Image Retrieval with Contrastive Learning

COMP 6831 - Applied Machine Learning, Fall 2025

Philippe Laporte ID 24172957

To get situated I reviewed the basic theoretical concepts regarding shallow CNNs and triplets from the extended abstract available from [1], with the standard PyTorch implementation at [2].

As stated in [1]:

> Learning with triplets involves training from samples of the form $\{\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{n}\}$, where $\boldsymbol{a}$ is the *anchor*, $\boldsymbol{p}$ is a *positive* example, which is a different sample of the same class as $\boldsymbol{a}$, and $\boldsymbol{n}$ is a *negative* example, belonging to a different class than $\boldsymbol{a}$. In our case, $\boldsymbol{a}$ and $\boldsymbol{p}$ are different viewpoints of the same physical point, and $\boldsymbol{n}$ comes from a different keypoint. The goal is to learn the embedding $f(\boldsymbol{x})$ s.t. $\delta_+ = ||f(\boldsymbol{a}) - f(\boldsymbol{p})||_2$ is low (i.e., the network brings $\boldsymbol{a}$ and $\boldsymbol{p}$ close in the feature space) and $\delta_- = ||f(\boldsymbol{a}) - f(\boldsymbol{n})||_2$ is high (i.e., the network pushes the descriptors of $\boldsymbol{a}$ and $\boldsymbol{n}$ far apart). With this aim, we examine two different loss functions

Dataset

I started out with the CNIST-10 dataset to come up with the basic structure.

I then chose the Stanford Online Products (SOP) dataset, a major benchmark used for image retrieval and metric learning [4]. It contains 120,053 images of online products sourced from eBay and Amazon, categorized into 22,634 unique classes. A class represents a unique product instance (e.g., a specific model of headphones), not just a product type.

I used a subset of 3000 samples for training and the same number for the evaluation to reflect the even split between training and test images in the entire set.

In this dataset, images belonging to the same product class often differ wildly due to "spurious attributes":

- Viewpoint: A shoe viewed from the side vs. from the top.
- Lighting/Color Balance: Product shot indoors vs. outdoors.
- Background: Product on a clean white background vs. a cluttered retail shelf.

The model's objective is to be invariant to these spurious differences.

Pipeline

The standard ResNet-18 model, as loaded from torchvision.models.resnet18, already contains all the necessary convolutional, ReLU, and pooling layers to process a 224 x 224 input and generate features. We need to replace its fully-connected layer to make sure that it is tuned for our specific purposes - note that the fully-connected layer does not appear in the diagram of the problem description – Pre-trained weights are used as dictated by the conclusion of the Pre-Visualization.

The DataLoader thus includes the transformation to this expected format. The standard transformation comes from the ResNet18 torch vision model documented at [3]

The embeddings dimension is 128 as reported in [1] and is thus the output features size for the fully-connected layer. I added normalization after the fully-connected layer.

I tried the much deeper ResNet-152 and quickly ran out of Memory.

