

VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
SOFTWARE ENGINEERING BACHELOR'S STUDY PROGRAMME

**Vaizdų panašumo metodų palyginimas naudojant
DISC21 duomenų rinkinį**

**Comparison of Image Similarity Methods Using the DISC21
Dataset**

Coursework

Author: 4th year, Group 5 student

Vytautas Mielkus

Supervisor: lect. Boleslovas Dapkūnas

Vilnius – 2026

Contents

ABBREVIATIONS	3
INTRODUCTION	4
1. THEORETICAL BACKGROUND	6
1.1. Transformer Architecture	6
1.1.1. Attention Mechanism	6
1.1.2. Transformer Encoder Structure	6
1.2. Vision Transformers	7
1.2.1. Treating Images as Sequences	7
1.2.2. ViT Model Sizes	8
1.3. Self-Supervised Learning and DINOv2	8
1.3.1. How DINO Works	9
1.3.2. DINOv2	9
1.4. Metric Learning and Loss Functions.....	9
1.4.1. Contrastive Loss	10
1.4.2. Triplet Loss	10
1.4.3. ArcFace Loss	11
1.4.4. Summary of Loss Functions	11
1.5. Evaluation Metrics	12
1.5.1. Precision at K	12
1.5.2. Mean Average Precision	12
1.5.3. Micro Average Precision	12
2. METHODOLOGY	13
2.1. DISC21 Dataset	13
2.2. Model Architecture	13
2.3. Data Augmentation.....	14
2.4. Experimental Setup	15
2.5. Evaluation Protocol	15
3. EXPERIMENTS AND RESULTS	16
3.1. Baseline Evaluation.....	16
3.2. Training Dynamics	16
3.3. Fine-tuning with Contrastive Loss	17
3.4. Fine-tuning with Triplet Loss	17
3.5. Fine-tuning with ArcFace Loss	17
3.6. Results Comparison	18
3.7. Discussion	18
RESULTS AND CONCLUSIONS	20
REFERENCES	21
APPENDIXES	22
Appendix 1. Detailed Training Logs	22
Appendix 2. Experimental Environment	24

Abbreviations

- CNN.** Convolutional Neural Network
DISC21. Dataset for Image Similarity Challenge 2021
DINOv2. Self-DIstillation with NO labels, version 2
mAP. Mean Average Precision
P@K. Precision at K
ViT. Vision Transformer
 μ AP. Micro Average Precision

Introduction

Image similarity detection is a fundamental computer vision task with significant practical applications in content moderation, copyright enforcement, and misinformation tracking on social media platforms [DTP⁺21]. The task involves determining whether a query image is a modified copy of any image in a reference database, where modifications may include cropping, filtering, rotation, text overlays, or other transformations.

Figure 1 illustrates concrete examples from the DISC21 dataset [DTP⁺21]. Each query image on the left is a modified version of the corresponding reference image on the right. The modifications include cropping, color adjustments, and composition changes. Despite these transformations, an effective similarity detection system should be able to match the query to its source reference.

