ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scuola di Scienze

Corso di Laurea in Informatica

Foundation and Frontiers in Visual Place Recognition: Advancing Architecture Evaluation and Adaptive Memory Learning

Relatore
Dott.
Lorenzo Pellegrini

Presentata da
Matteo Fornaini

Correlatore
Dott.
Matteo Scucchia

I Sessione
Anno Accademico 2024/2025

Abstract

Visual Place Recognition (VPR) refers to the task of identifying the geographic or semantic location depicted in a generic image. This problem is a particularly crucial component for Simultaneous Localization and Mapping (SLAM) in autonomous robotics in an environment in which GPS is not available, as it allows the loop closure algorithm for path correction. However, the practical use of VPR systems is constrained by conventional metrics used for model evaluation, generalization across changing conditions, and the impossibility of complete model retraining on resource-constrained devices.

This work deals with the limitations of standard ranking metrics in VPR by introducing an evaluation methodology based on the analysis of cosine similarity distributions, focusing on Average Precision (AP) to take into account the model's discriminative ability; finally, it proposes and validates a model-agnostic improvement for a continual learning strategy, where the proposed intelligent memory management system improves performance with minimal retraining epochs.

Sommario

Con Visual Place Recognition (VPR) ci si riferisce al compito di identificare la posizione geografica o semantica rappresentata in un'immagine qualsiasi. Si tratta di una componente particolarmente importante per la localizzazione e la mappatura simultanea (SLAM) nella navigazione autonoma in un ambiente in cui il GPS non sia disponibile, in quanto consente ad un algoritmo di loop closure di correggere gli errori accumulati nella rilevazione del percorso. L'uso pratico dei sistemi VPR è però condizionato dalle metriche di valutazione degli attuali modelli, dalla generalizzazione in condizioni mutevoli e dall'impossibilità di riaddestrare completamente i modelli su dispostivi con risorse limitate a disposizione.

Questo lavoro affronta i limiti delle metriche di ranking standard usate nel VPR, introducendo una metodologia di valutazione basata sull'analisi delle distribuzioni di cosine similarity e utilizzando l'Average Precision (AP) per tenere conto della capacità discriminativa del modello; infine, propone e poi convalida un miglioramento dei modelli di apprendimento continuo, in cui un nuovo sistema intelligente di gestione della memoria migliora le prestazioni con un basso numero di epoche di retraining.

Contents

Introduction

Visual Place Recognition (VPR) refers to the task of identifying the geographic or semantic location depicted in a generic image. While VPR can be applied in various domains, the main field for which it is used is autonomous navigation, as it allows autonomous robots to recognize previously visited places from visual input alone. This is a necessary component of the Simultaneous Localization and Mapping (SLAM) algorithm when GPS is not available, as it enables a cornerstone of SLAM: the loop closure algorithm, which corrects the drift that the robot accumulates when moving, when it recognizes the same place twice.

Motivation

This thesis is motivated by the need to develop and evaluate VPR systems that can adapt to diverse environments, with an emphasis on the SLAM task. Rather than simply ranking architectures, this work focuses on testing how different VPR algorithms and models perform, adapt, understand, and generalize across different datasets and conditions. Moreover, while many deep learning-based VPR models have demonstrated high performance in offline batch learning, on a real robotic platform, it is often computationally infeasible to store and retrain the entire history of observed data each time the environment is revisited. It is therefore necessary to have efficient continual learning strategies that can update the VPR model with new information without suffering from catastrophic forgetting, all without exceeding the hardware constraints of the robot.

Furthermore, the success of a VPR system within a SLAM framework depends not just on its ability to rank a correct match well, but mainly on its capacity to discriminate between true re-visitations and visually similar but distinct locations (perceptual aliasing). A high rate of false positives can lead to inaccurate mapping and catastrophic failures in localization. This motivates the use of metrics that are different from standard retrieval metrics, reflecting more accurately the discriminative ability of the learned descriptor extractor.

Problem Statement

This thesis addresses three problems in the domain of Visual Place Recognition:

1. **Evaluating Foundational VPR Architectures:** Many architectures and models have been proposed in the VPR field, but there is not yet a study to evaluate, rank, and understand the decisions taken by those models on different datasets and environments. This thesis analyzes the foundational VPR architectures to date by evaluating them

on multiple datasets, as well as testing their generalization and using explainability techniques to analyze the decisions taken. Furthermore, an analysis of the decision quality as a separation of descriptors is performed on all tests.

2. **Limitations of Standard VPR Metrics:** Conventional metrics for VPR, such as Recall@K (R@K) and Average Precision@K (AP@K), evaluate the ranking performance of a model on the first k elements. However, they do not comprehensively capture its ability to find a threshold between descriptors of the same or different places. This work proposes that a direct analysis of the distribution of descriptor similarities is necessary for a complete evaluation. The problem is to identify and validate a metric that effectively quantifies this discriminative capability, avoiding the existing statistical and classification measures that are not adapted to the imbalanced nature of the VPR task.

3. **Efficient Continual Learning for VPR:** Standard approaches for updating VPR models in changing environments often require extensive retraining, which is impractical for autonomous robots with limited computational resources. A model-agnostic framework that allows for rapid adaptation using only a limited memory of past experiences and current observations is thus needed. This work investigates memory storing techniques as a solution to make sure that the data kept for training are maximally informative, therefore improving generalization even with limited training epochs.

Thesis Outline

This thesis is structured with multiple chapters. It first establishes the necessary background, then details the proposed methodologies, and finally presents the experimental validation of these approaches.

- **Chapter 1: Background** gives an overview of the foundational concepts necessary to understand this work. It begins with an introduction to SLAM, then passes to the foundation of Neural Networks, with a focus on Convolutional Neural Networks and Visual Transformers. The chapter concludes with formal introductions to Continual Learning and Visual Place Recognition.

- **Chapter 2: Methodology** details instead the technical approach of this thesis. It describes the specific backbones and aggregators used to build the VPR models, the state-of-the-art continual learning models for VPR, the most used metrics for Visual Place Recognition, and the explainability techniques used during the tests.

- **Chapter 3: Datasets** contains a presentation of some classic VPR datasets for training and testing the models. These include standard VPR benchmarks like Tokyo-24/7 and GSV-Cities, as well as datasets designed for specific challenges such as indoor scenes, dynamic conditions, and continual learning.

- **Chapter 4: Experiments** reports the set of experiments conducted to compare architectures, analyze the proposed metrics, and check the improvement brought by the proposed method. The results are divided into two main sections: Batch Learning, which

establishes baseline performance, metrics, and cosine similarity separation, and Online Learning, which tests the effectiveness of the proposed continual learning framework.

- **Chapter 5: Conclusions** summarizes the key findings of this research, emphasizing the contributions made in addressing the problem statement, and discusses potential directions for future work.

Chapter 1

Background

1.1 Introduction to SLAM

The Simultaneous Localization and Mapping (SLAM) problem consists of the challenge of allowing a mobile robot to explore an unfamiliar environment, build an accurate map of its surroundings in real time, and determine its position within that map [1]. SLAM couples two interdependent tasks:

- **Mapping:** the robot incrementally constructs a representation of the environment using onboard sensors.
- **Localization:** the robot continuously estimates its pose relative to the evolving map.

Because mapping relies on knowing the robot's position and localization depends on having a map, SLAM must solve both together—a non-trivial problem that must take into account all the uncertainties of the sensors.

At each time step, the robot acquires new sensor measurements (e.g., laser scans, camera images, depth data), compares them against previous observations, and uses this comparison both to infer its motion and to enrich and improve the map. However, sensor noise and motion uncertainty accumulate over time, so without corrective mechanisms, the map and pose estimates drift. Depending on the sensors and algorithms chosen, SLAM systems draw on techniques from computer vision, graph theory, information theory, and other fields. [2]

1.1.1 Key Concepts

Odometry Odometry refers to the estimation of motion over time using sensor data [3]. Common variants include:

- **Wheel odometry:** Using wheel-encoder counts.
- **Visual odometry:** Tracking features across camera frames.
- **LiDAR odometry:** Registering successive laser scans.
- **Inertial odometry:** Integrating accelerometer and gyroscope readings.

Although odometry provides short-term motion estimates, it does not build a globally consistent map by itself. Instead, odometry typically serves as an input to the SLAM process, which uses these incremental motions along with loop closures, landmark detection, and global optimizations to maintain consistency. Simply using odometry data would result in a *drift*, an inevitable sum of small errors made by the sensors. [2]

Landmarks Landmarks are distinctive features of the environment that the robot can detect and recognize from different angles and poses. In visual SLAM, for example, "keypoints" or descriptors—highly discriminative image points that remain stable across viewpoints and positions, as in Fig. 1.1—are used to identify landmarks. During the exploration phase, landmarks are detected and added to the map. Because consistently re-observing the same landmarks reduces uncertainty, robust landmark detection is essential to achieve both an accurate localization and the ability to detect when the robot has returned to a previously visited area. [2]

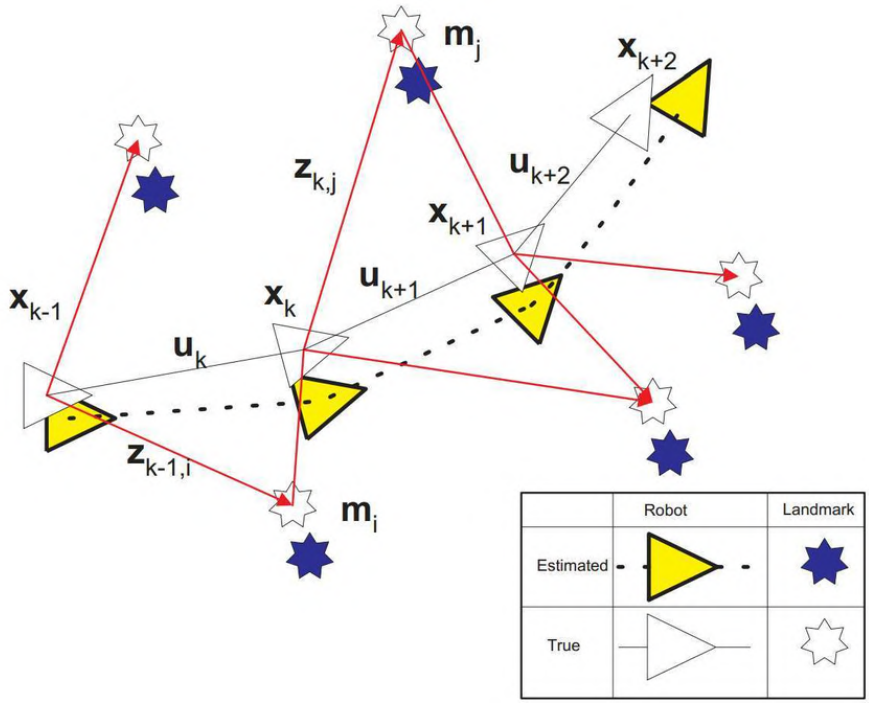


Figure 1.1: Landmarks being observed at different positions along the robot's trajectory. Figure courtesy: Time Bailey (DWB06)

Data Association Data association is the process of matching the observations of the current sensor to the features in the existing map—e.g., identifying the same landmark when seen twice (see Fig. 1.2). In the short term, recognizing the same landmark in successive frames supports motion estimation; in the long term, identifying a previously mapped location—known as *loop closure*—allows the system to correct the accumulated drift. However, real-world environments can be ambiguous (long, featureless corridors) or dynamic (moving

objects, changing lighting), making reliable data association one of SLAM's most difficult challenges. [2]

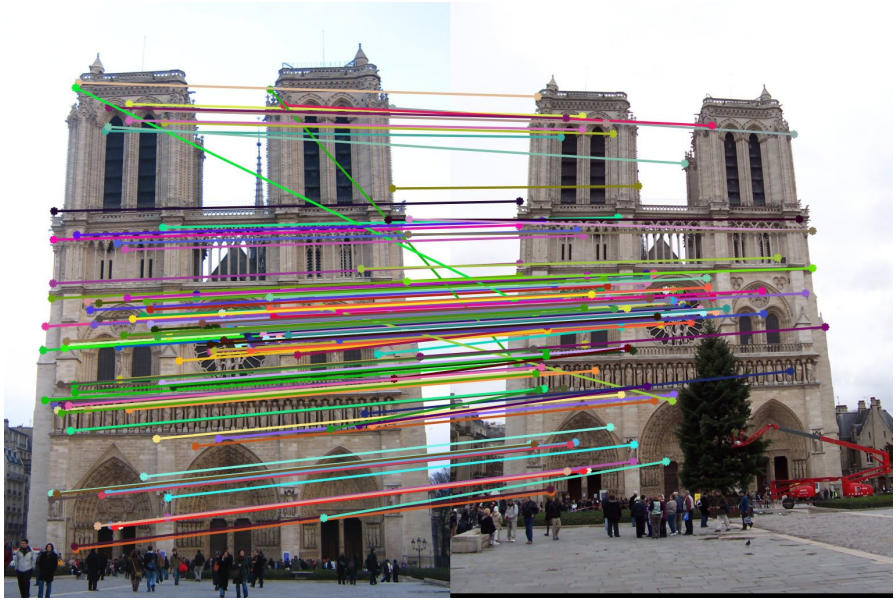


Figure 1.2: Association of similar points in different images.

Loop Closure Detection Loop closure detection determines whether the robot has returned to a previously visited spot by analyzing its sensor data. Since incremental motion estimates invariably drift due to noise, successfully detecting and enforcing loop closures allows the SLAM back-end to realign the map and correct the robot's pose, dramatically improving global consistency (see Fig. 1.3). [4]
When operating in areas where GPS is unreliable or unavailable (such as indoors, in cluttered urban canyons, or even in space explorations), robots cannot depend on external localization alone. SLAM gives autonomous systems the ability to navigate unknown areas, generate maps for tasks such as route planning, and adapt to dynamic environments. Moreover, by recognizing when it revisits the same place, the robot acquires a robust topological understanding of its surroundings, far beyond what odometry alone can offer.
However, misidentifications can have dire consequences: with false negative (missed loop), the robot fails to recognize a known place, allowing drift to persist; with a false positive (spurious loop), instead, the robot incorrectly merges a new location with an existing map feature, leading to catastrophic map distortions. Thus, robust loop closure detection—heavily dependent on accurate data association—is essential for any long-term autonomous navigation system. [2]

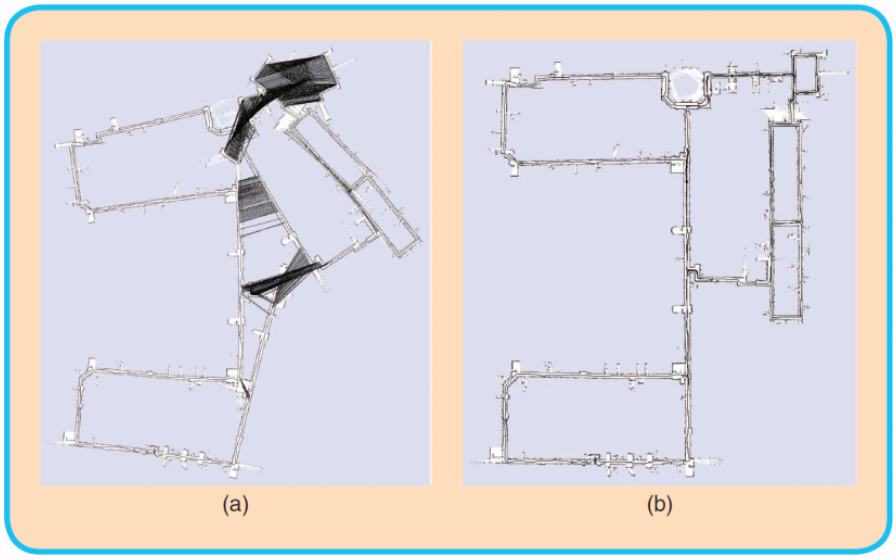


Figure 1.3: Illustration of loop closure detection showing the estimated path before (left, with drift) versus after (right, drift corrected) the loop closure realigns the map. [4]

1.1.2 Lifelong SLAM

Lifelong SLAM refers to long-term mapping and localization capable of managing changing environments to support extended robot autonomy. It remains an active research area, as practical, general-purpose solutions have yet to be finalized. Two key properties define Lifelong SLAM:

Robustness Robustness is the system's ability to handle failures in both software (algorithms) and hardware (sensors) over long navigations. Data association is the main challenge: fixed sets of visual landmarks may prove unreliable in the real world because of evolving settings. Failures can manifest with missed associations—preventing loop closure—or incorrect matches, causing mislocalization. To achieve robustness, SLAM frameworks must either maintain highly accurate data association or implement reversible processes that detect and undo incorrect associations, thus restoring map consistency. [2]

Scalability Scalability describes the system's capacity to map large-scale environments over time without performance degradation. In lifelong scenarios, the map size (landmarks, poses, graph size) grows without boundaries, increasing memory usage and matching costs. Approaches to improve scalability include compact map representations, efficient optimization algorithms, parallel computation, and intelligent memory management (e.g., deciding when to update or discard outdated map information and when to load or unload map segments). Designing strategies for selective forgetting and online map loading is critical to sustaining real-time performance as the environment evolves. [2]

1.1.3 Brief Mathematical Introduction

Moving through the environment and sensing landmarks, the robot at each time step t updates its state and acquires observations. These quantities are formalized as follows:

- $\mathbf{x}_t$: the robot's state (pose), combining position and orientation.
- $\mathbf{u}_t$: the control input applied at $t-1$, which drives the robot from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$.
- $\mathbf{z}_t$: the observation vector of sensor measurements; the ith landmark's measurement is z_t^i.
- $\mathbf{m}$: the map, whose ith landmark resides at location $\mathbf{m}_i$.

Let

$$X_{0:t} = \{\mathbf{x}_0, \dots, \mathbf{x}_t\}, \quad U_{1:t} = \{\mathbf{u}_1, \dots, \mathbf{u}_t\}, \quad Z_{0:t} = \{\mathbf{z}_0, \dots, \mathbf{z}_t\}$$

denote the histories of poses, controls, and observations up to time t.

In 3D SLAM, poses and controls live in the Special Euclidean group SE(3), i.e.

$$\mathrm{SE}(3) = \left\{ T = \begin{pmatrix} R & \mathbf{t} \\ 0 & 1 \end{pmatrix} \,\middle|\, R \in \mathrm{SO}(3),\, \mathbf{t} \in \mathbb{R}^3 \right\},$$

where $\mathrm{SO}(3) = \{R \in \mathbb{R}^{3\times 3} \mid RR^\top = I, \det R = 1\}$. For planar (2D) SLAM, we similarly use SE(2).

$\mathbf{x}_0$ is fixed at the map origin. Each control $\mathbf{u}_t$ is the relative transformation from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$. Hence

$$\mathbf{x}_t = \mathbf{x}_0 \circ \mathbf{u}_1 \circ \mathbf{u}_2 \circ \cdots \circ \mathbf{u}_t,$$

where "$\circ$" denotes group composition. Finally, the map $\mathbf{m}$ may be represented as a set of landmark coordinates $\{\mathbf{m}_i\}$ or in other formats depending on the sensors and estimation method employed. [2]
For further reading, [5] is suggested.

1.2 Introduction to Convolutional Neural Networks

1.2.1 Artificial Neural Networks

The idea of artificial neurons dates back to 1943 when McCulloch and Pitts[6] proposed a simplified mathematical model of a biological neuron. In the following decades, perceptrons [7] introduced the first trainable models, but various limitations halted early enthusiasm. The field was revitalized in the 1980s with the development of the backpropagation algorithm, allowing multilayer networks to be trained effectively. Since then, neural networks have been widely used on many tasks and have become the central tool in modern machine learning.
Artificial Neural Networks (ANNs) consist of interconnected *neurons*, each computing

$$y = \sigma\big(\mathbf{w}^\top \mathbf{x} + b\big), \tag{1.1}$$

where $\mathbf{x}$ is the input, $\mathbf{w}$ the weight vector, b a bias, and $\sigma(\cdot)$ a nonlinear activation (e.g. ReLU or sigmoid). Neurons are arranged in an input layer, one or more hidden layers, and an output layer, forming a feed-forward topology (see Fig. 1.4). When multiple hidden layers are stacked, we refer to the result as *deep learning*. [8]

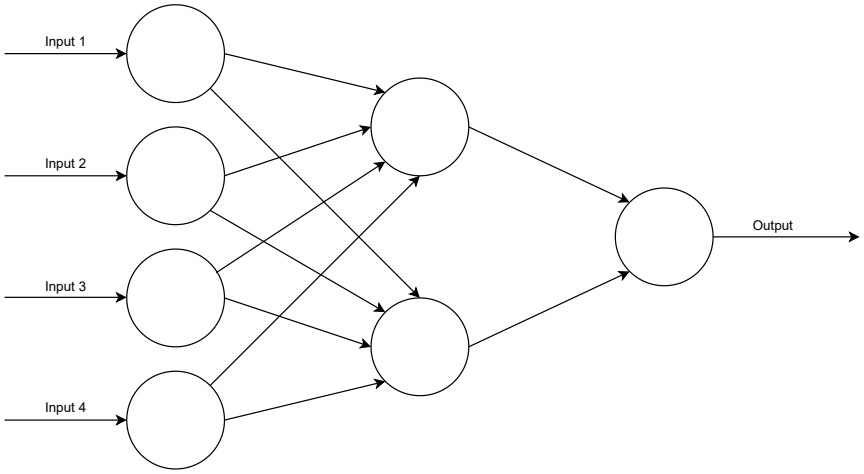


Figure 1.4: A simple three-layered feed-forward neural network (FNN) composed of an input layer, a hidden layer, and an output layer. This structure is the basis of several common ANN architectures.

Learning Paradigms

- **Supervised learning:** Training on labeled pairs ($\mathbf{x}$, $\mathbf{y}$), minimizing a loss function so predictions matching targets.

- **Unsupervised learning:** Learning from unlabeled data; models optimize internal criteria.

- **Self-supervised learning:** Learning useful representations by solving pretext tasks where labels are automatically generated from the data itself; supervision is derived from structure or augmentation of inputs.

Most image recognition tasks use supervised learning to train ANNs for classification or regression.

When training a neural network, the goal is to update its internal parameters—weights and biases—so that it improves at the task it is learning. After making a prediction, the model compares it with the true one using a loss function. Then, using backpropagation, it computes how much each weight and bias contributed to the error. These values are then corrected in the direction that reduces the loss, step by step, through gradient descent. Over time, this process should tune the network to perform better on the given task. [8]

1.2.2 Convolutional Neural Networks (CNNs)

CNNs were first introduced by LeCun et al. in the late 1980s and early 1990s, most notably with the LeNet [9] architecture for handwritten digit recognition. These models were inspired by the visual cortex and designed to exploit the spatial structure of images. However, widespread adoption only came later, with increased computational power, large labeled datasets, and GPU acceleration. A key turning point was AlexNet [10], which dramatically outperformed other methods in the ImageNet competition and brought CNNs to the spotlight of computer vision.

Motivation for Convolutional Networks

Fully connected ANNs scale poorly with image size. For a 28×28 grayscale image, a single hidden neuron requires $28 \times 28 = 784$ weights. For a $64 \times 64 \times 3$ color image, that jumps to $64 \times 64 \times 3 = 12{,}288$ weights per neuron—before adding more layers. Large networks become computationally expensive, memory-hungry, and prone to overfitting since many parameters must be estimated from limited data. [8]

Convolutional Networks

CNNs introduce two key ideas that exploit image structure:

- **Local receptive fields:** Each convolutional neuron connects only to a small patch (e.g., 3×3 or 5×5) of the previous layer, reducing parameters.
- **Weight sharing:** The same convolutional filter (kernel) is applied across all spatial locations, achieving translation invariance.

A convolutional layer with C filters transforms an $H \times W \times D$ input into an $H' \times W' \times C$ *feature map tensor*, where each channel encodes one learned pattern (see Fig. 1.5). Following convolutions, *pooling layers* (e.g., 2×2 max-pooling) downsample spatial dimensions, further reducing the computation and introducing robustness to small translations. [8]

Overfitting and Regularization

Deep networks with many parameters risk *overfitting*, learning noise instead of general features, or even learning entire parts of the training set. Strategies to mitigate overfitting include:

- **Architectural constraints:** Local connectivity and weight sharing in CNNs drastically limit parameter counts.
- **Pooling:** Reduces feature map size and filters out minor variations.
- **Dropout, weight decay, data augmentation:** common regularization techniques during training.

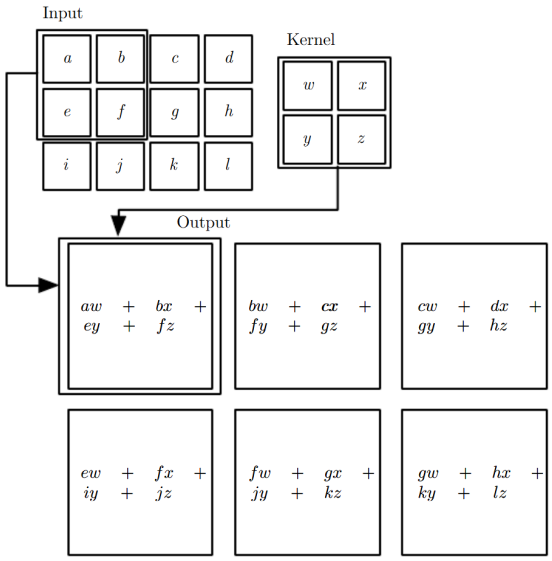


Figure 1.5: An example of 2-D convolution without kernel flipping. Boxes with arrows are drawn to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor. [11]

Overall CNN Architecture

A typical CNN for image classification or retrieval consists of

1. **Convolutional + activation:** Filters scan the input to detect local features, followed by nonlinearities (e.g., ReLU).
2. **Pooling**: Spatial downsampling (max- or average-pooling) to build invariance and shrink data size.
3. **(Repeated blocks)**: Stacking multiple conv–pool blocks to learn hierarchical representations.
4. **Fully connected or global pooling**: Collapse spatial dimensions to produce final descriptors or class scores.

Fig. 1.6 illustrates a simple five-layer CNN for MNIST digit classification. By combining local connectivity, shared weights, and pooling, CNNs achieve efficient and scalable feature learning that outperforms traditional ANNs on high-dimensional visual data. [8]

For further reading, [11] is suggested.

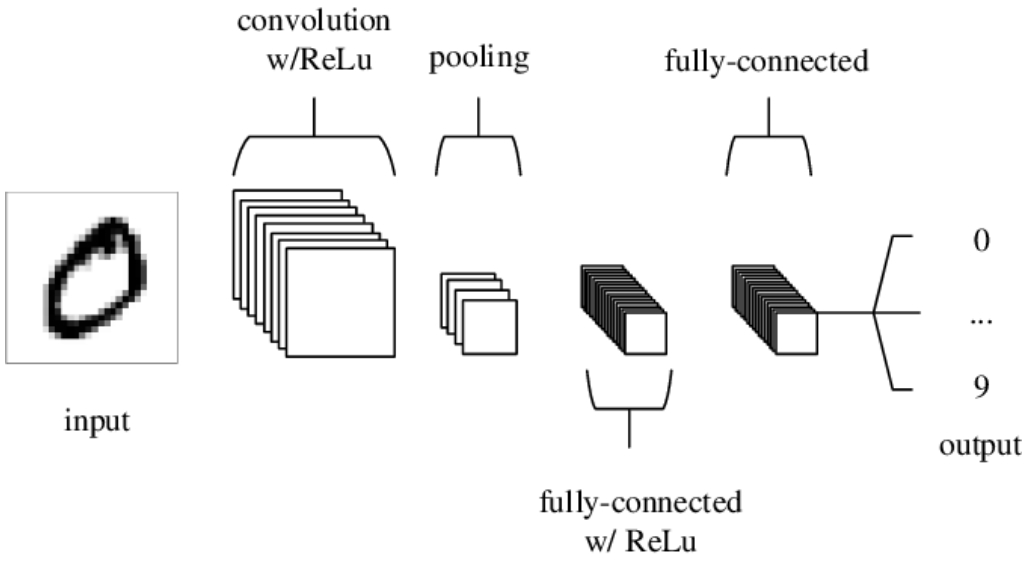


Figure 1.6: A simple CNN architecture composed of five layers. [8]

1.3 Brief introduction to Visual Transformers

Transformers were introduced in the seminal paper *"Attention Is All You Need"* by Vaswani et al. (2017) [12], in the context of Natural Language Processing (NLP). At the time, the dominant models were limited in their parallel computing power and long-term modeling. Transformers solved these limitations by relying entirely on a mechanism called **self-attention**, which allows the model to weigh the relevance of each element in the input sequence with respect to all others, regardless of distance or position. This change enabled both parallel computation and improved performance on long-range dependencies. For instance, in machine translation, a word at the end of a sentence could directly attend to the word at the beginning without being blocked by sequential steps.

Transformer Encoder-Decoder Architecture

Transformers use stacked encoder and decoder blocks, each composed of Multi-Head Self-Attention (MHSA) and feed-forward sublayers. Given an input sequence, the encoder applies six identical layers of MHSA where each token serves simultaneously as query, key, and value. It also applies a position-wise Multilayer Perceptron (MLP), allowing each token to attend globally across the sequence. The decoder mirrors this stack but adds a masked MHSA in each layer to prevent future token leakage, followed by cross-attention to encoder outputs. [13]

Basic Attention The core of the transformer is the **attention mechanism**, which allows each token to pay attention dynamically to all others in the input sequence. Given a token embedding, the model computes how relevant each other token is to it, assigning a higher weight to more informative ones.

To do so, the input sequence is linearly projected into three spaces:

- **Queries** $Q \in \mathbb{R}^{n \times d_k}$,
- **Keys** $K \in \mathbb{R}^{n \times d_k}$,
- **Values** $V \in \mathbb{R}^{n \times d_v}$,

where n is the sequence length. The attention output is then computed as a weighted sum over the values:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V. \tag{1.2}$$

The softmax function turns the similarity scores $QK^\top$ into probabilities, allowing the model to focus selectively on different tokens. The division by $\sqrt{d_k}$ scales the dot product, preventing large values from pushing the softmax into saturation, which would damage the learning phase.

