SemEval2023 Visual-WSD

Joint project CV/NLP

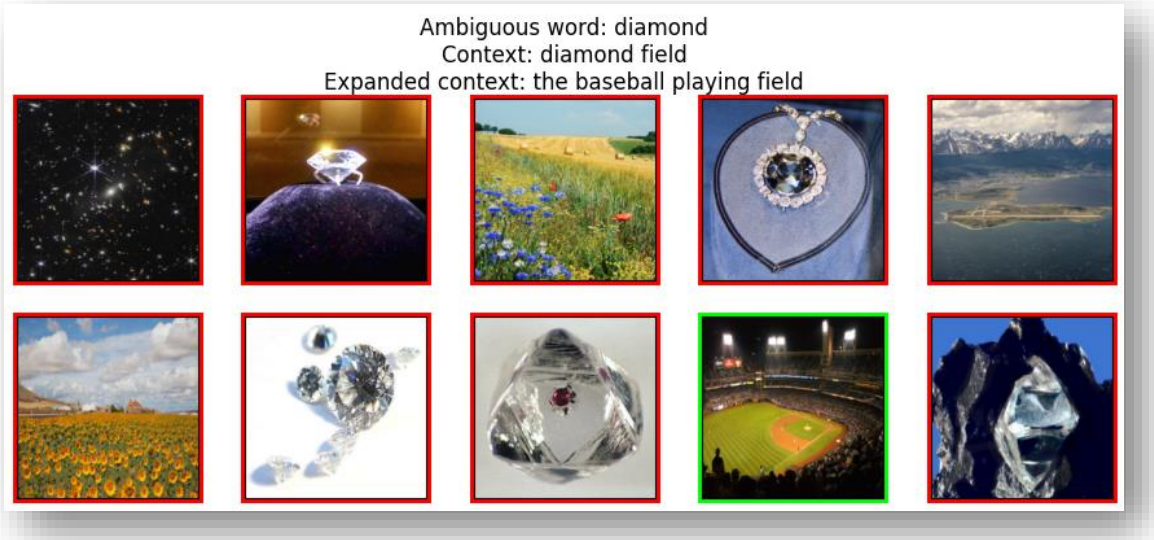
Francesco Bottalico

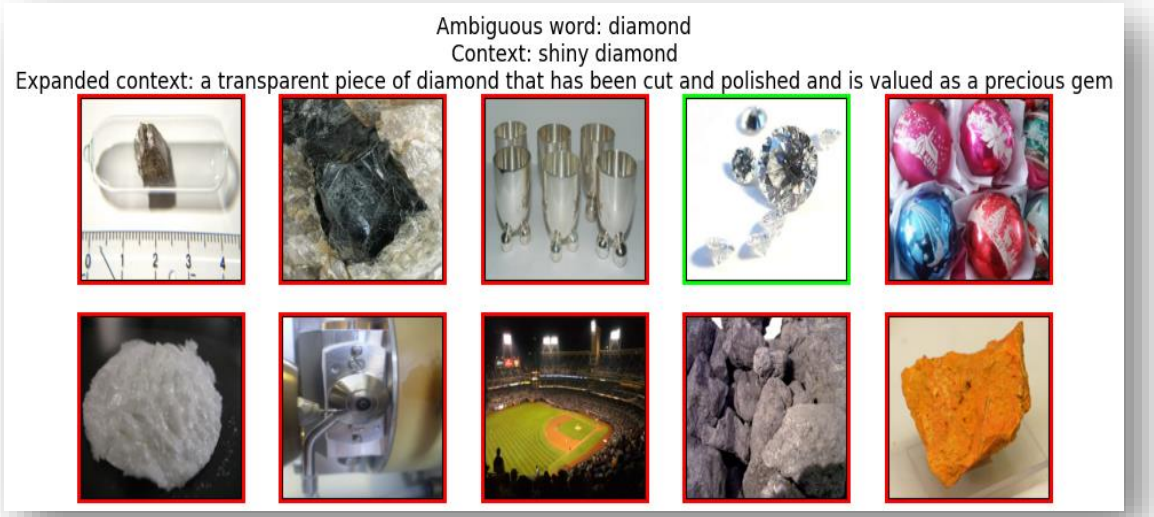
The task

The task of Visual Word Sense Disambiguation is defined as follows:

GIVEN
- A target **ambiguous** word
- A limited context
- A set of 10 candidate images