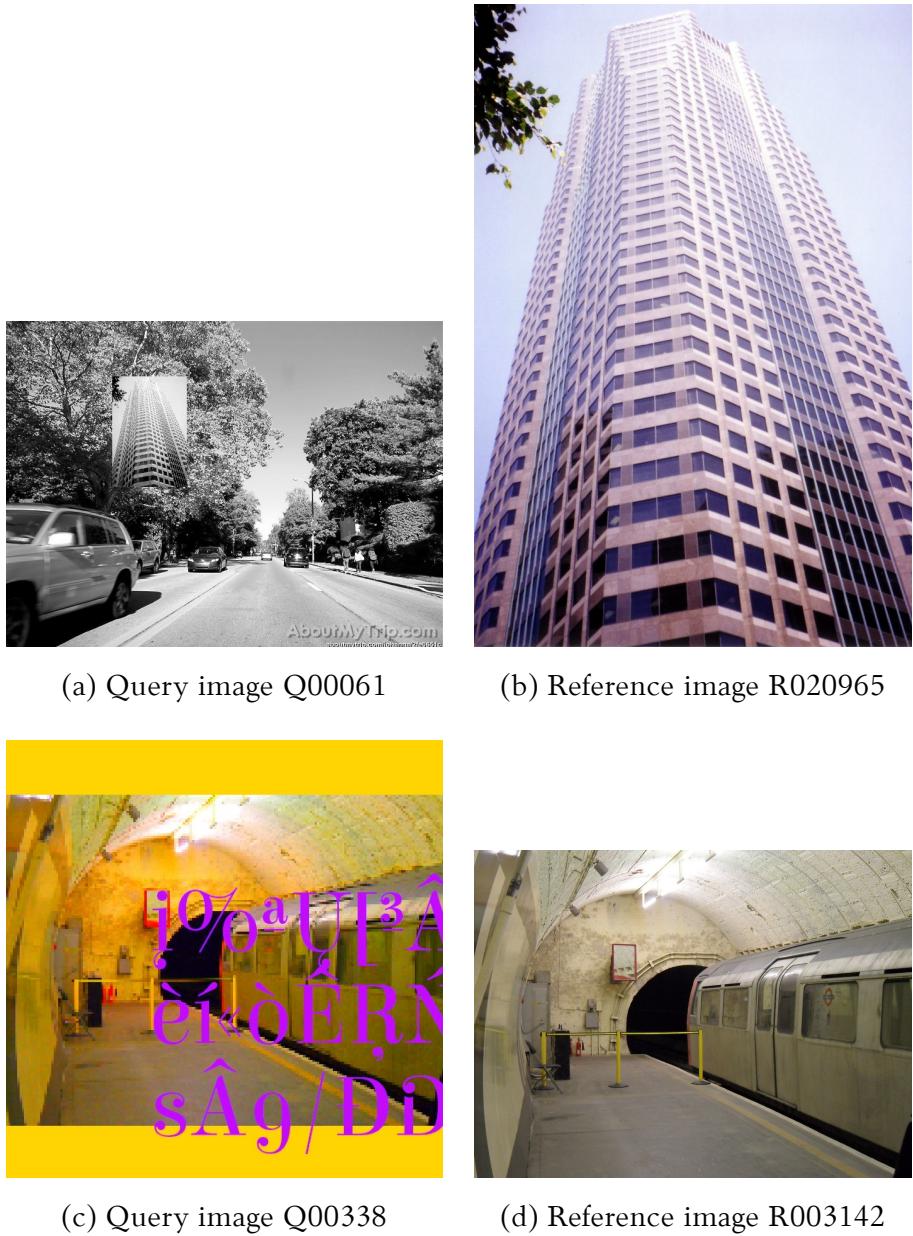


Figure 1. Examples of query-reference pairs from the DISC21 dataset. Queries (left) are modified versions of references (right). The task is to correctly match each query to its source reference.

Recent advances in self-supervised learning have produced powerful visual feature extractors that can be applied to similarity detection tasks. DINOv2 [ODM⁺24], developed by Meta AI, represents the state-of-the-art in learning robust visual features without supervision. Built on the Vision Transformer (ViT) architecture [DBK⁺21], DINOv2 produces embeddings that capture semantic image content while remaining invariant to various transformations.

The choice of loss function during fine-tuning significantly impacts the quality of learned embeddings for similarity tasks. Metric learning approaches such as contrastive loss [CKN⁺20], triplet loss [SKP15], and ArcFace loss [DGY⁺22] each offer different strategies for learning discriminative embeddings. Understanding their comparative effectiveness is important for developing practical image similarity systems.

Aim. The aim of this work is to compare the effectiveness of different loss functions for fine-tuning a pre-trained DINOv2 model on the image similarity challenge using the DISC21 dataset.

Objectives:

1. Establish a baseline by evaluating the pre-trained DINOv2 model without fine-tuning.
2. Implement and train models using three loss functions: contrastive loss, triplet loss, and ArcFace loss.
3. Evaluate all models using standard retrieval metrics: Precision@K, mean Average Precision (mAP), and micro Average Precision (μ AP).
4. Analyze and compare the results to determine which loss function is most effective for image similarity.

1. Theoretical Background

This section explains the key concepts needed to understand the image similarity methods used in this work. We start with the Transformer architecture, then explain how it was adapted for images (Vision Transformers), discuss self-supervised learning, and finally describe the loss functions used for training.

1.1. Transformer Architecture

The Transformer is a neural network architecture introduced by Vaswani et al. [VSP⁺17] in 2017. It was originally designed for text processing tasks like translation. The key idea is the **attention mechanism**, which allows the model to look at all parts of the input at once and decide which parts are most important for each output.

Before Transformers, most text models processed words one by one in order (like reading a sentence from left to right). This was slow and made it hard for the model to connect words that were far apart in a sentence. Transformers solve this by processing all words at the same time and using attention to find connections between any words, regardless of their position.

1.1.1. Attention Mechanism

The attention mechanism is the core idea behind Transformers. In simple terms, attention answers the question: "When processing this part of the input, which other parts should I pay attention to?"

Vaswani et al. [VSP⁺17] define the attention function using three components:

- **Query (Q)**: What we're looking for
- **Key (K)**: What each element offers
- **Value (V)**: The actual content to retrieve

The formula for computing attention, as defined in the original paper [VSP⁺17], is:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \quad (1)$$

In plain terms: the model computes a similarity score between each query and all keys (the QK^T part), normalizes these scores to sum to 1 (the softmax part), and uses them to create a weighted combination of values. The division by $\sqrt{d_k}$ (where d_k is the size of the key vectors) keeps the numbers from getting too large.

Multi-head attention runs several attention operations in parallel, each learning to focus on different types of relationships. For example, one "head" might learn to connect subjects with verbs, while another connects adjectives with nouns. The outputs are combined at the end.

1.1.2. Transformer Encoder Structure

The Transformer encoder is built by stacking multiple identical layers. Each layer has two main parts:

1. **Attention layer:** Lets each position look at all other positions
2. **Feed-forward layer:** Processes each position independently

Between these parts, the model uses "residual connections" (adding the input back to the output) and "layer normalization" (keeping values in a reasonable range). Figure 2 shows this structure.

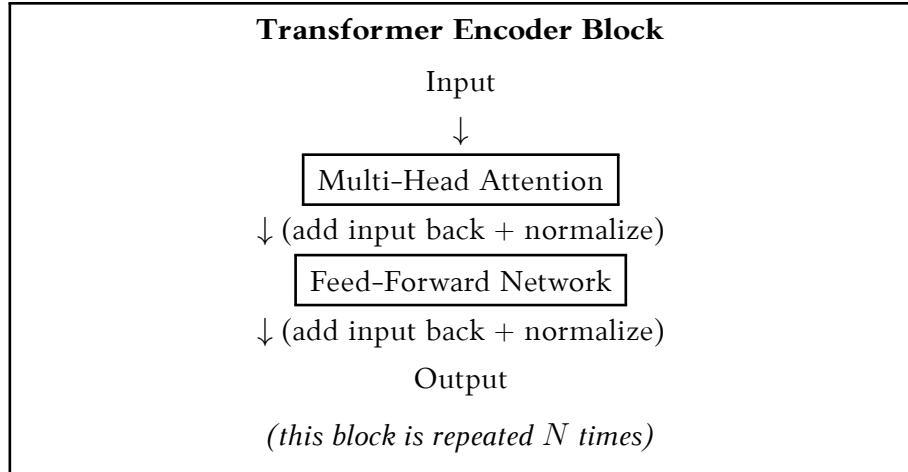


Figure 2. Structure of a Transformer encoder block, as described in [VSP⁺17].

