**HỌC VIỆN KỸ THUẬT QUÂN SỰ**

**HỌ VÀ TÊN: NGUYỄN VĂN QUYẾT**

**KHÓA: 15**

**HỆ ĐÀO TẠO: KỸ SƯ DÂN SỰ**

**ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC**

**CHUYÊN NGHÀNH: CÔNG NGHỆ GAME VÀ MÔ PHỎNG**

**Nghiên cứu một số phương pháp theo dõi đối tượng chuyển động và ứng dụng vào hệ thống giám sát mục tiêu**

**NĂM 2022**

**HỌC VIỆN KỸ THUẬT QUÂN SỰ**

**NGUYỄN VĂN QUYẾT**

**KHÓA 15**

**HỆ ĐÀO TẠO KỸ SƯ DÂN SỰ**

**ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC**

**NGHÀNH: KHOA HỌC MÁY TÍNH**

**MÃ SỐ: 52480101**

**Nghiên cứu một số phương pháp theo dõi đối tượng chuyển động và ứng dụng vào hệ thống giám sát mục tiêu**

***Cán bộ hướng dẫn khoa học: Trung tá, GV, TS.Hà Chí Trung***

**NĂM 2022**

|  |  |
| --- | --- |
| **HỌC VIỆN KỸ THUẬT QUÂN SỰ**  **KHOA CÔNG NGHỆ THÔNG TIN**  **BỘ MÔN KHOA HỌC MÁY TÍNH** | **CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc lập - Tự do - Hạnh phúc** |

**NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP**

Họ và tên: Nguyễn Văn Quyết, Lớp: CNG&MP15, Khóa: 15

Ngành: Khoa học máy tính, Chuyên ngành: Công nghệ game và mô phỏng

1. Tên đề tài:Nghiên cứu một số phương pháp theo dõi đối tượng chuyển động và ứng dụng vào hệ thống giám sát mục tiêu.

2. Các số liệu ban đầu:

- Quyết định Giao đồ án tốt nghiệp đại học – Học viện KTQS

- Tài liệu tham khảo

3. Nội dung bản thuyết minh:

- Mở đầu

- Chương 1: Tổng quan về bám đối tượng và ứng dụng.

- Chương 2: Thuật toán Yolo và DeepSort.

- Chương 3: Xây dựng chương trình và đánh giá kết quả.

- Kết luận và hướng phát triển.

- Tài liệu tham khảo

4. Số lượng, nội dung các bản vẽ (ghĩ rõ loại, kích thước và cách thực hiện các bản vẽ) và các sản phẩm cụ thể (nếu có):

5. Cán bộ hướng dẫn (ghi rõ họ tên, cấp bậc, chức vụ, đơn vị, hướng dẫn toàn bộ hay từng phần):

Cán bộ hướng dẫn : Hà Chí Trung

Cấp bậc : Trung tá

Chức vụ : Giảng viên

Đơn vị : Bộ môn Khoa học máy tính - Khoa Công nghệ thông tin.

Hướng dẫn toàn bộ.

Ngày giao: 06/01/2022 Ngày hoàn thành: 09/05/2022

*Hà Nội, ngày 09 tháng 05 năm 2022*

**Chủ nhiệm bộ môn Cán bộ hướng dẫn**

*(Ký, ghi rõ họ tên, học hàm, học vị)*

**TS. Hà Chí Trung**

|  |  |
| --- | --- |
|  | **Học viên thực hiện**  Đã hoàn thành và nộp đồ án, ngày 09 tháng 05 năm 2022  *(Ký và ghi rõ họ tên)*  **Nguyễn Văn Quyết** |

**KÝ HIỆU VIẾT TẮT**

|  |  |
| --- | --- |
| **Nội dung** | **Ký hiệu viết tắt** |
| Artificial Intelligence | AI |
| Machine Learning | ML |
| Deep Learning | DL |
| Object Detection | OD |
| Object Tracking | OT |
| Single Object Tracking | SOT |
| Multiple Object Tracking | MOT |
| Multiple Target Tracking | MTT |
| Detection Based Tracking | DBT |
| Detection Free Tracking | DFT |
| Convolutional Neural Network | CNN |
| Long Short Term Memory | LSTM |
| Recurent Neural Network | RNN |
| You Only Look Once | YOLO |
| Single Shot MultiBox Detector | SSD |
| Simple Online Realtime Object Tracking | SORT |
| Simple Online and Realtime Tracking with a Deep Association Metric | DeepSORT |
| False Positive | **FP** |
| False Negative | **FN** |
| Multiple Object Tracking Accuracy | **MOTA** |
| Multiple Object Tracking Precision | **MOTP** |
| Most Tracked Target | **MT** |
| Most Lost Target | **ML** |
| Frame Per Second | **FPS** |
| non-max suppression | **NMS** |

**DANH MỤC HÌNH ẢNH**

[Hình 1. 1: Bảng so sánh các mô hình phát hiện đối tượng phổ biến 7](#_Toc102774057)

[Hình 1. 2: Bảng so sánh các thuật toán MOT phổ biến. 7](#_Toc102774058)

[Hình 1.3: Lưu đồ thuật toán nhận dạng và bám đa đối tượng. 8](#_Toc102774059)

[Hình 2.1: Kiến trúc của YOLO. 9](#_Toc103154059)

[Hình 2.2: Các layer trong mạng darknet-53 10](#_Toc103154060)

[Hình 2.3: Kiến trúc một output của YOLO. 12](#_Toc103154061)

[Hình 2.4: Output của YOLO là 3 feature maps. 13](#_Toc103154062)

[Hình 2.5: Xác định anchor box cho một vật thể. 14](#_Toc103154063)

[Hình 2.6: Công thức ước lượng bounding box từ anchor box. 17](#_Toc103154064)

[Hình 2.7: non-max suppression. 17](#_Toc103154065)

[Hình 2.8: Mô tả sơ bộ thuật toán Hungary. 19](#_Toc103154066)

[Hình 2.9: Hình dung bước xử lý của Kalman filter. 21](#_Toc103154067)

[Hình 2. 10: Luồng xử lý của SORT. 23](#_Toc103154068)

[Hình 2. 11: Luồng xử lý của DeepSORT. 26](#_Toc103154069)

[Hình 2. 12: So sánh các thông số của SORT và DeepSORT. 27](#_Toc103154070)

[Hình 2.13: Độ đo khoảng cách của DeepSORT. 28](#_Toc103154071)

[Hình 2.14: Quá trình xây dựng chiến lược đối sánh theo tầng. 30](#_Toc103154072)

[Hình 2.15: Quản lý vòng đời của 1 track. 31](#_Toc103154073)

[Hình 3.2: Quá trình dán nhãn dữ liệu huấn luyện. 33](#_Toc102774049)

[Hình 3.3: Thư mục chứa dữ liệu ảnh đã được gán nhãn 34](#_Toc102774050)

[Hình 3.4: Cấu hình phần cứng Google Colab cung cấp 35](#_Toc102774051)

[Hình 3.5: Kết nối với google driver 36](#_Toc102774052)

[Hình 3.6: Kết quả train model ở vòng 1249. 36](#_Toc102774053)

[Hình 3.7: Kết quả từ thử nghiệm trong video. 46](#_Toc102774054)

[Hình 3.8: Kết quả từ việc từ việc tracking với camera. 47](#_Toc102774055)

[Hình 3.9: Bảng so sánh đánh giá các mô hình YOLOv4. 47](#_Toc102774056)

**MỤC LỤC**

[LỜI NÓI ĐẦU 1](#_Toc103684028)

[1. Lý do chọn đề tài. 1](#_Toc103684029)

[2. Mục đích nghiên cứu. 1](#_Toc103684030)

[3. Đối tượng nghiên cứu. 2](#_Toc103684031)

[4. Nhiệm vụ nghiên cứu. 2](#_Toc103684032)

[5. Phương pháp nghiên cứu. 2](#_Toc103684033)

[6. Phạm vi nghiên cứu. 2](#_Toc103684034)

[LỜI CẢM ƠN 3](#_Toc103684035)

[Chương 1 4](#_Toc103684036)

[TỔNG QUAN VỀ BÀI TOÁN BÁM ĐỐI TƯỢNG CHUYỂN ĐỘNG VÀ ỨNG DỤNG 4](#_Toc103684037)

[1.1. Bài toán bám đối tượng chuyển động 4](#_Toc103684038)

[1.2. Ứng dụng của bài toán bám nhiều đối tượng. 5](#_Toc103684039)

[1.3 Dataset và challenge 6](#_Toc103684040)

[1.4. Các vấn đề và hướng giải quyết bài toán bám đối tượng. 8](#_Toc103684041)

[1.4.1: Các vấn đề của bài toán bám nhiều đối tượng 8](#_Toc103684042)

[1.4.2: Hướng giải quyết. 10](#_Toc103684043)

[Chương 2 15](#_Toc103684044)

[BÁM ĐA ĐỐI TƯỢNG SỬ DỤNG YOLO VÀ DEEPSORT 15](#_Toc103684045)

[2.1. YOLO (You Only Look Once). 15](#_Toc103684046)

[2.1.1. YOLO là gì? 15](#_Toc103684047)

[2.1.2. Output của YOLO. 17](#_Toc103684048)

[2.1.3. Dự báo trên nhiều feature map. 18](#_Toc103684049)

[2.1.4. Anchor box 20](#_Toc103684050)

[2.1.5. Hàm loss function 21](#_Toc103684051)

[2.1.6. Dự báo bounding box 22](#_Toc103684052)

[2.1.7. Non-max suppression 23](#_Toc103684053)

[2.2. SORT (Simple Online Realtime Tracking) 24](#_Toc103684054)

[2.2.1. Giải thuật Hungary (Hungary Algorithm). 24](#_Toc103684055)

[2.2.2. Bộ lọc Kalman (Kalman Filter). 26](#_Toc103684056)

[2.2.3: Cách hoạt động của SORT 28](#_Toc103684057)

[2.3. DEEPSORT (Simple Online and Realtime Tracking with a Deep Association Metric) 31](#_Toc103684058)

[2.3.1. Các độ đo của DeepSORT 33](#_Toc103684059)

[2.3.2: Chiến lược đối sánh theo tầng và quản lý vòng đời 1 track 35](#_Toc103684060)

[2.3.3: Các thông số đánh giá. 37](#_Toc103684061)

[Chương 3 39](#_Toc103684062)

[XÂY DỰNG CHƯƠNG TRÌNH VÀ ĐÁNH GIÁ KẾT QUẢ 39](#_Toc103684063)

[3.1. Bộ dữ liệu huấn luyện YOLO. 39](#_Toc103684064)

[3.2. Môi trường và ngôn ngữ lập trình. 40](#_Toc103684065)

[3.2.1: Huấn luyện mô hình YOLO. 40](#_Toc103684066)

[3.2.2: Chương trình thử nghiệm. 41](#_Toc103684067)

[3.3. Cấu trúc chương trình. 41](#_Toc103684068)

[3.3.1: Quá trình huấn luyện mô hình YOLO. 41](#_Toc103684069)

[3.3.1.1: Chuẩn bị dữ liệu. 41](#_Toc103684070)

[3.3.1.2: Huấn luyện mô hình. 42](#_Toc103684071)

[3.3.2: Xây dựng chương trình bám đối tượng. 43](#_Toc103684072)

[3.3.3. Một số kết quả đạt được 49](#_Toc103684073)

[3.3.4: Đánh giá kết quả 51](#_Toc103684074)

[KẾT LUẬN 53](#_Toc103684075)

[TÀI LIỆU THAM KHẢO 55](#_Toc103684076)

# LỜI NÓI ĐẦU

## 1. Lý do chọn đề tài.

Xã hội ngày càng phát triển thì công nghệ cũng vàng phát triển theo từ các cuộc cách mạng công nghiệp đầu tiên cho đến hiện nay là cuộc cách mạng công nghiệp 4.0. Là cuộc cách mạng cốt lõi là công nghệ thông tin với các thiết bị thông minh như robot, nhà thông minh hay xe tự lái để tang năng suất lao động với mục đích phục vụ cho các nhu cầu đời sống con người. Tuy nhiên, vấn đề gặp phải chung ở đây là chúng vẫn còn rất ngây ngô để so chúng có trở nên thực sự thông minh thì con người phải dạy cho chúng hiểu và phân biệt được các sự vật trong thực tế nhằm đưa ra các vài toán xử lý chính xác nhằm đạt được mục đích công việc từ đó phát triển nên nhiều chương trình hay để giúp máy tính nhận biết được các đối tượng ngoài đời thực.

Trong lĩnh vực Thị giác máy tính (Computer Vision), việc phát hiện và theo dõi đối tượng đang là một lĩnh vực đang được quan tâm lớn. Trong những năm gần đây, những nỗ lực đáng kể đã được thực hiện để tìm kiếm các phương pháp nhận dạng và theo dõi đối tượng tốt nhất đang được nghiện cứu rất nhiều và có những kế quả đáng kể.

Xuất phát từ những lý do trên, đề tài: “Nghiên cứu một số phương pháp theo dõi đối tượng chuyển động và ứng dụng vào hệ thống giám sát mục tiêu.” tập trung vào tìm hiểu các kỹ thuật, mô hình học máy và học sâu cũng như khả năng áp dụng chúng với dữ liệu thu thập được trong đời sống con người.

## 2. Mục đích nghiên cứu.

Đồ Án này trình bày kỹ thuật khai thác dữ liệu dựa trên phương pháp mô hình phát hện đối tượng là Yolo kết hợp với theo dõi đa đối tượng là DeepSort để phát hiện và bám nhiều đối tượng có trong video.

Ngoài ra đồ án này cũng thảo luận về các phương pháp tiếp cận khai thác dữ liệu khác nhau đã được sử dụng để dự đoán và cũng tóm tắt về các phương pháp theo dõi đa đối tượng (các loại, mô hình, độ chính xác), thuật ngữ chung về theo dõi đa đối tượng, khai thác dữ liệu, thuật toán Yolo, và thuật toán DeepSort.

## 3. Đối tượng nghiên cứu.

* Thuật toán phát hiện đối tượng: YOLO
* Thuật toán theo dõi đa đối tượng: DeepSort
* Các bộ dữ liệu về theo dõi đa đối tượng: MOT [2], video đời sống thực tế.

## 4. Nhiệm vụ nghiên cứu.

* Nghiên cứu và hiểu được phương pháp theo dõi đối tượng sử dụng Yolo và DeepSort trong việc bám bắt đa đối tượng xuất hiện trong video.
* Lập trình và xây dựng chương trình bám đa đối tượng có trong video.

## 5. Phương pháp nghiên cứu.

* Phương pháp nghiên cứu tài liệu
* Tìm kiếm đọc các tài liệu, bài báo liên quan về Yolo (You Only Look Once) [5] và mở rộng hơn là các phiên bản mới nhỏ gọn để triển khai trên nhiều hệ thống nhỏ.
* Tìm kiếm đọc tài liệu liên, bài báo liên quan về thuật toán DeepSort [4]
* Tìm kiếm các bộ dữ liệu về theo dõi đa đối tượng phục vụ cho bài toán
* Phương pháp thực hành:
* Lập trình thử nghiệm bằng ngôn ngữ lập trình Python trên môi trường phát triển tích hợp Visual Studio Code hoặc Pycharm.
* Đánh giá độ chính xác của mô hình nghiên cứu

## 6. Phạm vi nghiên cứu.

Nghiên cứu ứng dụng thuật toán trong việc theo dõi đa đối tượng trong video được trính xuất từ camera có sẵn và mở rộng hơn là nhiều camera.

# LỜI CẢM ƠN

Để hoàn thành đồ án tốt nghiệp này, lời đầu tiên em xin cảm ơn chân thành đến toàn thể thầy cô trong khoa Công nghệ thông tin nói chung và bộ môn Khoa học máy tính nói riêng đã tận tình hướng dẫn, chỉ dạy và trang bị cho em những kiến thức bổ ích trong năm năm vừa qua.

Đặc biệt em xin chân thành gửi lời cảm ơn sâu sắc đến thầy giáo Tiến sĩ Hà Chí Trung, người đã trực tiếp hướng dẫn, trực tiếp chỉ bảo và tạo mọi điều kiện giúp đỡ em trong suốt quá trình làm đồ án tốt nghiệp.

Tuy đã rất cố gắng trong quá trình học tập, cũng như quá trình thực hiện đồ án tốt nghiệp, nhưng với điều kiện thời gian cũng như kinh nghiệm còn hạn chế của một sinh viên, đồ án này không thể tránh được những thiếu sót. Em rất mong nhận được sự chỉ bảo, đóng góp ý kiến của các thầy cô để em có điều kiện bổ sung, hoàn thiện kiến thức của bản thân hơn.

Em xin chân thành cảm ơn!

