Image Retrieval for Visual Geolocalization: Extensions and Experiments

Jiahao Zhang
DAUIN
Politecnico di Torino
Torino, Italy
s301185@studenti.polito.it

Tianming Qu
DAUIN
Politecnico di Torino
Torino, Italy
s302398@studenti.polito.it

Alessandro Masala
DAUIN
Politecnico di Torino
Torino, Italy
s305206@studenti.polito.it

***Abstract*—In this paper, we propose some additional features to a framework made 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 paper is to gain insights into some specific usage of particular features that may help in improving or at least understanding better how the base frameworks works and design choices in a VG pipeline to impact the final results, and so establishing a systematic evaluation protocol for comparing different methods. Using the given framework, we perform a large suite of experiments which provide criteria for choosing specific parameters depending on the use-case and requirements. In this paper we mainly assess the impact of engineering techniques like pre/post-processing techniques showing how they works and how they may have impact on the trained models**

I. 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) or Visual Place Recognition (VPR) and it is addressed using image matching and retrieval methods on a database of images of known locations.

Thanks to the benchmark [1] we are allowed to experiment some of the most common descriptor aggregators like NetVLAD [2] and GeM [3] but the scope of this paper is to focus on getting into more details regarding some specific aspects of pre/post processing methods and the use of more advanced optimizer/scheduler. In particular the tests will focus on:

- testing the newest optimizer (AdamW [10] and ASGD [4]) and combine their performance with the newest schedulers to obtain the optimal learning rate during training [9], [11]
- how does multi-scale testing consists of and how does it helps to improve the performance of the model with respect to specific data set
- the effects of simple data augmentation to improve day-night shift on training and testing
- Domain Adaptation on the day-night domain

The software produced is available in the following link: Image Retrieval for Visual Geolocalization.

II. Related Work

Visual Geolocalization (VG) has been a well-known Computer Vision tasks that try to establish a mapping between an image and a spatial location. Its 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. VG is usually addressed as a retrieval problem where the query position is estimated using the GPS tags of the top retrieved image. Among the deep learning representation methods, one that has proven very effective for VG is NetVLAD [2], a differentiable implementation of VLAD [5] trained end-to-end with the CNN backbone directly for place recognition. After its discovery, it has become the starting point of analysis of most further works. One of the biggest issues 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 or replacing NetVLAD with lighter pooling layers, such as GeM [3] and others. 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 triplet loss [2], SARE loss [6] or others.

III. Methodology

In this section, we will briefly describe the base VG pipeline setup for our experiments and a general descriptions on the used Datasets both for training and testing.

A. 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) and then it applies a nearest neighbors search in the descriptor space. In the 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. The benchmark used to

