# SemEval-2023 Visual-WSD
# (Colab notebook link)

**Francesco Bottalico [787587]**
f.bottalico15@studenti.uniba.it

## Abstract

The aim of this project is to explore and solve the task of Visual Word Sense Disambiguation as defined in the SemEval-2023 (Raganato et al., 2023) challenge. This is done by using different kinds of word sense disambiguation algorithms in order to expand the target word context, then connecting images and texts vector representations by using different variants of the CLIP (Radford et al., 2021) model. By using this approach we are able to obtain results over the baselines even without fine tuning the model. After fine tuning the smaller model we obtain similar performances to the biggest not fine tuned model, meaning that fine tuning is useful and we probably still have room for improvement on bigger models.

## 1 Introduction

The problem of Visual Word Sense Disambiguation as defined in the SemEval-2023 challenge constists in connecting an highly ambiguous target word with a very limited context (another 1 or 2 words) to the correct image. The task of connecting text and images is a very actual problem for which the SOTA approaches are all based on the OpenAI CLIP (Radford et al., 2021) models. This model uses a text and a image encoder to extract textual and visual features and project them in the same vector space, then these representations are brought closer together by using contrastive learning. In this way CLIP is able to connect texts and images in a bidiredictional manner: during the inference phase we can obtain the most coherent images given a textual query, but also the most coherent texts given a query image. Using only CLIP in this scenario does not work quite well as we have very short and ambiguous phrases. To solve this problem three different algorithms for word sense disambiguation are implemented: MPNet (Song et al., 2020), WordNet tree similarity, Lesk algorithm. These are used to find the correct WordNet synset for the word and use the additional informations given by it to expand the context in the phrase. Different variants of the CLIP model are tested by varying the visual encoders, in particular we will focus on ResNet-50 (He et al., 2015) and Vision Transformers (Dosovitskiy et al., 2021). We will explore various disambiguation algorithm together with several CLIP variants, without training them on the specific dataset (*zero-shot inference*), this can be done by leveraging the fact that CLIP is pre-trained on huge datasets. The SOTA approach for the SemEval-2023 task trains CLIP on a dataset built specifically for ambiguous word - image connection, which contains training triples of the type (word, sense, image), for every sense of a word. This dataset is over $500GB$, therefore the training is very computationally expensive. To overcome this problem, fine tuning will be done only using the original training dataset (on ResNet-50 CLIP) in order to increase even further performances.

## 2 Related Work

As already said, the SOTA on connecting text and images is achieved by CLIP (Radford et al., 2021), in this work we will use OpenCLIP (Cherti et al., 2022), an open source implementation of CLIP pre-trained on open datasets. The original model is trained on a 400M samples proprietary dataset, while OpenCLIP is trained on larger and freely available datasets such as LAION-2B (a subset of LAION-5B (Schuhmann et al., 2022)) and DataComp-1B (Gadre et al., 2023). We will focus on implementations of CLIP using a ResNet-50 (He et al., 2015) visual encoder, due to its smaller size, and Vision Transformers (Dosovitskiy et al., 2021), due to its better performances. For the word disambiguation, one approach involves the Microsoft's MPNet (Song et al., 2020) model, a transformer-based encoder (Vaswani et al., 2017) trained specifically for language semantic understanding. Transformer-based encoders are also

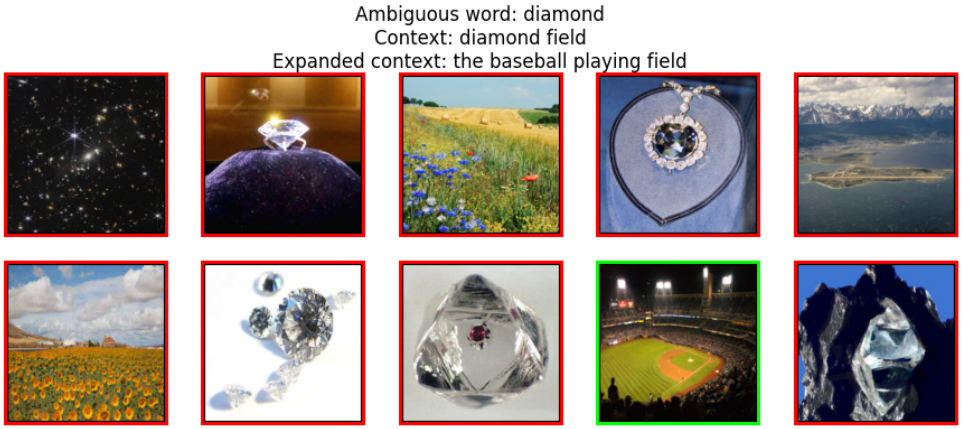


Figure 1: An example of the dataset entry. **Ambiguous word** is the target word, **Context** is the given context, **Expanded context** is the additional context obtained using the implemented disambiguation algorithms (it's not present in the dataset). The image in green is the target image.