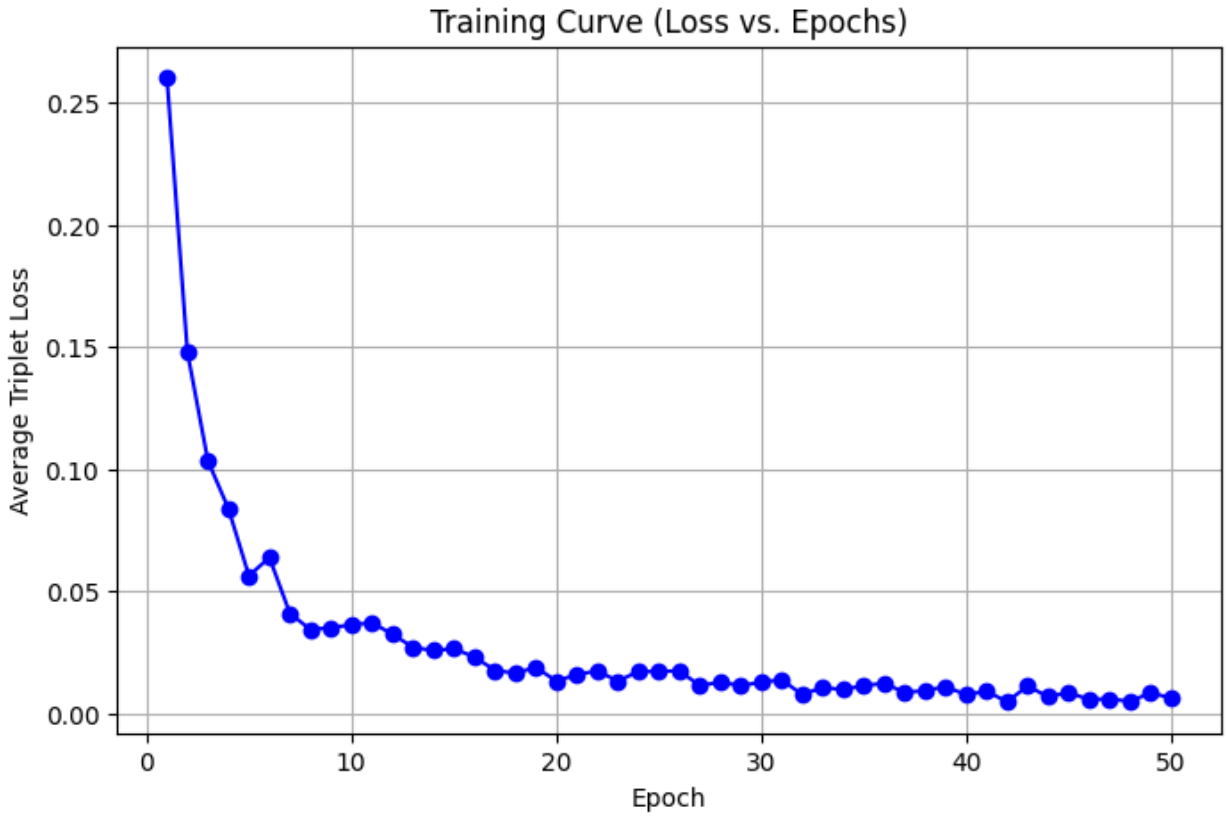
I wrote a generic TripletDataSet and used it with CNIST-10 before I wrote a pre-processing module for the SOP Dataset. I use the Stocastic Gradient Descent optimizer (Performed way better than Adam) and the standard Triplet Loss Module from [2].

After standard training with 50 epochs I turn to eval