Deep Metric Learning for Pet Breed Recognition – Samuel Vierny

1. Introduction

This report details my exploration into Deep Metric Learning (DML) using the Oxford-IIIT Pet Dataset[1]. My goal previously was to only apply simple classification, however now I want to train a model that understands visual similarity and can embed images such that pets of the same breed are grouped closely in the feature space and different breeds are pushed apart. This journey involved setting up a DML pipeline and experimenting with various loss functions, optimizing the training process, and evaluating the resulting embeddings through verification, retrieval, and few-shot learning tasks. I wanted to report my findings as I experimented to ensure version control and reproducibility. I documented the challenges, fixes, and insights I gained along the way.

2. Methodology: Building the Metric Learning Pipeline

My initial setup involved enabling different ResNet CNN backbones with a projection head. The core of DML lies in the type of loss function I choose, which should dictate the embedding process. I experimented with three common DML loss functions: Triplet Loss, Contrastive Loss, and ArcFace[2].

Initial Pipeline & Challenges:

My first steps involved configuring my old pipeline that I used for transfer learning. Eventually after common debugging and function/class configuration, the rough initial pipeline was created.

A first issue involved the environment and base code. I encountered initial setup issues, like missing requirements and TensorFlow oneDNN messages. I addressed these by creating a requirements file and adding code to silence TensorFlow messages.

Then I ran my first run using triplet loss! Unfortunately, I observed training and validation losses quickly dropping to zero, indicating a problem. This was traced back to several issues in the core pipeline and code:

- There was an incorrect handling of edge cases where no valid positive or negative pairs existed in a batch
- Gradient disconnection when handling these cases
- Problematic masking operations
- Incorrect model weight initialization

I rewrote the triplet loss function to handle edge cases correctly and ensured the gradient flow using torch.sum(triplet_loss) * 0 for zero-loss cases. I next used stable distance calculations, fix mining strategy bugs, and ensure proper data handling (tensor-typed labels, which would handle any corrupted images).

Configuration management was another hurdle. The initial script structure overrode configuration file settings with command-line parser defaults. I fixed this by setting parser defaults to None and only updating config values if a command-line argument was explicitly provided. This meant the config file was the primary source unless overridden. I added special handling for boolean flags like --pretrained.

Dataset:

As I previously worked on and am familiar with the Oxford-IIIT Pet Dataset, I used it and split it up into into training, validation, and test sets.

I apply resizing, random horizontal flips, 10-degree rotation, and color jittering to my training images. I maintained only resizing and normalization for validation and testing.

For few-shot learning evaluation, specific breeds were held out. Initial runs revealed issues with correctly identifying all specified holdout breeds due to naming inconsistencies and extraction logic. I realized I had the names of some breeds in the extraction logic to handle simple (e.g., 'Siamese') and not names with an underscore properly (e.g., 'american_bulldog') correctly. I fixed this iteratively with each run, by run 4 and 5 we had most of the few-shot cases fetched.

3. Experimental Runs and Analysis

I deployed several different runs. I adjusted configurations and analyzed results from the validation metrics. These final metrics from each run were each time from the final evaluation on the test set. I shall evaluate the models using ROC-AUC and Equal Error Rate (EER).

Run 1: Baseline Triplet Loss (ResNet18)

- **Config:** ResNet18, Embedding Size 128, Batch Size 64, LR 0.0001, Triplet Loss (margin 0.3), Batch Hard Mining.
- **Training:** 3 epochs. Best validation loss 0.2897 at epoch 1. Total time ~18 mins.
- **Evaluation:** ROC AUC 0.9795, EER 0.0729, P@1 0.8398, R@10 0.9700. Few-shot (2-way 5-shot due to limited classes found initially): 1.0000 accuracy.

- **Analysis:** This was a strong start. I was happy to see a good verification and retrieval performance so early on already, although we only had 2/5 from the few-shot for this one.

