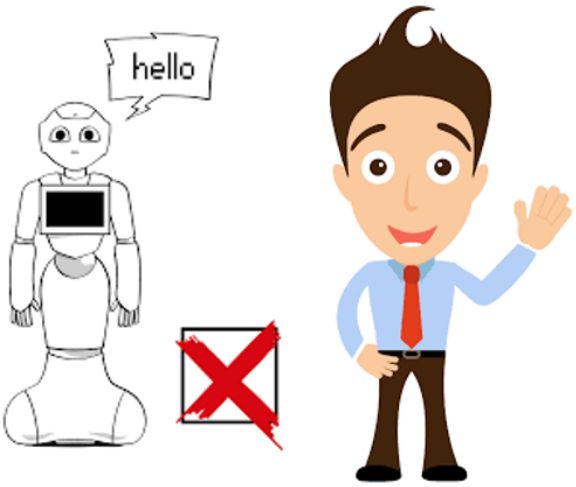
Report Hand Gesture Recognition

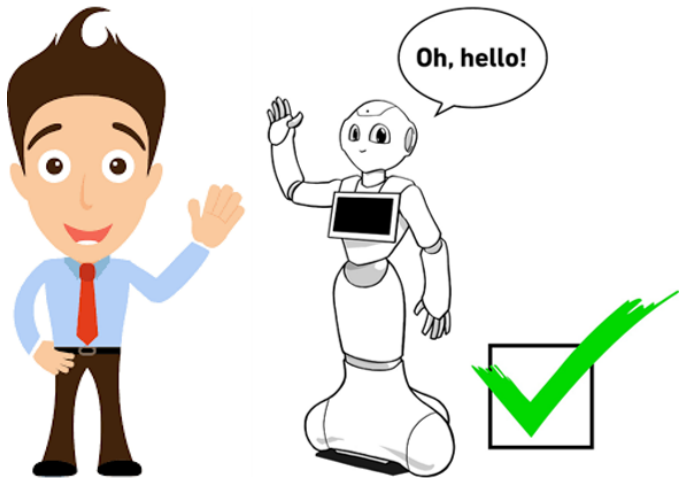
1. Tổng quan về hệ thống

Mục tiêu của dự án này là sẽ training cho robot học được những cử chỉ của bàn tay của con người và từ đó có thể đưa ra một số quyết định như là robot sẽ đưa ra một câu chào lại, hoặc robot có thể thực hiện những lệnh khác như điều khiển các thiết bị điện trong nhà như đèn quạt, loa, tivi,…

Ứng dụng này sẽ được nhúng vào con robot lễ tân ảo nhờ đó robot có thể hiểu được những cử chỉ điều khiển của con người và có thể đánh thức một số hoạt động của robot.



Hình 1: Khi robot không hiểu được cử chỉ con người



Hình 2 : Khi robot có thể hiểu được những cử chỉ của con người và ra quyết định chào lại

Đối với những bài toán về Recognition về cơ bản sẽ có sơ đồ hệ thống chung như hình 3, từ ảnh đầu vào đến khối hand detection, và từ đó ta bắt đầu qua khối Hand recognition để dự đoán và đưa ra kết quả.

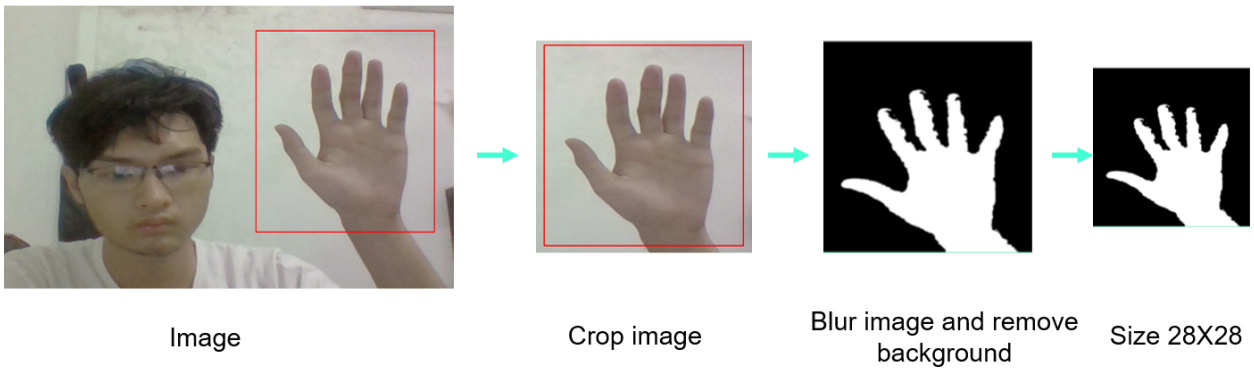
Hình 3: Sơ đồ hệ thống đối với bài toán Recognition

