Multiscale Registration of 3D Pointclouds

Miguel Francisco de Souza Dias

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Jacinto Carlos Marques Peixoto do Nascimento
Dr. Pedro Daniel dos Santos Miraldo

Examination Committee

Chairperson: Prof. José Eduardo Charters Ribeiro da Cunha Sanguino
Supervisor: Prof. Jacinto Carlos Marques Peixoto do Nascimento
Member of the Committee: Prof. José António Da Cruz Pinto Gaspar

December, 2022

Multiscale Registration of 3D Pointclouds

Miguel Francisco de Souza Dias

December, 2022

Declaration:

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the *Universidade de Lisboa*.

Declaração:

Declaro que o presente documento é um trabalho original da minha autoria e que cumpre todos os requisitos do Código de Conduta e Boas Práticas da Universidade de Lisboa.

Abstract

One of the main objectives of Computer Vision is to reproduce the human visual system by identifying the different features of what can be seen, such as colour, texture, shape and more. With the advancements being made on the field of Deep Learning, more detailed features are able to be extracted. Pointclouds are a data representation model consisting of a set of data points in a 3D coordinate system. The process of aligning two overlapping pointclouds is designated "pointcloud registration" and has proven to be useful in various fields of research for instance in 3D reconstruction, autonomous driving, medical imaging and more. One way to improve registration is to increase the level of resolution of the features extracted from the pointclouds. In this thesis, an explicit refinement of the pointcloud into multiple scales is implemented in order to extract features of various resolutions of the pointclouds. Furthermore, it is introduced in the neural network's architecture, a module that can aggregate the different features of each scale. Two hierarchical structures, that allow the free manipulation of these scales, are proposed. The developed method has obtained results showing that scales can introduce new information to the features extracted that improve the task of pointcloud registration.

Keywords: Multiscale, Pointlcoud Registration, Feature Aggregation, Deep Learning, Neural Network.

Resumo

Um dos principais objetivos na área da Visão de Computador é de reproduzir o sistema visual humano ao identificar as diferentes características que podem ser observadas, tais como cor, textura, forma e mais. Com o progresso na área da Aprendizagem Profunda, são capazes de ser extraídas características com maior poder discriminativo. As nuvens de pontos são um tipo de representação de pontos de dados num sistema de coordenadas 3D. O processo de alinhar duas nuvens de pontos sobrepostas é designado como "registo de nuvens de pontos" e tem sido cada vez mais útil em diversos ramos de investigação tais como reconstrução 3D, condução autónoma, imagiologia médica e mais. Um modo de melhorar o registo de nuvens de pontos é de aumentar o nível de resolução das características extraídas. Nesta tese, foi implementada uma divisão explícita das nuvens de pontos em múltiplas escalas de forma a extrair características de várias resoluções da nuvem de pontos. Para tal, é introduzida na arquitectura da rede neuronal, um módulo que agrega as difrentes características de cada escala. Duas estruturas hierárquicas, que permitem a livre manipulação destas escalas, são propostas. O método desenvolvido obteve resultados que mostram que as escalas podem introduzir nova informação nas características extraídas que melhorem o registo.

Palavras-chave: Multi-escala, Registo de Nuvens de Pontos, Agregação de Características, Aprendizagem Profunda, Rede Neuronal.

Acknowledgements

I would like to thank my family for all the support and encouragement provided throughout my academic journey and for pushing me to be where I am today.

Then, I would like to thank my supervisors Jacinto Nascimento and Pedro Miraldo for all the help, guidance, and support that were given to me as well as the knowledge and availability whenever I was more uncertain. Along with Gonçalo Pais, who was invaluable and always had a suggestion ready whenever I had an implementation problem, their help made this thesis possible. I would also like to thank my colleagues at my ISR group with whom I would interact in the day to day life.

I would like to thank all my colleagues that I have met and befriended since I began this course, with whom I shared great moments and made unforgettable memories.

A special thank you to all my friends that supported and accompanied me in many other facets of my life.

Contents

List of Figures

List of Tables

Acronyms

CNN Convolutional Neural Network. 1, 6, 10, 11, 15, 16

CV Computer Vision. 1, 5, 6, 11

DL Deep Learning. 2, 4, 15

DNN Deep Neural Network. 10, 12

FAM Feature Aggregation Module. 32

FCL Fully Connected Layer. 32, 33

FMR Feature Match Recall. 35, 36, 39, 40, 42

FPS Farthest Point Sampling. 23, 27, 28, 37, 40, 43

GPUs Graphical Processing Units. 10

ICP Iterative Closest Point. 13

IR Inlier Ratio. 36, 39, 40, 42, 43

KNN K-Nearest Neighbours. xiii, 23, 29, 30

ML Machine Learning. 1, 5, 6, 11

MLP Multi Layer Perceptron. 6, 16

NNs Neural Networks. 5, 6

PFH Point Feature Histogram. 14, 15

RMSE Root Mean Square Error. 35

RR Registration Recall. 35, 36, 39, 40, 43

SGD Stochastic Gradient Descent. 9, 10, 26

Chapter 1

Introduction

This chapter introduces the topic of the work to be developed, providing motivation and context behind it as well as the main goal and outline of the thesis.

1.1 Context and Motivation

Trying to recognize and analyze the different elements of an image or video has become one of the fundamental points addressed in Machine Learning (ML). This investment was possible due to the huge progress allowing more hardware and computational resources to be handled by modern computers. Amongst the most important advancements are Convolutional Neural Networks (CNNs) [25] which paved the way for deep neural networks by obtaining what were the best results at the time in a visual object recognition contest (ILSVRC [12]). Computer Vision (CV) evolved to use these ML techniques obtaining many new state of the art results ([36, 58, 10, 22]).

One of the main objectives of CV is to try to reproduce the human visual system by identifying the different features of what can be seen, for instance, colour, texture, shape and more. One of the most popular ways to model objects and environments is the pointcloud representation. Pointclouds are a data representation model consisting of a set of data points in a 3D coordinate system and may also contain additional information such as colour (not only). Its widespread use has only further increased due to the rising advancements in laser scans such as Lidar and RGB-D. However, these sensors have a limited range of vision. In order to recreate a real world environment or scene many pointclouds have to be matched and aligned. It is possible to align

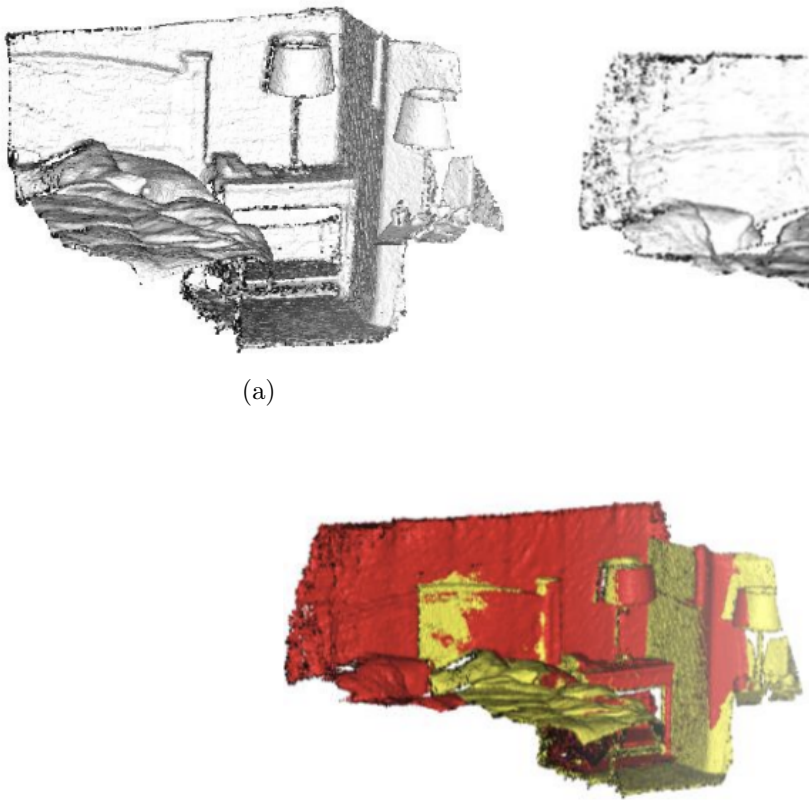
two overlapping pointclouds by computing and applying the transformation between two pointclouds. This process is known as pointcloud registration and it has proven to be useful to reconstruct different scenes for various fields of research such as 3D reconstruction ([47]), medical imaging ([6]), autonomous driving ([7]), 3D localization ([15]), pose estimation ([45]) and more. An example of the registration of two pointclouds can be seen in Figure 1.1. Typically, point cloud registration involves a source and a target pointcloud that have to be aligned. The source pointcloud will be subjected to a rigid transformation (rotation and translation), in such a way, that it is aligned with the target pointcloud. Significant of work has been put in place in order to optimize registration of pointclouds, most focusing on detecting and extracting features from pointclouds to find similarities between them and align them. The strategy above ranges from conventional registration techniques to Deep Learning (DL) techniques.

One issue that arises when registering two pointclouds is the presence of empty space, as is shown on the pointclouds in Figure 1.1. Conversely, in other areas there are a large number of points. Thus, pointclouds have density variations in their point distribution, i.e. they have some regions with more sparsity than others. This indicates that when computing features from the points there will be a variation in the resolution of said features. Some techniques have been developed to bypass this issue, for example in [50], the pointcloud is sampled in order to have a uniform density.

In order to tackle this issue, this thesis will focus on implementing multiscale feature extraction to pointclouds. Multiscale has been used in object recognition and semantic classification ([49, 21]) fields although they have been rising in popularity in the medical imaging field due to its ability in dealing with deformation and increasing accuracy in medical images ([6, 61]). Also, [11] shows that global context provides an improvement on performances and makes the network more robust and invariant to density variations. Thus, by extracting features at progressively larger scales it is possible to obtain a diverse set of features that provide different levels of context, from local to global.

1.2 Objective and Contributions

The objective of this thesis is to implement a pointcloud registration algorithm that uses multiscaling explicitly to adapt to varying degrees of density in the pointcloud. The proposed multiscale approach takes advantage of the different contexts that are available



(a) (b)

(c)

Figure 1.1: Example of source pointcloud (a) and target pointcloud (b) before and after (c) the registration process of a bedroom scene. Source: [11].

in different scales within a pointcloud. The multiscale approach to be developed, takes the state-of-the-art Predator method [22], as a baseline, and further explores 3D point densities within the pointclouds. By using together two distinct methods to group points (Farthest Point Sampling and K Nearest Neighbours) it is possible to have an explicit multiscale architecture. With its integration into the model with feature extraction and aggregation, this combination provides customization in the number and size of the scales allowing for a more personalized abstraction of the objects depending on the dataset at hand.

The main contributions are:

- Pre-processing of pointclouds into personalised multiple scales;
- Two flexible scale insertion structures;

- Hierarchical feature aggregation.

1.3 Thesis Outline

The outline of this work is as follows:

- Background - A brief introduction to Neural Networks along a revision of state-of-the-art methods addressing pointcloud registration either by using classical or DL based strategies for (i) feature extraction, (ii) correspondence matching and (iii) multiscale;
- Pointcloud Registration and Multiscale Pre-Processing and Feature Aggregation - A description of the pointcloud registration from Predator case study along with the proposed process of generating the features of pointclouds in multiple scales and to incorporate them in the network;
- Experimental Results - Details the training process, the evaluation metrics and reports on the results obtained with ablation studies;
- Conclusion - A discussion about the conclusions regarding the work that was done and further recommendations for improvement.

Chapter 2

Background

In this chapter, a brief overview of Neural Networks (NNs) is presented (Section 2.1) along with a comprehensive description of the works related to pointcloud registration (Section 2.2).

2.1 Theoretical Background

2.1.1 Computer Vision & Neural Networks

As was mentioned in Section 1.1, Computer Vision (CV) is one of the many fields of application that ML is present in, and has been trying to emulate the way the visual nervous system operates in order to have computers understand, analyze and process images, videos or 3D scans. The main path to achieve this is to develop algorithms that are able to identify visual patterns similarly as to what humans do when they are looking or observing visual media.

