layer to prevent overfitting and is also used to construct a Bayesian network. Normally, we turn off the dropout during the prediction, but here we keep it on to get samples from the approximate posterior distribution of models.

Predictive probability distributions for n runs with dropout are represented by \( p(y|x, w_1), p(y|x, w_2), \ldots, p(y|x, w_n) \). Mutual information between predictions and model posterior is calculated by the following equation:

\[
MI = E( \frac{1}{n} \sum_{i=1}^{n} p(y|x, w_n) - \frac{1}{n} \sum_{i=1}^{n} E(p(y|x, w_n)))
\]

where \( E() \) is the function of Shannon Entropy. High \( MI \) values suggest that the model is not confident in the predictions on samples on average, but different model parameters cause disagreement in predictions. In other words, each stochastic forward pass would have the highest predictive probabilities assigned to different classes. In this case, although the entropy of each run can be very small giving rise to a small value of the second term in equation (6), there is no significant large value in the averaged predictive probability distribution, which leads to the high entropy value for the model prediction, the first term in equation (6). Samples which maximize this \( MI \) metric are taken as informative data used for the next training. In our work, we calculate the average pointwise \( MI \) values within point cloud tiles and those causing uncertainties in model predictions are selected.

### 3.3. Semantic point cloud segmentation by neural networks

An important component in our framework is the deep learning based model. Currently, many point based networks are available as mentioned in Section 2 and our proposed framework is supposed to be adapted to those models. To demonstrate the effectiveness of the proposed learning strategy, we pick PointNet++ (Qi et al., 2017b) as the model in this paper. This is because Pointnet++ inherits MLP layers from PointNet to encode features in the local region, and the implementation of MLP layers which allow the network to directly consume points are still very popular in many deep learning based models (Landrieu and Simonovsky, 2018; Li et al., 2020; Wang et al., 2019b). PointNet++ (Qi et al., 2017b) recursively apply a set of MLP layers to construct a hierarchical neural network. The network capture contextual information in point clouds by using set abstraction modules at multiple scales. The set abstraction module includes three sub-phases, namely, sampling, grouping and PointNet. A subset of the point cloud is collected by iterative farthest point sampling (FPS) in the sampling phase. The FPS gives better coverage of the point cloud and minimizing the clustering of points in a small region. This sampling strategy also adapts receptive fields to the points’ distribution. In the grouping stage, neighbours around selected points are gathered. Then, selected points are taken as the input of the MLP. Here, a single input point relates to a small local region and represents a small point set, in which each member contains its features, like XYZ coordinates or features obtained from previous set abstraction modules.

In this paper, the Adam optimiser (Kingma and Ba, 2014) is utilized to optimize the weights in PointNet++. A weighted cross entropy loss function is used to cope with imbalanced data. The dropout technique which ignores some of the neurons is implemented in the process of the training to avoid overfitting and we keep validating the model on the validation dataset to obtain optimal model weights. Early stopping is applied to terminate the training.

### 3.4. Incremental learning

Some of the active learning strategies focus on how to select samples step by step but ignore the knowledge learned from the previous learning stage during the training. For example, Luo et al., (2018) train the model from the scratch for every step and Gal et al. (2017) train all
models starting from a pre-trained VGG16 CNN model for the image classification task. To make good use of the previously learned information and speed up the training process, in this paper, the model is incrementally fine-tuned from the model obtained in the previous step.

Most of the incremental learning methods mentioned in Section 2 deal with the case that new data are continuously added and this process may include new classes that are not used in the previous training. Also, old data is often unavailable in those cases and they can only train the model on the new data. However, in our study, the task is simpler. We do not introduce new classes during active learning steps and old data remain available. While keeping the model performance on all classes, we only need to make good use of the previous knowledge to speed up the training process instead of training from scratch for all models. Therefore, to speed up the training process, we modify the simple but effective strategy mentioned in Brust et al. (2020). Brust et al. (2020) suggest that parameters from the last active learning iteration can be used as the initialization of the current model in order to maintain the knowledge from previous training efforts. We incrementally fine-tune the models on both old tiles and newly selected tiles.

