Hierarchical Encoding of Sequential Data with Compact and Sub-linear Storage Cost

Huu Le[1], Ming Xu[1], Tuan Hoang[2], and Michael Milford[1]
[1] Queensland University of Technology (QUT), Australia
[2] Singapore University of Technology and Design (SUTD), Singapore

Abstract

Snapshot-based visual localization is an important problem in several computer vision and robotics applications such as Simultaneous Localization And Mapping (SLAM). To achieve real-time performance in very large-scale environments with massive amounts of training and map data, techniques such as approximate nearest neighbor search (ANN) algorithms are used. While several state-of-the-art variants of quantization and indexing techniques have been demonstrated to be efficient in practice, their theoretical memory cost still scales at least linearly with the training data (i.e., $O(n)$ where n is the number of observations in the database), since each observation must be associated with at least one code vector. To address these limitations, we present a novel hierarchical encoding approach that enables sub-linear *storage for* sequential *data, e.g. ordered frames in a video sequence commonly used in visual localization datasets. The algorithm exploits the widespread sequential nature of sensor information streams in robotics and autonomous vehicle applications and achieves, both theoretically and experimentally, sub-linear scalability in storage required for a given environment size. Furthermore, the associated query time of our algorithm is also of sub-linear complexity. We benchmark the performance of the proposed algorithm on several real-world benchmark datasets and experimentally validate the sub-linearity of our approach, while also showing that our approach yields competitive* ***absolute*** *storage performance as well.*

1. Introduction

In recent years, there has been a rapidly growing demand for improved capabilities underlying autonomous robot and vehicle applications, such as Simultaneous Localization And Mapping (SLAM). This demand has inspired a large body of work that address the visual place recognition problem, a key component of SLAM systems, with encouraging results [12, 22]. Among the existing approaches, snapshot-based methods, which primarily rely on image retrieval [33, 31, 11], still play a significant role since they are widely used not only for direct localization but also as an intermediate step that bootstrap large-scale structure-based systems. Generally, the underlying mechanism behind most retrieval-based variants is the utilization of global image descriptors [18, 1, 35, 16] to characterize locations. A query image can then be localized by searching for representations in a database that have similar descriptor values. It has been shown in recent work that the use of source coding techniques, represented by well-known algorithms such as vector quantization [14] and variants such as Product Quantization [17], Optimized Product Quantization [13], together with indexing techniques such as Inverted Index [19, 2] have enabled efficient approximate nearest neighbor search. As a consequence, several techniques have been proposed that can handle datasets containing billions of observations with real-time performance [4, 6].

In parallel to the development of snapshot-based algorithms, structure-based approaches have also attracted much attention from researchers [32]. Unlike their counterparts that provide only a rough location estimate, structured-based algorithms yield precise localization up to six degrees-of-freedom (6-DoF). The maps are represented by 3D point clouds, which can be obtained from registering depth sensor output, or built from the training images using common Structure-from-Motion (SfM) frameworks [34, 37]. Apart from the location information, each point in the 3D map also stores the associated local descriptors, so that a query image can be localized by conducting 2D-3D matching to identify the camera location and orientation. However, in order to achieve satisfactory real-time results, the input data must be of small or moderate size, as searching over very large point clouds is often computationally expensive.

Although many indexing algorithms in the literature have achieved sub-linear query time, most approaches require storing one code vector per image descriptor. Theoretically, no matter how short this code vector, for n observations in the dataset, existing algorithms require $O(n)$ stor-

age cost, i.e., the required amount of storage scales linearly with the number of observations. While recent research focuses on improving the precision-recall metrics, the scalability of the memory footprint is given little attention. Our work addresses this and introduces a theoretically sub-linear algorithm that can be used as an alternative for existing quantization approaches. By conducting experiments on real-world datasets, we show that our method yields competitive results on challenging visual place recognition problems.

Contributions Our contributions can be summarized as:

- We propose a new encoding algorithm for snapshot-based localization that *theoretically* achieves *sub-linear storage scale* w.r.t. the number of observations n, where existing state-of-the-art approaches are at best linear. The associated time complexity for a query is also sub-linear in this proposed approach.
- This sub-linearity is achieved through a hierarchical encoding scheme that exploits the sequential nature of data prevalent in sensor streams across mobile robotics and autonomous vehicles.
- Unlike some rigid existing systems, the proposed encoding scheme enables the user to explicitly trade prediction accuracy for memory footprint, enabling it to be tailored to the specific accuracy and memory requirements of different applications.
- We show *empirically* that the proposed technique achieves the desired sub-linearity in storage, while also yielding competitive *absolute* storage requirements compared to existing state-of-the-art techniques.

2. Background

Snapshot-based approaches cast visual localization as a nearest-neighbor search problem, where for a query image, we wish to find the most "similar" observation in a database of images that represents previously observed locations. Instead of storing the images directly, it is usually desirable to store a lower dimensional image descriptor, especially for this application. Specifically, we are given a dataset $\mathcal{D}$ containing n observations $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional image descriptor. Furthermore, we will explicitly assume that $\mathcal{D}$ is *sequential* visual localisation data, i.e., a sequence of frames from a video. Specifically, for i close to j, $\mathbf{x}_i$ is very similar to $\mathbf{x}_j$. We will discuss the importance of this assumption in more detail in Section 3.2. For a query vector $\mathbf{q} \in \mathbb{R}^d$, we wish to find $\mathbf{x}^* \in \mathcal{D}$ such that the distance between $\mathbf{q}$ and $\mathbf{x}^*$ is minimized using some measure of similarity. Formally, we express this as

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{D}} d(\mathbf{q}, \mathbf{x}), \tag{1}$$

where $d(\cdot, \cdot)$ is an appropriate metric that measures the similarity between two descriptors. In this paper, we assume that this metric is the standard Euclidean distance and so (1) can be rewritten as