One of the first works was published in 1966 ([33]), where the goal was to connect a camera to a computer and divide different aspects that involve identification and recognition of objects by assigning to individuals the processing of the different properties of each region such as local texture gradients, shading, surface shapes, etc. However, the project was unsuccessful as it proved to be very challenging and demanding due to the technology available. For a long time most artificial intelligence developments happened in other fields while in CV itself most progress occurred in the background for other subjects such as a more mathematical description of visual elements that were described above. One such case is the character recognition technique *Neocognitron* [18] developed

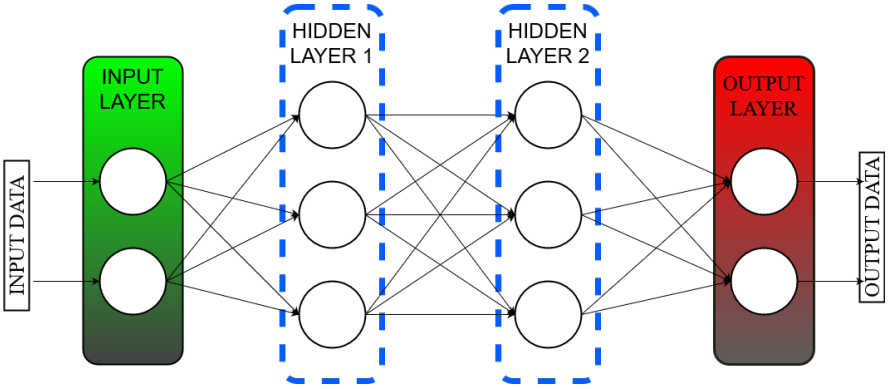


Figure 2.1: An example of a neural network.

in 1980 that was able to learn to identify patterns with and without the aid of labelled data using a primordial version of neural networks.

This paved the way for the biggest advancement in CV as well as for ML which is Artificial Neural Networks, or as they are more commonly known now Neural Networks NNs. The most important contributiont is the AlexNet [25] published in 2012 that popularised the concept of Convolutional Neural Networks (CNNs) by drastically decreasing the error rate of the ImageNet [12] database challenge from 26.2% to 15.3%. As technology advances and computer hardware becomes more and more capable to solve complex problems, NNs became more reliable as their ability to generalize on large amounts of data turned out to be a great asset for most ML tasks. They do so by attempting to simulate the way human brain works akin to how neurons transmit information hence their designation.

So what is a neural network and how does it work? A neural network is a computational system composed of layers that contains nodes which are connected between the layers. These layers come in the form of an input layer where the data inputs are fed to the model, an output layer that outputs the result to be classified, and a varied number of hidden layers where linear and non-linear operations take place. In Figure 2.1, it is illustrated a basic example of a fully connected neural network (or Multi Layer Perceptron (MLP)) which has an input, an output and two hidden layers with three nodes each all connected in between layers.

For a training set of size N, X denotes the input matrix corresponding to all the N elements of the input data and $x^{(i)}$ is the i-th element, as is expressed below,

$$X = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \cdots \\ x^{(N)} \end{bmatrix}, x^{(i)} \in \mathbb{R}^n. \tag{2.1}$$

The networks main goal is, through its computations, estimate the correct label that was assigned to the input data, so the output $\hat{Y}$ is a result of that prediction and is expressed by

$$\hat{Y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \cdots \\ \hat{y}^{(N)} \end{bmatrix}, \hat{y}^{(i)} \in \mathbb{R}^n. \tag{2.2}$$

Each connection from node to node has an associated weight and the value of each node is the linear combination the previous layer's nodes with their corresponding weights with the addition of a bias component, that is particular for each node, as is illustrated in Figure 2.2.

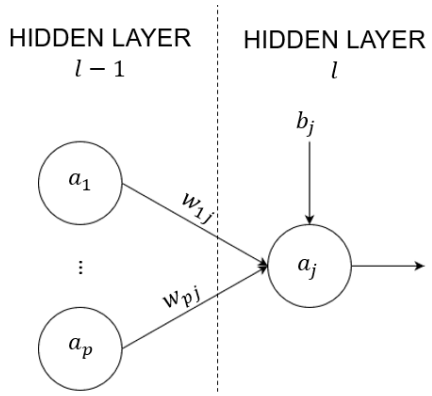


Figure 2.2: Connections between nodes.

Since most mappings that the network is attempting to learn are more than just linear combinations, having a non-linear transformation is important so that more complex problems can be tackled. This non-linear function is called activation function and is to be applied after adding the biases. So, to sum up, the value of node a_j in hidden layer l is obtained by computing the following equation

$$[H]a_j^l = g((\sum_i^p a_i^{l-1} w_{ij}) + b_j), \tag{2.3}$$

where $g()$ is the activation function for hidden layer l, w_{ij} is the weight that is associated from the connection of node i to node j and b_j is the bias of node j. Activation functions can be different depending on the necessity at each particular case, but in their essence they serve to activate (or deactivate) nodes by attributing them a low value if the node value is not desired, sort of like an ON/OFF switch, similar to how the human brain neurons get activated from different stimulus. Below it is shown, in Table 2.1, the most common activation functions and their formulas.

Name	Formula
ReLU	$max(0, x)$
LeakyReLU	$\begin{cases} x & \text{if } x > 0 \\ ax & \text{if } x < 0 \end{cases}$
Sigmoid	$\frac{1}{1+e^{-x}}$
Tanh	$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Table 2.1: Most used activation functions and their respective formula.

Following the calculations from the input all the way through the hidden layers to reaching the outputs is designated a forward pass. The forward pass ends with the computation of the error that was obtained as a result of the predictions that were made, commonly called the loss function. The loss function essentially measures how close the output prediction was to the correct output. As seen with the activation functions, there is also a number of loss functions that serve different purposes and, in Table 2.2, the most commonly used ones are shown.

From Eq. (2.3) the terms that are not known are the weights and the biases, and ideally these values should be such that it leads the network to predict the correct outputs for each of the input elements making the loss as small as possible. Since all problems have different goals we may ask the following question: How are these weights chosen? This is where the learning and training part comes into play. One could start by assigning random weights, proceed through the forward pass and then reassigning the weights that led to the wrong classification, repeating this process until every input

Name	Formula
Mean Squared Error(MSE)/L2	$\frac{1}{N}\sum_{i}^{N}(y^i - \hat{y}^i)^2$
Mean Absolute Error(MAE)/L1	$\frac{1}{N}\sum_{i}^{N}\left\|y^i - \hat{y}^i\right\|$
Negative Log Likelihood(NLL)	$-\sum_{i}^{N}(y_i log(\hat{y}_i) + (1-y_i) log(1-\hat{y}_i)))$
Cross Entropy(CE)	$-\sum_{i}^{N}(y_i log(\frac{e^{\hat{y}_i}}{\sum_{j=1}^{N} e^{\hat{y}_j}}))$

Table 2.2: Most used loss functions and their respective formula.

is correctly classified. The weights modification is done according to the perceptron training rule [31], however it only proves to be successful if the training examples are linearly separable.

So, the most common approach that is taken is the back-propagation algorithm based on Stochastic Gradient Descent (SGD). The weights are modified following the negative slope of the derivative of the loss function. This is basically an optimization problem where the goal is to minimize the loss or the error. This error is then propagated backwards iteratively, in what is called the backwards pass, until the gradients reach a local minima. As multilayer networks have a large amount units it is not guaranteed that back-propagation reaches the global minimum but it has been shown that in practice it still produces great results [31]. In SGD, the weight increment, Δw_i, for a single layer is obtained by computing the gradients of the loss function with regards to the weights of the layer,

$$\Delta w_i = -\eta \frac{\partial L}{\partial w_i}, \tag{2.4}$$

where η is denoted as the learning rate usually having a small magnitude value (e.g. 0.001). The negative sign is to allow the weights to update according to the higher decrease of the loss. Given that each path from the input to the output can be seen a chain of functions, back-propagation uses the chain rule to adjust the weights in each layer.

To summarize, the purpose of a neural network is to transform low level input data

into high level output features by computing the most appropriate weights that map the input data being fed to it into the desired output. The weights and biases are considered the trainable or learnable parameters. In order to compute the most appropriate weights one ought to calculate the loss obtained in the output and propagate it backwards updating the weights in the direction that minimizes the loss.

2.1.2 DNN and their Improvement

Deep Neural Networks (DNNs) have become widely used due to its capacity of having more parameters by adding more hidden layers and increasing optimization of the SGD. It could be said that as the problem becomes more complex to solve so does the network becomes "deeper". As more layers are added the level of the features extracted becomes higher at the cost of coming with much higher resource demands. The use of the term deep started with AlexNet [25] due to its amount of hidden layers and parameters. Although CNNs were used before, AlexNet took advantage of the existing computational hardware, namely GPUs, to efficiently increase training time and accuracy while having the capacity to take in large datasets such as ImageNet [12].

As can be seen in Figure 2.3, AlexNet is composed of eight layers that are learnable, five of which are convolutional and the other three are fully connected. The other layers consist of three max pooling layers and a softmax layer as the final layer. In a convolutional layer, as opposed to a fully connected one, the input is processed by a kernel that acts like a filter and applies a convolution operation on the area selected by the kernel. The kernel sweeps through the input by a spaced interval, defined as stride, resulting in a feature map. As it goes layer by layer this feature map becomes able to extract higher level features making CNNs useful to get more abstract information. Assuming that the input data is a 1000x1000 pixel image, in a fully connected layer where the first hidden layer has 1000 nodes, there would be 10^6x$10^3 = 10^9$ parameters. Since computations of this scope demand high amounts of resource (that are often not available), convolutions provides a good substitute as they are able to reduce the number of parameters whilst enhancing the model's descriptive capabilities [59].

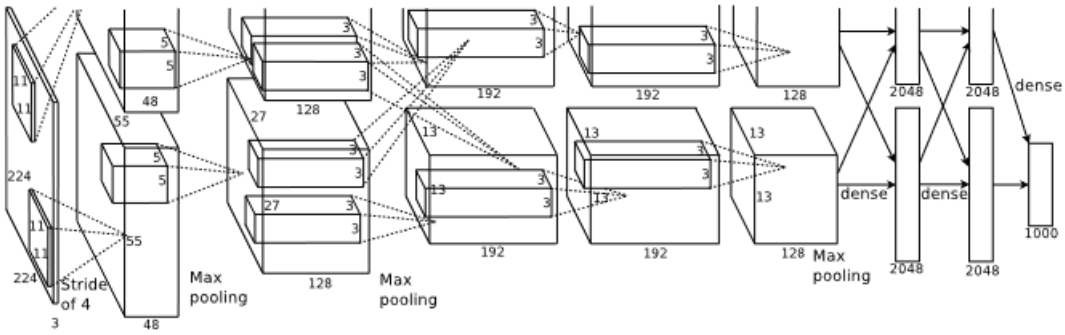


Figure 2.3: Architecture of AlexNet network [25]

A convolution operation featuring two discrete objects is defined as

$$(f * g)(x) = \sum_{z} f(z)g(x - z), \tag{2.5}$$

where, in the case of a CNN, $f()$ and $g()$ refer to the input and the kernel functions respectively. For easier understanding, in Figure 2.4 is represented a visual demonstration of the convolution operation for two dimensional data where the input is a 3x3 shaped tensor and the kernel 2x2 shaped. In this case, the stride is (1,0) which means the kernel slides by one square horizontally. The output is a 2x2 shaped tensor. The kernel's weights are the parameters that have to be trained

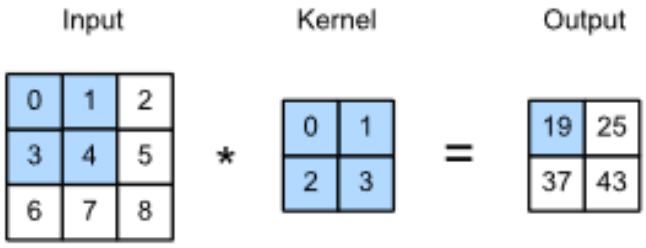


Figure 2.4: Visual demonstration of a convolution [59].

In total, AlexNet possesses 60 million parameters and 650 000 neurons and managed to achieve an error of 15.3% on the ImageNet dataset. Together with computational hardware advancements CNNs possess the capability to better analyse and recognize patterns in multidimensional data, proving to be a great addition not only for CV but for ML as a whole.

