

TRƯỜNG ĐẠI HỌC NHA TRANG
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN TỐT NGHIỆP

ĐỀ TÀI:

**TÌM KIẾM ẢNH DỰA TRÊN ĐỒ THỊ TRI THỨC VÀ ĐỐI TƯỢNG
LÁNG GIỀNG**

Giáo viên hướng dẫn	:	Phạm Thị Thu Thúy
Sinh viên thực hiện	:	Võ Đại Hiệp
Lớp	:	63.CNTT-CLC
Mã số sinh viên	:	63132946

KHÁNH HÒA – NĂM 2025

TRƯỜNG ĐẠI HỌC NHA TRANG
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN TỐT NGHIỆP

ĐỀ TÀI:

**TÌM KIẾM ẢNH DỰA TRÊN ĐỒ THỊ TRI THỨC VÀ ĐỐI TƯỢNG
LÁNG GIỀNG**

Giáo viên hướng dẫn	:	Phạm Thị Thu Thúy
Sinh viên thực hiện	:	Võ Đại Hiệp
Lớp	:	63.CNTT-CLC
Mã số sinh viên	:	63132946

KHÁNH HÒA – THÁNG 6/2025

LỜI CAM ĐOAN

Em xin cam đoan rằng mọi kết quả trong đề tài "Tìm kiếm ảnh dựa trên đồ thị tri thức và đối tượng láng giềng" là sản phẩm của em và được thực hiện dựa trên nghiên cứu và phân tích cẩn thận, sử dụng các tài liệu công khai, đáng tin cậy và dưới sự hướng dẫn nhiệt tình của Cô Phạm Thị Thu Thúy – Giáo viên hướng dẫn đã hỗ trợ trong suốt quá trình thực hiện đề tài. Đồng thời, em cam đoan mọi thông tin và kết quả đạt được là trung thực và minh bạch hoàn toàn, dưới sự nỗ lực không ngừng của em để có kết quả hoàn thiện nhất.

Nha Trang, ngày tháng năm 2025

Sinh viên cam đoan

Võ Đại Hiệp

LỜI CẢM ƠN

Trước tiên, em xin bày tỏ lòng biết ơn sâu sắc tới Trường Đại học Nha Trang – ngôi trường đã mang đến cho em một môi trường học tập năng động, hiện đại, và giàu tính thực tiễn. Trong suốt quá trình học tập, em đã có cơ hội tiếp cận với kiến thức vững chắc, cùng hệ thống cơ sở vật chất đầy đủ và điều kiện học tập thuận lợi.

Em xin trân trọng cảm ơn quý Thầy Cô trong Khoa Công nghệ Thông tin – những người đã tận tâm giảng dạy, truyền đạt kiến thức, kỹ năng và tinh thần học thuật cho em và các bạn sinh viên. Đặc biệt, em xin gửi lời cảm ơn chân thành đến cô Phạm Thị Thu Thủy, người đã trực tiếp hướng dẫn và đồng hành cùng em trong suốt quá trình thực hiện đề tài tốt nghiệp. Sự chỉ dẫn tận tình, những góp ý quý báu của cô đã giúp em hoàn thiện đề tài một cách bài bản và có định hướng rõ ràng hơn.

Bên cạnh đó, em cũng xin gửi lời cảm ơn sâu sắc đến gia đình và bạn bè – những người luôn bên cạnh ủng hộ, động viên và tiếp thêm động lực cho em trong hành trình học tập và nghiên cứu.

Do còn hạn chế về kiến thức chuyên môn cũng như kinh nghiệm thực tiễn, bản thân em không tránh khỏi những thiếu sót trong quá trình thực hiện đề tài. Em rất mong nhận được những ý kiến đóng góp quý báu từ quý Thầy Cô để hoàn thiện đề tài tốt hơn và rút ra nhiều bài học kinh nghiệm cho bản thân trong tương lai.

Một lần nữa, em xin kính chúc quý Thầy Cô dồi dào sức khỏe, luôn hạnh phúc và thành công trong sự nghiệp trồng người cao quý!

Em xin chân thành cảm ơn!

MỤC LỤC

LỜI CAM ĐOAN	i
LỜI CẢM ƠN.....	ii
MỤC LỤC	iii
DANH MỤC BẢNG	vi
DANH MỤC HÌNH ẢNH.....	vii
DANH MỤC TỪ VIẾT TẮT	viii
PHẦN MỞ ĐẦU	1
1. Lý do chọn đề tài	1
2. Mục tiêu đề tài	1
3. Phương pháp nghiên cứu	2
4. Bố cục trình bày của đồ án	3
CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI VÀ CƠ SỞ LÝ THUYẾT	4
1.1. Tổng quan về hệ thống tìm kiếm ảnh dựa trên đồ thị tri thức	4
1.1.1. Giới thiệu về bài toán tìm kiếm ảnh có tri thức.....	4
1.1.2. Giới thiệu về hệ thống tìm kiếm ảnh dựa trên đồ thị tri thức.....	4
1.1.3. Các chức năng của hệ thống tìm kiếm ảnh dựa trên đồ thị tri thức.....	6
1.1.4. Đánh giá hiện trạng	7
1.1.5. Mô tả bài toán.....	8
1.2. Cơ sở lý thuyết.....	9
1.2.1. Đồ thị tri thức và biểu diễn triplet RDF	9
1.2.2. Mạng nơ-ron đồ thị (GCN, GAT, R-GCN).....	12
1.2.3. Mô hình biểu diễn tri thức (TransE, TransH).....	15
1.2.4. Các phương pháp tìm kiếm ảnh truyền thống và hiện đại.....	17
1.2.5. Tích hợp NLP trong truy vấn ảnh (spaCy, OpenIE, BERT)	19
CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG	22
2.1. Kiến trúc tổng thể	22
2.2. Quy trình xử lý dữ liệu COCO	24
2.3. Thiết kế đồ thị tri thức từ caption và Detectron2	26
2.4. Thiết kế giao diện và API FastAPI + Streamlit.....	27

2.5. Lưu trữ KG với Neo4j	28
CHƯƠNG 3: XÂY DỰNG THUẬT TOÁN VÀ TRIỂN KHAI HỆ THỐNG	33
3.1. Mô hình TransE	33
3.1.1. Thư viện sử dụng trong cài đặt mô hình TransE	33
3.1.2. Đọc và tiền xử lý dữ liệu cho mô hình TransE	34
3.1.3. Xây dựng lớp mô hình TransE	35
3.1.4. Sinh dữ liệu âm (Negative Sampling)	37
3.1.5. Huấn luyện mô hình TransE.....	38
3.2. Mô hình GCN	40
3.2.1. Thư viện sử dụng trong cài đặt mô hình GCN	40
3.2.2. Đọc và tiền xử lý dữ liệu cho mô hình GCN.....	41
3.2.3. Chuyển đổi và chuẩn bị dữ liệu đầu vào cho mô hình GCN bằng DGL.....	42
3.2.4. Mô hình Graph Convolutional Network (GCN)	43
3.2.5. Tổng hợp vector embedding cho ảnh	44
3.2.6. Truy vấn ảnh tương tự bằng khoảng cách cosine.....	45
3.3. Mô hình R-GCN	46
3.3.1. Thư viện sử dụng trong cài đặt mô hình R-GCN	46
3.3.2. Đọc và tiền xử lý dữ liệu cho mô hình R-GCN.....	47
3.3.3. Xây dựng đồ thị DGL cho mô hình R-GCN	48
3.3.4. Xây dựng mô hình Relational Graph Convolutional Network (R-GCN) ...	49
3.3.5. Truy vấn ảnh tương tự bằng vector embedding	51
3.3.6. Truy xuất các ảnh liên quan và đánh giá	52
3.4. Mô hình GAT	54
3.4.1. Thư viện sử dụng trong cài đặt mô hình GAT	54
3.4.2. Đọc và tiền xử lý dữ liệu cho mô hình GAT.....	55
3.4.3. Xây dựng mô hình Graph Attention Network (GAT).....	56
3.4.4. Huấn luyện mô hình GAT	58
3.4.5. Trích xuất vector đặc trưng và tạo bản đồ ánh xạ từ entity đến ảnh	60
3.4.6. Lưu mô hình và dữ liệu sau huấn luyện	61
3.4.7. Truy xuất hình ảnh có nội dung tương đồng	62
CHƯƠNG 4: ĐÁNH GIÁ KẾT QUẢ.....	64
4.1. Đánh giá các mô hình (TransE, GCN, R-GCN, GAT).....	64

4.2. So sánh độ chính xác: Precision / Recall / F1-score.....	64
4.3. Truy vấn minh họa và ảnh kết quả	66
4.4. Phân tích ưu nhược điểm mô hình.....	70
4.4.1. Mô hình TransE.....	70
4.4.2. Mô hình GCN.....	71
4.4.3. Mô hình GAT	71
4.4.4. Mô hình R-GCN	72
CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	73
5.1. Kết quả đạt được.....	73
5.2. Hạn chế hệ thống hiện tại	73
5.3. Định hướng phát triển tương lai	74
TÀI LIỆU THAM KHẢO	75

DANH MỤC BẢNG

Bảng 1.1 So sánh hai phương pháp	19
Bảng 4.1 So sánh kết quả đánh giá giữa các mô hình	65

DANH MỤC HÌNH ẢNH

Hình 2.1 Sơ đồ tầng xử lý.....	23
Hình 2.2 Sơ đồ tầng truy xuất.....	24
Hình 2.3 Tất cả object trích xuất được từ COCO2017.....	27
Hình 2.4 Tất cả relationship trích xuất được từ COCO2017.....	27
Hình 2.5 Run app trên localhost.....	28
Hình 2.6 Giao diện app.....	28
Hình 2.7 Đồ thị tri thức được lưu trong Neo4j.....	29
Hình 2.8 Truy vấn Cypher tìm ảnh có entity là “dog” và “frisbee”	30
Hình 2.9 Hiển thị đồ thị sau truy vấn	31
Hình 2.10 Đồ thị sau khi truy vấn “person” và “motorcycle”.....	32
Hình 3.1 Nơi lưu trữ mô hình saved_models	62
Hình 4.1 Truy vấn “a boy riding a pony” của GAT	66
Hình 4.2 Truy xuất “teddy”, “bear”, “cake” của TransE	67
Hình 4.3 Truy xuất với đầu vào là ảnh bằng mô hình GCN	68
Hình 4.4 Kết quả của truy vấn bằng ảnh	68
Hình 4.5 Truy vấn đầy đủ subject, predicate, object của R-GCN.....	69
Hình 4.6 Truy vấn không đầy đủ chỉ predicate, object của R-GCN	69
Hình 4.7 Truy vấn không đầy đủ chỉ subject, predicate của R-GCN.....	70

DANH MỤC TỪ VIẾT TẮT

Từ viết tắt	Ý nghĩa
CNN	Convolutional Neural Network – Mạng nơ-ron tích chập
TransE	Translating Embedding–Mô hình ánh xạ dịch chuyển trong học biểu diễn tri thức
GCN	Graph Convolutional Network–Mạng nơ-ron tích chập trên đồ thị
GAT	Graph Attention Network–Mạng đồ thị sử dụng cơ chế tập trung
R-GCN	Relational Graph Convolutional Network–Mạng đồ thị tích chập có quan hệ
KG	Knowledge Graph–Đồ thị tri thức
BoVW	Bag of Visual Words–Mô hình túi từ thị giác
SIFT	Scale-Invariant Feature Transform–Phép biến đổi đặc trưng bất biến theo tỉ lệ
COCO	Common Objects in Context–Bộ dữ liệu thị giác máy tính
RDF	Resource Description Framework – Khung mô tả tài nguyên
GNN	Graph Neural Network – Mạng nơ-ron trên đồ thị
VSE	Visual Semantic Embedding – Nhúng ngữ nghĩa thị giác
CLIP	Contrastive Language-Image Pretraining – Mô hình tiền huấn luyện ngôn ngữ–hình ảnh đối sánh
HOG	Histogram of Oriented Gradients – Biểu đồ hướng gradient
SURF	Speeded-Up Robust Features – Đặc trưng mạnh được tăng tốc
URI	Uniform Resource Identifier – Định danh tài nguyên thống nhất
NLP	Natural Language Processing – Xử lý ngôn ngữ tự nhiên
OpenIE	Open Information Extraction – Trích xuất thông tin mở
BERT	Bidirectional Encoder Representations from Transformers – Biểu diễn mã hóa hai chiều từ mô hình Transformer

PHẦN MỞ ĐẦU

1. Lý do chọn đề tài

Trong bối cảnh bùng nổ dữ liệu hình ảnh trên các nền tảng số, việc xây dựng các hệ thống tìm kiếm ảnh hiệu quả và chính xác đang trở thành một nhu cầu cấp thiết trong nhiều lĩnh vực như thương mại điện tử, an ninh, giáo dục và chăm sóc sức khỏe. Tuy nhiên, phần lớn các hệ thống tìm kiếm ảnh hiện nay vẫn chủ yếu dựa trên đặc trưng thị giác thuần túy như màu sắc, hình dạng hoặc mô hình học sâu CNN, trong khi chưa khai thác đầy đủ được mối quan hệ ngữ nghĩa giữa các đối tượng trong ảnh.

Đồ thị tri thức nổi lên như một hướng tiếp cận hiệu quả để mô hình hóa mối quan hệ giữa các thực thể trong ảnh, nhờ khả năng biểu diễn ngữ cảnh và tương tác giữa các đối tượng theo cách có cấu trúc và dễ suy luận. Khi kết hợp đồ thị tri thức với các mô hình học sâu như Graph Convolutional Networks, đặc biệt là các biến thể tiên tiến như Relational Graph Convolutional Network hoặc Graph Attention Network, hệ thống có thể học được biểu diễn ngữ nghĩa sâu sắc và linh hoạt hơn.

Bên cạnh đó, các đặc trưng hình ảnh được trích xuất từ SIFT, Bag of Visual Words hoặc CNN cũng mang lại giá trị biểu diễn trực quan hữu ích. Việc kết hợp hai nguồn thông tin này – tri thức và hình ảnh – trong một không gian embedding thống nhất là một hướng nghiên cứu tiên tiến và đầy tiềm năng.

Xuất phát từ nhận định đó, em lựa chọn đề tài “Tìm kiếm ảnh dựa trên đồ thị tri thức và đối tượng láng giềng” với mong muốn giải quyết bài toán retrieval ảnh không chỉ dựa vào hình ảnh thuần túy, mà còn trên cơ sở lý giải ngữ nghĩa và quan hệ giữa các đối tượng trong ảnh. Đề tài đồng thời kết hợp nhiều hướng công nghệ hiện đại như Triplet Loss, Scene Graph, và truy vấn tri thức bằng Cypher trên Neo4j, hứa hẹn đem lại một hệ thống tìm kiếm ảnh mạnh mẽ, chính xác và có khả năng mở rộng cao.

2. Mục tiêu đề tài

Mục tiêu cốt lõi của đề tài là xây dựng một hệ thống tìm kiếm ảnh từ bộ dữ liệu COCO2017 bằng cách sử dụng mô hình biểu diễn tri thức dưới dạng đồ thị, trong đó mỗi ảnh được ánh xạ thành một tiểu đồ thị phản ánh các đối tượng và mối quan hệ giữa chúng. Khi người dùng đưa ra một truy vấn ngữ nghĩa dưới dạng câu hoặc triplet, hệ thống sẽ chuyển đổi truy vấn thành một đồ thị con tương ứng, sau đó tìm kiếm các ảnh

trong cơ sở dữ liệu có tiêu đề thị tương đồng hoặc gần nhất với tiêu đề truy vấn, dựa trên vector embedding học được từ mạng nơ-ron đồ thị.

Tiền xử lý và chuyển đổi dữ liệu COCO2017 thành các thành phần cấu thành đồ thị tri thức, bao gồm node (đối tượng), edge (mối quan hệ), và thuộc tính. Áp dụng các mô hình học sâu như TransE, GCN, GAT và R-GCN để trích xuất đặc trưng và học embedding biểu diễn ngữ nghĩa cho các đối tượng và mối quan hệ.

Thiết kế cơ sở dữ liệu đồ thị và xây dựng pipeline truy vấn từ ngôn ngữ tự nhiên. Phát triển hệ thống API và giao diện tìm kiếm ảnh tương tác theo thời gian thực. Đánh giá hiệu năng của hệ thống thông qua các chỉ số chuẩn như Precision, Recall, và F1-score với bộ truy vấn chuẩn.

Tóm lại xây dựng một hệ thống tìm kiếm ảnh dựa trên đồ thị tri thức từ dữ liệu COCO2017, trong đó mỗi ảnh được biểu diễn dưới dạng tiêu đề thị phản ánh các đối tượng và mối quan hệ giữa chúng. Thông qua việc áp dụng các mô hình học sâu như TransE, GCN, GAT và R-GCN, hệ thống học được vector biểu diễn ngữ nghĩa cho ảnh, cho phép truy vấn bằng ngôn ngữ tự nhiên hoặc triplet và tìm kiếm các ảnh tương đồng trong không gian embedding. Hệ thống được triển khai với cơ sở dữ liệu đồ thị và API truy vấn tương tác, và đánh giá được hiệu năng của mô hình.

3. Phương pháp nghiên cứu

- Phân tích và tiền xử lý dữ liệu COCO2017: Bộ dữ liệu được khai thác để trích xuất thông tin về đối tượng, vị trí và mối quan hệ giữa các đối tượng trong từng ảnh. Các chú thích annotations được chuẩn hóa và chuyển đổi thành các triplet ngữ nghĩa dạng subject, predicate, object từ đó xây dựng tập triplet toàn cục cho hệ thống.

- Xây dựng đồ thị tri thức hình ảnh: Từ các triplet thu được, mỗi ảnh được ánh xạ thành một tiêu đề thị ngữ nghĩa, với các node đại diện cho đối tượng, cạnh đại diện cho mối quan hệ và thuộc tính như loại đối tượng, chỉ số bounding box,... được gán dưới dạng property.

- Trích xuất đặc trưng và học embedding: Áp dụng Detectron2 để nhận diện đối tượng sinh triplet từ ảnh đầu vào mới và spaCy để trích triplet từ caption. Sử dụng TransE để học embedding cho thực thể và quan hệ trong đồ thị tri thức toàn cục. Sử dụng R-GCN và GAT để lan truyền và học biểu diễn ngữ nghĩa giữa các node và ảnh

thông qua kiến trúc mạng nơ-ron đồ thị. Đối với đặc trưng hình ảnh, kết hợp mô hình trích xuất sau đó kết hợp với embedding từ KG để tạo ra vector biểu diễn thống nhất.

- Thiết kế cơ chế truy vấn và tìm kiếm ảnh: Các truy vấn từ người dùng là triplet hoặc câu ngắn được chuyển đổi thành đồ thị con truy vấn, và ánh xạ vào không gian embedding. Hệ thống so sánh truy vấn với embedding của các ảnh trong tập dữ liệu thông qua khoảng cách cosine và trả về các ảnh có độ tương đồng cao nhất.

- Cài đặt hệ thống và đánh giá: Hệ thống được triển khai bằng Python, sử dụng thư viện PyTorch Geometric cho R-GCN và Neo4j để lưu trữ đồ thị tri thức. Giao diện người dùng được phát triển bằng Streamlit, cho phép thực hiện truy vấn và hiển thị ảnh liên quan theo thời gian thực. Độ chính xác của hệ thống đánh giá bằng các chỉ số Precision, Recall và F1-score trên tập truy vấn chuẩn.

4. Bố cục trình bày của đề án

Nội dung đề tài bao gồm có 5 chương:

- Chương 1: Tổng quan đề tài và cơ sở lý thuyết
- Chương 2: Phân tích và thiết kế hệ thống
- Chương 3: Xây dựng thuật toán và triển khai hệ thống
- Chương 4: Đánh giá kết quả
- Chương 5: Kết luận và hướng phát triển

CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI VÀ CƠ SỞ LÝ THUYẾT

1.1. Tổng quan về hệ thống tìm kiếm ảnh dựa trên đồ thị tri thức

1.1.1. Giới thiệu về bài toán tìm kiếm ảnh có tri thức

Tìm kiếm ảnh là một trong những bài toán quan trọng trong lĩnh vực thị giác máy tính và truy hồi thông tin. Với sự gia tăng nhanh chóng về số lượng và đa dạng của dữ liệu hình ảnh trên Internet, nhu cầu truy xuất ảnh chính xác theo nội dung ngữ nghĩa trở nên ngày càng cấp thiết trong các ứng dụng như thương mại điện tử, giám sát an ninh, y tế, truyền thông và giáo dục.

Tuy nhiên, phần lớn các hệ thống tìm kiếm ảnh truyền thống chỉ dựa vào các đặc trưng thị giác bề mặt như màu sắc, hình dạng hoặc texture, hoặc sử dụng các vector đặc trưng học được từ mô hình CNN. Các hệ thống này thường không thể hiểu sâu ngữ nghĩa của ảnh, đặc biệt là mối quan hệ giữa các đối tượng. Ví dụ, nếu người dùng nhập truy vấn “person riding horse”, một hệ thống truyền thống có thể chỉ tìm được các ảnh chứa cả người và ngựa, mà không đảm bảo đúng mối quan hệ “riding”.

Để khắc phục hạn chế trên, hướng tiếp cận sử dụng đồ thị tri thức đang nhận được sự quan tâm rộng rãi. Trong đó, mỗi ảnh được biểu diễn dưới dạng một đồ thị ngữ nghĩa gồm các node là đối tượng và các cạnh là quan hệ giữa chúng. Mỗi đồ thị ảnh thường được trích xuất từ kết quả phát hiện đối tượng và mô hình scene graph, phản ánh rõ ràng cấu trúc và ngữ cảnh của ảnh.

1.1.2. Giới thiệu về hệ thống tìm kiếm ảnh dựa trên đồ thị tri thức

Trong bối cảnh dữ liệu hình ảnh ngày càng phong phú và đa dạng, đặc biệt với sự phát triển của các nền tảng mạng xã hội, thương mại điện tử và kho lưu trữ dữ liệu trực tuyến, nhu cầu truy vấn ảnh một cách chính xác theo ngữ nghĩa ngày càng trở nên cấp thiết. Tuy nhiên, các hệ thống tìm kiếm ảnh truyền thống chủ yếu dựa trên các đặc trưng thị giác như màu sắc, hình dạng, hoặc vector đặc trưng học được từ mạng nơ-ron tích chập, thường không đủ khả năng biểu diễn các mối quan hệ phức tạp và ngữ cảnh sâu bên trong hình ảnh.

