PointNetVLAD: Deep Point Cloud Based Retrieval for Large-Scale Place Recognition

Mikaela Angelina Uy Gim Hee Lee
Department of Computer Science, National University of Singapore
{mikacuy,gimhee.lee}@comp.nus.edu.sg

Abstract

Unlike its image based counterpart, point cloud based retrieval for place recognition has remained as an unexplored and unsolved problem. This is largely due to the difficulty in extracting local feature descriptors from a point cloud that can subsequently be encoded into a global descriptor for the retrieval task. In this paper, we propose the PointNetVLAD where we leverage on the recent success of deep networks to solve point cloud based retrieval for place recognition. Specifically, our PointNetVLAD is a combination/modification of the existing PointNet and NetVLAD, which allows end-to-end training and inference to extract the global descriptor from a given 3D point cloud. Furthermore, we propose the "lazy triplet and quadruplet" loss functions that can achieve more discriminative and generalizable global descriptors to tackle the retrieval task. We create benchmark datasets for point cloud based retrieval for place recognition, and the experimental results on these datasets show the feasibility of our PointNetVLAD. Our code and datasets are publicly available on the project website [1].

1. Introduction

Localization addresses the question of "where am I in a given reference map", and it is of paramount importance for robots such as self-driving cars [12] and drones [10] to achieve full autonomy. A common method for the localization problem is to first store a map of the environment as a database of 3D point cloud built from a collection of images with Structure-from-Motion (SfM) [14], or LiDAR scans with Simultaneous Localization and Mapping (SLAM) [38]. Given a query image or LiDAR scan of a local scene, we then search through the database to retrieve the best match that will tell us the exact pose of the query image/scan with respect to the reference map.

A two-step approach is commonly used in image based localization [30, 29, 31, 43] - (1) place recognition [8, 7, 21, 40, 11], followed by (2) pose estimation [13]. In place recognition, a global descriptor is computed for each of the images used in SfM by aggregating local image descriptors, e.g. SIFT, using the bag-of-words approach [22, 35]. Each global descriptor is stored in the database together with the camera pose of its associated image with respect to the 3D point cloud reference map. Similar global descriptor is extracted from the query image and the closest global descriptor in the database can be retrieved via an efficient search. The camera pose of the closest global descriptor would give us a coarse localization of the query image with respect to the reference map. In pose estimation, we compute the exact pose of the query image with the Perspective-n-Point (PnP) [13] and geometric verification [18] algorithms.

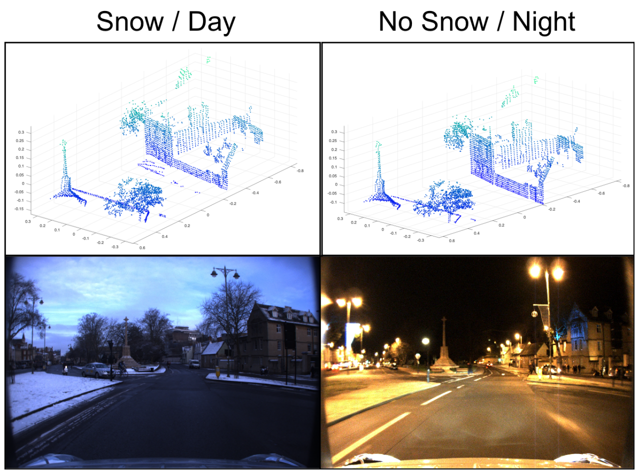

Figure 1. Two pairs of 3D LiDAR point clouds (top row) and images (bottom row) taken from two different times. It can be seen that the pair of 3D LiDAR point cloud remain largely invariant to the lighting and seasonal changes that made it difficult to match the pair of images. Data from [20].

The success of image based place recognition is largely attributed to the ability to extract image feature descriptors e.g. SIFT, that are subsequently aggregated with bag-of-words to get the global descriptor. Unfortunately, there is no algorithm to extract local features similar to SIFT for LiDAR scans. Hence, it becomes impossible to compute global descriptors from the bag-of-word approach to do Li-

[1] https://github.com/mikacuy/pointnetvlad.git

DAR based place recognition. Most existing approaches circumvent this problem by using readings from the Global Positioning System (GPS) to provide coarse localization, followed by point cloud registration, e.g. the iterative closest point (ICP) [33] or autoencoder based registration [9], for pose-estimation. As a result, LiDAR based localization is largely neglected since GPS might not be always available, despite the fact that much more accurate localization results can be obtained from LiDAR compared to images due to the availability of precise depth information. Furthermore, in comparison to images, the geometric information from LiDARs are invariant to drastic lighting changes, thus making it more robust to perform localization on queries and databases taken from different times of the day, e.g. day and night, and/or different seasons of the year. Fig. 1 shows an example of a pair of 3D LiDAR point clouds and images that are taken from the same scene over two different times (daytime in winter on the left column, and nighttime in fall on the right column). It is obvious that the lighting (day and night) and seasonal (with and without snow) changes made it difficult even for human eye to tell that the pair of images (bottom row) are from the same scene. In contrast, the geometric structures of the LiDAR point cloud remain largely unchanged.

In view of the potential that LiDAR point clouds could be better in the localization task, we propose the PointNetVLAD - a deep network for large-scale 3D point cloud retrieval to fill in the gap of place recognition in the 3D point cloud based localization. Specifically, our PointNetVLAD is a combination of the existing PointNet [23] and NetVLAD [2], which allows end-to-end training and inference to extract the global descriptor from a given 3D point cloud. We provide the proof that NetVLAD is a symmetric function, which is essential for our PointNetVLAD to achieve permutation invariance on the 3D point cloud input. We apply metric learning [6] to train our PointNetVLAD to effectively learn a mapping function that maps input 3D point clouds to discriminative global descriptors. Additionally, we propose the "lazy triplet and quadruplet" loss functions that achieve more generalizable global descriptors by maximizing the differences between all training examples from their respective hardest negative. We create benchmark datasets for point cloud based retrieval for place recognition based on the open-source Oxford RobotCar dataset [20] and three additional datasets collected from three different areas with a Velodyne-64 LiDAR mounted on a car. Experimental results on the benchmark datasets verify the feasibility of our PointNetVLAD.

2. Related Work

Unlike the maturity of handcrafted local feature extraction for 2D images [19, 4], no similar methods proposed for 3D point cloud have reached the same level of maturity. In NARF [36], Steder $et.\ al.$ proposed an interest point extraction algorithm for object recognition. In SHOT [39], Tombari $et.\ al.$ suggested a method to extract 3D descriptors for surface matching. However, both [36, 39] rely on stable surfaces for descriptor calculation and are more suitable for dense rigid objects from 3D range images but not for outdoor LiDAR scans. A point-wise histogram based descriptor - FPFH was proposed in [26, 27] for registration. It works on outdoor 3D data but requires high data density, thus making it not scalable to large-scale environments.

