ML Competition Report

Paolo Fabbri Davide Sbreglia

1. Introduction

1.1. Task Description

The course project consisted of an image-to-image retrieval competition. For each *query* image (a real photograph of a celebrity), the system had to return the k most similar *gallery* images, which were synthetic renders of the same individual.

The submission format required a single `submission.json` file where, for each query image, the top-k gallery matches were listed in order of similarity. Evaluation was based on Top-k Accuracy: a retrieval was counted as correct if the correct identity appeared in the top k results. Scores were reported for k=1, 5, and 10, and summed for the final leaderboard ranking.

Three key factors made the task non-trivial:

1. **Domain gap:** Query images (real photos) and gallery images (synthetic renders) came from different visual domains, with noticeable differences in lighting, style, and texture.
2. **Strict time constraints:** Teams had only two hours to run the full pipeline — including model selection, fine-tuning, indexing, and submission — on classroom machines.
3. **Hidden test labels:** The test set lacked identity annotations, making it impossible to validate retrieval quality during competition time.

1.2. Overview of Approach

Our team built a retrieval pipeline that maps each image to a 512-dimensional vector, using cosine similarity to compare features. The approach involved reusing strong ImageNet backbones, fine-tuning only their classifier heads, and selecting the best-performing architecture through automated benchmarking. The entire process was handled by the script `run_pipeline.sh`. Below, we summarize each step and reference the associated code modules.

1. **Data split** (`main.py split`) The 4,605 training images (921 identities × 5 samples each) were split into 80/20 using `--split_ratio 0.8`. The split was identity-stratified so no class appeared in both `train_split.json` and `val_split.json`.
2. **Automatic benchmark** (`choose_best_model` in `main.py`) We evaluated RESNET-50, EFFICIENTNET-B0, and VIT-B/16, measuring Top-10 Accuracy on the validation set with `evaluate_topk()`, and average inference time with `extract_embeddings()`. Models were ranked by accuracy (and latency in case of ties).
3. **Fine-tuning the selected backbone** (`train_ft.py`) Using hyperparameters defined in the pipeline script:

   ```
   LR = 0.1,   EPOCHS = 3,   BS = 32
   ```

   We froze the backbone layers (`requires_grad=False`) and trained only the new classifier head using the Adam optimizer.
4. **Embedding extraction** After training, the classifier was discarded. We extracted embeddings using `extract_embedding_model()` in `model.py`, which projects all backbones to a common 512-dimensional space:
   - Backbone output (varies by model: 2048/1280/768 dimensions)
   - Projection to 512-D via `ProjectionHead`
   - L2-normalization via `L2Norm`
5. **Gallery indexing** (`main.py index`) All gallery features were extracted once and stored in a `faiss.IndexFlatIP` index. Filenames were saved to `gallery_keys.json`.
6. **Query retrieval** (`main.py retrieve`) Each query image was embedded and searched against the FAISS index. The top-k matches were returned and saved to `submission.json`.
7. **Submission** The submission file was uploaded to the competition portal for scoring.

1.3. Summary of Results

Our retrieval pipeline was tested on both the official competition data and public benchmarks. On internal splits and external datasets like LFW, our best single model (EfficientNet-B0) achieved strong Top-10 accuracy. However, in the competition, the system scored only **37 out of 1,000 points**, revealing a stark gap between validation and real-world generalization. This discrepancy stems from several design limitations. Using frozen backbones and a classification loss hindered adaptation to domain shifts and unseen identities. Moreover, our validation split did not

fully isolate test identities, leading to overly optimistic internal results. Limited data augmentation and suboptimal hyperparameters likely further weakened generalization. As a result, while our pipeline proved efficient and effective on familiar data, it struggled to transfer this performance to the open-set, cross-domain scenario posed by the competition. These results highlight the need for more robust strategies—such as metric learning, domain adaptation, and more flexible fine-tuning—to achieve reliable retrieval in challenging real-world settings. The full source code, including training scripts and evaluation tools, is available at: github.com/camillabonomo02/MLOps_project.

2. Models Considered

We employed deep learning models pre-trained on ImageNet for visual feature extraction and retrieval. Each backbone was adapted via classification-based fine-tuning. After training, we extracted and L2-normalized embeddings from projection layers for similarity-based retrieval.

