Notes

Prerequisite Brush-up

- **1. Vectors and Cosine Similarity**
  - Think of both an image and a sentence as a single **vector** (a list of numbers) in a high-dimensional space. This vector represents its *semantic meaning*.
  - **Cosine Similarity** is a metric that measures how *aligned* two vectors are (i.e., how similar their directions are). A high score (near 1.0) means "similar meaning," while a low score (near 0 or -1) means "different." It's used because it cares about *content* (direction) not the vector's length (magnitude).
- **2. Embeddings and a Shared Space**
  - An **embedding** is just the technical term for this "meaning vector" (e.g., the *image embedding* or *text embedding*).
  - CLIP's main goal is to train its two encoders to map both images and their matching text into the *same* space. In this **shared embedding space**, the vector for a picture of a dog is very close to the vector for the text "a picture of a dog."
- **3. Encoders for Images and Text**
  - An **encoder** is the AI model that "encodes" raw data into an embedding.
  - The **Image Encoder** (like a ResNet or Vision Transformer) takes in pixels and outputs a single image embedding.
  - The **Text Encoder** (a Transformer) takes in text tokens and outputs a single text embedding.
- **4. Contrastive Learning**
  - This is the training technique. It teaches the model by "contrasting" examples.
  - It **pulls** the embeddings of matching image-text pairs (positives) *together*.
  - It **pushes** the embeddings of all non-matching pairs (negatives) *apart*.
- **5. Softmax and Cross-Entropy Loss**
  - This is the math that powers contrastive learning.
  - Inside a training batch of N pairs, the model makes an $N \times N$ grid of all similarity scores.
  - **Softmax** is used to convert these similarity scores (for one image vs. all N texts) into probabilities.
  - **Cross-Entropy Loss** then penalizes the model if it doesn't assign the highest probability to the *one* correct text. This forces the model to get good at finding the single true match.
- **6. Zero-Shot Classification and Prompting**
  - **Zero-shot** is the "magic" ability to classify new things without any fine-tuning.
  - CLIP does this by turning text labels into "prompts" (e.g., "a photo of a cat", "a photo

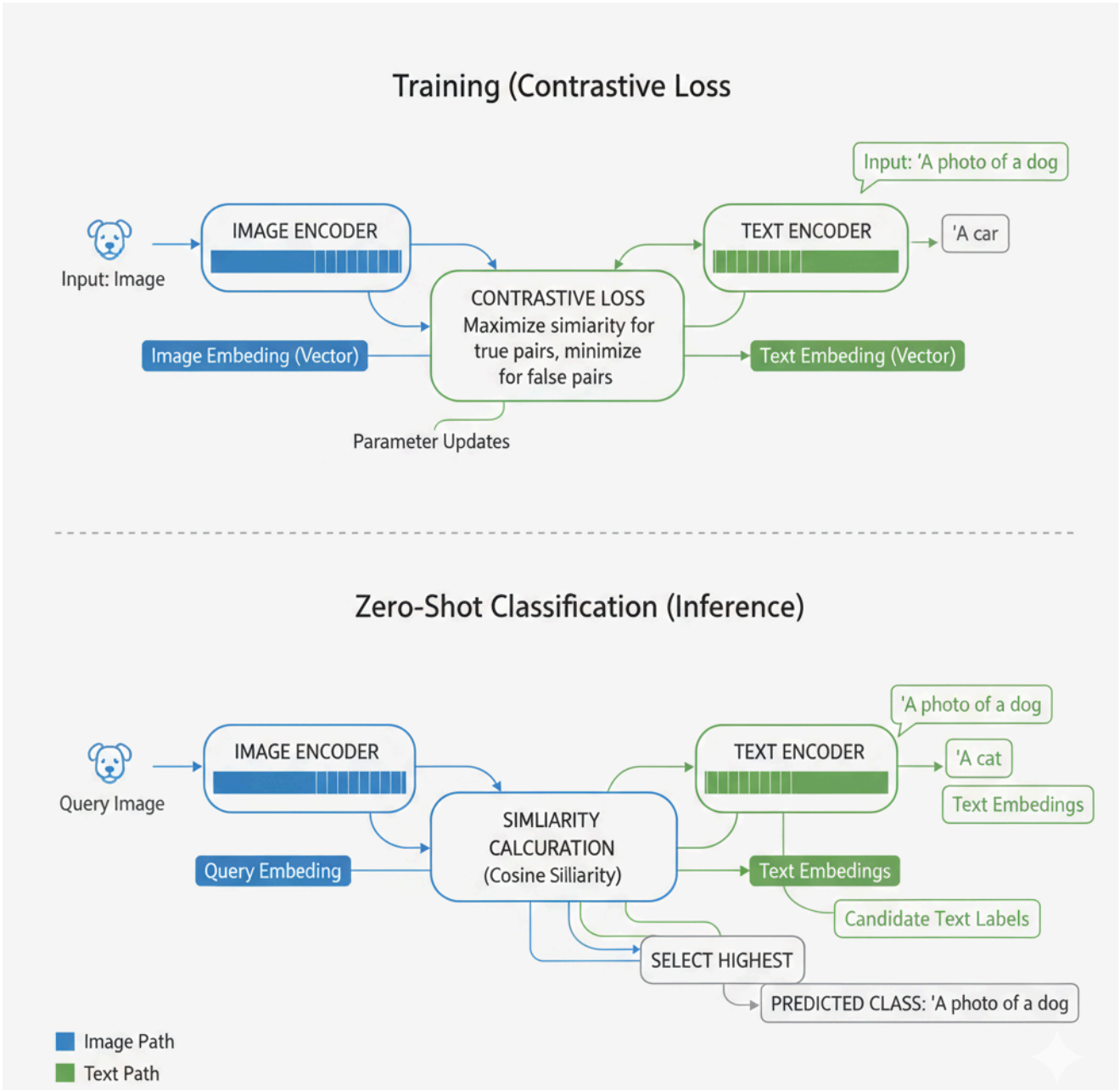
of a car").
    - It then encodes an image and all the text prompts, and simply checks which prompt embedding is *closest* to the image embedding. This "closest match" is the prediction.
- **7. Image–Text Supervision vs. Fixed Labels**
    - **Old way (Fixed Labels):** Training on a dataset like ImageNet, where every image is tied to one of a small, *fixed* set of 1000 labels (e.g., "tench", "goldfish").
    - **CLIP's way (Natural Language Supervision):** Training on 400 million (image, text) pairs from the internet. The text is a free-form caption ("a picture of my goldfish in its new bowl"), which is much richer and covers an almost infinite "open vocabulary."
- **8. Optimization Basics**
    - You just need to know that the model is trained with **gradient-based optimization**. It calculates its error (the loss) on a batch and slightly adjusts all its parameters to become a little bit better, repeating this process millions of times.