Hệ thống tìm kiếm ảnh dựa trên đồ thị tri thức là một hướng tiếp cận mới và đầy tiềm năng nhằm khắc phục các hạn chế trên. Trong đó, mỗi ảnh được biểu diễn dưới dạng một tiểu đồ thị ngữ nghĩa, bao gồm các thực thể như “person”, “bicycle”, “dog”

dưới dạng các nút và các mối quan hệ như “ride”, “lead”, “stand on” dưới dạng các cạnh nối giữa các thực thể. Mô hình đồ thị tri thức không chỉ nắm bắt sự tồn tại của các đối tượng trong ảnh mà còn biểu diễn rõ cách chúng tương tác với nhau, giúp nâng khả năng hiểu ảnh của hệ thống.

Một truy vấn ảnh trong hệ thống có thể được nhập vào dưới dạng câu ngôn ngữ tự nhiên hoặc triplet ngữ nghĩa (person-riding-horse). Truy vấn này được chuyển thành một đồ thị con tương ứng và được so khớp với các tiểu đồ thị trong tập ảnh thông qua các kỹ thuật học biểu diễn tri thức như TransE, R-GCN hoặc GAT. Quá trình này cho phép hệ thống phát hiện những ảnh có cấu trúc tri thức tương đồng hoặc gần nhất với truy vấn đầu vào.

- Đối tượng sử dụng:

+ Nhà nghiên cứu trí tuệ nhân tạo và thị giác máy tính, cần một công cụ truy vấn ảnh theo tri thức để phục vụ đánh giá mô hình và nghiên cứu học sâu đa phương thức.

+ Người dùng phổ thông, như sinh viên, nhà báo, hoặc chuyên viên sáng tạo nội dung cần tìm nhanh hình ảnh minh họa theo mô tả cụ thể bằng ngôn ngữ tự nhiên.

+ Các tổ chức doanh nghiệp, ví dụ như công ty truyền thông, thương mại điện tử, hay y tế, có nhu cầu truy xuất ảnh một cách chính xác và theo ngữ cảnh.

- Các tính năng nổi bật:

+ Tính tích hợp cao: Kết hợp nhiều mô-đun như phát hiện đối tượng, trích xuất quan hệ, biểu diễn tri thức và truy vấn ảnh trong một hệ thống thống nhất. Tích hợp công nghệ học sâu R-GCN, GCN, GAT và xử lý ngôn ngữ tự nhiên spaCy kết hợp cùng cơ sở dữ liệu đồ thị Neo4j.

+ Tính toàn diện: Hệ thống hiểu sâu ngữ nghĩa ảnh thông qua biểu diễn đồ thị tri thức (entity–relation–entity). Truy vấn được phân tích đến cấp độ quan hệ và ngữ cảnh thay vì chỉ nhận diện đối tượng đơn lẻ.

+ Tính linh hoạt: Hỗ trợ nhiều loại truy vấn: triplet, văn bản tự nhiên, truy vấn ảnh đầu vào. Linh hoạt trong lựa chọn mô hình biểu diễn tri thức TransE, GCN, R-GCN, GAT. Dễ mở rộng cho nhiều lĩnh vực ứng dụng khác nhau.

+ Dễ sử dụng: Giao diện thân thiện, nhập truy vấn tự nhiên, kết quả hiển thị trực quan. Có API sẵn sàng tích hợp vào hệ thống web hoặc ứng dụng. Tự động hóa quy trình trích xuất tri thức và truy vấn ảnh, giảm yêu cầu kỹ thuật cho người dùng.

1.1.3. Các chức năng của hệ thống tìm kiếm ảnh dựa trên đồ thị tri thức

- Tiền xử lý và xây dựng đồ thị tri thức ảnh:
 - + Ảnh được đưa qua mô hình phát hiện đối tượng Detectron2 để xác định các bounding box và phân loại thực thể.
 - + Từ các cặp đối tượng được phát hiện, hệ thống sử dụng quy tắc không gian vị trí, hướng tương đối và thông tin ngữ cảnh để trích xuất các quan hệ dạng subject–predicate–object.
 - + Kết quả được biểu diễn thành triplet RDF: entity 1, relation, entity 2 – tương ứng với các cạnh có hướng nối giữa hai node trong đồ thị.
 - + Mỗi ảnh sau đó được ánh xạ thành một tiêu đề đồ thị tri thức.
- Học biểu diễn tri thức ảnh (Embedding):
 - + Sử dụng các mô hình như TransE, R-GCN, GAT, GCN để ánh xạ mỗi tiêu đề đồ thị thành vector ngữ nghĩa.
 - + Học embedding cho cả ảnh và truy vấn nhằm phục vụ tìm kiếm theo không gian vector.
- Xử lý và ánh xạ truy vấn:
 - + Phân tích truy vấn ngôn ngữ tự nhiên hoặc triplet và chuyển thành đồ thị con tương ứng.
 - + Trích xuất từ khóa và quan hệ bằng spaCy hoặc OpenIE.
- Tìm kiếm và so khớp ảnh:
 - + Tính độ tương đồng thường là cosine similarity hoặc khoảng cách Euclidean giữa truy vấn và embedding của các ảnh trong tập dữ liệu.
 - + Sắp xếp và lựa chọn top-K ảnh có độ tương đồng cao nhất.
 - + Trả kết quả kèm theo thông tin giải thích các đối tượng và quan hệ trong ảnh phù hợp với truy vấn.

- Giao diện người dùng và API:

- + Giao diện người dùng trực quan: cho phép nhập truy vấn dạng văn bản, chọn truy vấn mẫu, xem kết quả kèm chú thích.

- + RESTful API: hỗ trợ tích hợp vào các nền tảng bên ngoài (như hệ thống học tập thông minh, thư viện ảnh số, ứng dụng quản lý y tế...).

- Đánh giá hệ thống:

- + Tính toán các chỉ số hiệu suất: Precision, Recall, F1-score.

- + So sánh giữa các mô hình embedding.

1.1.4. Đánh giá hiện trạng

- Trong bối cảnh dữ liệu hình ảnh phát triển bùng nổ như hiện nay, việc tìm kiếm ảnh theo ngữ nghĩa đang đặt ra nhiều thách thức. Các hệ thống tìm kiếm ảnh truyền thống, dựa vào từ khóa hoặc đặc trưng hình ảnh thuần túy, đang bộc lộ nhiều hạn chế:

- + Không hiểu được quan hệ giữa các đối tượng trong ảnh: Hầu hết các phương pháp chỉ nhận biết được có gì trong ảnh, nhưng không nắm bắt được mối quan hệ giữa các thành phần (who does what to whom). Điều này khiến hệ thống không thể xử lý các truy vấn như: “man holding phone” hoặc “dog lying next to baby”.

- + Thiếu khả năng biểu diễn ngữ nghĩa ảnh dưới dạng cấu trúc: Ảnh vốn chứa thông tin đa chiều (nhiều đối tượng, vị trí tương đối, hành động...) nhưng các vector biểu diễn đơn lẻ không phản ánh được toàn vẹn cấu trúc này.

- + Không tận dụng được thông tin ngữ cảnh hoặc mối quan hệ láng giềng giữa các ảnh: Trong thực tế, nhiều ảnh trong tập dữ liệu chia sẻ các cấu trúc đồ thị tương đồng (ví dụ: person – riding – horse, person – holding – racket) nhưng các hệ thống hiện tại chưa khai thác triệt để mối liên kết này để cải thiện độ chính xác khi tìm kiếm.

- + Truy vấn ngôn ngữ tự nhiên chưa được ánh xạ hiệu quả vào không gian hình ảnh: Mặc dù người dùng thường có xu hướng nhập truy vấn dạng ngữ nghĩa (câu hoặc mệnh đề), nhưng hệ thống truyền thống lại chỉ thực hiện so khớp theo từ vựng, dẫn đến độ chính xác thấp.

- Đề tài đề xuất một hệ thống tìm kiếm ảnh dựa trên đồ thị tri thức kết hợp học biểu diễn và khai thác thông tin láng giềng, với các giải pháp cụ thể như sau:

+ Biểu diễn tri thức ảnh bằng đồ thị RDF: Mỗi ảnh được chuyển thành tập các triplet (subject, predicate, object), hình thành tiểu đồ thị tri thức phản ánh mối quan hệ giữa các thực thể trong ảnh.

+ Trích xuất và học embedding với GCN, R-GCN, TransE, GAT: Các mô hình học sâu được sử dụng để học vector biểu diễn cho mỗi node và quan hệ trong đồ thị, sao cho giữ được ngữ nghĩa và khả năng tổng quát hóa. R-GCN giúp khai thác hiệu quả thông tin từ láng giềng gần trong đồ thị.

+ Tích hợp thông tin láng giềng để tăng cường khả năng suy luận: Ảnh không chỉ được so khớp với truy vấn mà còn với các ảnh có đồ thị tương tự về cấu trúc, nhờ đó tăng khả năng tìm đúng ảnh dù truy vấn không trùng khớp hoàn toàn.

+ Phân tích truy vấn ngôn ngữ tự nhiên bằng NLP: Sử dụng các công cụ như spaCy để chuyển đổi truy vấn thành một đồ thị con tương ứng, từ đó ánh xạ vào không gian embedding để tìm ảnh tương đồng nhất.

+ Phát triển hệ thống truy vấn và đánh giá kết quả: Hệ thống được thiết kế với API và giao diện web bằng Streamlit đơn giản, trực quan, hỗ trợ nhiều loại truy vấn. Đánh giá được các chỉ số như Precision, Recall, F1-score.

1.1.5. Mô tả bài toán

Đề tài “Tìm kiếm ảnh dựa trên đồ thị tri thức kết hợp thông tin láng giềng từ bộ dữ liệu COCO2017” được xây dựng nhằm phát triển một hệ thống có khả năng hiểu và phản hồi các truy vấn tìm kiếm ảnh mang tính ngữ nghĩa, thông qua việc khai thác cấu trúc tri thức ẩn trong mỗi bức ảnh, từ đó nâng cao độ chính xác và linh hoạt trong việc truy xuất hình ảnh.

Hệ thống hướng đến giải quyết bài toán khi người dùng đưa ra một truy vấn ở dạng ngôn ngữ tự nhiên hoặc dạng triplet (subject – predicate – object), hệ thống sẽ chuyển đổi truy vấn thành một tiểu đồ thị tri thức và tiến hành tìm kiếm các ảnh trong kho dữ liệu có cấu trúc tri thức tương tự hoặc gần giống. Mỗi bức ảnh trong tập dữ liệu COCO2017 được ánh xạ thành một tiểu đồ thị tri thức phản ánh các đối tượng trong ảnh, các mối quan hệ giữa chúng (như: person – holding – phone, dog – standing next to – child), và các thuộc tính liên quan (màu sắc, kích thước, vị trí...).

Trong giai đoạn đầu, hệ thống thực hiện trích xuất thông tin từ ảnh bằng cách sử dụng mô hình phát hiện đối tượng để xác định các thực thể và mối quan hệ giữa các thực thể đó, từ đó xây dựng tập hợp các triplet cấu thành nên đồ thị tri thức ảnh. Các triplet này tiếp tục nhúng embedding vào không gian vector thông qua các mô hình như TransE, GCN, R-GCN hoặc GAT để học biểu diễn ngữ nghĩa sâu hơn.

Tiếp theo, truy vấn của người dùng sẽ được phân tích ngữ nghĩa và chuyển thành đồ thị con nhờ vào các công cụ xử lý ngôn ngữ tự nhiên như spaCy. Đồ thị truy vấn này được nhúng vào không gian tương tự như các ảnh, và hệ thống tiến hành so khớp dựa trên độ tương đồng giữa các vector embedding, đồng thời tận dụng thông tin từ các láng giềng đồ thị để mở rộng phạm vi truy vấn và tăng khả năng tìm đúng ảnh ngay cả khi truy vấn không hoàn toàn trùng khớp với dữ liệu ảnh.

Hệ thống cung cấp giao diện truy vấn thân thiện, hỗ trợ cả nhập văn bản và chọn truy vấn mẫu. Người dùng có thể tương tác theo thời gian thực để tìm kiếm các ảnh chứa nội dung cụ thể như “person riding a bicycle”, “baby sitting on a chair”, hoặc các cấu trúc phức tạp hơn như “a boy, a girl and a dog are playing together”.

1.2. Cơ sở lý thuyết

1.2.1. Đồ thị tri thức và biểu diễn triplet RDF

1.2.1.1. Khái niệm đồ thị tri thức (Knowledge Graph)

Đồ thị tri thức là một mô hình biểu diễn tri thức bằng cấu trúc đồ thị, trong đó các thực thể (entity) được biểu diễn dưới dạng nút (node), và các mối quan hệ (relation) giữa chúng được biểu diễn dưới dạng cạnh (edge). Thông qua việc ánh xạ thông tin về thế giới thực thành đồ thị, KG cho phép lưu trữ, liên kết và suy luận trên tri thức một cách hiệu quả và có khả năng mở rộng.

Thực thể (Entities) biểu thị và lưu trữ thông tin chi tiết về con người, địa điểm, đối tượng hoặc tổ chức. Mỗi thực thể là một nút có một hoặc đôi khi là nhiều nhãn để xác định loại nút và tùy chọn có thể có một hoặc nhiều thuộc tính (attributes). Các nút đôi khi cũng được gọi là đỉnh. Ví dụ các thực thể trong biểu đồ tri thức thương mại điện tử thường biểu diễn các thực thể như khách hàng, sản phẩm và đơn hàng:

Quan hệ (relationships) xác định cách các thực thể liên kết với nhau. Giống như các nút, mỗi mối quan hệ có một nhãn xác định loại mối quan hệ và có thể tùy chọn có

một hoặc nhiều thuộc tính. Mỗi quan hệ đôi khi cũng được gọi là cạnh. Trong ví dụ thương mại điện tử, tồn tại mối quan hệ giữa khách hàng và các nút đặt hàng, nắm bắt mối quan hệ “ordered” giữa khách hàng và đơn hàng của họ.

Thuộc tính (Attributes) chứa thông tin bổ sung về thực thể và mối quan hệ.

Mỗi tri thức trong đồ thị thường được biểu diễn dưới dạng triplet (bộ ba):

(subject, predicate, object)

Trong đó:

- Subject là thực thể chủ thể (ví dụ: person, cat, dog...),
- Predicate là mối quan hệ (ví dụ: stand on, next to, riding...),
- Object là thực thể đối tượng được kết nối với chủ thể.

Chẳng hạn, câu “a man hold a phone” được ánh xạ thành triplet:

(man, hold, phone)

Ví dụ:

Organizing Principles (Nguyên tắc tổ chức): thể hiện cấu trúc phân loại của các loại sản phẩm:

- Category (Danh mục): Foodstuffs
- Types (Loại sản phẩm): Snacks, Fruit, Fresh Foods, Fish
- Các loại sản phẩm này được liên kết bằng quan hệ TYPE_OF, cho thấy thực phẩm có nhiều loại con như trái cây, cá, đồ ăn vặt.

Instance Data (Dữ liệu thực tế): thể hiện dữ liệu cụ thể về khách hàng, đơn hàng, địa chỉ và sản phẩm:

- Khách hàng (Person): Daniel và Sunita. Cả hai có quan hệ HAS_ADDRESS với cùng một địa chỉ 100 Main St.
- Đơn hàng (Order): Daniel đặt một đơn hàng (PLACED_ORDER) có ID: 100.
- Đơn hàng này chứa sản phẩm Apple (1 đơn vị) và Salmon (2 đơn vị), thể hiện bằng quan hệ CONTAINS.
- Sản phẩm (Product): Apple thuộc loại Fruit, Salmon thuộc loại Fish.

Hệ thống đồ thị tri thức có khả năng mở rộng dễ dàng, khi mỗi dữ liệu mới chỉ cần được ánh xạ thêm các triplet bổ sung vào đồ thị hiện có.

1.2.1.2. Biểu diễn triplet theo mô hình RDF

Resource Description Framework (RDF) là một mô hình dữ liệu tiêu chuẩn do W3C đề xuất để mô tả thông tin theo cấu trúc triplet. Trong RDF:

- Mỗi subject và object là một URI (Uniform Resource Identifier) hoặc literal.
- Mỗi predicate là một URI xác định loại quan hệ.
- RDF thường được biểu diễn dưới dạng tập hợp các câu lệnh (s, p, o), tương đương với đồ thị có hướng.

Ví dụ:

`:man1 :holds :phone1 .`

`:man1 rdf:type :Person .`

`:phone1 rdf:type :Device .`

Các triplet RDF này có thể dễ dàng được ánh xạ thành một đồ thị với nút man1 và phone1, cạnh holds, và các thông tin phân loại thêm thông qua rdf:type.

1.2.1.3. Ưu điểm của biểu diễn tri thức dạng đồ thị

Việc biểu diễn tri thức dưới dạng đồ thị mang lại nhiều lợi ích trong các hệ thống học máy và truy vấn ngữ nghĩa:

- Khả năng mô hình hóa quan hệ phức tạp giữa các thực thể, không bị giới hạn như các bảng dữ liệu phẳng.
- Tính linh hoạt và khả năng mở rộng cao, dễ dàng bổ sung các mối quan hệ mới vào hệ thống mà không làm ảnh hưởng đến cấu trúc cũ.
- Khả năng liên kết đa miền tri thức, ví dụ như kết hợp tri thức y tế, hình ảnh, văn bản, hoặc cảm biến trong một mạng tri thức thống nhất.
- Hỗ trợ mạnh mẽ cho suy luận logic, thông qua việc ứng dụng các kỹ thuật học biểu diễn tri thức như TransE, R-GCN, GAT, GCN.

1.2.1.4. Vai trò của đồ thị tri thức trong bài toán tìm kiếm ảnh

Trong đề tài này, mỗi ảnh từ bộ dữ liệu COCO2017 sẽ được biểu diễn thành một tiêu đồ thị tri thức, trong đó:

- Các đối tượng trong ảnh như người, chó, xe đạp sẽ là các nút.
- Các mối quan hệ như đứng cạnh, cầm là các cạnh có hướng giữa các thực thể.
- Thông tin vị trí, thuộc tính và tương tác trong ảnh được biểu diễn một cách có cấu trúc.

Cách biểu diễn này giúp cho việc so khớp truy vấn với ảnh trở nên chính xác hơn, vì hệ thống không chỉ xem xét sự xuất hiện của các đối tượng, mà còn xét đến mối quan hệ giữa chúng – điều vốn rất khó đạt được với các mô hình thị giác truyền thống.

1.2.2. Mạng nơ-ron đồ thị (GCN, GAT, R-GCN)

1.2.2.1. Tổng quan về GCN – Graph Convolutional Network

Mạng nơ-ron tích chập trên đồ thị (GCN) là một kiến trúc tiêu biểu trong GNN. GCN mở rộng ý tưởng tích chập trong mạng CNN sang miền đồ thị thông qua việc tổng hợp (aggregate) thông tin từ các nút láng giềng của một nút mục tiêu.

Công thức cập nhật biểu diễn của một nút tại lớp $l + 1$ như sau:

$$H^{(l+1)} = \sigma \left(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} H^{(l)} W^{(l)} \right)$$

Trong đó:

- $\tilde{A} = A + I$ là ma trận kề có thêm liên kết tự thân.
- \tilde{D} là ma trận bậc của \tilde{A}
- $H^{(l)}$ là ma trận đặc trưng đầu vào tại lớp l
- $W^{(l)}$ là trọng số học được
- σ là hàm kích hoạt thường dùng ReLU

GCN đặc biệt hiệu quả với đồ thị đồng nhất chỉ có một loại nút và cạnh, và có thể áp dụng tốt vào bài toán học biểu diễn nút trong đồ thị tri thức ảnh.

Tư tưởng cốt lõi của GCN là cho phép lan truyền và tổng hợp thông tin qua các tầng lân cận trong đồ thị bằng cách sử dụng phép biến đổi tuyến tính kết hợp với chuẩn hóa ma trận kề. Mỗi bước lan truyền trong GCN được hiểu như một quá trình “trao đổi thông tin” giữa một nút và các nút kề của nó, từ đó cập nhật biểu diễn vector của nút theo ngữ cảnh xung quanh. Điều này giúp các đỉnh không chỉ mang thông tin riêng biệt mà còn phản ánh được mối quan hệ cấu trúc trong toàn đồ thị.

Tuy nhiên, một hạn chế của GCN là nó giả định tất cả các quan hệ trong đồ thị là đồng nhất – tức không phân biệt được loại cạnh hay loại thực thể. Do đó, GCN phù hợp hơn với các bài toán trên đồ thị đơn giản, nơi tất cả các đỉnh và cạnh có cùng bản chất. Khi áp dụng cho bài toán biểu diễn đồ thị tri thức trong tìm kiếm ảnh, GCN vẫn mang lại kết quả khả quan nhờ khả năng lan truyền đặc trưng, song không thể hiện rõ sự khác biệt giữa các quan hệ ngữ nghĩa phức tạp như các mô hình có phân biệt loại cạnh.

1.2.2.2. Tổng quan về GAT – Graph Attention Network

Graph Attention Network (GAT) cải tiến GCN bằng cách đưa cơ chế attention (chú ý) vào quá trình tổng hợp thông tin từ các nút lân cận. Thay vì giả định các nút láng giềng đóng góp như nhau, GAT học một trọng số attention cho từng cặp nút.

Trọng số attention giữa nút i và j được tính bằng:

$$e_{ij} = \text{LeakyReLU} \left(\vec{a}^T \left[W \vec{h}_i || W \vec{h}_j \right] \right)$$

sau đó chuẩn hóa bằng hàm softmax:

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in N(i)} \exp(e_{ik})}$$

GAT có ưu điểm là tự động học được mức độ quan trọng của từng nút lân cận, do đó thích hợp với đồ thị không đồng nhất hoặc có độ kết nối không đồng đều, như trong truy vấn ảnh từ đồ thị tri thức.