In the recent years, handcrafted features have been increasingly replaced by deep networks that have shown amazing performances. The success of deep learning has been particularly noticeable on 2D images where convolution kernels can be easily applied to the regular 2D lattice grid structure of the image. However, it is more challenging for convolution kernels to work on 3D points that are orderless. Several deep networks attempt to mitigate this challenge by transforming point cloud inputs into regular 3D volumetric representations. Some of these works include: 3D ShapeNets [42] for recognition, volumetric CNNs [24] and OctNet [25] for classification. Additionally, 3DMatch [44] learns local descriptors for small-scale indoor scenes and Vote3D [41] is used for object detection on the outdoor KITTI dataset. Instead of volumetric representation, MVCNN [37] projects the 3D point cloud into 2D image planes across multiple views to solve the shape recognition problem. Unfortunately, volumetric representations and 2D projections based deep networks that work well on object and small-scale indoor levels do not scale well for our large-scale outdoor place recognition problem.

It is not until the recent PointNet [23] that made it possible for direct input of 3D point cloud. The key to its success is the symmetric max pooling function that enables the aggregation of local point features into a latent representation which is invariant to the permutation of the input points. PointNet focuses on the classification task: shape classification and per-point classification (i.e. part segmentation, scene semantic parsing) on rigid objects and enclosed indoor scenes. PointNet is however not shown to do large-scale point cloud based place recognition. Kd-network [16] also works for unordered point cloud inputs by transforming them into kd-trees. However, it is non-invariant/partially-invariant to rotation/noise that are both present in large-scale outdoor LiDAR point clouds.

In [2], Arandjelović $et.\ al.$ proposed the NetVLAD - a deep network that models after the successful bag-of-words approach VLAD [15, 3]. The NetVLAD is an end-to-end deep network made up of the VGG/Alexnet [34, 17] for local feature extraction, followed by the NetVLAD aggregation layer for clustering the local features into VLAD global descriptor. NetVLAD is trained on images obtained from the Google Street View Time Machine, a database consist-

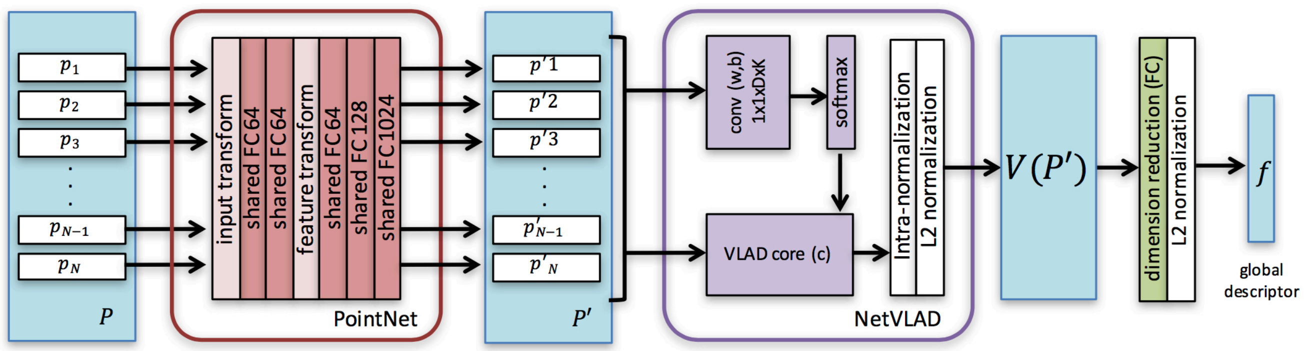

Figure 2. Network architecture of our PointNetVLAD.

ing of multiple instances of places taken at different times, to perform the image based place recognition tasks. Results in [2] show that using the NetVLAD layer significantly outperformed the original non-deep learning based VLAD and its deep learning based max pooling counterpart. Despite the success of NetVLAD for image retrieval, it does not work for our task of point cloud based retrieval since it is not designed to take 3D points as input.

Our PointNetVLAD leverages on the success of PointNet [23] and NetVLAD [2] to do 3D point cloud based retrieval for large-scale place recognition. Specifically, we show that our PointNetVLAD, which is a combination/modification of the PointNet and NetVLAD, originally used for point based classification and image retrieval respectively, is capable of doing end-to-end 3D point cloud based place recognition.

3. Problem Definition

Let us denote the reference map $\mathcal{M}$ as a database of 3D points defined with respect to a fixed reference frame. We further define that the reference map $\mathcal{M}$ is divided into a collection of M submaps $\{\mathsf{m}_1, ..., \mathsf{m}_M\}$ such that $\mathcal{M} = \bigcup_{i=1}^{M} \mathsf{m}_i$. The area of coverage ($AOC$) of all submaps are made to be approximately the same, i.e. $AOC(\mathsf{m}_1) \approx ... \ AOC(\mathsf{m}_M)$, and the number of points in each submap is kept small, i.e. $|\mathsf{m}_i| \ll |\mathcal{M}|$. We apply a downsampling filter $\mathcal{G}(.)$ to ensure that the number of points of all downsampled submaps are the same, i.e. $|\mathcal{G}(\mathsf{m}_1)| = ... \ |\mathcal{G}(\mathsf{m}_M)|$. The problem of large-scale 3D point cloud based retrieval can be formally defined as follows:

Definition 1 *Given a query 3D point cloud denoted as* q, *where* $AOC(\mathsf{q}) \approx AOC(\mathsf{m}_i)$ *and* $|\mathcal{G}(\mathsf{q})| = |\mathcal{G}(\mathsf{m}_i)|$, *our goal is to retrieve the submap* m_* *from the database* $\mathcal{M}$ *that is structurally most similar to* q.

Towards this goal, we design a deep network to learn a function $f(.)$ that maps a given downsampled 3D point cloud $\bar{\mathsf{p}} = \mathcal{G}(\mathsf{p})$, where $AOC(\mathsf{p}) \approx AOC(\mathsf{m}_i)$, to a fixed size global descriptor vector $f(\bar{\mathsf{p}})$ such that $d(f(\bar{\mathsf{p}}), f(\bar{\mathsf{p}}_r)) < d(f(\bar{\mathsf{p}}), f(\bar{\mathsf{p}}_s))$, if p is structurally similar to p_r but dissimilar to p_s. $d(.)$ is some distance function, e.g. Euclidean distance function. Our problem then simplifies to finding the submap $\mathsf{m}_* \in \mathcal{M}$ such that its global descriptor vector $f(\bar{\mathsf{m}}_*)$ gives the minimum distance with the global descriptor vector $f(\bar{\mathsf{q}})$ from the query q, i.e. $d(f(\bar{\mathsf{q}}), f(\bar{\mathsf{m}}_*)) < d(f(\bar{\mathsf{q}}), f(\bar{\mathsf{m}}_i)), \forall i \neq *$. In practice, this can be done efficiently by a simple nearest neighbor search through a list of global descriptors $\{f(\bar{\mathsf{m}}_i) \mid i \in 1, 2, .., M\}$ that can be computed once offline and stored in memory, while $f(\bar{\mathsf{q}})$ is computed online.

