POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Understanding and Enhancing Visual Place Recognition through Embedding Space Interpretability and Uncertainty Estimation

Supervisors

Prof. Carlo MASONE

Dr. Gabriele BERTON

Dr. Gabriele TRIVIGNO

Candidate

Davide SFERRAZZA

April 2025

Summary

Visual Place Recognition (VPR) involves determining the geographic location of a photo based solely on its visual content. Recent advancements in Deep Learning have enabled the representation of images in high-dimensional spaces, where photos taken in the same location tend to cluster together, while images from different places are spread apart. This spatial organization makes it easier to predict locations by performing similarity searches against a database of known places.

However, a key gap in current research is understanding the specific information retained in these image embeddings that allows for effective and reliable location prediction. Additionally, existing State-Of-The-Art (SOTA) deterministic methods in VPR are unable to quantify the uncertainty of their predictions. This is particularly problematic in safety-critical applications, such as autonomous driving, where knowing the confidence level in a system's decision is vital for ensuring safety.

This thesis addresses two main challenges: first, understanding and visualizing the essential information encoded in image embeddings, and second, providing uncertainty estimates for VPR models through post-hoc techniques. To overcome these challenges, the thesis employs Generative Artificial Intelligence models, particularly Latent Diffusion Models, to explore and visualize the content within image embeddings. Additionally, uncertainty estimation methods are incorporated to enhance the robustness and reliability of VPR systems.

The contributions of this thesis provide valuable insights into the interpretability and reliability of VPR systems, offering a framework for analyzing the output of these models and incorporating uncertainty quantification during inference.

Acknowledgements

This thesis marks the final chapter of a profound and significant period in my life. The journey has been incredible, beginning when I was an 18-year-old stepping into the challenges of the academic world.

Though I spent the past year and a half at the Polytechnic University of Turin, I want to express my heartfelt gratitude to the University of Palermo, which provided a strong knowledge foundation that allowed me to face any academic challenge. For me, it wasn't a farewell, but simply a "see you later."

In Turin, I had the chance to strengthen existing relationships with people I already knew, while also forming new, meaningful connections. The Polytechnic University of Turin has given me opportunities I truly believe I wouldn't have had in Palermo, both professionally and personally.

I'm especially grateful to the VANDAL lab, which welcomed me with open arms and made me feel at ease throughout the experience. A special thank you goes to Carlo, Bert, and Trivi, who supported me throughout the thesis development, and to Professor Tommasi, who initially gave me the opportunity to conduct my work in the lab.

Throughout these years, I've been fortunate enough to meet amazing people, both students and professors. Each of them has contributed a small piece to the puzzle of who I am today. Every experience, every little detail, every emotion, has left its mark in some way. I hope that everyone who knows me and has shared this journey with me will feel a sense of gratitude as they read this thesis.

I am profoundly grateful to my family–everyone, both those who are still with us and those who are no longer here. To my parents, who have always supported me in everything, including this major life transition, and who have filled my life with love. Dad and Mom, you are my pillars of strength. But my deepest thanks, as vast as infinity itself, go to my brother Gianluca. This journey would never have been possible without him. You've always been my guide, the example to follow, and a myth in my eyes. No words can ever fully express the admiration I feel for you.

Lastly, I want others to feel responsible for this achievement. I am just the product of my experiences, shaped by the people, moments, and decisions that

have influenced me along the way. It is these factors that define who we are. The only gift I want to give to myself is this thesis. This thesis is the true conclusion of a path that I've poured my heart and soul into. I've always believed that with the right determination, anyone can achieve any goal, and that belief will remain with me forever.

Thank you for everything.

"I am indeed amazed when I consider how weak my mind is and how prone to error"
Rene Descartes

Table of Contents

List of Tables VIII

List of Figures X

List of Algorithms XVI

Acronyms XVIII

1 Introduction 1
- 1.1 Focus . . . 1
- 1.2 Objectives . . . 2
- 1.3 Composition . . . 2

2 Background 4
- 2.1 Artificial Intelligence . . . 4
- 2.2 Machine Learning . . . 5
- 2.3 Deep Learning . . . 6
  - 2.3.1 The Perceptron and Biological Neurons . . . 6
  - 2.3.2 Neural Networks and Deep Learning . . . 7
  - 2.3.3 CNNs and the Visual Cortex . . . 9

3 Visual Place Recognition 11
- 3.1 Mathematical Formulation . . . 12
- 3.2 Visual Challenges . . . 13
- 3.3 Connection to Deep Metric Learning . . . 14
  - 3.3.1 Triplet Loss . . . 15
  - 3.3.2 Weakly Supervised Triplet Ranking Loss . . . 15
  - 3.3.3 Contrastive Loss . . . 15
  - 3.3.4 Generalized Contrastive Loss . . . 16
  - 3.3.5 Multi-Similarity Loss . . . 16
- 3.4 Evaluation Metrics . . . 16

4 Generative Artificial Intelligence 17
4.1 Variational Auto-Encoder . . . 18
4.1.1 Auto-Encoder . . . 18
4.1.2 Variational Lower Bound . . . 19
4.1.3 Vector Quantised-Variational Auto-Encoder . . . 20
4.2 Generative Adversarial Networks . . . 21
4.3 Diffusion Probabilistic Models . . . 23
4.3.1 Denoising Diffusion Probabilistic Models . . . 24
4.3.2 Denoising Diffusion Implicit Models . . . 29
4.3.3 Latent Diffusion Models . . . 33

5 Uncertainty Estimation 37
5.1 BayesCap . . . 38
5.2 ProbVLM . . . 41

6 Experiments 45
6.1 Embedding Information Inspection . . . 45
6.1.1 Inspection Framework . . . 45
6.1.2 Implementation Details . . . 46
6.1.3 Metrics . . . 47
6.1.4 Quantitative Results . . . 50
6.1.5 Fidelity to Conditioning Information . . . 53
6.1.6 Qualitative Results . . . 54
6.2 Prediction Uncertainty . . . 60
6.2.1 Uncertainty Framework . . . 60
6.2.2 Implementation Details . . . 61
6.2.3 From Per-Feature to Instance Level Uncertainty . . . 62
6.2.4 Metrics . . . 63
6.2.5 Quantitative Results . . . 63

7 Conclusions and Future Work 69
7.1 Conclusions . . . 69
7.1.1 Findings . . . 69
7.1.2 Limitations . . . 70
7.2 Future Work . . . 70

A Variational Lower Bound 72
A.1 Kullback-Leibler divergence non-negativity . . . 72

B Probability Distributions 73
B.1 Heavy-tailed distribution . . . 73

C Experimental Details and Additional Results 74
C.1 LDM training curves 74
C.2 Quantitative LDM results 83
C.3 BayesCap training curves 104
C.4 Binning strategy for BayesCap results 106
C.5 BayesCap uncertainty distributions 108

Bibliography 112

List of Tables

2.1 XOR function. 7

6.1 Visual Place Recognition models used for conditioning. 46

6.2 Training and validation sets derived from the training split of the SF-XL [52] dataset used for training LDM models. 47

6.3 Architectural and training hyperparameters for the LDM model (see Section 4.3.1, Section 4.3.3, and «High-resolution image synthesis with latent diffusion models» [42] for further details on notation and the meaning of the architectural hyperparameters). 48

6.4 Visual and fidelity metrics for the LDM conditioned on CosPlace's output space [52] with $d = 2048$, while varying the scale parameter s for CFG [39] and the number of images N. Metrics labeled with $\uparrow$ indicate better performance when higher, and those labeled with $\downarrow$ are better when lower. For brevity, 'P' stands for Precision and 'R' for Recall. The number of steps in the accelerated generation process is fixed at $S = 250$. The row representing the common evaluation practice is highlighted, with the chosen scale s indicated in a distinct color. 52

6.5 Summary of L_1 and L_2 metrics for CosPlace [52] classes in the validation set of Table 6.2. The second half of the table reports the metrics for our LDM model, with scale parameter $s \in \{1, 2\}$ for CFG [39]. 57

6.6 Grid search for the number of steps S in the accelerated generation process and the hyperparameter η controlling the level of stochasticity. 57

6.7 Training and validation sets derived from the training split of the EigenPlaces [19] dataset used for training BayesCap [58] models. . . 62

6.8 Architectural and training hyperparameters for BayesCap [58]. . . . 62

6.9 Summary of trained BayesCap [58] models. 64

C.1 Reference table showing the mean pairwise distances for all Visual Place Recognition models, calculated from $14k$ randomly selected images from the validation set of Table 6.2. The embedding dimension d is included to distinguish between different versions of the same VPR model. 83

List of Figures

2.1 Perceptron. Adapted from *Artificial intelligence: a new synthesis* [1]. 7
2.2 Implementation of the XOR function presented in Table 2.1. The numbers inside the perceptrons represent the threshold b for each perceptron. 8
2.3 Multi-Layer Perceptron, a type of Feedforward Neural Network. . . 8
2.4 Example of the convolution operation. 9

3.1 Common Visual Place Recognition pipeline. Adapted from «A Survey on Deep Visual Place Recognition» [10]. 12

4.1 Auto-Encoder model. 19
4.2 Generative Adversarial Network framework. 22
4.3 Scheme of the Markovian diffusion process. Adapted from «Denoising diffusion probabilistic models» [41]. 25
4.4 Illustration of the forward process. 30
4.5 Illustration of the reverse process. 31
4.6 Scheme of the non-Markovian diffusion process. Adapted from «Denoising diffusion implicit models» [40]. 31
4.7 Scheme of the accelerated diffusion process, with the first two elements of $\boldsymbol{\tau}$ set to 2 and 4. Adapted from «Denoising diffusion implicit models» [40]. 33
4.8 Diagram of the Latent Diffusion Model with conditioning through either concatenation or cross-attention. Adapted from «High-resolution image synthesis with latent diffusion models» [42]. 35

5.1 BayesCap architecture. Adapted from «BayesCap: Bayesian Identity Cap for Calibrated Uncertainty in Frozen Neural Networks» [58]. . . 40
5.2 ProbVLM architecture. Adapted from «ProbVLM: Probabilistic Adapter for Frozen Vision-Language Models» [65]. 44

6.1 Illustration of a LDM model conditioned on the output space of a VPR model. 45

6.2 Plots of visual metrics for LDM conditioned on CosPlace's output space [52] with $d = 2048$, using VGG-16, while varying the scale parameter s for CFG [39]. The number of steps in the accelerated generation process is fixed at $S = 250$. 54
6.3 Plots of visual metrics for LDM conditioned on CosPlace's output space [52] with $d = 2048$, using InceptionV3, while varying the scale parameter s for CFG [39]. The number of steps in the accelerated generation process is fixed at $S = 250$. 55
6.4 Plots of fidelity metrics for LDM conditioned on CosPlace's output space [52] with $d = 2048$, while varying the scale parameter s for CFG [39]. The number of steps in the accelerated generation process is fixed at $S = 250$. 56
6.5 Figure 1 showing five generated images conditioned on a query from the SF-XL [52] validation split (shown in the first column). The images are generated by varying the number of steps S in the accelerated reverse process and the hyperparameter η, which controls the level of stochasticity. The specific values used to generate each image set are indicated on the left of each corresponding set. 58
6.6 Figure 2 showing five generated images conditioned on a query from the SF-XL [52] validation split (shown in the first column). The images are generated by varying the number of steps S in the accelerated reverse process and the hyperparameter η, which controls the level of stochasticity. The specific values used to generate each image set are indicated on the left of each corresponding set. 59
6.7 Illustration of the viewpoint invariance property for Visual Place Recognition models. 60
6.8 Calibration plots and sparsification curves for all trained BayesCap [58] models, computed on the Pitts30k [15] validation set. 66
6.9 Calibration plots and sparsification curves for all trained BayesCap [58] models, computed on the SF-XL [52] validation set. 67
6.10 Calibration plots and sparsification curves for all trained BayesCap [58] models, computed on the MSLS [78] validation set. 68

C.1 Training plot of the Latent Diffusion Model conditioned on the output space of AP-GeM [14]. 75
C.2 Training plot of the Latent Diffusion Model conditioned on the output space of CliqueMining [13]. 75
C.3 Training plot of the Latent Diffusion Model conditioned on the output space of Conv-AP [25]. 76
C.4 Training plot of the Latent Diffusion Model conditioned on the output space of CosPlace [52] with $d = 32$. 76

C.5 Training plot of the Latent Diffusion Model conditioned on the output space of CosPlace [52] with $d = 64$. 77
C.6 Training plot of the Latent Diffusion Model conditioned on the output space of CosPlace [52] with $d = 128$. 77
C.7 Training plot of the Latent Diffusion Model conditioned on the output space of CosPlace [52] with $d = 512$. 78
C.8 Training plot of the Latent Diffusion Model conditioned on the output space of CosPlace [52] with $d = 2048$. 78
C.9 Training plot of the Latent Diffusion Model conditioned on the output space of CricaVPR [17]. 79
C.10 Training plot of the Latent Diffusion Model conditioned on the output space of EigenPlaces [19] with $d = 128$. 79
C.11 Training plot of the Latent Diffusion Model conditioned on the output space of EigenPlaces [19] with $d = 512$. 80
C.12 Training plot of the Latent Diffusion Model conditioned on the output space of EigenPlaces [19] with $d = 2048$. 80
C.13 Training plot of the Latent Diffusion Model conditioned on the output space of MixVPR [18]. 81
C.14 Training plot of the Latent Diffusion Model conditioned on the output space of NetVLAD [15]. 81
C.15 Training plot of the Latent Diffusion Model conditioned on the output space of SALAD [51]. 82
C.16 Training plot of the Latent Diffusion Model conditioned on the output space of SFRS [71]. 82
C.17 Plots of LDM conditioned on AP-GeM's output space [14]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 85
C.18 Plots of LDM conditioned on CliqueMining's output space [13]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 86
C.19 Plots of LDM conditioned on Conv-AP's output space [25]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 87

C.20 Plots of LDM conditioned on CosPlace's output space [52] ($d = 32$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 88
C.21 Plots of LDM conditioned on CosPlace's output space [52] ($d = 64$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 89
C.22 Plots of LDM conditioned on CosPlace's output space [52] ($d = 128$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 90
C.23 Plots of LDM conditioned on CosPlace's output space [52] ($d = 512$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 91
C.24 Plots of LDM conditioned on CricaVPR's output space [17]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 92
C.25 Plots of LDM conditioned on EigenPlaces's output space [19] ($d = 128$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 93
C.26 Plots of LDM conditioned on EigenPlaces's output space [19] ($d = 512$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 94
C.27 Plots of LDM conditioned on EigenPlaces's output space [19] ($d = 2048$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 95

C.28 Plots of LDM conditioned on MixVPR's output space [18]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 96

C.29 Plots of LDM conditioned on NetVLAD's output space [15]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 97

C.30 Plots of LDM conditioned on SALAD's output space [51]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 98

C.31 Plots of LDM conditioned on SFRS's output space [71]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39]. 99

C.32 Distribution plots of the distances for the 14,000 random images used to compute the metrics in Table C.1. The arrows highlight the points in the distribution corresponding to the L_2 distance of the LDM models, based on the scale parameter s for CFG [39]. The distributions for AP-GeM [14], CliqueMining [13], Conv-AP [25], and CricaVPR [17] are shown. 100

C.33 Distribution plots of the distances for the 14,000 random images used to compute the metrics in Table C.1. The arrows highlight the points in the distribution corresponding to the L_2 distance of the LDM models, based on the scale parameter s for CFG [39]. The distributions for MixVPR [18], NetVLAD [15], SALAD [51], and SFRS [71] are shown. 101

C.34 Distribution plots of the distances for the 14,000 random images used to compute the metrics in Table C.1. The arrows highlight the points in the distribution corresponding to the L_2 distance of the LDM models, based on the scale parameter s for CFG [39]. The distributions for CosPlace [52] with $d \in \{32, 64, 128, 512\}$ are shown. 102

C.35 Distribution plots of the distances for the 14,000 random images used to compute the metrics in Table C.1. The arrows highlight the points in the distribution corresponding to the L_2 distance of the LDM models, based on the scale parameter s for CFG [39]. The distributions for CosPlace [52] with $d = 2048$, and EigenPlaces [19] with $d \in \{128, 512, 2048\}$ are shown. 103
C.36 Training plot of BayesCap [58] on top of CosPlace [52], with a descriptor dimension of $d = 2048$. The model was trained for 30 epochs. 104
C.37 Training plot of BayesCap [58] on top of CosPlace [52], with a descriptor dimension of $d = 2048$. The model was trained for 50 epochs. 105
C.38 Training plot of BayesCapCycle on top of CosPlace [52], with a descriptor dimension of $d = 2048$. The model was trained for 30 epochs. 105
C.39 Uncertainty distributions for all trained BayesCap [58] models, computed on the Pitts30k [15] validation set. 109
C.40 Uncertainty distributions for all trained BayesCap [58] models, computed on the SF-XL [52] validation set. 110
C.41 Uncertainty distributions for all trained BayesCap [58] models, computed on the MSLS [78] validation set. 111

List of Algorithms

1 SGD optimization of GANs. 23
2 Training DDPM models. 28
3 Sampling from DDPM models. 29

Acronyms

SOTA
State-Of-The-Art

AI
Artificial Intelligence

Gen-AI
Generative Artificial Intelligence

ML
Machine Learning

DL
Deep Learning

SGD
Stochastic Gradient Descent

MLP
Multi-Layer Perceptron

NN
Neural Network

DNN
Deep Neural Network

BNN
Bayesian Neural Network

CNN

Convolutional Neural Network

VPR

Visual Place Recognition

AE

Auto-Encoder

VAE

Variational Auto-Encoder

VQ-VAE

Vector Quantised-Variational Auto-Encoder

GAN

Generative Adversarial Network

DM

Diffusion Model

DDPM

Denoising Diffusion Probabilistic Model

DDIM

Denoising Diffusion Implicit Model

LDM

Latent Diffusion Model

CFG

Classifier-Free Guidance

VLM

Vision-Language Model

KNN

K-Nearest Neighbors

FoV

Field of View

i.i.d.

independent and identically distributed

Chapter 1

Introduction

1.1 Focus

This thesis focuses on the Visual Place Recognition (VPR) task, which aims to determine the *location* where a *photo*–referred to as the *query*–was taken, based solely on the image content. Current methods build upon advancements in Deep Learning (DL), a subfield of Machine Learning (ML), by embedding the image into an *embedding space*. Ideally, in this space, images depicting the same location are positioned close together, while those depicting different locations are placed farther apart. To predict the location of the query, a *similarity search* is typically performed *on-line* against a pre-established *database* of images with known locations, where embeddings are precomputed *off-line*. The embedding space is often learned using *deep metric learning loss functions*, which impose some form of supervision on the relative distances between images. Consequently, the models autonomously identify which features are essential for distinguishing between different locations.

A key limitation of current *deterministic* State-Of-The-Art (SOTA) methods is that they do not provide any indication of *uncertainty* regarding their predictions. This is especially critical in *safety-critical scenarios*, such as *autonomous driving*, where incorrect predictions could have severe consequences, including harm to people or environmental damage. In such contexts, providing uncertainty scores–quantifying the model's confidence in its predictions–is essential. These scores allow both humans and systems to identify uncertain predictions and avoid making risky decisions.

The lack of supervision over the embedding information and the computation of uncertainty scores are key aspects of this thesis, as elaborated in the following section.

1.2 Objectives

The first objective of this thesis is to develop a framework that enables understanding the key information from photos that various VPR models deem important, and which is thus preserved in the extracted embeddings. To address this challenge, we utilize advances in Generative Artificial Intelligence (Gen-AI) with a class of models known as Latent Diffusion Models (LDMs). Typically, *generative* models are designed to create new images from random noise. LDMs operate similarly, but with the added capability of conditioning the generation process on external inputs. By conditioning the generation on the embeddings extracted by VPR models and varying the initial noise used for generation, we can infer which information is retained in the embeddings. We expect that the information encoded in the embeddings will remain consistent in the generated images, while other factors will vary.

The second objective of this thesis is to provide uncertainty scores for pre-trained VPR models using post-hoc methods derived from existing literature on *uncertainty estimation*. These post-hoc techniques can be applied to *frozen* models, thereby enhancing their robustness without compromising their inference performance.

1.3 Composition

This thesis encompasses both theoretical concepts and an analysis of empirical results. The structure of the work is as follows:

Chapter 1 - Introduction: The *Introduction* outlines the focus of the thesis, the objectives, and provides an overview of the structure of the work.

Chapter 2 - Background: The *Background* chapter provides foundational knowledge on Artificial Intelligence (AI), Machine Learning (ML), Deep Learning (DL), Neural Networks (NNs), and Convolutional Neural Networks (CNNs), along with a discussion on the relationship between *biological processes* and *artificial neurons*.

Chapter 3 - Visual Place Recognition: This chapter explains the VPR task, the primary pipeline used for predictions, the visual challenges involved, and the key loss functions typically employed in training DL models.

Chapter 4 - Generative Artificial Intelligence: This chapter offers an extensive overview of the current SOTA in Gen-AI, covering architectures such as Auto-Encoders (AEs), Variational Auto-Encoders (VAEs), Vector Quantised-Variational Auto-Encoders (VQ-VAEs), Generative Adversarial Networks (GANs), Denoising Diffusion Probabilistic Models (DDPMs), Denoising Diffusion Implicit Models (DDIMs), and Latent Diffusion Models (LDMs).

Chapter 5 - Uncertainty Estimation: This chapter explores the different types of uncertainties that are important in Deep Learning, the main approaches to uncertainty estimation, and the foundational works that underpin this thesis.

Chapter 6 - Experiments: This chapter illustrates the methodologies and frameworks proposed to achieve the goals of this thesis, including architectures, implementation details, evaluation metrics, quantitative and qualitative results.

Chapter 7 - Conclusions and Future Work: This chapter concludes this thesis by summarizing the main findings and limitations of the proposed approaches, along with possible real-world applications and avenues for future research.

Appendix A - Variational Lower Bound: This appendix contains a brief mathematical proof of the non-negativity of the Kullback-Leibler divergence.

Appendix B - Probability Distributions: This appendix provides the definition of heavy-tailed distribution.

Appendix C - Experimental Details and Additional Results: This appendix contains the training curves of the trained models, further implementation details, and additional results not included in Chapter 6.

Chapter 2

Background

2.1 Artificial Intelligence

Artificial Intelligence (AI) can be defined as the intelligent behavior exhibited by machines [1]. The question of whether machines can think, first raised by Turing in 1950, with the seminal paper «*Computing Machinery and Intelligence*»[2], has been a subject of fascination for philosophers, scientists, and engineers ever since. Despite the ill-defined nature of the question–since intelligence itself is not clearly defined even in humans [3]–the concept of machines mimicking human behavior has driven the development of a global research community. This community is characterized by diverse and often contradictory views on the capabilities of machines and the potential future of AI, both in the short and long term.

In their influential book *Artificial intelligence: a modern approach*, Russell and Norvig [4] proposed a framework for categorizing AI into four approaches:

Thinking humanly: AI aims to replicate human thought processes by studying introspection, psychology, and brain imaging, ultimately developing cognitive models of the human mind.

Thinking rationally: AI employs *logic* to formalize rational thought. It translates informal knowledge into formal terms, enabling the application of general deduction procedures to solve problems.

Acting humanly: AI systems are designed to pass the Turing's test [2], which requires capabilities such as natural language processing, knowledge representation, automated reasoning, machine learning, computer vision, and robotics. The goal is to create a machine whose behavior is indistinguishable from that of a human.

Acting rationally: AI agents are designed to act rationally, making decisions aimed at achieving the best possible outcome, even in the face of uncertainty. In

this approach, the focus is on the outcomes and decisions themselves, rather than the cognitive processes behind them. AI agents are formally defined as functions that map percept sequences (the history of sensory inputs) to actions, ensuring that the agent's behavior is optimized based on its perception of the environment.

Overall, intelligent behavior encompasses a range of abilities, including perception, reasoning, problem solving, learning from experience, understanding, communicating, and acting in complex, dynamic environments.

2.2 Machine Learning

Machine Learning (ML) is a subfield of AI that focuses on enabling systems to *learn* directly from *data* or *experience*. As defined by Tom Mitchell [5]:

> "A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

In this context, the program autonomously learns to perform tasks, without the need for human intervention in its competence development.

In ML, the process begins with data collection, often followed by a pre-processing step. The data is organized into *datasets*, which are typically divided into three subsets:

Training set: The *training set* is the largest subset and is used to train the ML models. During this phase, the models identify patterns and relationships in the data by adapting to the data's underlying structure to improve their predictions and generalize to new situations.

Validation set: The *validation set* serves multiple purposes. It is used to tune the models' *hyperparameters*, which are predefined values that are not learned by the model but must be manually specified by the designers before training. It is also used for *model selection* and to prevent *overfitting*–the phenomenon where a model memorizes the training data instead of learning generalizable patterns. Importantly, models do not learn from the validation set; it is used solely for evaluation and optimization purposes.

Test set: Once the models have been trained and tuned, they are evaluated on the *test set*. This step assesses the model's ability to generalize to new, unseen data, providing an estimate of how well the model is likely to perform in real-world scenarios.

Machine Learning can be categorized into three distinct, non-overlapping learning paradigms:

Supervised Learning: In *supervised learning*, models are trained on *labeled datasets*, where each training example consists of *input features* paired with corresponding *output labels*. The goal is for the model to learn a mapping function that can predict outputs from new, unseen inputs.

Unsupervised Learning: In *unsupervised learning*, models work with *unlabeled data*. Here, the objective is to identify hidden structures, patterns, or relationships within the data, without any explicit guidance in the form of labels.

Reinforcement Learning: in *reinforcement learning*, models, referred to as *agents*, learn by interacting with an environment. They receive feedback in the form of *rewards* (positive feedback) or *penalties* (negative feedback). The model aims to learn the best actions to take in specific situations to maximize long-term cumulative rewards.

In addition to the primary paradigms, several *hybrid* approaches integrate elements from the categories mentioned earlier. For example, in *semi-supervised learning*, models leverage both labeled data and large volumes of unlabeled data. In *self-supervised learning*, there is no reliance on labeled data, as the model generates its own labels from the input data; this approach is commonly used for pre-training in fields like *natural language processing* or *computer vision*. Finally, *active learning* involves querying the user (or *oracle*) for labels on the data samples that are most uncertain or informative.

2.3 Deep Learning

2.3.1 The Perceptron and Biological Neurons

In 1958, inspired by the functioning of *biological neurons*, Rosenblatt [6] introduced the *perceptron*. Biological neurons receive *electrical signals* through *dendrites* from other neurons. If the cumulative signal reaches a threshold at the *axon hillock*, the neuron 'fires' an *action potential* that travels down the *axon* to communicate with other neurons or muscles. The perceptron operates in a similar manner.

The perceptron is characterized by a *weight vector* $\boldsymbol{\omega} = (\omega_1, \omega_2, \ldots, \omega_n)$, with each component ω_i being a *weight*, and a *threshold* b. It takes as input a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, known as the *feature vector*, where each x_i is referred to as a *feature value*. The perceptron produces an output of 1 if the weighted sum of the inputs meets or exceeds the threshold, *i.e.*, if $\mathbf{x} \cdot \boldsymbol{\omega} = \sum_{i=1}^{n} \omega_i x_i \geq b$, and an output of 0 otherwise. A graphical representation of the perceptron is shown in Fig. 2.1. In this framework, the perceptron performs binary classification, where each feature vector $\mathbf{x}$ is labeled with $y \in \{0, 1\}$ during both training and evaluation. Its

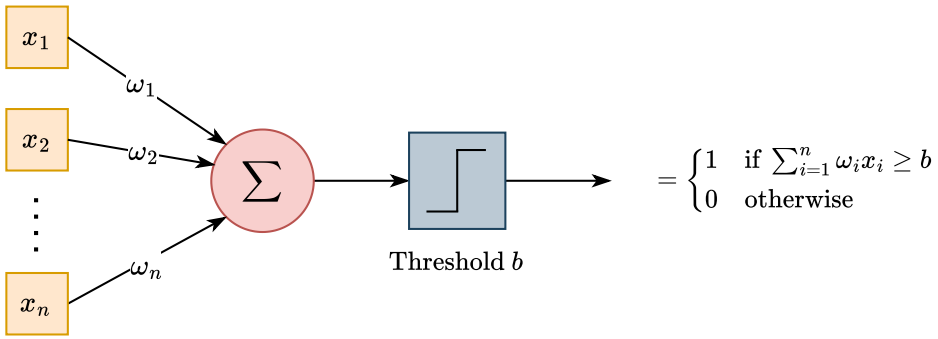


Figure 2.1: Perceptron. Adapted from *Artificial intelligence: a new synthesis* [1].

performance is typically measured using *accuracy*, *i.e.*, the proportion of correctly classified samples.

Since the perceptron's output is determined by a unit step function, it is inherently discrete and non-differentiable, which prevents the direct application of optimization methods like *gradient descent*. Moreover, when the data is *non-linearly separable*, such as in the case of the XOR function shown in Table 2.1, the training algorithm proposed by Rosenblatt may fail to converge. To overcome these challenges, the step function can be replaced by continuous non-linearities, such as the *sigmoid function* $\sigma(x) := \frac{1}{1+e^{-x}}$, transforming the perceptron from a deterministic binary classifier into a probabilistic one. In this context, the perceptron is referred to as a *neuron*, the threshold b is known as the *bias* and the non-linear function is called *activation function*.

x_1	x_2	o
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.1: XOR function.

2.3.2 Neural Networks and Deep Learning

By interconnecting multiple perceptrons and arranging them into *layers*, the XOR problem becomes solvable, as demonstrated in Fig. 2.2, which illustrates a possible solution.

The idea of connecting multiple neurons gave rise to various models, collectively referred to as Neural Networks (NNs). The simplest form of a Neural Network,

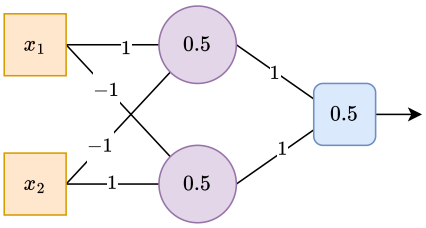


Figure 2.2: Implementation of the XOR function presented in Table 2.1. The numbers inside the perceptrons represent the threshold b for each perceptron.

illustrated in Fig. 2.3, is the Multi-Layer Perceptron (MLP), a network comprising multiple layers and fully connected neurons between them. A MLP is a type of *Feedforward Neural Network*, where information flows unidirectionally from input to output with no feedback loops.

A MLP consists of several layers: the first layer is the *input layer*, which contains the feature values; the last layer is the *output layer*, which contains the *output neurons*; situated between the input and output layers there are the *hidden layers*, which process information in ways not directly visible from either the input or output.

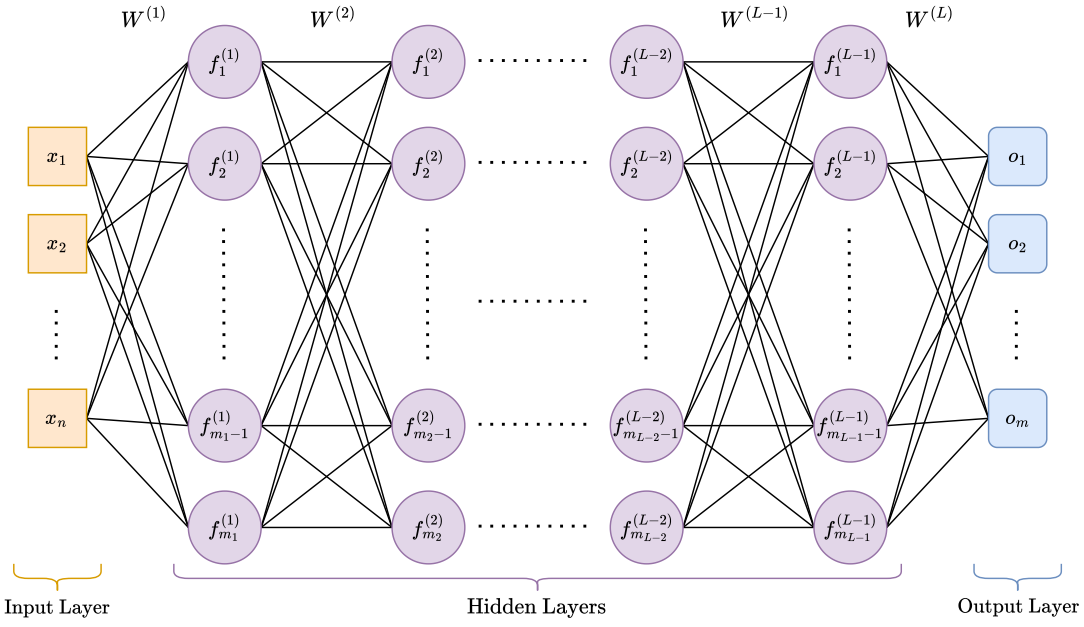


Figure 2.3: Multi-Layer Perceptron, a type of Feedforward Neural Network.

The neurons in both the hidden and output layers can utilize any form of activation function, as long as the *backpropagation algorithm* [7] can be applied to

adjust the *network weights* $\Theta := \bigcup_{i=1}^{L}\{W^{(i)}\}$, where L is the total number of layers and $W^{(i)}$ represents the *weight matrix* for the i-th layer, which includes both the weights and biases of the neurons in that layer.

Multi-Layer Perceptrons have been proven to be *universal approximators* [8], meaning they can approximate any continuous function with just one hidden layer. This capability makes them highly effective for function approximation when combined with optimization techniques, resulting in precise predictions. Expanding the network by adding more layers further enhances its representational power, giving rise to the field of Deep Learning (DL).

2.3.3 CNNs and the Visual Cortex

The concept of Convolutional Neural Networks (CNNs) was first introduced by LeCun et al. [9] in 1989. Similar to the perceptron introduced by Rosenblatt, CNNs are a family of Neural Networks inspired by the visual processing system of the brain in animals. The key operation behind CNNs is called *convolution*.

Let I be an image with elements indexed by x and y, and let $\boldsymbol{\omega}$ be a *filter* or *kernel* indexed by i and j. The convolution operation is performed in a sliding-window manner and is defined as:

$$(I * \boldsymbol{\omega})[x, y] = \sum_i \sum_j \boldsymbol{\omega}[i, j] I[x - i, y - j]$$

An example of the convolution operation is shown in Fig. 2.4.

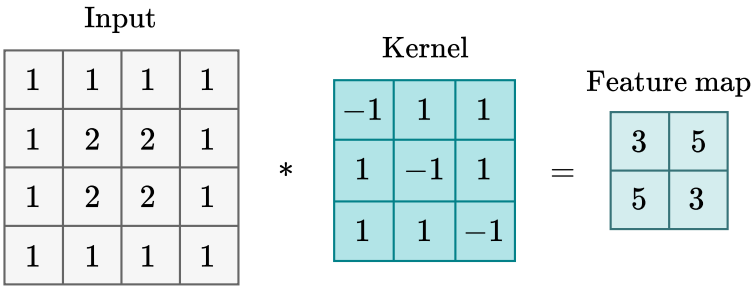


Figure 2.4: Example of the convolution operation.

In the context of CNNs, a neuron is a computational unit that performs the convolution operation, adds a bias b, and applies an activation function σ to produce the *feature map* f. This can be written as:

$$f[x, y] = \sigma\left(b + \sum_i \sum_j \boldsymbol{\omega}[i, j] I[x - i, y - j]\right)$$

A unique feature of CNNs is the *weight-sharing* property, where the weights $\boldsymbol{\omega}$ in the sliding-window process are shared across all positions in the image. This weight-sharing mechanism significantly reduces the number of parameters in comparison to fully connected layers in MLPs, where the number of parameters scales with the total number of connections. In CNNs, the number of parameters scales linearly with the number of kernels used.

Convolutional layers are particularly suited for *grid-like data*, such as images, which contain *structural priors* like repeating patterns, locality, and hierarchical structures. In this setup, individual neurons in a convolutional layer activate in a way that is similar to biological neurons in the *visual cortex*, which respond to stimuli from a restricted region of the visual field known as the *receptive field*. In the case of images, the receptive field corresponds to a small region of the original image.

As information passes through successive areas in the visual cortex, it becomes increasingly abstract, and higher-level structures are recognized. Similarly, by stacking multiple convolutional layers, CNNs are able to capture a hierarchical structure in the data: early layers detect low-level features like edges and textures, intermediate layers identify more complex patterns such as object parts, and deeper layers capture high-level features like objects or faces.

Chapter 3

Visual Place Recognition

Visual Place Recognition (VPR) [10] is the task of recognizing the location where an image was taken given only its visual content, *i.e.* the image itself. It is a building block of fundamental importance for applications in autonomous driving, augmented reality, robotics, absolute pose estimation, simultaneous localization and mapping, etc.

The problem is commonly treated as an image retrieval task, where a summary description of an image of interest, called a query, is computed and compared against a database of descriptions of images with known locations. The location and description change depending on the particular task for which a VPR block is needed and by the adopted methodology to build it.

The place depicted by an image could be annotated by the name of a point of interest, called a landmark, or by a finite sequence of numbers, *e.g.* GPS coordinates, depending on the task at hand.

The description of images has evolved significantly over the years, beginning with hand-crafted local features like SIFT [11] and global descriptors such as VLAD [12]. Recent advancements in Deep Learning (DL) have enabled the use of deep neural networks like a Convolutional Neural Network (CNN) to extract global image descriptors from their final fully connected layers or by the aggregation of the output of their final convolutional layers, allowing for their direct application to the Visual Place Recognition (VPR) task.

To determine the location of the query image, a summary description is computed for both the query and the database images. A similarity search is then conducted over the database using a distance metric, often implemented with a K-Nearest Neighbors algorithm. This algorithm identifies the K most similar images to the query based on their representations. Once the K nearest neighbors are retrieved, they contribute to the final location prediction. Typically, the location is predicted by selecting the most similar database descriptor, though in some cases, additional aggregation techniques may be used to refine the prediction.

A full Visual Place Recognition pipeline, incorporating the steps outlined above, is shown in Fig. 3.1. This pipeline includes an optional refinement step, which seeks to re-rank the nearest neighbors by utilizing local image features. However, this step is seldom used due to its time-consuming nature and high computational cost. The figure also highlights the processing time of the descriptors. While the descriptor for the query image is computed online, the descriptors for the database images are pre-computed and stored offline due to the large number of images involved. However, since the top-K nearest neighbors need to be retrieved in order to predict the location, they must be loaded into volatile memory prior to performing the similarity search. This, in turn, limits the size of the descriptors generated by the VPR model, as the entire database must be considered during the search.

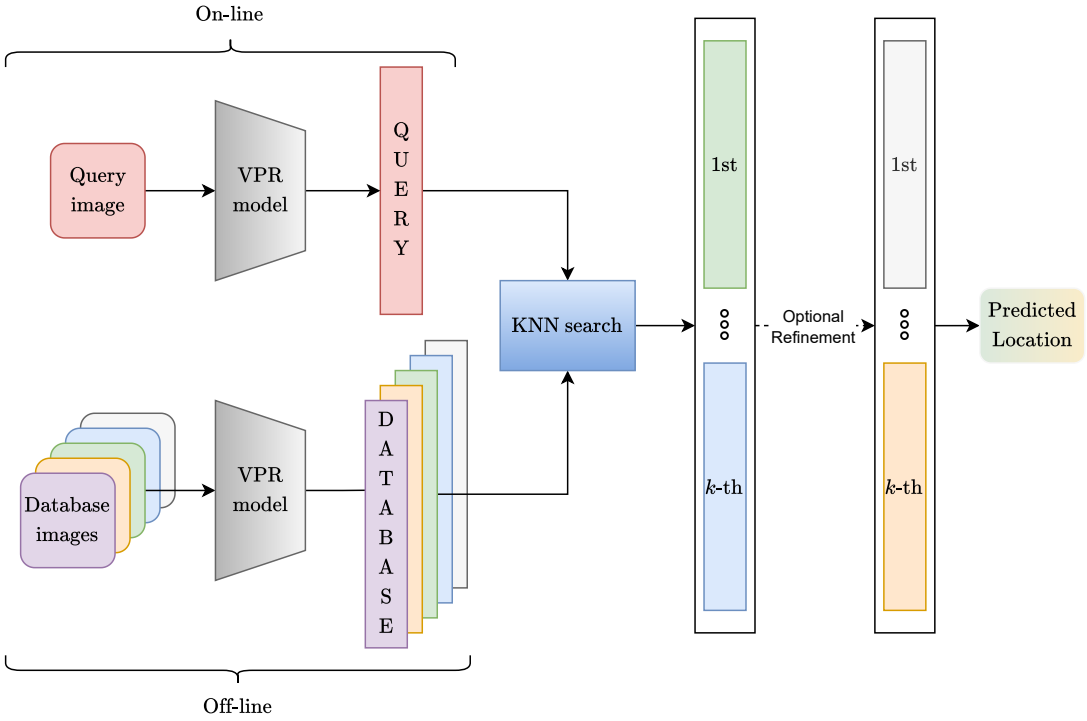


Figure 3.1: Common Visual Place Recognition pipeline. Adapted from «A Survey on Deep Visual Place Recognition» [10].

3.1 Mathematical Formulation

By partially aligning with the mathematical foundations introduced in [13, 14, 15, 16], the VPR task aims to map a query image $I_q \in \mathcal{I} \subseteq \mathbb{R}^{H \times W \times 3}$ in RGB format

into a d-dimensional space, $\mathcal{O} \subset \mathbb{R}^d$, through a mapping function $f : \mathcal{I} \to \mathcal{O}$. Currently, the mapping function f is typically realized using a Deep Learning network, which is parameterized by a set of learnable parameters Θ, with the dependence on these parameters denoted as f_Θ. Henceforth, we implicitly assume that the function is realized by a DL model, *i.e.* $f \coloneqq f_\Theta$.

Given a set of N database images from the image space $\mathcal{I}$ with known locations in $\mathcal{P}$, *i.e.* $\mathcal{D} \coloneqq \{(I_1, P_1), (I_2, P_2), \ldots, (I_N, P_N)\} \subseteq \{(I, P) \mid I \in \mathcal{I}, P \in \mathcal{P}\}$, the mapping function f is applied to each image, resulting in a set of embeddings $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N\} \subset \mathcal{O}$. Similarly, for the query image, the embedding $\mathbf{e}_q = f(I_q)$ is computed.

Following this initial step, a similarity search must be conducted. Depending on the structure of the embedding space $\mathcal{O}$–which may reside on a unit hypersphere $\mathcal{O} \subseteq \mathcal{S} \coloneqq \left\{\mathbf{e} \in \mathbb{R}^d \mid \|\mathbf{e}\|_2 = 1\right\}$, as a result of L_2 normalization applied at the end of the network, or not–the choice of the distance function $d : \mathcal{O} \times \mathcal{O} \to \mathbb{R}$ can vary. If the embedding is L_2-normalized, the dot product is typically used to compute the distance, defined as:

$$d_i \coloneqq d(\mathbf{e}_i, \mathbf{e}_q) = 1 - \mathbf{e}_i^T \mathbf{e}_q, \forall i \in \{1, 2, \ldots, N\}$$

In cases where the embedding is not L_2-normalized, the L_p norm is commonly employed, with $p = 2$ in most situations. The resulting distances are then computed as:

$$d_i \coloneqq d(\mathbf{e}_i, \mathbf{e}_q) = \|\mathbf{e}_i - \mathbf{e}_q\|_p, \forall i \in \{1, 2, \ldots, N\}$$

The similarity search is usually conducted using a K-Nearest Neighbors algorithm to retrieve the K most similar embeddings from the database, resulting in a reference set $\mathcal{R} = \left[\mathbf{e}_{(1)}, \mathbf{e}_{(2)}, \ldots, \mathbf{e}_{(K)}\right]$. The elements in this reference set are sorted by increasing distance, *i.e.* $d_{(1)} \leq d_{(2)} \leq \ldots \leq d_{(K)}$. The predicted location P_q of the query image is derived as a combination of the locations $\left\{P_{(1)}, P_{(2)}, \ldots, P_{(K)}\right\}$ corresponding to the nearest neighbors in R. However, in practice, the most common approach is to use only the location of the first nearest neighbor to predict P_q, *i.e.* $P_q \coloneqq P_{(1)}$.

3.2 Visual Challenges

The task involves analyzing an image's visual content to determine the location it represents. Given that the image could be captured at any time and from various sources, several challenges may arise. In cases where the image quality is significantly degraded, the system can use a detection mechanism to reject the image for location prediction, prompting the user to submit a higher-quality version. However, if the image is determined to be of sufficient quality, the system

will proceed with making the location prediction. In this case, the system must overcome three key challenges [17, 18, 19, 16]:

1. Variations in conditions resulting from short- and long-term changes in appearance, such as weather, occlusions, and illumination shifts.

2. Viewpoint variations relative to the database images.

3. Perceptual aliasing, where visually similar images correspond to geographically distant locations.

The first challenge has been tackled in the literature by leveraging advanced Deep Learning models that generate highly general feature representations. This has been achieved by Keetha et al. [20], for example, through the use of large-scale pre-trained models, referred to as *foundational models*, such as DINO [21] and DINOv2 [22], or by combining and automatically aggregating spatial features in a holistic manner, as demonstrated by Ali-Bey et al. [18].

To address the second challenge, Berton et al. [19] propose a novel training strategy that encourages models to generate perspective-invariant features by leveraging location coordinates and viewpoint angles. Alternatively, Leyva-Vallina et al. [23] suggest training models with a new loss function that employs similarities between images based on overlapping Field of View (FoV) or 3D pointclouds. Both approaches require the availability of supervised or weakly supervised information.

The third challenge is the most complex, as humans also experience perceptual aliasing, though to a lesser extent than DL models. This aliasing is particularly evident in images dominated by the sky or grass, which lack distinctive landmarks or keypoints that would help the model generate unique feature representations and differentiate between scenes. Lu et al. [17] propose leveraging the images within a training batch in a way that allows them to inform and enhance each other's representations. This approach can address this challenge, as well as the previous two, simultaneously. Specifically, images from the same location can promote invariance to conditions and viewpoints, while images from diverse locations can increase discriminativeness. Complementary to this, Zaffar et al. [16] suggest a test-time uncertainty estimation method that uses the locations within a reference set $\mathcal{R}$ as an indicator of perceptual aliasing, helping to prevent critical system failures.

