**THỊ GIÁC MÁY TÍNH**

**ĐỀ TÀI**

**DETECT WALDO IN PICTURE**

MỤC LỤC

[**CHƯƠNG 1** 2](#_Toc153699133)

[**CƠ SỞ LÝ THUYẾT** 2](#_Toc153699134)

[**1.** **TỔNG QUAN ĐỀ TÀI** 2](#_Toc153699135)

[**2.** **TENSORFLOW OBJECT DECTECTION API** 2](#_Toc153699136)

[**2.1.** **API TENSORFLOW OBJECTIVES DETECTION** 2](#_Toc153699137)

[**2.2.** **FEATURE EXTRACTION** 4](#_Toc153699138)

[**CHƯƠNG 2** 6](#_Toc153699139)

[**KẾT QUẢ THỰC TẾ** 6](#_Toc153699140)

[**1.** **NỘI DUNG CÁC THÀNH PHẦN CHÍNH** 6](#_Toc153699141)

[**1.1.** **TRAIN** 6](#_Toc153699142)

[**1.2.** **FIND\_WALLY VÀ FIND\_WALLY\_PRETTY** 11](#_Toc153699143)

[**1.2.1.** **FIND\_WALLY\_PRETTY** 11](#_Toc153699144)

[**1.2.2.** **FIND\_WALLY** 16](#_Toc153699145)

[**1.3.** **MAIN.PY** 24](#_Toc153699146)

[**CHƯƠNG 3** 31](#_Toc153699147)

[**KẾT LUẬN – ƯU ĐIỂM – HẠN CHẾ** 31](#_Toc153699148)

[**1.** **KẾT LUẬN** 31](#_Toc153699149)

[**2.** **ƯU ĐIỂM** 31](#_Toc153699150)

[**3.** **HẠN CHẾ** 31](#_Toc153699151)

[**CHƯƠNG 4** 31](#_Toc153699152)

[**TÀI LIỆU THAM KHẢO** 32](#_Toc153699153)

# CHƯƠNG 1

# CƠ SỞ LÝ THUYẾT

1. **TỔNG QUAN ĐỀ TÀI**

Detect Waldo in picture là một dự án Tensorflow bao gồm một mô hình được thiết kế để giải câu đố “Where’s Wally”. Dự án này được sử dụng một mô hình Faster R-CNN Inception V2, ban đầu được huấn luyện trên bộ dữ liệu COCO và sau đó được huấn luyện lại để tìm kiếm nhân vật Wally bằng phương pháp học chuyển giao sử dụng TensorFlow Object Dectection API.

Ở đây, chúng ta sẽ đi sâu vào cơ sở lý thuyết về TensorFlow Object Detection API.

1. **TENSORFLOW OBJECT DECTECTION API**
   1. **API TENSORFLOW OBJECTIVES DETECTION**

Là một thuật toán với nhiệm vụ phát hiện tất cả các đối tượng từ một hình ảnh cho trước. Về mặt hình thức, Object Detection là một kỹ thuật thị giác máy tính cho phép chúng ta xác định và định vị các đối tượng trong một hình ảnh hoặc video.

Với cách nhận dạng và bản địa hóa, tính năng này có thể sử dụng để đếm các đối tượng trong một cảnh và xác định, theo dõi vị trí chính xác của chúng trong khi ghi các nhãn chính xác.

Các ứng dụng của thuật toán này được phổ biến rộng rãi. Có thể được áp dụng vào các hệ thống an ninh, phương tiện tự hành, phân tích các hình ảnh y tế. Cùng với việc áp dụng TensorFlow Objective Dectection API sẽ có những công dụng hữu ích.

TensorFlow Object Detection API là một công cụ trong TensorFlow, một thư viện mã nguồn mở của Google dùng cho máy học và thị giác máy tính. API này được xây dựng để giúp đơn giản hóa quá trình triển khai và huấn luyện các mô hình nhận diện đối tượng dựa trên TensorFlow.

Các mô hình tiêu biểu được sử dụng:

Centernet: Object detection khung hơi khác so với các phương pháp Object detection không có mỏ neo truyền thống. Mô hình này dự đoán một bộ ba điểm trên hình ảnh để tìm một hình ảnh. Theo truyền thống, các mô hình sử dụng một cặp điểm chính để phát hiện một đối tượng trong ảnh.

EfficientDet: Thuật toán này sử dụng kiến ​​trúc mạng hiệu quả mới làm xương sống của nó, giúp nó hoạt động mạnh mẽ và nhanh chóng. Các đầu ra của mỗi lớp bên trong effnet được đưa vào mạng đặc tính hai chiều, sau đó sẽ xuất ra các lớp và hộp giới hạn.

SSD: Phát hiện một lần chụp có nghĩa là mỗi đối tượng trong hình ảnh được định vị và đánh dấu chỉ trong một lần chuyển tiếp duy nhất của mạng nơ-ron. Điều này làm giảm đáng kể thời gian suy luận vì chúng ta đang giảm số lượng các phép toán nhân ma trận.

Faster R-CNN: Thuật toán này nhanh hơn rất nhiều so với các thuật toán tiền nhiệm. Điều này sử dụng mạng đề xuất khu vực để tạo dự đoán trái ngược với tìm kiếm chọn lọc trong R-CNN nhanh. Tuy nhiên, một điều quan trọng cần lưu ý là thuật toán này yêu cầu một số lần chuyển hình ảnh qua CNN để đưa ra dự đoán chính xác.

API này có nhiều ưu điểm:

1. Tiện ích cho Nhiệm vụ Nhận diện Đối tượng: API cung cấp một loạt các mô hình đã được đào tạo trước và có thể được sử dụng trực tiếp cho nhận diện đối tượng, giảm đôi khi phức tạp của việc xây dựng mô hình từ đầu.

2. Dễ triển khai: TensorFlow Object Detection API giúp triển khai mô hình nhận diện đối tượng trên nhiều nền tảng, bao gồm cả máy tính để bàn và di động.

3. Hỗ trợ nhiều Mô hình: Cung cấp nhiều loại mô hình nhận diện đối tượng khác nhau, từ các mô hình nhỏ và nhanh đến các mô hình mạnh mẽ và chính xác hơn.

4. Công cụ Đánh giá và Huấn luyện: API đi kèm với các công cụ để đánh giá hiệu suất của mô hình và tiện ích để huấn luyện mô hình trên dữ liệu mới.

5. Hỗ trợ môi trường Cộng tác: TensorFlow Object Detection API thường được sử dụng trong các môi trường nơi nhiều người cùng làm việc trên cùng một dự án và chia sẻ các mô hình và kết quả.

Điều này làm cho TensorFlow Object Detection API trở thành một công cụ hữu ích cho việc triển khai các ứng dụng thị giác máy tính liên quan đến nhận diện đối tượng.