2. Chuẩn bị dữ liệu, xử lý dữ liệu

Đối với quá trình xử lý dữ liệu, thì em đã nghiên cứu và triển khai 2 phương án như sau:

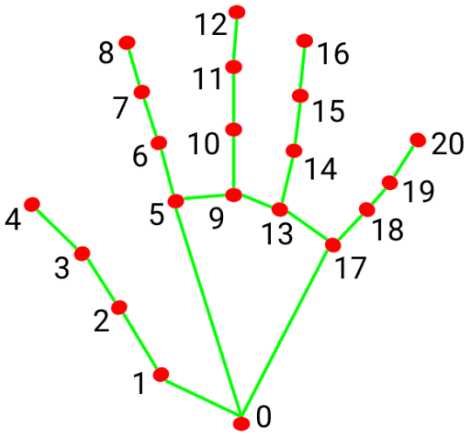
Xử lý dữ liệu bằng phương pháp blur image and remove background

- Ở đây em sử dụng phương pháp blur image và remove background để xử lý những điểm nhiễu ở phía sau bàn tay và từ đó tạo ra bộ dữ liệu.
- Ưu điểm: phương pháp này thuần xử lý ảnh nên có có thể tối ưu được tốc độ xử lý của phần cứng.
- Nhược điểm: phương pháp này có một nhược điểm chính là sẽ có nhiều trường hợp gặp những nhiễu khó có thể xử lý tốt được.



Hình 4: Xử lý dữ liệu bằng phương pháp Blur image and remove background

Xử lý dữ liệu bằng phương pháp hand landmarks



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Hình 5: Biểu diễn 21 điểm landmark trên bàn tay

- Để cải thiện những nhược điểm đó thì em đã nghiên cứu một phương án mới đó chính là sử dụng hand landmarks detection để có thể vẽ ra được 21 điểm và từ đó có thể biểu diễn thành những bức ảnh binary như vậy
- Ưu điểm: Phương pháp này có nhiều điểm vượt trội như là có thể cho ra một kết quả hình vẽ chính xác nhờ vào việc sử dụng 2 model để xử lý (đó chính là model palm detection and hand landmark detection).
- Nhược điểm: phương pháp này có một nhược điểm đó chính là tốc độ xử lý cao, nó mất nhiều thời gian để xử lý.

Hình 6: Xử lý dữ liệu bằng phương pháp hand landmark detection

Tạo dataset từ camera OAK-D

- Từ kết quả của phương pháp hand landmark detection trên (hình 6), em đã viết một đoạn script để có trên thu data một cách tự động.
- Chúng em tạo data bằng cách, một người sẽ đứng trước camera và sẽ training trực tiếp những icon hay những cử chỉ mà người trainer cần train cho model có thể học được, kết quả từ frame đó nó sẽ được lưu lại vào một tệp để tiện cho quá trình training sau này.
- Ở bộ dataset này chúng em xây dựng với 12 class cơ bản tương ứng cho những cử chỉ đơn giản như ví dụ ở hình 8

ID	Label	Gesture
1	G00	Non-Gesture
2	G01	Number zero
3	G02	Number one
4	G03	Number two or say Hi
5	G04	Number three
6	G05	Number four
7	G06	Number five or say Hello
8	G07	Icon like
9	G08	Icon dislike
10	G09	Icon stop
11	G10	Icon heart
12	G11	Icon okay