3.3 Connection to Deep Metric Learning

Metric Learning [24, 25], or Deep Metric Learning in the context of DL, is a branch of Machine Learning that involves learning an embedding space where the similarity

between objects is preserved. The objective is to learn feature representations that bring similar objects closer together while pushing dissimilar ones farther apart.

The VPR literature has heavily drawn on concepts from Metric Learning, particularly in the development and use of loss functions. These functions depend on positive and negative pairs. A *positive pair* consists of two images representing the same place, while a *negative pair* consists of images depicting different places. The loss functions presented in the following subsections all have a common objective: to reduce the distance between positive pairs while increasing the distance between negative pairs.

3.3.1 Triplet Loss

The Triplet Loss was defined by Hoffer et al. [26] in 2015. Given a tiplet of images (I_q, I_p, I_n), where (I_q, I_p) forms a positive pair and (I_q, I_n) forms a negative pair, the loss is computed as:

$$\mathcal{L}_{\text{TL}} = \max\left(d_p - d_n + m, 0\right)$$

where $d_p = d(\mathbf{e}_p, \mathbf{e}_q)$ and $d_n = d(\mathbf{e}_n, \mathbf{e}_q)$ represent the distances between the corresponding embeddings, and m is a margin hyperparameter.

3.3.2 Weakly Supervised Triplet Ranking Loss

Many datasets in Visual Place Recognition provide only weak supervision, where images that are considered close based on their (noisy) location labels may actually correspond to different scenes or places. In 2016, Arandjelovic et al. [15] introduced a new loss function designed to address this challenge. Given a tuple $\left(I_q, \{I_p^i\}, \{I_n^j\}\right)$ where $\{I_p^i\}$ represents a set of potential positive images and $\{I_n^j\}$ a set of definite negative images, the loss function is computed as:

$$\mathcal{L}_{\text{WTL}} = \sum_j \max\left(m - (d_n^j)^2 + \min_i (d_p^i)^2, 0\right)$$

with $d_p^i = d(\mathbf{e}_p^i, \mathbf{e}_q)\,\forall i$ and $d_n^j = d(\mathbf{e}_n^j, \mathbf{e}_q)\,\forall j$. Here, m is a margin hyperparameter, similar to the one used in the loss function presented in Section 3.3.1.

3.3.3 Contrastive Loss

The Contrastive Loss was first introduced by Hadsell et al. [27] in 2006 for training *Siamese* networks. It operates on pairs of images and is defined as:

$$\mathcal{L}_{\text{CL}} = y_{ij}\left[\frac{1}{2} d(\mathbf{e}_i, \mathbf{e}_j)^2\right] + (1 - y_{ij})\left[\frac{1}{2}\max\left(m - d(\mathbf{e}_i, \mathbf{e}_j), 0\right)^2\right]$$

where $y_{ij} = 1$ if the pair (I_i, I_j) is positive, and $y_{ij} = 0$ otherwise. The margin hyperparameter m prevents the distance between dissimilar images from growing without bound.

3.3.4 Generalized Contrastive Loss

Leyva-Vallina et al. [23] in 2023 argued that using a binary ground truth value to assess similarity between image pairs could lead to unreliable predictions for the VPR task. To address this, they propose estimating the similarity between images using continuous values, denoted as $\psi_{ij} \in [0, 1]$. The formulation remains the same as presented in Section 3.3.3, but the ground truth is replaced by the similarity score ψ_{ij}. The loss function is then given by:

$$\mathcal{L}_{\text{GCL}} = \psi_{ij} \left[\frac{1}{2} d(\mathbf{e}_i, \mathbf{e}_j)^2 \right] + (1 - \psi_{ij}) \left[\frac{1}{2} \max\left(m - d(\mathbf{e}_i, \mathbf{e}_j), 0\right)^2 \right]$$

The similarity scores, ψ_{ij}, are computed by measuring the overlap of the Field of View (FoV) or 3D pointclouds between image pairs.

3.3.5 Multi-Similarity Loss

Unlike previous loss functions, Wang et al. [28] in 2019 proposed a novel loss based on similarity scores computed via the dot product $S_{ij} \coloneqq \mathbf{e}_i^T \mathbf{e}_j$. For a batch of M images, the loss function is defined as:

$$\mathcal{L}_{\text{MSL}} = \frac{1}{M} \sum_{k=1}^{M} \left\{ \frac{1}{\alpha} \log \left[1 + \sum_{i \in \mathcal{P}_k} e^{-\alpha(S_{ki} - m)} \right] + \frac{1}{\beta} \log \left[1 + \sum_{j \in \mathcal{N}_k} e^{\beta(S_{kj} - m)} \right] \right\}$$

with $\mathcal{P}_k$ denotes the set of indices in the batch that correspond to positive pairs with the k-th image, while $\mathcal{N}_k$ represents the set of indices for negative pairs. The hyperparameters α, β and m guide the weighting scheme of the loss.

3.4 Evaluation Metrics

To evaluate and compare the performance of Visual Place Recognition models, the standard metric used is Recall@K on a given dataset. For a specified distance threshold τ in meters, the Recall@K measures the percentage of queries for which at least one of the top-K retrieved database images–determined by the K-Nearest Neighbors algorithm–is within τ meters of the ground-truth location of the query. A common value for τ is 25 meters.

Chapter 4

Generative Artificial Intelligence

Generative Artificial Intelligence (Gen-AI) is a key branch of AI with a wide range of real-world applications. In drug discovery, it plays a crucial role by designing novel molecules and optimizing their properties, accelerating the development of new treatments. In medical imaging, Gen-AI aids in enhancing diagnostic accuracy by generating and simulating medical images, offering more precise insights for healthcare professionals. Beyond healthcare, Gen-AI contributes to improving accessibility through advanced text-to-speech and speech-to-text systems, promoting greater inclusivity. It is also used in data augmentation, generating synthetic data to enrich training datasets and enhance the performance of AI models. With its impact across content creation, medical imaging, virtual environments, and more, Gen-AI is driving forward both the capabilities of AI and its real-world applications.

The general goal of Generative Artificial Intelligence can be summarized as follows: a Gen-AI model aims to learn a data distribution, which is represented by a *probability density function* in the continuous case and by a *probability mass function* in the discrete case, denoted as $p_{\text{data}}(\mathbf{x})$. To train the model, a finite set of samples $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ is drawn from the unknown data distribution $p_{\text{data}}(\mathbf{x})$. The goal is for the model to learn a distribution $p_{\text{model}}(\mathbf{x})$ such that samples $\hat{\mathbf{x}}_i \sim p_{\text{model}}(\mathbf{x})$, generated from this learned distribution, closely resemble the true data distribution $p_{\text{data}}(\mathbf{x})$.

A variety of models have been proposed in the literature to address this goal, with distinctions made between unimodal [29] and multimodal [30, 31] approaches. A multimodal system can process multiple types of inputs simultaneously, referred to as *modalities* (such as text, speech, and visual data), or generate multiple modalities as outputs. In contrast, a unimodal system is limited to working with a

single modality.

Remarkable progress has been made thanks to key innovations such as VAE [32], VQ-VAE [33], GAN [34, 35, 36, 37, 38], and Diffusion Probabilistic Models [39, 40, 41, 42]. While this work focuses on the latter, the earlier works laid the foundational for subsequent advancements, and are therefore discussed in the following sections.

4.1 Variational Auto-Encoder

The Variational Auto-Encoder (VAE) was introduced in 2013 by Kingma in the paper «Auto-encoding variational bayes» [32]. The work is based on the assumption that the dataset $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ consists of i.i.d. (independent and identically distributed) samples generated by a random process from an unobserved continuous latent random variable $\mathbf{z}$. If we had access to $\mathbf{z}$, we could recover the marginal likelihood of the data, $p_{\text{data}}(\mathbf{x})$, as follows:

$$p_{\text{data}}(\mathbf{x}) \coloneqq p_{\Theta^*}(\mathbf{x}) = \int p_{\Theta^*}(\mathbf{x} \mid \mathbf{z}) p_{\Theta^*}(\mathbf{z})\, d\mathbf{z} \tag{4.1}$$

where we assume that both the prior $p_{\Theta^*}(\mathbf{z})$ and the likelihood $p_{\Theta^*}(\mathbf{x} \mid \mathbf{z})$ are from parametric families of distributions. However, since both the latent variable $\mathbf{z}$ and the true parameters Θ^* are unknown, and the integral is typically intractable, the question becomes how to effectively perform inference and learn the parameters of the distributions. To address this, Kingma proposed a solution based on the Auto-Encoder.

4.1.1 Auto-Encoder

Auto-Encoder (AE) [43] is Neural Network architecture with various applications, which can be trained either in a supervised or unsupervised manner. In its original form, the model learns to reconstruct its input $\mathbf{x} \in \mathbb{R}^d$, producing an output $\hat{\mathbf{x}} \in \mathbb{R}^d$, by passing through an intermediate representation $\mathbf{z} \in \mathbb{R}^h$, minimizing the distance $d(\mathbf{x}, \hat{\mathbf{x}})$ between the input and the reconstruction. The architecture is illustrated in Fig. 4.1. It consists of two main components: an *encoder* that implements a mapping function $\mathcal{E} : \mathbb{R}^d \to \mathbb{R}^h$, and a *decoder* with a reverse mapping function $\mathcal{D} : \mathbb{R}^h \to \mathbb{R}^d$. Depending on the specific task, both the encoder and decoder can be realized using either a Multi-Layer Perceptron or a Convolutional Neural Network.

In 2006, Hinton et al. [44] employed this architecture to reduce the dimensionality of the data from a d-dimensional space to a much smaller h-dimensional space ($h \ll d$), while preserving the most important features and characteristics of the data. Additionally, Bengio et al. [45] utilized similar models for pre-training the layers of a Neural Network, improving the initialization of the optimization process and enhancing the model's generalization performance.

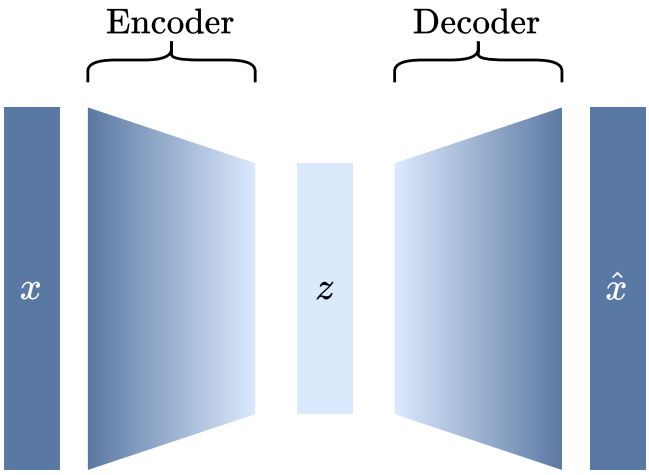


Figure 4.1: Auto-Encoder model.

More recently, the role of the decoder has been extended from merely reconstructing data to mapping to different data distributions, as demonstrated in tasks like semantic segmentation [46]. Furthermore, these architectures have found a wide range of applications, including denoising [47, 48], compression [49], and anomaly detection [50], among others.

4.1.2 Variational Lower Bound

Due to the intractability of Eq. 4.1, Kingma suggests an alternative method for computing the marginal distribution $p_\Theta(\mathbf{x})$ using the Bayes' rule, as follows:

$$p_\Theta(\mathbf{x}) = \frac{p_\Theta(\mathbf{x} \mid \mathbf{z}) p_\Theta(\mathbf{z})}{p_\Theta(\mathbf{z} \mid \mathbf{x})} \approx \frac{p_\Theta(\mathbf{x} \mid \mathbf{z}) p_\Theta(\mathbf{z})}{q_\Phi(\mathbf{z} \mid \mathbf{x})}$$

Here, $q_\Phi(\mathbf{z} \mid \mathbf{x})$ represents the encoder, which approximates the intractable true posterior $p_\Theta(\mathbf{z} \mid \mathbf{x})$, and $p_\Theta(\mathbf{x} \mid \mathbf{z})$ is modeled by the decoder. By considering the marginal log-likelihood and recognizing the independence of the marginal from $\mathbf{z}$, we can derive the following:

$$
\begin{aligned}
\log p_\Theta(\mathbf{x}) &= \mathbb{E}_{\mathbf{z}\sim q_\Phi(\mathbf{z}|\mathbf{x})}\left[\log p_\Theta(\mathbf{x})\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\Phi(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_\Theta(\mathbf{x}\mid\mathbf{z})p_\Theta(\mathbf{z})}{p_\Theta(\mathbf{z}\mid\mathbf{x})}\right)\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\Phi(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_\Theta(\mathbf{x}\mid\mathbf{z})p_\Theta(\mathbf{z})}{p_\Theta(\mathbf{z}\mid\mathbf{x})}\cdot\frac{q_\Phi(\mathbf{z}\mid\mathbf{x})}{q_\Phi(\mathbf{z}\mid\mathbf{x})}\right)\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\Phi(\mathbf{z}|\mathbf{x})}\left[\log p_\Theta(\mathbf{x}\mid\mathbf{z})\right] - \mathbb{E}_{\mathbf{z}\sim q_\Phi(\mathbf{z}|\mathbf{x})}\left[\log\frac{q_\Phi(\mathbf{z}\mid\mathbf{x})}{p_\Theta(\mathbf{z})}\right] \\
&\quad + \mathbb{E}_{\mathbf{z}\sim q_\Phi(\mathbf{z}|\mathbf{x})}\left[\log\frac{q_\Phi(\mathbf{z}\mid\mathbf{x})}{p_\Theta(\mathbf{z}\mid\mathbf{x})}\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\Phi(\mathbf{z}|\mathbf{x})}\left[\log p_\Theta(\mathbf{x}\mid\mathbf{z})\right] - D_{KL}\left(q_\Phi(\mathbf{z}\mid\mathbf{x})\,||\,p_\Theta(\mathbf{z})\right) \\
&\quad + D_{KL}\left(q_\Phi(\mathbf{z}\mid\mathbf{x})\,||\,p_\Theta(\mathbf{z}\mid\mathbf{x})\right)
\end{aligned}
$$

In this equations, $D_{KL}\left(P\,||\,Q\right)$ denotes the Kullback-Leibler divergence between distributions P and Q. The first expectation is computed via sampling and is differentiable due to the reparametrization trick[1]. The second term represents the KL divergence between the approximate and the true prior, which are often chosen to be Gaussian distributions, allowing this term to be computed in closed form. The third term, however, involves the intractable true posterio $p_\Theta(\mathbf{z}\mid\mathbf{x})$, which cannot be directly computed. Since the KL divergence is non-negative[2], we can still optimize the sets of parameters Θ and Φ by maximizing the following *variational lower bound* (ELBO):

$$
\log p_\Theta(\mathbf{x}) \geq \mathcal{L}\left(\mathbf{x};\Theta,\Phi\right) = \mathbb{E}_{\mathbf{z}\sim q_\Phi(\mathbf{z}|\mathbf{x})}\left[\log p_\Theta(\mathbf{x}\mid\mathbf{z})\right] - D_{KL}\left(q_\Phi(\mathbf{z}\mid\mathbf{x})\,||\,p_\Theta(\mathbf{z})\right) \quad (4.2)
$$

By setting the prior $p_\Theta(\mathbf{z})$ to a Gaussian distribution and training according to Eq. 4.2, once the parameters Θ and Φ are fully optimized, we can sample from $p_\Theta(\mathbf{z})$, pass the samples through the decoder $p_\Theta(\mathbf{x}\mid\mathbf{z})$, and generate new data.

4.1.3 Vector Quantised-Variational Auto-Encoder

In 2017, Van Den Oord et al. [33] extented the VAE framework to work with discrete latent representations that represent a more realistic fit to real world tasks like both spoken and written language, reasoning, planning, and others. They propose modifying the encoder architecture to predict discrete codes instead of continuous latent variables. Additionally, they introduce a learned prior over the

[1] For further details, see Section 2.4 of [32]

[2] See Appendix A.1 for the proof.

discrete codes, which is typically fixed in standard VAEs. The prior is learned using autoregressive distributions over the latent codes, but only after the discrete codes have been trained.

4.2 Generative Adversarial Networks

The introduction of the Generative Adversarial Network (GAN) framework by Goodfellow et al. [34] in 2014 marked a major breakthrough in image generation. Since then, Deep Learning models inspired by this framework have dominated the State-Of-The-Art (SOTA) in various domains.

The concept behind GANs originates from *Game Theory*, a field of mathematics that studies strategic interactions between rational agents using mathematical models. Specifically, GANs are based on the *minimax two-player game*, where two players engage in a game where each aims to maximize their own gain while minimizing their potential losses.

Goodfellow et al. [34] formalized this game-theoretic framework within the context of Gen-AI by defining two models: a generative model G, which seeks to learn the true data distribution $p_{\text{data}}(\mathbf{x})$, and a discriminative model D, whose task is to distinguish between real and generated samples, *i.e.*, samples from the true data distribution and those produced by the generator.

The generator $G(\mathbf{z}) \coloneqq G(\mathbf{z}; \Theta_G)$ takes noise inputs $\mathbf{z}$ from a prior distribution $p_{\mathbf{z}}(\mathbf{z})$ and maps them to the data space. Here, Θ_G represents the parameters of the generator network. In contrast, the discriminator $D(\mathbf{x}) \coloneqq D(\mathbf{x}; \Theta_D)$ operates directly on the data space, outputting a score that indicates the probability that a given input $\mathbf{x}$ is a real sample (*i.e.*, drawn from the true data distribution) rather than a generated one. The parameters of the discriminator network are denoted by Θ_D. The framework is illustrated in Fig. 4.2[3].

The learning objective can be formulated as:

$$\min_{\Theta_G} \max_{\Theta_D} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} \log \left(1 - D(G(\mathbf{z}))\right) \right]$$

In this formulation, the discriminator seeks to maximize both terms by distinguishing real data samples from generated ones. On the other hand, the generator minimizes the second term, aiming to produce more convincing samples that are difficult for the discriminator to distinguish from real data.

Goodfellow et al. [34] observed that early in training, when the generator G produces poor-looking images, the discriminator D can easily distinguish real from

[3]The images were generated using a LDM (see Section 4.3.3) model, conditioned on DINOv2 SALAD [51] embeddings derived from real images in the SF-XL dataset [52].

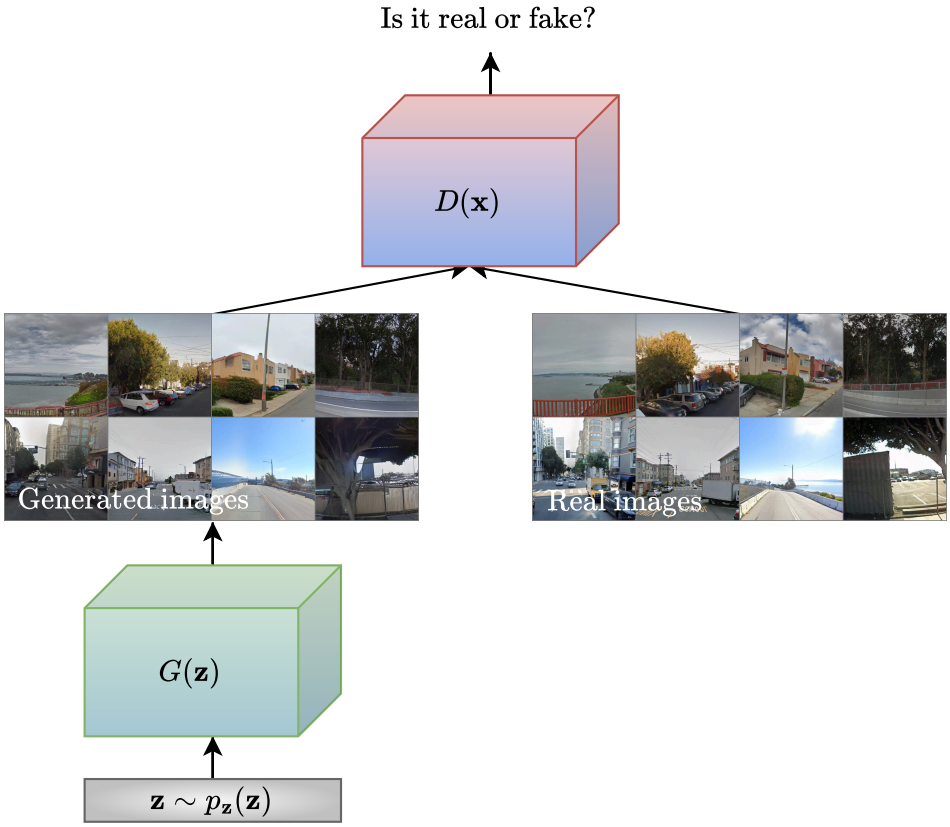


Figure 4.2: Generative Adversarial Network framework.

fake samples, leading to high-confidence decisions. This, however, results in weak gradient signals that do not effectively help improve the generator. To address this issue, they proposed an alternative objective in which the generator attempts to maximize the likelihood that the discriminator makes incorrect predictions:

$$\max_{\Theta_D} \left[\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})} \log\left(1 - D(G(\mathbf{z}))\right)\right]$$
$$\max_{\Theta_G} \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})} \log\left(D(G(\mathbf{z}))\right)$$

The training process utilizes the Stochastic Gradient Descent (SGD) optimization algorithm, alternating between updates to the discriminator and the generator weights. The complete training procedure is outlined in Algorithm 1.

Algorithm 1 SGD optimization of GANs.

/* k is the number of training steps to apply to the discriminator */
/* M is the batch size */
/* η_D is the learning rate for the discriminator */
/* η_G is the learning rate for the generator */
for number of iterations **do**
    for k steps **do**
        ▷ Sample M noise samples $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_M\}$ from the prior $p_{\mathbf{z}}(\mathbf{z})$
        ▷ Sample M data samples $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ from the real data distribution $p_{\text{data}}(\mathbf{x})$
        ▷ Update the discriminator by stochastic gradient ascent
        $$\Theta_D \leftarrow \Theta_D + \eta_D \nabla_{\Theta_D} \frac{1}{M} \sum_{i=1}^{M} [\log D(\mathbf{x}_i) + \log (1 - D(G(\mathbf{z}_i)))]$$
    end for
    ▷ Sample M noise samples $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_M\}$ from the prior $p_{\mathbf{z}}(\mathbf{z})$
    ▷ Update the generator by stochastic gradient ascent
    $$\Theta_G \leftarrow \Theta_G + \eta_G \nabla_{\Theta_G} \frac{1}{M} \sum_{i=1}^{M} \log D(G(\mathbf{z}_i))$$
end for

4.3 Diffusion Probabilistic Models

The framework outlined in Section 4.2 has significantly advanced the field of Generative Artificial Intelligence. However, the algorithm presented in Algorithm 1 is prone to training instability due to the simultaneous training of two networks. Additionally, the generator network may converge to a local minimum in the weight space, where it produces only a limited set of outputs (often just one or a few), failing to capture the full diversity of the source data distribution. This issue, known as *mode collapse*, has been widely discussed in the literature [38].

In 2015, Sohl-Dickstein et al. [53] introduced a new class of probabilistic models, called *Diffusion Probabilistic Models*, inspired by principles from nonequilibrium thermodynamics. As demonstrated by Ho et al. in «Denoising diffusion probabilistic models» [41], this approach can generate high-quality image samples. Since their introduction, Diffusion Probabilistic Models have been widely adopted for various tasks due to their stable training dynamics, diverse outputs, and exceptional image quality.

4.3.1 Denoising Diffusion Probabilistic Models

A *diffusion process* is a stochastic process that gradually adds noise to a signal over time, typically through a *Markov chain*[4], until the original signal is indistinguishable from noise. This can be thought of as a process that evolves the signal in such a way that, in the limit, the signal is lost.

A Diffusion Model (DM) [53] build upon this concept. Given data $\mathbf{x}_0$ that follows the distribution $q(\mathbf{x}_0)$ (*i.e.* $\mathbf{x}_0 \sim q(\mathbf{x}_0)$), DMs are latent variable models described by the following marginal distribution:

$$p_\Theta(\mathbf{x}_0) \coloneqq \int p_\Theta(\mathbf{x}_{0:T})\, d\mathbf{x}_{1:T}$$

where $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$ are latents of the same dimensionality as $\mathbf{x}_0$. The join distribution $p_\Theta(\mathbf{x}_{0:T})$, referred to as the *reverse process*, is defined by a Markov chain with learned Gaussian transitions starting from $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$. Specifically, it is expressed as:

$$p_\Theta(\mathbf{x}_{0:T}) \coloneqq p(\mathbf{x}_T) \prod_{t=1}^{T} p_\Theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

where $p_\Theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \coloneqq \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\Theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\Theta(\mathbf{x}_t, t))$. A distinctive feature of DMs is that the posterior $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$, known as the *forward process* (or *diffusion process*), is a fixed Markov chain that progressively adds Gaussian noise according to a variance schedule $\beta_1, \beta_2, \ldots, \beta_T$. This process is defined as:

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) \coloneqq \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

where $q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \coloneqq \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$. The Markovian diffusion process is depicted schematically in Fig. 4.3.

With this setup, we can train the model using the typical variational lower bound (see Eq. 4.2 in Section 4.1.2), which is given by:

$$\begin{aligned}
\mathbb{E}_q\left[-\log p_\Theta(\mathbf{x}_0)\right] &\leq \mathbb{E}_q\left[-\log \frac{p_\Theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right] \\
&= \mathbb{E}_q\left[-\log\left(p(\mathbf{x}_T)\prod_{t=1}^{T}\frac{p_\Theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}\right)\right] \\
&= \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t=1}^{T}\log\frac{p_\Theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}\right] =: \mathcal{L}
\end{aligned} \tag{4.3}$$

[4] A Markov chain is a stochastic process in which the probability of each state depends only on the previous state. This property is known as the *Markov property*, which means that the process has no memory of past events beyond the most recent one.

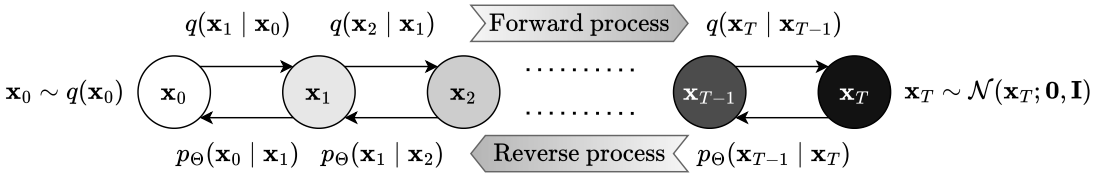


Figure 4.3: Scheme of the Markovian diffusion process. Adapted from «Denoising diffusion probabilistic models» [41].

The forward process allows for sampling $\mathbf{x}_t$ at an arbitrary timestep t in closed form. To illustrate this, we first define $\alpha_t \coloneqq 1-\beta_t$ and apply the reparameterization trick [34] to rewrite the following expression:

$$\begin{aligned} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_{t-1}, (1-\alpha_t)\mathbf{I}) \\ &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\mathcal{N}(\mathbf{0}, \mathbf{I}) \\ &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon} \end{aligned}$$

Thus, during the forward process, $\mathbf{x}_t$ is sampled starting from $\mathbf{x}_{t-1}$ as follows:

$$\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t$$

where $\boldsymbol{\epsilon}_t$ is sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$. We can recursively express $\mathbf{x}_{t-1}$ as follows:

$$\begin{aligned} \mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t \\ &= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_{t-1}}\boldsymbol{\epsilon}_{t-1}) + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t \\ &= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\boldsymbol{\epsilon}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t \\ &= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1}) + (1-\alpha_t)}\boldsymbol{\epsilon}_{t,t-1} \\ &= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\boldsymbol{\epsilon}_{t,t-1} \end{aligned}$$

Here, $\boldsymbol{\epsilon}_t$ and $\boldsymbol{\epsilon}_{t-1}$ are independent Gaussian noise terms. When we combine them, the result is a new Gaussian noise term $\boldsymbol{\epsilon}_{t,t-1}$, whose variance is the sum of the individual variances of $\boldsymbol{\epsilon}_t$ and $\boldsymbol{\epsilon}_{t-1}$. This process can be applied recursively for each timestep t until we reach $\mathbf{x}_0$, yielding the following expression for $\mathbf{x}_t$:

$$\mathbf{x}_t = \sqrt{\alpha_t\alpha_{t-1}\dots\alpha_1}\mathbf{x}_0 + \sqrt{1-\alpha_t\alpha_{t-1}\dots\alpha_1}\boldsymbol{\epsilon}_{t,t-1,\dots,1}$$

Finally, defining $\bar{\alpha}_t \coloneqq \prod_{s=1}^{t}\alpha_s$, we obtain:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_{t,t-1,\dots,1} \tag{4.4}$$

or equivalently, the distribution for $\mathbf{x}_t$ given $\mathbf{x}_0$ is:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{4.5}$$

The ability to sample $\mathbf{x}_t$ directly from $\mathbf{x}_0$ enables the optimization of random terms in the variational lower bound $\mathcal{L}$ using Stochastic Gradient Descent. Specifically, Eq. 4.3 can be rewritten in a way that promotes variance reduction, as follows:

$$\begin{aligned}
\mathcal{L} &= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t=1}^{T} \log \frac{p_\Theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right] \\
&= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t>1} \log \frac{p_\Theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} - \log \frac{p_\Theta(\mathbf{x}_0 \mid \mathbf{x}_1)}{q(\mathbf{x}_1 \mid \mathbf{x}_0)} \right] \\
&= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t>1} \log \left(\frac{p_\Theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)} \cdot \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)} \right) - \log \frac{p_\Theta(\mathbf{x}_0 \mid \mathbf{x}_1)}{q(\mathbf{x}_1 \mid \mathbf{x}_0)} \right] \\
&= \mathbb{E}_q \left[-\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T \mid \mathbf{x}_0)} - \sum_{t>1} \log \frac{p_\Theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)} - \log p_\Theta(\mathbf{x}_0 \mid \mathbf{x}_1) \right] \\
&= D_{KL}\left(q(\mathbf{x}_T \mid \mathbf{x}_0) \,||\, p(\mathbf{x}_T)\right) + \sum_{t>1} D_{KL}\left(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,||\, p_\Theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\right) \\
&\quad - \log p_\Theta(\mathbf{x}_0 \mid \mathbf{x}_1) = \mathcal{L}_T + \sum_{t>1} \mathcal{L}_{t-1} + \mathcal{L}_0
\end{aligned} \tag{4.6}$$

Here, the KL divergences can be computed in closed form since they only involve Gaussian distributions. This is due to the fact that the posteriors $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ are tractable when conditioned on $\mathbf{x}_0$, as shown below:

$$\begin{aligned}
q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) &= \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0) q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)} = \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)} \\
&= \frac{\mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_{t-1}, (1-\alpha_t)\mathbf{I})\, \mathcal{N}(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0, (1-\bar{\alpha}_{t-1})\mathbf{I})}{\mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})} \\
&= \mathcal{N}\left(\mathbf{x}_{t-1}; \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t, \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t\mathbf{I}\right) \\
&= \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I})
\end{aligned}$$

The derivation of the third equation in the previous block of equations is left to the reader[5].

Although the variances $\beta_1, \beta_2, \ldots, \beta_T$ could be learned through reparameterization, Ho et al. [41] opted to fix them, ensuring that the forward process contains

[5]The derivation is a long sequence of algebraic manipulations, which should be straightforward, though it may require some patience. I trust you ♡.

no learnable parameters. As a result, the term $\mathcal{L}_T$ in Eq. 4.6 can be disregarded. Furthermore, they chose to set $\boldsymbol{\Sigma}_\Theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$ in $p_\Theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$, where $\sigma_t^2 = \tilde{\beta}_t$.

Given the specific forms of $p_\Theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ and $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$, the KL divergence $\mathcal{L}_{t-1}$ between the two Gaussian distributions is:

$$\mathcal{L}_{t-1} = \mathbb{E}_q \left[\frac{1}{2\tilde{\beta}_t} \left\| \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\Theta(\mathbf{x}_t, t) \right\|^2 \right] + C$$

where C is a constant. Since we know the form of $\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0)$, we can express $\boldsymbol{\mu}_\Theta(\mathbf{x}_t, t)$ similarly:

$$\boldsymbol{\mu}_\Theta(\mathbf{x}_t, t) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t} \mathbf{x}_\Theta(\mathbf{x}_t, t) + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \mathbf{x}_t$$

Substituting this into the expression for $\mathcal{L}_{t-1}$, we obtain:

$$\mathcal{L}_{t-1} - C = \mathbb{E}_q \left[\frac{\beta_t \bar{\alpha}_{t-1}}{2(1 - \bar{\alpha}_t)(1 - \bar{\alpha}_{t-1})} \left\| \mathbf{x}_\Theta(\mathbf{x}_t, t) - \mathbf{x}_0 \right\|^2 \right]$$

Next, we can simplify this further, recognizing that $\mathbf{x}_t$ is an available input to the model during training. Since $\mathbf{x}_t = \mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}) = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}$ from Eq. 4.4, where $\boldsymbol{\epsilon}$ sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, we have:

$$\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}) - \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon} \right) \tag{4.7}$$

This allows us to rewrite the mean of the posterior $\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0)$ as:

$$\begin{aligned}
\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) &= \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t} \cdot \frac{1}{\sqrt{\bar{\alpha}_t}} \left(\mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}) - \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon} \right) + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}) \\
&= \left(\frac{\beta_t}{\sqrt{\alpha_t}(1 - \bar{\alpha}_t)} + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \right) \mathbf{x}_t(\mathbf{x}_0, \boldsymbol{\epsilon}) - \frac{\beta_t \sqrt{1 - \bar{\alpha}_t}}{\sqrt{\alpha_t}(1 - \bar{\alpha}_t)} \boldsymbol{\epsilon}
\end{aligned}$$

The same can be done for $\boldsymbol{\mu}_\Theta(\mathbf{x}_t, t)$, so we can write $\mathcal{L}_{t-1}$ as:

$$\begin{aligned}
\mathcal{L}_{t-1} - C &= \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\epsilon}} \left[\frac{\beta_t \bar{\alpha}_{t-1}}{2(1 - \bar{\alpha}_t)(1 - \bar{\alpha}_{t-1})} \cdot \frac{1 - \bar{\alpha}_t}{\bar{\alpha}_t} \left\| \boldsymbol{\epsilon}_\Theta(\mathbf{x}_t, t) - \boldsymbol{\epsilon} \right\|^2 \right] \\
&= \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\epsilon}} \left[\frac{\beta_t^2}{2\tilde{\beta}_t \alpha_t (1 - \bar{\alpha}_t)} \left\| \boldsymbol{\epsilon}_\Theta(\mathbf{x}_t, t) - \boldsymbol{\epsilon} \right\|^2 \right]
\end{aligned} \tag{4.8}$$

Thus, the model is designed to approximate the noise $\boldsymbol{\epsilon}$ defined in Eq. 4.4, by learning the function $\boldsymbol{\epsilon}_\Theta(\mathbf{x}_t, t)$.

At timestep $t = 0$, the goal of the model's reverse process is to predict the image $\mathbf{x}_0$ given $\mathbf{x}_1$ based on the distribution $p_\Theta(\mathbf{x}_0 \mid \mathbf{x}_1)$. This is captured during training

by the term $\mathcal{L}_0$ in Eq. 4.6. Since images are rescaled from integer values in the set $\{0, 1, \ldots, 255\}$ to range $[-1, 1] \subset \mathbb{R}$, Ho et al. [41] suggested computing $\mathcal{L}_0$ using a discrete decoder that assumes independence across the data dimensions. This approach is defined as follows:

$$p_\Theta(\mathbf{x}_0 \mid \mathbf{x}_1) = \prod_{i=1}^{D} \int_{\delta_-(x_0^i)}^{\delta_+(x_0^i)} \mathcal{N}(x; \mu_\Theta^i(\mathbf{x}_1, 1), \sigma_1^2)\, dx$$

$$\delta_+(x) = \begin{cases} \infty & \text{if } x = 1 \\ x + \frac{1}{255} & \text{if } x < 1 \end{cases} \qquad \delta_-(x) = \begin{cases} -\infty & \text{if } x = -1 \\ x - \frac{1}{255} & \text{if } x > -1 \end{cases}$$

where D is the dimensionality of the data and i refers to the i-th coordinate. This formulation enables training, with the reverse process using $\boldsymbol{\mu}_\Theta(\mathbf{x}_1, 1)$ during sampling.

Finally, Ho et al. [41] found that simplifying Eq. 4.8 by disregarding the weights resulted in both improved sample quality and simpler training. Specifically, they proposed optimizing the following function:

$$\begin{aligned} \mathcal{L}_{\text{simple}}(\Theta) &:= \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\epsilon}} \left[\|\boldsymbol{\epsilon}_\Theta(\mathbf{x}_t, t) - \boldsymbol{\epsilon}\|^2 \right] \\ &= \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\epsilon}} \left[\left\| \boldsymbol{\epsilon}_\Theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, t) - \boldsymbol{\epsilon} \right\|^2 \right] \end{aligned} \tag{4.9}$$

This approach leads to a weighted variational bound that places different emphasis on various aspects of reconstruction. In particular, terms associated with smaller timestep values are down-weighted, as they deal with denoising data that has very little noise. This allows the model to concentrate more on the more challenging denoising tasks associated with larger t-values. Additionally, the term $\mathcal{L}_0$ is simplified by approximating it using the Gaussian probability density function multiplied by the bin width.

Algorithm 2 Training DDPM models.

1: /* T is the total number of timesteps */
2: /* η is the learning rate */
3: **while** not converged **do**
4: ▷ Sample data $\mathbf{x}_0$ from $q(\mathbf{x}_0)$
5: ▷ Sample timestep t from Uniform($\{1, 2, \ldots, T\}$)
6: ▷ Sample noise $\boldsymbol{\epsilon}$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$
7: ▷ Update the model by taking gradient descent step
8: $\Theta \leftarrow \Theta - \eta \nabla_\Theta \left\| \boldsymbol{\epsilon}_\Theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, t) - \boldsymbol{\epsilon} \right\|^2$
9: **end while**

The complete procedures for training and sampling are outlined in Algorithm 2 and Algorithm 3, respectively. The forward and reverse processes are visually

Algorithm 3 Sampling from DDPM models.

/* T is the total number of timesteps */
▷ Sample $\mathbf{x}_T$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, T-1, \ldots, 1$ **do**
    if $t > 1$ **then**
        ▷ Sample $\mathbf{z}$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$
    else
        $\mathbf{z} \leftarrow \mathbf{0}$
    end if
    $\mathbf{x}_{t-1} \leftarrow \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\Theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

represented in Fig.4.4 and Fig.4.5. Specifically, Fig. 4.4 illustrates how noise is progressively added, eventually destroying the original image signals, while Fig. 4.5 depicts the gradual removal of noise over successive timesteps, leading to the generation of images[6].

4.3.2 Denoising Diffusion Implicit Models

Although DDPM models can generate images of comparable or even superior quality to GANs, they are constrained by their sequential nature. The reverse process must traverse all timesteps, starting from pure noise at timestep T and progressively denoising at each step until a data sample $\mathbf{x}_0$ is obtained. To ensure that the reverse process can be approximately modeled with Gaussian distributions, the number of timesteps T needs to be large. This requirement significantly limits the models' inference speed. As a result, DDPM models are orders of magnitude slower than GANs, which only require a single forward pass to generate new images.

Song et al. [40] proposed overcoming this limitation by extending the forward process to non-Markovian diffusion processes. Their reasoning is based on the observation that the objective in Eq. 4.8 depends only on the marginals $q(\mathbf{x}_t \mid \mathbf{x}_0)$. This implies that, as long as these marginals remain the same, the joint distribution $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$ can be modeled in any desired way. Specifically, they represented the

[6] In both figures, the images were generated using a LDM model (see Section 4.3.3), conditioned on CosPlace [52] with a ResNet-50 backbone and a descriptor dimension of 2048. The embeddings were derived from real images in the SF-XL dataset [52].

Figure 4.4: Illustration of the forward process.

joint distribution using a vector of standard deviations $\boldsymbol{\sigma} \in \mathbb{R}^T$ as follows:

$$q_{\boldsymbol{\sigma}}(\mathbf{x}_{1:T} \mid \mathbf{x}_0) \coloneqq q_{\boldsymbol{\sigma}}(\mathbf{x}_T \mid \mathbf{x}_0) \prod_{t=2}^{T} q_{\boldsymbol{\sigma}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$$

where the conditional distributions are given by:

$$q_{\boldsymbol{\sigma}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1} - \sigma_t^2} \cdot \frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}}, \sigma_t^2 \mathbf{I}\right)$$
$$q_{\boldsymbol{\sigma}}(\mathbf{x}_T \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_T; \sqrt{\bar{\alpha}_T}\mathbf{x}_0, (1 - \bar{\alpha}_T)\mathbf{I})$$

where the mean function is designed to ensure that the marginals in Eq. 4.5 are matched as intended. Under this framework, the forward process can be derived using the Bayes' rule:

$$q_{\boldsymbol{\sigma}}(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0) = \frac{q_{\boldsymbol{\sigma}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) q_{\boldsymbol{\sigma}}(\mathbf{x}_t \mid \mathbf{x}_0)}{q_{\boldsymbol{\sigma}}(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}$$

This formulation allows for flexible modeling of the dependencies of $\mathbf{x}_t$ on both $\mathbf{x}_{t-1}$ and $\mathbf{x}_0$, as well as the level of stochasticity in the forward process, which can

Figure 4.5: Illustration of the reverse process.

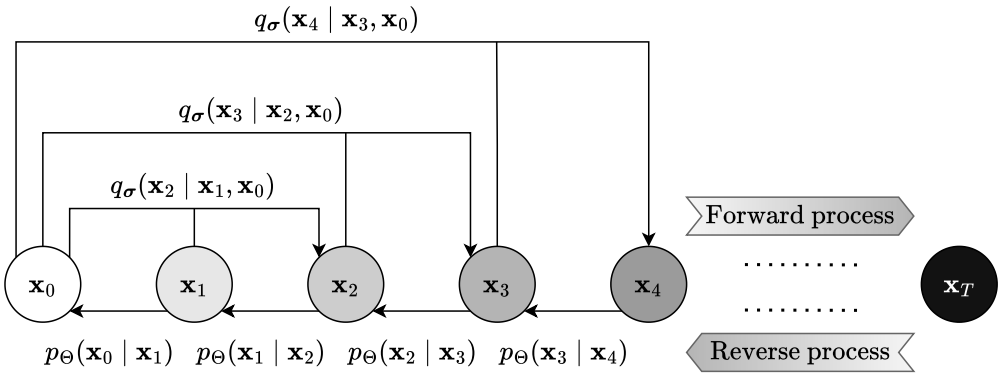


Figure 4.6: Scheme of the non-Markovian diffusion process. Adapted from «Denoising diffusion implicit models» [40].

be controlled by the vector $\boldsymbol{\sigma}$. A schematic representation of the non-Markovian diffusion process is shown in Fig. 4.6.

The reverse process remains mainly the same and is summarized as follows.

According to Eq. 4.8, the model $\boldsymbol{\epsilon}_\Theta$ predicts the noise that was added to obtain $\mathbf{x}_t$ starting from $\mathbf{x}_0$. Using this predicted noise, an approximation of the denoised observation can be made in a manner similar to Eq. 4.7:

$$\mathbf{x}_\Theta(\mathbf{x}_t, t) \coloneqq \frac{1}{\sqrt{\bar{\alpha}_t}} \left(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}_\Theta(\mathbf{x}_t, t) \right)$$

The reverse process is then defined as:

$$p_\Theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \begin{cases} \mathcal{N}(\mathbf{x}_0; \mathbf{x}_\Theta(\mathbf{x}_1, 1), \sigma_1^2 \mathbf{I}) & \text{if } t = 1 \\ q_{\boldsymbol{\sigma}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_\Theta(\mathbf{x}_t, t)) & \text{otherwise} \end{cases}$$

This process starts from the prior $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$. For the case $t = 1$, Gaussian noise with covariance $\sigma_1^2 \mathbf{I}$ is added to ensure that the reverse process is supported everywhere.

Song et al. [40] demonstrated that this formulation, when trained using the variational lower bound in Eq. 4.6, converges to the same solution as DDPM models trained with Eq. 4.9, provided that the model parameters are not shared across different timesteps t. This is because, when the parameters are not shared, each term is optimized independently, leading to the same global optimum. This enables the use of pre-trained DDPM models and allows for the manipulation of the reverse process through adjustments to $\boldsymbol{\sigma}$ to improve the quality of the generated samples. There are two special cases to note: first, by setting $\sigma_t^2 = \tilde{\beta}_t \, \forall t \in \{1, 2, \ldots, T\}$, the reverse process is identical to that of DDPM models, as this results in a Markovian forward process; second, by taking $\boldsymbol{\sigma} \to \mathbf{0}$, the model becomes an implicit probabilistic model, known as Denoising Diffusion Implicit Model (DDIM), where the forward process is deterministic (except for $t = 1$), and the reverse process eliminates the addition of noise at each timestep t.

Furthermore, Song et al. [40] proposed an accelerated image generation procedure, drawing similar conclusions about the optimization process. They suggested rewriting the forward process as:

$$q_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\mathbf{x}_{1:T} \mid \mathbf{x}_0) \coloneqq q_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\mathbf{x}_{\tau_S} \mid \mathbf{x}_0) \prod_{i=2}^{S} q_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\mathbf{x}_{\tau_{i-1}} \mid \mathbf{x}_{\tau_i}, \mathbf{x}_0) \prod_{t \in \bar{\boldsymbol{\tau}}} q_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\mathbf{x}_t \mid \mathbf{x}_0)$$

where $\boldsymbol{\tau}$ is a subsequence of $[1, 2, \ldots, T]$ of length S, with the constraint $\tau_S = T$, and $\bar{\boldsymbol{\tau}} \coloneqq \{1, 2, \ldots, T\} \setminus \boldsymbol{\tau}$. The conditional distributions are defined as:

$$q_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\mathbf{x}_{\tau_{i-1}} \mid \mathbf{x}_{\tau_i}, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{\tau_{i-1}}; \sqrt{\bar{\alpha}_{\tau_{i-1}}} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{\tau_{i-1}} - \sigma_{\tau_i}^2} \cdot \frac{\mathbf{x}_{\tau_i} - \sqrt{\bar{\alpha}_{\tau_i}} \mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_{\tau_i}}}, \sigma_{\tau_i}^2 \mathbf{I} \right)$$

$$q_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I})$$

These coefficients are chosen to ensure that the marginals in Eq. 4.5 are correctly matched. The corresponding reverse process is defined as:

$$p_\Theta(\mathbf{x}_{0:T}) \coloneqq p(\mathbf{x}_T) \prod_{i=2}^{S} p_\Theta(\mathbf{x}_{\tau_{i-1}} \mid \mathbf{x}_{\tau_i}) \prod_{t \in \bar{\boldsymbol{\tau}}} p_\Theta(\mathbf{x}_0 \mid \mathbf{x}_t)$$

where

$$\begin{aligned} p_\Theta(\mathbf{x}_{\tau_{i-1}} \mid \mathbf{x}_{\tau_i}) &= q_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\mathbf{x}_{\tau_{i-1}} \mid \mathbf{x}_{\tau_i}, \mathbf{x}_\Theta(\mathbf{x}_{\tau_i}, \tau_i)) \\ p_\Theta(\mathbf{x}_0 \mid \mathbf{x}_t) &= \mathcal{N}(\mathbf{x}_0; \mathbf{x}_\Theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I}) \end{aligned}$$

In this setup, only a subset of the conditionals–specifically, $p_\Theta(\mathbf{x}_{\tau_{i-1}} \mid \mathbf{x}_{\tau_i})$ and $p_\Theta(\mathbf{x}_0 \mid \mathbf{x}_{\tau_1})$–are used to generate samples, following the reverse of the subsequence $\boldsymbol{\tau}$, referred to as the *sampling trajectory*, thereby speeding up the sampling process. A diagram of the accelerated diffusion process is illustrated in Fig. 4.7.