# Chương 1

# TỔNG QUAN VỀ BÀI TOÁN BÁM ĐỐI TƯỢNG CHUYỂN ĐỘNG VÀ ỨNG DỤNG

Chương 1 sẽ tìm hiểu tổng quan bài toán bám đối tượngchuyển động trong video, các hướng nghiên cứu và ứng dụng trong thực tế.

## 1.1. Bài toán bám đối tượng chuyển động

Những năm gần đây, khi mà khả năng tính toán của các máy tính được nâng lên một tầm cao mới và lượng dữ liệu khổng lồ được thu thập bởi các hãng công nghệ lớn, Học máy đã tiến thêm một bước dài và một lĩnh vực mới được ra đời gọi là Học sâu (Deep Learning). Học sâu đã giúp máy tính thực thi những việc tưởng chừng như không thể vào 10 năm trước: phân loại cả ngàn vật thể khác nhau trong các bức ảnh, tự tạo chú thích cho ảnh, theo dõi đối tượng cụ thể trong ảnh, bắt chước giọng nói và chữ viết của con người, giao tiếp với con người, hay thậm chí cả sáng tác văn hay âm nhạc.

Bài toán bám đối tượng chuyển động (Object Tracking - OT) là bài toán theo dõi một hay nhiều đối tượng chuyển động theo thời gian. Hiểu một cách đơn giản hơn nó là bài toán nâng cao hơn của phát hiện đối tượng (Object Detection - OD) khi đầu vào không chỉ là một ảnh mà là chuỗi ảnh và video.

Bài toán OT có thể chia thành 2 cách tiếp cận chính:

* Bám đơn đối tượng (**Single Object Tracking** - SOT) [1]: SOT tập trung vào việc theo dõi một đối tượng duy nhất trong toàn bộ chuỗi các khung hình hoặc video. Vì chỉ chú ý đến một đối tượng duy nhất nên việc cung cấp một bounding box từ ban đầu là việc bắt buộc phải có để trình theo dõi có thể hoạt động. SOT khá đơn gian khi chỉ cần triển khai theo dõi trên một đối tượng nhất định. Tuy nhiên ứng dụng không được lớn và giành được sự quan tâm nhiều của giới nghiên cứu.
* Bám đa đối tượng (**Multiple Object Tracking** - MOT) [2]: MOT hướng tới các ứng dụng có tính mở rộng cao hơn. Bài toán cố gắng phát hiện đồng thời theo dõi tất cả các đối tượng trong tầm nhìn, kể cả các đối tượng mới xuất hiện trong khung hình/video. MOT có thể được xem là một bài toán mở rộng và phát triển sâu của SOT. MOT có nhiều ứng dụng thực tế và dễ dàng áp dụng triển khai hơn SOT. Vì điểu này, MOT thường là những bài toán khó hơn SOT và nhận được rất nhiều sự quan tâm của giới nghiên cứu.

Ngoài ra còn phân chia theo:

* Theo dõi dựa trên phát hiện (Detection Based Tracking – DBT): Tập trung vào mối liên kết chặt chẽ giữa OD và OT, từ đó dựa vào các kết quả của việc phát hiện để theo dõi đối tượng qua các frame.
* Theo dõi không phụ thuộc phát hiện (Detection Free Tracking – DFT): Coi video như 1 dạng dữ liệu dạng chuỗi, từ đó, áp dụng những phương pháp dành riêng cho chuỗi như RNN, LSTM...

Đồ án tốt nghiệp này sẽ đi sâu vào việc nghiên cứu bài toán theo dõi đa đối tượng và ứng dụng của nó.

## 1.2. Ứng dụng của bài toán bám nhiều đối tượng.

Theo dõi đa đối tượng đang được giới nghiên cứu rất quan tâm trong thời gian gần đây. Nhu cầu ứng dụng bài toán bám đối tượng trong thời kì công nghệ phát triển là rất lớn. Nó được sử dụng trong nhiều tình huống như đánh giá lưu lượng giao thông. Giữ khoảng cách giữa người với người đi bộ trên phố trong thời kì COVID. Theo dõi các vận động viên thể thao qua đó phân tích chiến thuật, đánh giá các chỉ số cơ thể. Theo dõi và đánh giá các nhóm động vật, đánh giá mật độ phát triển của loài,... Nhìn chung, 70% các nghiên cứu vể MOT là bài toán nghiên cứu về người đi bộ di chuyển và đánh giá quỹ đạo cũng như lưu lượng trên một khu vực nhất định. Do đó đây vẫn là đề tài cụ thể giành được nhiều người quan tầm nhất trong MOT.

Trong các trường hợp giám sát các đối tượng thuộc một danh mục nhất định, chẳng hạn như người hoặc xe hơi, máy dò được sử dụng để giúp theo dõi dễ dàng hơn. Thông thường, bài toán bám đối tượng được thực hiện theo hai bước: Phát hiện đối tượng và theo dõi thông qua các thuật toán tracking.

## 1.3 Dataset và challenge

Việc nắm bắt về các tập dữ liệu phổ biến cũng như các metric đánh giá là điều không thể thiếu. Về dataset, MOT thường đánh giá dựa trên các tập dữ liệu sau:

* MOT Challenge: MOT Challenge là một cuộc thi thường niên, dataset của MOT Challenge thường được sử dụng để đánh giá điểm chuẩn cho các phương pháp giải quyết bài toán Mutiple Object Tracking. (MOT15, MOT16, MOT17, MOT20, ...). Dữ liệu là chuỗi các video đã được gán nhãn của người đi bộ, được thu thập từ nhiều nguồn khác nhau, với sự đa dạng về độ phân giải, độ chiếu sáng,... MOT là một cuộc thi luôn dành được nhiều sự quan tâm từ giới nghiên cứu. Các thuật toán theo dõi đối tượng luôn nhắm tới mục tiêu giành được kết quả cao nhất trong các cuộc thi MOT Challenge. Tầm nhìn tương lai của MOT Challenge là thiết lập nó như một nền tảng chung để đánh giá theo dõi nhiều đối tượng, mở rộng ra ngoài theo dõi người đi bộ. Để đạt được điều này, cuộc thi không chỉ nhắm tới bài toán theo dõi 2D mà đã mở rộng ra bài toán trong không gian thực 3D. Bộ dữ liệu này bao gồm 2907 video, bao gồm 833 lớp bởi 17.287 bài hát. Đây được xem như bài đánh giá chuẩn nhất cho các thuật toán MOT và các thuật toán MOT ra đời cũng là để giành được kết quả cao nhất từ MOT Challenge. MOT Challenge cũng là một nền tảng để so sánh công bằng các phương pháp theo dõi hiện đại. Bằng cách cung cấp cho các tác giả dữ liệu sự thật cơ bản được tiêu chuẩn hóa, số liệu đánh giá, kịch bản, cũng như một tập hợp các phát hiện được đặt trước, tất cả các phương pháp được so sánh trong cùng một điều kiện, do đó cô lập hiệu suất của trình theo dõi từ các yếu tố khác. Trong quá khứ, một số lượng lớn các số liệu để đánh giá định lượng theo dõi nhiều mục tiêu đã được đề. Một mặt, mong muốn tóm tắt hiệu suất thành một số duy nhất để cho phép so sánh trực tiếp giữa các phương pháp. Mặt khác, người ta có thể muốn cung cấp các ước tính hiệu suất thông tin hơn bằng cách nêu chi tiết các loại lỗi mà các thuật toán thực hiện, điều này ngăn cản thứ hạng rõ ràng. Các biện pháp đánh giá này đưa ra một cái nhìn bổ sung về hiệu suất theo dõi. Đại diện chính của các biện pháp CLEAR-MOT, Độ chính xác theo dõi đa đối tượng (MOTA), được đánh giá dựa trên sự kết hợp khung hình giữa dự đoán theo dõi và sự thật mặt đất. Nó rõ ràng phạt các công tắc nhận dạng giữa các khung hình liên tiếp, do đó chỉ đánh giá hiệu suất theo dõi tại địa phương. Biện pháp này có xu hướng nhấn mạnh hơn vào hiệu suất phát hiện đối tượng so với tính liên tục của thời gian. Ngược lại, các biện pháp chất lượng theo dõi và IDF1 Ristani et al, thực hiện kết hợp dự đoán với sự thật trên mặt đất ở cấp độ quỹ đạo và nhấn mạnh quá mức khía cạnh liên tục thời gian của hiệu suất theo dõi. Trong phần này, trước tiên chúng tôi giới thiệu sự phù hợp giữa theo dõi dự đoán và chú thích sự thật mặt đất trước khi chúng tôi trình bày các biện pháp cuối cùng. Tất cả các kịch bản đánh giá được sử dụng trong điểm chuẩn của chúng tôi đều có sẵn công khai.
* ImageNet VID:  Bên cạnh những bộ dataset nổi tiếng về classification, object detection, ImageNet cũng cung cấp một bộ dataset đủ lớn về object tracking. Đây là chuỗi các video được gán nhãn từ 30 nhóm đối tượng khác nhau. [ImageNet](https://www.thegioimaychu.vn/blog/thuat-ngu/imagenet/?utm_source=tmblog&utm_medium=inlinekwd) là một cơ sở dữ liệu hình ảnh quy mô lớn được thiết kế để sử dụng trong nghiên cứu phần mềm nhận dạng đối tượng trực quan. Trên 14 triệu URL của hình ảnh đã được gán nhãn bằng tay bởi [ImageNet](https://www.thegioimaychu.vn/blog/thuat-ngu/imagenet/?utm_source=tmblog&utm_medium=inlinekwd) để chỉ ra đối tượng nào có trong những bức hình. ImageNet chứa hơn 20 nghìn danh mục, một danh mục điển hình, chẳng hạn như “quả bóng” hoặc “dâu tây”, chứa hàng trăm hình ảnh. Cơ sở dữ liệu nhãn hình ảnh của URL từ bên thứ ba cũng có sẵn miễn phí từ ImageNet; tuy nhiên, các hình ảnh thực tế không thuộc sở hữu của ImageNet. Từ năm 2010, dự án ImageNet thực hiện một cuộc thi phần mềm hàng năm, Thử thách nhận diện trực quan quy mô lớn của ImageNet (ImageNet Large Scale Visual Recognition Challenge – ILSVRC), nơi các phần mềm cạnh tranh để phân loại và phát hiện các đối tượng và cảnh vật một cách chính xác. ILSVRC sử dụng danh sách đã được “cắt xén” của một nghìn phân lớp không chồng chéo lên nhau.

## 1.4. Các vấn đề và hướng giải quyết bài toán bám đối tượng.

Tính năng phát hiện đối tượng đã phát triển đáng kể trong hai thập kỷ qua, với việc chuyển từ các phương pháp thống kê hoặc học máy truyền thống sang phương pháp tiếp cận học sâu dựa trên Convolution Neueral Network (CNN). Sự ra đời của học sâu đã cải thiện độ chính xác và khả năng phân tích của việc phát hiện đối tượng theo một thứ tự độ lớn

### 1.4.1: Các vấn đề của bài toán bám nhiều đối tượng

Các phương pháp MOT cố gắng hướng đến việc theo dõi tất cả các đối tượng xuất hiện trong khung hình bằng việc phát hiện và gắn định danh cho từng đối tượng. Bên cạnh đó, các ID đã được gán cho 1 đối tượng cần đảm bảo nhất quán qua từng khung hình. Vì vậy nó phát sinh một số vấn đề:

* Phát hiện "tất cả" các đối tượng: Đây vẫn luôn là vấn đề được quan tâm nhất trong OD và vẫn không ngừng có những phương pháp, những thuật toán cải thiện vấn đề này. Việc áp dụng đột phá và nhanh chóng của Deep Learning vào năm 2012 đã đưa vào sự tồn tại các thuật toán và phương pháp phát hiện đối tượng hiện đại và chính xác cao như R-CNN, Faster-RCNN, RetinaNet và nhanh hơn nhưng rất chính xác như SSD và YOLO. Sử dụng các phương pháp và thuật toán này, dựa trên Deep Learning và cũng dựa trên việc học máy đòi hỏi rất nhiều kiến thức về toán học và việc học sâu. Có hàng triệu chuyên gia lập trình và các nhà phát triển phần mềm muốn tích hợp và tạo ra các sản phẩm mới sử dụng OD. Trong OT, đặc biệt là DBT, việc đảm bảo tính chính xác của quá trình nhận dạng đối tượng cũng vô cùng quan trọng. Đây là bước quyết định xem thuật toán liệu có tốt hay không, có ứng dụng được hay không. Một thuật toán OD tốt là bước khởi đầu cho một bài toán theo dõi tốt.
* Đối tượng bị che khuất 1 phần hoặc toàn bộ: Khi có 1 ID được gán cho 1 đối tượng, ID cần đảm bảo nhất quán trong suốt video. Tuy nhiên, khi một đối tượng bị che khuất, nếu chỉ dựa riêng vào OD là không đủ để giải quyết vấn đề này. Việc đảm bảo đối tượng luôn được phát hiện và theo dõi chính xác luôn là bài toán khó cho việc theo dõi đối tượng. Phải đảm bảo ID luôn được nhận dạng chính xác và trong suốt video.
* Đối tượng ra khỏi phạm vi của khung hình và sau đó xuất hiện lại: Tương tự như vấn đề trước đó, đây là vấn đề về chỉ số ID\_switches. Cần giải quyết tốt vấn đề nhận dạng lại đối tượng kể cả việc che khuất hay biến mất để giảm số lượng ID\_switches xuống mức thấp nhất có thể. Trong khi đó khi đối tượng xuất hiện trở lại khung hình cũng cần phải đánh giá đúng và gán chính xác ID cho đối tượng đó. Đây là việc làm cần thiết để đảm bảo tính nhất quán và chính xác cho bài toán tracking.
* Các đối tượng có quỹ đạo chuyển động giao nhau hoặc chồng chéo lên nhau: Việc các đối tượng có quỹ đạo chống chéo lên nhau cũng có thể dẫn đến hậu quả gán nhầm ID cho các đối tượng, đây cũng là vấn đề cần chú ý xử lí khi làm việc với MOT. Các thuật toán hiện tại đang gặp vấn đề lớn khi hai đối tượng di chuyển cùng quỹ đạo và chồng chéo lên nhau. Việc gán nhầm ID khi gặp tình trạng này rất dễ xảy ra. Đặc biệt khi hai đối tượng có cùng một đặc điểm nhận dạng khiến cho các mạng Extract Feature dễ nhầm lần giữa các đối tượng với nhau khiến cho bài toán bị sai lệch đi kết quả.
* **Background distortion**: những backgrounds phức tạp sẽ gây khó khăn trong việc phát hiện những đối tượng nhỏ. Đối tượng mang màu sắc trùng với nền có thể gây khó khăn cho việc phát hiện và theo dõi trong khung hình. Đó không chỉ là vấn đề xác định toạ độ đối tượng mà còn là việc xem xét đối tượng đã ra khỏi khung hình hay chưa và quyết định có xoá bỏ khung hình khỏi trình theo dõi hay không.
* Image Illumnination: Độ chiếu sáng có ảnh hưởng quan trọng trong bài toán detection và recognition. Cùng một đối tượng sẽ trông khác nhau phụ thuộc vào điều kiện độ sáng. Việc độ sáng không đảm bảo có thể khiến đối tượng không được nhận dạng và phân lớp chính xác để trình theo dõi có thể theo dõi hay loại bỏ đối tượng
* Multiple Spatial Spaces, Deformation, or Object rotation: Đối tượng có thể bị bóp méo, thay đổi hình dạng, kích thước, tỉ lệ khung hình hoặc xoay theo những cách không thể biết trước, điều này làm gia tăng độ phức tạp cũng như gây hỗn loạn cho bài toán OD. Những detectors phải được train trên tập dữ liệu đa dạng có thể phủ tất cả các trường hợp nêu trên để có thể đạt được độ chính xác cao.
* Motion blurring: Sự nhiễu hoặc nhòe hình ảnh đọc từ đầu vào gây ra bởi sự chuyển động của camera cũng có thể gây khó khăn cho việc theo dõi và phát hiện đối tượng.

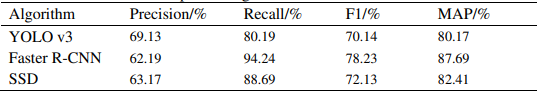
### 1.4.2: Hướng giải quyết.

