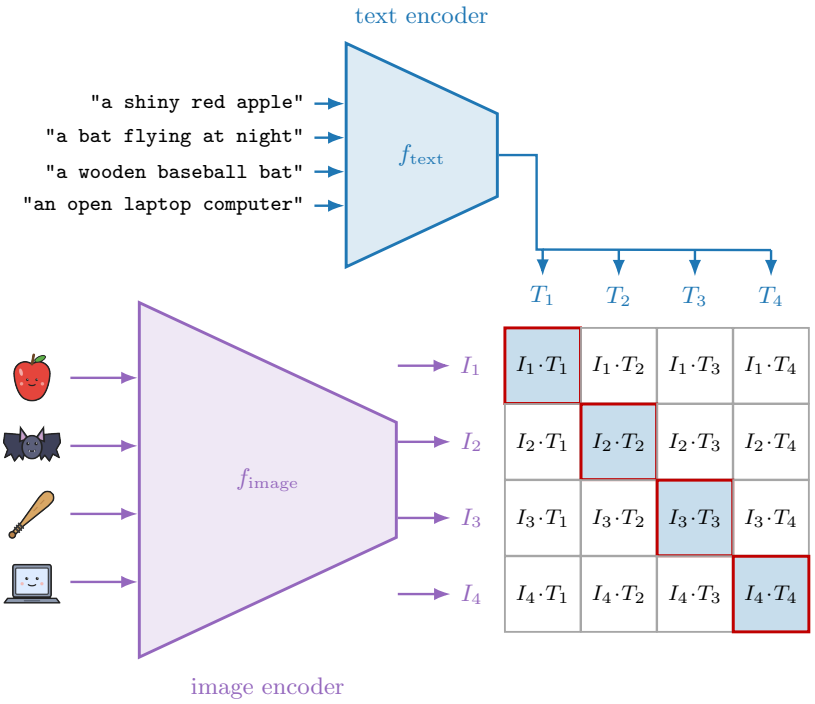
07-380 Notes
Pre-reading: Multimodal Learning

Carnegie Mellon University
School of Computer Science

Contents

1 CLIP: Connecting Text and Images **2**
1.1 Reminder: Word Embeddings . . . 2
1.2 Embedding Words and Images . . . 3
1.3 Bi-directional Loss . . . 3
1.4 Neural Network Encoders . . . 6
1.5 CLIP Training Data . . . 6
1.6 From Our Example to the CLIP Paper . . . 7



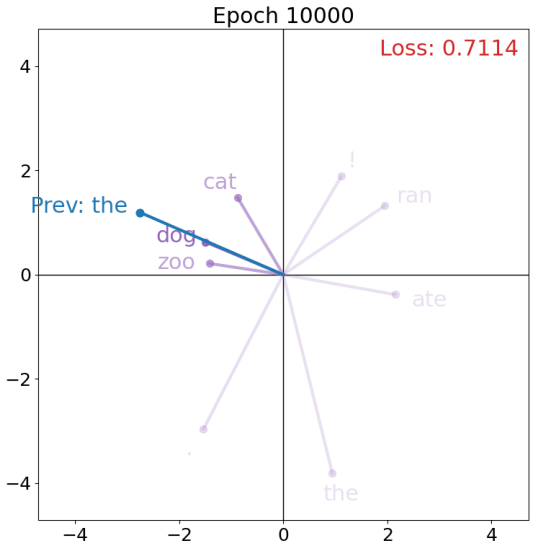
CLIP in one picture: a text encoder and an image encoder trained so that the matching (text, image) pairs in a batch, the diagonal of the similarity matrix, score higher than every other pairing (Section 1.5).

1 CLIP: Connecting Text and Images

A **modality** is a kind of data: text, images, audio, video, robot joint angles. Every model in this course so far has used one modality at a time. **Multimodal learning** connects two or more. These notes cover the simplest connection: a shared vector space in which a word and a picture of the same thing point in the same direction. The model that made this standard is **CLIP** (Contrastive Language–Image Pre-training; Radford et al., 2021), and we build it up from the 07-280 word embeddings.

1.1 Reminder: Word Embeddings

In 07-280 we built a tiny language model out of two embedding matrices. If the details are fuzzy, skim the word embeddings pre-reading and the lecture slides first.