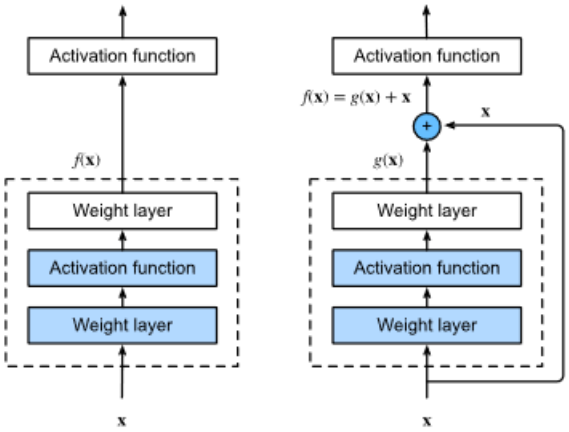


Figure 2.5: Regular block (left) and residual block (right) [59].

Another milestone in the DNNs evolution is the Residual Network [20], shortened to ResNet. Before, training error was increasing as layers were being added but ResNet showed that it is possible to, as the depth increases, diminish the training error by optimizing the residual mapping of blocks of layers instead of the original mapping. In Figure 2.5, it is shown the differences between a regular block and residual block of stacked layers.

For an input x, $f(x)$ denotes the mapping that is learned before the activation function. In a residual block, the mapping that is going to be learned is the residual mapping, $g(x)$. Thus, for a desired identity mapping of $F(x) = x$, the residual mapping can be expressed as,

$$g(x) = f(x) - x \Rightarrow g(x) = 0, \tag{2.6}$$

which facilitates the learning of the weights and biases of the layers in the dotted-line box by pushing them to zero [59].

Stacking layers into blocks and passing the output of the previous block to a deeper layer essentially makes the inputs skip layers at no cost of any additional parameter or computational complexity. This is called skip connection and has proved to help with the increase of accuracy as the network becomes deeper and deeper. ResNet proves to be a very flexible architecture in terms of layer design as blocks can be added or removed. The 152 layer network managed to win the 2015 ILSVRC contest with a testing error of 3.57% and even a 1000 layer network was tested and analysed although it suffered with

overfitting issues.

2.2 Related Works

2.2.1 Classical Methods

The pointcloud registration problem consists in first finding the points that match between the two pointclouds and then computing the transformation that minimizes these matches in the points.

One of the most prominent methods that first appeared is Iterative Closest Point (ICP) [4]. ICP is a technique which first starts out by finding the nearest points between each pointcloud and computing the transformation matrix that aligns the pointclouds. As it is an iterative process, ICP then repeats the above two-step procedure until finding the transformation. It utilizes Procrustes ([43]) to compute the transformation matrix. However, ICP requires a good initialization of correspondences otherwise it can converge to a non-optimal local minima. Since it gained popularity, multiple additions to ICP have been developed ([37, 5, 44]) in order to optimize the correspondences process.

ICP based methods do not need their points to already be paired beforehand. As previously mentioned, this posed a convergence problem. On the other hand, correspondence based techniques have been working on getting the best correspondences and have a final method, such as [52], that estimates the transformation parameters with great accuracy and no convergence issues. Before Deep Learning came in the picture, finding the best correspondence between points was the most challenging step, so early feature extraction methods have tried to improve on this stage. In terms of the evolution of extraction of 3D features approaches, the classical methods have focused on using handcrafted features to describe local geometry ([23], [39], [41]). These features, as opposed to learned features, are obtained by running an algorithm that analyses the pointcloud and retrieves low level information based on its properties, such as edges or corners.

Spin Images [23] utilizes surface matching to find the best correspondence between the surface that is sensed in the environment and the surface object that is stored in memory, as is seen in Figure 2.6. The objects are collected by using a shape representation. In this representation, the shapes of the surfaces are characterized by surface normals and 3D points that have a descriptive image associated. The image description above is

generated by taking the oriented point, which refers to the 3D point and its correspondent surface normal, creating a local basis around it on the surface of the object. Now, other points that are in this basis can have their coordinates be described by only two parameters (the distance to the oriented point in a two dimensional cylindrical coordinate system) which will be added to a 2D histogram, thus generating a local descriptor. By creating this image for each oriented point it becomes possible to differentiate between points in an object.

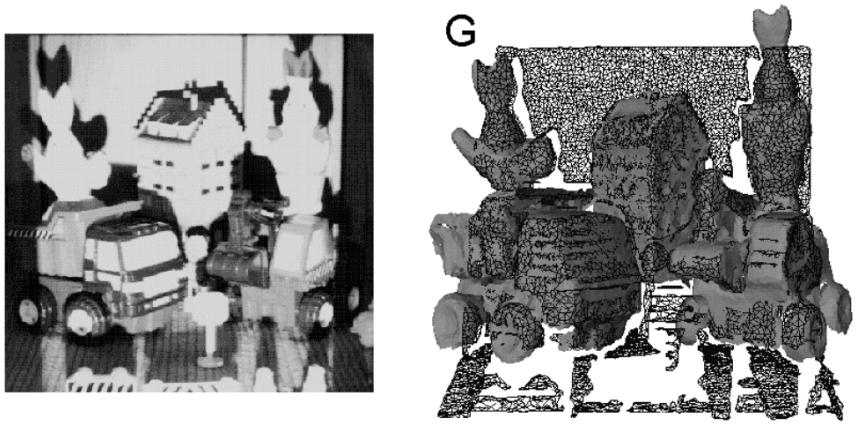


Figure 2.6: Recognition of model using surface mesh from Spin Images [23].

Unique Shape Context (USC) [51] takes a 3D spherical grid around each feature point. Each subdivision of the grid (bin) takes into account all of the surface points in it by doing a weighted sum of these points. However, by computing the descriptor of a feature point on a single local reference frame, USC avoids wasting resources and improves speed, efficiency and accuracy in the matching stages.

Point Feature Histogram (PFH) [39] focuses its descriptors for points correspondences between pointclouds registration. It computes the k neighbourhood for the selected point in a defined radius. For every point of pairs in the neighbourhood, four different geometric features are computed with the help of both the surface normal and the Darboux Frame of both points, preserving only the pairs that present the same values for all features. This outputs into 16 histogram bins formed for each point improving robustness to outliers and invariance to the density of the pointcloud.

Fast Point Feature Histogram (FPFH) [38] improves on PFH and adds a simplified version of FPFH to reduce the computational complexity that PFH imposes. It com-

putes one less feature than PFH and replaces the process of repeating the computations for every single pair of points in the neighbourhood of the selected point by doing computations only for the point itself and its neighbours. Also has the advantage of being able to be used online.

2.2.2 Data Driven Methods

Feature Learning

More recently, methods for feature extraction concerning pointcloud registration using DL have become more popular by consistently outperforming hand-crafted based feature extraction techniques. These methods focus on learning the optimal feature descriptors using distance metrics, such as triplet loss ([24]) and contrastive loss ([58]), to maximize the distance of non-matching correspondences of features while minimizing the distance of features that are closer. Then, the alignment process is achieved by RANSAC [17]. RANSAC is a very robust, efficient and reliable method of detecting and filtering out the outliers of the model. RANSAC is widely used in pointcloud registration since it can find the inlier correspondences and estimate a mathematical model that corresponds to the transformation matrix in one step.

3D Match [58] is another line of work that proposes a neural network, namely a 3D convolutional neural network, so that features can be learned by example for 3D geometry. It divides data into 3D patches around each interest point so that in each patch there is information of whether its center is on a surface or far from a surface. In training, a CNN is used (similar to a siamese 3D CNN [40]) for obtaining the feature representation. The contrastive loss function is used to evaluate whether the feature descriptors generated for the patches of point pairs match or not.

Compact Geometric Features (CGF) [24] learns descriptors directly from pointclouds as opposed to subdividing into volumetric regions as 3D Match previously used. Instead, spherical histograms centered at interest points are utilized and every point in the neighbourhood is binned into the histograms' subdivisions. As a result of this parametrization, a high dimension feature descriptor is created. A fully connected network is used in order to turn it into a more compact representation, i.e. reduce the dimensionality, together with triplet loss to minimize the error. To find corresponding points between pointclouds, CGF computes the nearest neighbour from interest points in one pointcloud

to another.

PPFNet [11] also uses raw point cloud data to take advantage of all the details of the pointclouds. Taking a set of points on an arbitrarily defined region and its normals, a representation for the local geometry is produced, with the denomination of local patches. From there, it is used Point Pair Features (PPF) [13], which consists of 4 dimensional descriptors, on pairs of points from the set. Then, these descriptors are fed to a PointNet [36] that extracts local features. This network consists of Multi Layer Perceptrons (MLP) and a max pooling function to aggregate the point features into a global feature, making it globally aware of the context of its descriptors, i.e. it can recognize that same features could belong to different objects. PPFNet also proposes a new loss function, N-tuple loss, that is similar to contrastive and triplet losses except it allows N number of input points instead of two and three respectively for contrastive and triplet losses.

3DFeat-Net [2] also uses similar architecture to the PointNet [36] network, that is illustrated in Figure 2.7, but adds a detector to identify keypoints. It takes three pointclouds as input: an anchor pointcloud, and two pointclouds which have a positive and a negative distance to the anchor. The three pointclouds go through a clustering stage and then the detection stage where clusters are put through three fully connected layers, a max pooling layer, two more fully connected layers, and two separate fully connected layers one for orientation and the other for attention. This outputs a desired orientation score for each cluster that makes the descriptor invariant to original orientations, and an attention score that characterizes the saliency of clusters. The clusters and orientations then go through the descriptor stage where descriptors that carry additional contextual information from clusters and individual points are generated. These descriptors go through the feature alignment stage where all pairs of individual features from different pointclouds are compared using triplet loss that takes into account the attention weights previously computed. These weights are helpful to separate the distinctiveness of each cluster.

Fully Convolutional Geometric Features [10] manages to greatly increase the speed of which the descriptors are generated while improving accuracy by applying the input pointcloud through a series of 3D fully convoluted layers [27]. This fully convolutional network diverges from CNNs by not having fully connected layers (only performs convolutions and elementwise non-linearities ([10])), vastly improving efficiency and speed. As

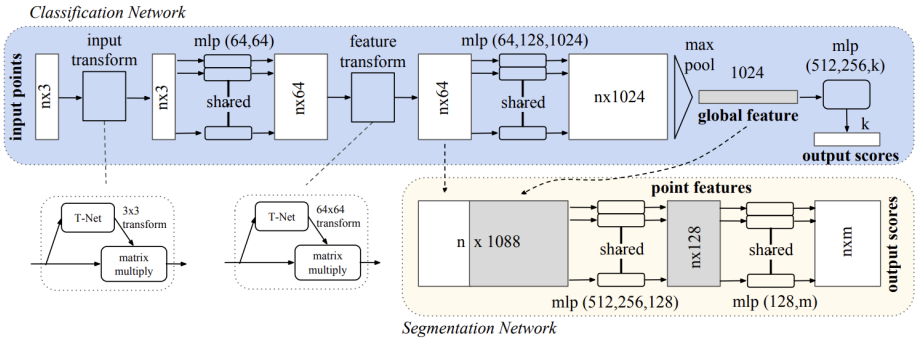


Figure 2.7: PointNet architecture [36].

a result, the generated features are not independent and identically distributed meaning they can be associated with other features in the vicinity. Due to this characteristic, a hard negative mining [56] is used to filter out possible false negatives found when computing contrastive and triplet losses originating a hardest-contrastive loss and a hardest-triplet loss.

End-to-end Learning

Conversely, in end-to-end methods, the registration process is done within the network. That is, both the feature extraction and transformation estimation are optimized during the training stage.

PointNetLK [1] combines the PointNet [36] network for feature descriptors generation with a modified Lucas & Kanade (LK) [30] algorithm, for the estimation and optimization of the transformation and alignment. PointNet takes the pointcloud as input and outputs a descriptor, that serves to compute the optimal transformation. The transform parameter is dependent on the optimal twist parameter that is calculated by a Jacobian matrix which is the formulation from LK. The modified LK algorithm proposed is used to incrementally update twist parameters by calculating the Jacobian only once meaning that the complex computations that it entails are done outside the optimization process making PointNetLK much more efficient. The Mean Square Error is then used to minimize between the estimated transform and the ground truth transform.