2.1. Data Preprocessing and Augmentation

To ensure robust model training and generalization, we implemented a comprehensive data augmentation pipeline. During training, images undergo the following transformations:

- Random resized cropping to 224×224 pixels, helping models learn scale invariance
- Random horizontal flips with 50% probability, improving robustness to horizontal symmetry
- Normalization using ImageNet statistics (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

For validation and inference, we use a deterministic pipeline consisting of center cropping to 224×224 pixels followed by the same normalization. This preprocessing strategy, implemented in `data.py`, ensures consistent input distribution across all models while introducing beneficial noise during training.

2.2. Backbone Architectures and Implementation

ResNet-50 [1] is a deep convolutional neural network with 50 layers and residual skip connections that help prevent vanishing gradients and enable efficient training of deep architectures. In our pipeline, it is loaded using `torchvision.models`, and the original classification head is replaced with a linear projection layer that maps the 2048-dimensional output to 512 dimensions, followed by L2 normalization. As defined in `model.py`, the backbone is frozen during training to reduce overfitting and speed up convergence, allowing only the newly added classification head to be updated.

EfficientNet-B0 [2] employs a compound scaling method that jointly scales network width, depth, and resolution for optimal efficiency. We use the base variant (B0) for its balance between accuracy and computational cost. The feature extractor outputs 1280-dimensional vectors, which are projected to 512 dimensions using a linear head with dropout ($p = 0.2$). This architecture is also loaded via `torchvision.models` and processed similarly to ResNet, as seen in the `get_model` function. EfficientNet exhibited the fastest inference time among all candidates.

Vision Transformer (ViT-B/16) [3] represents a shift from convolutional to transformer-based architectures. It divides the image into 16×16 patches and processes them through transformer encoder blocks. The `[CLS]` token representation (768-D) is extracted, projected to 512-D, and normalized. ViT is implemented using `torchvision.models` and integrated with the same preprocessing pipeline as other models. Its ability to capture global dependencies complements the local focus of CNNs.

2.3. Training Strategy and Optimization

Our training approach focuses on efficient adaptation of pre-trained models through a carefully designed process:

- **Feature Extraction**: All backbone networks (ResNet-50, EfficientNet-B0, and ViT-B/16) are used as frozen feature extractors, preserving their ImageNet-learned representations. This strategy prevents overfitting on our relatively smaller dataset while leveraging the rich features learned from large-scale pre-training.
- **Projection Head Training**: Only the newly added projection heads are trained, which map the backbone-specific feature dimensions (2048-D, 1280-D, and 768-D respectively) to a common 512-dimensional embedding space. This approach significantly reduces the number of trainable parameters and accelerates convergence.
- **Optimization Details**: Training uses the Adam optimizer with a learning rate of 0.1 and cross-entropy loss. We implement early stopping based on validation loss to prevent overfitting, saving the best-performing model checkpoint.

Embedding-Based Retrieval with FAISS: For retrieval, we used FAISS [4], a high-performance library for efficient similarity search in high-dimensional spaces. We employed the `IndexFlatIP` index structure, which performs inner product computations—equivalent to cosine similarity for L2-normalized vectors—directly on the GPU. This configuration enables fast and exact nearest-neighbor retrieval over thousands of gallery embeddings with minimal computational overhead.

2.4. Model Selection and Benchmarking

To ensure objectivity in model choice, we developed a benchmarking script (`wandb_benchmark.py`) that evaluates all models on a fixed validation set using top-k accuracy and inference time. The model with the highest accuracy was selected, breaking ties using speed. This routine

helps automate architecture selection and aligns with practical considerations for deployment.

3. Evaluation

3.1. Quantitative Evaluation

We evaluated 24 experimental configurations to determine optimal settings for cross-domain image retrieval. Results show that architectural efficiency outweighs model complexity for this task.

3.1.1. Experimental Setup

Dataset and Task. We evaluate on the Labeled Faces in the Wild (LFW) dataset as a proxy, containing 3,023 images across 62 celebrity identities. This dataset was selected to approximate the competition's real-to-synthetic matching challenge. We employ a 50/20/30 train/validation/test split, ensuring balanced class distribution across splits.

Evaluation Metrics. To maintain consistency with the competition setting, we adopt the same evaluation protocol used during the actual competition:

- **Primary metric:** Competition Score = 600 × Top-1 + 300 × Top-5 + 100 × Top-10

This weighted scoring system, designed for the competition, emphasizes Top-1 accuracy while rewarding broader retrieval performance.