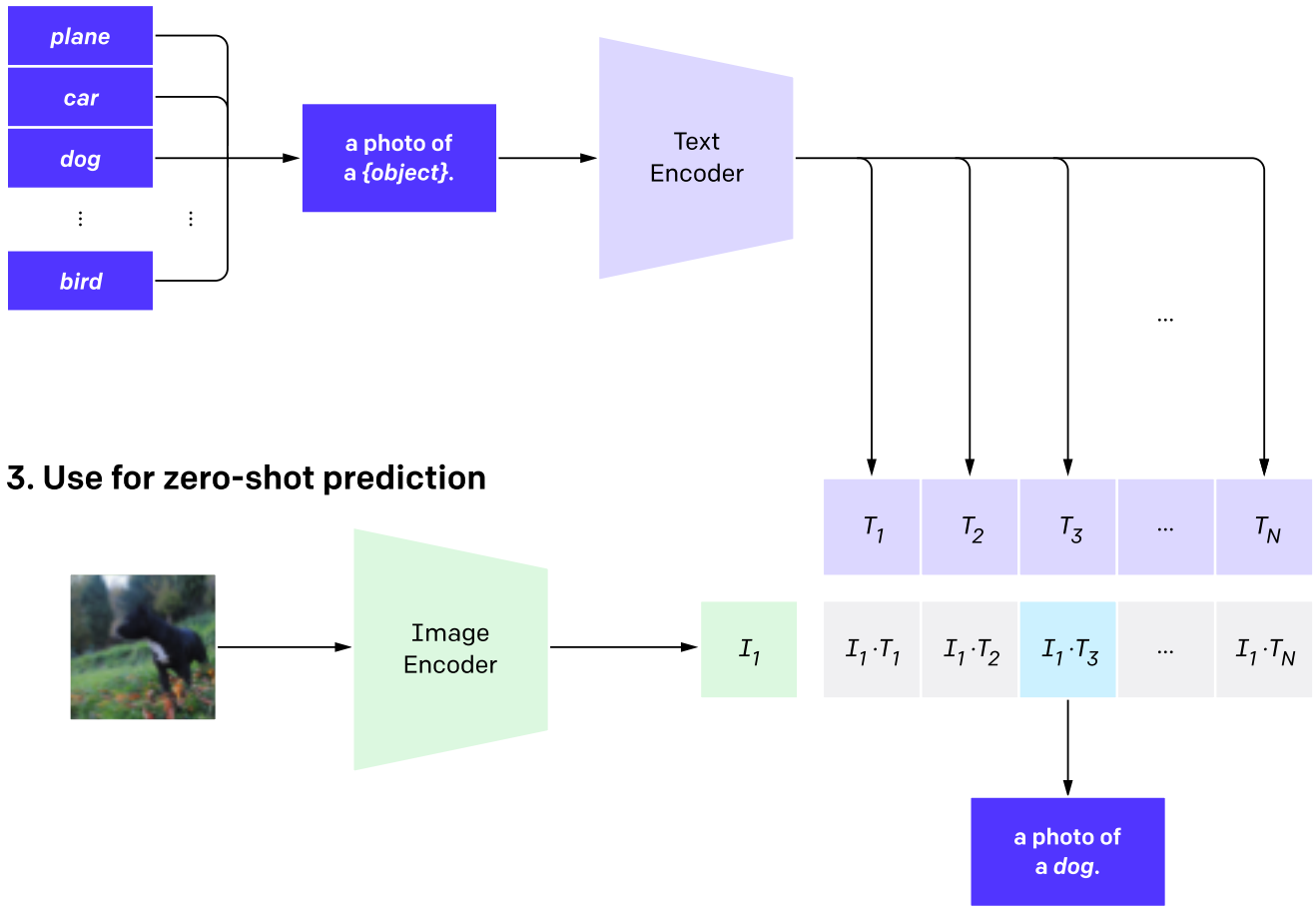
Part 1 : Learning Transferable Visual Models From Natural Language Supervision (CLIP)