DeepICP [29] on the other hand uses PointNet++ [35] for feature extraction, but adds an attention layer similar to the one utilized in 3DFeatNet [2] in order to better select the keypoints to make the corresponding process easier. To get a more detailed

descriptor, the features go through a final embedding stage which consists of three fully connected layers and a max pooling layer. Lastly, a similar approach to ICP is used trying to find the corresponding points. However, due to the ICP being non-differential during backpropagation when it is choosing the corresponding points, these corresponding points are instead generated based from the features extracted and their similarity using 3D convolutional neural networks. This step essentially replaces the final transformation parameter estimation step that was common in Feature Learning methods. The loss function implemented is a linear combination of the L1 loss to match the generated corresponding point to its ground truth computed from ground truth transformation, and the loss that matches the correct transformation parameters.

2.2.3 Matching Correspondences

Matching correspondences of points is an important part of pointcloud registration as it is the final step before finding the transformation parameters. A robust and invariant matching process helps improve registration performances. SIFT [28] is an efficient and robust method of feature detection and matching developed for 2D object recognition and image matching. The features detected are invariant to rotation, translation and scale and have a key location assigned that is used for indexing and matching. In SURF [3], the descriptors generated are invariant to different transformations than of SIFT and it managed to outperform SIFT in terms of running time and computational complexity.

Superglue [42] borrows inspiration from the transformer [53] architecture to make use of self and cross attention. It managed to register great improvement in match precision and is able to run in real time. Inspired by Superglue, LoFTR [45] introduces a novel approach that outperforms its predecessors by removing the feature detector from its feature extraction stage.

2.2.4 Multiscale Methods

In multiscale based approaches the goal is to detect low-level features (low number of dimensions of the feature descriptor, e.g. corners and edges) while having the capability to abstract and also be able to recognize the higher level features (high number of dimensions). It has proven to be useful in a large range of fields. In [6] a Multi-scale Convolutional Neural Network is proposed for the classification of images of cells.

In this work the convolutional neural networks are used multiple times in the input data each with different resolutions in order to obtain different features from the same dataset. In Multi-Instance Multi-Scale (MIMS) CNN [26], the feature maps obtained from a pretrained CNN are used as inputs that are fed into a second convolutional layer with varying sizes representing different scales, which proves to be useful to catch scale invariant features, as is seen in Figure 2.8.

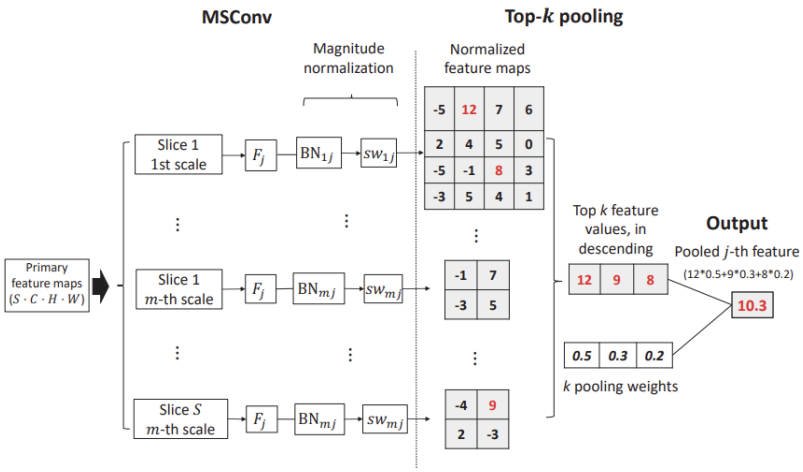


Figure 2.8: Multiscale convolution network used by MIMS [26].

MCNet [57] uses a pyramidal convolution [14] to process the input data for the purpose of cloud detection. For different levels of the pyramid, features are put through each level which has different kernel sizes emulating different scales. PointNet++ [35] utilizes a hierarchy concept when computing features at different scales. By progressively sampling and grouping differently sized point sets and then applying a PointNet [36] layer to each grouping it is possible to incorporate features from multiple scales. These multiscaled features are then interpolated so that they can propagate back to the original points, and are also used for classification (after going through fully connected layers).

In semantic classification, [49] implements multiscale neighbourhoods for 3D point-clouds. In these neighbourhoods, by implementing iterative subsampling of the cloud computational costs are reduced. The use of different scales helps to identify various sized objects by having equally sized neighbourhoods regardless of the density of points. In [21], it is proposed a multiscale network where in each layer features from different scales are produced. The goal is to have high level features in the beginning (where usu-

ally only low level features are extracted) without jeopardizing the high level features in the latter stages, since all the layers are connected to all the classifiers.

Multiscale Point Cloud Geometry Compression [54] proposes a multiscale framework that achieves great improvements in compressing sparse, unstructured pointclouds. It managed to reduce 75% if the memory space requirement while improving efficiency and reliability. In MSNet [55], it is proposed a multiscale voxelization that subdivides the pointcloud into different sized voxels so that various sized objects in a pointcloud can be classified. A multiscale convolutional network takes advantage of this voxelization to extract features of different resolutions by applying convolutional networks with differently sized kernels. Compared to similar methods, MSNet proves to have great generalization capabilities while improving classification results.

In [48], a combination of multiscales and attention is used in hierarchical way for semantic segmentation. Attention is used to optimize the weights between scales as opposed to averaging the weights. By combining the multiscale semantic predictions in a hierarchical structure, fewer scales need to be utilised improving accuracy whilst decreasing complexity and memory usage. ResNet-50 serves as the backbone for this implementation. As can be seen in Figure 2.9, the network uses attention to perform predictions using both scales.

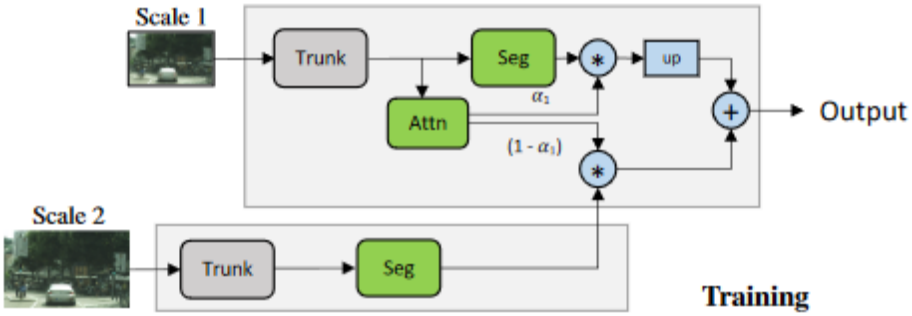


Figure 2.9: Hierarchical architecture for multiscale attention [48].

2.2.5 Summary

In this section, various works for pointcloud registration were presented ranging from specializing in extracting features to finding correspondences. It was seen how the transition from handcrafted features to learned features provided a big leap to the descriptive

quality of features. By the end, it details the way multiscale is incorporated into the network. Compared to the proposed approach, these works subdivide the pointcloud into scales within the network by varying the kernel size in convolutional layers and other variations. However, as it is going to be presented in the next chapter, this work experiments with defining the scales prior to the network in an explicit fashion, focusing more on the benefits that each scale can provide before they are encoded in the network.

Chapter 3

Pointcloud Registration & Multiscale Pre-Processing and Feature Aggregation

In this chapter, it is presented a case study on the Predator [22] algorithm (Section 3.1), following the paper where it was published, in order to introduce the pointcloud registration problem formulation and expand on its architecture which served as the baseline for this thesis's work. Also, the process of obtaining the multiple scales is described (Section 3.2) starting with the Farthest Point Sampling (FPS) that is applied to the pointclouds, following with the K Nearest Neighbours (KNN) that groups points closest to the sampled points. Next, the feature extraction and aggregation process is described (Section 3.3) in which the scaled features are processed in hierarchical fashion in order for them to have the same dimensional space as the features they are going to get aggregated into.

3.1 Pointcloud Registration - A case study on Predator

A pointcloud P can be defined as a set of N 3D points $\{p_i \in \mathbb{R}^3 | i = 1...N\}$. For two pointclouds P and $Q = \{q_i \in \mathbb{R}^3 | i = 1...N\}$, where $\{(p_i, q_i)\}$ are point correspondences, the main objective in pointcloud registration is to compute the rigid transformation parameters (rotation matrix $R \in SO(3)$ and translation vector $t \in \mathbb{R}^3$) that aligns P and Q as

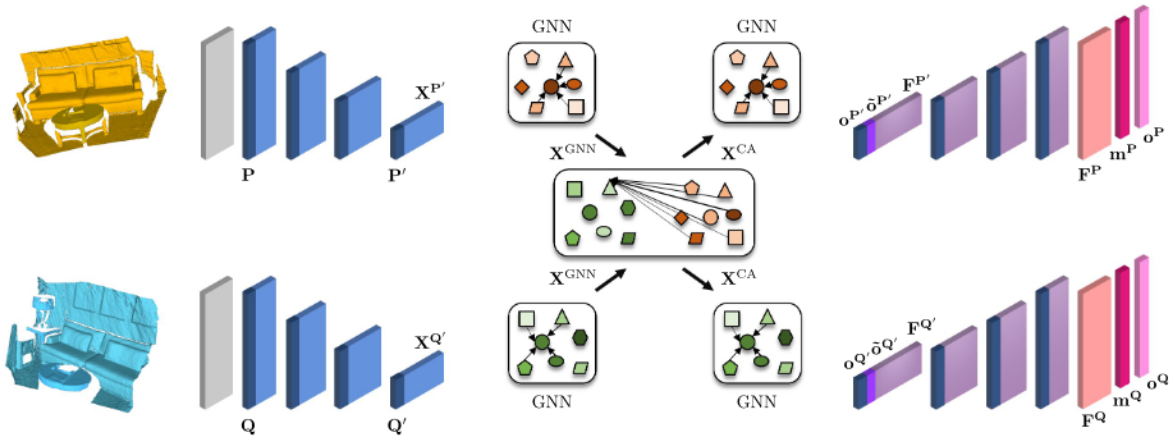


Figure 3.1: Network architecture of Predator [22].

$$R, t = \underset{R \in SO(3), t \in \mathbb{R}^3}{argmin} \sum_{n=1}^{N} \|q_n - (Rp_n + t)\|_2 , \tag{3.1}$$

where $\|.\|_2$ denotes the Euclidean norm. Still, it is only possible to calculate the optimal rotation matrix and translation vector if the correspondences are known.

Predator tackles a side of the pointcloud registration scene that is often not considered: pairs of pointclouds that have low overlap between them. That is because most works only consider cases where the overlap is higher than 30% as with high overlap between pointclouds the task to find correspondences and matches becomes somewhat easier.

Due to the fact that in this work the implementations were made on the Minkowski Engine version of Predator, this case study will focus on that version as opposed to the kernel point convolution one described in [22]. Minkowski Engine [9] is a neural networks focused library that utilizes sparse tensors to represent data as opposed to normal tensors. Contrary to normal tensors, sparse tensors only consider non-zero values making it easier to store and compute them. Sparse tensors favours high dimension types of data, such as pointclouds, as operations become faster and more efficient.

In Figure 3.1 it is represented the architecture of the network of Predator. It is composed by an encoder, an overlap module and a decoder.