mode after which I extract the embedding and save aside the labels from the test set, which are then fed to the Precision@K calculation. Finally I produce t-SNE, UMAP and PCA vizualizations.

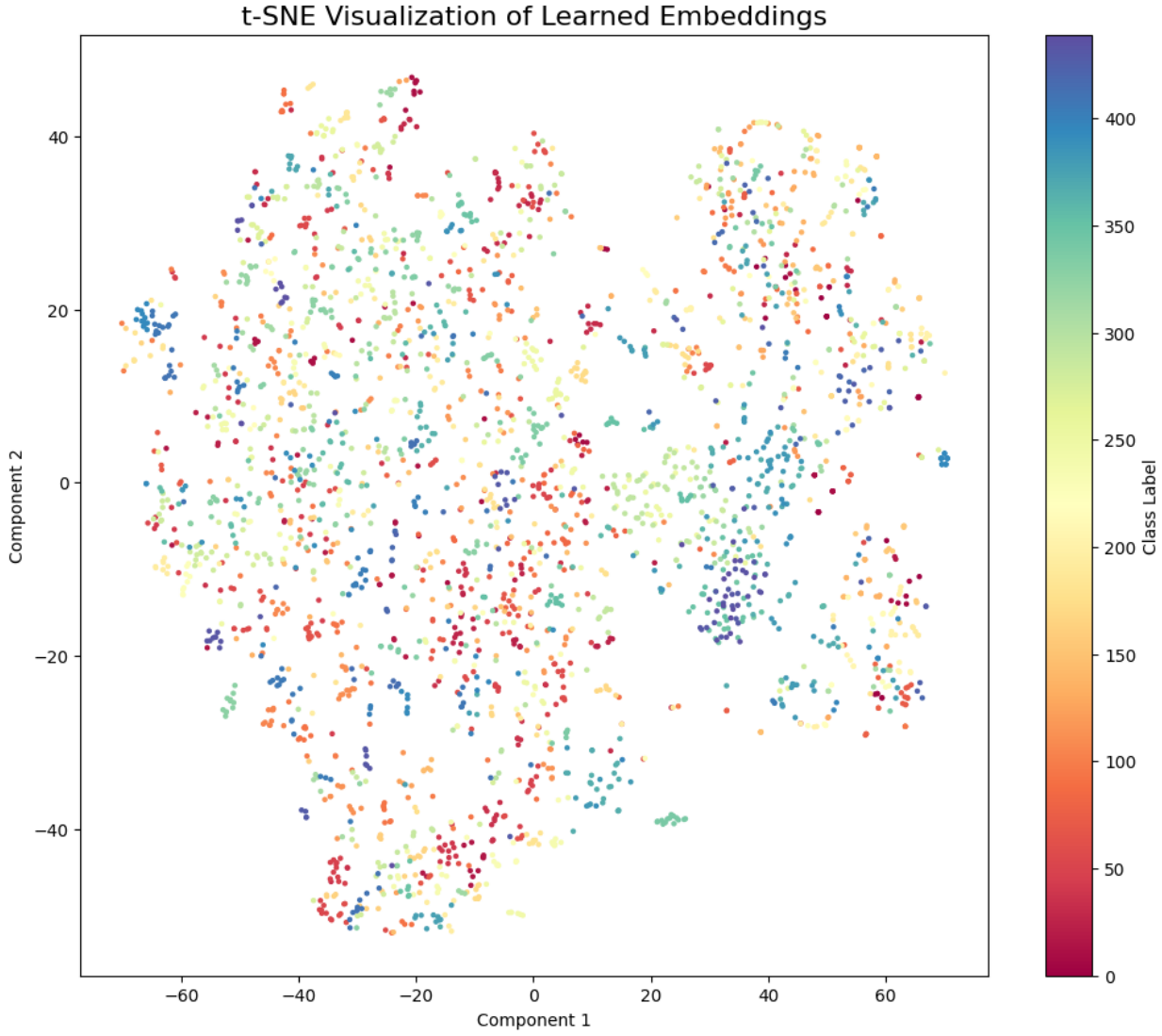
Trying to normalize with L2 actually produced worst results and a loss several order of magnitudes greater.