→ **Predict the correct image connected to the intended sense of the word**





Same word, different meaning

Data and preprocessing

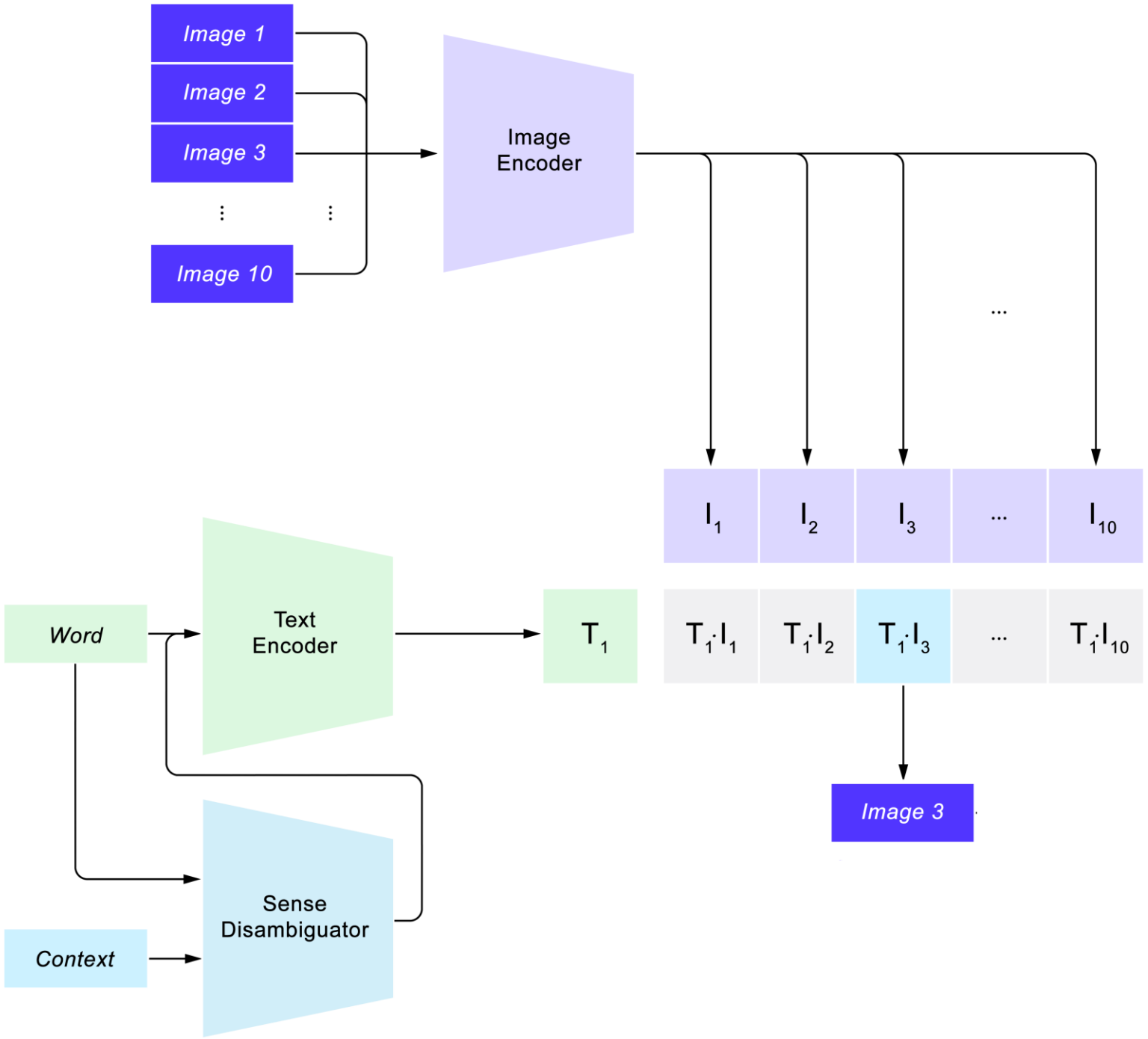
	Training set	Test set
Samples	12896	463
Unique target words	12527	259
Images	13158	8100
Max context lenght	2	4

Text

- Mainly done in disambiguation phase
- Remove stopwords from context
- Add <start_of_text> and <end_of_text> special tokens
- Padding/truncation to have exactly 77 tokens