Để giải quyết vấn đề đầu tiên về việc phát hiện toàn bộ các đối tượng trong khung hình. Bắt đầu từ năm 2012, sau khi mạng AlexNet giành giải nhất cuộc thi 2012 ILSVRC, mọi nghiên cứu về phân lớp dữ liệu đều sử dụng mạng CNN. Kể từ đó đến đây, CNN được coi như là thuật toán thống trị trên mọi publish paper về các bài toán phân lớp đối tượng. Trong khi đó, để nhận dạng 1 đối tượng trong ảnh, các đơn giản nhất là thiết lập một cửa sổ trượt có kích thước là window size trượt từ trái qua phải, từ trên xuống dưới, quét qua toàn bộ bức ảnh. Để phát hiện các đối tượng khác nhau ở các góc nhìn khác nhau, chúng ta sẽ sử dụng cửa sổ trượt có kích thước thay đổi và ảnh đầu vào có kích thước thay đổi. Các thuật toán học sâu đều cố gắng đặt ra mục tiêu có thể phát hiện toàn bộ các đối tượng và vẽ Bounding Box toạ độ cho các đối tượng trong khung hình. Bài toán OD là một vấn đề bao gồm hồi quy và phân lớp. Đầu tiên, để tính độ chính xác về vị trí object trong ảnh, ta cần loại bỏ các box có độ tin cậy thấp vì các mô hình thường cho nhiều kết quả hơn nhiều so với thực tế. Do đó chúng ta sử dụng Intersesion over Union (IoU) area, giá trị nằm giữa 0 và 1. Nó tương ứng là tỉ lệ vùng chồng lấn giữa kết quả dự đoán và thực tế.

Trong bài toán phân loại nhị phân, độ đo Average Precision (AP) là tổng của precision-recall curve. Độ đo thường được sử dụng trong OD là mean Average Precision (mAP). Nó là sự đơn giản hóa thành trung bình cộng của AP tính trên tất cả các classes.

mAP thường được tính cho một giá trị cố định của IoU. Tuy nhiên nếu giá trị của bounding bõx lớn có thể làm tăng số lương bounding boxes phù hợp. Do đó COCO challenge đã tính trung bình với IoU thay đổi để làm giảm ảnh hưởng của việc số lượng bounding boxes lớn nhưng phân loại sai.

Việc tìm ra một mô hình OD phù hợp đang nhận được sự quan tâm khá lớn. Trong vài năm trở lại đây, OD hay nhận dạng đối tượng là một trong những đề tài rất hot của Deep Learning bởi khả năng ứng dụng cao, dữ liệu dễ chuẩn bị và kết quả ứng dụng thì cực kì nhiều. Các thuật toán mới của OD như YOLO [5], SSD [7], Faster RCNN [8] có tốc độ khá nhanh và độ chính xác cao nên giúp cho OD có thể thực hiện được các tác vụ dường nhanh hơn so với con người mà độ chính xác không giảm. Các mô hình cũng trở nên nhẹ hơn nên có thể hoạt động trên các thiết bị IoT để tạo nên các thiết bị thông minh. Việc các mô hình ngày được cải tiến độ chính xác đáng kể cũng khiến cho việc nhận dạng các đối tượng trong một khung hình ngày càng tốt hơn và chính xác hơn.



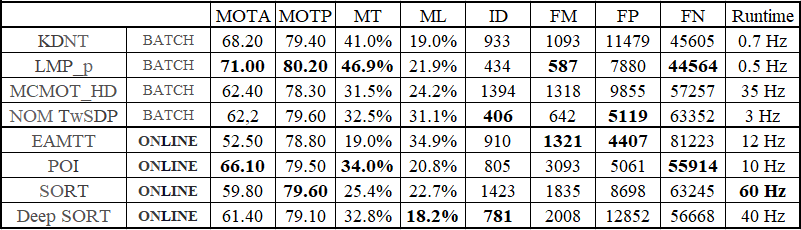
Hình 1. 1: Bảng so sánh các mô hình phát hiện đối tượng phổ biến

Bên cạnh đó việc giới thiệu mô hình YOLOv4 [6] trong thời gian gần đây là một phát triển đột phá mới của OD. Nó là một nó là một two-stage detector với nhiều components khác nhau. Cấu trúc mới này được chia làm 3 phần:

* **Backbone**: là một deep learning architecture hoạt động như là một feature extractor. Tất cả các backbone về cơ bản là các classification models. **VGG16**là một ví dụ điển hình – một trong những deep learning classifiers ra đời sớm nhất. **SqueezeNet, MobileNet, ShuffleNet** cũng là những backbone model phổ biến ngoài những cái đã được đề cập ở bên trên, tuy nhiên chúng chỉ có thể thực hiện training trên CPU.
* **Neck:**cũng là một điểm riêng biệt của YOLO, về cơ bản thì nó thu thập feature maps tại những giai đoạn khác nhau của backbone. Nói một cách đơn giản, nó là một feature aggreagator. Chúng ta sẽ bàn về neck trong object detection pipeline chi tiết hơn ở trong các phần sau.
* **Head:**được biết đến như là một object detector. Nó sẽ tìm những vùng có khả năng chứa object nhưng sẽ không xác định cho ta biết chính xác object là gì. Chúng ta có two-stage detectors và one-stage detectors (trong đó lại tiếp tục chia ra thành anchor-based và anchor-free detectors). Head sẽ được bàn thêm ở những phần sau.

Mô hình mới này có rất nhiều các kết hợp để hình thành nên một architecture hoàn chỉnh. Tiêu chí lựa chọn phải dựa trên sự cân bằng tối ưu giữa kích thước đầu vào, số lượng convolution layers, số lượng parameters và số lượng các bộ lọc. Hơn nữa, việc thêm vào các block để tăng thêm receptive field và chọn ra backbone và neck tốt nhất đối với bài toán làm gia tăng đáng kể cho độ chính xác của thuật toán qua đó khiến cho việc phát hiện đối tượng trong khung hình trở lên chính xác và gia tăng tốc độ dự đoán cùng với việc dễ dàng triển khai.

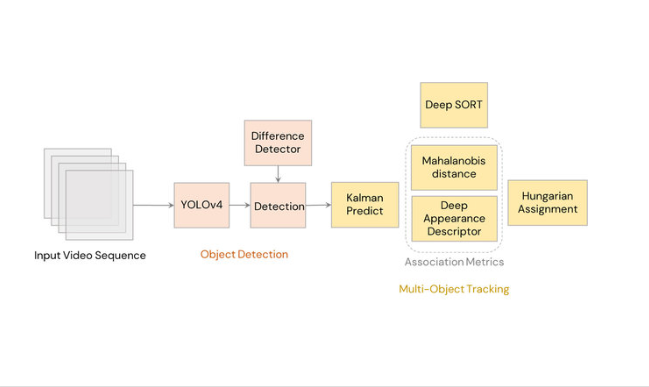
Ngoài ra, đối với các thuật toán bám đa đối tượng mới liên tục được đề xuất cũng giúp cho việc bám đối tượng và theo dõi quá trình di chuyển trong khung hình trở lên dễ dàng và chính xác hơn. Việc cho ra đời nhiều thuật toán MOT đã khiến cho việc khắc phục được hầu nhu các lỗi còn tồn tại trên bài toán này. Các thuật toán như SORT[3], POI [9], RNN\_LSTM [10] hay DeepSORT[4]... liên tục được giới thiệu và cải tiến đã khiến cho việc bám nhiều đối tượng đạt độ chính xác cao hơn trong các cuộc thi và ứng dụng thực tiễn. Việc các thuật toán ra đời nhằm giải quyết dần các vấn đề của MOT cũng là một bước tiến lớn đang được phát triển mạnh.

****

Hình 1. 2: Bảng so sánh các thuật toán MOT phổ biến.

Từ bảng thông số trên có thể thấy DeepSORT là một thuật toán phù hợp cho việc xây dựng chương trình bám da đối tượng. **Deep SORT là phương pháp tracking theo cascade-style (kiểu thác nước).** Đầu tiên chúng thực hiện dự đoán những bounding boxes đối với những đối tượng được phát hiện và sau đó trích xuất các features từ chúng để ước tính các re-ID features tương ứng. **Deep SORT cho phép thực hiện truy vết đối với những đối tượng bị chồng lấn trong khoảng thời gian dài hơn, đơn giản để implement và chạy trong real-time.** Deep SORT cũng tuân theo một phương pháp tracking sử dụng**recursive Kalman filering và liên kết dữ liệu frame-by-frame sử dụng thuật toán Hungarian.** Với việc đạt được Multiple Object Tracking Accuracy - MOTA và Multiple Object Tracking Precision - MOTP khá tốt đồng thời giải quyết được vấn đề Multiple Object Tracking Lost Target - ML giúp cho việc ứng dụng thuật toán trở nên dễ dàng và hiệu quả hơn.

Từ đây, hướng đi của đề tài sẽ theo mô hình OD là YOLO và kết hợp với thuật toán phổ biến của MOT là DeepSORT (Sẽ trình bày kỹ hơn ở Chương 2). Nội dung tiếp theo sẽ nghiên cứu chuyên sâu về thuật toán này. Dưới đây sẽ là luồng xử lý của bài toán:



Hình 1.3: Lưu đồ thuật toán nhận dạng và bám đa đối tượng.

# Chương 2

# BÁM ĐA ĐỐI TƯỢNG SỬ DỤNG YOLO VÀ DEEPSORT

Chương 2 sẽ tập trung nghiên cứu chuyên sâu về mô hình Yolo và thuật toán DeepSORT, ứng dụng trong bài toán bám nhiều đối tượng chuyển động.

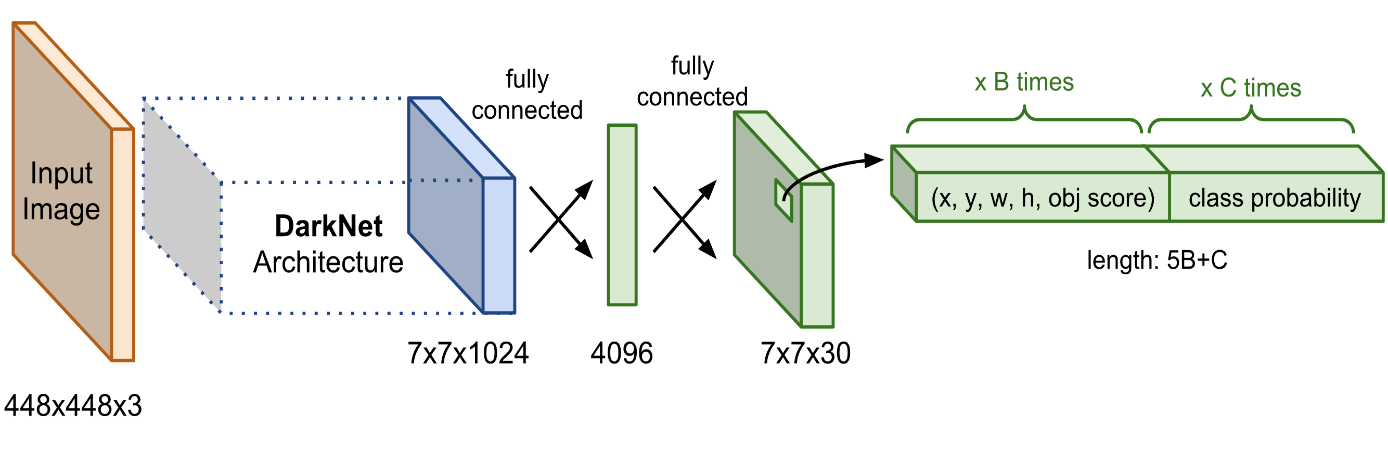
## 2.1. YOLO (You Only Look Once).

### 2.1.1. YOLO là gì?

Object Detection là một bài toán quan trọng trong lĩnh vực Computer Vision, thuật toán OD được chia thành 2 nhóm chính:

* Họ các mô hình RCNN (Region-Based Convolutional Neural Networks) để giải quyết các bài toán về định vị và nhận diện vật thể.
* Họ các mô hình về YOLO (You Only Look Once) dùng để nhận dạng đối tượng được thiết kế để nhận diện các vật thể thời gian thực.

YOLO trong OD có nghĩa là “You Only Look Once”. Tức là chúng ta chỉ cần nhìn 1 lần là có thể phát hiện ra vật thể. YOLO là thuật toán nhận dạng đối tượng nên mục tiêu của mô hình không chỉ là dự báo nhãn cho vật thể như các bài toán phân lớp mà nó còn xác định vị trí của vật thể. Do đó YOLO có thể phát hiện được nhiều vật thể có nhãn khác nhau trong một bức ảnh thay vì chỉ phân loại duy nhất một nhãn cho một bức ảnh.

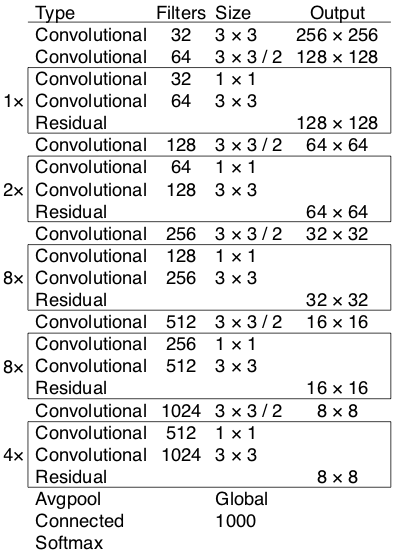


Hình 2.1: Kiến trúc của YOLO.

Kiến trúc YOLO bao gồm:

* Base network là các mạng Convolution làm nhiệm vụ trích xuất đặc trưng
* Phần phía sau là những Extra Layers được áp dụng để phát hiện vật thể trên feature map của base network.

Trong YOLO tác giả áp dụng một mạng feature extractor là darknet-53. Mạng này gồm 53 convolutional layers kết nối liên tiếp, mỗi layer được theo sau bởi một batch normalization và một activation Leaky Relu. Để giảm kích thước của output sau mỗi convolution layer, tác giả down sample bằng các filter với kích thước là 2. Mẹo này có tác dụng giảm thiểu số lượng tham số cho mô hình.



Hình 2.2: Các layer trong mạng darknet-53

Các bức ảnh khi được đưa vào mô hình sẽ được scale để về chung một kích thước phù hợp với input shape của mô hình và sau đó được gom lại thành batch đưa vào huấn luyện.

Hiện tại YOLO đang hỗ trợ 2 đầu vào chính là 416 x416 và 608x608. Mỗi một đầu vào sẽ có một thiết kế các layers riêng phù hợp với shape của input. Sau khi đi qua các layer convolutional thì shape giảm dần theo cấp số nhân là 2. Cuối cùng thu được một feature map có kích thước tương đối nhỏ để dự báo vật thể trên từng ô của feature map.

Kích thước của feature map sẽ phụ thuộc vào đầu vào. Đối với input 416x416 thì feature map có các kích thước là 13x13, 26x26 và 52x52. Và khi input là 608x608 sẽ tạo ra feature map 19x19, 38x38, 72x72.

### 2.1.2. Output của YOLO.

Output của mô hình YOLO là một véc tơ sẽ bao gồm các thành phần:

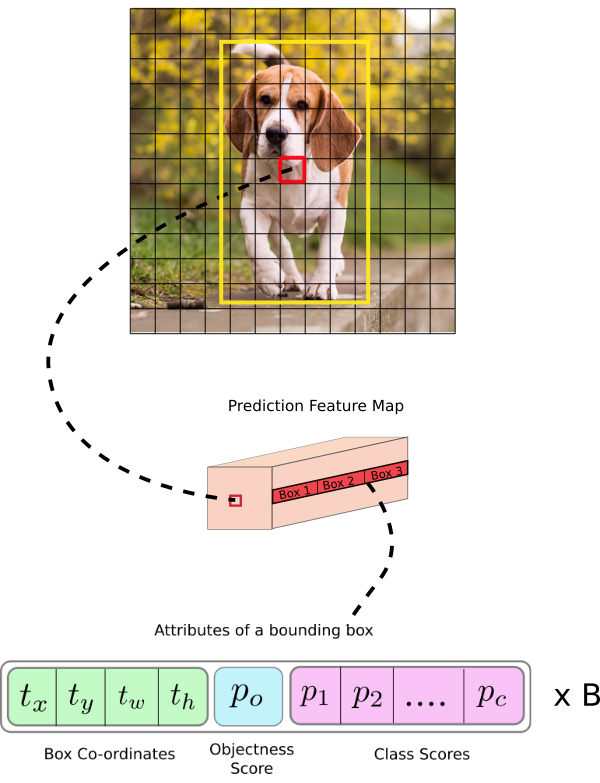
|  |  |
| --- | --- |
|  | (2.1) |

trong đó:

* là xác suất dự báo vật thể xuất hiện trong bounding box.
* giúp xác định bounding box. Trong đó là tọa độ tâm và là kích thước rộng, dài của bounding box.
* là véc tơ phân phối xác suất dự báo của các classes.

Output sẽ được xác định theo số lượng bằng công thức (n\_class + 5). Nếu huấn luyện 80 classes output là 85. Trường hợp áp dụng 3 anchors/cell thì số lượng tham số output sẽ là: (n\_class + 5) x 3 = 85 x 3 =255.