* 1. **FEATURE EXTRACTION**

Feature extraction là quá trình chuyển đổi dữ liệu đầu vào thành một tập hợp các đặc trưng (features) có ý nghĩa cao hơn và thường ít chiều hơn so với dữ liệu gốc. Quá trình này giúp giảm chiều dữ liệu và tạo ra một biểu diễn tinh giản nhưng vẫn bảo toàn thông tin quan trọng.

Đặc trưng (features) có thể là các thuộc tính hay đặc điểm của dữ liệu mà mô hình máy học sử dụng để học và làm các dự đoán. Quá trình feature extraction đặc biệt quan trọng trong việc xử lý dữ liệu đa chiều (high-dimensional data), nơi số lượng chiều của dữ liệu là lớn, và việc sử dụng toàn bộ dữ liệu có thể gặp vấn đề về hiệu suất tính toán và cũng có thể dẫn đến overfitting.

Các phương pháp feature extraction có thể bao gồm việc sử dụng kỹ thuật thống kê, biến đổi tuyến tính như Principal Component Analysis (PCA), hoặc sử dụng các mô hình học sâu để học các đặc trưng ẩn. Mục tiêu của feature extraction là giảm chiều dữ liệu mà vẫn giữ lại thông tin quan trọng, giúp cải thiện hiệu suất và tăng khả năng tổng quát hóa của mô hình máy học.

Ưu điểm:

1. Giảm Chiều Dữ Liệu (Dimensionality Reduction): Giúp giảm số chiều của dữ liệu, giảm độ phức tạp tính toán và yêu cầu bộ nhớ của mô hình.

2. Loại Bỏ Nhiễu (Noise Reduction): Feature extraction có thể giúp loại bỏ những đặc trưng không quan trọng hoặc nhiễu, tập trung vào thông tin quan trọng hơn.

3. Tăng Tốc Quá Trình Huấn Luyện Mô Hình: Dữ liệu giảm chiều thường dẫn đến thời gian huấn luyện mô hình nhanh hơn, đặc biệt là trong trường hợp của các mô hình phức tạp.

4. Khả Năng Tổng Quát Hóa (Generalization): Có thể giúp mô hình tổng quát hóa tốt hơn trên dữ liệu mới, giảm nguy cơ overfitting.

5. Hiểu Quả Cho Dữ Liệu Đa Chiều (High-Dimensional Data): Đối với dữ liệu đa chiều, việc giảm chiều có thể giúp mô hình dễ dàng học được các quy luật tổng quát hóa hơn.

Nhược điểm:

1. Mất Mát Thông Tin:\*Quá trình giảm chiều có thể dẫn đến mất mát một số thông tin, đặc biệt là khi loại bỏ các đặc trưng quan trọng nhưng không quan trọng từ góc độ giảm chiều.

2. Phức Tạp Cho Việc Lựa Chọn Đặc Trưng: Việc chọn lọc hoặc xây dựng đặc trưng mới có thể đòi hỏi kiến thức chuyên sâu và kinh nghiệm trong lĩnh vực cụ thể.

3. Phụ Thuộc vào Dữ Liệu Đầu Vào: Hiệu suất của quá trình feature extraction phụ thuộc nhiều vào tính chất của dữ liệu đầu vào.

4. Khả Năng Overfitting: Nếu không thực hiện đúng cách, quá trình giảm chiều có thể dẫn đến tình trạng underfitting hoặc overfitting.

5. Chi Phí Tính Toán: Trong một số trường hợp, quá trình giảm chiều có thể đòi hỏi nhiều tài nguyên tính toán, đặc biệt là khi sử dụng các phương pháp phức tạp như deep learning.

# CHƯƠNG 2

# KẾT QUẢ THỰC TẾ

1. NỘI DUNG CÁC THÀNH PHẦN CHÍNH
   1. TRAIN

Ảnh có chứa văn bản, ảnh chụp màn hình

Mô tả được tạo tự động

A screenshot of a computer program

Description automatically generated

**Hàm get\_configs\_from\_pipeline\_file():**

- Đọc cấu hình huấn luyện từ một file duy nhất được chỉ định bởi cờ pipeline\_config\_path.

- Tạo một đối tượng pipeline\_pb2.TrainEvalPipelineConfig để lưu trữ cấu hình được đọc.

- Sử dụng tf.gfile.GFile() để mở file và đọc nội dung vào biến pipeline\_config.

- Đọc các thông tin cấu hình từ pipeline\_config và gán cho model\_config, train\_config và input\_config. Các thông tin này được lấy từ các trường tương ứng trong pipeline\_config.

- Trả về model\_config, train\_config và input\_config.

**Hàm get\_configs\_from\_multiple\_files():**

- Đọc cấu hình huấn luyện từ nhiều file cấu hình khác nhau.

- Sử dụng tf.gfile.GFile() để mở từng file cấu hình và đọc nội dung vào các đối tượng tương ứng: model\_config, train\_config và input\_config.

- Các file cấu hình được chỉ định bởi các cờ model\_config\_path, train\_config\_path và input\_config\_path.

- Trả về model\_config, train\_config và input\_config.

A computer screen shot of a program code

Description automatically generated

* assert FLAGS.train\_dir, 'train\_dir is missing.': Đoạn mã này kiểm tra xem biến FLAGS.train\_dir đã được đặt giá trị hay chưa. Nếu không, nó sẽ kích hoạt một lỗi với thông báo 'train\_dir is missing.'.
* if FLAGS.pipeline\_config\_path: ... else: ...: Đoạn mã này kiểm tra xem cờ FLAGS.pipeline\_config\_path đã được đặt giá trị hay chưa. Nếu có, nó sẽ gọi hàm get\_configs\_from\_pipeline\_file() để lấy các cấu hình từ một tệp cấu hình duy nhất. Nếu không, nó sẽ gọi hàm get\_configs\_from\_multiple\_files() để lấy các cấu hình từ nhiều tệp cấu hình riêng biệt.
* model\_fn = functools.partial(model\_builder.build, model\_config=model\_config, is\_training=True): Đoạn mã này tạo một hàm mô hình (model\_fn) bằng cách sử dụng hàm model\_builder.build và truyền các đối số model\_config và is\_training=True thông qua functools.partial.
* create\_input\_dict\_fn = functools.partial(input\_reader\_builder.build, input\_config): Đoạn mã này tạo một hàm (create\_input\_dict\_fn) để tạo một từ điển đầu vào (input\_dict) bằng cách sử dụng hàm input\_reader\_builder.build và truyền đối số input\_config thông qua functools.partial.
* env = json.loads(os.environ.get('TF\_CONFIG', '{}')): Đoạn mã này trích xuất thông tin môi trường TensorFlow từ biến môi trường TF\_CONFIG và lưu trữ vào biến env.
* cluster\_data = env.get('cluster', None): Đoạn mã này lấy thông tin về cụm (cluster) từ biến env. Nếu không có thông tin về cụm, giá trị của cluster\_data sẽ là None.
* cluster = tf.train.ClusterSpec(cluster\_data) if cluster\_data else None: Đoạn mã này tạo một đối tượng ClusterSpec từ thông tin về cụm (cluster\_data). Nếu không có thông tin về cụm, cluster sẽ là None.
* task\_data = env.get('task', None) or {'type': 'master', 'index': 0}: Đoạn mã này lấy thông tin về tác vụ (task) từ biến env. Nếu không có thông tin về tác vụ, giá trị của task\_data sẽ là {'type': 'master', 'index': 0}.
* task\_info = type('TaskSpec', (object,), task\_data): Đoạn mã này tạo một đối tượng TaskSpec từ thông tin về tác vụ (task\_data). Đối tượng TaskSpec có thuộc tính và giá trị giống như thông tin tác vụ.
* ps\_tasks = 0: Đoạn mã này đặt số tác vụ ps\_tasks là 0.
* worker\_replicas = 1: Đoạn mã này đặt số lượng bản sao của công nhân (worker\_replicas) là 1.
* worker\_job\_name = 'lonely\_worker': Đoạn mã này đặt tên công việc của công nhân (worker\_job\_name) là 'lonely\_worker'.
* task = 0: Đoạn mã này đặt chỉ số của tác vụ (task) là 0.
* is\_chief = True: Đoạn mã này đặt giá trị is\_chief là True.
* master = '': Đoạn mã này đặt giá trị master là một chuỗi rỗng.