4. Our PointNetVLAD

In this section, we will describe the network architecture of PointNetVLAD and the loss functions that we designed to learn the function $f(.)$ that maps a downsampled 3D point cloud to a global descriptor. We also show the proof that the NetVLAD layer is permutation invariant, thus suitable for 3D point cloud.

4.1. The Network Architecture

Fig. 2 shows the network architecture of our PointNetVLAD, which is made up of three main components - (1) PointNet [23], (2) NetVLAD [2] and (3) a fully connected network. Specifically, we take the first part of PointNet, cropped just before the maxpool aggregation layer. The input to our network is the same as PointNet, which is a point cloud made up of a set of 3D points, $P = \{p_1, ..., p_N \mid p_n \in \mathbb{R}^3\}$. Here, we denote P as a fixed size point cloud after applying the filter $\mathcal{G}(.)$; we drop the bar notation on P for brevity. The role of PointNet is to map each point in the input point cloud into a higher dimensional space, i.e. $P = \{p_1, ..., p_N \mid p_n \in \mathbb{R}^3\} \longmapsto P' = \{p'_1, ..., p'_N \mid p'_n \in \mathbb{R}^D\}$, where $D \gg 3$. Here, PointNet can be seen as the component that learns to extract a D-dimensional local feature descriptor from each of the input 3D points.

We feed the output local feature descriptors from Point-

Net as input to the NetVLAD layer. The NetVLAD layer is originally designed to aggregate local image features learned from VGG/AlexNet into the VLAD bag-of-words global descriptor vector. By feeding the local feature descriptors of a point cloud into the layer, we create a machinery that generates the global descriptor vector for an input point cloud. The NetVLAD layer learns K cluster centers, i.e. the visual words, denoted as $\{c_1, ..., c_K \mid c_k \in \mathbb{R}^D\}$, and outputs a $(D \times K)$-dimensional vector $V(P')$. The output vector $V(P') = [V_1(P'), ..., V_K(P')]$ is an aggregated representation of the local feature vectors, where $V_k(P') \in \mathbb{R}^D$ is given by:

$$V_k(P') = \sum_{i=1}^{n} \frac{e^{w_k^T p'_i + b_k}}{\sum_{k'} e^{w_{k'}^T p'_i + b_{k'}}} (p'_i - c_k). \qquad (1)$$

$\{w_k\}$ and $\{b_k\}$ are the weights and biases that determine the contribution of local feature vector p'_i to $V_k(p')$. All the weight and bias terms are learned during training.

The output from the NetVLAD layer is the VLAD descriptor [15, 3] for the input point cloud. However, the VLAD descriptor is a high dimensional vector, i.e. $(D \times K)$-dimensional vector, that makes it computationally expensive for nearest neighbor search. To alleviate this problem, we use a fully connected layer to compress the $(D \times K)$ vector into a compact output feature vector, which is then L2-normalized to produce the final global descriptor vector $f(P) \in \mathbb{R}^{\mathcal{O}}$, where $\mathcal{O} \ll (D \times K)$, for point cloud P that can be used for efficient retrieval.

4.2. Metric Learning

We train our PointNetVLAD end-to-end to learn the function $f(.)$ that maps an input point cloud P to a discriminative compact global descriptor vector $f(P) \in \mathbb{R}^{\mathcal{O}}$, where $\|f(P)\|_2 = 1$. To this end, we propose the "Lazy Triplet" and "Lazy Quadruplet" losses that can learn discriminative and generalizable global descriptors. We obtain a set of training tuples from the training dataset, where each tuple is denoted as $\mathcal{T} = (P_a, P_{pos}, \{P_{neg}\})$. P_a, P_{pos} and $\{P_{neg}\}$ denote an anchor point cloud, a structurally similar ("positive") point cloud to the anchor and a set of structurally dissimilar ("negative") point clouds to the anchor, respectively. The loss functions are designed to minimize the distance between the global descriptor vectors of P_a and P_{pos}, i.e. $\delta_{pos} = d(f(P_a), f(P_{pos}))$, and maximize the distance between the global descriptor vectors of P_a and some $P_{neg_j} \in \{P_{neg}\}$, i.e. $\delta_{neg_j} = d(f(P_a), f(P_{neg_j}))$. $d(.)$ is a predefined distance function, which we take to be the squared Euclidean distance in this work.

Lazy triplet: For each training tuple $\mathcal{T}$, our lazy triplet loss focuses on maximizing the distance between $f(P_a)$ and the global descriptor vector of the closest/hardest negative in $\{P_{neg}\}$, denoted as $f(P^-_{neg_j})$. Formally, the lazy triplet loss is defined as

$$\mathcal{L}_{lazyTrip}(\mathcal{T}) = \max_j([\alpha + \delta_{pos} - \delta_{neg_j}]_+), \qquad (2)$$

where $[\ldots]_+$ denotes the hinge loss and α is a constant parameter giving the margin. The max operator selects the closest/hardest negative $P^-_{neg_j}$ in $\{P_{neg}\}$ that gives the smallest δ_{neg_j} value in a particular iteration. Note that $P^-_{neg_j}$ of each training tuple changes because the parameters of the network that determine $f(.)$ get updated during training, hence a different point cloud in $\{P_{neg}\}$ might get mapped to a global descriptor that is nearest to $f(P_a)$ at each iteration. Our choice to iteratively use the closest/hardest negatives over all training tuples ensures that the network learns from all the hardest examples to get a more discriminative and generalizable function $f(.)$.

Lazy quadruplet: The choice to maximize the distance between $f(P_a)$ and $f(P^-_{neg_j})$ might lead to an undesired reduction of the distance between $f(P^-_{neg_j})$ and another point cloud $f(P_{false})$, where P_{false} is structurally dissimilar to $P^-_{neg_j}$. To alleviate this problem, we maximize an additional distance $\delta_{neg^*_k} = d(f(P_{neg^*}), f(P_{neg_k}))$, where P_{neg^*} is randomly sampled from the training dataset at each iteration and is dissimilar to all point clouds in $\mathcal{T}$. The lazy quadruplet loss is defined as

$$\begin{aligned}\mathcal{L}_{lazyQuad}(\mathcal{T}, P_{neg^*}) = &\max_j([\alpha + \delta_{pos} - \delta_{neg_j}]_+) \\ &+ \max_k([\beta + \delta_{pos} - \delta_{neg^*_k}]_+),\end{aligned} \qquad (3)$$

where β is a another constant parameter giving the margin. The max operator of the second term selects the hardest negative $P^-_{neg_k}$ in $\{P_{neg}\}$ that give the smallest δ_{neg_k} value.