The encoder starts by downsampling the input source and target pointcloud, P and Q respectively, to have a uniform point density, then encodes the points into interest points

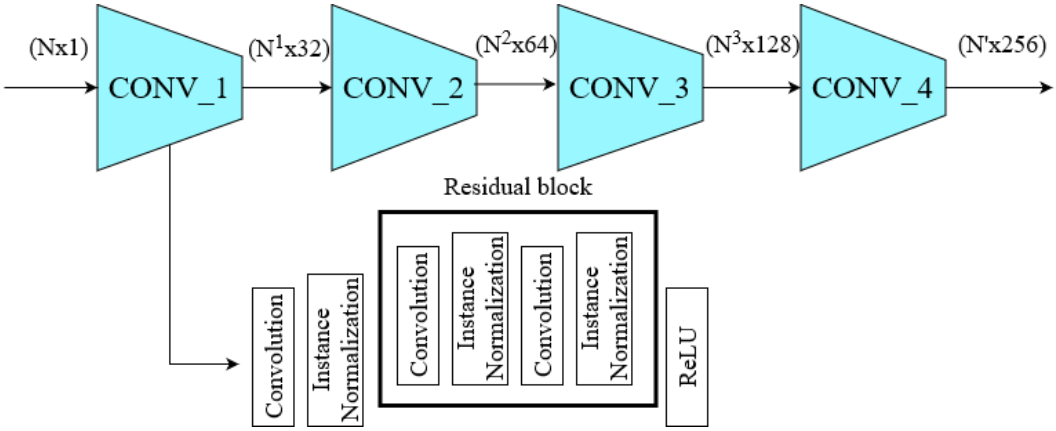


Figure 3.2: Encoder structure

with a series of convolutions that contains blocks of layers as was done with ResNet. These interest points, $P' \in \mathbb{R}^{N' \times 3}$ and $Q' \in \mathbb{R}^{M' \times 3}$, also carry their corresponding features that were extracted, $X^{P'} \in \mathbb{R}^{N' \times b}$ and $X^{Q'} \in \mathbb{R}^{M' \times b}$. The series of convolutions consists of four consecutive blocks of layers where each block contains a convolution layer, an instance normalization layer and a residual block, as can be seen in Figure 3.2. At the end of each block there is a ReLU activation function. Instance normalization is used to scale features and normalize them so there is not a disproportionate impact on the rest of the network of differently ranged feature vectors. In each block, there is an increase in the dimensionality of the features extracted as the level of detail wanted gets higher.

These points and features are then put through the overlap attention module. This module uses self and cross attention to infer contextual information within the point-cloud and with its paired pointcloud. Attention means inserting information into the descriptor that has context regarding its surroundings. Self-attention is implemented using a graphical neural network and proves to be of value because it takes into account the different interactions between each element in the input. Cross attention uses the Transformer architecture [53] which gained popularity by greatly reducing training time while having state of the art performances in the natural language processing field. The Transformer uses the attention technique throughout the layers to compute the attention weights in a parallel way regarding previous inputs. Predator takes advantage of this by mixing, in each point in pointcloud P', information of every point in pointcloud Q' and

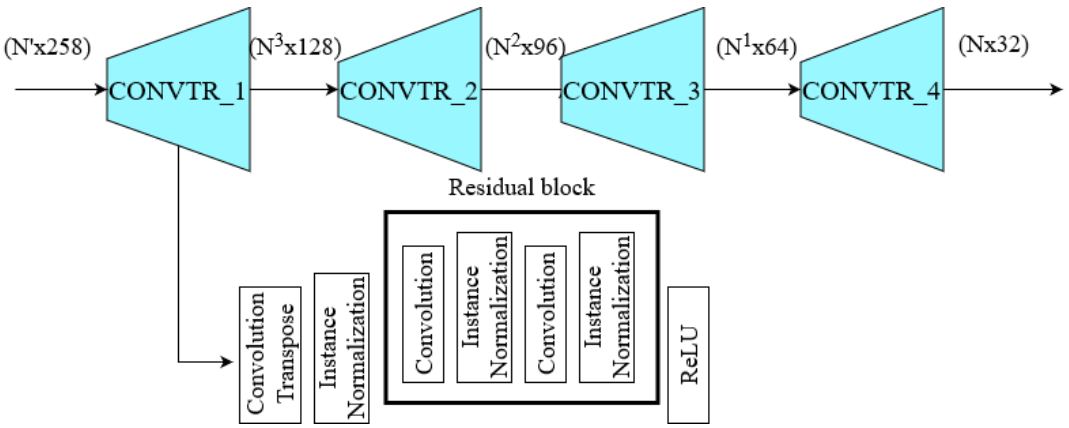


Figure 3.3: Decoder structure

vice versa. Given the goal is to improve registration in low overlap regions, this module proves to be an important step towards matching correspondences as pointclouds can share and learn the little information they have in common. This module also computes an overlap score, $o^{P'}$ and $o^{Q'}$, that can be seen as the probability of a point belonging in the overlap region and a matchability score that.

In the decoder, the features computed by the overlap attention module, $F^{P'}$ and $F^{Q'}$, and its respective points are forwarded into a similar structure as the encoder but in reverse order, to upsample back into the original dimensions of the pointcloud using a transposed convolution. The resulting output includes the features, F^P and F^Q, that represent the per-point descriptors, the overlap scores, o^P and o^Q, and matchability scores, m^P and m^Q, that computes the likelihood of a point matching correctly compared to the points in its pointcloud pair.

The optimizer used is the SGD which means the parameters will be trained by minimising the loss function until convergence is reached. Predator utilizes three different loss functions supervised by ground truth correspondences to make use of the overlap and matchability scores. Overlap loss and matchability loss are computed in similar fashion. First compute the negative log likelihood of the respective overlap or matchability score one time for the source and another time for the target pointclouds and then calculate the average. The third is the circle loss ([46]), which was introduced to be unified when handling both class labels and pair-wise labels. For pair-wise labels, circle loss degenerates into triplet loss with hard mining. This loss has been proven to improve optimization and allows for a better convergence point than triplet loss.

With the novel overlap attention module Predator proves that by giving attention to the overlap region between a pair of pointclouds it is easier to find point correspondences. It manages to produce state of the art results both on low overlap data as well as normal data compared to feature based methods and direct registration methods.

3.2 Multiscale

This section details the main contribution of this work which is the partitioning of the pointcloud in various scales, which in turn can be fined tuned and changed depending on the number of scales and the context and relations of different objects in the scene that is to be recognized. The multiscale implementation was coded in the pre-processing part of the architecture, outside of the network.

3.2.1 Farthest Point Sampling

FPS was first introduced in 1997 [16], when it was used to sample images. The main advantages were described as retaining its uniformity with the increased density, providing efficient means for sparse image sampling and display, and the points are irregularly spaced, exhibiting anti-aliasing properties [16]. When applied to 3D data, more specifically, 3D pointclouds, it was popularised by Pointnet++ [35] arguing that it allowed for a better coverage of the pointcloud than other sampling methods (such as random sampling for example) and does not ignore more sparse areas of a pointcloud.

In [16], FPS is defined as follows: For a sample set, $S = \{s_i\}_{i=0}^{i=N-1}$, over a region A, the next sample should take place at the point p, which is the farthest away from the previous samples, i.e,

$$d(p, S) = \max_{q \in A}(\min_{0 \leq i \leq N}(d(q, S_i))), \tag{3.2}$$

where N is the total number of points in the sample set, q is the previous sample and $d()$ the distance from a sample to each of the sampled set's points.

The iterative process (detailed in Algorithm 1) begins by selecting a random point from the pointcloud. This is going to be the first sampled point. Then, its distances to all other points are calculated and the point with the highest distance is selected to be the next sampled point. After this point is obtained the distances are computed again.

They are compared to the previous distances and updated if smaller values are produced. The points that were already sampled are promptly removed from the pointcloud.

Algorithm 1 Farthest Point Sampling

Input: Pointcloud with dimensions N×3 (N is the number of points in the pointcloud, 3 are the coordinates), and number of samples

```
Initialize the sampled points array
Initialize the distances array with an inf value
Select a random point and save it to the sampled points
Remove saved point from the pointcloud array
for i ← 1 to n_samples do
    Obtain i − 1 sampled point
    Calculate its distance to all other points
    if New distance is smaller than previous distance then
        Update distance
    end if
    Select point that has the highest distance
    Store it in sampled points array
    Remove it from the pointcloud array
end for
return Sampled points array
```

FPS inputs the pointcloud and the number of samples that the resulting pointcloud will have. The number of samples is calculated as a percentage of the total number of points of the input (i.e. n_samples=1 means the resulting pointcloud will have 100 points if the input one has 10000). This is done so it is easier to get a more consistent scaling as the total number of points each pointcloud contains varies in the dataset.

Although it is a great sampling tool, even more so for pointclouds, FPS is computationally expensive boasting a complexity of $O(N \times p)$, where N is the number of points int the pointcloud and p is the number of sampled points desired. This also makes it time consuming to process very large pointclouds when the number of samples to be acquired has a relatively small value. Given that pointclouds in the 3DMatch dataset can have up to 50000 points, using sampling to 1% increases the total runtime of the network by 80% whilst 2% has an increase of 156%.

3.2.2 K Nearest Neighbours

Afterwards the sampled pointcloud is put through the KNN method in which the original pointcloud is searched for the closest points in the neighbourhood of the previously sampled points.

The KNN method is a well known algorithm used for classification where an input sample gets attributed a class based on the points neighbouring the sample. However, in this particular case, the task at hand is to retrieve the points closest to the sampled point only, as is demonstrated in Figure 3.4.

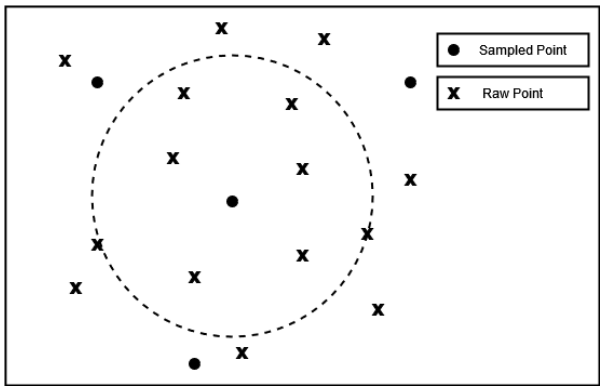


Figure 3.4: Visual demonstration of KNN's use.

In order to increase efficiency and speed the search was done using open3d's library [60] in conjunction with FLANN's KDTree [32]. KDTrees are a type of search tree that is used to structurally partition data in k dimensions. For pointclouds, k equals to 3. Seeing as the raw pointcloud is partitioned into its 3 dimensional space, finding its nearest neighbours proves to be much faster and optimized making the runtimes not alter much with the number of neighbours chosen.

The method of obtaining the nearest neighbours of the sampled pointcloud starts by building a KDTree of the raw pointcloud. Then, for each of the points in the sampled pointcloud, a search is done of its neighbouring points in the raw pointclouds according to the number of neighbours defined. As there cannot be a guarantee that some points will be acquired only one time, it is then necessary to remove any duplicate points that may be considered a neighbour for two different sampled points. The algorithm is detailed in Algorithm 2.

Algorithm 2 K-Nearest Neighbours

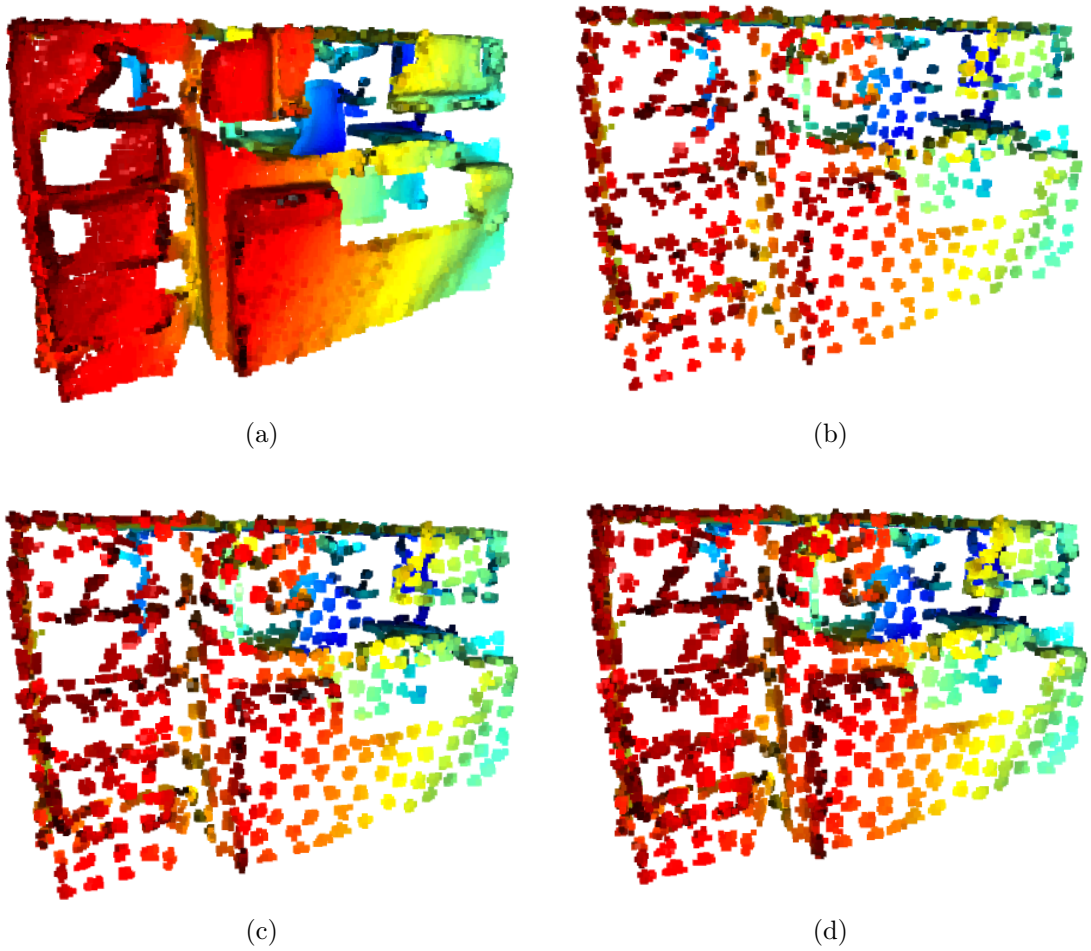
Input: Raw pointcloud with shape N×3, sampled pointcloud with shape S×3, number of neighbours (int)