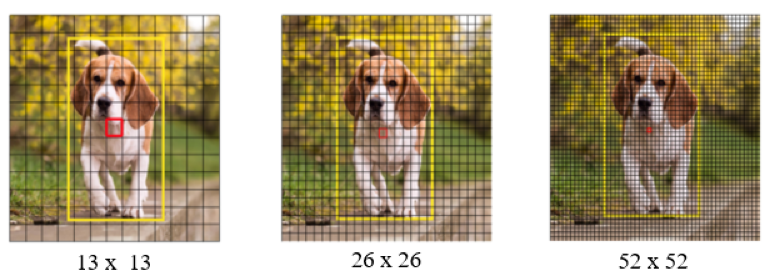
Trên mỗi một cell của feature map chúng ta lựa chọn ra 3 anchor boxes với kích thước khác nhau lần lượt là Box 1, Box 2, Box 3 sao cho tâm của các anchor boxes trùng với cell. Khi đó output của YOLO là một vector concatenate của 3 bounding boxes. Được mô tả như hình dưới đây.



Hình 2.3: Kiến trúc một output của YOLO.

### 2.1.3. Dự báo trên nhiều feature map.

Cũng tương tự như SSD, YOLO dự báo trên nhiều feature map. Những feature map ban đầu có kích thước nhỏ giúp dự báo được các object kích thước lớn. Những feature map sau có kích thước lớn hơn trong khi anchor box được giữ cố định kích thước nên sẽ giúp dự báo các vật thể kích thước nhỏ.



Hình 2.4: Output của YOLO là 3 feature maps.

Trên mỗi một cell của các feature map chúng ta sẽ áp dụng 3 anchor box để dự đoán vật thể. Như vậy số lượng các anchor box khác nhau trong một mô hình YOLO sẽ là 9 (3 feature map x 3 anchor map).

Đồng thời trên một feature map hình vuông s x s, mô hình YOLO sinh ra một số lượng anchor box là: s x s x 3. Như vậy số lượng anchor boxes trên một bức ảnh sẽ là: (13 x 13 + 26 x 26 + 52 x 52) x 3 = 10647 (anchor box)

Đây là một số lượng rất lớn và là nguyên nhân khiến quá trình huấn luyện mô hình YOLO vô cùng chậm bởi chúng ta cần dự báo đồng thời nhãn và bounding box trên đồng thời 10647 bounding boxes.

Một số lưu ý khi huấn luyện YOLO [14]:

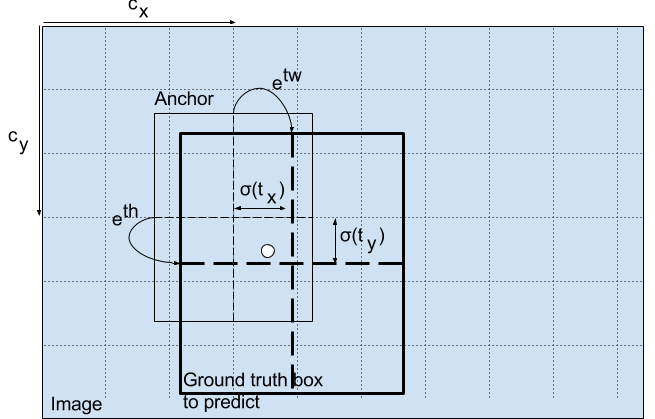
* Khi huấn luyện YOLO sẽ cần phải có RAM dung lượng lớn để save được 10647 bounding boxes như trong kiến trúc này.
* Không thể thiết lập các batch\_size quá lớn như trong các mô hình classification vì rất dễ Out of memory. Package darknet của YOLO đã chia nhỏ một batch thành các subdivisions cho vừa với RAM.

Thời gian xử lý của một bước trên YOLO lâu hơn rất rất nhiều lần so với các mô hình classification. Do đó nên thiết lập steps giới hạn huấn luyện cho YOLO nhỏ. Đối với các tác vụ nhận diện dưới 5 classes, dưới 5000 steps là có thể thu được nghiệm tạm chấp nhận được. Các mô hình có nhiều classes hơn có thể tăng số lượng steps theo cấp số nhân.

### 2.1.4. Anchor box

Để tìm được bounding box cho vật thể, YOLO sẽ cần các anchor box làm cơ sở ước lượng. Những anchor box này sẽ được xác định trước và sẽ bao quanh vật thể một cách tương đối chính xác. Sau này thuật toán regression bounding box sẽ tinh chỉnh lại anchor box để tạo ra bounding box dự đoán cho vật thể. Trong một mô hình YOLO:

* Mỗi một vật thể trong hình ảnh huấn luyện được phân bố về một anchor box. Trong trường hợp có từ 2 anchor boxes trở lên cùng bao quanh vật thể thì ta sẽ xác định anchor box mà có loU với ground truth bounding box là cao nhất.



Hình 2.5: Xác định anchor box cho một vật thể.

* Mỗi một vật thể trong hình ảnh huấn luyện được phân bố về một cell trên feature map mà chứa điểm mid point của vật thể. Chẳng hạn như hình chú chó trong hình bên trên sẽ được phân về cho cell màu đỏ vì điểm mid point của ảnh chú chó rơi vào đúng cell này. Từ cell ta sẽ xác định các anchor boxes bao quanh hình ảnh chú chó.

Như vậy khi xác định một vật thể ta sẽ cần xác định hai thành phần gắn liền với nó là (cell, anchor box). Không chỉ riêng mình cell hoặc chỉ mình anchor box.

Một số trường hợp ngoại lệ khi 2 vật thể bị trùng mid point, mặc dù rất hiếm khi xảy ra, thuật toán sẽ rất khó xác định được class cho chúng.

### 2.1.5. Hàm loss function

Hàm loss function của YOLO chia thành 2 phần: localization loss đo lường sai số của bounding box và confidence loss đo lường sai số của phân phối xác suất các classes.

|  |  |
| --- | --- |
|  | *(2.2)* |

* : Hàm indicator có giá trị 0, 1 nhằm xác định xem cell I có chứa vật thể hay không. Bằng 1 nếu chứa vật thể và bằng 0 nếu không chứa.
* : Cho biết bounding box thứ j của cell I có phải là bounding box của vật thể được dự đoán hay không?
* : Điểm tin cậy của ô i, P (contain object) \* IoU (predict bbox, ground truth box).
* : Điểm tự tin dự đoán.
* C: Tập hợp tất cả các lớp.
* : Xác suất có điều kiện, có hay không ô i có chứa một đối tượng của lớp.
* : Xác suất có điều kiện dự đoán

Có thể ban đầu công thức trên khá khó hiểu với người bắt đầu. Chúng ta hãy hiểu đơn giản hóa mục đích của chúng:

* là hàm mất mát của bounding box dự báo so với thực tế.
* là hàm mất mát của phân phối xác suất. Trong đó tổng đầu tiên là mất mát của dự đoán có vật thể trong cell hay không? Và tổng thứ 2 là mất mát của phân phối xác suất nếu có vật thể trong cell.

Ngoài ra để điều chỉnh phạt loss function trong trường hợp dự đoán sai bounding box ta thông qua hệ số điều chỉnh ⋋coord và ta muốn giảm nhẹ hàm loss function trong trường hợp cell không chứa vật thể bằng hệ số điều chỉnh ⋋noobj.

### 2.1.6. Dự báo bounding box

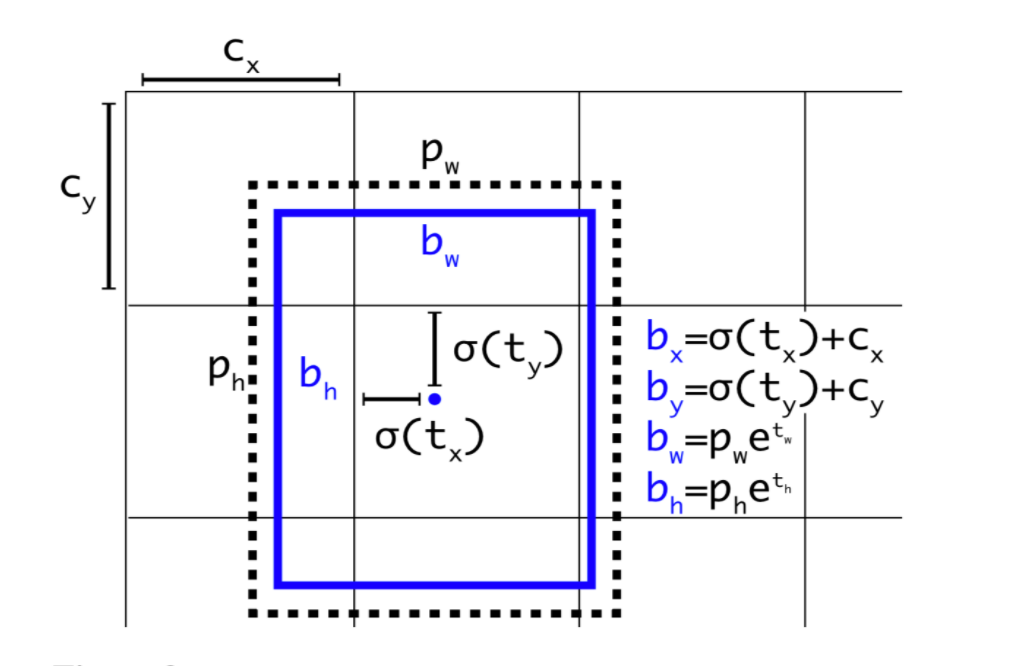
Để dự báo bounding box cho một vật thể chúng ta dựa trên một phép biến đổi từ anchor box và cell.

YOLO dự đoán bounding box sao cho nó sẽ không lệch khỏi vị trí trung tâm quá nhiều. Nếu bounding box dự đoán có thể đặt vào bất kỳ phần nào của hình ảnh, như trong mạng regional proposal network, việc huấn luyện mô hình có thể trở nên không ổn định.

Cho một anchor box có kích thước (*pw*, *ph*) tại cell nằm trên feature map với góc trên cùng bên trái của nó là (*cx*, *cy*), mô hình dự đoán 4 tham số (*tx*, *ty*, *tw*, *th*) trong đó 2 tham số đầu là độ lệch (offset) so với góc trên cùng bên trái của cell và 2 tham số sau là tỷ lệ so với anchor box. Và các tham số này sẽ giúp xác định bounding box dự đoán b có tâm (*bx*, *by*) và kích thước (*bw*, *bh*) thông qua hàm sigmoid và hàm exponential như các công thức bên dưới:

|  |  |
| --- | --- |
|  | (2.3) |

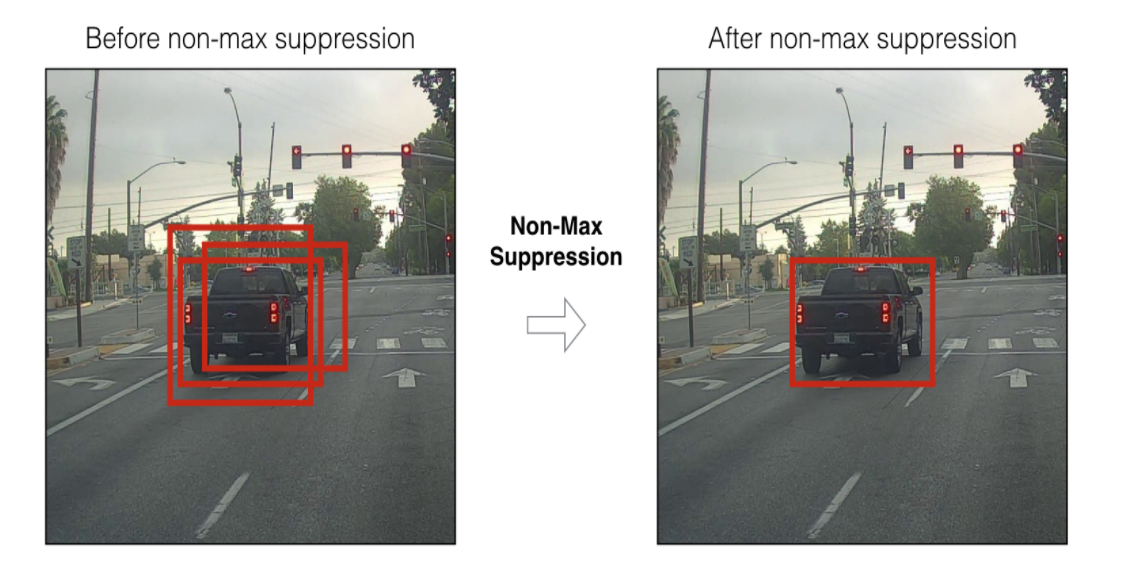
Ngoài ra, do các tọa độ đã được hiệu chỉnh theo width và height của bức ảnh nên luôn có giá trị nằm trong ngưỡng . Do đó khi áp dụng hàm sigmoid giúp ta giới hạn được tọa độ không vượt quá xa các ngưỡng này.



Hình 2.6: Công thức ước lượng bounding box từ anchor box.

### 2.1.7. Non-max suppression

Do thuật toán YOLO dự báo rất nhiều bounding box trên một bức ảnh nên đối với những cell có vị trí gần nhau, khả năng các khung hình bị overlap là rất cao. Trong trường hợp đó YOLO sẽ cần đến non-max suppression - NMS để giảm bớt số lượng các khung hình được sinh ra một cách đáng kể.



Hình 2.7: non-max suppression.

Các bước thực hiện của NMS:

* Bước 1: Đầu tiên chúng ta sẽ tìm cách giảm bớt số lượng các bounding box bằng cách lọc bỏ toàn bộ những bounding box có xác suất chứa vật thể nhỏ hơn một ngưỡng threshold nào đó, thường là 0.5.
* Bước 2: Đối với các bouding box giao nhau, non-max suppression sẽ lựa chọn ra một bounding box có xác suất chứa vật thể là lớn nhất. Sau đó tính toán chỉ số giao thoa IoU với các bounding box còn lại.

Nếu chỉ số này lớn hơn ngưỡng threshold thì điều đó chứng tỏ 2 bounding boxes đang overlap nhau rất cao. Ta sẽ xóa các bounding có xác suất thấp hơn và giữ lại bounding box có xác suất cao nhất. Cuối cùng, ta thu được một bounding box duy nhất cho một vật thể.

## 2.2. SORT (Simple Online Realtime Tracking)

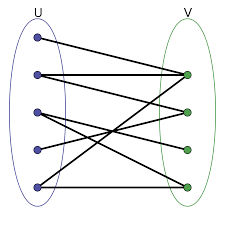
Simple Online RealtimeTracking – SORT, được giới thiệu lần đầu năm 2016, chỉnh sửa bổ sung phiên bản thứ 2 vào năm 2017, đề xuất giải pháp cho MOT. Sort là sự phát triển của khung theo dõi nhiều đối tượng trực quan dựa trên các kỹ thuật ước lượng trạng thái và liên kết dữ liệu thô. Đây là một thuật toán thuộc dạng theo dõi và phát hiện Tracking by Detection, bằng việc sử dụng hai thuật toán cốt lõi của SORT là Kalman Filter và giải thuật Hungary, SORT được thiết kế cho các ứng dụng theo dõi thời gian thực và phương pháp này tạo ra nhận dạng đối tượng một cách nhanh chóng.

### 2.2.1. Giải thuật Hungary (Hungary Algorithm).

Giải thuật Hungary (Hungary Algorithm) [11] được phát triển và công bố vào năm 1955, đề xuất giải bài toán phân công công việc cho nhiều đối tượng (Asignment Problem [12]).

**Ứng dụng giải thuật Hungary trong bài toán MOT**:

Có n detection và n track predicted . Để liên kết một detection i với một track j giả sử dựa vào 1 độ đo D - D là khoảng cách giữa i và j trong không gian vector. Bài toán đặt ra là cần liên kết mỗi detection với mỗi track tương ứng sao cho sai số của việc liên kết là nhỏ nhất.



Hình 2.8: Mô tả sơ bộ thuật toán Hungary.

Trước tiên, chúng ta mô hình hóa lại bài toán để giảm độ phức tạp khi xử lí

|  |  |
| --- | --- |
|  | (2.4) |

Các số  thỏa mãn các điều kiện trên gọi là một phương án phân công, hay ngắn gọn là một phương án, một phương án đạt cực tiểu của z được gọi là một phương án tối ưu hay lời giải của bài toán.