4. Experiments

Three active learning strategies are tested with ALS point clouds in our experiments. More details about the dataset, the specific structure of PointNet++, training parameters and how the proposed query functions are implemented are explained in the following paragraphs.

4.1. Data description

Actueel Hoogtebestand Nederland (AHN) dataset offers ALS point clouds with very high point density and high penetration from the multiple returns. It covers almost the entire area of the Netherlands. AHN31 is the latest version, covering more than half of the Netherlands. In this paper, two subsets of AHN3 datasets \(^1\) are chosen for the experiments. The subsets are captured by an IGI LM6800 system with a \(60^\circ\) field of view. The mean strip overlap is 30\% and the survey was designed to obtain the point density at 60 points/m\(^2\). One subset is located in the centre of Rotterdam (Fig. 3), covering a \(2 \times 2\) km\(^2\) area. It is a densely built-up area with high rise buildings surrounded by trees and there are river channels with bridges. The point clouds were acquired on 4th December 2016 and manually annotated with seven classes, namely ground, roof, water, façade, vegetation, artwork, and clutter. The other dataset is situated in Amsterdam and was captured on 2nd February 2014. Its size is much larger than the Rotterdam datasets covering an area of \(5 \times 6.25\) km\(^2\) (Fig. 4). The Amsterdam dataset not only includes the Amsterdam central area which is characterized by the buildings with complex shapes but also includes residential areas, parks and farmlands. Also, river channels in the Amsterdam dataset are crisscrossing and are much narrower than the river in the Rotterdam central area. As the Amsterdam dataset is quite large, we directly use the labels provided in the AHN3 dataset and classify points into 4 categories, namely ground,

---

\(^1\) https://www.pdok.nl/nl/ahn3-downloads
building, water and clutter.

4.2. Preprocessing

Since GPU memory is limited, it is unfeasible for a network to directly consume the whole study area. Therefore, the point cloud is cropped into $50 \times 50$ m$^2$ tiles and only XYZ coordinates are kept as the input of the network. We keep the Z-coordinates and normalize X- and Y-coordinates by the starting position of the tiles. In experiments, we randomly select 20,000 points as the input of the network. For tiles with more than 20,000 points, we select without replacement. For those with less than 20,000 points, all points are used as the input and the rest is compensated by random and repeated selection. With the purpose of making the model more robust to various orientations and noises, during the training, we randomly rotate point clouds around the Z-axis. Furthermore, Gaussian white noise with a $\sigma$ of 4 cm is added to XYZ coordinates and the maximum perturbation values are restricted to 15 cm. These values are chosen empirically to add noise that will not significantly change the geometrical features for target objects.

4.3. Network implementation

Fig. 4. The overview of the Amsterdam dataset. The training area is in the black box. The validation area is in the brown box and the testing area is in the grey box. The area to initialize the model is in the purple box.

<table>
<thead>
<tr>
<th>Level</th>
<th>Number of points</th>
<th>Search radius (m)</th>
<th>Number of neighbours</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20,000</td>
<td>[2, 4]</td>
<td>[16, 32]</td>
</tr>
<tr>
<td>1</td>
<td>4096</td>
<td>[4, 8]</td>
<td>[16, 32]</td>
</tr>
<tr>
<td>2</td>
<td>1024</td>
<td>[8, 16]</td>
<td>[16, 32]</td>
</tr>
<tr>
<td>3</td>
<td>256</td>
<td>[16, 32]</td>
<td>[16, 32]</td>
</tr>
</tbody>
</table>

As mentioned in Section 3.3, PointNet++ consists of a sequence of sampling and grouping layers. Table 1 shows the spatial scales of set abstraction modules. The first sampling and grouping layer selects 4096 points from 20,000 points in tile by iterative farthest point sampling.
strategy. Next, nearby points are grouped at two scales. 16 points are selected with a spherical search radius of 2 m and 32 are searched within 4 m. For the next set abstraction module, 4096 points are subsampled to be 1024 and neighbouring points are searched and gathered within two larger scales. Fewer points can be sampled by the abstraction modules at higher levels and this inevitably leads to the loss in information in the latter layers of the network but this is beneficial for the network to exploit relationships among points in a wider range.