Multi-Head Attention. Rather than computing a single attention function, the model runs multiple heads in parallel. Each head learns a different projection and attends to different parts of the sequence. The outputs of the h heads are concatenated and projected back:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O.$$

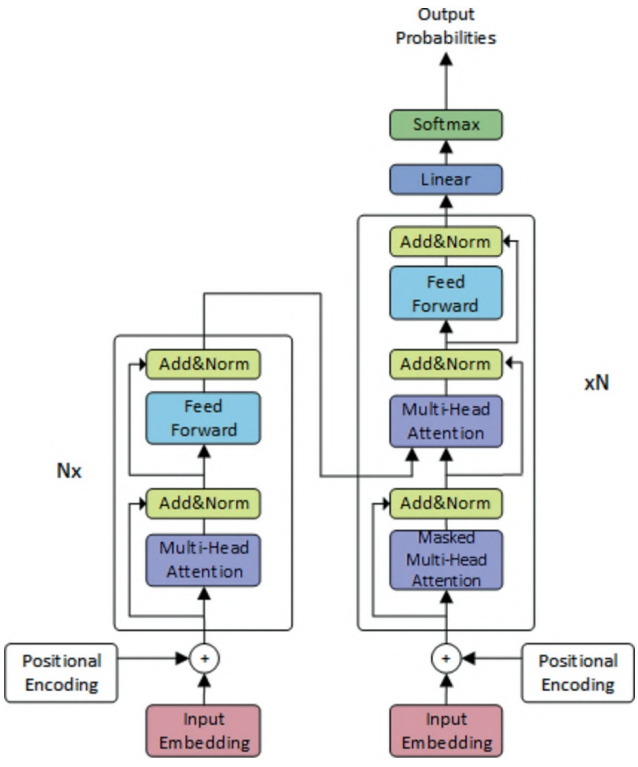


Figure 1.7: A transformer model structure. [13]

1.3.1 Visual Transformers

Despite their origin in NLP, given the amazing results they achieved, the transformer architecture began to spread to different domains of machine learning. In vision, early attempts like the 2018 Image Transformer [14] adapted attention to handling small image patches but did not yet rival convolutional approaches. Only in 2020 did the Vision Transformer (ViT) [15] demonstrate that pure self-attention could rival CNNs on large-scale image classification. A ViT divides each input image (e.g., 224×224) into non-overlapping patches (e.g., 16×16), flattens, and linearly projects them into a sequence of tokens. Then, it adds learnable positional embeddings and processes them through stacked transformer encoder blocks. By replacing convolutions with global self-attention, ViT captures long-range dependencies and outputs highly generalized feature representations. [13]

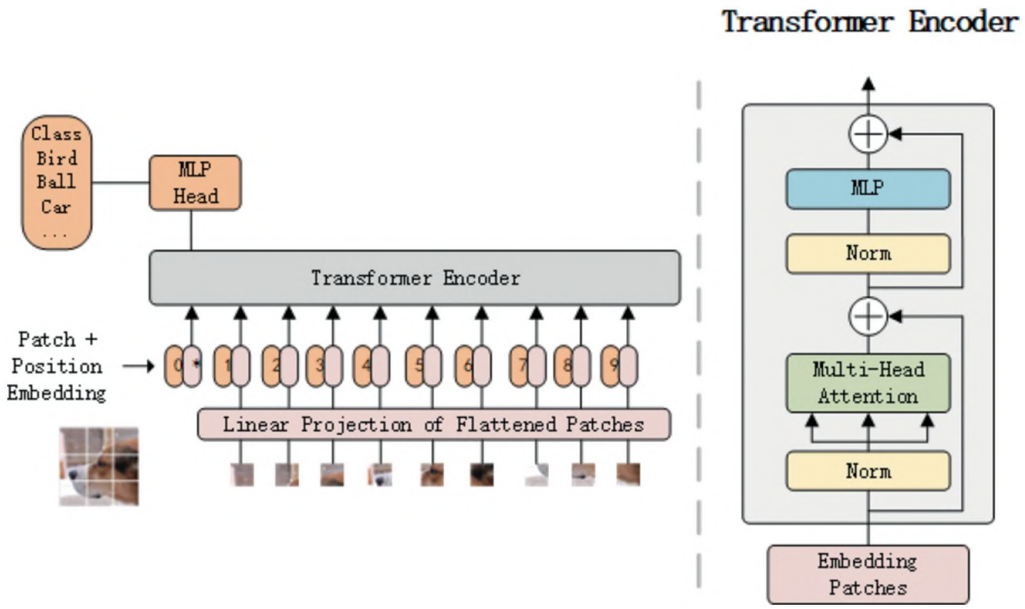


Figure 1.8: A Visual Transformer model structure: the image is fragmented into many patches, which are then fed to the attention-based transformer encoder. [13]

1.4 Introduction to Continual Learning

Continual learning (CL) is the ability of an artificial system to learn from data that comes as a stream over time and to adapt to new tasks or environments without forgetting past knowledge. In other words, a continual learner should be able to accumulate knowledge, apply it across different contexts, and update itself on the fly, much like biological systems do. [16]

While this sounds intuitive, current deep learning models are far from achieving it, as they do not have a memory for seen data. Unlike humans, who can adapt, retain, and store knowledge even in the face of changing environments, most neural networks forget almost everything they have learned when exposed to new data. This phenomenon is called *catastrophic forgetting*. This is largely due to how gradient descent works: optimizing for a new task typically means

overwriting the parameters learned from previous ones.
As soon as this issue was recognized, the community began developing methods to counter it. Over the years, CL has evolved into a research field of its own to design systems that can learn continuously, generalize across tasks, and selectively forget what is no longer useful. [16]

Requirements A continual learning system must be able to acquire new concepts over time, preserve relevant knowledge while discarding what is no longer relevant, and reuse (i.e., transfer) its experience across tasks.

Biological agents achieve this balance naturally—humans learn sequentially, adapt to sudden distribution shifts, and rarely suffer a catastrophic loss of past knowledge. Gradient-based neural networks, in contrast, are highly *plastic*: when optimized on new data, they tend to overwrite earlier memories. Addressing this interference has driven much of the recent progress in continual (a.k.a. lifelong or incremental) learning.

An autonomous robot, an always-on speech assistant, or any embodied agent deployed in the wild cannot afford to retrain from scratch each time its environment changes. Instead, it must learn on the fly under strict resource budgets—limited labels, bounded memory, finite compute power. In short, it must resolve the classic *stability–plasticity dilemma*: remain plastic enough to absorb novelty, yet stable enough to protect prior skills. [16]

Practical requirements A realistic continual learning module, therefore, should:

- **Learn with few (or zero) labels:** Mirror the curiosity and self-supervision of biological learners.
- **Operate without task boundaries:** Infer shifts autonomously rather than rely on external signals.
- **Handle unforeseen situations:** Generalize past the set of conditions seen during development.
- **Ingest streaming sensory data:** Update online instead of in large, pre-shuffled batches.
- **Respect tight resource budgets:** Function under fixed memory, energy, and compute constraints.
- **Forget well:** Forget outdated or misleading information in a controlled manner—an ability mostly ignored in current literature, yet commonplace in humans.

1.5 Introduction to Visual Place Recognition

Visual Place Recognition (VPR) refers to the task of identifying the geographic or semantic location depicted in an image (or sequence of images). Often formulated as an image retrieval problem, VPR relies on a database of reference images, each annotated with a location identifier (e.g., landmark name or GPS coordinate) [17]. Given a query image to localize, the system performs an encoding step, where each image (database and query) is transformed into

a fixed-length feature vector via descriptors or deep networks; a similarity search, which compares the query vector against all database vectors using a distance metric (usually Euclidean or cosine) to retrieve top matches; and then a post-processing operation, refining of the initial matches through geometric verification, spatial filtering, or learning-based re-ranking. This step is optional and not always implemented.

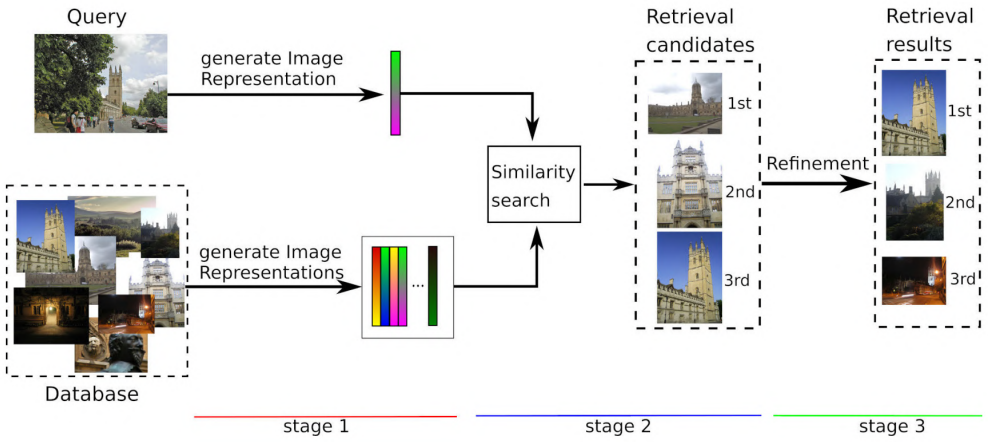


Figure 1.9: Three-stage VPR pipeline: (1) feature encoding computed offline for database and online for query, (2) similarity search retrieves top-K matches, (3) (optional) post-processing refines results. [17]

Related Fields

While VPR has been typically framed as an image retrieval problem for SLAM, it also intersects with several other areas, each offering mutual insights:

- **Image Retrieval:** Like general image retrieval, VPR involves matching visual content across databases. However, the goal in VPR is not just to find semantically similar images but to locate the exact place, often under varying conditions like day vs. night. This makes VPR more constrained and sensitive to perceptual aliasing, where different places may look alike. [18]
- **Video Retrieval:** Sequence-based VPR methods relate closely to video retrieval, where individual frame matches are aggregated across time. New approaches now represent entire sequences directly, improving robustness to appearance changes and temporal variation. [19]
- **Landmark Recognition:** Landmark recognition focuses on classifying images into specific, named landmarks. VPR, by contrast, targets ordinary, potentially unnamed locations. Despite this, advances in landmark retrieval—such as descriptor learning and large-scale matching—are proving useful for VPR as well. [20]
- **Overlap Detection:** VPR assumes partial visual overlap between query and database images. This links it to the task of visual overlap detection, which has led to new

ground truth definitions and evaluation strategies. Overlap-based metrics (e.g., normalized surface overlap) can even be used as supervision signals when GPS is unavailable. [21]

1.5.1 Describing Places

Before deep learning, VPR relied on hand-crafted image descriptors, specifically designed algorithms that extract informative patterns from the image. These features usually capture edges, textures, corners, or other local structures that a human would consider relevant for visual understanding.

They typically fall into two main categories, as shown in Fig. 1.12:

- **Global descriptors:** The image is encoded as a whole.
- **Local descriptors:** Features are extracted from multiple keypoints or regions and aggregated into a single representation.

Examples of hand-crafted features: Two classic examples of hand-crafted feature extractors are **Histogram of Oriented Gradients (HOG)[22]** and **Scale-Invariant Feature Transform (SIFT)[23]**. They both produce descriptors of an image, but while HOG focuses on capturing local edge directions (see Fig. 1.10), SIFT detects and describes distinctive keypoints, remaining robust to scale and rotation (see Fig. 1.11).

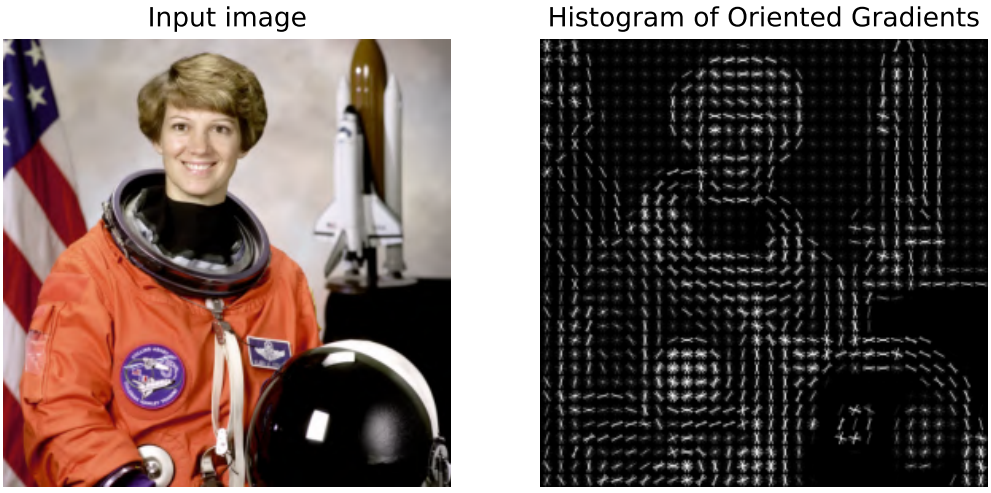


Figure 1.10: HOG encodes local gradients in fixed grids.

Local Descriptor Aggregation

Local descriptors are extracted around salient image patches or keypoints. Each patch is described using local texture and gradient information—SIFT being a classic example [24]. To a scale, costly descriptor-to-descriptor matching is avoided, aggregation strategies group these features into compact global vectors, allowing fast image retrieval via vector similarity.

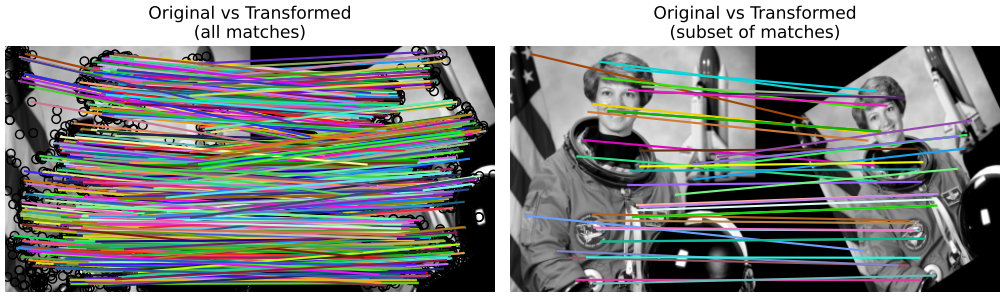


Figure 1.11: SIFT detects and describes scale and rotation-invariant keypoints.

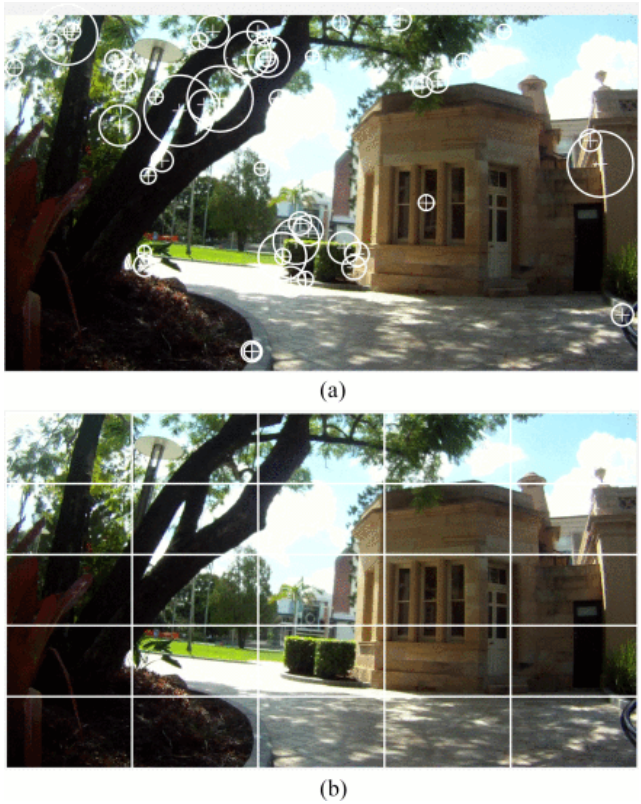


Figure 1.12: Local vs. global descriptors: (a) salient keypoint detection and description; (b) grid-based global encoding.

Visual Vocabulary Construction The first step is to generate a codebook by clustering descriptor samples (e.g., k-means) into visual words, and then each descriptor is assigned to its nearest word to form sparse histograms (Bag-of-Words); finally, weighting and retrieval are achieved by applying TF–IDF or binary weighting and computing cosine similarity. With inverted indices, the time required is sub-linear.
These methods follow a two-step paradigm:

1. **Embedding step:** Each local descriptor is mapped into a higher-dimensional space to enhance distinctiveness and suppress false positives;

2. **Aggregation step:** The embedded vectors are pooled into one fixed-length image descriptor.

For example, the Vector of Locally Aggregated Descriptors (VLAD) [25] embedding discards matches between features assigned to different centroids in the codebook.

Global Scene Descriptors

Global descriptors (e.g., HOG) encode the whole image rapidly. They are lower-cost and illumination-invariant but less robust to viewpoint and occlusion. These hand-crafted techniques laid the foundation for convolutional representations.

1.5.2 Deep Learned Representations

Convolutional Neural Networks (CNNs) are specialized for processing grid-like data such as images. Since the breakthrough of deep CNNs in visual tasks [26], they have been shown to produce transferable image representations [27]. In image retrieval, NN-derived features now outperform hand-crafted methods.

Fully Connected Representations

Early works [27] used activations from a network's final fully connected (FC) layer, pre-trained on ImageNet as global image descriptors. Training specifically for retrieval with triplet loss further improved performance [28]. However, FC networks lack robustness to occlusions and translation, require fixed input sizes and many parameters, and remain computationally heavy when extracted from sub-patches.

Convolutional Representations

Rather than using FC outputs, convolutional feature maps form an H×W×C tensor capturing local patterns. Naively flattening these maps does not fully leverage spatial structure, so two main strategies emerged:

Aggregated Representations Treat the H×W grid of C-dimensional descriptors like dense local features. Apply embedding and pooling inspired by hand-crafted methods:

- **VLAD/Fisher Vectors/ASMK (Aggregated Selective Match Kernels):** Embed each descriptor then aggregate into a single vector [25, 29, 30];

- **NetVLAD:** Differentiable VLAD layer for end-to-end learning [18];
- **CNN+Fisher:** Joint training of CNN and Fisher Vector module [31].

Pooled Representations Summarize convolutional maps with simple pooling:

- **MAC (Maximum Activations of Convolutions):** Max-pool each feature map [32];
- **SPoC (Sum-Pooled Convolutional features):** Sum-pool then whiten descriptors [33];
- **R-MAC (Regional Maximum Activations of Convolutions):** Aggregate regional max-pooled vectors over multiple scales [34];
- **GeM (Generalized Mean Pooling):** Parametric generalized-mean pooling, learnable and subsumes MAC/SPoC [35].

These CNN-based descriptors use the CNN as a backbone to extract features and then combine them with pooling or aggregation that preserves spatial information.

Visual Transformer Backbones

While CNNs remain the dominant architecture for visual representation learning, transformer-based models have recently emerged as a strong alternative. Visual Transformers (ViTs) [15] discard convolutions in favor of global self-attention, modeling long-range dependencies across image patches. Instead of scanning local neighborhoods, ViTs split an image into fixed-size patches, flatten and embed each, and process them as a sequence using transformer layers. This global context allows ViTs to capture semantic structure better and as a whole compared to CNNs. Pretrained ViTs yield powerful features that transfer well to downstream tasks, including retrieval [36, 37]. Their representations are spatially aware, flexible in input resolution, and tailored for token-level aggregation. Although computationally more demanding, ViT-based backbones are now outperforming CNNs on most image retrieval benchmarks. When used as a backbone, paired with an aggregator and a re-ranking algorithm, these models achieve incredible results on VPR tasks, such as the 100% recall rate on Tokyo 24/7. [38]

Chapter 2

Methodology

Problem Formulation

Visual Place Recognition can be cast as an image retrieval task. Let

$$\mathrm{DB} = \{I_i\}_{i=1}^{N}, \quad \mathrm{Q} = \{I_j\}_{j=1}^{M}$$

be the database of reference images and the query set, respectively. The goal is to find all pairs (i, j) such that $I_i \in \mathrm{DB}$ and $I_j \in Q$ depict the same place. For each image I a descriptor $\mathbf{f} = \phi(I) \in \mathbb{R}^D$ is extracted, where ϕ is a learned or hand-crafted feature extractor: descriptors of the same place lie close in $\mathbb{R}^D$, while those of different places lie far apart. The similarity matrix, which indicates the similarity between any two descriptors, is $S \in \mathbb{R}^{N \times M}$ with entries

$$s_{ij} = \mathrm{sim}\big(\phi(I_i),\, \phi(I_j)\big)$$

where sim may be cosine similarity or negative Euclidean distance. The retrieval step for each query j, ranks all database images i by descending s_{ij} and selects the top-K; i.e. the predicted matches are

$$\{(i, j) \mid I_i \in \mathrm{DB},\ I_j \in Q,\ i \in \mathrm{argtop}^{K}_{i' \in \mathrm{DB}}\, s_{i'j}\}.$$

VPR performance is classically evaluated by how accurately these top-K lists recover the true matching pairs. [39]

Feature extractors

As specified above, the feature extractors may be hand-crafted or learned. This thesis focuses on learned feature extractors; therefore, the following section will cover this type. Learned feature extractors are obtained by two main components: backbones and aggregators. The backbone component is the one that takes care of extracting the important features from the image, while the aggregator selects and aggregates the extracted features for the task.

2.1 Backbones

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have long been the preferred backbones for VPR tasks. By stacking convolutional filters and pooling operations, CNNs gradually build hierarchical feature maps that capture edges, textures, and increasingly more detailed scene patterns. Off-the-shelf architectures such as VGG and ResNet serve as feature extractors: the usual procedure is to remove the final classifier, pool the last convolutional activations (or use instead the penultimate fully connected layers), and use the resulting vector as the image descriptor $\phi(I)$. These descriptors, when compared via cosine similarity or learned embeddings, return strong place matches even under moderate viewpoint and illumination changes.

VGG

VGG [40] stands for Visual Geometry Group, and it is a CNN that uses a uniform stack of small 3×3 convolutions layers, each followed by ReLU activations and periodic 2×2 max-pooling to halve spatial resolution. After five convolutional blocks, three large, fully connected layers collapse the feature maps into a 4096-dimensional vector (see Fig. 2.1). VGG models are usually composed of either 16 or 19 layers. The model ends with a series of fully connected layers followed by a softmax for classification. The number of layers is capped at 19 because VGG and similar architectures share a significant issue: the weights of a neural network are updated through the backpropagation algorithm, which makes a minor change to each weight so that the loss of the model decreases. It does so by updating each weight so that it takes a step in the direction along which the loss decreases, according to the gradient of the weight, which can be found using the chain rule. However, as the gradient keeps flowing backward to the initial layers, the value of the gradient keeps becoming smaller and smaller, sometimes even vanishing. This problem is referred to as the *vanishing gradient*.

ResNet

ResNet [41] introduced identity-skip connections to ease the training of very deep nets to solve the *vanishing gradient* problem. A residual block computes

$$\mathbf{y} = \sigma\big(\mathbf{x} + \mathcal{F}(\mathbf{x})\big), \quad \mathcal{F}(\mathbf{x}) = W_2\,\sigma\big(W_1\,\mathbf{x} + b_1\big) + b_2,$$

where W_1, W_2 are 1×1 and 3×3 (or vice versa) convolutions, $\mathbf{x}$ is the input and σ is the activation function, usually a ReLU (see Fig. 2.2). Stacking these blocks yields networks of 50+ layers that learn features at multiple scales. For VPR, one uses the output of the final global-pooled layer (e.g., a 2048-D vector) as the image descriptor $\phi(I)$. The residual design improves gradient flow and yields more discriminative embeddings than plain CNNs.

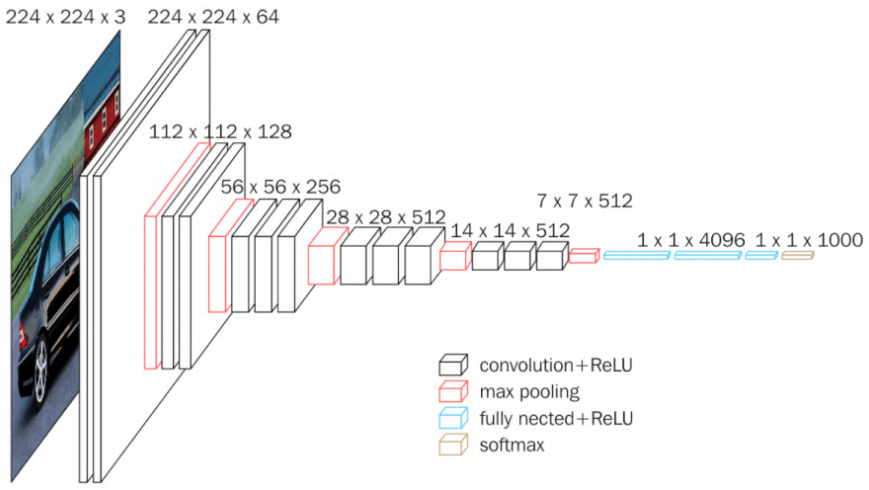


Figure 2.1: VGG classic model structure for classification: a stack of a series of convolutions followed by max pooling. The model ends with a series of fully connected layers followed by a softmax for classification. In a VPR task, we would remove the last softmax layer.

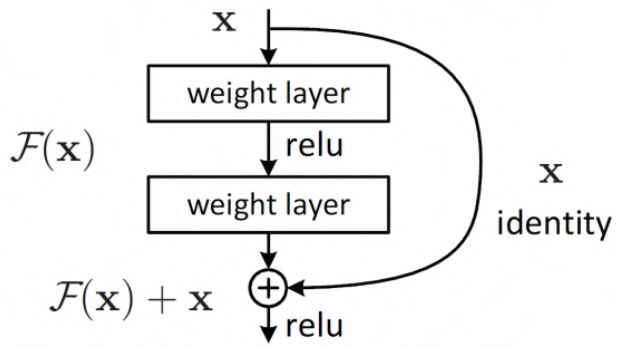


Figure 2.2: A classic ResNet block: the output of the block is a ReLU of both the skip connection (the input before being passed to the function) and the elaborated input by the intermediate layers.

2.1.2 Visual Transformers

More recently, Vision Transformers (ViTs) have emerged as powerful alternatives to CNNs as backbones. ViTs break images into patch tokens, add positional embeddings, and apply global self-attention to model global relationships across the entire scene, rather than only within local neighborhoods. The models are trained either with supervised labels or self-supervised objectives (e.g., DINOv2), and they produce descriptors that excel at capturing long-range spatial patterns, making them particularly effective in challenging VPR scenarios such as significant viewpoint shifts or dynamic environments. Most ViT models can be used as *foundation models*[42]: as foundation models, ViTs serve as the universal backbone, and they can be plugged in as task-specific heads or with or without fine-tuning to achieve state-of-the-art results with minimal labeled data. In practice, a ViT backbone yields a fixed-length class-token embedding, which, like a CNN's pooled feature, can be directly used for retrieval or handled by the aggregator.

DINO and DINOv2

Until 2020, Visual Transformers barely reached the same performance as CNNs while still being computationally more demanding, requiring more training data, and without any unique properties striking out from their features [36]. The breakthrough was achieved by switching from supervised to self-supervised learning: the attention mechanism was able to focus by itself on the most important part of the images, even in a complex environment (see Fig. 2.3). The first model to take this approach was the DINO model (**DI**stillation with **NO** labels).

Self-Supervised Learning via Knowledge Distillation

DINO frames self-supervision as a form of online knowledge distillation between a *student* network g_{θ_s} and a *teacher* g_{θ_t} (see Fig 2.4), where the teacher's weights θ_t are an exponential moving average (EMA) of the student's:

$$\theta_t \leftarrow \lambda\,\theta_t + (1-\lambda)\,\theta_s, \quad \lambda \in [0.996, 1]. \tag{2.1}$$

Given an image x, two *global* crops x_1^g, x_2^g and several smaller *local* crops x^ℓ are generated. The student and teacher each produce a *class token* embedding, projected by separate MultiLayer Perceptron (MLP) heads into K prototype logits. After softmax with temperatures τ_s, τ_t, distributions P_s and P_t are obtained by centering on the teacher outputs. The core loss aligns every student view x' to each teacher global view x:

$$\min_{\theta_s} \sum_{x \in \{x_1^g, x_2^g\}} \sum_{x' \neq x} H\big(P_t(x),\, P_s(x')\big), \tag{2.2}$$

where $H(a,b) = -\sum a \log b$. No negative pairs, queues, or predictors are required—DINO's momentum teacher and multi-crop cross-entropy are enough to yield rich and invariant ViT features for many different tasks, including VPR.

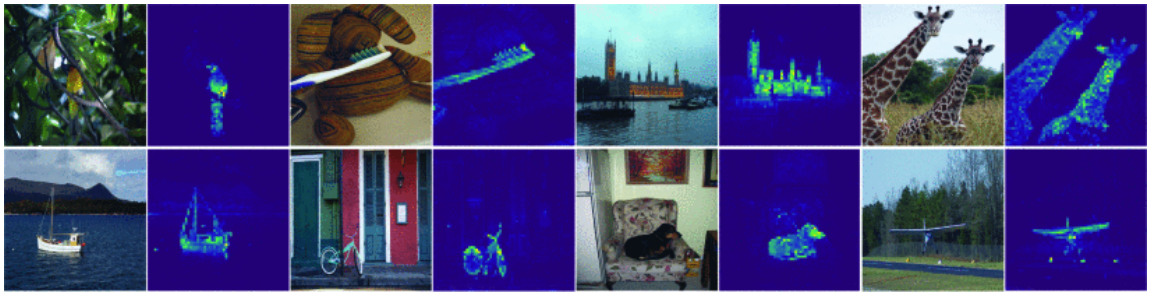


Figure 2.3: Self-attention from a Vision Transformer with 8 × 8 patches trained with no supervision. The self-attention of the [CLS] (starter) token on the heads of the last layer is highlighted. This token is not attached to any label or supervision. These maps show that the model automatically learns class-specific features, leading to unsupervised object segmentations. [36]

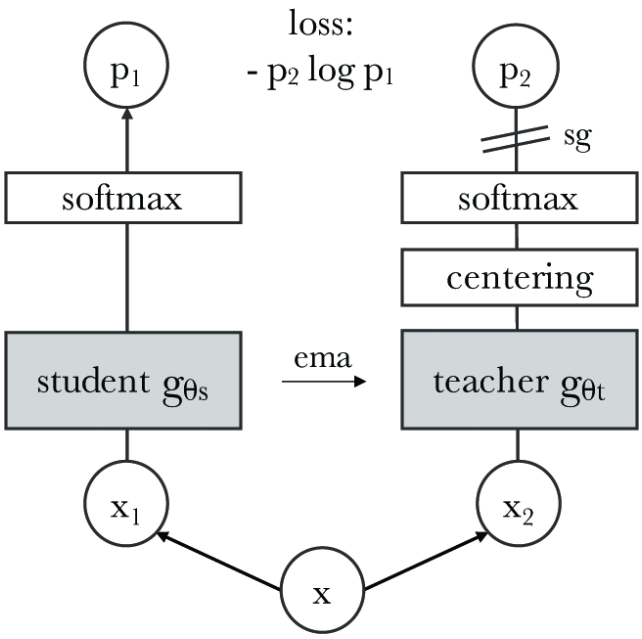


Figure 2.4: The model passes two different random transformations of an input image to the student and teacher networks. Both networks have the same architecture but different parameters. The output of the teacher network is centered with a mean computed over the batch. Each network outputs a K-dimensional feature that is normalized with a temperature softmax over the feature dimension. A stop-gradient (sg) operator is applied to the teacher to propagate gradients only through the student. [36]

DINOv2 improvements

DINOv2 [37] builds on the original DINO framework by vastly improving data efficiency and feature quality through a stronger ViT backbone and enhanced self-supervision. Instead of relying solely on the dual-crop image-level objective of DINO, DINOv2 combines both image and patch-level losses—specifically, the DINO and iBOT [43] objectives—while also maintaining the Sinkhorn-Knopp [44] centering scheme and incorporating a KoLeo [45] regularizer for improved feature distribution.

The image-level objective compares features extracted from different views (crops) of the same image via a student–teacher setup. Both networks process the image using their own ViT backbone and produce class tokens. The student output is passed through an MLP head and converted into a probability vector $p_s \in \mathbb{R}^K$ via softmax. The teacher does the same, and its output is normalized using centering with Sinkhorn-Knopp to obtain p_t. The DINO loss is then defined as a cross-entropy between these two distributions:

$$\mathcal{L}_{\text{DINO}} = -\sum_{k=1}^{K} (p_t)_k \log(p_s)_k \tag{2.3}$$

Simultaneously, DINOv2 applies a patch-level objective inspired by iBOT. Some patches are randomly masked in the student's input while the teacher sees the full image. The class probabilities $p_{s,i}, p_{t,i} \in \mathbb{R}^{K'}$ for patch position i are obtained via softmax from the corresponding tokens, and the loss compares each unmasked patch using the same cross-entropy formulation:

$$\mathcal{L}_{\text{iBOT}} = -\sum_{i=1}^{M} \sum_{k=1}^{K'} (p_{t,i})_k \log(p_{s,i})_k$$

where M is the number of visible (unmasked) patches and K' is the number of patch-level prototypes.

At scale, DINOv2 uses separate MLP heads for the image- and patch-level tasks, as untied heads yield better performance than sharing weights.

To further enhance representation uniformity, DINOv2 introduces the KoLeo regularizer based on the Kozachenko–Leonenko differential entropy estimator. This term encourages feature vectors to be well-distributed in the embedding space. Given n ℓ_2-normalized feature vectors $(x_1, \ldots, x_n)$, the loss is defined as:

$$\mathcal{L}_{\text{KoLeo}} = -\frac{1}{n} \sum_{i=1}^{n} \log(d_{n,i}), \quad \text{where } d_{n,i} = \min_{j \neq i} \|x_i - x_j\|_2 \tag{2.4}$$

where $d_{n,i}$ is the distance between x_i and its closest neighbor among the other vectors.

Finally, to improve fine detail capture—essential for pixel-level downstream tasks like segmentation—DINOv2 performs a short fine-tuning phase at higher resolution (518×518). This resolution upscale applied only at the end of training helps recover small object features without the heavy computational burden of training entirely at high resolution.

These improvements result in descriptors that are more discriminative and robust, particularly in challenging scenarios involving viewpoint shifts or appearance changes. [37]

SWIN

Swin Transformer[46]—short for "Shifted Window" Transformer—is a hierarchical Vision Transformer designed for efficiency and locality. It first embeds the image into non-overlapping patches and then processes them through stages that progressively reduce spatial resolution, just like a CNN's feature pyramid. Within each stage, self-attention is computed only inside fixed-size windows (e.g., 7×7), dramatically cutting the quadratic cost of global attention. To allow information to flow across window boundaries, consecutive layers "shift" the window grid by a fixed offset so that each token eventually attends to neighbors in adjacent windows (see Fig. 2.5). For VPR, features from the last stage (e.g., a 1024-D vector) are used directly or further aggregated to form $\phi(I)$.