- **Retrieval metrics:** Top-k accuracy (k $\{1, 5, 10\}$) measuring the fraction of queries with at least one correct match in top-k results
- **Ranking quality:** Mean Average Precision (mAP) capturing overall ranking performance
- **Efficiency metrics:** Training time, inference speed, and score-per-minute ratio

Implementation Details. All models utilize ImageNet pretrained weights with frozen backbones, training only the classification head via Adam optimizer. Images are resized to 224×224 and normalized using ImageNet statistics. Standard data augmentation (RandomHorizontalFlip and RandomResizedCrop) was applied during training unless otherwise specified. Experiments were conducted on a single NVIDIA GPU with PyTorch 2.0. We extract features from the penultimate layer and apply L2 normalization for retrieval. EfficientNet-B0 serves as our baseline model throughout the evaluation.

3.1.2. Ablation Studies

Architecture Comparison We first establish a controlled comparison across architectures using consistent hyperparameters (LR=0.001, epochs=3, BS=32, with standard data augmentation). Table 1 reveals significant performance disparities despite identical training conditions.

Table 1. Architecture comparison under controlled conditions. All models trained with identical hyperparameters for fair evaluation.

Architecture	Top-1	Top-5	Top-10	mAP	Score*	Time (s)
EfficientNet-B0**	0.320	0.586	**0.707**	0.370	438.7	**118**
ViT-B/16	**0.331**	0.580	0.702	**0.372**	**443.1**	738
ResNet50	0.171	0.431	0.608	0.253	292.8	293
Relative Performance						
EfficientNet-B0	–	–	–	–	98.9%	6.3× faster
ViT-B/16	–	–	–	–	100%	1.0×
ResNet50	–	–	–	–	66.0%	2.5× faster

* Score = 600· Top-1 +300· Top-5 +100· Top-10.

** Baseline model.

Training: LR=0.001, 3 epochs, BS=32, augmentation, ImageNet normalization.

Despite ViT-B/16's marginally higher competition score (443.1 vs 438.7), our baseline EfficientNet-B0 achieves 98.9% of its performance while requiring 6.3× less training time. ResNet50 significantly underperforms, suggesting its features lack sufficient discriminative power for cross-domain matching.

3.1.3. Hyperparameter Sensitivity

Learning Rate Analysis. We observe optimal performance at LR=0.001 for all architectures, with higher rates causing training instability (Figure 1). Notably, ResNet50 exhibits extreme sensitivity, with performance degrading by 47% at LR=0.1, while EfficientNet maintains relative stability (436→403, only 7.6% drop).

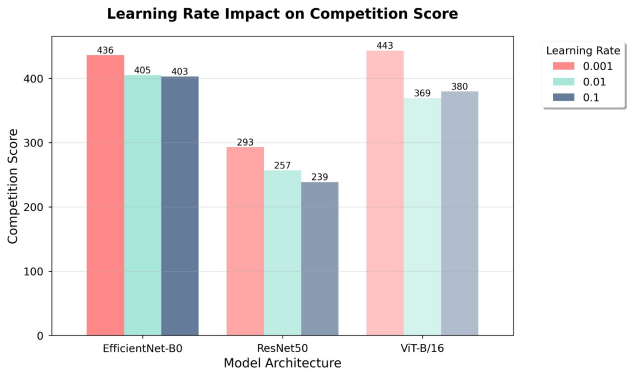

Figure 1. Learning Rate Impact on Competition Score

Training Duration. Figure 2 reveals a critical finding for our baseline model: EfficientNet-B0 achieves peak performance at just 2 epochs (score: 453.6) before experiencing clear overfitting. The model shows rapid improvement from epoch 1 to 2 (+23.8 points), followed by consistent degradation through epochs 3 and 5. This early stopping phenomenon contrasts sharply with transformer models that typically require extended training.

