BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA ĐIỆN – ĐIỆN TỬ

ĐỒ ÁN 1

NHẬN DIỆN THỜI GIAN THỰC NGÔN NGỮ KÝ HIỆU MỸ (ASL)

Giảng viên hướng dẫn: Huỳnh Thế Thiện

Thành viên nhóm thực hiện: Lê Phan Nguyên Đạt

Nguyễn Hoài Tâm

TPHCM - 2023

DANH MỤC HÌNH ẢNH

MỤC LỤC

CHƯƠNG 1: GIỚI THIỆU

1.1. Đặt vấn đề

Hiện nay việc bị mất các giác quan của con người đang ngày càng trở nên nghiêm trọng do bệnh tật, do di truyền, độc tố từ các thế hệ trước,…nặng thì mất thị lực hay bị mù hoàn toàn hoặc bị câm điếc không nghe được gì, đặc biết là bị khuyết tật mất hết các giác quan. Việc này đem lại cực kì nhiều khó khăn cho những con người không được may mắn ấy đặc biệt là trong giao tiếp với thế giới xung quanh. Nhưng hiện nay việc giao tiếp ấy được bù đắp bằng việc sử dụng ngôn ngữ chuyên dụng riêng dành cho người bị khuyết tật và có rất nhiều thể loại ngôn ngữ này như dấu chấm, hành động cử chỉ trong số đó kí hiệu bàn tay dễ dàng tiếp cận hơn. Tuy nhiên việc sử dụng ngôn ngữ thông qua bàn tay vẫn gây cản trở cho những người không quen khi họ phải giao tiếp với người khiếm thị và tốn cực kì nhiều thời gian để học và thành thạo.

Để giúp cho công việc giao tiếp trên trở nên dễ dàng và nhanh hơn thì với các kiến thức cùng sự tham khảo nên nhóm chọn đề tài "Nhận diện thời gian thực ngôn ngữ ký hiệu Mỹ (ASL) bằng Trí tuệ nhân tạo AI"

1.2. Mục tiêu đề tài

Nhằm tạo ra 1 phần mềm máy tính trong đó được tích hợp AI sẵn sẽ thực hiện nhiều công việc khác nhau kèm theo đó là các dữ liệu hình ảnh thu được từ tập "train" để AI có thể hiểu được các trường hợp hình ảnh của kí hiệu bàn tay. Từ đó đạt được việc hiển thị việc kí hiệu bàn tay đó có ý nghĩa là gì theo người bị khiếm thị đã đưa ra.

Đề tài được thực hiện với mục tiêu như sau:

Tìm kiếm, thu thập thật nhiều dữ liệu, hình ảnh về các kí hiệu của bàn tay dựa trên quy tắc ngôn ngữ của người khuyết tật để AI có thể thu thập được càng nhiều càng chính xác

Thông qua camera thu được ảnh của người ra hiệu, phần mềm ghi nhận và đưa hình ảnh sang AI để AI đối xứng, phân tích và đưa ra câu trả lời cho việc kí hiệu đó có ý nghĩa gì

1.3. Nội dung thực hiện

Đề tài được thực hiện với các nội dung sau:

Tìm hiểu quy tắc chữ cho người khiếm thị

Tìm và thụ thập các hình ảnh về bàn tay miêu tả câu nói của người ta

Nghiên cứu các mà AI có thể nhận và train ảnh có được từ thư mục chứa các ảnh về bàn tay

Xây dựng và viết code cho AI thực hiện việc train data.

Nghiên cứu việc sử dụng camera từ máy tình để thực hiện việc ghi nhận lại hình ảnh bàn tay.

Xây dựng phần mềm

Tiến hành kiểm thử và điều chỉnh

1.4. Kết quả đạt được

AI đã đạt được những kết quả đúng như mong muốn, cho ra được ảnh bàn thu được thông qua camera của máy tính có ý nghĩa như nào. Tuy nhiên việc train data còn tốn khá nhiều thời gian để cho ra đầu ra chính xác. Ngoài ra công đoạn train còn quá phụ thuộc vào dữ liệu ảnh khi cần nhiều ảnh hơn để độ tin cậy cao hơn ít sai số hơn nhưng việc thực thi phần mềm ngày càng chậm hơn.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Ngôn ngữ dành cho người khiếm thính

Ngôn ngữ cử chỉ là ngôn ngữ chữ cái và chữ số ước hiệu để giao tiếp với nhau, có thể biểu đạt chính xác những từ mà bạn muốn nói bằng cử chỉ, hành động. Tuy nhiên, quan trọng hơn, để phát huy tốt nhất hiệu quả của loại ngôn ngữ này, chúng ta phải chú ý đến cả những nét vui, buồn…trên khuôn mặt**.**

Quy ước: từng kí hiệu bàn tay có thể được hiểu theo bảng sau

Hình 1 Quy tắc ký hiệu

Tuy nhiên trong thực tế ngôn ngữ kí hiệu cực kì khó học và sử dụng

Đặc trưng của loại ngôn ngữ này là dùng những động tác kí hiệu của bàn tay để truyền đạt ý của mình đến người khác.

Tất cả những chữ cái, từ ngữ đều được quy ước với một dụng ý truyền đạt riêng. Tuy nhiên, rất nhiều động tác khi thực hiện lại có nhiều nét tương đồng, thậm chí rất giống nhau và có quá nhiều "từ" phức tạp, na ná nhau làm cho càng học càng lẫn lộn.

Vì thế chỉ cần làm sai đi dù chỉ một chút là thông điệp truyền tới người nghe đã bị lệch hoàn thoàn theo một hướng khác.

Đấy chính là lí do mà nhiều người ban đầu rất hào hứng nhưng chỉ sau một thời gian theo học, thấy khó khăn quá nên bỏ dở chừng.

Một lí do khác khiến cho việc học ngôn ngữ kí hiệu trở nên thật sự khó khăn là sự chưa thống nhất về quy ước giữa các vùng, thậm chí là trong một khu vực.

