Hands-on tutorial – CLIP for multimodal analysis

Savvas Zannettou

TU Delft

Agenda

- Introduction to the dataset to be used in this session
- Setting up our Computing Infrastructure (Kaggle in this case)
- Hands-on Session with OpenAI's CLIP
  - Loading CLIP model
  - Generating Embeddings
  - Performing Information Retrieval using CLIP
  - Performing Zero-Shot Classification using CLIP
  - Demonstrating relationships in CLIP embeddings

TUDelft

Dataset

Dataset

Exploring Hate Speech Detection in Multimodal Publications

Raul Gomez[1,2], Jaume Gibert[1], Lluis Gomez[2], Dimosthenis Karatzas[2]
[1]Eurecat, Centre Tecnològic de Catalunya, Unitat de Tecnologies Audiovisuals, Barcelona, Spain
[2]Computer Vision Center, Universitat Autònoma de Barcelona, Barcelona, Spain
{raul.gomez, jaume.gibert}@eurecat.org, {lgomez,dimos}@cvc.uab.es

- 150K image and text pairs
- Manually annotated as Racist, Homophobic, Sexist, Religion-based, other hate and no-hate

Figure 1. Tweets from MMHS150K where the visual information adds relevant context for the hate speech detection task.

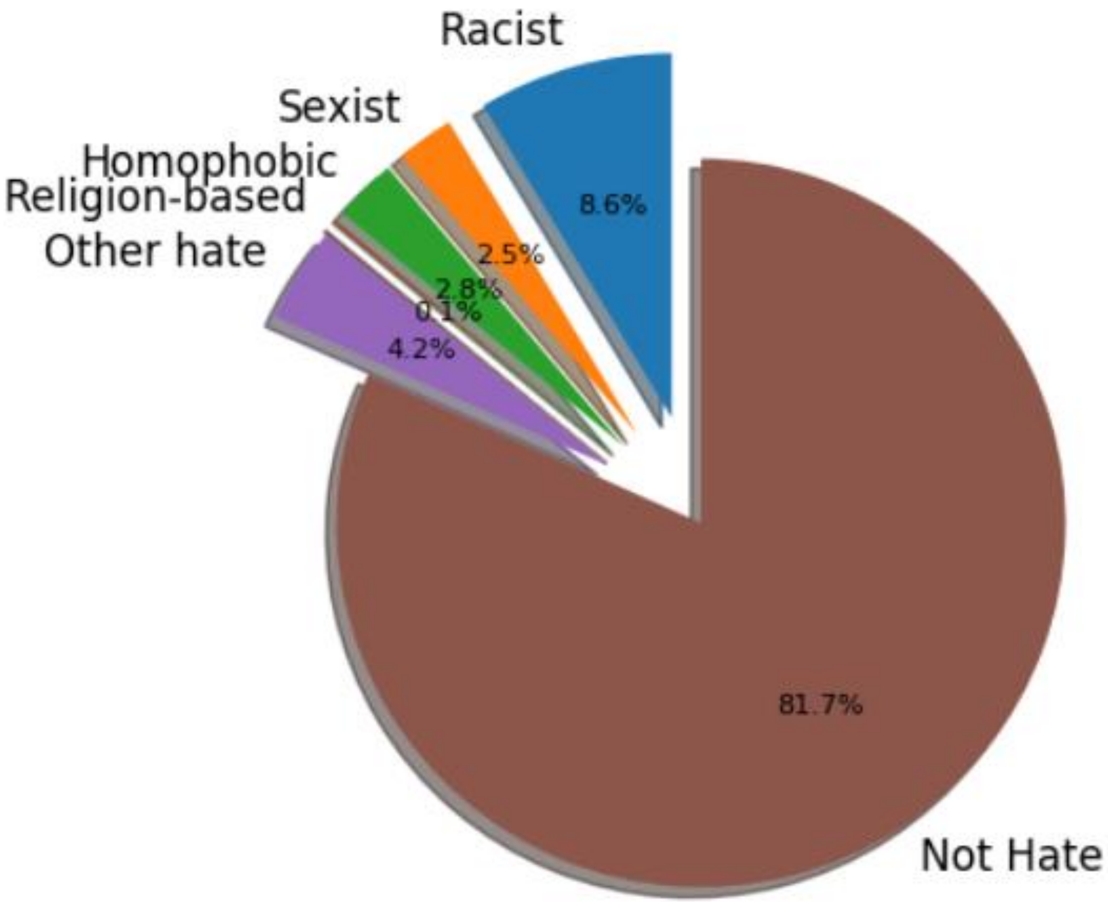
TUDelft

Data Collection Procedure

1. 51 terms included in the Hatebase dictionary
2. Collected tweets including these terms (Sep. 2018 – Feb. 2019)
3. Filtered tweets that include images
4. Annotated images using Mechanical Turk Workers

Data Annotation

- **Six classes:**
  - No attacks on any community
  - Racist
  - Sexist
  - Homophobic