Hình 7: 12 class của tập dataset

- Dưới đây là hình ảnh của 4 class trong 12 class của bộ dataset

Class 2: Number zero

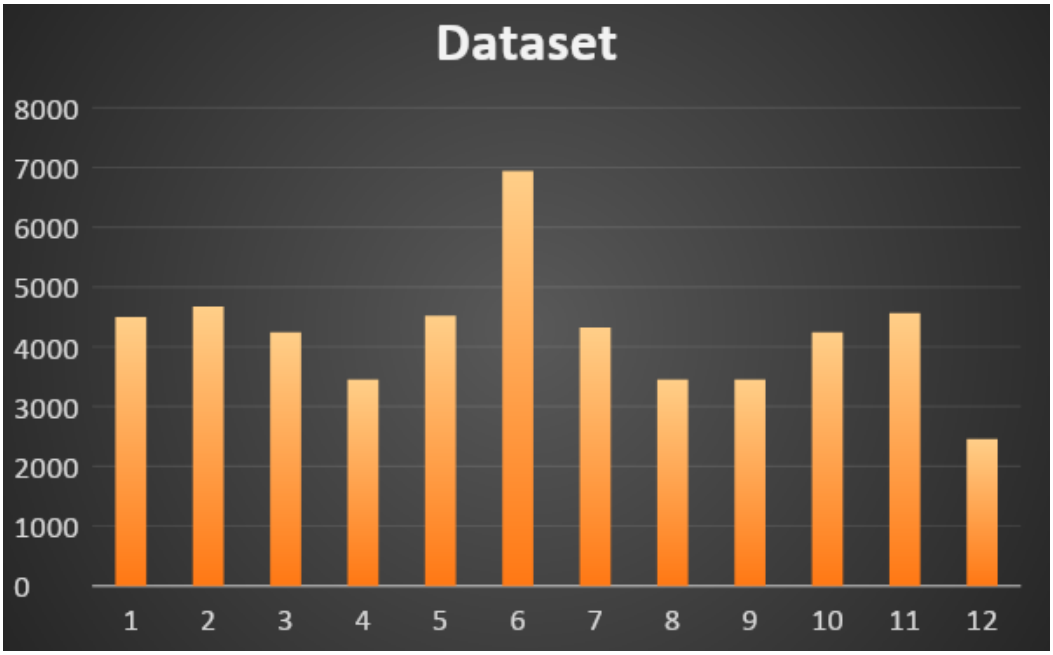
Class 7: Number five or say Hello

Class 10: Icon stop

Class 12: Icon okay

Hình 8: Hình ảnh minh họa những class

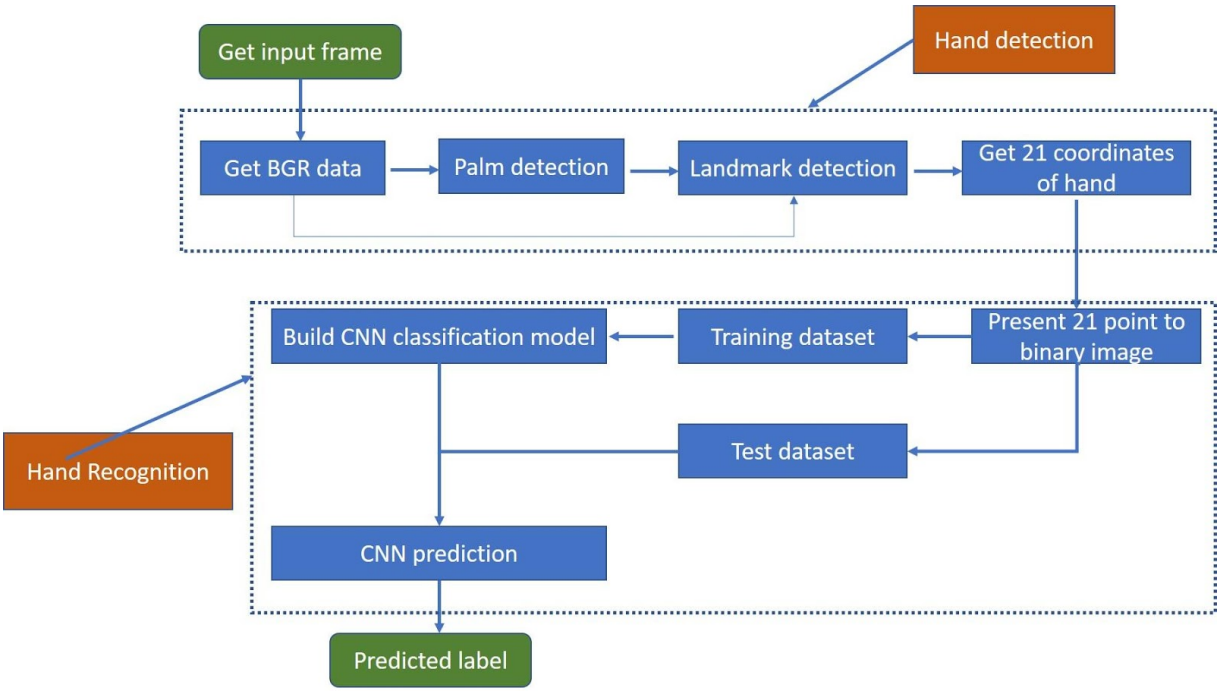
- Như mọi người cũng đã biết thì số lượng dataset cũng ảnh hưởng đến tốc độ chính xác của mô hình nên trong bài báo cáo này mình đã vẽ ra một biểu đồ để biểu diễn số lượng các Class trong tập dataset. Nhìn chung mà nói thì tập dataset này các class không lệch nhau nhiều lắm.