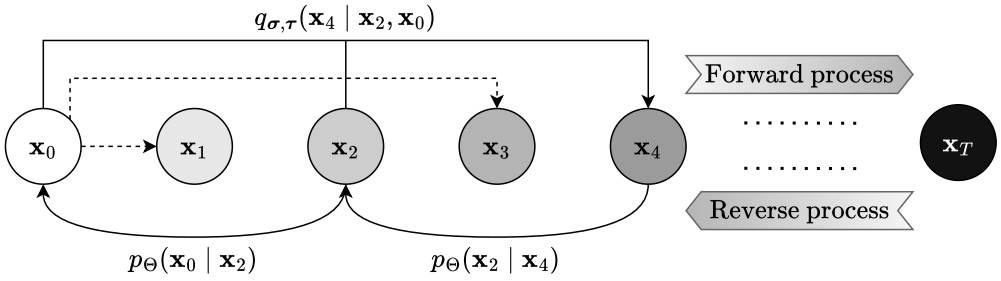


Figure 4.7: Scheme of the accelerated diffusion process, with the first two elements of $\boldsymbol{\tau}$ set to 2 and 4. Adapted from «Denoising diffusion implicit models» [40].

Finally, Song et al. [40] proposed computing the vector of standard deviations $\boldsymbol{\sigma}$ by indexing its elements according to $\boldsymbol{\tau}$, as follows:

$$\sigma_{\tau_i}(\eta) = \eta \sqrt{\frac{1 - \bar{\alpha}_{\tau_{i-1}}}{1 - \bar{\alpha}_{\tau_i}}} \sqrt{1 - \frac{\bar{\alpha}_{\tau_i}}{\bar{\alpha}_{\tau_{i-1}}}}$$

Here, $\eta \in \mathbb{R}^+$ is a hyperparameter: setting $\eta = 1$ recovers the original DDPM sampling process, while setting $\eta = 0$ results in the DDIM method.

4.3.3 Latent Diffusion Models

Although Song et al. [40] significantly enhanced the speed of sample generation by decreasing the number of timesteps needed in the reverse process, DMs still

face considerable limitations due to their high resource consumption during both training and sampling. These issues primarily stem from the latents, which reside in the pixel space.

In response to these limitations, Rombach et al. [42] introduced a novel architecture called Latent Diffusion Model (LDM), designed to lower computational requirements without sacrificing the performance typically associated with DMs. Their strategy is founded on the observation that likelihood-based models allocate significant computational resources on capturing high-frequency, often imperceptible details of the data during an initial learning phase known as *perceptual compression*. This stage is followed by a second learning phase, termed *semantic compression*, in which the generative model assimilates the semantic and conceptual composition of the data.

Their approach seeks to establish a perceptually equivalent space that is more computationally efficient for training high-resolution DMs. By leveraging inductive biases inherent in spatially structured data and utilizing a U-Net [46] backbone, they avoided extreme levels of compression that could degrade sampling quality. Specifically, they developed a perceptual compression model based on an autoencoder (details in Section 4.1.1).

For a given image $\mathbf{x} \in \mathbb{R}^{H\times W\times 3}$, the encoder maps it into a latent representation $\mathbf{z} = \mathcal{E}(\mathbf{x}) \in \mathbb{R}^{h\times w\times c}$, while the decoder reconstructs the image from the latent representation as $\hat{\mathbf{x}} = \mathcal{D}(\mathbf{z}) \in \mathbb{R}^{H\times W\times 3}$. The factor $f = \frac{H}{h} = 2^m$, where $m \in \mathbb{N}$, acts as a hyperparameter controlling the trade-off between compressing the latent space (which, as discussed later, accelerates both training and inference times for DMs) and preserving the information in the original image, ensuring perceptually similar outputs. The AEs are trained using an adversarial framework [54, 29] to better capture the image manifold by enhancing local, patch-based realism and alleviating the blurring effects typically encountered with simple reconstruction losses. The overall objective can be expressed as follows:

$$\begin{aligned}\mathcal{L} = \min_{\mathcal{E},\mathcal{D}} \max_{D} \big[&\lambda_1 \mathcal{L}_{\text{Rec}}(\mathbf{x}, \mathcal{D}(\mathcal{E}(\mathbf{x}))) + \lambda_2 \mathcal{L}_{\text{LPIPS}}(\mathbf{x}, \mathcal{D}(\mathcal{E}(\mathbf{x}))) \\ &+\lambda_3 \mathcal{L}_{\text{GAN}}(\mathbf{x}; \mathcal{E}, \mathcal{D}, D) + \lambda_4 \mathcal{L}_{\text{Reg}}(\mathbf{x}; \mathcal{E}, \mathcal{D})\big]\end{aligned}$$

In this equation, λ_i, $i \in \{1, 2, 3, 4\}$, are hyperparameters that dictate the relative weight of each loss component and D is a discriminator network (see Section 4.2). Here, $\mathcal{L}_{\text{Rec}}$ denotes the reconstruction loss, $\mathcal{L}_{\text{LPIPS}}$ represents the *Learned Perceptual Image Patch Similarity* loss introduced in [55], $\mathcal{L}_{\text{GAN}}$ is the patch-based adversarial loss [29], and $\mathcal{L}_{\text{Reg}}$ is a regularization loss designed to prevent arbitrary scaling of latent spaces. The regularization term $\mathcal{L}_{\text{Reg}}$ could take the form of a KL divergence term, similar to the one used in VAEs [32], or it could depend on the codebook size in a vector quantization layer, as in VQ-VAEs [33].

Once powerful AEs have been trained, both inference and training of DMs can occur within the learned latent spaces, thereby accelerating both processes. The

learning objective in Eq. 4.9 can now be reformulated as:

$$\mathcal{L}_{\text{LDM}}(\Theta) = \mathbb{E}_{t,\mathcal{E}(\mathbf{x}_0),\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon}_\Theta(\mathbf{z}_t, t) - \boldsymbol{\epsilon}\|^2\right]$$

The function $\boldsymbol{\epsilon}_\Theta(\mathbf{z}_t, t)$ can be realized using a time-conditional U-Net architecture. This enables the training to concentrate on the essential, semantic components of the data within a highly efficient, low-dimensional space.

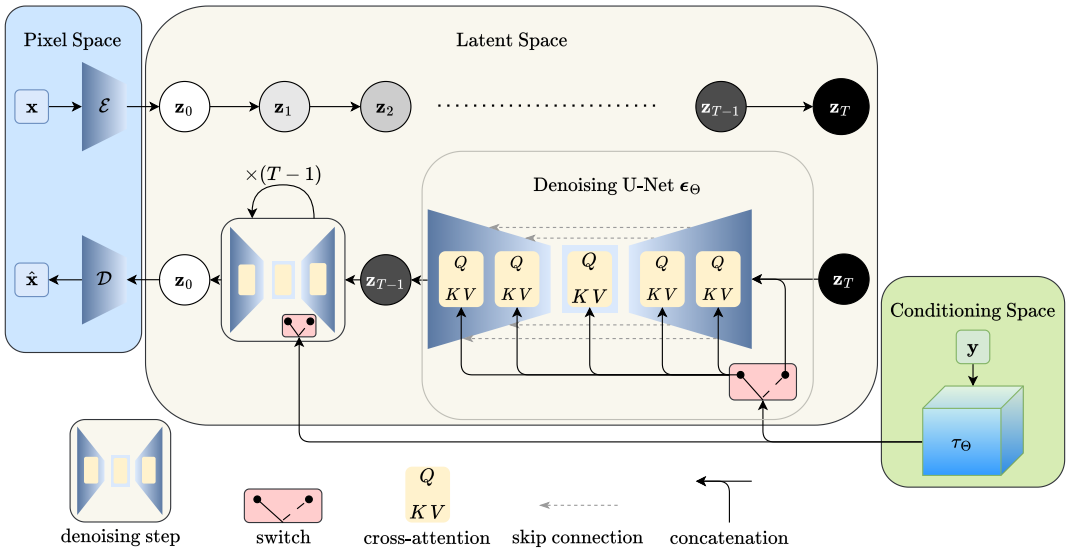


Figure 4.8: Diagram of the Latent Diffusion Model with conditioning through either concatenation or cross-attention. Adapted from «High-resolution image synthesis with latent diffusion models» [42].

They further extended DMs to model conditional distributions $p(\mathbf{z} \mid \mathbf{y})$ by incorporating a cross-attention mechanism [56] into the U-Net architecture. The condition $\mathbf{y}$ is first mapped to an intermediate representation $\tau_\Theta(\mathbf{y}) \in \mathbb{R}^{M \times d_\tau}$ using a modality-specific encoder τ_Θ. This representation is then integrated into the intermediate layers of the U-Net through cross-attention layers, as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) \cdot V$$

where:

$$Q = \varphi_i(\mathbf{z}_t) \cdot W_Q^{(i)}, \quad K = \tau_\Theta(\mathbf{y}) \cdot W_K^{(i)} \quad V = \tau_\Theta(\mathbf{y}) \cdot W_V^{(i)}$$

Here, $\varphi_i(\mathbf{z}_t) \in \mathbb{R}^{N \times d_\epsilon^{(i)}}$ is a flattened intermediate representation of the U-Net and $W_Q^{(i)} \in \mathbb{R}^{d_\epsilon^{(i)} \times d}$, $W_K^{(i)} \in \mathbb{R}^{d_\tau \times d}$ and $W_V^{(i)} \in \mathbb{R}^{d_\tau \times d}$ are learnable projection matrices.

The objective function for this approach is given by:

$$\mathcal{L}_{\text{LDM}}(\Theta) = \mathbb{E}_{t,\mathcal{E}(\mathbf{x}_0),\mathbf{y},\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon}_\Theta(\mathbf{z}_t, \tau_\Theta(\mathbf{y}), t) - \boldsymbol{\epsilon}\|^2\right] \tag{4.10}$$

In this formulation, both the model $\boldsymbol{\epsilon}_\Theta$ and the encoder τ_Θ are jointly optimized. A comprehensive diagram of the conditioned LDM model is shown in Fig. 4.8.

Chapter 5

Uncertainty Estimation

Deep Learning models have demonstrated remarkable performance across a wide range of tasks and are now broadly deployed in various real-world applications. However, these models often generate outputs based on high-dimensional data without providing any measure of confidence or uncertainty regarding their predictions. While their outputs are typically assumed to be correct, this lack of *uncertainty estimation* can lead to significant risks, as incorrect predictions may lead to catastrophic failures with serious consequences [57, 16]. As a result, there is an increasing need to integrate uncertainty estimation into DL systems to mitigate risks and improve reliability.

There are two major types of uncertainties that can be modeled: *aleatoric uncertainty*, which represents the inherent randomness or noise in the data, and *epistemic uncertainty*, which stems from uncertainty in the model parameters. While epistemic uncertainty can be reduced with more data and improved models, aleatoric uncertainty is irreducible, as it is an intrinsic part of the data generation process.

Uncertainty estimation can be addressed through a general framework known as *Bayesian Deep Learning*, where Bayesian Neural Networks (BNNs) are characterized by treating the weight parameters as distributions. This approach enables models to estimate both types of uncertainty [57]. However, it faces several challenges, such as difficulty in producing models that are competitive with non-Bayesian alternatives. BNNs typically require training from scratch, are challenging to optimize, and struggle to scale with large, high-dimensional datasets [58]. In 2024, Franchi et al. [59] proposed a method to convert pre-trained DNNs into BNNs by replacing their normalization layers with a novel layer called Bayesian Normalization Layer (BNL). While this method achieves its objective, it still necessitates modifications to the architecture and additional fine-tuning steps.

An alternative approach to addressing the problem is to enable models to generate uncertainty scores by using an additional output head [60], or by producing

probabilistic rather than deterministic outputs [61]. When multiple pre-trained models are available, Miao et al. [62] in 2024 proposed a novel probabilistic model ensemble framework that integrates prior knowledge from the pre-trained models. This ensemble enhances predictive performance in low-shot image classification while also quantifying uncertainty.

Furthermore, uncertainty estimates can play a key role in improving model performance and robustness. For instance, Upadhyay et al. [63, 64] showed how incorporating uncertainty information could improve model predictions or enable training in unsupervised settings, potentially leading to more accurate and/or resilient outputs in the face of challenging or ambiguous inputs.

In the following, we will concentrate on post-hoc approaches, where pre-trained DNNs remain unchanged, and uncertainty estimation techniques are applied on top of them [58, 65, 66].

5.1 BayesCap

In 2022, Upadhyay et al. [58] introduced a novel, data-efficient technique for post-hoc uncertainty estimation. This approach learns a Bayesian identity mapping over the outputs of a pre-trained deterministic Deep Neural Network, converting the deterministic outputs into probabilistic ones.

Consider a dataset $\mathcal{D} \coloneqq \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$, where each input $\mathbf{x}_i \in \mathcal{X} \subseteq \mathbb{R}^m$ and each output $\mathbf{y}_i \in \mathcal{Y} \subseteq \mathbb{R}^n$. Let $\mathbf{\Psi}_\Theta : \mathbb{R}^m \to \mathbb{R}^n$ represent a DNN parameterized by the learnable parameters Θ. After training the model on the dataset $\mathcal{D}$, the parameters Θ^* are learned, and the model is in a *frozen state*. The objective is to estimate the output distribution $p_{\mathcal{Y}|\mathcal{X}}$ using the point estimates $\bar{\mathbf{y}} = \mathbf{\Psi}_{\Theta^*}(\mathbf{x})$, where the distribution's parameters capture the irreducible aleatoric uncertainty.

To achieve the goal, a Bayesian identity mapping $\mathbf{\Omega}_\Phi : \mathbb{R}^n \to \mathbb{R}^{(k+1)\times n}$ is learned, which reconstructs the point estimates $\bar{\mathbf{y}}$ and generates the k parameters of the distribution $p_{\mathcal{Y}|\mathcal{X}}$. The learning process is conducted via maximum likelihood estimation. Specifically, the model $\mathbf{\Omega}_\Phi$ produces a set of outputs $\{\tilde{\mathbf{y}}_i, \tilde{\boldsymbol{\xi}}_i^1, \ldots, \tilde{\boldsymbol{\xi}}_i^k\} \coloneqq \mathbf{\Omega}_\Phi(\bar{\mathbf{y}}_i)$, which characterizes the distribution $p_{\mathcal{Y}|\mathcal{X}}(\mathbf{y}; \{\tilde{\mathbf{y}}_i, \tilde{\boldsymbol{\xi}}_i^1, \ldots, \tilde{\boldsymbol{\xi}}_i^k\})$, such that $\mathbf{y}_i \sim p_{\mathcal{Y}|\mathcal{X}}(\mathbf{y}; \{\tilde{\mathbf{y}}_i, \tilde{\boldsymbol{\xi}}_i^1, \ldots, \tilde{\boldsymbol{\xi}}_i^k\})$. The optimal parameters for the model are therefore determined by maximizing the likelihood $\mathscr{L}(\Phi; \mathcal{D}) \coloneqq \prod_{i=1}^N p_{\mathcal{Y}|\mathcal{X}}(\mathbf{y}_i; \{\tilde{\mathbf{y}}_i, \tilde{\boldsymbol{\xi}}_i^1, \ldots, \tilde{\boldsymbol{\xi}}_i^k\})$.

The choice of the functional form for $p_{\mathcal{Y}|\mathcal{X}}$ is typically guided by the need to express the uncertainty in a closed-form solution $\mathscr{F}$ that depends on the estimated

parameters, such as:

$$\Phi^* \coloneqq \underset{\Phi}{\operatorname{argmax}} \mathscr{L}(\Phi; \mathcal{D}) = \underset{\Phi}{\operatorname{argmax}} \prod_{i=1}^{N} p_{\mathcal{Y}|\mathcal{X}}(\mathbf{y}_i; \boldsymbol{\Omega}_\Phi(\bar{\mathbf{y}}_i))$$
$$= \underset{\Phi}{\operatorname{argmax}} \prod_{i=1}^{N} p_{\mathcal{Y}|\mathcal{X}}(\mathbf{y}_i; \{\tilde{\mathbf{y}}_i, \tilde{\boldsymbol{\xi}}_i^1, \ldots, \tilde{\boldsymbol{\xi}}_i^k\}) \tag{5.1}$$
$$\text{Uncertainty}(\tilde{\mathbf{y}}_i) = \mathscr{F}(\tilde{\boldsymbol{\xi}}_i^1, \ldots, \tilde{\boldsymbol{\xi}}_i^k) \tag{5.2}$$

The form of $p_{\mathcal{Y}|\mathcal{X}}$ is often chosen to be a *heteroscedastic*[1] Gaussian-like distribution, which models the per-feature residuals between predictions and ground-truth values. However, real-world inputs frequently contain outliers and artifacts, leading to residuals that often follow heavy-tailed distributions[2]. To account for these cases, Upadhyay et al. [58] proposed using a heteroscedastic generalized Gaussian distribution to model the residuals. This approach results in predictions for the *mean* $\tilde{\mathbf{y}}_i$, *scale* $\tilde{\boldsymbol{\alpha}}_i$, and *shape* $\tilde{\boldsymbol{\beta}}_i$ parameters. The likelihood and uncertainty expressions in Eq. 5.1 and Eq. 5.2 are then rewritten as follows:

$$\Phi^* \coloneqq \underset{\Phi}{\operatorname{argmax}} \mathscr{L}(\Phi; \mathcal{D}) = \underset{\Phi}{\operatorname{argmax}} \prod_{i=1}^{N} p_{\mathcal{Y}|\mathcal{X}}(\mathbf{y}_i; \{\tilde{\mathbf{y}}_i, \tilde{\boldsymbol{\alpha}}_i, \tilde{\boldsymbol{\beta}}_i\})$$
$$= \underset{\Phi}{\operatorname{argmax}} \prod_{i=1}^{N} \frac{\tilde{\boldsymbol{\beta}}_i}{2\tilde{\boldsymbol{\alpha}}_i \Gamma(\frac{1}{\tilde{\boldsymbol{\beta}}_i})} e^{-\left(\frac{|\tilde{\mathbf{y}}_i - \mathbf{y}_i|}{\tilde{\boldsymbol{\alpha}}_i}\right)^{\tilde{\boldsymbol{\beta}}_i}} \tag{5.3}$$
$$= \underset{\Phi}{\operatorname{argmin}} - \log \mathscr{L}(\Phi; \mathcal{D})$$
$$= \underset{\Phi}{\operatorname{argmin}} \sum_{i=1}^{N} \left(\frac{|\tilde{\mathbf{y}}_i - \mathbf{y}_i|}{\tilde{\boldsymbol{\alpha}}_i}\right)^{\tilde{\boldsymbol{\beta}}_i} - \log \frac{\tilde{\boldsymbol{\beta}}_i}{\tilde{\boldsymbol{\alpha}}_i} + \log \Gamma(\frac{1}{\tilde{\boldsymbol{\beta}}_i}) \tag{5.4}$$
$$\text{Uncertainty}(\tilde{\mathbf{y}}_i) = \tilde{\boldsymbol{\sigma}}_i^2 = \frac{\tilde{\boldsymbol{\alpha}}_i^2 \Gamma(\frac{3}{\tilde{\boldsymbol{\beta}}_i})}{\Gamma(\frac{1}{\tilde{\boldsymbol{\beta}}_i})} \tag{5.5}$$

where $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt$, $\forall x > 0$ denotes the Gamma function and $\boldsymbol{\sigma}_i^2$ represents the variance of the generalized Gaussian distribution. These uncertainties are used as per-feature predictions of the errors that will occur at inference time.

This formulation necessitates that the reconstructed outputs $\tilde{\mathbf{y}}_i$ closely align with the point estimates $\bar{\mathbf{y}}_i$. As the quality of the reconstructions improves, the

[1] A sequence of random variables is said to be heteroscedastic if the variance varies across the different variables. In our context, each feature is treated as a random variable, each with its own distinct variance.

[2] See Appendix B.1 for more details.

uncertainties predicted by the model should converge to those of the pre-trained model's predictions, which can be expressed as follows:

$$\tilde{\mathbf{y}}_i \to \bar{\mathbf{y}}_i \implies \tilde{\boldsymbol{\sigma}}_i^2 \to \bar{\boldsymbol{\sigma}}_i^2$$

Consequently, the training process aims to minimize both the reconstruction loss and the negative log-likelihood specified in Eq. 5.4 as follows:

$$\mathcal{L}(\Phi) = \lambda_1 \sum_{i=1}^{N} |\tilde{\mathbf{y}}_i - \bar{\mathbf{y}}_i| + \lambda_2 \sum_{i=1}^{N} \left(\frac{|\tilde{\mathbf{y}}_i - \mathbf{y}_i|}{\tilde{\boldsymbol{\alpha}}_i} \right)^{\tilde{\boldsymbol{\beta}}_i} - \log \frac{\tilde{\boldsymbol{\beta}}_i}{\tilde{\boldsymbol{\alpha}}_i} + \log \Gamma(\frac{1}{\tilde{\boldsymbol{\beta}}_i}) \tag{5.6}$$

In this objective, the hyperparameters λ_1 and λ_2 determine the relative importance of the reconstruction loss and the negative log-likelihood term, respectively, in the overall loss function.

The model $\boldsymbol{\Omega}_\Phi$ implements a Bayesian identity mapping, meaning its structure is independent of the architecture and specific task performed by $\boldsymbol{\Psi}_{\Theta^*}$. As demonstrated by Upadhyay et al. [58], their approach does not require task-specific tuning and is robust to variations in hyperparameters, such as learning rate and architecture. Moreover, because the mapping is applied on top of the pre-trained model, the output of the pre-trained model can be used directly, ensuring that the predictions of the original model remain unaffected and that there is no degradation in its predictive performance. A schematic representation of the complete model is provided in Fig. 5.1.

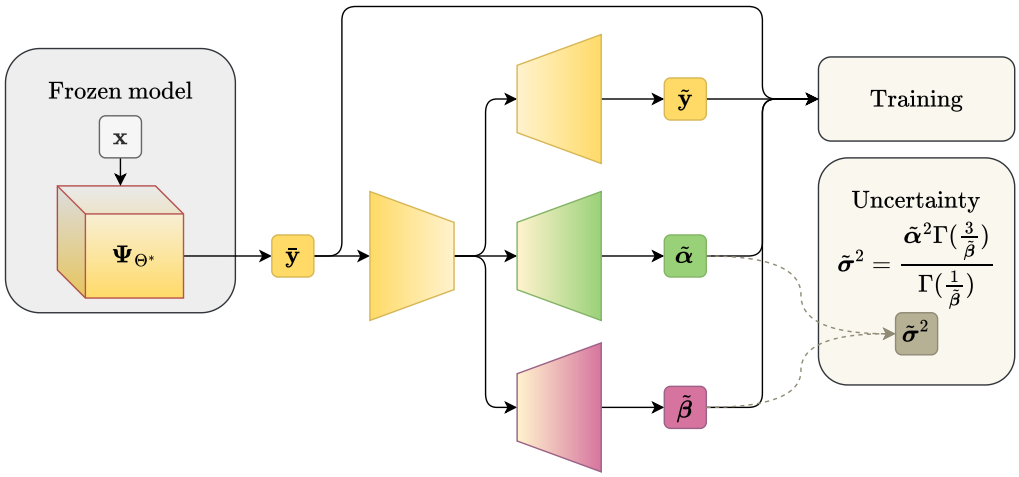


Figure 5.1: BayesCap architecture. Adapted from «BayesCap: Bayesian Identity Cap for Calibrated Uncertainty in Frozen Neural Networks» [58].

5.2 ProbVLM

In 2023, Upadhyay et al. [65] proposed a novel probabilistic adapter designed to estimate probability distributions for the embeddings of pre-trained *large-scale Vision-Language Models* [31, 67]. While these models align image and text modalities in a shared embedding space in a deterministic manner, they fail to capture the inherent ambiguity in the correspondences between the two: an image can have multiple captions, and a caption can correspond to multiple images.

Building on the concepts from BayesCap [58], Upadhyay et al. [65] extended the use of uncertainties from representing predictive errors at inference time to modeling the ambiguities present in the different modalities. They achieved this by employing a combination of intra-modal and cross-modal alignment objectives.

Formally, let $\mathcal{I}$ denote the image space and $\mathcal{C}$ the caption space. Consider a vision-language dataset $\mathcal{D} \coloneqq \{(I_1, C_1), (I_2, C_2), \ldots, (I_N, C_N)\} \subseteq \{(I, C) \mid I \in \mathcal{I}, C \in \mathcal{C}\}$. The dataset satisfies the condition that $\forall i, j \in \{1, 2, \ldots, N\}, i \neq j \Rightarrow I_i \neq I_j \vee C_i \neq C_j$. In other words, it is allowed that the same image I_i appears in multiple pairs, or the same caption C_i appears in multiple pairs, as long as they are associated with different counterparts. VLMs typically learn a shared embedding space $\mathcal{Z} \subseteq \mathbb{R}^n$ for both modalities, allowing the measurement of cross-modal similarity via distances between elements. These models learn the embedding space using modality-specific encoders, $\boldsymbol{\Psi}^{\mathcal{V}} \coloneqq \boldsymbol{\Psi}^{\mathcal{V}}_{\Theta^*_{\mathcal{V}}} : \mathcal{I} \to \mathcal{Z}$ for images and $\boldsymbol{\Psi}^{\mathcal{T}} \coloneqq \boldsymbol{\Psi}^{\mathcal{T}}_{\Theta^*_{\mathcal{T}}} : \mathcal{C} \to \mathcal{Z}$ for captions. Both encoders are assumed to be in a *frozen state*, with $\Theta^*_{\mathcal{V}}$ and $\Theta^*_{\mathcal{T}}$ representing the learned set of parameters of each encoder.

To capture the inherent ambiguity in the embeddings $\mathbf{z} \in \mathcal{Z}$, the conditional probability distribution $p_{\mathcal{Z}|\mathcal{X}} \coloneqq p_{\mathcal{Z}|\mathcal{X}}(\mathbf{z}; \{\tilde{\mathbf{z}}, \tilde{\boldsymbol{\xi}}^1, \ldots, \tilde{\boldsymbol{\xi}}^k\})$ is modeled in a post-hoc manner, on top of the pre-trained encoders, where $\mathcal{X} \coloneqq \mathcal{I} \cup \mathcal{C}$. Specifically, the model is defined as:

$$\boldsymbol{\Omega}_{\Phi} \coloneqq (\boldsymbol{\Omega}^{\mathcal{V}}, \boldsymbol{\Omega}^{\mathcal{T}})$$

where:

$$\boldsymbol{\Omega}^{\mathcal{V}} \coloneqq \boldsymbol{\Omega}^{\mathcal{V}}_{\Phi_{\mathcal{V}}} : \mathbb{R}^n \to \mathbb{R}^{(k+1)\times n}, \quad \boldsymbol{\Omega}^{\mathcal{T}} \coloneqq \boldsymbol{\Omega}^{\mathcal{T}}_{\Phi_{\mathcal{T}}} : \mathbb{R}^n \to \mathbb{R}^{(k+1)\times n}$$

Here, $\Phi \coloneqq \Phi_{\mathcal{V}} \cup \Phi_{\mathcal{T}}$ represents the full set of model parameters. The components $\boldsymbol{\Omega}^{\mathcal{V}}$ and $\boldsymbol{\Omega}^{\mathcal{T}}$ are Bayesian identity mapping models that are trained to reconstruct the point estimates $\bar{\mathbf{z}}_{\mathcal{V}} = \boldsymbol{\Psi}^{\mathcal{V}}(I)$ and $\bar{\mathbf{z}}_{\mathcal{T}} = \boldsymbol{\Psi}^{\mathcal{T}}(C)$, respectively. These models also generate the k parameters of the modality-specific distributions $p_{\mathcal{Z}|\mathcal{I}}$ and $p_{\mathcal{Z}|\mathcal{C}}$, respectively.

The learning process for $\boldsymbol{\Omega}_{\Phi}$ is structured to ensure both *intra-modal alignment* and *cross-modal alignment*. Intra-modal alignment ensures that the unimodal mapping models generate reconstructions that faithfully approximate the original unimodal embeddings, allowing the estimated uncertainties to serve as reliable proxies for the uncertainties of the pre-trained encoders. Cross-modal alignment,

on the other hand, ensures that the estimated distribution parameters effectively capture the inherent ambiguities and uncertainties both within individual modalities and across modalities.

The intra-modal alignment problem can be addressed by designing modality-specific components of $\mathbf{\Omega}_\Phi$ that reconstruct the original unimodal embeddings. This is achieved by assuming that the residuals are independent but *not* identically distributed, and by learning the heteroscedasticity of these residuals during the reconstruction process, allowing them to follow potentially heavy-tailed distributions. The modality-specific conditional distributions can be modeled in a similar manner to Eq. 5.3, using generalized Gaussian distributions $p_{\mathcal{Z}|\mathcal{X}}(\mathbf{z}; \{\tilde{\mathbf{z}}, \tilde{\boldsymbol{\alpha}}, \tilde{\boldsymbol{\beta}}\})$. These distributions are learned via maximum likelihood estimation based on the modality-specific embeddings $\{\bar{\mathbf{z}}_i\}_{i=1}^N$. As a result, the modality-specific components are trained by minimizing the same loss function as in Eq. 5.6, where the embeddings $\{\bar{\mathbf{z}}_i\}$ are used to learn the distribution parameters. The corresponding losses are denoted as $\mathcal{L}^{\mathcal{V}}_{\text{Intra}}(\Phi_{\mathcal{V}})$ and $\mathcal{L}^{\mathcal{T}}_{\text{Intra}}(\Phi_{\mathcal{T}})$ for the vision and text modalities, respectively.

The cross-modal alignment problem can be tackled by ensuring that the vision and text conditional distributions belonging to similar concepts are close to each other. Given the embedding pairs $\{(\bar{\mathbf{z}}_i^{\mathcal{V}}, \bar{\mathbf{z}}_i^{\mathcal{T}}) \coloneqq (\mathbf{\Psi}^{\mathcal{V}}(I_i), \mathbf{\Psi}^{\mathcal{T}}(C_i))\}_{i=1}^N$, the modality-specific conditional distributions $p_{\mathcal{Z}|\mathcal{I}}(\mathbf{z}; \{\tilde{\mathbf{z}}^{\mathcal{V}}, \tilde{\boldsymbol{\alpha}}^{\mathcal{V}}, \tilde{\boldsymbol{\beta}}^{\mathcal{V}}\})$ and $p_{\mathcal{Z}|\mathcal{C}}(\mathbf{z}; \{\tilde{\mathbf{z}}^{\mathcal{T}}, \tilde{\boldsymbol{\alpha}}^{\mathcal{T}}, \tilde{\boldsymbol{\beta}}^{\mathcal{T}}\})$ can be computed. The degree of alignment between the two distributions is given by the likelihood $p(\mathbf{z}^{\mathcal{V}} = \mathbf{z}^{\mathcal{T}})$, where $\mathbf{z}^{\mathcal{V}} \sim p_{\mathcal{Z}|\mathcal{I}}(\mathbf{z}; \{\tilde{\mathbf{z}}^{\mathcal{V}}, \tilde{\boldsymbol{\alpha}}^{\mathcal{V}}, \tilde{\boldsymbol{\beta}}^{\mathcal{V}}\})$ and $\mathbf{z}^{\mathcal{T}} \sim p_{\mathcal{Z}|\mathcal{C}}(\mathbf{z}; \{\tilde{\mathbf{z}}^{\mathcal{T}}, \tilde{\boldsymbol{\alpha}}^{\mathcal{T}}, \tilde{\boldsymbol{\beta}}^{\mathcal{T}}\})$. This alignment is expressed as:

$$p(\mathbf{z}^{\mathcal{V}} = \mathbf{z}^{\mathcal{T}}) \coloneqq \iint p_{\mathcal{Z}|\mathcal{I}}(\mathbf{z}^{\mathcal{V}}; \{\tilde{\mathbf{z}}^{\mathcal{V}}, \tilde{\boldsymbol{\alpha}}^{\mathcal{V}}, \tilde{\boldsymbol{\beta}}^{\mathcal{V}}\}) p_{\mathcal{Z}|\mathcal{C}}(\mathbf{z}^{\mathcal{T}}; \{\tilde{\mathbf{z}}^{\mathcal{T}}, \tilde{\boldsymbol{\alpha}}^{\mathcal{T}}, \tilde{\boldsymbol{\beta}}^{\mathcal{T}}\}) \delta(\mathbf{z}^{\mathcal{V}} - \mathbf{z}^{\mathcal{T}})\, d\mathbf{z}^{\mathcal{V}} d\mathbf{z}^{\mathcal{T}}$$

where $\delta(\cdot)$ denotes the *Dirac delta distribution*.

Considering that the conditional distributions are generalized Gaussian distributions, the random variable $\Delta\mathbf{z} \coloneqq \mathbf{z}^{\mathcal{V}} - \mathbf{z}^{\mathcal{T}}$ follows a distribution based on the *Bivariate Fox H-function*, which does not yield an appropriate training function for DNNs. To address this, Upadhyay et al. [65] proposed an approximation by interpreting the integral as a convolution between two distributions:

$$\begin{aligned} p(\mathbf{z}^{\mathcal{V}} = \mathbf{z}^{\mathcal{T}}) &\approx \frac{1}{2} \int p_{\mathcal{Z}|\mathcal{I}}(\mathbf{z}; \{\tilde{\mathbf{z}}^{\mathcal{V}}, \tilde{\boldsymbol{\alpha}}^{\mathcal{V}}, \tilde{\boldsymbol{\beta}}^{\mathcal{V}}\}) \delta(\mathbf{z} - \tilde{\mathbf{z}}^{\mathcal{T}})\, d\mathbf{z} \\ &\quad + \frac{1}{2} \int p_{\mathcal{Z}|\mathcal{C}}(\mathbf{z}; \{\tilde{\mathbf{z}}^{\mathcal{T}}, \tilde{\boldsymbol{\alpha}}^{\mathcal{T}}, \tilde{\boldsymbol{\beta}}^{\mathcal{T}}\}) \delta(\mathbf{z} - \tilde{\mathbf{z}}^{\mathcal{V}})\, d\mathbf{z} \\ &= \frac{1}{2}(p_{\mathcal{Z}|\mathcal{I}}(\tilde{\mathbf{z}}^{\mathcal{T}}; \{\tilde{\mathbf{z}}^{\mathcal{V}}, \tilde{\boldsymbol{\alpha}}^{\mathcal{V}}, \tilde{\boldsymbol{\beta}}^{\mathcal{V}}\}) + p_{\mathcal{Z}|\mathcal{C}}(\tilde{\mathbf{z}}^{\mathcal{V}}; \{\tilde{\mathbf{z}}^{\mathcal{T}}, \tilde{\boldsymbol{\alpha}}^{\mathcal{T}}, \tilde{\boldsymbol{\beta}}^{\mathcal{T}}\})) \end{aligned}$$

To avoid biased estimation caused by using the reconstructions as the means of the generalized Gaussians, during training, the modality-specific embeddings are used

instead. This leads to the following negative log-likelihood term:

$$\mathcal{L}_{\text{Cross}}(\Phi_{\mathcal{V}}, \Phi_{\mathcal{T}}) \coloneqq \lambda_1 |\tilde{\mathbf{z}}^{\mathcal{V}} - \bar{\mathbf{z}}^{\mathcal{T}}| + \lambda_2 \left[\left(\frac{|\tilde{\mathbf{z}}^{\mathcal{V}} - \bar{\mathbf{z}}^{\mathcal{T}}|}{\tilde{\boldsymbol{\alpha}}^{\mathcal{V}}} \right)^{\tilde{\beta}^{\mathcal{V}}} - \log \frac{\tilde{\boldsymbol{\beta}}^{\mathcal{V}}}{\tilde{\boldsymbol{\alpha}}^{\mathcal{V}}} + \log \Gamma(\frac{1}{\tilde{\boldsymbol{\beta}}^{\mathcal{V}}}) \right]$$
$$+ \lambda_1 |\tilde{\mathbf{z}}^{\mathcal{T}} - \bar{\mathbf{z}}^{\mathcal{V}}| + \lambda_2 \left[\left(\frac{|\tilde{\mathbf{z}}^{\mathcal{T}} - \bar{\mathbf{z}}^{\mathcal{V}}|}{\tilde{\boldsymbol{\alpha}}^{\mathcal{T}}} \right)^{\tilde{\beta}^{\mathcal{T}}} - \log \frac{\tilde{\boldsymbol{\beta}}^{\mathcal{T}}}{\tilde{\boldsymbol{\alpha}}^{\mathcal{T}}} + \log \Gamma(\frac{1}{\tilde{\boldsymbol{\beta}}^{\mathcal{T}}}) \right]$$

where the hyperparameters λ_1 and λ_2 function similarly to those in Eq. 5.6.

Finally, the overall loss function is expressed as:

$$\mathcal{L}(\Phi) = \mathcal{L}^{\mathcal{V}}_{\text{Intra}}(\Phi_{\mathcal{V}}) + \mathcal{L}^{\mathcal{T}}_{\text{Intra}}(\Phi_{\mathcal{T}}) + \lambda_{\text{Cross}} \mathcal{L}_{\text{Cross}}(\Phi_{\mathcal{V}}, \Phi_{\mathcal{T}})$$

where λ_{Cross} is a hyperparameter that controls the relative weight of the cross-modal term compared to the intra-modality terms. A schematic overview of the entire model is shown in Fig. 5.2.

The aleatoric uncertainty $\tilde{\boldsymbol{\sigma}}^2_{\text{aleatoric}}$ can be derived from the modality-specific components, as detailed in Eq. 5.5. Furthermore, if the DNNs corresponding to $\boldsymbol{\Omega}^{\mathcal{V}}$ and $\boldsymbol{\Omega}^{\mathcal{T}}$ include dropout layers, activating dropout during inference allows for the estimation of epistemic uncertainty $\tilde{\boldsymbol{\sigma}}^2_{\text{epistemic}}$. The epistemic uncertainty can be computed over M forward passes as follows:

$$\tilde{\boldsymbol{\sigma}}^2_{\text{epistemic}} = \frac{1}{M} \sum_{m=1}^{M} \left(\tilde{\mathbf{z}}_m - \frac{1}{M} \sum_{j=1}^{M} \tilde{\mathbf{z}}_j \right)^2 \tag{5.7}$$

The total uncertainty is then the sum of the aleatoric and epistemic uncertainties:

$$\tilde{\boldsymbol{\sigma}}^2_{\text{total}} = \tilde{\boldsymbol{\sigma}}^2_{\text{aleatoric}} + \tilde{\boldsymbol{\sigma}}^2_{\text{epistemic}} \tag{5.8}$$

Finally, Upadhyay et al. [65] also demonstrated that the text conditional distribution $p_{\mathcal{Z}|\mathcal{C}}$, conditioned on a text embedding $\bar{\mathbf{z}}_{\mathcal{T}}$, can be sampled to generate new embeddings. These embeddings can then be synthesized using a Latent Diffusion Model to visualize the predicted distribution.

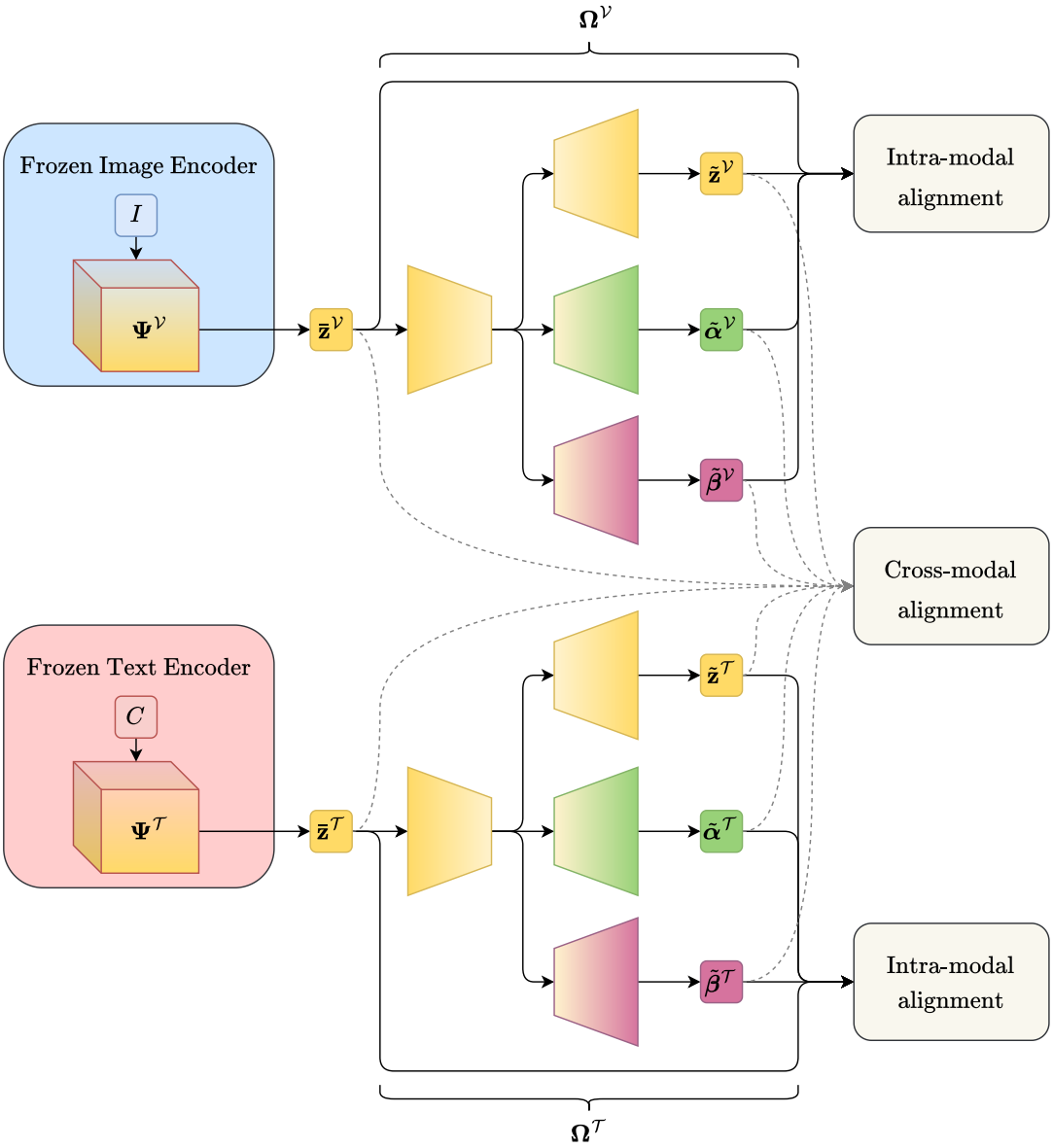


Figure 5.2: ProbVLM architecture. Adapted from «ProbVLM: Probabilistic Adapter for Frozen Vision-Language Models» [65].

Chapter 6

Experiments

6.1 Embedding Information Inspection

6.1.1 Inspection Framework

As outlined in Section 1.2, the primary aim of this thesis is to explore the information encoded in the embeddings produced by Visual Place Recognition (VPR) models. To achieve this, we propose leveraging Latent Diffusion Models (see Section 4.3.3), where the conditioning space shown in Fig. 4.8 will be aligned with the output space of a specific VPR model. For clarity, the same illustration is also presented in Fig. 6.1, with the relevant modifications highlighted to support our goal.

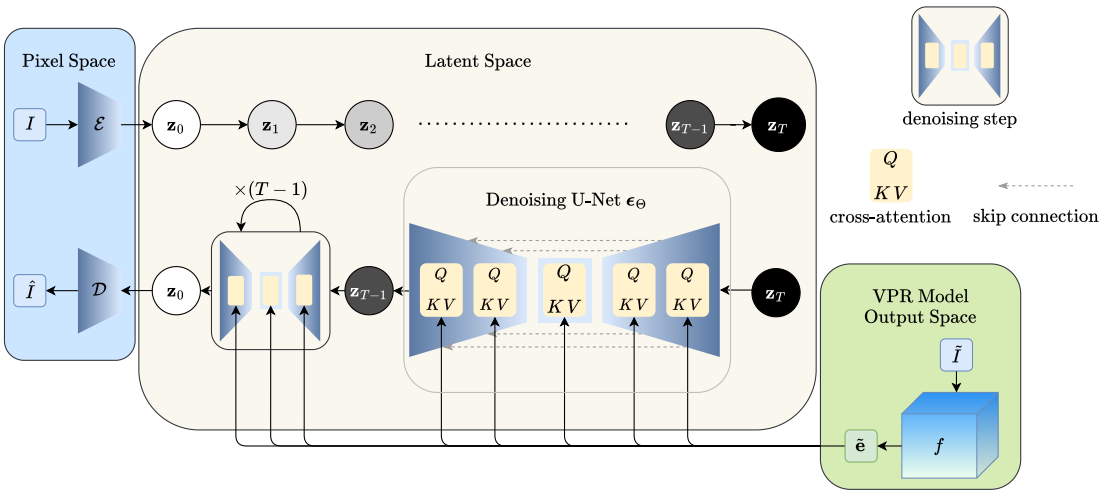


Figure 6.1: Illustration of a LDM model conditioned on the output space of a VPR model.