GAT là một trong những kiến trúc nổi bật giúp khắc phục hạn chế của GCN trong việc giả định mức đóng góp của các nút lân cận là như nhau. Thay vào đó, GAT sử dụng cơ chế attention để xác định trọng số khác nhau cho từng kết nối, cho phép mô hình học được mức độ ảnh hưởng thực sự của từng nút kề. Điều này đặc biệt hữu ích trong các đồ thị không đồng nhất, nơi mà không phải mọi quan hệ giữa các đỉnh đều mang ý nghĩa tương đương.

Một đặc điểm nổi bật khác của GAT là khả năng mở rộng với multi-head attention, cho phép mô hình học được nhiều khía cạnh quan hệ đồng thời. Các kết quả attention từ nhiều “đầu” khác nhau có thể được nối hoặc trung bình, tùy theo mục tiêu biểu diễn. Nhờ cơ chế này, GAT không những tăng tính ổn định của việc học mà còn giảm nguy cơ lệch hướng khi một nguồn thông tin chi phối quá mạnh.

Trong các bài toán như phân loại đỉnh, dự đoán liên kết hoặc tìm kiếm ảnh theo truy vấn tri thức, GAT tỏ ra đặc biệt hiệu quả nhờ khả năng chọn lọc thông tin phù hợp với ngữ cảnh. Cơ chế attention cũng mang lại tính giải thích cao hơn so với các mô hình đồ thị truyền thống, bởi nó cung cấp trực tiếp trọng số thể hiện mức độ liên quan giữa các nút.

1.2.2.3. Tổng quan về R-GCN – Relational Graph Convolutional Network

Relational Graph Convolutional Network (R-GCN) được thiết kế để làm việc với đồ thị có nhiều loại quan hệ – rất phù hợp với đồ thị tri thức, nơi mỗi cạnh mang nhãn đại diện cho một loại quan hệ (ví dụ: hold, near, on,...).

Công thức cập nhật embedding trong R-GCN:

$$\vec{h}_i^{(l+1)} = \sigma \left(\sum_{r \in R} \sum_{j \in N_r(i)} \frac{1}{c_{i,r}} W_r^{(l)} \vec{h}_j^{(l)} + W_0^{(l)} \vec{h}_i^{(l)} \right)$$

Trong đó:

R là tập các quan hệ

$N_r(i)$ là tập các nút láng giềng của i qua quan hệ r

$W_r^{(l)}$ là ma trận trọng số ứng với quan hệ r

$W_0^{(l)}$ là ma trận dành cho liên kết tự thân

$c_{i,r}$ là hệ số chuẩn hóa

σ là hàm kích hoạt

Bên cạnh khả năng tổng hợp thông tin lân cận như các kiến trúc GNN khác, điểm nổi bật của R-GCN là khả năng xử lý đồ thị dị thể với nhiều loại quan hệ khác nhau giữa các đỉnh. Điều này đặc biệt phù hợp với bài toán khai thác đồ thị tri thức, nơi mỗi cạnh không chỉ biểu thị kết nối mà còn mang theo một ý nghĩa ngữ nghĩa cụ thể (như “thuộc về”, “được gắn với”).

Không giống như GCN thông thường, R-GCN sử dụng một tập hợp các ma trận trọng số riêng biệt cho từng loại quan hệ, cho phép mô hình học được các phép biến đổi khác nhau tương ứng với từng kiểu liên kết. Điều này giúp mô hình phân biệt được vai trò của các đỉnh lân cận theo từng ngữ cảnh quan hệ cụ thể, từ đó nâng cao độ chính xác trong việc học biểu diễn nút.

Một vấn đề tiềm ẩn trong việc sử dụng nhiều ma trận trọng số là nguy cơ làm tăng số lượng tham số, dẫn đến overfitting khi tập dữ liệu không đủ lớn. Để khắc phục, R-GCN sử dụng các kỹ thuật như chia sẻ tham số hoặc ràng buộc dạng thừa (basis decomposition) nhằm kiểm soát số lượng tham số và cải thiện khả năng tổng quát hóa.

1.2.3. Mô hình biểu diễn tri thức (TransE, TransH)

1.2.3.1. Mô hình TransE

Translation Embedding (TransE) là một mô hình dùng để biểu diễn các quan hệ trong đồ thị tri thức bằng vector.

TransE là một mô hình đơn giản và hiệu quả cho việc biểu diễn đồ thị tri thức dưới dạng vector, được đề xuất bởi Bordes et al. (2013).

Mục tiêu của TransE là biểu diễn một triplet (h, r, t) sao cho:

$$h + r \approx t$$

Trong đó:

h : vector embedding của entity head (subject)

r : vector embedding của quan hệ

t : vector embedding của entity tail (object)

Với mỗi triplet đúng:

$$(h, r, t) \rightarrow \text{muốn có: } \text{embedding}(h) + \text{embedding}(r) \approx \text{embedding}(t)$$

Tức là mô hình học hướng dịch chuyển r trong không gian để đi từ h đến t

Embedding: Biểu diễn vector hóa của entity và relation trong không gian \mathbb{R}^d .
Mục tiêu là học được không gian sao cho các quan hệ được bảo toàn bằng phép cộng vector.

Biểu diễn toán học:

$h \in \mathbb{R}^d$: embedding của subject (head)

$r \in \mathbb{R}^d$: embedding của quan hệ

$t \in \mathbb{R}^d$: embedding của object (tail)

Mục tiêu của TransE:

$$h + r \approx t$$

Nghĩa là: $|h + r - t| \approx 0$ (cho triplet đúng) $|h' + r - t'|$ lớn (cho triplet sai)

Hàm đánh giá (Scoring Function) để đo mức độ "hợp lý" của một triplet (h, r, t) ,
TransE định nghĩa:

$$f(h, r, t) = |h + r - t|_1 \text{ or } |h + r - t|_2$$

Một triplet càng "hợp lý" thì khoảng cách này càng nhỏ.

Hàm loss của TransE với một batch các triplet đúng (h, r, t) và triplet sai (h', r, t') , loss của TransE là:

$$L = \sum \max(0, \gamma + |h + r - t| - |h' + r - t'|)$$

Trong đó:

γ : margin (khoảng cách an toàn giữa triplet đúng và sai)

Norm thường là L1 hoặc L2

1.2.3.2. Mô hình TransH

Được đề xuất bởi Wang et al. (2014), TransH mở rộng TransE bằng cách giả định rằng mỗi quan hệ không chỉ là một vector dịch mà còn gắn liền với một mặt phẳng chuyên biệt. Mỗi thực thể được chiếu lên mặt phẳng này trước khi áp dụng phép dịch.

Cho một quan hệ r , định nghĩa mặt phẳng bởi vector pháp tuyến w_r , các bước thực hiện gồm:

Chiếu thực thể lên mặt phẳng quan hệ:

$$h_{\perp} = h - w_r^T h \cdot w_r \text{ và } t_{\perp} = t - w_r^T t \cdot w_r$$

Ánh xạ quan hệ:

$$h_{\perp} + dr \approx t_{\perp}$$

Hàm mất mát:

$$L = \sum_{(h,r,t)} [\gamma + ||h_{\perp} + d_r - t_{\perp}|| - ||h'_{\perp} + d_r - t'_{\perp}||]$$

1.2.4. Các phương pháp tìm kiếm ảnh truyền thống và hiện đại

1.2.4.1. Phương pháp truyền thống

- Các phương pháp truyền thống thường dựa trên việc trích xuất thủ công các đặc trưng hình ảnh (hand-crafted features) như:

+ SIFT (Scale-Invariant Feature Transform): mô tả cục bộ các điểm đặc trưng bền vững với thay đổi ánh sáng, xoay, thu phóng.

+ SURF (Speeded Up Robust Features): cải tiến tốc độ so với SIFT, sử dụng integral image và Haar wavelet.

+ HOG (Histogram of Oriented Gradients): mô tả hướng cạnh cục bộ, thường dùng trong phát hiện người.

+ Color Histogram / Texture: biểu diễn phân bố màu hoặc mẫu kết cấu trong ảnh.

- Sau khi trích xuất đặc trưng, ảnh thường được biểu diễn bằng vector đặc trưng cố định, sau đó so sánh khoảng cách giữa các vector (thường dùng L2, cosine) để tìm ảnh tương đồng.

- Một phương pháp phổ biến là BoVW: Trích xuất SIFT → Clustering (KMeans) → ánh xạ ảnh thành histogram của “từ thị giác” → so sánh histogram.

- Hạn chế:

+ Không nắm bắt được ngữ nghĩa cấp cao.

+ Nhạy cảm với biến đổi hình học hoặc nhiễu.

+ Không phản ánh mối quan hệ giữa các đối tượng trong ảnh.

1.2.4.2. Phương pháp hiện đại dựa trên học sâu

- Với sự phát triển của deep learning, các phương pháp tìm kiếm ảnh hiện đại sử dụng mạng nơ-ron tích chập CNN để trích xuất đặc trưng có khả năng khái quát cao hơn. Một số mô hình và chiến lược nổi bật:

- CNN (VGG, ResNet, EfficientNet): dùng để trích xuất đặc trưng toàn cục hoặc vùng cục bộ từ ảnh.

- Faster R-CNN, YOLO, Mask R-CNN: dùng trong truy xuất ảnh dựa trên đối tượng, giúp ánh xạ từng đối tượng trong ảnh vào không gian đặc trưng.

- CLIP (Contrastive Language–Image Pretraining): mô hình học liên kết giữa văn bản và hình ảnh, cho phép tìm kiếm ảnh bằng ngôn ngữ tự nhiên.

- Visual-Semantic Embedding (VSE): ánh xạ cả ảnh và câu truy vấn vào cùng không gian vector để so sánh.

- Đặc biệt, các hệ thống dựa trên đồ thị tri thức (Knowledge Graphs) đang được áp dụng nhằm tăng khả năng hiểu ngữ cảnh và quan hệ giữa các đối tượng trong ảnh. Trong đó:

+ Ảnh được chuyển thành tiểu đồ thị bao gồm các thực thể (object) và những quan hệ (relation).

+ Dùng GNN để học embedding cho toàn bộ tiểu đồ thị.

+ Truy vấn ảnh trở thành bài toán so khớp giữa đồ thị truy vấn và các đồ thị ảnh.

1.2.4.3. So sánh hai hướng tiếp cận

Tiêu chí	Phương pháp truyền thống	Phương pháp hiện đại
Đặc trưng	Thủ công (SIFT, HOG, BoVW)	Học tự động bằng CNN/GNN
Biểu diễn quan hệ đối tượng	Không có	Có thể biểu diễn qua GNN, đồ thị tri thức
Khả năng hiểu ngữ nghĩa	Rất hạn chế	Cao nhờ học từ dữ liệu lớn
Truy vấn bằng ngôn ngữ tự nhiên	Không	Có nhờ CLIP, BERT, VSE
Khả năng mở rộng hệ thống	Thấp	Cao, dễ tích hợp API/hệ thống phân tán

Bảng 1.1 So sánh hai phương pháp

1.2.5. Tích hợp NLP trong truy vấn ảnh (spaCy, OpenIE, BERT)

1.2.5.1. spaCy – Phân tích cú pháp và trích xuất thực thể

spaCy là một thư viện NLP mã nguồn mở được phát triển với mục tiêu phục vụ ứng dụng thực tiễn, đặc biệt nhấn mạnh vào tốc độ và độ chính xác. Với bộ mô hình được huấn luyện trên các tập dữ liệu lớn như OntoNotes hoặc Universal Dependencies, spaCy cung cấp các chức năng then chốt trong việc xử lý truy vấn ảnh:

- Tokenization: chia câu truy vấn thành các đơn vị từ ngữ (tokens).
- Part-of-Speech Tagging (POS): gán loại từ (danh từ, động từ, tính từ...) cho từng token.
- Dependency Parsing: xây dựng cây cú pháp thể hiện quan hệ phụ thuộc giữa các từ trong câu.
- Named Entity Recognition (NER): nhận diện các thực thể có tên (ví dụ: người, địa điểm, vật thể).

Ví dụ: Truy vấn: “A girl is riding a bicycle on the street” spaCy cho kết quả:

- + Subject: girl
- + Predicate: riding
- + Object: bicycle
- + Prepositional phrase: on the street

Từ đó, câu truy vấn có thể được ánh xạ thành các triplets: (“girl”, “riding”, “bicycle”) và (“bicycle”, “on”, “street”). Những triplet này có thể được dùng để khớp với các tiêu đề thị trong ảnh.

1.2.5.2. OpenIE – Trích xuất bộ ba quan hệ tự động

Open Information Extraction (OpenIE) là kỹ thuật trích xuất thông tin không phụ thuộc miền (domain-independent), cho phép tự động phát hiện các quan hệ ngữ nghĩa trong câu văn mà không cần tập luật định nghĩa trước. Một số hệ thống nổi bật của OpenIE đến từ Stanford NLP hoặc AllenNLP.

Ví dụ: Truy vấn: “The man wearing a red shirt is holding a dog” Triplets trích xuất được: (“man”, “is wearing”, “red shirt”), (“man”, “is holding”, “dog”)

Điểm mạnh của OpenIE là khả năng mở rộng tốt, không giới hạn bởi từ vựng cố định và có thể trích xuất nhiều quan hệ phức tạp trong một câu. Điều này đặc biệt hữu ích trong hệ thống truy vấn ảnh vì cho phép biểu diễn một câu miêu tả thành nhiều quan hệ để tìm ảnh có cấu trúc ngữ nghĩa tương đồng.

1.2.5.3. BERT – Biểu diễn ngữ nghĩa truy vấn theo ngữ cảnh

BERT (Bidirectional Encoder Representations from Transformers) là mô hình ngôn ngữ dựa trên kiến trúc Transformer, được huấn luyện theo cơ chế hai chiều để nắm bắt ngữ cảnh một cách sâu sắc. BERT đã mang lại bước tiến vượt bậc trong các tác vụ NLP như phân loại văn bản, QA, và đặc biệt là biểu diễn câu dưới dạng vector embedding.

Trong hệ thống truy vấn ảnh, BERT được sử dụng để:

Mã hóa toàn bộ câu truy vấn thành một vector embedding phản ánh ý nghĩa ngữ cảnh tổng thể.

Ví dụ: Truy vấn: “A child sitting under a tree next to a bicycle”. BERT mã hóa toàn bộ truy vấn thành một vector 768 chiều (hoặc nhiều chiều hơn tùy phiên bản), có thể so sánh trực tiếp với vector biểu diễn ảnh được học từ các mô hình như CLIP, ViT hoặc CNN kết hợp GNN.

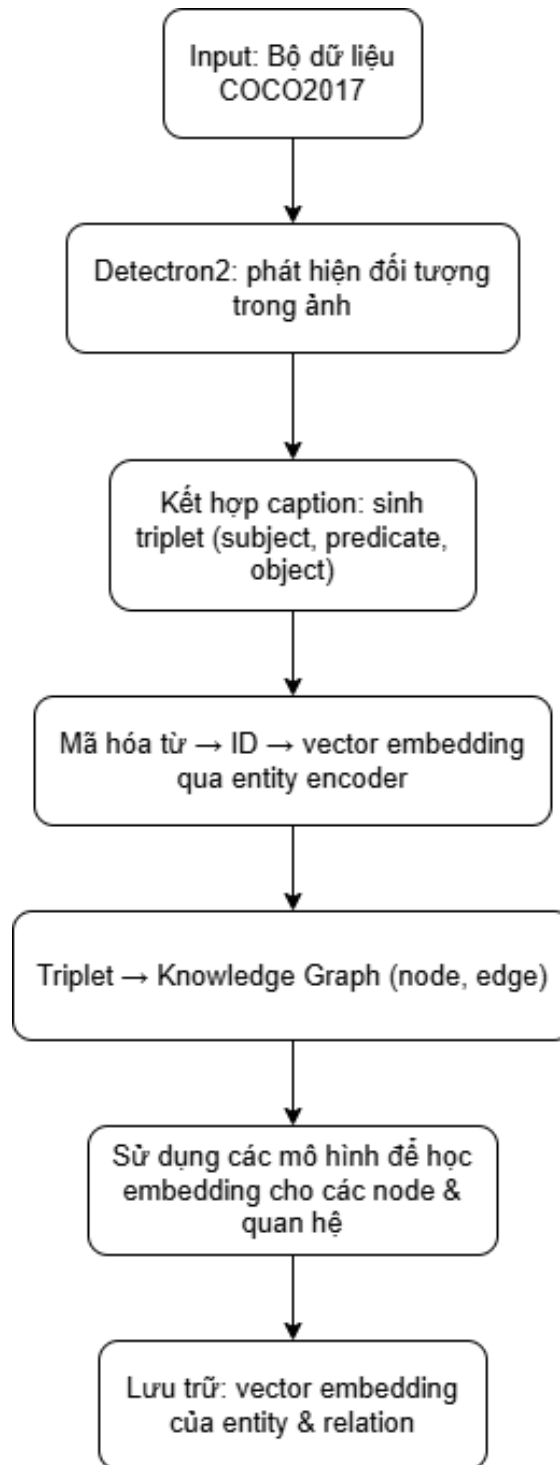
Ngoài ra, việc fine-tune BERT để thích ứng với tập dữ liệu truy vấn ảnh cụ thể giúp tăng độ chính xác trong việc so khớp ngữ nghĩa.

CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

2.1. Kiến trúc tổng thể

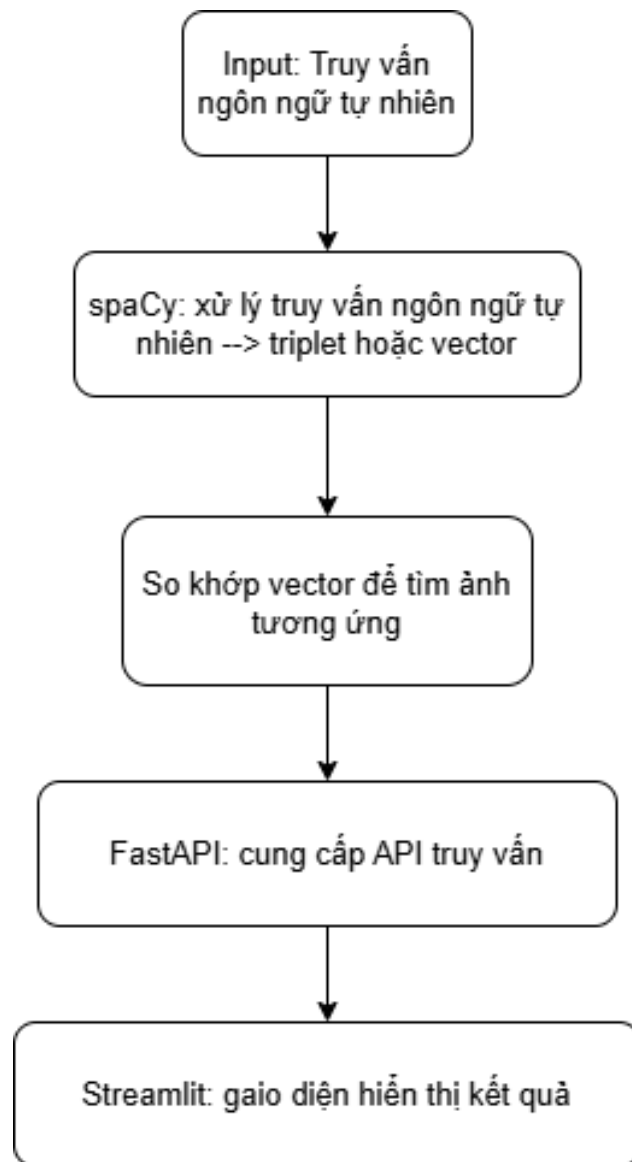
Hệ thống được thiết kế theo kiến trúc tầng bao gồm ba thành phần chính:

- Tầng trích xuất đặc trưng (Feature Extraction Layer): Sử dụng các mô hình học sâu như Detectron2 để phát hiện đối tượng trong ảnh, từ đó kết hợp caption để tạo thành triplets (subject, predicate, object).
- Tầng xử lý ngữ nghĩa (Semantic Processing Layer): Các triplet được ánh xạ thành đồ thị tri thức; sử dụng các mô hình như TransE, GCN, GAT, R-GCN để học vector embedding cho các đối tượng và quan hệ.



Hình 2.1 Sơ đồ tầng xử lý

- Tầng truy vấn và giao diện (Query & Interface Layer): Truy vấn ngôn ngữ tự nhiên được xử lý bằng spaCy, ánh xạ thành đồ thị con hoặc vector để so khớp với ảnh. Kết quả được hiển thị qua giao diện Streamlit và API FastAPI.



Hình 2.2 Sơ đồ tầng truy xuất

2.2. Quy trình xử lý dữ liệu COCO

Bộ dữ liệu COCO (Common Objects in Context) được thu thập và cung cấp bởi nhóm nghiên cứu tại Microsoft. Nó chứa hơn 330k hình ảnh, với hơn 200k hình ảnh đã được gán nhãn, bao gồm các đối tượng và chú thích về các mối quan hệ giữa chúng. COCO cung cấp dữ liệu cho nhiều nhiệm vụ như phát hiện đối tượng, phân đoạn đối tượng, chú thích ảnh và tạo câu mô tả ảnh.

COCO được thu thập từ internet, bao gồm những hình ảnh từ nhiều nguồn khác nhau với các đối tượng được gán nhãn theo các lớp khác nhau. Mỗi hình ảnh trong COCO có thể chứa một hoặc nhiều đối tượng, được phân loại và mô tả chi tiết.

COCO (Common Objects in Context) là một bộ dữ liệu lớn và đa dạng được sử dụng phổ biến trong các bài toán thị giác máy tính như:

- + Nhận diện đối tượng (object detection)
- + Phân đoạn ảnh (segmentation)
- + Chú thích ảnh (image captioning)
- + Hiểu cảnh vật (scene understanding)

Thành phần	Số lượng
+ Số ảnh train2017	~118,000
+ Số ảnh val2017	~5,000
+ Số object categories	80
+ Chú thích ảnh	Mỗi ảnh có 5 caption mô tả bằng tiếng Anh

Quy trình trích xuất thông tin từ mô tả ảnh (caption) trong bộ dữ liệu COCO và xây dựng thành đồ thị tri thức (Knowledge Graph) gồm các bước chính sau:

- Bước 1: Tiền xử lý caption
 - + Dữ liệu caption được lấy từ file captions_train2017.json.
 - + Các caption được nhóm lại theo image_id để gắn với từng ảnh cụ thể.
 - + Áp dụng phân tích cú pháp ngữ nghĩa (semantic parsing) để trích xuất các bộ ba (subject – predicate – object) từ câu mô tả.