Hình 9: Đồ thị biểu diễn số lượng các class của tập dataset

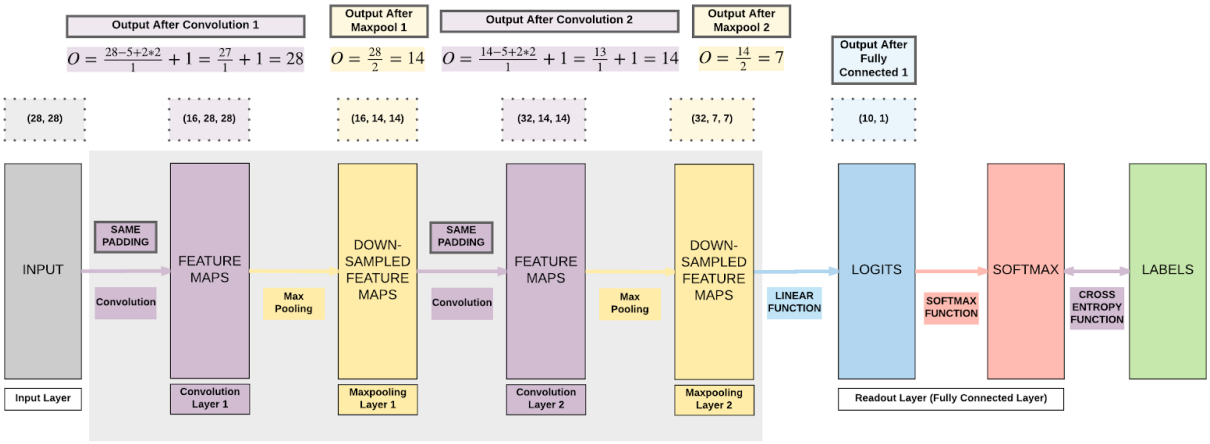
1. Training model

- Sau khi đã xây dựng xong được tập dataset, em bắt đầu đi vào xây dựng một mô hình CNN để training các tập dataset đó.
- Mọi người có thể nhìn thấy tập dataset sau khi được tạo khá là clear và đơn giản nhận biết nên theo lý thuyết thì chỉ cần một vài lớp CNN thì model có thể học được bộ dataset đó với độ chính xác cao.
- Để hiểu rõ hơn về tổng quan của hệ thống, mọi người có thể nhìn vào hình 9 ở bên dưới, hình bên dưới thể hiện tổng quan hệ thống, từ quá trình xử lý data đến quá trình training.
- Đầu tiên, thiết bị sẽ thu hình ảnh từ camera rồi từ đó đưa hình ảnh đi qua một model Palm detection để phát hiện ra được 5 điểm trên lòng bàn tay, và từ kết quả đó ta sẽ kết hợp với hình ảnh ban đầu đi qua tiếp 1 model Landmark để detect ra 21 điểm tương ứng trên bàn tay như hình 5, sau khi đã detect được 21 điểm trên bàn tay em sẽ tiến hành vẽ chúng trên một mặt phẳng 2D khác để tạo ra một hình ảnh cho quá trình training và recognition sau này. Tiếp theo đó là chia tập dữ liệu thành training và testing để đưa vào model và huấn luyện cho model AI có thể học được bộ dataset đó



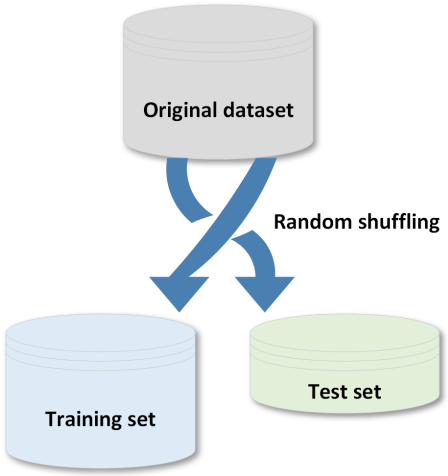
Hình 10: Sơ đồ tổng quan của hệ thống

- Xây dựng mô hình huấn luyện CNN như hình 11, ở đây chúng ta có thể build một mô hình CNN cơ bản chỉ một vài lớp CNN để training cho tập dataset, mọi người có thể thấy rõ những parameter em đã note ở trên hình rất là cụ thể.



