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

**MỤC LỤC**

[1. Xác định yêu cầu và mục tiêu 4](#_Toc193470652)

[2. Mô tả cách thu thập dữ liệu, mô tả cấu trúc bộ dữ liệu COCO 4](#_Toc193470653)

[3. Mô tả đồ thị tri thức 5](#_Toc193470654)

[4. Mô tả chi tiết các bước cài đặt đồ thị tri thức cho bộ dữ liệu ảnh, cụ thể là COCO 6](#_Toc193470655)

[Các công cụ phần mềm 6](#_Toc193470656)

[Các bước cài đặt 7](#_Toc193470657)

[5. Trích xuất đối tượng bằng Detecron2 9](#_Toc193470658)

[6. Các bước trong thuật toán Faster R-CNN 12](#_Toc193470659)

[RPN (Region Proposal Network) 12](#_Toc193470660)

[Hồi quy Bounding Box 12](#_Toc193470661)

[Classification and Bounding Box Regression 12](#_Toc193470662)

[Hồi quy Bounding Box 12](#_Toc193470663)

[Kết quả của Faster R-CNN 12](#_Toc193470664)

[Tài liệu tham khảo 14](#_Toc193470665)

[7. Mô tả chi tiết thuật toán, công thức gốc và cách áp dụng thuật toán 14](#_Toc193470666)

[Thuật toán sử dụng: GCN (Graph Convolutional Network) 14](#_Toc193470667)

[Công thức gốc của thuật toán GCN 14](#_Toc193470668)

[Thuật toán GCN trong mô hình 14](#_Toc193470669)

[Ví dụ tính toán GCN 14](#_Toc193470670)

[Kết quả 16](#_Toc193470671)

[Cài đặt thuật toán 16](#_Toc193470672)

[Kết quả trên bộ dữ liệu COCO 17](#_Toc193470673)

[Tài liệu tham khảo 17](#_Toc193470674)

[8. Truy vấn Cypher 17](#_Toc193470675)

[a) Truy vấn tất cả các node có trong đồ thị 17](#_Toc193470676)

[b) Truy vấn tất cả các cạnh có mối quan hệ NEXT TO trong đồ thị 18](#_Toc193470677)

[c) Truy vấn tất cả thuộc tính của node có trong đồ thị 18](#_Toc193470678)

[9. So sánh kết quả độ chính xác của các thuật toán 19](#_Toc193470679)

[10. Mô tả chi tiết kết quả các đối tượng và quan hệ mà em đã thu thập 21](#_Toc193470680)

[Thu thập đối tượng và quan hệ trong COCO 21](#_Toc193470681)

[Cách biểu diễn đồ thị trên Neo4j 22](#_Toc193470682)

[11. Tìm hiểu thêm thuật toán VF2 Algorithm để kiểm tra đồ thị con trong tập dữ liệu (so khớp đồ thị) 24](#_Toc193470683)

[Khái Niệm Subgraph Isomorphism 24](#_Toc193470684)

[Ứng Dụng của VF2 Algorithm 24](#_Toc193470685)

[Cách Hoạt Động của VF2 Algorithm 24](#_Toc193470686)

[Đặc Trưng và Tính Năng Của VF2 Algorithm 25](#_Toc193470687)

[Công Thức và Tính Toán của VF2 25](#_Toc193470688)

[Tài Liệu Tham Khảo VF2 Algorithm 26](#_Toc193470689)

[Ứng Dụng Của VF2 Trong COCO 26](#_Toc193470690)

[12. Mở rộng bộ dữ liệu Flick hoặc Visual Genome 26](#_Toc193470691)

[13. Đánh giá hiệu suất thuật toán tìm kiếm ảnh 28](#_Toc193470692)

# 1. Xác định yêu cầu và mục tiêu

* **Mục tiêu chính**: Tìm kiếm ảnh từ COCO dựa trên mối quan hệ giữa các đối tượng trong ảnh, sử dụng đồ thị tri thức (KG) và thông tin về các đối tượng láng giềng.
* **Ứng dụng thực tế**: Truy vấn ảnh theo ngữ cảnh, tìm kiếm ảnh thông minh, hỗ trợ hệ thống gợi ý hình ảnh.

# 2. Mô tả cách thu thập dữ liệu, mô tả cấu trúc bộ dữ liệu COCO

**Thu thập 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.

* **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.

**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.

# 3. Mô tả đồ thị tri thức

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.

