DeClip: Hard Negative Mining through Image Decomposition and Inpainting

Zhanhao Liu (zhanhaol@umich.edu), Huanchen Jia (jhuanch@umich.edu),
Qiulin Fan (rynnefan@umich.edu), Lingyu Meng (jmly@umich.edu)

03/14/2025

1 Abstract

Contrastive Language–Image Pretraining(CLIP) excels in multimodal learning, enabling zero-shot classification, cross-modal retrieval, and transfer learning. However, its reliance on global image-text alignment limits its ability to capture localized features, weakening performance in fine-grained visual tasks. To address this, DeClip introduces image-based hard negatives by modifying key image-caption pairs, enhancing model robustness and discrimination. This improves CLIP's ability to distinguish fine-grained details, strengthening its effectiveness in contrastive learning and multimodal tasks.

2 Introduction

CLIP[1] (Radford et al., 2021), trained on full images, struggles to learn localized features due to its reliance on global image-text alignment, limiting its effectiveness in fine-grained visual-text tasks like object classification and caption-based retrieval. This limitation hinders its ability to capture subtle visual differences, which is especially problematic in safety-critical domains such as medical imaging and robotics, where precise distinctions—like identifying diseased tissue or ensuring accurate tool positioning—are crucial. Solving this problem would improve CLIP's ability to learn localized visual details, enhancing its adaptability to tasks requiring fine-grained reasoning and ultimately making it more effective in real-world multimodal applications.

3 Proposed method

3.1 Overall Approach

Prior research suggests that, for each pair of correct (image, caption) example, by rewriting the caption, e.g. reordering and changing its constituents, we can generate into a very hard negative. (Fan, Krishnan, et. al.[2]). Our idea has similarities and extensions. We propose another idea to mine hard negaives on the image side. For each true example in a subset of the dataset, we remove some key components from the image and keep the rest, leave the resulting image a high degree of similarity to the original image, but does not fit the caption, as a hard negative. We then dually remove the keywords on the caption, as a new positive example.

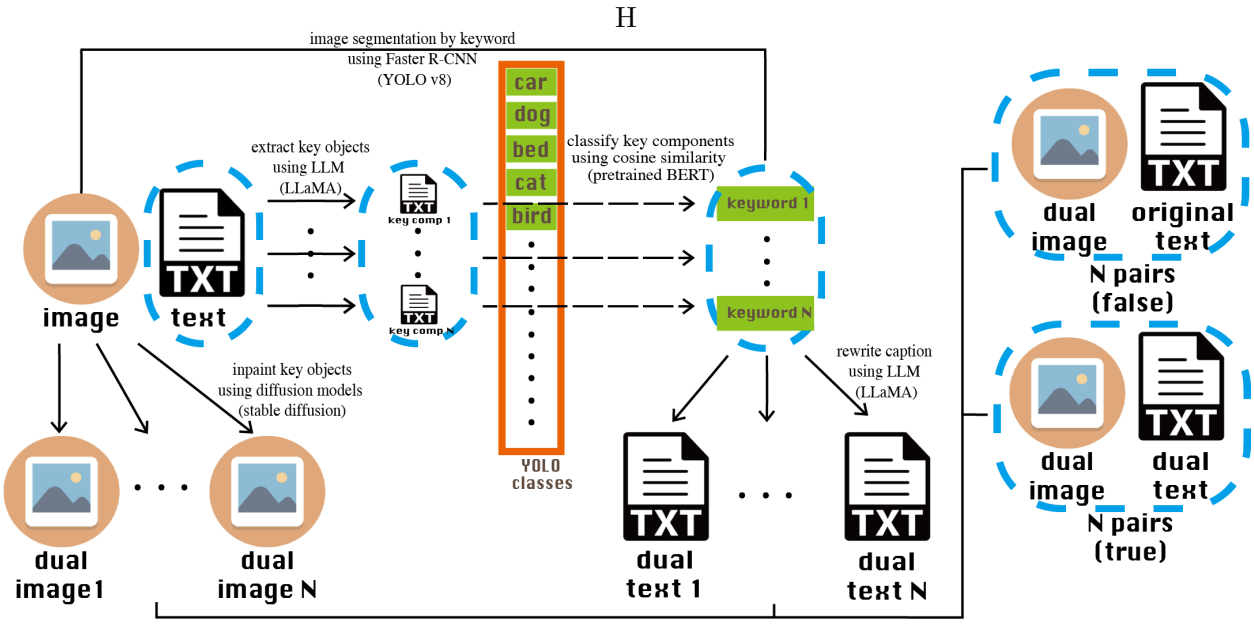

Figure 1: Overall approach

For the new image after removing the keyword, we call it **dual image**; for the new caption after removing the keyword, we call it **dual text**.