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{D}} \|\mathbf{q} - \mathbf{x}\|_2, \tag{2}$$

where $\|\cdot\|_2$ denotes the Euclidean norm. Naively, we can solve this problem easily by evaluating $\|\mathbf{q} - \mathbf{x}\|_2$ for every $\mathbf{x} \in \mathcal{D}$, however this takes $O(nd)$ operations, which is computationally infeasible for large n. To alleviate this problem, the nearest neighbor search must be approximated, thus quantization techniques are employed to map n distinct observations into $k \ll n$ "code vectors", so that data can be efficiently queried. Using these approximate search techniques, the original problem (2) is replaced by

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{D}} \|\mathbf{q} - g(\mathbf{x})\|_2, \tag{3}$$

where $g : \mathcal{D} \to \mathbb{R}^d$ maps an observation into a quantized version of $\mathcal{D}$ with $k \ll n$ distinct values [14, 17, 13]. A simple example of this is quantizing the dataset by applying K-means clustering [24] for $k \ll n$. Let $g(\cdot)$ map $\mathbf{x}$ to its nearest K-means centroid. This is a specific example of the vector quantization approach (VQ) [14], which partitions the training data $\mathcal{D}$ into k clusters, and assign $g(\mathbf{x})$ as the cluster centroid that is closest to $\mathbf{x}$. With code-books containing k code-words, a descriptor vector can be stored using $\log_2 k$ bits, and the distance calculation can be sped up using pre-computed lookup tables. Due to the computational hardness of K-Means clustering [24, 25], the use of VQ is infeasible for large k (e.g., 2^{64}). Several variants of VQ [17, 13, 20, 15, 21] have been proposed to address such shortcomings. Leveraged from VQ, Product Quantization [17] (and its improvements [17, 13]) splits observations into m distinct subvectors, each with dimension d/m. Each group of subvectors forms a subspace, which is then separately quantized using conventional VQ methods, generating m "code-books", each containing k "code-words". The total number of possible code-words (hence quantiles) is therefore k^m, which is substantially larger than the amount allowed under traditional VQ.

Despite all of this, the use of quantization alone does not attain real-time performance on consumer hardware for extremely large n, since a query vector still needs to be compared against all words in the code books which may still be large. To further speed up the search, indexing techniques such as IVF [19] or Multi-Index [19] are used. These techniques propose partitioning data into several regions where each region is associated with a list of indices. A query vector therefore needs to be compared against this list of candidates rather than the whole dataset. In addition, advanced tree indexing methods such as [3, 5, 6] can also be used to achieve significant improvements in performance.

Disclaimer Our system is not a replacement for visual SLAM and localization systems like [28, 29], since we abstract away the choice of feature descriptors and focus on

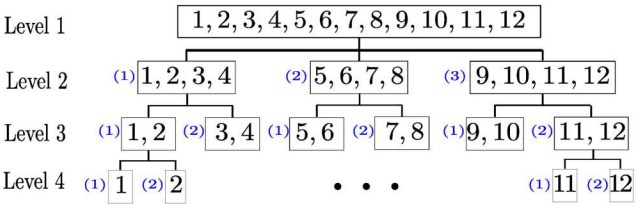

Figure 1. Example of our hierarchical encoding scheme for a dataset containing $n = 12$ observations. An $l = 4$ level tree is used here. The numbers of chunks in the levels are given by $h_1 = 3$, $h_2 = 2$, $h_3 = 2$, respectively. The numbers in each node (rectangle) are the indices of the observations belonging to that particular node. The blue numbers (in parentheses) denote the chunk index.

nearest neighbour search. However, we can foresee our methodology being used for instance as one of the modules in ORB-SLAM [29] where given an image, we wish to locate the closest keyframe associated to that image in a large database of keyframes.

3. Approach

As we have previously discussed, although achieving promising results, existing methods require the storage footprint to scale linearly with n. In this section, we introduce a new quantization approach that achieves sub-linear storage cost which can be used for real-time visual localization applications. Our approach is first based on the observation that in visual localization, given a query vector $\mathbf{q}$, we only desire the index of the nearest vector $\mathbf{x}^*$ since this represents a *location* in an environment. Specifically, in the context of (2), we would like find i^*, where

$$i^* = \arg\min_i \|\mathbf{q} - \mathbf{x}_i\|_2, \quad \forall \mathbf{x}_i \in \mathcal{D}. \tag{4}$$

We again argue that existing methods described in Section 2 which return the full vector $\mathbf{x}^*$ are redundant for visual localization where only the index i is sufficient. In Section 3.1, we discuss our proposed quantization approach.

3.1. Hierarchical Representation

Similar to other quantization techniques, in our approach, each data instance $\mathbf{x} \in \mathcal{D}$ is also associated with a code vector $c(\mathbf{x})$. However, in contrast to existing algorithms that store the codes $\{c(\mathbf{x}_i)\}_{i=1}^n$ with the training data (which scales with $O(n)$) , our approach does not require this. Instead, we only store information of our model parameters, which consists of transformation matrices (see Sec. 3.3) and separating hyper-planes (see Sec. 3.2) learnt from the training data. Now, given a query vector $\mathbf{q}$, the code of its nearest neighbor $c(\mathbf{q})$ and an estimate of i^* from (4) is estimated directly using the model.

In order to obtain the index i^* of a vector $\mathbf{x}$ from a code vector $c(\mathbf{x})$, we employ a hierarchical encoding approach. Specifically, we recursively divide the database into into chunks, where each chunk contains a set of consecutive observations. This partitioning process is performed up to l levels to form a tree structure as illustrated by Fig. 1. For each node in the j-th level, the data belonging to that particular node is divided into h_{j+1} chunks (note that for the leaf nodes, $h_l = 1$). Note that the order of the observations in $\mathcal{D}$ is left unchanged (i.e., the division into chunks is performed purely based on the indices, as illustrated in Fig. 1). This process is recursively repeated further down the tree until the $(l-1)$-th level. The values of $\{h_j\}_{j=1}^{l-l}$ are chosen such that $h_1 h_2 \cdots h_l = n$. Using the example shown in Fig. 1, the tree has $l = 4$ levels, and the values of h_1 to h_4 are $h_1 = 3$, $h_2 = 2$, $h_3 = 2$ and $h_4 = 1$ respectively, so $h_1 h_2 h_3 h_4 = 12$. Other choices of l and $\{h_j\}$ exist, such as $l = 3$, $h_1 = 4$ and $h_2 = 3$. The effect of different choices of tree structure and partitioning schemes will be discussed in the later sections and evaluated in the experiments.

