Image Geolocalization Through Retrieval

Gioele Costa
Politecnico di Torino
s318947@studenti.polito.it

Federico Rondano
Politecnico di Torino
s292292@studenti.polito.it

Federico Sisci
Politecnico di Torino
s304657@studenti.polito.it

Abstract

In this work we address the Visual Place Recognition (VPR) task which consists in localizing a place depicted in a query image. Only computer vision's techniques are exploited. In the first place, we train our model according to the GSV-Cities framework. Next, we test our best model on two benchmark datasets. Finally, we analyze and compare the results obtained from using different model aggregators, loss functions, mining techniques and optimizers. We also visualize some queries and their predictions to better understand the reasoning behind our model's decisions. We show that the model utilizing MixVPR as aggregator outperforms other evaluated configurations.
Our code is available at this link.

1. Introduction

Visual place recognition (VPR) has become of increasing interest over the years and represents a crucial point in many applications of machine learning. It refers to a system's capability to identify if a location shown in a query image has been encountered before. This process involves checking a database of images from previously-visited sites and comparing the query image to those stored in the database.
This process is very ambitious due to the changes that can modify and make a place unrecognizable. Actually two or more images can look very different despite representing the same place if, for example, it has been photographed in different seasons or at different times during the day when the lighting is not the same. Or more, in a urban context, it's hard to distinguish locations due to repetitive structures such as similar buildings and vegetation.
To overcome these problems and to make image recognition as good as possible, throughout the years many approaches have been developed. An attempt is to cast the place recognition problem as an instance retrieval task: a query image location is estimated using the location of the most visually similar images, which are retrieved from a geotagged database.
In our project, taking advantage of Convolutional Neural Networks (CNNs), we present an analysis that aims to compare the performance of the most recent aggregation techniques, in combination with different losses, miners and optimizers, in order to understand which is the most effective in the VPR task.

2. Related Work

The image representation suitable for the comparison was initially obtained through hand-crafted aggregation techniques. For instance the SIFT algorithm (Scale-Invariant Features Transform) allowed to extract local descriptors identifying key points in an image that are stable under scale changes and rotation. These features were then aggregated into a single vector representation such as Bag-Of-Visual Words (BoVW) or VLAD, which captures information about the statistics of local descriptors aggregated over the image. Given the advances in the computer vision's field, a better performing representation consists in the aggregation of the features learned from Convolutional Neural Networks. A first relevant example was NetVLAD, an end-to-end trainable version of the VLAD layer which aggregates deep features into one descriptor. Due to the success of the latter technique, many variants have been proposed.

3. Methodology

3.1. Data collection and overview

Before delving into our experimental results, it is essential to perform an examination of the datasets used in our study, specifically the GSV_XS (training), SF_XS (validation and test) and Tokyo_XS (test) datasets. Before doing an accurate analysis for each dataset, we need to specify that all these datasets are reduced versions of other datasets, for example GSV_XS is a smaller format of GSV-CITIES which is an image dataset covering more than 40 cities across all continents over a 14-year period.

3.1.1 GSV_XS

The images are collected from Google Street View Time Machine (GSV-TM), providing geotagged data of urban environments with a highly accurate ground truth. The dataset includes a wide range of scenes representing streets, buildings, vehicles, and pedestrians. These were captured under various lighting and weather and conditions, and during a fourteen years lapse ensuring variability and robustness in training (Figure 1). In our case, as we can see in Figure 2, the dataset is sufficiently large to train deep learning models effectively and each image is annotated, in the file name, with relevant labels, such as city, place id, coordinates and the date in which it was taken.

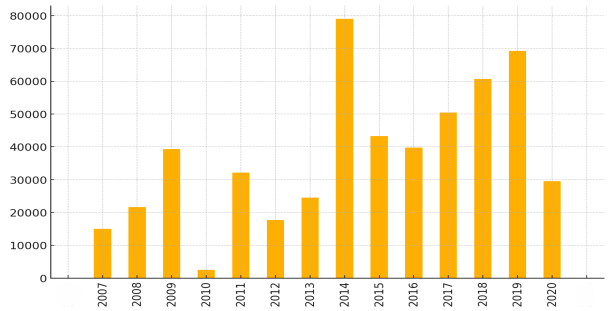


Figure 1. Distribution of the photos taken over the years.

