

**ĐẠI HỌC QUỐC GIA HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỒ ÁN
TRUY VẤN THÔNG TIN ĐA PHƯƠNG TIỆN
CS336.P12**

**ĐỀ TÀI
IMAGE RETRIEVAL FROM TEXT**

Giảng viên hướng dẫn: Đỗ Văn Tiến

Thành viên nhóm:

Nguyễn Vẹn Toàn	22521492
Đào Văn Tuấn	22521599
Vũ Anh Tuấn	22521614

Thành Phố Hồ Chí Minh, 26 Tháng 1 Năm 2025

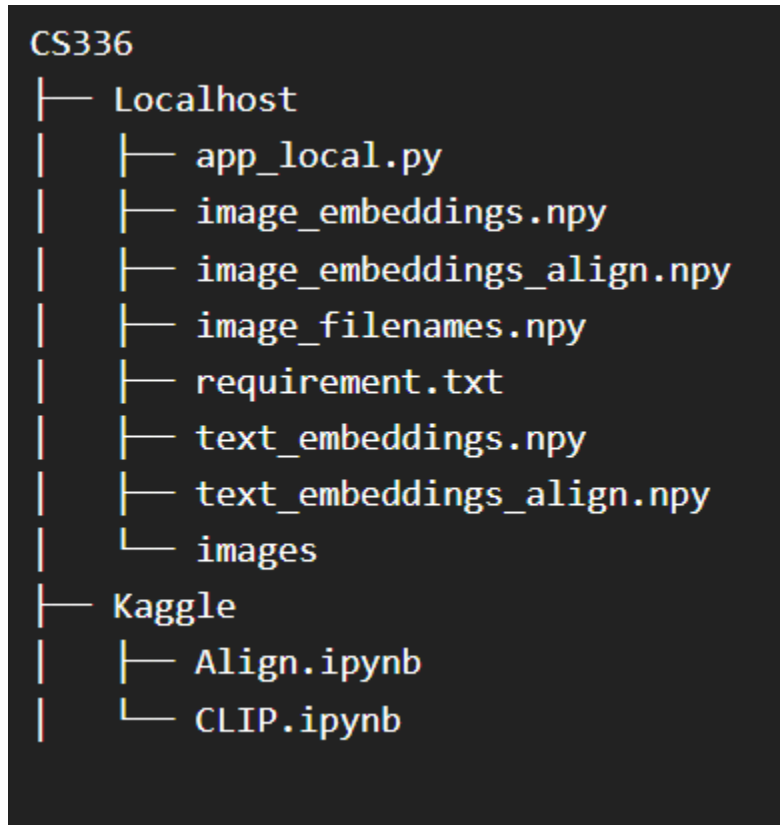
Mục lục

0. Link github và hướng dẫn sử dụng.....	2
1. Giới thiệu	3
2. Dataset	4
3. Model.....	4
3.1. CLIP	4
3.2. ALIGN	5
4. Quy Trình	6
Bước 1: Chuẩn bị dữ liệu	6
Bước 2: Tải mô hình Pretrained.....	6
Bước 3: Tạo Embedding/Tải Embedding	7
Bước 4: Tính độ tương đồng, trả về kết quả	9
5. Đánh giá.....	10
6. Tài liệu tham khảo	12

0. Link github và hướng dẫn sử dụng

Link github: <https://github.com/NguyenVenToan/CS336>

Cấu trúc cây thư mục đầy đủ:



Do giới hạn upload file của github là 25MB mà các file embedding và thư mục images trong thư mục Localhost lại có dung lượng lớn hơn nên nhóm em đã upload 5 file trừ file .py và file .txt và thư mục images chứa ảnh lên Hugging Face.



Link Hugging Face:

- + Với Embeddings: <https://huggingface.co/datasets/Lippovn04/Embeddings>
- + Với ảnh: <https://huggingface.co/datasets/Lippovn04/images>

Cách sử dụng:

Với các file thư mục Kaggle:

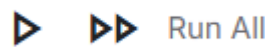
- **B1:** Đăng nhập tài khoản Kaggle.
- **B2:** Tạo notebook mới rồi import file .ipynb muốn sử dụng.
- **B3:** Kiểm tra dataset có đủ không. Phải đủ hình sau:

- ▶  embedding-clip-vitbert-align
- ▶  flickr30k-dataset

Nếu chưa đủ thì chọn “Add Input”, chọn “Dataset”, sau đó nhập từ khóa:

- + Với ảnh thì nhập từ khóa “Flickr30k-dataset” (được upload bởi Toàn Nguyễn) sau đó nhấn dấu + để thêm input.
- + Với embedding thì nhập từ khóa “Embedding_clip_vitbert_align” (được upload bởi Toàn Nguyễn) sau đó nhấn dấu + để thêm input.

