Robotics: Science and Systems 2022
New York City, NY, USA, June 27-July 1, 2022

SEER: Unsupervised and sample-efficient environment specialization of image descriptors

Peer Neubert and Stefan Schubert
Chemnitz University of Technology, Germany
{peer.neubert, stefan.schubert}@etit.tu-chemnitz.de

***Abstract*—Image descriptor based place recognition is an important means for loop-closure detection in SLAM. The currently best performing image descriptors for this task are trained on large training datasets with the goal to be applicable in many different environments. In particular, they are not optimized for a specific environment, e.g. the city of Oxford. However, we argue that for place recognition, there is always a specific environment – not necessarily geographically defined, but specified by the particular set of descriptors in the database. In this paper, we propose SEER, a simple and efficient algorithm that can learn to create better descriptors for a specific environment from such a potentially very small set of database descriptors. The new descriptors are better in the sense that they will be more suited for image retrieval on these database descriptors. SEER stands for Sparse Exemplar Ensemble Representations. Both sparsity and ensemble representations are necessary components of the proposed approach. This is evaluated on a large variety of standard place recognition datasets where SEER considerably outperforms existing methods. It does not require any label information and is applicable in online place recognition scenarios. Open source code is available.**[1]

I. Introduction

In the general visual place recognition setup, we are given a set of database images and one or multiple query images and the task is to find one or all images from the database that show the same place as the query images. The best performing methods today typically use deep learning based descriptors that were trained on large image datasets, e.g. NetVLAD [2], HybridNET [6] or HDC-DELF [28, 31]. Typically, these descriptors are optimized to work for a wide range of environments. Nevertheless, their performance is usually better if the application environment is more similar to the training data. A well known example is the observation that NetVLAD [2] descriptors trained on the urban Pitts30k dataset with images from the city of Pittsburgh do not perform well on the Nordland [37] dataset with its natural environment and large open areas. Presumably, NetVLAD trained on natural images more similar to the environment from the Nordland dataset would perform considerably better. Thinking further in this direction, we could also argue that it would be even better to train the NetVLAD descriptor (or any other descriptor) directly on the Nordland dataset to then apply it to the Nordland dataset. Of course, there are several impediments:

We want to thank Maren Gröne for the very valuable discussions. This work was supported by the Federal Ministry for Economic Affairs and Climate Action, Germany.

[1]https://www.tu-chemnitz.de/etit/proaut/SEER

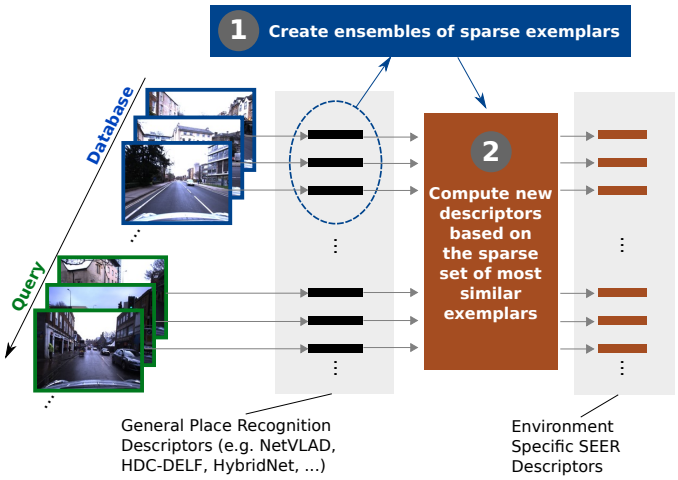

Fig. 1. The proposed SEER approach uses a set of database image descriptors from a particular environment (e.g. the city of Oxford) for an unsupervised creation of ensembles of sparse exemplars. These can be used to convert general place recognition descriptors (like NetVLAD or HDC-DELF) into environment-specific descriptors that are better suited for image retrieval in this environment.

- The typical descriptors [2, 6, 31] are trained in a supervised fashion. Thus, we would require at least some ground-truth knowledge about image pairs that show the same or different places.
- Deep learning based descriptors usually require large datasets for training.
- Training might be very time and energy consuming which hampers applicability, e.g. to loop-closure detection in an online SLAM approach.

While the latter two could partially be addressed by clever usage of pre-training and fine-tuning, the lack of ground-truth knowledge prevents the application of supervised learning. In particular, we are thinking of the practically very important application of loop-closure detection in visual SLAM: By the general definition of the SLAM [14] problem, we do not have (ground-truth) knowledge about associations between the database images or between database and query images (otherwise we would not need loop-closure detection). If we are in an online SLAM [36] problem, there are also restrictions regarding runtime and typically also energy consumption. This motivates the main research question of this work: *Is it possible to adapt, improve or fine-tune general place recognition descriptors for a particular environment (e.g. the city of Oxford) in a completely unsupervised fashion and with*

low sample (and runtime) complexity?

Especially due to the absence of any kind of labels or ground-truth information about associations in the database, there are only few existing approaches in the literature as will be discussed in Sec. II. The main contribution of this work is SEER, a novel algorithmic approach that addresses the above research question. It can learn to create environment-specific descriptors based on existing general place recognition descriptors from a potentially very small database without any label information about matchings in the database. These specialized descriptors are then better suited for matching against this particular database.

SEER stands for Sparse Exemplar Ensemble Representations. It is a greedy unsupervised learning algorithm, which is simple but not trivial. The underlying idea is to use an ensemble of sparsified versions of the known database descriptors as a basis representation for the new descriptors. Sec. III will present the algorithmic steps and discuss that the concepts of sparsity and ensemble representations are essential for its performance. This will then be experimentally validated in Sec. V. The experiments will also show a considerable improvement over existing methods and demonstrate the application of SEER in batch and online place recognition scenarios. Although SEER is designed for image descriptors and the task of place recognition, its properties are also appealing for other robotic tasks. This will be shortly outlined in the discussion of Sec. VI.

II. Related Work

Visual place recognition in changing environments is a subject of active research. An overview of existing methods was given by Lowry et al. [21] in 2016. The recent paper [25] supplements this overview with a survey on deep learning methods for visual place recognition. A detailed discussion of open questions and particular challenges can be found in [33].

Pipelines for visual place recognition often build upon pairwise comparison of holistic image descriptors like DenseVLAD [38], HybridNet [6], NetVLAD [2], or HDC-DELF [28]. To integrate these image retrieval methods into the robotics context of place recognition, there is a wide range of methods, e.g. [26, 35, 29], that exploit the structure of the robotics task or additional available information about the database and query set for performance improvements. For example, SeqSLAM [26] exploits sequences in the database and query set and ICM [35] additionally leverages descriptor similarities within these sets.

The proposed SEER builds upon the well established concept of ensembles [8] as is used in approaches like bagging [4], boosting [32, 10], or extremely randomized trees [11]. In SEER, ensembles are used as an internal representation of a particular environment created from a set of incoming image descriptors. This is strongly related to SLAM [14]. We want to emphasis two important points: First, the goal of SLAM is usually to infer a metric or semi-metric/topological map. SEER has no spatial representation of any kind. Even a topological map usually consists of connections between temporally neighbored places (e.g. from visual odometry) and connections from loop-closure detections. SEER only provides the latter. The second important point is that SEER is intended to be used as a part *within* a SLAM system, i.e. to perform loop-closure detection, or more specifically, create descriptors that are better suited for loop-closure detection in a particular environment.