Let $\mathcal{C}(\mathbf{x}) = \{c_j(\mathbf{x})\}_{j=1}^{l-1}$ denote the chunk indices and code vector corresponding to an observation $\mathbf{x} \in \mathcal{D}$. For example, let $\mathbf{x}_2$ be observation 2 in the example presented in Fig. 1. In this case, $\mathcal{C}(\mathbf{x}_2) = \{1, 1, 2\}$, which is given by following observation 2 through the tree and recording the sequence of chunk indices. Similarly, $\mathcal{C}(\mathbf{x}_{12}) = \{3, 2, 2\}$. We can evaluate the index i of observation $\mathbf{x}$ given its corresponding chunk indices $\mathcal{C}(\mathbf{x})$ by evaluating

$$i = (c_1 - 1)\frac{n}{h_1} + \cdots + (c_{l-2} - 1)\frac{n}{h_1 \ldots h_{l-2}} + c_{l-1}. \tag{5}$$

For brevity, we use c_i to represent $c_i(\mathbf{x})$ in (5). During the query stage, a query vector $\mathbf{q}$ is propagated down through tree from the root node down to a leaf node. We do this by feeding $\mathbf{q}$ through our proposed model, discussed in more detail in Sections 3.2 and 3.3, which returns the chunk indices $\mathcal{C}(\mathbf{q})$. These are used to compute the index of observation $\mathbf{x}^*$ as per (5). In Fig. 1, if $\mathbf{x}_1$ is the nearest neighbor of $\mathbf{q}$, the desired values of $\mathcal{C}(\mathbf{q})$ should be $\{1, 1, 1\}$.

3.2. Learning Separating Hyperplanes

We describe our model for estimating $\mathcal{C}(\mathbf{q})$ for any given query vector $\mathbf{q}$. In other words, we aim to learn a parameterized function f which takes a query vector and returns its corresponding code vector, specifically $f(\mathbf{q}) = \mathcal{C}(\mathbf{q})$. Our predictions from this learned f, denoted $\hat{\mathcal{C}}(\mathbf{q})$ will be used to estimate i^* as per (5)). We choose to fit multi-class Support Vector Machine (SVM) classifiers to $\mathcal{D}$ for each level to learn separating hyperplanes that divides $\mathcal{D}$ into the desired chunks. Specifically, for each level $j \in \{1, \ldots, l-1\}$, an SVM classifier is fitted to the collection of observation/chunk index pairs $\{\mathbf{x}_i, c_j(\mathbf{x}_i)\}_{i=1}^n$. Subsequently, we end up with a collection of $l-1$ SVM classifiers $\{f_j(\cdot)\}_{j=1}^{l-1}$, where $f_j(\mathbf{q}) \in \{1, \ldots, h_j\}$ $\forall \mathbf{q}$ is the SVM classifier that predicts the corresponding chunk index of a query vector in

level j. From this, we can exress f as $f(\mathbf{q}) = \{f_j(\mathbf{q})\}_{j=1}^{l-1}$. For example, using Level 2 from Figure 1, we wish to train an SVM on $\mathcal{D}$ that classifies $\mathbf{q}$ similar to observations 1 to 4 into chunk 1, $\mathbf{q}$ similar to observations 5 to 8 into chunk 2 and so on. This is repeated for all other levels. In total, we will end up with 3 SVM classifiers fitted to $\mathcal{D}$ with the class labels used to train each SVM given by the chunk labels shown. Note that each level has only **one** SVM, and SVMs are not fitted separately for each node. This is required to maintain the sub-linearity of our approach.

We choose to utilize the framework of the Linear Support Vector Machine (LSVM) [7] and forego any kernel based methods [9]. This is because kernel based methods typically involve storing storing a Gram matrix with complexity $O(n^2)$, violating the sub-linearity required. From [10], training $f_j(\cdot)$ with h_j classes amounts to finding a set of h_j separating hyperplanes $\{\mathbf{w}_k^j\}_{k=1}^{h_j}$ that satisfies

$$\begin{aligned} \min_{\{\mathbf{w}_k\},\{\beta_i\}} \quad & \frac{1}{2}\sum_{k=1}^{h_j}\|\mathbf{w}_k\|^2 + C\sum_{i=1}^{n}\beta_i \\ \text{s.t.} \quad & \mathbf{w}_{y_i}^T\mathbf{x}_i - \mathbf{w}_m^T\mathbf{x}_i \geq e_m^i - \beta_i \;\forall m, i, \end{aligned} \tag{6}$$

where $C > 0$ is a regularization parameter, $e_m^i = 1 - \mathbb{I}(y_i = m)$ and $\mathbb{I}(\cdot)$ is an indicator function. We used LIBSVM [8] to solve (6), although other libraries can also be used.

A crucial assumption for the applicability of LSVMs is that the observations are linearly separable in the original input space according to the corresponding labels. Given that the labels are given by the chunking indices, this assumption does not hold for any general unordered dataset since these labels may not reflect any "natural" clustering present in the dataset. However, the majority of sensory streams received from mobile robots and autonomous vehicles consist of consecutive observations that are locally similar; our allocation of chunk indices gives the same label to sequences of consecutive observations, hence correspond to "natural" clusters in the data. However, for levels closer to the leaf nodes, labels no longer correspond to coherent sequences (indeed, for the leaf nodes, chunk labels alternate for each consecutive observation) and we would expect classification performance to be worse here.

3.3. Data Transformation and Subspace Clustering

In practice, LSVMs yield poor performance in practice when applied directly to the raw observations and chunking indices. This is because data in the original input space is rarely ever linearly separable, even with inherent structure, motivating kernel methods that aim to find a separating hyperplane in a higher dimensional "feature space" instead. Since we are avoiding the use of kernel methods here to maintain sub-linearity, we instead find an explicit transformation of the data at each level $\mathbf{L}_j$ such that the transformed observation/label pairs $\{\mathbf{L}_j\mathbf{x}_i, c_j(\mathbf{x}_i)\}_{i=1}^n$ are more easily separable, allowing the use of an LSVM. For our approach, we propose learning a linear transformation $\mathbf{L}_j \in \mathbb{R}^{d\times m_j}$, where $m_j \leq d$.