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

Table 1: Visual-WSD dataset statistics.

used as textual features extractor for the CLIP model, the main differences with the original transformer architecture are: normalization of the input instead of the output, $GELU$ (Hendrycks and Gimpel, 2023) is used as the activation function instead of the $ReLU$.

## 3 Task and dataset

The SemEval-2023 task of Visual Word Sense Disambiguation (Visual-WSD) constists in the following assigment: given an ambiguous target word with a limited context you have to select the correct image among ten candidates (Figure 1). The training set has 12896 samples with 12527 different target words and 13158 images, the test set has a total of 463 samples with 259 unique target words and 8100 images.

### 3.1 data preprocessing - NLP

Not much preprocessing has to be done on the textual data as we have only one target word. For the context we only have to do stopword removal as in some cases there are some. Some preprocessing is needed for the disambiguation part but will be described in details in section 4.1. After obtaining the expanded context using the disambiguation algorithms, the input word and the context are put together and are given as input to the text encoder. For this tokenization is needed, in particular the tokenizer spilts the tokens using regular expressions, a <*start_of_text*> and a <*end_of_text*> special token are added respectively at the start and at the end of the phrase, lastly padding or truncation are used in order to have exactly 77 tokens.

### 3.2 data preprocessing - CV

Visual data needs to be preprocessed both in the fine tuning and in the testing phase, in particular during testing images have to be resized in order to have a size compatible to that of the model (224px×224px). This is done by using the *transforms* module of the *torchvision* library:

- Images are resized to a dimension of 224px×224px keeping their aspect ratio
- If the image is not of the wanted size, a center crop is used to obtain the correct size.

On the other hand, during training images are randomly cropped in order to avoid any kind of overfitting.

## 4 Method

The pipeline of the proposed method (Figure 3) can be described as follows:

1. The target word and its context are given as input to the disambiguator that gives us the intended sense of the word.
2. The target word and the sense are merged and given as input to the CLIP text encoder.

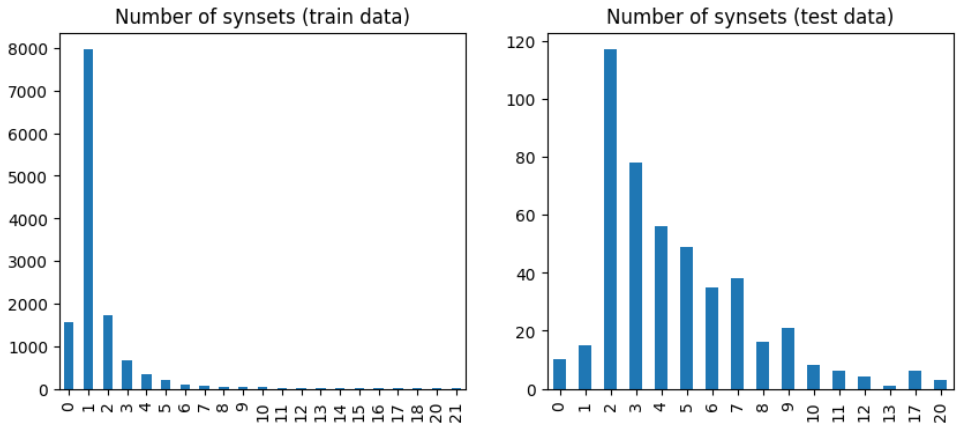


Figure 2: Number of synsets for each word in training and test set.

3. Concurrently, the ten candidate images are passed to the CLIP image encoder.

4. Lastly, CLIP gives us the scores for each (text, image) pair. We take the one with the highest score as the prediction.

Now we discuss every step in more detail.