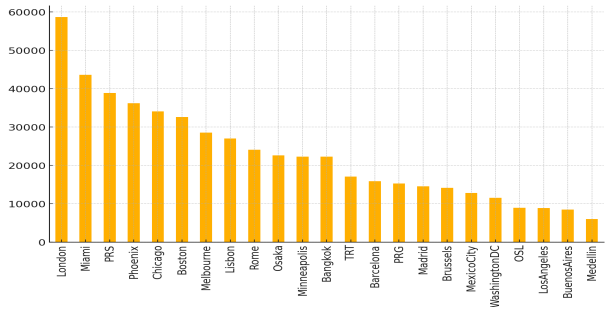


Figure 2. Number of images for every city.

3.1.2 SF_XS val

This dataset is used to select the best model during training and to tune some of the model hyper-parameters. It is designed to provide an efficient validation environment reflecting real-world urban conditions. The images are captured from specific and consistent locations within San Francisco. In Figure 3, we showcase that the dataset's structure is not generic but follows a precise path: the same scenes (equal place and coordinates) are repeatedly observed from different angles, in a way that every location's image set is rotated, for example of 0°, 30°, 60°, 90° and so on. We believe this provides robustness to rotational changes. Each image is annotated similarly to the training ones, providing the necessary labels in the file name. All the validation and test datasets are splitted into two subdirectories: database and queries.

3.1.3 SF_XS test

Here, images are still selected from various locations in San Francisco, offering a diverse and representative sample of the city's urban landscape and scenes. It includes a wide range of different streets, buildings, and urban elements, captured under various lighting and weather conditions to ensure a robust evaluation. Each image in the test set is annotated with the same level of detail as those in the training and validation set ensuring consistency in evaluation metrics. Furthermore, unlike in the validation dataset, here we do not have the orientation parameter. This dataset is used to evaluate the model's performance, providing information about its capabilities and effectiveness in handling real-world situations.

3.1.4 Tokyo_XS

In order to analyze our model in more depth, we employed the Tokyo_XS dataset as an additional test set. The photos are taken from Tokyo, which is the Japan's capital and one of the most populous cities in the country. This dataset is smaller than SF_XS presented before and is designed to provide a diverse and challenging set of urban imagery. Indeed, it allows to check that our model does not overfit to any specific urban environment: if we obtain good results on a different city, architectural styles, and cultural contexts, it means that the model is able to generalize well.

3.2. Customized ResNet Architecture

In our model we use a ResNet18 architecture as a backbone, which allows us to freeze and crop specific layers. It consists of several residual blocks, each one composed of convolutional, batch normalization and ReLU activation layers, which produce feature maps, representing the input image in terms of feature extracted by the filters applied.
In particular, specific layers can be frozen to prevent their weights from being updated during training.
Furthermore, it is possible to crop specific residual layers: this is useful for customizing the depth of the network. If the third or fourth layer is cropped, the number of output channels is adjusted accordingly.
The input images are resized, normalized and transformed into tensors of dimension $[224 \times 224]$. Normalization makes the pixel values more balanced and decreases the variance. The pipeline for the data flow through the backbone is composed of different steps: an initial convolutional layer, a max pooling operation and three residual blocks (we truncate at the conv3 layer). The output is of size $[256, 14, 14]$.

(a) Orientation: 30 degrees (b) 60 degrees (c) 90 degrees (d) 150 degrees

(e) 180 degrees (f) 240 degrees (g) 300 degrees (h) 360 degrees

Figure 3. Example of a sequence of records for SF_XS val dataset: same place (coordinates), different point of view angle.

3.3. Aggregators

Now we provide a detailed description of the considered aggregators:

3.3.1 AVG

Average Pooling is a down-sampling technique used in Convolutional Neural Networks. Its purpose is to reduce the spatial dimensions (height and width) of the input tensor while keeping unchanged the depth (number of channels). The AVG layer performs an averaging of the input elements, generating a smoothed version of the feature maps. This can be useful in reducing the impact of noise and minor variations in the input, helping prevent overfitting
The resulting tensor is flattened and normalized using L^2 normalization.

