ELEC 475 Computer Vision

Lab 4 – Fine-tuning ResNet50 for CLIP

Queen's University

Faculty of Engineering and Applied Science

Nathan Duncan	20283699
Daniel Dubinko	20229482

December 2, 2025

Table of Contents

Introduction

Explain the motivation and structure of CLIP.

Motivation

CLIP (Contrastive Language-Image Pre-training) is an effective multimodal deep learning architecture introduced by OpenAI. Its function is to understand the relationship between two distinct data types: text and images. This implementation bridges the gap between traditional, rigid image classification systems and flexible natural language processing. In contrast, standard classification models, such as YOLOv8, are trained solely on thousands of images and a fixed set of specific class labels. If an image of a "wrench" has the ground truth label "wrench," the model is limited to identifying only those known labels. If such a model encounters an object outside its predefined label set, it cannot identify the object. CLIP operates on a distinct principle. Instead of being assigned a single, strict label, it is presented with an image and a list of natural language captions, and its task is to predict which caption offers the optimal match. These captions serve as a natural language description of the visual content, often referred to as alt text.

The entire process relies on a dual-encoder system. The Text Encoder generates a high-dimensional vector (embedding) for the text caption. The Image Encoder, which in this case uses ResNet50, generates a vector for the image. The core objective is alignment: to maximize the cosine similarity (represented by the dot product) between the resulting image vector and the corresponding text vector, ideally resulting in a value close to 1. This metric quantifies the similarity, or alignment, between the two embeddings in the shared latent space. Crucially, the text encoder's weights are frozen (they are NOT updated during backpropagation). The rationale is that the text encoder has already established an effective understanding of language. Therefore, the goal of training is to fine-tune the ResNet50 image encoder's weights to align its output vector with this established text vector space. This alignment is what enables CLIP's key capability of zero-shot learning: identifying and classifying images based on a descriptive text query, even for objects it has never encountered during training.

Structure

CLIP has two phases. The training phase (building the shared embedded space and the inference phase (zero-shot classification). The training phase takes a batch of images and their paired captions. Then, both encoders convert their inputs into normalized vectors (in the embedded space). Next, the model calculates the dot product between the image vector and the caption vectors, if the caption A and Image A dot product is low then infoNCE loss function generates a large error, if Image A and caption B score a low score then the error generated is low. The error gets converted to a gradient and then an optimizer like Adam uses this gradient to adjust the weights in the proper direction based on the target task. 0 for different captions and Images and 1 for same captions and images.

Methodology

Summarize your model and training design.

**all PascalCase names are classes*

Model Design

The model is a simplified CLIP model. The goal is two take a image vector and text vector and map them into the same embedded vector space of 512.

The **ImageEncoder** has trainable weights and functions on the backbone of ResNet50, pre-trained on ImageNet. The line '**self.backbone.fc = nn.Identity()**' chops of the head and instead of classes the ResNet50 ouputs a raw feature vector. Since the text model outputs vectors of size 512, the image encoder uses **ProjectHead** to map the 2048 image vector to size 512.

The **TextEncoder** class loads a pretrained *clip-vit-base-patch32* model that OpenAI already trained to understand the relationship between English and Images. The weights are frozen because the model acts a ground truth. We are forcing the image model to match these existing vectors. The text model weights are not updated.

The **CLIPModel** class puts everything together. It takes a batch of images and a batch of tokenized text and passes them through the **ImageEncoder** and **TextEncoder** that then both output a 512-dim vector.

In addition, it has a learnable temperature parameter called **self.temperature**, that is crucial for scaling the logits in the contrastive loss function. It ensures similar logits are scaled further away from each other to prevent similar learned parameters.

Data Flow:

1. Image Input (Batch, 3, 224, 224) -> ResNet -> (Batch, 2048) -> Projection -> (Batch, 512)
2. Text Input (Batch, Sequence Length) -> BERT/CLIP Text -> (Batch, 512)
3. Result: Two Matrices of shape (Batch, 12) that are normalized

Training Design

Non-Augmentation

Dataset and Data-loading

CocoDataset: Loads the COCO 2014 Captions dataset and creates a flat list of (image_filename, caption_string) pairs. Each image has multiple captions, and each pairing is treated as an individual sample.