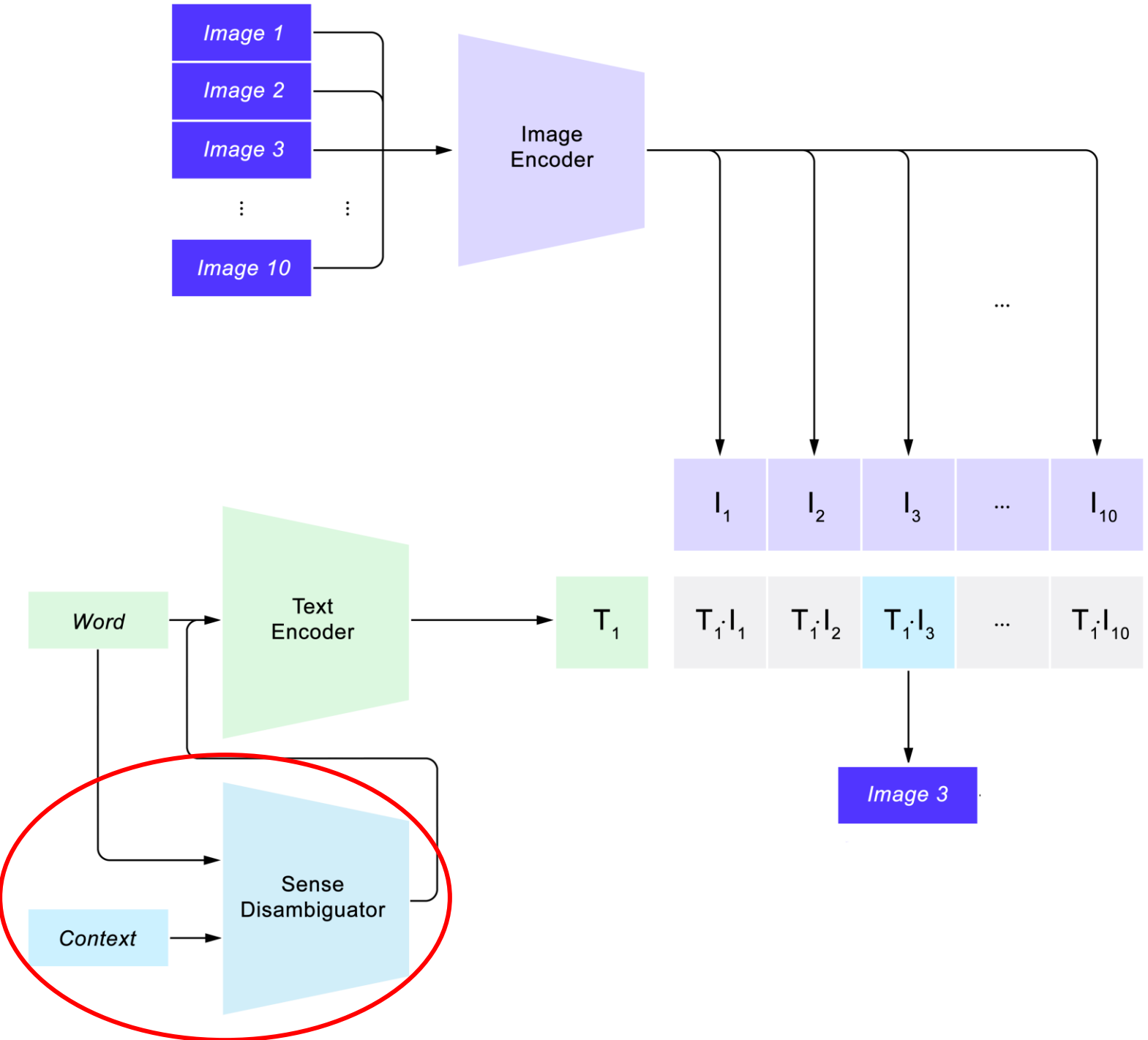
Images

- Resize to 224x224 (mantaing the aspect ratio)
- If not possible → center crop



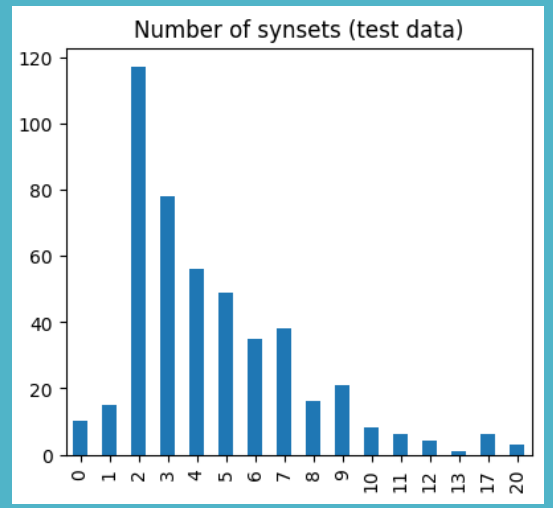
Pipeline

- The target word and the context are used as the **disamibuguator**'s input
- The disambiguator gives us a richer context
- The expanded context and the target word are merged and given to the **text encoder**
- The ten candidate images are given to the **image encoder**
- Cosine similarity + softmax is used to compute the score between text and images



Disambiguator

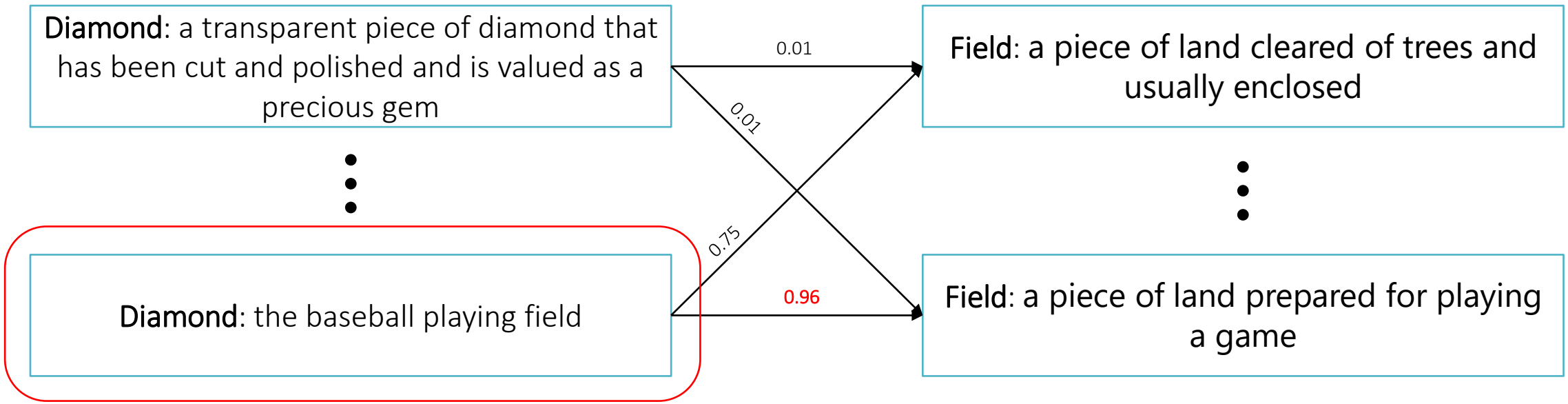
- It uses WordNet to expand the limited context of a word
- Three algorithms have been implemented:
  - **MPNet**
  - **WordNet** tree distance
  - **Lesk** algorithm

