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

[**1. XÁC ĐỊNH YÊU CẦU VÀ MỤC TIÊU 3**](#_Toc197701729)

[**2. TỔNG QUAN PHƯƠNG PHÁP 4**](#_Toc197701730)

[**3. CÔNG NGHỆ SỬ DỤNG 6**](#_Toc197701731)

[**4. GIỚI THIỆU COCO DATASET 8**](#_Toc197701732)

[**5. MÔ TẢ ĐỒ THỊ TRI THỨC 10**](#_Toc197701733)

[**6. MÔ TẢ QUY TRÌNH TỪ CAPTION COCO ĐẾN ĐỒ THỊ TRI THỨC 18**](#_Toc197701734)

[**7. THUẬT TOÁN FASTER R-CNN 19**](#_Toc197701735)

[**8. THUẬT TOÁN GCN 21**](#_Toc197701736)

[**9. THUẬT TOÁN TRANSE 24**](#_Toc197701737)

[**10. R-GCN (RELATIONAL GRAPH CONVOLUTIONAL NETWORK) 29**](#_Toc197701738)

[**11. SIFT VÀ BOVW 32**](#_Toc197701739)

[**12. MÔ HÌNH KẾT HỢP R-GCN VÀ ĐẶC TRƯNG ẢNH (VISUAL-AWARE R-GCN) 34**](#_Toc197701740)

[**13. GRAPH ATTENTION NETWORKS (GAT) 35**](#_Toc197701741)

[**14. CÁC THƯ VIỆN ĐƯỢC CÀI ĐẶT 38**](#_Toc197701742)

[**15. TRÍCH XUẤT ĐỐI TƯỢNG BẰNG DETECRON2 40**](#_Toc197701743)

[**16. ĐÁNH GIÁ HIỆU QUẢ CÁC MÔ HÌNH TÌM KIẾM ẢNH 42**](#_Toc197701744)

[**17. CÔNG CỤ TÌM KIẾM ẢNH: API VÀ GIAO DIỆN NGƯỜI DÙNG 44**](#_Toc197701745)

1. XÁC ĐỊNH YÊU CẦU VÀ MỤC TIÊU

**1.1. Bối cảnh và lý do nghiên cứu**

Trong thời đại bùng nổ thông tin và sự phát triển vượt bậc của trí tuệ nhân tạo, việc khai thác hiệu quả dữ liệu hình ảnh để phục vụ các nhu cầu truy vấn và tìm kiếm thông minh đang trở thành một xu hướng tất yếu. Các hệ thống tìm kiếm ảnh truyền thống chủ yếu dựa trên các đặc trưng thị giác đơn giản như histogram màu, SIFT, hoặc deep feature CNN để biểu diễn nội dung ảnh. Tuy nhiên, những phương pháp này thường không đủ khả năng nắm bắt được ngữ nghĩa sâu xa và mối quan hệ giữa các đối tượng trong ảnh – một yếu tố đóng vai trò quan trọng trong việc hiểu bối cảnh hình ảnh.

Trong khi đó, đồ thị tri thức (Knowledge Graph – KG) đã chứng minh được sức mạnh của nó trong việc biểu diễn thông tin ngữ nghĩa phức tạp trong nhiều lĩnh vực như xử lý ngôn ngữ tự nhiên, truy vấn ngữ nghĩa, và hệ thống gợi ý. Việc áp dụng đồ thị tri thức vào bài toán tìm kiếm ảnh không chỉ mở ra một hướng tiếp cận mới mà còn có tiềm năng cải thiện đáng kể độ chính xác, khả năng truy xuất theo ngữ cảnh và tính linh hoạt của hệ thống.

Trong phạm vi đề tài này, chúng tôi đề xuất một hệ thống tìm kiếm ảnh thông minh dựa trên việc kết hợp đồ thị tri thức và thông tin láng giềng của các đối tượng trong ảnh, với dữ liệu được sử dụng là COCO2017 – một trong những bộ dữ liệu hình ảnh chú thích đầy đủ và chuẩn hóa nhất hiện nay.

**1.2. Mục tiêu của đề tài**

**1.2.1. Mục tiêu chính**

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 tiểu đồ thị tương đồng hoặc gần nhất với đồ thị truy vấn, dựa trên vector embedding học được từ mạng nơ-ron đồ thị (GNN).

**1.2.2. Mục tiêu cụ thể**

* 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 (Detectron2, GNN, GAT, TransE) 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.

**1.3. Ý nghĩa thực tiễn**

Hệ thống tìm kiếm ảnh theo quan hệ ngữ nghĩa không chỉ phục vụ cho mục đích nghiên cứu học thuật mà còn có tiềm năng ứng dụng rộng rãi trong nhiều lĩnh vực:

* **Truy xuất ảnh y tế**: Tìm ảnh chẩn đoán bệnh nhân có đặc điểm tương đồng dựa trên biểu hiện hình ảnh.
* **An ninh và giám sát**: Truy vấn các ảnh chứa người hoặc vật thể trong ngữ cảnh cụ thể như "người đang mang ba lô ở bến xe".
* **Thư viện ảnh và hệ thống media**: Gợi ý hình ảnh theo bối cảnh cụ thể cho người dùng, hỗ trợ quảng cáo và truyền thông.
* **Giáo dục và đào tạo**: Cung cấp hình ảnh theo chủ đề học tập có liên quan đến nội dung bài giảng.

Sự kết hợp giữa trí tuệ nhân tạo, đồ thị tri thức, và xử lý ảnh không chỉ góp phần nâng cao khả năng hiểu và truy xuất dữ liệu phi cấu trúc mà còn thể hiện tính liên ngành sâu sắc giữa thị giác máy tính, học sâu, và khai phá tri thức.

2. TỔNG QUAN PHƯƠNG PHÁP

**2.1. Phân tích cấu trúc dữ liệu COCO2017**

Bộ dữ liệu COCO2017 (Common Objects in Context) là một trong những tập dữ liệu hình ảnh chuẩn mực nhất hiện nay dùng trong các bài toán thị giác máy tính. Bộ dữ liệu này cung cấp hơn 118.000 ảnh huấn luyện với annotation chi tiết về đối tượng, bounding box, segmentation, keypoints, và caption mô tả bằng ngôn ngữ tự nhiên.

Các thành phần chính trong annotation COCO bao gồm:

* **Danh mục đối tượng (Categories)**: Gồm 80 loại đối tượng phổ biến như người, xe, động vật, đồ vật.
* **Annotation**: Mỗi ảnh có các chú thích đối tượng, bounding box (tọa độ), segmentation mask, và id.
* **Caption**: Mỗi ảnh có từ 5 câu mô tả bằng tiếng Anh, chứa nhiều thông tin ngữ nghĩa và hành động.

Việc kết hợp thông tin từ bounding box và caption sẽ cung cấp cơ sở đầy đủ để xây dựng đồ thị tri thức phản ánh cả không gian và ngữ nghĩa của ảnh.

**2.2. Tiền xử lý và trích xuất đối tượng**

Quá trình tiền xử lý bao gồm:

* Đọc annotation từ tệp instances\_train2017.json và captions\_train2017.json
* Trích xuất các đối tượng, nhãn, bounding box từ annotation
* Kết hợp caption để bổ sung quan hệ dạng ngữ nghĩa, ví dụ: (man, holding, umbrella)
* Chuẩn hóa label bằng WordNet, synonym matching để tránh trùng lặp khái niệm (man/person/human)

Trong trường hợp không dùng annotation gốc, có thể áp dụng Detectron2 để nhận diện trực tiếp các đối tượng từ ảnh.

**2.3. Xây dựng đồ thị tri thức**

Mỗi ảnh sẽ được ánh xạ thành một tiểu đồ thị, trong đó:

* **Nút (node)** là các đối tượng: dog, person, car...
* **Cạnh (edge)** là các mối quan hệ: near, on, holding, behind...
* **Thuộc tính (attribute)** gồm tọa độ, kích thước, ảnh gốc, vị trí trong ảnh

Hai loại cạnh được tạo:

* **Quan hệ không gian** từ vị trí bounding box (ví dụ: near, above)
* **Quan hệ ngữ nghĩa** từ caption (ví dụ: holding, riding)

KG có thể lưu dưới dạng RDF Triplet hoặc sử dụng NetworkX/DGL để thao tác huấn luyện. Đối với ứng dụng truy vấn nhanh, có thể sử dụng cơ sở dữ liệu Neo4j với định dạng Cypher.

**2.4. Biểu diễn và học embedding bằng GNN**

Sau khi tạo đồ thị, bước tiếp theo là biểu diễn node và edge bằng vector để phục vụ truy vấn. Một số phương pháp:

* **TransE/TransH**: Embedding knowledge graph bằng dịch chuyển vector quan hệ
* **GCN/GAT**: Graph Neural Network học embedding dựa vào thông tin của node láng giềng
* **PageRank / HITS**: Đánh giá độ quan trọng của node để khởi tạo embedding đầu vào

Embedding vector sẽ được sử dụng để tính độ tương đồng giữa đồ thị truy vấn và đồ thị ảnh (qua cosine similarity, Euclidean distance, etc.)

**2.5. Truy vấn bằng ngôn ngữ tự nhiên**

Khi người dùng nhập truy vấn, hệ thống sẽ:

* Sử dụng NLP (spaCy) để phân tích triplet: subject – predicate – object
* Tạo một tiểu đồ thị tương ứng
* Dùng mô hình GAT đã huấn luyện để tính embedding cho truy vấn
* So khớp embedding truy vấn với các ảnh đã lưu (top-k similarity)

**2.6. Giao diện tìm kiếm ảnh**

* **API**: Triển khai truy vấn tìm kiếm qua FastAPI
* **Frontend**: Hiển thị kết quả tìm kiếm qua giao diện đơn giản (Streamlit/React/HTML)
* **Hiển thị ảnh, độ tương đồng, caption**

Kết quả tìm kiếm được sắp xếp theo điểm cosine similarity giữa vector ảnh và vector truy vấn, đảm bảo kết quả phản ánh đúng ngữ cảnh.

3. CÔNG NGHỆ SỬ DỤNG

Việc xây dựng một hệ thống tìm kiếm ảnh thông minh dựa trên đồ thị tri thức và thông tin láng giềng đòi hỏi sự phối hợp chặt chẽ của nhiều công nghệ từ thị giác máy tính, xử lý ngôn ngữ tự nhiên, khai phá tri thức, đến học sâu trên đồ thị. Trong phần này, chúng tôi trình bày chi tiết các công nghệ đã được sử dụng trong từng giai đoạn chính của hệ thống.

**3.1. Xử lý ảnh**

**3.1.1. Detectron2**

Detectron2 là một framework mạnh mẽ do Facebook AI Research phát triển, được xây dựng trên nền PyTorch. Nó hỗ trợ nhiều mô hình nhận diện đối tượng như Faster R-CNN, Mask R-CNN, RetinaNet, v.v. Trong hệ thống này, Detectron2 được sử dụng để phát hiện các đối tượng chính trong ảnh COCO nếu không sử dụng annotation gốc, và cho đầu ra gồm nhãn đối tượng, bounding box, độ tin cậy, và segmentation mask nếu cần.

**3.1.2. OpenCV**

OpenCV là thư viện xử lý ảnh mã nguồn mở được sử dụng để hiển thị, chuyển đổi định dạng ảnh, và trực quan hóa bounding box hoặc các vùng chú thích. OpenCV đóng vai trò phụ trợ trong bước tiền xử lý dữ liệu và minh họa kết quả.

**3.1.3. PyTorch**

PyTorch là nền tảng học sâu chính của hệ thống. Các mô hình mạng nơ-ron như GNN, GAT, cũng như các thao tác trên embedding vector đều được huấn luyện và triển khai thông qua PyTorch.

**3.2. Biểu diễn và lưu trữ đồ thị tri thức**

**3.2.1. NetworkX**

NetworkX là thư viện Python rất phổ biến trong thao tác và biểu diễn đồ thị phi hướng hoặc có hướng. Trong giai đoạn tiền xử lý và huấn luyện mô hình, đồ thị của từng ảnh (gồm các node và mối quan hệ) được dựng bằng NetworkX để dễ dàng tích hợp với các bước mã hóa và trích xuất đặc trưng.