1.2. Vision Transformers

The Vision Transformer (ViT) was introduced by Dosovitskiy et al. [DBK⁺21] in 2020. The main idea is simple: instead of using Transformers for text, use them for images. But images are not sequences of words, so how do we adapt the architecture?

1.2.1. Treating Images as Sequences

The solution is to split the image into small square pieces called **patches**. For example, a 224×224 pixel image can be divided into 196 patches of 16×16 pixels each. Each patch is then converted into a vector (a list of numbers), similar to how words are converted to vectors in text models.

The process works as follows:

1. Split the image into N non-overlapping patches (e.g., $14 \times 14 = 196$ patches)
2. Flatten each patch into a 1D vector
3. Project each vector to a fixed size using a linear layer
4. Add position information so the model knows where each patch came from
5. Process through the Transformer encoder

A special "classification token" [CLS] is added at the beginning. After processing, this token's output is used as the image representation. Figure 3 illustrates this process.

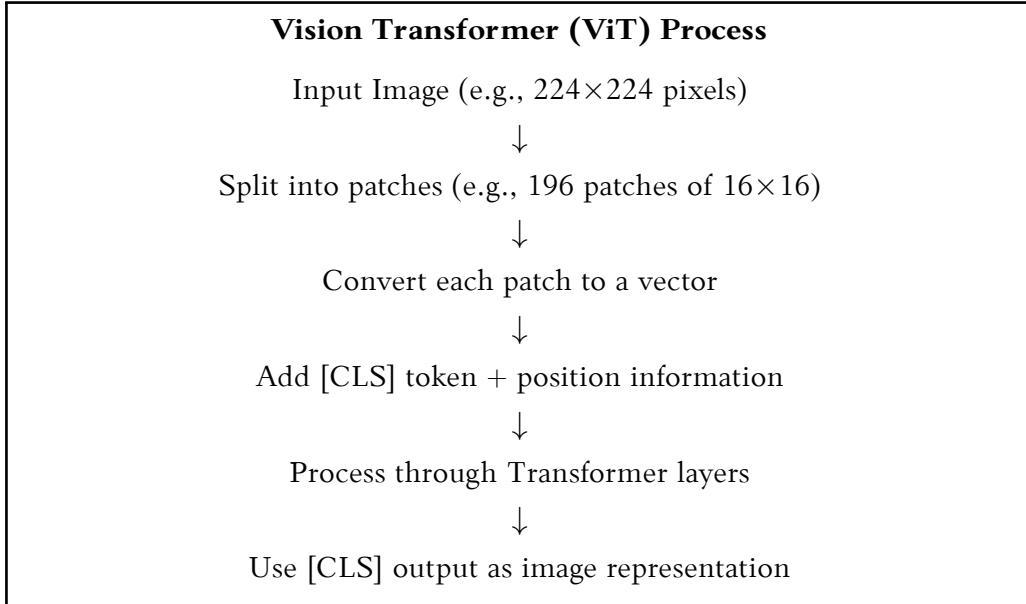


Figure 3. How Vision Transformer processes an image, based on [DBK⁺21].

1.2.2. ViT Model Sizes

ViT comes in different sizes, from small to huge. Larger models can learn more complex patterns but require more computing power. Table 1 shows the standard configurations. The naming convention ViT-X/Y means variant X with patch size Y (e.g., ViT-S/14 is the Small model with 14×14 patches).

Table 1. Vision Transformer model sizes, as defined in [DBK⁺21]

Model	Layers	Hidden Size	Heads	Parameters
ViT-S (Small)	12	384	6	22M
ViT-B (Base)	12	768	12	86M
ViT-L (Large)	24	1024	16	307M
ViT-H (Huge)	32	1280	16	632M

In this work, we use the ViT-S (Small) variant because it offers a good balance between performance and computational requirements.

1.3. Self-Supervised Learning and DINOv2

Self-supervised learning is a way to train neural networks without manually labeled data. Instead of telling the model "this image is a cat", the model learns by solving tasks it creates for itself. For example, it might learn to match different views of the same image, or to predict missing parts of an image.

This is important because labeling millions of images is expensive and time-consuming. Self-supervised learning allows models to learn from huge amounts of unlabeled images from the internet.

1.3.1. How DINO Works

DINO (Self-**DI**stillation with **NO** labels) [CTM⁺21] uses a clever training approach with two networks:

- **Student network:** The model being trained
- **Teacher network:** A slowly-updated copy of the student

During training, both networks see the same image but with different random changes applied (cropping, color changes, etc.). The student tries to produce the same output as the teacher. Since the teacher changes slowly, it provides stable targets for the student to learn from.

This approach works surprisingly well: the model learns to recognize objects and scenes without ever being told what they are. The resulting image representations capture the semantic meaning of images.