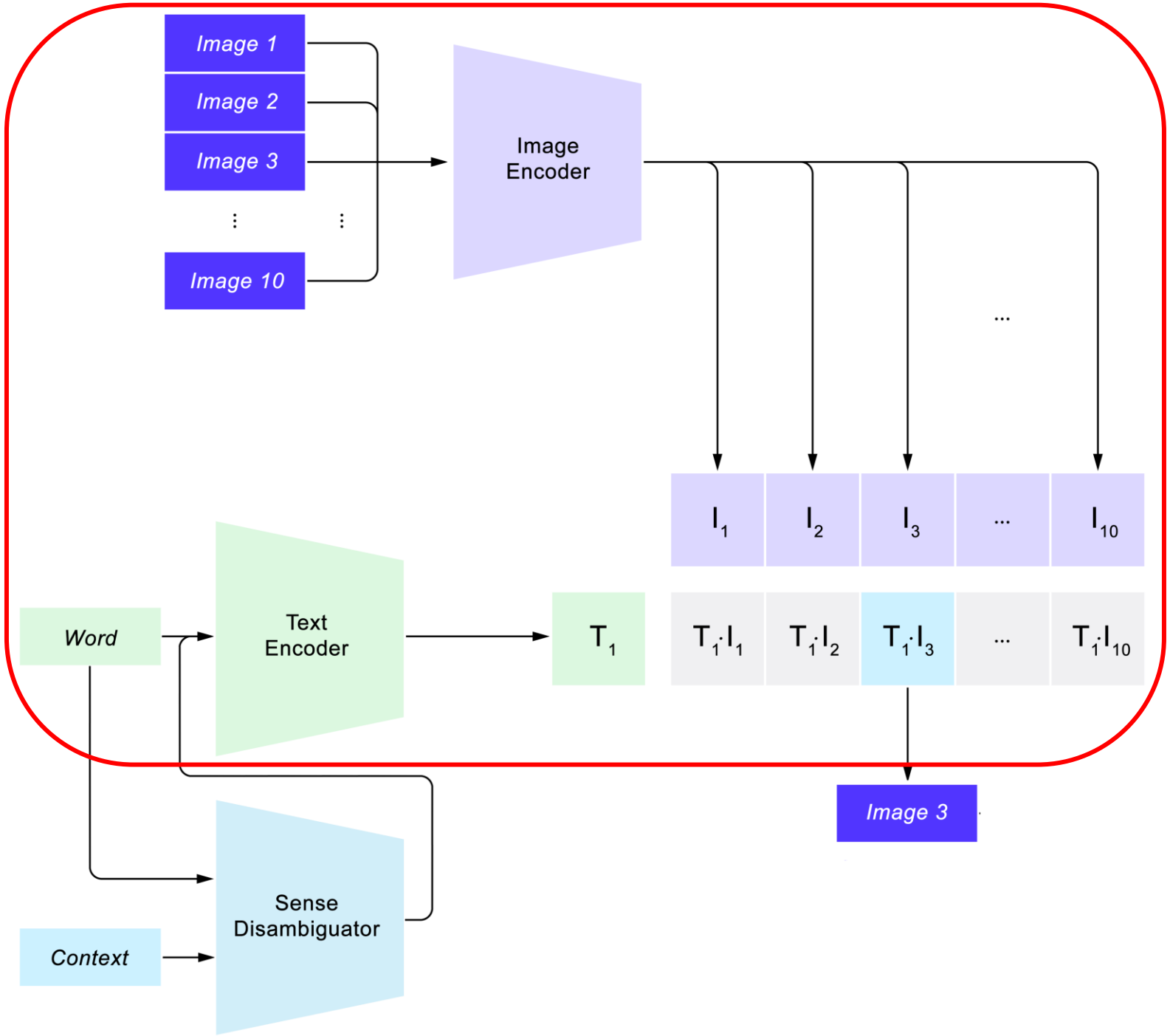


Disambiguator

- All the synsets relative to the target and the context word are retrieved
- We compute the **similarities** between each pair
- We take the synset with the highest score and use its WordNet definition to create the expanded context
- We can take into account PoS tags of the words during WordNet search (OPTIONAL)

