ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BÀI TẬP LỚN CUỐI KÌ

Chủ đề:

FACE TRACKING VÀ EYE TRACKING PHÁT HIỆN DẤU HIỆU TỰ KỈ VÀ CÁC VẤN ĐỀ VỀ MẮT Ở TRẺ

Giảng viên hướng dẫn: TS. Phạm Đức Quang

Nguyễn Sỹ Sơn	**-**	**20021578**
Ngô Hoàng Khánh Văn	**-**	**20021599**
Đỗ Xuân Hiếu	**-**	**20021527**
Vũ Hoàng Anh	**-**	**20021492**
Nguyễn Hữu Huy	**-**	**20021539**

Hà Nội, tháng 12 năm 2023

Dưới đây là bảng phân công công việc trong nhóm 6:

STT	Tên thành viên	Nội dung	
1	Nguyễn Sỹ Sơn	Triển khai hệ thống lên nền tảng web, xây dựng cơ sở dữ liệu.	20%
2	Ngô Hoàng Khánh Văn	Face tracking: Tìm hiểu mô hình MobileNet và huấn luyện mô hình nhận diện cảm xúc.	20%
3	Đỗ Xuân Hiểu	Face tracking: Tìm hiểu dấu hiệu tự kỷ qua cảm xúc và sử dụng mô hình để đưa ra chuẩn đoán.	20%
4	Vũ Hoàng Anh	Eye tracking: Tìm hiểu, sử dụng OpenCV+ Mediapie trích xuất những đặc trưng của mắt để chuẩn đoán bệnh nhãn khoa.	20%
5	Nguyễn Hữu Huy	Eye tracking: Tìm hiểu, sử dụng OpenCV+ Mediapie trích xuất những đặc trưng của mắt để chuẩn đoán dấu hiệu bệnh tự kỷ.	20%

LỜI CẢM ƠN

Đầu tiên, nhóm 6 môn học "**Quang điện tử**" xin được gửi lời cảm ơn chân thành TS. Phạm Đức Quang, người tận tình giúp đỡ cũng như đã tạo điều kiện tốt nhất giúp nhóm 6 trong project cuối kỳ này.

Mặc dù đã cố gắng hết sức mình để hoàn thiện báo cáo nhưng do sự hạn chế về kiến thức, kinh nghiệm nên bài báo cáo không thể tránh khỏi những thiếu sót, nhóm em rất mong nhận được sự đóng góp ý kiến của thầy! Bài báo cáo gồm 8 phần như sau:

Phần 1: Giới thiệu

Phần 2: Tổng quan hệ thống

Phần 3: CNN sử dụng kiến trúc MobileNet trong việc nhận diện trẻ tự kỷ

A. Dấu hiệu nhận biết tự kỷ qua cảm xúc

B. Giới thiệu về CNN, MobileNet

C. Training

D. Sử dụng model để chuẩn đoán

E. Kết quả

F. Đánh giá

Phần 4: Eye tracking để nhận diện bệnh về mắt và một số dấu hiệu của trẻ tự kỷ.

A. Giới thiệu

B. Tổng quan về mô hình thực hiện

C. Công Nghệ và Thư Viện Sử Dụng

D. Thuật toán và phương pháp

E. Phương pháp chẩn đoán bệnh về mắt và phát hiện trẻ tự kỷ

F. Kết quả

G. Đánh giá

Phần 5: Triển khai hệ thống trên nền tảng web

A. Giới thiệu về FLASK

B. Cơ sở dữ liệu

C. Kết quả triển khai

D. Đánh giá kết quả triển khai lên web

Phần 6: Kết luận toàn bộ hệ thống

Phần 7: Tài liệu tham khảo

Cuối cùng nhóm em xin kính chúc quý thầy sức khỏe dồi dào và gặt hái được nhiều thành công trong cuộc sống cũng như sự nghiệp đào tạo những hệ tri thức tiếp theo cho đất nước.

Nhóm 6 xin chân thành cảm ơn!

PHẦN 1. Giới thiệu

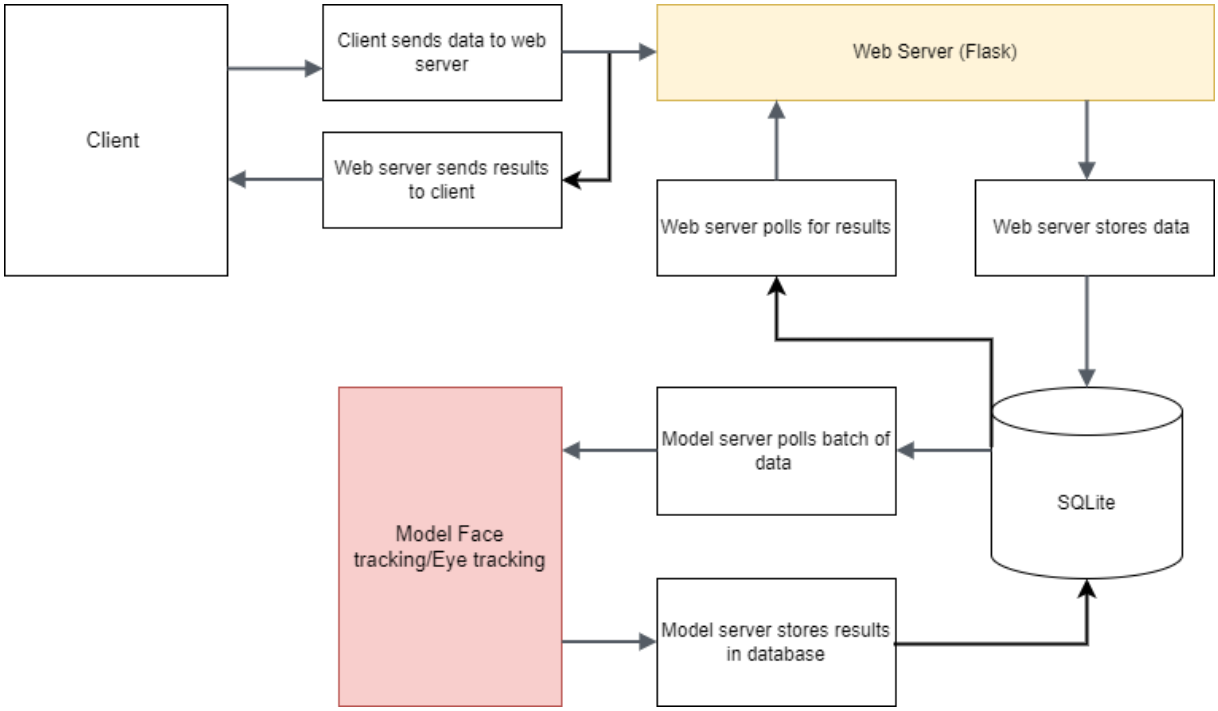
Tự kỷ là một căn bệnh ngày càng trở thành mối đe dọa đối với sự phát triển của trẻ nhỏ đồng thời cũng gây ra nhiều lo lắng cho các bậc phụ huynh. Chính vì vậy, việc phát hiện sớm các dấu hiệu của bệnh tự kỷ sẽ giúp đưa ra phương pháp điều trọ kịp thời để trẻ sớm hòa nhập với cuộc sống xung quanh.

Việc phát hiện sớm những dấu hiệu của trẻ tự kỷ là vô cùng quan trọng vì khi phát hiện sớm và đưa ra những biện pháp can thiệp kịp thời, chúng ta sẽ giúp trẻ tránh khỏi những hậu quả nghiêm trọng và nhanh chóng đưa trẻ trở lại với cuộc sống bình thường như những trẻ em khác.

Bên cạnh đó, trẻ em ngày nay cũng thường xuyên sử dụng điện thoại di động để giải trí, học tập và giao tiếp. Tuy nhiên, việc sử dụng điện thoại một cách quá mức có thể đặt ra nhiều rủi ro cho sức khỏe của trẻ, trong đó có nguy cơ mắc bệnh nhãn khoa và tự kỷ. Bài toán phát hiện trẻ có bệnh về mắt thông qua các hệ thống chẩn đoán có thể triển khai tại nhà là một bài toán khá quan trọng nhằm giúp cha mẹ có thể phát hiện sớm các dấu hiệu bệnh nhãn khoa của trẻ nhỏ, từ đó có thể đưa ra liệu pháp điều trị kịp thời tạo sự phát triển tốt nhất về thể chất cho trẻ.

Như vậy, kết hợp 2 vấn đề trên, nhóm em quyết định làm 1 hệ thống face tracking và eye tracking để phát hiện sớm những dấu hiệu về bệnh tự kỷ hoặc các bệnh về mắt ở trẻ.

PHẦN 2. Tổng quan hệ thống



Hình 1: Sơ đồ khối toàn bộ hệ thống

Hệ thống gồm 2 phần cốt lõi đó chính là mô hình AI face tracking và eye tracking có chức năng phát hiện dấu hiệu tự kỷ và các bệnh nhãn khoa. Sau đó tích hợp 2 mô hình này lên web và có sử dụng cơ sở dữ liệu nhằm mục đích lưu trữ, thống kê. Người dùng có thể truy cập từ xa vào trang web thông qua internet.

Về cách thức hoạt động của trang web, trẻ em cần chuẩn đoán sẽ được xem 1 đoạn video test, trong quá trình đó sẽ record lại khuôn mặt của trẻ. Sau đó video record được đưa vào mô hình AI để xử lý và đưa ra kết quả chuẩn đoán.

PHẦN 3. Sử dụng kiến trúc MobileNet dựa trên mô hình mạng neural tích chập CNN trong việc nhận diện trẻ tự kỷ

A. Dấu hiệu nhận biết trẻ qua cảm xúc

Tự kỷ là tập hợp các rối loạn phát triển với nhiều mức độ khác nhau từ nhẹ tới nặng, thường được khởi phát trước khi trẻ 3 tuổi và diễn ra biến kéo dài vào những năm sau đó và thậm chí là suốt đời.

Đã có rất nhiều những nghiên cứu về dấu hiệu sớm của trẻ tự kỷ trên thế giới. Các dấu hiệu chủ yếu xoay quanh về cảm xúc, hành vi…Một số dấu hiệu phổ biến như: bất thường về ngôn ngữ; bất thường về hành vi; trẻ có xu hướng ngại giao tiếp xã hội, sống thu mình; có những tài năng đặc biệt. Trong dự án này, nhóm em tập chung khai thác dấu hiệu về sự bất thường cảm xúc của trẻ.

Trẻ bị tự kỷ thường có nhưng bất ngờ về cảm xúc, vui buồn lẫn lộn. Cảm xúc của chúng có thể thay đổi một cách bất ngờ, đang vui cười bỗng nhiên gào khóc rất khó nắm bắt.

Hình 2: Dấu hiệu về cảm xúc của trẻ mắc tự kỷ

B. Kiến trúc mobile Net và mô hình mạng neural tích chập CNN

1. Định nghĩa Huấn luyện mô hình (train model)

Huấn luyện mô hình (train model) là quá trình sử dụng dữ liệu huấn luyện để điều chỉnh các tham số của một mô hình máy học hoặc mô hình học sâu (deep learning) có sẵn (pretrained model) để nó có thể thực hiện công việc cụ thể của người dùng. Trong quá trình này, mô hình sẽ học từ dữ liệu bằng cách điều chỉnh các tham số nội tại của nó để tối ưu hóa một hàm mục tiêu, ví dụ như giảm thiểu sai số dự đoán.

Các bước cơ bản của việc huấn luyện mô hình bao gồm:

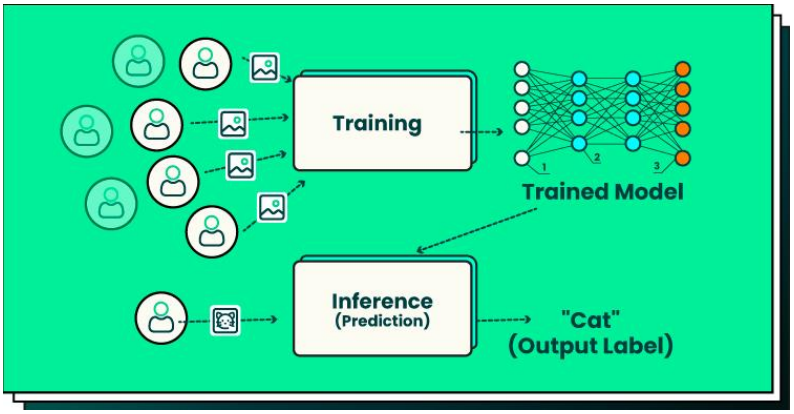
Bước 1: Chuẩn bị dữ liệu: Đưa dữ liệu vào định dạng phù hợp để huấn luyện mô hình, bao gồm việc tiền xử lý, chia thành tập huấn luyện và tập kiểm tra.

Bước 2: Chọn mô hình: Lựa chọn kiến trúc mô hình phù hợp với bài toán cụ thể. Có thể là một mô hình học máy cơ bản như hồi quy tuyến tính (Linear Regression), máy vector hỗ trợ (Support Vector Machine), hoặc một mô hình sâu hơn như mạng nơ-ron sâu (deep neural network).

Bước 3: Huấn luyện mô hình: Áp dụng dữ liệu huấn luyện vào mô hình để điều chỉnh các tham số sao cho mô hình có thể dự đoán chính xác hơn. Quá trình này thường sử dụng các thuật toán tối ưu hóa như gradient descent để tối ưu hóa hàm mất mát.

Bước 4: Đánh giá mô hình: Đánh giá hiệu suất của mô hình trên tập dữ liệu kiểm tra hoặc tập dữ liệu chưa từng được cho vào mô hình trước đó. Điều này giúp đánh giá xem mô hình có đủ khả năng tổng quát hóa hay không.

Bước 5: Tinh chỉnh và cải thiện: Dựa trên kết quả đánh giá, mô hình có thể được tinh chỉnh (fine-tuning) bằng cách thay đổi kiến trúc, tham số, hoặc phương pháp huấn luyện để cải thiện hiệu suất.



2. CNN (Convolutional Neural Network)

CNN, hay Convolutional Neural Network, là một loại mạng nơ-ron sâu (deep neural network) thường được sử dụng rộng rãi trong việc xử lý ảnh và nhận diện hình ảnh. CNN có khả năng học các đặc trưng cấp thấp đến cấp cao của dữ liệu hình ảnh thông qua việc sử dụng các lớp convolution (tích chập) và pooling (tích hợp).

