

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN- ĐIỆN TỬ
BỘ MÔN VIỄN THÔNG**



LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC

**ĐỀ TÀI: HỆ THỐNG PHÁT HIỆN TÉ NGÃ
SỬ DỤNG THỊ GIÁC MÁY TÍNH**

GVHD: Ths Nguyễn Khánh Lợi

SVTH: Phan Nguyên Trung - 1814519

Võ Thành An - 1811392

TP.HỒ CHÍ MINH, THÁNG 06 NĂM 2022

Lời cảm ơn

Đầu tiên, chúng em xin bày tỏ lòng biết ơn chân thành sâu sắc tới giáo viên hướng dẫn, thầy ThS Nguyễn Khánh Lợi - Giảng viên bộ môn Viễn Thông đã trực tiếp giúp đỡ, hướng dẫn chúng em hoàn thành luận văn này.

Tuy nhiên, với điều kiện thời gian cũng như vốn kiến thức còn nhiều hạn chế, mặc dù nhóm đã cố gắng hết sức nhưng chắc chắn luận văn khó có thể tránh khỏi những thiếu sót và nhiều chỗ còn chưa chính xác. Nhóm em rất mong nhận được sự chỉ bảo, đóng góp ý kiến của các quý thầy cô để nhóm có điều kiện bổ sung, nâng cao ý thức của mình, phục vụ tốt hơn công tác thực tế sau này.

Xin chân thành cảm ơn.

Tp. Hồ Chí Minh,...tháng 06 năm 2022.

Sinh viên

Lời cam đoan

Chúng tôi tên: Phan Nguyên Trung, Võ Thành An. Là sinh viên chuyên ngành Kỹ thuật Điện tử - Truyền thông, khóa 2018, tại Trường Đại học Bách Khoa - Đại học Quốc gia thành phố Hồ Chí Minh. Chúng tôi xin cam đoan những nội dung sau đây là sự thật: (i) Công trình nghiên cứu này hoàn toàn do chính chúng tôi thực hiện; (ii) Các tài liệu và trích dẫn trong luận văn này được tham khảo từ các nguồn thực tế, có uy tín và độ chính xác cao; (iii) Các số liệu và kết quả của công trình này được tôi tự thực hiện một cách độc lập và trung thực.

Tp. Hồ Chí Minh,...tháng 06 năm 2022.

Sinh viên

Tóm tắt luận văn

Phát hiện té ngã (**fall detection**) là một trong những đề tài được quan tâm nghiên cứu trong những năm gần đây. Việc xác định té ngã chính xác và kịp thời có thể giúp con người (đặc biệt là người lớn tuổi) giảm thiểu các trường hợp tử vong hay chấn thương do ngã. Với sự phát triển của ngành trí tuệ nhân tạo, hệ thống phát hiện ngã áp dụng các kỹ thuật thị giác máy tính (**computer vision**) đã cho hiệu quả đáng kể.

Luận văn với mục tiêu nghiên cứu, thiết kế một hệ thống phát hiện té ngã dựa trên các kỹ thuật xử lý ảnh và thị giác máy tính. Đồng thời tìm hiểu và đánh giá các mô hình mạng áp dụng cho bài toán.

ABSTRACT

Fall detection is one of the topics of interest in research in recent years. Accurate fall identification can help people (especially the elderly) reduce deaths or injuries from falls. With the development of artificial intelligence industry in recent years, the fall detection system applying computer vision techniques has given remarkable efficiency.

The thesis aims to research and design a fall detection system based on computer vision techniques. At the same time, we will learn and evaluate network models applied to the problem.

Mục lục

LỜI CẢM ƠN	i
LỜI CAM ĐOAN	ii
TÓM TẮT LUẬN VĂN	iii
ABSTRACT	iv
DANH SÁCH HÌNH	viii
DANH SÁCH BẢNG	ix
1 GIỚI THIỆU	1
1.1 Tổng quan	1
1.2 Mục tiêu đề tài	2
2 CƠ SỞ LÝ THUYẾT	3
2.1 Một số phương pháp xây dựng hệ thống té ngã	3
2.1.1 Hệ thống với các thiết bị đặt trên cơ thể người (Wearable systems)	3
2.1.2 Hệ thống sử dụng các cảm biến được đặt cố định	4
2.1.3 Hệ thống sử dụng các công nghệ xử lý ảnh, thị giác máy tính	4
2.2 Giới thiệu về YOLO và bài toán phát hiện đối tượng.	5
2.2.1 Grid system	6
2.2.2 Convolutional Neural Network	8
2.2.3 Cấu trúc của YOLOv4	13
2.3 Mạng trí nhớ ngắn hạn định hướng dài hạn (Long short term memory)	15
2.3.1 Ý tưởng phía sau LSTM	16
2.3.2 Trình tự các bước LSTM	16
2.4 Sigmoid Function	18
2.5 Bài toán Human pose estimation	19
2.6 Mediapipe Pose	21
2.6.1 Pipeline của model Mediapipe Pose	22
2.6.2 Model pose detection cho bộ Pose detector	22
2.6.3 Tracking model cho bộ Pose Tracker	24
2.7 Những phần mềm và thư viện áp dụng trong luận văn	26
2.7.1 Ngôn ngữ lập trình Python	26
2.7.2 Opencv	27
2.7.3 Tensorflow	28
2.7.4 Flask	29

2.7.5	HTML	30
3	KẾT QUẢ VÀ PHÂN TÍCH	32
3.1	Phương pháp A: Nhận diện các tư thế sử dụng mô hình YOLOv4.	32
3.1.1	Sơ đồ giải thuật.	33
3.1.2	Sưu tầm dữ liệu và gán nhãn cho việc huấn luyện.	34
3.1.3	Huấn luyện mô hình.	35
3.1.4	Kết quả và phân tích	37
3.2	Phương pháp B: sử dụng Mediapipe kết hợp LSTM	40
3.2.1	Sơ đồ giải thuật	41
3.2.2	Model phân loại tê ngã.	43
3.2.3	Sưu tầm dữ liệu	44
3.2.4	Lấy mẫu dữ liệu cho việc huấn luyện	45
3.2.5	Sử dụng LSTM để huấn luyện mô hình	46
3.2.6	Kết quả và phân tích	47
3.3	Xây dựng ứng dụng web theo dõi hệ thống phát hiện tê ngã	51
3.3.1	Giới thiệu về ứng dụng web (web application)	51
3.3.2	Phần backend của trang web	52
3.3.3	Phần frontend giao diện người dùng của trang web.	53
3.3.4	Kết quả	57
4	Kết luận	58
4.1	Tóm tắt và kết luận chung	58
4.2	Hạn chế và hướng cải thiện mô hình.	59
4.3	Hướng phát triển	59
A	Code chương trình phát hiện té ngã của hệ thống	60
B	Code chương trình web cho hệ thống	61

Danh sách hình vẽ

1.1	Hiện tượng té ngã xảy ra phổ biến với nhóm người cao tuổi [14]	1
2.1	Một hệ thống phát hiện té ngã với các thiết bị có thể mang trên cơ thể người [28]	3
2.2	Một hệ thống phát hiện té ngã với các cảm biến được đặt cố định [27]	4
2.3	Một hệ thống phát hiện té ngã sử dụng công nghệ xử lý ảnh [27]	5
2.4	Ảnh sau khi detect.	6
2.5	Tâm của đối tượng.	7
2.6	Độ lệch a,b trong ô vuông chứa tâm.	8
2.7	Thông tin dự đoán ở mỗi ô.	8
2.8	Cấu trúc CNN [10].	9
2.9	Ví dụ minh họa [17]	11
2.10	Chỉ số IOU [17]	12
2.11	Cấu trúc CSP[8].	13
2.12	Cấu trúc DenseNet[8].	13
2.13	Một số cấu trúc phần cổ (Neck)[8].	14
2.14	Cấu trúc YOLOv4 [8].	14
2.15	Sự lặp lại kiến trúc module trong mạng RNN một tầng ẩn [16].	15
2.16	Sự lặp lại kiến trúc module trong mạng LSTM chứa 4 tầng ẩn [16]	15
2.17	các kí hiệu trong đồ thị mạng nơ ron [16].	15
2.18	Dường đi của ô trạng thái (cell state) trong mạng LSTM [16].	16
2.19	Cổng của hàm sigmoid trong LSTM [16].	16
2.20	Tầng cổng quên [16].	17
2.21	Cập nhật giá trị cho ô trạng thái: kết hợp 2 kết quả từ tầng cổng vào và tầng ẩn hàm tanh [16].	17
2.22	Ô trạng thái mới [16].	17
2.23	Chỉnh thông tin ở đầu ra thông qua hàm tanh [16].	18
2.24	Hàm Sigmoid [26].	18
2.25	Một ví dụ về object tracking [19]	19
2.26	Ước lượng tư thế người trong các hoạt động hàng ngày [6]	20
2.27	Model OpenPiPaf [21]	20
2.28	Model Mediapipe Pose [12]	21
2.29	Kết quả khi sử dụng model Mediapipe Pose [12]	21
2.30	Pipeline của model Mediapipe Pose [1]	22
2.31	Phát hiện khuôn mặt bằng model Blaze face [3]	23
2.32	Bức họa Virtruvian Man [5]	24
2.33	Vùng ROI toàn bộ cơ thể người [2]	24
2.34	Mediapose khi xác định được các điểm mốc [12]	25
2.35	Sơ đồ 33 điểm mốc [2]	25
2.36	Kiến trúc model Pose tracking cho pose tracker [2]	26

2.37 Ngôn ngữ lập trình python.[11]	26
2.38 Opencv.[22]	27
2.39 Xử lý ảnh trong opencv.[9]	28
2.40 Thư viện Tensorflow.	29
2.41 Framework Flask. [7]	30
2.42 Một tệp ví dụ về HTML.	31
3.1 So sánh YOLOv4 với các model phát hiện đối tượng khác[8].	32
3.2 Sơ đồ giải thuật.	33
3.3 Ảnh sưu tầm [18].	34
3.4 Ảnh tự chụp.	34
3.5 Gán nhãn sử dụng LabelImg.	35
3.6 Chỉ số mAP, loss trong quá trình huấn luyện.	36
3.7 Trường hợp bình thường.	37
3.8 Trường hợp té ngã 1.	37
3.9 Trường hợp té ngã 2.	38
3.10 Trường hợp té ngã 3.	38
3.11 Trường hợp phát hiện sai.	39
3.12 Một trường hợp về chuỗi tư thế của một hành động ngã.	40
3.13 Sơ đồ khối phương pháp B.	41
3.14 Model phân loại té ngã	43
3.15 Dữ liệu sưu tầm.	44
3.16 Phát hiện khung xương.	45
3.17 Tọa độ các điểm nút của 50 khung hình.	45
3.18 Mô hình huấn luyện LSTM.	46
3.19 Chỉ số loss của mô hình	46
3.20 Chỉ số accuracy của mô hình	47
3.21 Trường hợp bình thường và té ngã ngoài trời.	48
3.22 Trường hợp té ngã và bình thường trong phòng học.	49
3.23 Trường hợp bình thường và té ngã tại cơ quan làm việc.	49
3.24 Trường hợp không phát hiện được khung xương.	50
3.25 Trường hợp nhầm lẫn giữa ngủ và té ngã.	50
3.26 Hình ảnh minh họa 6.	51
3.27 Sơ đồ khối mới cho backend.	52
3.28 Dữ liệu thông tin đăng nhập của người dùng	53
3.29 Dữ liệu thông tin lịch sử ngã.	53
3.30 Trang chủ của website	54
3.31 Trang đăng ký tài khoản.	54
3.32 Trang đăng nhập	55
3.33 Trang lịch sử.	55
3.34 Ảnh của lần té ngã khi người dùng bấm vào nút view.	56
3.35 Trang trạng thái	56

Danh sách bảng

3.1 Confusion matrix	47
--------------------------------	----

Chương 1

GIỚI THIỆU

1.1 Tổng quan

Té ngã là sự mất thăng bằng ngoài ý muốn khiến cho cơ thể bất ngờ rơi xuống mặt đất, sàn nhà. Đây là một hiện tượng phổ biến trong cuộc sống thường ngày. Các chấn thương có thể xảy ra sau khi ngã bao gồm gãy xương chân, gãy xương tay, đứt dây chằng hay thậm chí là chấn thương sọ não. Tất cả mọi người đều không ngừng đối mặt với nguy cơ té ngã khi đang thực hiện các công việc thường ngày. Nguy cơ này càng tăng cao hơn đối với đối tượng người già. Theo tổ chức y tế thế giới 28-30% người có độ tuổi trên 65 bị ngã và gây tổn hại đến sức khỏe hằng năm [15]. Tỷ lệ này tăng nhanh đến 32 – 42% đối với nhóm người già trên 70 tuổi. Việc té ngã đối với người bình thường có thể ít gây nguy hiểm, tuy nhiên hiện tượng này rất nguy hiểm với nhóm người cao tuổi. Là nhóm người có sức khỏe, xương khớp hay các bộ phận khác bị lão hóa và dễ bị tổn thương. Ngoại trừ các chấn thương về thể chất, các chấn thương tâm lý như sợ ngã dẫn đến giảm các hoạt động hàng ngày khiến người bệnh thậm chí còn yếu hơn và dễ bị ngã trở lại.

Việc xác định té ngã chính xác và kịp thời có thể giúp con người (đặc biệt là người lớn tuổi) giảm thiểu các trường hợp tử vong hay chấn thương do ngã. Với sự phát triển của khoa học kỹ thuật, nhiều hệ thống phát hiện té ngã đã ra đời. Một hệ thống như vậy sẽ xác định các trường hợp ngã, cảnh báo mọi người về tình trạng khẩn cấp khi sự cố té ngã vừa xảy ra.



Hình 1.1: Hiện tượng té ngã xảy ra phổ biến với nhóm người cao tuổi [14]

Có khá nhiều cách khác nhau để thiết kế một hệ thống phát hiện tê ngã. Các hệ thống như vậy phụ thuộc vào loại thiết bị (phần cứng) được chọn để thu thập dữ liệu, vị trí đặt các thiết bị này và cách áp dụng tính năng phát hiện tê ngã của thiết bị.

1.2 Mục tiêu đề tài

Luận văn này được đưa ra với mục đích xây dựng và phát triển một hệ thống phát hiện tê ngã. Thông qua luận văn tìm hiểu các phương pháp thực hiện đề tài, nghiên cứu các công nghệ thị giác máy tính, các mô hình mạng học sâu (Deep learning). Hệ thống phát hiện tê ngã được xây dựng với các yêu cầu như sau:

- Hệ thống có thể hoạt động real-time.
- Tốc độ xử lý khung hình trên giây (fps) cao.
- Dễ dàng sử dụng và lắp đặt
- Độ chính xác cao.
- Xây dựng một trang web lưu lại các thông tin giúp người dùng theo dõi.

Thông qua việc xây dựng hệ thống tê ngã trên, tìm hiểu và học hỏi các phương pháp xử lý bài toán, các mô hình mạng học sâu cũng như các framework áp dụng vào đề tài.

Chương 2

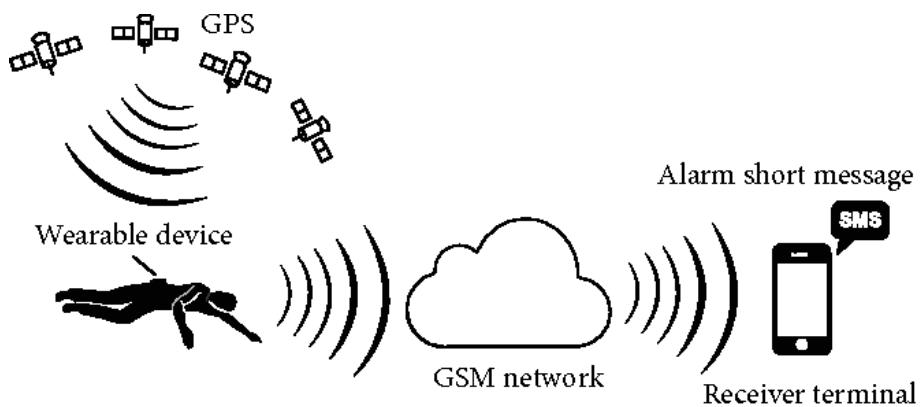
CƠ SỞ LÝ THUYẾT

2.1 Một số phương pháp xây dựng hệ thống té ngã

Có khá nhiều cách khác nhau để thiết kế một hệ thống phát hiện té ngã. Các hệ thống như vậy phụ thuộc vào loại thiết bị (phần cứng) được chọn để thu thập dữ liệu, vị trí đặt các thiết bị này và cách áp dụng tính năng phát hiện té ngã của thiết bị. Nhìn chung thì sẽ có 3 kiểu hệ thống phát hiện té ngã: hệ thống với các thiết bị đặt trên cơ thể người, hệ thống sử dụng các cảm biến được đặt cố định, hệ thống sử dụng các công nghệ xử lý ảnh, thị giác máy tính. Chúng ta sẽ tìm hiểu kĩ hơn về 3 kiểu hệ thống này ở phần tiếp theo.

2.1.1 Hệ thống với các thiết bị đặt trên cơ thể người (Wearable systems)

Hệ thống với các thiết bị đặt trên cơ thể người (Wearable systems) sử dụng các cảm biến được mang theo cơ thể người như gia tốc kế, cảm biến áp suất, máy đo độ nghiêng, con quay hồi chuyển, ... Ngày nay các cảm biến này thường được nhúng vào đồng hồ, điện thoại thông minh hay thậm chí quần áo của người được giám sát. Hệ thống này được thiết kế dựa trên ý tưởng thực tế rằng hầu hết các cú ngã đều sẽ gây ra sự thay đổi đáng kể về định hướng hoặc vị trí của cơ thể đột ngột. Sự thay đổi đó có thể được đo chính xác bằng các cảm biến như gia tốc kế, con quay hồi chuyển, máy đo độ cao khí quyển,...

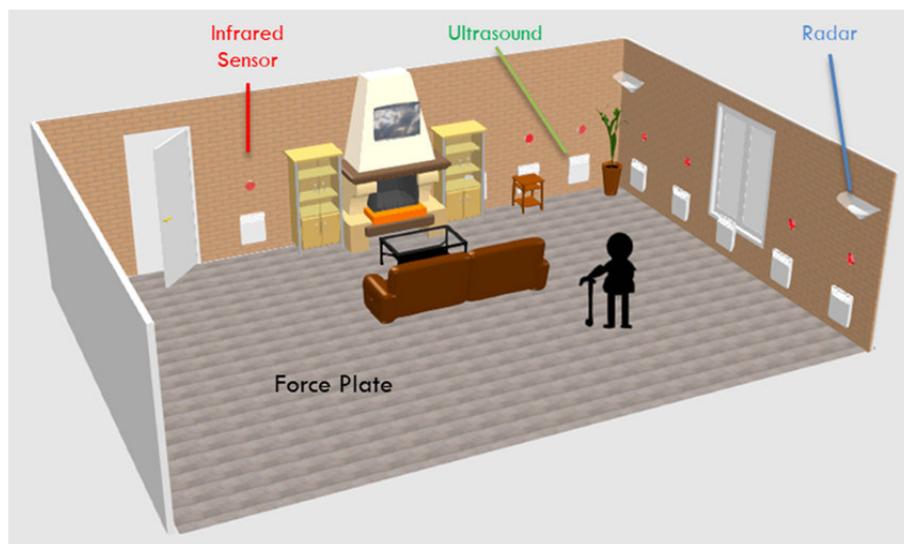


Hình 2.1: Một hệ thống phát hiện té ngã với các thiết bị có thể mang trên cơ thể người [28]

Hình 2.1 là một ví dụ về hệ thống phát hiện té ngã với các thiết bị đặt trên cơ thể người. Khi phát hiện té ngã thì các thiết bị này lập tức gửi thông báo đến các sever. Sever sau khi nhận được thông báo sẽ chuyển tiếp đến thiết bị đầu cuối báo hiệu cho người nhà hay người bảo hộ. Ưu điểm của hệ thống loại này là tính linh động, kín đáo. Nhược điểm lớn nhất của chúng là độ chính xác và hiệu suất chưa cao khi sử dụng thực tế vì các cảm biến rất dễ bị ảnh hưởng bởi các tác nhân môi trường gây nhiễu thiết bị.

2.1.2 Hệ thống sử dụng các cảm biến được đặt cố định

Hệ thống sử dụng các cảm biến được đặt cố định bao gồm các cảm biến như áp suất, hồng ngoại, âm thanh, tần số vô tuyến,... Chúng được đặt cố định xung quanh khu vực được giám sát như nhà ở, viện dưỡng lão. Ý tưởng chính của phương pháp này là chúng quan sát môi trường xung quanh và theo dõi tư thế của con người. Các thông số quan sát được bởi các cảm biến này chủ yếu là chuyển động của tư thế người tư thế, áp lực trên mặt đất, âm thanh và thời gian rơi cũng như thời gian nằm trên mặt đất sau khi rơi. Khác với hệ thống với các thiết bị đặt trên cơ thể người chỉ theo dõi sự thay đổi của tư thế con người, hệ thống loại này còn dựa vào các yếu tố của môi trường xung quanh được giám sát. Do đó nó sẽ giảm thiểu các tác nhân gây nhiễu bên ngoài ảnh hưởng đến việc xác định việc té ngã.



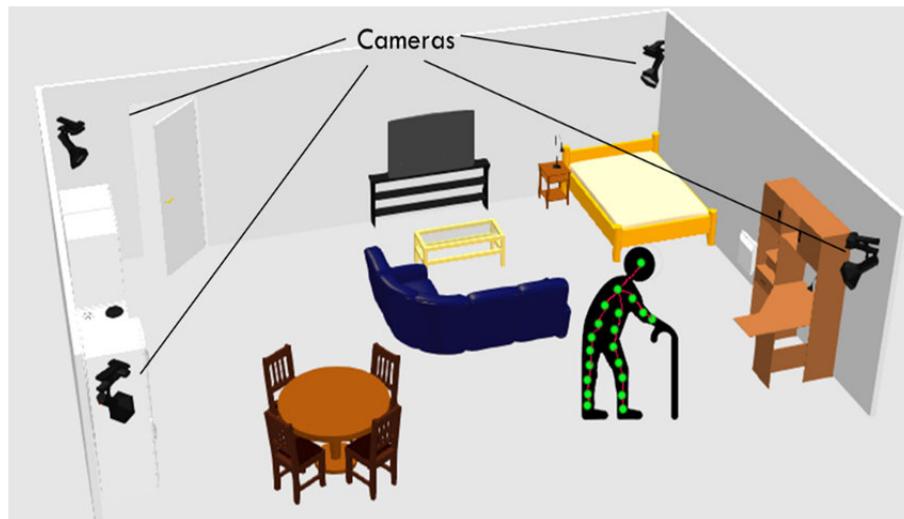
Hình 2.2: Một hệ thống phát hiện té ngã với các cảm biến được đặt cố định [27]

Hình 2.2 là một ví dụ về hệ thống phát hiện té ngã sử dụng các cảm biến được đặt cố định. Các cảm biến được gắn âm trong tường và dưới sàn nhà. Ưu điểm của hệ thống phát hiện té ngã sử dụng các cảm biến được đặt cố định là có thể hoạt động trong những môi trường ánh sáng yếu, cung cấp phân tích toàn diện hơn về tư thế của người, bảo vệ tính riêng tư của người dùng. Hạn chế của hệ thống loại này là chỉ phù hợp với môi trường trong nhà, chỉ có thể sử dụng cho một người, quá trình lắp đặt hệ thống phức tạp và tốn kém.

2.1.3 Hệ thống sử dụng các công nghệ xử lý ảnh, thị giác máy tính

Trong những năm qua, lĩnh vực trí tuệ nhân tạo và thị giác máy tính đã có sự phát triển

nhanh chóng. Sự phát triển của việc sử dụng các mạng lưới thần kinh nhân tạo giúp máy móc có khả năng nhận biết và phát hiện trực quan các đối tượng, sự kiện và hành động của con người. Đặc biệt là sự ra đời của các API về tracking đã giúp cho việc phát hiện té ngã trở nên hiệu quả. Ý tưởng chính của phương pháp này là sử dụng các camera để thu thập hình ảnh, sau đó dựa trên những hình ảnh thu thập được qua các bước tiền xử lý, trích xuất đặc trưng và nhận dạng sự kiện. Việc trích xuất đặc trưng có nhiều cách khác nhau như: giám sát sự thay đổi tư thế, xác định tư thế người dùng, theo dõi các điểm mốc trên cơ thể người dùng.