A computer screen shot of a program code

Description automatically generated

* if cluster\_data and 'worker' in cluster\_data: ...: Đoạn mã này kiểm tra xem có thông tin về cụm và tác vụ công nhân trong cluster\_data hay không. Nếu có, nó sẽ thực hiện các bước cấu hình cho huấn luyện phân tán.
* worker\_replicas = len(cluster\_data['worker']) + 1: Đoạn mã này đặt số lượng bản sao của công nhân (worker\_replicas) bằng tổng số tác vụ công nhân (len(cluster\_data['worker'])) cộng thêm 1 để bao gồm công việc của "master".
* if cluster\_data and 'ps' in cluster\_data: ...: Đoạn mã này kiểm tra xem có thông tin về tác vụ PS (parameter server) trong cluster\_data hay không. Nếu có, nó sẽ thực hiện các bước cấu hình cho tác vụ PS.
* ps\_tasks = len(cluster\_data['ps']): Đoạn mã này đặt số lượng tác vụ PS (ps\_tasks) bằng số tác vụ PS có trong cluster\_data.
* if worker\_replicas > 1 and ps\_tasks < 1: ...: Đoạn mã này kiểm tra xem có ít nhất 1 tác vụ PS cần thiết cho huấn luyện phân tán hay không. Nếu số lượng bản sao của công nhân (worker\_replicas) lớn hơn 1 và không có tác vụ PS (ps\_tasks) nào, nó sẽ kích hoạt lỗi với thông báo 'At least 1 ps task is needed for distributed training.'
* if worker\_replicas >= 1 and ps\_tasks > 0: ...: Đoạn mã này kiểm tra xem có ít nhất 1 bản sao của công nhân và ít nhất 1 tác vụ PS hay không. Nếu có, nó sẽ tiến hành cấu hình huấn luyện phân tán.
* server = tf.train.Server(tf.train.ClusterSpec(cluster), protocol='grpc', job\_name=task\_info.type, task\_index=task\_info.index): Đoạn mã này tạo một đối tượng máy chủ (Server) TensorFlow phân tán với thông tin về cụm (tf.train.ClusterSpec(cluster)) và các thông tin về công việc và chỉ số tác vụ (job\_name=task\_info.type, task\_index=task\_info.index). Giao thức giao tiếp được sử dụng là 'grpc'.
* if task\_info.type == 'ps': ...: Đoạn mã này kiểm tra xem loại công việc là tác vụ PS hay không. Nếu là tác vụ PS, nó sẽ chờ các tác vụ khác và kết thúc.
* worker\_job\_name = '%s/task:%d' % (task\_info.type, task\_info.index): Đoạn mã này tạo tên công việc của công nhân (worker\_job\_name) bằng cách kết hợp loại công việc và chỉ số tác vụ.
* task = task\_info.index: Đoạn mã này đặt giá trị task bằng chỉ số tác vụ.
* is\_chief = (task\_info.type == 'master'): Đoạn mã này đặt giá trị is\_chief là True nếu loại công việc là "master", ngược lại is\_chief sẽ là False.
* master = server.target: Đoạn mã này đặt giá trị master bằng địa chỉ mục tiêu củamáy chủ (server).
* trainer.train(create\_input\_dict\_fn, model\_fn, train\_config, master, task, FLAGS.num\_clones, worker\_replicas, FLAGS.clone\_on\_cpu, ps\_tasks, worker\_job\_name, is\_chief, FLAGS.train\_dir): Đoạn mã này gọi hàm trainer.train() để bắt đầu quá trình huấn luyện. Các đối số được truyền vào bao gồm các hàm và thông số cần thiết như create\_input\_dict\_fn, model\_fn, train\_config, master, task, FLAGS.num\_clones, worker\_replicas, FLAGS.clone\_on\_cpu, ps\_tasks, worker\_job\_name, is\_chief, và FLAGS.train\_dir.
  1. **FIND\_WALLY VÀ FIND\_WALLY\_PRETTY**
     1. **FIND\_WALLY\_PRETTY**

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Hệ điều hành

Mô tả được tạo tự động

* draw\_box(box, image\_np): Đây là hàm chính để vẽ hình chữ nhật trên hình ảnh. Hàm này nhận đầu vào là box (mảng numpy biểu diễn tọa độ của hình chữ nhật) và image\_np (mảng numpy biểu diễn hình ảnh). Hàm trả về đối tượng fig và ax để có thể hiển thị hình vẽ.
* tf.Graph(): Đây là hàm trong TensorFlow để tạo một đồ thị tính toán mới.
* with detection\_graph.as\_default():: Đoạn mã này đặt đồ thị detection\_graph làm đồ thị tính toán mặc định trong phạm vi của khối with. Điều này đảm bảo rằng các hoạt động tính toán trong khối with sẽ được thực hiện trên đồ thị detection\_graph.
* od\_graph\_def = tf.compat.v1.GraphDef(): Đoạn mã này tạo một đối tượng od\_graph\_def để lưu trữ định nghĩa của đồ thị TensorFlow.
* with tf.io.gfile.GFile(model\_path, 'rb') as fid:: Đoạn mã này mở một tệp mô hình (đường dẫn được đặt trong biến model\_path) với chế độ đọc nhị phân ('rb') và đặt đối tượng tệp vào biến fid.
* serialized\_graph = fid.read(): Đoạn mã này đọc toàn bộ nội dung của tệp mô hình và lưu trữ trong biến serialized\_graph.
* od\_graph\_def.ParseFromString(serialized\_graph): Đoạn mã này phân tích nội dung đã đọc từ tệp mô hình và cập nhật đối tượng od\_graph\_def với định nghĩa của đồ thị TensorFlow.
* tf.import\_graph\_def(od\_graph\_def, name=''): Đoạn mã này import định nghĩa của đồ thị od\_graph\_def vào đồ thị tính toán TensorFlow hiện tại. Các hoạt động và biến được định nghĩa trong đồ thị od\_graph\_def sẽ trở thành phần của đồ thị tính toán hiện tại.