2.2. Artificial intelligence: Trí tuệ nhân tạo

AI hay artificial intelligence còn được gọi là trí tuệ nhân tạo là công nghệ mô phỏng các quá trình suy nghĩ và học tập của con người cho hệ thống máy tính do chính con người tạo ra. Được sử dụng để thực hiện tự động hoá các hoạt động cần nhiều sự tính toán phức tạp hoặc công việc có nhiều giai đoạn mà trên thực tế cần rất nhiều người để có thể hoàn thành công việc đó. Đây cũng tạo nên điểm lợi khi sử dụng trí tuệ nhân tạo sẽ giúp cắt giảm bớt nhân công là con người cũng như cho ra độ chuẩn xác ngang hoặc cao hơn.

Hiện nay AI được sử dụng nhiều trong các công việc tính toán để training như phân tích dữ liệu, big data hay xử lý ảnh,… Đặc biết hiện nay AI còn được sử dụng như công cụ trong cuộc sống hằng ngày như ChatGPT được sử dụng như một "Google thông minh", AI trong nhà thông minh tự động tích hợp giọng nói, AI trong robot công nghiệp, trong lĩnh vực sức khoẻ,…

Trong công nghệ xử lý ảnh, AI đã cho thấy tiềm năng mạnh mẽ với các điểm mạnh như:

+ Nhận dạng và phân loại ảnh: AI có khả năng nhận dạng và phân loại các đối tượng, đặc điểm và mẫu trong ảnh với độ chính xác cao. Các mô hình học sâu như mạng nơ-ron tích chập (CNN) có khả năng học các đặc trưng phức tạp từ dữ liệu ảnh và đưa ra dự đoán chính xác về nội dung của ảnh.

+ Phát hiện và theo dõi đối tượng: AI có thể phát hiện và theo dõi đối tượng trong ảnh, cho phép tự động nhận biết và theo dõi vị trí, hướng di chuyển và hành vi của các đối tượng. Điều này có ứng dụng rất rộng, từ giám sát an ninh đến xe tự lái và robot hợp tác.

+ Tăng cường và cải thiện ảnh: AI có thể tăng cường và cải thiện chất lượng của ảnh. Các thuật toán xử lý ảnh có thể tự động điều chỉnh độ sáng, tăng cường độ tương phản, loại bỏ nhiễu và khôi phục các chi tiết bị mờ trong ảnh.

+ Tạo ảnh và đồ họa: AI có khả năng tạo ra ảnh và đồ họa mới dựa trên dữ liệu đầu vào. Ví dụ, có thể sử dụng mô hình học sâu để tạo ra ảnh mới dựa trên các mẫu đã học từ tập dữ liệu lớn.

+ Tăng tốc và tự động hóa công việc: AI có thể giúp tăng tốc và tự động hóa các công việc xử lý ảnh. Thay vì phải xử lý và phân tích ảnh thủ công, các thuật toán AI có thể tự động thực hiện các nhiệm vụ này với hiệu suất và độ chính xác cao.

Ứng dụng thực tế AI trong xử lý ảnh

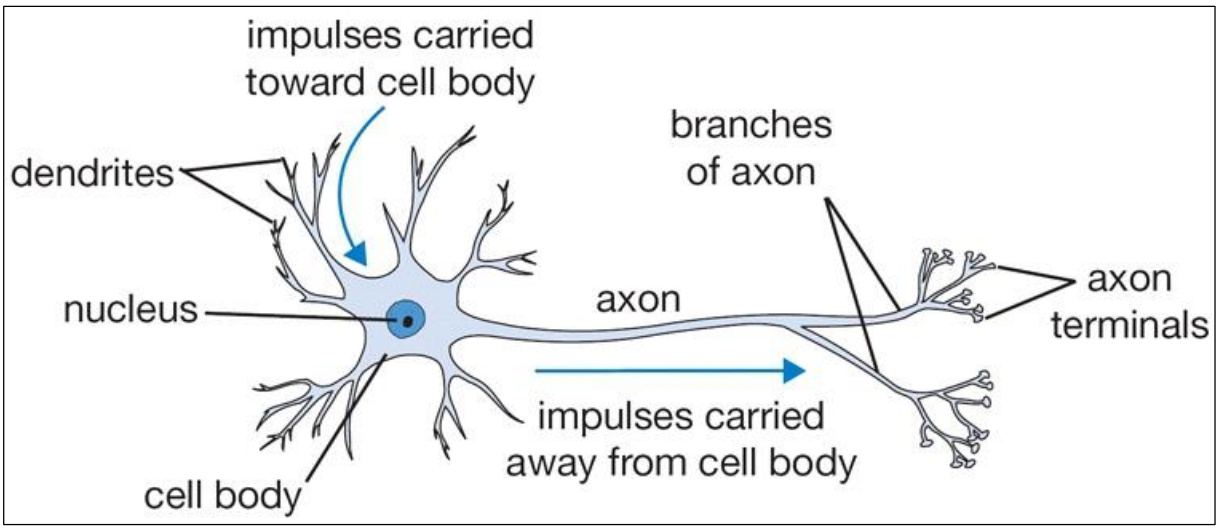
Trong y tế: các hệ thống hỗ trợ phát hiện (computer-aided detection – CADe) và hỗ trợ chẩn đoán (computer-aided diagnosis – CADx) được thiết kế để giúp bác sĩ đưa ra quyết định nhanh và chính xác hơn [1, 2]. Cụ thể, các hệ thống này cho phép phân tích và đánh giá các bất thường từ dữ liệu y khoa trong thời gian ngắn. Chúng có thể giúp cải thiện chất lượng hình ảnh y khoa, làm nổi bật các cấu trúc bất thường bên trong cơ thể và thực hiện đo đạc các chỉ số lâm sàng [3]. Các hệ thống CADe/x được xây dựng dựa trên các công nghệ lõi gồm xử lý hình ảnh, thị giác máy tính, và đặc biệt là AI.