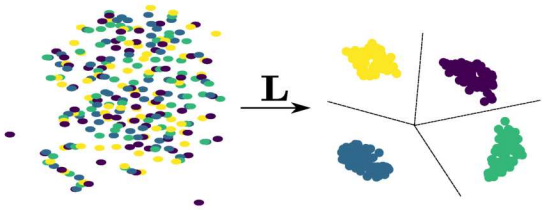

Figure 2. Illustration of the data transformation and classification process visualized using a t-SNE embedding [23] in 2D. Left: Original data that is not linearly separable. Right: The transformed data which is easily divided using a set of hyper-planes.

We now discuss how we learn this optimal transformation $\mathbf{L}_j$. Our approach is inspired by [26], with specific modifications to allow supervised clustering with class labels. We wish to find an orthogonal matrix $\mathbf{V} \in \mathbb{R}^{d\times d}$ to solve the following optimization problem:

$$\begin{aligned} \min_{\mathbf{V}\in\mathbb{R}^{d\times d},\{\boldsymbol{\mu}_k\}} \quad & \sum_{k=1}^{h_j}\sum_{\mathbf{x}\in\mathcal{L}_k}\left\|\mathbf{P}_m^T\mathbf{V}^T\mathbf{x} - \mathbf{P}_m^T\mathbf{V}^T\boldsymbol{\mu}_k\right\|_2^2 \\ & + \sum_i\left\|\tilde{\mathbf{P}}_m^T\mathbf{V}^T\mathbf{x}_i - \tilde{\mathbf{P}}_m^T\mathbf{V}^T\boldsymbol{\mu}_0\right\|_2^2, \\ \text{s.t.} \quad & \mathbf{V}^T\mathbf{V} = \mathbf{I}, \end{aligned} \tag{7}$$

where $\mathcal{L}_k$ represents the set of data points with label k, i.e., $\mathcal{L}_k = \{\mathbf{x}_i | \mathbf{x}_i \in \mathcal{D}, c(\mathbf{x}_i) = k\}$, $\{\boldsymbol{\mu}_k\}_{k=1}^{h_j}$ are cluster centroids corresponding to the each of the h_j labels, while $\boldsymbol{\mu}_0$ is the centroid associated with the so-called "noise" subspace that does not contain structural information. The main difference to [26] is that we fix $\mathcal{L}_k$ while optimizing (7) as opposed to updating the cluster allocations using K-means at each iteration. Furthermore, the matrices $\mathbf{P}_m$ and $\tilde{\mathbf{P}}_m$ are subspace projection matrices given by

$$\mathbf{P}_m = \begin{bmatrix} \mathbf{I}_m \\ \mathbf{0}_{(d-m)\times m} \end{bmatrix} \quad \tilde{\mathbf{P}}_m = \begin{bmatrix} \mathbf{I}_{d-m} \\ \mathbf{0}_{m\times(d-m)} \end{bmatrix} \tag{8}$$

Here we use the notation $\mathbf{I}_m$ to denote the identity matrix of size $m \times m$, and $\mathbf{0}_{a\times b}$ denotes a matrix of size $a \times b$ that contains all zeros. From this, $\mathbf{L} = \mathbf{P}_m^T\mathbf{V}^T$.

The intuition behind optimizing (7) is that we aim to find an orthogonal matrix $\mathbf{V} \in \mathbb{R}^{d\times d}$ that transforms observations $\{\mathbf{x}_i\}$ in such a way that the first m dimensions of the transformed observations, i.e., $\{\mathbf{P}_m^T\mathbf{V}^T\mathbf{x}\}$ form h_j clusters according to the corresponding chunk indices. Observe that the objective function in (7) is designed to minimize the within cluster sum of squares (WCSS) of all data points. This scheme is analogous to several distance metric learning approaches [38, 36] that learn a Mahalanobis distance that

optimally clusters "similar" observations (e.g., class labels) under this new distance. These methods come with a corresponding linear transformation, such that the transformed data is much more easily separable. Due to the availability of this transformation, metric learning techniques are also applicable to our algorithm. We tested Large Margin Nearest Neigbours (LMNN) [36] as an alternative to our method, however we found that this performed poorer overall. We believe this is because there is no natural dimensionality reduction built into LMNN, and specifying $m < d$ involves applying dimensionality reduction techniques such as Principal Components Analysis (PCA) to the data as a preprocessing step. PCA does not necessarily optimally reduce dimensionality based on the clusters we desire given by the chunk labels, since it is unsupervised.

Note that since we are restricting ourselves to linear transformations, there is no guarantee of separating observations in general. For instance, data dispersed along a line will be mapped to another line under a linear transformation and cannot be separated using this method. We find that in practice, we can find reasonable transformations for sequential data due to the "natural" clustering present in the data. In Section 3.3.1, we outline the solutions to (7).

3.3.1 Solve for $\{\boldsymbol{\mu}_k\}$

The optimal values of $\{\boldsymbol{\mu}_k^*\}_{k=1}^{h_j}$ in (7) can be computed in closed-form. This is performed by computing the gradient of the objective function in (7) w.r.t $\boldsymbol{\mu}_k$ and setting this to the 0 vector (see the supplementary materials for a full derivation). This yields

$$\boldsymbol{\mu}_k = \frac{1}{|\mathcal{L}_k|} \sum_{\mathbf{x} \in \mathcal{L}_k} \mathbf{x}, \quad \forall k = 1, \cdots, h_j. \tag{9}$$

Note that the solution is independent of $\mathbf{V}$ and m.

