

TRƯỜNG ĐẠI HỌC CÀN THƠ
TRƯỜNG CÔNG NGHỆ THÔNG TIN - TRUYỀN THÔNG



LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC
NGÀNH HỆ THỐNG THÔNG TIN

Đề tài
HỆ THỐNG
PHÁT HIỆN HÀNH VI LEO RÀO

Sinh viên: Chau Si Quyчch Di

Mã số: B1805684

Khóa: K44

Cần Thơ, 12/2022

TRƯỜNG ĐẠI HỌC CÀN THƠ
TRƯỜNG CÔNG NGHỆ THÔNG TIN - TRUYỀN THÔNG

LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC
NGÀNH HỆ THỐNG THÔNG TIN

Đề tài
**HỆ THỐNG
PHÁT HIỆN HÀNH VI LEO RÀO**

Người hướng dẫn
TS. Phạm Thị Ngọc Diễm

Sinh viên: Chau Si Quých Đì
Mã số: B1805684
Khóa: K44

Cần Thơ, 12/2022

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
KHOA HỆ THÔNG THÔNG TIN

**XÁC NHẬN CHỈNH SỬA LUẬN VĂN
THEO YÊU CẦU CỦA HỘI ĐỒNG**

Tên luận văn (tiếng Việt và tiếng Anh):

**Hệ Thống Phát Hiện Hành Vi Leo Rào
Fence Climbing Behavior Detection System**

Họ tên sinh viên: CHAU SI QUÝCH ĐÌ MASV: B1805684

Mã lớp: DI1895A2

Đã báo cáo tại hội đồng ngành: Hệ Thống Thông Tin

Ngày báo cáo: 15/12/2022

Hội đồng báo cáo gồm:

TS. Nguyễn Thanh Hải

TS. Nguyễn Minh Khiêm

TS. Phạm Thị Ngọc Diễm

Luận văn đã được chỉnh sửa theo góp ý của Hội đồng.

Cần Thơ, ngày 22 tháng 12 năm 2022

Giáo viên hướng dẫn

(Ký và ghi họ tên)

Phạm Thị Ngọc Diễm

LỜI CẢM ƠN

Thẩm thoát bốn năm rưỡi trên giảng đường học tại trường Đại Học Cần Thơ cũng đã kết thúc, cảm ơn những ngày nắng ngày mưa vội vàng cắp sách đến lớp học, cảm ơn những tiếng cười tha thiết từ Thầy (Cô) xoa dịu đi những lúc căng thẳng và mệt mỏi. Cảm ơn toàn thể quý Thầy (Cô) trường Đại Học Cần Thơ nói chung và Thầy (Cô) Trường Công Nghệ Thông Tin & Truyền Thông nói riêng lời cảm ơn chân thành và sâu sắc.

Nhưng hơn hết em xin gửi lời cảm ơn đến cô Phạm Thị Ngọc Diễm đã tạo điều kiện cho em được làm việc cùng cô trong suốt thời gian thực hiện đề tài luận văn vừa qua, em xin chân thành cảm ơn những sự chỉ bảo, sự hướng dẫn tâm đắc nhất từ cô. Em không biết viết gì thêm ngoài những lời cảm ơn chân thành sâu sắc này nhǎn nhủ đến cô, chỉ mong rằng cô có thật nhiều sức khỏe và đạt được nhiều thành tích trong công việc.

Em cảm ơn thầy Nguyễn Thanh Hải và thầy Nguyễn Minh Khiêm đã có những lời góp ý để Luận văn của em được hoàn thiện hơn.

Đặc biệt con xin cảm ơn ba mẹ đã tạo điều kiện cho con có ăn có học có được như ngày hôm nay. Khi con sắm sửa bước ra môi trường Trường Đại học, bàn tay ba mẹ không còn mềm mại như ngày xưa mà nay dường như đã khô và cứng vì những đồng tiền cho con đến trường. Con xin cảm ơn vì sự hy sinh lớn lao của ba mẹ đã dành cho con. Cảm ơn Anh Hai đã hiểu và chia sẻ dụng cụ học tập cho em để em không bị thiếu thốn bất cứ thứ gì trong quá trình học tập.

Cảm ơn những người bạn đã biết quan tâm, chia sẻ, giúp đỡ cùng nhau trong quá trình học tập cũng như trong thời gian làm luận văn.

Cảm ơn cộng đồng ‘Hội những anh em thích ăn mì AI’ của anh Nguyễn Chiến Thắng đã cho em có nhiều thông tin hữu ích cũng như được học hỏi từ những người có kinh nghiệm trong lĩnh vực thi giác máy tính.

Mặc dù đã cố gắng hoàn thành đề tài luận văn nhưng không thể tránh được những sai sót, rất mong được sự thông cảm và nhận được sự góp ý từ quý bạn đọc.

Em xin chân thành cảm ơn.

Cần Thơ, tháng 12 năm 2022

Sinh viên thực hiện

Chau Si Quých Đí

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

Cần Thơ, tháng 12 năm 2022

Giảng viên hướng dẫn

TS. Phạm Thị Ngọc Diễm

NHẬN XÉT CỦA GIẢNG VIÊN PHẢN BIỆN

Cần Thơ, tháng 12 năm 2022

Giảng viên phản biện

TÓM TẮT

Ngày nay lĩnh vực thị giác máy tính trở nên rất phổ biến vì tính hiệu quả mà nó mang lại trong cuộc sống, những thiết bị thông minh phát hiện hành vi bất thường, phát hiện hành vi gian lận trong thi cử, v.v... được ra đời.

Trong đề tài '**Hệ thống phát hiện hành vi leo rào**' tập trung nghiên cứu vào các mô hình phát hiện đối tượng, cụ thể là YOLO v4, YOLO v7, SDD và Faster-RCNN với mục tiêu phát hiện các đối tượng có hành vi leo rào và gửi cảnh báo đến người dùng. Trong quá trình đào tạo các mô hình trên 5340 ảnh đã thu thập được trong thực tế, kết quả mô hình YOLO v4 áp dụng kỹ thuật tăng cường dữ liệu cho kết quả tốt nhất so với các giải thuật còn lại, cụ thể độ chính xác (IoU) của mô hình đạt 71% và F1-score đạt trên 87%, về kết quả thực nghiệm của mô hình với video kích thước 1280x720 trong phạm vi từ camera đến hàng rào là 5 mét mô hình phát hiện được đối tượng leo rào với độ chính xác đạt từ 98 đến 99%, tốc độ xử lý khung hình đạt 48 fps.

Từ khóa: Hành vi bất thường của con người, phát hiện hành vi leo rào, YOLO v4, mô hình học sâu.

ABSTRACT

The topic 'Fence Climbing Behavior Detection System' focuses on object detection models, namely YOLO v4, YOLO v7, SDD and Faster-RCNN models with the goal of detecting objects with climbing behavior and sending alerts to users. During the training of the models on 5340 images collected in practice, the YOLO v4 model results applied data augmentation techniques for the best results compared to the rest of the algorithms, specifically the accuracy (IoU) of the model reached 71% and the F1-score reached over 87%, About the experimental results of the model with video size 1280x720 in the range from camera to fence is 5 meters The model detected the object climbing the fence with an accuracy of 98 to 99% frame processing speed reaching 48 fps.

Keywords: abnormal actions of people, hedge-climbing action detection, YOLOv4, deep learning model

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU.....	1
1.1. Đặt vấn đề.....	1
1.2. Các nghiên cứu liên quan	1
1.3. Mục tiêu đề tài.....	3
1.4. Đối tượng và phạm vi đề tài.....	3
1.5. Nội dung đề tài	3
1.6. Những đóng góp chính của đề tài	3
1.7. Bố cục của luận văn	4
1.8. Tổng kết chương	4
CHƯƠNG 2. MÔ TẢ BÀI TOÁN	5
2.1. Mô tả chi tiết bài toán.....	5
2.2. Cơ sở lý thuyết	5
2.2.1. Deep Learning (Mô hình học sâu)	5
2.2.2. Mạng nơ-ron tích chập CNN.....	6
2.2.3. Mô hình YOLO	9
2.2.4. Mô hình YOLO v4 [13]	10
2.2.5. Mô hình YOLO V7	14
2.2.6. Mô hình SSD.....	17
2.2.7. Mô hình Faster-RCNN [21]	19
2.3. Phương pháp đánh giá.....	20
2.4. Tổng kết chương	21
CHƯƠNG 3. THIẾT KẾ VÀ CÀI ĐẶT GIẢI PHÁP	22
3.1. Đào tạo mô hình phát hiện hành vi leo rào	22
3.1.1. Giới thiệu môi trường và tham số đào tạo	22
3.1.2. Giới thiệu tập dữ liệu	23
3.1.3. Đào tạo mô hình YOLO v4.....	26
3.1.4. Đào tạo mô hình YOLO v7	30
3.1.5. Đào tạo mô hình SSD MobileNet v2	33
3.1.6. Đào tạo mô hình Faster-RCNN.....	37
3.1.7. Đánh giá kết quả đào tạo mô hình phát hiện hành vi leo rào.....	40
3.1.8. Kết luận	40
3.2. Thiết kế và cài đặt hệ thống	45
3.2.1. Môi trường thiết kế và cài đặt hệ thống	45

3.2.2. Kiến trúc hệ thống	45
3.3. Tổng kết chương	47
CHƯƠNG 4. KIỂM THỬ VÀ ĐÁNH GIÁ	48
4.1. Kiểm thử.....	48
4.1.1. Kiểm thử trên mô hình YOLO v4	48
4.1.2. Kiểm thử trên hệ thống	48
4.2. Kết quả kiểm thử	52
4.2.1. Kết quả kiểm thử trên mô hình YOLO v4	52
4.2.2. Kết quả kiểm thử trên hệ thống.....	53
4.3. Tổng kết chương	62
CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	63
5.1. Kết luận	63
5.2. Hướng phát triển	63
TÀI LIỆU THAM KHẢO.....	64
PHỤ LỤC	67
7.1. Hướng dẫn cài đặt telegram và tạo chat bot.....	67
7.2. Tích hợp Telegram vào hệ thống	70
7.3. Đào tạo các mô hình YOLO v4, YOLO v7, SSD và Faster-RCNN.....	71
7.3.1. Cách Đào tạo mô hình YOLO v4	71
7.3.2. Cách Đào tạo mô hình YOLO v7	76
7.3.3. Cách đào tạo mô hình SSD	79
7.3.4. Cách đào tạo mô hình với Faster-RCNN	90

DANH MỤC HÌNH

Hình 2.1 Các thuật toán thuộc bài toán phát hiện đối tượng	6
Hình 2.2 Ví dụ tích chập ảnh ma trận 5x5 với bộ lọc 3x3	7
Hình 2.3 Ví dụ cách trượt stride = 2 và dùng đệm padding = 0	7
Hình 2.4 Ví dụ tầng pooling sử max pooling	8
Hình 2.5 Tầng kết nối đầy đủ nhận đầu vào dữ liệu làm phẳng	8
Hình 2.6 Kiến trúc mạng YOLO	9
Hình 2.7 Hệ thống chia ảnh đầu vào thành SxS ô [12].....	10
Hình 2.8 Kiến trúc mô hình YOLO V4	11
Hình 2.9 Mô hình YoLo-SPP	12
Hình 2.10 Mô hình mạng PAN [14]	12
Hình 2.11 Mô hình PAN cải tiến [13]	13
Hình 2.12 dự đoán bounding box	13
Hình 2.13 Ảnh đầu vào đưa vào khối Stem và cấu tạo của ELAN [18].....	14
Hình 2.14 Các ELAN liên kết với nhau bằng transition.....	14
Hình 2.15 Kiến trúc SPPCSPC [18]	15
Hình 2.16 Kiến trúc Neck trong YOLO v7 [18].....	16
Hình 2.17 Implicit Knowledge trong YOLOR [19]	16
Hình 2.18 Mô hình Auxiliary head [17]	17
Hình 2.19 Kiến trúc mô hình SDD	17
Hình 2.20 Kiến trúc MobileNet v2 [20]	18
Hình 2.21 Kiến trúc mô hình Faster-RCNN [21]	19
Hình 2.22 Mô hình ResNet50 [22]	20
Hình 2.23 Minh họa công thức tính IoU [23]	20
Hình 3.1 Kỹ thuật tổng quan đào tạo các mô hình	22
Hình 3.2 Mô phỏng sử dụng Google Colaboratory	22
Hình 3.3 Ví dụ dữ liệu ảnh chứa hành vi leo rào và bình thường	23

Hình 3.4 ví dụ về ảnh xoay và ảnh kéo dãn	24
Hình 3.5 Công cụ LabelImg hỗ trợ gắn nhãn	25
Hình 3.6 Ví dụ nhãn ở định dạng YOLO	25
Hình 3.7 ví dụ nhãn được tạo ở PASCAL/VOC ‘.xml’	26
Hình 3.8 Mô phỏng quá trình huấn luyện YOLO v4	27
Hình 3.9 Biểu đồ mAP/Avg loss mô hình YOLO v4	28
Hình 3.10 Biểu đồ loss trong quá trình huấn luyện và đánh giá YOLO v7	31
Hình 3.11 Mô phỏng huấn luyện mô hình SSD MobileNet v2	33
Hình 3.12 Biểu đồ loss của SSD MobileNet v2	34
Hình 3.13 Biểu loss trong quá trình đào tạo mô hình Faster-RCNN	38
Hình 3.14 Kết quả thực nghiệm trên mô hình tăng cường dữ liệu và cải tiến mô hình	44
Hình 3.15 Kiến trúc tổng quan hệ thống phát hiện hành vi leo rào	45
Hình 3.16 Giao diện đăng nhập	46
Hình 3.17 Giao diện làm việc chính	46
Hình 3.18 Người dùng nhận cảnh báo từ hệ thống trên chat-bot	47
Hình 4.1 Kết quả đánh giá mô hình trên 154 tập dữ liệu	52
Hình 4.2 Các chức năng chính dành cho người dùng	53
Hình 4.3 Chức năng chế độ phân giải và ngưỡng chính xác	53
Hình 4.4 Chức năng Phát Hình Ảnh yêu cầu chọn file ảnh	54
Hình 4.5 Chức năng Phát Video yêu cầu chọn file Video	54
Hình 4.6 Giao diện chức năng Xuất Video	55
Hình 4.7 Giao diện chức năng Kho Ảnh	55
Hình 4.8 Chức năng Kho Video	56
Hình 4.9 Kết quả phát hiện hành vi bằng chức năng Phát Ảnh (1)	56
Hình 4.10 Kết quả phát hiện hành vi bằng chức năng Phát Ảnh (2)	57
Hình 4.11 Kết quả phát hiện hành vi bằng chức năng Phát Video (1)	57
Hình 4.12 Kết quả phát hiện hành vi bằng chức năng Phát Video (2)	58
Hình 4.13 Kết quả thông báo xuất video thành công	58

Hình 4.14 Video xuất ra được lưu trữ trên Kho Video.....	59
Hình 4.15 Kết quả người dùng chọn hình ảnh để xem lại	59
Hình 4.16 Kết quả người dùng chọn video xem lại video	60
Hình 4.17 Kết quả hệ thống phát hiện bằng chức năng Phát Camera (1).....	60
Hình 4.18 Kết quả hệ thống phát hiện bằng chức năng Phát Camera (2).....	61
Hình 4.19 Hệ thống yêu cầu tắt chế độ Live	61

DANH MỤC HÌNH PHỤ LỤC

Phụ lục hình 1 Giao diện ứng dụng Telegram sau khi đăng nhập thành công	67
Phụ lục hình 2 Tìm kiếm BotFater.....	67
Phụ lục hình 3 Tạo chat bot	67
Phụ lục hình 4 Đặt tên chat bot thành công	68
Phụ lục hình 5 kết quả đặt username ‘Htleoraobot’	68
Phụ lục hình 6 Kiểm tra chat bot	69
Phụ lục hình 7 Người dùng nhắn tin với chat bot	69
Phụ lục hình 8 Lấy ID người dùng	69
Phụ lục hình 9 Thiết lập ID và TOKEN người dùng vào hệ thống	70
Phụ lục hình 10 Gọi hàm send_telegrame vào app.py.....	70
Phụ lục hình 11 Thiết lập hàm alert()	70
Phụ lục hình 12 Kiểm tra số frame chứa đối tượng leo rào và gửi cảnh báo.....	71
Phụ lục hình 13 Kết nối Google Colab	71
Phụ lục hình 14 Kết nối Google Coalb với Drive.....	72
Phụ lục hình 15 cài đặt môi trường Darknet.....	72
Phụ lục hình 16 Cài đặt thành công Darknet	72
Phụ lục hình 17 Tạo file yolo.names	72
Phụ lục hình 18 Thư mục darknet/data	72
Phụ lục hình 19 File yolov4-custom.config chứa trong cfg	73
Phụ lục hình 20 file yolov4-custom.cfg	73
Phụ lục hình 21 Chia train.txt và val.txt	74
Phụ lục hình 22 Kết quả chia train.txt và val.txt.....	74
Phụ lục hình 23 Tạo file yolo.data	75
Phụ lục hình 24 Thiết lập Makefile.....	75
Phụ lục hình 25 Biên dịch mã nguồn	75
Phụ lục hình 26 Cài đặt mạng cơ sở	75

Phụ lục hình 27 Chạy đào tạo mô hình YoLo v4.....	76
Phụ lục hình 28 Thực nghiệm với hình ảnh.....	76
Phụ lục hình 29 Thực nghiệm với video.....	76
Phụ lục hình 30 Tạo môi trường Yolo v7	76
Phụ lục hình 31 Chia train và val	77
Phụ lục hình 32 Cấu trúc thư mục train và val	77
Phụ lục hình 33 file coco.yaml	78
Phụ lục hình 34: cấu hình custom_data.yaml	78
Phụ lục hình 35 Chính sửa file yolov7-custom.yaml.....	78
Phụ lục hình 36 Lệnh đào tạo YOLO v7	79
Phụ lục hình 37 Thực nghiệm mô hình YOLO v7 với hình ảnh	79
Phụ lục hình 38 tập dữ chuẩn bị chứa trong raw_data.....	80
Phụ lục hình 39 kết quả chia train và test	80
Phụ lục hình 40 Code chia tập train và test.....	81
Phụ lục hình 41 Convert train test thành ‘.csv’ (a)	81
Phụ lục hình 42 Convert train test thành ‘.csv’ (b).....	82
Phụ lục hình 43 File train_labels.csv	82
Phụ lục hình 44 File val_labels.csv.....	82
Phụ lục hình 45 generate_tfrecord.py	83
Phụ lục hình 46 chỉnh sửa VOC_LABELS trong file generate_tfrecord.py	83
Phụ lục hình 47 cài đặt file generate_tfrecord.py	83
Phụ lục hình 48 Chuyển ‘csv’ thành ‘record’	84
Phụ lục hình 49 file label_map.txt	84
Phụ lục hình 50 cài đặt model.....	84
Phụ lục hình 51 Biên dịch.....	84
Phụ lục hình 52 Cài đặt API	84
Phụ lục hình 53 Cài đặt API thành công.....	85
Phụ lục hình 54 Test API	85

Phụ lục hình 55 Tải MobileNet v2.....	85
Phụ lục hình 56 File pipeline.config	86
Phụ lục hình 57 Lệnh đào tạo mô hình SSD.....	86
Phụ lục hình 58 Đánh giá.....	87
Phụ lục hình 59 đánh giá mô hình trên tensorflow	87
Phụ lục hình 60 Xuất Model SSD.....	87
Phụ lục hình 61 Kết quả saved_model.pb xuất thành công	88
Phụ lục hình 62 Load model saved_model.pb và import các thư viện	89
Phụ lục hình 63 chạy hàm inference	89
Phụ lục hình 64 Chạy hàm nhận dạng	90
Phụ lục hình 65 Cấu trúc thư mục dataset	90
Phụ lục hình 66 chỉnh sửa file chuyển csv thành record.....	91
Phụ lục hình 67 file train.record tạo thành công	91
Phụ lục hình 68 File nhãn Faster-RCNN	91
Phụ lục hình 69 Cài đặt môi trường Faster-RCNN.....	92
Phụ lục hình 70 Tải mô hình ResNet50	92
Phụ lục hình 71 File ResNet50 tải thành công.....	93
Phụ lục hình 72 Đào tạo mô hình Faster-RCNN	94
Phụ lục hình 73 Lỗi đào tạo Faster-RCNN	94
Phụ lục hình 74 Sửa lỗi Faster-RCNN.....	94
Phụ lục hình 75 Hàm tải ảnh.....	95

DANH MỤC BẢNG

Bảng 2.1 Ví dụ về một Confusion Matrix.....	20
Bảng 3.1 Mô tả tập dữ liệu.....	24
Bảng 3.2 Mô tả các thông số cài đặt đào tạo YOLO v4	27
Bảng 3.3 Kết quả 5 lần đào tạo mô hình YOLO v4.....	27
Bảng 3.4 Kết quả thực nghiệm mô hình YOLO v4	29
Bảng 3.5 Mô tả thông số đào tạo YOLO v7	30
Bảng 3.6 Kết quả đào tạo mô hình YOLO v7.....	30
Bảng 3.7 Kết quả thực nghiệm mô hình YOLO v7	32
Bảng 3.8 Các thông số đào tạo mô hình SSD MobileNet v2.....	33
Bảng 3.9 Kết quả đào tạo mô hình SSD MobileNet v2	34
Bảng 3.10 Kết quả thực nghiệm mô hình SSD MobileNet v2.....	36
Bảng 3.11 Các thông số đào tạo mô hình Faster-RCNN	37
Bảng 3.12 Kết quả đào tạo mô hình Faster-RCNN	37
Bảng 3.13 Kết quả thực nghiệm mô hình Faster-RCNN	39
Bảng 3.14 Kết quả đánh giá mô hình phát hiện đối tượng một giai đoạn	40
Bảng 3.15 Kết quả đánh giá mô hình phát hiện hai giai đoạn	40
Bảng 3.16 Kết quả đào tạo mô hình YOLO v4 kỹ thuật tăng cường dữ liệu	41
Bảng 3.17 Mô tả thông số cải tiến YOLO v4	42
Bảng 3.18 kết quả đào tạo mô hình YOLO v4 cải tiến tham số	43
Bảng 3.19 Cấu hình máy tính.....	45
Bảng 4.1 Mô tả kịch bản hệ thống	48
Bảng 4.2 Đánh giá kết quả mô hình bằng bảng ma trận nhầm lẫn	52

DANH MỤC TỪ CHUYÊN NGÀNH

Viết tắt **Giải thích**

CV	Thị giác máy tính (Computer Vision)
CNN	Mô hình mạng no-ron tích chập (Convolution Neural Network)
YOLO	Thuật toán phát hiện đối tượng YOLO (You Only Look Once)
SSD	Thuật toán phát hiện đối tượng SSD (Single Shot Detector)
ResNet	Mạng ResNet (Residual Network)
R-CNN	Thuật toán phát hiện đối tượng RCNN (Region Convolutional Neural Networks)
FPS	Độ đo tốc độ xử lý khung hình trên giây (Frame Per Second)
RPN	Mạng đề xuất khu vực (Region Proposal Network)
IoU	Chỉ số đánh giá mô hình phát hiện đối tượng, sử dụng để đo độ chính xác (Intersection over Union)

CHƯƠNG 1. GIỚI THIỆU.

1.1. Đặt vấn đề

Hiện nay, công nghệ thông tin ngày càng được chú trọng và được quan tâm hơn hết từ tất cả con người trên toàn thế giới. Trong số đó là họ luôn quan tâm và mong muốn có được một ngôi nhà thông minh. Một ngôi nhà được gọi là thông minh là một ngôi nhà được lắp đặt các thiết bị điều khiển, hệ thống an ninh, sử dụng những thiết bị có kết nối internet, đặc biệt hơn là camera giám sát theo dõi hành trình được camera ghi hình lại theo thời gian thực giúp cho người dùng có thể xem lại những diễn biến đã xảy ra. Mục đích của camera giám sát là chủ yếu cho mục đích an ninh. Chính vì mục đích an ninh nên camera giám sát được đồng đảo mọi người lựa chọn sử dụng. Nhưng đối với việc lựa chọn camera giám sát với mục đích giám sát kẻ trộm đột nhập vào nhà cửa thì trở nên khó khăn và đáng lo ngại khi người dùng họ mong muốn biết kẻ trộm vào nhà tức thì.

Chính vì những vấn đề khó khăn và đáng lo ngại đó đè tài "**Hệ Thống Phát Hiện Hành Vi Leo Rào**" được đề xuất nghiên cứu và xây dựng lên hệ thống tích hợp camera AI nhằm phát hiện các đối tượng có hành vi leo rào và tự động gửi thông báo đến người dùng khi có hành vi leo rào. Hệ thống giúp cho người dùng không ngại rời khỏi nhà, hệ thống vẫn đảm nhiệm việc theo dõi giám sát các hành vi của con người và gửi thông báo đến người dùng bất kỳ ở nơi đâu khi có đối tượng leo rào. Hệ thống được xây dựng với chi phí thấp mặc dù khoảng cách phát hiện vẫn còn hạn chế.

1.2. Các nghiên cứu liên quan

Trong những năm gần đây, lĩnh vực thị giác máy tính trở nên một lĩnh vực đang được nhiều người quan tâm và khai thác trong và ngoài nước. Chính vì sự tác động từ môi trường bên ngoài của cuộc sống như hành vi trộm cắp tài sản ở các nơi bệnh xá, nhà, trường, những nơi công cộng ngày càng nhiều và phổ biến, vì thế những thiết bị giám sát như camera là một phần quan trọng và thiết yếu. Bên cạnh đó những bài toán về lĩnh vực thị giác máy tính cũng được khai thác và được áp dụng rộng rãi với nhiều mục đích khác nhau. Những thiết bị phát hiện các hành vi bất thường của con người, hệ thống đo thân nhiệt, hệ thống phát hiện hành vi trộm cắp, hành vi phát hiện sâu bệnh trên lá. Dưới đây là một số thành tựu các nhóm tác giả đã nghiên cứu và mang lại tính ứng dụng vào thực tế.

Tác giả [1] đã đề xuất một kiến trúc để phát hiện những hành vi bất thường của con người. Theo [1] tác giả tập trung vào việc phát hiện hành vi bất thường của con người trong nhà, đầu tiên tác giả sử dụng mô hình Gaussian để phân đoạn ảnh nền của từng khung hình trong video và thuật toán FCM (thuật toán phân cụm dữ liệu) để phát hiện các ngoại lệ trong

các mẫu dữ liệu. [2] tác giả nghiên cứu sử dụng toán YOLO v4 để nhận diện và phân loại phương tiện giao thông trong luồng video trực tiếp, [2] được thử nghiệm trên tập video trực tiếp của camera giám sát, [2] cho kết quả thực nghiệm độ chính xác đạt khoảng 95% với phương tiện là ô tô và trên 80% với phương tiện là xe máy. Trong [3] tác giả tập trung nghiên cứu vào việc phát hiện tai người theo thời gian thực, [3] sử dụng YOLOV3 để phát hiện tai người và kết hợp RetinaFace để nâng cao độ chính xác, phương pháp được tác giả đánh giá trên tập dữ liệu không ràng buộc. [4] tác giả sử dụng thuật YOLOv3 kết hợp mạng cơ sở VGG19 phát hiện và nhận dạng biển báo giao thông trong nhiều hình ảnh, [4] kết quả cho thấy độ chính xác của mô hình đạt hơn 90%, [4] cho biết rằng với sự kết hợp của YOLOV3 với mạng VGG16 cho kết quả vượt trội cho tất cả các loại biển báo giao thông trong điều kiện khác nhau. [5] Phát hiện ngã của người cao tuổi dựa trên việc trích xuất khớp của con người và phát hiện đối tượng, [5] nghiên cứu vào 3 trạng thái của người cao tuổi gồm 3 trạng thái như ngã, nằm và các trạng thái khác, tác giả đã sử dụng giải thuật YOLO v4 để giải quyết bài toán và kết quả thực nghiệm của tác giả khi phát hiện một ông già có trạng thái ngã đạt 99.3% và độ nhạy và độ đặc hiệu của mô hình đạt 79.3% và 72.1% [5].

Trong [6] tác giả có sự so sánh giữa mô hình SSD MobileNet v2 và YOLO v3 về đề tài thiết bị phát hiện đeo khẩu trang và cảm biến đo thân nhiệt trên nền tảng Chip SoC, kết quả đạt được tác giả cho biết mô hình SSD MobileNet v2 cho kết quả tốt và tối ưu về hiệu suất để phát hiện người có đeo khẩu trang trên thiết bị di động như raspberry pi. [7] Nhóm tác giả nghiên cứu các phương pháp phát hiện đối tượng một giai đoạn ưu tiên về tốc độ như YOLO, SSD và RetinaNet và phương pháp hai giai đoạn như Faster R-CNN, Mask R-CNN, và Cascade R-CNN, [7] Faster-RCNN và SSD có độ chính xác cao hơn trong khi YOLO hoạt động tốt hơn khi tốc độ được ưu tiên hơn là độ chính xác, mục đích của [7] là nghiên cứu xây dựng phương pháp phát hiện đối tượng trong đó SSD kết hợp với mạng cơ sở được đào tạo sẵn trước là MobileNet cho kết quả nhanh và chính xác.

Tác giả [8] đã so sánh giữa SSD và Faster-RCNN về phát hiện đối tượng trong ảnh, kết quả thử nghiệm trên bộ dữ liệu PASCAL VOC, COCO và ILSVRC xác nhận rằng SSD có độ chính xác cao, [8] kết quả với input đầu vào kích thước 300x300 đạt 74.3% mAP trên VOC2007 và kết quả đầu vào là 512x512 đạt được 76.9% mAP. [9] sử dụng thuật toán SSD để nhận dạng hành vi của con người từ video giám sát [9] tác giả cũng có sự so sánh thuật toán SSD và mạng thần kinh CNN [9] cho thấy rằng SSD có độ chính xác nhận dạng cao với tốc độ phát hiện khung hình 0.146 fps và độ chính xác đạt 82.8% trên tập dữ liệu khác nhau. [10] sử dụng giải thuật SSD để phục vụ việc phát hiện hành vi đeo khẩu trang trên thời gian thực, [10] đào tạo mô hình để phát hiện 3 đối tượng trên 853 tập dữ liệu ảnh: người đeo khẩu trang, người đeo khẩu trang không đúng cách và người không đeo khẩu trang, kết quả mAP đạt được 70.2% trên tập dữ liệu Pascal VOC và fine-tune (tinh chỉnh).

[11] Tác giả nghiên cứu sử dụng giải thuật SSD phát hiện đám cháy và báo động trên thời gian thực được tác giả tích hợp vào hệ thống giám sát khu vực bằng thiết bị bay không người lái [11] đạt kết quả độ chính xác trung bình là 92.7% và tốc độ là 26 fps trên SSD MobileNet thay vì sử dụng VGG-16/ResNet.

Có thể thấy rằng các mô hình phát hiện và nhận dạng đối tượng trong lĩnh vực thị giác máy tính đã mang lại cho con người nhiều công trình ứng dụng đáp ứng những nhu cầu khác nhau mà nó mang lại.

1.3. Mục tiêu đề tài

Mục tiêu đề tài là xây dựng hệ thống phát hiện các hành vi leo rào và gửi thông tin cảnh báo đến người dùng.

1.4. Đối tượng và phạm vi đề tài

Đề tài tập trung vào việc phát hiện các đối tượng người có hành vi leo rào qua các đoạn video và hình ảnh. Các loại rào bao gồm rào sắt, rào cây, rào bức tường.

Các video có độ phân giải như 1920x1080, 1280x720, 852x480. Khoảng cách từ camera đến rào là 5m.

1.5. Nội dung đề tài

Đề tài tập trung vào các nghiên cứu sau:

- Tìm hiểu và nghiên cứu về kiến thức mô hình học sâu, các bài toán hỗ trợ phát hiện đối tượng YOLO v4, YOLO v7, SSD và Faster-RCNN.
- Thu thập dữ liệu ảnh và video chứa đối tượng con người có hành vi leo rào, hành vi bình thường (đứng, ngồi, chạy).
- Đào tạo các mô hình phát hiện đối tượng.
- Thủ nghiệm, đánh giá kết quả thử nghiệm với các video, hình ảnh có chứa đối tượng con người có hành vi leo rào và không leo rào trên mô hình đề xuất.
- Nghiên cứu ngôn ngữ lập trình Python, thư viện Tkinter và CustomTkinter hỗ trợ xây dựng giao diện.

1.6. Những đóng góp chính của đề tài

Đóng góp chính của đề tài:

- Thu thập được 5430 tập dữ liệu ảnh chứa các hành vi leo rào và hành vi bình thường.
- Thu thập được 254 tập dữ liệu ảnh trong môi trường nhiều và ít nhiễu dùng để đánh giá mô hình.

1.7. Bộ cục của luận văn

Chương 1: Giới Thiệu.

Chương 2: Mô Tả Bài Toán.

Chương 3: Thiết Kế và Cài Đặt Giải Pháp.

Chương 4: Kiểm Thử và Đánh Giá.

Chương 5: Kết Luận và Hướng Phát Triển.

1.8. Tổng kết chương

Trong chương này mô tả những vấn đề xoay quanh đề tài những tình huống leo rào trộm cắp tài sản diễn ra trong cuộc sống thường ngày trở nên phổ biến từ đó hệ thống phát hiện hành vi leo rào được đề xuất và nghiên cứu. Giới thiệu sơ lược một số các công trình nghiên cứu đã được áp dụng rộng rãi vào thực tiễn xoay quanh các mô hình của đề tài. Những mục tiêu kết quả của hệ thống cần đạt được và kết quả đóng góp của đề tài đã đạt được cũng đã đề cập và mô tả chi tiết trong chương này.

CHƯƠNG 2. MÔ TẢ BÀI TOÁN

2.1. Mô tả chi tiết bài toán

Đề tài ‘**Hệ thống phát hiện hành vi leo rào**’ với mục đích phát hiện được đối tượng người có hành vi leo rào dựa trên đoạn video phát trực tiếp từ camera giám sát, các đoạn video có độ phân giải cao thấp khác nhau và hình ảnh. Thông qua đó hệ thống tiến hành tự động gửi cảnh báo đến người dùng khi có hành vi leo rào.

2.2. Cơ sở lý thuyết

2.2.1. Deep Learning (Mô hình học sâu)

Mô hình học sâu là một nhánh của mô hình máy học, tập trung vào việc giải quyết vấn đề liên quan đến mạng thần kinh, mô hình học sâu cũng được hiểu như là một bộ não của máy tính cho những hệ thống thông minh khi cho nó tự học hỏi với nhiều tập dữ liệu khác nhau, mô hình có khả năng ghi nhớ tốt sẽ được chọn tích hợp vào các hệ thống nhằm phục vụ cho con người áp dụng vào các công trình thực tiễn với mục đích ý nghĩa khác nhau. Mỗi mô hình thuộc mô hình học sâu sẽ có những ứng dụng và đáp ứng các yêu cầu khác nhau. Ví dụ các ứng dụng thực tiễn mà mô hình học sâu mang lại như: Hệ thống lái xe tự động, trợ lý ảo, nhận diện hình ảnh, trích xuất văn bản từ âm thanh và chuẩn đoán liên quan đến lĩnh vực y tế, v.v...

Các bài toán trong lĩnh vực thị giác máy tính (Computer Vision) như bài toán phân lớp, bài toán định vị đối tượng, bài toán phát hiện đối tượng và bài toán phân vùng ảnh.

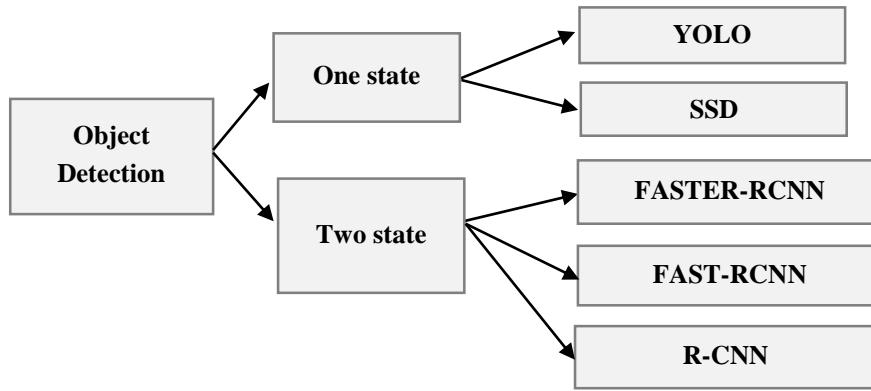
Bài toán Classification (phân lớp ảnh): Đây là bài toán dự đoán nhãn của một đối tượng khi cho đầu vào là một ảnh chứa một đối tượng và đầu ra đối tượng trong ảnh đó thuộc nhãn nào (hay nói cách khác đầu ra là dự đoán nhãn cho đối tượng).

Bài toán Object localization (định vị đối tượng): là một bài toán xác định vị trí của đối tượng, đối tượng phát hiện được sẽ có một hộp hình chữ nhật bao quanh đối tượng đó.