**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".

# 4. Mô tả chi tiết các bước cài đặt đồ thị tri thức cho bộ dữ liệu ảnh, cụ thể là COCO

## Các công cụ phần mềm

1. **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.

1. **Anaconda**: là một nền tảng phân phối Python và R, cung cấp một bộ công cụ mạnh mẽ cho việc cài đặt, quản lý thư viện và môi trường cho các dự án khoa học dữ liệu và học máy. Nó hỗ trợ rất nhiều thư viện phổ biến như NumPy, Pandas, Matplotlib, TensorFlow, PyTorch, và các công cụ khác cần thiết cho việc phân tích và xử lý dữ liệu..

Quản lý môi trường: Anaconda cho phép bạn tạo và quản lý nhiều môi trường ảo (virtual environments) cho từng dự án, giúp bạn sử dụng các phiên bản khác nhau của Python và các thư viện mà không bị xung đột giữa các dự án.

Quản lý gói (package management): Sử dụng conda, Anaconda cho phép bạn cài đặt, cập nhật và quản lý các gói phần mềm (packages) rất dễ dàng. Bạn có thể cài đặt gói từ kênh chính thức Anaconda hoặc từ conda-forge.

Hỗ trợ các công cụ khoa học dữ liệu: Anaconda đi kèm với nhiều công cụ khoa học dữ liệu mạnh mẽ, bao gồm **Jupyter**, Spyder, và nhiều công cụ khác.

Quản lý tài nguyên hệ thống: Anaconda giúp bạn dễ dàng quản lý các phụ thuộc và phiên bản của các thư viện, giúp tiết kiệm thời gian trong việc cài đặt và duy trì môi trường làm việc.

1. **Jupyter Notebook**: là một ứng dụng web mã nguồn mở giúp bạn tạo và chia sẻ tài liệu văn bản có chứa mã nguồn, đồ thị, hình ảnh và văn bản mô tả. Nó đặc biệt hữu ích cho các nghiên cứu trong lĩnh vực khoa học dữ liệu, học máy và phân tích dữ liệu.

Cài đặt và sử dụng dễ dàng: Anaconda đã tích hợp sẵn Jupyter Notebook, giúp bạn dễ dàng cài đặt và sử dụng mà không cần phải lo lắng về các vấn đề tương thích giữa các thư viện.

Hỗ trợ mạnh mẽ cho khoa học dữ liệu: Jupyter tích hợp tốt với các thư viện khoa học dữ liệu như NumPy, Pandas, Matplotlib, TensorFlow, PyTorch... giúp bạn thực hiện phân tích và trực quan hóa dữ liệu ngay trong môi trường notebook.

## Các bước cài đặt

1. Cài đặt **Neo4j**:
   * Cài đặt Neo4j Desktop hoặc Neo4j Community Edition.
   * Khởi động Neo4j và tạo một cơ sở dữ liệu mới.
2. Cài đặt **Jupyter Notebook** trong **Anaconda**:

* *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.

# 5. 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.

# 6. Các bước trong thuật toán Faster R-CNN

## 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

## 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 RegressionSau 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

## 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)

# 7. Mô tả chi tiết thuật toán, công thức gốc và cách áp dụng thuật toán

## Thuật toán sử dụng: GCN (Graph Convolutional Network)

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)

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.

## 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:**

=

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].

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)*

*return x*

Kết quả trên bộ dữ liệu COCO**:**

Độ chính xác (*Accuracy*): **0.6912 (69%)**

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



# 8. Truy vấn Cypher

##### Truy vấn tất cả các node có trong đồ thị

*query\_nodes = """*

*MATCH (n:Object)*

*RETURN n.name AS object\_name, ID(n) AS node\_id*

*"""*

*nodes = query\_neo4j(query\_nodes)*

*print(nodes)*

**Kết quả:** Thu được **68 node**

[Record({'object\_name': 'Unknown\_0', 'node\_id': 68}), Record({'object\_name': 'broccoli', 'node\_id': 69}), Record({'object\_name': 'chair', 'node\_id': 70}), Record({'object\_name': 'mouse', 'node\_id': 71}), Record({'object\_name': 'remote', 'node\_id': 72}), Record({'object\_name': 'tv', 'node\_id': 73}), ... Record({'object\_name': 'traffic light', 'node\_id': 133}), Record({'object\_name': 'umbrella', 'node\_id': 134}), Record({'object\_name': 'zebra', 'node\_id': 135})]