During the training, the learning rate for the initial model starts from 0.005 with a decay rate of 0.7 at every 75 training iterations. The learning rate keeps decreasing until it is less than 0.0001. Then the rate is kept at 0.0001 for the rest of the training. An early-stop strategy is applied to avoid overfitting. As the Rotterdam validation dataset is small, we check the performance on the validation dataset every epoch and stop training when the performance fails to improve over 15 epochs.

For the Amsterdam dataset, the validation data contain about 2500 tiles. Here, it is not feasible to validate the model performance every epoch because the validation time is much longer than the training time. To balance the time spent on training and validation, we only check the model performance every several epochs. The check frequency changes with the size of the labelled data pool. Suppose we have $J$ samples in the initial labelled training data, the check frequency is calculated by $\left\lceil \frac{2500}{J+K_n} \right\rceil$. Here $K$ represents the number of newly added samples in the $n$th iteration. The training is stopped when there is no improvement in model performance for 2 checks.

Due to the sampling, some points remain unlabelled in original point clouds. Therefore, pointwise predictions are propagated to the whole original tiles by nearest neighbour interpolation. To obtain the prediction on the test dataset, the data pass through the network 10 times and the predictive probability is averaged.

4.4. Accuracy assessment

Intersection over Union (IoU) is utilized to evaluate network performance. IoU per class is computed from true positives (TP), false negatives (FN) and false positives (FP) in confusion matrices as $\frac{TP}{TP+FN+FP}$.

4.5. Active learning setup

For the Rotterdam dataset, an area where all seven classes (ground, roof, water, façade, vegetation, artwork, and clutter) exist should be selected to initialize the first model. The model performances initialized with different numbers of tiles ($\mathcal{O}$) are shown in Fig. 5. Model performance is quite similar when $\mathcal{O}$ is set as 50 and 107. When $\mathcal{O}$ is 200, 410 tiles are required to achieve the full training mIoU. In the following experiments, the area covering $600 \times 600$ m$^2$, consisting of 107 tiles is selected to initialize the first model. After excluding very sparse tiles, 783 tiles are in the unlabelled pool, waiting to be selected. Then the next question is how many tiles we need to select in each iteration, that is the value of $K$ mentioned in Algorithm 1. Some studies only select a single sample in each iteration. As this requires to run the training and selecting process many times, it would be a very time-consuming process. Therefore, multiple point cloud tiles are queried in every iteration. Yet, it is not a wise choice to query a large portion of data like a quarter of the tiles for annotation because less important tiles which can only make little contributions to model performance will be...
selected and all tiles will be annotated in four iterations which conflicts with the purpose of the active learning, namely saving annotation efforts. Fig. 6 below illustrates how the model performance changes with increasing training tiles when selecting different numbers of tiles in each iteration. We test three sizes 10, 35 and 70 which corresponding to about 1%, 4% and 8% of the tiles in the training area respectively. When adding 35 tiles each iteration, after training and querying for 2.9 h, 282 tiles are selected and fed to the model and the model performance becomes stable and fluctuated around the mIoU obtained by the full trained model. When taking $K$ as 70, although the training and selecting process takes 2.5 h, the model requires 387 annotated tiles to achieve the same status. When $K$ equals 10, the model takes 19 steps (6.9 h) with 297 annotated tiles to reach the full train IoU. It fails to reduce the number of required tiles and leads to longer training time. Therefore, the labelled training pool is updated by 35 tiles in each iteration.

