

BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN



**BÁO CÁO KHÓA LUẬN TỐT NGHIỆP
ĐỀ TÀI: ỨNG DỤNG DEEP LEARNING
XÂY DỰNG HỆ THỐNG IOT CHO
SIÊU THỊ THÔNG MINH**

GVHD: TS. ĐẶNG THỊ PHÚC

SINH VIÊN : NGUYỄN MẠNH HOÀNG

ĐÀM VĂN THẠCH

MSSV : 16027041

16061481

LỚP: DHKHMT12A

TP.HCM, tháng 6 năm 2021

INDUSTRIAL UNIVERSITY OF HO CHI MINH CITY
FACULTY OF INFORMATION TECHNOLOGY



NGUYEN MANH HOANG
DAM VAN THACH

**DEEP LEARNING APPLICATION
CONSTRUCTION IOT SYSTEM FOR
SMART SUPERMARKET**

Major: Computer science

Supervisor: Dr. DANG THI PHUC

HO CHI MINH CITY, 2020

CONTENT SUMMARY

Title: Build a system to identify people and divide customer groups by totality and gender using deep learning technology.

Summary:

- Reason for writing this issue is important for real-world scenarios. Take a look at some of the applications for which the solution to this problem can be very useful: Identifying people, classification of products.
- Human detection and identification is one of the important functions of the intelligent billboard system. Currently, with the development of IoT technology incorporating artificial intelligence, the sign recognition problem has been significantly improved in terms of time, accuracy and recognition performance. In the article, we build an IoT system that divides customers through the cameras. To solve the problem, we use deep learning techniques with the YOLOv4 model trained on a data set of videos captured through labeled dashcam. The trained model is deployed on NVIDIA Jetson Nano. Test results show that the system ensures accuracy with as well as recognition speed.

LỜI CAM ĐOAN

Chúng em xin cam đoan việc nghiên cứu đề tài được thực hiện một mình cùng với giáo viên hướng dẫn luận án và chưa được sử dụng để báo cáo cho bất kì đồ án hay môn học nào.

Chúng em xin cam đoan mọi sự giúp đỡ đã được cảm ơn, các thông tin trích dẫn luận văn đã được ghi chú đầy đủ và bảo đảm rõ nguồn gốc không xâm phạm quyền tác giả.

Thành Phố Hồ Chí Minh ngày 30, Tháng 5, Năm 2021.

TÁC GIẢ LUẬN VĂN

NGUYỄN MẠNH HOÀNG

ĐÀM VĂN THẠCH

LỜI CẢM ƠN

Trong quá trình thực hiện luận văn tốt nghiệp này chúng em được giáo viên hướng dẫn tận tình chỉ bảo, quan tâm dẫn dắt từng bước chúng em thực hiện, bên cạnh đó chúng em xin cảm ơn sâu sắc đến tất cả những cá nhân, tập thể đã tạo điều kiện giúp đỡ chúng em trong suốt quá trình thực hiện khóa luận tốt nghiệp này.

Đầu tiên chúng em xin cảm ơn ban giám hiệu trường đại học Công Nghiệp Thành Phố Hồ Chí Minh, phòng đào tạo khoa công nghệ thông tin cùng những thầy cô là người đã trau dồi cho chúng em những kiến thức kỹ năng trong suốt quá trình học tập và rèn luyện.

Đặc biệt chúng em xin trân trọng cảm ơn giảng viên TS- Đặng Thị Phúc, người đã trực tiếp chỉ bảo, tận tình hướng dẫn, cung cấp cho chúng em thật nhiều kiến thức từ cơ bản đến nâng cao, hướng dẫn khoa học cũng như động viên quan tâm, giúp đỡ từng bước nghiên cứu của chúng em trong suốt quá trình thực hiện luận văn này.

Và cuối cùng chúng em xin gửi lời cảm ơn chân thành đến tất cả mọi người đã giúp đỡ đồng hành cùng chúng em trong suốt quá trình thực hiện luận văn.

Do giới hạn kiến thức và khả năng của bản thân còn nhiều thiếu sót và hạn chế, kính mong sự chỉ dẫn và đóng góp của các thầy cô giáo để khóa luận của chúng em được hoàn thiện hơn.

Xin trân trọng cảm ơn !

TP. Hồ Chí Minh, ngày....., Tháng 6, Năm 2021

NHẬN XÉT VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN HƯỚNG DẪN

TP. Hồ Chí Minh, ngày..., Tháng 6, Năm 2021

GIÁNG VIÊN HƯỚNG DẪN

ĐĂNG THỊ PHÚC

NHẬN XÉT VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN PHẢN BIỆN

TP. Hồ Chí Minh, ngày..., Tháng 6, Năm...2021

GIÀNG VIÊN PHẢN BIỆN

MỤC LỤC

DANH MỤC BẢNG, SƠ ĐỒ, ĐỒ THI	1
DANH MỤC CÁC THUẬT NGỮ, CHỮ VIẾT TẮT	3
CÁC THƯ VIỆN PYTHON ĐƯỢC SỬ DỤNG	5
CHƯƠNG I: TỔNG QUAN	6
1. Đặt vấn đề.....	6
2. Mô tả bài toán.....	7
3. Phương pháp nghiên cứu của đề tài.....	8
4. Tóm tắt quá trình thực hiện đề tài	9
CHƯƠNG II: GIỚI THIỆU THIẾT BỊ	10
Giới thiệu về Jetson Nano:	10
Giới thiệu về thẻ nhớ:.....	11
Giới thiệu về nguồn:.....	11
Giới thiệu camera:.....	12
CHƯƠNG III : TÌM HIỂU DEEP LEARNING, YOLO VÀ NGÔN NGỮ PYTHON	13
1. Deep learning (Học sâu):.....	13
2. Giới thiệu về yolo:.....	13
YOLOv1:.....	14
YOLOv2:.....	14
YOLOv3:.....	15
YOLOv4:.....	15
Mô hình thuật toán của YOLO:.....	17
Grid System (hệ thống lưới):.....	17
CNN for YOLO Object Detection:.....	19
Loss function:	19
Classification Loss:.....	20
Localization Loss:.....	20
Confidence Loss:.....	21
Dự đoán lớp và tạo độ boun box sau quá trình huấn luyện – Inference.....	21
Hạn chế của YOLO:	22
2. Tìm hiểu về python:	23
Lịch sử phát triển:	23

CHƯƠNG IV : XÂY DỰNG ỦNG DỤNG.....	24
1. Thư viện GPU NVIDIA hỗ trợ.....	24
CUDA.....	24
cuDNN.....	24
Nhận diện đối tượng realtime hoặc qua video và hình ảnh.....	24
2. Tiến trình hoàn thành dự án:	25
Bước 1 : Chuẩn bị dữ liệu ở dạng image.	26
Bước 2 : Tiết xử lý dữ liệu (Bouding box, gán nhãn cho dữ liệu)	28
Bước 3 : Training data.	29
Kết quả sau khi train ta có một model :	32
3. Tính độ chính xác cho model:	32
Chạy Demo với tập test.....	40
4. Bắt đầu chuyển model yolov4 sang tensorRT:	42
TensorRT là gì?	42
Lợi ích của TensorRT:	44
Các lựa chọn thay thế để sử dụng TensorRT bao gồm:.....	45
Ai có thể hưởng lợi từ TensorRT?.....	45
Chuyển đổi yolov4 to tensorrt 7.2:.....	47
5. Triển khai thực tế và deploy lên web.	47
Ý tưởng:	47
Giới thiệu về NodeJS	47
Lý do nên học NodeJS là gì?.....	48
Sơ đồ hoạt động của jetson server và website node :	48
Jetson nano server:.....	49
Server socket:	50
Web server (nodejs):.....	51
6. Hoạt động thực tế:	53
CHƯƠNG V : KẾT LUẬN	57
Kết quả đạt được.	57
Ưu điểm.....	57
Nhược điểm :	57
Hướng phát triển.	57
Tài liệu tham khảo :.....	58

DANH MỤC BẢNG, SƠ ĐỒ, ĐỒ THỊ

Hình 1. Ảnh giới thiệu	6
Hình 2. MS COCO Object Detection (https://twitter.com/alexeyab84/status/1380699160748421121).....	8
Hình 3. Jetson nano	10
Hình 4. Thẻ nhớ sandisk ultra 32gb	11
Hình 5. Bộ nguồn Pi 3 / Pi 4+ 5V 2.5A.....	11
Hình 6. Camera raspberry pi v2 8mp	12
Hình 7. VGG16 (https://in4tintuc.com/vgg16-la-gi/)	15
Hình 9. Darknet-53 của yolov4 (https://paperswithcode.com/method/darknet-53)	16
Hình 8. Cấu trúc của YOLOv4 (https://devai.info/2021/02/24/series-yolo-4-tim-hieu-cau-truc-yolov1v2v3-va-v4-phan-2/).....	16
Hình 10. Hình ảnh minh họa Grid System	17
Hình 11. Hình ảnh minh họa Grid.....	18
Hình 12. Ví dụ.....	18
Hình 13. Mô hình CNN (https://pbcquoc.github.io/yolo/)	19
Hình 14. Classification Loss(https://pbcquoc.github.io/yolo/).....	20
Hình 15. IOU giữ 2 box	22
Hình 16. Sơ đồ các bước thực hiện dự án.....	25
Hình 17. Hình ảnh sau khi được thu thập (7000 hình).....	26
Hình 18. Dữ liệu IUH	26
Hình 19. Dữ liệu IUH	26
Hình 20. Dữ liệu IUH	27
Hình 21. Dữ liệu IUH	27
Hình 22. Dữ liệu Gigamall Thủ Đức	27
Hình 23. Dữ liệu trung tâm thương mại Sài Gòn CENTRE.....	27
Hình 24. Dữ liệu CCTV in Shopping Mall.....	27
Hình 25. Dữ liệu CCTV Camera.....	27
Hình 26. Dữ liệu từ camera mái âm	27
Hình 27. Dữ liệu từ camera mái âm	27
Hình 28. File classes.txt.....	28
Hình 29.Giao diện làm việc của ứng dụng labelImg.....	28
Hình 30. File lable (7000 file *.txt).....	29
Hình 31. Tạo file yolo.name	30
Hình 32. Tạo file yolo.data	30
Hình 33. Tạo file train.txt và file val.txt.....	30
Hình 34. Lệnh make file	31
Hình 35. Output training model	32
Hình 36. File model	32
Hình 37. Minh họa cách tính IOU.....	33

Hình 38. Minh họa IoU.....	34
Hình 39. True Positive	34
Hình 40. False Positive	34
Hình 41. False Negative	35
Hình 42. Biểu đồ quá trình train có output mAP	36
Hình 43. Char.png	37
Hình 44 mAP yolov4	38
Hình 45. mAP từng class	39
Hình 46. Kết quả test bằng colab	40
Hình 47. Hình detect IUH.....	40
Hình 48. Hình detect IUH.....	40
Hình 49. Hình detect IUH.....	41
Hình 50. Hình detect Gigamall	41
Hình 51. TensorRT là một công cụ tối ưu hóa suy luận mạng nơ-ron và công cụ thời gian chạy hiệu suất cao để triển khai sản xuất.....	43
Hình 52. Model tensorrt.....	47
Hình 53. Sơ đồ hoạt động của hệ thống trên jetson nano.....	48
Hình 54. Sơ đồ jetson nano server	49
Hình 55. Sơ đồ server socket python.....	50
Hình 56. Sơ đồ web server nodejs.....	51
Hình 57. Mô hình hoạt động tổng quát của hệ thống	52
Hình 58. Jetson nano thực tế	53
Hình 59. Bộ video quảng cáo.....	53
Hình 60. Camera Raspberry Pi V2.....	53
Hình 61. Quá trình hệ thống hoạt động	54
Hình 62. Hình ảnh class man thực tế từ camera	55
Hình 63. Hình ảnh chạy thực tế của quảng cáo ứng với class man	55
Hình 64. Hình ảnh class girl thực tế từ camera.....	56
Hình 65. Hình ảnh chạy thực tế của quảng cáo ứng với class girl.....	56

DANH MỤC CÁC THUẬT NGỮ, CHỮ VIẾT TẮT

Viết tắt	Tiếng Anh
AI	Artificial Intelligence
YOLO	You Only Look Once
COCO	Common Objects in Context
R-CNN	Region-Based Convolutional Neural Networks
ANN	Artificial Neural Networks
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
SSD	Single Shot MultiBox Detector
DNN	Deep neural Network
RPN	Loss Function
IOT	Internet of Things
FCNN	Full Convolution Neural Network
AP	Average Precision
IOU	Intersection Over Union
GPU	Graphics Processing Unit
CUDA	Compute Unified Device Architecture
CuDNN	CUDA® Deep Neural Network
SO-DIMM	Small Outline Dual Inline Memory Module
RAM	Random Access Memory
LPDDR4	Low-Power Double Data Rate (generation 4)
USB	Universal Serial Bus

HDMI	High-Definition Multimedia Interface
i2c	Inter-Integrated Circuit
SPI	Serial Peripheral Bus
UART	Universal Asynchronous Receiver – Transmitter
M.2	NVMe (Non-Volatile Memory Express)
MIPI-CSI	Mobile Industry Processor Interface- Camera Serial Interface
eMMC	Embedded Multi-Media Controller
CCTV	Closed Circuit Television
API	Application Programming Interface
FPS	Frames Per Second
FPN	Feature Pyramid Networks
HOG	Histogram of Oriented Gradients
Linear SVM	Linear Support Vector Machine
TP	True Positive
FP	False Positive
FN	False Negative
ACU	Area Under the Curve

CÁC THƯ VIỆN PYTHON ĐƯỢC SỬ DỤNG

Pytorch
Numpy
Opencv
Argpase
Pandas
Tensonflow
TensorRT(7.2)
Time
Pycuda
Threading
Os
Flash
Socket
http.server
Queue
Json
Select
Subprocess
Ctypes
...

CHƯƠNG I: TỔNG QUAN

Chương này trình bày tình hình nghiên cứu trong và ngoài nước về đề tài nhận diện đối tượng khách hàng phục vụ cho mục đích giới thiệu những sản phẩm phù hợp với từng đối tượng khác hàng cụ thể. Đây chính là lí do để chọn đề tài và tóm tắt quá trình thực hiện.

1. Đặt vấn đề

Trong vài năm trở lại đây, Artificial Intelligence (trí tuệ nhân tạo) hay cụ thể là Machine Learning (Máy học) đang phát triển. Trí tuệ nhân tạo (AI) đang đi vào cuộc sống một cách mạnh mẽ, thay thế nhiều công việc thủ công, tốn sức lao động. Hiện nay trên thế giới, các cường quốc đều xây dựng chiến lược phát triển riêng cho AI, lấy công nghệ AI làm cốt lõi cho sự tăng tốc của nền kinh tế. Ứng dụng của AI rất đa dạng và ảnh hưởng tích cực đến đời sống như: Ô tô tự lái, hệ thống dịch tự động, chatbox tự động trả lời,... và còn nhiều ứng dụng khác phục vụ cho con người. Trong đó Object Detection được biết đến là một trong những lĩnh vực quan trọng của Trí tuệ nhân tạo (Artificial Intelligence) và thị giác máy (Computer Vision). Computer Vision là một lĩnh vực bao gồm các phương pháp thu nhận, xử lý ảnh kỹ thuật số, phân tích và nhận dạng các hình ảnh, phát hiện các đối tượng, tạo ảnh, siêu phân giải hình ảnh và nhiều hơn vậy.

Object Detection đề cập đến khả năng của hệ thống máy tính và phần mềm để định vị các đối tượng trong một hình ảnh và xác định từng đối tượng. Object Detection đã được sử dụng rộng rãi để phát hiện khuôn mặt, phát hiện xe, đếm số người đi bộ, hệ thống bảo mật và xe không người lái. Có nhiều cách để nhận diện đối tượng có thể được sử dụng cũng như trong nhiều lĩnh vực thực hành.



Hình 1. Ảnh giới thiệu

Từ đó đã có nhiều công ty làm về mảng dịch vụ quảng cáo đã đưa Object Detection vào các sản phẩm của mình, một trong đó việc phát hiện và phân chia các nhóm đối tượng khách hàng thông qua thị giác máy tính đang là một trong những bài toán được sử dụng nhiều nhất.

2. Mô tả bài toán

Trong thời đại phát triển như hiện nay, dữ liệu lớn, tính tự động và thông minh của máy móc đang phát triển rất nhanh, có vị trí rất quan trọng. Trong đó các ngành dịch vụ đang phát triển mạnh nhằm phục vụ tốt cho mọi người. Hiện nay cách ngành dịch vụ quảng cáo đã đưa những biển quảng cáo màn hình LED thay thế những biển quảng cáo giấy. Việc sử dụng biển quảng cáo điện tử đã giúp cho các nhà quảng cáo tận dụng được tối đa diện tích quảng cáo với đa dạng các mặt hàng, người xem cũng không còn cảm thấy nhàm chán với những biển quảng cáo cũ. Nhưng hạn chế của biển quảng cáo mới này là những quảng cáo đang được hiện thị có thể không phù hợp với đối tượng đang đứng xem. Vì vậy việc được AI vào những biển quảng cáo sẽ giúp chúng ta đưa được những quảng cáo phù hợp với những đối tượng khách hàng là vô cùng cần thiết.

Bài toán phát hiện và phân loại nhóm đối tượng khác hàng và quảng cáo các sản phẩm theo độ tuổi, theo giới tính. Giúp khách hàng tiếp cận một cách nhanh nhất về các sản phẩm khách hàng đang cần. Giúp khách hàng tiết kiệm được thời gian khi lựa chọn sản phẩm. Giúp các nhà sản xuất có thể giới thiệu sản phẩm của họ đến khách hàng một cách nhanh và chính xác theo độ tuổi, Giúp tiết kiệm năng lượng.

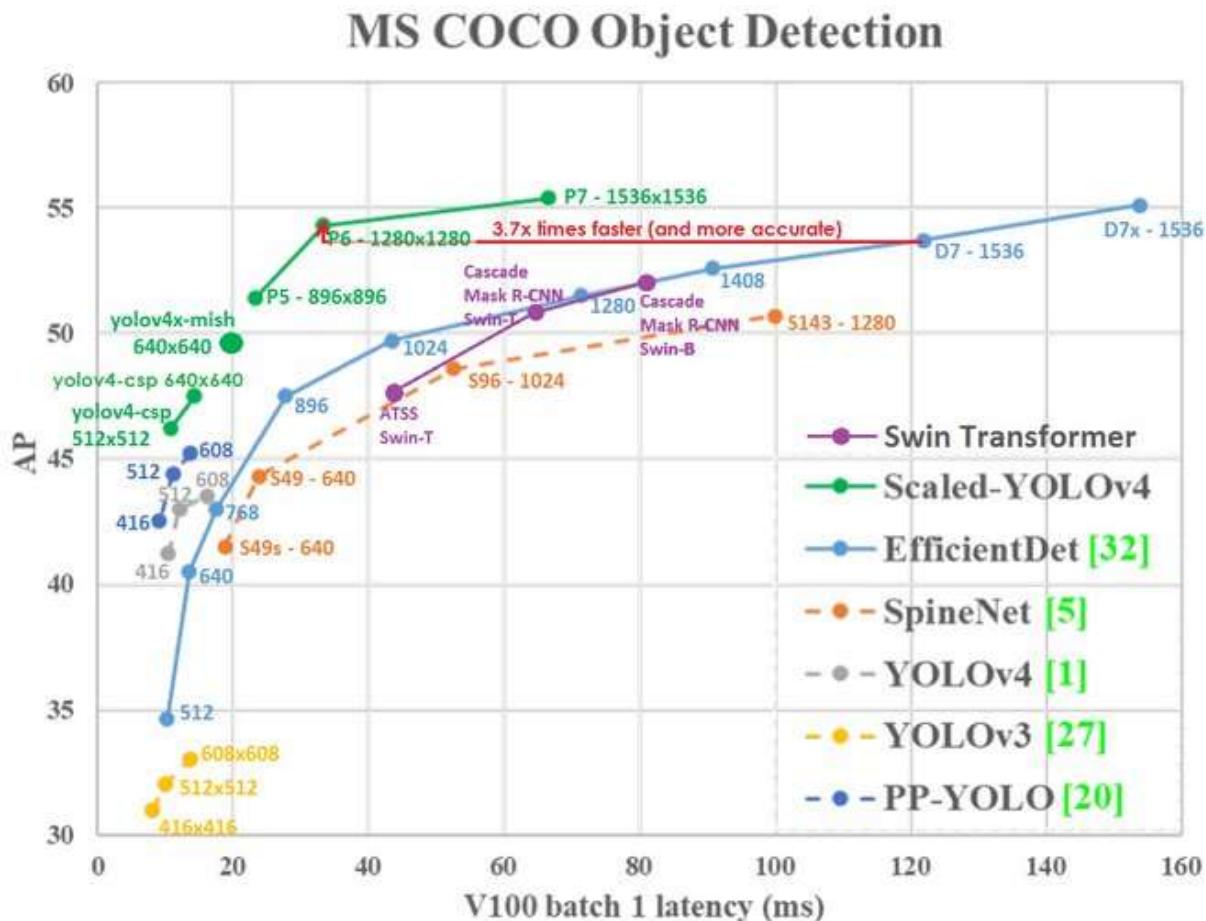
Lĩnh vực này đã có nghiên cứu từ những năm 1980. Cũng đã có nhiều bài toán về nhận diện đối tượng, đa số nhận diện hình ảnh, đưa hình ảnh vào và trả về kết quả đối tượng đó gì. Các thuật toán có thể giải quyết được vấn đề phân loại biển báo qua hình ảnh như KNN, CNN, support vector machine (SVM),.. và các thuật toán phát hiện object real-time như You Only Look Once (YOLO), single shot multi-box detector (SSD), faster region CNN (F-RCNN), ...