There are other methods that exploit a set of known database descriptors to improve place recognition performance. For example, TF-IDF [7] statistics can be used to weight the descriptiveness of words from a vocabulary. A simple but effective technique is the feature-wise standardization of the database and query descriptors [34]. While in [34] the feature-wise means and standard deviations of the database and query descriptors were used, even a standardization of the query descriptors with the statistics of the database descriptors are sufficient for performance improvements.

Sparse coding [19] is a class of algorithms that is related to the idea of using descriptor vectors to learn a specialized representation: Given a matrix $X \in \mathbb{R}^{N \times d_X}$ of N descriptors with dimensionality d_X, its basic task is to learn a dictionary $D \in \mathbb{R}^{L \times d_X}$ with L basis vectors together with sparse coefficients $s \in \mathbb{R}^{N \times L}$ to minimize a reconstruction error $\|X - s \cdot D\|_2$. This dictionary D is comparable to the internal representation M of SEER. As presented in [19], there is a wide range of methods to optimize this error by inferring D and s. For instance, K-SVD [1] uses a singular value decomposition (SVD) to update the dictionary D before it adapts the coefficients s with an orthogonal matching pursuit (OMP). Wright et al. [39] use sparse representations from an overcomplete dictionary whose base elements are the training samples themselves. As argued in [24], sparse autoencoders [30] can be considered as an alternative approach that learns sparse coding using a neural network. In another direction, instead of increasing the sparsity of representations, Change Removal by Lowry and Milford [20] is an approach to use dimensionality reduction techniques to create better descriptors. They use a principal component analysis to remove the p most descriptive principal components from the input descriptors X, which are supposed to be mainly influenced by the environmental conditions.

The method MCN [29] is a neurologically inspired sequence processing method that is based on the ideas of the hierarchical temporal memory (HTM) theory by Hawkins [16]. MCN is composed of a spatial pooler [15] and a temporal memory [17]. While the temporal memory in MCN is responsible for the exploitation of sequence information, the spatial pooler only learns to encode input descriptors into sparse binary vectors based on previously seen vectors without any further assumptions. Therefore, the spatial pooler of MCN is related to SEER and will be compared in the experimental evaluation of Sec. V.

Similar to the spatial pooler [15], the hashing function CCA-ITQ [13] learns to map an input descriptor to a binary output descriptor. Jain et al. [18] showed that CCA-ITQ not

only learns a hashing of descriptors but concurrently improves their performance. However, it requires additional labels about which of the given descriptors show same places. To handle both viewpoint and appearance changes, descriptor pairs of the same place within the database but also between the database and query set should be provided. SEER does not require such label information.

III. Algorithmic Approach: SEER
Sparse Exemplar Ensemble Representation

This is the high level perspective: The input to SEER is a set of general purpose database descriptors X_{DB} of a particular environment (e.g. NetVLAD descriptors of images from the city of Oxford). The algorithm performs a simple greedy learning of an internal representation M that can then be applied to a descriptor x in order to create a better descriptor y. Where "better" means, y is better suited for image retrieval on the database.

A. Rationale behind SEER and its relation to Sparse Coding

The core idea of SEER is to use the known database descriptors as a new basis for a representation of all descriptors in this environment. Since the database descriptors are not linearly independent, they do not form an orthonormal vector basis but an overcomplete frame [9] (i.e. the subspace spanned by the database descriptors is (very likely) not changed if we remove one descriptor). To address this, SEER uses a sparse combination of vectors similar to Sparse Coding (SC) [19]. As was outlined in the related work section II, SC uses sparse linear combinations of basis vectors from a dictionary in order to minimize a reconstruction error. Instead of using a universal dictionary that is suitable for arbitrary input data, Sparse Coding can use training data to learn a tailored dictionary for a specific task (e.g. for face recognition [39]). The internal representation M of SEER is comparable to such a tailored dictionary. However, different to SC, SEER does not use linear combinations of basis vectors from the dictionary and does not use the proxy task of optimizing a reconstruction error. Instead, since the target task of place recognition builds on the similarity between descriptor vectors, SEER directly uses the similarity to the most similar vectors from the dictionary as sparse output representation.

Beside the sparsity in the output y there is a second level of sparsity in the basis vectors themselves. This allows a straightforward implementation of the concept of ensembles [8]: Instead of storing the single database descriptor, we propose to store a small set of randomly sparsified exemplars of this descriptor in the internal representation M. We use this ensemble to increase the robustness towards variations in descriptors of the same place. The experimental evaluation will demonstrate the contribution of both sparsity and ensembles to the overall performance.

B. Input x, output y and the internal representation M

Input to SEER are standard general purpose place recognition descriptors $x \in X_{DB}$, each is a d_X dimensional vector. Sec. V will show experiments with NetVLAD [2], HybridNet [6] and HDC-DELF [28] (only one descriptor type is used at a time). Using descriptors with distributed representations [16] is preferable (where each piece of information is spread across many or all dimensions), e.g. HDC-DELF. A non-distributed descriptor can be easily converted to a distributed representation with a Gaussian random projection [28]. Input descriptors have to be L2 normalized.

The main internal representation is a matrix M where each column is an exemplar, i.e. a sparse vector from $\mathbb{R}^{d_X}$ with d_M many non-zero entries. The length of the output y will be equal to the number of columns in M. The number of non-zero elements in y is controlled by two parameters: k (which is the minimum number of exemplars per input descriptor) and λ (a factor to increase the density in the output).

Algorithm 1:
Sparse Exemplar Ensemble Representation (SEER)

Input: • a L2-normalized input descriptor $x \in \mathbb{R}^{d_X}$, beneficial if this is a distributed representation
• a (potentially empty) list M of known *sparse* exemplars $\in \mathbb{R}^{d_X}$, each with d_M non-zero entries, $d_M \ll d_X$
• a flag $updateM\_flag$ to toggle the extension of M with new exemplars

Parameters: • d_M: the number of non-zero entries in the sparse exemplars in M (default: $d_M = 200$)
• k: the minimum number of similar exemplars for each input descriptor (default: $k = 50$)
• λ: the reactivation factor, $\lambda \cdot k$ is the maximum number of non-zero entries in the output y (default: $\lambda = 2$)

Output: • y a sparse output descriptor based on similarities to M
• the potentially updated M

function *[y, M] = createSEER(x, M, updateM_flag)*

```
// Compare x to each element from M using
   dot product
S = x^T · M

// Extend M with new sparse exemplars if
   requested, this will also extend S
if updateM_flag then
    // Count similarities above an expected
       similarity
    expectedSimilarity = d_M / d_X
    c = count(S ≥ expectedSimilarity)

    // Create new exemplars if c < k
    for i=c; i < k; i = i + 1 do
        // sample a new sparse exemplar and
           append to M
        M = append(M, sample(x, d_M))

        // update S with the similarity to
           the new exemplar
        S = append(S, x^T · M_end)

// Only keep the λ·k highest similarities
   and set all others to zero
y = sparsify(S, λ · k)

return y, M
```

C. Algorithmic steps

The computational steps of SEER are listed in Alg. 1. This simple and efficient algorithm creates both M as well as the output descriptor y and works as follows:

For a general application scenario, let us assume we are given a set of general place recognition database descriptors X_{DB}. We can start with an empty M and iteratively call the function *createSEER* with the descriptors $x \in X_{DB}$ to incrementally build M while simultaneously computing the output descriptor y for the current x (alternatives will be discussed in Sec. III-D).

