Abstract

In this paper, we propose a new open-source benchmarking framework for Visual Geo-localization (VG) that allows to build, train, and test a wide range of commonly used architectures, with the flexibility to change individual components of a geo-localization pipeline. The purpose of this framework is twofold: i) gaining insights into how different components and design choices in a VG pipeline impact the final results, both in terms of performance (recall@N metric) and system requirements (such as execution time and memory consumption); ii) establish a systematic evaluation protocol for comparing different methods. Using the proposed framework, we perform a large suite of experiments which provide criteria for choosing backbone, aggregation and negative mining depending on the use-case and requirements. We also assess the impact of engineering techniques like pre/post-processing, data augmentation and image resizing, showing that better performance can be obtained through somewhat simple procedures: for example, downsampling the images’ resolution to 80% can lead to similar results with a 36% savings in extraction time and dataset storage requirement. Code and trained models are available at https://deep-vg-bench.herokuapp.com/.

1. Introduction

The task of coarsely estimating the place where a photo was taken based on a set of previously visited locations is called Visual (Image) Geo-localization (VG) [35, 40, 81] or Visual Place Recognition (VPR) [20, 42] and it is addressed using image matching and retrieval methods on a database of images of known locations. We are witnessing a rapid growth of this field of research, as demonstrated by the increasing number of publications [2, 10, 14, 21–23, 27, 29, 34, 35, 40, 42, 44, 55, 58, 69, 70, 73–76, 78, 82], but this expansion is accompanied by two major limitations:

i) A focus on single metric optimization, as it is common practice to compare results solely based on the recall on chosen datasets and ignoring other factors such as execution time, hardware requirements, and scalability. All these aspects are important constraints in the design of a real-world VG system. For instance, one might gladly accept a 5% drop in accuracy if this leads to a 90% decrease of descriptors size as the resulting reduction in memory requirements enables a better scalability. Similarly, computational time and descriptor dimensionality are crucial constraints in real-time applications, given a target hardware platform.

ii) A lack of a standardized framework to train and test VG models. It is common practice to perform direct comparisons among off-the-shelf methods that use different set-ups (e.g., data augmentation, initialization, training dataset, etc.) [35, 64, 80], which can hide the improvement (or lack thereof) obtained by algorithmic changes and it does not allow to pinpoint the impact of each individual component.

Table 1 shows how some simple engineering choices can have big effects on the recall metric.

Although previous benchmarks for VPR [80] and the related task of Visual Localization [56, 62] offer interesting insights, they do not address the aforementioned issues. For these reasons, we propose a new open-source benchmark that provides researchers with an all-inclusive tool to build, train, and test a wide range of commonly used VG architectures, offering the flexibility to change each component of a geo-localization pipeline. This allows to rigorously examine how each element of the system influences the final
results while providing information computed on-the-fly regarding the number of parameters, FLOPs, descriptors dimensionality, etc.

Using our framework, we run numerous experiments aiming to understand which components are the most suitable for a real-world application, and derive good practices depending on the target dataset and one’s hardware availability. For example, we find that ResNet-50 [28] provides a good trade-off between accuracy, FLOPs and model size, and that Visual Transformers can successfully replace the CNN backbones and achieve better geo-localization performances when trained on larger datasets. Furthermore, we observed that partial negative mining and reduced resolution yield important decrease in computations without significantly compromising the performance, or even yielding gains in some cases.

The benchmark’s software and models are hosted at https://deep-vg-bench.herokuapp.com/.

2. Related Work

Representation learning for visual retrieval and localization. Visual Geo-localization (VG), Visual Localization (VL), and Landmark Retrieval (LR) are three well-known Computer Vision tasks that try to establish a mapping between an image and a spatial location, albeit with some nuances. In VG the goal is to find the geographical location of a given query image and the predicted coordinates are considered correct if they are roughly close to ground truth position [2, 10, 23, 35, 39, 40, 74, 75]. VL focuses on precisely estimating the 6 DoF camera pose of a query image within a known scene. VG methods can be used as a part of a VL pipeline, combined with other processing stages that reduce the differences when used in a VL task. Therefore the evaluation papers on VL [56, 62, 71] might not be indicative of VG performance, justifying a separate benchmark on the latter. LR is a particular case of Image Retrieval (IR) in which queries contain some landmark, and the goal is to identify all database instances depicting the same landmark, regardless of their visually overlap with the query photo. Since VG is usually addressed as a retrieval problem where the query position is estimated using the GPS tags of the top retrieved image, several methods originally proposed for LR (or IR in general) have carried over to VG. LR datasets, both on a city-scale (Oxford and Paris Buildings [53, 54]) and on a global scale (Google Landmarks [47, 77]), consists of a discrete set of landmarks, whereas VG datasets usually cover a continuous geographical area.