Ví dụ: "A dog is catching a frisbee" → (dog, catch, frisbee)

- Bước 2: Chuẩn hóa thực thể
 - + Chuẩn hóa các danh từ đồng nghĩa về cùng một thực thể: ví dụ “man”, “boy”, “woman” → “person”.
 - + Loại bỏ các triplet trùng lặp hoàn toàn (giống cả 3 thành phần) hoặc giống cặp (subject, object) có cùng quan hệ.
- Bước 3: Tạo đồ thị tri thức
 - + Node: Đại diện cho thực thể (subject và object), có thuộc tính như tên, loại, v.v.

- + Edge: Đại diện cho quan hệ (predicate) giữa hai node.
- + Gắn thông tin bổ sung như image_id và caption gốc vào các cạnh như metadata.
- Bước 4: Lưu đồ thị vào Neo4j
- + Sử dụng ngôn ngữ truy vấn Cypher để chèn node và quan hệ vào cơ sở dữ liệu Neo4j. Sau chuẩn hóa và loại trùng, từ hơn 36.000 triplet ban đầu chỉ còn: 63 node (thực thể duy nhất), 544 quan hệ (edge không trùng)

2.3. Thiết kế đồ thị tri thức từ caption và Detectron2

Mỗi ảnh sau khi xử lý sẽ được ánh xạ thành một tiểu đồ thị $G = (V, E)$:

- + Tập nút V: Mỗi nút đại diện cho một đối tượng hoặc khái niệm trong ảnh.
- + Tập cạnh E: Mỗi cạnh là một quan hệ định hướng giữa hai đối tượng, lấy từ caption hoặc từ dữ liệu Visual Genome.

Ví dụ: Caption: “A man is riding a horse”

→ Triplet: (man, riding, horse)

→ Đồ thị: Node: man, horse; Edge: man → horse [riding]

Mỗi triplet tạo ra một cạnh định hướng nối hai node, từ đó cấu thành nên KG của ảnh. Đồ thị được xây dựng bằng thư viện NetworkX, sau đó chuyển đổi thành định dạng Cypher để đưa vào Neo4j.

36766	394940	1960073	dining tabl	1.67	402.5	426	632.5
36767	394940	2164128	person	0	77.18	425.79	490.84
36768	15335	98619	couch	2.06	143.48	547.1	317.94
36769	15335	1216332	person	365.92	15.07	640	473.54
36770	15335	1230490	person	173.66	139.15	381.84	474.61
36771	15335	1248228	person	237.45	46.43	337.09	153.58
36772	15335	1277736	person	1.08	71.89	214.05	448.11
36773	15335	1282318	person	508.83	35.38	558.24	164.66
36774	15335	1318954	person	554.15	23.64	590.77	87.5
36775	15335	1538947	bowl	0.28	429.52	98.93	480
36776	15335	1694331	person	541.45	21.21	616.52	162.93
36777	15335	1705756	person	343.63	53.65	511.25	165.39
36778	15335	1753431	person	160.39	66.68	238.38	151.37
36779	15335	1879878	cup	599.96	422.9	640	480
36780	15335	1976132	cell phone	2.47	306.54	47.35	323.29
36781	15335	2025828	person	362.44	102.44	386.67	130.14
36782	15335	2028889	person	535.82	21.64	565.97	68.67
36783							

Hình 2.3 Tất cả object trích xuất được từ COCO2017

1631	576427	person	use	laptop	a person is using a laptop while sitting at a desk				
1632	535130	person	get	laptop	A person sits on an exam table, while a medial person gets the laptop.				
1633	376831	person	cut	pizza	A person is cutting pizza with a pizza cutter.				
1634	437964	person	have	laptop	A person with striped tights has a laptop on their legs.				
1635	198196	person	eat	pizza	A person is eating pizza at the computer desk.				
1636	263450	person	grab	bowl	A person is grabbing a bowl of food in the oven.				
1637	140999	laptop	have	mouse	This laptop has a mouse and headphones attached.				
1638	268579	person	show	clock	A young person is showing a clock displayed on a cell phone screen.				
1639	541868	person	hold	scissors	A person in a clown mask is holding giant scissors.				
1640	362966	person	hold	fork	A person holds a fork near a plate of food.				
1641	240545	person	sit	bench	PERSON SITTING ON A BENCH IN BETWEEN TWO OVER SIZED BEARS				
1642	254795	person	use	scissors	The person is using the scissors to cut the money.				
1643	361451	clock	read	clock	A clock reads almost 4 o clock on a tall tower				
1644	72583	bird	hold	bowl	The monkey is climbing the rope going toward the vase while the bird holds a bowl of fruit.				
1645	298721	person	stab	person	a person on the ground stabbing a person on the ground				
1646	492183	refrigerator	have	bottle	A small refrigerator has a bottle of wine it.				
1647	39470	person	hold	scissors	A person is holding scissors over some fabric.				
1648	510731	person	hold	donut	Person holding up a small donut in their hand.				
1649	138092	person	hold	fork	The person is holding a fork and butter knife over a plate of food.				
1650	421086	person	slice	pizza	A person wearing gloves is slicing pizza and is pulling out of the pan.				
1651									

Hình 2.4 Tất cả relationship trích xuất được từ COCO2017

2.4. Thiết kế giao diện và API FastAPI + Streamlit

FastAPI đảm nhận việc xây dựng các route RESTful cho các thao tác như:

+ /query/triplet: Truy vấn ảnh theo triplet.

+ /query/text: Truy vấn ảnh theo văn bản tự nhiên.

+ /embedding: Trả về vector embedding của truy vấn hoặc ảnh.

Streamlit cung cấp giao diện người dùng đơn giản, dễ thao tác:

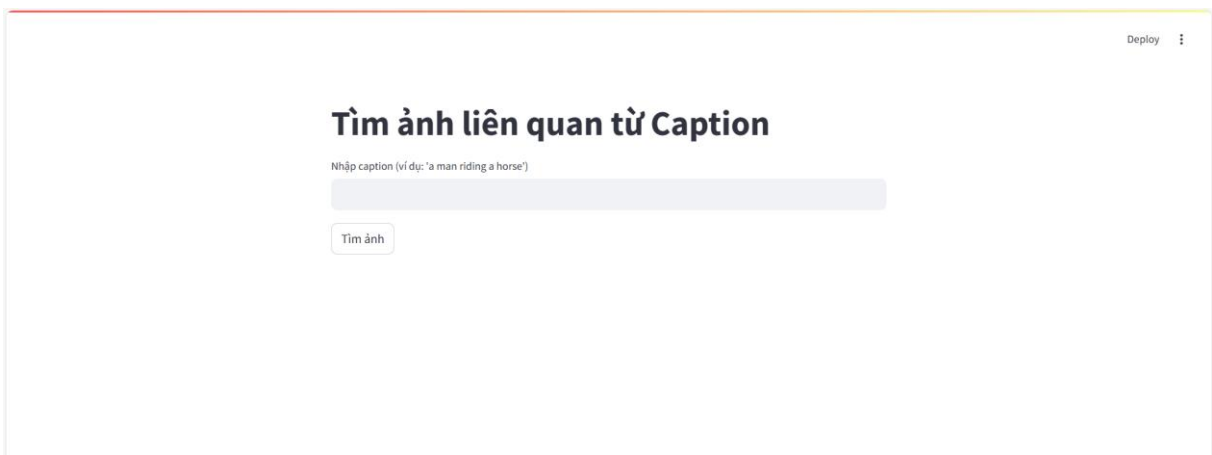
- + Nhập truy vấn: có thể là văn bản hoặc dạng RDF.
- + Hiển thị kết quả: các ảnh phù hợp, độ tương đồng, đồ thị ảnh.

```
(coco_kg) C:\Users\admin>streamlit run app_streamlit.py

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://172.100.136.96:8501
```

Hình 2.5 Run app trên localhost



Hình 2.6 Giao diện app

Toàn bộ giao diện đảm bảo tính trực quan và khả năng tích hợp dễ dàng với backend nhờ Streamlit rõ ràng cùng API.

2.5. Lưu trữ KG với Neo4j

Để biểu diễn bộ dữ liệu COCO dưới dạng đồ thị trong Neo4j, ta sử dụng các node để đại diện cho các đối tượng và các relationship để mô tả mối quan hệ giữa chúng. Đây là cách ta có thể biểu diễn đồ thị trong Neo4j:

- Nodes (Nút): Mỗi đối tượng trong ảnh là một node trong đồ thị. Các đặc trưng của node có thể bao gồm:

- + id: ID của đối tượng.
- + class_id: Loại đối tượng (ví dụ: "person", "car").

+ confidence: Độ tin cậy của mô hình đối với đối tượng.

+ :NEXT_TO: Khi các đối tượng nằm gần nhau.

+ :PART_OF: Khi một đối tượng là một phần của đối tượng khác.

Ví dụ, trong một bức ảnh, nếu có người và xe gần nhau, ta có thể tạo ra các node cho "person" và "car", cùng với mối quan hệ :NEXT_TO giữa chúng.

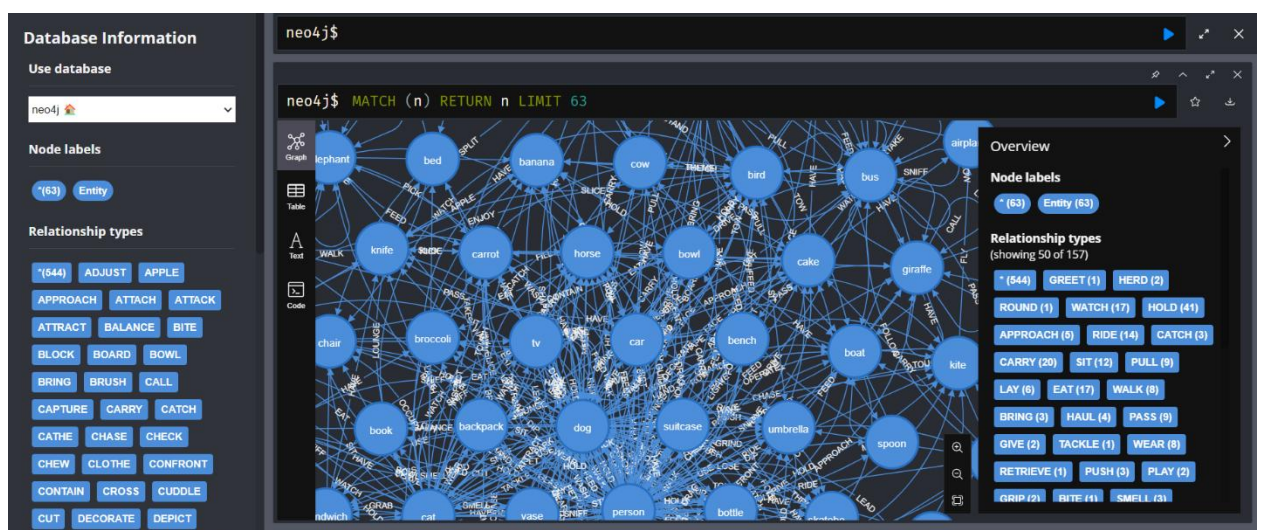
Sau khi đưa 36782 objects và 1650 relationships thì số object và relationship được lưu lại trong Neo4j là 63 objects và 544 relationships

Trong COCO instances_train2017.json, có chỉ 80 categories chính thức (dog, person, frisbee, etc.). Khi trích từ caption, các từ như "man", "person", "boy" có thể bị chuẩn hóa lại thành "person" trước khi đưa vào KG.

→ Vậy nên 36k node đầu vào (từ triplets) có thể chỉ map vào 63 node thực sự (không trùng tên) sau khi merge.

Có 1.650 triplet trong file, nhưng nếu: Nhiều triplet trùng: cùng (subject, relation, object) Hoặc cùng cặp (subject, object) nhưng quan hệ giống nhau.

→ Dẫn đến số quan hệ giảm còn 544



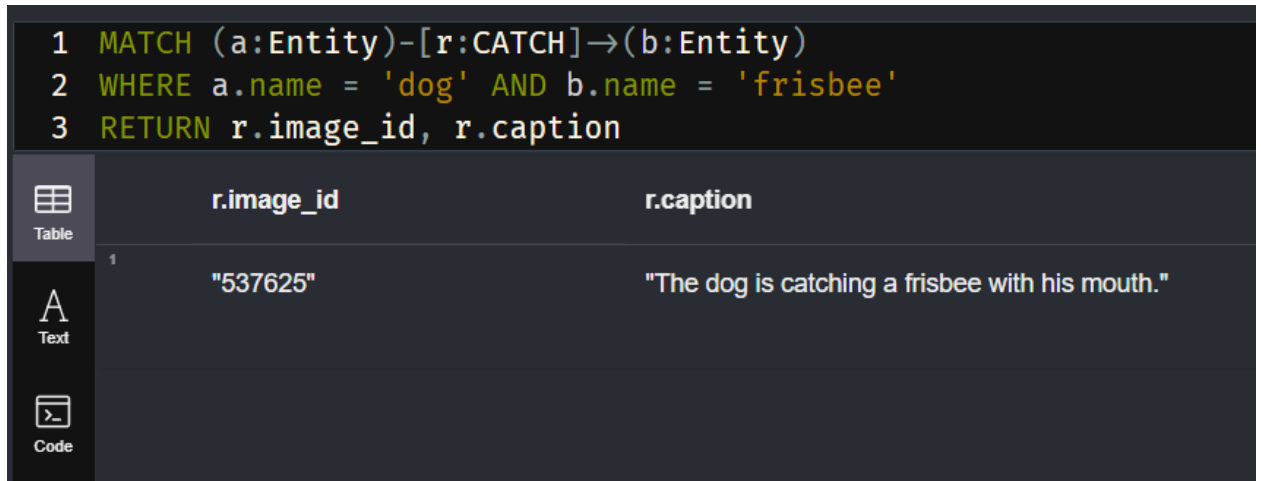
Hình 2.7 Đồ thị tri thức được lưu trong Neo4j

Truy vấn Cypher

MATCH (a:Entity)-[r:CATCH]->(b:Entity)

WHERE a.name = 'dog' AND b.name = 'frisbee'

RETURN r.image_id, r.caption)

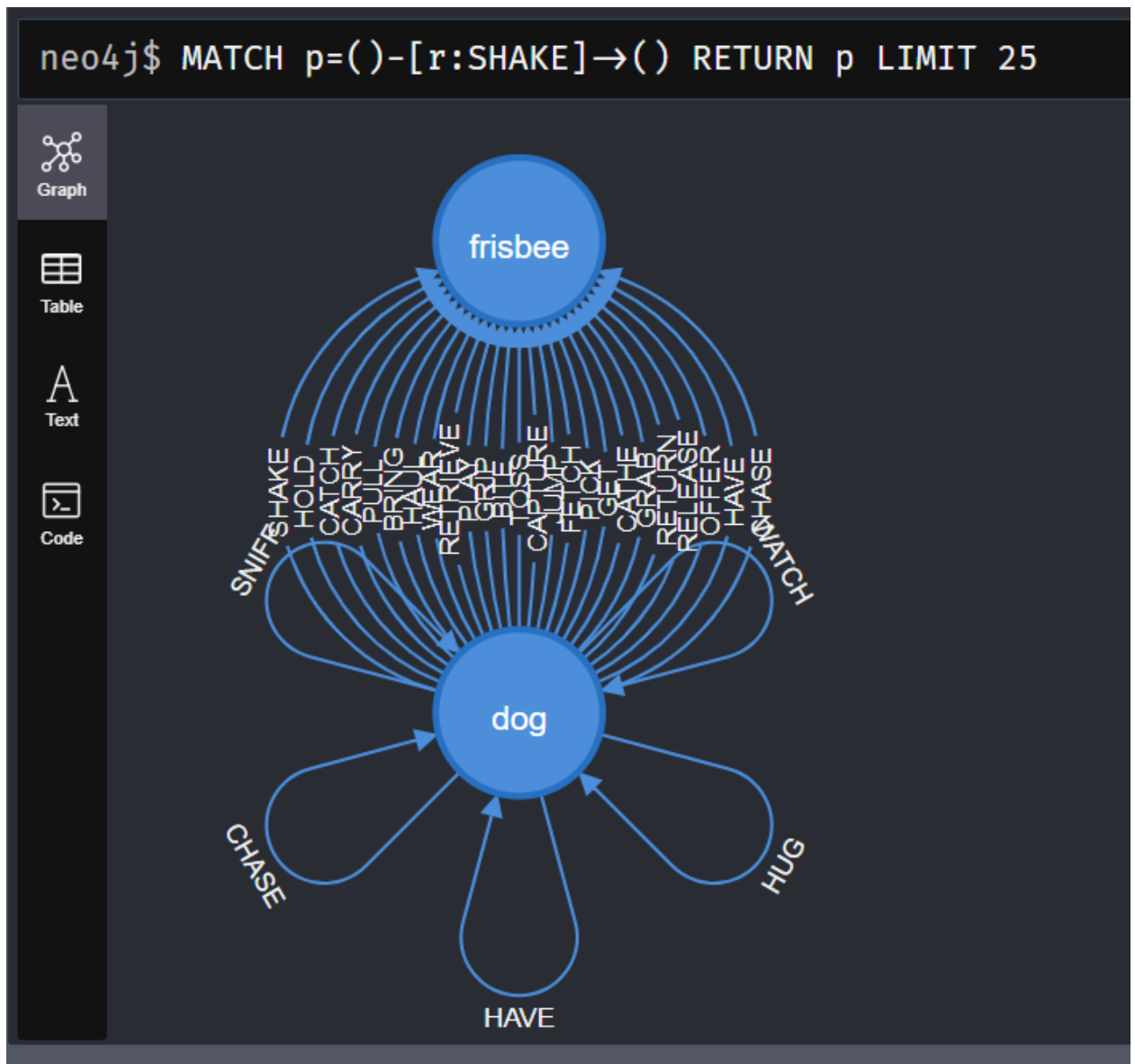


```
1 MATCH (a:Entity)-[r:CATCH]→(b:Entity)
2 WHERE a.name = 'dog' AND b.name = 'frisbee'
3 RETURN r.image_id, r.caption
```

	r.image_id	r.caption
1	"537625"	"The dog is catching a frisbee with his mouth."

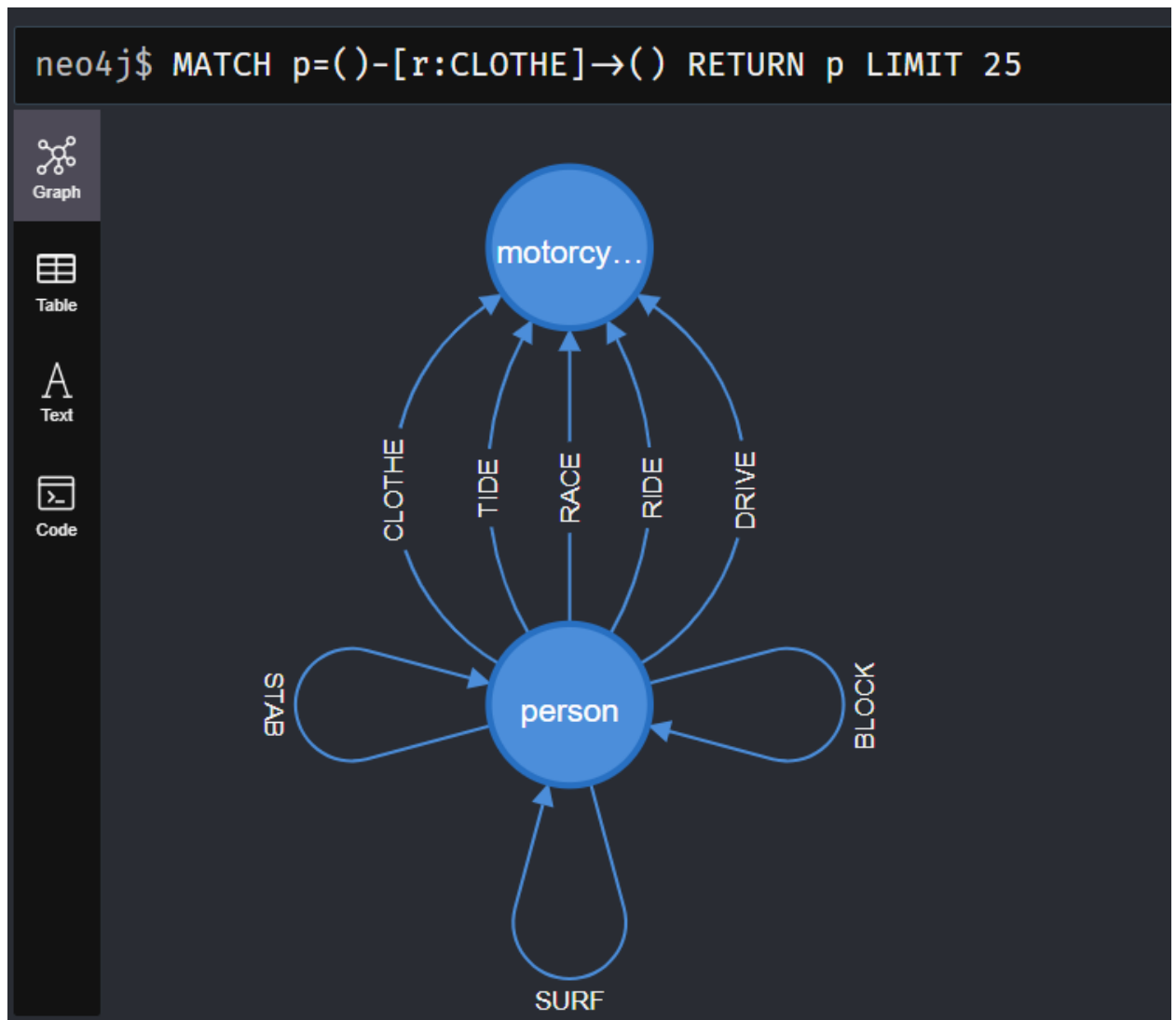
Hình 2.8 Truy vấn Cypher tìm ảnh có entity là “dog” và “frisbee”

MATCH p=()-[r:SHAKE]->() RETURN p LIMIT 25



Hình 2.9 Hiển thị đồ thị sau truy vấn

MATCH p=()-[r:CLOTHE]->() RETURN p LIMIT 25



Hình 2.10 Đồ thị sau khi truy vấn “person” và “motorcycle”

CHƯƠNG 3: XÂY DỰNG THUẬT TOÁN VÀ TRIỂN KHAI HỆ THỐNG

3.1. Mô hình TransE

Mô hình TransE là một phương pháp học biểu diễn (embedding) cho các thành phần trong đồ thị tri thức (entities và relations). Ý tưởng cốt lõi của TransE là ánh xạ mỗi thực thể và quan hệ về không gian vector sao cho quan hệ giữa hai thực thể có thể được mô hình hóa như phép dịch chuyển vector học được. Với mỗi bộ ba (head, relation, tail), TransE cố gắng đảm bảo rằng:

$$h + r \approx t$$

h , r , t lần lượt là vector embedding của subject (head), predicate (relation), và object (tail).