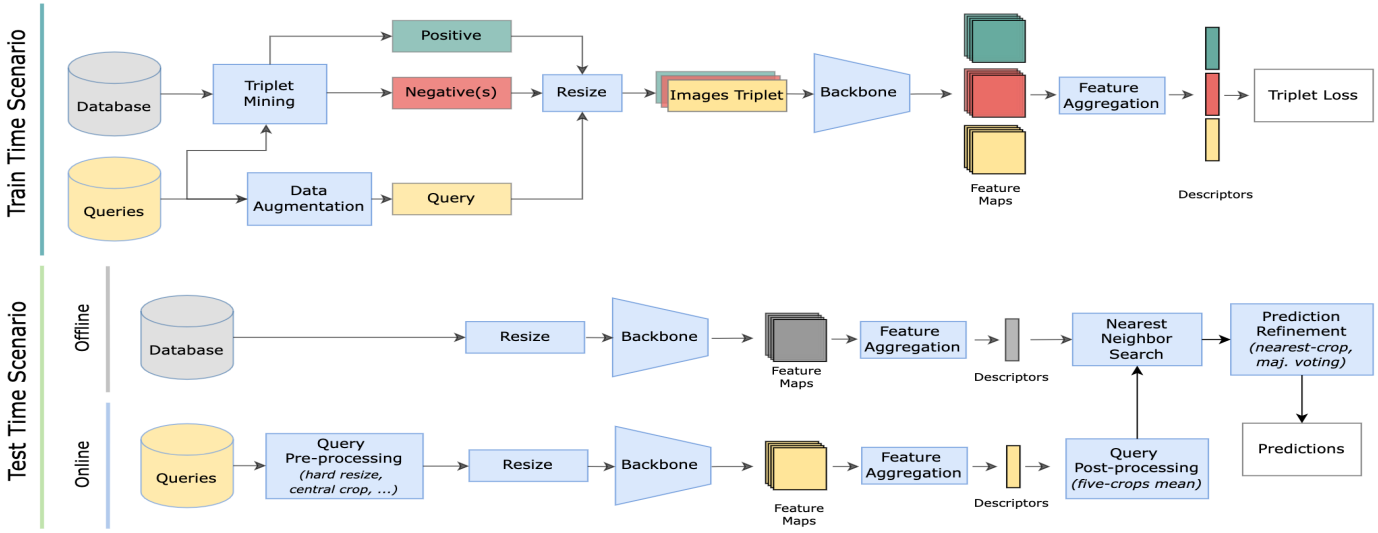

Fig. 1. Diagram of a Visual Geo-localization system

implement the VG system uses the modular architecture shown in Fig. 1 as a canvas to reproduce most VG methods based on CNN backbones. As additional features we added some important components that give more possibilities for test and train while ensuring consistency and reproducibility of results.

B. Dataset

For the purpose of the paper, four highly heterogeneous datasets which together cover a variety of real-world scenarios have been used. While Pitts30k has been used for train-val-test for all the experiments, San Francisco-small, Tokyo-small and Tokyo-night have been used only to test the trained models for evaluation of the added features. Here we will go into details for each dataset :

- **Pittsburgh** (Pitts30k) : it is a subset of Pitts250k and contains 10k database images in each of the train/validation/test sets, which are geographically disjoint.
- **San Francisco-small** (sf-xs) : it is composed of a large database collected by a car-mounted camera and orders of magnitude less queries taken by phone.
- **Tokyo-small** (Tokyo-xs) ; presents a relatively large database (from Google Street View) against a smaller number of queries, which are split into three equally sized sets: day, sunset and night.
- **Tokyo-night** : it is composed of the same dataset as Tokyo small but contains only the night domain images as query.

Details about the train-val-test distribution in local map are reported in Fig 2.

C. Training details

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 R@5 and, following common practice in recent literature [1], [2], we use 25 meters as a distance threshold. Due to limitation on the computation calculation, all results are averaged over two repetitions of experiment. To compare our experimental result, we trained 2 baselines with the basic setting of parameters which are :

- Backbone : ResNet18conv4
- Pretrained model : ImageNet365
- Descriptor aggregator : NetVLAD / GeM
- Loss = triplet loss
- Optimizer = Adam
- Learning rate = 1e-05

Table I shows the results of baseline.

TABLE I
RESULTS OF BASELINE

	NetVLAD		GeM	
Datasets	Recall@1	Recall@5	Recall@1	Recall@5
Pitts30k	86.5	93.1	77.7	89.9
sf-xs	35.7	50.6	16.0	26.7
Tokyo-xs	65.7	75.2	34.9	51.1
Tokyo-night	36.2	50.5	9.50	25.7

All experiments that follows are performed using NetVLAD as descriptor aggregator and as backbone ResNet18conv4 pretrained on ImageNet. For simplicity, the result on the NetVLAD baseline will be reported in all the tables for an easy comparison between the values.

