

Disambiguation of Word Senses by Identifying Visual Representations

Kyle Arechiga
School of Information
University of Arizona
Tucson, Arizona
kylearechiga@arizona.edu

Abstract

I present an approach for SemEval-2023 Task 1 which entails the classification of ambiguous word senses with limited context by identifying a representative image from a list of 10 images. The utilized methods rely on using pre-trained image and text embeddings and fusing their representations into a joint embedding space to determine which image accurately depicts the word sense of the input text. I elaborate in detail the process and challenges that were faced while working towards training an optimal model. The results on the test dataset indicate that the model performs much better when using a sentence transformer for the text embeddings, cosine similarity for combining the image and text embeddings, and shuffling the default image options provided by the original dataset. The model achieved 29.4% top 1 accuracy, 61% top 3 accuracy, and a mean reciprocal rank of 0.5029 on the test dataset. The model performance showed that even with a relatively low-cost model, a model can be trained to learn various word senses and apply them to classifying representative images. There is plenty of room for improvement for future iterations of visual word sense disambiguation. Code and results are available at <https://github.com/karechiga/visual-word-sense>.

I. INTRODUCTION

Among the most common challenges in Natural Language Processing (NLP) is the ability for models to distinguish semantic meaning between similar tokens. Words like “mouse” or “drop” can have multiple meanings and often depend on the context. Are we referring to a computer mouse to navigate a screen, or the small rodent? The contextual words, “computer” and “rodent,” help determine the correct interpretation of the word “mouse.” The English language is full of multi-use words, and it is imperative that NLP models have a way to decipher meaning given their varying contexts.

The goal of SemEval-2023 Task 1 is to train a model to select an image from a set of 10 image options given a limited text input. This task provides a way to characterize how well a model can disambiguate word senses by tasking the model to choose a visual that best represents the ambiguous word. Many of the image options may be similar themselves, making the task even more of a challenge for the model.

This paper will go over the process of designing, training, and evaluating our classification model. We will discuss the dataset and training validation splits. Then will go over the deep learning methods used to train the model. The model was then evaluated on an unseen test dataset; the results of which will be discussed in detail. Finally, we will look to reflect on the future of this task in the field and consider the ethical implications.

II. TRAINING AND DEVELOPMENT DATA

The SemEval task provides a given set of 12,869 samples to perform training and validation on. Each sample consists of

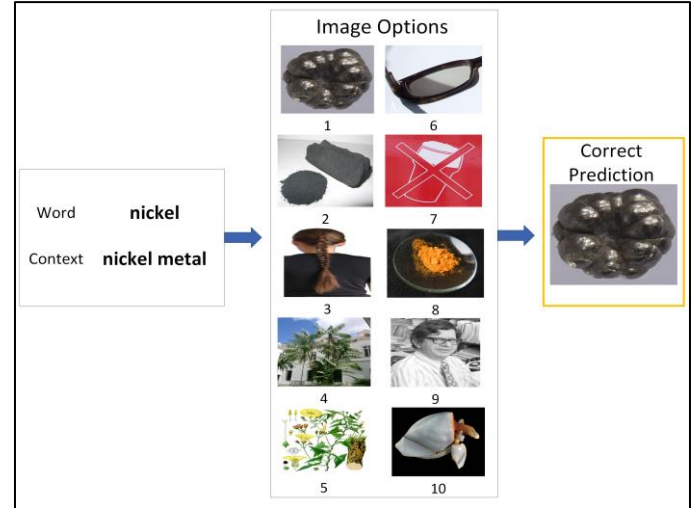


Fig. 1. One sample from the SemEval-2023 Task 1 dataset. Given “nickel” as the word and “nickel metal” as the context, the model is asked to predict which of the 10 image options best represents the text input.

a word, the limited context for the word, and a set of 10 images to choose from. *Figure 1* illustrates one of the samples. These samples were split into a training set (80%) and a validation set (20%) to test out the model during development, before eventually running the model on 463 test samples that were not observed during training.