3.1.1. Thư viện sử dụng trong cài đặt mô hình TransE

```
import pandas as pd

import torch

import torch.nn as nn

import torch.nn.functional as F

import random

import numpy as np

from collections import defaultdict
```

- Thư viện *pandas* được sử dụng để đọc và xử lý dữ liệu ở dạng bảng. Cụ thể trong đoạn mã, *pandas* giúp:

- + Đọc tệp .csv chứa các bộ ba (triplet) từ ảnh (f_coco_triplets.csv);
- + Lọc ra các thực thể và quan hệ duy nhất;
- + Chuyển đổi dữ liệu về dạng phù hợp để sử dụng trong PyTorch.

- *NumPy* là thư viện hỗ trợ thao tác mảng và tính toán đại số tuyến tính hiệu quả.

Trong đoạn mã, *numpy* được dùng để:

- + Chuyển đổi danh sách triplet sang mảng số nguyên;
- + Hỗ trợ xử lý batch trong huấn luyện.

- *PyTorch* là thư viện học sâu mã nguồn mở, được sử dụng để xây dựng và huấn luyện mô hình TransE. Các thành phần chính được dùng:

- + *torch.nn*: cung cấp các lớp cơ bản như *nn.Module*, *nn.Embedding* để xây dựng mô hình;

- + *torch.nn.functional*: hỗ trợ các hàm toán học, trong đó có thể dùng các hàm mát mát và chuẩn hóa (nếu cần);

- + *torch*: chứa các hàm tensor, khởi tạo ngẫu nhiên và các phép toán cần thiết cho huấn luyện mạng nơ-ron.

- Thư viện chuẩn *random* của Python có thể được dùng trong bước tạo bộ ba âm (negative sampling), khi cần thay đổi ngẫu nhiên subject hoặc object để sinh các bộ ba sai – bước quan trọng để học được embedding phân biệt.

- *defaultdict* thường dùng để lưu các bộ ba theo từng quan hệ hoặc từng thực thể, phục vụ cho việc sinh dữ liệu hoặc phân tích mạng lưới.

3.1.2. Đọc và tiền xử lý dữ liệu cho mô hình TransE

```
df = pd.read_csv('f_coco_triplets.csv')

entities = pd.unique(df[['subject', 'object']].values.ravel())

relations = df['predicate'].unique()

entity2id = {e: i for i, e in enumerate(entities)}

relation2id = {r: i for i, r in enumerate(relations)}

triplets = df.apply(lambda row: [entity2id[row['subject']],
relation2id[row['predicate']], entity2id[row['object']]], axis=1)

triplets = np.array(list(triplets))
```

Dữ liệu đầu vào được lưu trữ trong tệp *f_coco_triplets.csv*, bao gồm các trường "*subject*", "*predicate*" và "*object*" mô tả mối quan hệ ngữ nghĩa được trích xuất từ ảnh và caption. Việc sử dụng thư viện pandas cho phép đọc và xử lý dữ liệu nhanh chóng dưới dạng DataFrame – cấu trúc dữ liệu dạng bảng rất thuận tiện cho thao tác.

Các thực thể (entities) trong đồ thị tri thức bao gồm tất cả các giá trị xuất hiện ở cả hai cột "*subject*" và "*object*". Lệnh *pd.unique(...ravel())* đảm bảo loại bỏ các trùng

lập, trả về danh sách các thực thể duy nhất. Tương tự, quan hệ (relations) được lấy từ cột "*predicate*".

Mỗi thực thể và quan hệ được gán một chỉ số nguyên duy nhất. Đây là bước mã hóa rất quan trọng để sử dụng với *nn.Embedding* trong PyTorch, vì các lớp embedding yêu cầu chỉ số nguyên làm đầu vào.

Cuối cùng, toàn bộ DataFrame được chuyển đổi thành mảng triplets dưới dạng số nguyên, biểu diễn các bộ ba (*head_id*, *relation_id*, *tail_id*). Mảng này là đầu vào trực tiếp cho mô hình TransE để huấn luyện vector embedding cho từng thực thể và quan hệ.

3.1.3. Xây dựng lớp mô hình TransE

```
class TransE(nn.Module):
```

```
    def __init__(self, num_entities, num_relations, embedding_dim=300, margin=1.0):
```

```
        super(TransE, self).__init__()
```

```
        self.margin = margin
```

```
        self.embedding_dim = embedding_dim
```

```
        self.ent_embeddings = nn.Embedding(num_entities, embedding_dim)
```

```
        self.rel_embeddings = nn.Embedding(num_relations, embedding_dim)
```

```
        nn.init.xavier_uniform_(self.ent_embeddings.weight.data)
```

```
        nn.init.xavier_uniform_(self.rel_embeddings.weight.data)
```

```
    def forward(self, pos_triples, neg_triples):
```

```
        pos_h = self.ent_embeddings(pos_triples[:, 0])
```

```
        pos_r = self.rel_embeddings(pos_triples[:, 1])
```

```
        pos_t = self.ent_embeddings(pos_triples[:, 2])
```

```
        neg_h = self.ent_embeddings(neg_triples[:, 0])
```

```
        neg_r = self.rel_embeddings(neg_triples[:, 1])
```

```
        neg_t = self.ent_embeddings(neg_triples[:, 2])
```

```
        pos_score = torch.norm(pos_h + pos_r - pos_t, p=2, dim=1)
```

```
neg_score = torch.norm(neg_h + neg_r - neg_t, p=2, dim=1)

return pos_score, neg_score
```

Ý tưởng chính của mô hình là coi mối quan hệ giữa hai thực thể như một phép dịch chuyển vector trong không gian embedding.

Lớp TransE kế thừa từ `nn.Module` trong thư viện PyTorch bao gồm bốn tham số khởi tạo sau:

- + *num_entities*: số lượng thực thể trong đồ thị;
- + *num_relations*: số lượng quan hệ;
- + *embedding_dim*: số chiều không gian biểu diễn vector;
- + *margin*: hệ số biên (margin) dùng trong hàm mất mát.

Hai lớp embedding được khởi tạo cho thực thể và quan hệ, cho phép ánh xạ từng thực thể và quan hệ về một vector $\in \mathbb{R}^d$. Các vector này được tối ưu trong quá trình huấn luyện để mô hình hóa tri thức dạng $h + r \approx t$.

Phép khởi tạo *Xavier Uniform* giúp đảm bảo sự lan truyền tín hiệu ổn định trong mạng nơ-ron, đặc biệt hữu ích cho các lớp embedding có số chiều cao.

Hàm forward định nghĩa logic tính toán cho một lần lan truyền của mô hình, nhận vào như sau:

- + *pos_triples*: bộ ba đúng (từ dữ liệu gốc);
- + *neg_triples*: bộ ba sai (do người dùng sinh ra thông qua kỹ thuật negative sampling).

Các chỉ số trong *pos_triples* được dùng để tra cứu embedding vector cho từng thành phần: *head* (*h*), *relation* (*r*) và *tail* (*t*). Tương tự cho các bộ ba sai (*neg_triples*).

Khoảng cách giữa $h + r$ và t được tính bằng chuẩn L2 (Euclidean distance). Mô hình TransE được thiết kế để tối thiểu hóa khoảng cách này cho các bộ ba đúng, đồng thời tối đa hóa khoảng cách cho bộ ba sai.

Hàm *forward* trả về hai tensor đại diện cho độ lệch (score) của các bộ ba đúng và sai. Kết quả này sẽ được dùng trong bước huấn luyện để tính hàm mất mát *margin-based ranking loss*:

$$L = \sum_{\text{triplet}} \max(0, \gamma + d(h + r, t) - d(h' + r', t'))$$

Trong đó γ là biên margin, d là khoảng cách L2, và (h', r', t') là bộ ba sai.

3.1.4. Sinh dữ liệu âm (Negative Sampling)

```
def corrupt_batch(batch, num_entities):
    corrupted = []
    for h, r, t in batch:
        if random.random() < 0.5:
            h_ = random.randint(0, num_entities - 1)
            corrupted.append([h_, r, t])
        else:
            t_ = random.randint(0, num_entities - 1)
            corrupted.append([h, r, t_])
    return corrupted
```

Trong quá trình huấn luyện mô hình TransE, mục tiêu là làm cho các bộ ba (triplets) đúng trong đồ thị tri thức có khoảng cách nhỏ (embedding hợp lý), đồng thời các bộ ba sai phải có khoảng cách lớn. Tuy nhiên, dữ liệu gốc thường chỉ chứa các bộ ba đúng. Để mô hình học được ranh giới giữa "đúng" và "sai", ta cần sinh *bộ ba âm* (negative triplets) thông qua kỹ thuật gọi là *negative sampling*.

- Hàm nhận đầu vào là:
 - + *batch*: danh sách các bộ ba đúng (dạng $[h, r, t]$);
 - + *num_entities*: tổng số thực thể trong đồ thị.
- Với mỗi bộ ba đúng (h, r, t) trong batch, hàm sẽ sinh ra một bộ ba sai bằng cách:
 - + 50% xác suất thay thế thực thể đầu (head): tạo bộ ba (h', r, t) với $h' \neq h$
 - + 50% xác suất còn lại thay thế thực thể cuối (tail): tạo (h, r, t') với $t' \neq t$

Quá trình lựa chọn ngẫu nhiên này đảm bảo mô hình không bị "học vẹt" mà phải phân biệt được giữa các thực thể và quan hệ có ý nghĩa thực sự.

Vai trò trong huấn luyện: Các bộ ba âm sinh ra từ hàm này sẽ được dùng cùng với bộ ba đúng trong hàm lan truyền (*forward*) của mô hình TransE, từ đó tính toán *margin-based loss* theo công thức:

$$L = \sum_{\text{triplet}} \max(0, \gamma + d(h + r, t) - d(h^1 + r', t'))$$

3.1.5. Huấn luyện mô hình TransE

```
def train_transe(model, train_data, num_entities, num_epochs=300, batch_size=512,
lr=0.001, patience=10):
```

```
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

```
    criterion = nn.MarginRankingLoss(margin=model.margin)
```

```
    best_loss = float('inf')
```

```
    wait = 0
```

```
    model.train()
```

```
    for epoch in range(num_epochs):
```

```
        total_loss = 0
```

```
        np.random.shuffle(train_data)
```

```
        for i in range(0, len(train_data), batch_size):
```

```
            batch_pos = train_data[i:i+batch_size]
```

```
            batch_neg = corrupt_batch(batch_pos, num_entities)
```

```
            batch_pos = torch.tensor(batch_pos, dtype=torch.long)
```

```
            batch_neg = torch.tensor(batch_neg, dtype=torch.long)
```

```
            y = torch.ones(batch_pos.size(0))
```

```
            pos_score, neg_score = model(batch_pos, batch_neg)
```

```
            loss = criterion(pos_score, neg_score, y)
```

```

optimizer.zero_grad()

loss.backward()

torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)

optimizer.step()

total_loss += loss.item()

avg_loss = total_loss / (len(train_data) / batch_size)

print(f"Epoch {epoch+1} | Avg Loss: {avg_loss:.4f}")

# Early stopping

if avg_loss < best_loss:

    best_loss = avg_loss

    wait = 0

    torch.save(model.state_dict(), "best_transe_model.pth")

else:

    wait += 1

    if wait >= patience:

        print(f"Early stopping at epoch {epoch+1} (best loss: {best_loss:.4f})")

        break

```

Quá trình huấn luyện mô hình TransE được thực hiện thông qua hàm *train_transe*, kết hợp việc tạo dữ liệu âm, lan truyền tiến (*forward*), tối ưu hóa hàm mất mát và chiến lược dừng sớm (*early stopping*). Mục tiêu của huấn luyện là điều chỉnh vector embedding của thực thể và quan hệ sao cho các bộ ba đúng có khoảng cách nhỏ hơn các bộ ba sai trong không gian embedding.

Trình tối ưu được sử dụng là *Adam*, một phương pháp phổ biến và hiệu quả trong huấn luyện mạng nơ-ron.

Hàm mất mát là *MarginRankingLoss*, thực hiện mục tiêu tối thiểu hóa khoảng cách bộ ba đúng và tối đa hóa khoảng cách bộ ba sai theo công thức:

$$L = \sum_{\text{triplet}} \max(0, \gamma + d(h + r, t) - d(h^1 + r', t'))$$

Mỗi vòng epoch là một lượt huấn luyện toàn bộ tập dữ liệu. Trước mỗi epoch, dữ liệu được xáo trộn ngẫu nhiên (shuffling) để tránh hiện tượng overfitting dữ liệu.

Dữ liệu huấn luyện được chia thành các mini-batch để tăng hiệu suất tính toán. Với mỗi batch, tạo ra tập negative samples bằng hàm `corrupt_batch` đã định nghĩa trước đó. Các batch được chuyển sang tensor PyTorch.

Vector y chứa toàn số 1, biểu thị rằng mô hình cần phân biệt rõ ràng bộ ba đúng (dấu +) và bộ ba sai (dấu -).

Hàm lan truyền của mô hình TransE trả về hai vector khoảng cách (score) cho batch đúng và batch sai. Hàm mất mát *MarginRankingLoss* sẽ đánh giá sự khác biệt này.

Lan truyền ngược (*backward*) và cập nhật trọng số (*step*) được thực hiện mỗi batch. Sử dụng kỹ thuật *gradient clipping* để giới hạn độ lớn đạo hàm, tránh hiện tượng *exploding gradients*.

Nếu không cải thiện trong nhiều epoch liên tiếp (theo *patience*), vòng huấn luyện sẽ dừng sớm để tiết kiệm thời gian và tránh overfitting.

3.2. Mô hình GCN

Trong phương pháp sử dụng Graph Convolutional Network (GCN) để biểu diễn tri thức hình ảnh, bước đầu tiên là biểu diễn dữ liệu tri thức dưới dạng đồ thị có hướng. Mỗi triplet (subject, predicate, object) được ánh xạ thành một cấu trúc đồ thị phản ánh quan hệ ngữ nghĩa giữa các thực thể, đồng thời lưu thông tin ảnh chứa quan hệ đó.

3.2.1. Thư viện sử dụng trong cài đặt mô hình GCN

```
import dgl

from dgl.nn import GraphConv

from sklearn.metrics.pairwise import cosine_similarity

from collections import defaultdict
```

- *DGL (Deep Graph Library)* là thư viện học sâu chuyên biệt cho dữ liệu đồ thị, được sử dụng để biểu diễn, xử lý và huấn luyện các mô hình như GCN. Các thành phần chính được dùng:

- + *dgl.graph*: cho phép khởi tạo đồ thị có hướng từ danh sách các cạnh, hỗ trợ hiệu quả trên GPU;
- + *dgl.add_self_loop*: thêm các cạnh tự nối (self-loop) để đảm bảo mỗi node có thể giữ lại thông tin của chính nó trong quá trình lan truyền;
- + *dgl.DGLGraph*: là cấu trúc đồ thị được tối ưu để sử dụng trong mô hình học sâu, đặc biệt là với PyTorch.

- *DGL.nn.GraphConv* là mô-đun lan truyền thông tin giữa những node trong mô hình GCN. Trong đó có *GraphConv*: là lớp tích chập đồ thị, thực hiện phép lan truyền và tổng hợp thông tin từ các node lân cận theo trọng số đã học, từ đó cập nhật embedding cho mỗi node qua các lớp GCN.

- *Scikit-learn (sklearn.metrics.pairwise)* là thư viện học máy phổ biến được dùng để tính toán độ tương đồng giữa các vector. Trong đó *cosine_similarity*: được sử dụng để tính độ tương đồng giữa embedding của ảnh truy vấn và các ảnh còn lại trong tập dữ liệu, giúp thực hiện việc truy vấn ảnh tương tự theo ngữ nghĩa.

- *Collections (collections.defaultdict)* là mô-đun của Python giúp xử lý dữ liệu có cấu trúc dạng từ điển. Trong đó *defaultdict(list)*: cho phép gom tất cả các node có cùng *image_id* một cách hiệu quả, phục vụ cho việc tính vector trung bình của ảnh.

3.2.2. Đọc và tiền xử lý dữ liệu cho mô hình GCN

```
df = pd.read_csv('f_coco_triplets.csv')
```

```
df = df.dropna()
```

```
df = df.astype(str)
```

```
triplets = list(zip(df['subject'], df['predicate'], df['object'], df['image_id']))
```

Tập dữ liệu bao gồm các bộ ba (*subject*, *predicate*, *object*) kèm theo *image_id*, mô tả mối quan hệ được trích xuất từ ảnh và caption. Loại bỏ giá trị rỗng (NaN) để đảm bảo độ đầy đủ của đồ thị.

Ép kiểu tất cả các giá trị về chuỗi (string), giúp xử lý thống nhất các thực thể và quan hệ. Tạo danh sách triplets chứa các tuple dạng (*subject*, *predicate*, *object*, *image_id*) để xây dựng đồ thị tri thức.

3.2.3. Chuyển đổi và chuẩn bị dữ liệu đầu vào cho mô hình GCN bằng DGL

```
all_nodes = list(G.nodes)

le = LabelEncoder()

node_ids = le.fit_transform(all_nodes)

node_id_map = dict(zip(all_nodes, node_ids))

edges_src = [node_id_map[u] for u, v in G.edges()]
edges_dst = [node_id_map[v] for u, v in G.edges()]

g = dgl.graph((edges_src, edges_dst))

g = dgl.add_self_loop(g)

features = torch.eye(len(all_nodes))
```

Chuẩn hóa và mã hóa dữ liệu để làm đầu vào cho mô hình GCN. Trong bước này, sử dụng DGL (Deep Graph Library) để định nghĩa và xử lý đồ thị, đồng thời tạo đặc trưng ban đầu của các node trong đồ thị G được trích xuất dưới dạng danh sách.

Sử dụng *LabelEncoder* từ *sklearn* để ánh xạ tên node (chuỗi) sang ID số nguyên liên tục – đây là định dạng bắt buộc để khởi tạo đồ thị trong DGL. Kết quả là từ điển *node_id_map* ánh xạ tên node \rightarrow ID.

Duyệt qua tất cả các cạnh trong đồ thị ban đầu (*networkx*) để lấy danh sách nguồn (*src*) và đích (*dst*). DGL yêu cầu biểu diễn đồ thị theo dạng tuple (*src_nodes*, *dst_nodes*). Kết quả là đồ thị g dạng *dgl.DGLGraph* có hướng, sẵn sàng cho huấn luyện GCN.

GCN cần mỗi node có thể giữ lại thông tin ban đầu của chính nó trong quá trình lan truyền. Do đó, thêm self-loop là một bước bắt buộc để tránh các node bị "trắng thông tin" nếu không có cạnh vào. Điều này đặc biệt quan trọng nếu đồ thị có các node biên hoặc bị cô lập (dangling nodes).

Mỗi node ban đầu được biểu diễn bằng một vector one-hot, tức là vector có đúng 1 giá trị bằng 1 tại vị trí tương ứng với ID node.

Kích thước ma trận đặc trưng: ($số_node$, $số_node$). Đây là đặc trưng phi học (non-trainable), và được mô hình GCN cập nhật thông qua lan truyền (message passing).

3.2.4. Mô hình Graph Convolutional Network (GCN)

```
class GCN(nn.Module):  
  
    def __init__(self, in_feats, h_feats, out_feats):  
        super(GCN, self).__init__()  
        self.conv1 = GraphConv(in_feats, h_feats)  
        self.conv2 = GraphConv(h_feats, out_feats)  
  
    def forward(self, g, in_feat):  
        h = self.conv1(g, in_feat)  
        h = torch.relu(h)  
        h = self.conv2(g, h)  
        return h  
  
model = GCN(features.shape[1], 64, 32)  
model.eval()  
  
with torch.no_grad():  
    node_embeddings = model(g, features)
```

Graph Convolutional Network (GCN) là một kiến trúc học sâu được thiết kế để học đặc trưng từ dữ liệu có cấu trúc đồ thị. Khác với các mạng nơ-ron truyền thống, GCN khai thác cấu trúc liên kết giữa các node để lan truyền và tổng hợp thông tin. Trong bài toán biểu diễn tri thức từ ảnh, GCN cho phép học các vector embedding cho thực thể và quan hệ bằng cách truyền thông tin qua các cạnh ngữ nghĩa.

- Mô hình gồm hai lớp convolution trên đồ thị:
 - + *conv1*: ánh xạ đặc trưng đầu vào (*input feature*) sang không gian ẩn (*hidden layer*);
 - + *conv2*: ánh xạ tiếp từ không gian ẩn sang vector embedding đầu ra.

- *GraphConv* là lớp lan truyền thông tin giữa các node thông qua các cạnh có hướng của đồ thị.

- Trong hàm *forward*, đầu tiên thực hiện phép lan truyền thông tin qua lớp thứ nhất, áp dụng hàm kích hoạt phi tuyến ReLU, sau đó tiếp tục lan truyền thông tin qua lớp thứ hai để thu được embedding đầu ra cho mỗi node.

- Khởi tạo mô hình với:

+ *features.shape[1]*: số chiều của đặc trưng đầu vào mô hình (mỗi node có vector one-hot)

+ 64: số chiều lớp ẩn

+ 32: số chiều embedding đầu ra (có thể được sử dụng để so khớp ảnh)

- Mô hình được đặt trong chế độ đánh giá (*eval()*), và phép lan truyền được thực hiện trong khối *torch.no_grad()* để tiết kiệm bộ nhớ và tránh tính đạo hàm khi không huấn luyện.

- Kết quả *node_embeddings* là một tensor kích thước (*số node*, 32), biểu diễn vector embedding cho mỗi thực thể và quan hệ trung gian trong đồ thị tri thức.

3.2.5. Tổng hợp vector embedding cho ảnh