Các thành phần cơ bản của một CNN bao gồm:

- Lớp Convolutional (Convolutional Layer): Lớp này sử dụng các bộ lọc (kernels) để thực hiện phép tích chập trên ảnh đầu vào. Bằng cách trượt các bộ lọc này qua ảnh, mô hình học được các đặc trưng như cạnh, texture, hoặc đặc điểm phức tạp hơn.

- Lớp Pooling (Pooling Layer): Lớp này thường được sử dụng sau các lớp convolution để giảm kích thước không gian của đầu ra (downsampling). Phổ biến nhất là lớp max pooling, trong đó chỉ giữ lại giá trị lớn nhất trong mỗi vùng được quét.

- Lớp Activation (Activation Layer): Sau mỗi lớp convolutional hoặc pooling, một hàm kích hoạt như ReLU (Rectified Linear Activation) thường được áp dụng để tạo tính phi tuyến tính và đưa ra quyết định về sự kích hoạt của nơ-ron.

- Lớp Fully Connected (Fully Connected Layer): Sau các lớp convolutional và pooling, thông thường sẽ có một hoặc nhiều lớp fully connected ở cuối để thực hiện phân loại hoặc dự đoán dữ liệu dựa trên đặc trưng đã học.

Ưu điểm của CNN bao gồm:

- Giữ được thông tin không gian: CNN giữ lại thông tin không gian của ảnh, giúp học các đặc trưng cục bộ.

- Chia sẻ trọng số (weight sharing): Các tham số của bộ lọc được chia sẻ, giúp giảm đáng kể số lượng tham số cần học.

- Khả năng học hiệu quả các đặc trưng hình ảnh: CNN có khả năng học các đặc trưng phức tạp từ dữ liệu hình ảnh một cách tự động.

CNN đã có những đóng góp quan trọng trong nhiều ứng dụng như nhận diện khuôn mặt, nhận diện vật thể, xe tự lái, xử lý ảnh y khoa, và nhiều lĩnh vực khác liên quan đến xử lý ảnh và dữ liệu không gian.

3. MobileNet

3.1. Lí do sử dụng

Ngày nay, với sự phát triển của sự hỗ trợ về phân cứng như GPU, TPU,… ngày ngày CNN càng sâu, càng nhiều tham số, càng phức tạp để tăng độ chính xác lên cao hơn. Đã có những mô hình khổng lồ từ vài trăm triệu tham số (parameter) đến hàng tỷ tham số.

Tham số của 1 mô hình bao gồm:

- Mult-Adds (Số lượng phép toán)
- Parameters (Số lượng trọng số

Ví dụ: với bài toán hồi quy tuyến tính (Linear Regression) với đầu vào x, đầu ra là y. Ta có mô hình như sau:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Trong đó:

- $\hat{y}$: là giá trị cần dự đoán

- θ: tham số của mô hình

Nếu có n phép cộng thì Mult-Adds = n

n+1 thelta (trọng số) thì Parameters = n+1

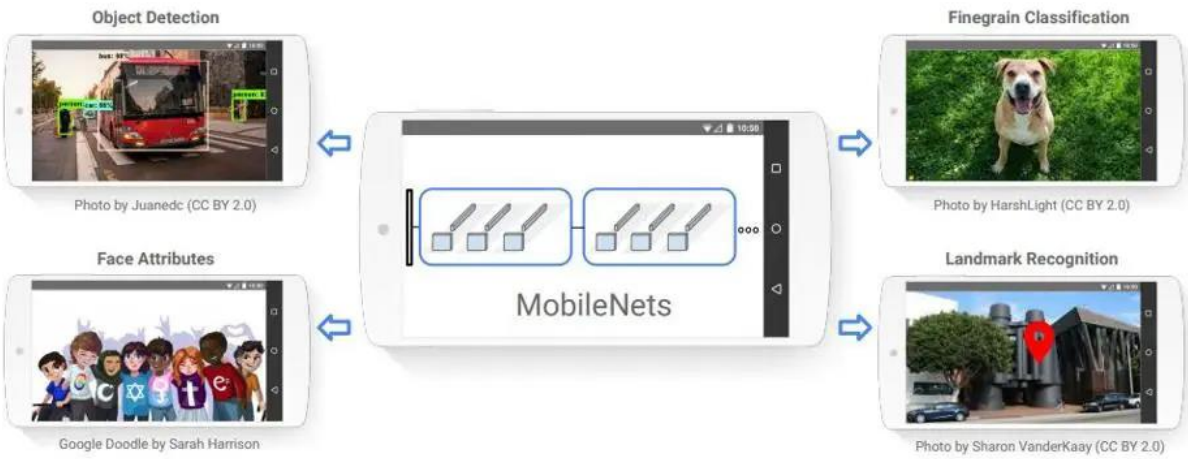
Do đó nếu:

Mô hình có ít tham số hơn thì kích thước model nhỏ hơn

Mô hình có ít phép tính cộng trừ nhân chia hơn thì độ phức tạp sẽ nhỏ hơn

Vì mục tiêu dài hạn là có thể deploy được model trên điện thoại kết hợp internet để sử dụng, nhóm em quyết định huấn luyện mô hình sử dụng kiến trúc MobileNet

3.2. Kiến trúc MobileNet



Hình 3: Một số ứng dụng của mô MobileNets

MobileNet là kiến trúc được phát triển dựa trên cấu trúc CNN được giới thiệu lần đầu vào năm 2017 [1]. Mô hình MobileNet sử dụng cách tính tích chập có tên là

Depthwise Separable Convolution để giảm kích thước mô hình và giảm độ phức tạp tính toán. Do đó mô hình sẽ hữu ích khi chạy các ứng dụng trên thiết bị nhúng hoặc di động (edge device).

Mô hình kiến trúc:

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Hình 4. MobileNet Body Architecture [2]

Chúng ta thấy rằng mô hình có 30 lớp với các đặc điểm mỗi Depthwise layer sẽ được theo sau là Pointwiselayer:

Lớp 1: Convolution layer với kích thước stride bằng 2

Lớp 2: Depthwise layer

Lớp 3: Pointwise layer

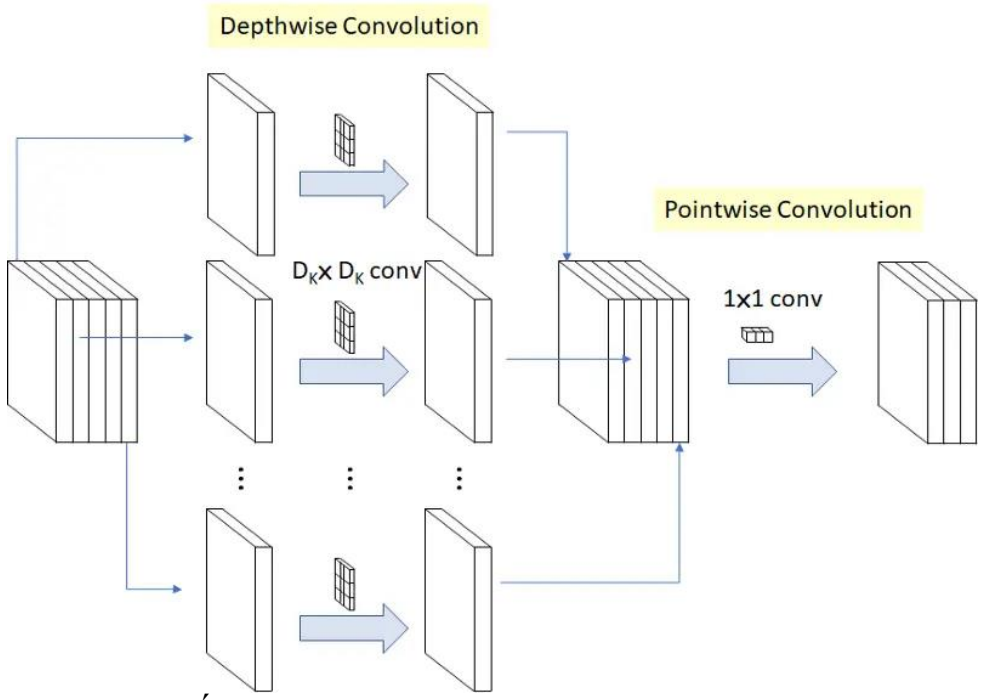
Lớp 4: Depthwise layer với stride bằng 2 (khác với lớp 2, dw lớp 2 có kích thước stride bằng 1)

Từ lớp 5 đến lớp 27: Depthwise layer và Pointwise layer với stride tương ứng được xếp xen kẽ nhau.
Lớp 28: Average pooling với stride bằng 1 giúp giảm độ phức tạp của mô hình, giảm số lượng tham số và tính toán nhưng vẫn giữ lại các đặc trưng quan trọng.
Lớp 29: fully connected với stride bằng 1

Lớp 30: Softmax dùng để phân lớp.

3.3. Phân tích chi tiết Depthwise Separable Convolution

Depthwise Separable Convolutions chia CNN cơ bản ra làm hai phần: **Depthwise Convolution** theo sau bởi **Pointwise Convolution** được mô tả như hình bên dưới:



Hình 5: Cấu trúc của một Depthwise Separable Convolution

- Depthwise convolution: là một channel-wise D_kx D_k spatial convolution (sử dụng tích chập với từng kênh ở đầu vào). Ví dụ ở hình trên, ta có 5 channels do đó chúng ta sẽ có 5 D_kx D_k spatial convolution tương ứng với 5 channel trên.

- Pointwise convolution: đơn giản là một convolution có kích thước 1x1 (như hình ở trên).

- Với M là số lượng input channel, N là số lượng output channel, D_k là kernel size, D_f là feature map size (với dataset ImageNet chọn để train thì input có kích thước là 224x224, do đó feature map ban đầu có D_f = 224), chúng ta có thể tính được:

Chi phí tính toán của Depthwise convolution là:

$$D_k . D_k . M . D_f . D_f$$

Chi phí tính toán của Pointwise convolution là:

$$M . N . D_f . D_f$$

Tổng chi phí tính toán của Depthwise Separable Convolution là:

$$D_k . D_k . M . D_f . D_f + M . N . D_f . D_f$$

Nếu chúng ta không sử dụng Depthwise Separable Convolution mà sử dụng thuần phép convolution, chi phí tính toán là:

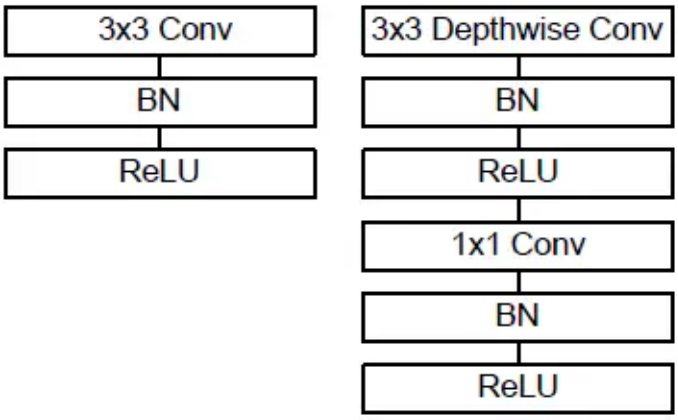
$$D_k . D_k . M . N . D_f . D_f$$

Do đó, chi phí tính toán sẽ giảm:

$$\frac{D_k . D_k . M . D_f . D_f + M . N . D_f . D_f}{D_k . D_k . M . N . D_f . D_f} = \frac{1}{N} + \frac{1}{{D_k}^2}$$

Giả sử, chúng ta chọn kernel size D_k = 3, chúng ta sẽ giảm từ 8 đến 9 lần phép tính nhân => giảm chi phí tính toán đi rất nhiều.

Chú ý: về kiến trúc MobileNet, sau mỗi convolution MobileNet sẽ sử dụng Batch Normalization (BN) và ReLU như hình bên dưới:



Standard Convolution bên trái, Depthwise separable convolution với BN và ReLU bên phải

- Hiệu ứng chính của Batch Normalization (BN) là giúp cho việc huấn luyện nhanh hơn và tránh hiện tượng vanishing gradient hoặc exploding gradient, đặc biệt khi mạng nơ-ron sâu và phức tạp.

- ReLU (Rectified Linear Activation) là một hàm kích hoạt phi tuyến tính được sử dụng rộng rãi trong các lớp ẩn của mạng nơ-ron sâu. Hàm ReLU đơn giản là f(x) = max(0, x), tức là giữ lại giá trị dương và đặt giá trị âm bằng 0. ReLU giúp giải quyết vấn đề của các hàm kích hoạt trước đó như sigmoid và tanh trong việc

giảm thiểu hiện tượng gradient vanishing. Nó không bị bão hòa ở giá trị lớn và không có đạo hàm tại các giá trị âm, giúp mô hình học nhanh hơn.

BN và ReLU thường được sử dụng kết hợp với nhau trong các kiến trúc mạng nơ-ron sâu (như CNNs) để cải thiện tính ổn định và tốc độ học của mô hình, cũng như để tránh các vấn đề như vanishing gradient và overfitting trong quá trình huấn luyện.

3.4. Đánh giá

So sánh kết quả của việc sử dụng mạng 30 layer sử dụng thuần Convolution và mạng 30 layer sử dụng Depthwise Separable Convolution (MobileNet) trên tập dữ liệu ImageNet, ta có bảng kết quả bên dưới:

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Hình 6. Depthwise Separable vs Full Convolution MobileNet [3]

MobileNet giảm 1% độ chính xác, nhưng số lượng tham số của mô hình và số lượng phép tính toán giảm đi rất rất nhiều, gần xấp xỉ 90% (tham số giảm từ 29.3 triệu xuống 4.2 triệu, số lượng phép tính toán giảm từ 4866 xuống 569). Giúp tối ưu tài nguyên cho thiết bị

3.5. So sánh MobileNet với các State-of-the-art đương thời

Khi so sánh 1.0 MobileNet-224 với GoogleNet và VGG 16 (hình bên dưới), ta thấy rằng độ chính xác của cả 3 thuật toán là hầu như tương đương nhau. Nhưng 1.0 MobileNet-224 có số lượng tham số ít (75% so với GoogleNet) và số lượng phép toán nhỏ hơn rất nhiều vì thế tốc độ xử lý của chương trình sẽ nhanh hơn.