  - Religion-based attacks
  - Other Hate



- Three annotators per tweet considering both text and image

An additional filtering step from my side

- Upon inspection of the dataset, I noticed that there were some images that were sexually explicit
  - E.g., showing human genitals
- Run a lightweight nudity detection model in Python (i.e., nudenet)
- Identified and removed from the dataset 1.2K images that include explicit sexually explicit content

TU Delft

Warning on the dataset

- This dataset contains **hate speech, offensive language, and harmful stereotypes**.
- Also, while I made a best effort to remove sexually explicit images, some sexually explicit images might still be in the dataset
- These materials are used **solely for research and educational purposes**.
- **We do not endorse or support** the views, language, or imagery included.

→ If you are not comfortable with working with this dataset for this hand-on session, please let me know and we can find an alternative dataset

Hands-on session

Setting up the Computing Infrastructure (Kaggle)

Using Kaggle

- Visit https://www.kaggle.com/ and create a free account
  - Creating with Google account is the fastest way

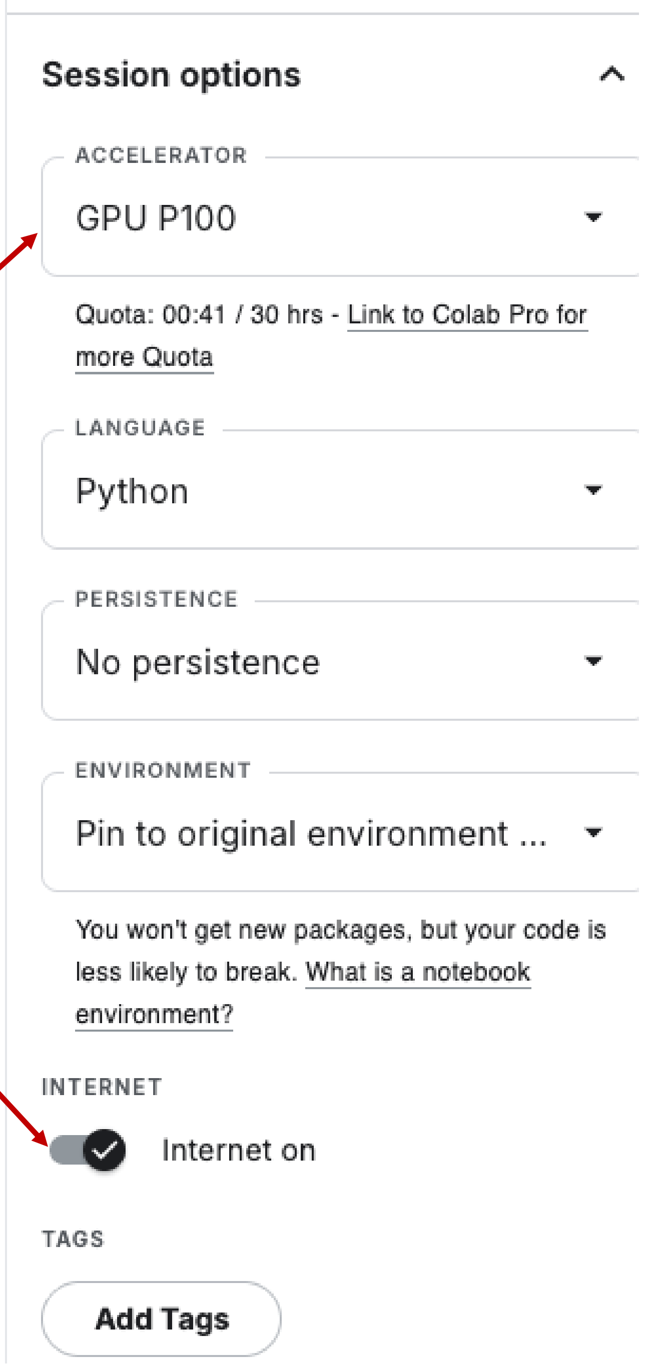
- Verify your account with your phone number so you get access to GPU resources
  - Press the top right profile icon
  - Go to “Settings”
  - Then “phone verify”
  - Input your phone number and follow the on-screen instructions