IR [3, 16, 67] is traditionally performed via nearest neighbors search using fixed-size image representations [15, 30, 31, 33, 52, 63, 69] obtained from the aggregation of highly informative local [1, 8, 41] or global [48, 79] features. Convolutional neural networks (CNNs) have become the de-facto standard to extract the features for IR, using various methods to concatenate them [7, 59] or pool them [4, 5, 68] to create image descriptors. Among the deep learning representation methods, one that has proven very effective for VG is NetVLAD [2], a differentiable implementation of VLAD [33] trained end-to-end with the CNN backbone directly for place recognition. The layer has since been used in numerous works [10, 19, 22, 23, 27, 40, 74, 75]. One downside of NetVLAD is that it outputs high-dimensional descriptors, leading to steep memory requirements for VG systems. This problem has inspired research on more compact descriptors, either using dimensionality reduction techniques [7, 11, 24, 49, 57, 83] or replacing NetVLAD with lighter pooling layers, such as GeM [58] and R-MAC [25].

It has also been shown that attention modules can be used to focus feature extraction and aggregation towards the most salient parts of the scene for the geo-localization task [11, 35, 39, 46]. The Contextual Reweighting Network (CRN) [35] is a variation of NetVLAD that adds a contextual modulation to produce a weighting mask based on semi-global context. Visual Transformers based on self-attention such as ViT [17] and DeiT [72] have also been used in IR [12, 18], but not yet in VG. All these architectures used in VG to learn image representations are trained with metric embedding objectives commonly used in learning-to-rank problems, such as the contrastive loss [49, 57, 58], the triplet loss [2, 25, 35] and the SARE loss [40].

Our benchmark analyzes how the combination of popular backbone networks, pooling strategies, data augmentation, and engineering choices impacts geo-localization performance and other aspects, such as memory and computational requirements.

Benchmarking. The only available benchmark focused specifically on VG/VPR is VPR-Bench [80]. In contrast to our work, [80] (as well as [56] for VL) directly compares off-the-shelf models because it is mainly concerned with the performance of VG in practical settings, where one would likely prefer using a pre-trained model rather than having to fine-tune or train it. On the other hand, we are more interested in measuring the impact of algorithmic changes, which requires performing comparisons where all other factors are the same. To this end, we propose a modular framework that allows a fair evaluation of each element of a VG system under identical conditions, ensuring clarity and reliability of the results.

While [80] also provides insights on descriptors dimensionality and retrieval time, we focus on more general hardware-agnostic statistics, such as FLOPs and model size (Sec. 4.1), training complexity (Sec. 4.4), storage requirements (Sec. 4.6).
3. Methodology

This section describes the VG pipeline used in our benchmark (cf. Fig. 1) and our experimental setup.

3.1. Visual Geo-localization System

The VG task is commonly tackled using an image retrieval pipeline: given a new photo (query) to be geolocalized, its location is estimated by matching it to a database of geo-tagged images. A VG system is thus an algorithm that first extracts descriptors for the database images (offline) and for the query photo (online), then it applies a nearest neighbors search in the descriptor space. The orange blocks in Fig. 1 show that a VG system is built through several design choices, including network architectures, negative mining methods, and engineering aspects such as image sizes and data augmentation. All of these choices impact the behavior of the system, both in terms of performance and required resources. We propose a new benchmark to systematically investigate the impact of the components of VG systems, using the modular architecture shown in Fig. 1 as a canvas to reproduce most VG methods based on CNN backbones and to develop new models based on Visual Transformers.

This abstract model contains several components that can be modified, both during training and test time: the backbone (Sec. 4.1); feature aggregation (Sec. 4.2); mining training examples (Sec. 4.4); image resizing (Sec. 4.6); data augmentation (Sec. 4.5). We conduct a series of tests focused individually on each of these elements, to systematically show each component’s influence. Due to limited space, we only summarize here the results of some experiments, while detailed results and additional experiments on pre/post-processing methods and predictions refinement, effect of pre-training and many other aspects are provided in the supp. material.

The code of the benchmark follows the modular structure shown in Fig. 1, where each component can be modified. We further provide scripts to download and format a number of datasets, and to train and test the models making easy to perform a large number of experiments while ensuring consistency and reproducibility of results. Our codebase allows to easily reproduce the architectures used in a wide range of works [2, 25, 35, 40, 58, 61, 68, 75] and commonly used training protocols [2, 40, 75]. More details on the software are provided in the supp. material.