Discussion: Original triplet and quadruplet losses use the sum instead of the max operator proposed in our "lazy" variants. These losses have been shown to work well for different applications such as facial recognition [32, 5]. However, maximizing δ_{neg_j} for all $\{P_{neg}\}$ leads to a compounding effect where the contribution of each negative training data diminishes as compared to the contribution from a single hardest negative training data. As a result, the original triplet and quadruplet losses tend to take longer to train, and lead to a less discriminative function $f(.)$ that produces inaccurate retrieval results. Experimental results indeed show that both our "lazy" variants outperform the original losses by a competitive margin with the lazy quadruplet loss slightly outperforming the lazy triplet loss.

4.3. Permutation Invariance

Unlike its image counterpart, a set of points in a point cloud are unordered. Consequently, a naive design of the

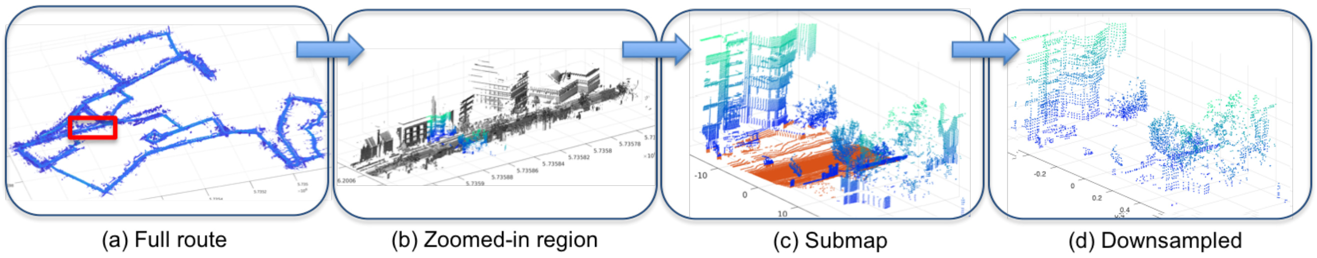

Figure 3. Dataset preprocessing: (a) A full route from the Oxford RobotCar dataset. (b) Zoomed-in region of the 3D point cloud in the red box shown in (a). (c) An example of submap with the detected ground plane shown as red points. (d) A downsampled submap that is centered at origin and all points within [-1,1]m.

network could produce different results from different orderings of the input points. It is therefore necessary for the network to be input order invariant for it to be suitable for point clouds. This means that the network will output the same global descriptor $f(P)$ for point cloud P regardless of the order in which the points in P are arranged. We rigorously show that this property holds for PointNetVLAD.

Given an input point cloud $P = \{p_1, p_2, \ldots, p_N\}$, the layers prior to NetVLAD, i.e. PointNet, transform each point in P independently into $P' = \{p'_1, p'_2, \ldots, p'_N\}$, hence it remains to show that NetVLAD is a symmetric function, which means that its output $V(P')$ would be invariant to the order of the points in P' leading to an output global descriptor $f(P')$ that is order invariant.

Lemma 1 *NetVLAD is a symmetric function*

Proof: Given the feature representation of input point cloud P as $\{p'_1, p'_2, \ldots, p'_N\}$, we have the output vector $V = [V_1, V_2, \ldots, V_K]$ of NetVLAD such that $\forall k$,

$$V_k = h_k(p'_1) + h_k(p'_2) + \ldots + h_k(p'_N) = \sum_{t=1}^{N} h_k(p'_t), \quad (4)$$

where

$$h_k(p') = \frac{e^{w_k^T p' + b_k}}{\sum_{k'} e^{w_{k'}^T p' + b_{k'}}} (p' - c_k). \quad (5)$$

Suppose we have another point cloud $\tilde{P} = \{p_1, \ldots, p_{i-1}, p_j, p_{i+1}, \ldots, p_{j-1}, p_i, p_{j+1}, \ldots, p_N\}$ that is similar to P except for reordered points p_i and p_j. Then the feature representation of $\tilde{P}$ is given by $\{p'_1, \ldots, p'_{i-1}, p'_j, p'_{i+1}, \ldots, p'_{j-1}, p'_i, p'_{j+1}, \ldots, p'_N\}$. Hence $\forall k$, we have

$$\begin{aligned}\tilde{V}_k =& h_k(p'_1) + \ldots + h_k(p'_{i-1}) + \\ & h_k(p'_j) + h_k(p'_{i+1}) + \ldots + h_k(p'_{j-1}) + \\ & h_k(p'_i) + h_k(p'_{j+1}) + \ldots + h_k(p'_N) \\ =& \sum_{t=1}^{N} h_k(p'_t) = V_k.\end{aligned} \quad (6)$$

Thus, $f(P) = f(\tilde{P})$ and completes our proof for symmetry.

5. Experiments

5.1. Benchmark Datasets

We create four benchmark datasets suitable for LiDAR-based place recognition to train and evaluate our network: one from the open-source Oxford RobotCar [20] and three in-house datasets of a university sector (U.S.), a residential area (R.A.) and a business district (B.D.). These are created using a LiDAR sensor mounted on a car that repeatedly drives through each of the four regions at different times traversing a 10km, 10km, 8km and 5km route on each round of Oxford, U.S., R.A. and B.D., respectively. For each run of each region, the collected LiDAR scans are used to build a unique reference map of the region. The reference map is then used to construct a database of submaps that represent unique local areas of the region for each run. Each reference map is built with respect to the UTM coordinate frame using GPS/INS readings.

Submap preprocessing The ground planes are removed in all submaps since they are non-informative and repetitive structures. The resulting point cloud is then downsampled to 4096 points using a voxel grid filter [28]. Next, it is shifted and rescaled to be zero mean and inside the range of [-1, 1]. Each downsampled submap is tagged with a UTM coordinate at its respective centroid, thus allowing supervised training and evaluation of our network. To generate training tuples, we define structurally similar point clouds to be at most 10m apart and those structurally dissimilar to be at least 50m apart. Fig. 3 shows an example of a reference map, submap and downsampled submap.

Data splitting and evaluation We split each run of each region of the datasets into two disjoint reference maps used for training and testing. We further split each reference map into a set of submaps at regular intervals of the trajectory in the reference map. Refer to the supplementary material for more details on data splitting. We obtain a total of 28382 submaps for training and 7572 submaps for test from the Oxford and in-house datasets. To test the performance of our network, we use a submap from a testing reference map as a query point cloud and all submaps from another reference map of a different run that covers the same region as the database. The query submap is successfully localized if it retrieves a point cloud within 25m.