Computing y: SEER first computes the vector S of similarities between the input x and each of the existing exemplars in M in line 2. We use the dot product as similarity measure. In combination with the sparsity of the exemplars, this allows for a fast implementation. This choice is further discussed in Sec. III-E. To also sparsify this similarity vector S (as a second level of sparse vectors), we only keep the $\lambda \cdot k$ highest similarities and set all other values to zero in line 9. λ and k are two parameters of the algorithm that are strongly connected to the way how M is created; they will also be discussed in Sec. III-E. If SEER is set up to not update the internal representation M (lines 6-8), these two steps already compute the output descriptor that can be used for place recognition as described in Sec. III-D.

Extending M (and y): The underlying idea is that for each input descriptor x, there should be at least k similar exemplars in M. We ensure this with a greedy approach and simply create the missing number as new exemplars in M (the loop in lines 6-8). To count the number of existing similar exemplars, their similarities are compared to an *expected similarity* (line 5). This value depends on the used similarity measure from line 2. Since we use the dot product, we can compute the expected similarity as d_M/d_X (the sparsity of the exemplars). For example, during the first call of this function (with an empty M) k new exemplars will be created.

To create a new exemplar, the *sample* function in line 7 randomly selects d_M dimensions from the input descriptor x. The probability for dimension i to be sampled is proportional to the absolute value $|x_i|$. Thus, dimensions with a high positive or negative activation are more likely represented in the exemplars for this input descriptor. We simply linearly map the range between the lowest and highest absolute values to the interval $[0, 1]$.

If new exemplars are added to M for input x, then this is also reflected in the similarities S in line 8 by appending the similarity to this new exemplar. After all new exemplars have been created and incorporated in S, the sparse set of highest similarities in S becomes the output descriptor y (line 9).

D. How to apply SEER: Batch and online scenarios

We distinguish two place recognition scenarios for SEER:

- **Batch**: The database is given as a whole and we can create the representation M on the complete dataset. Query image descriptors might come in a batch or individually.
- **Online**: We start from an empty database and iteratively add incoming image descriptors to the database and the goal is to compare the current image descriptor to all previously seen descriptors.

In the **online** case, we start from an empty M and iteratively call the *createSEER* function on incoming descriptors to immediately obtain the output descriptor y that can then be compared to all previous descriptors (e.g. using standard cosine similarity).

To address the increasing size of M over time and accordingly the larger length of later created y vectors, missing entries are filled with zeros for the comparison with longer vectors. This has only limited influence on the results since the sparse y vectors have the same number of non-zero entries ($\lambda \cdot k$), independent of their length.

However, we still want to address the different amount of information in each dimension of y: A particular dimension of y is associated with a particular exemplar in M that was created at some point in time. Very importantly, in the online scenario, only input descriptors after that point in time are compared to this exemplar. So only the first exemplars are compared against all input descriptors, later created exemplars only to fewer. This number decreases roughly linearly over the creation time and thus over according dimensions in y. To reflect this, we apply the following linear weighting to y:

$$\tilde{y}_i = y_i \cdot \frac{|y| - i + 1}{|y|} \text{ for } i = \{1, ..., |y|\} \tag{1}$$

Where y_i is the i-th dimension of y and $|y|$ is the number of elements in y after appending the required zero values to achieve the target length. $\tilde{y}$ is then used for place recognition.

In the **batch** case, there is no need to simultaneously compute M and y for the database descriptors. Here we propose to run the database twice through the *createSEER* function: The first iteration over all database descriptors is used to create M (the output y can be ignored). The second run over all database descriptors is without updating M but uses the previously computed M to get the y descriptor for the database. For the query descriptors, only a single run in the same configuration as the second database run is required. In this configuration, the weighting from eq. 1 is not necessary since all database and query descriptors have been compared against all exemplars.

Of course, it is possible to enable updates of M again, if we expect exploration of new environments that we want to reflect in M for later recognition.

E. Parameters and computational complexity

SEER has three parameters, d_M, k, and λ. Their basic meaning and default values are listed in the header in Alg. 1. Their values are not sensitive to the particular dataset or input descriptor. We use the same default values in all experiments (of course, except for the parameter evaluation). The factor $\lambda = 2$ is introduced to reactivate more exemplars ($\lambda \cdot k$ many) than are created for a descriptor (k many). This is useful to

reduce the number of created exemplars, since the size of M is the main influence on the runtime and storage complexity.

An important design decision is the usage of the dot product for similarity computation. We prefer this over the usual cosine similarity because the dot product omits the normalization of the vector lengths. This normalization would dominate the computational effort for SEER since each exemplar uses a different set of dimensions of the input x, thus cosine similarity would require to individually normalize each of these sparsified versions of x. However, we L2 normalize the complete input descriptor x once before usage in SEER.

Due to the usage of sparse representations, the overall number of operations and memory requirement in SEER is of the same order of magnitude as a straight-forward comparison of the original database and query descriptors. This is elaborated in the Appendix. Very importantly, approximate matching techniques that speed up the comparison of original descriptors can potentially also speed-up the similarity computation in SEER – unless they conflict with the sparsity of the exemplars.

IV. Experimental Setup

A. Datasets and evaluation metric

We evaluate the image retrieval performance of SEER on standard place recognition datasets from mobile robotics. For evaluation, we compute similarity matrices between database and query images and compare them to ground-truth knowledge about place matchings using a series of thresholds [27]. We report the average precision (AP) computed as area under the resulting precision-recall curve. AP is a typical measure in image retrieval. It evaluates the balance between precision and recall and allows to evaluate multiple loop closures for each query (e.g. to add more constraints to the pose graph optimization in SLAM).

We use 23 sequence combinations from six datasets with different characteristics regarding environment, appearance changes, single or multiple visits of places, possible stops, or viewpoint changes: **Nordland1000** [37], **StLucia** (Various Times of the Day) [12], **CMU** Visual Localization [3], **GardensPointWalking**[2], **OxfordRobotCar**[3] [22], and **SFU-Mountain** [5]. For OxfordRobotCar, we sampled sequences at 1Hz; where available we used the accurate ground-truth data [23]. For Nordland1000, we sampled 1,000 images of unique places from each season (without tunnels).

B. Compared approaches and implementations

Our method is compared to five approaches from the literature – STD [34], K-SVD [1], sparse autoencoders [30], change removal [20], and MCN [29] – and to a custom implementation for sparse coding [19]. To account for potentially non-optimal parameter tuning for the baseline algorithms, they partially use an oracle that selects the best parameter setting for each dataset from a limited number of possibilites. Of course, this is not applied to our proposed approach.