CocoCollator: This is the key component for batch preprocessing. It uses the **CLIPProcessor** (with the same openai/clip-vit-base-patch32 model name) to perform the following:

- **Image Preprocessing:** Resizes, crops, and normalizes the PIL images.
- **Text Tokenization:** Tokenizes and encodes the caption strings, adding necessary padding and truncation. It returns a dictionary containing the processed pixel_values (image tensor), input_ids, and attention_mask (text tensors).

DataLoader: Utilizes the CocoCollator to prepare batches of size BATCH_SIZE=32 for training.

We created a verification script **verify_dataset.py**, to ensure that caption-image pairs were being loaded properly. Figure 1 shows three sample images validating proper pairing.

Figure 1: Verification of caption-image pairings during loading.

Contrastive Loss Function

The model is trained using the **InfoNCE loss**, which is the core of the CLIP objective. For a batch of N image-caption pairs, the goal is to maximize the similarity between the N correct (image, caption) pairs and minimize the similarity with the N×(N−1) incorrect (image, caption) pairs (the negative samples).

- **Logits Calculations:** The similarity matrix L is calculated as the scaled dot product between all the image embeddings I and the all the text embeddings T:

$$L = (IT^T) * \exp(\tau)$$

$$I \epsilon R^{N*D}, T \epsilon R^{N*D}, D = 512, \tau \; is \; temperature$$

- **Targets:** The target labels are the diagonal of the matrix, labels=[0,1,…,N−1], as the correct match is always $I_i \; with \; T_i$
- **Symmetric Loss:** The final InfoNCE loss is the average of two cross-entropy losses:
  - **Image-to-Text Loss** $(\mathcal{L}_i)$: Treating rows of L as image-to-caption scores, encouraging each image to match its correct caption:
    $$\mathcal{L}_i = CrossEntropy(L, labels)$$
  - **Text-to-Image Loss** $(\mathcal{L}_t)$: Treating columns of L as caption-to-image scores, encouraging each caption to match its correct image:
    $$\mathcal{L}_i = CrossEntropy(L, labels)$$
    $$\mathcal{L} = \frac{1}{2}(\mathcal{L}_i + \mathcal{L}_t)$$

Training Loop

- **Optimizer:** AdamW is used with LEARNING_RATE = 1e-5
- **Process:** The model is trained for EPOCH = 5.
  1. Image and text inputs are moved to the DEVICE (mps or cuda)
  2. Forward pass: *img_emb, txt_emb = model (…)*
  3. Loss calculation: *loss = info_nce_loss(img_emb, txt_emb, model.temperature)*
  4. Backpropagation and optimizer step
- **Monitoring and Saving:** training loss is tracked, an average loss is printed per epoch, and model checkpoints are saved to the checkpoints directory after each epoch.
- **Plotting:** a training loss curve is generated and saved.

Augmentation

The model was the augmented with the following:

1. **Data augmentation:** replaced the static resizing with *RandomResizedCrop,RandomHorizontalFlip*, and *ColorJitter*. This forces the model to learn invariant features rather than memorizing specific pixel arrangements.
2. **Architectural Regularization (in model_augmentation.py):** I added Dropout layers to the Projection Head to prevent overfitting to the training data.
3. **Training Optimization (in train_augmentation.py):**
    a. Learning Rate Scheduler: Implemented CosineAnnealingLR to smooth out convergence.
    b. **Mac M4 Optimization:** Added explicit support for the mps (Metal Performance Shaders) device, which allows PyTorch to use the GPU on your M4 Pro. This was done since the augmented model was trained on a Macbook Pro M4 Pro.

Hyper Parameters and Hardware

Describe all hyperparameters used in training, and describe the hardware used. How long did the training take? Include the loss plot.

Table 1 summarizes the key hyperparameters, hardware, and training time of the augmented and non-augmented clip models.

Table 1: Training configuration for non-augmented and augmented models.

SCENARIO	HYPERPARAMETERS	HARDWARE	TIME
NON-AUGMENTATION	BATCH_SIZE = 32 LEARNING_RATE = 1e-4 EPOCHS = 5 TEMPERATURE = 0.7	Colab (T4 GPU)	398.383 minutes
AUGMENTATION	BATCH_SIZE = 32 LEARNING_RATE = 1e-4 EPOCHS = 5 TEMPERATURE = 0.7	M4 Pro Chip (GPU)	382.84 minutes.