Oxford Dataset We use 44 sets of full and partial runs from the Oxford RobotCar dataset [20], which were collected at different times with a SICK LMS-151 2D LiDAR scanner. Each run is geographically split into 70% and 30% for the training and testing reference maps, respectively. We further split each training and testing reference map into submaps at fixed regular intervals of 10m and 20m, respectively. Each submap includes all 3D points that are within a 20m trajectory of the car. This resulted in 21,711 training submaps, which are used to train our baseline network, and 3030 testing submaps ($\sim$ 120-150 submaps/run).

In-house Datasets The three in-house datasets are constructed from Velodyne-64 LiDAR scans of five different runs of each of the regions U.S., R.A. and B.D. that were collected at different times. These are all used as testing reference maps to test the generalization of our baseline network trained only on Oxford. Furthermore, we also geographically split each run of U.S. and R.A. into training and testing reference maps, which we use for network refinement. Submaps are taken at regular intervals of 12.5m and 25m for each training and testing reference maps, respectively. All 3D points within a 25m$\times$25m bounding box centered at each submap location are taken. Table 1 shows the breakdown on the number of training and testing submaps used in the baseline and refined networks.

	Training$^+$		Test$^\times$	
	Baseline	Refine	Baseline	Refine
Oxford	21711	21711	3030	3030
U.S.	-	} 6671	400*	80*
R.A.	-		320* } 4542	75* } 1766
B.D.	-	-	200*	200*

Table 1. Number of training and testing submaps for our baseline and refined networks. *approximate number of submaps/run is given because the number of submaps differ slightly between each run; $^+$overlapping and $^\times$disjoint submaps.

5.2. Results

We present results to show the feasibility of our PointNetVLAD (PN_VLAD) for large-scale point cloud based place recognition. Additionally, we compare its performance to the original PointNet architecture with the maxpool layer (PN_MAX) and a fully connected layer to produce a global descriptor with output dimension equal to ours; this is also trained end-to-end for the place recognition task. Moreover, we also compare our network with the state-of-the-art PointNet trained for object classification on rigid objects in ModelNet (PN_STD) to investigate whether the model trained on ModelNet can be scaled to large-scale environments. We cut the trained network just before the softmax layer hence producing a 256-dim output vector.

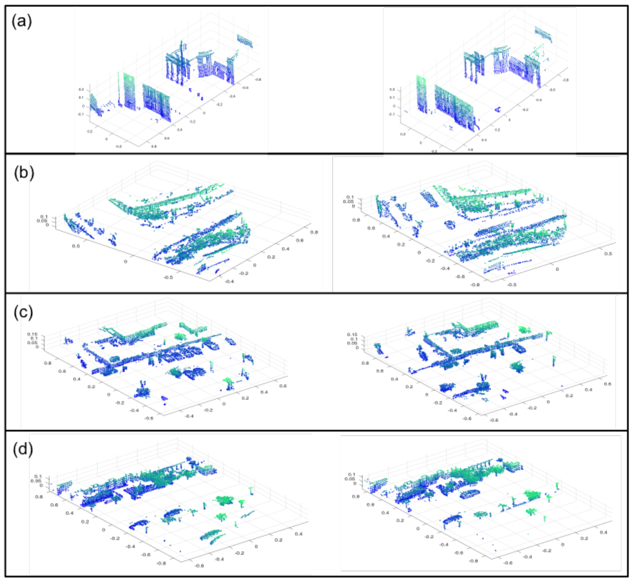

Figure 4. Sample point clouds from (a) Oxford, (b) U.S., (c) R.A. and (d) B.D., respectively: left shows the query submap and right shows the successfully retrieved corresponding point cloud.

	PN_VLAD	PN_MAX	PN_STD
Oxford	**80.31**	73.44	46.52
U.S.	**72.63**	64.64	61.12
R.A.	**60.27**	51.92	49.07
B.D.	**65.30**	54.74	53.02

Table 2. Baseline results showing the average recall (%) at top 1% for each of the models.

Baseline Networks We train the PN_STD, PN_MAX and our PN_VLAD using only the Oxford training dataset. The network configurations of PN_STD and PN_MAX are set to be the same as [23]. The dimension of the output global descriptor of PN_MAX is set to be same as our PN_VLAD, i.e. 256-dim. Both PN_MAX and our PN_VLAD are trained with the lazy quadruplet loss, where we set the margins $\alpha = 0.5$ and $\beta = 0.2$. Furthermore, we set the number of clusters in our PN_VLAD to be $K = 64$. We test the trained networks on Oxford. The Oxford RobotCar dataset is a challenging dataset due to multiple roadworks that caused some scenes to change almost completely. We verify the generalization of our network by testing on completely unseen environments with our in-house datasets. Table 2 shows the top1% recall of the different models on each of the datasets. It can be seen that PN_STD does not generalize well for large scale place retrieval, and PN_MAX does not generalize well to the new environments as compared to our PN_VLAD. Fig. 5 (top row) shows the recall curves of each model for the top 25 matches from each database pair for the four test datasets, where our network outperforms the rest. Note that the recall rate is the average recall rate of all query results from each submap in the test data.