Training is performed until recall@5 on the validation set does not improve for 3 epochs. Given the variability in datasets size, we define an epoch as a pass over 5,000 queries. We use the Adam optimizer [8] for training, as in general it leads

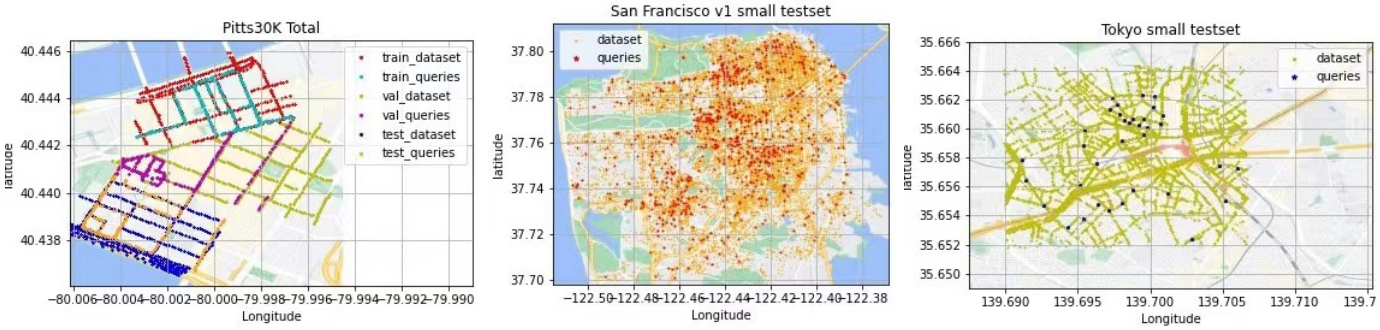

Fig. 2. Diagram of a visual geo-localization system

to faster convergence and better performance. Following the widely used training protocol defined in [1], we use a batch size of 4 triplets, where each triplet is composed of an anchor (the query), a positive and 10 negatives. Following standard practice [2] at train time, the positive is selected as the nearest database image in features space among those within a 10 meters radius from the query and negative images selected from those further than 25.

IV. Experiments and Results

In this section we show the characteristics of new additional features

A. *Optimizer and Scheduler*

In this section, we will apply different optimization algorithms when we train the model, and find the differences in the performance. Furthermore, we also added two different schedulers during training to achieve dynamic learning rates.

The optimization problem is one of the most important research directions in computational mathematics. In the field of deep learning, the choice of the optimization algorithm is also the top priority of a model. Even when the dataset and model architecture is exactly the same, different optimization algorithms are likely to result in very different training results.

Gradient descent Is one of the most widely used optimization algorithms in neural networks. Gradient descent means that, given the model parameters to be optimized $\theta \in \mathbb{R}^d$ and the objective function $J(\theta)$, the algorithm minimizes $J(\theta)$ by back propagation updating θ of the gradient $\nabla_\theta J(\theta)$. The learning rate η determines the update step size at each moment.

As for the baseline, [1] selects ***Adam*** as the default optimizer algorithm with a learning rate 1×10^{-5} and weight decay in 1×10^{-3},and the parameter update formula is:

$$\theta_t = \theta_{t-1} - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon}$$

Which is the most widely used optimization algorithm. We will introduce other 3 different optimizers for our experiment and fine-tune them to obtain the best learning rate and weight decay.

- **SGD With Momentum**
  Stochastic Gradient Descent (SGD), also known as Incremental Gradient Descent, is an iterative method for optimizing a differentiable objective function. This method iteratively updates the weights and bias terms by computing the gradient of the loss function over mini-batches of data. The SGD with momentum has higher stability and convergence speed than the original SGD, and the parameter update formula is:

$$\nu d\omega = \beta \cdot \nu d\omega + (1 - \beta) \cdot \frac{dJ(\theta)}{d\theta}$$

$$\theta_t = \theta_{t-1} - \alpha \cdot \nu d\omega$$

- **ASGD**
  Averaged Stochastic Gradient Descent (ASGD) [4]. Which applies the same parameter update formula as SGD, but averages the weights that are calculated in every iteration.
- **AdamW**
  AdamW [10] is very similar to Adam. It only differs in the way how the weight decay is implemented. The way how it is implemented in Adam came from the good old vanilla SGD optimizer which is not mathematically correct. AdamW fixes this implementation mistake.