Hình 2.3: Một hệ thống phát hiện té ngã sử dụng công nghệ xử lý ảnh [27]

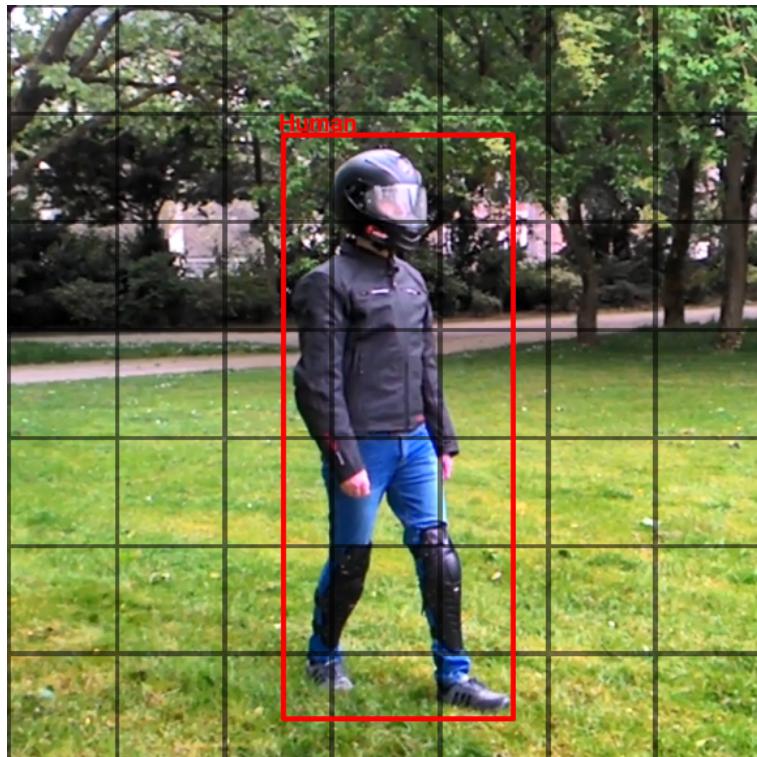
Hình 2.3 là một ví dụ về hệ thống phát hiện té ngã sử dụng các công nghệ xử lý ảnh, thị giác máy tính. Các camera được đặt xung quanh môi trường để quan sát để thu thập các hình ảnh từ môi trường. Hệ thống này trích xuất đặc trưng từ các điểm mốc ở cơ thể người. Nhìn chung, ưu điểm về các hệ thống loại này là việc xác định nhanh với độ chính xác cao, có thể phát hiện được nhiều mục tiêu và real-time, việc lắp đặt tương đối dễ dàng. Ngoài những lợi thế trên thì hệ thống này có các hạn chế như chi phí lắp đặt cao, các vấn đề về quyền riêng tư của người dùng, không hiệu quả với những trường hợp rơi vào điểm mù của camera.

Luận văn này trình bày hai phương pháp xây dựng hệ thống sử dụng các công nghệ xử lý ảnh, thị giác máy tính. Phương pháp A sử dụng **Yolov4** [8] để nhận dạng tư thế ở mỗi khung hình và so sánh sự thay đổi đột ngột tư thế của hai khung hình liên tiếp để kết luận có xuất hiện té ngã hay không. Phương pháp B ứng dụng module tracking của **MediaPipe Pose** [12] trích xuất các điểm mốc trên cơ thể người theo thời gian. Từ đó dự đoán sự thay đổi các điểm trên thông qua một mô hình mạng học sâu **LSTM** [20] để phát hiện té ngã.

2.2 Giới thiệu về YOLO và bài toán phát hiện đối tượng.

You only look once (YOLO) [17] là một mô hình CNN để nhận dạng đối tượng với ưu điểm nổi trội là nhanh hơn nhiều so với những mô hình cũ. Thậm chí có thể chạy tốt trên những IOT device như raspberry pi. Chúng ta có YOLO v1, về sau chúng ta còn có YOLO v2, v3, v4 chạy nhanh hơn nhưng phức tạp hơn và khó cài đặt.

Đầu vào của mô hình này là một bức ảnh, đối với bài toán phát hiện đối tượng, chúng ta không chỉ phải phân loại được đối tượng trong ảnh mà còn phải xác định được vị trí của đối tượng đó.

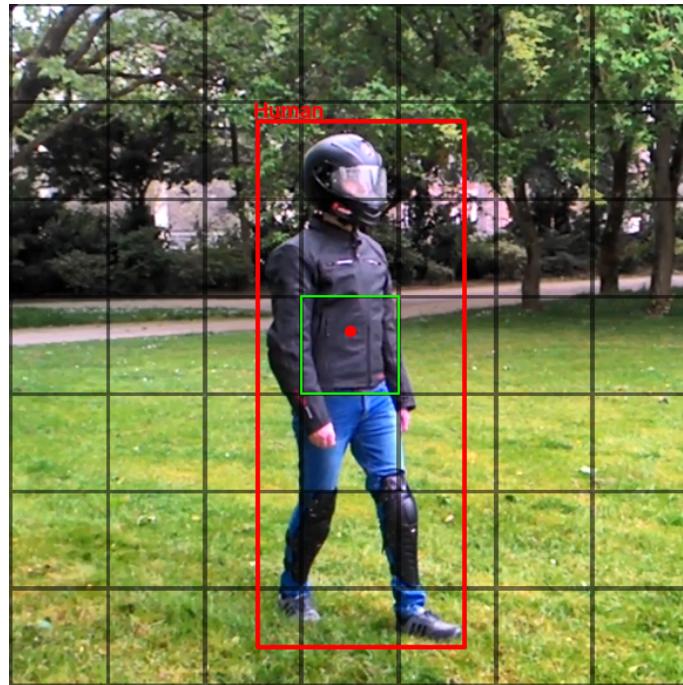


Hình 2.4: Ảnh sau khi detect.

2.2.1 Grid system

Chia ảnh thành ma trận ô vuông 7×7 , mỗi ô vuông sẽ gồm một tập các thông tin mà mô hình cần phải dự đoán. Mỗi ô vuông đó chứa đối tượng duy nhất. Tâm của đối tượng cần xác định nằm trong ô vuông nào thì xem như ô vuông đó chứa đối tượng đó. Ví dụ tâm của người trong hình nằm trong ô vuông màu xanh, do đó mô hình phải dự đoán được nhãn của ô vuông đó là người. Lưu ý, cho dù phần ảnh người có nằm ở ô vuông khác mà tâm không thuộc ô vuông đó thì vẫn không tính là ô vuông này chứa người.

Ngoài ra, nếu có nhiều tâm nằm trong một ô vuông thì chúng ta vẫn chỉ gán một nhãn cho ô vuông đó thôi. Chính ràng buộc mỗi ô vuông chỉ chứa một đối tượng là nhược điểm của mô hình này. Nó làm cho ta không thể phát hiện những đối tượng có tâm nằm cùng một ô vuông. Tuy nhiên chúng ta có thể tăng grid size từ 7×7 lên kích thước lớn hơn để có thể phát hiện được nhiều đối tượng hơn. Ngoài ra, kích thước của ảnh đầu vào phải là bội số của grid size.

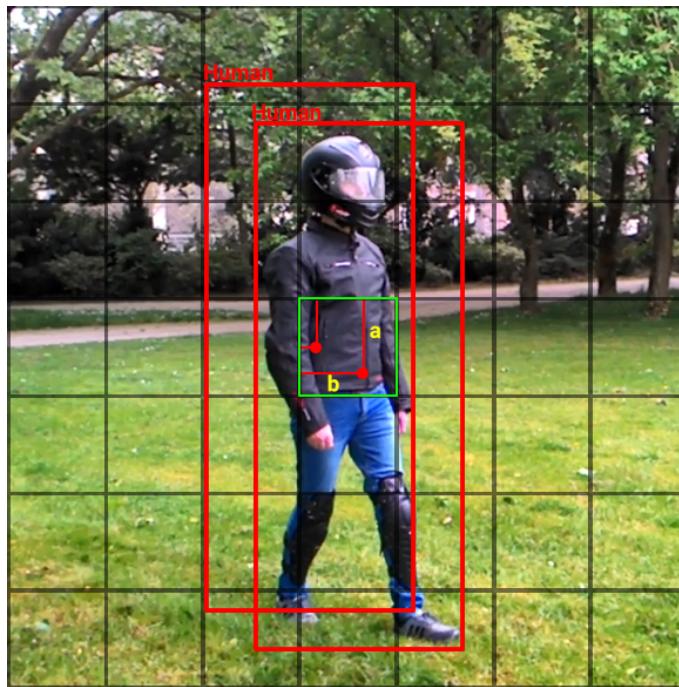


Hình 2.5: Tâm của đối tượng.

Mỗi ô vuông chịu trách nhiệm dự đoán 2 boundary box của đối tượng. Mỗi boundary box dự đoán có chứa đối tượng hay không và thông tin vị trí của boundary box gồm tâm, chiều dài, chiều rộng boundary box của đối tượng đó. Ví vụ ô vuông màu xanh cần dự đoán 2 boundary box chứa người như hình minh họa ở dưới. Lưu ý, lúc cài đặt chúng ta không dự đoán giá trị pixel mà cần phải chuẩn hóa kích thước ảnh về đoạn từ [0-1] và dự đoán độ lệch của tâm đối tượng đến box chứa đối tượng đó. Ví dụ, chúng ta thay vì dữ đoán vị trí pixel của điểm màu đỏ, thì cần dữ đoán độ lệch a, b trong ô vuông chứa tâm đối tượng.

Tổng hợp lại, với mỗi ô vuông chúng ta cần dữ đoán các thông tin sau :

- Ô vuông có chứa đối tượng nào hay không?
- Dự đoán độ lệch 2 box chứa object so với ô vuông hiện tại.
- Lớp của object đó.



Hình 2.6: Độ lệch a,b trong ô vuông chứa tâm.

Như vậy với mỗi ô vuông chúng ta cần dãy đoán một vector có ($n_{box} + 4 * n_{box} + n_{class}$) chiều. Ví dụ, chúng ta cần dự đoán 2 box, và 3 lớp đối với mỗi ô vuông thì chúng sẽ có một ma trận 3 chiều $7 \times 7 \times 13$ chứa toàn bộ thông tin cần thiết.

Object 1?	Object 2?	Offset x1	Offset y1	Width 1	Height 1	Offset x2	Offset y2	Width 2	Height 2	0	0	1
-----------	-----------	-----------	-----------	---------	----------	-----------	-----------	---------	----------	---	---	---

Hình 2.7: Thông tin dự đoán ở mỗi ô.

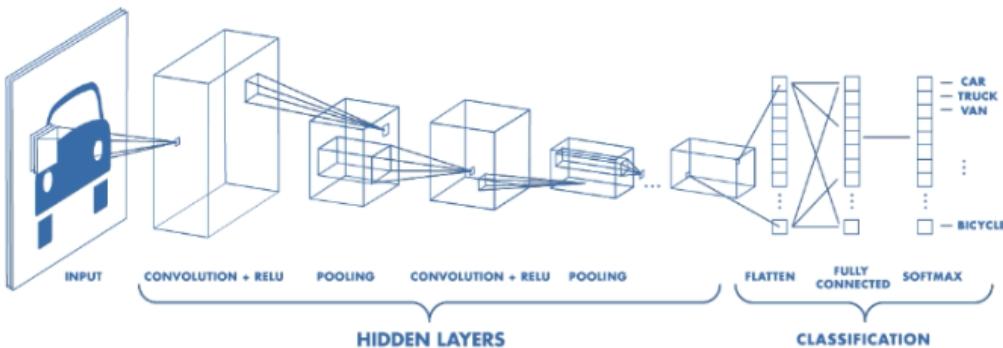
2.2.2 Convolutional Neural Network

Tích chập được ứng dụng phổ biến trong lĩnh vực thị giác máy tính. Thông qua các phép tích chập, các đặc trưng chính từ ảnh được trích xuất và truyền vào các tầng tích chập (layer convolution). Mỗi một tầng tích chập sẽ bao gồm nhiều đơn vị mà kết quả ở mỗi đơn vị là một phép biến đổi tích chập từ layer trước đó thông qua phép nhân tích chập với bộ lọc [10]. Một mạng nơ tron tích chập (convolution newral network) có ba quá trình cơ bản như sau:

- Quá trình tích chập (convolution): Dùng để trích xuất đặc trưng 2 chiều bằng việc tính tích chập giữa ma trận đầu vào với bộ lọc để tạo ra ma trận mới (ma trận đặc trưng). Quá trình này diễn ra ở đầu mạng và thường kèm với hàm kích hoạt (activation function) là hàm ReLU.
- Quá trình tổng hợp (max pooling): Các tầng càng về sau khi trích xuất đặc trưng sẽ cần số lượng tham số lớn do chiều sâu được qui định bởi số lượng các kênh ở các tầng sau thường tăng tiến theo cấp số nhân. Điều đó làm tăng số lượng tham số và khối lượng tính

toán trong mạng nơ ron. Do đó để giảm tải tính toán chúng ta sẽ cần giảm kích thước các chiều của khối ma trận đầu vào hoặc giảm số đơn vị của tầng. Vì mỗi một đơn vị sẽ là kết quả đại diện của việc áp dụng 1 bộ lọc để tìm ra một đặc trưng cụ thể nên việc giảm số đơn vị sẽ không khả thi. Giảm kích thước khối ma trận đầu vào thông qua việc tìm ra 1 giá trị đại diện cho mỗi một vùng không gian mà bộ lọc đi qua sẽ không làm thay đổi các đường nét chính của bức ảnh nhưng lại giảm được kích thước của ảnh.

- Quá trình kết nối hoàn toàn (fully connected): Sau khi đã giảm kích thước đến một mức độ hợp lý, ma trận cần được trải phẳng ra (flatten) thành một vector và sử dụng các kết nối hoàn toàn giữa các tầng. Quá trình này sẽ diễn ra cuối mạng CNN và sử dụng hàm kích hoạt là ReLU. Tầng kết nối hoàn toàn cuối cùng (fully connected layer) sẽ có số lượng đơn vị bằng với số lớp cần phân loại ban đầu (classes) và áp dụng hàm kích hoạt là softmax nhằm mục đích tính phân phối xác suất.



Hình 2.8: Cấu trúc CNN [10].

Chúng ta đã cần biết phải dự đoán những thông tin nào đối với mỗi ô vuông, điều quan trọng tiếp theo là xây dựng một mô hình CNN có cho ra output với shape phù hợp theo yêu cầu của chúng ta, tức là gridsize x gridsize x (nbox+4*nbox+nclass). Ví dụ với gridsize là 7x7 là mỗi ô vuông dự đoán 2 boxes, và có 3 loại object tất cả thì chúng ta phải cần output có shape 7x7x13 từ mô hình CNN.

YOLO sử dụng linear regression để dự đoán các thông tin ở mỗi ô vuông. Do đó, ở layer cuối cùng chúng ta sẽ không sử dụng bất kì hàm kích hoạt nào cả. Với ảnh đầu vào là 448x448, mô hình CNN có 6 tầng max pooling với size 2x2 sẽ giảm 64 lần kích thước ảnh xuống còn 7x7 ở output đầu ra. Đồng thời thay vì sử dụng tầng full connected ở các tầng cuối cùng, chúng ta có thể thay thế bằng tầng 1x1 conv với 13 feature maps để output shape dễ dàng cho ra 7x7x13.

Chúng ta đã định nghĩa được những thông tin mà mô hình cần phải dự đoán, và kiến trúc của mô hình CNN. Nay giờ là lúc mà ta sẽ định nghĩa hàm lỗi YOLO sử dụng hàm độ lỗi bình phương giữ dự đoán và nhãn để tính toán độ lỗi cho mô hình. Cụ thể, độ lỗi chung của chúng ta sẽ là tổng của 3 độ lỗi con như sau:

- Độ lỗi của việc dự đoán nhãn đối tượng (Classification loss)
- Độ lỗi của dự đoán tọa độ vị trí cũng như chiều dài, rộng của boundary box (Localization loss)
- Độ lỗi của ô vuông có chứa đối tượng hay không (Confidence loss)

Trong quá trình huấn luyện, mô hình sẽ nhìn vào những ô vuông có chứa object. Tăng classification score lớp đúng của object đó lên. Sau đó, cũng nhìn vào ô vuông đó, tìm boundary box tốt nhất trong 2 boxes được dự đoán. Tăng localization score của boundary box đó lên, thay đổi thông tin boundary box để gần đúng với nhãn. Đối với những ô vuông không chứa object, giảm confidence score và chúng ta sẽ không quan tâm đến classification score và localization score của những ô vuông này. Kế tiếp, ta sẽ đi lần lượt vào chi tiết ý nghĩa của các độ lỗi trên.

- Classification loss:

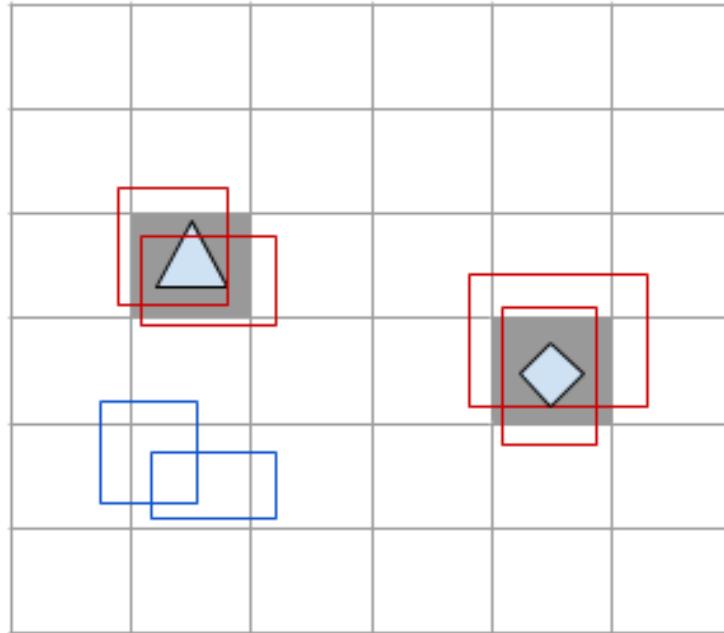
Chúng ta chỉ tính classification loss cho những ô vuông được đánh nhãn là có chứa đối tượng. Classification loss tại những ô vuông đó được tính bằng độ lỗi bình phương giữa nhãn được dự đoán và nhãn đúng của nó.

$$L_{\text{classification}} = \sum_{i=0}^{S^2} \mathbb{I}_i^{\text{obj}} \sum_{c \in \text{class}} (p_i(c) - \hat{p}_i(c))^2$$

$\mathbb{I}_i^{\text{obj}}$: bằng 1 nếu ô vuông đang xét có object, ngược lại bằng 0.

$\hat{p}_i(c)$: là xác suất của lớp c tại ô vuông tương ứng mà mô hình dự đoán.

Ví dụ, trong hình minh họa ở trên, chúng ta có 2 đối tượng tại ô vuông có tọa độ (3,2) và (4,5), chứa đối tượng là hình tam giác và hình tứ giác đều. Độ lỗi classification loss chỉ tính cho 2 đối tượng này mà ko quan tâm đến những ô vuông khác. Lúc cài đặt chúng ta phải nhân với một mask để triệt tiêu giá trị lỗi tại những ô vuông không có đối tượng.



Hình 2.9: Ví dụ minh họa [17]

- Localization Loss

Localization loss dùng để tính giá trị lỗi cho boundary box được dự đoán bao gồm offset x,y và chiều dài, rộng so với nhãn chính xác của chúng. Ta không tính toán trực tiếp giá trị lỗi này trên kích thước của ảnh mà cần chuẩn dưới kính thước ảnh về đoạn [0-1] đối với tọa độ điểm tâm, và không dữ đoán trực tiếp điểm tâm mà phải dự đoán giá trị lệch offset x,y so với ô vuông tương ứng. Việc chuẩn hóa kích thước ảnh và dự đoán offset làm cho mô hình nhanh hối tự hơn so với việc dự đoán giá trị mặc định.

$$L_{\text{localization}} = \sum_{i=0}^{s^2} \sum_{j=0}^B I_{ij}^{\text{obj}} \left[(\text{offset } x_i - \text{off set } x_i)^2 + (\text{offsety } - \text{off sety } y_i)^2 + (\text{width}_i - \text{width}_i)^2 + (\text{height}_i - \text{height}_i)^2 \right]$$

Độ lỗi localization loss được tính bằng tổng độ lỗi bình phương của offsetx, offsety và chiều dài, rộng trên tất cả các ô vuông có chứa đối tượng. Tại mỗi ô vuông đúng, ta chọn 1 boundary box có IOU (Intersect over union) tốt nhất, rồi sau đó tính độ lỗi theo các boundary box này. Theo hình minh họa trên chúng ta có 4 boundary box tại ô vuông đúng có viền màu đỏ, chúng ta chọn 1 box tại mỗi ô vuông để tính độ lỗi. Còn box xanh được bỏ qua. Localization loss là độ lỗi quan trọng nhất trong 3 loại độ lỗi trên. Vì vậy, ta cần đặt trọng số cao hơn cho độ lỗi này.

- Confidence loss

Confidence loss thể hiện độ lỗi giữa dự đoán boundary box đó chứa đối tượng so với nhãn thực tế tại ô vuông đó. Độ lỗi này tính trên tất cả ô vuông có chứa và không chứa đối tượng.

$$L_{\text{confidence}} = \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{\text{bj}} (C_i - C_i)^2 + \lambda_{\text{noobject}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{\text{noobj}} (C_i - C_i)^2$$

Độ lỗi này là độ lỗi bình phương của dự đoán boundary đó chứa object với nhãn thực tế của ô vuông tại vị trí tương ứng, chúng ta lưu ý rằng, độ lỗi tại ô vuông mà nhãn chứa object quan trọng hơn là độ lỗi tại ô vuông không chứa object, vì vậy chúng ta cần sử dụng hệ số lambda để cân bằng.

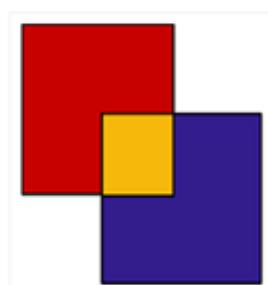
Cuối cùng, tổng lỗi của chúng ta sẽ bằng tổng của 3 loại độ lỗi trên:

$$L_{\text{total}} = L_{\text{classification}} + L_{\text{localization}} + L_{\text{confidence}}$$

Dự đoán lớp và tạo boundary box sau quá trình huấn luyện, ta chỉ giữ lại những boundary box mà có chứa đối tượng. Để làm điều này, chúng ta cần tính tích của xác suất có điều kiện ô vuông thuộc về lớp i nhãn với xác suất ô vuông đó chứa object, chỉ giữ lại những boundary box có giá trị này lớn hơn ngưỡng nhất định.

Mỗi đối tượng có thể có nhiều boundary box khác nhau do mô hình dự đoán. Để tìm boundary box tốt nhất các đối tượng, chúng ta có thể dùng thuật toán non-maximal suppression để loại những boundary box giao nhau nhiều, tức là có IOU giữ 2 boundary box lớn.

Để tính IOU giữ 2 box chúng ta cần tính diện tích giao nhau giữa 2 box chia cho tổng diện tích của 2 box đó.



$$iou = \frac{S_{\text{vàng}}}{S_{\text{đỏ}} + S_{\text{tím}} + S_{\text{vàng}}}$$

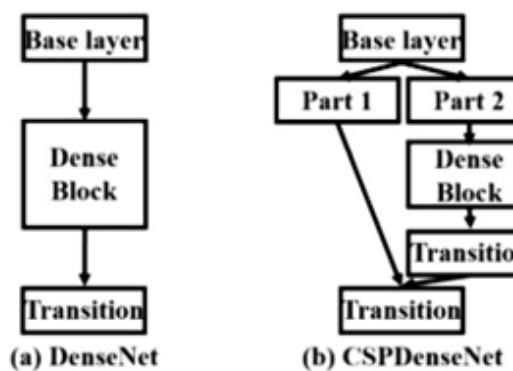
Hình 2.10: Chỉ số IOU [17]

2.2.3 Cấu trúc của YOLOv4

Cấu trúc của Yolov4 [8] được tác giả chia làm 4 phần:

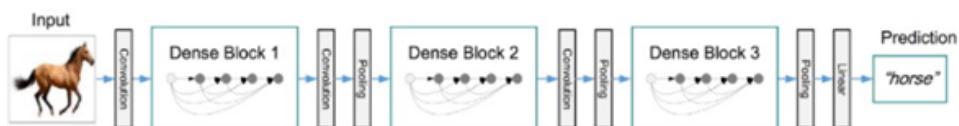
- Backbone (xương sống): trích xuất đặc trưng sử dụng CSPDarknet53.