3.2. Datasets

We use six highly heterogeneous datasets (see Tab. 2 and maps in the supp. material), which together cover a variety of real-world scenarios: different scales, degree of inter-image variability, different camera types. For training, we use Pitts30k [2] and Mapillary Street-Level Sequences (MSLS) [75] datasets, as they provide a small and large amount of images, respectively. While Pitts30k is very homogeneous, i.e. all images share the same resolution, weather conditions and camera, MSLS represents a wide range of conditions from very diverse cities. Regarding MSLS, given the lack of labels for the test set, we follow [27] and report validation recalls computed on the validation set. To assess inter-dataset robustness, we also test all models on four other datasets: Tokyo 24/7 [69], Revisited San Francisco (R-SF) [13,38,71], Eynsham [16] and St Lucia [45]. Further details on these datasets, such as their geographical coverage, are included in the supp. material.

3.3. Benchmark Protocol

In all experiments, unless otherwise specified, we use the metric of recall@N (R@N) measuring the percentage of queries for which one of the top-N retrieved images was taken within a certain distance of the query location. We mostly focus on R@1 and, following common practice in
4. Results

Throughout this section, we explore how each block from Fig. 1 influences the results. Specifically, we first investigate the use of different architectures, with a focus on backbones (Sec. 4.1), aggregation methods (Sec. 4.2) and Transformers-based networks (Sec. 4.3). We then move to train-time components (i.e., negative mining Sec. 4.4 and data augmentation Sec. 4.5), to understanding how the resolution of the images influences a VG system (Sec. 4.6), and finally we explore the use of efficient nearest neighbor search algorithms (Sec. 4.7). Given the limited amount of space in the manuscript, a thorough extension over each one of these sections can be found in the supp. material, as well as further experiments on various metrics and more.

4.1. CNN Backbones

Table 2. Summary of the datasets: "panorama" means images are cropped from a 360° panorama (including undistortion); "front-view" means that only one (forward facing) view is available; "phone" means photos were collected with a smartphone. "panorama" and "front-view" images were taken with car-rooftop cameras. *Variable resolution.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># train/val</th>
<th># test</th>
<th>Dataset size</th>
<th>Database type</th>
<th>Queries type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pitts30k</td>
<td>20K / 15K</td>
<td>10K / 6.8K</td>
<td>2.0 GB</td>
<td>panorama</td>
<td>480×640 panorama</td>
</tr>
<tr>
<td>MSLS</td>
<td>934K / 514K</td>
<td>19K / 11K</td>
<td>56 GB</td>
<td>front-view</td>
<td>480×640 front-view</td>
</tr>
<tr>
<td>R-SF</td>
<td>0 / 0</td>
<td>1.05M / 599</td>
<td>36 GB</td>
<td>panorama</td>
<td>480×640 phone*</td>
</tr>
<tr>
<td>Eyesham</td>
<td>0 / 0</td>
<td>24K / 24K</td>
<td>1.2 GB</td>
<td>panorama</td>
<td>512×384 panorama</td>
</tr>
<tr>
<td>St. Lucia</td>
<td>0 / 0</td>
<td>1.5K / 1.5K</td>
<td>124 MB</td>
<td>front-view</td>
<td>480×640 front-view</td>
</tr>
</tbody>
</table>

Table 3 shows the results of our experiments. For all ResNets, we use the feature maps extracted from the last pooling layer before the classifier part.

Preliminary results have shown on average better recall and efficiency rather than using until conv4 layer1. For VGG16, we use all the convolutional layers, excluding the last pooling before the classifier part. Table 3 shows the results of our experiments.

Discussion. We can see that deeper ResNets, such as ResNet-50 and ResNet-101, achieve better results w.r.t. their shallower counterparts. In particular, ResNet-50 shows recalls on par with ResNet-101, but with the advantage of less than half the FLOPs and model size, making the former a more practically relevant option than the latter. ResNet-18 performs worse, but allows for much faster and lighter computation, making it the most efficient, lightweight backbone. Moreover, results considerably depend on the training data: as an example, training the same network on Pitts30k or MSLS yields a 30% gap testing the model on St. Lucia, as well as a noticeable difference on other datasets too. This effect demonstrates that comparing models trained on different datasets, as done in [80], can be misleading.