Using Kaggle

- Click here to get access to the Jupyter Notebook with the code
  - Press ”Copy and Edit” to see the code

- Change the session options (right hand side) so that:
  - You have a GPU accelerator (P100)
  - Internet is on (needed to install necessary Python packages)

- Then, you can run all the cells by clicking “Run all” on the top



Procedure

1. Loading CLIP model

2. Use pretrained OpenAI CLIP model to
   - Generate image and text embeddings (for a small sample of the dataset ~20K)

3. Demonstrate three use-cases
   - Information Retrieval
   - Zero-Shot Classification
   - Relationships between embeddings (text + text, text + image, weighted text+image)

TU Delft

Loading CLIP model and Basic Demo

```python
import torch
from PIL import Image
import open_clip
from pathlib import Path
from typing import List, Tuple
import numpy as np
import json
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

model, _, preprocess = open_clip.create_model_and_transforms('ViT-B-32', pretrained='openai')
model.eval()
tokenizer = open_clip.get_tokenizer('ViT-B-32')

image = preprocess(Image.open("/kaggle/input/clip-example/CLIP.png")).unsqueeze(0)
text = tokenizer(["a diagram", "a dog", "a cat"])

with torch.no_grad(), torch.autocast("cuda"):
    image_features = model.encode_image(image)
    text_features = model.encode_text(text)
    image_features /= image_features.norm(dim=-1, keepdim=True)
    text_features /= text_features.norm(dim=-1, keepdim=True)

    text_probs = (100.0 * image_features @ text_features.T).softmax(dim=-1)

print("Label probs:", text_probs)  # prints: [[1., 0., 0.]]
```

1. Load OpenAI's CLIP model

2. Define image (CLIP's diagram in this case) and textual prompts

3. Generate Embeddings for the image + text prompts

4. Calculate probabilities based on image and text embeddings

```
open_clip_model.safetensors: 100%  605M/605M [00:02<00:00, 257MB/s]
Label probs: tensor([[0.9940, 0.0040, 0.0020]])
```