- V and U are $N_{vocab} \times K$ matrices. Row i of V is $\mathbf{v}_i$, the embedding of token i as the previous token (blue); row j of U is $\mathbf{u}_j$, the embedding of token j as the next token (purple). Both start random and are learned.
- To predict the token after `'the'`: look up its index i, take $\mathbf{v}_i$, and score every next-token embedding by its dot product with $\mathbf{v}_i$:

$$\mathbf{s} = U\mathbf{v}_i, \qquad s_j = \mathbf{u}_j^\top \mathbf{v}_i.$$

- $\hat{\mathbf{y}} = g_{\text{softmax}}(\mathbf{s})$ is the probability of each candidate. In the figure each purple vector is faded by its probability; `dog` is the most likely next token.
- The loss for one (previous, next) pair from the corpus is the cross-entropy $J = -\log \hat{y}_{j^*}$, where j^* is the index of the token that actually came next. SGD updates U and V to lower the average loss over the corpus.

In one sentence: **the blue vector is the query, the purple vectors are the candidates, and the candidate that points most in the query's direction wins.** Everything below is a variation on this.

1.2 Embedding Words and Images

Now the query is still a word, but the candidates are images. The running example has four words and four images:

index	1	2	3	4
word (V)	apple	bat	computer	fruit
image (U)				

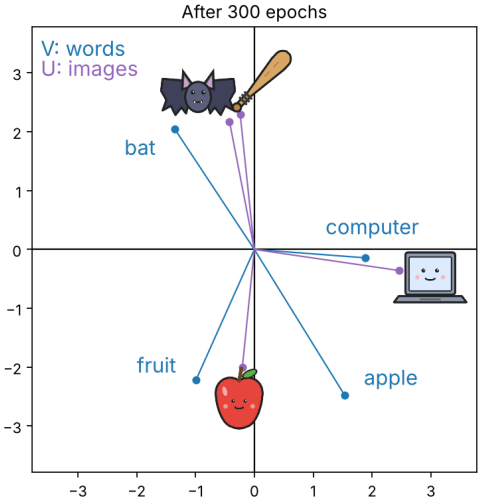
Two of the words have two meanings here: `bat` is the animal or the baseball bat, and `apple` is the fruit or the computer company. `fruit` also goes with the apple image. That gives six matching (word, image) pairs:

(`apple`, ) (`apple`, ) (`bat`, ) (`bat`, ) (`computer`, ) (`fruit`, )

V is as before: row i is the embedding $\mathbf{v}_i$ of word i. U now embeds the images: row j is the embedding $\mathbf{u}_j$ of image j. Both are linear embeddings, exactly like the two matrices of the word embedding model; only the meaning of the rows of U has changed. We use $K = 2$ so that we can draw them. Here are V and U after training (Section 1.3 says how):

```
V (words):
apple:          1.531, -2.472
bat:           -1.351,  2.043
computer:       1.880, -0.143
fruit:         -0.991, -2.218

U (images):
apple:         -0.204, -2.002
bat:           -0.422,  2.171
baseball bat:  -0.241,  2.292
computer:       2.455, -0.358
```

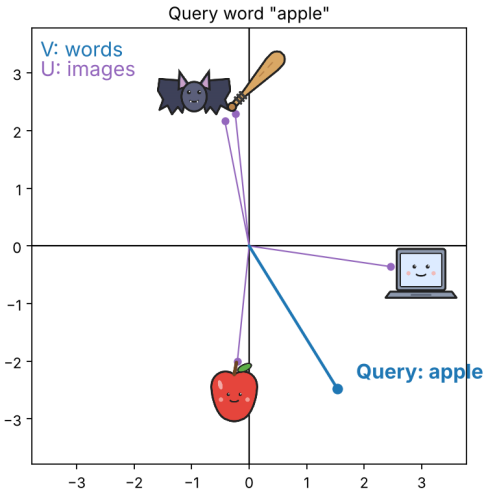


Each word points along the image or images it matches: `fruit` and `computer` along one image each, `apple` and `bat` between their two.