Bài toán Object Detection (phát hiện đối tượng): là sự kết hợp của 2 bài toán phân lớp ảnh và bài toán định vị đối tượng, nhưng có sự khác biệt hơn là bài toán phát hiện đối tượng không chỉ đơn thuần là phát hiện và tạo hộp giới hạn cho một đối tượng đang được quan tâm mà bài toán còn phân biệt được nhiều đối tượng khác nhau, mỗi đối tượng được phát hiện sẽ có tên của đối tượng đó và hộp giới hạn tương ứng.

Bài toán Image Segmentation (phân vùng ảnh): bài toán phân vùng ảnh có tác dụng phân vùng các đối tượng được quan tâm ra thành nhiều phần khác nhau.

Trong đề tài ***Phát hiện hành vi leo rào*** tập trung nghiên cứu đến bài toán phát hiện đối tượng nhằm phát hiện được đối tượng có hành vi leo rào và hành vi bình thường. Các bài toán phát hiện đối tượng được mô phỏng như **Hình 2.1**.



Hình 2.1 Các thuật toán thuộc bài toán phát hiện đối tượng

Bài toán phát hiện một giai đoạn (one stage): mô hình phát hiện đối tượng trên các vùng đề xuất, bao gồm các bài toán YOLO và SSD.

Bài toán phát hiện hai giai đoạn (two stage): sau khi phát hiện được các đối tượng từ vùng đề xuất, các đối tượng này được trích xuất ra thành từng phần đối tượng đưa vào một mạng khác để dự đoán xác suất phát hiện đối tượng, bao gồm các bài toán R-CNN, FAST-RCNN và FASTER-RCNN.

2.2.2. Mạng nơ-ron tích chập CNN

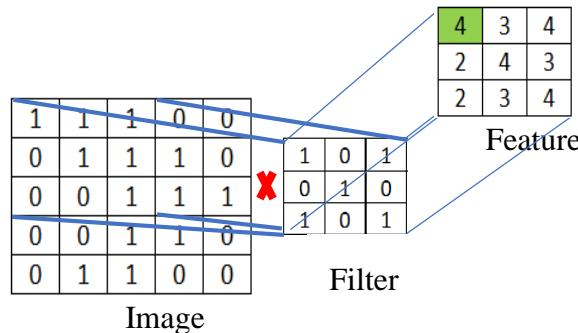
Hiện nay có rất nhiều hệ thống thông minh được ra đời phục vụ vào đời sống xã hội ngày càng nhiều nhờ vào tính hiệu quả, tốc độ tính toán và độ chính xác của mạng CNN mang lại.

Về mặt kỹ thuật mạng CNN bao gồm các thành phần: Lớp tích chập (Convolution layer), Lớp tổng hợp (Pooling Layer) và lớp kết nối đầy đủ (Fully Connected Layer)

Lớp tích chập (Convolution Layer)

Đây là lớp đầu tiên cũng là lớp quan trọng nhất trong mô hình mạng CNN là thành phần đảm nhiệm việc phát hiện và tính mọi phép toán.

Lớp tích chập nhận đầu vào là một ma trận hình ảnh sau đó ma trận này được áp dụng với nhiều bộ lọc khác nhau. Sử dụng phép tích chập giữa ảnh đầu vào với các bộ lọc để tạo ra bản đồ đặc trưng.

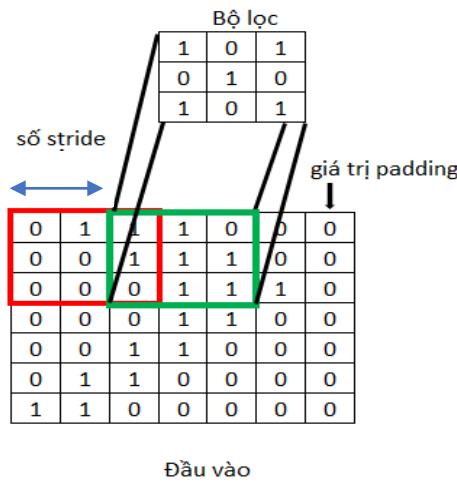


Hình 2.2 Ví dụ tích chập ảnh ma trận 5×5 với bộ lọc 3×3

Như ví dụ **Hình 2.2** thể hiện tính tích chập nhận đầu vào là một ma trận ảnh kết hợp với bộ lọc kích thước 3×3 . Hai ma trận này tiến hành thực hiện phép tích chập từ trái sang phải từ trên xuống dưới một khoảng trượt (stride), theo như ví dụ **Hình 2.2** khoảng cách trượt là 1. Mỗi lần thực hiện tích chập sẽ tạo ra 1 giá trị trong bản đồ đặc trưng. Quá trình tích chập hoàn thành sau khi nhận được một ma trận bản đồ đặc trưng hoàn chỉnh và làm đầu vào cho tầng tiếp theo.

Stride: Là khoảng cách dịch chuyển của bộ lọc sau mỗi lần tích chập trên từng vị trí ảnh. Ví dụ Stride = 2 thì mỗi lần tính tích chập bộ lọc tiến hành dịch chuyển sang phải 2 ô, việc dịch chuyển xuống dưới cũng tương tự

Padding: có giá trị 0 hoặc 1 được dùng làm bộ đệm cho ma trận đầu vào, padding được sử dụng phổ biến nhất là 0. Ví dụ về sử dụng khoảng trượt và padding được mô phỏng dưới **Hình 2.3**.

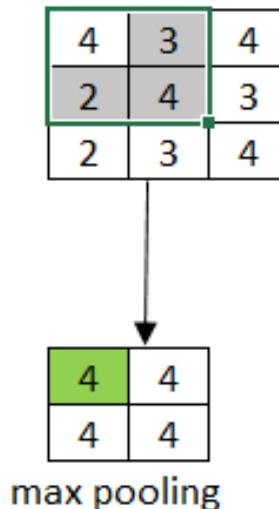


Hình 2.3 Ví dụ cách trượt stride = 2 và dùng đệm padding = 0

Hàm Relu: là một hàm tuyến tính được sử dụng nhiều nhất khi đào tạo mô hình. Có chức năng loại bỏ những giá trị âm và giữ lại các giá trị dương.

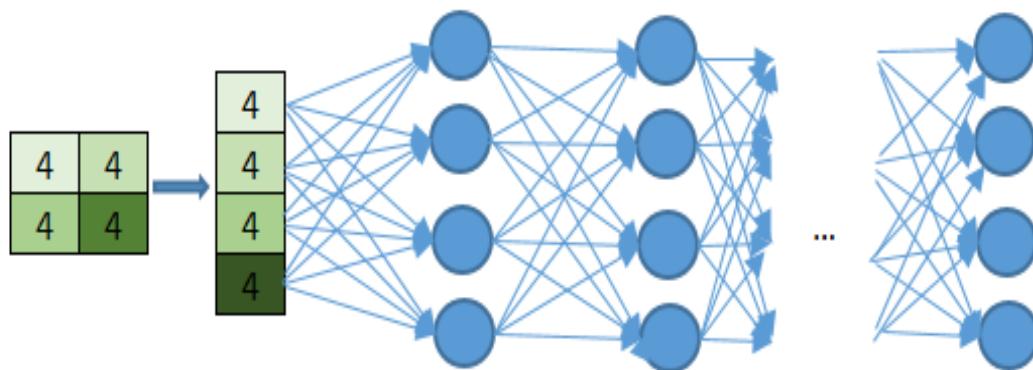
Lớp tổng hợp (Pooling Layer): Lớp tổng hợp được sử dụng sau tầng tích chập, tầng này làm đầu vào cho tầng tổng hợp có tác dụng làm giảm kích thước ma trận đầu vào. Việc giảm kích thước ở tầng này vẫn bảo toàn được các đặc trưng của dữ liệu đầu vào. Dạng tầng tổng hợp thường xuyên sử dụng là: Max Pooling và Avg Pooling

Max Pooling: trả về giá trị lớn nhất trong khu vực ảnh được chọn, ví dụ **Hình 2.4** về max pooling tích chập bộ lọc (2,2)



Hình 2.4 Ví dụ tầng pooling sử max pooling

Lớp kết nối đầy đủ (Fully Connected Layer): Lớp kết nối đầy đủ có chức năng tạo ra mô hình sau và phân loại đầu ra khi quá trình các ma trận làm phẳng kết nối đến các mạng neuron đã được học hỏi với nhau (xác thực chéo với nhau). Các ma trận được làm phẳng kết nối đến các mạng neuron được mô phỏng như **Hình 2.5**

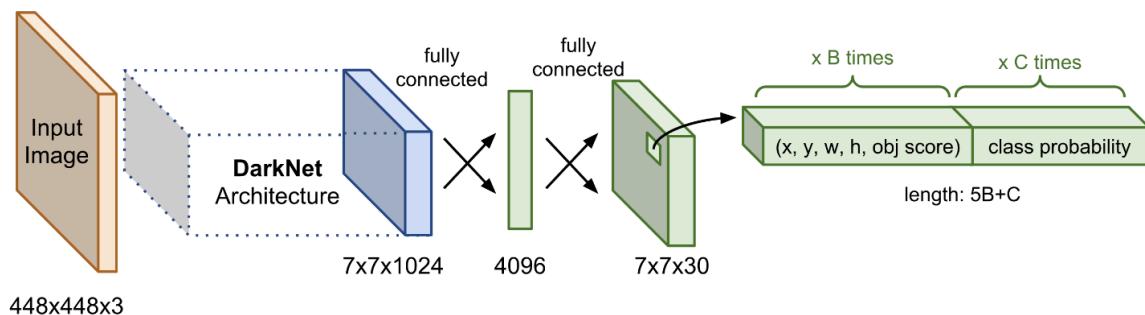


Hình 2.5 Tầng kết nối đầy đủ nhận đầu vào dữ liệu làm phẳng

2.2.3. Mô hình YOLO

Tính đến thời điểm hiện tại YOLO đã phát triển đến phiên bản thứ 7 (YOLO v7), mỗi phiên bản mang lại tính hiệu quả, tính cải tiến khác nhau. Theo tác giả về độ chính xác của YOLO vẫn còn hạn chế nhưng về tốc độ xử lý khung hình đã được cải thiện. YOLO là thuật toán trả lời câu hỏi đối tượng đang được quan tâm nằm ở vị trí nào và mỗi đối tượng đang quan tâm là đối tượng gì.

Kiến trúc YOLO được mô phỏng như **Hình 2.6** gồm hai thành phần chính là phần Base Network có nhiệm vụ tích chập trích xuất đặc trưng và phần Extra Layers (fully connected) có nhiệm vụ dự đoán xác suất và các tọa độ kích thước của đối tượng để tạo hộp giới hạn.



Hình 2.6 Kiến trúc mạng YOLO¹

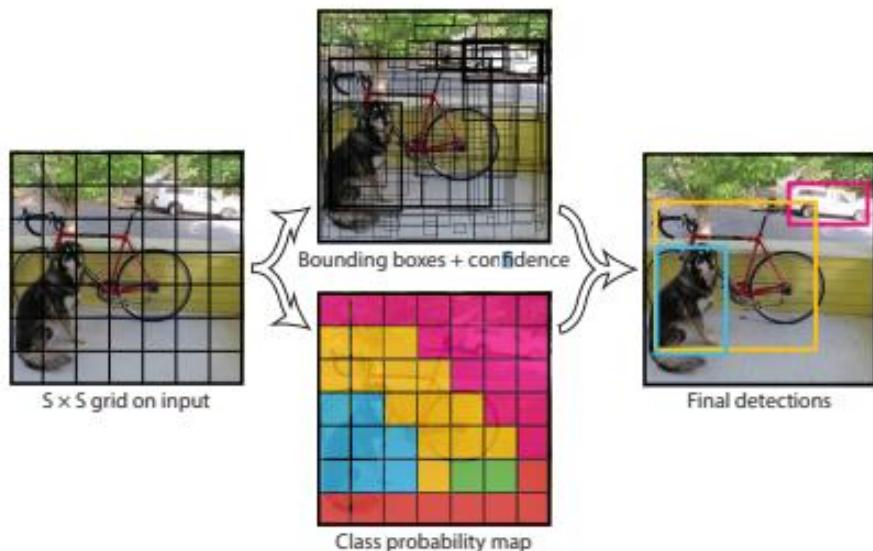
Mạng YOLO là một mạng con của CNN. Trong đó phần tích chập của CNN được YOLO thay bằng kiến trúc Darknet làm phần mạng cơ sở có nhiệm vụ trích xuất đặc trưng của ảnh và cho kết quả đầu ra là bản đồ đặc trưng (feature maps) kích thước $7 \times 7 \times 1024$ khi nhận đầu vào có kích thước 448×448 . Mô hình tiến hành tạo bản đồ đặc trưng kích thước $7 \times 7 \times 1024$ làm phẳng ma trận thành 4096 đặc trưng làm đầu vào cho phần Extra Layers (tầng kết nối đầy đủ) và đầu ra là tensor ma trận $7 \times 7 \times 30$, ma trận này sẽ được dự đoán và tính xác suất nhãn trên mỗi ô.

Cách thức hoạt động của YOLO: Mô hình nhận đầu vào là một hình ảnh sau đó mô hình tiến hành chia hình ảnh thành lưới có kích thước $S \times S$ ô và áp vào đó với các bộ lọc. Tiếp theo mô hình tiến hành tích chập để trích xuất các đặc trưng của ảnh đến giai đoạn đầu ra là bản đồ đặc trưng có kích thước $7 \times 7 \times 1024$ sau đó bản đồ đặc trưng này tiến hành được ma trận đầy thành một ma trận phẳng có 4096 đặc trưng đưa và tầng kết nối đầy đủ. Sau khi kết thúc 2 tầng kết nối đầy đủ mô hình cho đầu ra một tensor bản đồ đặc trưng kích thước $7 \times 7 \times 30$, trên mỗi ô của bản đồ đặc trưng mô hình tiến hành dự đoán trên từng ô nếu tâm của của đối tượng nằm trong ô đang được xem xét thì ô đó được xem là ô ứng viên.

¹https://phamdinhkhanh.github.io/assets/images/20200309_YOLOAlgorithm/h1.png

Trên mỗi ô dự đoán cho ra B bounding box và độ tin cậy của bounding box (obj score). Độ tin cậy này là xác suất trong ô đó có chứa đối tượng hay là không nếu ô đang được xem xét không chứa đối tượng thì độ tin cậy bằng 0 ngược lại độ tin cậy này được xác định bởi IoU ((Intersection over Union) giữa hai đường bao (predicted box and ground truth). Trong B bounding box này bao gồm 5 thành phần chính cần được làm rõ đó là: x, y, w, h, obj score. Trong đó (x,y) là tọa độ tâm của B (w,h) là chiều rộng và chiều cao và (obj score) là xác suất dự đoán chứa đối tượng có trong ô bounding box (hoặc độ tin cậy). Trên mỗi ô được dự đoán cũng dự đoán C xác suất (C là số lớp nhãn dự đoán) và C xác suất này dựa vào (obj score).

Tóm lại trên mỗi ô cần dự đoán $S \times S \times (5B + C)$ tensor. Giá trị số 5 đại diện cho 5 thông số (x,y,w,h,obj score)



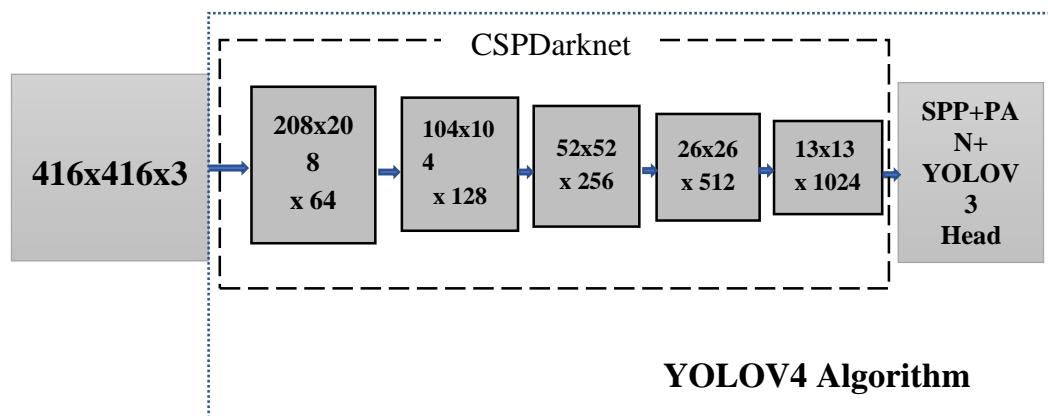
Hình 2.7 Hệ thống chia ảnh đầu vào thành SxS ô [12]

Hình 2.7 hệ thống chia ảnh đầu vào thành 7x7 ô. Mỗi ô cần dự đoán 3 bounding box và 3 xác suất lớp (chó, xe đạp và xe hơi) suy ra chúng ta cần dự đoán mỗi ô là $7 \times 7 \times (5 \times 3 + 3)$ tensor.

2.2.4. Mô hình YOLO v4 [13]

YOLO v4 được cải tiến và phát triển từ YOLO v3 được nghiên cứu bởi tác giả Alexey Bochkovskiy [13]. Tác giả tìm hiểu và thử nghiệm các phương pháp state-of-art trong các mô hình phát hiện đối tượng để đánh giá với các tính năng mới như WRC, CSP, CmBN, SAT, Mish activation, Mosaic data augmentation, CmBN, DropBlock regularization, and CIoU loss trên tập dữ liệu MS COCO đạt 43.5% AP (65.7% AP₅₀) và tốc độ xử lý trên thời gian thực đạt gần 65 FPS trên Tesla V100 [13].

Về cấu trúc mô hình YOLO V4 được mô phỏng như **Hình 2.8**



Hình 2.8 Kiến trúc mô hình YOLO V4

Về cấu trúc YOLO V4 bao gồm các thành phần: Input (Đầu vào), Backbone (Xương sống), Neck (Phần Cỗ), Head (Phần đầu).

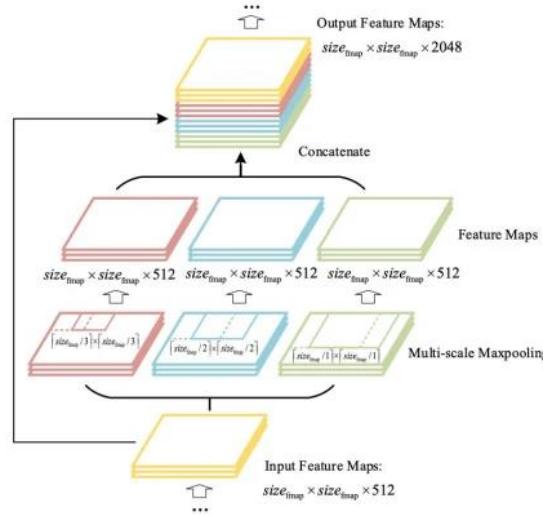
❖ **Phần Backbone (xương sống):** sử dụng kiến trúc CSPDarknet

Các dữ liệu làm đầu vào cho mô hình, được mô hình thay đổi về chung kích thước 416x416x3 là ảnh màu. Mô hình tiến hành thực hiện phép tích chập trên ảnh đầu vào với bộ lọc là 32 sử dụng tích chập 3x3, khoảng cách trượt là 1, sử dụng bộ đệm cho đầu vào có padding bằng 1 và hàm kích hoạt là Mish, thu được biểu đồ đặc trưng kích thước 208x208. Sau đó mô hình tiến hành tích chập trên biểu đồ đặc trưng thu được có kích thước 208x208 áp với bộ lọc là 64, tích chập là 3x3 và khoảng cách trượt là 1, padding sử dụng là 1, hàm kích hoạt là Mish. Mô hình tiếp tục tiến hành tích chập giảm kích thước đặc trưng đến đặc trưng thu được có kích thước 13x13x1024 và đưa vào mạng SPP+PAN (mạng SPP+PAN được mô tả chi tiết phần dưới), YOLO v4 sử dụng max-pooling để giảm kích thước biểu đồ đặc trưng. Và cuối cùng bản đồ đặc trưng thu được có kích thước 13x13x21. Sử dụng lại YOLO v3 để tiến hành dự đoán hộp giới hạn và nhãn.

❖ **Neck:** SPP, PAN [13]

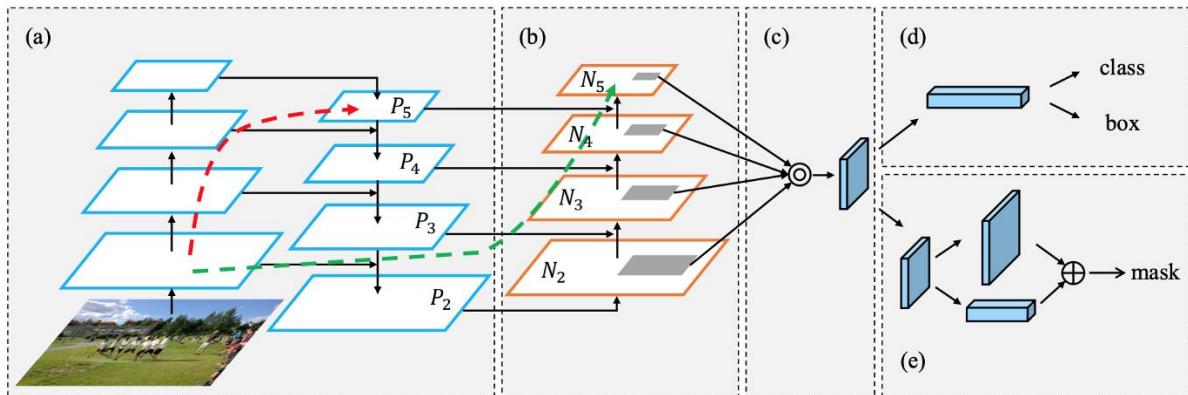
Mạng SPP này đã được sử dụng trong mô hình YOLO v3 (YOLO3-SPP) và với YOLO v4 được tác giả tiếp tục sử dụng mạng SPP này, nhưng đối với YOLO v3 trước đây mạng SPP nhận input bản đồ đặc trưng với các kích thước kernel size khác nhau và các input bản đồ đặc trưng chia thành các mảnh bản đồ đặc trưng khác nhau sử dụng Multi-scale Maxpooling để thu được các bản đồ đặc trưng có cùng kích thước, sau đó sử dụng toán tử Concatenate kết hợp các bản đồ đặc trưng thu được output bản đồ đặc trưng. Riêng đối với YOLO v4 không còn sử dụng cách tách các bản đồ đặc trưng thành các mảnh bản

đồ đặc trưng nữa mà YOLO-SPP này sử dụng tích chập Max-pooling để thu bén đồ đặc trưng và chuyển thẳng đến output bản đồ đặc trưng được xếp chồng lên nhau như **Hình 2.9**.



Hình 2.9 Mô hình YoLo-SPP²

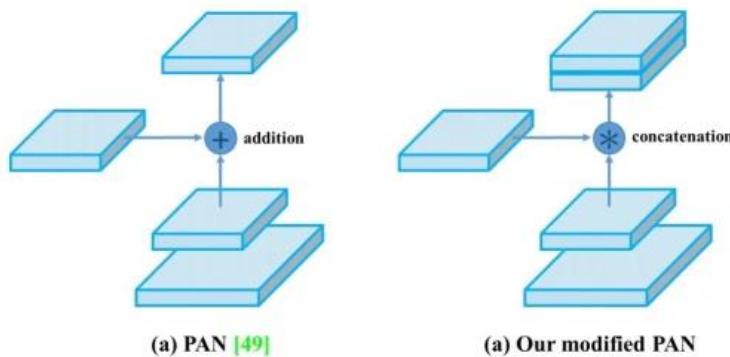
Mạng PAN là mô hình mạng cải tiến FPN. Nhờ mô hình mạng PAN mà các đối tượng được quan tâm có kích thước nhỏ sau mỗi lần downsample bản đồ đặc trưng được cải thiện và ít bị mất mát thông tin nhờ kỹ thuật top-down và bottom-up trong đó PAN được thêm một bottom-up nằm trên top-down của FPN cụ thể là N₄ xét theo chiều giảm kích thước lên như trong (b) của **Hình 2.10**



Hình 2.10 Mô hình mạng PAN [14]

Tại (b) trên **Hình 2.10** các layer này sẽ nhận đầu vào là bản đồ đặc trưng của tầng trước đó đi qua một tích chập 3x3. Output được gộp (concatenation) với bản đồ đặc trưng của top-down pathway.

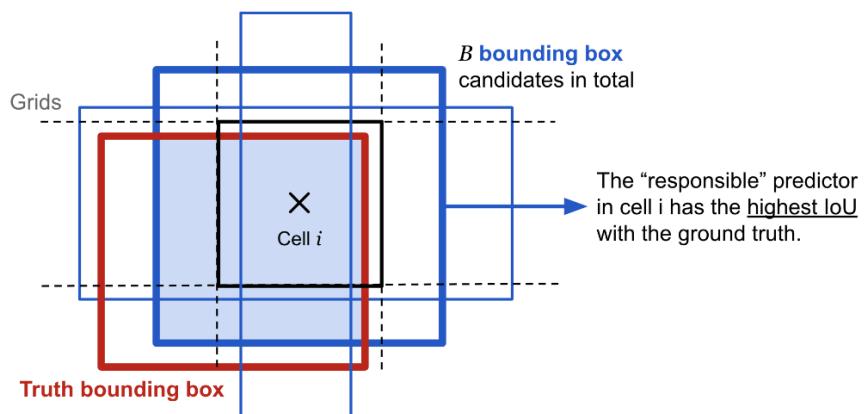
² https://dothanhblog.files.wordpress.com/2020/05/yolov4_spp.jpeg



Hình 2.11 Mô hình PAN cải tiến [13]

Hình 2.11 các bản đồ đặc trưng sử dụng phép gộp để tạo ra khối tầng bản đồ đặc trưng thay vì sử dụng addition. Các đặc trưng của đối tượng được quan tâm sẽ được đưa qua một mạng SPP tiến hành tích chập 4 lần lấy bản đồ đặc trưng, các bản đồ đặc trưng này tiến hành làm phẳng và đẩy vào tầng kết nối dày đủ tiến hành dự đoán.

Head: Trong phần Head này YOLO v4 sử dụng lại cách dự đoán hộp giới hạn và nhãn tương tự như YOLO v3 được mô phỏng như **Hình 2.12**.



Hình 2.12 dự đoán bounding box

Dự đoán hộp giới hạn: YOLO v3 sử dụng phép hồi quy logistic regression để dự đoán object score cho đối tượng, nếu bounding box được vẽ mà do mô hình phát hiện được (gọi là anchor) trùng với ground-truth bounding box hoặc có giá trị (IoU) lớn hơn các anchor còn lại thì object score bằng 1. Ngược lại nếu trong các anchor không có giá trị IoU nào lớn nhất hoặc thấp hơn ngưỡng IoU thì object score bằng 0 đồng nghĩa không xét đến.

Dự đoán nhãn: Một bounding box sẽ đoán cho nó một lớp nhãn (class), theo tác giả để tăng kết quả dự đoán tốt tác giả sử dụng các logistic classifiers độc lập với nhau thay vì sử dụng softmax trong bài toán multilabel classification [15] [16]

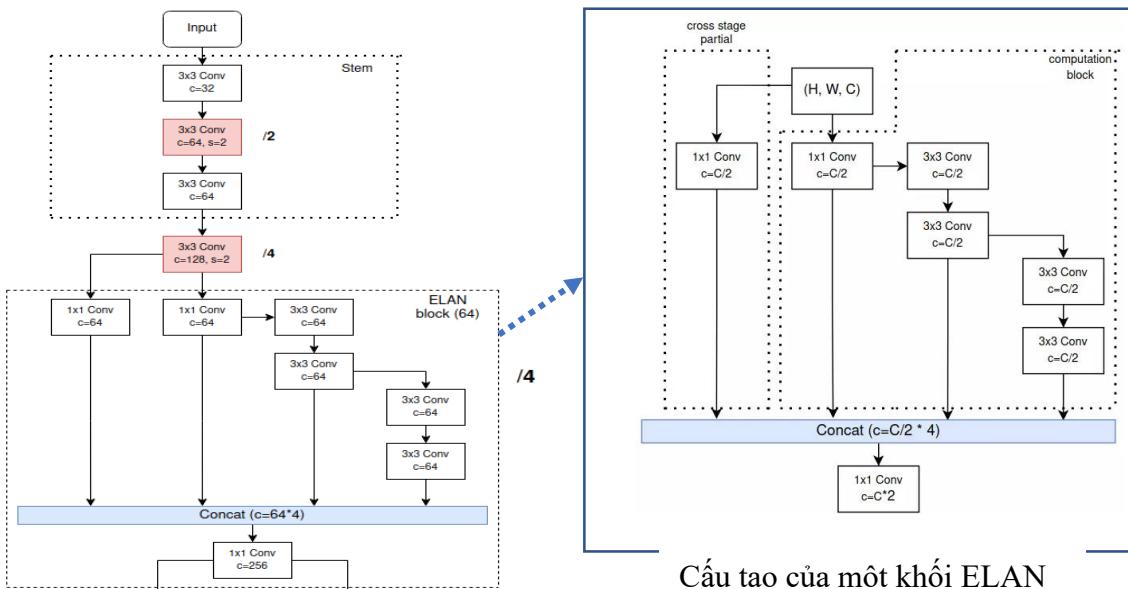
2.2.5. Mô hình YOLO V7

Mô hình YOLO v7 vượt qua mọi mô hình phát hiện đối tượng về tốc độ lẫn độ chính xác trong đó FPS từ 5 đến 160 và độ chính xác đạt cao nhất với 56.8% AP [17]. So với các mô hình YOLO trước đây YOLO v7 được đào tạo trên tập dữ liệu MS COCO từ đầu không còn sử dụng bộ dữ liệu nào khác hoặc các trọng số được đào tạo trước (pre-train).

Kiến trúc YOLO v7 bao gồm các thành phần:

- **Backbone (xương sống):** ELAN, E-ELAN [17]
- **Neck (cỗ):** SPPCSPC + PANet + RepConv [17]
- **Head:** YOLOR và Auxiliary head [17]

Phản Backbone: được tạo từ khói gốc (Stem Block), khói Stem này có nhiệm vụ trích xuất các đặc trưng từ ảnh đầu vào và làm đầu vào cho ELAN như **Hình 2.13**



Cấu tạo của một khói ELAN

Hình 2.13 Ảnh đầu vào đưa vào khói Stem và cấu tạo của ELAN [18]

Cấu tạo của ELAN gồm 3 phần: Cross Stage Partial, Computation Block và phép PointWiseConv [18]

Khi khói Stem tạo ra bản đồ đặc trưng và đưa vào khói ELAN, khói ELAN này được liên kết với nhau thông qua transition như **Hình 2.14**, mỗi một liên kết tới transition sẽ cho đầu ra bản đồ đặc trưng được giảm kích cỡ. Transition giống như tầng tống hợp (Polling Layer) có chức năng giảm kích thước bản đồ đặc trưng.



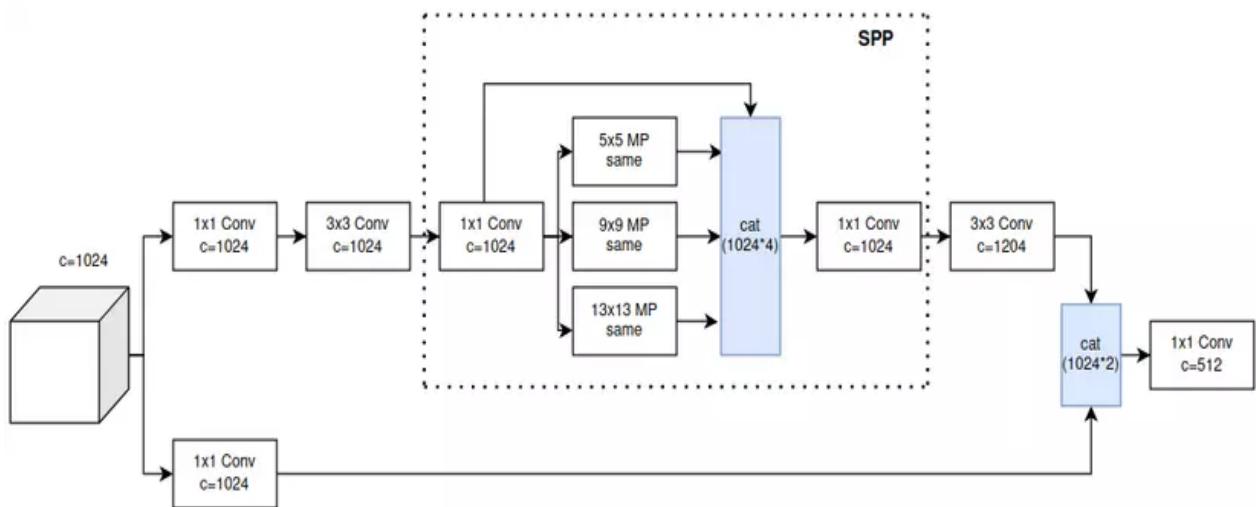
Hình 2.14 Các ELAN liên kết với nhau bằng transition

Phản Neck: CSPDPP + PANet + RepConv [18]

SPPCSPC: được cải tiến từ SPP trong YOLO v4

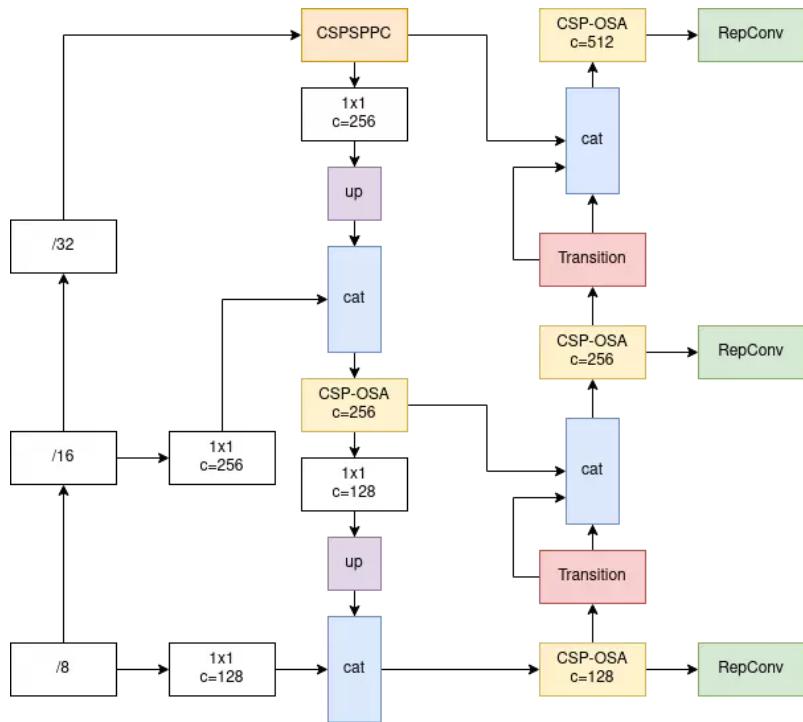
Kiến trúc SPPCSPC được mô phỏng trên **Hình 2.15**. Đầu tiên biểu đồ đặc trưng có số kênh $c=1024$ đưa vào kiến trúc SPPCSPC chia làm hai nhánh một nhánh tích chập 1×1 với số kênh $c=1024$ kết nối tắt đi thẳng đến toán tử concatenate để thực hiện phép toán với nhánh thứ 2 có số kênh được nhân đôi lên, nhánh còn lại tiến hành tích chập 1×1 và 3×3 với số kênh $c=1024$ sau đó thu được bản đồ đặc trưng có số kênh $c=1024$ và được kết nối đến với nhánh thứ nhất tạo ra bản đồ đặc trưng có kích thước kênh c bằng phân nửa với số kênh bản đầu đặc trưng ở bản đồ đặc trưng làm đầu vào [18].

Điều này cũng có thể khẳng định SPPCSPC nặng và chậm bởi vì khi đưa vào SPP, kiến trúc SSP đã giảm số kênh đi $\frac{1}{2}$ bản đồ đặc trưng đầu vào, còn SPPCSPC các số kênh đều giống với số kênh của bản đồ đặc trưng đầu vào.



Hình 2.15 Kiến trúc SPPCSPC [18]

PANet trong YOLO v7, các bản đồ đặc trưng được sinh ra từ các scale được đưa vào RepConv, trước khi đây vào RepConv các bản đồ đặc trưng sinh ra từ scale được cộng lại với nhau bằng toán tử concatenate sau đó sinh ra một bản đồ đặc trưng gian và tiếp tục được một mô hình (CSP-OSA [18]) xử lý. Minh PANet trong Neck của YOLO v7 minh họa như hình **Hình 2.16**.



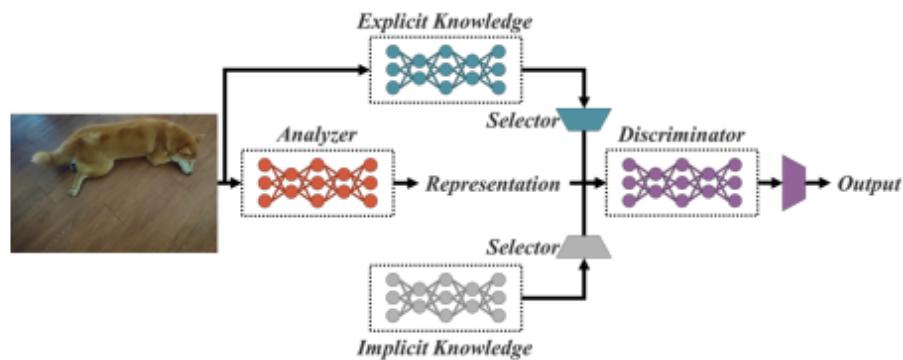
Hình 2.16 Kiến trúc Neck trong YOLO v7 [18]

RepConv: Các bản đồ đặc trưng được sinh ra từ CSP-OSA sẽ được đưa vào RepConv như **Hình 2.16**. RepConv sử dụng kỹ thuật Model re-parameterization [17] kết hợp với VGG cho hiệu suất cao hơn ResNet.