An image $\tilde{I}$ is given as input to the VPR model f, producing as output an

embedding vector $\tilde{\mathbf{e}} = f\left(\tilde{I}\right)$. The cross-attention mechanism enables this embedding vector $\tilde{\mathbf{e}}$ to guide the reverse process in the Diffusion Model, leading to the generation of an image that reflects the information encoded in the output vector. Formally, given M noise samples $\{\mathbf{z}_T^i\}_{i=1}^M$ drawn from the prior $p(\mathbf{z}_T) = \mathcal{N}(\mathbf{z}_T; \mathbf{0}, \mathbf{I})$, *i.e.*, $\mathbf{z}_T^i \sim p(\mathbf{z}_T)$ for all $i = 1, 2, \ldots, M$, the reverse process should generate $\{\hat{I}^i\}_{i=1}^M$ images that preserve the same information content as the VPR model has learned to predict the location of image $\tilde{I}$. We hypothesize that the VPR model has captured distinctive features, such as building facades, roads, intersections, relative spacing of elements, and viewpoint, while disregarding transient features like vehicles and occlusions, in order to identify the location depicted in an image. If this hypothesis holds, these same distinctive elements should appear in the generated images.

6.1.2 Implementation Details

To implement the framework outlined in Section 6.1.1, we build upon the official repository[1] of Rombach et al. [42]. We trained several Latent Diffusion Models by conditioning on the outputs from the VPR models listed in Table 6.1.

Method	Backbone	Embedding Dimension d
AP-GeM [14]	ResNet-101 [68]	2048
CliqueMining [13]	DINOv2 [22] (ViT-B/14 [69])	8448
Conv-AP [25]	ResNet-50 [68]	4096
CosPlace [52]	ResNet-50 [68]	32
CosPlace [52]	ResNet-50 [68]	64
CosPlace [52]	ResNet-50 [68]	128
CosPlace [52]	ResNet-50 [68]	512
CosPlace [52]	ResNet-50 [68]	2048
CricaVPR [17]	DINOv2 [22] (ViT-B/14 [69])	10752
EigenPlaces [19]	ResNet-50 [68]	128
EigenPlaces [19]	ResNet-50 [68]	512
EigenPlaces [19]	ResNet-50 [68]	2048
MixVPR [18]	ResNet-50 [68]	4096
NetVLAD [15]	VGG-16 [70]	4096
SALAD [51]	DINOv2 [22] (ViT-B/14 [69])	8448
SFRS [71]	VGG-16 [70]	4096

Table 6.1: Visual Place Recognition models used for conditioning.

[1] https://github.com/CompVis/latent-diffusion/tree/main

The training process is carried out using the SF-XL [52] dataset. We split the dataset into training and validation sets based on the latitude of the images: images with latitude 37.70 are assigned to the validation set, while all other images are used for training the LDM models. The number of images in each set is detailed in Table 6.2. We chose the SF-XL [52] dataset due to its large size, which enables

Set	Latitude	# Images
Training	$37.71 - 37.81$	5,350,506
Validation	37.70	256,914

Table 6.2: Training and validation sets derived from the training split of the SF-XL [52] dataset used for training LDM models.

the Diffusion Models to effectively learn the relationship between the VPR model's output space and the original images from which the embeddings are derived. Additionally, by separating the data based on latitude, we can assess whether the Diffusion Models can generalize to geographic regions that were not included in the training data.

In detail, the training follows the procedure outlined in Section 4.3.3. A batch of images, denoted as $\mathbf{B}$, is used to compute the loss function in Eq. 4.10, while conditioning on the embeddings $\mathbf{E} = f(\mathbf{B})$. We employed the AdamW optimizer [72] and used the same LDM model architecture for all VPR models. The architectural and training hyperparameters are listed in Table 6.3, where d corresponds to the embedding dimension of the respective VPR model. The training process behaves similarly across all VPR models, as shown in the plots in Appendix C.1. A closer inspection of the curves reveals that models with larger embedding dimensions d tend to have lower loss values at the end of training. This suggests that a larger embedding size allows for more information to be encoded from the original image, making it easier for the DM to generate images that are more faithful to the input images.

6.1.3 Metrics

To evaluate the performance of Gen-AI models in computer vision, various metrics have been introduced in recent years [38, 36, 35, 37, 73, 74]. In this work, we follow the approach of Rombach et al. [42] and assess our models using three statistical measures: FID [36], Precision, and Recall [37].

The *Fréchet Inception Distance* (FID) is a metric introduced by Heusel et al. [36] in 2017 to address a limitation of the Inception Score [38], which does not directly compare the statistics of real and generated data. FID uses the Inception model [75, 76] to extract vision-relevant features from images. Assuming that the

Hyperparameter	Value
f	8
z-shape	$32 \times 32 \times 4$
T	1000
Loss Type	L_1
Noise Scheduler	Linear
Channels	192
Channel Multiplier	$[1, 2, 2, 4, 4]$
Attention Resolutions	$[32, 16, 8, 4]$
# Residual Blocks	2
# Heads	8
Batch Size	96
Iterations	$500k$
Learning Rate	5×10^{-5}
Image Shape	$256 \times 256 \times 3$
Context Dimension d_τ	d

Table 6.3: Architectural and training hyperparameters for the LDM model (see Section 4.3.1, Section 4.3.3, and «High-resolution image synthesis with latent diffusion models» [42] for further details on notation and the meaning of the architectural hyperparameters).

feature vectors from the real data distribution $p_{\text{data}}(\mathbf{x})$ and the generated data distribution $p_{\text{model}}(\mathbf{x})$ follow multidimensional Gaussians distributions, the FID score is computed as follows:

$$d^2((\mathbf{m}_{\text{data}}, \mathbf{C}_{\text{data}}), (\mathbf{m}_{\text{model}}, \mathbf{C}_{\text{model}})) = \|\mathbf{m}_{\text{data}} - \mathbf{m}_{\text{model}}\|_2^2 + \text{Tr}(\mathbf{C}_{\text{data}} + \mathbf{C}_{\text{model}} - 2(\mathbf{C}_{\text{data}}\mathbf{C}_{\text{model}})^{\frac{1}{2}})$$

Here, $(\mathbf{m}_{\text{data}}, \mathbf{C}_{\text{data}})$ and $(\mathbf{m}_{\text{model}}, \mathbf{C}_{\text{model}})$ are the mean vector and covariance matrix of the real and generated data distributions, respectively. A lower FID score indicates that the generated data distribution is closer to the real data distribution, meaning the model has learned to generate more realistic data. Thus, a smaller FID value reflects better model performance.

In 2019, Kynkäänniemi et al. [37] demonstrated that the FID score conceals an undesirable tradeoff between the *quality* and *diversity* of generated samples. To address this issue, they proposed distinguishing between the two aspects by explicitly modeling the manifolds of the data distribution, $p_{\text{data}}(\mathbf{x})$, and the model distribution, $p_{\text{model}}(\mathbf{x})$. This is achieved by using an intermediate model, g, to extract feature vectors. Specifically, two sets are formed by sampling N samples

from both $p_{\text{data}}(\mathbf{x})$ and $p_{\text{model}}(\mathbf{x})$, and applying g to map these samples into a d-dimensional space. That is, the sets are defined as:

$$\mathcal{S}_{\text{data}} = \{g(\mathbf{x}) \mid \mathbf{x} \sim p_{\text{data}}(\mathbf{x})\}$$
$$\mathcal{S}_{\text{model}} = \{g(\hat{\mathbf{x}}) \mid \hat{\mathbf{x}} \sim p_{\text{model}}(\mathbf{x})\}$$

where $|\mathcal{S}_{\text{data}}| = |\mathcal{S}_{\text{model}}| = N$. To approximate the manifold for each distribution, they place each sample within a hypersphere defined by the k-nearest neighbor of that sample. For this, they introduce two functions, $\kappa_{\text{data}}(\cdot)$ and $\kappa_{\text{model}}(\cdot)$, which return the k-th nearest neighbor of a given sample in $\mathcal{S}_{\text{data}}$ and $\mathcal{S}_{\text{model}}$, respectively. These functions lead to the following definitions:

$$\pi_{\text{data}}(g(\hat{\mathbf{x}})) = \begin{cases} 1 & \text{if } \exists\, g(\mathbf{x}) \in \mathcal{S}_{\text{data}} : \|g(\mathbf{x}) - g(\hat{\mathbf{x}})\|_2 \leq \|g(\mathbf{x}) - \kappa_{\text{data}}(g(\mathbf{x}))\|_2 \\ 0 & \text{otherwise} \end{cases}$$
$$\pi_{\text{model}}(g(\mathbf{x})) = \begin{cases} 1 & \text{if } \exists\, g(\hat{\mathbf{x}}) \in \mathcal{S}_{\text{model}} : \|g(\hat{\mathbf{x}}) - g(\mathbf{x})\|_2 \leq \|g(\hat{\mathbf{x}}) - \kappa_{\text{model}}(g(\hat{\mathbf{x}}))\|_2 \\ 0 & \text{otherwise} \end{cases}$$

These definitions allow Precision and Recall to be calculated as follows:

$$\text{precision}(\mathcal{S}_{\text{data}}, \mathcal{S}_{\text{model}}) = \frac{1}{N} \sum_{g(\hat{\mathbf{x}}) \in \mathcal{S}_{\text{model}}} \pi_{\text{data}}(g(\hat{\mathbf{x}}))$$
$$\text{recall}(\mathcal{S}_{\text{data}}, \mathcal{S}_{\text{model}}) = \frac{1}{N} \sum_{g(\mathbf{x}) \in \mathcal{S}_{\text{data}}} \pi_{\text{model}}(g(\mathbf{x}))$$

From these equations, we observe that Precision quantifies the proportion of generated data that lies within the real data manifold, whereas Recall measures the proportion of real data that falls within the generated data manifold. Specifically, if generated data are close to the real data distribution, this suggests high-quality samples, and Precision serves as a measure of that quality. Conversely, if real data points are captured within the generated data distribution, it indicates the model can capture the full range of real data variations, meaning that Recall reflects the diversity of the generated samples. In both cases, higher values correspond to better model performance.

While Heusel et al. [36] defined g as an Inception model, Kynkäänniemi et al. [37] suggested using a VGG-16 [70], based on the findings of Brock [77]. They demonstrated that the feature space generated by the second fully connected layer of VGG-16 produces meaningful nearest neighbors, representing semantically similar images.

In our experiments, we utilize both the Inception model and VGG-16 to assess whether the LDM models are trained correctly, aiming to obtain results comparable to those reported by Rombach et al. [42] in their study. Moreover, since our primary objective is to examine the information captured in the embeddings generated by

the VPR models, we also propose evaluating the same metrics by setting $g \coloneqq f$, *i.e.*, using the VPR model that conditions the generation process. These metrics will help us determine whether the real and generated data distributions align from the perspective of the VPR model. Additionally, as we expect the information in the generated images to closely match that of the real images used for conditioning, we propose computing the L_1 and L_2 metrics as follows:

$$L_1 \coloneqq \frac{1}{N} \sum_{(f(\mathbf{x}), f(\hat{\mathbf{x}}))} \|f(\mathbf{x}) - f(\hat{\mathbf{x}})\|_1$$
$$L_2 \coloneqq \frac{1}{N} \sum_{(f(\mathbf{x}), f(\hat{\mathbf{x}}))} \|f(\mathbf{x}) - f(\hat{\mathbf{x}})\|_2$$

where in each pair $(f(\mathbf{x}), f(\hat{\mathbf{x}}))$, the generated image $\hat{\mathbf{x}}$ is produced by conditioning on the embedding $f(\mathbf{x})$.

6.1.4 Quantitative Results

To quantitatively evaluate how both visual and fidelity metrics perform for our trained Latent Diffusion Models, we use the model conditioned on the output space of CosPlace [52] with embedding dimension $d = 2048$ as a baseline. This serves as the reference for setting up the evaluation strategy for all other models.

In 2022, Ho et al. [39] introduced the Classifier-Free Guidance (CFG) technique for Diffusion Models, which steers the generation process towards conditioning information without requiring a classifier. This idea was later adopted by Rombach et al. [42] for LDM models. Specifically, during sampling, the noise score estimate is influenced not only by the conditioned score $\boldsymbol{\epsilon}_\Theta(\mathbf{z}_t, \tau_\Theta(\mathbf{y}), t)$ but also by an unconditional score $\boldsymbol{\epsilon}_\Theta(\mathbf{z}_t, t)$. This relationship is expressed as follows:

$$\tilde{\boldsymbol{\epsilon}}_\Theta(\mathbf{z}_t, \tau_\Theta(\mathbf{y}), t) = \boldsymbol{\epsilon}_\Theta(\mathbf{z}_t, t) + s \cdot (\boldsymbol{\epsilon}_\Theta(\mathbf{z}_t, \tau_\Theta(\mathbf{y}), t) - \boldsymbol{\epsilon}_\Theta(\mathbf{z}_t, t))$$

where $s \in \mathbb{R}^+$ is a positive scalar, known as the *scale*. By adjusting s, we can balance the tradeoff between generating images that are highly faithful to the conditioning information and those that are more creative and diverse while still respecting the condition. In our case, the condition is defined as $\tau_\Theta(\mathbf{y}) \coloneqq \tilde{\mathbf{e}}$, *i.e.*, the embedding vector produced by a Visual Place Recognition model. Thus, the CFG formula becomes:

$$\tilde{\boldsymbol{\epsilon}}_\Theta(\mathbf{z}_t, \tilde{\mathbf{e}}, t) = \boldsymbol{\epsilon}_\Theta(\mathbf{z}_t, \mathbf{0}, t) + s \cdot (\boldsymbol{\epsilon}_\Theta(\mathbf{z}_t, \tilde{\mathbf{e}}, t) - \boldsymbol{\epsilon}_\Theta(\mathbf{z}_t, \mathbf{0}, t))$$

where $\mathbf{0}$ denotes the *null vector*, representing the absence of conditioning information. We analyze the effect of this scaling parameter in detail to optimize the fidelity of the generated images with respect to the conditioning information. To do this,

we compute all relevant metrics by selecting $s \in \{1, 1.5, 2, 3, 5, 10\}$ and varying the number of generated images $N \in \{1k, 2k, 3.5k, 7k, 14k, 25k, 50k, 75k\}$. This allows us to assess whether the resulting trends align with those observed by Kynkäänniemi et al. [37]. Additionally, we fix the number of steps S in the accelerated image generation process outlined in Section 4.3.2 to $S = 250$. Further details on this choice can be found in Section 6.1.6.

In Table 6.4, we provide the full set of metric values, while the corresponding curves are presented in Fig. 6.2–6.4. To improve readability and better fit the page, all plots use a logarithmic scale on the x-axis. Additionally, the FID, and occasionally the L_1 and L_2 metrics, have been normalized by dividing by their respective maximum values, ensuring that all metrics fall within the range $[0, 1]$. The legend displays the actual maximum and minimum values for reference.

As observed, the visual metrics largely follow the trends reported by Kynkäänniemi et al. [37] as we vary the number of generated images. Specifically, the FID continues to improve with a larger number of images (lower values are better), while Precision and Recall deteriorate (higher values are better), with all metrics stabilizing at higher values. The same pattern holds for Precision and Recall in fidelity metrics. The L_1 and L_2 metrics, however, stabilize at $14k$ images, meaning that increasing the number of images beyond this point no longer significantly impacts their values.

Regarding the scaling parameter s used for CFG [39], increasing its value significantly impacts the visual metrics, leading to worse performance. In contrast, modestly higher values enhance the fidelity metrics, with the best performance across all image counts observed at $s = 2$. However, further increasing the value causes performance to degrade, ultimately resulting in worse performance than the $s = 1$ case (*i.e.*, without Classifier-Free Guidance [39]) at $s = 10$.

Based on these trends and following the common practice of reporting visual metrics at $50k$ images, we set the number of images to $N = 50k$ for evaluating both visual and fidelity metrics in the remaining models in Appendix C.2. Additionally, for the rest of the thesis, we fix the scale parameter at $s = 2$, as it provides the best performance for our primary goal of explaining the output embeddings of Visual Place Recognition models with respect to fidelity metrics.

Table 6.4: Visual and fidelity metrics for the LDM conditioned on CosPlace's output space [52] with $d = 2048$, while varying the scale parameter s for CFG [39] and the number of images N. Metrics labeled with ↑ indicate better performance when higher, and those labeled with ↓ are better when lower. For brevity, 'P' stands for Precision and 'R' for Recall. The number of steps in the accelerated generation process is fixed at $S = 250$. The row representing the common evaluation practice is highlighted, with the chosen scale s indicated in a distinct color.

		VGG-16			InceptionV3			CosPlace ($d = 2048$)			
N	s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
$1k$	1	212.18	0.81	0.75	30.49	0.70	0.80	1.00	1.00	31.18	0.86
	1.5	259.43	0.75	0.77	31.98	0.70	0.81	1.00	0.99	28.93	0.80
	2	357.71	0.66	0.82	35.85	0.68	0.79	1.00	0.99	28.64	0.79
	3	587.65	0.46	0.81	44.71	0.57	0.77	1.00	0.99	29.08	0.80
	5	974.46	0.24	0.77	58.96	0.38	0.71	0.99	0.99	30.10	0.83
	10	1485.11	0.12	0.66	79.67	0.21	0.58	0.99	0.99	31.46	0.87
$2k$	1	140.58	0.75	0.71	18.22	0.70	0.77	0.99	1.00	30.97	0.86
	1.5	187.49	0.70	0.75	20.16	0.69	0.78	0.99	1.00	28.80	0.80
	2	289.79	0.59	0.77	24.84	0.63	0.76	1.00	1.00	28.58	0.79
	3	518.67	0.39	0.77	33.46	0.49	0.73	1.00	1.00	29.15	0.81
	5	885.57	0.20	0.70	47.00	0.34	0.65	0.99	1.00	30.30	0.84
	10	1406.66	0.09	0.61	66.85	0.18	0.55	0.99	0.97	31.59	0.87
$3.5k$	1	107.23	0.73	0.69	12.50	0.67	0.72	0.99	1.00	30.95	0.86
	1.5	154.99	0.67	0.77	14.71	0.66	0.74	0.99	1.00	28.73	0.80
	2	251.99	0.56	0.77	18.75	0.60	0.74	0.99	1.00	28.50	0.79
	3	467.87	0.37	0.76	27.19	0.47	0.67	0.99	1.00	29.08	0.80
	5	836.84	0.18	0.68	41.24	0.30	0.57	0.98	0.98	30.27	0.84
	10	1349.77	0.09	0.59	60.03	0.16	0.48	0.98	0.96	31.61	0.87
$7k$	1	78.34	0.70	0.68	8.29	0.64	0.70	0.96	1.00	31.02	0.86
	1.5	125.53	0.63	0.73	10.36	0.62	0.71	0.97	1.00	28.79	0.80
	2	226.07	0.50	0.73	14.51	0.55	0.69	0.97	1.00	28.52	0.79
	3	443.32	0.31	0.70	23.02	0.42	0.63	0.97	1.00	29.10	0.81
	5	809.48	0.14	0.64	36.52	0.27	0.53	0.96	0.96	30.27	0.84
	10	1334.58	0.05	0.52	55.58	0.13	0.42	0.93	0.93	31.64	0.88
$14k$	1	69.19	0.67	0.65	6.33	0.59	0.68	0.89	1.00	30.97	0.86
	1.5	113.63	0.59	0.70	8.30	0.57	0.69	0.92	1.00	28.78	0.80
	2	212.15	0.47	0.71	12.31	0.50	0.66	0.93	0.99	28.54	0.79
	3	430.48	0.28	0.67	20.78	0.37	0.60	0.91	0.97	29.09	0.81
	5	807.14	0.12	0.60	34.45	0.22	0.49	0.89	0.94	30.27	0.84
	10	1330.50	0.04	0.49	53.18	0.11	0.38	0.85	0.87	31.67	0.88
$25k$	1	63.68	0.62	0.64	5.40	0.56	0.66	0.78	0.99	30.96	0.86
	1.5	106.34	0.54	0.69	7.43	0.53	0.67	0.84	0.99	28.77	0.80
	2	203.27	0.42	0.69	11.45	0.46	0.64	0.85	0.98	28.53	0.79
	3	422.05	0.23	0.65	19.90	0.33	0.57	0.83	0.95	29.09	0.81

Continued on next page

Continued from previous page

		VGG-16			InceptionV3			CosPlace ($d = 2048$)			
N	s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
	5	799.25	0.10	0.56	33.28	0.19	0.47	0.80	0.89	30.26	0.84
	10	1326.81	0.03	0.46	52.00	0.09	0.36	0.75	0.81	31.66	0.88
$50k$	1	59.58	0.56	0.63	4.88	0.49	0.64	0.57	0.99	30.98	0.86
	1.5	103.11	0.48	0.67	6.94	0.47	0.64	0.67	0.98	28.78	0.80
	2	201.32	0.36	0.66	11.01	0.39	0.60	0.68	0.96	28.54	0.79
	3	420.28	0.20	0.62	19.41	0.27	0.52	0.65	0.90	29.10	0.81
	5	793.25	0.07	0.54	32.87	0.15	0.41	0.60	0.80	30.27	0.84
	10	1327.46	0.02	0.41	51.53	0.06	0.32	0.54	0.69	31.67	0.88
$75k$	1	57.64	0.53	0.63	4.64	0.45	0.63	0.43	0.99	30.96	0.86

6.1.5 Fidelity to Conditioning Information

To better assess how accurately the generated images reflect the conditioning information, we compare the L_1 and L_2 metrics in Table 6.4 with the same metrics calculated for the CosPlace [52] classes in the validation set of Table 6.2. For this comparison, we use the official repository[2] from Berton et al. [52], which provides code to generate the classes. In this context, classes are defined with strict constraints: two images belong to the same class if they were taken within 10 meters of each other and point in the same direction, with a heading difference (orientation) of no more than 30 degrees. For each class, we compute the L_1 and L_2 metrics by considering all possible pairwise distances between the image embeddings.

After filtering out classes with fewer than 10 images, we are left with 1842 classes in total. The metrics for each class are computed and summarized in Table 6.5, alongside the L_1 and L_2 metrics for our Latent Diffusion Model model.

As shown in the table, both scale values $s = 1$ and $s = 2$ achieve L_1 and L_2 metrics that are below the mean value of all classes. Notably, the case with $s = 2$ results in a substantial improvement, pushing the metrics toward the minimum value of the classes and even falling outside the standard deviation range.

This further highlights that our Latent Diffusion Model model has effectively learned to generate images that preserve the same informational content as the conditioning embeddings, emphasizing the model's ability to faithfully replicate and maintain crucial details from the conditioning input.

[2] https://github.com/gmberton/CosPlace

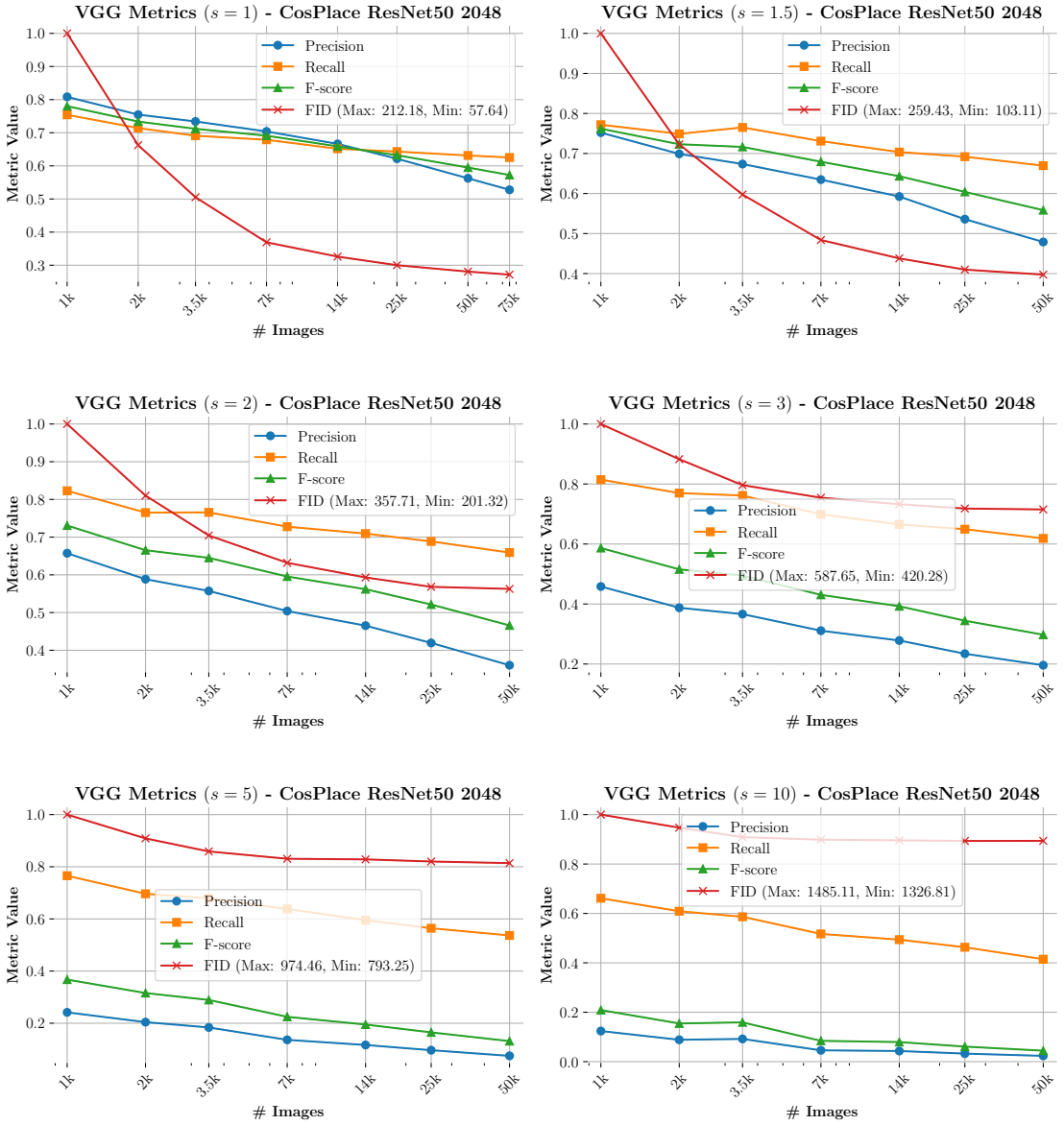


Figure 6.2: Plots of visual metrics for LDM conditioned on CosPlace's output space [52] with $d = 2048$, using VGG-16, while varying the scale parameter s for CFG [39]. The number of steps in the accelerated generation process is fixed at $S = 250$.

6.1.6 Qualitative Results

In Section 6.1.4, the Latent Diffusion Model conditioned on CosPlace [52] with an embedding dimension of $d = 2048$ was used as a baseline to guide the evaluation

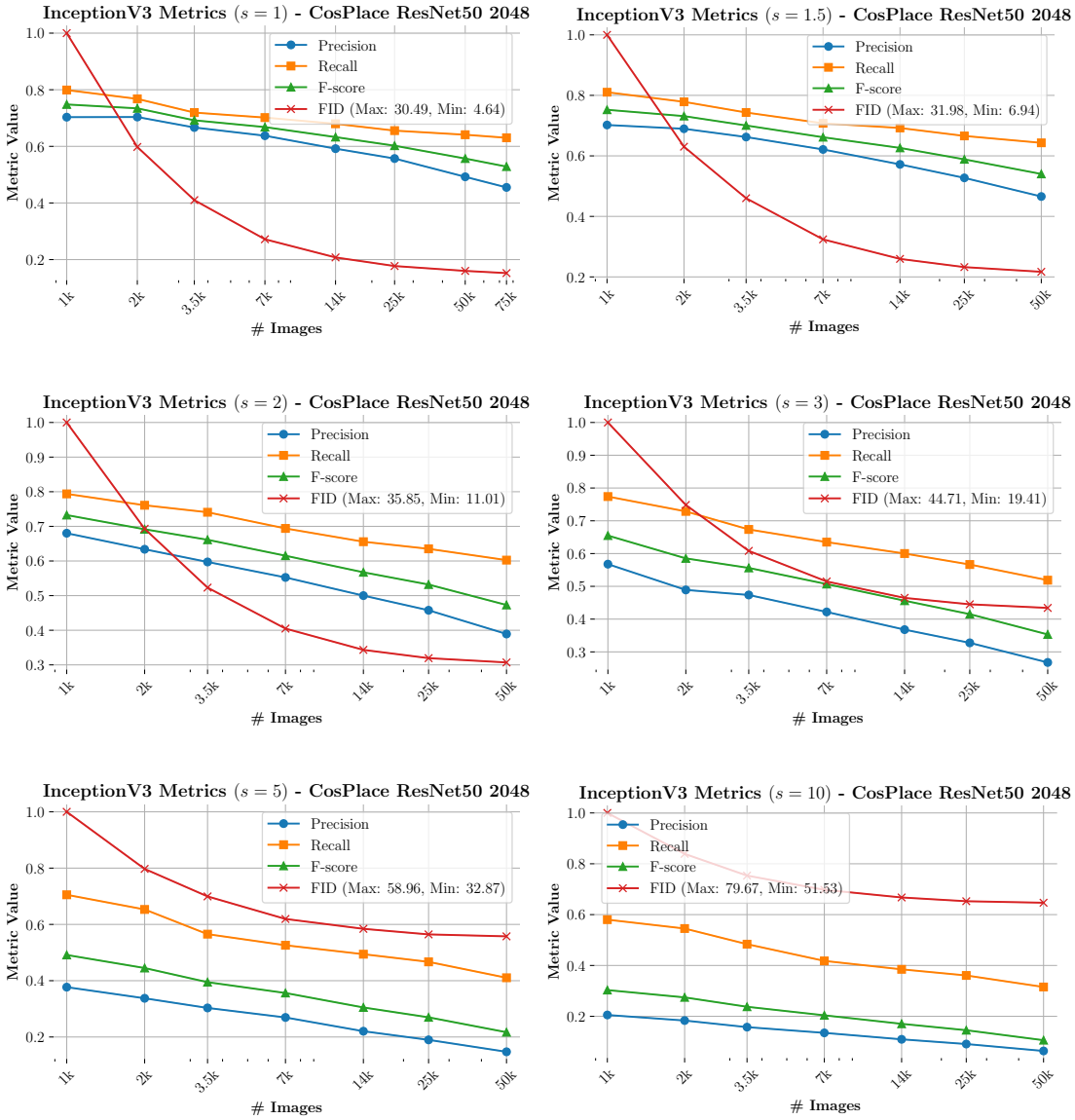


Figure 6.3: Plots of visual metrics for LDM conditioned on CosPlace's output space [52] with $d = 2048$, using InceptionV3, while varying the scale parameter s for CFG [39]. The number of steps in the accelerated generation process is fixed at $S = 250$.

strategy for the other models. In this section, we also use this model to select hyperparameters that influence the DM reverse process, which contribute to the qualitative results.

The relevant hyperparameters in this context are η, which controls the level of

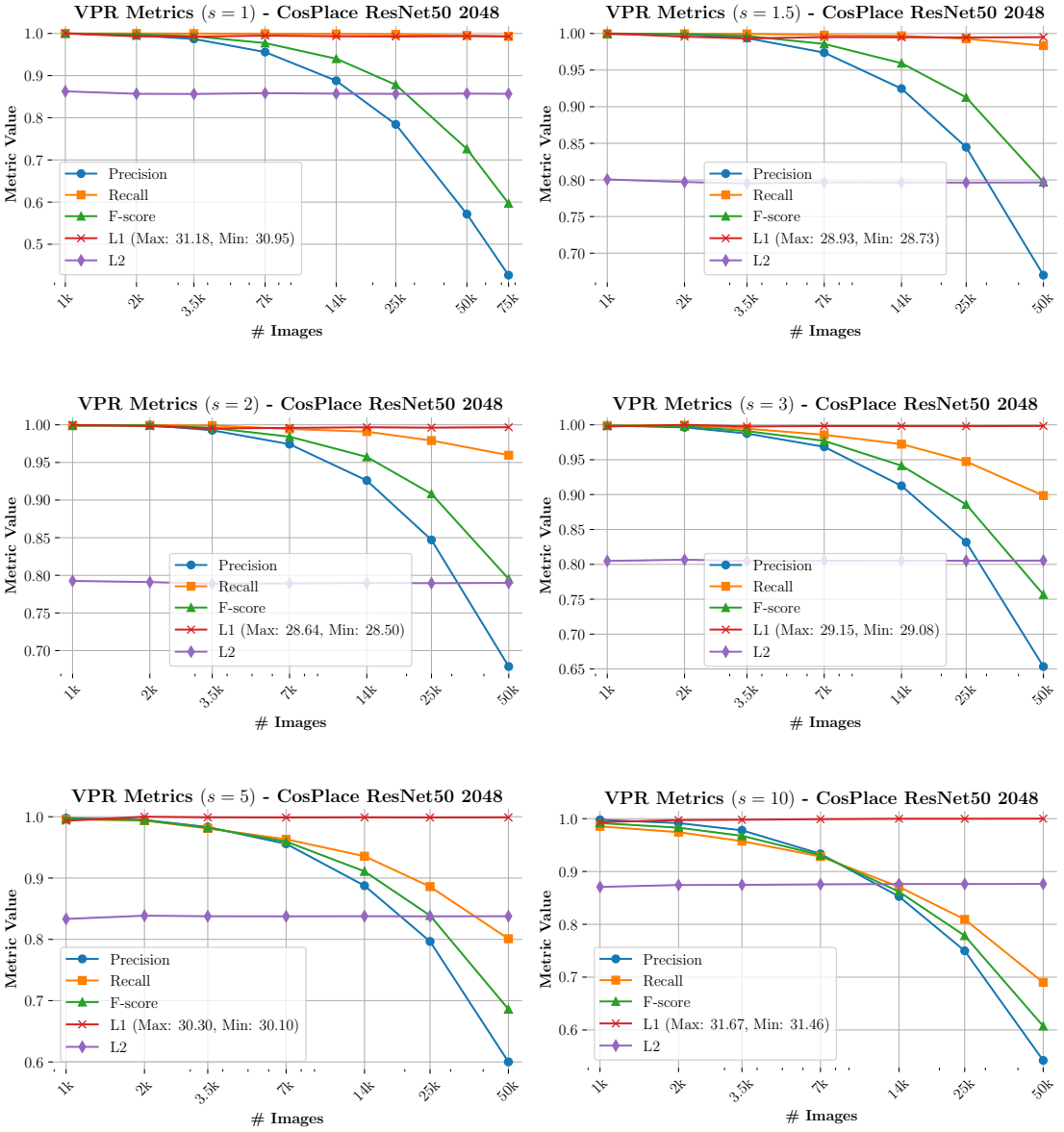


Figure 6.4: Plots of fidelity metrics for LDM conditioned on CosPlace's output space [52] with $d = 2048$, while varying the scale parameter s for CFG [39]. The number of steps in the accelerated generation process is fixed at $S = 250$.

stochasticity in the reverse process, and S, which determines the number of steps in the accelerated reverse process (see Section 4.3.2 for more details). To set these values, we propose the grid search outlined in Table 6.6.

In Fig. 6.5–6.6, we compare five generated images for each pair of values from the rows of Table 6.6, conditioned on the embedding of the leftmost real image,

Statistic	L_1	L_2
Maximum value	46.90	1.30
Minimum value	21.11	0.58
Mean value	33.54	0.93
Standard Deviation	4.57	0.13
$s = 1$ CFG [39]	30.97	0.86
$s = 2$ CFG [39]	28.54	0.79

Table 6.5: Summary of L_1 and L_2 metrics for CosPlace [52] classes in the validation set of Table 6.2. The second half of the table reports the metrics for our LDM model, with scale parameter $s \in \{1, 2\}$ for CFG [39].

S	η
200	0.0
200	1.0
250	0.0
250	1.0
500	0.0
500	1.0
1000	1.0

Table 6.6: Grid search for the number of steps S in the accelerated generation process and the hyperparameter η controlling the level of stochasticity.

which is randomly selected from the queries in the SF-XL [52] validation set[3].

As observed, increasing the number of steps S does not significantly affect the visual appearance of the generated images. However, using $\eta = 1$ produces slightly more aesthetically pleasing results. It is important to note that, for evaluating the information content, $\eta = 1$ introduces noise at each step of the reverse process, while in the case of $\eta = 0$, the noise is limited to the initial random noise $\mathbf{z}_T \sim p(\mathbf{z}_T) = \mathcal{N}(\mathbf{z}_T; \mathbf{0}, \mathbf{I})$ (see Section 4.3.2).

Based on these observations, we choose to fix $\eta = 0$ throughout this thesis to control the level of stochasticity. The number of steps S is determined with computational considerations in mind, especially considering that Rombach et al. [42] found diminishing improvements in image quality for values of $S > 250$. We choose $S = 250$ as a balanced option, as it allows us to achieve all results within a

[3]For improved visualization, please consider zooming in on the images.

reasonable time frame.

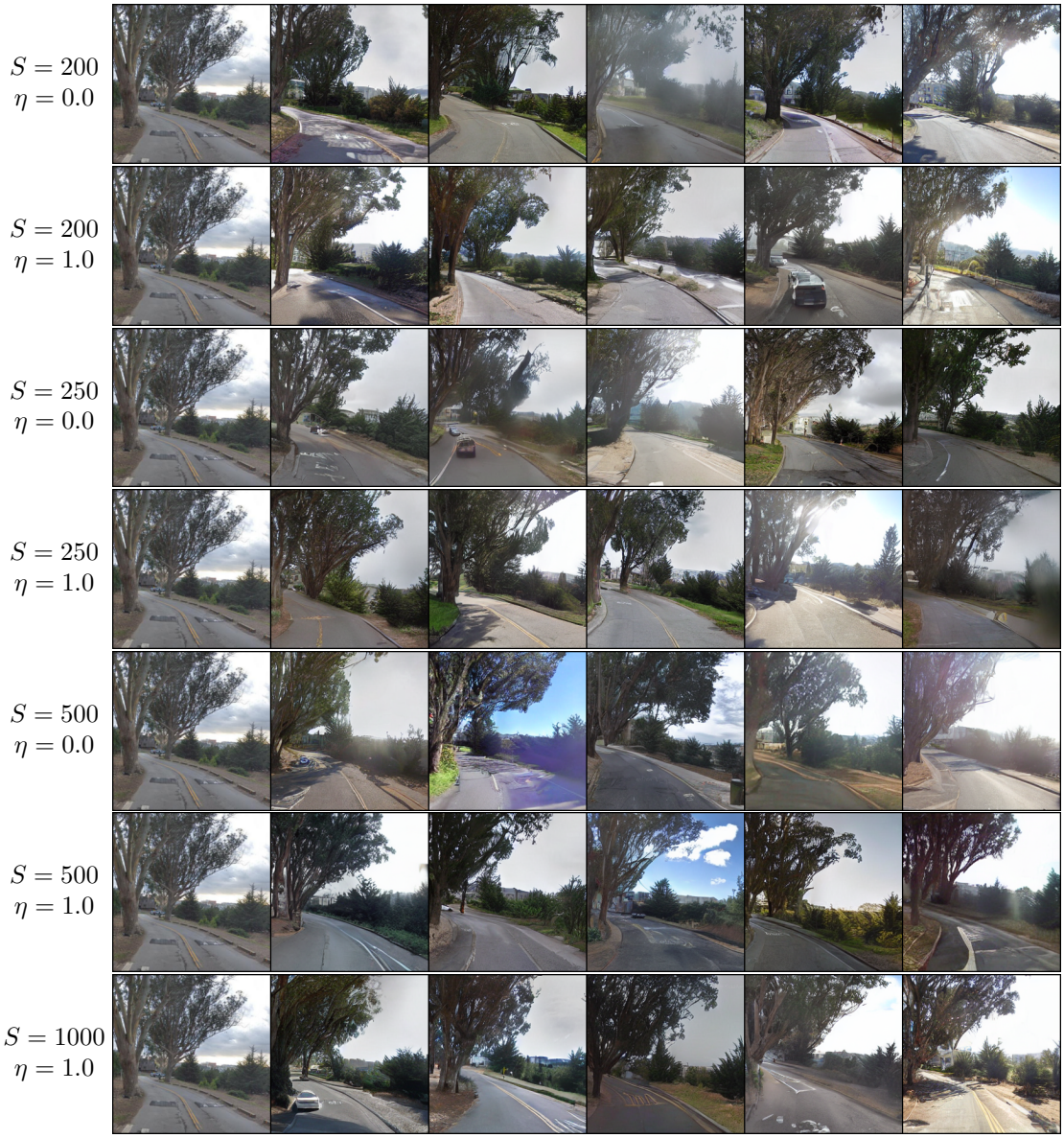


Figure 6.5: Figure 1 showing five generated images conditioned on a query from the SF-XL [52] validation split (shown in the first column). The images are generated by varying the number of steps S in the accelerated reverse process and the hyperparameter η, which controls the level of stochasticity. The specific values used to generate each image set are indicated on the left of each corresponding set.

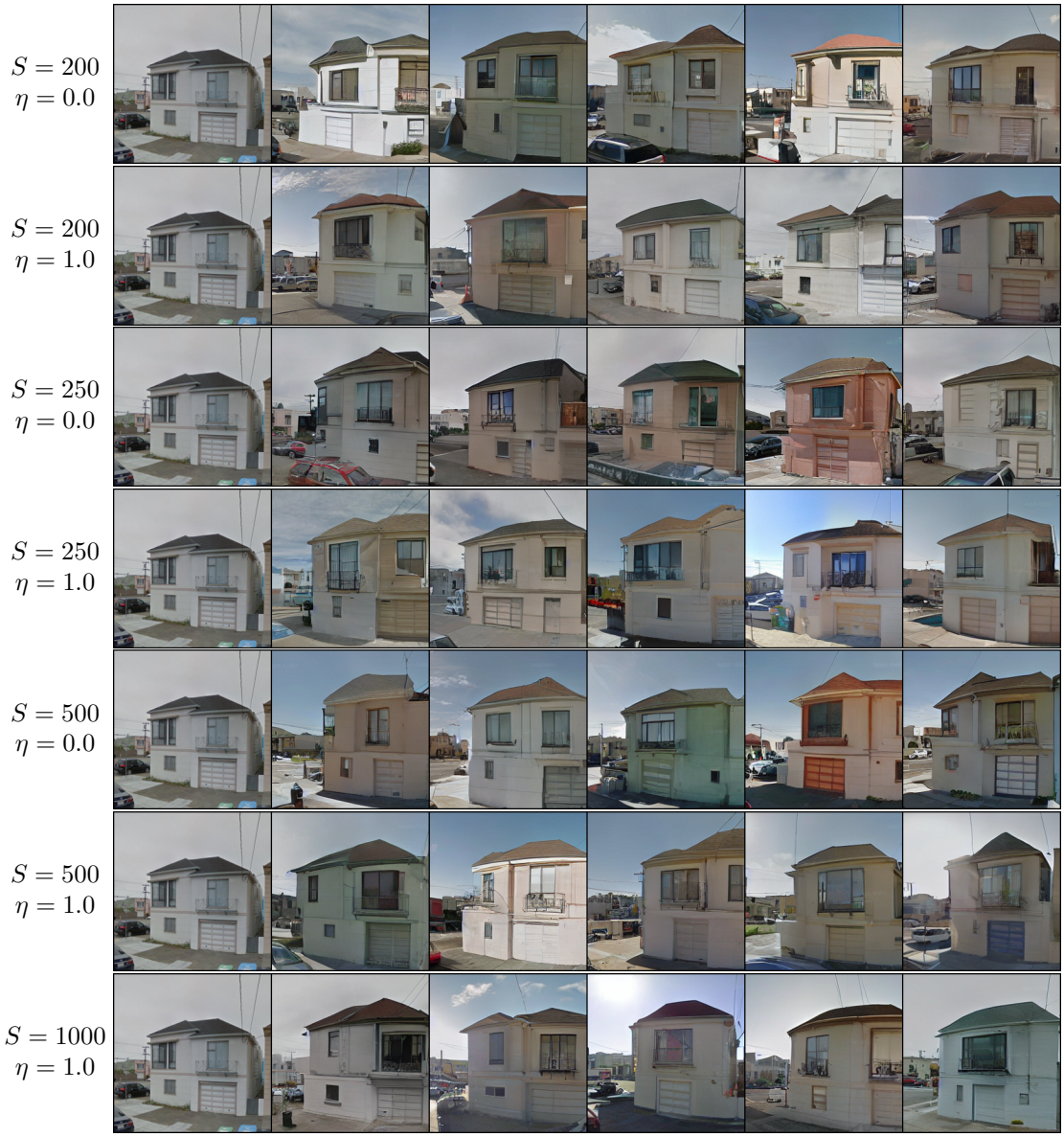


Figure 6.6: Figure 2 showing five generated images conditioned on a query from the SF-XL [52] validation split (shown in the first column). The images are generated by varying the number of steps S in the accelerated reverse process and the hyperparameter η, which controls the level of stochasticity. The specific values used to generate each image set are indicated on the left of each corresponding set.

6.2 Prediction Uncertainty

6.2.1 Uncertainty Framework

In Section 1.2, we outlined the second objective of this thesis: enhancing the robustness of pre-trained Visual Place Recognition models by incorporating uncertainty scores during inference. To achieve this, in Section 5.1 and Section 5.2, we introduced two post-hoc techniques proposed by Upadhyay et al. [58, 65]. Building upon their work, we propose a training technique tailored to our specific task.