In concrete:

- We assign a small hyperparameter N (e.g., $N = 3$)
- Utilize the in-context learning capability of large language models to extract N key components (if possible), and categorize them into a few pretrained classes.
- Perform image segmentation using Fast RCNN models (e.g., `YOLO v8`) to extract N key components from the image, generate masks.
- Use diffusion model to do image inpaining, generating N new images by respectively removing the N components on the original image.
- Rewrite the original caption by LLM to remove each keyword from the text, and get N dual texts accordingly.

Our idea invokes the manifold hypothesis, aims at improving CLIP's composition-reasoning ability. Prior work has shown that the embedding manifold of CLIP is characterized as double-ellipsoids[3] (Levi, Gilboa, 2024), and we hypothesize: dual images with critical components removed characterize the local structure of the original image, as a valid interpolation; and the generation of dual texts can let the model learn the mapping between the sample image and the open neighborhood near the caption. We conjecture that our approach increases the combinatorial inference ability and robustness of the model on training with small datasets.

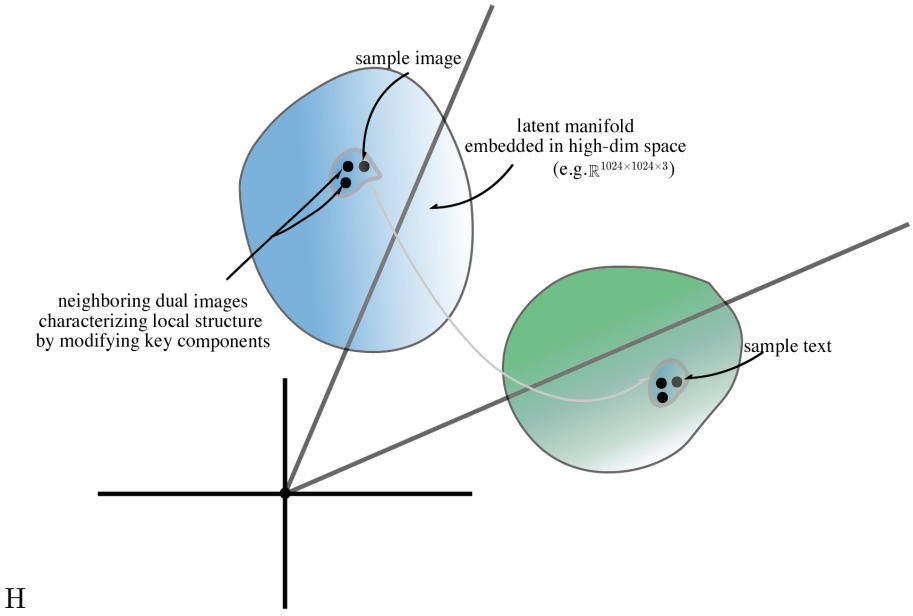

Figure 2: Embedding data manifold in Clip Space

4 Related work

4.1 Existing Methods:

CLIP's performance is shown recently that it could be enhanced by leverging the difficulty in training.
Currently the exisiting methods mostly improve in the following perspectives.

- **Hard Negative Mining**: Hard negatives closely resemble positive samples yet belong to different classes, making them difficult to distinguish. In contrastive learning, two main strategies address this: generation-based and regularization-based. Generation-based methods (e.g., NegCLIP[4] ((Yuksekgonul et al., 2022)) modify positive samples to create semantically similar but incorrect pairs, forcing models to learn subtle distinctions. Regularization-based methods use techniques like intra-modal contrastive loss and cross-modal ranking of hard negatives to refine learning.
- **Fine-Grained Cross-Modal Alignment Fine-Tuning**: Some previous works such as FILIP[5] (Yao et al., 2021) enhances multimodal alignment by matching local image patches with specific text tokens via a similarity matrix. Unlike CLIP's global image-text matching, it computes token-level contrastive losses, forcing visual and textual tokens to mutually select their most relevant counterparts. The regional information and details could be better learned this way.
- **Rewrite Empowered by Large Language Model**: LaCLIP[2] ((Fan, Krishnan, et. al) generates diverse text rewrites for each image caption via LLMs while preserving semantic meaning, boosting the performances significantly on datsets such as CC12M and LAION-400M .