4.2. Aggregation and Descriptor Dimensionality

Aggregations methods are layers tasked with processing the output features of the backbone. Over the years, a number of such methods have been proposed, from shallow pooling layers [5, 60] to more complex modules [2, 35]. Our framework allows to compute results with a number of them, namely SPOC [5], MAC [60], R-MAC [68], RRM [37], GeM [58], NetVLAD [2] and CRN [35]. While a complete list of results with all aggregation methods is shown in the supp. material, in Tab. 4 we report the performance of the best performing aggregators: GeM, NetVLAD and CRN. Given the difference in size of the outputted descriptors, we apply PCA or a fully connected (FC) layer to even their dimensionality.

Discussion. The results in Tab. 4 show that performance strongly depends on the training set. When training on the small Pitts30k, the best results are obtained globally with CRN, even when reducing its dimension to be the same as GeM. However, when training on the much larger MSLS, the advantage of CRN is reduced, and both CRN
and NetVLAD end up being significantly outclassed on Tokyo and R-SF\(^2\) by GeM, making it a more compelling choice. Furthermore, the dimensionality reduction via PCA yields a significant drop in performance for NetVLAD and CRN, while adding a fully connected layer on top of GeM gives best results when trained on a large scale dataset, which is the type of scenario for which GeM was proposed \cite{58}. Note that while the CRN aggregator yields the most robust results, it has the drawbacks of requiring a two-stage training process that almost doubles the training time and three times more hyperparameters w.r.t. NetVLAD. In addition, depending on the initialization of its modulation layer, training does not always converge.

### 4.3. Visual Transformers

In this section we investigate how Visual Transformers compare to more traditional CNN-based methods in VG. For this analysis we use two popular Transformer architectures, the Vision Transformer (ViT) \cite{17}, which processes the images by splitting them into sequences of flattened 2D patches, and the Compact Convolutional Transformer (CCT) \cite{26}, which incorporates convolutional layers to insert the inductive bias of CNNs. Following \cite{18}, we use as a global descriptor the CLS token, which is the output state of the preprend learnable embedding to the sequence of patches \cite{17}. Moreover, we test the use of CCT in conjunction with traditional aggregation methods, such as GeM \cite{58} and NetVLAD \cite{2}, and with SeqPool, which was specifically introduced in \cite{26} for Transformers.

\[^2^\text{The reason could be that these two datasets have different query and database image types, i.e. phone-taken and panorama images, respectively.}\]

---

**Table 3.** Results and computational requirements with different convolutional backbones. Extraction time is the average over a 1000 forward passes.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Aggregation Method</th>
<th>Features Dim</th>
<th>FLOPs (GF)</th>
<th>Model Size (MB)</th>
<th>Extraction Time (ms)</th>
<th>Training on Pitts30k R@1</th>
<th>Training on MSLSR@1 R@1</th>
<th>Training on Tokyo 24/7 R@1</th>
<th>Training on St Lucia R@1</th>
<th>Training on Eynsham R@1</th>
<th>Training on St Lucia R@1</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>VG-16</td>
<td>GeM</td>
<td>512</td>
<td>188.01</td>
<td>56.13</td>
<td>12.3</td>
<td>78.5</td>
<td>43.4</td>
<td>39.9</td>
<td>40.4</td>
<td>70.2</td>
<td>66.7</td>
<td>43.6</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>GeM</td>
<td>256</td>
<td>17.29</td>
<td>10.63</td>
<td>4.1</td>
<td>77.5</td>
<td>35.3</td>
<td>35.3</td>
<td>34.2</td>
<td>64.3</td>
<td>46.2</td>
<td>71.6</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>GeM</td>
<td>1024</td>
<td>40.61</td>
<td>32.71</td>
<td>6.7</td>
<td>82.0</td>
<td>38.0</td>
<td>41.5</td>
<td>54.4</td>
<td>66.3</td>
<td>59.0</td>
<td>77.4</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>GeM</td>
<td>1024</td>
<td>86.29</td>
<td>105.36</td>
<td>9.6</td>
<td>82.4</td>
<td>39.6</td>
<td>44.0</td>
<td>52.5</td>
<td>69.0</td>
<td>57.6</td>
<td>77.2</td>
</tr>
<tr>
<td>VGG-16</td>
<td>NetVLAD</td>
<td>32768</td>
<td>188.09</td>
<td>56.38</td>
<td>13.0</td>
<td>83.2</td>
<td>50.9</td>
<td>61.4</td>
<td>64.6</td>
<td>74.4</td>
<td>50.1</td>
<td>79.0</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>NetVLAD</td>
<td>16384</td>
<td>10.76</td>
<td>4.4</td>
<td>86.4</td>
<td>47.4</td>
<td>63.4</td>
<td>61.4</td>
<td>78.6</td>
<td>57.6</td>
<td>81.6</td>
<td>75.6</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>NetVLAD</td>
<td>65536</td>
<td>40.51</td>
<td>33.21</td>
<td>8.5</td>
<td>86.0</td>
<td>50.7</td>
<td>69.8</td>
<td>67.1</td>
<td>77.7</td>
<td>60.2</td>
<td>80.9</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>NetVLAD</td>
<td>65536</td>
<td>86.06</td>
<td>105.86</td>
<td>11.5</td>
<td>86.5</td>
<td>51.8</td>
<td>72.2</td>
<td>67.5</td>
<td>74.0</td>
<td>63.6</td>
<td>80.8</td>
</tr>
</tbody>
</table>