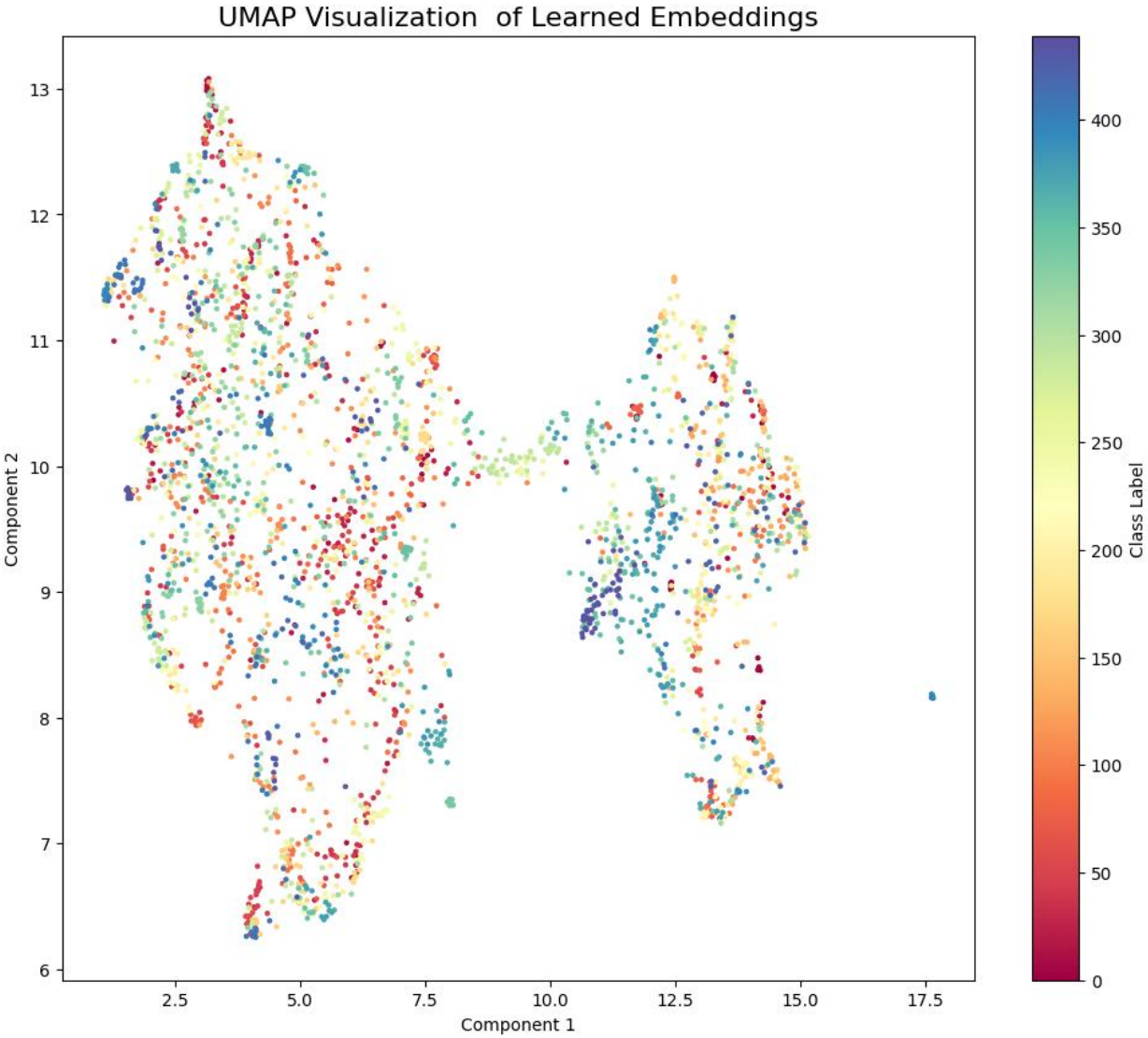
I observed a clear runtime impact in increasing the number of workers for the DataLoader to 4, and no further improvement beyond that number.