Đối với bài toán nhận diện đối tượng khách hàng, tức là nhận diện bức ảnh đầu vào là con người và cho biết người đó là nam hay nữ và nằm trong khoảng độ tuổi bao nhiêu. Đối với ảnh đầu vào, ta phân loại vào 1 classes nhất định. Đây là bài toán phân lớp (Classification). Hiện nay có rất nhiều thuật toán để phân lớp dữ liệu từ cổ điển đến hiện đại. Nhưng mạng Nơ-ron tích chập được đánh giá là thuật toán hiệu quả, tránh được tác động từ môi trường, nhiễu, sự thay đổi khoảng cách ảnh đến camera. Trong đề tài này, mục tiêu chúng em đặt ra là xây dựng mô hình nhận diện con người trong ảnh, video và (real-time). Từ đó so sánh hiệu quả giữa các mô hình khác nhau.

3. Phương pháp nghiên cứu của đề tài.

Trong đề tài đồ án này, sẽ sử dụng thuật toán mới nhất của YOLO là YOLOv4 để nhận diện, YOLOv4 có đáp ứng thời gian thực không khác so với YOLOv3 nhưng cho kết quả nhận diện chính xác hơn. Và vận hành trên một thiết bị nhúng (jetson nano) với một thư viện hỗ trợ Tensorrt (7.2). Tiếp đó dùng thư viện flask để xây dựng trang web trình chiếu video quảng cáo.

Dưới đây là bảng so sánh của các model detection với tập dữ liệu COCO



Hình 2. MS COCO Object Detection (<https://twitter.com/alexeyab84/status/1380699160748421121>)

Tập dữ liệu COCO là một trong những tập dữ liệu hình ảnh phổ biến nhất hiện có, với các ứng dụng như phát hiện đối tượng, phân đoạn và chú thích - khá ngạc nhiên là có rất ít hướng dẫn toàn diện nhưng đơn giản, từ đầu đến cuối tồn tại.

4. Tóm tắt quá trình thực hiện đề tài

Quá trình thực hiện đề tài hiện đề tài được chia thành nhiều giai đoạn, mỗi giai đoạn thực hiện một công việc như đã trình bày ở phần mục lục. Chi tiết cho từng giai đoạn như sau:

- Tìm hiểu về Deep Learning, các ứng dụng và hệ thống nhúng để thử hiện dự án.
- Nghiên cứu xây dựng một mô hình CNN, YOLO.
- Tham khảo cách sử dụng các thư viện lựa chọn các thư viện phù hợp.
- Quay phim, thu thập ảnh và tiền xử lý dữ liệu. Phân chia dataset thành tập training, tập validation và tập test.
- Xây dựng mô hình phân loại và nhận diện con người.
- Triển khai trên thiết bị IOT.

CHƯƠNG II: GIỚI THIỆU THIẾT BỊ

Giới thiệu về Jetson Nano:

- Jetson Nano là một máy tính tích hợp cỡ nhỏ có kích thước của một bảng mạch. Kích thước và các chuẩn kết nối của Jetson Nano là SO-DIMM.
- Trái tim của Jetson Nano chính là con chip Tegra X1 của Nvidia có tên mã là Erista.
- Jetson Nano được trang bị 4GB Ram LPDDR4 và 16GB bộ nhớ trong eMMC.
- Jetson Nano có bốn nhân Cortex A57 với xung nhịp tối đa là 1,43Ghz. GPU tích hợp 128 nhân hoạt động.
- Sản phẩm này còn được trang bị số lượng cổng kết nối khá đầy đủ: 4 cổng USB (gồm ba cổng USB 2.0 và 1 cổng USB 3.0), cổng HDMI, DisplayPort và một cổng Ethernet, cùng với đó là những chuẩn kết nối phổ biến như SDIO, I2C, SPI và UART. Jetson Nano còn có cả cổng kết nối M.2, giao thức MIPI-CSI để kết nối với máy ảnh.
- Sản phẩm này sẽ được bán ra với mức giá 129 USD (khoảng 3 triệu VND) dưới dạng một máy tính hoàn chỉnh.



Hình 3. Jetson nano

Giới thiệu về thẻ nhớ:

- Chuẩn giao tiếp : MicroSDHC.
- Dung lượng : 32GB.
- Tốc độ đọc : 100MB/S.
- Tốc độ ghi 10MB/S.
- Có thẻ chống nước, nhiệt độ, chống shock, chống bể cong.



Hình 4. Thẻ nhớ sandisk ultra 32gb

Giới thiệu về nguồn:

Nguồn Pi 3 / Pi 3+ 5V 2.5A dành cho Raspberry là nguồn có chất lượng rất tốt, với dòng ra lên tới 2.5A, cung cấp đủ nguồn điện với dòng cho Pi để hoạt động ổn định, đạt hiệu suất làm việc tốt nhất.

• Thông Số Kỹ Thuật:

Đầu vào :AC100- 240VAC(50/60Hz).

Đầu ra: 5V - 2.5A.

Đi kèm: dây nguồn MicroUSB.

Kiểu chân cắm: Dẹt.

Độ dài dây cáp: 1m.

Ngõ ra: MicroUSB.

Có chip riêng chống sốc điện.

Kiểm soát nhiệt độ, tự động ngắt điện nếu quá nóng

• Đặc điểm nổi bật:

Với dòng điện ra lên tới 2.5A đảm bảo đủ dòng điện cho mạch hoạt động ổn định.

Nguồn có chip kiểm soát nhiệt độ, tự động ngắt nếu quá nóng.



Hình 5. Bộ nguồn Pi 3 / Pi 4+ 5V 2.5A

Giới thiệu camera:

Camera Raspberry Pi V2 IMX219 8MP là phiên bản Camera Module dành cho Raspberry Pi mới nhất sử dụng cảm biến ảnh IMX219 8-megapixel từ Sony thay cho cảm biến cũ là OV5647. Với cảm biến IMX219 8-megapixel từ Sony, Camera Module cho Raspberry Pi đã có được sự nâng cấp vượt trội về cả chất lượng hình ảnh, video cũng như độ bền. Camera Raspberry Pi V2 IMX219 8MP có thể sử dụng với Raspberry Pi để chụp hình, quay phim với chất lượng HD 1080p30, 720p60 hoặc VGA90, cách sử dụng cũng như lập trình với Camera Module trên Raspberry Pi cũng rất dễ dàng, chỉ cần cắm vào cổng Camera trên Raspberry Pi và qua 1 vài bước thiết lập là có thể dùng được. Camera Raspberry Pi V2 IMX219 8MP có thể điều khiển thông qua MMAL và V4L APIs, có rất nhiều bộ thư viện được cộng đồng Raspberry Pi phát triển trên Python giúp cho việc tìm hiểu và sử dụng trở nên dễ dàng hơn rất nhiều.



Hình 6. Camera raspberry pi v2 8mp

Thông số kỹ thuật:

Camera Raspberry Pi V2 8MP dùng cho máy tính nhúng Raspberry Pi.

- | | |
|---|---|
| <ul style="list-style-type: none"> ○ Cảm biến IMX219 từ Sony. ○ Số điểm ảnh: 8MP. ○ Lens: Fixed focus. ○ Camera specifications: ○ CCD size : 1/4inch. ○ Aperture (F) : 2.0. ○ Focal Length : 3.04mm . ○ Angle of View (diagonal) : 62.2 degree. | <ul style="list-style-type: none"> ○ Camera Resolution: 3280 x 2464 pixel stills. ○ Video Resolution: HD 1080p30, 720p60 and 640x480p90 video. ○ Dimensions: 25mm x 23mm x 9mm. ○ Connector: ribbon connector. ○ Interface: CSI. |
|---|---|

CHƯƠNG III : TÌM HIỂU DEEP LEARNING, YOLO VÀ NGÔN NGỮ PYTHON

1. Deep learning (Học sâu):

Deep Learning là một nhánh nhỏ ở bên trong Machine Learning, liên quan đến các thuật toán lấy cảm hứng từ cấu trúc và hoạt động của bộ não được gọi là Mạng thần kinh nhân tạo (Artificial Neural Networks). Hiện nay, Deep Learning đang được phát triển rất mạnh mẽ, ngày càng có thêm các thuật toán ra đời với độ chính xác cao, phục vụ cho rất nhiều mục đích trong cuộc sống. Deep Learning có 2 nhánh chính là thị giác máy tính (Computer Vision) và xử lý ngôn ngữ tự nhiên (Natural Language Processing) được ứng dụng trong nhận diện hình ảnh, nhận diện giọng nói, xử lý ngôn ngữ tự nhiên, chuyển đổi văn sang giọng nói, ... Hiện nay Deep Learning được sử dụng rất nhiều để giải quyết các vấn đề do nó có thể giải quyết các bài toán có kích thước đầu vào lớn với hiệu năng cũng như độ chính xác vượt trội so với các phương pháp phân lớp truyền thống.

Mạng thần kinh nhân tạo ANN là một hệ tập hợp các chương trình và cấu trúc dữ liệu mô phỏng cách hoạt động của bộ não con người. Còn Machine learning là những thuật toán có khả năng huấn luyện máy tính “học” từ một lượng lớn dữ liệu được cung cấp tùy vào nhu cầu thì dữ liệu sẽ được dán nhãn trước hoặc không gán nhãn,

Ý tưởng về mạng thần kinh nhân tạo xuất hiện từ rất sớm khoảng những năm 50 của thế kỷ trước. Nhưng việc tạo ra những mạng nơ-ron hoạt động hiệu quả là một điều không hề dễ dàng, mạng nơ-ron sẽ cho kết quả tốt khi:

- Có bộ dữ liệu đầu vào lớn
- Kiến trúc mạng neural lớn hơn
- Khả năng tính toán của máy tính tốt hơn

Deep learning có 2 mô hình lớn là Convolutional Neural Network (CNN) cho bài toán có input là ảnh và Recurrent neural network (RNN) cho bài toán dữ liệu dạng chuỗi (sequence) . Với cơ sở bài toán này là xử lý với input là hình ảnh, ta có thể sử dụng CNN để có thể giải quyết việc phân loại đối tượng khách hàng.

2. Giới thiệu về yolo:

You only look once (YOLO) là một mô hình CNN để detect object mà một ưu điểm nổi trội là nhanh hơn nhiều so với những mô hình cũ. Thậm chí có thể chạy tốt trên những IOT device như raspberry pi, jetson nano, jeston tx1, ...

Sự phát triển của mạng neural đang dần làm hiện thực hoá khái niệm chúng ta vẫn thường gọi là Computer Vision - Thị giác máy tính. Tuy nhiên, tại thời điểm ban đầu, việc sử dụng Neural Network vẫn còn gặp nhiều khó khăn. Khi bạn muốn phát hiện ra object trong một bức ảnh, sau đó đánh nhãn cho object đó, các phương pháp lúc bấy giờ quá chậm để phục vụ trong real-time, hoặc đòi hỏi thiết bị mạnh mẽ, đắt đỏ, cho đến khi YOLO ra đời. YOLO và sau này là YOLOv2 có khả năng gán nhãn cho toàn bộ object trong khung hình với chỉ duy nhất một operation. Có thể nói YOLO đã xây dựng một hướng tiếp cận đầu tiên giúp đưa Object detection thực sự khả thi trong cuộc sống.

Những Region Proposal Classification network khác (Fast RCNN) thực hiện việc phát hiện trên các đề xuất khu vực (Region proposal), do đó sau cùng sẽ phải thực hiện dự đoán nhiều lần cho các region khác nhau, scale khác nhau trong một ảnh.

Mô hình này có nhiều điểm vượt trội so với các hệ thống classifier-based. Tại test phase, YOLO sẽ "nhìn" toàn bộ bức ảnh (thay vì từng phần bức ảnh), vì vậy những prediction của nó được cung cấp thông tin bởi nội dung toàn cục của bức ảnh. Ngoài ra, dự đoán được đưa ra chỉ với một mạng đánh giá duy nhất, thay vì hàng nghìn như R-CNN. Vì vậy tốc độ của YOLO là cực nhanh, nhanh gấp hàng nghìn lần so với R-CNN, hàng trăm lần so với Fast R-CNN.

YOLOv1:

Yolov1 sử dụng framework Darknet được train trên tập ImageNet-1000. Nó không thể tìm thấy các object nhỏ nếu chúng xuất hiện dưới dạng một cụm. Phiên bản này gặp khó khăn trong việc phát hiện các đối tượng nếu hình ảnh có kích thước khác với hình ảnh được train.

YOLOv2:

YOLOv2 sử dụng vài trick để cải thiện quá trình huấn luyện và nâng cao performance. cũng sử dụng Full-convolutional model như SSD, tuy nhiên vẫn train toàn bộ các bức ảnh thay vì chỉ hard negative. Theo hướng của Faster R-CNN, YOLOv2 sẽ điều chỉnh thông số bounding box trước, thay vì predict ngay w và h, tuy nhiên toạ độ (x, y) vẫn được dự đoán trực tiếp.

Cải thiện độ chính xác và chuẩn hóa hàng loạt, thêm chuẩn hóa hàng loạt trong các lớp chập. Điều này loại bỏ sự cần thiết phải bỏ (dropouts and pushes) học và đẩy mAP lên 2%. Phân loại độ phân giải cao.

Việc đào tạo Yolo bao gồm 2 giai đoạn. Đầu tiên, Yolo đào tạo một mạng phân loại như VGG16. Sau đó, Yolo thay thế các lớp được kết nối đầy đủ bằng một lớp chập và giữ lại từ đầu đến cuối để phát hiện đối tượng. Yolo huấn luyện bộ phân loại với các hình ảnh có kích thước 224×224 , sau đó là các hình ảnh có kích thước 448×448 để phát hiện đối tượng. Yolo v2 bắt đầu với các hình ảnh 224×224 cho đào tạo phân loại nhưng sau đó thử lại bộ phân loại một lần nữa với các hình ảnh 448×448 sử dụng ít kỷ nguyên hơn nhiều. Điều này làm cho việc đào tạo máy dò dễ dàng hơn và di chuyển mAP lên 4%.

type	patch size/ stride	output size	depth	#1x1	#3x3 reduce	#3x3	#5x5 reduce	#5x5	pool proj	params	ops
convolution	$7 \times 7/2$	$112 \times 112 \times 64$	1							2.7K	34M
max pool	$3 \times 3/2$	$56 \times 56 \times 64$	0								
convolution	$3 \times 3/1$	$56 \times 56 \times 192$	2		64	192				112K	360M
max pool	$3 \times 3/2$	$28 \times 28 \times 192$	0								
inception (3a)		$28 \times 28 \times 256$	2	64	96	128	16	32	32	159K	128M
inception (3b)		$28 \times 28 \times 480$	2	128	128	192	32	96	64	380K	304M
max pool	$3 \times 3/2$	$14 \times 14 \times 480$	0								
inception (4a)		$14 \times 14 \times 512$	2	192	96	208	16	48	64	364K	73M
inception (4b)		$14 \times 14 \times 512$	2	160	112	224	24	64	64	437K	88M
inception (4c)		$14 \times 14 \times 512$	2	128	128	256	24	64	64	463K	100M
inception (4d)		$14 \times 14 \times 528$	2	112	144	288	32	64	64	580K	119M
inception (4e)		$14 \times 14 \times 832$	2	256	160	320	32	128	128	840K	170M
max pool	$3 \times 3/2$	$7 \times 7 \times 832$	0								
inception (5a)		$7 \times 7 \times 832$	2	256	160	320	32	128	128	1072K	54M
inception (5b)		$7 \times 7 \times 1024$	2	384	192	384	48	128	128	1388K	71M
avg pool	$7 \times 7/1$	$1 \times 1 \times 1024$	0								
dropout (40%)		$1 \times 1 \times 1024$	0								
linear		$1 \times 1 \times 1000$	1							1000K	1M
softmax		$1 \times 1 \times 1000$	0								

Hình 7. VGG16 (<https://in4tintuc.com/vgg16-la-gi/>)

YOLOv3:

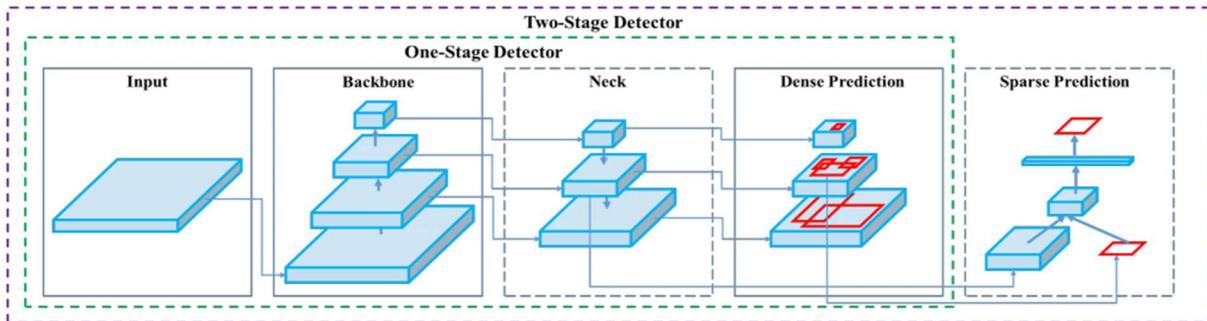
Yolov3 cực kỳ nhanh và chính xác. Trong mAP được đo ở 0,55 IOU YOLOv3 ngang bằng với Focal Loss nhưng nhanh hơn khoảng 4 lần. Hơn nữa, bạn có thể dễ dàng đánh đổi giữa tốc độ và độ chính xác chỉ bằng cách thay đổi kích thước của mô hình, không cần đào tạo lại.

YOLOv4:

Yolov4 là một mô hình mạng CNN cho việc phát hiện, nhận dạng, phân loại đối tượng. Yolo được tạo ra từ việc kết hợp giữa các convolutional layers và connected layers. Trong đó các convolutional layers sẽ trích xuất ra các feature của ảnh, còn full-connected layers sẽ dự đoán ra xác suất đó và tọa độ của đối tượng.

Một mô hình object detection được cấu tạo bởi các thành phần:

- Input.
- Backbone.
- Neck.
- Head.