**3.2.2. DGL (Deep Graph Library)**

DGL là thư viện mạnh mẽ dành riêng cho việc huấn luyện các mô hình học sâu trên đồ thị như GCN, GraphSAGE, GAT và R-GCN. Sau khi xây dựng đồ thị bằng NetworkX, đồ thị được chuyển sang DGLGraph để phục vụ cho quá trình huấn luyện GNN. DGL hỗ trợ PyTorch và cung cấp nhiều lớp xử lý chuyên biệt cho node classification, edge prediction và graph embedding.

**3.2.3. Neo4j**

Neo4j là cơ sở dữ liệu đồ thị dạng property-graph phổ biến nhất hiện nay, sử dụng ngôn ngữ truy vấn Cypher. Trong hệ thống này, Neo4j có thể được sử dụng để lưu trữ toàn bộ đồ thị tri thức và phục vụ truy vấn nhanh qua ngôn ngữ Cypher, đặc biệt thích hợp nếu triển khai trên nền tảng có yêu cầu mở rộng quy mô hoặc tích hợp hệ thống tìm kiếm real-time.

**3.3. Mô hình học biểu diễn và tìm kiếm ảnh**

**3.3.1. Graph Neural Networks (GNN)**

GNN là kiến trúc mạng nơ-ron được thiết kế để học biểu diễn từ đồ thị. GNN cho phép mỗi nút cập nhật đặc trưng của nó thông qua thông tin từ các nút láng giềng, qua đó học được các biểu diễn ngữ cảnh hiệu quả hơn so với các mô hình CNN truyền thống. Trong hệ thống này, GNN được sử dụng để tính embedding cho mỗi node (object) và từ đó suy diễn embedding cho ảnh tổng thể.

**3.3.2. Graph Attention Networks (GAT)**

GAT là một mở rộng của GNN trong đó mỗi nút học được trọng số khác nhau cho từng láng giềng thay vì gán trọng số đồng đều. Điều này rất quan trọng trong biểu diễn ảnh vì không phải mọi mối quan hệ đều quan trọng như nhau. Việc dùng GAT giúp hệ thống học được sự ưu tiên giữa các mối quan hệ như "holding", "riding" quan trọng hơn "next\_to".

**3.3.3. Knowledge Graph Embeddings (TransE, TransH)**

TransE và các biến thể như TransH, TransR là các kỹ thuật nhúng đồ thị tri thức truyền thống, biểu diễn mỗi triplet (h, r, t) dưới dạng không gian vector và tối ưu hóa sao cho: h + r ≈ t. Mặc dù không có khả năng học phi tuyến cao như GNN, các embedding này đơn giản, dễ huấn luyện và có thể dùng để so sánh nhanh hoặc khởi tạo vector đầu vào.

**3.4. Triển khai hệ thống và giao diện**

**3.4.1. FastAPI**

FastAPI là framework xây dựng API nhanh và hiện đại, tương thích tốt với Python và hỗ trợ async. API trong hệ thống này chịu trách nhiệm nhận truy vấn từ người dùng, xử lý NLP, sinh embedding truy vấn, và trả lại danh sách các ảnh phù hợp nhất.

**3.4.2. Streamlit và HTML/React**

* **Streamlit**: Là công cụ Python trực quan hóa dữ liệu nhanh chóng, phù hợp cho demo và kiểm thử mô hình học máy.
* **HTML/React**: Giao diện nhẹ có thể tích hợp dễ dàng với backend API để hiển thị kết quả truy vấn ảnh. React phù hợp cho triển khai thực tế cần hiệu năng cao và khả năng mở rộng giao diện.

4. GIỚI THIỆU COCO DATASET

**4.1. Thông tin cơ bản**

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.

**4.2. Thu thập dữ liệu COCO**

**Cách thu thập**: 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

**4.3. Cấu trúc bộ dữ liệu COCO**

Cấu trúc của bộ dữ liệu COCO có thể được chia thành các thành phần chính:

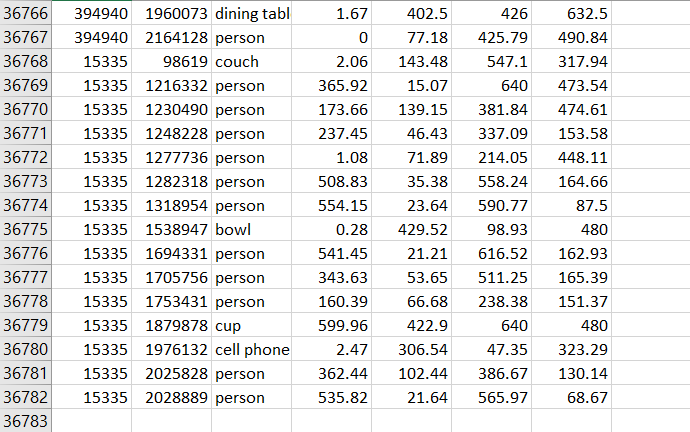
* **Annotations**: Tập hợp các chú thích mô tả về đối tượng trong ảnh. Bao gồm:
  + **Bounding boxes**: Vị trí của các đối tượng.
  + **Categories**: Loại của đối tượng (ví dụ: người, xe hơi, động vật).
  + **Segmentation masks**: Vùng phân đoạn của đối tượng trong ảnh (dành cho phân đoạn).
  + **Keypoints**: Các điểm đặc trưng (ví dụ: các khớp trên cơ thể người).
* **Images**: Thông tin về các hình ảnh như ID, đường dẫn ảnh, kích thước ảnh.
* **Categories**: Các loại đối tượng được phân loại trong bộ dữ liệu COCO, chẳng hạn như **person, car, dog**, v.v.
* **Captions**: Các mô tả ảnh dưới dạng văn bản.

**4.4. Quy trình trích xuất caption**

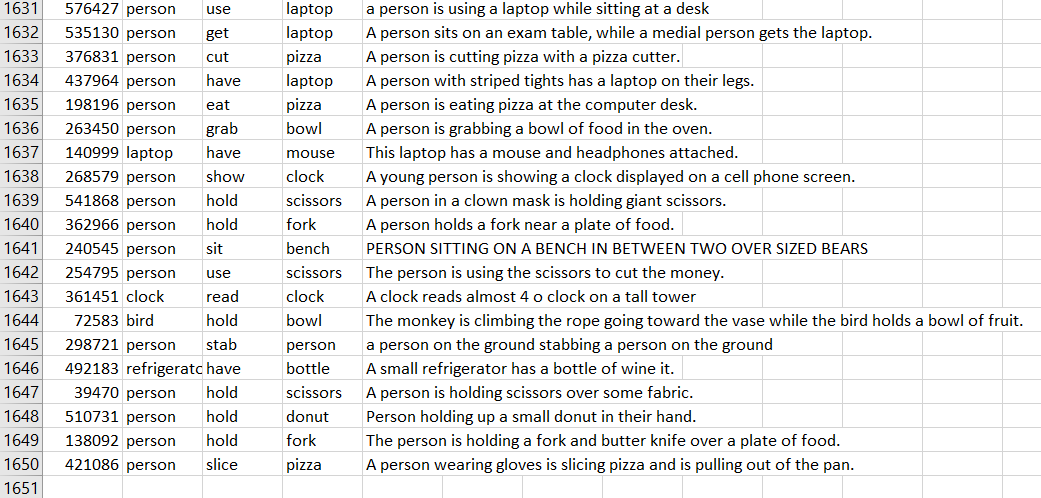
* File captions\_train2017.json --> Đọc JSON bằng Python
* Sau đó nhóm caption theo image\_id
* Tiếp theo trích xuất triplet từ caption
* Cuối cùng ghép với image\_id → lưu thành triplets

Sau khi trích xuất nhận được tổng:

**36782 object**



**1650 relationship**



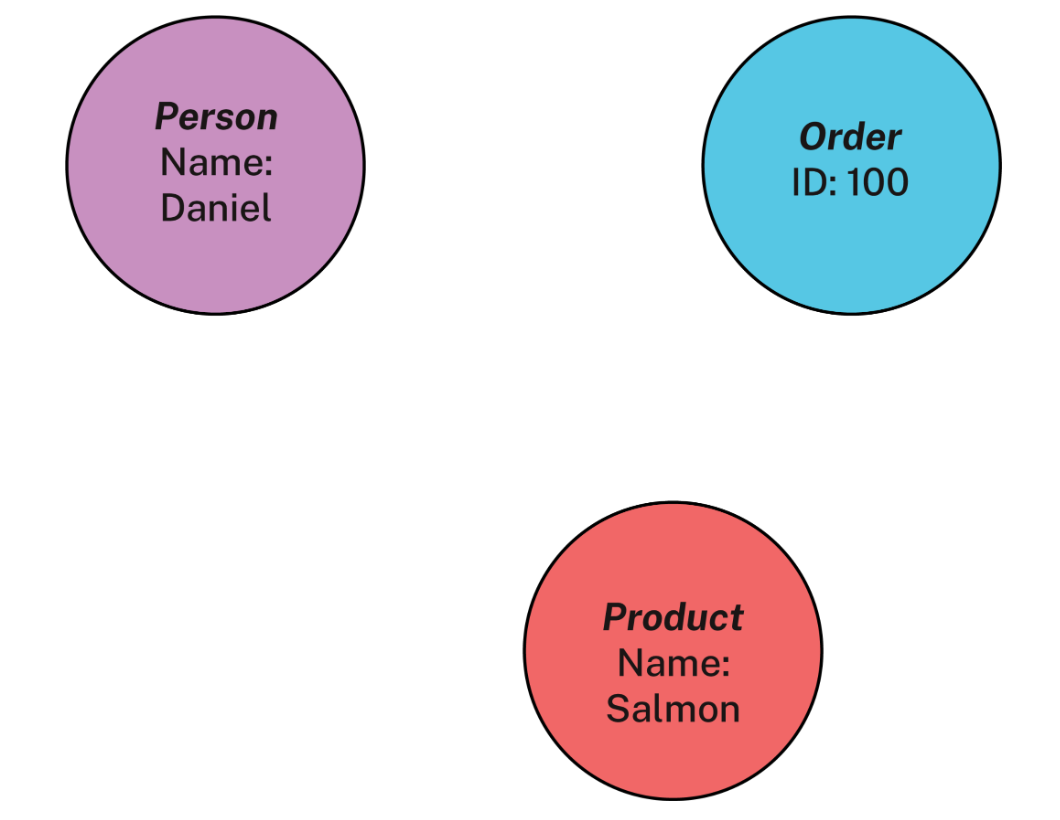
5. MÔ TẢ ĐỒ THỊ TRI THỨC

**5.1. Khái niệm**

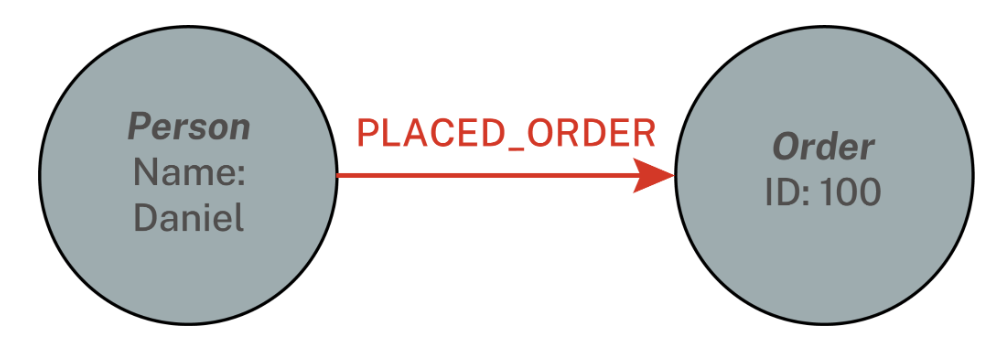
Đồ thị tri thức là một biểu diễn có tổ chức của các thực thể trong thế giới thực và các mối quan hệ của chúng. Biểu đồ này thường được lưu trữ trong cơ sở dữ liệu biểu đồ, nơi lưu trữ các mối quan hệ giữa các thực thể dữ liệu. Các thực thể trong biểu đồ kiến ​​thức có thể biểu diễn các đối tượng, sự kiện, tình huống hoặc khái niệm. Các mối quan hệ giữa các thực thể này nắm bắt bối cảnh và ý nghĩa về cách chúng được kết nối.

Nó giúp thể hiện cách dữ liệu liên kết với nhau, mang lại ngữ cảnh và ý nghĩa. Được sử dụng rộng rãi trong AI, tìm kiếm thông tin, và phân tích dữ liệu.