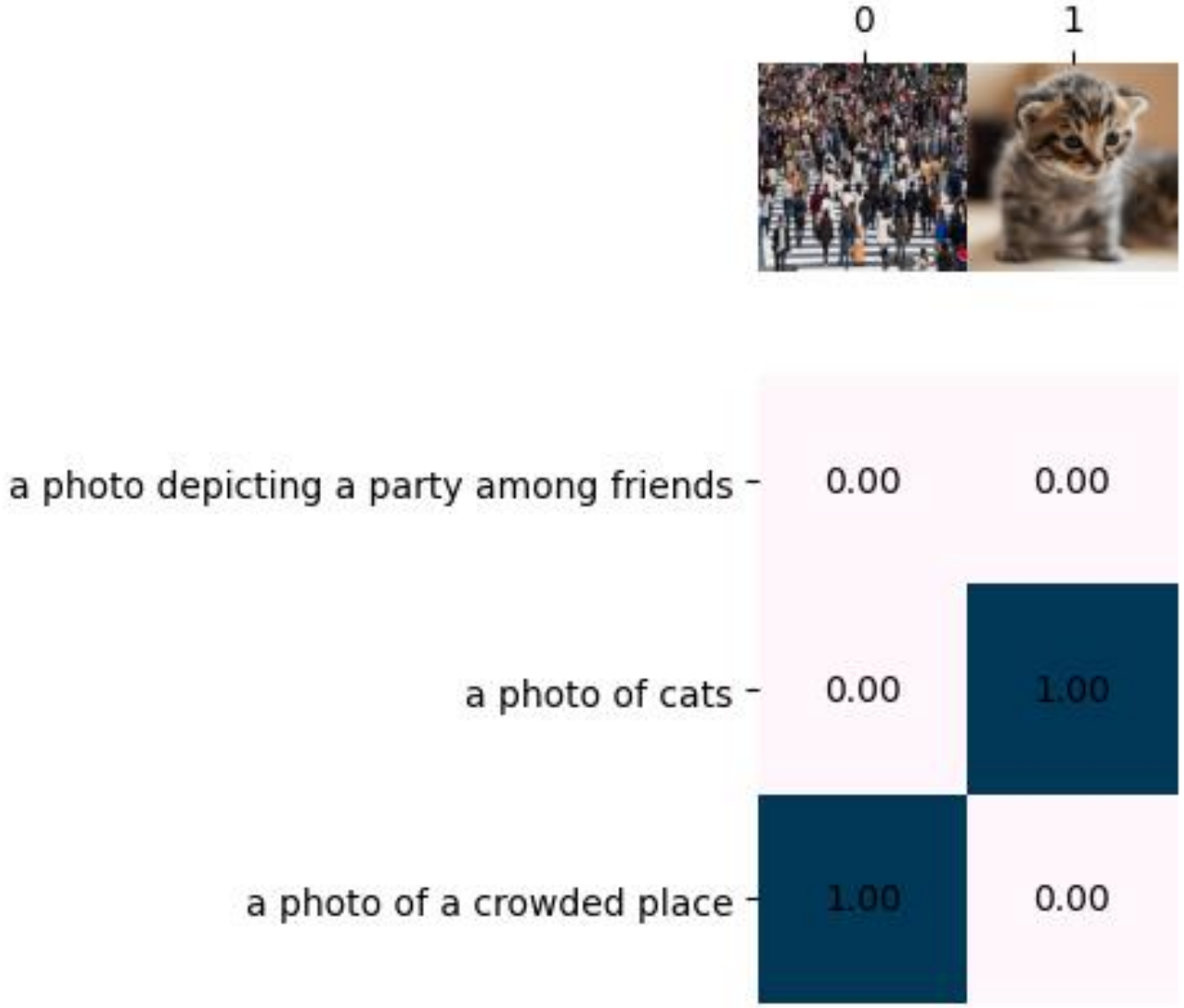
MPNet → a transformer based encoder
WordNet → exploit the tree structure of WN to interpret distances between nodes as similiarity
Lesk → Synset definition converted to a BoW representation and inner product is used to compute similarities





CLIP

- OpenAI's model trained on connecting image-text pairs
- <u>Texts and images embeddings live in the same latent space</u> (1024 dimensions)
- Positive pairs are close in the vector space
- **Text encoder**: 12-layer stack Transformer encoders
- **Image encoder**: ResNet-50 (modified), Vision Transformer Base, Vision Transformer Large