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Hình 7: So sánh 1.0 MobileNet-224 với GoogleNet và VGG 16 trên tập ImageNet

C. Training

Nhóm em thực hiện việc xây dựng một mô hình mạng nơ-ron sử dụng kiến trúc MobileNet để huấn luyện trong việc phân loại hình ảnh trong bài toán phân loại cảm xúc (emotion classification) giúp chuẩn đoán sớm bệnh tử kỷ ở trẻ em. Dưới đây là phân tích về từng bước xây dựng mô hình:

1. Dataset

Bộ dữ liệu cảm xúc được nhóm em thu thập từ một số nguồn từ Internet, đặc biệt là trên nền tảng Kaggle. Kaggle là một nền tảng khoa học dữ liệu và cộng đồng trực tuyến gồm các nhà khoa học dữ liệu và người thực hành máy học thuộc Google LLC. Kaggle cho phép người dùng tìm và xuất bản tập dữ liệu, khám phá và xây dựng mô hình trong môi trường khoa học dữ liệu dựa trên web, làm việc với các nhà khoa học dữ liệu và kỹ sư máy học khác, đồng thời tham gia các cuộc thi để giải quyết các thách thức về khoa học dữ liệu. Bộ datasets phục vụ huấn luyện bao gồm 5 phần chính, đại diện cho 5 trạng thái cảm xúc: ‘Angry’; ‘Happy’; ‘Neutral’; ‘Sad’ và ‘Surprise’. Tổng có hơn 6000 tấm ảnh được dán nhãn theo đúng trạng thái cảm xúc.

Hình 8: 5 trạng thái cảm xúc được huấn luyện trong mô hình

2. Xây dựng mô hình

1. Import các thư viện và modules cần thiết từ Keras và TensorFlow:

```
from keras.applications import MobileNet
from keras.models import Sequential,Model
from keras.layers import Dense,Dropout,Activation,Flatten,GlobalAveragePooling2D
from keras.layers import Conv2D,MaxPooling2D,ZeroPadding2D
from keras.preprocessing.image import ImageDataGenerator
```

- Model sẽ sử dụng các layer: 2D Convolution (Conv2D), GlobalAveragePooling2D, fully connected (Dense)
- Kết hợp lớp Dense với activation='relu' và activation='softmax' trong một mô hình, nhóm em sử dụng ReLU để học các đặc trưng ở các lớp ẩn (hidden layers) và sử dụng Softmax ở lớp cuối cùng để tạo ra dự đoán, phân loại xác suất cho nhiều lớp.

2. Tạo một mô hình MobileNet được cấu hình để làm việc với hình ảnh đầu vào kích thước 224x224 pixels với 3 kênh màu (RGB).

```
# MobileNet is designed to work with images of dim 224,224
img_rows,img_cols = 224,224

MobileNet = MobileNet(weights='imagenet',include_top=False,input_shape=(img_rows,img_cols,3))
```

- Nhóm em sử dụng trọng số được huấn luyện trước của ImageNet (ImageNetWeight) bởi:

+ Trọng số ImageNet giúp giảm thiểu thời gian cần thiết để huấn luyện mô hình trên tập dữ liệu mới. Thay vì huấn luyện từ đầu, ta có thể bắt đầu từ trạng thái đã được học trước và chỉ cần điều chỉnh một số trọng số cho phù hợp với bài toán của nhóm.

+ Trọng số được huấn luyện trước từ ImageNet cung cấp một trạng thái ban đầu tốt cho mô hình. Điều này giúp mô hình học sâu khởi đầu với khả năng học tốt hơn, do đã học được các đặc trưng cơ bản từ dữ liệu lớn ImageNet.

3. Đóng băng (freeze) các layer cuối cùng của MobileNet, chỉ định rằng chỉ các lớp đầu tiên sẽ được huấn luyện lại trong quá trình tinh chỉnh (fine-tuning)

```
for layer in MobileNet.layers:
    layer.trainable = True

# Let's print our layers
for (i,layer) in enumerate(MobileNet.layers):
    print(str(i),layer.__class__.__name__,layer.trainable)
```

4. Xây dựng phần đỉnh (top) của mô hình sử dụng kiến trúc MobileNet

```
def addTopModelMobileNet(bottom_model, num_classes):

    top_model = bottom_model.output
    top_model = GlobalAveragePooling2D()(top_model)
    top_model = Dense(1024,activation='relu')(top_model)

    top_model = Dense(1024,activation='relu')(top_model)

    top_model = Dense(512,activation='relu')(top_model)

    top_model = Dense(num_classes,activation='softmax')(top_model)

    return top_model

num_classes = 5
```

Hàm addTopModelMobileNet:

Đây là một hàm được định nghĩa để tạo phần đỉnh (top) của mô hình. Hàm này nhận hai tham số: bottom_model - mô hình cơ bản (phần dưới) và num_classes - số lượng lớp đầu ra.

Hàm sử dụng output của bottom_model (bottom_model.output) như là đầu vào để xây dựng phần đỉnh mới của mô hình.

Xây dựng phần đỉnh mới của mô hình:

- top_model = bottom_model.output: Đầu tiên, lấy output của bottom_model và gán cho top_model. Điều này sẽ sử dụng output của mô hình MobileNet (phần dưới) để tiếp tục xây dựng phần đỉnh.
- top_model = GlobalAveragePooling2D()(top_model): Sử dụng lớp GlobalAveragePooling2D để thực hiện global average pooling trên top_model. Điều này sẽ chuyển đổi dữ liệu từ tensor 3D thành vector 1D bằng cách tính trung bình cộng của tất cả các giá trị trong mỗi kênh (channel).

Tiếp theo, sử dụng một chuỗi các lớp Dense để thêm các lớp fully connected:

- top_model = Dense(1024, activation='relu')(top_model): Thêm một lớp Dense với 1024 units và hàm kích hoạt là ReLU.
- top_model = Dense(512, activation='relu')(top_model): Thêm một lớp Dense với 512 units và hàm kích hoạt ReLU.
- top_model = Dense(num_classes, activation='softmax')(top_model): Cuối cùng, thêm một lớp Dense với số units bằng num_classes (trong trường hợp này là 5 cảm xúc: Happy, Sad, Angry, Supprise, Neutral) và hàm kích hoạt softmax để đưa ra xác suất dự đoán cho từng lớp.

Trả về phần đỉnh mới của mô hình:

Hàm sẽ trả về top_model, nó sẽ được sử dụng làm output của mô hình tổng thể.

Xây dựng mô hình hoàn chỉnh:

```
FC_Head = addTopModelMobileNet(MobileNet, num_classes)

model = Model(inputs = MobileNet.input, outputs = FC_Head)

print(model.summary())
```

Sử dụng Model API từ Keras để tạo mô hình hoàn chỉnh bằng cách kết hợp MobileNet.input (đầu vào của MobileNet) và FC_Head (phần đỉnh mới vừa được tạo ra).

3. Chuẩn bị dữ liệu huấn luyện và kiểm tra

1.

```
train_data_dir = './train'
validation_data_dir = './validation'
```

Định nghĩa đường dẫn và tạo generators cho dữ liệu huấn luyện và kiểm tra:

- Đường dẫn tới thư mục chứa dữ liệu huấn luyện (train_data_dir) và dữ liệu kiểm tra (validation_data_dir) được xác định.

```
train_datagen = ImageDataGenerator(
                    rescale=1./255,
                    rotation_range=30,
                    width_shift_range=0.3,
                    height_shift_range=0.3,
                    horizontal_flip=True,
                    fill_mode='nearest'
                                )

validation_datagen = ImageDataGenerator(rescale=1./255)
```

- ImageDataGenerator được sử dụng để tạo generators cho dữ liệu huấn luyện và kiểm tra. Các tham số như rescale, rotation_range, width_shift_range, height_shift_range, horizontal_flip, fill_mode được thiết lập để tăng cường dữ liệu.

2. Thiết lập batch size và tạo generators cho dữ liệu:

```
batch_size = 32

train_generator = train_datagen.flow_from_directory(
                        train_data_dir,
                        target_size = (img_rows,img_cols),
                        batch_size = batch_size,
                        class_mode = 'categorical'
                        )

validation_generator = validation_datagen.flow_from_directory(
                            validation_data_dir,
                            target_size=(img_rows,img_cols),
                            batch_size=batch_size,
                            class_mode='categorical')
```

- batch_size được định nghĩa là 32. Đây là kích thước của mỗi batch dữ liệu được sử dụng trong quá trình huấn luyện.

- train_datagen.flow_from_directory và validation_datagen.flow_from_directory được sử dụng để tạo generators từ các thư mục dữ liệu huấn luyện và kiểm tra tương ứng.

4. Huấn luyện mô hình

1. Thiết lập các callbacks như ModelCheckpoint, EarlyStopping, ReduceLROnPlateau để lưu lại model tốt nhất, dừng sớm để tránh overfitting và điều chỉnh learning rate.

```
from keras.optimizers import RMSprop,Adam
from keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceLROnPlateau

checkpoint = ModelCheckpoint(
                              'emotion_face_mobilNet.h5',
                              monitor='val_loss',
                              mode='min',
                              save_best_only=True,
                              verbose=1)

earlystop = EarlyStopping(
                            monitor='val_loss',
                            min_delta=0,
                            patience=10,
                            verbose=1,restore_best_weights=True)

learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=5,
                                            verbose=1,
                                            factor=0.2,
                                            min_lr=0.0001)

callbacks = [earlystop,checkpoint,learning_rate_reduction]
```

- Các callbacks như ModelCheckpoint, EarlyStopping, ReduceLROnPlateau được khởi tạo và lưu vào biến callbacks.

+ ModelCheckpoint sẽ lưu mô hình có val_loss tốt nhất.

+ EarlyStopping sẽ dừng huấn luyện sớm nếu val_loss không cải thiện sau một số epochs nhất định.

+ ReduceLROnPlateau sẽ giảm learning rate nếu val_acc không cải thiện sau một số epochs nhất định.

2.

```
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=0.001),
              metrics=['accuracy']
              )
```

Compile mô hình với loss function là categorical_crossentropy, optimizer là Adam với learning rate là 0.001 và metrics là accuracy.

- Compile mô hình: Mô hình được biên dịch (compile) với loss function là categorical_crossentropy, optimizer là Adam với learning rate là 0.001 và metrics là accuracy.

3. Định nghĩa số lượng mẫu huấn luyện và kiểm tra:

```
nb_train_samples = 24176
nb_validation_samples = 3006

epochs = 25

history = model.fit_generator(
            train_generator,
            steps_per_epoch=nb_train_samples//batch_size,
            epochs=epochs,
            callbacks=callbacks,
            validation_data=validation_generator,
            validation_steps=nb_validation_samples//batch_size)
```

- nb_train_samples và nb_validation_samples định nghĩa số lượng mẫu huấn luyện và kiểm tra tương ứng.

- model.fit_generator được sử dụng để huấn luyện mô hình. Quá trình huấn luyện sẽ diễn ra trong epochs lượt với dữ liệu từ train_generator và validation_generator. Các callbacks đã được định nghĩa trước đó cũng được sử dụng để theo dõi quá trình huấn luyện và điều chỉnh mô hình.

4. Cuối cùng lưu model có performance tốt nhất sau quá trình huấn luyện với tên ‘Emotion_Detection.h5’ để sử dụng trong chương trình.

D. Sử dụng model để đưa ra chuẩn đoán.

Với tính năng đưa ra chuẩn đoán về trẻ có bị tự kỷ hay không, nhóm em sử dụng model đã được train để áp dụng vào bài toán thực tế. Chương trình sử dụng mô hình để nhận diện các trạng thái cảm xúc như vui mừng, buồn bã, tức giận, bình thường và ngạc nhiên từ đó đưa ra chuẩn đoán sơ bộ có dấu hiệu tự kỷ hay không.

Chương trình đưa ra dự đoán trẻ tự kỷ có 6 phần chính gồm:

+ Khởi tạo mô hình: Sử dụng thư viện Keras, chương trình tải mô hình đã được huấn luyện trên dữ liệu cảm xúc và kết hợp với một bộ lọc khuôn mặt sử dụng Haarcascades.

+ Xử lý video: Đọc từng khung hình từ video đầu vào và chuyển đổi ảnh màu sang ảnh đen trắng để tối ưu hóa quá trình nhận diện khuôn mặt.

+ Nhận diện khuôn mặt: Sử dụng bộ lọc khuôn mặt, chương trình xác định vị trí của khuôn mặt trong khung hình và vẽ khung xung quanh chúng.

+ Tiền xử lý dữ liệu: Resize và chuẩn hóa kích thước khuôn mặt để đưa vào mô hình nhận diện cảm xúc.

+ Dự đoán cảm xúc: Sử dụng mô hình đã tải, chương trình dự đoán cảm xúc từ khuôn mặt và hiển thị kết quả trực tiếp lên video.

+ Đưa ra kết luận: Theo dõi và lưu trữ các dự đoán cảm xúc để đưa ra kết luận cuối cùng về trạng thái của trẻ được chuẩn đoán.

1. Khởi tạo mô hình:

Phần này tập trung vào quá trình khởi tạo mô hình sử dụng thư viện Keras và tích hợp nó với bộ lọc khuôn mặt Haarcascades để xây dựng hệ thống nhận diện cảm xúc.

```
from keras.models import load_model
from keras.preprocessing.image import img_to_array
import cv2
import numpy as np

face_classifier = cv2.CascadeClassifier('./haarcascade_frontalface_default.xml')
classifier =load_model('./Emotion_Detection.h5')
videopath = './abnormal.mp4'
class_labels = ['Angry','Happy','Neutral','Sad','Surprise']
```

Mô hình huấn luyện trước có tên Emotion_Detection.h5 đã được tải bằng hàm 'load_model'. Đồng thời bộ lọc khuôn mặt cũng được tải bằng hàm CascadeClassifier của thư viện OpenCv. Bộ lọc Haarcascades có tên là 'Haarcascade_frontalface_default.xml'. Đây là một tập tin XML chứa thông tin về các đặc điểm của khuôn mặt để phát hiện chúng trong ảnh. Đồng thời khởi tạo một mảng gồm 5 giá trị là các trạng thái cơ bản của cảm xúc. Quá trình này cho phép kết hợp sức mạnh của mô hình học sâu trong việc nhận diện cảm xúc với khả năng xác định khuôn mặt một cách chính xác nhờ vào bộ lọc khuôn mặt.