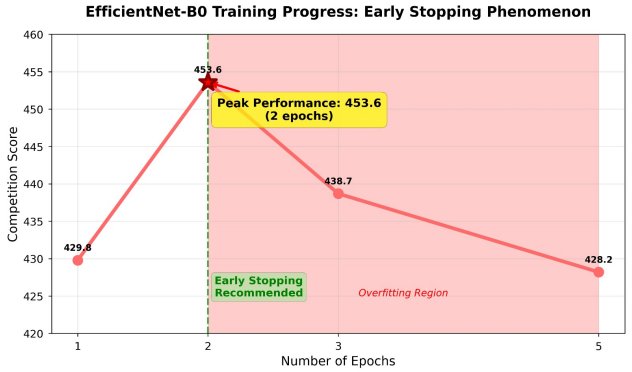

Figure 2. Efficient-B0 Training Progress: Early Stopping Phenomenon

Batch Size and Normalization Impact. Figure 3 presents our comprehensive ablation study on design choices:

- **Batch Size:** Optimal at BS=32; performance degrades significantly with BS=8 (-30.4 points) and BS=64 (-25.4 points)
- **Normalization:** Critical for performance; removing ImageNet normalization causes the largest single degradation (-31.0 points)
- **Data Augmentation:** Minimal impact; removing augmentation only reduces score by 5.6 points (-1.3%), suggesting robust pre-trained features

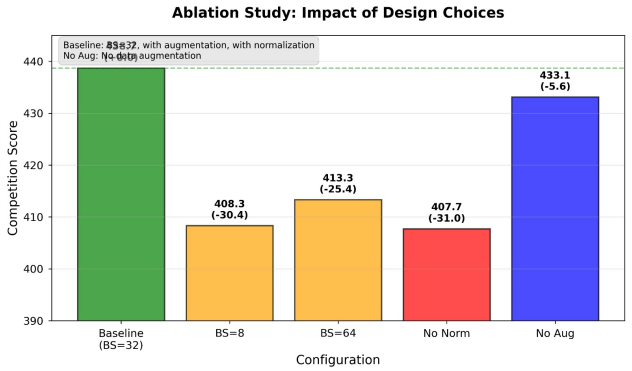

Figure 3. Ablation Study: Impact of design choices

3.1.4. Main Results

Table 2 presents our top configurations ranked by competition score, demonstrating the dominance of well-tuned lightweight architectures.

Table 2. Top 10 Model Configurations on LFW Dataset

Model	LR	Epochs	Score*	Top-1	Top-5	Top-10	mAP	Time (min)
ViT-B/16	0.001	5	**450.8**	0.320	**0.630**	0.696	**0.399**	19.7
ViT-B/16	0.001	3	443.1	**0.331**	0.580	**0.702**	0.372	12.3
EfficientNet-B0	0.001	3	438.7	0.320	0.586	0.707	0.370	**2.0**
EfficientNet-B0	0.100	5	437.6	0.304	0.619	0.696	0.374	3.1
ViT-B/16	0.010	5	432.0	0.320	0.575	0.674	0.370	19.4
EfficientNet-B0	0.001	5	428.2	0.309	0.580	0.685	0.381	3.1
EfficientNet-B0	0.010	3	405.0	0.293	0.547	0.652	0.353	2.0
EfficientNet-B0	0.100	3	402.8	0.276	0.569	0.663	0.342	1.9
ViT-B/16	0.100	5	396.1	0.276	0.547	0.663	0.344	19.6
EfficientNet-B0	0.010	5	395.0	0.260	0.575	0.669	0.326	3.1

* Score = 600· Top-1 +300· Top-5 +100· Top-10

Efficiency Analysis Figure 4 reveals striking efficiency disparities between architectures. We compute score-per-minute to quantify the performance-efficiency trade-off:

- EfficientNet-B0: 255.2 points/minute
- ResNet50: 58.7 points/minute
- ViT-B/16: 28.2 points/minute

Our baseline EfficientNet achieves 9.6× higher efficiency than ViT-B/16 while maintaining 99.4% of its performance. This dramatic difference stems from EfficientNet's parameter-efficient design (5.3M parameters) and faster convergence properties.