1: Build KDTree of raw pointlcoud
2: Initialize empty index array
3: **for** point in sampled pointcloud **do**
4: Search raw pointcloud's KDTree for the nearest neighbours of the sampled pointcloud's points
5: Obtain the neighbours and store them in the index array
6: **end for**
7: Remove possible duplicate indexes
8: Create new pointcloud containing only the points from the index array
9: **return** New pointcloud

The number of neighbours chosen is what is going to determine the final size of the scales. Given that pointclouds have different amounts of points it is difficult to get a consistent scale that groups objects that have varying size. Hence, three different scales are used by changing the number of neighbours that are going to be searched. The number of neighbours are the following: 10, 15 and 20. These were chosen as to provide a good way to encompass the global features of the differently sized objects that can be found in the pointclouds. As is shown in Figure 3.5, the scales can perceive different amounts of details with the grouping produced by KNN and can carry new information that the raw pointcloud may not contain.

3.3 Feature Extraction & Aggregation

After the pre-processing is finished, at the input of the network there are 4 pairs of pointclouds. The raw pointclouds and the pointclouds associated with each scale, whose features are going to be combined. So how is the new information produced from the scales communicated to the raw pointcloud? Inspired by [48], a hierarchical feature extraction and aggregation is developed so that the most useful features of each scale are combined and aggregated into the raw pointcloud. The structure is deployed alongside Predator's encoder and is displayed in Figure 3.6. Scale 0 refers to the raw pointcloud and the $\oplus$ represents a concatenation along the feature axis.



(a) (b)

(c) (d)

Figure 3.5: Raw pointcloud (a), scaled pointcloud with 10 neighbours (b), scaled pointcloud with 15 neighbours (c) and scaled pointcloud with 20 neighbours (d).

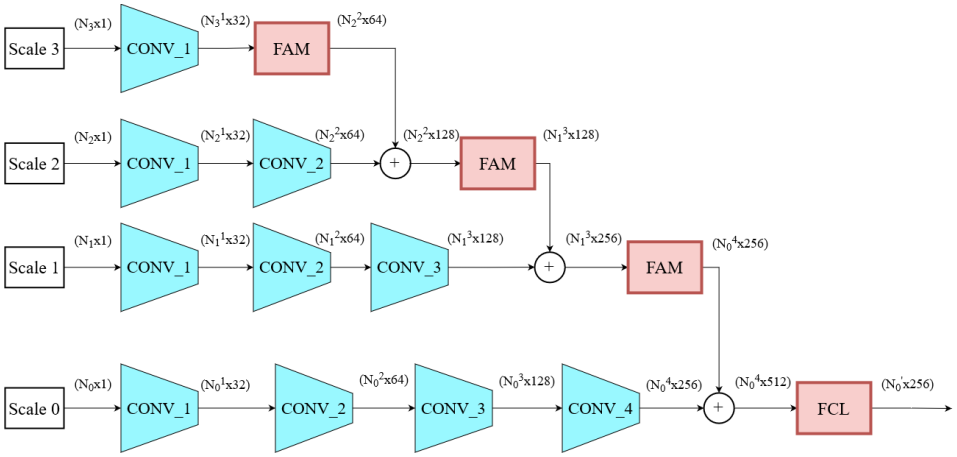


Figure 3.6: First encoder structure with hierarchical feature extraction and aggregation.

Instead of hierarchically combining scales with each other, a second different structure was also implemented where each individual scale goes directly into the raw pointcloud, as is shown in Figure 3.7.

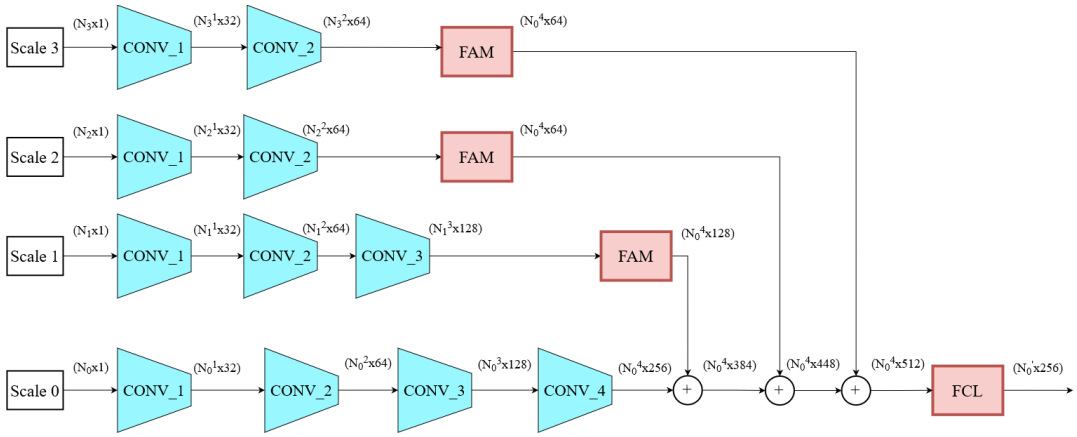


Figure 3.7: Second encoder structure with hierarchical feature extraction and aggregation.

Each scale is encoded in a distinct manner resulting in varying sizes for the feature vectors. Since the different scales do not have the same number of points, a feature aggregation module is inserted so that concatenation between scales' features is possible. The scales are ordered according to their sizes so the bigger scales get their features inserted in the smaller scales as in a hierarchical fashion.

In the feature aggregation module (FAM) (FAM block in Figure 3.6) the feature vectors are resized while making them more robust and bringing up its most distinct features. As it is shown in Figure 3.8, for a feature vector with size $N \times b$ where N is the number of points and b is the number of features, the desired output is a feature vector of size $M \times b$ where M is the number of points that the next scale possesses. Given that the input to the feature aggregation module is a feature vector of concatenated features from different scales, a fully connected layer (FCL) is applied so as to encode them into the scale. Given that features that have more interest usually contain higher values, a global max pooling is used to reduce the dimensionality into a $1 \times b$ size so that only highest valued features in each dimension are conserved. The resulting feature vector consequently gets repeated M times so as to create a $M \times b$ shaped vector.

Figure 3.8: Feature aggregation module.

After the final scale is concatenated into scale 0, a final FCL block is applied to encode the previous scales features with the raw pointcloud's encodings. There is also a dimensionality reduction to make the feature vector fit the overlap attention module.

With these two structures, it is possible to combine features between multiple scales making the network learn a richer and higher level of information. They prove to be flexible to the point where each part of the hierarchy can be removed or added with ease. Along with scales being adjustable, as explained in Section 3.2, this combination shows great power in versatility and adaptation of the variety of ways that scales can be used to infer local and global context of information in pointclouds and other types of data.

Chapter 4

Experimental Results

This chapter presents the experimental evaluation that was done, including a description of the dataset and evaluation metrics (Section 4.1), going through the process of training the network (Section 4.2), presenting and discussing the results obtained (Section 4.3) and ablation studies (Section 4.4).

4.1 Dataset and Evaluation Metric

The data sets were used from the 3DMatch [58] dataset which contains 62 indoor environments such as kitchens, living rooms, offices, etc. taken from from different sensors (a few examples of the scans from this dataset can be seen in Figure 4.1). They are separated into 46 environments for training, 8 for validation and the other 8 for testing as was utilized in [22]. Each environment is made of segments or fragments that were extracted from a RGB-D video in order to remove some of the noise. This has the added benefit of being more complete and reliable than individual ranged images.

The main evaluation metrics are Registration Recall (RR) and Feature Match Recall (FMR) [11].

As explained in [10] and [22], RR measures in percentage the amount of pointclouds pairs that are considered true positive in regards to the ground truth, i.e. the number of pairs for which the registration successfully estimates translation and rotation parameters. RR computes the Root Mean Square Error (RMSE) for every point pair p and q of the pointcloud and checks the fraction for which $RMSE < 0.2$. The RMSE is defined as follows,

Figure 4.1: Some examples of the scans used in the 3DMatch dataset [58].

$$RMSE = \sqrt{\frac{1}{H_{ij}^*} \sum_{(p,q) \in H_{ij}^*} \left\| \hat{T}_P^Q(p) - q \right\|_2^2}, \tag{4.1}$$

where H_{ij}^* are the ground truth correspondences and $\hat{T}_P^Q(p)$ is the estimated transformation from pointcloud P to pointcloud Q ([8]).

As for FMR, it measures the likelihood that a correct transformation can be estimated based on the inlier matches and is defined by

$$IR > 0.05, \tag{4.2}$$

where IR is the Inlier Ratio. IR measures how close is the distance, within a threshold, between two putative correspondences matched by their features after the ground truth transformation is applied. Following [22], IR is defined as

$$IR = \frac{1}{|K_{PQ}|} \sum_{(p,q) \in K_{PQ}} \left[\left\| T_P^Q(p) - q \right\|_2 < 0.1 \right], \tag{4.3}$$

where K_{PQ} represents the corresponding matches between pointclouds P and Q, T_P^Q the ground truth transformation and $[\cdot]$ denotes the Iverson bracket (see Appendix A). In short, FMR measures how good the features generated are.

As per [22], FMR is not able to guarantee that a transformation can be obtained from the correspondences which makes RR the more reliable metric since it evaluates the registration of the pointclouds.

4.2 Experimental Setup

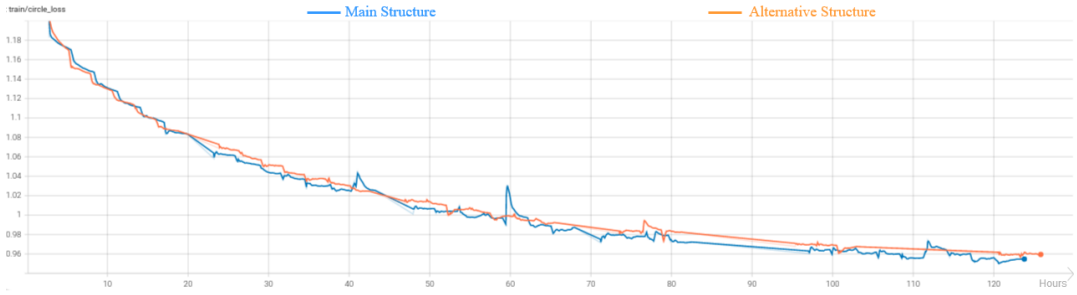
Training and testing of the algorithm were performed using the PyTorch library ([34]) and Minkowski Engine ([9]) on a GeForce RTX 2080 SUPER GPU. The number of epochs was set to 30, with the FPS sample number at 1% with 3 scales each with 10, 15 and 20 neighbours. Following Predator, it was used an initial learning rate of 0.005 with an exponential decay of 0.05 for each epoch. The model was trained from scratch using 3DMatch's dataset that contains 20642 training samples.