Phần Head

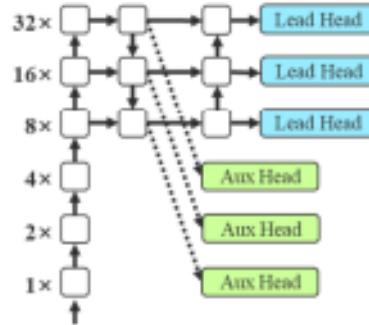
❖ YOLOR [19]

YOLOR áp dụng Implicit Knowledge **Hình 2.17** nhằm cải thiện độ chính xác và cải thiện tốc độ của mô hình. YOLOR đưa Implicit Knowledge vào vào mạng nơ-ron bằng 2 phép toán cộng và nhân để cho mô hình học hỏi tiếp xúc trực tiếp với các input [18], trong quá trình đào tạo Implicit Knowledge sẽ tự rút ra được những thứ mà nó đã học được. YOLOR biểu diễn Implicit Knowledge dưới dạng vector [18].



Hình 2.17 Implicit Knowledge trong YOLOR [19]

❖ **Auxiliary Head:** Có nhiệm vụ bắt dự đoán mô hình trên các thông tin từ lớp nông và bản đồ đặc trưng được sinh ra từ mô hình đem đi thực hiện dự đoán có dạng multi-scale [18].

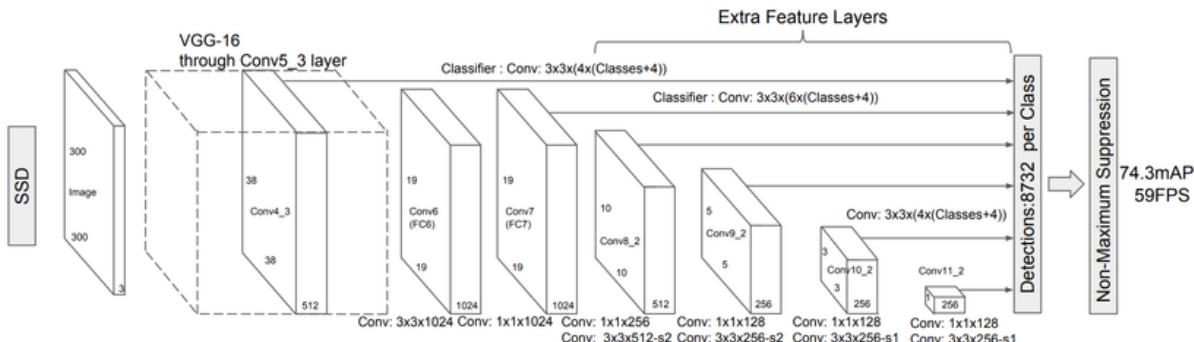


Hình 2.18 Mô hình Auxiliary head [17]

Như **Hình 2.18** mô hình loại bỏ các Auxiliary head chỉ sử dụng Lead Head (Head gốc) để đưa ra dự đoán trong quá trình suy luận.

2.2.6. Mô hình SSD

Bên cạnh thuật toán YOLO thì SSD cùng thuộc nhóm bài toán phát hiện đối tượng trong một giai đoạn.



Hình 2.19 Kiến trúc mô hình SSD³

Trên **Hình 2.19** về kiến trúc SSD gồm 2 thành phần: Phần trích xuất đặc trưng trong phần base network (mạng cơ sở) thuộc tầng Conv5_3 đã loại bỏ các lớp cuối cùng và các mạng hỗ trợ cho phần base network hay còn gọi là tầng SSD thay thế các lớp của phần mạng cơ sở đã loại bỏ. Việc kết hợp dùng các mạng hỗ trợ này cho phần base network để tạo ra các bản đồ đặc trưng có kích thước khác nhau và có tác dụng phát hiện và dự đoán đối tượng trên từng bản đồ đặc trưng đó.

³ <https://images.viblo.asia/1a86f5a6-0da5-4459-adba-0f1b96b6ff76.png>

Trong nghiên cứu này phần mạng cơ sở được sử dụng là kiến trúc MobileNet v2 để trích xuất đặc trưng từ ảnh đầu vào. Cụ thể

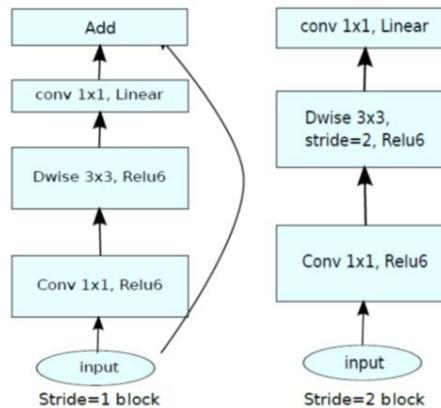
Ảnh đầu vào kích thước $300 \times 300 \times 3$ (ảnh màu). Tại tầng CONV5_3 sử dụng kiến trúc MobileNet v2 và tạo ra bản đồ đặc trưng kích thước 38×38 với bộ lọc 512 ($38 \times 38 \times 512$).

Tại Conv4_3 mô hình tiến hành thực hiện phép tích chập như mạng CNN trên bản đồ đặc trưng kích thước $38 \times 38 \times 512$ và cho ra có bản đồ đặc trưng kích thước $19 \times 19 \times 1024$. Sau khi thu được bản đồ đặc trưng mô hình tiến hành áp dụng classifier bộ lọc tích chập $3 \times 3 \times (4(3+4))$ tương ứng với $3 \times 3 \times (4 \times (\text{classes} + 4))$, (trong đó 3×3 là lớp tích chập, giá trị 4 là số bounding dự đoán, classes là số lớp nhãn là + thêm lớp nhãn background và giá trị 4 cuối cùng tương ứng với 4 offset x,y,w,h). Quá trình này cho ra các vị trí hộp giới hạn trên các ô của bản đồ đặc trưng. Conv7 mô hình thực hiện tương tự như Conv4.

Các lớp thuộc Extra feature layers (các mạng hỗ trợ phần mạng cơ sở) trích xuất đặc trưng và giảm kích thước ảnh đầu vào. Mỗi lần thu được bản đồ đặc trưng mô hình tiến hành đưa vào nhận dạng và tạo hộp giới hạn dự đoán xác suất đối tượng.

Kiến trúc MobileNet V2 [20]

SSD sử dụng kiến trúc MobileNet v2 nhằm mục đích tăng tốc đào tạo và suy luận mô hình làm cho mô hình có tốc độ xử lý và tính toán nhanh.

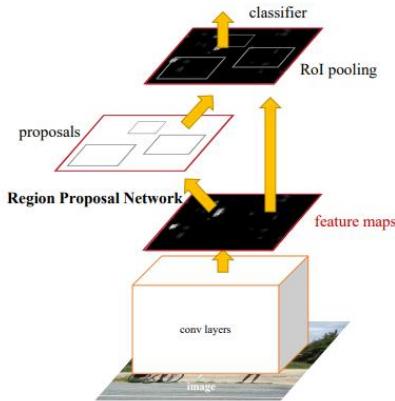


Hình 2.20 Kiến trúc MobileNet v2 [20]

Kiến trúc mạng MobileNet v2 bao gồm các tầng đầu vào, tầng tích chập, hàm kích hoạt và tầng kết nối đầy đủ. Kiến trúc MobileNet v2 bao gồm 2 block theo như **Hình 2.20**. Block thứ nhất có stride bằng 1 và block thứ hai có stride bằng 2. Đầu vào của mô hình được kết nối tắt bằng cách đưa đầu vào của mô hình cộng vào đầu ra của khối dư như **Hình 2.20**.

Mô hình Mobilenet v2 chứa 32 bộ lọc lớp tích chập và 19 lớp nút cỗ chai, sử dụng hàm kích hoạt phi tuyến tính Relu6 vì độ bền của mô hình [20].

2.2.7. Mô hình Faster-RCNN [21]



Hình 2.21 Kiến trúc mô hình Faster-RCNN [21]

Faster-RCNN là mô hình cải tiến từ Fast-RCNN. Kiến trúc mạng Faster-RCNN được mô tả như **Hình 2.21**. Kiến trúc Faster-RCNN gồm 2 thành phần

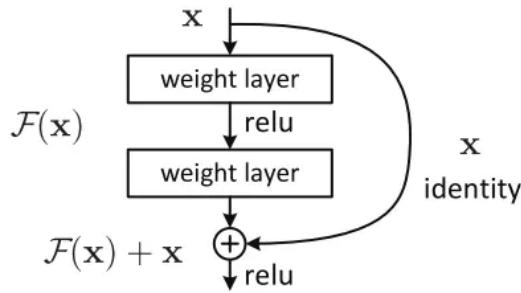
Vùng tìm các vùng đề xuất (region proposal) trên bản đồ đặc trưng của ảnh đầu vào hay còn gọi là Region Proposal Network (RPN)

Vùng Fast R-CNN là mạng CNN có nhiệm vụ trích xuất các đặc trưng từ các vùng đề xuất của RPN sau đó các đặc trưng của ảnh này sẽ qua lớp ROI Pooling và dự đoán các bounding box và nhãn cho từng bounding box đó.

RPN nhận đầu vào là một bản đồ đặc trưng của hình ảnh khi cho qua một backbone (mạng cơ sở) của mạng CNN thường là các pre-train model đã được đào tạo. Trong đề tài này sử dụng kiến trúc mạng ResNet50 để thu được bản đồ đặc trưng từ ảnh đầu vào. Sau đó một mạng con của RPN (region proposal) tìm các vùng có khả năng chứa đối tượng ra trên bản đồ đặc trưng của ảnh đầu vào và cho ra output của RPN là những bản đồ đặc trưng chứa các đối tượng có nhiều kích thước khác nhau sau đó các bản đồ đặc trưng có những kích thước khác nhau này qua một ROI Pooling để chuyển các bản đồ đặc trưng về cùng kích thước. Và các ROI này sẽ chuyển tiếp đến các lớp kết nối đầy đủ sau đó chia làm hai nhánh để dự đoán nhãn và hộp giới hạn

Mạng ResNet50 [22]

Mạng ResNet50 giống như kiến trúc mô hình MobileNet được trình bày ở phần **Mô hình SSD**. Theo **Hình 2.22** đầu vào của mạng được kết nối tắt cộng vào khối dư layer cuối cùng (hoặc đầu ra của của weight layer) bỏ qua các mạng trung gian. Mô hình sử dụng tích chập 3x3, hàm kích hoạt ReLu. Mạng ResNet có độ sâu lên tới 152 lớp khoảng 26 triệu tham số [22].

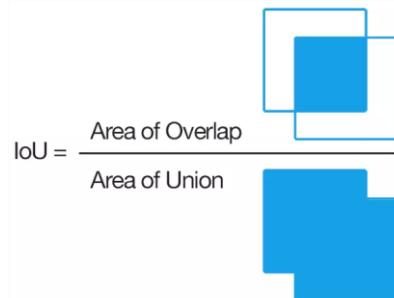


Hình 2.22 Mô hình ResNet50 [22]

2.3. Phương pháp đánh giá

Các độ đo chính xác (IoU), Precision, Recall, F1-Score được sử dụng để đánh giá mô hình. Trong đó với bài toán phát hiện hành vi leo rào thì IoU và F1-score là hai độ đo được quan tâm trong đề tài này.

Độ chính xác (IoU): được tính bằng tỉ lệ diện tích phần giao nhau (Area of Overlap) của hộp giới hạn được mô hình đã đào tạo dự đoán được (predicted bounding box) với hộp giới hạn đã được gắn nhãn trước đó (ground-truth bounding box) trên diện tích phần hợp giữa hộp giới hạn mô hình phát hiện được (predicted bounding box) với hộp giới hạn đã được gắn nhãn trước đó (ground-truth bounding box). **Hình 2.23** minh họa công thức IoU



Hình 2.23 Minh họa công thức tính IoU [23]

Precision: Là hàm đánh giá độ tin cậy của mô hình được tính bằng tỉ lệ giữa True Positive (TP) trên tổng của True Positive (TP) với False Positive (FP)

$$\text{Precision} = \frac{TP}{TP+FP} [23]$$

Bảng 2.1 Ví dụ về một Confusion Matrix

	Climbing (Positive)	Normal (Negative)
Climbing (Positive)	TP	FN
Normal (Negative)	FP	TN

Trong đó:

- TP là số lượng đối tượng được dự đoán đúng thuộc đối tượng Climbing
- FP là số lượng đối tượng Climbing bị dự đoán sai (nhầm lẫn) là Normal
- TN là số lượng đối tượng được dự đoán đúng thuộc đối tượng Normal
- FN là số lượng đối tượng Normal bị dự đoán sai (nhầm lẫn) là Climbing
- **Recall:** là hàm đánh giá khả năng nhớ lại của mô hình được tính bằng tỉ lệ giữa True Positive (TP) trên tổng của True Positive (TP) với False Negative (FN)

$$Recall = \frac{TP}{TP+FN} [23]$$

- **F1-score** là hàm đánh giá giữa precision và recall được tính bằng 2 lần tỉ lệ giữa tích của Precision và Recall với tổng của Precision và Recall

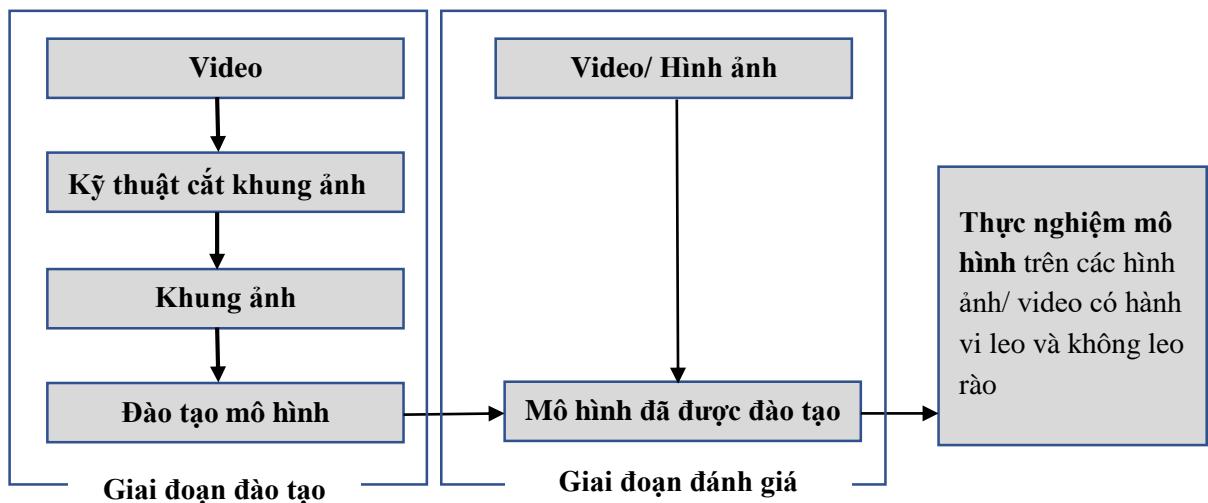
$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} [23]$$

2.4. Tổng kết chương

Trong chương này mô tả đến hướng tiếp cận các lý thuyết về mạng CNN các bài toán phát hiện đối tượng một giai đoạn và hai giai đoạn liên quan đến đề tài với mục tiêu phát hiện đối tượng có hành vi leo rào, trong đó các mô hình hỗ trợ cho việc phát hiện đối tượng như thuật toán YOLO v4, YOLO v7, SSD và Faster-RCNN và các phương pháp đánh giá mô hình cũng đã được mô tả giới thiệu.

CHƯƠNG 3. THIẾT KẾ VÀ CÀI ĐẶT GIẢI PHÁP

3.1. Đào tạo mô hình phát hiện hành vi leo rào



Hình 3.1 Kỹ thuật tổng quan đào tạo các mô hình

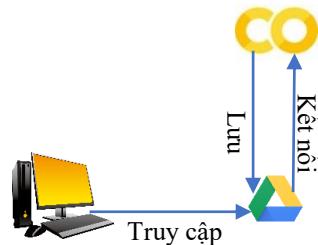
Hình 3.1 mô phỏng tổng quan đào tạo các mô hình YOLO v4, YOLO v7, SSD và Faster-RCNN, dữ liệu chuẩn bị đào tạo các mô hình được mô tả ở phần **3.1.2**.

Kỹ thuật đào tạo các mô hình gồm 3 giai đoạn: giai đoạn một đào tạo mô hình, giai đoạn hai đánh giá mô hình bằng các tập dữ liệu Video/ hình ảnh trên mô hình đã được đào tạo, giai đoạn ba thực nghiệm mô hình trên các đoạn video/ hình ảnh có hành vi leo và không leo rào.

3.1.1. Giới thiệu môi trường và tham số đào tạo

➤ Môi trường đào tạo mô hình

Sử dụng Google Colaboratory sản phẩm của Google hỗ trợ viết ngôn ngữ python, hỗ trợ việc đánh giá thực nghiệm các mô hình máy học. Cách sử dụng môi trường Google Colaboratory như **Hình 3.2**



Hình 3.2 Mô phỏng sử dụng Google Colaboratory

Sử dụng thư viện hỗ trợ máy học OpenCV, Numpy

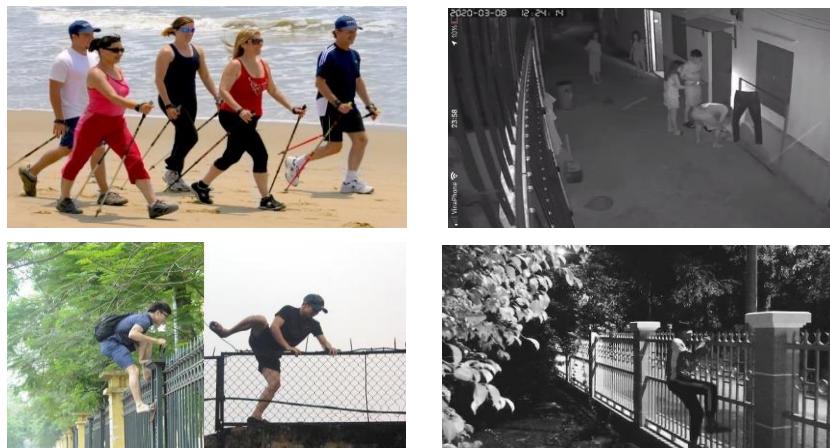
Sử dụng thư viện Tensorflow hỗ trợ việc tính toán và lấy kết quả đánh giá trong quá trình đào tạo mô hình.

➤ Khái niệm các kiểu thông số sử dụng trong đào tạo mô hình

- Input: dữ liệu đầu vào cho mô hình (ví dụ: Input: 416x416 đều này có ý nghĩa tất cả các ảnh được thay đổi kích thước đầu vào về cùng kích thước 416x416 trước khi đưa vào mô hình)
- Bath-size (Kích thước lô): Tập dữ liệu đào tạo mô hình có số lượng lớn vì thế cần phải chia lô ra để đưa vào mô hình tính toán. Việc này có tác dụng tránh làm cho mô hình gặp sự cố tràn bộ nhớ.
- Decay: là kỹ thuật giảm độ phức tạp của mô hình
- Learning-rate: tỉ lệ học tập cho mô hình, mô hình bắt đầu học với tỉ lệ học tập cao sau đó giảm theo số tỉ lệ học tập đã được thiết lập.
- Step_train (Epoch): Số vòng đào tạo mô hình.
- Number of class: Số lớp nhãn
- Ignore_thresh: Là thông số xác định so sánh ngưỡng IoU có lớn hơn ngưỡng này hay không để tránh bị một đối tượng được phát hiện có nhiều hộp giới hạn.

3.1.2. Giới thiệu tập dữ liệu

Các dữ liệu chuẩn bị cho việc đào tạo được thu thập từ nhiều nguồn (youtube, facebook, website), các video tự quay vào ban ngày và ban đêm có độ dài khoảng 10 giây đến 5 phút và các video được xuất ra từ Camera-IP. Các video chuẩn bị cho việc đào tạo được cắt thành từng khung ảnh (frame) bằng cách sử dụng OpenCV và Python. Những khung ảnh không chứa các đối tượng, khung ảnh bị mờ sẽ được loại bỏ.



Hình 3.3 Ví dụ dữ liệu ảnh chứa hành vi leo rào và bình thường

Ví dụ như **Hình 3.3**, các dữ liệu hình ảnh thu thập được chứa các đối tượng con người có hành vi leo rào và các hành vi bình thường như đi bộ, ngồi, đứng. Chi tiết số lượng ảnh thu thập được đã qua bước tiền xử lý được mô tả chi tiết như **Bảng 3.1**

Bảng 3.1 Mô tả tập dữ liệu

Loại hình ảnh	Số lượng mẫu
Hình ảnh chứa hành vi leo rào	2497
Hình ảnh chứa hành vi khác	2385
Ảnh xoay	231
Ảnh kéo dãn	227
Tổng cộng	5340

Tập dữ liệu được chia thành hai phần: Tập ảnh gốc và tập ảnh tăng cường dữ liệu.

- **Tập ảnh gốc (Original images)** chứa 4882 ảnh trong đó chứa 2497 ảnh có hành vi leo hàng rào và 2385 ảnh chứa hành vi khác.
- **Tập ảnh tăng cường dữ liệu (Augmented images)** bao gồm **230 ảnh được lấy ra** ngẫu nhiên trong tập ảnh gốc, 231 ảnh xoay⁴ và 227 ảnh kéo dãn⁵, **Hình 3.4** ví dụ về các ảnh xoay và ảnh kéo dãn.

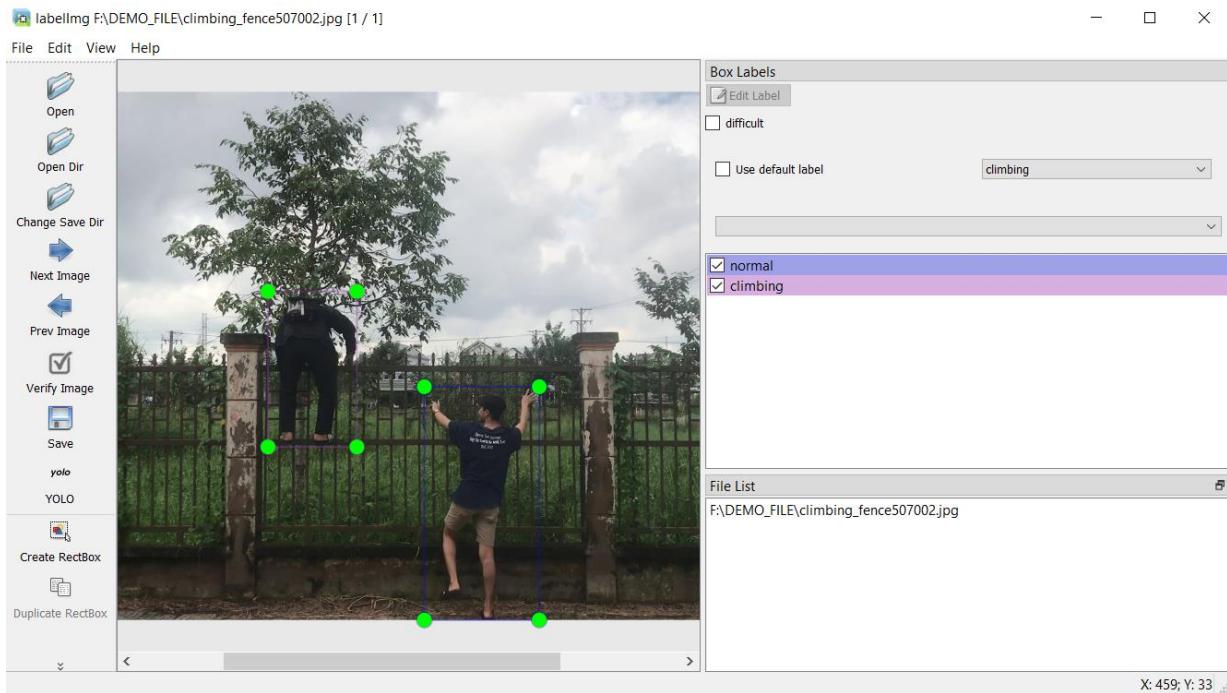


Hình 3.4 ví dụ về ảnh xoay và ảnh kéo dãn

⁴ Ảnh xoay: hình ảnh được xoay 180 từ ảnh gốc

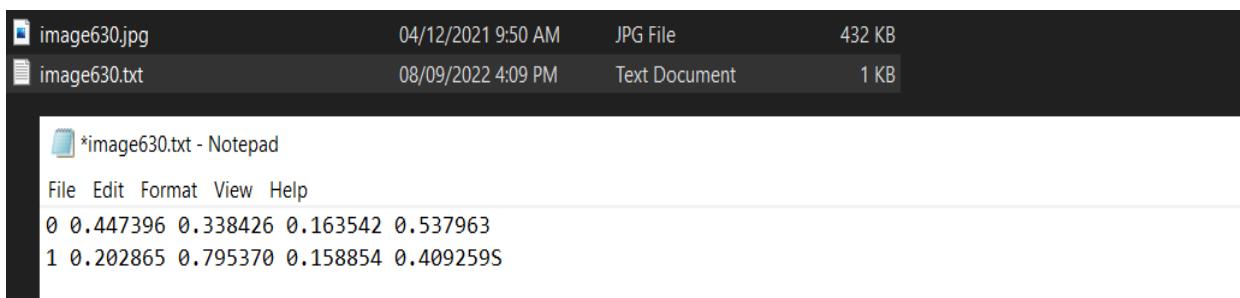
⁵ Ảnh kéo dãn: hình ảnh được kéo dãn kích thước được chọn ngẫu nhiên từ ảnh xoay để kéo dãn kích thước

Sử dụng công cụ LabelImg như **Hình 3.5** để gắn nhãn bằng thủ công cho 5340 ảnh đã thu thập được. Trong đề tài này tập trung vào phát hiện hai hành vi là leo rào và hành vi bình thường như đứng, ngồi, nằm. Vì vậy lớp nhãn được sử dụng cho các đối tượng là “climbing” (leo rào) và “normal” (bình thường).



Hình 3.5 Công cụ LabelImg hỗ trợ gắn nhãn

Trong các bài toán hiện đối tượng, mỗi bài toán sử dụng nhãn ở những định dạng có phần mở rộng khác nhau. Cụ thể mô hình YOLO sử dụng nhãn ở định dạng YOLO có phần mở rộng là ‘txt’, mô hình SSD và Faster-RCNN sử dụng nhãn ở định dạng PASCAL/VOC có phần mở rộng là ‘xml’.



Hình 3.6 Ví dụ nhãn ở định dạng YOLO

Sau khi gắn nhãn cho đối tượng ở dạng YOLO file nhãn được tạo ra như **Hình 3.6**, mỗi một dòng chứa tên nhãn và tọa độ của đối tượng. Cụ thể là các thành phần:

<object-class> <x> <y> <width> <height>

Trong đó:

<object-class> là nhãn của đối tượng được mã hóa về 0 và 1 (nhãn 0 là climbing và nhãn 1 là normal).

<x> <y> <width> <height> lần lượt là tọa độ tâm và kích thước chiều rộng chiều cao của đối tượng. Các giá trị được chuẩn hóa lại và luôn nằm trong khoảng [0;1].

```
<annotation>
  <folder>hinhanh_bc</folder>
  <filename>image630.jpg</filename>
  <path>F:\VIETBC\hinhanh_bc\image630.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>1920</width>
    <height>1080</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>climbing</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>702</xmin>
      <ymin>75</ymin>
      <xmax>1016</xmax>
      <ymax>656</ymax>
    </bndbox>
  </object>
  <object>
    <name>normal</name>
    <pose>Unspecified</pose>
    <truncated>1</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>237</xmin>
      <ymin>638</ymin>
      <xmax>542</xmax>
      <ymax>1080</ymax>
    </bndbox>
  </object>
</annotation>
```

Đối với mô hình SSD và Faster-RCNN file nhãn được tạo ra ở định dạng PASCAL/VOC có phần mở rộng là ‘xml’ như **Hình 3.7**.

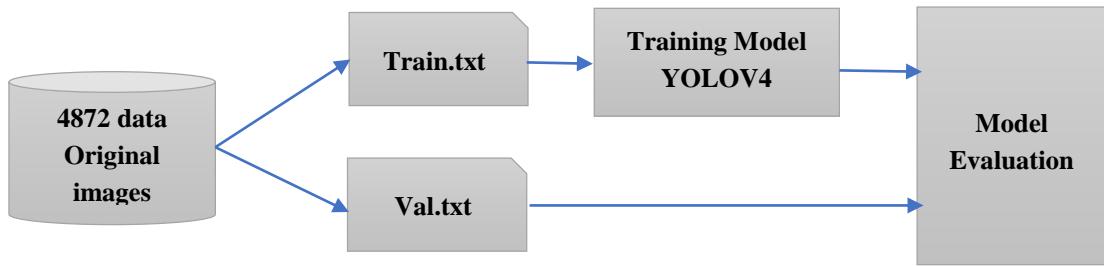
Hình 3.7 ví dụ nhãn được tạo ở
PASCAL/VOC ‘.xml’

Sau khi dữ liệu đã được gắn nhãn, bước tiếp theo là cài đặt và đào tạo mô hình.

3.1.3. Đào tạo mô hình YOLO v4

❖ Chia tập dữ liệu

Dùng tập dữ liệu gốc để đào tạo mô hình, sử dụng dụng phương pháp Hold-out chia tập dữ liệu thành hai phần, trong đó 80% dành cho đào tạo và 20% còn lại dùng để đánh giá. Mô phỏng việc đào tạo mô hình YOLO v4 như **Hình 3.8**



Hình 3.8 Mô phỏng quá trình huấn luyện YOLO v4

❖ Cấu hình thông số đào tạo mô hình YOLO v4

Trước khi đào tạo mô hình, các thông số được cấu hình cho quá trình đào tạo được mô tả chi tiết như **Bảng 3.2**.

Bảng 3.2 Mô tả các thông số cài đặt đào tạo YOLO v4

Description of hyper-parameters					
No.	Type	Values	No.	Type	Values
1	Input-image	416x416	5	Number of classes	2
2	Batch size	16	6	Decay	0.0005
3	Subdivision	32	7	Momentum	0.949
4	Learning rate	0.001	8	Num-step	6000

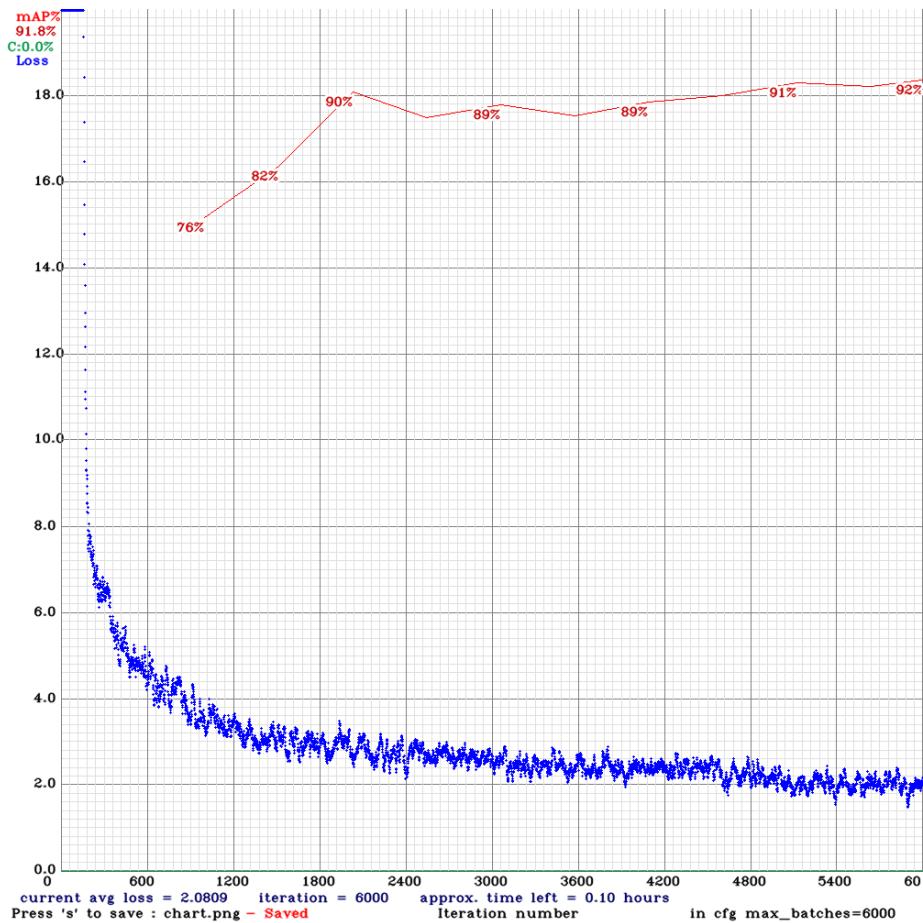
❖ Kết quả đào tạo mô hình YOLO v4

Kết quả 5 lần đào tạo được mô tả chi tiết như **Bảng 3.3**

Bảng 3.3 Kết quả 5 lần đào tạo mô hình YOLO v4

Số lần	IoU	Precision	Recall	F1-score	mAP
1	0.69	0.85	0.89	0.87	0.93
2	0.69	0.85	0.87	0.86	0.92
3	0.68	0.83	0.88	0.86	0.92
4	0.71	0.86	0.88	0.88	0.92
5	0.71	0.86	0.89	0.88	0.92

Hình 3.9 mô tả kết quả đánh giá mAP và loss ở lần đào tạo thứ 5 được sinh ra trong quá trình đào tạo. Biểu đồ cho biết sự hội tụ của mô hình qua từng giai đoạn. Theo như **Hình 3.9** hàm loss tăng giảm theo thời gian và hội tụ tốt, tương tự khi loss tăng mAP giảm và ngược lại. Đến giai đoạn đào tạo gần 6000 vòng độ hội tụ của loss giảm xuống khoảng 2.0 và mAP tăng lên 92%.



Hình 3.9 Biểu đồ mAP/Avg loss mô hình YOLO v4

Chú thích **Hình 3.9**

— Giá trị mAP

 Giá trị Loss

❖ Thực nghiệm mô hình YOLO v4

Sau mỗi lần đào tạo, mô hình được đào tạo được sinh ra. Theo kết quả ở mỗi lần đào tạo như **Bảng 3.3** kết quả ở lần thứ 5 có kết quả đánh giá mô hình tốt nhất. **Bảng 3.4** mô tả kết quả thực nghiệm trên mô hình đã đào tạo lần thứ 5.

Bảng 3.4 Kết quả thực nghiệm mô hình YOLO v4

Đầu vào	Kết quả	Độ chính xác	Nhận xét
<p>Đầu vào: hình ảnh. Kích thước: 741x486.</p> <p>Mục tiêu: phát hiện được 6 đối tượng có hành vi bình thường.</p> 		Normal 84% Normal 89% Normal 91% Normal 93% Normal 94% Normal 99%	Mô hình phát hiện tốt đối tượng không có sự nhầm lẫn giữa các hành vi. Cụ thể đạt được mục tiêu 6/6 đối tượng bình thường
<p>Đầu vào: Video. Kích thước: 852x480.</p> <p>Mục tiêu: phát hiện được 3 đối tượng có hành vi leo rào và 5 đối tượng bình thường.</p> 		Climbing 63% Climbing 78% Climbing 97% Normal 83% Normal 87% Normal 91% Normal 96% Normal 97%	Mô hình phát hiện tốt đối tượng với độ phân giải thấp không có sự nhầm lẫn. Cụ thể đạt 3/3 đối tượng leo và 5/5 đối tượng bình thường.

3.1.4. Đào tạo mô hình YOLO v7

❖ Chia tập dữ liệu

Việc chia tập dữ liệu được thực hiện như trong mô hình YOLO v4.

❖ Cấu hình thông số đào tạo mô hình YOLO v7

Các thông số được sử dụng để cấu hình cho việc đào tạo mô hình YOLO v7 được mô tả như **Bảng 3.5**.