1.3.2. DINOv2

DINOv2 [ODM⁺24] is an improved version released by Meta AI in 2023. The main improvements are:

- **More training data:** 142 million carefully selected images
- **Better training:** Combines multiple learning objectives
- **Multiple sizes:** From small (22M parameters) to giant (1.1B parameters)

The key advantage of DINOv2 for our task is that it produces image embeddings that are:

- **Meaningful:** Similar images get similar embeddings
- **Robust:** Embeddings stay similar even when images are cropped, rotated, or color-adjusted
- **General:** Works well for many different tasks without task-specific training

These properties make DINOv2 well-suited for image similarity detection.

1.4. Metric Learning and Loss Functions

Metric learning is about teaching a neural network to measure similarity between images. The goal is to convert images into vectors (called embeddings) such that:

- Similar images have embeddings that are **close together**
- Different images have embeddings that are **far apart**

This is illustrated in Figure 4. The key question is: how do we train a model to do this? The answer is through **loss functions** that define what "good" embeddings look like.

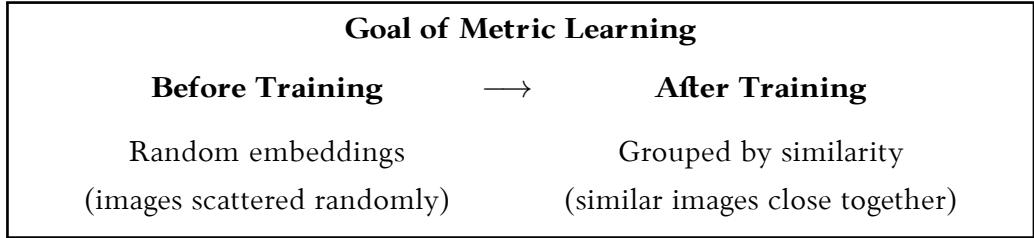


Figure 4. Metric learning teaches the model to place similar images close together in the embedding space.

Three different loss functions were tested, each with a different approach to this problem. The formulas come from the original papers that introduced these methods.

1.4.1. Contrastive Loss

Contrastive loss, popularized by SimCLR [CKN⁺20], works with **pairs** of images. For each pair, we know whether they should be similar (positive pair) or different (negative pair).

The formula [CKN⁺20] is:

$$\mathcal{L}_{\text{contrastive}} = y \cdot d^2 + (1 - y) \cdot \max(0, m - d)^2 \quad (2)$$

In plain terms:

- $y = 1$ means "these images should be similar" → minimize the distance d
- $y = 0$ means "these images should be different" → push them apart until distance is at least m

The parameter m (margin) defines how far apart negative pairs should be. Once they're far enough, the model stops pushing them further.

1.4.2. Triplet Loss

Triplet loss, introduced by Schroff et al. [SKP15] for face recognition, works with **three** images at a time:

- **Anchor**: The reference image
- **Positive**: An image that should be similar to anchor
- **Negative**: An image that should be different from anchor

The formula [SKP15] is:

$$\mathcal{L}_{\text{triplet}} = \max(0, d_{ap} - d_{an} + m) \quad (3)$$

The idea is simple: the anchor should be closer to the positive than to the negative, by at least a margin m . Figure 5 illustrates this.

Triplet Loss Requirement

$$\text{distance}(\text{Anchor}, \text{Positive}) + \text{margin} < \text{distance}(\text{Anchor}, \text{Negative})$$

In other words: the similar image should be closer than the different image, with some room to spare.

Figure 5. Triplet loss requires the positive to be closer to the anchor than the negative by at least margin m .

Triplet mining is important: if we pick random triplets, most will already satisfy the constraint ($\text{loss} = 0$) and the model learns nothing. "Hard" triplet mining finds difficult cases where the negative is closer than expected.

1.4.3. ArcFace Loss

ArcFace [DGY⁺22] takes a different approach. Instead of comparing pairs or triplets, it treats each image as a separate class and trains a classifier. The clever part is adding an **angular margin** that forces the model to create well-separated clusters.

The formula from [DGY⁺22] is complex:

$$\mathcal{L}_{\text{arcface}} = -\log \frac{e^{s \cdot \cos(\theta_y + m)}}{e^{s \cdot \cos(\theta_y + m)} + \sum_{j \neq y} e^{s \cdot \cos(\theta_j)}} \quad (4)$$

The key idea in simpler terms: the model learns to place each image's embedding close to its "class center" (a learned vector), and the margin m ensures there's clear separation between different classes. The parameters s (scale) and m (margin) control how strict this separation should be.

Figure 6 shows the difference from standard classification.

ArcFace vs Standard Classification

Standard

Classes can be close together	Margin forces separation
May overlap at boundaries	Clear gaps between classes

ArcFace

Figure 6. ArcFace adds a margin that forces clearer separation between image classes.

1.4.4. Summary of Loss Functions

Table 2 summarizes the three loss functions.

Table 2. Comparison of metric learning loss functions

Property	Contrastive	Triplet	ArcFace
Input type	Pairs	Triplets	Single samples
Requires labels	Pair labels	Class labels	Class labels
Mining needed	No	Yes	No
Formulation	Distance-based	Distance-based	Angular

1.5. Evaluation Metrics

To measure how well our models perform at finding similar images, we use several standard metrics from information retrieval.

1.5.1. Precision at K

Precision@K (P@K) answers a simple question: "Is the correct match among the top K results?"

- **P@1:** Is the #1 result correct? (most strict)
- **P@5:** Is the correct match in the top 5?
- **P@10:** Is the correct match in the top 10?

We calculate the percentage of queries where this is true. For example, P@1 = 60% means the correct match was ranked first for 60% of queries.

1.5.2. Mean Average Precision

Mean Average Precision (mAP) measures not just whether we find the correct match, but how high we rank it. A system that consistently ranks correct matches near the top will have higher mAP than one that finds them but ranks them lower.