Input: { Image, Patches, Image Pyramid, ... }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], ... }

Neck: { FPN [44], PANet [49], Bi-FPN [77], ... }

Head:
 Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], ... }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], ... }

Hình 9. Cấu trúc của YOLOv4 (<https://devai.info/2021/02/24/series-yolo-4-tim-hieu-cau-truc-yolov1v2v3-va-v4-phan-2/>)

Type	Filters	Size	Output
Convolutional	32	3×3	256×256
Convolutional	64	$3 \times 3 / 2$	128×128
1x	32	1×1	
	64	3×3	
	Residual		128×128
Convolutional	128	$3 \times 3 / 2$	64×64
2x	64	1×1	
	128	3×3	
	Residual		64×64
Convolutional	256	$3 \times 3 / 2$	32×32
8x	128	1×1	
	256	3×3	
	Residual		32×32
Convolutional	512	$3 \times 3 / 2$	16×16
8x	256	1×1	
	512	3×3	
	Residual		16×16
Convolutional	1024	$3 \times 3 / 2$	8×8
4x	512	1×1	
	1024	3×3	
	Residual		8×8
Avgpool		Global	
Connected		1000	
Softmax			

Hình 8. Darknet-53 của yolov4
(<https://paperswithcode.com/method/darknet-53>)

Mô hình thuật toán của YOLO:

Một trong những ưu điểm mà YOLO đem lại đó là chỉ sử dụng thông tin toàn bộ bức ảnh một lần và dự đoán toàn bộ object box chứa các đối tượng, mô hình được xây dựng theo kiểu end-to-end nên được huấn luyện hoàn toàn bằng gradient descent. Sau đây, mình sẽ trình bày chi tiết về mô hình YOLO.

Grid System (hệ thống lưới):

Ảnh được chia thành ma trận ô vuông 7×7 , mỗi ô vuông bao gồm một tập các thông tin mà mô hình phải dự đoán. Đối tượng duy nhất mà ô vuông đó chứa. Tâm của đối tượng cần xác định nằm trong ô vuông nào thì ô vuông đó chứa đối tượng đó. Ví dụ tâm của cô gái nằm trong ô vuông màu xanh, do đó mô hình phải dự đoán được nhãn của ô vuông đó là cô gái. Lưu ý, cho dù phần ảnh cô gái có nằm ở ô vuông khác mà tâm không thuộc ô vuông đó thì vẫn không tính là chứa cô gái, ngoài ra, nếu có nhiều tâm nằm trong một ô vuông thì chúng ta vẫn chỉ gán một nhãn cho ô vuông đó thôi. Chính ràng buộc mỗi ô vuông chỉ chứa một đối tượng là nhược điểm của mô hình này. Nó làm cho ta không thể detect



Hình 10. Hình ảnh minh họa Grid System

những object có tâm nằm cùng một ô vuông. Tuy nhiên chúng ta có thể tăng grid size từ 7×7 lên kích thước lớn hơn để có thể detect được nhiều object hơn. Ngoài ra, kích thước của ảnh đầu vào phải là bội số của grid size.

Mỗi ô vuông chịu trách nhiệm dự đoán 2 boundary box của đối tượng. Mỗi boundary box dự đoán có chứa object hay không và thông tin vị trí của boundary box gồm trung tâm

boundary box của đối tượng và chiều dài, rộng của boundary box đó. Ví vuông màu xanh cần dự đoán 2 boundary box chứa cô gái như hình minh họa ở dưới. Một điều cần lưu



Hình 11. Hình ảnh minh họa Grid

ý, lúc cài đặt chúng ta không dự đoán giá trị pixel mà cần phải chuẩn hóa kích thước ảnh về đoạn từ [0-1] và dự đoán độ lệch của tâm đối tượng đến box chứa đối tượng đó. Ví dụ, chúng ta thay vì dữ đoán vị trí pixel của điểm màu đỏ, thì cần dự đoán độ lệch a,b trong ô vuông chứa tâm object.

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

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

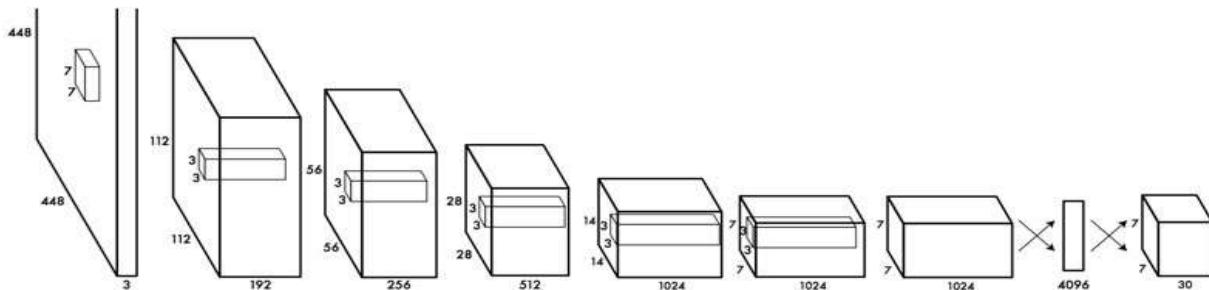
Như vậy ta có với mỗi ô vuông thì chúng ta cần dữ đoán một vector sẽ có (nbox+4*nbox+nclass) chiều. Ví dụ, chúng ta cần dự đoán 2 box, và 3 lớp đối với mỗi ô vuông thì chúng sẽ có một ma trận 3 chiều 7x7x30 chứa toàn bộ thông tin cần thiết.

object 1?	object 2?	offset x1	offset y1	width 1	height 1	offset x2	offset y2	width 2	height 2	0	0	1
-----------	-----------	-----------	-----------	---------	----------	-----------	-----------	---------	----------	---	---	---

Hình 12. Ví dụ

CNN for YOLO Object Detection:

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



Hình 13. Mô hình CNN (<https://pbcquoc.github.io/yolo/>)

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

Loss function:

Chúng ta đã định nghĩa được những thông tin mà mô hình cần phải dự đoán, và kiến trúc của mô hình CNN. Nay giờ là lúc mà chúng ta sẽ định nghĩa hàm lỗi.

YOLO sử dụng hàm độ lỗi bình phương giữ dự đoán và nhãn để tính độ lỗi cho mô hình. Cụ thể, độ lỗi tổng của chúng ta sẽ là tổng của 3 độ lỗi con sau:

- Độ lỗi của việc dự đoán loại nhãn của object - Classification loss.
- Độ lỗi của dự đoán tạo độ cũng như chiều dài, rộng của boundary box - Localization loss.
- Độ lỗi của ô vuông có chứa object nào hay không - Confidence loss.

Chúng ta mong muốn hàm lỗi có chức năng sau. Trong quá trình huấn luyện, mô hình sẽ nhìn vào những ô vuông có chứa object. Tăng classification score lớp đúng của object đó lên. Sau đó, cũng nhìn vào ô vuông đó, tìm boundary box tốt nhất trong 2 boxes được dự đoán. Tăng localization score của boundary box đó lên, thay đổi thông tin boundary box để gần đúng với nhãn. Đối với những ô vuông không chứa object, giảm confidence score

và chúng ta sẽ không quan tâm đến classification score và localization score của những ô vuông này.

Tiếp theo, chúng ta sẽ đi lần lượt vào chi tiết ý nghĩa của các độ lỗi trên.

Classification Loss:

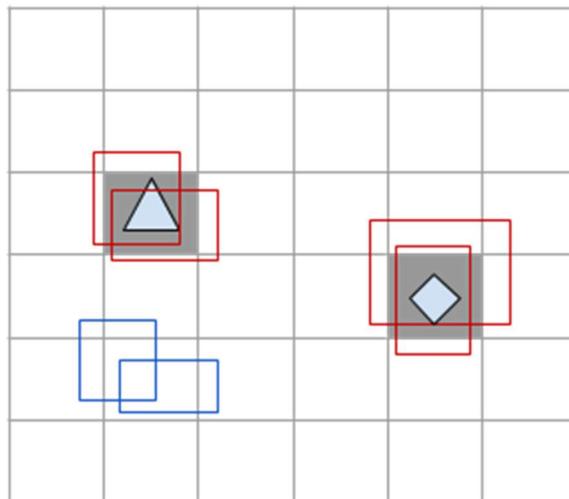
Chúng ta chỉ tính classification loss cho những ô vuông được đánh nhãn là có object. Classification loss tại những ô vuông đó được tính bằng lỗi bình phương giữa nhãn được dự đoán và nhãn đúng của nó.

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

Trong đó:

\prod_i^{obj} : bằng 1 nếu ô vuông đang xét có object ngược lại bằng 0.

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



Hình 14. Classification Loss(<https://pbcquoc.github.io/yolo/>)

Ví dụ, trong hình minh họa ở trên, chúng ta có 2 object tại ô vuông (dòng,cột) là (2,1) và (3,4), chưa object là hình tam giác và hình tứ giác đều. Độ lỗi classification loss chỉ tính cho 2 object này mà ko quan tâm đến những ô vuông khác. Lúc cài đặt chúng ta cần lưu ý phải nhân với một mask để triệt tiêu giá trị lỗi tại những ô vuông không quan tâm.

Localization Loss:

Localization loss dùng để tính giá trị lỗi cho boundary box được dự đoán bao gồm offset x,y và chiều dài, rộng so với nhãn chính xác của chúng ta. Các bạn nên lưu ý rằng, chúng ta không tính toán trực tiếp giá trị lỗi này trên kích thước của ảnh mà cần chuẩn dưới kính thước ảnh về đoạn [0-1] đối với tọa độ điểm tâm, và không dữ đoán trực tiếp điểm tâm mà

phải dự đoán giá trị lệch offset x,y so với ô vuông tương ứng. Việc chuẩn hóa kích thước ảnh và dự đoán offset làm cho mô hình nhanh hơn so với việc dự đoán giá trị mặc định.

$$L_{localization} = \sum_{i=0}^{S^2} \sum_{j=0}^B \prod_{ij}^{obj} [(offsetx_i - offsetx_i)^2 + (offsety_i - offsety_i)^2 + (width_i - width_i)^2 + (height_i - height_i)^2]$$

Độ lỗi localization loss được tính bằng tổng độ lỗi bình phương của offsetx, offsety và chiều dài, rộng trên tất cả các ô vuông có chứa object. Tại mỗi ô vuông đúng, ta chọn 1 boundary box có IOU (Intersect over union) tốt nhất, rồi sau đó tính độ lỗi theo các boundary box này. Theo hình minh họa trên chúng ta có 4 boundary box tại ô vuông đúng có viền màu đỏ, chúng ta chọn 1 box tại mỗi ô vuông để tính độ lỗi. Còn box xanh được bỏ qua.

Localization loss là độ lỗi quan trọng nhất trong 3 loại độ lỗi trên. Do đó, ta cần đặt trọng số cao hơn cho độ lỗi này.

Confidence Loss:

Confidence loss thể hiện độ lỗi giữa dự đoán boundary box đó chứa object so với nhãn thực tế tại ô vuông đó. Độ lỗi này tính nên cả những ô vuông chứa object và không chứa object.

$$L_{confidence} = \sum_{i=0}^{S^2} \sum_{j=0}^B \prod_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobject} \sum_{i=0}^{S^2} \sum_{j=0}^B \prod_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

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

Tổng kết lại, tổng lỗi của chúng ta sẽ bằng tổng của 3 loại độ lỗi trên.

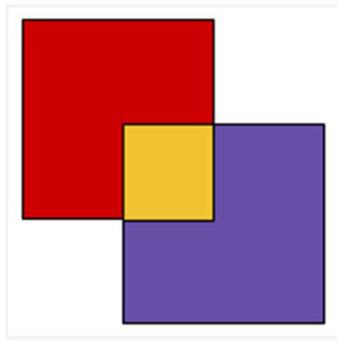
$$L_{total} = L_{classification} + L_{localization} + L_{confidence}$$

Dự đoán lớp và tạo độ boun box sau quá trình huấn luyện – Inference.

Chúng ta chỉ giữ lại những boundary box mà có chứa object nào đó. Để làm điều này, chúng ta cần tính tích của xác suất có điều kiện ô vuông thuộc về lớp i nhân với xác suất ô vuông đó chứa object, chỉ giữ lại những boundary box có giá trị này lớn hơn ngưỡng nhất định.

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

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



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

Hình 15. IOU giữa 2 box

Hạn chế của YOLO:

YOLO áp đặt các ràng buộc về không gian trên những bounding box, mỗi grid cell chỉ có thể predict rất ít bounding box và duy nhất một class. Các ràng buộc này hạn chế khả năng nhận biết số object nằm gần nhau, cũng như đối với các object có kích thước nhỏ.

YOLO sử dụng các feature tương đối thô để predict bounding box, do model sử dụng nhiều lớp downsampling từ ảnh đầu vào. Bởi các hạn chế này của model khi huấn luyện để predict bounding box từ data, dẫn đến YOLO không thực sự tốt trong việc nhận diện các object với tỉ lệ hình khối mới hoặc bất thường so với tập data. YOLOv2 đã khắc phục phần nào vấn đề này, nhưng vẫn thua kém nhiều so với FRCNN.

Ngoài ra, trong quá trình training, loss function không có sự đánh giá riêng biệt giữa error của bounding box kích thước nhỏ so với error của bounding box kích thước lớn. Việc coi chúng như cùng loại và tổng hợp lại làm ảnh hưởng đến độ chính xác toàn cục của mạng. Error nhỏ trên box lớn nhìn chung ít tác hại, nhưng error nhỏ với box rất nhỏ sẽ đặc biệt ảnh hưởng đến giá trị IOU.

(Nguồn: <https://pbcquoc.github.io/yolo/>)

2. Tìm hiểu về python:

Python là ngôn ngữ lập trình hướng đối tượng, cấp cao, mạnh mẽ, được tạo ra bởi Guido van Rossum. Nó dễ dàng để tìm hiểu và đang nổi lên như một trong những ngôn ngữ lập trình nhập môn tốt nhất cho người lần đầu tiếp xúc với ngôn ngữ lập trình. Python hoàn toàn tạo kiểu động và sử dụng cơ chế cấp phát bộ nhớ tự động. Python có cấu trúc dữ liệu cấp cao mạnh mẽ và cách tiếp cận đơn giản nhưng hiệu quả đối với lập trình hướng đối tượng. Cú pháp lệnh của Python là điểm cộng vô cùng lớn vì sự rõ ràng, dễ hiểu và cách gõ linh động làm cho nó nhanh chóng trở thành một ngôn ngữ lý tưởng để viết script và phát triển ứng dụng trong nhiều lĩnh vực, ở hầu hết các nền tảng.

Lịch sử phát triển:

Vào cuối những năm 1980, Guido Van Rossum làm việc trong Amoeba, phân phôi một nhóm hệ điều hành. Ông muốn sử dụng một ngôn ngữ thông dịch như ABC (ABC có cú pháp rất dễ hiểu) để truy cập vào những cuộc gọi hệ thống Amoeba. Vì vậy, ông quyết định tạo ra một ngôn ngữ mở rộng. Điều này đã dẫn đến một thiết kế của ngôn ngữ mới, chính là Python sau này.

Các phiên bản của python:

Phiên bản	Ngày phát hành
Python 1.0 (bản phát hành chuẩn đầu tiên)	01/1994
Python 1.6 (phiên bản 1.x cuối cùng)	05/09/2000
Python 2.0 (giới thiệu list comprehension)	16/10/2000
Python 2.7 (phiên bản 2.x cuối cùng)	03/07/2010
Python 3.0 (loại bỏ cấu trúc và mô-đun trùng lặp)	03/12/2008
Python 3.9.5 (bản mới nhất tính đến thời điểm cập nhật bài)	03/05/2021

(Nguồn: <https://vi.wikipedia.org/wiki/Python>)

CHƯƠNG IV : XÂY DỰNG ỨNG DỤNG

Để thực hiện đồ án, chúng em sử dụng Anaconda để đơn giản hóa việc cài đặt, quản lý và triển khai các packages, đặc biệt hạn chế tối đa việc xung đột các thư viện với nhau, thư viện Yolo/Darknet, ngôn ngữ lập trình Python, cùng các thư viện và môi trường cần thiết khác để xử lý hình ảnh, thuật toán như yolov4.

1. Thư viện GPU NVIDIA hỗ trợ.

CUDA.

CUDA (Compute Unified Device Architecture - Kiến trúc thiết bị tính toán hợp nhất) là một kiến trúc tính toán song song do NVIDIA phát triển. Nói một cách ngắn gọn, CUDA là động cơ tính toán trong các GPU (Graphics Processing Unit - Đơn vị xử lý đồ họa) của NVIDIA, nhưng lập trình viên có thể sử dụng nó thông qua các ngôn ngữ lập trình phổ biến. Lập trình viên dùng ngôn ngữ C for CUDA, dùng trình biên dịch PathScale Open64 C[1], để cài đặt các thuật toán chạy trên GPU. Kiến trúc CUDA hỗ trợ mọi chức năng tính toán thông qua ngôn ngữ C. Các bên thứ ba cũng đã phát triển để hỗ trợ CUDA trong Python, Fortran, Java và MATLAB.

CUDA cho phép các nhà phát triển truy nhập vào tập các chỉ lệnh ảo và bộ nhớ của các phần tử tính toán song song trong đơn vị xử lý đồ họa của CUDA (CUDA GPU). Sử dụng CUDA, các GPU mới nhất do NVIDIA sản xuất có thể dễ dàng thực hiện các tính toán như những CPU. Tuy nhiên, không giống như các CPU, các GPU có kiến trúc song song trên toàn bộ giúp cho sự tập trung vào khả năng thực thi một cách chậm rãi nhiều luồng dữ liệu một lúc, hơn là thực thi rất nhanh một luồng dữ liệu. Cách tiếp cận giải quyết các vấn đề có mục đích tổng quát này trên các GPU được gọi là GPGPU.

cuDNN.

CuDNN – NVidia CUDA® Deep Neural Network: Là một thư viện nền tảng cho các deep neural network được tăng tốc bởi GPU. cuDNN cung cấp các thiết lập được tinh chỉnh cho các thủ tục được chuẩn hóa như forward and backward convolution, pooling, normalization và các lớp kích hoạt. cuDNN là một phần của Deep Learning SDK do NVidia cung cấp.