Hình 11: Kiến trúc CNN cho quá trình recognition

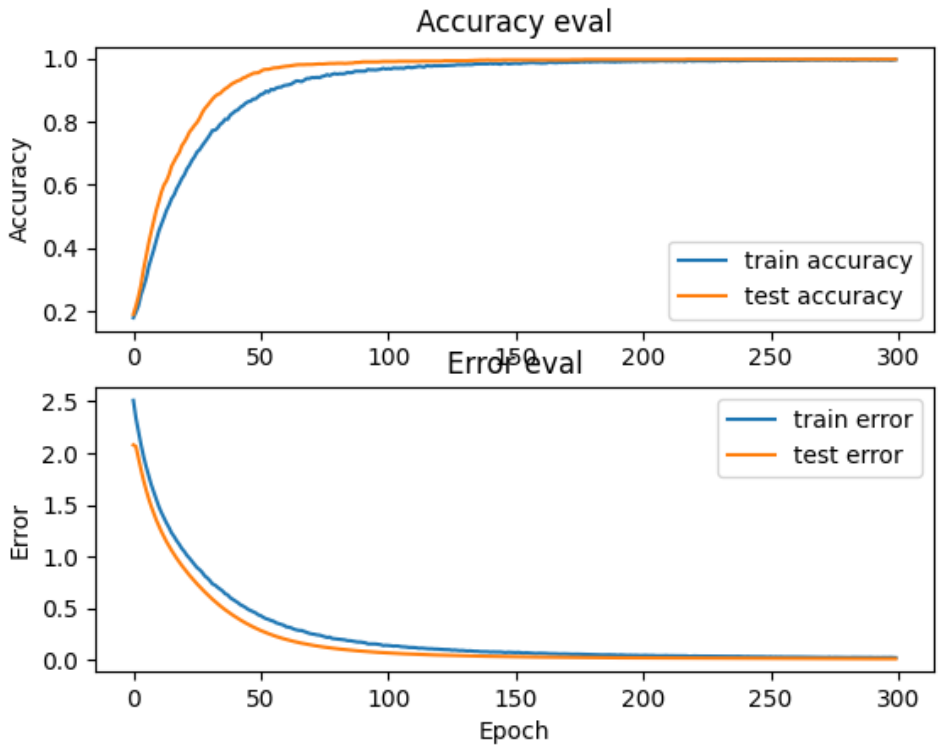
- Trước khi đưa vào mạng CNN để training, thì em đã load dataset đã tạo được ở mục trên và sẽ chia tập dataset thành tập training và testing, với batch size là 100