3.3.2 GeM

Generalized Mean Pooling is a pooling layer that extends the concept of average pooling by introducing a parameterized power followed by a root. p is a learnable hyperparameter that controls the pooling behaviour. Note that when $p = 1$, GeM is equivalent to average pooling explained before and when $p \to \infty$ it becomes a max pooling (takes directly the maximum value in the filter). In particular, the values 2 and 3 for p provide a balance between the smoothness of average pooling and the sensitivity to high-intensity features of max pooling.

3.3.3 ConvAP

Convolutional Attention Poooling is structured with a convolution followed by adaptive pooling of which the output has a specific size. The expression is:

$$y = \text{AAP}_{s_1 \times s_2}(\text{Conv}_{1\times 1}(x))$$

where x is the input, $\text{Conv}_{1\times 1}$ is a 2D convolution with a 1×1 kernel which reduces the number of channels and applies a learned weight to each channel. In particular it linearly projects each spatial descriptor into a compact d-dimensional representation space. $\text{AAP}_{s_1 \times s_2}$ is an Adaptive Average Pooling which determines the stride and the window size of the average pooling operation so that the output feature map is of size $s1 \times s2 \times d$, where d is the number of channels. The choice of these parameters allows us focusing on the most relevant features. Finally the output is flattened and L_2 normalized.

3.3.4 MixVPR

Unlike traditional methods that focus on local features, MixVPR aggregates mid-level features using a stack of MLP blocks called Feature-Mixer. The starting point is a 3D tensor F, considered as a set of 2D features of size $h \times w$:

$$F = \{X_i\}, \quad i = \{1, \ldots, c\}$$

where X_i corresponds to the i-th activation map, with each feature map carrying specific information regarding the image. We reshape each 2D feature X_i into a 1D representation resulting in flattened feature maps $F \in \mathbb{R}^{c \times n}$, where $n = h \times w$.

The flattened feature maps are then fed into the Feature-Mixer: a cascade of L MLP blocks of identical structure. The Feature-Mixer incorporates global relationships into each $X_i \in F$ as follows:

$$X_i \leftarrow W_2(\sigma(W_1 X_i)) + X_i, \quad i = \{1, \ldots, c\}$$

where W_1 and W_2 are the weights of two fully connected layers that compose the MLP, and σ is a nonlinearity (ReLU in our case).
The Feature-Mixer generates an output $Z \in \mathbb{R}^{c \times n}$ of the same shape of the input since all the subsequent blocks have an identical structure:

$$Z = \text{FM}_L(\text{FM}_{L-1}(\ldots \text{FM}_1(F) \ldots))$$

The output Z is usually highly dimensional, so its dimensionality is reduced depth-wise (channel-wise) and then row-wise successively using two fully connected layers. This process can be seen as a weighted pooling operation that controls the size of the final global descriptor. First, a depth-wise projection maps Z from $\mathbb{R}^{c \times n}$ to $\mathbb{R}^{d \times n}$:

$$Z' = W_d(\text{Transpose}(Z))$$

where W_d are the weights of a fully connected layer. Next, a row-wise projection maps Z' from $\mathbb{R}^{d \times n}$ to $\mathbb{R}^{d \times r}$:

$$O = W_r(\text{Transpose}(Z'))$$

where W_r are the weights of another fully connected layer. The final output O has a dimensionality of $d \times r$, which is flattened and L2-normalized, as is customary in VPR.

3.4. Loss functions

In the VPR context it is common to compute tuples (q, P^q, N^q) where q is the query image, P^q is a subset of positives (images depicting the same place) and N^q is a subset of negatives (images depcting different places). The loss function operates on these pairs/triplets to minimize an objective that maximizes the similarity between instances of the same place and minimizes the ones of different places. Positives images do not necessarily depict the same place, since they might represent different orientation of the same panorama, so in most techniques weakly supervised triplet loss is used. Here we leverage the accurate labels provided by the GSV-Cities dataset so that losses can be used in a supervised manner. The creation of the tuples is performed by the miners of which we provide more details in the next section. Now we make a description of the loss used.