As was previously mentioned, the dataset is split in training, validation and testing. The training data is where the parameters training occurs, i.e., where back-propagation optimizes the weights of the network. The validation set is used to evaluate the model during training in order to fine tune the parameters. Finally the testing data is where unused data is evaluated on the final trained model.

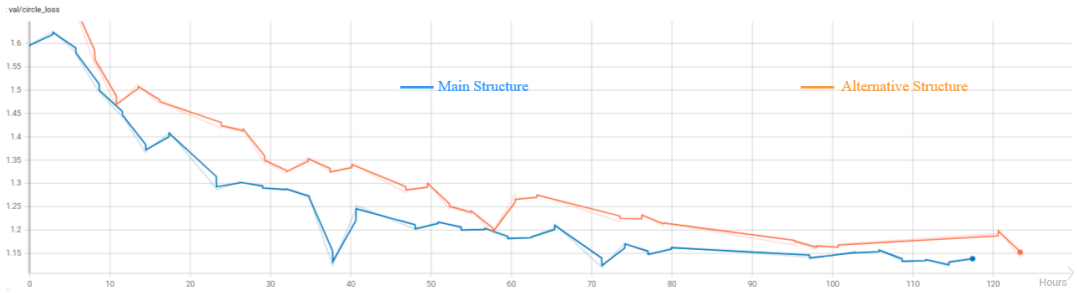
Training was done in the same manner for the two feature extraction structures with both taking approximately the same execution times. From here on out, the structure showed in Figure 3.6 will be denoted as Main Structure and the one showed in Figure 3.7 will be denoted as Alternative Structure.

During the training and validation cycle, the circle and overlap losses as a function of the number of hours of the training are tracked as Figures 4.2 and 4.3 illustrate. As mentioned in Section 3.1, circle loss is used to, just like triplet loss, maximize the distance of non-matching correspondences while minimizing matching correspondences distances. However, circle loss optimizes better in maximizing closer non-matching correspondences and minimizing the furthest matching correspondences [46]. Overlap loss is computed with the negative log likelihood function to estimate overlap probability using the overlap scores. Overall, both in training and in validation the losses are slowly stabilizing, and getting closer to convergence meaning there is overfitting. Main Structure appears to be performing slightly better than the Alternative.

The recall was also tracked during training and validation and is shown in Figure 4.4.

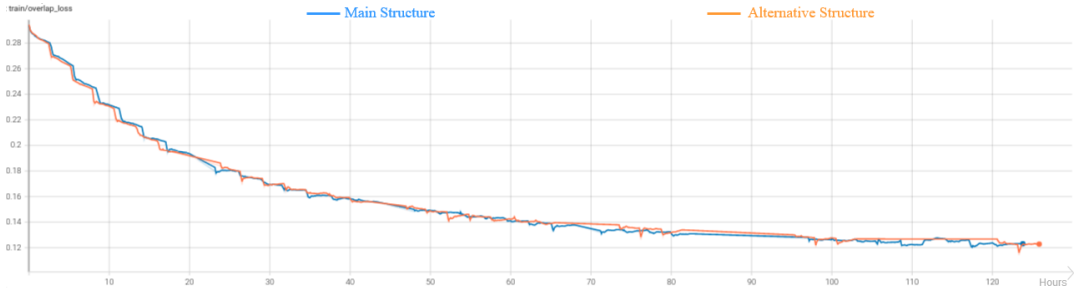


(a) Circle loss during training.

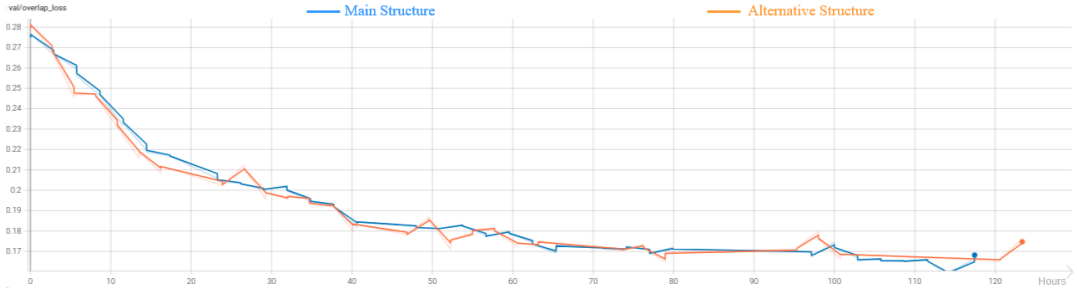


(b) Circle loss during validation.

Figure 4.2: Circle loss of the model with three scales.

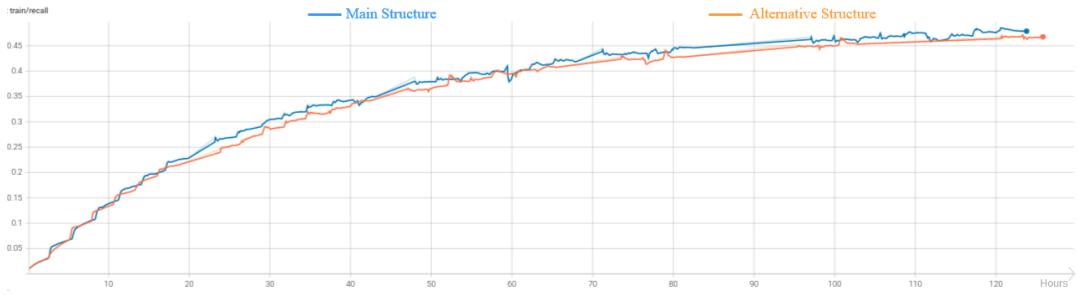


(a) Overlap loss during training.

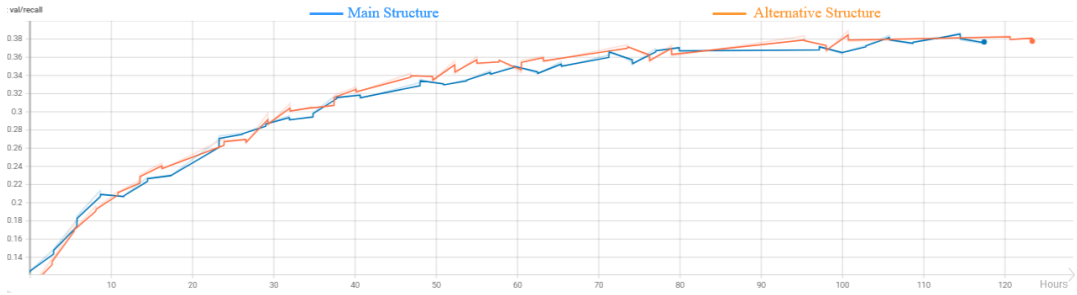


(b) Overlap loss during validation.

Figure 4.3: Overlap loss of the model with three scales.



(a) Recall during training.



(b) Recall during validation.

Figure 4.4: Recall of the model with three scales.

Recall tests the network's ability to find true positives and is defined as follows,

$$recall = \frac{t_P}{t_P + f_N}, \tag{4.4}$$

where t_P is the true positives and f_N the false negatives. In this case, the true positives refer to the points that were correctly paired according to a ground truth prediction. As the number of steps increase so does the recall meaning that the network is increasing its power to identify correct matches.

4.3 Results and Discussion

Using Predator's benchmark evaluation script, the transformation parameters can now be estimated by using RANSAC [17]. The features and scores obtained during testing will also serve to compute the IR, the FMR and the RR.

In Table 4.1 it is shown the comparison of the metrics of Predator and the proposed model with the two structures.

	FMR(%)	IR(%)	RR(%)
Predator Baseline	96.4	57.5	87.7
3 scales (Main)	**96.6**	**58.5**	**89.0**
3 scales (Alternative)	96.3	**57.6**	87.6

Table 4.1: Comparison of the evaluation metrics for the models with different structures.

As evidenced by both the testing results and the training losses, injecting the features directly into the raw pointcloud, as is done with the Alternative Structure, does not bring much improvement. It can be seen that with the Main Structure three scales model, there is an increase in all the evaluation metrics compared to the Minkowski Engine version of Predator which was used as the baseline for this work. Increases in FMR and IR confirms the positive influence of the scales in the quality of the features extracted. At the same time, an increase to RR shows the improvement that adding the scales had on correspondence matching. Meaning that the increase in the quality of features happened mainly to features that had not yet been optimized to minimize the matching distance to find more correspondences. Scales combining the features by themselves first and only after with the raw pointcloud, as was done in the Main Structure, passes the distinct and salient information more effectively.

4.4 Ablation Studies

To have a better understanding about the performance of the approach, tests were done by tweaking scaling parameters and observing their impact on the Main Structure. First, the number of samples obtained from FPS was doubled to 2% while leaving the number of neighbours unchanged. And secondly, the neighbours' values were tripled while leaving the number of samples unchanged at 1%. The effect that these changes have on the training time compared to the Main Structure with three scales model can be seen in Table 4.2, where in the left column it shown how it affects for a whole epoch and in the right column the impact on each individual step.

	Epoch runtime (min)	Step runtime (s)
3 scale	162	0.470
2% samples	231	0.669
Neighbours×3	180	0.523

Table 4.2: Average execution time for tests with varying scaling parameters.

It is observed, in Table 4.3, the difference in the evaluating metrics between the models with differing scaling parameters and the baseline Predator with the Minkowski Engine version.

	FMR(%)	IR(%)	RR(%)
Predator Baseline	96.4	57.5	87.7
2% samples	**96.5**	**59.8**	87.7
Neighbours ×3	96.2	**58.8**	87.0

Table 4.3: Comparison of the evaluation metrics for the models with varying scaling parameters.

To study the influence of removing scales of the network in the model, new models were trained where scales were progressively taken out from the encoder in the Main Structure. The first removal was the scale containing 10 neighbours, leaving two scales as seen in Figure 4.5a, and the second removal took out the scale containing 15 neighbours, leaving only one scale as is shown in Figure 4.5b.

The influence that each scale has in the runtime of the network training is shown in Table 4.4.

	Epoch runtime (min)	Step runtime (s)
1 scale	141	0.410
2 scales	156	0.453
3 scales	162	0.470

Table 4.4: Average execution time that each scale adds up to the training.

Compared to the influence that the number of samples has in the runtime (as was seen in Table 4.2) removing or adding scales does not make the network training time much slower.

Table 4.5 shows the results obtained along with baseline Predator's results.

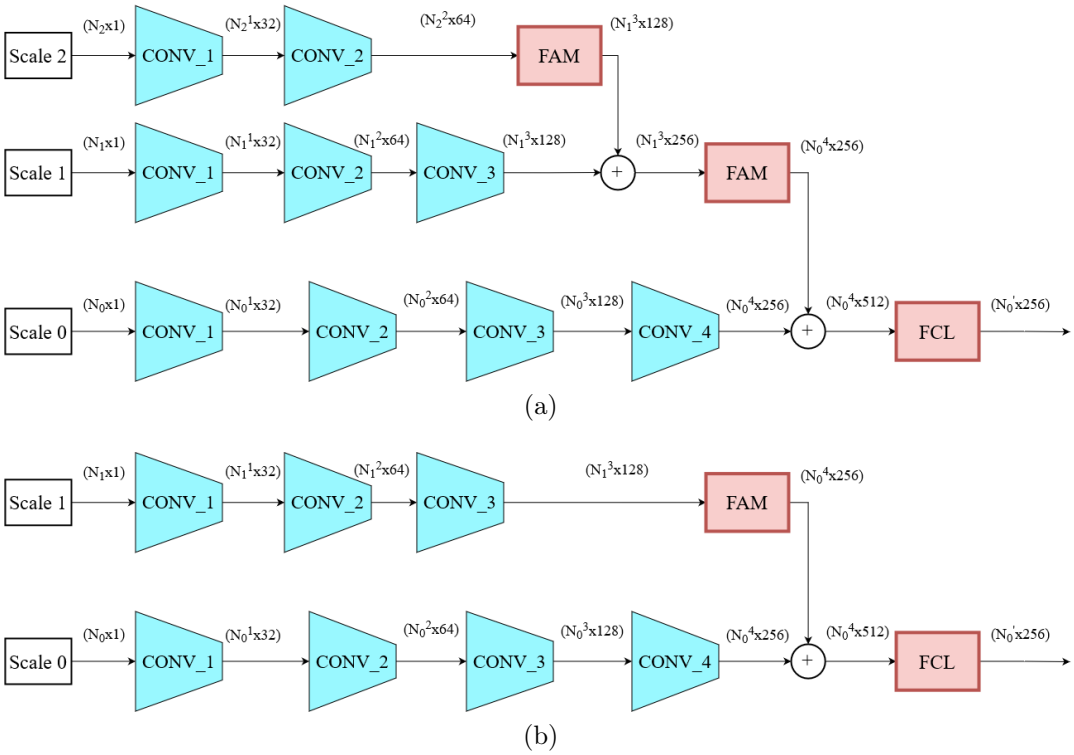


Figure 4.5: Encoder hierarchy structure with 2 scales (a), and 1 scale (b).