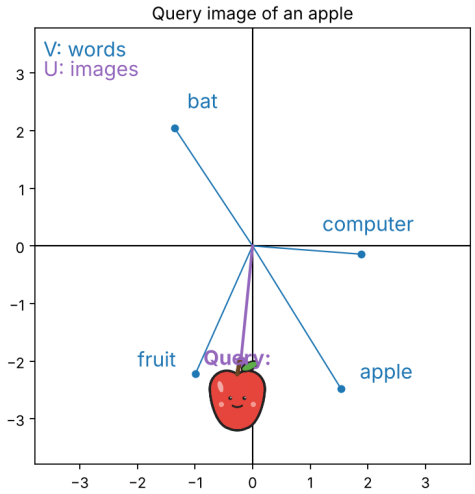
1.3 Bi-directional Loss

Word → image. Given the word `apple`, $i = 1$: take $\mathbf{v}_i$, score every image with $\mathbf{s} = U\mathbf{v}_i$ (so $s_j = \mathbf{u}_j^\top \mathbf{v}_i$), and take the softmax over the images. The trained embeddings split the probability between the apple image and the computer image, the two images that `apple` matches.

Image → word. Given the apple image, $j = 1$: take $\mathbf{u}_j$, score every word with $\mathbf{s} = V\mathbf{u}_j$ (so $s_i = \mathbf{v}_i^\top \mathbf{u}_j$), and take the softmax over the words. The similarity is the same dot product, $\mathbf{v}_i^\top \mathbf{u}_j = \mathbf{u}_j^\top \mathbf{v}_i$; only the set we take the softmax over changes. Here the probability splits between `apple` and `fruit`.

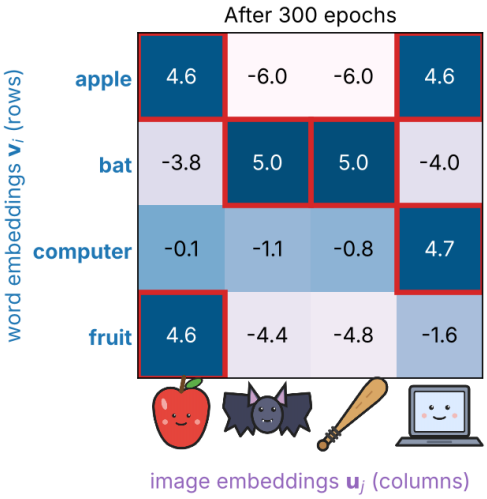


s = U v_apple	score	prob
apple	4.636	0.498
bat	-6.015	0.000
baseball bat	-6.036	0.000
computer	4.645	0.502



s = V u_apple	score	prob
apple	4.636	0.496
bat	-3.814	0.000
computer	-0.098	0.004
fruit	4.642	0.499

Both at once. The **similarity matrix** $S = VU^\top$, with $S_{ij} = \mathbf{v}_i^\top \mathbf{u}_j$, holds both directions: row i is the word $\to$ image score vector $U\mathbf{v}_i$ and column j is the image $\to$ word score vector $V\mathbf{u}_j$. The six matching pairs are outlined in red.



CLIP trains both directions. Number the pairs $n = 1, \ldots, N$ and write $(i^{(n)}, j^{(n)})$ for the word and image indices of pair n. For pair n,