Summary

CLIP solves the "fixed-label cage" problem of traditional computer vision, where models were expensive and limited to a predefined set of categories. It does this by learning from 400 million (image, text) pairs found on the internet, using efficient contrastive learning to create a shared embedding space, which enables powerful **zero-shot classification**.

Diagram


Training (Contrastive Loss
Input: 'A photo of a dog
Input: Image
IMAGE ENCODER
TEXT ENCODER
'A car
CONTRASTIVE LOSS
Maximize simiarity for
true pairs, minimize
for false pairs
Image Embeding (Vector)
Text Embeding (Vector)
Parameter Updates
Zero-Shot Classification (Inference)
'A photo of a dog
Query Image
IMAGE ENCODER
TEXT ENCODER
'A cat
Text Embedings
SIMLIARITY
CALCURATION
(Cosine Silliarity)
Query Embeding
Text Embedings
Candidate Text Labels
SELECT HIGHEST
PREDICTED CLASS: 'A photo of a dog
Image Path
Text Path



Key Points

- **Problem:** Previous vision models were expensive, inflexible, and couldn't generalize beyond their fixed training labels (e.g., ImageNet's 1000 categories).
- **Big Idea:** Use the internet's vast supply of images *with their existing captions* as a broader, more flexible source of supervision.
- **Dataset:** Trained on a massive new dataset of **400 million (image, text) pairs**.
- **Method:** Uses **contrastive learning**, which is far more efficient than trying to predict the exact caption for an image.
- **Architecture:** Employs two separate encoders: an **Image Encoder** (ResNet/ViT) and a **Text Encoder** (Transformer).
- **Core Task:** The model learns to match the correct image embedding to the correct text embedding within a large batch, "contrasting" it against all incorrect pairs.
- **Zero-Shot Magic:** Can classify images for *any* category by creating a classifier "on the fly" from simple text prompts (e.g., "A photo of a dog").
- **Result:** Matched the performance of a fully-supervised ResNet-50 on ImageNet *without ever being trained on a single ImageNet label*.

Step-by-Step Explanation

1. Training: Contrastive Learning

This is the method shown in Figure 1 of the paper.

1. A large batch of *N* (image, text) pairs is taken (e.g., N=32,768).
2. All *N* images are passed through the **Image Encoder** to get *N* image embeddings.
3. All *N* texts are passed through the **Text Encoder** to get *N* text embeddings.
4. An *N x N* matrix of cosine similarity scores is computed, comparing every image to every text.
5. **The Goal:** The model is trained to maximize the similarity scores for the *N* correct pairs (the diagonal of the matrix) and minimize the scores for all N^2-*N* incorrect pairs (everything off-diagonal).

2. Inference: Zero-Shot Transfer

This is the "magic" payoff of the training method.

1. You don't retrain or fine-tune the model.
2. You define your classes with text prompts (e.g., "A photo of a dog," "A photo of a cat," "A photo of a plane").
3. These text prompts are fed into the **Text Encoder** to create a set of "classifier" embeddings (one for each class).
4. You feed your new, unseen image into the **Image Encoder** to get a single image embedding.
5. You measure the **cosine similarity** between your single image embedding and all your text prompt embeddings.
6. The text prompt with the highest similarity is the model's prediction for the image's class.

Example

To classify a new picture, you don't need a model with a "dog" or "cat" output layer.

Instead, you give the CLIP Text Encoder two prompts:

1. "A photo of a dog"
2. "A photo of a cat"

You give the CLIP Image Encoder your picture (e.g., an image of a golden retriever). The model then checks if the image's embedding is closer to the "dog" prompt embedding or the "cat" prompt embedding in the shared space. It will report a higher similarity for "A photo of a dog," thus classifying it correctly.

Part 2 : Jupyter Notebook: Simplified CLIP Implementation

Summary

This notebook is an educational lab that provides a hands-on, simplified demonstration of the core concepts from the CLIP paper. It shows how to train a small-scale version of CLIP from scratch on the **CIFAR-10** dataset (instead of the paper's 400M pairs) and then successfully use it to perform **zero-shot classification** with high accuracy.

Key Points

- **Educational Purpose:** The notebook's goal is to "Demonstrate the mechanics of Contrastive Language–Image Pretraining" in a simple, understandable way.
- **Key Simplification (Data):** It uses the **CIFAR-10** dataset, which has 10 classes. It simulates natural language by creating text prompts for each class (e.g., `"a photo of a cat"`, `"a photo of a ship"`).
- **Key Simplification (Model):**
  - The **Image Encoder** is a standard (pretrained) **ResNet-18**.
  - The **Text Encoder** is *highly* simplified. Instead of the large Transformer from the paper, it uses a simple "Bag of Words" style model (an `Embedding` layer followed by a `mean()`). This is much faster to train.
- **Core Method:** It correctly implements the paper's core idea:
  1. It defines an **Image Encoder** and a **Text Encoder** that project features into a shared embedding space.
  2. It trains them using the **symmetric contrastive loss** (`clip_loss` function) from the paper, which aligns matching image-text pairs.
- **Zero-Shot Demonstration:** After training, it performs zero-shot classification on the test set by comparing each image's embedding against the embeddings of the 10 text prompts.
- **Successful Result:** Even with these simplifications, the model achieves **~94.05% zero-shot accuracy** on CIFAR-10 (as seen in the code output), proving the power of the concept.

Step-by-Step Explanation

The notebook code is broken down into clear sections:

1. **Setup (Cell 1):**
   - Loads the CIFAR-10 dataset.
   - Creates the 10 text prompts (e.g., `"a photo of a frog"`) that will act as the "natural language supervision".
2. **Model Definition (Cell 1):**
   - `ImageEncoder`: A ResNet-18 model that takes an image and outputs a normalized feature vector.
   - `TextEncoder`: A simple model that takes the tokens for a prompt (e.g., `["a", "photo", "of", "a", "cat"]`) and outputs a single normalized feature vector.
3. **Loss Function (Cell 1):**
   - The `clip_loss` function is the heart of the notebook.
   - It calculates the cosine similarity between a batch of image vectors and text vectors, creating a similarity matrix.
   - It then uses `cross_entropy_loss` to "classify" the correct pair, forcing the model to pull matching pairs together. This is the **symmetric contrastive loss**.
4. **Training (Cell 2):**
   - The notebook trains the two encoders together for 5 epochs.
   - The printed output shows the `clip_loss` decreasing from **~2.25 to ~1.97**, indicating the model is successfully learning to align the images and text.
5. **Loss Visualization (Cell 3):**
   - This cell simply plots the training loss, showing a clear downward curve as the model learns.
6. **Evaluation: Zero-Shot Classification (Cell 4):**
   - This is the "magic" test, just like in the paper.
   - **Step A:** It first embeds all 10 text prompts (e.g., "a photo of a bird," "a photo of a deer"...) to create the "classifier".
   - **Step B:** It loops through the *test images*, embeds them one by one, and calculates the similarity between each image and all 10 text prompts.
   - **Step C:** The prompt with the highest similarity score is the prediction.
   - **Result:** The notebook prints a final **Zero-Shot Top-1 Accuracy of 94.05%**.
7. **Prediction Visualization (Cell 5):**
   - This cell shows 5 sample test images.
   - For each one, it shows the "True" label (from CIFAR-10) and the "Pred" label (from the zero-shot classifier).
   - In the sample output, all 5 predictions are correct (e.g., True: cat, Pred: cat), visually confirming the high accuracy.
8. Accuracy Plot

   - Example

The core logic of CLIP is captured in this simplified `clip_loss` function from the notebook:

```
# 4. CONTRASTIVE LOSS (CLIP Loss)
def clip_loss(image_features, text_features, temperature=0.07):
    # Calculate similarity scores between all pairs
    # logits shape: [batch_size, batch_size]
    logits = (image_features @ text_features.T) / temperature

    # Create the "correct" labels (the diagonal: 0, 1, 2, 3...)
    labels = torch.arange(len(logits), device=device)

    # Calculate loss for (Image -> Text) and (Text -> Image)
    loss_i = F.cross_entropy(logits, labels)
    loss_t = F.cross_entropy(logits.T, labels)

    # Return the average (symmetric) loss
    return (loss_i + loss_t) / 2
```