Đoạn mã trên định nghĩa một hàm load\_image\_into\_numpy\_array để chuyển đổi một đối tượng hình ảnh (image) thành một mảng numpy.

* (im\_width, im\_height) = image.size: Dòng này lấy kích thước của hình ảnh (image) bằng cách sử dụng phương thức size. Kích thước của hình ảnh được trả về dưới dạng một bộ có hai phần tử (width, height).
* return np.array(image.getdata()).reshape((im\_height, im\_width, 3)).astype(np.uint8): Đoạn mã này thực hiện các bước chính để chuyển đổi hình ảnh thành một mảng numpy.
* image.getdata() trả về một danh sách chứa các giá trị pixel của hình ảnh. Mỗi giá trị pixel là một bộ RGB (màu đỏ, màu xanh lá cây, màu xanh lam).
* np.array(image.getdata()) tạo một mảng numpy từ danh sách giá trị pixel. Kết quả là một mảng 1D chứa tất cả giá trị pixel của hình ảnh.
* .reshape((im\_height, im\_width, 3)) thay đổi hình dạng của mảng thành một mảng 3D có kích thước (im\_height, im\_width, 3). Điều này đảm bảo rằng mảng có cấu trúc giống với hình ảnh ban đầu, trong đó trục thứ ba có kích thước 3 đại diện cho các kênh màu RGB.
* .astype(np.uint8) chuyển đổi kiểu dữ liệu của mảng numpy thành np.uint8, tức là kiểu dữ liệu không dấu 8 bit. Điều này đảm bảo rằng các giá trị pixel trong mảng nằm trong phạm vi từ 0 đến 255, tương ứng với các giá trị màu RGB.
* return trả về mảng numpy đã chuyển đổi từ hình ảnh ban đầu.

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phông chữ

Mô tả được tạo tự động

* with detection\_graph.as\_default():: Đoạn mã này đặt đồ thị detection\_graph làm đồ thị tính toán mặc định trong phạm vi của khối with. Điều này đảm bảo rằng các hoạt động tính toán trong khối with sẽ được thực hiện trên đồ thị detection\_graph
* with tf.compat.v1.Session(graph=detection\_graph) as sess:: Đoạn mã này tạo một phiên TensorFlow (sess) với đồ thị detection\_graph làm đồ thị tính toán và đặt phiên làm phiên mặc định trong phạm vi của khối with. Các hoạt động tính toán được thực hiện trong khối with sẽ sử dụng phiên này.
* parser = argparse.ArgumentParser(): Đoạn mã này tạo một đối tượng ArgumentParser để xử lý các đối số dòng lệnh.
* parser.add\_argument('image\_path'): Đoạn mã này định nghĩa một đối số dòng lệnh có tên image\_path để truyền đường dẫn đến hình ảnh cần phân loại.
* args = parser.parse\_args(): Đoạn mã này phân tích các đối số dòng lệnh và lưu trữ kết quả vào biến args.
* image\_np = load\_image\_into\_numpy\_array(Image.open(args.image\_path)): Đoạn mã này mở hình ảnh từ đường dẫn được chỉ định trong args.image\_path, sau đó chuyển đổi hình ảnh thành mảng numpy bằng cách sử dụng hàm load\_image\_into\_numpy\_array.
* image\_tensor = detection\_graph.get\_tensor\_by\_name('image\_tensor:0'): Đoạn mã này lấy tensor đầu vào (image\_tensor) từ đồ thị detection\_graph bằng cách sử dụng phương thức get\_tensor\_by\_name. 'image\_tensor:0' là tên của tensor đầu vào.
* boxes = detection\_graph.get\_tensor\_by\_name('detection\_boxes:0'): Đoạn mã này lấy tensor boxes từ đồ thị detection\_graph bằng cách sử dụng phương thức get\_tensor\_by\_name. 'detection\_boxes:0' là tên của tensor boxes.
* scores = detection\_graph.get\_tensor\_by\_name('detection\_scores:0'): Đoạn mã này lấy tensor scores từ đồ thị detection\_graph bằng cách sử dụng phương thức get\_tensor\_by\_name. 'detection\_scores:0' là tên của tensor scores.
* classes = detection\_graph.get\_tensor\_by\_name('detection\_classes:0'): Đoạn mã này lấy tensor classes từ đồ thị detection\_graph bằng cách sử dụng phương thức get\_tensor\_by\_name. 'detection\_classes:0' là tên của tensor classes.
* num\_detections = detection\_graph.get\_tensor\_by\_name('num\_detections:0'): Đoạn mã này lấy tensor num\_detections từ đồ thị detection\_graph bằng cách sử dụng phương thức get\_tensor\_by\_name. 'num\_detections:0' là tên của tensor num\_detections.
* (boxes, scores, classes, num\_detections) = sess.run([boxes, scores, classes, num\_detections], feed\_dict={image\_tensor: np.expand\_dims(image\_np, axis=0)}): Đoạn mã này thực hiện phân loại hình ảnh bằng cách chạy các tensor boxes, scores, classes, num\_detections trong phiên sess. Hình ảnh được cung cấp cho tensor đầu vào image\_tensor thông qua feed\_dict. Kết quả của phân loại được lưu trữ trong các biến boxes, scores, classes, num\_detections.
* if scores[0][0< 0.1:
* sys.exit('Wally not found :(')
* Nếu điểm số (scores) của vị trí đầu tiên ([0][0]) nhỏ hơn 0.1, tức là kết quả phân loại cho thấy không tìm thấy đối tượng cần tìm (Wally), thì chương trình sẽ kết thúc và hiển thị thông báo "Wally not found :(".
* print('Wally found'): Nếu điểm số của vị trí đầu tiên lớn hơn hoặc bằng 0.1, chương trình sẽ in ra thông báo "Wally found".
* fig, ax = draw\_box(boxes[0][0], image\_np): Đoạn mã này gọi hàm draw\_box để vẽ một hình chữ nhật (box) xung quanh đối tượng tìm thấy. Kết quả là fig (đối tượng hình vẽ) và ax (trục của hình vẽ).
* ax.imshow(image\_np): Đoạn mã này sử dụng trục ax để hiển thị hình ảnh (image\_np) đã phân loại lên hình vẽ.
* plt.show(): Đoạn mã này hiển thị hình vẽ với hình ảnh đã phân loại và hình chữ nhật vẽ xung quanh đối tượng tìm thấy.
  + 1. FIND\_WALLY