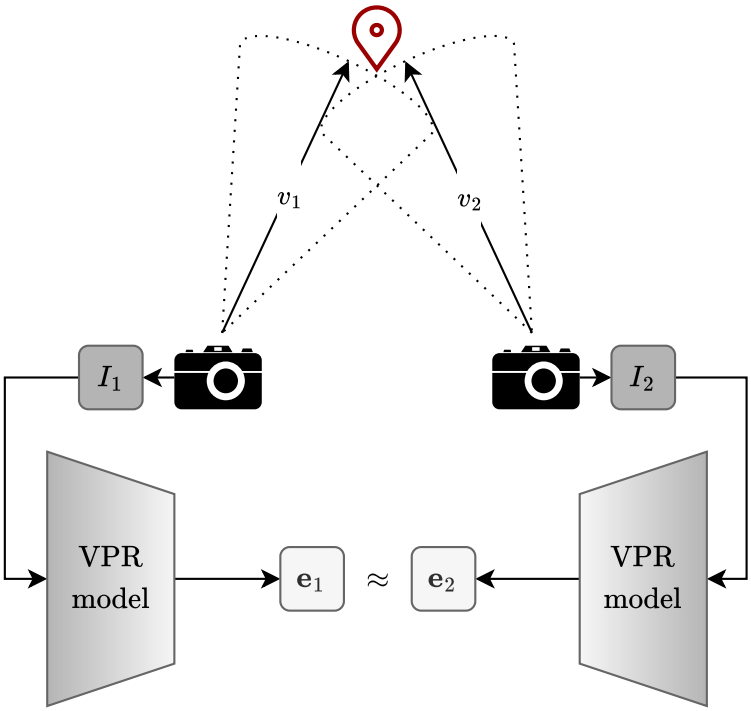


Figure 6.7: Illustration of the viewpoint invariance property for Visual Place Recognition models.

An effective VPR model f is expected to extract embedding vectors $\mathbf{e}_i = f(I_i)$ from images I_i that are *viewpoint invariant*, meaning that images of the same place taken from different viewpoints should result in highly similar embedding vectors. This concept is visually depicted in Fig. 6.7. However, even models designed with viewpoint invariance in mind may not always satisfy this condition. For instance, when occlusions occur and landmarks become obstructed, the Visual Place Recognition model might produce embedding vectors that are far apart from one another. Such situations often arise in real-world environments, particularly in crowded areas such as urban settings. In these cases, the VPR model's predictions

may be negatively affected, highlighting the need for an uncertainty estimate to make more informed decisions.

Given the absence of ground-truth labels for our task, we propose leveraging the viewpoint invariance property by training BayesCap [58] on top of pre-trained VPR models. To do this, we reformulate the loss function from Eq. 5.6 as follows:

$$\mathcal{L}_{\text{BC}} = \lambda_1 \sum_{i=1}^{N} \left| \tilde{\mathbf{y}}_i - \mathbf{e}_i^{(1)} \right| + \lambda_2 \sum_{i=1}^{N} \left(\frac{\left| \tilde{\mathbf{y}}_i - \mathbf{e}_i^{(2)} \right|}{\tilde{\boldsymbol{\alpha}}_i} \right)^{\tilde{\boldsymbol{\beta}}_i} - \log \frac{\tilde{\boldsymbol{\beta}}_i}{\tilde{\boldsymbol{\alpha}}_i} + \log \Gamma(\frac{1}{\tilde{\boldsymbol{\beta}}_i})$$

where $\mathbf{e}_i^{(1)}$ and $\mathbf{e}_i^{(2)}$ are embedding vectors extracted from images of the same location but taken from different viewpoints, and $\tilde{\mathbf{y}}_i$ represents the reconstruction of the embedding $\mathbf{e}_i^{(1)}$.

Additionally, inspired by the cross-modal alignment of ProbVLM [65], discussed in Section 5.2, we propose further training BayesCap [58] by imposing a form of cycle consistency between reconstructions $\tilde{\mathbf{y}}_i^{(1)}$ and $\tilde{\mathbf{y}}_i^{(2)}$ generated from embedding vectors $\mathbf{e}_i^{(1)}$ and $\mathbf{e}_i^{(2)}$, respectively. This leads to the following loss function, resulting in the model we term BayesCapCycle:

$$\begin{aligned}
\mathcal{L}_{\text{BCC}} = & \frac{\lambda_1}{2} \sum_{i=1}^{N} \left| \tilde{\mathbf{y}}_i^{(1)} - \mathbf{e}_i^{(1)} \right| + \frac{\lambda_2}{2} \sum_{i=1}^{N} \left(\frac{\left| \tilde{\mathbf{y}}_i^{(1)} - \mathbf{e}_i^{(2)} \right|}{\tilde{\boldsymbol{\alpha}}_i^{(1)}} \right)^{\tilde{\boldsymbol{\beta}}_i^{(1)}} - \log \frac{\tilde{\boldsymbol{\beta}}_i^{(1)}}{\tilde{\boldsymbol{\alpha}}_i^{(1)}} + \log \Gamma(\frac{1}{\tilde{\boldsymbol{\beta}}_i^{(1)}}) \\
& + \frac{\lambda_1}{2} \sum_{i=1}^{N} \left| \tilde{\mathbf{y}}_i^{(2)} - \mathbf{e}_i^{(2)} \right| + \frac{\lambda_2}{2} \sum_{i=1}^{N} \left(\frac{\left| \tilde{\mathbf{y}}_i^{(2)} - \mathbf{e}_i^{(1)} \right|}{\tilde{\boldsymbol{\alpha}}_i^{(2)}} \right)^{\tilde{\boldsymbol{\beta}}_i^{(2)}} - \log \frac{\tilde{\boldsymbol{\beta}}_i^{(2)}}{\tilde{\boldsymbol{\alpha}}_i^{(2)}} + \log \Gamma(\frac{1}{\tilde{\boldsymbol{\beta}}_i^{(2)}})
\end{aligned}$$

By considering both embedding vectors, $\mathbf{e}_i^{(1)}$ and $\mathbf{e}_i^{(2)}$, we expect the model to converge more quickly, yielding improved results with fewer training epochs. This aspect is explored in more detail in Section 6.2.5.

6.2.2 Implementation Details

To apply the training strategies outlined in Section 6.2.1, we utilize the EigenPlaces [19] dataset, employing the same latitude-based split described in Section 6.1.2. To ensure that images correspond to the same location, we use panoramas. The number of panoramas in both the training and validation sets is presented in Table 6.7.

For BayesCap [58], we fix the architecture to a 3-layer MLP with ReLU activations in the first two layers and dropout applied only in the second layer. The architectural and training hyperparameters are detailed in Table 6.8, where d represents the embedding dimension of the upstream VPR model.

Set	Latitude	# Panoramas
Training	$37.71 - 37.81$	3,267,427
Validation	37.70	163,665

Table 6.7: Training and validation sets derived from the training split of the EigenPlaces [19] dataset used for training BayesCap [58] models.

Hyperparameter	Value
Input Dimension	d
1-st layer output dimension	$d/2$
2-nd layer output dimension	$d/2$
3-rd layer output dimension	d
Activation Function	ReLU
Dropout Probability	0.1
λ_1	1.0
λ_2	10^{-4}
Batch Size	96
Iterations per epoch	$5k$
Learning Rate	10^{-4}

Table 6.8: Architectural and training hyperparameters for BayesCap [58].

6.2.3 From Per-Feature to Instance Level Uncertainty

BayesCap models [58] are capable of modeling both aleatoric and epistemic uncertainties, as defined in Eq. 5.5 and Eq. 5.7, respectively. These uncertainties are then summed to compute the total uncertainty, as shown in Eq. 5.8. However, these uncertainties are computed at feature level, so they must be aggregated to derive a single uncertainty score for the embedding $\mathbf{e}_q = f(I_q)$, obtained from the query image I_q.

Let $[\cdot]_i$ denote the function that extracts the i-th component of its argument. Each feature's uncertainty score $[\tilde{\boldsymbol{\sigma}}_q^2]_i$ for the query I_q has its own range of variation. This means the ranges of $[\tilde{\boldsymbol{\sigma}}_q^2]_i$ and $[\tilde{\boldsymbol{\sigma}}_q^2]_j$, for $i \neq j$, may be disjoint and significantly different. Consequently, using a simple arithmetic mean to aggregate the uncertainties would be inappropriate.

To address this variability, the geometric mean is used instead, as it prevents disproportionately weighting features with larger uncertainty scores. Therefore, for

the remainder of the thesis, the uncertainty u_q for query I_q is computed as follows:

$$u_q = \left(\prod_{i=1}^{d} [\tilde{\boldsymbol{\sigma}}_q^2]_i \right)^{\frac{1}{d}}$$

where d is the dimensionality of embedding vector $\mathbf{e}_q$.

6.2.4 Metrics

To evaluate the quality of uncertainty estimates produced by trained BayesCap [58] models, we use the same metrics as Upadhyay et al. [65]: *Spearman's rank correlation coefficient* (S) and the *coefficient of determination* (R^2). Additionally, we include the *Area Under the Sparsification Curve* (AUSC), as introduced by Warburg et al. [66].

To compute these metrics, we first calculate the uncertainty scores u_q for all queries I_q in a dataset. We then apply an adaptive binning strategy to ensure each bin B_i contains an adequate number of samples (*i.e.*, not too few) and has a sufficiently wide range (*i.e.*, not too narrow). Once M bins are created, we compute the mean uncertainty score ($\bar{u}_i$) and mean Recall@1 ($\bar{r}_i$) for each bin. After ranking the data, ensuring that both mean values become ordinal and fall within the same range, we calculate both S and R^2. From here on, we will refer to the ranked mean uncertainty scores as "uncertainty levels".

The Spearman's rank correlation coefficient $S \in [-1, 1]$ measures the *monotonic relationship* between uncertainty levels and ranked mean Recall@1. Since we expect that higher uncertainty levels correspond to lower mean Recall@1, the ideal relationship is a strictly decreasing monotonic one, which would yield $S = -1$.

The coefficient of determination $R^2 \in [-\infty, 1]$ evaluates the regression fit between uncertainty levels and ranked mean Recall@1, allowing us to assess if the decline in performance follows a linear trend. The more linear the trend, the higher the score, with the ideal value being $R^2 = 1$.

Finally, the sparsification curve is generated by iteratively removing queries with the highest uncertainty scores and recalculating the Recall@1 on the remaining samples. The curve should increase monotonically, and the area under the curve, AUSC, provides a way to compare different models.

6.2.5 Quantitative Results

For the quantitative evaluation using the metrics outlined in Section 6.2.4, we focus on BayesCap [58] models trained on top of CosPlace [52] with embedding dimension $d = 2048$, consistent with the baseline used in Sections 6.1.4–6.1.6.

In total, we trained three BayesCap [58] models: one corresponding to BayesCap-Cycle, which was trained using the loss function $\mathcal{L}_{\text{BCC}}$, and two others trained

using the loss function $\mathcal{L}_{\text{BC}}$. A summary of these trained models is provided in Table 6.9, and their training curves are available in Appendix C.3.

Method	Loss Function	VPR Model	d	# Epochs
BayesCap [58]	$\mathcal{L}_{\text{BC}}$	CosPlace [52]	2048	30
BayesCap [58]	$\mathcal{L}_{\text{BC}}$	CosPlace [52]	2048	50
BayesCapCycle	$\mathcal{L}_{\text{BCC}}$	CosPlace [52]	2048	30

Table 6.9: Summary of trained BayesCap [58] models.

The adaptive binning strategy mentioned in Section 6.2.4 aims to balance the minimum number of samples and the minimum bin width for each bin B_i, while also considering constraints on the number of bins M, including both minimum and maximum limits. Starting with an initial set of bins with equal width, where the bin edges are determined by combining the constraints on minimum width, maximum number of bins, and minimum number of bins, the algorithm iteratively merges adjacent bins (*i.e.*, B_i, B_j such that $|i - j| \leq 1$) until all bins meet the minimum sample size requirement or a maximum number of merge attempts is reached. The full Python code for this strategy is provided in Appendix C.4.

We evaluate all three models on the validation sets of Pittsburgh-30k (Pitts30k) [15], SF-XL [52], and MSLS [78]. The corresponding results are shown in Fig. 6.8, Fig. 6.9, and Fig. 6.10. We present both calibration plots, illustrating how the mean Recall@1 varies with increasing uncertainty levels, and sparsification curves, where the x-axis represents the percentage of queries removed and the y-axis represents Recall@1 computed on the remaining queries.

Since the calibration plot depends on the adaptive binning strategy, which varies the number of uncertainty levels and bin widths across models, it is not a fair comparison metric. As such, we focus primarily on the sparsification curves for model comparison.

On the Pitts30k [15] validation set, all models exhibit a monotonically increasing sparsification curve. BayesCapCycle and the BayesCap [58] model trained for 50 epochs show nearly identical curves, with the latter achieving a higher AUSC value. The BayesCap [58] model trained for 30 epochs struggles to follow the same trend, as seen by the curve not reaching a Recall@1 of 0.98. The similarity in the curves between BayesCapCycle and the 50-epoch BayesCap [58] model supports our hypothesis that using the $\mathcal{L}_{\text{BCC}}$ loss function leads to better results with fewer training epochs.

Results on the SF-XL [52] validation set are harder to interpret, as all models perform poorly. This may be due to CosPlace [52] being specifically trained on SF-XL [52], making it harder to distinguish between certain and uncertain queries. Nevertheless, BayesCapCycle and the BayesCap [58] model trained for 50 epochs

show similar trends.

The trend shifts on the more challenging MSLS [78] validation set, where the BayesCap [58] model trained for 50 epochs delivers the best performance. However, as with the previous validation sets, BayesCapCycle demonstrates a better sparsification curve than the BayesCap [58] model with the same number of epochs.

Overall, the results suggest that BayesCapCycle offers a good trade-off between performance and computational efficiency, as it delivers strong results even with fewer epochs compared to BayesCap [58], making it a suitable choice when computational resources are limited.

Additionally, further plots showing how the uncertainties calculated by all models distribute according to Recall@1 are available in Appendix C.5.

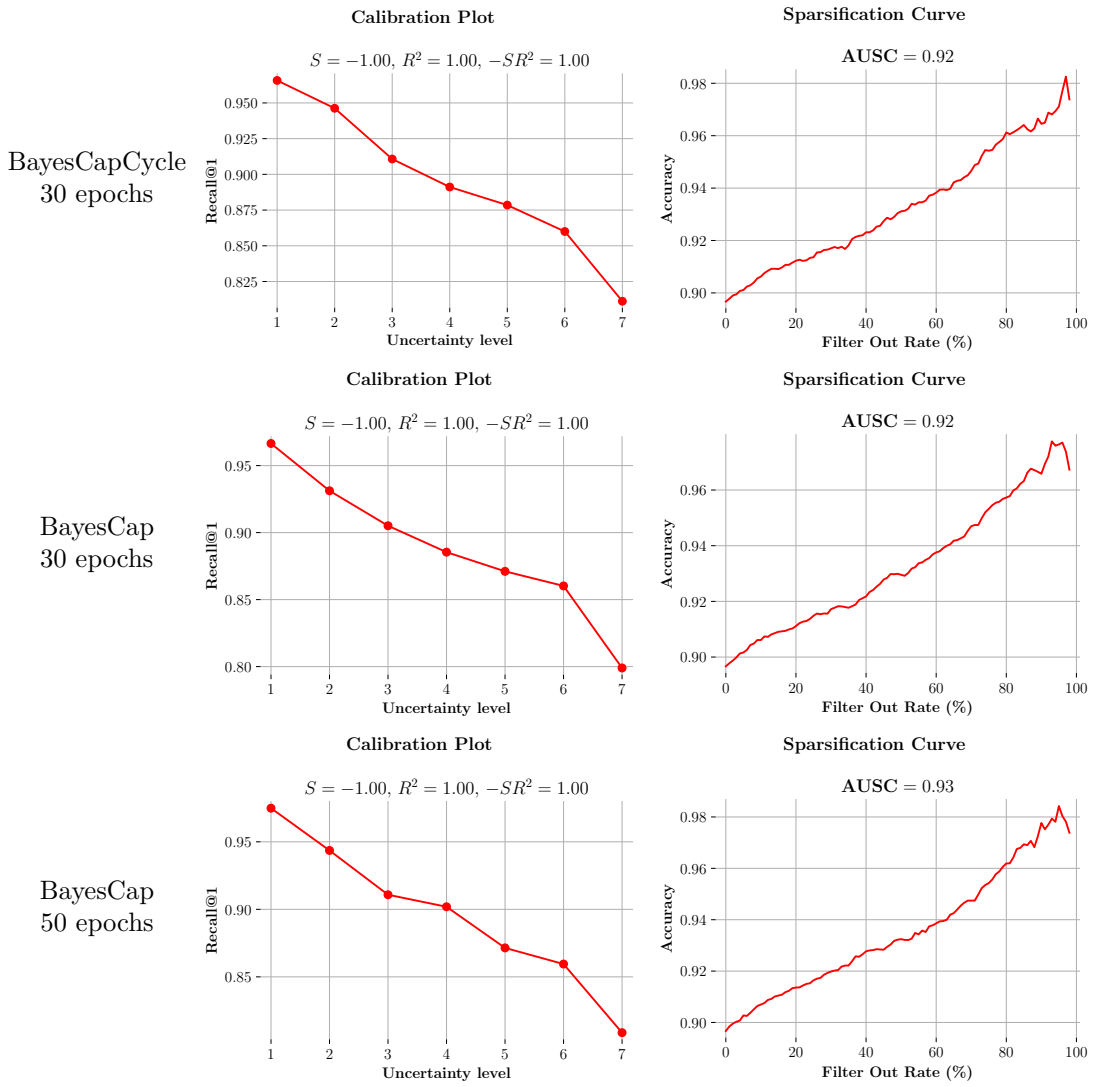


Figure 6.8: Calibration plots and sparsification curves for all trained BayesCap [58] models, computed on the Pitts30k [15] validation set.

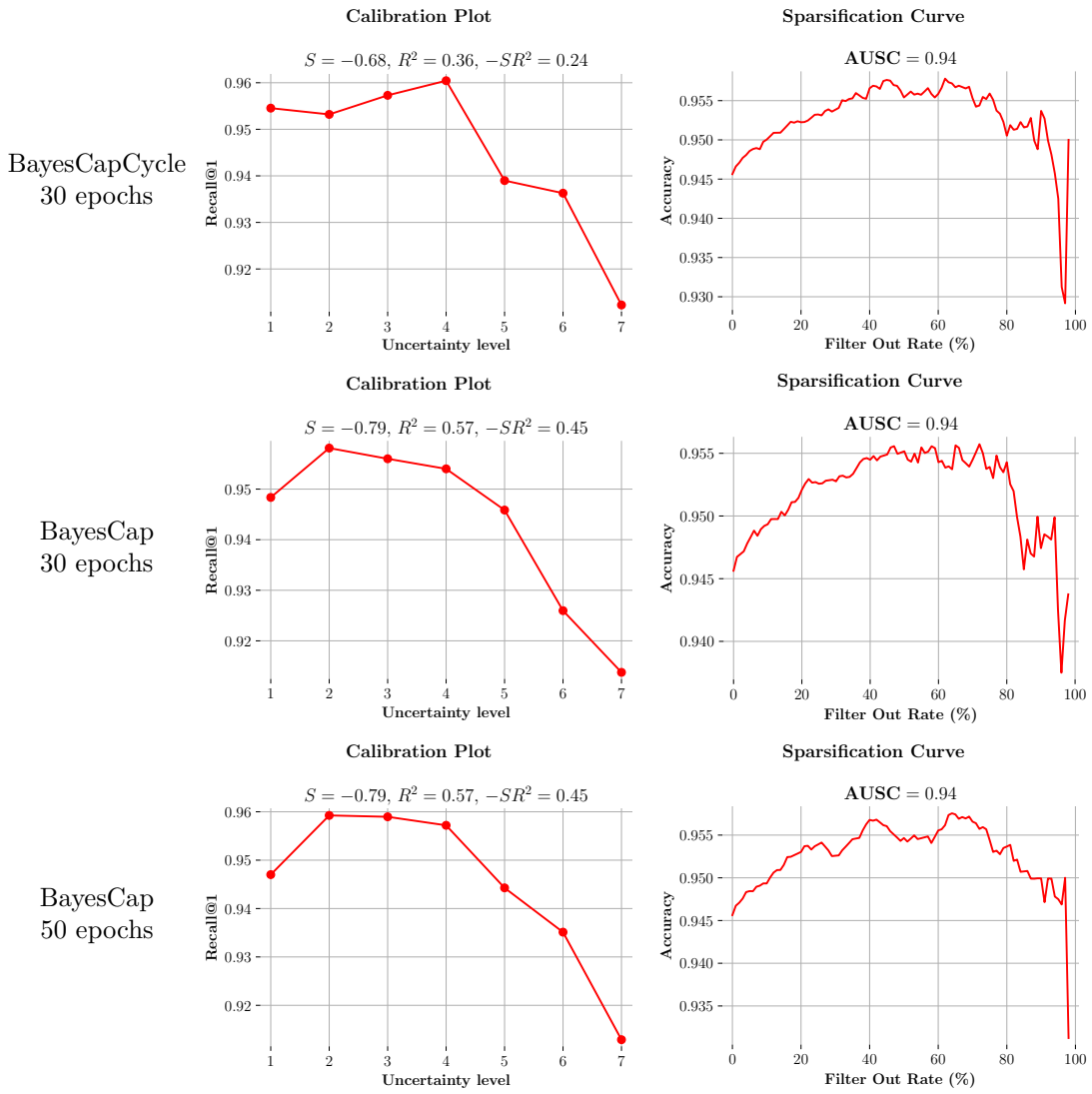


Figure 6.9: Calibration plots and sparsification curves for all trained BayesCap [58] models, computed on the SF-XL [52] validation set.

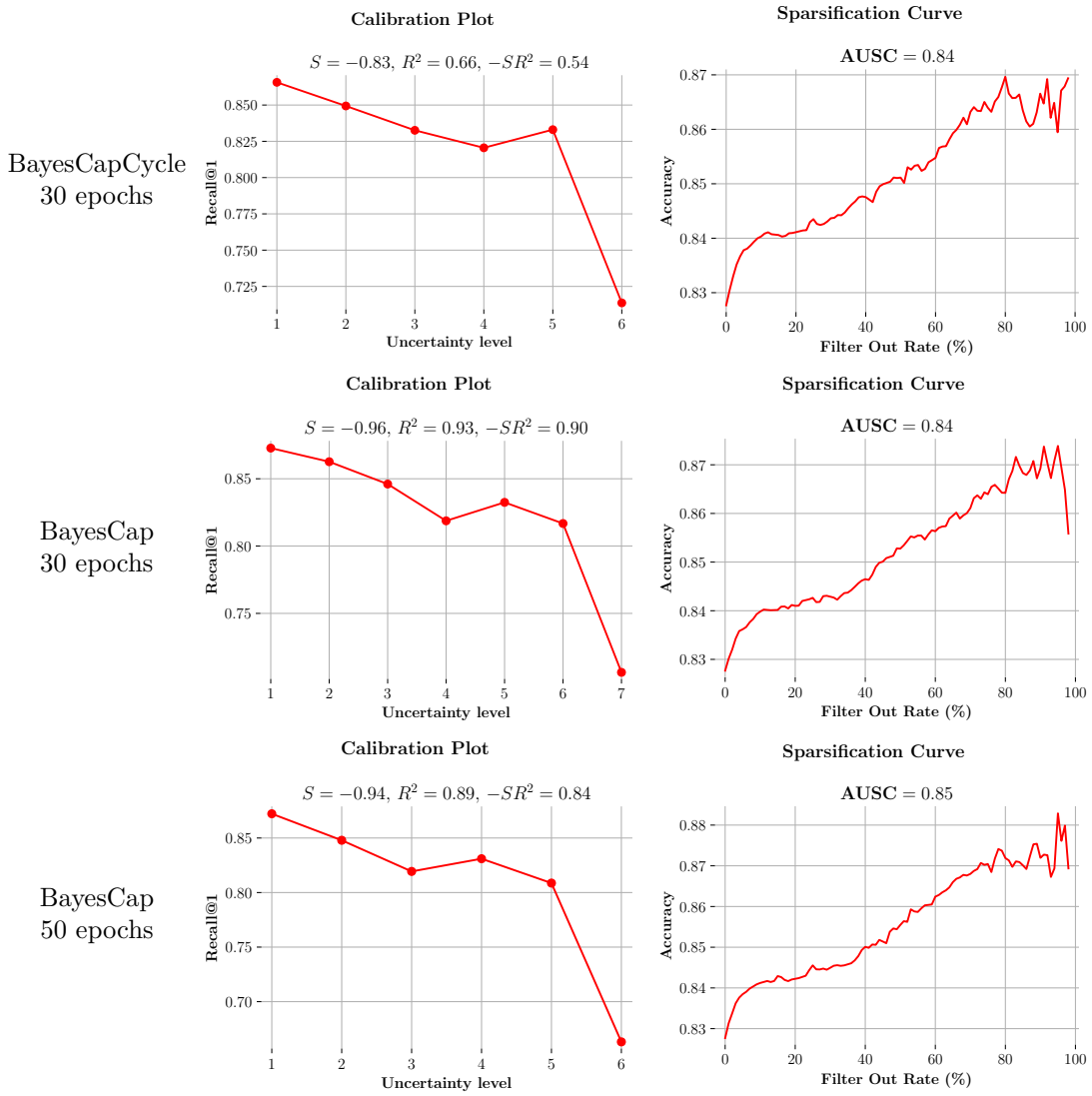


Figure 6.10: Calibration plots and sparsification curves for all trained BayesCap [58] models, computed on the MSLS [78] validation set.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, two research gaps about the Visual Place Recognition (VPR) task have been explored. In the following, the primary findings and limitations associated with the methodologies employed are summarized.

7.1.1 Findings

The first research question addresses the type of information retained in image embeddings produced by VPR models. This was examined by utilizing Latent Diffusion Models, conditioned on the outputs of VPR models. The experiments demonstrated that these Generative Artificial Intelligence (Gen-AI) models effectively capture the relationships between original images, in pixel space, and the embeddings, in the VPR model's output space. As shown numerically in Section 6.1.5 and Appendix C.2, these Gen-AI models can generate synthetic images that retain the informational content of the conditioning embeddings. Notably, generating multiple images based on the same embedding reveals that consistent features of real images preserved in the embeddings appear across all generated images. Meanwhile, varying or transient elements not represented in the embeddings are introduced solely by the LDM's reverse process. This capability enhances the interpretability of VPR models, making their output space more transparent. Insights into the output space can be gained by sampling and visualizing hypothetical embeddings. A practical application of this is the interpretation of centroids generated by clustering methods, which might not correspond to real images in a dataset (*e.g.*, no matching image exists). Additionally, CAV [79] directions can be

interpreted, enabling speculation about the features that are likely to emerge as we progress along a specific direction in the output space. Moreover, comparing the outputs of different VPR models for the same original image allows for examining the varying information preserved by each model and which aspects of the original image they emphasize. Lastly, cross-model translations are possible, as embeddings sampled in one VPR model's output space can be synthesized and projected onto another VPR model's output space.

The second research question, focusing on how to produce uncertainty estimates for the deterministic predictions of VPR models, was tackled by proposing adapted and novel training strategies for BayesCap [58], an existing post-hoc uncertainty estimation technique in the current literature. The results, as presented in Section 6.2.5, indicate that these uncertainty scores enhance the robustness and reliability of pre-trained Visual Place Recognition models during inference. However, there is still significant room for improvements.

7.1.2 Limitations

The main limitations of the methodologies in this thesis stem from the interpretability framework that relies on Latent Diffusion Models. These models are highly computationally intensive, both during training and inference. For example, training a model from scratch on an NVIDIA A100 GPU takes over 7 days, which makes it impractical in many scenarios. Additionally, the success of the interpretability framework depends on the LDM model's ability to learn the relationships between image embeddings and their corresponding original images. If the LDM model fails in this task, the embeddings cannot be effectively interpreted. Furthermore, to fully understand the information within the image embeddings, multiple images must be generated, which increases inference time. However, preliminary experiments indicate that generating between 7 and 9 images is typically sufficient, also considering that generating too many images can make analysis challenging for humans.

The uncertainty estimation framework also has its limitations, as demonstrated in Appendix C.5, where the uncertainty distributions do not clearly differentiate between correctly and incorrectly matched queries. This suggests potential failures in uncertainty estimation, where a query that is ultimately mispredicted by the VPR model may have a low uncertainty score, preventing the error from being effectively identified and mitigated during production.

7.2 Future Work

The interpretability framework presented in this thesis opens up several potential real-world applications and avenues for further research. First, by examining the

information embedded in image embeddings, insights into the failure modes of Visual Place Recognition models can be gained. This could enable the refinement of training strategies and the design of improved model architectures. Advanced clustering techniques can be used to explore fine-grained features captured in the embeddings, enhancing the interpretability of VPR models' output spaces. Additionally, investigating the transferability of embeddings across different VPR models could lead to better generalization across diverse domains and datasets, or facilitate the effective integration of multiple models. Generating images in less densely populated regions of the output space could boost model performance by fine-tuning the model on these synthetic images or even enable the creation of new synthetic VPR datasets. Moreover, this framework is not limited to the VPR task and can be applied to any model that outputs an embedding vector. For instance, Concept Bottleneck Models [80] and Concept Embedding Models [81] could benefit from this interpretability tool, helping to understand how concept intervention translates into pixel space, ensuring that activating or modulating a concept has the intended effect. Furthermore, Latent Diffusion Models trained on different data distributions but conditioned on the same model can facilitate the translation of embeddings across domains, such as in style transfer.

The uncertainty estimation framework warrants further investigation to improve the separation between correct and incorrect queries. One way to achieve this is by incorporating regularization terms into the loss function during training, ensuring that incorrect queries are associated with higher uncertainty scores. Another approach could involve designing a new loss function that aggregates multiple desired properties of VPR models, where queries that satisfy all properties are assigned low uncertainty scores, while those that meet fewer constraints are assigned progressively larger uncertainty values.

Appendix A

Variational Lower Bound

A.1 Kullback-Leibler divergence non-negativity

To prove the non-negativity of the KL divergence $D_{KL}(P \,||\, Q)$ between distributions P and Q, we must show that $\log x \leq x - 1 \, \forall x \geq 0$.

The function $\log x$ is concave, meaning it is always bounded above by its first-order Taylor expansion. Specifically, we can use the following approximation at a point x_0:

$$\log x \leq \log x_0 + (x - x_0) \left. \frac{d}{dx} \log x \right|_{x=x_0} = \log x_0 + \frac{x - x_0}{x_0}$$

If we choose $x_0 = 1$, this gives:

$$\log x \leq \log 1 + \frac{x - 1}{1} = x - 1$$

Thus, we have $\log x \leq x - 1$, , which is the required inequality.

Now, applying this to the KL divergence, we proceed as follows:

$$\begin{aligned} -D_{KL}(P \,||\, Q) &= -\int P(x) \log \frac{P(x)}{Q(x)} \, dx = \int P(x) \log \frac{Q(x)}{P(x)} \, dx \\ &\leq \int P(x) \left(\frac{Q(x)}{P(x)} - 1 \right) dx = -\int P(x) \, dx + \int Q(x) \, dx \\ &= -1 + 1 = 0 \end{aligned}$$

Thus, $D_{KL}(P \,||\, Q) \geq 0$, proving that the KL divergence is always non-negative.

Appendix B

Probability Distributions

B.1 Heavy-tailed distribution

Let X be a random variable. The *tail distribution function* is defined as follows:

$$\bar{F}(x) = \Pr[X > x]$$

Here, $\Pr[X > x]$ represents the probability that the random variable X takes values greater than x. The distribution of X is classified as a *heavy-tailed distribution* if the tail distribution function $\bar{F}(x)$ decays polynomially as x increases, specifically:

$$\lim_{x \to \infty} e^{tx} \bar{F}(x) = \infty, \forall t > 0$$

In simpler terms, this implies that there is a significant probability of observing very large values of X.

Appendix C

Experimental Details and Additional Results

C.1 LDM training curves

In this section, we present the training curves (Fig. C.1–C.16) of the Latent Diffusion Models, following the process outlined in Section 6.1.2. Each training runs for 100 epochs, with one epoch consisting of 5,000 iterations and a batch size of 96. The plots are organized by the Visual Place Recognition models specified in Table 6.1.

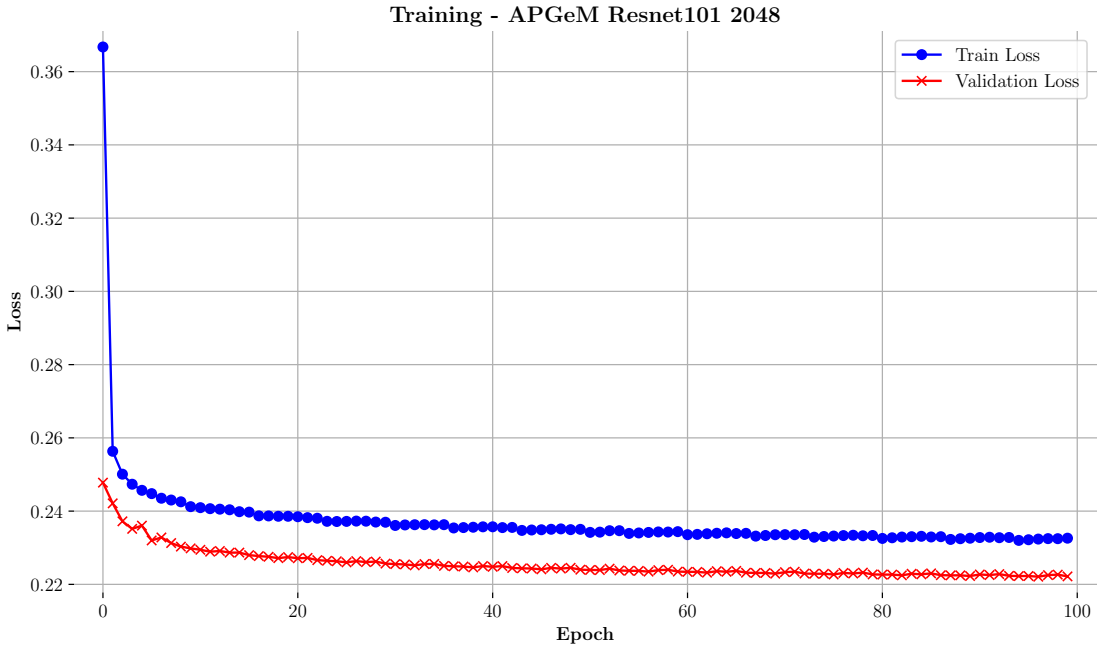


Figure C.1: Training plot of the Latent Diffusion Model conditioned on the output space of AP-GeM [14].

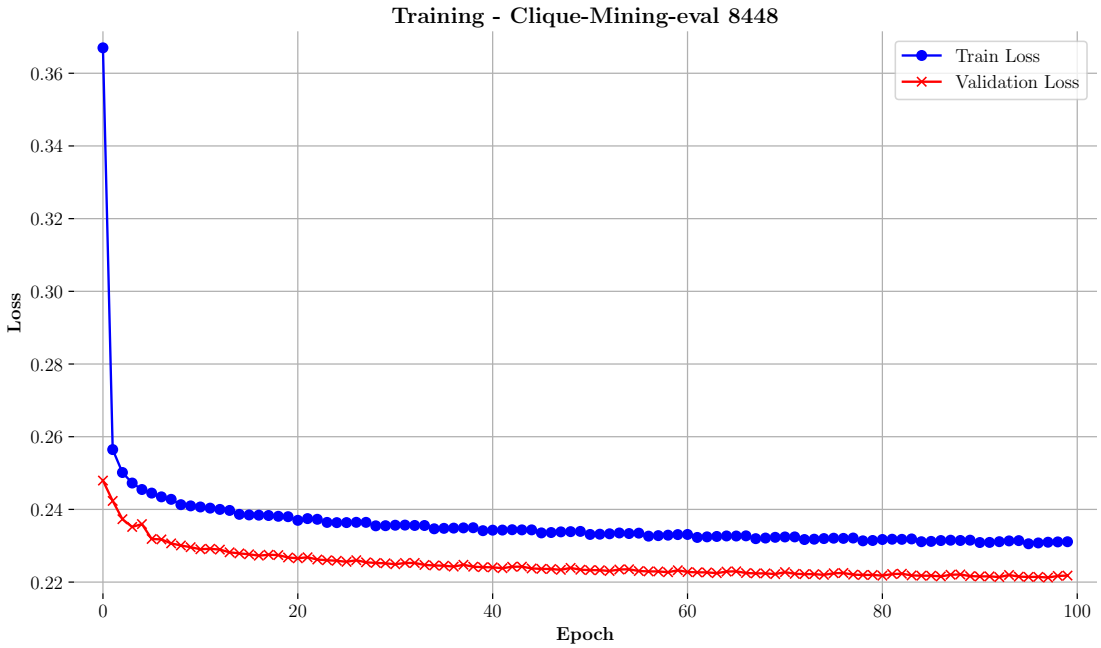


Figure C.2: Training plot of the Latent Diffusion Model conditioned on the output space of CliqueMining [13].

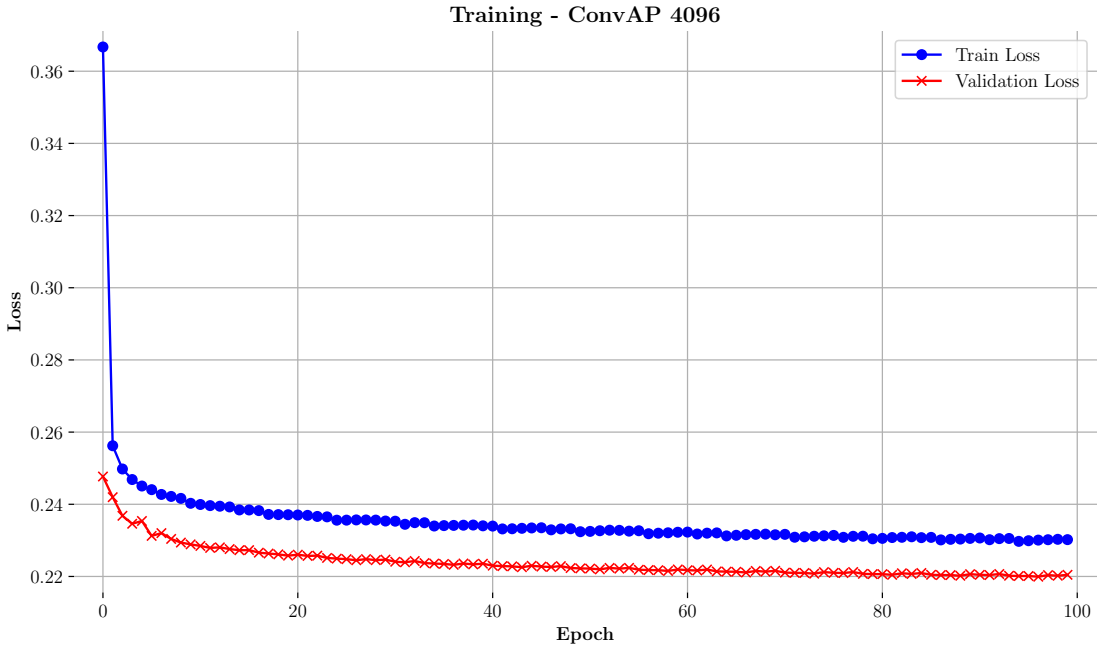


Figure C.3: Training plot of the Latent Diffusion Model conditioned on the output space of Conv-AP [25].

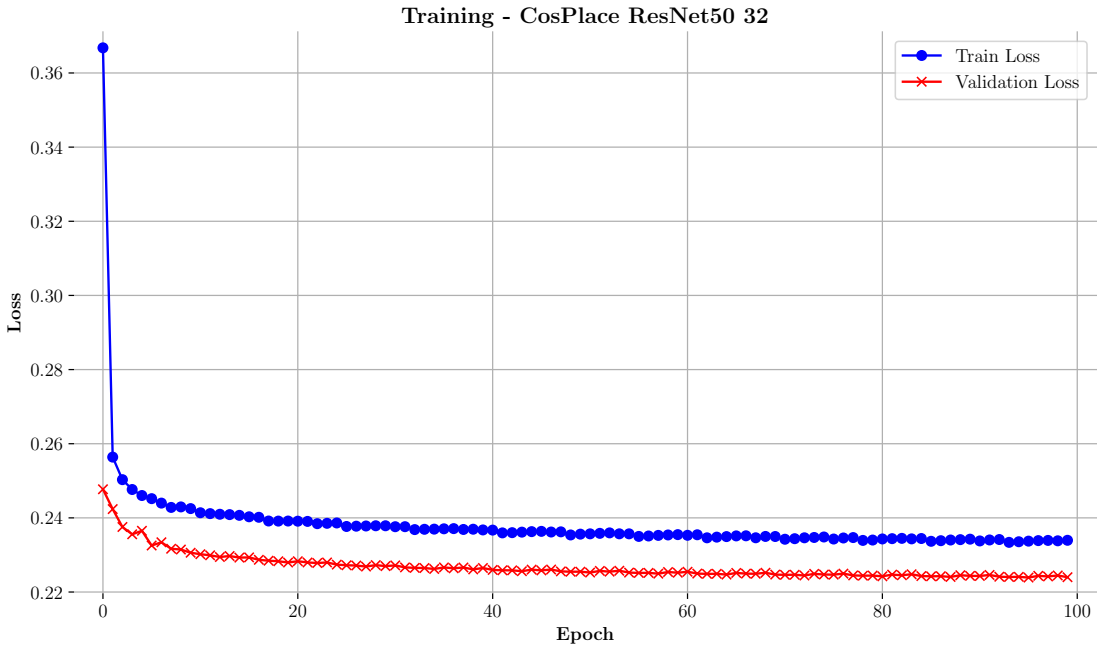


Figure C.4: Training plot of the Latent Diffusion Model conditioned on the output space of CosPlace [52] with $d = 32$.

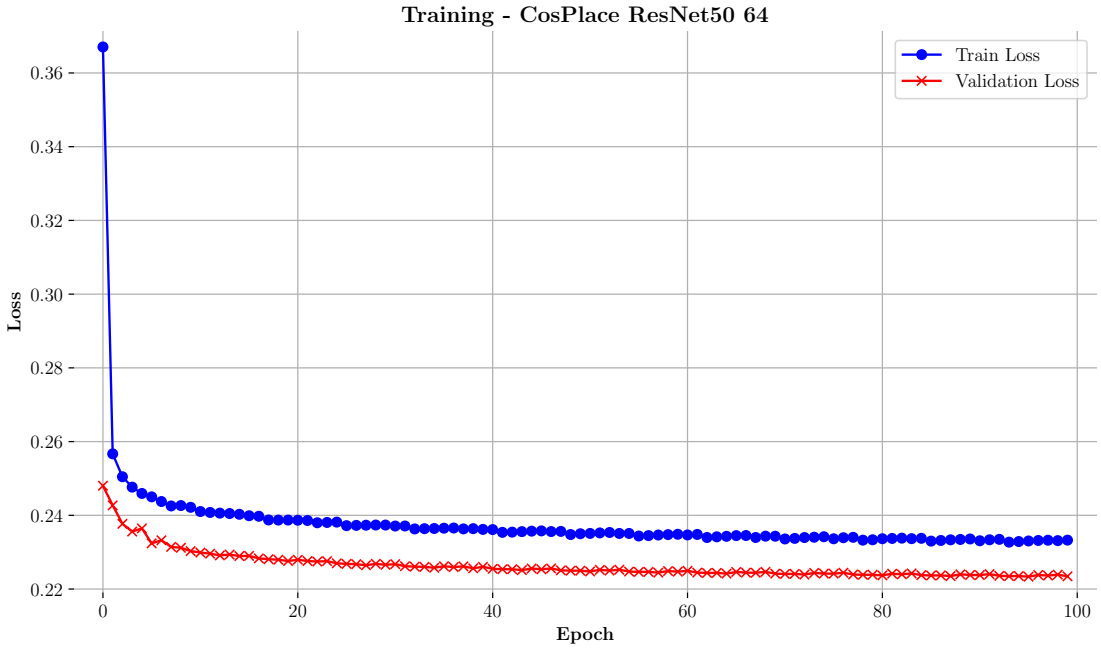


Figure C.5: Training plot of the Latent Diffusion Model conditioned on the output space of CosPlace [52] with $d = 64$.

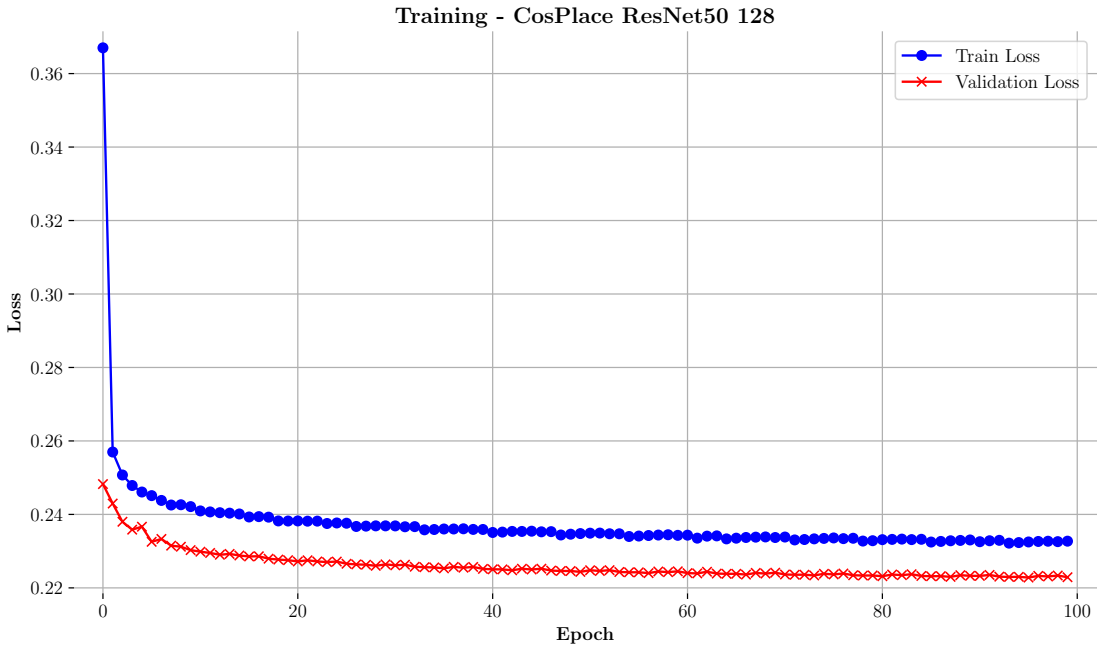


Figure C.6: Training plot of the Latent Diffusion Model conditioned on the output space of CosPlace [52] with $d = 128$.