- **B4:** Nhấn để chạy code



Với Localhost:

- **B1:** Tải file app_local.py từ github về, tải thư mục ảnh và embedding từ Hugging Face về và giải nén sau đó sắp xếp như cây thư mục bên trên.
- **B2:** Tải các thư viện cần thiết có trong file requirements.txt.
- **B3:** Mở folder chứa code và dữ liệu cần thiết trên Visual Studio Code(File -> Open Folder. Ở đây như cây thư mục trên thì chọn Localhost) sau đó chọn “Terminal” -> “New Terminal” sau đó nhập vào terminal “streamlit run app_local.py”.

1. Giới thiệu

Trong thời đại số hóa, lượng thông tin hình ảnh ngày càng gia tăng, đòi hỏi các phương pháp hiệu quả để tìm kiếm và quản lý dữ liệu. Image Retrieval from Text (Truy xuất hình ảnh từ văn bản) là một lĩnh vực nghiên cứu quan trọng trong lĩnh vực truy vấn thông tin và thị giác máy tính, cho phép người dùng tìm kiếm hình ảnh thông qua các mô tả bằng ngôn ngữ tự nhiên.

Trong đề tài này, nhóm giới thiệu phương pháp tích hợp các mô hình học sâu tiên tiến như CLIP (Contrastive Language–Image Pretraining) và ALIGN (A Large-scale Image and Noisy-text embedding) để giải quyết bài toán truy vấn hình ảnh từ văn bản. Bằng cách khai thác sức mạnh của các mô hình trên, nhóm kỳ vọng đạt được những kết quả nổi bật, đồng thời cung cấp một cái nhìn toàn diện về khả năng ứng dụng của các mô hình đa phương thức trong thực tế.

2. Dataset

Đối với đề tài này, nhóm sử dụng bộ dữ liệu Flickr30k cho quá trình huấn luyện và kiểm tra mô hình. Flickr30k là bộ dữ liệu nổi tiếng trong lĩnh vực trí tuệ nhân tạo, đặc biệt được sử dụng trong các bài toán Image Captioning (mô tả hình ảnh bằng văn bản) và Image-Text Retrieval (truy xuất hình ảnh từ văn bản và ngược lại). Đây là phiên bản mở rộng của tập dữ liệu Flickr8k, được thiết kế để thúc đẩy nghiên cứu trong việc liên kết ngữ nghĩa giữa hình ảnh và ngôn ngữ tự nhiên.

Đặc điểm chính:

- **Quy mô:** Gồm 31,783 hình ảnh từ nền tảng Flickr, với các ngữ cảnh đời sống thực tế đa dạng. Mỗi hình ảnh đi kèm 5 mô tả văn bản (caption) được viết bởi con người.
- **Nội dung:** Hình ảnh phản ánh nhiều tình huống khác nhau: con người, động vật, vật thể, hoạt động,... và các mô tả văn bản cung cấp thông tin chi tiết về ngữ cảnh, đối tượng và hành động tương ứng với mỗi hình ảnh.
- **Phân chia dữ liệu:** Training set: ~25,000 hình ảnh, validation set: ~6,000 hình ảnh.

3. Model

3.1. CLIP

CLIP (Contrastive Language–Image Pretraining) là một mô hình học sâu tiên tiến được thiết kế để học các mối quan hệ giữa văn bản và hình ảnh từ bộ dữ liệu quy mô lớn. Mục tiêu của CLIP là xây dựng một mô hình có khả năng thực hiện tốt trên nhiều tác vụ mà không cần huấn luyện lại hoặc tinh chỉnh trên từng bài toán cụ thể (zero-shot learning).

CLIP được huấn luyện trên hàng trăm triệu cặp văn bản-hình ảnh thu thập từ internet, sử dụng phương pháp học tương phản (contrastive learning). Kiến trúc của CLIP bao gồm hai thành phần chính:

- **Image Encoder:** Một mạng Vision Transformer (ViT) hoặc ResNet được sử dụng để mã hóa đặc trưng hình ảnh.
- **Text Encoder:** Một mạng Transformer mã hóa đặc trưng văn bản (BERT, GPT,...)

Hai bộ mã hóa này ánh xạ hình ảnh và văn bản vào cùng một không gian nhúng, cho phép tính toán điểm tương đồng giữa chúng. Mô hình được tối ưu hóa để tăng điểm tương đồng cho các cặp văn bản-hình ảnh đúng và giảm điểm tương đồng cho các cặp sai.