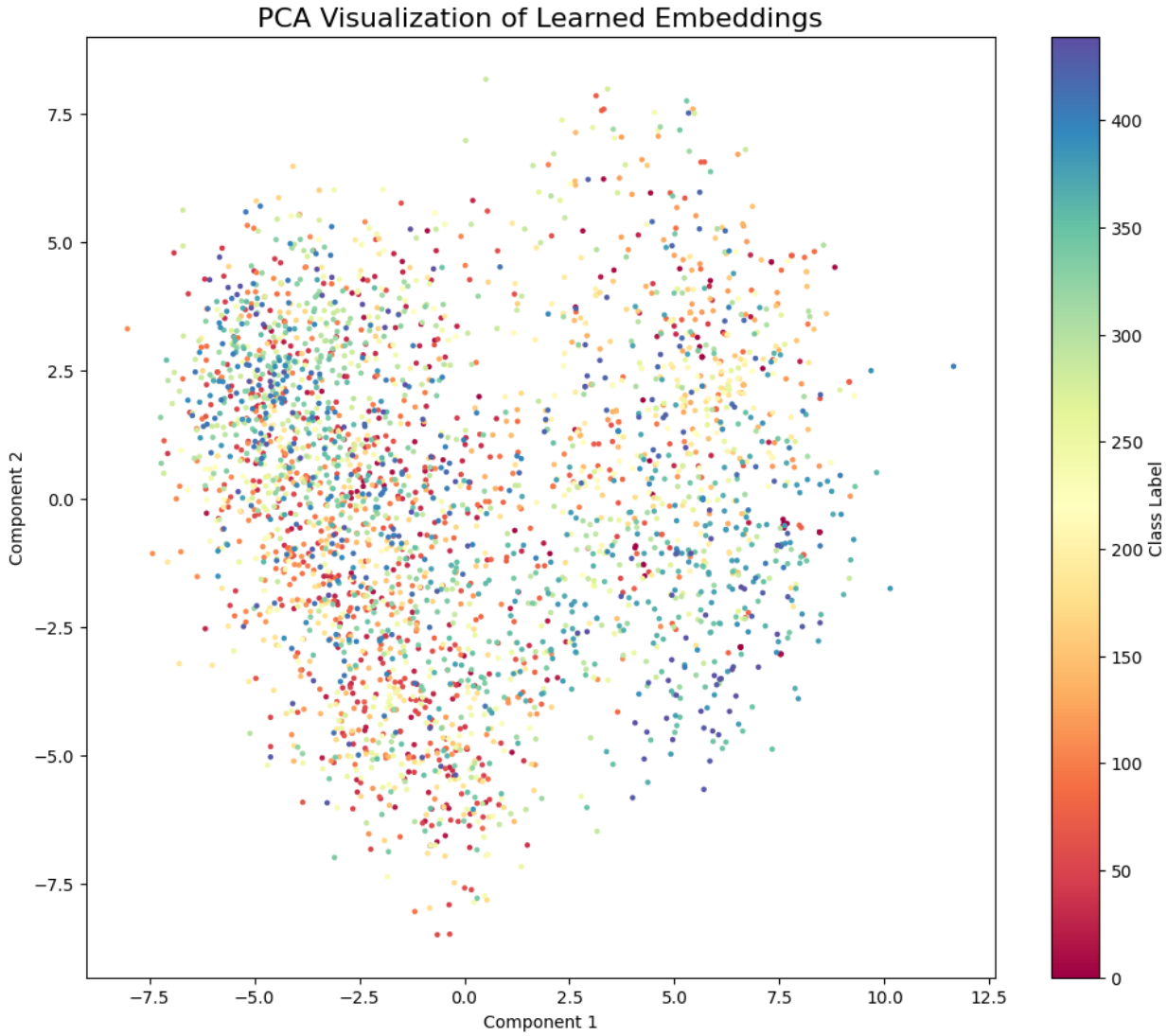
Training Curve



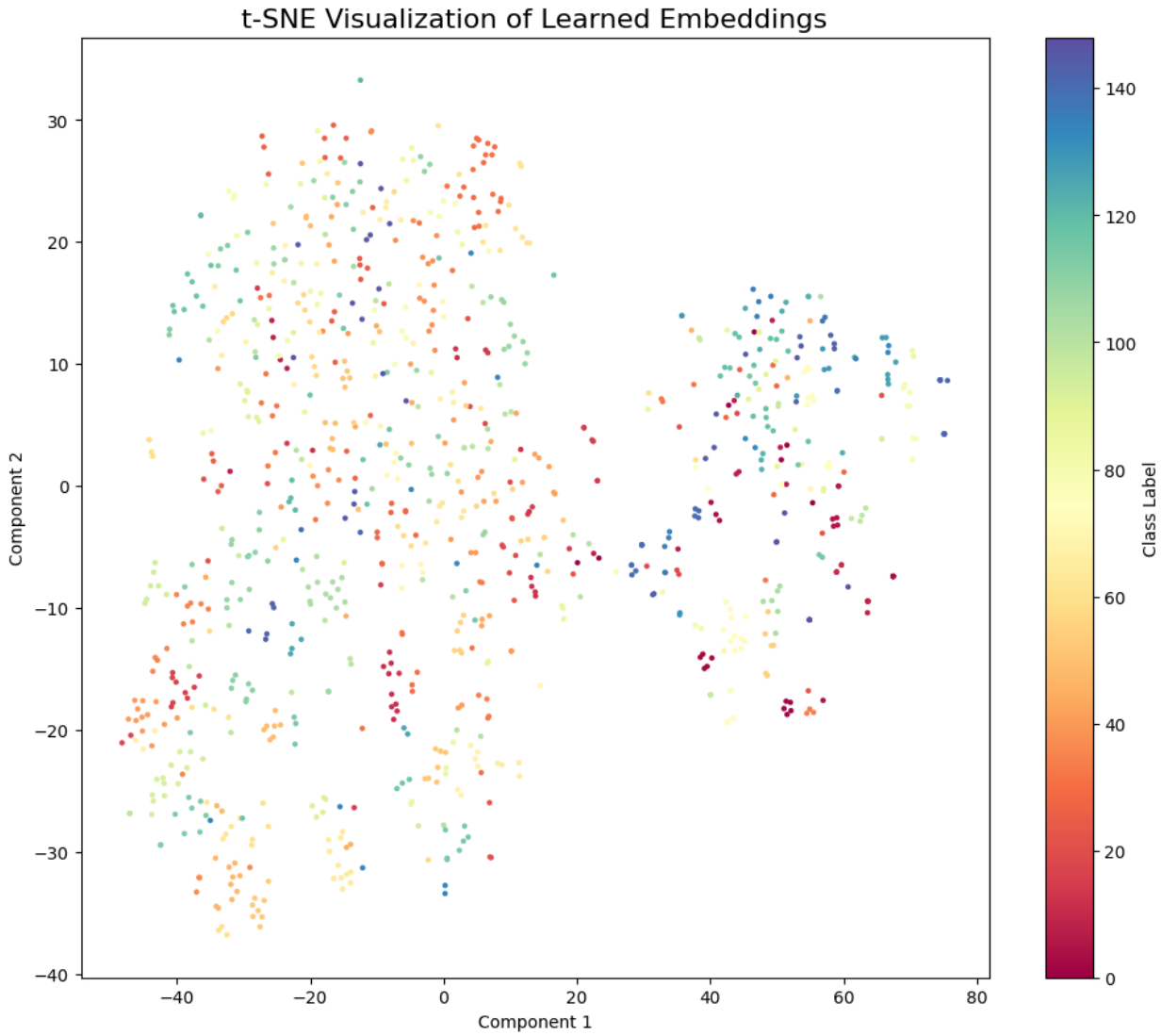
Embedding Visualizations


t-SNE Visualization of Learned Embeddings
Component 1
Component 2
Class Label
−60
−40
−20
0
20
40
60
40
20
0
−20
−40
400
350
300
250
200
150
100
50
0


UMAP Visualization of Learned Embeddings
Component 2
Component 1
Class Label
13
12
11
10
9
8
7
6
2.5
5.0
7.5
10.0
12.5
15.0
17.5
400
350
300
250
200
150
100
50
0


PCA Visualization of Learned Embeddings
Component 1
Component 2
Class Label
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
400
350
300
250
200
150
100
50
0

The Classification is clearer with 1000 samples



Evaluation results

Analysis of Precision@k

The observed Precision@10 of 0.2622 (26.22%) is a strong indicator of successful metric learning, especially considering the constraints of the fine-grained dataset and limited sample size.

Validation of Triplet Loss: the model successfully structured the embedding space such that, on average, over 2.6 of the top 10 nearest neighbors for any product query belong to the same

product class. This demonstrates the model's ability to learn discriminative, fine-grained features that standard classification often misses.

Justification for constraints: given that the Stanford Online Products dataset is highly sparse and contains thousands of unique classes, achieving a Precision@10 exceeding 25% using only a small subset of 3,000 images is very promising. It indicates that the pretrained ResNet backbone provided strong initial features, and the Triplet Loss efficiently fine-tuned those features even with limited positive examples.

Performance ceiling: The score suggests that the model's performance is likely limited by the data volume. Training on the full validation/test set would almost certainly yield a significantly higher Precision@k, as the model would encounter more diverse positive and negative pairs during training.

Interpretation of Embedding Visualizations

- The clear visual evidence of separation in the embedding space is the most critical confirmation that the Triplet Loss objective was met.