In the training dataset, the context for each sample is structured as only two words: the given word that we want to predict, and one word that provides the context. Occasionally the input words contain hyphens that represent word phrases rather than just singular words. For example, we see words like “black-patent” with a context of “black-patent jacket” or “lion’s-ear” with a context of “lion’s-ear herb”. Some terms like “ballroomdance” appear to contain multiple words combined into one term, even though they may not appear that way in a normal grammatical sentence. Many of the other words are in the context of a very specific biological genus or family (i.e. “hyacinthaceae family” or “urocystis genus”) and are rarely used terms in general contexts. The text inputs contain a wide range of word senses, which gives our model plenty of data to generalize.

There are 12,999 total images in the given dataset. 12,869 of the images are the correct output for exactly one sample, and all images are scattered throughout the dataset as false options. Some of the samples have similar images which make it more of a challenge for the model to make the correct prediction. However, most of the given image options are completely unrelated to the target image. For the sample in *Figure 1*, the model would try to find the target image for “nickel metal”. Among the given options for this sample, there could only be one or two other images that may be considered

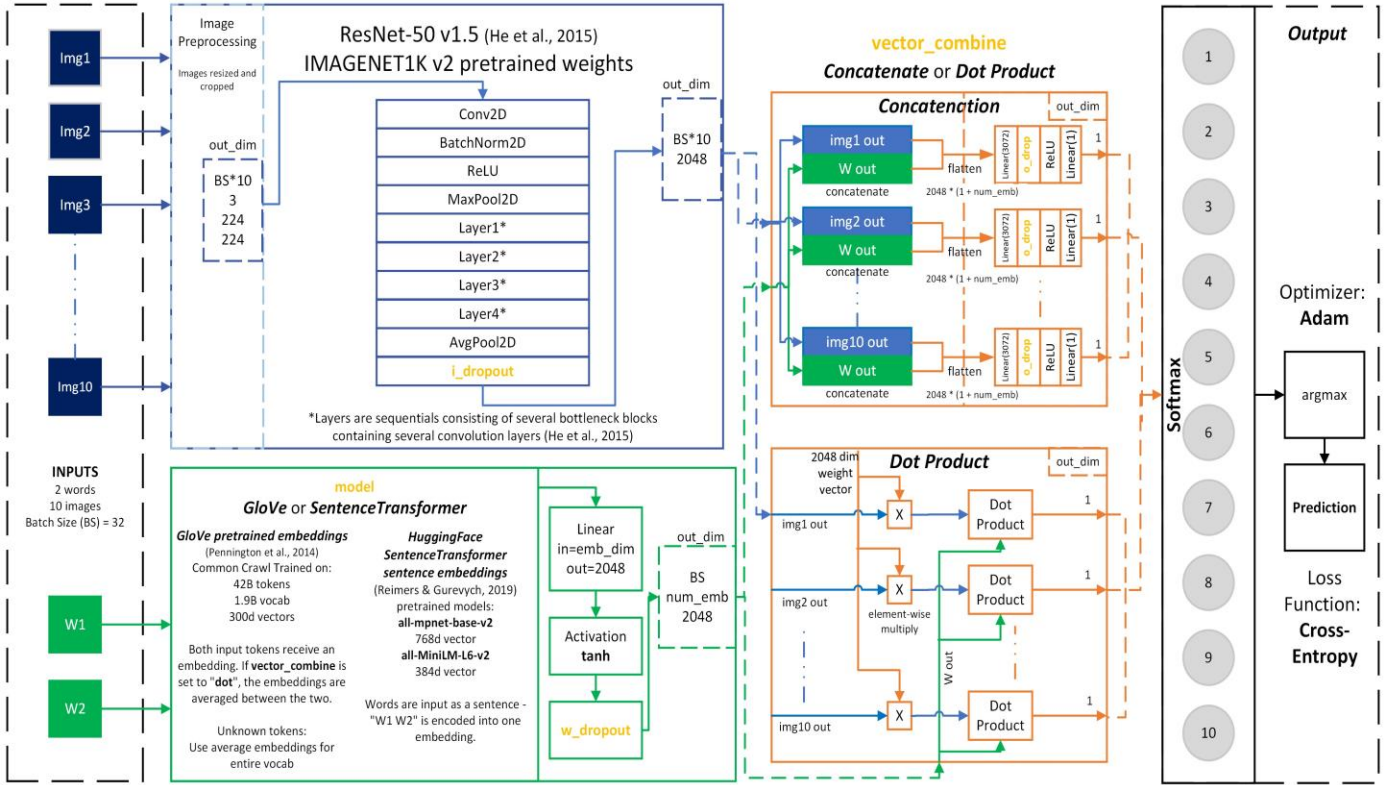


Fig. 2. Model Architecture. The figure shows both the image processing and the text processing that occurs, then the tensor combination step, and followed by the softmax output layer where the model makes its prediction. The parameters with the yellow font were hyperparameters that were optimally identified through multiple training sweeps.

“related” to the target (image **1**) or the word “*nickel*”. Many of these images are obviously not related according to human intuition, and because of this, the model may not always be challenged enough to learn some of the more difficult and ambiguous samples.

The given samples are imperfect for the task, but with some manipulation, they can be used to optimally train the model to generalize. One potential method was simply to leave the image options as is, and randomly shuffle the samples as 80% training and 20% development. The possible benefit of leaving the options as-is is that the ambiguous image options that are intentionally given to some of the samples will stay together, which may force the model to learn more by distinguishing between similar images. However, this method would mean that many image options from the