3.4.1 Contrastive Loss

Unlike other loss functions that sum over samples, the contrastive loss runs over pair of samples. The general idea is to maximize the similarity between positive pairs and minimizing it for negative pairs. Formally:

$$L_{\text{contrast}} = (1 - I_{ij})[S_{ij} - m]_+ - I_{ij} S_{ij}$$

where m is a margin that avoids optimizing indefinitely the negative pairs that are already dissimilar enough. $I_{ij} = 1$ indicates a positive pair (0 otherwise) and S_{ij} is the cosine similarity of the pair (z_i, z_j), where z_i and z_j are learned representations of two images.

3.4.2 Triplet Margin Loss

This particular loss function operates on triplets of images: an anchor, a positive example, and a negative example. The aim is to ensure that the positive example is closer to the anchor than the negative example by at least a margin m, applying better discrimination between similar and dissimilar images. Formally we have:

$$L_{\text{triplet}} = [S_{ij} - S_{ik} + m]_+$$

where the objective is to reduce the similarity S_{ik} of the negative pair (z_i, z_k) and at the same time increase the similarity S_{ij} of the positive pair (z_i, z_j).

3.4.3 Multi Similarity Loss

Finally we test the Multi Similarity loss that incorporates advanced pair weighting schemes to better balance the importance of positive and negative pairs in a batch. This is made by adjusting the contributions of pairs based on their similarities, focusing on the most informative relationships during training. Mathematically:

$$L_{MS} = \frac{1}{N} \sum_{i=1}^{N} \left\{ \frac{1}{\alpha} \log \left[1 + \sum_{j \in P_i} e^{-\alpha \cdot (S_{ij} - m)} \right] + \frac{1}{\beta} \log \left[1 + \sum_{k \in N_i} e^{-\beta \cdot (S_{ik} - m)} \right] \right\}$$

where for each instance z_i in the batch P_i is the set of indices that formed a positive pairs and, on the contrary, N_i is the set of indices which formed a negative pairs. Finally, α, β and m are hyperparameters that allows to control the weighting scheme.

3.5. Miners

In the *VPR* task, a miner uses similarity metrics to select the positive image and the negative image to construct the training triplet. When weakly supervised losses are considered it is usual to exploit hard negative mining which consists into selecting the hardest negatives from the entire dataset to compensate the presence of easy positive images.

In our case, given the high accuracy of our dataset, online hard mining is leveraged which selects positives and negatives from the current batch (i.e, online) and so it is less computationally expensive.
We decided to use the following losses and miners coupled:

- Contrastive Loss + Pair Margin Miner
- Triplet Margin Loss + Triplet Margin Miner
- Multi Similarity Loss + Multi Similarity Miner

3.6. Optimizers

In the machine learning context, an optimizer is an algorithm which adjusts the hyperparameters of the model searching the optimal parameter's combination. Its generic purpose is to minimize the loss function's value in order to maximize the performance during the training step.
Let's see in details the different optimizers tested.

3.6.1 Adam

Adaptive Moment Estimation is a stochastic gradient descend optimizer which computes adaptive learning rates for each parameter. It's a combination of other two important optimizer: Momentum and RMSProp.
Mathematically, it updates the weights in this iteratively way:

$$w^{(t+1)} = w^{(t)} - \alpha \frac{p^{(t)}}{\sqrt{s^{(t)}}}$$

where α is the learning rate or step size that controls how much the parameters are updated at each iteration thus how quickly the algorithm converges to optimal solution. The choice of it is crucial and require a tradeoff because larger value provides faster convergence but can cause divergence or oscillations and smaller value brings slower convergence but more stable and precise adjustments to the model weights.
$p^{(t)}$ and $s^{(t)}$ are respectively the momentum gradient and the mean square gradient at step t:

$$p^{(t)} = \beta_1 \cdot p^{(t-1)} + (1 - \beta_1) \cdot g^{(t)}$$

$$s^{(t)} = \beta_2 \cdot s^{(t-1)} + (1 - \beta_2) \cdot (g^{(t)})^2$$

where $g^{(t)}$ is the minibatch gradient and β_i with $i = \{1, 2\}$ is a decay factor.