	FMR(%)	IR(%)	RR(%)
Predator Baseline	96.4	57.5	87.7
1 scale	**96.8**	**61.1**	87.6
2 scales	96.0	**58.2**	87.5
3 scales (Main)	**96.6**	**58.5**	**89.0**

Table 4.5: Comparison of the evaluation metrics for each model.

Going through each evaluation metric one by one, and starting with FMR, the highest value was obtained by the model with one scale, followed by the model with three scales and the model with 2% of samples. One can infer that with these models there exists slightly more inlier correspondences being detected than with baseline Predator which means the information that is introduced with added scales is relevant and useful.

The IR is superior in all of the models compared with the baseline with the one scale model boasting a 6.26% increase. This means that features that contain multiple scales share a closer distance in the feature space to their corresponding pointcloud pair. It

can be concluded that inserting features with extra contextual information increases the chances of matching more pairs of points despite the Alternative Structure's IR being only marginally higher.

As the RR actually evaluates the registration of the pointclouds, the models with one scale, two scales, triple neighbours and Alternative Structure have performed worse when compared to the baseline. This means that despite having higher quality features, which is the case with the one scale model, RANSAC did not manage to better estimate the transformation parameters. Except for the 2% samples model that did not increase nor decrease and for the model with 3 scales where there is a 1.48% increase compared to the baseline.

It can be concluded that just by having a minimum of one scale integrated there is an increase in the inlier ratio meaning there is important information being introduced by the scales that helps improve matchability. However, the model with two scales performs comparatively worse in all metrics compared to the one scale and three scales models. By tweaking the scaling parameters, it was attempted to observe the effects of adding more points into the scales. Doubling the number of samples used by FPS and tripling the number of neighbours manages to increase the matchability of the features but they are not necessarily high quality features as indicated by the other metrics.

Chapter 5

Conclusion

The aim of this work was to produce an explicit multiscale pointcloud registration problem using a deep neural network, that can use the features extracted at these scales to introduce context. This helps so that there can be a better understanding of surrounding objects in pointclouds, improving registration. A multiscale sampling and grouping pre-processing step was developed to obtain solid and strong scales, two feature extraction hierarchical structures were created and a feature aggregation module was implemented to inject them into the network. The experimental setup consisted of training distinct models with different scaling parameters, models where each scale gets progressively removed and models with a different features extraction structure. It was proved that scales can increase context and enrich features so they are more easily matched. The goal was not only to optimize registration but to also understand if scales with context defined explicitly can be an asset to improving feature based correspondence which was also achieved with the model that contains three scales.

Although there was an improvement on the results, the multiscale pre-processing is limited by the delay it causes to the total training time. A new sampling method could be developed that consumes less computational resources making the training process more efficient and faster. Further work can be done to improve this approach, such as:

- Adding/removing scales with the different permutations of the scaling parameters;
- Aggregate the features in the decoder instead of the encoder, in order to better preserve the multiscale influence;
- Experiment with new datasets that contain a smaller amount of points.

Bibliography

[1] Y. Aoki, H. Goforth, R. A. Srivatsan, and S. Lucey. PointnetLK: Robust & efficient point cloud registration using Pointnet. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2019. 17

[2] X. Bai, Z. Luo, L. Zhou, H. Fu, L. Quan, and C.-L. Tai. 3Dfeat: Joint learning of dense detection and description of 3d local features. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2020. 16, 17

[3] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. Speeded-up robust features (surf). In *Computer Vision and Image Understanding (CVIU)*, pages 346–359, 2008. 18

[4] P. Besl and H. McKay. A method for registration of 3-D shapes. In *IEEE Trans. Pattern Analysis and Machine Intelligence (T-PAMI)*, pages 239–256, 1992. 13

[5] S. D. Billings, E. M. Boctor, and R. H. Taylor. Iterative most-likely point registration (imlp): a robust algorithm for computing optimal shape alignment. In *PloS one, 10(3)*, 2015. 13

[6] P. Buyssens, A. Elmoataz, and O. Lezoray. Multiscale convolutional neural networks for vision–based classification of cells. In *Asian Conf. Computer Vision (ACCV)*, volume 7725, 2012. 2, 18

[7] S. Chen, B. Liu, C. Feng, C. Vallespi-Gonzalez, and C. Wellington. 3d point cloud processing and learning for autonomous driving: Impacting map creation, localization, and perception. *IEEE Acoustics, Speech, and Signal Processing Newsletter*, 38(1):68–86, 2021. Publisher Copyright: © 1991-2012 IEEE. 2

[8] S. Choi, Q.-Y. Zhou, and V. Koltun. Robust reconstruction of indoor scenes. In *CVPR*, pages 5556–5565. IEEE Computer Society, 2015. 36

[9] C. Choy, J. Gwak, and S. Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019. 24, 37

[10] C. Choy, J. Park, and V. Koltun. Fully convolutional geometric features. In *IEEE Int'l Conf. Computer Vision (ICCV)*, 2019. 1, 16, 35

[11] H. Deng, T. Birdal, and S. Ilic. PpfNet: Global context aware local features for robust 3d point matching. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018. xiii, 2, 3, 16, 35

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. 1, 6, 10

[13] B. Drost, M. Ulrich, N. Navab, , and S. Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 998–1005, 2010. 16

[14] I. C. Duta, L. Liu, F. Zhu, and L. Shao. Pyramidal convolution: Rethinking convolutional neural networks for visual recognition. *arXiv preprint arXiv:2006.11538*, 2020. 19

[15] G. Elbaz, T. Avraham, and A. Fischer. 3d point cloud registration for localization using a deep neural network auto-encoder. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2472–2481, 2017. 2

[16] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 1997. 27

[17] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. In *Commun. ACM*, 1981. 15, 39

[18] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybernetics 36, 193–202*, 1980. 5

[19] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, Reading, 1994. 55

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition,. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, June 2016. 12

[21] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-scale dense networks for resource efficient image classification. In *Int'l Conf. Learning Representations (ICLR)*, 2018. 2, 19

[22] S. Huang, Z. Gojcic, M. Usvyatsov, and A. Wieser. Predator: Registration of 3d point clouds with low overlap. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2021. xiii, 1, 3, 23, 24, 35, 36

[23] A. E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. In *IEEE Trans. Pattern Analysis and Machine Intelligence (T-PAMI)*, pages 433–449, 1999. xiii, 13, 14

[24] M. Khoury, Q.-Y. Zhou, and V. Koltun. Learning compact geometric features. In *IEEE Int'l Conf. Computer Vision (ICCV)*, 2017. 15

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012. xiii, 1, 6, 10, 11

[26] S. Li, Y. Liu, X. Sui, C. Chen, G. Tjio, D. Ting, and R. Goh. Multi-instance multi-scale cnn for medical image classification. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 531–539, 2019. xiii, 19

[27] J. Long, E. Shelhamer, , and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2015. 16

[28] D. G. Lowe. Distinctive image features from scale-invariant keypoints. In *Int'l J. Computer Vision (IJCV)*, pages 91–110, 2004. 18

[29] W. Lu, G. Wan, Y. Zhou, X. Fu, P. Yuan, and S. Song. Deepvcp: An end-to-end deep neural network for point cloud registration. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2019. 17

[30] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, 1981. 17

[31] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1999. 9

[32] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *In International Conference on Computer Vision Theory and Applications (VISAPP'09)*, 2009. 29

[33] S. A. Papert. The summer vision project. *MIT AI Memos (1959 - 2004)*, 1966. 5

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 37

[35] C. Qi, L. Yi, H. Su, and L. Guibas. Pointnet++: Deep hierarchical feature learning on pointsets in a metric space. In *Advances in Neural Information Processing Systems (NIPS)*, 2017. 17, 19, 27

[36] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, 2017. xiii, 1, 16, 17, 19

[37] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001. 13

[38] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3D registration. In *IEEE Int'l Conf. Robotics and Automation (ICRA)*, 2009. 14

[39] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. In *IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, 2008. 13, 14

[40] C. S., H. R., and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546 vol. 1, 2005. 15

[41] S. Salti, F. Tombari, , and L. di Stefano. SHOT: Unique signatures of histograms for surface and texture description. In *Computer Vision and Image Understanding (CVIU)*, page 125, 2009. 13

[42] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich. Superglue: Learning feature matching with graph neural networks. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2020. 18

[43] P. Schönemann. A generalized solution of the orthogonal procrustes problem. In *Psychometrika*, pages 31:1–10, 2020. 13

[44] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *In Robotics: science and systems, volume 2*, page 435, 2009. 13

[45] J. Sun, Z. Shen, Y. Wang, H. Bao, and X. Zhou. LoFTR: Detector-free local feature matching with transformers. *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2021. 2, 18

[46] Y. Sun, C. Cheng, Y. Zhang, C. Zhang, L. Zheng, Z. Wang, and Y. Wei1. Circle loss: A unified perspective of pair similarity optimization. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2020. 26, 37

[47] R. Takimoto, M. Tsuzuki, R. Vogelaar, T. Martins, A. Sato, Y. Iwao, T. Gotoh, and S. Kagei. 3d reconstruction and multiple point cloud registration using a low precision rgb-d sensor. *Mechatronics*, 35(C):11–22, 2016. 2

[48] A. Tao, K. Sapra, and B. Catanzaro. Hierarchical multi-scale attention for semantic segmentation. In *arXiv preprint arXiv:2005.10821*, 2020. xiii, 20, 30

[49] H. Thomas, J.-E. Deschaud, B. Marcotegui, F. Goulette, and Y. L. Gall. Semantic classification of 3d point clouds with multiscale spherical neighborhoods. *2018 International Conference on 3D Vision (3DV)*, pages 390–398, 2018. 2, 19

[50] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012. 2

[51] F. Tombari, S. Salti, and L. D. Stefano. Unique shape context for 3d data description. In *ACM Workshop on 3D Object Retrieval*, 2010. 14

[52] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Analysis and Machine Intelligence (T-PAMI)*, 13(4):376–380, 1991. 13

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin. Attention is all you need. In *European Conf. Computer Vision (ECCV)*, pages 126–142, 2020. 18, 25

[54] J. Wang, D. Ding, Z. Li, and Z. Ma. Multiscale point cloud geometry compression. In *IEEE DCC*, 2020. 20

[55] L. Wang, Y. Huang, J. Shan, and L. He. Msnet: Multi-scale convolutional network for point cloud classification. *Remote Sensing*, 10:612, 04 2018. 20

[56] H. Xuan, A. Stylianou, X. Liu, and R. Pless. Hard negative examples are hard, but useful. In *European Conf. Computer Vision (ECCV)*, pages 126–142, 2020. 17

[57] Z. Yao, J. Jia, and Y. Qian. Mcnet: Multi-scale feature extraction and content-aware reassembly cloud detection model for remote sensing images. *Symmetry*, 13(1), 2021. 19

[58] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3DMatch: learning local geometric descriptors from RGB-D reconstructions. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2017. xiii, 1, 15, 35, 36

[59] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021. xiii, 10, 11, 12

[60] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 29

[61] W. Zhu, Y. Huang, D. Xu, Z. Qian, W. Fan, and X. Xie. Test-time training for deformable multi-scale image registration. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13618–13625, 2021. 2

Appendix A

Iverson Bracket

For a statement P that can be true or false, the Iverson bracket [19] is defined as follows,

$$[P] = \begin{cases} 1 & \text{if } P \text{ is True,} \\ 0 & \text{otherwise.} \end{cases} \tag{A.1}$$