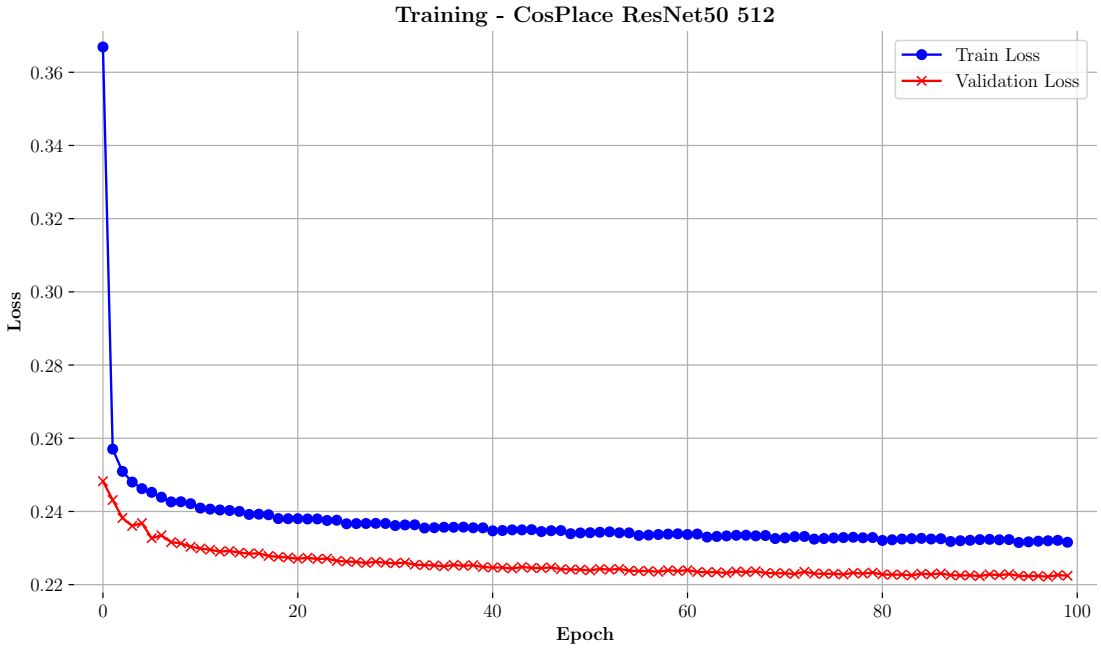


Figure C.7: Training plot of the Latent Diffusion Model conditioned on the output space of CosPlace [52] with $d = 512$.

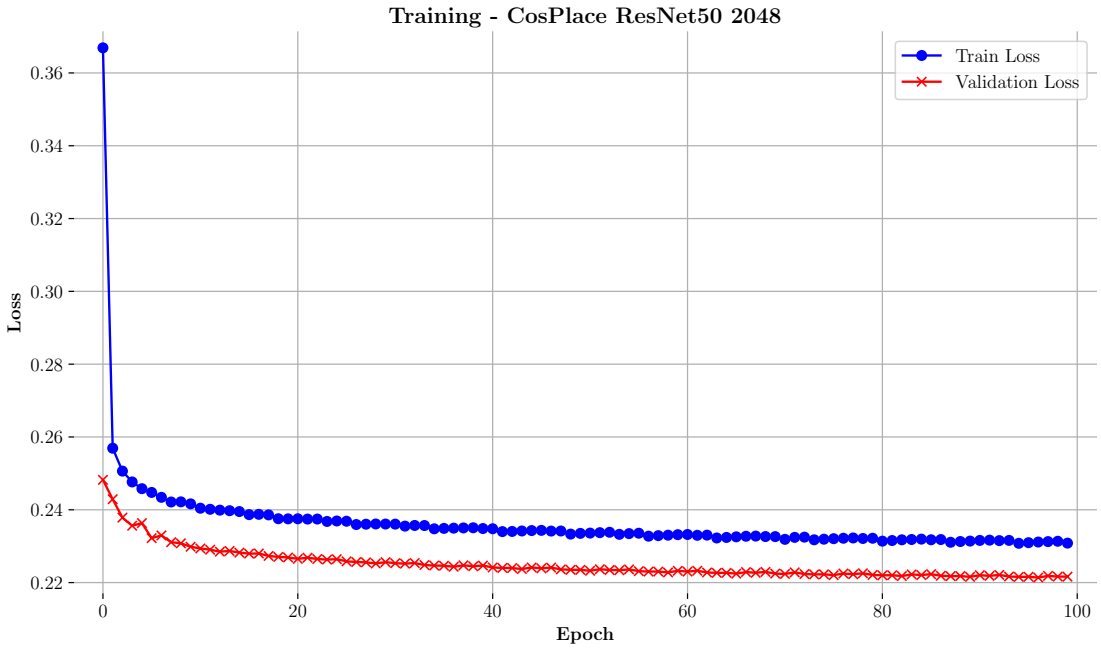


Figure C.8: Training plot of the Latent Diffusion Model conditioned on the output space of CosPlace [52] with $d = 2048$.

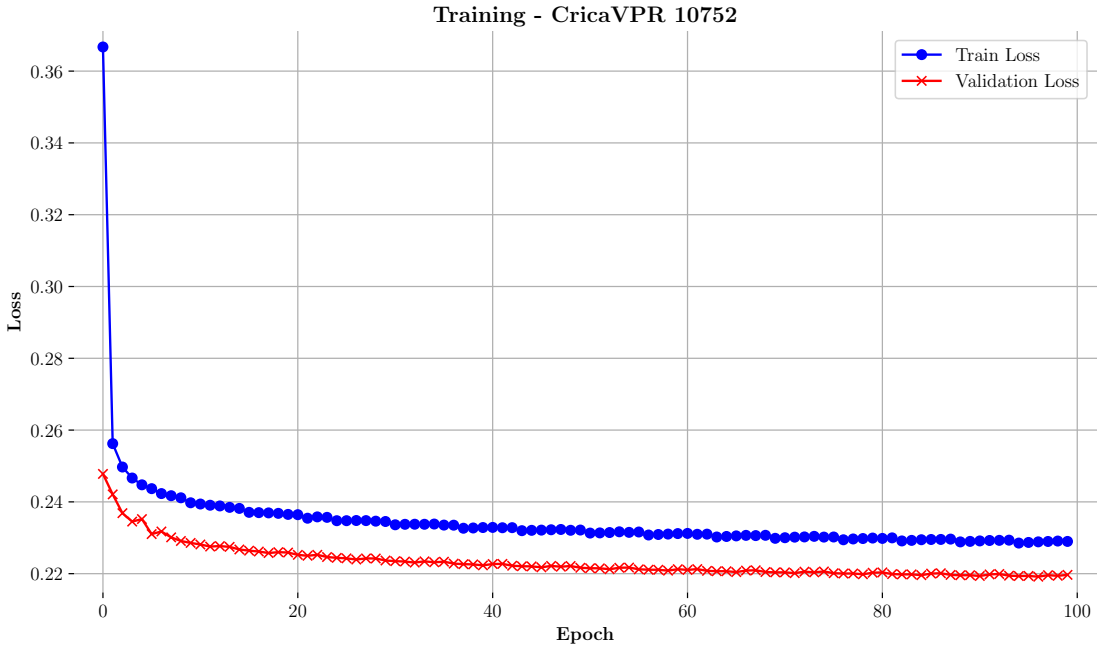


Figure C.9: Training plot of the Latent Diffusion Model conditioned on the output space of CricaVPR [17].

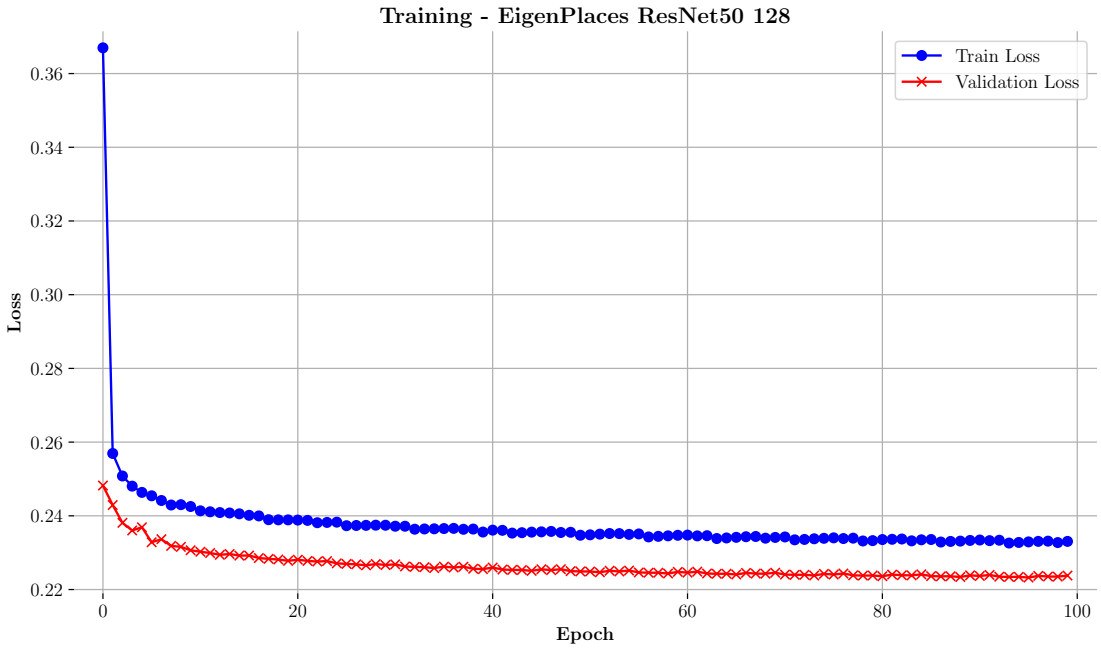


Figure C.10: Training plot of the Latent Diffusion Model conditioned on the output space of EigenPlaces [19] with $d = 128$.

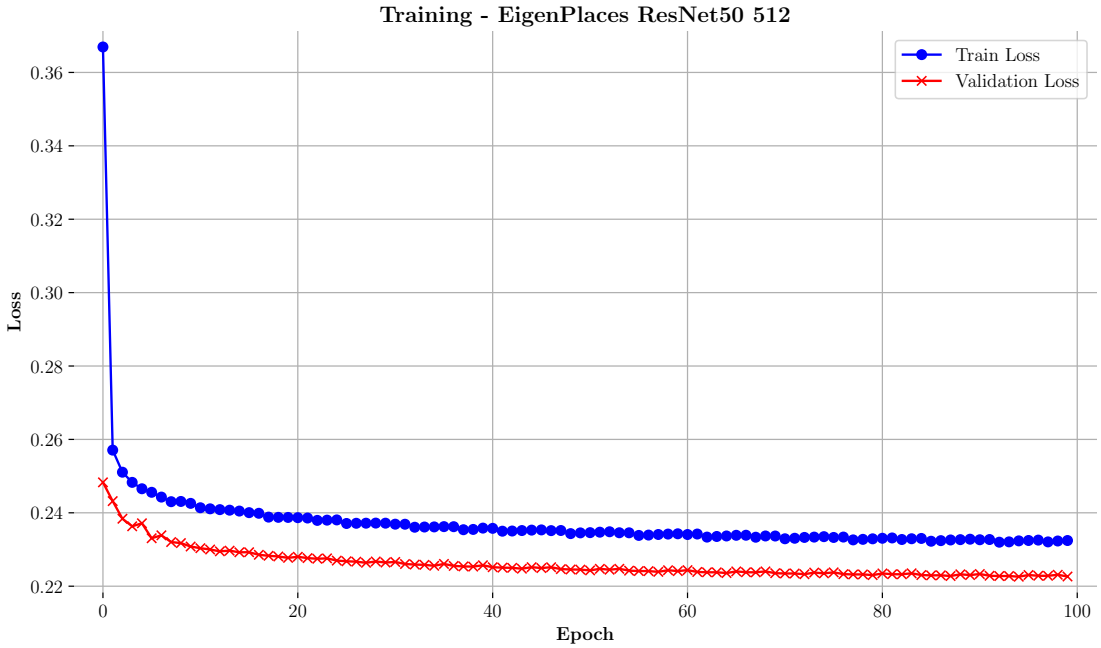


Figure C.11: Training plot of the Latent Diffusion Model conditioned on the output space of EigenPlaces [19] with $d = 512$.

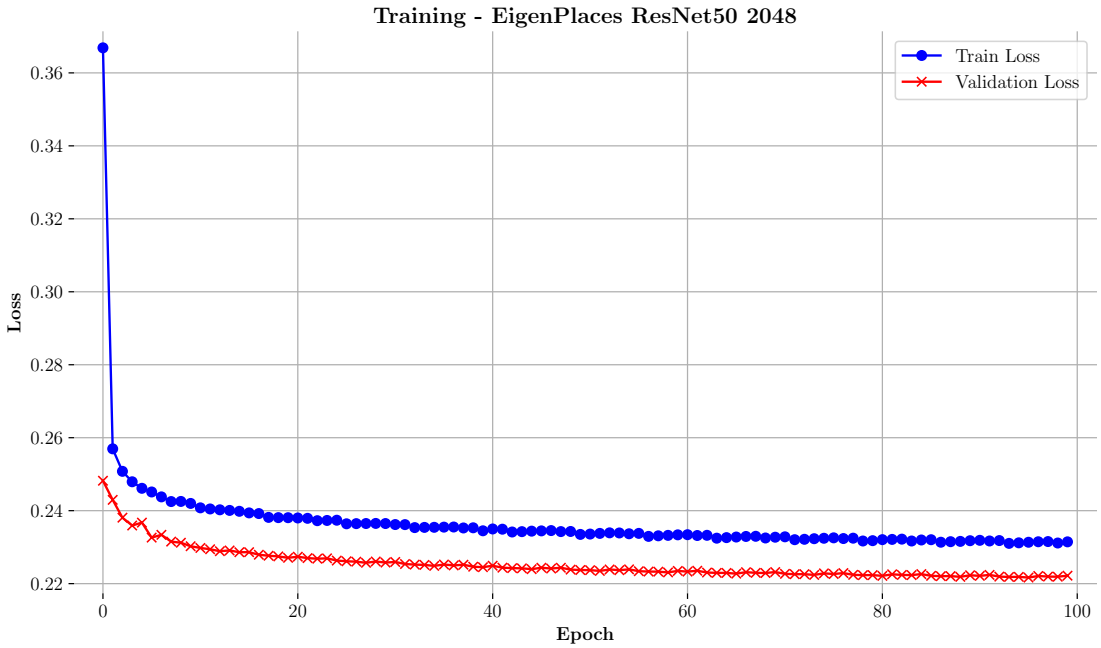


Figure C.12: Training plot of the Latent Diffusion Model conditioned on the output space of EigenPlaces [19] with $d = 2048$.

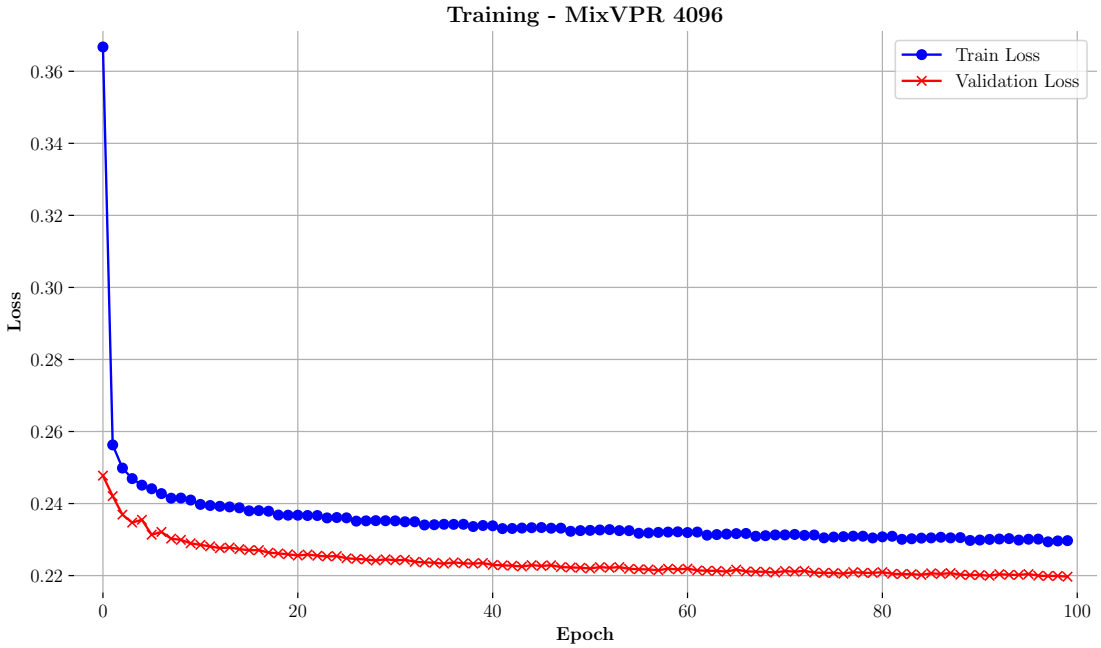


Figure C.13: Training plot of the Latent Diffusion Model conditioned on the output space of MixVPR [18].

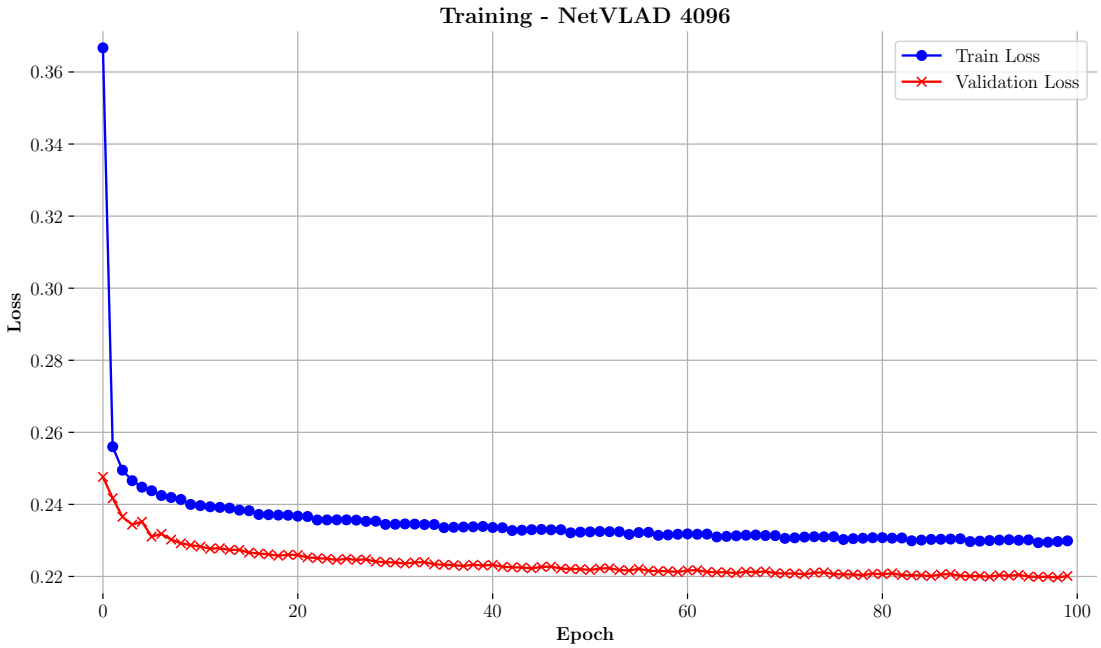


Figure C.14: Training plot of the Latent Diffusion Model conditioned on the output space of NetVLAD [15].

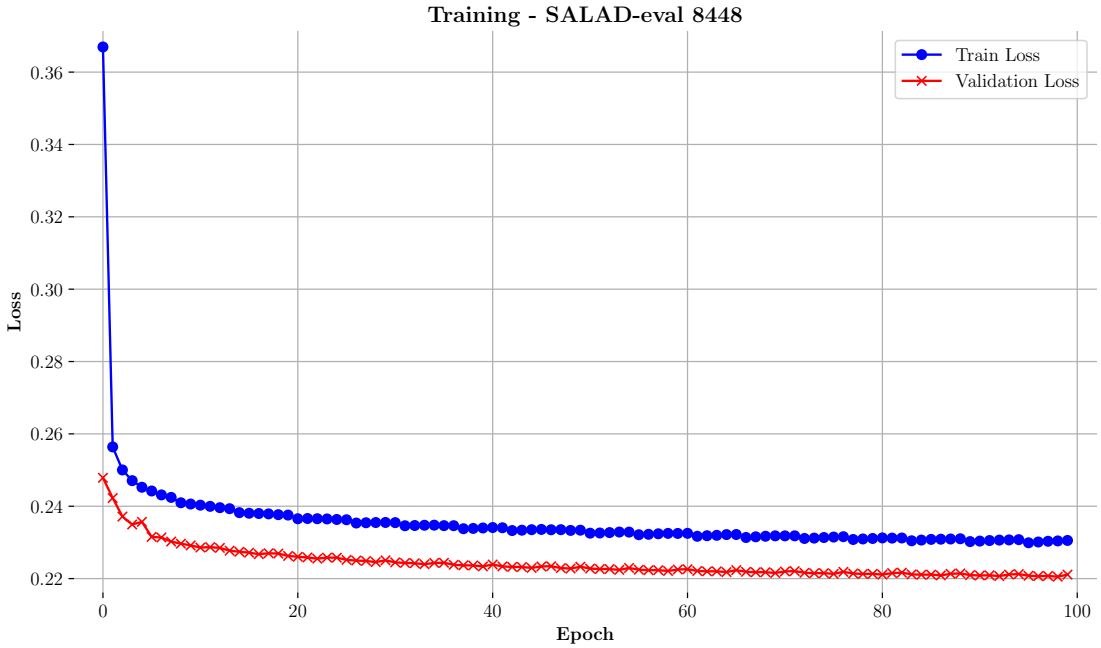


Figure C.15: Training plot of the Latent Diffusion Model conditioned on the output space of SALAD [51].

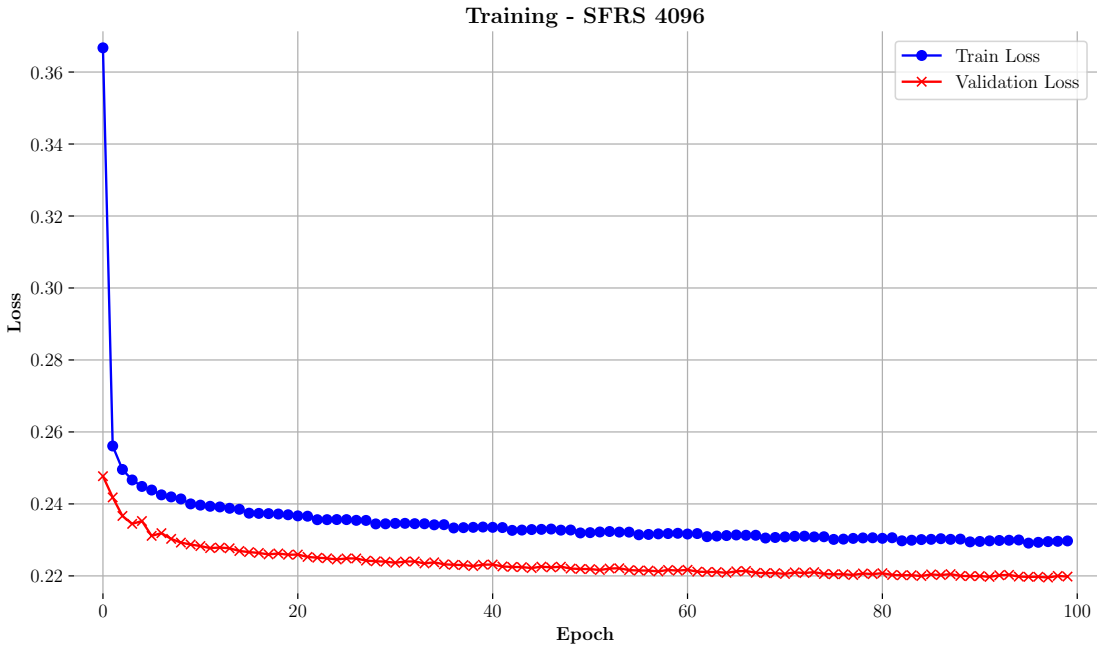


Figure C.16: Training plot of the Latent Diffusion Model conditioned on the output space of SFRS [71].

C.2 Quantitative LDM results

In this section, we further analyze the results from Section 6.1.4 for the additional Visual Place Recognition models listed in Table 6.1. For each model, we provide a table of metrics calculated using $50k$ images. Here, we focus specifically on the scale parameter $s \in \{1, 2\}$, which regulates the strength of the CFG [42, 39].

For comparison, Table C.1 reports the L_1 and L_2 metric values for all VPR models. These values are derived from randomly selecting $14k$ images from the validation set of Table 6.2 and computing the mean of all possible pairwise distances, totaling $\binom{14{,}000}{2} = 97{,}993k$ pairwise comparisons. The selection of $14k$ images is based on the observation that these metrics stabilize and do not change significantly with larger sample sizes.

Method	L_1	L_2
AP-GeM [14]	35.84	1.07
CliqueMining [13]	102.48	1.40
Conv-AP [25]	69.96	1.38
CosPlace ($d = 32$) [52]	6.35	1.39
CosPlace ($d = 64$) [52]	8.99	1.40
CosPlace ($d = 128$) [52]	12.70	1.40
CosPlace ($d = 512$) [52]	25.34	1.40
CosPlace ($d = 2048$) [52]	50.70	1.40
CricaVPR [17]	113.22	1.38
EigenPlaces ($d = 128$) [19]	12.65	1.40
EigenPlaces ($d = 512$) [19]	25.27	1.40
EigenPlaces ($d = 2048$) [19]	50.50	1.40
MixVPR [18]	70.14	1.39
NetVLAD [15]	70.55	1.39
SALAD [51]	101.58	1.39
SFRS [71]	70.51	1.39

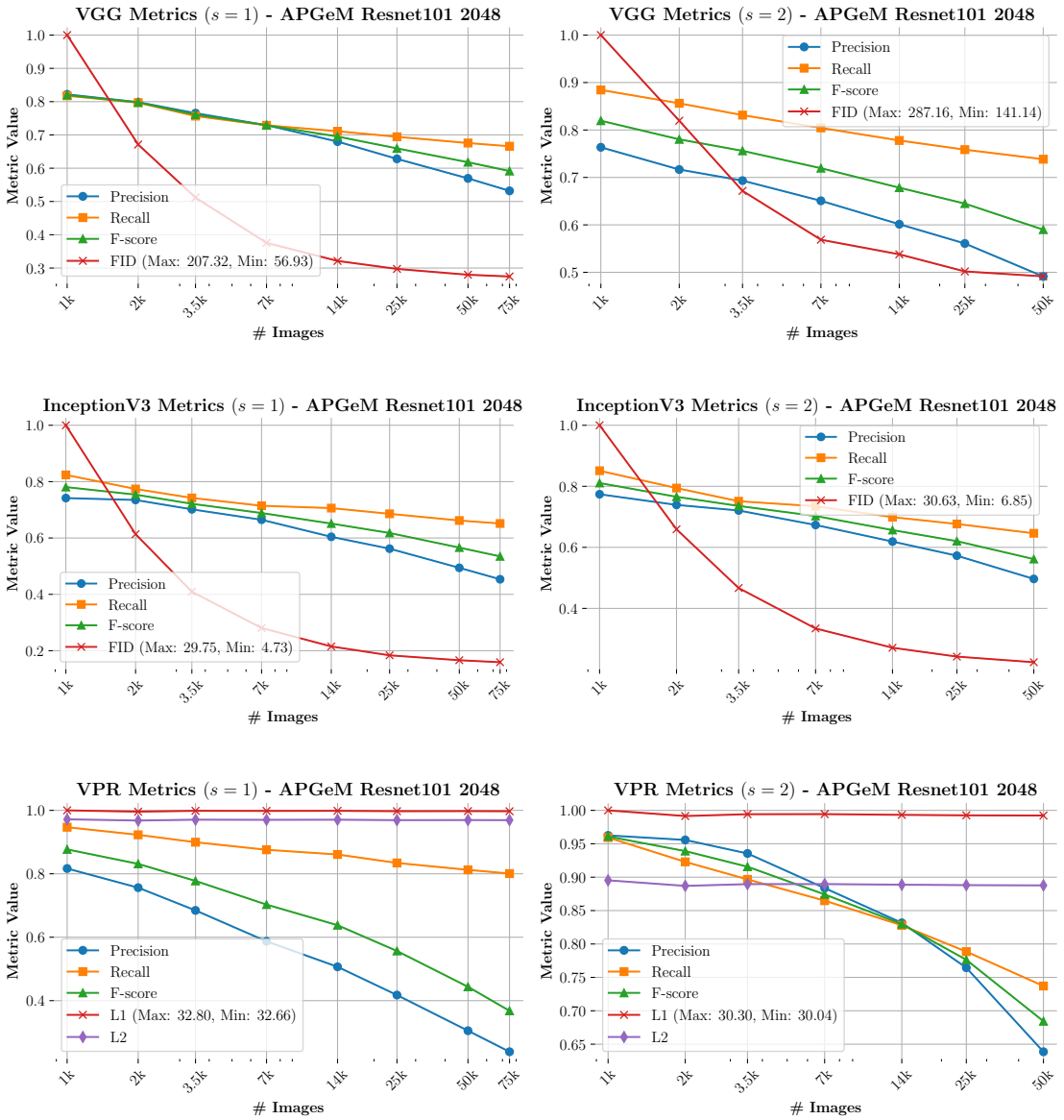
Table C.1: Reference table showing the mean pairwise distances for all Visual Place Recognition models, calculated from $14k$ randomly selected images from the validation set of Table 6.2. The embedding dimension d is included to distinguish between different versions of the same VPR model.

As illustrated in the plots[1] and numerical values in the tables of Fig. C.17–C.31, all models show similar trends. With $s = 1$, we achieve the best metrics for the visual quality of the generated images, as measured by both VGG-16 and InceptionV3. However, this comes at the cost of reduced fidelity to the embeddings produced by the Visual Place Recognition models, as reflected in the Precision and

[1]To improve readability and better fit the page, the x-axis is presented on a logarithmic scale.

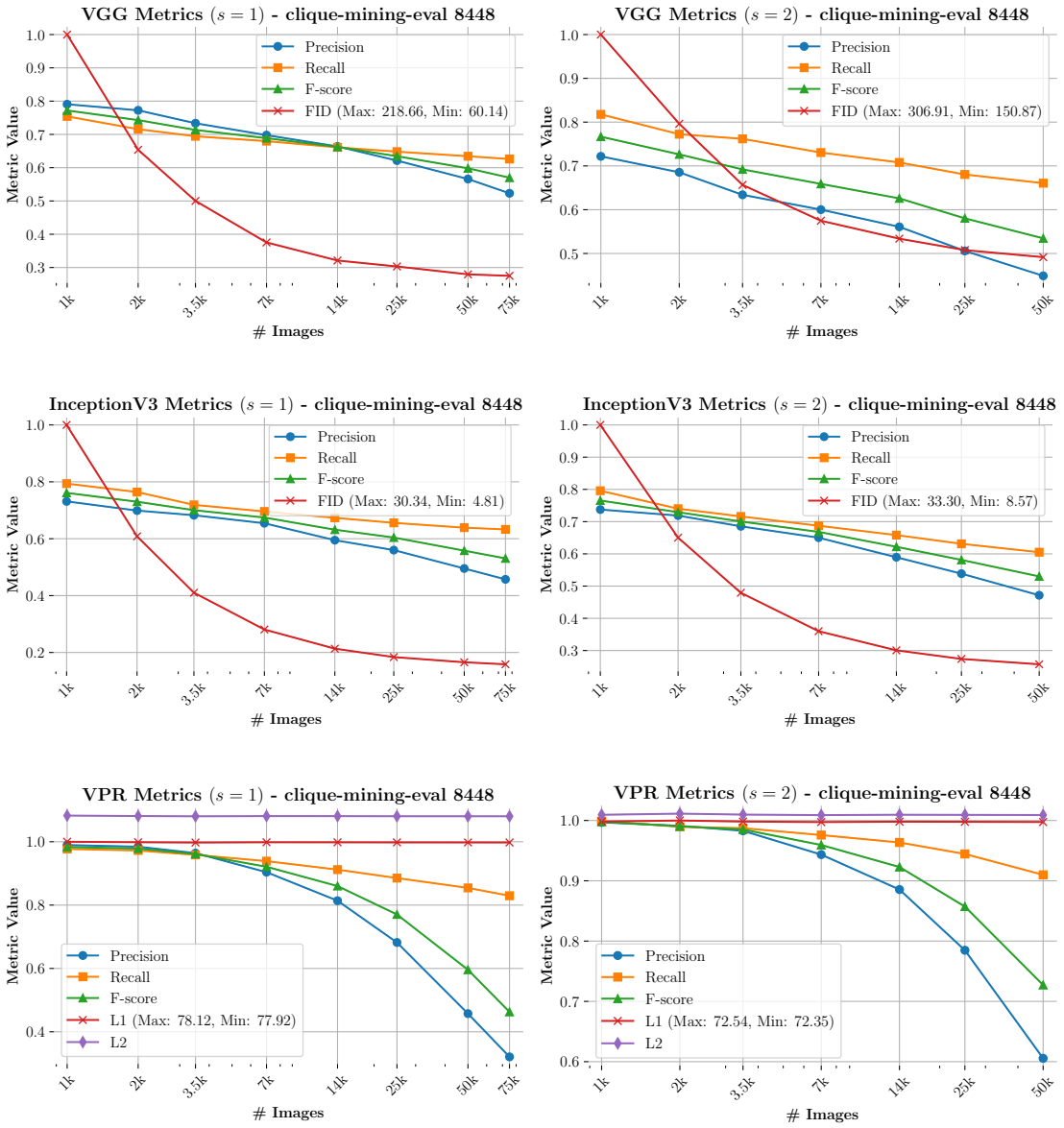
Recall values under each model's name. When $s = 2$, the generation process aligns more strongly with the conditioning information, leading to a significant increase in Precision within the output space of the VPR models. While Recall either remains stable or decreases slightly (with a maximum drop of 16% for NetVLAD), except for CliqueMining and SALAD, which both show a 6% improvement, this trend is offset by a notable increase in Precision. For NetVLAD, this results in a 28% increase in Precision, which is the second-largest gain across all models. The largest gain is seen for AP-GeM, with an impressive 33% improvement in Precision.

In summary, for all models, when $s = 2$, the embeddings $f(\hat{\mathbf{x}})$ of the generated images are closer to the embeddings $f(\mathbf{x})$ of real images, as indicated by the L_1 and L_2 metrics. This confirms the conclusions drawn in Section 6.1.4 regarding CosPlace [52] with embedding dimension $d = 2048$. Furthermore, in Fig. C.32–C.35, we present the distributions of the distances used to calculate the values in Table C.1, along with the corresponding L_2 metrics of the Latent Diffusion Models.



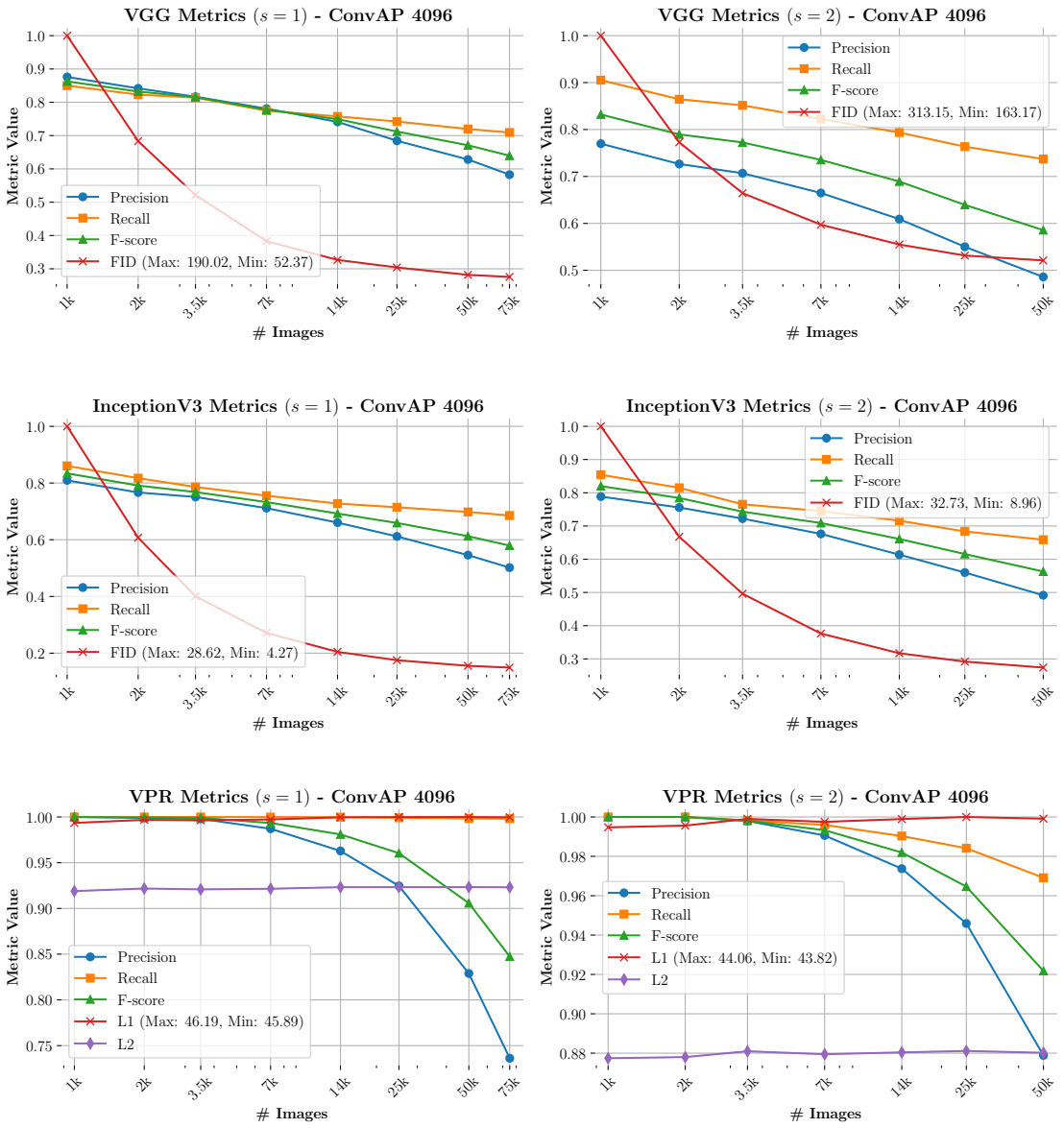
$50k$	VGG-16			InceptionV3			AP-GeM			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**58.02**	**0.57**	0.68	**4.94**	0.49	**0.66**	0.31	**0.81**	32.72	0.97
2	141.14	0.49	**0.74**	6.85	**0.50**	0.65	**0.64**	0.74	**30.07**	**0.89**

Figure C.17: Plots of LDM conditioned on AP-GeM's output space [14]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



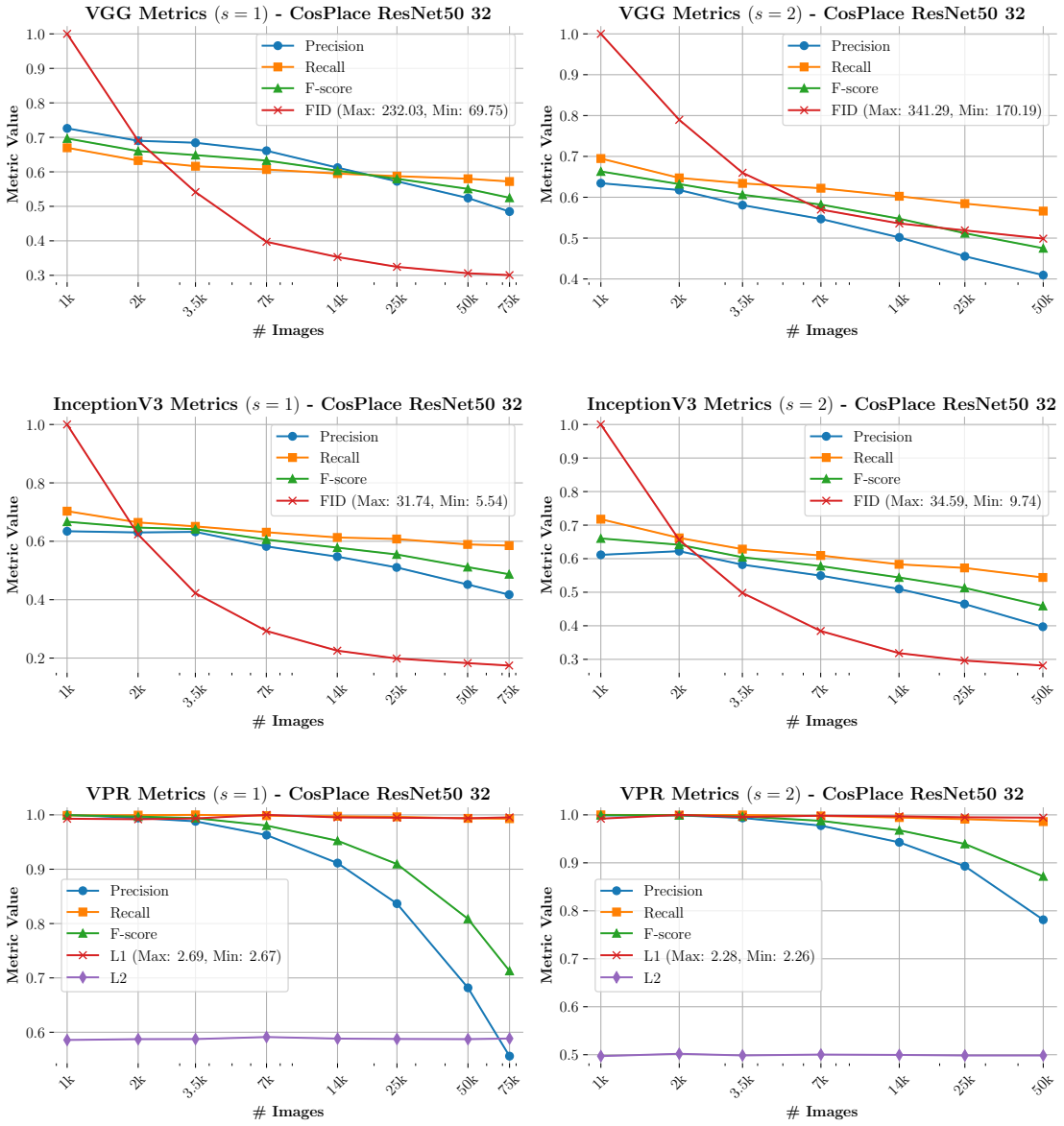
50k	VGG-16			InceptionV3			CliqueMining			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**61.12**	**0.57**	0.63	**5.03**	**0.50**	**0.64**	0.46	0.85	77.95	1.08
2	150.87	0.45	**0.66**	8.57	0.47	0.61	**0.61**	**0.91**	**72.36**	**1.01**

Figure C.18: Plots of LDM conditioned on CliqueMining's output space [13]. The table shows metrics for 50k images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



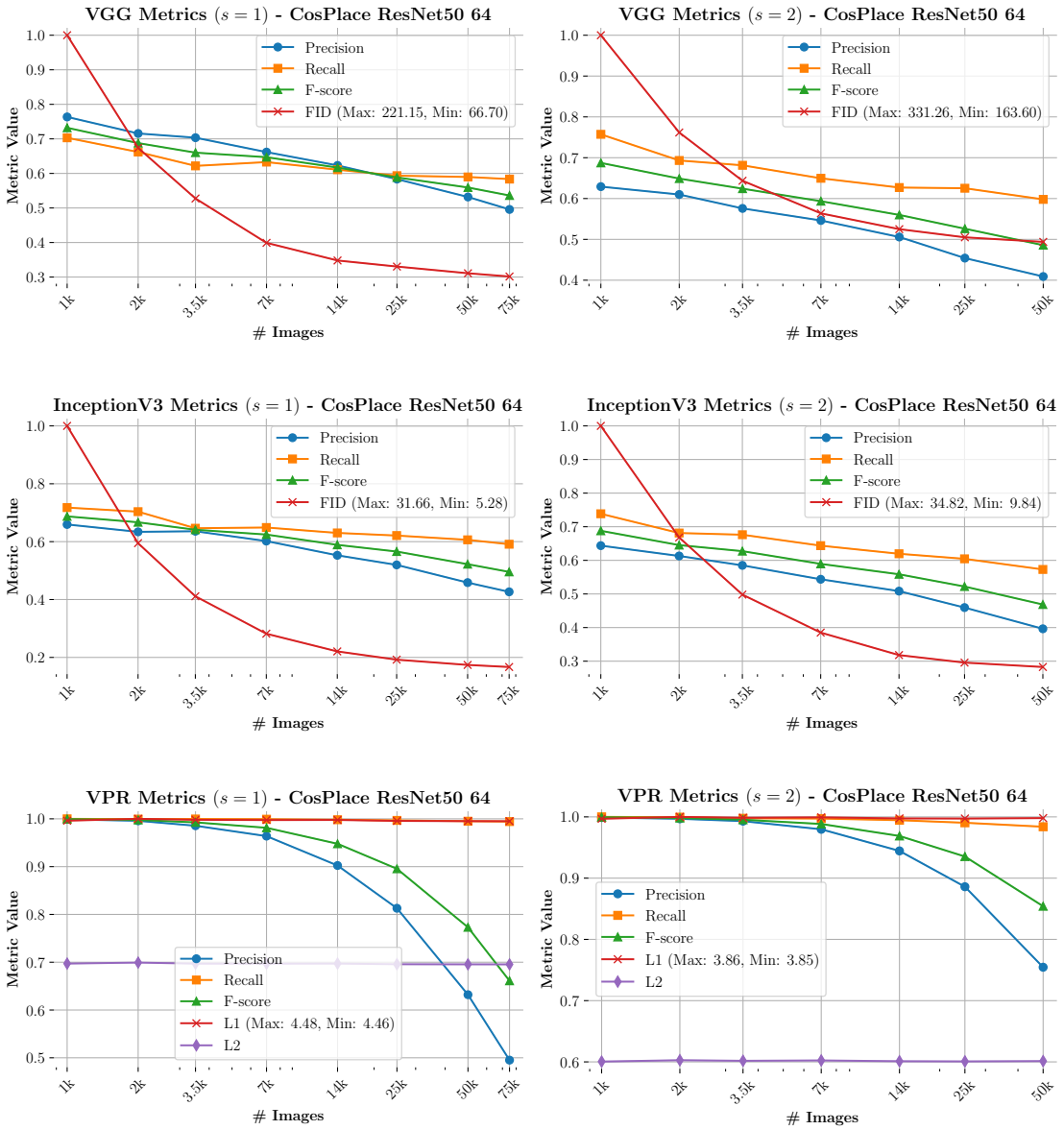
$50k$	VGG-16			InceptionV3			Conv-AP			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**53.54**	**0.63**	0.72	**4.45**	**0.55**	**0.70**	0.83	**0.99**	46.19	0.92
2	163.17	0.49	**0.74**	8.96	0.49	0.66	**0.88**	0.97	**44.02**	**0.88**

