Learning to Clean Heritage Point Clouds

by

Luc Hayward

Dissertation presented for the degree of
Master of Science in the department of Computer Science

UNIVERSITY OF CAPE TOWN

October 2023

Plagiarism Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the required convention for citation and referencing. Each contribution to and quotation in this thesis from the work(s) of other people has been attributed, and has been cited and referenced.
3. This thesis is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.
5. I acknowledge that copying someone else's work, or part of it, is wrong, and declare that this is my own work.

Luc Hayward, October 2023

Signed by candidate

Abstract

Laser scanning technology enables accurate distance measurements between a scanner and the environment producing a grid of points in 3D space. Many laser scans can be registered together to produce a "point cloud". This is commonly used in the Cultural Heritage domain to create highly detailed 3D models. However, the raw point cloud often contains unwanted details and artefacts which require manual removal through a time-consuming process called point cloud cleaning.

Whilst previous works have shown promising results, the application of deep learning in the cultural heritage domain has not been fully explored. In this study, we investigate the use of deep learning models for the binary segmentation of semantically varied cultural heritage sites using a few-shot fine-tuning approach. By relying only on learned geometric information without additional point attributes such as colour, we demonstrate the effective application of modern deep learning approaches in this domain.

We present a method in which the user is presented with an unlabelled scan and asked to label a percentage between 2.5-50% of the scene. This labelled data is used to train the model and predict labels for the remainder of the scene. We compare the performance of three different deep learning models (Pointnet++, KPConv, and Point Transformer) against a baseline Random Forest model. We consider factors such as speed, pretraining, and the amount of hand labelling required.

Our results show that minimal human labelling is needed to provide sufficient data for modern deep learning approaches and simple scenes are efficiently labelled using Random Forests. We achieve up to 90% mean Intersection over Union on the remaining points, with as little as 5% of the scene labelled for training, in under two hours on consumer hardware. This translates to a reduction in manual labelling effort of up to 90%.

Among the models tested, KPConv reliably produces good results with minimal need for human labelling. The remaining models are slower to train and required more human labelling, while the Random Forest is effective only on simple scans, where a machine learning model would be inefficient to train.

Our work demonstrates the potential of deep learning methods for label-efficient and accurate point cloud cleaning in the cultural heritage domain, reducing the time spent on manual cleaning.

Acknowledgements

Thank you to my Supervisor, Prof Patrick Marais, for his guidance and understanding over the course of the last two years. Patrick's (usually) weekly meetings kept me and this project on track.

Many thanks to my co-supervisor, Prof. Dr. Jan Dirk Wegner for providing his insights and inviting me to visit his research group at the University of Zurich and providing additional compute resources. The final stage of this thesis exploring the Point Transformer could not have been realised otherwise. The researchers I met during my time provided invaluable advice for working with transformer models.

Thank you to my fellow researchers in Cape Town for the feedback and entertainment during long days on campus.

Last but not least my family and friends, without whom I would have been far hungrier and far slower to understand the problems I faced.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cultural Heritage sites throughout the world are a source of religious, cultural, historical and archaeological importance. The popularity with both tourists and locals alike makes many sites difficult to reach for study, with rising tourism leading to some of these sites rapidly eroding. Furthermore, natural disasters risk permanently damaging these sites. In 2016 an earthquake in Central Italy damaged several heritage buildings including the Church of Sant'Agostino (Figure 1.1). This 13$^{\text{th}}$ century church now stands damaged beyond repair and can no longer be explored by the public.

To address these challenges, various technologies have been employed in the field of cultural heritage preservation, including 3D scanning, photogrammetry, and remote sensing. One approach is the creation of detailed 3D models which can be used for initial research or in education without the need to travel on-site. These models are often constructed from a collection of 3D points — a point cloud — which densely sample locations on surfaces around the site. In order to extract meaningful information these point clouds may be "segmented", assigning labels to each point indicating some geometric feature or semantic grouping.

FIGURE 1.1: The Church of Sant'Agostino before and after damage

1.1 Point Cloud Segmentation

Point cloud segmentation is a key task in the field of 3D data processing, with wide ranging applications in robotics, computer vision and cultural heritage preservation.

Segmenting point clouds for cleaning is a challenging problem as the data is often noisy. The noise in the raw data can be split into two broad categories. Firstly there are the capture errors in which the scanner incorrectly produces a point in 3D space where it should not exist. This regularly happens at the edge or corner of objects where the scanner sometimes overshoots into the distance creating a broken edge and a "phantom" edge far in the distance. The second common reason for removal, particularly in a cultural heritage domain, is the "modelling decision". Within a scene, there may be objects which are undesirable in the final scan or regions of the data which are too noisy or low resolution to be used for creating the final 3D model. These regions (often large) must be removed and may be geometrically similar to other regions that the cleaner decides to keep. This is a key problem for training a general model as each scene may differ drastically in its modelling and cleaning requirements [98].

Traditional methods for point cloud segmentation have relied on hand-crafted features which can be time consuming to compute and lead to high dimensional data for the resulting model. Additionally, previous work in this space has focused on single scans and used multi-class labels covering different semantic groups such as trees, ground, and walls. In contrast, we first combine the individual scans (known as registration, see Section 2.2) into a single cohesive scene to provide a more realistic and constrained representation of the scene to the model. We further constrain the problem of segmenting the point cloud into two classes, keep and discard. Points labelled "keep" should be retained for model construction, while those labelled as "discard" correspond to noise or unwanted structure.

1.2 Proposed Solution and Motivation

In this thesis, we investigate the use of modern deep learning based methods, as well as a Random Forest model, which have shown promise in the semantic segmentation of large benchmark datasets. These models are evaluated on large, complex scenes rather than individual scans making them well suited for our task. In contrast to previous work by Marais et al. [67], in which multiscale handcrafted features were precalculated from the original point clouds, we rely only on the geometric (XYZ coordinates) as input to a deep learning model. This deep learning model then learns its own representation of the pointcloud in a learned feature space. These features are not necessarily as easily

interpretable by humans compared to handcrafted features. The learned features may be able to extract information from the pointcloud in a way that is non-obvious to humans but is useful to the machine learning model.

More specifically, we investigate several different models: Random Forests[14] [1] (RFs), PointNet++[84][2], KPConv[112] [3], and Point Transformer[127] [4]. Each of these models (where applicable) is tested on our datasets both by training from scratch as well as fine-tuning after pre-training on the Stanford 3D indoor benchmark (S3DIS). This thesis explores the performance of these models across a range of training set sizes and scenes, as well as methods to minimise the amount of human labelling needed to train the model for a given scene. To evaluate the performance of the models we make use of the mean Intersection over Union (mIoU).

Motivated by previous work by Mulder [70], Pocock [82], and Marais et al. [67], we chose not to attempt to train a general model. In Chapter 4 we detail the diversity of our scenes. In general cultural heritage sites are highly varied and the classes of objects to be removed or kept is highly scene dependant and may be contradictory between scenes. As such our method focuses on learning to label each scene in solation using a small subset of the scene as training data.

These experiments are motivated by the rise in popularity of deep learning methods for point cloud semantic segmentation and the move to using fully registered scenes in the cleaning process. This presents a need to revisit previous work and attempt to adapt it to the new workflow using modern techniques.

1.3 Research Questions

With the aim of reducing the time and labelling requirements for cleaning a given scan, we propose to answer the following questions:

1. *Can modern deep learning models be used to clean registered point clouds of cultural heritage sites with greater than 90% mIoU?* As the models chosen are benchmarked on urban scenes, it is uncertain how they will perform on the irregular geometry of cultural heritage scenes. Whilst it may be biased on the complexity of the scene in question, we consider 90% to be a reasonable upper bound.

[1] Sourced from Scikit Learn Random Forest Classifiers
[2] Sourced from Pointnet_Pointnet2_pytorch on Github
[3] Sourced from KPConv on Github
[4] Sourced from point-transformer on Github

2. *Can our models reduce the total labelling effort by more than 50%?* We approach the concept of labelling effort on two fronts (see Section 7.1 for details). Firstly, the proportion of the scene's points which must be labelled by the user, including training labels and correcting the model predictions. And secondly, the total time required for labelling including the use of the model. We consider a 50% reduction in user effort to be a significant improvement to the cleaning workflow.

3. *Is the relative performance of the three deep learning models on S3DIS*[5] *maintained when compared to heritage point clouds?* Different Machine Learning algorithms can perform differently depending on the amount of data available for fine-tuning and the nature of that data (are the classes diverse or similar). When moving from the benchmark datasets to our Cultural Heritage data, the performance of the models may be different than expected.

4. *To what extent does the percentage of labelled training data affect the overall labelling performance?* Smaller models often do not need as much data for training as larger models, but larger models may be able to achieve better performance if given sufficient training data. As data must initially be hand labelled for training there exists a trade off where labelling more data is less useful than simply using a smaller model and fixing any remaining errors by hand.

5. *To what extent does pre-training on the S3DIS dataset affect the overall labelling performance?* The benchmark dataset is scanned in a similar manner to the heritage sites we explore in this paper. However, the geometry of the scenes in the benchmark data is more regular (being an office building) whilst the heritage sites are both diverse and irregular. It remains to be seen whether the pre-training can transfer to a new domain effectively.

6. *Can Active Learning be used to reduce the manual labelling needed in a scene?* The underlying theme of the thesis is to reduce the need for human labelling when cleaning heritage point clouds. Identifying the most efficient regions of a scan to label is a difficult task, and though fast, random selection may produce poor results. Active learning has the potential to improve the accuracy of classifiers by labelling the most informative data first.

7. *Does changing to a registered point cloud directly increase the performance of the Random Forest Classifier when compared to previous work by Marais et al. [67] on individual scans?*

[5] Following the completion of the paper, we were made aware of a Cultural Heritage specific dataset, ArCH[68]

1.4 Scope and Limitations

This thesis focuses on methods for reducing the amount of time a human needs to spend labelling during the cleaning process. There are many classification models which could be explored for the cleaning of point clouds, with new models being released at a regular pace. We focus on deep learning models with a range of model architectures. We acknowledge there is a trade off made between evaluating mature (but potentially less performant) models and newer (but potentially more performant) models. Having reviewed the literature we chose to focus on more established models in PointNet++ and KPConv, in addition to the Random Forest baseline. We include the Point Transformer model as a final addition to our experiments. At the time of testing it was the best performing model with a stable implementation on GitHub.

A true comparison to manual labelling is out of scope since it requires extensive user studies and the availability of trained expert users to ensure accuracy. Instead, we estimate any potential speed increases based on previously reported labelleing time estimates. Lastly, we are limited by the number of scenes provided, five by the ISTI-CNR[6], and one by HumLab in Sweden[7], representing a range of cleaning requirements (see Chapter 4).

1.5 Overview of Thesis

The remainder of the thesis is structured as follows. Chapter 2 provides a high level overview of the point cloud cleaning pipeline from capture through to the cleaned model. In it we provide sufficient background for understanding this thesis and touch on previous approaches in this area. In Chapter 3 we describe the models chosen for this evaluation as well and acknowledge recent advances in the field. In Chapter 4 we present the datasets and highlight the differences between them which allowed for a broad range of scene types to be evaluated. In Chapter 5 we expand on the details of the prototype implementation. In Chapter 6 we explore the effectiveness of Active Learning for selecting training samples for the models. In Section 7.1 we lay out the experimental design as well as the systems used for training and evaluation and the difficulties one needs to take note of when reproducing our results. In Chapter 7 we describe the experimental design as well as the systems used for training and evaluation and the difficulties one needs to take note of when reproducing our results. This is followed up with a discussion and evaluation of our results. Finally in Chapter 8 we reiterate our conclusions and make suggestions for directions in which to begin exploring future work.

[6] https://www.isti.cnr.it/en/
[7] Stefan Lindgren, HumLab, University of Lund, Sweden

Chapter 2

Background

The Zamai Group [125] is an initiative aimed at digitally preserving cultural heritage sites in Africa using 3D models, making use of laser-scanned point clouds as described below.

Recent work involving the Zamani project has attempted to improve and automate the cleaning process for these heritage point clouds. These approaches were based on non-deep learning methods such as Random Forests (RFs)[14], Support Vector Machines (SVMs)[16], and simple Multi-Layer Perceptrons (MLPs)[95]. They focused on hand-crafted features and operated on individual scans rather than fully registered point clouds [67, 72, 82]. This allowed for faster computation times per scan but does not incorporate an overall view of the final point cloud.

A preliminary study by Mulder [71] motivated the use of random forests to help automate the classification of points in a scan provided initial "seed regions" labelled by the user. Pocock [82] continued to work on individual scans with handcrafted features, and developed a method of automatically selecting representative training scans. The results suggested that RFs with a diverse set of features produced the highest overall accuracy and inspired further work by Marais et al. [67] in which they explore user-guided online training of random forest classifiers.

The lack of registered scans in these previous studies begs the question of whether improved performance is as simple as pre-registering the scans, allowing more efficient user interaction. Additionally thus far Random Forest classifiers have been focused on; with modern deep learning approaches left unexplored.

The following sections provide the background on machine learning as it relates to this thesis such that the research can be understood without the need for prior experience.

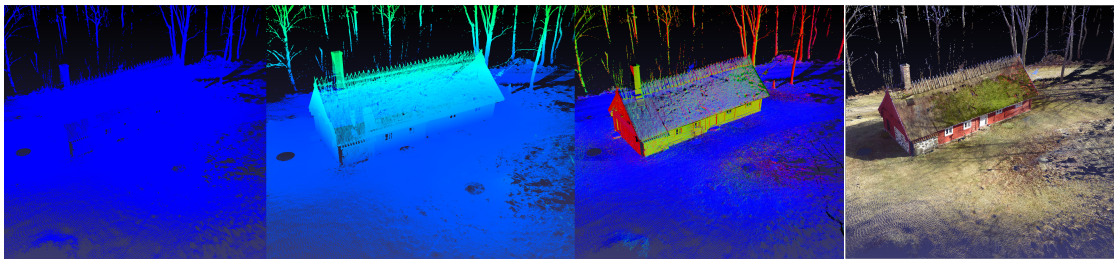


FIGURE 2.1: Example of additional data included with the raw XYZ points from the Lunnahoja site. From left: Raw cloud, false coloured by Z coordinate, normals, and RGB

Topics include semantic segmentation, machine learning, and artificial neural networks (ANNs). Later chapters provide more details as and when they are needed.

2.1 Point Clouds

A point cloud is a set of points in 3D space. Each point has an XYZ coordinate as well as potentially other information that may be included during the creation of the point cloud. This can include colour (as RGB data), intensity (a measure of how reflective a surface is) or other scalar values [31] (Figure 2.1).

LiDAR (Light Detection and Ranging) is a method for creating these point clouds. Using pulses of light these scanners can derive the distance from a scanner to an object's surface and thus create a 3D representation of the scene (see Section 2.2).

Point Cloud Features refer to any additional scalar values attached to each point. Features can be captured during the scanning process (such as colour in the form of RGB data) or computed and added to the point cloud offline. Computed features typically describe the local or global properties of the point cloud. Local features include normals or surface descriptors such as planarity and roughness. These features can be used to provide additional information to the segmentation models.

2.2 Scanning Procedure

LiDAR was first developed in the 1960s for use in meteorological research and has since been adapted for use in drones, self-driving cars (Figure 2.2) and of course 3D modelling pipelines [18, 45, 60].

LiDAR enambles the distance to a surface from the scanner to be derived. The two main methods for outdoor large-scale scanning are known as time-of-flight (ToF) and phase

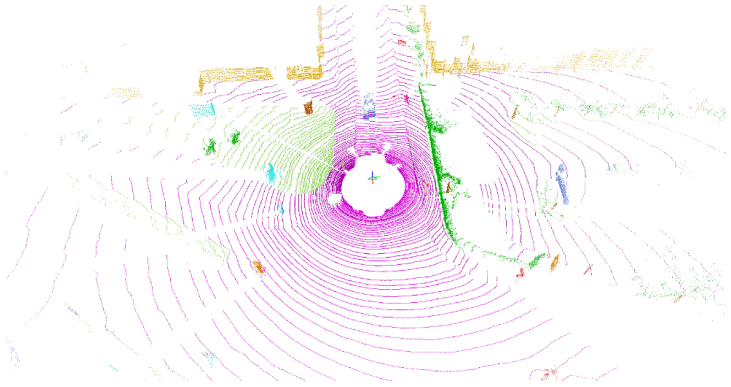


FIGURE 2.2: A single frame from a car-mounted LiDAR scanner [6]

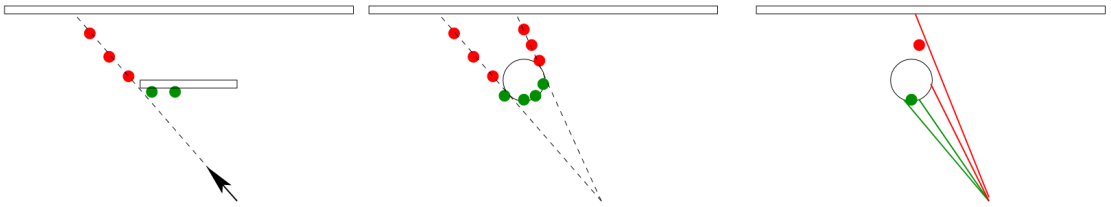


FIGURE 2.3: Mixed pixel phenomena: green points are valid, red points are invalid. Sourced from [115]

shift scanners. Both of these scanners are based on deriving the amount of time taken for a laser pulse to return to the scanner after hitting a surface. TOF sensors make use of a pulsed laser and a receiver to measure the amount of time taken from when the pulse is emitted until it is detected again to derive the distance to the surface of the object. Phase shift scanners emit a constant laser beam in multiple known phases. The phase difference between the emitted and reflected waves is then measured and can be directly related to the distance to the object.

ToF sensors have a much longer range than Phase Shift sensors, however, for cultural heritage campaigns, the newer Phase Shift scanners have become the preferred method as they are significantly faster to work with. Where ToF scanners used to take the Zamani group two to three hours per scan, Phase Shift scanners can produce scans in only three to five minutes. This lets a single operator perform over 100 scans in a single day reducing the time needed to complete a campaign [97, 98].

Both approaches suffer from issues with edges being poorly scanned and decreasing resolution as distance increases. One such issue is the "mixed pixel phenomenon" (Figure 2.3). When a laser strikes the edge of an object it may only partially cover the edge, resulting in the return beam being averaged from the closer "correct" surface and a farther "incorrect" surface [115]. This results in what should be a consistent thin line of points representing the corner of an object having "holes" where some points appear in the correct position and others are at a distance behind the edge.

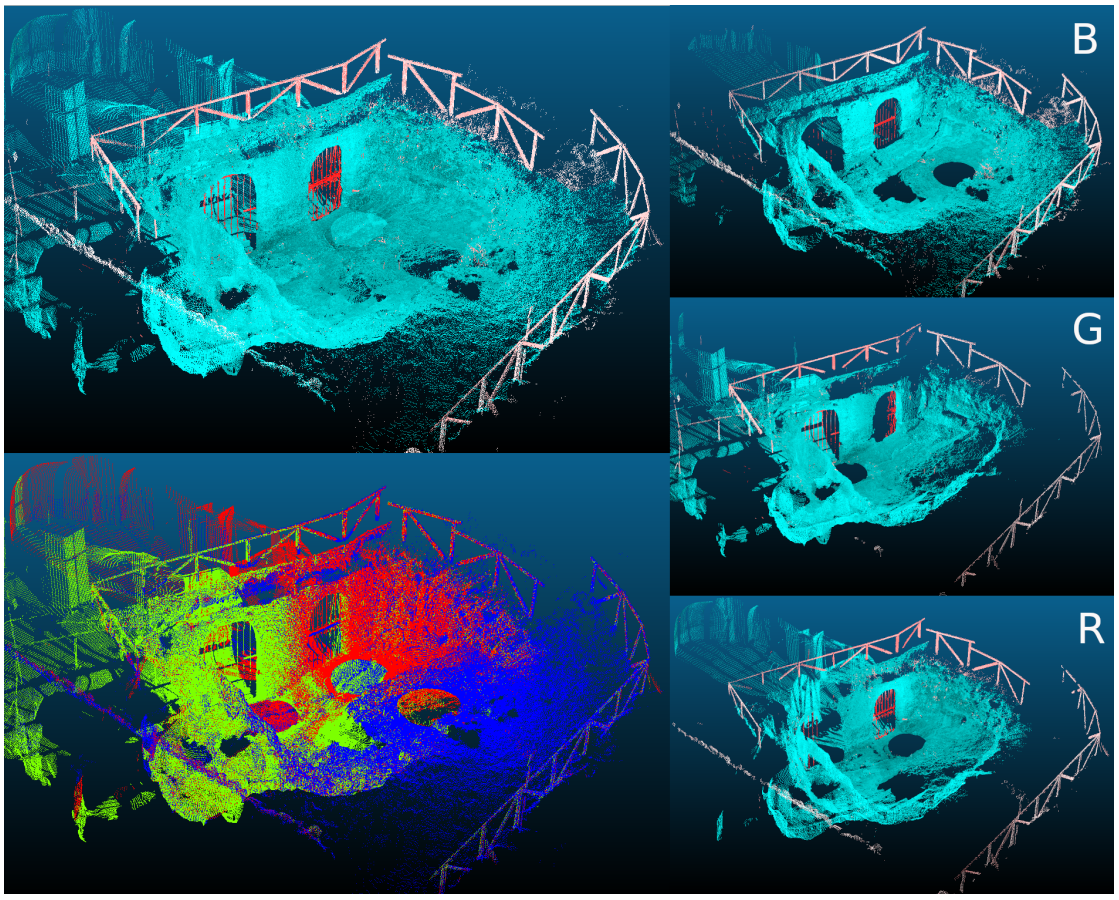


FIGURE 2.4: A subset of the "Church" scans illustrating 3 individual scans being merged together. Blue points are labelled "keep", red points are labelled "discard". Top Left: merged scans. Bottom Left: merged scans in false colour as blue, green, and red. Right: Individual scans corresponding to blue, green, and red from the top down.

2.3 Modelling Pipeline

The process of creating a point cloud for a new site involves acquiring scans from a number of positions in order to ensure adequate coverage of all the surfaces [98]. The scans are then registered (Figure 2.4) and cleaned to remove all unwanted points. The next stage in the pipeline (Figure 2.5) is the modelling procedure. The focus of this thesis is on improving the cleaning phase after registration.

Registration is the process whereby many individual scans are merged together into a single unified coordinate space. This allows objects to be seen from multiple perspectives and provides a more uniform density in the final point cloud. A common method for doing this is known as Iterative Closest Point (ICP) [9]. ICP requires that there be overlapping regions between each of the scans to be registered and that these overlapping regions be "well scanned". In this case "well scanned" means that the surfaces must have

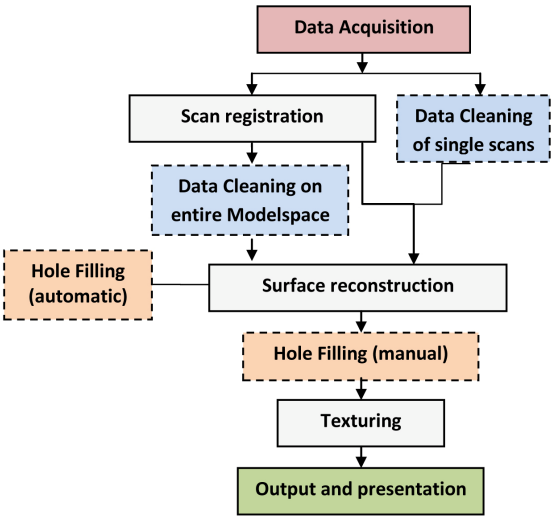


FIGURE 2.5: Zamani Group scanning and modelling pipeline [98]

sufficient detail and not be scanned at a steep angle which could introduce inconsistencies and reduces the effective resolution of the scans in that area.

This process can be made easier by using "registration targets" which are placed around the site and provide specific points of reference for registration [11]. In a heritage context, it may not be possible to place these targets due to a combination of difficult to reach locations and the need to preserve sensitive sites [96].

Modelling begins with running a surface reconstruction algorithm before creating a mesh for the model. During this, it is often necessary to fill holes left in the model where the laser scanners were unable to view the surfaces. This most often happens when there are surfaces above where the scanner can see such as balconies and overhanging roofs [98].

Texturing is done by projecting photographs onto the model. Although many scanners now contain cameras of their own or have tools to allow easy capture of panoramic images during the scanning process, the Zamani group found this to be problematic to use. During the course of scanning a site, the lighting will change due to changes in weather or time of day that a scan is performed leading to inconsistent colours in the photos. It is more efficient to take the pictures separately from scanning using conventional cameras or utilising drones to ensure that all the photos are taken under the same lighting conditions. Some experimentation has been done with integrating structure from motion and photogrammetry into the texturing pipeline but at the time they were not found to be useful enough to regularly use [97].

FIGURE 2.6: Example image segmentation task, sourced from SAS

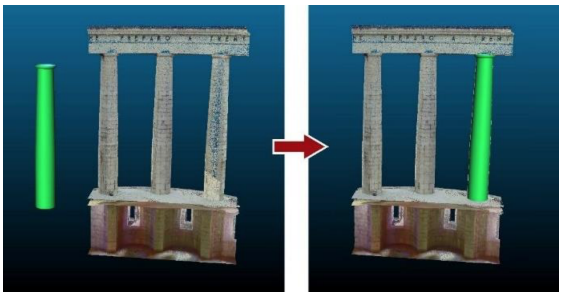


FIGURE 2.7: Fitting a cylinder to a set of points for segmentation, sourced from Grilli et al. [36]

2.4 Semantic Segmentation

Semantic segmentation is the task of assigning labels to pixels in an image, or points in a point cloud, that represent semantically meaningful concepts (Figure 2.6). This might be based on shared features such as colours and textures. In point cloud segmentation, labels might be based on geometric properties such as position and surface features. Semantic segmentation is potentially a more complex task than classification (see Section 2.5) or object detection. Each pixel/point in an image/point cloud must be assigned its own label, rather than only a single label for the entire image/scene. However this is dependent on the use case, available data and classes in question.

Point cloud segmentation differs from image segmentation in several ways. Points are less uniform, with changing densities depending on how the scan was captured and whether the point cloud is from a single scan or multiple registered scans. This can make feature extraction more difficult.

Various methods can be used for point cloud segmentation including conventional techniques and machine learning approaches. Conventional techniques include model fitting, region growing, and clustering; although these may be limited in their ability to handle complex scenes [122].

Model fitting methods attempt to group points based on geometric features and simple shapes such as planes or spheres, grouping points together that are part of the same shape (Figure 2.7). RANdom SAmple Consensus (RANSAC) [27] is a common method with good performance and can be effectively used for tasks such as plane extraction (e.g.ground plane extraction) [99].

Region growing involves combining neighbouring points together to divide a scene into dissimilar segments. This can be done in a top-down (unseeded) manner by iteratively subdividing segments until some threshold is reached (e.g. minimum number of points in a segment). Conversely, bottom-up (seeded) region growing begins with a set of initial seeds and "grows" the segments. Region growing is sensitive to the initial seed choice and the number of segments to create [77]. Region growing has been used for segmenting planar regions in sparse point clouds [83] and segmenting building features and surfaces [117].

Clustering methods such as K-Means and hierarchical clustering form segments based on grouping extracted point features and can be thought of similarly to region-growing methods, although clustering is typically done on extracted features [36].

Machine learning approaches have shown promising results in a variety of scenarios. Many previous approaches for segmenting heritage point clouds involved handcrafted features and/or worked only on individual scans (rather than fully registered) [37, 67, 72, 82].

2.5 Machine Learning

Machine learning is a subset of "Artificial Intelligence" centred around teaching machines to learn from data without explicitly programming the rules or algorithms to be learned. Fundamentally it is the extracting of patterns from raw data in order to make predictions on new data. Examples of machine learning use cases are the spam filter on an email or deciding whether an image contains a dog or a cat. We call this the machine learning "model" [31, 35].

There are several sub-categories to the field of machine learning but they can be broadly classified as supervised and unsupervised learning. The key difference between these two is whether the data is labelled or not [35]. In unsupervised learning, the model learns to extract patterns from raw unlabelled data. In the example of classifying email as spam or not, an unsupervised model would be given a large number of emails and would cluster these into similar groups based on the patterns picked up. Based on this a researcher could look at the emails in one of the clusters and infer that all the emails in that cluster were either spam or not spam (see Section 2.5.2 for details). On the other hand supervised learning involves learning from a large set of labelled examples. This can be broken down further into classification and regression models, of which we will focus on the former.

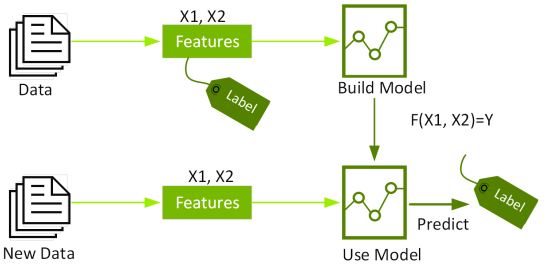


FIGURE 2.8: A supervised model is trained on some data, and then used to predict labels for unseen data with the same features (via NVIDIA)

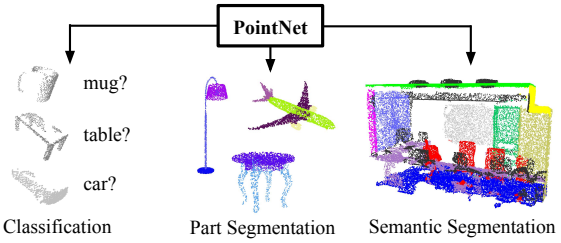


FIGURE 2.9: The three types of point cloud classification (via Qi et al. [84]). Not specifically related to the PointNet Model.

Classification Models learn to predict a label or class based on a predefined set of labels present in its training data (Figure 2.8). For example, a model that learns to classify handwritten digits into numbers from 0-9 as in the MNIST dataset [56].

There are a plethora of classification models available, each having its own pros and cons and suited for different tasks or different types of data. We will describe the types of models explored in this thesis in more detail in Chapter 3.

Point Cloud Classification labels points based on geometric or semantic features. The term point cloud "classification" can be used to describe three related but distinct types of models [84]. Figure 2.9 illustrates the three types, although they are not specifically related to the PointNet model itself. In the literature classification typically refers to the process of assigning a single global label to a point cloud. For example, given a point cloud of a chair, it would assign the label "chair".

"Classification" is also used to describe "part segmentation" models. In this case, a model applies a label to each point in the cloud, grouping similar features together. In our chair example, this might give all the points for the legs one label, for the seat another and for the back a third.

Lastly, there is the "semantic segmentation" model. In this case, each point in the scene is assigned a predefined label categorising it as belonging to one of several classes. These classes are "semantic labels". This is used in scenes with more than one object in, for example, a classroom in which the "semantic labels" might be chair, wall, floor, table etc. In semantic segmentation, no distinction needs to be made between different instances of each semantic label (i.e. two chairs will both be labelled "chair", rather than "chair 1" and "chair 2").

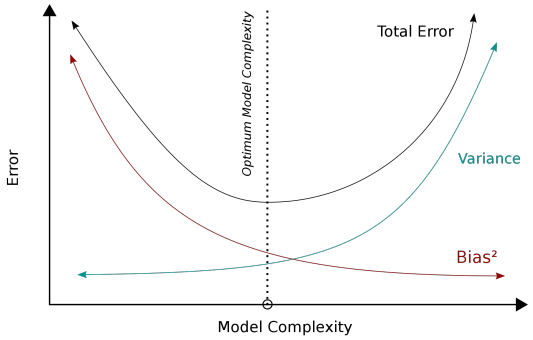


FIGURE 2.10: As model complexity increases variance increases and bias decreases and vice versa

Ensembling is one method of addressing the overfitting problem (see Section 2.5.1) and can be divided into two categories: bagging and boosting. By combining predictions from different models together, the models should be unlikely to make the same errors on new data, and a better final prediction can be made.

Bagging methods, such as Random Forests (see Section 3.2), independently train multiple models and combine the predictions. On average this improves predictive performance by reducing the variance [13]. Boosting methods, such as AdaBoost[28] or XGBoost[15] (see Section 3.2) train multiple weaker models sequentially where each model is trained primarily using the examples that the previous model got wrong. On average this reduces the bias of the combined models [109]. When using decision trees as the base model, ensemble methods can use shallow trees (sometimes called decision stumps) or more complex, deeper trees. The optimal choice is dependent on the task at hand [29, 39].

Early Stopping It is often observed that large models, with the ability to overfit the data, will continue to improve on the training data for a long time. However, the performance on the validation data will begin decreasing at some previous point in the training.

By returning to the model at the point where the validation performance was greatest (or the loss was lowest), it is hoped that the model will also have better performance on the test set. This is achieved by saving the weights of the model each time the validation set performance reaches a new maximum. The only downside to early stopping is the need for a validation set, meaning the model will not see every training sample available [35].

2.5.1 Regularisation

In certain situations, a model may "learn" the training data too well, fitting an overly complex function that matches the noise in the dataset rather than more meaningful patterns. This is known as overfitting and results in a model that does not generalise to unseen data correctly [110]. This is analogous to a student rote learning for a test. Provided the student sees the same questions in the test they will do well, but as soon as the questions change slightly the student cannot get the correct answer.

Regularisation is the process of managing the complexity of a model to reduce overfitting whilst still allowing the model to learn enough to be useful. This is known as the bias-variance trade-off (see Figure 2.10).

A model with high variance is sensitive to changes in the training data, overfitting to the noise in the data leading to poor generalisability on unseen data. On the other hand, a model with high bias is said to have underfit the data and is unable to learn the important patterns in the training data sufficiently. Decreasing the variance of a model typically increases the bias. The model must have low enough bias to fit the training data as well as low enough variance to generalise to new data. Several methods exist and are often used in conjunction with each other to produce the desired level of regularisation [49].

Common and easily implementable approaches are L1 and L2 regression, dropout, and model size adjustments.

L1 and L2 regularisation each add a term to the loss function that penalises large weights. Both introduce a small amount of bias, in turn reducing the variance of the model. L1 regularisation (also known as lasso regression) is useful when some of the weights may not be useful for the prediction, while L2 regularisation (also known as ridge regression) is more useful when all the weights should have at least some influence. In practice, L1 can be helpful in dealing with outliers but L2 is more commonly used [35, 76].

Mathematically, given the simple cost function in Equation (2.1), we can illustrate the L1 and L2 normalisation terms in Equation (2.2) and Equation (2.3) respectively:

$$Loss = Error(y, \hat{y}) \tag{2.1}$$

$$Loss = Error(y, \hat{y}) + \lambda \Sigma_{i=1}^{N} |w_i| \tag{2.2}$$

$$Loss = Error(y, \hat{y}) + \lambda \Sigma_{i=1}^{N} w_i^2 \tag{2.3}$$

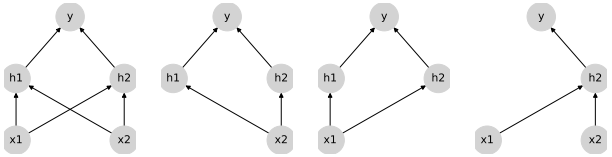


FIGURE 2.11: Example of dropout applied to a small MLP, implemented as the "removal" of non-output neurons. Sourced from Goodfellow et al. [35]

Dropout is a simple technique where a random subset of the non-output neurons are "hidden" or set to zero during each training step (see Figure 2.11). This effectively prevents hidden layer neurons from becoming co-dependent and forces them to "learn" features independently [41, 106, 107]. DropConnect is a more general case in which the weights are "dropped" rather than the neurons but is otherwise functionally the same [118]. This reduces overfitting (aka variance) by introducing noise into the training process. Dropout can also be used to approximate ensembling and uncertainty estimation of a network's predictions [32] (see Chapter 6).

Model size can be used to control the variance by reducing the number of trainable parameters. In an ANN this is done by reducing the number of hidden layers (shallower) or reducing the size of the hidden layers (narrower) [33]. For a given size of model increasing the available training data will typically decrease the variance [12].

2.5.2 Clustering

Clustering is a form of unsupervised learning in which data points are "clustered" together based on similarities in their features without the need for specific labels. A common use case for clustering is for dimensionality reduction, attempting to compress the information into a smaller representation (see Section 2.5). In the context of point cloud segmentation, clustering can be used as a preprocessing step to help identify meaningfully similar structures or regions together [54, 80, 121].

K-Means K-Means is one of the simplest and most popular methods for grouping unlabelled data into k clusters, where k is chosen by the user [63, 66]. It clusters the data by attempting to group samples into groups of equal variance, minimising the distance between the centroids and the data points in each cluster. The centroids need not (and typically are not) points from the dataset, but exist in the same dimensional space.

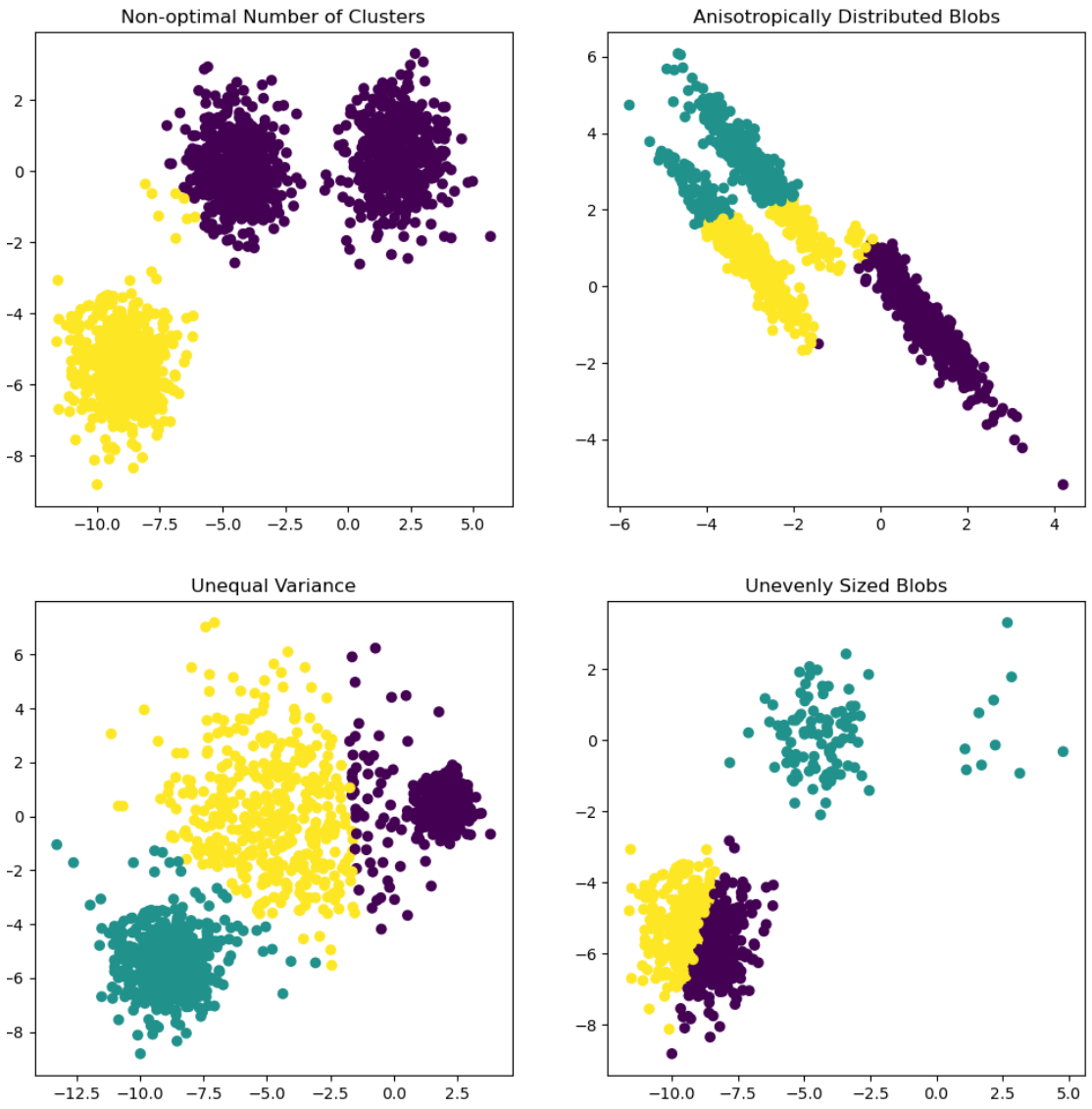


FIGURE 2.12: Examples of poor clustering by k-means due to various issues in the initialisation of data, arbitrary axes. Sourced from Scikit-Learn.