For each dataset, all methods were trained on the corresponding database descriptors X_{DB}. **STD** [34] is implemented as simple dimension-wise mean-centering with the mean over all database descriptors. The same database mean is applied to both the database and the query descriptors. For **kSVD** [1], we use the group's implementation[4] to learn a dictionary D with a varying number of atoms and coefficients s with 100 non-zeros elements. The sparse autocoder **SAE** [30] is based on the Matlab implementation[5] and is composed of two layers with a *satlin*-activation function after the encoder. The sparse autoencoder was trained to transform an input descriptor into a 1000-dimensional sparse representation. Change removal **CR** [20] uses a principal component analysis (PCA) to remove the first p principle components with the highest eigenvalues. In our implementation, a singular value decomposition (SVD) with $p = 1$ or $p = 2$ was used. For **MCN** [29], we built upon the author's implementation[6] and use the set of active mini-columns from the spatial pooler as the output descriptors (with a separate training run on the database). Our sparse coding **SC** [19] implementation minimizes the squared reconstruction error $\|X_{DB} - s \cdot D\|_2^2$ with an expectation maximization like iterative approach: Using a least-squares optimization, each iteration first computes an optimal dictionary D before the coefficients s are updated. After each update of s, it is sparsified to the 50 (CMU, Oxford) or 100 (otherwise) highest values. The number of atoms in D was set to $0.3 \cdot |X_{DB}|$.

C. Setup of SEER

We combine SEER with the following descriptors: NetVLAD **NV** [2]: We use the authors' VGG-16 version[7] with whitening trained on the Pitts30k dataset (4,096-D). HybridNet **HN** [6]: We use the authors' version[8] (43k-D descriptors). HDC-DELF **HDC** [28]: We use the authors' version[9] in combination with the TensorFlow Hub[10] implementation of DELF [31] (4,096-D). For SEER, each descriptor is first converted to a 4,096-D distributed representation using a Gaussian random projection and then L2 normalized. In the batch scenario, SEER uses STD on the input descriptors as described above. STD is not applied in the online scenario. In all experiments, the default parameters from Alg. 1 are used.

V. Results

A. Image retrieval performance for place recognition

Table I shows results of SEER in combination with NetVLAD, HybridNet and HDC-DELF input descriptors and in comparison to other techniques. If a batch of database descriptors is available and can be used for standardization,

[2] goo.gl/tqmWyq

[3] Used Oxford datasets: 2014-11-25-09-18-32, 2014-12-09-13-21-02, 2014-12-16-18-44-24, 2015-02-03-08-45-10, 2015-05-19-14-06-38, 2015-08-28-09-50-22

[4] http://www.cs.technion.ac.il/~ronrubin/software.html

[5] https://de.mathworks.com/help/deeplearning/ref/trainautoencoder.html

[6] https://www.tu-chemnitz.de/etit/proaut/seqloc

[7] github.com/Relja/netvlad

[8] github.com/scutzetao/DLfeature_PlaceRecog_icra2017

[9] https://www.tu-chemnitz.de/etit/proaut/hdc_desc

[10] tfhub.dev/google/delf/1

TABLE I

AVERAGE PRECISION FOR THREE DIFFERENT DESCRIPTORS COMBINED WITH DIFFERENT METHODS FOR ENVIRONMENT SPECIALIZATION. COLORED ARROWS INDICATE LARGE ($\geq$25% BETTER/ WORSE) OR MEDIUM ($\geq$5%) DEVIATION COMPARED TO THE DESCRIPTORS WITH STANDARDIZATION (+STD). BOLD NUMBERS INDICATE THE BEST PERFORMANCE PER ROW.