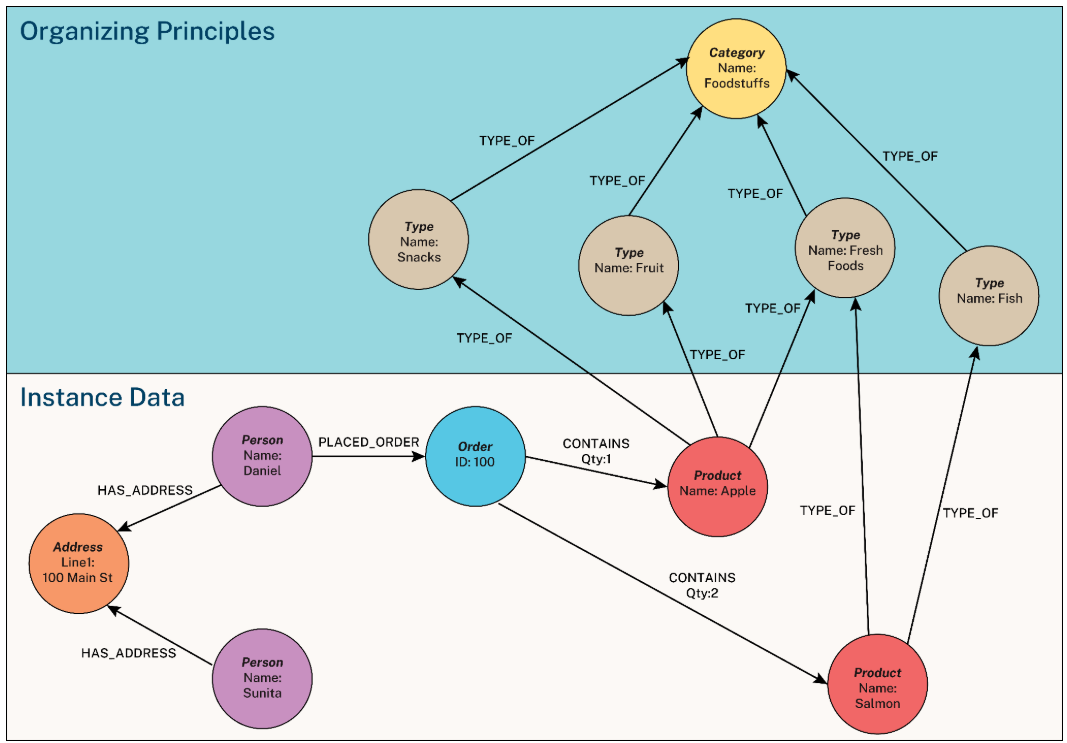
Thực thể (Entities) biểu thị và lưu trữ thông tin chi tiết về con người, địa điểm, đối tượng hoặc tổ chức. Mỗi thực thể là một nút có một (hoặc đôi khi là nhiều) nhãn để xác định loại nút và tùy chọn có thể có một hoặc nhiều thuộc tính (attributes). Các nút đôi khi cũng được gọi là đỉnh. Ví dụ các thực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



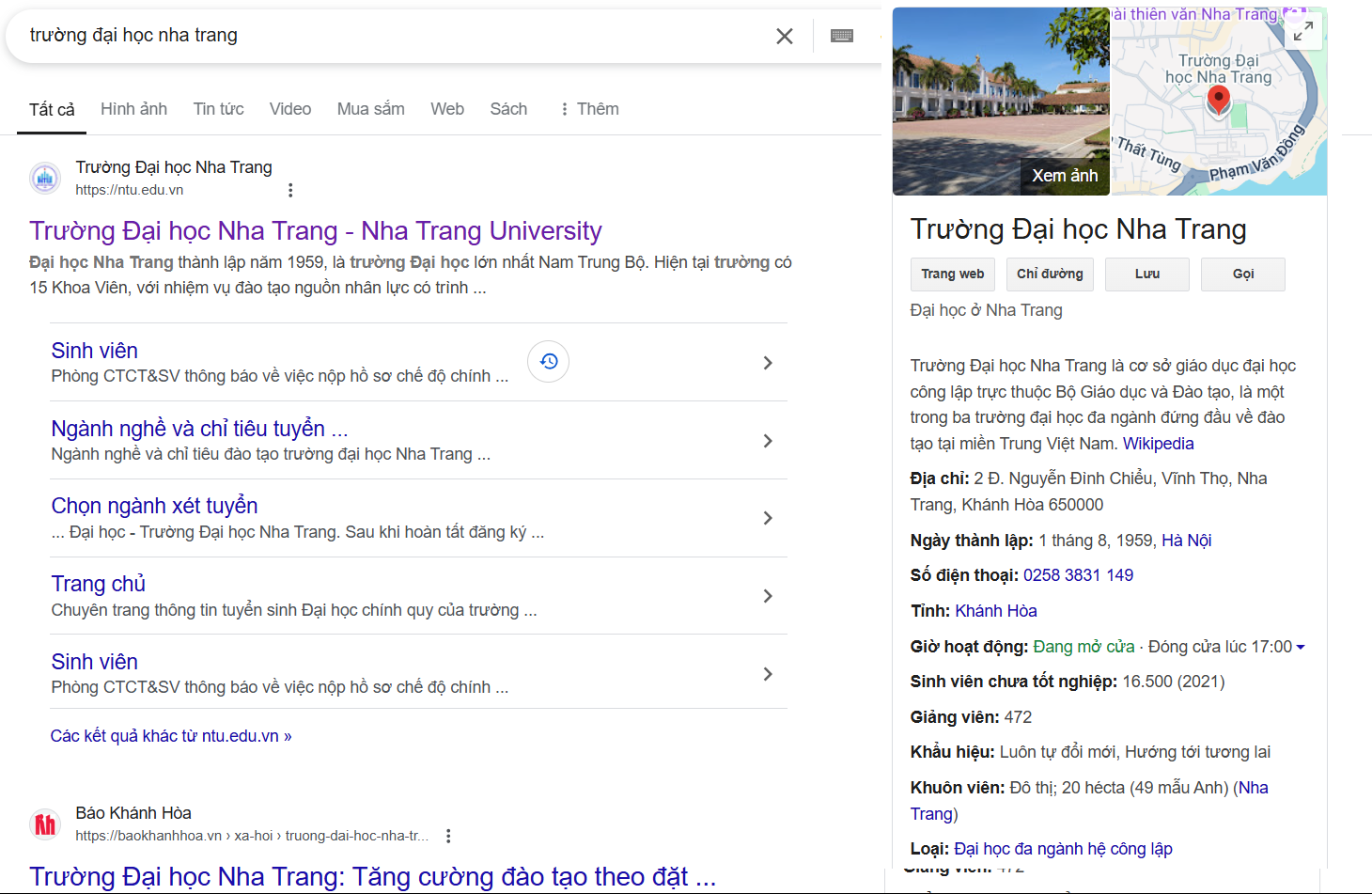
Quan hệ (relationships) xác định cách các thực thể liên kết với nhau. Giống như các nút, mỗi mối quan hệ có một nhãn xác định loại mối quan hệ và có thể tùy chọn có một hoặc nhiều thuộc tính. Mối quan hệ đôi khi cũng được gọi là cạnh. Trong ví dụ thương mại điện tử, tồn tại mối quan hệ giữa khách hàng và các nút đặt hàng, nắm bắt mối quan hệ “đã đặt hàng” giữa khách hàng và đơn hàng của họ:



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



Google Knowledge Graph đã thay đổi cách chúng ta tìm kiếm và tìm thông tin trên Web. Nó thu thập các dữ kiện về con người, địa điểm và sự vật thành một mạng lưới các thực thể có tổ chức. Khi bạn thực hiện tìm kiếm thông tin trên Google, nó sử dụng các kết nối giữa các thực thể để đưa ra các kết quả có liên quan nhất trong ngữ cảnh, ví dụ, trong hộp mà Google gọi là "Knowledge Panel.”



**5.2. Thành phần**

**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. Mối 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 “đã đặt hàng” giữa khách hàng và đơn hàng của họ:

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

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 đã 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.

**5.3. RDF và OWL**

**RDF (Resource Description Framework)**: Biểu diễn tri thức dưới dạng bộ ba gồm: Chủ ngữ (Subject), Vị ngữ (Predicate), và Tân ngữ (Object).

Ví dụ: <Hà Nội> <là thủ đô của> <Việt Nam>.

**OWL (Web Ontology Language)**: Mở rộng RDF, cung cấp ngữ nghĩa phong phú hơn, hỗ trợ mô tả các mối quan hệ phức tạp và các ràng buộc logic.

Ví dụ: Định nghĩa "mọi người dân sống ở Hà Nội đều sống ở Việt Nam".

**5.4. Hệ quản trị cơ sở dữ liệu đồ thị Neo4j**

Là một hệ quản trị cơ sở dữ liệu đồ thị (Graph Database Management System), được thiết kế để lưu trữ, quản lý và truy vấn dữ liệu dưới dạng đồ thị. Dữ liệu trong Neo4j được mô hình hóa bằng các node (đỉnh), relationship (cạnh), và properties (thuộc tính). Đây là một công cụ rất mạnh mẽ trong việc xử lý các loại dữ liệu có mối quan hệ phức tạp, như mạng xã hội, đồ thị tri thức, và các ứng dụng khác cần lưu trữ quan hệ giữa các thực thể.

Truy vấn bằng Cypher: Neo4j sử dụng một ngôn ngữ truy vấn riêng gọi là Cypher, rất mạnh mẽ và dễ hiểu, cho phép người dùng dễ dàng truy vấn các đồ thị phức tạp.

Tối ưu cho dữ liệu đồ thị: Neo4j có thể tối ưu hóa các phép toán đồ thị, như tìm kiếm các node hoặc cạnh gần nhau, tìm kiếm ngắn nhất đường đi giữa các node, hoặc các phép toán phân tích đồ thị.

Tích hợp tốt với các công cụ phân tích và học máy: Neo4j cung cấp các thư viện tích hợp để làm việc với các công cụ học máy, phân tích dữ liệu.

**5.5. Các bước cài đặt Neo4j**

Cài đặt Neo4j Desktop hoặc Neo4j Community Edition.

Khởi động Neo4j và tạo một cơ sở dữ liệu mới.

**5.6. Biểu diễn dữ liệu triplet vào lưu trong 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 (Đỉnh)**:
  + 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").
    - bounding\_box: Tọa độ bounding box của đối tượng (x1, y1, x2, y2).
    - confidence: Độ tin cậy của mô hình đối với đối tượng.
* **Relationships (Mối quan hệ)**:
  + Các mối quan hệ giữa các đối tượng có thể là:
    - :NEXT\_TO: Khi các đối tượng nằm gần nhau.
    - :IN: Khi một đối tượng nằm trong một đối tượng khác.
    - :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.