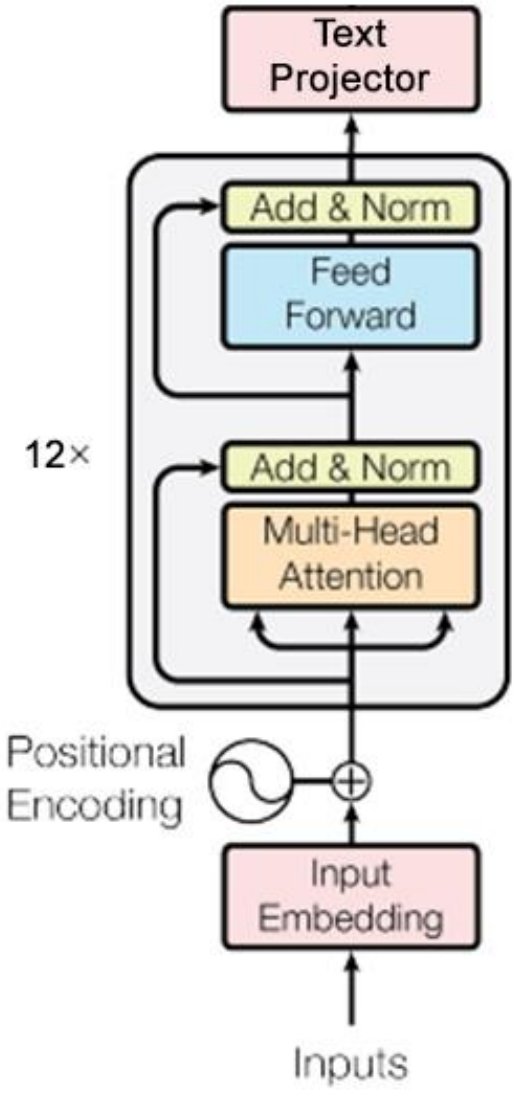


CLIP example. Connecting the correct text to each image

CLIP - Example

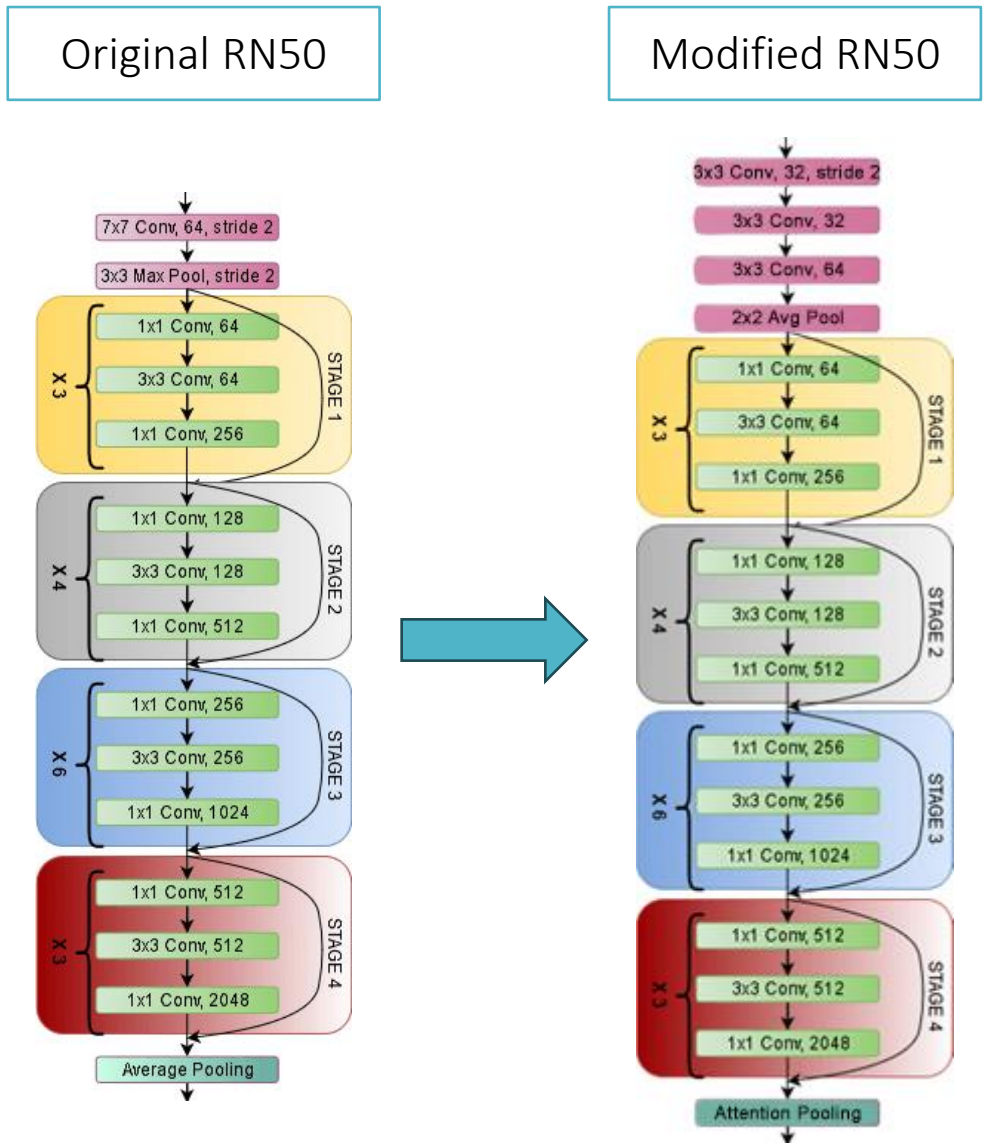
- CLIP works both ways:
  - Connect text to images (softmax is computed on the columns)
  - Connect image to texts (softmax is computed on the rows)
- Interactive demo on the COLAB notebook
- For our task we are going to pick the correct image given a text

CLIP – text encoder



- 12-layer Transformer encoder
- Normalization on inputs instead of the outputs
- **GELU** instead of **ReLU**
- 512 as hidden dimension
- **Text projector** to project textual embedding in the same image's vector space