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm

Mô tả được tạo tự động

* detection\_graph = tf.Graph(): Đoạn mã này tạo một đối tượng đồ thị TensorFlow mới có tên detection\_graph.
  + with detection\_graph.as\_default():: Đoạn mã này đặt đồ thị detection\_graph làm đồ thị tính toán mặc định trong phạm vi của khối with. Điều này đảm bảo rằng các hoạt động tính toán trong khối with sẽ được thực hiện trên đồ thị detection\_graph.
  + od\_graph\_def = tf.compat.v1.GraphDef(): Đoạn mã này tạo một đối tượng GraphDef để lưu trữ định nghĩa của đồ thị.
  + with tf.io.gfile.GFile(model\_path, 'rb') as fid: ...: Đoạn mã này mở tệp model\_path dưới dạng đọc nhị phân ('rb') và đọc nội dung của tệp vào biến serialized\_graph.
  + od\_graph\_def.ParseFromString(serialized\_graph): Đoạn mã này phân tích nội dung của serialized\_graph và lưu trữ nó trong od\_graph\_def.
  + tf.import\_graph\_def(od\_graph\_def, name=''): Đoạn mã này nhập định nghĩa đồ thị (od\_graph\_def) vào đồ thị TensorFlow (detection\_graph). Tham số name='' chỉ định rằng các tên trong định nghĩa đồ thị sẽ được thêm vào không gian tên mặc định.
  + def load\_image\_into\_numpy\_array(image): ...: Đoạn mã này định nghĩa một hàm load\_image\_into\_numpy\_array để chuyển đổi hình ảnh thành mảng numpy.
  + (im\_width, im\_height) = image.size: Đoạn mã này lấy kích thước chiều rộng (im\_width) và chiều cao (im\_height) của hình ảnh.
  + return np.array(image.getdata()).reshape((im\_height, im\_width, 3)).astype(np.uint8): Đoạn mã này chuyển đổi dữ liệu pixel của hình ảnh thành mảng numpy. Hàm image.getdata() trả về một danh sách các giá trị pixel, và np.array() được sử dụng để chuyển đổi danh sách này thành mảng numpy. Sau đó, hàm reshape() được sử dụng để thay đổi hình dạng của mảng thành (chiều cao, chiều rộng, 3) để phù hợp với định dạng hình ảnh RGB. Cuối cùng, astype(np.uint8) được sử dụng để chuyển đổi các giá trị pixel thành kiểu dữ liệu unsigned 8-bit (uint8).
  + label\_map = label\_map\_util.load\_labelmap('./trained\_model/labels.txt'): Đoạn mã này tải danh sách nhãn từ tệp labels.txt sử dụng hàm load\_labelmap trong label\_map\_util. Đường dẫn ./trained\_model/labels.txt chỉ định vị trí của tệp danh sách nhãn.
  + categories = label\_map\_util.convert\_label\_map\_to\_categories(label\_map, max\_num\_classes=1, use\_display\_name=True): Đoạn mã này chuyển đổi danh sách nhãn thành danh sách các đối tượng categories sử dụng hàm convert\_label\_map\_to\_categories trong label\_map\_util. Tham số max\_num\_classes chỉ định số lượng lớn nhất các lớp được chuyển đổi và use\_display\_name=True chỉ định sử dụng tên hiển thị của các lớp.
  + category\_index = label\_map\_util.create\_category\_index(categories): Đoạn mã này tạo một bản đồ từ danh sách categories sử dụng hàm create\_category\_index trong label\_map\_util. Bản đồ này gántiếp tục:
  + category\_index = label\_map\_util.create\_category\_index(categories): Đoạn mã này tạo một bản đồ từ danh sách categories sử dụng hàm create\_category\_index trong label\_map\_util. Bản đồ này gán một số nguyên duy nhất cho mỗi nhãn trong danh sách và cung cấp thông tin về tên hiển thị tương ứng với mỗi nhãn.
  + with detection\_graph.as\_default(): with tf.compat.v1.Session(graph=detection\_graph) as sess: ...: Đoạn mã này tạo một phiên TensorFlow (sess) sử dụng đồ thị detection\_graph. Điều này cho phép chúng ta thực hiện các hoạt động tính toán trên mô hình.
  + parser = argparse.ArgumentParser(): Đoạn mã này tạo một đối tượng ArgumentParser để xử lý các đối số dòng lệnh.
  + parser.add\_argument('image\_path'): Đoạn mã này định nghĩa một đối số dòng lệnh image\_path để chứa đường dẫn đến hình ảnh đầu vào.
  + args = parser.parse\_args(): Đoạn mã này phân tích các đối số dòng lệnh và lưu trữ chúng vào biến args.
  + image\_np = load\_image\_into\_numpy\_array(Image.open(args.image\_path)): Đoạn mã này mở hình ảnh từ đường dẫn args.image\_path bằng Image.open, sau đó chuyển đổi hình ảnh thành mảng numpy bằng cách gọi hàm load\_image\_into\_numpy\_array.
  + image\_tensor = detection\_graph.get\_tensor\_by\_name('image\_tensor:0'): Đoạn mã này lấy tham số đầu vào của mô hình (tensor hình ảnh) từ đồ thị detection\_graph bằng cách sử dụng get\_tensor\_by\_name. 'image\_tensor:0' là tên của tensor hình ảnh trong đồ thị.
  + boxes = detection\_graph.get\_tensor\_by\_name('detection\_boxes:0'): Đoạn mã này lấy tensor detection\_boxes từ đồ thị detection\_graph. 'detection\_boxes:0' là tên của tensor boxes trong đồ thị.
  + scores = detection\_graph.get\_tensor\_by\_name('detection\_scores:0'): Đoạn mã này lấy tensor detection\_scores từ đồ thị detection\_graph. 'detection\_scores:0' là tên của tensor scores trong đồ thị.
  + classes = detection\_graph.get\_tensor\_by\_name('detection\_classes:0'): Đoạn mã này lấy tensor detection\_classes từ đồ thị detection\_graph. 'detection\_classes:0' là tên của tensor classes trong đồ thị.
  + num\_detections = detection\_graph.get\_tensor\_by\_name('num\_detections:0'): Đoạn mã này lấy tensor num\_detections từ đồ thị detection\_graph. 'num\_detections:0' là tên của tensor num\_detections trong đồ thị.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