Bảng 3.5 Mô tả thông số đào tạo YOLO v7

Description of hyper-parameters	
Input	640x640
Batch size	16
Learning rate	0.01
Number of classes	2 (Climbing and Normal)
Decay	0.0005
Momentum	0.937
Num_step_train	250 epoch

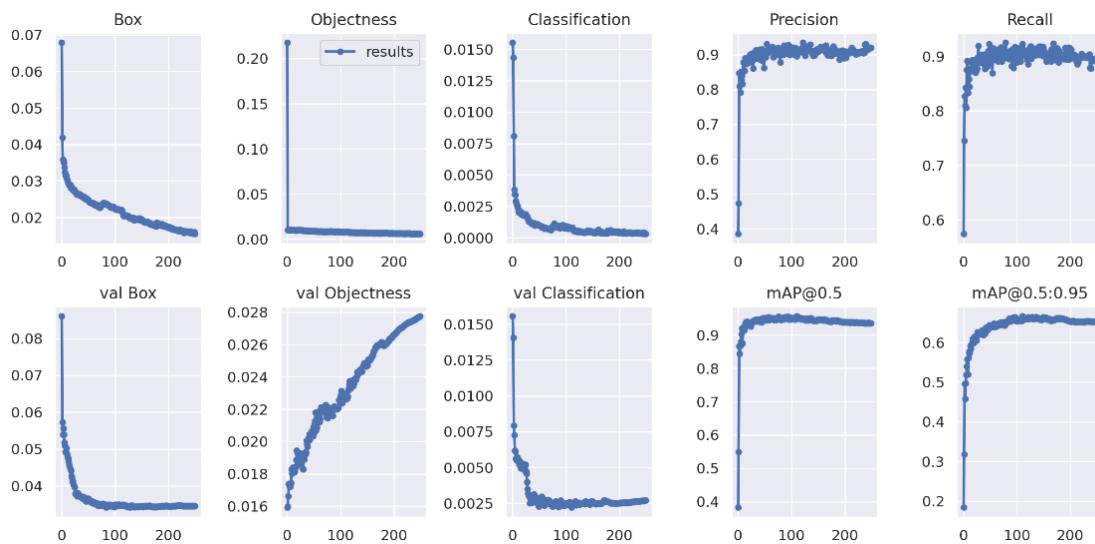
❖ Kết quả đào tạo mô hình YOLO v7

Bảng 3.6 mô tả kết quả 5 lần đào tạo mô hình YOLO v7.

Bảng 3.6 Kết quả đào tạo mô hình YOLO v7

Số lần	IoU@50	Precision	Recall	F1-score	mAP
1	0.50	0.90	0.90	0.90	0.92
2	0.50	0.92	0.90	0.91	0.97
3	0.50	0.91	0.89	0.90	0.93
4	0.50	0.91	0.91	0.91	0.94
5	0.50	0.91	0.90	0.90	0.93

Sau quá trình đào tạo mô hình YOLO v7, biểu đồ đánh giá hàm loss thu được ở lần đào tạo thứ 5 được thể hiện như **Hình 3.10**.



Hình 3.10 Biểu đồ loss trong quá trình huấn luyện và đánh giá YOLO v7

Biểu đồ loss YOLO v7 đánh giá trên tập đào tạo và tập đánh giá sau 250 epoch đào tạo mô hình. Theo **Hình 3.10** có thể thấy độ hội tụ của các hộp giới hạn Box, Objectness, Classification đều hội tụ tốt sau 250 epoch đào tạo, độ hội tụ không có tình trạng tăng giảm quá sâu điều này cũng chứng tỏ mô hình đào tạo có kết quả tốt.

❖ Thực nghiệm mô hình YOLO v7

Lựa chọn mô hình đã được đào tạo của lần đào tạo thứ 2 để thực nghiệm mô hình. Kết quả được mô tả như **Bảng 3.7**

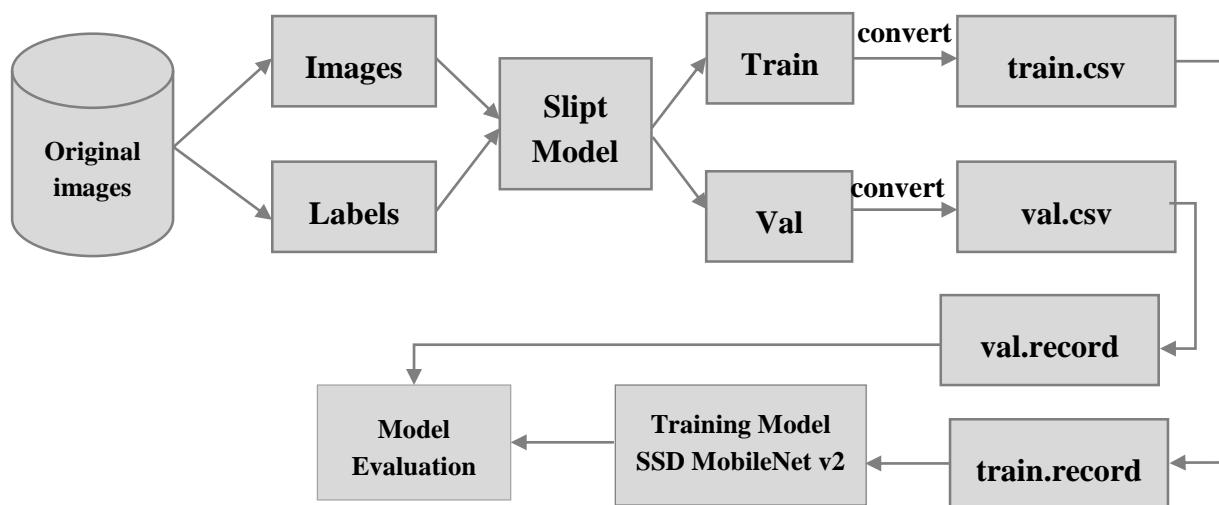
Bảng 3.7 Kết quả thực nghiệm mô hình YOLO v7

Đầu vào	Kết quả	Độ chính xác	Nhận xét
<p>Đầu vào: hình ảnh. Kích thước: 741x486.</p> <p>Mục tiêu: phát hiện được 6 đối tượng có hành vi bình thường.</p> 		Normal 72% Normal 95%	Mô hình phát hiện bỏ sót đối tượng. Cụ thể đạt 2/6 đối tượng bình thường
<p>Đầu vào: Video. Kích thước: 852x480.</p> <p>Mục tiêu: phát hiện 3 đối tượng có hành vi leo rào và 5 đối tượng bình thường.</p> 		Climbing 92% Climbing 67% Climbing 68% Normal 95% Normal 95% Normal 94%	Mô hình phát hiện bỏ sót đối tượng. Cụ thể đạt 3/3 đối tượng leo và 3/5 đối tượng bình thường.

3.1.5. Đào tạo mô hình SSD MobileNet v2

❖ Chia tập dữ liệu

Các tập ảnh và nhãn được chia thành 2 tập dữ liệu riêng biệt từ tập dữ liệu gốc, sau đó dùng kỹ thuật Split model kết nối 2 tập ảnh và tập nhãn lại tiến hành chia ngẫu nhiên tập dữ liệu thành 80% tập Train chứa ảnh/nhãn và 20% tập Val chứa ảnh/nhãn. Tiếp theo dùng kỹ thuật chuyển tập dữ liệu Train và Val thành tập Train.csv và Val.csv. Tiếp tục dữ liệu train/val.csv dùng kỹ thuật chuyển ‘csv’ thành threcord. Như vậy tập dữ liệu đào tạo và đánh giá có dạng train.record và val.record. Chi tiết các bước được mô tả như **Hình 3.11**.



Hình 3.11 Mô phỏng huấn luyện mô hình SSD MobileNet v2

❖ Cấu hình thông số đào tạo mô hình SSD MobileNet v2

Các thông số đào tạo mô hình SSD MobileNet v2 được cấu hình như **Bảng 3.8** cho 5 lần đào tạo mô hình

Bảng 3.8 Các thông số đào tạo mô hình SSD MobileNet v2

Description of hyper-parameters	
Batch size	4
Learning rate	0.8
Number of classes	2 (Climbing and Normal)
Decay	1
Momentum	0.9
Num_step	50000

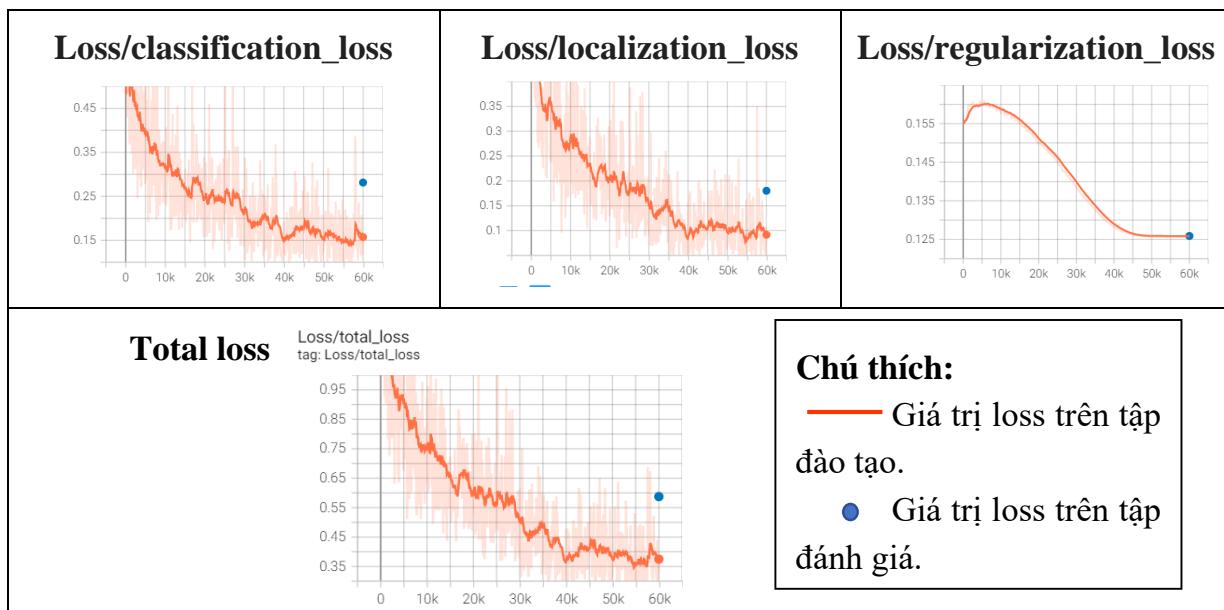
❖ Kết quả đào tạo mô hình SSD MobileNet v2

Kết quả 5 lần đào tạo mô hình SSD MobileNet v2 được mô tả trong **Bảng 3.9**

Bảng 3.9 Kết quả đào tạo mô hình SSD MobileNet v2

Số lần	IoU@50	Precision	Recall	F1-score	mAP
1	0.50	0.49	0.65	0.54	0.83
2	0.50	0.51	0.66	0.58	0.86
3	0.50	0.50	0.66	0.57	0.84
4	0.50	0.50	0.66	0.57	0.84
5	0.50	0.50	0.66	0.57	0.85

Sau mỗi một quá trình đào tạo, biểu đồ đánh giá hàm loss trong quá trình đào tạo và đánh giá mô hình SSD MobileNet được sinh ra. Tương tự biểu đồ đánh giá hàm mất mát của lần đào tạo thứ 5 được thể hiện như **Hình 3.12**.



Hình 3.12 Biểu đồ loss của SSD MobileNet v2

Classification loss (Hàm mất mát phân loại) đánh giá sự mất mát của hộp giới hạn, hộp giới hạn được tính là khi có hộp có chứa đối tượng.

Localization loss (Hàm mất mát vị trí): đánh giá sự mất mát giữa hộp giới hạn mô hình phát hiện với hộp giới hạn cơ sở

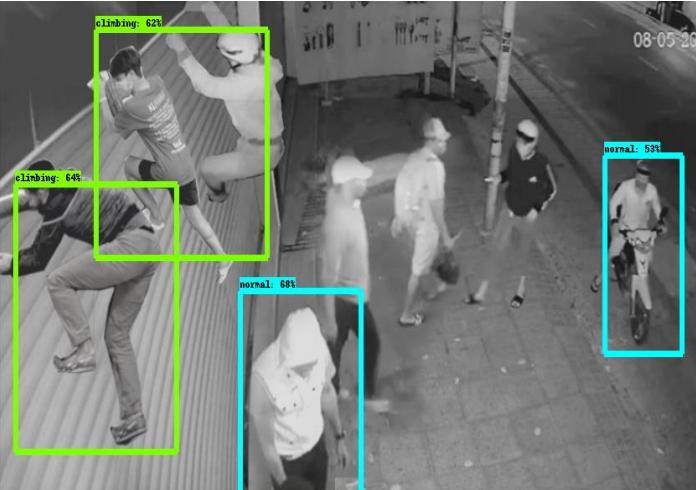
Regularization loss: đánh giá sự mất mát độ phức tạp của mô hình, độ phức tạp giảm đồng nghĩa mô hình tốt.

Theo **Hình 3.12** hàm loss của mô hình SSD MobileNet V2 sau 6000 đào tạo của lần đào tạo thứ 5, độ hội tụ của tổng tất cả hàm mất mát của mô hình giảm dần theo quá trình đào tạo tiến đến gần 0.35. Thông thường các hàm mất mát giảm xuống đến gần giá trị 0 được gọi là một mô hình tốt. Cụ thể các giá trị mất mát Classification_loss, Localization_loss, Regularization_loss, tổng các hàm mất mát được đánh giá trên tập đào tạo lần lượt có giá trị hội tụ về gần 0.15, 0.09, 0.35 và các giá trị mất mát Classification_loss, Localization_loss, Regularization_loss, tổng các hàm mất mát trên tập đánh giá lần lượt là 0.25, 0.17, 0.125 và 0.59.

❖ Thực nghiệm mô hình SSD MobileNet v2

Từ kết quả đào tạo mô hình, sử dụng mô hình ở số lần đào tạo thứ 2 đã được đào tạo để thử nghiệm mô hình với đầu vào là hình ảnh và video.

Bảng 3.10 Kết quả thực nghiệm mô hình SSD MobileNet v2

Đầu vào	Kết quả	Độ chính xác	Nhận xét
<p>Đầu vào: hình ảnh. Kích thước: 741x486.</p> <p>Mục tiêu: phát hiện được 6 đối tượng có hành vi bình thường.</p> 		Normal 65%	Mô hình phát hiện bỏ sót đối tượng. Cụ thể đạt 1/6 đối tượng bình thường
<p>Đầu vào: Video. Kích thước: 852x480.</p> <p>Mục tiêu: phát hiện 3 đối tượng có hành vi leo rào và 5 đối tượng bình thường.</p> 		Climbing 62% Climbing 64% Normal 68% Normal 53%	Mô hình phát hiện bỏ sót đối tượng. Cụ thể đạt 2/3 đối tượng leo và 2/5 đối tượng bình thường.

3.1.6. Đào tạo mô hình Faster-RCNN

❖ Chia tập dữ liệu

Mô hình Faster-RCNN sử dụng bước chia tập dữ liệu như mô hình SSD

❖ Cấu hình thông số đào tạo mô hình Faster-RCNN

Thông số được sử dụng cho quá trình đào tạo mô hình được mô tả như **Bảng 3.11**

Bảng 3.11 Các thông số đào tạo mô hình Faster-RCNN

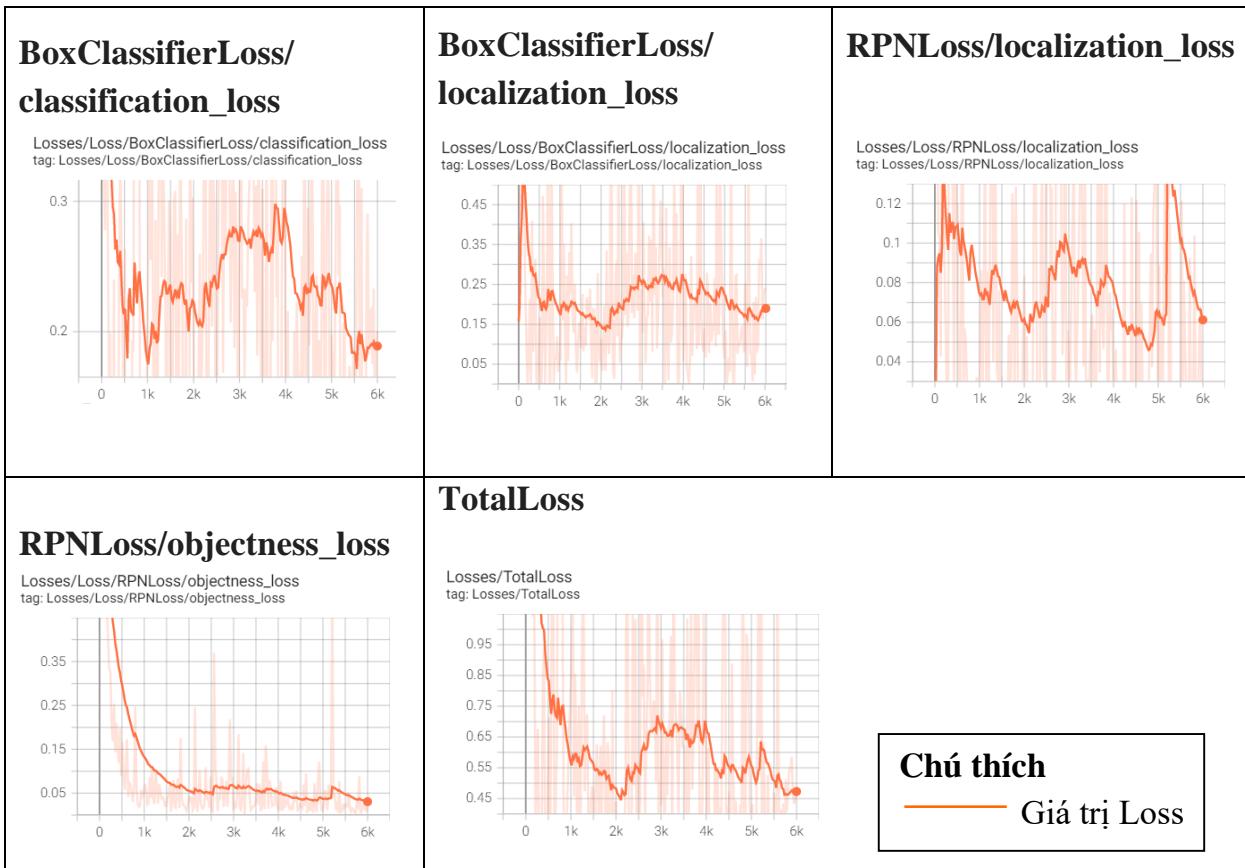
Description of hyper-parameters	
InputImage	640x640
Batch size	1
Learning rate	0.0002
Number of classes	2 (Climbing and Normal)
Decay	--
Momentum	0.9
Num_step_train	6000

❖ Kết quả đào tạo mô hình Faster-RCNN

Kết quả 5 lần đào tạo mô hình Faster-RCNN được mô tả chi tiết trong **Bảng 3.12**

Bảng 3.12 Kết quả đào tạo mô hình Faster-RCNN

Faster-RCNN							
Độ đo	Số lần	1	2	3	4	5	Avg
BoxClassifierLoss/classificationLoss	0.1	0.11	0.2	0.24	0.07	0.14	
BoxClassifierLoss/localizationLoss	0.07	0.06	0.24	0.23	0.05	0.13	
RPNLoss/localization_loss	0.12	0.01	0.02	0.06	0.02	0.05	
RPNLoss/objectness_loss	0.13	0.01	0.02	0.03	0.01	0.04	
TotalLoss	0.41	0.19	0.48	0.57	0.15	0.36	



Hình 3.13 Biểu loss trong quá trình đào tạo mô hình Faster-RCNN

Loss Faster-RCNN đánh giá sự mất mát của mô hình. Cụ thể đánh giá trên 4 hàm loss, hai hàm loss của đầu ra RPN và 2 hàm loss đánh giá phân loại đầu ra cuối cùng (Fast-RCNN). Vì kỹ thuật của mô hình Faster-RCNN phát hiện đối tượng trong hai giai đoạn.

Loss của RPN bao gồm RPN/localization_loss (sự mất mát của hộp giới hạn) và RPN/objectness_loss (sự mất mát phân loại đối tượng với nền (background)). Loss RPN tương ứng với đầu ra của RPN.

Loss phân loại cuối cùng (Fast-RCNN) gồm BoxClassifierLoss/classification_loss và BoxClassifierLoss/localization_loss

Theo **Hình 3.13** kết quả đánh giá hàm mất mát của lần đào tạo thứ 3, tổng của các loss RPN và loss loại cuối cùng (Fast-RCNN) sau quá trình đào tạo 6000 vòng độ hội tụ loss giảm gần 0.48 chứng tỏ mô hình Faster-RCNN tốt.

❖ Thực nghiệm mô hình Faster-RCNN

Sử dụng mô hình đã được đào tạo ở số lần thứ 5 để thực nghiệm mô hình, vì độ hội tụ của hàm loss tốt hơn các số lần còn lại. Đầu vào cho mô hình là hình ảnh và video có kích thước độ phân giải thấp, kết quả thực nghiệm được mô tả chi tiết trong **Bảng 3.13**

Bảng 3.13 Kết quả thực nghiệm mô hình Faster-RCNN

Đầu vào	Kết quả	Độ chính xác	Nhận xét
<p>Đầu vào: hình ảnh. Kích thước: 741x486.</p> <p>Mục tiêu: phát hiện được 6 đối tượng có hành vi bình thường.</p> 		Normal 56% Normal 65% Normal 58% Normal 89% Climbing 52%	Mô hình phát hiện bỏ sót và nhầm lẫn đối tượng. Cụ thể đạt 5/6 đối tượng
<p>Đầu vào: Video. Kích thước: 852x480.</p> <p>Mục tiêu: phát hiện 3 đối tượng có hành vi leo rào và 5 đối tượng bình thường.</p> 		Climbing 62% Climbing 74% Climbing 74% Normal 55% Normal 90%	Mô hình phát hiện bỏ sót và nhầm lẫn đối tượng . Cụ thể phát hiện 2/3 đối tượng leo rào, 3/5 đối tượng bình thường. Trong đó có 1 đối tượng phát hiện nhầm lẫn.

3.1.7. Đánh giá kết quả đào tạo mô hình phát hiện hành vi leo rào

Việc đánh giá các mô hình đã được đào tạo giúp cho chúng ta so sánh các mô hình lại với nhau từ đó lựa chọn mô hình tốt nhất để tích hợp vào hệ thống. **Bảng 3.14 & Bảng 3.15** mô tả kết quả đánh giá trên giá trị trung bình của các độ đo.

Bảng 3.14 Kết quả đánh giá mô hình phát hiện đối tượng một giai đoạn

Mô hình phát hiện đối tượng một giai đoạn					
(Models) Mô hình	Avg IoU	Avg Precision	Avg Recall	Avg F1-score	Avg mAP
YOLO v4	0.70	0.85	0.88	0.87	0.92
YOLO v7	0.50	0.91	0.90	0.90	0.94
SSD	0.50	0.50	0.66	0.57	0.84

Bảng 3.15 Kết quả đánh giá mô hình phát hiện hai giai đoạn

Mô hình phát hiện đối tượng hai giai đoạn	
FASTER-RCNN	
	Avg
BoxClassifierLoss/classificationLoss	0.14
BoxClassifierLoss/localizationLoss	0.13
RPNLoss/localization_loss	0.05
RPNLoss/objectness_loss	0.04
TotalLoss	0.36

3.1.8. Kết luận

Từ kết quả bảng so sánh đánh giá mô hình một giai đoạn và mô hình hai giai đoạn. Kết quả của mô hình YOLO v4 có kết quả tốt nhất, trong đó trung bình độ chính xác (Avg IoU) và Avg F1-score của mô hình YOLO v4 lần lượt là 70% và 87% tốt hơn so với mô hình SSD độ chính xác đạt trong ngưỡng 50% và đạt 57% về avg F1-score.

Mặc dù YOLOv7 có độ đo trung bình F1-score đạt trên 90% về độ chính xác nằm trong ngưỡng 50%, nhưng về độ chính xác nằm trong ngưỡng từ 50% đến 95% (IoU@0.50:0.95) các độ đo Precision, Re-call và F1-score đều giảm.

Mô hình phát hiện hai giai đoạn Faster-RCNN mặc dù không lấy được các độ đo để so sánh nhưng kết quả thử nghiệm sẽ là thước đo để đánh giá.

Kết quả đánh giá thử nghiệm giữa các mô hình được mô tả sau quá trình đào tạo mô hình phát hiện hành vi leo rào. Mô hình YOLO v4 phát hiện được tất cả các đối tượng đạt được mục tiêu yêu cầu của bài toán ngay cả khi cho đầu vào là hình ảnh/ video có độ phân giải thấp (**Bảng 3.4**). Cụ thể mô hình SSD, Faster-RCNN chỉ phát hiện được các đối tượng có kích thước lớn nên bỏ sót nhiều đối tượng không đạt được mục tiêu của bài toán, **Bảng 3.13 & Bảng 3.10** mô tả kết quả thực nghiệm của hai mô hình SSD và Faster-RCNN. YOLO v7 mặc dù phát hiện tốt đối tượng nhưng vẫn còn bỏ sót đối tượng, kết quả thực nghiệm mô hình YOLO v7 được mô tả trên **Bảng 3.7**.

3.1.8.1. Nhận xét

Từ kết luận trên, mô hình YOLO v4 có kết quả đánh giá lắn thực nghiệm tốt hơn các mô hình YOLO v7, SSD và Faster-RCNN. Để kiểm tra được hiệu suất của mô hình YOLO v4 có tốt hơn khi đào tạo YOLO v4 áp dụng kỹ thuật tăng cường dữ liệu và kỹ thuật cài tiền tham số, mô hình YOLO v4 sẽ cho kết quả như thế nào. Trong phần tiếp theo sẽ áp dụng 2 kỹ thuật nêu trên để đào tạo và đánh giá hiệu suất trên mô hình YOLO v4.

3.1.8.2. Kỹ thuật tăng cường dữ liệu

❖ Chia tập dữ liệu

Sử dụng phương pháp Hold-out để chia tập dữ liệu ảnh gốc thành 80% tập train và 20% tập val (điều kiện tập dữ liệu này đã loại bỏ các ảnh được mô tả trong tập ảnh tăng cường dữ liệu), sau đó dùng tất cả ảnh có trong tập dữ liệu tăng cường đưa vào tập train, các ảnh có trong tập dữ liệu tăng cường không được chứa trong tập val.

❖ Cấu hình thông số đào tạo

Các thông số cài đặt đào tạo mô hình YOLO v4 áp dụng kỹ thuật tăng cường dữ liệu tương tự như **Bảng 3.2**.

❖ Kết quả đào tạo mô hình YOLO v4 tăng cường dữ liệu

Kết quả 4 lần đào tạo mô hình YOLO v4 dùng kỹ thuật tăng cường dữ liệu được mô tả chi tiết như bảng **Bảng 3.16**.

Bảng 3.16 Kết quả đào tạo mô hình YOLO v4 kỹ thuật tăng cường dữ liệu

Độ đo Số lần	IoU	Precision	Recall	F1-score	mAP
1	0.71	0.87	0.87	0.87	0.91
2	0.71	0.86	0.88	0.87	0.91

Số lần	Độ đo	IoU	Precision	Recall	F1-score	mAP
3		0.70	0.86	0.88	0.87	0.90
4		0.71	0.86	0.88	0.87	0.92
Avg		0.71	0.86	0.88	0.87	0.91

3.1.8.3. Kỹ thuật cải tiến tham số

❖ Chia tập dữ liệu

Dùng tập dữ liệu đã được chia tập train và tập val ở bước đào tạo mô hình YOLO v4 áp dụng kỹ thuật tăng cường dữ liệu để đào tạo mô hình YOLO v4 cải tiến tham số.

❖ Cấu hình thông số đào tạo mô hình YOLO v4 cải tiến tham số

Các thông số cải tiến được cấu hình như **Bảng 3.17**

Bảng 3.17 Mô tả thông số cải tiến YOLO v4

Description of hyper-parameters					
No.	Type	Values	No.	Type	Values
1	Input	608x608 [416x416]	6	Decay	0.0005
2	Batch size	64	7	Momentum	0.949
3	Subdivision	64	8	Ignore_thresh	0.9
4	Learning rate	0.001	9	IoU_normalizer	0.5
5	Number of classes	2	10	IoU_loss	Giou

Cải tiến tham số được áp dụng hai đầu vào kích thước 608x608 và 416x416. Sử dụng kích thước lô 64 để đưa vào mô hình đào tạo được nhiều hơn. Trong phần Head của mô hình YOLO v4 sử dụng random=1, max=200, Ignore_thresh=0.9, IoU_normalizer=0.5 và IoU_loss = Giou ở trong phần Head (tức là tầng dự đoán nhãn và tạo hộp giới hạn của YOLO v3) điều này giúp cho mô hình tăng độ chính xác ở những độ phân giải khác nhau khi dùng random =1 và tăng khả năng nhận dạng đối tượng có số lượng lớn trong bức ảnh khi dùng max=200. Tăng ngưỡng độ chính xác Ignore_thresh mặc định từ 0.5 lên 0.9.

Thay hàm IoU_loss = IoU hoặc ciou bằng Giou [24], bởi vì IoU là độ đo tỉ lệ phần phần giao và phần hội của hai hộp giới hạn. Khi mô hình không phát hiện được phần hội giữa hai hộp giới hạn thì IoU = 0. Chính vì vậy Giou được thay thế.

❖ Kết quả đào tạo mô hình YOLO v4 cải tiến

Kết quả đào tạo mô hình YOLO v4 cải tiến tham số được mô tả như **Bảng 3.18**

Bảng 3.18 kết quả đào tạo mô hình YOLO v4 cải tiến tham số

Kết quả đào tạo với tham số cải tiến với Input 608x608					
Độ đo Lần	IoU	Precision	Recall	F1-score	mAP
1	0.73	0.87	0.86	0.87	0.90
2	0.73	0.86	0.85	0.86	0.91
Avg	0.73	0.87	0.86	0.87	0.91
Kết quả đào tạo với tham số cải tiến với Input 416x416					
1	0.73	0.87	0.88	0.87	0.91

Từ kết quả đào tạo mô hình YOLO v4 áp dụng kỹ thuật tăng cường dữ liệu và kỹ thuật cải tiến tham số được mô tả trên **Bảng 3.16 & Bảng 3.18** có kết luận như sau:

Kết quả mô hình YOLO v4 dùng kỹ thuật tăng cường dữ liệu cho kết quả tốt nhất so với mô hình YOLO v4 cải tiến với trung bình độ chính xác là 0.71 mặc dù độ chính xác (IoU) của mô hình YOLO v4 cải tiến cao hơn 0.02. Nhưng xét về độ thực nghiệm mô hình YOLO v4 có cải tiến thì mô hình phát hiện bỏ sót nhiều đối tượng, có độ chính xác thấp và tốc độ xử lý chậm, so với mô hình YOLO v4 có tăng cường dữ liệu mô hình phát hiện đúng đối tượng và không bỏ sót, tốc độ xử lý nhanh. Vì vậy, mô hình YOLO v4 dùng kỹ thuật tăng cường dữ liệu được áp dụng (tích hợp) vào **hệ thống phát hiện hành vi leo rào**.

Hình 3.14 mô tả kết quả thực nghiệm giữa hai mô hình áp dụng kỹ thuật tăng cường dữ liệu và kỹ thuật cải tiến tham số.

➤ Kết quả so sánh thực nghiệm mô hình YOLO v4 áp dụng kỹ thuật tăng cường dữ liệu và cải tiến tham số.



Hình 3.14 Kết quả thực nghiệm trên mô hình tăng cường dữ liệu và cải tiến mô hình

Nhận xét: Kết quả thực nghiệm (**Hình 3.14**) cho thấy mô hình YOLO v4 áp dụng kỹ thuật tăng cường dữ liệu phát hiện tốt hơn mô hình cải tiến. Các đối tượng nhỏ đều được phát hiện, những đối tượng có hành vi leo rào được phát hiện với độ chính xác cao hơn là mô hình YOLO v4 có cải tiến tham số.

3.2. Thiết kế và cài đặt hệ thống

3.2.1. Môi trường thiết kế và cài đặt hệ thống

Môi trường thiết kế *hệ thống phát hiện hành vi leo rào* sử dụng máy tính CPU có cấu hình như **Bảng 3.19**.

Bảng 3.19 Cấu hình máy tính

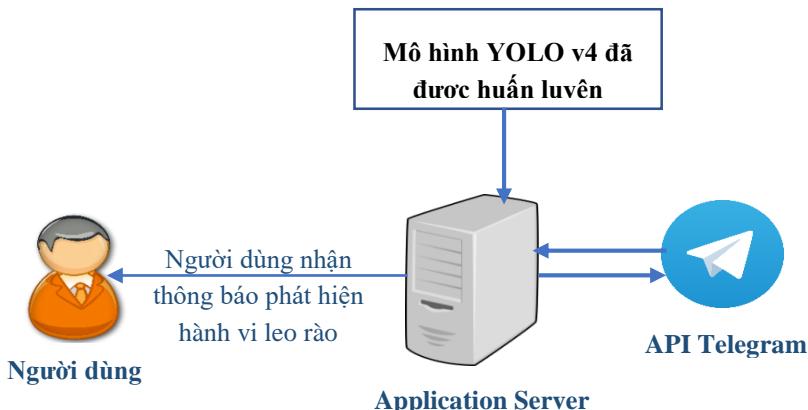
CPU	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
RAM	4.00 GB
SSD	120 GB

Dùng công cụ hỗ trợ lập trình Visual Studio Code , và Anaconda hỗ trợ cài đặt các gói Python, Numpy, OpenCV, các thư viện hỗ trợ xây dựng giao diện bằng Tkinter và CustomTkinter.

Sử dụng gói thư viện python-telegram-bot của ứng dụng Telegram để tích hợp API vào hệ thống. Telegram là ứng dụng mã xã hội phổ biến ngày nay có tác dụng nghe gọi, gửi và trả lời tin nhắn miễn phí

3.2.2. Kiến trúc hệ thống

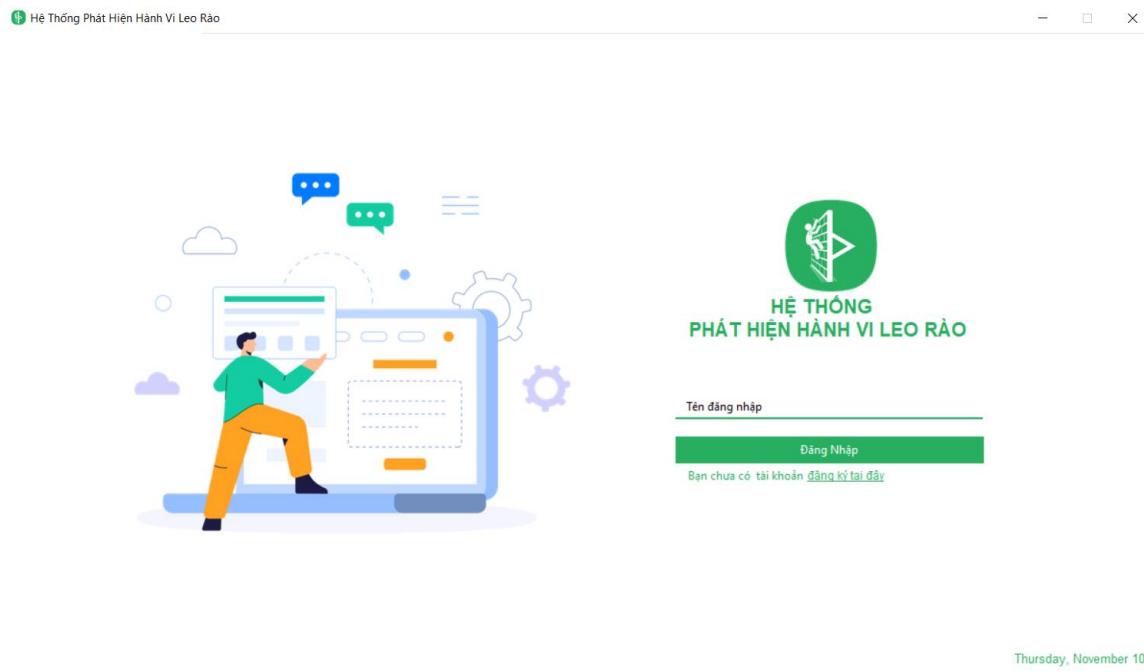
Kiến trúc tổng quan của hệ thống được minh họa như **Hình 3.15**.