- T-SNE, UMAP, and PCA show separation: The fact that all three visualization techniques (t-SNE, UMAP, and PCA) show a clear beginning of separation of data is paramount.
- The PCA plot confirms that separation exists even along the axes of maximum linear variance, while the t-SNE/UMAP plots (which focus on local structure) confirm that the individual product clusters are tight and distinct from neighboring product clusters.
- This result is in stark contrast to the undifferentiated cloud observed in the initial, pre-training t-SNE visualization (on fixed ResNet features). This difference visually proves the effectiveness of the metric learning fine-tuning over generic feature extraction.

Better classification visualization with less samples?

The primary reason why the t-SNE visualization might appear better clustered with 1000 samples than with 3000 samples is the computational complexity and sensitivity of the t-SNE algorithm. This is a known limitation of the t-SNE algorithm, which struggles with computational load and parameter sensitivity on denser, larger datasets.

Yet, the Precision@10 score (0.2622) is the *primary* evidence that the model is successfully learning a clustered space, as the numerical retrieval metric is a more robust measure of embedding quality than the visual clarity of the complex t-SNE output.

In summary, the evaluation provides both quantitative evidence (Precision@10 = 0.2622) and qualitative evidence (clear clusters) that the implemented Triplet Loss pipeline effectively learned a robust, fine-grained metric space despite being constrained to a small subset of the total dataset.

References

[1] Paper on Learning local feature descriptors with triplets and shallow convolutional neural networks https://bmva-archive.org.uk/bmvc/2016/papers/paper119/index.html

[2] https://docs.pytorch.org/docs/stable/generated/torch.nn.TripletMarginLoss.html

[3] https://docs.pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html

[4] Stanford Online Products dataset at https://cvgl.stanford.edu/projects/lifted_struct/

[5] Custom datasets https://docs.pytorch.org/tutorials/beginner/basics/data_tutorial.html