"a diagram" received a probability of 0.994

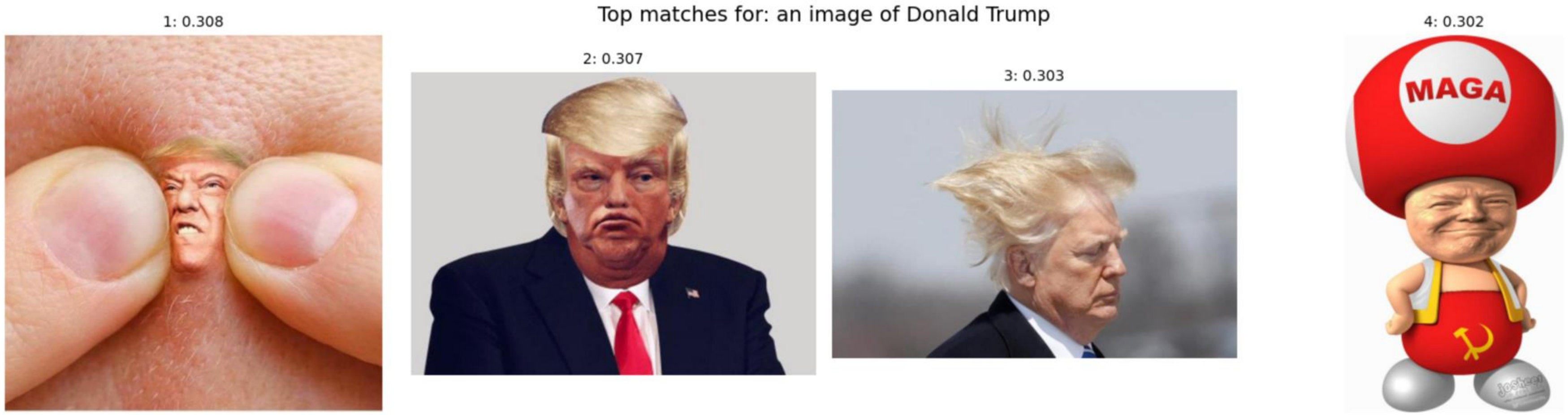
Use-Case Information Retrieval

```python
"""Rank images by similarity to a text prompt. Returns (idx, paths, scores)."""
@torch.no_grad()
def topn_for_text(prompt: str,
                  img_emb: np.ndarray,
                  image_paths: List[Path],
                  model, tokenizer,
                  topn: int = 4) -> Tuple[np.ndarray, List[Path], np.ndarray]:
    device = next(model.parameters()).device
    # text → embedding (normalized)
    toks = tokenizer([prompt]).to(device)
    t = model.encode_text(toks)
    q = (t / t.norm(dim=-1, keepdim=True)).detach().cpu().float().numpy()  # [1, D]

    # (re)normalize images for safety; then cosine via dot product
    img = img_emb / (np.linalg.norm(img_emb, axis=1, keepdims=True) + 1e-9)
    scores = (img @ q.T).squeeze(1)  # [N]

    idx = np.argsort(-scores)[:topn]
    return idx, [image_paths[i] for i in idx], scores[idx]

"""Simple grid viz of top-N images + scores."""
def show_topn(paths: List[Path], scores: np.ndarray, cols: int = 4, title: str = None):
    import math
    n = len(paths); rows = math.ceil(n / cols)
    fig, axes = plt.subplots(rows, cols, figsize=(4*cols, 4*rows))
    if title: fig.suptitle(title, fontsize=14)
    axes = np.atleast_2d(axes)
    for i in range(rows * cols):
        r, c = divmod(i, cols)
        ax = axes[r, c]; ax.axis("off")
        if i < n:
            ax.imshow(Image.open(paths[i]).convert("RGB"))
            ax.set_title(f"{i+1}: {scores[i]:.3f}", fontsize=10)
    plt.tight_layout(); plt.show()
```

Method to find top-n images based on text embedding

Helper to visualize results

Information Retrieval - Results



TU Delft

Use Case – Zero Shot Classification

```python
# 1) Build text features for a list of people (prompt ensemble + mean)
@torch.no_grad()
def build_people_text_features(
    names: List[str],
    model,
    tokenizer,
    templates: List[str] = None
) -> Tuple[List[str], torch.Tensor]:
    device = next(model.parameters()).device
    if templates is None:
        templates = [
            "a photo of {}",
            "an image of {}",
            "a portrait of {}",
        ]
    feats = []
    for name in names:
        prompts = [t.format(name) for t in templates]
        toks = tokenizer(prompts).to(device)
        t = model.encode_text(toks)
        t = t / t.norm(dim=-1, keepdim=True)      # normalize each prompt
        f = t.mean(dim=0, keepdim=True)           # ensemble → mean
        f = f / f.norm(dim=-1, keepdim=True)      # re-normalize
        feats.append(f)
    text_feats = torch.cat(feats, dim=0)          # [C, D]
    return names, text_feats
```