Trong giao thông vận tải: AI được sử dụng rất nhiều trong các ứng dụng đặc biệt là hệ thống giám sát, điều hành thông minh. Hệ thống thường bao gồm các thành phần chính như màn hình tại trung tâm hiển thị bình đồ số, giúp giám sát tình trạng giao thông, kiểm soát tình trạng tuyến đường; AI camera giúp phát hiện, ghi nhận thông tin các thông tin về phương tiện (biển số, màu sắc, nhãn hiệu, chủng loại,...) và sự cố, vi phạm. Hệ thống giám sát, điều hành giao thông thông minh cung cấp bằng chứng chính xác cho lực lượng chức năng, hỗ trợ quá trình xử phạt và công tác quản lý. Từ đó, góp phần nâng cao ý thức tham gia giao thông của người dân.

Tận dụng sức mạnh của AI trong xử lý ảnh, đồ án này sẽ sử dụng trí tuệ nhân tạo để có thể nâng cao sự hiệu quả của xử lý ảnh và tạo thành sản phẩm hiểu quả và hoàn thiện.

2.3. Mạng nơ-ron

Mạng nơ-ron (Neural network) là một hệ thống tính toán lấy cảm hứng từ sự hoạt động của các nơ-ron hệ thần kinh.



Hình 2 Mạng nơ-ron thần kinh

Nơ-ron là đơn vị cơ bản cấu tạo hệ thống thần kinh và là một phần quan trọng nhất của não. Não chúng ta gồm khoảng 10 triệu nơ-ron và mỗi nơ-ron liên kết với 10.000 nơ-ron khác.

Ở mỗi nơ-ron có phần thân (soma) chứa nhân, các tín hiệu đầu vào qua sợi nhánh (dendrites) và các tín hiệu đầu ra qua sợi trục (axon) kết nối với các nơ-ron khác. Hiểu đơn giản mỗi nơ-ron nhận dữ liệu đầu vào qua sợi nhánh và truyền dữ liệu đầu ra qua sợi trục, đến các sợi nhánh của các nơ-ron khác.

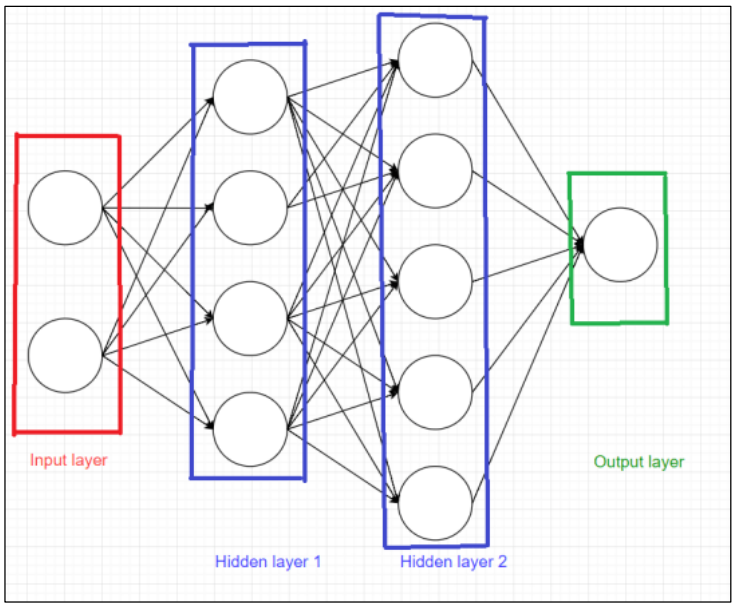
Mỗi nơ-ron nhận xung điện từ các nơ-ron khác qua sợi nhánh. Nếu các xung điện này đủ lớn để kích hoạt nơ-ron, thì tín hiệu này đi qua sợi trục đến các sợi nhánh của các nơ-ron khác.

Tuy nhiên mạng nơ-ron(NN) chỉ là lấy cảm hứng từ não bộ và cách nó hoạt động, chứ không phải bắt chước toàn bộ các chức năng của nó. Việc chính của chúng ta là dùng mô hình đấy đi giải quyết các bài toán chúng ta cần.

Mô hình tổng quát của mạng nơ-ron:

Layer đầu tiên là input layer, các layer ở giữa được gọi là hidden layer, layer cuối cùng được gọi là output layer. Các hình tròn được gọi là node.

Mỗi mô hình luôn có 1 input layer, 1 output layer, có thể có hoặc không các hidden layer. Tổng số layer trong mô hình được quy ước là số layer – 1 (Không tính input layer).



Hình 3: Mô hình tổng quát mạng nơ-ron trong học máy

Ví dụ như ở hình trên có 1 input layer, 2 hidden layer và 1 output layer. Số lượng layer của mô hình là 3 layer.

Mỗi node trong hidden layer và output layer :

Liên kết với tất cả các node ở layer trước đó với các hệ số weight *w* riêng.

Mỗi node có 1 hệ số bias *b* riêng.

Diễn ra 2 bước: tính tổng linear và áp dụng activation function.

2.4. Mạng nơ-ron tích chập

Convolutional Neural Network (CNNs – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning tiên tiến. Nó giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay.

CNN được sử dụng nhiều trong các bài toán nhận dạng các object trong ảnh. Để tìm hiểu tại sao thuật toán này được sử dụng rộng rãi cho việc nhận dạng (detection), chúng ta hãy cùng tìm hiểu về thuật toán này.

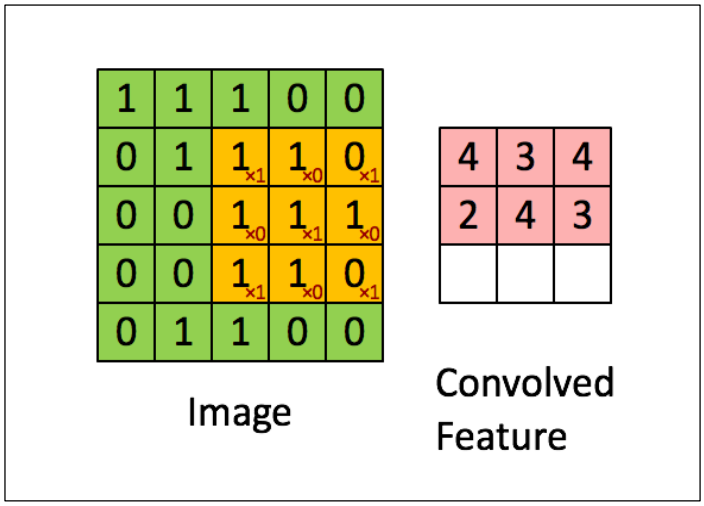
2.4.1. Convolution

Là một cửa sổ trượt (Sliding Windows) trên một ma trân như mô tả hình dưới:

Các convolutional layer có các parameter(kernel) đã được học để tự điều chỉnh lấy ra những thông tin chính xác nhất mà không cần chọn các feature.