CLIP đạt hiệu suất vượt trội trên nhiều tác vụ như phân loại hình ảnh, tìm kiếm hình ảnh theo văn bản, và tìm kiếm văn bản theo hình ảnh mà không yêu cầu thêm dữ liệu nhãn.

Phiên bản nhóm sử dụng là clip-vit-base-patch32 đã được huấn luyện trên gần 400 triệu cặp dữ liệu được thu thập từ internet.

3.2. ALIGN

ALIGN (A Large-scale Image and Noisy-text embedding) được phát triển bởi Google AI nhằm học biểu diễn đa phương thức từ văn bản và hình ảnh, tương tự như CLIP. ALIGN tận dụng bộ dữ liệu huấn luyện khổng lồ, bao gồm các cặp văn bản-hình ảnh được thu thập từ internet. Dù dữ liệu chứa nhiều (không hoàn toàn sạch), kích thước lớn của tập dữ liệu mang lại hiệu suất đáng kể trong các bài toán đa phương thức.

Kiến trúc ALIGN bao gồm:

- **Image Encoder:** Một mạng EfficientNet để mã hóa đặc trưng hình ảnh.
- **Text Encoder:** Một mạng Transformer để mã hóa đặc trưng văn bản (BERT, GPT,...).

ALIGN sử dụng phương pháp học tương phản để tối ưu hóa không gian nhúng chung, nơi văn bản và hình ảnh liên quan được ánh xạ gần nhau.

Mô hình này được thiết kế để hỗ trợ các tác vụ như tìm kiếm đa phương thức và phân loại không nhãn (zero-shot classification). Với khả năng xử lý dữ liệu lớn, ALIGN mang lại hiệu quả cao trong việc khái quát hóa các biểu diễn đa phương thức.

Phiên bản nhóm sử dụng là align-base đã được huấn luyện với gần 1.8 tỷ cặp dữ liệu ảnh và văn bản, trong đó văn bản là dữ liệu “noisy” (văn bản có thể không mô tả trực tiếp nội dung của hình ảnh hoặc có thông tin không đúng).

4. Quy Trình

Bước 1: Chuẩn bị dữ liệu

Sử dụng dữ liệu đã được nhóm tìm thấy trên Hugging Face sau đó được nhóm tải lên kaggle (để chế độ công khai) để dễ dàng sử dụng. Tên các bộ dữ liệu đã được đề cập ở phần 0.

```
# Path to the dataset
csv_path = '/kaggle/input/flickr30k-dataset/flickr_annotations_30k.csv'
images_dir = '/kaggle/input/flickr30k-dataset/flickr30k-images/flickr30k-images'
```

Sử dụng tệp CSV chứa các chú thích (annotations) và đường dẫn ảnh liên quan. Làm sạch dữ liệu bằng cách loại bỏ các dòng trống.

```
# Load the CSV file
df = pd.read_csv(csv_path)
df = df.dropna() # Drop any empty rows if they exist
print("Dataset shape:", df.shape)
df.head()
```

Bước 2: Tải mô hình Pretrained

Đối với mô hình CLIP:

- Tải mô hình từ thư viện HuggingFace với model pretrained: openai/clip-vit-large-patch14.
- Tải processor tương ứng để chuẩn bị đầu vào cho mô hình.

```
# Load CLIP model and processor
model = CLIPModel.from_pretrained("openai/clip-vit-large-patch14").to(device)
processor = CLIPProcessor.from_pretrained("openai/clip-vit-large-patch14")
```

Đối với mô hình ALIGN:

- Tải mô hình từ thư viện HuggingFace với model pretrained: kakaobrain/align-base.
- Tải processor tương ứng để chuẩn bị đầu vào cho mô hình.

```
# Load ALIGN for image-text embeddings
align_model = AutoModel.from_pretrained("kakaobrain/align-base").to(device)
align_processor = AutoProcessor.from_pretrained("kakaobrain/align-base")
```

Bước 3: Tạo Embedding/Tải Embedding

1. Ảnh:

Chuẩn bị và xử lý ảnh:

- Chuyển đổi định dạng ảnh sang RGB.
- Áp dụng các phép biến đổi (resize, center crop, chuẩn hóa) thông qua bộ xử lý của từng mô hình (CLIP hoặc ALIGN).

Tính embedding:

- Với CLIP: Sử dụng hàm `model.get_image_features()`.
- Với ALIGN: Sử dụng hàm `align_model.get_image_features()`.