* (boxes, scores, classes, num\_detections) = sess.run([boxes, scores, classes, num\_detections], feed\_dict={image\_tensor: np.expand\_dims(image\_np, axis=0)}): Đoạn mã này chạy phiên TensorFlow (sess) để lấy kết quả phát hiện đối tượng từ hình ảnh. Các biến boxes, scores, classes và num\_detections sẽ chứa các giá trị tương ứng của hộp giới hạn, điểm số, lớp và số lượng phát hiện.
* if scores[0][0] < 0.1: sys.exit('Wally not found :('): Đoạn mã này kiểm tra xem điểm số của phát hiện đầu tiên có nhỏ hơn 0.1 hay không. Nếu điều này đúng, nghĩa là không tìm thấy đối tượng được gọi là "Wally" (có thể là một tên khác trong bài toán thực tế), chương trình sẽ kết thúc với thông báo 'Wally not found :('.
* print('Wally found'): Đoạn mã này in ra thông báo 'Wally found' nếu điểm số của phát hiện đầu tiên lớn hơn hoặc bằng 0.1, cho biết rằng đối tượng "Wally" đã được tìm thấy.
* vis\_util.visualize\_boxes\_and\_labels\_on\_image\_array(image\_np, np.squeeze(boxes), np.squeeze(classes).astype(np.int32), np.squeeze(scores), category\_index, use\_normalized\_coordinates=True, line\_thickness=8): Đoạn mã này gọi hàm visualize\_boxes\_and\_labels\_on\_image\_array trong vis\_util để vẽ các hộp giới hạn và nhãn lên hình ảnh image\_np. Các tham số đầu vào bao gồm image\_np (hình ảnh gốc), boxes (các hộp giới hạn), classes (các lớp), scores (điểm số), category\_index (bản đồ các nhãn và chỉ số) và các tham số khác để định dạng và độ dày của các đường vẽ.
* plt.figure(figsize=(12, 8)): Đoạn mã này tạo một hình vẽ mới với kích thước 12x8.
* plt.imshow(image\_np): Đoạn mã này hiển thị hình ảnh image\_np trong hình vẽ.
* plt.show(): Đoạn mã này hiển thị hình vẽ và hình ảnh đã được vẽ lên.

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm

Mô tả được tạo tự động

* detection\_graph = tf.Graph(): Đoạn mã này tạo một đối tượng đồ thị TensorFlow mới có tên detection\_graph.
  + with detection\_graph.as\_default():: Đoạn mã này đặt đồ thị detection\_graph làm đồ thị tính toán mặc định trong phạm vi của khối with. Điều này đảm bảo rằng các hoạt động tính toán trong khối with sẽ được thực hiện trên đồ thị detection\_graph.
  + od\_graph\_def = tf.compat.v1.GraphDef(): Đoạn mã này tạo một đối tượng GraphDef để lưu trữ định nghĩa của đồ thị.
  + with tf.io.gfile.GFile(model\_path, 'rb') as fid: ...: Đoạn mã này mở tệp model\_path dưới dạng đọc nhị phân ('rb') và đọc nội dung của tệp vào biến serialized\_graph.
  + od\_graph\_def.ParseFromString(serialized\_graph): Đoạn mã này phân tích nội dung của serialized\_graph và lưu trữ nó trong od\_graph\_def.
  + tf.import\_graph\_def(od\_graph\_def, name=''): Đoạn mã này nhập định nghĩa đồ thị (od\_graph\_def) vào đồ thị TensorFlow (detection\_graph). Tham số name='' chỉ định rằng các tên trong định nghĩa đồ thị sẽ được thêm vào không gian tên mặc định.
  + def load\_image\_into\_numpy\_array(image): ...: Đoạn mã này định nghĩa một hàm load\_image\_into\_numpy\_array để chuyển đổi hình ảnh thành mảng numpy.
  + (im\_width, im\_height) = image.size: Đoạn mã này lấy kích thước chiều rộng (im\_width) và chiều cao (im\_height) của hình ảnh.
  + return np.array(image.getdata()).reshape((im\_height, im\_width, 3)).astype(np.uint8): Đoạn mã này chuyển đổi dữ liệu pixel của hình ảnh thành mảng numpy. Hàm image.getdata() trả về một danh sách các giá trị pixel, và np.array() được sử dụng để chuyển đổi danh sách này thành mảng numpy. Sau đó, hàm reshape() được sử dụng để thay đổi hình dạng của mảng thành (chiều cao, chiều rộng, 3) để phù hợp với định dạng hình ảnh RGB. Cuối cùng, astype(np.uint8) được sử dụng để chuyển đổi các giá trị pixel thành kiểu dữ liệu unsigned 8-bit (uint8).
  + label\_map = label\_map\_util.load\_labelmap('./trained\_model/labels.txt'): Đoạn mã này tải danh sách nhãn từ tệp labels.txt sử dụng hàm load\_labelmap trong label\_map\_util. Đường dẫn ./trained\_model/labels.txt chỉ định vị trí của tệp danh sách nhãn.
  + categories = label\_map\_util.convert\_label\_map\_to\_categories(label\_map, max\_num\_classes=1, use\_display\_name=True): Đoạn mã này chuyển đổi danh sách nhãn thành danh sách các đối tượng categories sử dụng hàm convert\_label\_map\_to\_categories trong label\_map\_util. Tham số max\_num\_classes chỉ định số lượng lớn nhất các lớp được chuyển đổi và use\_display\_name=True chỉ định sử dụng tên hiển thị của các lớp.
  + category\_index = label\_map\_util.create\_category\_index(categories): Đoạn mã này tạo một bản đồ từ danh sách categories sử dụng hàm create\_category\_index trong label\_map\_util. Bản đồ này gán tiếp tục:
  + category\_index = label\_map\_util.create\_category\_index(categories): Đoạn mã này tạo một bản đồ từ danh sách categories sử dụng hàm create\_category\_index trong label\_map\_util. Bản đồ này gán một số nguyên duy nhất cho mỗi nhãn trong danh sách và cung cấp thông tin về tên hiển thị tương ứng với mỗi nhãn.
  + with detection\_graph.as\_default(): with tf.compat.v1.Session(graph=detection\_graph) as sess: ...: Đoạn mã này tạo một phiên TensorFlow (sess) sử dụng đồ thị detection\_graph. Điều này cho phép chúng ta thực hiện các hoạt động tính toán trên mô hình.
  + parser = argparse.ArgumentParser(): Đoạn mã này tạo một đối tượng ArgumentParser để xử lý các đối số dòng lệnh.
  + parser.add\_argument('image\_path'): Đoạn mã này định nghĩa một đối số dòng lệnh image\_path để chứa đường dẫn đến hình ảnh đầu vào.
  + args = parser.parse\_args(): Đoạn mã này phân tích các đối số dòng lệnh và lưu trữ chúng vào biến args.
  + image\_np = load\_image\_into\_numpy\_array(Image.open(args.image\_path)): Đoạn mã này mở hình ảnh từ đường dẫn args.image\_path bằng Image.open, sau đó chuyển đổi hình ảnh thành mảng numpy bằng cách gọi hàm load\_image\_into\_numpy\_array.
  + image\_tensor = detection\_graph.get\_tensor\_by\_name('image\_tensor:0'): Đoạn mã này lấy tham số đầu vào của mô hình (tensor hình ảnh) từ đồ thị detection\_graph bằng cách sử dụng get\_tensor\_by\_name. 'image\_tensor:0' là tên của tensor hình ảnh trong đồ thị.
  + boxes = detection\_graph.get\_tensor\_by\_name('detection\_boxes:0'): Đoạn mã này lấy tensor detection\_boxes từ đồ thị detection\_graph. 'detection\_boxes:0' là tên của tensor boxes trong đồ thị.
  + scores = detection\_graph.get\_tensor\_by\_name('detection\_scores:0'): Đoạn mã này lấy tensor detection\_scores từ đồ thị detection\_graph. 'detection\_scores:0' là tên của tensor scores trong đồ thị.
  + classes = detection\_graph.get\_tensor\_by\_name('detection\_classes:0'): Đoạn mã này lấy tensor detection\_classes từ đồ thị detection\_graph. 'detection\_classes:0' là tên của tensor classes trong đồ thị.
  + num\_detections = detection\_graph.get\_tensor\_by\_name('num\_detections:0'): Đoạn mã này lấy tensor num\_detections từ đồ thị detection\_graph. 'num\_detections:0' là tên của tensor num\_detections trong đồ thị.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