1.5.3. Micro Average Precision

Micro Average Precision (μ AP) is the official metric used in the DISC21 challenge [DTP⁺21]. As explained in the DISC21 paper, this metric treats each query-reference pair equally, which is important for real-world scenarios where some queries might have multiple correct matches.

The DISC21 authors chose μ AP because copy detection operates in a "needle in haystack" setting—most query images do not have matches, and when matches exist, finding them with high confidence matters more than finding all possible matches.

2. Methodology

This section describes the dataset, model architecture, and experimental setup used in this work.

2.1. DISC21 Dataset

The DISC21 dataset [DTP⁺21] was created by Meta AI for the Image Similarity Challenge at NeurIPS 2021. It is designed to evaluate large-scale image copy detection systems in realistic scenarios.

The dataset contains over 2 million images with various transformations including:

- Geometric transformations: crops, rotations, flips, scaling
- Photometric changes: color filters, brightness/contrast adjustments
- Overlays: text, logos, borders, watermarks
- Complex edits: collages, memes, screenshots
- Adversarial perturbations: designed to fool detection systems

For this work, a subset of approximately 50,000 images per category was used due to computational constraints (see Table 3).

Table 3. DISC21 dataset subset used in experiments

Split	Images	Purpose
Training set	50,000	Fine-tuning with loss functions
Reference set	50,000	Database to search against
Development queries	50,000	Model validation and tuning
Test queries	50,000	Final evaluation

Ground truth files provide 10,000 query-to-reference mappings for both development and test sets, indicating which queries match which references.

2.2. Model Architecture

The model architecture consists of a pre-trained DINOv2 backbone followed by a trainable projection head (see Figure 7).

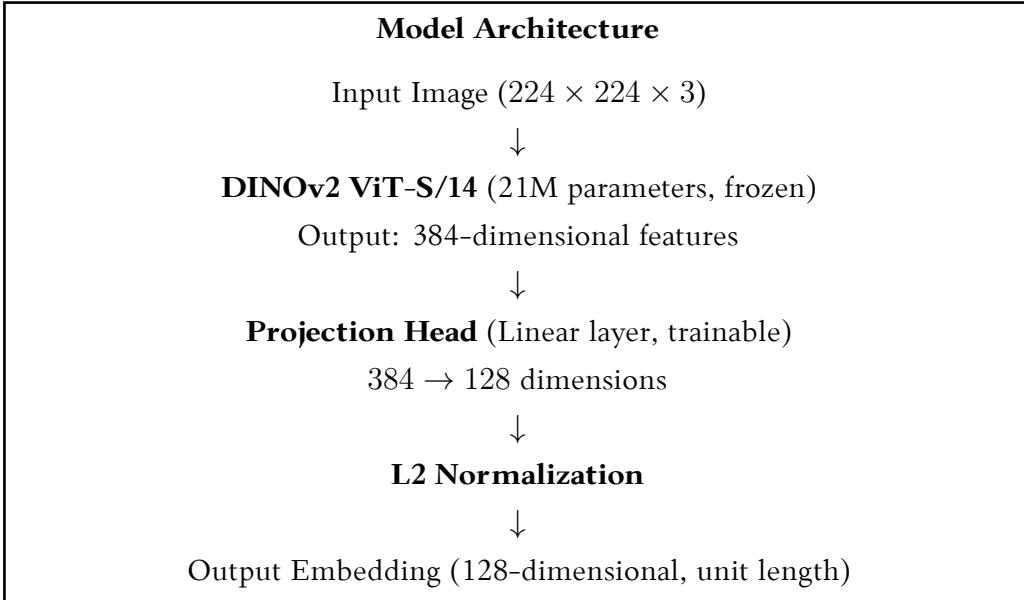


Figure 7. Model architecture: DINOv2 backbone with projection head

We used DINOv2 ViT-S/14 (the Small variant) because it is small enough to train on available hardware while still providing good results.

During training, we kept the DINOv2 backbone frozen (its weights were not updated). Only the projection head was trained. We did this because DINOv2 was already trained on millions of images and has learned useful features. If we trained the entire model, these features might get overwritten and lost. By freezing the backbone, we keep what DINOv2 already knows and only train the small projection head for our specific task.

The projection head takes the 384-dimensional output from DINOv2 and reduces it to 128 dimensions. This smaller size makes similarity comparisons faster. Finally, L2 normalization is applied so that all embeddings have the same length, which is needed for calculating cosine similarity.

2.3. Data Augmentation

Data augmentation plays a crucial role in metric learning by creating positive pairs from single images. During training, the following augmentations were applied:

- **Random Resized Crop:** Images were randomly cropped to 80–100% of the original area and resized to 224×224 pixels.
- **Horizontal Flip:** Applied with 50% probability.
- **Color Jitter:** Random adjustments to brightness ($\pm 20\%$), contrast ($\pm 20\%$), saturation ($\pm 20\%$), and hue ($\pm 10\%$).
- **Normalization:** ImageNet mean and standard deviation normalization.

For evaluation, only deterministic center cropping and normalization were applied to ensure reproducible results.

2.4. Experimental Setup

All experiments were conducted on the VU MIF HPC cluster using NVIDIA Tesla V100 GPUs with 32GB memory. The PyTorch Metric Learning library [MBL20] was used for implementing the loss functions and mining strategies. Training parameters are summarized in Table 4.