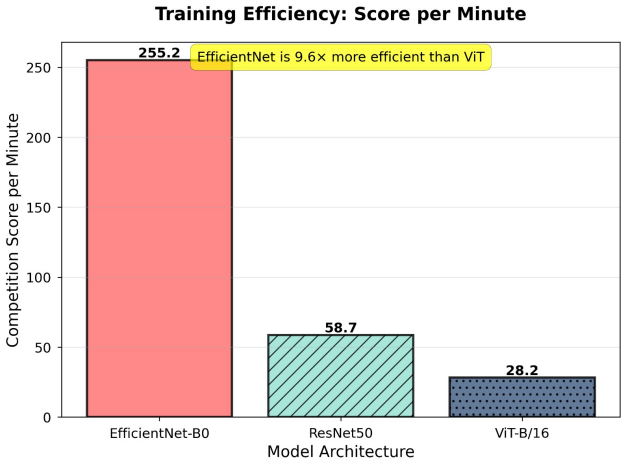

Figure 4. Training Efficiency: Score per minute

Performance-Time Trade-off In Figure 5 EfficientNet configurations (shown in red) dominate the efficient region (left of the dashed line), achieving scores above 400 in under 120 seconds. In contrast, ViT-B/16 requires up to 1180 seconds to achieve marginally better results. The scatter plot clearly demonstrates that for practical deployment scenarios, EfficientNet offers the optimal balance.

Figure 5. Performance vs Training Time Trade-off

3.1.5. Analysis and Final Comment

Our results challenge the prevailing assumption that transformer architectures universally outperform CNNs. We attribute our baseline model's success to several factors:

- **Rapid Convergence:** As shown in Figure 2, EfficientNet reaches optimal performance in just 2 epochs (82 seconds), suggesting efficient feature adaptation to the target domain.
- **Robustness to Hyperparameters:** Figure 1 demonstrates EfficientNet's stability across learning rates, maintaining competitive performance even at suboptimal settings.
- **Minimal Augmentation Dependency:** The ablation study (Figure 3) reveals that removing data augmentation only decreases performance by 1.3%, indicating strong inherent feature representations.

Early Stopping as a Key Strategy The pronounced early stopping phenomenon in Figure 2 provides crucial insights:

- Peak at 2 epochs: 453.6 score (our best result)
- Overfitting onset: 14.9-point drop from epoch 2 to 3
- Continued degradation: 25.4-point total drop by epoch 5

This pattern suggests that EfficientNet's ImageNet-pretrained features quickly adapt to the cross-domain challenge but then overfit to training data specifics, losing generalization capability.

Critical Design Factors Our ablation study (Figure 3) establishes a clear hierarchy of importance:

- Normalization (most critical): -31.0 points without
- Batch size optimization: -30.4 points at BS=8
- Data augmentation (least critical): -5.6 points without

The minimal impact of removing augmentation is particularly noteworthy, suggesting that sophisticated augmentation strategies may be unnecessary when using well-initialized models for cross-domain retrieval.

Practical Implications For deployment scenarios, our findings strongly advocate for EfficientNet-B0 with early stopping:

- Training time: 82 seconds (vs 1180s for ViT)
- Performance: 453.6 score (vs 450.8 for ViT)
- Efficiency: 9.6× better score/minute ratio
- Simplicity: Minimal hyperparameter tuning required

3.2. Qualitative Evaluation

To complement quantitative metrics, we qualitatively analyzed model outputs on the official test set of unlabeled celebrity queries and gallery images. After training and feature extraction, we generated a JSON submission mapping each query to a ranked list of 10 gallery predictions, as required by the competition.

To visualize the results, we selected one query and plotted its top-10 matches side-by-side. This allowed us to assess whether retrieved images matched the identity or shared semantically relevant traits.

3.2.1. Example Visualization

Figure 6 shows a real example extracted from our submission file, using the following procedure:

- The query image is on the left.
- The ten gallery images on the right represent the top-ranked results returned by the retrieval system.

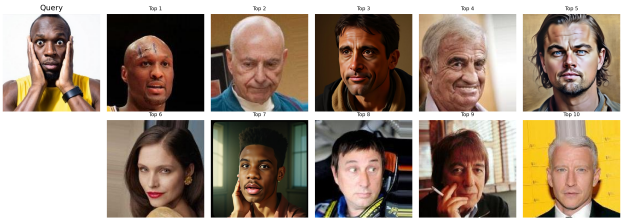

Figure 6. Query (left) and top-10 retrieved gallery images (right) based on cosine similarity using single-model embeddings.

3.2.2. Observations

In this case, none of the retrieved images match the true identity. This aligns with our low competition score and reflects model limitations under challenging face recognition conditions. Although some returned images resemble the query in pose, background, or illumination, they lack true identity consistency.