Many algorithms exist for initialising the centroids. A simple method is to select K random points from the dataset as the initial centroid positions. However, this runs the risk of initialising the centroids close together which can produce poor results. Many methods have been proposed to alleviate this such as density-based initialisation and Furthest First initialisation [5].

K-means makes the assumption that the clusters are convex which can lead to issues when the clusters are more elongated or have irregular shapes (Figure 2.12).

The "elbow method" [21, 113] is a heuristic approach to selecting the optimal number

of clusters, k. In order to measure how "good" a cluster is, one can measure the intra-cluster similarity using the Within-Cluster Sum of Squares (WCSS):

$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} Distance(x - \overline{C_i})^2$$

where k is the number of clusters, C_i is the set of data points assigned to the i^{th} cluster, x is a data point in C_i, and $\overline{C_i}$ is the mean of C_i.

By plotting the WCSS as a function of the number of clusters k, and identifying the "elbow" point where the rate of decrease in WCSS slows down significantly. The theory is this is the point where adding another cluster does not significantly improve performance, as measured by WCSS, however, this is considered to be a subjective and unreliable heuristic.

The Silhouette score [92] is another measure of cluster quality, the intuition being that not only should points within a cluster be cohesive, but they should also be distinct from points in other clusters. The Silhouette score for a data point x_i is defined as:

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

Where a_i is the average distance between x_i and all other data points in the same cluster, and b_i is the average distance between x_i and all data points in the nearest cluster (other than the one to which x_i belongs). The silhouette score for a dataset is the mean of the Silhouette scores of all the points.

By combining both the Silhouette score and the Elbow method, researchers can better estimate the number of clusters to use.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [26] is a density-based clustering algorithm that can handle clusters with arbitrary shapes and sizes. This is in contrast to k-means which requires clusters to be linearly separable. Unlike k-means, DBSCAN automatically determines the number of clusters. DBSCAN groups points using a distance measure and density threshold. "Core" points are chosen to be points that have a minimum number of neighbours within a given radius and core points that are within each other's neighbourhood are connected.

2.6 Artificial Neural Networks

ANNs are a type of machine learning model consisting of layers of interconnected nodes, conceptually similar to the interconnected neurons of the brain. ANNs are widely used to perform tasks from image recognition to natural language processing and beyond. Mathematically an ANN attempts to approximate a function F^* that maps some input X to some output Y by learning the parameters θ as can be seen in Equation (2.4). It does this by iteratively adjusting the weights of the connections (aka parameters) between the different layers in order to minimise some loss function during the training process [35].

In this thesis, we cover three types of neural network architectures. Namely multilayer perceptrons (MLPs), Convolutional Neural Networks (CNNs), and Transformers. The following sections cover the background of these architectures, see Chapter 3 for more on how they can be applied in the context of point cloud segmentation.

2.6.1 MLPs

MLPs are the typical type of neural network and the simplest to explain. An MLP consists of an input layer, some number of intermediate or "hidden" layers, and ends with an output layer. Each layer is made up of a variable number of neurons. Each neuron can take a variable number of inputs and outputs a single value. The output of each neuron in a layer is typically connected to the input of each other neuron, called a fully connected network. An example MLP network is illustrated in Figure 2.13.

$$F^*(X, \theta) = Y \tag{2.4}$$

Each layer of the MLP is made up of several neurons. Each of these neurons applies a mathematical function on an input x_i to produce an output y_i. This is called the activation function. In an MLP the function, $F*$ from Equation (2.4), can be decomposed into several functions which are composed together to give the final result Y. These composed functions are equivalent to the layers in the network.

Consider an MLP with two inputs, a hidden layer with three neurons, and a single output node. We could decompose this network as $F^*(x) = f^2(f^1(x)) = y$ where each layer is a function acting on the output from the previous layer.

The leftmost layer, the input layer, consists of a set of neurons $\{x_i|x_1, x_2\}$. Each neuron in the hidden layer transforms the previous layer by taking the weighted sum $w_1x_1+w_2x_2$, followed by the activation function. In practice for efficiency, this is done by taking the

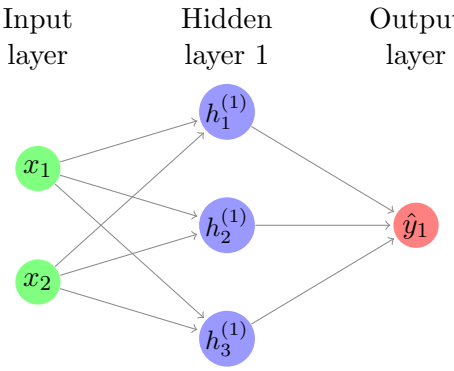


FIGURE 2.13: A simple MLP

outputs of the previous layer as a vector x and the weights connecting the two layers as a matrix W where each element w_{ij} represents the weight connecting the i-th input neuron to the j-th hidden neuron. To compute the hidden layer we calculate the matrix product $\mathbf{W}^{(1)}\mathbf{x}$. We then apply the activation function to each element of the vector to obtain the new output vector x for this hidden layer. This process is repeated until a final output is produced. This is called the forward pass for the network, see Section 2.6.5 for details on the reverse of this operation and the key to learning for neural networks.

2.6.2 Activation Functions

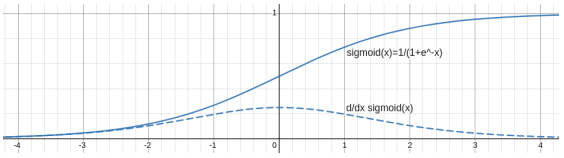
In theory, any differentiable function could be used as the activation function for a neuron, but in practice, it is important to use a non-linear function. If the network used linear activation functions then composing these together would have the same results as a single linear function. Non-linear activation functions allow deep neural networks to model more complex functions than linear models can. Provided the neural network has at least one hidden layer then it can be proven to be a "universal function approximator". However, for this to be true it may require the hidden layer to be unreasonably wide leading to a network that would be too difficult to train. This is known as the Universal Approximation Theorem, the specifics of which are beyond the scope of this paper [20, 43].

Activation functions should be differentiable, computationally inexpensive, and non-saturating. Differentiable to allow the calculation of gradients, computationally efficient as they must be evaluated at every node in the network, and non-saturating to help prevent issues with exploding or vanishing gradients. Early networks often used the sigmoid (Equation (2.5)) or *tanh* (Equation (2.6)) activation functions, however, this is typically no longer used (except at the output node) as they are examples of saturating functions. Saturating functions are considered to make training more difficult though there is still some contention on the matter [123].

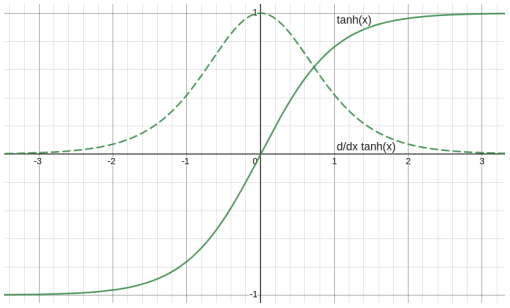
$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.6}$$

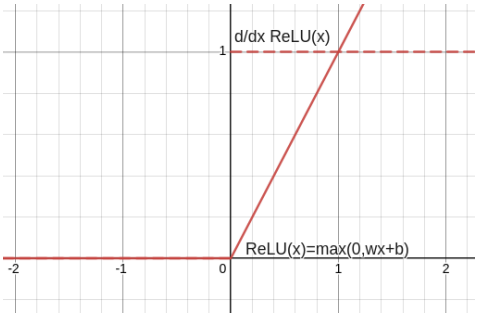
$$ReLU(x) = max(0, x) \tag{2.7}$$



(a) Sigmoid/Logistic function



(b) Tanh (Hyperbolic Tangent)



(c) ReLU function

FIGURE 2.14: The sigmoid, tanh and ReLU activation functions and their derivatives (dashed lines)

A saturating function is defined as any function where large positive and negative inputs are asymptotically bounded. Intuitively this looks like a function where the curve "flattens out". We can formalise this as:

$$(|lim_{x \to -\infty} f(x)| = \infty) \vee (|lim_{x \to \infty} f(x)| = \infty) \tag{2.8}$$

The ReLU activation function (Rectified Linear Unit [75], Equation (2.7)) is an example of a non-saturating function which is commonly used in deep neural networks today. Popularised by Krizhevsky et al. [51] to train a deep (for the time) CNN on the ImageNet dataset [22], they estimated ReLU to be the most important addition to their model architecture at the time. Deep CNNs with ReLU train several times faster than an equivalent network using sigmoid or *tanh*.

ReLU improves both empirically and theoretically over the sigmoid function by being cheaper to compute (the exponent in the sigmoid function is relatively expensive) and non-saturating. Although strictly speaking it is non-differentiable at the cutoff value, in practice libraries will simply define that point as having a gradient of either zero or one [10].

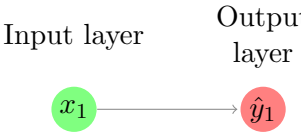


FIGURE 2.15: A minimal network with no bias unit $y = mx$

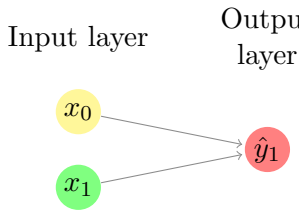


FIGURE 2.16: A minimal network with bias unit

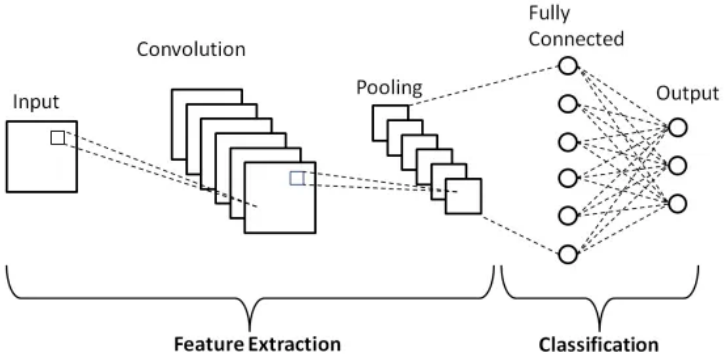


FIGURE 2.17: Outline of a CNN showing convolutions operating on a region of the input, followed by pooling and a fully connected layer to produce a classification. Sourced from Techipedia.

In practice, each layer (except the input layer) usually includes a "bias" term which is a trainable scalar parameter. The bias term is a single node in each layer that always has the value 1 and is connected to every node in the following layer. By adding this bias term the network is able to shift the activation function.

Consider the network in Figure 2.15 below with one input and one output and the identity as the activation function (i.e.a network representing a straight line function through the origin). The output of the network is given by multiplying the input (x_1) by the weight. By altering the weight value the network can alter the slope of the line but it will always pass through the origin when the input is 0.

For some data, this may be acceptable but other data may require the network to output zero when x is some other value. This cannot be done by changing only the weight (the slope of the graph), there needs to be a way to shift the network left or right. This is done by adding the bias term, represented by x_0 in Figure 2.16:

In this network, the bias term is performing the same role as the "vertical shift" term in the equation for a straight-line graph.

2.6.3 CNNs

CNNs were popularised for image recognition tasks in LeCun et al. [56] and remain the de facto model choice for many computer vision problems in deep learning, although recently transformer models (see Section 2.6.4) have been adapted for use on image problems with great success [24]. The introduction of diffusion models for image generation is beyond the scope of this paper.

The advantage of CNNs for image tasks over MLP models is two-fold:

1. The number of parameters in the model can be greatly reduced through weight sharing. This reduces complexity, speeding up training times.

2. A CNN encodes in its structure certain prior assumptions which are useful for working with image data. Namely, that images' statistical properties are *invariant to translation*, a photo of a cat is still a photo of a cat even if the cat is shifted slightly. CNNs take advantage of this by computing the same feature or convolution (a hidden unit with the same weights) across different locations in the input image. Convolutional networks exploit this to force the network to extract local features early in the network.

CNNs often compose several convolutional and pooling operations, ending with a fully connected layer to produce the final output (in the case of classification networks, see Figure 2.17). At each layer, the network abstracts further from the input image by learning new feature maps. When dealing with images early layers have been shown to learn low-level features, edges and colours, with later layers learning higher-level abstractions that may not be human interpretable [78].

2.6.4 Transformers

Introduced in Vaswani et al. [116] for use in machine translation and English constituency parsing, Transformers have largely replaced Long Short-Term Models (LSTMs) [42] as the de facto model architecture for language modelling tasks. Whilst transformer networks are mainly used for processing sequential data such as natural language, they have also been effectively applied to non-sequential tasks such as image processing as in Vision Transformer [24]. LSTM models, and the RNNs [25] they build upon, require hidden states for each timestep or element which must be computed sequentially. This led to slow training times and the problem of vanishing/exploding gradients (which LSTMs help to alleviate). Conversely, Transformers allow for parallel computation of

input elements via the attention mechanism, which allows them to learn how different elements of the input sequence influence each other.

The self-attention mechanism computes the weighted sum of the input elements for each output element, where the weights are determined by how "relevant" each input element is to the output. This allows attention to capture both local and global dependencies as well as handle variable length inputs and outputs.

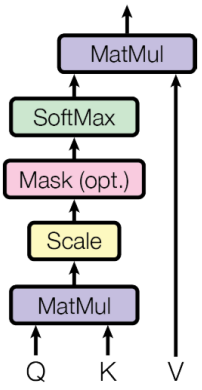


FIGURE 2.18: Scaled Dot-Product Attention [116]

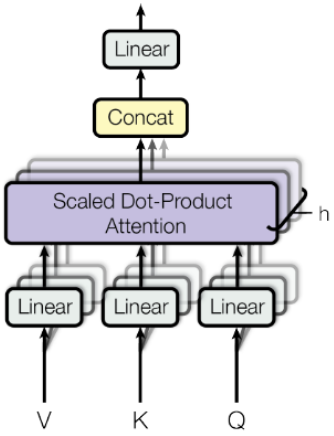


FIGURE 2.19: Multi-Head Attention [116]

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{dimensionofvectorK}})V \tag{2.9}$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.10}$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.11}$$

The attention function starts with the three vectors, $Query$, Key, and $Value$, calculated for each input element by a learnable linear transform of the embedding. For each word, multiply the *query* with every *key* and compute the $Softmax$. Multiply the result with that word's *value* vector to give the new word embedding (see Figure 2.18).

In practice, the q, k, v for each element are combined to give Q, K, V matrices for more efficient computation as in Equation (2.9).

Multi-head attention (Figure 2.19) is simply several single-head attention layers computed in parallel as in Equation (2.10). These heads learn different representations (including different linear transformations to produce the q, k, v vectors) and can attend to different aspects of the input sequence. Thus allowing the model to be more effective. The resulting vectors are concatenated and reprojected to the expected dimensionality.

Positional encoding is another important part of transformer models, allowing the model to make use of the order of the input sequence. The positional encodings can be learned or handcrafted, and are added to the input embeddings before being passed to the attention block. Vaswani et al. [116] show that both learned and handcrafted encodings produce equivalent results.

For more detail on how the attention mechanism can be applied to point cloud segmentation, see Section 3.5.

2.6.5 Backpropagation

After making the forward pass (see Section 2.6.1), the output of the network is likely to be incorrect. The backpropagation algorithm [95] allows the network to adjust the weights in such a way as to reduce this error. This error is calculated by the loss function, a measure of how "incorrect" the output is. This process of minimising the loss is known as gradient descent.

Backpropagation works by calculating the derivatives with respect to the weights at each layer by means of the chain rule. The recursive application of this chain rule is a limitation on deep neural networks as explained in Section 2.5.1 as repeated multiplication leads to the gradients (the derivatives) becoming very small or very large, leading to the weights not updating or becoming unstable respectively.

In practice gradient descent can be extended to mini-batch and stochastic gradient descent. The specifics of these are not necessary to explore at this stage.

2.6.6 Transfer Learning

Transfer learning is a technique that allows pre-trained models to be reused to improve the learning capabilities on a target task (Task B) by making use of the learning that has already occurred on a related task (Task A). This can reduce training time and lead to better performance than training from scratch [124].

As Deep neural networks learn increasingly abstract representations of their input data through successive layers [8], the assumption is that these abstract features will be broadly transferable to different but related tasks.

Transfer learning covers three main types, feature extractors, fine-tuning, and multi-task learning [79]. In feature extraction, the classification head is retrained for the target task whilst freezing the weights of the earlier layers. Fine-tuning uses the pretrained model

as the initialisation for the target task, allowing the whole model to be trained on the new data. Multi-task learning involves training over multiple tasks simultaneously. One might also consider "domain adaptation", the performance of a model on Task B having trained only on Task A, as a subset of transfer learning as well [7, 114].

2.7 Related Work and Existing Systems

Prior to the introduction of PointNet [84] (see Section 3.3), deep learning on 3D data was often done by transforming the points and using traditional CNN based approaches. This could be done using voxel grids[1] with 3D CNNs, however, these networks are computationally expensive and much of the voxel space will be empty or represent few points in the original cloud. This is known as a sparse grid. An alternative approach to classification (of the entire point cloud) involves rendering representative 2D images and applying traditional 2D CNNs to classify them, however, this approach is not suitable for semantic segmentation as the individual points are not recoverable [69, 85, 108].

Manual segmentation tools allow users the highest degree of control but are time consuming to use. Lasso (or polygon) selection allows the user to draw two-dimensional regions around points on the screen, however, hidden points behind the intended selection may be selected accidentally. This problem is exacerbated in complex registered scenes. An alternative method is by using a selection brush allowing the user to "paint" over points.

In addition to previous tools produced for Zamani, general point cloud tools exist including semi-automated approaches. These can speed up certain tasks by clustering based on distance and density. Meshlab[2], CloudCompare[3], and Leica Cyclone[4] are only a few examples of the existing software available for point cloud processing tasks.

Laser scanners produce denser point clouds near the scanner which become sparser as the points become farther away. This results in clustering many small, meaningless clusters on the periphery of the point cloud (Figure 2.20). Some programs automatically segment points into certain categories such as ground points, vegetation and buildings, however, such tools often do not work in a heritage domain and misclassify points as vegetation due to the uneven nature of the buildings' surfaces (Figure 2.21)[71].

[1] A voxel can be thought of as a "3D pixel", representing a value on a regular grid in 3D space in the same way pixels do in 2D

[2] Available at https://www.meshlab.net

[3] Available at https://www.danielgm.net/cc/

[4] Learn more at https://leica-geosystems.com/products/laser-scanners/software/leica-cyclone/leica-cyclone-register-360/simplified-point-cloud-processing-software

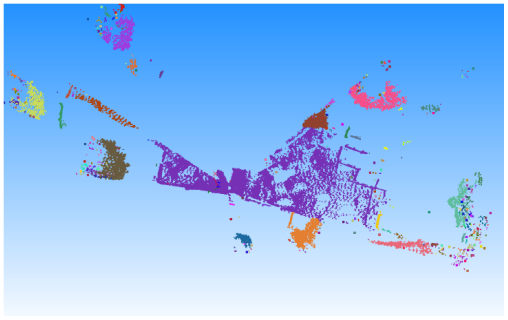


FIGURE 2.20: Distance clustering may struggle to produce meaningful clusters (via 3DReshaper)

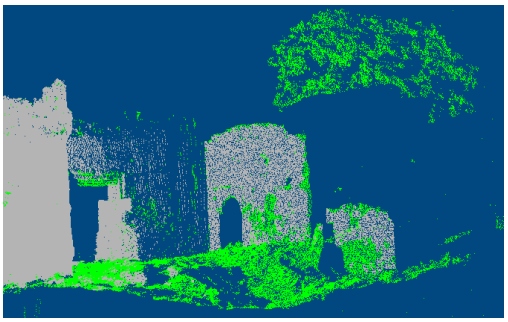


FIGURE 2.21: Automated segmentation tools for vegetation may incorrectly classify irregular building surfaces as foliage (via Pocock [82], VRMesh)

2.8 Summary

This chapter has provided a brief overview of the background on the deep learning techniques we felt were most needed to allow the reader to understand the methodology and models used in this study. This is not intended to be an exhaustive or comprehensive background on all of the deep learning topics related to this thesis but provides the necessary baseline understanding for continuing with understanding. We describe the broad approaches to point cloud processing and existing manual and semi-automated segmentation tools. The following chapter provides further detail into the specific model architectures that will be used in our experiments.

Chapter 3

Neural Networks for Point Cloud Segmentation

This chapter provides the background for the models chosen for the study and positions them within the context of point cloud segmentation. In Chapter 5 we will provide more detail on our use of these models.

First, the naive-baseline decision tree classifiers are described, followed by the three core deep learning models. PointNet++, KPConv, and Point Transformer, represent three distinct model architectures, based on MLPs, CNNs, and Transformers, respectively. We chose these models as each is, or was at its release, near state of the art (SOTA) on the Stanford 3D Indoor Scene Dataset (S3DIS) [4] (Table 3.1) and represents a different network architecture.

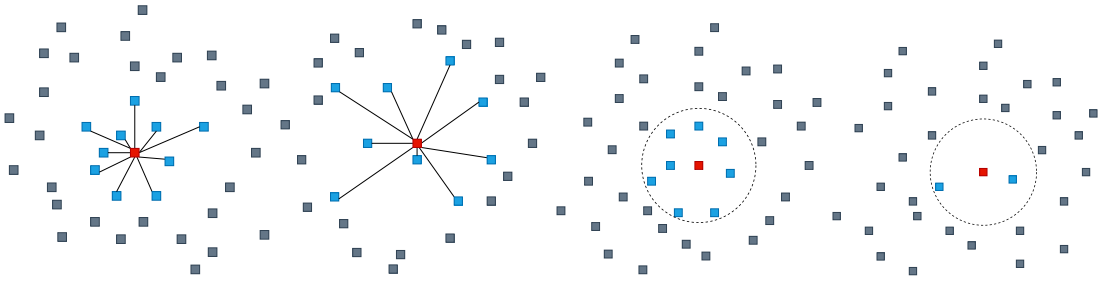
3.1 Neighbourhood Queries

Neighbourhood queries are an essential component of modern point cloud processing networks. They enable the network to aggregate local points or features together at each abstraction layer of the network, providing contextual information about the structure and relationships within the point cloud.

There are two main approaches to obtaining neighbouring points: K-Nearest Neighbours (kNN) [17] and Ball Query [86] illustrated in Figure 3.1. The neighbourhood obtained by kNN has a fixed number of points, but the neighbourhood volume is variable. In dense regions, the neighbourhood will be small and in sparse regions, the neighbourhood may be large. Conversely, ball query limits the neighbourhood to a fixed radius but does not guarantee a number of points. In dense regions, there may be a large number of points

TABLE 3.1: Semantic Segmentation results for S3DIS Area 5 for mIoU, mean class accuracy, and overall accuracy (OA). Methods are in chronological order, best results are in **bold**, and models that we use in our experiments with are underlined.

Method	S3DIS Area 5		
	mIoU%	mAcc%	OA%
PointNet [84]	41.1	49.0	-
PointNet++ [86]	53.5	-	83.0
RF_MSFF [111]	49.8	-	-
KPConv [112]	67.1	72.8	-
Point Transformer [127]	70.4	76.5	90.2/8
PointNet++ (PointNeXT) [87]	63.2	-	87.5
PointNeXt-XL [87]	70.5	-	90.6
StratifiedFormer [52]	72.0	78.1	**91.5**
Point Transformer + PAGWN [119]	71.4	77.9	90.5
StratifiedFormer + PAGWN [119]	**72.2**	**78.2**	91.4



(a) kNN selecting 10 nearest points

(b) kNN selecting 10 nearest points in a sparse region

(c) Ball Query selects points in a fixed radius

(d) In sparse regions ball query may not have enough points to select neighbours

FIGURE 3.1: Comparison of kNN and Ball Query neighbourhood selection methods, red is the centroid, blue is additional selected points

while in a sparse region, there may be none at all. Models using ball query are better able to handle non-uniform point densities as it ensures that the local context for a point is a fixed size in Euclidean space[40, 86, 112]. Both PointNet++ and KPConv below use ball query for their neighbourhood lookups for this reason. Point Transformer uses kNN ($k = 16$), to the best of our knowledge there is no clear reason for them to have not used ball query as other attention-based methods use ball query [81, 128].

3.2 Tree Ensemble Models

We explore both Random Forests [14] (introduced in Section 2.5) and XGBoost models [15] as naive baselines.

A distinction between tree based and neural network based classifiers which will become apparent in our results is the different classification boundaries they can create. Whilst

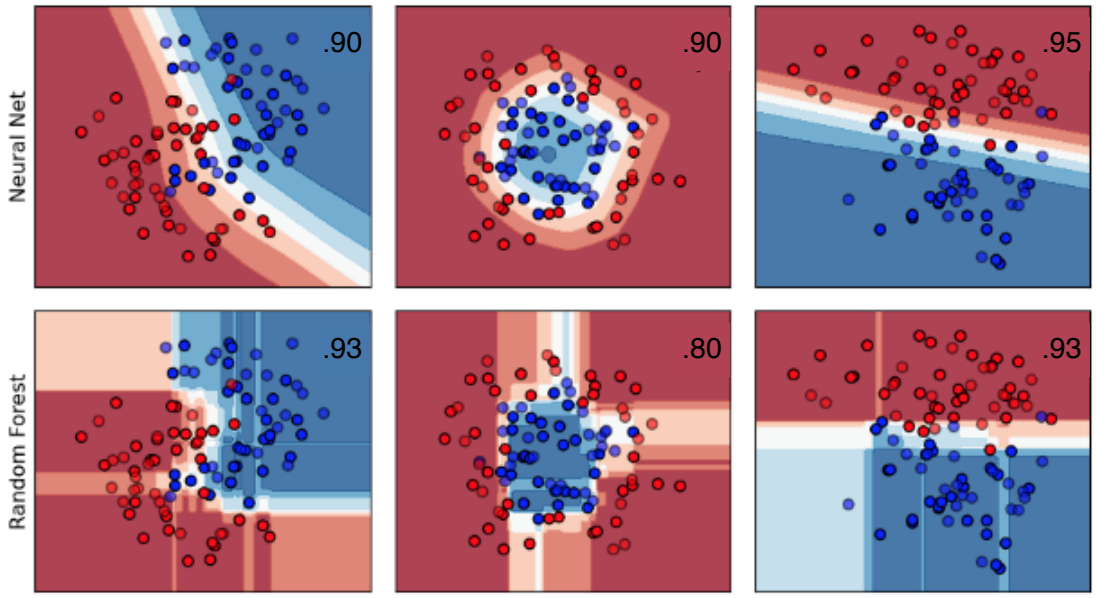


FIGURE 3.2: Toy datasets illustrating the "straight" decision boundaries of Random Forests compared to the "smooth" boundaries created by neural networks (via Scikit-Learn)

neural networks are able to create "smooth" decision boundaries, standard decision tree based models create what is known as an Axis Parallel Decision Boundary (Figure 3.2). Alternative methods exist including boolean combinations of multiple features, oblique decision trees, and non-linear decision trees [47, 73, 120]. These methods are beyond the scope of this thesis but represent potential alternatives for further research.

This means that the decision boundary is a hyperplane perpendicular to the feature axis used to split the tree at that node. For our purposes, this means that the decision boundary always lies parallel to the XYZ axes. If the true boundary is not parallel to these axes it will create a complex boundary structure, making it difficult for the model to generalise to the remainder of the scene.

Random Forests (RFs) train multiple decision trees independently on random subsets of the dataset (also known as bagging) and combine the outputs for improved accuracy. Although individual trees are expected to have high variance and overfit their training data, the model's combined variance, and thus the likelihood of overfitting, is minimised.

We make use of Scikit-Learn's implementation[1] which averages the probability score from each tree instead of the original voting method. This implementation introduces a second source of randomness by considering only a subset of the input features each time a tree is split. Although it was not necessary for our experiments, Nvidia's CuML

[1] See Scikit-Learn

framework provides a drop-in replacement for running on the GPU[2] which can provide $20 - 40x$ speedups. We show in Chapter 7 that the Scikit-Learn implementation is sufficient to enable training and prediction in less than seven minutes and typically less than five on our consumer hardware (see Appendix A.2), which we consider to be fast enough for efficient experimentation. The train-predict loop itself takes between ten and one hundred seconds depending on the scene and amount of training data, enabling near real-time interaction with the labeller.

XGBoost is a fast and popular library which combines decision trees with gradient boosting. AdaBoost [28] was the original boosting method presented for binary classification. Gradient Boosted Decision Trees generalize the concept of boosting to use gradient descent [30].

The algorithm starts by training a weak model, M_1, on all the samples in the dataset. During subsequent training iterations, samples that the previous models predicted incorrectly are given a higher weighting. The goal is to minimise the bias and likelihood of underfitting. XGBoost can be further accelerated using GPUs during both training and inference and has extensive support for distributed training.

The most recent work by Marais et al. [67] used RFs for semantic segmentation of LiDAR scans. As RF models have a limited capacity to learn feature abstractions in the way that deep learning models do, RFs must be augmented with hand-crafted point features (see Appendix C). Marais et al. [67] use both ball query and kNN depending on the neighbourhood feature being calculated. An additional cylindrical neighbourhood is used to calculate explicit height features. Multiscale point features were incorporated by downsampling the point cloud and recomputing the features at a coarser resolution. This was effective as they worked with one scan at a time rather than a fully registered point cloud. However, other papers that work with the fully registered point clouds commonly adopt a ball query (spherical neighbourhoods) sampling strategy, which can similarly incorporate multiscale sampling if needed [111].

RF models make for an effective baseline as they are inexpensive to train and do not require a GPU to run effectively[3]. By including these models we are able to better compare against the previous works on the problem of automating the cleaning of heritage point clouds.

[2] Nvidia Technical Blog

[3] GPU accelerated versions of Random Forests do exist but they are not "required" in the same way as deep learning models require a GPU.

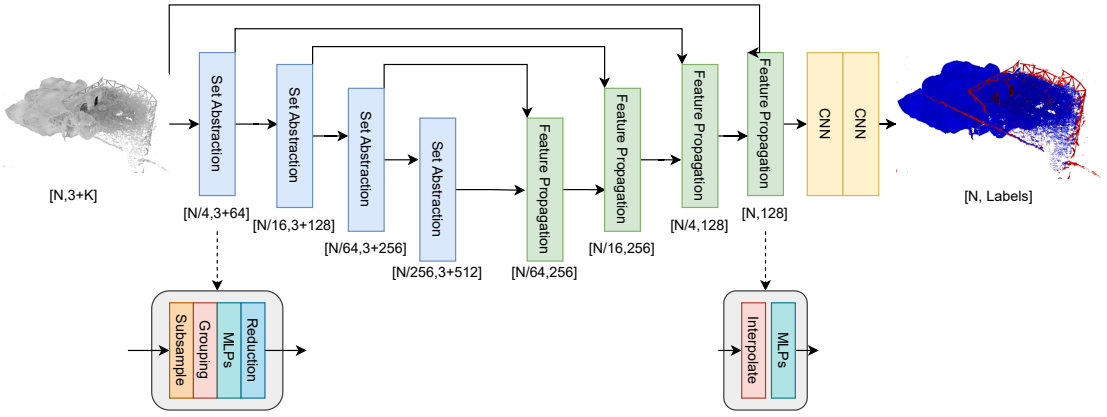


FIGURE 3.3: PointNet++ segmentation model architecture

3.3 PointNet++

Whilst numerous previous approaches exist based on voxelisation and 2D projection of the points, PointNet [84] marked the start of viable deep learning models operating directly on the point data for large 3D point clouds and was followed up shortly after by PointNet++ [86]. PointNet++ improves on PointNet by taking into account local neighbourhood information with hierarchical spatial structures, allowing the model to account for both local and global contexts.

The segmentation version of PointNet++ has a U-Net [91] like architecture (Figure 3.3). The model takes in a set of N points and downsamples these using ball query (or kNN) to select neighbouring points, followed by an MLP and MaxPooling layer to aggregate local features (the set abstraction blocks). The point features are then interpolated back to the original N points through a series of upsampling and concatenation steps (the feature propagation blocks with skip connections). Finally, the labels are predicted for each point using a classification head.

Whilst PointNet++ is a simple and efficient architecture which many future models use as a backbone, its performance suffers compared to newer, larger models described below.

3.4 KPConv

KPConv represents a mature model (last updated in mid-2022) based on point convolutions. When released, it achieved SOTA results on several point segmentation and classification benchmarks, including S3DIS with an Area 5 mIoU of 67.1% (at the time of writing it is within 5% of the SOTA).

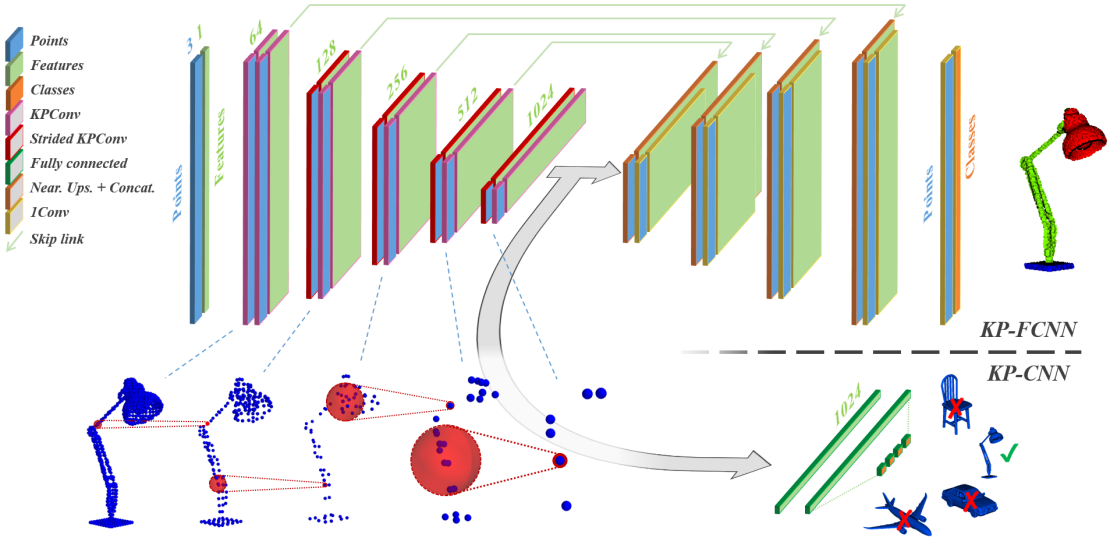


FIGURE 3.4: Representation of the KPConv architecture. The top path shows the semantic segmentation network, whilst the bottom path shows the classification network. The bottom left shows the progressive downsampling of the input points. (sourced from [112])

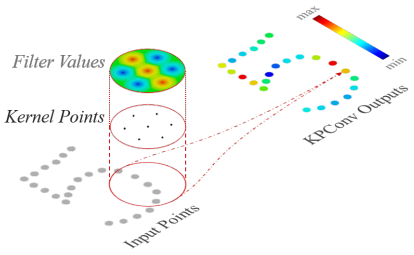


FIGURE 3.5: KPConv convolution demonstrated on 2D points. (sourced from [112])

KPConv is an efficient CNN-based approach for point cloud classification and segmentation tasks [112]. Unlike PointNet++ and Point Transformer, KPConv utilizes convolutional layers specifically designed for point cloud data.

A novel Kernel Point Convolution is introduced where the weights are located in Euclidean space and applied to points close to them, making the network point order invariant. The weights are continuous and learnable during training (Figure 3.5). This is in contrast to prior CNN approaches which required the sparse point cloud data to be projected onto a regular 3D grid. This is computationally expensive as much of the scene contains no point data [69, 89].

Furthermore, the authors introduce a *deformable* kernel, where the kernel points can be learned by the network, theoretically allowing the model to adapt to more complex geometries. However, for this thesis, we focus on the rigid convolutions.[4]

[4]Recommendation after personal communication with Hugues Thomas (author), July 2022

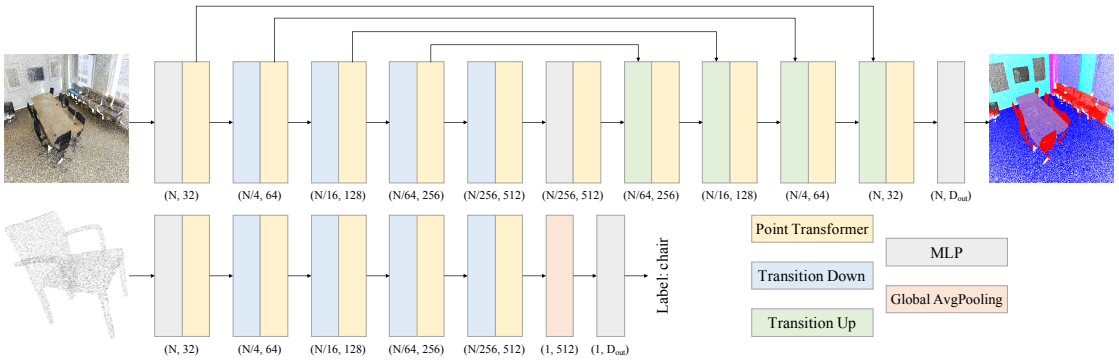


FIGURE 3.6: Point transformer network for semantic segmentation (sourced from [127])

KPConv is designed to consider point geometries as "structural elements" whilst any additional features are the "real data". In order to ensure that points each retain a baseline level of importance in the network, a constant feature is added to each point regardless of other features that may be recorded.

The architecture follows a U-Net style approach consisting of a series of convolutional and pooling layers followed by a fully connected layer (Figure 3.4). Regular downsampling paired with ball query is used to improve robustness in non-uniform density regions of the point cloud.

KPConv has the advantage of more successfully capturing local structures in the point cloud, however, its complexity requires a more powerful GPU for training compared to PointNet++. It also enforces a grid-subsampling to reduce the effect of noisy or non-uniform point densities.

3.5 Point Transformer

Point Transformer [127] is an attention-based method which, at the time of release, was SOTA on S3DIS Area 5 and the first to exceed a 70% mIoU with 70.4%[5] The self-attention mechanism, which allows the model to weigh the importance of different parts of the input against each other, can be used to capture the relationships between points in 3D space.