2. Xử lý video.

Phần này chịu trách nhiệm đọc từng khung hình từ video đầu vào và chuẩn bị chúng cho quá trình nhận diện cảm xúc.

```
cap = cv2.VideoCapture(1)
status = []


while True:
    # Lấy một khung hình video
    ret, frame = cap.read()
    if ret is False or frame is None:  # Kiểm tra xem frame có giá trị hay không
        break
```

Sử dụng hàm 'read' của OpenCV để đọc mỗi khung hình từ video. Kết quả trả về chứa hai thông tin: 'ret' là một cờ cho biết liệu việc đọc có thành công hay không, và 'frame' là khung hình đó. Kiểm tra xem khung hình có giá trị hay không. Nếu không, chương trình thoát khỏi vòng lặp, kết thúc quá trình xử lý video.

3. Nhận Diện Khuôn Mặt.

Phần này sử dụng bộ lọc khuôn mặt để xác định vị trí của khuôn mặt trong từng khung hình.

```
gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
faces = face_classifier.detectMultiScale(gray,1.3,5)
```

Ảnh sẽ được chuyển sang đen trắng để giảm chiều sâu của dữ liệu và tang hiệu suất của bộ lọc khuôn mặt. Sau đó sử dụng hàm ‘detectMultiScale’ để xác định vị trí của khuôn mặt trong ảnh đen trắng. Kết quả trả về sẽ là hình chữ nhật chứa tọa độ của khuôn mặt.

4. Tiền xử lý dữ liệu.

Phần này chuẩn bị dữ liệu của khuôn mặt để đưa vào mô hình nhận diện cảm xúc.

```
for (x,y,w,h) in faces:
    cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h,x:x+w]
    roi_gray = cv2.resize(roi_gray,(48,48),interpolation=cv2.INTER_AREA)


    if np.sum([roi_gray])!=0:
        roi = roi_gray.astype('float')/255.0
        roi = img_to_array(roi)
        roi = np.expand_dims(roi,axis=0)
```

Một hình chữ nhật được vẽ tại nơi khuôn mặt được xác định. Biến ‘roi_gray’ được khởi tạo bằng cách trích xuất một khu vực của ảnh đen trắng ‘gray’ trước đó. Sau đó sử dụng hàm ‘resize’ để thay đổi kích thước của khu vực ‘roi_gray’ về 48x48. Quá trình này giúp đảm bảo ảnh đầu vào cho mô hình nhận diện cảm xúc có kích thước chuẩn. Sau khi thực hiện ‘resize’ vùng ‘roi_gray’, chương trình sẽ kiểm tra vùng ‘roi_gray’ có chứa thông tin không. Nếu có thì tiến hành chuẩn hóa dữ liệu bằng cách chia giá trị pixel cho 255.0, sau đó chuyển đổi thành mảng numpy và

mở rộng chiều của mảng để tạo thành dữ liệu đầu vào cho mô hình nhận diện cảm xúc.

5. Dự đoán cảm xúc.

Sử dụng mô hình để dự đoán cảm xúc từ khu vực quan tâm ‘Roi’ và hiển thị kết quả lên khung hình.

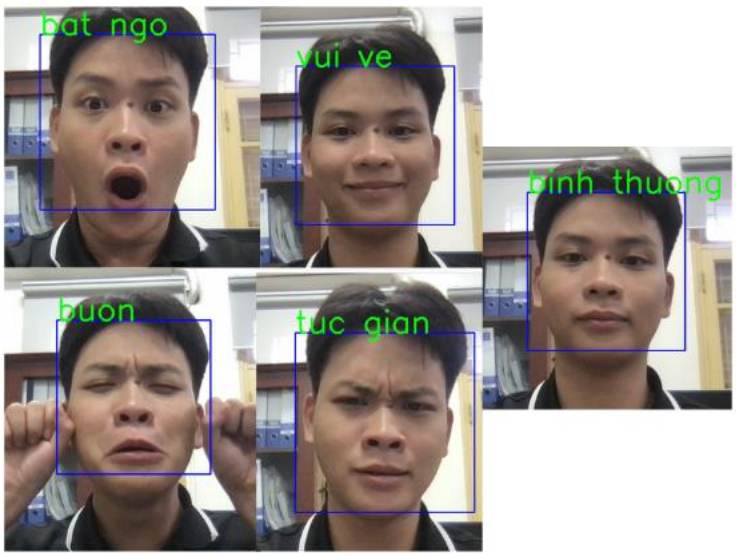
```
    preds = classifier.predict(roi)[0]
    print("\nprediction = ",preds)
    label=class_labels[preds.argmax()]
    print("\nprediction max = ",preds.argmax())
    status.append(preds.argmax())
    print("\nlabel = ",label)
    label_position = (x,y)
    cv2.putText(frame,label,label_position,cv2.FONT_HERSHEY_SIMPLEX,2,(0,255,0),3)
else:
    cv2.putText(frame,'No Face Found',(20,60),cv2.FONT_HERSHEY_SIMPLEX,2,(0,255,0),3)
```

Sử dụng mô hình đã được tải ‘classifier’ để dự đoán cảm xúc từ dữ liệu đầu vào ‘roi’. Kết quả trả về là một mảng xác suất của các lớp cảm xúc. Nhãn tương ứng của cảm xúc sẽ được xác định bằng cách so sánh với chỉ số của lớp cảm xúc và đồng thời lưu trữ chỉ số của lớp cảm xúc có xác suất cao nhất vào mảng ‘status’. Điều này sẽ được sử dụng để đưa ra nhận định về trạng thái của trẻ em có nguy cơ mắc tự kỷ hay không. Và cuối cùng hàm ‘putText’ được sử dụng để hiển thị nhãn cảm xúc lên khung ảnh.

6. Đưa ra nhận định

Sau khi các giá trị của cảm xúc sau mỗi lần xác định được đưa vào mảng trạng thái. Nếu cảm xúc thay đổi nhiều lần sẽ đưa ra nhận định “có dấu hiệu của bệnh tự kỷ” và ngược lại, nếu không có quá nhiều trạng thái cảm xúc thì sẽ đưa ra nhận định “bình thường”.

E. Kết quả.



Hình 9: Các trạng thái cảm xúc được nhận diện

F. Đánh giá.

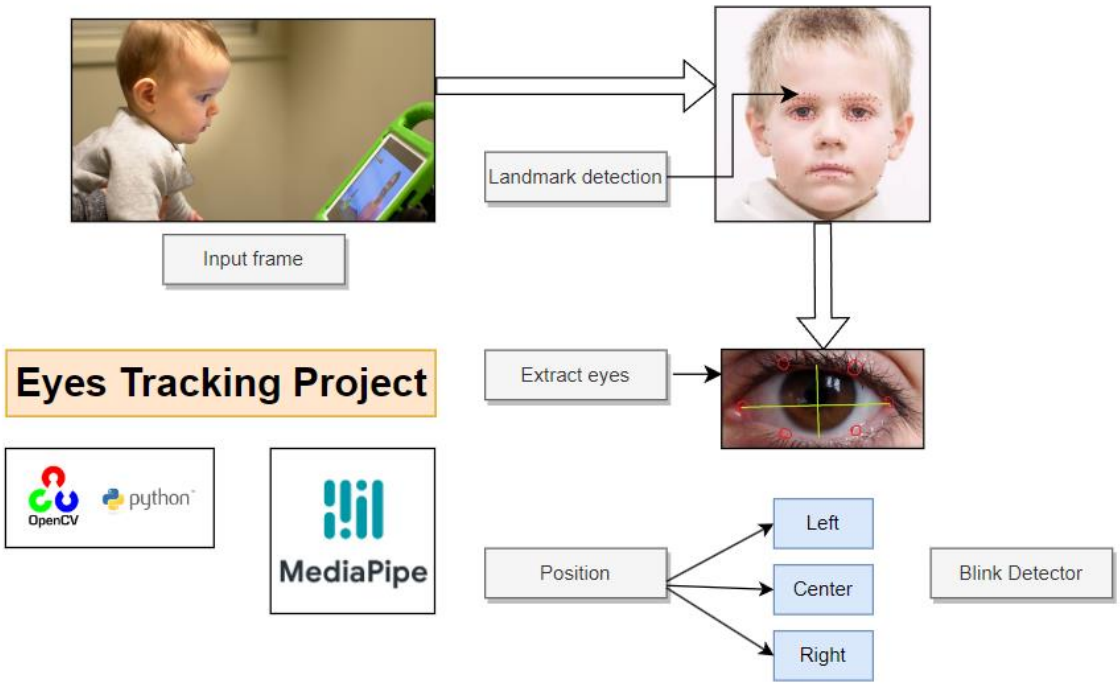
Nhìn chung, chương trình theo dõi cảm xúc trên khuôn mặt để đưa ra nhận định về dấu hiệu của bệnh tự kỷ ở trẻ chạy tốt, nhận diện đúng các trạng thái cảm xúc ở thực tế. Hiệu suất ổn định khi chạy với những video có số khung hình/giây thấp. Tuy nhiên, chương trình còn gặp hạn chế ở một số thời điểm do không nhận diện được khuôn mặt và cần tham khảo ý kiến của bác sĩ, chuyên gia để có số liệu cụ thể để đưa nhận định chính xác hơn.

PHẦN 4. Eye tracking

A. Giới thiệu

Dự án này nhằm áp dụng kỹ thuật Eye Tracking, sử dụng thư viện MediaPipe và OpenCV, để xác định các biểu hiện đặc trưng bệnh nhãn khoa thông qua phân tích hành vi nhìn của trẻ.

B. Tổng quan về mô hình thực hiện



Hình 10. Mô hình tổng quan của hệ thống eye tracking

Tổng quan về mô hình thực hiện trong dự án phát hiện trẻ có bệnh nhãn khoa và dấu hiệu tự kỷ thông qua Eye Tracking có thể được mô tả như sau:

- Thư viện Mediapipe của Google và OpenCV thực hiện việc nhận diện và theo dõi các điểm đặc trưng trên khuôn mặt, đặc biệt là các điểm liên quan đến mắt.

- Landmarks Detection: Đối với mỗi khung hình từ video, mô hình sẽ sử dụng Mediapipe để xác định vị trí của các điểm đặc trưng trên khuôn mặt, đặc biệt là vị trí của mắt.

- Dựa trên vị trí của các điểm đặc trưng, mô hình tính toán tỷ lệ nháy mắt (blink ratio) bằng cách đo khoảng cách giữa các điểm liên quan đến mắt.

- Eyes Position Estimation: Sử dụng ảnh nhị phân của mắt, mô hình ước lượng vị trí của mắt trong khung hình, xác định xem mắt ở vị trí nào (trái, giữa, phải).

- Đánh giá: Dựa trên tỷ lệ nháy mắt và vị trí của mắt, mô hình sẽ cho biết việc trẻ có đang nháy mắt hay không và thu thập dữ liệu chuyển động của nhãn cầu. Sau đó, dự án thực hiện đánh giá bằng cách đếm số lần nháy mắt, so sánh dữ liệu chuyển động của mắt và dựa vào đó đưa ra kết luận về tình trạng sức khỏe của trẻ.

- Hiển thị kết quả: Kết quả cuối cùng được hiển thị trực tiếp trên video, bao gồm cả tỷ lệ nháy mắt, số lần nháy mắt và đánh giá về tình trạng sức khỏe của trẻ.

Tóm lại, mô hình kết hợp nhiều kỹ thuật xử lý ảnh và tính toán để theo dõi và đánh giá hành vi nháy mắt của trẻ, từ đó đưa ra nhận định về tình trạng sức khỏe nhãn khoa của trẻ.

C. Công Nghệ và Thư Viện Sử Dụng

Ngôn ngữ lập trình được sử dụng cho hệ thống là Python hai thư viện được sử dụng chính là : OpenCV và Mediapipe

OpenCV, hay Open Source Computer Vision Library, là một thư viện mã nguồn mở chuyên về thị giác máy tính và xử lý ảnh. Được phát triển bởi Intel, thư viện này đã trở thành một công cụ quan trọng cho nhiều ứng dụng thị giác máy tính, xử lý ảnh, và thị giác máy học.

Mediapipe là một framework mã nguồn mở được cung cấp bởi Google, chuyên về phát hiện và theo dõi nhiều loại đặc trưng, bao gồm khuôn mặt, tay, đôi mắt. Nó có một loạt các công cụ để phát hiện và theo dõi các điểm đặc trưng trong thời gian thực

D. Thuật toán và phương pháp

1. Xử Lý Ảnh và Nhận Diện Đặc Trưng Khuôn Mặt

Xử lý hình ảnh:

"Xử Lý Hình Ảnh" trong dự án bao gồm một số bước quan trọng để chuẩn bị dữ liệu hình ảnh trước khi áp dụng các thuật toán theo dõi và nhận diện:

Đọc Mỗi Khung Hình từ Camera:

Sử dụng hàm ‘camera.read()’ để đọc mỗi khung hình từ camera.

Mỗi khung hình được lưu trong biến ‘frame’.

```
ret, frame = camera.read()
```

Thay Đổi Kích Thước Hình Ảnh:

Sử dụng hàm **cv.resize()** để thay đổi kích thước hình ảnh, từ đó có thể trích xuất được tọa độ các điểm đặc trưng theo kích thước ảnh

Kích thước mới được đặt bằng cách nhân kích thước ban đầu với một hệ số, trong trường hợp này là **fx=1.5** và **fy=1.5**.

```
frame = cv.resize(frame, None, fx=1.5, fy=1.5, interpolation=cv.INTER_CUBIC)
```

Chuyển Đổi Màu Hình Ảnh từ RGB Sang BGR:

Hình ảnh từ camera thường được đọc dưới dạng RGB, nhưng thư viện OpenCV sử dụng định dạng BGR.