Cross-Stage-Partial connections (CSP) có nguồn gốc từ kiến trúc DenseNet sử dụng đầu vào trước đó và nối nó với đầu vào hiện tại trước khi chuyển vào Dense layer. Nó có nhiệm vụ chia đầu vào của khối thành 2 phần, một phần sẽ qua các khối chập, và phần còn lại đi thẳng tới cuối khối. Sau đó hai phần sẽ được cộng lại và đưa vào khối tiếp theo. Ý tưởng ở đây là loại bỏ các nút thắt tính toán trong DenseNet và cải thiện việc học bằng cách chuyển phiên bản chưa chỉnh sửa của feature maps (bản đồ đặc trưng).



Hình 2.11: Cấu trúc CSP[8].

DenseNet (Dense connected convolutional network) là một trong những netwok mới nhất cho visual object recognition. Nó cũng gần giống Resnet nhưng có một vài điểm khác biệt. Densenet có cấu trúc gồm các dense block và các transition layers. Được stack dense block-transition layers-dense block- transition layers như hình vẽ. Với CNN truyền thống nếu chúng ta có L layer thì sẽ có L connection, còn trong densenet sẽ có $L(L+1)/2$ connection (tức là các lớp phía trước sẽ được liên kết với tất cả các lớp phía sau nó).

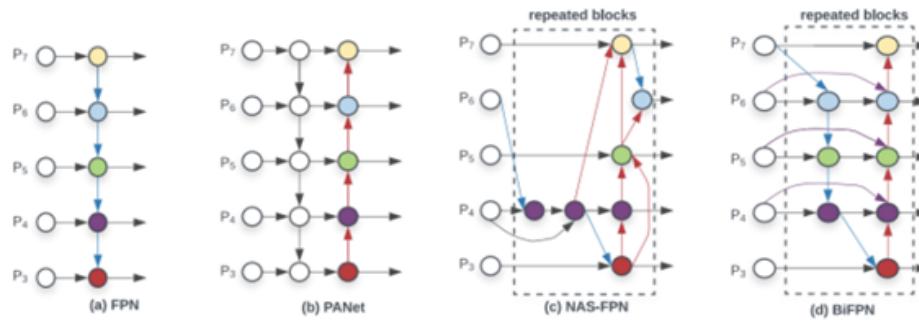


Hình 2.12: Cấu trúc DenseNet[8].

- Neck (cổ): tổng hợp đặc trưng, có nhiệm vụ trộn và kết hợp các features map đã học được thông qua quá trình trích xuất đặc trưng ở backbone và quá trình nhận dạng Dense prediction.

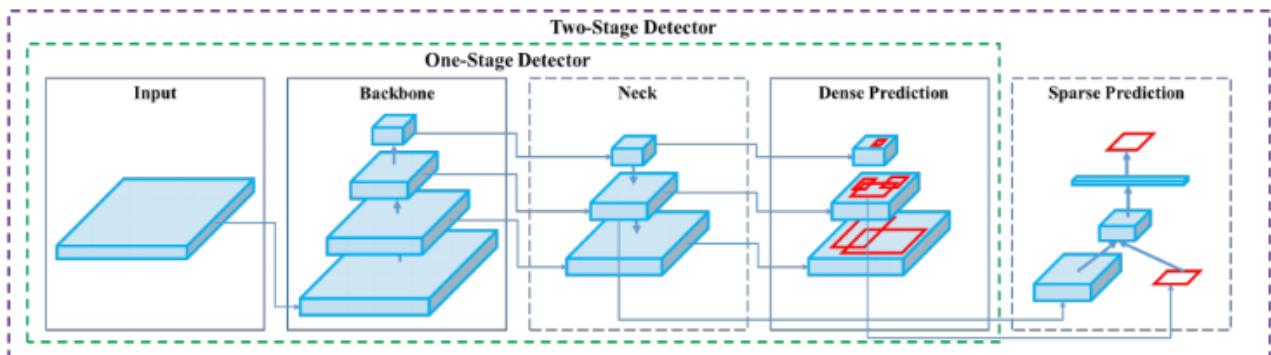
Neck có nhiệm vụ trộn và kết hợp các features map đã học được thông qua quá trình trích xuất đặc trưng ở backbone và quá trình nhận dạng Dense prediction. Với mỗi lần thực hiện detect với các kích thước ảnh rescale khác nhau tác giả đã thêm các luồng đi từ dưới lên và các luồng đi từ trên xuống vào cùng nhau hoặc được nối với nhau trước khi đưa vào head(phần đầu), từ đó lớp nhận dạng sẽ chứa thông tin phong phú hơn.

Tác giả của YOLOv4 đã cho phép tùy biến sử dụng các cấu trúc cho phần Neck: FPN, PAN, NAS-FPN, BiFPN, ASFF, SFAM, SSP.[8]



Hình 2.13: Một số cấu trúc phần cổ (Neck)[8].

- Dense prediction(dự đoán dày đặc): sử dụng các one-stage-detection như YOLO, Single Shot Detector (SSD).
- Sparse Prediction (dự đoán thưa thớt): sử dụng các Two-stage-detection như RCNN.

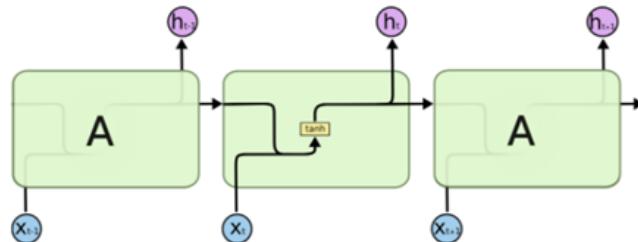


Hình 2.14: Cấu trúc YOLOv4 [8].

2.3 Mạng trí nhớ ngắn hạn định hướng dài hạn (Long short term memory)

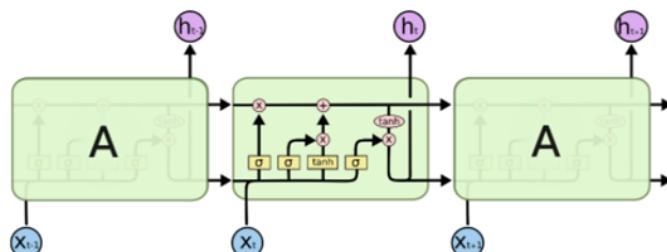
Mạng trí nhớ ngắn hạn định hướng dài hạn hay còn được viết tắt là **LSTM** là một kiến trúc đặc biệt của **Recurrent Neural Network (RNN)** có khả năng học được sự phức tạp trong dài hạn (long-term dependencies) được giới thiệu bởi Hochreiter và Schmidhuber vào năm 1997. Kiến trúc này đã được phổ biến và sử dụng rộng rãi cho tới ngày hôm nay.

Tuy là cấu trúc của LSTM có phần phức tạp hơn RNN nhưng vẫn giữ đặc tính cơ bản là sao chép kiến trúc theo dạng chuỗi. Một mạng RNN cơ bản sẽ có cấu trúc rất đơn giản như là tầng ẩn có một hàm tanh như hình bên dưới.



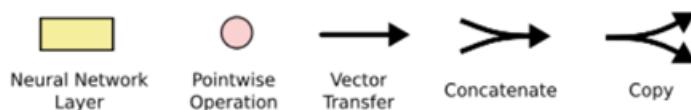
Hình 2.15: Sự lặp lại kiến trúc module trong mạng RNN một tầng ẩn [16].

LSTM cũng có chuỗi dạng như thế nhưng phần kiến trúc lặp lại khác biệt hơn. Thay vì có một tầng đơn như RNN, chúng có tới bốn tầng ẩn (3 sigmoid và 1 tanh) hoạt động với nhau theo cấu trúc đặc biệt.



Hình 2.16: Sự lặp lại kiến trúc module trong mạng LSTM chứa 4 tầng ẩn [16]

Một số ký hiệu:



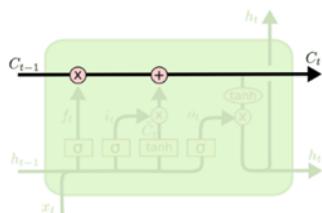
Hình 2.17: các ký hiệu trong đồ thị mạng nơ ron [16].

Trong sơ đồ trên, mỗi phép tính đều được triển khai trên một vector. Trong đó, hình tròn màu hồng (Pointwise Operation) biểu diễn một toán tử đối với vector như là phép cộng, nhân

vô hướng các vector. Ô màu vàng (Neural Network Layer) thể hiện hàm activation mà mạng nơ ron sử dụng để học trong tầng ẩn, thường là các hàm phi tuyến như tanh, sigmoid. Ký hiệu hai đường thẳng nhập vào (Concatenate) thể hiện phép chập kết quả trong khi ký hiệu hai đường thẳng rẽ nhánh (Copy) thể hiện cho nội dung vector trước đó được sao chép để đi tới một phần khác của mạng nơ ron.

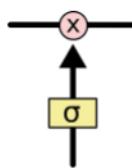
2.3.1 Ý tưởng phía sau LSTM

Thành phần chính của LSTM là ô trạng thái (cell state). Chính là đường thẳng chạy ngang qua như hình bên dưới. Ô trạng thái là một dạng băng chuyên chạy thẳng xuyên suốt toàn bộ chuỗi với chỉ một vài tương tác tuyến tính nhỏ giúp cho thông tin có thể truyền dọc theo đồ thị mạng nơ ron ổn định.



Hình 2.18: Đường đi của ô trạng thái (cell state) trong mạng LSTM [16].

LSTM có khả năng xóa và thêm thông tin vào ô trạng thái và điều chỉnh các luồng thông tin này một cách cẩn thận thông qua các cấu trúc gọi là cổng. Cổng là cơ chế đặc biệt để điều chỉnh luồng thông tin đi qua. Chúng được tổng hợp bởi một tầng ẩn của hàm activation sigmoid và với một toán tử nhân như hình sau.

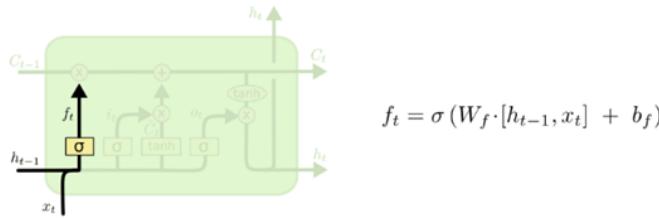


Hình 2.19: Cổng của hàm sigmoid trong LSTM [16].

Hàm sigmoid sẽ cho đầu ra là một giá trị xác suất nằm trong khoảng từ 0 đến 1, thể hiện rằng có bao nhiêu phần thông tin sẽ đi qua cổng. Giá trị bằng 0 nghĩa là không cho phép thông tin nào đi qua, giá trị bằng 1 sẽ cho toàn bộ thông tin đi qua. Một mạng LSTM sẽ có 3 cổng có kiến trúc dạng này để bảo vệ và kiểm soát các ô trạng thái (cell state).

2.3.2 Trình tự các bước LSTM

Bước đầu tiên của LSTM là sẽ quyết định xem những thông tin nào sẽ được cho phép vào ô trạng thái và được kiểm soát bởi hàm sigmoid trong tầng quên (forget gate layer). Đầu vào



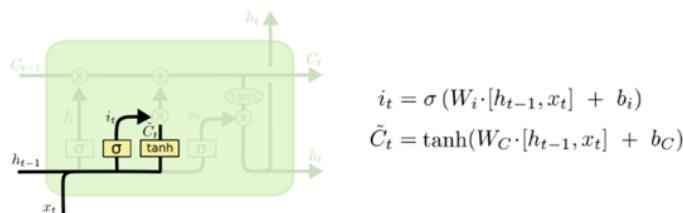
Hình 2.20: Tầng cổng quên [16].

nhận hai giá trị $h(t-1)$ và x_t sau đó trả về một giá trị từ 0 đến 1. Nếu giá trị bằng 1 tức là giữ lại toàn bộ thông tin và ngược lại

Ví dụ một mô hình ngôn ngữ đang cố dự đoán từ kế tiếp dựa vào các từ trước đó. Trong trường hợp này, ô trạng thái có thể bao gồm loại chủ ngữ và giới tính,... và những thành phần này luôn luôn thay đổi. Do đó tầng quên cho phép cập nhật thông tin mới và lưu giữ giá trị của nó khi có thay đổi theo thời gian.

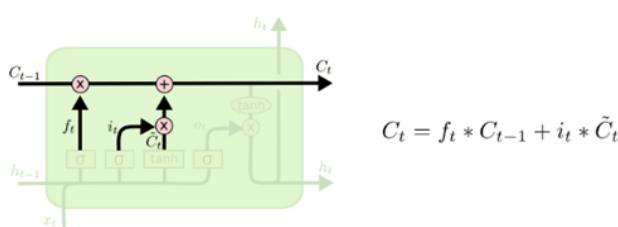
Bước kế tiếp sẽ quyết định loại thông tin nào sẽ được lưu trữ trong ô trạng thái. Bước này có 2 phần. Phần đầu là một tầng ẩn của hàm sigmoid được gọi là tầng cổng vào (input gate layer) quyết định giá trị nào sẽ được cập nhật. Tiếp theo, tầng ẩn hàm tanh sẽ tạo ra một vector của một giá trị trạng thái mới mà có thể được thêm vào trạng thái. Sau cùng kết hợp kết quả của 2 tầng này để tạo thành một cập nhật cho trạng thái.

Trong mô hình ngôn ngữ, chúng ta muốn thêm loại của một chủ ngữ mới vào ô trạng thái để thay thế phần trạng thái cũ muốn quên đi.



Hình 2.21: Cập nhật giá trị cho ô trạng thái: kết hợp 2 kết quả từ tầng cổng vào và tầng ẩn hàm tanh [16].

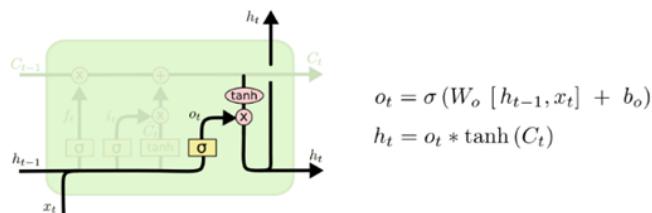
Đây là thời điểm để cập nhật một ô trạng thái cũ C_{t-1} , sang một trạng thái mới C_t . Những bước trước đã quyết định làm việc gì thì tại bước này sẽ thực hiện nó.



Hình 2.22: Ô trạng thái mới [16].

Ta nhân trạng thái cũ với f_t tương ứng để quên những thứ quyết định được phép quên trước. Sau đó cộng với $i_t * \tilde{C}_t$, đây là các giá trị ứng cử mới, được chia tỷ lệ theo mức độ

mà hệ thống quyết định cập nhật từng giá trị trạng thái. Cuối cùng cần quyết định xem đầu những gì sẽ được trả về. Kết quả ở đầu ra sẽ dựa trên ô trạng thái, nhưng sẽ là một phiên bản được lọc. Đầu tiên, chúng sẽ đi qua một tầng sigmoid nơi quyết định phần nào của ô trạng thái sẽ ở đầu ra. Sau đó, ô trạng thái được đưa qua hàm tanh (để chuyển giá trị về khoảng -1 và 1) và nhân nó với đầu ra của một cỗng sigmoid, do đó chỉ trả ra phần mà chúng ta quyết định.



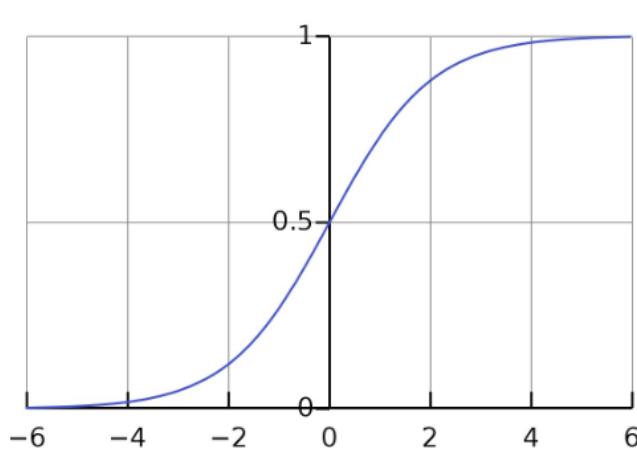
Hình 2.23: Cảnh thông tin ở đầu ra thông qua hàm tanh [16].

2.4 Sigmoid Function

Hàm **Sigmoid** là một hàm toán học đặc biệt có đường cong hình chữ S thể hiện cho sự biến đổi giá trị trong khoảng 0 đến 1. Sigmoid là một hàm kích hoạt (activation function) được sử dụng rộng rãi nhất là trong máy học (machine learning).

Tất cả các hàm Sigmoid đều có đặc điểm chung là chúng có thể chuyển những con số đầu vào thành một phạm vi nhỏ nhất định. Cụ thể chính là 0 đến 1 hoặc -1 và 1. Nghĩa là, hàm sigmoid dùng để chuyển một giá trị thực thành một giá trị kiểu xác suất. Hàm Sigmoid sẽ nhận những con số đầu vào (input) và thực hiện những công việc sau:

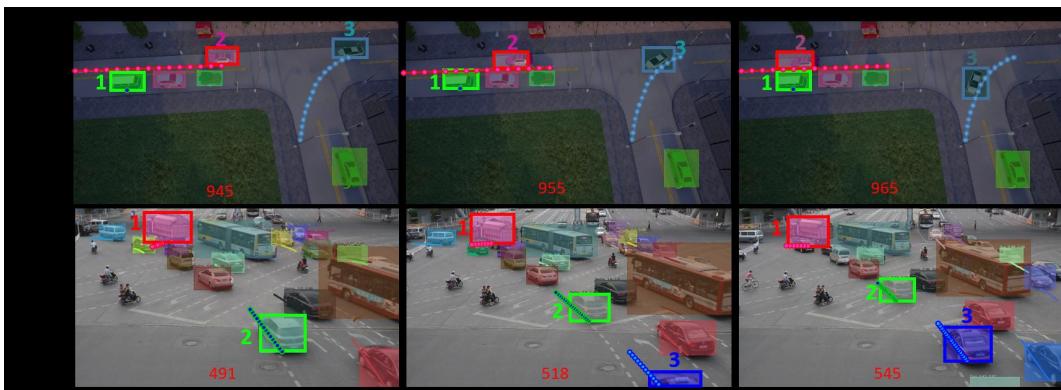
- Nếu số đầu vào âm, hàm Sigmoid sẽ chuyển tất cả thành một số gần với 0.
- Nếu đầu vào dương, hàm Sigmoid sẽ biến đầu vào thành một số gần với 1.
- Nếu đầu vào tương đối gần 0, hàm Sigmoid sẽ chuyển thành số bất kỳ từ 0 đến 1.



Hình 2.24: Hàm Sigmoid [26].

2.5 Bài toán Human pose estimation

Theo dõi vật thể (**object tracking**) là một trong những mảng được quan tâm nhất trong ngành thị giác máy (computer vision). Nhiệm vụ của bài toán này là từ video đầu vào, theo dõi sự di chuyển của đối tượng theo thời gian. Để theo dõi được một đối tượng trước hết mô hình cần phải phát hiện được đối tượng đó (**object detection**) trong từng frame sau đó sẽ cho qua một thuật toán về tracking. Do đó object tracking và object detection có mối quan hệ rất chắc chắn với nhau. Việc theo dõi đối tượng có chính xác hay không phụ thuộc rất nhiều vào việc phát hiện đối tượng tốt đến đâu.



Hình 2.25: Một ví dụ về object tracking [19]

Ước lượng tư thế người(**human pose estimation**) là một trong những đề tài của theo dõi vật thể. Nó được áp dụng để xử lý rất nhiều bài toán như nhận dạng ngôn ngữ ký hiệu, nhận dạng cử chỉ của cơ thể. Có khá nhiều phương pháp để xử lý bài toán ước lượng tư thế người. Trong đó ý tưởng phổ biến nhất là từ video đầu vào, tạo bản đồ nhiệt (bản đồ thể hiện sự thay đổi) cho từng tọa độ điểm dọc theo cơ thể người. Thông qua sự thay đổi các tọa độ điểm đó để xác định tư thế người.

Có thể thấy việc ước lượng tư thế người một cách chính xác là một thách thức khá lớn. Tư thế người có rất nhiều kiểu, và có những tư thế làm cho các điểm được xác định trên cơ thể biến mất hoặc chồng lấn lên nhau. Tuy nhiên với sự phát triển trong những năm gần đây của ngành trí tuệ nhân tạo nói chung và thị giác máy nói riêng, nhiều model giải quyết bài toán xác định tư thế người đã ra đời. Có thể kể đến như: **Mediapipe Pose** [12], **OpenPipaf** [21], **Alphapose**[6],..

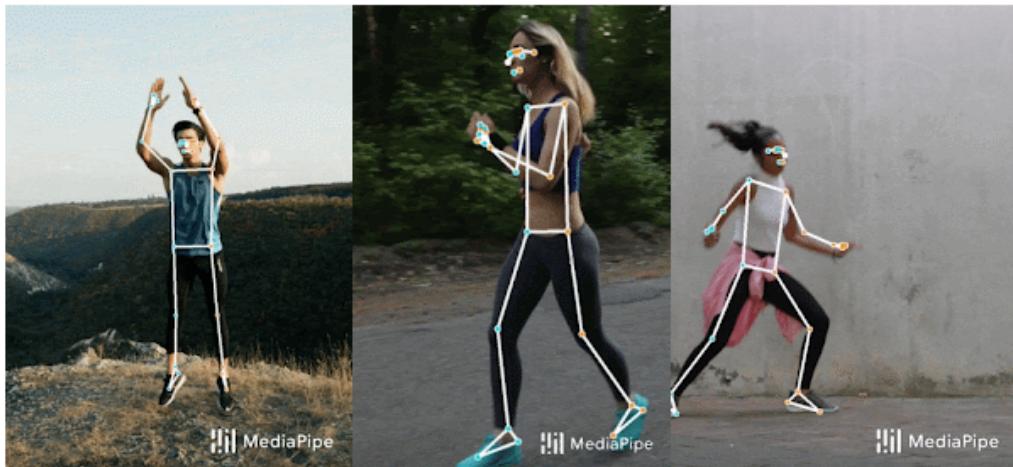


Hình 2.26: Ước lượng tư thế người trong các hoạt động hàng ngày [6]



Hình 2.27: Model OpenPiPaf [21]

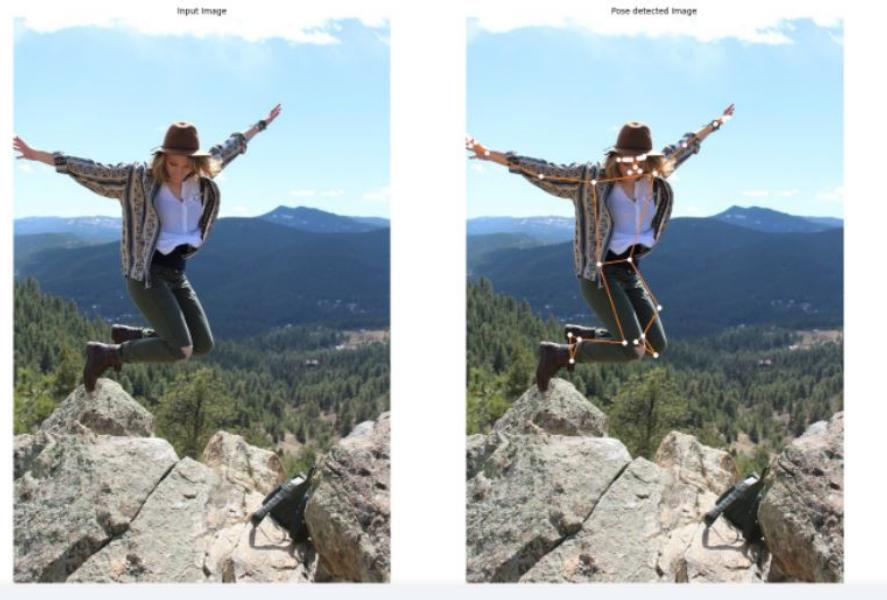
Phát hiện tẽ ngã cũng là một trong những ứng dụng của xác định tư thế người. Trong đề tài này chúng em sẽ sử dụng model **Mediapipe Pose** [12] để thực hiện. Do đó, để hiểu rõ hơn về model này, chúng ta sẽ đi tìm hiểu nó ở phần 2.6.