3.3.2 Finding the optimal transformation matrix

Following [26], we can rewrite the optimization problem w.r.t. $\mathbf{V}$ given a fixed $\{\boldsymbol{\mu}\}_{k=1}^{h_j}$ and m (see supplementary material for more details) as

$$\min_{\mathbf{V}} \text{trace}(\mathbf{P}_m \mathbf{P}_m^T \mathbf{V}^T \boldsymbol{\Sigma} \mathbf{V}) + \text{trace}(\mathbf{V}^T \mathbf{S}_0 \mathbf{V}), \tag{10}$$

where the $\mathbf{S}_0$ and $\boldsymbol{\Sigma}$ are given by

$$\mathbf{S}_0 = \sum_{\mathbf{x} \in \mathcal{D}} (\mathbf{x} - \boldsymbol{\mu}_0)(\mathbf{x} - \boldsymbol{\mu}_0)^T, \tag{11}$$

and

$$\boldsymbol{\Sigma} = \sum_{k=1}^{h_j} \sum_{\mathbf{x} \in \mathcal{L}_k} (\mathbf{x} - \boldsymbol{\mu}_k)(\mathbf{x} - \boldsymbol{\mu}_k)^T - \mathbf{S}_0. \tag{12}$$

Using the fact that $\mathbf{V}$ is orthogonal along with the invariance of the trace operator under cyclic permutations, $\text{trace}(\mathbf{V}^T \mathbf{S}_0 \mathbf{V}) = \text{trace}(\mathbf{V}\mathbf{V}^T \mathbf{S}_0) = \text{trace}(\mathbf{S}_0)$, and hence the second term in (10) does not depend on $\mathbf{V}$.

Furthermore, observe that $\mathbf{P}_m \mathbf{P}_m^T$ is a diagonal matrix with the first m diagonal elements being 1, with the rest 0. From this, $\text{trace}(\mathbf{P}_m \mathbf{P}_m^T \mathbf{V}^T \boldsymbol{\Sigma} \mathbf{V})$ is the trace of the leading principal submatrix of dimension m of $\mathbf{V}^T \boldsymbol{\Sigma} \mathbf{V}$. Using the fact that $\boldsymbol{\Sigma}$ is symmetric, we can use a well-known result from constrained optimization to show that a solution for columns of the optimal $\mathbf{V}$ are simply the eigenvectors of $\boldsymbol{\Sigma}$ sorted by their corresponding eigenvalue from lowest to highest (see the supplementary materials for a full derivation). Specifically, the first column of $\mathbf{V}$ corresponds to the eigenvector associated with the lowest eigenvalue of $\boldsymbol{\Sigma}$.

3.3.3 Finding the optimal m

We now select the optimal m. Note that our solution for $\mathbf{V}$ does not depend on m, and so we can optimize m separately. Recall that the trace of a square matrix is equal to the sum of its eigenvalues, and using the fact that $\mathbf{V}$ corresponds to normalized eigenvectors of $\boldsymbol{\Sigma}$ we can easily show that

$$\text{trace}(\mathbf{P}_m \mathbf{P}_m^T \mathbf{V}^T \boldsymbol{\Sigma} \mathbf{V}) = \sum_{i=1}^{m} \lambda_i, \tag{13}$$

where λ_i corresponds to the i-th lowest eigenvalue of $\boldsymbol{\Sigma}$. We wish to select m that minimizes (13). Note that if $\boldsymbol{\Sigma}$ contains no negative eigenvalues, then $m = 0$, whereas if there are k negative eigenvalues of $\boldsymbol{\Sigma}$, then $m = k$. While $\mathbf{S}_0$ and the first sum in (12) are symmetric positive semi-definite (hence only has positive eigenvalues), there is no guarantee that $\boldsymbol{\Sigma}$ is too, due to the subtraction of $\mathbf{S}_0$ in (12). In practice, we find that the optimal m typically corresponds to h_j, the number of chunks.

3.3.4 Storage Analysis

Based on the discussions in the previous sections, the storage requirements for our models include:

- The transformation matrices $\{\mathbf{L}_j\}_{j=1}^{l-1}$, where each $\mathbf{L}_j \in \mathbb{R}^{d \times m_j}$ is the learned transformation and m_j is the optimal value of m at the j-th level of the tree. For each $\mathbf{L}_j$, we store $O(\sum_{j=1}^{l-1} m_j d)$ parameters (see supplementary material for more details).

  Note that for $l = 1$, $\sum_{j=1}^{l-1} m_j = n$, hence storage scales linearly with n. However, if we structure the hierarchical model to be a binary tree, then $\sum_{j=1}^{l-1} m_j = 2\log_2(n)$, yielding $O(d \log_2 n)$ storage. Setting $l > 1$ will guarantee sub-linear storage requirements.

- The support vectors characterising the SVM classifiers at each level $\{\mathbf{W}_j\}_{j=1}^{l-1}$, where $\mathbf{W}_j = \{\mathbf{w}_k^j\}_{k=1}^{h_j-1}$ is the collection of m_j-dimensional support vectors for level j. In this case, we must store $O(\sum_{j=1}^{l-1} m_j h_j)$ values. Similar to the above, this scales linearly with n for one level but is sub-linear for $l > 1$.

Overall, our method is sub-linear for $l > 1$, and we will discuss how to select l in Section 4.2.2.

4. Experimental Results

In this section, we evaluate the performance of our proposed algorithm against other approaches for retrieval-based localization. The performance of our method under various hyperparameter configurations are also analyzed and discussed.

Datasets: Two large-scale visual localization datasets are used:

- Nordland Dataset[1]: Nordland consists of four video sequences captured in each season (Fall, Summer, Spring, and Winter) by a camera mounted on a moving train over 10 hours. The video frames are resized and extracted to ensure an equal distance between frames, generating around 20,000 images per season.
- Brisbane Day and Night [27]: Similar to Nordland, this dataset contains video sequences captured from a camera mounted in front of a car traveling in an urban area. We selected two traversals, one captured during the day and the other captured at night.

For each dataset, we extract DenseVLAD [35] $d = 4096$ dimensional descriptors from the training and test data and use these as our image descriptors. We chose DenseVLAD as it is considered state-of-the-art for image retrieval. Other types of descriptors [1, 18, 30] can also be used with our proposed approach with similar performance characteristics. Note that our proposed methodology is purely a means to retrieve the approximate nearest neighbor of a query under the Euclidean metric. For challenging visual localisation datasets where standard image descriptors can result in substantial perceptual aliasing, it is possible to have distant locations with very similar descriptors. The performance of our methodology is bottlenecked by the choice of image descriptor used and how effective that descriptor is in finding unique representations for different locations.