Training loss of these models are discussed in the Results section and shown in Figures 2 and 3.

Mathematical Intuition behind CLIP loss

Explain the mathematical intuition behind the CLIP loss (anchors, positives, and negatives)

Anchors: The specific image vector you are currently trying to classify

Positives: The one text caption in the batch that describes this image

Negatives: The N-1 other captions in the batch that describe other images

The goal of the loss function is to maximize the similarity between the Anchor and the positive and to minimize the similarity between the anchor and all the negatives

The code implements this using the infoNCE loss.

The infoNCE loss function acts like a tug-of-war using the softmax probability.

$$P(Positive|Anchor) = \frac{\exp\left(Positive\ score\right)}{\exp(Positive\ score) + \sum\exp\left(Negative\ Scores\right)}$$

The **numerator** is the pull; we want the positive score (dot product) to be high. This pulls the correct pairs together.

The **denominator** is the push; we want the sum of all the scores to be dominated by the positive. This pushes the negative away.

By making the positive scores dominate and the negative scores to approach 0, the probability approaches 1 or in other words 100%

Given a batch size of N, the data loader ensures that for every image I_i there is a corresponding correct caption T_i. This makes it so that the correct matches are always on the main diagonal and the incorrect matches are the off diagonal. The model then pushes the main diagonal scores to score high while pushing the off-diagonal scores to score low.

A temperature parameter scales the dot products. High temperature, the model is ‘relaxed’ and tolerates weak matches. A low temperature, the model is ‘strict’ and forces the positive match to be significantly stronger than any negative.

Results

Present quantitative metrics (Recall@K, loss curves).

As mentioned previously we trained two CLIP models; a non-augmentation base case, and an augmented model with data modifications, dropout layers and hyperparameter adjustments. Figure 2 shows the training and validation for then non-augmented model with a final training InfoNCE loss of 2.33.

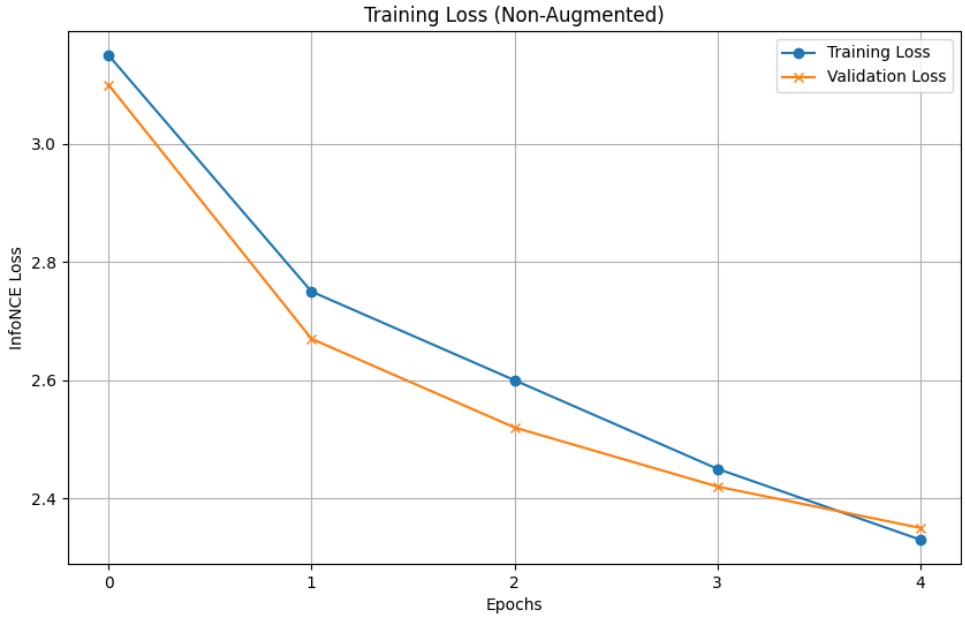


Figure 2: Training and Validation InfoNCE loss of the non-augmented model over 5 epochs.

Figure 3 shows the training and validation of the augmented model. We can see the adjustments to the learning rate and its scheduler had a strong impact on the decrease in loss over the five training epochs reaching a new final InfoNCE loss of 0.483.