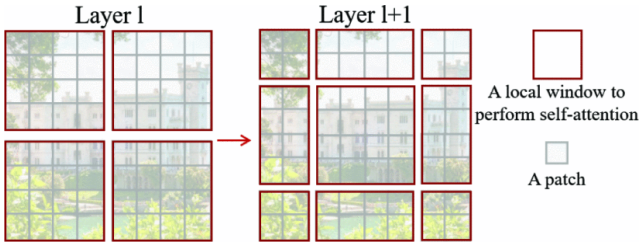


Figure 2.5: An illustration of the shifted window approach for computing self-attention in the proposed Swin Transformer architecture. In layer l (left), a regular window partitioning scheme is adopted, and self-attention is computed within each window. In the next layer, l + 1 (right), the window partitioning is shifted, resulting in new windows. The self-attention computation in the new windows crosses the boundaries of the previous windows in layer l, providing connections among them. [46]

2.2 Aggregators

Image Vector Representations

Image-level representations can be built by aggregating local descriptors into a fixed-size vector. The prototypes of aggregators- such as BoF, VLAD, and Fisher Kernel -were static and unable to "learn" anything from the data. The state of the art now requires instead a few final layers on the Neural Network to learn the correct parameters for the aggregation step, as in GeM and NetVLAD.

Bag of Features (BoF) Local descriptors are clustered into k visual words using k-means. Each descriptor is assigned to its nearest centroid, and an image is represented as a k-dimensional histogram counting assignment. Normalization (typically ℓ_2) and IDF reweighting improve discriminability. Variants such as soft assignment further improve the quantization step.

Fisher Kernel Fisher Vectors generalize BoF by modeling the distribution of descriptors with a Gaussian Mixture Model (GMM). An image is encoded by the gradient of the log-

likelihood of its descriptors with respect to the GMM parameters. FVs capture higher-order statistics and often outperform BoFs with fewer visual words. [29]

VLAD VLAD (Vector of Locally Aggregated Descriptors) bridges BoF and Fisher by retaining local residuals while discarding probabilistic modeling. After computing a k-means codebook $c_1, \ldots, c_k$, each descriptor x is assigned to its nearest centroid c_i=NN(x). VLAD accumulates residuals $x - c_i$ for each c_i:

$$v_{i,j} = \sum_{x \mid NN(x)=c_i} x_j - c_{i,j}$$

The final vector v is ℓ_2-normalized. Despite its simplicity, VLAD yields strong performance even with small k ($16 \leq k \leq 256$). [25]

2.2.1 Generalized-Mean Pooling (GeM)

Fully convolutional CNNs are used by discarding the fully connected layers from standard architectures such as ResNet or VGG. Given an input image, the network produces a 3D tensor $X_c \in \mathbb{R}^{W \times H \times C}$, where C is the number of channels in the final convolutional layer. Each feature map X_c is a spatial activation map of size $W \times H$, for $c \in \{1...C\}$. [35]

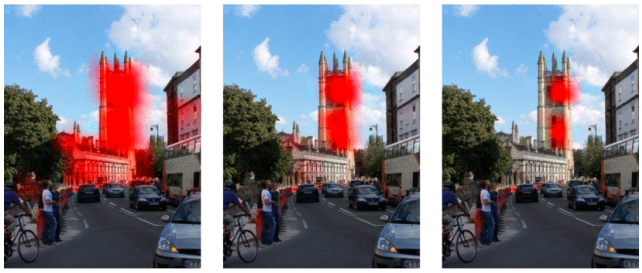
GeM pooling. A global pooling layer aggregates each X_c into a scalar using generalized-mean pooling:

$$\mathbf{f}^{(g)} = [\mathrm{f}_1^{(g)} \ldots \mathrm{f}_c^{(g)} \ldots \mathrm{f}_C^{(g)}]^\top, \quad \mathrm{f}_c^{(g)} = \left(\frac{1}{|X_c|} \sum_{x \in X_c} x^{p_c} \right)^{\frac{1}{p_c}}.$$

The resulting descriptor is $\mathbf{f}$ and is subsequently ℓ_2-normalized. Max pooling and average pooling are recovered as edge cases for $p_c \to \infty$ and $p_c = 1$, respectively. The pooling exponent p_c can be fixed or optimized during training. Either a shared or per-channel configuration is supported.

Larger values of p_c produce sharper, more localized responses, enhancing spatial selectivity and descriptor discriminability (see Fig. 2.6).

Whitening. A discriminative whitening step is optionally applied after training to improve descriptor separability. In this step, PCA is applied to a transformation of the feature space based on intra-class and inter-class statistics, resulting in a linear projection matrix. Descriptors are centered, projected, and re-normalized. This whitening is performed offline as a post-processing operation. [35]



$p = 1$ $p = 3$ $p = 10$

Figure 2.6: Visualization of X^{p_c} projected on the original image for three different values of p. Case $p = 1$ corresponds to SPoC, and larger p corresponds to GeM before the summation. Examples shown use the off-the-shelf VGG. [35]

2.2.2 NetVLAD: Differentiable VLAD Pooling

Standard image–retrieval pipelines first extract local descriptors and then aggregate them in an order-less fashion, providing robustness to translation, partial occlusion, and viewpoint changes. NetVLAD embeds this two-step process in a fully differentiable CNN layer, allowing for end-to-end training. [18]

Local-descriptor stage. The backbone CNN is truncated at the last convolutional layer. For an input image, the truncated network outputs a tensor $X \in \mathbb{R}^{H\times W\times C}$, which is interpreted as $N = H \times W$ local descriptors $\{x_i\}_{i=1}^{N}$ with dimensionality C.

VLAD aggregation. Classic VLAD pools these descriptors around a codebook $\{c_k\}_{k=1}^{K}$ ("visual words"). With hard assignment, the (k, c) element of the VLAD matrix $V \in \mathbb{R}^{K\times C}$ is

$$V_{k,c} = \sum_{i=1}^{N} a_k(x_i)\left(x_i^{(c)} - c_k^{(c)}\right), \quad a_k(x_i) = \begin{cases} 1 & \text{if } k = \arg\min_{k'} \|x_i - c_{k'}\|_2 \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

where $x_i^{(c)}$ and $c_k^{(c)}$ denote the c-th component of descriptor x_i and cluster center c_k, respectively. The matrix is intra-normalized and then ℓ_2-normalized after vectorization.

Differentiable soft assignment. The hard assignment in Eq. 2.5 contains non-differentiable `argmin` operations, making it unsuitable for end-to-end training via backpropagation. NetVLAD addresses this by replacing it with a learnable soft assignment.

The starting point is to assign descriptors to clusters based on their proximity, weighted by a parameter α that controls the assignment softness:

$$\tilde{a}_k(x_i) = \frac{\exp(-\alpha\|x_i - c_k\|_2^2)}{\sum_{k'=1}^{K}\exp(-\alpha\|x_i - c_{k'}\|_2^2)}.$$

As $\alpha \to \infty$, this formulation recovers the original hard assignment. By expanding the squared norm $\|x_i - c_k\|_2^2 = \|x_i\|_2^2 - 2c_k^\top x_i + \|c_k\|_2^2$, the term $\exp(-\alpha\|x_i\|_2^2)$ cancels from the numerator

and denominator. This yields a simplified and more general form:

$$\tilde{a}_k(x_i) = \frac{\exp(w_k^\top x_i + b_k)}{\sum_{k'=1}^{K} \exp(w_{k'}^\top x_i + b_{k'})}, \tag{2.6}$$

where the new parameters are related to the old ones by $w_k = 2\alpha c_k$ and $b_k = -\alpha \|c_k\|_2^2$. This soft-assignment mechanism can be efficiently implemented as a 1×1 convolution (with weights w_k and biases b_k) applied to the local descriptors, followed by a softmax activation across the cluster dimension. [18]

NetVLAD layer and learnable parameters. Substituting the soft assignment $\tilde{a}_k(x_i)$ from Eq. 2.6 into the VLAD formulation yields the final *NetVLAD* pooling layer:

$$V_{k,c} = \sum_{i=1}^{N} \tilde{a}_k(x_i) \left(x_i^{(c)} - c_k^{(c)}\right), \qquad \forall\, k \in [1, K],\ c \in [1, C]. \tag{2.7}$$

NetVLAD decouples the parameters, treating $\{w_k, b_k, c_k\}$ as three independent sets that are optimized jointly with the backbone network. This allows for greater flexibility and differentiability than the original VLAD, where the assignments are strictly tied to the cluster centers c_k and are not updatable with the network.

Normalization and output. The resulting $(K \times C)$ NetVLAD matrix is subjected to two sequential normalization steps. First, it is intra-normalized by applying ℓ_2-normalization to each of its K vectors representing a visual word. Second, the entire matrix is flattened into a single vector and ℓ_2-normalized again to produce the final, compact image descriptor ready for similarity search. The resulting operation can be seen in Fig. 2.7, as the clusters move instead of staying fixed.

Implementation. The NetVLAD equation can be expressed as a small directed acyclic graph (DAG) of standard CNN layers: a 1×1 convolution (w_k, b_k), a channel-wise softmax, a residual computation $(x_i - c_k)$, a weighted sum, and two normalization steps (Fig. 2.8). The resulting layer faithfully mimics VLAD while remaining fully differentiable and trainable end-to-end.

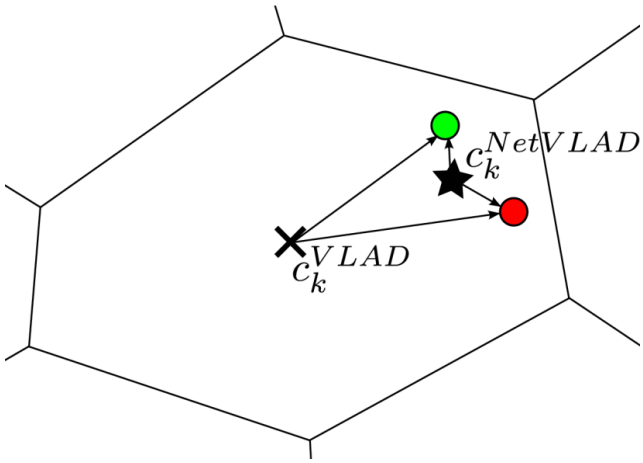


Figure 2.7: **Benefits of supervised VLAD.** Red and green circles are local descriptors from two different images assigned to the same cluster. Under the VLAD encoding, their contribution to the similarity score between the two images is the scalar product (as final VLAD vectors are ℓ_2-normalized) between the corresponding residuals, where a residual vector is computed as the difference between the descriptor and the cluster's anchor point. The anchor point c_k can be interpreted as the origin of a new coordinate system local to the specific cluster k. In standard VLAD, the anchor is chosen as the cluster center ($\times$) in order to evenly distribute the residuals across the database. However, in a supervised setting where the two descriptors are known to belong to images that should not match, it is possible to learn a better anchor ($\star$), which causes the scalar product between the new residuals to be small. [18]

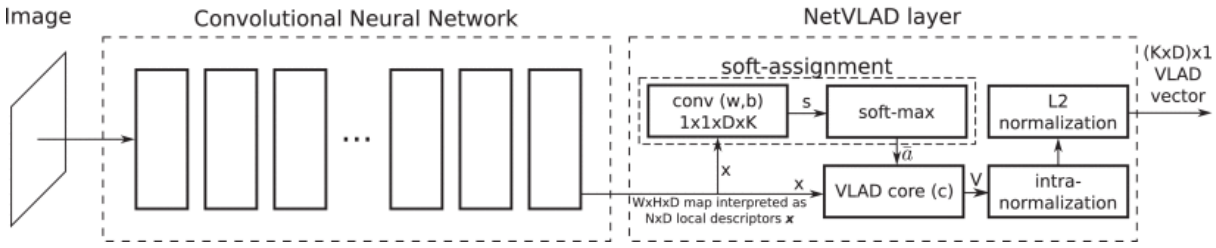


Figure 2.8: **CNN architecture with the NetVLAD layer.** The layer can be implemented using standard CNN layers (convolutions, softmax, ℓ_2-normalization) and one easy-to-implement aggregation layer to perform aggregation in equation (2.7) ("VLAD core"), joined up in a directed acyclic graph. Parameters are shown in brackets. [18]

2.3 Continual Learning

In Visual Place Recognition, environments are rarely static—seasons change, objects move, and lighting shifts. Systems deployed in the real world must deal with this evolving visual landscape. In some cases, training a model and hoping it will generalize enough to work on all the situations is not enough. That is where continual learning comes in. The idea is simple: allow the model to keep learning over time, adapting to new situations without forgetting what it already knows.

2.3.1 AirLoop

AirLoop [47] is a model proposed in 2022, and it addresses the challenge of continual loop closure detection by allowing deep models to learn from new environments while preserving previously acquired knowledge. Traditional CNN-based methods perform well when trained offline on big datasets, but do not generalize to novel environments over time when applied to dynamic environments. Simply fine-tuning the model on new data leads to catastrophic forgetting, while retraining from scratch on all past data may be infeasible, especially on power-constrained devices.

To mitigate this, AirLoop integrates a lifelong learning framework specifically designed for visual SLAM. The method relies on two key components:

- **Relational Memory Aware Synapses (RMAS):** an extension of MAS that preserves not just parameter values but their impact on descriptor similarity, ensuring that the structure of the learned feature space is retained over time. The importance Ω_i of a parameter θ_i is estimated from past tasks by:

$$\Omega_i = \mathbb{E}_{(I_1,I_2)}\left[\left(\frac{\partial \text{sim}(\phi(I_1),\phi(I_2))}{\partial \theta_i}\right)^2\right], \tag{2.8}$$

  therefore, the RMAS penalty becomes:

$$\mathcal{L}_{\text{RMAS}} = \sum_i \Omega_i(\theta_i - \theta_i^*)^2, \tag{2.9}$$

  where θ^* are the parameter values after learning the previous task.

- **Relational Knowledge Distillation (RKD):** a distillation mechanism that penalizes changes in pairwise similarity relationships between image descriptors, helping to maintain relational consistency across sequential tasks. Instead of preserving absolute predictions, RKD preserves *relative* similarities between pairs. Let $s_{ij} = \text{sim}(\phi(I_i), \phi(I_j))$ denote the cosine similarity under the previous model and $\hat{s}_{ij}$ under the current one. The RKD loss is:

$$\mathcal{L}_{\text{RKD}} = \sum_{(i,j)} \left(s_{ij} - \hat{s}_{ij}\right)^2. \tag{2.10}$$

A key concept is the introduction of a fixed-size, similarity-aware memory buffer. This buffer stores a subset of past descriptors along with similarity labels, allowing the use of

contrastive learning via triplet sampling online. Despite the streaming setup, this buffer, whose capacity is of a fixed predefined size, as commonly happens in continual learning setups, allows for continual access to informative positive and negative samples.
AirLoop learns using a triplet loss:

$$\mathcal{L}_{\text{triplet}} = \max(0, \|\bar{\mathbf{f}}_q - \bar{\mathbf{f}}_p\|_2^2 - \|\bar{\mathbf{f}}_q - \bar{\mathbf{f}}_n\|_2^2 + m), \tag{2.11}$$

where $\bar{\mathbf{f}}_q$ is the normalized descriptor of a query image, $\bar{\mathbf{f}}_p$ a positive (same place), and $\bar{\mathbf{f}}_n$ a negative (different place), and m is a margin.

The training objective is a combination of standard triplet loss and the two regularization terms:

$$\mathcal{L} = \mathcal{L}_{\text{triplet}} + \lambda_1 \mathcal{L}_{\text{RMAS}} + \lambda_2 \mathcal{L}_{\text{RKD}}, \tag{2.12}$$

where λ_1 and λ_2 control the influence of the respective terms.

Descriptors are extracted using a VGG-19 backbone followed by a GeM pooling layer, producing compact 1024-dimensional embeddings. The compact global descriptor $\mathbf{f} = \phi(I) \in \mathbb{R}^D$ for an image I is then normalized as $\bar{\mathbf{f}} := \mathbf{f}/\|\mathbf{f}\|_2$ and used for similarity-based retrieval. AirLoop is trained sequentially, one environment at a time, with no access to past data, making it suitable for real-world robotics where memory and computing are limited. Results on TartanAir, Nordland, and Oxford RobotCar datasets show that AirLoop outperforms baseline continual learning methods in most metrics, demonstrating its robustness to environmental drift. [47]

2.3.2 VIPeR: Visual Incremental Place Recognition

VIPeR [48] improves upon AirLoop by simultaneously re-designing the three pillars of a continual VPR system—metric learning, memory management, and regularisation. Figure 2.9 illustrates its main working paradigm. Its contributions are:

- **Adaptive–Mining Triplet Loss**, which dynamically modulates sampling difficulty during training;
- **Multi-stage Memory Bank**, which separates short- and long-term rehearsal to curb both overfitting and forgetting;
- **Probabilistic Knowledge Distillation** (PKD), which aligns pairwise-similarity *distributions* rather than single logits.

Notation: Superscripts "anc", "pos", and "neg" mark anchor, positive, and negative samples, respectively. Epoch index is k, the margin is $\delta > 0$, thresholds for difficulty adaptation are T_d (too difficult) and T_e (too easy).

Adaptive–Mining Triplet Loss

Given a triplet $\big(I^{\text{anc}}, I^{\text{pos}}, I^{\text{neg}}\big)$, similarity between anchor and positive s^{ap} and similarity between anchor and negative s^{ar} are computed as such:

$$s^{\text{ap}} = \text{sim}\big(\phi(I^{\text{anc}}), \phi(I^{\text{pos}})\big), \quad s^{\text{an}} = \text{sim}\big(\phi(I^{\text{anc}}), \phi(I^{\text{neg}})\big).$$

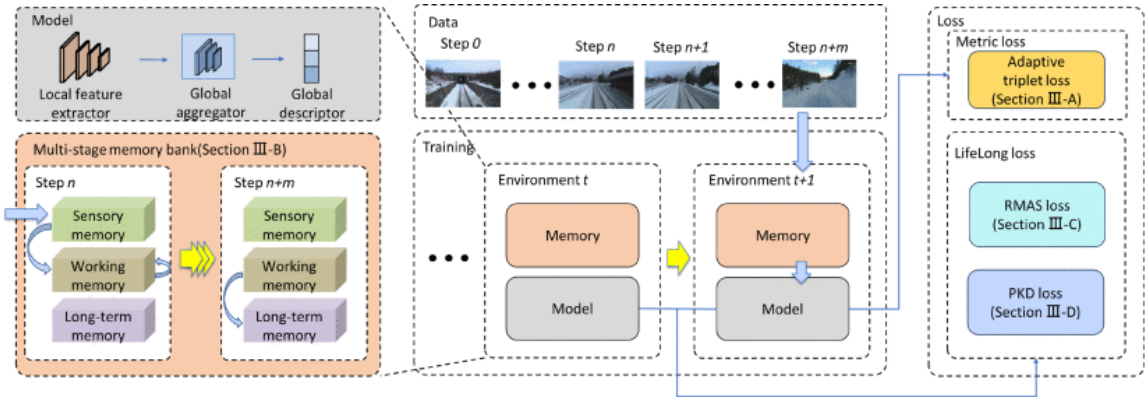


Figure 2.9: VIPeR accompanies the place recognition model with a multi-stage memory bank for rehearsal, adaptive mining, relational memory aware synapses (RMAS), and probabilistic knowledge distillation (PKD) for regularization. [48]

The change in triplet loss across epochs, $\Delta\mathcal{L}^{(k)}_{\text{triplet}} = \left|\mathcal{L}^{(k)}_{\text{triplet}} - \mathcal{L}^{(k-1)}_{\text{triplet}}\right|$, determines the *mining regime*:

$$(i,j) = \begin{cases} \text{easier samples}, & \Delta\mathcal{L}_{\text{triplet}} > T_d, \\ \text{harder samples}, & \Delta\mathcal{L}_{\text{triplet}} < T_e, \\ \text{keep current}, & \text{otherwise.} \end{cases}$$

The resulting loss is

$$\mathcal{L}_{\text{ada}} = \max\{\, s^{\text{an}} - s^{\text{ap}} + \delta,\, 0\} \tag{2.13}$$

Multi-stage Memory Bank

The Multi-stage Memory Bank is divided into three stages: Sensory, Working, and Long-Term. Images pass from the sensory in a FIFO order and are then promoted to working and long-term memory based on a probability threshold.

Stage	Capacity ($l_\star$)	Lifetime	Role
Sensory ($\mathcal{M}^{\text{sn}}$)	l_{sn}	a few steps	most-recent context
Working ($\mathcal{M}^{\text{wk}}$)	l_{wk}	current env.	on-line rehearsal
Long-Term($\mathcal{M}^{\text{lt}}$)	l_{lt}	all envs.	global rehearsal

When $\mathcal{M}^{\text{wk}}$ overflows, newly arriving descriptors are promoted into $\mathcal{M}^{\text{lt}}$ with probability $p = l_{\text{wk}}/n_{\text{seen}}$, where n_{seen} is the number of frames processed in the current environment.

Probabilistic Knowledge Distillation (PKD)

For each mini-batch, a similarity matrix $H^t_{uv} = f_t(I_u)\cdot f_t(I_v)/\sqrt{d}$, is formed and the corresponding teacher matrix H^{t-1} is computed with the frozen model f_{t-1}. Their softmax-normalized distributions P^t and P^{t-1} are aligned via Kullback–Leibler divergence:

$$\mathcal{L}_{\text{PKD}} = \sum\nolimits_{u,v} P^{t-1}_{uv} \log \frac{P^{t-1}_{uv}}{P^t_{uv}} \tag{2.14}$$

NetVLAD and DINOv2

To reduce reliance on hand-crafted pooling and to leverage large-scale self-supervised representations, VIPeR optionally switches its back-end from the GeM aggregator to NetVLAD, fed by descriptors extracted from either VGG-19 or DINOv2:

- Local features $\mathbf{x} \in \mathbb{R}^{H\times W\times C}$ are obtained from a *frozen* DINOv2 ViT; they capture generic semantics robustly across domains.
- NetVLAD clusters those features into K centres and returns a $d{=}K\times C$ vector which replaces the GeM descriptor in all equations above (no change is required to the loss functions).
- Fine-tuning with DINOv2 will only happen to the NetVLAD layer and the final linear projector, leaving the ViT backbone untouched; this keeps the memory footprint identical to the GeM variant while giving a $\sim$15% average-recall boost. Similar results are obtained with VGG-19 and NetVLAD; in this case, VGG is unfrozen and fine-tuned.

Final Objective

$$\mathcal{L} = \mathcal{L}_{\text{ada}} + \lambda_1 \mathcal{L}_{\text{RMAS}} + \lambda_2 \mathcal{L}_{\text{PKD}} \tag{2.15}$$

where $\lambda_1, \lambda_2 \geq 0$ balance rehearsal and distillation terms. $\mathcal{L}_{\text{RMAS}}$ is identical to that of *AirLoop*, now evaluated over the multi-stage memory samples.

2.4 Metrics

Good metrics are crucial for evaluating the performance of ranking and classification models. Below are the standard metrics used in VPR tasks for evaluating ordered lists of results.

2.4.1 $\mathbf{F}_1$ and $\mathbf{F}_\beta$ Score

Precision and **Recall** are two core metrics in binary classification tasks:

- **Precision:** The ratio of correctly predicted positive samples to all predicted positives:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}. \tag{2.16}$$

  It reflects how many of the retrieved results are relevant.

- **Recall:** The ratio of correctly predicted positive samples to all actual positives:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}. \tag{2.17}$$

  It reflects how many of the relevant results are successfully retrieved.

F_1-score. The **F_1-score** is the harmonic mean of precision and recall. It is used when both false positives and false negatives are considered equally costly:

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{2.18}$$

This metric is practical in imbalanced datasets, where accuracy alone may be misleading.

F_β-score. A more general form is the **F_β-score**, which introduces a weight $\beta > 0$ to control the trade-off between precision and recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}. \tag{2.19}$$

- $\beta = 1$ recovers the F_1 score.
- $\beta > 1$ gives more weight to recall (penalizing false negatives).
- $\beta < 1$ gives more weight to precision (penalizing false positives).

F_β is useful when task-specific needs dictate favoring one type of error over the other.

2.4.2 Recall@K

Recall at K, denoted as **Recall@K** (or **R@K**), measures the fraction of relevant items that are successfully retrieved in the top-K recommendations. It answers the question: "Out of all the items that are relevant, how many did we find in the top K suggestions?"

This metric is used when the user is not expected to look past the first few results. It is defined as:

$$\text{Recall@K} = \frac{\text{Number of relevant items in the top K}}{\text{Total number of relevant items}} \tag{2.20}$$

For a given user, if five images are of the same place in the entire dataset, and the network suggests three of those images in its top ten list (i.e., K=10), the Recall@10 would be:

$$\text{Recall@10} = \frac{3}{5} = 0.6 = 60\%$$

2.4.3 AP@K (Average Precision at K)

Average Precision at K (AP@K) is a single-number summary of the quality of a ranked list of results of length K. It is the average of the precision values calculated at the position of each relevant item in the list. Unlike Recall@K, AP@K rewards models that place relevant items higher up in the ranking.

The formula for Average Precision@K is:

$$\text{AP@}K = \frac{1}{R} \sum_{k=1}^{K} P(k) \cdot \text{rel}(k) \tag{2.21}$$

where:

- K is the cutoff length of the ranked list;
- $R = \min(\#\text{ relevant items}, K)$ is the number of relevant items up to position K;
- $P(k)$ is the precision at rank k, i.e., the proportion of relevant items among the top k;
- $\text{rel}(k) \in \{0, 1\}$ is an indicator function equal to 1 if the item at rank k is relevant, 0 otherwise.

Essentially, the algorithm iterates through the ranked list, and every time it encounters a relevant item, it calculates the precision at that point and then averages these precision scores over the total number of relevant items.

For example, if there are three relevant items and our model returns a ranked list `[Relevant, Irrelevant, Relevant, Relevant, Irrelevant]`:

- At rank 1 (Relevant): $P(1) = 1/1 = 1.0$
- At rank 2 (Irrelevant): No calculation needed since $\text{rel}(2) = 0$
- At rank 3 (Relevant): $P(3) = 2/3 \approx 0.67$
- At rank 4 (Relevant): $P(4) = 3/4 = 0.75$
- At rank 5 (Irrelevant): No calculation needed since $\text{rel}(5) = 0$

The AP@5 would be the average of the precision scores at relevant positions: $\text{AP@5} = \frac{1.0+0.67+0.75}{3} \approx 0.81$

2.4.4 Recall@100P (Recall at 100% Precision)

Recall at 100% Precision, often denoted as **Recall@100P** (or **R@100P**), is a diagnostic metric that evaluates recall at the strictest possible decision threshold. It answers the practical question: "If we adjust our confidence threshold to the point where we make absolutely zero false-positive errors, what fraction of the total relevant items are we able to find?"

This metric is not defined by a static formula but by a procedure based on the model's ranked scores. It is the maximum recall achievable at a precision of 1.0.

$$\text{Recall@100P} = \text{Recall at a threshold } T \text{ where Precision}(T) = 1.0 \tag{2.22}$$

This metric is exceptionally useful in operational scenarios where false positives are extremely costly (e.g., medical diagnosis, financial fraud detection, or, in this case, strict loop-closure), and the goal is to define a "safe" threshold of confidence or similarity to only obtain true positives.

2.4.5 Average Precision (for Classification)

When applied to binary classification, the **Average Precision (AP)** score summarizes a model's performance across all classification thresholds. It is computed as the weighted sum of precisions at each threshold, where the weight is the increase in recall from the previous threshold:

$$\text{AP} = \sum_n (R_n - R_{n-1})\, P_n \tag{2.23}$$

where:

- P_n and R_n are the precision and recall at the n-th threshold;
- Thresholds are defined by the decision scores at each prediction;
- The summation is taken over all points where recall increases (i.e., each time a positive sample is encountered).

This formulation is used by libraries such as `scikit-learn` in the `average_precision_score` function. Note that it does not rely on interpolation and is distinct from computing the area under the precision-recall curve via the trapezoidal rule, which tends to be overly optimistic.

Unlike AP@K, this metric considers the complete ranked list of predictions and is especially appropriate for imbalanced datasets, where precision-recall analysis offers more informative insights than ROC curves.

Notation: Given that AP@K can be confused with AP (Average Precision for Classification), from this point forward, Average Precision for Classification will be referred to as **Average Precision Score (APS)**

2.5 Explainability

Explainability (or interpretability) refers to the degree to which a human can understand the cause of a decision made by a machine learning model. This is particularly important in computer vision, where models often make decisions based on subtle patterns that are not immediately obvious to human observers. In this section, LIME and Occlusion Sensitivity are reported because they are the standard for Computer Vision tasks, but other methodologies exist, such as SHAP and LEMNA. Other XAI (eXplainable AI) methods, thought for Computer Vision models, have not been tested and are thus not reported, such as Grad-CAM. [49]

2.5.1 LIME (Local Interpretable Model-Agnostic Explanations)

LIME [50] is a technique that approximates any black box machine learning model with a local and interpretable model to explain each prediction. LIME is one of the few methods that work for tabular data, text, and images.

For image classification, LIME works by:

- Generating perturbed versions of the input image by turning superpixels on or off

- Getting predictions from the black box model for these perturbed images
- Training a simple, interpretable model (e.g., linear regression) on these perturbed samples
- Using the interpretable model's coefficients to identify which superpixels are most important for the prediction

The key advantage of LIME is its model-agnostic nature, meaning it can explain any classifier regardless of its internal architecture. An example of its work, shown by highlighting the relevant areas, can be seen in Fig. 2.10.

2.5.2 Occlusion Sensitivity

Occlusion sensitivity [51] is a straightforward approach to understanding which regions of an image are important for a model's prediction. The method systematically occludes (masks or removes) different parts of the input image and observes how the model's confidence in its prediction changes.

The process works by:

- Sliding a patch (usually gray or black) across the entire image
- Recording the model's prediction confidence at each position of the occluding patch
- Creating a sensitivity map where each pixel's value represents how much the prediction confidence drops when that region is occluded

Regions that cause a more substantial drop in confidence when occluded are considered the most important for the model's decision. While computationally expensive due to the need for multiple forward passes, occlusion sensitivity is relatively simple to implement, and it produces intuitive and reliable explanations that directly measure the causal impact of different image regions on the model's output. An example of its work, shown with the classic heatmap, can be seen in Fig. 2.10.

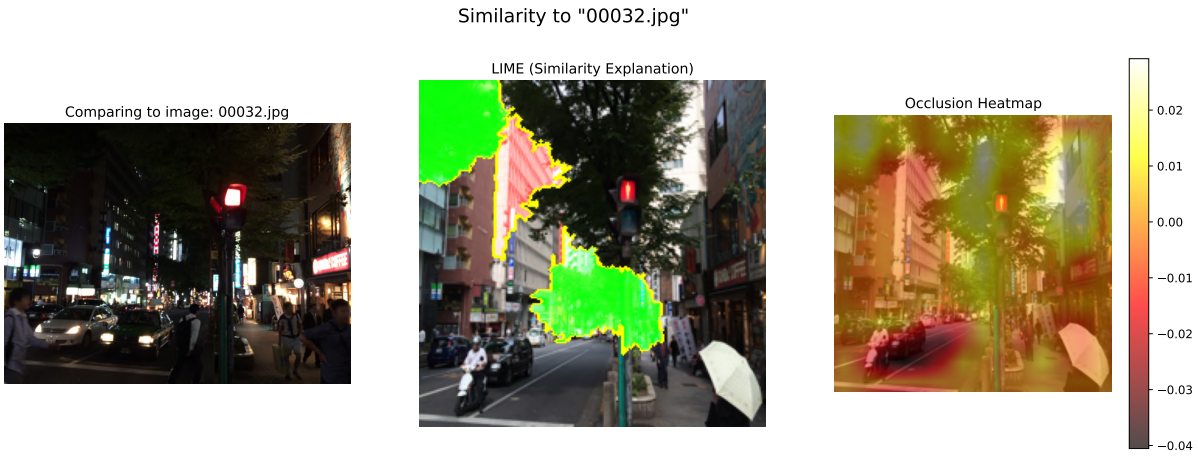


Figure 2.10: Example of LIME and Occlusion Sensitivity explanations of images classification.