			NetVLAD [2]			**HybridNet** [6]			**HDC-DELF** [28]								
			Raw	**+STD**	**+SEER**	**Raw**	**+STD**	**+SEER**	**Raw**	**+STD**	**+kSVD**	**+CR**	**+SC**	**+SAE**	**+MCN**	**+SEER**	**+SEER simultaneous**
	Database	**Query**		[34]	[ours]		[34]	[ours]		[34]	[1]	[20]	[19]	[30]	[29]	[ours]	[ours]
GardensPoint	day_right	day_left	0.97	0.99	**1.00** →	0.59	0.56	0.75 ↑	0.79	0.82	0.89 ↗	0.81 →	0.87 ↗	0.80 →	0.91 ↗	0.89 ↗	0.86 ↗
Walking	day_right	night_right	0.51	0.59	0.65 ↗	0.50	0.62	0.76 ↑	0.71	0.79	0.87 ↗	0.80 →	0.86 ↗	0.84 ↗	0.86 ↗	**0.90** ↗	0.88 ↗
	day_left	night_right	0.40	0.48	0.50 →	0.18	0.22	0.33 ↑	0.40	0.47	**0.56** ↗	0.46 →	0.45 ↘	0.45 ↘	0.51 ↗	**0.56** ↗	0.55 ↗
OxfordRobotCar	2014-12-09	2015-05-19	0.78	0.89	0.98 ↗	0.25	0.62	0.93 ↑	0.88	0.91	0.76 ↘	0.92 →	0.93 →	0.92 →	0.98 ↗	**0.99** ↗	0.97 ↗
	2014-12-09	2015-08-28	0.60	0.66	**0.94** ↑	0.09	0.33	0.60 ↑	0.44	0.71	0.42 ↓	0.72 →	0.81 ↗	0.68 →	0.85 ↗	0.91 ↑	0.86 ↗
	2014-12-09	2014-11-25	0.87	0.91	0.97 ↗	0.41	0.69	0.86 ↑	0.71	0.82	0.78 →	0.82 →	0.93 ↗	0.87 ↗	0.95 ↗	**0.98** ↗	0.96 ↗
	2014-12-09	2014-12-16	0.55	0.11	0.14 ↗	0.08	0.40	0.54 ↑	0.52	0.80	0.18 ↓	0.81 →	0.53 ↓	0.62 ↘	0.78 →	**0.89** ↗	0.85 ↗
	2015-05-19	2015-02-03	0.92	0.93	**0.98** ↗	0.42	0.72	0.95 ↑	0.67	0.78	0.53 ↓	0.79 →	0.96 ↗	0.83 ↗	0.93 ↗	0.97 ↗	0.94 ↗
	2015-08-28	2014-11-25	0.61	0.59	0.70 ↗	0.11	0.35	0.52 ↑	0.64	0.71	0.65 ↘	0.73 →	0.81 ↗	0.77 ↗	0.77 ↗	**0.85** ↗	0.84 ↗
SFUMountain	dry	dusk	0.33	0.48	0.63 ↑	0.58	0.62	0.71 ↗	0.68	0.81	0.86 ↗	0.81 →	0.74 ↘	0.85 ↗	0.85 ↗	**0.88** ↗	**0.88** ↗
	dry	jan	0.19	0.22	0.31 ↑	0.28	0.51	0.57 ↑	0.51	0.57	0.68 ↗	0.63 ↗	0.52 ↘	0.61 ↗	0.63 ↗	**0.70** ↗	0.69 ↗
	dry	wet	0.22	0.40	0.54 ↑	0.53	0.57	0.65 ↗	0.63	0.75	0.84 ↗	0.76 →	0.71 ↘	0.77 →	0.82 ↗	**0.87** ↗	0.86 ↗
CMU	20110421	20100901	0.73	0.71	0.81 ↗	0.55	0.60	0.77 ↑	0.70	0.75	0.67 ↘	0.77 →	0.76 →	0.79 →	0.82 ↗	**0.83** ↗	0.79 →
	20110421	20100915	0.77	0.77	**0.80** →	0.67	0.70	0.77 ↗	0.69	0.73	0.60 ↘	0.74 →	0.70 →	0.73 →	0.78 ↗	0.78 ↗	0.74 →
	20110421	20101221	0.56	0.54	0.49 ↘	0.40	0.47	0.61 ↑	0.64	0.64	0.39 ↓	0.63 →	0.68 ↗	0.66 →	0.64 →	**0.70** ↗	**0.70** ↗
	20110421	20110202	0.61	0.62	0.72 ↗	0.37	0.46	0.65 ↑	0.67	0.72	0.52 ↓	0.74 →	0.75 →	0.72 →	0.77 ↗	**0.81** ↗	0.80 ↗
Nordland1000	spring	winter	0.01	0.02	0.08 ↑	0.78	0.77	0.76 →	0.68	0.77	0.79 →	0.80 →	0.73 →	0.82 ↗	0.73 →	**0.89** ↗	**0.89** ↗
	spring	summer	0.06	0.20	0.40 ↑	0.75	0.80	**0.84** ↗	0.59	0.74	0.77 →	0.77 →	0.75 →	0.76 →	0.72 →	0.83 ↗	**0.84** ↗
	summer	winter	0.01	0.05	0.08 ↑	0.55	0.70	**0.76** ↑	0.36	0.45	0.55 ↗	0.57 ↑	0.49 ↗	0.48 ↗	0.46 →	0.59 ↑	0.59 ↑
	summer	fall	0.22	0.53	0.74 ↑	0.90	**0.94**	**0.94** →	0.80	0.90	0.93 →	0.92 →	0.90 →	0.92 →	0.84 ↘	**0.94** →	**0.94** →
StLucia	100909_0845	180809_1545	0.02	0.08	0.10 ↑	0.43	0.49	**0.59** ↑	0.30	0.46	0.50 ↗	0.47 →	0.50 ↗	0.48 ↗	0.54 ↗	0.56 ↗	0.52 ↗
	100909_1000	190809_1410	0.07	0.19	0.26 ↑	0.52	0.61	0.71 ↑	0.43	0.64	0.66 →	0.64 →	0.65 →	0.65 →	0.72 ↗	**0.74** ↗	0.72 ↗
	100909_1210	210809_1210	0.51	0.61	0.62 →	0.59	0.66	0.75 ↑	0.52	0.70	0.75 ↗	0.71 →	0.75 ↗	0.72 →	**0.77** ↗	**0.77** ↗	0.75 ↗
Worst case			0.01	0.02	0.08 ↑	0.08	0.22	0.33 ↑	0.30	0.45	0.18 ↓	0.46 →	0.45 →	0.45 →	0.46 →	**0.56** ↗	0.52 ↗
Best case			0.97	0.99	**1.00** →	0.90	0.94	0.95 ↗	0.88	0.91	0.93 →	0.92 →	0.96 ↗	0.92 →	0.98 ↗	0.99 ↗	0.97 ↗
Average case (mAP)			0.46	0.50	0.58 ↗	0.46	0.58	0.71 ↑	0.61	0.71	0.66 ↘	0.73 →	0.73 →	0.73 →	0.77 ↗	**0.82** ↗	0.80 ↗

then for all three input descriptors, there is a considerable improvement of using STD over the raw descriptors. The combination of input descriptors with STD will be considered as the baseline due to its simplicity, performance, and wide usage (i.e., the colored arrows in Table I compare against STD).

Most importantly for this work, the combination with SEER can considerably improve the average performance over the STD baseline for all input descriptors in this batch scenario as indicated by the colored arrows. Only for the combination with NetVLAD, there is one case with a noticeable degradation. In this comparison, NetVLAD is the worst performing input descriptor. The benefit from the combination with SEER further improves with better descriptors. The overall best combination is SEER applied on HDC-DELF descriptors. Therefore, further evaluations will only include this combination.

To evaluate the performance in an online scenario, the last column in Table I shows the evaluation of a single run of *createSEER* on all database images. This run simultaneously creates the SEER-descriptors for the database and the exemplars in M. We then use the same M to compute SEER-descriptors for the query set. This is slightly different from the description of "online" in Sec. III-D since we use separate database and query sets. However, this allows to directly compare the results to the batch case. Experiments exactly following the online setup in Sec. III-D will be presented in Sec. V-E. In this online-like experiment from Table I, the overall performance decreases slightly but is still considerably better than the baseline.

B. Comparison to other approaches

Table I also shows the comparison to other approaches from the literature. For all sequence comparisons SEER provides the best performance. The colored arrow beside each approach illustrates the comparison to the STD baseline. For each of the compared algorithms, we tested whether they performed better either with or without additional standardization before running the approach. We then use the better setup for all sequence comparisons in Table I. Except for kSVD, all approaches can further improve the performance over the STD baseline. The best approach from the literature is the spatial pooler from MCN with $mAP = 0.77$ compared to SEER with $mAP = 0.82$. The runtime of the available MCN implementation is considerably higher than of SEER (about factor 10). An interesting property of MCN is that the output are sparse binary vectors with a very low memory footprint. An accordingly modified version of SEER is part of the following ablation study.

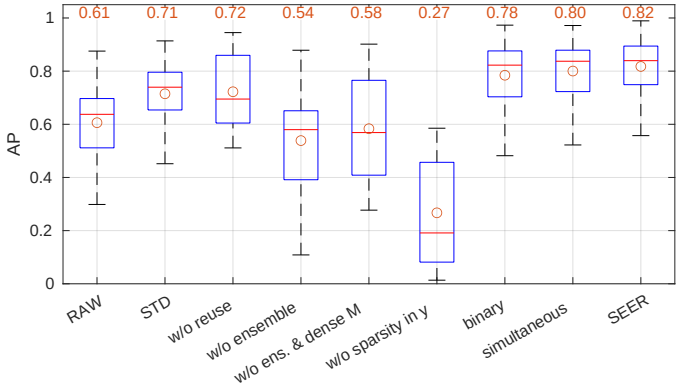

Fig. 2. Ablation study over all datasets in the batch case with HDC-DELF. Orange circles and text are mAP values for comparison with Tab. I.

C. Ablation study

Fig. 2 shows boxplots and mAP values over all 23 sequence comparisons from Table I for different variants of SEER in combination with HDC-DELF input descriptors. For comparison, it also shows the statistics for the original HDC-DELF descriptors (RAW) and with standardization (STD). The performance of the full SEER is shown on the very right ($mAP = 0.82$).