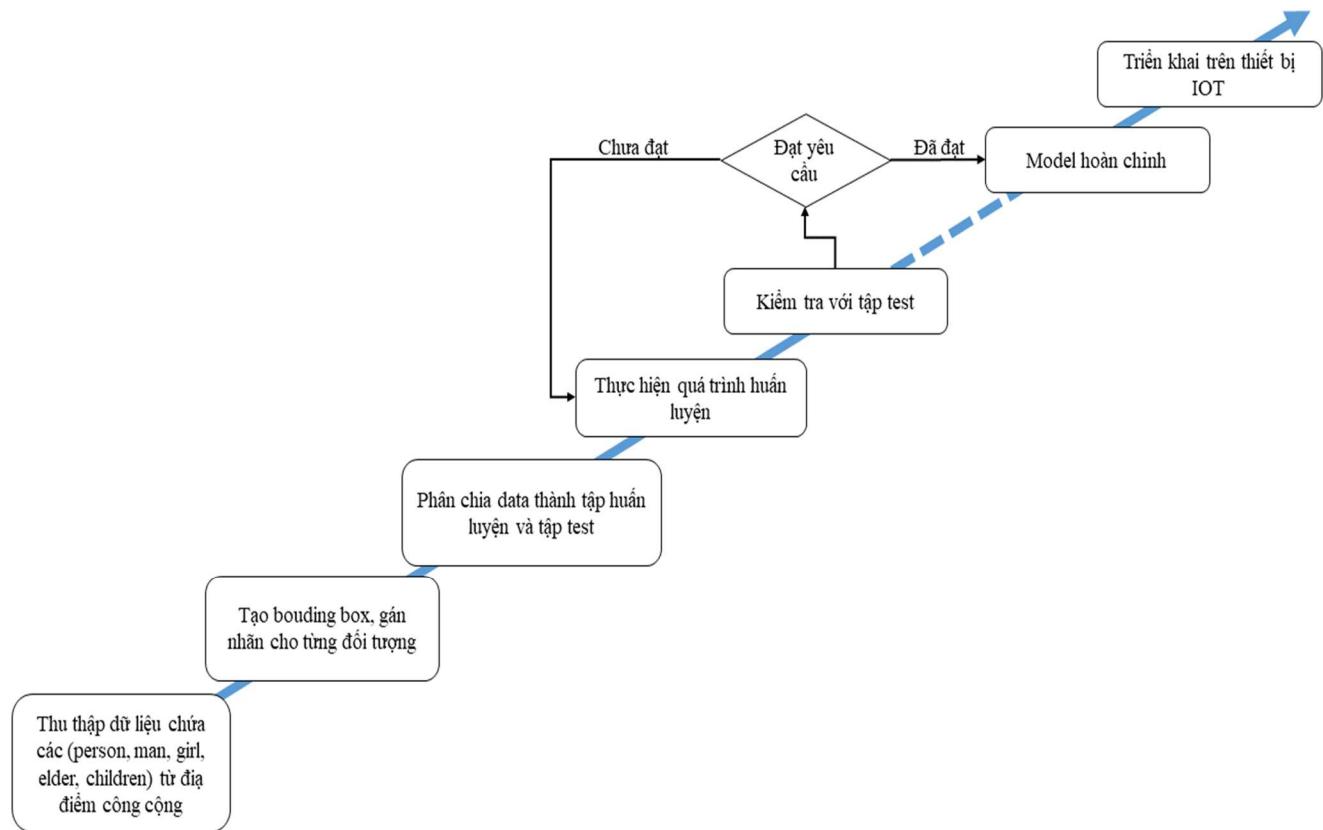
Nhận diện đối tượng realtime hoặc qua video và hình ảnh.

Trước khi nhận dạng vật thể cần phân loại thì ta cần tạo dataset cho AI sau đó training cho máy học. Sau đó mới bắt đầu nhận diện vật thể trong ảnh, video mà ta cần nhận diện.

(Nguồn: <https://vi.wikipedia.org/wiki/CUDA>)

2. Tiến trình hoàn thành dự án:

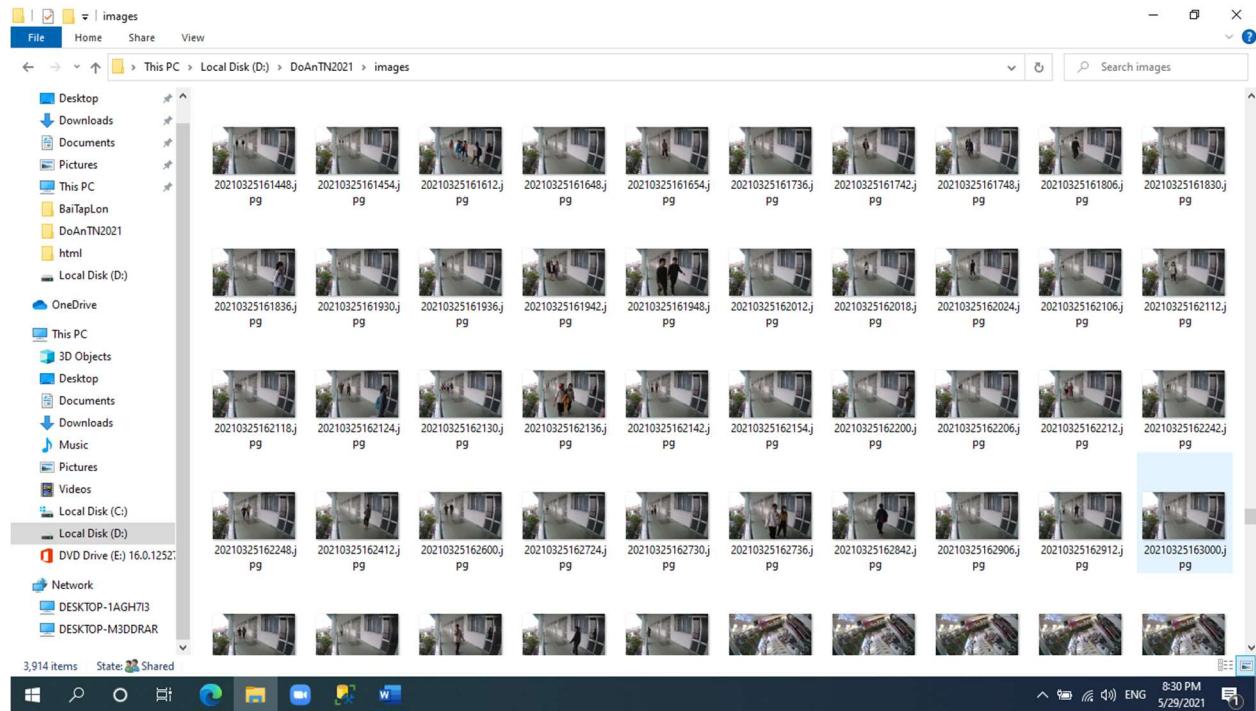
Hướng phát triển dự án:



Hình 16. Sơ đồ các bước thực hiện dự án

Bước 1 : Chuẩn bị dữ liệu ở dạng image.

Cần thu thập dữ liệu về con người ở những nơi có thể đặt bảng quảng cáo tại những nơi công cộng như các trung tâm thương mại. Với bộ dữ liệu 7000 hình thu thập từ trường Đại



Hình 17. Hình ảnh sau khi được thu thập (7000 hình)

Học Công Nghiệp TP Hồ Chí Minh, Gigamall Thủ Đức, trung tâm thương mại Sài Gòn Centre, CCTV in Shopping Mall. Bao gồm: person, man, girl, elder, children.

Dưới đây là một vài hình ảnh được lấy ra từ trong bộ dữ liệu:



Hình 18. Dữ liệu IUH



Hình 19. Dữ liệu IUH



Hình 20. Dữ liệu IUH



Hình 21. Dữ liệu IUH



Hình 22. Dữ liệu Gigamall Thủ Đức



Hình 23. Dữ liệu trung tâm thương mại Sài Gòn CENTRE



Hình 24. Dữ liệu CCTV in Shopping Mall



Hình 25. Dữ liệu CCTV Camera



Hình 26. Dữ liệu từ camera mái ấm



Hình 27. Dữ liệu từ camera mái ấm

Bước 2 : Tiền xử lý dữ liệu (Bouding box, gán nhãn cho dữ liệu)

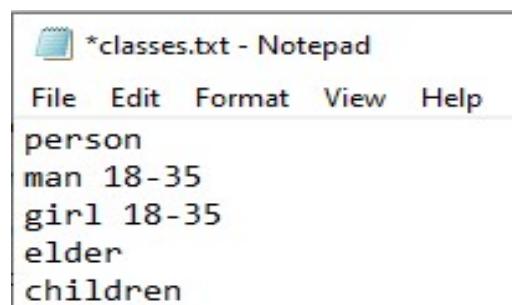
Sau khi thu thập dữ liệu, các hình ảnh sẽ được gán nhãn cho các đối tượng có trong ảnh. Các ảnh sẽ được xác định tọa độ của các box chứa đối tượng và gán nhãn cho box đó bằng chương trình LabelImg của tác giả tzutalin

Với hệ điều hành Windows, thì sử dụng anaconda để tạo một môi trường python cho tool labelImg hoạt động. (git download <https://github.com/tzutalin/labelImg>)

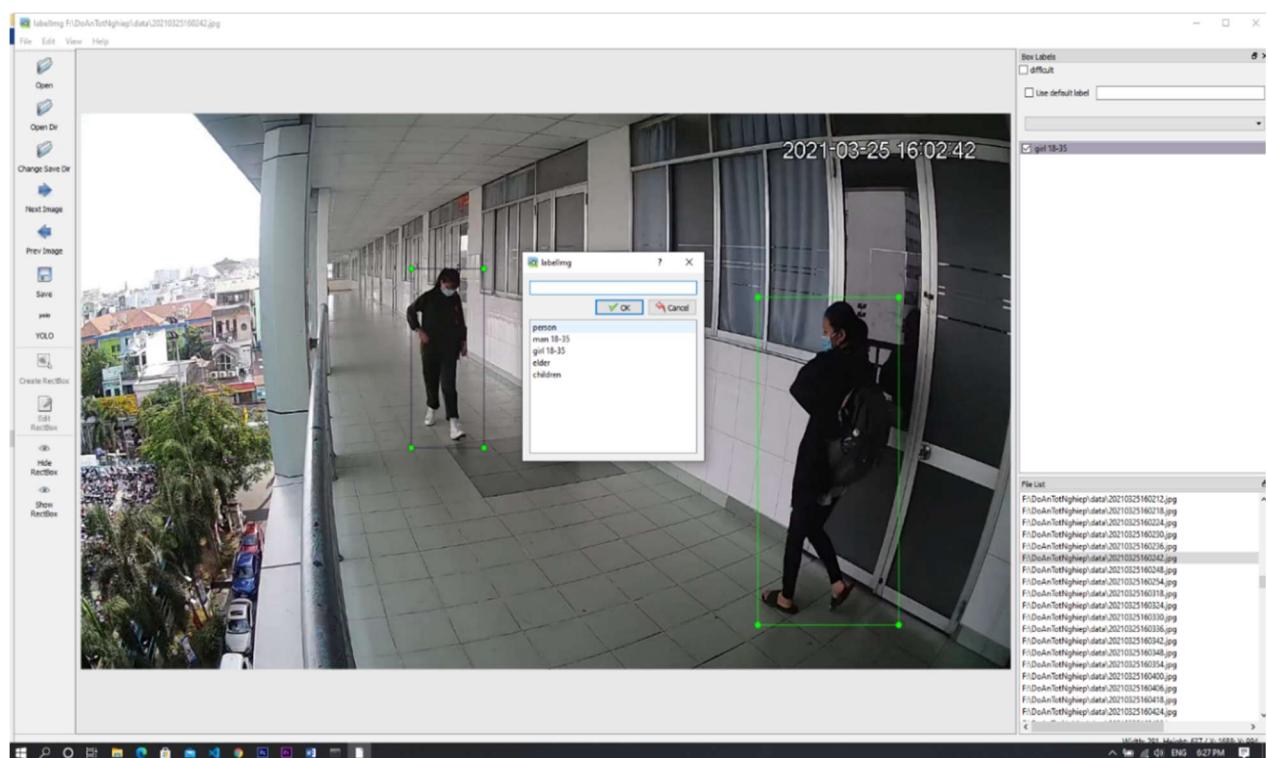
Tiếp đến cần cài các thư viện sau vào môi trường anaconda mới tạo. Install Python, PyQt5 and install lxml. Tạo một file mang tên classes.txt. Để trong thư mục lable của cho tool labelImg

Số lượng classes mà chương trình này xây dựng là 5 class. Tiến hành Bouding box, gán nhãn cho đối tượng tương ứng với các classes bằng ứng dụng labelImg.

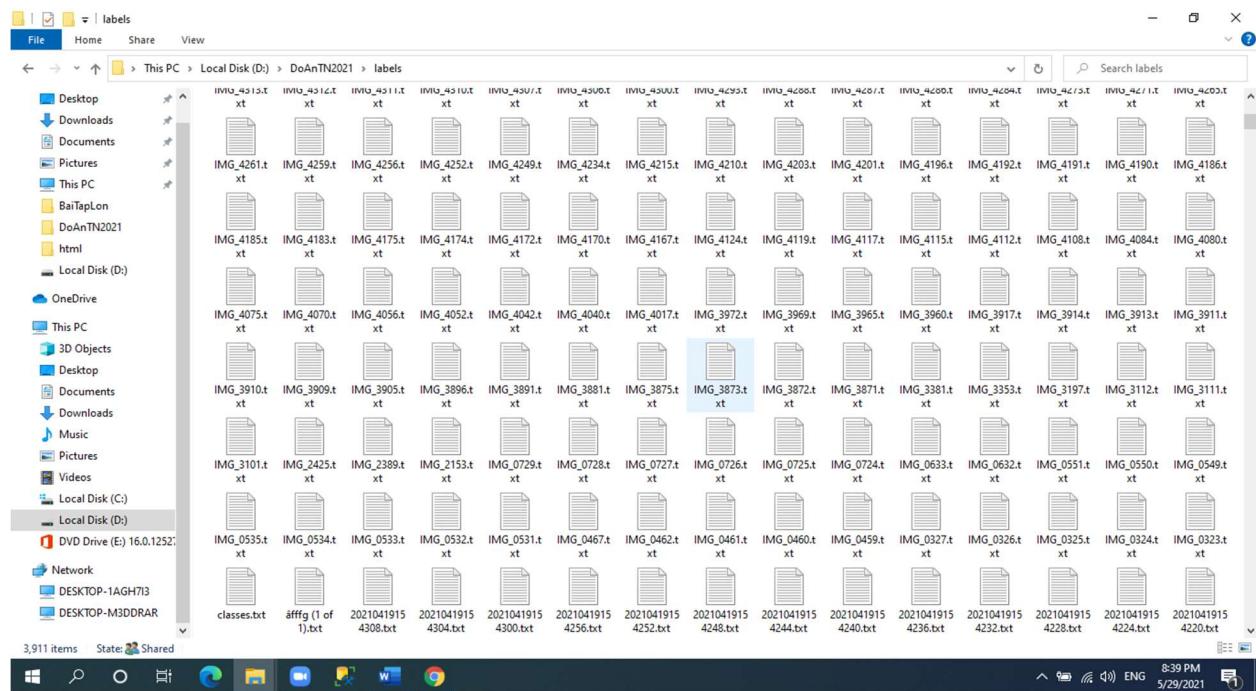
Kết quả ta có những file *.txt chứa thông tin label trong thư mục lable.



Hình 28. File classes.txt



Hình 29. Giao diện làm việc của ứng dụng labelImg



Hình 30. File table (7000 file *.txt)

Bước 3 : Training data.

Bước 3.1. File config.

Có nhiều tham số được sử dụng trong thuật toán YOLOv4 tuy nhiên ở đây chỉ đề cập đến một số tham số quan trọng ảnh hưởng lớn đến quá trình huấn luyện, các tham số còn lại được giữ mặc định theo cấu hình của tác giả.

Classes: Số class trong model 5 class

Batch size: Batch size với kích thước là 64. Các trọng số sẽ được cập nhập với gradient trung bình của từng batch.

Max batches: Số lần lặp là 15000 lần lặp tương ứng với 15000 lần cập nhật tham số. Tuy nhiên khi hàm Loss trở nên bão hòa, quá trình lặp sẽ được dừng lại.

Steps: 8000, 9000 đây là vị trí các steps sẽ giảm dần learning rate vì thuật toán đã đạt tới các điểm hội tụ không cần thiết phải có learning rate cao

Learning rate và decay: Tốc độ học khởi tạo là 0.001 với decay là 0.0005

Momentum: Momentum được đặt với giá trị 0.949

Input size: Kích thước ảnh đầu vào sẽ được resize lại thành 416 x 416 x 3.

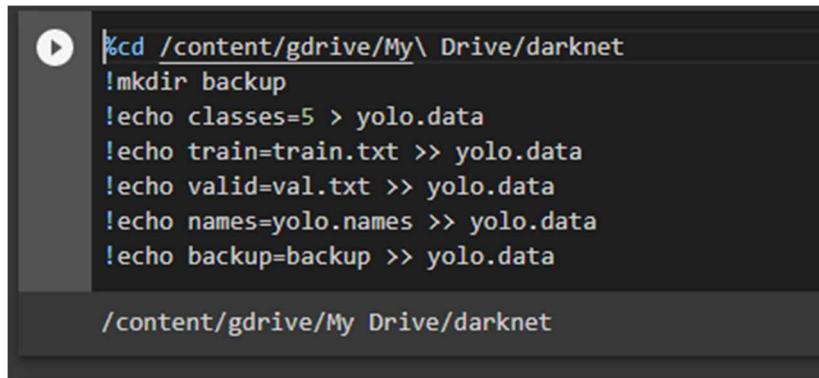
Bước 3.2. Tạo file yolo.name bằng code trong colab.



```
%cd /content/gdrive/My\ Drive/darknet
!echo "person" > yolo.names
!echo "man" >> yolo.names
!echo "girl" >> yolo.names
!echo "elder" >> yolo.names
!echo "children" >> yolo.names
```

Hình 31. Tạo file yolo.name

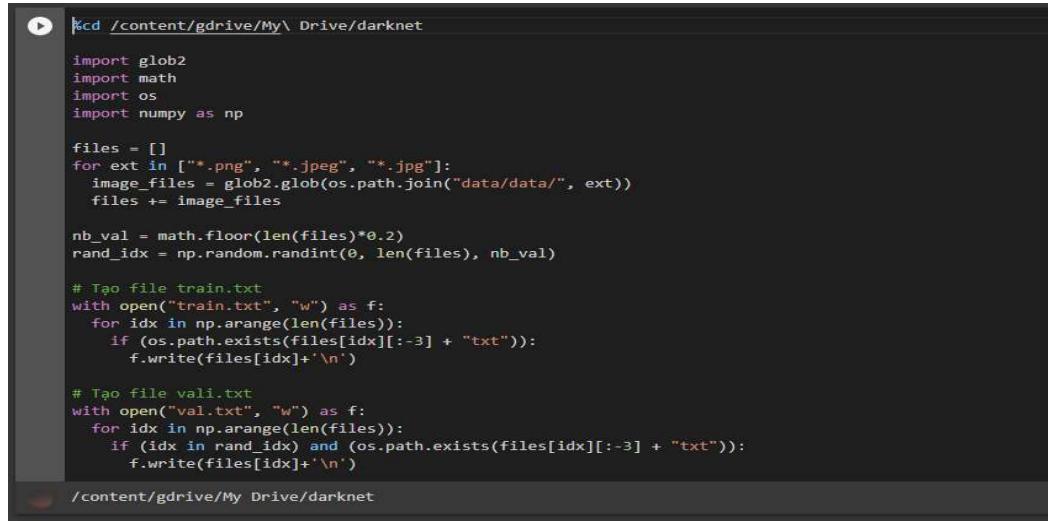
Bước 3.3. Tạo file yolo.data bằng code trong colab.



```
%cd /content/gdrive/My\ Drive/darknet
!mkdir backup
!echo classes=5 > 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
```

Hình 32. Tạo file yolo.data

Bước 3.4. Tạo ra hai file là file train.txt and val.txt.



```
%cd /content/gdrive/My\ Drive/darknet

import glob2
import math
import os
import numpy as np

files = []
for ext in ["*.png", "*.jpeg", "*.jpg"]:
    image_files = glob2.glob(os.path.join("data/data/", ext))
    files += image_files

nb_val = math.floor(len(files)*0.2)
rand_idx = np.random.randint(0, len(files), nb_val)

# Tạo file train.txt
with open("train.txt", "w") as f:
    for idx in np.arange(len(files)):
        if (os.path.exists(files[idx][:-3] + "txt")):
            f.write(files[idx]+"\n")

# Tạo file val.txt
with open("val.txt", "w") as f:
    for idx in np.arange(len(files)):
        if (idx in rand_idx) and (os.path.exists(files[idx][:-3] + "txt")):
            f.write(files[idx]+"\n")
```

Hình 33. Tạo file train.txt và file val.txt

File train.txt sẽ lưu đường dẫn của hình dùng để huấn luyện, còn file val.txt sẽ lưu đường dẫn của hình dùng để test.