Build a small ensemble with 3 text prompts for classification

```python
# 2) Turn image embeddings + text features into probabilities with CLIP
@torch.no_grad()
def probs_for_subset(
    img_emb_subset: np.ndarray,      # [M, D] (L2-normalized)
    text_feats: torch.Tensor,        # [C, D] (L2-normalized)
    model
) -> np.ndarray:                     # [M, C] probabilities
    # cosine = dot product for normalized vectors
    img_t = torch.from_numpy(img_emb_subset).float()          # CPU is fine for M=8
    cls_t = text_feats.float()
    logits = img_t @ cls_t.T                                  # [M, C]
    # apply CLIP temperature (logit_scale)
    logit_scale = model.logit_scale.exp().item()
    logits = logits * logit_scale
    probs = torch.softmax(logits, dim=-1).cpu().numpy()
    return probs
```

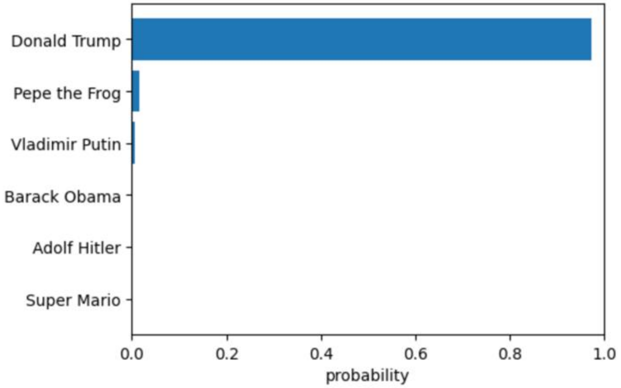
Calculate probabilities

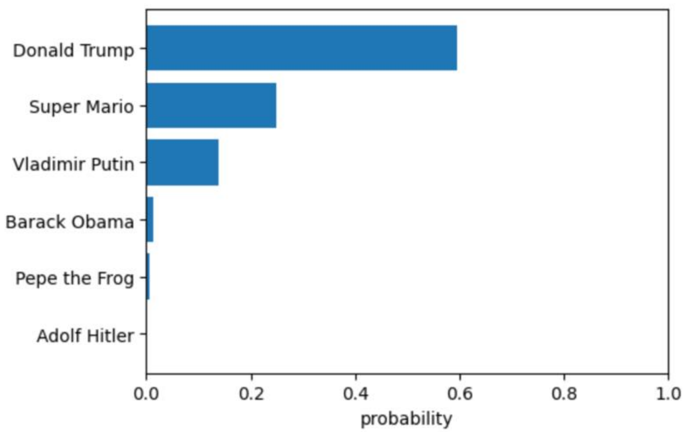
```python
# 3) Plot: each row = image, right column = horizontal bar chart of top-k classes
def plot_images_with_probs(
    paths: List[str],
    probs: np.ndarray,              # [M, C]
    classnames: List[str],
    topk: int = 5,
    figsize_per_row: float = 3.5
):
    M, C = probs.shape
    fig, axes = plt.subplots(M, 2, figsize=(10, figsize_per_row * M))
    if M == 1:
        axes = np.expand_dims(axes, 0)
    for i in range(M):
        # left: image
        ax_img = axes[i, 0]
        ax_img.axis("off")
        try:
            ax_img.imshow(Image.open(paths[i]).convert("RGB"))
        except Exception as e:
            ax_img.text(0.5, 0.5, f"load error\n{e}", ha="center", va="center")

        # right: top-k bar chart
        ax_bar = axes[i, 1]
        p = probs[i]
        order = np.argsort(-p)[:min(topk, C)]
        labels = [classnames[j] for j in order]
        vals = p[order]
        ax_bar.barh(range(len(order)), vals)
        ax_bar.set_yticks(range(len(order)))
        ax_bar.set_yticklabels(labels)
        ax_bar.invert_yaxis()
        ax_bar.set_xlim(0, 1)
        ax_bar.set_xlabel("probability")
    plt.tight_layout()
    plt.show()
```