Hình 2.28: Model Mediapipe Pose [12]

2.6 Mediapipe Pose

Mediapipe Pose là một mô hình theo dõi tư thế người với độ trung thực cao. Ưu điểm của Mediapipe Pose so với những model khác là việc có thể hoạt động với những nhu cầu phần cứng không quá lớn như điện thoại, máy tính bàn, laptop, ... Đặc biệt là Mediapipe Pose có thể hoạt động real-time. Đầu vào là các frame RGB từ ảnh hoặc video, model sẽ cho ra kết quả là 33 điểm mốc 3D và mặt nạ phân vùng nền trên toàn bộ cơ thể người.

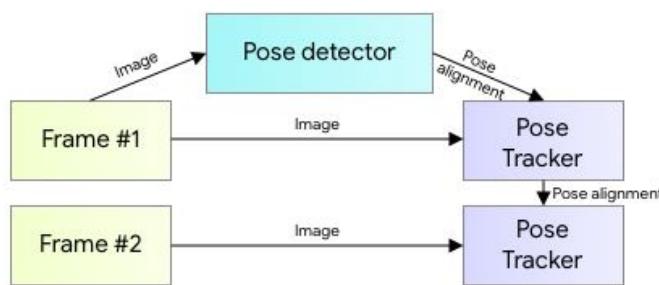


Hình 2.29: Kết quả khi sử dụng model Mediapipe Pose [12]

Vậy làm thế nào để model Mediapipe Pose có thể làm được điều đó, chúng ta sẽ đi tìm hiểu ở phần tiếp theo.

2.6.1 Pipeline của model Mediapipe Pose

Mediapipe Pose được xây dựng từ phương pháp ước tính tư thế con người BlazePose [1]. Để ước tính tư thế người, MediaPipe Pose sử dụng một pipeline bao gồm nhiều bộ chức năng hoạt động cùng nhau theo từng bước. Đầu tiên sử dụng một bộ xác định vị trí vùng quan tâm (ROI) của cơ thể người (Pose detector) trong frame. Sau đó bộ theo dõi (Pose tracker) dự đoán tất cả 33 điểm quan trọng từ vùng ROI ở trên. Điểm đặc biệt ở pipeline này là với những đầu vào là các video, bộ xác định vị trí vùng quan tâm (ROI) của cơ thể người chỉ hoạt động ở frame đầu tiên. Đối với các frame tiếp theo, vùng quan tâm này sẽ được suy ra từ 33 điểm chính đã được xác định ở frame trước đó. Việc xác định được vị vị trí vùng quan tâm (ROI) trong frame có vai trò rất quan trọng đến kết quả đầu ra. Vì việc theo dõi tư thế người có chính xác hay không phụ thuộc rất nhiều vào việc phát hiện vùng ROI trong frame tốt đến đâu.



Hình 2.30: Pipeline của model Mediapipe Pose [1]

Có thể quan sát được trong 2.30 có trong pipeline của model mediapipe pose có 2 model quan trọng:

- Pose detector: Model xác định vị trí vùng quan tâm (ROI) của người từ frame đầu vào.
- Pose tracker: Dự đoán 33 điểm dọc theo cơ thể người.

Với mục đích rằng Mediapipe Pose có thể hoạt động trong thời gian thực, mỗi thành phần trong pipeline trên cần phải xử lý với tốc độ cao (vài mili giây cho mỗi khung hình). Để hiểu rõ hơn về cấu trúc và cách thực hiện của mô hình, chúng ta sẽ đi tìm hiểu từng bộ chức năng thành phần.

2.6.2 Model pose detection cho bộ Pose detector

Với mục đích Mediapipe có thể hoạt động real-time, bộ Pose detector cần phải có tốc độ nhanh nhưng độ chính xác cũng phải tốt. Vì bộ Pose detector sẽ quyết định phần lớn độ chính xác của kết quả đầu ra. Với những mục tiêu vậy, các nhà nghiên cứu Google [2] phát triển một model phát hiện khuôn mặt người thông qua một model phát hiện khuôn mặt người (Face detector). Vì họ nhận ra rằng cách nhanh nhất, dễ nhất để một mô hình mạng học sâu phát hiện cơ thể người là phát hiện khuôn mặt người trước. Vì khuôn mặt người có

các tính chất với độ tương phản cao, ít các biến thể hơn so với ngoại hình của con người. Từ khuôn mặt người vừa xác định được sẽ đi xác định vùng ROI của cơ thể người.

Với ý tưởng như trên, Mediapipe Pose đào tạo một model phát hiện khuôn mặt (face detection), thông qua kết quả model này để phát hiện tư thế người. Model phát hiện khuôn mặt được xây dựng với cảm hứng từ mô hình Blaze Face [3]. Mô hình phát hiện khuôn mặt người dựa trên ảnh đầu vào.

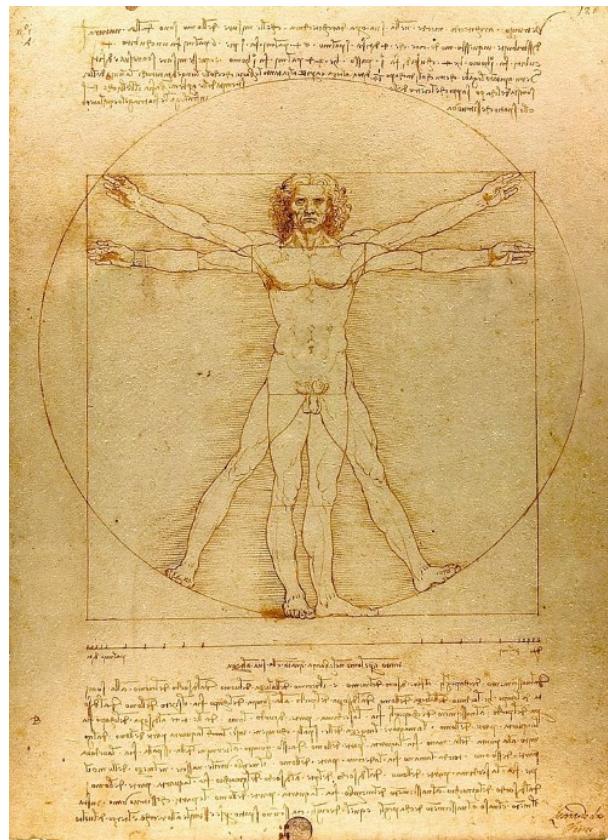


Hình 2.31: Phát hiện khuôn mặt bằng model Blaze face [3]

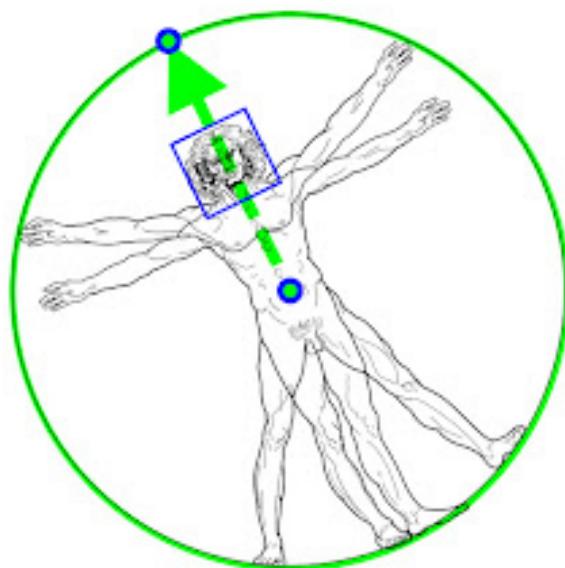
Sau khi phát hiện được khuôn mặt như ở hình 2.31, bộ pose detector sẽ sử dụng một số thuật toán để phát hiện vùng quan tâm (ROI) của toàn bộ cơ thể người. Dựa theo ý tưởng về bức họa của Vitruvian Man [5] của Leonardo Da Vinci, từ khuôn mặt được phát hiện được, mediapipe pose ước lượng được 2 điểm phụ mà từ hai điểm này có thể vẽ một đường tròn ngoại tiếp toàn bộ cơ thể người.

Hình 2.33 là cách Mediapipe xác định vùng ROI cho cơ thể người. Có thể thấy từ khuôn mặt sau khi xác định được thông qua model Blaze Face, mô hình sẽ ước lượng 2 điểm phụ với một điểm nằm ở ngay rốn cơ thể con người và một điểm cách đỉnh đầu một khoảng. Từ hai điểm này vẽ đường tròn ngoại tiếp cơ thể người với tâm là điểm ở rốn và bán kính là khoảng cách hai giữa hai điểm trên. Đường tròn này có ý nghĩa rằng mọi tư thế của con người khi thực hiện các tư thế như xoay, dang tay, dang chân, ... đều nằm gói gọn trong vòng tròn này. Điều này dẫn đến việc theo dõi nhất quán cơ thể người ngay cả đối với những trường hợp rất phức tạp, chẳng hạn như các tư thế yoga cụ thể.

Bộ Pose detector chỉ hoạt động khi Mediapipe Pose chưa xây dựng được cấu trúc liên kết của các điểm dọc theo cơ thể người. Nghĩa là từ video đầu vào, nó hoạt động cho đến khi Mediapipe Pose xác định được 33 điểm mốc dọc theo cơ thể rồi dừng. Khi 33 điểm này mất đi thì bộ Pose detector mới quay trở lại hoạt động lại.



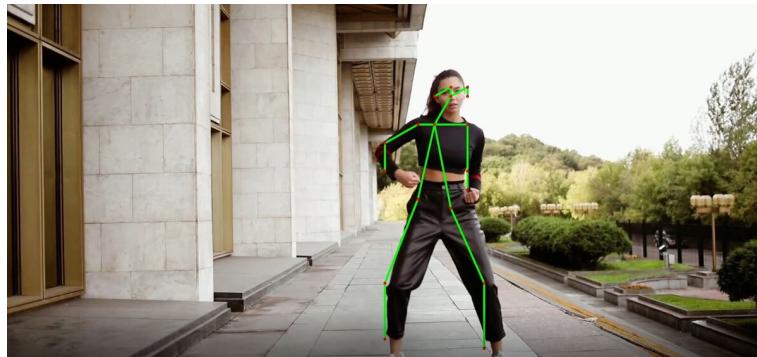
Hình 2.32: Bức họa Vitruvian Man [5]



Hình 2.33: Vùng ROI toàn bộ cơ thể người [2]

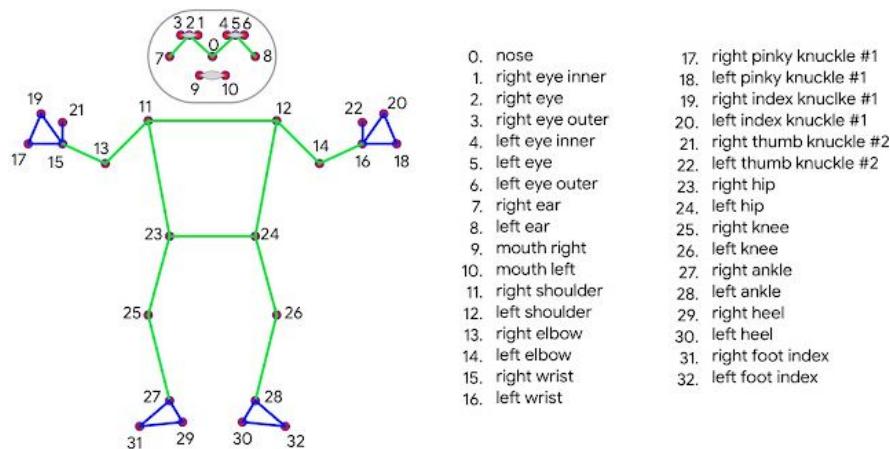
2.6.3 Tracking model cho bộ Pose Tracker

Pose tracker là bộ chức năng được sử dụng sau bộ Pose detector. Nó có chức năng dự đoán 33 điểm mốc dọc theo cơ thể người như trong hình 2.35 . Mỗi điểm sẽ có 3 thành phần là tọa



Hình 2.34: Mediapipe khi xác định được các điểm mốc [12]

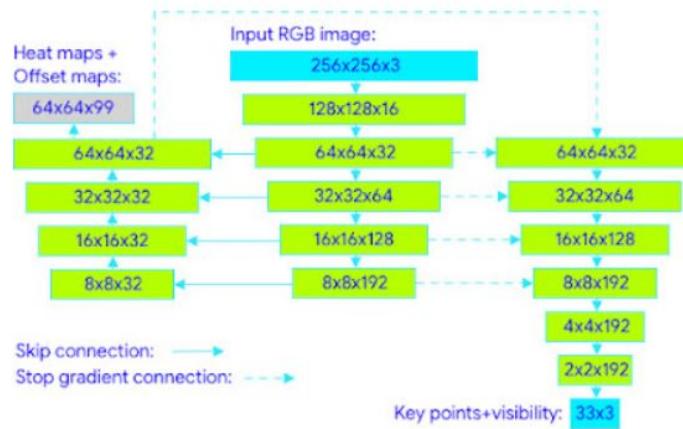
độ (x,y) và mức độ hiển thị. Ngoài ra còn dự đoán 2 điểm phụ đã xác định được từ bộ Pose detector ở phần 2.6.2.



Hình 2.35: Sơ đồ 33 điểm mốc [2]

Để dự đoán 33 điểm mốc như hình 2.35, có rất nhiều phương pháp. Các phương pháp cũ sẽ dựa vào bản đồ nhiệt (heatmap) của từng điểm để tính toán và suy ra tọa độ chính xác những điểm trên. Với mediapipe pose, mô hình của sử dụng áp dụng kết hợp bản đồ nhiệt (heatmap), độ dời (offset) và hồi quy (regression) để làm việc đó.

Hình 2.36 thể hiện kiến trúc model Tracking trong mediapipepose. Cụ thể, trong quá trình đào tạo, đầu tiên áp dụng model Heat maps kết hợp Offset map của Rui Zhang [29] để đào tạo phần trung tâm và bên trái của mạng. Sau đó, loại bỏ các đầu ra chỉ để lại phần cuối của model Heats map để đào tạo bộ mã hóa hồi quy (regression encode) nằm ở phần bên phải kiến trúc. Việc này giống như việc ta thực hiện nhúng nhẹ ảnh đầu qua model Heat maps để đưa vào phần mã hóa hồi quy ở bên phải. Phương pháp này lấy cảm hứng từ Newell [13]. Model mã hóa hồi quy được train và sẽ cho ra kết quả là 33 điểm có cấu trúc như hình 2.35. Kiến trúc trong hình 2.36 cũng tích cực sử dụng các skip-connection giữa tất cả các lớp mạng để đạt được sự cân bằng giữa các tính năng cấp thấp cao cấp. Tuy nhiên, các gradient từ bộ mã hóa hồi quy không được truyền ngược trở về phần model heatmap offset trong quá trình training.



Hình 2.36: Kiến trúc model Pose tracking cho pose tracker [2]

2.7 Những phần mềm và thư viện áp dụng trong luận văn

2.7.1 Ngôn ngữ lập trình Python

Python, là một ngôn ngữ lập trình thông dịch bậc cao (hight-level), hướng đối tượng (OOP). Với tính linh hoạt và tiện dụng, python cho phép người dùng làm nhiều việc, từ đơn giản đến phức tạp.



Hình 2.37: Ngôn ngữ lập trình python.[11]

Những năm gần đây, python ngày càng được người dùng ưu tiên sử dụng vì có những tính năng và ưu điểm sau:

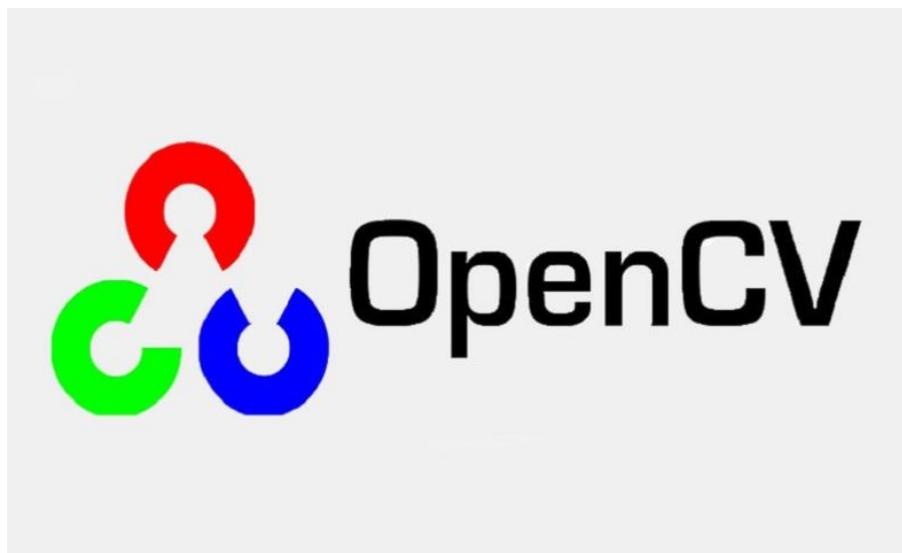
- Dễ học và sử dụng: Python có ít từ khóa, cấu trúc đơn giản và cú pháp được định nghĩa rõ ràng. Điều này cho phép người mới học tiếp cận ngôn ngữ một cách nhanh chóng.
- Vừa hướng thủ tục vừa hướng đối tượng: Mọi thứ trong Python đều là hướng đối tượng. Lập trình hướng đối tượng (OOP) giúp giải quyết những vấn đề phức tạp một cách trực quan.

- Có chức năng xử lý lỗi: Python hỗ trợ bắt lỗi và xử lý lỗi bằng ngoại lệ (exception).
- Hỗ trợ các module và package: Python cho phép người dùng vừa sử dụng các module, package từ nguồn khác. Vừa cho phép người dùng tạo module, package dễ dàng.

Với những lợi ích như vậy, chúng em sẽ sử dụng Python là ngôn ngữ lập trình chính và các module và package đính kèm để thực hiện đề tài luận văn này.

2.7.2 Opencv

OpenCV là tên viết tắt của open source computer vision library, có thể được hiểu là một thư viện nguồn mở cho máy tính.[22]. Opencv là nơi lưu trữ các mã nguồn mở được dùng để xử lý hình ảnh, video và phát triển các ứng dụng đồ họa trong thời gian thực.

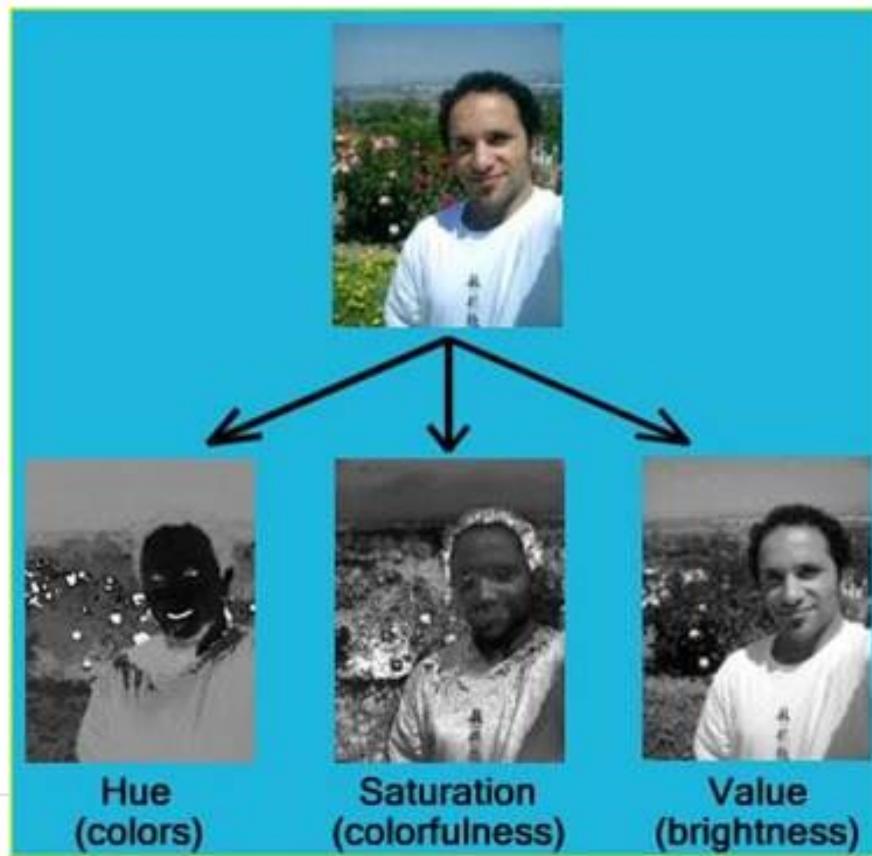


Hình 2.38: Opencv.[22]

OpenCV có một cộng đồng người dùng khá hùng hậu hoạt động trên khắp thế giới. Vì xu hướng chạy sử dụng computer vision của các công ty công nghệ nên nhu cầu cần đến nó ngày càng tăng theo. Lý do opencv được ưa chuộng trong các bài toán xử lý ảnh là vì nó có nhiều tính năng như :

- Xử lý và hiển thị hình ảnh, video
- Phát hiện vật thể.
- Khả năng xử lý tính toán trên các ảnh, video.
- Có thể dùng với cuda (GPU).

Vì có nhiều tính năng như trên nêu trên hỗ trợ người dùng xử lý nhiều bài toán từ đơn giản đến phức tạp. Do đó nó có rất nhiều ứng dụng như: phân tích hình ảnh y học, tìm kiếm và phục hồi hình ảnh/video, phim – cấu trúc 3D từ chuyên động, xác định vật thể,...



Hình 2.39: Xử lý ảnh trong opencv.[9]

Opencv có trên các ngôn ngữ lập C++, C, Python và Java. Nó hỗ trợ các hệ điều hành Windows, Linux, Mac OS, iOS và Android. OpenCV được thiết kế để hỗ trợ hiệu quả về tính toán và chuyên dùng cho các ứng dụng real-time (thời gian thực). Nếu được viết trên C/C++ tối ưu.

Với những tính năng và ứng dụng như trên, trong luận văn này với việc xử lý các ảnh được trích xuất từ camerra, chúng em sẽ sử dụng thư viện OpenCV với ngôn ngữ lập trình Python.

2.7.3 Tensorflow

TensorFlow chính là thư viện mã nguồn mở được phát triển bởi các nhà nghiên cứu Google. Đây là thư viện hỗ trợ mạnh mẽ các phép toán học để tính toán trong machine learning và deep learning.

Lý do Tensorflow được lựa chọn để sử dụng cho các ứng dụng trong machine learning cũng như deep learning là nó giúp việc tiếp cận các bài toán trở nên đơn giản, nhanh chóng và tiện lợi. Người dùng có thể dễ dàng xây dựng các mô hình học sâu từ đơn giản đến phức tạp. Ngoài ra, Tensorflow còn hỗ trợ việc trực quan các mô hình. Các thành phần chính trong Tensorflow bao gồm:

- Tensor: Trong TensorFlow, tất cả các tính toán đều liên quan tới các tensor. 1 tensor là 1 vector hay ma trận của n-chiều không gian đại diện cho tất cả loại dữ liệu. Tất cả giá trị



Hình 2.40: Thư viện Tensorflow.

trong 1 tensor chứa đựng loại dữ liệu giống hệt nhau với 1 shape đã biết (hoặc đã biết 1 phần). Shape của dữ liệu chính là chiều của ma trận hay mảng. Một tensor có thể được bắt nguồn từ dữ liệu input hay kết quả của 1 tính toán. . Mỗi operation được gọi là 1 op node (operation node) và được kết nối với nhau.