Figure C.19: Plots of LDM conditioned on Conv-AP's output space [25]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



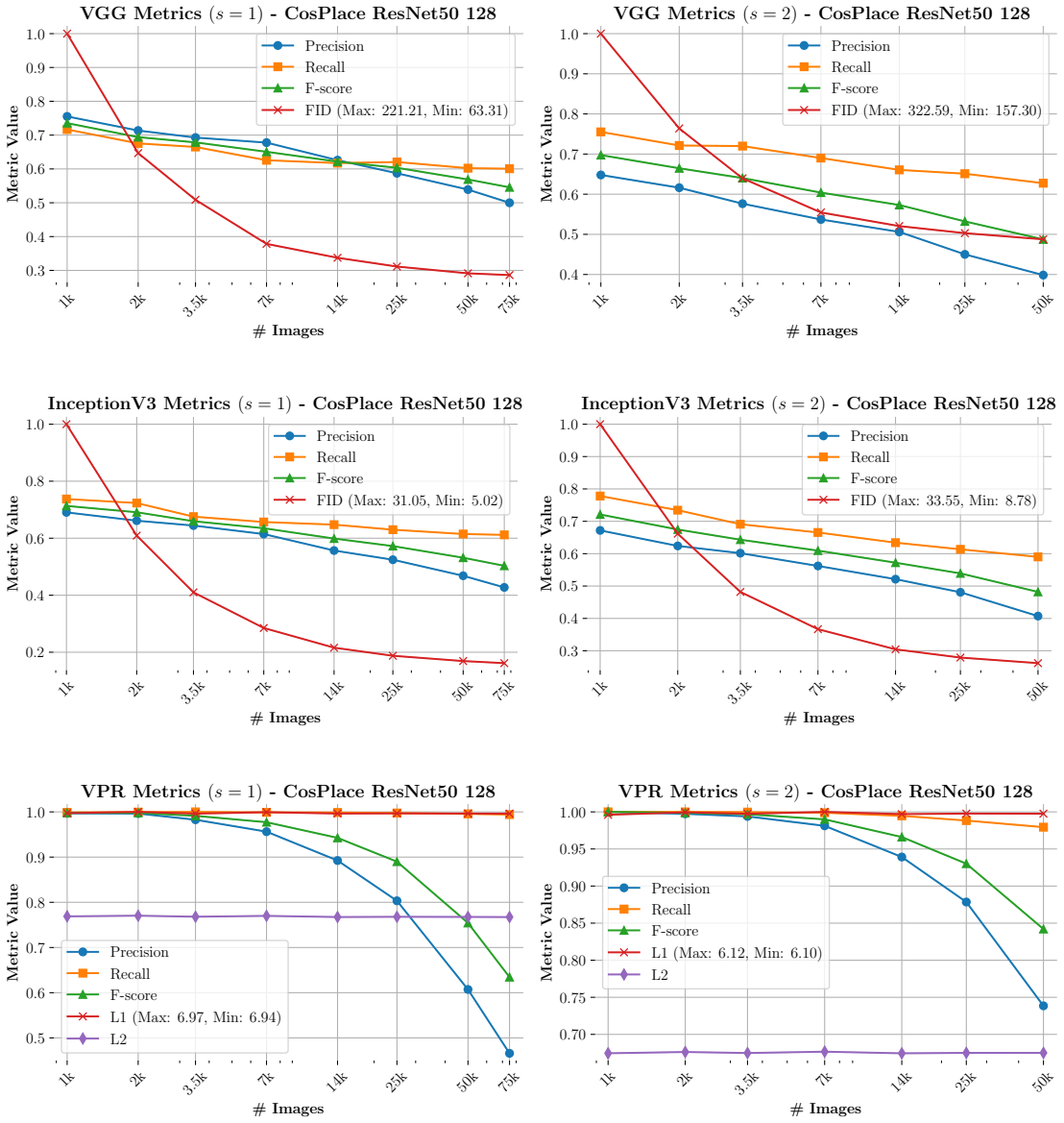
$50k$	VGG-16			InceptionV3			CosPlace ($d = 32$)			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**71.01**	**0.52**	**0.58**	**5.80**	**0.45**	**0.59**	0.68	**0.99**	2.67	0.59
2	170.19	0.41	0.57	9.74	0.40	0.54	**0.78**	**0.99**	**2.27**	**0.50**

Figure C.20: Plots of LDM conditioned on CosPlace's output space [52] ($d = 32$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



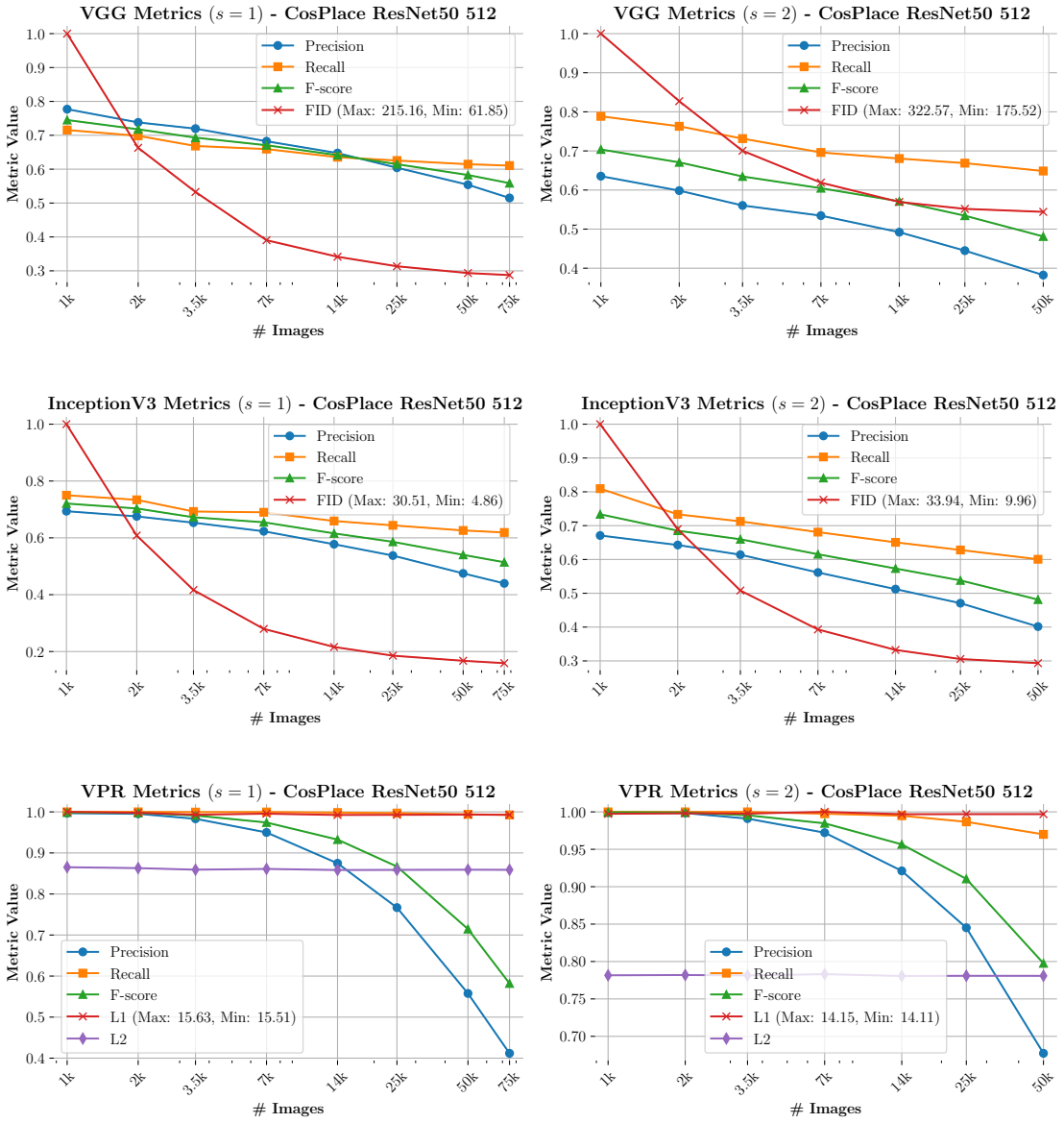
$50k$	VGG-16			InceptionV3			CosPlace ($d = 64$)			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**68.78**	**0.53**	0.59	**5.51**	**0.46**	**0.61**	0.63	**0.99**	4.46	0.70
2	163.60	0.41	**0.60**	9.84	0.40	0.57	**0.75**	0.98	**3.85**	**0.60**

Figure C.21: Plots of LDM conditioned on CosPlace's output space [52] ($d = 64$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



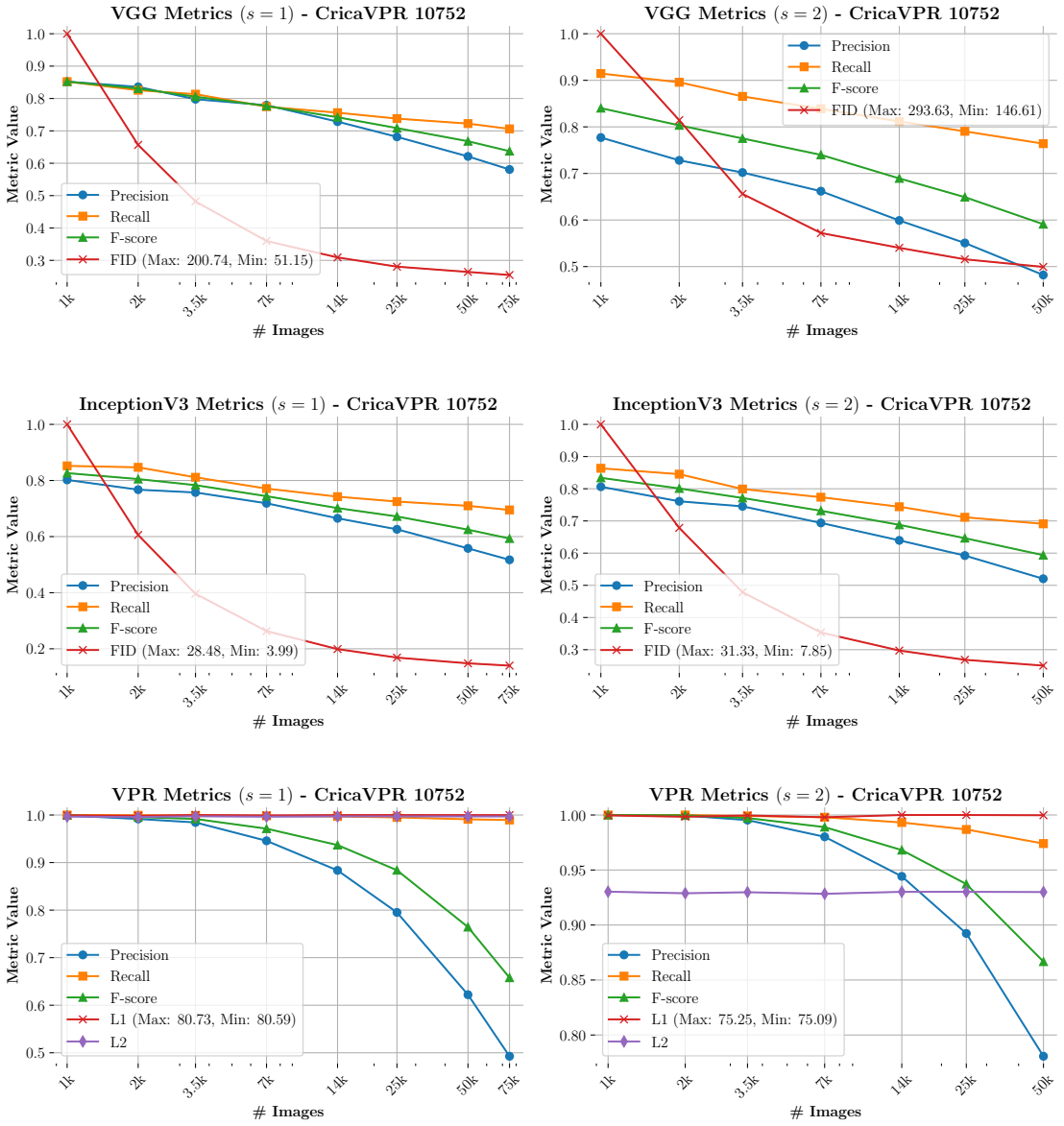
$50k$	VGG-16			InceptionV3			CosPlace ($d = 128$)			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**64.45**	**0.54**	0.60	**5.25**	**0.47**	**0.61**	0.61	**0.99**	6.94	0.77
2	157.30	0.40	**0.63**	8.78	0.41	0.59	**0.74**	0.98	**6.11**	**0.68**

Figure C.22: Plots of LDM conditioned on CosPlace's output space [52] ($d = 128$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



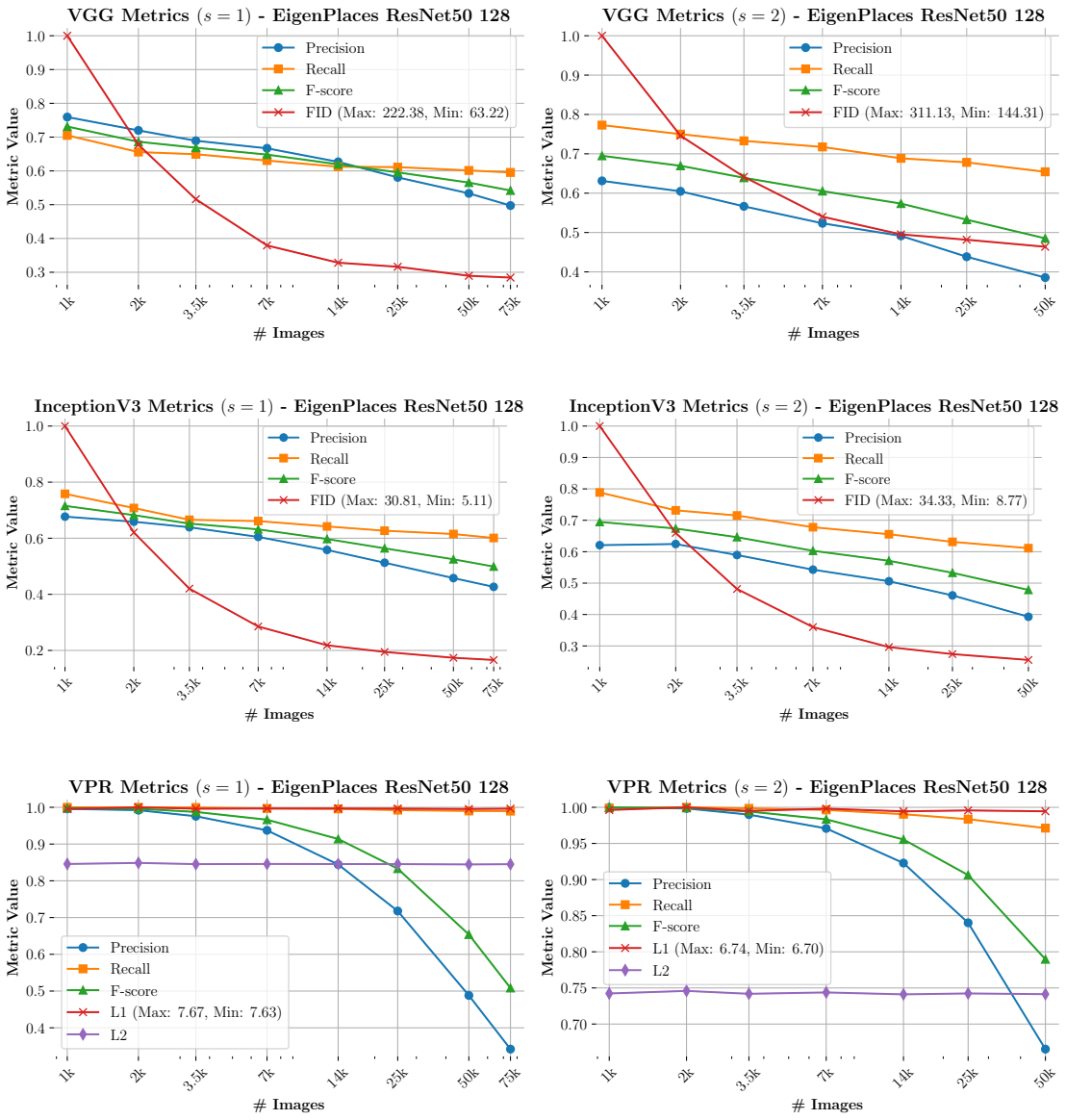
$50k$	VGG-16			InceptionV3			CosPlace ($d = 512$)			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**63.12**	**0.55**	0.61	**5.11**	**0.47**	**0.63**	0.56	**0.99**	15.53	0.86
2	175.52	0.38	**0.65**	9.96	0.40	0.60	**0.68**	0.97	**14.11**	**0.78**

Figure C.23: Plots of LDM conditioned on CosPlace's output space [52] ($d = 512$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



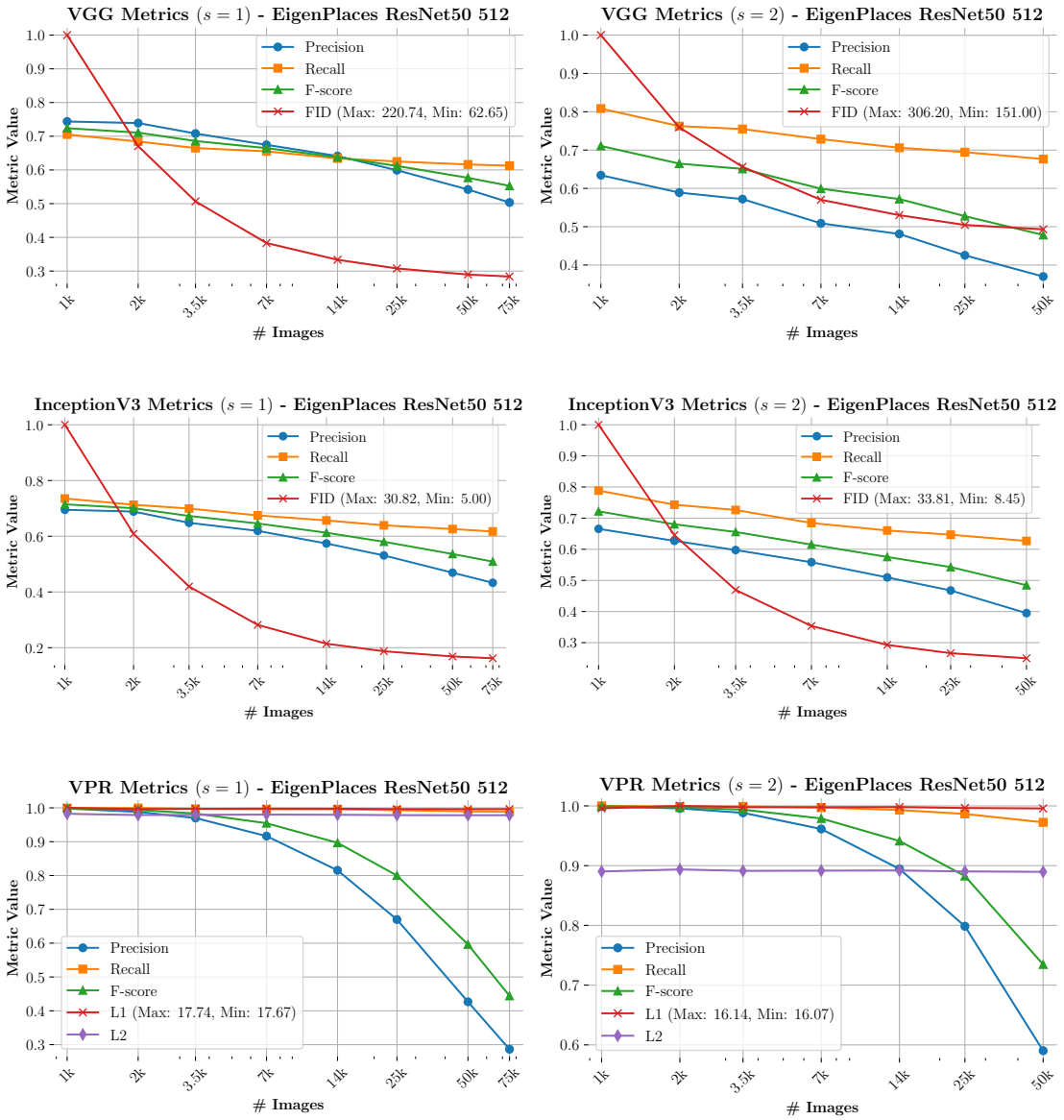
$50k$	VGG-16			InceptionV3			CricaVPR			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**53.05**	**0.62**	0.72	**4.22**	**0.56**	**0.71**	0.62	**0.99**	80.71	0.99
2	146.61	0.48	**0.76**	7.85	0.52	0.69	**0.78**	0.97	**75.23**	**0.93**

Figure C.24: Plots of LDM conditioned on CricaVPR's output space [17]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



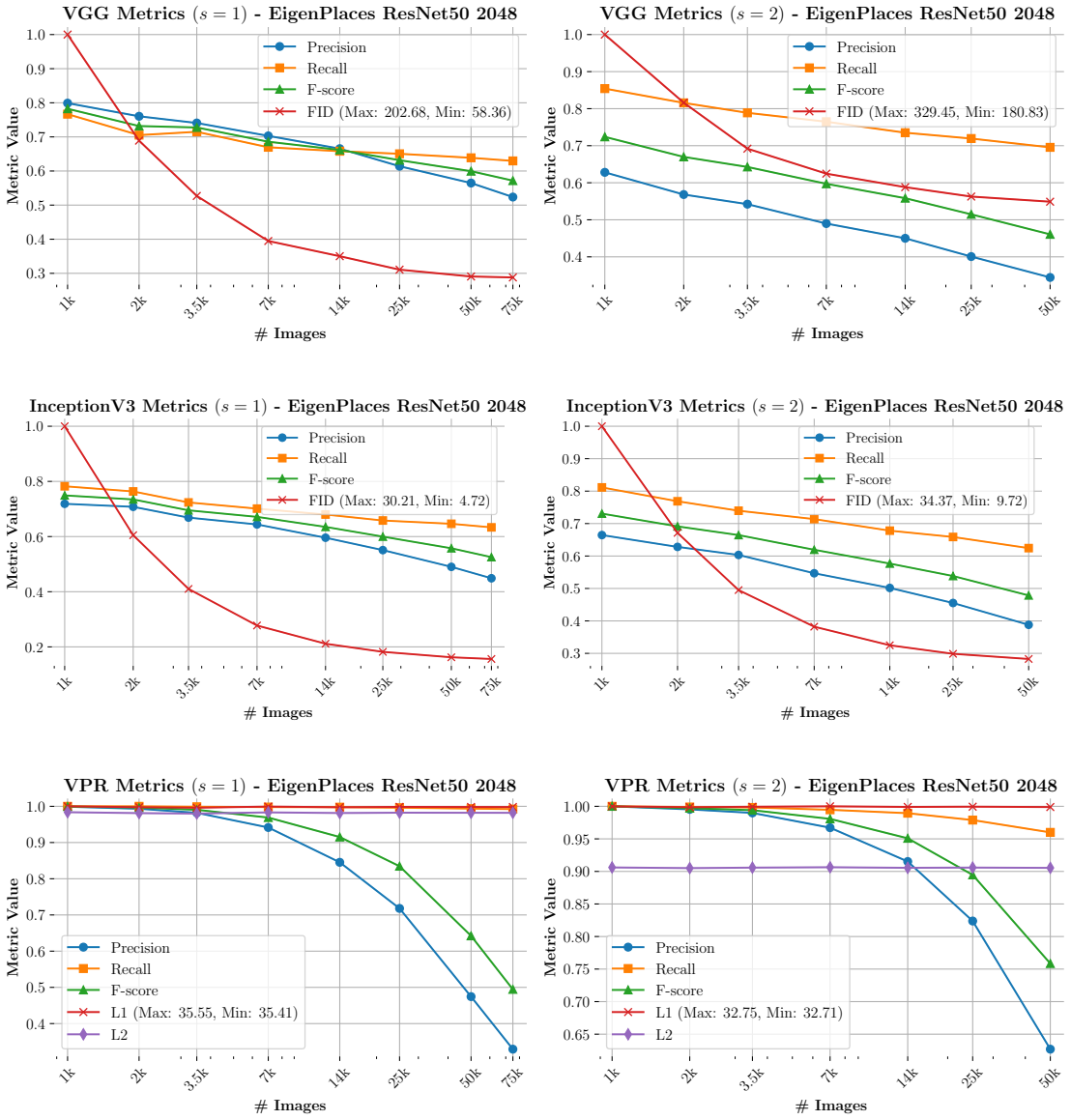
$50k$	VGG-16			InceptionV3			EigenPlaces ($d = 128$)			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**64.39**	**0.53**	0.60	**5.35**	**0.46**	**0.62**	0.49	**0.99**	7.63	0.84
2	144.31	0.39	**0.65**	8.77	0.39	0.61	**0.67**	0.97	**6.70**	**0.74**

Figure C.25: Plots of LDM conditioned on EigenPlaces's output space [19] ($d = 128$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



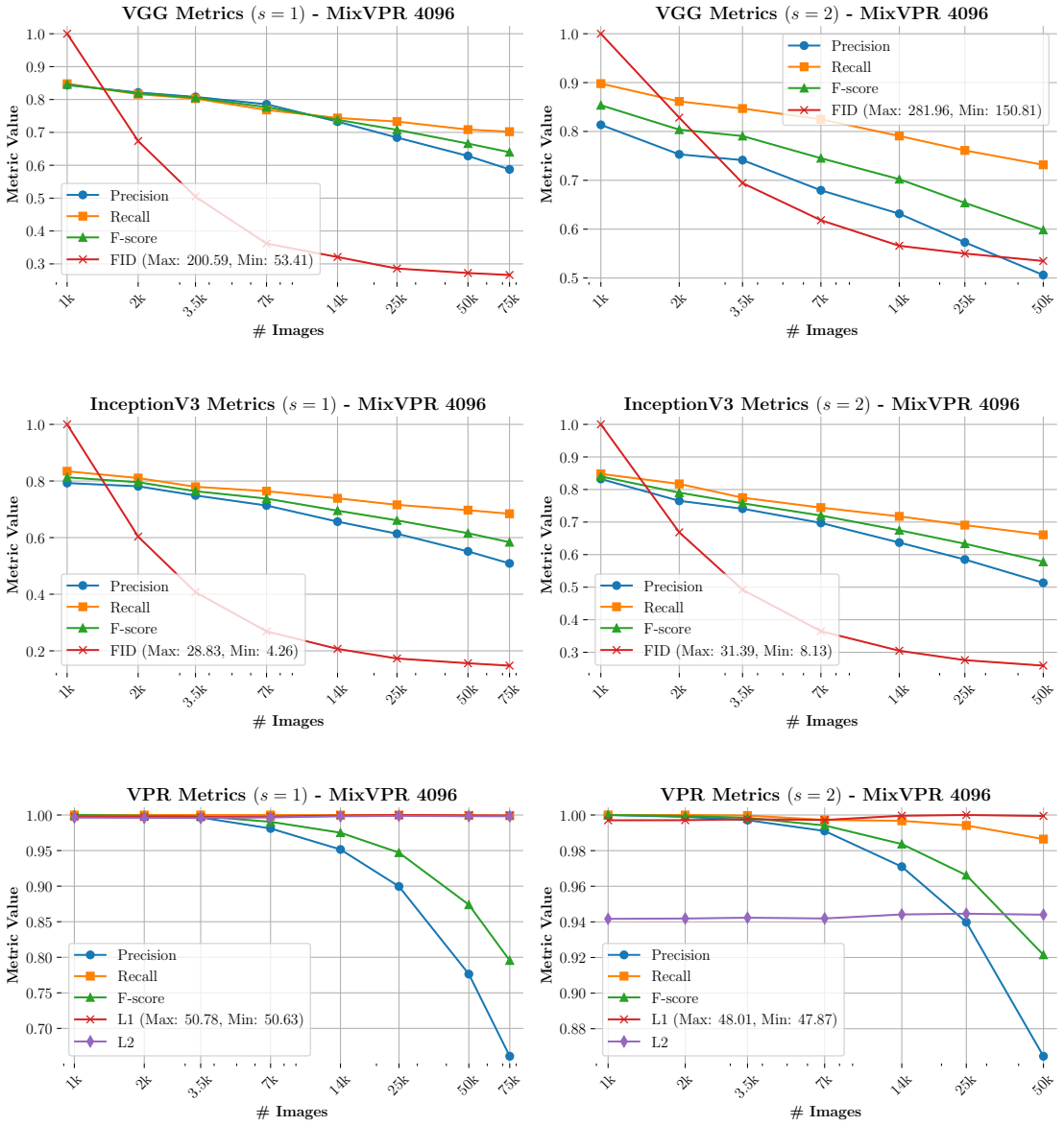
$50k$	VGG-16			InceptionV3			EigenPlaces ($d = 512$)			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**63.94**	**0.54**	0.62	**5.19**	**0.47**	**0.63**	0.43	**0.99**	17.67	0.98
2	151.00	0.37	**0.68**	8.45	0.39	**0.63**	**0.59**	0.97	**16.07**	**0.89**

Figure C.26: Plots of LDM conditioned on EigenPlaces's output space [19] ($d = 512$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



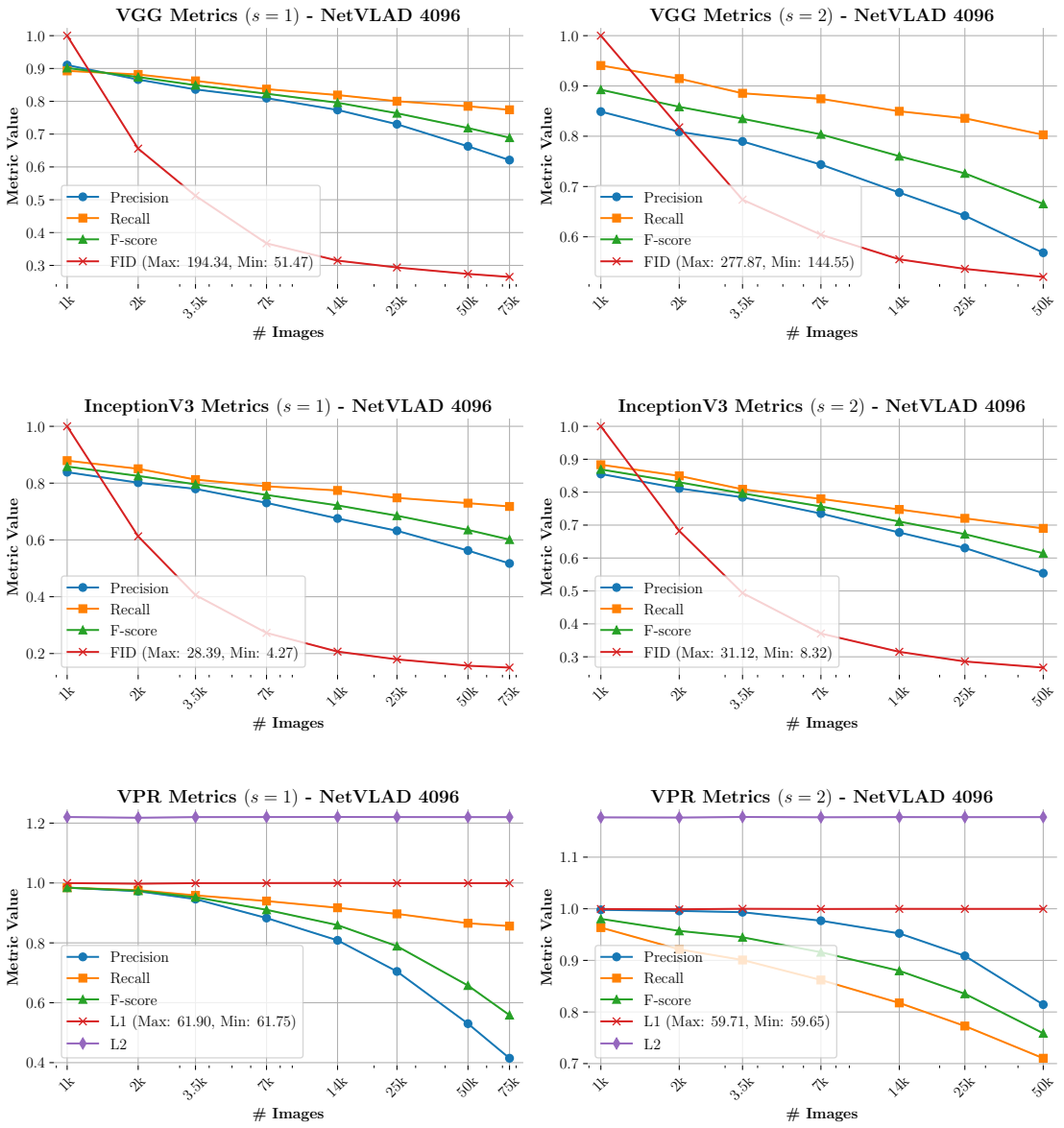
$50k$	VGG-16			InceptionV3			EigenPlaces ($d = 2048$)			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**58.92**	**0.56**	0.64	**4.90**	**0.49**	**0.65**	0.47	**0.99**	35.49	0.98
2	180.83	0.34	**0.70**	9.72	0.39	0.62	**0.63**	0.96	**32.71**	**0.91**

Figure C.27: Plots of LDM conditioned on EigenPlaces's output space [19] ($d = 2048$). The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



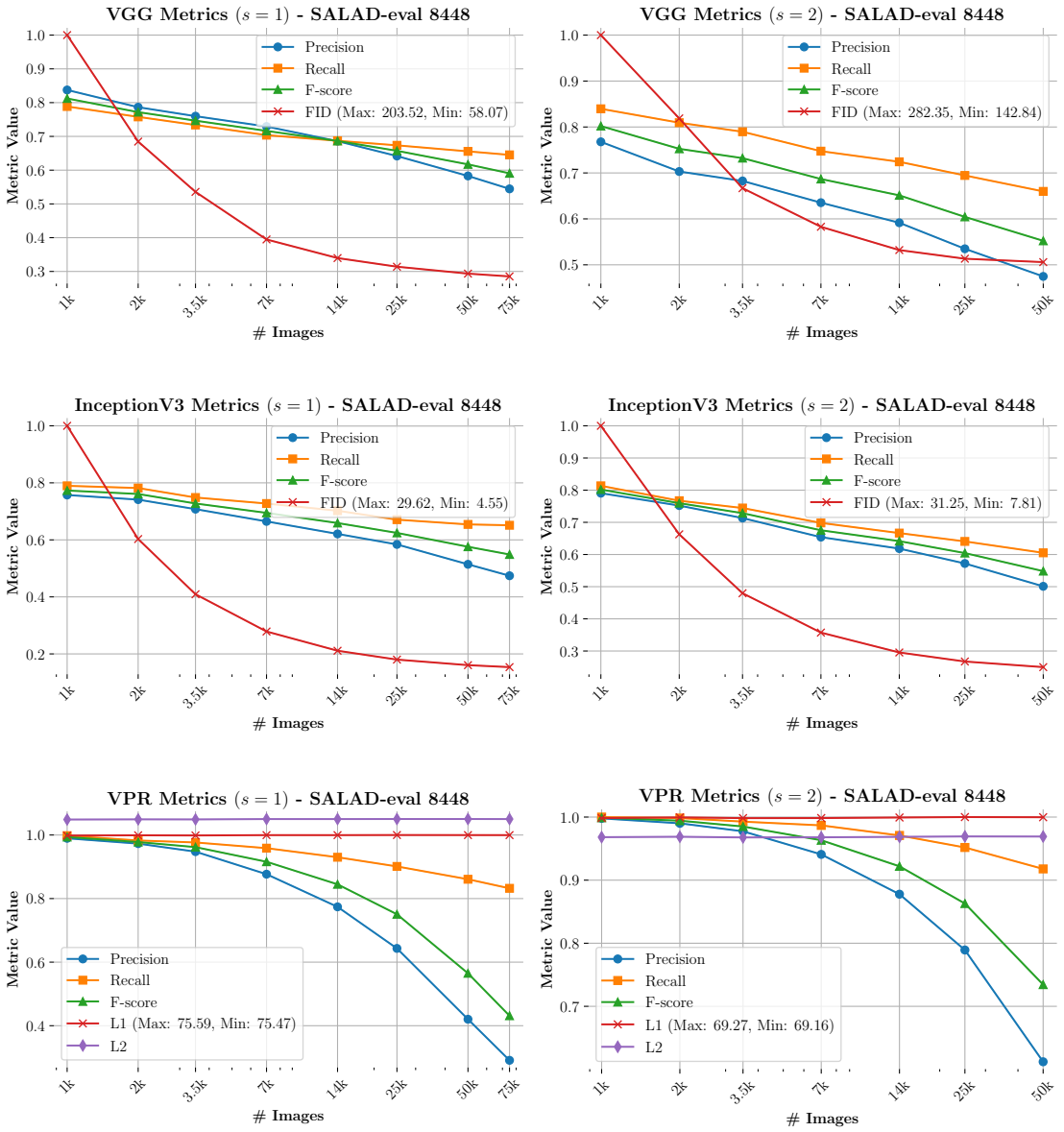
$50k$	VGG-16			InceptionV3			MixVPR			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**54.61**	**0.63**	0.71	**4.50**	**0.55**	**0.70**	0.78	**1.00**	50.75	1.00
2	150.81	0.51	**0.73**	8.13	0.51	0.66	**0.86**	0.99	**47.98**	**0.94**

Figure C.28: Plots of LDM conditioned on MixVPR's output space [18]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



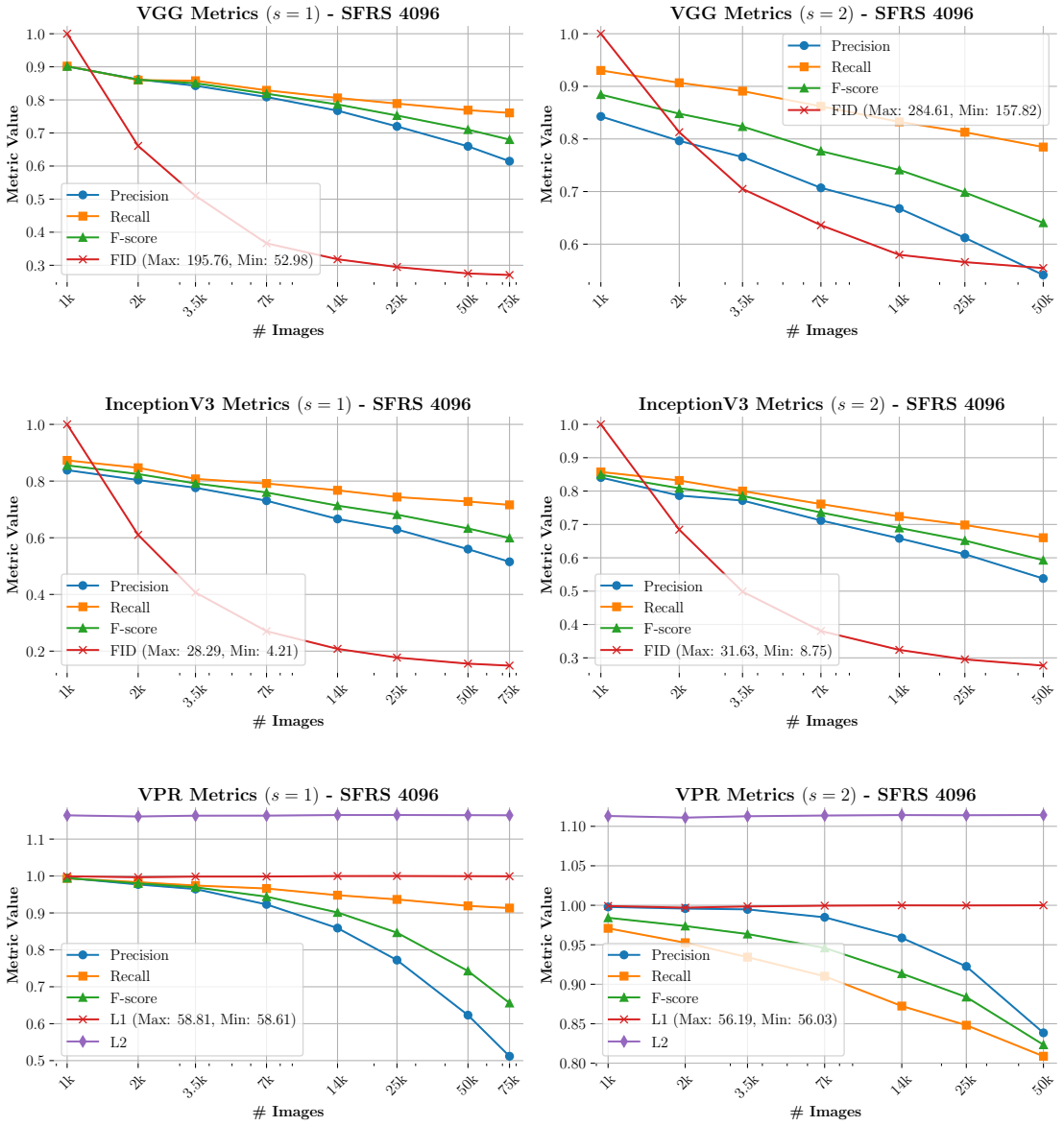
50k	VGG-16			InceptionV3			NetVLAD			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**53.27**	**0.66**	0.78	**4.46**	**0.56**	**0.73**	0.53	**0.87**	61.88	1.22
2	144.55	0.57	**0.80**	8.32	0.55	0.69	**0.81**	0.71	**59.70**	**1.18**

Figure C.29: Plots of LDM conditioned on NetVLAD's output space [15]. The table shows metrics for 50k images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



$50k$	VGG-16			InceptionV3			SALAD			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**59.74**	**0.58**	**0.66**	**4.76**	**0.51**	**0.65**	0.42	0.86	75.58	1.05
2	142.84	0.47	**0.66**	7.81	0.50	0.61	**0.61**	**0.92**	**69.25**	**0.97**

Figure C.30: Plots of LDM conditioned on SALAD's output space [51]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].



$50k$	VGG-16			InceptionV3			SFRS			
s	FID ↓	P ↑	R ↑	FID ↓	P ↑	R ↑	P ↑	R ↑	L_1 ↓	L_2 ↓
1	**53.89**	**0.66**	0.77	**4.41**	**0.56**	**0.73**	0.62	**0.92**	58.79	1.16
2	157.82	0.54	**0.78**	8.75	0.54	0.66	**0.84**	0.81	**56.19**	**1.11**

Figure C.31: Plots of LDM conditioned on SFRS's output space [71]. The table shows metrics for $50k$ images, with the best results in **bold**. Metrics marked with ↑ are better when higher, and those with ↓ are better when lower. 'P' = Precision, 'R' = Recall, and s is the scale parameter for CFG [39].

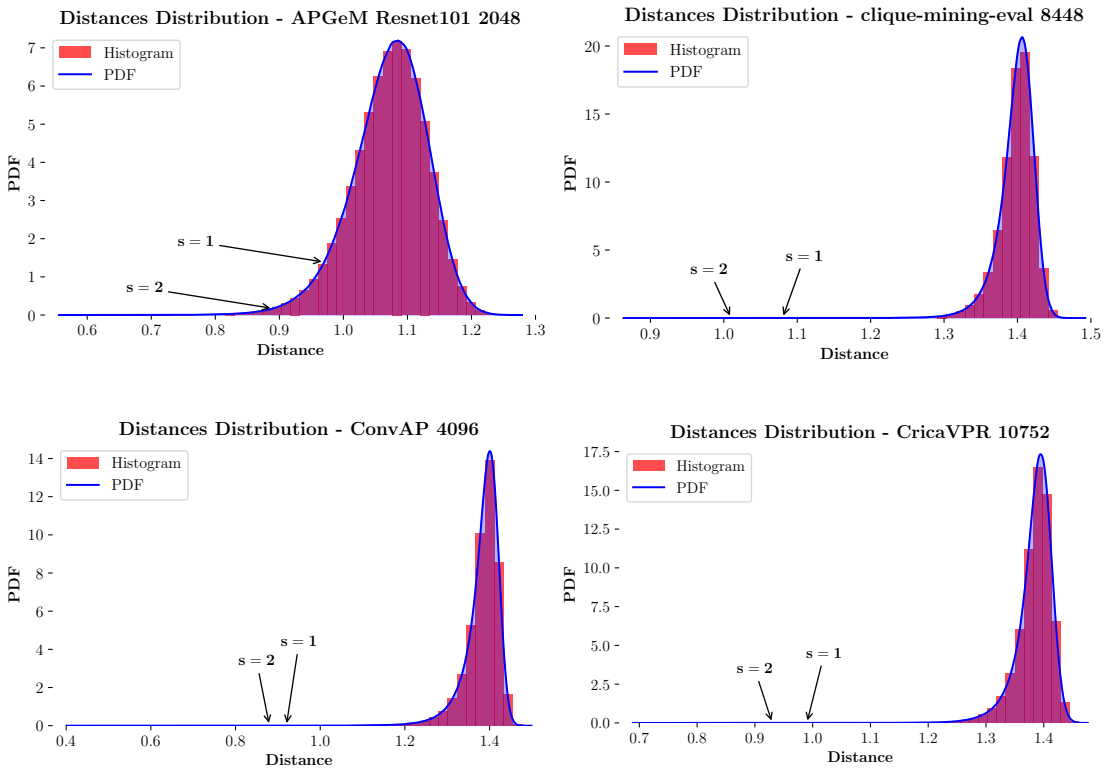


Figure C.32: Distribution plots of the distances for the 14,000 random images used to compute the metrics in Table C.1. The arrows highlight the points in the distribution corresponding to the L_2 distance of the LDM models, based on the scale parameter s for CFG [39]. The distributions for AP-GeM [14], CliqueMining [13], Conv-AP [25], and CricaVPR [17] are shown.