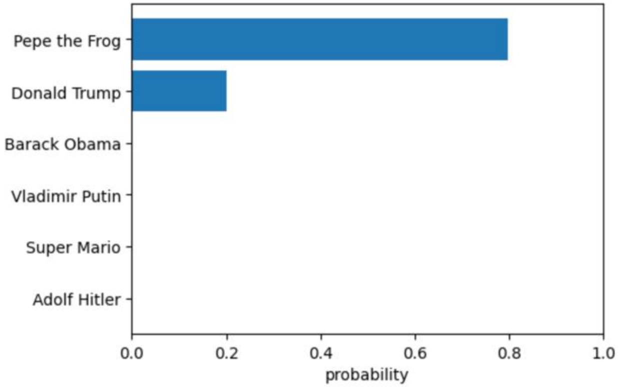
Helper method to visualize results

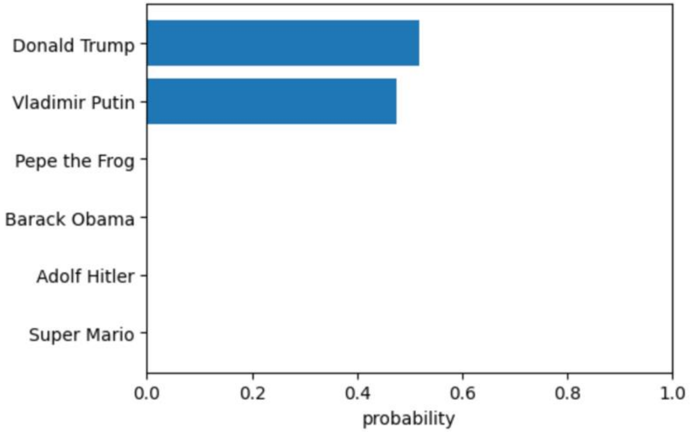
```python
# 1) Choose classes
people = [
    "Donald Trump", "Vladimir Putin", "Pepe the Frog", "Barack Obama", "Super Mario", "Adolf Hitler"
]

classnames, text_feats = build_people_text_features(people, model, tokenizer)

# 2) Get probabilities for the 8 selected images (use the precomputed embeddings)
img_emb_subset = img_emb[idx]      # shape [8, D], already L2-normalized
probs = probs_for_subset(img_emb_subset, text_feats, model)   # [8, 8]

# 3) Visualize: image + top-k class bars
plot_images_with_probs(paths, probs, classnames, topk=20)
```