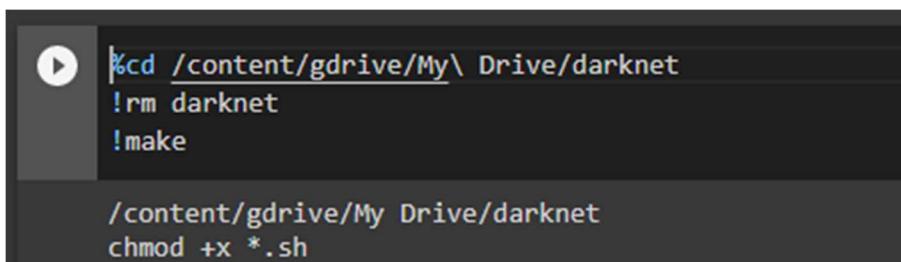
Bước 3.5. Sau khi chuẩn bị các file trên thành công ta bắt đầu tải (PRETRAIN WEIGHTS) Chúng ta sẽ tận dụng weights của phần Convolution đã được train của Darknet bằng cách tải file yolov4.conv.137 về bằng codeblock: (https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137)

Bước 3.6. Upload thư mục dataset bao gồm hình và lable lên thư mục /darknet/data/drive (hình & lable nằm chung một thư mục).

Bước 3.7. Sửa lại Makefile với những thông số dưới đây:

```
GPU=1
CUDNN=1
CUDNN_HALF=1
OPENCV=1
AVX=0
OPENMP=1
LIBSO=1
ZED_CAMERA=0
ZED_CAMERA_v2_8=0
ARCH= -gencode arch=compute_53,code=[sm_53,compute_53]
```

Bước 3.8. Tạo code block và gõ lệnh & run (hệ thống sẽ tìm và biên dịch các thư viện cần thiết có trong Makefile).



```
%cd /content/gdrive/My\ Drive/darknet
!rm darknet
!make

/content/gdrive/My Drive/darknet
chmod +x *.sh
```

Hình 34. Lệnh make file

Bước 3.9. Bắt đầu train (hết khoảng 84 tiếng với tập dataset 5 class với số max_batches=15000).

```

cd /content/gdrive/My\ Drive/darknet
./darknet detector train yolo.data cfg/yolov4-custom.cfg yolov4.conv.137 -dont_show -map

/content/gdrive/My Drive/darknet
CUDA-version: 11000 (11020), cuDNN: 7.6.5, GPU count: 1
OpenCV version: 3.2.0
Prepare additional network for mAP calculation...
0 : compute_capability = 750, cudnn_half = 0, GPU: Tesla T4
net.optimized_memory = 0
mini_batch = 1, batch = 16, time_steps = 1, train = 0
    layer   filters size/strd(dil)      input           output
    0 Create CUDA-stream - 0
Create cudnn-handle 0
conv    32      3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x   32 0.299 BF
    1 conv    64      3 x 3/ 2    416 x 416 x   32 ->  208 x 208 x   64 1.595 BF
    2 conv    64      1 x 1/ 1    208 x 208 x   64 ->  208 x 208 x   64 0.354 BF
    3 route  1                  ->  208 x 208 x   64
    4 conv    64      1 x 1/ 1    208 x 208 x   64 ->  208 x 208 x   64 0.354 BF
    5 conv    32      1 x 1/ 1    208 x 208 x   64 ->  208 x 208 x   32 0.177 BF
    6 conv    64      3 x 3/ 1    208 x 208 x   32 ->  208 x 208 x   64 1.595 BF
    7 Shortcut Layer: 4, wt = 0, wn = 0, outputs: 208 x 208 x   64 0.003 BF
    8 conv    64      1 x 1/ 1    208 x 208 x   64 ->  208 x 208 x   64 0.354 BF
    9 route  8 2                  ->  208 x 208 x 128
    10 conv   64      1 x 1/ 1    208 x 208 x 128 ->  208 x 208 x   64 0.709 BF
    11 conv   128     3 x 3/ 2    208 x 208 x   64 ->  104 x 104 x 128 1.595 BF
    12 conv   64      1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x   64 0.177 BF
    13 route 11                  ->  104 x 104 x 128
    14 conv   64      1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x   64 0.177 BF
    15 conv   64      1 x 1/ 1    104 x 104 x   64 ->  104 x 104 x   64 0.089 BF
    16 conv   64      3 x 3/ 1    104 x 104 x   64 ->  104 x 104 x   64 0.797 BF
    17 Shortcut Layer: 14, wt = 0, wn = 0, outputs: 104 x 104 x   64 0.001 BF
    18 conv   64      1 x 1/ 1    104 x 104 x   64 ->  104 x 104 x   64 0.089 BF
    19 conv   64      3 x 3/ 1    104 x 104 x   64 ->  104 x 104 x   64 0.797 BF
    20 Shortcut Layer: 17, wt = 0, wn = 0, outputs: 104 x 104 x   64 0.001 BF
    21 conv   64      1 x 1/ 1    104 x 104 x   64 ->  104 x 104 x   64 0.089 BF
    22 route 21 12                  ->  104 x 104 x 128
    23 conv   128     1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x 128 0.354 BF
    24 conv   256     3 x 3/ 2    104 x 104 x 128 ->  52 x 52 x 256 1.595 BF
    25 conv   128     1 x 1/ 1    52 x 52 x 256 ->  52 x 52 x 128 0.177 BF
    26 route 24                  ->  52 x 52 x 256
    27 conv   128     1 x 1/ 1    52 x 52 x 256 ->  52 x 52 x 128 0.177 BF
    28 conv   128     1 x 1/ 1    52 x 52 x 128 ->  52 x 52 x 128 0.089 BF
    29 conv   128     3 x 3/ 1    52 x 52 x 128 ->  52 x 52 x 128 0.707 BF

```

Hình 35. Output training model

Kết quả sau khi train ta có một model :



Ta thu được một file nhị phân với đuôi là *.weights.

yolov4-custom_last.weights

Hình 36. File model

3. Tính độ chính xác cho model:

Đối với mỗi model Object detection sau khi đào tạo, cần có những thang điểm để đánh giá sự chính xác của nó. Theo bản thân chúng em biết, hiện tại chúng ta đánh giá một model dựa trên: loss function, IOU avg, mAP, đánh giá trực quan, và nhiều bài báo cũng như các trang web lớn thường sử dụng mAP như là thước đo chính.

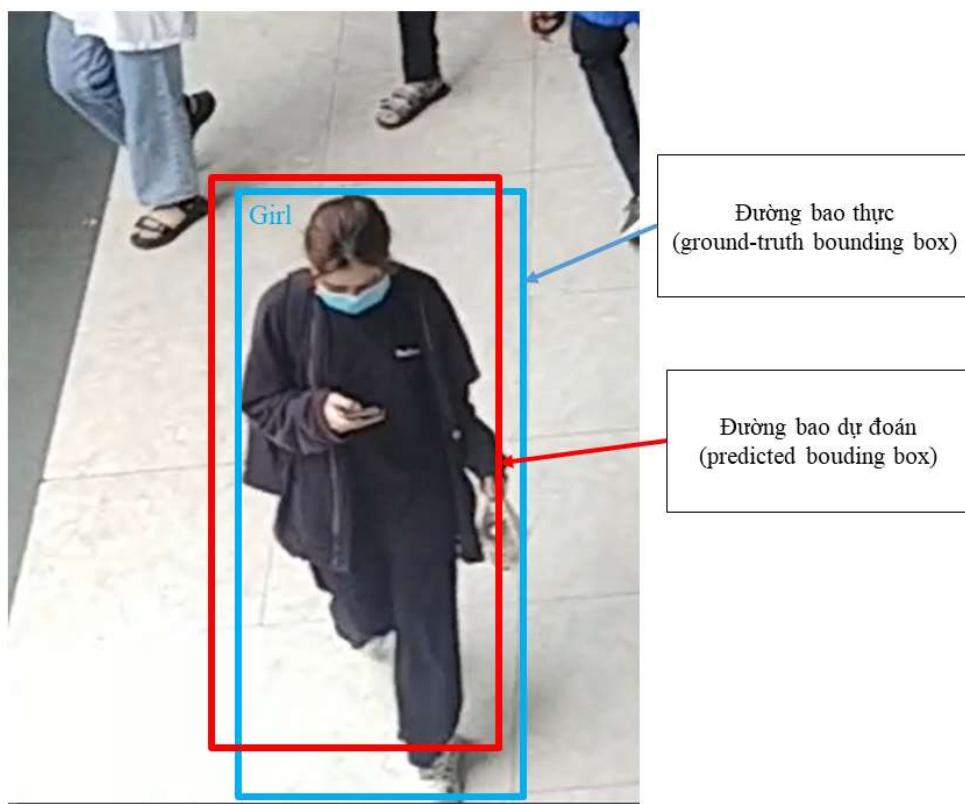
- IOU(Intersection over union)

Intersection over Union là chỉ số đánh giá được sử dụng để đo độ chính xác của phát hiện đối tượng trên tập dữ liệu cụ thể. Chỉ số này thường được gặp trong các Object Detection Challenge. IOU thường được đánh giá hiệu năng của các bộ phát hiện đối tượng như HOG Linear SVM và mạng nơ ron tích chập (R-CNN, FastR-CNN, YOLO,...).

Để áp dụng được IoU để đánh giá cần:

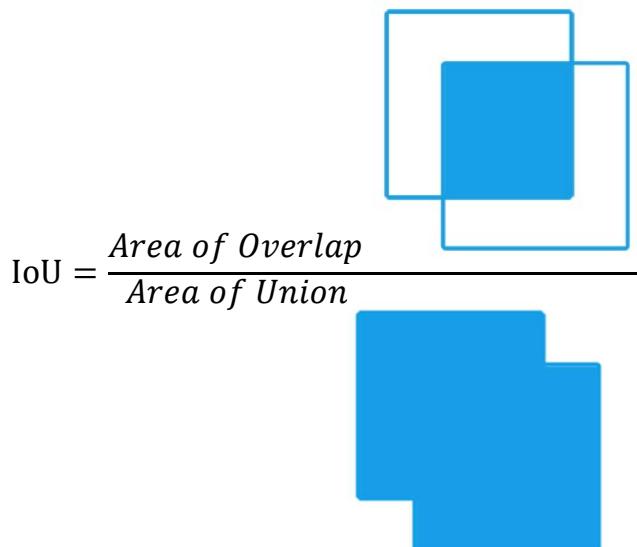
- Đường bao thực (ground-truth bounding box): là đường bao mà chúng ta gán cho vật thể bằng labelImg tool.
- Đường bao dự đoán (predicted bounding box): là đường bao chúng ta sử dụng file Weights sau khi đào tạo để nhận dạng.

Dưới đây là ví dụ về đường bao thực và đường bao được dự đoán. Đường bao được dự đoán được vẽ bằng màu vàng, trong khi đó đường bao thực được vẽ bằng màu xanh lá. Mục tiêu ta là tính toán IOU (Intersection over Union) giữa hai đường bao.



Hình 37. Minh họa cách tính IOU

Tỷ lệ này là IOU là tỉ lệ giữa độ lồng nhau giữa hai đường bao (thường là đường bao dự đoán và đường bao thực) để nhằm xác định hai khung hình có bị đè chòng lên nhau không. Tỷ lệ này được tính dựa trên phần diện tích giao nhau giữa 2 đường bao với phần tổng diện tích giao nhau và không giao nhau giữa chúng.



Hình 38. Minh họa IoU

Các tiêu chí được dùng để đánh giá:

Đối tượng được nhận dạng đúng với tỉ lệ IOU> 0.5 (True positive : TP).



Hình 39. True Positive

Đối tượng được nhận dạng sai với tỉ lệ IOU < 0.5 (False positive : FP).



Hình 40. False Positive

Đối tượng không được nhận dạng (False negative: FN).



Hình 41. False Negative

▪ **Precision và Recall** (chúng em để nguyên bản tiếng anh vì khó dịch sang tiếng Việt). Dự đoán (Precision) đo lường mức độ chính xác là dự đoán của mô hình tức là tỷ lệ phần trăm dự đoán của mô hình là chính xác.

“Recall” đo lường như thế nào tốt mô hình tìm thấy tất cả các mẫu tích cực. Ví dụ: chúng ta có thể tìm thấy 80% các trường hợp tích cực có thể có trong các dự đoán K hàng đầu của mô hình.

Mô tả toán học của precision và recall:

		Thực tế	
		Positive	Negative
Mô hình phân loại	Positive	TP	FP
	Negative	FN	TN

$$Precision = \frac{TP}{TP + FP} = \frac{\text{Số dự đoán chính xác}}{\text{Tổng số lần dự đoán}}$$

$$Recall = \frac{TP}{TP + FN} = \frac{\text{Số lần dự đoán chính xác}}{\text{Số lần nhận dạng đúng có thể có}}$$

▪ **AP và mAP:**

Từ precision và recall đã được định nghĩa ở trên chúng ta cũng có thể đánh giá mô hình dựa trên việc thay đổi một ngưỡng và quan sát giá trị của Precision và Recall. Khái niệm Area Under the Curve (AUC) cũng được định nghĩa tương tự. Với Precision-Recall Curve, AUC còn có một tên khác là Average precision (AP).

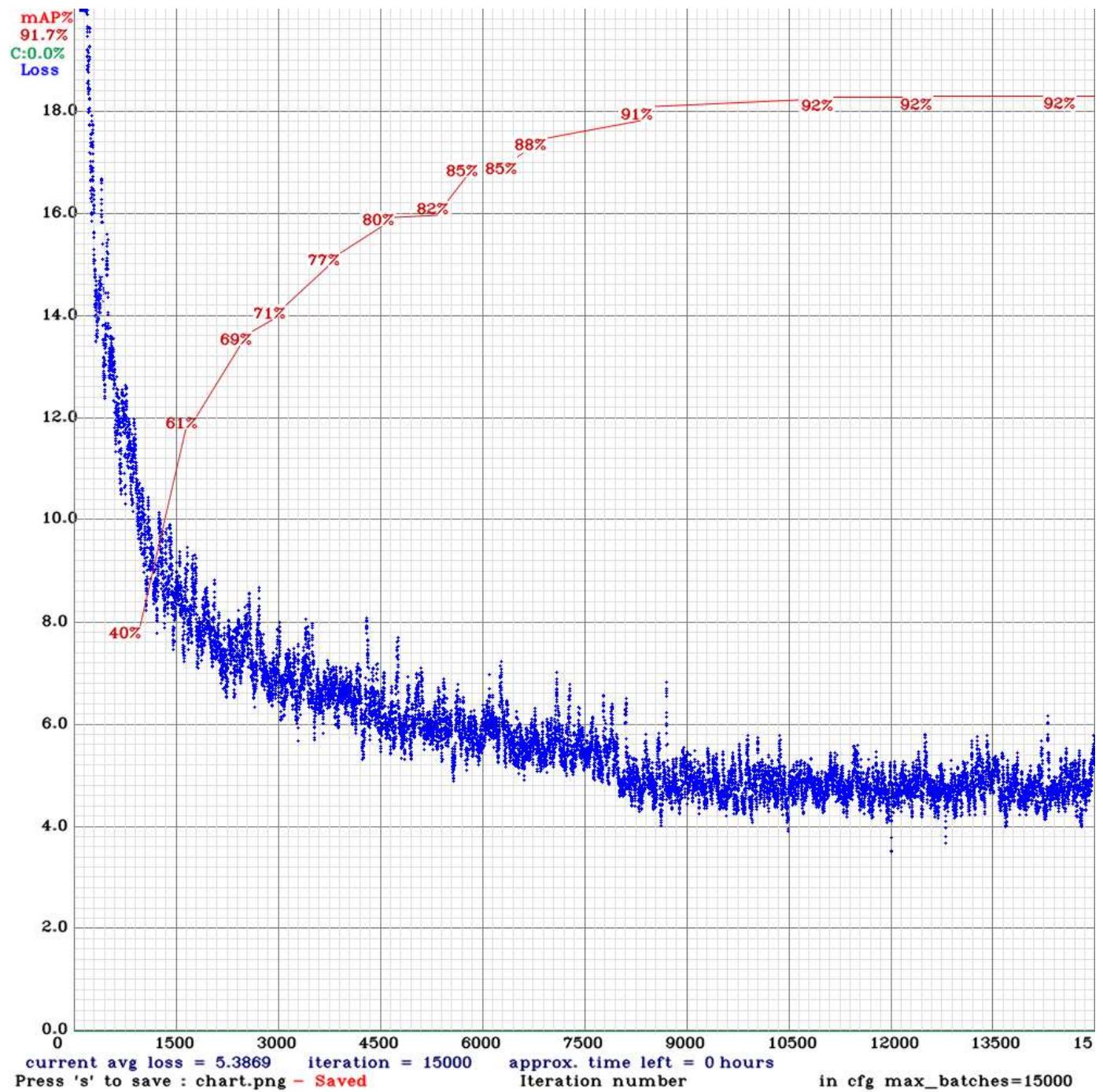
Giả sử có N ngưỡng để tính precision và recall, với mỗi ngưỡng cho một cặp giá trị precision, recall là $Pn, n=1,2,\dots,N$. Precision-Recall curve được vẽ bằng cách vẽ từng điểm có tọa độ (Pn) trên trục tọa độ và nối chúng với nhau. AP được xác định bằng:

$$AP = \sum (Rn - Rn-1) Pnn$$

Và mAP là trung bình của AP được tính cho tất cả các lớp.