Several previous works [23, 57, 62] have explored using global dot-product attention for point cloud models. However, this method is too computationally expensive for large scenes containing millions of points. Inspired by recent self-attention-based image models, Point Transformer applies self-attention on local regions around each data point and makes use of vector attention [126] which is able to "attend" to each feature channel

[5]reproduced at 70% by POSTECH_CVLab

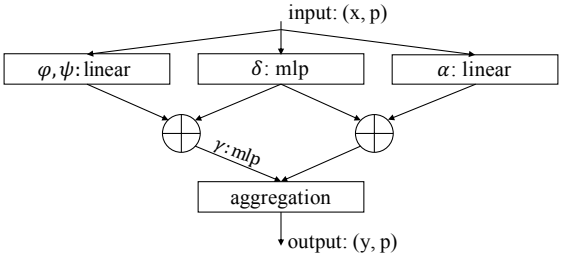


FIGURE 3.7: Point transformer layer. Here, x, y are feature vectors, and p represents the point coordinates. The input feautre-point pair (x, p) are processed through two linear transformation branches and a MLP. The outputs are aggregated to produce the final feature-point output (y, p). (sourced from [127])

individually. This increased performance by $> 5.5\%$ over standard scalar attention used by Vaswani et al. [116], although it may not be as effective in our experiments which do not make use of additional features beyond position.

The Point Transformer architecture (Figure 3.6) is once again a U-net style model, conceptually similar to the previous two models. Unlike PointNet++ and KPConv, the authors choose to use kNN instead of Ball Query for the neighbourhood lookups.

The model consists of multiple point transformer blocks, each containing a point transformer layer (Figure 3.7) between two linear layers. The model uses a learned positional encoding parameterised by the difference in the point coordinates for each pair of points i and j: $\delta = \Theta(p_i - p_j)$. An ablation study showed this to be an improvement of 3.9% over using absolute positions of the points.

An advantage of the Point Transformer model is its flexibility in processing a variable number of input points, rather than a fixed number as in PointNet++ and KPConv. The model is also able to scale to tens of thousands of points during inference in under one second. When training on S3DIS the model is designed to segment an entire room, $\pm 80k$ points, in one pass. During our experiments, we found that with a 12GB GPU, the model can handle $\pm 150 - 160k$ points.

An important disadvantage of the point transformer is the large amount of memory and training data needed to achieve good performance. For example, training the model on the S3DIS dataset takes two days on four RTX3090s; in contrast to KPConv and PointNet++ which take 22 hours and two days respectively on a single RTX3080. An RTX3090 is approximately $15 - 30\%$ faster than the RTX3080.[6]

[6]Based on estimates from AIME, Digital Trends, Linus Tech Tips – RTX3090 review

3.6 Recent Advances

Several new models have been released since the conclusion of our experiments which improve upon the Point Transformer on the S3DIS benchmark. These include PointNeXT [87] and Stratified Transformer [52]. PointNeXT builds upon PointNet++ and suggests that the majority of improvement from newer models can be attributed to improved data augmentation and optimisation techniques, allowing larger models, rather than any particularly impactful architectural change. Colour Drop, from KPConv, is one example in which colour information is randomly dropped (set to zero) on each point. This accounts for a 5.9% mIoU improvement on its own and validates our approach of using only geometric information in our experiments. Qian et al. [87] hypothesise that this forces the network to focus more on the point geometry. Without changing the architecture they achieve an increase in mIoU for S3DIS Area 5 of 9.5%. However, PointNeXT is more computationally expensive and memory-intensive than Point Transformer.

The Stratified Transformer[52] proposes capturing long-range dependencies by sampling key points more densely the closer they are to the query point in the attention model. This provides the model with a greater effective receptive field [65], allowing it to examine a larger area around a point of interest. This comes with a minimal increase in computation cost. Building on Point Transformer and Stratified Transformer, Wang et al. [119] introduce *Window Normalization*. By normalising the attention weights within each local window separately, the model is more robust to variations in point density, reducing the influence of noisy points. This improves performance on smaller object segmentation. The authors show that their Pre-Abstraction Groupwise Window Normalisation module (PAGWN) can be applied to existing architectures (Point Transformer and Stratified Transformer) to directly increase mIoU scores by 0.8% and 0.5% respectively.

We believe that the improvements of the newer models would not significantly affect the outcome of our experiments, although these could be explored in future works. The changes proposed in PointNeXT are of particular interest as they demonstrate the majority of their performance via improved data augmentation strategies.

3.7 Summary

In this chapter, we have provided an overview of the models used in our study, including Random Forests, PointNet++, KPConv, and Point Transformer. We explained the motivation behind their selection, and the known trade-offs, and provided a brief overview

of their working mechanisms. Furthermore, we have mentioned recent advances in the field, which could be considered for future studies. In the following chapter, we provide a detailed explanation of our dataset used in the experiments for this study.

Chapter 4

Datasets

In this chapter, we describe the six scenes examined in our experiments. The dataset covers a range of environments, including three urban and three rural scenes, and the percentage of points to be discarded ranges from 5% to 95%. These are the same scenes as those used by Marais et al. [67], however, we use the fully registered point clouds instead of treating them as individual scans. The Lunnahoja scene is provided by Stefan Lindgren from HumLab[1] with ISTI-CNR[2] group in Italy provididing the remaining scenes. The point clouds are provided as collections of scans in the .ptx[3] format. This is an ASCII format which includes the registration matrices needed to align each collection into a single point cloud. We perform this alignment in CloudCompare[4]. The initial data received did not include the correct registration matrices, this slowed down initial experimentation and was resolved after several months.

Each point cloud consists of XYZ point coordinates and per-point labels for keep and discard. Many of the scans include intensity (a measure of the reflectance of the surface) and the Lunnahoja scene also includes colour and normal data. As mentioned in Section 1.1, the keep and discard labels are chosen by hand by expert labellers. The choice of what features or objects should be kept or discarded is partly practical and partly artistic in nature. Practically, points representing noise or errors in the scanning process as well as malformed objects such as moving people and animals are generally removed. This ensures that the scan can be cleanly converted to a 3D model at later stages in the processing pipeline as seen in Figure 2.5. Further to this, the discard points include any region of the scene which is not considered to be "important" in the final model. Depending on the focus of a scene large regions may be removed from a scan even when

[1] University of Lund, Sweden
[2] https://www.isti.cnr.it/en/
[3] See Photon Wiki
[4] Available at https://www.cloudcompare.org/

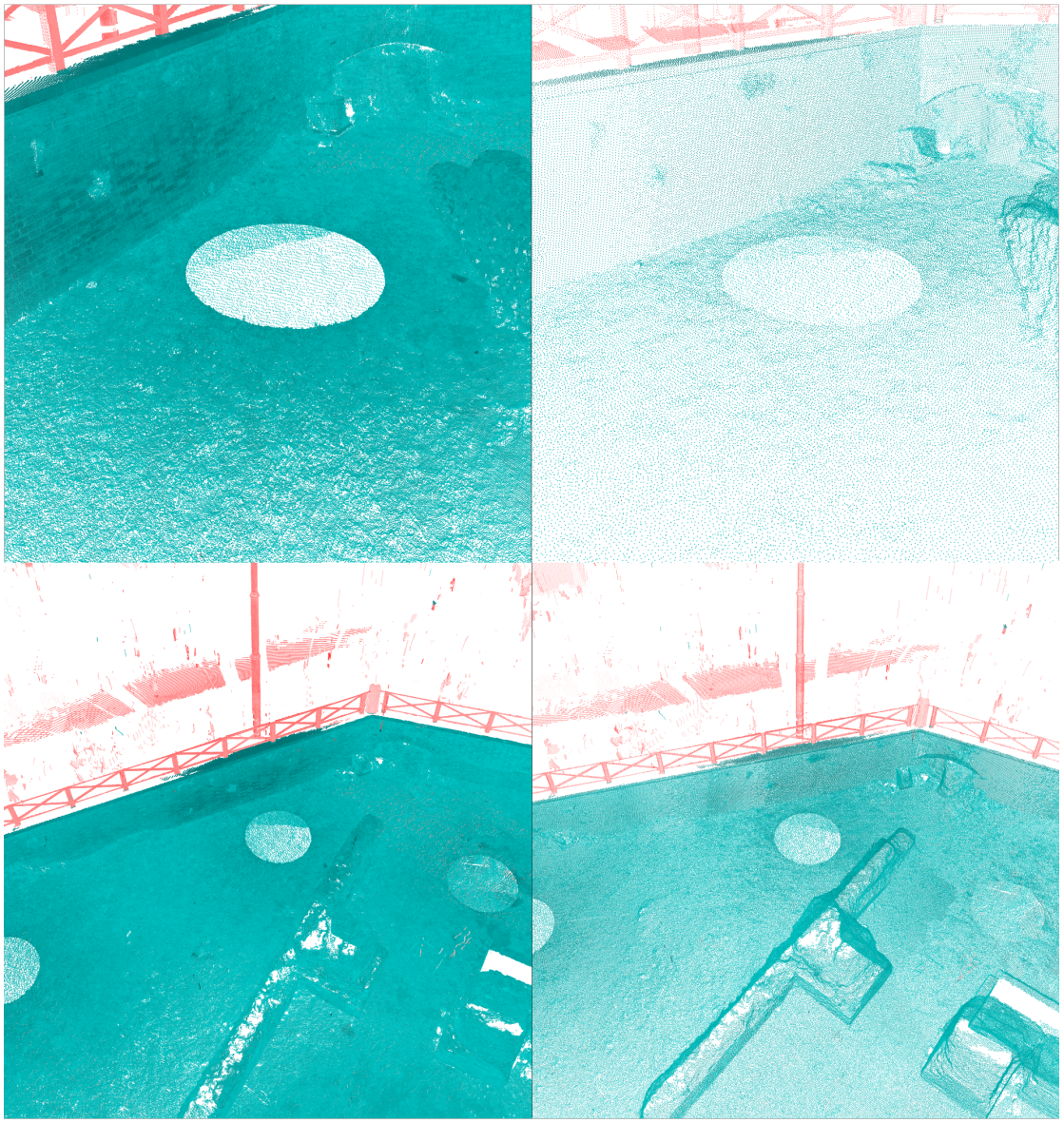
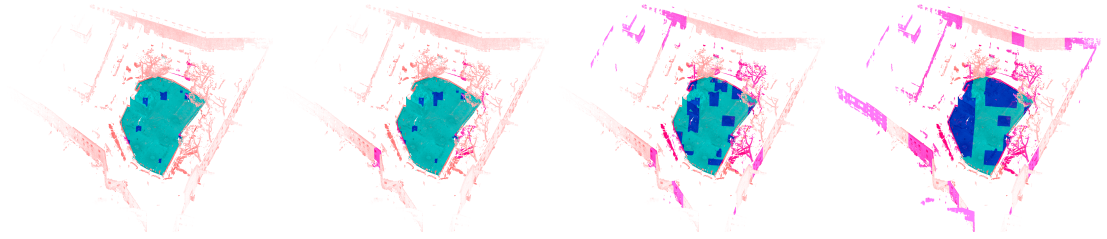


FIGURE 4.1: Comparison of the original (left) and subsampled (right) point cloud density (on Bagni di Nerone). Red points are labelled discard, blue are labelled as keep.

those regions contain features that would otherwise be kept in another portion of this scan or in another scene where the focus was different.

Due to the large size of the original point clouds, which hinders efficient processing on consumer hardware, we first subsample each scene to a minimum point-to-point distance of 2cm. The labels in the subsampled point cloud are assigned to the majority label from the nearest neighbours in the original cloud. We show empirically in Table 4.1 and through visualisation in Figure 4.1 that the subsampled point clouds can be reprojected onto the original point clouds using nearest neighbour reprojection with at least 98% accuracy. As such we are confident that the subsampling did not adversely affect our



(a) 2.5% (b) 5% (c) 25% (d) 50%

FIGURE 4.2: Bagni di Nerone: Each training split is a strict subset of the next. Dark blue/pink indicates the training set whilst light blue/pink indicates the target labels to be predicted. As above, red represents discard whilst blue represents keep.

results.

We preprocess the point clouds to remove isolated points. Using a $1m^2$ grid in the XY plane, any cells containing fewer than 100 points are removed. This removes less than 0.1% of points in all scenes without incorrectly discarding any points. For each point cloud, we create four training/test splits. These splits are determined by area to better approximate labelling effort, as point density is generally unrelated to this effort. Using the column grid method described in Section 5.2, we select 5%, 10%, 25% and 50% of the scene, where each training split is a strict subset of the next (Figure 4.2).

Early experiments showed that it was infeasible to create a third representative split for validation or to randomly select samples for the training set. This is due to how scenes are typically focused around a central point of interest and often have imbalanced label and geometry distributions.

Instead, we choose the splits by hand, aiming to mimic a real-world interaction where the labeller is asked to select "representative and informative" regions for labelling, ensuring the labels are not visible during selection. Building on this scenario, a "cleaner" would be likely struggle to select training and validation sets that are both representative of the entire scene while requiring only a minimal proportion of the scene for labelling. Previous approaches, such as those by Lin et al. [61], neither count the validation set in their labelling budget nor make use of small datasets where this issue would arise. They use a 2×2km scene whilst our scenes are on the order of 100×100m.

In the following sections, we provide visualisations of the scenes and describe any important features or irregularities in the ground truth labellings. For each of the "label" visualisations, blue denotes "keep" points and red denotes "discard" points. Additional visualisations can be found in Appendix E.

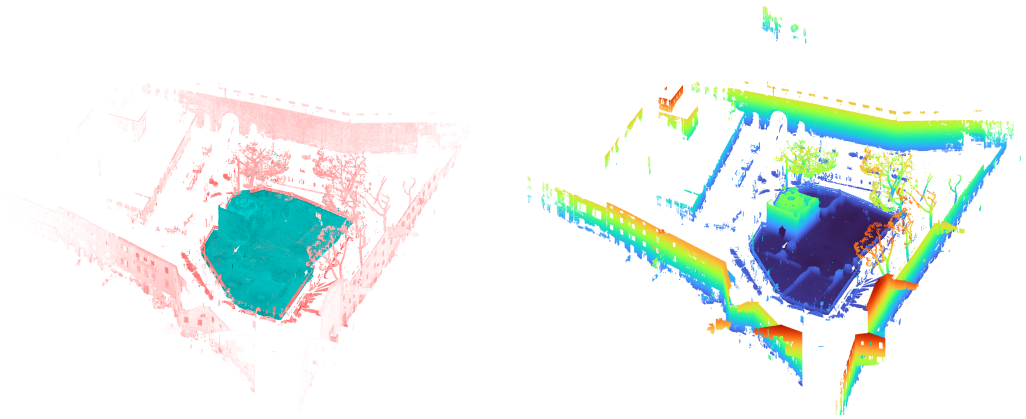
TABLE 4.1: Number of points and percentage "discard" labels for each point cloud as well as reprojection accuracy. The central columns representing the number of points and the percentage labelled for discard each contain two measurements seperated by a "/". These are the measurements on the subsampled and the original point clouds.

Scene	Subsampled / Original		Reprojection Accuracy	Area m^2
	Num. Points	Discard (%)		
Bagni di Nerone	5.4 / 55.4M	2.4 / 6.8M (44.8% / 12.3%)	99.5%	19k
Church	1.0 / 21.9M	0.05 / 0.3M (4.6% / 1.4%)	100%	4.7k
Lunnahoja	2.2 / 30.7M	1.6 / 12.4M (70.5% / 40.6%)	96.9%	11k
Montelupo	3.1 / 115.4M	1.9 / 52.6M (62.1% / 45.6%)	99.6%	2.9k
Monument	2.8 / 49.4M	2.7 / 40.9M (97.4% / 82.7%)	99.9%	3.5k
Piazza della Signoria	16.2 / 55.1M	0.7 / 1.4M (4.2% / 2.5%)	99.8%	52k

TABLE 4.2: Descriptions of datasets used for experiments

Scene	Description	Discarded data and labelling inconsistencies
Bagni di Nerone	Ancient Roman bath site	Structures around the area of interest, vegetation, railings, cars, trees, and people, noisy ground labels and parts of railing, tree foliage and people
Church	Underground Church with stepped courtyard	Railings, gates, and interior scaffolding, gate frames and ground labels
Lunnahoja	Wood and stone cabin in the woods	Trees and surrounding buildings, points inside main building, ground labels around perimeter of the main building are noisy
Montelupo	Old church site with lots of clutter and foliage, includes a deep alcove with irregular geometry	Vegetation around the area of interest and unintuitive labelling of people and wall in one area
Monument	Statue in a courtyard	Everything but the monument
Piazza della Signoria	Busy Piazza in Florence	People and vehicles, scattered data due to window glass, buildings at the periphery of the scene

4.1 Bagni di Nerone



(a) Coloured by label (b) Coloured by Z-height

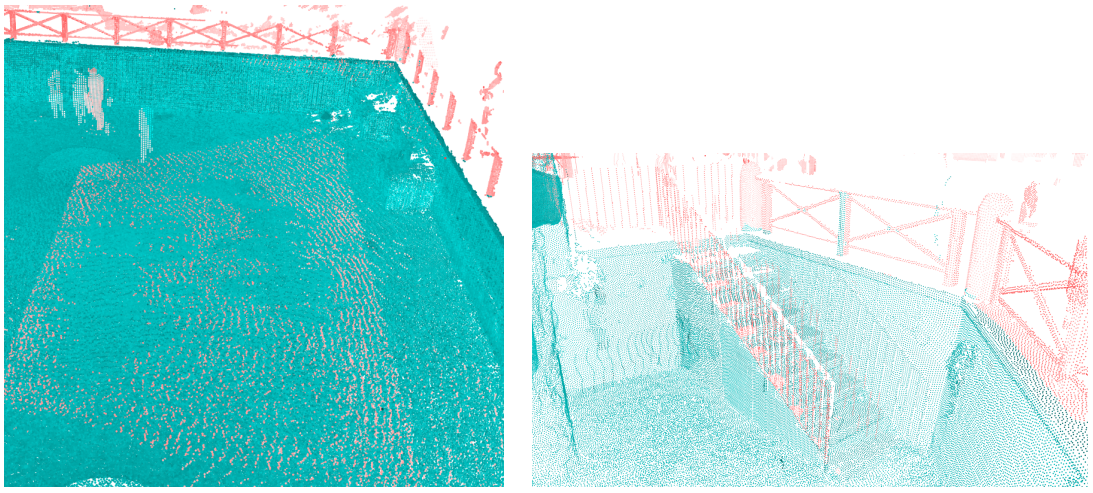
FIGURE 4.3: Bagni di Nerone: Isometric view. Points coloured by label and by Z-Height

Figure 4.3 visualises Bagni di Nerone (The Baths of Nero), the only Roman remains still standing in Pisa which now sits below street level. The octagonal hot room (Figure 4.4) is the best preserved structure on the site. The ruins have been partially restored at several points since their rediscovery in the 16^{th} and 17^{th} centuries [34].

The points to be removed from the site are some large trees along the edge of the railing around the ruins, followed by large 2D walls (i.e.they are only scanned from the side facing the main site) of multistorey buildings. Much of the geometry is missing due to scanner occlusions.

FIGURE 4.4: Bagni di Nerone: Comparison of photograph and scan taken at the same position (image via [44])

As with most of the scenes that we experimented on, this scene contains some unintuitive and arguably incorrectly labelled points. This is sometimes a modelling decision where points that may otherwise have been useful are discarded so as to not create problems later in the modelling pipeline (e.g. low-density areas). However, in Figure 4.5 we show two examples of incorrectly labelled regions that cannot be explained by a modelling



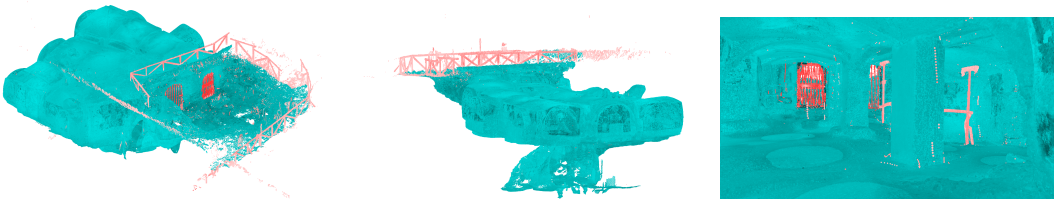
(a) Ground has noisy block of discard points from one scan, and one person is half labelled keep

(b) Inconsitent labelling of fence posts

FIGURE 4.5: Bagni di Nerone: Labelling inconsistencies. Red points are labelled discard, blue are labelled as keep.

decision. These regions are small relative to the rest of the scene however and are unlikely to have significantly impacted the model training.

4.2 Church



(a) Isometric Couryard

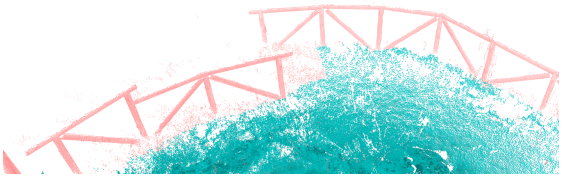
(b) "Outside" side view

(c) Interior showing scaffolding

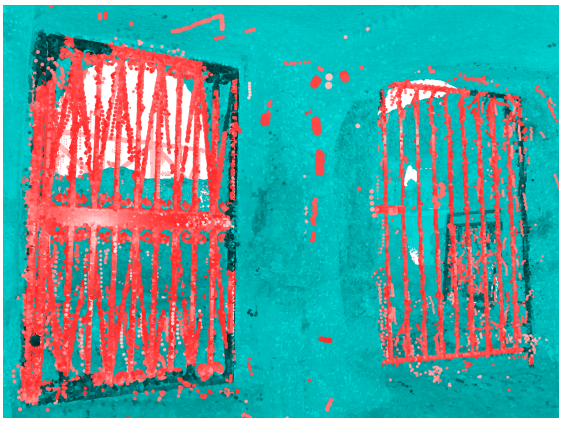
FIGURE 4.6: Overview of the Church scene coloured by label. Red points are labelled discard, blue are labelled as keep.

Figure 4.6 visualises an underground church with an exterior stepped courtyard (location unknown). The interior of the church is partially supported by scaffolding near an open stairway to the lower levels which are poorly mapped.

This is our main experimental scene on which we tuned all the models. It was chosen as it was the first scene we received the registration data for, and was also one of the more complicated scenes to label. The rationale is that methods which worked to train on this scene would be effective for training on the remaining scenes. Furthermore, this ensures that the remaining scenes better represent a "real world" test set where the models are not tuned further before testing.



(a) Corner of courtyard unexpectedly labelled discard



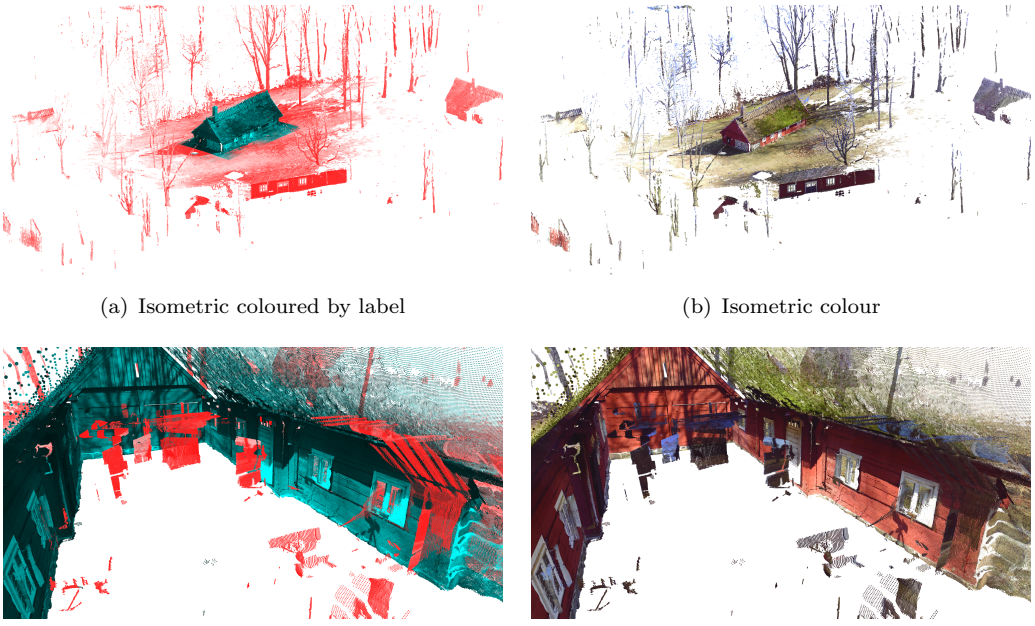
(b) Perimeter of gates are inconsistently labelled keep

FIGURE 4.7: Church labelling inconsistencies. Red points are labelled discard, blue are labelled as keep.

In Figure 4.7 we show several labelling inconsistencies that are significant enough to potentially impact the training process for the models. Specifically, in the edge around the railing of the outside area, it appears a "modelling decision" was made that is inconsistent with the surrounding labels. The two gates made up of thin bars are also poorly labelled and likely caused issues in the training process. The edges of the gates where they meet up with the walls are often misclassified as keep, it is unlikely this was an intentional decision by the original labellers.

4.3 Lunnahoja

Figure 4.8 visualises a wood and stone cabin in the woods scanned in fairly high detail with many partially scanned trees and additional buildings appearing at the periphery of the scene. The inside of the building also contains several free-floating regions to be removed due to the scanners interacting with the glass windows. This is a trivially simple scan to classify for a human as the required area is the cabin and a $\pm 1m^2$ perimeter around it. This is also the only scene for which we received colour and normal data, although these were not used it made understanding the scene far easier. However, for the purposes of the experiment, it is still a relatively tough scene as the ground truth



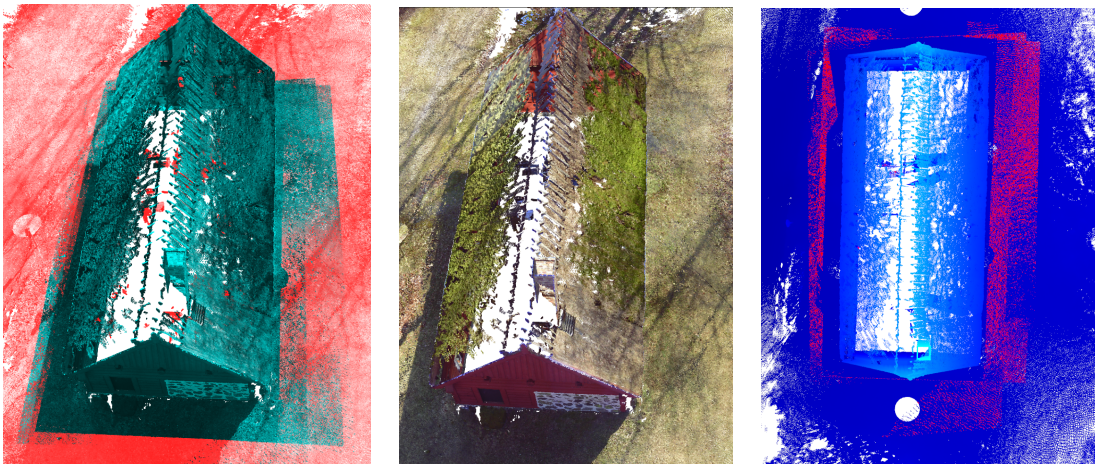
(a) Isometric coloured by label

(b) Isometric colour

(c) Inside coloured by label

(d) Inside colour

FIGURE 4.8: Lunnahoja overview, red points are labelled discard, blue are labelled as keep.



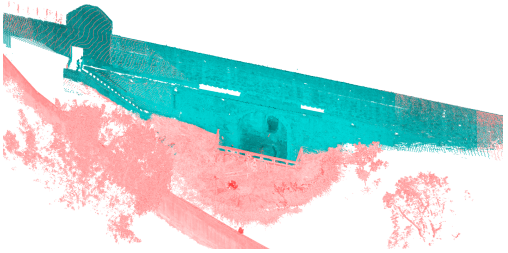
(a) Label

(b) Colour

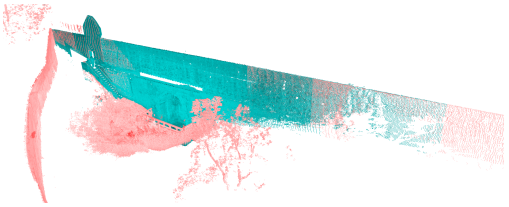
(c) Reprojection accuracy, red is incorrect

FIGURE 4.9: The ground around the cabin is noisily labelled (a,b) which leads to poor reprojection accuracy (c). Red points are labelled discard, blue are labelled as keep.

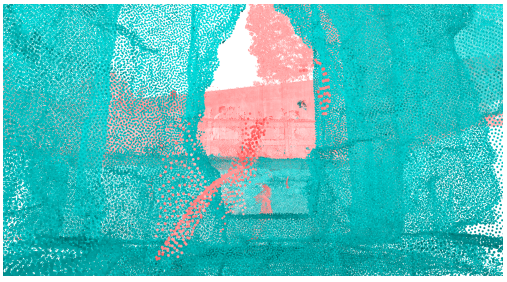
labels are very noisy. Figure 4.9 shows these incorrect labellings and how it directly cause the high reprojection accuracy seen in Table 4.1.



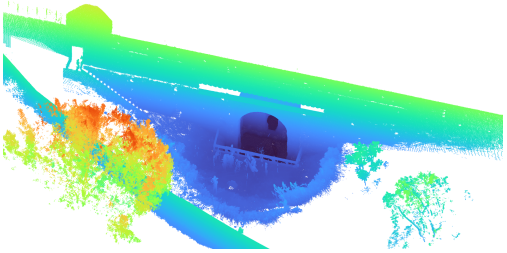
(a) Isometric coloured by label



(b) Montelupo labelling inconsistencies: banding along the wall



(c) View from within the alcove looking out showing the height below ground, coloured by label



(d) Isometric coloured by Z-Height, shapes in the middle are people and clutter around the railings

FIGURE 4.10: Montelupo Overview. (a-c): Red points are labelled discard, blue are labelled as keep.

4.4 Montelupo

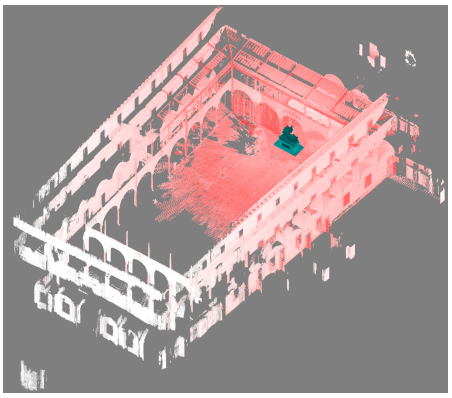
Figure 4.10 visualises the outside of an old church site with lots of clutter and foliage. The building is only scanned from the outside (one wall) and includes a below ground stone alcove with irregular geometry.

The upsampling errors are small and occur only at the boundary between the church wall and the ground.

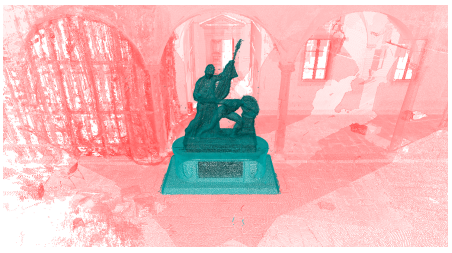
The points to be discarded are everything except the building. The ground truth irregularities are shown in Figure 4.10(b) appear on the far left and right wall of the building where it appears a single scan was incorrectly labelled discard.

4.5 Monument

Figure 4.11 visualises a statue, approximately 2×4m, in a courtyard with a large amount of open space. The points to be discarded are everything except the statue. Similar to Lunnahoja this is a trivial scene to clean for a human but continues to present challenges for training a model. Unfortunately, upon examining the results (Section 7.7) we find that the training splits (Figure E.4) were not created in a manner that allowed for meaningful model evaluation to take place.



(a) Isometric coloured by label



(b) Statue colured by label

FIGURE 4.11: Monument Overview, red points are labelled discard, blue are labelled as keep.

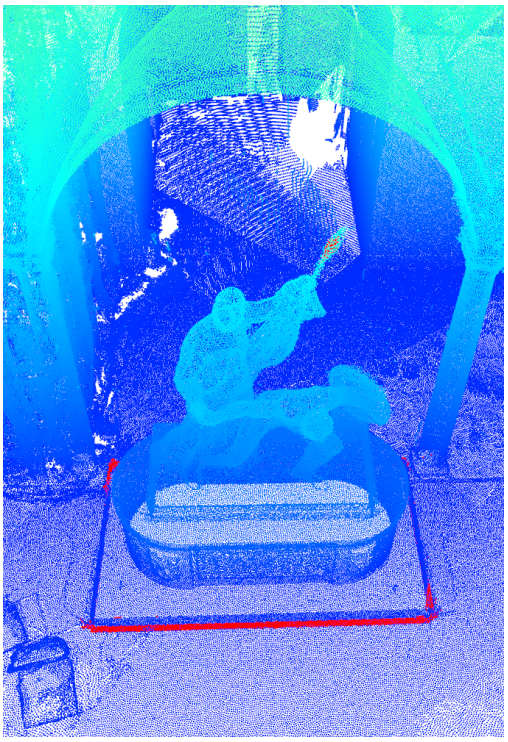
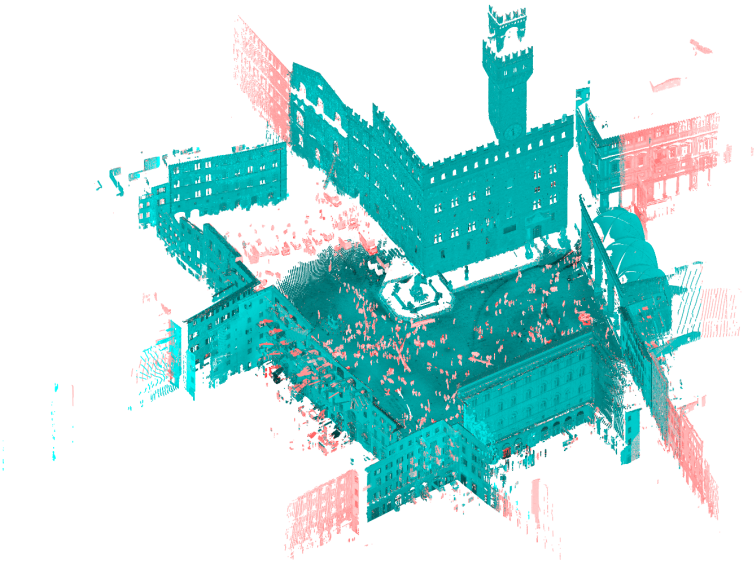


FIGURE 4.12: Monument upsampling reprojection errors, red points are incorrect.

Unlike the other scenes, the ground truth labels contain very few abnormalities. As for upsampling inaccuracies (Figure 4.12), these occur around the base of the statues where there is a boundary between keep and discard labels. Due to the focus on the small statue, there is an unusually high density of points in this area.



(a) Isometric coloured by label

(b) Horse statue, coloured by label

(c) Horse statue image [19]

(d) View of the plaza, coloured by label

(e) View of the plaza via [Inside Inferno]

FIGURE 4.13: Piazza Overview, red points are labelled discard, blue are labelled as keep.

4.6 Piazza

Figure 4.13 visualises the Piazza, a busy public square and the largest scene in our dataset containing several statues and many people and cars. The points to be removed are the people, who typically do not stand still and leave streaks in the scans (Figure 4.14), and the cars and buildings that extend beyond the edges of the main square.

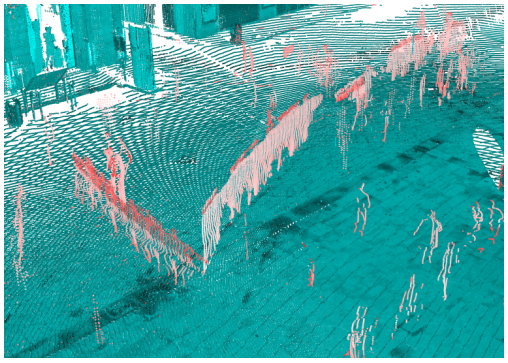


FIGURE 4.14: Scans of moving people result in smears, coloured by label, red points are labelled discard, blue are labelled as keep.

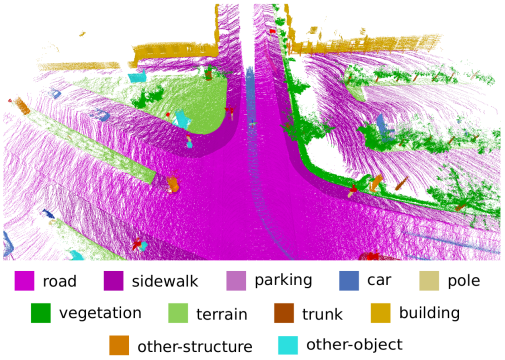
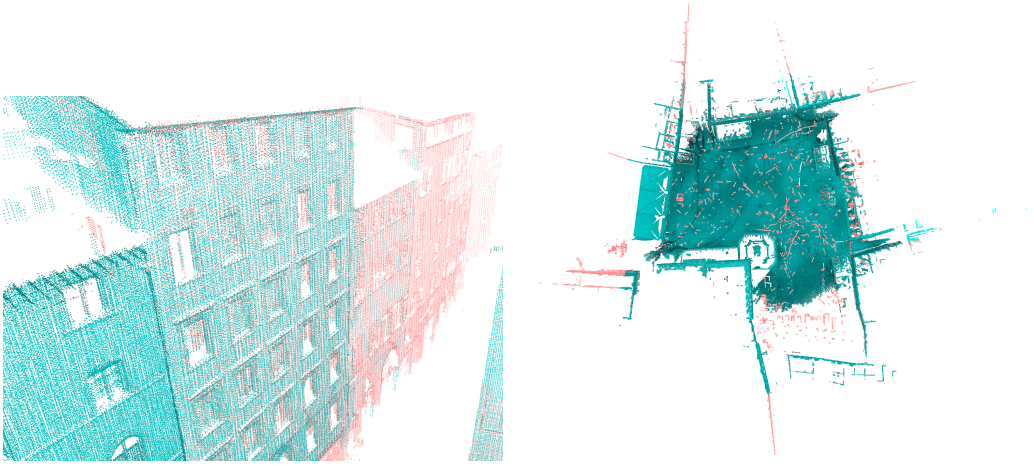


FIGURE 4.15: An example portion of the Semantic KiTTi dataset

In some areas the regions to be kept still extend far into the edges of the scan and seem as though they should have been discarded. Many of the interiors of the rooms in the surrounding buildings are partially scanned through the windows, however, the ground truth labels label these without any discernible consistency and will likely impact negatively on the model's performance metrics. Figure 4.16 shows where the ground truth labels are noisy and inconsistent with the surrounding labelling on the buildings far from the main piazza. This can also be seen in the top-down shot.



(a) Inconcistent building labelling

(b) Topdown, points beyond the main square inconsistently labelled

FIGURE 4.16: Piazza inconsistencies, coloured by label, red points are labelled discard, blue are labelled as keep.