training data would also show up in the validation data and could mean that the validation performance does not truly represent how well the model generalizes. Another option that was considered was to randomly shuffle the image set and split the images as 80% exclusively for training and 20% exclusively for validation data. This way, there would be no leakage of image options showing up in both the training and the development sets, and perhaps more importantly, it would provide a way to augment “new” samples by shuffling the image choices for every sample in each training epoch.

Another possibility that was not attempted in this research would be to train a separate model to identify the 9 most similar images for each target image and use those options for each sample. This would have been a more expensive approach but could have helped with optimally training the model to identify the target out of a group of similar images.

III. METHODS

The first step in creating the model for this task was to create a baseline framework for what the model would need to look like. Given that the inputs feature two different data types (images and text), there needs to be a way to combine their features and allow the model to make a prediction. The way I achieve this is by using pre-trained models to transform the image and text inputs into equal dimensional tensors. Then the tensors need to be combined, so they can then be fed into the softmax layer, where the model will make its predictions. The full model architecture is shown in *Figure 2*. Once the architecture is laid out, the next step was to start training and exploring the hyper-parameters.

I utilized PyTorch [1] with a CUDA compatible GPU device to develop and train the deep-learning model. The full code and usage information for this project resides at <https://github.com/karechiga/visual-word-sense>. To store model performance history and track hyper-parameters, I utilized Weights and Biases [2].

A. Image Model

To train the model to learn the visual information for each sample, I implemented a pre-trained image embedding model. There are several state-of-the-art image models today that can achieve over 90% accuracy on the ImageNet dataset [3], however, many of them have billions of parameters which makes the computation very expensive. Using only one GPU with 32 GB of RAM and a batch-size of 32 samples (meaning 320 total images per batch), I chose to go with a model that could fit these constraints while also achieving a

high degree of accuracy. ResNet-50 v1.5 [4] fits the hardware capacity and has top 1 accuracy up to 80.858% on the ImageNet dataset using TorchVision's pre-trained weights [5].