Sử dụng hàm 'cv.cvtColor()' để chuyển đổi màu từ RGB sang BGR.

```
rgb_frame = cv.cvtColor(frame, cv.COLOR_RGB2BGR)
```

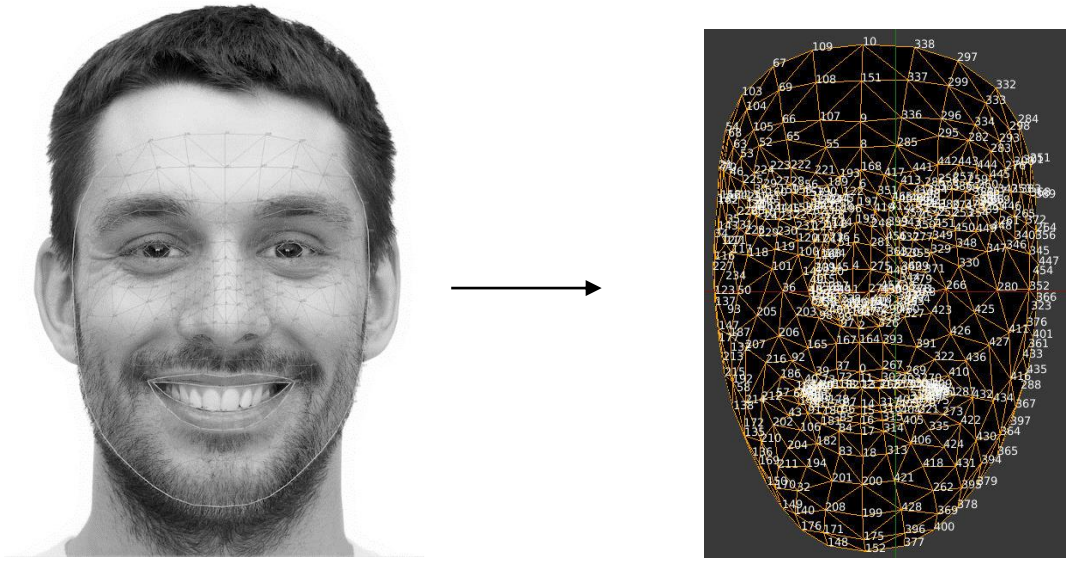
Các bước trên là quan trọng để chuẩn bị dữ liệu đầu vào cho các thuật toán xử lý hình ảnh tiếp theo, như nhận diện khuôn mặt và landmarks, và theo dõi trạng thái của đôi mắt. Điều này sẽ giúp:

- Giảm độ phức tạp
- Tăng tốc độ xử lý
- Giữ lại được những thông tin quan trọng

2. Mediapipe Face Mesh

- Sử dụng Mediapipe để nhận diện và theo dõi các điểm mốc trên khuôn mặt.
- Lấy tọa độ của các điểm mốc, đặc biệt là mắt.

Hình 11: Face landmark



3. Theo dõi nháy mắt (Blink Detection)

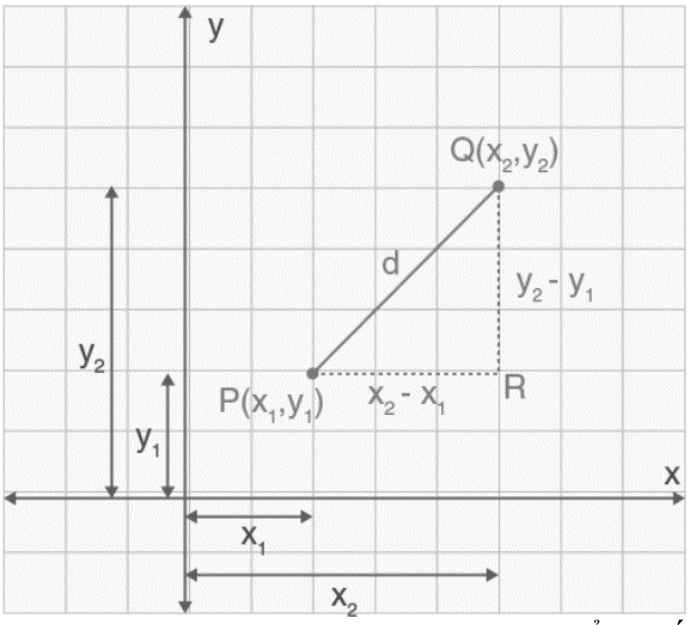
Tính toán tỷ lệ mở và đóng của mắt bằng cách sử dụng Euclidean Distance giữa các điểm mốc mắt

Euclidean Distance được sử dụng để đo lường khoảng cách giữa các điểm mốc trên khuôn mặt, đặc biệt là trong việc tính toán tỷ lệ mở và đóng của mắt. Công thức Euclidean Distance giữa hai điểm (x_1,y_1) và (x_2,y_2) trên mặt phẳng 2 chiều được tính bằng công thức:

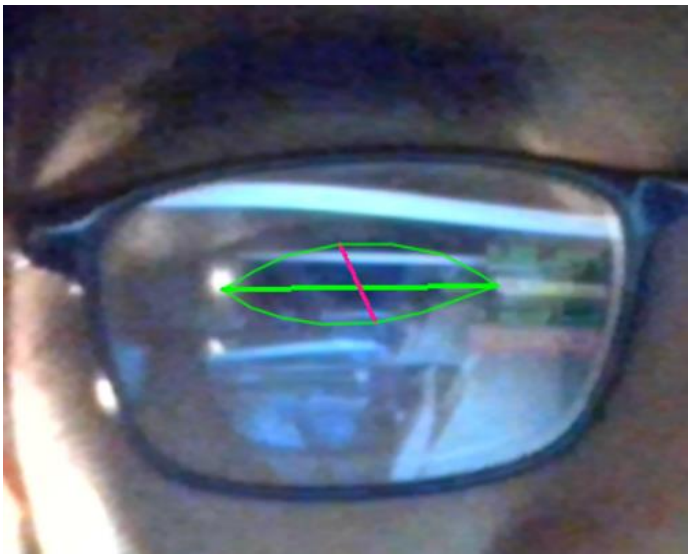
$$Distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Từ đó ta có thể tính toán được đồ dài theo chiều dọc và theo chiều ngang của mắt

Thuật toán phát hiện nháy mắt:



Sau khi tính toán các khoảng cách Euclidean giữa các điểm mốc để đo lường kích thước của mắt. Ta sử dụng các khoảng cách này để tính tỉ lệ giữa chiều dài và chiều rộng của mắt. Đối với mỗi mắt, vẽ các đường ngang và dọc để xác định kích



Hình 12. Mô hình hóa chiều dọc, chiều ngang của mắt

thước mắt. Từ đó, tính tỉ lệ chiều dài và chiều rộng, sau đó lấy trung bình của hai tỉ lệ này

$$rightRatio = \frac{\text{rhDistance}}{\text{rvDistance}}$$

$$leftRatio = \frac{\text{lhDistance}}{\text{lvDistance}}$$

$$ratio = \frac{\text{rightRatio} + \text{leftRatio}}{2}$$

Dựa vào tỉ lệ đã tính, đưa ra quyết định liệu mắt có đang nhắm hay không. Nếu tỉ lệ vượt quá một ngưỡng nhất định (trong trường hợp này là 5.5), thì được xem xét là mắt đã nhắm

Ratio > 5.5 => Blink

Nhưng để được xem là 1 lần nháy mắt thì mắt phải thực hiện đủ chu kì (mắt đóng lại và mở ra). Do đó thời gian khi mắt đã nhắm được tính toán phải lớn hơn 3 Frames (so với tốc độ khung hình đoạn video được đưa vào)

Đoạn code xử lý việc theo dõi nháy mắt:

```
def blinkRatio(img, landmarks, right_indices, left_indices):
    # Right eyes
    # horizontal line
    rh_right = landmarks[right_indices[0]]
    rh_left = landmarks[right_indices[8]]
    # vertical line
    rv_top = landmarks[right_indices[12]]
    rv_bottom = landmarks[right_indices[4]]
    # draw lines on right eyes
    cv.line(img, rh_right, rh_left, utils. GREEN, 2)
    cv.line(img, rv_top, rv_bottom, utils.PINK, 2)
```

```
# LEFT_EYE
# horizontal line
lh_right = landmarks[left_indices[0]]
lh_left = landmarks[left_indices[8]]

# vertical line
lv_top = landmarks[left_indices[12]]
lv_bottom = landmarks[left_indices[4]]

rhDistance = euclaideanDistance(rh_right, rh_left)
rvDistance = euclaideanDistance(rv_top, rv_bottom)

lvDistance = euclaideanDistance(lv_top, lv_bottom)
lhDistance = euclaideanDistance(lh_right, lh_left)

reRatio = rhDistance/rvDistance
leRatio = lhDistance/lvDistance

ratio = (reRatio+leRatio)/2
return ratio
```

4. Theo dõi hướng nhìn của mắt (Eye Position Estimation)

4.1. Blur và Loại Bỏ Nhiễu

Ảnh mắt được chuyển đổi sang ảnh xám để giảm độ phức tạp tính toán và tăng cường hiệu suất. Áp dụng các bước làm mờ, bao gồm Gaussian Blur và Median Blur, để giảm nhiễu và làm mềm ảnh.

4.2. Gaussian blur

Gaussian blur làm mờ/mịn Gaussian) được sử dụng để giảm nhiễu muối tiêu. Kỹ thuật này được sử dụng để giảm nhiễu ảnh và các chi tiết. Hiệu ứng hình ảnh của kỹ thuật Gaussian tương tự như việc nhìn một hình ảnh qua màn hình mờ. Trong thị giác máy tính, Gaussian được sử dụng để tăng cường hình ảnh ở các quy mô khác nhau hoặc như một kỹ thuật tăng cường dữ liệu trong học sâu.

Cách hoạt động của hàm làm mịn Gaussian:
Bước 1: Xây dựng lớp gaussian kernel (Để nhân với ảnh thật)
Gaussian kernel thường được tạo ra với một kích thước (kernel size) xác định (ví dụ: 3x3, 5x5, 7x7).

Trọng số của từng phần tử trong kernel được tính toán bằng cách sử dụng hàm Gaussian:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Bước 2: Áp dụng kernel lên ảnh

Từng kernel được nhân theo thứ tự lần lượt với từng ô pixel tương ứng của ảnh, và tổng các giá trị được chia cho tổng trọng số để thu được giá trị trung bình từng pixel.

4.3. Median Blur

Median Blur tính toán trung bình của cường độ pixel bao quanh pixel trung tâm trong kernel n x n. Sau đó, giá trị trung vị sẽ thay thế cường độ pixel của pixel trung tâm. Median Blur thực hiện việc loại bỏ salt-and-pepper noise tốt hơn so với Gaussian filter

Tuy nhiên, trong thư viện OpenCV đã hỗ trợ xử lý với 2 kĩ thuật này. Việc còn lại là điều chỉnh các tham số sao cho phù hợp

```
# remove the noise from images
gaussain_blur = cv.GaussianBlur(cropped_eye, (9,9),0)
median_blur = cv.medianBlur(gaussain_blur, 3)
```

4.4. Thresholding và Chuyển Đổi Ảnh Nhị Phân

Áp dụng ngưỡng để chuyển đổi ảnh thành ảnh nhị phân:

Sử dụng một quy trình ngưỡng để chuyển đổi ảnh xám thành ảnh nhị phân. Nếu các giá trị điểm ảnh lớn hơn ngưỡng sẽ đưa về giá trị 255 (màu trắng), nếu nhỏ hơn ngưỡng sẽ đưa về giá trị 0 (màu đen). Ngưỡng được lựa chọn trong đoạn mã là 130

```
# applying thrsholding to convert binary_image
ret, threshed_eye = cv.threshold(median_blur, 130, 255, cv.THRESH_BINARY)
```

Sử dụng ngưỡng để tạo ra mặt nạ cho phần mắt

Tạo một mặt nạ (mask) màu đen (là các giá trị 0) với kích thước tương tự với ảnh xám. Vùng mắt được vẽ trên mặt nạ với màu trắng, sử dụng toán tử bitwise AND giữa ảnh xám và mặt nạ (giữ lại các phần tử # 0)

Hình 16. Vùng mặt nạ sau khi được tạo

Lớp mặt nạ này sẽ giúp giữ lại các điểm ảnh trong vùng mắt để ta có thể tập trung xử lý

4.5. Xác định hướng nhìn của mắt (Eye Position Estimation)

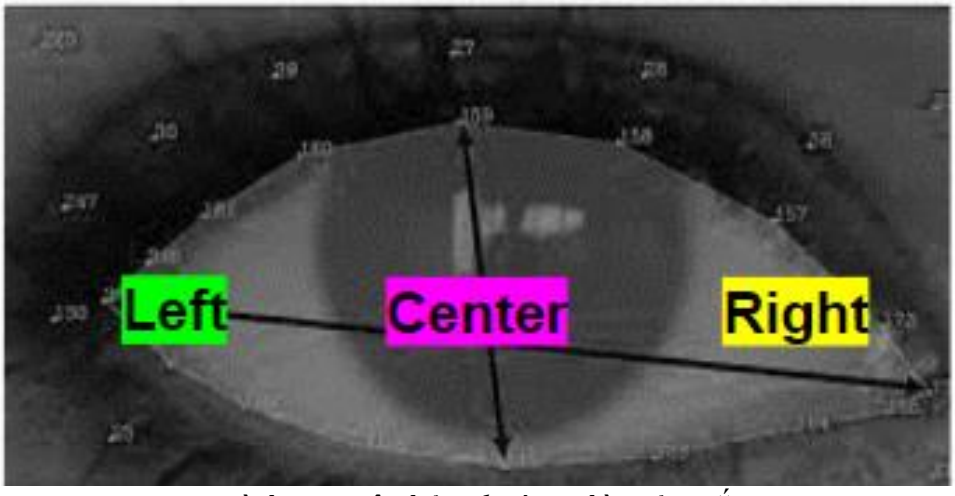
Bước 1: Chia Mắt Thành Ba Phần:

Mắt được chia thành ba phần bằng cách cắt theo chiều ngang thành ba đoạn bằng nhau.

Bước 2: Xác định vị trí mắt

Xác định số lượng chấm Đen Pixel: Số lượng pixel đen được đếm trong mỗi phần (phần phải, phần giữa và phần trái).

Dựa vào số lượng pixel đen trong mỗi phần, xác định vị trí của mắt là "LEFT," "CENTER," hoặc "RIGHT." Nếu phần nào có số lượng pixel đen lớn nhất, vị trí của mắt được xác định là phần tương ứng.



Hình 13: Mô phỏng hướng nhìn của mắt

E. Phương pháp chẩn đoán bệnh về mắt và phát hiện trẻ tự kỷ

1. Chẩn đoán bệnh về mắt

Sau khi đã detect và tracking việc nháy mắt và hướng nhìn của mắt, từ đó có thể đưa ra nhận định về tình trạng sức khỏe nhãn khoa của trẻ:

Phát hiện bệnh nhờ đo chuyển động của mắt:

Khi cho người kiểm tra xem một video chuyển động của một chấm tròn trên màn hình, chúng ta sẽ đo được chuyển động của từng con mắt một. Sau đó chúng ta sẽ có dữ liệu về chuyển động của mắt so với dữ liệu chuyển động của chấm tròn trên màn hình.