Metric evaluation: Unlike other methods that returns a list of nearest neighbors, our algorithm predicts a single index for a particular query vector. Therefore, in order to evaluate the localization accuracy, we measure and report the percentage of correctly localized query images. A query is regarded as correctly localized if its predicted location is less than t frames away from the ground-truth location. Throughout the experiments, we test the methods with for $t \in \{1, 5, 10, 20, 40, 80, 160\}$. Note that for existing ANN approaches, we use the best nearest neighbor match to measure accuracy. As we focus on a storage-efficient encoding algorithm, we also report the amount of memory required (in MB) for the storage of the methods' parameters.

We compare our proposed technique against commonly used state-of-the-art quantization approaches, including PQ [17], ITQ [15], OPQ [13], and LO-PQ [20]. In addition, the method proposed in [39] (RYTH), which also achieves sub-linear storage scale, is also evaluated. Our implementation is in Python[2] and was tested on a 4.2GHz machine running Ubuntu with 32GB of RAM. LIBSVM was run with the recommended parameter settings.

4.1. Localization Accuracy

In this section, we conduct experiments to benchmark the accuracy of our proposed algorithm against existing approaches. The experiments are conducted with $n = 20,000$ training and test frames (we extract n frames for training and n corresponding frames for testing), where for Nordland, the systems are trained on the Fall sequence and tested on the three remaining sequences. To simulate limited storage settings, all methods are tuned such that their storage footprint cannot exceed 10MB.

Fig. 3 (top) shows the results produced by the benchmarks, where we plot the localization accuracy with varying error tolerance t as per above. The same experiment is repeated where the system is trained on Summer and tested on the other three sequences, and the results are plotted in Fig. 3 (bottom). The average accuracy across all datasets is summarized in Fig. 4.

As shown in Fig. 3, for a given amount of storage, HESSL produces competitive results compared to state-of-the-art algorithms such as PQ, OPQ and LOPQ. When tested on the Spring and Winter sequences (which are considered challenging, even for state-of-the-art algorithms), our results are comparable to PQ and OPQ for small t. If high error tolerance is allowed, as demonstrated in Fig. 3, we outperform other approaches by a large margin. Moreover, compared to [39], we achieve significantly better results, while attaining the sub-linear storage growth w.r.t. n. The above experiment is repeated for the Brisbane Day and Night dataset and the results are plotted in Fig. 5, where the same conclusions apply.

4.2. Ablation Studies

4.2.1 On the System's Scalability

In this experiment, we investigate the performance of our algorithm when n grows given a fixed amount of storage. For

[1] https://nrkbeta.no/2013/01/15/nordlandsbanen-minute-by-minute-season-by-season/

[2] Our source code is available at https://github.com/intellhave/HESSL

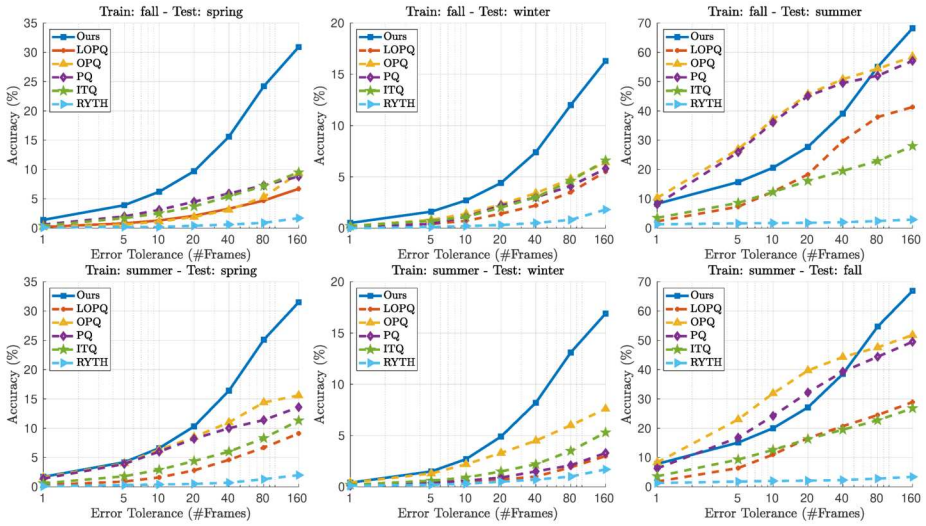

Figure 3. Performance of the benchmarks on the Nordland dataset ($n = 20,000$ training and test images), where the model is trained on Fall (top row) and Summer (bottom row) and tested on the remaining sequences. The storage memory is limited to 9 MB for all methods.

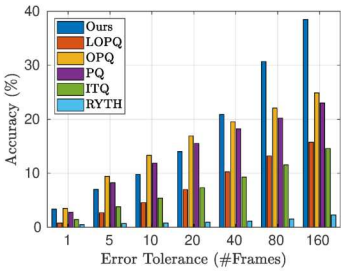

Figure 4. Performance Summary.

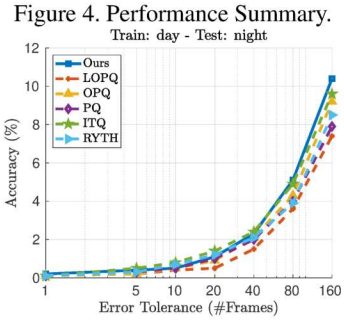

Figure 5. Results of the Brisbane Day Night ($n = 3600$, storage 5 MB).

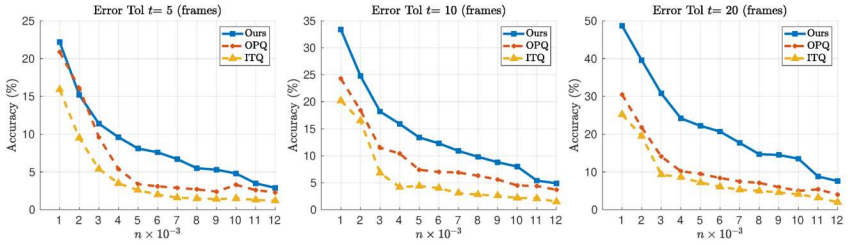

Figure 6. Evaluations of the scalability of our method compared to OPQ and ITQ for a storage budget of 5 MB, with n ranging from 1000 to 12000. Left to Right: Accuracy for different values of error tolerance t.