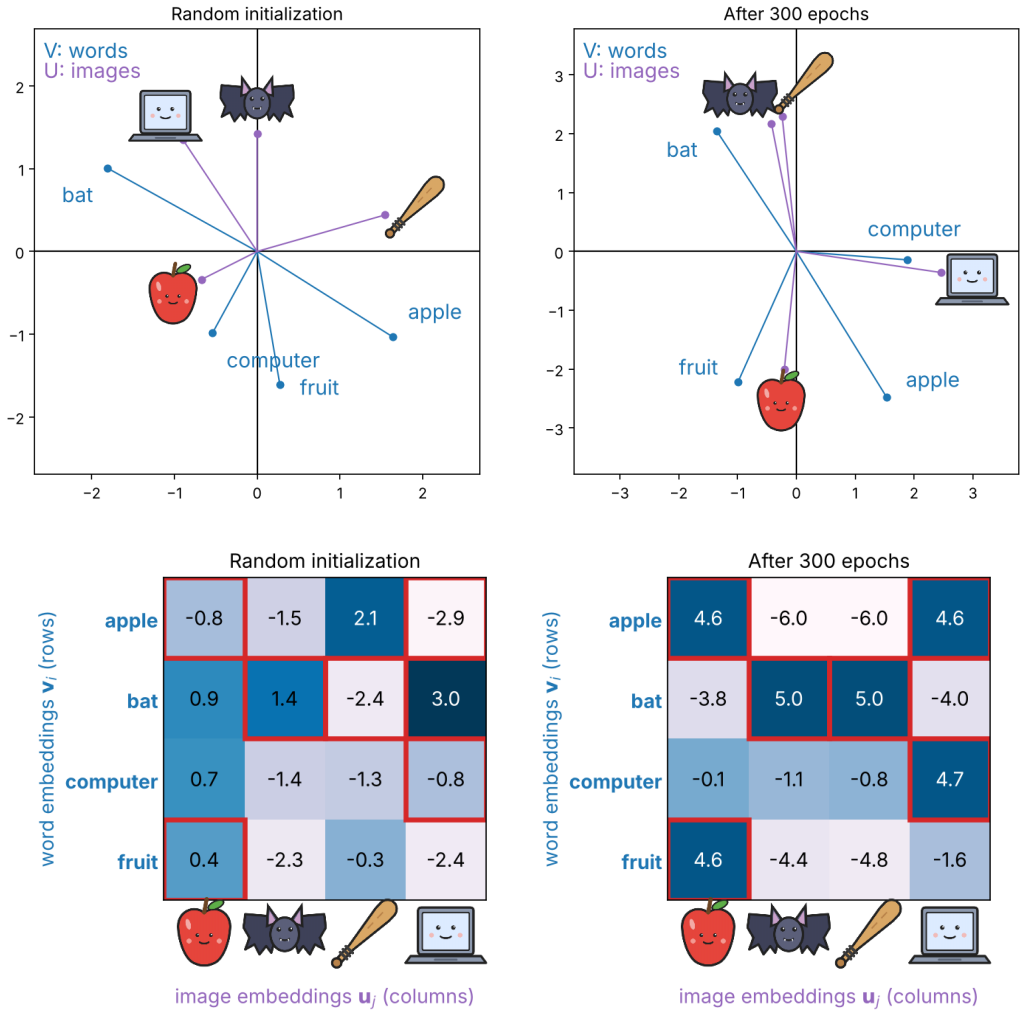
- word $\to$ image: softmax *along row* $i^{(n)}$ of S; the correct answer is entry $j^{(n)}$, the pair's image;
- image $\to$ word: softmax *down column* $j^{(n)}$; the correct answer is entry $i^{(n)}$, the pair's word.

Each direction is a cross-entropy, $-\log$ of the softmax output at the correct answer, as in Section 1.1. The **bi-directional loss** averages over the pairs in each direction and then over the two directions:

$$J(U,V) = \underbrace{\frac{1}{2}\frac{1}{N}\sum_{n=1}^{N} -\log g_{\text{softmax}}\big(U\mathbf{v}_{i^{(n)}}\big)_{j^{(n)}}}_{\text{word}\to\text{image (rows)}} + \underbrace{\frac{1}{2}\frac{1}{N}\sum_{n=1}^{N} -\log g_{\text{softmax}}\big(V\mathbf{u}_{j^{(n)}}\big)_{i^{(n)}}}_{\text{image}\to\text{word (columns)}},$$

with $N = 6$ pairs here. For the trained embeddings above, $J = 0.465$.

Training. Gradient descent on every entry of U and V, starting from random embeddings. Every epoch uses all six pairs in both directions. The figure shows the random initialization on the left ($J = 3.03$) and the embeddings of Section 1.2 after 300 epochs with learning rate 0.5 on the right ($J = 0.465$), first as vectors and then as similarity matrices:



The loss cannot reach zero. `apple` matches two images, so the best its row softmax can do is 0.5 each; the same holds for `bat`, and for the apple and computer images down their columns. The smallest possible loss is 0.462, and the trained model is close to it: the splits above are 0.498/0.502 for `apple` and 0.496/0.499 for the apple image, while `computer` gives its image 0.984 and `fruit` gives the apple image 0.998.