Use Case - Zero-Shot Classification Results









Use Case – Relationships between CLIP Embeddings

Main method to compose query and identify top-n

```python
@torch.no_grad()
def topn_composed_query(
    base_emb: np.ndarray,
    add_text: str = None,
    sub_text: str = None,
    add_img_emb: np.ndarray = None,
    sub_img_emb: np.ndarray = None,
    weight_base: float = 1.0,
    weight_add_text: float = 1.0,
    weight_sub_text: float = 1.0,
    weight_add_img: float = 1.0,
    weight_sub_img: float = 1.0,
    img_emb: np.ndarray = None,
    image_paths=None,
    model=None, tokenizer=None,
    topn: int = 4
):
    device = next(model.parameters()).device

    # start from base embedding
    q = torch.from_numpy(_ensure_2d(base_emb)).to(device) * weight_base

    # add/sub text
    if add_text:
        t = torch.from_numpy(text_emb_for(add_text, model, tokenizer, device))
        q += weight_add_text * t
    if sub_text:
        t = torch.from_numpy(text_emb_for(sub_text, model, tokenizer, device))
        q -= weight_sub_text * t

    # add/sub image
    if add_img_emb is not None:
        q += weight_add_img * torch.from_numpy(_ensure_2d(add_img_emb)).to(device)
    if sub_img_emb is not None:
        q -= weight_sub_img * torch.from_numpy(_ensure_2d(sub_img_emb)).to(device)

    # normalize query
    q = q / (q.norm(dim=-1, keepdim=True) + 1e-9)
    q_np = q.detach().cpu().float().numpy()

    # cosine with gallery
    img = img_emb.astype(np.float32)
    img = img / (np.linalg.norm(img, axis=1, keepdims=True) + 1e-9)
    scores = (img @ q_np.T).squeeze(1)

    idx = np.argsort(-scores)[:topn]
    return idx, [image_paths[i] for i in idx], scores[idx]
```

```python
@torch.no_grad()
def text_emb_for(prompt: str, model, tokenizer, device=None, normalize=True) -> np.ndarray:
    """Return a [1, D] numpy embedding for a single text prompt."""
    if device is None:
        device = next(model.parameters()).device
    toks = tokenizer([prompt]).to(device)
    t = model.encode_text(toks)
    if normalize:
        t = t / (t.norm(dim=-1, keepdim=True) + 1e-9)
    return t.detach().cpu().float().numpy()  # [1, D]

@torch.no_grad()
def image_emb_for(path, model, preprocess, device=None, normalize=True) -> np.ndarray:
    """Return a [1, D] numpy embedding for a single image path."""
    if device is None:
        device = next(model.parameters()).device
    x = preprocess(Image.open(path).convert("RGB")).unsqueeze(0).to(device)  # [1, 3, H, W]
    e = model.encode_image(x)
    if normalize:
        e = e / (e.norm(dim=-1, keepdim=True) + 1e-9)
    return e.detach().cpu().float().numpy()  # [1, D]


def _ensure_2d(a: np.ndarray) -> np.ndarray:
    """Make sure array is [1, D] (accepts [D] or [1, D])."""
    a = np.asarray(a, dtype=np.float32)
    if a.ndim == 1:
        a = a[None, :]
    return a
```

Helper functions (extracting embeddings + normalizing arrays)

Relationships in CLIP Embeddings

Text + Text →
"An image of Donald Trump"
+ "Pepe the Frog"

Image + Text →
Donald Trump Image +
"Vladimir Putin"

Weighted Image(0.2) + Text(0.8) →
Donald Trump Image + "Vladimir
Putin"

Next Steps – Hands on Exercises

1. Prompt engineering for retrieval

- Pick 5–10 simple queries (e.g., "a crowd", "a protest sign", "a cartoon character", "a politician at a podium"). For each, retrieve top-k images with topn_for_text, then slightly vary the prompt (add/remove adjectives like "close-up", "outdoor", "old photo") and and compare the grids.

2. Build a tiny "hateful vs. benign" zero-shot probe

- MMHS150K has a labels_str column; construct a tiny balanced subset (e.g., 100 "likely hateful" vs 100 "likely benign" based on labels_str). Create two prompts like "an image that conveys hateful content" vs "an image that does not convey hateful content" (or a small template set). Compute the softmax probabilities and report a simple AUC or accuracy using a threshold on the hateful prompt's probability.

3. "Targeted retrieval" with vector composition

- Pick a base image (image_emb_for) and steer it with text, e.g., base = an image with Donald Trump + add_text = "wearing sunglasses". Compare results across weights (weight_base, weight_add_text ∈ {0.2,0.5,0.8}).

TUDelft

Thank you for your attention

Savvas Zannettou