3.6.2 AdamW

AdamW is made adding to Adam optimizer a penalty proportional to the weights multiplied for a weight decay factor λ:

$$w^{(t+1)} = w^{(t)} - \alpha \frac{p^{(t)}}{\sqrt{s^{(t)}}} - \alpha \lambda w^{(t)}$$

It is a regularization technique that adds a penalty to the weights iteration function. Such as in the case of the parameter α, we have to make a tradeoff because larger value of λ leads to a stronger regularization which can help prevent overfitting but might provides underfitting if too high. Contrariwise smaller value allows the model to fit the training data more closely by applying smaller corrections. Observe that if weight decay factor $\lambda = 0$, we obtain Adam optimizer.

4. Experimental results

In this section we describe the experiments done on the datasets.
For every trial, we performed 30 training epochs on the GSV_XS dataset. The aim is comparing the Recall results obtained using the different aggregators presented above and understanding why some of them are capable to perform better for the task of Visual Place Recognition.
In particular, this is the inference process. Initially the descriptors of all queries and reference images are computed. Then, for each query, 25 reference images are predicted according to the Euclidean distance and sorted by increasing order. Now, the Recall@k (Rk in the tables below), with $k = 1, 5, 10, 15, 20$ is computed. This measure is the portion of the queries for which, among the first k predictions, there is at least one correctly predicted image according to the ground truth.
For each query image, the ground truth is built in this way: we iterate through all reference images dataset and calculate the distance (on UTM coordinates). If it is less than or equal to a specified threshold (25 meters), the index of the reference image is added to the ground truth list for that query. This is done for SF_XS val, SF_XS test and Tokyo_XS.

4.1. Results with different aggregators

The first test we do is to evaluate the performance of aggregators presented before with the same loss function and miner. We choose the Multi Similarity Loss and Miner and fix the following setting of aggregator's hyperparamters:

- Gem, p = 3;
- ConvAP, s_1 = 2, s_2 = 2;
- MixVPR, mix_depth = 5.

Note from the results showed in Tables 1, 2 and 3 that the model with MixVPR features pooling layer performs better than the others due to his ability to integrate global relationships into each feature map.

Aggregator	R1	R5	R10	R15	R20
AVG	53.85	69.41	75.20	78.51	80.76
GeM	56.66	71.66	77.56	80.73	82.73
ConvAP	71.64	79.06	82.03	83.92	85.14
MixVPR	**76.27**	**83.50**	**86.60**	**88.43**	**89.42**

Table 1. Results obtained on SF_XS val.

Aggregator	R1	R5	R10	R15	R20
AVG	21.30	34.30	41.30	44.70	47.20
GeM	23.30	35.60	43.30	47.20	49.50
ConvAP	46.40	58.30	65.00	67.90	69.80
MixVPR	**55.10**	**66.50**	**71.50**	**73.30**	**74.80**

Table 2. Results obtained on SF_XS test.

Aggregator	R1	R5	R10	R15	R20
AVG	29.21	47.62	55.24	63.17	68.25
GeM	35.24	56.19	62.22	69.52	74.60
ConvAP	68.89	82.86	86.98	86.98	87.62
MixVPR	**73.65**	**85.40**	**89.52**	**91.43**	**93.65**

Table 3. Results obtained on Tokyo_XS.

4.2. Results of different losses with MixVPR aggregator

From the analysis above it is clear that MixVPR has great performance than other aggregators. For this reason we decided to test it with different losses and associated miners. In particular, we test the following combinations: Contrastive Loss + Pair Margin Miner, Triplet Loss + Triplet Margin Miner and MultiSimilarity Loss + Multi Similarity Miner.
Let's show the results in Table 4, 5 and 6.