Table 4. Training hyperparameters

Parameter	Value
Batch size	256 (local) / 1024 (HPC)
Learning rate	10^{-4} (local) / 4×10^{-4} (HPC)
Optimizer	AdamW (weight decay = 0.01)
Learning rate schedule	Cosine annealing with warm restarts
Epochs	20
Embedding dimension	128
Backbone	DINOv2 ViT-S/14 (frozen)
Margin (triplet/contrastive)	0.5
ArcFace scale	64
ArcFace margin	0.5

For contrastive and triplet losses, positive pairs were created by applying data augmentation to the same image twice, producing two different views of the same content. Negative pairs were sampled from different images within the same batch. For triplet loss, hard negative mining was employed to select the most challenging triplets.

For ArcFace loss, each training image was treated as its own class. This approach transforms the similarity learning problem into a classification problem with 50,000 classes, where the angular margin penalty encourages better separation between classes.

2.5. Evaluation Protocol

The evaluation pipeline consists of the following steps:

1. **Feature Extraction:** Extract 128-dimensional embeddings for all query and reference images using the trained model.
2. **Similarity Computation:** Compute cosine similarity between each query embedding and all reference embeddings.
3. **Ranking:** For each query, rank all references by descending similarity score.
4. **Metric Calculation:** Compare rankings against ground truth to compute P@K, mAP, and μ AP.

Evaluation was performed every 5 epochs on the development query set. The model checkpoint with the best mAP score was saved for final evaluation. Feature extraction for 100,000 images (50,000 queries + 50,000 references) took approximately 6 minutes on a V100 GPU.

3. Experiments and Results

This section presents the experimental results comparing the baseline DINOv2 model with models fine-tuned using different loss functions. We analyze the training dynamics, compare final performance metrics, and discuss the implications of our findings.

3.1. Baseline Evaluation

The baseline was established by evaluating the pre-trained DINOv2 model without any fine-tuning. Features were extracted using the frozen backbone followed by the randomly initialized projection head.

The baseline achieves strong performance ($P@1 = 56.71\%$) even without task-specific training, demonstrating that DINOv2’s pre-trained features are already well-suited for image similarity tasks. This strong baseline provides a challenging benchmark for evaluating the effectiveness of fine-tuning approaches.

3.2. Training Dynamics

Figure 8 shows the training loss curves for all three loss functions over 20 epochs. The curves reveal distinctly different learning behaviors.

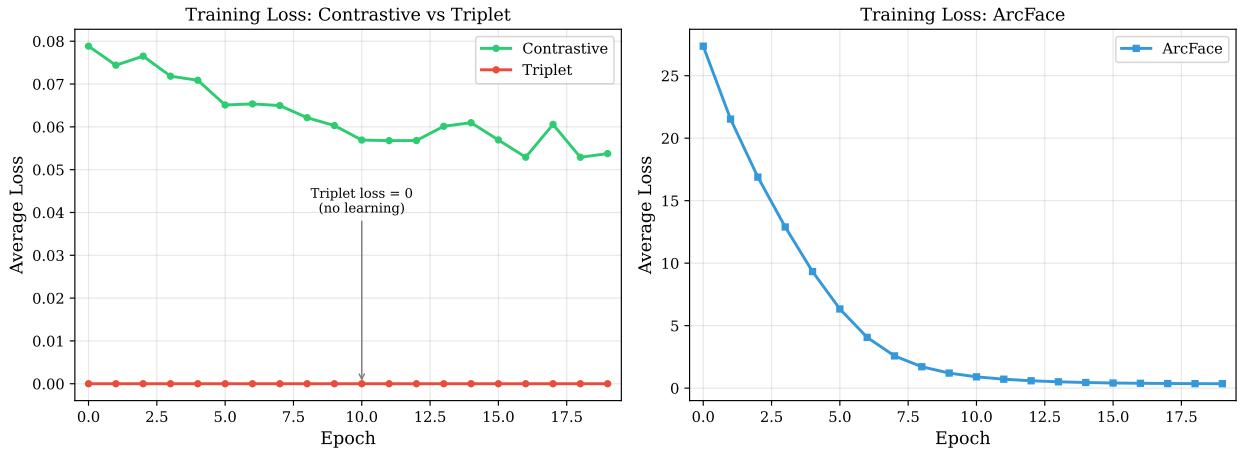


Figure 8. Training loss curves for different loss functions. Left: Contrastive and Triplet losses (same scale). Right: ArcFace loss (different scale due to higher initial values).

Key observations from the training curves:

- **Contrastive loss** shows gradual decrease from 0.079 to 0.054, indicating stable convergence.
- **Triplet loss** remains at exactly 0.0 throughout training—a surprising result analyzed in Section 3.4.
- **ArcFace loss** shows rapid initial decrease from 27.36 to approximately 0.35, indicating strong learning signal.

3.3. Fine-tuning with Contrastive Loss

Training with contrastive loss produced positive pairs by applying random augmentations to the same image, while negative pairs came from different images within the batch. The decreasing loss values indicate that the model learned to distinguish between positive and negative pairs.

However, the resulting model showed mixed performance compared to baseline:

- P@1 decreased by 1.51% ($56.71\% \rightarrow 55.20\%$)
- P@5 decreased by 0.95% ($61.25\% \rightarrow 60.30\%$)
- μAP **improved** by 6.92% ($41.44\% \rightarrow 48.36\%$)

This trade-off suggests that contrastive learning improved the model’s ability to find relevant images overall (higher μAP) but at the cost of placing the exact match at rank 1 (lower P@1). The model may have learned to group similar images together more broadly rather than precisely distinguishing source-copy pairs.

3.4. Fine-tuning with Triplet Loss

An unexpected result was observed with triplet loss training: the loss remained at exactly 0.0 throughout all 20 epochs. This indicates that the hard triplet mining strategy could not find any triplets that violated the margin constraint.