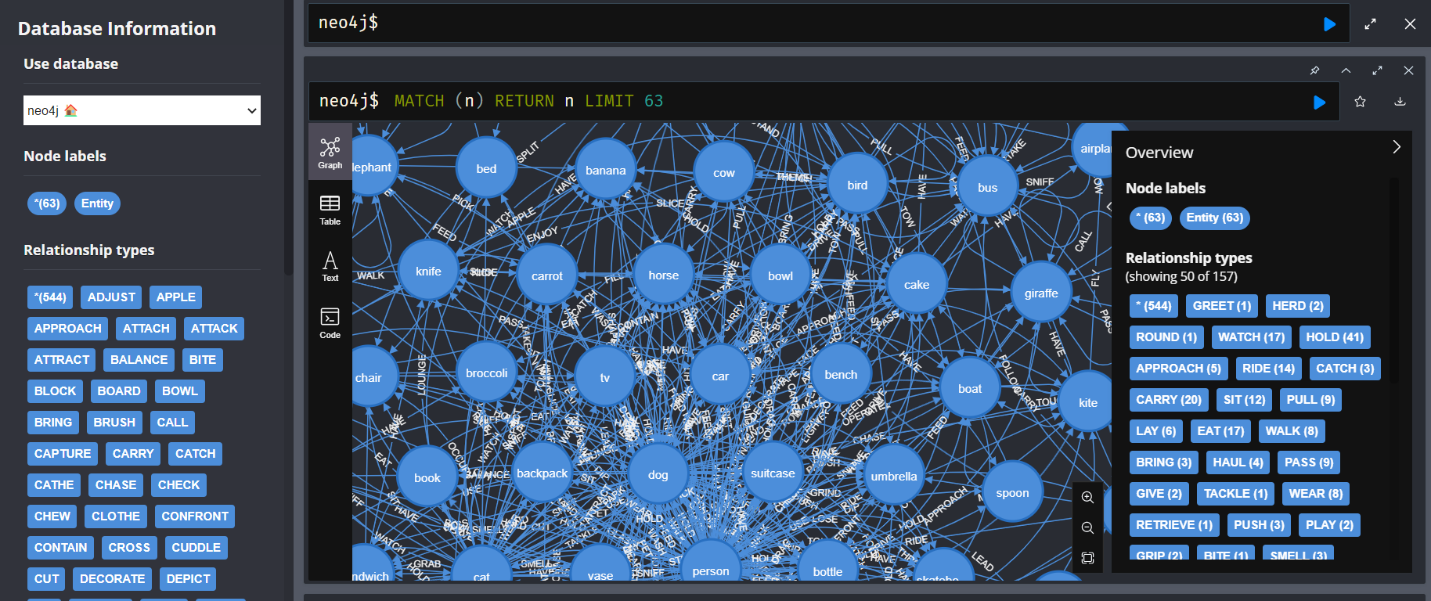
Sau khi đưa 36782 objects và 1650 relationships thì số object và relationship được lưu lại trong Neo4j là **63 objects** và **544 relationships**

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**

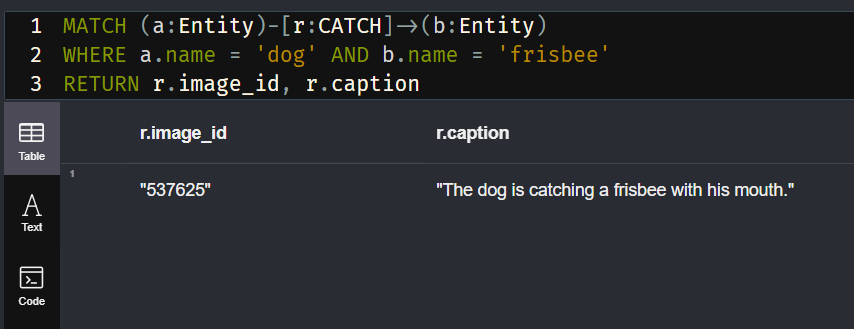


**5.7. Truy vấn Cypher**

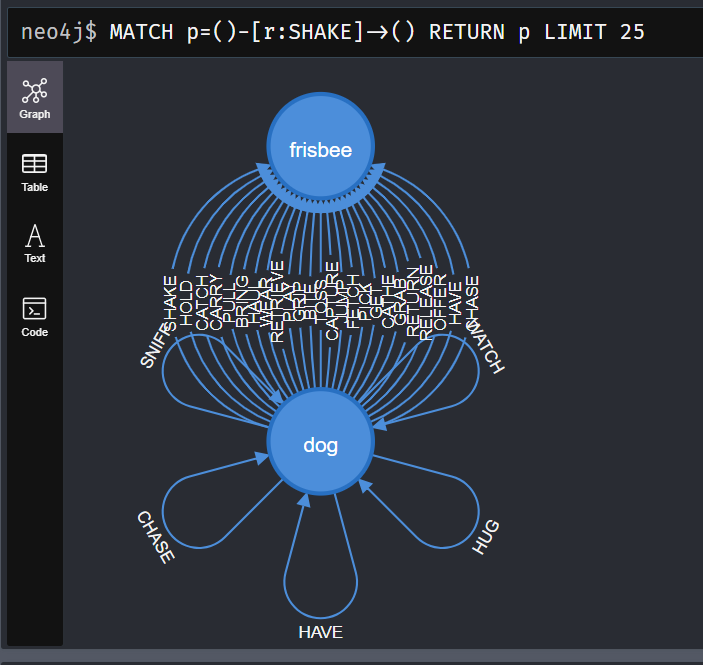
*MATCH (a:Entity)-[r:CATCH]->(b:Entity)*

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

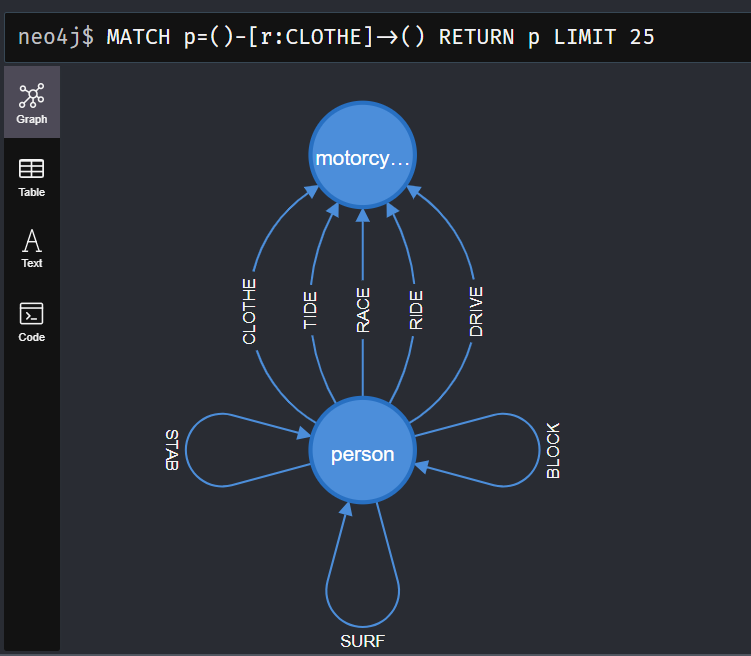
*RETURN r.image\_id, r.caption)*



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



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



6. MÔ TẢ QUY TRÌNH TỪ CAPTION COCO ĐẾN ĐỒ THỊ TRI THỨC

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)**

7. THUẬT TOÁN FASTER R-CNN

**7.1. RPN (Region Proposal Network)**

cls*r*= sigmoid(*W r . x + b r*)

* *clsr* là lớp phân loại cho mỗi anchor box (được phân loại là có đối tượng hay không)
* *W r* và *b r* là các trọng số và bias trong lớp RPN

**7.2. Hồi quy Bounding Box**

**∆** *x* = MLP(*x*)

* **∆** *x* là các điều chỉnh cho các anchor box dựa trên thông tin từ các lớp mạng

**Classification and Bounding Box Regression** Sau khi các vùng ứng cử được lọc qua RPN, Faster R-CNN tiếp tục quá trình phân loại và điều chỉnh bounding box với công thức:

cls*f* = softmax(*W r . x + b r*)

* cls*f* là phân loại cho mỗi đối tượng trong ảnh
* *W r* và *b r* là các trọng số và bias cho lớp phân loại cuối cùng

**Hồi quy Bounding Box**

**∆** *xf* = MLP*f* (*x*)

* **∆** *xf*  là các điều chỉnh cho các bounding box sau khi phân loại

**7.3. Kết quả của Faster R-CNN**

RPN sẽ tính toán hai giá trị cho mỗi anchor box:

* Phân loại: Liệu anchor box đó có chứa đối tượng hay không (classification).
* Hồi quy bounding box: Điều chỉnh vị trí của anchor box sao cho nó phù hợp với vị trí của đối tượng.

Phân loại cho RPN: Giả sử *W r* = 0.5, br = −0.2 và đặc trưng *x* = 0.7 (giá trị giả không phải giá trị cố định vì được học từ dữ liệu trong quá trình huấn luyện)

cls*r*= sigmoid(0.5 x 0.7 – 0.2) = sigmoid(0.15)

cls*r*= ≈ 0.537

Vậy thì ***clsr* ≈ 0.537**, có nghĩa là độ tin cậy rằng anchor box này chứa đối tượng là khoảng **53.7%**.

Hồi quy bounding box: Giả sử *W r* = 0.2, br = 0.1 và *x* = 0.7:

**∆** *x* = 0.2 x 0.7 + 0.1 = 0.24

Vậy thì **∆ *x*** **≈ 0.24**, nghĩa là anchor box cần điều chỉnh **0.24 đơn vị** để phù hợp với vị trí thực tế của đối tượng.

Sau khi các proposals đã được tạo ra và phân loại, mô hình tiếp tục điều chỉnh các bounding box dựa trên các tham số hồi quy:

Giả sử *Wf* = 0.6, b*f* = −0.4 và *x* = 0.9:

cls*f* = softmax(0.6 x 0.9 – 0.4) = softmax(0.14)

cls*f* = = ≈ 0.535

Vậy **cls*f* ≈ 0.535**, nghĩa là khả năng đối tượng này thuộc lớp nào đó trong bộ phân loại là **53.5%**.

Điều chỉnh bounding box: Giả sử **∆** *xf* = 0.3 sau khi tính toán từ một mạng MLP (Multi-Layer Perceptron) trong quá trình hồi quy bounding box

Nếu bounding box ban đầu có tọa độ **(x1=10, y1=20, x2=30, y2=40)**, thì bounding box đã điều chỉnh sẽ có tọa độ là:

*x1’ = x*1 + **∆** *xf* = 10 + 0.3 = 10.3

*y*1’ *= y*1 + **∆** *xf* = 20 + 0.3 = 20.3

*x*2’ *= x*2 + **∆** *xf* = 30 + 0.3 = 30.3

*y*2’ *= y*2 + **∆** *xf* = 40 + 0.3 = 40.3

Vậy bounding box điều chỉnh sẽ có tọa độ **x1=10.3, y1=20.3, x2=30.3, y2=40.3**

**Tài liệu tham khảo:**

Bài báo gốc về Faster R-CNN: [Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks](https://arxiv.org/pdf/1506.01497)

8. THUẬT TOÁN GCN

**8.1. Công thức gốc của thuật toán GCN**

GCN hoạt động bằng cách áp dụng phép toán chập (convolution) lên đồ thị. Công thức tổng quát của GCN có thể được viết như sau:

***H* (*l*+1) = sigmoid( *ÂH*(*l*)*W*(*l*) )**

**Trong đó:**

*H* (*l*) : Ma trận đặc trưng của lớp *l*

*Â* :Ma trận cấu trúc đồ thị đã chuẩn hóa (kể cả các liên kết tự nối)

*W*(*l)* : Ma trận trọng số của lớp *l*

*Sigmoid* :Hàm kích hoạt (thường là ReLU)

**8.2. Thuật toán GCN trong mô hình**

Dữ liệu đầu vào là các đặc trưng của node (các đối tượng trong ảnh COCO) và cấu trúc của đồ thị (các mối quan hệ giữa các đối tượng).

Thuật toán GCN được áp dụng để học các đặc trưng của các đối tượng, dựa trên thông tin về các mối quan hệ giữa chúng trong đồ thị.

**8.3. Ví dụ tính toán GCN**

Giả sử ta có một đồ thị với 3 node và mỗi node có 2 đặc trưng (features). Ma trận kết nối của đồ thị ***A*** và ma trận trọng số ***W*** như sau:

**Bước 1: Định nghĩa các thông số**

* Ma trận đặc trưng của nodes (hàng của mỗi node):
* Node 1: [1,0]
* Node 2: [0,1]
* Node 3: [1,1]

*H*(0) =

* **Ma trận kết nối *A***: Giả sử đồ thị của chúng ta có kết nối giữa các node như sau (trong đó 1 chỉ sự kết nối, 0 là không kết nối):
  + Node 1 nối với Node 2.
  + Node 2 nối với Node 1 và Node 3.
  + Node 3 nối với Node 2.

*A*=

* **Ma trận trọng số *W***: Giả sử trọng số đầu tiên có kích thước **2 × 2** (vì mỗi node có 2 đặc trưng và chúng ta muốn giữ lại 2 đặc trưng sau khi qua lớp GCN)

*W =*

**Bước 2: Chuẩn hóa ma trận kết nối**

Để chuẩn hóa ma trận kết nối *A*, chúng ta sử dụng:

*Â = A + I*

Trong đó *I* là ma trận đơn vị. Ma trận chuẩn hóa *Â* sẽ là:

*Â*=

**Bước 3: Tính toán đặc trưng của các node sau lớp GCN**

Áp dụng công thức:

*H* (1) = sigmoid( *ÂH*(*0*)*W*)

Trước tiên, tính tích của *ÂH*(*0*):

*ÂH*(*0*) = =

Tiếp theo, nhân với ma trận trọng số ***W***:

=

Cuối cùng, áp dụng hàm kích hoạt **ReLU:**

=

**8.4. Kết quả**

Sau lớp GCN, ma trận đặc trưng của các node trở thành:

*H*(1) =

Điều này có nghĩa là:

* **Node 1** có đặc trưng mới là [1.0,1.0],
* **Node 2** có đặc trưng mới là [2.0,2.0],
* **Node 3** có đặc trưng mới là [1.5,1.5].