The ResNet-50 architecture includes several Convolutional Neural Network (CNN) and Batch Normalization (BatchNorm) layers with Rectified Linear Unit activation functions (ReLU). The images first get resized to 224 x 224 and the values are scaled and normalized before being input into the ResNet-50 layers. The network outputs a 1,000-dimension logistic layer, which can be input into a softmax layer to classify the image as one of the 1,000 categories in ImageNet. For the purposes of this model, I remove the logistic layer and output a 2048-dimension vector containing the necessary visual information. Additionally, I appended an optional dropout to the last layer to be utilized during training.

During training, I ran into a challenge when trying to maximize the validation accuracy. I was able to overfit the model on the training data, but the validation accuracy would not exceed 25%. What I found was that during evaluation mode, the batch normalization layers in the ResNet-50 network behave differently than in training mode. The reason for this is that during training, the model calculates mean and standard deviation for each batch and updates the running average for each batch. During evaluation mode, the batch normalization layers use the average mean and standard deviation that was calculated during training and do not update them. The problem my original model had was that I was not actually inputting the images into the ResNet-50 network **in batches**; they were being input one image at a time, then the outputs were appended at the end. This causes the batch normalization calculations to be severely unstable and the model struggles in evaluation mode as a result. The solution I ended up implementing was to preprocess all images in the batch first, then input all 320 images (batch size of 32 and 10 images per sample) as a single batch into the ResNet-50 layers. This outputs a 320 x 2048 tensor and effectively calculated the running mean and standard deviation for the batch normalization layers, allowing the model to learn during training and generalize for evaluation mode.

B. Word Model

The goal with the word model is to transform the text input into a 2048-dimension tensor that contains the semantic information about the multi-word input. The way this can be achieved is by using pre-trained embeddings for the input text. I initially tried to train a model using static *GloVe* embeddings [6] but ended up using a *SentenceTransformer* model [7] to encode the entire input phrase.

1) *GloVe*

Initially, I used pre-trained, static *GloVe* 300-dimension embeddings which were trained on a 42 billion token corpus and contained a 1.9-million-word vocabulary. Each of the 1.9 million words were assigned a token index. Any words that were not in the vocabulary received an '<unk>' token index and were assigned an embedding that is the average of all other embeddings.

As stated in Section II, all the given SemEval training data contained a total of two tokens per sample. The two tokens each receive an embedding vector and are either concatenated

together or averaged into one vector. Then they go through a feed forward network and are output as a 2048-dimension vector, matching the dimensionality of the output of the ResNet model.

The main issue with using the *GloVe* embeddings was that over 21% of the provided inputs in the training and development data contained unknown tokens. The model still yielded okay results on the development data, however, using *SentenceTransformer* ended up being the better choice due its coverage of all tokens in the data and also utilizing a state-of-the-art *Sentence-BERT* model.

2) *SentenceTransformer*

I used Hugging Face's *SentenceTransformer* library which includes multiple high-performing *Sentence-BERT* models for evaluating semantic textual similarity between sentences[7]. I chose to use *all-mpnet-base-v2* based on its high score on sentence embedding tasks as well as being relatively small and fast model. I also attempted to use the smaller model, *all-MiniLM-L12-v2*, but after several sweeps of hyper-parameters, *all-mpnet-base-v2* stood out above the other model combinations.

The input context phrases were batched and directly input into the sentence transformer which only takes milliseconds to generate 768-dimensional embeddings. Using this sentence transformer proved to be a much more effective and robust way to represent the entire phrase's meaning than using *GloVe* to retrieve static individual embeddings for words and concatenate or average their embeddings. The 768-dimensional embeddings were fed through a feedforward network and output as a 2048-dimensional tensor with dropout applied to it.