- Graph: TensorFlow sử dụng framework dạng graph - biểu đồ. Trong thư viện này, tất cả các hoạt động được tiến hành bên trong 1 biểu đồ. Biểu đồ là 1 tập hợp tính toán được diễn ra liên tiếp. Nó cũng tập hợp và mô tả tất cả các chuỗi tính toán được thực hiện trong quá trình training. Nhờ vậy mà người dùng có thể xem các luồng tính toán trong lúc xây dựng và training model.

Để xây dựng một mô hình học sâu để phân loại các chuỗi hành động có phải là té ngã hay không trong phương pháp B, chúng em sẽ lựa chọn Tensorflow là thư viện hỗ trợ giúp hiện thực công việc này.

2.7.4 Flask

Flask là một micro-framework được xây dựng bằng ngôn ngữ lập trình Python. Nó cho phép người dùng xây dựng các ứng dụng web từ đơn giản tới phức tạp.

Lí do Flask là một microframework bởi không yêu cầu tool hay thư viện cụ thể nào. Nó cung cấp một lõi chức năng “súc tích” nhất cho ứng dụng web nhưng người dùng cũng có thể mở rộng bất cứ lúc nào nhờ vào việc tích hợp các tiện ích khác. Flask được xây dựng dựa theo các nền tảng là:

- Web Server Gateway Interface (Giao diện cổng vào máy chủ Web): Thường được viết tắt là WSGI. WSGI được sử dụng như một tiêu chuẩn để phát triển ứng dụng web Python. WSGI mang đặc điểm kỹ thuật giao diện chung giữa máy chủ web và ứng dụng web.



Hình 2.41: Framework Flask. [7]

- Werkzeug: Là một bộ công cụ WSGI thực hiện các yêu cầu, phản hồi đối tượng, và các chức năng tiện ích. Điều này cho phép xây dựng một web framework trên đó. Flask sử dụng Werkzeug làm một trong những cơ sở xây dựng web của nó.
- Jinja2: Là một công cụ mẫu phổ biến cho Python. Một hệ thống mẫu web kết hợp một mẫu với một nguồn dữ liệu cụ thể để hiển thị một trang web động.

Nhờ Flask, chúng ta có thể xây dựng các API nhỏ, các ứng dụng web chẳng hạn như các trang web, blog, trang wiki hoặc một website dựa theo thời gian hay thậm chí là một trang web thương mại. Với những ưu điểm đó, trong đề tài này chúng em sẽ sử dụng Flask để xây dựng một ứng dụng web theo dõi và quản lý hệ thống phát hiện té ngã.

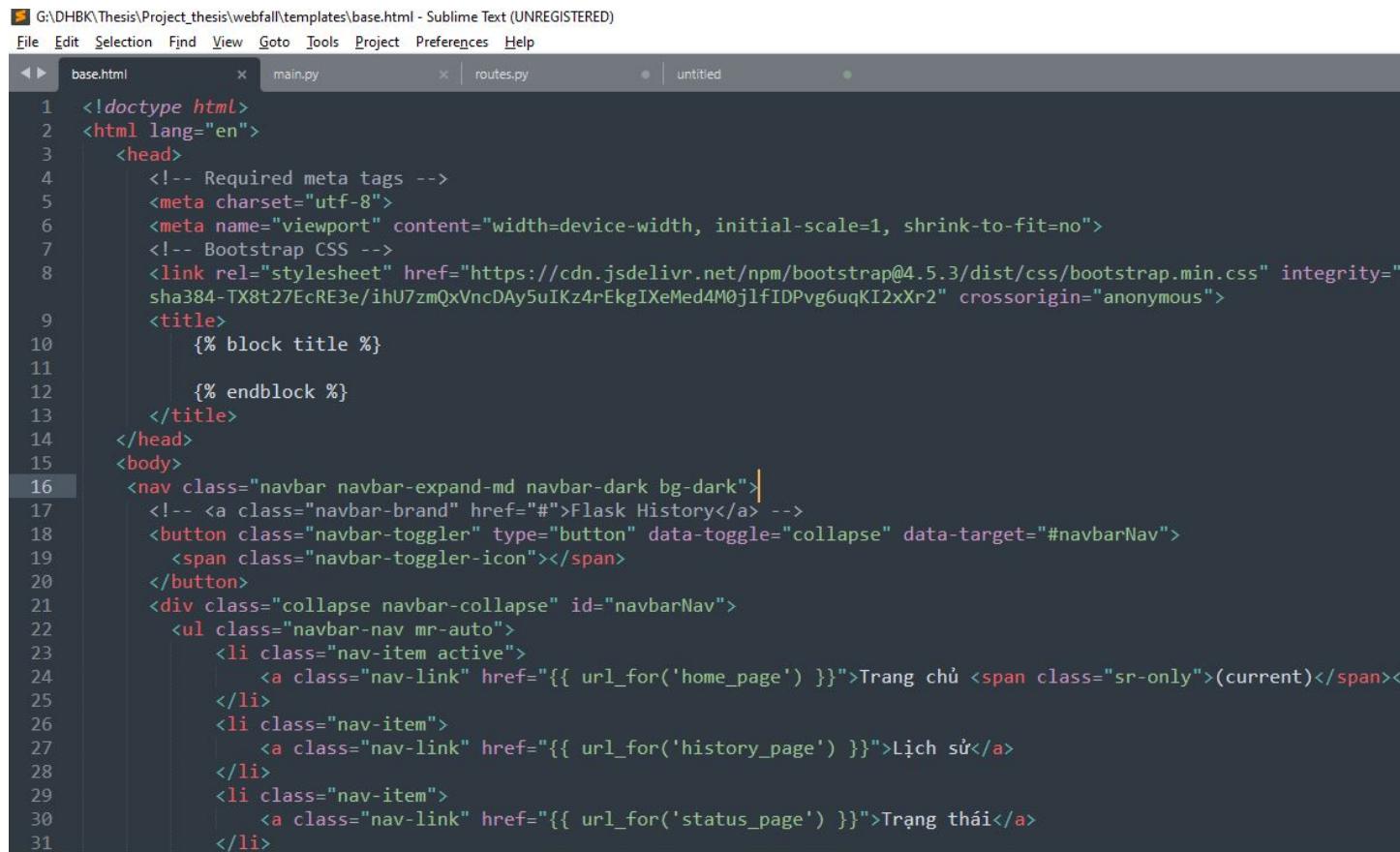
2.7.5 HTML

Khi chúng ta truy cập một trang web, những nội dung của trang web đó được hiện ra và theo các bố cục rõ ràng. Có bao giờ bạn tự hỏi rằng làm cách nào trang web đó lại có thể hiện ra các thông tin với các cách bố trí, trang trí và giao diện như vậy. Đó là nhờ vào HTML.

HTML là chữ viết tắt của Hypertext Markup Language (Ngôn ngữ đánh dấu siêu văn bản). Nó giúp người dùng tạo và cấu trúc các thành phần trong trang web hoặc ứng dụng. Vì mỗi website có thể chứa nhiều trang nội dung nên mỗi trang sẽ là một tài liệu HTML.

Lưu ý rằng, HTML không phải là một ngôn ngữ lập trình. Nó đóng vai trò giúp người dùng có thể định dạng, thiết kế cấu trúc các thành phần của một trang web hay các ứng dụng, heading, links, hoặc phân chia giữa các đoạn văn,... Nói cách khác thì HTML hoạt động như Microsoft Word, tức là chỉ dùng để định dạng và bố cục nội dung hiển thị trên trang web.

Một tài liệu HTML được hình thành bởi các phần tử HTML (HTML Elements) được quy định bằng các cặp thẻ (tag và attributes). Các cặp thẻ này được bao bọc bởi một dấu ngoặc nhọn (ví dụ <html>) và thường là sẽ được khai báo thành một cặp, bao gồm thẻ mở và thẻ đóng. Ví dụ, chúng ta có thể tạo một đoạn văn bằng cách đặt văn bản vào trong cặp tag mở



The screenshot shows a Sublime Text interface with three tabs: 'base.html' (active), 'main.py', and 'routes.py'. The 'base.html' tab contains the following code:

```
1 <!doctype html>
2 <html lang="en">
3     <head>
4         
5         <meta charset="utf-8">
6         <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
7         
8         <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqK12xXr2" crossorigin="anonymous">
9             <title>
10                 {% block title %}</title>
11             {% endblock %}
12         </head>
13     <body>
14         <nav class="navbar navbar-expand-md navbar-dark bg-dark">
15             <!-- <a class="navbar-brand" href="#">Flask History</a> -->
16             <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav">
17                 <span class="navbar-toggler-icon"></span>
18             </button>
19             <div class="collapse navbar-collapse" id="navbarNav">
20                 <ul class="navbar-nav mr-auto">
21                     <li class="nav-item active">
22                         <a class="nav-link" href="{{ url_for('home_page') }}>Trang chủ <span class="sr-only">(current)</span>
23                     </li>
24                     <li class="nav-item">
25                         <a class="nav-link" href="{{ url_for('history_page') }}>Lịch sử</a>
26                     </li>
27                     <li class="nav-item">
28                         <a class="nav-link" href="{{ url_for('status_page') }}>Trạng thái</a>
29                     </li>
30                 </ul>
31             </div>
32         </nav>
```

Hình 2.42: Một tệp ví dụ về HTML.

và đóng văn bản như sau <p> Hệ thống phát hiện té ngã </p>.

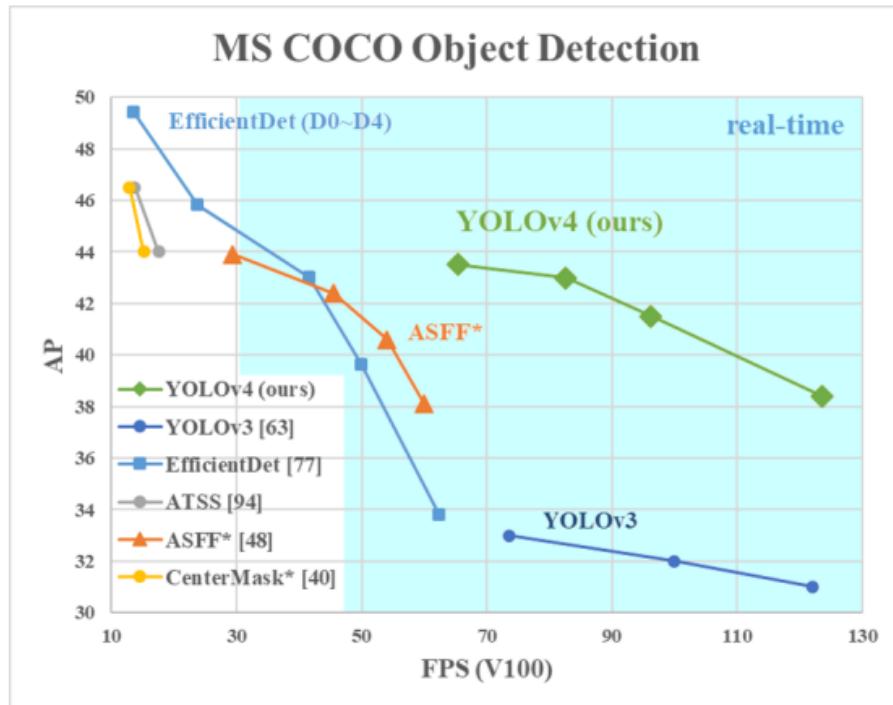
Trong luận văn này, chúng em sẽ sử dụng HTML để định dạng và thiết kế trang web quản lý hệ thống phát hiện té ngã của mình.

Chương 3

KẾT QUẢ VÀ PHÂN TÍCH

3.1 Phương pháp A: Nhận diện các tư thế sử dụng mô hình YOLOv4.

Do ý tưởng ban đầu là phần lớn trường hợp ngã xuất phát từ tư thế đứng hoặc đang di chuyển sau đó nằm dưới sàn. Song, phải phân biệt được giữa trường hợp té ngã và trường hợp nằm ngủ. Vì vậy nhóm đã sử dụng YOLOv4 để nhận dạng tư thế của từng khung hình (sẽ trình bày rõ hơn ở phần sơ đồ giải thuật). Mặc khác, sử dụng YOLOv4 để huấn luyện cho mô hình bởi vì phiên bản này có nhiều sự cải tiến đặc biệt giúp tăng độ chính xác và tốc độ hơn đối với các phiên bản trước, ví dụ như: đối với YOLOv3, chỉ số mAP và FPS của YOLOv4 tăng tương ứng 10% và 12%, tốc độ nhanh hơn 2 lần so với EfficientDet...[8].

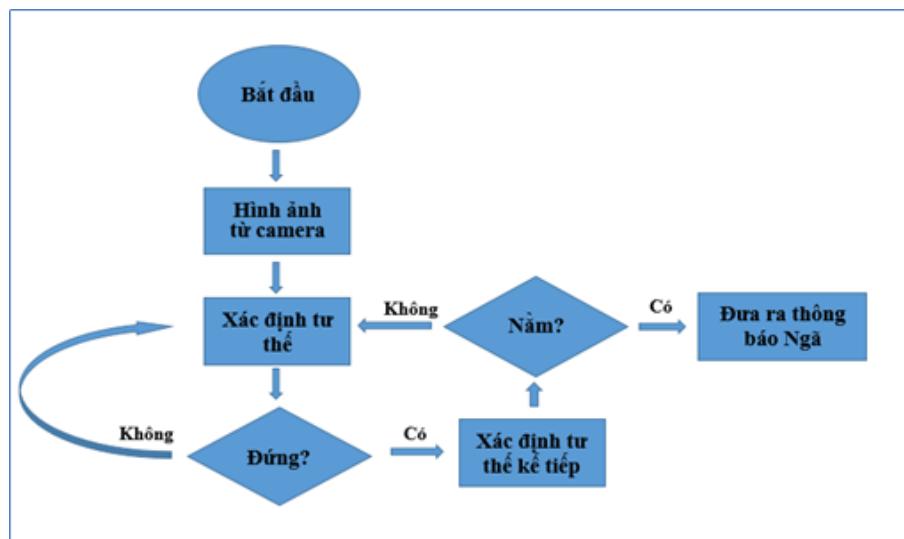


Hình 3.1: So sánh YOLOv4 với các model phát hiện đối tượng khác[8].

3.1.1 Sơ đồ giải thuật.

Vậy lúc này ta sử dụng model để nhận dạng 3 tư thế chính: Đứng – Ngồi – Nằm. Khi phát hiện được tư thế "Đứng" thì hệ thống sẽ xét khung hình ngay sau đó và nếu khung hình này là tư thế "Nằm" thì hệ thống sẽ thông báo là "Ngã". Đối với các trường hợp khác thì bình thường.

Ta có sơ đồ giải thuật như sau :



Hình 3.2: Sơ đồ giải thuật.

Quá trình thực hiện diễn ra như sau:

- **Bước 1:** Đọc từng khung hình từ camera sau đó đưa vào model dự đoán.
- **Bước 2:** Sử dụng model Yolov4 xác định đối tượng (ở đây là người) và tư thế trong khung hình đang ở tư thế nào trong khung hình bằng cách đưa ra output là chỉ số của bounding boxes như là tâm (x,y), chiều cao (height), chiều rộng (width) và kèm với nhãn của tư thế. Sau đó vẽ lên khung hình.
- **Bước 3:** Nếu model xác định được là tư thế đứng thì thực hiện bước tiếp theo, nếu là tư thế khác như đang ngồi, nằm thì quay lại bước 2.
- **Bước 4:** Xác định khung hình kế tiếp xem có phải là tư thế nằm hay không. Nếu đúng sẽ đưa ra thông báo ngã. Sau đó quay lại bước 2.

Ví dụ:

- + Khung hình 1: "Đứng" + Khung hình 2: "Nằm" = Thông báo "Ngã".
- + Khung hình 1: "Đứng" + Khung hình 2: "Ngồi" = Không có thông báo.
- + Khung hình 1: "Ngồi" hoặc "Nằm" thì sẽ không xét khung hình tiếp theo.

3.1.2 Sưu tầm dữ liệu và gán nhãn cho việc huấn luyện.

Thu thập được hơn 20 video về trường hợp té ngã [18] và mỗi trường hợp té ngã sẽ có 8 góc quay từ 8 camera được bố trí xung quanh phòng. Sau đó tự quay thêm dữ liệu bằng camera điện thoại tại phòng ngủ.

Thực hiện quan sát từng video và chụp lại các trường hợp có tư thế đứng, nằm và ngồi. Với tổng số lượng chụp được khoảng 1100 ảnh cho 3 tư thế: đứng(300), ngồi(460), nằm(340).

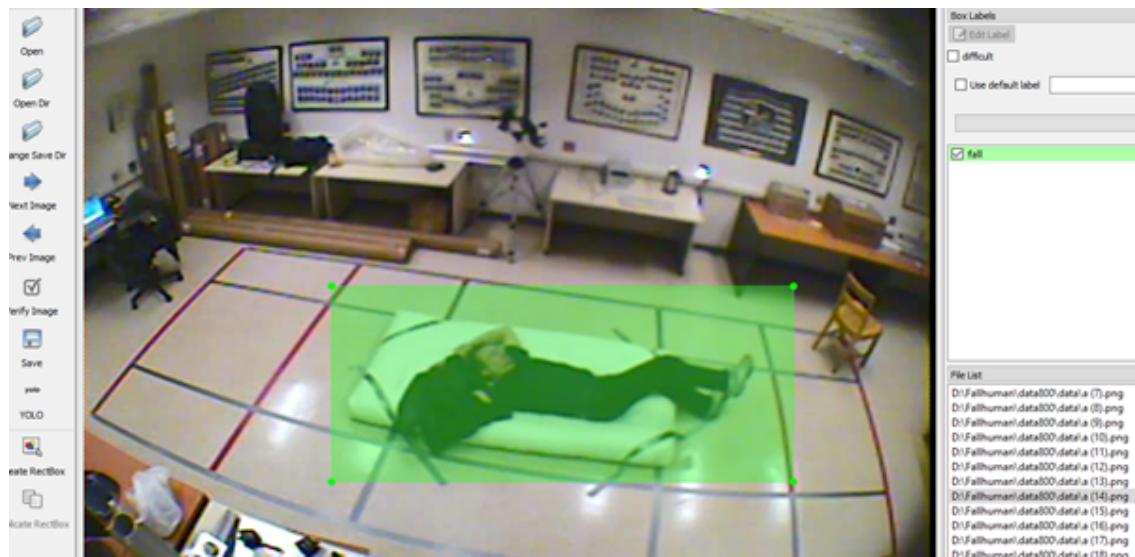


Hình 3.3: Ảnh sưu tầm [18].



Hình 3.4: Ảnh tự chụp.

Sau đó sử dụng phần mềm LabelImg để gán nhãn cho từng ảnh. Gồm có 3 lớp : đúng, nằm và ngồi [25].

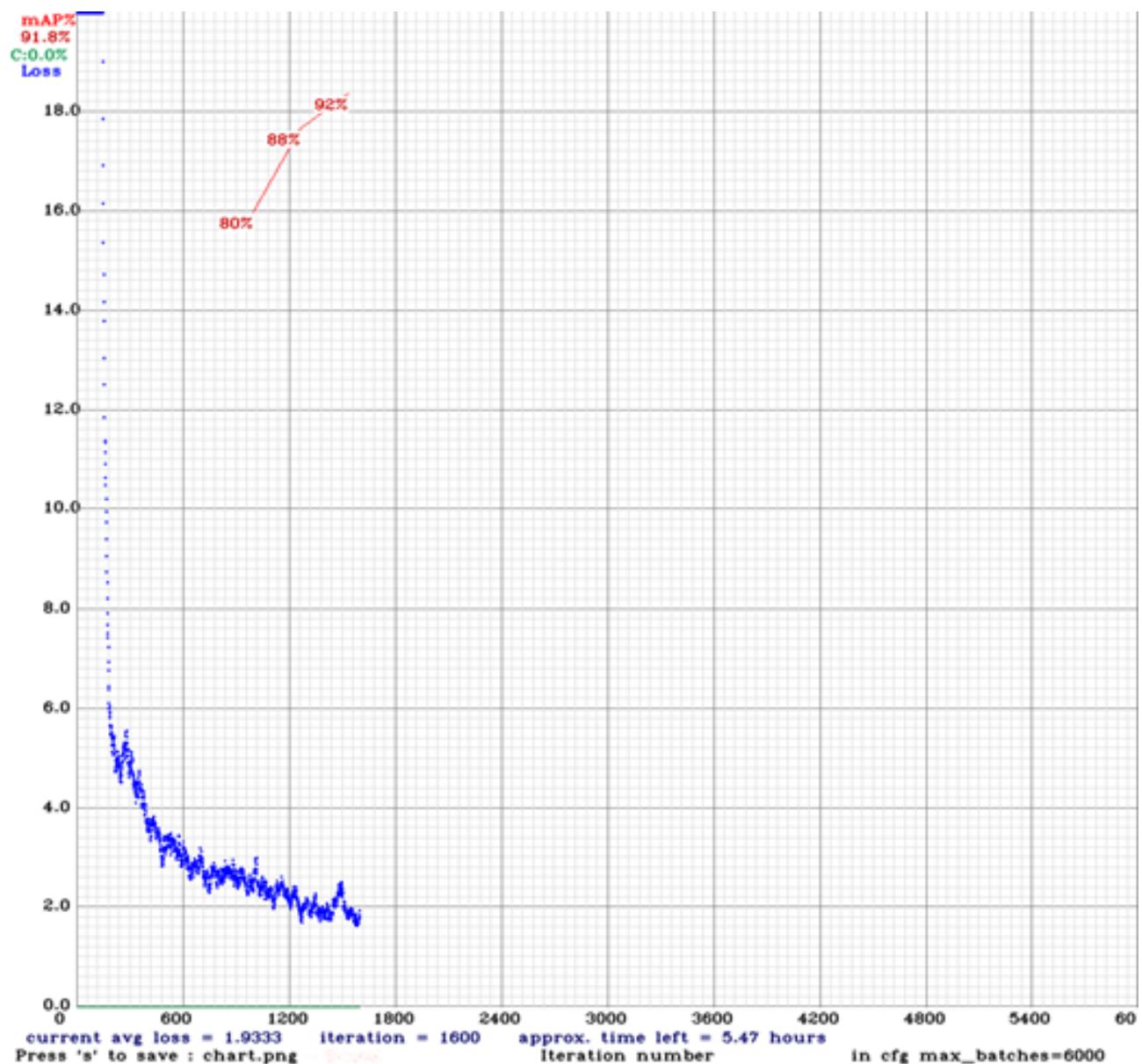


Hình 3.5: Gán nhãn sử dụng LabelImg.

3.1.3 Huấn luyện mô hình.

Ta bắt đầu thực hiện huấn luyện mô hình:

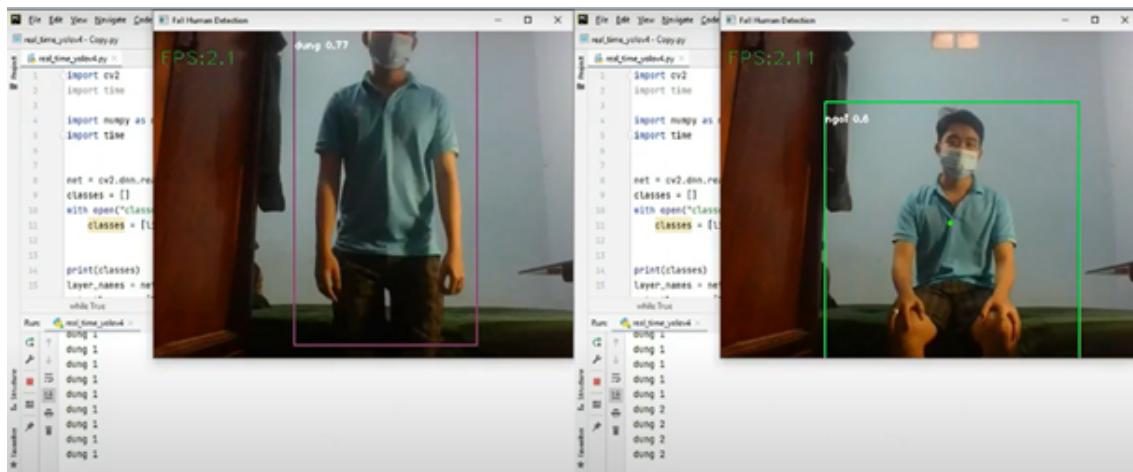
- + Số lượng ảnh lúc này: 1100 ảnh (3 class).
- + Tỉ lệ chia tập train,val lần lượt là 80%,20%.
- + Số vòng lặp: 2000 vòng.
- + Kích thước ảnh đầu vào: 416x416.
- + Batchsize: 32.
- + Thời gian huấn luyện: 6h



Hình 3.6: Chỉ số mAP, loss trong quá trình huấn luyện.