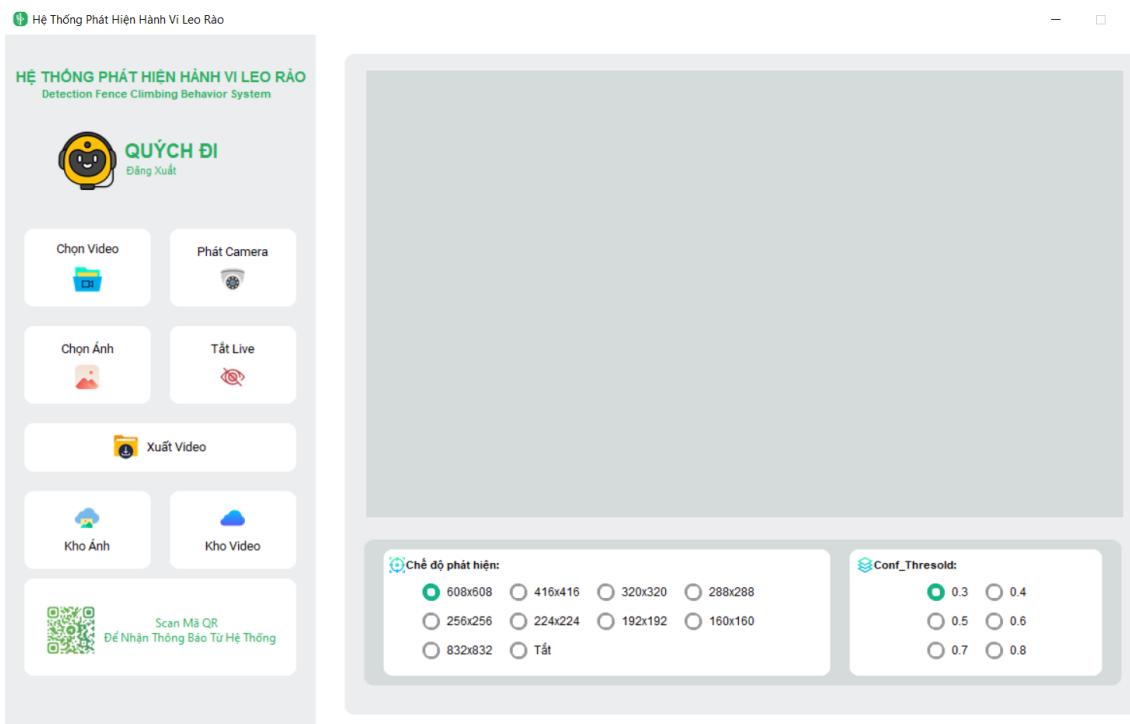
Hình 3.15 Kiến trúc tổng quan hệ thống phát hiện hành vi

Mô hình YOLO v4 được tích hợp hệ thống. Hệ thống có nhiệm vụ phát hiện hành vi các đối tượng thông qua các đoạn video phát trực tiếp trên camera giám sát, video hoặc hình ảnh. Khi phát hiện được đối tượng leo rào hệ thống tự động gửi thông tin cảnh báo đến người dùng thông qua API Telegram được tích hợp vào hệ thống (**Hình 3.18**). Người dùng cài đặt ứng dụng telegram trên máy điện thoại Android hoặc IOS hoặc Telegram phiên bản PC để gia nhập vào nhóm chat bot của hệ thống nhận cảnh báo phát hiện hành vi leo

rào, các bước cách tạo ứng dụng telegram được mô tả trong phần phụ lục **Hướng dẫn cài đặt telegram và tạo chat bot**. Để sử dụng được hệ thống, người dùng cần có tài khoản để đăng nhập (**Hình 3.16**). Khi đăng nhập thành công hệ thống hiển thị giao diện làm việc chính (**Hình 3.17**).



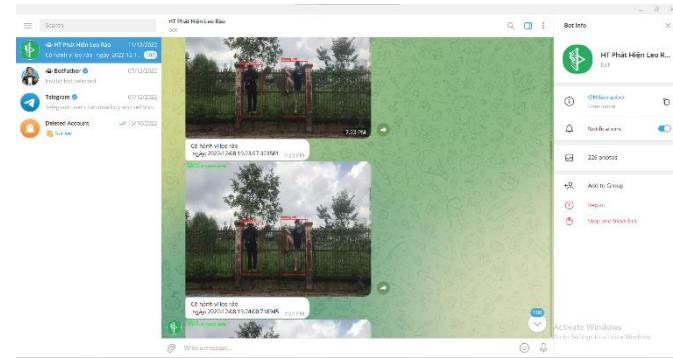
Hình 3.16 Giao diện đăng nhập



Hình 3.17 Giao diện làm việc chính



Trên Mobile APP



Trên CPU

Hình 3.18 Người dùng nhận cảnh báo từ hệ thống trên chat-bot

3.3. Tổng kết chương

Trong chương này mô tả, giới thiệu các môi trường cài đặt đào tạo mô hình cũng như môi trường cài đặt hệ thống. Các kết quả đánh giá trên các độ đo, kết quả thực nghiệm của mô hình YOLO v4, YOLO v7, SSD và Faster-RCNN cũng đã mô tả chi tiết trong chương này.

CHƯƠNG 4. KIỂM THỬ VÀ ĐÁNH GIÁ

4.1. Kiểm thử

Trong phần kiểm thử bao gồm hai kịch bản chính:

Kịch bản thứ nhất: Kiểm thử trên mô hình YOLO v4

Kịch bản thứ hai: Kiểm thử trên hệ thống

4.1.1. Kiểm thử trên mô hình YOLO v4

Dùng 154 tập dữ liệu ảnh được thu thập bằng cách cắt từng khung ảnh từ video được quay trong môi trường ít nhiễu có kích thước 1920x1080, khoảng cách quay từ camera đến hàng rào sắt là 5m. Bước thực hiện được mô tả cụ thể như sau:

❖ **Bước 1:** Gắn nhãn cho 154 tập dữ liệu ảnh thu thập được (Bước này gọi là ground-truth). Tổng tất cả các đối tượng có trong mỗi hình ảnh có 128 đối tượng có hành vi leo rào và 65 đối tượng có hành vi bình thường.

❖ **Bước 2:** Đưa dữ liệu vào cho mô hình tự kiểm thử và đánh giá.

Kết quả đánh giá được mô tả ở phần **4.2.1**

4.1.2. Kiểm thử trên hệ thống

Trong phần này tập trung kiểm thử vào các chức năng trên **hệ thống phát hiện hành vi leo rào**. Chi tiết kịch bản các chức năng thể hiện dưới **Bảng 4.1**.

Bảng 4.1 Mô tả kịch bản hệ thống

Tên chức năng	Kịch bản chính	Kịch bản phụ
1. Đăng nhập.	Bước 1: Người dùng đăng nhập vào hệ thống với số SĐT đã đăng ký Hình 3.16 . Nếu đúng chuyển sang bước 2 . Ngược lại chuyển sang kịch bản phụ Bước 2: Hệ thống chuyển sang giao diện trang làm việc chính của hệ thống Hình 3.17	Bước 1: Hệ thống thông báo ‘Lỗi đăng nhập’ yêu cầu đăng nhập lại. Bước 2: Quay về bước 1 trong kịch bản chính.
2. Đăng xuất.	Bước 1: Người dùng chọn vào ‘Đăng xuất’ trên màn hình làm việc chính của hệ thống Hình 3.17	Hệ thống trở về màn hình làm việc chính Hình 3.17

Tên chức năng	Kịch bản chính	Kịch bản phụ
	<p>Bước 2: Hệ thống hiện thông báo yêu cầu xác nhận đăng xuất. Nếu chọn ‘OK’ hệ thống chuyển sang bước 3. Ngược lại hệ thống chuyển sang luồng kịch bản phụ.</p> <p>Bước 3: Hệ thống trả người dùng về giao diện đăng nhập Hình 3.17</p>	
3. Chọn Phát camera	<p>Bước 1: Người dùng chọn vào chức năng ‘Phát camera’ (Hình 4.2).</p> <p>Bước 2: Hệ thống bật camera và hiển thị kết quả nhận dạng phát hiện đối tượng theo thời gian thực lên màn hình chính của hệ thống (Hình 4.17 & Hình 4.18).</p>	
4. Chọn file video để phát.	<p>Bước 1: Người dùng chọn vào chức năng ‘Phát file video’ (Hình 4.2).</p> <p>Bước 2: Hệ thống hiện cửa sổ yêu cầu người dùng chọn file video (Hình 4.5). Nếu file video được chọn hệ thống chuyển đến bước 3. Ngược lại chuyển sang kịch bản phụ.</p> <p>Bước 3: Hệ thống trả kết quả nhận dạng phát hiện đối tượng theo thời gian thực lên màn hình chính (Hình 4.11 & Hình 4.12).</p>	Hệ thống trả về màn hình làm việc chính (Hình 3.17)
5. Chọn file ảnh để phát.	<p>Bước 1: Người dùng chọn ‘Phát file ảnh’ (Hình 4.2).</p> <p>Bước 2: Hệ thống hiển thị cửa sổ yêu cầu người dùng chọn file ảnh (Hình 4.4). Nếu file ảnh được chọn hệ thống chuyển đến bước 3. Ngược lại chuyển sang kịch bản phụ.</p>	Hệ thống trả về màn hình làm việc chính (Hình 3.17).

Tên chức năng	Kịch bản chính	Kịch bản phụ
	Bước 3: Hệ thống xử lý và trả kết quả lên màn hình chính (Hình 4.9 & Hình 4.10).	
6. Xuất video.	<p>Bước 1: Người dùng chọn chức năng ‘Xuất video’ (Hình 4.2).</p> <p>Bước 2: Hệ thống kiểm tra dữ liệu có chứa các khung hình đã lưu lại trong thư mục lưu trữ video hay không. Nếu có chuyển sang bước 3. Ngược lại chuyển sang kịch bản phụ.</p> <p>Bước 3: Hệ thống tiến hành xuất khung hình thành video và hiển thị thông báo ‘Xuất video thành công’ (Hình 4.6).</p>	<p>Bước 1: Hệ thống hiện thông báo ‘Xuất video không thành công’ do thư mục chứa các khung hình rỗng.</p> <p>Bước 2: Hệ thống chuyển sang màn hình làm việc chính (Hình 3.17)</p>
7. Xem lại video đã xuất.	<p>Bước 1: Người dùng chọn vào ‘Kho video’ (Hình 4.2).</p> <p>Bước 2: Hệ thống hiển thị cửa sổ chứa các video yêu cầu chọn video để xem (Hình 4.14). Nếu video được chọn chuyển sang bước 3, ngược lại chuyển sang kịch bản phụ.</p> <p>Bước 3: Hệ thống hiển thị kết quả phát video cho người dùng xem (Hình 4.16).</p>	Hệ thống chuyển sang màn hình làm việc chính (Hình 3.17).
8. Xem lại ảnh trong kho.	<p>Bước 1: Người dùng chọn vào ‘Kho ảnh’ (Hình 4.2).</p> <p>Bước 2: Hệ thống hiển thị cửa sổ yêu cầu người dùng chọn hình ảnh để xem (Hình 4.7). Nếu ảnh được chọn chuyển sang bước 3, ngược lại chuyển sang kịch bản phụ.</p>	Hệ thống chuyển sang màn hình làm việc chính. (Hình 3.17).

Tên chức năng	Kịch bản chính	Kịch bản phụ
	Bước 3: Hệ thống hiển thị hình ảnh cho người dùng xem (Hình 4.15).	
9. Chọn chế độ phân giải.	<p>Bước 1: Người dùng chọn vào 1 chế độ kích thước phân giải để áp vào video/ camera/ hình ảnh đầu vào cho hệ thống xử lý (Hình 4.3).</p> <p>Bước 2: Hệ thống kiểm tra có tiến trình nào trả kết quả của các chức năng 3 hoặc 4 hoặc 5 đang thực thi hay không. Nếu có hệ thống chuyển sang kịch bản phụ, ngược lại chuyển sang bước 3.</p> <p>Bước 3: Chế độ áp dụng kích thước độ phân giải được áp dụng thành công.</p>	<p>Bước 1: Hệ thống yêu cầu người dùng chọn chức năng 11 (Hình 4.25).</p> <p>Bước 2: Nếu được chọn chức năng 11 hệ thống chuyển sang bước 1 trong kịch bản chính.</p>
10. Chọn ngưỡng độ chính xác.	<p>Bước 1: Người dùng chọn vào 1 chế độ ‘Ngưỡng độ chính xác’ (Hình 4.3).</p> <p>Bước 2: Hệ thống kiểm tra có tiến trình nào trả kết quả của các chức năng 3 hoặc 4 hoặc 5 đang thực thi hay không. Nếu có hệ thống chuyển sang kịch bản phụ, ngược lại chuyển sang bước 3.</p> <p>Bước 3: Chế độ áp dụng ngưỡng độ chính xác thành công.</p>	<p>Bước 1: Hệ thống yêu cầu người dùng chọn chức năng 11 (Hình 4.2).</p> <p>Bước 2: Nếu được chọn chức năng 11 hệ thống chuyển sang bước 1 trong kịch bản chính.</p>
11. Tắt Live.	<p>Bước 1: Người dùng chọn ‘Tắt Live’ (Hình 4.2).</p> <p>Bước 2: Hệ thống kiểm tra có tiến trình nào của các chức năng 3 hoặc 4 hoặc 5 đang thực thi hay không. Nếu có chuyển sang bước 3.</p> <p>Bước 3: Hệ thống tắt tất cả quá trình thực thi của hệ thống (Hình 3.17).</p>	

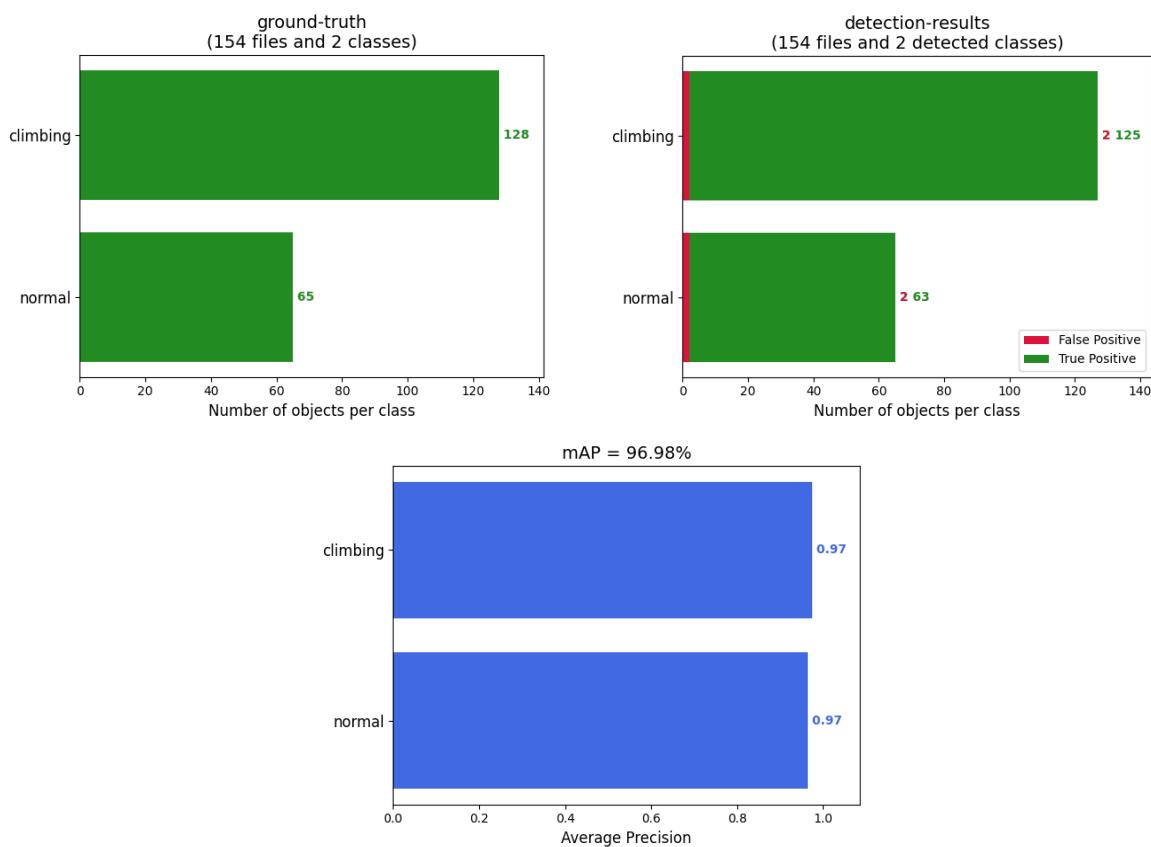
4.2. Kết quả kiểm thử

4.2.1. Kết quả kiểm thử trên mô hình YOLO v4

Hình 4.1 mô tả kết quả đánh giá mô hình thể hiện bằng biểu đồ. Trong 128 đối tượng có hành vi leo rào (ground-truth) mô hình phát hiện đúng có hành vi leo rào là 125 (TP) và mô hình phát hiện hành vi leo rào là hành vi khác (sai) là 2 đối tượng (FP). Xét hành vi bình thường có 65 đối tượng (ground-true), mô hình phát hiện đúng 63 đối tượng có hành vi bình thường (TN) và mô hình phát hiện sai đối tượng có hành vi bình thường là hành vi leo là 2 đối tượng (FN). **Bảng 4.2** mô tả lại kết quả mô hình phát hiện bằng bảng ma trận nhầm lẫn.

Bảng 4.2 Đánh giá kết quả mô hình bằng bảng ma trận nhầm lẫn

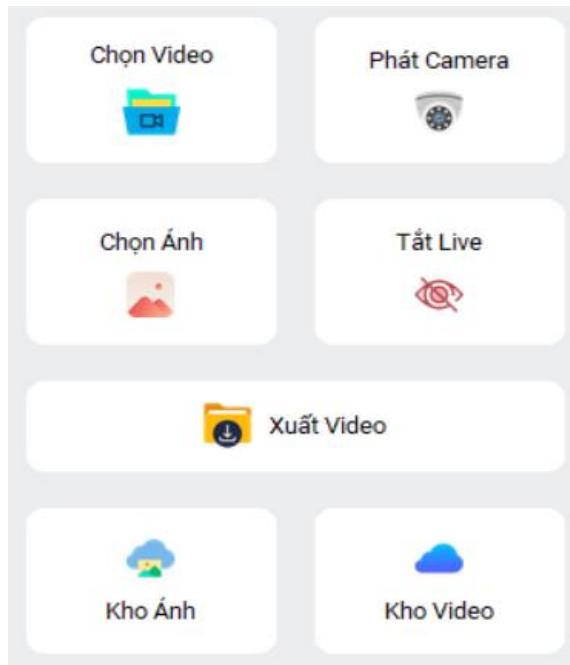
	Climbing (Positive)	Normal (Negative)
Climbing (Positive)	125	2
Normal (Negative)	2	63



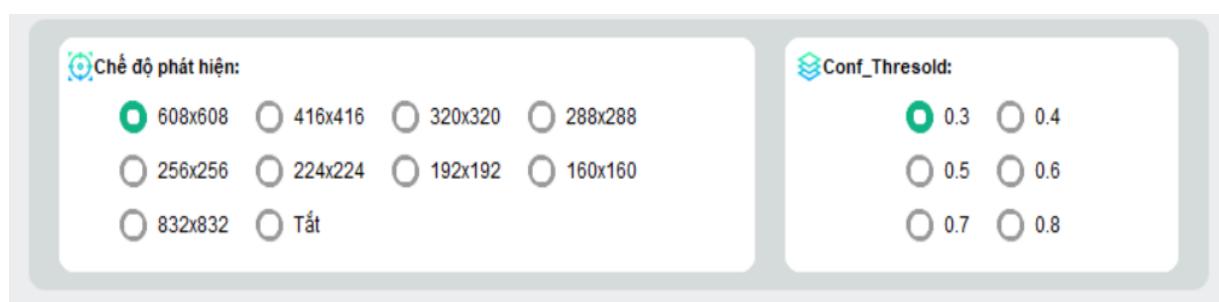
Hình 4.1 Kết quả đánh giá mô hình trên 154 tập dữ liệu

4.2.2. Kết quả kiểm thử trên hệ thống

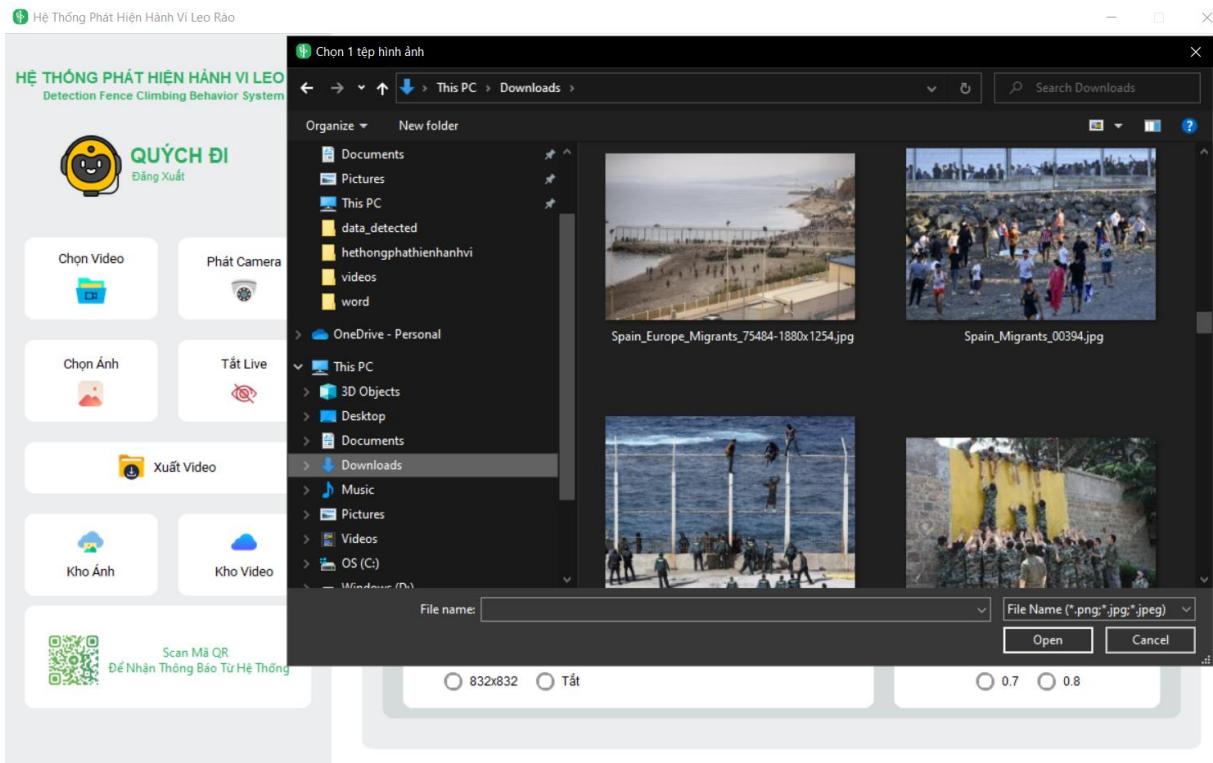
Phần này trình bày kết quả kiểm thử trên hệ thống với kết quả là giao diện hệ thống.



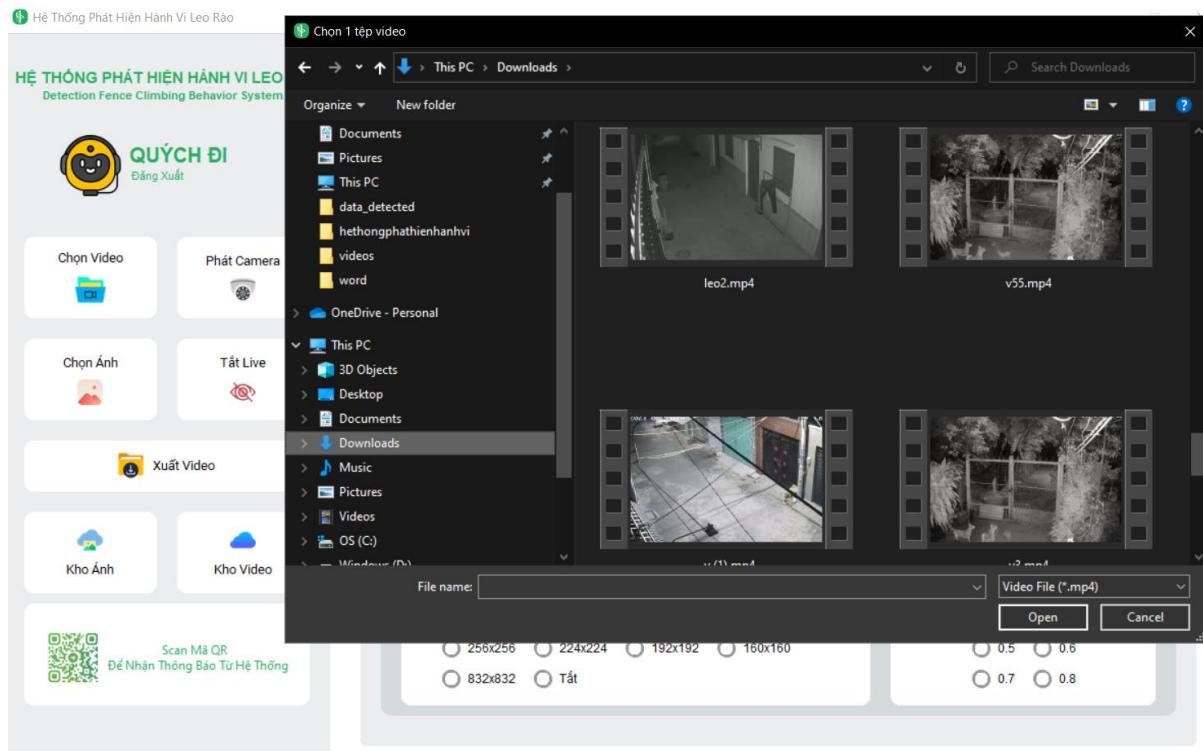
Hình 4.2 Các chức năng chính dành cho người dùng



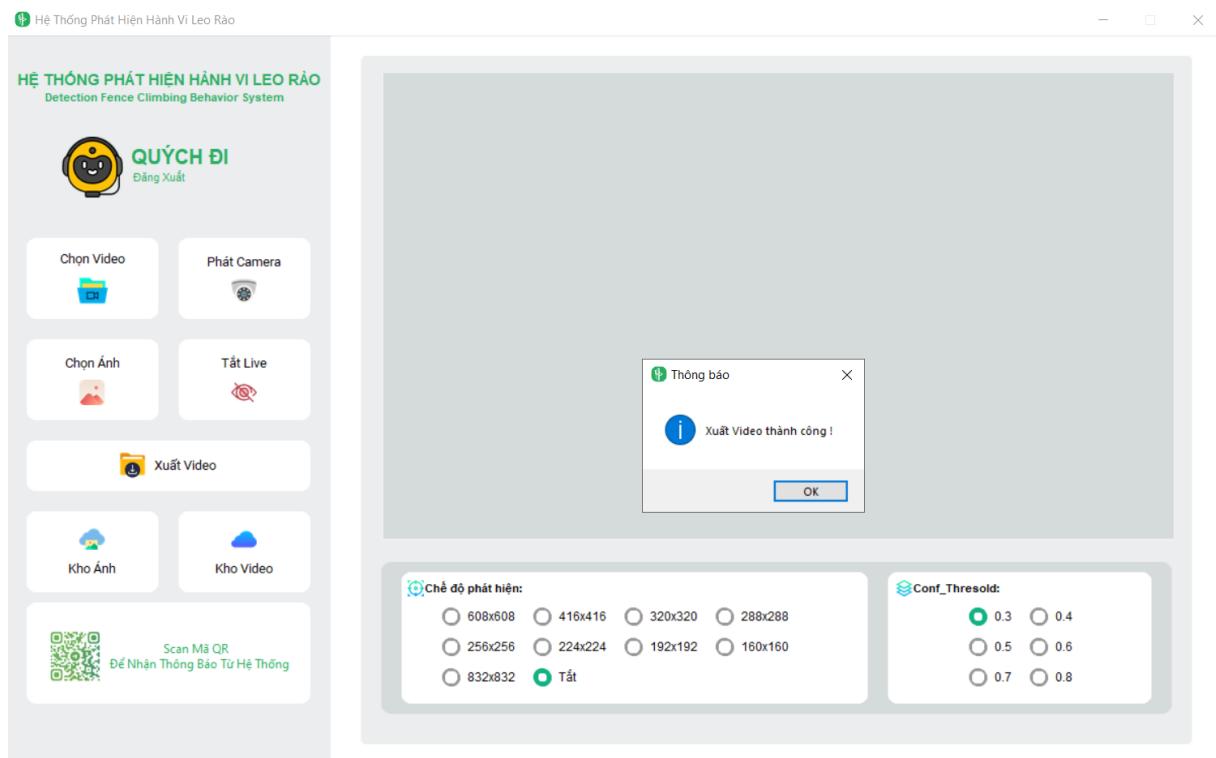
Hình 4.3 Chức năng chè độ phân giải và ngưỡng chính xác



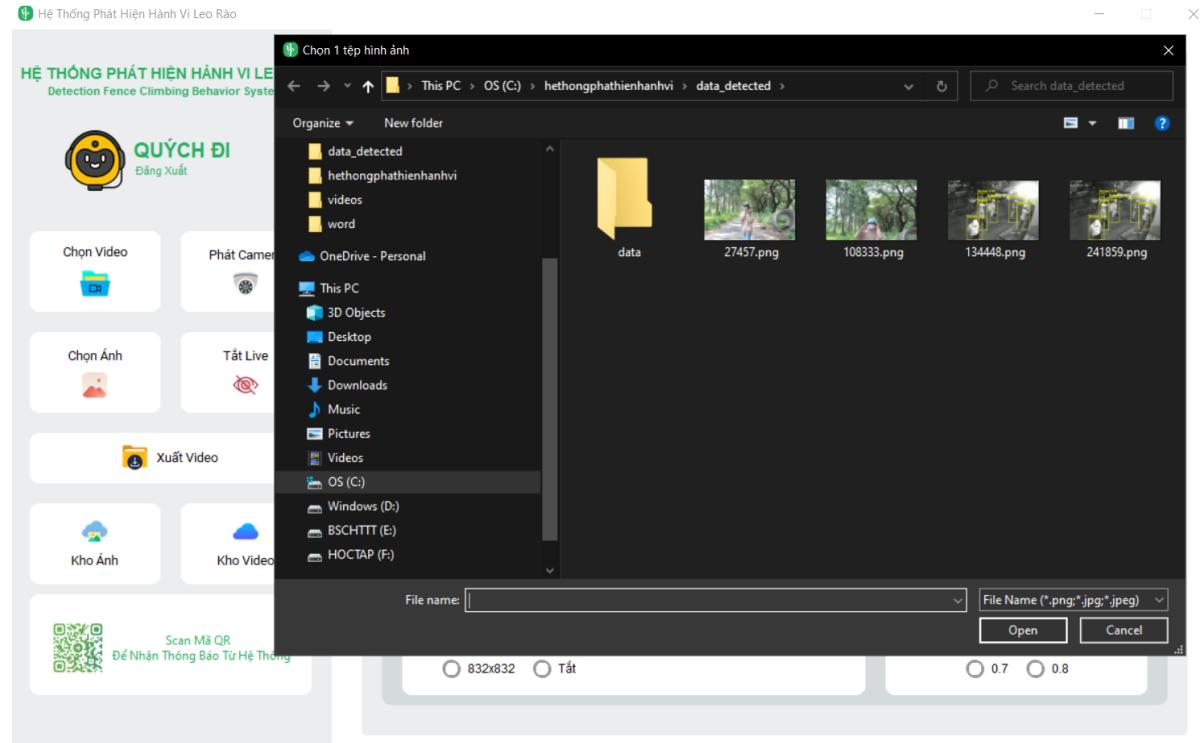
Hình 4.4 Chức năng Phát Hình Ánh yêu cầu chọn file ảnh



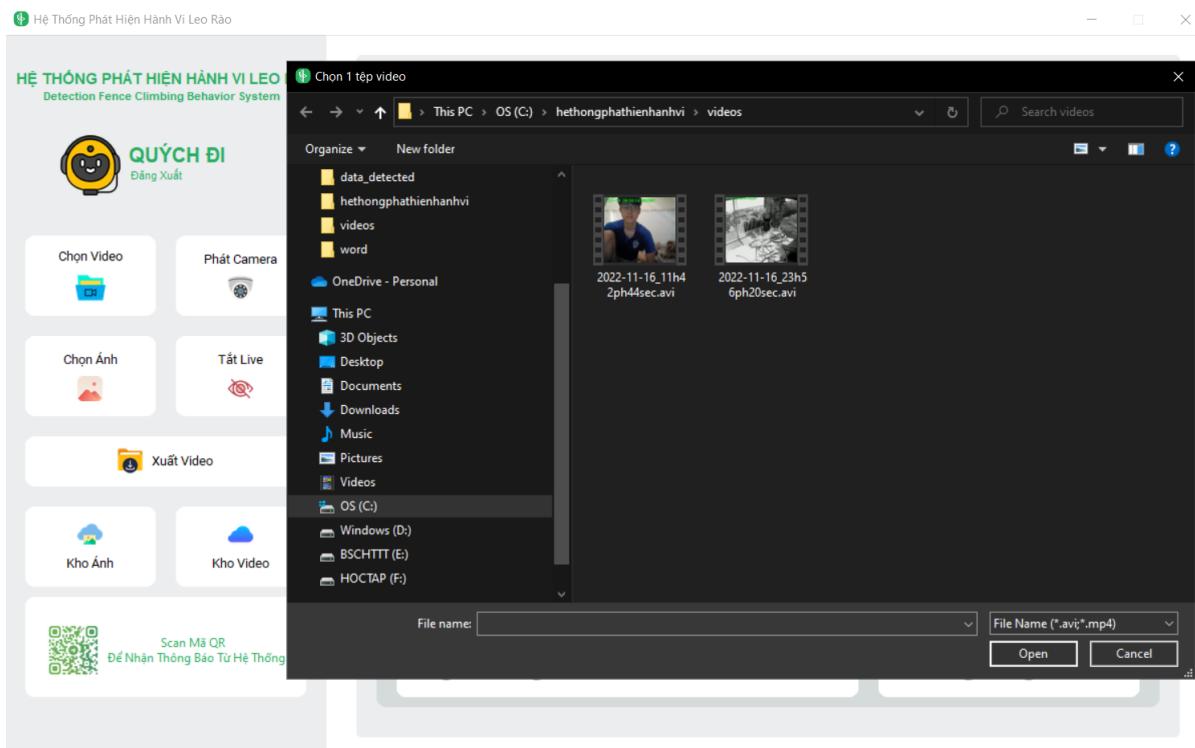
Hình 4.5 Chức năng Phát Video yêu cầu chọn file Video