Loss	R1	R5	R10	R15	R20
Contrastive	71.37	79.34	82.60	84.34	85.75
Triplet	64.22	76.15	80.90	83.52	85.40
MultiSimilarity	**76.27**	**83.50**	**86.60**	**88.43**	**89.42**

Table 4. Results obtained on SF_XS val.

Loss	R1	R5	R10	R15	R20
Contrastive	50.30	62.50	67.30	69.20	70.60
Triplet	27.50	43.10	49.10	53.30	55.40
MultiSimilarity	**55.10**	**66.50**	**71.50**	**73.30**	**74.80**

Table 5. Results obtained on SF_XS test.

Loss	R1	R5	R10	R15	R20
Contrastive	71.11	84.76	87.62	88.57	90.48
Triplet	37.78	60.32	71.75	77.78	79.68
MultiSimilarity	**73.65**	**85.40**	**89.52**	**91.43**	**93.65**

Table 6. Results obtained on Tokyo_XS.

Note that the combination Constrastive and MultiSimilarity have better results than Triplet; farther MultiSmilarity presents the best score of Recall metrics. We expected this due to its more complex and suitable structure.
The results obtained with Triplet loss were particularly bad because the possible number of triplets grows cubically, and consequentially also the uninformative triplets.

4.3. Optimizer AdamW analysis

Considering the above, we decide to use the best configuration to test another possible extension in our project: the AdamW optimizer. We first performed a 10 epochs analisys to better understand which combination of hyperparameters is better for our model and our task. We tested the following configurations:

- 1. learning rate = 0.00005, weight decay = 0.001;
- 2. learning rate = 0.0005, weight decay = 0.001;
- 3. learning rate = 0.001, weight decay = 0.001;
- 4. learning rate = 0.0005, weight decay = 0.0001.

The results are shown in Tables 7, 8 and 9. In order to have a better point of view of which is the best configuration, we decide also to make a plot of results in Figure 4, 5 and 6.

Configuration	R1	R5	R10	R15	R20
1	71.17	79.57	83.16	84.74	86.28
2	76.62	84.14	86.91	88.64	89.78
3	**77.09**	**84.81**	**87.73**	**89.43**	**90.68**
4	76.72	84.29	87.45	89.25	90.28

Table 7. Results obtained on SF_XS val.

Configuration	R1	R5	R10	R15	R20
1	48.10	62.40	67.30	69.80	70.70
2	54.10	66.50	70.20	72.80	74.70
3	55.60	66.20	71.20	**73.70**	**75.50**
4	**56.10**	**67.90**	**71.70**	72.60	74.30

Table 8. Results obtained on SF_XS test.

Configuration	R1	R5	R10	R15	R20
1	66.35	79.68	83.49	89.21	91.43
2	73.33	84.13	86.67	87.94	90.79
3	71.11	84.13	87.62	89.52	90.79
4	**73.97**	**85.08**	**90.16**	**91.43**	**92.06**

Table 9. Results obtained on Tokyo_XS.

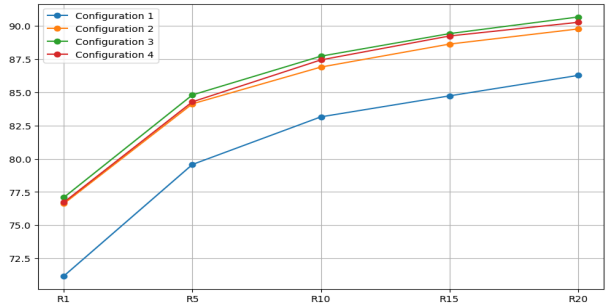


Figure 4. Results obtained on SF_XS val.

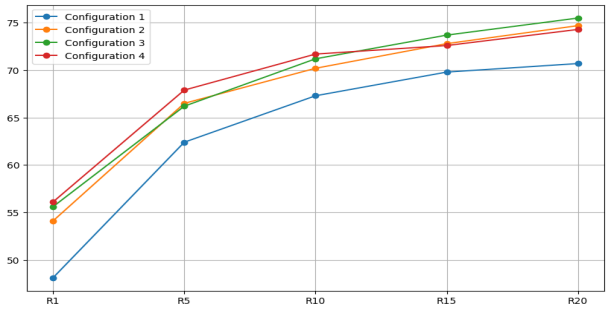


Figure 5. Results obtained on SF_XS test.