Chuẩn hóa vector nhúng (L2-norm):

```
def get_image_embeddings(image_dir, filenames, model, processor):
    embeddings = []
    for filename in tqdm(filenames, desc="Processing Images"):
        img_path = os.path.join(image_dir, filename)
        image = Image.open(img_path).convert("RGB")
        inputs = processor(images=image, return_tensors="pt").to(device)
```

CLIP:

```
img_emb = model.get_image_features(**inputs)
img_emb = img_emb / img_emb.norm(p=2, dim=-1, keepdim=True) # Normalize embeddings
embeddings.append(img_emb.cpu().numpy())
return np.vstack(embeddings)
```

ALIGN:

```
with torch.no_grad():
    outputs = model.get_image_features(**inputs) # Extract image features
    img_emb = outputs / outputs.norm(p=2, dim=-1, keepdim=True) # Normalize
    embeddings.append(img_emb.cpu().numpy())
return np.vstack(embeddings)
```

2. Văn bản:

Chuẩn bị và xử lý văn bản:

- Token hóa và chuẩn hóa thông qua processor của CLIP và ALIGN.

Tính embedding:

- Với CLIP: Sử dụng hàm `model.get_text_features()`.
- Với ALIGN: Sử dụng hàm `align_model.get_text_features()`.

Chuẩn hóa vector nhúng (L2-norm).

CLIP:

```
# Text preprocessing
def get_text_embeddings(captions, model, processor):
    embeddings = []
    for caption in tqdm(captions, desc="Processing Captions"):
        inputs = processor(text=caption, return_tensors="pt", truncation=True).to(device)
        with torch.no_grad():
            text_emb = model.get_text_features(**inputs)
            text_emb = text_emb / text_emb.norm(p=2, dim=-1, keepdim=True) # Normalize embeddings
            embeddings.append(text_emb.cpu().numpy())
    return np.vstack(embeddings)
```

ALIGN:

```
def get_text_embeddings(captions, model, processor):
    embeddings = []
    for caption in tqdm(captions, desc="Processing Captions"):
        inputs = processor(text=caption, return_tensors="pt", truncation=True, padding=True).to(device)
        with torch.no_grad():
            outputs = model.get_text_features(**inputs) # Extract text features
            text_emb = outputs / outputs.norm(p=2, dim=-1, keepdim=True) # Normalize
            embeddings.append(text_emb.cpu().numpy())
    return np.vstack(embeddings)
```

Từ tập dữ liệu, lấy danh sách tên tệp ảnh (`image_filenames`) và mô tả văn bản (`captions`).

```
# Get image and text embeddings
image_filenames = df['filename'].values
captions = df['raw'].values
```

Sau đó tạo embedding với các hàm đã tạo phía trên cho từng mô hình.

```
image_embeddings = get_image_embeddings(images_dir, image_filenames, model, processor)
text_embeddings = get_text_embeddings(captions, model, processor)

image_embeddings = get_image_embeddings(images_dir, image_filenames, align_model, align_processor)
text_embeddings = get_text_embeddings(captions, align_model, align_processor)
```

Tiếp theo, chúng ta có thể lưu lại các embedding này để tiết kiệm thời gian, tái sử dụng để tính toán độ tương đồng và đánh giá mô hình.

```
# Lưu image_embeddings và text_embeddings cho mô hình CLIP
np.save("image_embeddings.npy", image_embeddings)
np.save("text_embeddings.npy", text_embeddings)
print("Embeddings đã được lưu thành công!")
```

```
# Lưu image_embeddings và text_embeddings cho mô hình ALIGN
np.save("image_embeddings_align.npy", image_embeddings)
np.save("text_embeddings_align.npy", text_embeddings)
print("Embeddings đã được lưu thành công!")
```

Khi cần sử dụng ta tải các embedding đã lưu và đã được nhóm tải lên công khai trên kaggle.

```
# Tải lại image_embeddings và text_embeddings cho mô hình CLIP
image_embeddings = np.load("/kaggle/input/embedding-clip/image_embeddings.npy")
text_embeddings = np.load("/kaggle/input/embedding-clip/text_embeddings.npy")
print("Embeddings đã được tải lại thành công!")
```

```
# Tải lại image_embeddings và text_embeddings cho mô hình ALIGN
image_embeddings = np.load("/kaggle/input/embedding-clip-vitbert-align/image_embeddings_align.npy")
text_embeddings = np.load("/kaggle/input/embedding-clip-vitbert-align/text_embeddings_align.npy")
print("Embeddings đã được tải lại thành công!")
```

Bước 4: Tính độ tương đồng, trả về kết quả

Tiền xử lý truy vấn văn bản:

- Văn bản truy vấn được đưa qua processor của mô hình để chuyển đổi thành định dạng tensor phù hợp:

```
# Preprocess query text
inputs = processor(text=[query], return_tensors="pt", truncation=True).to(device)
```

- Dữ liệu đầu vào được chuyển đến thiết bị xử lý (GPU hoặc CPU).