```
embeddings = node_embeddings.numpy()
```

```
image_embeddings = defaultdict(list)
```

```
for node, idx in node_id_map.items():
```

```
    if 'image_id' in G.nodes[node]:
```

```
        img_id = G.nodes[node]['image_id']
```

```
        image_embeddings[img_id].append(embeddings[idx])
```

```
# Trung bình embedding cho mỗi ảnh
```

```
for img in image_embeddings:
```

```
    vecs = np.stack(image_embeddings[img])
```

```
    image_embeddings[img] = np.mean(vecs, axis=0)
```

Các node trung gian trong đồ thị (*dạng s p o*) chứa thông tin *image_id*, đại diện cho ảnh mà triplet đó được trích xuất. Do đó, để tạo vector biểu diễn cho mỗi ảnh, ta tiến hành gom tất cả embedding tương ứng với các node có cùng *image_id*:

- + *node_embeddings* là output của GCN, chứa embedding cho tất cả node trong đồ thị.

- + *image_embeddings* là một từ điển chứa các vector được nhóm theo từng ảnh.

Với mỗi node trung gian, nếu node chứa *image_id*, ta lưu embedding của node đó vào danh sách các vector của ảnh tương ứng.

Sau khi thu thập xong các vector, mỗi ảnh được gán một vector embedding trung bình (*mean pooling*). Cách tổng hợp này giúp kết hợp thông tin từ nhiều triplet để tạo ra một biểu diễn duy nhất, giàu ngữ nghĩa.

3.2.6. Truy vấn ảnh tương tự bằng khoảng cách cosine

```
def find_similar_images(query_img_id, top_k=5):  
    if query_img_id not in image_embeddings:  
        return []  
    query_vec = image_embeddings[query_img_id].reshape(1, -1)  
    all_img_ids = list(image_embeddings.keys())  
    all_vecs = np.stack([image_embeddings[img] for img in all_img_ids])  
    sims = cosine_similarity(query_vec, all_vecs)[0]  
    top_indices = sims.argsort()[-top_k:][::-1]  
    return [(all_img_ids[i], float(sims[i])) for i in top_indices if all_img_ids[i] !=  
            query_img_id]
```

Khi đã có embedding cho tất cả các ảnh, ta có thể so sánh độ tương đồng giữa các ảnh dựa trên khoảng cách *cosine* – thước đo phổ biến trong các bài toán nhúng vector, đặc biệt hữu ích trong không gian chiều cao.

Sử dụng hàm *cosine_similarity* từ thư viện *scikit-learn* để tính độ tương đồng giữa vector truy vấn và các vector ảnh khác.

Tìm vector ảnh tương ứng với *query_img_id*, rồi chuẩn bị danh sách tất cả vector ảnh khác để tính điểm tương đồng.

argsort() sắp xếp các điểm tương đồng tăng dần \rightarrow lấy *top_k* điểm cao nhất (tức là ảnh tương tự nhất)

Kết quả trả về là danh sách *top_k* ảnh tương tự (loại trừ chính ảnh truy vấn), kèm theo điểm tương đồng cosine.

3.3. Mô hình R-GCN

Relational Graph Convolutional Network (R-GCN) là một mở rộng của GCN cho các đồ thị dị thể, nơi các cạnh (edges) mang nhiều loại quan hệ khác nhau. Trái ngược với GCN thông thường – chỉ truyền tín hiệu qua cấu trúc đồ thị, R-GCN học cách lan truyền thông tin phân biệt theo từng loại quan hệ.

Để xây dựng một R-GCN hiệu quả, bước đầu tiên là xây dựng đồ thị tri thức có hướng từ các bộ ba triplet, sau đó mã hóa node và quan hệ thành chỉ số số nguyên để dùng cho huấn luyện.

3.3.1. Thư viện sử dụng trong cài đặt mô hình R-GCN

```
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.metrics import precision_score, recall_score, f1_score
```

```
from sklearn.metrics.pairwise import cosine_similarity
```

- *LabelEncoder* là công cụ chuyển đổi các nhãn phân loại dạng chuỗi (string) thành dạng số nguyên liên tục. Tất cả các tên node (có thể là subject, object hoặc node quan hệ) cần được mã hóa sang chỉ số nguyên để phù hợp với yêu cầu đầu vào của mô hình học sâu và thư viện DGL.

- *precision_score*, *recall_score*, *f1_score* là ba thước đo quan trọng dùng để đánh giá độ chính xác của hệ thống phân loại hoặc truy vấn. Các thước đo này được dùng để so sánh danh sách ảnh truy xuất so với ground truth ảnh liên quan, đánh giá độ hiệu quả của embedding học được.

- *cosine_similarity* tính toán độ tương đồng cosine giữa các vector – phản ánh mức độ tương tự về hướng trong không gian đa chiều. Sau khi embedding ảnh được

tính, cosine similarity được dùng để đo độ tương đồng giữa ảnh truy vấn và các ảnh còn lại, từ đó truy xuất ảnh có nội dung ngữ nghĩa tương đồng.

3.3.2. Đọc và tiền xử lý dữ liệu cho mô hình R-GCN

```
df = pd.read_csv('f_coco_triplets.csv')

df = df.dropna().astype(str)

triplets = list(zip(df['subject'], df['predicate'], df['object'], df['image_id']))

G = nx.DiGraph()

for s, p, o, img_id in triplets:

    mid = f'{s}_{p}_{o}'

    G.add_edge(s, mid, relation=p)

    G.add_edge(mid, o, relation=p)

    G.nodes[mid]['image_id'] = img_id

all_nodes = list(G.nodes)

le = LabelEncoder()

node_ids = le.fit_transform(all_nodes)

node_id_map = dict(zip(all_nodes, node_ids))

edges_src = [node_id_map[u] for u, v in G.edges()]

edges_dst = [node_id_map[v] for u, v in G.edges()]

edge_types = [G[u][v]['relation'] for u, v in G.edges()]

rel_encoder = LabelEncoder()

edge_type_ids = rel_encoder.fit_transform(edge_types)

edge_type_tensor = torch.tensor(edge_type_ids, dtype=torch.int64)
```

- Tập dữ liệu COCO được tổ chức dưới dạng các triplet (*subject*, *predicate*, *object*) trích xuất từ caption của ảnh. Mỗi triplet đi kèm *image_id*, biểu thị ảnh mà triplet đó xuất hiện.

- Mỗi triplet được biểu diễn bằng một node trung gian (*mid*) để phản ánh bản chất "dị thể" của đồ thị: thực thể (*entity*) và quan hệ (*relation*) được tách riêng.

- Việc ánh xạ quan hệ thành node trung gian (giữa *subject* và *object*) là một kỹ thuật giúp đồ thị phù hợp với *R-GCN*, nơi mà các loại quan hệ đóng vai trò quan trọng trong truyền tín hiệu.

- Tất cả các node (bao gồm cả *entity* và node quan hệ trung gian) được gán một chỉ số nguyên bằng *LabelEncoder*.

- Việc mã hóa này đảm bảo đồ thị được biểu diễn đúng trong *DGL* hoặc *PyTorch Geometric*, nơi node *ID* phải là *int*.

- *edges_src* và *edges_dst* là danh sách các cặp node ($u \rightarrow v$) đã được mã hóa *ID*.

- *edge_types* lưu giữ loại quan hệ *p* gắn với mỗi cạnh – rất quan trọng trong mô hình *R-GCN* vì *R-GCN* sử dụng ma trận trọng số riêng cho từng loại quan hệ.

- Các quan hệ được mã hóa thành số nguyên liên tiếp ($0, 1, 2, \dots, R-1$) để dùng làm chỉ số *relation-type* trong mô hình.

- *edge_type_tensor* sẽ được cung cấp vào lớp *RGCNConv* như tham số *edge_type*, cho phép mô hình lan truyền ngữ nghĩa phân biệt theo từng loại quan hệ.

3.3.3. Xây dựng đồ thị DGL cho mô hình R-GCN

```
g = dgl.graph((edges_src, edges_dst), num_nodes=len(all_nodes))
```

```
g = dgl.add_self_loop(g)
```

```
num_self_loops = g.number_of_edges() - len(edge_type_tensor)
```

```
self_loop_type = torch.full((num_self_loops,),  
fill_value=len(set(edge_type_ids)), dtype=torch.int64)
```

```
edge_type_tensor = torch.cat([edge_type_tensor, self_loop_type], dim=0)
```

```
features = torch.eye(len(all_nodes))
```

```
num_rels = len(set(edge_type_tensor.tolist()))
```

Sử dụng *dgl.graph()* để khởi tạo đồ thị có hướng từ danh sách cặp (*source_node*, *destination_node*) đã mã hóa. Tham số *num_nodes* đảm bảo tất cả các node (bao gồm cả node không có cạnh vào/ra) đều được đưa vào đồ thị.

Trong R-GCN cũng như GCN, *self-loop* (cạnh tự nối) rất quan trọng để cho phép node giữ lại thông tin ban đầu trong quá trình lan truyền. Khi có *self-loop*, mỗi node sẽ nhận thông tin từ chính nó ngoài việc nhận từ các lân cận.

Sau khi thêm *self-loop*, số lượng cạnh đã tăng thêm. Để mô hình R-GCN xử lý đúng, ta cần chỉ định loại quan hệ cho những *self-loop* này. *Self-loop* được gán một loại quan hệ mới, cụ thể là *relation_id = số lượng quan hệ ban đầu*. *edge_type_tensor* được mở rộng để thêm các *self-loop* này vào đúng vị trí của cạnh.

Mỗi node được biểu diễn bằng *vector one-hot*, trong đó kích thước vector bằng số lượng node.

Đây là đặc trưng phi huấn luyện (*non-trainable*), và sẽ được mô hình cập nhật dần qua các lớp lan truyền của *R-GCN*.

Số lượng quan hệ (*num_rels*) bao gồm cả các quan hệ gốc và *một loại quan hệ đặc biệt dành cho self-loop*.

Giá trị này sẽ được truyền vào mô hình *R-GCN* như một siêu tham số để định nghĩa số lượng ma trận biến đổi W_r trong lớp *RelGraphConv*.

3.3.4. Xây dựng mô hình Relational Graph Convolutional Network (R-GCN)

```
class RGCN(nn.Module):
```

```
    def __init__(self, in_feats, h_feats, out_feats, num_rels):
```

```
        super(RGCN, self).__init__()
```

```
        self.conv1 = dgl.nn.RelGraphConv(in_feats, h_feats, num_rels)
```

```
        self.conv2 = dgl.nn.RelGraphConv(h_feats, out_feats, num_rels)
```

```
    def forward(self, g, x, etype):
```

```
        h = self.conv1(g, x, etype)
```

```
        h = torch.relu(h)
```

```
        h = self.conv2(g, h, etype)
```

```
        return h
```

```
model = RGCN(features.shape[1], 64, 32, num_rels)
```

```
model.eval()
```

```
with torch.no_grad():
```

```
node_embeddings = model(g, features, edge_type_tensor)
```

R-GCN là một biến thể mở rộng của Graph Convolutional Network (GCN), thiết kế đặc biệt cho đồ thị dị thể (heterogeneous graphs) với nhiều loại quan hệ (multi-relational). Thay vì sử dụng một hàm lan truyền duy nhất cho mọi cạnh, R-GCN học một tập các ma trận chuyển đổi riêng biệt theo từng loại quan hệ.

- + *in_feats*: số chiều đặc trưng đầu vào (với one-hot, thường bằng số lượng node).
- + *h_feats*: số chiều của lớp ẩn (hidden layer).
- + *out_feats*: số chiều đầu ra – thường là kích thước vector embedding cho node.
- + *num_rels*: tổng số loại quan hệ trong đồ thị, bao gồm cả self-loop.

Mỗi lớp *RelGraphConv* trong DGL được định nghĩa để học các phép lan truyền thông tin theo từng loại quan hệ.

Với mỗi loại quan hệ r , có một ma trận chuyển đổi W_r , và công thức lan truyền cơ bản là:

$$\vec{h}_i^{(l+1)} = \sigma \left(\sum_{r \in R} \sum_{j \in N_r(i)} \frac{1}{c_{i,r}} W_r^{(l)} \vec{h}_j^{(l)} + W_0^{(l)} \vec{h}_i^{(l)} \right)$$

- Hàm lan truyền forward:

- + g : đồ thị DGL đã xây dựng;
- + x : đặc trưng ban đầu của node (*features*);
- + *etype*: tensor chứa chỉ số loại quan hệ tương ứng với từng cạnh.
- + Hai lớp R-GCN được xếp chồng, xen giữa là hàm kích hoạt *ReLU*.
- + Hàm trả về embedding cuối cùng cho từng node trong đồ thị.

- Tính toán embedding:

- + Kích thước đầu vào = số node (do sử dụng one-hot),

+ Lớp ẩn = 64 chiều,

+ Đầu ra = 32 chiều (embedding cho từng node).

Mô hình được đưa vào chế độ đánh giá (*eval*) và lan truyền tính toán embedding không tính gradient (*no_grad*) để tiết kiệm bộ nhớ.

Kết quả *node_embeddings* có kích thước (*số node*, 32), chứa vector biểu diễn ngữ nghĩa cho mọi node (bao gồm thực thể và node quan hệ trung gian).

3.3.5. Truy vấn ảnh tương tự bằng vector embedding

```
def find_similar_images(query_img_id, top_k=5):  
    if query_img_id not in image_embeddings:  
        return []  
  
    query_vec = image_embeddings[query_img_id].reshape(1, -1)  
    all_img_ids = list(image_embeddings.keys())  
    all_vecs = np.stack([image_embeddings[img] for img in all_img_ids])  
    sims = cosine_similarity(query_vec, all_vecs)[0]  
    top_indices = sims.argsort()[-top_k:][::-1]  
    return [all_img_ids[i] for i in top_indices if all_img_ids[i] != query_img_id]
```

Các vector này được tổng hợp lại để tính embedding đại diện cho mỗi ảnh (bằng cách trung bình các node có cùng *image_id*). Từ đó, ta có thể xây dựng một hệ thống tìm kiếm ảnh dựa trên tri thức, nơi truy vấn ảnh được thực hiện trong không gian ngữ nghĩa vector.

Kiểm tra xem ảnh truy vấn (*query_img_id*) có tồn tại trong tập *image_embeddings* không. Nếu không có vector tương ứng, hàm trả về rỗng có nghĩa là không tìm được ảnh tương tự.

Vector embedding của ảnh truy vấn được đưa về dạng ma trận 2 chiều. *all_vecs*: ma trận chứa tất cả các vector ảnh trong cơ sở dữ liệu.

Sử dụng *cosine similarity* để đo độ tương đồng giữa vector ảnh truy vấn và tất cả các vector ảnh còn lại. Hàm *cosine_similarity* trả về một mảng *sims* với giá trị từ -1 đến

1, trong đó giá trị càng cao thì hai vector càng đồng hướng → ảnh càng giống nhau về mặt ngữ nghĩa.

`argsort()` sắp xếp chỉ số ảnh theo thứ tự tăng dần điểm tương đồng → lấy `top_k` ảnh có độ tương đồng cao nhất. Loại bỏ chính ảnh truy vấn ra khỏi kết quả. Trả về danh sách các `image_id` tương tự nhất với ảnh truy vấn theo không gian embedding học được.

3.3.6. Truy xuất các ảnh liên quan và đánh giá

```
def estimate_prf(top_k=5, random_k=5, sample_size=100):
```

```
    img_ids = list(image_embeddings.keys())
```

```
    precisions, recalls, f1s = [], [], []
```

```
    sampled_queries = random.sample(img_ids, min(sample_size, len(img_ids)))
```

```
    for query_img_id in sampled_queries:
```

```
        predicted = find_similar_images(query_img_id, top_k)
```

```
        if not predicted:
```

```
            continue
```

```
        negatives = [img for img in img_ids if img not in predicted and img !=  
query_img_id]
```

```
        if len(negatives) < random_k:
```

```
            continue
```

```
        random_negatives = random.sample(negatives, random_k)
```

```
        y_true = [1] * len(predicted) + [0] * len(random_negatives)
```

```
        y_pred = [1] * (len(predicted) + len(random_negatives))
```

```
        precisions.append(precision_score(y_true, y_pred, zero_division=0))
```

```
        recalls.append(recall_score(y_true, y_pred, zero_division=0))
```

```
        f1s.append(f1_score(y_true, y_pred, zero_division=0))
```

```
    if not precisions:
```

```
        return {"Precision": 0.0, "Recall": 0.0, "F1-score": 0.0, "Samples Used": 0}
```

```

return {
    "Precision": round(np.mean(precisions), 4),
    "Recall": round(np.mean(recalls), 4),
    "F1-score": round(np.mean(f1s), 4),
    "Samples Used": len(precisions)
}

```

Truy vấn ảnh theo embedding học được từ đồ thị tri thức, ta cần đo lường khả năng truy xuất đúng các ảnh liên quan (*true positives*), đồng thời tránh nhầm lẫn với các ảnh không liên quan (*false positives*). Hàm *estimate_prf* được thiết kế nhằm mục tiêu đó, sử dụng ba chỉ số phổ biến: Precision, Recall và F1-score.

- + *top_k*: số lượng ảnh tương tự mà mô hình truy xuất;
- + *random_k*: số ảnh ngẫu nhiên giả định là không liên quan (âm tính);
- + *sample_size*: số ảnh truy vấn được chọn ngẫu nhiên để đánh giá.

Từ toàn bộ tập ảnh có vector embedding, chọn ngẫu nhiên *sample_size* ảnh làm truy vấn. Điều này cho phép đánh giá mô hình mà không cần *ground truth* toàn cục.

Với mỗi ảnh truy vấn, hệ thống gọi *find_similar_images* để trả về *top_k* ảnh được xem là *tích cực (positive)*. Sau đó, chọn ngẫu nhiên *random_k* ảnh khác không trùng truy vấn hay kết quả → xem như ảnh *tiêu cực (negative)*.

Tập nhãn thật *y_true*: gán 1 cho ảnh được truy vấn, 0 cho ảnh tiêu cực ngẫu nhiên. Tập nhãn dự đoán *y_pred*: mô hình giả định rằng *tất cả ảnh được truy vấn đều là liên quan* (tức là hệ thống đánh nhãn "1" cho mọi ảnh).

Trung bình các chỉ số theo số lượng truy vấn đã đánh giá thành công. Số lượng ảnh được sử dụng thực tế có thể nhỏ hơn *sample_size* nếu một số truy vấn bị loại bỏ (ví dụ: không tìm được ảnh tiêu cực đủ số lượng).

3.4. Mô hình GAT

3.4.1. Thư viện sử dụng trong cài đặt mô hình GAT

```
from torch_geometric.data import Data

from torch_geometric.nn import GATConv

import pickle
```

- `torch_geometric.data.Data` là lớp dữ liệu lõi trong thư viện PyTorch Geometric (PyG) – một thư viện học sâu mã nguồn mở chuyên xử lý dữ liệu đồ thị. Lớp `Data` cho phép biểu diễn một đồ thị đơn lẻ trong định dạng tối ưu hóa cho mô hình học sâu. Các thành phần chính được dùng:

- + `x`: ma trận đặc trưng đầu vào của các node, có kích thước $(num_nodes, num_features)$;
- + `edge_index`: ma trận biểu diễn các cạnh dưới dạng chỉ số, có kích thước $(2, num_edges)$, trong đó hàng đầu là node nguồn, hàng thứ hai là node đích;
- + `y` (tùy chọn): nhãn của node hoặc đồ thị, dùng trong bài toán phân loại;
- + `Data(...)`: đối tượng được sử dụng làm đầu vào cho các lớp học sâu như GCN, GAT, GraphSAGE trong `torch_geometric.nn`.

- `torch_geometric.nn.GATConv` là lớp tích chập mạng đồ thị (Graph Attention Convolution Layer) trong *PyTorch Geometric*, triển khai kiến trúc *Graph Attention Network (GAT)*. Lớp này cho phép mô hình học *trọng số chú ý (attention weight)* giữa các node lân cận để điều chỉnh mức độ ảnh hưởng trong quá trình truyền tín hiệu. Các thành phần chính được dùng:

- + `in_channels` và `out_channels`: xác định kích thước đầu vào và đầu ra của node embedding;
- + `heads`: số lượng "đầu chú ý" (multi-head attention) được sử dụng để tăng cường tính biểu đạt của mô hình;
- + `GATConv(...)`: thực hiện lan truyền thông tin theo công thức có trọng số động:

$$h'_i = \sigma \left(\sum_{j \in N(i)} \alpha_{ij} W_{h_j} \right)$$

trong đó α_{ij} là trọng số chú ý học được giữa node i và lân cận j .

- *pickle* là thư viện tiêu chuẩn của Python được sử dụng để *tuần tự hóa* (*serialization*) và *giải tuần tự hóa* (*deserialization*) các đối tượng Python. Trong học sâu, pickle thường dùng để lưu trữ hoặc nạp lại mô hình, embedding, graph object,... nhằm tiết kiệm thời gian xử lý. Các thành phần chính được dùng:

- + *pickle.dump(obj, file)*: ghi đối tượng *obj* vào file nhị phân;
- + *pickle.load(file)*: nạp lại đối tượng từ file nhị phân đã lưu;
- + Ứng dụng điển hình: lưu *node_embeddings*, *model.state_dict()*, *Data* object hoặc cấu hình đồ thị huấn luyện để tái sử dụng mà không cần tạo lại từ đầu.

3.4.2. Đọc và tiền xử lý dữ liệu cho mô hình GAT