Với những người mắc bệnh rung giật nhãn cầu, triệu chúng mà người mắc bệnh hay gặp nhất đó là mắt chuyển động mất kiểm soát. Nhờ việc đo chuyển động nhãn cầu, chúng ta có thể phát hiện những bất thường của sự di chuyển nhãn cầu, giúp hỗ trợ phát hiện sớm những bệnh nhân mắc bệnh rung giật nhãn cầu.

Phát hiện bệnh nhờ tần số nháy mắt:

Sau mỗi khoảng thời gian (60 giây trong trường hợp này), hệ thống sẽ chẩn đoán dựa trên tổng số nháy mắt. Nếu tổng số nháy mắt lớn hơn 12, nghĩa là có dấu hiệu bệnh nhãn khoa, thông báo sẽ được hiển thị. Ngược lại, nếu tổng số nháy mắt ít hơn hoặc bằng 12, thông báo cho biết mọi thứ bình thường sẽ được hiển thị.

Phát hiện dấu hiệu bệnh qua sự thay đổi hướng nhìn của mắt:

Cũng sau khoảng thời gian (60 giây trong trường hợp này), hệ thống sẽ chẩn đoán dựa trên sự thay đổi hướng nhìn của mắt. Nếu sự thay đổi hướng nhìn lớn hơn một ngưỡng, nghĩa là có dấu hiệu bệnh nhãn khoa, thông báo sẽ được hiển thị. Ngược lại, nếu sự thay đổi hướng nhìn ít hơn hoặc bằng 8, thông báo cho biết mọi thứ bình thường sẽ được hiển thị.

2. Chẩn đoán trẻ tự kỷ

2.1. Phương pháp

Theo các nghiên cứu về theo dõi các chuyển động về mắt của các bệnh nhân ASD (Autisum Spectrum Disoder), các nhà nghiên cứu đã cho biết một phương pháp

mới kiểm tra mắt có thể tiết lộ sự suy yếu ở tiểu não giúp xác định những người mắc rối loạn bệnh tự kỷ (ASD).

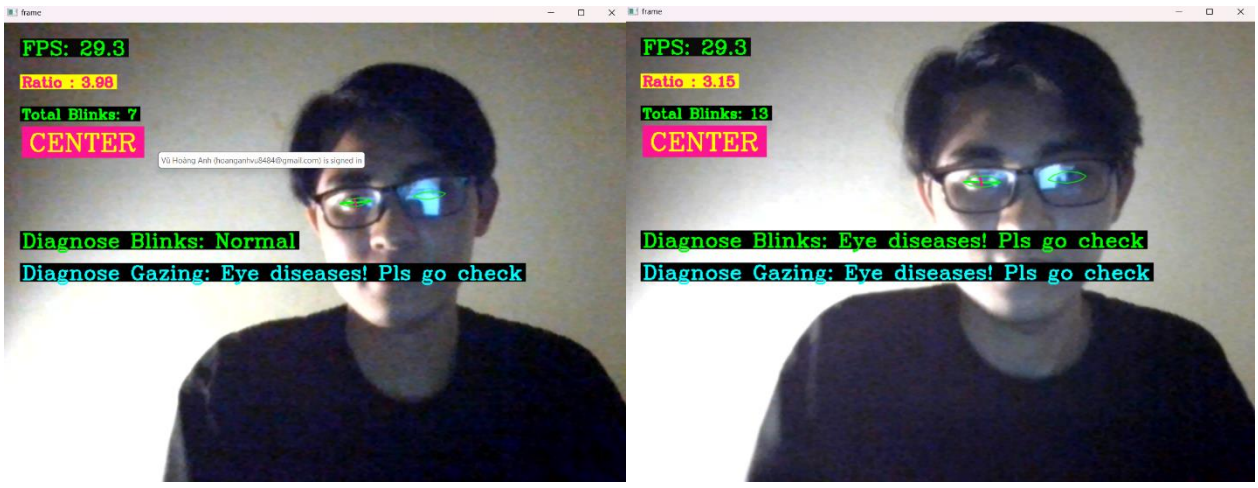
Tiểu não có chức năng phối hợp và điều chỉnh hoạt động của cơ kiểm soát chuyển động mắt khi chuyển chú ý từ vật này sang vật kia, được gọi là sự di chuyển mắt đột ngột. Ở những người khỏe mạnh, khả năng di chuyển mắt đột ngột là nhanh và chính xác. Còn ở những người tự kỷ, chức năng hoạt động này đã bị suy giảm, do sự thay đổi cấu trúc của tiểu cầu.(Theo các nhà nghiên cứu tại Đại học Rochester, New York, Mỹ được công bố trong tạp chí European Journal of Neuroscience).

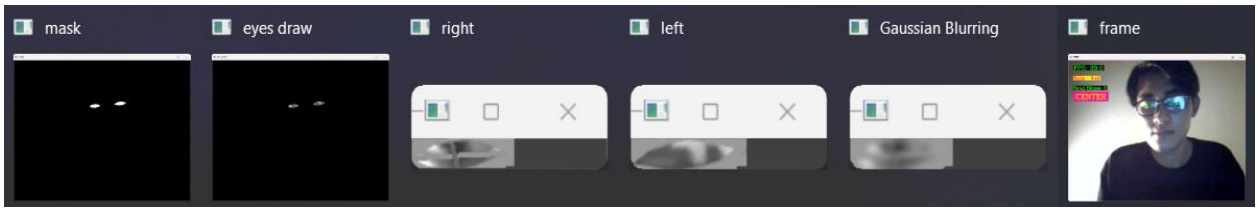
Những người tham gia thử nghiệm được yêu cầu theo dõi mục tiêu thị giác xuất hiện ở các vị trí khác nhau trên màn hình. Ở những người khỏe mạnh, não sẽ điều chỉnh chính xác sự chuyển động của mắt khi quá trình được lặp đi lặp lại. Tuy nhiên với các bệnh nhân mắc ASD họ sẽ bị bỏ lỡ mục tiêu được đề xuất trên màn hình, do các cơ cảm giác trong tiểu não chịu trách nhiệm kiểm soát chuyển động của mắt đã bị suy giảm.

Từ nghiên cứu trên, nhóm em sẽ sử dụng phương pháp eye tracking đã được trình bày ở trên để phát hiện trẻ mắc bệnh tự kỷ bằng cách sau: Đầu tiên người được chẩn đoán sẽ xem một video chấm tròn được di chuyển ngẫu nhiên trên màn hình. Sau đó, bằng phương pháp eye tracking bọn em sẽ thu được dữ liệu chuyển động mắt của người được chẩn đoán và so sánh với dữ liệu chuyển động mắt của những người bình thường. Với những người mắc bệnh tự kỷ chuyển động mắt sẽ có xu hướng phản ứng chậm hơn mục tiêu được đề xuất. Từ đó với dữ liệu thu thập được từ eye tracking chúng ta có thể phát hiện sớm những người có dấu hiệu của bệnh tự kỷ.

F. Kết quả

Hình ảnh giao diện phần mềm eye tracking:

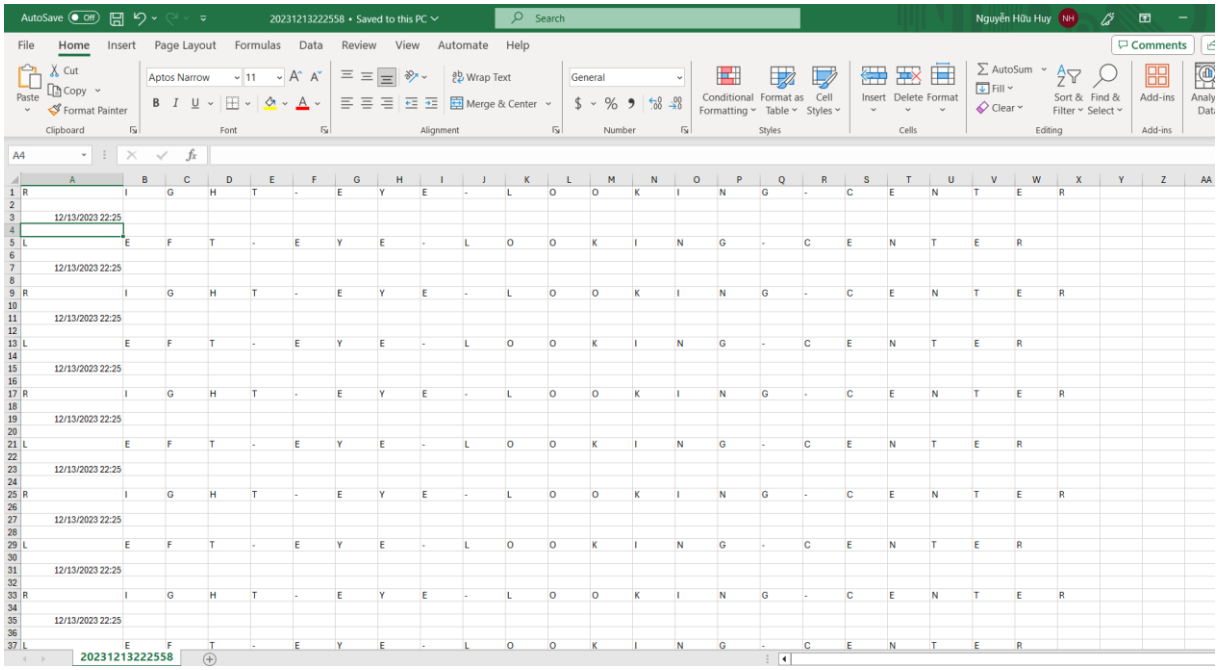




Hình 14. Kết quả mô phỏng chẩn đoán bệnh về mắt của trẻ thông qua việc theo dõi mắt

Hệ thống sẽ tự động cảnh báo nếu có sự bất thường về chuyển động của mắt

Sau khi phần mềm được chạy, sự chuyển động của nhãn cầu trong quá trình chạy sẽ được lưu kết quả ra file excel như sau:



Hình 15: Kết quả thu được

Dữ liệu thu thập được bao gồm vị trí của nhãn cầu và thời gian thực tương ứng. Từ những kết quả trên, chúng ta có thể dễ dàng hơn cho việc so sánh để sớm phát hiện những bất thường của mắt và sớm phát hiện triệu chứng của bệnh tự kỷ.

G. Đánh giá

Nhìn chung, hệ thống này đã giải quyết được một số bài toán được đưa ra:

- Sử Dụng Mediapipe và OpenCV: Việc tích hợp thư viện Mediapipe và OpenCV giúp nhanh chóng và chính xác nhận diện các điểm mốc trên khuôn mặt, đặc biệt là mắt.

- Nhận Diện Nháy Mắt: Dự án thực hiện việc nhận diện nháy mắt thông qua tỷ lệ giữa các đoạn đo của mắt, giúp đưa ra dấu hiệu về trạng thái nhắm hay mở của đôi mắt.

- Theo Dõi Hướng Nhìn: Chương trình cung cấp chức năng theo dõi hướng nhìn của mắt, thu thập dữ liệu chuyển động của nhãn cầu theo thời gian thực.

- Chẩn Đoán Bệnh Nhãn Khoa: Tính năng chẩn đoán dựa trên số lượng nháy mắt có thể cung cấp thông tin tiềm ẩn về tình trạng sức khỏe của trẻ, như bệnh nhãn khoa.

- Phát hiện dấu hiệu trẻ tự kỷ: Chương trình cũng giúp hỗ trợ thu thập dữ liệu khá chính xác góp một phần trong việc phát hiện dấu hiệu của những bệnh nhân mắc bệnh tự kỷ.

- Xử Lý Ảnh Hiệu Quả: Việc sử dụng các kỹ thuật xử lý ảnh như làm mờ và chuyển đổi thành ảnh nhị phân giúp tăng cường chất lượng và độ chính xác của quá trình nhận diện.

Tuy nhiên, hệ thống vẫn còn một số điểm hạn chế:
- Thuật toán dựa chủ yếu vào điểm ảnh nên nếu hệ thống được trang bị camera với độ phân giải kém thì hệ thống sẽ hoạt động thiếu chính xác
- Sẽ phải hiệu chỉnh đoạn mã thủ công với mỗi video đầu vào có tốc độ khung hình khác nhau
- Mới chỉ dừng lại được việc chẩn đoán các dấu hiệu bệnh đơn giản

PHẦN 5. Triển khai hệ thống lên web

Việc triển khai 2 mô hình AI trên lên web sẽ giúp người dùng dễ dàng truy cập từ xa thông qua internet, giúp phụ huynh có thể kiểm tra sơ bộ con của mình trước khi đến bệnh viện. Đồng thời việc này cũng sẽ xây dựng 1 cơ sở dữ liệu, có thể dùng cho việc sàng lọc, thống kê sau này.

A. Giới thiệu về FLASK

1. Flask là gì ?

Flask là một framework web nhẹ và dễ sử dụng được viết bằng Python, được thiết kế để linh hoạt và đơn giản, Flask là một công cụ mạnh mẽ cho việc xây dựng ứng dụng web.

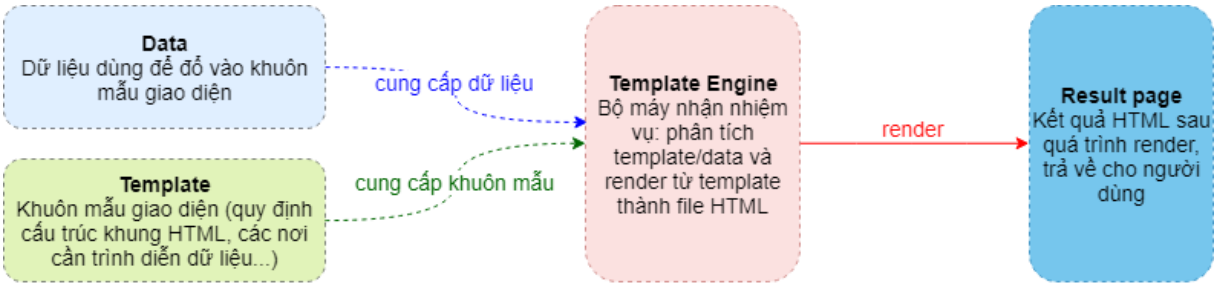
Flask giúp đơn giản hóa quá trình xây dựng và triển khai ứng dụng web, cho phép chúng ta tập trung vào logic chính của ứng dụng mà không cần quá nhiều về cấu trúc và quản lý dự án. Vì vậy nhóm em dùng Flask cho việc tích hợp mô hình AI lên ứng dụng web, giúp đơn giản hóa quá trình tương tác của người dùng với mô hình AI.