CLIP – image encoder



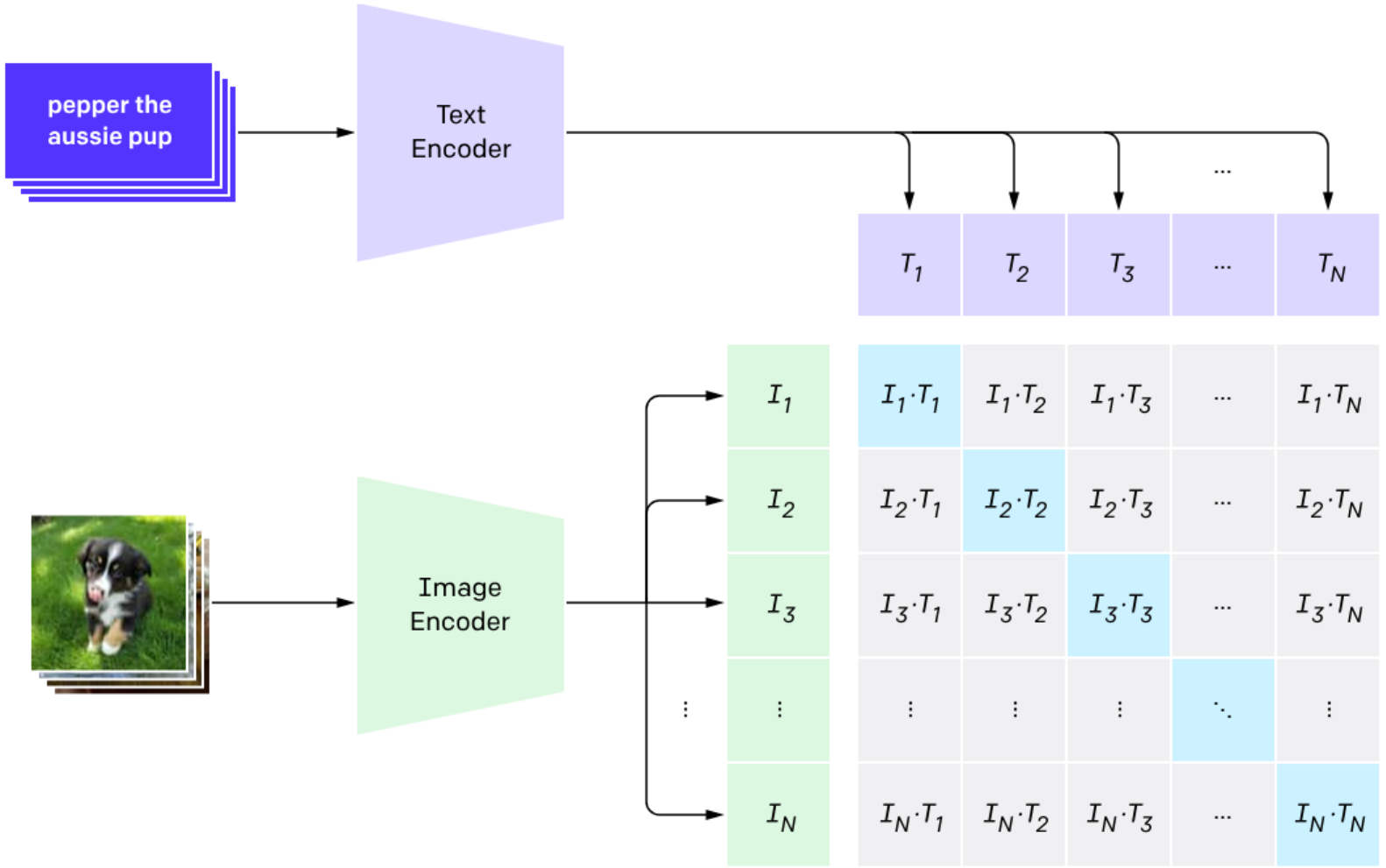
- ResNet-50
  - **3 stem convolutions** instead of 1
  - **2x2 average pooling** instead of 3x3 max pooling
  - Final **attention pooling** instead of average pooling
- ViT-B-16
  - 16x16 patch size
  - 768 as hidden size
  - 12 attention heads
- ViT-L-14
  - 14x14 patch size
  - 1024 as hidden size
  - 16 attention heads

CLIP variants

The only variant we try to fine tune

Model	Parameters	Pre-training dataset
Text encoder + ResNet-50	102M	OpenAI 400M
Text encoder + ViT-B-16	150M	LAION-2B
Text encoder + ViT-L-14	427M	DataComb-1B

CLIP – training



- Pre-trained on huge collections of text-image pairs

ORIGINAL TRAINING METHOD

- Batch of size N
- N^2 total pairs in the batch
- Maximize the probability of the N positive pairs in contrast to the $N(N-1)$ negative ones
- 1 positive, $N-1$ negative pairs for each batch input
- Negative pairs are randomly obtained

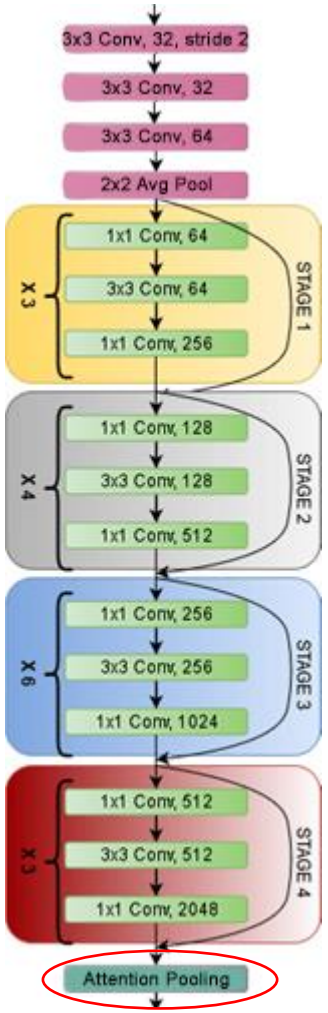
OUR TRAINING METHOD

- Batch of size N
- $N * 10$ total pairs in the batch
- Maximize the probability of the N positive pairs in contrast to the $N * 9$ negative ones
- 1 positive, 9 negative pairs for each batch input
- Negative pairs are specified in the dataset