C. Combining the Image and Word Vectors

Before asking the model to make predictions, the 2048-dimensional outputs from the word and image models needed to be combined somehow. For each image in each sample, the output of the word model is combined with the output of the image. The options were to either concatenate the vectors, then flatten them into one vector, and input into a feed-forward network, or compute the dot product (cosine similarity) between the vectors, which would output directly into the logistic output layer. Concatenation seemed to work better when using the *GloVe* embeddings, however, cosine similarity in combination with the sentence transformer word model yielded the best performance, as shown in *Figure 3*. This may be because cosine similarity is what *Sentence-BERT* models are designed for, as they perform very well in comparing sentence to sentence embeddings. In this case, the comparison is effectively with an image embedding.

D. Training the Model and Tuning Hyper-Parameters

The output of the model is a 10-dimension logistic vector that becomes a percentage for each image choice when it is transformed with a softmax function. The model uses a multi-class cross entropy loss function to reward confident and correct picks and especially discourage confident picks that

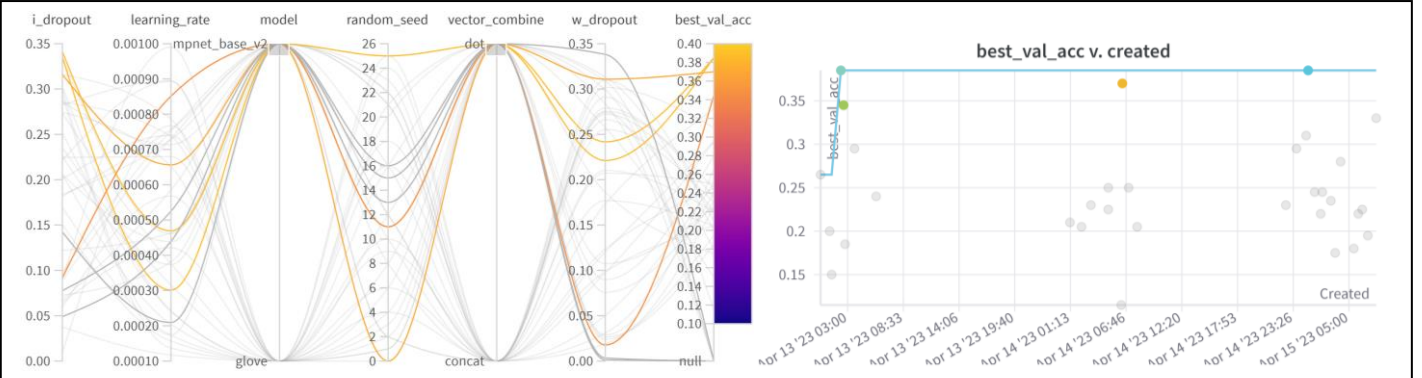


Fig. 3. Hyper-Parameter Sweep: Several sweeps were ran using smaller training and dev sizes. This particular sweep illustrates the effectiveness of using the mpnet-base-v2 model in conjunction with a dot product to combine the image and text output tensors. The top four models in this sweep all use the same model and sentence combination. Other factors, such as dropout on the output of the image and word layers ($i_dropout$ and $w_dropout$) and learning rate, do not seem to have as great an impact on validation accuracy for this particular sweep.

are incorrect. Once the loss is calculated for each batch, I chose to use an Adam[8] optimizer to update weights for the next training iteration. The training loop continues for multiple epochs until the early stopping threshold is not reached, at which case the program saves the best model up to that point in terms of validation accuracy. For most of the model training, I set the early stopping at 5 epochs without improvement and a threshold of 0.01, meaning that the validation accuracy needed to improve by at least 1% to reset the early stopping counter.