Trong hình ảnh ví dụ trên, ma trận bên trái là một hình ảnh trắng đen được số hóa. Ma trận có kích thước 5×5 và mỗi điểm ảnh có giá trị 1 hoặc 0 là giao điểm của dòng và cột.

Sliding Window hay còn gọi là kernel, filter hoặc feature detect là một ma trận có kích thước nhỏ như trong ví dụ trên là 3×3.



Hình 4 Cửa sổ trượt Slideing Windows

Convolution hay tích chập là nhân từng phần tử bên trong ma trận 3×3 với ma trận bên trái. Kết quả được một ma trận gọi là Convolved feature được sinh ra từ việc nhân ma trận Filter với ma trận ảnh 5×5 bên trái.

2.4.2. Cấu trúc của mạng CNN và cách chọn tham số

2.4.2.1. Cấu trúc mạng CNN

Mạng CNN là một tập hợp các lớp Convolution chồng lên nhau và sử dụng các hàm nonlinear activation như ReLU và tanh để kích hoạt các trọng số trong các node. Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo.

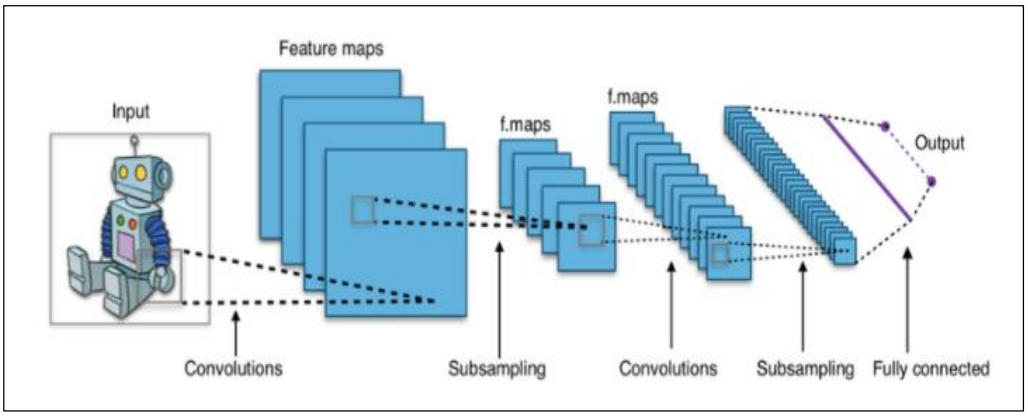
Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo. Trong mô hình mạng truyền ngược (feedforward neural network) thì mỗi neural đầu vào (input node) cho mỗi neural đầu ra trong các lớp tiếp theo.

Mô hình này gọi là mạng kết nối đầy đủ (fully connected layer) hay mạng toàn vẹn (affine layer). Còn trong mô hình CNNs thì ngược lại. Các layer liên kết được với nhau thông qua cơ chế convolution.

Layer tiếp theo là kết quả convolution từ layer trước đó, nhờ vậy mà ta có được các kết nối cục bộ. Như vậy mỗi neuron ở lớp kế tiếp sinh ra từ kết quả của filter áp đặt lên một vùng ảnh cục bộ của neuron trước đó.

Mỗi một lớp được sử dụng các filter khác nhau thông thường có hàng trăm hàng nghìn filter như vậy và kết hợp kết quả của chúng lại. Ngoài ra có một số layer khác như pooling/subsampling layer dùng để chắt lọc lại các thông tin hữu ích hơn (loại bỏ các thông tin nhiễu).

Trong quá trình huấn luyện mạng (traning) CNN tự động học các giá trị qua các lớp filter dựa vào cách thức mà bạn thực hiện. Ví dụ trong tác vụ phân lớp ảnh, CNNs sẽ cố gắng tìm ra thông số tối ưu cho các filter tương ứng theo thứ tự raw pixel > edges > shapes > facial > high-level features. Layer cuối cùng được dùng để phân lớp ảnh



Hình 5 Mô hình cấu trúc mạng CNN

Trong mô hình CNN có 2 khía cạnh cần quan tâm là tính bất biến (Location Invariance) và tính kết hợp (Compositionality). Với cùng một đối tượng, nếu đối tượng này được chiếu theo các gốc độ khác nhau (translation, rotation, scaling) thì độ chính xác của thuật toán sẽ bị ảnh hưởng đáng kể.

Pooling layer sẽ cho bạn tính bất biến đối với phép dịch chuyển (translation), phép quay (rotation) và phép co giãn (scaling). Tính kết hợp cục bộ cho ta các cấp độ biểu diễn thông tin từ mức độ thấp đến mức độ cao và trừu tượng hơn thông qua convolution từ các filter.

Đó là lý do tại sao CNNs cho ra mô hình với độ chính xác rất cao. Cũng giống như cách con người nhận biết các vật thể trong tự nhiên.