**Table 4.** Aggregation methods: we report results with different aggregation methods downscaled or upsampled to equivalent dimensionality.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Aggregation Method</th>
<th>Features Dim</th>
<th>FLOPs (GF)</th>
<th>Model Size (MB)</th>
<th>Extraction Time (ms)</th>
<th>Training on Pitts30k R@1</th>
<th>Training on MSLSR@1 R@1</th>
<th>Training on Tokyo 24/7 R@1</th>
<th>Training on St Lucia R@1</th>
<th>Training on Eynsham R@1</th>
<th>Training on St Lucia R@1</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50</td>
<td>GeM</td>
<td>1024</td>
<td>82.0</td>
<td>38.0</td>
<td>41.5</td>
<td>45.4</td>
<td>66.3</td>
<td>59.0</td>
<td>77.4</td>
<td>72.0</td>
<td>55.4</td>
<td>45.7</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>NetVLAD + PCA 1024</td>
<td>1024</td>
<td>83.9</td>
<td>46.5</td>
<td>59.4</td>
<td>53.2</td>
<td>72.5</td>
<td>57.7</td>
<td>77.4</td>
<td>74.8</td>
<td>51.3</td>
<td>39.0</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>CRN + PCA 1024</td>
<td>1024</td>
<td>84.1</td>
<td>49.9</td>
<td>64.6</td>
<td>58.8</td>
<td>74.3</td>
<td>63.4</td>
<td>77.3</td>
<td>75.6</td>
<td>51.8</td>
<td>38.8</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>GeM + FC 2048</td>
<td>2048</td>
<td>80.1</td>
<td>33.7</td>
<td>43.6</td>
<td>48.2</td>
<td>70.0</td>
<td>56.0</td>
<td>79.2</td>
<td>73.5</td>
<td>64.0</td>
<td>55.1</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>NetVLAD + PCA 2048</td>
<td>2048</td>
<td>84.4</td>
<td>47.9</td>
<td>62.6</td>
<td>56.0</td>
<td>74.1</td>
<td>58.9</td>
<td>78.5</td>
<td>75.4</td>
<td>52.8</td>
<td>42.6</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>CRN + PCA 2048</td>
<td>2048</td>
<td>84.7</td>
<td>51.2</td>
<td>67.1</td>
<td>62.3</td>
<td>75.8</td>
<td>65.0</td>
<td>78.3</td>
<td>76.3</td>
<td>54.3</td>
<td>42.8</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>GeM + FC 65536</td>
<td>65536</td>
<td>80.8</td>
<td>35.8</td>
<td>45.6</td>
<td>49.0</td>
<td>72.5</td>
<td>59.6</td>
<td>79.0</td>
<td>74.4</td>
<td>69.2</td>
<td>58.4</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>NetVLAD</td>
<td>65536</td>
<td>86.0</td>
<td>50.7</td>
<td>69.8</td>
<td>67.1</td>
<td>77.7</td>
<td>60.2</td>
<td>80.9</td>
<td>76.9</td>
<td>62.8</td>
<td>51.5</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>CRN</td>
<td>65536</td>
<td>85.8</td>
<td>54.0</td>
<td>73.1</td>
<td>70.9</td>
<td>79.7</td>
<td>65.9</td>
<td>80.8</td>
<td>77.8</td>
<td>63.6</td>
<td>53.4</td>
</tr>
</tbody>
</table>