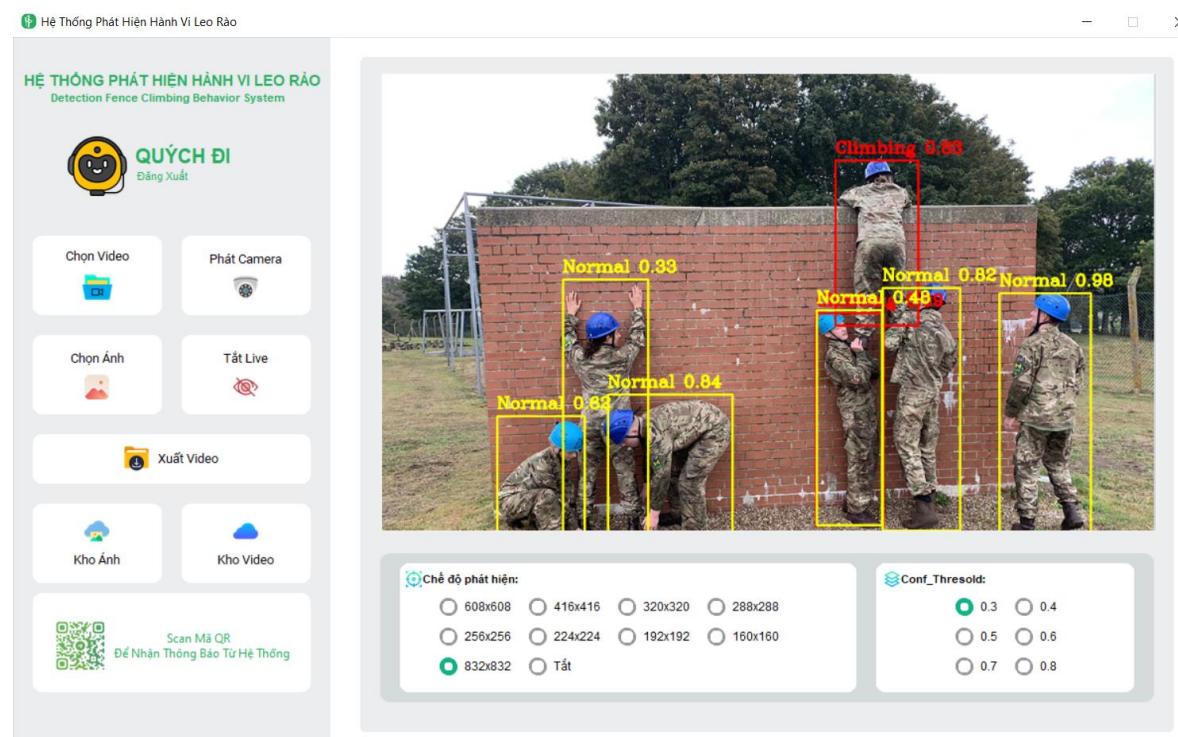
Hình 4.6 Giao diện chức năng Xuất Video



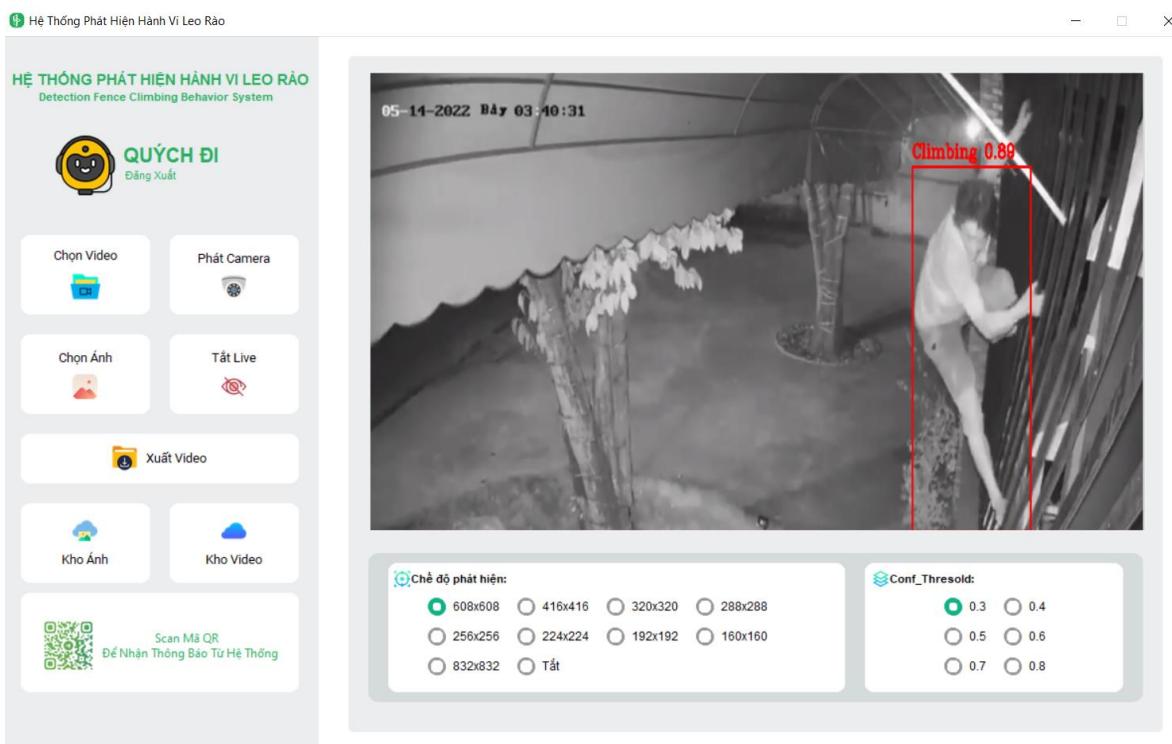
Hình 4.7 Giao diện chức năng Kho Ánh



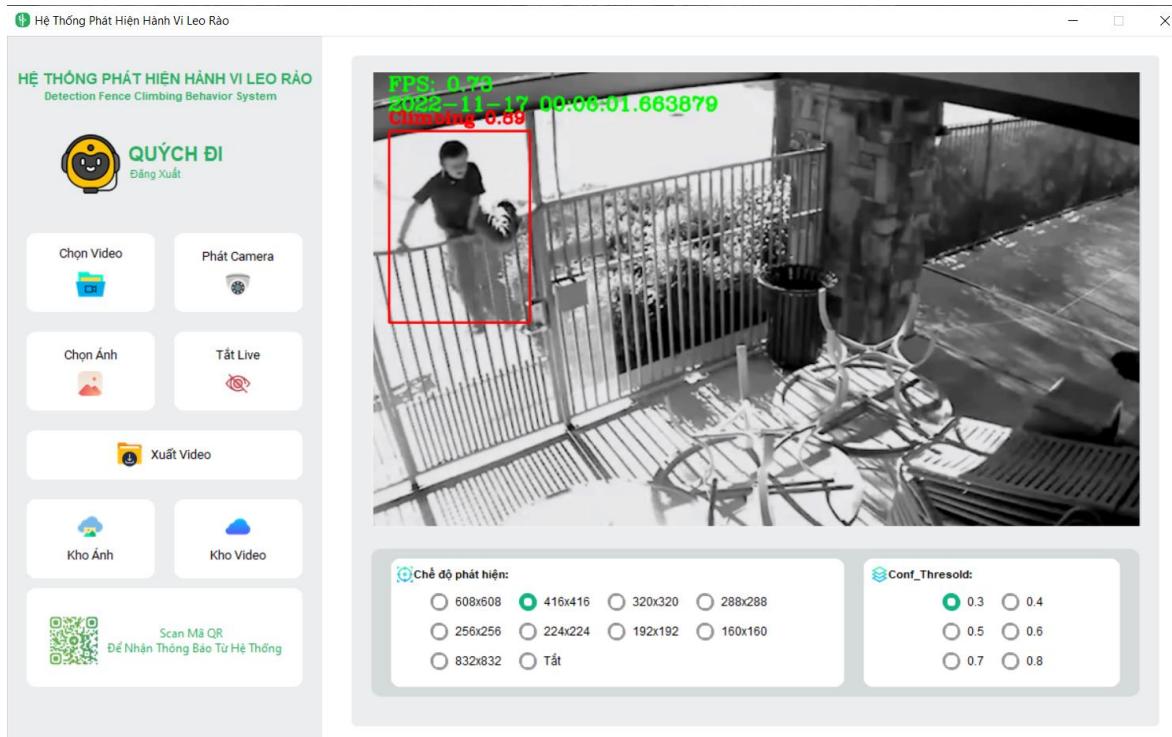
Hình 4.8 Chức năng Kho Video



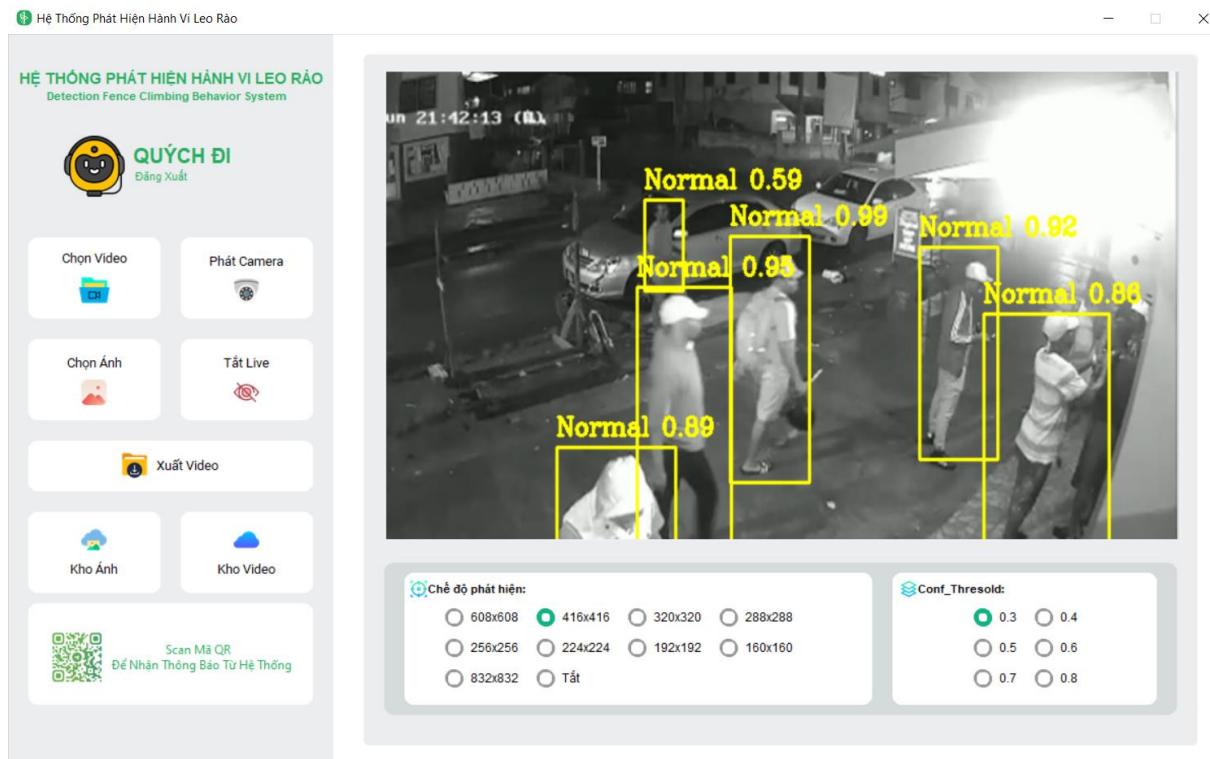
Hình 4.9 Kết quả phát hiện hành vi bằng chức năng Phát Ánh (1)



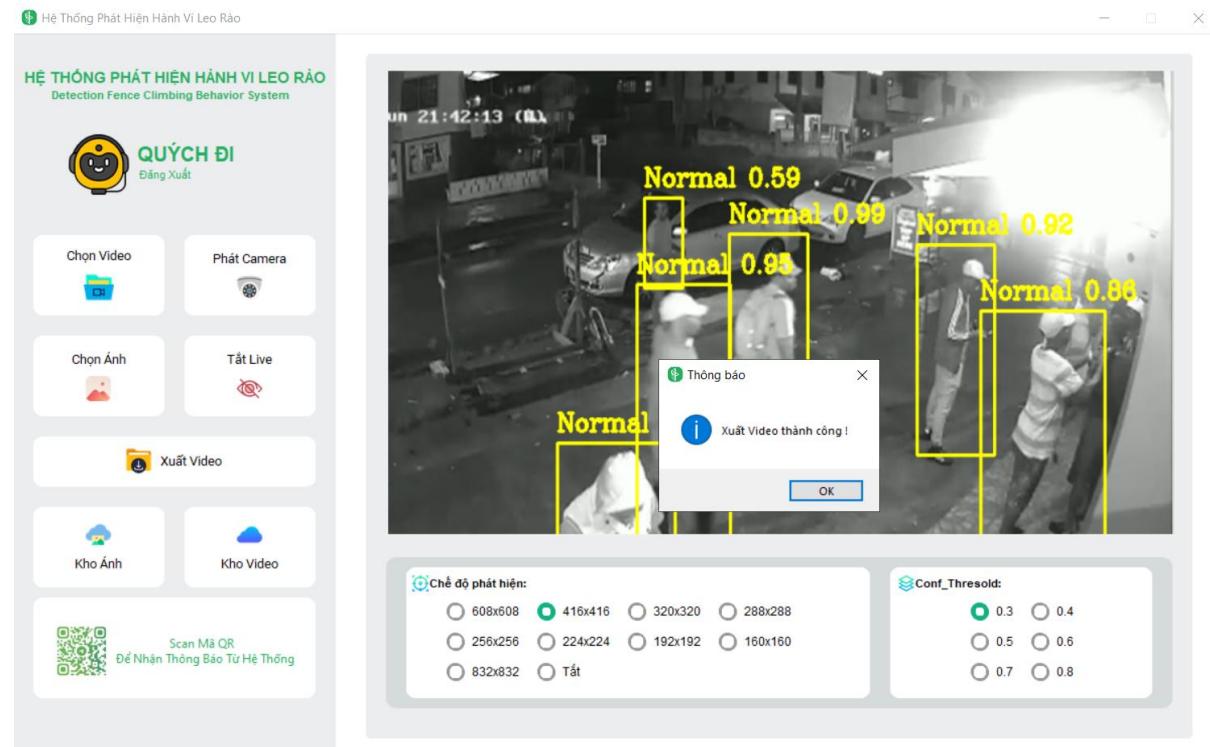
Hình 4.10 Kết quả phát hiện hành vi bìng chúc năng Phát Ảnh (2)



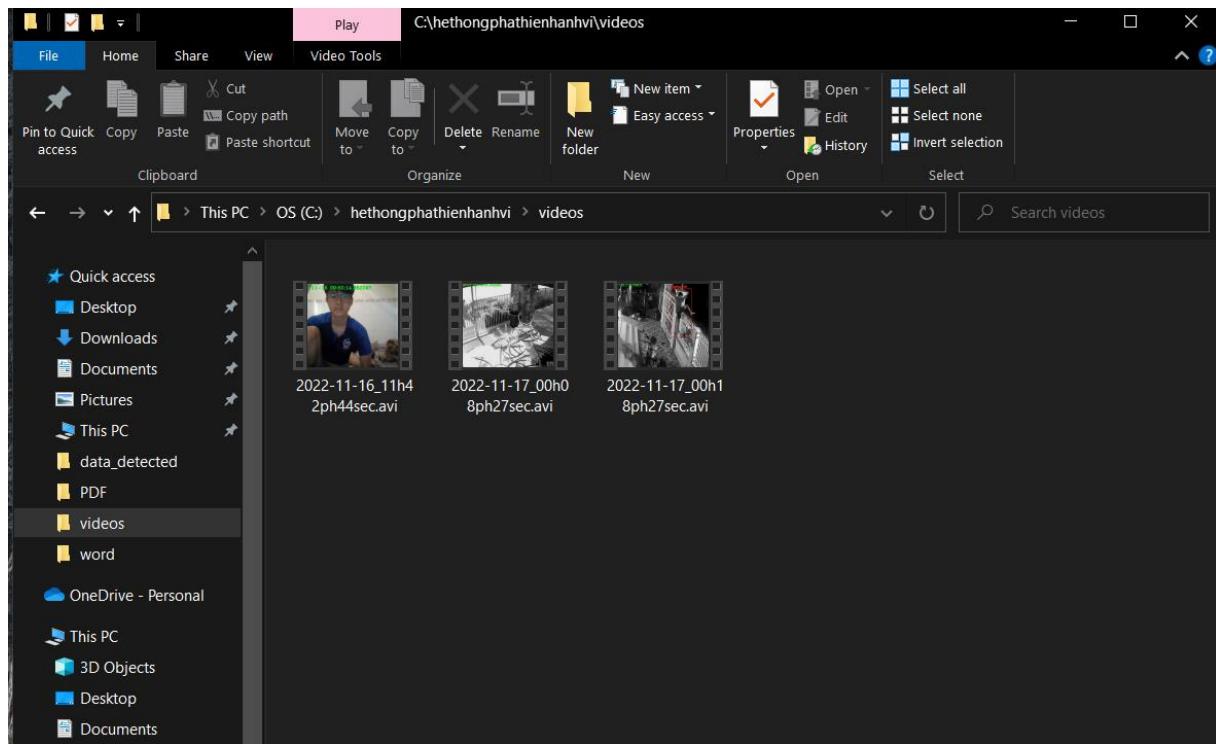
Hình 4.11 Kết quả phát hiện hành vi bìng chúc năng Phát Video (1)



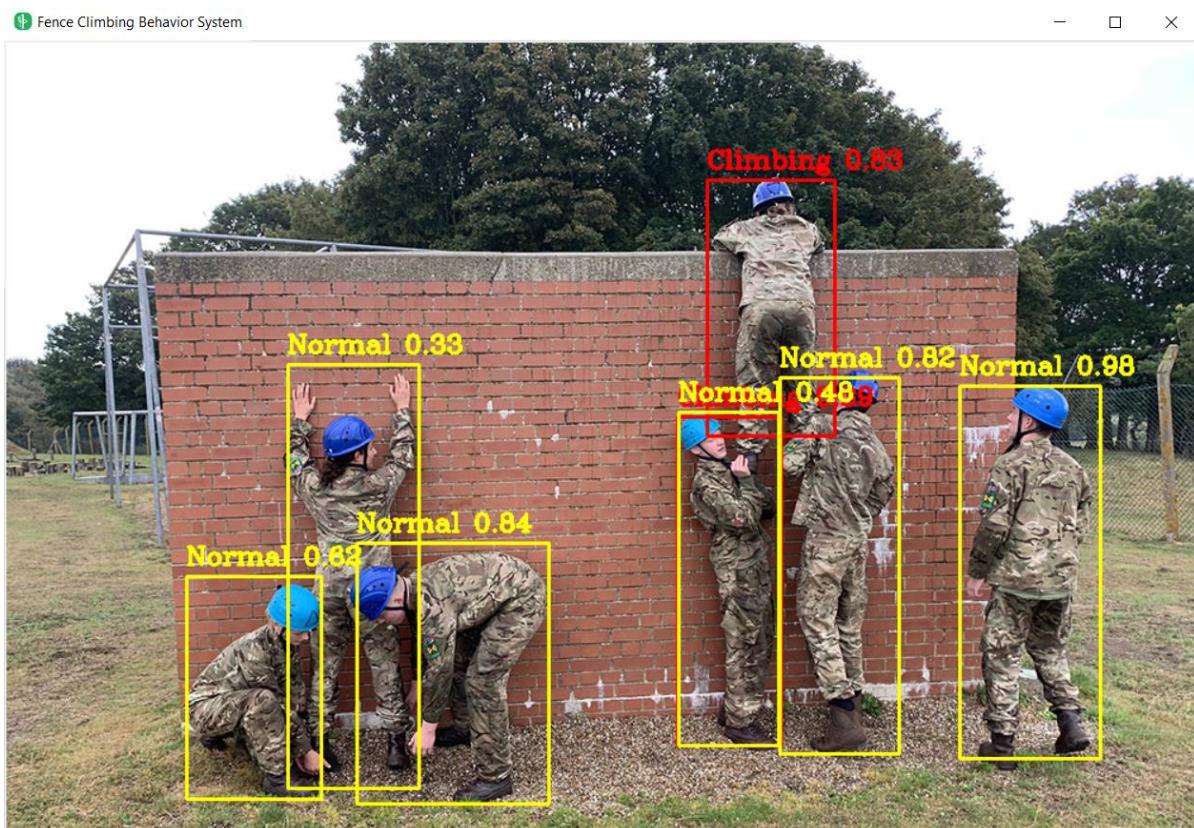
Hình 4.12 Kết quả phát hiện hành vi bắc chúc năng Phát Video (2)



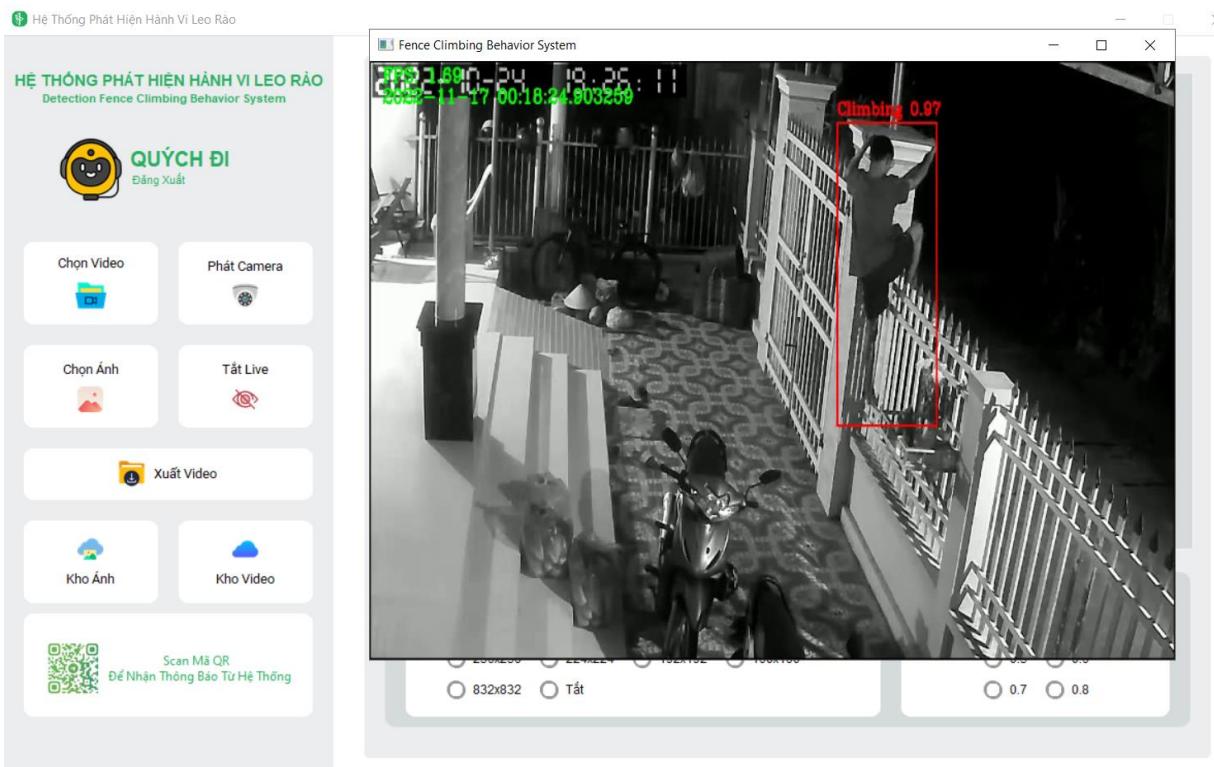
Hình 4.13 Kết quả thông báo xuất video thành công



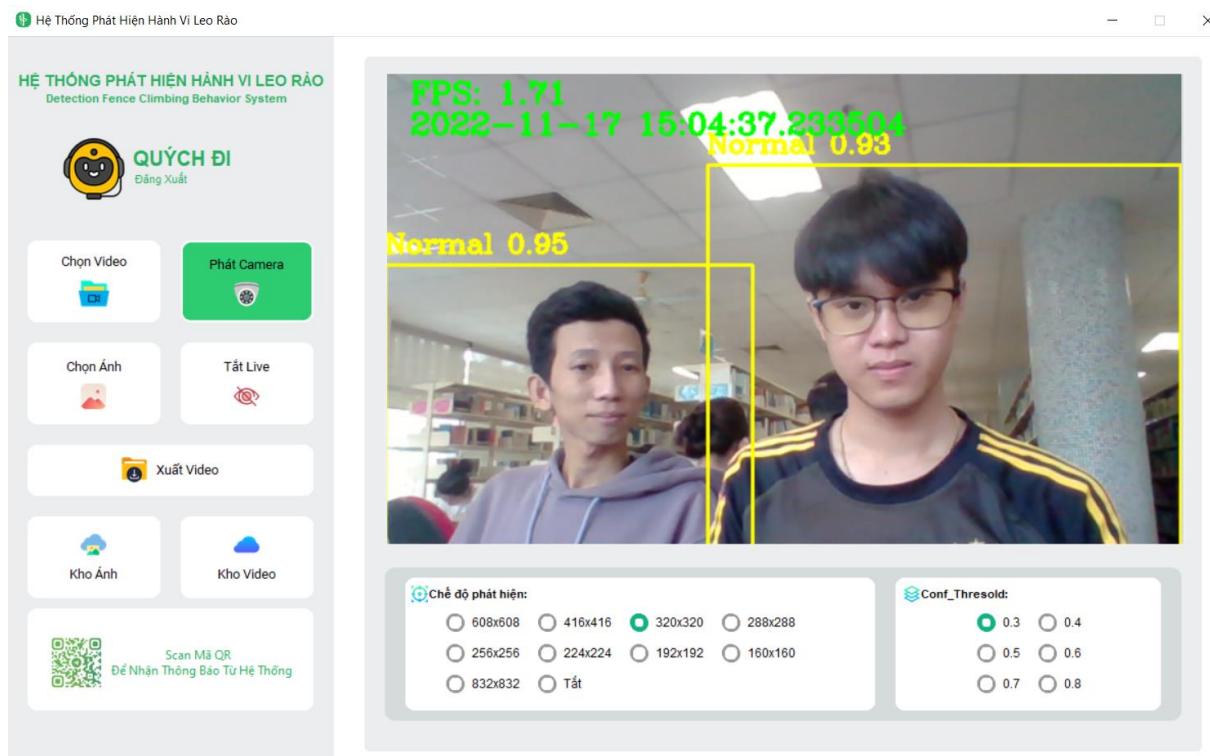
Hình 4.14 Video xuất ra được lưu trữ trên Kho Video



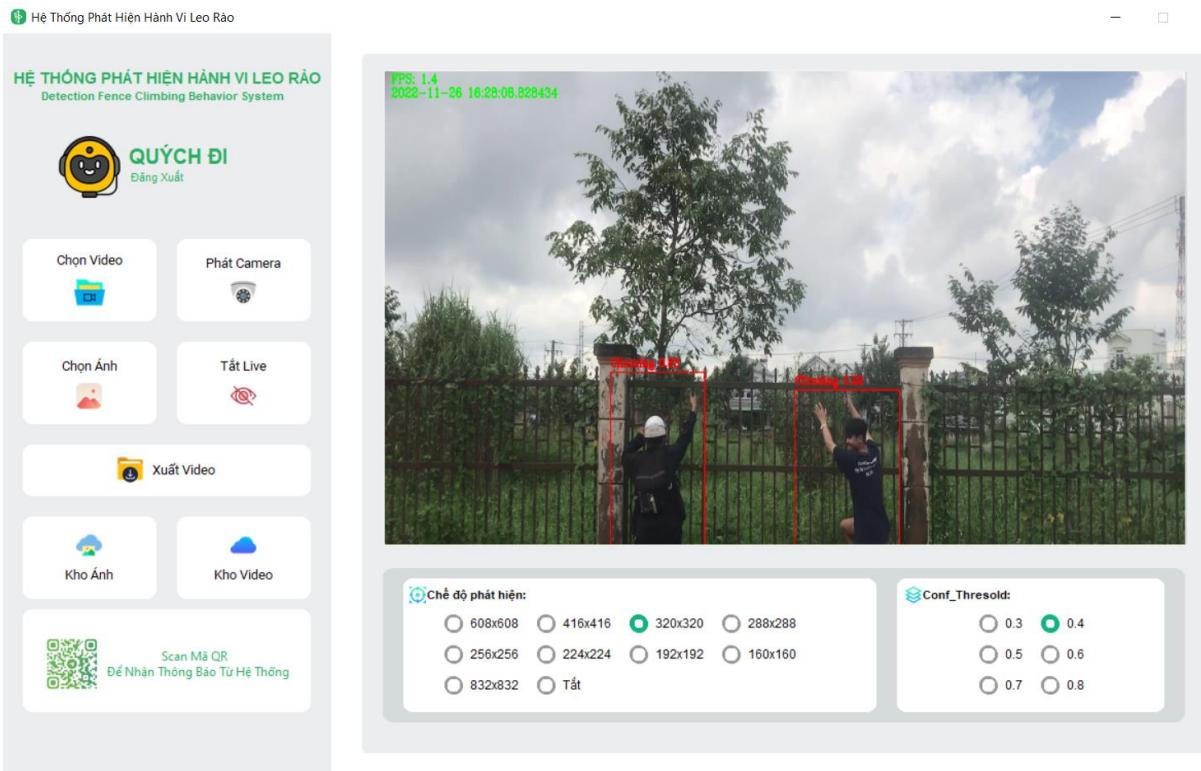
Hình 4.15 Kết quả người dùng chọn hình ảnh để xem lại



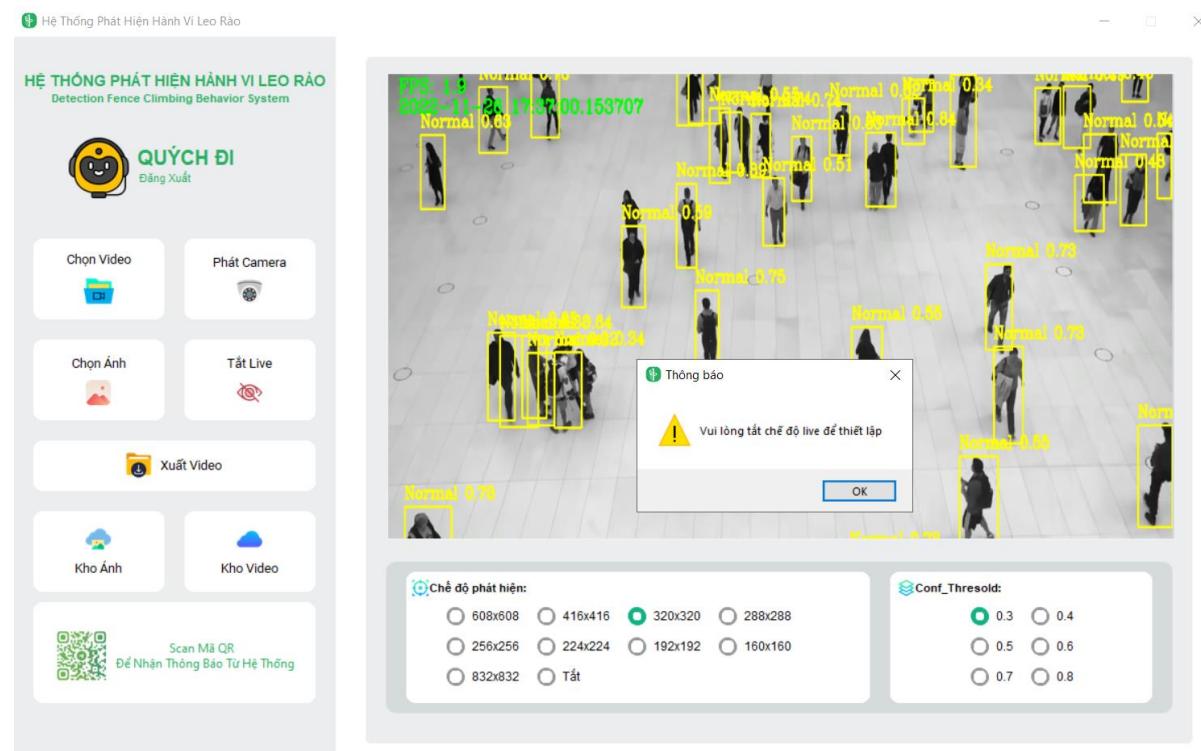
Hình 4.16 Kết quả người dùng chọn video xem lại video



Hình 4.17 Kết quả hệ thống phát hiện bằng chức năng Phát Camera (1)



Hình 4.18 Kết quả hệ thống phát hiện bằng chức năng Phát Camera (2)



Hình 4.19 Hệ thống yêu cầu tắt chế độ Live

4.3. Tổng kết chương

Chương này mô tả chi tiết về kết quả đánh giá thực nghiệm độ hiệu suất trên mô hình YOLO v4 và trên hệ thống phát hiện hành vi leo rào

CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Kết luận

Sau khi quá trình thực hiện đề tài **Hệ thống phát hiện hành vi leo rào** kết quả đạt được như sau:

- Xây dựng và lấy kết quả đánh giá, so sánh thực nghiệm giữa các mô hình YOLO v4, YOLO v7, SSD và Faster-RCNN.
- Chọn được mô hình có tính thử nghiệm và hiệu suất tốt để xây dựng lên hệ thống ứng dụng
- Đạt được mục tiêu của đề tài đã đặt ra trong **Chương 1**.

Bên cạnh những kết quả đạt được của đề tài thì không thể thiếu được những mặt hạn chế của đề tài:

➤ Về phía đào tạo mô hình

- Các mô hình Faster-RCNN và YOLO v7 chưa lấy được đầy đủ các độ đo.

➤ Về phía xây dựng hệ thống

- Tốc độ xử lý phát hiện đối tượng trên video/ camera trên hệ thống vẫn còn chậm khi cho đầu vào với độ phân giải cao.

5.2. Hướng phát triển

- Thu thập các dữ liệu chuẩn bị cho việc đào tạo có kích thước to rõ, độ phân giải cao ít nhiễu. Nếu tập dữ liệu là một ảnh trắng đen có găng thu thập các dữ liệu rõ ràng chính xác độ phân giải tốt nhất có thể.
- Cân bằng dữ liệu 50% cho hành vi leo rào và 50% cho hành vi bình thường tức là chứa hành vi đi đứng, ngồi, v.v...
- Về phía hệ thống: Xây dựng hệ thống có tốc độ phát hiện xử lý nhanh với đầu vào có độ phân giải cao, khoảng cách phát hiện được đối tượng cần được mở rộng từ 5 đến 6m trở lên. Mô hình tích hợp trong hệ thống phát hiện đủ tốt không bỏ sót đối tượng. Kết hợp quản lý cơ sở dữ liệu chứa các thông tin người dùng.

TÀI LIỆU THAM KHẢO

- [1] H.-Y. M. L.-H. Y. Y.-H. W. P.-Y. C. J.-W. H. Chien-Yao Wang, “CSPNET: A NEW BACKBONE THAT CAN ENHANCE LEARNING,” 27 11 2019. [Trực tuyến]. Available: <https://arxiv.org/pdf/1911.11929.pdf>. [Đã truy cập 11 11 2022].
- [2] T. N. M. C. ThS. Vũ Văn Rực, “Nghiên cứu thuật toán phân loại phương tiện giao thông dựa trên thị giác máy tính,” 25 6 2021. [Trực tuyến]. Available: <https://sti.vista.gov.vn/tw/Lists/TaiLieuKHCN/Attachments/325988/CVb12S72021092.pdf>. [Đã truy cập 1 11 2022].
- [3] V. T. H. Huy Nguyen Quoc, “Real-Time Human Ear Detection Based on the Joint of Yolo and RetinaFace,” 08 11 2021. [Trực tuyến]. Available: <https://www.hindawi.com/journals/complexity/2021/7918165/>. [Đã truy cập 22 11 2022].
- [4] X. Y. Q. T. Jing Yu, “Traffic Sign Detection and Recognition in Multiimages Using a Fusion Model With YOLO and VGG Network,” IEEE, 2 5 2022. [Trực tuyến]. Available: <https://ieeexplore.ieee.org/document/9766167>. [Đã truy cập 1 11 2022].
- [5] S. H. M. L. Haiguang Chen, “The Elderly Fall Detection Algorithm Based on Human Joint Extraction and Object Detection,” 11 12 2020. [Trực tuyến]. Available: <https://ijcjournal.org/index.php/InternationalJournalOfComputer/article/view/1852>. [Đã truy cập 12 11 2022].
- [6] A. M. R. N. Z. Z. M.M.M. Al-Khalidy, “Impact of performance: SSD vs YOLOV3 machine learning for mask detection and temperature sensing device,” IET, 09 05 2022. [Trực tuyến]. Available: <https://ieeexplore.ieee.org/document/9770676>. [Đã truy cập 1 12 2022].
- [7] K. K. Tejal Palwankar, “Real Time Object Detection using SSD and MobileNet,” 3 2022. [Trực tuyến]. Available: <https://www.ijraset.com/best-journal/real-time-object-detection-using-ssd-and-mobilenet>. [Đã truy cập 1 11 2022].
- [8] D. A. D. E. C. S. S. R. C.-Y. F. & A. C. B. Wei Liu, “Single Shot MultiBox Detector,” Springer, 14 11 2016. [Trực tuyến]. Available: https://link.springer.com/chapter/10.1007/978-3-319-46448-0_2. [Đã truy cập 12 11 2022].
- [9] Y. L. & D. Z. Husheng Pan, “Recognizing human behaviors from surveillance videos using the SSD algorithm,” Springer, 4 1 2021. [Trực tuyến]. Available: <https://link.springer.com/article/10.1007/s11227-020-03578-3>. [Đã truy cập 10 11 2022].

- [10] R. t. f. m. d. w. SSD, "Real time face mask detection with SSD," IEEE, 1 12 2021. [Trực tuyến]. Available: <https://ieeexplore.ieee.org/document/9626095>. [Đã truy cập 2 11 2022].
- [11] H. T. N. V. C. T. H. X. P. J. P. A. Q. Nguyen, "A Visual Real-time Fire Detection using Single Shot MultiBox Detector for UAV-based Fire Surveillance," IEE, [Trực tuyến]. Available: <https://ieeexplore.ieee.org/document/9352080>. [Đã truy cập 10 11 2022].
- [12] S. D. R. G. A. F. Joseph Redmon, "You Only Look Once Unified, Real-Time Object Detection," 9 5 2016. [Trực tuyến]. Available: <https://arxiv.org/pdf/1506.02640.pdf>. [Đã truy cập 12 11 2022].
- [13] C.-Y. W. H.-Y. M. L. Alexey Bochkovskiy, "YOLOv4: Optimal Speed and Accuracy of Object Detection," 23 4 2020. [Trực tuyến]. Available: <https://arxiv.org/pdf/2004.10934v1.pdf>. [Đã truy cập 27 11 2022].
- [14] hamhochoi, "Yolov4," Hamhochoi, 8 5 2020. [Trực tuyến]. Available: <https://dothanhblog.wordpress.com/2020/05/08/yolov4/>. [Đã truy cập 22 11 2022].
- [15] L. T. Toan, "HỆ THỐNG SỐ HÓA TỰ ĐỘNG GIẤY CHỨNG NHẬN QUYỀN SỬ DỤNG ĐẤT," Cần Thơ, 2021.
- [16] T. V. Huy, "YOLOv3 Core Ideas," huytranvan2010, 06 07 2021. [Trực tuyến]. Available: https://huytranvan2010.github.io/YOLOv3-Core-Ideas/?utm_source=zalo&utm_medium=zalo&utm_campaign=zalo. [Đã truy cập 30 11 2022].
- [17] A. B. a. H.-Y. M. L. Chien-Yao Wang, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object," 6 7 2022. [Trực tuyến]. Available: <https://arxiv.org/abs/2207.02696>. [Đã truy cập 27 11 2022].
- [18] M. Nguyen, "[Paper Explain] YOLOv7: Sử dụng các "trainable bag-of-freebies" đưa YOLO lên một tầm cao mới (phần 3)," Viblo, 10 10 2022. [Trực tuyến]. Available: <https://viblo.asia/p/paper-explain-yolov7-su-dung-cac-trainable-bag-of-freebies-dua-yolo-len-mot-tam-cao-moi-phan-3-018J253M4YK>. [Đã truy cập 28 11 2022].
- [19] I.-H. Y. H.-Y. M. L. Chien-Yao Wang, "You Only Learn One Representation: Unified Network for Multiple Tasks," 10 5 2021. [Trực tuyến]. Available: <https://arxiv.org/pdf/2105.04206.pdf>. [Đã truy cập 29 11 2022].
- [20] A. H. M. Z. A. Z. L.-C. C. Mark Sandler, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 21 3 2019. [Trực tuyến]. Available: <https://arxiv.org/pdf/1801.04381.pdf>. [Đã truy cập 9 12 2022].
- [21] K. H. R. G. a. J. S. Shaoqing Ren, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," 6 1 2016. [Trực tuyến]. Available: <https://arxiv.org/pdf/1506.01497.pdf>. [Đã truy cập 26 11 2022].