4.7 Stanford 3D Indoor Scene

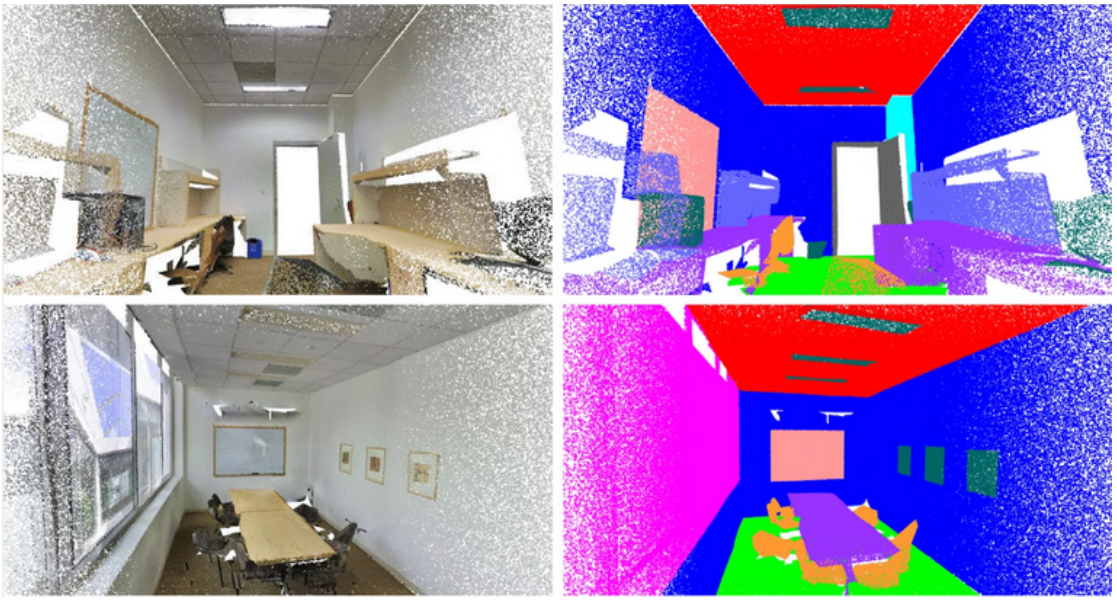


FIGURE 4.17: An example of rooms from the S3DIS dataset and their accompanying segmentation (source unknown via web)

The Stanford 3D Indoor Scene (S3DIS) is a large-scale indoor dataset split into six regions with 271 rooms and 273 million points across $6000m^2$. Each point is annotated with one of the 13 semantic categories. The data is collected using Matterport Cameras[5] which produce RGB-D depth images.

[5] https://matterport.com/

The dataset is relatively uniform between the areas as the dataset is scanned from a typical office and teaching environment. As such, the overall scene structure does not vary significantly. Walls, ceilings, and doors are all uniformly similar in shape, size, and surface texture. This is quite different from the heritage environments in which surfaces are far more irregular.

This uniformity and the different scanning method (compared to our dataset) may reduce the effectiveness of transfer learning. Nevertheless, it produces a point cloud that is closer in structure and density to our experimental dataset than other benchmarks.

Both SemanticKITTI[6] and Paris Lille[94] were considered as alternative pretraining datasets and represent car-mounted LiDAR scans. These create a specific circular ring pattern (Figure 4.15) in the point cloud that may be "learned" by the model. Additionally, S3DIS is an indoor scene, and the inclusion of roofs may help in the transfer learning for datasets containing interior rooms. Furthermore, each of the models we experiment with had been trained on the S3DIS datasets allowing us to compare baseline model performance before beginning experiments. Finally, following the completion of this paper, the authors were made aware of the existence of the ArCH dataset[68]. This is specifically a collection of cultural heritage buildings created via terestiral laser scanning, UAVs, and photogrammetry methods. Future methods may benefit from including this as a pretraining dataset rather than the S3DIS dataset. The S3DIS still has the benefit of having been used in the evaluation of each of our chosen deep learning models, however, the ArCH dataset does not to the best of our knowledge have published results on either KPConv, or Point Transformer.

4.8 Summary

This chapter gave an overview of the different datasets used during our experiments. In the following chapter, we will further describe the implementation details of the models we tested.

Chapter 5

Prototype Implementation and Exploratory Experiments

In this chapter, we provide the remaining details relating to our implementation of the models described in Chapter 3 for use in our experiments.

The following sections include the details of our dataset tiling, a discussion on point features and data augmentation, early stopping procedures and memory constraints to be aware of when training point cloud processing models. We end with a discussion of our exploratory experiments.

5.1 Point Features

Whilst each of the scenes contains at minimum the XYZ and label for each point, several of the scenes contain intensity values and Lunnahoja includes RGB and normal data as well.

Intensity is a measure of the reflectance of a surface based on the strength of the laser return. We initially planned to include intensity as visually it appeared to be useful in segmenting objects based on their surfaces. However, during the testing of PointNet++ (see Section 5.5.1) we decided to focus our experiments on purely geometric features as including intensity did not reliably improve the performance of the models. Figure 5.1 illustrates the difference made by adding intensity as a feature to the PointNet++ model. Similar to our Active Learning experiment (see Section 6.4.3), for experiments that include intensity and involve pretraining on S3DIS, the RGB channels averaged to approximate intensity.

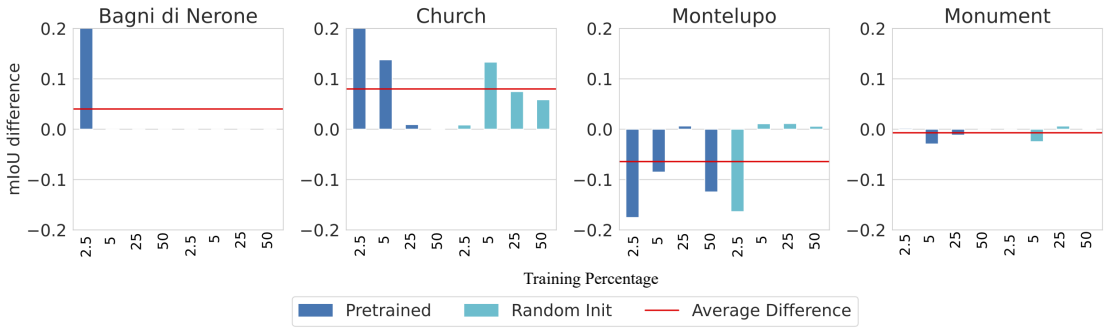


FIGURE 5.1: Difference in mIoU when including intensity for four of our scenes

Although RGB data may seem useful to include it was only available for one scene. As heritage scenes may take several hours or even days to capture a site, colours will change due to changing weather and time of day. This makes it unreliable as a segmentation feature [67].

Previous studies by Mulder [71], Pocock [82], and Marais et al. [67], made use of single scans and "human interpretable features". By this, we mean features that can be computed and relate to human understandable concepts in the real world. These are primarily derived from the eigenvalues and the local neighbourhood around each point and are easily calculated in software such as CloudCompare[1]. A full list of the available features can be found in Appendix C. We choose not to include these and focus on learning features directly from the raw point positions as is done in KPConv [112].

5.2 Dataset tiling

The original sample selection strategy used by PointNet++ assumes the data is pre-split into small regions with relatively few points, mimicking the rooms in S3DIS. Directly applying PointNet++'s sampling strategy on our scenes led to long training times, ±7.5 minutes per epoch on the 50% Church split. Each time a sample was needed, every point's location relative to a "seed" point would need to be computed. This was prohibitively expensive even on smaller scenes where each epoch consisted of only a few hundred samples, larger scenes containing thousands of samples would compound this issue.

We first worked on an efficient method of splitting the scene up into workable regions. In PointPillars[55], the scene is split into pillars in the XY plane. This allows the point cloud to be "flattened" into a high dimensional representation of each "cell" which can then be processed using any standard 2D CNN for object detection.

[1] Available at https://www.cloudcompare.org/

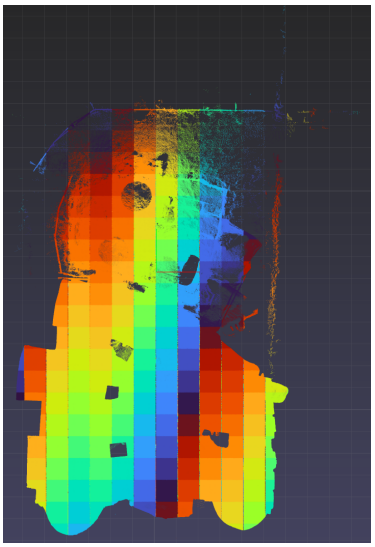


FIGURE 5.2: Grid overlay of Church scene viewed top down after splitting the scene into $1m^2$ columns. False coloured to better separate each of the column grid cells.

Adapting this approach to our needs, we split the scene into columnar segments. We use a $1m^2$ column size as we found it to be a good tradeoff between granularity and preserving geometric details. This creates a checkboard-style pattern as illustrated in Figure 5.2 by colouring each cell. For efficiency, we precompute the splits when creating the training sets such that there is no overlap with the test data (see Chapter 4). For the same 50% Church set, our method reduces the training epoch time to two minutes and precomputes the columns in 75 seconds.

Our implementation of PointNet++ makes use of this for its sample selection to ensure that regions are sampled evenly during both training and evaluation. Both KPConv and Point Transformer use a spherical sampling strategy. A random point in the scene is selected around which a spherical lookup can be done efficiently for arbitrary radii and numbers of points (by randomly subsampling the resulting sphere of points).

Whilst Point Transformer computes neighbourhoods dynamically at runtime, KPConv makes use of precomputed neighbourhoods and KD-Trees. First, the model precomputes the KD-Trees for the point cloud. Query points are selected and a spherical radius is queried around those points such that every point in the scene is covered by at least k spheres. The size of the spheres is typically set based on the size of the objects in the scene, with larger objects allowing larger radii to be used at the cost of lower detail (for the same number of points per sphere).

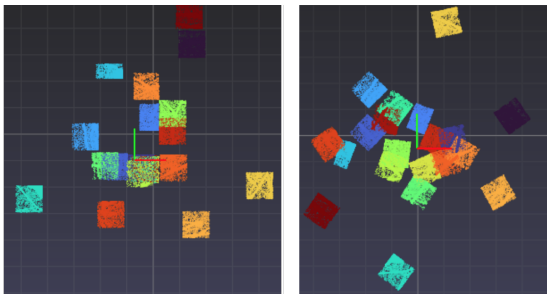


FIGURE 5.3: A batch of samples (false colour) before and after rotating in the Z-axis

5.3 GPU Memory Constraints

During training a common issue relating to VRAM (the GPU memory) usage was encountered which has not been adequately addressed to the best of the author's knowledge.

The models regularly need to find the distance between every pair of points in an input sample for use in radius or kNN lookups. For N points this results in N^2 distances being calculated and stored in VRAM. Although this matrix is only created for a short moment it requires a significant amount of VRAM (potentially several gigabytes) to be available to prevent crashing the training run. Due to the variable number of points in a batch this may only occur later in the training depending on the dataset.

This N^2 approach allows a custom function to be more easily written in C++ or CUDA using vector maths and reduces the number of computations that must be performed. Whilst this improves the forward pass speed it greatly reduces the number of points per sample, and samples per batch, that can be processed in each forward pass.

5.4 Data Augmentation

Data augmentation techniques can help improve the generalisation capabilities of the models and make them more robust to noise. This is done by augmenting the samples in a way that the resulting samples could exist in the original dataset [35].

On point clouds, this can be achieved through random perturbation of the points (adding some noise or jitter), shifting colour, sample scaling, rotation and translation.

Figure 5.3 shows how each sample in a batch can be independently rotated about the Z-axis. This changes the orientation of features in a way that is plausible to exist in the scene. We do not rotate about the X- or Y-axis.

The default augmentations used are Z-rotation, scaling, and random perturbation of points. Removing these augmentations neither meaningfully increased nor decreased performance for PointNet++. As such, we keep these augmentations active for both KPConv and PointTransformer.

5.5 Exploratory Experiments

In this section, we describe any additional implementation details for each of the deep learning models and describe the initial exploratory experiments where applicable. All hyperparameters for each of the models can be found in Appendix D.

During the development and exploration process, we only use the Church scene as described in Chapter 4. As part of the initial pretraining experiments with each model, we attempted to finetune the model by freezing the body and only training the classification head. For all three models this approach did not yield promising results and we continued the remaining experiments with the full network being trained.

5.5.1 PointNet++

PointNet++ was the main model we explored during development. During the initial experiments, the training loss was erratic and it was difficult to prevent the model from learning to only predict the majority class due to the 99 : 1 class imbalance in the Church scene.

We began by training a randomly initialised PointNet++ on a randomly selected 75% training split (Figure 5.4) of the Church scene as a proof of concept. After training for 150 epochs, the best test mIoU was achieved at $epoch22$ and took 2 hours to train at $\pm 5 min/epoch$. Initial results were promising based on a training mIoU of 79.7% and accuracy of 98.5%, however when the data is this imbalanced, an accuracy of less than 99% is arguably a failure from a certain point of view. However, the model was unable to generalise to test data with an mIoU of only 28.9%. Both the loss and mIoU on the test set have large oscillations at the start of training before flattening and ceasing to learn (Figure 5.5).

Figure 5.6 illustrates how the model fits well to the training data but is unable to generalise to the test data. The majority of the test set is classified as “keep”, the majority class, with only some of the isolated noise and the ends of the scaffolding pipes being labelled as discard points.

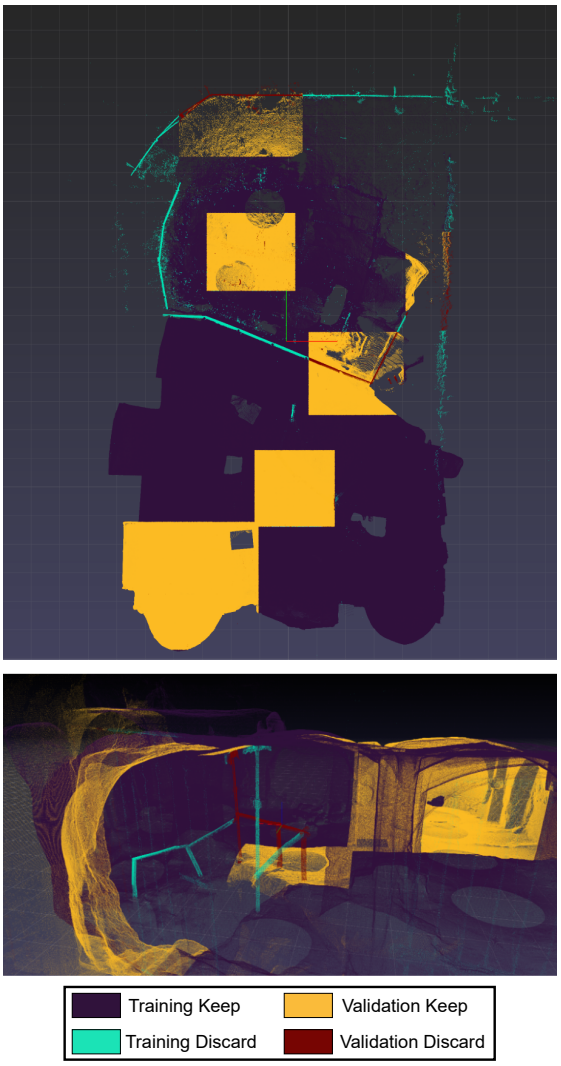


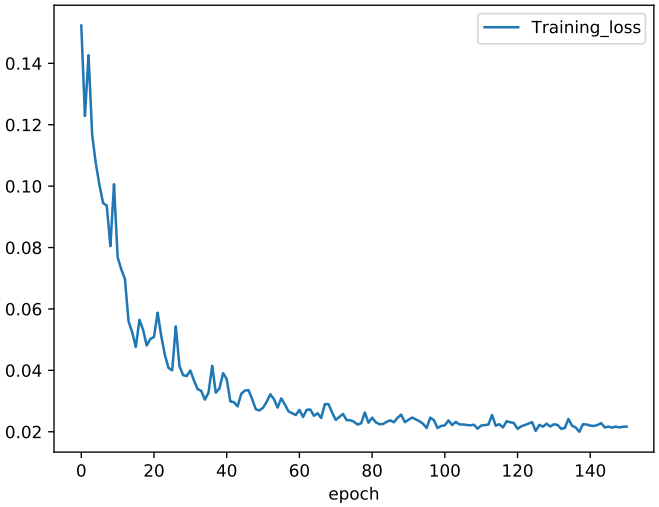
FIGURE 5.4: Top down and interior view of the data split for Church scene with 75% : 25% training : validation

In order to improve these results we began tuning the hyperparameters using the 25% training set of the Church scene as described in Chapter 4. We made use of a grid search over the values shown in Table 5.1.

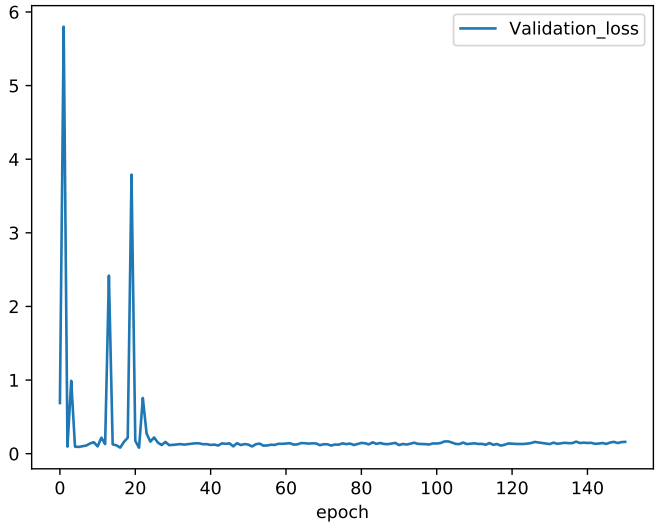
TABLE 5.1: PointNet++ Grid Search Space, final values in **bold**

Parameter	Possible Values	Description
learning_rate	0.0001, **0.001**, 0.01	Learning rate
weight_decay	0.0001, 0.001, **0.01**	Weight decay
num_points	2048, **4096**, 8192	Number of points per sample
use_intensity	true, **false**	Whether to use intensity
use_local_xyz	true, **false**	Whether to use local coordinates
augment_points	true, **false**	Whether to augment points

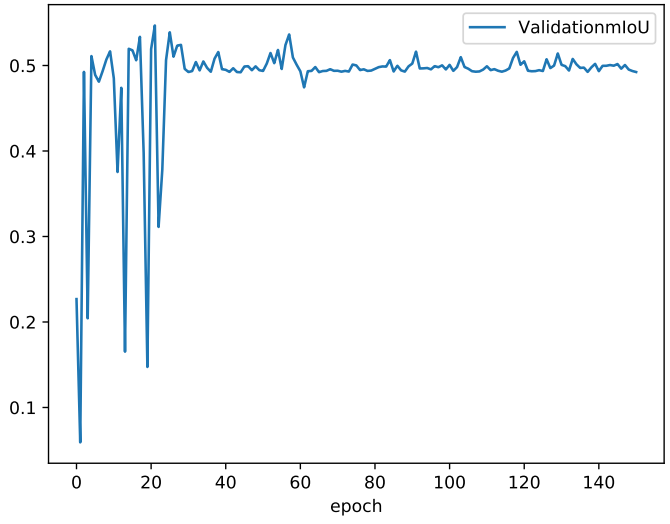
We also explored pretraining the network on S3DIS and from here onwards we constrain our experiments to use only global XYZ coordinates without additional features. We



(a) Training Loss



(b) Test Loss



(c) Test mIoU

FIGURE 5.5: Training and Test graphs for early PointNet++ prototyping on the Church scene (note the difference in y-axis scale)

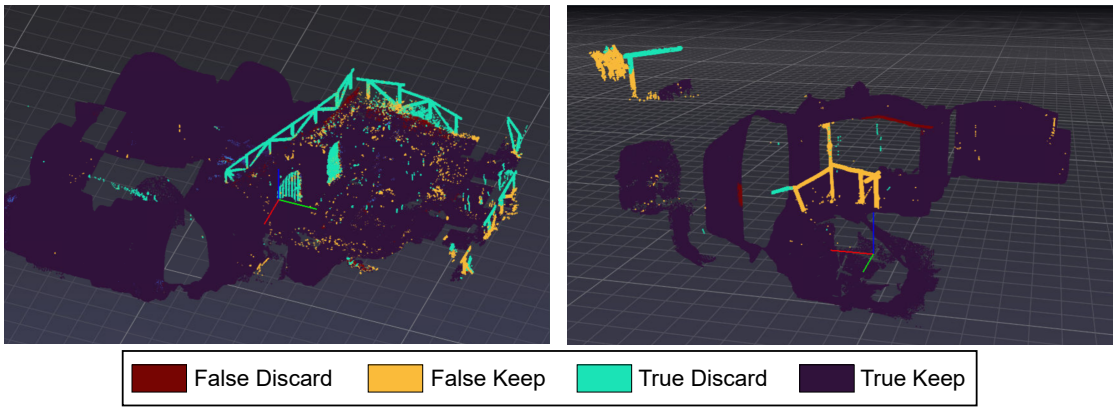


FIGURE 5.6: Early results from PointNet++ trained on 75% of the Church scene. Left: Training, Right: Validation

tested including a set of XYZ coordinates relative to the centre of the sample. For reasons that are unclear, adding relative coordinates reduced the performance of the models on test data.

The pretrained model generally appears to improve faster in the early stages of training, as expected. We expected that the pretrained (and then finetuned) model would outperform the model trained from scratch when the percentage of the cultural heritage scene used for training/finetuning data was smaller. The "improvement via pretraining" was expected to reduce as more cultural heritage data was trained on. We show in Section 7.4, however, that the difference between the pretrained and randomly initialised models becomes more, not less, as more of the cultural heritage site is finetuned/used for training. This is contrary to our expectations that the gap between the pretrained and randomly intialised model would narrow (i.e. the benefit of pretraining would decrease) as more training data was provided from our scene.

The largest improvement came from increasing the weight decay parameter in the Adam optimizer [50] from 0.0001 to 0.01, at the cost of slower convergence. Strictly speaking the implementation of Adam in Pytorch is applying weight decay to the gradients as L2-normalisation, rather than the weights themselves. AdamW [64] is proposed as a simple fix, however, this was only pointed out to the authors after the final experiments were completed. Due to time constraints, it was not possible to repeat the experiments with the corrected AdamW optimiser.

After pretraining and tuning the hyperparameters, we reevaluate the model on the original 75% split. The best result achieved by the model on the test data was 99.61% accuracy, an F1 score of 85.35% and an mIoU of 81.16%. In Figure 5.7 a comparison between the models' training and test accuracy is shown. This illustrates that with sufficient data PointNet++ can accurately label the remaining points in a scene. At a more granular level, 80% of the points to be discarded are correctly labelled compared to

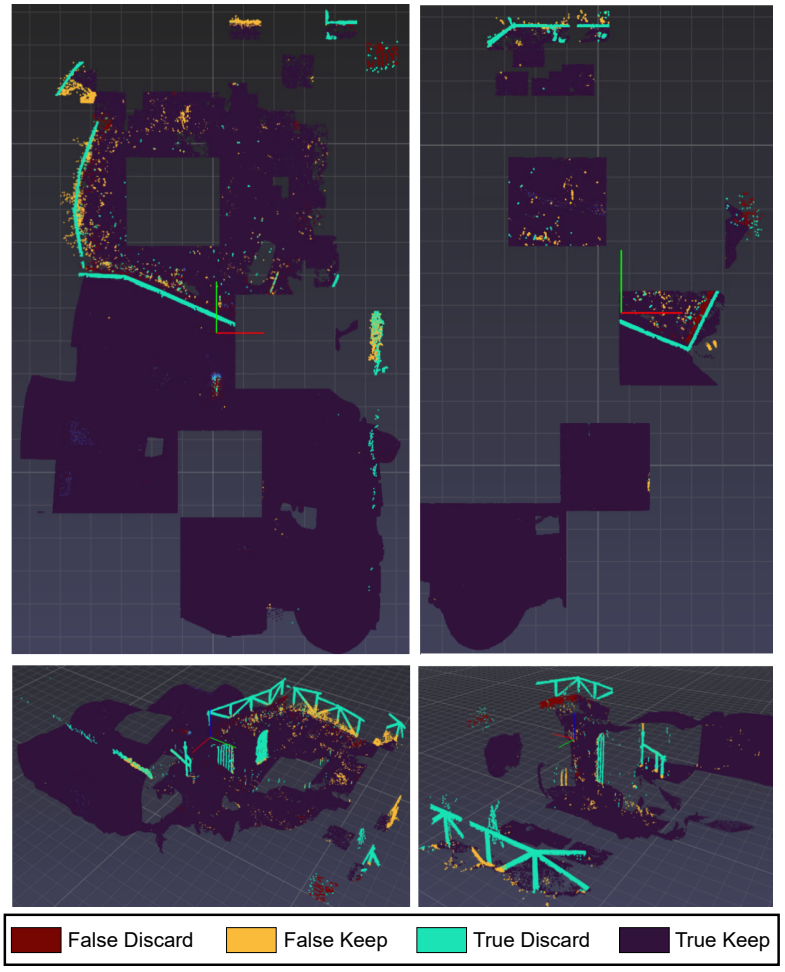


FIGURE 5.7: Church 75% training results.
Training on the left, Test on the right

5% in the initial proof of concept experiment. Similarly, only 0.1% of points that should have been kept were incorrectly labelled as discard. This validated that PointNet++ had the capacity to learn the scene and produce a useful segmentation.

5.5.2 KPConv

Although complex, the codebase provided by the author[2] was easily adaptable to our dataset and experiments.

KPConv has two sampling strategies. Either samples are drawn to ensure the entire scene is covered equally by tracking how many times each point has been sampled (described in the paper as "potentials"), alternatively, samples can be drawn in a manner that aims to keep examples of each class balanced. Based on suggestions by the author[3], we set $use\_potentials = False$ to help mitigate the effect of the imbalanced scenes. We also

[2] Available on Github
[3] See discussions on Github

follow the advice to use the non-deformable network if possible as it is slightly faster to train. Our initial experiments below suggest that the increased complexity of the deformable network is not necessary. We use a small initial radius to better pick up fine detail structures such as the gates and scaffolding in the Church scene.

Initial experiments with KPConv suggested significant improvements over PointNet++ could be achieved far beyond the performance improvement on the S3DIS benchmark. Whilst the performance plateaued at ±80% mIoU, it was able to achieve this using as little as 2.5% training data, with spikes as high as 98.1%mIoU when starting with 25% training data. Due to the strong performance in our initial tests, we keep the default hyperparameters.

5.5.3 PointTransformer

Whilst the authors do not make their original implementation available, POSTECH CVLab from Pohang University provide a reimplementation with the help of one of the authors.[4]

Unlike the previous models, Point Transformer requires more memory during training and can be significantly slower than KPConv, but can handle a much larger receptive field. The increased memory requirement means this cannot be run on an 8GB GPU such as the GTX 1070 on which previous development was conducted. At least 12GB of VRAM is required to train this model effectively on our dataset.

We performed thorough hyperparameter tuning on this model. Starting with the default parameters, we use the Weights and Biases Sweeps[5] framework to perform bayesian hyperparameter optimisation. This creates a probabilistic model, using a Gaussian process, that chooses new hyperparameter configurations with a high probability of improving the optimisation metric, the test loss. This allowed us to specify a plausible range for each of the hyperparameters and the sweep would automatically focus on adjusting only the hyperparameters that most improved the model (Table 5.2).

It is important to note the issues around leaking information to the model given that we are tuning on the *test* loss. We wish to reiterate the strategy of using *only* the Church dataset for this process. Throughout this thesis the Church scene has been treated as the primary experimentation scene and we fully acknowledge that tuning the models on this scene will skew the results in favour of this scene.

[4] Available on Github
[5] Available at Weights and Biases Sweeps

TABLE 5.2: Point Transformer Sweep Configuration

Parameter	Possible Values	Description
base_lr	0.3, 0.5, 0.7	Learning rate
freeze_body	true, false	Whether to freeze body or retrain the full network
power	0.3, 0.6, 0.9	Controls the rate of decrease of learning rate when using the warmup scheduler
scheduler	warmup, default	Scheduler type
voxel_max	5000, 10000, 20000, 30000, 40000	Max points per sample
warmup_length	10, 25, 50	Warmup length
weight_decay	0.01, 0.001, 0.0001	Weight decay

We found the performance to be heavily impacted by the chosen hyperparameters, unlike the previous two models which were relatively stable. Additionally depending on the training set size different hyperparameters were optimal. See Appendix D for details of what we found to be the optimal hyperparameters at each training split.

5.6 Summary

In this chapter, we discussed the changes we made for integrating the deep learning models into our testing framework and any notable issues we encountered in the process. We presented the details of our exploratory experiments and position KPConv as the expected best tradeoff between performance and compute resources. In the following chapter, we describe our final experimental design.

Chapter 6

Active Learning

A Preliminary Feasibility Study

Active Learning is an incremental training process that aims to maximize learning potential by having the *model* select training samples to query from an oracle[1], as opposed to random sampling. This recognises that all samples are not equally useful at all stages of the training process; some samples may be redundant, or irrelevant. By querying an oracle for labels to the most informative samples selected by the model, active learning can reduce the amount of training data needed to achieve a given level of performance [100].

Whilst Marais et al. [67] also study a human-in-the-loop approach which appears similar to this, an important distinction is the choice of what should be sampled next. In Active Learning the model itself, rather than the human annotator (the oracle) selects the next sample.

For a highly simplified albeit intuitive introduction to the ideas behind active learning, consider the image classification problem of cat vs dog. In traditional supervised

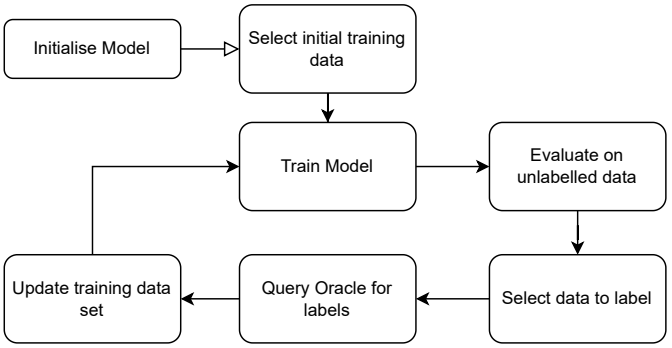


FIGURE 6.1: Active Learning Loop

[1] An oracle is anything with assumed perfect knowledge such as a human expert or a pre-labelled dataset

learning, the training set would contain images drawn randomly from the available data. However, we can imagine a scenario in which the model becomes better at labelling cat images than dog images. At this point, training on more cat images is not going to help the model to improve as much as if it were to train on dog images (where it is performing poorly). Active learning seeks to automatically detect this change.

In the following sections, we discuss various Active Learning strategies and techniques in more accurate detail, including the importance of calibration, and our feasibility study.

6.1 Query Strategies

The *active* portion of Active Learning relies on the *model* choosing samples to query. Several methods can be used when selecting new training samples. Common methods include uncertainty sampling [59], query-by-committee [103], expected error reduction [93], and density weighted method [101]. In this thesis, we make use of a form of uncertainty sampling inspired by REDAL (see Section 6.4.1). Many of these methods make use of Monte Carlo dropout [32] as a method of Bayesian approximation of model uncertainty.

Uncertainty Sampling selects samples with the lowest confidence prediction of the model. A general method of measuring uncertainty is using *entropy [104]*. For more on why relying solely on the model's logits as a confidence measure is ineffective see Section 6.2. Intuitively, certain samples are more informative and reduce the model's error more (i.e. the model can "learn more"). However, noisy and outlying samples may be more likely to be selected or a single class may be more favoured.

Query-by-committee uses multiple competing models (traditionally trained on the same data but can also be trained on different subsets of the data, sometimes called query-by-bagging or query-by-boosting [2, 88]) to vote on the label of an unlabeled sample. The sample with the most disagreement among the members is selected for querying. However, this may be ineffective if the models are too similar or too dissimilar, and requires more computational resources.

Expected error reduction selects the sample that, when labelled and added to the training set, is expected to reduce the generalisation error the most. This expected reduction is the weighted average of all possible labels (as given by the posterior probabilities). While theoretically near-optimal, in practice it is not widely used as the model

must be retrained for each potential sample. Newer Bayesian sampling approaches have been proposed to improve efficiency [74].

Density weighted methods help to address the likelihood of selecting outliers when using uncertainty sampling or query-by-committee. By combining the informativeness (via uncertainty sampling or query-by-committee) and a similarity score with other samples in the dataset, samples that are representative of a larger portion of the dataset can be queried. Provided that density scores can be pre-computed the training times will be largely unaffected by the additional criteria [1].

Querying in Batches is a strategy for improving the efficiency of active learning by querying several samples from the oracle at once instead of one at a time. This approach is particularly useful when training time is long, such as with neural networks. To ensure the effectiveness of this method, the selected samples should be diverse to avoid redundancy while still providing informative data [102].

6.2 ANN Calibration

Modern NNs are known to exhibit poor calibration and tend towards overconfidence [38]. Calibration refers to the closeness of predicted probabilities compared to the observed probabilities. Put another way, models may be overly confident in their predictions when the answer is uncertain.

In a well-calibrated model, if we take all the predictions with an output probability of $X\%$, a well-calibrated model will have an accuracy of $X\%$, whereas a poorly-calibrated model will not. For example, when the prediction score is 0.8, the model should be correct approximately 80% of the time.

Guo et al. [38] performed a variety of experiments on different classification models to explore the issues around model calibration. Figure 6.2 highlights this concept for an image classifer on the CIFAR[2] dataset. The "correctly calibrated" histogram is marked in red and the "incorrectly calibrated" model in blue. The model is more likely to predict more extreme outputs and under-predict more moderate/uncertain outputs.

Overconfidence can harm decision-making processes that rely on the predicted probabilities. For example, in a medical classifier; if the true probability of a negative diagnosis is 70% and the model (overconfidently) outputs a 95% probability this might lead a doctor to make a decision they would not have made had they known the true probability.

[2] Available at https://www.cs.toronto.edu/~kriz/cifar.html

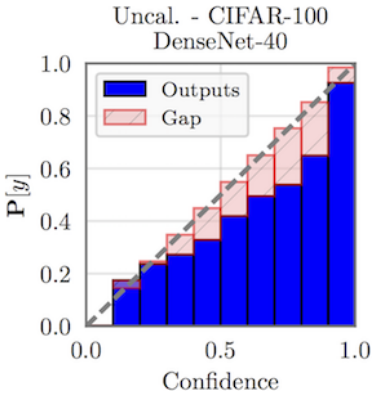


FIGURE 6.2: Poorly calibrated prediction probabilities (via Guo et al. [38])

Several factors may contribute to miscalibration including overfitting and choice of the loss function (cross-entropy loss encourages more extreme outputs).

In the context of active learning, if the goal is to query samples for which the model exhibits low confidence, a poorly calibrated model's logits cannot be directly interpreted as the "model confidence".

Two methods for calibrating ANN outputs are temperature scaling and ensemble-based querying. Temperature scaling is a simple post-processing method that adjusts the softmax outputs of the neural network to better reflect the true probabilities, provided there is a validation set [38]. Ensemble models have also been demonstrated to be better calibrated than using a single ANN [53, 58], however, they are more computationally expensive to train.

6.3 Related works

Recent papers on active learning for point cloud cleaning helped motivate our experiments on heritage scenes. Depending on the dataset, active learning on pointcloud segmentation can achieve 90% the performance of fully supervised training using from 50% to as low as 5% of the available data [48, 121].

Shi et al. [105] explored different sampling strategies and sample selection methods for active learning based point cloud segmentation tasks. Lin et al. [61] propose a tile based active learning strategy on point clouds using PointPointNet++; a near parallel to our experiments below. Their experiments differ from ours in the dataset used (large area, low density, urban environment). Their experiments show comparable performance to fully supervised training with only 40% of the dataset labelled. However, it is unclear if they account for the validation set in that labelling budget, an additional 19%.

6.4 Heritage Scene Experiments

In heritage point cloud settings there is a need for accurate, manually collected labels. Labelling these scenes by hand is time-consuming. We explore Active Learning as a strategy for reducing the amount of labelling which must be done in order to train a model on a particular scene. We use PointPointNet++[86] and a Random Forest as the base models (see Chapter 3) and adapt the technique described in REDAL[121] for query selection.

Active Learning did not meaningfully improve the performance of the models on the datasets it was applied to and other approaches were thus pursued (see Sections 3.4 and 3.5).

In the remainder of this chapter, we provide the implementation details and an analysis of the results, as well as avenues for future improvement on this approach.

6.4.1 ReDAL

ReDAL (Region-based and Diversity-aware Active Learning) is an active learning method proposed for point cloud semantic segmentation which we partially implemented to select our queries. With only 15% of the total training data, 90% of the performance of a fully supervised approach on the S3DIS benchmark can be achieved [121].

A high-level overview of the ReDAL algorithm is illustrated by Figure 6.3. First, each sample (created by over-segmenting the scene using the VCCS algorithm [80]) is scored based on its uncertainty using softmax entropy, colour discontinuity, and structural complexity to give a "region information score" (RIS). Samples are grouped into K clusters using feature embeddings and k-means, and sorted based on their RIS. To avoid redundancy, scores of subsequent samples in the same cluster are penalized by a factor of P. The result is a ranking of regions based on both uncertainty and diversity, allowing for an efficient selection of samples to annotate.

Our implementation (algorithm 1) follows ReDAL but does away with the colour discontinuity and structure complexity measures when calculating the RIS. In their ablation study, the authors found that the colour and structure measures were the least impactful. We rely on the sum of the variance of the predictions in each sample based on Monte Carlo Dropout (MC-Dropout) ensembling. This combines the concepts of uncertainty sampling and query by committee whilst improving the model's calibration. Additionally, we make use of the column samples (see Section 3.3) rather than the suggested over-segmenting approach.