The learning rate in the Adam optimizer is one of the many hyper-parameters that I had to tune during development. Using Weights and Biases[2], I ran several sweeps of my model’s hyper-parameters on smaller datasets to learn what works the best for learning the training data and generalizing for the dev data. As expected, learning rate had the biggest impact on performance, as generally the best models had a learning rate between 0.0002 and 0.0006. Once the possible learning rates were narrowed down, I was able to explore other parameters more closely such as the word models and vector combinations as shown in Figure 3. Then, running a final sweep with the mpnet-base-v2 and dot product vector combination, I was able to identify optimal dropout values for the word and image outputs, which turned out to be between 20% and 30%. In earlier sweeps, I looked at the number of layers in feedforward steps as well as activations, and found that their impacts were mostly negligible, so I settled on a single feedforward layer in the word model with a tanh activation function. I also investigated the structure of the sentences to be input into the sentence transformer. This parameter turned out to be most effective with the default order of context words, “word0 word1”, instead of changing to order to “word1 word0” or inputting “word0 in the context of word1” into the sentence transformer. Once I found all my optimal hyper-parameters, I was able to train my models with the full dataset and evaluate them on the test data.

IV. RESULTS

I trained four models with the given data and evaluated them on the 463 test samples provided by SemEval-2023 task 1. The four models had the results shown in Table I. I hoped to see what the difference was in shuffling training data image options or leaving the options as is. Additionally, two of the models concatenated the output image and word vectors,

whereas the other two used a dot product to combine them. The results show clearly that shuffling the image options and using the dot product on output vectors yields the best results in comparison to the other three models. Shuffling the image choices had a particularly huge effect on the model’s ability to generalize on the test dataset. Model 2 did not fit the training and validation datasets as well as the other three models, but it still had better test results than the two models where the image choices were left the same. Likewise, Model 1 had similar validation results to Models 3 and 4, but it was able to stand out in performance due to randomizing the image choices and combining the word and image vectors using a dot product.

Model 1 was far from perfect, but its 29.4% accuracy performed well above the random guessing baseline of 10%. Even when it was incorrect, it was still often on the right track. Model 1 had a top 3 accuracy of 61%, and only 13% of the correct answers were predicted to be in the bottom four choices. The histogram in Figure 4 illustrates this distribution. Of course, the model had some challenges as well.

Table II shows a few examples of predictions with mixed results for different word senses. I used these examples to demonstrate some instances where the model struggled and why it happened to struggle on some samples in contrast with other samples that were successful predictions. The test data was much more challenging for the model than the training

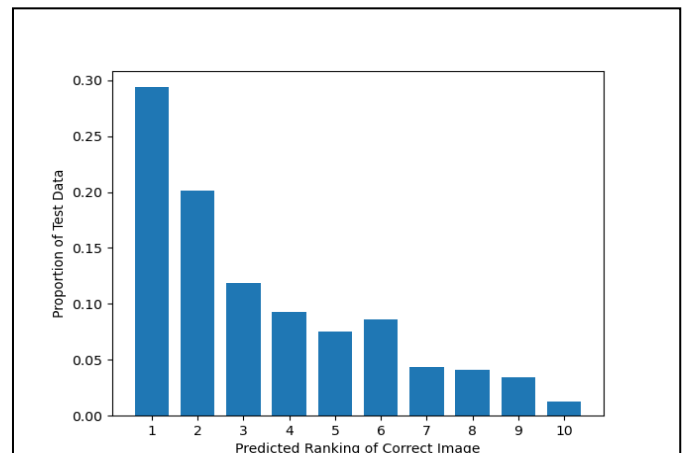


Fig. 4. Ranking Distribution for Model 1: Over 61% of the correct answers were predicted in the model’s top 3 choices, and only 13% were predicted in the bottom 4 choices.

| # | Word Model | Image Choices | Image-Word Combine | Train Accuracy | Validation Accuracy | Test Accuracy | Test MRR |
|---|-------------------|---------------|--------------------|----------------|---------------------|---------------|---------------|
| 1 | all-mpnet-base-v2 | shuffled | dot product | 0.8457 | 0.5714 | 0.2937 | 0.5029 |
| 2 | all-mpnet-base-v2 | shuffled | concatenate | 0.5768 | 0.4505 | 0.2333 | 0.4472 |
| 3 | all-mpnet-base-v2 | not shuffled | dot product | 0.7564 | 0.5859 | 0.1857 | 0.4140 |
| 4 | all-mpnet-base-v2 | not shuffled | concatenate | 0.7607 | 0.5633 | 0.1879 | 0.4010 |

TABLE I. MODEL PERFORMANCE ON TEST DATA

and development datasets and often was able to fool the model with very similar image choices. *Table II* illustrates some of the difficult choices the model had to make.