Cụ thể hơn, có thể chia thuật toán thành các bước sau:

* Bước 1 (Bước chuẩn bị). Trừ các phần tử trên mỗi hàng của C cho phần tử nhỏ nhất trên hàng đó, tiếp theo trừ các phần tử trên mỗi cột cho phần tử nhỏ nhất trên cột đó. Kết quả ta nhận được ma trận C' có tính chất: trên mỗi hàng, cột có ít nhất một phần tử 0 và bài toán giao việc với ma trận C' có cùng lời giải như bài toán với ma trận C.
* Bước 2: Vẽ một số tối thiểu các đường thẳng trên dòng và cột để đảm bảo mọi phần tử 0 đều được đi qua.
* Bước 3: Nếu có n đường thẳng được vẽ, kết thúc thuật toán và tiến hành phân công công việc. Nếu số đường thẳng được vẽ nhỏ hơn n, vẫn chưa tìm được phương án phân công tối ưu, tiến hành bước tiếp theo.
* Bước 4: Mỗi hàng (hoặc cột) có đường thẳng vẽ qua, ta gọi các hàng (cột) đó là các hàng (cột) thiết yếu. Các hàng (cột) còn lại là các hàng (cột) không thiết yếu. Tìm phần tử nhỏ nhất không nằm trong các hàng (cột) thiết yếu, tiến hành trừ mỗi hàng không thiết yếu cho phần từ nhỏ nhất ấy và cộng giá trị nhỏ nhất ấy cho cột thiết yếu. Ta được ma trận C’’ có cùng lời giải với ma trận C’. Sau đó quay lại Bước 2.

Khi khởi tạo một trình bám đối tượng mới, giá trị hiệp phương sai của trình bám đối tượng nên để các giá trị lớn để thể hiện sự không chắc chắn của mô hình. Bên cạnh đó, các giá trị vận tốc ban đầu được cũng được khởi tạo bằng 0 do tác giả chú trọng vào vấn đề theo dõi frame-to-frame. Ngoài ra, giả định mô hình vận tốc không đổi cũng là 1 giả định kém trong thực tế, vậy nên nếu để ngưỡng ​quá cao thậm chí có thể làm giảm độ chính xác theo dõi.

### 2.2.2. Bộ lọc Kalman (Kalman Filter).

Bộ lọc Kalman (Kalman Filter) [13] là một mô hình Linear-Gaussian State Space Model, được giới thiệu lần đầu năm 1960 và ứng dụng trong rất nhiều lĩnh vực khác nhau: Xe tự lái, thực tế ảo, kinh tế lượng, bám mục tiêu, điều khiển tối ưu...

Trong OT, bộ lọc Kalman được biết đến nhiều nhất với vai trò dự đoán các trạng thái của đối tượng hiện tại dựa vào các track trong quá khứ và update lại các detection sau khi đã được liên kết với các track trước đó.

Quá trình cần xử lí là 1 quá trình ngẫu nhiên với các mô hình đã được định nghĩa từ trước:

|  |  |
| --- | --- |
| ​ | (2.5) |

ở đây:

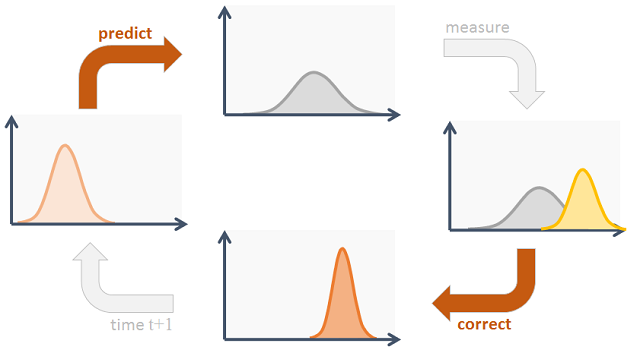
* là giá trị biến trạng thái của quá trình, thường là các giá trị ẩn, không thể quan sát được.
* là giá trị đo được, quan sát được của quá trình.
* là các mô hình định nghĩa từ trước.
* ​ lần lượt là nhiễu của quá trình và nhiễu trong lúc đo đạc.

Linear Kalman Filter giả định các mô hình của quá trình đều là các mô hình tuyến tính. Khi đó:

|  |  |
| --- | --- |
|  | (2.6) |

với:

* các ma trận vuông cỡ n (state transistion matrix) và ma trận cỡ *m*x*n* (measurement function)



Hình 2.9: Hình dung bước xử lý của Kalman filter.

Các bước xử lí tiếp theo của Kalman Filter có thể chia làm 2 phần chính bằng cách tiếp cận dựa trên xác suất:

* Bước dự đoán (Prediction):

Để dự đoán các giá trị trạng thái của quá trình ngẫu nhiên, ta dự đoán các giá trị mean và covariance . Theo tính chất kỳ vọng và ma trận hiệp phương sai với vector ngẫu nhiên, ta có:

* Bước hiệu chỉnh (Update):

Quá trình hiệu chỉnh phức tạp hơn 1 chút, ta có:

Áp dụng [định lí Bayes cho Linear Gaussian System](http://web4.cs.ucl.ac.uk/staff/C.Bracegirdle/bayesTheoremForGaussians.pdf), [đồng nhất thức ma trận Woodbury](https://en.wikipedia.org/wiki/Woodbury_matrix_identity), ta có :

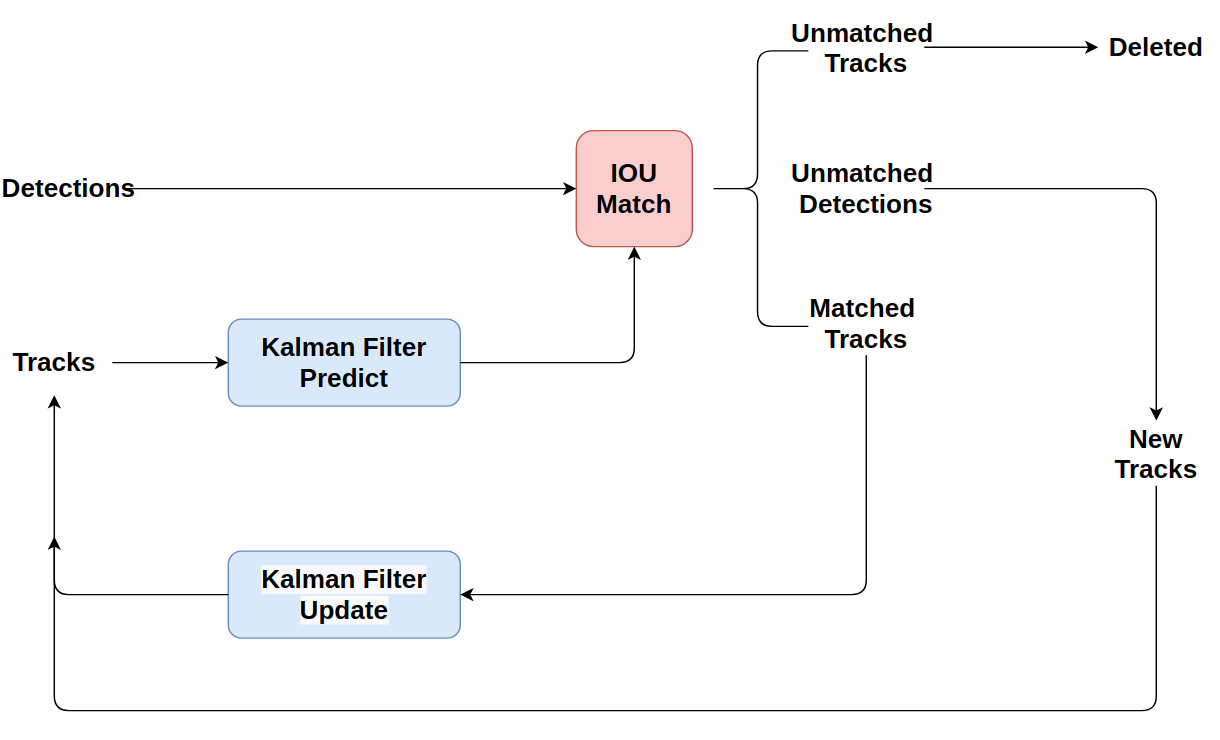
|  |  |
| --- | --- |
| )  ​ | (2.7) |

### 2.2.3: Cách hoạt động của SORT

SORT tập trung vào vấn đề liên kết giữa các nhận diện và bám đối tượng sau khi đã nhận điện được từ khung hình, do đó, phần OD có thể là bất cứ mô hình nào hiện nay như: **YOLO**, **SSD**,các mô hình họ**RCNN** như RCNN, Fast RCNN, Faster RCNN... Trong bài báo công bố chính thức, tác giả sử dụng **Faster Region CNN** (Faster RCNN) với backbone là **VGG16 [15].**

Một đặc điểm của lớp các thuật toán Tracking by Detection là tách đối tượng cần xác định ra như một bài toán riêng biệt và cố gắng tối ưu kết quả trong bài toán này. Công việc sau đó là tìm cách liên kết các hộp giới hạn thu được ở mỗi khung và gán ID cho từng đối tượng. Do đó, thu được một khung quá trình xử lí như sau:

* + Detect: phát hiện vị trí các đối tượng trong frame
  + Predict: Dự đoán vị trí mới của các đối tượng dựa vào các frame trước đó.
  + Associate: Liên kết các vị trí detected với các vị trí dự đoán được để gán ID tương ứng.



Hình 2. 10: Luồng xử lý của SORT.

Từ sơ đồ hoạt động trên có thể thấy SORT trải qua luồng xử lý 5 bước chính như sau:

* Bước 1: SORT tiến hành sử dụng Kalman Filter để dự đoán các trạng thái track mới dựa trên các track trong quá khứ.

Để ứng dụng được Kalman Filter, việc xác định được các dạng biến cũng như mô hình ban đầu của quá trình là điều bắt buôc cần có. Với giả định các đối tượng chuyển động đều, và độc lập với các đối tượng khác, một track được xác định bằng

với:

* *x* có ma trận hiệp phương sai ban đầu được khởi tạo với giá trị lớn để thẻ hiện sử không chắc chắn của trạng thái.
* *u*, *v* lần lượt là tọa độ của tâm đối tượng (ở đây là tâm bounding box).
* *s* là diện tích của bounding box.
* *r* là ti lệ aspect ratio của bounding box
* lần lượt là các giá trị vận tốc tương ứng của x, y, s.

do giả định các đối tượng chuyển động đều, ta có

Khi đó, phương trình

|  |  |
| --- | --- |
|  | (2.8) |

tương đương với được khởi tạo tuân theo phân phối chuẩn có mean = 0 và covariance không đổi.

và phương trình

|  |  |
| --- | --- |
|  | (2.9) |

tương đương với được khởi tạo tuân theo phân phối chuẩn có mean = 0 và covariance không đổi.

* Bước 2: Sử dụng những track vừa dự đoán được, kết hợp với các phát hiện thu được, xây dựng ma trận chi phí cho Assignment Problem. Chi phí được sử dụng để đánh giá ở đây là giá trị IOU giữa các bouding box của track và detection.
* Bước 3: Sử dụng giải thuật Hungary giải bài toán Assignment Problem với ma trận chi phí vừa lập.
* Bước 4: Xử lí, phân loại các phát hiện
* Bước 5: Sử dụng bộ lọc Kalman để update những detection đã được liên kết với track.

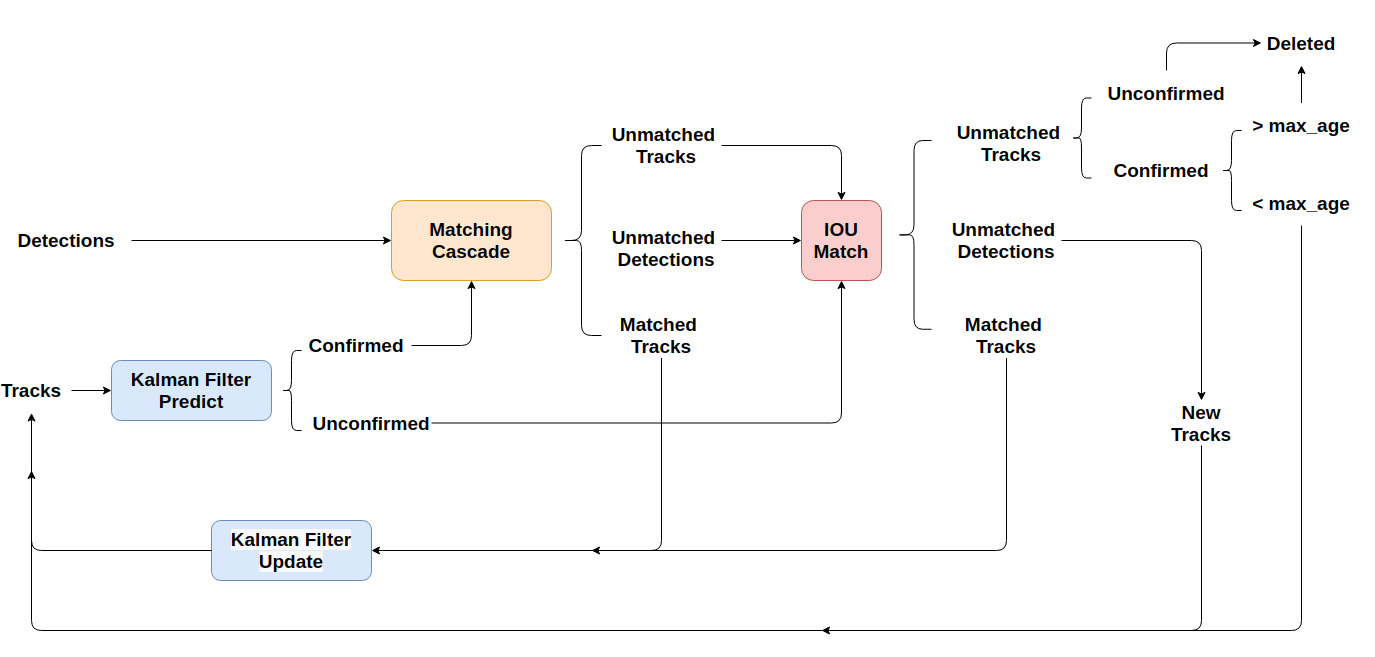
Một số lưu ý của SORT:

* Khi khởi tạo một track mới, giá trị hiệp phương sai của track nên để các giá trị lớn để thể hiện sự không chắc chắn của mô hình. Bên cạnh đó, các giá trị vận tốc ban đầu được cũng được khởi tạo bằng 0
* Đặt ngưỡng ​ để lọc đi những track không được theo dõi nữa sau ​​ khung hình . Trong bài báo được công bố, ngưỡng ​được đặt bằng 1, do tắc giả chú trọng vào vấn đề theo dõi từng khung hình. Ngoài ra, giả định mô hình vận tốc không đổi cũng là 1 giả định kém trong thực tế, vậy nên nếu để ​​ quá cao thậm chí có thể làm giảm độ chính xác theo dõi

## 2.3. DEEPSORT (Simple Online and Realtime Tracking with a Deep Association Metric)

DeepSORT được Nicolai Wojke và Alex Bewley phát triển ngay sau SORT nhằm giải quyết các vấn đề thiếu sót liên quan đến số lượng ID\_switches cao. Hướng giải quyết mà DeepSORT đề xuất dựa trên việc sử dụng Deep Learning để trích xuất các đặc trưng của đối tượng nhằm tăng độ chính xác trong quá trình liên kết dữ liệu. Ngoài ra, một chiến lược liên kết cũng được xây dựng mang tên Matching Cascade giúp việc liên kết các đối tượng sau khi đã biến mất 1 thời gian được hiệu quả hơn.

Do SORT đã mô hình hóa bài toán khá ổn, nên DeepSORT và các thành phần trong trạng thái track không thay đổi quá nhiều. Các mô hình trạng thái và mô hình quan sát của DeepSORT hầu như giống với SORT.

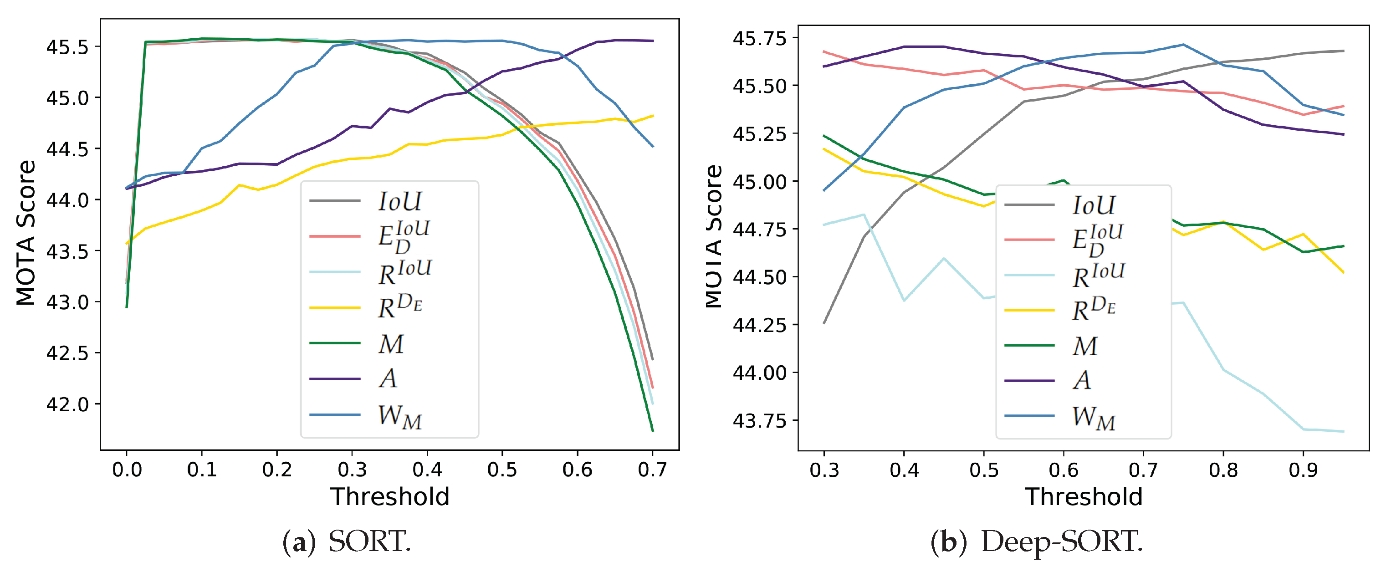


Hình 2. 11: Luồng xử lý của DeepSORT.