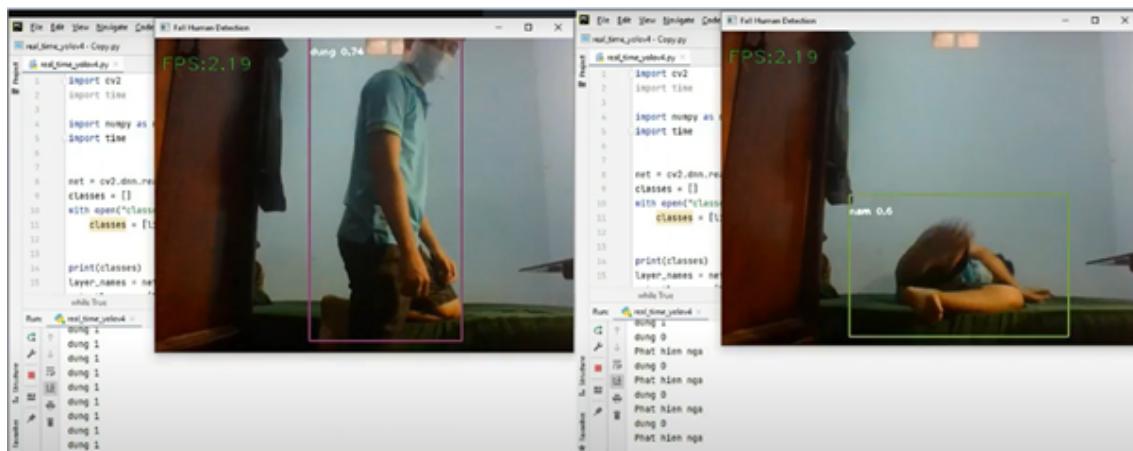
3.1.4 Kết quả và phân tích

Dưới đây là một số kết quả thu được sau khi đưa vào mô hình một số đoạn ghi hình để kiểm tra. Đầu tiên là trường hợp bình thường. Khung hình 1 là "Đứng" và khung hình 2 là "Ngồi"

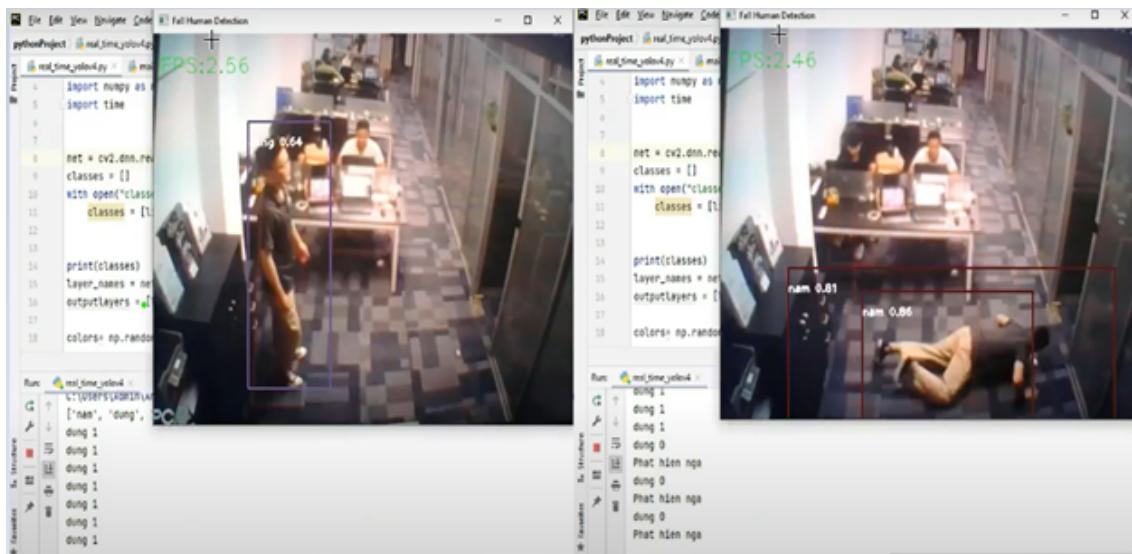


Hình 3.7: Trường hợp bình thường.

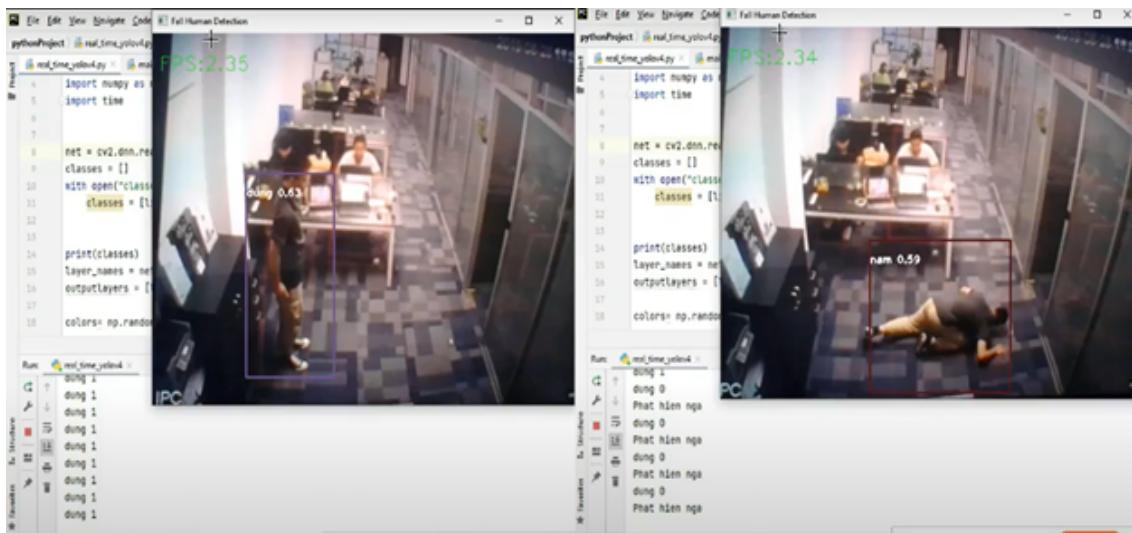
Tiếp đến là trường hợp té ngã 1,2 và 3. Dự đoán trường hợp té ngã trong 2 căn phòng khác nhau. Sau khi thỏa điều kiện : khung hình 1 là "Đứng" và khung hình 2 là "Nằm". Sau đó hệ thống đưa ra thông báo là " Phát hiện ngã"



Hình 3.8: Trường hợp té ngã 1.



Hình 3.9: Trường hợp té ngã 2.



Hình 3.10: Trường hợp té ngã 3.

Cuối cùng là trường hợp hệ thống phát hiện sai khi nhầm lẫn khung hình 2 là "Nàng" nhưng thực tế không phải như vậy.



Hình 3.11: Trường hợp phát hiện sai.

Từ kết quả của quá trình huấn luyện và kiểm tra cho thấy hệ thống đạt được chỉ số mAp cao, dự đoán khá chính xác các lớp vs độ tin cậy cao.

Tuy nhiên, quá trình huấn luyện mất quá nhiều thời gian (gần 6 giờ cho một lần huấn luyện), phải huấn luyện trên google colab do phần cứng laptop không đủ điều kiện và việc tinh chỉnh dữ liệu mất rất nhiều thời gian khi đưa dữ liệu lên google drive [23]. Hệ thống hoạt động chưa tốt với tốc độ khá chậm, chỉ hơn 2 khung hình trên giây (2.3 FPS), rất dễ bỏ lỡ chi tiết té ngã, chỉ phát hiện chính xác nhưng trường hợp đơn giản do ý tưởng ban đầu là việc té ngã bắt đầu từ tư thế đứng, di chuyển và dẫn đến nhầm lẫn khá nhiều trường hợp như ngã chậm, khụy gối, đang ngồi trên ghế sau đó ngã về phía trước hay ngã từ giường ngủ xuống đất.

Mặc khác, hệ thống đang vận hành trên phần cứng của laptop nhưng với tốc độ này sẽ càng khó khăn hơn rất nhiều khi triển khai lên phần cứng khác nhỏ gọn và ít kinh phí hơn. Vì vậy phương pháp này đường như bất khả thi và đòi hỏi một giải pháp tối ưu hơn.

3.2 Phương pháp B: sử dụng Mediapipe kết hợp LSTM

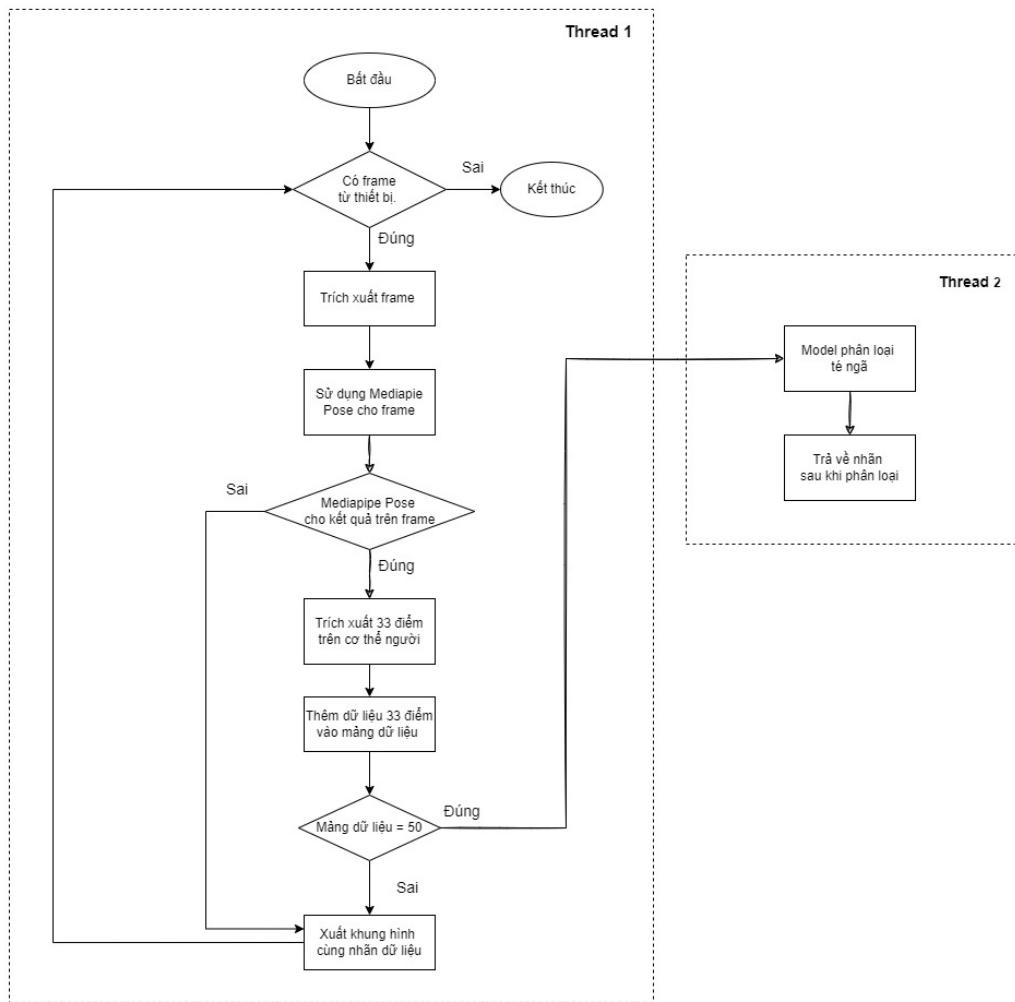
Các hành động của con người như đi, đứng, nằm, chạy, nhảy,... là việc thay đổi liên tục tư thế cơ thể. Do đó chúng ta có thể phân loại được các hành động trên bằng cách phân biệt các chuỗi tư thế liên tiếp mô tả sự thay đổi đấy.



Hình 3.12: Một trường hợp về chuỗi tư thế của một hành động ngã.

Phương pháp B này sẽ lấy ý tưởng trên để thực hiện. Có thể thấy chuỗi hành động ngã như hình 3.12 là sự thay đổi liên tiếp của các tư thế. Phương pháp này sẽ sử dụng Mediapipe Pose để xác định tư thế người và dùng một mô hình mạng LSTM để phân loại các hành động đọc được từ camera (hoặc webcam) có phải là hành động ngã hay không. Chúng ta sẽ đi tìm hiểu chi tiết ở những phần tiếp theo.

3.2.1 Sơ đồ giải thuật



Hình 3.13: Sơ đồ khối phương pháp B.

Hình 3.13 là sơ đồ khối mô tả chi tiết cách thực hiện của phương án B. Chúng ta sẽ đi tìm hiểu chi tiết các bước và các thành phần trong sơ đồ khối này.

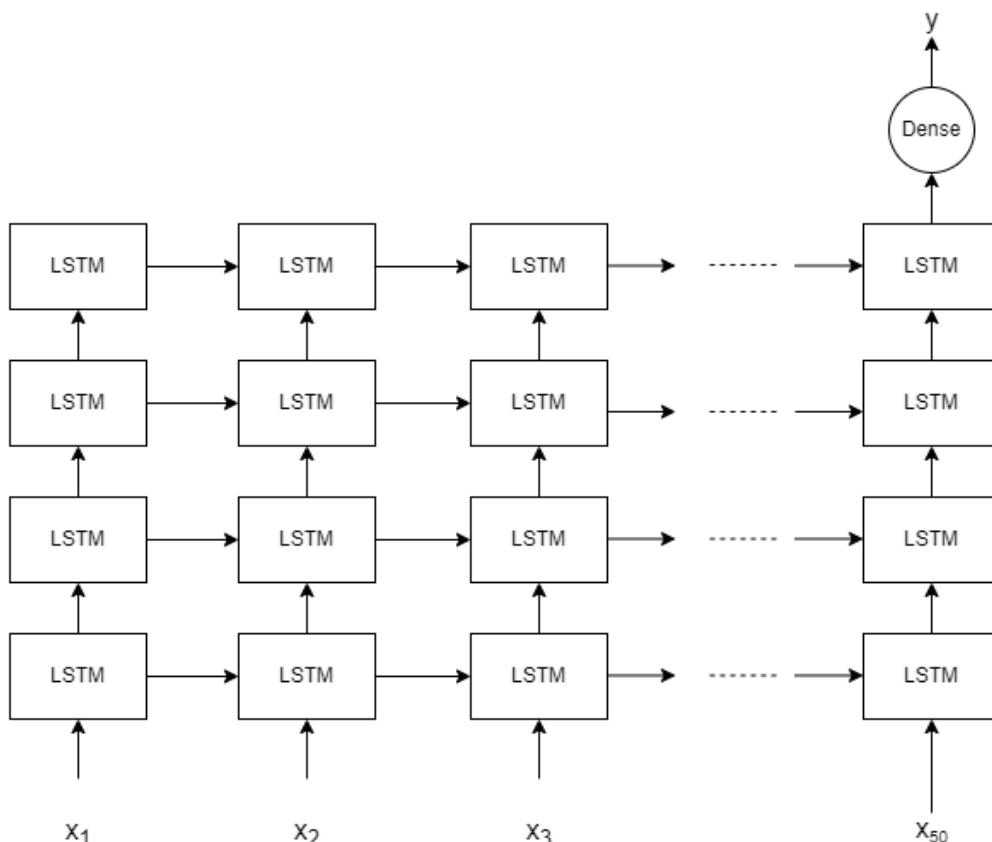
Hệ thống phát hiện té ngã của chúng ta sẽ chạy song song trên hai luồng (thread). Luồng thứ nhất có chức năng đọc các frame liên tục từ camera và trích xuất 33 điểm trên cơ thể người nếu có từ các frame đó. Luồng thứ 2 có tác dụng phân biệt chuỗi frame (có kích thước 50 frame) tương ứng cho chuỗi hoạt động liên tiếp của người có phải là té ngã hay không rồi trả về nhãn để xuất ra cùng khung hình cho người dùng.

Hệ thống sẽ hoạt động theo các bước như sau:

- **Bước 1:** Bắt đầu chương trình, khởi tạo nhãn ban đầu là "No Fall". Tạo mảng dữ liệu rỗng để với mỗi phần tử trong mảng là tọa độ 33 điểm mốc mô tả tư thế người của từng frame.
- **Bước 2:** Hệ thống kiểm tra xem có frame nào từ camera hay webcam hay không. Nếu có thì chuyển qua **bước 3**. Nếu không thì kết thúc chương trình (trường hợp này diễn ra khi đọc frame từ video).
- **Bước 3:** Trích xuất frame từ camera hoặc webcam và đưa frame qua Mediapipe Pose để xử lý.
- **Bước 4:** Kiểm tra Mediapipe Pose có trả kết quả trên frame không. Nghĩa là trên frame Mediapipe Pose có phát hiện được tư thế người hay không. Nếu không thì xuất khung hình cùng nhãn dữ liệu cho người dùng rồi quay lại **bước 2**. Nếu Mediapipe Pose phát hiện được tư thế người trên frame sẽ chuyển qua **bước 5**.
- **Bước 5:** Sau khi Mediapipe Pose xác định được tư thế người trên frame và cho ra tọa độ 33 điểm mốc trên cơ thể người, ta trích xuất 33 điểm đó rồi thêm vào mảng dữ liệu đã được tạo ở **bước 1**.
- **Bước 6:** Kiểm tra mảng dữ liệu đã đủ 50 phần tử tương ứng với 50 frame hay chưa. Nếu đủ rồi thì sẽ đưa vào model phân loại để xác định nhãn.
- **Bước 7:** Xuất khung hình cùng nhãn dữ liệu rồi quay lại **bước 2**.

3.2.2 Model phân loại té ngã.

Model phân loại té ngã là model có thể xác định được hành động đưa vào là ngã hay không. Do đó nó sẽ có hai nhãn là 1 (ứng với hành động ngã) và 0 (ứng với hành động không phải là ngã). Với mục tiêu hệ thống phải được hoạt động được trong thời gian thực với độ chính xác có thể chấp nhận, model phân loại té ngã phải là model nhẹ và thời gian hoạt động phải rất nhanh (mili giây). Sau khi thử nghiệm xây dựng các loại mô hình khác nhau, chúng em xin đề xuất sử dụng một mô hình gồm 4 lớp LSTM xếp chồng lên nhau và một lớp Dense ở cuối. Ngoài ra cũng thêm vào các lớp Dropout sau mỗi lớp LSTM để tránh model bị overfitting[24].

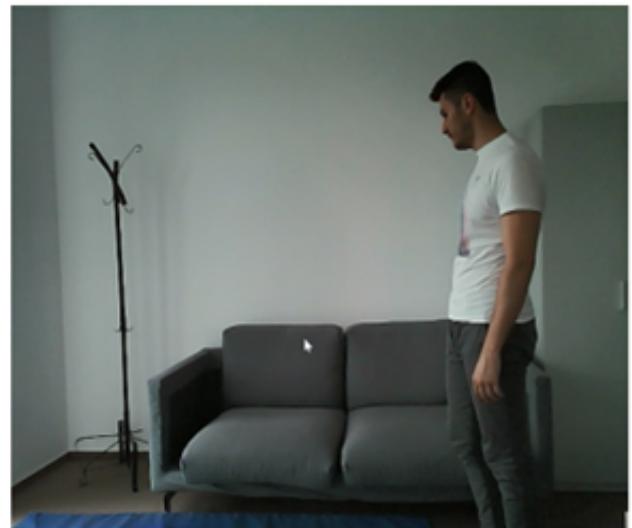
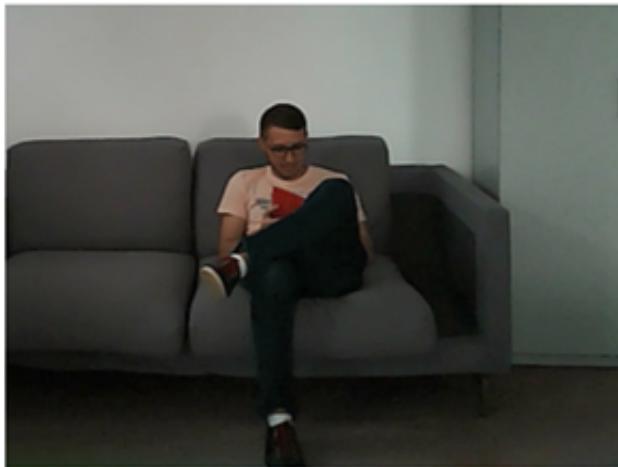


Hình 3.14: Model phân loại té ngã

Quan sát model ở hình 3.14, đầu vào của model là 50 điểm dữ liệu liên tiếp (từ x_1 đến x_{50}). Qua 4 lớp LSTM xếp chồng lên nhau (ngõ ra của lớp này là ngõ vào của lớp tiếp theo), lưu ý sau mỗi lớp LSTM đều có sử dụng một lớp Dropout để model tránh bị hiện tượng overfitting[24]. Sau khi qua các lớp LSTM, sẽ đi qua lớp cuối cùng là lớp Dense với hàm kích hoạt Sigmoid để cho ngõ ra của model phân loại là y . Với $y = 1$ thì hành động đầu vào là hành động té ngã, $y = 0$ model xác định hành động không phải là té ngã.

3.2.3 Sưu tầm dữ liệu

Số lượng dữ liệu hơn 1000 video. Tự quay khoảng 200 video và sưu tầm 800 video. Trong đó khoảng 350 video về té ngã và 700 video trưởng hợp bình thường, Tuy nhiên, sau quá trình chọn lọc và lấy mẫu thì còn lại 303 video về trưởng hợp té ngã và 566 video trưởng hợp bình thường. Cuối cùng là chia tập dữ liệu này thành 3 tập con là training, validation và testing theo tỉ lệ 70%, 20% và 10%.



Hình 3.15: Dữ liệu sưu tầm.

3.2.4 Lấy mẫu dữ liệu cho việc huấn luyện

Sử dụng thư viện Mediapipe để phát hiện khung xương người. Sau đó theo dõi và tìm ra thời điểm từ lúc sắp ngã đến lúc ngã để lấy mẫu.



Hình 3.16: Phát hiện khung xương.

Ta lấy tọa độ (x,y,z và visibility) các 32 điểm nút của 50 khung hình của trường hợp té ngã và không ngã vào file csv.

```
,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,3  
0,0.21566960215568542,0.5102907419204712,-0.1158093586564064,0.9996187090873718,0.  
1,0.21043367683887482,0.5095376372337341,-0.1550440490245819,0.9996339082717896,0.  
2,0.204297736287117,0.5086557269096375,-0.1636011004447937,0.9996485710144043,0.20  
3,0.20422665774822235,0.5066446661949158,-0.14304588735103607,0.9996579885482788,0  
4,0.2029283344745636,0.5048965811729431,-0.13192488253116608,0.999671459197998,0.2  
5,0.2017451822757721,0.5028430223464966,-0.1290769875049591,0.999686598777771,0.20  
6,0.19884243607521057,0.5026583075523376,-0.1460493952035904,0.9996904730796814,0.  
7,0.19744247198104858,0.5024811029434204,-0.1659678965806961,0.9996811747550964,0.  
8,0.1967579424381256,0.501403272151947,-0.12566769123077393,0.9996597170829773,0.1  
9,0.19443973898887634,0.501850426197052,-0.1782514601945877,0.9996618628501892,0.1  
10,0.1937936395406723,0.5022894144058228,-0.16093792021274567,0.9996199607849121,0  
11,0.1915711716710656,-0.50251007771697005,-0.1711117175257006,-0.9995921692772927
```

Hình 3.17: Tọa độ các điểm nút của 50 khung hình.