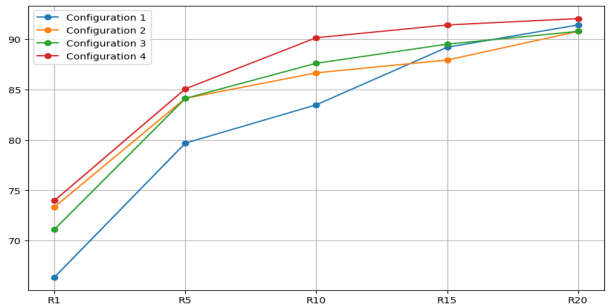


Figure 6. Results obtained on Tokyo_XS.

In general, we are satisfied with the results obtained: considering that they were only trained on 10 epochs, underlining the functionality in AdamW of adding a penalty parameter to the Adam optimizer. In particular we can see that the first combination performs worse than other (beacause we tried a smaller learning rate). Instead the configuration 3. and 4. offer better results.
For this reason we perform a complete 30 epochs training run on these best configurations. We report the outcome in Tables 10 and 11.

3.	R1	R5	R10	R15	R20
SF_XS val	78.19	86.04	88.93	90.47	91.63
SF_XS test	56.30	67.80	72.00	74.50	76.40
Tokyo_XS	72.38	83.81	87.30	89.52	91.43

Table 10. Results with 30 epochs on configuration 3.

4.	R1	R5	R10	R15	R20
SF_XS val	77.82	85.66	88.48	90.13	91.37
SF_XS test	55.80	67.40	71.40	73.40	74.90
Tokyo_XS	72.70	85.08	89.52	90.79	91.75

Table 11. Results with 30 epochs on configuration 4.

Note that, it this two cases, a little overfit occurs on San Francisco datasets. In fact, the performance on Tokyo test was better with Adam optimizer.
Trying to summarize what we obtained in this section and to find out what is the best configuration for our model between the ones we tested, we present a graphical comparison in plot 7, 8 and 9. Note that we reported the results with the best combination of loss and miner.

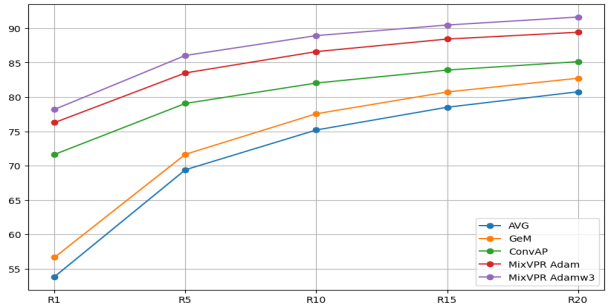


Figure 7. Results obtained on SF_XS val.

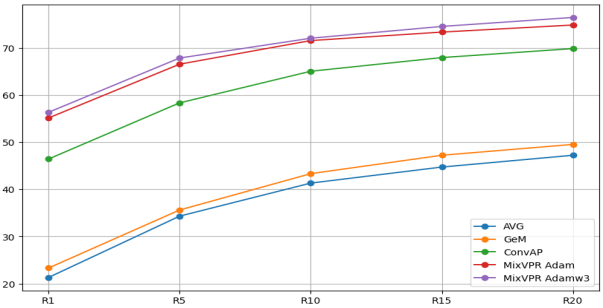


Figure 8. Results obtained on SF_XS test.

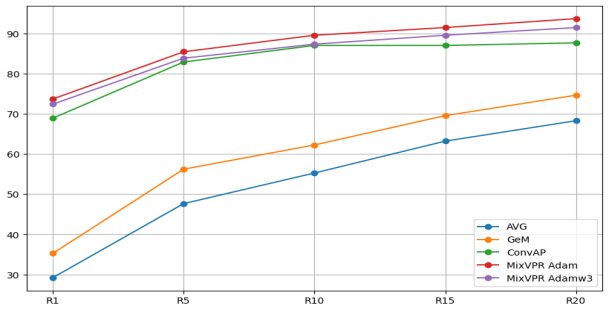


Figure 9. Results obtained on Tokyo_XS.