For the Amsterdam dataset, the model performances initialized with different numbers of tiles ($L_0$) are shown in Fig. 7. When the model is initialized by 200 tiles, it requires more than 600 labelled point cloud tiles to reach the full training mIoU. Here we select a 500 $\times$ 500 m$^2$ area, which contains 100 tiles to initialize the first model. We compare three sizes of queried tiles, namely 50, 100, 200 which corresponding to about 1%, 2% and 4% of the tiles in the training area respectively (Fig. 8). It can be seen that $K$ equals 50 firstly approach the around the mIoU obtained by the full trained model while the other two values still require more iterations. To save manual annotation efforts, the training data is updated by 50 tiles step by step.

In our experiments, the active learning strategies based on point

<table>
<thead>
<tr>
<th>Method</th>
<th>New data</th>
<th>All data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training time (hours)</td>
<td>2.6</td>
<td>3.8</td>
</tr>
</tbody>
</table>
entropy, mutual information and segment entropy are compared with the baseline method in which unlabelled tiles are randomly selected. For the mutual information metric which evaluates disagreements among various model variants, each point cloud tile is predicted under 10 different parameter settings. The process of querying and training is run for 10 iterations to see which strategy first makes the model achieve a high level with the least training samples. To demonstrate the effectiveness of the proposed method, for each query, experiments are repeated 3 times and results are averaged.

4.6. Incremental learning setup

In our experiments, we incrementally fine tune models with all available data. Although fine-tuned with only newly selected data takes less time to train the model (Table 2), its model performance is much worse compared with using all available data (see Fig. 9).

To set the learning rate for fine-tuning, the effects of the learning rate on model performance are presented in Fig. 10 and computation times are listed in Table 3. The active and incremental learning process ends up with similar model performance with different fine-tuning learning rates. However, when the learning rate is 0.005, the model takes a longer time to converge when it is compared to the other two values. Here we set the fine-tuning learning rate as 0.0001 to avoid models fall into local optima.

4.7. Results

4.7.1. Comparison of selection queries

Model performances of various active learning functions are presented in Fig. 11 and Fig. 16.

4.7.1.1. Rotterdam.

Fig. 11 and Table 4 illustrate how the model performance on the Rotterdam dataset changes with an increasing number of samples selected by different active learning query functions. It can be seen that for all functions model performance tends to increase with some fluctuations. Point entropy, segment entropy and mutual information give better results than the baseline method. The random selection leads to an unstable model performance which is illustrated by the large standard deviation in the mIoU. When it comes to the other three query functions, the standard deviation is only large at the beginning which can be explained by the asynchronous improvements among different runs. Then the standard deviation becomes relatively small in the later iterations where mIoU is similar to the value obtained from the full training. This suggests that the selected samples do provide useful information to improve model performance. When comparing three query functions, segment entropy performs the best and it first reaches full train accuracy at the fifth iteration where 282 tiles, 31.7% of the tiles in the training area are used. Point entropy also reaches full train mIoU at the fifth iteration but its mIoU is a little bit lower than that of segment entropy. Mutual information reaches the full train mIoU at the 6th iteration where 35.6% of the tiles in the training area are used. In terms of mIoU, all query functions select meaningful data for model training and can be used to save manual annotation efforts.

Fig. 12 shows the change in IoU for different classes (lines) and the
variation of training data distribution (columns) with more selected training samples. It can be seen that the three query functions improve model performance for the classes ground, water, clutter, and artwork. For other classes, the IoU values are comparable to those of the baseline but the results are more robust as the standard deviations are much smaller than those of the baseline. One possible reason for the insignificant improvement in these classes is that they are relatively easy classes for the model and the model can easily acquire enough information to differentiate them and then reach high accuracy. As a result, newly added samples are hard to enrich the model knowledge of these classes. One observation is that all three query functions are able to select tiles with difficult classes like artwork and clutter. When selecting samples by our query functions, the IoU values for artwork and clutter are higher than those of the baseline, especially artwork. This suggests that all active query functions achieve the objective of selecting informative samples for deep learning models.