```
df = pd.read_csv("f_coco_triplets.csv")
all_entities = pd.concat([df['subject'], df['object']]).unique()
entity_encoder = LabelEncoder()
entity_encoder.fit(all_entities)
df['subject_id'] = entity_encoder.transform(df['subject'])
df['object_id'] = entity_encoder.transform(df['object'])
relation_encoder = LabelEncoder()
df['predicate_id'] = relation_encoder.fit_transform(df['predicate'])
num_nodes = len(entity_encoder.classes_)
x = torch.eye(num_nodes)
edge_index = torch.tensor([df['subject_id'].values, df['object_id'].values],
dtype=torch.long)
data = Data(x=x, edge_index=edge_index)
```

GAT là một mô hình học sâu trên đồ thị, khai thác cơ chế self-attention để học trọng số kết nối giữa các node lân cận. Trước khi huấn luyện mô hình GAT, cần chuẩn hóa dữ liệu dưới dạng *torch_geometric.data.Data* gồm các thành phần: *node features*, *edge index*, và nhãn.

Tập dữ liệu đầu vào chứa các bộ ba (*subject*, *predicate*, *object*) được trích xuất từ caption và hình ảnh trong tập COCO.

Tập tất cả các thực thể (*entities*) được hợp nhất từ cột *subject* và *object*.

Mỗi thực thể được ánh xạ thành một chỉ số nguyên duy nhất bằng *LabelEncoder* để phục vụ xây dựng đồ thị.

Mỗi quan hệ (*predicate*) cũng được mã hóa thành chỉ số nguyên, giúp xử lý thống nhất trong các mô hình mở rộng như R-GCN hoặc để phân tích sau này. Tuy nhiên, trong GAT chuẩn, thông tin về quan hệ không được sử dụng trực tiếp – đồ thị coi tất cả các cạnh là đồng nhất.

Đặc trưng ban đầu của mỗi node là một *vector one-hot* (ma trận đơn vị). Đây là cách biểu diễn mặc định nếu chưa có đặc trưng nội tại (như hình ảnh, ngữ nghĩa, embedding từ trước). Kích thước của ma trận đặc trưng: (*num_nodes*, *num_nodes*) – mỗi node là một vector có duy nhất một giá trị 1.

edge_index là tensor có kích thước (2, *num_edges*) đại diện cho các cạnh của đồ thị: Hàng đầu là chỉ số node nguồn (*subject*) và hàng dưới là chỉ số node đích (*object*). Đây là định dạng mặc định để xây dựng đồ thị trong PyTorch Geometric.

Đối tượng Data là cấu trúc dữ liệu trung tâm trong PyG, chứa tất cả thông tin cần thiết để huấn luyện GAT. Các thành phần chính trong *data* gồm *x* là ma trận đặc trưng đầu vào cho node và *edge_index* là cấu trúc cạnh của đồ thị.

3.4.3. Xây dựng mô hình Graph Attention Network (GAT)

```
class GATModel(nn.Module):
```

```
    def __init__(self, in_channels, hidden_channels, out_channels):
```

```
        super().__init__()
```

```
        self.gat1 = GATConv(in_channels, hidden_channels, heads=4)
```

```
        self.gat2 = GATConv(hidden_channels * 4, out_channels, heads=1)
```

```
    def forward(self, data):
```

```
        x, edge_index = data.x, data.edge_index
```

```

x = self.gat1(x, edge_index)

x = F.relu(x)

x = self.gat2(x, edge_index)

return x

def contrastive_loss(x_i, x_j, label, margin=1.0):

    dist = F.pairwise_distance(x_i, x_j)

    return (label * dist.pow(2) + (1 - label) * F.relu(margin - dist).pow(2)).mean()

```

Khởi tạo mô hình với *in_channels*: số chiều đầu vào của vector đặc trưng cho mỗi node (thường là số lượng node nếu one-hot). *hidden_channels*: số chiều vector trung gian sau lớp attention đầu tiên. *out_channels*: số chiều embedding đầu ra, có thể dùng để tính khoảng cách (nếu dùng cho retrieval).

Sử dụng 4 attention heads song song (*multi-head attention*), mỗi head học một cách biểu diễn khác nhau.

Lớp *gat1*: có *heads=4*, tức là có 4 "bộ attention" song song, giúp học được nhiều biểu diễn khác nhau từ lân cận;

Lớp *gat2*: có *heads=1*, tạo đầu ra chuẩn với số chiều *out_channels* để biểu diễn embedding cuối cùng.

Luồng dữ liệu *forward* để dữ liệu lan truyền từ node gốc đến lân cận, kết hợp trọng số attention. Hàm ReLU giúp đưa phi tuyến tính vào mạng. Lớp cuối cùng tạo vector embedding dùng cho downstream task (tìm ảnh, phân cụm, v.v.)

Hàm Contrastive Loss Cho vector embedding x_i và x_j , khoảng cách Euclidean là:

$$D(x_i, x_j) = \|x_i - x_j\|_2$$

Hàm tổn thất:

$$L = \frac{1}{N} \sum_{i=1}^N [y_i \cdot D^2 + (1 - y_i) \cdot \max(0, m - D^2)]$$

Trong đó:

+ $y_i \in \{0,1\}$: nhãn của cặp

- + m : margin (ngưỡng)
- + D : khoảng cách giữa hai vector

Tính tổn thất theo nhãn:

- + Với nhãn = 1: tổn thất = khoảng cách bình phương \rightarrow ép gần nhau.

+ Với nhãn = 0: tổn thất = $(\text{margin} - \text{dist})^2$ nếu khoảng cách nhỏ hơn margin \rightarrow ép cách xa.

3.4.4. Huấn luyện mô hình GAT

```
def train_model(data, model, epochs=100, lr=0.005, samples_per_epoch=2000):
```

```
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

```
    edge_index = data.edge_index.t().tolist()
```

```
    all_nodes = list(range(data.num_nodes))
```

```
    existing_edges = set(tuple(e) for e in edge_index)
```

```
    for epoch in range(epochs):
```

```
        model.train()
```

```
        optimizer.zero_grad()
```

```
        embeddings = model(data)
```

```
        # Positive samples: từ các cạnh trong đồ thị
```

```
        pos_samples = random.sample(edge_index, min(samples_per_epoch,
len(edge_index)))
```

```
        # Negative samples: node pair không kết nối
```

```
        neg_samples = []
```

```
        while len(neg_samples) < len(pos_samples):
```

```
            i, j = random.sample(all_nodes, 2)
```

```
            if (i, j) not in existing_edges and (j, i) not in existing_edges:
```

```
                neg_samples.append((i, j))
```

```
        # Lấy embedding
```

```

pos_i = torch.stack([embeddings[i] for i, j in pos_samples])
pos_j = torch.stack([embeddings[j] for i, j in pos_samples])
neg_i = torch.stack([embeddings[i] for i, j in neg_samples])
neg_j = torch.stack([embeddings[j] for i, j in neg_samples])

# Tính loss

loss_pos = contrastive_loss(pos_i, pos_j, torch.ones(len(pos_i)))
loss_neg = contrastive_loss(neg_i, neg_j, torch.zeros(len(neg_i)))
loss = loss_pos + loss_neg

loss.backward()

optimizer.step()

print(f"Epoch {epoch}: Loss = {loss.item():.4f}")

return model

```

Sử dụng các *cặp đối lập* (*contrastive sampling*) như vậy giúp mô hình học được sự khác biệt giữa liên kết thực và không thực, từ đó phản ánh rõ nét hơn cấu trúc đồ thị trong không gian embedding:

- + *Positive pairs* được chọn ngẫu nhiên từ danh sách các cạnh hiện có trong đồ thị (*edge_index*). Các cặp này đại diện cho mối quan hệ thực sự được ghi nhận trong dữ liệu gốc.

- + *Negative pairs* được sinh bằng cách chọn ngẫu nhiên hai node không có liên kết trực tiếp. Để đảm bảo tính hiệu quả, các cặp này được lọc kỹ để tránh trùng với bất kỳ cạnh nào có thật.

Tính hàm mất mát tương phản (Contrastive Loss):

- + Nếu $label = 1$ (tức là positive pair), mô hình cố gắng giảm khoảng cách giữa hai vector đặc trưng.

- + Nếu $label = 0$ (tức là negative pair), mô hình đẩy hai vector ra xa, sao cho khoảng cách ít nhất là *margin*.

Hàm tối ưu *Adam* được sử dụng để cập nhật tham số mạng nơ-ron, đảm bảo hội tụ nhanh và ổn định. Trước mỗi cập nhật, gradient được khởi tạo lại bằng *optimizer.zero_grad()*. Sau khi tính loss, thuật toán lan truyền ngược được kích hoạt với *loss.backward()*, và trọng số mô hình được điều chỉnh thông qua *optimizer.step()*.

Không cần nhãn phân loại cố định: Chỉ cần biết cặp nào liên kết hoặc không liên kết. Tận dụng tốt cấu trúc đồ thị: Mô hình học từ chính kết nối trong đồ thị thay vì gán nhãn cứng. Tối ưu hóa khoảng cách embedding: Thích hợp với các bài toán tìm kiếm ảnh liên quan hoặc gợi ý dựa trên đồ thị tri thức.

Sau khi huấn luyện, mô hình sẽ trả về một ma trận embedding, trong đó mỗi hàng là vector đặc trưng của một thực thể (node) trong đồ thị.

3.4.5. Trích xuất vector đặc trưng và tạo bản đồ ánh xạ từ entity đến ảnh

```
node_embeddings = trained_model(data).detach().numpy()

entity_idx_to_images = defaultdict(set)

for i, row in df.iterrows():

    entity_idx_to_images[row['subject_id']].add(row['image_id'])

    entity_idx_to_images[row['object_id']].add(row['image_id'])
```

Sau quá trình huấn luyện, mô hình GAT đã học được trọng số tối ưu để biểu diễn mỗi node (thực thể) trong đồ thị dưới dạng vector đặc trưng có ý nghĩa ngữ nghĩa.

+ *trained_model(data)*: gọi forward pass của mô hình GAT đã huấn luyện với toàn bộ dữ liệu đồ thị.

+ *.detach()*: tách kết quả khỏi đồ thị tính toán của PyTorch để không tính gradient (do không cần thiết trong giai đoạn suy luận).

+ *.numpy()*: chuyển kết quả từ tensor sang mảng NumPy, tiện cho các thao tác vector hóa và tính toán khoảng cách sau này.

Kết quả là một ma trận *node_embeddings* có kích thước (*số lượng node* × *chiều embedding*), trong đó mỗi dòng là biểu diễn vector của một thực thể trong đồ thị.

Mỗi thực thể trong đồ thị có thể liên quan đến nhiều ảnh khác nhau, thông qua các bộ ba *subject*, *predicate*, *object*. Để phục vụ cho truy vấn ảnh theo thực thể, cần xây dựng một ánh xạ ngược từ node ID sang danh sách các ảnh liên quan.

- + Duyệt qua từng dòng trong *DataFrame* gốc (đã mã hóa *ID*).

- + Ghi nhận rằng cả *subject_id* và *object_id* trong mỗi triplet đều có liên quan đến *image_id*.

- + Lưu các ảnh tương ứng vào một set để loại bỏ trùng lặp, với *entity_idx_to_images* là một từ điển ánh xạ từ *ID* của thực thể sang tập các ảnh có chứa thực thể đó.

3.4.6. Lưu mô hình và dữ liệu sau huấn luyện

```
_dir = "saved_models"

os.makedirs(output_dir, exist_ok=True)

with open(os.path.join(output_dir, "entity_encoder.pkl"), "wb") as f:

    pickle.dump(entity_encoder, f)

with open(os.path.join(output_dir, "relation_encoder.pkl"), "wb") as f:

    pickle.dump(relation_encoder, f)

np.save(os.path.join(output_dir, "node_embeddings.npy"), node_embeddings)

torch.save(trained_model.state_dict(), os.path.join(output_dir,
"gat_model_weights.pt"))

with open(os.path.join(output_dir, "entity_idx_to_images.pkl"), "wb") as f:

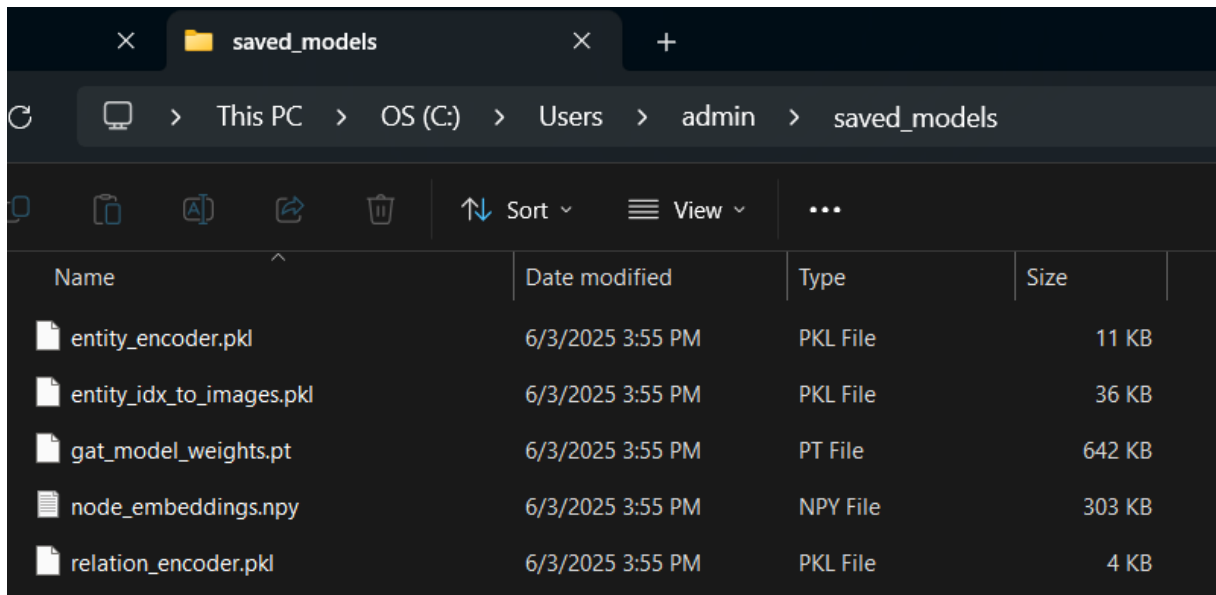
    pickle.dump(entity_idx_to_images, f)
```

Sau khi mô hình đã được huấn luyện và sinh ra các vector embedding đại diện cho các node trong đồ thị, bước tiếp theo là lưu toàn bộ các thành phần quan trọng để sử dụng lại trong giai đoạn suy luận (inference) hoặc triển khai (deployment). Quá trình này bao gồm nhiều phần:

- + Lưu bộ mã hóa LabelEncoder cho thực thể

- + Lưu bộ mã hóa LabelEncoder cho quan hệ

- + Lưu node embeddings đã huấn luyện
- + Lưu trọng số của mô hình GAT
- + Lưu ảnh xạ entity → image



Hình 3.1 Nơi lưu trữ mô hình saved_models

3.4.7. Truy xuất hình ảnh có nội dung tương đồng

```
def find_related_images(query_image_id, top_k=5):
    related_entities = set(df[df['image_id'] == query_image_id]['subject_id']) | \
        set(df[df['image_id'] == query_image_id]['object_id'])

    if not related_entities:
        return []

    query_vec = sum(torch.tensor(node_embeddings[i]) for i in related_entities)
    / len(related_entities)

    scores = cosine_similarity(query_vec.reshape(1, -1), node_embeddings)[0]

    top_entity_indices = scores.argsort()[-top_k:][::-1]

    related_images = set()

    for idx in top_entity_indices:
        related_images.update(entity_idx_to_images[idx])
```

return list(related_images - {query_image_id}][:top_k]

Ảnh đầu vào (*query_image_id*) có thể liên kết với nhiều thực thể khác nhau (ở vị trí subject hoặc object trong tập triplet). Tập hợp *related_entities* thu thập tất cả ID của các thực thể này, sử dụng phép hợp tập hợp (\cup), giúp mô hình xem xét tổng thể nội dung có trong ảnh truy vấn.

Mỗi thực thể đã được ánh xạ thành vector embedding thông qua mô hình GAT. Vector biểu diễn của ảnh được tính bằng trung bình cộng các embedding của các thực thể liên quan, thể hiện ngữ nghĩa tổng quát của ảnh.

Hàm *cosine_similarity* được sử dụng để đo độ tương đồng ngữ nghĩa giữa vector của ảnh truy vấn và tất cả các vector node (thực thể) trong toàn bộ đồ thị. Kết quả là một mảng *scores*, trong đó mỗi phần tử thể hiện mức độ tương đồng giữa ảnh đầu vào với một thực thể cụ thể.

Lấy chỉ số của *top_k* thực thể có độ tương đồng cao nhất với ảnh đầu vào. Đây là những thực thể có khả năng đại diện cho các chủ đề tương đồng hoặc đồng ngữ cảnh.

Với mỗi thực thể gần nhất tìm được, mô hình truy vấn tất cả các ảnh đã từng chứa thực thể đó (từ ánh xạ *entity_idx_to_images*). Tập hợp *related_images* chính là những ảnh liên quan về mặt ngữ nghĩa với ảnh đầu vào.

Ảnh truy vấn gốc bị loại khỏi kết quả. Trả về tối đa *top_k* ảnh khác để phục vụ truy vấn.

CHƯƠNG 4: ĐÁNH GIÁ KẾT QUẢ

4.1. Đánh giá các mô hình (TransE, GCN, R-GCN, GAT)

Mỗi mô hình được huấn luyện trên tập dữ liệu triplet trích xuất từ COCO2017 và ánh xạ vào không gian vector nhằm hỗ trợ việc tìm kiếm ảnh liên quan thông qua truy vấn ngôn ngữ tự nhiên hoặc ảnh đầu vào.

- TransE:

- + Dựa trên embedding học từ triplets (h, r, t).
- + Tính khoảng cách $||h+r-t||$ cho cặp ảnh dương và âm.
- + Được đánh giá qua tỉ lệ phân biệt chính xác giữa triplet thật và giả.

- GCN (Graph Convolutional Network):

- + Dùng cấu trúc đồ thị không có loại cạnh (homogeneous).
- + Tạo node embedding bằng cách truyền thông tin qua các lớp GCN.
- + Không phân biệt loại quan hệ, phù hợp với đồ thị đơn giản.

- GAT (Graph Attention Network):

- + Mở rộng từ GCN bằng cơ chế attention giữa các đỉnh.
- + Cho phép mô hình học mức độ quan trọng khác nhau giữa các node lân cận.
- + Embedding thu được giàu tính ngữ cảnh hơn, hữu ích cho phân biệt thực thể.

- R-GCN (Relational GCN):

- + Mở rộng GCN với khả năng xử lý nhiều loại quan hệ.
- + Mỗi loại cạnh (quan hệ) có ma trận truyền riêng.
- + Phù hợp cho đồ thị tri thức như COCO-triplets với nhiều predicate khác nhau.

4.2. So sánh độ chính xác: Precision / Recall / F1-score

Để đánh giá định lượng hiệu năng của các mô hình, ba chỉ số chính được sử dụng là Precision, Recall và F1-score. Cụ thể:

- + Precision đo lường tỷ lệ ảnh được truy vấn đúng trong tập kết quả.

$$Precision = \frac{Số ảnh đúng}{Tổng số ảnh truy vấn}$$

+ Recall phản ánh khả năng bao phủ ảnh đúng từ toàn bộ tập dữ liệu liên quan.

$$Recall = \frac{Số ảnh đúng}{Tổng số ảnh liên quan}$$

+ F1-score là trung bình điều hòa của Precision và Recall, phản ánh sự cân bằng giữa độ chính xác và khả năng bao phủ.

$$F1 - score = 2 \cdot \frac{P \cdot R}{P + R}$$

Kết quả thống kê trên tập kiểm thử với truy vấn ảnh ngẫu nhiên cho thấy:

	Precision	Recall	F1-score
TransE	0.008	0.004	0.0013
GCN	0.3412	0.738	0.4671
GAT	0.6163	0.1418	0.1503
R-GCN	0.4444	1.0	0.6154

Bảng 4.1 So sánh kết quả đánh giá giữa các mô hình

Mô hình TransE: Hiệu suất rất thấp ở cả ba chỉ số → không phù hợp cho bài toán tìm kiếm ảnh với đặc trưng phức tạp như trong COCO2017. Lý do có thể đến từ: Biểu diễn vector đơn giản ($s + r$) không đủ phân biệt các quan hệ ngữ cảnh hình ảnh. Thiếu tính cấu trúc đồ thị đa chiều (không tận dụng được quan hệ kết nối như GCN/GAT).

Mô hình GCN: Kết quả tương đối tốt, đặc biệt là recall cao → mô hình thu thập được hầu hết các ảnh đúng, dù còn nhiều ảnh nhiễu. Điều này cho thấy GCN khai thác được mối quan hệ lân cận trong đồ thị tri thức, nhưng vẫn chưa phân biệt tốt ảnh chính xác nhất.

Mô hình GAT: Đạt Precision cao nhất, tức là trong top ảnh trả về có tỷ lệ đúng cao, nhưng recall rất thấp → bỏ sót nhiều ảnh đúng. GAT có cơ chế attention nên học được trọng số tập trung, nhưng có thể bị quá chọn lọc, dẫn đến thiếu bao phủ.

Mô hình R-GCN: Đạt hiệu suất tổng thể tốt nhất. Mô hình khai thác rất tốt cấu trúc quan hệ có hướng và loại quan hệ trong đồ thị, nhờ đó bao quát gần như đầy đủ ảnh liên quan mà vẫn giữ mức chính xác cao.