4.2 Comparisons:

DeCLIP combines the idea of **Hard Negative Mining** and **Fine-Grained Cross-Modal Alignment Fine-Tuning** such that the model can deepen its understanding of the entire image by paying attention to the regional changes of the image. From the proposed method section, DeCLIP uses Large language model to capture keywords from the caption in the data set and replaces those keywords one at a time with pre-trained catogeries, forming our final traning set. It is different from **Hard Negative Mining** in applying the changes to **images** instead of **texts**, while it is improved based on **Fine-Grained Cross-Modal Alignment Fine-Tuning** by considering contexts of texts and layout of images

4.3 Core Model and Tool References:

Our framework integrates state-of-the-art multimodal tools. For text extraction, `spaCy` [6](SPACY · Industrial-strength Natural Language Processing in Python, n.d.) identifies and merges compound nouns. We then map these nouns to pre-trained categories via `word2vec-google-news-300` [7] (Google Code Archive - Long-term Storage for Google Code Project Hosting., n.d.) embeddings. For image segmentation, `yolov8n-seg.pt` [8] (Ultralytics, 2025)(a lightweight YOLOv8 variant pretrained on COCO) is used.
Inpainting leverages `runwayml/stable-diffusion-inpainting` [9](Stable-diffusion-v1-5/Stable-diffusion-inpainting · Hugging Face, n.d.) and `CLIP`, ensuring robust image–text understanding.

5 Preliminary

Pipeline for Caption-Guided Object Removal in Images

We have completed a pipeline that uses the image caption to guide the removal of specific objects. Currently, our system operates on a single image–caption pair. The code is provided in the Appendix A.

Example

Figure 3: **Caption:** *"A policeman stops on a street with a search dog."*

Step 1: Extraction of Nouns/Noun Phrases

Method: We use SpaCy[6] (SPACY · Industrial-strength Natural Language Processing in Python, n.d.)'s small English language model to extract nouns and noun phrases from the caption. For compound phrases such as "search dog," the entire phrase is treated as a single unit.
Example:

- Extracted Terms: "policeman", "street", "search dog"

Step 2: Classification into Object Categories

Method: The extracted noun phrases are mapped to one of the 80 object classes used by YOLOv8[8] (Ultralytics, 2025). Mapping is achieved using a pre-trained Word2Vec model.
Example:

- `policeman` → `person` | `street` → `bicycle` | `search dog` → `dog`

Ambiguous mappings (like "street") are discarded later if they do not appear in the detection step. When dealing with phrases like "search dog," we ensure that word phrases are considered based on the weight of each word.

Step 3: Object Detection and Mask Generation

Method:
YOLOv8 [8] (Ultralytics, 2025) is applied to the image to detect each of the classified objects. If an object is not detected, it is discarded from the pipeline. For each detected object, a binary mask is generated—white for the object and black for the background.
Example:

Figure 4: Dog-mask Person-mask

Step 4: Object Removal via Inpainting

Method:
We use a Stable Diffusion[9](Stable-diffusion-v1-5/Stable-diffusion-inpainting · Hugging Face, n.d.) inpainting model to remove each detected object using its corresponding binary mask. This process is guided by a prompt such as: *"Only blend the highlighted block seamlessly into the background."*
Example:
Future Work:
Removing multiple objects in a single mask is a straightforward extension of this approach.
Challenges:

Figure 5: Inpainted Output (Dog Removed)(Person Removed)

1. **Key Noun Extraction:** Initial attempts using large language models (e.g., GPT, LLaMA) were unsatisfactory. We opted to extract *all* nouns instead. Future work may explore random selection.

2. **Classification Difficulties:** BERT-based methods (even with the entire-caption context) did not yield accurate classifications.

3. **Bounding Box Precision:** Early tests using rectangular bounding boxes led to imprecise masks. More precise detection methods now provide smoother, more accurate masks.

6 Conclusion

In this work, we developed a caption-guided object removal pipeline that leverages YOLOv8-based object detection, stable diffusion inpainting, and large language models to extract key objects from image-caption pairs, classify them into predefined categories, and generate precise masks for targeted object removal. By modifying images, while preserving structural relevance, we created hard negative images that misalign with their original captions, enhancing contrastive learning in CLIP-based models and improving their ability to distinguish fine-grained differences between text and images. Additionally, the altered images were incorporated into the training dataset as false samples to find tune the existing model. While our approach demonstrates the feasibility, future work should focus on refining object classification, supporting multi-object handling, and scaling data generation to improve DeClip's effectiveness in multimodal learning.