We observe a few cases in *Table II* where the model ranked the correct answer very low. The model struggled with the “*tea herb*” example because it likely had a hard time recognizing the correct image as *tea* or an *herb*. This is a challenging example, and one that may not have been represented well in the ImageNet dataset that the ResNet model weights were trained on. The model ended up predicting something that looks like tea with some tea leaves inside the glass. This is not a completely unreasonable prediction and one that some humans might make given the context words and image options. For “*tea beverage*”, the model had the right intuition to infer that the correct image would include a beverage, but it did not select the correct image where it may have been distracted by the other elements in the photo not relating to the tea beverages.

For the phrase “*chamaeleon reptile*” the model predicts the image of a crocodile, meaning the model was able to make the classification of a reptile, however, the true correct answer was ranked 10 out of 10 for this sample. The initial thought was that maybe the model had a hard time with the picture since the chamaeleon is blending in. To test this theory out, I ran this image through the ResNet-50 model with the initial pre-trained weights, and it correctly classifies the image as index 47 of ImageNet, an African chameleon or Chamaeleo chamaeleon [3]. When I change the input phrase from “*chamaeleon reptile*” to “*lizard reptile*” or “*chameleon lizard*”, the model still chooses the crocodile image, but when I change it to “*chamaeleon lizard on a plant*”, the model moves the correct picture from 10th place to 3rd place (14% confidence) behind another picture with a lizard on a plant and a picture of a plant. This seems to be a case where the model does not fully understand the word “*chamaeleon*” in the context of *lizards* and *reptiles* and does not know how to differentiate from other lizard-looking reptiles without adding additional context. On the other hand, “*chamaeleon constellation*” was predicted correctly by the model with 35% confidence. In second place (33%) was another image of an outer space galaxy. In this case the model had a good understanding of the context and made the correct prediction.

In the case of “*cricket insect*” and “*cricket game*,” the model was confident in a wrong, but related image in both cases. It seemed to understand that *cricket game* was referring to an image of a sport but did not have enough knowledge of what the game of *cricket* looks like. Likewise, it knows that *cricket insect* is a bug of some sort, but it likely went with the image that was most clearly an insect.

V. DISCUSSION

Overall, the model performed reasonably well, and there is still a lot of room for more improvement. The results show that the model, while not perfect, has learned how to relate ambiguous words with minimal context to representative images. It demonstrated a broad understanding of many different word senses but lacks some knowledge when it comes to some of the rarer and more niche meanings. There are several ways to train an even better model.

Randomizing the training choices had a huge impact on this model’s performance on the test data. It seems that shuffling these options augments additional data to the model, since in each epoch, the samples will have new options to choose from. It would be interesting to consider what would happen if we had an optimal and deterministic way to shuffle the data. As mentioned in Section II, if we trained an image model to identify some of the most similar images to the solution image, that may be a better way to challenge the model to identify the subtle details that distinguish the similar image classes.

Another possibility would be to augment more samples by slightly modifying the context input with synonyms or adding adjectives. Perhaps a separate sentence transformer model could have been used for this to find other contexts that have similar semantic meaning. These additional text inputs could be added as new samples with same output and different image choices than the original.

In addition, the pre-trained models can be improved to more state-of-art ones. In particular ResNet-50, while have an impressive benchmark for a being a relatively smaller model, could be replaced with a model like CoCa[9] which has achieved 91% accuracy on the ImageNet dataset. However, this would be a very computationally expensive model to train on due to having over 2 billion parameters. A possibly more viable model would be TinyViT (Tiny Vision Transformer) [10] which still performs extremely well with classification tasks on the ImageNet dataset.