##### Truy vấn tất cả các cạnh có mối quan hệ NEXT TO trong đồ thị

*query\_edges = """*

*MATCH (a:Object)-[:NEXT\_TO]->(b:Object)*

*RETURN ID(a) AS node\_a, ID(b) AS node\_b*

*"""*

*edges = query\_neo4j(query\_edges)*

*print(edges)*

**Kết quả:** thu được tất cả **53 cạnh**

[Record({'node\_a': 68, 'node\_b': 69}), Record({'node\_a': 68, 'node\_b': 73}), Record({'node\_a': 68, 'node\_b': 74}), Record({'node\_a': 68, 'node\_b': 75}), Record({'node\_a': 68, 'node\_b': 76}), Record({'node\_a': 68, 'node\_b': 85}), Record({'node\_a': 68, 'node\_b': 86}), Record({'node\_a': 68, 'node\_b': 92}), Record({'node\_a': 68, 'node\_b': 101}), Record({'node\_a': 68, 'node\_b': 104}), ... Record({'node\_a': 113, 'node\_b': 105}), Record({'node\_a': 117, 'node\_b': 74}), Record({'node\_a': 117, 'node\_b': 76}), Record({'node\_a': 117, 'node\_b': 117}), Record({'node\_a': 121, 'node\_b': 74}), Record({'node\_a': 123, 'node\_b': 68}), Record({'node\_a': 135, 'node\_b': 134})]

##### Truy vấn tất cả thuộc tính của node có trong đồ thị

*query\_node\_features = """*

*MATCH (n:Object)*

*RETURN n.class\_id AS class\_id, n.confidence AS confidence*

*"""*

*node\_features = query\_neo4j(query\_node\_features)*

*print(node\_features)*

**Kết quả:** thu được tất cả **68 đặc trưng**

[Record({'class\_id': 0, 'confidence': 0.996264398097992}), Record({'class\_id': 56, 'confidence': 0.9938883185386658}), Record({'class\_id': 62, 'confidence': 0.9871870279312134}), Record({'class\_id': 74, 'confidence': 0.9214431047439576}), Record({'class\_id': 75, 'confidence': 0.9150474667549132}), Record({'class\_id': 72, 'confidence': 0.9061191082000732}), ... Record({'class\_id': 42, 'confidence': 0.9155292510986328}), Record({'class\_id': 53, 'confidence': 0.956544816493988}), Record({'class\_id': 19, 'confidence': 0.8024840950965881}), Record({'class\_id': 10, 'confidence': 0.9972031116485596}), Record({'class\_id': 28, 'confidence': 0.9241483211517334}), Record({'class\_id': 24, 'confidence': 0.7591325044631958})]

# 9. So sánh kết quả độ chính xác của các thuật toán

Bộ dữ liệu **COCO** là một trong những bộ dữ liệu lớn và phổ biến nhất trong lĩnh vực nhận diện đối tượng và phân đoạn ảnh. COCO bao gồm nhiều nhiệm vụ:

* **Nhận diện đối tượng (Object detection)**: Phát hiện và phân loại đối tượng trong ảnh.
* **Phân đoạn đối tượng (Instance segmentation)**: Phân đoạn đối tượng từng pixel trong ảnh.
* **Captioning**: Mô tả các đối tượng trong ảnh.
* **Keypoint detection**: Phát hiện các điểm quan trọng trên cơ thể người (ví dụ: khuôn mặt, khớp).

Bộ dữ liệu COCO bao gồm hàng triệu đối tượng được gắn nhãn, với khoảng 80 lớp đối tượng khác nhau, bao gồm xe hơi, động vật, người, v.v.

**GCN** **(Graph Convolutional Network)**: Dùng cho việc phân tích dữ liệu dạng đồ thị, đặc biệt hữu ích khi có mối quan hệ giữa các node trong đồ thị.

* GCN có thể được sử dụng trong các bài toán như **công cụ học đồ thị**, **đồ thị tri thức**, và **phân tích mạng xã hội**, nhưng trong trường hợp nhận diện đối tượng, GCN có thể khó áp dụng trực tiếp mà không phải qua bước tiền xử lý phức tạp.