A first modification is to not reuse exemplars but create an individual set of exemplars for each incoming descriptor. This is called "w/o reuse" in Fig. 2 and implemented by setting the number of existing similar exemplars $c = 0$ in line 5 of Alg. 1. This considerably decreases the performance ($mAP = 0.72$) and is only slightly better than the baseline ($mAP = 0.71$).

If we remove the ensemble ("w/o ensemble", implemented by setting $k = 1$), the performance falls below the baseline and even below the raw descriptors. In variants without ensemble, it is not necessary to use sparsity within the exemplars (which is used in SEER to create different versions of the input descriptor). Using dense exemplars ("w/o ens. & dense M", implemented by setting $k = 1$ and $d_M = d_X$) can only slightly improve over the previous variant and achieves about the performance of raw descriptors. Thus, the ensemble, i.e. using multiple variants of each input descriptor, is considered a necessary part of SEER.

The second necessary part is the sparsity of the output descriptors. If we use the dense vector of all similarities to exemplars ("w/o sparsity in y"), the performance significantly drops below the raw descriptors ($mAP = 0.27$).

A potentially interesting variant is to output not the similarity to the exemplar but the sparse binary vector that indicates the $\lambda \cdot k$ most similar exemplars ("binary"). This smaller output descriptor still achieves a performance considerably above the baseline ($mAP = 0.78$).

So far in this comparison, all SEER variants use independent runs to create the exemplars M and then the database descriptors y. If both is done simultaneously ("simultaneous", same as in the last column in Table I), the performance moderately decreases from $mAP = 0.82$ to $mAP = 0.80$.

D. Environment-specific nature

To demonstrate that the performance improvement is due to the adaptation to the specific environment, we ran a series of experiments where we created the exemplars M from a different environment than the database and query sets. So far, we used the database set of each sequence comparison to generate individual exemplars M for each environment. If we instead, for example, use the exemplars M created from the first sequence of the Oxford dataset for *all* sequence comparisons, the performance decreases to $mAP = 0.43$ (in particular, this is also worse than the raw descriptor). We repeated the same experiment using the first sequence of each dataset to generate the exemplars M for all comparisons:

- M always from GardensPoint day_right: mAP=0.29
- M always from Oxford 2014-12-09: mAP=0.43
- M always from SFUMountain dry : mAP=0.21
- M always from CMU 20110421: mAP=0.27
- M always from Nordland1000 spring: mAP=0.36
- M always from StLucia 100909_0845: mAP=0.28
- M from the *specific environment* (database): **mAP=0.82**

The performance considerably decreases in all cases if the exemplars in M are not generated for the specific environment.

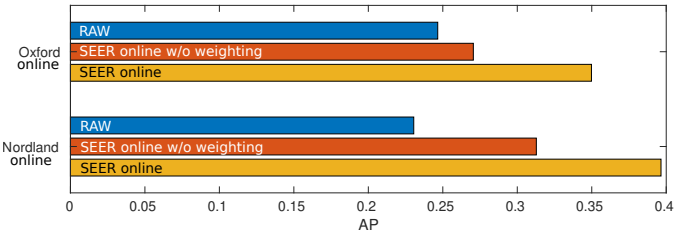

Fig. 3. Evaluation of the online scenario. All sequences of the Oxford and Nordland datasets are concatenated to form two sequences with multiple loops. 'Weighting' refers to eq. 1.

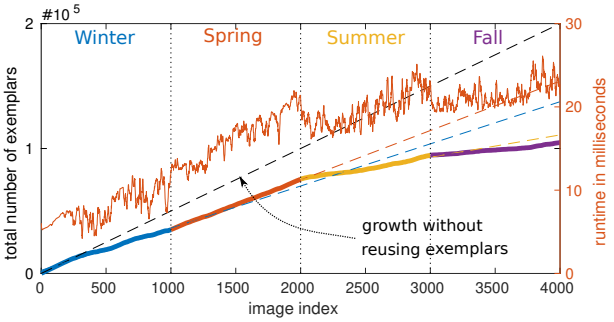

Fig. 4. Evaluation of the growth of the number of exemplars when consecutively processing Nordland winter, spring, summer, and fall images. The black dashed line illustrates the theoretical growth if no exemplars would be reused. The colored dashed lines show fitted lines that illustrate the average growth in each season.

E. Online application and computational effort

Although the "simultaneous" variant from the ablation study in Sec. V-C compared a database set to a disjoint query set, the application of SEER is already very similar to the online scenario as it was described in Sec III-D. To evaluate the exact online scenario of an iteratively increasing database that is compared to itself, we created two datasets with multiple loops by concatenating all sequences from the Oxford and Nordland datasets. Images are iteratively given to the place recognition algorithm and compared to all previous images. Standardization is not possible, since this requires the batch of database descriptors.

Fig. 3 compares the performance of the raw descriptors with a single run of SEER that simultaneously created the exemplars and the output descriptors. We also additionally evaluate the influence of removing the weighting from SEER that was described in eq. 1. The SEER approach can considerably improve the place recognition performance on both datasets and the weighting of the descriptor dimensions makes a significant contribution to the performance.

Fig. 4 illustrates how the total number of exemplars increases over time. On average, only half of the potential k exemplars were created (as illustrated by the dashed black line). The runtime is roughly proportional to the number of stored exemplars. In this experiment, the runtime of SEER for the final images was about 23 milliseconds per image using a moderately optimized Matlab implementation on a standard desktop CPU (AMD Ryzen 7 3700X).

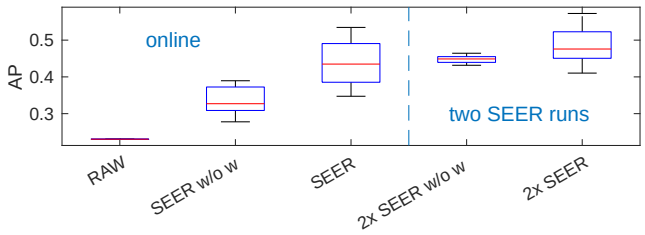

Fig. 5. Evaluation over all 24 possible orderings of the four Nordland sequences. E.g. spring-summer-fall-winter, summer-spring-winter-fall, ...)

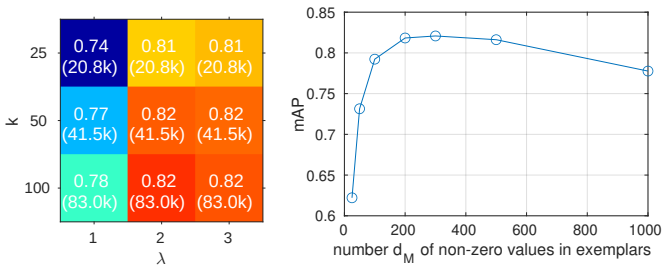

Fig. 6. Parameter evaluation. *(left)* mAP values of different combinations of values for k and λ. The numbers in parentheses are the numbers of exemplars (in thousands). *(right)* mAP values for different levels of sparsity in the exemplars as indicated by number of non-zero elements d_M.