Furthermore, we also implemented a scheduler to add a dynamic learning rate during the training process. We use two schedulers:

- **ReduceLROnPlateau**
  ReduceLROnPlateau [9] is a scheduling technique that monitors a quantity and decays the learning rate when the quantity stops improving.
  The improvement of the quantity is based on whether it increases or decreases by a certain minimum amount. This minimum amount is the threshold. We try various thresholds and obtain acceptable performance.
- **CosineAnnealingLR**
  CosineAnnealingLR [11] is a scheduling technique that starts off with a very large learning rate and then aggressively decreases it to a value near 0, before again increasing the learning rate.

TABLE II
TRAINING WITH DIFFERENT OPTIMIZER AND SCHEDULER

Optimizer	Learning Rate	Weight Decay	Scheduler	Pitts30k		sf-xs		Tokyo-xs		Tokyo-night	
				Recall@1/	Recall@5	Recall@1/	Recall5	Recall@1/	Recall@5	Recall @1/	Recall@5
Adam(**Baseline**)	$1e-5$	$1e-3$	None	86.5	93.1	35.7	50.6	65.7	75.2	36.2	50.5
SGD	$1e-5$	$1e-3$	None	78.8	89.8	13.3	24.2	47.6	63.5	19.0	28.6
	$1e-3$	$1e-2$	None	85.8	92.8	34.4	49.6	61.0	73.0	34.3	50.5
	$1e-3$	$1e-3$	None	86.2	93.1	37.0	52.2	64.1	76.2	35.2	49.5
	$1e-3$	$1e-4$	None	86.8	93.1	37.5	52.7	65.7	75.9	36.2	49.5
	$1e-3$	$1e-5$	None	**86.6**	**93.1**	**39.0**	**54.1**	**66.0**	**77.0**	**37.1**	**54.3**
	$1e-2$	$1e-3$	ReduceLROnPlateau	**86.4**	**93.0**	**38.2**	**52.5**	**63.2**	**76.5**	**37.1**	**53.3**
	$1e-3$	$1e-3$	CosineAnnealingLR	84.3	92.2	25.1	39.3	57.1	70.8	25.7	40.0
ASGD	$1e-5$	$1e-3$	None	71.3	85.3	11.1	20.0	42.9	58.1	15.2	23.8
	$1e-5$	$1e-4$	None	72.9	86.2	10.3	19.0	46.3	60.6	17.1	25.7
	$1e-3$	$1e-2$	None	85.6	93.0	27.9	41.1	60.0	74.6	27.6	46.7
	$1e-3$	$1e-3$	None	85.6	92.9	30.3	43.4	61.3	76.8	34.3	52.4
	$1e-3$	$1e-4$	None	84.6	92.6	23.7	36.2	56.8	74.0	24.8	47.6
	$1e-3$	$1e-5$	None	84.4	92.5	20.3	33.7	54.6	71.7	21.0	43.8
	$1e-3$	$1e-3$	ReduceLROnPlateau	83.2	91.9	19.9	32.3	55.6	70.2	24.8	39.0
	$1e-3$	$1e-3$	CosineAnnealingLR	79.7	90.3	14.4	25.7	48.6	64.8	17.1	33.3
AdamW	$1e-5$	$1e-5$	None	86.2	93.0	34.4	47.5	59.4	74.3	24.8	44.8
	$1e-5$	$1e-4$	None	**86.7**	93.0	**36.9**	49.8	62.9	75.6	28.6	47.6
	$1e-5$	$1e-3$	None	85.8	92.9	32.6	47.9	62.9	73.3	32.4	44.8
	$1e-5$	$1e-2$	None	85.3	92.6	28.0	39.9	57.9	74.6	28.6	46.7
	$1e-3$	$1e-3$	None	80.0	90.4	20.3	33.1	36.5	54.6	3.80	21.0
	$1e-3$	$1e-3$	ReduceLROnPlateau	74.6	87.6	10.5	20.6	28.6	43.2	5.70	15.2
	$1e-3$	$1e-3$	CosineAnnealingLR	73.3	85.6	16.7	28.9	24.4	38.1	6.70	15.2

This variation of the learning rate happens according to the cosine annealing schedule. Mathematically, the learning rate is given as

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + \cos(\frac{T_{cur}}{T_{max}}\pi))$$

Where η_t is the current learning rate, η_{max} and η_{min} are the maximum and the minimum learning rates respectively and T_{cur} is the current number of accumulated epochs.