Luồng xử lí của DeepSORT được thực hiện tuần tự qua các bước dưới đây:

* Bước 1: Sử dụng Faster-RCNN hoặc các thuật toán OD khác để phát hiện các đối tượng trong khung hình hiện tại.
* Bước 2: DeepSORT sử dụng Kalman Filter để dự đoán các trạng thái track mới dựa trên các track trong quá khứ. Các trạng thái này lúc mới khởi tạo sẽ được gán 1 giá trị mang tính thăm dò (tentative). Giá trị này nếu vẫn đảm bảo duy trì được trong 3 frame tiếp theo, trạng thái sẽ chuyển từ thăm dò sang xác nhận và sẽ cố gắng được duy trì theo dõi trong 30 frame tiếp theo. Ngược lại, nếu mất dấu khi chưa đủ 3 frame, trạng thái sẽ bị xóa khỏi trình theo dõi.
* Bước 3: Sử dụng những track đã được xác nhận, tiến hành đưa vào chiến lược đối sánh phân tầng (matching cascade) nhằm liên kết với các detection phát hiện được dựa trên độ đo về khoảng cách và đặc trưng.
* Bước 4: Các track và các detection chưa được liên kết sẽ được đưa đến 1 lớp lọc tiếp theo. Sử dụng giải thuật Hungary giải bài toán phân công với ma trận chi phí IOU để liên kết lần 2
* Bước 5: Xử lí, phân loại các detection và các track
* Bước 6: Sử dụng Kalman filter để hiệu chỉnh lại giá trị của track từ những detection đã được liên kết với track và khởi tạo các track mới.

Với chiến lược liên kết cũng như sử dụng những độ đo phù hợp, DeepSORT đã cải thiện được vấn đề của SORT. Lượng ID\_switches giảm từ 1423 xuống còn 781, tức giảm 45% đồng thời giảm các lỗi liên quan đến đối tượng bị che khuất hoặc biến mất 1 thời gian. Dù tốc độ xử lỉ có giảm nhẹ, DeepSORT vẫn đảm bảo tốc độ xấp xỉ thời gian thực (realtime) nếu sử dụng GPU.



Hình 2. 12: So sánh các thông số của SORT và DeepSORT.

### 2.3.1. Các độ đo của DeepSORT

Deep SORT đề xuất 2 độ đo hoàn toàn mới, được thêm vào để cải thiện cho quá trình liên kết dữ liệu.

* Khoảng cách Mahalanobis

|  |  |
| --- | --- |
|  | (2.10) |

Trong đó  là giá trị kì vọng và ma trận hiệp phương sai của biến ngẫu nhiên track thứ i, và  là giá trị của detection thứ j.

Khoảng cách Mahalanobis là thước đo khoảng cách giữa điểm P và phân phối D, do P. C. Mahalanobis đưa ra vào năm 1936. Nó là sự tổng quát hóa đa chiều về ý tưởng đo lường độ lệch chuẩn giữa P so ới giá trị kì vọng của D. Khoảng cách này bằng 0 nếu P ở giá trị kì vọng của D và tăng lên khi P di chuyển ra xa giá trị kì vọng này.

Ngoài việc đo lường khoảng cách giữa track và detection, khoảng cách Mahalanobis còn được dùng để loại trừ các liên kết không chắc chắn bằng cách lập ngưỡng khoảng cách Mahalanobis ở khoảng tin cậy 95%.

với

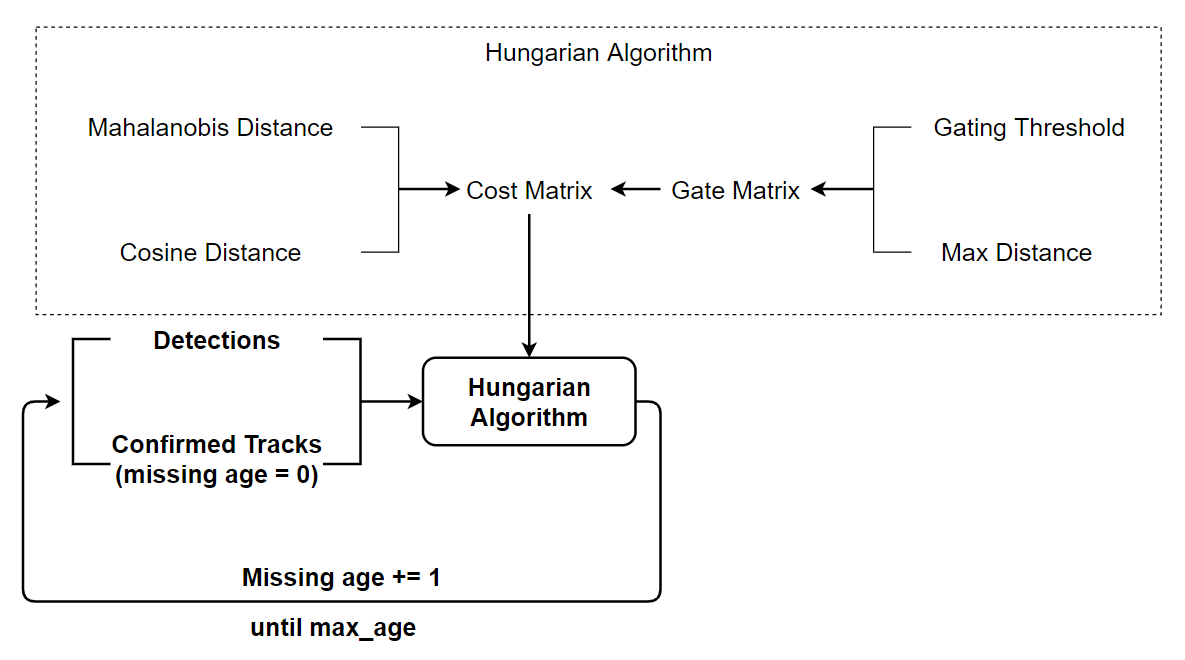
* **Khoảng cách cosine**

DeepSORT đồng thời sử dụng một độ đo khác về đặc trưng của đối tượng, nhằm đảm bảo việc liên kết chuẩn xác dù đối tượng đã biến mất và sau đó xuất hiện trở lại trong khung hình. Đó chính là các đặc trưng học được từ cosine metric learning đã được trình này ở trên. Với mỗi detection, đặc trưng ​ được trích xuất với . Với mỗi track, một danh sách với độ dài khoảng 100 được sử dụng để lưu trữ đặc trưng của 100 track gần nhất. Khi đó, độ đo mới giữa track và detection được tính bằng khoảng cách cosine:

|  |  |
| --- | --- |
|  | (2.11) |

và

với tuỳ vào tập dữ liệu sử dụng.



Hình 2.13: Độ đo khoảng cách của DeepSORT.

Tóm lại, khoảng cách Mahalanobis cung cấp các thông tin về vị trí đối tượng dựa trên chuyển động tức thời, đặc biệt hữu ích cho các dự đoán ngắn hạn. Mặt khác, khoảng cách cosine xem xét thông tin về đặc trưng của đối tượng, đặc biệt hữu ích cho các dự đoán dài hạn hoặc các đối tượng khó phân biệt. Bằng việc kết hợp 2 độ đo với trọng số phù hợp, DeepSORT tạo ra một độ đo mới:

|  |  |
| --- | --- |
|  | (2.12) |

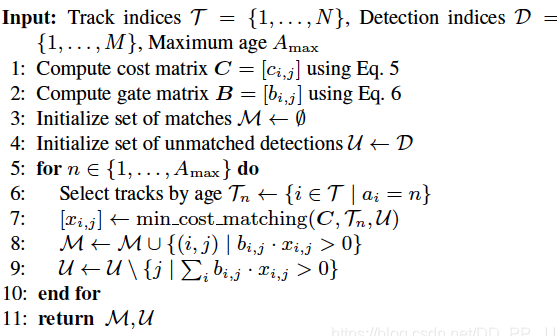
với

Giá trị  sau đó được sử dụng trong chiến lược liên kết matching cascade. Giá trị  có thể được điều chỉnh tùy theo trường hợp sử dụng để cân bằng giữa 2 độ đo. Trong bài báo, tác giả để , tức là không sử dụng đến khoảng cách Mahalanobis. Dù vậy, khoảng cách Mahalanobis vẫn hữu ích, do có thể lọc bỏ những detection có độ liên kết không đảm bảo ngưỡng.

### 2.3.2: **Chiến lược đối sánh theo tầng và quản lý vòng đời 1 track**

Chiến lược đối sánh theo tầng (**Matching Cascade)** được DeepSORT sử dụng nhằm cải thiện độ chính xác của liên kết, chủ yếu là vì khi đối tượng biến mất trong thời gian dài, độ không chắc chắn của bộ lọc Kalman sẽ tăng lên rất nhiều và sẽ dẫn đến phân tán xác suất dự đoán liên tục. Vì vậy, nếu dự đoán liên tục không được cập nhật, phương sai của phân phối chuẩn sẽ ngày càng lớn. Khi đó, giá trị của khoảng cách Mahalanobis giữa các điểm xa giá trị kì vọng và các điểm gần giá trị kì vọng là như nhau. Chiến lược đối sánh theo tầng tiến hành lấy lần lượt từng track ở các frame trước đó, để tiến hành xây dựng ma trận chi phí và giải bài toán phân công theo từng tầng.

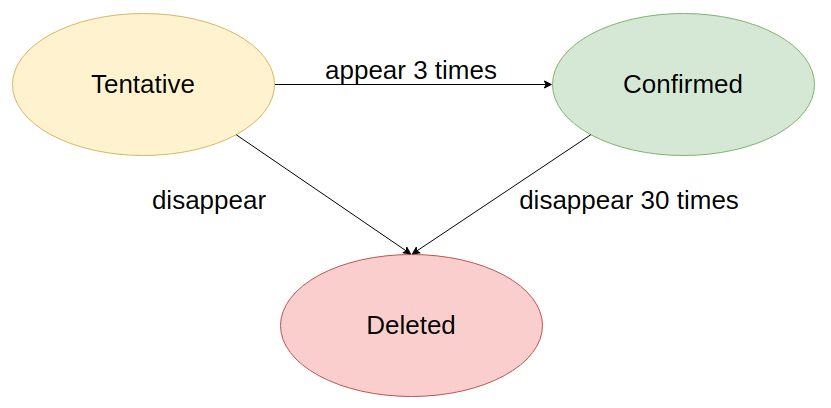
Thuật toán **Matching Cascade** được trình bày cụ thể hơn ở mã giả dưới đây:



Hình 2.14: Quá trình xây dựng chiến lược đối sánh theo tầng.

Trong khi đó, Deep SORT quản lí vòng đợi của 1 track dựa trên 1 biến trạng thái với 3 giá trị (tentative, confirmed, deleted). Cụ thể hơn:

* Các trạng thái này lúc mới khởi tạo sẽ được gán 1 giá trị mang tính thăm dò (tentative).
* Giá trị này nếu vẫn đảm bảo duy trì được trong 3 frame tiếp theo, trạng thái sẽ chuyển từ thăm dò sang xác nhận.
* Các track có trạng thái xác nhận sẽ cố gắng được duy trì theo dõi, dù bị biến mất thì DeepSORT vẫn sẽ duy trì theo dõi trong 30 frame tiếp theo.
* Ngược lại, nếu mất dấu khi chưa đủ 3 frame, trạng thái sẽ bị xóa khỏi trình theo dõi.



Hình 2.15: Quản lý vòng đời của 1 track.

### 2.3.3: Các thông số đánh giá.

Do được xây dựng và phát triển từ SORT nên các thành phần trong trạng thái track không thay đổi quá nhiều. Mỗi track bao gồm 8 thành phần:

Với (u, v) là tọa độ tâm của đối tượng (ở đây là tâm của bounding box), *γ* là tỉ lệ khung hình, h là chiều cao của bounding box, và các vận tốc tương ứng . Các mô hình trạng thái và mô hình quan sát của DeepSORT hầu như giống với SORT.

Về metric đánh giá cần quan tâm các metric sau:

* FP: tổng số lần xuất hiện một đối tượng được phát hiện mặc dù không có đối tượng nào tồn tại
* FN: tổng số lần mà đối tượng hiện có không được phát hiện.
* ID\_Switches: tổng số lần 1 đối tượng bị gán cho 1 ID mới trong suốt quá trình tracking video
* MOTA: Độ chính xác theo dõi nhiều đối tượng

|  |  |
| --- | --- |
|  | (2.13) |

* MOTP: Độ chính xác theo dõi nhiều đối tượng với độ phân tán thấp

|  |  |
| --- | --- |
|  | (2.14) |

* MT: tính trong 80% video
* ML: tính trong 20% video
* Hz: Tốc độ tracking

Nhìn chung, dựa trên các thí nghiệm đã thực hiện, SORT có thể đạt được kết quả chính xác và độ chính xác cao hơn, trong khi phương pháp DeepSORT thu được các giá trị tốt nhất của FN, ID và MT. Hơn nữa đề xuất một chỉ số liên kết dữ liệu đã đạt được hiệu suất tốt nhất trên cả hai đối tượng được đánh giá các phương pháp theo dõi. Chỉ số liên kết dữ liệu cho thấy sự cải thiện đáng kể về chỉ số đánh giá MT, có thể rất quan trọng đối với các nhiệm vụ điều hướng thành công trên nền tảng. Kết quả cho thấy, hiệu suất tổng thể của đối tượng theo dõi có phụ thuộc vào hiệu suất của phương pháp theo dõi. SORT nhanh hơn DeepSORT, đạt FPS cao hơn trên tổng thể (YOLO + SORT). Tuy nhiên DeepSORT cho kết quả chính xác hơn. Do đó, việc xem xét nhiệm vụ trong các nền tảng hỗ trợ và cũng xem xét các vấn đề liên quan đến thuật toán phát hiện đối tượng, thuật toán DeepSORT thu được kết quả mạnh mẽ hơn. Do đó, có thể là một cách tiếp cận phù hợp hơn đối với bài toán này.

# Chương 3

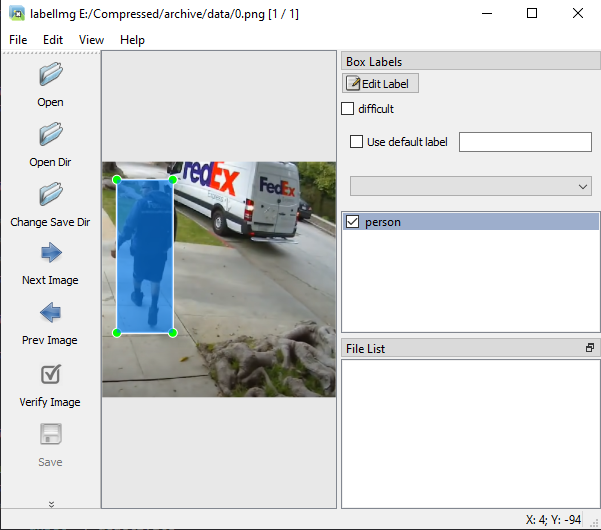
# XÂY DỰNG CHƯƠNG TRÌNH VÀ ĐÁNH GIÁ KẾT QUẢ

Chương 3 tập trung xây dựng chương trình theo dõi đa đối tượng sử dụng thuật toán Yolo và DeepSort**.** Tiến hành phân tích và đánh giá mô hình trên các bộ dữ liệu về theo dõi đa đối tượng.