Tính toán vector nhúng của truy vấn:

- Vector nhúng cho truy vấn được tính toán thông qua hàm `get_text_features()`:

```
query_emb = model.get_text_features(*inputs)
```

- Vector này được chuẩn hóa bằng L2-norm để đảm bảo giá trị đồng nhất:

```
query_emb = query_emb / query_emb.norm(p=2, dim=-1, keepdim=True)
```

Tính độ tương đồng cosine:

- Sử dụng hàm `cosine_similarity` từ thư viện `scikit-learn` để tính toán độ tương đồng giữa vector nhúng của truy vấn và các vector nhúng của ảnh:

```
similarities = cosine_similarity(query_emb.cpu().numpy(), image_embeddings)
```

- Kết quả là một mảng giá trị, trong đó mỗi giá trị biểu diễn độ tương đồng giữa truy vấn và một ảnh trong tập dữ liệu.

Xác định các ảnh phù hợp nhất:

- Chọn ra top-k chỉ số của các ảnh có độ tương đồng cao nhất:

```
top_indices = similarities[0].argsort()[-top_k:][::-1] # Top-k indices
```

Trả về và hiển thị kết quả:

- Lặp qua danh sách các chỉ số vừa tìm được và hiển thị các ảnh tương ứng:

```
# Display top-k retrieved images
print(f"Top {top_k} retrieved images for query: '{query}'")
for idx in top_indices:
    img_path = os.path.join(images_dir, image_filenames[idx])
    display(Image.open(img_path))
```

Kết quả trả về:

- Với mỗi truy vấn văn bản (query), hàm trả về danh sách top-k ảnh phù hợp nhất, được sắp xếp theo mức độ tương đồng giảm dần.
- Các ảnh được hiển thị trực tiếp để người dùng dễ dàng đánh giá độ phù hợp.

5. Đánh giá

Để thuận tiện cho việc đánh giá, nhóm sử dụng độ đo Top-k Accuracy và Recall@k, trong đó k là tổng số truy vấn thực hiện.

- **Top-k accuracy:** là tỷ lệ phần trăm các mẫu mà trong đó nhãn đúng xuất hiện trong k dự đoán hàng đầu (top-k) của mô hình. Độ đo này tập trung vào khả năng xếp hạng chính xác.

$$\text{Top-k Accuracy} = \frac{\text{Số mẫu mà nhãn đúng xuất hiện trong top-k dự đoán}}{\text{Tổng số mẫu}}$$

- **Recall@k:** đo lường tỷ lệ mẫu mà trong đó nhãn đúng được tìm thấy trong top-k kết quả trả về.

$$\text{Recall@k} = \frac{\text{Số mẫu tìm được nhãn đúng trong top-k kết quả}}{\text{Tổng số mẫu thực sự có nhãn đúng}}$$

Kết quả sau thực nghiệm:

Độ đo / Model	CLIP	ALIGN
Top-1 Accuracy	0.6008	0.7746
Top-5 Accuracy	0.8410	0.9357
Recall@10	0.9096	0.9678

Nhận xét: Hiệu suất đạt được của mô hình ALIGN tốt hơn mô hình CLIP ở tất cả các độ đo sử dụng nhưng không phải là quá nhiều: hơn 17.28% ở top-1 accuracy, hơn 9.47% ở top-5 accuracy và hơn 5.82% ở recall@10. Mô hình ALIGN cho ra kết quả tốt hơn CLIP vì ALIGN đã được huấn luyện trên bộ dữ liệu lớn hơn rất nhiều so với CLIP (~1.8 tỷ > ~400 triệu), thêm vào đó, ALIGN sử dụng dữ liệu “noisy text” để huấn luyện giúp mô hình tổng quát tốt hơn và chính xác cho các truy vấn trong thực tế.

Tổng kết: Bằng cách khai thác sức mạnh của các mô hình đa phương thức tiên tiến ở trên và áp dụng vào bài toán thực tế (truy xuất ảnh bằng văn bản), nhóm đã đạt được những kết quả tương đối tốt và đáp ứng được các mục tiêu đã đề ra.

6. Tài liệu tham khảo

https://huggingface.co/docs/transformers/model_doc/clip
https://huggingface.co/docs/transformers/model_doc/align
<https://arxiv.org/abs/2102.05918v2>
<https://arxiv.org/abs/2103.00020>