During the experiment of training, we fine-tuned learning rate and weight decay to find the best combinations. Because of Adam's adaptive learning rate that can show excellent performance even at low learning rates, the model have poor performance when using schedulers on Adam. Without loss of generality, the same concept can be applied on AdamW. An important aspect that has been found in SGD is that, when applying the pair $(1 \times 10^{-5}, 1 \times 10^{-3})$ (best parameter for Adam), which needs 26 epochs to reach the convergence, the performance is worse than baseline but when the SGD is trained on different parameters (in this particular case on the pair $(1\times10^{-3}, 1\times10^{-5})$, it gives better results. This somehow marks out, without loss of generality, that each optimizers has its own parameters for the best performance.

Applying the scheduler to the optimizer, through the adjusted parameters, we can implement **dynamic learning rate** during training in order to reduce the number of epochs needed for training without losing too much performance on the model. Fig. 3 shows the dynamic learning rate during training for the three models.

Furthermore, when applying ReduceLROnPlateau scheduler to the SGD optimizer, we got slightly better performance than the baseline, while other optimizers did not perform as expected after applying the scheduler. Table II shows the result of our research.

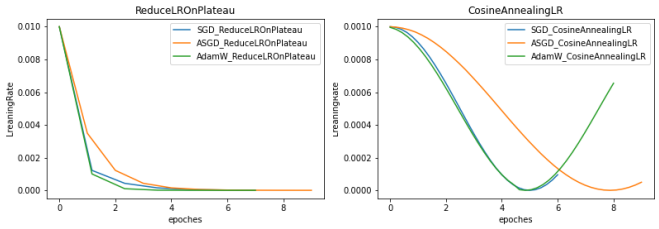

Fig. 3. Dynamic learning rate of scheduler

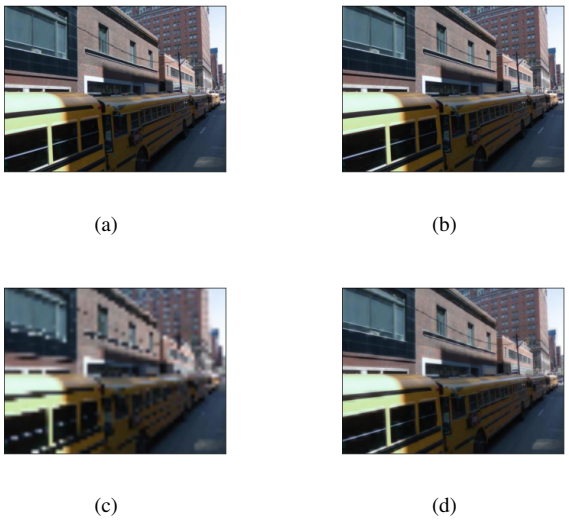

Fig. 4. Example of image resizing and averaging. (a) Original resolution image; (b) higher resolution image; (c) lower resolution image; (d) *avg* of the images (a-c).

B. *Multi-scale testing*

This part takes inspiration from the Deep Visual Geo-localization benchmark paper [1], which shows how the re-

size of the images can affect the results. Particularly, [1] demonstrate that scaling down the images of 60% of their original resolution can lead to comparable (and sometimes better) results in testing.
Therefore, we decided to implement a multi-scale method that allow us to perform experiments by testing models on images of varied resolutions. An example of an image with different resolutions is shown in Figure 4

The images are then passed through the model and the obtained descriptors are aggregated thanks to one of four methods that we propose: *avg, sum, max, min*. In the following list we briefly describe the results obtained for each dataset:

- **Pitts30k**: in this case the computed recalls are slightly worse than the baseline's one with all the proposed methods. However, the methods *avg* and *sum* perform narrowly better than the others.
- **sf-xs**: recalls are consistently better than the baseline's ones. Particularly, even though all the methods leads to increased recalls, the best one turns out to be the *avg* one.
- **Tokyo-xs**: the recalls are negligibly worse than the baseline's ones for R@1, but some better values can be seen in the other recalls. As well as in the previous case, the best recalls are obtained with the *avg* method.
- **Tokyo-night**: in this case the recalls are marginally worse than the baseline's one for R@1, but there are some improvement for the other recalls. The best performances can be obtained using the *avg* method.