The matching pairs are called **positives**, and the wrong pairings, the remaining entries of S, are **negatives**: each softmax pushes them down. Nobody had to collect wrong pairings; the other pairs

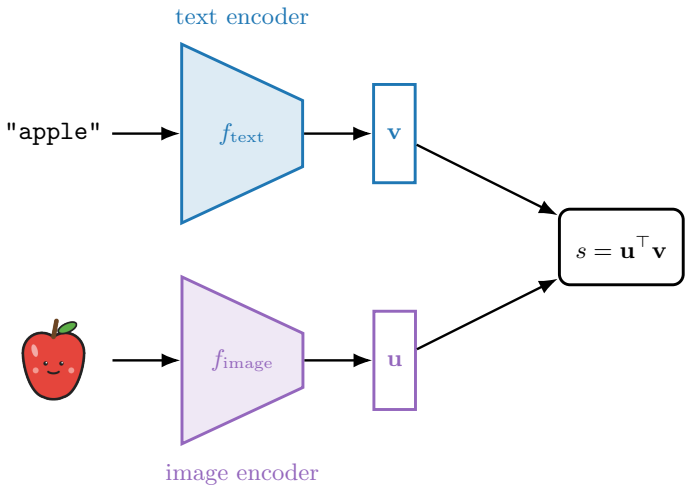
supply them. A loss that pulls positives together and pushes negatives apart is a **contrastive loss**, the C in CLIP.

1.4 Neural Network Encoders

A linear embedding has one row of V per word in its vocabulary, and a word outside the vocabulary has no embedding. The same holds for images and U, so the model of Section 1.3 can only embed the four words and four images it was trained on. A new photo of an apple has no row.

CLIP replaces the two linear embeddings with two neural networks that read their input: a **text encoder** f_{text} that reads the characters (or tokens) of the text, and an **image encoder** f_{image} that reads the pixels. Each outputs one K-dimensional embedding:

$$\mathbf{v} = f_{\text{text}}(\text{text}), \qquad \mathbf{u} = f_{\text{image}}(\text{image}).$$



In CLIP the text encoder is a Transformer and the image encoder is a convolutional network or a vision transformer; for these notes they are trapezoids. Nothing else changes. For N (text, image) pairs, the N text embeddings are the rows of V, the N image embeddings are the rows of U, $S = VU^\top$, and the loss is the bi-directional loss of Section 1.3. Backpropagation carries its gradient through the dot products into the encoder weights. The gain is generalization: the encoders embed text and images they have never seen, and a new photo of an apple should land near the embedding of the text `apple`.

1.5 CLIP Training Data

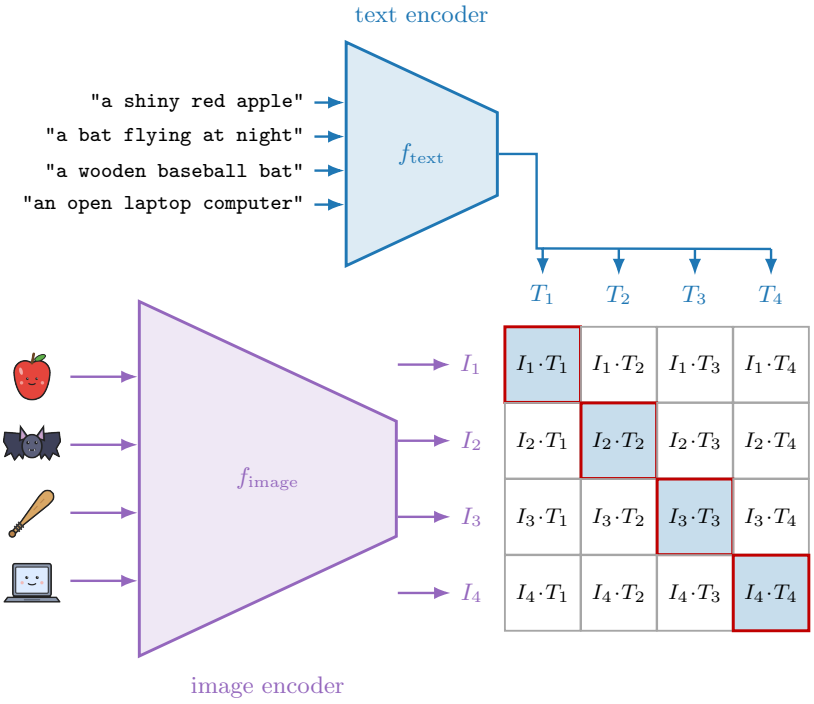
CLIP was trained on 400 million (image, text) pairs collected from the internet: each image with its caption, alt-text, or nearby text. The authors call the dataset WebImageText (WIT). They built it by searching for about 500,000 **queries** (every word with at least 100 occurrences in English Wikipedia, plus common bigrams, article titles, and WordNet synsets) and keeping up to 20,000 pairs per query.