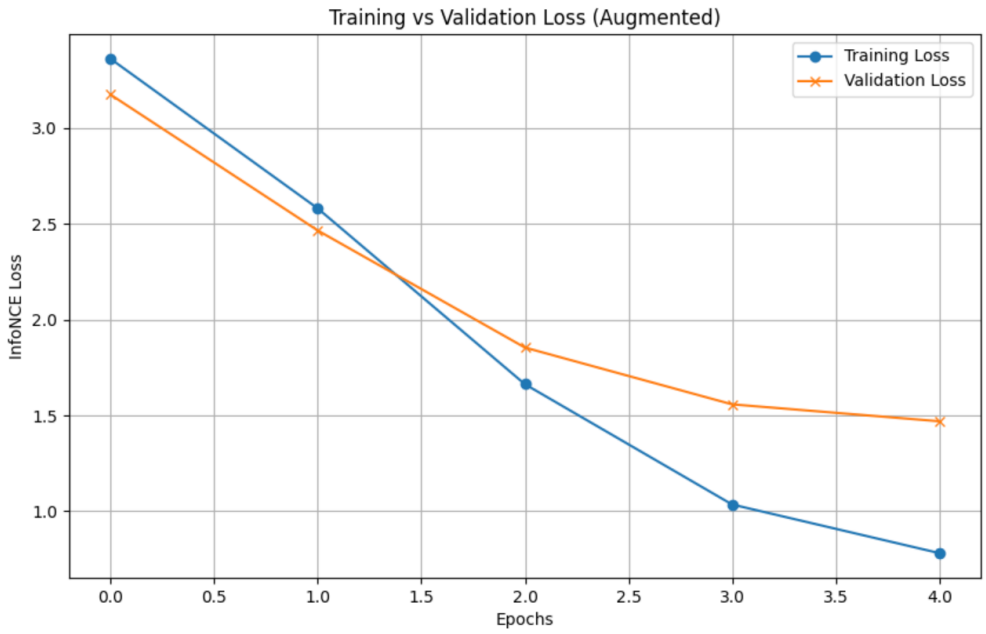


Figure 3: Training and Validation InfoNCE loss of the non-augmented model over 5 epochs.

For evaluation we used only a subset of the dataset, N, as computing similarity matrix for the entire dataset was too intensive and caused Google Colab to dropout. Table 2 compares the recall statistics for image to text and text to image retrievals. We can see that our augmented model achieves superior recall statistics, recalling the correct image/caption on average about 5% more often through all classes.

Table 2: Recall@K metrics for image to text and text to image calls of the two models.