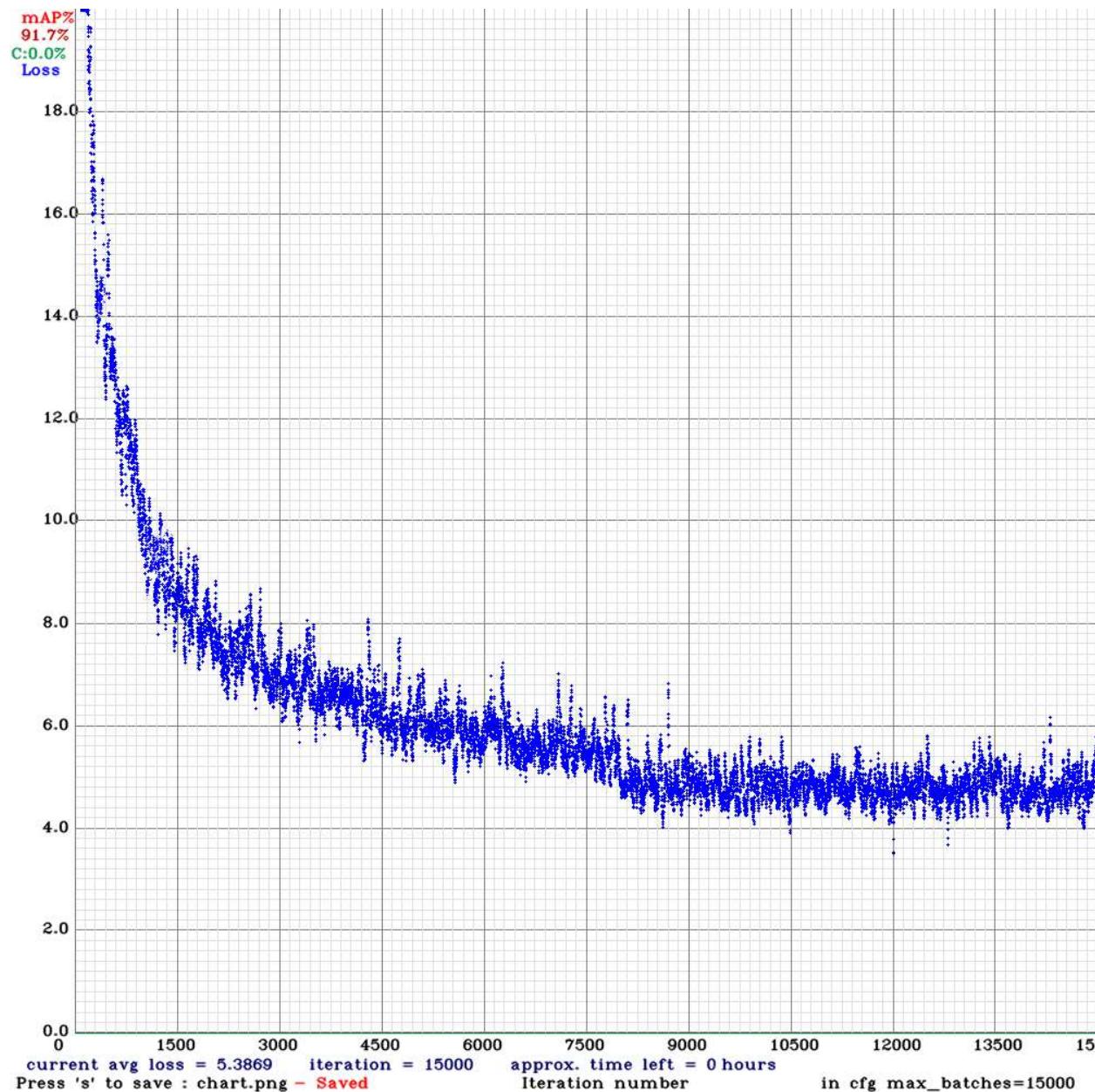
(<https://devai.info/2020/12/17/tim-hieu-mapmean-average-precision-danh-gia-mo-hinh-object-detection-su-dung-yolov4/>)

Biểu đồ tổng quát:



Hình 42. Biểu đồ quá trình train có output mAP

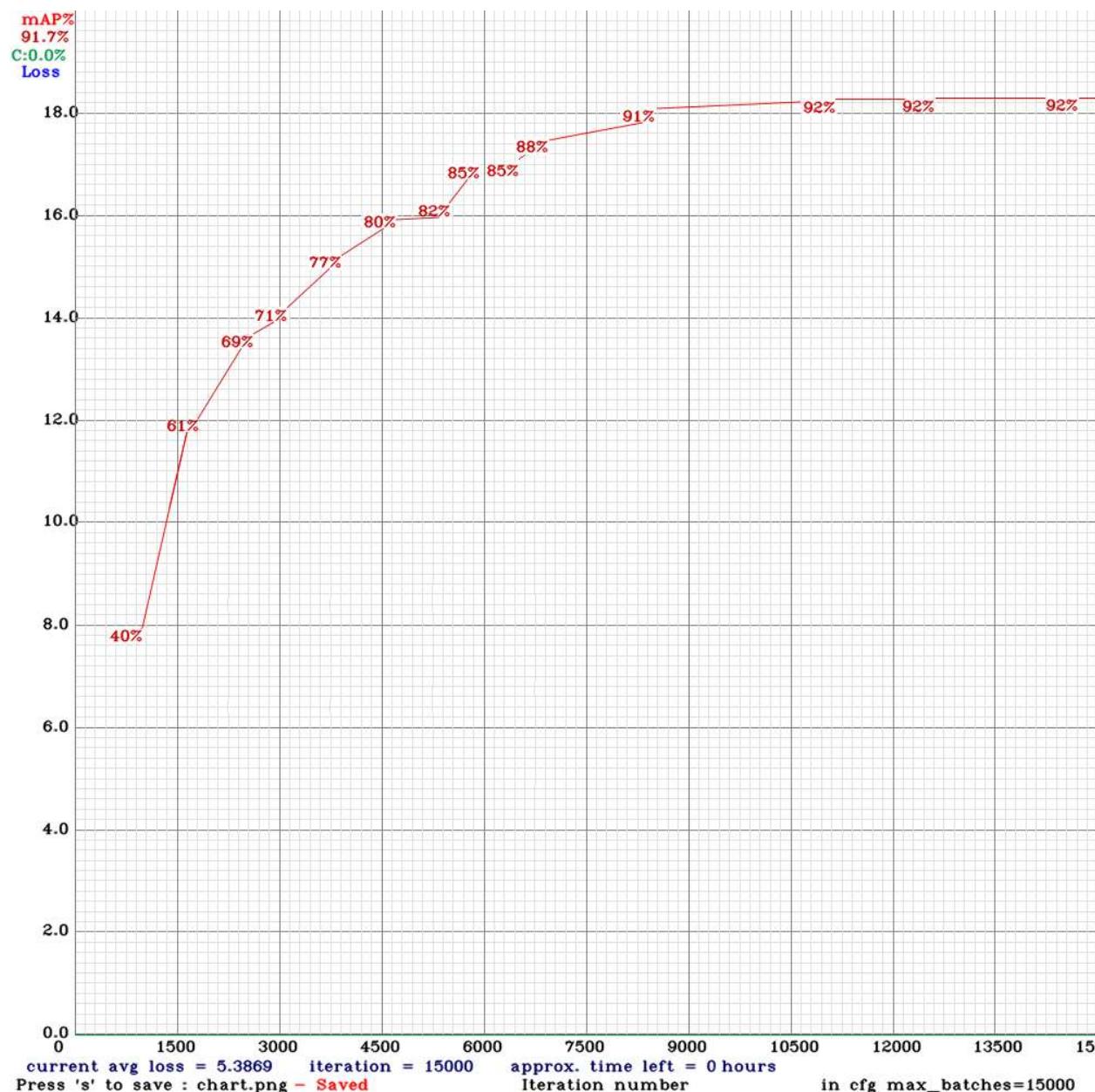
Biểu đồ loss của model:



Hình 43. Char.png

Từ biểu đồ trên ta có thể thấy quá trình train mất 15000 vòng lặp, trong quá trình train ta có thể thấy biểu đồ giảm loss mạnh từ vòng lặp đầu tiên tới vòng lặp khoảng 8000. Từ 8000 trở đi ta thấy loss của model giao động trong khoảng lớn hơn 4 và nhỏ hơn 6, loss thấp nhất mà model đạt được nhỏ hơn 4. Nhưng quá trình train kết thúc thì loss đạt 5.3869.

Biểu đồ mAP của yolov4:



Hình 44 mAP yolov4

Từ biểu đồ trên ta có thể thấy độ chính xác của model mAP: 40% tại thời điểm vòng lặp thứ 1000 với avg loss vào khoảng 8.2 và kết thúc với mAP: 91.7% tại max_batches=15000 với avg loss 5.3869.

Đánh giá chi tiết mAP của từng class:

```
(next mAP calculation at 15142 iterations)
Last accuracy mAP@0.5 = 91.91 %, best = 91.91 %
15000: 4.663130, 5.386854 avg loss, 0.000010 rate, 7.464029 seconds, 960000 images, 0.424456 hours left
Resizing to initial size: 416 x 416 try to allocate additional workspace_size = 52.43 MB
CUDA allocate done!

calculation mAP (mean average precision)...
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
1228
detections_count = 29810, unique_truth_count = 8704
class_id = 0, name = person, ap = 89.97%          (TP = 4257, FP = 1250)
class_id = 1, name = man, ap = 93.74%            (TP = 1327, FP = 288)
class_id = 2, name = girl, ap = 90.41%           (TP = 1501, FP = 454)
class_id = 3, name = elder, ap = 88.66%          (TP = 171, FP = 61)
class_id = 4, name = children, ap = 95.69%        (TP = 455, FP = 69)

for conf_thresh = 0.25, precision = 0.78, recall = 0.89, F1-score = 0.83
for conf_thresh = 0.25, TP = 7711, FP = 2122, FN = 993, average IoU = 62.99 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.916951, or 91.70 %
Total Detection Time: 182 Seconds

Set -points flag:
`-points 101` for MS COCO
`-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
`-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

mean_average_precision (mAP@0.5) = 0.916951
Saving weights to backup/yolov4-custom_last.weights
Saving weights to backup/yolov4-custom_final.weights
If you want to train from the beginning, then use flag in the end of training command: -clear
```

Hình 45. mAP từng class

mAP của các class:

person = 89.97%

children = 95.69%

man = 93.74%

average IoU = 62.9%

girl = 90.41%

IoU threshold = 50%

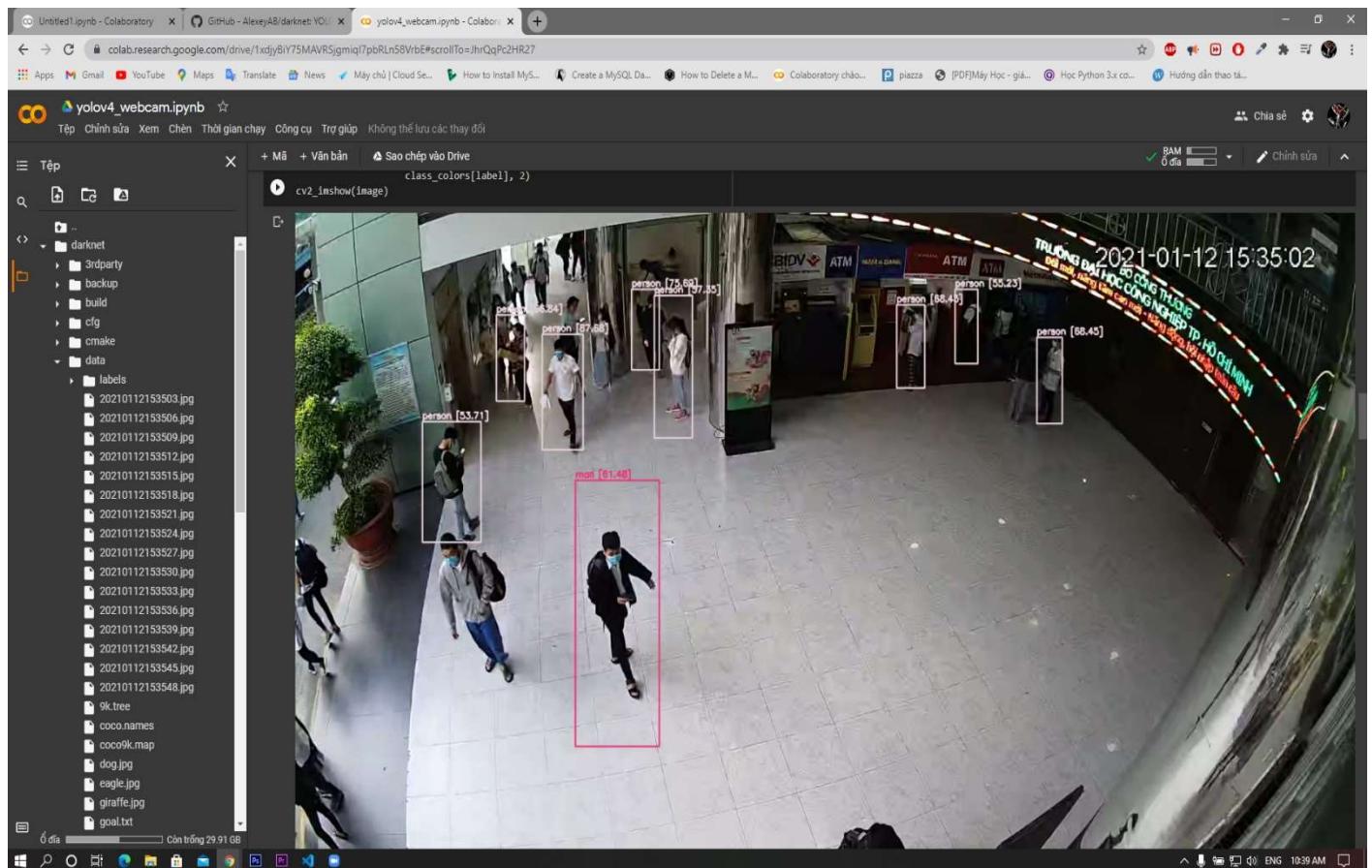
elder = 88.66%

mAP@0.50 = 0.916951 => 91.70

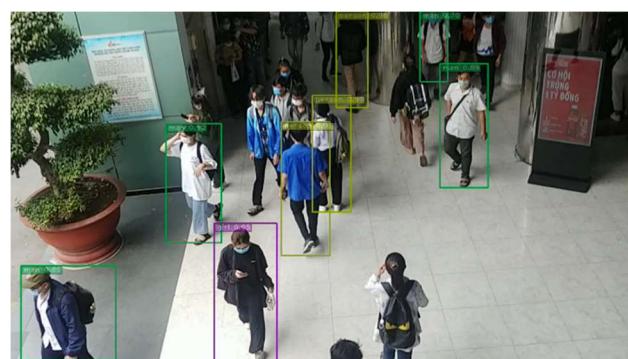
Chạy Demo với tập test.

Tiến hành test model bằng colab với tập test đã được tạo sẵn.

Kết quả:



Hình 46. Kết quả test bằng colab



Hình 47. Hình detect IUH



Hình 48. Hình detect IUH



Hình 49. Hình detect IUH



Hình 50. Hình detect Gigamall

Hình 47 kết quả detect tại IUH và hình 48 kết quả detect tại gigamall. Từ hai kết quả trên ta có thể thấy độ chính xác của model được đánh giá ở mức tương đối và bị ảnh hưởng bởi nhiều yếu tố như vị trí đặt, độ sáng và độ phân giải của camera. Đó cũng là những nhược điểm của model cũng như nơi triển khai.

4. Bắt đầu chuyển model yolov4 sang tensorRT:

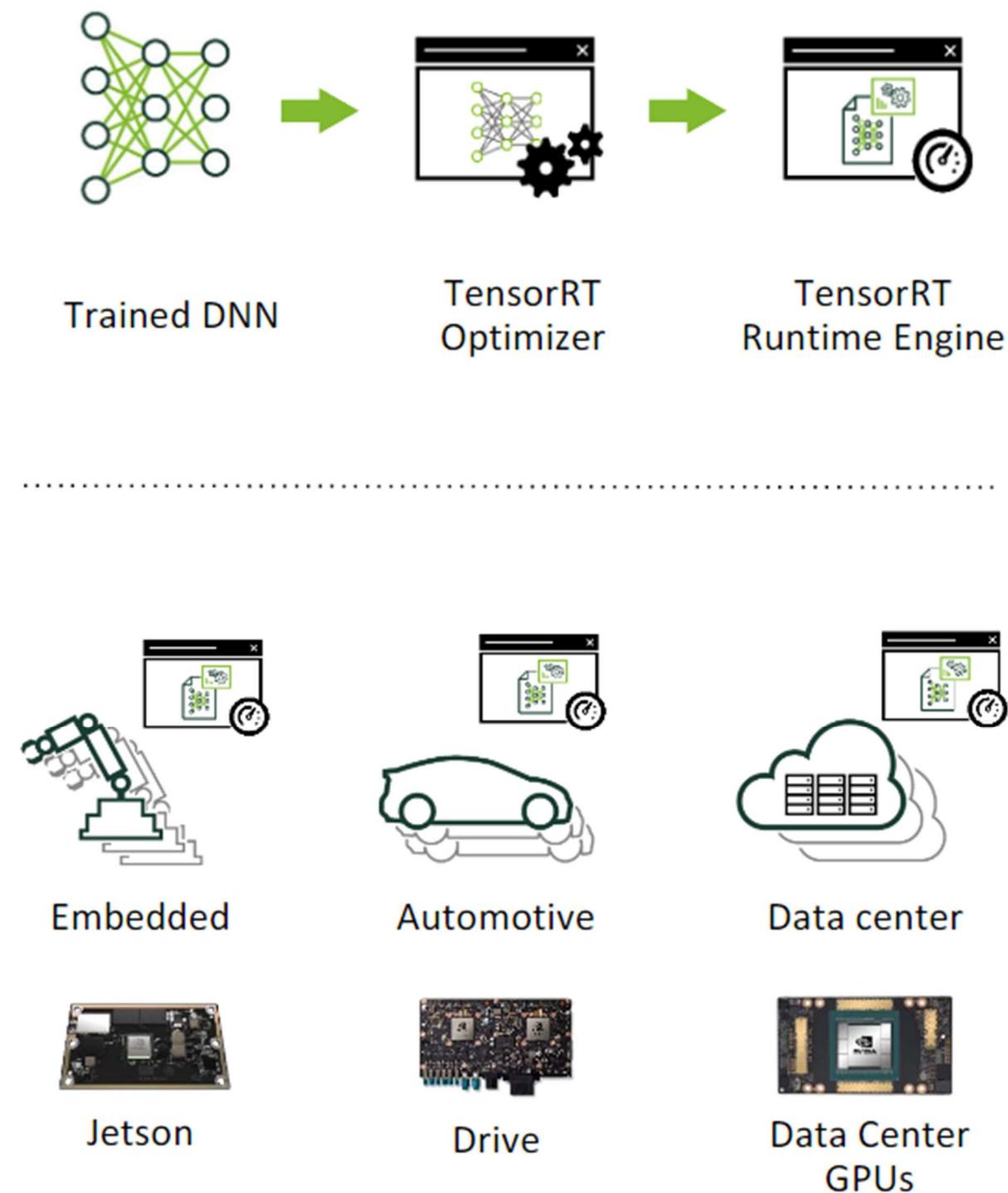
Công việc này giúp model vận hành với tốc độ nhanh hơn của thẻ với model chưa được chuyển đổi thì ta thu được fps vào khoảng 0.89~1 với (width = 416, height = 416) và kết quả sau khi chuyển đổi ta thu được fps khoảng 4.78~4.90 với cùng width & height. Nhưng bù lại các tham số trong mạng đã bị làm tròn nên độ chính xác của model giảm nhưng không đáng kể.

TensorRT là gì?

Cốt lõi của NVIDIA® TensorRT™ là thư viện C++ hỗ trợ suy luận hiệu suất cao trên các đơn vị xử lý đồ họa NVIDIA (GPU). Nó được thiết kế để hoạt động theo kiểu bổ sung với các khung đào tạo như TensorFlow, PyTorch, MXNet, v.v. Nó tập trung đặc biệt vào việc chạy một mạng đã được đào tạo một cách nhanh chóng và hiệu quả trên GPU. Một số khuôn khổ đào tạo như TensorFlow đã tích hợp TensorRT để nó có thể được sử dụng để tăng tốc suy luận trong khuôn khổ. Ngoài ra, TensorRT có thể được sử dụng như một thư viện trong một ứng dụng người dùng. Nó bao gồm trình phân tích cú pháp ONNX và các API C++ và Python để xây dựng các mô hình theo chương trình.

TensorRT tối ưu hóa mạng bằng cách kết hợp các lớp và tối ưu hóa lựa chọn hạt nhân để cải thiện độ trễ, thông lượng, hiệu quả sử dụng năng lượng và mức tiêu thụ bộ nhớ. Nếu ứng dụng chỉ định, nó cũng sẽ tối ưu hóa mạng để chạy với độ chính xác thấp hơn, tăng hiệu suất hơn nữa và giảm yêu cầu bộ nhớ.

Các API TensorRT bao gồm việc triển khai đối với hầu hết các lớp học sâu chung. Để biết thêm thông tin về các lớp, hãy xem Lớp TensorRT. Bạn cũng có thể sử dụng API plugin C++ hoặc API plugin Python để cung cấp triển khai cho các lớp không thường xuyên được sử dụng hoặc các lớp đối mới hơn không được TensorRT hỗ trợ.



Hình 51. TensorRT là một công cụ tối ưu hóa suy luận mạng nơ-ron và công cụ thời gian chạy hiệu suất cao để triển khai sản xuất.

Loi ích của TensorRT:

Sau khi mạng nơ-ron được đào tạo, TensorRT cho phép mạng được nén, tối ưu hóa và triển khai dưới dạng thời gian chạy mà không cần khung làm việc.

TensorRT kết hợp các lớp, tối ưu hóa lựa chọn hạt nhân và cũng thực hiện chuẩn hóa và chuyển đổi sang phép toán ma trận được tối ưu hóa tùy thuộc vào độ chính xác được chỉ định (FP32, FP16 hoặc INT8) để cải thiện độ trễ, thông lượng và hiệu quả.

Đối với suy luận học sâu, có 5 yếu tố quan trọng được sử dụng để đo lường phân mềm.

Thông lượng:

Khối lượng đầu ra trong một khoảng thời gian nhất định. Thường được đo bằng suy luận / giây hoặc mẫu / giây, thông lượng trên mỗi máy chủ rất quan trọng đối với việc mở rộng quy mô hiệu quả về chi phí trong các trung tâm dữ liệu.