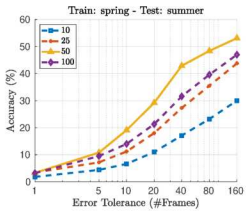

Figure 7. Accuracy for various m for $n = 2500$ on the Nordland dataset. The optimal value m^* is 50 for both levels.

Nordland, we use Spring for training and Fall for testing. Fixing the storage available at 5 MB, we gradually increase the training and test size n and run the methods. The results are shown in Fig. 6, where we demonstrate that given a fixed amount of storage, our approach yields better scalability. In particular, for small n, all methods produce comparable accuracy. However, as the database size grows, although the performance of all methods drop, our technique consistently outperforms existing methods.

4.2.2 On the Choice of Tree Levels l

Table 1 shows the performances of our system with different system configurations, including the number of levels l and the number of chunks in each level. For each configuration, we report the amount of memory required to store the parameters and the corresponding accuracy for a dataset containing $n = 10,000$ points. Observe that as l increases, less storage is required by our method. This is because a tree with higher levels requires smaller values for $\{h_j\}$ (i.e., the number of chunks per level), resulting in smaller transformation matrices $\{\mathbf{L}_j\}$. Also, a smaller h_j requires fewer hyper-planes to be stored for chunk classification, thus the overall storage can be further reduced. This is the explicit accuracy/storage cost alluded to earlier in the text.

l	3	4	5
$\{h_j\}_{j=1}^{l-1}$	$\{100, 100\}$	$\{20, 22, 23\}$	$\{10, 10, 10, 10\}$
Storage	6.40MB	2.04MB	1.25MB
Accuracy ($t = 5$)	5.2%	2.8%	1.3%
Accuracy ($t = 10$)	8.4%	4.5%	2.1%
Accuracy ($t = 20$)	13.4%	7.3%	3.2%
Accuracy ($t = 160$)	34.2%	19.2%	11.9%

Table 1. Analysis of system performance with different tree partitioning schemes for a dataset containing $n = 10,000$ images, trained on Spring and tested on Summer.

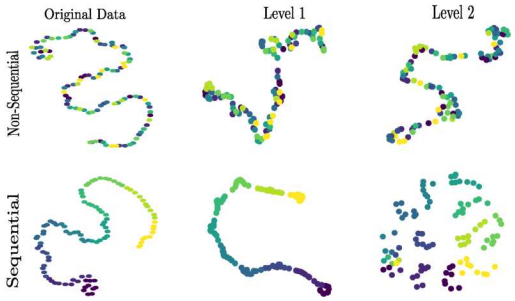

Figure 8. Visualization of transformed data for sequential and non-sequential data.

4.2.3 On the Reduced Dimension m

To analyze the effect of m on the performance of HESSL , we run the same experiment for $m \in \{10, 25, 50, 100\}$ with $n = 2500$ observations taken from Nordland Spring as training and Summer for testing, $l = 2$ and $h_1 = h_2 = 50$. Fig. 7 shows the results. Interestingly, for this particular setting, the optimal value of m obtained as described in Sec. 3.3.3 is given by $m^* = h_1 = h_2 = 50$, which is consistent to in 7. Specifically, when $m < m^*$, the dimensionality of the transformed space is insufficient for the system to separate observations adequately. However, if $m > m^*$, the overall performance also drops, possibly due to redundancy in the training data that degrades the accuracy of the LSVMs. At $m = m^* = 50$, the system attains best accuracy. This validates that the parameter m should be optimized using our learning technique.

4.2.4 On the Effect of Non-sequential Data

As we have previously discussed, sequential data admits some form of natural clustering, which yields our strong empirical results. To illustrate the effect of non-sequential data, we plot in the first column of Fig. 8 the t-SNE embedding of an $n = 100$ frame long video sequence where the frames have been shuffled (top row) along with the original ordering (bottom row), where the frames are divided into $\{h_1, h_2\} = \{10, 10\}$ chunks. It is clear from Fig. 8 that observations that are near each other w.r.t. Euclidean distance are also given the same labels. In the second and third columns we show the t-SNE embedding of the transformed data in the first and second layer of the tree, respectively. Observe that for sequential data, the transformed data in both levels show a tendency to be linearly separated, while this is not the case for non-sequential data. This also demonstrates that if the data is sequential, our data mapping technique is able to transform data such that they can be classified using LSVMs.

4.2.5 Sequence Filtering

Many robotic and autonomous vehicle applications offer the potential to temporally filter data over multiple frames to improve performance, while not requiring any notable growth in storage requirements, but at some cost in increased compute requirements and latency in matching. Although filtering is not the primary purpose or expectation of our proposed method, to represent what performance may be achievable with filtering, we implemented a lightweight filtering technique, which can be performed for a window of size w, where the localization of a new query vector is corrected if its predicted index is off by $w/2 + a$ frames from the median of its immediate window. Fig. 9 shows example results for the proposed filtering scheme for a dataset containing 1000 frames, with two variants of window sizes $w \in \{5, 20\}$, and $a = 3$ frames. As expected, sequence filtering has considerably improved the overall accuracy of our algorithm, demonstrating that filtering is another tool by which a user could choose the balance between accuracy, compactness, and latency.

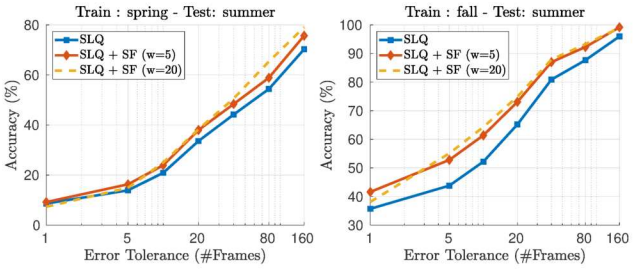

Figure 9. Performance with Sequence Filtering (SF).