* (boxes, scores, classes, num\_detections) = sess.run([boxes, scores, classes, num\_detections], feed\_dict={image\_tensor: np.expand\_dims(image\_np, axis=0)}): Đoạn mã này chạy phiên TensorFlow (sess) để lấy kết quả phát hiện đối tượng từ hình ảnh. Các biến boxes, scores, classes và num\_detections sẽ chứa các giá trị tương ứng của hộp giới hạn, điểm số, lớp và số lượng phát hiện.
* if scores[0][0] < 0.1: sys.exit('Wally not found :('): Đoạn mã này kiểm tra xem điểm số của phát hiện đầu tiên có nhỏ hơn 0.1 hay không. Nếu điều này đúng, nghĩa là không tìm thấy đối tượng được gọi là "Wally" (có thể là một tên khác trong bài toán thực tế), chương trình sẽ kết thúc với thông báo 'Wally not found :('.
* print('Wally found'): Đoạn mã này in ra thông báo 'Wally found' nếu điểm số của phát hiện đầu tiên lớn hơn hoặc bằng 0.1, cho biết rằng đối tượng "Wally" đã được tìm thấy.
* vis\_util.visualize\_boxes\_and\_labels\_on\_image\_array(image\_np, np.squeeze(boxes), np.squeeze(classes).astype(np.int32), np.squeeze(scores), category\_index, use\_normalized\_coordinates=True, line\_thickness=8): Đoạn mã này gọi hàm visualize\_boxes\_and\_labels\_on\_image\_array trong vis\_util để vẽ các hộp giới hạn và nhãn lên hình ảnh image\_np. Các tham số đầu vào bao gồm image\_np (hình ảnh gốc), boxes (các hộp giới hạn), classes (các lớp), scores (điểm số), category\_index (bản đồ các nhãn và chỉ số) và các tham số khác để định dạng và độ dày của các đường vẽ.
* plt.figure(figsize=(12, 8)): Đoạn mã này tạo một hình vẽ mới với kích thước 12x8.
* plt.imshow(image\_np): Đoạn mã này hiển thị hình ảnh image\_np trong hình vẽ.
* plt.show(): Đoạn mã này hiển thị hình vẽ và hình ảnh đã được vẽ lên.
  1. MAIN.PY

A computer screen shot of text

Description automatically generated

A computer screen shot of text

Description automatically generated

Hàm run\_wally\_finder nhận đầu vào là đường dẫn của một hình ảnh (image\_path). Nhiệm vụ của hàm này là sử dụng mô hình phát hiện đối tượng để tìm kiếm và hiển thị vùng chứa đối tượng "Wally" trên hình ảnh.

* image\_np = np.array(Image.open(image\_path)): Đoạn mã này đọc hình ảnh từ đường dẫn image\_path bằng thư viện PIL và chuyển đổi nó thành một mảng numpy (image\_np).
* with detection\_graph.as\_default():: Đoạn mã này đặt detection\_graph (đồ thị phát hiện đối tượng) làm đồ thị mặc định.
* with tf.compat.v1.Session(graph=detection\_graph) as sess:: Đoạn mã này tạo một phiên TensorFlow (sess) với detection\_graph làm đồ thị của phiên.
* image\_tensor = detection\_graph.get\_tensor\_by\_name('image\_tensor:0'): Đoạn mã này lấy tensor đầu vào của mô hình từ detection\_graph. Tensor này được đặt tên là 'image\_tensor:0'.
* boxes = detection\_graph.get\_tensor\_by\_name('detection\_boxes:0'): Đoạn mã này lấy tensor chứa thông tin về hộp giới hạn từ detection\_graph. Tensor này được đặt tên là 'detection\_boxes:0'.
* scores = detection\_graph.get\_tensor\_by\_name('detection\_scores:0'): Đoạn mã này lấy tensor chứa thông tin về điểm số từ detection\_graph. Tensor này được đặt tên là 'detection\_scores:0'.
* classes = detection\_graph.get\_tensor\_by\_name('detection\_classes:0'): Đoạn mã này lấy tensor chứa thông tin về lớp của đối tượng từ detection\_graph. Tensor này được đặt tên là 'detection\_classes:0'.
* num\_detections = detection\_graph.get\_tensor\_by\_name('num\_detections:0'): Đoạn mã này lấy tensor chứa thông tin về số lượng phát hiện từ detection\_graph. Tensor này được đặt tên là 'num\_detections:0'.
* (boxes, scores, classes, num\_detections) = sess.run([boxes, scores, classes, num\_detections], feed\_dict={image\_tensor: np.expand\_dims(image\_np, axis=0)}): Đoạn mã này chạy phiên TensorFlow (sess) để lấy kết quả phát hiện đối tượng từ hình ảnh. Các biến boxes, scores, classes và num\_detections sẽ chứa các giá trị tương ứng của hộp giới hạn, điểm số, lớp và số lượng phát hiện.
* if scores[0][0] < 0.1: sg.popup\_error('Wally not found :(', title='Wally Finder') return: Đoạn mã này kiểm tra xem điểm số của phát hiện đầu tiên có nhỏ hơn 0.1 hay không. Nếu điều này đúng, nghĩa là không tìm thấy đối tượng "Wally" (có thể là một tên khác trong bài toán thực tế), sẽ hiển thị một thông báo lỗi và kết thúc hàm.
* print('Wally found'): Đoạn mã này in ra thông báo 'Wally found' nếu điểm số của phát hiện đầu tiên lớn hơn hoặc bằng 0.1, cho biết rằng đối tượng "Wally" đã được tìm thấy.
* fig, ax = draw\_box(boxes[0][0], image\_np): Đoạn mã này gọi hàm draw\_box để vẽ một hộp giới hạn xung quanh đối tượng "Wally" trên hình ảnh image\_np. Kết quả trả về là fig (đối tượng hình vẽ13. ax.imshow(image\_np): Đoạn mã này hiển thị hình ảnh image\_np trên trục ax của hình vẽ.
* plt.show(): Đoạn mã này hiển thị hình vẽ và hình ảnh đã được vẽ lên.
* except Exception as e: logging.exception("Error in run\_wally\_finder: %s", e): Đoạn mã này xử lý các ngoại lệ và ghi log nếu có lỗi xảy ra trong quá trình chạy hàm run\_wally\_finder.