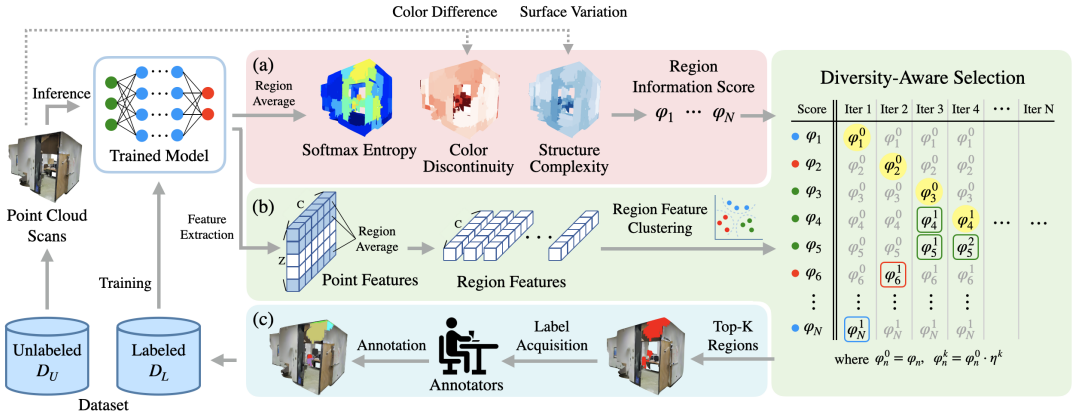


FIGURE 6.3: ReDAL algorithm: The red "Region Information Score" is replaced in our implementation with the prediction variance. (sourced from [121])

Algorithm 1: Active Learning Process

Input : model, unlabelled_set, mc_dropout_repeats, percentage_increase

Initialize: model, unlabelled_set

Label *initial_training_set*
while *stopping condition not met*:
    Train *model* for *training_epochs*
    for each *sample* **in** *unlabelled_set*:
        Calculate *average_point_features*
        Calculate *prediction_variance* using *mc_dropout_repeats*

    `// REDAL`
    Rank *unlabelled_set* by *prediction_variance*
    Apply k-means on *average_point_features*
    Reorder *sample_regions* based on clustering results
    Select and annotate top-N *sample_regions*
    Update *labelled_set*

An alternative approach by Rodríguez et al. [90] suggests ranking each sample based on Equation (6.1) consisting of a joint measure of uncertainty and diversity. The uncertainty is given by the individual variance of the prediction $\sigma^2(\hat{y}_i)$ relative to the total variance of all predictions within the set U. A higher relative variance indicates a greater uncertainty. The diversity component is given by the squared distance of a sample from the mean in the feature space z relative to the sum of squared distances for all samples in U.

$$g(x_i) = \underbrace{\frac{\sigma^2(\hat{y}_i)}{\sum_{j \in \mathcal{U}} \sigma^2(\hat{y}_j)}}_{\text{Uncertainty}} + \underbrace{\frac{d^z(x_i, \mu)^2}{\sum_{j \in \mathcal{U}} d^z(x_j, \mu)^2}}_{\text{Diversity}} \tag{6.1}$$

To prevent outliers from dominating the query set, samples are clustered using k-means and only those samples lying closest to each cluster's centre are selected.

6.4.2 Distance Metrics

As part of our experiments below, we explored the use of several additional distance metrics for clustering using the 128-dimension feature embedding from PointPointNet++. We additionally compared using the interpolated features and the final feature embeddings from the PointPointNet++ model and found the feature embeddings to be more expressive, as expected (Figure 6.4).

Although Euclidean distances are often the default choice, they break down in high-dimensional spaces, an instance of the "curse of dimensionality". Typically when the dimensionality of the data grows it is more effective to use the cosine distance, which measures the "similarity" of two vectors regardless of the magnitude[3].

Using both the Silhouette Score [92] and Elbow method [113] we explored Square Euclidean, cosine, Canberra, Braycurtis, and correlation metrics. We chose to use the standard Euclidean distance-based k-means (with $k = 10$) as the alternate distance metrics suggested an optimal cluster count of ± 30 which is higher than the number of sample regions in our scenes.

6.4.3 Experimental Results

We present three sets of experiments for which the best results were achieved, two on PointPointNet++ and one on the baseline Random Forest. The experiments were performed starting with only 5% of the Church dataset labelled and increasing in increments of 5% (as suggested in [61]) up to 25% of the full dataset.

The point features are $[X, Y, Z, Intensity]$ and the PointPointNet++ models are pre-trained on the S3DIS dataset with $mean(rgb)$ used to approximate an intensity feature.

The two PointPointNet++ experiments were performed using a 5-fold and 10-fold MC-Dropout with 5 and 12 epochs of training per iteration respectively. MC-dropout is used as an efficient approximation of training multiple models, giving an estimate of model confidence by calculating the variance of the predictions. The amount of MC-dropout required is scenario-dependent, but beyond 20 iterations is likely unnecessary [58].

To keep this applicable to a real-world scenario (in which only the training data is known, see Chapter 4), we begin each active learning iteration starting with the model

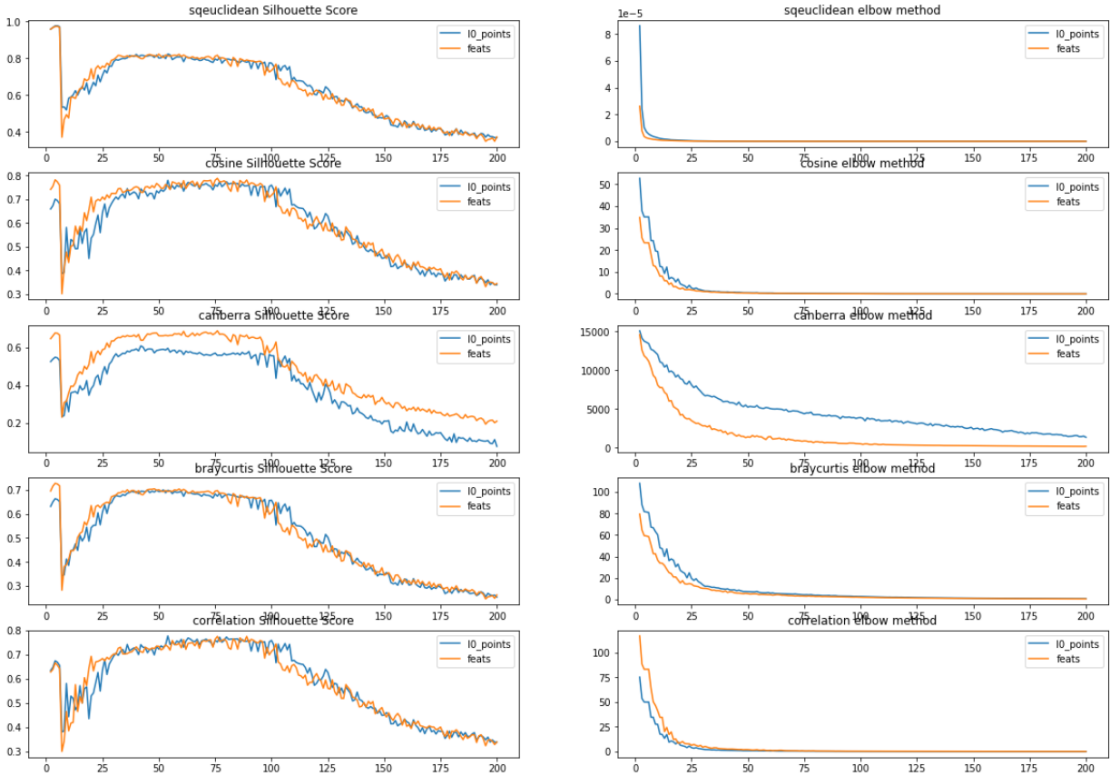
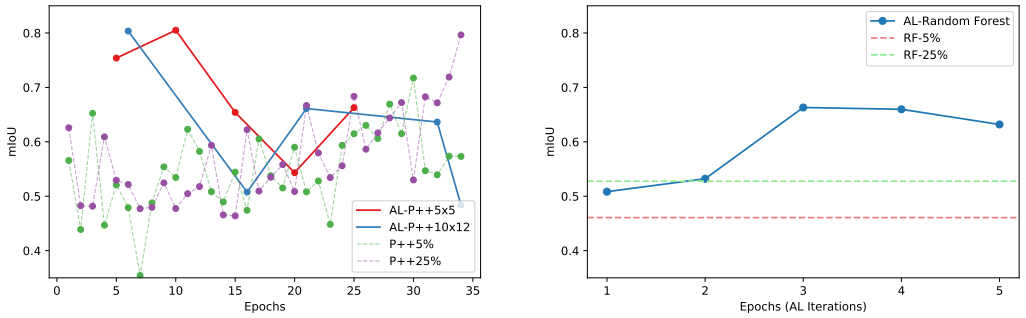


FIGURE 6.4: A comparison of clustering distance metrics for k-means using features at different layers of PointPointNet++. X-axis is increasing values of k



(a) PointPointNet++ using 5x / 10x MC-Dropout and 5 / 12 Training Epochs per active learning iteration

(b) Random Forest

FIGURE 6.5: Active Learning experiments compared to training directly on the 5% and 25% Church datasets. Measuring mIoU vs Training Epochs

state at the best training set performance of the previous iteration. The first iteration begins with a pretrained model. For this reason, we keep the training epochs relatively low to try and prevent overfitting. The Random forest is fully retrained on the updated training set at each iteration.

Figure 6.5 represents the validation mIoU of each of our three active learning experiments at the end of each iteration compared against the mIoU achieved for a given number of training epochs.

Our experiments do not support the use of Active Learning for the remainder of this thesis. At the time that Active Learning was considered and decided not to be included in further experiments only the Church dataset was available for use. Given then poor performance described below we chose not to continue exporing active learning methods during the rest of this thesis. Both PointPointNet++ experiments do not surpass the mIoU achieved by training directly on the 25% labelling in the same number of epochs. The training was erratic (a common theme on our data with PointPointNet++), leading to the performance decreasing rather than increasing at times. Only the RF model was effective, showing consistent improvements and surpassing the 25% training model at 10% of the data.

Reasons for the poor results could be several:

- The model may have been overfitting the training data.
- Models often exhibited reduced performance as the training dataset size increased. Since models were evaluated on a reducing unlabelled set, each iteration removes a portion of this data. The removed data may, in some cases, have been better labelled than the remaining regions (even though the model was "unconfident" on the data. See Section 7.1 for alternative evaluation strategies to account for this.
- The variance of the predictions is zero for the majority of cells. This means that most cells were chosen randomly negating any AL benefit. It is possible that a larger MC-Dropout could have alleviated this.
- Improvements in mIoU on the remaining unlabelled data appear to be more from removing "difficult" regions for the model to learn rather than the model learning from these "difficult" regions and then applying that to produce better predictions.

6.5 Summary

This chapter introduced active learning and the results of our initial feasibility study on our heritage scenes. Our experiments with PointPointNet++ and Random Forest models with a modified ReDAL technique did not improve model performance compared to training directly on the training dataset, in spite of related work finding good results. We chose not to pursue active learning further. In the next chapter, we describe the models used for the remainder of this thesis in detail.

Chapter 7

Experimental Results and Discussion

In this chapter, we present the results of our final experiments across the six datasets. We begin by describing the details of the experimental design and choice of evaluation metrics. This is followed by the presentation of our results and analysis of the outcomes including visualisations of the qualitative performance of several experiments.

7.1 Experimental Design

Typically, models tested against the benchmark datasets report mean classwise Intersection over Union (mIoU), mean classwise accuracy (mAcc), and overall accuracy (OA). For each metric, higher values are better. We focus on reporting mIoU rather than accuracies as it provides a more informative measure for our imbalanced binary labelling task. The majority class may disproportionately influence the other metrics. We also introduce a secondary metric "Effective mIoU", that we define as the effective performance over the entire scene after taking into account the "perfect" labelling of the training set. We use the formula $Predicted\ mIoU * (1 - labelling\%) + labelling\%$ where $labelling\%$ is the percentage of the scene by area which is labelled by the user.

Recall that the primary aim of our research is to investigate the capabilities of modern deep learning models for cleaning point clouds of cultural heritage sites. Specifically, (i) does the relative performance of the models on S3DIS hold on a new dataset, (ii) to what extent can pretraining improve the performance of the models, (iii) how much labelling is needed to achieve greater than 90% mIoU, and (iv) how do these models compare to a naive tree ensemble baseline. Our experiments investigate the effectiveness of our chosen

models – PointNet++, KPConv, Point Transformer, Random Forests and XGBoost – on the scenes described in Chapter 4.

We argue in Chapter 4 that it was not feasible to create a representative training set *and* a validation set for our scenes, as such there are only training and test splits. However, we only tuned the models on the Church scene. The remaining scenes are tested blind so as to not unduly bias the results.

To ensure a comprehensive evaluation of each model's performance, we compare the S3DIS pretrained models with the randomly initialised models on each of our scenes, at each of the four levels of pre-labelling: 2.5%, 5%, 25%, and 50%. We follow this up by reevaluating the resulting trained models on the respective 50% holdout sets. This enables us to investigate potential biases when training on limited data, as it is possible that the model trained on a smaller subset (e.g. 5%) of the scene's data may perform better on the additional unseen data (e.g., the next 20% that would make up the 25% training set), thus skewing the results in favour of the small training splits. Investigating this aspect allows for a more accurate comparison of each model's ability to generalise to a fixed unseen set given differing amounts of training data.

For each scene, we focus on extracting the following key insights from the raw results:

1. *Pretrained vs Randomly Initialised Weights* for the three deep learning models. How much does the pretraining affect the performance?

2. *Testing vs 50% Holdout* for each of the models. How do different training percentages perform when labelling the same 50% holdout of the scene? We expect to see an improvement as we increase the amount of training data. Negative results mean the 50% Holdout score was worse; i.e. the hypothesis that some training data is easier than others is upheld. This happens when the "extra" points (e.g.for a 5% training there is an extra 45% of the scene to evaluate on) are "easy" to predict and unfairly skew the performance. This could happen for instance where there is a large simple region (e.g.ground plane) in these "extra" points that are providing a "buffer" for the performance.

3. *Effective mIoU* is a metric we define as the overall labelling performance of the scene when taking into account the "perfect" labelling of the training data. This measures the real-world effectiveness of using the models for labelling. We can further evaluate this by measuring the *Total Labelling Effort* needed to fully label the scene. This is derived from the effective mIoU as $100\% - \text{effective mIoU } \% + \text{training labels } \%$.

4. *Performance relative to Random Forests*, is there value in the significantly greater training times and compute requirements to train a neural network?

5. *Accuracy*, while potentially a problematic metric for imbalanced scenes, allows us to compare against the previous results from Marais et al. [67]. When comparing the accuracies of the models, we also compare the total time required for training and classification of each scene. Any improvements from our experiments should not be attributable to a faster device, as the device on which we run the Random Forest and XGBoost experiments is slower than that used by Marais et al. [67].

   It is important to reiterate that Marais et al. [67] examined single scans which result in different point densities and geometric representations. Whilst this makes for a less direct comparison, the aim of this comparison is to determine whether this change to a registered point cloud can directly improve results.

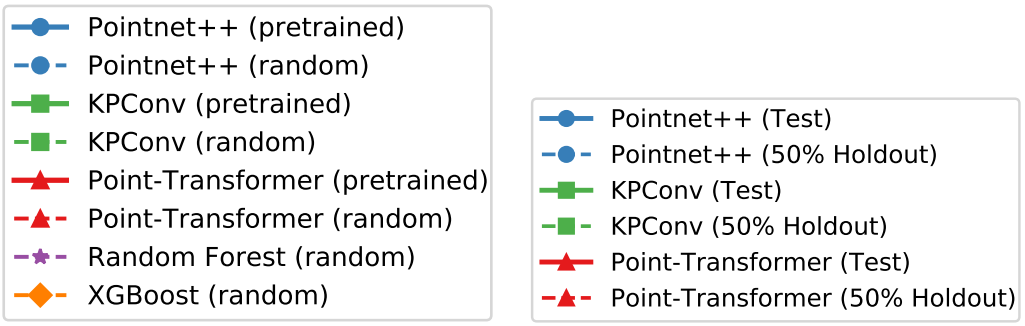
7.2 Presentation of Results

A more familiar approach to the presentation of such results would be to group the experiments together based on the research question or insight to be explored. The results from each of the datasets would then be presented and discussed together in one section. For this thesis, we feel that this would not be the most effective format for communicating our results. Consider, for example, the *Pretrained vs Randomly Initialised Weights* experiment. Using the typical approach we would need to present the results of six models, trained on 4 training splits, for five scenes, and evaluated on both a test and 50% holdout set for each, for a total of 240 results for this experiment.

Instead, we present our experiments grouped by dataset. For each dataset, we present the results and discuss each of the insights above, allowing for a more focused discussion. We end by discussing any trends identified across the datasets between the models.

These results and insights are presented using six line graphs. For consistency, we use the same layout for each of the six scenes. Rather than repeating information, we show the graph legends once in Figure 7.1. Note that the first five graphs use *legend (a)* and combine the results for both the pretrained and randomly initialised models on the Test data. *Legend (b)* is reserved for the "mIoU Improvement via Pretraining" graphs and combines the Test and 50% Holdout results. Note also that while the layout is the same the *scale* used by the graphs may be different between the six scenes.

In addition to these key insights, we examined several additional views of the data. For completeness we provide these, as well as all our raw results, in Appendix A.1 as we felt did not meaningfully add to the conclusions of the study.



(a) (b)

FIGURE 7.1: The two legends used by the six results line graphs, (a) for all except Improvement via Pretraining which uses (b)

7.3 Bagni di Nerone

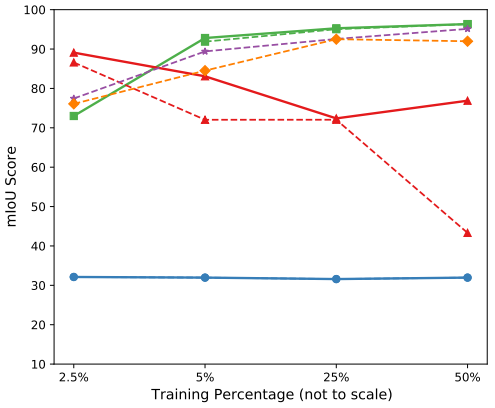
Figure 7.2 presents the key results for the experiments on Bagni di Nerone. We expect this dataset to be relatively simple to label given that it is segmented largely based on location rather than geometry. Due to corrupted recording of the randomly initialised 2.5% training run for KPConv, we are unable to provide results for that experiment.

PointNet++ fails to generalise to the Test data, falling into a local minimum of predicting only keep labels as the majority class. It is unable to learn the training set effectively. When the training set is kept small at 2.5%, the model achieves a 73% mIoU falling to $53 \pm 2\%$ on the remaining training set sizes.

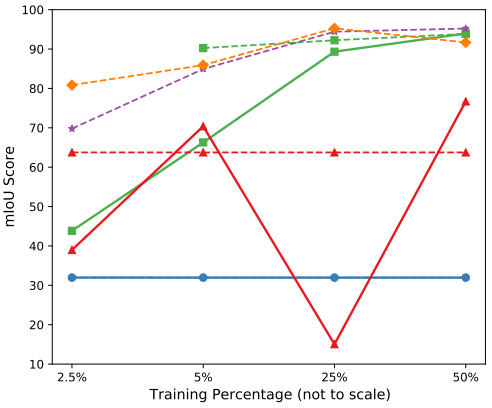
KPConv demonstrates the best performance on this dataset for both pretrained and randomly initialised versions, exceeding 92% mIoU when finetuned with just 5% training data, and 96% when fine-tuned on 50% of the scene.

In Figure 7.4 we show an overview of the scene for the 2.5% model with both the target and predicted results. This includes a closeup of one of the problematic areas in which the ground truth labels appear to be incorrect.

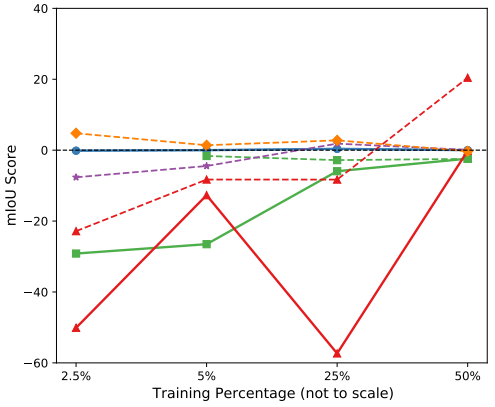
Looking at the overall predictions, KPConv appears to segment the "thin" objects, and to mostly keep the larger surfaces. This works well in the main site but leads to trees and the buildings on the periphery being poorly labelled. Figure 7.3 shows the front face of the main building and one of the trees as the only areas in which the 50% trained model makes incorrect predictions. It is not clear why the model struggles with the tree. The building surfaces that are incorrectly predicted are scanned with the same detail as the other areas of the building but contain several holes which are difficult to convey in a 2D image. The holes may be detected as similar to the noisy tree canopies.



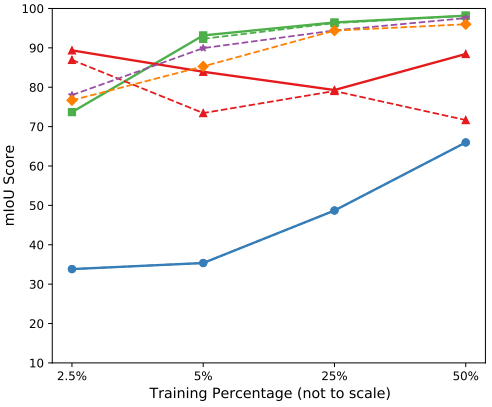
(a) Test mIoU



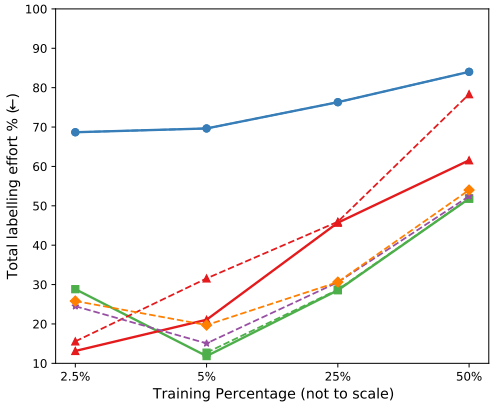
(b) 50% Holdout mIoU



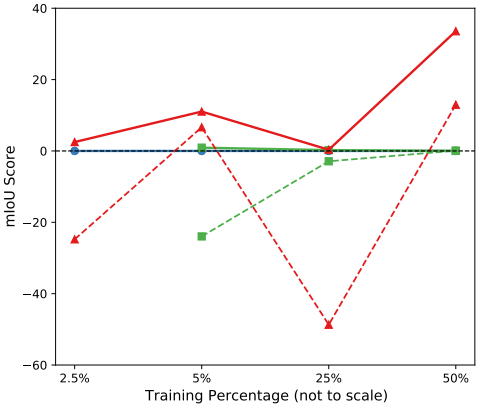
(c) Holdout Difference mIoU



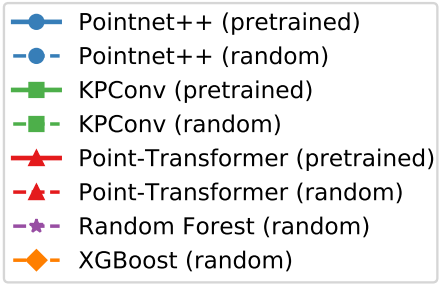
(d) Effective mIoU



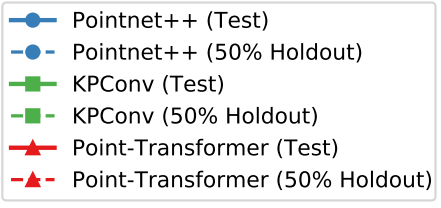
(e) Total Labelling Effort



(f) mIoU Improvement via Pretraining



(g) Legend for the first 5 graphs



(h) Legend only for Improvement via Pretraining

FIGURE 7.2: Bagni di Nerone: Line graphs comparing the mIoU% of the models at different training percentages

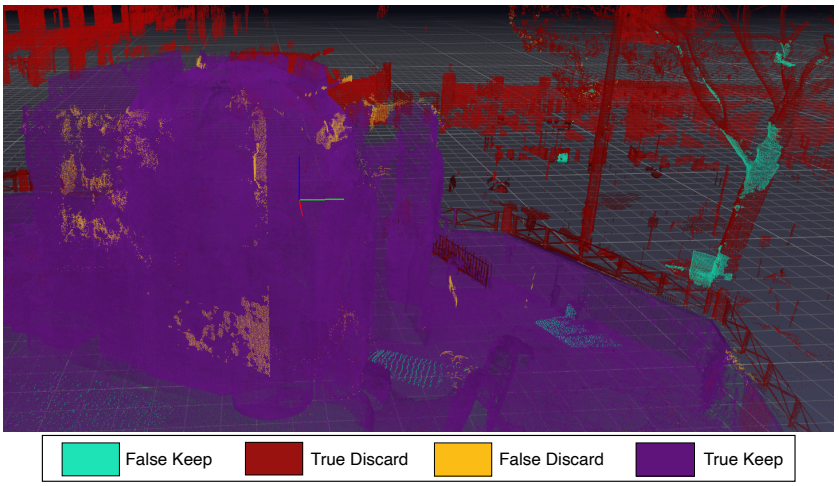


FIGURE 7.3: Bagni di Nerone: 50% KPConv Pretrained model incorrect predictions.

In Figure 7.4(c) there are two ground patches incorrectly labelled discard, and some thin scanner errors that are incorrectly labelled keep. In the predictions for those areas, the model correctly learns that the ground and scanner errors should be labelled keep and discard respectively. Unfortunately, in this same figure, we can see the model incorrectly labels the railings on the right hand side of the stairs.

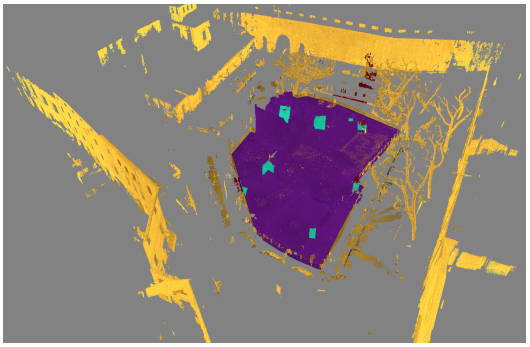
In practical terms, this is still fairly useful as the labeller would first discard the outer regions with a single polygon selection and spend the majority of their labelling effort inside the area of interest.

7.3.1 Pretrained vs Randomly Initialised Weights

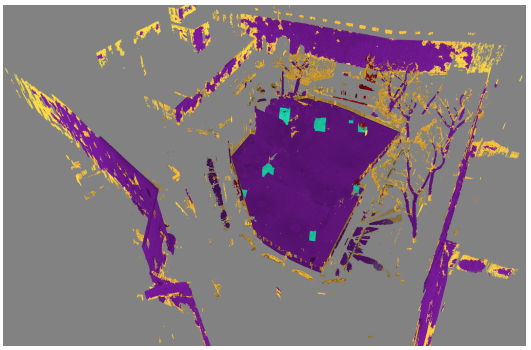
As can be seen in Figure 7.2(f), pretraining does not greatly affect the performance of PointNet++ or KPConv on the regular Test data. For Point Transformer, it does generally increase performance, although note that the large jump on the 50% training data is due to unusually poor performance from the randomly initialised run. This run collapsed unexpectedly and did not recover.

Whilst KPConv and both tree-based models predictably trend up as the training percentage is increased, PointNet++ and Point Transformer struggle to learn the scene.

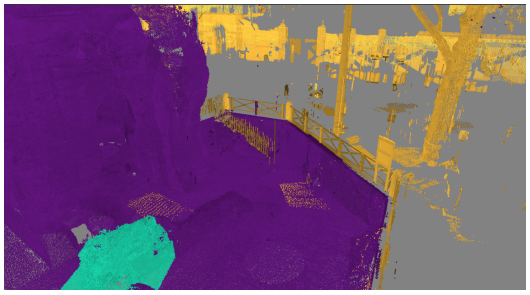
The Point Transformer, which kept a more stable performance when pretrained, unexpectedly performs very poorly on the 25% training run with the 50% holdout set. In this instance, the model performs worse than randomly guessing or predicting the majority class.



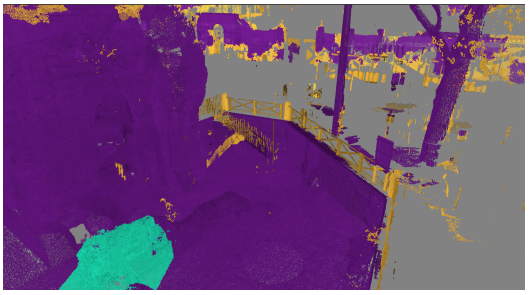
(a) 2.5% Ground Truth



(b) 2.5% Predictions



(c) 2.5% Ground Truth Back Corner Railings



(d) 2.5% Predictions Back Corner Railings

FIGURE 7.4: Bagni di Nerone: KPConv Pretrained prediction results with 2.5% training data.

It is unclear what causes the large drop in the 50% randomly initialised point transformer's performance. Unlike the other randomly initialised runs on this scene, this run rapidly collapsed to predicting a single class.

7.3.2 Testing vs 50% Holdout

As shown in Figure 7.2(c), with the exception of XGBoost the 50%*holdout* set performance was worse than on the Test data. The exception to the trend is XGBoost which starts off with slightly better holdout performance at very small training set sizes, it is unclear what the cause may be.

The reduced performance from the models on the 50% holdout set is expected. The 2.5 and 5% runs are of particular interest as the initially lower performance for KPConv, Point Transformer and the Random Forest validates our suspicions that simply looking at the Test data for each experiment could skew the results. This supports the hypothesis that increasing the training data will increase the models' performance on the holdout data.

7.3.3 Improvement via Training Percentage

As we increase the training data, the performance on the 50% holdout steadily increases, with the exception of an unusually low 25% score for the pretrained Point Transformer. Both PointNet++ models and the randomly initialised Point Transformer do not change their results with increased training data on the holdout set, suggesting that they were unable to learn anything meaningful for this scene. However, whilst we know that while PointNet++ simply predicted the majority class, the Point Transformer was able to learn at least some patterns, as it consistently resulted in a mean accuracy of 73%.

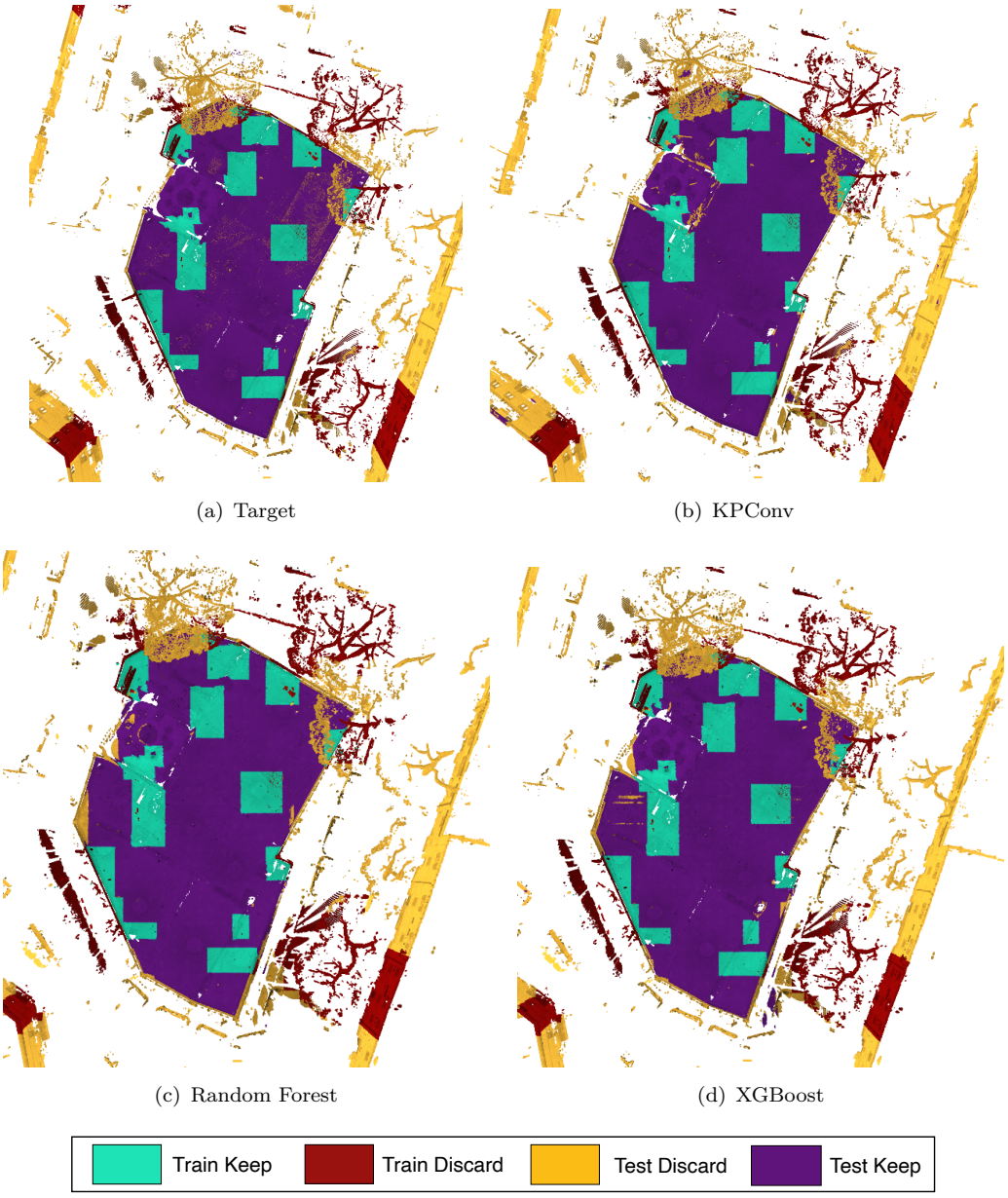
7.3.4 Effective mIoU

The effective mIoU and consequently, the total labelling effort are presented in Figures 7.2(d) and 7.2(e). Having labelled only 5% of the scene, KPConv achieves an mIoU of 92.8%, giving an effective mIoU of 93.2%, leaving only 6.8% of the scene to correct. A total labelling effort of 11.8%.

We can see that 5% training data is the optimal amount of training data across all the models tested to minimise the total labelling effort needed. Another way to look at this is that it has the greatest labelling efficiency. Despite higher mIoUs (including effective mIoU) and better results on the 50% holdout tests when using $25 - 50\%$ training data, in the real world, the goal is to reduce labelling effort.

7.3.5 Performance Relative to Random Forests and PointNet++

In terms of real-world efficiency, and to make the comparison to our benchmarks obvious, all models outperform PointNet++ (except for the aforementioned abnormally low 50% holdout experiment for the 25% Point Transformer) by 50% on average due to PointNet++ failing to learn. This is the only scene in which this problem is so pronounced. However, the fastest model to train is the Random Forest model, training in 4.25 minutes with 5% training data for a total labelling effort of 15%. In Figure 7.5 we illustrate the difference between the KPConv, Random Forest, and XGBoost models on the 25% training split.



(a) Target (b) KPConv

(c) Random Forest (d) XGBoost

FIGURE 7.5: Bagni di Nerone: Comparison of KPConv, Random Forest, and XGBoost labelling with 25% training data, top-down view.

Model (Training labelling %)	**Overall Accuracy %**	**Time (minutes)**
Marais et al. [67] RF post-processed	98.1	75
PointNet++ (2.5% / 5%)	63 ± 0.5	163 / 151
KPConv (2.5% / 5% pretrained)	84.4 / 96.3	217 / 239
Point Transformer (2.5% / 5% pretrained)	**99.5** / 96.2	67
RF (ours) (2.5% / 5%)	87.6 / 94.4	**0.4** / 0.42
XGBoost (2.5% / 5%)	86.8 / 91.6	0.58 / 0.51

TABLE 7.1: Best Overall Accuracy and processing time on Bagni di Nerone for each of our models (less than 5% training labels) andMarais et al. [67], including time required. Best accuracy and in **bold**.

7.3.6 Comparison to Previous Works

As shown in Table 7.1, in their single scan random forest-based approach, Marais et al. [67] achieved an average (across the 15 scans) overall accuracy of 97.2/98.1% with/without post-processing. Their system took one hour and fifteen minutes compared to an expert labelled, who took one hour and thirty-five minutes.

The Point Transformer achieves the highest accuracy in a shorter time than Marais et al. [67]. Whilst the Random Forest falls short of Marais et al. [67] by 3.7%, it takes the shortest total amount of time being 30× faster than the Point Transformer. Although the Random forest underperforms both of these approaches, there is significantly more time available for correcting the remaining regions. This may make the overall labelling *time* faster at the cost of more manual labelling to correct the scene.

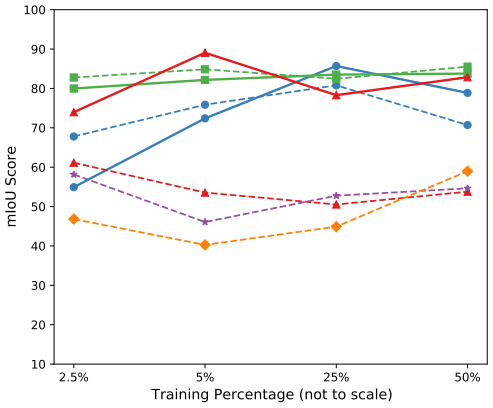
In summary, these experiments on Bagni di Nerone have shown that KPConv and the tree-based models, such as Random Forest and XGBoost, perform well compared to PointNet++ and the Point Transformer. In spite of the strong accuracy results for the 2.5% Point Transformer, the dip in performance at $5-25\%$ training data suggests that it is not learning as effectively as it may appear. The best real-world performance was achieved by KPConv with 5% training data, resulting in the lowest total labelling effort of 11.8%. Furthermore, the Random Forest model stands out for its fast training time and competitive accuracy, making it an attractive option for practical applications.

7.4 Church

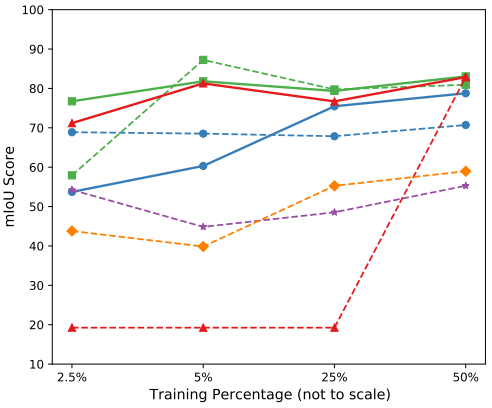
Figures 7.6(a) and 7.6(b) show KPConv to perform the most consistently. On the Test set KPConv has the highest average mIoU and is within the two highest mIoU scores for each training set size. On the 50% holdout, KPConv outperforms all other models.

As expected, increasing the amount of training data generally leads to improved performance on the 50% holdout set. However, it is important to note that this increase is not consistently observed in the randomly initialized deep learning models, with the exception of a sudden spike at 50% training for the Point Transformer model and an early improvement from 2.5% to 5% for the randomly initialized KPConv model.