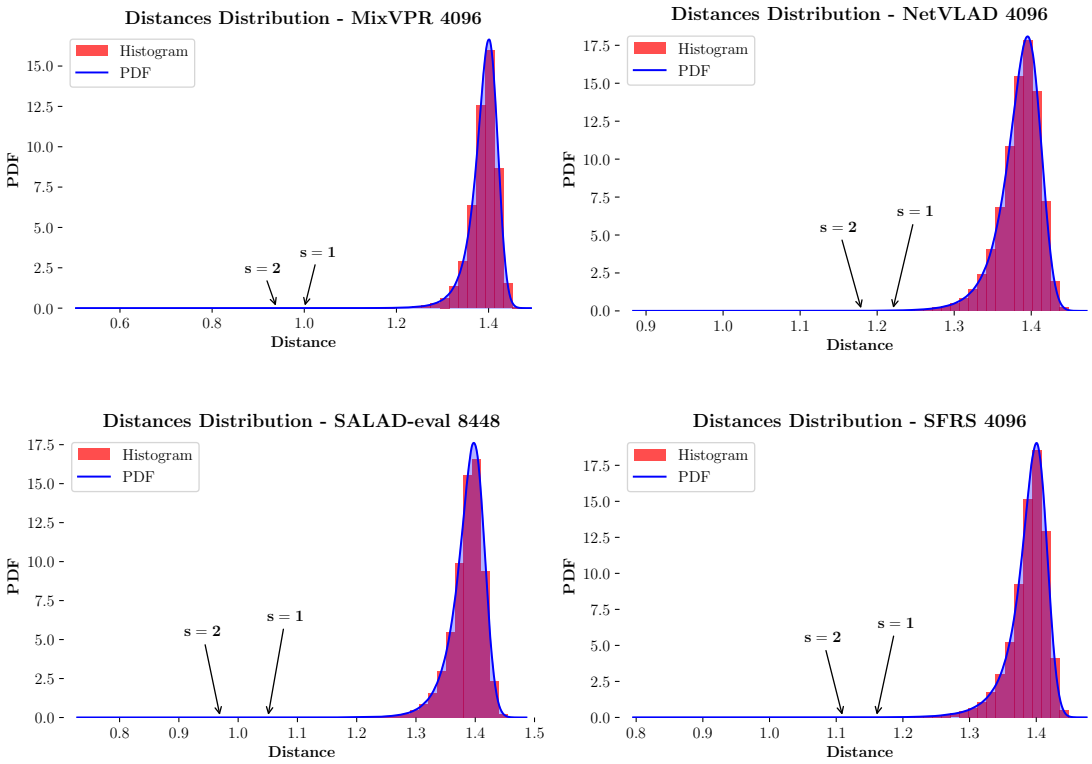


Figure C.33: Distribution plots of the distances for the 14,000 random images used to compute the metrics in Table C.1. The arrows highlight the points in the distribution corresponding to the L_2 distance of the LDM models, based on the scale parameter s for CFG [39]. The distributions for MixVPR [18], NetVLAD [15], SALAD [51], and SFRS [71] are shown.

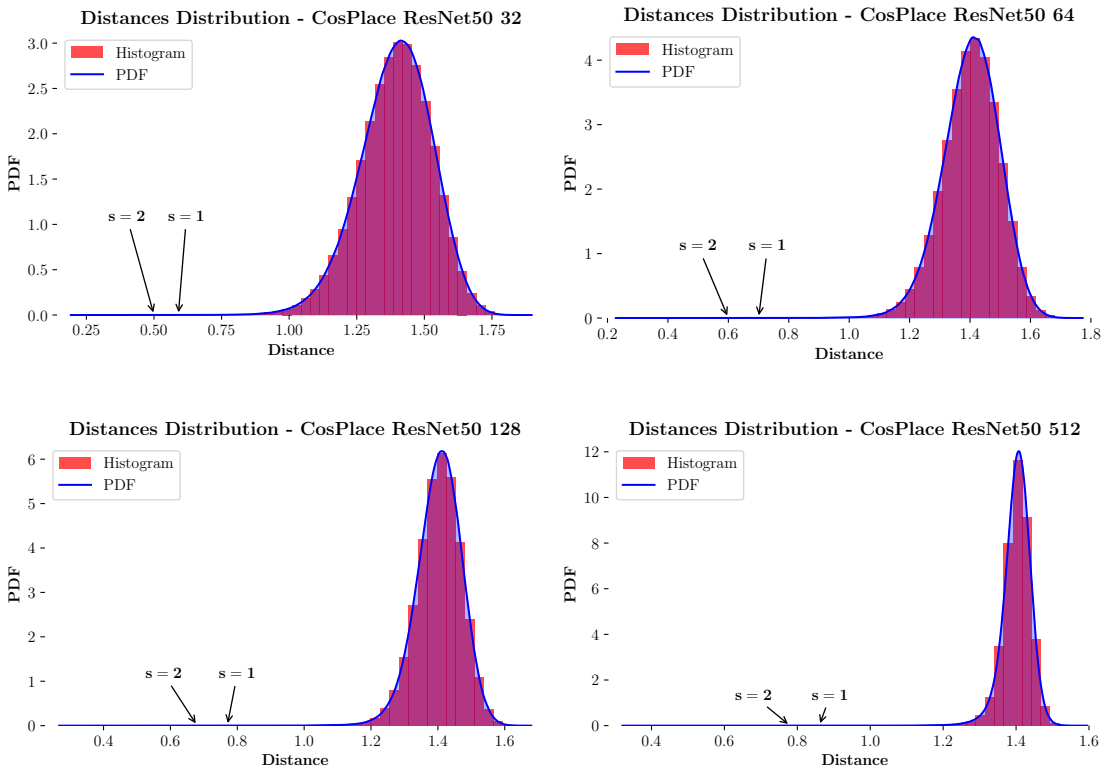


Figure C.34: Distribution plots of the distances for the 14,000 random images used to compute the metrics in Table C.1. The arrows highlight the points in the distribution corresponding to the L_2 distance of the LDM models, based on the scale parameter s for CFG [39]. The distributions for CosPlace [52] with $d \in \{32, 64, 128, 512\}$ are shown.

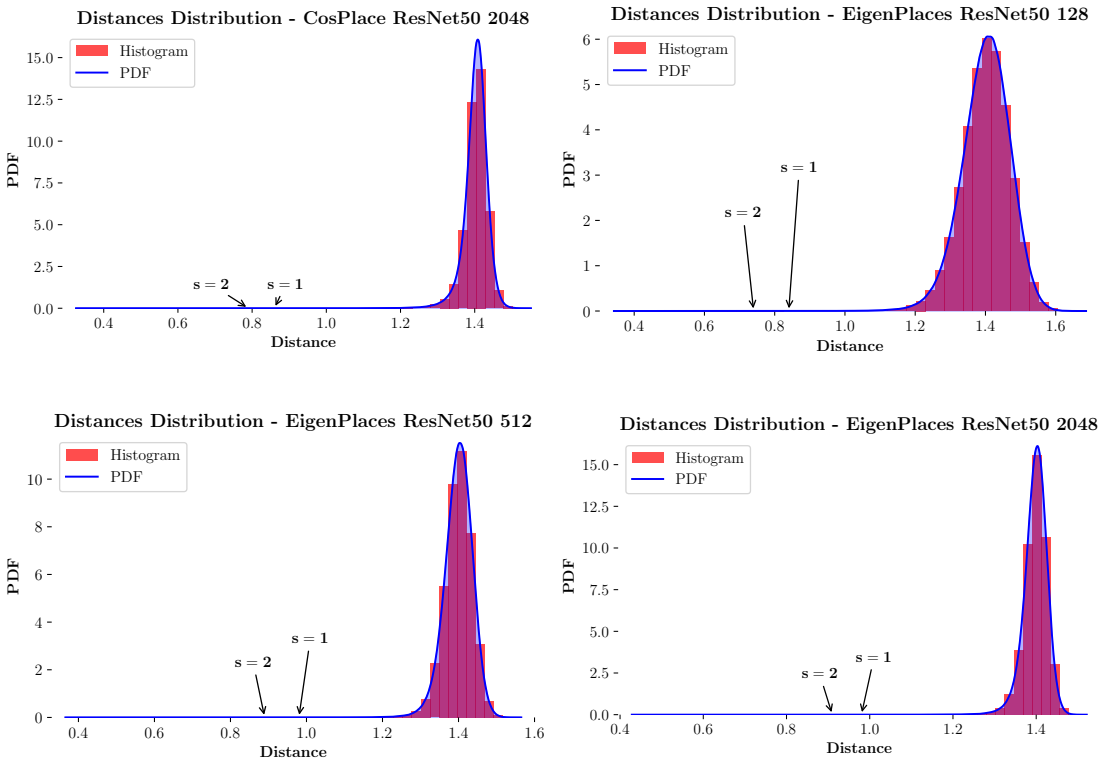


Figure C.35: Distribution plots of the distances for the 14,000 random images used to compute the metrics in Table C.1. The arrows highlight the points in the distribution corresponding to the L_2 distance of the LDM models, based on the scale parameter s for CFG [39]. The distributions for CosPlace [52] with $d = 2048$, and EigenPlaces [19] with $d \in \{128, 512, 2048\}$ are shown.

C.3 BayesCap training curves

In this section, we present the training curves of the BayesCap [58] models, as described in Section 6.2.2. Each training session consists of 5,000 iterations per epoch, with a batch size of 96. Three training curves are shown here. Figures C.36 and C.37 display the training curves for BayesCap, differing only in the number of epochs–30 epochs and 50 epochs, respectively. Figure C.38 shows the training curve for BayesCapCycle, which was trained for 30 epochs.

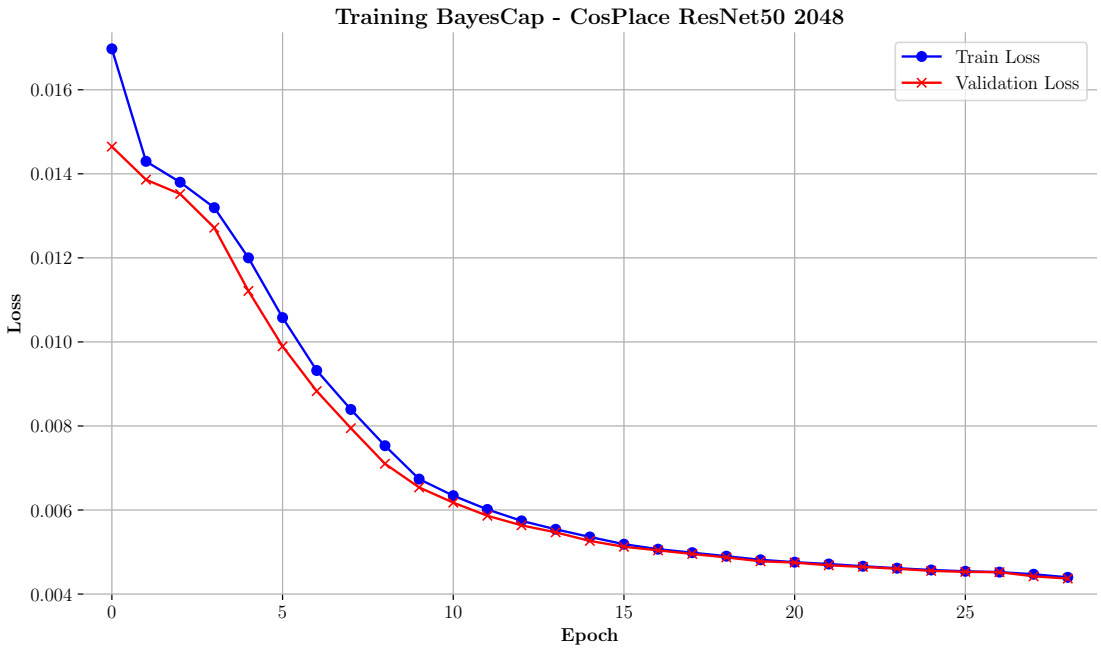


Figure C.36: Training plot of BayesCap [58] on top of CosPlace [52], with a descriptor dimension of $d = 2048$. The model was trained for 30 epochs.

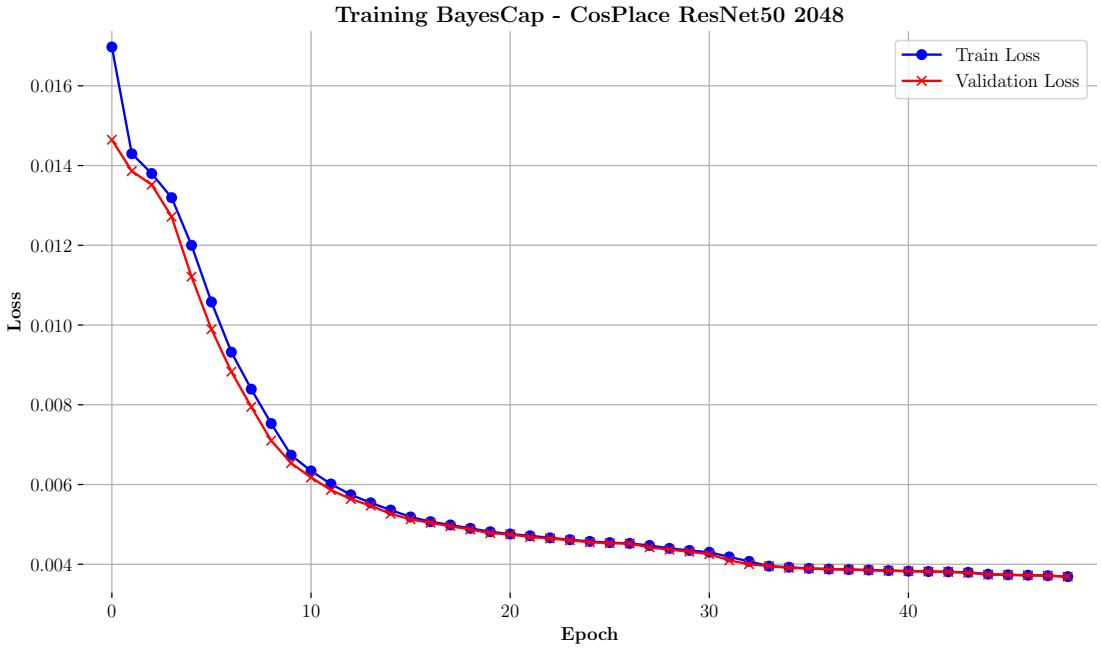


Figure C.37: Training plot of BayesCap [58] on top of CosPlace [52], with a descriptor dimension of $d = 2048$. The model was trained for 50 epochs.

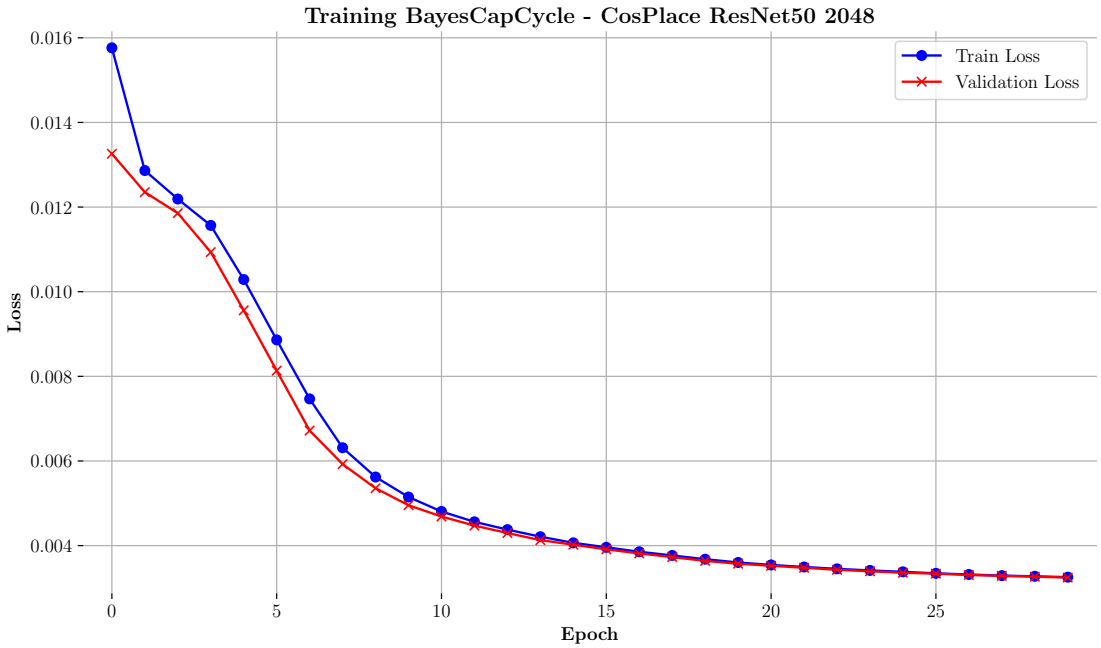


Figure C.38: Training plot of BayesCapCycle on top of CosPlace [52], with a descriptor dimension of $d = 2048$. The model was trained for 30 epochs.

C.4 Binning strategy for BayesCap results

This section details the adaptive binning strategy introduced in Section 6.2.4 and Section 6.2.5 for the quantitative evaluation of BayesCap [58] models. The full Python code is given in Listing C.1.

```
import numpy as np

def get_bins_adaptive(uncertainties, min_num_samples, max_num_bins,
    ↪ min_num_bins, min_width_factor=0.05, max_merge_attempts=100):
    uncertainties_sorted = np.sort(uncertainties)
    min_val, max_val = uncertainties_sorted[0], uncertainties_sorted[-1]

    q25, q75 = np.percentile(uncertainties, [25, 75])
    iqr = q75 - q25

    min_width = min_width_factor * iqr
    range_uncertainties = max_val - min_val
    bin_width = max(min_width, range_uncertainties / max_num_bins)
    num_bins = int(np.ceil(range_uncertainties / bin_width))
    num_bins = min(max(num_bins, min_num_bins), max_num_bins)
    bin_width = range_uncertainties / num_bins
    bin_edges = np.linspace(min_val, max_val, num_bins + 1)
    binned_samples = np.histogram(uncertainties, bins=bin_edges)[0]

    merge_attempts = 0

    while any(binned_samples < min_num_samples) and merge_attempts <
    ↪ max_merge_attempts:
        merge_attempts += 1
        new_bin_edges = []

        i = 0
        while i < len(bin_edges) - 1:
            if binned_samples[i] < min_num_samples and i < len(bin_edges) - 2:
                new_bin_edges.append(bin_edges[i])
                i += 2
            else:
                new_bin_edges.append(bin_edges[i])
                i += 1

        if binned_samples[-1] < min_num_samples and len(bin_edges) > 1:
            new_bin_edges[-1] = bin_edges[-1]
        elif binned_samples[-1] >= min_num_samples:
            new_bin_edges.append(bin_edges[-1])

        bin_edges = np.array(new_bin_edges)
        binned_samples = np.histogram(uncertainties, bins=bin_edges)[0]
        bin_edges = np.unique(bin_edges)
```

```
42
43          if len(bin_edges) <= min_num_bins:
44              break
45
46      return bin_edges
```

Listing C.1: Full Python code for the adaptive binning strategy of Section 6.2.4 and Section 6.2.5.

C.5 BayesCap uncertainty distributions

In this section, we build upon the results presented in Section 6.2.5 by examining how uncertainties, computed on the validation sets of Pitts30k [15], SF-XL [52], and MSLS [78], are distributed with respect to Recall@1.

Specifically, we define a query as *positive* if its first nearest neighbor in the database is within 25 meters of its true location, and *negative* otherwise. The distributions of these uncertainties are shown in Fig. C.39–C.41. The x-axis represents the uncertainty values, while the y-axis shows the match result. To enhance the visibility of the points' density, we also add a small amount of Gaussian noise to the y-axis.

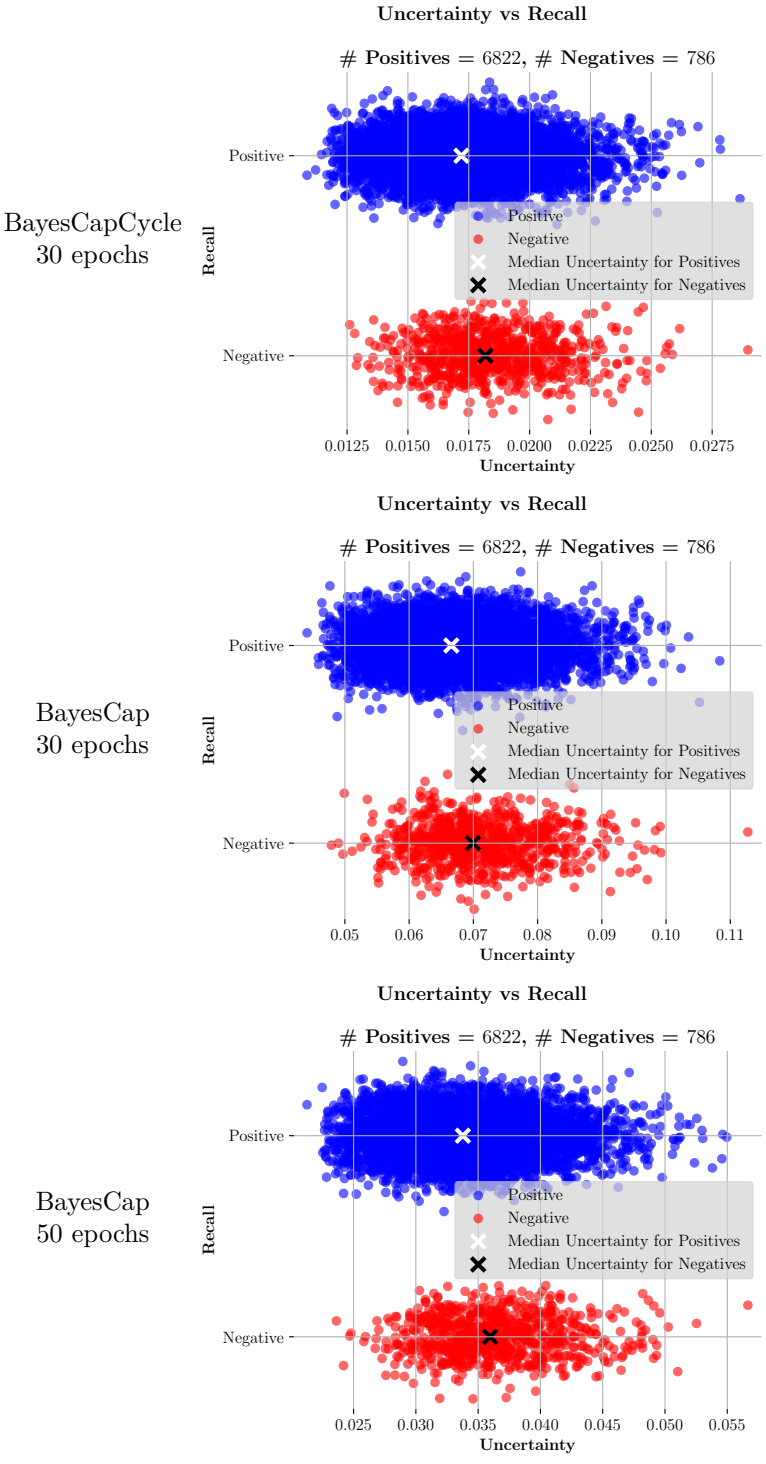


Figure C.39: Uncertainty distributions for all trained BayesCap [58] models, computed on the Pitts30k [15] validation set.

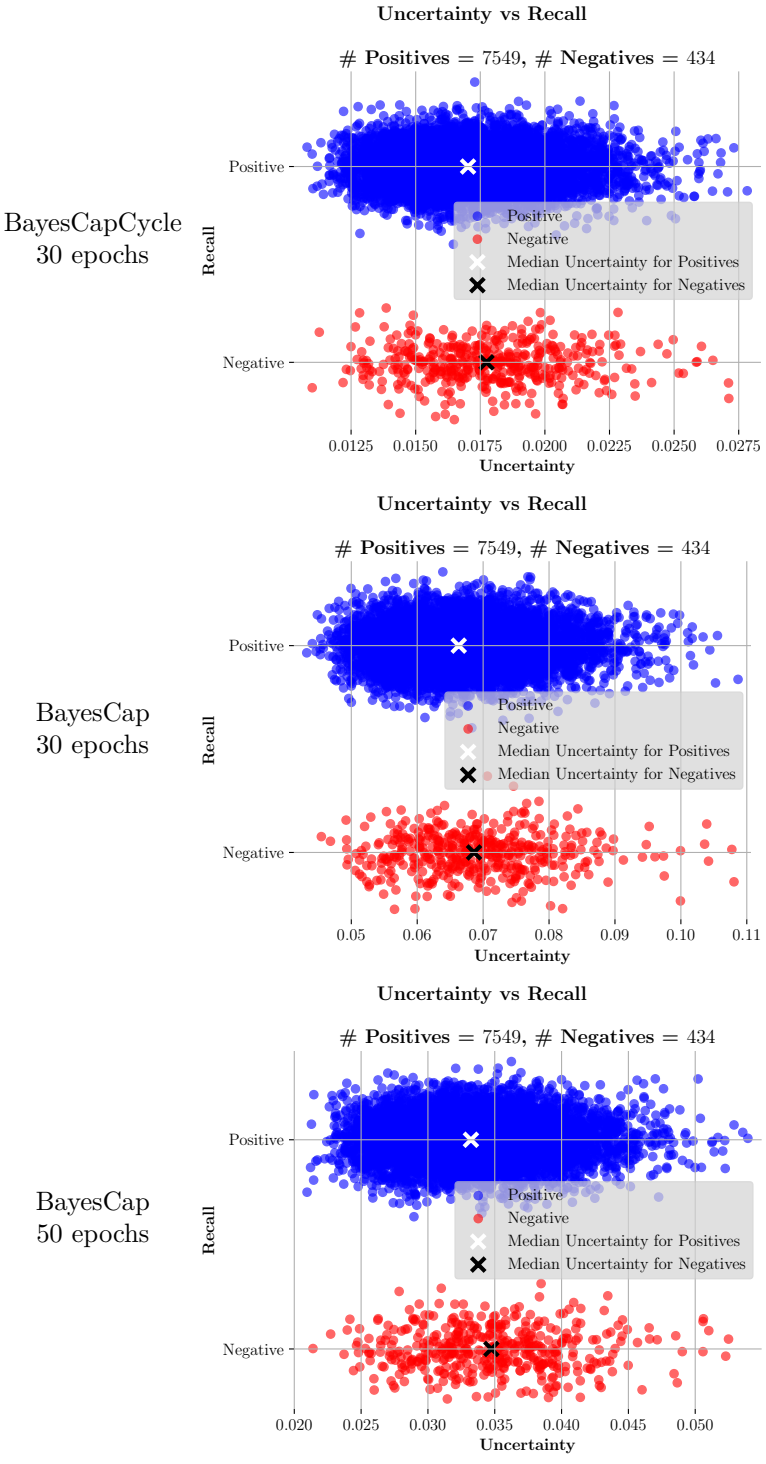


Figure C.40: Uncertainty distributions for all trained BayesCap [58] models, computed on the SF-XL [52] validation set.

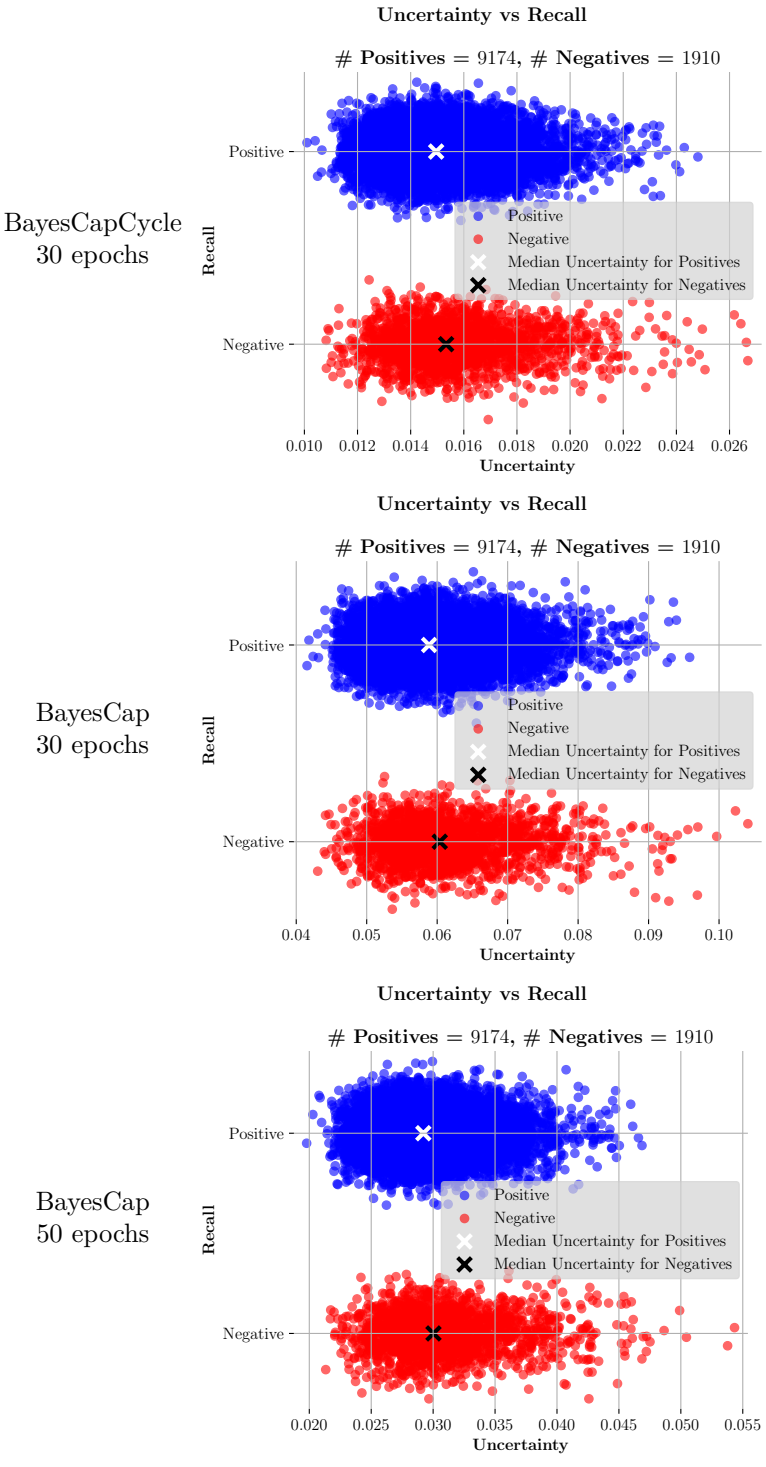


Figure C.41: Uncertainty distributions for all trained BayesCap [58] models, computed on the MSLS [78] validation set.

Bibliography

[1] Nils J Nilsson. *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998 (cit. on pp. 4, 7).

[2] A. M. TURING. «I.—COMPUTING MACHINERY AND INTELLIGENCE». In: *Mind* LIX.236 (Oct. 1950), pp. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/61209000/mind_lix_236_433.pdf. URL: https://doi.org/10.1093/mind/LIX.236.433 (cit. on p. 4).

[3] Melanie Mitchell et al. «Artificial intelligence: A guide for thinking humans». In: (2019) (cit. on p. 4).

[4] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016 (cit. on p. 4).

[5] Tom M Mitchell and Tom M Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997 (cit. on p. 5).

[6] Frank Rosenblatt. «The perceptron: a probabilistic model for information storage and organization in the brain.» In: *Psychological review* 65.6 (1958), p. 386 (cit. on pp. 6, 7, 9).

[7] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. «Learning representations by back-propagating errors». In: *nature* 323.6088 (1986), pp. 533–536 (cit. on p. 8).

[8] George Cybenko. «Approximation by superpositions of a sigmoidal function». In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314 (cit. on p. 9).

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. «Backpropagation Applied to Handwritten Zip Code Recognition». In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541 (cit. on p. 9).

[10] Carlo Masone and Barbara Caputo. «A Survey on Deep Visual Place Recognition». In: *IEEE Access* 9 (2021), pp. 19516–19547. DOI: 10.1109/ACCESS.2021.3054937 (cit. on pp. 11, 12).

[11] David G. Lowe. «Distinctive Image Features from Scale-Invariant Keypoints». In: *International Journal of Computer Vision* 60 (2004), pp. 91–110. URL: https://api.semanticscholar.org/CorpusID:174065 (cit. on p. 11).

[12] Relja Arandjelovic and Andrew Zisserman. «All About VLAD». In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 1578–1585. DOI: 10.1109/CVPR.2013.207 (cit. on p. 11).

[13] Sergio Izquierdo and Javier Civera. «Close, But Not There: Boosting Geographic Distance Sensitivity in Visual Place Recognition». In: *arXiv preprint arXiv:2407.02422* (2024) (cit. on pp. 12, 46, 75, 83, 86, 100).

[14] Jerome Revaud, Jon Almazán, Rafael S Rezende, and Cesar Roberto de Souza. «Learning with average precision: Training image retrieval with a listwise loss». In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 5107–5116 (cit. on pp. 12, 46, 75, 83, 85, 100).

[15] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. «NetVLAD: CNN architecture for weakly supervised place recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5297–5307 (cit. on pp. 12, 15, 46, 64, 66, 81, 83, 97, 101, 108, 109).

[16] Mubariz Zaffar, Liangliang Nan, and Julian FP Kooij. «On the Estimation of Image-matching Uncertainty in Visual Place Recognition». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 17743–17753 (cit. on pp. 12, 14, 37).

[17] Feng Lu, Xiangyuan Lan, Lijun Zhang, Dongmei Jiang, Yaowei Wang, and Chun Yuan. «CricaVPR: Cross-image Correlation-aware Representation Learning for Visual Place Recognition». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 16772–16782 (cit. on pp. 14, 46, 79, 83, 92, 100).

[18] Amar Ali-Bey, Brahim Chaib-Draa, and Philippe Giguere. «Mixvpr: Feature mixing for visual place recognition». In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2023, pp. 2998–3007 (cit. on pp. 14, 46, 81, 83, 96, 101).

[19] Gabriele Berton, Gabriele Trivigno, Barbara Caputo, and Carlo Masone. «Eigenplaces: Training viewpoint robust models for visual place recognition». In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 11080–11090 (cit. on pp. 14, 46, 61, 62, 79, 80, 83, 93–95, 103).

[20] Nikhil Keetha, Avneesh Mishra, Jay Karhade, Krishna Murthy Jatavallabhula, Sebastian Scherer, Madhava Krishna, and Sourav Garg. «Anyloc: Towards universal visual place recognition». In: *IEEE Robotics and Automation Letters* (2023) (cit. on p. 14).

[21] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. «Emerging properties in self-supervised vision transformers». In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 9650–9660 (cit. on p. 14).

[22] Maxime Oquab et al. «Dinov2: Learning robust visual features without supervision». In: *arXiv preprint arXiv:2304.07193* (2023) (cit. on pp. 14, 46).

[23] María Leyva-Vallina, Nicola Strisciuglio, and Nicolai Petkov. «Data-efficient large scale place recognition with graded similarity supervision». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 23487–23496 (cit. on pp. 14, 16).

[24] Deen Dayal Mohan, Bhavin Jawade, Srirangaraj Setlur, and Venu Govindaraju. «Deep metric learning for computer vision: A brief overview». In: *Handbook of Statistics* 48 (2023), pp. 59–79 (cit. on p. 14).

[25] Amar Ali-bey, Brahim Chaib-draa, and Philippe Giguere. «Gsv-cities: Toward appropriate supervised visual place recognition». In: *Neurocomputing* 513 (2022), pp. 194–203 (cit. on pp. 14, 46, 76, 83, 87, 100).

[26] Elad Hoffer and Nir Ailon. «Deep metric learning using triplet network». In: *Similarity-based pattern recognition: third international workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*. Springer. 2015, pp. 84–92 (cit. on p. 15).

[27] Raia Hadsell, Sumit Chopra, and Yann LeCun. «Dimensionality reduction by learning an invariant mapping». In: *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*. Vol. 2. IEEE. 2006, pp. 1735–1742 (cit. on p. 15).

[28] Xun Wang, Xintong Han, Weilin Huang, Dengke Dong, and Matthew R Scott. «Multi-similarity loss with general pair weighting for deep metric learning». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 5022–5030 (cit. on p. 16).

[29] Patrick Esser, Robin Rombach, and Bjorn Ommer. «Taming transformers for high-resolution image synthesis». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 12873–12883 (cit. on pp. 17, 34).

[30] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. «Zero-shot text-to-image generation». In: *International conference on machine learning*. Pmlr. 2021, pp. 8821–8831 (cit. on p. 17).

[31] Alec Radford et al. «Learning transferable visual models from natural language supervision». In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763 (cit. on pp. 17, 41).

[32] Diederik P Kingma. «Auto-encoding variational bayes». In: *arXiv preprint arXiv:1312.6114* (2013) (cit. on pp. 18–20, 34).

[33] Aaron Van Den Oord, Oriol Vinyals, et al. «Neural discrete representation learning». In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 18, 20, 34).

[34] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. «Generative adversarial nets». In: *Advances in neural information processing systems* 27 (2014) (cit. on pp. 18, 21, 25).

[35] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. «Demystifying mmd gans». In: *arXiv preprint arXiv:1801.01401* (2018) (cit. on pp. 18, 47).

[36] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. «Gans trained by a two time-scale update rule converge to a local nash equilibrium». In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 18, 47, 49).

[37] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. «Improved precision and recall metric for assessing generative models». In: *Advances in neural information processing systems* 32 (2019) (cit. on pp. 18, 47–49, 51).

[38] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. «Improved techniques for training gans». In: *Advances in neural information processing systems* 29 (2016) (cit. on pp. 18, 23, 47).

[39] Jonathan Ho and Tim Salimans. «Classifier-free diffusion guidance». In: *arXiv preprint arXiv:2207.12598* (2022) (cit. on pp. 18, 50–52, 54–57, 83, 85–103).

[40] Jiaming Song, Chenlin Meng, and Stefano Ermon. «Denoising diffusion implicit models». In: *arXiv preprint arXiv:2010.02502* (2020) (cit. on pp. 18, 29, 31–33).

[41] Jonathan Ho, Ajay Jain, and Pieter Abbeel. «Denoising diffusion probabilistic models». In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851 (cit. on pp. 18, 23, 25, 26, 28).

[42] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. «High-resolution image synthesis with latent diffusion models». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695 (cit. on pp. 18, 34, 35, 46–50, 57, 83).

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 18).

[44] Geoffrey E Hinton and Ruslan R Salakhutdinov. «Reducing the dimensionality of data with neural networks». In: *science* 313.5786 (2006), pp. 504–507 (cit. on p. 18).

[45] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. «Greedy layer-wise training of deep networks». In: *Advances in neural information processing systems* 19 (2006) (cit. on p. 18).

[46] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. «U-net: Convolutional networks for biomedical image segmentation». In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer. 2015, pp. 234–241 (cit. on pp. 19, 34).

[47] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. «Extracting and composing robust features with denoising autoencoders». In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1096–1103 (cit. on p. 19).

[48] Lovedeep Gondara. «Medical Image Denoising Using Convolutional Denoising Autoencoders». In: *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. 2016, pp. 241–246. DOI: 10.1109/ICDMW.2016.0041 (cit. on p. 19).

[49] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. «Lossy image compression with compressive autoencoders». In: *arXiv preprint* (2017). arXiv:1703.00395 (cit. on p. 19).

[50] Mayu Sakurada and Takehisa Yairi. «Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction». In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. MLSDA'14. Gold Coast, Australia QLD, Australia: Association for Computing Machinery, 2014, pp. 4–11. ISBN: 9781450331593. DOI: 10.1145/2689746.2689747. URL: https://doi.org/10.1145/2689746.2689747 (cit. on p. 19).

[51] Sergio Izquierdo and Javier Civera. «Optimal transport aggregation for visual place recognition». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 17658–17668 (cit. on pp. 21, 46, 82, 83, 98, 101).

[52] Gabriele Berton, Carlo Masone, and Barbara Caputo. «Rethinking visual geo-localization for large-scale applications». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 4878–4888 (cit. on pp. 21, 29, 46, 47, 50, 52–59, 63, 64, 67, 76–78, 83, 84, 88–91, 102–105, 108, 110).

[53] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. «Deep unsupervised learning using nonequilibrium thermodynamics». In: *International conference on machine learning*. PMLR. 2015, pp. 2256–2265 (cit. on pp. 23, 24).

[54] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. «Image-to-image translation with conditional adversarial networks». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134 (cit. on p. 34).

[55] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. «The unreasonable effectiveness of deep features as a perceptual metric». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595 (cit. on p. 34).

[56] A Vaswani. «Attention is all you need». In: *Advances in Neural Information Processing Systems* (2017) (cit. on p. 35).

[57] Alex Kendall and Yarin Gal. «What uncertainties do we need in bayesian deep learning for computer vision?» In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 37).

[58] U. Upadhyay, S. Karthik, Y. Chen, M. Mancini, and Z. Akata. «BayesCap: Bayesian Identity Cap for Calibrated Uncertainty in Frozen Neural Networks». In: *European Conference on Computer Vision (ECCV 2022)*. 2022 (cit. on pp. 37–41, 60–68, 70, 104–106, 109–111).

[59] Gianni Franchi, Olivier Laurent, Maxence Leguéry, Andrei Bursuc, Andrea Pilzer, and Angela Yao. «Make Me a BNN: A Simple Strategy for Estimating Bayesian Uncertainty from Pre-trained Models». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 12194–12204 (cit. on p. 37).

[60] Roman Kail, Kirill Fedyanin, Nikita Muravev, Alexey Zaytsev, and Maxim Panov. «Scaleface: Uncertainty-aware deep metric learning». In: *2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2023, pp. 1–10 (cit. on p. 37).

[61] Sanghyuk Chun, Seong Joon Oh, Rafael Sampaio De Rezende, Yannis Kalantidis, and Diane Larlus. «Probabilistic embeddings for cross-modal retrieval». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 8415–8424 (cit. on p. 38).

[62] Yibo Miao, Yu Lei, Feng Zhou, and Zhijie Deng. «Bayesian Exploration of Pre-trained Models for Low-shot Image Classification». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 23849–23859 (cit. on p. 38).

[63] Uddeshya Upadhyay, Yanbei Chen, Tobias Hebb, Sergios Gatidis, and Zeynep Akata. «Uncertainty Guided Progressive GANs for Medical Image Translation». In: *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer. 2021 (cit. on p. 38).

[64] U. Upadhyay, Y. Chen, and Z. Akata. «Robustness via Uncertainty-aware Cycle Consistency». In: *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*. 2021 (cit. on p. 38).

[65] U. Upadhyay, S. Karthik, M. Mancini, and Z. Akata. «ProbVLM: Probabilistic Adapter for Frozen Vision-Language Models». In: *International Conference on Computer Vision (ICCV 2023)*. 2023 (cit. on pp. 38, 41–44, 60, 61, 63).

[66] Frederik Warburg, Marco Miani, Silas Brack, and Søren Hauberg. «Bayesian metric learning for uncertainty quantification in image retrieval». In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 69178–69190 (cit. on pp. 38, 63).

[67] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. «Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation». In: *International conference on machine learning*. PMLR. 2022, pp. 12888–12900 (cit. on p. 41).

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep residual learning for image recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on p. 46).

[69] Alexey Dosovitskiy. «An image is worth 16x16 words: Transformers for image recognition at scale». In: *arXiv preprint arXiv:2010.11929* (2020) (cit. on p. 46).

[70] Karen Simonyan. «Very deep convolutional networks for large-scale image recognition». In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on pp. 46, 49).

[71] Yixiao Ge, Haibo Wang, Feng Zhu, Rui Zhao, and Hongsheng Li. «Self-supervising fine-grained region similarities for large-scale image localization». In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*. Springer. 2020, pp. 369–386 (cit. on pp. 46, 82, 83, 99, 101).

[72] I Loshchilov. «Decoupled weight decay regularization». In: *arXiv preprint arXiv:1711.05101* (2017) (cit. on p. 47).

[73] Tero Karras. «A Style-Based Generator Architecture for Generative Adversarial Networks». In: *arXiv preprint arXiv:1812.04948* (2019) (cit. on p. 47).

[74] Anton Obukhov, Maximilian Seitzer, Po-Wei Wu, Semen Zhydenko, Jonathan Kyl, and Elvis Yu-Jing Lin. *High-fidelity performance metrics for generative models in PyTorch*. Version v0.3.0. Version: 0.3.0, DOI: 10.5281/zenodo.4957738. 2020. DOI: 10.5281/zenodo.4957738. URL: https://github.com/toshas/torch-fidelity (cit. on p. 47).

[75] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. «Going deeper with convolutions». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9 (cit. on p. 47).

[76] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. «Rethinking the inception architecture for computer vision». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826 (cit. on p. 47).

[77] Andrew Brock. «Large Scale GAN Training for High Fidelity Natural Image Synthesis». In: *arXiv preprint arXiv:1809.11096* (2018) (cit. on p. 49).

[78] Frederik Warburg, Soren Hauberg, Manuel Lopez-Antequera, Pau Gargallo, Yubin Kuang, and Javier Civera. «Mapillary Street-Level Sequences: A Dataset for Lifelong Place Recognition». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020 (cit. on pp. 64, 65, 68, 108, 111).

[79] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. «Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)». In: *International conference on machine learning*. PMLR. 2018, pp. 2668–2677 (cit. on p. 69).

[80] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. «Concept bottleneck models». In: *International conference on machine learning*. PMLR. 2020, pp. 5338–5348 (cit. on p. 71).

[81] Mateo Espinosa Zarlenga et al. «Concept embedding models: Beyond the accuracy-explainability trade-off». In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 21400–21413 (cit. on p. 71).