Chapter 3

Datasets

Visual place recognition requires diverse and challenging datasets to evaluate the robustness of algorithms under various conditions. This chapter presents six datasets, each addressing different aspects of the visual place recognition challenge: seasonal variations, a wide variety of places and architectural styles, anonymized 360-degree images, illumination changes, computer-generated images and indoor environments.

3.1 Nordland

The Nordland [52] dataset captures the same railway journey during different seasons. The dataset is designed for place recognition across seasons, showing extreme appearance changes, from the snow in winter to the flowering trees in spring. This seasonal variation creates extreme changes in the visual appearance of identical geographical locations. The images are organized in folders, already split into test and train sets, and split again for each season. Each season folder contains section folders in which images extracted from the video are enumerated and ordered by the time the picture was taken.

This dataset is one of the oldest, having been published in 2013, but it is still used for training and testing in the state of the art (e.g., in VIPeR citeviper).

Figure 3.1: Frames extracted from the Nordland dataset and from the same place in spring, summer, fall, and winter. [53]

3.2 GSV-Cities

The GSV-Cities [54] dataset is a large-scale dataset with a wide variety of perceptual changes over 14 years, covering 40 cities spread across all continents. This dataset provides highly accurate ground truth, allowing for straightforward mini-batches. All of the 530,000 images

are obtained from Google Street View Time Machine. The images are divided into more than 62,000 different places, each containing from 4 to 20 images and each at least 100 meters away from any other place in the dataset (see Fig. 3.2). The images are organized in folders, one for each city, and each city has a corresponding CSV file that contains information on the spatial location of all the images, as well as the place they belong to and the period when the photo was taken. The authors of the paper suggest using this dataset for training a general-purpose model and testing it on other datasets, as it should generalize well enough on multiple tasks, given the sizable number of highly different images and places.

Figure 3.2: Three examples of places in GSV-Cities. Each place is depicted by a set of images (here, six in a row) representing the same physical location. As such, they are indexed by the same ID. The number of images depicting one place in GSV-Cities varies from 4 to 20. [54]

3.3 NYC-Indoor-VPR

The NYC-Indoor-VPR [55] dataset is a collection of over 36,000 images compiled from 13 distinct crowded scenes in New York City taken under varying lighting conditions with appearance changes, with each scene having multiple revisits across a year. This dataset comprises images from different crowded scenes in New York City, taken under varying lighting conditions with seasonal and appearance changes.

The dataset is composed of images recorded in New York City from April 2022 to April 2023, with footage captured using hand-held Insta360 One X2 spherical cameras, generating videos with a resolution of 1920x960. Afterward, people and cars are segmented and removed, as can be seen from Fig. 3.3. The recorded images are of 13 different floors/scenes within the six buildings. The chosen buildings are the Oculus, New York University Silver Center for Arts and Science, Elmer Holmes Bobst Library, Morton Williams Supermarket, and the Metropolitan Museum of Art. These settings represent a broad range of indoor spaces, including shopping malls, teaching buildings, libraries, supermarkets, and museums.

To establish ground truth for this dataset, a semi-automatic annotation approach is used. The dataset thus contains annotations that divide images into scenes, but they do not correspond exactly to different "places" as the division is merely spatial. On the one hand, this may create difficult situations where images a human would consider the same place are labeled as different places; on the other hand, the models are forced to learn how to recognize the same

place without having any liberty from a perspective or partial overlap.

The dataset's long-term nature, spanning an entire year, allows an evaluation of how well algorithms handle gradual changes in indoor environments, such as seasonal decorations, lighting variations, and crowd dynamics in public indoor spaces.

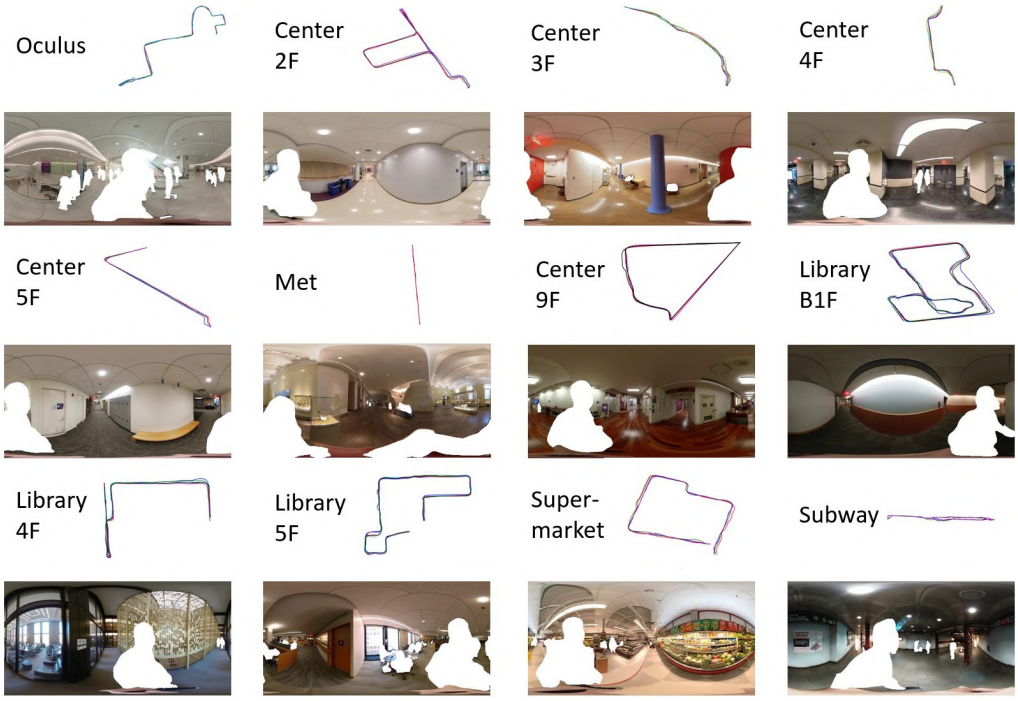


Figure 3.3: Trajectories annotated by the semi-automatic method and example images of 12 scenes in NYC-Indoor-VPR. [55]

3.4 Tokyo-24/7

The Tokyo-24/7 [56] dataset consists of 375 distinct query locations taken at day/evening/night (see Fig. 3.4) for a total of 1125 query images, all with their corresponding coordinates. The authors suggest the use of Google Street View, given the coordinates, to expand the database. In this thesis, only the query images will be used. This dataset addresses the problem of visual place recognition for situations where the scene undergoes a significant change in appearance, for example, due to illumination (day/night), the number of people, and structural modifications over time, such as different parts of the same building being lit and advertisements changing. Each place in the query set is captured at different times of day: daytime, sunset, and night, with corresponding database street-view images at close-by positions and longitude and latitude coordinates. The 1125 images are captured by Apple iPhone 5s and Sony Xperia smartphones. These images are taken at 125 distinct locations, facing three different directions, yielding 375 distinct "places", each with three pictures taken at different times of the day. This temporal diversity makes Tokyo 24/7 a standard benchmark for evaluating algorithms' robustness to the lighting changes that occur throughout a 24-hour cycle.

Figure 3.4: The first few query images of Tokyo-24/7 with midday, sunset, and night for each place.

3.5 TartanAir

TartanAir [57] is a dataset for robot navigation tasks, with data collected in photo-realistic simulation environments with the presence of moving objects, changing light, and various weather conditions. By collecting data in simulations, the dataset provides multi-modal sensor data and precise ground truth labels such as stereo RGB images.

A special goal of the dataset is to focus on challenging environments with changing light conditions, adverse weather, and dynamic objects, where state-of-the-art SLAM algorithms struggle in tracking camera pose and constantly get lost in some sequences.

The simulation-based nature of TartanAir allows for perfect ground truth data, which is often difficult to obtain in real-world scenarios. This enables precise evaluation of algorithmic performance and provides researchers with a controlled environment to test specific hypotheses about visual place recognition under various challenging conditions.

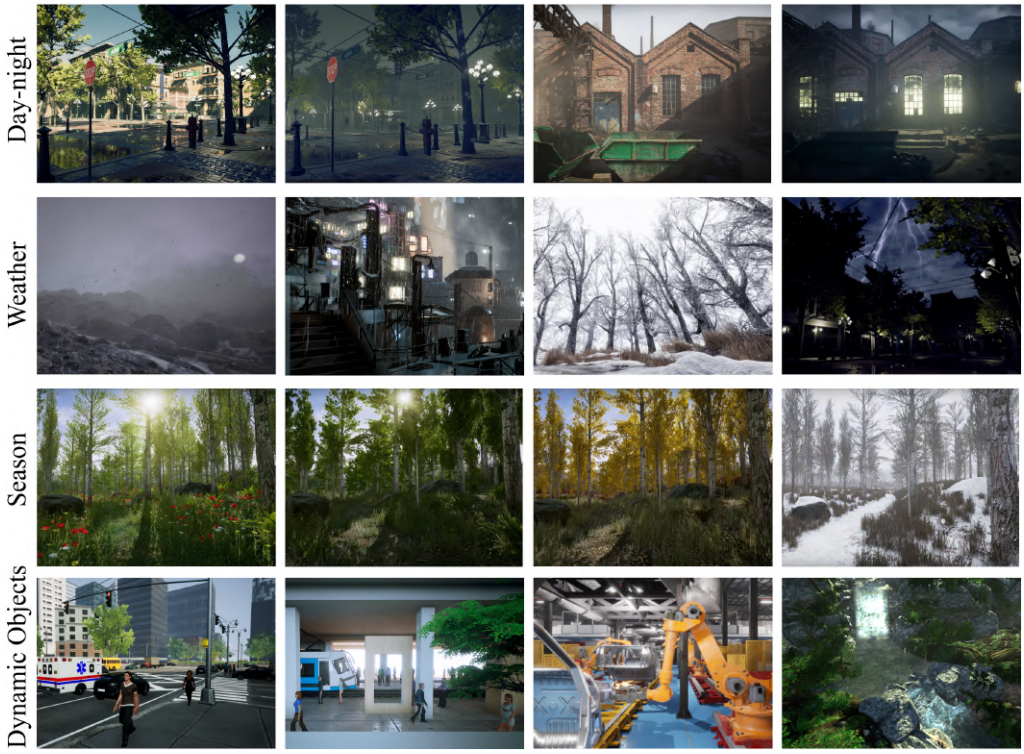


Figure 3.5: Examples of images under different conditions from the TartainAir dataset. [57]

3.6 OpenLORIS-Scene

The OpenLORIS-Scene [58] dataset is designed for lifelong SLAM and continual learning in robotics. The dataset aims to help evaluate SLAM and scene understanding algorithms for real-world deployment by providing visual, inertial, and odometry data recorded with real robots in real scenes and ground-truth robot trajectories acquired by a motion capture system or high-resolution LiDARs.

The data were collected by a wheeled robot moving at human walking speed or slower. The primary sensors include a RealSense D435i camera and a RealSense T265 camera, both mounted at a fixed height of about 1m. The color images and depth images from D435i are recommended for monocular/RGB-D algorithms, while the dual fisheye images from T265 are recommended for stereo algorithms. Both provide IMU measurements with hardware synchronization with corresponding images. Odometry data from wheel encoders is also provided. The ground-truth trajectory of the robot is obtained by an OptiTrack motion capture system for the office scene and from offline LiDAR SLAM based on the Hokuyo laser scans for other scenes. Each picture taken is thus paired through the timestamp to odometry data (see Fig. 3.6). Using specific tools[1], it is possible to extract loop-closure images that belong to the same place.

[1] https://github.com/scumatteo/loop-closure-inspector

Figure 3.6: Examples of color images in the OpenLORIS-Scene datasets. The upper and lower images in each column show approximately the same place in different data sequences, but the scene has been changed. [58]

Chapter 4

Experiments

This thesis has three primary objectives: first, to compare and select the most suitable architectures for various batch learning scenarios; second, to select or define a metric for evaluating the quality of extracted embeddings, and third, to advance the state of the art in online learning contexts. All experiments were conducted on the raptor-08 server, made available for this research by the Department of Computer Science and Engineering (DISI) of the University of Bologna, Cesena campus. The server is equipped with an NVIDIA RTX A4000 GPU (16GB), an Intel® Xeon® W5-3423 CPU, 64 GB of DDR5 4400 MT/s RAM (4x16 GB), and a 1 TB NVMe Gen4 SSD.

Notation: All the metrics reported are formatted as percentages, rounded to one decimal place.

4.1 Batch Learning

In the batch learning scenarios, multiple datasets representing different VPR tasks were used to train and test multiple combinations of backbones and aggregators. After the training, an evaluation with the mainstream metrics of Recall@K and AP@K was performed, as well as a qualitative study with explainability.
The innovative part of the study is the analysis of the distribution of cosine similarity of descriptors, obtained by calculating for each query its cosine similarity with all the DB elements and creating two distributions of embeddings, one composed of the cosine similarity with descriptors from the same place, the other instead composed of different cosine similarity of embeddings of images from different places. This is done because the model should not only rank images well, as it is measured by Recall@K and AP@K, but it should also be able to discriminate well between same-place images and different-place images. As the distributions are highly imbalanced (the same place images are always far less than the different place images), a metric that takes this into account was necessary. The problem of evaluating loop closure is not a new one, and a few metrics have already been proposed, but the cosine distributions have never been analyzed in the VPR context before.
The first attempts were made by trying to measure the distance between distributions with specific statistical metrics, such as Wasserstein Distance and Jensen-Shannon Distance. Nei-

ther of these metrics was perfectly tailored for this task because they gave equal importance to the two distributions, while the positive one is the one that we mainly want to discriminate. In particular, Wasserstein Distance was unable to perform well enough on multiple occasions, as it did not take into account relative shift but only absolute distance, and this was unacceptable in some cases with GeM as the aggregator, as sometimes it spread the cosine similarities on a small range of values. Afterward, other classification metrics such as ROC AUC and Overlap coefficient were tested, but both gave too much emphasis to the true negatives (the well-predicted different places). In this case, the only images we are interested in are the true positives (same place, well predicted), the false positives (different place, wrongly predicted as the same), and the false negatives (same place, wrongly predicted as different). With these premises, the most obvious metric is the F_1-score, which takes just these factors into account. The problem with the F_1-score is that it is threshold-dependent and does not evaluate the overall performance of the model.
The other two viable options at this point are the Recall at 100% Precision, which checks the recall while having zero false positives (100% precision), and that has already been used by previous work on VPR tasks [48]; and the Average Precision Score. While Recall@100P is the mainstream metric on many benchmarks, including the ones used in the Online Learning section, it does not take the general behavior of the model into account, and it risks selecting a highly specialized model: a model could be tuned to excel only at that single point of 100% precision while generalizing poorly. The Average Precision Score instead considers the entire performance of the model, even if it is not restricted to 0% false positives.
For a SLAM algorithm, false positives may be filtered out by an excessive distance from the view or other parameters; thus, the proposed metric to evaluate the "distance" between descriptors and the one that will be used in this section is the Average Precision Score. Still, the Recall at 100% Precision is also reported in the results table to give a comprehensive idea of the performance of the model under strict requirements.

Warning: the following sections contain plots with different values on the y-axis. The plots were not merged because of the large disparity between y values, but this may lead to similar visual trends while the values are different.

4.1.1 Tokyo-24/7

The initial experiments were conducted on the Tokyo-24/7 [56] dataset, using only the provided query images. The primary goal was to evaluate and compare the performance of different model architectures in a Visual Place Recognition task under the illumination changes that happen during a complete day-night cycle.

Experiment Setup

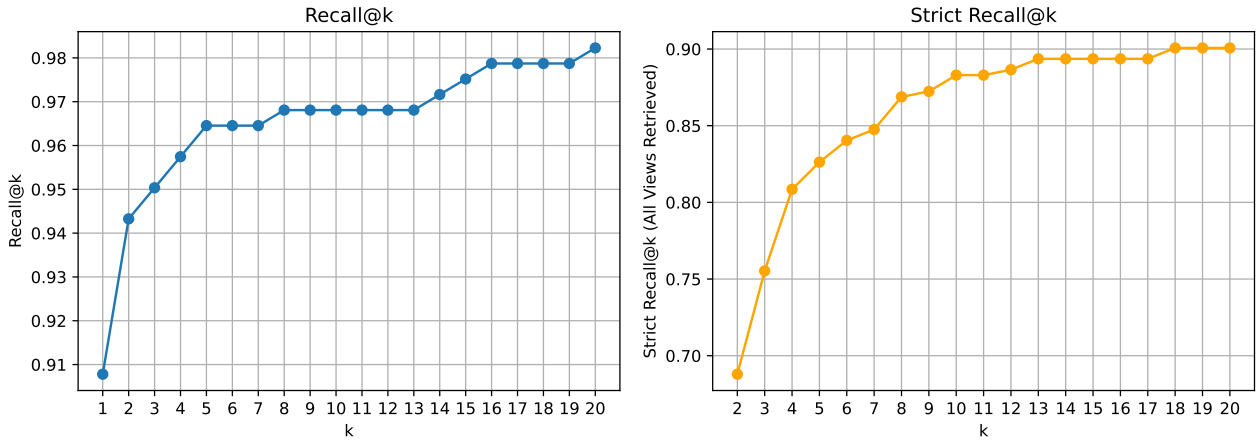
The methodology is as follows: the 1125 query images are organized into 375 triplets, with each triplet containing images of the same location at different hours of the day: midday, sunset, and night. These locations are then split into training and test sets at a 3:1 ratio. Each model architecture, composed of a backbone and an aggregator, is trained for 10 epochs using a triplet loss function with a margin of 0.3 and with an ADAM optimizer with a learning

rate of 1×10^{-4}. The loss is then calculated between an anchor image, a positive image (from the same place at a different time), and a negative image (from a different place). When NetVLAD was used as an aggregator, the best results were achieved with a number of clusters equal to 32 (16, 64, 96, and 128 were also tested).

After training, feature descriptors are extracted for each image in the test set. During this inference stage, the process is timed to estimate the computational time cost per image. Afterward, a cosine similarity matrix is generated by L2-normalizing the embeddings and multiplying the resulting matrix by its transpose. For quantitative evaluation, Recall@K and a "strict" Recall@K metric, where all the two images belonging to a place must be correctly retrieved, are calculated and plotted. A general plot of the cosine similarity value of correct and wrong images rescaled on density is also calculated to check the degree of separation between the two. The ideal situation is when the two distributions do not overlap, so that we can choose a threshold to decide when an image is in the same place. Afterward, the F_1-score on various thresholds is calculated, and the one that maximizes the F_1-score is selected. The Average Precision Score is also calculated to emphasize true positives, false negatives, and false positives only, excluding true negatives from the metrics, as they do not particularly concern this task. For completeness, Recall at 100% Precision is measured as well. Finally, for qualitative analysis, LIME and occlusion sensitivity algorithms are used to generate explainability maps. These maps highlight the image regions most critical to the model's descriptor-matching process.
For this first experiment, given the excellent results achieved by DINOv2 -another ViT foundation model is used as a backbone to compare results: SwinV2, version large window12 192, pre-trained on ImageNet 22k.
When using a VGG backbone, VGG19 was used, as similar but slightly worse results were obtained with a VGG16. The identical process was applied for ResNet50 and ResNet18. The DINOv2 version used is the base one. Both DINOv2 and SWINv2, when used as a backbone, are frozen; VGG and NetVLAD are instead retrained.

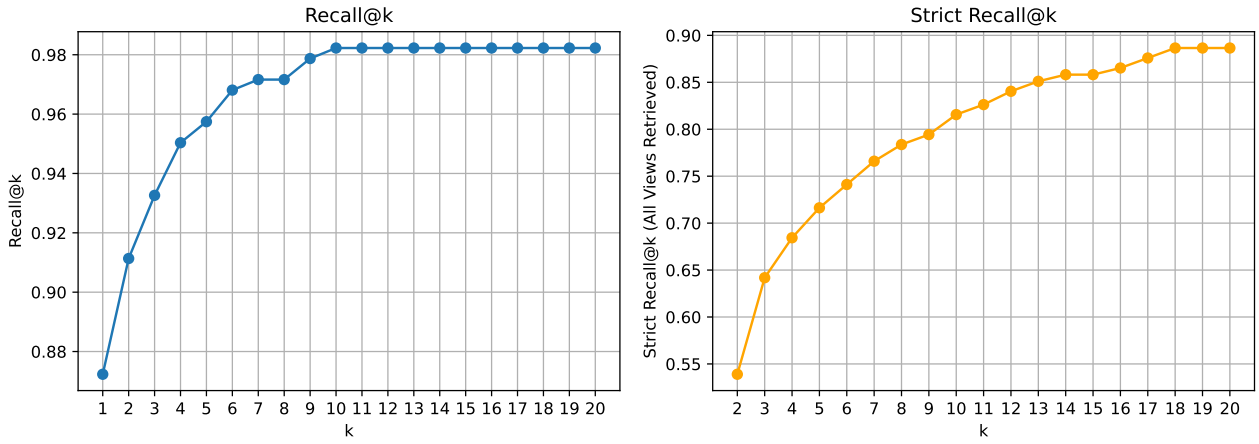
Results

GeM With GeM as the aggregator, relatively good results were reached. The performance, on average, is still worse than NetVLAD, and, in some cases, it collapses the embeddings extracted from the backbone and averages them to a narrow embedding space. More on this will be discussed in the Conclusions section.
In Fig. 4.1, it is possible to see the performance of VGG19 when GeM is applied as an aggregator. The results are quite good, but they are still inferior to the NetVLAD version (see Fig. 4.5) and to the corresponding DINOv2 GeM results (see Fig. 4.3). ResNet50 results instead (Fig. 4.2) outperform the NetVLAD ones (Fig. 4.6), but while the Recall is acceptable, the Strict Recall has quite a lower performance. The SwinV2 results—which are visible in Fig. 4.4—are underwhelming, reaffirming the specialness of DINO as a ViT foundation model.



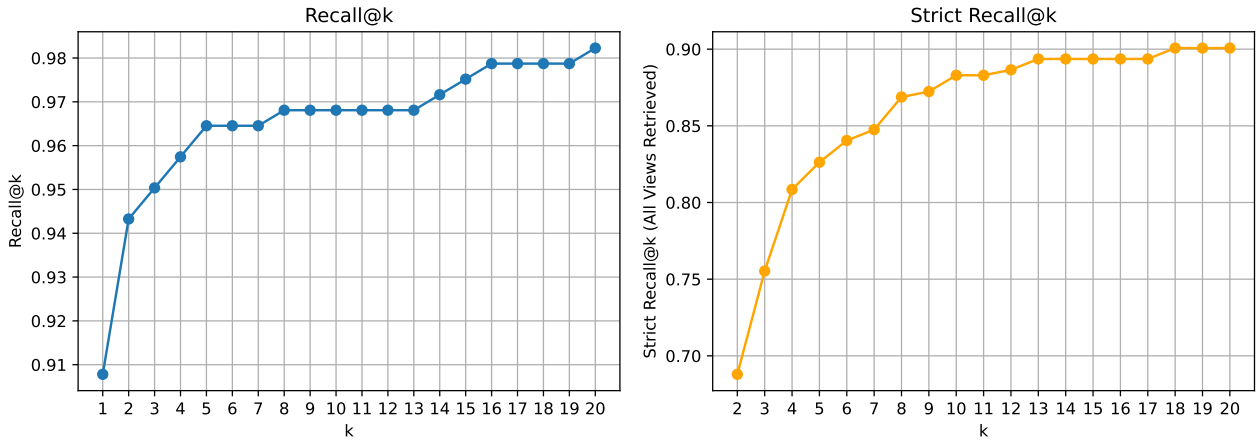
(a) Recall@K (b) Strict Recall@K

Figure 4.1: VGG19 + GeM performance on Tokyo-24/7 test set.



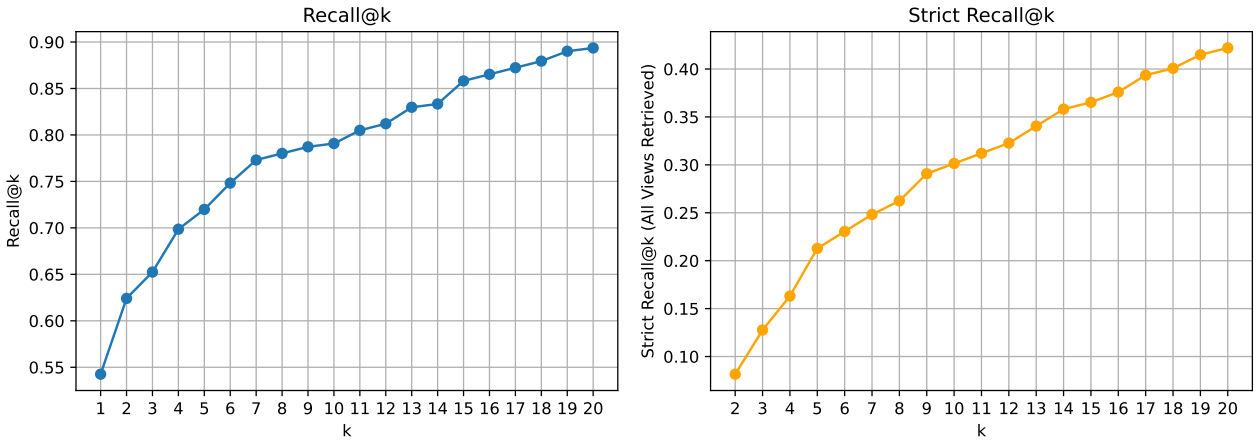
(a) Recall@K (b) Strict Recall@K

Figure 4.2: ResNet50 + GeM performance on Tokyo-24/7 test set.



(a) Recall@K (b) Strict Recall@K

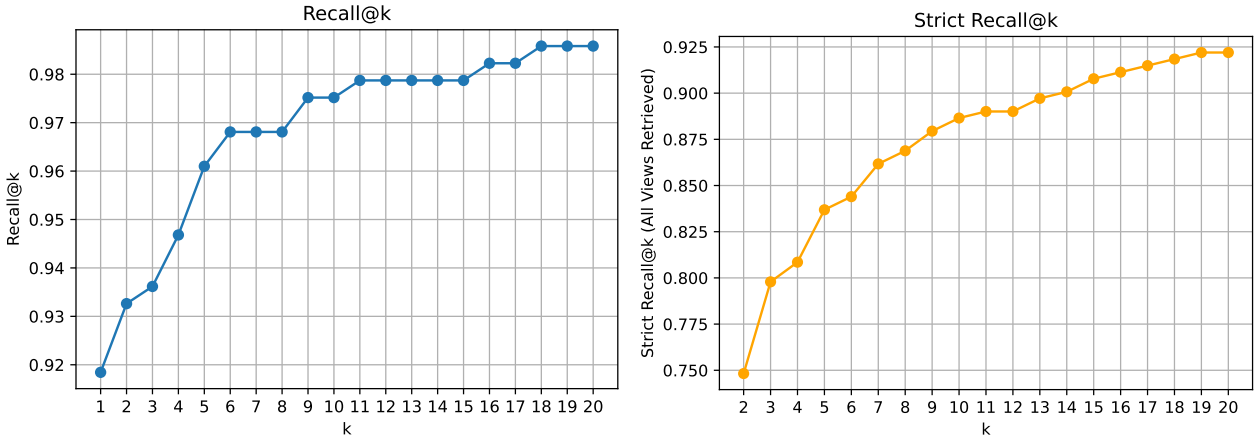
Figure 4.3: DINOv2 + GeM performance on Tokyo-24/7 test set.



(a) Recall@K (b) Strict Recall@K

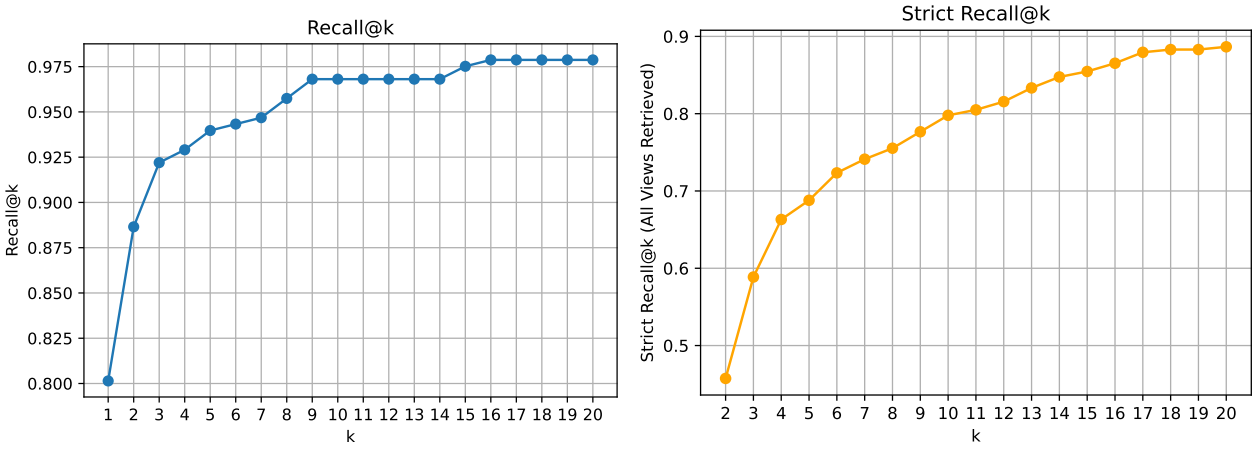
Figure 4.4: SwinV2 + GeM performance on Tokyo-24/7 test set.

NetVLAD With NetVLAD as an aggregator and the number of clusters set to 32, the best results overall were reached. The combination of a general-purpose model and a few images allows the model to reach almost 100% of the recall rate in a small image range. NetVLAD results as the best aggregator for both Recall and Strict Recall for VGG19 (Fig. 4.5), DINOv2 (Fig. 4.7), and SWIN (Fig. 4.8). For ResNet50, the results as plotted in Fig. 4.6 are good but are still outperformed by GeM ones. The accuracy achieved by combining DINOv2 and NetVLAD is excellent; without even using a reranking algorithm, almost 100% of accuracy is reached quickly for both Recall and Strict Recall metrics. These results are also in the final comparison Table 4.1, where this architecture dominates with all the best scores on almost all metrics.



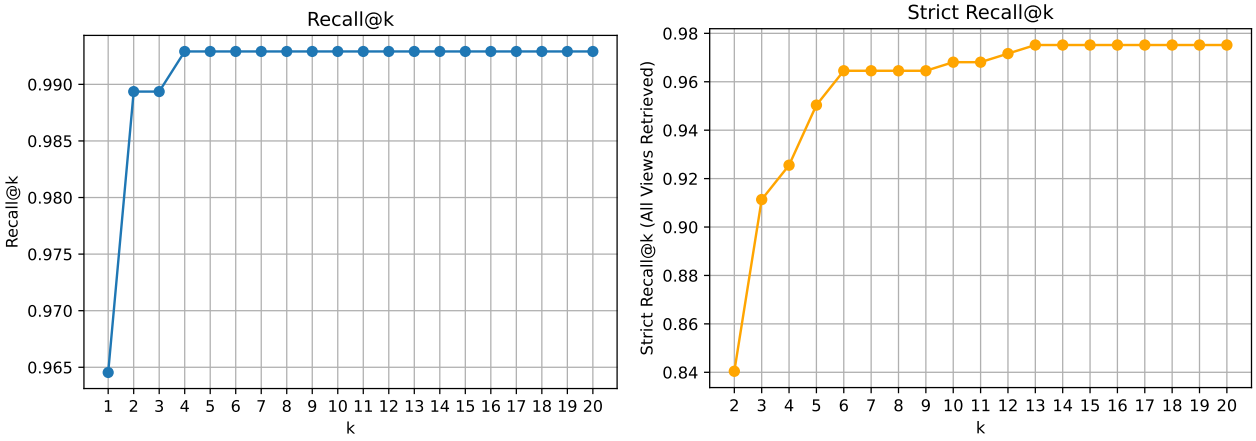
(a) Recall@K (b) Strict Recall@K

Figure 4.5: VGG19 + NetVLAD performance on Tokyo-24/7 test set.