While there is vast room for improvement for this task and other related word sense disambiguation tasks, it is important to discuss and understand the ethical concerns of this research. With any kind of machine learning, human biases and negative stereotypes can leak into the datasets if we are not careful. For this visual word sense disambiguation task, a model can learn harmful language and representations for groups of people and create even stronger biases if released publicly. There are some sensitive words and images in this dataset, and it is important to be cautious about how the model is trained to understand certain language and images. Additionally, privacy is often a concern with language and image models. Many of the images contain people and entities

















| Word Inputs | Prediction | Correct Answer | Word Inputs | Prediction | Correct Answer |
|--------------------|--|---|--------------------------|---|---|
| tea herb |  Confidence: 29.02% |  Confidence: 3.51% (8) | tea beverage |  Confidence: 96.45% |  Confidence: 0.00015% (7) |
| chamaeleon reptile |  Confidence: 90.50% |  Confidence: 0.0096% (10) | chamaeleon constellation |  Confidence: 35.06% |  Confidence: 35.06% (1) |
| cricket game |  Confidence: 98.97% |  0.30% (2) | cricket insect |  Confidence: 96.72% |  Confidence: 2.78% (2) |
| viola music |  Confidence: 51.62% |  Confidence: 51.62% (1) | viola shrub |  Confidence: 44.29% |  Confidence: 44.29% (1) |

TABLE II. NOTEABLE PREDICTIONS FOR DIFFERENT WORD SENSES

that should be protected from public release unless they give their permission.

VI. CONCLUSION

While the accuracy achieved by the best model exceeds the random baseline, further improvements are necessary before real-world applications can be implemented. We show that even a relatively low-cost model architecture like this one can learn and generalize word senses with minimal context on unseen test data. However, the model struggles with specific and rarer words and can be improved in future work. There are some opportunities for data augmentation by creating separate models to manipulate and add new samples. Additionally, enhancing the pre-trained ResNet-50 model to a state-of-the-art image model could improve the performance of the image processing. Combining these approaches and continuing to research how humans interpret and distinguish word senses could hold significant potential for enhancing Visual Word Sense Disambiguation models and fostering research progress in natural language processing.

REFERENCES

- [1] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library." arXiv, Dec. 03, 2019. doi: 10.48550/arXiv.1912.01703.
- [2] L. Biewald, "Experiment Tracking with Weights and Biases." 2020. [Online]. Available: <https://www.wandb.com/>
- [3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255. doi: 10.1109/CVPR.2009.5206848.
- [4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition." arXiv, Dec. 10, 2015. doi: 10.48550/arXiv.1512.03385.
- [5] V. Vryniotis, "How to Train State-Of-The-Art Models Using TorchVision's Latest Primitives," Nov. 18, 2021.

- <https://pytorch.org/blog/how-to-train-state-of-the-art-models-using-torchvision-latest-primitives/>
- [6] J. Pennington, R. Socher, and C. Manning, "Glove: Global Vectors for Word Representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1532–1543. doi: 10.3115/v1/D14-1162.
- [7] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks." arXiv, Aug. 27, 2019. doi: 10.48550/arXiv.1908.10084.
- [8] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization." arXiv, Jan. 29, 2017. doi: 10.48550/arXiv.1412.6980.
- [9] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu, "CoCa: Contrastive Captioners are Image-Text Foundation Models." arXiv, Jun. 13, 2022. Accessed: May 01, 2023. [Online]. Available: <http://arxiv.org/abs/2205.01917>
- [10] K. Wu *et al.*, "TinyViT: Fast Pretraining Distillation for Small Vision Transformers." arXiv, Jul. 21, 2022. Accessed: May 01, 2023. [Online]. Available: <http://arxiv.org/abs/2207.10666>