**Table 5.** Transformers Comparison of traditional CNN architectures with novel Transformers-based approaches.

**Discussion.** Table 5 compares traditional CNN-based methods with novel Visual Transformer based approaches, never used before specifically for VG. The main findings of this set of experiments is that they represent a viable alternative to CNN-based backbones even without an additional aggregation step using directly the compact and robust representation provided by the CLS token. Further improvements can be obtained when combined with aggregators such as GeM, SeqPool, NetVLAD, as shown in the table. Overall, the results show that these architectures possess better generalization capabilities than their CNN counterparts, and ViT proves to be competitive even with the much bigger NetVLAD descriptors, albeit with higher computational requirements. As for CCT, despite being incredibly lightweight, with a cost comparable to a ResNet-18, consistently outperforms the ResNet-18 and, in many cases, also the ResNet-50, which has roughly double the computational cost. Concluding, it seems that the SeqPool aggregator enhances the robustness of the CCT descriptors, providing better generalization and that NetVLAD coupled with CCT outperforms CNN-based methods. We observe similar behaviors when trained on Pitts30k (see supp. material).
Figure 2. **Data Augmentation.** Results obtained applying popular augmentation techniques during training. We used PyTorch’s transforms, and the x axis relates to the parameter passed to the class; the higher the parameter, the heavier the transform effect (i.e., x = 0 equals to the identity transformation). Refer to supp. material for further details on the transforms.

The main limitations of these architectures is the lack of an all-around best configuration. In other words, for each use case, an additional tuning on where to truncate/freeze the network was required, unlike the CNNs which were consistently used up to their \textit{com4} layer.

### 4.4. Negative Mining

An important step in a VG pipeline is the mining of negatives: ideally, we want to select images of different scenes that appear visually similar to the query to ensure that the model learns highly informative features for the task. We extensively compare three main mining strategies: full database mining \cite{2, 75}, partial database mining \cite{75} where only a reasonable subset of images is ranked, and random negative sampling. Details about the mining strategies, full set of results and their analyses can be found in the supp. material, here we present in Tab. 6 only a subset of our results as illustration and summarize our main findings.

**Discussion.** As expected, both full and partial database mining outperform the random negative sampling. The latter, in spite of its low cost yields in average 5% lower results on Pitts30k, due to the low variability of the dataset. Indeed, on the larger MSLS results drop of 10% or more. On the other hand, full database mining does not provide always best performance and on average its gain over partial mining is around 1%. Furthermore, on large scale datasets such as MSLS full mining is not feasible in a reasonable time. These results clearly show that partial mining is, in general, a great compromise between cost and accuracy.

### 4.5. Data Augmentation

Here we investigate if and which data augmentation are beneficial for VG methods, and if the improvements are domain-specific or can generalize to diverse datasets. We apply data augmentation to the query, with the sole exception of random horizontal flipping, for which we either flip or not flip the whole triplet. We run experiments with many popular augmentation techniques, training a ResNet-18 with NetVLAD on Pitts30k.

**Discussion.** Plots of the results are in Fig. 2 (shown in higher resolution in the supp. material). Depending on the test dataset, we observe different impact of these augmentations. On one hand, on Pitts30k augmentation only worsens results, probably due to dataset homogeneity between train and test. On the other hand, we see that some techniques can improve robustness on unseen datasets, in particular color jittering methods that change brightness, contrast and saturation. As an example, setting contrast\textsuperscript{3} up to 2 can improve recall@1 by more than 3% on MSLS, 5% on Tokyo 24/7, 5% on St Lucia, with a less than 1% drop on Pitts30k and Eynsham. Although most augmentations fail to produce consistent improvements, two notable exceptions are random horizontal flipping (with probability 50%) and random resized cropping, where crops are as small as 50% of the image size (and then resized to full resolution).

### 4.6. Resize

While common VG datasets have images of resolutions around 480x640 pixels, it is interesting to investigate how resizing them can affect the results. To this end, we perform experiments by training and testing models on images of lower resolution, by reducing both sides of the images from 80% to 20% of their original size, both at train and test time on Pitts30k. We conduct this analysis with CNNs

\textsuperscript{3}This refers to PyTorch’s \texttt{ColorJittering()} function.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Aggregation Method</th>
<th>Mining Method</th>
<th>Space &amp; Time Complexity</th>
<th>Training on Pitts30k</th>
<th>Pitts30k</th>
<th>MSLS</th>
<th>Tokyo 24/7</th>
<th>R@1</th>
<th>R@1</th>
<th>R@1</th>
<th>R@1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>GeM</td>
<td>Random</td>
<td>O(1)</td>
<td>73.7</td>
<td>30.5</td>
<td>31.3</td>
<td>24.0</td>
<td>58.2</td>
<td>41.0</td>
<td>62.2</td>
<td>50.6</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>GeM</td>
<td>Full database</td>
<td>O(kdb + kq)</td>
<td>77.8</td>
<td>35.3</td>
<td>35.3</td>
<td>34.2</td>
<td>64.3</td>
<td>46.2</td>
<td>70.1</td>
<td>61.8</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>GeM</td>
<td>Partial database</td>
<td>O(kq + #pos)</td>
<td>76.5</td>
<td>34.2</td>
<td>33.9</td>
<td>32.9</td>
<td>64.0</td>
<td>45.6</td>
<td>71.6</td>
<td>65.3</td>
</tr>
</tbody>
</table>

**Table 6. Negative mining methods.** "Space & Time Complexity" refers to the complexity of building the cache, which normally is done after iterating over 1000 triplets \cite{2, 75}. \#db and \#q are the numbers of database and query images, kdb and kq are chosen constants (usually set to 1000), and \#pos is the number of positives for the considered queries, which depends on the queries and database density.
followed by GeM or NetVLAD, since such architectures do not require a fixed input image resolution.