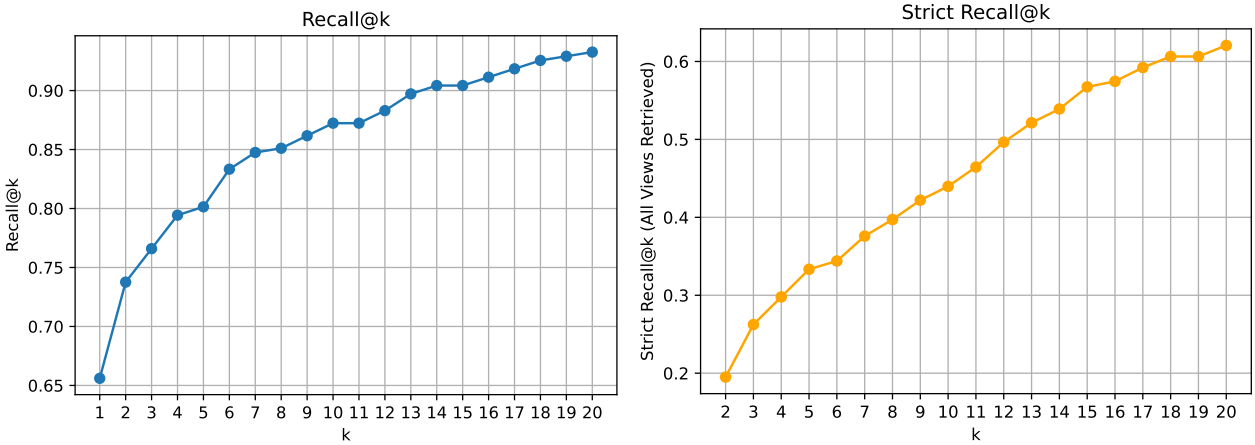
(a) Recall@K (b) Strict Recall@K

Figure 4.6: ResNet50 + NetVLAD performance on Tokyo-24/7 test set.



(a) Recall@K (b) Strict Recall@K

Figure 4.7: DINOv2 + NetVLAD performance on Tokyo-24/7 test set.



(a) Recall@K (b) Strict Recall@K

Figure 4.8: SwinV2 + NetVLAD performance on Tokyo-24/7 test set.

Qualitative Results and Explainability For each of the possible architecture combinations, a qualitative study with explainability has been applied to give a better understanding of the influence on the decisions taken by the model. Two test images, one at midday and one at sunset, are selected, and the top 5 similar embeddings are retrieved and then shown, marking on top of each image both the cosine similarity and a boolean that indicates whether it is the same place or not.

VGG + GeM behaves perfectly on the selected images, even though the third image selected has a similarity score that is quite high in the second case. According to the explainability of the decisions taken, this architecture seems to focus on the trees or on the bigger elements in the images.

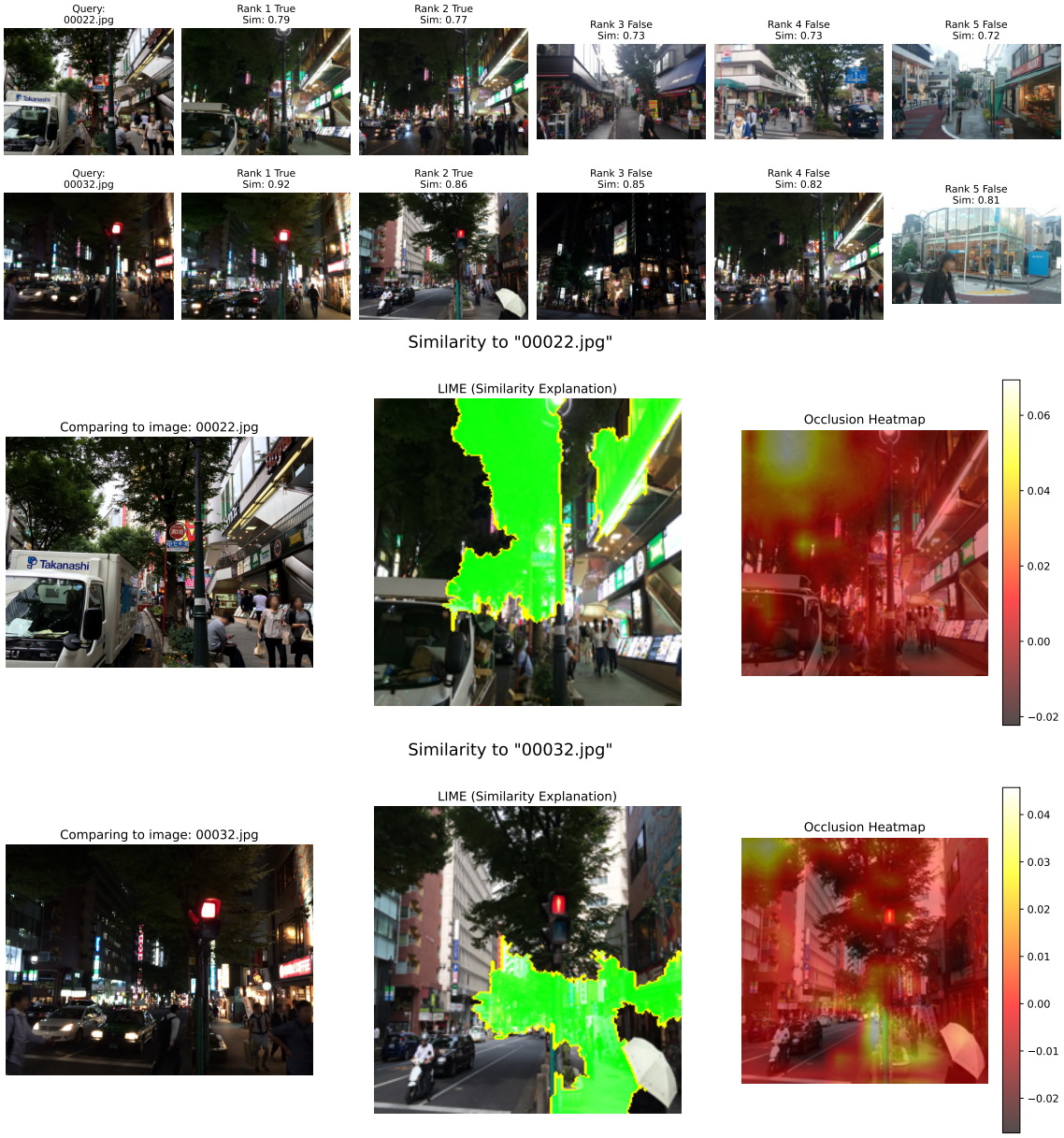


Figure 4.9: VGG19 GeM examples and explainability.

VGG + NetVLAD also behaves perfectly on the selected images, still with the third image selected with a high similarity score in the second case. According to the explainability of the decisions taken, this architecture seems to focus less on the bigger elements in the images and more on details such as building structure and so on. This is in line with how VLAD works with respect to GeM, allowing a better focus on more element types.

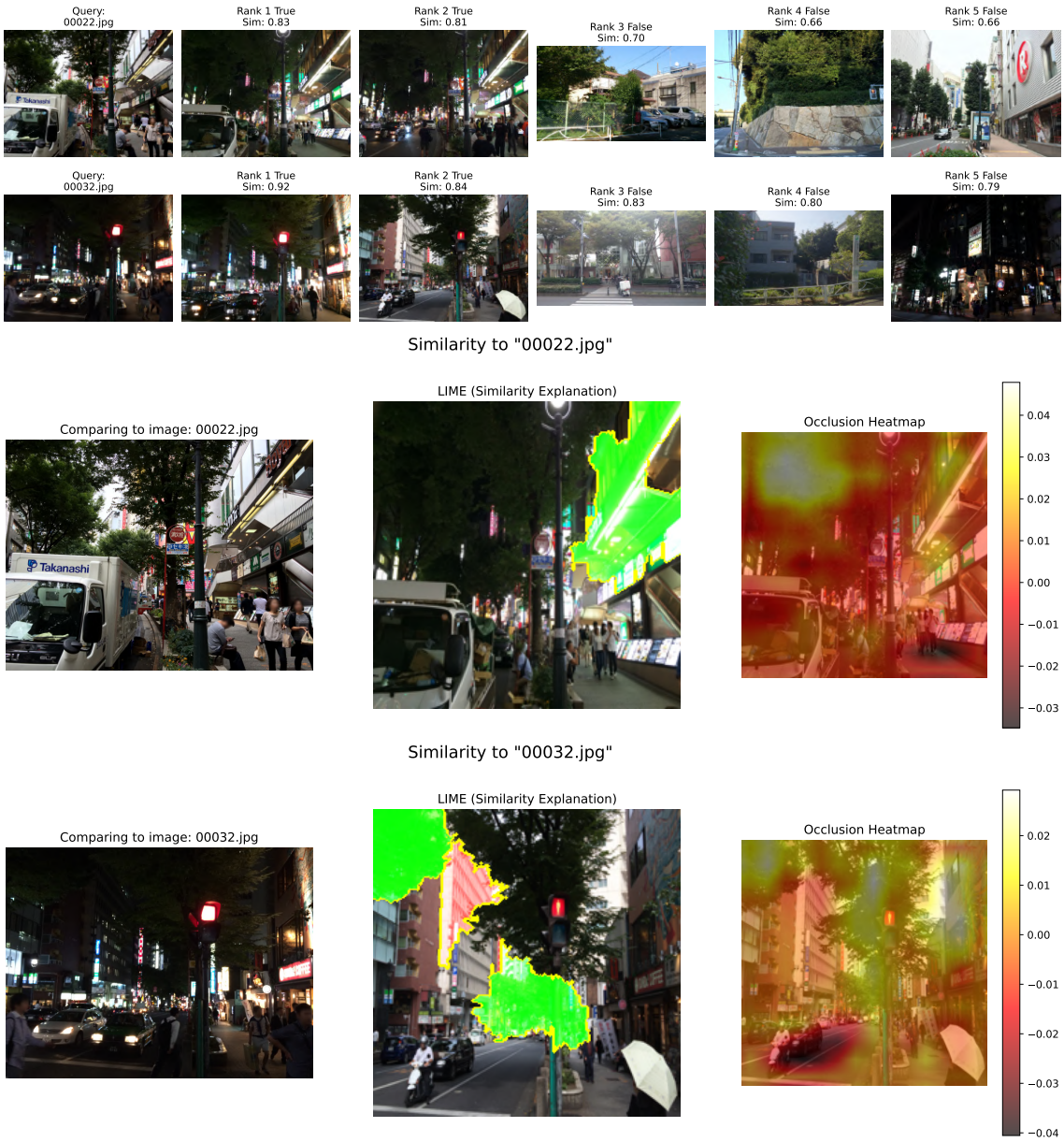


Figure 4.10: VGG19 NetVLAD example and explainability.

ResNet50 + GeM perfectly selects the first images and has too high similarity on the first case instead of on the second, in contrast to VGG. According to the explainability of the decisions taken, this architecture seems to focus on building parts and road elements instead of trees, as VGG19 and GeM did.

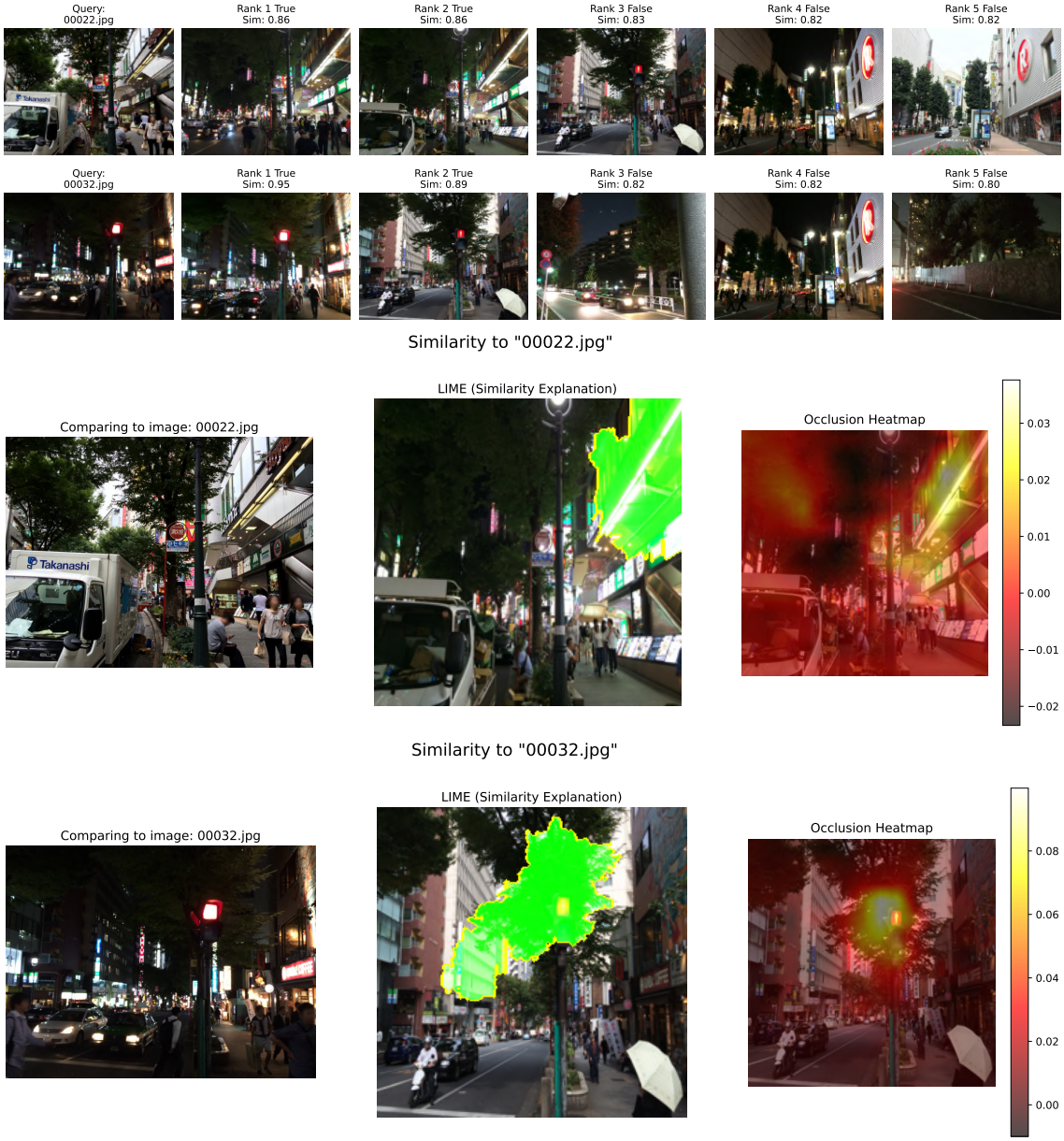


Figure 4.11: ResNet50 GeM example and explainability.

ResNet50 + NetVLAD As it is possible to see from the plots above, NetVLAD performs worse than GeM on ResNet. Not only is the similarity quite low in general, as well as almost uniform, but in the second case, the second most similar image was also retrieved wrongly. By the explainability, it is possible to observe the fact that the architecture is still considered important, but for the image misclassified, the big tree above was a deciding factor.

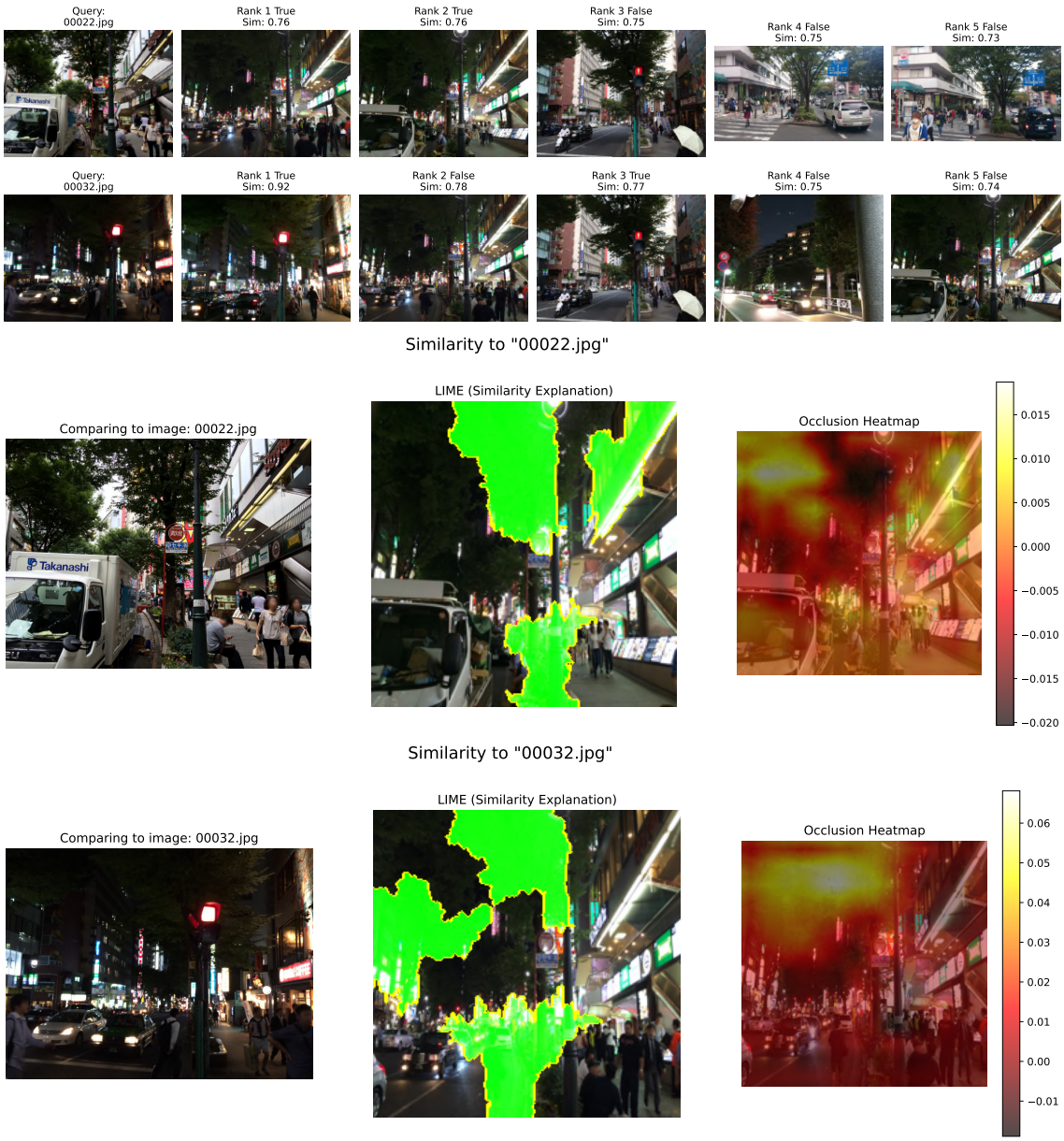


Figure 4.12: ResNet50 NetVLAD example and explainability.

DINOv2 + GeM DINOv2 and GeM also commit the same mistake on the second batch of retrieved images as ResNet50 and NetVLAD. In general, the results are good, but the similarity is squashed on a very high number (this will be analyzed in more detail below). By the explainability, it is possible to observe the fact that the architecture is still considered important, but for the image misclassified, the big tree above was a deciding factor. This model seems to take into account various factors when extracting embeddings, among which are also the light lamp and the sidewalls. Probably, this is also one of the reasons for the misclassification in the second case.

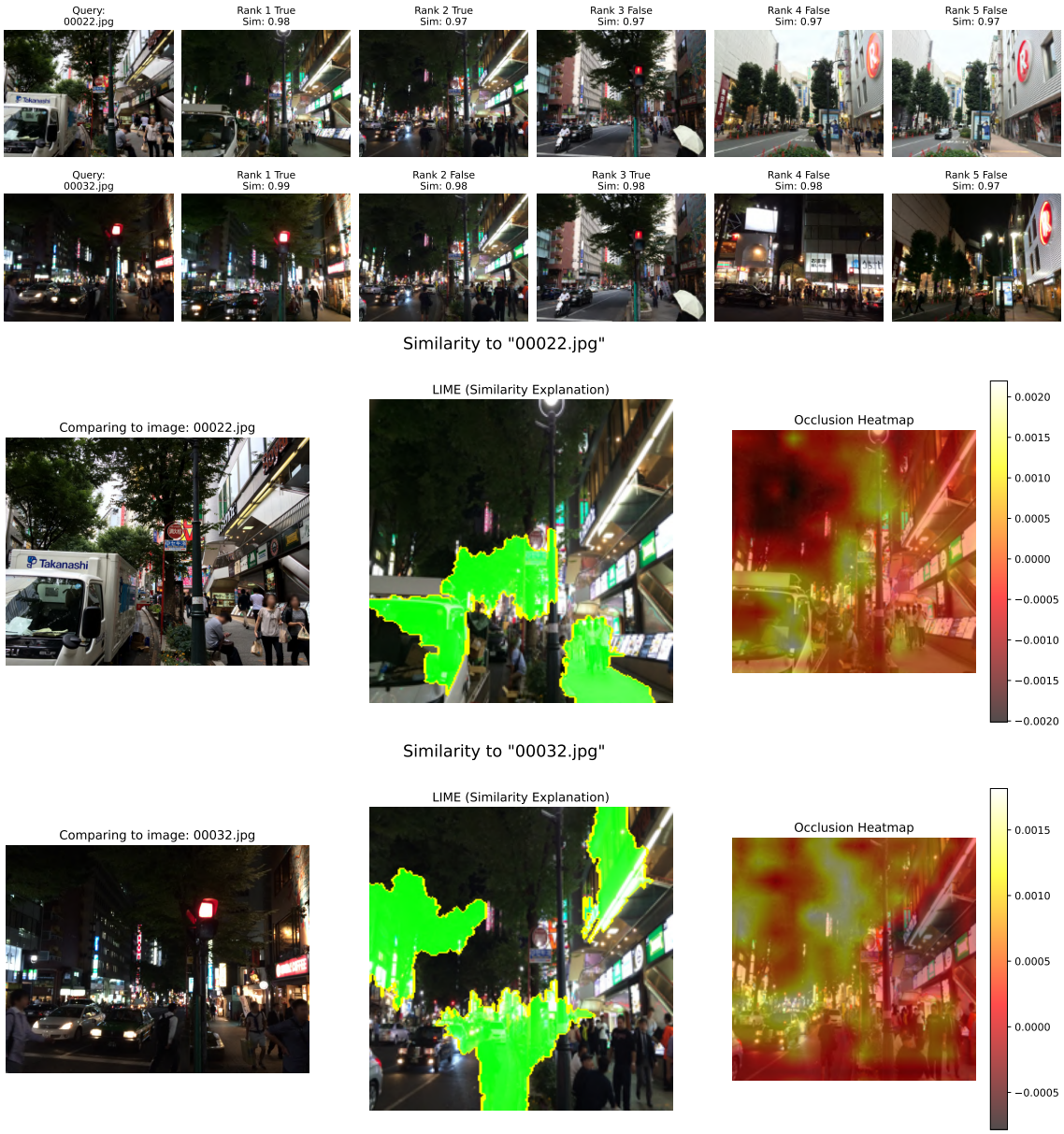


Figure 4.13: DINOv2 GeM example and explainability.

DINOv2 + NetVLAD DINOv2 and NetVLAD also commit the same mistake as they did with GeM. It seems like this backbone has some problems with light changes, as the similarity between images with the same light is far higher than that of images with different light. The explainability results are similar, with the exception of a few different points of emphasis on the heatmap and the LIME selection.

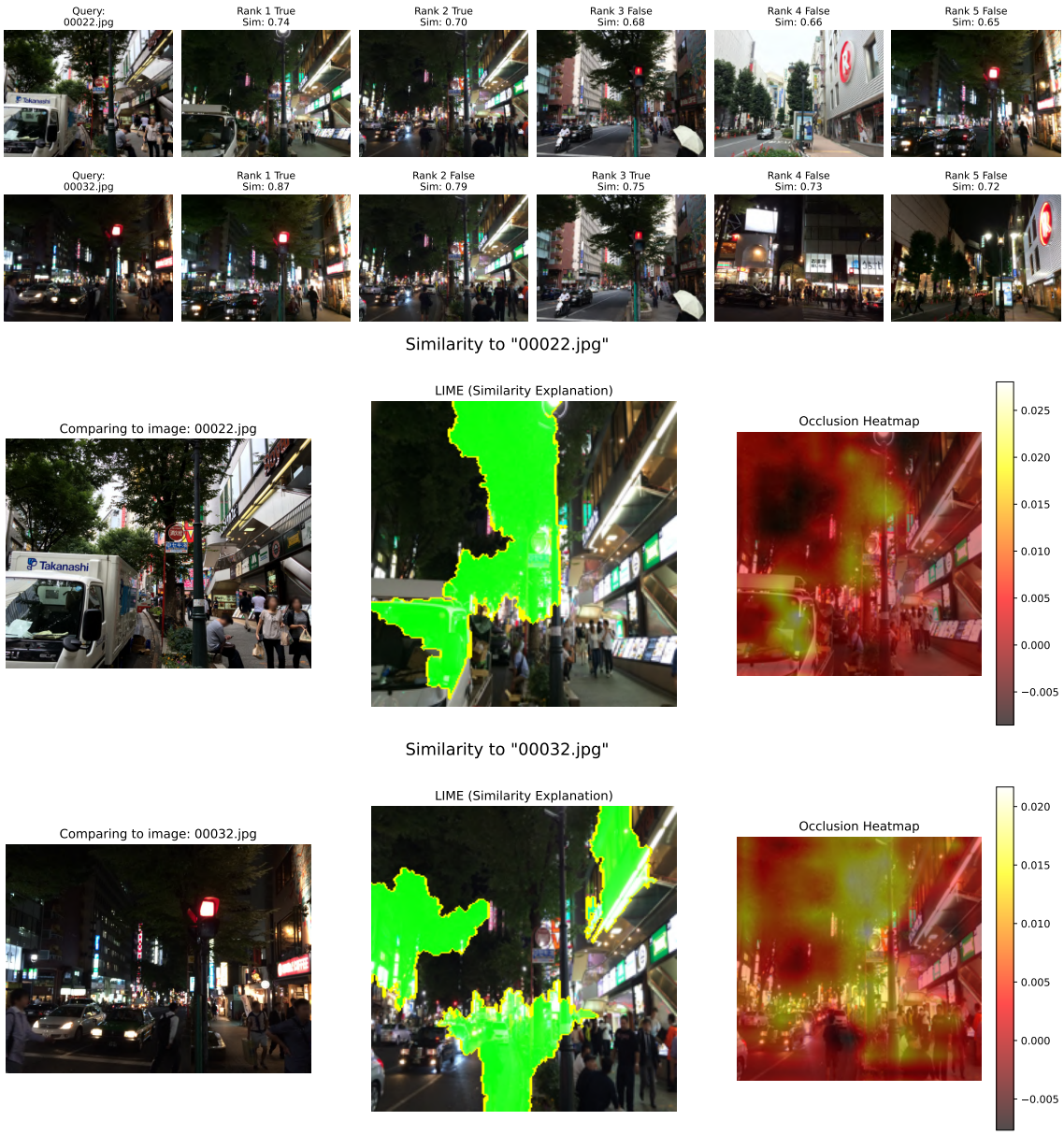


Figure 4.14: DINOv2 NetVLAD example and explainability.

SwinV2 + GeM As it happened with DINO, GeM seems to squish similarity towards 0.99 with frozen networks. As the metrics plots above already show, this model does not perform well. Most of the images are not in the same place and do not have the same illumination conditions. The explainability lets us see a good emphasis on big objects and architecture, but without a good embedding extraction.

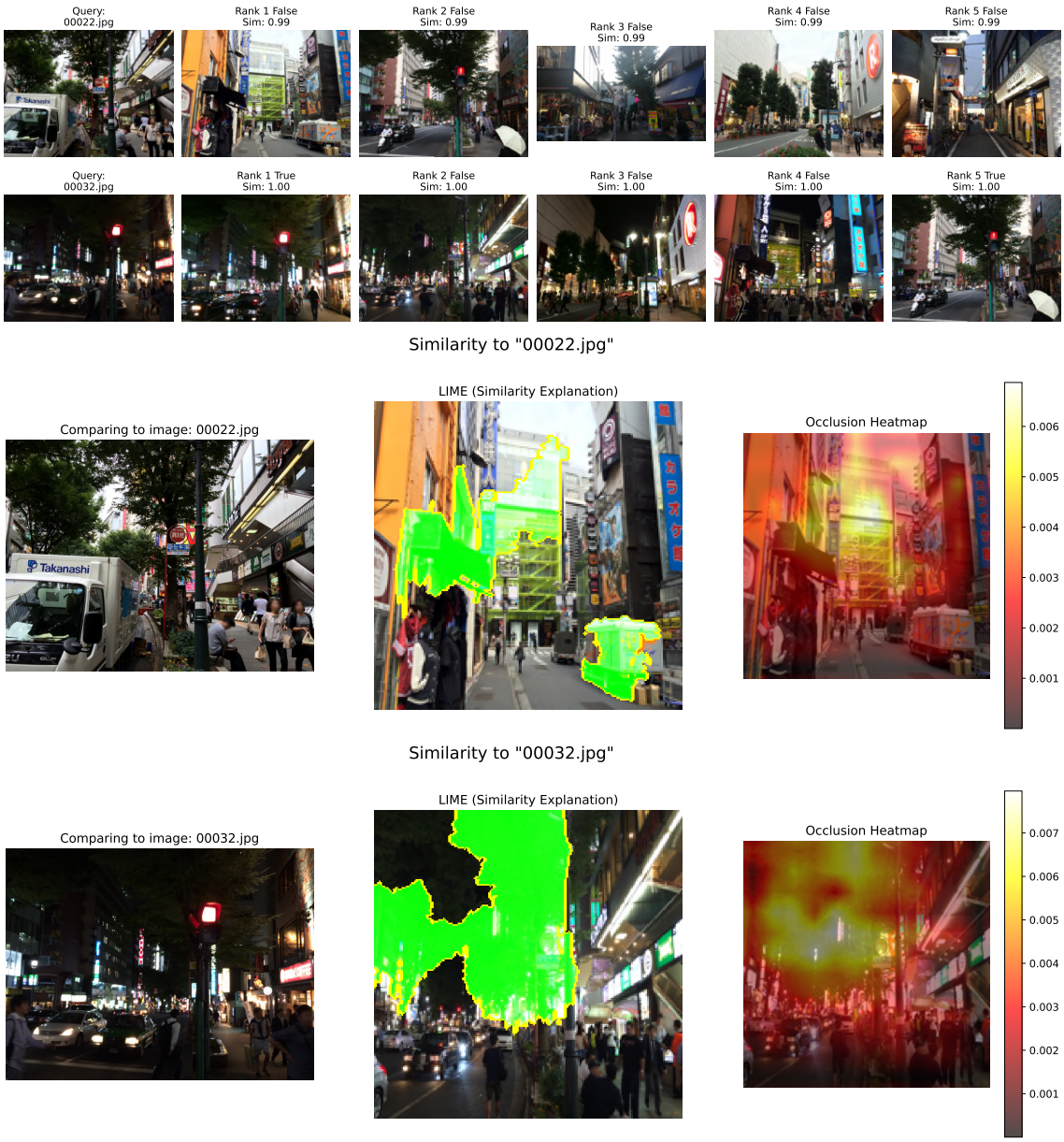


Figure 4.15: SwinV2 GeM example and explainability.

SwinV2 + NetVLAD the embeddings have now spread apart from 0.99, but the results are still bad and have the same problem as the previous model. While the main focus is still on big objects such as trees and light lamps, considerable importance is given to unimportant elements for place recognition, such as cars and people.