Mạng CNN sử dụng 3 ý tưởng cơ bản:

Các trường tiếp nhận cục bộ (local receptive field)

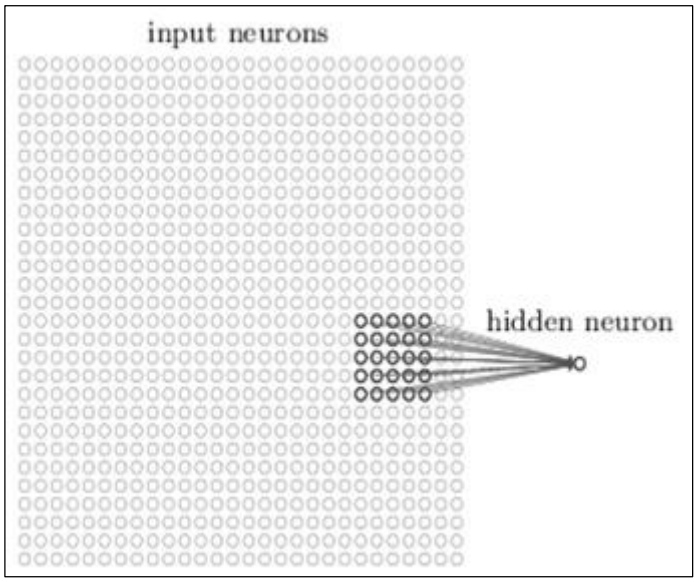
Trọng số chia sẻ (shared weights)

Tổng hợp (pooling).

a) Các trường tiếp nhận cục bộ (Local receptive field)

Đầu vào của mạng CNN là một ảnh. Ví dụ như ảnh có kích thước 28×28 thì tương ứng đầu vào là một ma trận có 28×28 và giá trị mỗi điểm ảnh là một ô trong ma trận. Trong mô hình mạng ANN truyền thống thì chúng ta sẽ kết nối các neuron đầu vào vào tầng ảnh.

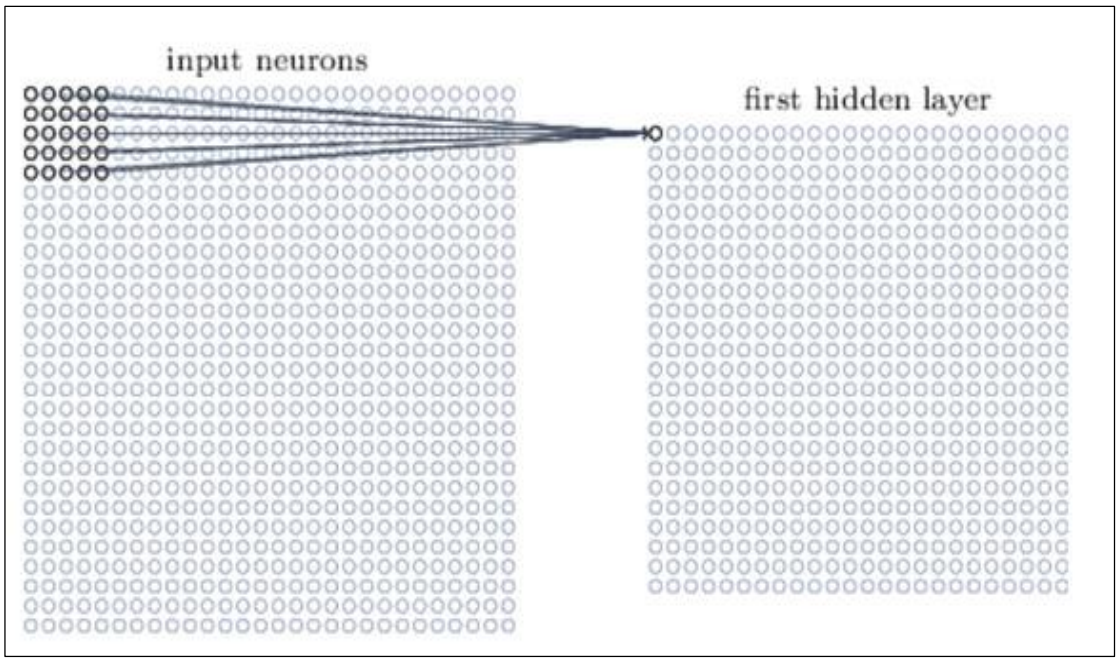
Tuy nhiên trong CNN chúng ta không làm như vậy mà chúng ta chỉ kết nối trong một vùng nhỏ của các neuron đầu vào như một filter có kích thước 5×5 tương ứng (28-5 + 1) 24 điểm ảnh đầu vào. Mỗi một kết nối sẽ học một trọng số và mỗi neuron ẩn sẽ học một bias. Mỗi một vùng 5×5 đấy gọi là một trường tiếp nhận cục bộ.



Hình 6 Ảnh đầu vào có kích thước 28x28

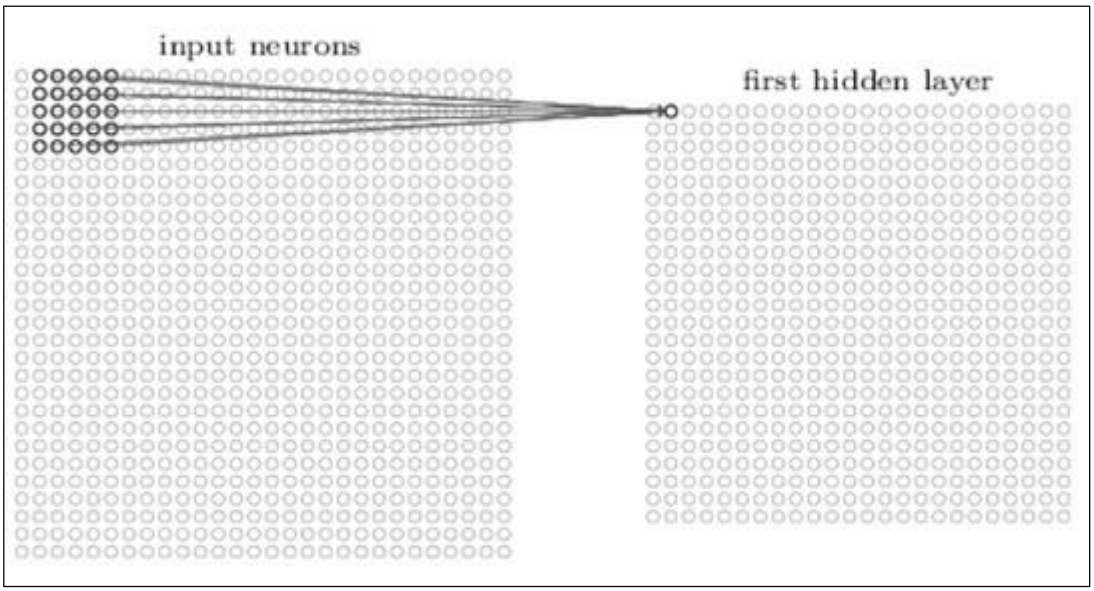
Một các tổng quan ta có thể tóm tắt các bước tạo ra 1 hidden layer bằng các cách sau:

Tạo ra Neuron ẩn đầu tiên trong lớp ẩn 1



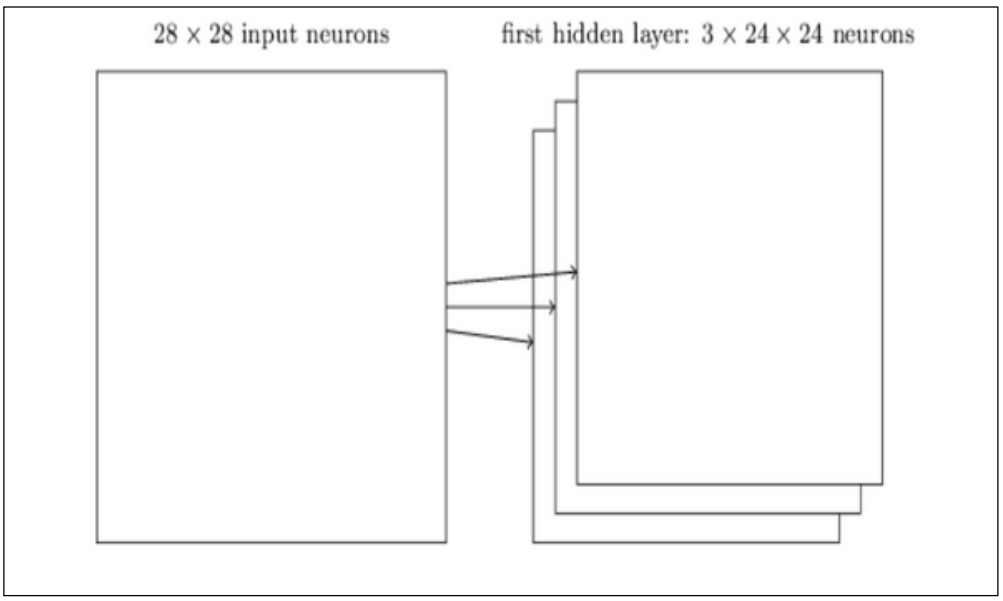
Hình 7 Nơ-ron đầu tiên trong lớp ẩn đầu tiên (First hidden layer)

Dịch filter qua bên phải một cột sẽ tạo được neuron ẩn thứ 2.



Hình 8: Nơ-ron thứ hai trong lớp ẩn đầu tiên (First hidden layer)

Với bài toán nhận dạng ảnh người ta thường gọi ma trận lớp đầu vào là feature map, trọng số xác định các đặc trương là shared weight và độ lệch xác định một feature map là shared bias. Như vậy đơn giản nhất là qua các bước trên chúng ta chỉ có 1 feature map. Tuy nhiên trong nhận dạng ảnh chúng ta cần nhiều hơn một feature map.



Hình 9: Các feature map được tạo ra từ ảnh đầu vào 28x28

Như vậy, local receptive field thích hợp cho việc phân tách dữ liệu ảnh, giúp chọn ra những vùng ảnh có giá trị nhất cho việc đánh giá phân lớp.

b) Trọng số chia sẻ

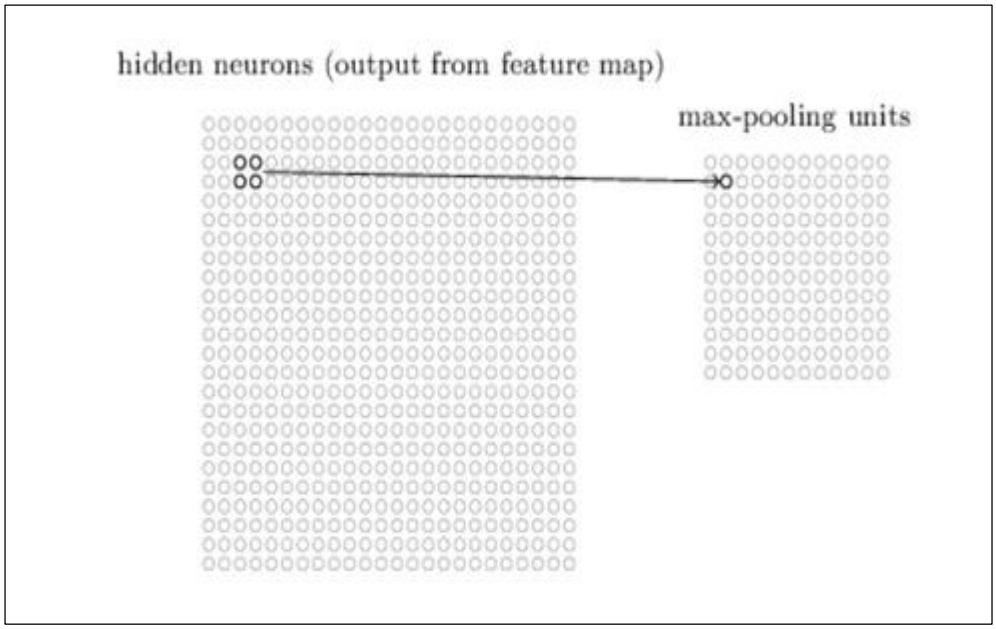
Đầu tiên, các trọng số cho mỗi filter (kernel) phải giống nhau. Tất cả các nơ-ron trong lớp ẩn đầu sẽ phát hiện chính xác feature tương tự chỉ ở các vị trí khác nhau trong hình ảnh đầu vào. Chúng ta gọi việc map từ input layer sang hidden layer là một feature map. Vậy mối quan hệ giữa số lượng Feature map với số lượng tham số là gì?

Tóm lại, một convolutional layer bao gồm các feature map khác nhau. Mỗi một feature map giúp detect một vài feature trong bức ảnh. Lợi ích lớn nhất của trọng số chia sẻ là giảm tối đa số lượng tham số trong mạng CNN.

c) Lớp tổng hợp (pooling layer)

Lớp pooling thường được sử dụng ngay sau lớp convulational để đơn giản hóa thông tin đầu ra để giảm bớt số lượng neuron.