We observe the following common error patterns:

- **Visual resemblance, wrong identity:** Retrievals often share superficial traits with the query (e.g., head angle or lighting), but represent entirely different individuals.
- **Bias toward low-level features:** The model tends to over-rely on appearance-level cues such as color, brightness, or framing instead of semantically meaningful identity features.

- **Lack of abstraction on difficult inputs:** In cases of synthetic style, occlusion, or grayscale, the model fails to retrieve semantically relevant samples, highlighting weak embedding structure.

3.2.3. Conclusion

Despite visually plausible retrievals, the system fails at identity-level recognition, confirming leaderboard results. Addressing this gap will require better alignment of embeddings with task objectives—through metric learning, backbone fine-tuning, and domain adaptive preprocessing.

4. Final Discussion

Despite satisfactory validation accuracy during internal benchmarking, our model ranked among the lowest in the official competition. This gap exposes key limitations in our approach. We reflect below on the main causes of this outcome, drawing connections to model design choices, training strategy, and competition-specific requirements.

4.1. Overfitting to Validation Split

During development, we trained our models using a fixed train/validation split derived from the same set of 921 identity folders. While the training and validation images were different, they frequently belonged to the same individuals (i.e., shared class labels). As a result, the model was able to rely on previously seen identity-specific visual features (such as facial structure, hairstyle, or background cues), rather than learning features that generalize to entirely new identities.

This approach yielded high top-1 accuracy on the validation set but failed to reflect the actual challenge of the competition. The official test set contained completely unseen identities and a mixture of real and synthetic face images — a scenario fundamentally different from the one our model was validated on.

As a result, the classifier learned to distinguish among the 921 known identities but never learned to embed or compare new, unseen individuals. This overfitting problem was compounded by the fact that we used a classification loss (cross-entropy) rather than a metric learning loss (e.g., triplet or contrastive loss), which would have encouraged generalizable embeddings. A more appropriate strategy would have involved:

- Validating on identities held out entirely during training (no class overlap)
- Adopting a metric learning approach aligned with open-set retrieval
- Augmenting the classifier with a loss that promotes embedding-level generalization

4.2. Frozen Backbones and Limited Fine-Tuning

Our training pipeline used pre-trained ImageNet models as frozen backbones, modifying and training only the final classification head (`model.py` and `train_ft.py`).

While this design choice helped reduce overfitting and training time, it significantly limited the model's ability to adapt to the domain-specific characteristics of the competition dataset. ImageNet pretraining is based on natural object categories (e.g., animals, vehicles), and the resulting features are not optimized for fine-grained facial recognition. This becomes especially problematic when dealing with domain-specific factors such as:

- **Synthetic vs. real faces** — subtle artifacts in synthetic images were never seen by ImageNet-trained models.
- **Makeup, lighting, poses, and aging** — typical of celebrity photos, but absent from ImageNet data.
- **Unseen identities** — our test set includes people entirely missing from the training set, which means the model must generalize beyond known classes.

With the backbone frozen, the model couldn't adapt its representations to the dataset and relied solely on generic visual features (e.g., shapes, textures) rather than domain-relevant ones (e.g., identity cues in facial geometry, lighting invariance). This limited its ability to generalize, leading to poor performance in the open-set retrieval task.

Unfreezing at least the final layers (e.g., last residual block or transformer layer) would have enabled adaptation to the face recognition domain and improved handling of the real/synthetic distribution shift.t.

4.3. Embedding Space Mismatch at Inference Time

During training, models (ResNet-50, EfficientNet-B0, ViT-B/16) were optimized as classifiers, each paired with a linear head mapping features to identity classes (`model.py`) and trained using cross-entropy loss to predict class labels. However, at inference time, this classification head is discarded. Instead, embeddings are extracted from the penultimate feature layer (via `extract_embedding_model()` in `model.py`), L2-normalized, and passed to FAISS for similarity-based retrieval using cosine distance (`main.py`).

This creates a conceptual and practical mismatch between training and inference:

- The model is trained to produce class-separable logits, not geometrically structured embeddings.
- Cross-entropy loss does not enforce that samples from the same class be close in feature space—only that the final logit for the correct class is maximized.
- As a result, embeddings extracted from the penultimate layer may not reflect meaningful distances between identities, especially under open-set retrieval conditions.