4.3. Truy vấn minh họa và ảnh kết quả

Để đánh giá trực quan khả năng truy xuất ảnh của các mô hình đã huấn luyện, nhóm thực hiện tiến hành truy vấn minh họa với một số triplet tiêu biểu có tính phổ biến cao trong tập dữ liệu COCO2017. Một trong những truy vấn được lựa chọn là (boy, riding, pony), tương ứng với mô tả bằng ngôn ngữ tự nhiên là “a boy riding a pony”. Truy vấn này nhằm kiểm tra mức độ nhận diện các ảnh có chứa đồng thời hai thực thể “boy” và “pony”, trong bối cảnh các mô hình có thể xử lý tốt các thực thể nhưng chưa chắc phân biệt được mối quan hệ giữa chúng.

```
Trích triplet: (boy, riding, pony)
'boy' → 'person' → entity_id = 748
'pony' → 'horse' → entity_id = 497
'riding' → 'riding' → entity_id = 851
```

Tổng cộng 10 ảnh liên quan:

- 000000441442.jpg
- 000000244099.jpg
- 000000199236.jpg
- 000000040036.jpg
- 000000553990.jpg
- 000000454798.jpg
- 000000140270.jpg
- 000000576052.jpg
- 000000242678.jpg
- 000000191614.jpg

Ảnh phù hợp theo mức độ ưu tiên: ['000000441442.jpg', '000000244099.jpg', '000000199236.jpg', '000000040036.jpg', '000000553990.jpg', '000000454798.jpg', '000000140270.jpg', '000000576052.jpg', '000000242678.jpg', '000000191614.jpg']



Hình 4.1 Truy vấn “a boy riding a pony” của GAT

Kết quả truy vấn được thể hiện cụ thể trong hình cho thấy với mỗi mô hình hiển thị các ảnh có điểm tương đồng cao nhất so với vector truy vấn. Đối với mô hình TransE, do cấu trúc biểu diễn truy vấn đơn giản dưới dạng phép cộng vector ($s + r$), ảnh kết quả mang tính ngẫu nhiên cao, thiếu sự gắn kết nội dung ví dụ như “teddy”, “bear”, “cake”. Hầu hết các ảnh được truy xuất chỉ chứa một trong hai thực thể, dẫn đến chất lượng truy vấn thấp, điều này phù hợp với các chỉ số đánh giá định lượng đã trình bày ở mục trước.

Truy vấn: (teddy - bear - cake)
Top ảnh liên quan:



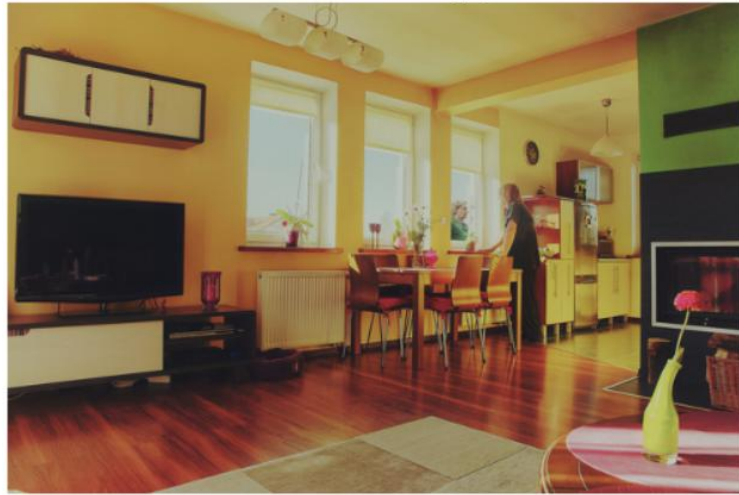
Hình 4.2 Truy xuất “teddy”, “bear”, “cake” của TransE

Tuy mô hình GCN đạt chỉ số Recall cao và cho thấy năng lực bao phủ tốt trong quá trình truy vấn, nhưng khi quan sát kết quả truy xuất ảnh cụ thể, vẫn ghi nhận hiện tượng một số ảnh bị lệch so với ảnh truy vấn. Cụ thể, trong truy vấn cung cấp ảnh có bao gồm các object như “person”, “table”, “vase”, “tv”, “room”, GCN có thể thu hồi được các ảnh chứa một trong hai thực thể, song không đảm bảo sự kết hợp hài hòa cả hai thành phần cùng xuất hiện trong cùng một ngữ cảnh.

```
: # Hiển thị ảnh truy vấn và ảnh tương tự  
show_similar_images("000000000139", top_k=5)
```

Ảnh truy vấn: 000000000139

000000000139.jpg



Hình 4.3 Truy xuất với đầu vào là ảnh bằng mô hình GCN

```
: # Hiển thị ảnh thuộc cùng một cụm  
show_images_in_cluster(0, max_img=5)
```

000000382734.jpg

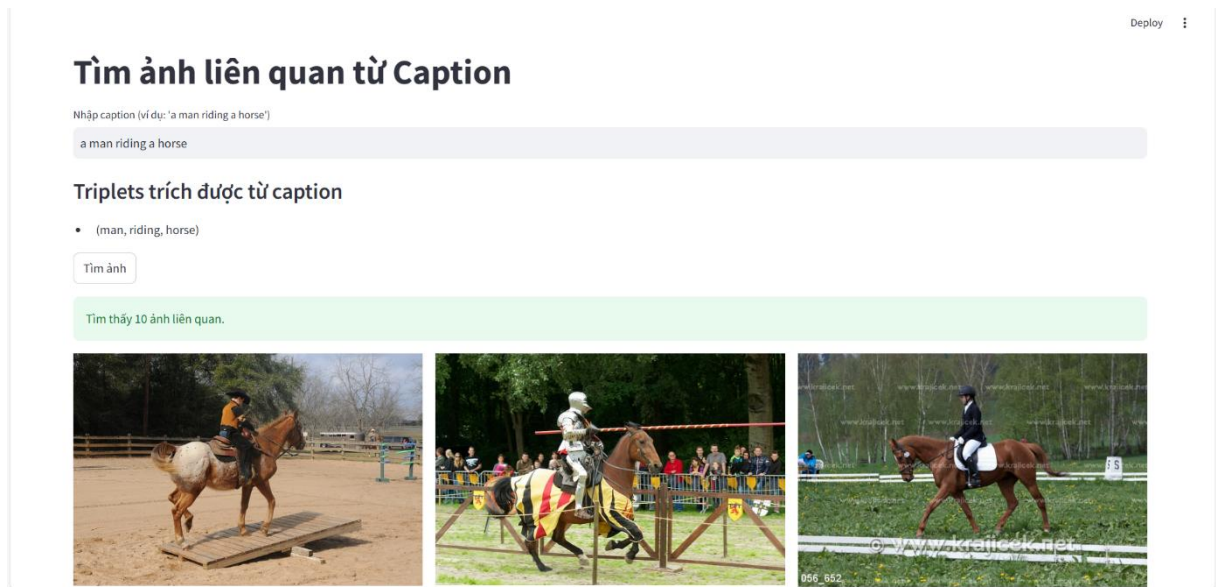


```
: # Hiển thị ảnh thuộc cùng một cụm  
show_images_in_cluster(0, max_img=5)
```



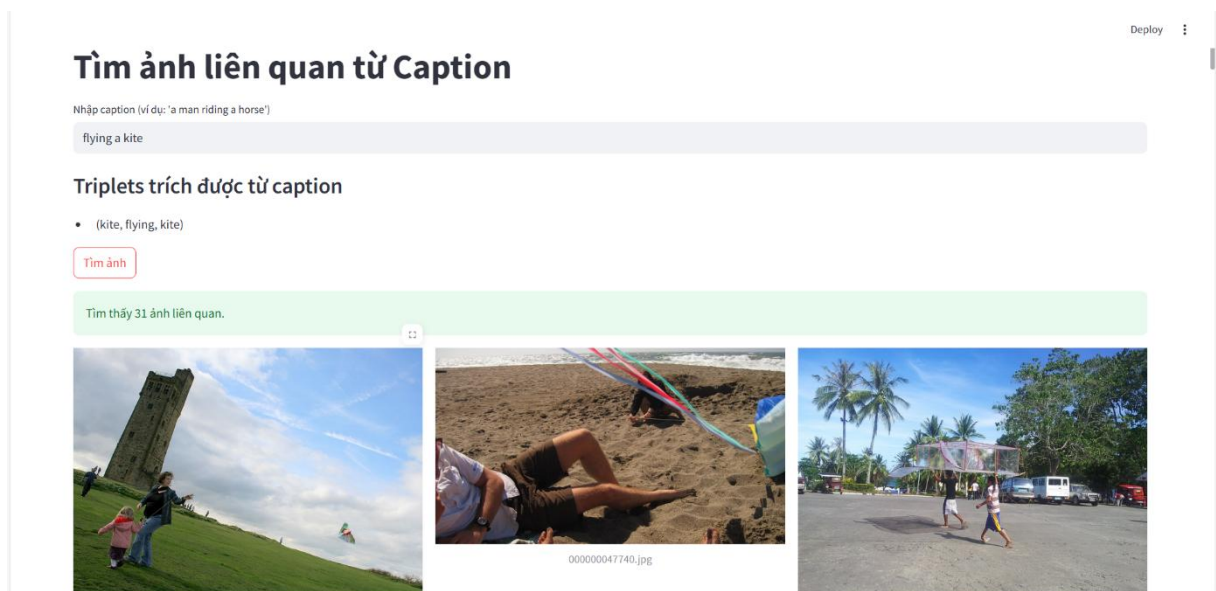
Hình 4.4 Kết quả của truy vấn bằng ảnh

Nguyên nhân của hiện tượng này có thể đến từ đặc trưng lan truyền tín hiệu của GCN theo kiểu đồng nhất (homogeneous propagation), nơi mà tất cả các đỉnh lân cận đều đóng góp tương đương trong quá trình tổng hợp thông tin. Điều này khiến cho các thực thể có liên kết lỏng lẻo, ngữ cảnh ít đặc trưng vẫn được đưa vào kết quả truy vấn.



Hình 4.5 Truy vấn đầy đủ subject, predicate, object của R-GCN

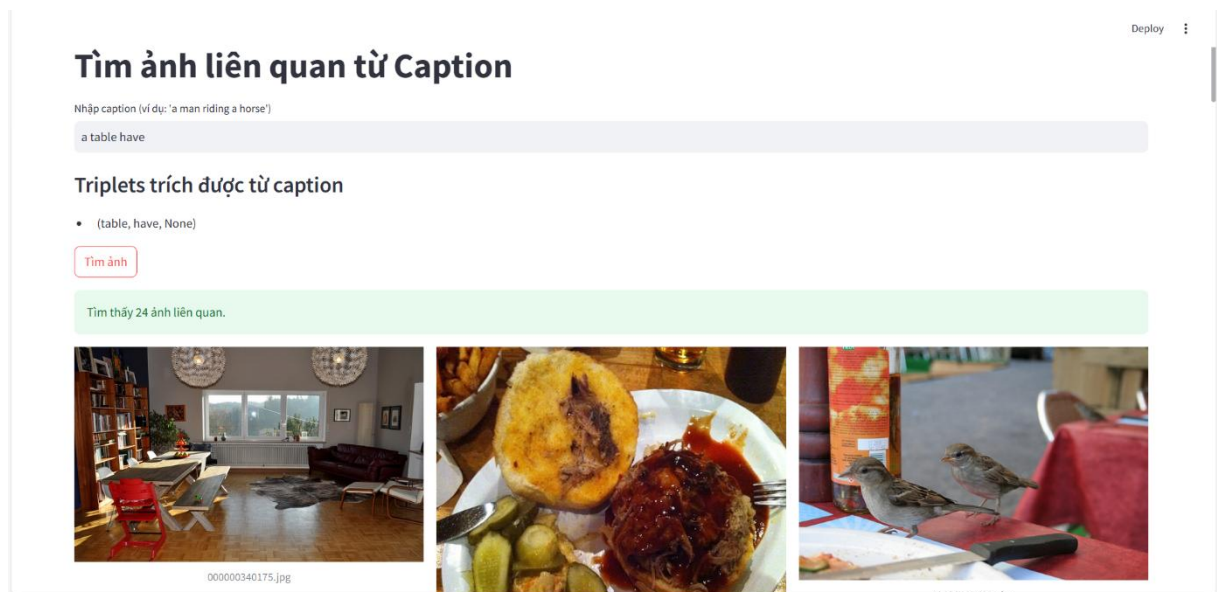
Mô hình R-GCN trong truy vấn đầu tiên với caption “a man riding a horse”, hệ thống đã trích xuất chính xác triplet (man, riding, horse) và truy xuất được 10 ảnh liên quan, thể hiện rõ mối quan hệ giữa chủ thể là người nam và đối tượng là ngựa, cùng hành động cưỡi ngựa. Các ảnh thu được mang tính đa dạng về bối cảnh (trường đấu, sân huấn luyện, biểu diễn cưỡi ngựa), cho thấy khả năng biểu diễn ngữ nghĩa của mô hình là hiệu quả và ổn định.



Hình 4.6 Truy vấn không đầy đủ chỉ predicate, object của R-GCN

Với truy vấn “flying a kite”, mặc dù caption không có chủ ngữ rõ ràng, hệ thống vẫn xác định được hành động “flying” và đối tượng “kite”, từ đó truy xuất thành công

31 ảnh thể hiện hoạt động thả điều với sự hiện diện của con người và cánh điều trong các bối cảnh phong phú. Điều này cho thấy mô hình có khả năng tổng quát hóa tốt, kể cả khi đầu vào không tuân thủ cấu trúc ngữ pháp đầy đủ.



Hình 4.7 Truy vấn không đầy đủ chỉ subject, predicate của R-GCN

Truy vấn thứ ba “a table have” tiếp tục minh chứng cho tính linh hoạt của hệ thống, khi caption chỉ chứa chủ thể và động từ nhưng vẫn truy xuất được 24 ảnh có liên quan trực tiếp đến đối tượng “table”, chủ yếu là ảnh bàn ăn hoặc bàn làm việc với các vật dụng kèm theo.

Các kết quả này cho thấy mô hình R-GCN không chỉ hoạt động hiệu quả khi đủ ba thành phần subject–predicate–object, mà còn thể hiện độ ổn định khi phải xử lý caption thiếu thành phần hoặc mơ hồ về ngữ nghĩa.

4.4. Phân tích ưu nhược điểm mô hình

4.4.1. Mô hình TransE

Ưu điểm:

- + Cấu trúc đơn giản, dễ huấn luyện và tính toán.
- + Phù hợp với các tập dữ liệu tri thức có số lượng quan hệ vừa phải, ít nhiễu.

Nhược điểm:

- + Thiếu khả năng phân biệt mối quan hệ phức tạp giữa các thực thể do biểu diễn quan hệ chỉ bằng phép cộng tuyến tính ($s + r \approx t$).
- + Không tận dụng được cấu trúc đồ thị ngữ cảnh xung quanh thực thể.
- + Kết quả truy vấn ảnh nghèo nàn về độ chính xác và ngữ nghĩa, thể hiện qua Precision và F1-score gần như bằng 0.

4.4.2. Mô hình GCN

Ưu điểm:

- + Khai thác được cấu trúc đồ thị nhờ lan truyền thông tin qua các tầng lân cận.
- + Cho độ bao phủ cao trong truy vấn (Recall cao), giúp thu hồi được nhiều ảnh liên quan đến thực thể.

Nhược điểm:

- + Không phân biệt được loại quan hệ trong đồ thị, gây nhiều khi các kết nối không thực sự mang tính ngữ nghĩa.
- + Một số ảnh được truy xuất có thể lệch khỏi truy vấn do mô hình đánh đồng đóng góp của các đỉnh lân cận.
- + Tính chọn lọc chưa cao, Precision và F1-score chỉ ở mức trung bình.

4.4.3. Mô hình GAT

Ưu điểm:

- + Sử dụng cơ chế attention để điều chỉnh trọng số giữa các đỉnh lân cận, từ đó làm nổi bật các kết nối quan trọng.
- + Precision cao nhất trong các mô hình, tức là ảnh top-k có độ liên quan cao hơn.

Nhược điểm:

- + Do đặc tính chọn lọc mạnh, mô hình dễ bỏ sót các ảnh có thể đúng nhưng ít nổi bật \rightarrow dẫn đến Recall thấp.
- + F1-score không cao do sự mất cân bằng giữa Precision và Recall.
- + Không tận dụng được thông tin loại quan hệ như R-GCN.

4.4.4. Mô hình R-GCN

Ưu điểm:

- + Tích hợp đầy đủ thông tin về cả cấu trúc lẫn loại quan hệ trong đồ thị, giúp mô hình hiểu được ngữ cảnh sâu sắc hơn.
- + Đạt kết quả tốt nhất ở cả ba chỉ số: Precision, Recall và F1-score.
- + Truy vấn minh họa cho thấy khả năng truy xuất các ảnh có ngữ nghĩa phù hợp, đảm bảo cả độ đầy đủ và độ chính xác.

Nhược điểm:

- + Cấu trúc mô hình phức tạp hơn, thời gian huấn luyện và yêu cầu bộ nhớ cao hơn so với các mô hình khác.
- + Có thể bị overfitting nếu số loại quan hệ quá lớn mà không có cơ chế regularization thích hợp.

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Kết quả đạt được

Trong đề tài này, em đã xây dựng thành công hệ thống tìm kiếm ảnh dựa trên biểu diễn đồ thị tri thức, với dữ liệu đầu vào là các triplet (subject, predicate, object) trích xuất từ tập dữ liệu COCO2017. Hệ thống được triển khai và đánh giá với bốn mô hình biểu diễn tri thức gồm: TransE, GCN, GAT và R-GCN. Các mô hình đã được huấn luyện và kiểm thử trên tập dữ liệu thống nhất, cho phép so sánh trực tiếp hiệu suất dựa trên các chỉ số đánh giá phổ biến như Precision, Recall và F1-score.

Kết quả thực nghiệm cho thấy mô hình R-GCN đạt hiệu suất tổng thể tốt nhất, với F1-score lên đến 0.6154, vượt trội so với các mô hình còn lại nhờ khả năng khai thác cả cấu trúc và loại quan hệ trong đồ thị. Mô hình GAT cho kết quả Precision cao nhất, trong khi GCN đạt Recall tốt nhất, phản ánh ưu thế từng phần của từng kiến trúc mạng. Mô hình TransE, mặc dù đơn giản và hiệu quả tính toán cao, song không đáp ứng tốt yêu cầu phân biệt ngữ nghĩa phức tạp, dẫn đến kết quả thấp trên toàn bộ các chỉ số.

Ngoài ra, hệ thống đã được thiết kế với khả năng trích xuất vector truy vấn từ caption người dùng, ánh xạ ngữ nghĩa thông qua *synonym_map*, và trả về danh sách ảnh có mức độ liên quan cao nhất. Kết quả truy vấn minh họa cho thấy các mô hình dựa trên mạng đồ thị có thể tái hiện tương đối chính xác mối liên kết ngữ nghĩa giữa các thực thể hình ảnh, góp phần mở ra hướng ứng dụng trong các hệ thống tìm kiếm thông minh, đặc biệt trong bối cảnh dữ liệu phi cấu trúc đang ngày càng phổ biến.

5.2. Hạn chế hệ thống hiện tại

Hiệu quả của mô hình TransE thấp trên tập dữ liệu COCO, phản ánh sự không phù hợp của các mô hình embedding đơn giản trong môi trường có ngữ cảnh thị giác phức tạp.

Việc đánh giá mới dừng ở mức top-k truy vấn (top-5), chưa bao quát các tiêu chí như mAP, NDCG hoặc mức độ đa dạng trong kết quả tìm kiếm.

Chưa có cơ chế lọc hoặc xác thực mối quan hệ semantic trong truy vấn đầu vào, đặc biệt trong các caption chứa từ đa nghĩa hoặc mối quan hệ trừu tượng.

Tập dữ liệu COCO2017 chỉ là ảnh đại diện, chưa kiểm chứng khả năng mô hình trong các dữ liệu chuyên biệt và lớn hơn như y tế, công nghiệp, ảnh vệ tinh.

5.3. Định hướng phát triển tương lai

Tối ưu hóa kiến trúc mô hình bằng cách kết hợp R-GCN với attention đa chiều hoặc kiến trúc hybrid giữa mạng đồ thị và CNN, từ đó tận dụng cả thông tin ngữ nghĩa và đặc trưng hình ảnh.

Mở rộng tập dữ liệu huấn luyện và kiểm thử, không chỉ dừng ở COCO mà bao gồm các tập khác như Visual Genome, Flickr30k Entities để kiểm tra tính khái quát hóa.

Cải thiện giao diện người dùng sử dụng Streamlit để cho phép truy vấn hình ảnh bằng văn bản một cách trực quan, từ đó phục vụ nghiên cứu UX/UI hoặc demo.

Phát triển các mô hình học đa phương thức (multimodal) để kết hợp embedding từ hình ảnh (ResNet, CLIP) với biểu diễn đồ thị tri thức, nhằm nâng cao chất lượng truy vấn trong các trường hợp caption không rõ ràng.

Ứng dụng trong các bài toán truy vết đối tượng theo ngữ nghĩa trong video, khai thác liên kết thời gian và tương tác đa thực thể trong chuỗi khung hình.

TÀI LIỆU THAM KHẢO

- [1] Nguyễn Thanh Tùng (2021). *Khóa luận tốt nghiệp: Biểu diễn tri thức bằng mạng nơ-ron đồ thị (GCN, GAT, R-GCN) trong truy xuất văn bản*. Trường Đại học Công nghệ – ĐHQGHN.
- [2] Nguyễn Văn Dũng (2020). *Ứng dụng đồ thị tri thức trong hệ thống hỏi đáp tiếng Việt sử dụng mô hình TransE*. Luận văn Thạc sĩ, Đại học Bách khoa Hà Nội
- [3] Bùi Quang Minh (2022). *Học sâu trên đồ thị: Mạng GCN và ứng dụng trong phân loại thực thể*. Tài liệu học thuật nội bộ, Viện Công nghệ Thông tin – Viện Hàn lâm KH&CN Việt Nam.
- [4] Đỗ Đức Thắng (2020). *Học sâu – Ứng dụng trong xử lý ảnh và thị giác máy tính*. Tài liệu nội bộ Đại học Bách khoa Hà Nội.
- [5] Bordes, A., Usunier, N., Garcia-Durán, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. *Advances in Neural Information Processing Systems*, 26, 2787–2795.
- [6] Kipf, T. N., & Welling, M. (2017). *Semi-supervised classification with graph convolutional networks*. *Proceedings of ICLR 2017*. <https://arxiv.org/abs/1609.02907>
- [7] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). *Graph Attention Networks*. *Proceedings of ICLR 2018*. <https://arxiv.org/abs/1710.10903>
- [8] W3C (2004). *Resource Description Framework (RDF): Concepts and Abstract Syntax*. Retrieved from <https://www.w3.org/TR/rdf-concepts/>