7 Future Plan

- We will need to automate the procedure for relatively large dataset (e.g. 3 million images), requiring the balance between efficiency and quality of inpainting, otherwise we might introduce noisy generated images. So we may need to tweak the tech stack and architecture slightly after practice.

- We will further optimize the structure: if a class of examples are already well-learned, then we do not need to improve the accuracy of shotting concerning images. We will not generate $2N$ new examples for all existing examples, but reselect a subset of all images in a dataset for computation cost.

8 Contributors and Roles

- **Zhanhao Liu**
  - Authored the *Preliminary Section* of the report.
  - Implemented the *word matching* and *Stable Diffusion* code.
- **Huanchen Jia**
  - Authored the *Related Work* section of the report.
  - Researched state-of-the-art methods and reviewed the literatures for the group.
  - Implemented the *noun extraction* and *compound word emerging* code.
- **Qiulin Fan**
  - Conceived the idea.
  - Authored the *Proposed Method* section of the report.
  - Implemented the *object detection and mask generation* code.
- **Lingyu Meng**
  - Researched *datasets and benchmarks* for future work.
  - Authored the *Introduction* and *Conclusion* sections of the report.

References

[1] A. Radford, J. W. Kim, C. Hallacy, *et al.*, *Learning transferable visual models from natural language supervision*, 2021. arXiv: 2103.00020 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2103.00020.

[2] L. Fan, D. Krishnan, P. Isola, D. Katabi, and Y. Tian, *Improving clip training with language rewrites*, 2023. arXiv: 2305.20088 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2305.20088.

[3] M. Y. Levi and G. Gilboa, *The double-ellipsoid geometry of clip*, 2024. arXiv: 2411.14517 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2411.14517.

[4] M. Yuksekgonul, F. Bianchi, P. Kalluri, D. Jurafsky, and J. Zou, *When and why vision-language models behave like bags-of-words, and what to do about it?* 2023. arXiv: 2210.01936 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2210.01936.

[5] L. Yao, R. Huang, L. Hou, *et al.*, *Filip: Fine-grained interactive language-image pre-training*, 2021. arXiv: 2111.07783 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2111.07783.

[6] E. AI, *Spacy: Industrial-strength natura language processing in python*, version 3.7.0, Accessed: 2025-03-13, 2025. [Online]. Available: https://spacy.io.

[7] Google, *Word2vec-google-news-300: Pretrained word embeddings*, 2025. [Online]. Available: https://code.google.com/archive/p/word2vec/.

[8] Ultralytics, *Segment - ultralytics yolov8 docs*, Accessed: 2025-03-13, 2023. [Online]. Available: https://docs.ultralytics.com/tasks/segment/.

[9] S. AI, *Stable diffusion v1-5 inpainting*, Hugging Face Model Hub, Accessed: 2025-03-13, 2023. [Online]. Available: https://huggingface.co/stable-diffusion-v1-5/stable-diffusion-inpainting.

Appendix A: Code

```python
import spacy
from collections import Counter
nlp = spacy.load("en_core_web_sm")
def extract_nouns(sentence, top_n=5):
    doc = nlp(sentence)

    # Merge consecutive nouns into phrases (e.g., "plant species")
    phrases = []
    current_phrase = []
    for token in doc:
        if token.pos_ in ['NOUN', 'PROPN']:
            current_phrase.append(token.text)
        else:
            if current_phrase:
                phrases.append(" ".join(current_phrase))
                current_phrase = []
    # Process any remaining noun phrase
    if current_phrase:
        phrases.append(" ".join(current_phrase))

    # Count phrase frequency
    counts = Counter(phrases)
    return [phrase for phrase, _ in counts.most_common(top_n)]

sentence = "A policeman stops on a street with a search dog."
list_A = extract_nouns(sentence)
print("Extracted nouns/phrases:", list_A)
```

Listing 1: Code for Noun extraction

```python
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from gensim.models import KeyedVectors

# Predefined YOLO keywords
YOLOKeyWords = [
    "person", "bicycle", "car",
    # ...
    "vase", "scissors", "teddy bear",
    "hair drier", "toothbrush"
]

# Load the Word2Vec model (Google News vectors)
model = api.load("word2vec-google-news-300")