Fig. 13 demonstrates some samples selected by point entropy uncertainty. It can be seen that high uncertainty values are around object boundaries, clutter objects and on sloped ground. The model is uncertain on points within slanted ground segments because those segments are similar to slanted roofs in terms of the geometry. However, this uncertainty is not visible for segment entropy because the segmentation
algorithm separates flat ground and slant ground into different parts. Most of the sloped ground points are predicted to be roof leading to a low value in segment entropy. Although ground points are dominant in tiles selected by segment entropy, most of them are on the flat surface. Different types of geometry for ground points explain the higher IoU values in the ground for point entropy at the initial iterations.

Fig. 12 shows that mutual information selects tiles with abundant tree points which are also shown in Fig. 14. Although most of the vegetation and ground points are correctly predicted, the mutual information is still high on the ground points because with dropout during the testing, some models are quite confident in predicting ground points as vegetation. However, selecting tiles that only have ground and vegetation points makes little contribution to model knowledge because vegetation and ground points already got relatively high accuracy. Similar to segment entropy, mutual information can also detect the uncertainty at object boundaries.

Relating to the distribution columns in Fig. 12, segment entropy prefers tiles with more ground points comparing to point entropy and mutual information. For example, in Fig. 15, PointNet++ is not good at object boundaries and some ground points surrounding clutter objects are predicted as clutter. As we enforce the consistency within segments, those wrongly predicted flat ground points enlarge the segment entropy of tiles. Although its IoU values in ground are not better than the other two methods, this could help solve the confusion between clutter and ground points and explain the better accuracy for clutter (Fig. 12). Also, segment entropy selects scenes where trees are quite close to building facades and part of the canopy is likely to be predicted as façade points. As a tree canopy is always taken as one segment according to the unsupervised segmentation algorithm, the inconsistency of predicted labels in a canopy segment leads to large segment entropy over the tile.

Table 5 shows the network bias on unsupervised segments. We estimate the percentage of points that are incorrectly predicted but have
no contribution to the segment entropy. In Table 5 a), it can be seen that the bias on artwork is quite large at the initial stage of the active learning. 47.35% of the artwork points in the unlabelled tiles are predicted as ground and their segment constrained labels are also ground. This is in accordance with the low percentage of artwork points are selected for the training of iteration 1 in Fig. 12. With \( M_0 \), the segment

The network bias is alleviated by selected informative samples. Table 6 demonstrates the computation time to train models and query samples for the Rotterdam dataset. The table indicates that the entropy is also insensitive to select artwork segments that are wrongly predicted as clutter. Trained with more informative samples, \( M_{10} \) produces much less bias comparing to the \( M_0 \), especially for the artwork. The network bias is alleviated by selected informative samples.

Table 6 demonstrates the computation time to train models and query samples for the Rotterdam dataset. The table indicates that the
training times required for different methods are quite similar but the mutual information metric takes much longer time to select informative point cloud tiles. This is because as mentioned in Section 3.2.3, point clouds are supposed to pass the network multiple times, in order to estimate the disagreement among the predictions. In comparison, point entropy and segment entropy are much faster to select informative samples.