Why did this happen? DINOv2’s pre-trained embeddings are already highly discriminative. When computing triplet distances:

$$d(a, p) + m < d(a, n) \quad (\text{margin constraint already satisfied}) \quad (5)$$

For all potential triplets in the training data, the positive (augmented view of the same image) was already much closer to the anchor than any negative, by more than the margin $m = 0.5$. Since the loss is $\max(0, d(a,p) - d(a,n) + m)$, and $d(a,p) - d(a,n) + m < 0$ for all triplets, the loss was always zero.

Implications: This finding highlights an important consideration when fine-tuning state-of-the-art pre-trained models: the loss function must provide a learning signal beyond what the model has already learned. Triplet loss with standard margins may be insufficient for highly capable backbones.

3.5. Fine-tuning with ArcFace Loss

ArcFace loss showed strong learning dynamics with loss decreasing from 27.36 to 0.35 over 20 epochs. The high initial loss is expected because the model starts with 50,000 classes (one per training image) and random classification weights.

The angular margin approach fundamentally differs from pairwise losses:

1. Each image is treated as a separate class
2. The model learns to project images close to their class center (weight vector)

3. The angular margin ensures large separation between different classes

Results showed consistent improvement across all metrics:

- P@1 improved by 1.70% (56.71% → 58.41%)
- mAP improved by 1.66% (58.80% → 60.46%)
- μ AP improved by 8.60% (41.44% → 50.04%)

3.6. Results Comparison

Table 5 summarizes the performance of all methods on the development query set.

Table 5. Comparison of image similarity methods on DISC21 development set

Method	P@1	P@5	P@10	mAP	μ AP
Baseline (no fine-tuning)	56.71%	61.25%	62.95%	58.80%	41.44%
Contrastive Loss	55.20%	60.30%	62.76%	57.71%	48.36%
Triplet Loss	56.71%	62.19%	64.08%	59.43%	42.21%
ArcFace Loss	58.41%	63.14%	63.71%	60.46%	50.04%

Figure 9 provides a visual comparison of the results.

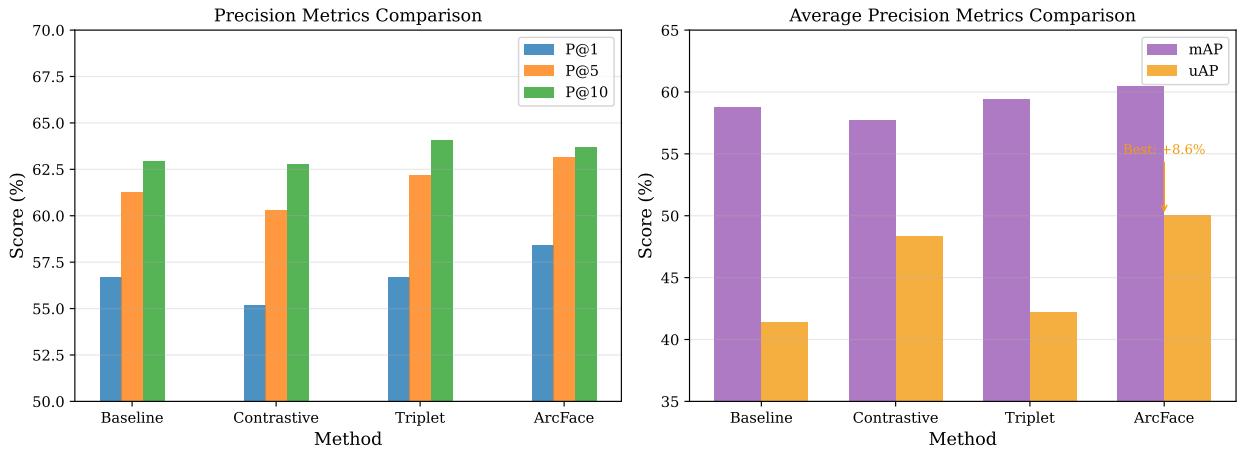


Figure 9. Visual comparison of evaluation metrics. Left: Precision metrics (P@1, P@5, P@10). Right: Average precision metrics (mAP, μ AP).

3.7. Discussion

The experimental results reveal several important findings:

1. **Loss function choice matters significantly.** The three loss functions produced very different outcomes despite using the same backbone and training data. ArcFace consistently outperformed others, while triplet loss failed to provide any learning signal.
2. **Pre-training quality affects fine-tuning strategy.** The failure of triplet loss demonstrates that not all metric learning approaches are suitable for fine-tuning state-of-the-art models. The choice of loss function should consider what the pre-trained model has already learned.

3. Different metrics capture different aspects. Contrastive loss improved μ AP while decreasing P@1, suggesting that optimizing for broad relevance may conflict with precise top-1 accuracy. The choice of evaluation metric should align with the application requirements.

4. Angular margin approaches are robust. ArcFace’s formulation as a classification problem with angular margin provides a consistent learning signal regardless of the pre-training quality, making it a safer choice for fine-tuning.

Table 6 summarizes the changes from baseline for each method.

Table 6. Performance change from baseline (% points)

Method	Δ P@1	Δ P@5	Δ P@10	Δ mAP	Δ μ AP
Contrastive	-1.51	-0.95	-0.19	-1.09	+6.92
Triplet	0.00	+0.94	+1.13	+0.63	+0.77
ArcFace	+1.70	+1.89	+0.76	+1.66	+8.60

Results and Conclusions

This work compared three loss functions—contrastive, triplet, and ArcFace—for fine-tuning a pre-trained DINOv2 model on the image similarity task using the DISC21 dataset.