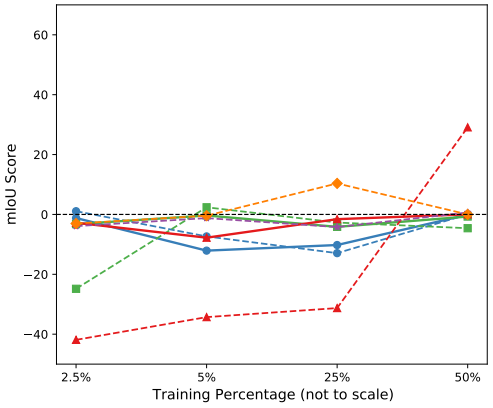
Point Transformer has more variable performance, although, with 5% training data, it is the best-performing model in terms of total labelling effort (Figure 7.6(e)). However, it shows a slight decrease in effective mIoU for both pretrained and random weights as the training data increases, this warrants further investigation. The performance drops



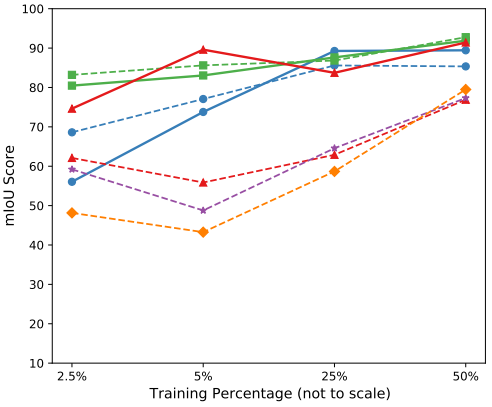
(a) Test mIoU



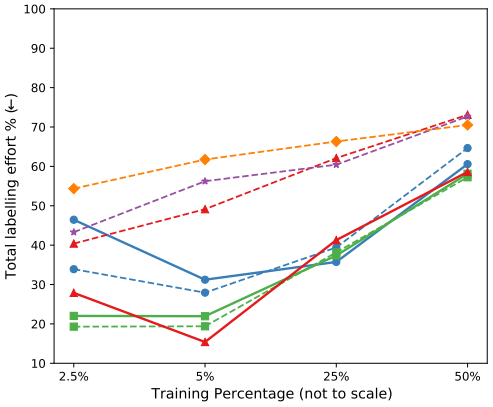
(b) 50% Holdout mIoU



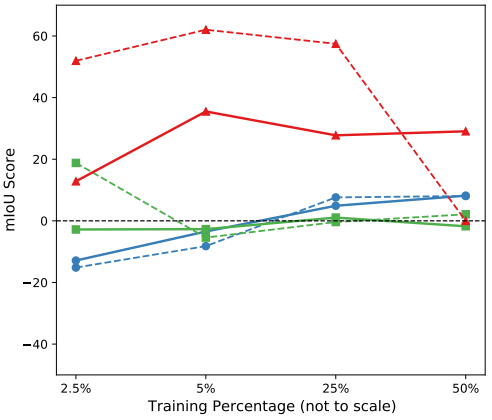
(c) Holdout Difference mIoU



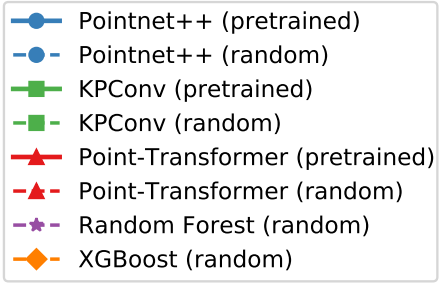
(d) Effective mIoU



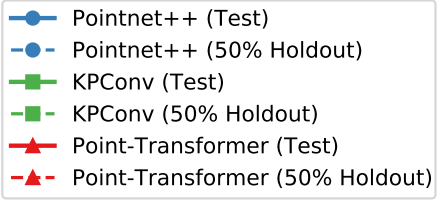
(e) Total Labelling Effort



(f) mIoU Improvement via Pretraining



(g) Legend for the first 5 graphs



(h) Legend only for Improvement via Pretraining

FIGURE 7.6: Church: Line graphs comparing the mIoU% of the models at different training percentages

significantly when using random weights, with an unusual jump in holdout performance on the random initialisation 50% training run.

PointNet++ as our baseline has the lowest performance of the deep learning models (except for the randomly initialised Point Transformer) on the holdout set and is typically the lowest on the test sets (not including the randomly initialised Point Transformer).

Both tree-based models struggle to achieve more than 50% mIoU on either the test or holdout sets.

To summarize, KPConv is the best model for the Church dataset, followed by Point-Transformer (with pretrained weights). The distinctive difference between most of the keep and discard objects (surfaces vs pipe-like objects) is likely to have been easily differentiated by both of these models. The poor performance of the tree based models, and to an extent PointNet++, is likely due to the need to extract specific detailed geometries, such as thin scaffolding and gates. Since the tree based models were naive baselines operating on raw XYZ coordinate data, it is unlikely for them to learn such complex interactions. However, as we will demonstrate in subsequent scenes, these models excel when the dataset is simpler and when labelling boundaries can be learned based on the location in the scene rather than the geometry of the objects.

7.4.1 Pretrained vs Randomly Initialised Weights

Figure 7.6(f) shows pretraining has a significant impact on Point Transformer performance, a ±30% improvement. For both KPConv and PointNet++, the pretrained models have higher mIoU scores compared to their counterparts with random initialization at higher training set sizes. However with lower amounts of training data the random initialised models perform slightly better.

We hypothesise that the randomly initialised models are able to more quickly learn details that are useful to this specific scene without the biases from pretraining on a larger scene, and once there is a more varied set of training samples the general patterns learnt during pretraining help the model to generalise. This indicates that transfer learning is beneficial for these deep learning models when dealing with the Church dataset.

7.4.2 Improvement via Training Percentage

For all models increasing training data improves the performance of the remaining data. With lower percentages of data, deep learning models, especially KPConv, show a clear

Model	**Overall Accuracy %**	**Time (minutes)**
Marais et al. [67] RF	99.1	14.75
PointNet++ (2.5% / 5% random init.)	98.3 / 96.9	247 / 264
KPConv (2.5% / 5% random init.)	98.6 / 98.8	135 / 104
Point Transformer	96.4 / **99.6**	63
(2.5% / 5% pretrained)		
RF (ours) (2.5% / 5%)	98.3 / 88.4	**1.7** / 1.8
XGBoost (2.5% / 5%)	89.3 / 78.2	1.8 / 2.4

TABLE 7.2: Best Overall Accuracy and processing time on Church for each of our models (less than 5% training labels) andMarais et al. [67]. Best accuracy and time in **bold**.

advantage over the tree-based models. As the percentage of data increases, the tree-based models start to catch up, with Random Forest even surpassing Point-Transformer with random weights at 25% and 50%.

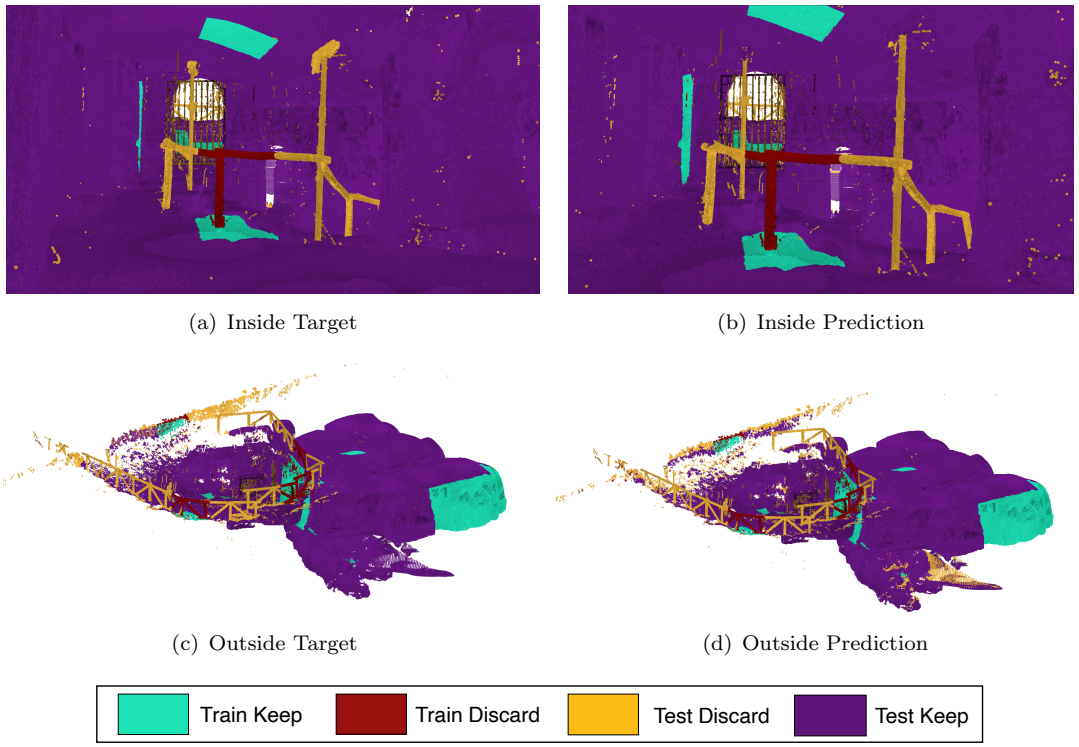
The correlation between training data and performance is easily attributed to the varying geometry and small amount of discarded objects making it difficult to select an adequately representative sample when training percentages are lower. The lower performance of the tree models can be attributed to the complexity in this scene requiring the model to learn the specific geometric patterns rather than a more simple positional pattern as in the previous Bagni di Nerone scene. Learning these would require trees to be grown too deeply and would likely result in worse overfitting. The dip in performance for the tree models with the 5% training set is likely due to a removal of a "easy" region for the model to label.

7.4.3 Effective mIoU

Examining Figure 7.6(e) we can evaluate the real world expected labelling effort (and thus time) needed. This is an alternative view to the accuracy based comparison presented in the following section.

KPConv and the pretrained PointTransormer models make the most effective use of the training data at small percentages with the 5% pretrained Point Transformer needing only an additional 10.4% of the scene to be corrected.

In Figure 7.7 we show the qualitative performance of the randomly initialised KPConv model with 5% training.



(a) Inside Target (b) Inside Prediction

(c) Outside Target (d) Outside Prediction

FIGURE 7.7: Church: Target and predicted labellings by the randomly initialised KP-Conv with 5% training data

7.4.4 Comparison to Previous Works

Marais et al. [67] achieved an overall accuracy of 99.1% in approximately 15 minutes. Although the 5% pretrained Point Transformer achieved the highest accuracy of 99.6% (correlating with the highest mIoU at 89.1% and 87.2% on the test and holdout sets respectively), the processing time was still significantly longer than any of the three tree-based methods. Our random forest is within 1% of the accuracy of Marais et al. [67] whilst taking 11.5% of the time, leaving an additional 13 minutes to correct for the error.

It is worth noting that the imbalanced nature of this dataset (95.4% keep) makes accuracy a challenging metric to compare fairly in this instance. It is tempting to conclude from Figure 7.6(a) that there is generally a correlation between the accuracy and the mIoU. However, this is not always the case. PointNet++ trained on 2.5% of the scene achieves an accuracy of 98.3% but an mIoU of only 54.9%. Conversely, when trained on 25% of the scene the accuracy drops to 96.8% but the mIoU rises to 85.7%, the highest for PointNet++ on this scene. The drop in accuracy accompanied by the rise in mIoU means that whilst overall the model labelled fewer "keep" points correctly, it was less likely to incorrectly label the points that should be marked as "discard". We know this because the scene is heavily skewed towards having more points labelled as "keep".

7.5 Lunnahoja

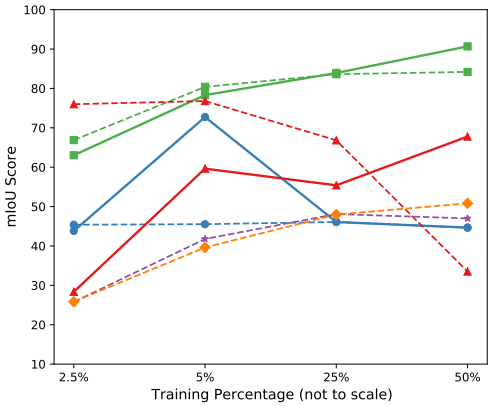
The Lunnahoja cabin helps explore how the models perform when the labelling decision is entirely constrained to a single bounding box, albeit with noisy ground truth labels. We hypothesised that the simple nature of the cleaning would be of benefit to the simpler tree based models, although practically speaking the user will always label such a trivial scene by hand. The labelling near the cabin is noisy with many of the ground plane points labelled as discard/keep semi-randomly (Figure 4.9). This could make it difficult for the models to learn particularly when training data is kept small.

PointNet++ nearly immediately (within the first epoch) collapses to predicting only a single class (discard) on all but the 5% training split. The 2.5% model's training is highly erratic with test loss ranges of over 1000 units, yet despite this the Test mIoU curve is smooth. On the other hand, the training loss and both losses on the 5% model staying below 1, despite the 5% model only predicting discard.

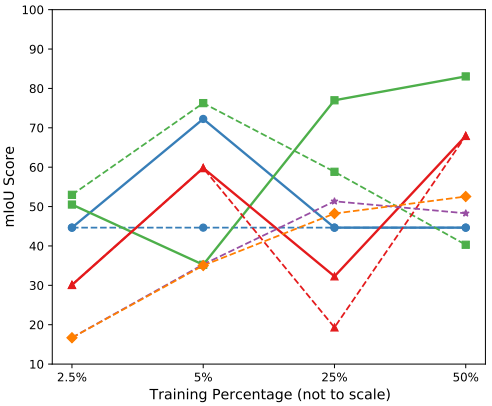
This is unexpected, we would expect that more training data would lead to better generalisation. As this is the only example of PointNet++ learning on this scene across 8 different experiments something is clearly making it more difficult to learn. It is tempting to believe the hyperparameters are merely poorly tuned or that the model is heavily over/underfitting the training data. However, examining the training/test loss curves in Figure 7.9 is not so simple (note the change from linear to log scale). It is not clear to the authors why the model only learns the scene in this one specific instance when the other models do not exhibit similar inabilities to learn.

KPConv is the best performing model, although we see a greater difference between the test and holdout sets than in the previous two scenes. KPConv also requires more data than the previous two scenes, only breaking the 80% mIoU mark after finetuning on the 25% training data. In Figure 7.10 we show that on both the training and test data the model learns a more organic boundary (rather than the straight boundary in the ground truth) between the log cabin and the remainder of the scene. The model also corrects many of the "errors" in the ground truth labelling. On the test set a similarly organice boundary is seen. Overall KPConv is generally accurate except for several patches of "keep" labels predicted in the ground plane. Importantly the model accurately separates the noisy to be discard from the interior of the building.

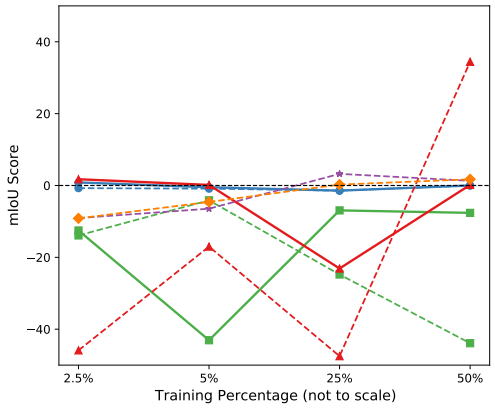
Unlike the previous two scenes, the tree-based models are unable to learn this scene at all, and in fact, do worse than simple random guessing or predicting the majority class. This is due to the cabin facing diagonal to the XY axis (see Section 3.2), and the test data extending beyond the bounds of the training data in 3D space. In Figure 7.11 we illustrate the issue using the 50% training split.



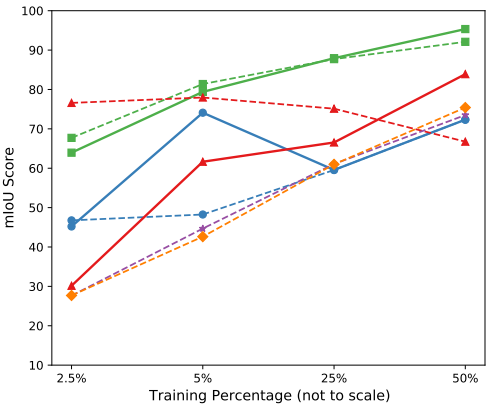
(a) Test mIoU



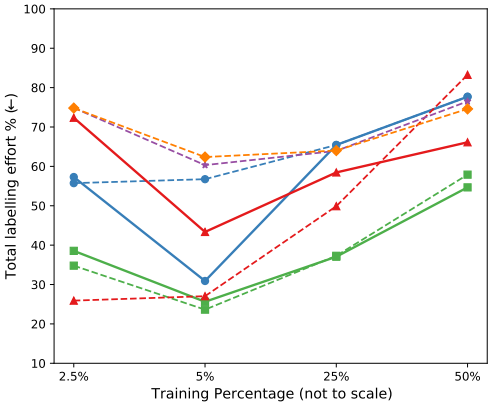
(b) 50% Holdout mIoU



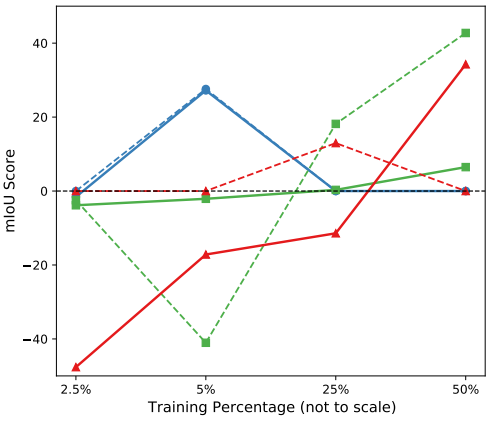
(c) Holdout Difference mIoU



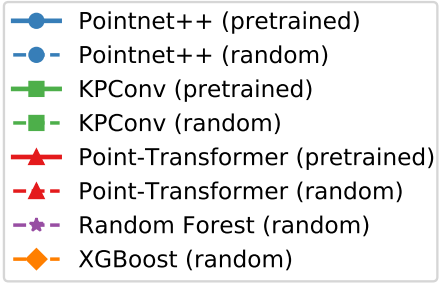
(d) Effective mIoU



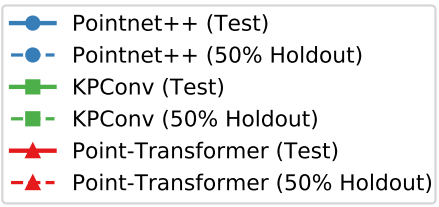
(e) Total Labelling Effort



(f) mIoU Improvement via Pretraining

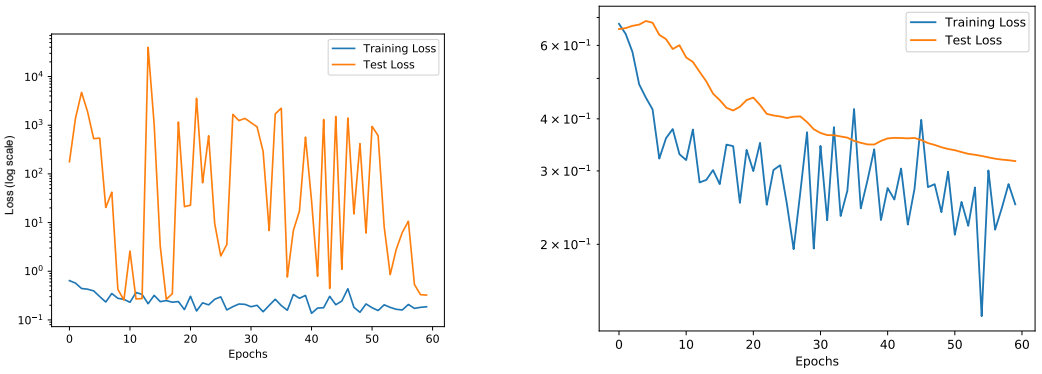


(g) Legend for the first 5 graphs



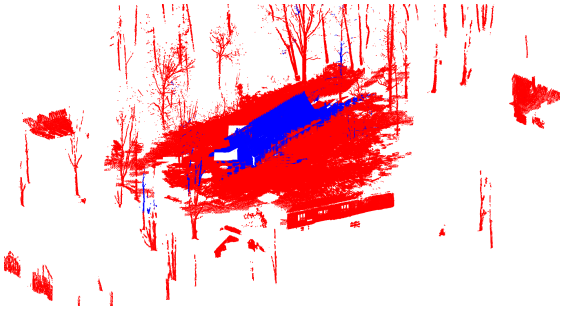
(h) Legend only for Improvement via Pretraining

FIGURE 7.8: Lunnahoja: Line graphs comparing the mIoU% of the models at different training percentages



(a) 2.5% Model Loss Curves

(b) 5% Model Loss Curves



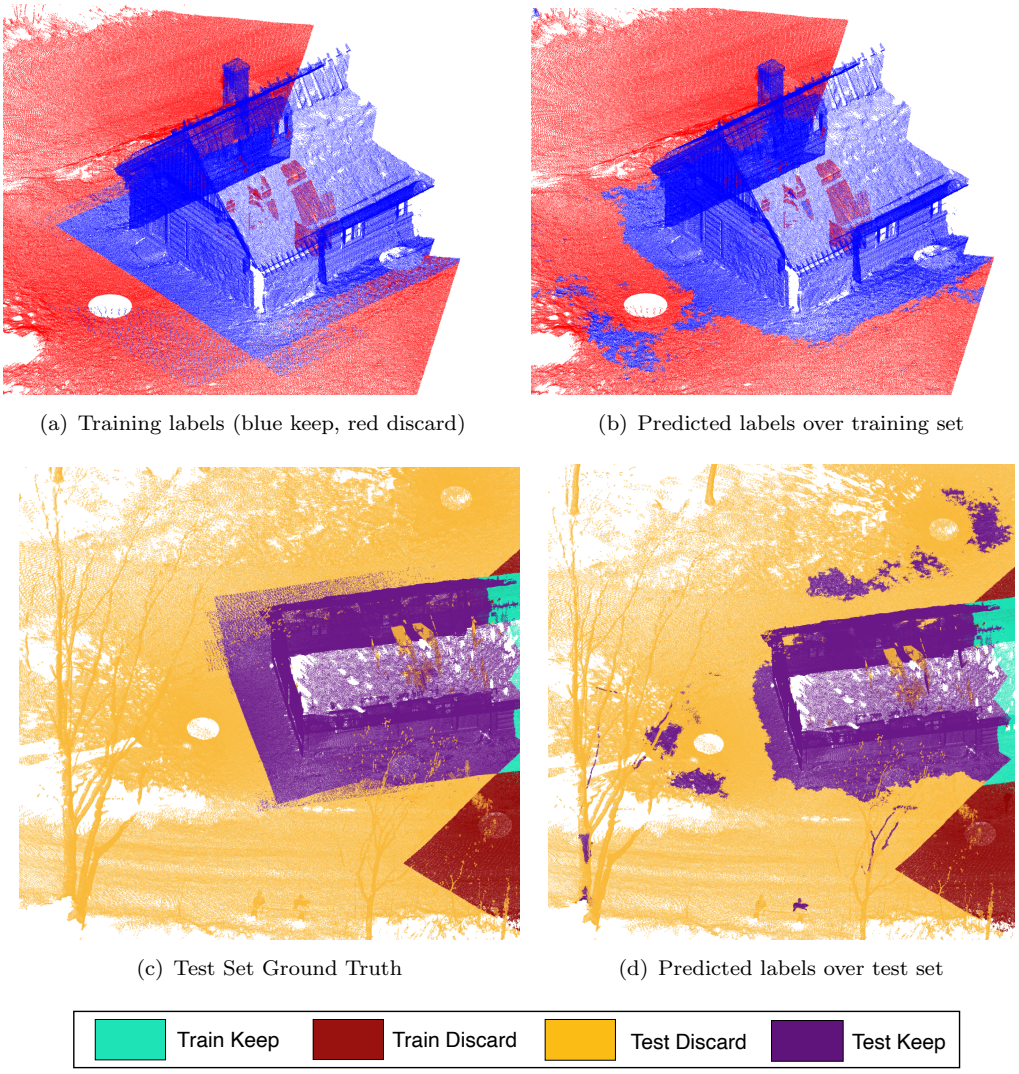
(c) 5%Model predictions, blue is keep, red is discard

FIGURE 7.9: Lunnahoja: PointNet++ pretrained 2.5% vs 5% test set results

7.5.1 Pretrained vs Randomly Initialised Weights

Pretraining appears to have a small positive impact, with some outliers raising mIoU scores by 20 – 40% as shown in Figure 7.8(c). The major exception was the 2.5% Point Transformer where pretraining reduced performance drastically. Both the 5% and 25% are also worse when pretrained, though to a lesser degree.

Figure 7.12 compares the test loss curves of both 2.5% training runs, the training loss' match closely and are omitted from the graph for readability. Notably the model is neither predicting a single class nor random labels, instead, it has learned incorrectly, suggesting that the 2.5% training set may not fully represent the rest of the scene. The divergence of the pretrained model suggests that the scene is too different from the S3DIS



(a) Training labels (blue keep, red discard)

(b) Predicted labels over training set

(c) Test Set Ground Truth

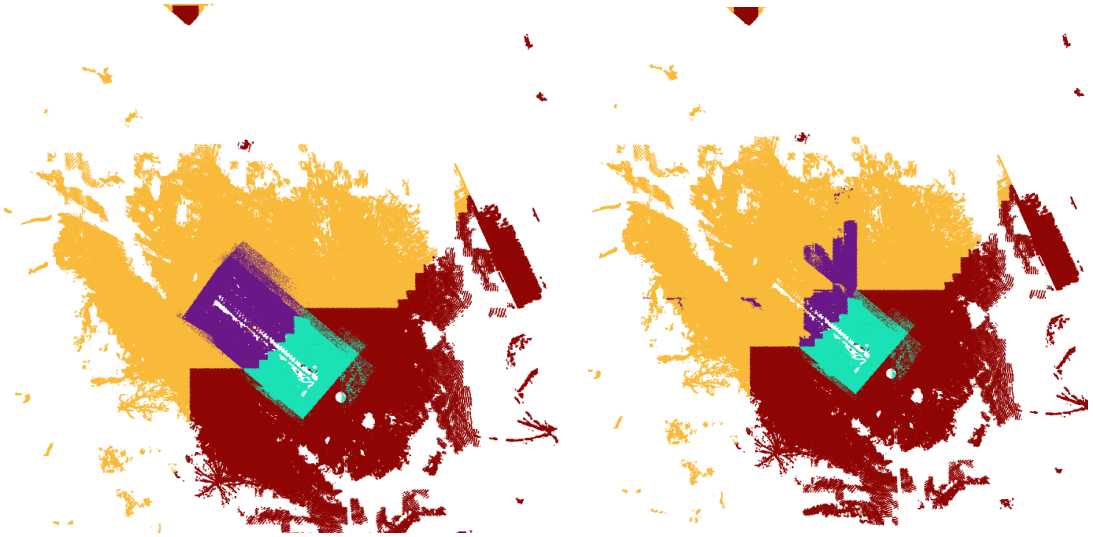
(d) Predicted labels over test set

FIGURE 7.10: Lunnahoja: On the 50% training set, KPConv (pretrained) learns an organic boundary whilst partially correcting for the noise in the training labels.

scene to allow for transfer learning to take place, or that the model is overly sensitive to initial conditions. With more time, additional training runs would have allowed us to explore this discrepancy further.

7.5.2 Testing vs 50% Holdout

Both KPConv and Point Transformer show a notable decrease in performance of $20-45\%$ between the testing and holdout sets (with the exception of the Point Transformer randomly initialised 50% model). This is more apparent in the randomly initialised models.



(a) Target (b) Predictions

FIGURE 7.11: Lunnahoja: Random Forest 50% experiments, top-down view, showing the axis aligned decision boundaries

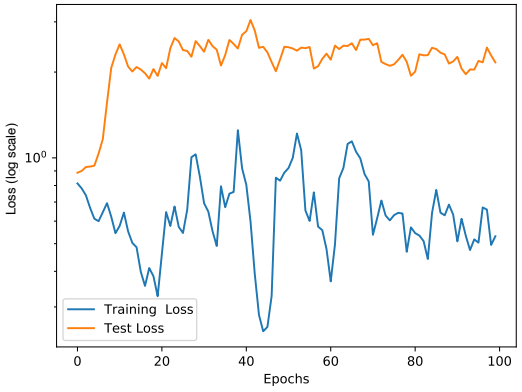


FIGURE 7.12: Lunnahoja: Comparison of the test loss curves of the pretrained and randomly initialised Point Transformer models when trained on the 2.5% training split.

7.5.3 Effective mIoU

All models show a steady increase in effective mIoU with increasing training sizes as expected, with the exception of the randomly initialised Point Transformer. The most efficient with respect to the total labelling effort is again the 5% KPConv model, as well as the randomly initialised 2.5% and 5% Point Transformer. However, the 2.5% Point Transformer is merely "getting lucky", as the average class accuracy is only 63% compared to over 90% for the 5% KPConv models.

We do not compare to the previous works by [67] for this scene as it is a new scene and not part of their original study. We do not feel the accuracy results for our models alone significantly add to the discussion.

7.6 Montelupo

Montelupo represents a mix of interior and exterior regions similar to the Church scene, although it is less complex inside and contains more foliage and flat surfaces. The labelling criteria are a mix of object types and positions in the scene.

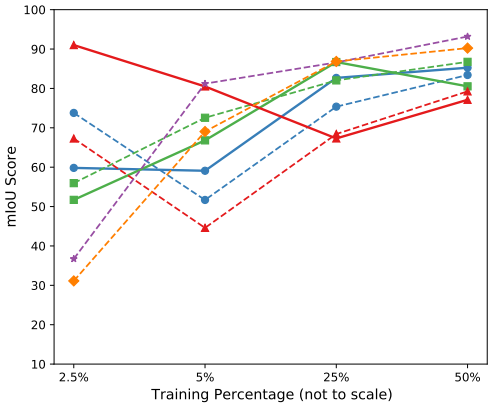
7.6.1 Pretrained vs Randomly Initialised Weights

Pretraining improves the performance of the Point Transformer the most (20-50%), with both PointNet++ and KPConv showing slight improvements at 5/25% training for the test sets.

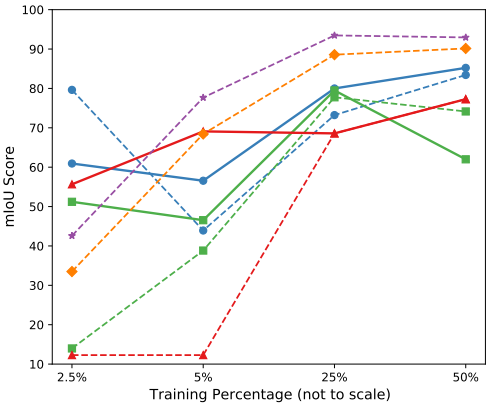
On the Holdout set, pretraining mainly improves performance for models trained with 2.5/5% training data, except for PointNet++ with 2.5% training data which did unusually better on the 50% holdout set.

7.6.2 Testing vs 50% Holdout

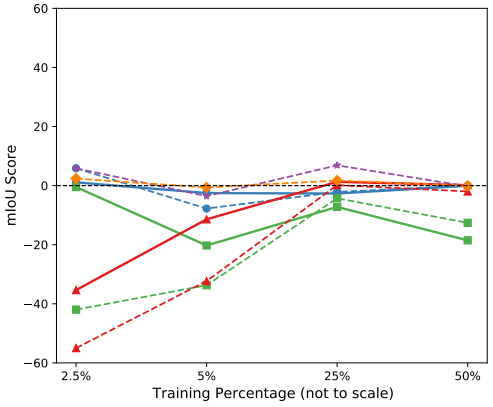
For both KPConv and Point Transformer performance on the holdout set is significantly worse, greater than 30%, on small training sizes and equalises at the 25% training split. PointNet++ and the Decision Tree models maintain similar mIoU scores between the test and holdout sets regardless of training data size.



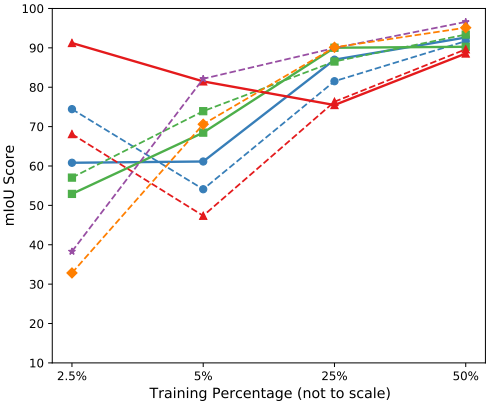
(a) Test mIoU



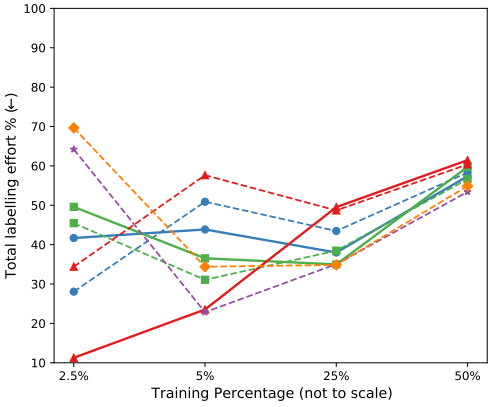
(b) 50% Holdout mIoU



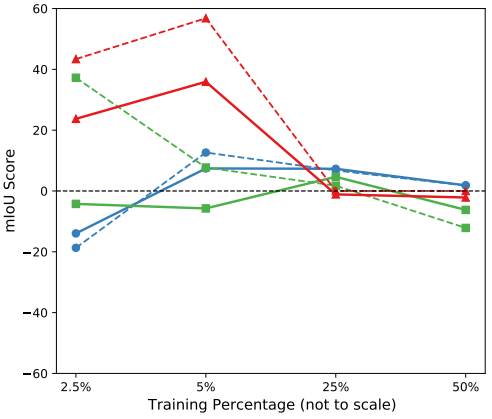
(c) Holdout Difference mIoU



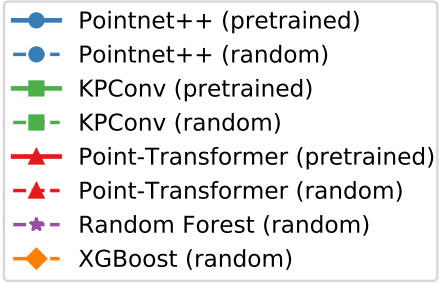
(d) Effective mIoU



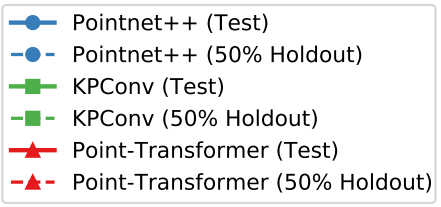
(e) Total Labelling Effort



(f) mIoU Improvement via Pretraining



(g) Legend for the first 5 graphs



(h) Legend only for Improvement via Pretraining

FIGURE 7.13: Montelupo: Line graphs comparing the mIoU% of the models at different training percentages

Model	**Overall Accuracy %**	**Time (minutes)**
Marais et al. [67] RF post-processed	93.7	81
PointNet++ (2.5% random init. / 5% pretrained)	88.7 / 79.7	169 / 226
KPConv (2.5% / 5% random init.)	76.8 / 85.3	282 / 232
Point Transformer (2.5% / 5% pretrained)	**95.7** / 90.1	60
RF (ours) (2.5% / 5%)	65.6 / 90.3	**0.19** / 0.20
XGBoost (2.5% / 5%)	62.2 / 83.4	0.25 / 0.3

TABLE 7.3: Best Overall Accuracy and processing time on Montelupo for each of our models (less than 5% training labels) and Marais et al. [67]. Best accuracy and time in **bold**.

7.6.3 Improvement via Training Percentage

Increasing the training data slowly increases the mIoU on both test and holdout sets, with the largest improvements being seen on the randomly initialised models. This is the only scene where the randomly initialised models are far more improved with additional data compared to the pretrained models.

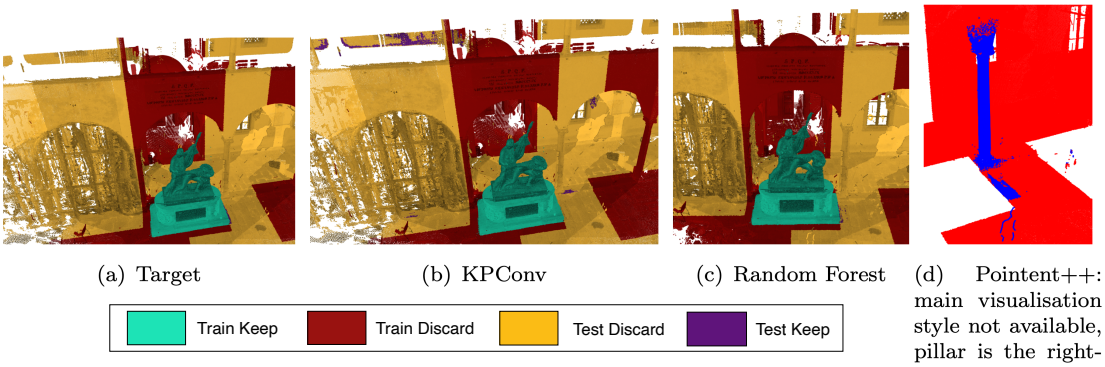
7.6.4 Effective mIoU

For both pretrained and randomly initialised models, an mIoU of above 80% is achieved with 25% or more training data. The pretrained Point Transformer has unexpectedly good effective performance at small training amounts but this drops off until the 50% training experiment. Unfortunately without the ability to visualise this, it is unclear if this translates to an effective labelling in the real world.

Although both Decision tree models exceed 95% with 50% training data, in the context of reducing the labelling effort the pretrained Point Transformer with 2.5% training data is the best performing model. At the 5% labelling stage, the Random Forest matches the Point Transformer, however, it is significantly faster to train and label the scene as we show in Table 7.3 below. Outside of these two examples labelling this scene requires a total labelling effort of 30-50% regardless of whether it is done before training or as corrections after the models predicted labels.

7.6.5 Comparison to Previous Works

Marais et al. [67] achieved an overall accuracy of 93.7%, beaten only by the 2.5% trained Point Transformer. Although not included in the above table, PointNet++ once again only reaches 90+% accuracies when pretrained and 25% of the scene is labelled, the



(a) Target (b) KPConv (c) Random Forest (d) Pointent++: main visualisation style not available, pillar is the right-hand side of the statue, red is keep, blue is discard.

FIGURE 7.14: Monument: Qualitative comparison of results for KPConv, PointNet and Random Forest trained on 25% of the scene. Illustrates the imbalance in the chosen training split.

same point at which the mIoU increases above 80%. Given the fast training times of the random forest model, and the relatively small amount of data in the scene, the additional labelling effort from 2.5-5% increases the model accuracy far more than the additional training data added.

7.7 Monument

Monument is our second "trivial" scene to label. Similar to Lunnahoja, a single bounding box is sufficient to fully segment the statue from the scene. Unlikely Lunnahoja this scene contains more complex geometry which may "confuse" the training of the models. Due to an oversight during the training split creation, the entire test portion of the 50% split is composed of discard points, as well as the vast majority of the 25% test region (0.03% labelled keep). This confounds the mIoU metric as it is not possible to exceed 50% mIoU on the 50% training experiments. As a result of this, the majority of the models trained learn to predict the majority class. We acknowledge that due to these constraints, this scene cannot be used to draw meaningful conclusions about the models' performance in general and particularly above the 2.5-5% training sets or on the holdout set.

In Figure 7.14 we compare the qualitative performance of the pretrained PointNet++, KPConv, and Random Forest trained on 25% of the scene. Despite mIoU scores of 53%, 69.6% and 61.4% respectively, the Random Forest results in an objectively more "useful" labelling. Although we choose mIoU specifically to address the shortcomings of accuracy on imbalanced datasets, qualitative evaluation is still necessary to ensure the

quantitative results are reflective of the qualitative labelling performance, particularly in severely imbalanced scenes.

7.8 Piazza

The Piazza is the largest and "noisiest" of our datasets, with the majority of objects to be removed being either outlying building surfaces, people and cars, or the inside of rooms. Given the inconsistency in the labelling of the room interiors (see Section 4.6 and Figure 7.16) we expect this to be a difficult scene to label.