Hình 12: Chia tập dữ liệu dataset

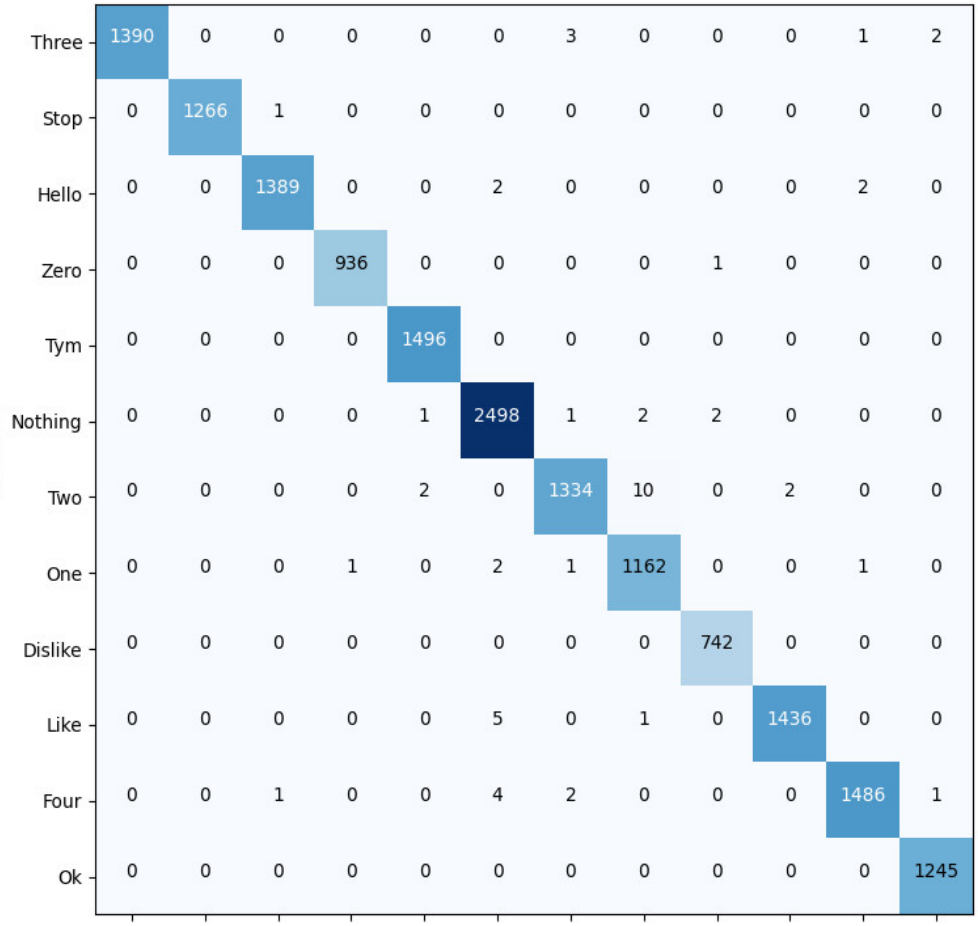
2. Đánh giá model

- Sau khi traning xong model, bước tiếp the khác là quan trọng đó chính là đánh giá model và kiểm tra model có hoạt động tốt hay không.



Hình 13: Kết quả đánh giá sau khi training model

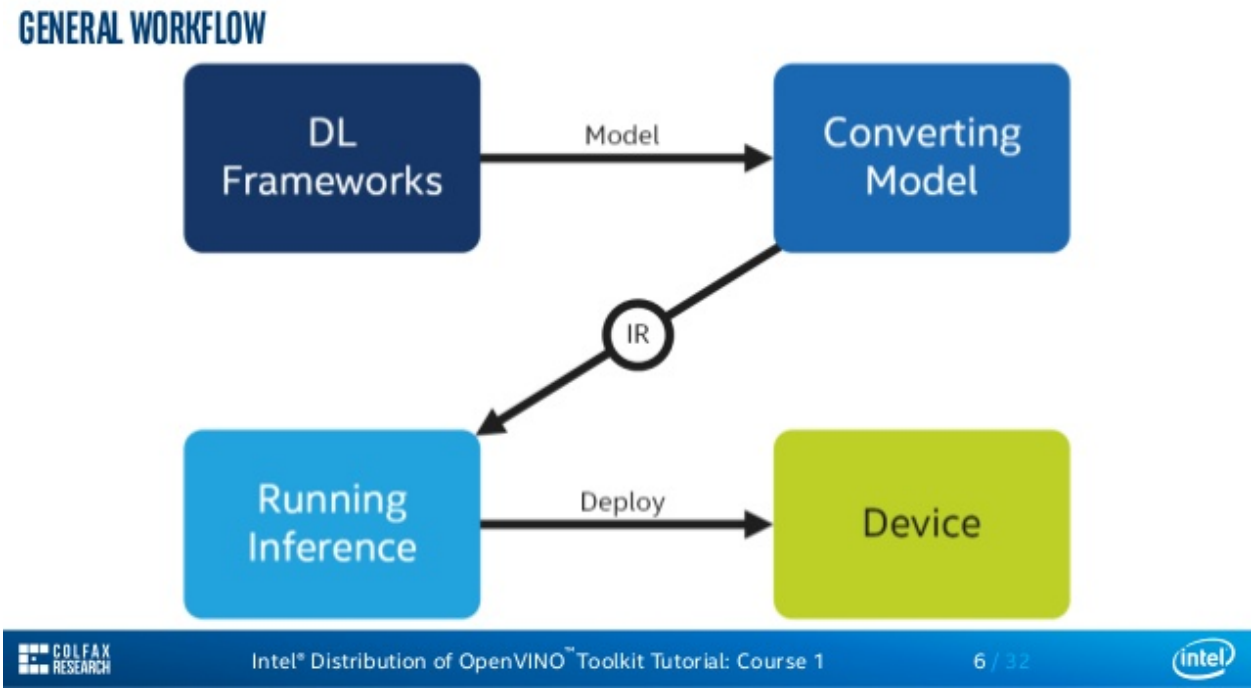
- Nhìn vào hình 13 đây chính là đồ thị biểu diễn kết quả của Accuracy và Loss sau khi training với epoch là 300.
- Từ đồ thị ta có thể thấy được rằng giá trị train và giá trị test khá là điều nhau, và được biểu diễn theo một đường cong hàm log, nên chúng ta có thể khẳng định được rằng model đã cho ra kết quả tốt so với những tiêu chí đề ra.
- Và có thể kiểm tra được bộ model có bị overfitting hay không thì em đã vẽ ra một bảng ma trận nhầm lẫn dựa trên quá trình dự đoán sau khi training từ tập testset (hình 14)
- Mọi người nhìn vào có thể hiểu được ngay, có rất ít những class dự đoán ra kết quả sao
- Và từ kết quả của ma trận nhầm lẫn đó ta có thể tính toán và đưa ra được độ chính xác của model, và độ chính xác của model được tính bằng công thức (Accuracy = tổng số kết quả dự đoán chính xác / tổng số lượng dataset)
- **Accuracy = (16426 – 51) / 16426 = 0.9968 = 99.68 %**