3.2.5 Sử dụng LSTM để huấn luyện mô hình

Dầu vào là 243 file csv của lớp fall và 538 file csv lớp nofall và tất cả đều có kích thước là 1x50x132.

```

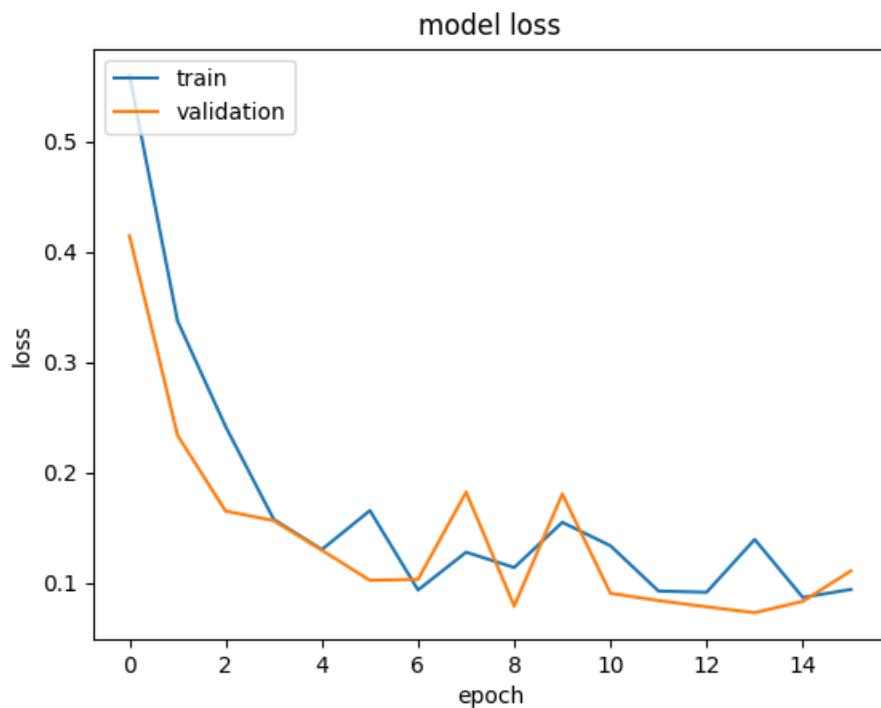
model = Sequential()
model.add(LSTM(units = 50, return_sequences = True, input_shape = (x.shape[1], x.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
model.add(LSTM(units = 50))
model.add(Dropout(0.2))
model.add(Dense(units = 1, activation="sigmoid"))
model.compile(optimizer="adam", metrics = ['accuracy'], loss = "binary_crossentropy")

history = model.fit(X_train, y_train, epochs=16, batch_size=32, validation_data=(X_test, y_test))

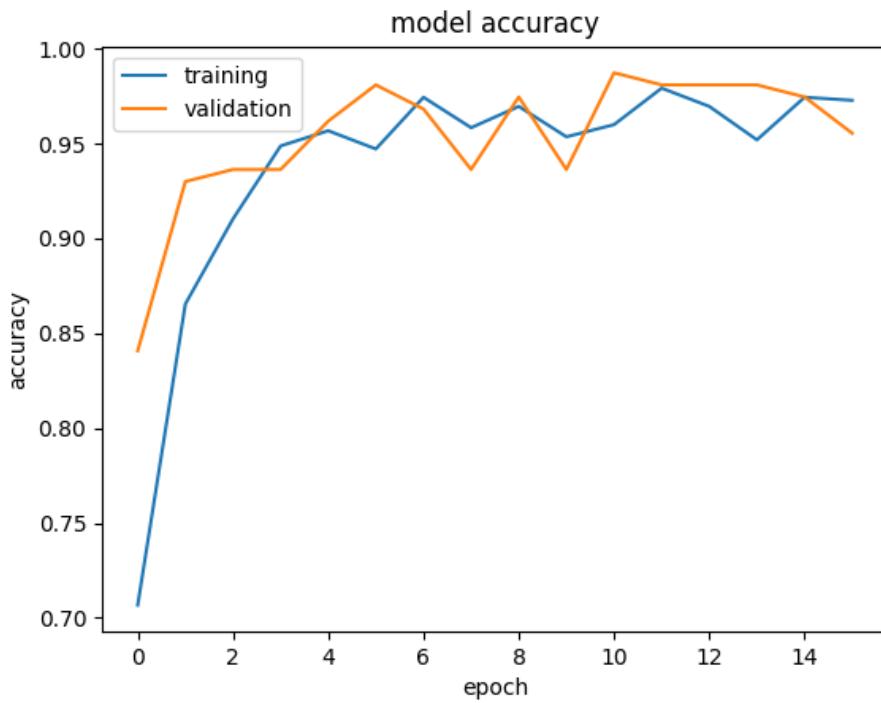
```

Hình 3.18: Mô hình huấn luyện LSTM.

Sau khi hoàn tất việc huấn luyện với thời gian khá nhanh(26 giây), ta có được chỉ số loss và accuracy mô hình qua từng epoch.



Hình 3.19: Chỉ số loss của mô hình



Hình 3.20: Chỉ số accuracy của mô hình

3.2.6 Kết quả và phân tích

Sau khi kiểm tra mô hình thông qua tập test gồm 88 video thì ta được kết quả như bên dưới.

		Thực tế	
		Fall	NoFall
Dự đoán	Fall	TP (53)	FP (3)
	NoFall	FN (7)	TN (25)

Bảng 3.1: Confusion matrix

Độ chính xác :

$$Accuracy = \frac{TP + TN}{P + N} = \frac{53 + 25}{60 + 28} \approx 0.89$$

Độ nhạy - Tỷ lệ dương tính thực: chỉ số này càng cao thì số điểm là Positive bị bỏ sót càng ít. Recall = 1, nghĩa là tất cả số điểm có nhãn là Positive đều được mô hình nhận ra.

$$Recall = \frac{TP}{TP + FN} = \frac{53}{53 + 7} \approx 0.88 \quad (1)$$

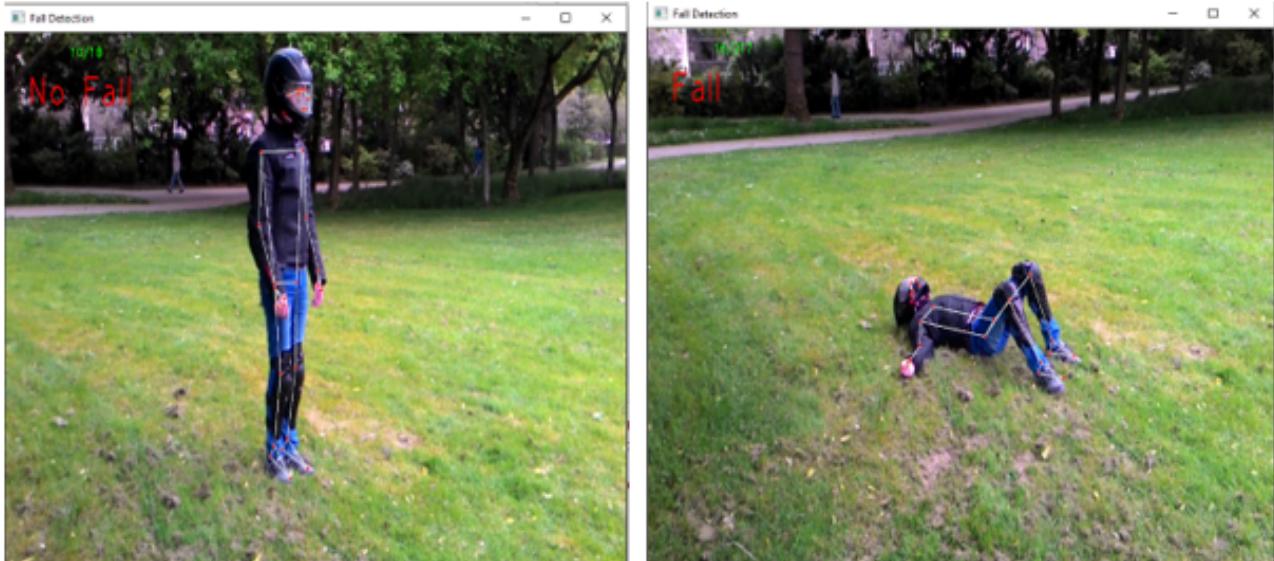
Tỉ lệ dương tính đoán đúng: chỉ số này càng cao thì số điểm mô hình dự đoán là Positive đều là Positive càng nhiều. Precision = 1, tức là tất cả số điểm mô hình dự đoán là Positive đều đúng, hay không có điểm nào có nhãn là Negative mà mô hình dự đoán nhầm là Positive.

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{53}{53 + 3} \approx 0.95 \quad (2)$$

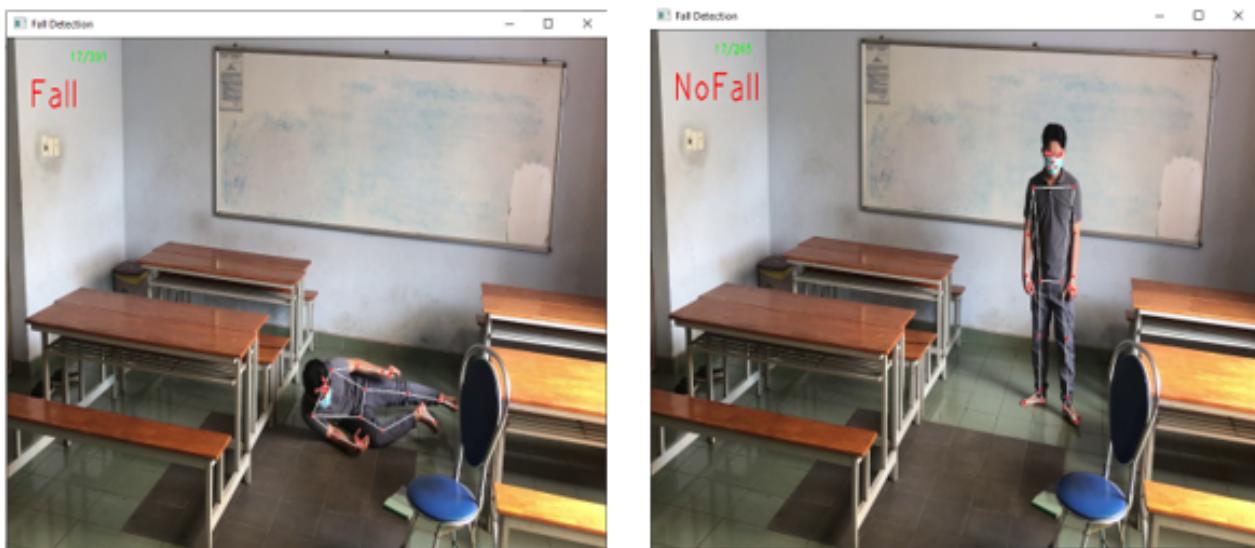
Tuy nhiên, nếu một trong hai chỉ số Recall hay Precision cao thì chưa đủ để đánh giá mô hình có tốt hay không. Vì vậy ta có chỉ số F1-Score trung bình điều hòa giữa 2 yếu tố trên. Từ (1) và (2) ta có:

$$F1 = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 * 0.88 * 0.95}{0.95 + 0.88} \approx 0.914$$

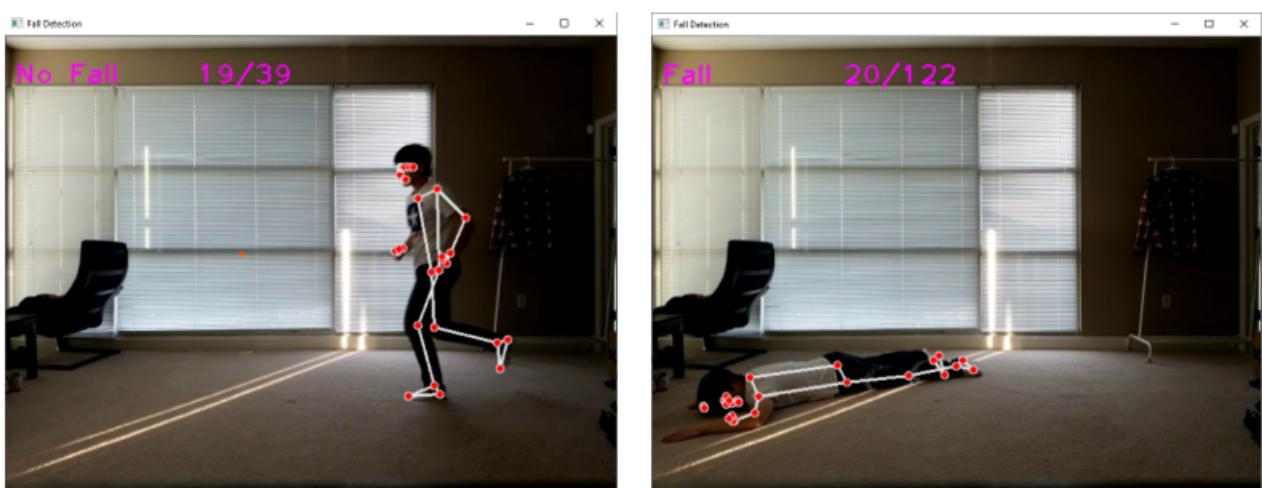
Một số trường hợp dự đoán chính xác trong quá trình kiểm tra model bao gồm trường hợp bình thường và té ngã như ở ngoài trời, trong phòng học, tại cơ quan.



Hình 3.21: Trường hợp bình thường và té ngã ngoài trời.



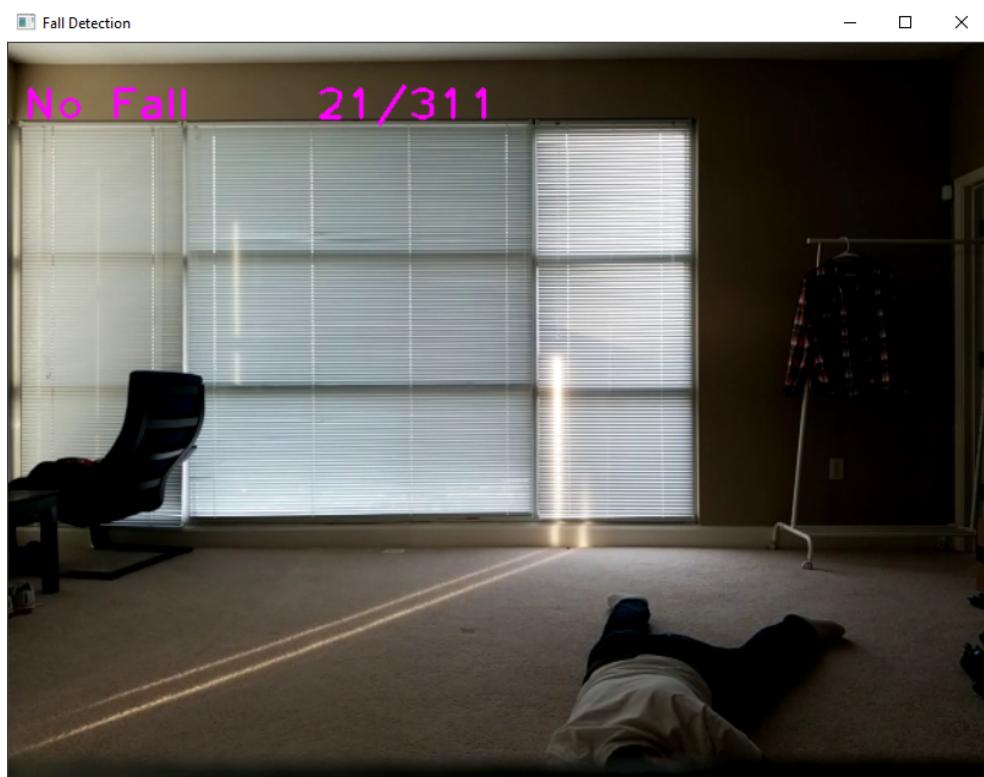
Hình 3.22: Trường hợp té ngã và bình thường trong phòng học.



Hình 3.23: Trường hợp bình thường và té ngã tại cơ quan làm việc.

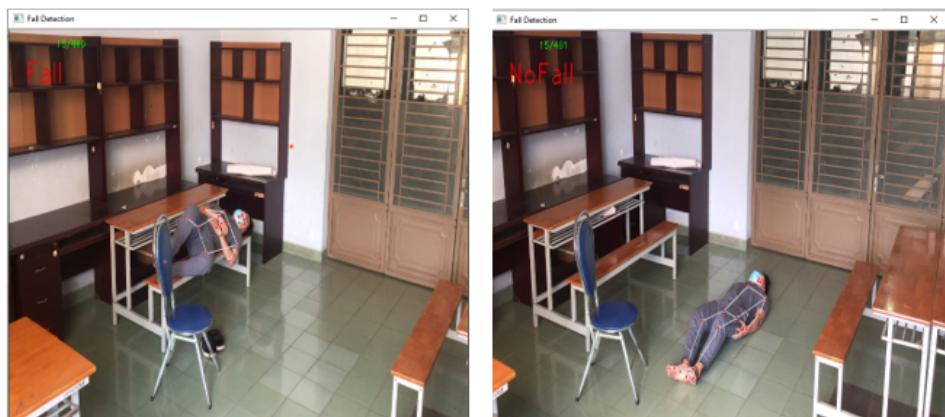
Một số trường hợp hệ thống bị nhầm lẫn hoạt động không phát hiện được như dưới đây:

- + Hệ thống không phát hiện được khung xương do đối tượng quá gần camera.



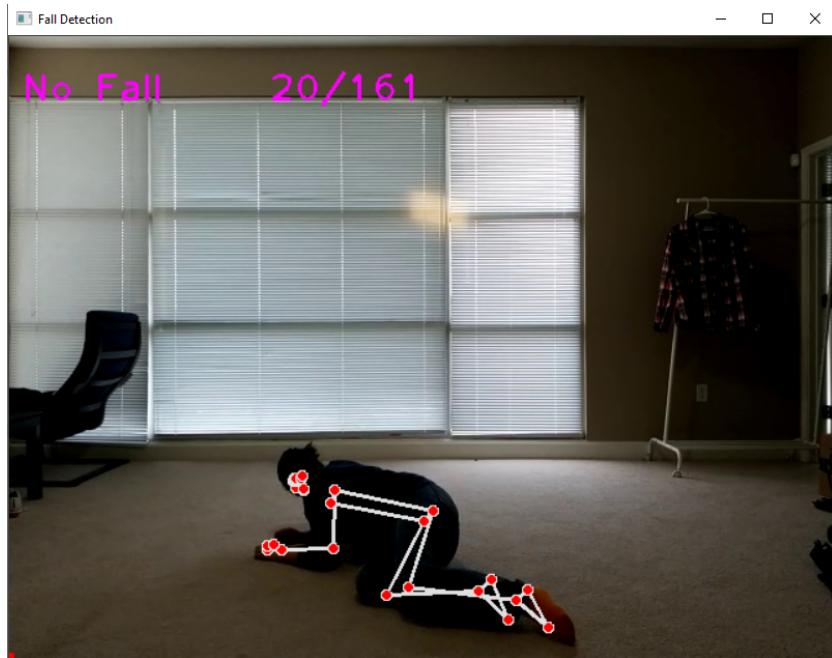
Hình 3.24: Trường hợp không phát hiện được khung xương.

- + Cùng là trường hợp nằm ngủ nhưng trường hợp ngủ trên ghế lại bị nhầm lẫn và đưa là kết quả là ngã. Nguyên nhân phần do thiếu hụt dữ liệu ở trường hợp này và hành động ngủ cũng khá giống với té ngã.



Hình 3.25: Trường hợp nhầm lẫn giữa ngủ và té ngã.

- + Trường hợp không phát hiện được té ngã do nhầm lẫn với tư thế quỳ.



Hình 3.26: Hình ảnh minh họa 6.

Nhìn chung phương pháp trên có những cải thiện vượt trội hơn rất nhiều so với phương pháp sử dụng YOLOv4 về thời gian chuẩn bị và huấn luyện, tốc độ xử lý trung bình và hơn hết là độ chính xác của mô hình. Do đó, ta sẽ chọn phương pháp này để xây dựng hệ thống phát hiện té ngã có tích hợp trang web cho người dùng.

3.3 Xây dựng ứng dụng web theo dõi hệ thống phát hiện té ngã

3.3.1 Giới thiệu về ứng dụng web (web application)

Trong thời kì chuyển đổi số hiện nay thì nhu cầu lưu trữ tất cả các dữ liệu cần thiết và có phương tiện xử lý toàn bộ các dữ liệu này, sau đó tiến hành trình bày kết quả cho người dùng ngày càng lớn. Các ứng dụng web (web application) là thứ có thể giải quyết các vấn đề này.

Web application là các ứng dụng web sử dụng kết hợp các server-side scripts (PHP và ASP) để xử lý việc lưu trữ và truy xuất thông tin, và client-side scripts (JavaScript và HTML) để trình bày thông tin cho người dùng [4]. Điều này cho phép người dùng tương tác với các ứng dụng bằng biểu mẫu trực tuyến, hệ thống quản lý nội dung, giỏ hàng mua sắm và hơn thế nữa. Ngoài ra, các ứng dụng còn cho phép nhân viên tạo tài liệu, chia sẻ thông tin, cộng tác trên các dự án và làm việc trên các tài liệu chung bất kể ở vị trí hoặc thiết bị nào.

Trong luận văn này chúng em sẽ xây dựng một ứng dụng web để theo dõi và quản lý hệ thống phát hiện té ngã. Ứng dụng sẽ có những chức năng cơ bản sau đây:

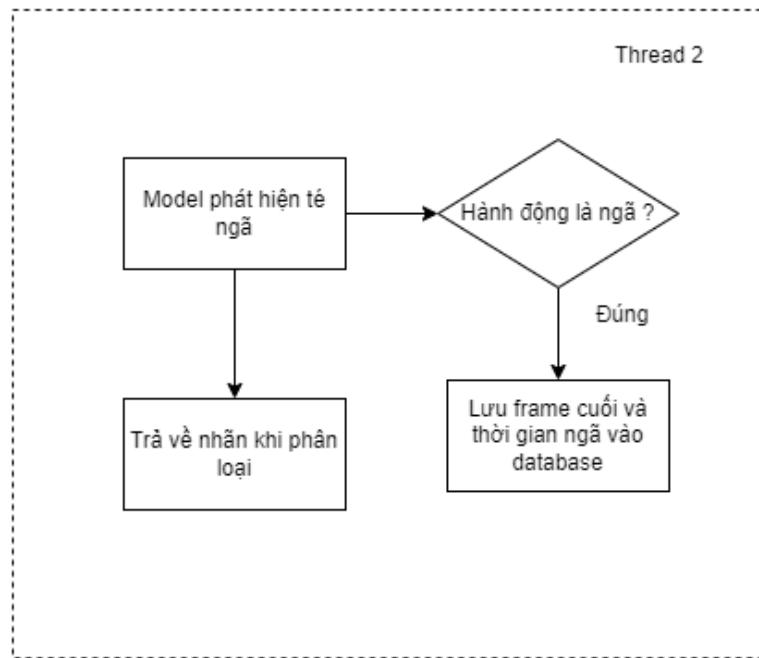
- Truy cập vào hệ thống cần có tài khoản đăng nhập đã được đăng ký và có bảo mật

- Theo dõi trạng thái của hệ thống và xuất ra khung hình của lần té ngã gần nhất.
- Lưu lại lịch sử các lần ngã.

Để xây dựng ứng dụng web với những chức năng trên, chúng em sẽ sử dụng thư viện Flask, là một micro-framework được xây dựng bằng ngôn ngữ lập trình Python. Flask cung cấp các công cụ, các thư viện và các công nghệ hỗ trợ xây dựng các API nhỏ, ứng dụng web chẵng hạn như các trang web, blog, trang báo,...

3.3.2 Phần backend của trang web

Phần backend của trang web chính là phần hệ thống phát hiện té ngã ở phần 3.2.1. Tuy nhiên để backend và frontend có thể giao tiếp và trao đổi thông tin với nhau, cần có một cơ sở dữ liệu. Và cũng chính vì thế mà sơ đồ khối ở phần 3.2.1 của hệ thống sẽ có thay đổi một chút để phù hợp với việc xây dựng web.



Hình 3.27: Sơ đồ khối mới cho backend.

Như hình 3.27, có sự thay đổi một chút ở Thread 2 so với sơ đồ khối ở 3.13. Sau khi mảng dữ liệu đủ 50 frame và được đưa vào model phân loại té ngã. Khi hành động được xác định là ngã thì lập tức các thông tin về hành động đó như frame cuối, thời gian ngã sẽ lưu vào cơ sở dữ liệu.