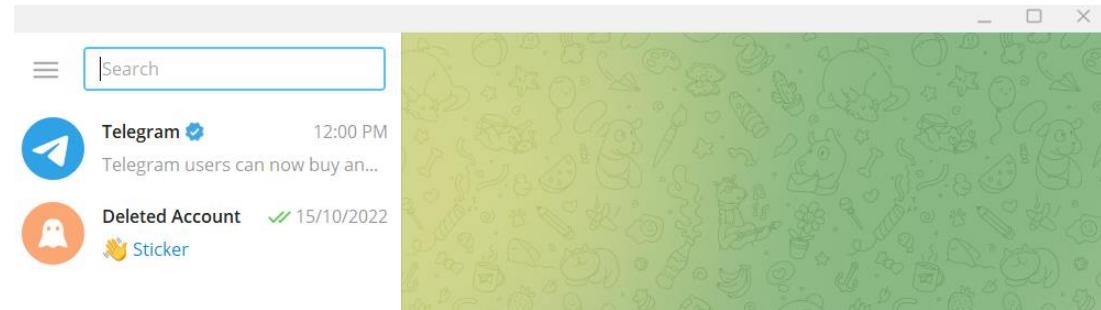
- [22] X. Z. S. R. J. S. Kaiming He, “Deep Residual Learning for Image Recognition,” 10 12 2015. [Trực tuyến]. Available: <https://arxiv.org/pdf/1512.03385.pdf>. [Đã truy cập 10 12 2022].
- [23] K. Rzechowski, “instance segmentation evaluation criteria,” REASONFIELDLAB, 16 8 2022. [Trực tuyến]. Available: <https://www.reasonfieldlab.com/post/instance-segmentation-evaluation-criteria>. [Đã truy cập 12 11 2022].
- [24] P. W. W. L. J. L. R. Y. D. R. Zhaozheng, “Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression,” 19 11 2019. [Trực tuyến]. Available: <https://arxiv.org/pdf/1911.08287.pdf>. [Đã truy cập 10 12 2022].
- [25] A. S. F. S. A. C. Anirudha Ghosh, “Fundamental Concepts of Convolutional Neural Network,” [Trực tuyến]. Available: https://www.researchgate.net/publication/337401161_Fundamental_Concepts_of_Convolutional_Neural_Network.
- [26] S. A. Afshin Amidi, “CS 230 - Deep Learning,” [Trực tuyến]. Available: <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks>.
- [27] X. Z. S. R. a. J. S. Kaiming He, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” [Trực tuyến]. Available: <https://arxiv.org/pdf/1406.4729.pdf>.
- [28] L. Q. H. Q. J. S. J. J. Shu Liu, “Path Aggregation Network for Instance Segmentation,” [Trực tuyến]. Available: <https://arxiv.org/pdf/1803.01534.pdf>.
- [29] H. Cường, “Tìm hiểu cấu trúc YOLOv1,v2,v3 và v4 – Phần 2,” DevAI, 24 2 2021. [Trực tuyến]. Available: <https://devai.info/2021/02/24/series-yolo-4-tim-hieu-cau-truc-yolov1v2v3-va-v4-phan-2/#:~:text=YOLOv4%2D%20Optimal%20Speed%20and%20Accuracy%20of%20Object%20Detection%E2%80%8B.&text=C%E1%BA%A5u%20tr%C3%BAc%20c%E1%BB%A7a%20v4%20%C4%91%C6%B0%E1%BB%A3c,detect>. [Đã truy cập 20 11 2022].
- [30] B. Q. Manh, “RepVGG - Sự trở lại của một tượng đài,” Viblo, 5 8 2021. [Trực tuyến]. Available: <https://viblo.asia/p/repvgg-su-tro-lai-cua-mot-tuong-dai-GrLZDGenKk0>. [Đã truy cập 29 11 2022].

PHỤ LỤC

7.1. Hướng dẫn cài đặt telegram và tạo chat bot

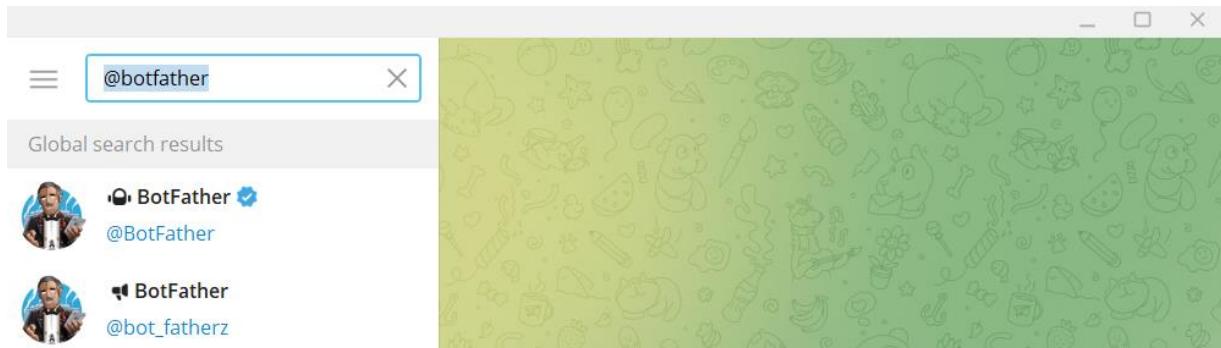
Hướng dẫn tích hợp Chat bot ứng dụng Telegram vào hệ thống phát hiện hành vi leo rào. Trước khi tạo được chatbot cần phải cài đặt sẵn ứng dụng telegram phiên bản web hoặc Adroid hoặc IOS. Phần này hướng dẫn tạo chatbot bằng phiên bản Telegram trên CPU đã cài đặt trên máy tính.

Bước 1: Tạo tài khoản người và đăng nhập vào ứng dụng Telegram **Phụ lục hình 1**



Phụ lục hình 1 Giao diện ứng dụng Telegram sau khi đăng nhập thành công

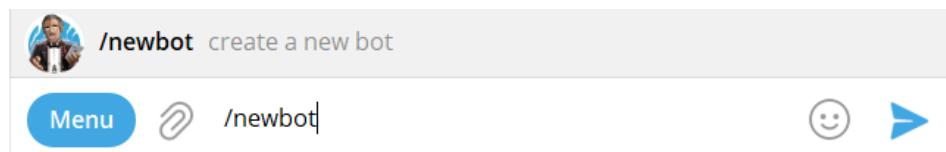
Bước 2: Gõ vào ô tìm kiếm (Search) '@botfather'. Chọn chào tài khoản BotFather có dấu tick xanh (được đánh dấu tài khoản chính chủ của BotFather) như **Phụ lục hình 2**.



Phụ lục hình 2 Tìm kiếm BotFater

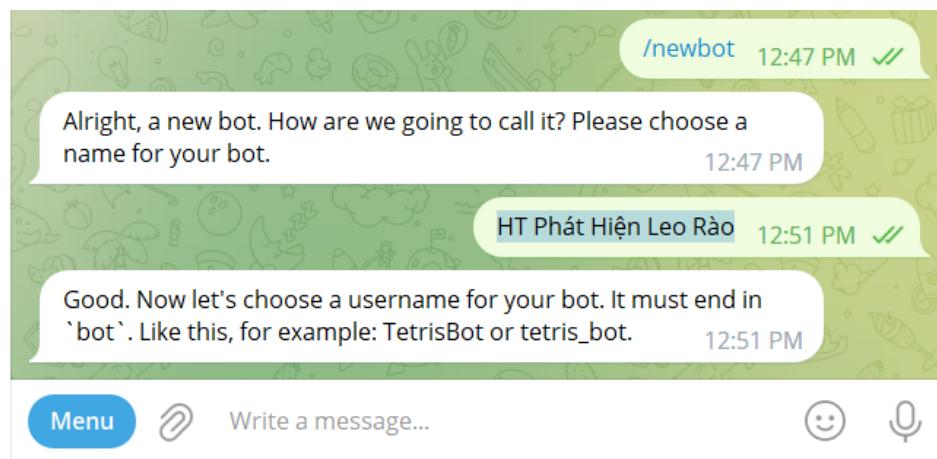
Bước 3: Chọn vào nút Start nằm cùng phần ô soạn tin nhắn

Bước 4: Gõ vào ô soạn tin nhắn '/newbot' như **Phụ lục hình 3**. Chọn nút gửi tin nhắn.



Phụ lục hình 3 Tạo chat bot

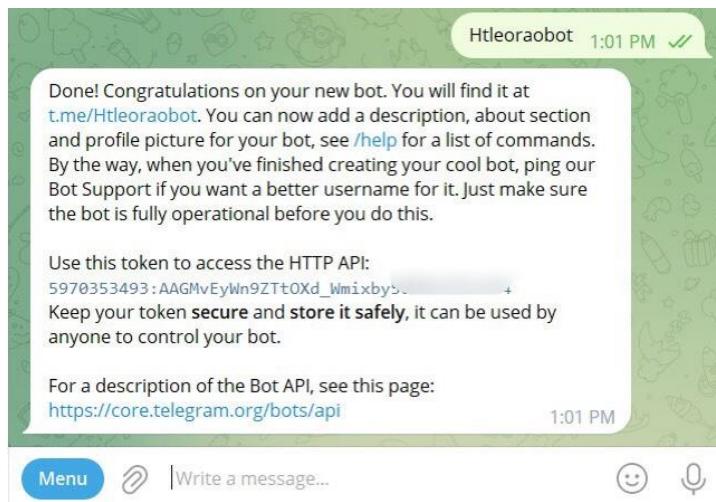
Bước 5: Sau khi hoàn thành **bước 4**. BotFather yêu cầu đặt tên cho chat bot như **Phụ lục hình 4**



Phụ lục hình 4 Đặt tên chat bot thành công

Bước 6: Gõ tên vào ô soạn tin ‘HT Phát Hiện Leo Rào’ để đặt tên, sau đó nhấn vào nút gửi tin nhắn. Lúc này BotFather thông báo tên chat bot tạo thành công như **Phụ lục hình 4** và yêu cầu đặt username.

Bước 7: Gõ vào ô tin nhắn ‘Htleoraobot’ để đặt username như



Phụ lục hình 5 kết quả đặt username ‘Htleoraobot’

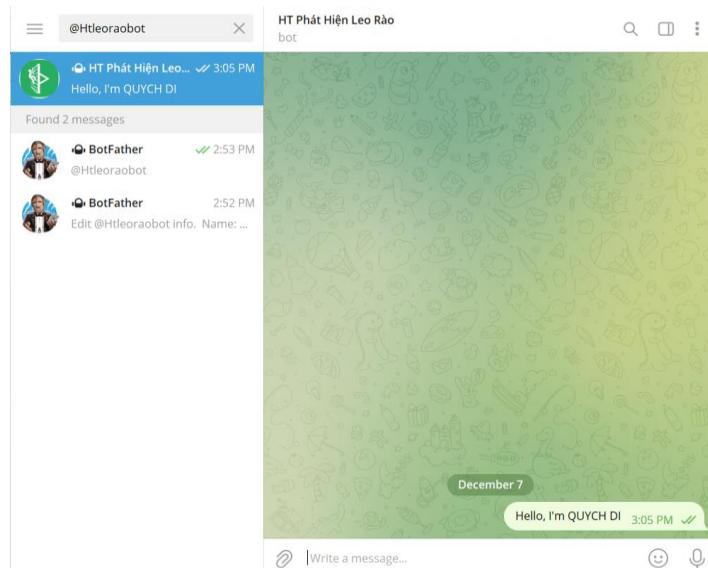
Khi đặt username đặt thành công, BotFather gửi Token để kết nối với API của Telegram như **Phụ lục hình 5**. Sử dụng Token này để tích hợp vào hệ thống phát hiện hành vi leo rào.

Tiếp theo kiểm tra chat bot đã tạo đã tồn tại chưa bằng cách gõ tên username của chat bot đã tạo ở **bước 7** trên thanh tìm kiếm (search). Chat bot tạo thành công có kết quả như **Phụ lục hình 6**.



Phụ lục hình 6 Kiểm tra chat bot

Tiếp theo lấy ID người dùng bằng cách nhấn với nội dung bất kì vào ô soạn tin nhắn trong chat bot HT Phát Hiện Leo Rào theo như **Phụ lục hình 7**



Phụ lục hình 7 Người dùng nhắn tin với chat bot

Bước 9: Sau khi nhắn tin với chat bot, truy cập vào đường dẫn '[https://api.telegram.org/bot\[TOKEN\]/getUpdates](https://api.telegram.org/bot[TOKEN]/getUpdates)' để lấy ID người dùng. Thay [TOKEN] bằng token nhận được ở kết quả của **bước 7**.

```
← → C api.telegram.org/bot5970353493:AAGMvEyWn9ZTHOXd_Wmixby5081Kof5sCE4/getUpdates
https://towardsdata... Khoa học dữ liệu epoch GitHub - AlexeyAB... Understanding dark... Thẻ mới Darknet YOLOv4... BẢO cáo lý THUYẾT... Tim Hiểu Mô Hình... VAAN DỰNG TÓ T...
{"ok":true,"result":[{"update_id":520076793,"message":{"message_id":1,"from":{"id":5642582108,"is_bot":false,"first_name":"WD","last_name":"SI","language_code":"en"},"chat": {"id":5642582108,"first_name":"WD","last_name":"SI","type":"private"}, "date":1670394244,"text":"/start","entities":[{"offset":0,"length":6,"type":"bot_command"}]}],("update_id":520076793,"message":{"message_id":2,"from":{"id":5642582108,"is_bot":false,"first_name":"WD","last_name":"SI","language_code":"en"},"chat": {"id":5642582108,"first_name":"WD","last_name":"SI","type":"private"}, "date":1670394246,"text":"quychdi")}}
```

Phụ lục hình 8 Lấy ID người dùng

Theo **Phụ lục hình 8**, ID mà người dùng đã thực hiện gửi tin nhắn với chat bot 'HT Phát Hiện Leo Rào' có ID là: 5642582108 trong phần được tô đậm. Kết thúc quá trình tạo chat bot.

Kết quả cần đạt:

- Lấy được TOKEN
- Lấy được ID người dùng

7.2. Tích hợp Telegram vào hệ thống.

Sau khi lấy được TOKEN và ID người dùng.

Đầu tiên cài đặt gói python-telegram-bot: **py install python-telegramme-bot**

Sau đó sử dụng đoạn code trên **Phụ lục hình 9**. Đặt tên file ‘text_telegram.py’ và tạo hàm send_telegram().

```
*text_telegram.py - Notepad
File Edit Format View Help
from re import T
import telegram
from datetime import datetime
// lay TOKEN và ID chat bot
TELEGRAM_BOT_TOKEN = '5970353493:AAGMvEyWn9ZTtoxd_Wmixby5081KofSsCE4'
TELEGRAM_CHAT_ID = '5642582108'

// Thiết lập hàm gửi telegram
def send_telegram():
    try:
        # PHOTO_PATH = photo_path
        bot = telegram.Bot(token=TELEGRAM_BOT_TOKEN)
        bot.send_photo(chat_id=TELEGRAM_CHAT_ID, photo=open(
            'data_detected/phathienleorao.png', 'rb'))
        bot.send_message(chat_id=TELEGRAM_CHAT_ID,
                         text="Có hành vi leo rào \n ngày: " + str(datetime.now()))
    except Exception as ex:
        print("erro:", ex)
```

Phụ lục hình 9 Thiết lập ID và TOKEN người dùng vào hệ thống

Gọi hàm send_telegram từ text_telegram.py như **Phụ lục hình 10** vào phần code chính (ví dụ app.py)

```
from text_telegram import send_telegram
```

Phụ lục hình 10 Gọi hàm send_telegram vào app.py

Tạo hàm alert() và gọi send_telegram vào alert() như **Phụ lục hình 11**.

```
def alert():
    global nCount
    thred = Thread(target=send_telegram)
    thred.start()
    return nCount == 0
    # send_to_user(img)
# KẾT THÚC HÀM GỬI THÔNG BÁO-----
```

Phụ lục hình 11 Thiết lập hàm alert()

Gọi hàm alert() vào để xử lý phát hiện đối tượng.

```
# Kiểm tra frame leo rào
if isExit == True:
    nCount += 1
    if nCount > 8:
        cv2.imwrite('data_detected/phathienleorao.png', frame)
        alert()
        print(nCount)
        nCount = 0
    # trên 10 frame gửi thông báo
else:
    isExit = False
    nCount = 0
return cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
```

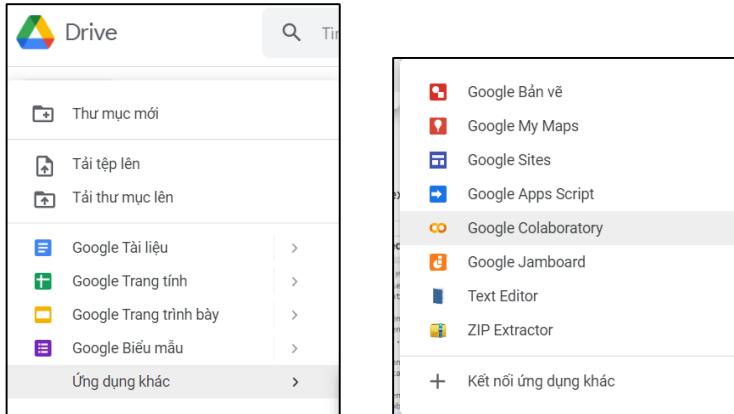
Phụ lục hình 12 Kiểm tra số frame chứa đối tượng leo rào và gửi cảnh báo

Trước khi gọi hàm alert(), đầu tiên phải kiểm tra frame có chứa đối tượng leo rào nào hay không, nếu có frame có đối tượng leo rào phát hiện liên tục từ 8 frame trở lên thì xuất 1 frame ảnh chứa đối tượng được phát hiện lưu vào hệ thống và gọi hàm alert().

7.3. Đào tạo các mô hình YOLO v4, YOLO v7, SSD và Faster-RCNN

Môi trường đào tạo các mô hình dùng Google Colaboratory tận dụng các RAM và GPU cao cấp miễn phí từ Google Colaboratory.

Để sử dụng môi trường đào tạo trên Google Colaboratory chọn vào Drive → Ứng dụng khác → Google Colaboratory. Mô phỏng cách kết nối như **Phụ lục hình 13**



Phụ lục hình 13 Kết nối Google Colab

7.3.1. Cách Đào tạo mô hình YOLO v4

❖ Chuẩn bị dữ liệu

Tạo thư mục data chứa tất cả file ảnh và nhãn cụ thể có 4882 file ảnh và 4882 file nhãn nằm trong cùng thư mục

❖ Đào tạo mô hình

Bước 1: Kết nối Google Colab với Drive

```
# Step 1. Mount drive
from google.colab import drive
drive.mount('/content/gdrive')
```

Phụ lục hình 14 Kết nối Google Coalb với Drive

Bước 2: Cài đặt môi trường darknet cho YOLO v4 về drive như **Phụ lục hình 15**.

```
[ ] # Step 2. Tai ma nguon YOLO ve drive
!rm -rf darknet
%cd /content/gdrive/My\ Drive
!git clone https://github.com/AlexeyAB/darknet
%cd /content/gdrive/My\ Drive/darknet
!rm -rf data
!mkdir data
```

Phụ lục hình 15 cài đặt môi trường Darknet

Môi trường yolov4 được cài đặt về thành công trên drive như **Phụ lục hình 16**.



Phụ lục hình 16 Cài đặt thành công Darknet

Bước 3. Tạo file yolo.names vào trong darknet như **Phụ lục hình 17**.

Đề tài tập trung vào hai đối tượng ‘climbing và normal’

```
#Tạo file yolo.names
%cd /content/gdrive/My\ Drive/darknet
!echo "climbing" > yolo.names
!echo "normal" >> yolo.names
```

Phụ lục hình 17 Tạo file yolo.names

Bước 4. Upload dữ liệu đã chuẩn bị vào thư mục Darknet/ data (**Phụ lục hình 18**)

Drive của tôi > darknet			
Tên	Chú sở hữu	Sửa đổi lần cuối	Kích cỡ tệp
cfg	tôi	14 thg 8, 2022	—
cmake	tôi	14 thg 8, 2022	—
data	tôi	14 thg 8, 2022	—

Phụ lục hình 18 Thư mục darknet/data

Bước 5: Cấu hình lại file config bằng cách vào Darknet → cfg → yolov4-custom.cfg

Drive của tôi > darknet > cfg			
Tên ↑	Chủ sở hữu	Sửa đổi lần cuối	Kích cỡ tệp
yolov4-csp-x5w1sr1r.cfg	tôi	14/08/2022 10:14	13 KB
yolov4-cfg	tôi	14/08/2022 10:14	13 KB
yolov4-custom.cfg	tôi	14/08/2022 10:14	12 KB

Phụ lục hình 19 File yolov4-custom.config chứa trong cfg

Tiến hành cấu hình

```
yolov4-custom.cfg x
1 [net]
2 # Testing
3 #batch=1
4 #subdivisions=1
5 # Training
6 batch=16
7 subdivisions=32
8 width=416
9 height=416
10 channels=3
11 momentum=0.949
12 decay=0.0005
13 angle=0
14 saturation = 1.5
15 exposure = 1.5
16 hue=.1
17
18 learning_rate=0.001
19 burn_in=1000
20 max_batches = 6000
21 policy=steps
22 steps=4800,5400
23 scales=.1,.1
24
25 #cutmix=1
26 mosaic=1
27
28 #:104x104 54:52x52 85:26x26 104:13x13 for 416
29
30 [convolutional]
31 batch_normalize=1
```

Phụ lục hình 20 file yolov4-custom.cfg

- Thay đổi ở dòng 20 **max_batches = 6000** lý do max(số class*2000, 6000).
- Dòng 22 số bước steps = 4800, 5400.
- Chính sửa toàn bộ các khác có số khác 2 ở dòng 970, 1058, 1146 (vì lý do bài toán phát hiện hành vi leo rào chỉ cần 2 lớp climbing và normal).
- Chính sửa filters ở dòng 963, 1051, 1139 **filters=21** (filters=<số class + 5)*3)
- Thay đổi **subdivisions = 32** (dòng 7)
- Thay kích thước ảnh đầu vào ở dòng 8 và dòng 9 **width=416 height= 416**

Sau khi câu lệnh xong save lại file.

Bước 6: Sử dụng đoạn code như **Phụ lục hình 21** chia 80% train.txt và 20% file val.txt

```
%cd /content/gdrive/My\ Drive/darknet/
import glob2
import math
import os
import numpy as np

all_files = []

for ext in ["*.jpg"]:
    images = glob2.glob(os.path.join('data/data/', ext))
    all_files += images
    print(len(all_files))
for i in range(0, 3):
    rand_idx = np.random.randint(0, len(all_files), int(0.20*len(all_files)))
    # Create train.txt
    with open("train.txt", "w") as f:
        for idx in np.arange(len(all_files)):
            if idx not in rand_idx:
                f.write(all_files[idx]+'\n')

    # Create valid.txt
    with open("val.txt", "w") as f:
        for idx in np.arange(len(all_files)):
            if idx in rand_idx:
                f.write(all_files[idx]+'\n')
```

Kết quả sau khi chia train.txt và val.txt **Phụ lục hình 22.**

Phụ lục hình 21 Chia train.txt và val.txt

 train.txt	tôi	17 thg 8, 2022 tôi	132 KB
 val.txt	tôi	17 thg 8, 2022 tôi	42 KB

Phụ lục hình 22 Kết quả chia train.txt và val.txt

Bước 7: Tạo file yolo.data như Phụ lục hình 23

Trong file yolo.data được thiết lập trước đào tạo chứa các đường dẫn chỉ đến các file cần thiết cho quá trình đào tạo. Theo **Phụ lục hình 23**, khi chạy mô hình đào tạo thư mục

backup sẽ được tạo ra và chứa các mô hình được đào tạo sau mỗi 1000 vòng đào tạo. tương tự train=train.txt chỉ dẫn đường kết nối đến file train.txt.

```
[ ] # Step 7. Tạo file yolo.data
%cd /content/gdrive/My\ Drive/darknet
!mkdir backup
!echo classes=2 > yolo.data
!echo train=train.txt >> yolo.data
!echo valid=val.txt >> yolo.data
!echo names=yolo.names >> yolo.data
!echo backup=backup >> yolo.data
```

Phụ lục hình 23 Tạo file yolo.data

Bước 8: Chỉnh sửa file Makefile:

Vào Darknet → mở **Makefile**, thiết lập như **Phụ lục hình 24**

```
Makefile ×
1 GPU=1
2 CUDNN=1
3 CUDNN_HALF=0
4 OPENCV=1
5 AVX=0
6 OPENMP=0
7 LIBSO=0
8 ZED_CAMERA=0
9 ZED_CAMERA_V2_8=0
10
11 # set GPU=1 and CUDNN=1 to speedup on GPU
```

Dòng 1 thay đổi **GPU=1**

Dòng 2 thay đổi **CUDNN=1**

Dòng 4 thay đổi **OPENCV=1**

Phụ lục hình 24 Thiết lập Makefile

Bước 9: Biên dịch mã nguồn Darknet theo như **Phụ lục hình 25**.

```
[ ] # Step 8. Make darknet
%cd /content/gdrive/My\ Drive/darknet
!rm darknet
!make
```

Phụ lục hình 25 Biên dịch mã nguồn

Bước 10: Tải file Pretrain Weights của Darknet như **Phụ lục hình 26**.

```
[ ] # Step 9. Download pretrain weight
%cd /content/gdrive/My\ Drive/darknet
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137
```

Phụ lục hình 26 Cài đặt mạng cơ sở

Bước 11: Đào tạo mô hình

Theo cú pháp train

```
!./darknet detector train [data config file] [model config file] [pretrain-model-weight]  
- dont_show > [file log saved] -map
```

Sau khi train đủ số vòng ta thu được các file [file log saved], biểu đồ train theo dõi mAP-loss và các file mô hình dùng để thử nghiệm ở trong thư mục backup.

```
▶ # Step 10. Train  
%cd /content/gdrive/My\ Drive/darknet  
!./darknet detector train yolo.data cfg/yolov4-custom.cfg backup/yolov4-custom_5000.weights -dont_show > testtrain3-yolov4.log -map
```

Phụ lục hình 27 Chạy đào tạo mô hình YoLo v4

Trong quá trình đào tạo nếu gặp sự cố overfit, tiến hành giảm số lô batch-size.

❖ Thực nghiệm với mô hình thu được

Chạy dòng lệnh như Phụ lục hình 28 để thực nghiệm với hình ảnh.

```
[6] %cd /content/gdrive/My\ Drive/darknet  
!./darknet detector test yolo.data cfg/yolov4-custom.cfg backup/q1yolov4-custom_best.weights leo.png -dont_show
```

Phụ lục hình 28 Thực nghiệm với hình ảnh

Chạy dòng lệnh như Phụ lục hình 29 để thực nghiệm với video.

```
[ ] %cd /content/gdrive/My\ Drive/darknet  
!./darknet detector demo yolo.data cfg/yolov4-custom.cfg backup/q1yolov4-custom_last.weights /content/gdrive/MyDrive/darknet/1s.mp4 -i 0 -out_filename kqvideo2.mp4 -dont_show
```

Phụ lục hình 29 Thực nghiệm với video

7.3.2. Cách Đào tạo mô hình YOLO v7

❖ Chuẩn bị dữ liệu

Tạo thư mục tên Images chứa 4882 ảnh và thư mục tên Labels chứa 4882 nhãn. YOLOv7 ảnh và nhãn không nằm cùng thư mục.

❖ Huân luyện mô hình

Bước 1: Kết nối Drive với Google Colab như Phụ lục hình 13.

Bước 2: Tạo môi trường cho YoLo v7, bằng cách thực hiện như Phụ lục hình 30.

```
▶ %cd /content/gdrive/MyDrive/  
!git clone https://github.com/WongKinYiu/yolov7.git
```

Phụ lục hình 30 Tạo môi trường Yolo v7

Bước 3: Trên Drive tạo thư mục data và upload tập dữ liệu đã chuẩn bị vào thư mục data (nếu tập .zip hoặc .tar thì giải nén ra), sau đó chạy lệnh như Phụ lục hình 31 để chia 80% train và 20% val.

```

# 1.chạy lệnh chia train test
import os
import glob
import random
import shutil

raw_data_path = "./images"
train_data_path = "./data/train"
test_data_path = "./data/test"

if not os.path.exists("./data/"):
    os.mkdir("./data")

try:
    shutil.rmtree(train_data_path)
    shutil.rmtree(test_data_path)
except:
    pass

os.mkdir(train_data_path)
os.mkdir(test_data_path)

total_files = glob.glob(raw_data_path + "/*.jpg")
print("Sample file = ", total_files[0])
print("Total file = ", len(total_files))

indices = list(range(len(total_files)))
train_indices = random.sample(indices, k = int(len(total_files)*0.8))
print("Total train file = ", len(train_indices))

for i in indices:

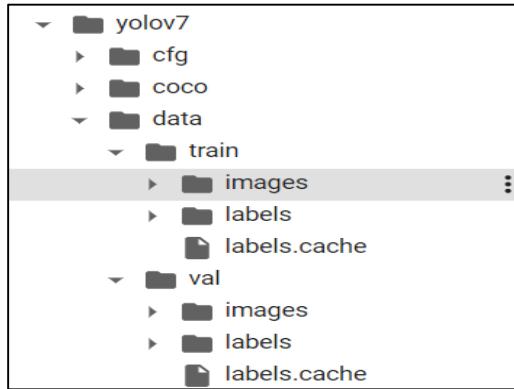
    if (i in train_indices):
        # Copy to train
        shutil.copy(total_files[i], train_data_path + total_files[i].replace(raw_data_path,""))
        shutil.copy(total_files[i].replace("images","labels").replace(".jpg",".txt"),
                    train_data_path + total_files[i].replace(raw_data_path, "").replace(".jpg",".txt"))

    else:
        # Copy to test
        shutil.copy(total_files[i], test_data_path + total_files[i].replace(raw_data_path, ""))
        shutil.copy(total_files[i].replace("images","labels").replace(".jpg", ".txt"),
                    test_data_path + total_files[i].replace(raw_data_path, "").repl

```

Phụ lục hình 31 Chia train và val

Sau khi chạy đoạn kết trên, kết quả thu được mô tả theo cấu trúc thư mục như **Phụ lục hình 32**



Phụ lục hình 32 Cấu trúc thư mục train và val

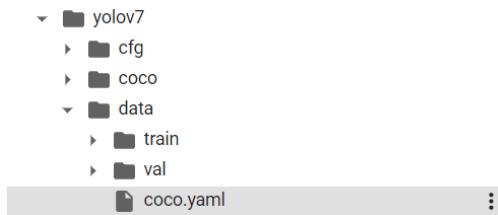
Bước 4: Tải COCO starting checkpoint

```
%cd /content/gdrive/MyDrive/yolov7
```

```
!pip install -r requirements.txt
```

```
!wget "https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7.p7"
```

Bước 5: Chỉnh file coco.yaml



Phụ lục hình 33 file coco.yaml

Tìm file coco.yaml như **Phụ lục hình 33** trong thư mục yolov7/data/ sau đó đổi tên file coco.yaml hoặc nhân bản file coco.yaml thành custom_data.yaml

Tiếp theo mở file custom_data.yaml và cấu hình như **Phụ lục hình 34**.

```
custom_data.yaml ×  
1 train: ./data/train  
2 val: ./data/val  
3  
4 # number of classes  
5 nc: 2  
6  
7 # class names  
8 names: [ 'climbing', 'normal' ]
```

Phụ lục hình 34: cấu hình custom_data.yaml

Trong đó

Dòng 1 **train**: đưa đường dẫn đến thư mục train chứa các images và labels của train

Dòng 2 **val**: đưa đường dẫn đến thư mục val chứa các images và labels của val

Dòng 5 **nc**: số class tương ứng với số lớp nhãn cần đào tạo ‘climbing và normal’

Dòng 8 **names**: chứa tên nhãn

Sau đó save lại file

Bước 6: Chỉnh sửa các tham số trong file cfg/yolov7.yaml

Vào yolov7 → cfg → training/yolov7-custom.yaml

```
yolov7-custom.yaml ×  
1 # parameters  
2 nc: 2 # number of classes  
3 depth_multiple: 1.0 # model depth multiple  
4 width_multiple: 1.0 # layer channel multiple  
5  
6 # anchors  
7 anchors:  
8 - [12,16, 19,36, 40,28] # P3/8
```

Phụ lục hình 35 Chỉnh sửa file yolov7-custom.yaml

Chỉnh dòng 2 bằng **nc**: 2, tương ứng với số lớp nhãn sau đó lưu lại và bắt đầu đào tạo.

Bước 7: Đào tạo mô hình

Cú pháp

train.py --batch-size <số lượng batch-size đưa vào> --epoch <số lượng vòng lặp train> --image <kích thước đầu vào mặc định là 640 640> --cfg <đường dẫn đến file config> --data <đường dẫn đến file custom_data.yaml> --weights “file weight train” --device 0

```
[ ] # run this cell to begin training  
%cd /content/gdrive/MyDrive/yolov7  
!python train.py --batch-size 16 --epochs 250 --img 640 640 --cfg cfg/training/yolov7-custom.yaml --data data/custom_data.yaml --weights 'yolov7.pt' --device 0
```

Phụ lục hình 36 Lệnh đào tạo YOLO v7

Sau khi train xong tất cả các file thu được, được lưu ở thư mục yolov7/runs/trains bao gồm các kết quả đánh giá mô hình như biểu đồ đánh giá hàm mất mát, biểu đồ loss.

❖ Thực nghiệm với mô hình thu được

Chạy dòng lệnh như **Phụ lục hình 37** để thực nghiệm mô hình với hình ảnh

```
!python detect.py --weights <file weight> --img 640 --conf 0.5 --source <file ảnh>
```

```
%cd /content/gdrive/MyDrive/yolov7  
!python detect.py --weights /content/gdrive/MyDrive/yolov7/runs/train/exp7_endv1/weights/last.pt --img 640 --conf 0.5 --source /content/gdrive/MyDrive/yolov7/video/916f0eb1c5a11cff45b0.jpg
```

Phụ lục hình 37 Thực nghiệm mô hình YOLO v7 với hình ảnh

Tương tự thực nghiệm mô hình với video như thực nghiệm với hình ảnh, thay phần source <file ảnh> thành source <file video>

7.3.3. Cách đào tạo mô hình SSD

❖ Chuẩn bị dữ liệu

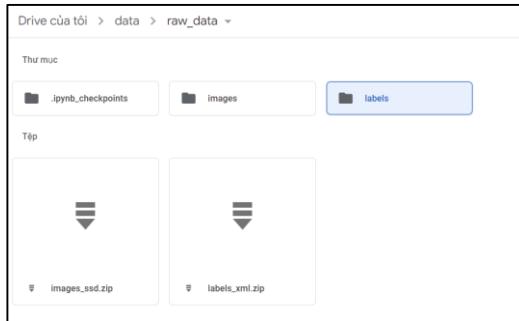
Chuẩn bị tập dữ liệu ảnh và nhãn (4882 ảnh và 4882 nhãn). Các file ảnh và nhãn không nằm cùng thư mục, thư mục images chứa 4882 ảnh và thư mục labels chứa 4882 nhãn. Tất cả các nhãn được sử dụng cho SSD Mobilenet là dạng PASCALVOC đã được chuyển đổi từ YOLO sang PASCAL/VOC có phần mở rộng ‘.xml’.

Tạo thư mục draw_data chứa 2 thư mục chứa các file ảnh images và nhãn labels

❖ Đào tạo mô hình

Bước 1: Kết nối Google Colaboratory với Drive như YOLO v7

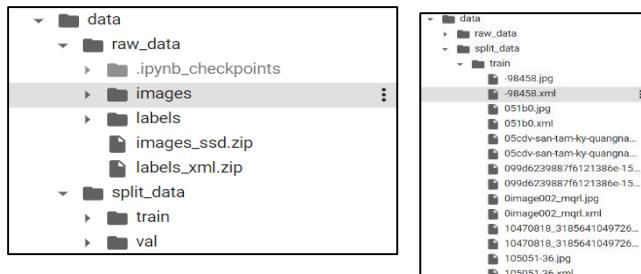
Tạo thư mục data/raw_data trong drive sau đó upload tập dữ liệu đã chuẩn bị bao gồm images và labels.



Phụ lục hình 38 tập dữ chuẩn bị chứa trong raw_data

Bước 2: Sử dụng đoạn code như **Phụ lục hình 40** để tạo thư mục split_data và chia train, test.