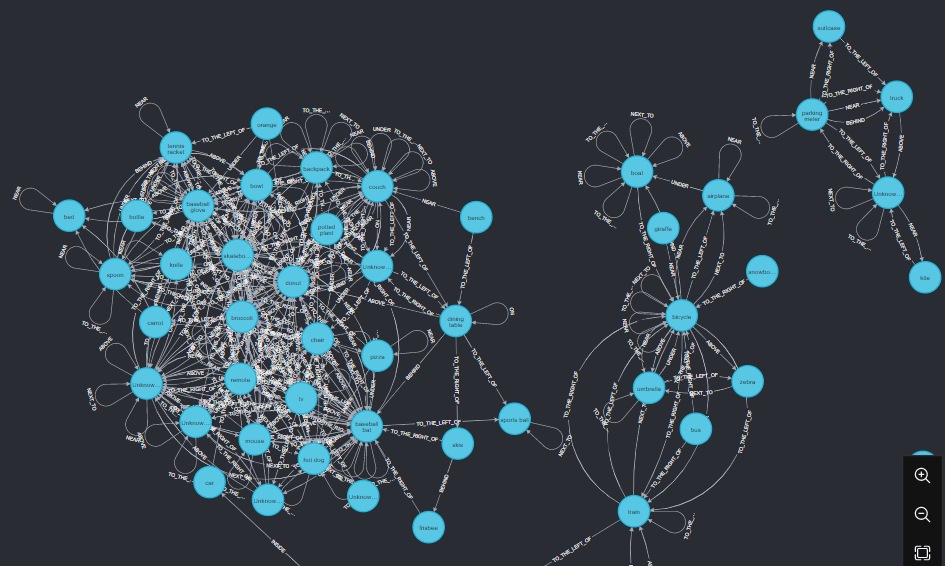
**Faster R-CNN (Region-based Convolutional Neural Network)**: Đây là một trong những mô hình tiên tiến trong nhận diện đối tượng. Faster R-CNN kết hợp các bước **Region Proposal Network (RPN)** để tạo ra các vùng ứng cử và **Fast R-CNN** để phân loại các vùng này.

# 10. Mô tả chi tiết kết quả các đối tượng và quan hệ mà em đã thu thập

## Thu thập đối tượng và quan hệ trong COCO

Bộ dữ liệu COCO chứa thông tin rất phong phú về các đối tượng trong ảnh, các điểm quan trọng và các mối quan hệ giữa chúng. Để thu thập dữ liệu từ COCO, ta có thể sử dụng API hoặc các công cụ hỗ trợ, chẳng hạn như **pycocotools** (thư viện Python cho bộ dữ liệu COCO). Dưới đây là một số thông tin có thể thu thập được từ COCO:

* **Đối tượng (Objects)**:
  + **ID đối tượng**: Mỗi đối tượng có một ID duy nhất.
  + **Tên đối tượng (class)**: Các đối tượng trong COCO thuộc nhiều lớp khác nhau, chẳng hạn như "person", "car", "dog", v.v.
  + **Bounding box**: Là hình chữ nhật bao quanh đối tượng, được xác định bởi tọa độ (x1, y1, x2, y2), trong đó (x1, y1) là góc trên bên trái và (x2, y2) là góc dưới bên phải của box.
  + **Confidence (độ tin cậy)**: Là độ chính xác mà mô hình nhận diện đối tượng đưa ra, phản ánh mức độ tự tin của mô hình trong việc phân loại đối tượng đó.
* **Mối quan hệ (Relationships)**: Các mối quan hệ không gian hoặc logic giữa các đối tượng trong ảnh.
  + Ví dụ:
    - :NEXT\_TO (nằm gần nhau): Chỉ rằng một đối tượng nằm gần một đối tượng khác.
    - :IN (ở trong): Chỉ rằng một đối tượng nằm trong một đối tượng khác, ví dụ "người trong xe".
    - :PART\_OF (một phần của): Ví dụ, "cánh tay là một phần của người".
* **Điểm quan trọng (Keypoints)**: Trong một số trường hợp (ví dụ như trong bài toán nhận diện con người), bộ dữ liệu COCO cũng cung cấp các điểm quan trọng như mắt, mũi, miệng của người.