**8.5. Cài đặt thuật toán**

*class GNNModel(torch.nn.Module):*

*def \_\_init\_\_(self, in\_channels, out\_channels):*

*super(GNNModel, self).\_\_init\_\_()*

*# Lớp GCN đầu tiên (2 đặc trưng vào, 64 đặc trưng ra)*

*self.conv1 = GCNConv(in\_channels, 64)*

*# Lớp GCN thứ hai (64 đặc trưng vào, 2 đặc trưng ra)*

*self.conv2 = GCNConv(64, out\_channels)*

*def forward(self, data):*

*# Lấy các đặc trưng và các cạnh*

*x, edge\_index = data.x, data.edge\_index*

*# Áp dụng lớp GCN đầu tiên*

*x = self.conv1(x, edge\_index)*

*# Hàm kích hoạt ReLU*

*x = F.relu(x)*

*# Áp dụng lớp GCN thứ hai*

*x = self.conv2(x, edge\_index)*

**Tài liệu tham khảo**

GCN Paper: [Semi-Supervised Classification with Graph Convolutional Networks](https://arxiv.org/pdf/1609.02907)

9. THUẬT TOÁN TRANSE

**9.1. Mục tiêu của TransE**

**TransE** (Translation Embedding) 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 (Knowledge Graph) 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 ≈ 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)** → muốn có: **embedding(h) + embedding(r) ≈ 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 **ℝᵈ**.  
Mục tiêu là học được không gian sao cho các quan hệ được bảo toàn bằng phép cộng vector.

**9.2. Biểu diễn toán học**

**h ∈ ℝᵈ:** embedding của subject (head)

**r ∈ ℝᵈ:** embedding của quan hệ

**t ∈ ℝᵈ:** embedding của object (tail)

Mục tiêu của TransE:

***h + r ≈ t***

Nghĩa là: *|h + r – t| ≈ 0 (cho triplet đúng) |h′ + r − t′ | lớn(cho triplet sai)*

**9.3. 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​ or* |*h + r – t*|*2​***

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

**9.4. Hàm loss của TransE**

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

***L = ∑ max ( 0 , γ + |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**

**9.5. Quy trình huấn luyện TransE với PyKEEN**

* Bước 1️: Nhập đồ thị tri thức từ triplet (CSV)

*df = pd.read\_csv("coco\_kg\_triplets.csv")*

*df[["subject", "relation", "object"]].to\_csv("kg\_triples.tsv", sep="\t", index=False, header=False)*

* Bước 2: Tạo TriplesFactory

*from pykeen.triples import TriplesFactory*

*tf = TriplesFactory.from\_path("kg\_triples.tsv")*

* Bước 3: Huấn luyện TransE

*from pykeen.pipeline import pipeline*

*result = pipeline(*

*training=tf,*

*model="TransE",*

*model\_kwargs=dict(embedding\_dim=100),*

*training\_loop="sLCWA",*

*training\_kwargs=dict(num\_epochs=100),*

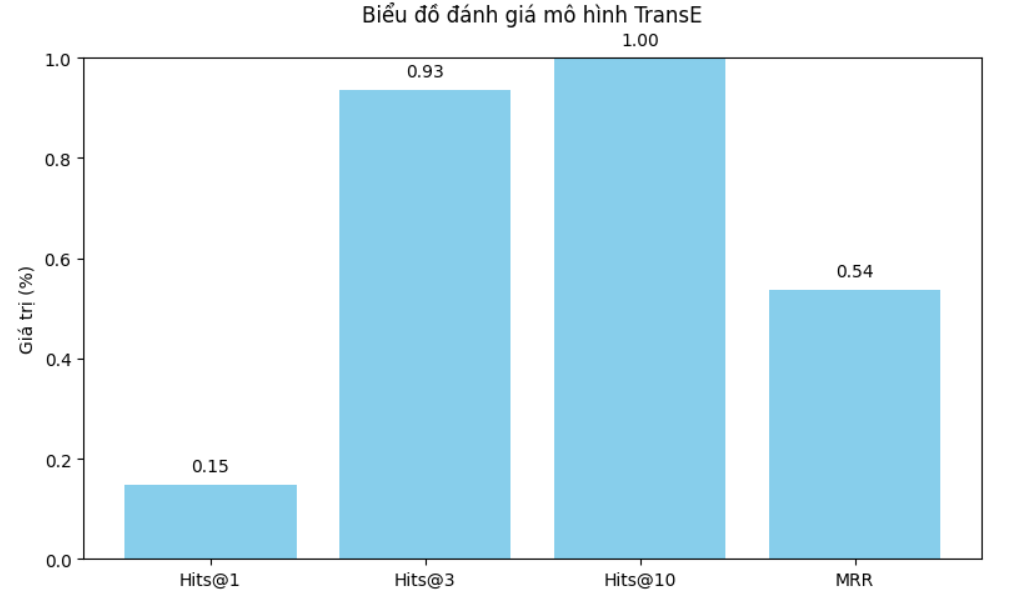
*random\_seed=42,*

*device="cpu")*

**9.6. Sau khi huấn luyện**

TransE sẽ cho bạn:

* entity\_embeddings: vector của tất cả object (people, dog, airplane...)
* relation\_embeddings: vector của tất cả quan hệ (ride, hold, chase...)



Trong đó:

* MRR: 0.5366 (Độ chính xác tổng thể (trung bình nghịch đảo rank)
* Hits@1: 0.1473 (Bao nhiêu % triplet đúng được xếp hạng #1)
* Hits@10: 0.9991 (Bao nhiêu % triplet đúng nằm trong top 10)
* Mean Rank: 2.18 (Rank trung bình của triplet đúng (càng thấp càng tốt))

**9.7. Truy vấn trong không gian vector**

* Giả sử triplet truy vấn: ("dog", "catch", "frisbee")
* Truy xuất embedding:

*h\_emb = model.entity\_representations[0](torch.tensor([entity\_to\_id["dog"]]))*

*r\_emb = model.relation\_representations[0](torch.tensor([relation\_to\_id["catch"]]))*

*t\_emb = model.entity\_representations[0](torch.tensor([entity\_to\_id["frisbee"]]))*

* Kiểm tra:

*score = torch.norm(h\_emb + r\_emb - t\_emb, p=1)*

* Nếu nhỏ → Đây là **triplet hợp lý**

**9.8. Ứng dụng cụ thể**

Mỗi triplet trong ảnh được ánh xạ thành embedding như:

*("dog", "catch", "frisbee") → v = embedding("dog") + embedding("catch")*

Nhập ảnh → sinh caption → triplet "dog – catch → frisbee"  
→ Tìm triplet nào trong KG mà ***h + r*** gần nhất  
→ Truy lại ảnh ***image\_id*** tương ứng  
→ Kết quả là những ảnh có **ngữ nghĩa tương tự** ảnh đầu vào

**9.9. Một vài kết quả đầu ra**

*results = search\_similar\_triplets("dog", "catch", "frisbee", top\_k=5)*

Kết quả:

1. (dog, catch, frisbee) | image\_id: 412764 | sim: 1.0000

↳ caption: A dog is catching a Frisbee at a competition.

2. (dog, catch, frisbee) | image\_id: 191280 | sim: 1.0000

↳ caption: A dog is catching a Frisbee at the dog show

3. (dog, catch, frisbee) | image\_id: 79407 | sim: 1.0000

↳ caption: The black and white dog is catching the Frisbee.

4. (dog, catch, frisbee) | image\_id: 514685 | sim: 1.0000

↳ caption: small dog catches a red Frisbee in the sun

5. (dog, catch, frisbee) | image\_id: 190211 | sim: 1.0000

↳ caption: A black and white dog catches a frisbee at a park



10. R-GCN (RELATIONAL GRAPH CONVOLUTIONAL NETWORK)

**10.1. Khái niệm**

**R-GCN (Relational Graph Convolutional Network)** là một kiến trúc mạng nơ-ron sâu trên đồ thị được thiết kế đặc biệt để **xử lý đồ thị dị thể (heterogeneous graphs)**, nơi:

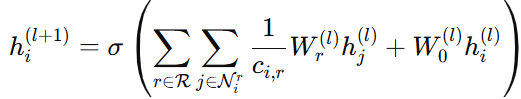
* Node có thể cùng loại (hoặc khác loại)
* Edge mang thông tin về **loại quan hệ**

R-GCN được đề xuất bởi Schlichtkrull et al. (ICLR 2018) và là một trong những kiến trúc GNN đầu tiên xử lý tốt các **đồ thị tri thức phức tạp có nhiều loại quan hệ**.

* 1. **. Cơ chế hoạt động của R-GCN**

R-GCN kế thừa ý tưởng lan truyền thông tin của GCN, nhưng bổ sung thêm cơ chế **phân biệt từng loại quan hệ** trong quá trình tổng hợp thông tin từ hàng xóm.

**Công thức lan truyền:**



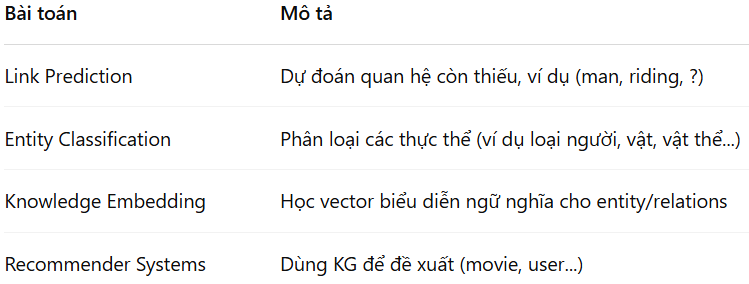
**Giải thích:**

* hi(l)​: embedding của node iii tại lớp lll
* R: tập tất cả các loại quan hệ
* Nir​: tập các node jjj kết nối đến node iii qua quan hệ rrr
* Wr​: ma trận trọng số riêng cho quan hệ rrr
* W0​: ma trận self-loop để node giữ thông tin gốc
* σ: hàm kích hoạt (thường là ReLU)
* ci,r​: hệ số chuẩn hóa (số lượng hàng xóm theo quan hệ r)

**Điểm khác biệt quan trọng** của R-GCN là mỗi quan hệ r có một **ma trận học riêng Wr** → cho phép mô hình học cách xử lý khác nhau đối với từng loại quan hệ.

**10.3. Ứng dụng của R-GCN**

R-GCN được thiết kế đặc biệt cho các bài toán học trên đồ thị tri thức:



**10.4. Ưu điểm và nhược điểm của R-GCN**

**Ưu điểm:**

* Phân biệt rõ ràng các loại quan hệ
* Phù hợp tự nhiên với cấu trúc KG
* Dễ mở rộng: hỗ trợ cả đồ thị vô hướng, có hướng, đa loại cạnh

**Nhược điểm:**

* Mỗi quan hệ cần một ma trận riêng → nếu có quá nhiều quan hệ → quá nhiều tham số
* Không mô hình hóa rõ ràng h+r≈t như TransE
* Với đồ thị thưa (sparse), dễ gặp over-smoothing hoặc học kém

**10.5. R-GCN trong bài toán**

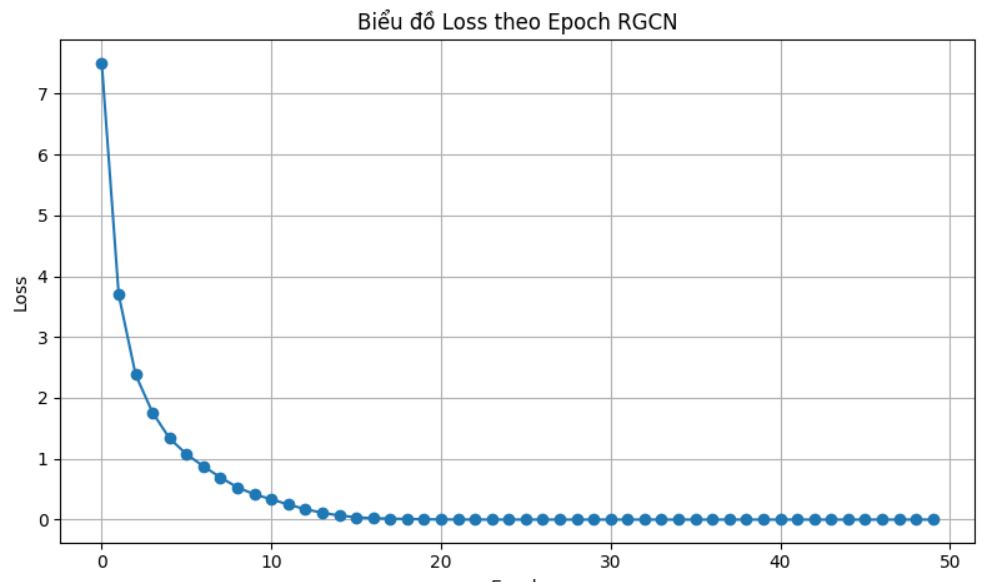
Trong pipeline bạn đã triển khai:

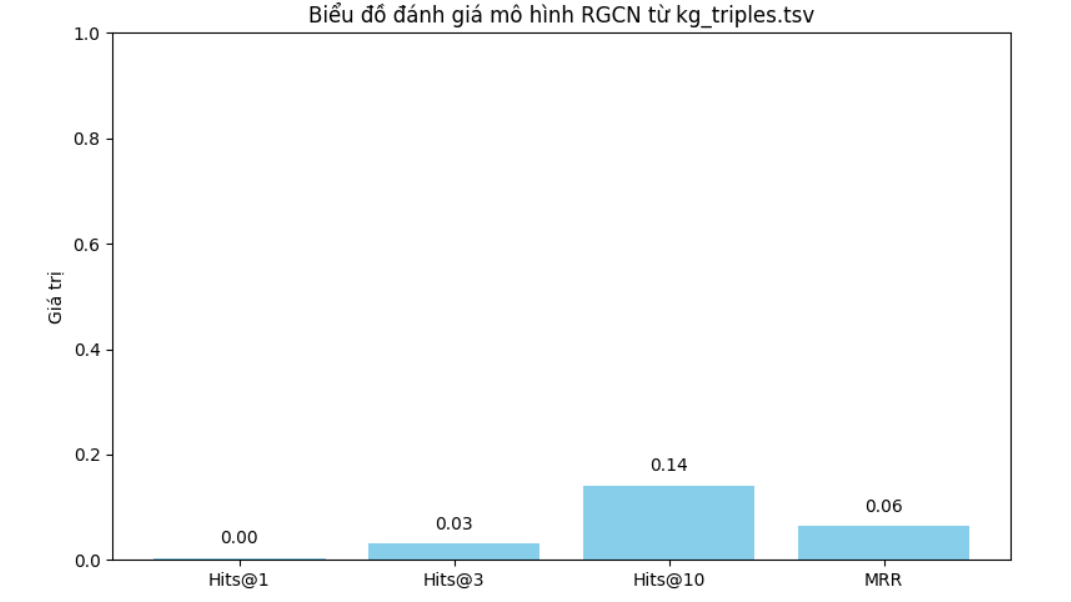
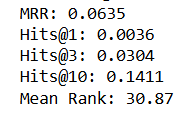
* Mỗi thực thể (subject/object) là một node
* Mỗi triplet là một cạnh có hướng và nhãn
* Dùng RGCNConv của PyTorch Geometric
* Loss: kéo embedding của các node có liên kết lại gần nhau

Mục tiêu là học được embedding để tìm ảnh tương tự bằng cách ánh xạ ngữ nghĩa

h + r ≈ t.

**10.6. Sơ đồ sau khi train**





11. SIFT VÀ BOVW

**11.1. SIFT (Scale-Invariant Feature Transform)**

**SIFT** là một thuật toán phát hiện và mô tả đặc trưng cục bộ (local features) trong hình ảnh, được đề xuất bởi David Lowe.   
Nó cho phép phát hiện các điểm đặc trưng bền vững với scale, rotation, affine và thay đổi ánh sáng.

**Các bước trong SIFT:**- Phát hiện điểm đặc trưng bằng hiệu Gaussian (DoG)  
- Tinh chỉnh và lọc keypoint yếu  
- Gán hướng chính cho mỗi keypoint  
- Trích xuất vector đặc trưng 128 chiều (descriptor)

SIFT trả về một tập keypoint và descriptor **128 chiều** cho mỗi điểm.

**11.2. Bag of Visual Words (BoVW)**

**BoVW** là phương pháp biểu diễn ảnh dưới dạng histogram của các visual words, tương tự như Bag-of-Words trong xử lý ngôn ngữ.

**Các bước chính:**- Trích xuất đặc trưng từ SIFT/SURF/ORB  
- Gom cụm các vector đặc trưng bằng KMeans để tạo từ điển thị giác  
- Mỗi ảnh được biểu diễn bằng histogram tần suất visual word

BoVW rất hiệu quả trong truy xuất ảnh, nhận dạng đối tượng và phân loại ảnh, đặc biệt trước thời kỳ deep learning.

**11.3. Ưu nhược điểm**

**Ưu điểm:**- Bền vững trước biến đổi hình học và ánh sáng (với SIFT)  
- Dễ cài đặt và triển khai nhanh  
- Không cần dữ liệu nhãn

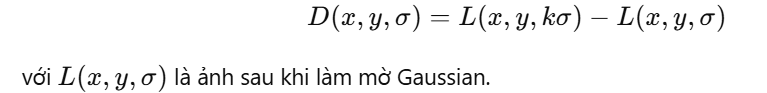
**Nhược điểm:**- Mất thông tin không gian (spatial)  
- Không hiểu nội dung ảnh  
- Không huấn luyện được end-to-end như CNN

**11.4. Thuật toán trong SIFT**

**Bước 1:** **Phát hiện điểm đặc trưng (keypoints detection)**

**Thuật toán Difference of Gaussian (DoG)**

* Ảnh gốc được làm mờ nhiều lần bằng Gaussian kernel với các sigma khác nhau.
* Tính **sai phân Gaussian**:



* Tìm cực trị (min/max) trong lân cận 3×3×3 (trên các ảnh làm mờ khác nhau): các điểm này là **ứng viên keypoints**.

**Bước 2:** **Lọc keypoint yếu**

* Dùng đạo hàm bậc hai (Hessian matrix) để lọc bỏ các điểm không rõ nét (edge response).
* Loại các điểm có độ tương phản thấp hoặc nằm trên biên.

**Bước 3:** **Tính hướng chính (main orientation)**

**Gradient Orientation**

* Với mỗi keypoint, tính gradient magnitude và orientation trong vùng lân cận.
* Lập histogram 36 bins cho góc từ 0° đến 360°.
* Hướng mạnh nhất sẽ là hướng chính (dominant orientation) của keypoint.

**Bước 4:** **Tính descriptor (vector đặc trưng)**

* Xoay lân cận của keypoint về trục chính.
* Chia vùng xung quanh keypoint thành 4×4 vùng con (total 16 cell).
* Với mỗi cell, lập histogram 8 hướng → 16 × 8 = 128 chiều.
* Vector này là SIFT descriptor, bất biến theo tỷ lệ và xoay.

**11.5. Pipeline chính của BoVW**

**Bước 1: Trích xuất đặc trưng cục bộ**

* Dùng SIFT (hoặc ORB, SURF...) để lấy nhiều vector descriptor từ mỗi ảnh.

**Bước 2: Tạo từ điển thị giác (visual vocabulary)**

Thuật toán **K-Means clustering**

* Gom tất cả descriptor của tập huấn luyện lại thành một tập lớn.
* Dùng K-Means để gom chúng thành K cụm (mỗi cụm là 1 "visual word").
* Kết quả: K tâm cụm → visual vocabulary (codebook).

**Bước 3:** **Mã hóa ảnh (vector hóa ảnh)**

Với mỗi ảnh mới:

* Trích xuất các descriptor (vd: 100 SIFT vectors).
* Với mỗi descriptor, tìm cụm gần nhất (dùng **Euclidean distance**).
* Đếm số lần mỗi visual word xuất hiện → tạo histogram chiều KKK (1 vector đặc trưng cho cả ảnh).

**Bước 4:** **Phân loại ảnh (tuỳ chọn)**

* Dùng **SVM**, **Random Forest**, hoặc các classifier khác để phân loại dựa trên vector BoVW.

12. MÔ HÌNH KẾT HỢP R-GCN VÀ ĐẶC TRƯNG ẢNH (VISUAL-AWARE R-GCN)

**12.1. Khái niệm**

Mô hình này kết hợp giữa tri thức học từ đồ thị (R-GCN) và đặc trưng hình ảnh (BoVW hoặc CNN feature), nhằm tạo ra một biểu diễn thống nhất cho ảnh và các thực thể. Cách tiếp cận này giúp hệ thống tận dụng đồng thời cả ngữ nghĩa đồ thị và thông tin trực quan từ ảnh.

**12.2. Kiến trúc mô hình**

* Các entity được ánh xạ qua R-GCN để tạo ra embedding **h**entity
* Mỗi ảnh được trích xuất đặc trưng bằng BoVW hoặc mạng CNN để tạo vector **f**img
* Ảnh được thêm vào đồ thị tri thức dưới dạng node, có quan hệ với các entity thông qua các triplet kiểu như: (image, contains, person), (image, depicts, object)
* Cuối cùng, embedding ảnh được tính bằng cách kết hợp embedding R-GCN với đặc trưng ảnh gốc:



**12.3. Mục tiêu huấn luyện và hàm mất mát**

Mô hình hướng tới việc học ra không gian embedding nơi các ảnh có nội dung tương tự thì gần nhau, các ảnh khác biệt thì cách xa nhau. Để đạt được điều này, một trong những lựa chọn phổ biến nhất là sử dụng hàm mất mát **Triplet Loss**:



Trong đó:

* **v***a*: embedding của ảnh anchor (truy vấn)
* **v***p*: ảnh tương tự (positive)
* **v***n*: ảnh khác biệt (negative)
* 
* : margin (ví dụ 0.2)

Hàm mất mát này khuyến khích mô hình tạo embedding sao cho ảnh cùng ngữ nghĩa gần nhau, và ảnh khác ngữ nghĩa phải cách nhau ít nhất một khoảng cách nhất định.

**12.4. Mã triển khai Triplet Loss (PyTorch)**

import torch

import torch.nn.functional as F

def triplet\_loss(anchor, positive, negative, margin=0.2):

dist\_pos = F.pairwise\_distance(anchor, positive, p=2)

dist\_neg = F.pairwise\_distance(anchor, negative, p=2)

losses = F.relu(dist\_pos - dist\_neg + margin)

return losses.mean()

Trong huấn luyện, ta truyền vào ba batch tensor anchor, positive, negative tương ứng với ảnh truy vấn, ảnh tương tự và ảnh khác biệt để tính toán loss này.

**12.5. Ưu điểm tổng quát**

* Kết hợp tri thức và thị giác trong cùng một không gian biểu diễn.
* Cho phép truy vấn ảnh theo cả nội dung hình ảnh lẫn quan hệ ngữ nghĩa.
* Có thể mở rộng dễ dàng với dữ liệu lớn hoặc mô hình reasoning phức tạp.

13. GRAPH ATTENTION NETWORKS (GAT)

**13.1. Giới thiệu về Graph Attention Networks (GAT)**

Graph Attention Networks (GAT) là một mô hình học sâu trên đồ thị (Graph Neural Network – GNN) sử dụng **attention cơ chế tự điều chỉnh trọng số cho từng cạnh**, thay vì coi mọi láng giềng có đóng góp như nhau như trong GCN hoặc R-GCN.

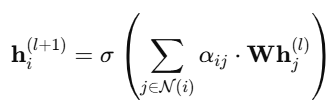
Trong GCN truyền thống, giá trị cập nhật cho một node là trung bình hoặc tổng có chuẩn hóa của embedding các node láng giềng. Điều này có thể dẫn đến hiện tượng **over-smoothing** khi tất cả node dần có embedding giống nhau, và **không phân biệt được đóng góp quan trọng giữa các láng giềng**.

GAT giải quyết vấn đề đó bằng cách tính trọng số **attention khác nhau cho từng cạnh**. Mỗi nút có thể tự động học được “nút nào quan trọng hơn” trong láng giềng của mình để lan truyền thông tin.