Cơ sở dữ liệu được sử dụng trong hệ thống này là Sqlalchemy. Chúng em tạo cơ sở dữ liệu và đặt tên là **webfall.db** trong cơ sở dữ liệu này gồm hai bảng. Một bảng để lưu thông tin tài khoản đăng nhập người dùng, bảng còn lại sẽ lưu thông tin lịch sử các lần té ngã.

	id	username	password_hash
	Filter	Filter	Filter
1	1	TrungPhan	\$2b\$12\$4oPburlPeASJq0H2CzCCGuWmifql2szg...
2	2	TrungPhan10	\$2b\$12\$ia1qeF3NB7P7NavDOtePluZsRdvIvzDpbX...
3	3	PhanTrung	\$2b\$12\$Uhefx8ZGPZOa19wMgLfGReG8pttL2oFKJ...
4	4	TrungPhan123	\$2b\$12\$2i.nRPn.0XvB4Vd46D7RWOTyJJS/...
5	5	Falldetection	\$2b\$12\$5OlpuU/...

Hình 3.28: Dữ liệu thông tin đăng nhập của người dùng

Hình 3.28 là bảng lưu trữ thông tin đăng nhập người dùng. Mỗi hàng là tên đăng nhập của người dùng và mật khẩu đã được hash để tăng tính bảo mật.

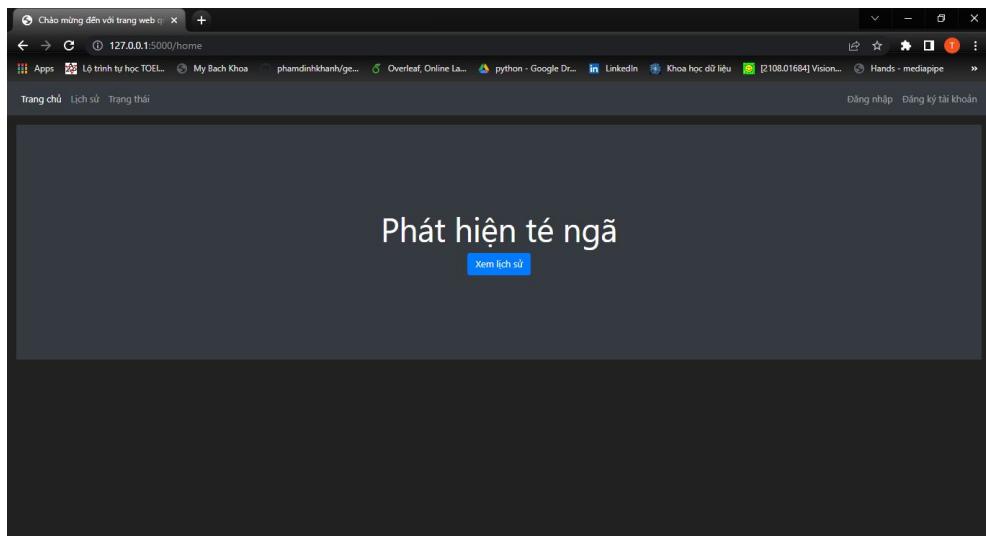
	id	time	image
	Filter	Filter	Filter
1	1	14:11:00 21/05/2022	14_11_00_21_05_2022
2	2	14:11:33 21/05/2022	14_11_33_21_05_2022
3	3	14:26:03 21/05/2022	14_26_03_21_05_2022
4	4	14:26:58 21/05/2022	14_26_58_21_05_2022
5	5	14:29:47 21/05/2022	14_29_47_21_05_2022
6	6	14:30:38 21/05/2022	14_30_38_21_05_2022
7	7	14:34:48 21/05/2022	14_34_48_21_05_2022
8	8	16:50:55 21/05/2022	16_50_55_21_05_2022
9	9	18:53:39 30/05/2022	18_53_39_30_05_2022

Hình 3.29: Dữ liệu thông tin lịch sử ngã.

Hình 3.29 là bảng lưu trữ thông tin lịch sử các lần té ngã. Mỗi hàng là mô tả một lần ngã với các thông tin như thời gian ngã và tên file hình của lần ngã đó.

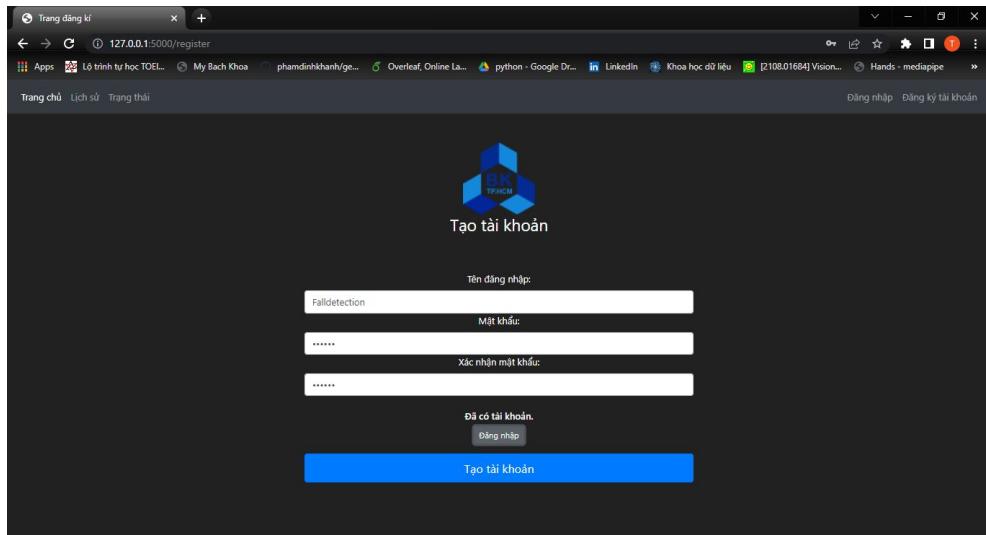
3.3.3 Phản frontend giao diện người dùng của trang web.

Từ phần backend đã xây dựng ở phần 3.3.2, chúng ta sẽ đi xây dựng giao diện người dùng bằng ngôn ngữ HTML.



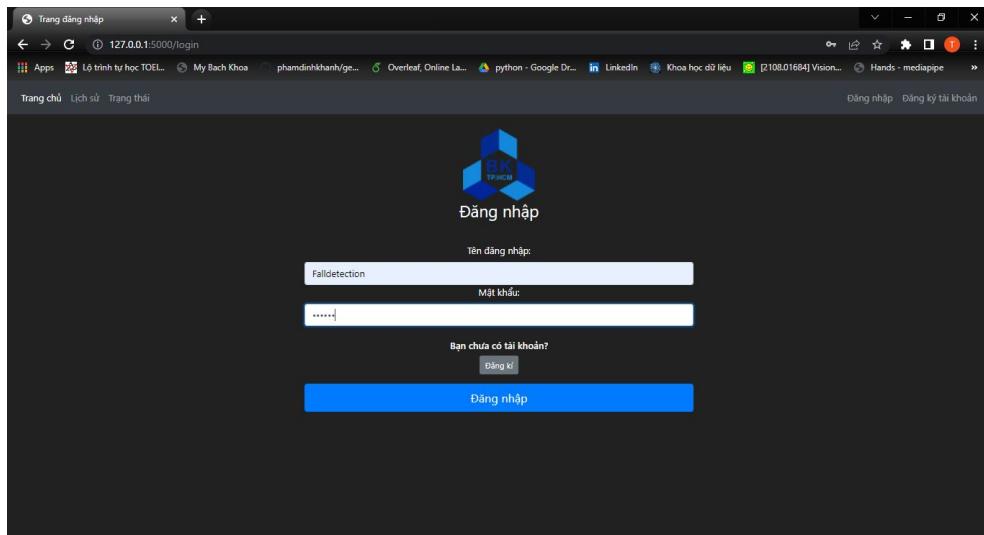
Hình 3.30: Trang chủ của website

Hình 3.30 là trang chủ và cũng là giao diện chính của website. Nội dung trang này người dùng nào cũng có thể truy cập được. Khi ấn vào nút xem lịch sử thì website sẽ điều hướng đến trang lịch sử như 3.33. Tuy nhiên để vào trang lịch sử người dùng cần phải đăng nhập.



Hình 3.31: Trang đăng kí tài khoản.

Hình 3.31 là trang đăng ký tài khoản người dùng. Tại đây người dùng đăng ký với tên đăng nhập, mật khẩu. Khi đăng ký cần điền đầy đủ thông tin vào form sau đó nhấn nút đăng ký. Các thông tin đăng ký người dùng sẽ được chuyển về user database để lưu trữ.



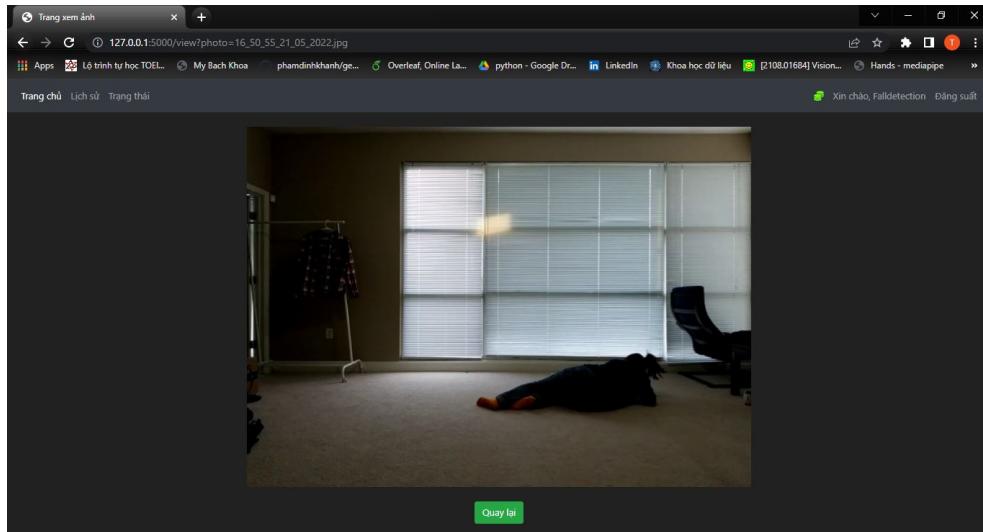
Hình 3.32: Trang đăng nhập

Sau khi có tài khoản đăng nhập vào ứng dụng web. Người dùng sẽ đăng nhập trang login như hình 3.32. Người dùng chỉ có thể đăng nhập được khi nhập đúng tên người dùng và mật khẩu đã được đăng ký. Sau Khi người dùng đã nhập đầy đủ thông tin vào form đăng nhập, nhấn vào nút đăng nhập để vào trang web. Khi đó các thông tin đăng nhập của form này được gửi về sever và kiểm tra xem các thông tin này có trùng khớp với user database không. Nếu hợp lệ thì đăng nhập thành công.

STT	Thời gian	Ảnh
1	16:50:55 21/05/2022	View
2	14:34:48 21/05/2022	View
3	14:30:38 21/05/2022	View
4	14:29:47 21/05/2022	View
5	14:26:58 21/05/2022	View
6	14:26:03 21/05/2022	View
7	14:11:33 21/05/2022	View
8	14:11:00 21/05/2022	View

Hình 3.33: Trang lịch sử.

Sau khi đăng nhập vào trang web với tài khoản đã đăng ký, người dùng có thể xem các thông tin về lịch sử các lần té ngã như ở hình 3.33. Tại đây có các thông tin về thời gian về lần té ngã và ảnh của lần té ngã. Muốn xem ảnh của lần té ngã tương ứng, người dùng chỉ cần nhấp vào nút view thì lập tức ảnh sẽ hiện ra như hình 3.34. Muốn quay lại trang lịch sử như hình 3.33 thì chỉ cần nhấp nút quay lại.



Hình 3.34: Ảnh của lần té ngã khi người dùng bấm vào nút view.

Người dùng cũng có thể xem lần hệ thống phát hiện trường hợp té ngã gần đây nhất bằng cách vào trang trạng thái như ở hình 3.35. Tại đây sẽ hiển thị ảnh và thời gian của lần té ngã gần nhất.



Hình 3.35: Trang trạng thái

3.3.4 Kết quả

Web ứng dụng được đã có được đầy đủ chức năng như mục tiêu đề ra. Tuy nhiên khi ứng dụng web còn khá thụ động. Khi có hành động té ngã được phát hiện mới, nếu ta đang ở trạng thái hoặc lịch sử thì nó sẽ không cập nhật trạng thái tự động. Cần phải tải lại trang thì trang web mới thực hiện được việc cập nhật.

Chương 4

Kết luận

4.1 Tóm tắt và kết luận chung

Thông qua đề tài này, hai phương pháp xây dựng hệ thống té ngã đã được trình bày. Phương pháp A (3.1) sử dụng Yolov4 để nhận dạng tư thế ở mỗi khung hình và so sánh sự thay đổi đột ngột tư thế của hai khung hình liên tiếp để kết luận có xuất hiện té ngã hay là không. Phương pháp B (3.2) sử dụng module **Mediapipe Pose** cùng với một mô hình mạng học sâu để phân loại một chuỗi khung hình có chứa thông tin tư thế của con người có phải là hành động té ngã. Mặc khác, sau quá trình phân tích kiểm tra và so sánh gần 90 video có té ngã, nhóm nhận thấy phương pháp B có kết quả và ưu điểm vượt trội hơn so với phương pháp A. Vì vậy nhóm đã chọn phương pháp B để tiếp tục tối ưu và nhận diện cho hệ thống. Và đã xây dựng một website để cho người dùng tiện theo dõi khi có sự cố té ngã xảy ra.

Sau khi thực hiện đề tài , nhóm đã rút ra được rất nhiều kinh nghiệm từ việc sưu tầm , chọn lọc, lấy mẫu dữ liệu , gán nhãn, huấn luyện , hiểu được các chỉ số huấn luyện, cải thiện mô hình và quan trọng nhất là phải chọn mô hình phù hợp với bài toán cần giải quyết.

Dựa vào phân tích và kết quả thực hiện của chương 3, nhóm đã hoàn thành được những mục tiêu đã đề ra ban đầu như sau:

- Hệ thống có thể hoạt động real-time tốt.
- Tốc độ xử lý khung hình cao (20 FPS).
- Độ chính xác cao (F1-Score 0.914).
- Đã xây dựng một trang web lưu lại các thông tin giúp người dùng theo dõi.

4.2 Hạn chế và hướng cải thiện mô hình.

Những hạn chế mà nhóm chưa thực hiện được và hướng giải quyết:

- Mặc dù độ chính xác cao, nhưng ở những trường hợp đặc biệt như nằm ngủ thì hệ thống vẫn còn nhầm lẫn. Cần phải thu thập và thực hiện thêm nhiều dữ liệu để huấn luyện cho mô hình.
- Hệ thống chỉ áp dụng cho một đối tượng, khi có hai đối tượng trở lên thì hệ thống sẽ chỉ theo dõi đối tượng đầu tiên mà camera nhận diện. Vì vậy cần một giải thuật tốt hơn để nhận diện và phân biệt giữa các đối tượng để hỗ trợ cho việc nhận diện.
- Hiện đang vận hành trên phần cứng của laptop, cần phải tối ưu phần cứng nhỏ gọn hơn, ít tốn chi phí hơn nhưng phải đảm bảo hiệu suất của hệ thống.

4.3 Hướng phát triển

Hướng phát triển của nhóm là sẽ nâng cấp hệ thống để sử dụng được cho nhiều đối tượng trong ở một khu vực nhất định như bệnh viện, viện dưỡng lão và có còi báo hiệu. Có thể tích hợp việc theo dõi trực tiếp camera thông qua web.

Mặc khác, hệ thống phải được vận hành trên một phần cứng khác thay vì laptop mà nhóm đang hướng đến là Raspberry.

Phụ lục A

Code chương trình phát hiện té ngã của hệ thống

```
PeoplePose = PeoplePose()
model = keras.models.load_model('model.h5')
lm_list = []
cap = cv2.VideoCapture(name_video)
while cap.isOpened():
    no_frame +=1
    ret, frame = cap.read()
    if ret:
        frame_save = frame.copy()
        frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = PeoplePose.pose.process(frameRGB)
        if results.pose_landmarks:
            lm = PeoplePose.make_landmark_timestep(results)
            lm_list.append(lm)
            # Dua vao model nhan dien
            if len(lm_list)==50:
                t1 = threading.Thread(target = detect, args=(model,lm_list,frame_save,))
                t1.start()
                lm_list = []
            cTime = time.time()
            fps= 1/(cTime-pTime)
            pTime = cTime
            # Draw FPS frame
            cv2.putText(frame, label,(10,50),cv2.FONT_HERSHEY_PLAIN,2,(255,0,255),2 )
            frame_shape = np.array(frame).shape
            ratio = frame_shape[0]/frame_shape[1]
            frame = cv2.resize(frame,(int(600/ratio),600))
            cv2.imshow('Fall Detection',frame)
            if cv2.waitKey(10) & 0xFF ==ord('q'):
                break
            else:
                break
    cap.release()
    cv2.destroyAllWindows()
```

Toàn bộ code được push lên github với đường dẫn <https://github.com/M10TrungPhan/Falldetection>.

Phụ lục B

Code chương trình web cho hệ thống

```
@app.route('/')
@app.route('/home')
def home_page():
    return render_template('home.html')

@app.route('/history')
def history_page():
    history = History.query.order_by(desc(History.id)).all()
    index = len(history)
    return render_template('history.html', history=history, index = index)

@app.route('/status')
def status_page():
    return render_template('status.html', time=time)

@app.route('/register', methods=['GET', 'POST'])
def register_page():
    form = RegisterForm()
    if form.validate_on_submit():
        user_to_create = User(username=form.username.data,
                              password=form.password1.data)
        db.session.add(user_to_create)
        db.session.commit()
        login_user(user_to_create)
        return redirect(url_for('history_page'))

@app.route('/login', methods=['GET', 'POST'])
def login_page():
    form = LoginForm()
    if form.validate_on_submit():
        attempted_user = User.query.filter_by(username=form.username.data).first()
        return render_template('login.html', form=form)

@app.route('/logout')
def logout_page():
    logout_user()
    return redirect(url_for("home_page"))

@app.route('/view')
def view_page():
    photo = request.args.get('photo', default = "")
    return render_template('view.html', photo=photo)
```

Toàn bộ code được push lên github với đường dẫn <https://github.com/M10TrungPhan/Falldetection>.

Tài liệu tham khảo

- [1] Valentin Bazarevsky and Ivan Grishchenko. *BlazePose: On-device Real-time Body Pose tracking*. 2020. arXiv: [2006.10204 \[cs.CV\]](https://arxiv.org/abs/2006.10204).
- [2] Valentin Bazarevsky and Ivan Grishchenko. *On-device, Real-time Body Pose Tracking with MediaPipe BlazePose*. URL: <https://ai.googleblog.com/2020/08/on-device-real-time-body-pose-tracking.html>. (Truy cập: 05.05.2022).
- [3] Valentin Bazarevsky et al. *BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs*. 2019. arXiv: [1907.05047 \[cs.CV\]](https://arxiv.org/abs/1907.05047).
- [4] Giang. *Web App là gì? Có gì khác với Website*. URL: <https://bizflycloud.vn/tin-tuc/web-application-la-gi-co-gi-khac-voi-website-20180619105652369.html>. (Truy cập: 05.2022).
- [5] Cát Hải. “*Vitruvian Man*” của Leonardo Da Vinci và tǐ lệ hoàn mỹ. URL: <https://trithucvn.org/van-hoa/leonardo-da-vinci-ti-le-hoan-mi-cua-co-the-nguo.html>. (Truy cập: 08.05.2022).
- [6] HaoyiZhu. <https://github.com/vita-epfl/openpifpaf>. 2018. URL: <https://github.com/MVIG-SJTU/AlphaPose>.
- [7] Nguyễn Văn Hiếu. *Flask python là gì? Thư viện flask trong lập trình Python*. URL: <https://nguyenvanhieu.vn/thu-vien-flask-python-la-gi/>.
- [8] Jonathan Hui. *Yolov4*. 2020. URL: <https://jonathan-hui.medium.com/yolov4-c9901eaa8e61>.
- [9] Hồ Sỹ Hùng. *OpenCV và các ứng dụng của nó hiện nay*. 4/07/2016. URL: <https://techmaster.vn/posts/33943/opencv-vacac-ung-dung-cua-no-hiennay>.
- [10] Phạm Dinh Khánh. *Convolutional Neural Network*. 2019. URL: <https://phamdinhkhanh.github.io/2019/08/22/convolutional-neural-network.html>.
- [11] Kika. *Giới thiệu ngôn ngữ lập trình python*. 11/7/2021. URL: <https://moitruongso.com/gioi-thieu-ngon-ngu-lap-trinh-python/>.
- [12] MediaPipe. *MediaPipe Pose*. 2020. URL: <https://google.github.io/mediapipe/solutions/pose.html>.
- [13] Alejandro Newell, Kaiyu Yang, and Jia Deng. *Stacked Hourglass Networks for Human Pose Estimation*. 2016. arXiv: [1603.06937 \[cs.CV\]](https://arxiv.org/abs/1603.06937).
- [14] Võ Khắc Khôi Nguyễn. *Hội chứng sau té ngã ở người cao tuổi*. URL: https://www.vinmec.com/vi/tin-tuc/thong-tin-suc-khoe/suc-khoe-tong-quat/hoi-chung-sau-te-nga-o-nguoi-cao-tuoi/?link_type=related_posts.
- [15] Phương Thu Nguyễn. *Phòng chống té ngã ở người cao tuổi*. URL: https://moh.gov.vn/web/phong-chong-tai-nan-thuong-tich/tin-noi-bat/-/asset_publisher/iinMRn208ZoI/content/phong-chong-te-nga-o-nguoi-cao-tuoi. (Truy cập: 05.05.2022).

- [16] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [17] Bảo Phạm. *Tìm Hiểu Mô Hình YOLO Cho Bài Toán Object Detection - Understanding YOLO*. URL: <https://pbcquoc.github.io/yolo/>. (accessed: 01.02.2022).
- [18] E. Auvinet; J. Rousseau. “Multiple cameras fall dataset”. In: *Technical report 1350, DIRO - Université de Montréal* (7.201).
- [19] shijieS. *Deep Motion Modeling Tracker*. 2019. URL: <https://github.com/shijieS/DMMN>.
- [20] Ralf C. Staudemeyer and Eric Rothstein Morris. *Understanding LSTM. A tutorial into Long Short-Term Memory.Recurrent Neural Networks*. 2020. arXiv: [1909.09586 \[cs.CV\]](https://arxiv.org/abs/1909.09586).
- [21] Alexandre Alahi Sven Kreiss Lorenzo Bertoni. <https://github.com/vita-epfl/openpifpaf>. 2021. URL: <https://github.com/vita-epfl/openpifpaf1>.
- [22] Teky. *OpenCV là gì? Cách sử dụng OpenCV như thế nào?* 1/08/2021. URL: <https://teky.edu.vn/blog/opencv-la-gi/>.
- [23] Nguyễn Chiến Thắng. *Train YOLO v4 train trên googlecolab chi tiết và đầy đủ*. 2020. URL: <https://www.miai.vn/2020/05/25/yolo-series-train-yolo-v4-train-tren-colab-chi-tiet-va-day-du-a-z/>.
- [24] Vũ Hữu Tiệp. *Bài 15: Overfitting*. URL: <https://machinelearningcoban.com/2017/03/04/overfitting/>.
- [25] Tzutalin. *LabelImage*. 2015. URL: <https://github.com/tzutalin/labelImg>.
- [26] Got It Vietnam. *Tìm hiểu Sigmoid Function và lịch sử hình thành của nó*. 2021. URL: <https://vn.got-it.ai/blog/tim-hieu-sigmoid-function-va-lich-su-hinh-thanh-cua-no>.
- [27] Zhuo Wang et al. “Possible Life Saver: A Review on Human Fall Detection Technology”. In: *robotics* (7.201).
- [28] Falin Wu et al. “Development of a Wearable-Sensor-Based Fall Detection System”. In: *Hindawi Publishing Corporation International Journal of Telemedicine and Applications* (8.2014).
- [29] Rui Zhang. *Exploiting Offset-guided Network for Pose Estimation and Tracking*. 2019. arXiv: [1906.01344 \[cs.CV\]](https://arxiv.org/abs/1906.01344).