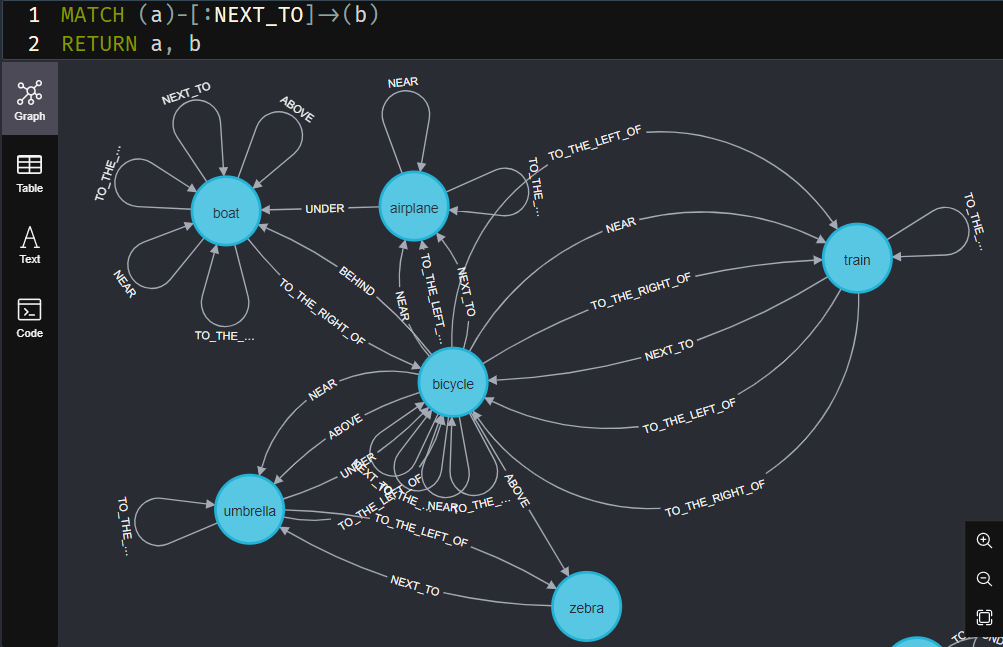


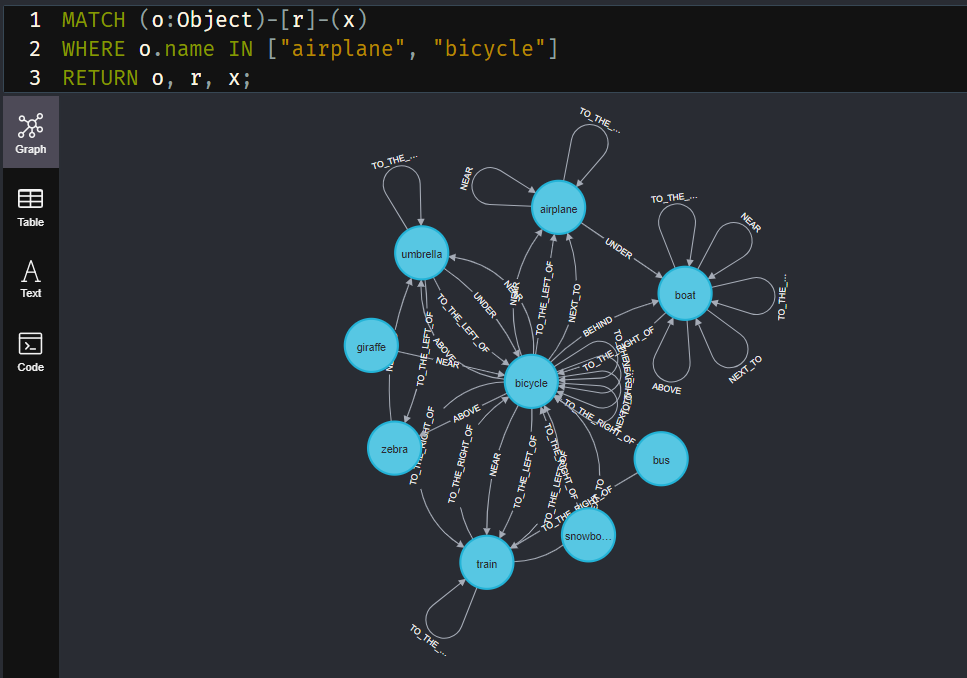
## Cách biểu diễn đồ thị trên Neo4j

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.





# 11. Tìm hiểu thêm thuật toán VF2 Algorithm để kiểm tra đồ thị con trong tập dữ liệu (so khớp đồ thị)

**VF2 Algorithm** là một thuật toán được thiết kế để kiểm tra **subgraph isomorphism**, nghĩa là kiểm tra xem một đồ thị con có tồn tại trong một đồ thị lớn hơn hay không. Thuật toán này được sử dụng phổ biến trong việc so khớp đồ thị, đặc biệt trong các bài toán tìm kiếm các phần tử tương tự trong các cấu trúc đồ thị phức tạp, như trong bài toán nhận diện đối tượng hoặc phân tích mạng xã hội.