```

```python
def get_phrase_vector(phrase, model):
    tokens = phrase.split()  # Basic whitespace split
    token_vectors = []
    for token in tokens:
        try:
            token_vectors.append(model[token])
        except KeyError:
            print(f"Token '{token}' not found in vocabulary; skipping.")
    if token_vectors:
        return np.mean(token_vectors, axis=0)
    else:
        print("No valid tokens found in the phrase.")
        return None

def classify_word_by_similarity(phrase, class_words, model, threshold=0.0):
    phrase_vector = get_phrase_vector(phrase, model)
    if phrase_vector is None:
        return None

    class_vectors = []
    valid_class_words = []
    for w in class_words:
        try:
            vec = model[w]
            class_vectors.append(vec)
            valid_class_words.append(w)
        except KeyError:
            continue

    if not class_vectors:
        return None

    class_vectors = np.array(class_vectors)
    similarities = cosine_similarity(phrase_vector.reshape(1, -1), class_vectors).
        flatten()
    best_index = np.argmax(similarities)
    best_similarity = similarities[best_index]

    if best_similarity < threshold:
        return None
    return valid_class_words[best_index], best_similarity

# Example usage
# word = "policeman"
# result = classify_word_by_similarity(word, YOLOKeyWords, model, threshold=0.3)
# print("Nearest-Neighbor Class:", result)

# Classify a list of words
keywords = []
print(list_A)
for word in list_A:
    result = classify_word_by_similarity(word, YOLOKeyWords, model, threshold=0.3)
    if result:
        keywords.append(result[0])

print(keywords)
```

Listing 2: Code for words match

```python
# @title YOLO
from ultralytics import YOLO
import cv2
import numpy as np
import os

# Load YOLO model
model = YOLO("yolov8n-seg.pt")

# Detect and segment objects in the input image
results = model(input_image_path)
img = cv2.imread(input_image_path)
height, width = img.shape[:2]

# Initialize dictionary to store combined masks for each keyword
combined_masks = {k: np.zeros((height, width), dtype=np.uint8) for k in keywords}
label_detected = []

# Iterate over detection results and extract segmentation masks
for result in results:
    if result.masks is not None:
        for i in range(len(result.boxes)):
            class_id = int(result.boxes.cls[i])
            label = model.names[class_id]
            if label in keywords:
                label_detected.append(label)

                # Extract mask and binarize
                mask = result.masks.data[i].cpu().numpy()
                mask = (mask > 0.2).astype(np.uint8) * 255

                # Resize if necessary
                if mask.shape[0] != height or mask.shape[1] != width:
                    mask = cv2.resize(mask, (width, height), interpolation=cv2.
                        INTER_NEAREST)

                # Combine masks for the same label
                combined_masks[label] = cv2.bitwise_or(combined_masks[label], mask)

# Save each mask to the specified folder
maskFolder = "maskFolder"
os.makedirs(maskFolder, exist_ok=True)

for label, mask in combined_masks.items():
    mask_path = f"mask_{label}.png"
    output_image_path = os.path.join(maskFolder, mask_path)
    cv2.imwrite(output_image_path, mask)
    print(f"Saved mask for '{label}' as {output_image_path}")
```

Listing 3: Code for Object detection and mask generation

```python
# @title Stable Diffusion
```

```python
from diffusers import StableDiffusionInpaintPipeline
import torch
from PIL import Image
import os

# Load the inpainting model
model_id = "runwayml/stable-diffusion-inpainting"
pipe = StableDiffusionInpaintPipeline.from_pretrained(
    model_id, torch_dtype=torch.float16
)

# Set device
device = "cuda" if torch.cuda.is_available() else "cpu"
pipe = pipe.to(device)

# Load input image
input_image_path = "image.png"  # Replace with your actual image path
image = Image.open(input_image_path).convert("RGB")

# Ensure the output directory exists
outputfolder = "OutputFolder"
os.makedirs(outputfolder, exist_ok=True)

# Process each detected label
for label in label_detected:
    # Load corresponding mask
    mask_path = os.path.join(maskFolder, f"mask_{label}.png")
    mask = Image.open(mask_path).convert("RGB")

    # Define output path
    output_image_path = os.path.join(outputfolder, f"output_{label}.png")

    # Perform inpainting
    result = pipe(prompt=prompt, image=image, mask_image=mask).images[0]

    # Save the result
    result.save(output_image_path)
    print(f"Saved inpainted image for '{label}' as {output_image_path}")
```

Listing 4: Code for Noun extraction