By fine-tuning the resolutions, we find out that a balanced approach between lower and higher resolutions is the best to obtain better performances. TABLE III shows the results for each dataset with the different methods[1]:

TABLE III
MODEL TESTS WITH MULTI-SCALE

dataset	method	resolutions	R@1	R@5	R@10	R@20
Pitts30k	baseline	-	86.5	93.1	95.3	96.8
	avg	[0.526, 0.588, 1, 1.7, 1.9]	86.4	93.0	95.2	96.8
	sum	[0.526, 0.588, 1, 1.7, 1.9]	86.4	93.0	95.2	96.8
	max	[0.526, 0.588, 1, 1.7, 1.9]	86.2	93.0	95.1	96.6
	min	[0.526, 0.588, 1, 1.7, 1.9]	86.1	93.0	95.0	96.6
sf-xs	baseline	-	37.5	50.6	56.2	62.8
	avg	[0.526, 0.588, 1, 1.7, 1.9]	**43.7**	**57.3**	**63.2**	**68.4**
	sum	[0.526, 0.588, 1, 1.7, 1.9]	**43.5**	**56.5**	**62.6**	**67.7**
	max	[0.526, 0.588, 1, 1.7, 1.9]	**42.6**	**55.6**	**61.6**	**66.5**
	min	[0.526, 0.588, 1, 1.7, 1.9]	**41.9**	**55.9**	**61.2**	**65.8**
Tokyo-xs	baseline	-	65.7	75.2	79.4	83.5
	avg	[0.526, 1, 1.7]	60.6	**75.6**	**80.3**	**84.4**
	sum	[0.526, 0.588, 1, 1.7, 1.9]	59.4	73.3	**80.6**	**83.8**
	max	[0.526, 0.588, 1, 1.7, 1.9]	59	73.3	79	83.5
	min	[0.526, 0.588, 1, 1.7, 1.9]	56.8	72.4	79	83.5
Tokyo-night	baseline	-	36.2	50.5	55.2	62.9
	avg	[0.667, 1, 1.5]	32.4	**53.3**	**60.0**	**65.7**
	sum	[0.526, 0.588, 1, 1.7, 1.9]	30.5	46.7	**60.0**	**65.7**
	max	[0.526, 0.588, 1, 1.7, 1.9]	30.5	45.7	**56.2**	62.9
	min	[0.526, 0.588, 1, 1.7, 1.9]	25.7	48.6	**57.1**	**64.8**

C. *Smart Data Augmentation*

Night domain adaptation has always been a challenging task to perform, especially when the training database is characterized only by daylight images (in our case, we can do training only in Pitts30k which is composed only by daylight images). In this section we propose a very simple change on the database that allows to obtain models that have slightly better performance when tested on night domain images by just changing the brightness of the training queries, with the advantage of having no negative influence on the overall training time. In particular, the method *adjust_brightness* [7] implemented in PyTorch transform class has been applied to the training query images, making these pictures darker with respect to the original one, as shown in Fig. 5

Fig. 5. Example of different light. (a) Original image; (b) brightness : 0.10; (c) brightness : 0.30; (d) brightness : 1.50.

TABLE IV
MODEL PERFORMANCE ON TOKYO-NIGHT

brightness	R@1	R@5	R@10	R@20
	36.2*	50.5*	55.2*	62.9*
0.05	31.4	50.5	54.3	63.8
0.08	**40.0**	49.5	59	**68.6**
0.10	**38.1**	50.5	**61.9**	**72.4**
0.12	34.3	50.5	56.2	60.0
0.15	35.2	46.7	54.3	61.9
0.20	30.5	46.7	50.5	62.9
0.30	35.2	47.6	55.2	64.8
0.50	34.3	51.4	56.2	64.8

*baseline performance

The example images shows how images changes with respect to the parameter, and shows that higher the parameter the brighter the output image is.