## Khái Niệm Subgraph Isomorphism

* **Subgraph isomorphism** là bài toán tìm một phần của đồ thị lớn (đồ thị mẫu) khớp hoàn toàn với một phần của đồ thị tìm kiếm.
* Để xác định hai đồ thị G1G\_1G1​ và G2G\_2G2​ có là **subgraph isomorphic** hay không, nghĩa là xác định xem có một đồ thị con của G1G\_1G1​ mà là bản sao của G2G\_2G2​ không.

Ví dụ: Nếu chúng ta có một đồ thị mẫu với ba node và các cạnh nối giữa chúng, ta muốn xác định xem có tồn tại một phần trong đồ thị lớn có ba node và các cạnh nối giống hệt như vậy không.

## Ứng Dụng của VF2 Algorithm

Thuật toán VF2 rất hữu ích trong các bài toán như:

* **Phân tích mạng xã hội**: Xác định các mẫu tương tự trong mạng xã hội như các nhóm người dùng có mối quan hệ tương tự.
* **Nhận diện hình ảnh và đồ thị**: Xác định các đối tượng trong hình ảnh hoặc video dựa trên các mô hình đồ thị.
* **Phân tích hóa học**: Trong hóa học, thuật toán này có thể được dùng để tìm kiếm các cấu trúc phân tử giống nhau.

## Cách Hoạt Động của VF2 Algorithm

**VF2** là một thuật toán **tìm kiếm sâu** (depth-first search) để so khớp các node và mối quan hệ trong đồ thị. Thuật toán này sử dụng các chiến lược tối ưu hóa như **kiểm tra tính tương thích** và **hạn chế tìm kiếm** để giảm bớt độ phức tạp của bài toán. Thuật toán giúp xác định xem có một đồ thị con của một đồ thị mẫu có thể khớp với đồ thị tìm kiếm hay không.

1. **Các bước chính của VF2:**

* **Khởi tạo**: Xác định một số thuộc tính ban đầu của đồ thị, chẳng hạn như các node, các mối quan hệ, số lượng node và các đặc trưng của chúng.
* **Kiểm tra sự tương thích**: Lọc ra những node không thể khớp với nhau, giảm bớt số trường hợp cần kiểm tra. Sự tương thích giữa hai node có thể được kiểm tra thông qua các thuộc tính của node như kiểu (class) hoặc các thuộc tính liên quan.
* **Tìm kiếm sâu**: Dùng phương pháp tìm kiếm sâu để kiểm tra các node và mối quan hệ giữa chúng. Thuật toán này sẽ cố gắng tìm một sự kết hợp giữa các node và mối quan hệ sao cho các thuộc tính của đồ thị con phù hợp với đồ thị mẫu.
* **Kết thúc**: Nếu một đồ thị con thỏa mãn tất cả các điều kiện và mối quan hệ trong đồ thị mẫu, thuật toán sẽ trả về kết quả là "tìm thấy một đồ thị con isomorphic", nếu không sẽ trả về "không tìm thấy".

1. **Các bước cụ thể trong VF2:**
2. **Duyệt qua các node**: Kiểm tra từng node trong đồ thị mẫu và tìm kiếm node tương ứng trong đồ thị tìm kiếm.
3. **So sánh mối quan hệ**: Đảm bảo rằng các mối quan hệ (edges) giữa các node trong đồ thị mẫu khớp với các mối quan hệ trong đồ thị tìm kiếm.
4. **Quản lý các ràng buộc**: Các node và mối quan hệ phải đáp ứng các ràng buộc về tính tương thích, chẳng hạn như các thuộc tính phải giống nhau hoặc các mối quan hệ giữa chúng phải giống nhau.
5. **Quay lui**: Nếu không tìm thấy sự kết hợp phù hợp, thuật toán sẽ quay lại và thử các node khác hoặc thử các cách kết hợp khác.

## Đặc Trưng và Tính Năng Của VF2 Algorithm

* **Tìm kiếm sâu (Depth-first search)**: Thuật toán VF2 sử dụng phương pháp tìm kiếm sâu để kiểm tra các node và mối quan hệ, giúp kiểm tra tất cả các trường hợp có thể.
* **Giảm bớt tính toán**: Thuật toán sử dụng các chiến lược tối ưu hóa để giảm bớt các tính toán không cần thiết, chẳng hạn như loại bỏ các node không thể khớp.
* **Khả năng mở rộng**: VF2 có thể áp dụng cho các đồ thị phức tạp, với nhiều node và mối quan hệ.
* **Hiệu quả trong bài toán đồ thị con**: VF2 được tối ưu hóa cho các bài toán đồ thị con, đặc biệt là trong các ứng dụng nhận diện đối tượng, phân tích mạng xã hội và các bài toán liên quan đến đồ thị.