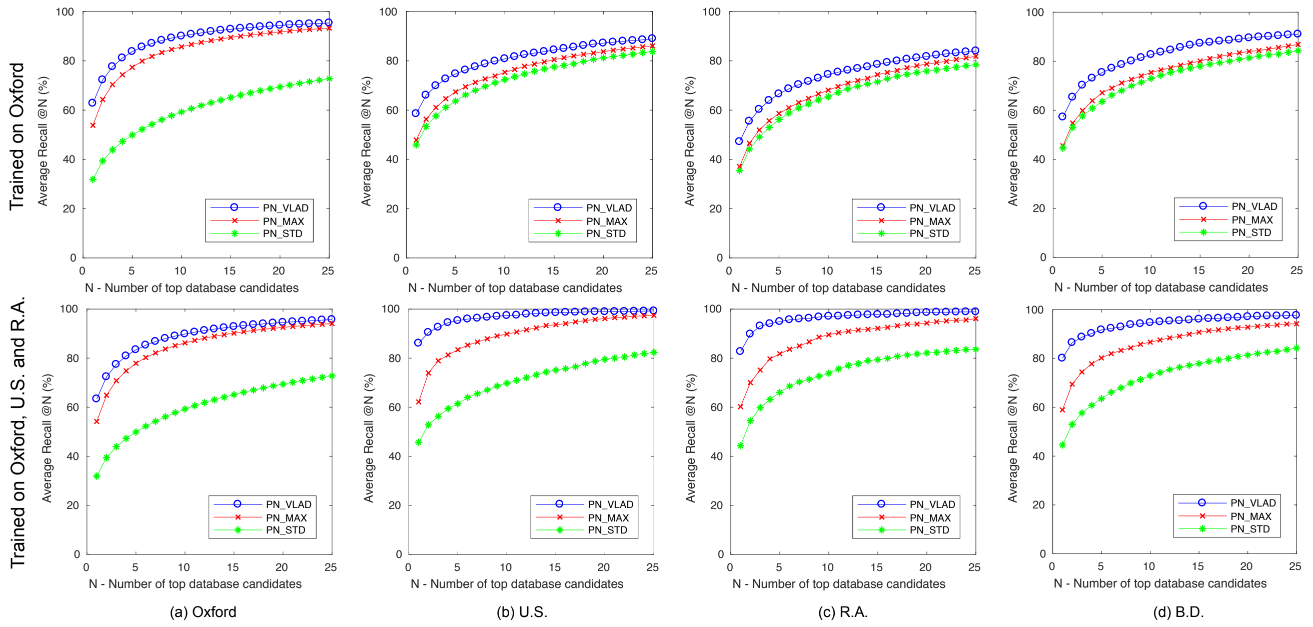

Figure 5. **Average recall of the networks.** Top row shows the average recall when PN_VLAD and PN_MAX were only trained on Oxford. Bottom row shows the average recall when PN_VLAD and PN_MAX were trained on Oxford, U.S. and R.A.

	PN_VLAD			PN_MAX		
	D-128	D-256	D-512	D-128	D-256	D-512
Ox.	**74.60**	80.31	80.33	71.93	73.44	**74.79**
U.S.	**66.03**	72.63	76.24	61.15	64.64	**65.79**
R.A.	**53.86**	60.27	63.31	49.25	51.92	**52.32**
B.D.	**59.84**	65.30	66.75	53.25	54.74	**56.63**

Table 3. Average recall (%) at top1% on the different datasets for output dimensionality analysis of PN_VLAD and PN_MAX. All models were trained on Oxford. Here, D- refers to global descriptors with output length D-dim.

	Average recall
Triplet Loss	71.20
Quadruplet Loss	74.13
Lazy Triplet Loss	**78.99**
Lazy Quadruplet Loss	**80.31**

Table 4. Results representing the average recall (%) at top1% of PN_VLAD tested and trained using different losses on Oxford.

Output dimensionality analysis We study the discriminative ability of our network over different output dimensions of global descriptor f for both our PN_VLAD and PN_MAX. As show in Table 3, the performance of our PN_VLAD with output length of 128-dim is on par with PN_MAX with output length of 512-dim on Oxford, and marginally better on our in-house datasets. The performance of our network increases from the output dimension of 128-dim to 256-dim, but did not increase further from 256-dim to 512-dim. Hence, we chose to use an output global descriptor of 256-dim in most of our experiments.

	Ave recall @1%			Ave recall@1		
	PN_ VLAD	PN_ MAX	PN_ STD	PN_ VLAD	PN_ MAX	PN_ STD
Ox.	80.09	73.87	46.52	63.33	54.16	31.87
U.S.	90.10	79.31	56.95	86.07	62.16	45.67
R.A.	93.07	75.14	59.81	82.66	60.21	44.29
B.D.	**86.49**	69.49	53.02	**80.11**	58.95	44.54

Table 5. Final results showing the average recall (%) at top 1% (@1%) and at top 1 (@1) after training on Oxford, U.S. and R.A.

Comparison between losses We compared our network's performance when trained on different losses. As shown in Table 4, our network performs better when trained on our lazy variants of the losses. Hence we chose to use the lazy quadruplet loss to train our PN_VLAD and PN_MAX.

Network refinement We further trained our network with U.S. and R.A. in addition to Oxford. This improves the generalizability of our network on the unseen data B.D. as can be seen from the last row of Table 5 and second row of Fig. 5-(d). We have shown the feasibility and potential of our PointNetVLAD for LiDAR based place recognition by achieving reasonable results despite the smaller database size compared to established databases for image based place recognition (*e.g.* Google Street View Time Machine and Tokyo 24/7 [40]). We believe that given more publicly available LiDAR datasets suitable for place recognition our network can further improve its performance and bridge the gap of place recognition in LiDAR based localization.

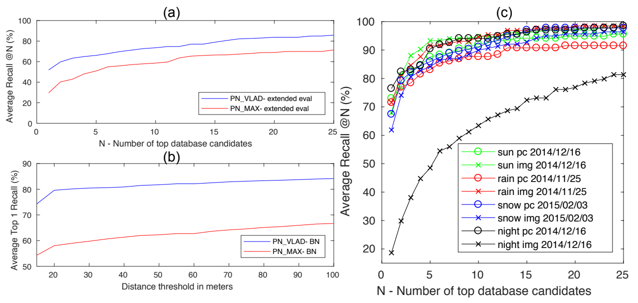

Figure 6. (a) Average recall @N for retrieval from all reference areas. (b) Average recall at B.D. with varying distance thresholds. (c) Average recall @N with point clouds (pc) and images (img) as queries under various scene conditions, and retrieving from an overcast database in Oxford dataset.

Extended evaluation Fig. 6-(a) shows the average recall when queries from Oxford, U.S., R.A. and B.D. are retrieved from an extended database containing all four areas ($\sim$ 33km). Moreover, Fig. 6-(b) shows the top 1 recall on unseen data B.D. with varying distance thresholds. It can be seen that on these extended evaluation metrics, our PN_VLAD still outperforms PN_MAX.

Image based comparisons under changing scene conditions We compare the performance of our point cloud based approach to the image based counterpart. We train NetVLAD according to the specifications specified in [2] with images from the center stereo camera of [20]. These images are taken at the corresponding location of each point cloud submap used to train our PN_VLAD. Fig. 6-(c) shows retrieval results when query was taken from various scene conditions against an overcast database in the Oxford dataset. The performance of image based NetVLAD is comparable to our point cloud based PN_VLAD in all cases, except for overcast (day) to night retrieval (a well-known difficult problem for image based methods) where our PN_VLAD significantly outperforms NetVLAD. It can be seen that the use of point clouds makes the performance more robust to scene variations as they are more invariant to illumination and weather changes.

Qualitative Analysis Fig. 1 and 4 show some of the successfully recognized point clouds, and it can be seen that our network has learned to ignore irrelevant noise such as ground snow and cars (both parked and moving). Fig. 7 shows examples of unsuccessfully retrieved point clouds, and we can see that our network struggles on continuous roads with very similar features (top row) and heavily occluded areas (bottom row).