The PointNet++ model is not usable, predicting the majority class in almost all instances. The 2.5% training KPConv model begins by randomly predicting keep/discard labels for each sample sphere over flat surfaces and quickly learns to discard the thin "streaky" human points (see Appendix A.1).

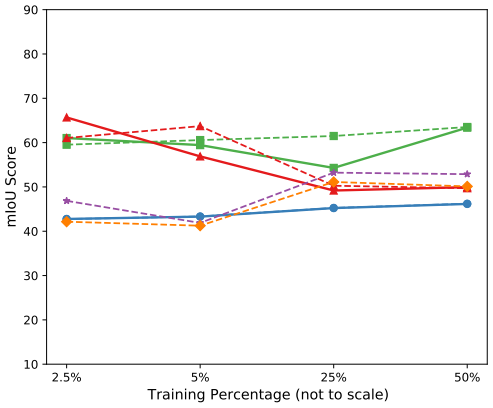
For PointNet++, it is possible that our columnar sampling strategy is partially the cause. Due to the large height variation across the scene, the majority of samples would either be columns containing dense points close to the ground or sparse points extending too high above the ground. The fixed number of points in each sample results in dense and sparse samples respectively, potentially making it difficult to detect noise which is often a low density feature.

After fully training, both KPConv models accurately label the human points, although each continues to struggle with the surrounding structures and larger objects. It is possible that a larger neighbourhood radius may have improved the model performance, however, this could also have reduced the performance on smaller objects such as people.

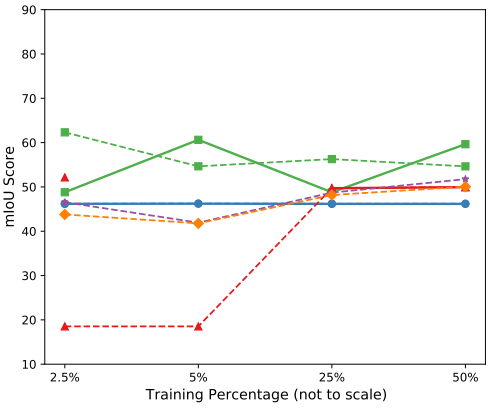
7.8.1 Pretrained vs Randomly Initialised Weights

With pretraining KPConv sometimes improves and sometimes worsens the results on the 50% holdout set, with both test and holdout sets having worse performance with the pretrained models at 25% training data.

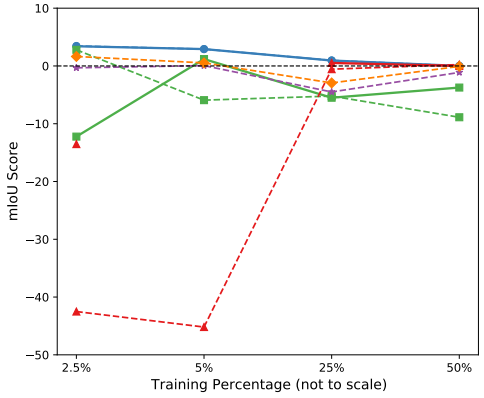
The randomly initialised PointTransformer with 2.5% training data spends an unusually long time appearing not to learn, hovering around 50% training mIoU before suddenly increasing on the training set and slowly on the test set. It is possible that increasing the batch size may have sped up the training time. Conversely, the pretrained model begins learning from the start of the training.



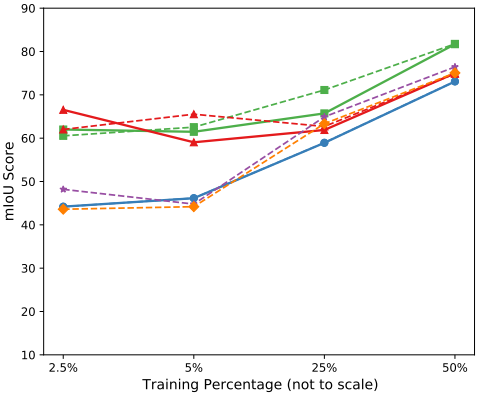
(a) Test mIoU



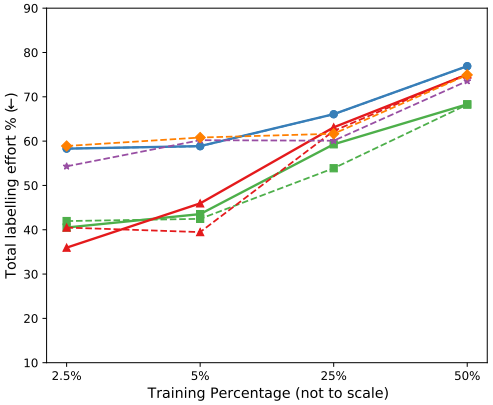
(b) 50% Holdout mIoU



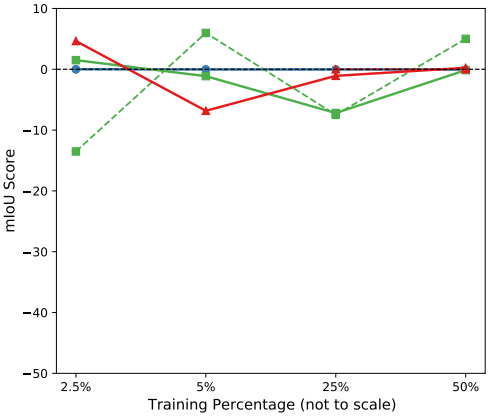
(c) Holdout Difference mIoU



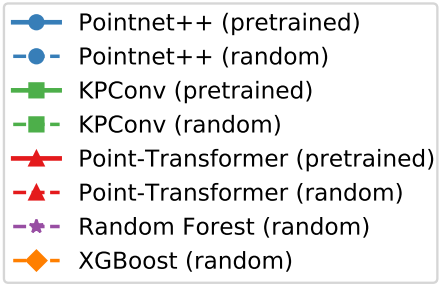
(d) Effective mIoU



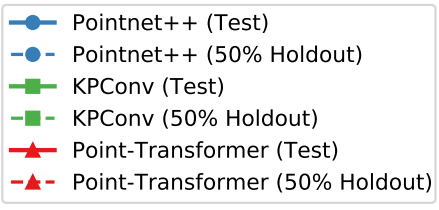
(e) Total Labelling Effort



(f) mIoU Improvement via Pretraining

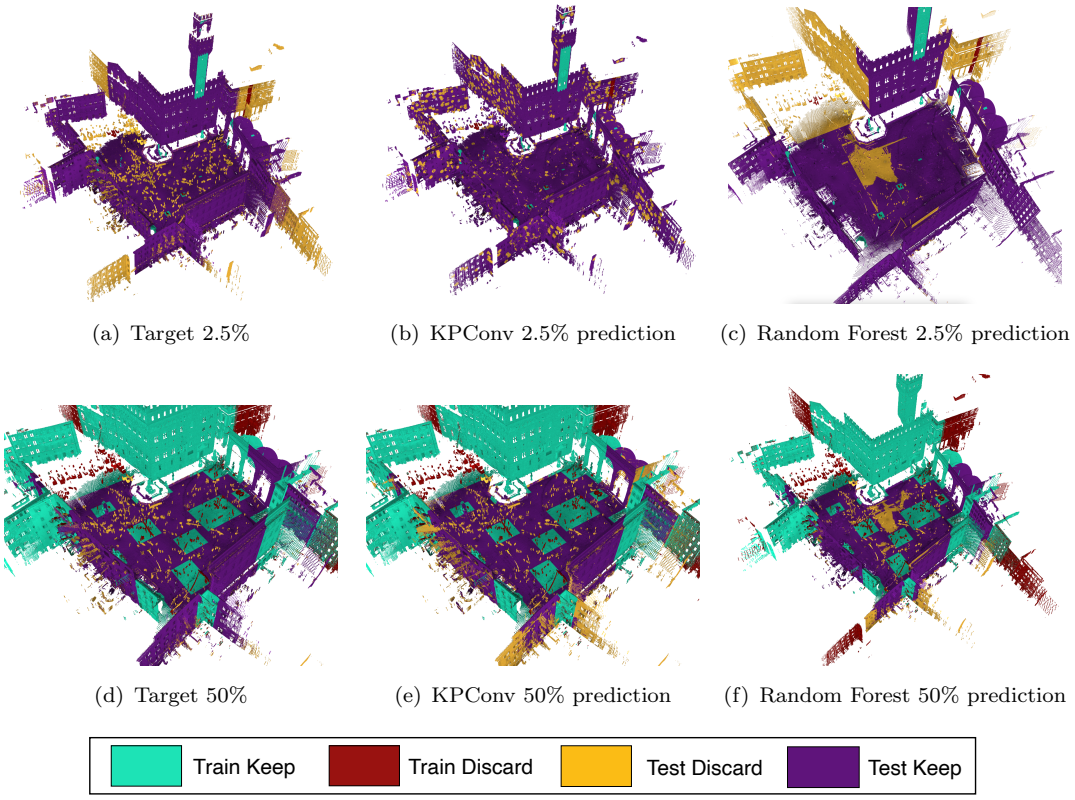


(g) Legend for the first 5 graphs



(h) Legend only for Improvement via Pretraining

FIGURE 7.15: Piazza: Line graphs comparing the mIoU% of the models at different training percentages



(a) Target 2.5% (b) KPConv 2.5% prediction (c) Random Forest 2.5% prediction

(d) Target 50% (e) KPConv 50% prediction (f) Random Forest 50% prediction

FIGURE 7.16: Piazza: Comparison of labelling performance for KPConv and Random Forest models as training data is increased.

7.8.2 Improvement via Training Percentage

For the Random Forest models, training percentages below 25% do not accurately classify the unseen portions of the scene. For the deep learning models, the mIoU does not change by more than a few percentage points as training data is increased. The exceptions to this are the 5% KPConv pretrained model performance changing (both decreasing and increasing), and the Point Transformer discussion below.

This may appear insignificant, however, Figure 7.16 provides a qualitative comparison of the improvements in KPConv and Random Forest performance as the data is increased. The counterintuitive drop in Point Transformer performance as training data is increased further is due to the models falling into a local minimum and predicting only the majority (keep) class. It is unclear what the cause of this may be, given that the models do this on both the training and test data.

7.8.3 Testing vs 50% Holdout

The 50% holdout is typically within 5% mIoU of the test set, likely due to the larger size of the dataset making it easier to get a representative subset of the scene for training

Model	Overall Accuracy %	Time (minutes)
Marais et al. [67] RF post-processed	**97.9**	298
PointNet++ (2.5% / 5% random init.)	85.5 / 86.5	1242 / 2130
KPConv (2.5% / 5% pretrained)	95 / 95.7	207 / 229
Point Transformer	97.8 / 97.36	67
(2.5% random init. / 5% pretrained)		
RF (ours) (2.5% / 5%)	84.8 / 76.5	**1.2** / 1.25
XGBoost (2.5% / 5%)	78.2 / 76.7	1.26 / 1.51

TABLE 7.4: Best Overall Accuracy and processing time on Piazza di Signoria for each of our models (less than 5% training labels) andMarais et al. [67]. Best accuracy and time in **bold**.

without labelling as large a proportion as required by other models.

7.8.4 Effective mIoU

In terms of total labelling effort, the quantitative results show that the Point Transformer and KPConv are the most label-efficient models. Both 2.5% and 5% training sets require 43±3% of the points to be labelled, although the randomly initialised models are slightly more efficient at the 5% training compared to the pretrained models.

7.8.5 Comparison to Previous Works

Marais et al. [67] achieved an overall accuracy of 97.9%, whilst the Point Transformer trained on 2.5% of the training data is close behind with 97.8% accuracy, coinciding with it having the highest mIoU on the test set as mentioned above. As with the Church scene, the majority of points are labelled as "keep" making accuracy a difficult metric to evaluate.

Whilst the Random Forest model is the fastest and appears to produce somewhat acceptable results based on the accuracies above, Figure 7.15(e) shows that KPConv is significantly more efficient for the labeller, requiring 20% fewer points to be labelled overall. This is supported by the qualitative results in Figure 7.16.

7.9 Trends Between models

In this section, we discuss several of the trends in the performance of the models. We present these using a similar set of key topics as in the individual experiments above. In general, we observe that the Random Forest and PointNet++ models achieve higher mIoU scores on simple scenes, while KPConv and Point Transformer perform better

on more complex scenes. Our aim of 90% mIoU was only achieved on Bagni Nerone and Montelupo, and Lunnahoja on the test set. However, we were consistently able to achieve above 80% on the test set with just 25% of the scene labelled for all but the Piazza scene.

7.9.1 Pretrained vs Randomly Initialised Weights

From the perspective of maximising mIoU over an unseen test set, pretraining is on average beneficial to model training. However, for the purposes of reducing the total labelling effort needed to clean these scenes, only Montelupo represents more than a five percent difference in performance.

In some cases, pretraining unexpectedly reduced the performance of the models on the test set, typically on scenes which are less complex with labelling decisions based on position in the scene rather than the semantics of the object.

PointNet++ typically benefits from pretraining with only Montelupo 2.5% and Church 2.5/5% performing better when trained from scratch.

KPConv is more variable, particularly over the 50% holdout set when training with a small amount of data. On the test set the model is generally less affected by pretraining at lower training percentages with most of the change coming at higher percentage training splits. This is the opposite finding of the other two deep learning models.

The Point Transformer model was the most affected by pretraining, especially in smaller training runs. Except for Lunnahoja, and Piazza on the *test set*, as well as Bagni di Nerone on the *holdout set*, pretraining improved the performance of the Point Transformer model. On average, pretraining improved performance by 10% on the test set, and 15% on the 50% holdout set. This supports our theory that the Transformer model requires more training data compared to others.

7.9.2 Testing vs 50% Holdout

Holdout differences were most prevalent in the Point Transformer model, possibly due to a susceptibility to initial conditions and hyperparameters, leading to unstable results. Randomly initialized models appear to be more affected, except for the Bagni Nerone dataset. This is consistent with the expectation that pretraining should reduce the amount of training data needed to train the model for a new scene, thus making it more resilient to holdout differences.

Whilst on the test set we achieve 90% mIoU with KPConv and Random Forests on both Bagni Nerone; and Lunnahoja and Montelupo respectively. Point Transformer also achieves this on Montelupo although the performance unexpectedly decreases as more training data is added. Whereas on the 50% holdout set, the results worsen with only KPConv achieving 90% mIoU on only Bagni Nerone whilst the Random Forests maintain both scenes above 90%.

7.9.3 Effective mIoU

Unsurprisingly, pre-labelling 50% of the scene results in the least post-prediction corrections, with an average effective mIoU of 90.3%. Our analysis shows that using 2.5-5% training labels is the optimal proportion of training labels needed to minimize total labelling effort after correcting the resulting predictions.

If time is to be optimised, the random forest is an order of magnitude faster to train and predict labels for the remainder of the scene than the deep learning models. Provided the scene is relatively simple such as Bagni di Nerone, Montelupo, or Monument, the additional 5-10% of labels that must be corrected is more efficient than waiting on a deep learning model. This is particularly impactful if there is ever a need to do the cleaning on-site where the user is unlikely to have access to a powerful GPU-equipped desktop machine. The Random Forest models can easily be run on a consumer laptop without a GPU.

If the primary goal is to reduce the labelling effort, the latency cost for using deep learning models becomes less important. Among the models studied, the randomly initialized KPConv model consistently requires the least total labelling effort (pretraining and post-prediction label correction combined). Although the Point Transformer model occasionally performs better in certain instances, its performance and pretraining improvement are inconsistent, making it unreliable for real-world use cases.

7.9.4 Random Forests and XGBoost

With few exceptions, we show XGBoost to be worse than the Random Forest classifier. Whilst XGBoost is typically considered the “default” baseline for tabular datasets, it often requires more tuning than the simpler Random Forest model. It is possible that with more tuning and additional precomputed features as in Marais et al. [67] the XGBoost model would be able to pull ahead as it is known to perform better on larger datasets with many features.

As a scene's segmentation boundaries may not necessarily be axis-aligned (see Section 3.2), the Random Forest and XGBoost models struggle on more complex scenes. A strong potential for future work lies in exploring additional features as inMarais et al. [67] and alternative methods for splitting the trees. Additional features would not significantly increase end-to-end prediction times, with the training complexity of both models increasing linearly and inference complexity remaining unaffected. Alternative splitting methods which allow for non-aligned boundaries could improve predictive power at the cost of additional training complexity. As such both methods should be explored both independently and jointly.

7.9.5 Training dynamics of KPConv and Point Transformer

KPConv has a very fast initial spike in performance but can take a long time to reach the maximum mIoU for an experiment, whereas the Point Transformer often trained for less time, but was more variable in its performance. This likely means that despite the hyperparameter tuning performed on the Church set (in which the Point Transformer achieved the best performance on the unseen set and needed the least total labelling effort), the Point Transformer is still sensitive to the initial conditions. Lowering the learning rate further and increasing training times (similar to KPConv) could smooth out the learning. This is another potential avenue for future work.

7.10 Summary

In this chapter, we discussed the results of our main and final set of experiments across each of our models and scenes. We presented our analysis and suggested potential causes of any inconsistencies. In the following chapter, we summarise the conclusions from the study and recommend potential avenues for future work.

Chapter 8

Conclusion

Point cloud cleaning is a manual and time intensive process to remove scanner errors and objects which are not wanted in the final model. Several methods exist for assisting with parts of the cleaning process via automatic plane detection or foliage removal for example. However, these tools are designed primarily for urban environments. In a Cultural Heritage context, varied cleaning requirements and irregular surfaces often make these tools less effective. Previous approaches for cleaning heritage point clouds relied on hand crafted features and focused on cleaning single scans at a time, rather than the fully registered point cloud.

In this thesis, we aimed to evaluate the effectiveness of deep learning models for cleaning these registered point clouds. Using five laser scanned scenes, we compared three deep learning models, PointNet++, KPConv, and Point Transformer, against a baseline Random Forest model. These models represent popular backbones for point cloud segmentation, with KPConv and Point Transformer performing near the top of the S3DIS benchmark.

The scenes cover a diverse range of cleaning requirements and scene complexities. A mix of both indoor and outdoor regions, including scenes with foliage and urban scenes with many people to remove. As the label ratio may be heavily imbalanced, we chose mIoU as our main evaluation metric rather than accuracy which may be an optimistic estimate of performance.

The primary aim was to reduce the total labelling effort required to clean the point clouds, whilst considering factors such as speed, pretraining, and the amount of hand labelling required. We set out to answer several questions:

- Can modern deep learning models be used to clean our scenes with greater than 90% mIoU?

- Is the relative performance of the deep learning models maintained from the S3DIS benchmark?
- To what extent does pretraining on S3DIS affect overall model performance?
- Can our models reduce the total labelling effort by more than 50%?
- Can Active Learning reduce the manual labelling needed to train our models?
- Does changing to a registered point cloud directly improve the performance of a Random Forest classifier compared to previous work by Marais et al. [67]

Taken together we can summarise the answers to these questions by examining the raw performance of the models as well as the total labelling effort needed when used as a cleaning assistant.

8.0.1 Raw Performance

We approached the raw model performance from tow sides: the test set (all remaining data to be labelled), and a 50% holdout set which is shared across four training splits.

On the test set, Bagni Nerone can be segmented above 90% mIoU by both KPConv and the Random Forest, on Lunnahoja this is only achieved by the KPConv model, and on Montelupo both Point Transformer and Random Forest are able to reach 90% mIoU. Looking at the 50% holdout set, Bagni Nerone is largely unchanged, Lunnahoja is no longer achieved by KPConv, and on Montelupo only the Random Forest reaches 90%.

With respect to the performance on the 50% holdout set, we show pretraining is generally beneficial for maximising the mIoU. The pretraining benefit is largest when only a small amount of training data is used for finetuning the model. PointNet++ is minimally affected by pretraining here although this is often due to the model failing to meaningfully learn. In contrast, KPConv is only intermittently improved, whilst Point Transformer benefits the most, improving by an average of 15%. This suggests the point transformer required more data to train effectively than the other models. The effects of pretraining are less evident on the test sets (and thus the total labelling effort below). Due to the nature of the heritage scenes, there are often large “easy to clean” areas which are less dominant in the 50% holdout set. We also observed a pattern of reduced performance when the labelling criteria are simpler and based largely on position rather than the type of object. Random Forest models are most effective at labelling these scenes where learning abstract features is less important.

8.0.2 Total Labelling Effort

The "Total Labelling Effort" approximates the usefulness of the model in the real world. Despite the Point Transformer most often achieving the lowest labelling effort, it is highly inconsistent in both the amount of training data needed and the effect pretraining will have. In contrast, the randomly initialised KPConv model trained on 5% of the scene produces consistently good results, a total labelling effort of at most 43%, with an average of just $25 \pm 11\%$. The pretrained model is within 5% of the performance with neither consistently outperforming the other.

The tradeoff for this consistency is a need for longer training times. KPConv took on average three to four hours to train compared to only one hour for the Point Transformer. Extending this further, the Random Forest Models are able to train and classify the scenes in under two minutes.

Unfortunately, the Random Forest models prove to be less effective on more complex scenes owing to their inability to learn complex geometry or decision boundaries which are not axis-aligned. On both Bagni Nerone and Montelupo, where much of the labelling is based on position in the scene, the labelling effort is within 3% and 11% respectively of the best-performing model on those datasets. However, despite the simple labelling required, the Lunnahoja scene cannot be segmented as the labelling boundary is diagonal to the XY axes.

We also show that simply operating on registered point clouds in a single pass is insufficient to improve the performance compared to the results found by Marais et al. [67]'s iterative approach. Our Random Forest models underperform their accuracy results (note, not mIoU) by 3.5% on simpler scenes and 10 to 23% on the more complex scenes. Although notably, our use of Random Forest classifiers is at least an order of magnitude faster.

With respect to an iterative cleaning process, we find that incorporating active learning did not improve the performance of either the PointNet++ or Random Forest models in our preliminary study. We speculate that the poor results may be caused by the difficulty of automatically selecting subsequent query regions given the relatively small number of potential samples in our scenes.

Our results demonstrate the potential of modern machine learning methods for label and time-efficient cleaning of cultural heritage sites. The KPConv model is the most effective labelling assistant for cultural heritage sites given its applicability to a variety of scenes and relatively short time requirements on consumer hardware. On the other

hand, the Random Forest, though taking less than two minutes to complete the scenes, requires 20% more labelling than KPConv to correct the predictions.

8.1 Limitations and Future Work

We identified the following as the main limitations of our study:

1. With only a small selection of scenes, the conclusions drawn on this study may be more specific to our dataset than is apparent without further data.
2. PointNet++'s use of a columnar sampling strategy rather than the spherical sampling used by KPConv and Point Transformer may have negatively impacted the results. Particularly in the Piazza where there are regions which are far taller than the rest of the scenes.
3. The results from Point Transformer were not qualitatively examined. In certain scenes, the qualitative performance may differ from the reported mIoU, such as when the ground truth contains inconsistent or erroneous labellings.

The most promising avenue with real world practicality involves the Random Forest model. With the speed at which the Random Forest models can be trained, it is feasible to create a real time interactive labelling assistant. A user study could examine the real world speedup in the end to end cleaning time.

The study could explore adding additional hand-crafted features or using a more expressive splitting function such as oblique decision trees. These additions may allow the random forest to capture more complex scene geometries. Alternatively, rather than hand crafted features a pretrained deep learning model such as KPConv or Point Transformer could be used as a feature extractor. This could provide more expressive features than could be easily hand-crafted.

Appendix A

Hardware, Software and Raw Results

A.1 Source Code and Results

The source code for the framework containing all the designed and implemented methods is publicly available at: https://github.com/LucHayward/Masters-Hub.

The full tables of results can be accessed in the same repository. We provide the raw mIoU tables as well as several additional views into the data that we felt did not meaningfully further the main discussion of the results.

A.2 Computing Facilities

During the initial development and exploratory testing, all models were trained on an Intel i5-4590, GTX1070 and 16GB DDR3 RAM; with the exception of the Point-Transformer which was trained on an AMD-5900X, RTX3080 and 32GB DDR4 RAM.

Our thanks to the University of Cape Town's ICTS High Performance Computing team hpc.uct.ac.za for providing GPU computing for much of our final experiments. They provided access to the Nividia A100 shared cluster and exclusive access to a 16-GPU P100 cluster significantly shortening the time needed to conduct our research.

Appendix B

Active Learning

B.1 Active Learning model details

B.1.1 Random Forest

We make use of the Random Forest implementation in the Scikit-Learn library in Python. We adjust the following hyperparameters:

- **max_depth** = 32
- **min_samples_split** = 20
- **n_estimators** = 32

B.1.2 Pointnet++

Two pointnet++ models were tested:

Model	Training Epochs	Dropout Repeats	Training Data Increase
1	5	5	5%
2	12	10	5%

TABLE B.1: Details of the Two PointNet++ Models Tested

Appendix C

Human Interpretable Features

Name	S / M	Definition	Notes
Height	S	$\mathbf{p}_z$	The z value returned by the scanner in the scanner coordinate frame
Distance	S	$\lvert\mathbf{p}\rvert$	Distance from scanner (depth)
Curvature 1	S	κ_1	Principal curvatures at $\mathbf{p}$
Curvature 2	S	κ_2	
Cylinder 1	M	$Z_{\max} - Z_{\min}$	Cylindrical neighbourhood features
Cylinder 2	M	$\mathbf{p}_z - Z_{\min}$	
Cylinder 3	M	$Z_{\max} - \mathbf{p}_z$	
Anisotropy	M	$\frac{\lambda_1 - \lambda_3}{\lambda_1}$	Features computed from eigenvalues/vectors of the structure tensor
Planarity	M	$\frac{\lambda_2 - \lambda_3}{\lambda_1}$	
Sphericity	M	$\frac{\lambda_3}{\lambda_1}$	
Omnivariance	M	$\sqrt[3]{\lambda_1 \lambda_2 \lambda_3}$	
Linearity	M	$\frac{\lambda_1 - \lambda_2}{\lambda_1}$	
Surface variation	M	$\frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}$	
Eigen entropy	M	$-\sum_{i=1}^{3} \lambda_i \log \lambda_i$	
Verticality	M	$1 - (0, 0, 1) \bullet \mathbf{e}_3$	
Density	M	$(k+1)/\frac{4}{3}\pi r_k^3$	for k nearest neighbours

FIGURE C.1: Human interpretable features used in previous study by Marais et al.

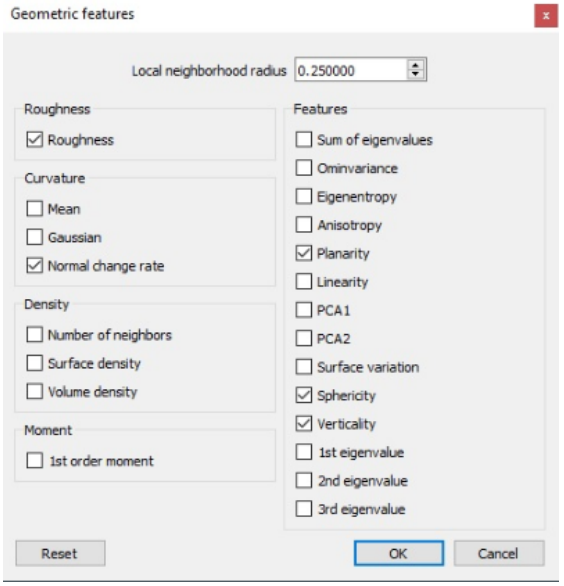


FIGURE C.2: Human interpretable features which can be generated via cloudcompare

Appendix D

Deep Learning Model Hyperparameters

The hyperparameters used in the final experiments for each of the three deep learning models are as follows. If a hyperparameter is not listed then it is kept as the default value, and not all hyperparameters are necessarily used by the final model code:

D.1 Pointnet++

Table D.1: Pointnet++ Hyperparameters

Hyperparameter	**Value**
Model	pointnet2_sem_seg
Epoch	60
Optimizer	Adam
Decay Rate	0.01
Step Size	10
Learning Rate Decay	0.7
Batch Size	16
Learning Rate	0.001
Npoint	4096
Block Size	1.0
Augment Points	False
Force Even	False
Sample All Validation	True
Relative Point Coords	False

D.2 KPConv

Table D.2: KPConv++ Hyperparameters

Parameter	Value
num_kernel_points	15
in_radius	1.2
first_subsampling_dl	0.02
conv_radius	2.5
deform_radius	5.0
KP_extent	1.2
KP_influence	'linear'
aggregation_mode	'sum'
first_features_dim	128
in_features_dim	1
modulated	False
use_batch_norm	True
batch_norm_momentum	0.02
deform_fitting_mode	'point2point'
deform_fitting_power	1.0
deform_lr_factor	0.1
repulse_extent	1.2
max_epoch	100
learning_rate	1e-2
momentum	0.98
lr_decays	{i: 0.1 ** (1 / 150) for i in range(1, max_epoch)}
grad_clip_norm	100.0
batch_num	6
epoch_steps	500
validation_size	50
augment_scale_anisotropic	True
augment_symmetries	[True, False, False]
augment_rotation	'vertical'
augment_scale_min	0.9
augment_scale_max	1.1
augment_noise	0.001
segloss_balance	'none'

D.3 Point Transformer Sweep Configuration

We make use of the Weights and Biases Sweeps framework for choosing hyperparameters over the Church scene. The hyperparameters and range of values to be explored are:

Table D.3: Point Transformer Church Sweep Configuration

Parameter	**Possible Values**	**Description**
base_lr	0.3, 0.5, 0.7	Learning rate
freeze_body	true, false	Whether to freeze body or retrain the full network
power	0.3, 0.6, 0.9	Controls the rate of decrease of learning rate when using the warmup scheduler
scheduler	warmup, default	Scheduler type
voxel_max	5000, 10000, 20000, 30000, 40000	Max points per sample
voxel_size	0.04, 0.02	Voxel size
warmup_length	10, 25, 50	Warmup length
weight_decay	0.01, 0.001, 0.0001	Weight decay

D.4 Point Transformer Hyperparameters

Note that we use slightly different hyperparemeters for different amounts of training data. 2.5% and 5% are as below. For 25% training, we set weight_decay=0.001, base_lr=0.3 and batch_size=5 and no warmup on the learning rate scheduler. For 50% training, we set weight_decay=0.0001 and base_lr=0.5.

Table D.4: Point Transformer Hyperparameters

Parameter	**Value**
arch	pointtransformer_seg_repro
loop	30
rank	0
power	0.3
epochs	100
base_lr	0.01
classes	2
fea_dim	3
use_xyz	true
momentum	0.9
drop_rate	0.5
scheduler	warmup
voxel_max	20000
batch_size	7
multiplier	0.1
step_epoch	30
voxel_size	0.02
freeze_body	false
manual_seed	7777
weight_decay	0.02
warmup_length	25
batch_size_val	1

Appendix E

Training Splits

Training splits for the models at 2.5, 5, 25 and 50%. Light blue and pink are the prediction targets; dark blue and pink are the training points.

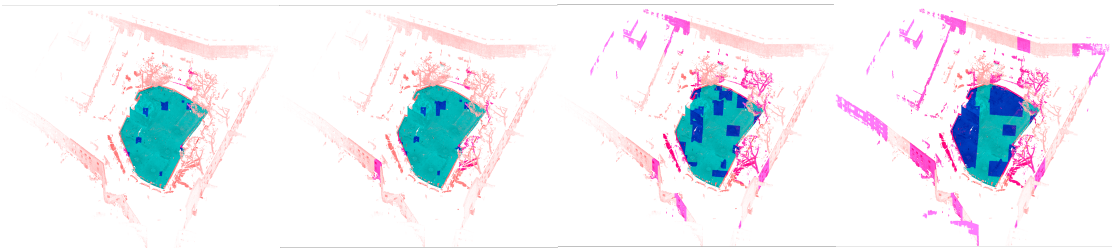


FIGURE E.1: Bagni di Nerone: Training splits

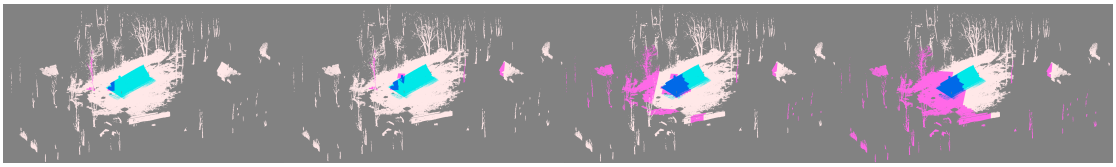


FIGURE E.2: Lunnahoja: Training splits

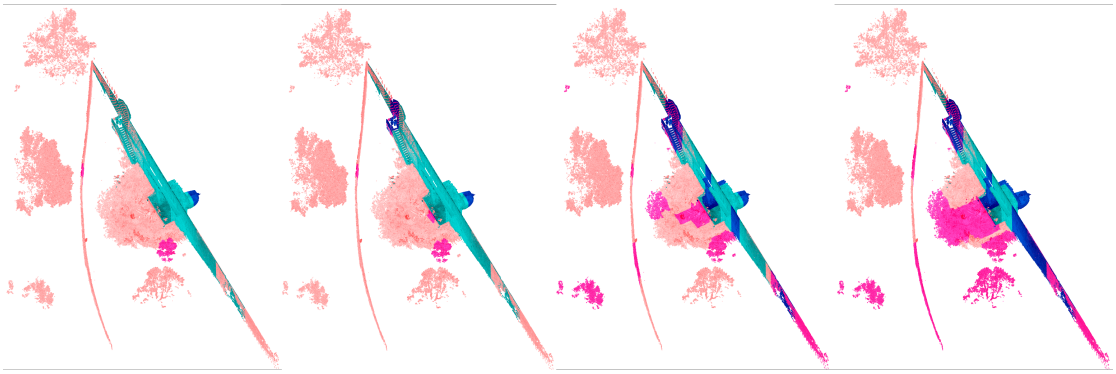


FIGURE E.3: Montelupo: Training splits

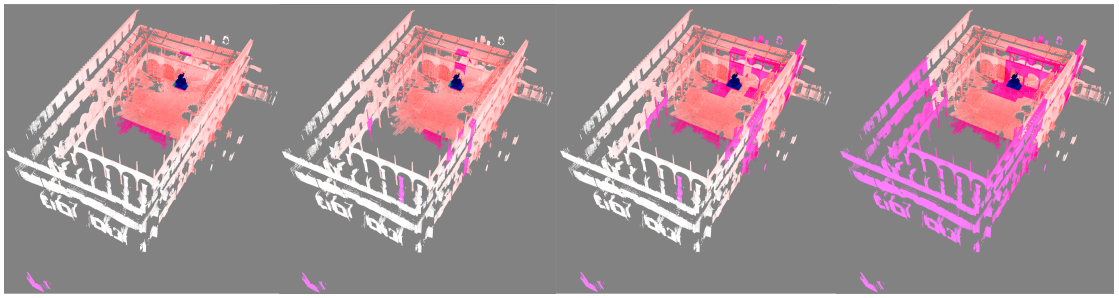


FIGURE E.4: Monument: Training splits

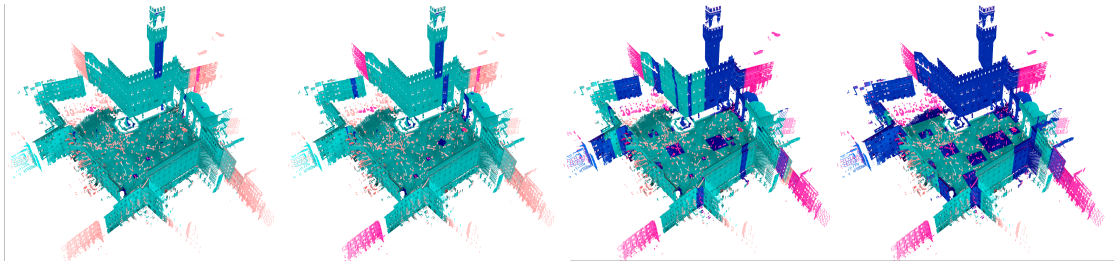


FIGURE E.5: Piazza: Training splits

Appendix F

Additional Dataset Visualisations

In the following sections, we provide additional visualisations of the datasets which did not fit into any particular chapter within the main thesis body. Nevertheless, they may provide additional context to a reader.

F.1 Bagni di Nerone

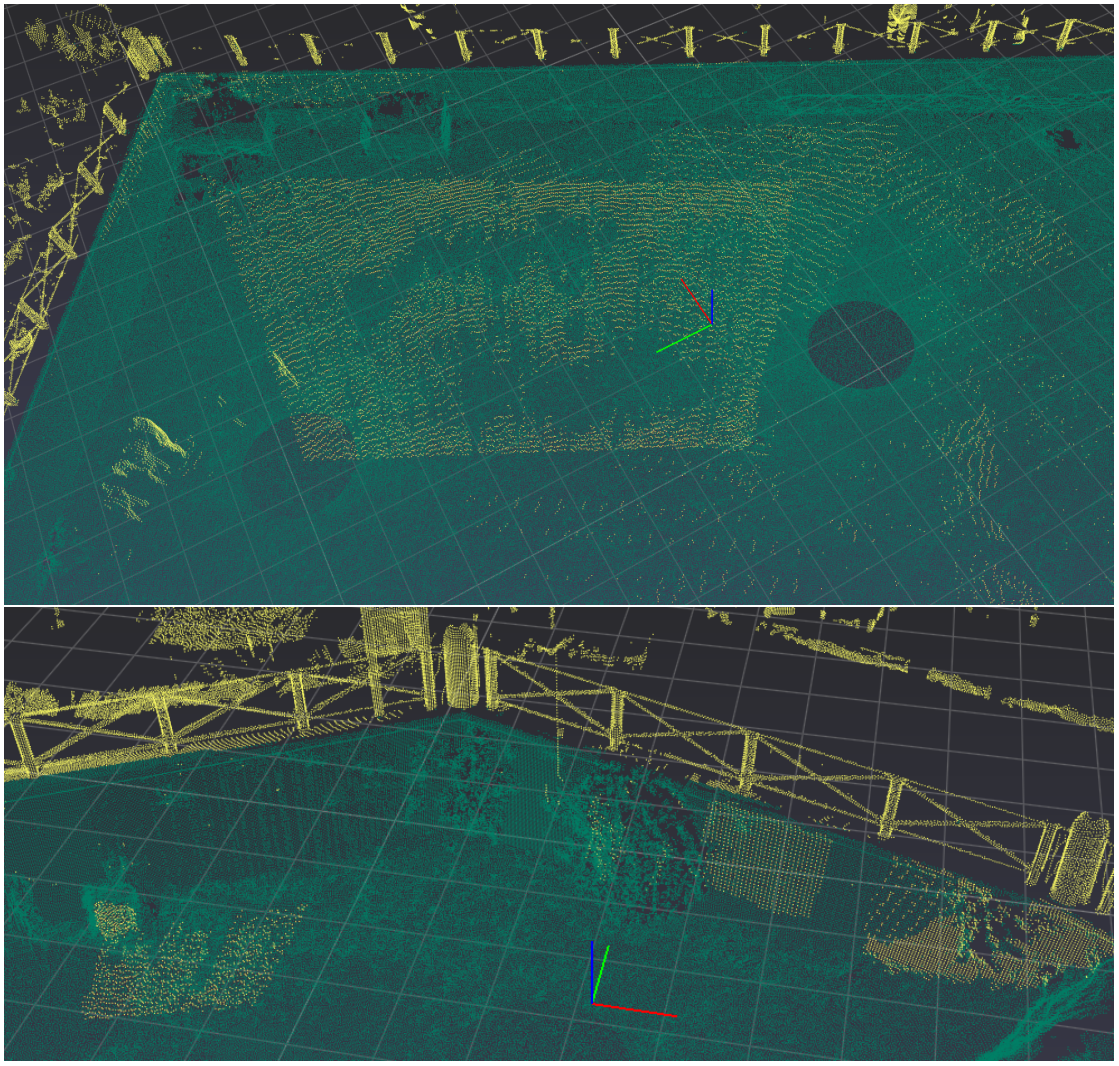


FIGURE F.1: Bagni Nerone: Two examples of likely incorrect labellings in the ground truth.