## Công Thức và Tính Toán của VF2

Dưới đây là mô tả sơ bộ về các công thức và tính toán trong thuật toán VF2.

**Kiểm tra sự tương thích giữa các node**:  
Thuật toán sẽ kiểm tra tính tương thích giữa các node trong đồ thị mẫu và đồ thị tìm kiếm dựa trên các thuộc tính của chúng. Điều này có thể bao gồm kiểm tra các thuộc tính như:

* **Loại node** (class)
* **Độ kết nối** (degree)
* **Các thuộc tính bổ sung** (ví dụ: màu sắc, hình dạng, v.v.)

**Kiểm tra mối quan hệ giữa các node**:  
Các mối quan hệ giữa các node trong đồ thị mẫu cũng cần được kiểm tra. Điều này có thể bao gồm:

* Kiểm tra các cạnh (edges) giữa các node.
* Kiểm tra sự tương thích về mối quan hệ, chẳng hạn như loại quan hệ (ví dụ: "next\_to", "inside", "part\_of").

**Xây dựng các ràng buộc**:  
VF2 sử dụng một hệ thống ràng buộc để đảm bảo rằng các node và mối quan hệ tương thích với nhau. Các ràng buộc này có thể bao gồm:

* **Ràng buộc về tính tương thích**: Node trong đồ thị mẫu và đồ thị tìm kiếm phải có các thuộc tính tương tự.
* **Ràng buộc về kết nối**: Các node trong đồ thị con phải được kết nối giống như trong đồ thị mẫu.

## Tài Liệu Tham Khảo VF2 Algorithm

* **VF2: An Efficient Algorithm for Subgraph Isomorphism** - Đọc tài liệu gốc của thuật toán VF2 để hiểu rõ hơn về các nguyên lý và phương pháp tối ưu hóa mà thuật toán sử dụng.
* **Link tài liệu**: [VF2 Algorithm - Wikipedia](https://en.wikipedia.org/wiki/Subgraph_isomorphism_problem)

## Ứng Dụng Của VF2 Trong COCO

Trong trường hợp **COCO** hoặc bộ dữ liệu ảnh khác, thuật toán VF2 có thể được áp dụng để:

1. **So khớp các đối tượng** trong các ảnh, xác định xem có đối tượng nào giống nhau trong các ảnh khác nhau hay không.
2. **So khớp các hành động** trong các video, xác định xem hành động trong một đoạn video có giống với một hành động mẫu nào không.
3. **Tìm kiếm các mẫu đồ họa tương tự** trong bộ dữ liệu lớn, giúp xác định các cấu trúc hoặc hình ảnh tương tự trong bộ dữ liệu COCO.

# 12. Mở rộng bộ dữ liệu Flick hoặc Visual Genome

* COCO, Flick đều có ảnh + caption → có thể trích xuất quan hệ từ ngôn ngữ (caption-based Knowledge Graph)
* Visual Genome cung cấp sẵn Scene Graph → gần như đã là Knowledge Graph

|  |  |  |  |
| --- | --- | --- | --- |
| **Thành phần** | **COCO** | **Flick** | **Visual Genome** |
| Ảnh | 330K | 30k | 108k |
| Caption | Có | 5 caption/ảnh | Ít caption, chủ yếu mô tả SG |
| Bounding Box | Có | Trích xuất từ mô hình | Có |
| Object/ Relation | Từ annotation | Từ caption (spaCy/VLP) | Có sẵn Object và Relationship |
| Graph | Tự xây dựng (KG) | Trích từ caption | Có sẵn Scene Graph |

1. **Flickr30k Dataset**

**Dữ liệu:**

* 30,000 ảnh
* 5 mô tả (caption) mỗi ảnh

**Mục tiêu:**

* Từ caption → trích xuất quan hệ object
* Dùng spaCy, hoặc các mô hình như **BLIP / VLP** để sinh ra KG
* Có thể dùng YOLO / Detectron2 để trích object

**Các bước:**

1. Load ảnh + caption từ Flickr30k(<https://www.kaggle.com/datasets/hsankesara/flickr-image-dataset>)

2. Với mỗi ảnh:

- Dùng spaCy để phân tích caption → triplets (subject, relation, object)