Usability We further studied the usability of our network for place recognition. Fig. 9 shows heat maps of correctly recognized submaps for a database pair in B.D. before and after network refinement. The chosen database pair

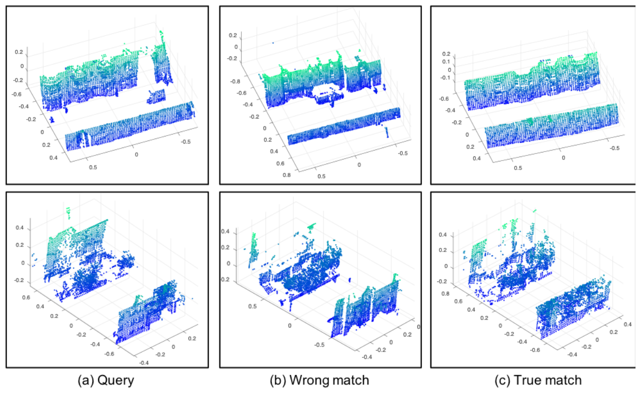

Figure 7. Network limitations: These are examples of unsuccessfully retrieved point clouds by our network, where (a) shows the query, (b) shows the incorrect match to the query and (c) shows the true match.

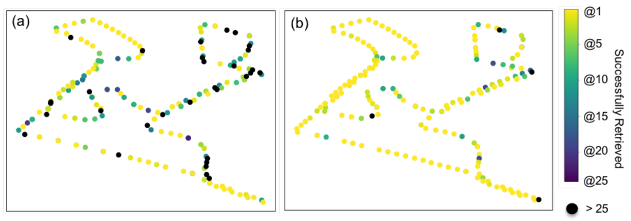

Figure 8. Figure shows the retrieved map of our PointNetVLAD for a randomly selected database-query pair of the unseen B.D. for (a) baseline model and (b) refined model.

is the pair with the lowest initial recall before network refinement. It is shown that our network indeed has the ability to recognize places almost throughout the entire reference map. Inference through our network implemented on Tensorflow[1] on an NVIDIA GeForce GTX 1080Ti takes $\sim$ 9ms and retrieval through a submap database takes $O(\log n)$ making this applicable to real-time robotics systems.

6. Conclusion

We proposed the PointNetVLAD that solves large scale place recognition through point cloud based retrieval. We showed that our deep network is permutation invariant to its input. We applied metric learning for our network to learn a mapping from an unordered input 3D point cloud to a discriminative and compact global descriptor for the retrieval task. Furthermore, we proposed the “lazy triplet and quadruplet” loss functions that achieved more discriminative and generalizable global descriptors. Our experimental results on benchmark datasets showed the feasibility and usability of our network to the largely unexplored problem of point cloud based retrieval for place recognition.

Acknowledgement We sincerely thank Lionel Heng from DSO National Laboratories for his time and effort spent on assisting us in data collection.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016. 8

[2] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. NetVLAD: CNN architecture for weakly supervised place recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2, 3, 8

[3] R. Arandjelovic and A. Zisserman. All about (vlad). In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013. 2, 4

[4] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Journal Computer Vision and Image Understanding (CVIU)*, 2008. 2

[5] W. Chen, X. Chen, J. Zhang, and K. Huang. Beyond triplet loss: a deep quadruplet network for person re-identification. *Computing Research Repository (CoRR)*, 2017. 4

[6] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2

[7] M. Cummins and P. Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *Intl. J. of Robotics Research*, 2008. 1

[8] M. Cummins and P. Newman. Invited Applications Paper FAB-MAP: Appearance-Based Place Recognition and Mapping using a Learned Visual Vocabulary Model. In *International Conference on Machine Learning (ICML)*, 2010. 1

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

[10] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys. Vision-based autonomous mapping and exploration using a quadrotor MAV. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012. 1

[11] D. Gálvez-López and J. D. Tardós. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 2012. 1

[12] C. Häne, L. Heng, G. H. Lee, F. Fraundorfer, P. Furgale, T. Sattler, and M. Pollefeys. 3d visual perception for self-driving cars using a multi-camera system: Calibration, mapping, localization, and obstacle detection. *CoRR*, abs/1708.09839, 2017. 1

[13] R. Haralick, D. Lee, K. Ottenburg, and M. Nolle. Analysis and solutions of the three point perspective pose estimation problem. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 592–598, 1991. 1

[14] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004. 1

[15] H. Jegou, M. Douze, C. Schmid, and P. Prez. Aggregating local descriptors into a compact image representation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. 2, 4

[16] R. Klokov and V. S. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *IEEE International Conference on Computer Vision (ICCV)*, 20178. 2

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012. 2

[18] G. H. Lee and M. Pollefeys. Unsupervised learning of threshold for geometric verification in visual-based loop-closure. In *IEEE International Conference on Robotics and Automation (ICRA)*. 1

[19] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 2004. 2

[20] W. Maddern, G. Pascoe, C. Linegar, and P. Newman. 1 Year, 1000km: The Oxford RobotCar Dataset. *The International Journal of Robotics Research (IJRR)*, 2017. 1, 2, 5, 6, 8, 10

[21] M. J. Milford and G. F. Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012. 1

[22] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006. 1

[23] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *IEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 3, 6

[24] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. *Computing Research Repository (CoRR)*, 2016. 2

[25] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[26] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3d registration. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009. 2

[27] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. In *EEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008. 2

[28] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011. 5

[29] T. Sattler, M. Havlena, F. Radenovic, K. Schindler, and M. Pollefeys. Hyperpoints and fine vocabularies for large-scale location recognition. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. 1

[30] T. Sattler, M. Havlena, K. Schindler, and M. Pollefeys. Large-scale location recognition and the geometric bursti-

ness problem. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1

[31] T. Sattler, A. Torii, J. Sivic, M. Pollefeys, H. Taira, M. Okutomi, and T. Pajdla. Are large-scale 3d models really necessary for accurate visual localization? In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1

[32] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. *Computing Research Repository (CoRR)*, 2015. 4

[33] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *Proceedings of Robotics: Science and Systems (RSS)*, 2009. 2

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. 2014. 2

[35] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *IEEE International Conference on Computer Vision (ICCV)*, 2003. 1

[36] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. Narf: 3d range image features for object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010. 2

[37] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. 2

[38] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. 1

[39] F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In *European Conference on Computer Vision (ECCV)*, 2010. 2

[40] A. Torii, R. Arandjelovic, J. Sivic, M. Okutomi, and T. Pajdla. 24/7 place recognition by view synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1, 7

[41] D. Z. Wang and I. Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems (RSS)*, 2015. 2

[42] Z. Wu, S. Song, A. Khosla, F. Yu, X. T. L. Zhang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 2

[43] B. Zeisl, T. Sattler, and M. Pollefeys. Camera pose voting for large-scale image-based localization. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. 1

[44] A. Zeng, S. Song, M. Nießner, M. Fisher, and J. Xiao. 3dmatch: Learning the matching of local 3d geometry in range scans. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

Supplementary Materials

A. Benchmark Datasets

We provide additional information on the benchmark datasets that are used to train and evaluate the network, which are based on the Oxford RobotCar dataset [20] and three in-house datasets. Figure 9 (top row) shows a sample reference map for each of the four regions. Figure 9 (middle row) shows sample submaps from the different regions. Figure 10 illustrates data splitting into disjoint reference maps, which was done by randomly selecting 150m $\times$ 150m regions.