4.7.1.2. Amsterdam. Fig. 16 and Table 7 show how the model performance on the Amsterdam dataset responds to an increasing number of samples selected by different active learning query functions. In terms of the mIoU, the performance of the baseline is quite stable after the third iteration. The improvement in model performance is insignificant with an increasing number of randomly selected tiles and the mIoU is much lower than that of the other three methods at the final iteration. For the

<table>
<thead>
<tr>
<th>Iteration</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.546</td>
<td>0.643</td>
<td>0.629</td>
<td>0.677</td>
<td>0.682</td>
<td>0.685</td>
<td>0.686</td>
<td>0.691</td>
<td>0.689</td>
<td>0.689</td>
<td>0.699</td>
</tr>
<tr>
<td>Point entropy</td>
<td>0.546</td>
<td>0.636</td>
<td>0.657</td>
<td>0.661</td>
<td>0.692</td>
<td>0.690</td>
<td>0.696</td>
<td>0.734</td>
<td>0.735</td>
<td>0.749</td>
<td>0.762</td>
</tr>
<tr>
<td>Mutual information</td>
<td>0.546</td>
<td>0.631</td>
<td>0.668</td>
<td>0.693</td>
<td>0.695</td>
<td>0.703</td>
<td>0.743</td>
<td>0.742</td>
<td>0.759</td>
<td>0.767</td>
<td>0.772</td>
</tr>
<tr>
<td>Segment entropy</td>
<td>0.546</td>
<td>0.618</td>
<td>0.668</td>
<td>0.666</td>
<td>0.732</td>
<td>0.735</td>
<td>0.740</td>
<td>0.775</td>
<td>0.773</td>
<td>0.785</td>
<td>0.780</td>
</tr>
</tbody>
</table>

Table 7
Mean IoU scores of baseline and active learning strategies with different query functions for the Amsterdam dataset. For every function, the first value to reach the full train mean IoU is in bold.

Table 8
Comparison of the computation time required for the Amsterdam dataset using different query functions.

<table>
<thead>
<tr>
<th>Training time (hour)</th>
<th>Baseline</th>
<th>Point Entropy</th>
<th>Mutual information</th>
<th>Segment Entropy</th>
<th>Full train</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.04</td>
<td>10.11</td>
<td>10.33</td>
<td>9.35</td>
<td>10.16</td>
<td>4.78</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>3.84</td>
<td>30.65</td>
<td>4.15</td>
<td>0.00</td>
</tr>
<tr>
<td>4.97</td>
<td>14.17</td>
<td>40.00</td>
<td>14.31</td>
<td>4.78</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 16. Mean IoU scores of baseline and active learning strategies with different query functions for the Amsterdam dataset. The horizontal axis represents the iteration. Error bars represent standard deviations.
other three query functions, the mIoUs gradually increase to a level which is slightly above the mIoU achieved by using all samples during the training. Segment entropy firstly reaches the full-train mIoU at the seventh iteration where only 9% of the tiles are used for training. Mutual information achieves 0.759 (mIoU) using 10% of the tiles at the eighth iteration and point entropy only reaches 0.762 (mIoU) with 12% of the tiles at the tenth iteration. Computation times to train models and query samples for the Amsterdam dataset are presented in Table 8. Except mutual information, point entropy and segment entropy take over 9 more hours compared to full training while it significantly reduces annotation efforts which is more time-consuming.

When analysing the IoU for each class in Fig. 17, it can be seen that the IoU values for the classes ground, building and clutter are quite similar for all strategies and the baseline method even has a higher IoU in clutter and building before the sixth iteration. The main differences lie in the IoU for the water where the lines for the three query functions are above the line for baseline. Unlike the Rotterdam dataset where the river is wide and easy to be recognized, the identification of water points by PointNet++ is challenging in the Amsterdam dataset, because the canals in the dataset are crisscrossing and narrow. Our query functions select tiles with more water points and contribute to higher IoU values for the water. This is similar to the selection of more tiles with artwork and clutter improved the performance in the Rotterdam dataset. This suggests that selected samples enrich the model knowledge in difficult classes.

Fig. 18 presents the spatial distribution of all selected tiles according to different active learning strategies. Selected tiles for the baseline method are randomly spread over the training area while the other three queries select tiles located in the southern part of the training area which is dominated by farmlands, different from the densely built-up area in Amsterdam centre. The knowledge of farmlands brings slight advantages over the baseline in the IoU for ground before the third iteration.