Hiệu quả:

Lượng thông lượng phân phối trên một đơn vị công suất, thường được biểu thị bằng hiệu suất / watt. Hiệu quả là một yếu tố quan trọng khác để mở rộng quy mô trung tâm dữ liệu hiệu quả về chi phí, vì máy chủ, giá đỡ máy chủ và toàn bộ trung tâm dữ liệu phải hoạt động trong phạm vi ngân sách điện cố định.

Độ trễ:

Thời gian để thực hiện một suy luận, thường được đo bằng mili giây. Độ trễ thấp là yếu tố quan trọng để cung cấp các dịch vụ dựa trên suy luận thời gian thực, đang phát triển nhanh chóng.

Sự chính xác:

Khả năng cung cấp câu trả lời chính xác của mạng nơ-ron được đào tạo.

Sử dụng bộ nhớ:

Máy chủ và bộ nhớ thiết bị cần được dự trữ để thực hiện suy luận trên mạng phụ thuộc vào các thuật toán được sử dụng. Điều này hạn chế những mạng nào và những tổ hợp mạng nào có thể chạy trên một nền tảng suy luận nhất định. Điều này đặc biệt quan trọng đối với các hệ thống cần nhiều mạng và tài nguyên bộ nhớ bị hạn chế - chẳng hạn như các mạng phát hiện đa lớp xếp tầng được sử dụng trong phân tích video thông minh và hệ thống lái xe tự động nhiều camera, đa mạng.

Các lựa chọn thay thế để sử dụng TensorRT bao gồm:

- Sử dụng chính khung đào tạo để thực hiện suy luận.
- Viết một ứng dụng tùy chỉnh được thiết kế đặc biệt để thực thi mạng bằng cách sử dụng các thư viện cấp thấp và các phép toán.

Việc sử dụng khung đào tạo để thực hiện suy luận rất dễ dàng, nhưng có xu hướng dẫn đến hiệu suất thấp hơn nhiều trên một GPU nhất định so với khả năng có thể có với một giải pháp tối ưu hóa như TensorRT. Khung đào tạo có xu hướng triển khai mã mục đích chung hơn, nhấn mạnh tính tổng quát và khi chúng được tối ưu hóa, các tối ưu hóa có xu hướng tập trung vào đào tạo hiệu quả.

Hiệu quả cao hơn có thể đạt được bằng cách viết một ứng dụng tùy chỉnh chỉ để thực thi mạng nơ-ron, tuy nhiên, nó có thể khá tốn công sức và đòi hỏi kiến thức chuyên môn để đạt được mức hiệu suất cao trên GPU hiện đại. Hơn nữa, các tối ưu hóa hoạt động trên một GPU có thể không chuyển hoàn toàn sang các GPU khác trong cùng một họ và mỗi thẻ hệ GPU có thể giới thiệu các khả năng mới chỉ có thể được tận dụng bằng cách viết mã mới.

TensorRT giải quyết những vấn đề này bằng cách kết hợp API cấp cao giúp tóm tắt các chi tiết phần cứng cụ thể và triển khai tối ưu hóa suy luận cho thông lượng cao, độ trễ thấp và dấu chân bộ nhớ thiết bị thấp.

Ai có thể hưởng lợi từ TensorRT?

TensorRT được thiết kế để sử dụng bởi các kỹ sư chịu trách nhiệm xây dựng các tính năng và ứng dụng dựa trên các mô hình học sâu mới hoặc hiện có hoặc triển khai các mô hình vào môi trường sản xuất. Các triển khai này có thể vào các máy chủ trong trung tâm dữ liệu hoặc đám mây, trong một thiết bị nhúng, rô bốt hoặc phương tiện hoặc phần mềm ứng dụng sẽ chạy trên các máy trạm của bạn.

TensorRT đã được sử dụng thành công trong nhiều tình huống, bao gồm:

Rô bót:

Các công ty bán robot sử dụng TensorRT để chạy các loại mô hình thị giác máy tính khác nhau nhằm tự động dẫn đường cho hệ thống máy bay không người lái bay trong môi trường động.

Xe tự hành:

TensorRT được sử dụng để tăng cường thị giác máy tính trong các sản phẩm NVIDIA Drive.

Máy tính Khoa học và Kỹ thuật:

Một gói tính toán kỹ thuật phổ biến nhúng TensorRT để cho phép thực thi thông lượng cao của các mô hình mạng nơ-ron.

Khung triển khai và đào tạo Deep Learning:

TensorRT được bao gồm trong một số khung công tác học sâu phổ biến bao gồm TensorFlow và MXNet. Để biết ghi chú phát hành vùng chứa TensorFlow và MXNet, hãy xem ghi chú phát hành TensorFlow và Ghi chú phát hành MXNet.

Phân tích video:

TensorRT được sử dụng trong sản phẩm DeepStream của NVIDIA để cung cấp các giải pháp phân tích video tinh vi ở cả lợi thế với nguồn cấp dữ liệu từ 1 đến 16 camera và trong trung tâm dữ liệu nơi hàng trăm hoặc thậm chí hàng nghìn nguồn cấp dữ liệu video có thể kết hợp với nhau.

Nhận dạng giọng nói tự động:

TensorRT được sử dụng để hỗ trợ nhận dạng giọng nói trên thiết bị để bàn / máy tính để bàn nhỏ. Từ vựng hạn chế được hỗ trợ trên thiết bị với hệ thống nhận dạng giọng nói từ vựng lớn hơn có sẵn trên đám mây.

(Nguồn: <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html>)

Chuyển đổi yolov4 to tensorrt 7.2:

- + Cài đặt pycuda.
- + Cài đặt onnx==1.4.1.
- + Chuyển đến thư mục con "plugins /" và xây dựng plugin "yolo_layer". Khi hoàn tất, một "libyolo_layer.so" sẽ được tạo.
- + Chuyển model yolov4 qua model ONNX là model trung gian.
- + Chuyển model ONNX sang model Tensorrt 7.2 (yêu cầu cần có thư viện Tensorflow) Git hướng dẫn : (https://github.com/jkjung-avt/tensorrt_demos)

Model yolov4 sau khi chuyển sang tensorrt:

Name	Date modified	Type	Size
yolov4-custom.trt	5/27/2021 9:39 PM	TRT File	288,524 KB

Hình 52. Model tensorrt

5. Triển khai thực tế và deploy lên web.

Giờ bắt đầu đưa model vào thực tế dự án.

Các bước triển khai thực tế.

Ý tưởng:

Hệ thống chạy trên jetson nano và gửi giá trị sang web để chiếu video quảng cáo phù hợp với đối tượng.

- + Xây dựng hệ thống nhận diện với model trên. Ta sử dụng Python.
- + Xây dựng phương thức truyền dữ liệu trong hệ thống. Ta sử dụng socket python.
- + Xây dựng một web. Ta sử dụng Nodejs.

Giới thiệu về NodeJS

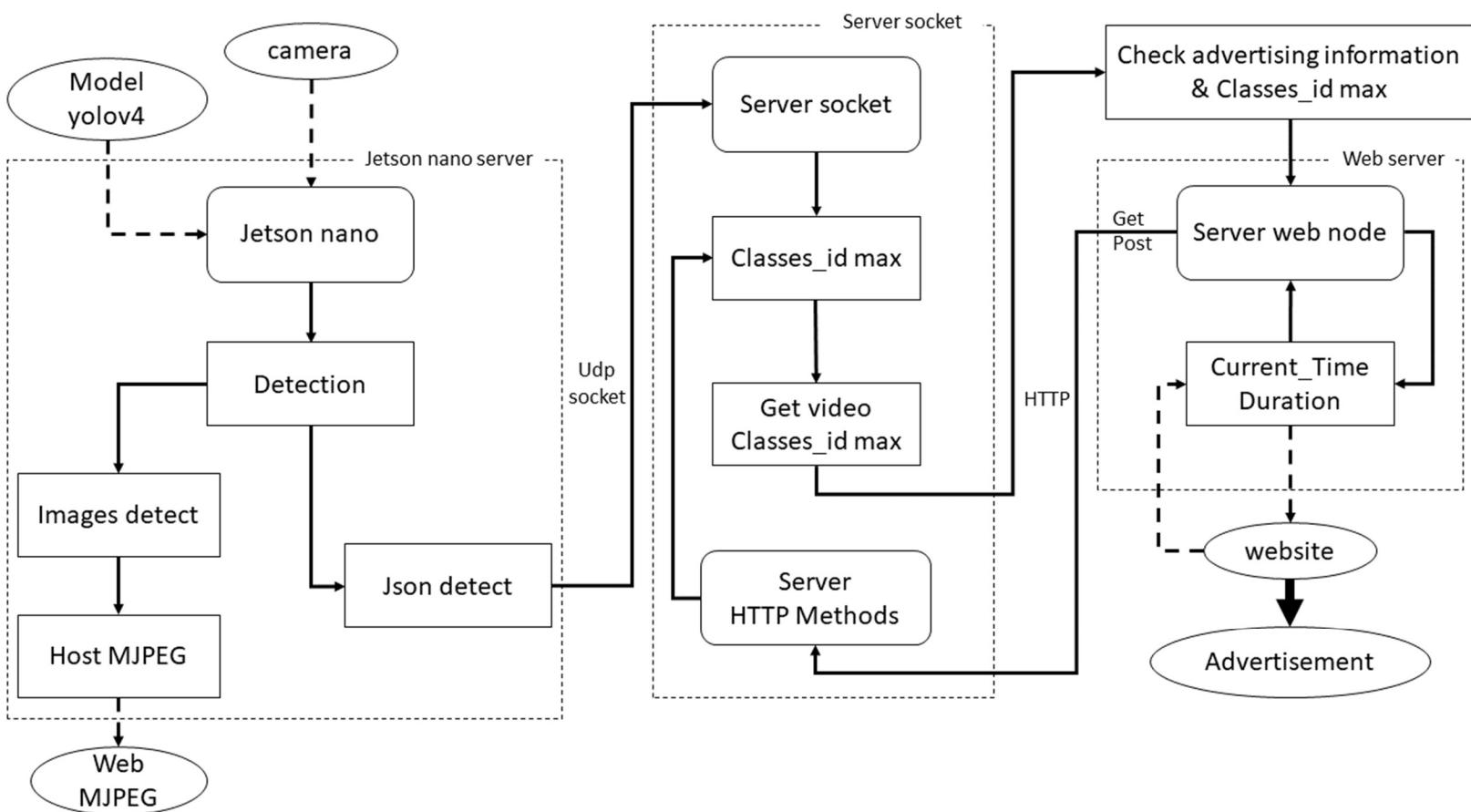
NodeJS là một nền tảng được xây dựng trên V8 JavaScript Engine – trình thông dịch thực thi mã JavaScript, giúp xây dựng các ứng dụng web một cách đơn giản và dễ dàng mở rộng.

NodeJS được phát triển bởi Ryan Dahl vào năm 2009 và có thể chạy trên nhiều hệ điều hành khác nhau: OS X, Microsoft Windows, Linux.

Lý do nên học NodeJS là gì?

- ❖ NodeJS được viết bằng JavaScript với cộng đồng người dùng lớn mạnh. Nếu bạn cần hỗ trợ gì về NodeJS, sẽ nhanh chóng có người hỗ trợ bạn.
- ❖ Tốc độ xử lý nhanh. Nhờ cơ chế xử lý bất đồng bộ (non-blocking), NodeJS có thể xử lý hàng ngàn kết nối cùng lúc mà không gặp bất cứ khó khăn nào.
- ❖ Dễ dàng mở rộng. Nếu bạn có nhu cầu phát triển website thì tính năng dễ dàng mở rộng của NodeJS là một lợi thế cực kỳ quan trọng.

Sơ đồ hoạt động của jetson server và website node :



Hình 53. Sơ đồ hoạt động của hệ thống trên jetson nano

Jetson nano server:

Bước 1: Input jetson nano

- Khi hệ thống khởi động sẽ tự động input model yolov4 đã được convert sang dạng tensorrt vào jetson nano.
- Khởi động camera và lấy dữ liệu từ camera.

Bước 2: Công việc trong jetson nano.

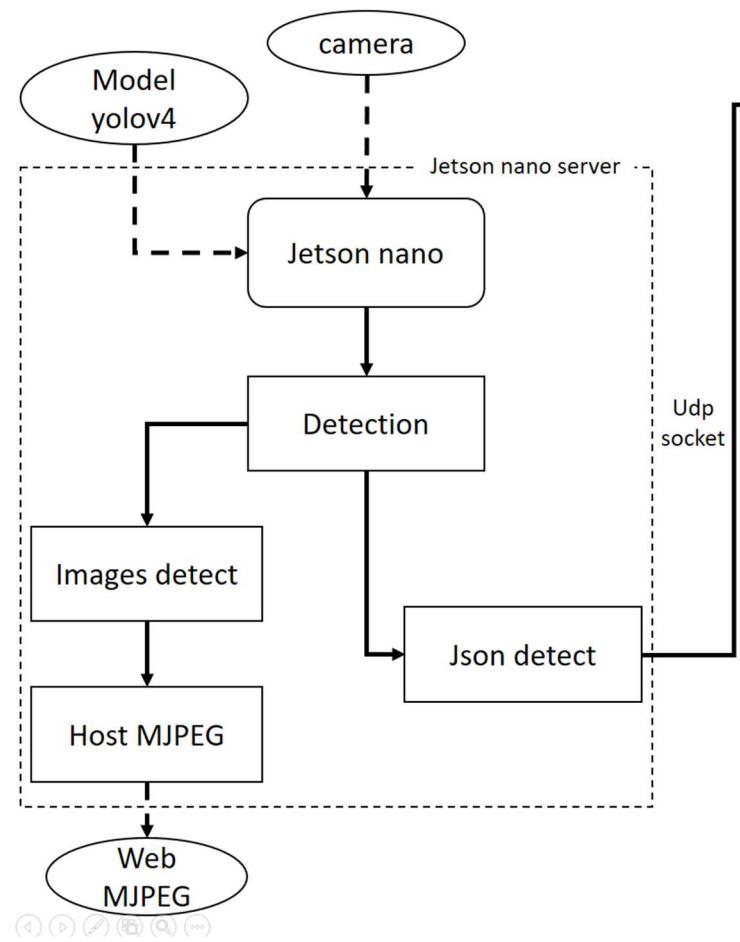
- Dùng thư viện openCV để đọc dữ liệu từ camera sau đó cắt từng frame chuyển vào model để nhận diện.
- Jetson nano thông qua model nhận diện được các đối tượng rồi trả về.

Bước 3: Host stream mjpeg.

- Trả ra các mjpeg.
- Truyền các mjpeg vào server rồi gửi lên web khi có yêu cầu.

Bước 4: Host json.

- Trả ra file có obj ở dạng chuỗi json.



Hình 54. Sơ đồ jetson nano server

Server socket:

Bước 1: Server socket.

- Tạo kết nối tới jetson nano thông qua socket giao thức (udp socket).
- Nhận chuỗi json từ jetson nano.
- Tạo một danh sách lưu các chuỗi json.
- Thực hiện bước 3 nếu chưa có server node kết nối.

Bước 2: Classes_id max.

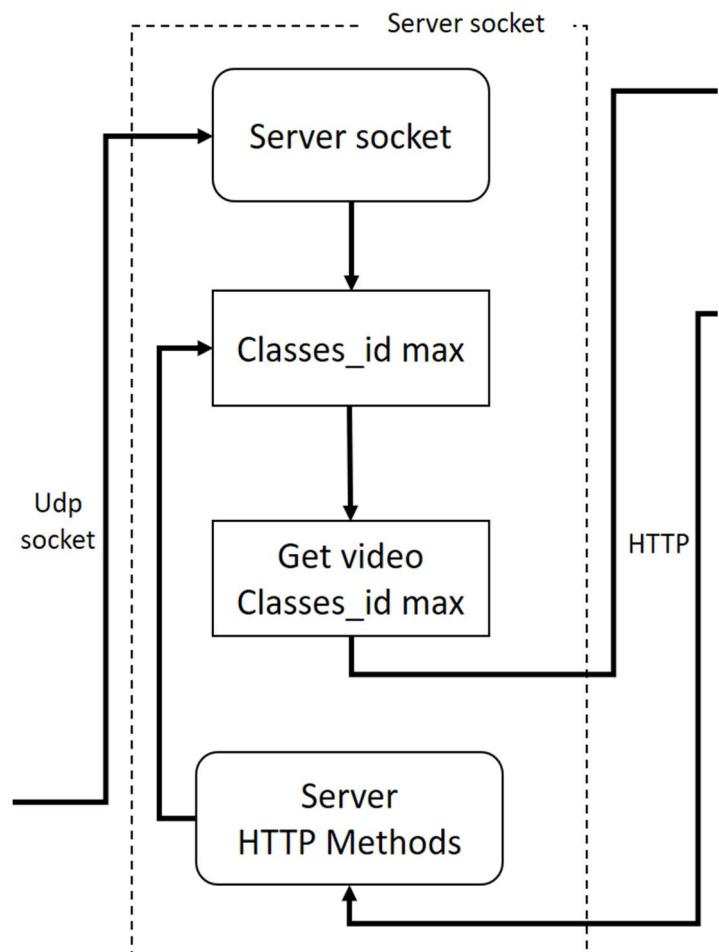
- Đợi server node trả về lệnh get.
- Khi có lệnh get sẽ lấy chuỗi json trong 20s bắt đầu từ lúc có lệnh get sau đó thực hiện bước 3 (chỉ hoạt động khi node truy cập lần đầu).
- Đợi server node trả về lệnh post thời gian còn lại của video quảng cáo. Sau đó xuống bước 3.

Bước 3: Get video.

- Tính tổng số obj của từng class sau tìm ra class có tổng số lớn nhất trừ class person.
- Nếu các class đều bằng không thì trả về class person.
- Tìm video quảng cáo tương ứng với class có tổng số lớn nhất.
- Nếu các class đều bằng không thì sẽ lấy video ngẫu nhiên.

Bước 4: Server HTTP Methods.

- Nhận giá trị post từ server node
- Tạo thời gian bắt đầu đếm số obj trong các class.
- Quay lại bước 2.



Hình 55. Sơ đồ server socket python

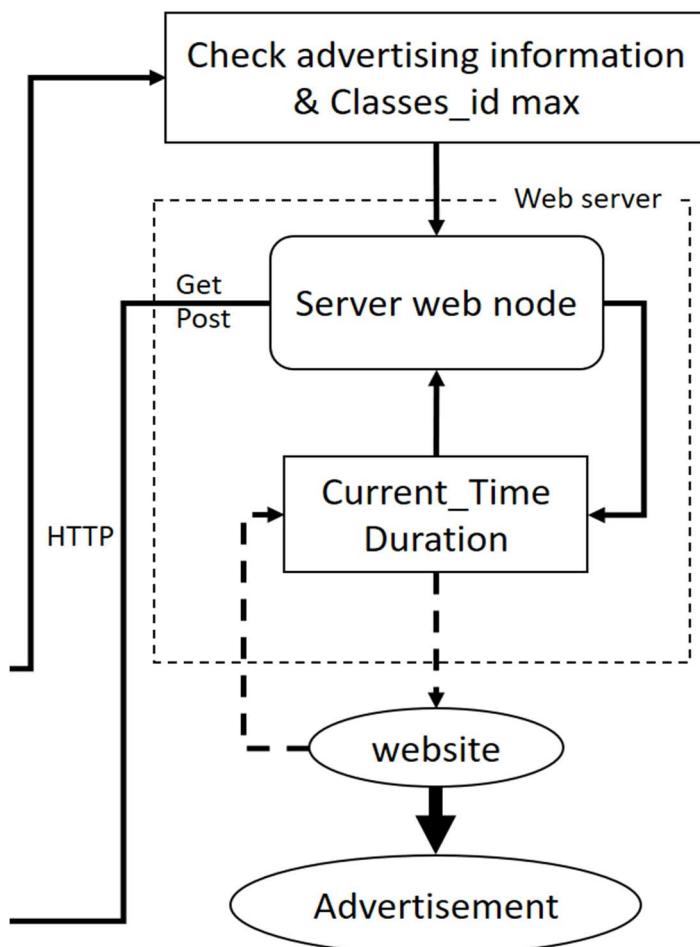
Web server (nodejs):

Bước 1: Server web node

- Get server socket lấy đường dẫn video quảng cáo.
- Post thời gian còn lại của video quảng cáo.