Hình 14: Bảng ma trận nhầm lẫn của tập test

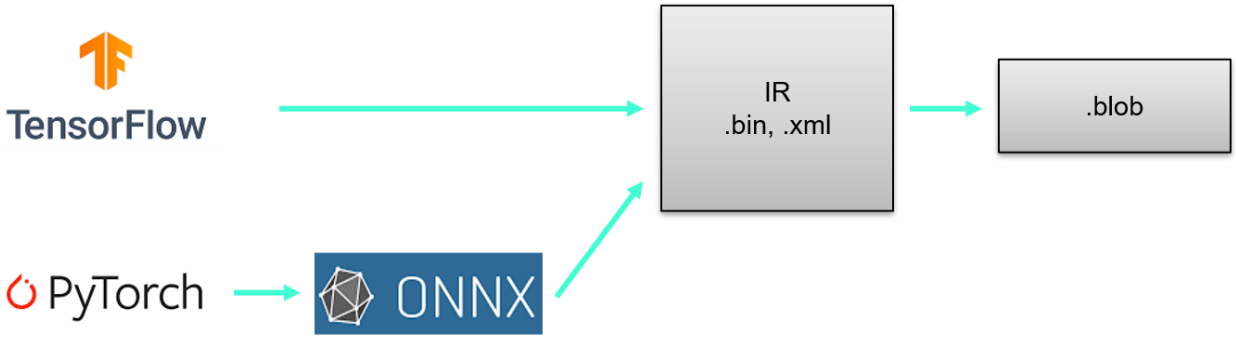
3. Chuyển model thành định dạng openvino

- Để có thể deploy xuống những thiết bị phần cứng có cấu hình thấp hay những thiết bị embedded system thì em sẽ chuyển những model trên thành định dạng model openvino để tối ưu tốc độ tính toán dưới các thiết bị phần cứng
- Openvino là một tool được phát triển bởi một nhóm nghiên cứu của công ty intel, nhằm tối ưu các model như Tensorflow, Pytorch, Caffe, Keras, … thành định dạng IR openvino và có thể thực thi dưới các thiết bị phần cứng hỗ trợ xử lý AI như camera OAK-D



Hình 15: Cách thức chuyển đổi từ model training sang định dạng openvino

- Trong dự án này chúng ta cần chuyển đổi 3 model từ định dạng Tflite, Tensorflow, Pytorch thành định dạng openvino và định dạng .blob để có thể run trên các thiết bị phần cứng có cấu hình thấp.



Hình 16: Chuyển đổi model sang định dạng openvino

4. Thực thi trên thiết bị phần cứng và camera OAK-D

- Sau khi thực hiện xong quá trình chuyển đổi các model sang định dạng openvino, bước tiếp theo cũng cực kỳ quan trọng đó chính là nhúng các model đó xuống các thiết bị phần cứng có cấu hình khác nhau như Intel Core i7 và Raspberry Pi4 để thực hiện quá trình đánh giá và chạy real time.

intel® core™ i7-8550u cpu 16GB

Raspberry pi 4 8GB

Camera OAK-D

Hình 17: Thực hiện dự án trên thiết bị phần cứng và camera OAK-Ds

- Sau khi model được chạy dưới các thiết bị phần cứng, hình 18 dưới đây là đánh giá tốc độ xử lý của các các model dựa theo tiêu chí thời gian, trên các thiết bị phần cứng khác nhau.