Run 2: Contrastive Loss (ResNet18)

- **Change:** Switched to Contrastive Loss (margin 0.3). Config otherwise similar.
- **Training:** 3 epochs. Best validation loss 0.0559 at epoch 3. Total time ~12 mins.
- **Evaluation:** ROC AUC 0.9536, EER 0.1017, P@1 0.6738, R@10 0.9313. Few-shot (2-way 5-shot): 0.9985 accuracy.
- **Analysis:** Numerically lower loss values than Triplet, but significantly worse performance on verification and retrieval metrics. This highlighted that absolute loss values across different function types are not directly comparable due to scale and objective differences between the loss functions and how they work.

Run 3: ArcFace Loss (ResNet18)

- **Change:** Switched to ArcFace Loss. Config otherwise similar. Dataset split slightly different due to reprocessing.
- **Training:** 3 epochs. Best validation loss 3.5520 at epoch 3. Total time ~12 mins. Note: ArcFace loss values operate on a different scale.
- **Evaluation:** ROC AUC 0.9806, EER 0.0575, P@1 0.8999, R@10 0.9624. Few-shot (1-way 5-shot due to only 1 holdout class found): 1.0000 accuracy.
- **Analysis:** ArcFace significantly outperformed Contrastive loss and showed improvements over Triplet loss in most metrics, particularly EER and Precision@1, although I had to bear in mind that only 1/5 from the few-shot cases were used. This is likely due to ArcFace's angular margin optimization providing better class separation. I also resolved an OpenMP library conflict warning during this phase by setting an environment variable in the main.py file.

It became clear that ArcFace was superior for this task because it works in angular space rather than Euclidean space. It seems to learn optimal class representations through class-center learning and creates more discriminative boundaries between classes through the angular margin that it uses. Despite having numerically higher loss values, ArcFace yielded better practical performance on downstream tasks than both Triplet and Contrastive losses. For Arcface, I will consistently use hard negative mining throughout all runs, to ensure best results and stabilize analysis for other variables and parameters.

Run 4: ArcFace, Increased Embedding Size & Epochs (ResNet18)

- **Change:** Increased Embedding Size to 256, Epochs to 5. Resolved few-shot holdout breed extraction, finding 4 breeds.
- **Training:** 5 epochs. Best validation loss 3.6377 at epoch 5. Total time ~21 mins.
- **Evaluation:** ROC AUC 0.9879, EER 0.0433, P@1 0.9150, R@10 0.9621. Few-shot (4-way 5-shot): 0.9730 accuracy.
- **Analysis:** Significant improvements across all metrics. The larger embedding size and extra epochs boosted performance. Slight signs of validation loss plateauing indicated potential early overfitting. Considering this I decided regularization or LR scheduling could help.

Run 5: Introducing Regularization & LR Scheduling (ResNet18)

- **Change:** Added Dropout (0.2) and Warmup Cosine LR Scheduler. Increased Epochs to 8. Fixed few-shot setup to find all 5 holdout breeds. Addressed minor coding issues like missing imports and methods in the trainer.py after integrating these features.
- **Training:** 8 epochs. Final validation loss 2.9014. LR decreased from 1e-4 to 1e-6. Total time ~34 mins.
- **Evaluation:** ROC AUC 0.9923, EER 0.0326, P@1 0.9311, R@10 0.9718. Few-shot (5-way 5-shot): 0.9816 accuracy.
- **Analysis:** Further significant gains across all metrics. Dropout and the scheduler effectively mitigated overfitting despite longer training, leading to better generalization. The model achieved excellent few-shot performance, demonstrating strong generalization.

Run 6: Final Model - ResNet50 Backbone (ArcFace)

- **Change:** Switched to ResNet50 backbone. Increased Epochs to 11. Kept Dropout (0.2) and Warmup Cosine LR Scheduler.
- **Training:** 11 epochs. Final validation loss 1.4952. Total time ~60 mins.
- **Evaluation:** ROC AUC 0.9937, EER 0.0217, P@1 0.9562, R@10 0.9781. Few-shot (5-way 5-shot): 0.9980 accuracy
- **Analysis:** This was our best performing model, achieving near-perfect few-shot accuracy and P@1 > 95%. The deeper ResNet50 architecture, combined with optimized training, clearly captured more complex features, justifying the increased computational cost. The validation loss was significantly lower than ResNet18's best.