Two properties of this data matter:

- The text is a sentence, not a word: `"a bat flying at night"` or `"a wooden baseball bat"` rather than `"bat"`. A sentence usually says which bat or which apple it means, so the two meanings that shared a word in Section 1.3 become two different texts.

- There are no class labels and no fixed set of classes. The text that happened to accompany each image is the only supervision; the authors call this **natural language supervision**.

Training uses minibatches of N pairs. The N texts go through the text encoder, the N images through the image encoder, and every pair in the batch counts as distinct, so the $N \times N$ similarity matrix has the N matching pairs on its diagonal and $N^2 - N$ wrong pairings everywhere else. CLIP uses $N = 32{,}768$, so each batch has about a billion negatives. The figure shows $N = 4$.



1.6 From Our Example to the CLIP Paper

The paper's pseudocode for one training step (its Figure 3):

```
# 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
```

The differences from our example:

- **Notation.** I is a batch of images and T a batch of texts. `I_f` and `T_f` are the encoder outputs; the embeddings `I_e` and `T_e` are our U and V, one row per image or text. `logits` is `I_e @ T_e.T`: images on the rows and texts on the columns, the transpose of our $S = VU^\top$.
- **A projection after each encoder.** The encoders output vectors of different sizes, d_i and d_t. W_i and W_t are linear layers that map both into one d_e-dimensional space. Our U and V already had the same width K. (A small non-linear network in place of each W made no difference.)
- **Normalization and temperature.** Each embedding is divided by its length (`l2_normalize`), so the dot product is the cosine of the angle between the two vectors, in $[-1, 1]$; ours was unnormalized. Since cosines are small, the logits are multiplied by a learned scale e^t before the softmax. The paper calls $1/e^t$ the **temperature** τ, initialized to 0.07 (a scale of about 14) and capped at a scale of 100 for stability.
- **Symmetric loss.** `loss_i` and `loss_t` are the two cross-entropies, one with the softmax over the texts for each image (image $\rightarrow$ text) and one with the softmax over the images for each text (text $\rightarrow$ image), with the correct answer for pair n at index n (`labels = np.arange(n)`; the pseudocode's `n` is our N). Their average is the bi-directional loss of Section 1.3.
- **Encoders, trained from scratch.** The image encoder is a modified ResNet-50 (five sizes) or a vision transformer (ViT-B/32, ViT-B/16, ViT-L/14; the last, at 336-pixel resolution, is the best CLIP model). The text encoder is a 12-layer, 512-wide, 8-head Transformer (63M parameters) over lower-cased byte-pair-encoded tokens, at most 76 of them, whose activation at the final `[EOS]` token is `T_f`. Neither encoder starts from pre-trained weights: no ImageNet weights, no pre-trained language model. The contrastive loss is the only training signal.
- **Scale.** 400 million pairs, batches of 32,768, 32 epochs. The largest vision transformer took 12 days on 256 V100 GPUs; the largest ResNet took 18 days on 592.

Zero-shot classification. The image $\rightarrow$ word direction is an image classifier whose classes are words, and with the trained encoders the classes can be any list. Write each class name into a sentence such as `"A photo of a bat."`, embed the sentences with the text encoder, embed the image, and pick the class with the highest cosine similarity. The classes are given in English at test time, so one model classifies dog breeds, satellite images, and our four cartoons without retraining. The paper calls this **zero-shot** classification. The template matters because the training texts were sentences: `"A photo of a {label}."` beats the bare label.

References

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, I. Sutskever. Learning transferable visual models from natural language supervision. ICML 2021. https://arxiv.org/abs/2103.00020

OpenAI. CLIP: Connecting text and images. Blog post, 2021. https://openai.com/research/clip