Đầu tiên đoạn code tự tạo thư mục split_data trong thư mục data. Sau đó tạo thư mục split_data/train và split_data/test. Tiếp theo đoạn code tiến hành chia 80% train và 20% test kết quả sau khi chạy đoạn code được minh họa như **Phụ lục hình 39**.



Phụ lục hình 39 kết quả chia train và test

```

[ ] # 1. Tạo thư mục data/split_data và chạy lệnh chia train test
import os
import glob
import random
import shutil

raw_data_path = "/content/gdrive/MyDrive/data/raw_data/images"
train_data_path = "/content/gdrive/MyDrive/data/split_data/train"
test_data_path = "/content/gdrive/MyDrive/data/split_data/test"

if not os.path.exists("/content/gdrive/MyDrive/data/split_data/"):
    os.mkdir("/content/gdrive/MyDrive/data/split_data/")

try:
    shutil.rmtree(train_data_path)
    shutil.rmtree(test_data_path)
except:
    pass

os.mkdir(train_data_path)
os.mkdir(test_data_path)

total_files = glob.glob(raw_data_path + "/*.jpg")
print("Sample file = ", total_files[0])
print("Total file = ", len(total_files))

❸ indices = list(range(len(total_files)))
train_indices = random.sample(indices, k = int(len(total_files)*0.9))
print("Total train file = ", len(train_indices))

for i in indices:

    if (i in train_indices):
        # Copy to train
        shutil.copy(total_files[i], train_data_path + total_files[i].replace(raw_data_path,""))
        shutil.copy(total_files[i].replace("images","labels").replace(".jpg",".xml"),
                    train_data_path + total_files[i].replace(raw_data_path, "").replace(".jpg",".xml"))
    else:
        # Copy to test
        shutil.copy(total_files[i], test_data_path + total_files[i].replace(raw_data_path, ""))
        shutil.copy(total_files[i].replace("images", "labels").replace(".jpg", ".xml"),
                    test_data_path + total_files[i].replace(raw_data_path, "").replace(".jpg", ".xml"))

❹ Total file = 4882

```

Phụ lục hình 40 Code chia tập train và test

Bước 3: Chuyển các tập dữ liệu train và test thành tập dữ liệu có phần mở rộng ‘.csv’

```

[ ] # 3.Convert XML label sang CSV. File CSV lưu trong thư mục split_data
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET

def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df

```

Phụ lục hình 41 Convert train test thành ‘.csv’ (a)

```

for directory in ['train', 'test']:
    image_path = os.path.join('/content/gdrive/MyDrive/data/split_data/{}'.format(directory))
    xml_df = xml_to_csv(image_path)
    xml_df.to_csv('/content/gdrive/MyDrive/data/split_data/{}_labels.csv'.format(directory), index=None)
    print('Successfully converted xml to csv.')

```

Phụ lục hình 42 Convert train test thành ‘.csv’ (b)

Sau khi chạy đoạn code trên **Phụ lục hình 41** cộng **Phụ lục hình 42** thành, kết quả 2 tệp train_labels.csv (**Phụ lục hình 43**) và val_labels.csv (**Phụ lục hình 44**) ở trong thư mục data/split_data được tạo.

1 to 100 of 9355 entries <input type="button" value="Filter"/> <input type="button" value="Copy"/>							
filename	width	height	class	xmin	ymin	xmax	ymax
273278.cbad00002faa3850.jpg	4288	2848	normal	134	1488	773	2848
273278.cbad00002faa3850.jpg	4288	2848	normal	745	1167	1588	2848
273278.cbad00002faa3850.jpg	4288	2848	normal	1484	1348	2091	2827
273278.cbad00002faa3850.jpg	4288	2848	normal	1984	1117	2559	2848
273278.cbad00002faa3850.jpg	4288	2848	normal	2392	1131	2921	2277
273278.cbad00002faa3850.jpg	4288	2848	normal	2735	1331	3367	2848
273278.cbad00002faa3850.jpg	4288	2848	normal	3070	1012	3698	2848
273278.cbad00002faa3850.jpg	4288	2848	normal	3617	1149	4285	2848
273278.c895c0003d827c27.jpg	750	500	normal	2	144	67	215
273278.c895c0003d827c27.jpg	750	500	normal	45	162	162	337
273278.c895c0003d827c27.jpg	750	500	normal	112	98	200	187

Phụ lục hình 43 File train_labels.csv

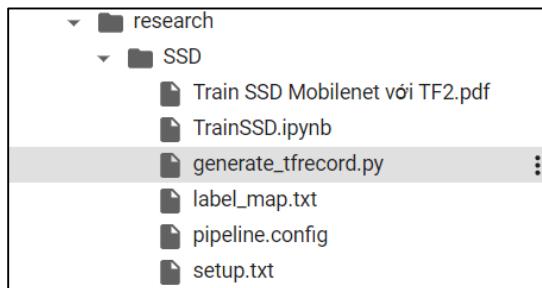
1 to 10 of 2427 entries <input type="button" value="Filter"/> <input type="button" value="Copy"/>							
filename	width	height	class	xmin	ymin	xmax	ymax
273278.c9bc7000108b5ec2.jpg	1200	485	normal	32	96	380	451
273278.c9bc7000108b5ec2.jpg	1200	485	normal	328	102	497	444
273278.c9bc7000108b5ec2.jpg	1200	485	normal	464	70	661	441
273278.c9bc7000108b5ec2.jpg	1200	485	normal	661	75	1001	435
273278.c9bc7000108b5ec2.jpg	1200	485	normal	957	33	1200	424
273278.b6c06000b7723eca.jpg	780	519	normal	89	161	181	477
273278.b6c06000b7723eca.jpg	780	519	normal	180	150	266	470
273278.b6c06000b7723eca.jpg	780	519	normal	250	142	293	210
273278.b6c06000b7723eca.jpg	780	519	normal	283	159	367	472
273278.b6c06000b7723eca.jpg	780	519	normal	333	129	384	195

Show 10 per page

1 2 10 100 200 240 243

Phụ lục hình 44 File val_labels.csv

Nếu chuyển XML sang CSV bị lỗi chỉnh lại file generate_tfrecord.py đã tải xuống từ github. Tìm file generate_tfrecord.py (**Phụ lục hình 45**) để chỉnh sửa.



Phụ lục hình 45 generate_tfrecord.py

```
30 def class_text_to_int(row_label):
31     VOC_LABELS = [
32         'none': (0, 'Background'),
33         'climbing': (1, 'climbing'),
34         'normal': (2, 'normal')]
35     return VOC_LABELS[row_label][0]
36     # if row_label == 'mobile':
37     #     return 1
38     # else:
39     #     None
40
41
```

Phụ lục hình 46 chỉnh sửa VOC_LABELS trong file generate_tfrecord.py

Chỉnh lại các lớp nhãn cho phù hợp ở trong VOC_LABELS, trong ddos labels ‘Background’ có id=0 là mặc định, chỉ chỉnh sửa lại các nhãn cho phù hợp với đề tài.

Bước 4: Tải generate_tfrecord.py để chuyển file ‘.csv’ sang dạng TFRecord (**Phụ lục hình 47**)

```
[ ] # 8. CSV to TFRecord. With help from quychdi git hub
import os
%cd /content/gdrive/MyDrive/models/research
!git clone https://github.com/QuychDi/SSD.git
!cp SSD/generate_tfrecord.py .
!cp SSD/label_map.txt /content/gdrive/MyDrive/data
```

Phụ lục hình 47 cài đặt file generate_tfrecord.py

Sau khi cài đặt thành công chạy đoạn code sau (**Phụ lục hình 48**) để chuyển ‘csv’ thành ‘record’

```
[ ] # 8. CSV to TFRecord. With help from quychdi git hub
import os
%cd /content/gdrive/MyDrive/models/research
!git clone https://github.com/QuychDi/SSD.git
!cp SSD/generate_tfrecord.py .
!cp SSD/label_map.txt /content/gdrive/MyDrive/data

if not os.path.exists("/content/gdrive/MyDrive/data/tfrecord_data/"):
    os.mkdir("/content/gdrive/MyDrive/data/tfrecord_data/")

!python generate_tfrecord.py --image_dir=/content/gdrive/MyDrive/data/split_data/train \
--csv_input=/content/gdrive/MyDrive/data/split_data/train_labels.csv \
--output_path=/content/gdrive/MyDrive/data/tfrecord_data/train.record \
!python generate_tfrecord.py --image_dir=/content/gdrive/MyDrive/data/split_data/test --csv_input=/content/gdrive/MyDrive/data/split_data/test_labels.csv --
```

Phụ lục hình 48 Chuyển ‘csv’ thành ‘record’

Sau khi chạy đoạn phía trên thành công (**Phụ lục hình 48**) trong thư mục data sinh ra tệp label_map.txt chứa tên nhãn của đối tượng (**Phụ lục hình 49**) và thư mục tfrecord_data sinh ra chứa 2 tệp train.record và val.record

```
label_map.txt x ...
1 item { id:1, name:'climbing'}
2 item { id:2, name:'normal'}
```

Phụ lục hình 49 file label_map.txt

Bước 4: Cài đặt môi trường Tensorflow Object Detection API (**Phụ lục hình 50**) vào thư mục model

```
%cd /content/gdrive/MyDrive/
!git clone https://github.com/tensorflow/models.git

Cloning into 'models'...
remote: Enumerating objects: 76658, done.
remote: Counting objects: 100% (479/479), done.
remote: Compressing objects: 100% (262/262), done.
remote: Total 76658 (delta 238), reused 425 (delta 213), pack-reused 76179
Receiving objects: 100% (76658/76658), 596.83 MiB | 16.56 MiB/s, done.
Resolving deltas: 100% (54325/54325), done.
Checking out files: 100% (3146/3146), done.
```

Phụ lục hình 50 cài đặt model

Bước 5: Biên dịch các giao thức (**Phụ lục hình 51**)

```
# 5. Compile the protos -
# Protocol Buffers (ProtocolBuf) is a free and open-source cross-platform data format used to serialize structured data.
%cd /content/gdrive/MyDrive/models/research
!protoc object_detection/protos/*.proto --python_out=.

Cloning into 'models'...
remote: Enumerating objects: 76658, done.
remote: Counting objects: 100% (479/479), done.
remote: Compressing objects: 100% (262/262), done.
remote: Total 76658 (delta 238), reused 425 (delta 213), pack-reused 76179
Receiving objects: 100% (76658/76658), 596.83 MiB | 16.56 MiB/s, done.
Resolving deltas: 100% (54325/54325), done.
Checking out files: 100% (3146/3146), done.
```

Phụ lục hình 51 Biên dịch

Bước 6: Cài đặt API (**Phụ lục hình 52**)

```
# 6. Cài đặt API
%cd /content/gdrive/MyDrive/models/research
!cp object_detection/packages/tf2/setup.py .
!python -m pip install .
```

Phụ lục hình 52 Cài đặt API

Nếu cài đặt thành công kết quả như **Phụ lục hình 53**

```
Uninstalling cloudpickle-1.5.0...
Successfully uninstalled cloudpickle-1.5.0
Successfully installed apache-beam-2.43.0 avro-python3-1.10.2 cloudpickle-2.2.0 colorama-0.4.6 dill-0.3.1.1 docopt-0.6.2 fastavro-1.2.1
```

Phụ lục hình 53 Cài đặt API thành công

Bước 7: Test API (Phụ lục hình 54)

```
# 7. Test API
%cd /content/gdrive/MyDrive/models/research
!python object_detection/builders/model_builder_tf2_test.py
```

Phụ lục hình 54 Test API

Bước 8: Tải file cấu MobileNet v2 cho SSD.

Thư mục pretrained được sinh ra chứa các file config và checkpoint sau khi chạy đoạn code phía dưới (**Phụ lục hình 55**)

```
# 9. Download pretrain and config
import os
if not os.path.exists("./content/gdrive/MyDrive/pretrained"):
    os.mkdir("./content/gdrive/MyDrive/pretrained")

%cd ./content/gdrive/MyDrive/pretrained
!wget http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz
# Unzip
!tar -xvf ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz

# Config pipeline.config, label_map.txt if needed
```

Phụ lục hình 55 Tải MobileNet v2

Bước 9: Chính sửa cấu hình file pipeline.config trong model vừa tải về

Mở file pipeline.config và cấu hình như sau:

Dòng 3 thiết lập num_classes: 2 (số nhãn ‘climbing’ và ‘normal’)

```
pipeline.config x
1 model {
2   ssd {
3     num_classes: 2
```

Dòng 135 batch_size: 4

```
134 train_config {
135   batch_size: 4
136   data_augmentation_options {
137     random_horizontal_flip {
138       }
139 }
```

Dòng 165: **fine_tune_checkpoint**: gắn đến đường dẫn checkpoint của pretrained

```
165 fine_tune_checkpoint: "/content/gdrive/MyDrive/pretrained/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/ckpt-0"
166 num_steps: 50000
167 startup_delay_steps: 0.0
168 replicas_to_aggregate: 8
169 max_number_of_boxes: 100
170 unpad_groundtruth_tensors: false
171 fine_tune_checkpoint_type: "detection"
172 fine_tune_checkpoint_version: V2
173 }
174 train_input_reader {
175   label_map_path: "/content/gdrive/MyDrive/data/label_map.txt"
176   tf_record_input_reader {
177     input_path: "/content/gdrive/MyDrive/data/tfrecord_data/train.record"
178   }
179 }
180 eval_config {
181   metrics_set: "coco_detection_metrics"
182   use_moving_averages: false
183 }
184 eval_input_reader {
185   label_map_path: "/content/gdrive/MyDrive/data/label_map.txt"
186   shuffle: false
187   num_epochs: 1
188   tf_record_input_reader {
189     input_path: "/content/gdrive/MyDrive/data/tfrecord_data/val.record"
190   }
191 }
192
```

Phụ lục hình 56 File pipeline.config

Dòng 166: **num_steps**: số lượng vòng train (mặc định 50000 vòng)

Dòng 171: **fine_tune_checkpoint_type: detection**

Dòng 175, 185: **label_map_path**: gắn đường dẫn đến file label_map.txt (file nhãn)

Dòng 177: **input_path**: gắn đường dẫn đến tập file train.record

Dòng 189: **input_path**: gắn đường dẫn đến tập file val.record

Save lại file pipeline.config

Bước 10: Đào tạo mô hình

Sử dụng lệnh đào tạo mô hình như Phụ lục hình 57

```
▶ # Train
%cd /content/gdrive/MyDrive/models/research
%cp /content/gdrive/MyDrive/models/research/object_detection/model_main_tf2.py .

#train
!python model_main_tf2.py \
-pipeline_config_path=/content/gdrive/MyDrive/pretrained/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/pipeline.config \
--model_dir=/content/gdrive/MyDrive/output_model-V5 --alsologtosterr \
--num_train_steps=60000
```

Phụ lục hình 57 Lệnh đào tạo mô hình SSD

Sử dụng **model_main_tf2** để đào tạo mô hình

Trong đó:

--pipeline_config_path=<đường dẫn đến file config đã được chỉnh sửa cấu hình lại>

--model_dir = <đường dẫn lưu các file được sinh ra trong quá trình đào tạo>
--num_train_steps = <số bước vòng train>

Sau khi đào tạo xong mô hình, trong thư mục output_model-V5 (--model_dir) chưa có các checkpoint trong quá trình đào tạo.

Bước 11: Đánh giá model và lấy kết quả, chạy dòng lệnh như **Phụ lục hình 58**.

```
▶ %cd /content/gdrive/MyDrive/models/research  
%cp /content/gdrive/MyDrive/models/research/object_detection/model_main_tf2.py .  
  
#train  
!python model_main_tf2.py \  
--model_dir=/content/gdrive/MyDrive/output_model-V5 \  
--pipeline_config_path=/content/gdrive/MyDrive/pretrained/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/pipeline.config \  
--checkpoint_dir=/content/gdrive/MyDrive/output_model-V5
```

Phụ lục hình 58 Đánh giá

--model_dir = <đường dẫn đến thư mục output_model được sinh ra ở bước train>.
--pipeline_config_path = <đường dẫn đến file config>.
--checkpoint_dir = <đường dẫn đến thư mục output_model được sinh ra chứa các file checkpoint>.

Sau khi thực hiện lệnh đánh giá mô hình **này Phụ lục hình 58**, có thể đánh giá mô hình trên Tensorflow bằng cách sử dụng lệnh như **Phụ lục hình 59**.

```
%cd /content/gdrive/MyDrive  
# !kill 1341  
%load_ext tensorboard  
%tensorboard --logdir 'output_model-V1'
```

Phụ lục hình 59 đánh giá mô hình trên tensorflow

%load_ext tensorboard
%tensorboard --logdir ‘<thư mục output_model>’

Bước 13: Xuất model và lấy model để kiểm thử như **Phụ lục hình 60**.

```
▶ # Export model  
%cd /content/gdrive/MyDrive/models/research  
%cp /content/gdrive/MyDrive/models/research/object_detection/exporter_main_v2.py .  
  
!python exporter_main_v2.py \  
--trained_checkpoint_dir=/content/gdrive/MyDrive/output_model \  
--pipeline_config_path=/content/gdrive/MyDrive/pretrained/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/pipeline.config \  
--output_directory=/content/gdrive/MyDrive/export_model
```

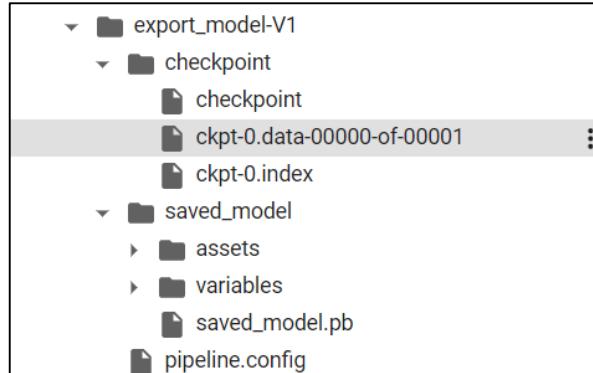
Phụ lục hình 60 Xuất Model SSD

Xuất model file **saved_model.pb** đã được train trong thư mục output_model.

--trained_checkpoint_dir = <đường dẫn đến thư mục output_model>

--pipeline_config_path = <đường dẫn đến file config>

--checkpoint_dir = <đường dẫn thư mục để sinh ra thư mục export_model chứa file save_mode.pb>



Phụ lục hình 61 Kết quả saved_model.pb xuất thành công

Trong quá trình export model nếu báo lỗi thì chạy đoạn code sau:

```
[ ] !apt install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11.2
```

❖ Thực nghiệm

Thực nghiệm với model **saved_model.pb** được xuất ra với hình ảnh. Load model saved_model.pb và import các thư viện nhận dạng cần thiết. sử dụng đoạn code phía dưới để chạy thực nghiệm (**Phụ lục hình 62 Phụ lục hình 63 & Phụ lục hình 64**).

```

import io
import os
import scipy.misc
import numpy as np
import six
import time
import glob
from IPython.display import display

from six import BytesIO

import matplotlib
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw, ImageFont

import tensorflow as tf
from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util

#Load model
tf.keras.backend.clear_session()
model = tf.saved_model.load("/content/gdrive/MyDrive/export_model-v5/saved_model")

```

Phụ lục hình 62 Load model saved_model.pb và import các thư viện

Chạy các hàm inference như **Phụ lục hình 63**

```

# Các hàm inference

import cv2
def run_inference_for_single_image(model, image):

    image = np.asarray(image)
    input_tensor = tf.convert_to_tensor(image)
    input_tensor = input_tensor[tf.newaxis,...]

    model_fn = model.signatures['serving_default']
    output_dict = model_fn(input_tensor)

    num_detections = int(output_dict.pop('num_detections'))
    output_dict = {key:value[0, :num_detections].numpy()
                  for key,value in output_dict.items()}
    output_dict['num_detections'] = num_detections
    output_dict['detection_classes'] = output_dict['detection_classes'].astype(np.int64)

    if 'detection_masks' in output_dict:
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
            output_dict['detection_masks'], output_dict['detection_boxes'],
            image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5,
                                           tf.uint8)
        output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()

    return output_dict

def load_image_into_numpy_array(path):
    img_data = tf.io.gfile.GFile(path, 'rb').read()
    image = Image.open(BytesIO(img_data))
    (im_width, im_height) = image.size
    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)

```

Phụ lục hình 63 chạy hàm inference

```

%cd /content/gdrive/MyDrive/models/research
category_index = label_map_util.create_category_index_from_labelmap("/content/gdrive/MyDrive/data/label_map.txt", use_display_name=True)

image_path = '/content/gdrive/MyDrive/detect/h5.jpg'
image_np = load_image_into_numpy_array(image_path)
print("Done load image ")
image_np = cv2.resize(image_np, dsize=None, fx=1, fy=1)
output_dict = run_inference_for_single_image(model, image_np)
print("Done inference")
vis_util.visualize_boxes_and_labels_on_image_array(
    image_np,
    output_dict['detection_boxes'],
    output_dict['detection_classes'],
    output_dict['detection_scores'],
    category_index,
    instance_masks=output_dict.get('detection_masks_reframed', None),
    use_normalized_coordinates=True,
    line_thickness=20)
print("Done draw on image ")
display(Image.fromarray(image_np))

```

Phụ lục hình 64 Chạy hàm nhận dạng

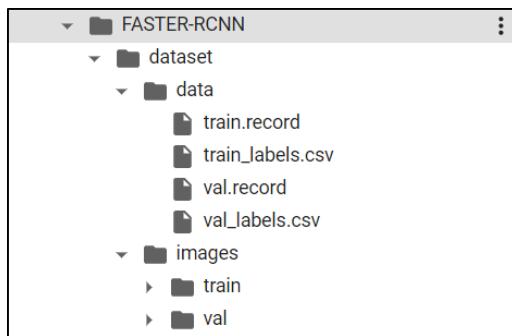
7.3.4. Cách đào tạo mô hình với Faster-RCNN

- ❖ Chuẩn bị dữ liệu

Các bước chuẩn bị dữ liệu như chia train, test, convert (chuyển xml sang csv, từ csv sang record) được thực hiện tương tự như mô hình SSD.

Cấu trúc thư mục dataset (thư mục chứa dữ liệu) được tạo trên Drive có cấu trúc như

Phụ lục hình 65



Phụ lục hình 65 Cấu trúc thư mục dataset

Thư mục images: chứa 80% ảnh và nhãn chung trong thư mục train. 20% còn lại chứa trong thư mục val bao gồm ảnh và nhãn. Chạy đoạn code chuyển xml sang csv, khi có được csv thì chạy tiếp đoạn code chuyển csv sang ‘record’, các đoạn code hỗ trợ chuyển đổi đã được nêu ở bước đào tạo mô hình SSD. Các thư mục sinh ra được lưu ở thư mục dataset/data.

Trong quá trình thực hiện chuyển đổi csv thành dạng record được thực hiện tương tự như SSD gặp lỗi, thực hiện các bước dưới:

Tải nguồn code theo đường dẫn sau để hỗ trợ việc chuyển tập dữ liệu từ csv thành dạng record https://github.com/QuychDi/FASTER_RCNN.git

Sau khi tải xong mở file generate_tfrecord.py lên thiếp lập như hình **Phụ lục hình 66**. Lưu ý không nên return về 0.

```
29 # TO-DO replace this with label map
30 def class_text_to_int(row_label):
31     if row_label == 'climbing':
32         return 1
33     else:
34         return 2
35
```

Phụ lục hình 66 chỉnh sửa file chuyển csv thành record

Sau khi hoàn thành sử dụng dòng lệnh sau để chuyển csv thành record:

```
[ ]  !python generate_tfrecord.py --csv_input=[ĐƯỜNG DẪN ĐẾN file train_labels.csv] --output_path=[đường dẫn lưu file train.record]
# Create test data:
!python generate_tfrecord.py --csv_input=[ĐƯỜNG DẪN ĐẾN file test_labels.csv] --output_path=[đường dẫn lưu file test.record]
```

Kiểm tra file record tạo thành công bằng cách chọn vào tệp train.record hoặc val.record, nếu tệp mở lên trống rỗng thì thực hiện sai, ngược lại xuất hiện trạng thái load vòng như **Phụ lục hình 67** thì file thực hiện thành công.



Phụ lục hình 67 file train.record tạo thành công

❖ Đào tạo mô hình

Bước 1: Kết nối Google Colab với Drive

Bước này thực hiện tương tự như SSD

Bước 2: Tạo file nhãn label_map.pbtxt chứa trong thư mục dataset (nhãn không được chứa id=0) (**Phụ lục hình 68**)

```
label_map.pbtxt ×
1 item { id:1,name:'climbing'}
2 item { id:2,name:'normal'}
```

Phụ lục hình 68 File nhãn Faster-RCNN

Bước 3: Cài đặt một số thư viện cần thiết

```
▶ %cd /content/gdrive/MyDrive/FASTER-RCNN
!pip install tensorflow_gpu==1.15
```

```
▶ !pip install numpy==1.19.5
```

Bước 4: Cài đặt môi trường cho Faster-RCNN như **Phụ lục hình 69**.

```
%cd /content/gdrive/MyDrive/FASTER-RCNN  
!apt-get install protobuf-compiler python-pil python-lxml python-tk  
!pip install Cython tf_slim  
!pip install -q pycocotools  
!pip install -q Cython contextlib2 pillow lxml matplotlib  
!git clone https://github.com/tensorflow/models.git  
%cd /content/gdrive/MyDrive/FASTER-RCNN/models/research  
!protoc object_detection/protos/*.proto --python_out=.  
%set_env PYTHONPATH=/content/gdrive/MyDrive/FASTER-RCNN/models/research:/content/gdrive/MyDrive/FASTER-RCNN/models/research/slim  
import os  
os.environ['PYTHONPATH'] += ':/content/gdrive/MyDrive/FASTER-RCNN/models'  
import sys  
sys.path.append('/content/gdrive/MyDrive/FASTER-RCNN/models')  
!python /content/gdrive/MyDrive/FASTER-RCNN/models/research/object_detection/builders/model_builder_test.py
```

Phụ lục hình 69 Cài đặt môi trường Faster-RCNN

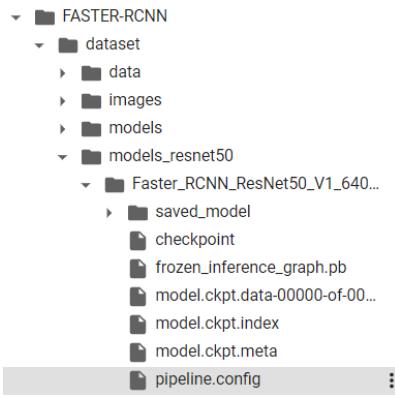
Bước 5: Tạo thư mục models_resnet50 chứa trong thư mục dataset và chạy vòng lệnh phía dưới để tải file pretrain sử dụng resnet50

```
%mkdir /content/gdrive/MyDrive/FASTER-RCNN/dataset/models_resnet50  
%cd /content/gdrive/MyDrive/FASTER-RCNN/dataset/models_resnet50  
  
import os  
import shutil  
import glob  
import urllib.request  
import tarfile  
  
MODEL = 'faster_rcnn_inception_v2_coco_2018_01_28'  
MODEL_FILE = MODEL + '.tar.gz'  
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'  
DEST_DIR = 'Faster R-CNN ResNet50 V1 640x640'  
  
#if not (os.path.exists(MODEL_FILE)):  
#    opener = urllib.request.URLopener()  
#    opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)  
  
with urllib.request.urlopen(DOWNLOAD_BASE+MODEL_FILE) as response, open(MODEL_FILE, 'wb') as out_file:  
    shutil.copyfileobj(response, out_file)  
  
tar = tarfile.open(MODEL_FILE)  
tar.extractall()  
tar.close()  
  
os.remove(MODEL_FILE)  
if (os.path.exists(DEST_DIR)):  
    shutil.rmtree(DEST_DIR)  
os.rename(MODEL, DEST_DIR)
```

▶ /content/gdrive/MyDrive/FASTER-RCNN/dataset/models_resnet50

Phụ lục hình 70 Tải mô hình ResNet50

Khi thực hiện thành công ta nhận được các file ở trong thư mục dataset/models_resnet50 như **Phụ lục hình 71**.



Phụ lục hình 71 File ResNet50 tải thành công

Bước 6: Thiết lập cài đặt cấu hình đào tạo. Mở file pipeline.config chứa trong dataset/models_resnet50/Faster_RCNN_ResNet50 như **Phụ lục hình 71**. Và tiến hành chỉnh như sau:

Tại dòng số 3: **số num_classes: 2**

3	num_classes: 2
---	-----------------------

Tại dòng 108 **fine_tune_checkpoint**: “thiết lập đường dẫn đến file checkpoint”

108	fine_tune_checkpoint: "/content/gdrive/MyDrive/FASTER-RCNN/dataset/models_resnet50/Faster_RCNN_ResNet50_V1_640x640/model.ckpt"
-----	---

Thiết lập số vòng đào tạo ở dòng 110

110	num_steps: 6000
-----	------------------------

Thiết lập **train** và **val**

<pre> 112 train_input_reader { 113 label_map_path: "/content/gdrive/MyDrive/FASTER-RCNN/dataset/label_map.pbtxt" 114 tf_record_input_reader { 115 input_path: "/content/gdrive/MyDrive/FASTER-RCNN/dataset/data/train.record" 116 } 117 }</pre>	<pre> 123 eval_input_reader { 124 label_map_path: "/content/gdrive/MyDrive/FASTER-RCNN/dataset/label_map.pbtxt" 125 shuffle: false 126 num_readers: 1 127 tf_record_input_reader { 128 input_path: "/content/gdrive/MyDrive/FASTER-RCNN/dataset/data/val.record" 129 }</pre>
---	--

Thiết lập **label_map_path**: “chứa đường dẫn đến file nhãn”

Thiết lập **input_path**: “chứa đường dẫn đến file train.record/val.record”

Bước 7: Đào tạo mô hình

Sử dụng file train.py trong thư mục FASTER_RCNN vừa tải từ trên github về để hỗ trợ đào tạo mô hình.

Tạo thư mục training để nhận kết quả sau mỗi vòng đào tạo và chạy đào tạo mô hình như lệnh phía dưới.

```
%cd /content/gdrive/MyDrive/FASTER-RCNN  
!python train.py --logtostderr \  
--train_dir=training \  
--pipeline_config_path=/content/gdrive/MyDrive/FASTER-RCNN/dataset/models_resnet50/Faster_RCNN_ResNet50_V1_640x640/pipeline.config
```

Phụ lục hình 72 Đào tạo mô hình Faster-RCNN

--train_dir= “đường dẫn đến thư mục training”

--pipeline_config_path= “đường dẫn đến file pipeline.config”

Trong quá trình đào tạo nếu có lỗi như **Phụ lục hình 73**.

```
raise ValueError('First step cannot be zero.')  
ValueError: First step cannot be zero.
```

Phụ lục hình 73 Lỗi đào tạo Faster-RCNN

chỉnh sửa trong file pipeline.config của Reset50 như **Phụ lục hình 74**.

```
88   optimizer {  
89     momentum_optimizer {  
90       learning_rate {  
91         manual_step_learning_rate {  
92           initial_learning_rate: 0.0002  
93           schedule {  
94             step: 900000  
95             learning_rate: .00002  
96           }  
97           schedule {  
98             step: 1200000  
99             learning_rate: .000002  
100           }  
101         }  
102       }  
103       momentum_optimizer_value: 0.899999976158  
104     }  
105     use_moving_average: false  
106   }
```

Phụ lục hình 74 Sửa lỗi Faster-RCNN

Sau khi đào tạo đủ số vòng ta thu được các file model và checkpoint-model trong thư mục training.

Chạy lệnh sau để đánh giá kết quả model dựa trên hàm loss

```
[ ] %load_ext tensorboard  
%tensorboard --logdir '/content/gdrive/MyDrive/FASTER-RCNN/training'
```

Bước 8: Thực nghiệm mô hình

❖ Xuất mô hình

Đầu tiên tạo thư mục inference_graph_6klan4 theo lệnh phía dưới để chứa file model_saved sau khi chạy lệnh xuất mô hình.

```
%cd /content/gdrive/MyDrive/FASTER-RCNN
```

```
!mkdir inference_graph_6klan4
```

Chạy lệnh phía dưới để xuất mô hình

```
%cd /content/gdrive/MyDrive/FASTER-RCNN/models/research/object_detection  
!python export_inference_graph.py --input_type image_tensor  
--pipeline_config_path /content/gdrive/MyDrive/FASTER-RCNN/dataset/models_resnet50/Faster_RCNN_ResNet50_V1_640x640/pipeline.config  
--trained_checkpoint_prefix /content/gdrive/MyDrive/FASTER-RCNN/training_6KLAN4/model.ckpt-6000  
--output_directory /content/gdrive/MyDrive/FASTER-RCNN/inference_graph_6klan4
```

--trained_checkpoint_dir=<chứa đường dẫn đến file pipeline.config của resnet50>

--trained_checkpoint_prefix=<chứa đường dẫn đến model.ckpt đã xuất trong thư mục training>

--output_directory=<chứa thư mục để lưu model khi chạy lệnh xuất model>

Chạy lệnh sau để thực nghiệm với mô hình

Hàm chọn file ảnh dùng để thử nghiệm với mô hình (**Phụ lục hình 75**)

```
#detect  
%cd /content/gdrive/MyDrive/FASTER-RCNN  
  
from google.colab import files  
from os import path  
  
uploaded = files.upload()  
  
for name, data in uploaded.items():  
    with open('image1.jpg', 'wb') as f:  
        f.write(data)  
        f.close()  
    print('saved file ' + name)
```

Phụ lục hình 75 Hàm tải ảnh

Tiếp theo chạy các lệnh sau để nhận dạng phát hiện với ảnh đã tải lên

```

from matplotlib.font_manager import FontManager
from tensorflow.python.ops.gen_array_ops import size
%cd /content/gdrive/MyDrive/FASTER-RCNN/models/research/object_detection
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
import tensorflow.compat.v1 as tf
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image, ImageFont

sys.path.append('..')
from object_detection.utils import ops as utils_ops
%matplotlib inline

from utils import label_map_util

from utils import visualization_utils as vis_util

PATH_TO_CKPT = '/content/gdrive/MyDrive/FASTER-RCNN/inference_graph_lan1_6k' + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('/content/gdrive/MyDrive/FASTER-RCNN/dataset', 'label_map.pbtxt')

NUM_CLASSES = 2

detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.GraphDef()
    with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

def load_image_into_numpy_array(image):
    (im_width, im_height) = image.size
    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)

# If you want to test the code with your images, just add path to the images to the TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = '/content/gdrive/MyDrive/FASTER-RCNN'
TEST_IMAGE_PATHS = [os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image1.jpg'.format(i)) for i in range(1, 4) ]

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)

```

```

def run_inference_for_single_image(image, graph):
    with graph.as_default():
        with tf.Session() as sess:
            # Get handles to input and output tensors
            ops = tf.get_default_graph().get_operations()
            all_tensor_names = {output.name for op in ops for output in op.outputs}
            tensor_dict = {}
            for key in [
                'num_detections', 'detection_boxes', 'detection_scores',
                'detection_classes', 'detection_masks'
            ]:
                tensor_name = key + ':0'
                if tensor_name in all_tensor_names:
                    tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
                        tensor_name)
            if 'detection_masks' in tensor_dict:
                # The following processing is only for single image
                detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
                detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
                # Reframe is required to translate mask from box coordinates to image coordinates and fit the image size.
                real_num_detection = tf.cast(tensor_dict['num_detections'][0], tf.int32)
                detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -1])
                detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num_detection, -1, -1])
                detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
                    detection_masks, detection_boxes, image.shape[0], image.shape[1])
                detection_masks_reframed = tf.cast(
                    tf.greater(detection_masks_reframed, 0.5), tf.uint8)
                # Follow the convention by adding back the batch dimension
                tensor_dict['detection_masks'] = tf.expand_dims(
                    detection_masks_reframed, 0)
            image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')

```

```

# Run inference
output_dict = sess.run(tensor_dict,
                       feed_dict={image_tensor: np.expand_dims(image, 0)})

# all outputs are float32 numpy arrays, so convert types as appropriate
output_dict['num_detections'] = int(output_dict['num_detections'][0])
output_dict['detection_classes'] = output_dict[
    'detection_classes'][0].astype(np.uint8)
output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
output_dict['detection_scores'] = output_dict['detection_scores'][0]
if 'detection_masks' in output_dict:
    output_dict['detection_masks'] = output_dict['detection_masks'][0]
return output_dict

for image_path in TEST_IMAGE_PATHS:
    image = Image.open(image_path)
    image_np = load_image_into_numpy_array(image)
    # Expand dimensions since the model expects images to have shape: [1, None, None, 3
    image_np_expanded = np.expand_dims(image_np, axis=0)
    # Actual detection.
    output_dict = run_inference_for_single_image(image_np, detection_graph)
    # Visualization of the results of a detection.
    vis_util.visualize_boxes_and_labels_on_image_array(
        image_np,
        output_dict['detection_boxes'],
        output_dict['detection_classes'],
        output_dict['detection_scores'],
        category_index,
        instance_masks=output_dict.get('detection_masks'),
        use_normalized_coordinates=True,
        line_thickness=8)
    # iou = output_dict['detection_scores']
    # print(iou)
    # plt.figure()
    # plt.figure(figsize=IMAGE_SIZE, dpi=200)
    plt.imshow(image_np)

```

--HÉT--