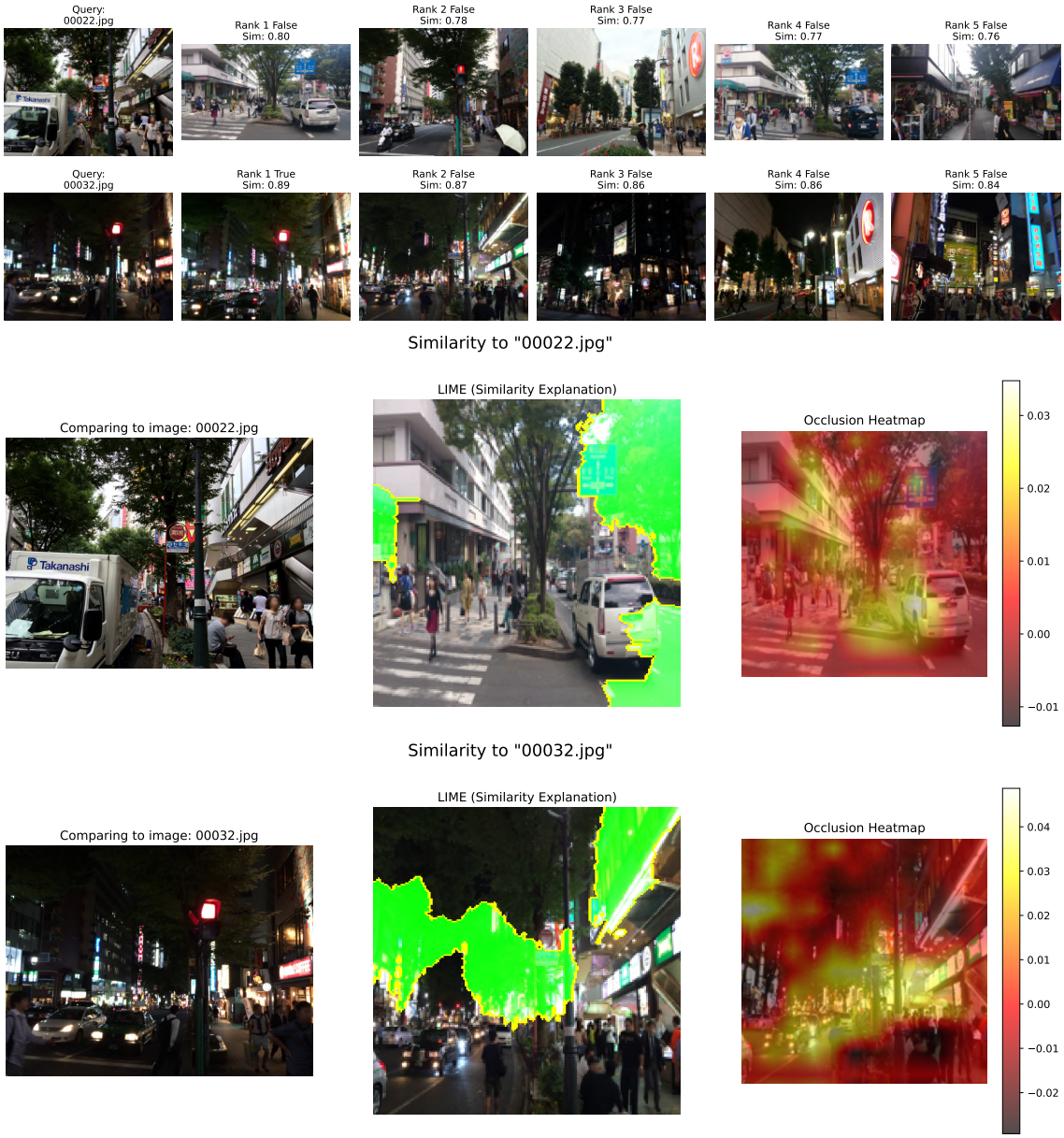
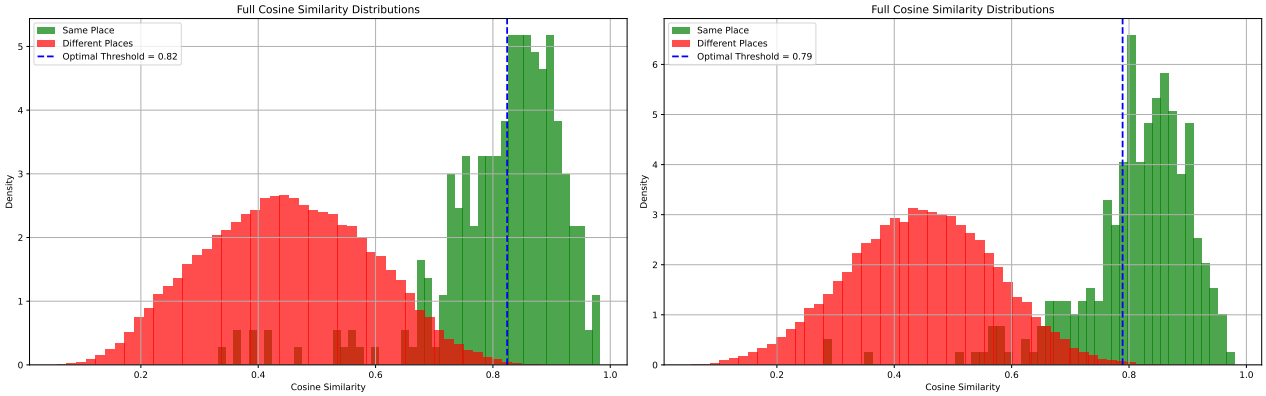


Figure 4.16: SwinV2 NetVLAD example and explainability.

Conclusions

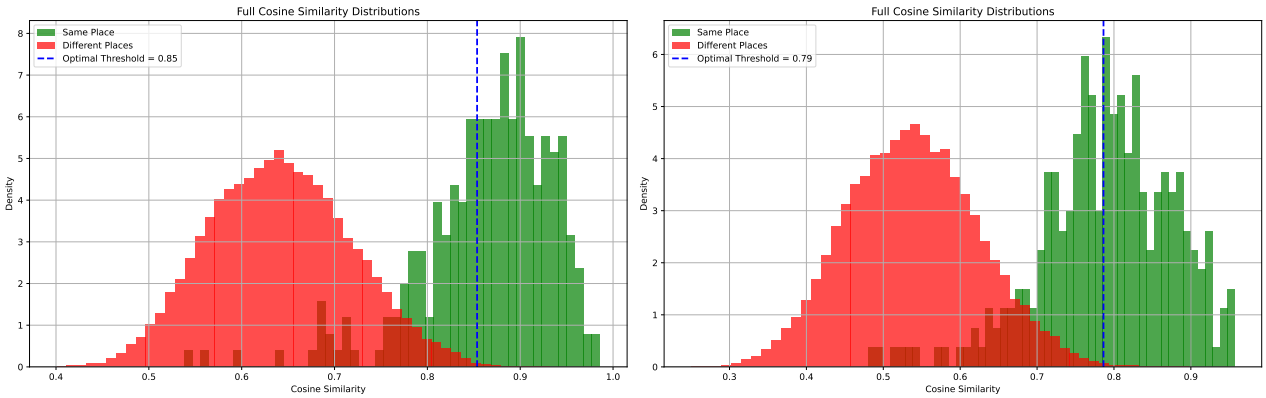
Before analyzing the final results, a few plots are needed to show the embedding distribution while using different architectures. An important metric to analyze how well the model works for a VPR task is how well we can fix a threshold that, given a reference image and a query image, calculates cosine similarity, allowing us to separate the correct same-place embeddings from the different-place ones. The plots below show with the same backbone how aggregators

distribute embeddings into space as a probability distribution, and a black dashed line is drawn as the optimal threshold that maximizes the F_1-score. To do this, for each embedding, all the cosine similarities are taken, and the distribution of the same place and different places is then normalized (as integral equals 1) and plotted. The final F_1-score calculated is reported in Table 4.1, as well as the Average Precision Score and the Recall at 100% Precision. A crucial fact to keep in account is that the distributions are normalized to integrate to 1 for probability reasons, but they are actually imbalanced: a small red area corresponds to a larger number of images than a green one. The red area represents a distribution of 78,678 images, while the green ones are just 564 images. While VGG19 (Fig. 4.17) and ResNet50 (Fig. 4.18) plots have standard values, DINO (Fig. 4.19) and Swin (Fig. 4.20) with GeM as the aggregator have values squashed near 0.99. While this seems quite strange from a conceptual point of view, all the images have high values of cosine similarity if the precision is kept high enough. Therefore, it is possible to see how there is still a widespread distribution, just concentrated on high numbers. In this case, GeM behaves like a max-pool, where dominant features oppress the other ones, leading to nearly identical vectors.



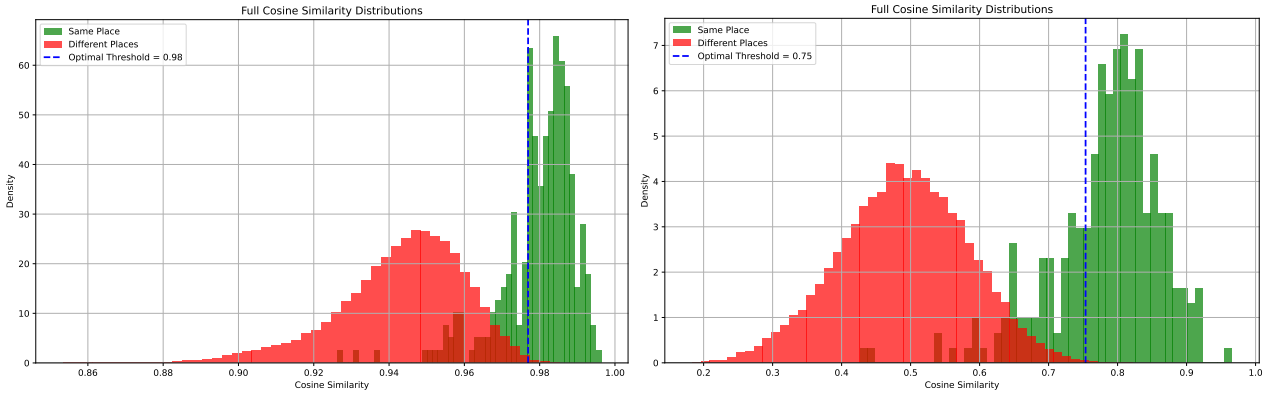
(a) VGG19 + GeM (b) VGG19 + NetVLAD

Figure 4.17: VGG19 encoding cosine similarity mean with GeM (left) and NetVLAD (right).



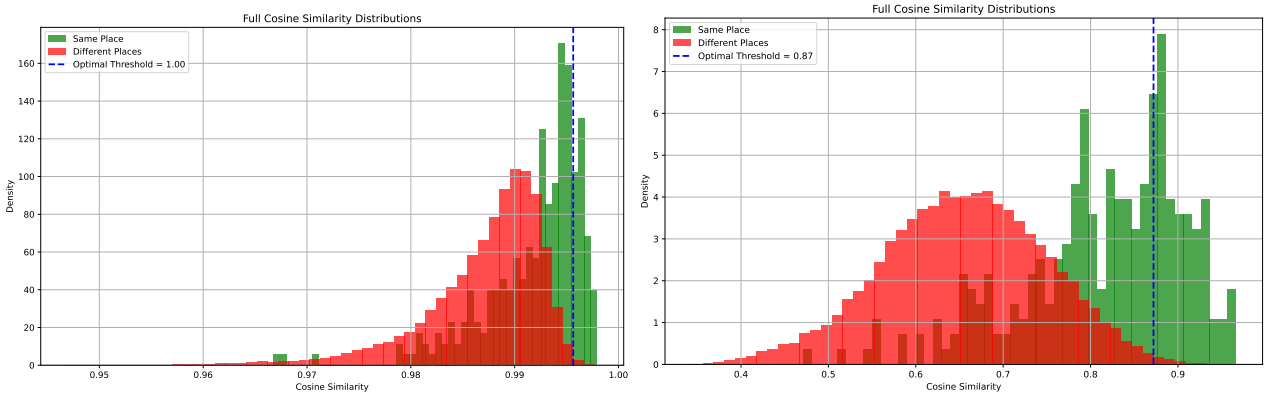
(a) ResNet50 + GeM (b) ResNet50 + NetVLAD

Figure 4.18: ResNet50 encoding cosine similarity mean with GeM (left) and NetVLAD (right).



(a) DINOv2 + GeM (b) DINOv2 + NetVLAD

Figure 4.19: DINOv2 encoding cosine similarity mean with GeM (left) and NetVLAD (right).



(a) SwinV2 + GeM (b) SwinV2 + NetVLAD

Figure 4.20: SwinV2 encoding cosine similarity mean with GeM (left) and NetVLAD (right).

The overall performance is reported in Table 4.1: the best scores for each column are written in bold, with DINOv2 with NetVLAD dominating for almost all the metrics, as well as for the degree of embeddings separation given by the F_1 metric. As for time taken for inference, ResNet50 and VGG19 are the fastest, as expected from CNNs. DINOv2 is three to five times slower but still competitive, with just 6 milliseconds of inference time. Given all the results above, it is clear how, while DINOv2 still dominates in terms of metrics and score, VGG19 with NetVLAD is an alternative that is viable. It is three times faster, the results are just four percentage points inferior to DINOv2's, and the F_1-score, as well as the Average Precision Score, are the second best overall, even surpassing DINOv2 with GeM. VGG19 with NetVLAD separates better the largest amount of descriptors, as it achieves the best Recall@100Precision score. Furthermore, from the qualitative analysis, VGG's ability to adapt to light changes while still having competitive scores is amazing. Obviously, given the right resources, fine-tuning DINOv2 would probably be able to achieve better results and adapt to drastic light changes. These experiments also confirm the dominance of DINOv2 as a foundation model, as SWIN transformers failed to meet expectations with their parameters frozen.

As concerns aggregators, NetVLAD resulted as the best for most architectures, even though ResNet50 seems to favor GeM, from what emerges from the results. The fact that NetVLAD surpasses GeM in most cases is expected, as it does not do a simple learnable pooling as GeM does, but learns for a number of codebooks that are optimal for place description.

Table 4.1: Performance comparison across architectures and aggregators on Tokyo 24/7. SR@K stands for Strict Recall@K, R@K for Recall@K, Time is the mean and standard deviation of inference time of the model once the image is loaded on the GPU in milliseconds, and F_1 is the maximum F_1-score achieved by trying various thresholds when applied to the distribution of embeddings. APS instead stands for Average Precision Score, and it is a metric used in classification to check how well the precision and recall are embodied by the model. R@100P indicates the Recall at 100% Precision.

Backbone	**Aggregator**	**R@1**	**R@5**	**SR@2**	**SR@10**	**Time (μ/σ)**	**F_1**	**APS**	**R@100P**
VGG19	GeM	90.8	96.5	68.8	88.3	2.17 / 0.01	69.1	72.7	28.4
VGG19	NetVLAD	91.8	96.1	74.8	88.7	2.18 / 0.01	73.8	78.0	**34.4**
ResNet50	GeM	87.2	95.7	53.9	81.5	**1.18 / 0.01**	67.3	70.2	16.3
ResNet50	NetVLAD	80.1	94.0	45.7	80.5	**1.18 / 0.01**	62.1	64.8	9.6
DINOv2	GeM	93.6	98.9	70.0	92.9	6.23 / 0.10	69.7	73.4	31.2
DINOv2	NetVLAD	**96.5**	**99.2**	**84.0**	**96.8**	6.38 / 0.07	**78.9**	**82.8**	33.0
SwinV2	GeM	54.2	72.0	8.2	30.1	8.89 / 0.04	24.6	15.4	1.8
SwinV2	NetVLAD	65.6	80.1	19.5	44.0	8.95 / 0.03	30.9	20.8	1.8

4.1.2 GSV-Cities

The second main battery of experiments was conducted on the GSV-Cities[54] dataset, using more than 500k images and 64k places. The primary goal was to evaluate and compare the performance of different model architectures in a Visual Place Recognition task with far more samples than before, as well as handling season and weather changes. The illumination changes, in this case, are the exception and not the norm.

Experiment Setup

The methodology is as follows: the more than 64k places are split into training and test sets at an 85:15 ratio. Each model architecture, composed of a backbone and an aggregator, is trained for 10 epochs using a triplet loss function with a margin of 0.3 and with an ADAM optimizer with a learning rate of 1×10^{-4}. The loss is then calculated between an anchor image, a positive image (from the same place at a different time), and a negative image (from a different place). When NetVLAD was used as an aggregator, the best results were achieved with a number of clusters equal to 96 (64 and 128 were also tested).

After training, feature descriptors are extracted for each image in the test set. During this inference stage, the process is timed to estimate the computational cost per image. Afterward, a cosine similarity matrix is generated by L2-normalizing the embeddings and multiplying the resulting matrix by its transpose. For quantitative evaluation, Recall@K and Average

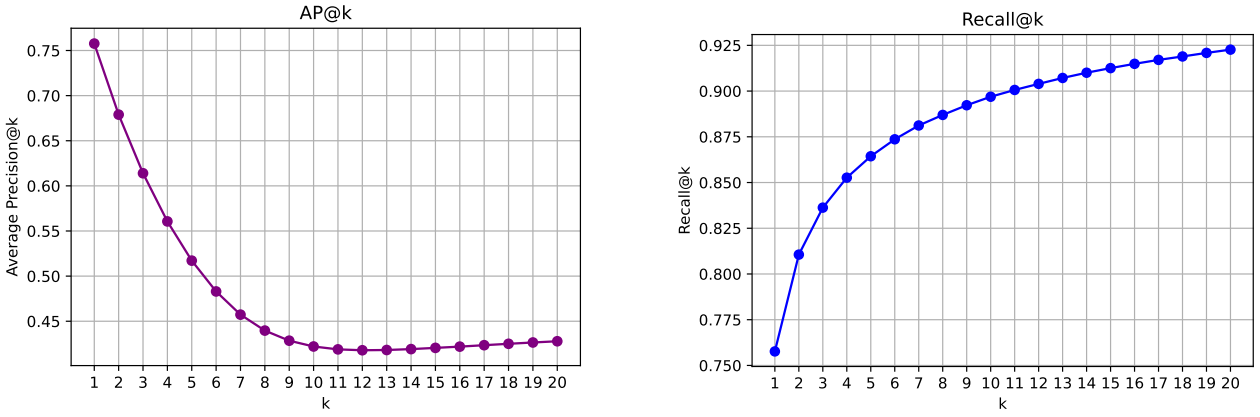
Performance@K metrics are calculated and plotted. A general plot of the cosine similarity value of correct and wrong images rescaled on density is also calculated to check the degree of separation between the two. Given the sizable number of test images, these statistics are done on a subset of relevant proportions randomly chosen (1/4 of the total), and the Average Precision Score and Recall at 100% Precision are calculated. Finally, for qualitative analysis, LIME and occlusion sensitivity algorithms are used to generate explainability maps. These maps highlight the image regions most critical to the model's descriptor-matching process.

Results

Given the results obtained in the previous case, SwinV2 transformers were not used as a backbone in this case, and while ResNet was tested, the performance was subpar, so the results are not shown. Still, DINOv2 and VGG19 results are shown below.

GeM With GeM as the aggregator, DINOv2 managed to reach high scores, as seen in Fig. 4.22, while VGG19 has an average score, far worse than DINO (see Fig. 4.21), but still better than the NetVLAD version (see Fig. 4.23). The optimal generalization on the about 10k different places of the test set with DINO was expected, as it is trained to be able to generalize in every situation. VGG, instead, is struggling to keep up with the generalization task, as even the R@1 is just around 75%.

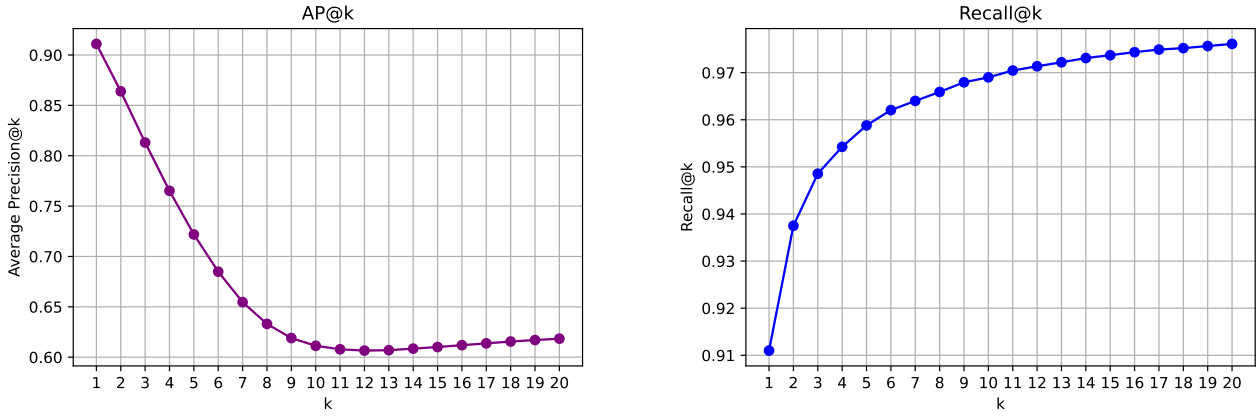
NetVLAD with NetVLAD as the aggregator, better results are reached by DINOv2 (see Fig. 4.24), outshining its previous version with GeM, reaching even scores above 95%. VGG19 performs worse than with GeM, just as ResNet did on the previous dataset.



(a) AP@K

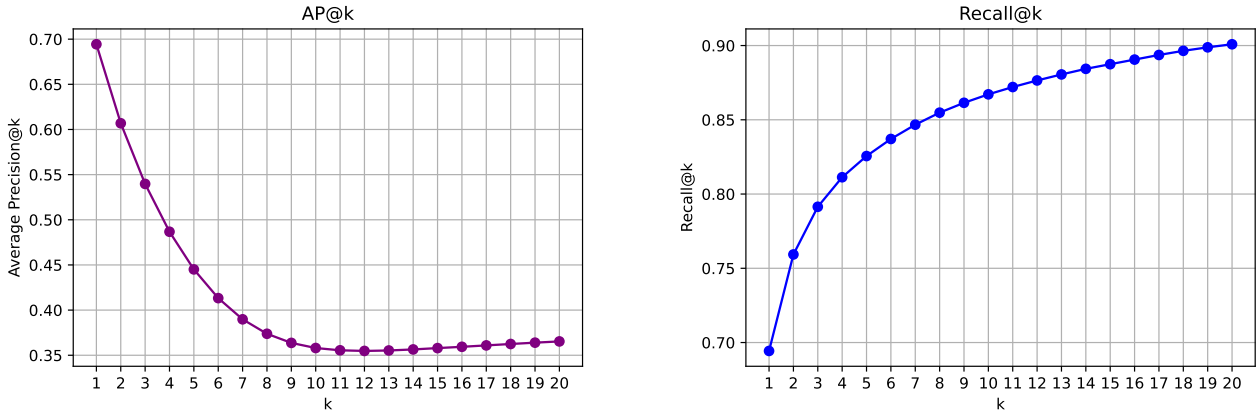
(b) Recall@K

Figure 4.21: VGG19 + GeM performance on GSV-Cities test set.



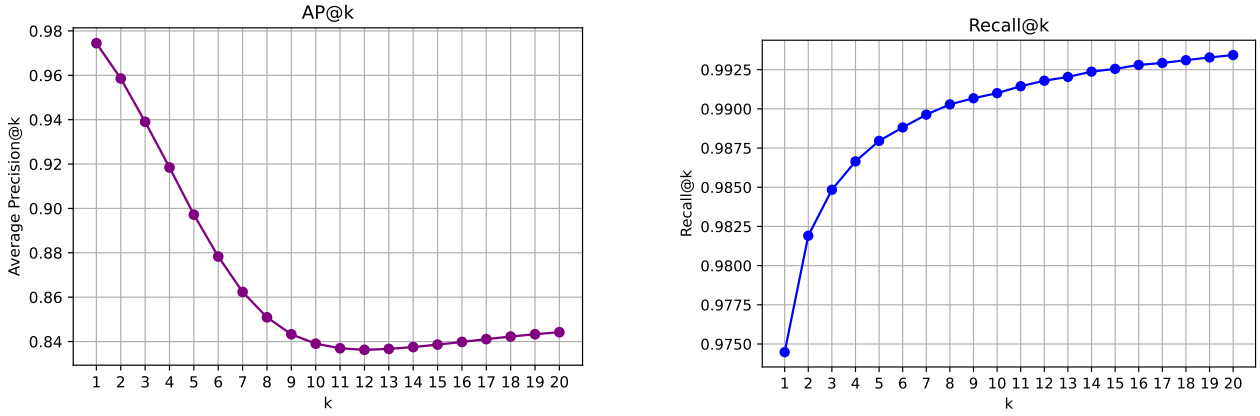
(a) AP@K (b) Recall@K

Figure 4.22: DINOv2 + GeM performance on GSV-Cities test set.



(a) AP@K (b) Recall@K

Figure 4.23: VGG19 + NetVLAD performance on GSV-Cities test set.



(a) AP@K (b) Recall@K

Figure 4.24: DINOv2 + NetVLAD performance on GSV-Cities test set.

Qualitative Results and Explainability For each of the possible architecture combinations, a qualitative study with explainability has been applied to give a better understanding of the influence on the decisions taken by the model. Two test images, one from Bangkok (a road) and one from Boston (a square with a red building), are selected, and the five top similar embeddings are retrieved and then shown, marking on top of each image both the cosine similarity and a boolean that indicates whether it is the same place or not.

VGG + GeM behaves quite well on the selected images, even though the gap between similarity scores is not as high as it could be. According to the explainability of the decisions taken, this architecture seems to focus on the trees or, in the first case, on the sky and on the building, while in the second case, it manages to recognize the picturesque red house as an important landmark.

Figure 4.25: VGG19 GeM examples and explainability.

VGG + NetVLAD does not perform very well overall, and in this particular case, it seems to focus on the wrong things. In the first batch, the focus is on the sky, a terrible choice for a VPR algorithm. In the second batch, a quite difficult image was recognized correctly, but all the others were wrong, so the result cannot be overall positive.

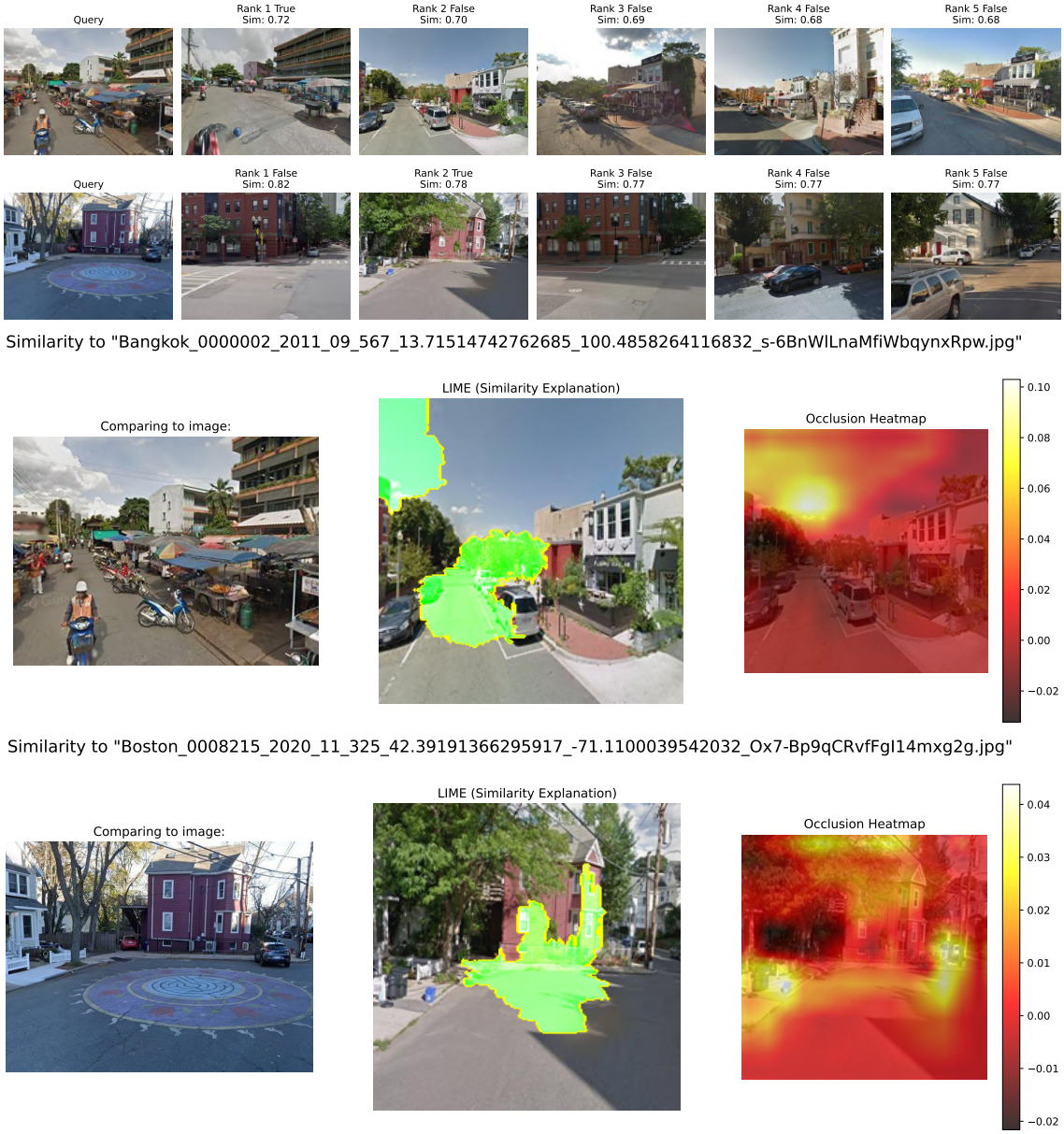


Figure 4.26: VGG19 NetVLAD example and explainability.

DINOv2 + GeM with GeM, the frequency of the right images is high, not as good as NetVLAD, but still quite high. The cosine similarities are again squashed towards 0.98, but this, as before, is not a problem in the final distribution. The explainability, this time, emphasizes buildings and special markings as the pattern on the square of the second query.

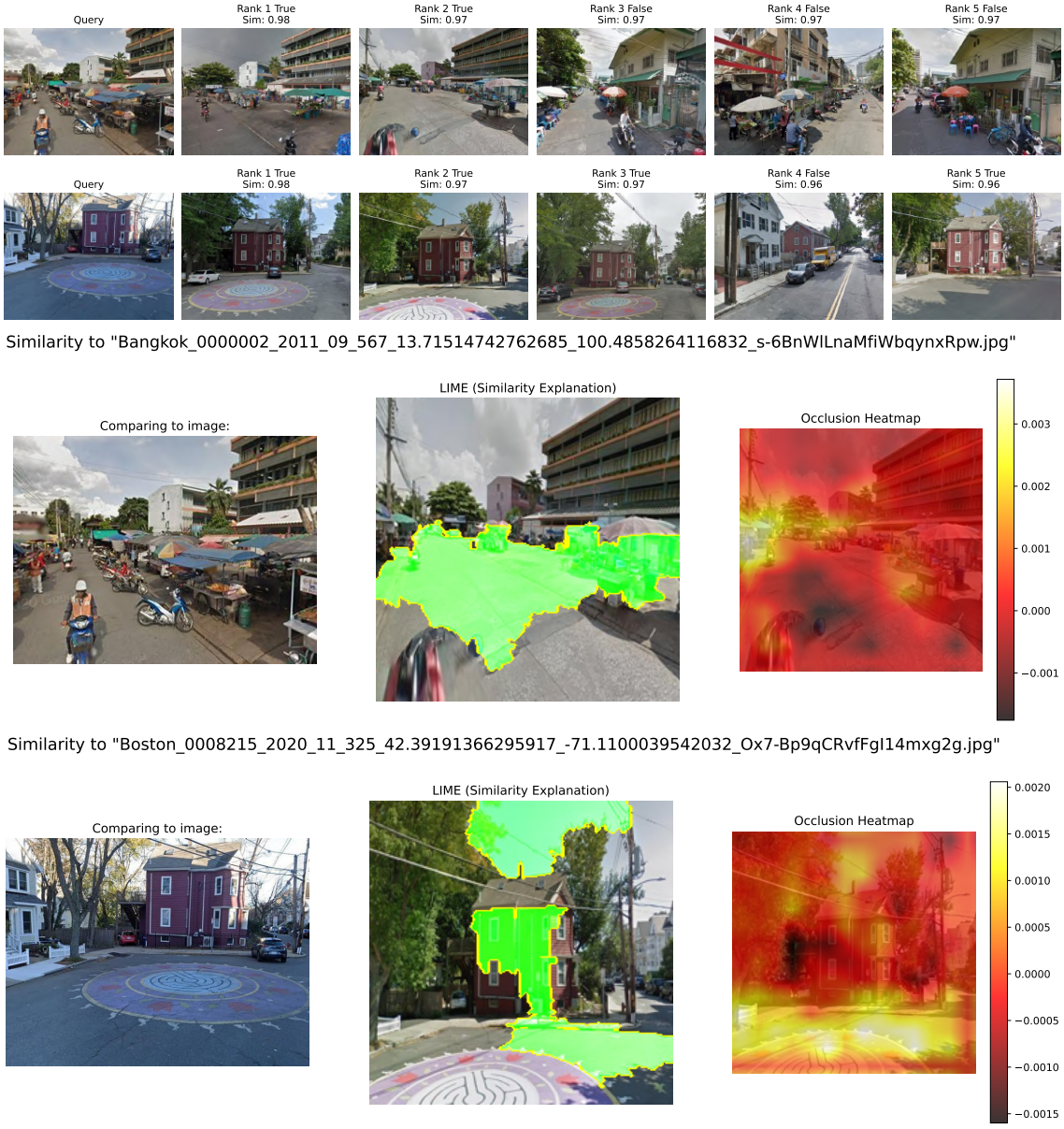


Figure 4.27: DINOv2 NetVLAD example and explainability.