- Dùng YOLO để nhận diện object (có bbox)

3. Xây dựng KG → lưu vào Neo4j

4. Tìm kiếm ảnh theo triplet hoặc caption

1. **Visual Genome Dataset**

**Dữ liệu:**

* 108,000 ảnh

Có sẵn:

* Object (bbox + name)
* Attribute
* Relationship (Scene Graph)

**Mục tiêu:**

* Load trực tiếp Scene Graph → chuyển thành KG
* Mỗi ảnh sẽ có một KG hoàn chỉnh (nodes + edges)

**Các bước:**

1. Tải Visual Genome từ (<https://visualgenome.org>)

2. Load file `objects.json` và `relationships.json`

3. Với mỗi ảnh:

- Mỗi object → node

- Quan hệ → edge (relation: "holding", "on top of", etc.)

4. Tạo KG bằng NetworkX / Py2Neo

5. Đưa vào Neo4j

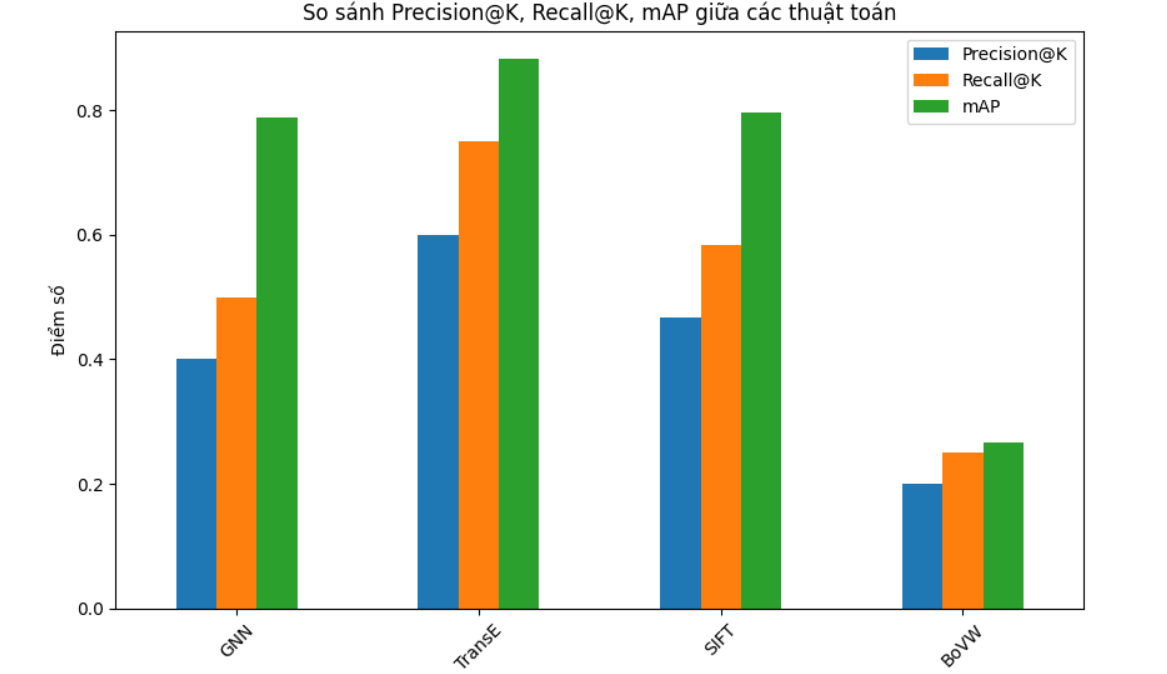
# 13. Đánh giá hiệu suất thuật toán tìm kiếm ảnh

Hiệu suất thuật toán tìm kiếm ảnh có thể đo bằng nhiều cách. Phổ biến nhất là:

* **Precision@K** – Độ chính xác trong K kết quả đầu tiên
* **Recall@K** – Tỉ lệ ảnh liên quan được tìm thấy trong K kết quả
* **Mean Average Precision (mAP)** – Trung bình độ chính xác qua nhiều truy vấn

Cách thực hiện:

1. Thu thập ground-truth (GT) – Tập hợp ảnh liên quan cho từng truy vấn
2. Chạy từng thuật toán – GNN, TransE, SIFT, BoVW
3. So sánh Precision, Recall, mAP



GNN & TransE có hiệu suất cao nhất, đặc biệt về Precision@K & mAP.

SIFT & BoVW kém hơn, do không sử dụng ngữ cảnh từ KG.