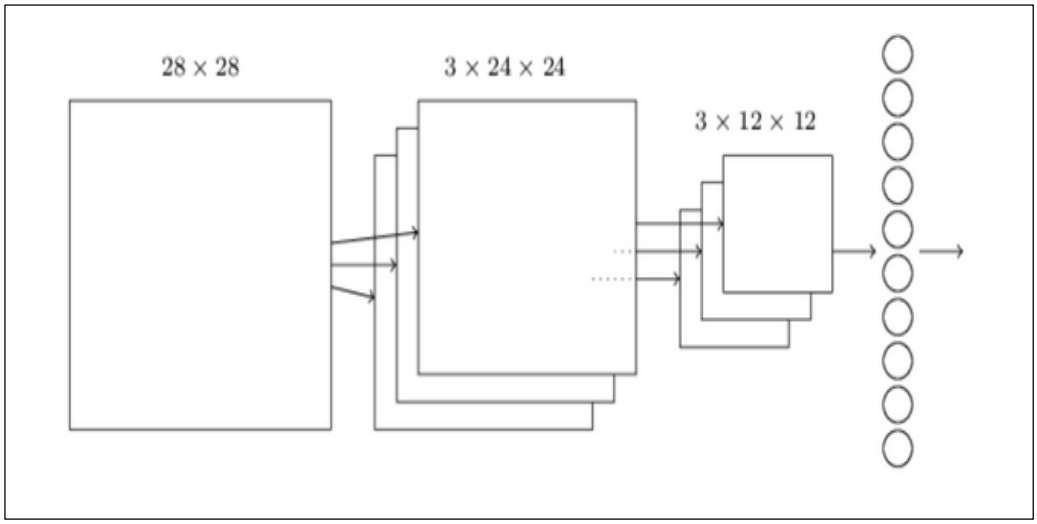
Thủ tục pooling phổ biến là max-pooling, thủ tục này chọn giá trị lớn nhất trong vùng đầu vào 2×2.



Hình 10: Max-pooling (2x2)

Như vậy qua lớp Max Pooling thì số lượng neuron giảm đi phân nửa. Trong một mạng CNN có nhiều Feature Map nên mỗi Feature Map chúng ta sẽ cho mỗi Max Pooling khác nhau. Chúng ta có thể thấy rằng Max Pooling là cách hỏi xem trong các đặc trưng này thì đặc trưng nào là đặc trưng nhất. Ngoài Max Pooling còn có L2 Pooling.

Cuối cùng ta đặt tất cả các lớp lại với nhau thành một CNN với đầu ra gồm các neuron với số lượng tùy bài toán.



Hình 11: Lớp Fully connected

Hai lớp cuối cùng của các kết nối trong mạng là một lớp đầy đủ kết nối (fully connected layer) . Lớp này nối mọi nơron từ lớp max pooled tới mọi nơron của tầng ra.

2.4.2.2. Cách chọn tham số cho CNN

Số các convolution layer: càng nhiều các convolution layer thì performance càng được cải thiện. Sau khoảng 3 hoặc 4 layer, các tác động được giảm một cách đáng kể

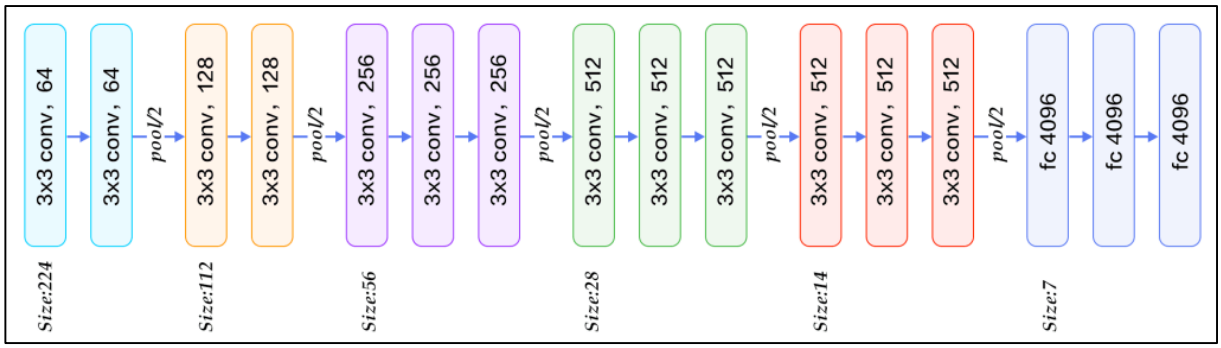
Filter size: thường filter theo size 5×5 hoặc 3×3

Pooling size: thường là 2×2 hoặc 4×4 cho ảnh đầu vào lớn

Cách cuối cùng là thực hiện nhiều lần việc train test để chọn ra được param tốt nhất.

2.4.3 Mạng VGG16

VGG16 là mạng convolutional neural network được đề xuất bởi K. Simonyan and A. Zisserman, University of Oxford. Model sau khi train bởi mạng VGG16 đạt độ chính xác 92.7% top-5 test trong dữ liệu ImageNet gồm 14 triệu hình ảnh thuộc 1000 lớp khác nhau. Giờ áp dụng kiến thức ở trên để phân tích mạng VGG16.



Hình 12: Cấu trúc mạng VGG16

Kiến trúc mạng VGG16: conv: convolutional layer, pool: pooling layer, fc: fully connected layer.

Phân tích:

- Convolutional layer: kích thước 3*3, padding=1, stride=1. Padding có nhiệm vụ thêm các giá trị 0 xung quanh ảnh đầu vào trước khi áp dụng phép tích chập. Điều này giúp duy trì kích thước của đầu ra sau phép tích chập, stride xác định số bước nhảy khi áp dụng phép tích chập trên ảnh đầu vào. Mặc định sẽ là stride=1 và padding để cho output cùng width và height với input 224x224x3.

- Pool/2 : max pooling layer với size 2*2

- 3*3 conv, 64: thì 64 là số kernel áp dụng trong layer đấy, hay depth của output của layer đấy.