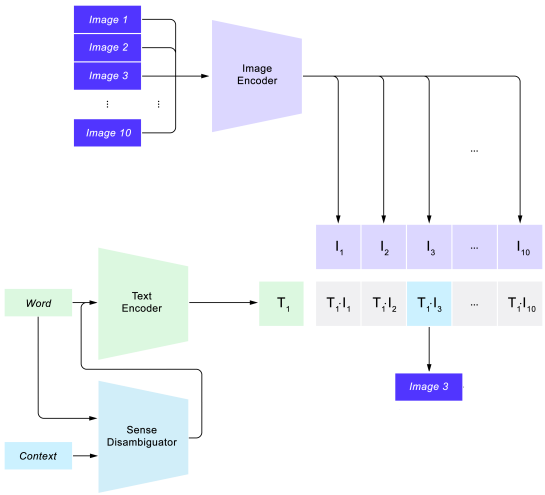


Figure 3: Pipeline of the proposed method.

### 4.1 Disambiguator - NLP

The role of the *Disambiguator* class is to extract from WordNet the synset of the target word associated to the correct meaning, given the context. This is done following these steps:

1. Retrieve the **target word** synsets, if not found the word cannot be disambiguated.

2. Take the context, remove the target word from it together with every stopword to obtain the **context word**, then retrieve all of its synsets.

3. Compute the similarity between every pair of **target word - context word** synset.

4. Take the synset of the target word associated with the highest score, then return its WordNet definition. We will refer to it as the **expanded context** (Figure 1).

The similarity measure depends on the selected algorithm. Since we rely on WordNet synsets to disambiguate and expand the context of a word, it is useful to explore the presence or absence of every word of the dataset in the WordNet ontology. In Figure 2 we can see that a low number of words are not found on WordNet and many are polysemic. To overcome the first problem we could think about using a bigger ontology like *BabelNet*.

**MPNet** The first algorithm uses a subsymbolic approach to compute similarities between synsets, in particular MPNet (Song et al., 2020) is used to encode the definition of every synset of the target and the context word. Then dot product is employed to compute similarities.

**WordNet** The second approach exploits the tree structure of WordNet to compute the distance between the nodes in the tree associated to the two synsets. This is done via the *nltk* library methods.

**Lesk** The last option is the Lesk algorithm. We compute a bag-of-word representation of every synset's definition by:

1. Tokenizing the definition.

2. Removing all the stopwords.

3. Adding to the BoW all the lemma's synonyms.

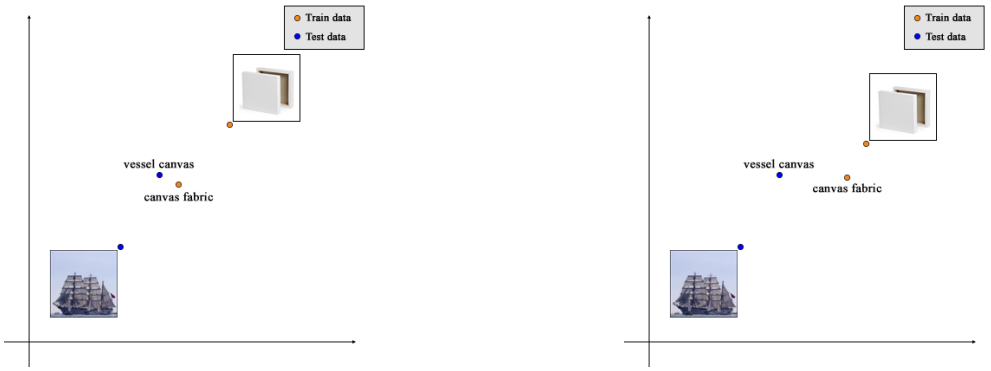


Figure 4: **Left**. vector space before fine tuning. **Right**. vector space after fine tuning.

4. Applying Porter stemming.

Similarity is then obtained as the dot product between two BoW (which gives us the number of words in common).

**PoS tagging** Optionally we can use PoS tagging to assing a label (NOUN, VERB, ADJ) to the target and the context word, this can be used to search with the highest priority the synset of that specific type. By default the synset for a word are searched in the following order: NOUN $\rightarrow$ VERB $\rightarrow$ ADJ. If PoS tagging is used, we run the *nltk* PoS tagger on the context and extract the tag for the target and the context word, we then use it to search first synsets of that type. E.g A word is identified as a *VERB*, synsets are searched in the order: VERB $\rightarrow$ NOUN $\rightarrow$ ADJ.