4. Evaluation Tasks

Verification: My goal was to predict if two images were of the same breed. My best model (ResNet50, Run 6) achieved a ROC AUC of 0.9937 and an EER of 0.0217. This was very high accuracy in distinguishing same-breed vs. different-breed pairs.

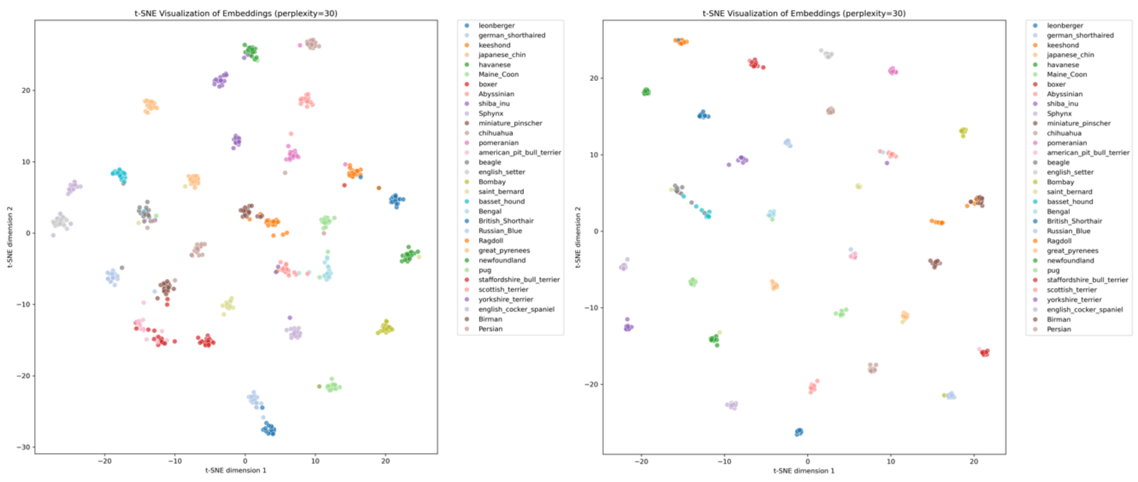
Retrieval: Given a query image, I retrieved the most similar images. The ResNet50 model achieved Precision@1 of 0.9562 and Recall@10 of 0.9781. This means over 95% of the time, the top retrieved image was the correct breed, and nearly 98% of relevant images were found within the top 10 results.

Few-Shot Classification: This tested the model's ability to classify breeds it wasn't explicitly trained on using only a few examples (5-shot) of the new breeds. My final ResNet50 model achieved 99.80% accuracy on a 5-way 5-shot task, demonstrating remarkable generalization capabilities learned through metric learning.

Baseline Comparison: My previous work using standard transfer learning (ResNet50 fine-tuned for classification) achieved a strong test accuracy of 91.85%. While direct comparison is complex (classification vs. embedding evaluation), the DML approach, particularly with ResNet50 and ArcFace, yielded superior results on verification, retrieval, and especially few-shot tasks (EER 0.0217, P@1 0.9562, Few-shot 99.80%). This really does demonstrate the power and effectiveness in learning fine-grained, generalizable feature representations.

5. Visualizations and Interpretability

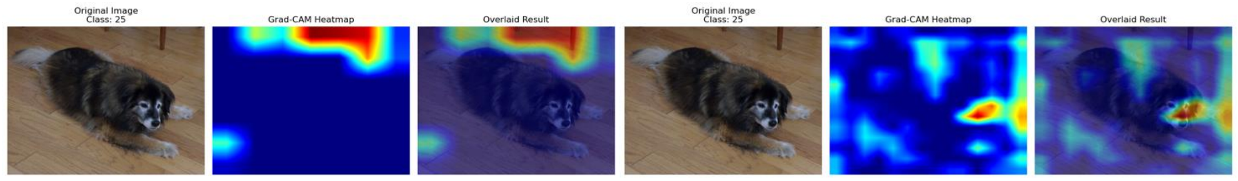
Embedding space for breeds



Left shows our 5th run with Resnet18 Arcface. The right shows 6th run with Resnet50 Arcface (our best model)

Embedding Space: I visualized the test set embeddings using t-SNE and UMAP to qualitatively assess breed clustering. These visualizations confirmed that the ArcFace loss successfully separated the breeds in the embedding space. The visualization script initially had issues (missing main function call, class name mismatch, model loading errors, lack of logging). I fixed these though and eventually produced the plots. Further refinements silenced warnings and improved error handling.