Main results:

1. **ArcFace loss is the most effective** for fine-tuning DINOv2 on image similarity tasks, achieving improvements of +1.7% in P@1 and +8.6% in μ AP compared to the baseline.
2. **Triplet loss is ineffective** for fine-tuning highly pre-trained models like DINOv2. The model's embeddings already satisfy the triplet constraint, resulting in zero loss and no learning.
3. **Contrastive loss produces mixed results**, improving coverage (μ AP) at the expense of top-rank precision (P@K).
4. **DINOv2's pre-trained features are highly effective** for image similarity even without fine-tuning, achieving P@1 of 56.71%.

Conclusions:

1. For practical image similarity systems using pre-trained Vision Transformers, ArcFace loss is recommended for fine-tuning.
2. Not all metric learning losses are suitable for fine-tuning state-of-the-art pre-trained models—triplet loss may fail when embeddings already satisfy its constraints.
3. The angular margin approach of ArcFace creates better-separated embedding clusters, which is particularly beneficial for retrieval tasks.

Limitations and future work:

- This study used a subset of the DISC21 dataset; full-scale experiments may yield different insights.
- Only one model size (ViT-S) was evaluated; larger variants (ViT-B, ViT-L) may show different behavior.
- Unfreezing the backbone with careful learning rate scheduling could potentially improve results further.
- Combining multiple loss functions or using curriculum learning strategies could be explored.

References

- [CKN⁺20] T. Chen, S. Kornblith, M. Norouzi, G. Hinton. A Simple Framework for Contrastive Learning of Visual Representations. In: *Proceedings of the 37th International Conference on Machine Learning (ICML)*. 2020, pp. 1597–1607.
- [CTM⁺21] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, A. Joulin. Emerging Properties in Self-Supervised Vision Transformers. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 9650–9660.
- [DBK⁺21] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In: *International Conference on Learning Representations (ICLR)*. 2021. Available also from: <https://arxiv.org/abs/2010.11929>.
- [DGY⁺22] J. Deng, J. Guo, J. Yang, N. Xue, I. Kotsia, S. Zafeiriou. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2022, volume 44, number 10, pp. 5962–5979. Available from: <https://doi.org/10.1109/TPAMI.2021.3087709>.
- [DTP⁺21] M. Douze, G. Tolias, E. Pizzi, Z. Papakipos, et al. The 2021 Image Similarity Dataset and Challenge. In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. 2021. Available also from: <https://github.com/facebookresearch/isc2021>.
- [MBL20] K. Musgrave, S. Belongie, S.-N. Lim. *PyTorch Metric Learning*. 2020. Available also from: <https://github.com/KevinMusgrave/pytorch-metric-learning>.
- [ODM⁺24] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, et al. *DINOv2: Learning Robust Visual Features without Supervision*. 2024. Available also from: <https://arxiv.org/abs/2304.07193>.
- [SKP15] F. Schroff, D. Kalenichenko, J. Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 815–823. Available from: <https://doi.org/10.1109/CVPR.2015.7298682>.
- [VSP⁺17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin. Attention Is All You Need. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 5998–6008.

Appendices

1 Detailed Training Logs

This appendix provides the detailed epoch-by-epoch training statistics for each loss function, taken directly from the HPC training logs.

Contrastive Loss Training Log:

Epoch	Avg Loss	Epoch	Avg Loss
0	0.0788	10	0.0569
1	0.0744	11	0.0568
2	0.0765	12	0.0568
3	0.0718	13	0.0601
4	0.0709	14	0.0610
5	0.0651	15	0.0569
6	0.0654	16	0.0529
7	0.0650	17	0.0606
8	0.0621	18	0.0529
9	0.0603	19	0.0537

Table 7. Contrastive loss training progression (actual HPC log data)

Triplet Loss Training Log:

Epoch	Avg Loss	Epoch	Avg Loss
0	0.0	10	0.0
1	0.0	11	0.0
2	0.0	12	0.0
3	0.0	13	0.0
4	0.0	14	0.0
5	0.0	15	0.0
6	0.0	16	0.0
7	0.0	17	0.0
8	0.0	18	0.0
9	0.0	19	0.0

Table 8. Triplet loss training progression—loss remained 0.0 throughout all epochs because the hard triplet miner could not find any triplets violating the margin constraint

ArcFace Loss Training Log:

Epoch	Avg Loss	Epoch	Avg Loss
0	27.36	10	0.90
1	21.53	11	0.72
2	16.88	12	0.59
3	12.90	13	0.51
4	9.33	14	0.45
5	6.33	15	0.41
6	4.06	16	0.39
7	2.58	17	0.37
8	1.71	18	0.36
9	1.20	19	0.35

Table 9. ArcFace loss training progression (actual HPC log data)

2 Experimental Environment

All experiments were conducted using the following hardware and software configuration:

Hardware:

- **HPC Cluster:** VU MIF HPC (Vilnius University)
- **GPU:** NVIDIA Tesla V100 (32GB HBM2)
- **CPU:** Intel Xeon (4 cores allocated per job)
- **RAM:** 64GB per job

Software:

- **Python:** 3.10
- **PyTorch:** 2.0 with CUDA 11.8
- **PyTorch Metric Learning:** 2.3.0
- **DINOv2:** torch.hub (dinov2_vits14)

Training Time:

- Baseline evaluation: ~10 minutes
- Contrastive training (20 epochs): ~45 minutes
- ArcFace training (20 epochs): ~60 minutes
- Triplet training (20 epochs): ~40 minutes