5. Conclusions

We have presented a new quantization approach for snapshot-based visual localization, where the required storage footprint scales *sub-linearly* with the training data. Our method works well for datasets consisting of coherent sequences. Such datasets are typical in many snapshot-based visual localization problems such as autonomous driving. The method is developed around a hierarchical encoding scheme and a learning approach that jointly transforms the data and performs subspace clustering. Experimental results on large-scale datasets show that we achieve competitive performance in term of localization accuracy compared to existing state-of-the-art. Our work is one of the first proof-of-concepts for further work on sub-linear encoding methods. A prominent direction for future work is to extend and improve our method to enable it to work with non-sequential data. Such an extension can be achieved by replacing the current linear transformation with non-linear mappings where the parameters of the mapping do not violate sub-linearity.

Acknowledgements This work was supported by an Asian Office of Aerospace Research and Development Grant FA2386-16-1-4027 and an ARC Future Fellowship FT140101229 to Michael Milford.

References

[1] Relja Arandjelović, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. NetVLAD: CNN architecture for weakly supervised place recognition. In *CVPR*, 2016. 1, 6

[2] Artem Babenko and Victor Lempitsky. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence*, 37(6):1247–1260, 2015. 1

[3] Artem Babenko and Victor Lempitsky. Tree quantization for large-scale similarity search and classification. In *CVPR*, 2015. 2

[4] Artem Babenko and Victor Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2055–2063, 2016. 1

[5] Artem Babenko and Victor S Lempitsky. Annarbor: Approximate nearest neighbors using arborescence coding. 2

[6] Artem Babenko and Victor S Lempitsky. Product split trees. In *CVPR*, pages 6316–6324, 2017. 1, 2

[7] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992. 4

[8] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27, 2011. 4

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. 4

[10] Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. *Machine learning*, 47(2-3):201–233, 2002. 4

[11] Thanh-Toan Do, Tuan Hoang, Dang-Khoa Le Tan, Trung Pham, Huu Le, Ngai-Man Cheung, and Ian Reid. Binary Constrained Deep Hashing Network for Image Retrieval without Manual Annotation. In *IEEE Winter conference on Application of Computer Vision (WACV)*, 2019. 1

[12] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 43(1):55–81, 2015. 1

[13] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization for approximate nearest neighbor search. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2946–2953. IEEE, 2013. 1, 2, 6

[14] Allen Gersho and Robert M Gray. *Vector quantization and signal compression*, volume 159. Springer Science & Business Media, 2012. 1, 2

[15] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, 2013. 2, 6

[16] Tuan Hoang, Thanh-Toan Do, Dang-Khoa Le Tan, and Ngai-Man Cheung. Selective deep convolutional features for image retrieval. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 1600–1608. ACM, 2017. 1

[17] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE TPAMI*, 33(1):117–128, 2011. 1, 2, 6

[18] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, 2010. 1, 6

[19] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 861–864. IEEE, 2011. 1, 2

[20] Yannis Kalantidis and Yannis Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2321–2328, 2014. 2, 6

[21] Dang-Khoa Le-Tan, Huu Le, Tuan Hoang, Thanh-Toan Do, , and Ngai-Man Cheung. DeepVQ: A Deep Network Architecture for Vector Quantization. In *IEEE Computer Vision and Patern Recognition (CVPR) Workshop*, 2018. 2

[22] Stephanie Lowry, Niko Sünderhauf, Paul Newman, John J Leonard, David Cox, Peter Corke, and Michael J Milford. Visual place recognition: A survey. *IEEE Transactions on Robotics*, 32(1):1–19, 2016. 1

[23] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. 4

[24] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967. 2

[25] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is NP-hard. *Theoretical Computer Science*, 442:13–21, 2012. 2

[26] Dominik Mautz, Wei Ye, Claudia Plant, and Christian Böhm. Towards an optimal subspace for k-means. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 365–373. ACM, 2017. 4, 5

[27] Michael Milford, Chunhua Shen, Stephanie Lowry, Niko Suenderhauf, Sareh Shirazi, Guosheng Lin, Fayao Liu, Edward Pepperell, Cesar Lerma, Ben Upcroft, and Ian Reid. Sequence searching with deep-learnt depth for condition- and viewpoint-invariant route-based place recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2015. 6

[28] Michael J Milford and Gordon F Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1643–1649. IEEE, 2012. 2

[29] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: A versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015. 2, 3

[30] Filip Radenović, Giorgos Tolias, and Ondřej Chum. CNN image retrieval learns from BoW: Unsupervised fine-tuning with hard examples. In *ECCV*, 2016. 6

[31] Filip Radenović, Giorgos Tolias, and Ondrej Chum. Fine-tuning cnn image retrieval with no human annotation. *IEEE TPAMI*, 2018. 1

[32] Torsten Sattler, Will Maddern, Carl Toft, Akihiko Torii, Lars Hammarstrand, Erik Stenborg, Daniel Safari, Masatoshi Okutomi, Marc Pollefeys, Josef Sivic, et al. Benchmarking 6DOF outdoor visual localization in changing conditions. In *CVPR*, 2018. 1

[33] Torsten Sattler, Tobias Weyand, Bastian Leibe, and Leif Kobbelt. Image retrieval for image-based localization revisited. 1

[34] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, 2016. 1

[35] Akihiko Torii, Relja Arandjelovic, Josef Sivic, Masatoshi Okutomi, and Tomas Pajdla. 24/7 place recognition by view synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1808–1817, 2015. 1, 6

[36] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009. 4, 5

[37] Changchang Wu et al. Visualsfm: A visual structure from motion system. 2011. 1

[38] Eric P Xing, Michael I Jordan, Stuart J Russell, and Andrew Y Ng. Distance metric learning with application to clustering with side-information. In *Advances in neural information processing systems*, pages 521–528, 2003. 4

[39] Litao Yu, Adam Jacobson, and Michael Milford. Rhythmic representations: Learning periodic patterns for scalable place recognition at a sublinear storage cost. *IEEE Robotics and Automation Letters*, 3(2):811–818, 2018. 6