PoS tagging is disabled by default as it does not increase performances, probably due to the context being too short.

### 4.2 CLIP - CV

After loading a pre-trianed CLIP model, the zero-shot evaluation is done by loading the ten candidate images for every text input in the batch, then encoding them through the visual encoder and the text through the text encoder. We end up with $N$ 1024-dimensional vectors for each textual input and $10 \cdot N$ 1024-dimensional vectors for each image. We then compute the **cosine similarity** between the textual embeddings and the image embedding of the ten candidate images, ending up with a similarity matrix of size $N \times 10$. Finally we apply the softmax function on the rows with a temperature parameter $\tau = 0.01$ and take the highest scores as the predictions.

**Fine tuning** The training during the fine tuning phase for the CLIP model should be implemented differently from the training done during the pre-training phase, as our task is more specific than the task on which CLIP is originally trained. Traditionally, given a batch of size $N$ we take $\{T_1, T_2, ..., T_N\}$ textual inputs and $\{I_1, I_2, ..., I_N\}$ images from the dataset, such that $(T_i, I_i)$ are positive pairs and every other is a negative one. We compute the similarities between every text and every image to obtain a $N \times N$ matrix. Then we have to maximize the probability of the positive pairs $(T_i, I_i)$ over all the negative ones $(T_i, I_j)$ with $j \neq i$, and the probability of the positive pairs $(I_i, T_i)$ over all the negative ones $(I_i, T_j)$ with $j \neq i$ (Figure 5). This is done by computing the softmax function on both the rows and the columns. Note that in this scenario we have one positive and $N - 1$ negative pairs for every input in the batch, for a total of $N^2$ pairs.

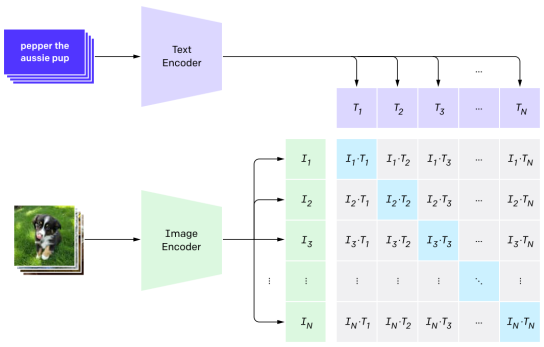


Figure 5: The original CLIP's training method.

In our case we have a batch of $\{T_1, T_2, ..., T_N\}$ target words and ten images associated to each of them, so the negative pairs are specified in the training set and are not generated when randomly building the batch: for every target word we have exactly one positive and nine negative pairs for every input, no matter what the batch size is. This means that we have a total of $10 \cdot N$ pairs, way

| | 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 |

Table 2: Experimental results.

less than the $N^2$ as in the previous case, and we can leverage this fact during fine tuning in order to use less computational power and set larger batch sizes.

For fine tuning the model we first freeze the features extraction layers, namely the text Transformer and the ResNet-50. What will be trained are the **text projector**, A linear layer after the text encoder that projects the textual features from 512 to 1024 dimensions, and the last layer of the ResNet-50, which is not an *average pooling* layer as in the original architecture but a **self attention layer**. The rationale behind this is that the text and the image encoders are already well trained on extracting features on huge datasets, we only want to move the images and texts representations in the 1024-dimensional space in the hope to reduce ambiguities. A visual representation of this idea can be seen in figure 4.

By apporting the just explained changes to the training phase of CLIP, we can fine tune the model pretty easily without much computational efforts.

## 5 Experimental settings

For the experiments we try three different configurations of CLIP, all of them use a stack of 12 transformer encoders with an hidden dimension of 512 as a text encoder but different image encoders:

1. The first variant uses a **ResNet-50**. This model is a modified version of the original one, it has 3 "stem" convolution instead of 1 followed by an average pooling instead of a max pooling and a self attention layer instead of an average pooling as the last layer,

2. The second one uses the base version of the Vision Transformer (**ViT-B**) with an hidden size of 768, 12 attention heads and a $16 \times 16$ patch size.