One can clearly see that the reuse of exemplars already started within the winter sequence, but then decreased with beginning of the spring season with new appearances of places. Presumably due to the higher visual similarity to previous seasons and the larger set of already existing exemplars, a higher rate of previous exemplars was reused again during summer and fall. Due to the iterative nature of the SEER algorithm, the ordering of incoming descriptors influences the creation and reuse of exemplars. To evaluate this influence, Fig. 5 shows performance statistics over all 24 possible orderings of the four Nordland sequences. The RAW descriptor is not influenced by the ordering. Similar to the previous results, the online SEER approach can considerably improve the performance over RAW descriptors and dimension weighting (cf. eq. 1) is valuable. However, the boxplots clearly show the large influence of the different orderings which can result in considerably varying place recognition results. Noticeably, if we have the opportunity to make a second run of SEER to recompute the database descriptors like in the batch scenario, this does significantly reduce the variance of the results. Then there is also only little benefit of additional dimension weighting which again increases the variance.

F. Parameter evaluation

Finally, Fig. 6 evaluates different values of the parameters k (minimum number of exemplars per input descriptor), λ (the factor on k to control the number of non-zero elements in the output descriptors), and d_M (the number of non-zero elements in each exemplar). All but the varied parameters are set to the default values stated in Alg. 1. Increasing the minimum number of exemplars per input descriptor improves the performance. However, this comes at the cost of increased number of exemplars and the according increase in runtime and memory consumption. The factor $\lambda > 1$ increases the number of exemplars that are represented in the output without increasing the overall number of exemplars. This helps to maintain a similarly good performance with fewer exemplars but comes at the cost of increased density of the output descriptor.

The right part of this figure evaluates the influence of the sparsity in the exemplars. There is a clear sweet-spot around the selected value of $d_M = 200$. This resulting 5% density of representations (200 from 4,096 dimensions) is in line with findings about sparsity in biological systems [16].

VI. Discussion and Conclusion

SEER is a simple algorithm to learn to create environment-specific descriptors from a small set of database descriptors. It builds on the concepts of sparsity and ensembles. The experimental evaluation showed that both components are necessary for the overall performance and that SEER considerably improves the performance compared to existing approaches. Of course, creating a specialized descriptor for a particular environment reduces its applicability to other environments. We do not expect a SEER descriptor to work well in any other environment than that of the used database.

The output are sparse descriptors that allow a fast comparison. However, we do not consider SEER to be a technique to speed-up place recognition or to reduce the memory footprint of descriptors. The computational effort and storage requirements are moved from the comparison of the output descriptors to the comparison with the exemplars and their storage. The runtime and memory increase with the size of the processed database. Very importantly, the total number of operations and the memory footprint is of the same order of magnitude to what is required for a direct comparison of the original descriptors against the database. A situation where SEER could considerably reduce the memory footprint is when the number of stored query descriptors is (much) larger than the number of database descriptors.

Currently, the main benefit of SEER is the considerably improved image retrieval performance for place recognition at low additional complexity. With a broader perspective, SEER can potentially also be used for learning to create representations for other domains and other tasks. The combination of unsupervised specialization learning from small sample sizes could, for example, potentially be interesting for anomaly detection or out-of-distribution detection. The online capability could be interesting for continual learning tasks. Both directions are highly relevant for robotic applications in open world scenarios. However, our current experiments are limited to image descriptors and place recognition. Based on the good performance on this task, the foundation on established working principles (sparsity and ensembles), and the simplicity of the approach, we hope this is a valuable basis for further research and broader application.

APPENDIX: RUNTIME AND MEMORY COMPLEXITY

This section compares the number of operations (NOO) and memory requirement of SEER and the baseline of straightforward comparison of an original query descriptor vector to the database based on pairwise cosine similarity. The NOO for the baseline of comparing a d_X-dimensional query vector against a database $X_{DB} \in \mathbb{R}^{d_X \times |X_{DB}|}$ can be estimated by

$$NOO_{baseline} \geq 2 \cdot d_X \cdot |X_{DB}| \tag{2}$$

This estimate is based on cosine similarity computed by dot product of normalized database and query vectors. The right side ignores the effort for normalization and, for simplicity, assumes one addition and one multiplication per dimension comparison between any pair of vectors.

For SEER, the main computational effort is the similarity comparison in line 2 of Alg. 1 which requires $|M|$ many vector comparisons. When using the dot product as similarity measure, it is important to keep in mind that the effort for each comparison is not influenced by the size of the input descriptor (e.g. 4,096-D in the experiments) but only by the number of non-zero entries in exemplars ($d_M = 200$ in the experiments). For an exemplar $m \in M$ the computation of dot-product similarity $S_i = \sum_{m_j \neq 0} x_j \cdot m_j$ then requires d_M multiplications and $d_M - 1$ additions. As above for $NOO_{baseline}$, we will treat this as $2 \cdot d_M$ many operations. The overall NOO for comparison to M is then

$$NOO_{SEER} = 2 \cdot d_M \cdot |M| \leq 2 \cdot d_M \cdot |X_{DB}| \cdot k \tag{3}$$

The inequality is based on the observation that the upper bound for the number of exemplars $|M|$ is to create k new exemplars for each descriptor from the database X_{DB}, that is $|M| \leq |X_{DB}| \cdot k$.

Therefore, the increase in computational effort is:

$$\frac{NOO_{SEER}}{NOO_{baseline}} \leq \frac{d_M \cdot k}{d_X} = \frac{200 \cdot 50}{4,096} = 2.4 \tag{4}$$

The numbers are taken from our experimental setup. In practice, many exemplars will be reused. For the example experiment in Fig. 4, the actual number of exemplars is only 52 % of this upper bound. Therefore the overall NOO reduces to $0.52 \cdot (200 \cdot 50)/4,096 = 1.27$ times the NOO of comparing the initial descriptors.

However, for SEER we still have to compare the final descriptors for place recognition. Since they are sparse with at most $\lambda \cdot k$ non-zero elements, the additional NOO is comparatively small: $2 \cdot \lambda \cdot k \cdot |X_{DB}|$, which is $\frac{\lambda}{d_M} \cdot NOO_{SEER} = \frac{2}{200} \cdot NOO_{SEER} = 0.01 \cdot NOO_{SEER}$. So the majority of the computational effort is shifted into the creation of the SEER descriptor and the final comparison adds only 1% extra effort.

It is important to keep in mind that approaches that speed up the comparison for the baseline (e.g. approximate nearest neighbor techniques), can potentially also speed up the computation within SEER – unless they conflict with the sparsity of the exemplars. On the downside, due to the sparsity of the representations, cache effects of standard computing hardware are presumably used less efficiently.

Similar estimates can be made for the memory consumption. For the initial database descriptors it is $MEM_{X_{DB}} = d_X \cdot |X_{DB}|$. The memory consumption of indices and values of M in SEER is: $MEM_M \leq 2 \cdot d_M \cdot k \cdot |X_{DB}|$. This results in twice the ratio $\frac{MEM_M}{MEM_{X_{DB}}} \leq \frac{2 \cdot d_M \cdot k}{X_{DB}} = 4.8$ as for the runtime in eq. 4, with the same reduction in practice (e.g. to about $2 \cdot 1.27 = 3.54$ in the experiment from Fig. 4).