Grad-CAM results



Left shows our 6th run with Resnet50 Arcface (our best model). The right shows a control run ran afterward, using Resnet18 again.

Grad-CAM Analysis: I used Grad-CAM[3] to understand feature attention in my ResNet50 model. Surprisingly, the high-performing model often focused on background elements or seemingly irrelevant objects (like the chair legs) rather than solely on the pet. This suggested potential "shortcut learning," whereby the model exploits spurious correlations. I confirmed this wasn't data leakage.

To investigate further, I ran a controlled experiment with ResNet18 for 4 epochs. This model achieved strong results (ROC AUC 0.9923, EER 0.0312, P@1 0.9233, R@10 0.9844, few-shot accuracy 0.9934). More importantly, the Grad-CAM visualizations showed a marked difference: ResNet18 focused much more clearly on the actual pets (faces, bodies), with broader, more diffuse activation patterns and less fixation on background elements. In contrast, ResNet50 showed more concentrated hotspots, sometimes on seemingly irrelevant features, yet achieved better metrics.

This comparison revealed that the deeper network exhibited potentially misleading feature associations when visualized through GradCAM. I learnt that GradCAM operates at the final network layers. While ResNet50 achieved superior metrics, its deeper architecture led to attention maps that diverged from expected animal-focused regions. In contrast, ResNet18's shallower architecture produced more interpretable visualizations concentrated on the animals themselves. This highlights a tradeoff between performance metrics and the alignment of model attention with human intuition.

6. Conclusion and Future Directions

Through systematic experimentation, I found ArcFace loss combined with a ResNet50 backbone, optimized with dropout and learning rate scheduling, returned state-of-the-art results on verification, retrieval, and few-shot learning tasks.

Key Learnings:

- ArcFace outperformed Triplet and Contrastive loss for this task
- Deeper backbones (ResNet50) provide significant performance gains over shallower ones (ResNet18)
- LR scheduling and regularization are crucial for stable training and preventing overfitting in DML
- High-performing models might learn "shortcuts," highlighting a potential tradeoff between accuracy and interpretability, as revealed by Grad-CAM

Limitations & Future Work:

The potential reliance on background cues (shortcut learning[4]) is a key limitation needing further investigation through robustness tests (e.g., background manipulation). Other future directions therefore might include:

- Exploring more advanced data augmentation to mitigate shortcut learning
- Multi-level feature analysis
- Cross-dataset evaluation
- Model distillation for efficiency
- Further optimization of mining strategies
- I can vary our K more for P@K and R@K. I only use P@1 and R@10 in our tests for consistency, to analyze other changes that should be applied for better results.

Overall, the developed DML models provide a strong foundation for applications requiring nuanced visual similarity understanding.

Bibliography:

[1] ‘The Oxford-IIIT Pet Dataset’. Accessed: Apr. 18, 2025. [Online]. Available: https://www.kaggle.com/datasets/tanlikesmath/the-oxfordiiit-pet-dataset

[2] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, ‘ArcFace: Additive Angular Margin Loss for Deep Face Recognition’, presented at the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 4690–4699. Accessed: Apr. 18, 2025. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2019/html/Deng_ArcFace_Additive_Angular_Margin_Loss_for_Deep_Face_Recognition_CVPR_2019_paper.html

[3] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, ‘Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization’, *Int. J. Comput. Vis.*, vol. 128, no. 2, pp. 336–359, Feb. 2020, doi: 10.1007/s11263-019-01228-7.

[4] R. Geirhos *et al.*, ‘Shortcut learning in deep neural networks’, *Nat. Mach. Intell.*, vol. 2, no. 11, pp. 665–673, Nov. 2020, doi: 10.1038/s42256-020-00257-z.