By fine-tuning the parameter of brightness, we can see a range where there is the best performance on test set (Tokyo-night, the only one available for our experiments). Additional experiments are done on changing slightly other parameters provided by the PyTorch framework such as adjust_contrast

[1]Note: only the best combinations of methods and resolutions are reported for each dataset.

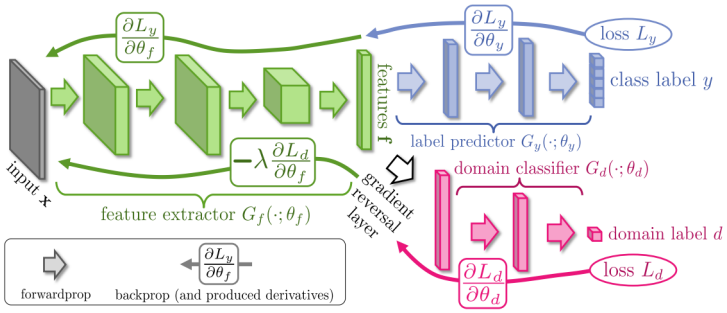

Fig. 6. Domain Adaptation schema

and adjust_saturation, but they did not have improvement with respect to the baseline. It is important to notice that by adjusting the brightness of the image, we are reducing the light of all the pixels. Sometimes this may lead to making some elements completely dark like shadow part, buildings and elements with relative dark color while our scope is generally to make pixel darker because of the night, but still keeping the important colors of the objects because if the same picture was taken during the night it would have kept its original color because of the streetlamps or the object's light itself.

D. *Domain Adaptation*

A further step that could be done to obtain better trained models for the night domain is the Domain Adaptation with consist on adding an extra neural network as domain classifier and its function is to classifier the output of the backbone whether it is from the source distribution or the target distribution. For this step we create a new training dataset that contains queries and dataset as Pitts30k as training source and queries of Tokyo-night as target source since our scope is to tackle the night domain. Passing both sources to the feature extractor (backbone), it will try to perform some transformation on the source instances such that the transformed instances appear as if it is coming from the same distribution. The basic idea is that we would like the domain classifier to not be able to classify correctly the domain of the transformed instances. This can be achieved by training both the feature extractor and domain classifier in such a way that the feature extractor is trained to maximize the domain classification loss, while domain classifier will try to minimize the domain classification loss. This approach is similar to adversarial training wherein the feature extractor is trying to confuse domain classifier by bringing the two distributions closer. Meanwhile, for the transformed source instances, the label predictor (NetVLAD) will be trained for predicting the labels only on the source instances. The feature extractor will therefore be trained to minimize classification loss of the label predictor and maximize classification loss of domain predictor, thus obtaining a min-max strategy. The label predictor and domain predictor will be trained to minimize there respective classification loss. Thus using the above three components, our description aggregator will learn to produce discriminative and domain-invariant features.

An overview of the approach is shown in Fig. 6.

For training the feature extractor in order to maximize the classification loss of domain predictor, a Gradient Reversal Layer [12] is placed between the feature extractor and domain classifier and it acts as an identity transform in the forward pass, while in the backward pass multiplies the gradient by $-\lambda$, where $\lambda > 0$. The use of this layer effectively sets up a min-max strategy, where the discriminator tries to minimize the domain classification loss, that we choose to be the entropy loss, while the feature extractor learns to produce domain-invariant embeddings, acting as an adversary to the classifier.

For the experiment, we choose the train queries of the **Pitts30k** dataset as the source set, which contains 7416 images, and the test queries of the **Tokyo-night** dataset as the night-domain target set with 105 images. We can intuitively see the imbalance between the two datasets. In order to overcome this problem, [13] provides a good solution that consists on picking the images depending on a random number: since we only have two datasets, we divide the DataLoader's indexes by the number of classes and take the remainder. Because the index value has the same probability of being odd or even, we can pick the images from the source and target set with the same probability even if they are imbalanced. After this split we put the images into the feature extractor to compute the rest.