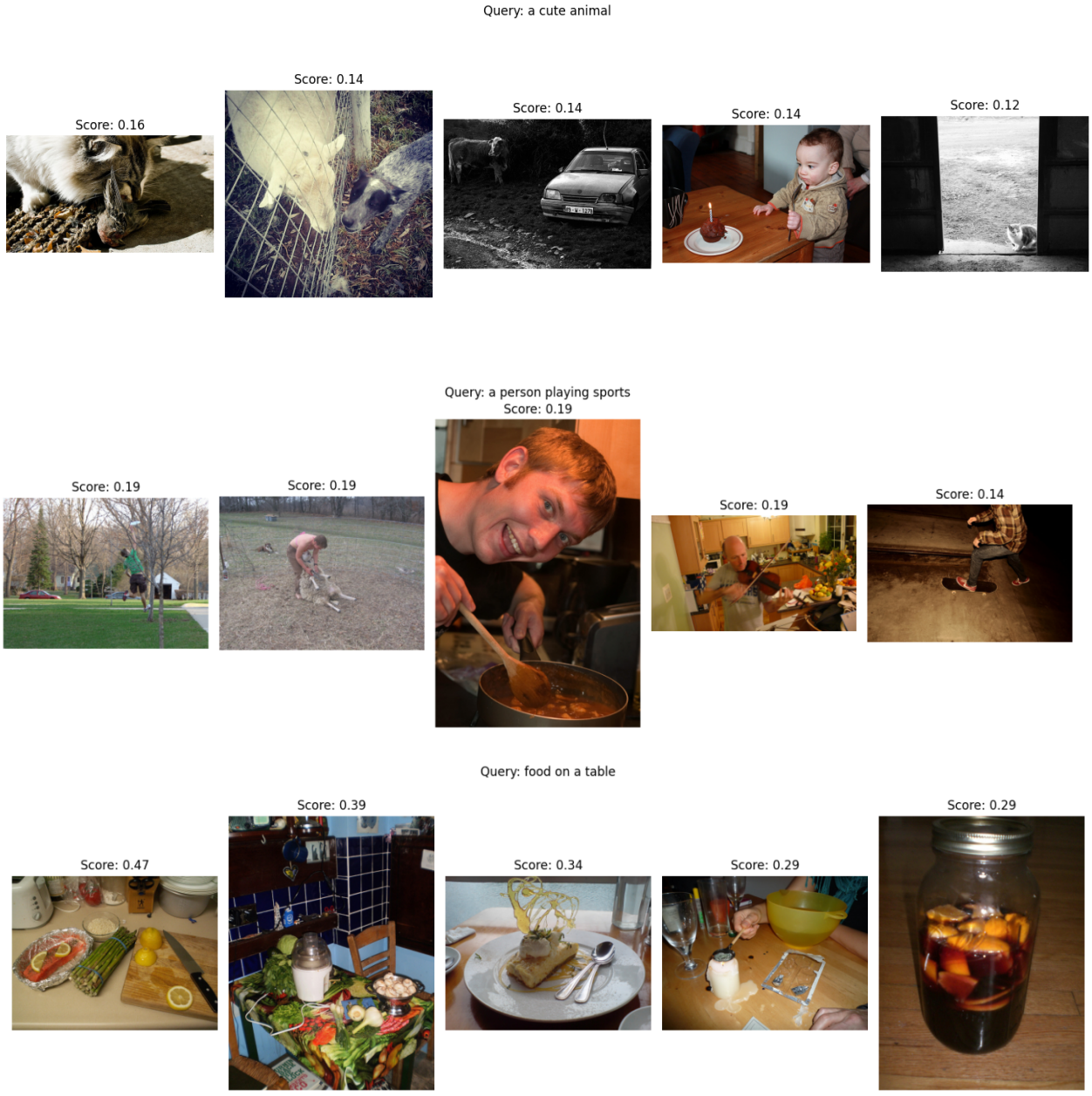
METRIC	MODEL	N=2000
IMAGE-TO-TEXT RECALL@1	Non-Augmentation	0.2385
	Augmentation	0.3085
IMAGE-TO-TEXT RECALL@5	Non-Augmentation	0.5305
	Augmentation	0.6260
IMAGE-TO-TEXT RECALL@10	Non-Augmentation	0.6820
	Augmentation	0.7570
TEXT-TO-IMAGE RECALL@1	Non-Augmentation	0.3440
	Augmentation	0.4095
TEXT-TO-IMAGE RECALL@5	Non-Augmentation	0.6330
	Augmentation	0.7200

TEXT-TO-IMAGE RECALL@10	Non-Augmentation	0.7660
	Augmentation	0.8255

Retrieval Examples

Provide qualitative retrieval examples demonstrating semantic alignment between modalities.