**Discussion.** Interestingly, it can be seen in Figure 3 that using the highest available resolution is in most cases superfluous, and often even detrimental. On average, NetVLAD’s descriptors seem to better handle higher resolutions than their GeM counterparts. Lower resolutions, as low as 40%, show improved results especially when there is a wide domain gap between train and test sets: this is exemplified by the results on the St Lucia dataset, which is very different from Pitts30k (the former has only forward views) and shows best R@1 performance when using 40% of the original resolution. This behaviour can be explained by the disappearance of domain specific low-level patterns (e.g., texture and foliage) when the size of the image is reduced. In general 60% is a good compromise, suggesting that for geo-localization, which is strongly related to appearance-based retrieval, fine details are not too important.

Finally, note that 40% resolution means reducing it to 192x256, with FLOPs going down to (40%)² = 16% w.r.t. full resolution images. Storage needs also decrease in the same fashion as FLOPs, and although images are not directly needed in a retrieval system (only descriptors and coordinates are used for kNN), they can be used useful for post-processing, e.g., spatial verification, or to generate a visual response for users.

4.7. Nearest Neighbor Search and Inference Time

In practical applications, one of the most relevant factors for a VG system is inference time ($t_i$). Once the application is deployed and has to serve the user’s needs, the perceived delay depends only on $t_i$. Inference time can be divided into: i) **extraction time** ($t_e$), defined as the elapsed time to extract the features of an image, which solely depends on model and resolution; ii) **matching time** ($t_m$), i.e., duration of the kNN to find the best matches in the database, which depends on the parameter $k$ (i.e., number of candidates), the size of the database, the dimension of the descriptors, and the type of searching algorithm.

In Fig. 4a, we report a plot on how matching time linearly depends on the sizes of the database and descriptors. Figure 4b shows how the use of efficient nearest neighbor search algorithm impacts computation and memory footprint. Besides exhaustive kNN, we investigate the use of inverted file indexes (IVF) [66], product quantization without and with IVF (PQ and IVFPQ) [32], inverted multi index (MultiIndex) [6] and hierarchical navigable small world graphs (HNSW) [43]. In Fig. 4b we report results computed with a ResNet-50 + GeM descriptors on R-SF. See more experiments and thorough discussions in the supp. material.

**Discussion** Figure 4a shows that as the database grows, inference time is dominated by matching time whereas the extraction time is generally fixed at around 10 milliseconds (see Tab. 3 and supp. material). On the other hand, Fig. 4b shows that the choice of neighbor search algorithm can bring huge benefits on time and memory footprint, with little to no loss in recalls. Among the most interesting results, IVFPQ reduces both matching time and memory footprint by 98.5%, with a drop in accuracy from 45.4% to 41.4%. Note that memory footprint is an important factor in image retrieval, since for fast computation all vectors should be kept in RAM, making large scale VG application expensive in terms of memory. For example, R-SF dataset’s descriptors, with a ResNet-50 + NetVLAD, require roughly $1.05M \times 65536 \times 4B = 256$GB of memory, thus making a RAM-efficient search technique (e.g., product quantization) very useful. When memory is not a critical constraint, us-

![Figure 3. Changing the images’ resolution.](image_url)

![Figure 4. (a) Matching time for one query.](image_url)
Table 7. Comparison between recent SOTA methods, and a simple ResNet-18+NetVLAD where we use all the insight gained from the benchmark to find its optimal configuration: training with data augmentation, resize 80%, and majority voting post-processing for Tokyo 24/7 (since queries have different resolutions).