Evaluate Model							
Model recognition	**Device**	**Camera**	**Size model**	**Detect time (ms)**	**Predict time (ms)**	**Total time (ms)**	**Fps**
Tensorflow	Intel i7, 16GB	OAK-D	123.6 kB	20,82	35,26	56,08	17
Pytorch	Intel i7, 16GB	OAK-D	91.5 kB	21,99	1,49	23,48	32
Openvino convert by Tensorflow	Intel i7, 16GB	OAK-D	162.3 kB	70,89	0,93	71,82	14
Openvino convert by Pytorch	Intel i7, 16GB	OAK-D	47.8 kB	70,36	0,98	71,34	14
Model recognition	**Device**	**Camera**	**Size model**	**Detect time (ms)**	**Predict time (ms)**	**Total time (ms)**	**Fps**
Tensorflow	Raspberry pi4, 8GB	OAK-D	123.6 kB	160,36	40,56	200,92	5
Pytorch	Raspberry pi4, 8GB	OAK-D	91.5 kB	161,68	20,76	182,44	6
Openvino convert by Tensorflow	Raspberry pi4, 8GB	OAK-D	162.3 kB	75,68	1,56	77,24	13
Openvino convert by Pytorch	Raspberry pi4, 8GB	OAK-D	47.8 kB	76,67	1,67	77,67	13

Hình 18: Hình ảnh đáng giá tốc độ xử lý của các model

- Hình 18 là bảng đánh giá tốc độ xử lý của các model dựa trên các tiêu chí về tốc độ xử lý, và thiết bị phần cứng, và có thể thấy rõ rằng các yếu tố này ảnh hưởng rất lớn đến tốc độ xử lý của model
- Model chạy trên phần cứng Intel Core i7 và camera OAK – D
    - Đối với những model chưa được convert sang định dạng openvino cho thấy tốc độ xử lý tốt và ổn định bởi vì các model hiện tại đang chạy dưới một thiết bị phần cứng có cấu hình cao Intel Core i7, và ngay lúc này camera OAK-D chưa xử lý tác vụ AI nào.
    - Sau khi những model đã được convert sang định dạng openvino thì chúng ta có thể thấy tốc độ xử lý chậm hơn so với những model chưa convert, lý do bởi vì ở đây những mạng NN AI đang được xử lý trên thiết bị camera OAK – D chứ không phải ở thiết bị phần cứng Intel Core i7 (một điều hiển nhiên là phần cứng của camera OAK không thể mạnh bằng phần cứng của chip intel core i7 được nên việc quá trình xử lý chậm hơn là điều dễ hiểu).
- Model chạy trên thiết bị phần cứng Raspberry pi 4 8GB và camera OAK-D
    - Đối với những model chưa được convert sang định dạng openvino cho thấy kết quả xử lý rất lâu, chỉ được 5 đến 6

FPS, lý do vì hiện tại thiết bị phần cứng Pi đang chịu trách nhiệm xử lý AI cả 3 model nên tốn nhiều thời gian là đều hiển nhiên và tại lúc này thiết bị camera OAK – D chưa có tác vụ xử lý AI.

  - Sau khi những model đã được convert sang định dạng openvino thì chúng ta có thể thấy rằng tốc độ xử lý đã cải thiện rất nhiều và có thể lên đến 12 -> 14 FPS, và có thể nhận thấy ràng tốc độ xử lý gần bằng với tốc độ xử lý khi thực hiện với thiết bị phần cứng Intel Core i7.

- Từ những kết quả trên chúng ta có thể kết luận rằng, khi sử dụng những model convert sang định dạng openvino thì những model sẽ được thực thi trên thiết bị phần cứng của camera, và nó có thể giúp cho những phần cứng ở dưới như Raspberry hay Intel Core i7 không cần phải xử lý những tính toán quá phức tạp.

5. Demo

6. Tài liệu tham khảo

[1]. https://google.github.io/mediapipe/solutions/hands.html
[2]. https://github.com/luxonis/depthai-experiments
[3]. https://luxonis.com/depthai
[4]. https://docs.openvinotoolkit.org/latest/index.html
[5]. https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html