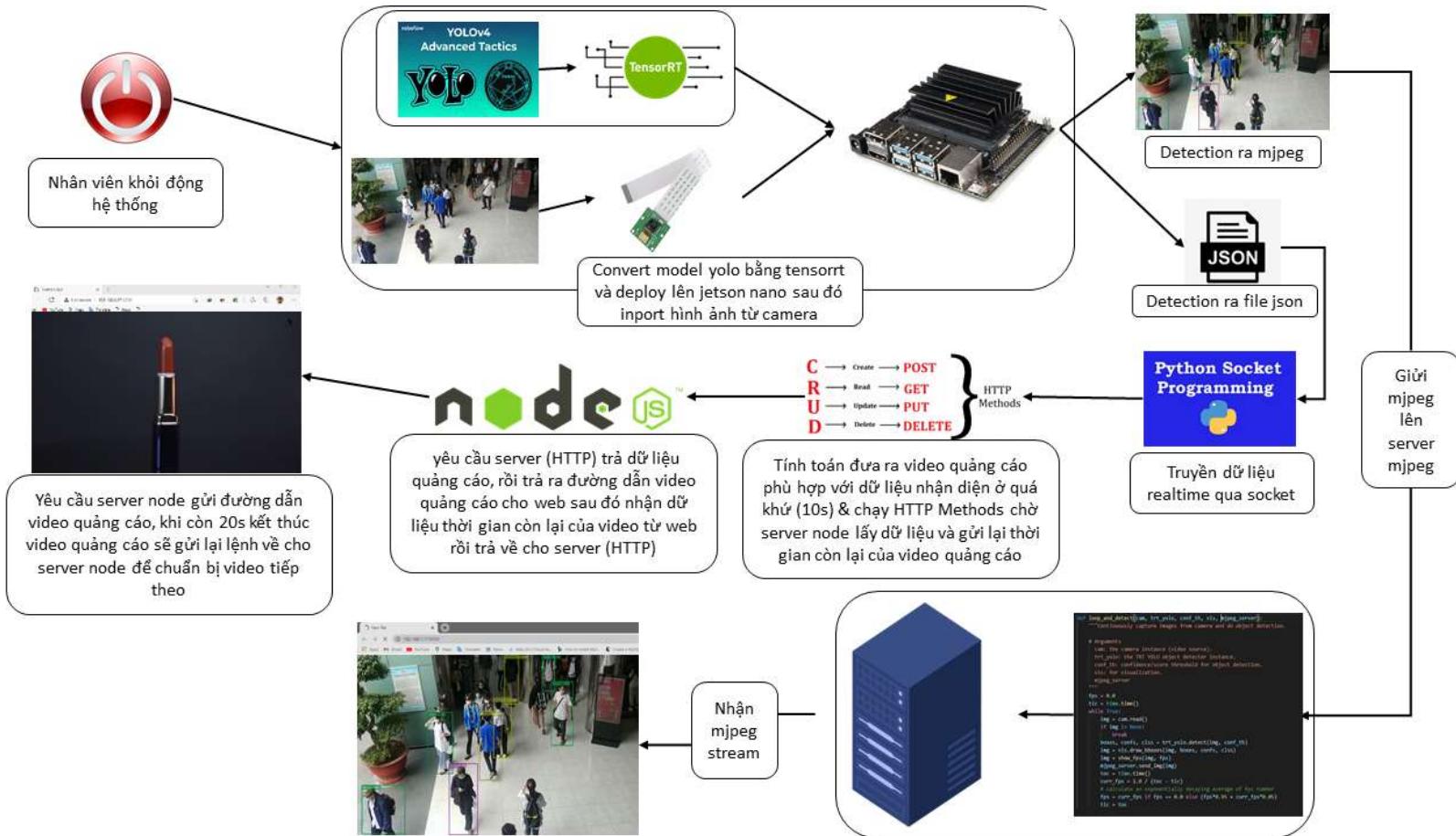
Bước 2: Lấy thời lượng còn lại của video.

- Lấy video và truyền ra website.
- Kiểm tra nếu thời lượng video còn lại 20s thì yêu cầu server node post lên server socket.
- Khi còn 2s hết video quảng cáo quay lại bước 1.



Hình 56. Sơ đồ web server nodejs

Sơ đồ tổng quát của hệ thống:

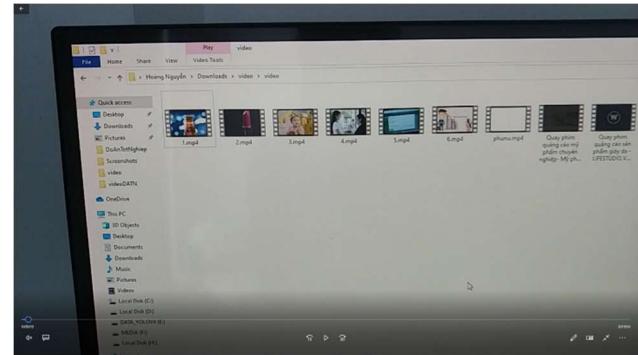


6. Hoạt động thực tế:

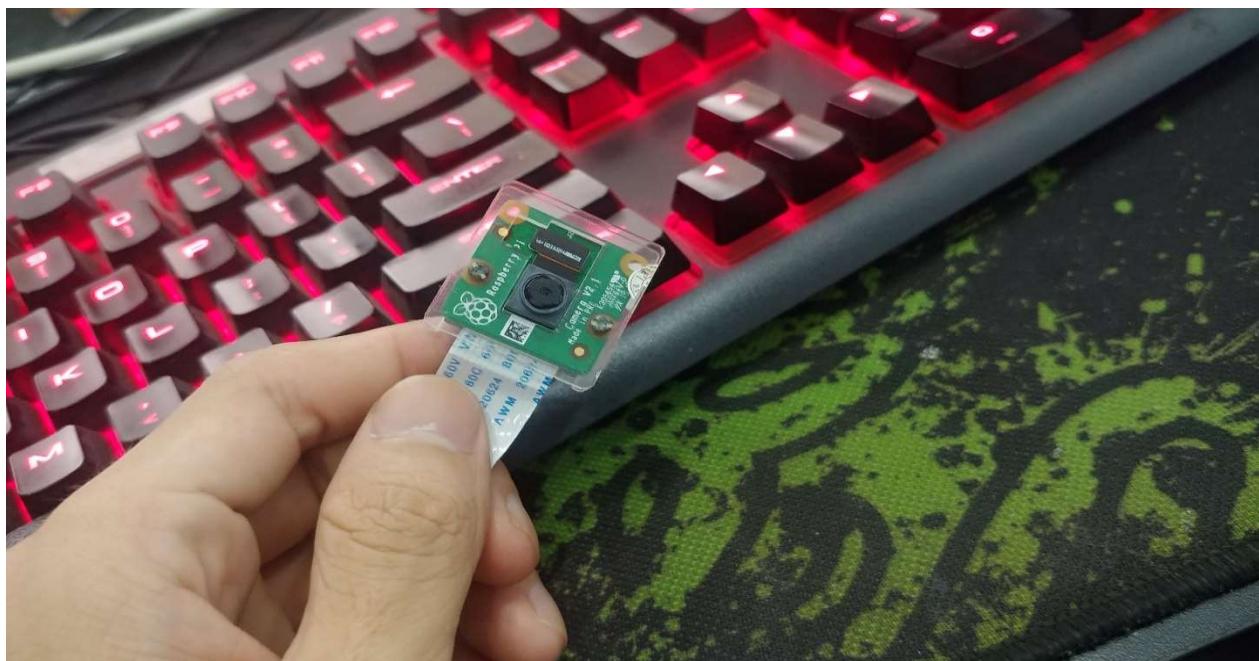
Khi hệ thống vẫn hành sẽ lấy các video quảng cáo trong thư mục video đã được tạo sẵn và cấu hình sẵn.



Hình 58. Jetson nano thực tế



Hình 59. Bộ video quảng cáo

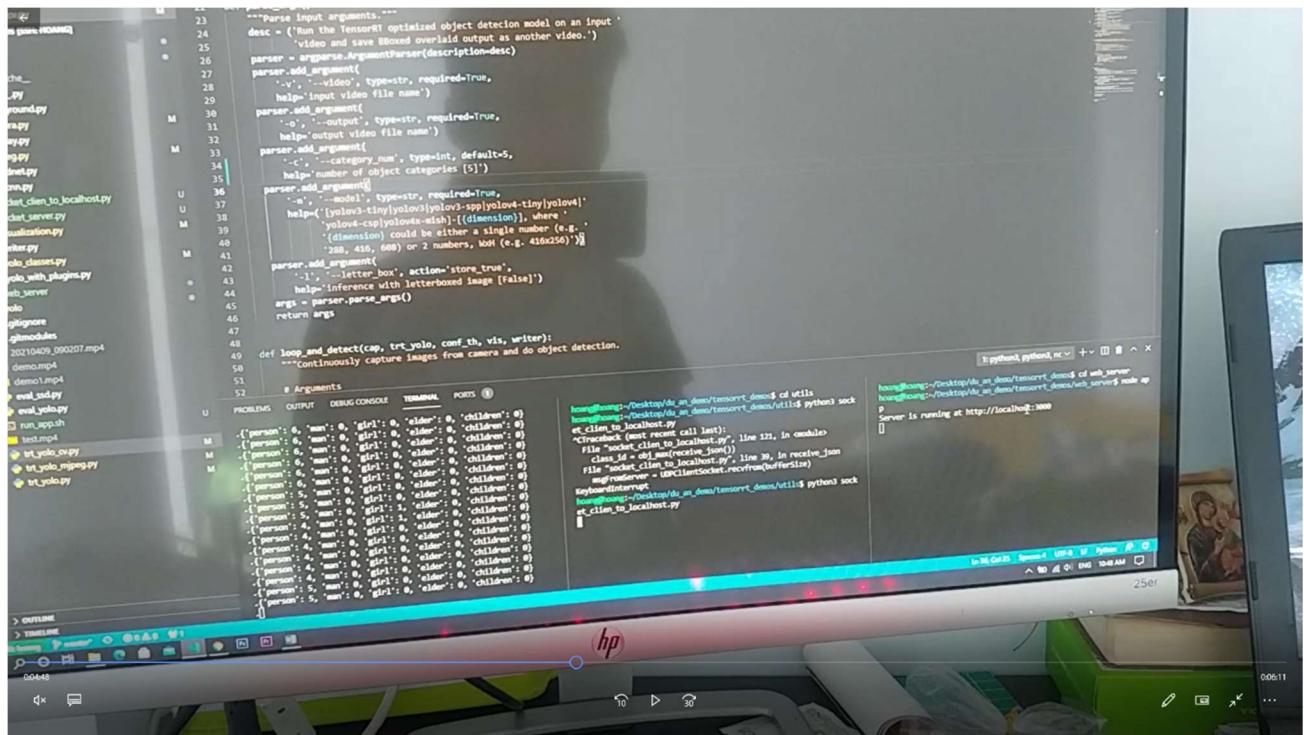


Hình 60. Camera Raspberry Pi V2

Hệ thống sẽ khởi động 3 host nhỏ:

- Host detect hình ảnh từ camera ra các classes và chuyển thành một json và gửi đi.
- Host Server socket nhận json từ host detect và tính toán đưa ra video gửi đi.
- Host web server nhận video và trình chiếu, video gần hết sẽ yêu cầu Host Server socket video mới.

Dưới đây là hình ảnh dữ liệu truyền đi giữa các host



Hình 61. Quá trình hệ thống hoạt động

Ta có thể theo dõi quá trình detect real time thông qua trình duyệt web với địa chỉ:
https://ip<jetson_nano>:8000



Hình 62. Hình ảnh class man thực tế từ camera

Khi hệ thống nhận diện được là nam 18-35 trả ra video như dưới hình là video quảng cáo sản phẩm cùi thẻ là bia úng với classes man 18-35



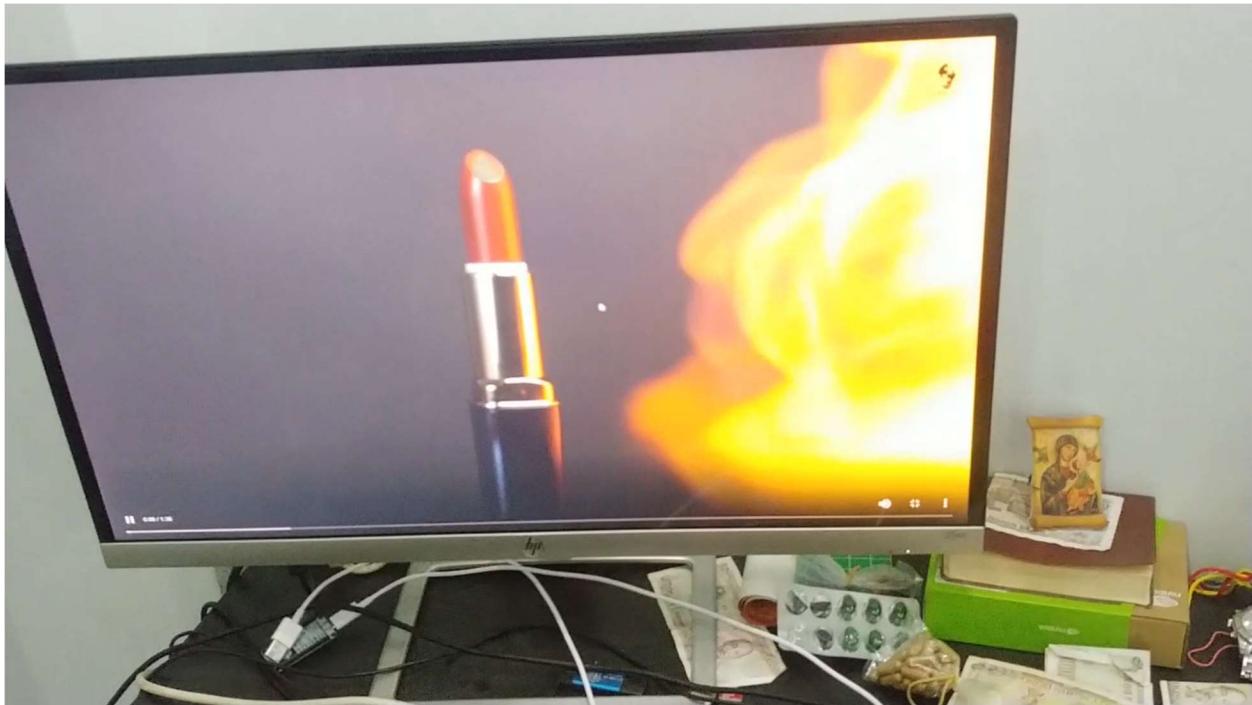
Hình 63. Hình ảnh chạy thực tế của quảng cáo úng với class man

Với kết quả nhận diện ra 1 người và 1 nữ thì video quảng cáo là mĩ phẩm



Hình 64. Hình ảnh class girl thực tế từ camera

Từ đó ta có thể thấy hệ thống hoạt động khá tốt



Hình 65. Hình ảnh chạy thực tế của quảng cáo ưng với class girl

CHƯƠNG V : KẾT LUẬN

Kết quả đạt được.

❖ Ưu điểm :

Nhận diện các vật thể đã được train với độ chính xác cao

Dễ dàng sử dụng

Bounding Box Predictions: cung cấp score mỗi bounding boxes sử dụng logistic regression.

Class Predictions: sử dụng logistic classifiers cho mọi class thay vì softmax.

Feature Pyramid Networks(FPN).

Darknet-53 được cập nhật mới.

Tốc độ khi chuyển đổi qua tensorrt tương đối nhanh và ổn định.

Thời gian phản hồi của server nhanh.

Object detection xử lý thời gian thực và chính xác là một trong những tiêu chí chính trên thế giới, một trong những ứng dụng là có thể áp dụng được vào những chiếc xe tự lái...

❖ Nhược điểm :

Thời gian train khá lâu (lớn hơn 49h khi train bằng google colab với gpu tesla T4)

Model nhận diện phụ thuộc nhiều vào chất lượng và vị trí của camera, vẫn còn lỗi trong quá trình làm việc.

Cần một lượng database rất lớn để cho máy học.

Model luôn luôn hoạt động gây tốn tài nguyên máy.

Hướng phát triển.

Với kết quả mà dự án đã làm được thì sẽ tạo cơ hội cho nhiều dự án mới phát triển lên dựa theo dự án này để áp dụng vào đời sống hiện tại của chúng ta giúp ích cho xã hội ngày càng hiện đại hơn như áp dụng vào ngành y tế có thể cho AI quét thân nhiệt con người giúp phát hiện bệnh nhân bị sốt là một thứ rất cần trong tình hình thế giới đang xảy ra dịch Covid-19 hiện nay, hay áp dụng vào trong ngành giao thông ta cho AI nhận diện xe oto lưu thông trên đường giúp cảnh báo lượng xe cộ trên đường hoặc khi có tai nạn sẽ trực tiếp gửi thông tin qua y tế để gọi cấp cứu một cách nhanh nhất.

Tài liệu tham khảo :

- <https://github.com/AlexeyAB/darknet>
- <https://github.com/tzutalin/labelImg>
- https://github.com/jkjung-avt/tensorrt_demos
- <https://developer.nvidia.com/tensorrt>
- <https://pypi.org/>
- <https://docs.python.org/3/howto/sockets.html>
- <https://codelearn.io/sharing/lap-trinh-socket-voi-tcpip-trong-python>
- <https://viblo.asia/p/lap-trinh-socket-bang-python-jvEla084Zkw>
- <https://www.it-swarm-vi.com/vi/javascript/van-de-python-flask-cors/1051130597/>
- <https://blog.miguelgrinberg.com/post/video-streaming-with-flask>
- <https://github.com/miguelgrinberg/flask-video-streaming>
- <https://www.pyimagesearch.com/2019/09/02/opencv-stream-video-to-web-browser-html-page/>
- <https://vntalking.com/xay-dung-ung-dung-web-voi-nodejs-expressjs.html>
- <https://www.w3schools.com/>
- <https://miae.vn/2020/05/25/yolo-series-train-yolo-v4-train-tren-colab-chi-tiet-vadu-a-z/>
- <https://github.com/AlexeyAB/darknet/wiki>
- <https://openplanning.net/11415/python-class-object>
- <https://www.howkteam.vn/course/lap-trinh-huong-doi-tuong-voi-python/lop-vadoi-tuong-trong-lap-trinh-huong-doi-tuong-voi-python-3877>
- <https://viblo.asia/p/oop-voi-python-E375zQGblGW>
- https://www.w3schools.com/python/python_json.asp
- <https://docs.python.org/3/library/json.html>
- <https://viblo.asia/p/huong-dan-cai-dat-moi-truong-cho-jetson-nano-phuc-vu-nghien-cuu-machine-learning-3Q75w8V BKWb>

<https://aicurious.io/posts/2020-04-02-thiet-lap-ban-dau-cho-jetson-nano/>

<https://www.tensorflow.org/>

<https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html>

<https://forums.developer.nvidia.com/t/official-tensorflow-for-jetson-nano/71770>

<https://opencv.org/>

<https://qengineering.eu/install-opencv-4.5-on-jetson-nano.html>

<https://pypi.org/project/opencv-python/>

<https://pivietnam.com.vn/huong-dan-su-dung-ssh-winscp-remote-desktop-cho-nvidia-jetson-nano-pivietnam-com-vn-mlab-vn.html>

https://www.youtube.com/watch?v=VeOj_C6rAF4&ab_channel=cytrontech

<https://devai.info/2020/12/17/tim-hieu-mapmean-average-precision-danh-gia-mo-hinh-object-detection-su-dung-yolov4/>

https://www.w3schools.com/tags/ref_httpmethods.asp

<https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html>