## 3.1. Bộ dữ liệu huấn luyện YOLO.

Mô tả dữ liệu: Bộ dữ liệu cho quá trình huấn luyện được lấy từ camera đời sống của con người dưới dạng các file ảnh. Kết hợp với các bộ dữ liệu được công khai trên các nguồn như Kaggle, COCO, MOT...

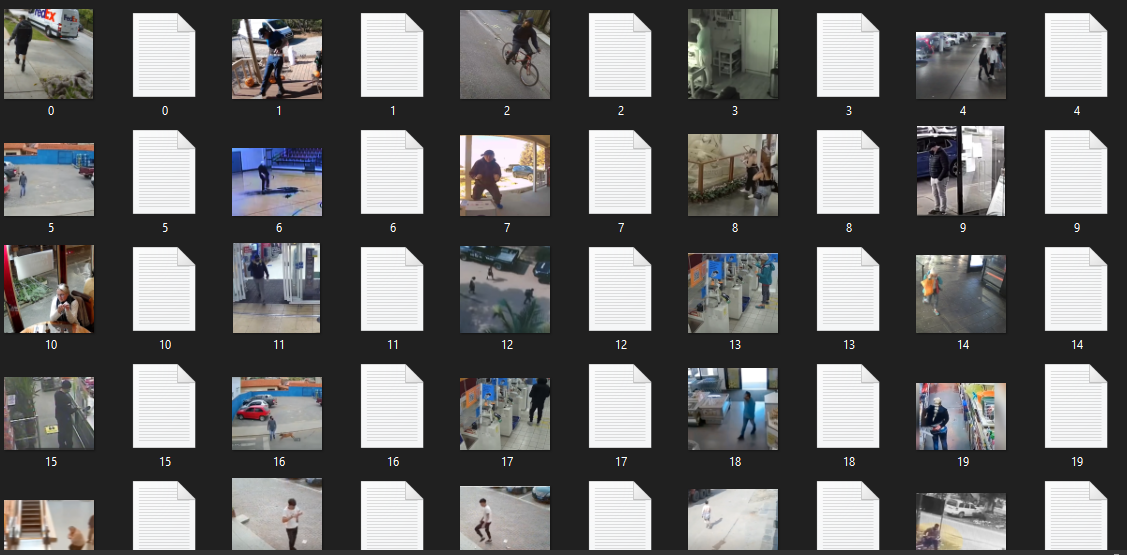
Tập nhãn là các file chứa thông tin về lớp, vị trí khung các đối tượng có xuất hiện trong ảnh. Có cùng tên với file ảnh. Được sinh ra từ quá trình gán nhãn dữ liệu thủ công nhờ công cụ LabelImg.



Hình 3.2: Quá trình dán nhãn dữ liệu huấn luyện.

Sau khi gán nhãn xong sẽ có một thư mục chứa dữ liệu ảnh. Mỗi ảnh sẽ có 2 file:

* File có đuôi jpg, png… là file ảnh đầu vào
* File có đuôi txt: là file nhãn của ảnh tương ứng
* Hai file này có tên y hệt nhau, khác mỗi phần đuôi định dạng



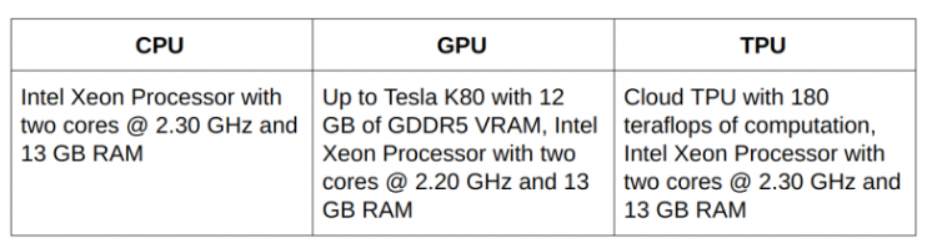
Hình 3.3: Thư mục chứa dữ liệu ảnh đã được gán nhãn

## 3.2. Môi trường và ngôn ngữ lập trình.

### 3.2.1: Huấn luyện mô hình YOLO.

Quá trình huấn luyện mô hình sẽ được huấn luyện trên COLAB. Bởi vì COLAB train khỏe, nhanh có GPU khủng và miễn phí. Với AI, Deeplearning, Google không chỉ có những sản phẩm dự án đã và đang phát triển, Google còn có công cụ giúp cho các developer phát triển, train model một cách dễ dàng nhất.

Colab cung cấp nhiều loại GPU, thường là Nvidia K80s, T4s, P4s và P100s, tuy nhiên người dùng không thể chọn loại GPU trong Colab, GPU trong Colab thay đổi theo thời gian. Vì là dịch vụ miễn phí, nên Colab sẽ có thứ tự ưu tiên trong việc sử dụng tài nguyên hệ thống, cũng như giới hạn thời gian sử dụng, thời gian sử dụng tối đa lên tới 12 giờ.



Hình 3.4: Cấu hình phần cứng Google Colab cung cấp

### 3.2.2: Chương trình thử nghiệm.

Chương trình thử nghiệm được cài đặt trên môi trường Windows 10. Sử dụng ngôn ngữ lập trình: Python 3.8.5. Chương trình có sử dụng các thư viện quan trọng: Numpy, Opencv, Tensorflow, Scikit-learn, Scipy…

## 3.3. Cấu trúc chương trình.

### 3.3.1: Quá trình huấn luyện mô hình YOLO.

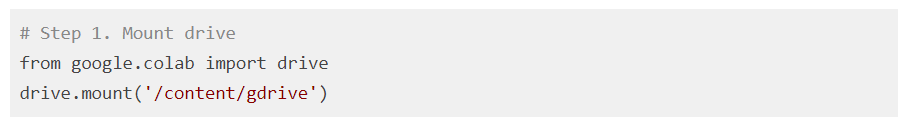
#### 3.3.1.1: Chuẩn bị dữ liệu.

Đầu tiên nén mục data chứa tập ảnh và nhãn lại thành file data.zip. Sau đó đưa lên Google Drive. Tiếp tục, cần phải chỉnh sửa file yolo4-tiny-custom.cfg như bên dưới:

* Đầu tiên cần xác định số bao nhiêu loại dối tượng cần theo dõi. Với bài toán này tập chung vào việc theo dõi 1 đối tượng là con người trong video nên nên chỉ có 1 class person.
* Thay đổi max\_batches = max (<số class>\*2000,6000). Nghĩa là nếu <số class>\*2 mà nhỏ hơn 6000 thì lấy 6000 và ngược lại.
* Thay đổi steps = 80%, 90% của max\_batches.
* Thay đổi lại toàn bộ các dòng có “class = 80” thành “class=<số class>”. Ở chương trinh hiện tại chỉ theo dõi 1 lớp người nên sẽ hay đổi thành 1.
* Thay đổi lại các dòng có “filters=255” thành “filters = (số class + 5) \* 3”. Ở chương trình này sẽ là 18.
* Thay đổi lại width = 416, height = 416. Nếu đã đúng không cần sửa lại.

#### 3.3.1.2: Huấn luyện mô hình.

Như đã nói ở trên, sau 10 tiếng Colab sẽ kill sạch, xoá sạch chương trình nên nếu như để lại tài liệu trên đó sẽ bị xoá hết. Tạo một code block mới sau đó viết đoạn code bên dưới để kết nối với Google Drive.



Hình 3.5: Kết nối với google driver

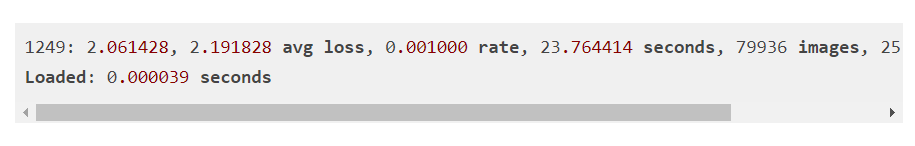
Sau khi đã kết nối tới Drive để lấy dữ liệu và lưu lại mô hình sau khi train xong ta sẽ đi tới quá trình huấn luyện mô hình.

Trong quá trình train thì trong file chứa model của các bạn sẽ sinh ra các file sau:

* File yolo-tiny-custom\_last.weights là file weights cuối cùng có được.
* File yolo-tiny-custom\_1000.weights yolo-tiny-custom\_2000.weight… là các file weight tại 1000 vòng, 2000 vòng…

Tùy tình hình dữ liệu, bài toán, cần dùng weights nào thì lấy file đó.

Trong quá trình train các sẽ thấy 1 số dòng có dạng như này:



Hình 3.6: Kết quả train model ở vòng 1249.

* 1249: Số vòng train đến hiện tại, ở đây là 1249.
* 2.061428, 2.191828 avg loss: cái này nghĩa là loss của vòng hiện tại là 2.061428 còn loss trung bình cho đến vòng hiện tại là 2.191828
* Chương trình sẽ tự dừng khi train xong max\_batches được định nghĩa bên trên. Ở đây sẽ là 6000 vòng nếu train lấy file weights cuối cùng.
* Khi quan sát một thời gian thấy loss bão hòa, không giảm quá nhiều sau một khoảng tầm 100 vòng có thể dừng quá trình huấn luyện và lấy mô hình ra kiểm tra và thực hiện đánh độ chính xác.

### 3.3.2: Xây dựng chương trình bám đối tượng.

Đầu tiên cần thêm các modul vào môi trường (enviroment) Python. Có thể sử dụng những công cụ như Pycharm, Visual Code Studio, hoặc Jupyter Notebook (tệp tin conda.yaml đi kèm trong thư mục này chứa thông tin module để cài các thư viện cần thiết cho việc triển khai code trên các máy khác).

Tiếp theo sau khi train được file weights của mô hình YOLO đưa file này vào cùng thư mục chứa code của thuật toán DeepSORT và bắt đầu quá trình xây dựng chương trình.

Trước tiên là quá trình xây dựng model Tensorflow từ weights của Yolo. Điều này được thực hiện trong file save\_model.py như sau:

|  |
| --- |
| *import* tensorflow *as* tf  *from* absl *import* app, flags, logging  *from* absl.flags *import* FLAGS  *from* core.yolov4 *import* YOLO, decode, filter\_boxes  *import* core.utils *as* utils  *from* core.config *import* cfg  flags.DEFINE\_string('weights', './data/yolov4-tiny.weights', 'path to weights file')  flags.DEFINE\_string('output', '.  /yolov4-416-tiny', 'path to output')  flags.DEFINE\_boolean('tiny', True, 'is yolo-tiny or not')  flags.DEFINE\_integer('input\_size', 416, 'define input size of export model')  flags.DEFINE\_float('score\_thres', 0.2, 'define score threshold')  flags.DEFINE\_string('framework', 'tf', 'define what framework do you want to convert (tf, trt, tflite)')  flags.DEFINE\_string('model', 'yolov4', 'yolov3 or yolov4')  *def* save\_tf():    STRIDES, ANCHORS, NUM\_CLASS, XYSCALE = utils.load\_config(FLAGS)    input\_layer = tf.keras.layers.Input([FLAGS.input\_size, FLAGS.input\_size, 3])    feature\_maps = YOLO(input\_layer, NUM\_CLASS, FLAGS.model, FLAGS.tiny)    bbox\_tensors = []    prob\_tensors = []  *for* i, fm in enumerate(feature\_maps):  *if* i == 0:        output\_tensors = decode(fm, FLAGS.input\_size // 16, NUM\_CLASS, STRIDES, ANCHORS, i, XYSCALE, FLAGS.framework)  *else*:        output\_tensors = decode(fm, FLAGS.input\_size // 32, NUM\_CLASS, STRIDES, ANCHORS, i, XYSCALE, FLAGS.framework)      bbox\_tensors.append(output\_tensors[0])      prob\_tensors.append(output\_tensors[1])    pred\_bbox = tf.concat(bbox\_tensors, axis=1)    pred\_prob = tf.concat(prob\_tensors, axis=1)    boxes, pred\_conf = filter\_boxes(pred\_bbox, pred\_prob, score\_threshold=FLAGS.score\_thres, input\_shape=tf.constant([FLAGS.input\_size, FLAGS.input\_size]))    pred = tf.concat([boxes, pred\_conf], axis=-1)    model = tf.keras.Model(input\_layer, pred)    utils.load\_weights(model, FLAGS.weights, FLAGS.model, FLAGS.tiny)    model.summary()    model.save(FLAGS.output)  *def* main(\_argv):    save\_tf()  *if* \_\_name\_\_ == '\_\_main\_\_':  *try*:          app.run(main)  *except* SystemExit:  *pass* |

Tiếp theo xây dựng chương trình tracking trong file tracker.py như sau:

|  |
| --- |
| *import* os  *# comment out below line to enable tensorflow logging outputs*  os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '3'  *import* time  *import* tensorflow *as* tf  physical\_devices = tf.config.experimental.list\_physical\_devices('GPU')  *if* len(physical\_devices) > 0:      tf.config.experimental.set\_memory\_growth(physical\_devices[0], True)  *from* absl *import* app, flags, logging  *from* absl.flags *import* FLAGS  *import* core.utils *as* utils  *from* core.yolov4 *import* filter\_boxes  *from* tensorflow.python.saved\_model *import* tag\_constants  *from* core.config *import* cfg  *from* PIL *import* Image  *import* cv2  *import* numpy *as* np  *import* matplotlib.pyplot *as* plt  *from* tensorflow.compat.v1 *import* ConfigProto  *from* tensorflow.compat.v1 *import* InteractiveSession  *# deep sort imports*  *from* deep\_sort *import* preprocessing, nn\_matching  *from* deep\_sort.detection *import* Detection  *from* deep\_sort.tracker *import* Tracker  *from* tools *import* generate\_detections *as* gdet  *# Define argument*  flags.DEFINE\_string('weights', './yolov4-416-tiny',                      'path to weights file')  flags.DEFINE\_integer('size', 416, 'resize images to')  flags.DEFINE\_boolean('tiny', True, 'yolo or yolo-tiny')  flags.DEFINE\_string('model', 'yolov4', 'yolov3 or yolov4')  flags.DEFINE\_string('video', './data/video/test.mp4',           'path to input video or set to 0 for webcam')  flags.DEFINE\_string('output', None   , './output/test.avi')  flags.DEFINE\_string('output\_format', 'XVID',           'codec used in VideoWriter when saving video to file')  flags.DEFINE\_float('iou', 0.45, 'iou threshold')  flags.DEFINE\_float('score', 0.50, 'score threshold')  flags.DEFINE\_boolean('dont\_show', False, 'dont show video output')  flags.DEFINE\_boolean('info', True, 'show detailed info of tracked objects')  flags.DEFINE\_boolean('count', True, 'count objects being tracked on screen') |

Load mô hình YOLO và khởi tạo các tham số cần thiết cho DeepSORT:

|  |
| --- |
| *# Definition of the parameters*      max\_cosine\_distance = 0.4      nn\_budget = None      nms\_max\_overlap = 1.0  *# initialize deep sort*      model\_filename = 'model\_data/mars-small128.pb'      encoder = gdet.create\_box\_encoder(model\_filename, batch\_size=1)  *# calculate cosine distance metric*      metric = nn\_matching.NearestNeighborDistanceMetric("cosine",                  max\_cosine\_distance, nn\_budget)  *# initialize tracker*      tracker = Tracker(metric)  *# load configuration for object detector*      config = ConfigProto()     session = InteractiveSession(config=config)      STRIDES, ANCHORS, NUM\_CLASS, XYSCALE = utils.load\_config(FLAGS)      input\_size = FLAGS.size      video\_path = FLAGS.video      saved\_model\_loaded = tf.saved\_model.load(FLAGS.weights,               tags=[tag\_constants.SERVING])      infer = saved\_model\_loaded.signatures['serving\_default'] |

Tiếp tục là load video từ máy, nếu dùng camera thì truyền tham số --video 0 vào lệnh command khi chạy chương trình:

|  |
| --- |
| *# begin video capture*  *try*:          vid = cv2.VideoCapture(int(video\_path))  *except*:          vid = cv2.VideoCapture(video\_path)      out = None |

Sau đó bắt từng bounding boxs được detect ra:

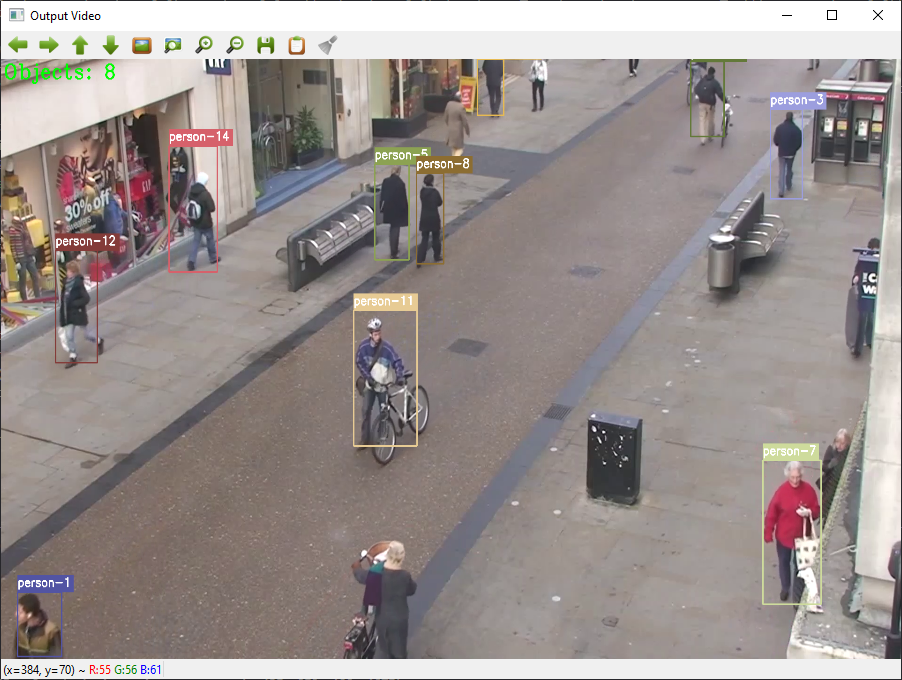
|  |
| --- |
| *while* True:          return\_value, frame = vid.read()  *if* return\_value:              frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)              image = Image.fromarray(frame)  *else*:              print('Video has ended!')  *break*          frame\_num +=1          print('Frame #: ', frame\_num)          frame\_size = frame.shape[:2]          image\_data = cv2.resize(frame, (input\_size, input\_size))          image\_data = image\_data / 255.          image\_data = image\_data[np.newaxis, ...].astype(np.float32)          start\_time = time.time()          batch\_data = tf.constant(image\_data)          pred\_bbox = infer(batch\_data)  *for* key, value in pred\_bbox.items():              boxes = value[:, :, 0:4]              pred\_conf = value[:, :, 4:]          boxes, scores, classes, valid\_detections = tf.image.combined\_non\_max\_suppression(              boxes=tf.reshape(boxes, (tf.shape(boxes)[0], -1, 1, 4)),              scores=tf.reshape(                  pred\_conf, (tf.shape(pred\_conf)[0], -1, tf.shape(pred\_conf)[-1])),              max\_output\_size\_per\_class=50,              max\_total\_size=50,              iou\_threshold=FLAGS.iou,              score\_threshold=FLAGS.score          )  *# convert data to numpy arrays and slice out unused elements*          num\_objects = valid\_detections.numpy()[0]          bboxes = boxes.numpy()[0]          bboxes = bboxes[0:int(num\_objects)]          scores = scores.numpy()[0]          scores = scores[0:int(num\_objects)]          classes = classes.numpy()[0]          classes = classes[0:int(num\_objects)]  *# format bounding boxes from normalized ymin, xmin, ymax, xmax ---> xmin, ymin, width, height*          original\_h, original\_w, \_ = frame.shape          bboxes = utils.format\_boxes(bboxes, original\_h, original\_w)  *# store all predictions in one parameter for simplicity when calling functions*          pred\_bbox = [bboxes, scores, classes, num\_objects]  *# read in all class names from config*          class\_names = utils.read\_class\_names(cfg.YOLO.CLASSES)  *# by default allow all classes in .names file*  *# allowed\_classes = list(class\_names.values())*  *# custom allowed classes (uncomment line below to customize tracker for only people)*          allowed\_classes = ['person']  *# loop through objects and use class index to get class name, allow only classes in allowed\_classes list*          names = []          deleted\_indx = []  *for* i in range(num\_objects):              class\_indx = int(classes[i])              class\_name = class\_names[class\_indx]  *if* class\_name not in allowed\_classes:                  deleted\_indx.append(i)  *else*:                  names.append(class\_name)          names = np.array(names)          count = len(names)  *if* FLAGS.count:              cv2.putText(frame, "Objects: {}".format(count), (5, 35), cv2.FONT\_HERSHEY\_COMPLEX\_SMALL, 2, (0, 255, 0), 2)              print("Objects being tracked: {}".format(count))  *# delete detections that are not in allowed\_classes*          bboxes = np.delete(bboxes, deleted\_indx, axis=0)          scores = np.delete(scores, deleted\_indx, axis=0)  *# encode yolo detections and feed to tracker*          features = encoder(frame, bboxes)          detections = [Detection(bbox, score, class\_name, feature) *for* bbox, score, class\_name, feature in zip(bboxes, scores, names, features)]  *#initialize color map*          cmap = plt.get\_cmap('tab20b')          colors = [cmap(i)[:3] *for* i in np.linspace(0, 1, 20)]  *# run non-maxima supression*          boxs = np.array([d.tlwh *for* d in detections])          scores = np.array([d.confidence *for* d in detections])          classes = np.array([d.class\_name *for* d in detections])          indices = preprocessing.non\_max\_suppression(boxs, classes, nms\_max\_overlap, scores)          detections = [detections[i] *for* i in indices]  *# Call the tracker*          tracker.predict()          tracker.update(detections) |

Ở trên với mỗi boxes sẽ tương ứng với các features sau khi đã nhận dạng được đối tượng, indices là những phần tử được xử lý với NMS để lấy ra được bounding boxs tốt nhất có thể, sau đó hàm tracker.predict() có nhiệm vụ track các objects đã được dự đoán và update tiếp tục tương tự với các đối tượng khác với tracker.update(detections). Thực hiện bám theo IDs theo từng bounding boxs và draw boxs trên từng frames:

|  |
| --- |
| *# update tracks*  *for* track in tracker.tracks:  *if* not track.is\_confirmed() or track.time\_since\_update > 1:  *continue*              bbox = track.to\_tlbr()              class\_name = track.get\_class()  *# draw bbox on screen*              color = colors[int(track.track\_id) % len(colors)]              color = [i \* 255 *for* i in color]              cv2.rectangle(frame, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), color, 2)              cv2.rectangle(frame, (int(bbox[0]), int(bbox[1]-30)), (int(bbox[0])+(len(class\_name)+len(str(track.track\_id)))\*17, int(bbox[1])), color, -1)              cv2.putText(frame, class\_name + "-" + str(track.track\_id),(int(bbox[0]), int(bbox[1]-10)),0, 0.75, (255,255,255),2)  *if* cv2.waitKey(1) & *0x*FF == ord('q'): *break*      cv2.destroyAllWindows() |

### 3.3.3. Một số kết quả đạt được

Kết quả của bài toán được thử nghiệm trên cả 2 mục, bám đối tượng từ video cũng như là từ camera. Cụ thể có kết quả như sau:



Hình 3.7: Kết quả từ thử nghiệm trong video.

Đối với việc sử dụng nguồn video trích xuất trực tiếp từ camera thiết bị thu được kết quả sau đây:

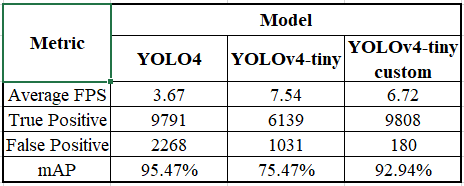


Hình 3.8: Kết quả từ việc từ việc tracking với camera.

### 3.3.4: Đánh giá kết quả

Bằng việc triển khai model Yolov4-tiny custom đã giúp cho việc thuật toán nhẹ nhàng hơn và dễ thực hiện hơn trên các máy cấu hình nhỏ.

Thử nghiệm trên tập dữ liệu 546 ảnh thì kết quả thu được độ chính xác là 93% đối với bộ dữ liệu tự xây dựng này.



Hình 3.9: Bảng so sánh đánh giá các mô hình YOLOv4.

Kết quả cho thấy mô hình cơ sở YOLOv4 đạt mAP cao nhất là 95,47% nhưng FPS trung bình là 3.67 khung hình/giây. YOLOv4-tiny đạt FPS trung bình 7.54 khung hình/giây nhưng chỉ đạt 75,47% mAP, thấp hơn đáng kể khi so sánh với các mô hình. Mô hình Yolov4-tiny custom được đào tạo lại vẫn giữ được thời gian xử lý cao của Yolov4-tiny với tỷ lệ phần trăm chênh lệch chỉ 4,75% đồng thời tăng đáng kể độ chính xác dẫn đến 92,94% mAP. DeepSORT có thể theo dõi tất cả các hộp giới hạn được tạo thành công mà không xảy ra bất kỳ chuyển đổi ID nào bất kỳ lúc nào khi kết hợp với mô hình này.

Dựa trên kết quả của nghiên cứu, Yolov4-tiny custom có thể phát hiện các đối tượng với mAP là 92,94% và FPS trung bình là 6.72. Thuật toán DeepSORT là có thể xác định thành công tất cả các bounding box. Hơn nữa, sử dụng Yolov4-tiny custom với DeepSORT thích hợp cho các ứng dụng như phát hiện người do độ chính xác cao và thời gian xử lý nhanh chóng. Dữ liệu được trích xuất chẳng hạn như tổng số người đếm trên khung hình, số người hiện tại đếm hơn khung và cách xa xã hội qua khung có thể được sử dụng để giám sát hiệu quả các điểm ưa thích để cải thiện hiểu biết về lưu lượng đám đông hoặc lưu lượng người đi bộ có thể được sử dụng để thực hiện các biện pháp tốt hơn và hiệu quả hơn.

Thực nghiệm nhận được kết quả tốt hơn so với một số bài toán nghiên cứu khác nhờ việc triển khai trên model Yolov4 nên đã khắc phục được nhiều nhược điểm còn tồn tại của các version Yolo cũ hơn. Tuy nhiên chưa có nhiều các mô hình được đào tạo cùng phiên bản để so sánh chi tiết hơn nên chưa thể đánh giá cụ thể hiệu quả của mô hình.

# KẾT LUẬN

1. **Kết luận.**

Sau thời gian nghiên cứu cùng với sự giúp đỡ của thầy, cô giáo và sự góp ý của các bạn, em đã hoàn thành đồ án tốt nghiệp với đề tài Nghiên cứu một số phương pháp theo dõi đối tượng chuyển động và ứng dụng vào hệ thống giám sát mục tiêu.

Qua đó nắm bắt được những công việc cần làm để xây dựng một chương trình Nghiên cứu một số phương pháp theo dõi đối tượng chuyển động và ứng dụng vào hệ thống giám sát mục tiêu. Sau đây là một số đánh giá:

**Hạn chế của YOLO.**

YOLO áp đặt các ràng buộc về không gian trên những bounding box, mỗi grid cell chỉ có thể dự đoán rất ít bounding box và duy nhất một lớp. Các ràng buộc này hạn chế khả năng nhận biết số object nằm gần nhau, cũng như đối với các đối tượng có kích thước nhỏ. YOLO sử dụng các feature tương đối thô để dự đoán bounding box, do model sử dụng nhiều lớp downsampling từ ảnh đầu vào. Bởi các hạn chế này của model khi huấn luyện để predict bounding box từ data, dẫn đến YOLO không thực sự tốt trong việc nhận diện các đối tượng với tỉ lệ hình khối mới hoặc bất thường so với tập data. Phiên bản YOLOv4 đã khắc phục phần nào vấn đề này, nhưng vẫn thua kém nhiều so với mô hình Faster RCNN.  
    Ngoài ra, trong quá trình huấn luyện mô hình, loss function không có sự đánh giá riêng biệt giữa error của bounding box kích thước nhỏ so với error của bounding box kích thước lớn. Việc coi chúng như cùng loại và tổng hợp lại làm ảnh hưởng đến độ chính xác toàn cục của mạng. Error nhỏ trên box lớn nhìn chung ít tác hại, nhưng lỗi nhỏ với box rất nhỏ sẽ đặc biệt ảnh hưởng đến giá trị IOU.

**Hạn chế của DeepSort.**

Một số tồn tại chung của bài toán MOT là khi một số đối tượng bị che khuất, hoặc nhiễu lớn, hoặc đối tượng khi ra khỏi video và xuất hiện trở lại, thì việc phát hiện và gán nhãn sẽ bị sai sót. Giả định tuyến tính: SORT đang sử dụng Linear Kalman Filter trong thuật toán cốt lõi, điều này trong thực tế là chưa phù hợp. Để cải thiện vấn đề này, cần quan tâm đến các Kalman Filter phức tạp hơn, như Extended Kalman Filter, Unscented Kalman filter,... Đối với ID\_Switches: Đây là vấn đề lớn nhất của SORT/DeepSORT hiện tại. Bài toán sẽ chính xác cao nếu mật độ đối tượng thưa và di chuyển theo một quỹ đạo đơn giản. Nếu mật độ đối tượng cao sẽ dễ bị gán nhầm ID cho đối tượng đang theo dõi. Do việc liên kết giữa phát hiện và theo dõi chỉ đơn giản dựa trên độ đo IOU (tức là thuật toán chỉ quan tâm đến hình dạng của đối tượng), điều này gây ra hiện tượng số lượng ID\_Switches của 1 đối tượng là vô cùng lớn khi đối tượng bị che khuất, khi quỹ đạo trùng lặp.

**2. Hướng phát triển.**

Để phát triển bài toán tốt hơn cũng như khắc phục được những hạn chế của DeepSORT và Yolov4 hướng nghiên cứu tiếp theo sẽ là việc ứng dụng các mô hình và thuật toán mới hơn vào bài toán này. Với những tồn tại của DeepSORT giải pháp của được đưa ra là một bản nâng cấp mạnh mẽ hơn là StrongSORT [16]. StrongSORT đang nghiên cứu và khắc phục những vấn đề còn tồn tại khó khắc phục của MOT và DeepSORT như đối tượng bị che khuất, ID\_Swiches. Bên cạnh đó mô hình Yolov5 mới được giới thiệu gần đây cho độ chính xác rất cao trong khi yêu cầu tài nguyên hệ thống giảm đi nhiều. Hơn nữa có thể mở rộng bài toán sang những bài toán như kiểm soát tốc độ đối tượng di chuyển, đánh giá mật độ, phân loại phương tiện giao thông, phát hiện sai làn...

# TÀI LIỆU THAM KHẢO

1. Soleimanitaleb, Z., & Keyvanrad, M. A. (2022). *Single Object Tracking: A Survey of Methods, Datasets, and Evaluation Metrics*. arXiv preprint arXiv:2201.13066.
2. Luo, W., Xing, J., Milan, A., Zhang, X., Liu, W., & Kim, T. K. (2021). *Multiple object tracking: A literature review*. Artificial Intelligence, 293, 103448.
3. Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016, September). *Simple online and realtime tracking*. In 2016 IEEE international conference on image processing (ICIP) (pp. 3464-3468). IEEE.
4. Wojke, N., Bewley, A., & Paulus, D. (2017, September). *Simple online and realtime tracking with a deep association metric*. In 2017 IEEE international conference on image processing (ICIP) (pp. 3645-3649). IEEE.
5. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You only look once: Unified, real-time object detection*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).
6. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). *Yolov4: Optimal speed and accuracy of object detection*. arXiv preprint arXiv:2004.10934.
7. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). *SSD: Single shot multibox detector*. In *European conference on computer vision* (pp. 21-37). Springer, Cham.
8. Ren, S., He, K., Girshick, R., & Sun, J. (2015). *Faster r-cnn: Towards real-time object detection with region proposal networks*. Advances in neural information processing systems, 28.
9. Yu, F., Li, W., Li, Q., Liu, Y., Shi, X., & Yan, J. (2016, October). *Poi: Multiple object tracking with high performance detection and appearance feature*. In European Conference on Computer Vision (pp. 36-42). Springer, Cham.
10. Milan, A., Rezatofighi, S. H., Dick, A., Reid, I., & Schindler, K. (2017, February). *Online multi-target tracking using recurrent neural networks*. In Thirty-First AAAI conference on artificial intelligence.
11. Mills-Tettey, G. A., Stentz, A., & Dias, M. B. (2007). *The dynamic hungarian algorithm for the assignment problem with changing costs.*Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-27.
12. Loiola, E. M., de Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., & Querido, T. (2007). *A survey for the quadratic assignment problem*. European journal of operational research, 176(2), 657-690.
13. Welch, G., & Bishop, G. (1995). *An introduction to the Kalman filter*.
14. Mahrishi, M., Morwal, S., Muzaffar, A. W., Bhatia, S., Dadheech, P., & Rahmani, M. K. I. (2021). *Video index point detection and extraction framework using custom YoloV4 Darknet object detection model*. IEEE Access, 9, 143378-143391.
15. Simonyan, K., & Zisserman, A. (2014). *Very deep convolutional networks for large-scale image recognition.*arXiv preprint arXiv:1409.1556.
16. Du, Y., Song, Y., Yang, B., & Zhao, Y. (2022). *StrongSORT: Make DeepSORT Great Again*. arXiv preprint arXiv:2202.13514.