Flask dựa trên Werkzeug (một thư viện tiện ích WSGI) và Jinja2 (template engine).

WSGI là Giao diện Cổng Máy chủ Web. Đây là một đặc tả mô tả cách máy chủ web giao tiếp với các ứng dụng web và cách các ứng dụng web có thể được kết nối với nhau để xử lý một yêu cầu.

2. Template engine là gì ? Jinja2 template

Template đó chính là 1 mẫu bố cục chung cho tất cả các trang có sử dụng lại những thành phần giống nhau mà không phải viết lại toàn bộ, từ đó trên mỗi trang, chỉ cần thay đổi ở một số nơi được chỉ định trên trang từ template.

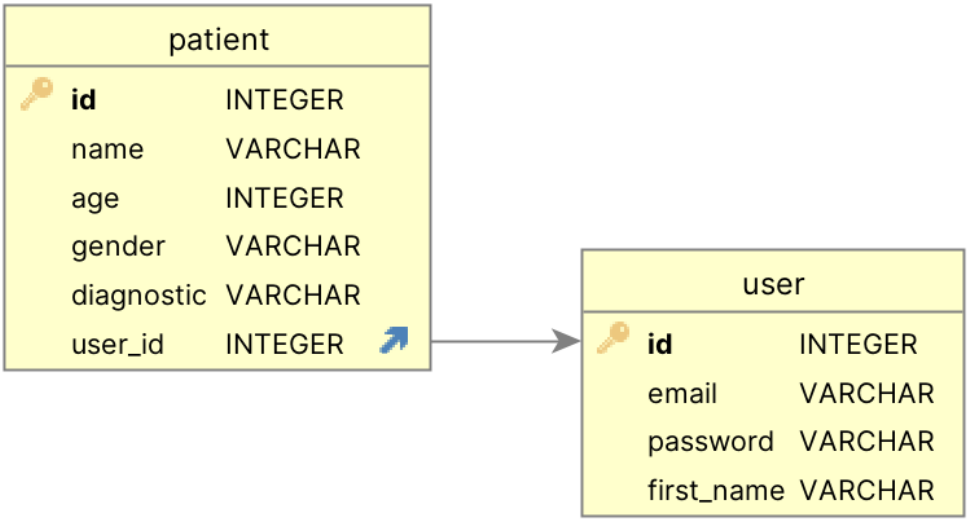


Hình 16: Nhiệm vụ của Template Engine

Jinja2 là một hệ thống template engine mạnh mẽ và linh hoạt được tích hợp sâu vào Flask, giúp đơn giản hóa quá trình tạo và hiển thị nội dung động trong ứng dụng web của bạn. Được xây dựng trên Python, Jinja2 cung cấp cú pháp dễ đọc và hiệu quả, làm cho việc kết hợp mã Python với HTML trở nên linh hoạt và thuận tiện.

B. Cơ sở dữ liệu

Cơ sở dữ liệu bao gồm các thông tin của bác sĩ (phụ huynh) và bệnh nhân (trẻ em cần theo dõi).\



Hình 17: Cơ sở dữ liệu của hệ thống

User (bác sĩ, phụ huynh) có các thông tin như id (khóa chính), email, password, tên dùng khi đăng nhập vào tài khoản.

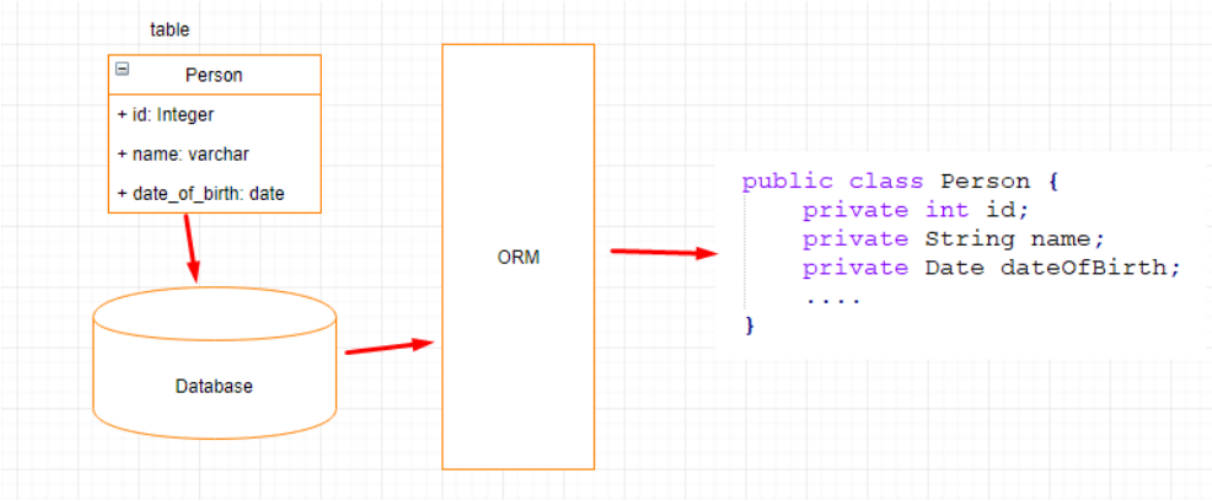
Bệnh nhân (patient) sẽ có các thông tin như: id (khóa chính), tên, tuổi, giới tính, kết quả chẩn đoán, user_id (khóa ngoài, quan hệ 1 – nhiều, tức là 1 bác sĩ có thể có nhiều bệnh nhân)

1. Công cụ SQLALCHEMY

SQLAlchemy là một ORM (Object-Relational Mapping) cho Python, cho phép tương tác với cơ sở dữ liệu mà không cần sử dụng các truy vấn SQL trực tiếp. Khi tích hợp với Flask, SQLAlchemy trở thành một công cụ mạnh mẽ để quản lý cơ sở dữ liệu và thực hiện các thao tác liên quan.

2. ORM (Object-Relational Mapping)

ORM là một kỹ thuật trong lập trình giúp biểu diễn các dòng dữ liệu (record) trong cơ sở dữ liệu bằng các đối tượng, vật thể (object) tương ứng trong ngôn ngữ lập trình. Nhờ đó, ta có thể tương tác, xử lí những data record tương tự như những object mà không cần các câu lệnh SQL



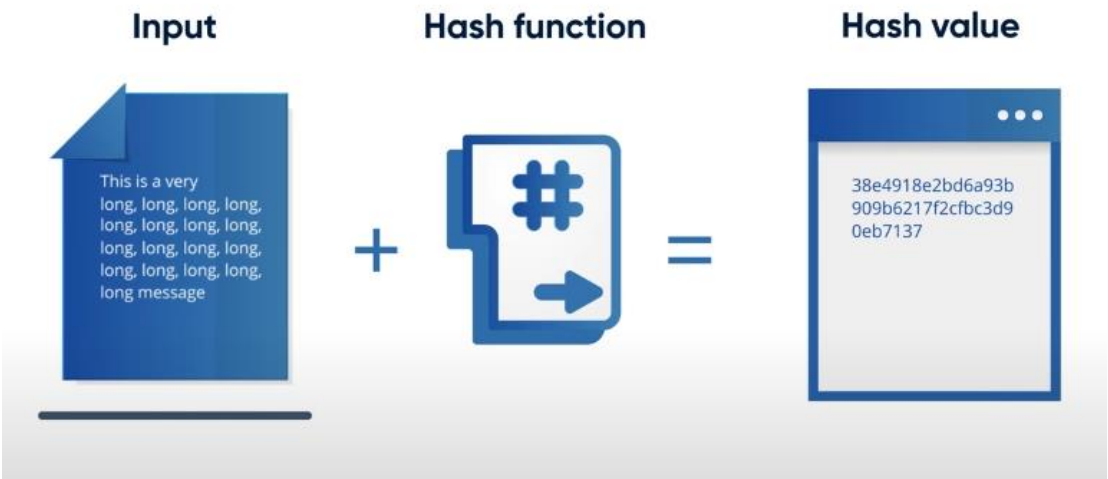
Hình 18: Ví dụ về ORM

3. Thuật toán băm (Hashing)

3.1. Giới thiệu về Hashing

Thuật toán băm được dùng để bảo vệ mật khẩu của người dùng, ví dụ như khi cơ sở dữ liệu bị lấy cắp, mật khẩu của người dùng vẫn an toàn và không bị lộ.

Hashing là một quá trình biến đổi dữ liệu đầu vào thành một giá trị cố định có độ dài cố định, thường gọi là "hash value" hoặc "hash code". Mục tiêu của quá trình hashing là tạo ra một giá trị duy nhất đại diện cho dữ liệu đầu vào. Thực tế việc băm dữ liệu là điều vô cùng phổ biến trong khoa học máy tính và được sử dụng cho rất nhiều mục đích khác nhau. Trong đó có mật mã (cryptography), nén (compression), lập chỉ mục cho dữ liệu (data indexing) hay tạo tổng kiểm tra (checksum generation).



Hình 19: Minh họa hashing

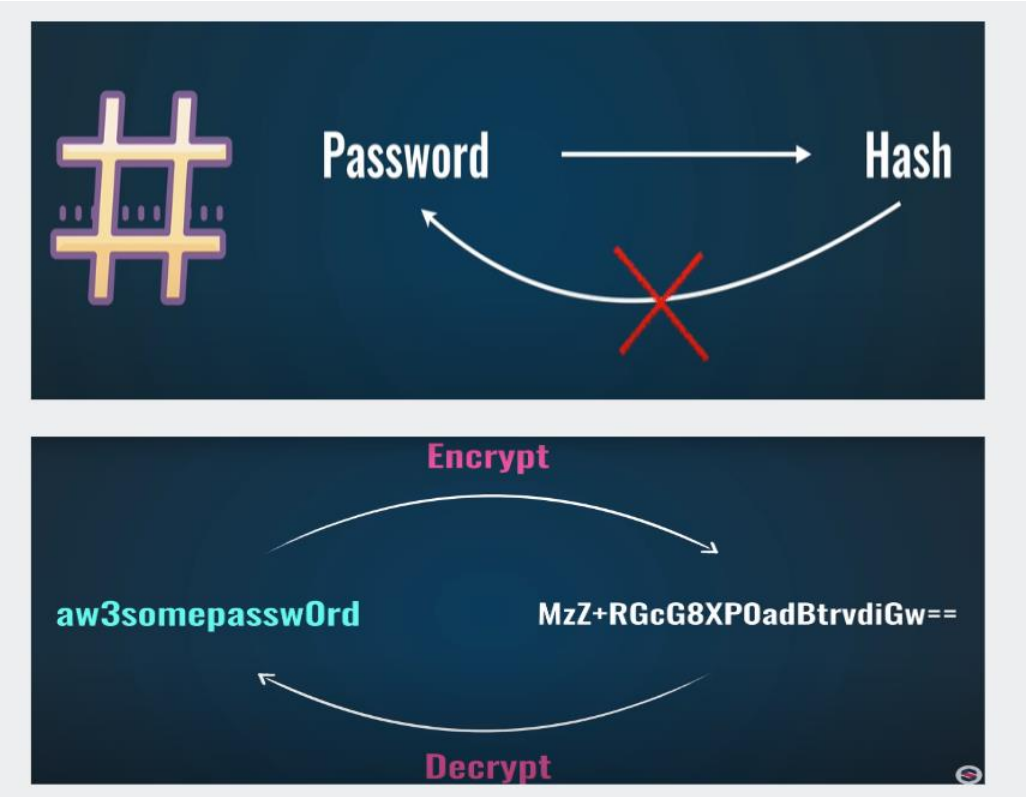
Các đặc điểm chính của Hashing:

Độ dài cố định: Hashing tạo ra một chuỗi có độ dài cố định, ngay cả khi dữ liệu đầu vào có kích thước lớn.

Chuỗi dạng "vân tay": Mỗi đầu vào khác nhau sẽ tạo ra một hash value hoàn toàn khác biệt, nhưng thay đổi nhỏ trong đầu vào sẽ tạo ra một thay đổi lớn trong hash value.

Không thể đảo ngược: Việc lấy ngược từ hash value để tìm ra dữ liệu đầu vào là không thể hoặc rất khó khăn.

Chính vì vậy dùng cho bảo mật password mạnh hơn là dùng kỹ thuật mã hóa.



Hình 20: So sánh hashing với encrypt

3.2 Thuật toán băm sha256

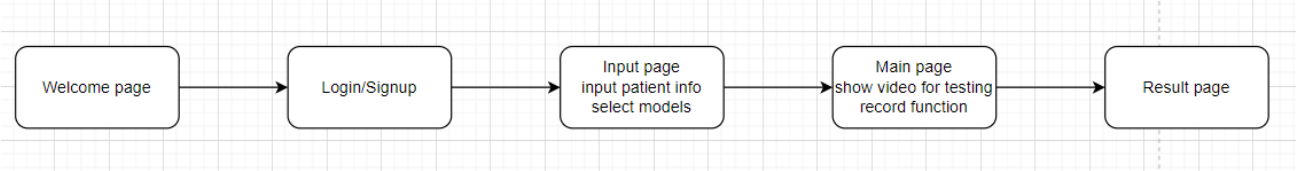
SHA-256 là viết tắt của "Thuật toán băm an toàn 256-bit", nhận được một giá trị dài 256 bit (32 byte). Như bản chất của hàm băm, cùng một giá trị luôn thu được từ cùng một dữ liệu. Mặt khác, thậm chí dữ liệu hơi khác một chút cũng mang lại các giá trị hoàn toàn khác. Nó cũng được thiết kế để gây khó khăn cho việc tìm kiếm các dữ liệu khác có cùng giá trị băm dựa trên một dữ liệu.

Ưu điểm:

SHA-256 tạo ra một giá trị hash có độ dài 256 bit, cung cấp một không gian rất lớn của các giá trị hash duy nhất.