<table>
<thead>
<tr>
<th>Method</th>
<th>Feat. Dim</th>
<th>R@1</th>
<th>R@1</th>
<th>R@1</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16 + NetVLAD + PCA [51]</td>
<td>4096</td>
<td>82.4</td>
<td>81.5</td>
<td>68.9</td>
</tr>
<tr>
<td>VGG16 + NetVLAD</td>
<td>32768</td>
<td>-</td>
<td>81.1</td>
<td>60.0</td>
</tr>
<tr>
<td>SRLNet (ICRA21) [50]</td>
<td>4096</td>
<td>-</td>
<td>78.2</td>
<td>72.1</td>
</tr>
<tr>
<td>SRLNet (ICRA21) [50]</td>
<td>32768</td>
<td>85.1</td>
<td>85.8</td>
<td>68.6</td>
</tr>
<tr>
<td>APPSVR (ICCV21) [51]</td>
<td>4096</td>
<td>87.4</td>
<td>88.8</td>
<td>77.1</td>
</tr>
<tr>
<td>APPSVR (ICCV21) [51]</td>
<td>32768</td>
<td>-</td>
<td>86.6</td>
<td>68.3</td>
</tr>
<tr>
<td>ResNet-18 + NetVLAD + PCA (Ours)</td>
<td>4096</td>
<td>86.8</td>
<td>87.9</td>
<td>72.2</td>
</tr>
<tr>
<td>ResNet-18 + NetVLAD (Ours)</td>
<td>16384</td>
<td>87.2</td>
<td>88.1</td>
<td>73.7</td>
</tr>
</tbody>
</table>

Inference time and kNN search. We propose an extensive study for VG, unique in its kind, comparing advanced kNN search algorithms and compact representations. This study has shown that the choice of a good neighbor search algorithm can have a huge impact on time and memory footprint, with little impact on the performance. Furthermore, we observe that advanced kNN methods might nullify the gap in terms of both memory footprint and matching time between larger and smaller descriptors.

5. Discussions and Findings

This work introduces a modular framework that allows to build, train and test a wide range of VG architectures, with the flexibility to change each component of a geo-localization pipeline. Our experiments provide valuable insights on how different engineering choices implemented at training and test time can affect both the performance and the required resources (FLOPs, storage, time).

Architecture. We found that ResNet-50 is an excellent choice as a CNN backbone, yielding close to the best results at a reasonable cost. We also demonstrate for the first time the use of Visual Transformers for VG and find that they provide compelling results compared to their CNN counterparts. Among them, CCT is particularly interesting because it is incredibly lightweight, with a cost comparable to a ResNet-18, but it performs better than a heavier ResNet-50. Regarding the feature aggregation layers, the best performance is generally obtained with CRN, nevertheless requiring a significant training cost. At the same time, the GeM pooling, which is much more efficient, has shown a better generalization power, especially when training the model on a large and heterogeneous dataset. The best results overall are obtained with CCT combined with NetVLAD.

Negative mining. In general for metric learning for retrieval, negative mining is a crucial element. This was confirmed by our experiments, where we have additionally shown that partial mining can yield similar or sometimes even better performance than full mining, but at a fraction of the (computational) cost.

Training dataset. Unsurprisingly, using a large-scale training set, with a wide range of conditions and collected from very diverse cities, leads to significantly better results.

This confirms the importance of the training set and the evidence that comparisons amongst models trained on different datasets, as commonly done in many papers [35, 80], are not fair and should be avoided if possible.

5403
References

[27] Stephen Hausler, Sourav Garg, Ming Xu, Michael Milford, and Tobias Fischer. Patch-netvlad: Multi-scale fusion of


[33] Hervé Jégou, Matthijs Douze, Jorge Sánchez, Patrick Perez, and Cordelia Schmid. Aggregating local image descriptors into compact codes. IEEE transactions on pattern analysis and machine intelligence, 34, 12 2011. 2


[38] Yunpeng Li, Noah Snavely, Daniel Huttenlocher, and Pascal Fua. Worldwide Pose Estimation using 3D Point Clouds. In European Conference on Computer Vision, 2012. 3


[40] Liu Liu, Hongdong Li, and Yuchao Dai. Stochastic Attraction-Repulsion Embedding for Large Scale Image Localization. In IEEE International Conference on Computer Vision, 2019. 1, 2, 3, 4, 8


[47] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. Large-scale image retrieval with attentive deep local features. In IEEE International Conference on Computer Vision, 2017. 2


[52] Florent Perronnin, Yan Liu, Jorge Sánchez, and Herve Poirier. Large-scale image retrieval with compressed fisher vectors. In IEEE Conference on Computer Vision and Pattern Recognition, pages 3384–3391, 06 2010. 2


[54] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In IEEE Conference on Computer Vision and Pattern Recognition, June 2008. 2

[55] Nathan Piasco, Désiré Sidibé, Cédric Demonceaux, and Valérie Gouet-Brunet. A survey on visual-based localization:


