CLIP: Contrastive Language-Image Pre-Training

Paper Title: Learning Transferable Visual Models From Natural Language Supervision

Paper link: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://proceedings.mlr.press/v139/radford21a/radford21a.pdf

Intro

- Multimodal learning architecture developed by OpenAI
- Bridges the gap between text and visual data
- By jointly training a CLIP model on a large-scale dataset containing images and their corresponding textual descriptions
- Similar to the zero-shot capabilities of GPT-2 and GPT-3
- Effectiveness comes from a large-scale, diverse dataset of images and texts.

Intro

- CLIP is trained on a vast dataset containing 400 million image-text pairs collected online.
- This extensive training data allows it to generalize across various domains and tasks.
- One of CLIP's standout features is its ability to perform zero-shot learning.
- *It can handle new tasks without requiring task-specific training data, simply by understanding the task description in natural language. How? Answer this at the end of the lecture!*

CLIP uses **self-supervised learning** because it learns from the natural pairings of images and text, without needing explicit human-provided labels.

- The 20,000 (image, text) pair cap per query is a crucial step for balancing the dataset.
- By limiting the number of pairs per query, the dataset avoids being dominated by frequent concepts. This ensures greater diversity and prevents the model from overfitting to a small subset of the data.

Natural Language as a training signal

- ➢ Predicting which caption goes with which image
- ➢ Collected a dataset of 400M image, text pairs from the internet
- ➢ Self-supervised pre-training
- ➢ Enabling zero-shot transfer to down-stream tasks

WebImageText (WIT) Dataset

- Constructed a dataset of 400M image/text pairs
- Based on 500000 queries collected from high frequent (+100) words in English Wikipedia
- Balancing the dataset with a cap of 20000 (image, text) pair per query

Motivation:

Supervised learning requires high-quality annotations (gold labels)

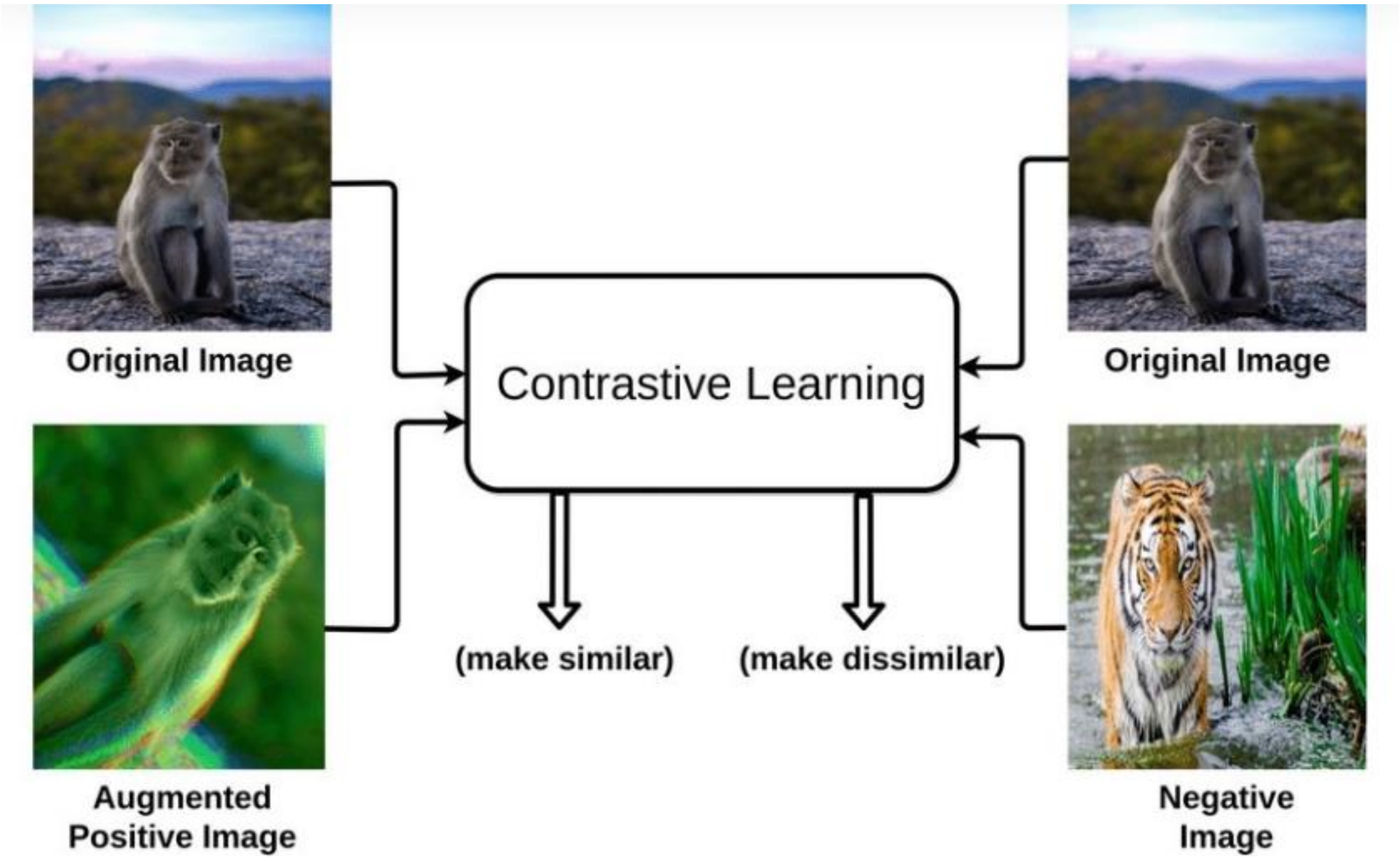
- Difficult to obtain labels in specialized domains
- High annotation cost
- Limited labeled data

➔ Pre-training using self-supervised learning with abundant textual data

What is contrastive learning?

- Contrastive learning is a technique used in machine learning, particularly in the field of unsupervised learning.
  - A method to teach an AI model to recognize similarities and differences of a large number of data points.
- Imagine you have a main item (the “anchor sample”), a similar item (“positive”), and a different item (“negative sample”).
  - The goal is to make the model understand that the anchor and the positive item are alike, so it brings them closer together (with a score or rank) while recognizing that the negative item is different and pushing it away.

https://viso.ai/deep-learning/clip-machine-learning/



https://viso.ai/deep-learning/clip-machine-learning/

Why CLIP?

- Before CLIP, most deep learning models for image classification followed a **supervised learning** approach, where they were trained on labeled datasets like **ImageNet**. These models had several limitations:
- **Dependency on Labeled Data:**
  - Training a classification model required large amounts of manually labeled images (e.g., “dog,” “cat,” “car” in ImageNet).
  - Collecting and labeling data is expensive and time-consuming.
- **Fixed Number of Classes:**
  - A model trained on ImageNet (with 1,000 categories) cannot classify objects outside this predefined set.
  - If a new category is introduced (e.g., “electric scooter”), the model must be retrained.
- **Poor Generalization to New Domains:**
  - If a model is trained on standard dataset images (e.g., stock photos), it struggles with real-world images from different contexts.

How CLIP overcomes this?

- CLIP extends a **new paradigm** where it learns from **image-text pairs** rather than manually labeled datasets. This provides key advantages:
- **Learns from Internet-Scale Data:**
  - Instead of training on predefined labels, CLIP learns from millions of images with natural language descriptions found on the internet.
  - This allows it to generalize well to unseen objects without additional training.
- **Zero-Shot Learning Capability:**
  - CLIP doesn’t require explicit training for specific tasks.
  - Once trained, it can classify images into **any category specified by natural language** without needing extra labeled data.
  - Example: A CLIP model trained on internet images can classify “Tesla Cybertruck” without ever seeing a labeled image of it.
- **Flexible Text-Based Classification:**
  - Traditional classifiers assign images to fixed categories (e.g., “dog,” “cat”), while CLIP allows classification **by providing text prompts**.
  - Example: Instead of defining fixed classes, you can query CLIP with any text label:
    - *“This is a picture of a lion”* vs. *“This is a picture of a tiger”*
    - CLIP ranks the text descriptions based on similarity to the image.

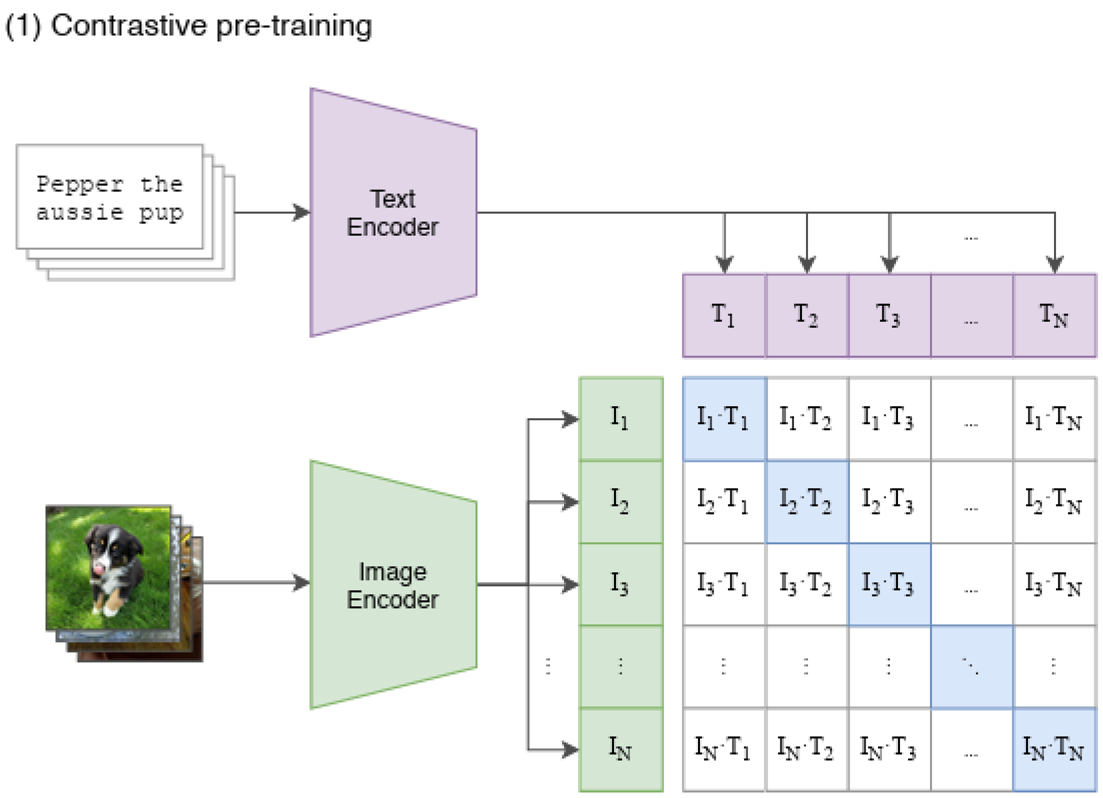
CLIP Architecture (Training)

- CLIP uses a dual-encoder architecture to map images and text into a shared latent space.
  - It works by jointly training two encoders. One encoder for images (Vision Transformer) and one for text (Transformer-based language model).

- **Image Encoder:** The image encoder extracts salient features from the visual input. This encoder takes an 'image as input' and produces a high-dimensional vector representation. It typically uses a ViT or ResNet, for extracting image features.

- **Text Encoder:** The text encoder encodes the semantic meaning of the corresponding textual description. It takes a 'text caption/label as input' and produces another high-dimensional vector representation. It often uses a transformer-based architecture, like a Transformer or BERT, to process text sequences.

- **Shared Embedding Space:** The two encoders produce embeddings in a shared vector space. These shared embedding spaces allow CLIP to compare text and image representations and learn their underlying relationships.



- **Objective** of Contrastive Learning:
- Maximize similar image-text pair embeddings
- Minimize dissimilar image-text pair embeddings

- CLIP's architecture consists of two main components:

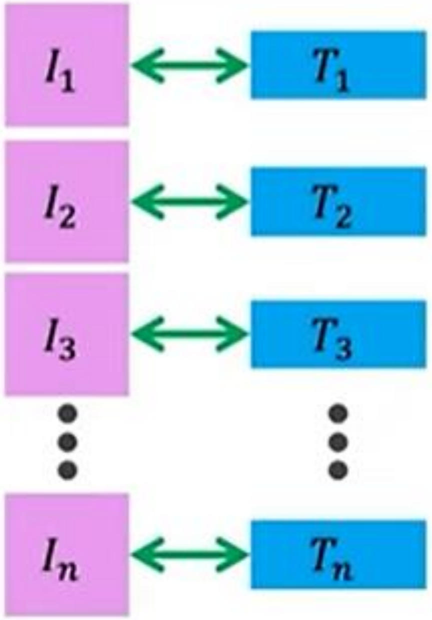
1. **A Vision Encoder** (for processing images)
2. **A Text Encoder** (for processing text descriptions)

- Both of these encoders map their respective inputs into a **shared embedding space**, where similar images and text align closely.

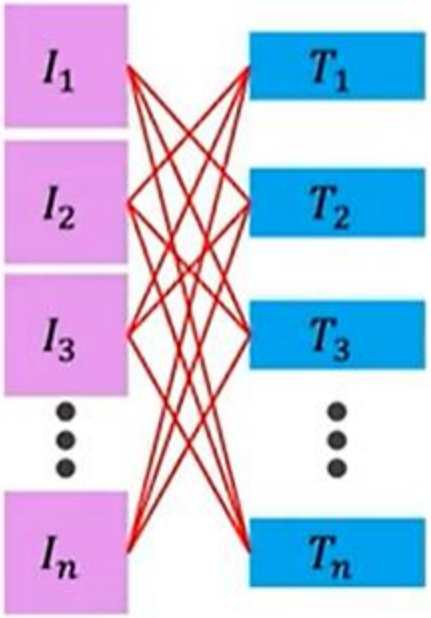
CLIP Objective: Contrastive loss on (image, text) pairs

Which text (as a whole, not word-by-word) goes with which image

- Increase the cosine similarity of **correct** pairs in a batch



- Reduce the cosine similarity of $n^2 - n$ **incorrect** pairings



How do u get $n^2 - n$ incorrect pairings?

https://www.youtube.com/watch?v=jXD6O93Ptks

- Let's say you have a batch size of *n = 4* image-text pairs.

- **Total Pairings (n^2):** 4 * 4 = 16
- **Correct Pairings (n):** 4
- **Incorrect Pairings (n^2 - n):** 16 - 4 = 12
- **Why n^2 - n is Important for CLIP:**

- CLIP's training objective is to maximize the similarity between the correct (positive) pairs and minimize the similarity between the incorrect (negative) pairs.
- The $n^2 - n$ incorrect pairings provide ***a large number of negative examples*** that help the model ***learn to distinguish between matching and non-matching image-text combinations.***
- This contrastive learning process is what allows CLIP to develop a robust understanding of the relationship between images and text.

Vision Encoder (ViT or CNN)

- For image encoding, CLIP primarily uses a **Vision Transformer (ViT)**, although it can also use a **ResNet**. The process:

1. The image is passed through **ViT**.
2. ViT **converts the image into smaller patches** (like dividing the image into grids).
3. These patches are **linearly embedded and processed through Transformer layers** to extract deep visual features.
4. The final output is an **image embedding vector** that represents the image in a high-dimensional space.

- This embedding captures the semantic content of the image—objects, textures, relationships—without requiring explicit labels.

Text Encoder (Transformer for NLP/caption)

- For text encoding, CLIP uses a **Transformer-based language model** (similar to GPT or BERT) to process the textual description.

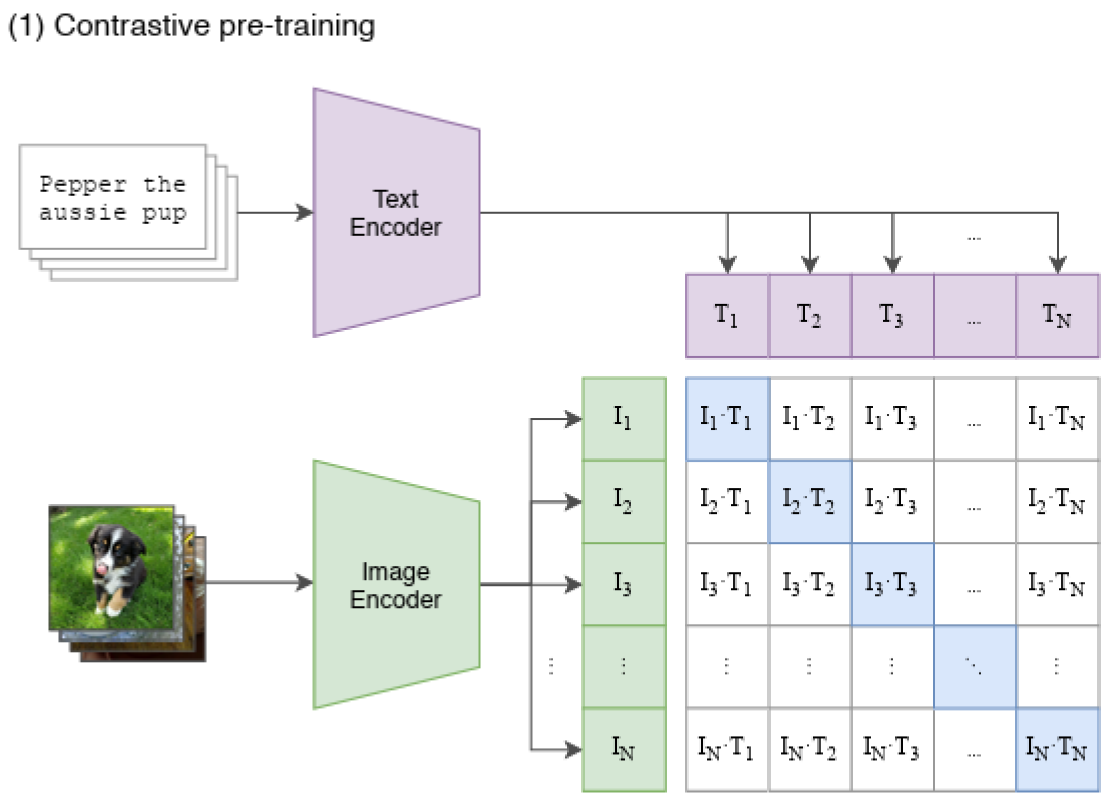
1. The input text (e.g., *“A photo of a cat”*) is tokenized into individual words (e.g., [A, photo, of, a, cat]).
2. These tokens are passed through an **embedding layer** that converts them into dense numerical vectors.
3. The **Transformer processes the tokens** using self-attention mechanisms to understand their relationships.
4. The **final text representation** is obtained from the last layer of the Transformer.

Single Embedding for the Whole Caption?

- **Transformer Output**: The text transformer processes the caption and produces a sequence of output embeddings, one for each word/token.

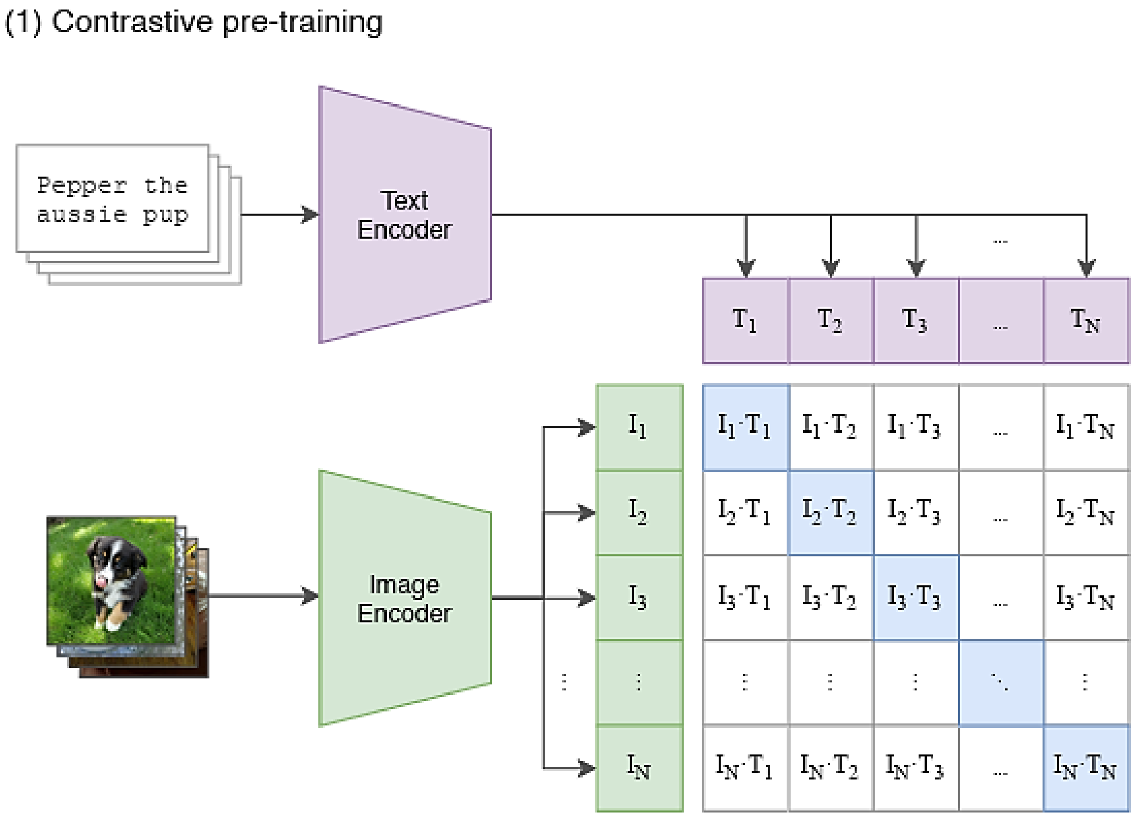
- **Mean Pooling:** CLIP then applies mean pooling to these output embeddings. This means it simply averages the embeddings across all tokens in the caption (including any special start or end tokens).

- **Text Embedding:** The result of this mean pooling is the final text embedding used by CLIP.
- The authors experimented with using a [CLS] token but found that mean pooling performed just as well or even better in their experiments. Mean pooling is simpler and computationally less expensive than training a dedicated [CLS] token.



At the end of the encoding process:

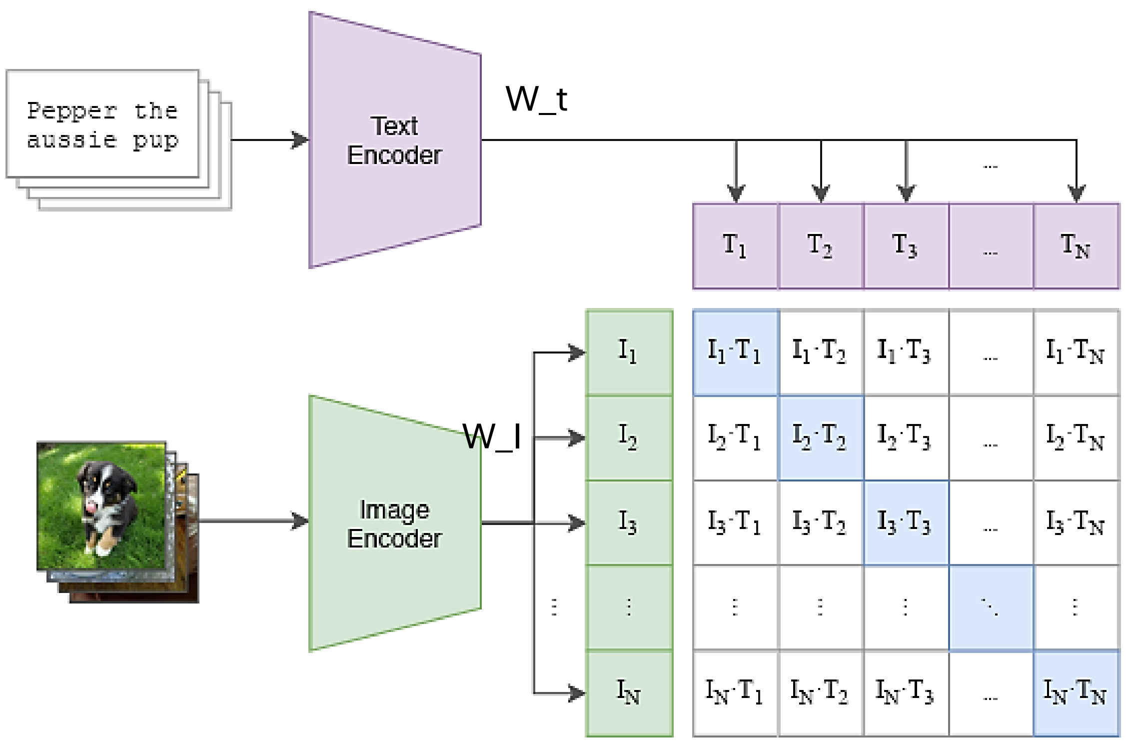
- The **Vision Encoder** provides an **image embedding**.
- The **Text Encoder** provides a **text embedding**.
- A linear layer is applied at both ends (image n text encoder) to match the dimensions for text and image.
- Both embeddings exist in the **same space**, allowing comparison.
- It then **compares every image** with **every text description** using cosine similarity.
- The model is trained to **maximize similarity** for the **correct** image-text pair and **minimize** similarity for incorrect pairs.
- Cosine similarity for the correct image-caption pair will be higher and lower for the incorrect pairs.
- The diagonal values represent the correct image-caption pair.



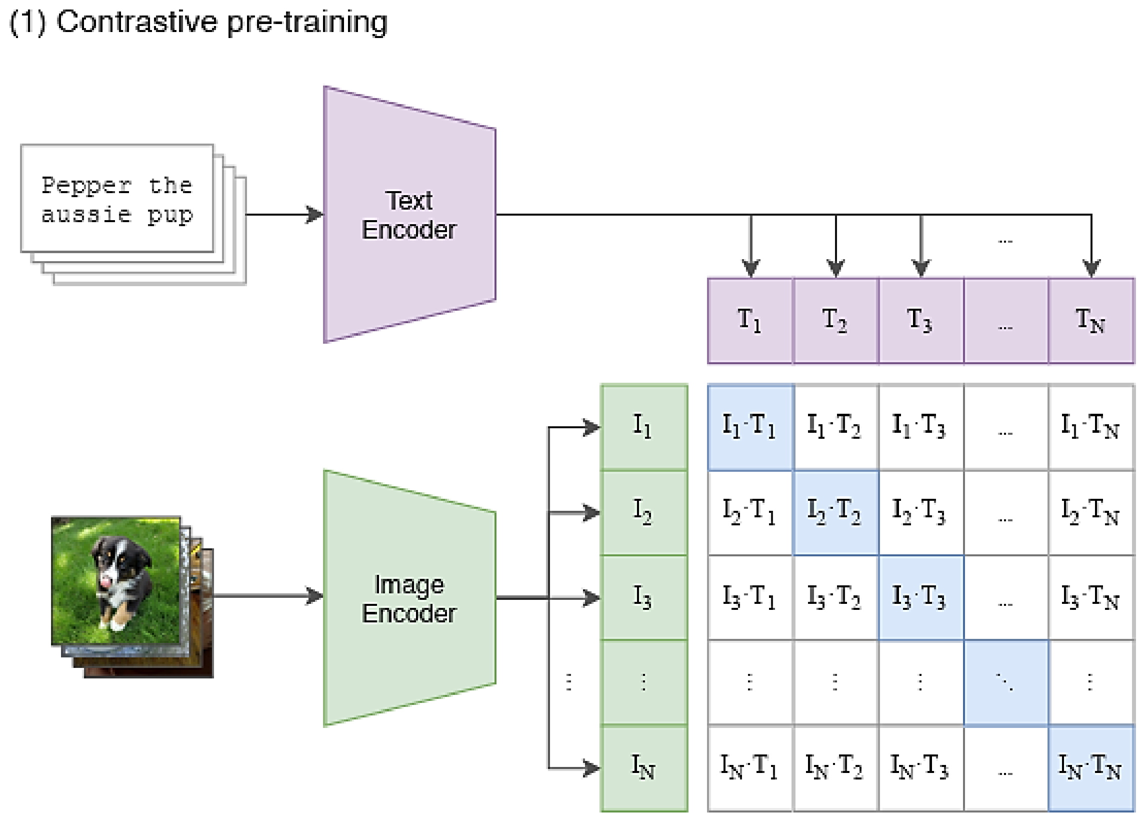

```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

images and text captions are mapped into a **joint embedding space** and trained using a **contrastive loss**.

(1) Contrastive pre-training




```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```




```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

We normalize the embeddings to have **unit length (||x|| = 1)** so that:

1. **Cosine similarity becomes equivalent to the dot product** when the vectors are unit vectors.
2. **Distances become comparable** across different embeddings, bcz on same scale (magnitude 1).
3. **Prevents large embeddings from dominating similarity scores** due to magnitude differences.

axis=1 → Normalizes **each row independently** (each sample).
L2 normalization enforces **unit vectors**, making cosine similarity a **dot product**.
Ensures similarity is **only based on direction**, not magnitude.

cosine_similarity(a, b) = $a \cdot b$ / (||a|| * ||b||)

Since ||a|| = ||b|| = 1 (due to L2 normalization), the equation simplifies to:

cosine_similarity(a, b) = $a \cdot b$

Example: L2 Normalization of a Vector

Suppose we have a **vector**:

$$A = [3, 4]$$

To normalize it, we compute its **L2 norm** (also called **Euclidean norm**):

$$\|A\| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

Now, we **divide each element by the L2 norm**:

$$A_{\text{normalized}} = \left[\frac{3}{5}, \frac{4}{5}\right] = [0.6, 0.8]$$

✅ **Now, this is a unit vector because its magnitude is 1:**

$$\sqrt{(0.6)^2 + (0.8)^2} = \sqrt{0.36 + 0.64} = \sqrt{1} = 1$$

1. **L2 Norm Calculation:** The L2 norm (or Euclidean norm) of a vector $v = [v_1, v_2, ..., v_n]$ is calculated as:

   $||v|| = \sqrt{(v_1^2 + v_2^2 + ... + v_n^2)}$

2. **L2 Normalization:** L2 normalization involves dividing each element of the vector by its L2 norm:

   $v\_normalized = [v_1/||v||, v_2/||v||, ..., v_n/||v||]$

3. **Magnitude of Normalized Vector:** Now, let's calculate the magnitude of the normalized vector:

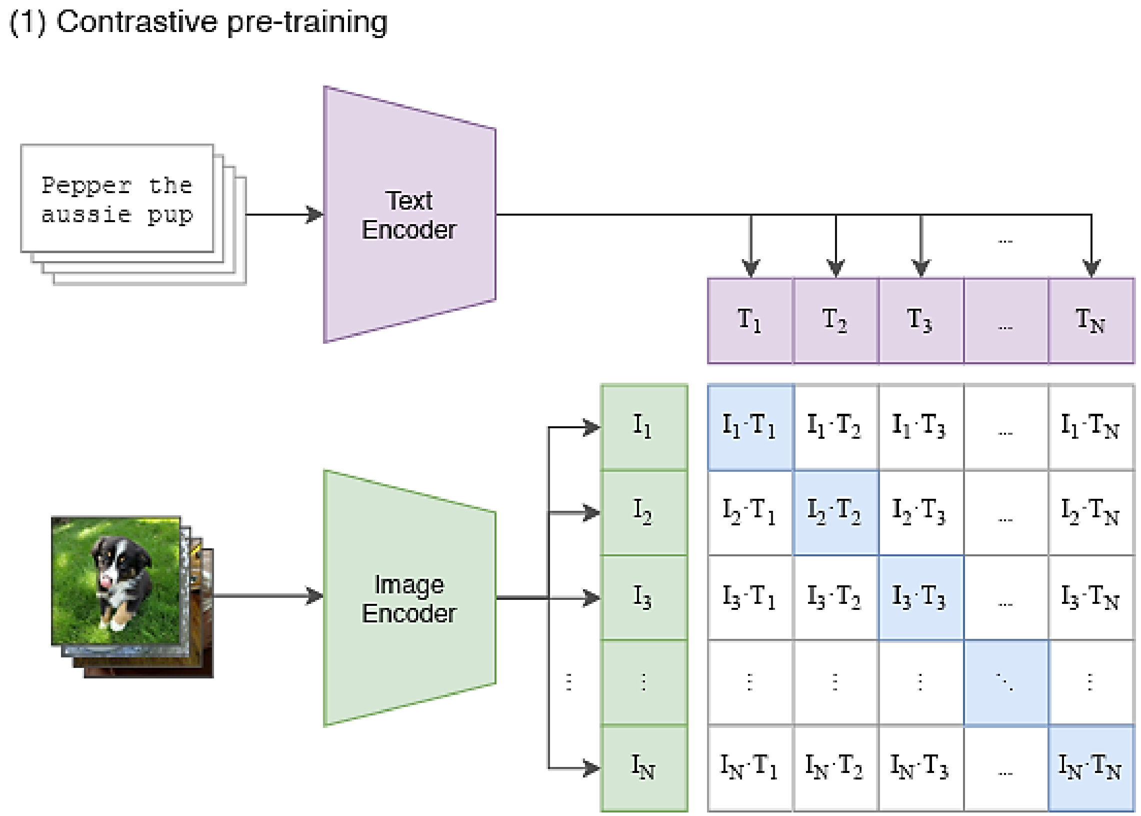
   $||v\_normalized|| = \sqrt{((v_1/||v||)^2 + (v_2/||v||)^2 + ... + (v_n/||v||)^2)}$

   $= \sqrt{((v_1^2 + v_2^2 + ... + v_n^2) / ||v||^2)}$

   $= \sqrt{(||v||^2 / ||v||^2)}$

   $= \sqrt{1}$

   $= 1$

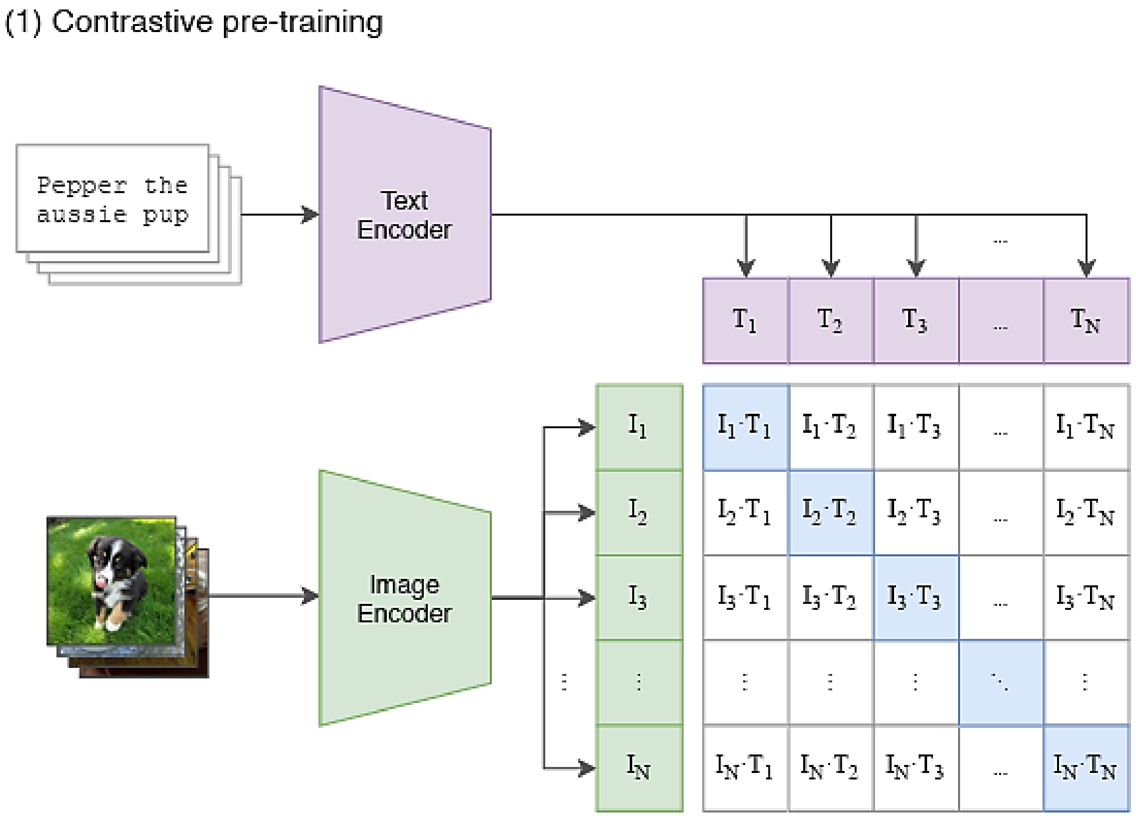



```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

Temperature T

Higher T → More **uniform distribution**, better **generalization**, but may hinder learning distinctions.
Lower T→ More **sharp distinctions but** can lead to **overfitting**.
T is learnable → The model **optimizes it automatically** for best performance.

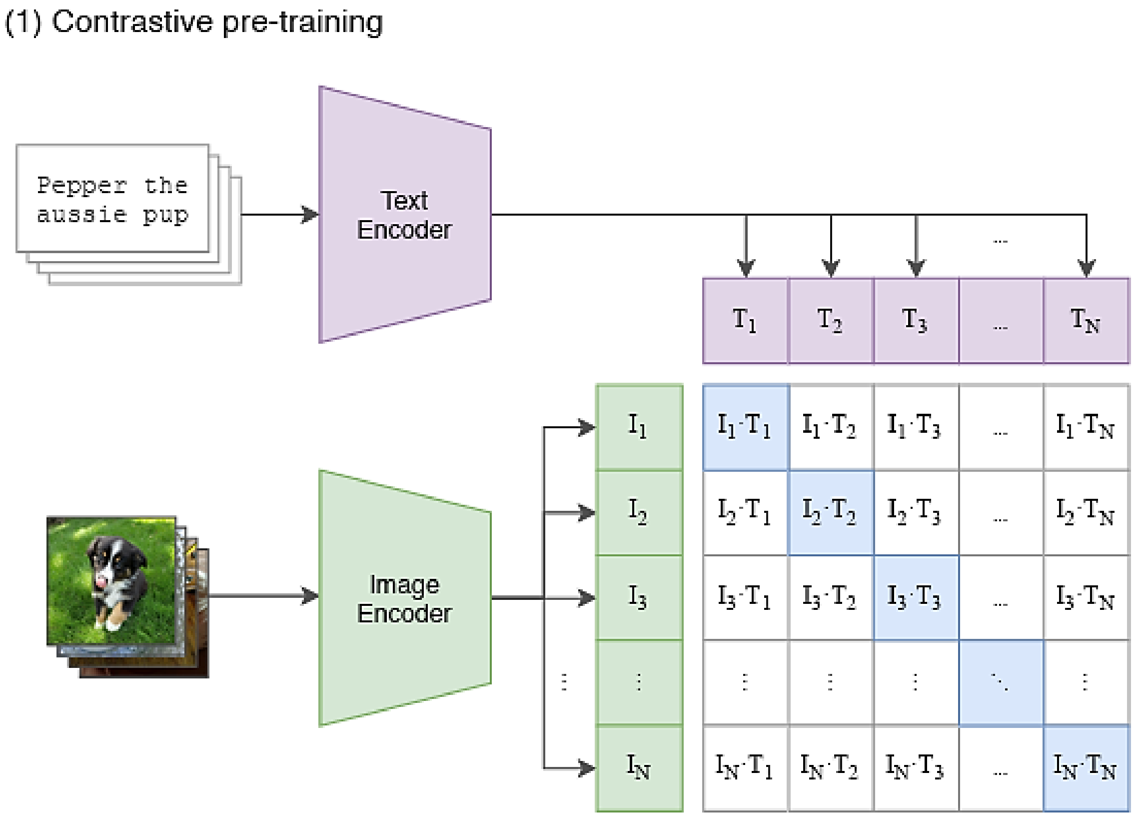
The temperature parameter plays a crucial role in *calibrating* the model's predictions. Calibration means that the predicted probabilities reflect the model's true confidence in its predictions. A well-calibrated model will have predicted probabilities close to 1 for correct predictions and close to 0 for incorrect predictions.




```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

What will happen if we remove temperature? How will the learning be impacted?

The temperature parameter plays a crucial role in *calibrating* the model's predictions. Calibration means that the predicted probabilities reflect the model's true confidence in its predictions. A well-calibrated model will have predicted probabilities close to 1 for correct predictions and close to 0 for incorrect predictions.

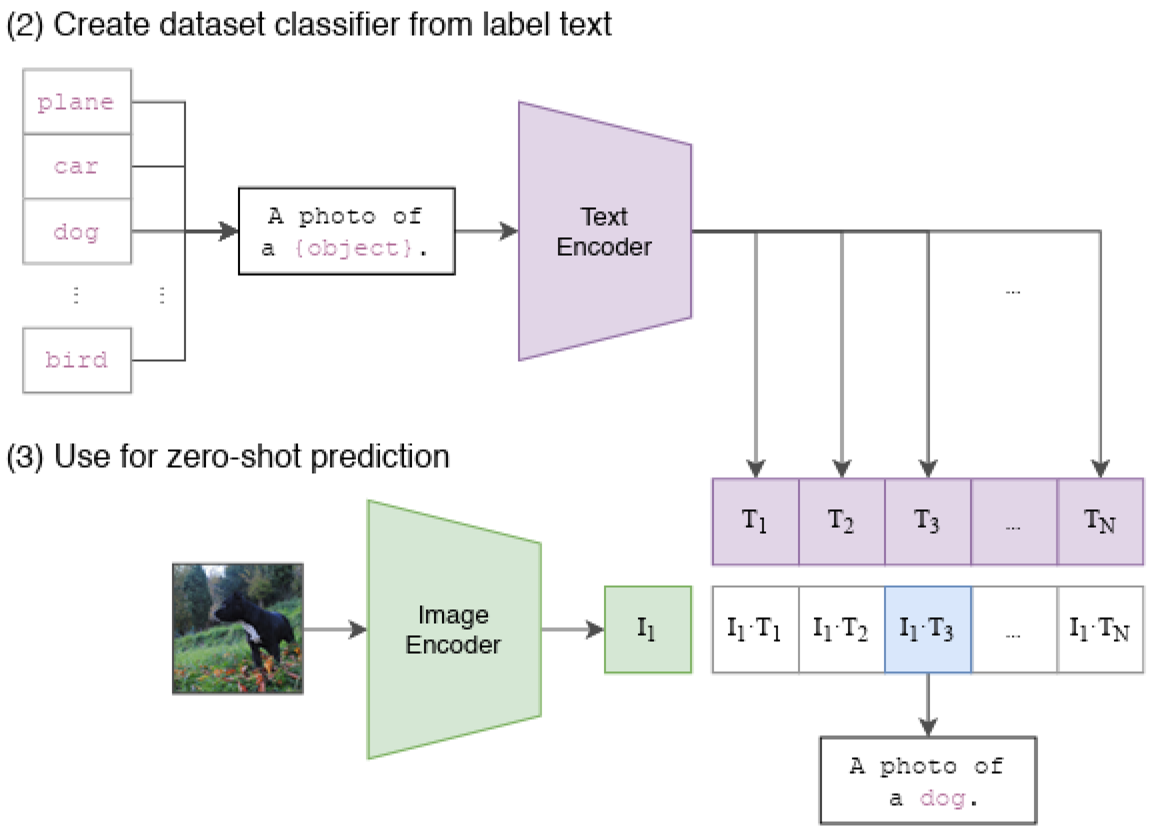



```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

•labels = np.arange(n): Each image should be most similar to **its corresponding text** (diagonal elements of logits).
•cross_entropy_loss(logits, labels, axis=0): Computes **image-to-text** contrastive loss.
•cross_entropy_loss(logits, labels, axis=1): Computes **text-to-image** contrastive loss.

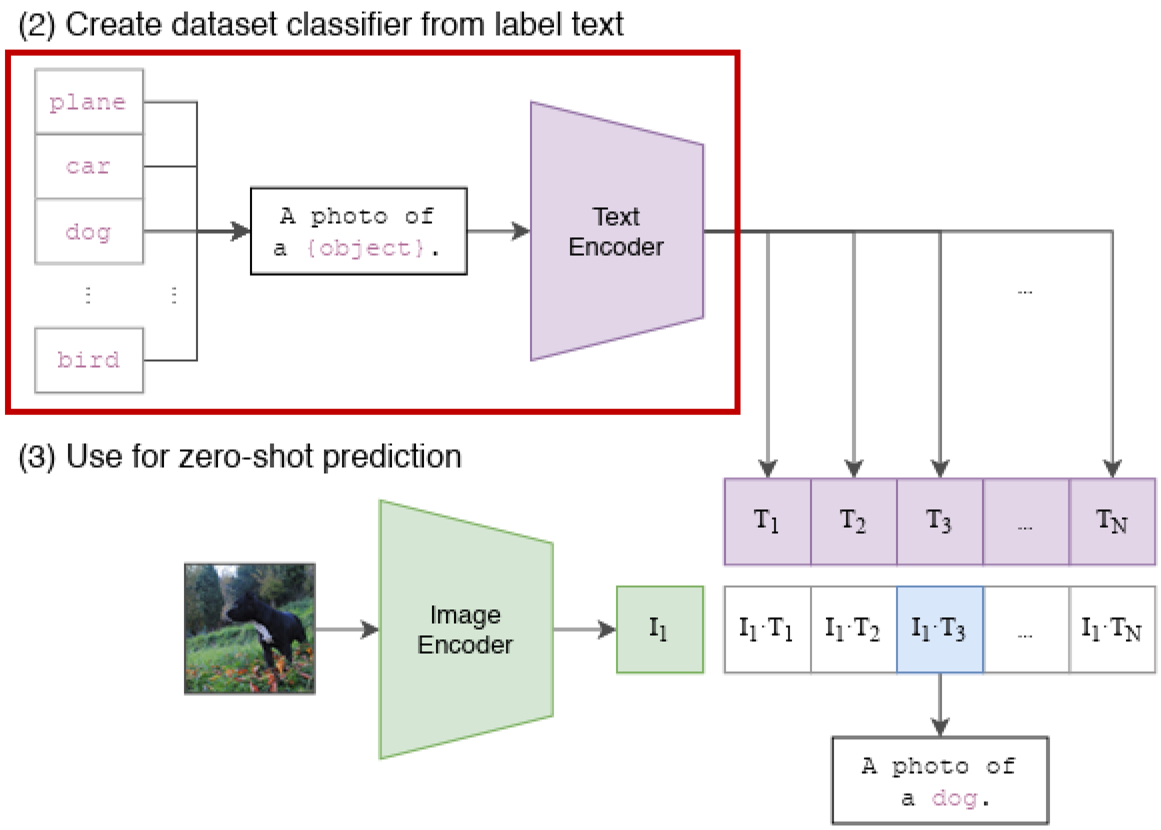
•**Why Both Losses?** Ensures **both** modalities (images and texts) align properly.
•loss = (loss_i + loss_t) / 2: Averages the losses to treat both directions symmetrically.

Inference: CLIP



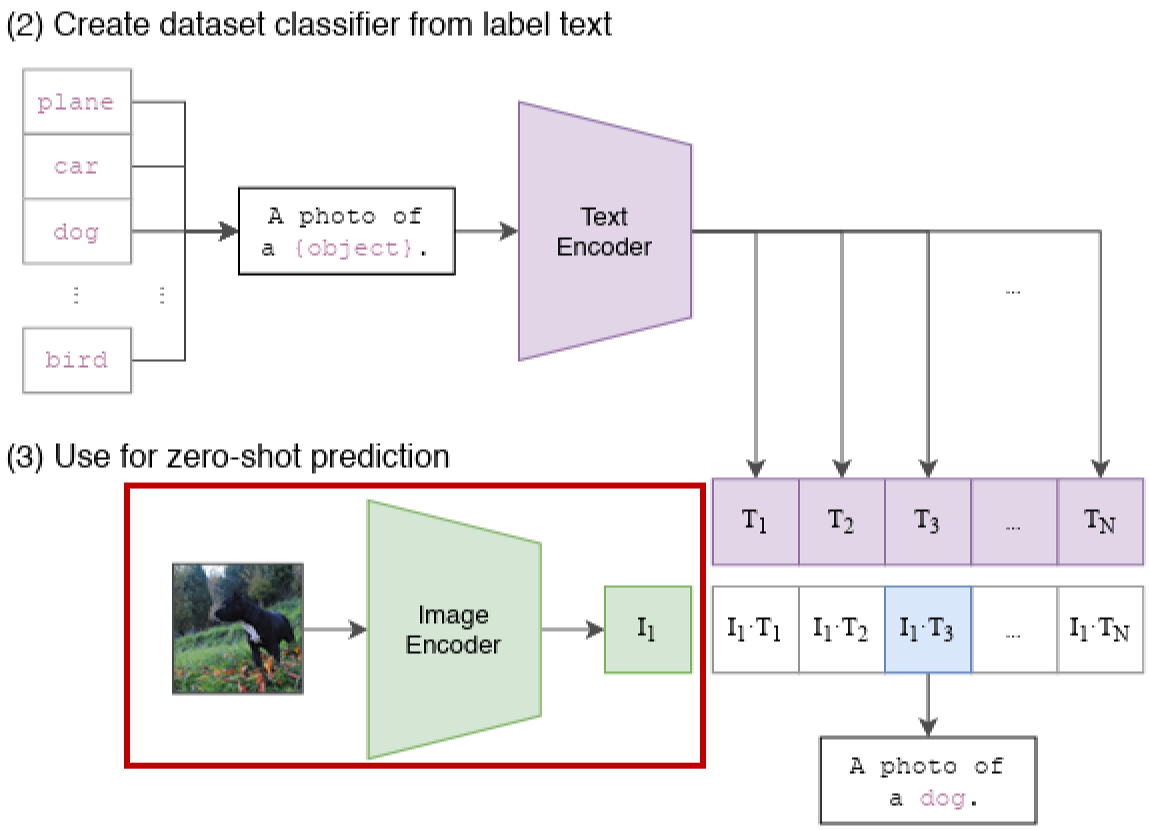
- At test time the **learned** text encoder synthesizes a **zero-shot linear classifier** by embedding the names or descriptions of the target dataset's classes.
- **Zero Short Learning:** the **model** can classify new images without being **explicitly trained** on labeled examples from that category.
- Instead of relying on a **fixed classifier** trained with labeled data, **CLIP can predict any category** by comparing images to textual descriptions.

Inference: CLIP



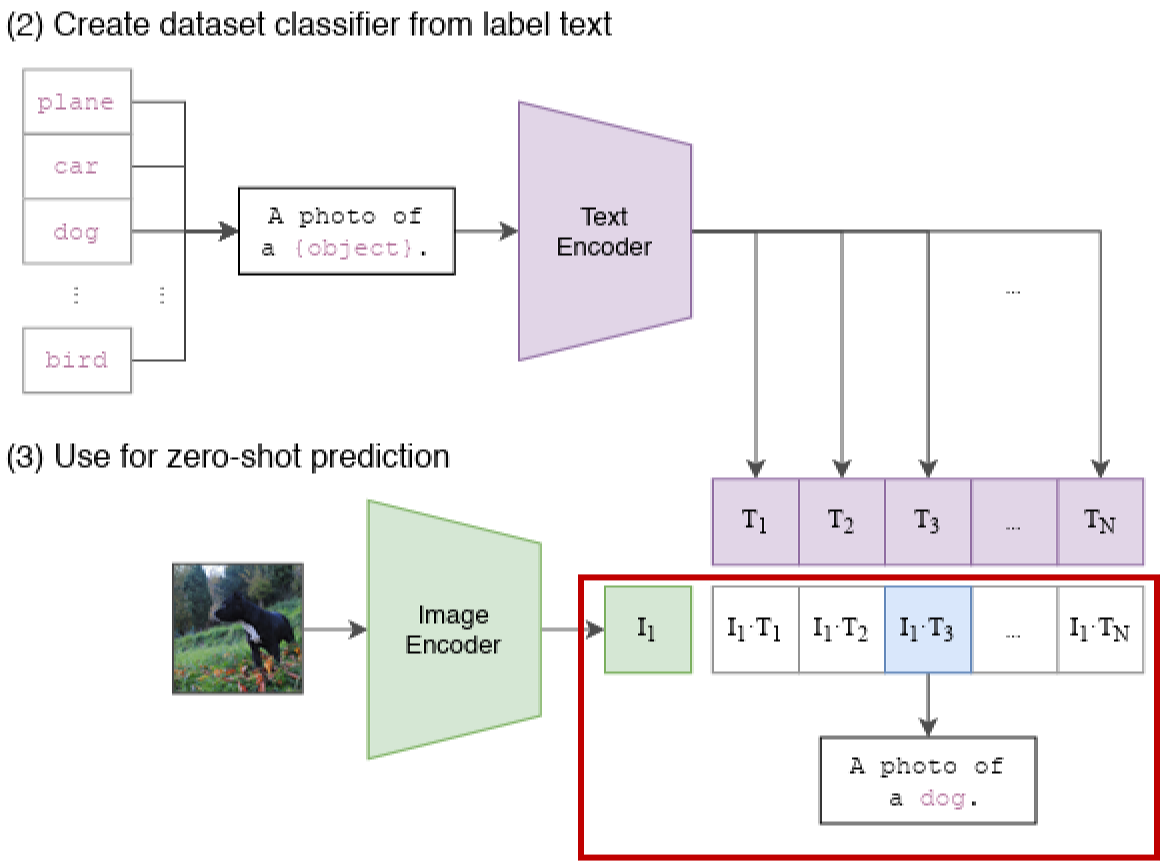
- **Step 1: Constructing Text-Based Classifier**
- We are given a set of class labels (e.g., "dog", "cat", "bird", "car").
- Instead of training a classifier, we convert these class names into text prompts, such as:
- “A photo of a dog.”
- “A photo of a cat.”
- “A photo of a car.”
- These prompts are then passed through the Text Encoder, which converts them into text embeddings (T1, T2, …, TN).
- These embeddings act as class prototypes—i.e., they represent what it means to be a dog, cat, car, etc., in vector space.

Inference: CLIP



- **Step 2: Encoding** the Image
- When we input an **unseen image** (e.g., an image of a dog), the **Image Encoder** (a ViT or ResNet) **extracts its feature embedding** I1
- Step 3: Compute **Cosine Similarity** Between **Image** and **Text** Embeddings

Inference: CLIP



- **Step 2: Encoding** the Image
- When we input an **unseen image** (e.g., an image of a dog), the **Image Encoder** (a ViT or ResNet) **extracts its feature embedding** I1
- Step 3: Compute **Cosine Similarity** Between **Image** and **Text** Embeddings
- We compute the **cosine similarity** between the image embedding I1 and all text embeddings T1,T2,...,TN.
- This results in a **similarity score matrix**:

$I_1 \cdot T_1$ (similarity with “a photo of a plane”)

$I_1 \cdot T_2$ (similarity with “a photo of a car”)

$I_1 \cdot T_3$ (similarity with “a photo of a dog”)

- The most similar text embedding is chosen as the **predicted class.**
- I1·T3 (for “a photo of a dog”) has the **highest score**, so the model predicts **"dog"**.

Food101

guacamole (90.1%) Ranked 1 out of 101 labels

✓ a photo of **guacamole**, a type of food.

× a photo of **ceviche**, a type of food.

× a photo of **edamame**, a type of food.

× a photo of **tuna tartare**, a type of food.

× a photo of **hummus**, a type of food.

SUN397

television studio (90.2%) Ranked 1 out of 397 labels

✓ a photo of a **television studio**.

× a photo of a **podium indoor**.

× a photo of a **conference room**.

× a photo of a **lecture room**.

× a photo of a **control room**.

PatchCamelyon (PCam)

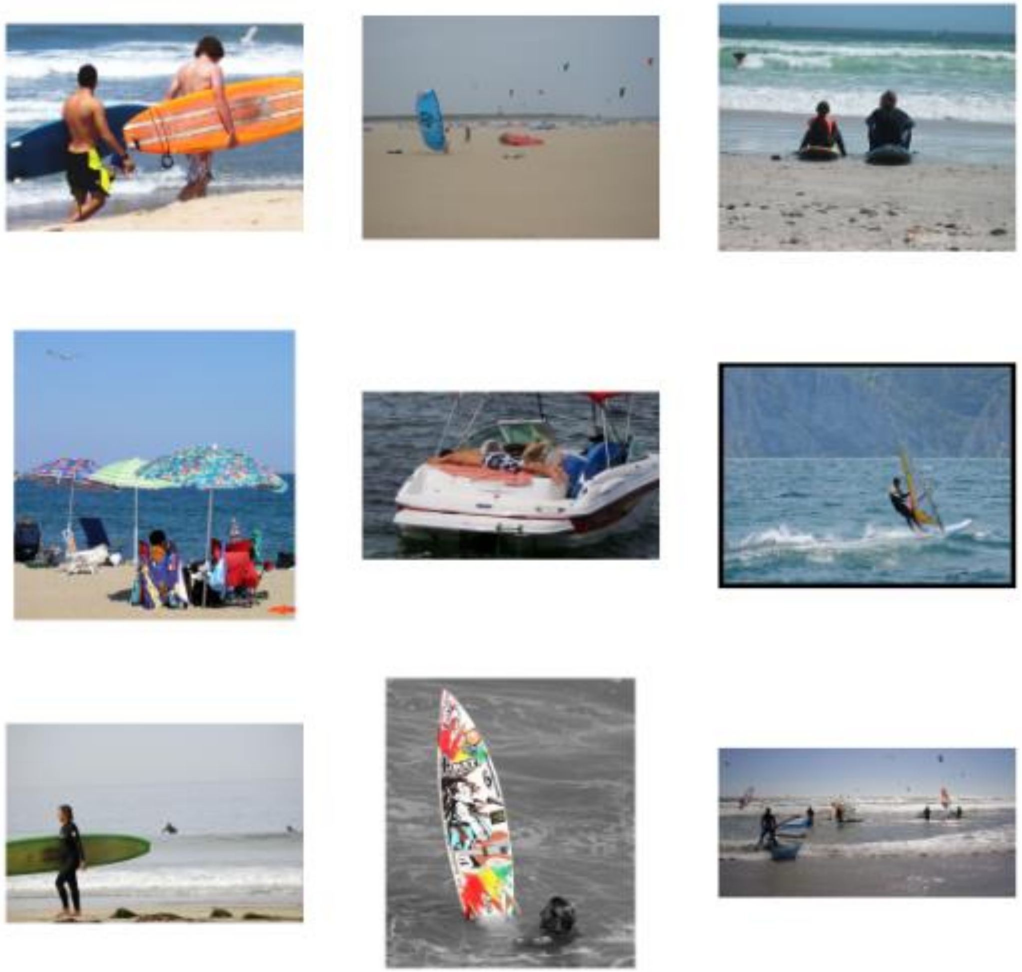
healthy lymph node tissue (77.2%) Ranked 2 out of 2 labels

× this is a photo of **lymph node tumor tissue**

✓ this is a photo of **healthy lymph node tissue**

https://openai.com/index/clip/

```
query = "a family standing next to the ocean on a sandy beach with a surf board"
matches = find_matches(image_embeddings, [query], normalize=True)[0]

plt.figure(figsize=(20, 20))
for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(mpimg.imread(matches[i]))
    plt.axis("off")
```



Outputs top 9 images against this caption
It doesn't generate but ranks the images

https://keras.io/examples/vision/nl_image_search/#evaluate-the-retrieval-quality

How CLIP generalizes?

- During training, it might have seen captions like:
  - *"A cute animal is running."*
  - *"A fluffy pet is playing in the grass."*
  - *"A furry mammal with four legs."*
- Even if "dog" was never explicitly labeled, the model **learned to associate visual patterns with textual concepts**.
- Now, at test time, when we provide the prompt **"A photo of a dog."**, CLIP **compares it with the image** and recognizes that this text is most similar to what it has learned about similar animals.

- The **output** of the CLIP model is usually a set of **similarity scores** or **similarity rankings**.
- CLIP **does not generate captions** by itself, but it can **rank or match existing captions to images**.
- Its output is usually a **similarity score** or ranking, showing how well each caption or class matches the image.
- **Image-to-Text (or Text-to-Image) Matching**: You provide an image and a set of textual descriptions (like class names, captions, etc.), and CLIP will predict which description is most relevant to the image.
- **Zero-shot Learning**: Since CLIP has been trained on vast datasets with both images and text, it can generalize to new tasks without needing further fine-tuning.
    - For example, you can give it a new set of images and text that it has never seen before, and it will still be able to rank the matching relevance between them.

Applications

- **Image Classification**: Given a set of possible categories (e.g., "dog," "cat," "car"), CLIP can tell you which category is most likely for a given image.
- **Image-to-Text Search (Retrieval)**: You can input an image and a large set of text captions and ask CLIP to retrieve the most relevant caption for the image. Essentially, CLIP can act like an **image search engine** using natural language descriptions.
- **Text-to-Image Search (Retrieval)**: Similarly, you can input a **text** and a set of **images**, and CLIP can rank the images based on which one best matches the text query.
- **Zero-shot Learning**: CLIP excels in tasks where you provide new categories or descriptions (that it hasn't specifically been trained on), and it can still perform well without needing extra training.
- **Cross-modal Retrieval**: This is when you search for one modality using another. For example, you can search for images using text queries or search for text descriptions using images.

Limitations & Challenges of CLIP

- **Bias in Training Data:**
  - Since CLIP learns from unfiltered internet data, it inherits biases present in the dataset, such as gender stereotypes, racial biases, or associating certain objects with specific demographics.
- **Dependence on Image-Text Alignment:**
  - Works best when image-text pairs are well-aligned; struggles when captions are ambiguous, incomplete or inaccurate.
- **Lack of Fine-Grained Understanding:**
  - CLIP focuses on high-level concepts but may miss details in complex images. For example, it might struggle to differentiate between closely related species of birds or identify specific artistic styles in paintings.
- **Sensitivity to Prompts**
  - CLIP's performance can be sensitive to the specific wording of the text prompts used for zero-shot classification or image retrieval. Slight changes in wording can sometimes lead to significant changes in the model's output.

- https://medium.com/towards-data-science/simple-implementation-of-openai-clip-model-a-tutorial-ace6ff01d9f2
- https://openai.com/index/clip/
- https://summergeometry.org/sgi2024/a-deeper-understanding-openais-clip-model/
- https://viso.ai/deep-learning/clip-machine-learning/
- https://medium.com/@paluchasz/understanding-openais-clip-model-6b52bade3fa3
- https://www.youtube.com/watch?v=jXD6O93Ptks

Explore code:

- https://github.com/moein-shariatnia/OpenAI-CLIP/blob/master/OpenAI%20CLIP%20Simple%20Implementation.ipynb