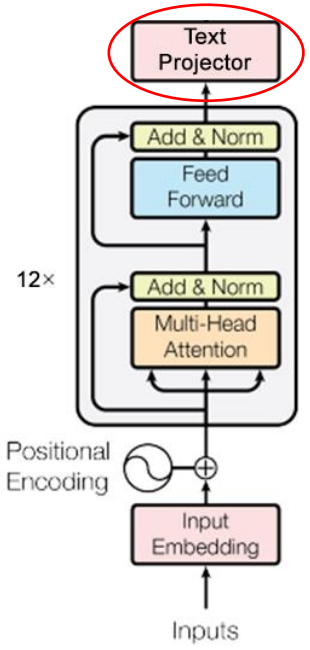
CLIP – fine tuning

- RN50 CLIP fine tuned for 2 epochs
- Text and image feature extractor are freezed (entities in the latent space are only moved around)
- **AdamW** optimizer with small learning rate ($2 * 10^{-5}$)
- Batch size of 8 (limited GPU memory)
- **Gradient accumulation** to cope with the limited GPU memory, we can simulate a batch of size 64/128 by accumulating the gradients for 8/16 iterations without updating the weights
- Caching is implemented to speed up epochs after the first one (it avoids preprocessing the same image more than once)

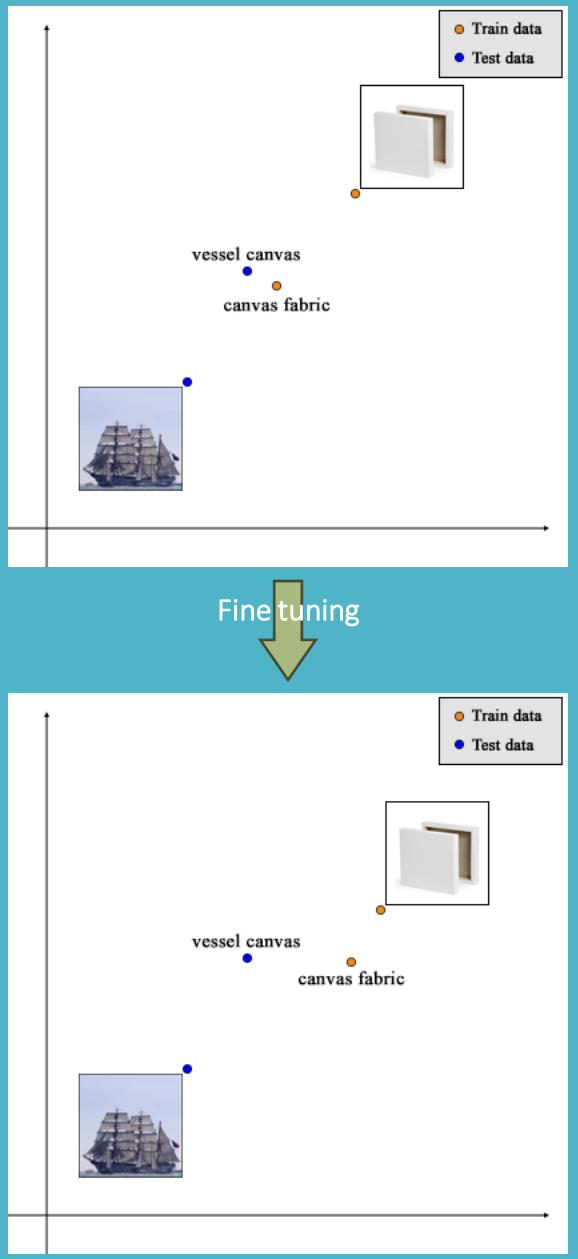
The model



RN50



Text encoder



Experimental results

	Model	MRR	Hits@1	Hits@3
Baselines	Random	0.317	0.125	-
	Organizers	0.739	0.605	-
	Winners	0.895	0.840	-
0-shot	RN50 (MPNet)	0.772	0.654	0.857
	RN50 (Lesk)	0.757	0.629	0.857
	RN50 (WordNet)	0.723	0.579	0.825
	ViT-B-16 (MPNet)	0.810	0.708	0.894
	ViT-B-16 (Lesk)	0.788	0.674	0.881
	ViT-L-14 (MPNet)	**0.833**	**0.737**	**0.933**
	ViT-L-14 (Lesk)	0.817	0.706	0.922
Fine tuned	RN50 (MPNet)	**0.833**	**0.737**	0.920

- **Wordnet** based disambiguation doesn't work well, **MPNet** is the best
- **Lesk** is less accurate but it's faster
- Performances increases almost linearly with model complexity
- Fine tuning the simplest model significantly boost performances

Thanks for your attention