REFERENCES

[1] M. Aharon, M. Elad, and A. Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, 2006. doi: 10.1109/TSP.2006.881199.

[2] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. NetVLAD: CNN architecture for weakly supervised place recognition. *Transations on Pattern Analysis and Machine Intelligence*, 40(6), 2018. doi: 10.1109/TPAMI.2017.2711011.

[3] H. Badino, D. Huber, and T. Kanade. Visual topometric localization. In *Intelligent Vehicles Symposium (IV)*, 2011. doi: 10.1109/IVS.211.5940504.

[4] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996. doi: 10.1007/BF00058655.

[5] Jake Bruce, Jens Wawerla, and Richard Vaughan. The SFU mountain dataset: Semi-structured woodland trails under changing environmental conditions. In *Int. Conf. on Robotics and Automation (ICRA) Workshop on Visual Place Recognition in Changing Environments*, 2015.

[6] Zetao Chen, Adam Jacobson, Niko Sünderhauf, Ben Upcroft, Lingqiao Liu, Chunhua Shen, Ian D. Reid, and Michael Milford. Deep learning features at scale for visual place recognition. In *Int. Conf. on Robotics and Automation (ICRA)*, 2017.

[7] Mark Cummins and Paul Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *The Int. J. of Robotics Research*, 27(6), 2008.

[8] Thomas G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[9] R. J. Duffin and A. C. Schaeffer. A class of nonharmonic fourier series. *Transactions of the American Mathematical Society*, 72(2):341–366, 1952. URL https://www.jstor.org/stable/1990760.

[10] Y. Freund and R. E. Schapire. A Decision Theoretic Generalization of On-Line Learning and an Application to Boosting. In *Second European Conference on Computational Learning Theory (EuroCOLT)*, 1995. URL citeseer.nj.nec.com/freund95decisiontheoretic.html.

[11] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006. doi: 10.1007/s10994-006-6226-1.

[12] A. J. Glover, W. P. Maddern, M. J. Milford, and G. F. Wyeth. FAB-MAP + RatSLAM: Appearance-based

SLAM for multiple times of day. In *Int. Conf. on Robotics and Automation (ICRA)*, 2010. doi: 10.1109/ROBOT.2010.5509547.

[13] 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. doi: 10.1109/TPAMI.2012.193.

[14] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010. doi: 10.1109/MITS.2010.939925.

[15] J. Hawkins, S. Ahmad, S. Purdy, and A. Lavin. Biological and machine intelligence (bami). Initial online release 0.4, 2016. URL https://numenta.com/resources/biological-and-machine-intelligence/.

[16] Jeff Hawkins. *On Intelligence (with S. Blakeslee)*. Times Books, 2004.

[17] Jeff Hawkins and Subutai Ahmad. Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in Neural Circuits*, 10, 2016. doi: 10.3389/fncir.2016.00023.

[18] Unnat Jain, Vinay P. Namboodiri, and Gaurav Pandey. Compact environment-invariant codes for robust visual place recognition. In *Conference on Computer and Robot Vision (CRV)*, 2017. doi: 10.1109/CRV.2017.22.

[19] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, 2007. URL https://proceedings.neurips.cc/paper/2006/file/2d71b2ae158c7c5912cc0bbde2bb9d95-Paper.pdf.

[20] S. Lowry and M. J. Milford. Supervised and unsupervised linear learning techniques for visual place recognition in changing environments. *IEEE Transactions on Robotics*, 32(3):600–613, 2016. doi: 10.1109/TRO.2016.2545711.

[21] S. Lowry, N. Sünderhauf, P. Newman, John J. Leonard, David Cox, Peter Corke, and Michael J. Milford. Visual place recognition: A survey. *Trans. Rob.*, 32(1), 2016. doi: 10.1109/TRO.2015.2496823.

[22] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The oxford robotcar dataset. *The Int. Journal of Robotics Research*, 36(1):3–15, 2017.

[23] Will Maddern, Geoffrey Pascoe, Matthew Gadd, Dan Barnes, Brian Yeomans, and Paul Newman. Real-time Kinematic Ground Truth for the Oxford RobotCar Dataset. *CoRR*, abs/2002.10152, 2020.

[24] Alireza Makhzani and Brendan Frey. k-sparse autoencoders, 2014. URL https://arxiv.org/abs/1312.5663.

[25] Carlo Masone and Barbara Caputo. A survey on deep visual place recognition. *IEEE Access*, 9:19516–19547, 2021. doi: 10.1109/ACCESS.2021.3054937.

[26] M. Milford and G. F. Wyeth. SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights. In *Int. Conf. on Robotics and Automation (ICRA)*, 2012.

[27] Peer Neubert. *Superpixels and their Application for Visual Place Recognition in Changing Environments. PhD: TU Chemnitz* . dissertation, Chemnitz University of Technology, 2015. URL http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-190241.

[28] Peer Neubert and Stefan Schubert. Hyperdimensional computing as a framework for systematic aggregation of image descriptors. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[29] Peer Neubert, Stefan Schubert, and Peter Protzel. A neurologically inspired sequence processing model for mobile robot place recognition. *IEEE Robotics and Automation Letters*, 4(4):3200–3207, 2019.

[30] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72, 2011.

[31] H. Noh, A. Araujo, J. Sim, T. Weyand, and B. Han. Large-scale image retrieval with attentive deep local features. In *Int. Conf. on Computer Vision (ICCV)*, 2017. doi: 10.1109/ICCV.2017.374.

[32] R. E. Schapire. The Strength of Weak Learnability. *Machine Learning*, 5:197–227, 1990.

[33] Stefan Schubert and Peer Neubert. What makes visual place recognition easy or hard? *CoRR*, abs/2106.12671, 2021.

[34] Stefan Schubert, Peer Neubert, and Peter Protzel. Unsupervised learning methods for visual place recognition in discretely and continuously changing environments. In *Int. Conf. on Robotics and Automation (ICRA)*, 2020.

[35] Stefan Schubert, Peer Neubert, and Peter Protzel. Fast and memory efficient graph optimization via ICM for visual place recognition. In *Robotics: Science and Systems (RSS)*, 2021. doi: 10.15607/RSS.2021.XVII.091.

[36] Niko Sünderhauf. *Robust Optimization for Simultaneous Localization and Mapping . PhD: TU Chemnitz* . dissertation, Chemnitz University of Technology, 2012. URL https://nbn-resolving.org/urn:nbn:de:bsz:ch1-qucosa-86443.

[37] Niko Sünderhauf, Peer Neubert, and Peter Protzel. Are we there yet? challenging seqslam on a 3000 km journey across all four seasons. *Int. Conf. on Robotics and Automation (ICRA) Workshop on Long-Term Autonomy*, 2013.

[38] A. Torii, R. Arandjelović, J. Sivic, M. Okutomi, and T. Pajdla. 24/7 place recognition by view synthesis. In *Conf. on Computer Vision and Pattern Recognition*, 2015.

[39] John Wright, Allen Y. Yang, Arvind Ganesh, S. Shankar Sastry, and Yi Ma. Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):210–227, 2009. doi: 10.1109/TPAMI.2008.79.