3. The last one uses the large version of the Vision Transformer (**ViT-L**) with an hidden size of 1024, 16 attention heads and a $14 \times 14$ patch size.

These models are ordered by their complexity (Table 3). Every model will be evaulated using all of the three disambiguation algorithms.

| Model | Parameters |
|---|---|
| Text encoder + ResNet-50 | 102M |
| Text encoder + ViT-B-16 | 150M |
| Text encoder + ViT-L-14 | 427M |

Table 3: Number of parameters for each model.

For zero-shot predictions the first model is pretrained on a OpenAI's proprietary dataset of 400M samples, the second is trained on LAION-2B (Schuhmann et al., 2022) and the last one on DataComp-1B (Gadre et al., 2023).

For fine tuning we use AdamW (Loshchilov and Hutter, 2019) as the optimizer with a small learning rate $\gamma = 2 \cdot 10^{-5}$ and a weight decay $\lambda = 0.01$. We fine tune the model for only 2 epochs and we use *gradient accumulation* to simulate larger batch sizes. This works by computing the gradient over $k$ different batches without updating the weights, then updating them after. By doing this we can simulate larger batch sizes while having low GPU memory that phisically limit us to a maximum batch size of 8. For example with $k = 16$ we update the weights every 16 batch iterations, simulating a $8 \cdot 16 = 128$ batch size. Lastly, as large

computational effort is made during the image preprocessing phase (images are loaded in memory and transformed as explained in section 3.2), a caching system is implemented for the transformed images so that every loaded image is transformed only once, in order to reduce the training times especially after the first epoch.

As for the metrics used to evaluate the models, we will use **Mean Reciprocal Rank**, **Hits@1** and **Hits@3**.

## 6 Results

Results are shown in table 2. We can see that even the simplest model is able to surpass baselines thanks to the disambiguation algorithms, out of which **Lesk** and **MPNet** are the better ones. We drop *WordNet* algorithm for the other experiments as it behaves worse than baselines. This is probably due to the fact that using a WordNet tree based distance as a similarity measure does not work well in disambiguation and, consequently, too much noise is added in the text encoder input. MPNet works better than Lesk but is much slower and should be ran with GPU acceleration, so if we need a faster alternative, we can trade-off efficiency for performances by using Lesk-based disambiguation.

As for the model complexity, performances increases as we use more powerful models with **ViT-L-14** being the best performing. Lastly we notice that, through fine tuning, the least powerful model behaves the same as the most powerful one, meaning that fine tuning works quite well and we could have margin for improvement even on the bigger models if we fine tune them.

## 7 Conclusions

We have seen how to extract the correct sense of a target word given a limited context via different algorithms and how to adapt a pre-trained CLIP model to the task of Visual Word Sense Disambiguation, done by editing its original framework structure to make predictions. We then evaluated different CLIP variants of different sizes with these disambiguation algorithms and we saw how to fine tune the model in order to increase performances, while taking into account the limited GPU capacity that we had.

We saw how results became better as the model complexity goes up and that fine tuning is useful to gain some performances.

## References


Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. 2022. Reproducible scaling laws for contrastive language-image learning.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale.

Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, Eyal Orgad, Rahim Entezari, Giannis Daras, Sarah Pratt, Vivek Ramanujan, Yonatan Bitton, Kalyani Marathe, Stephen Mussmann, Richard Vencu, Mehdi Cherti, Ranjay Krishna, Pang Wei Koh, Olga Saukh, Alexander Ratner, Shuran Song, Hannaneh Hajishirzi, Ali Farhadi, Romain Beaumont, Sewoong Oh, Alex Dimakis, Jenia Jitsev, Yair Carmon, Vaishaal Shankar, and Ludwig Schmidt. 2023. Datacomp: In search of the next generation of multimodal datasets.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition.

Dan Hendrycks and Kevin Gimpel. 2023. Gaussian error linear units (gelus).

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision.

Alessandro Raganato, Iacer Calixto, Asahi Ushio, Jose Camacho-Collados, and Mohammad Taher Pilehvar. 2023. SemEval-2023 Task 1: Visual Word Sense Disambiguation. In *Proceedings of the 17th International Workshop on Semantic Evaluation (SemEval-2023)*, Toronto, Canada. Association for Computational Linguistics.

Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. 2022. Laion-5b: An open large-scale dataset for training next generation image-text models.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.