In this way, we construct an **unsupervised domain adaptation** structure that achieves a significant improvement over the baseline in the night domain. TABLE V shows the results.

TABLE V
RESULT WITH DOMAIN ADAP STRUCTURE

	Baseline		Domain Adaptation	
Datasets	Recall@1	Recall@5	Recall@1	Recall@5
Pitts30k	86.5	93.1	86.1	93.0
sf-xs	35.7	50.6	**40.2**	**53.1**
Tokyo-xs	65.7	75.2	**69.8**	**80.6**
Tokyo-night	36.2	50.5	**46.7**	**61.0**

V. Discussions and findings

This work tries to extend the Deep Visual Geo-localization benchmark code [1] implementing methods that can either improve robustness on *night domain* or lead to *general improvements*. Our experiments show how specific changes on the code can affect, to a greater or lesser extent, the performances.

Optimizer and Scheduler Using varied combinations of optimizers, schedulers, LR and weight decay we find out that performances are close to the baseline in most of the cases. However, the use of SGD optimizer combined with ReduceLROnPlateau performs slightly better than the baseline. Moreover, AdamW have performances very similar to the baseline and very narrowly increase in performances using a particular combination of parameters. Also ASGD has recalls very similar to the baseline's ones.

Multi-scale testing As empirically demonstrated in [1], using full resolution images might be superfluous. In our case, using different resolution images turned out to be useful. In fact, we obtained slightly or consistent increase in the performances, especially in **sf-xs** dataset.

Smart Data Augmentation As usually observed for deep models, data augmentation generally helps. Moreover, in our case the reduction of the brightness simulate a night environment that helps when the tests are performed in night domain. Particularly, the test performed on **Tokyo-night** present a narrowly increase in performances.

Domain Adaptation The use of a Domain Adaptation structure turned out to be really useful, in fact we were able to increase the baseline performances for all the datasets.

References

[1] Gabriele Berton, Riccardo Mereu, Gabriele Trivigno, Carlo Masone, Gabriela Csurka, Torsten Sattler, Barbara Caputo; Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 5396-5407

[2] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pa-jdla, and Josef Sivic. NetVLAD: CNN architecture for weakly supervised place recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 40(6):1437– 1451, 2018

[3] F. Radenovic, G. Tolias, and O. Chum. Fine-tuning CNN ´ Image Retrieval with No Human Annotation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2018

[4] B. T. Polyak, A. B. Juditsky. Acceleration of Stochastic Approximation by Averaging. SIAM Journal on Control and Optimization Volume 30 Issue 4 pp 838–855, 1992

[5] Jégou, H., Perronnin, F., Douze, M., Sánchez, J., Pérez, P., & Schmid, C. (2011). Aggregating local image descriptors into compact codes. IEEE transactions on pattern analysis and machine intelligence, 34(9), 1704-1716.

[6] Liu, Liu, Hongdong Li, and Yuchao Dai. "Stochastic attraction-repulsion embedding for large scale image localization." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019.

[7] https://pytorch.org/vision/main/generated/torchvision.transforms.functional.adjust_brightness.html

[8] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. International Conference on Learning Representations, 12 201

[9] Chollet, Francois. 2015. Keras: Deep Learning library for Theano and TensorFlow.

[10] Ilya Loshchilov, Frank Hutter. Decoupled Weight Decay Regularization. ArXiv, abs/1711.05101, 2019

[11] Loshchilov, Ilya, and Frank Hutter. "Sgdr: Stochastic gradient descent with warm restarts." arXiv preprint arXiv:1608.03983 (2016).

[12] Ganin, Yaroslav, and Victor Lempitsky. "Unsupervised domain adaptation by backpropagation." International conference on machine learning. PMLR, 2015.

[13] Berton G M, Paolicelli V, Masone C, et al. Adaptive-attentive geolocalization from few queries: A hybrid approach[C]//Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. 2021: 2918-2927.