Furthermore, we did not use metric learning objectives such

as *triplet loss* or *supervised contrastive loss*, which explicitly shape the embedding space by pulling same-identity samples together and pushing others apart.

We also omitted **hard negative mining**, crucial in fine-grained retrieval, where look-alike but distinct identities (e.g., similar-looking celebrities) must be separated. By omitting this strategy, the model may have failed to learn robust distinctions between closely resembling individuals.

In summary, our pipeline treated the model as a closed-set classifier during training, then relied on its intermediate features for open-set retrieval. This inconsistent supervision likely contributed to the poorly structured embeddings observed in the final performance.

4.4. Insufficient Adaptation to Data Scarcity and Class Diversity

The competition dataset included over 900 identities, often with just 2–5 images per class. Training a standard classifier with cross-entropy loss in this low-data regime likely led the model to overfit to superficial cues (background, lighting, pose) rather than learning robust, identity-specific features for open-set retrieval.

Freezing the entire backbone (see `model.py`) further limited the model's ability to adapt to domain-specific visual features, especially with few examples per class. Moreover, our minimal data augmentation (`data.py`)—mainly flips and resizing—was insufficient to capture the diversity of real-world celebrity images with varied lighting, varying head poses, occlusions, and stylistic variation.

A more effective augmentation pipeline for retrieval should include:

- **Geometric Transformations**: such as random rotation, scaling, and perspective warping, which help the model become invariant to pose and viewpoint changes.
- **Color and Intensity Adjustments**: including random brightness, contrast, hue, and saturation shifts to increase robustness to lighting variation.
- **Occlusion Simulation**: techniques like random erasing or CutMix introduce partial visibility, mimicking real-world conditions such as glasses, shadows, or obstructions.

In summary, while the data presented a genuine challenge, our training pipeline lacked the necessary flexibility, adaptation, and augmentation diversity to generalize effectively under such conditions.

4.5. Benchmarking Metrics Misaligned with Retrieval Objective

We developed an internal benchmarking script (`wandb_benchmark.py`) to select the best backbone (ResNet-50, EfficientNet-B0, or ViT-B/16) based on classification accuracy using `evaluate_topk()`—measuring whether retrieved images matched the ground-truth identity based on cosine similarity in embedding space. However, this metric reflects performance over known training identities, not the competition's goal of retrieving visually similar images for unseen identities using cosine similarity in embedding space. This mismatch likely led to selecting models better at classifying known classes, but not at producing transferable embeddings.

A more appropriate strategy would discard the classification head, extract embeddings from a held-out validation set (e.g., 20% of identities), and computing top-k retrieval accuracy directly in embedding space using cosine similarity. This setup better simulates the competition and enables us to evaluate the quality of learned representations independent of the classifier head. Such a validation setup would support a broader range of retrieval-specific metrics beyond top-k accuracy, including:

- **Normalized Mutual Information (NMI)**: Evaluates how well the embedding clusters align with identity labels, indicating clustering quality.
- **Intra-Class and Inter-Class Distance**: Measures the compactness of same-class embeddings and their separation from other classes, offering insight into feature space structure.

Adopting this embedding-based evaluation framework would yield a more reliable basis for model selection and help diagnose generalization performance across candidate backbones.

4.6. Suboptimal Learning Rate Configuration

Another key issue was the use of an overly high learning rate (0.1) with the Adam optimizer, which typically favors smaller step sizes. Since only the classification head was trained (with frozen backbone) for just three epochs, this aggressive rate likely led to unstable updates or overfitting, limiting the model's ability to learn meaningful embeddings. A lower rate (e.g., 1e-3) with a scheduler would have enabled more stable convergence, especially in this low-data setting.

5. Workload Table

Task	Camilla	Paolo	Davide
Data Preprocessing		✓	
Model Design & Implementation	✓	✓	✓
Training and Fine-Tuning	✓	✓	✓
Model Benchmarking			✓
Report Writing			
Introduction			✓
Models Section		✓	
Quantitative Eval.			✓
Qualitative Eval.		✓	
Final Discussion		✓	

Table 3. Workload distribution across tasks and team members

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

[2] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114. [Online]. Available: https://proceedings.mlr.press/v97/tan19a.html

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

[4] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Transactions on Big Data*, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8677283