Non-Augmentation

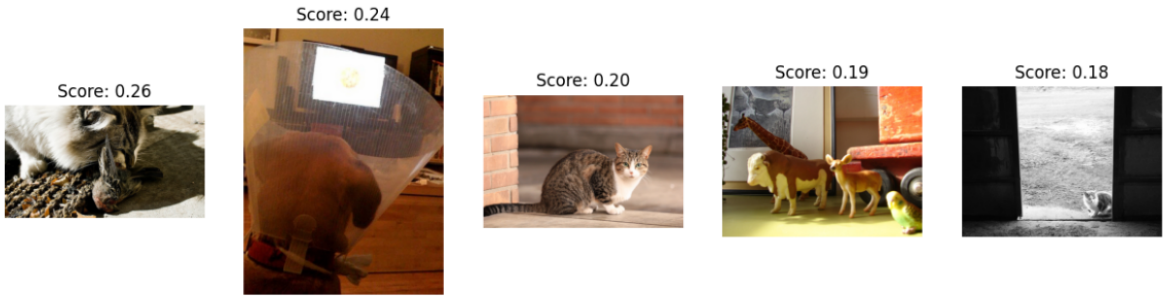


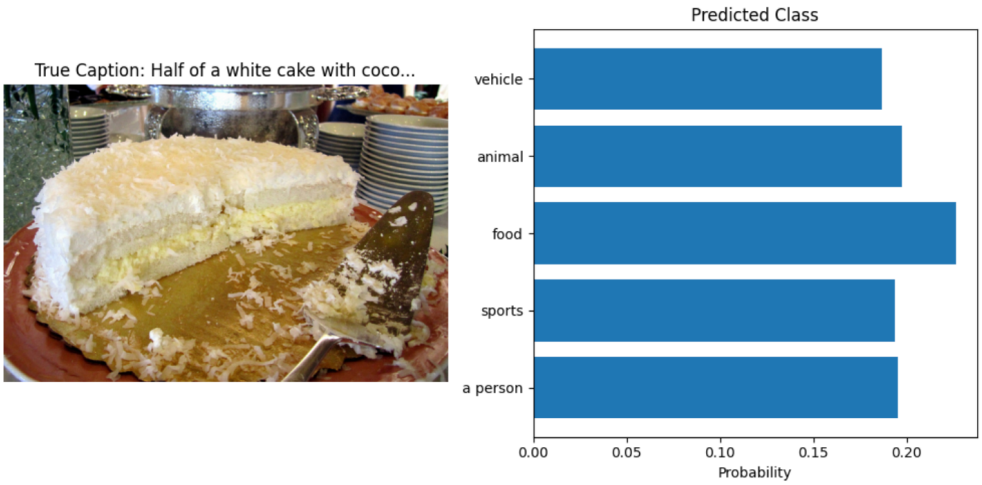
Augmentation

Query: food on the table

Query: a person playing sports

Query: a cute animal





Discussion

Interpret results and analyze model behavior.

Both models performed well in determining the “food on the table” query. The model likely uses just two nouns: “food” and “table.” However, it struggled with the query “a person is playing a sport.” For both models, a person was engaged in an activity, but not a sport, likely confusing the word ‘playing’ with non-sport instances. Similarly, for the “a cute animal” query, the model learned the noun “animal” but struggled with the adjective “cute,” as one of the photos shows a cat eating a dead bird or mouse.

With 82,784 training images and only 5 epochs, the model has only seen each image-caption pair five times. For contrastive learning methods like CLIP, models typically need to see examples many times to effectively separate embeddings of “dissimilar” concepts and create greater separation between similar concepts. The scores are low and clustered together (0.26, 0.24, 0.20, 0.19, 0.18), indicating a lack of confidence in the model’s predictions. In a more converged model, the top match and the fifth match scores should be more widely spaced. Currently, the embeddings remain clustered rather than being distinctly separated into categories like “cute animals” or “wild animals.”

Furthermore, the batch size is limited to 32, which may not provide enough negative examples in each batch. With a small batch size, the model doesn’t exert much effort to find the best match and might create lazy features, such as “it has fur = animal,” rather than more detailed features.

Conclusion

Reflect on what was learned and potential improvements.

This project successfully implemented a CLIP model on the COCO 2014 dataset, demonstrating effective object-level retrieval for queries like "food on the table." However, the evaluation highlighted significant limitations imposed by constrained compute resources. The model's failure to distinguish "cute animal" from a generic "animal" indicates it prioritized dominant visual features (nouns) over fine-grained semantic attributes (adjectives), resulting in "lazy" feature learning.

This underperformance is primarily attributed to the small batch size (32) and short training duration (5 epochs). In contrastive learning, small batches provide insufficient negative samples, allowing the model to minimize loss without learning robust discriminative features. Future work must prioritize scaling the batch size and increasing training epochs (15-20) to force the separation of dissimilar embeddings and create further uncoupling for similar embeddings. With the increase in batch size and epochs, the model now has high quality and robust embeddings. With the increased confidence, the model can afford to decrease the temperature to further minimize loss.

LLM

Reflect on your use of LLMs or chatbots during the lab; e.g., what queries were helpful and how you validated their outputs. Include a link to your conversation.

The Google Gemini LLM was used for content generation in this lab.

LLM Conversation Links:

1. https://gemini.google.com/share/6adebe0cee85 (Non-augmentation)
2. https://gemini.google.com/share/1ba90138f0d9 (Augmentation)

It was found to be very effective, especially when given small, specific prompts or example code to modify. It was also helpful for Colab configuration. The LLM did produce some errors, but it could often self-correct when provided with the console output and a description of what went wrong.