B. Implementation Details

We use a batch size of 3 tuples in each training iteration. Each tuple is generated by selecting an anchor point cloud P_a from the set of submaps in the training reference map followed by an on-line random selection of P_{pos} and $\{P_{neg}\}$ for each anchor. Each training tuple contains 18 negative point clouds, i.e. $|\{P_{neg}\}| = 18$. Hard negative mining is used for faster convergence by selecting the hardest/closest negatives from 2000 randomly sampled negatives to construct $\{P_{neg}\}$ for each anchor P_a in an iteration. The hard negatives are obtained by selecting the 18 closest submaps from the cached global descriptors f of all submaps in the training reference map, and the cache is updated every 1000 training iterations. We also found network training to be more stable when we take the best/closest of 2 randomly sampled positives to P_a in each iteration.

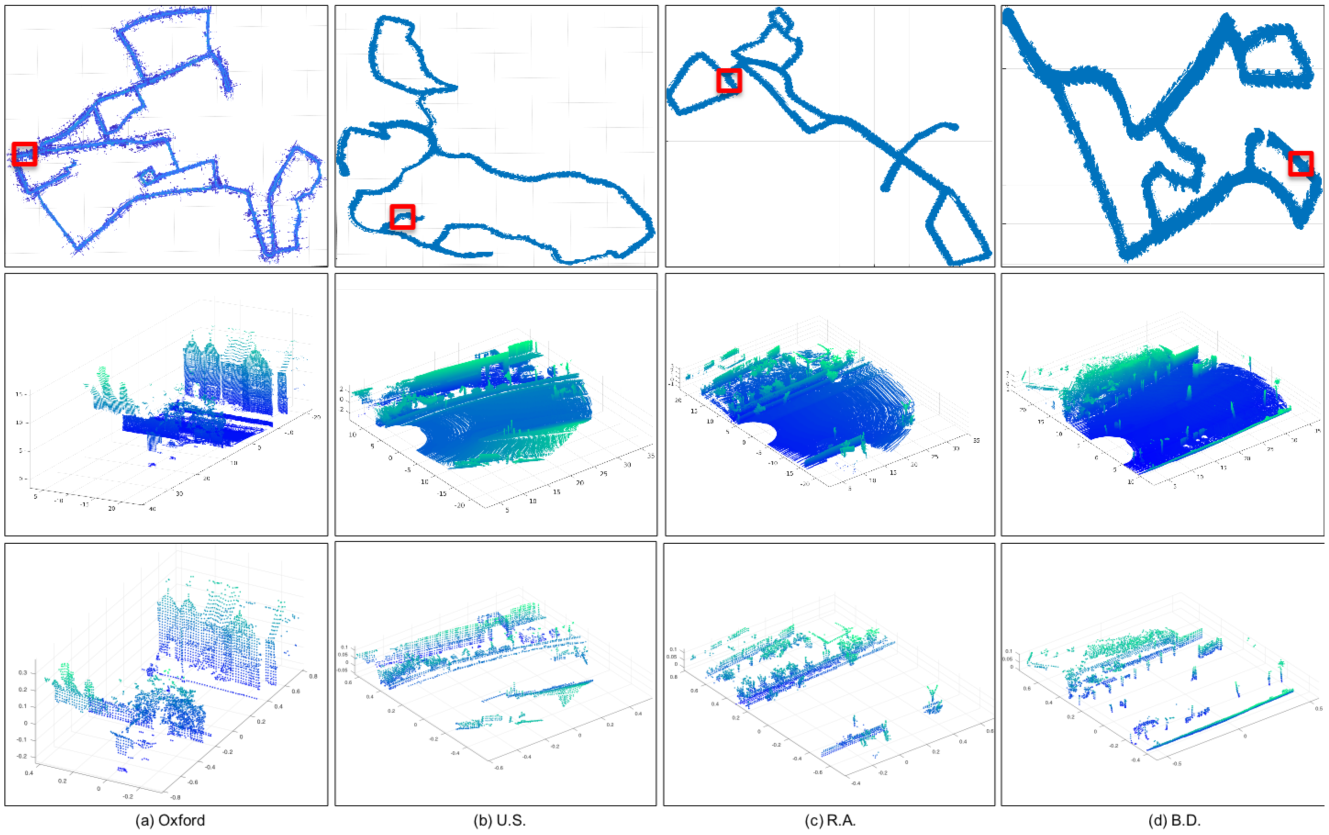

Figure 9. Top row shows a sample reference map from (a) Oxford, (b) U.S., (c) R.A and (d) B.D.. Middle row shows a sample submap from each of the regions representing the local area marked by the red box on the reference map. Bottom row shows the corresponding preprocessed submaps of the local areas from the middle row.

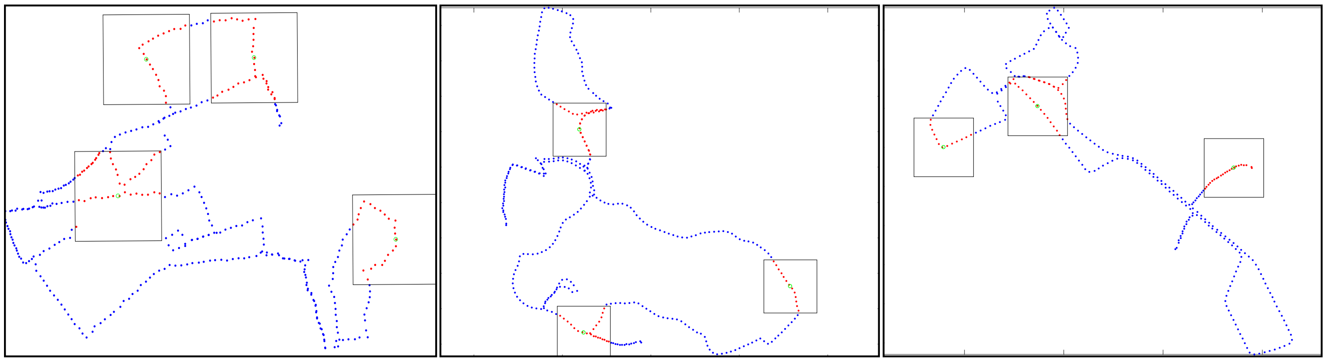

Figure 10. Data splitting: Blue points represent submaps in the training reference map and red points represent submaps in the testing reference map. The data split was done by randomly selecting regions in the full reference map.