5. Conclusion

In this work we compare different VPR techniques according to their performance on this task.
First of all, we studied all the datasets trying to understand their structure and their main properties. Then, we trained our model on a very accurate dataset for this task: a reduced version of GSV-Cities dataset. This allowed us to make an evaluation from the simplest aggregator (AVG) to the more complex ones (ConvAP, MixVPR) on our test datasets. As expected, the latter ones obtained better performances.
Also, we choose the best model to test some losses and optimizers configurations, understanding the main reasons behind the different choices performances.

References

[1] Amar Ali-bey, Brahim Chaib-draa, and Philippe Giguère. GSV CITIES: TOWARD APPROPRIATE SUPERVISED VISUAL PLACE RECOGNITION

[2] Amar Ali-bey, Brahim Chaib-draa, and Philippe Giguere. MixVPR: Feature Mixing for Visual Place Recognition

[3] Zhenxun Zhuang, Mingrui Liu, Ashok Cutkosky, and Francesco Orabona. Understanding AdamW through Proximal Methods and Scale-Freeness

[4] Filip Radenovic, Giorgos Tolias, and Ondrey Chum. Fine-tuning CNN Image Retrieval with No Human Annotation

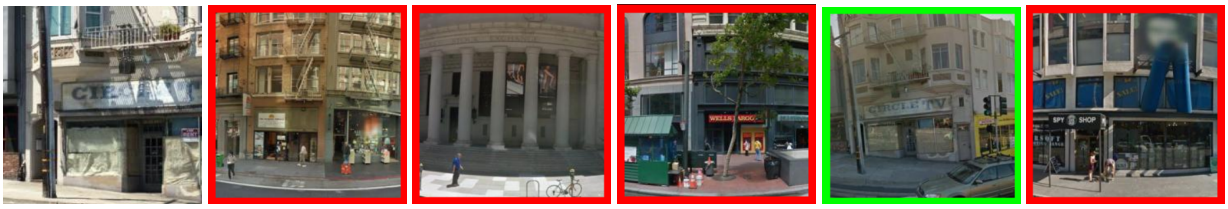
[5] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In Defense of the Triplet Loss for Person Re-Identification

[6] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef SivicNetVLAD. CNN architecture for weakly supervised place recognition

Appendices

Here we perform a qualitative analysis illustrating five randomly selected queries from SF_XS test and Tokyo_XS with their nearest prediction from each aggregator.

(a) Query 1 SF_XS test (b) AVG (c) Gem (d) ConvAP (e) MixVPR Adam (f) MixVPR AdamW



(a) Query 2 SF_XS test (b) AVG (c) Gem (d) ConvAP (e) MixVPR Adam (f) MixVPR AdamW

(a) Query 3 SF_XS test (b) AVG (c) Gem (d) ConvAP (e) MixVPR Adam (f) MixVPR AdamW

(a) Query 4 SF_XS test (b) AVG (c) Gem (d) ConvAP (e) MixVPR Adam (f) MixVPR AdamW

(a) Query 5 SF_XS test (b) AVG (c) Gem (d) ConvAP (e) MixVPR Adam (f) MixVPR AdamW

(a) Query 1 Tokyo test (b) AVG (c) Gem (d) ConvAP (e) MixVPR Adam (f) MixVPR AdamW

(a) Query 2 Tokyo test (b) AVG (c) Gem (d) ConvAP (e) MixVPR Adam (f) MixVPR AdamW

(a) Query 3 Tokyo test (b) AVG (c) Gem (d) ConvAP (e) MixVPR Adam (f) MixVPR AdamW

(a) Query 4 Tokyo test (b) AVG (c) Gem (d) ConvAP (e) MixVPR Adam (f) MixVPR AdamW

(a) Query 5 Tokyo test (b) AVG (c) Gem (d) ConvAP (e) MixVPR Adam (f) MixVPR AdamW