SHA-256 được thiết kế để có khả năng chống lại tấn công collision, nơi hai dữ liệu khác nhau tạo ra cùng một giá trị hash

C. Kết quả triển khai



Hình 21: Sơ đồ luồng của trang web

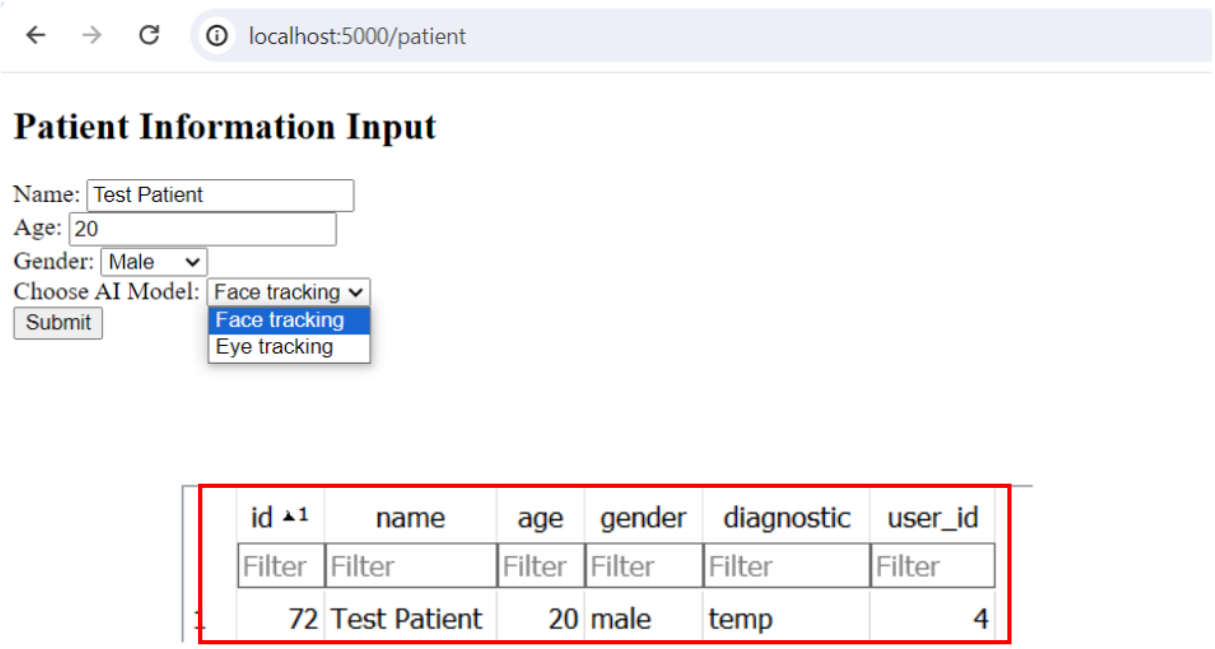
Hình 22: Trang welcome

- Trang welcome: Hiện lựa chọn đăng ký hoặc đăng nhập

Hình 23: Đăng nhập và Đăng ký

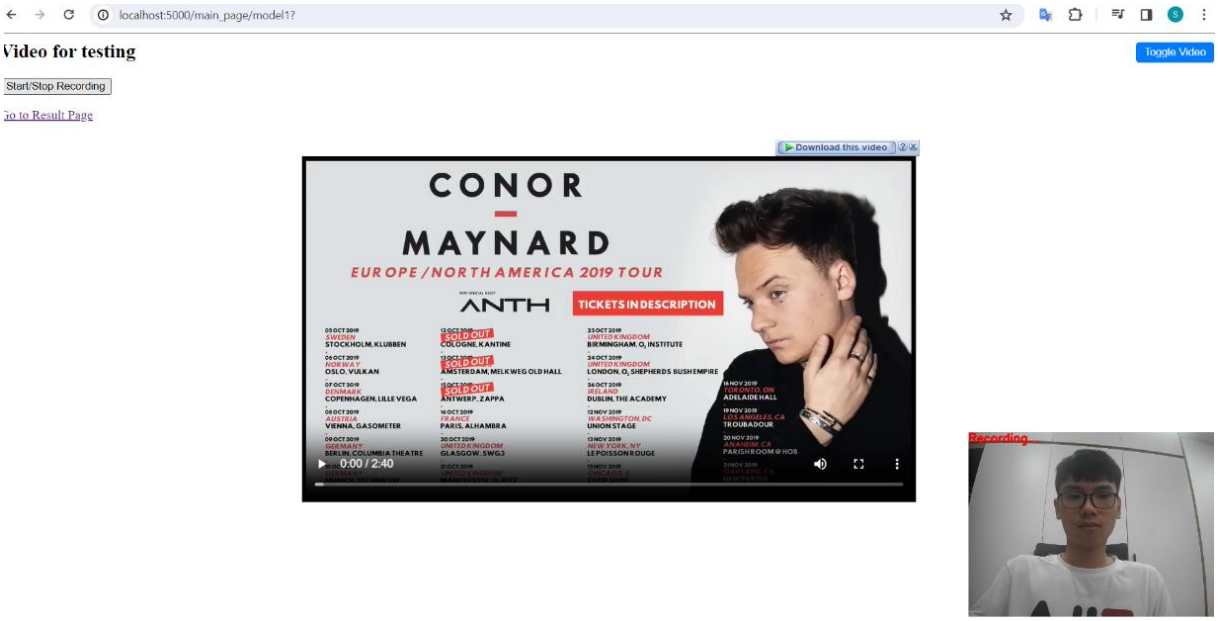
- Nếu lựa chọn đăng ký, người dùng được chuyển đến trang đăng ký và nhập các thông tin. Sau đó thông tin đó sẽ dược lưu vào cơ sở dữ liệu

- Khi đăng nhập, nếu nhập sai thông tin sẽ thông báo lỗi.



Hình 24: Nhập thông tin bệnh nhân

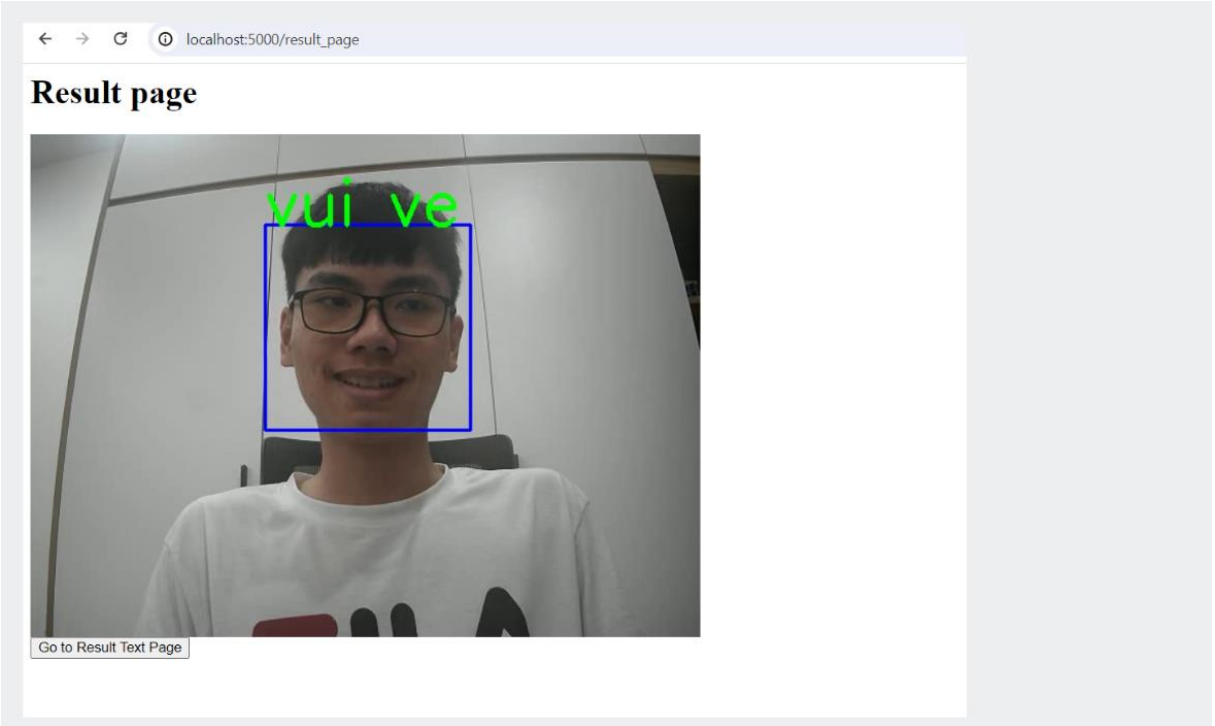
- Sau khi đăng nhập, người dùng sẽ nhập thông tin của bệnh nhân và chọn 1 trong 2 mô hình AI để kiểm tra bệnh nhân. Thông tin này cũng sẽ được lưu vào cơ sở dữ liệu (khớp với user_id = 4 của tài khoản bác sĩ). Trường "chuẩn đoán" (diagnostic) sẽ được cập nhật sau khi có kết quả bệnh.



Hình 25: Trang web chính để xem video và record

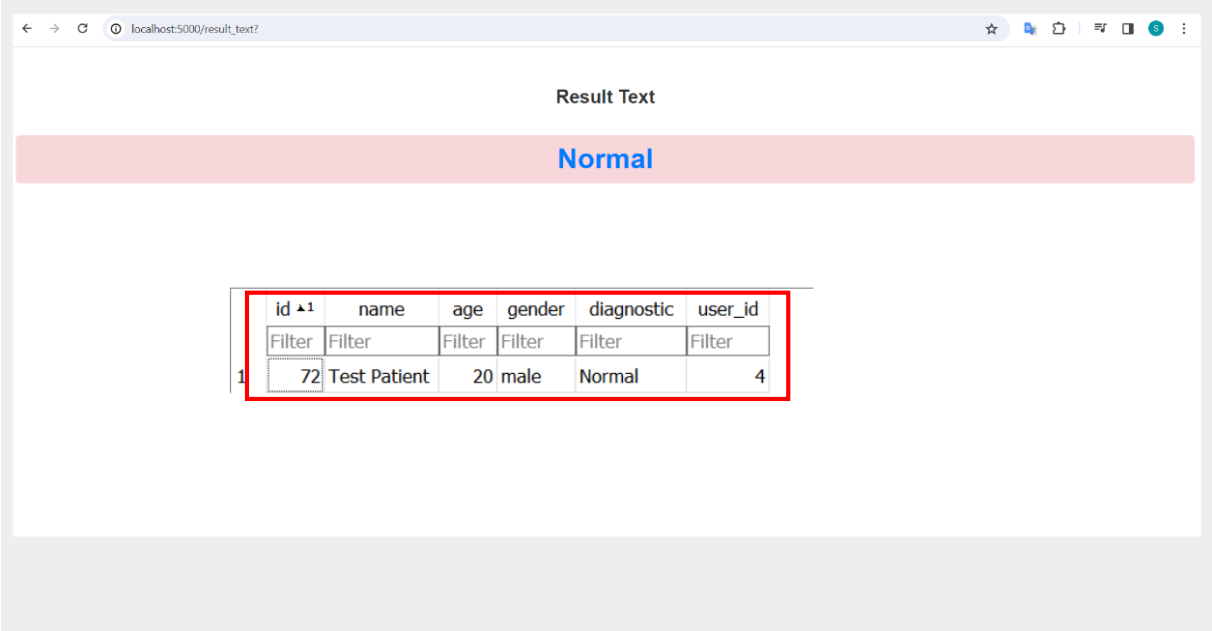
- Sau đó người dùng sẽ chuyển đến trang test, hiển thị 1 video để bệnh nhân xem, trong khi đó sẽ record lại khuôn mặt trong quá trình xem (góc phải)

(Video trong ảnh là để tạm vì chưa tìm thấy video từ người có chuyên môn về tự kỷ)



Hình 26: Trang video kết quả

- Sau khi xem video xong và bấm stop record, video record sẽ được truyền đến mô hình AI và nó sẽ thực hiện xử lý và hiển thị hình ảnh trực tiếp lên cho người dùng quan sát.



Hình 27: Trang hiện thị kết quả cuối cùng

- Sau khi xem xong, người dùng chuyển đến trang kết quả, đơn giản là 1 text để hiện thị kết quả cuối cùng (trong ảnh là "Normal"). Sau đó kết quả chuẩn đoán sẽ được lưu vào cơ sở dữ liệu trước đó.

D. Đánh giá kết quả triển khai lên web

Mô hình AI có thể xử lý realtime từ camera của client (người dùng) với điều kiện kết nối internet ổn định và camera tốt.

Giao diện web còn sơ sài vì là web tĩnh nhưng các chính năng chính vẫn hoạt động tốt.

PHẦN 6: Kết luận

Qua quá trình thực hiện và kiểm thử hệ thống, nhóm em đưa ra kết luận như sau:

Hệ thống chạy ổn định, giống mục tiêu mà nhóm đã đề ra. Tính chính xác của hệ thống khá cao trong môi trường điều kiện tốt như: ánh sáng, camera, Internet. Cả hai chương trình face tracking và eye tracking đã được kết hợp và triển khai trên nền tảng web với một giao diện dễ sử dụng vì vậy tính khả thi của dự án trong thực tế khá cao. Trong tương lai, nhóm em mong muốn hoàn thiện, bổ

sung nhiều tính năng hơn để dự án có thể sử dụng được trong thực tế, góp phần giúp nền y học phát triển, tăng hiệu suất làm việc cho các bác sĩ cũng như giúp các bậc phụ huynh có them công cụ để theo dõi tình tình sức khỏe cho con mình.

Tuy nhiên, hệ thống vẫn còn một số nhược điểm như: Giao diện người dung chưa được tối ưu, còn thô sơ. Hệ thống vẫn có sai sót trong môi trường ánh sáng kém.

Trong quá trình thực hiện dự án và làm việc nhóm, nhóm đưa ra kết luận như sau:

Các thành viên có cơ hội được tìm hiểu và lĩnh hội thêm các kiến thức mới về học máy, computer vision, lập trình web. Mỗi thành viên đều được trau dồi, bổ sung thêm kỹ năng về phần mềm và kiểm thử.

Phân công công việc theo đúng năng lực từng thành viên, tiến độ hoàn thành đúng với tiến độ trong timeline đã đề ra ở đầu kỳ. Các thành viên đoàn kết, kết hợp hiệu quả để hoàn thành công việc.

PHẦN 7: Tài liệu tham khảo

1. Bệnh viện Hồng Ngọc, *Dấu hiệu trẻ tự kỷ.*
2. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, *"MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Application"*, 2017.
3. Kaggle, *"Emotion Detection dataset",* 2013.
4. Viblo, *"Cách hoạt động của SHA-256",* 2021.
5. Tiffany Y. Tang, Guanxing Chen, Pinata Winoto, *"Emotion Recognition via Face Tracking with RealSenseTM 3D Camera for Children with Autism"*, IDC 2017.