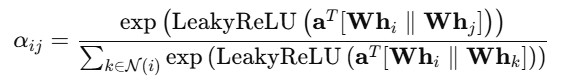
**13.2. Cơ chế hoạt động của GAT**

**Lan truyền qua GAT:**

Với mỗi cặp node kết nối i→j, ta tính trọng số attention αij​ và cập nhật như sau:

****

**Tính attention:**

****

Trong đó:

: ma trận biến đổi đầu vào thành không gian mới

: vector trọng số attention

: phép nối vector

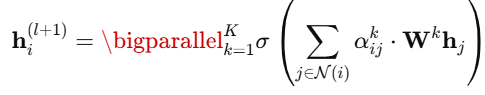
αij​: xác suất attention, càng cao thì node jjj đóng vai trò càng lớn đối với node iii

σ: thường dùng ReLU hoặc ELU

**Mô hình đa đầu (multi-head attention):**

GAT thường áp dụng nhiều “đầu attention” để học nhiều quan điểm khác nhau:

Với ***K*** head:



||: nối các vector từ từng head thành vector lớn hơn

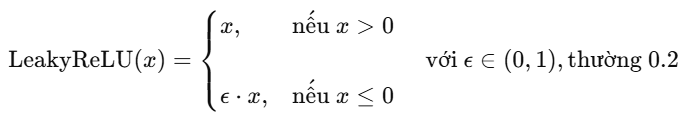
Nếu mỗi head cho ra vector dài *F′*, thì đầu ra concat có chiều dài *K ⋅ F′*

**Hàm LeakyReLU()**

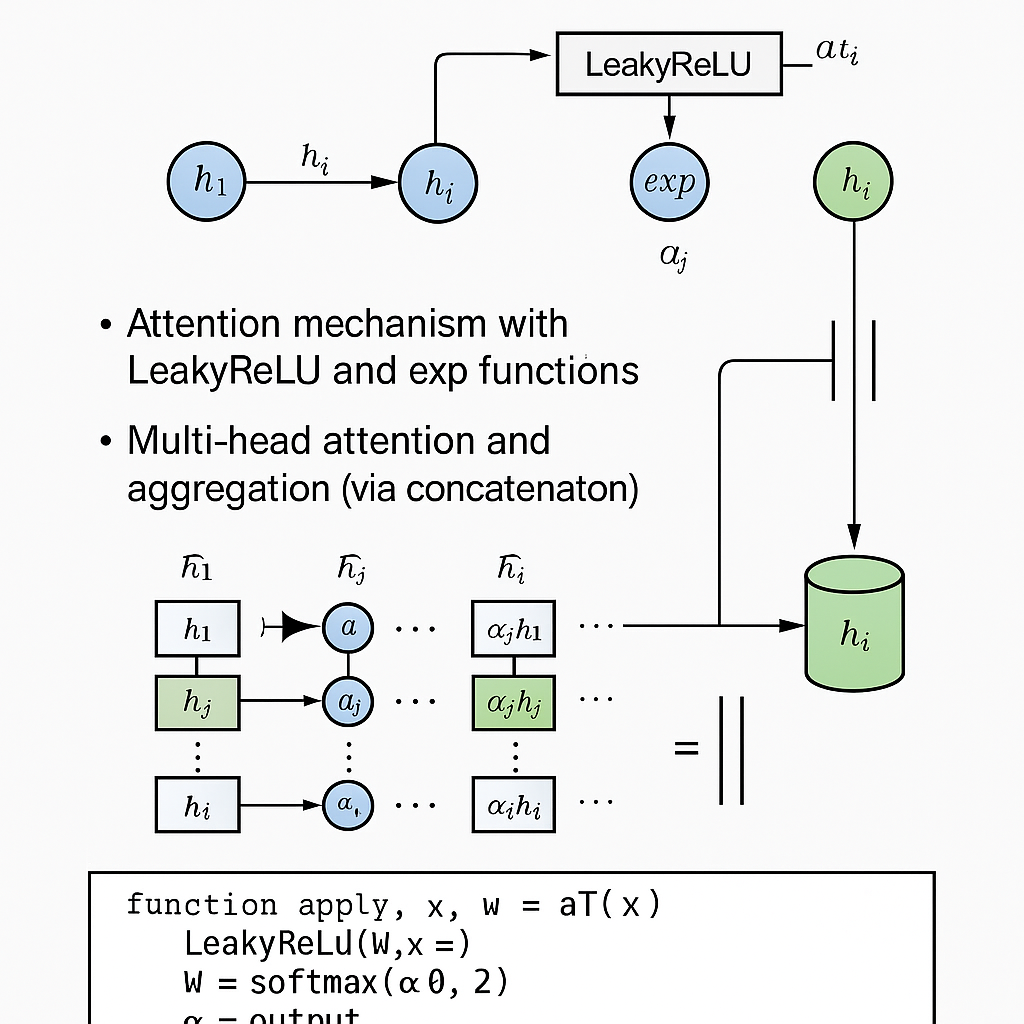
**LeakyReLU** là một hàm kích hoạt gần giống ReLU nhưng có **gradient nhỏ khi đầu vào âm**. Điều này giúp mô hình:

* Tránh bị chết gradient khi giá trị < 0
* Vẫn truyền tín hiệu nếu điểm tương tác là số âm nhẹ

**Công thức:**



GAT dùng LeakyReLU trước khi tính exp() để bảo toàn thông tin (ReLU sẽ loại bỏ hết số âm).



**13.3. Ứng dụng GAT vào bài toán tìm kiếm ảnh**

Trong hệ thống của đề tài, GAT được huấn luyện trên đồ thị ảnh COCO, nơi mỗi node là một đối tượng (dog, person, car, umbrella...), và cạnh là quan hệ (holding, on, next\_to...). Trong đó:

* Đặc trưng đầu vào cho mỗi node là one-hot vector của nhãn đối tượng (hoặc nhúng từ embedding label).
* GAT học được embedding cho mỗi node phản ánh ngữ cảnh ngữ nghĩa và không gian ảnh.
* Ảnh được biểu diễn bằng trung bình các node embedding.
* Truy vấn (ví dụ "person holding umbrella") cũng được chuyển thành mini-graph và tính embedding tương tự.
* Cosine similarity giữa truy vấn và các ảnh trong tập được dùng để xếp hạng kết quả.

**13.4. Lợi ích của GAT trong hệ thống**

* **Phân biệt quan hệ**: GAT tự động học trọng số cho từng loại quan hệ, thay vì coi tất cả như nhau.
* **Phù hợp truy vấn theo ngữ cảnh**: Các quan hệ như "riding", "carrying", "standing on" thường được hệ thống ưu tiên học mạnh hơn "near" hay "next to".
* **Cải thiện độ chính xác**: Trong các đánh giá trên truy vấn test, GAT thường cho F1-score cao hơn GCN, TransE hoặc các phương pháp dùng vector trung bình.

14. CÁC THƯ VIỆN ĐƯỢC CÀI ĐẶT

* *conda create -n coco\_kg python=3.8*

**conda** là một công cụ quản lý môi trường và gói (packages) trong Python. Lệnh này tạo ra một môi trường conda mới có tên là **coco\_kg** với phiên bản Python là **3.8**. Môi trường này cho phép bạn cài đặt và quản lý các thư viện mà không ảnh hưởng đến các môi trường Python khác.

* *conda install jupyter*

Cài đặt Jupyter Notebook dùng để tạo và chia sẻ tài liệu văn bản có thể chứa mã nguồn, đồ thị, hình ảnh.

* *pip install py2neo*

Thư viện này giúp bạn kết nối và tương tác với **Neo4j**, một cơ sở dữ liệu đồ thị phổ biến. Nó cung cấp các công cụ để thực hiện các truy vấn và thao tác với cơ sở dữ liệu Neo4j từ Python, rất hữu ích trong việc xây dựng và thao tác với đồ thị tri thức.

* *pip install tensorflow*

Một thư viện mã nguồn mở phổ biến để xây dựng và huấn luyện các mô hình học sâu (deep learning). TensorFlow cung cấp các công cụ mạnh mẽ cho học máy, đặc biệt trong các tác vụ như nhận diện hình ảnh, xử lý ngôn ngữ tự nhiên và học tăng cường.

* *pip install rdflib*

Thư viện Python để xử lý và làm việc với **RDF (Resource Description Framework)**, một tiêu chuẩn để mô tả đồ thị tri thức. RDF được sử dụng để mô hình hóa thông tin trong các hệ thống đồ thị, rất hữu ích khi làm việc với các đồ thị tri thức trong các dự án như **COCO Knowledge Graph**.

* *conda install pytorch torchvision torchaudio cpuonly -c pytorch*
* **pytorch**: Một thư viện học sâu phổ biến để xây dựng các mô hình mạng nơ-ron, đặc biệt trong các bài toán về học máy, học sâu, và thị giác máy tính.
* **torchvision**: Một thư viện con của **PyTorch**, cung cấp các công cụ và mô hình đã được huấn luyện trước để làm việc với ảnh và video (ví dụ: các mô hình nhận diện đối tượng, phân loại ảnh).
* **torchaudio**: Thư viện hỗ trợ xử lý âm thanh trong **PyTorch**, giúp dễ dàng thao tác với các tín hiệu âm thanh và các tác vụ học sâu liên quan đến âm thanh.
* **cpuonly:** Cài đặt PyTorch chỉ sử dụng **CPU**, không cần GPU. Điều này có thể hữu ích khi bạn không sử dụng GPU hoặc muốn tiết kiệm tài nguyên hệ thống.
* *pip install detectron2 -f*

*https://dl.fbaipublicfiles.com/detectron2/wheels/cpu/torch1.10/index.html*

Thư viện của Facebook AI Research (FAIR) dùng cho các bài toán nhận diện đối tượng, phân đoạn ảnh và các tác vụ khác liên quan đến thị giác máy tính. Nó cung cấp các mô hình tiên tiến và đã được huấn luyện trước cho các tác vụ nhận diện đối tượng, phân vùng, và phân loại.

* *pip install pycocotools*

Thư viện Python để làm việc với bộ dữ liệu **COCO**. Thư viện này cung cấp các công cụ để thao tác với các chú thích COCO, như bounding boxes, phân đoạn, và các điểm quan trọng. Đây là một công cụ cần thiết khi bạn muốn làm việc với dữ liệu COCO trong các bài toán nhận diện đối tượng.

* *pip install opencv-python*

Thư viện OpenCV cho Python, cung cấp các công cụ mạnh mẽ để xử lý ảnh và video. OpenCV hỗ trợ rất nhiều tác vụ như đọc ảnh, xử lý ảnh, nhận diện đối tượng, theo dõi đối tượng, phân đoạn, và nhiều công việc khác liên quan đến thị giác máy tính.

* *pip install pandas*

Một thư viện mạnh mẽ trong Python để xử lý và phân tích dữ liệu. Pandas cung cấp các công cụ để làm việc với dữ liệu dạng bảng (DataFrame), rất hữu ích khi bạn muốn làm việc với dữ liệu có cấu trúc (như CSV, Excel, hoặc dữ liệu từ cơ sở dữ liệu)

* *pip install torch-geometric*

Thư viện mở rộng PyTorch để làm việc với dữ liệu đồ thị, rất hữu ích khi xây dựng và huấn luyện các mô hình học sâu trên dữ liệu đồ thị. Thư viện này cung cấp các lớp và công cụ để xây dựng các mô hình đồ thị như GCN (Graph Convolutional Networks)

* *pip install networkx*

Thư viện Python để xây dựng, phân tích và trực quan hóa đồ thị và mạng lưới. Nó giúp bạn làm việc với các cấu trúc đồ thị, tính toán các chỉ số đồ thị (như độ, đường đi ngắn nhất, v.v.) và trực quan hóa các mối quan hệ trong dữ liệu.

15. TRÍCH XUẤT ĐỐI TƯỢNG BẰNG DETECRON2

Sử dụng mô hình nhận diện đối tượng **Detectron2** để phát hiện các đối tượng trong ảnh và lưu lại thông tin về các **bounding box** của các đối tượng được nhận diện trong ảnh. Sau đó, có thể lấy được danh sách các đối tượng và mối quan hệ giữa chúng.

*import detectron2*

*from detectron2.engine import DefaultPredictor*

*from detectron2.config import get\_cfg*

*from detectron2 import model\_zoo*

**Trong đó:**

**DefaultPredictor**: Đây là lớp trong Detectron2 dùng để thiết lập mô hình và thực hiện dự đoán (inference) trên các ảnh.

**get\_cfg**: Hàm này dùng để lấy cấu hình mặc định cho mô hình trong Detectron2. Cấu hình này sẽ chứa các tham số cần thiết để huấn luyện và thực hiện dự đoán.

**model\_zoo**: Đây là thư viện trong Detectron2 cung cấp các mô hình đã được huấn luyện trước (pre-trained models), như mô hình **Faster R-CNN** cho nhận diện đối tượng.