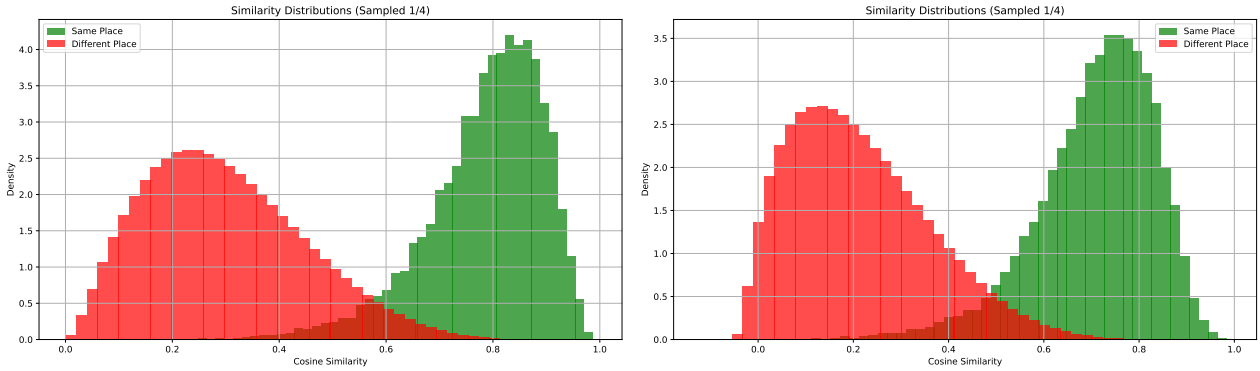
DINOv2 + NetVLAD In this case, DINOv2 manages to find the right place in the first five images. The explainability shows a certain focus both on bigger landmarks, buildings, and ample empty spaces, and on finer details, such as the power lines in the second query.

Figure 4.28: DINOv2 NetVLAD example and explainability.

Conclusions

Before analyzing the final results, a few plots are needed to show the embedding distribution while using different architectures. Just as it was analyzed in the previous case, the plots below show with the same backbone how aggregators distribute embeddings into space as a probability distribution. The process is similar to the one used on Tokyo's dataset, but uses a random but representative amount of images as a query (1/4 of the 79474 images present). In this case, the optimal F_1-score was not calculated, but the Average Precision Score is instead

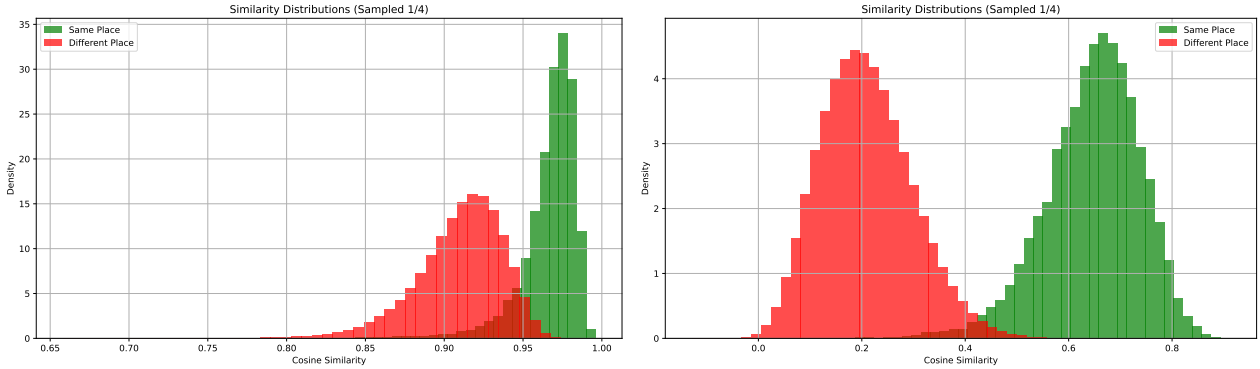
computed and reported in Table 4.2, along with the R@100P. A crucial fact to keep in account is that the distributions are normalized to integrate to 1 for probability reasons, but they are actually very imbalanced: a small red area corresponds to a far bigger number of images than a green one. The red area represents a distribution of 197,337,503 images, while the green one is just 21,275 images. While VGG (Fig. 4.39) plots have normal values with a large overlap area (reflected by the metrics calculated before), DINOv2 (Fig. 4.40) with GeM as the aggregator has values squashed near 0.98. This time, even when keeping the precision high enough, the overlap area is far greater than the one on NetVLAD.



(a) VGG19 + GeM

(b) VGG19 + NetVLAD

Figure 4.29: VGG19 encoding cosine similarity mean with GeM (left) and NetVLAD (right).



(a) DINOv2 + GeM

(b) DINOv2 + NetVLAD

Figure 4.30: DINOv2 encoding cosine similarity mean with GeM (left) and NetVLAD (right).

The overall performance is reported in Table 4.2: the best scores for each column are written in bold, with DINOv2 with NetVLAD dominating for almost all the metrics, as well as for the degree of embeddings separation given by the Average Performance Score. As for the time taken for inference, VGG19 is the fastest, as expected from a CNN. DINOv2 is about four times slower but still competitive, with just 6 milliseconds of inference time. Unlike what happened with Tokyo-24/7, VGG19 is not a competitive option against DINOv2. Not only were the samples far more, but they were also heterogeneous and very different in multiple settings. VGG19 did not manage to generalize well enough to achieve competitive results.

As for what concerns aggregators, GeM works well for the CNN, but with DINOv2, it fails to achieve good results on the Average Performance Score. This means that NetVLAD would be the preferred one in a SLAM case when corrections are possible: if strict loop closure without any correction of false positives is needed, then the one with the best, even if very low, Recall at 100% Precision is DINOv2 with GeM.

Table 4.2: Performance comparison across architectures and aggregators on GSV-Cities. R@K stands for Recall@K, AP@K stands for Average Precision (used in VPR) at k, Time is the mean and standard deviation of inference time of the model once the image is loaded on the GPU in milliseconds, and APS instead stands for Average Precision Score, while R@100P stands for Recall at 100% Precision. R@1 and AP@1 are equivalent.

Backbone	**Aggregator**	**R@1**	**R@5**	**AP@3**	**AP@5**	**Time (μ/σ)**	**APS**	**R@100P**
VGG19	GeM	75.8	86.4	61.4	51.7	**2.15 / 0.01**	18.4	0.0
VGG19	NetVLAD	69.4	82.6	54.0	44.5	**2.15 / 0.01**	11.9	< 0.1
DINOv2	GeM	91.1	95.9	81.3	72.1	6.97 / 0.07	35.9	**2.6**
DINOv2	NetVLAD	**97.4**	**98.8**	**93.9**	**89.7**	7.42 / 0.07	**70.7**	1.0

4.1.3 Generalization

Before passing to the indoor VPR section, a few comments on generalization are needed. The models trained on Tokyo-24/7, probably also because of the low number of images available, do not perform very well on the GSV-Cities test set, with a significant loss of performance. The exception is DINO, which not only performs well, but the version with NetVLAD aggregator trained on Tokyo-24/7 outperforms the native version with DINO and GeM on all metrics but the R@100P, as it is visible in Table 4.3. The models trained on GSV-Cities instead have excellent results on the Tokyo-24/7 dataset (Table 4.4). VGG19 has scores comparable to the ResNet trained specifically on Tokyo, while DINOv2 with the NetVLAD model outperforms the best DINOv2 model trained on Tokyo for all metrics, APS, and R@100P included. DINOv2 with GeM on GSV also has high scores, comparable to the one that VGG19 with NetVLAD trained on Tokyo has, but it does not seem to generalize as well as NetVLAD does. The idea to train models on big datasets and try to use them as general-purpose tools is not new and is the one that GSV's paper [54] authors thought of, but it is great to see it working on those two datasets. DINOv2 trained on a large enough dataset enables the creation of a truly general-purpose VPR model that may be trained once on enough data and then used on all the relevant occasions without needing retraining or fine-tuning.

Table 4.3: Performance comparison across architectures on GSV-Cities. R@K stands for Recall@K, AP@K stands for Average Precision (used in VPR) at k, APS instead stands for Average Precision Score, and R@100P for Recall at 100% Precision. R@1 and AP@1 are equivalent.

Backbone	Aggregator	Training	R@1	R@5	AP@5	APS	R@100P
VGG19	GeM	Tokyo	48.7	62.4	25.9	2.4	< 0.1
VGG19	GeM	GSV	75.8	86.4	51.7	18.4	0.0
VGG19	NetVLAD32	Tokyo	45.0	59.1	23.2	0.5	0.0
VGG19	NetVLAD96	GSV	69.4	82.6	44.5	12.0	< 0.1
DINOv2	GeM	Tokyo	90.2	95.3	71.6	36.4	2.3
DINOv2	GeM	GSV	91.1	95.9	72.1	35.9	**2.6**
DINOv2	NetVLAD32	Tokyo	92.6	96.7	76.4	47.6	0.6
DINOv2	NetVLAD96	GSV	**97.4**	**98.8**	**89.7**	**70.7**	1.0

Table 4.4: Performance comparison across architectures on Tokyo 24/7. SR@K stands for Strict Recall@K, R@K for Recall@K, and F_1 is the maximum F_1-score achieved by trying various thresholds when applied to the distribution of embeddings. APS instead stands for Average Precision Score and R@100P for Recall at 100% Precision.

Backbone	Aggregator	Training	R@1	R@5	SR@2	SR@10	F_1	APS	R@100P
VGG19	GeM	GSV	85.1	94.3	58.9	79.8	67.8	69.8	19.5
VGG19	GeM	Tokyo	90.8	96.5	68.8	88.3	69.1	72.7	28.4
VGG19	NetVLAD96	GSV	73.4	81.6	33.0	58.2	49.7	46.0	9.9
VGG19	NetVLAD32	Tokyo	91.8	96.1	74.8	88.7	73.8	78.0	34.4
DINOv2	GeM	GSV	93.3	98.9	69.1	91.8	69.3	72.8	30.1
DINOv2	GeM	Tokyo	93.6	98.9	70.0	92.9	69.7	73.4	31.2
DINOv2	NetVLAD96	GSV	**96.8**	**99.3**	**89.0**	**97.2**	**82.1**	**87.5**	**45.7**
DINOv2	NetVLAD32	Tokyo	96.5	99.2	84.0	96.8	78.9	82.8	33.0

Indoor

After seeing the success of these architectures on Outdoor VPR, a few tests were run in Indoor VPR scenarios. Indoor VPR is an often overlooked problem, and there are not many datasets adapted for this task.

4.1.4 OpenLoris-Scene Test

The OpenLoris-Scene [58] is a dataset containing the outputs of sensors mounted on a robot that moves along a specific path in an indoor space. It did not come with specific places

divided, so a few of them were mined with a tool that takes into account the position and orientation of the robot at each step and tries to find images that represent "places". This means that not only must the images be far enough from each other, but also that the orientation of the robot when taking the picture must not overlap as much as possible. In this way, from the office dataset, only five pairs of different places were mined. This was considered an acceptable amount, as the scope of this test was to check if the DINOv2 model can perform optimally in indoor environments without fine-tuning.
Taken the DINOv2 model with NetVLAD trained on Tokyo-24/7, the Recall@1 is 100% on these few images. No place was mismatched, and all the correct places have a cosine similarity above 0.80, while the different places always have a cosine similarity below 0.80. This means that, in this small example, the APS and R@100P are both 100%, and that a threshold can be chosen to completely divide the same place and different place images. This experiment is not enough to draw a meaningful conclusion, but this was just a test to understand if DINOv2 could perform well in indoor environments.

Figure 4.31: A few queries of OpenLoris-Scene office with their embeddings extracted and ranked by the DINOv2 with NetVLAD model. Every image retrieved at Rank 1 is the correct one.

4.1.5 NYC-Indoor-VPR

Given the preliminary results obtained above, a more challenging problem was tackled with the NYC-Indoor-VPR[55] dataset. This dataset is particularly challenging for three main reasons: first of all, the images are taken with a 360° camera and are thus highly distorted; secondly, the place labeling is not always accurate, or at least it does not correspond to the definition of place that we used until now, as it considers as different places also images that have the same subjects but are a few meters apart; thirdly the dataset is anonymized by removing from the images the people and leaving blank pixels in their place. The results given these conditions

are suboptimal, but this was expected, as the authors of the paper have achieved very low recall rates. The process used by them is more advanced, so their scores are also higher than the ones that will be reported below.

Experiment Setup

The dataset this time was already split into train, validation, and test. Moreover, a CSV file containing information on each image, with its spatial position, file path, and place, was provided. In this case, "hard mining" of triplets was used, as instead of sampling a random positive and negative for each anchor, negatives are images near spatially but not belonging to the same place (see Fig. 4.32). Afterward, the model plus aggregator is trained for 10 epochs using a triplet loss function with a margin of 0.3 and with an ADAM optimizer with a learning rate of 1×10^{-4}. When NetVLAD was used as an aggregator, the best results were achieved with a number of clusters equal to 96 (32, 64, and 128 were also tested).

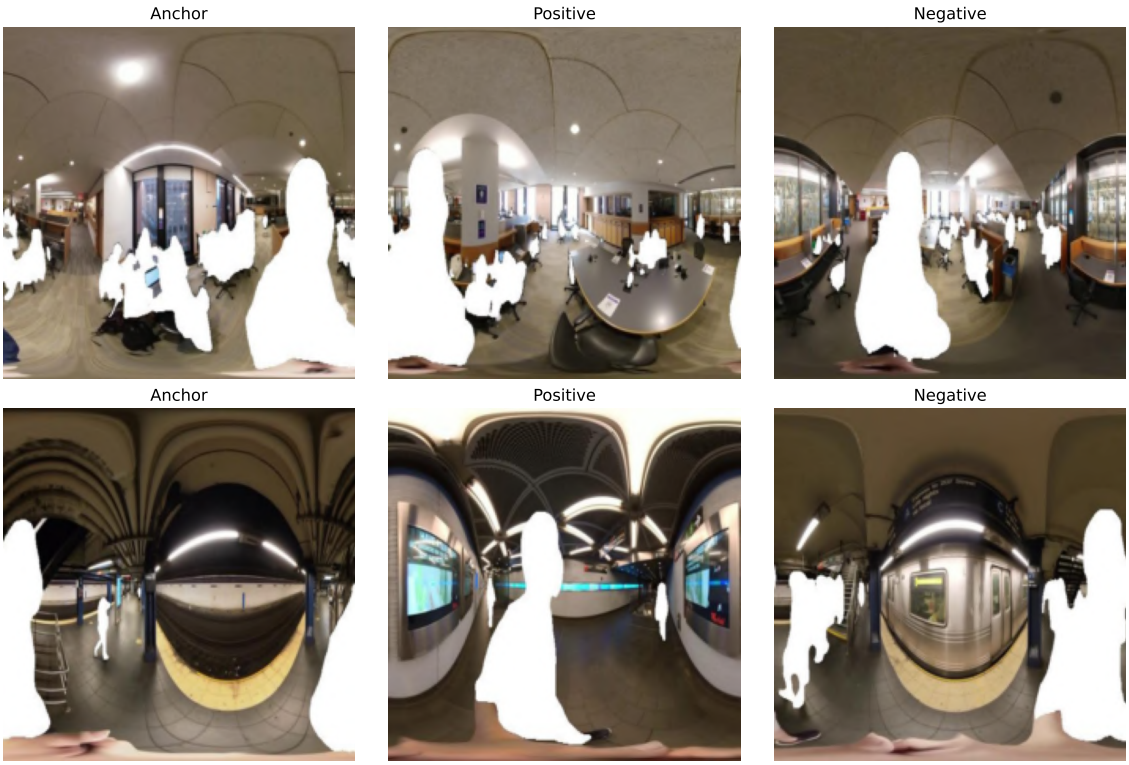


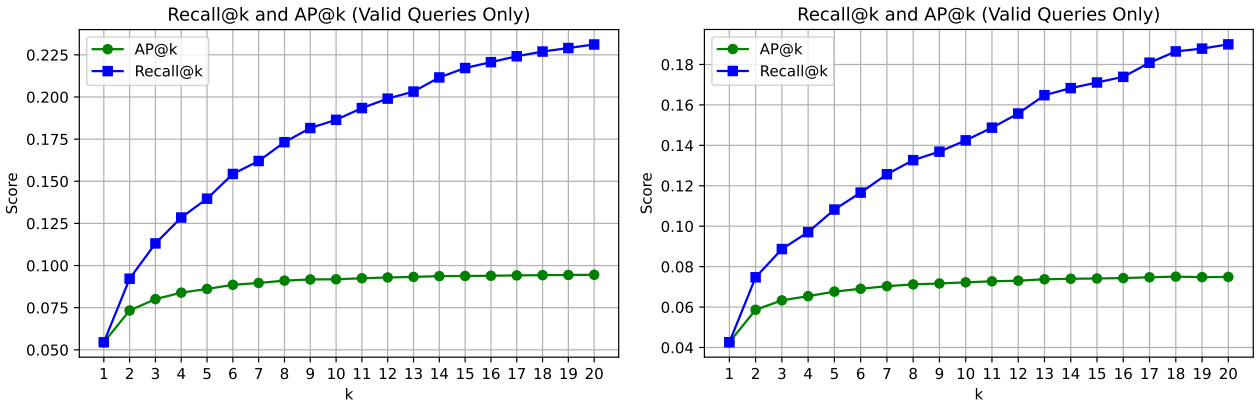
Figure 4.32: An example of hard triplets mined for training from the NYC-Indoor-VPR dataset.

After training, the validation set is cleaned for places with singleton images, and feature descriptors are extracted for each valid image left. A cosine similarity matrix is generated by L2-normalizing the embeddings and multiplying the resulting matrix by its transpose. For quantitative evaluation, Recall@K and Average Performance@K metrics are calculated and plotted. A general plot of the cosine similarity value of correct and wrong images rescaled on density is also calculated to check the degree of separation between the two. F_1 best score,

Average Precision Score, and Recall at 100% Precision are calculated. Finally, for qualitative analysis, LIME and occlusion sensitivity algorithms are used to generate explainability maps.

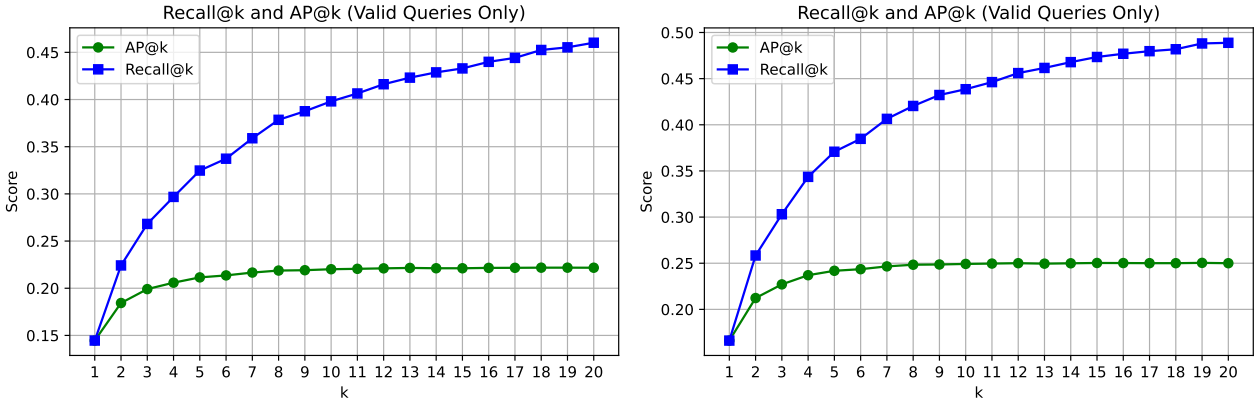
Results

As it is possible to see from the figure above (Fig. 4.32), it is very challenging for a human as well to distinguish the same place images from the examples, both because of the distortion and because of the noise from anonymization. The results are indeed low in general, having scores around 5 to 10% on Recall@1. VGG19 performs worse than DINOv2, while GeM improves VGG19 performance against NetVLAD but cannot reach the results achieved by NetVLAD with DINOv2 once again, as it is possible to see from Fig. 4.33 and Fig. 4.34



(a) GeM (b) NetVLAD

Figure 4.33: AP@K and Recall@K of VGG19 with GeM (left) and NetVLAD (right) on NYC-Indoor-VPR validation set.



(a) GeM (b) NetVLAD

Figure 4.34: AP@K and Recall@K of DINOv2 with GeM (left) and NetVLAD (right) on NYC-Indoor-VPR validation set.

Qualitative results and Explainability For each of the possible architecture combinations, a qualitative study with explainability has been applied to give a better understanding

of the influence on the decisions taken by the model. One test image is selected, and the 5 top similar embeddings are retrieved and then shown, marking on top of each image both the cosine similarity and an 'X' mark or "OK" word that indicates whether it is the same place or not.

VGG + GeM does not recall even one right image. At least the images recalled are very similar, but due to the strict place marking, they are not the same. According to the explainability of the decisions taken, excessive focus is given to the blank spaces that represent anonymized people. This is a problem and probably also one of the leading reasons for the low scores the model achieved.

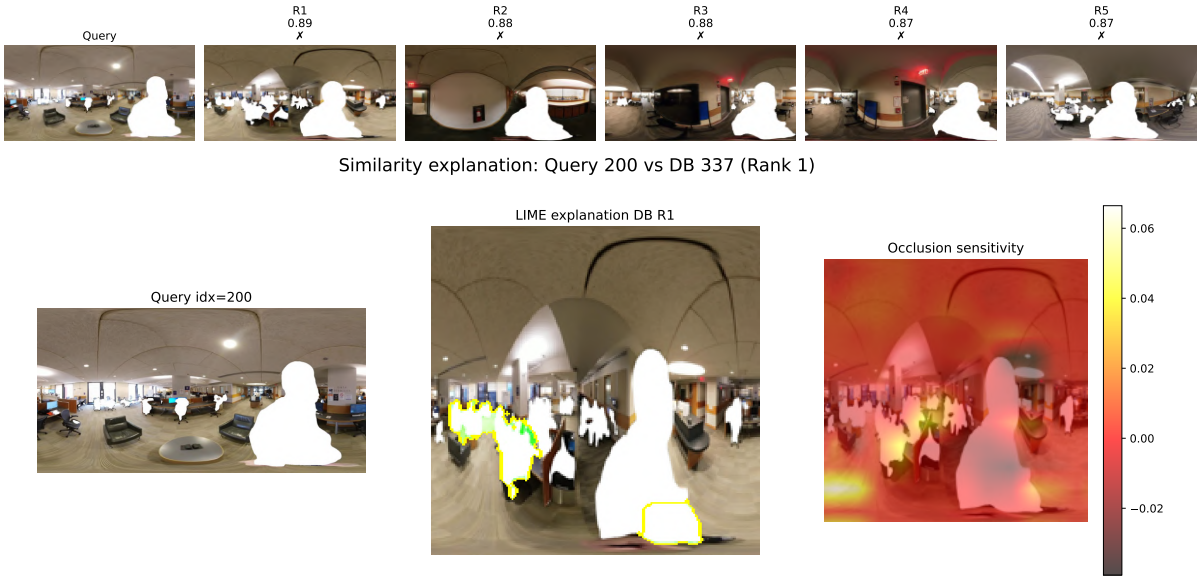


Figure 4.35: VGG19 GeM example and explainability.

VGG + NetVLAD as it is possible to see in this section's Conclusions, NetVLAD collapses images into embeddings very near to 1.0, 0.8, and a few more values. This still did not find the same place, and the focus still seems to be on people at a specific point of the ceiling, without taking anything else into account.

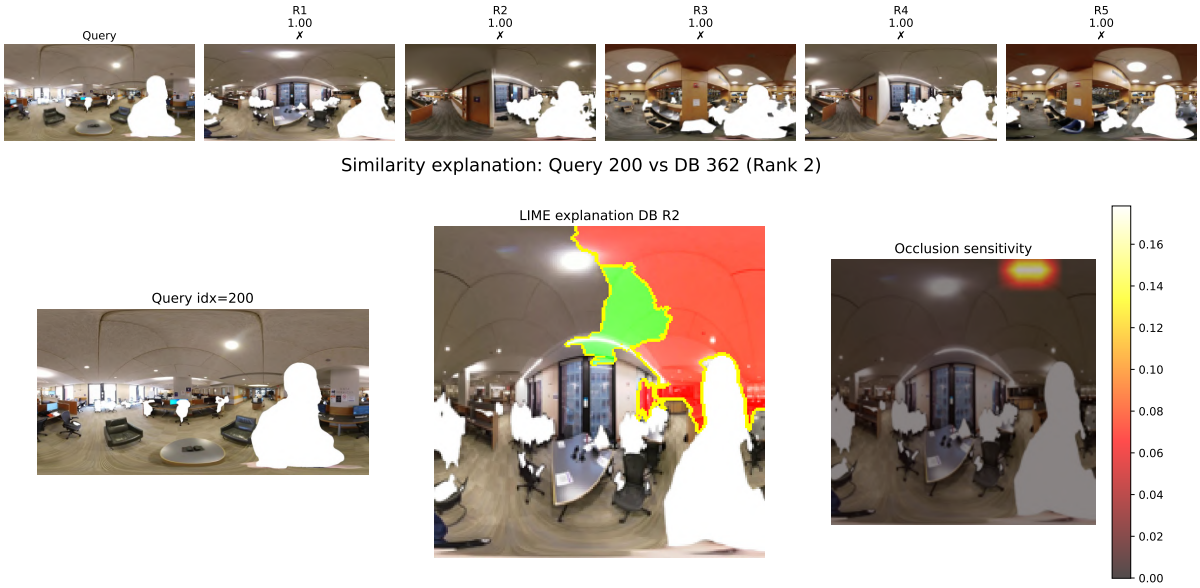


Figure 4.36: VGG19 NetVLAD example and explainability.

DINOv2 + GeM also does not recall even one right image. On the bright side, from the explainability, it seems to focus on places instead of people: DINOv2, as usual, generalizes better.

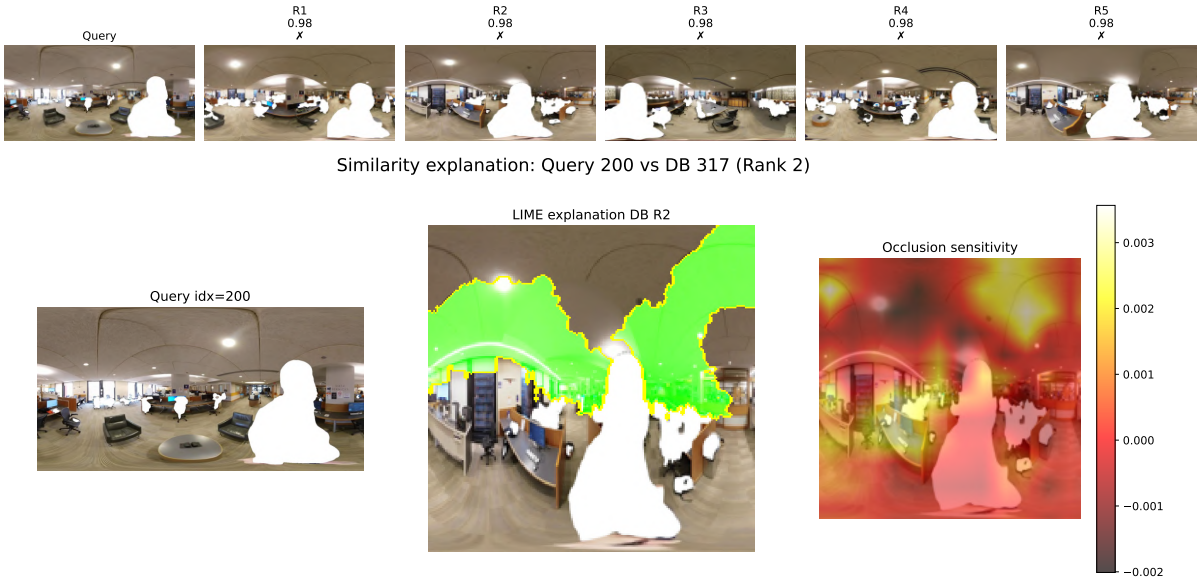


Figure 4.37: DINOv2 GeM example and explainability.

DINOv2 + NetVLAD is the first one to get at least one image right. The focus seems similar to that of GeM but with less emphasis on the ceiling and more on the central part of the image.

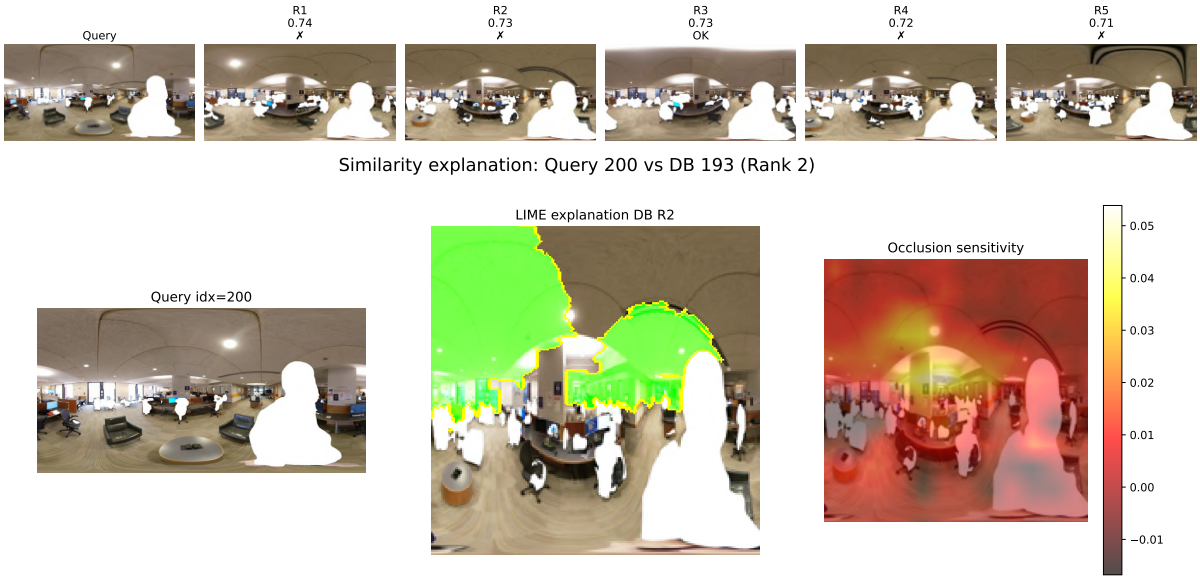
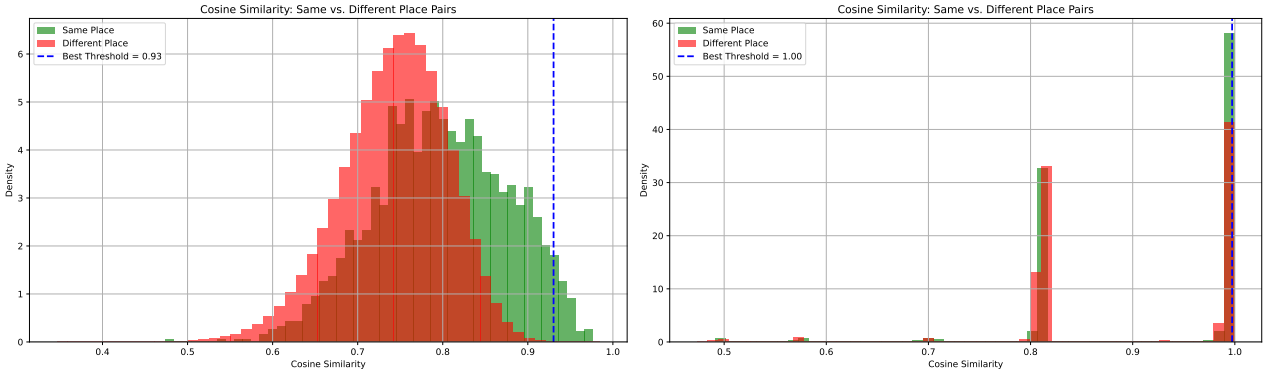


Figure 4.38: DINOv2 NetVLAD example and explainability.