F.2 Church

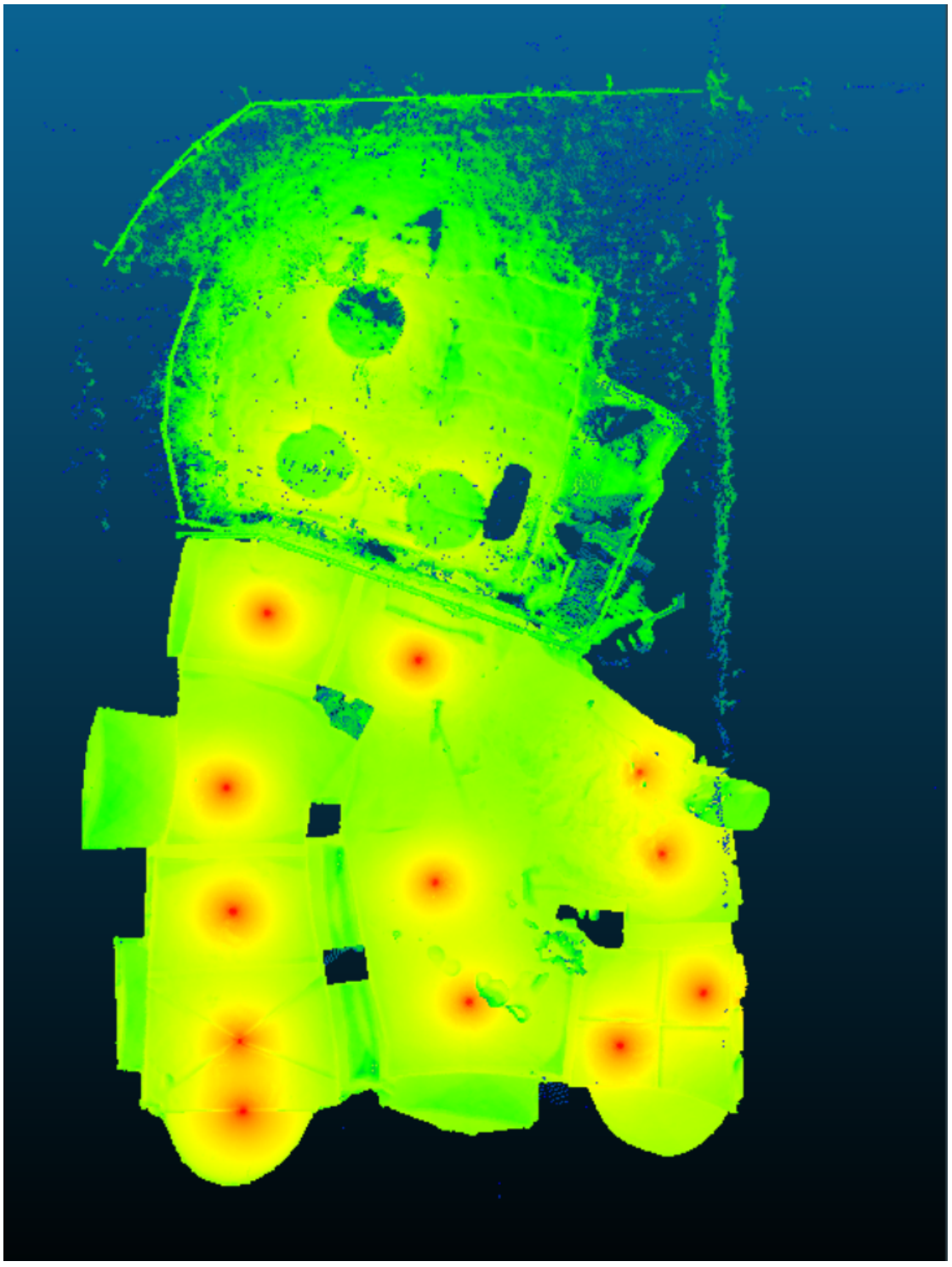


FIGURE F.2: Church: False coloured by density (blue to red), illustrating the high density of points immediately above the scanner positions. Incidentally also reveals the scanner positions inside.

F.3 Lunnahoja

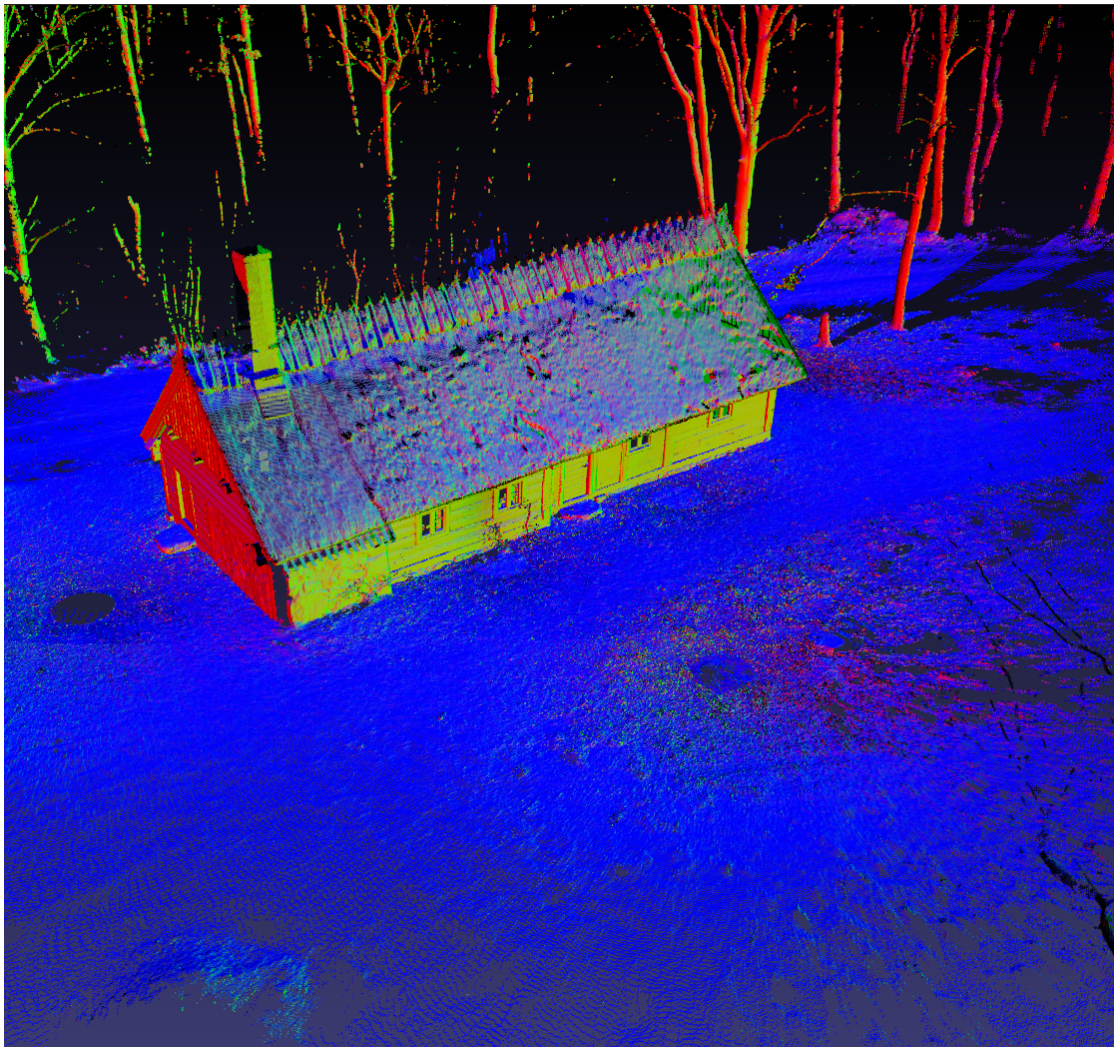
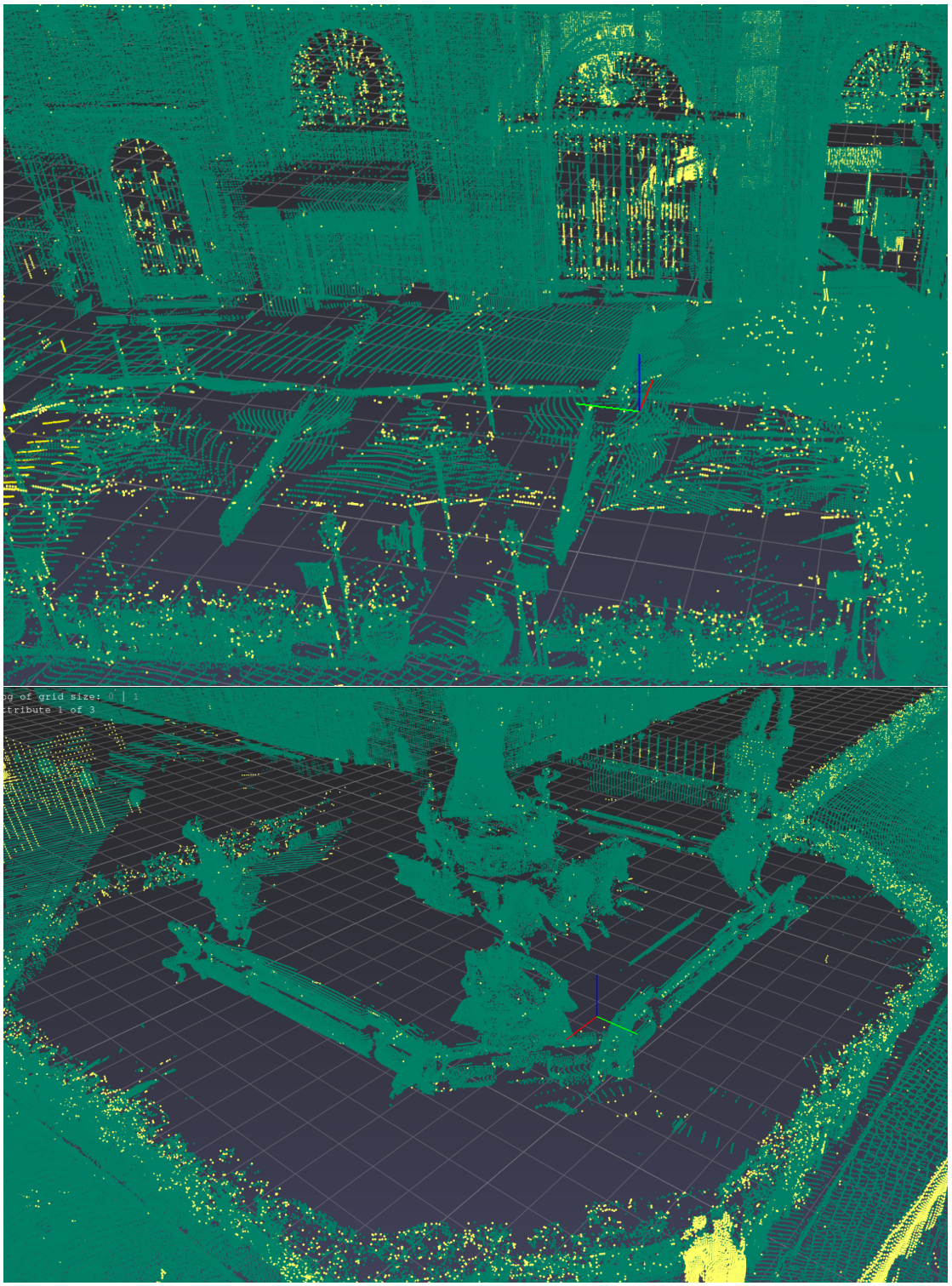


FIGURE F.3: Lunnahoja: Normal data for each point was also included with this scene, although we did not make use of it in our models.

F.4 Piazza



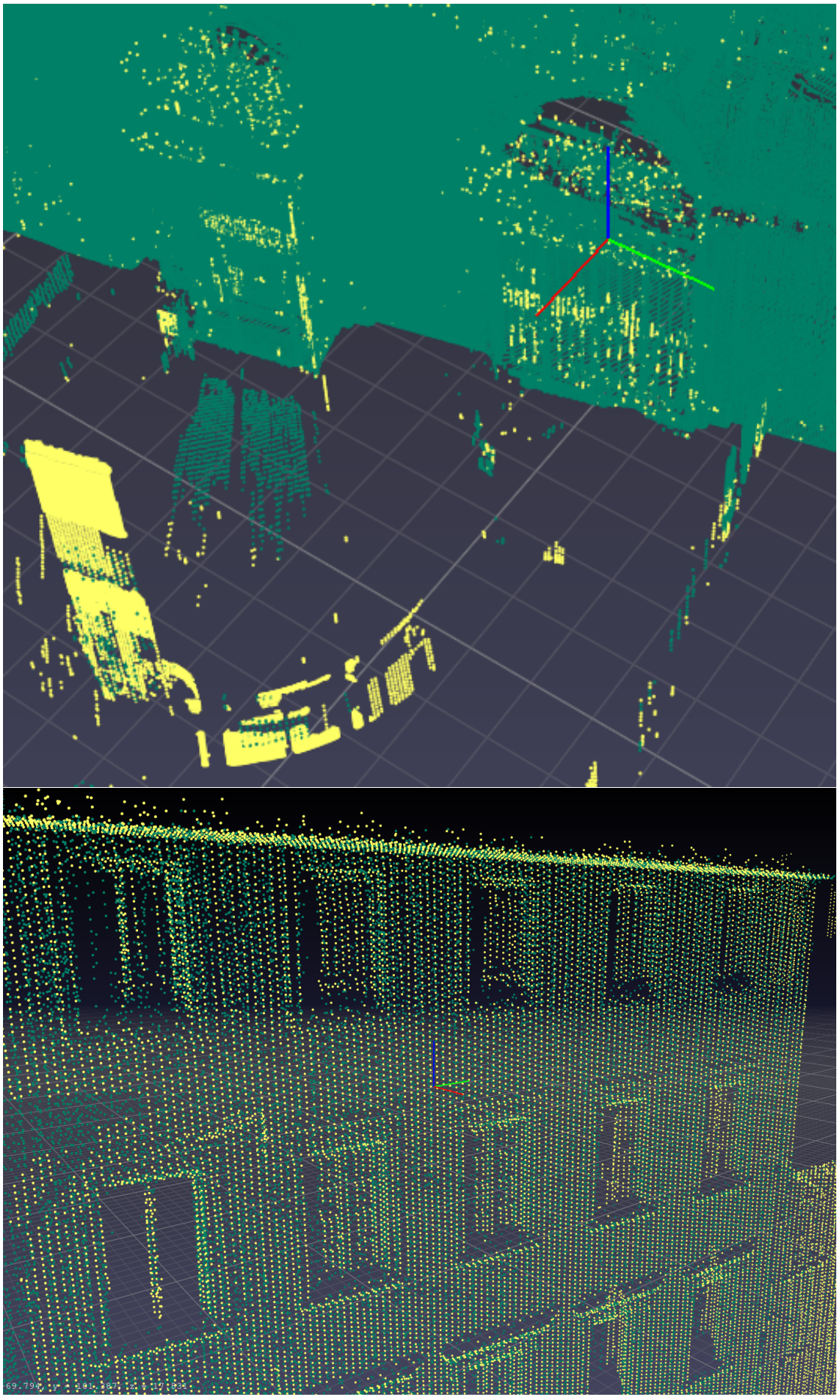


FIGURE F.3: The Piazza scene has many examples of noisy labellings due to its size and variety of objects in the scene.

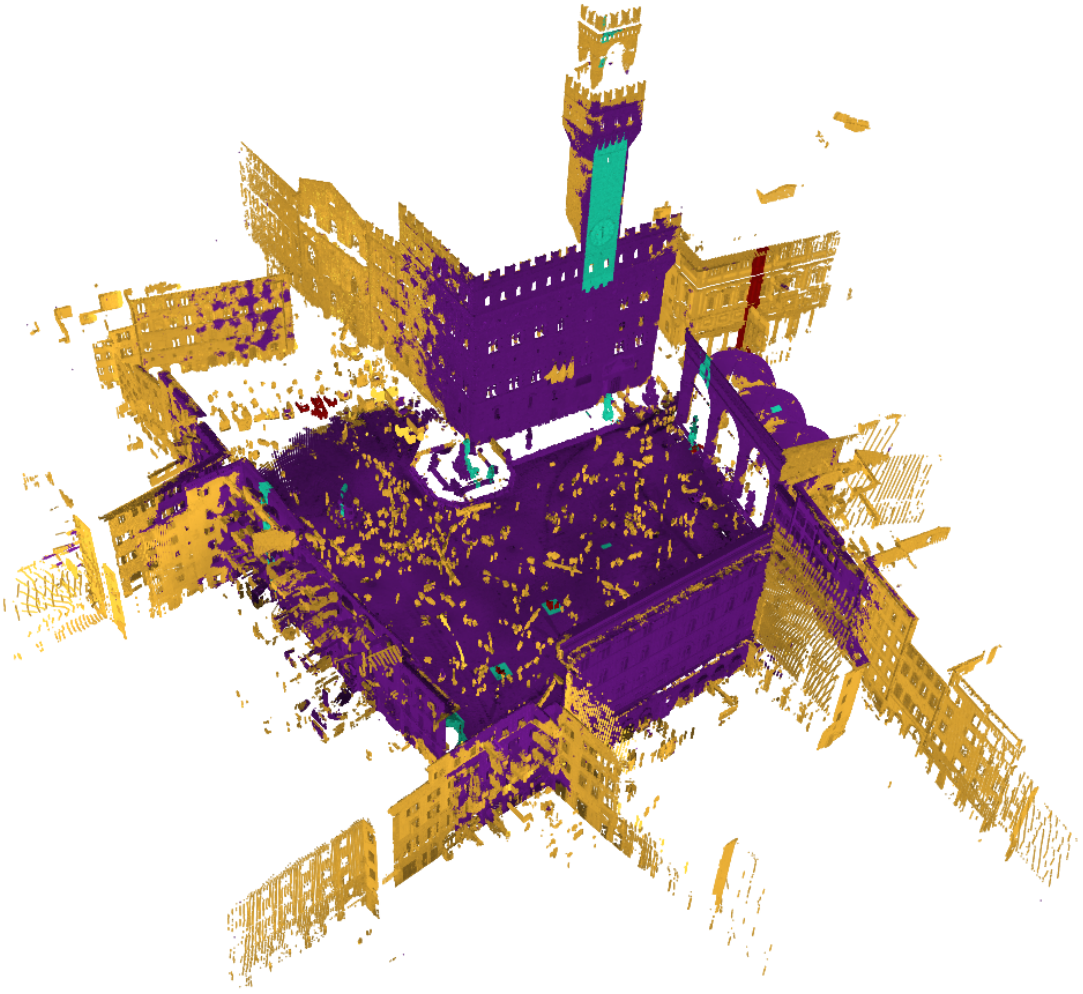
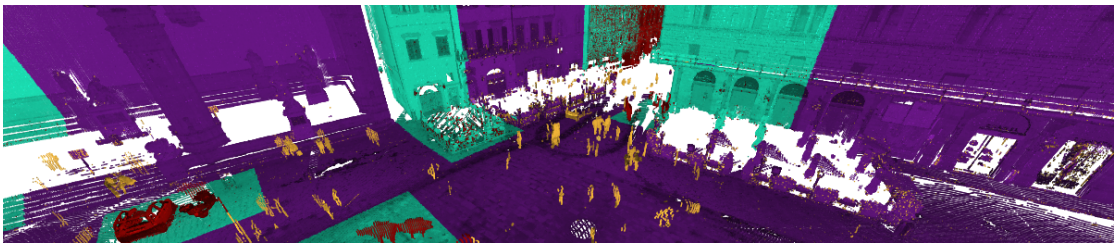
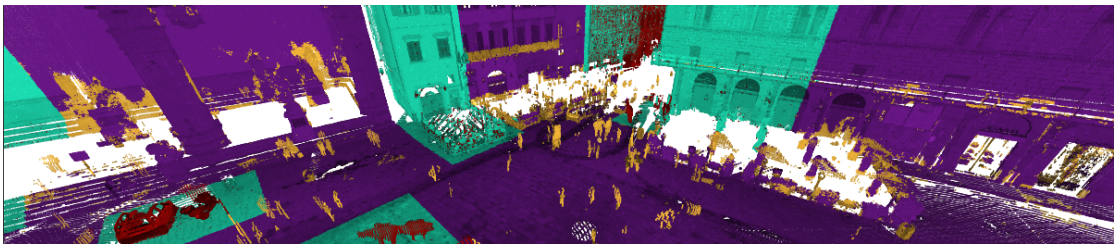


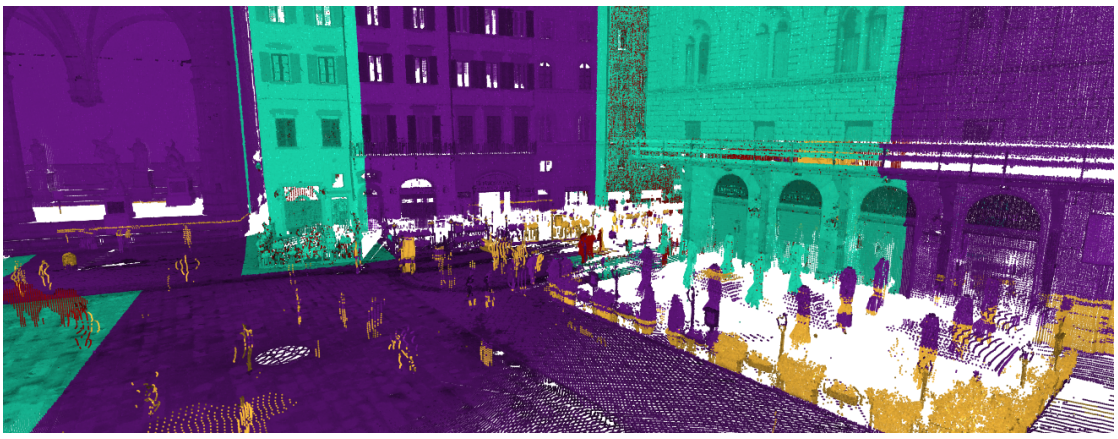
FIGURE F.4: KPConv trained on 2.5% begins by randomly labelling each sample sphere



(a) Target



(b) KPConv



(c) Random Forest

FIGURE F.5: Piazza: An example of regions where the target labellings are noisy and potentially ambiguous, predictions from the 50%

Bibliography

[1] Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., Fieguth, P., Cao, X., Khosravi, A., Acharya, U. R., Makarenkov, V., and Nahavandi, S. (2021). A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297.

[2] Abe, N. and Mamitsuka, H. (1998). Query learning strategies using boosting and bagging. In *Proceedings of the 25th international conference on Machine learning*, volume 388, pages 1–9.

[3] Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the Surprising Behavior of Distance Metrics in High Dimensional Space. In den Bussche, J. and Vianu, V., editors, *Database Theory — ICDT 2001*, pages 420–434, Berlin, Heidelberg. Springer Berlin Heidelberg.

[4] Armeni, I., Sener, O., Zamir, A. R., Jiang, H., Brilakis, I., Fischer, M., and Savarese, S. (2016). 3D Semantic Parsing of Large-Scale Indoor Spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*.

[5] Arthur, D. and Vassilvitskii, S. (2007). k-means++: the advantages of careful seeding. In *ACM-SIAM Symposium on Discrete Algorithms*.

[6] Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., and Gall, J. (2019). SemanticKITTI: A dataset for semantic scene understanding of LiDAR sequences. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob(iii):9296–9306.

[7] Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. (2010). A theory of learning from different domains. *Machine learning*, 79:151–175.

[8] Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36. JMLR Workshop and Conference Proceedings.

[9] Bernardini, F. and Rushmeier, H. (2002). The 3D model acquisition pipeline. In *Computer Graphics Forum*, volume 21, pages 149–172. Wiley Online Library.

[10] Bertoin, D., Bolte, J., Gerchinovitz, S., and Pauwels, E. (2021). Numerical influence of ReLU'(0) on backpropagation. *Advances in Neural Information Processing Systems*, 34:468–479.

[11] Besl, P. J. and McKay, N. D. (1992). Method for registration of 3-D shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. Spie.

[12] Brain, D. and Webb, G. I. (1999). On the effect of data set size on bias and variance in classification learning. In *Proceedings of the Fourth Australian Knowledge Acquisition Workshop, University of New South Wales*, pages 117–128.

[13] Breiman, L. (1996). Bagging predictors. *Machine learning*, 24:123–140.

[14] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[15] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

[16] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Mach. Learn.*, 20(3):273–297.

[17] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

[18] Cracknell, A. P. and Hayes, L. B. (1991). *Introduction to remote sensing*. CRC press.

[19] CuddlyNest (2022). Top 6 Things to See In Piazza Della Signoria, Florence.

[20] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

[21] Dangeti, P. (2017). The Elbow Method. In *Statistics for machine Learning*. Packt Publishing.

[22] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

[23] Deng, S. and Dong, Q. (2021). GA-NET: Global attention network for point cloud semantic segmentation. *IEEE Signal Processing Letters*, 28:1300–1304.

[24] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*.

[25] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

[26] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press.

[27] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.

[28] Freund, Y. and Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 904(1):23–37.

[29] Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407.

[30] Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378.

[31] Fröhlich, C., Mettenleiter, M., and Others (2004). Terrestrial laser scanning–new perspectives in 3D surveying. *International archives of photogrammetry, remote sensing and spatial information sciences*, 36(8):W2.

[32] Gal, Y. and Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *33rd International Conference on Machine Learning, ICML 2016*, 3:1651–1660.

[33] Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, 4(1):1–58.

[34] Giangaldoni, P. (2007). *Chiese e Porte nella storia di Pisa*. C.L.D. Libri Srl.

[35] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[36] Grilli, E., Menna, F., and Remondino, F. (2017). A review of point clouds segmentation and classification algorithms. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42:339.

[37] Grilli, E., Özdemir, E., and Remondino, F. (2019). Application of machine and deep learning strategies for the classification of heritage point clouds. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 42(4/W18):447–454.

[38] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On Calibration of Modern Neural Networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR.

[39] Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.

[40] Hermosilla, P., Ritschel, T., Vázquez, P.-P., Vinacua, À., and Ropinski, T. (2018). Monte carlo convolution for learning on non-uniformly sampled point clouds. *ACM Transactions on Graphics (TOG)*, 37(6):1–12.

[41] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

[42] Hochreiter, S. and Schimdhuber, J. (1997). Long Short-Term Memory. *Nerual Computation*, 9(8):1735–1780.

[43] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

[44] I, S. (2009). Bagni di nerone, pisa.

[45] Iijima, Y. (1965). The Laser and its Application to Navigation. *The Journal of the Nautical Society of Japan*, 32(0):1–6.

[Inside Inferno] Inside Inferno. Piazza della Signoria. Accessed: 2023-04-19.

[47] Ittner, A. and Schlosser, M. (1996). Non-Linear Decision Trees - Ndt. *Icml*, pages 252–257.

[48] Jain, S., Munukutla, S., and Held, D. (2019). Few-Shot Point Cloud Region Annotation with Human in the Loop. Technical report.

[49] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

[50] Kingma, D. P. and Lei Ba, J. (2015). ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. In *International Conference on Learning Representations*.

[51] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classificaton with Deep Convolutional Neural Networks. *NeurIPS*.

[52] Lai, X., Liu, J., Jiang, L., Wang, L., Zhao, H., Liu, S., Qi, X., and Jia, J. (2022). Stratified Transformer for 3D Point Cloud Segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8500–8509.

[53] Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6405–6416, Red Hook, NY, USA. Curran Associates Inc.

[54] Landrieu, L. and Simonovsky, M. (2018). Large-Scale Point Cloud Semantic Segmentation with Superpoint Graphs. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 4558–4567.

[55] Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., and Beijbom, O. (2019). PointPillars: Fast Encoders for Object Detection From Point Clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12689–12697.

[56] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[57] Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. (2019). Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR.

[58] Lemay, A., Hoebel, K., Bridge, C. P., Befano, B., De Sanjosé, S., Egemen, D., Rodriguez, A. C., Schiffman, M., Campbell, J. P., and Kalpathy-Cramer, J. (2022). Improving the repeatability of deep learning models with Monte Carlo dropout. *npj Digital Medicine*, 5(1):174.

[59] Lewis, D. D. and Gale, W. A. (1994). A Sequential Algorithm for Training Text Classifiers. In Croft, B. W. and van Rijsbergen, C. J., editors, *SIGIR '94*, pages 3–12, London. Springer London.

[60] Lim, H. S. M. and Taeihagh, A. (2019). Algorithmic decision-making in AVs: Understanding ethical and technical concerns for smart cities. *Sustainability (Switzerland)*, 11(20):5791.

[61] Lin, Y., Vosselman, G., Cao, Y., and Yang, M. (2020). Efficient training of semantic point cloud segmentation via active learning. *ISPRS annals of the photogrammetry, remote sensing and spatial information sciences*, 2:243–250.

[62] Liu, X., Han, Z., Liu, Y.-S., and Zwicker, M. (2019). Point2sequence: Learning the shape representation of 3d point clouds with an attention-based sequence to sequence network. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 8778–8785.

[63] Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

[64] Loshchilov, I. and Hutter, F. (2017). Fixing Weight Decay Regularization in Adam. *CoRR*, abs/1711.0.

[65] Luo, W., Li, Y., Urtasun, R., and Zemel, R. (2016). Understanding the Effective Receptive Field in Deep Convolutional Neural Networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 4905–4913, Red Hook, NY, USA. Curran Associates Inc.

[66] MacQueen, J. (1967). Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297. University of California Los Angeles LA USA.

[67] Marais, P., Dellepiane, M., Cignoni, P., Scopigno, R., Dellepiane, . M., Cignoni, P., and Scopigno, R. (2019). Semi-automated Cleaning of Laser Scanning Campaigns with Machine Learning. *ACM Journal on Computing and Cultural Heritage*, 12(3):16.

[68] Matrone, F., Lingua, A., Pierdicca, R., Malinverni, E. S., Paolanti, M., Grilli, E., Remondino, F., Murtiyoso, A., and Landes, T. (2020). A benchmark for large-scale heritage point cloud semantic segmentation. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 43:1419–1426.

[69] Maturana, D. and Scherer, S. (2015). Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE.

[70] Mulder, R. (2012). Fast semi-automated point cloud cleaning (proposal).

[71] Mulder, R. (2017). *Accelerating Point Cloud Cleaning*. PhD thesis, University of Cape Town.

[72] Mulder, R. L. and Marais, P. (2016). Accelerating point cloud cleaning. In Catalano, C. E. and Luca, L. D., editors, *2016 Eurographics Workshop on Graphics and Cultural Heritage, GCH 2016*, pages 211–214. The Eurographics Association.

[73] Murthy, S. K., Kasif, S., and Salzberg, S. (1994). A System for Induction of Oblique Decision Trees. *Journal of Artificial Intelligence Research*, 2:1–32.

[74] Mussmann, S., Reisler, J., Tsai, D., Mousavi, E., O'Brien, S., and Goldszmidt, M. (2022). Active learning with expected error reduction. *arXiv preprint arXiv:2211.09283*.

[75] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[76] Ng, A. Y. (2004). Feature selection, L 1 vs. L 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78.

[77] Nguyen, A. and Le, B. (2013). 3D point cloud segmentation: A survey. In *2013 6th IEEE conference on robotics, automation and mechatronics (RAM)*, pages 225–230. IEEE.

[78] Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature Visualization. *Distill*, 2(11).

[79] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

[80] Papon, J., Abramov, A., Schoeler, M., and Worgotter, F. (2013). Voxel Cloud Connectivity Segmentation - Supervoxels for Point Clouds. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2027–2034. IEEE.

[81] Park, J., Lee, S., Kim, S., Xiong, Y., and Kim, H. J. (2023). Self-positioning point-based transformer for point cloud understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21814–21823.

[82] Pocock, C. (2019). *3D Scan Campaign Classification with Representative Training Scan Selection*. PhD thesis, University of Cape Town.

[83] Pu, S., Vosselman, G., and Others (2006). Automatic extraction of building features from terrestrial laser scanning. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(5):25–27.

[84] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 77–85.

[85] Qi, C. R., Su, H., Nießner, M., Dai, A., Yan, M., and Guibas, L. J. (2016). Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656.

[86] Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). PointNet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems*, 2017-Decem:5100–5109.

[87] Qian, G., Li, Y., Peng, H., Mai, J., Hammoud, H. A. A. K., Elhoseiny, M., and Ghanem, B. (2022). PointNeXt: Revisiting PointNet++ with Improved Training and Scaling Strategies. pages 1–18.

[88] RayChaudhuri, T. and Hamey, L. G. C. (1995). Minimisation of data collection by active learning. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 3, pages 1338–1341 vol.3.

[89] Riegler, G., Osman Ulusoy, A., and Geiger, A. (2017). Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3577–3586.

[90] Rodríguez, A. C., D'Aronco, S., Schindler, K., and Wegner, J. D. (2021). Mapping oil palm density at country scale: An active learning approach. *Remote Sensing of Environment*, 261.

[91] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer.

[92] Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.

[93] Roy, N. and McCallum, A. (2001). Toward Optimal Active Learning through Sampling Estimation of Error Reduction. In *International Conference on Machine Learning*.

[94] Roynard, X., Deschaud, J.-E., and Goulette, F. (2018). Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *The International Journal of Robotics Research*, 37(6):545–557.

[95] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

[96] Ruther, H., Bhurtha, R., Schroeder, R., Wessels, S., Division, G., Town, C., and Africa, S. (2014). Spatial Documentation of the Petra World. *Africa Geo*, 1:1–12.

[97] Rüther, H., Held, C., Bhurtha, R., Schröder, R., and Wessels, S. (2011). Challenges in heritage documentation with terrestrial laser scanning. In *Proceedings of the 1st AfricaGEO Conference, Capetown, South Africa*, volume 30.

[98] Rüther, H., Held, C., Bhurtha, R., Schroeder, R., and Wessels, S. (2012). From Point Cloud to Textured Model, the Zamani Laser Scanning Pipeline in Heritage Documentation. *South African Journal of Geomatics*, 1(1):44.

[99] Schnabel, R., Wahl, R., and Klein, R. (2007). Efficient RANSAC for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library.

[100] Settles, B. (2009). Active Learning Literature Survey. Number January.

[101] Settles, B. and Craven, M. (2008). An analysis of active learning strategies for sequence labeling tasks. In *proceedings of the 2008 conference on empirical methods in natural language processing*, pages 1070–1079.

[102] Settles, B., Editor, I., Guyon, G., Cawley, G., Dror, V., Lemaire, A., and Statnikov (2011). From Theories to Queries: Active Learning in Practice. 16:1–18.

[103] Seung, H. S., Opper, M., and Sompolinsky, H. (1992). Query by committee. In *Annual Conference Computational Learning Theory*.

[104] Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(4):623–656.

[105] Shi, X., Xu, X., Chen, K., Cai, L., Foo, C. S., and Jia, K. (2021). Label-Efficient Point Cloud Semantic Segmentation: An Active Learning Approach. Technical report.

[106] Srivastava, N. (2013). *Improving Neural Networks with Dropout*. PhD thesis, University of Toronto, Toronto.

[107] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[108] Su, H., Maji, S., Kalogerakis, E., and Learned-Miller, E. (2015). Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953.

[109] Sutton, C. D. (2005). Classification and Regression Trees, Bagging, and Boosting. In Rao, C. R., Wegman, E. J., and Solka, J. L., editors, *Handbook of Statistics*, volume 24 of *Handbook of Statistics*, pages 303–329. Elsevier.

[110] Tetko, I. V., Livingstone, D. J., and Luik, A. I. (1995). Neural network studies. 1. Comparison of overfitting and overtraining. *Journal of Chemical Information and Computer Sciences*, 35(5):826–833.

[111] Thomas, H., Goulette, F. F., Deschaud, J.-E. E., Marcotegui, B., Gall, Y. L., and LeGall, Y. (2018). Semantic classification of 3d point clouds with multiscale spherical neighborhoods. In *Proceedings - 2018 International Conference on 3D Vision, 3DV 2018*, number September, pages 390–398.

[112] Thomas, H., Qi, C. R., Deschaud, J.-E., Marcotegui, B., Goulette, F., and Guibas, L. J. (2019). KPConv: Flexible and Deformable Convolution for Point Clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6411–6420.

[113] Thorndike, R. L. (1953). Who belongs in the family? *Psychometrika*, 18:267–276.

[114] Torrey, L. and Shavlik, J. (2009). Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264.

[115] Tuley, J., Vandapel, N., and Hebert, M. (2005). Analysis and Removal of artifacts in 3-D LADAR Data. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2203–2210. IEEE.

[116] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is All you Need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

[117] Vo, A.-V., Truong-Hong, L., Laefer, D. F., and Bertolotto, M. (2015). Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104:88–100.

[118] Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR.

[119] Wang, Q., Shi, S., Li, J., Jiang, W., and Zhang, X. (2022). Window normalization: Enhancing point cloud understanding by unifying inconsistent point densities.

[120] Wickramarachchi, D. C., Robertson, B. L., Reale, M., Price, C. J., and Brown, J. (2016). HHCART: An oblique decision tree. *Computational Statistics and Data Analysis*, 96:12–23.

[121] Wu, T. H., Liu, Y. C., Huang, Y. K., Lee, H. Y., Su, H. T., Huang, P. C., and Hsu, W. H. (2021). ReDAL: Region-based and Diversity-aware Active Learning for Point Cloud Semantic Segmentation. *Proceedings of the IEEE International Conference on Computer Vision*, pages 15490–15499.

[122] Xie, Y., Tian, J., and Zhu, X. X. (2020). Linking Points With Labels in 3D. *IEEE Geoscience and Remote Sensing Magazine*, 8(March):38–59.

[123] Xu, B., Huang, R., and Li, M. (2016). Revise Saturated Activation Functions. *arXiv preprint arXiv:1602.05980*, pages 1–7.

[124] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27.

[125] Zamani-Group (2021). Zamani project. Available at https://zamaniproject.org Accessed: 2021-09-21.

[126] Zhao, H., Jia, J., and Koltun, V. (2020). Exploring self-attention for image recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10076–10085.

[127] Zhao, H., Jiang, L., Jia, J., Torr, P. H. S., and Koltun, V. (2021). Point transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16259–16268.

[128] Zhou, C., Luo, Z., Luo, Y., Liu, T., Pan, L., Cai, Z., Zhao, H., and Lu, S. (2022). PTTR: Relational 3D Point Cloud Object Tracking with Transformer. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2022-June:8521–8530.