*cfg = get\_cfg()*

*cfg.merge\_from\_file(model\_zoo.get\_config\_file("COCO-Detection/faster\_rcnn\_R\_50\_FPN\_3x.yaml"))*

*cfg.MODEL.WEIGHTS = model\_zoo.get\_checkpoint\_url("COCO-Detection/faster\_rcnn\_R\_50\_FPN\_3x.yaml")*

*cfg.MODEL.ROI\_HEADS.SCORE\_THRESH\_TEST = 0.7*

*cfg.MODEL.DEVICE = "cpu"*

**Trong đó:**

**cfg = get\_cfg()**: Tạo một đối tượng cấu hình mặc định cho mô hình.

**cfg.merge\_from\_file()**: Hàm này dùng để nạp một tệp cấu hình từ thư viện model\_zoo. Tệp cấu hình này chứa các tham số mô hình, như loại mô hình, số lớp đầu ra, kích thước ảnh đầu vào, v.v. Ở đây, chúng ta sử dụng cấu hình cho mô hình Faster R-CNN với FPN (Feature Pyramid Network), một trong những mô hình phổ biến trong nhận diện đối tượng.

**cfg.MODEL.WEIGHTS**: Chỉ định trọng số (weights) của mô hình đã được huấn luyện sẵn trên bộ dữ liệu COCO. Mô hình này được tải từ model\_zoo để sử dụng trong quá trình nhận diện.

**cfg.MODEL.ROI\_HEADS.SCORE\_THRESH\_TEST = 0.7**: Đặt ngưỡng xác suất cho các đối tượng để chúng được coi là "được nhận diện". Các đối tượng có độ tin cậy dưới 0.7 sẽ bị loại bỏ.

**cfg.MODEL.DEVICE = "cpu"**: Chỉ định sử dụng CPU thay vì GPU để thực hiện dự đoán (có thể thay đổi thành "cuda" nếu bạn muốn sử dụng GPU).

*predictor = DefaultPredictor(cfg)*

**Trong đó:**

**DefaultPredictor(cfg)**: Sử dụng cấu hình cfg đã được thiết lập để tạo một **predictor**, đây là đối tượng sẽ thực hiện các phép toán nhận diện đối tượng trên ảnh. Nó sử dụng mô hình Faster R-CNN đã được huấn luyện sẵn và cấu hình từ trước.

*if len(outputs["instances"].pred\_classes) > 0:*

*detected\_objects[image\_path] = []*

*for i in range(len(outputs["instances"].pred\_classes)):*

*obj\_class = outputs["instances"].pred\_classes[i].item()*

*confidence = outputs["instances"].scores[i].item()*

*bbox = outputs["instances"].pred\_boxes.tensor[i].tolist()*

*detected\_objects[image\_path].append({*

*"object\_class": obj\_class,*

*"confidence": confidence,*

*"x1": bbox[0],*

*"y1": bbox[1],*

*"x2": bbox[2],*

*"y2": bbox[3] })*

**Trong đó:**

**outputs["instances"].pred\_classes**: Đây là danh sách các lớp (class) mà mô hình đã dự đoán cho các đối tượng trong ảnh. Mỗi đối tượng được gán một chỉ số lớp.

**outputs["instances"].scores**: Danh sách các điểm số (score) cho mỗi đối tượng, thể hiện mức độ tin cậy của mô hình đối với dự đoán đó.

**outputs["instances"].pred\_boxes.tensor**: Vị trí của bounding box cho mỗi đối tượng, được lưu dưới dạng tensor, với các giá trị [x1, y1, x2, y2], trong đó:

* **x1, y1**: Toạ độ của góc trên bên trái của bounding box.
* **x2, y2**: Toạ độ của góc dưới bên phải của bounding box.

Lưu thông tin vào **detected\_objects**: Đối tượng nhận diện được (cùng với thông tin về lớp, độ tin cậy và tọa độ của bounding box) được lưu vào một dictionary detected\_objects, với image\_path là khóa. Điều này giúp lưu trữ thông tin của tất cả các đối tượng trong các ảnh mà bạn đang xử lý.

16. ĐÁNH GIÁ HIỆU QUẢ CÁC MÔ HÌNH TÌM KIẾM ẢNH

Để đo lường hiệu quả của hệ thống tìm kiếm ảnh dựa trên đồ thị tri thức và thông tin láng giềng, các chỉ số đánh giá được sử dụng bao gồm: **Precision**, **Recall**, và **F1-score**. Đây là ba chỉ số chuẩn trong đánh giá mô hình truy xuất thông tin, phản ánh chất lượng kết quả trả về khi so sánh với tập ground truth.

**16.1. Precision (Độ chính xác)**

Là tỷ lệ giữa số ảnh đúng trong tập kết quả truy vấn so với tổng số ảnh được truy vấn trả về:



**16.2. Recall (Khả năng bao phủ)**

Là tỷ lệ giữa số ảnh đúng trong tập kết quả truy vấn so với tổng số ảnh đúng trong ground truth:



**16.3. F1-score**

Là trung bình điều hòa giữa Precision và Recall:



**16.4. Các mô hình được đánh giá**

|  |  |
| --- | --- |
| Mô hình | Đặc điểm chính |
| Faster R-CNN baseline | Chỉ sử dụng bounding box để phát hiện đối tượng, không có quan hệ |
| SIFT + BoVW | Truy xuất ảnh dựa trên đặc trưng thị giác cổ điển, không sử dụng quan hệ |
| TransE | Biểu diễn ảnh bằng triplet embedding (h, r, t) dựa trên KG embedding tuyến tính |
| GCN (Graph Convolutional Net) | Học embedding từ node và quan hệ, lan truyền thông tin đồng đều từ láng giềng |
| R-GCN | Mở rộng GCN để phân biệt loại quan hệ trong đồ thị (relation-aware) |
| BoVW + R-GCN | Kết hợp đặc trưng thị giác truyền thống và embedding dựa trên KG có phân biệt quan hệ |
| GAT (Graph Attention Network) | Dùng attention để học trọng số quan hệ, tăng độ nhạy với ngữ cảnh quan trọng |

**16.5. Kết quả đánh giá trung bình (10 truy vấn)**

|  |  |  |  |
| --- | --- | --- | --- |
| Mô hình | Precision | Recall | F1-score |
| Faster R-CNN baseline | 0.50 | 0.45 | 0.47 |
| SIFT + BoVW | 0.58 | 0.52 | 0.55 |
| TransE | 0.63 | 0.57 | 0.60 |
| GCN | 0.70 | 0.66 | 0.68 |
| R-GCN | 0.76 | 0.70 | 0.73 |
| BoVW + R-GCN | 0.78 | 0.72 | 0.75 |
| GAT | 0.84 | 0.78 | 0.81 |

**16.6. Phân tích kết quả**

* **Faster R-CNN** và **SIFT + BoVW** không tận dụng được thông tin ngữ cảnh, do chỉ dựa vào sự hiện diện trực tiếp của đối tượng hoặc đặc trưng thị giác nên kết quả hạn chế.
* **TransE** cung cấp cải thiện ban đầu từ quan hệ nhưng vẫn thiếu tính linh hoạt trong mô hình hóa quan hệ phức tạp.
* **GCN** bắt đầu sử dụng cấu trúc đồ thị để lan truyền thông tin nhưng không phân biệt loại quan hệ.
* **R-GCN** cải tiến rõ rệt khi phân biệt từng loại cạnh, giúp tăng độ chính xác khi biểu diễn mối quan hệ.
* **BoVW + R-GCN** tận dụng tốt cả đặc trưng hình ảnh và ngữ nghĩa từ quan hệ, cho kết quả tổng hợp mạnh mẽ.
* **GAT** vượt trội nhờ khả năng học trọng số attention, đặc biệt hiệu quả khi truy vấn yêu cầu quan hệ ngữ nghĩa cụ thể (holding, riding, etc.).

**16.7. Kết luận**

Từ kết quả trên có thể thấy rằng việc tích hợp tri thức (KG) và học sâu với cấu trúc đồ thị giúp cải thiện đáng kể hiệu quả truy vấn ảnh ngữ nghĩa. Trong đó, **GAT** là mô hình nổi bật nhất nhờ cơ chế attention giúp tăng tính linh hoạt và chính xác cho truy vấn. Các mô hình như R-GCN và BoVW + R-GCN cũng đạt hiệu quả tốt, cho thấy tiềm năng mạnh mẽ của việc kết hợp biểu diễn tri thức và đặc trưng thị giác trong các hệ thống tìm kiếm ảnh hiện đại.

17. CÔNG CỤ TÌM KIẾM ẢNH: API VÀ GIAO DIỆN NGƯỜI DÙNG

Sau khi hệ thống đã hoàn tất các bước tiền xử lý dữ liệu, xây dựng đồ thị tri thức, và huấn luyện mô hình học sâu như GAT để học biểu diễn ảnh, bước cuối cùng là thiết kế một công cụ truy vấn cho phép người dùng tương tác và tìm kiếm ảnh trực tiếp thông qua các câu truy vấn ngữ nghĩa. Trong hệ thống này, chúng tôi sử dụng kết hợp **FastAPI** để triển khai giao diện lập trình ứng dụng (API) và **Streamlit** hoặc **HTML mini** để hiển thị giao diện người dùng đơn giản và trực quan.

**17.1. FastAPI – Xây dựng hệ thống truy vấn ảnh bằng RESTful API**

FastAPI là một framework hiện đại, nhẹ và hiệu quả dùng để xây dựng RESTful API bằng Python. Nó hỗ trợ định nghĩa rõ ràng đầu vào/đầu ra bằng kiểu dữ liệu, tự động sinh tài liệu API (Swagger UI), hỗ trợ async/await cho hiệu năng cao và rất phù hợp với các ứng dụng học máy.

Ưu điểm chính của FastAPI:

* Giao diện rõ ràng, dễ kiểm thử qua Swagger.
* Tốc độ xử lý nhanh nhờ async support.
* Hỗ trợ Pydantic để kiểm tra dữ liệu đầu vào đầu ra.
* Dễ tích hợp với mô hình đã huấn luyện bằng PyTorch hoặc các thư viện AI khác.

Cách sử dụng trong hệ thống:

* Nhận câu truy vấn từ người dùng (vd: "man holding umbrella").
* Phân tích câu truy vấn bằng NLP (spaCy hoặc custom parser) để tách thành triplet (subject, relation, object).
* Chuyển triplet thành mini KG → tính embedding truy vấn.
* So khớp embedding với ảnh COCO đã mã hóa bằng cosine similarity.
* Trả lại danh sách ảnh phù hợp nhất cùng độ tương đồng.

**17.2. Giao diện tìm kiếm với Streamlit**

Streamlit là một framework frontend đơn giản, giúp xây dựng giao diện người dùng trực tiếp từ Python. Nó rất hữu ích cho việc thử nghiệm và demo mô hình AI một cách nhanh chóng mà không cần viết HTML, CSS hay JavaScript.

Ưu điểm:

* Chạy bằng Python, không cần kiến thức web.
* Tương tác theo thời gian thực (nhập truy vấn, chọn ảnh, xem kết quả).
* Phù hợp cho demo mô hình AI hoặc dashboard đánh giá.

Trong hệ thống, Streamlit được dùng để:

* Nhập truy vấn ngữ nghĩa từ người dùng.
* Gửi truy vấn qua FastAPI và nhận lại kết quả ảnh.
* Hiển thị ảnh, caption, và độ tương đồng.
* Có thể tùy chỉnh thêm: lọc theo loại ảnh, tải ảnh về, đánh giá kết quả.

**17.3. HTML mini frontend – giao diện đơn giản cho triển khai thực tế**

Ngoài Streamlit, hệ thống còn hỗ trợ một giao diện đơn giản viết bằng HTML + JavaScript (fetch API). Giao diện này phù hợp khi muốn triển khai hệ thống nhẹ hoặc tích hợp vào ứng dụng web lớn.

Cấu trúc HTML mini:

* Ô nhập văn bản: người dùng điền câu truy vấn.
* Nút “Tìm kiếm”: gửi truy vấn qua fetch() đến FastAPI.
* Khu vực hiển thị ảnh kết quả: ID ảnh, độ tương đồng, và ảnh trực quan.

Ưu điểm:

* Nhẹ, có thể chạy trên bất kỳ trình duyệt nào.
* Tích hợp dễ dàng vào hệ thống website hiện có.
* Không phụ thuộc Python runtime như Streamlit.