A screen shot of a computer program

Description automatically generated

Hàm main là hàm chính của ứng dụng. Nó tạo giao diện người dùng bằng thư viện PySimpleGUI và quản lý các sự kiện từ người dùng để thực hiện các thao tác như chọn hình ảnh và chạy mô hình phát hiện đối tượng "Wally"

* sg.theme('LightGrey1'): Đoạn mã này thiết lập chủ đề giao diện cho cửa sổ ứng dụng
* layout: Biến này chứa cấu trúc giao diện người dùng của ứng dụng. Nó được định nghĩa dưới dạng một danh sách các thành phần giao diện, bao gồm một vùng chọn hình ảnh, các nút điều khiển và một hình ảnh để hiển thị kết quả.
* window = sg.Window("Wally Finder App", layout, finalize=True): Đoạn mã này tạo một cửa sổ giao diện với tiêu đề "Wally Finder App" và cấu trúc giao diện được xác định bởi layout. Tham số finalize=True được sử dụng để hoàn thiện việc khởi tạo cửa sổ.
* while True:: Đây là một vòng lặp vô hạn để duy trì hiển thị cửa sổ và lắng nghe các sự kiện từ người dùng.
* event, values = window.read(): Đoạn mã này đọc sự kiện và giá trị từ cửa sổ. Biến event chứa tên sự kiện và values chứa các giá trị được nhập vào trong các thành phần giao diện.
* if event == sg.WIN\_CLOSED or event == "Exit": break: Đoạn mã này kiểm tra xem người dùng đã đóng cửa sổ hoặc nhấn nút "Exit" hay chưa. Nếu điều này đúng, vòng lặp sẽ bị thoát và chương trình kết thúc.
* if event == "Run Wally Finder": ...: Đoạn mã này kiểm tra xem người dùng đã nhấn nút "Run Wally Finder" hay chưa. Nếu điều này đúng, nó sẽ lấy đường dẫn hình ảnh từ values["image\_path"] và gọi hàm run\_wally\_finder để thực hiện phát hiện đối tượng.
* if event == "image\_path": ...: Đoạn mã này kiểm tra xem người dùng đã chọn một đường dẫn hình ảnh hay chưa. Nếu điều này đúng, nó sẽ cập nhật hình ảnh được hiển thị trong giao diện với đường dẫn hình ảnh mới.
* image = Image.pen(image\_path): Đoạn mã này mở hình ảnh từ đường dẫn image\_path bằng thư viện PIL.
* image.thumbnail((400, 400)): Đoạn mã này thu nhỏ hình ảnh để có kích thước tối đa là (400, 400). Điều này giúp hiển thị hình ảnh trong kích thước nhỏ hơn trên giao diện.
* bio = io.BytesIO(): Đoạn mã này tạo một đối tượng BytesIO để lưu trữ dữ liệu hình ảnh dưới dạng bytes.
* image.save(bio, format="PNG"): Đoạn mã này lưu hình ảnh đã thu nhỏ vào đối tượng BytesIO (bio) dưới định dạng PNG.
* window["-IMAGE-"].update(data=bio.getvalue()): Đoạn mã này cập nhật hình ảnh trong giao diện với dữliệu hình ảnh trong đối tượng BytesIO (bio) bằng cách sử dụng phương thức update của thành phần sg.Image.
* window.close(): Đoạn mã này đóng cửa sổ giao diện khi vòng lặp kết thúc.

A computer screen shot of text

Description automatically generated

Trong đoạn mã trên, có một điểm quan trọng để giải thích là if \_\_name\_\_ == "\_\_main\_\_":. Đoạn mã trong khối này chỉ được thực thi khi tập lệnh được chạy trực tiếp (tức là không phải là một module được import vào từ một module khác). Điều này cho phép đoạn mã trong khối này chỉ được thực thi khi chạy file chứa đoạn mã này như là một chương trình độc lập.

* import tensorflow as tf: Đoạn mã này import module TensorFlow để sử dụng trong việc xử lý mạng neural.
* import matplotlib.patches as patches: Đoạn mã này import module matplotlib.patches để sử dụng các đối tượng hình học như hình chữ nhật.
* detection\_graph = tf.Graph(): Đoạn mã này tạo một đồ thị TensorFlow mới để lưu trữ mô hình phát hiện đối tượng.
* with detection\_graph.as\_default():: Đoạn mã này đặt đồ thị TensorFlow vừa tạo làm đồ thị mặc định để thực hiện các phép tính liên quan đến mô hình phát hiện đối tượng.
* od\_graph\_def = tf.compat.v1.GraphDef(): Đoạn mã này tạo một đối tượng GraphDef để lưu trữ định nghĩa của đồ thị.
* with tf.io.gfile.GFile(model\_path, 'rb') as fid: ...: Đoạn mã này mở file model\_path (đường dẫn đến file chứa định nghĩa mô hình) và đọc nội dung file.
* serialized\_graph = fid.read(): Đoạn mã này đọc nội dung file và lưu trữ vào biến serialized\_graph.
* od\_graph\_def.ParseFromString(serialized\_graph): Đoạn mã này phân tích thông tin từ serialized\_graph và lưu trữ vào od\_graph\_def.
* tf.import\_graph\_def(od\_graph\_def, name=''): Đoạn mã này import định nghĩa đồ thị đã được phân tích vào đồ thị TensorFlow hiện tại.
* main(): Đoạn mã này gọi hàm main() để chạy ứng dụng.

1. **KẾT QUẢ**

A screenshot of a computer

Description automatically generated

A computer screen shot of a message

Description automatically generated

A screen shot of a computer

Description automatically generated**A large crowd of people

Description automatically generated**

A screenshot of a computer

Description automatically generated

# **CHƯƠNG 3**

# **KẾT LUẬN – ƯU ĐIỂM – HẠN CHẾ**

1. **KẾT LUẬN**

Phần mềm đã hoàn thành được yêu cầu của bài toán: nhận diện, phát hiện nhân vật Waldo trong một bức ảnh có nhiều yếu tố gây nhiễu. Qua đó, ta có thể áp dụng thuật toán này vào những công nghệ và các hoạt động khác như: camera an ninh, phát hiện tội phạm hoặc những chuẩn đoán bệnh trong y học.

1. **ƯU ĐIỂM**

Thời gian phát hiện đối tượng nhanh.

Khoanh vùng được đối tượng cụ thể.

Hoạt động được trên mọi loại máy nếu có đủ thư viện yêu cầu.

1. **HẠN CHẾ**

Không nhận diện được đối tượng thứ hai nếu cùng xuất hiện trong cùng một bức ảnh.

**CHƯƠNG 4**

**TÀI LIỆU THAM KHẢO**

Source

[TensorflowObjectDetectionAPI](https://github.com/tensorflow/models/tree/master/research/object\_detection)

[Training Images](https://github.com/vc1492a/Hey-Waldo)

Related projects

[Gathering and Analyzing Hardware Performance Data During Deep Network Training](https://github.com/amerus/BenchmarkingTensorflow/)