Conclusions

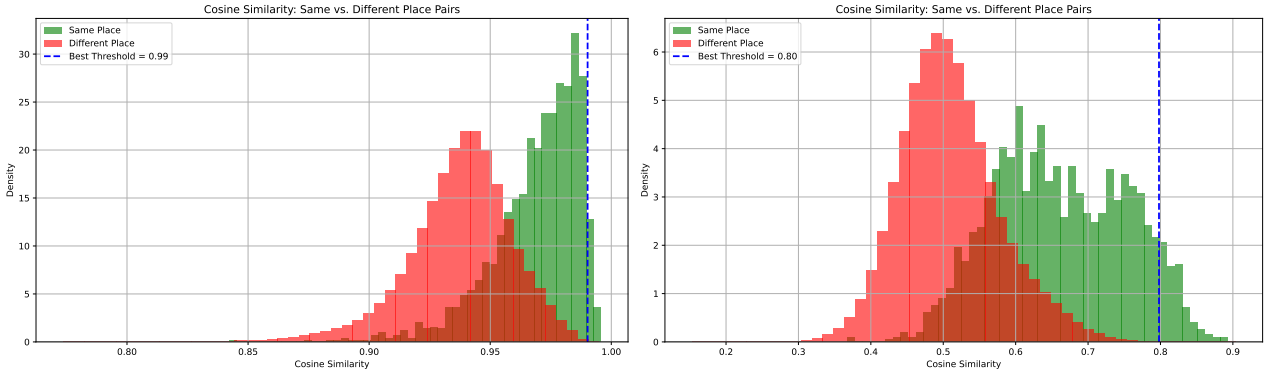
As usual, before the final results, here are a few plots of the distributions of cosine similarity between embeddings calculated between the DB and query images. As the models did not perform well, we expect a large overlap area on all the images, especially considering that the red images are far more than the green ones, even if they were normalized (14,918,738 to 1,888). As expected, the best F_1-score is pushed on the right, so much so that the results reported in Table 4.5 are low values. The behavior of VGG with NetVLAD is peculiar: it collapses embedding similarity in a few specific points, such as 1.0 or 0.8.



(a) VGG19 + GeM

(b) VGG19 + NetVLAD

Figure 4.39: VGG19 encoding cosine similarity mean with GeM (left) and NetVLAD (right).



(a) DINOv2 + GeM

(b) DINOv2 + NetVLAD

Figure 4.40: DINOv2 encoding cosine similarity mean with GeM (left) and NetVLAD (right).

The overall results cannot be called good. Some of them give hope for a future breakthrough, as the fact that DINOv2 and NetVLAD can still achieve results that are far above the others. Still, it is clear that 360° images, as well as anonymized ones, are a new field of study that will need new architectures to tackle the challenges they offer. In Table 4.5, a general summary is reported, where DINOv2 dominance is once again emphasized. VGG19 with NetVLAD this time not only failed on the recall metrics but also on the cosine separation metrics, such as F_1-score and APS. GeM with DINOv2 is the only model that managed to classify 0.1% of the right images without having false positives. The result is abysmal, but it is better than the 0.0% that all the other models achieved. As usual, VGG is about three times faster than DINOv2.

Table 4.5: Performance comparison across architectures and aggregators on GSV-Cities. R@K stands for Recall@K, AP@K stands for Average Precision (used in VPR) at k, Time is the mean and standard deviation of inference time of the model once the image is loaded on the GPU in milliseconds, F_1 is the best F_1-score calculated on various threshold, APS for Average Precision Score, and R@100P for Recall at 100% Precision. R@1 and AP@1 are equivalent.

Backbone	**Aggregator**	**R@1**	**R@5**	**AP@5**	**AP@10**	**Time (μ/σ)**	**F_1**	**APS**	**R@100P**
VGG19	GeM	5.5	14.0	8.6	9.2	**1.4 / 0.1**	3.8	0.5	0.0
VGG19	NetVLAD	4.3	10.8	6.8	7.2	1.7 / 0.1	0.1	0.2	0.0
DINOv2	GeM	14.5	32.5	21.2	22.0	4.3 / 0.1	4.9	1.2	**0.1**
DINOv2	NetVLAD	**16.6**	**37.1**	**24.2**	**24.9**	4.5 / 0.1	**7.0**	**1.5**	0.0

4.2 Online Learning

In the online learning case, the aim was to improve the latest model available: VIPeR [48]. VIPeR's pipeline is reported in Figure 4.41: the main improvements it implements compared to AirLoop are a multi-stage memory bank, new losses, and adaptive triplets mining.

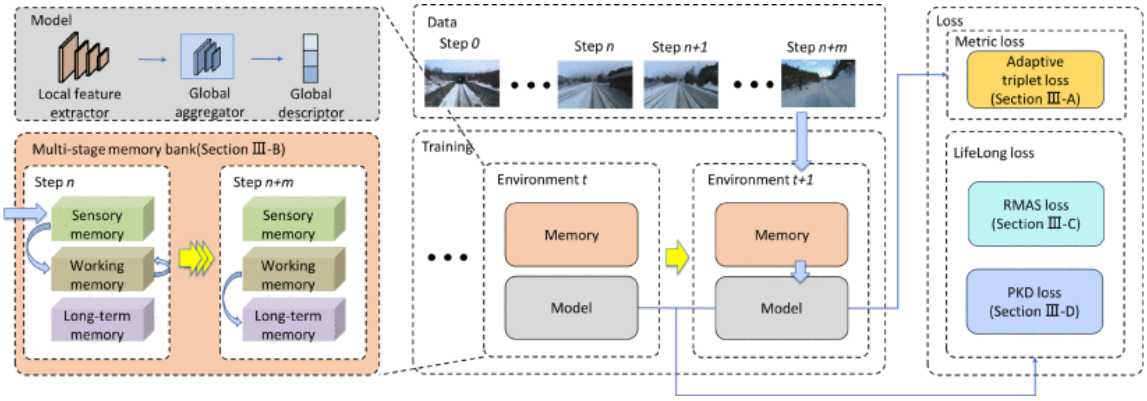


Figure 4.41: VIPeR accompanies the place recognition model with a multi-stage memory bank for rehearsal, adaptive mining, relational memory aware synapses (RMAS), and probabilistic knowledge distillation (PKD) for regularization. [48]

Both the adaptive triplet mining and the new losses are well thought out and use state-of-the-art techniques, which are difficult to improve with limited time and resources. The memory bank, instead, uses a probabilistic approach in which all images are substituted based on a probability given by a hyperparameter chosen before starting the training of the model. The base memory functions promote from sensory memory to current (working) with a random factor and from current (working) to eternal (long-term) with another random factor of replacement of λ, set to 0.5 in these experiments to balance retention and adaptation. Thus, two main approaches were tried to improve the model: a diversity-based approach selects which images to maintain based on their diversity according to the internal relevance metric; a global descriptor-based approach tries to keep in memory images that have very different global descriptors. The idea in both cases is to allow the memory to store not just random images but rather to keep images relevant to fully describe the environment seen by the model. This means retaining the embeddings that cover the largest amount of latent space, which should help the neural network generalize better. For further details on the code, the algorithms are reported in Appendix A.

Experiment Setup

The VIPeR model has a public GitHub repository; the baseline code is derived from AirLoop [47], with modified sections for memory, lifelong losses, and triplet sampling. All the tests were conducted by editing the functions for storing images in the working and eternal memory in the memory file. The losses file was also modified to pass the global descriptor to memory when needed. Because of time and memory constraints, all tests were made on the Nordland dataset with VGG as a backbone. DINOv2 was not tested, as it is not retrained and the

intelligent method would hel only the aggregator.
The Nordland dataset contains images for each of the four seasons, and the models are trained and tested on each of them, starting from *spring* and ending with *winter*. In the joint training case, the seasons are mixed to check the overall performance of the model on the dataset, while in the sequential training case, starting from spring to winter, the model sees all the images of each season sequentially. Joint training is used to evaluate the upper limit of the method, while sequential training represents a real-life example where pictures are taken following a linear temporal sequence. At the end of the training, the model weights are saved and then used to evaluate the performance as Recall@100Precision and Average Precision Score for each season. The following section reports the average performance across all seasons. All the models were trained with the same seed for three epochs, which is the equivalent of doing three laps across all seasons in this case.

Results

First of all, joint training was used to test the baseline and the edited versions of VIPeR. As reported in Table 4.6, the results of the first epoch metrics are similar and have no meaningful gap between the base model performance and the modified one. On the third epoch, both diversity and global descriptor memories show a performance that is two percentage points better than the baseline version. With GeM as the aggregator, the diversity increases the performance with respect to the baseline and global descriptor one, while with NetVLAD, the global descriptor-based memory excels.

Table 4.6: Performance comparison of VIPeR joint training with VGG as a backbone and different aggregators on Nordland with the base VIPeR version, with diversity memory, and with global descriptor memory. E1 and E3 stand for epochs 1 and 3, respectively, while R@100P stands for Recall@100Precision, and APS stands for Average Precision Score. Both R@100P and APS are the averages of the model performance across the four seasons in Nordland.

Version	**Aggregator**	**E1 R@100P**	**E1 APS**	**E3 R@100P**	**E3 APS**
Base	GeM	58.6	72.9	60.3	74.9
Diversity	GeM	60.2	74.3	62.8	77.2
Global Descriptor	GeM	58.0	72.5	61.5	76.0
Base	NetVLAD	69.4	82.8	71.7	84.5
Diversity	NetVLAD	69.1	82.7	72.7	85.3
Global Descriptor	NetVLAD	**69.8**	**83.0**	**73.5**	**86.0**

After the joint training test, a sequential training that simulates a real-life environment situation was employed. The models used for evaluation are trained sequentially on all four seasons, with winter being the last season seen. The results are reported in Table 4.7, in which, in the first epoch, the base NetVLAD model chose the images well, as it dominates with the best results. In the third epoch, both with GeM and with NetVLAD, the edited

memories surpass the baseline one by about three percentage points both in mean R@100P and in mean APS.

Table 4.7: Performance comparison of full cycle VIPeR sequential training with VGG as a backbone and different aggregators on Nordland with the base VIPeR version, with diversity memory, and with global descriptor memory. E1 and E3 stand for epochs 1 and 3, respectively, while R@100P stands for Recall@100Precision, and APS stands for Average Precision Score. Both R@100P and APS are the averages of the model performance across the four seasons in Nordland.

Version	**Aggregator**	**E1 R@100P**	**E1 APS**	**E3 R@100P**	**E3 APS**
Base	GeM	62.6	76.9	66.2	80.5
Diversity	GeM	65.0	78.6	67.2	80.8
Global Descriptor	GeM	65.4	78.8	67.8	81.2
Base	NetVLAD	**64.6**	78.5	65.0	79.0
Diversity	NetVLAD	62.1	76.2	68.9	81.9
Global Descriptor	NetVLAD	64.4	**78.6**	**69.6**	**82.5**

Conclusions

While the improvement on the first epoch is limited, on multiple epochs, the edited memory versions outperform the base random memory: global descriptor-based memory and NetVLAD yield the best results on the third epoch, both in the joint and in the sequential training case. The improvement brought by the edited memory is from two to three percentage points; it is significant as it is model-agnostic, and it generalizes with all backbones and aggregators tested. To further improve VIPeR, apart from structural modification, such as the memory one tested above, a more advanced aggregator and backbone can be plugged in to obtain higher performance scores. A single test was also made with ConvAP [54], a state-of-the-art aggregator, and it achieved even better results. This was not comprehensively tested in this thesis, as it is not really a structural improvement to the model, but just using more advanced components in a framework that already works.
On real robotic platforms, where it is not possible to redo the training on the whole dataset every time, having a good performance increase with only the data in memory and the current data, trained as the robot drives, for only three epochs, can be a good starting point and a good compromise to be able to apply these methods on real-life scenarios.

Chapter 5

Conclusions

This thesis investigated the domain of Visual Place Recognition (VPR) with three primary objectives: to conduct a comprehensive evaluation of foundational VPR architectures, to propose and validate metrics that take into account embedding separation when assessing model performance, and to improve the state-of-the-art continual learning framework for VPR.

5.1 Summary of Contributions

This thesis addressed the three core problems outlined in the Introduction, obtaining the following contributions:

1. **Evaluating Foundational VPR Architectures:** This work evaluated multiple combinations of backbones and aggregators across various benchmarks. From the results, it is clear that the **DINOv2 backbone paired with a NetVLAD aggregator** achieves the best performance in most cases, not only in standard retrieval metrics but also in the proposed descriptor separation analysis. From the explainability results, the focus in images of the DINOv2 model is the most impressive one, often marking as important the same segments that a human would. Furthermore, the experiments confirmed that training on large-scale datasets allows models to generalize well, as models trained on GSV-Cities generalized well on the Tokyo-24/7 dataset, excelling also in unseen environments.

2. **Limitations of Standard VPR Metrics:** To deal with the limitations of standard VPR metrics, this thesis proposed a methodology centered on the analysis of cosine similarity distributions. By visualizing the separation between "same-place" and "different-place" descriptors and quantifying it with metrics often used on SLAM problems, such as the **Average Precision Score (APS) and Recall at 100% Precision (R@100P)**, this work evaluates the model's discriminative capability, and not only its ranking quality. Although R@100P is already quite widespread in the VPR environment, the use of APS is not, but this thesis shows that it is the one that best represents the overall performance of the model.

3. **Efficient Continual Learning for VPR:** To improve the best available model to date for online learning VPR, this research proposed and validated an intelligent memory

storing strategy. The tests show that **storing the most informative images, rather than random ones, achieves a performance improvement** of two to three percentage points when training for only a few epochs. This model-agnostic improvement is a practical compromise for real-world scenarios, improving continual adaptation without the infeasible requirement of retraining on the entire dataset.

5.2 Discussion of Key Findings

The experimental results show several key findings. The dominance of DINOv2 across nearly all tests without any retraining emphasizes the power of the Visual Transformer foundation model to create visual feature extractors that are also effective for VPR tasks. While computationally more expensive than traditional CNNs like VGG19, its excellent performance and generalization capabilities justify its use in most applications. On some datasets, however, VGG19 with NetVLAD achieved remarkable results, especially on R@100P, becoming a viable, faster alternative.

Regarding aggregators, NetVLAD was consistently the superior choice for Transformer-based backbones, learning a more descriptive global embedding than the simpler pooling mechanism of GeM. The analysis of cosine similarity distributions provided a visual and quantitative confirmation of this, showing a wider separation between positive and negative pairs for NetVLAD. Still, on challenging datasets such as GSV-Cities and NYC-Indoor-VPR, GeM is the one that managed to achieve better scores on R@100P, even if with small values. On CNN instead, GeM and NetVLAD alternate as the best aggregators, based on the type of dataset and backbone. GeM has a better average performance, but NetVLAD manages to achieve the best results when fine-tuning the cluster size.

Focusing on metrics instead, apart from the two usual recall metrics, two more were selected: Average Performance Score (APS) and Recall at 100% Precision (R@100P). While R@100P is commonly used in SLAM contests as it evaluates the quality of a threshold without false positives, it does not take into account the possible correction and filtering that a SLAM algorithm can run to exclude false positives. All the tests still contain it, as it is an important metric; the preferred one is the Average Precision Score, which takes into account the Recall and Precision on different thresholds and the overall performance of the model. This metric is better suited to evaluate different architectures' performance for VPR problems; in the experiments on large datasets, the R@100P is hard to improve, achieving scores that are less than 5%, while the APS manages to get over 70%. The APS often reflects the ranking performance as well, while R@100P is a strict metric that leaves no room for error. Some works [59, 60] have proposed to use the Area Under the Precision-Recall Curve, but while the metric is very similar, it is slightly more optimistic than the APS, and was thus discarded.

On the online learning side, memory modification with intelligent storing is performed as well as the random one on a single epoch, and better on multiple epochs. As this edit is model-agnostic, this improvement will work with all backbones and aggregators—as long as they are retrained—thus demonstrating the effectiveness of this approach.

5.3 Limitations of the Study

Despite the comprehensive coverage of the experiments, this work has several limitations that should be taken into account:

- The evaluations were primarily conducted with frozen backbones when using Visual Transformers. As previously noted, fine-tuning these powerful models, particularly DINOv2, could potentially raise the performance gains and improve adaptation to specific domains.
- The performance on challenging indoor and 360° datasets (NYC-Indoor) remained low across all tested architectures. This is a new challenge that may not be solvable with current standard models and requires dedicated architectural designs.
- No re-ranking algorithms were tested, even if the state of the art on batch learning uses them to achieve the best possible results.
- Some promising state-of-the-art components, such as the ConvAP aggregator, were only tested cursorily and not integrated into the comparative analysis.

5.4 Future Work

The findings and limitations of this thesis open a few paths for future research:

- **Analyzing Fine-Tuning and Re-ranking Foundation Models for VPR:** A dedicated study on the effects of fine-tuning models like DINOv2 on VPR-specific datasets, along with re-ranking, could improve performance in the challenging indoor and urban datasets where they currently falter.
- **Architectural Innovation for 360° and Anonymization:** Future work should focus on developing backbones, aggregators, or losses specifically designed to handle the geometric distortions of 360° fisheye lenses or the feature-poor nature of anonymized indoor environments.
- **Analysis of Cosine Similarity Distributions:** Further statistical analysis of cosine similarity distributions may bring up interesting conclusions that are not evident from the metrics used now. Testing a more elaborate distribution distance than the one tested in this work may show optimal results for the thresholding task.
- **Online Learning Model Improvement Strategies:** Research could explore more advanced memory management techniques, with surprise-based or hybrid criteria, to create an even more efficient and robust memory buffer for continual learning. The lifelong losses and sampling parts are also model-agnostic, and further research may bring a generalized improvement.

Appendix A

Pseudocode of Various Memory Versions in Continual Learning

The main functions edited are the *StoreCurrent* and *StoreEternal* functions, which respectively promote memory from sensory to working and from working to long-term. *StoreEternal* function is called at the end of each epoch, while the *StoreCurrent* one is called by the memory buffer, which is itself called whenever new data is pushed to the memory. Every memory has a separate maximum of images it can keep, called *cap*; the class also stores the numbers of *steps* taken since the beginning of the epoch.
Three memory update strategies were tested:

- **Base:** images are replaced randomly with probability proportional to memory capacity and steps. This is the original VIPeR approach and is reported as Algorithm 1.
- **Diversity-based:** least diverse images (based on internal offset difference) are replaced. The offset metric is dataset-specific, and it's decided when creating the loader. The algorithm is reported as Algorithm 2.
- **Global descriptor-based:** images most similar to new embeddings (via cosine similarity) are replaced to maximize coverage of latent space. This instead is model-agnostic, but it requires an edit in the loss class to pass the global descriptor calculated to the memory class. The only part reported is the edited memory in Algorithm 3.

The replacement factor λ was set to 0.5 in all experiments to balance both memory stability and adaptation.

Algorithm 1 Base Memory Strategy (Random Sampling)

procedure StoreCurrent(batch)
    $L \leftarrow$ number of new images
    if current_num $+ L \leq$ current_cap **then**
        Store new images at available slots
    else
        $p \leftarrow$ current_cap/steps
        if random() $< p$ **then**
            Randomly sample L slots from current memory
            Overwrite them with new images
        end if
    end if
end procedure
procedure StoreEternal
    if eternal memory is empty **then**
        Sample eternal_cap random items from current memory
        Copy to the eternal memory
    else
        $\lambda \leftarrow 0.5$
        Sample $\lambda \cdot$ eternal_cap random current items
        Replace $\lambda \cdot$ eternal_cap random eternal items
    end if
end procedure

Algorithm 2 Diversity-Based Memory Strategy

procedure StoreCurrent(batch)
    $L \leftarrow$ number of new images
    if current_num + $L \leq$ current_cap **then**
        Store directly at the end of current memory
    else
        Compute pairwise difference matrix of 'offset' feature
        Compute diversity score as the sum of absolute differences
        Rank items by redundancy (lower = more redundant)
        Replace L lowest-scoring items with new images
    end if
end procedure
procedure StoreEternal
    if eternal memory is empty **then**
        Fill eternal memory with random samples from current memory
    else
        $\lambda \leftarrow 0.5$
        $k \leftarrow \lambda \cdot$ eternal_cap
        Randomly sample k new items from current memory
        Rank eternal items by redundancy (based on offset similarity)
        Replace k most redundant eternal entries with new ones
    end if
end procedure

Algorithm 3 Global Descriptor-Based Memory Strategy

procedure StoreCurrent(batch)
    for all image x in batch **do**
        if space available in current memory **then**
            Store x directly
        else if x has valid GD (not NaN) **then**
            Compute cosine similarity with all GDs in memory
            Identify the most similar item
            Replace it with x
        else
            Replace a random memory slot with x
        end if
    end for
end procedure
procedure StoreEternal
    if eternal memory is empty **then**
        Select top eternal_cap entries from current memory based on GD norm
        Fill eternal memory
    else
        $\lambda \leftarrow 0.5$
        $k \leftarrow \min(\lambda \cdot \text{eternal\_cap}, \text{current\_num})$
        Select top-k entries from current memory with highest GD norms
        Replace k random entries in eternal memory
    end if
end procedure

References

[1] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I". In: *IEEE Robotics & Automation Magazine* 13.2 (2006), pp. 99–110. DOI: 10.1109/MRA.2006.1638022.

[2] M. Scucchia. "Simultaneous Localization And Mapping for robots: an experimental analysis using ROS". PhD thesis. Univerity of Bologna, 2021. URL: https://amslaurea.unibo.it/id/eprint/25354/.

[3] S. A.S. Mohamed et al. "A Survey on Odometry for Autonomous Navigation Systems". In: *IEEE Access* PP (July 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2929133.

[4] G. Grisetti et al. "A Tutorial on Graph-Based SLAM". In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43. DOI: 10.1109/MITS.2010.939925.

[5] X. Gao and T. Zhang. *Introduction to Visual SLAM From Theory to Practice*. https://github.com/gaoxiang12/slambook-en/blob/master/slambook-en.pdf. Publishing House of Electronics Industry, 2021.

[6] H. D. Landahl, W. S. McCulloch, and W. Pitts. "A statistical consequence of the logical calculus of nervous nets". In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 135–137. ISSN: 1522-9602. DOI: 10.1007/BF02478260. URL: https://doi.org/10.1007/BF02478260.

[7] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain". In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[8] K. O'Shea and R. Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE]. URL: https://arxiv.org/abs/1511.08458.

[9] Y. Le Cun et al. "Handwritten digit recognition: applications of neural network chips and automatic learning". In: *IEEE Communications Magazine* 27.11 (1989), pp. 41–46. DOI: 10.1109/35.41400.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[11] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[12] A. Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[13] S. Ma et al. "A Review of Visual Transformer Research". In: *Proceedings of International Conference on Image, Vision and Intelligent Systems 2023 (ICIVIS 2023)*. Ed. by P. You, S. Liu, and J. Wang. Singapore: Springer Nature Singapore, 2024, pp. 349–356. ISBN: 978-981-97-0855-0.

[14] N. Parmar et al. *Image Transformer*. 2018. arXiv: 1802.05751 [cs.CV]. URL: https://arxiv.org/abs/1802.05751.

[15] A. Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. URL: https://arxiv.org/abs/2010.11929.

[16] L. Pellegrini. "Continual learning for computer vision applications". PhD thesis. alma, Giugno 2022. URL: https://amsdottorato.unibo.it/id/eprint/10401/.

[17] C. Masone and B. Caputo. "A Survey on Deep Visual Place Recognition". In: *IEEE Access* 9 (2021), pp. 19516–19547. DOI: 10.1109/ACCESS.2021.3054937.

[18] R. Arandjelović et al. *NetVLAD: CNN architecture for weakly supervised place recognition*. 2016. arXiv: 1511.07247 [cs.CV]. URL: https://arxiv.org/abs/1511.07247.

[19] M. J. Milford and G. F. Wyeth. "SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights". In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 1643–1649. DOI: 10.1109/ICRA.2012.6224623.

[20] M. Teichmann et al. *Detect-to-Retrieve: Efficient Regional Aggregation for Image Search*. 2019. arXiv: 1812.01584 [cs.CV]. URL: https://arxiv.org/abs/1812.01584.

[21] A. Rau et al. *Predicting Visual Overlap of Images Through Interpretable Non-Metric Box Embeddings*. 2020. arXiv: 2008.05785 [cs.CV]. URL: https://arxiv.org/abs/2008.05785.

[22] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[23] D. Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[24] S. Lowry et al. "Visual Place Recognition: A Survey". In: *IEEE Transactions on Robotics* 32.1 (2016), pp. 1–19. DOI: 10.1109/TRO.2015.2496823.

[25] H. Jégou et al. "Aggregating local descriptors into a compact image representation". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 3304–3311. DOI: 10.1109/CVPR.2010.5540039.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[27] A. S. Razavian et al. *CNN Features off-the-shelf: an Astounding Baseline for Recognition*. 2014. arXiv: 1403.6382 [cs.CV]. URL: https://arxiv.org/abs/1403.6382.

[28] J. Wan et al. "Deep Learning for Content-Based Image Retrieval: A Comprehensive Study". In: *Proceedings of the 22nd ACM International Conference on Multimedia*. MM '14. Orlando, Florida, USA: Association for Computing Machinery, 2014, pp. 157–166. ISBN: 9781450330633. DOI: 10.1145/2647868.2654948. URL: https://doi.org/10.1145/2647868.2654948.

[29] F. Perronnin, J. Sánchez, and T. Mensink. "Improving the Fisher Kernel for Large-Scale Image Classification". In: *Computer Vision - ECCV 2010, 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV*. Ed. by K. Daniilidis, P. Maragos, and N. Paragios. Vol. 6314. Lecture Notes in Computer Science. Springer, 2010, pp. 143–156. ISBN: 978-3-642-15560-4. DOI: 10.1007/978-3-642-15561-1_11. URL: http://dx.doi.org/10.1007/978-3-642-15561-1_11.

[30] G. Tolias, Y. Avrithis, and H. Jégou. "To Aggregate or Not to aggregate: Selective Match Kernels for Image Search". In: *2013 IEEE International Conference on Computer Vision*. 2013, pp. 1401–1408. DOI: 10.1109/ICCV.2013.177.

[31] E.-J. Ong, S. Husain, and M. Bober. "Siamese Network of Deep Fisher-Vector Descriptors for Image Retrieval". In: (Feb. 2017). DOI: 10.48550/arXiv.1702.00338.

[32] G. Tolias, R. Sicre, and H. Jégou. "Particular Object Retrieval With Integral Max-Pooling of CNN Activations". In: *International Conference on Learning Representations*. International Conference on Learning Representations. San Juan, Puerto Rico, May 2016, pp. 1–12. URL: https://inria.hal.science/hal-01842218.

[33] A. Babenko and V. Lempitsky. *Aggregating Deep Convolutional Features for Image Retrieval*. 2015. arXiv: 1510.07493 [cs.CV]. URL: https://arxiv.org/abs/1510.07493.

[34] J. Mao et al. "Learning to Fuse Multiscale Features for Visual Place Recognition". In: *IEEE Access* 7 (2019), pp. 5723–5735. DOI: 10.1109/ACCESS.2018.2889030.

[35] F. Radenović, G. Tolias, and O. Chum. *Fine-tuning CNN Image Retrieval with No Human Annotation*. 2018. arXiv: 1711.02512 [cs.CV]. URL: https://arxiv.org/abs/1711.02512.

[36] M. Caron et al. *Emerging Properties in Self-Supervised Vision Transformers*. 2021. arXiv: 2104.14294 [cs.CV]. URL: https://arxiv.org/abs/2104.14294.

[37] M. Oquab et al. *DINOv2: Learning Robust Visual Features without Supervision*. 2024. arXiv: 2304.07193 [cs.CV]. URL: https://arxiv.org/abs/2304.07193.

[38] S. Hausler and P. Moghadam. "Pair-VPR: Place-Aware Pre-Training and Contrastive Pair Classification for Visual Place Recognition With Vision Transformers". In: *IEEE Robotics and Automation Letters* 10.4 (2025), pp. 4013–4020. DOI: 10.1109/LRA.2025.3546512.

[39] S. Schubert et al. "Visual Place Recognition: A Tutorial [Tutorial]". In: *IEEE Robotics & Automation Magazine* 31.3 (Sept. 2024), pp. 139–153. ISSN: 1558-223X. DOI: 10.1109/mra.2023.3310859. URL: http://dx.doi.org/10.1109/MRA.2023.3310859.

[40] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV]. URL: https://arxiv.org/abs/1409.1556.

[41] K. He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[42] R. Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2022. arXiv: 2108.07258 [cs.LG]. URL: https://arxiv.org/abs/2108.07258.

[43] J. Zhou et al. *iBOT: Image BERT Pre-Training with Online Tokenizer*. 2022. arXiv: 2111.07832 [cs.CV]. URL: https://arxiv.org/abs/2111.07832.

[44] R. Sinkhorn and P. Knopp. "Concerning nonnegative matrices and doubly stochastic matrices". eng. In: *Pacific journal of mathematics* 21.2 (1967), pp. 343–348. ISSN: 0030-8730.

[45] A. Sablayrolles et al. *Spreading vectors for similarity search*. 2019. arXiv: 1806.03198 [stat.ML]. URL: https://arxiv.org/abs/1806.03198.

[46] Z. Liu et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV]. URL: https://arxiv.org/abs/2103.14030.

[47] D. Gao, C. Wang, and S. Scherer. "AirLoop: Lifelong Loop Closure Detection". In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 10664–10671. DOI: 10.1109/ICRA46639.2022.9811658.

[48] Y. Ming et al. "VIPeR: Visual Incremental Place Recognition With Adaptive Mining and Continual Learning". In: *IEEE Robotics and Automation Letters* 10.3 (2025), pp. 3038–3045. DOI: 10.1109/LRA.2025.3539093.

[49] W. Yang et al. "Survey on Explainable AI: From Approaches, Limitations and Applications Aspects". In: *Human-Centric Intelligent Systems* 3.3 (Sept. 2023), pp. 161–188. ISSN: 2667-1336. DOI: 10.1007/s44230-023-00038-y. URL: https://doi.org/10.1007/s44230-023-00038-y.

[50] M. T. Ribeiro, S. Singh, and C. Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1135–1144. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778. URL: https://doi.org/10.1145/2939672.2939778.

[51] M. D. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks". In: *Computer Vision – ECCV 2014*. Ed. by D. Fleet et al. Cham: Springer International Publishing, 2014, pp. 818–833. ISBN: 978-3-319-10590-1.

[52] N. Sünderhauf, P. Neubert, and P. Protzel. "Are we there yet? Challenging SeqSLAM on a 3000 km journey across all four seasons". In: *Proc. of workshop on long-term autonomy, IEEE international conference on robotics and automation (ICRA)*. 2013, p. 2013.

[53] Z. Zeng et al. "Place Recognition: An Overview of Vision Perspective". In: *Applied Sciences* 8 (Nov. 2018), p. 2257. DOI: 10.3390/app8112257.

[54] A. Ali-bey, B. Chaib-draa, and P. Giguère. "GSV-Cities: Toward appropriate supervised visual place recognition". In: *Neurocomputing* 513 (2022), pp. 194–203.

[55] D. Sheng et al. *NYC-Indoor-VPR: A Long-Term Indoor Visual Place Recognition Dataset with Semi-Automatic Annotation*. 2024. arXiv: 2404.00504 [cs.CV]. URL: https://arxiv.org/abs/2404.00504.

[56] A. Torii et al. "24/7 place recognition by view synthesis". In: *CVPR*. 2015.

[57] W. Wang et al. "TartanAir: A Dataset to Push the Limits of Visual SLAM". In: (2020).

[58] X. Shi et al. "Are We Ready for Service Robots? The OpenLORIS-Scene Datasets for Lifelong SLAM". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 3139–3145. DOI: 10.1109/ICRA40945.2020.9196638.

[59] M. Zaffar, L. Nan, and J. F. P. Kooij. *On the Estimation of Image-matching Uncertainty in Visual Place Recognition*. 2024. arXiv: 2404.00546 [cs.CV]. URL: https://arxiv.org/abs/2404.00546.

[60] D. Sferrazza et al. *To Match or Not to Match: Revisiting Image Matching for Reliable Visual Place Recognition*. 2025. arXiv: 2504.06116 [cs.CV]. URL: https://arxiv.org/abs/2504.06116.