4.7.2. Comparison between fine-tuning and training from scratch

Fig. 19 shows the model performance on the Rotterdam dataset when training from scratch and fine-tuning the model from the previous model. It can be seen that fine-tuning can achieve the accuracy comparable to training from scratch but it effectively saves training efforts.

Fig. 20 shows how many updates the model requires during the training. More updates mean a longer time for training. When using all tiles in the training area, the batch accuracy gradually increases with fluctuations which are caused by the randomness of point cloud tiles in each batch. When all models are trained from scratch in each active
learning iteration, the batch accuracy drops dramatically because no previous knowledge is involved and it takes some updates for the model to learn. While fine-tuning requires more updates compared to full training for all active learning strategies, it saves about half of the training efforts comparing to the training from scratch. It can be seen that the model keeps the previous knowledge and avoids low batch accuracy at the beginning of the training in each iteration. Table 9 demonstrates the training time required for training from scratch, fine-tuning and full training. Incrementally fine-tuning is much faster than training from scratch.

5. Conclusion

Existing supervised deep learning networks for semantic point cloud segmentation require a large number of labelled points for training. This research proposes an active and incremental learning workflow to effectively reduce annotation efforts by iteratively selecting informative samples and incrementally enriching the model knowledge. Firstly, point clouds are split into tiles and separated into labelled and unlabelled groups. Then the labelled tiles are used for training. For the initial iteration, the network is trained from scratch. For the rest of the steps, fine-tuning is implemented to incrementally enlarge the model knowledge based on the previous model. In each iteration, after the training, the informativeness of point cloud tiles in the pool of unlabelled training tiles is evaluated by the trained network according to three uncertainty metrics, namely point entropy, segment entropy and mutual information. Both point entropy and segment entropy assess the data dependent uncertainty while the segment entropy considers the interactions among neighbouring points within geometrical homogenous units. Mutual information, which estimates the model dependent uncertainty, is derived from Bayesian networks. The idea is to analyse the disagreements in model predictions caused by the uncertainty of model parameters. The most informative tiles are labelled and added to the labelled training pool for the next training.

The framework is tested on two subsets of AHN3 datasets. Experimental results show that compared to the random selection, all three metrics are capable of selecting informative point clouds like tiles dominated by difficult classes and samples diversifying geometry of target objects in the labelled training pool. Among the three query functions, segment entropy performs the best. For the 7 class classification in the Rotterdam dataset, it takes 31.7% of the whole training area to reach the mIoU obtained from the model trained on the whole training area. When it comes to the 4 class classification in the Amsterdam dataset, it only requires 9% of the whole training area to achieve the full training mIoU. Also, the effectiveness of incremental learning is verified on the Rotterdam dataset. It saves about half of the training efforts comparing to the training from scratch for each active learning iteration.

The proposed framework is successfully tested with two ALS datasets using PointNet++. Although we perform experiments on PointNet++, the three uncertainty metrics can also be applied to many other state of the art network architectures. Point entropy requires the predictive probability for each class and mutual information needs to turn on dropout during the testing. These conditions can be easily met by most of the networks like PointCNN, KPconv and SPG. However, segment entropy, evaluating the interactions among points within segments, can only be applied to point based networks. It is invalid for segment based method, like SPG, where segments are taken as homogenous units and

Fig. 19. Comparison of model performance on the Rotterdam dataset under different training strategies, training from scratch (TFS) and fine-tune (FT). The vertical axis represents the mIoU and the horizontal axis represents the iteration.
points within a segments share the same label. In addition to ALS data, the framework is also possibly generalized to point clouds from terrestrial mobile laser scanners or indoor scenes.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. Yaping Lin acknowledges the financial support from China Scholarship Council. Zhishuang Yang from Wuhan University is acknowledged for the suggestions on code implementation and proof-reading. The authors also acknowledge NVIDIA Corporation for the donated GPUs.

References