- Càng các convolutional layer sau thì kích thước width, height càng giảm nhưng depth càng tăng.

- Sau khá nhiều convolutional layer và pooling layer thì dữ liệu được flatten và cho vào fully connected layer.

Một trong những lý do chính để chọn mô hình VGG16 trong dự án này là vì tính hiệu quả và khả năng phân loại cao của mô hình này:

- Kiến trúc mạng nơ-ron tích chập sâu: VGG16 là một trong những mô hình nổi tiếng của Deep Learning với kiến trúc mạng nơ-ron tích chập sâu. Kiến trúc này có khả năng học được các đặc trưng phức tạp từ dữ liệu hình ảnh và hiệu quả trong việc phân loại.

- Độ sâu và đa tầng: VGG16 có kiến trúc sâu với 16 tầng, bao gồm các lớp tích chập và pooling. Số lượng lớp nhiều giúp mô hình có khả năng học các đặc trưng từ

cấp độ thấp đến cấp độ cao, từ các đặc trưng cơ bản như đường viền và góc đến các đặc trưng phức tạp hơn như hình dạng và cấu trúc.

- Đã được huấn luyện trước trên ImageNet: Mô hình VGG16 đã được huấn luyện trước trên tập dữ liệu rất lớn gọi là ImageNet, chứa hàng triệu hình ảnh và hàng nghìn lớp phân loại khác nhau. Quá trình huấn luyện trước này giúp mô hình có khả năng trích xuất đặc trưng tốt và có sẵn các thông tin phổ quát về hình ảnh.

- Transfer learning: Bằng cách sử dụng mô hình VGG16 đã được huấn luyện trước, chúng ta có thể thụ động các kiến thức đã học được từ ImageNet và áp dụng vào bài toán cụ thể của chúng ta. Điều này giúp giảm thiểu thời gian và công sức huấn luyện từ đầu và tăng cường khả năng tổng quát hóa của mô hình.

Tuy nhiên, điều này đồng nghĩa với việc mô hình có số lượng tham số lớn và đòi hỏi nhiều tài nguyên tính toán. Để giải quyết vấn đề này, trong dự án này, nhóm chỉ lấy các tầng tích chập của VGG16 và thêm các lớp fully connected phía trên để thực hiện phân loại hành động tay.

Việc này được thực hiện để tận dụng khả năng trích xuất đặc trưng mạnh mẽ của các tầng tích chập của VGG16 và kết hợp với các lớp fully connected tùy chỉnh để phù hợp với mô hình. Các tầng tích chập của VGG16 được giữ lại vì chúng có khả năng tìm hiểu các đặc trưng ảnh cục bộ thông qua việc áp dụng các bộ lọc và kết hợp các kích thước khác nhau trên ảnh đầu vào. Những đặc trưng này sau đó được trích xuất và truyền vào các lớp fully connected để thực hiện phân loại hành động tay.

2.5. Transfer learning(Học chuyển giao)

Chắc hẳn, nhiều người cũng đã biết về các model nổi tiếng, được train trên các dataset lớn (MNIST, CIFAR-100, ImageNet, …) và source code cũng như Weights của model được public cho cộng đồng (chủ yếu là trên GitHub). Chúng ta gọi những Model đi kèm Weights như vậy là một Pretrained Model.

Model mới sử dụng một phần hay toàn bộ pretrained model như một phần của nó để học một tasks mới được gọi là Transfered Model.

Những Pretrained Model như vậy thường được train trên một hoặc một vài bộ datasets nhất định, tương thích và cho accuracy cao với một task hoặc nhiều tasks (multi-task deep learning) nào đó mà nó được train. Chúng ta gọi các tasks mà pretrained model đó được train để thực hiện là source tasks.

Nhiệm vụ của chúng ta là tạo ra một model mới để thực hiện một hoặc nhiều tasks nào đó. Những tasks cần được thực hiện của model này có thể trùng hoặc không trùng với tasks mà pretrained model được train (thường thì sẽ không trùng), chúng ta gọi tasks này là target tasks.

Transfer Learning cũng chính là cách để các model truyền đạt cho nhau khả năng mà mỗi model có thể làm được. Một model có thể học trên source tasks nào đó và rồi pretrained model này được sử dụng cho model khác để model mới đó học trên target tasks nhanh hơn.

Cụ thể, Transfer Learning trong Deep Learning là một kỹ thuật mà trong đó:

Một pretrained model đã được train trên source tasks cụ thể nào đó, khi đó một phần hay toàn bộ pretrained model có thể được tái sử dụng phụ thuộc vào nhiệm vụ của mỗi layer trong model đó.

Một model mới sử dụng một phần hay toàn bộ pretrained model để học một target tasks và tùy vào nhiệm vụ của mỗi layer mà model mới có thể thêm các layer khác dựa trên pretrained model sẵn có.

Việc sử dụng pretrained model là một bước tiến lớn để những người đi sau tiếp bước những thành quả của các bậc tiền bối, tận dụng những pretrained model sẵn có để tạo ra những model mới phục vụ cho các target tasks cụ thể hơn, mang tính ứng dụng thực tiễn hơn.

Có 2 loại transfer learning: Feature extrator và Fine tuning

Feature extractor: Sau khi lấy ra các đặc điểm của ảnh bằng việc sử dụng ConvNet của pretrained model, thì ta sẽ dùng linear classifier (linear SVM, softmax classifier,..) để phân loại ảnh.

Fine tuning: Sau khi lấy ra các đặc điểm của ảnh bằng việc sử dụng ConvNet của pretrained model, thì ta sẽ coi đây là input của một CNN mới bằng cách thêm các ConvNet và Fully Connected layer để tạo thành một model mới.

Có 2 yếu tố quan trọng nhất để dùng transfer learning đó là kích thước của dữ liệu bạn có và sự tương đồng của dữ liệu giữa mô hình bạn cần train và pretrained model.

Dữ liệu bạn có nhỏ và tương tự với dữ liệu ở pretrained model. Vì dữ liệu nhỏ nên nếu dùng fine-tuning thì model sẽ bị overfitting. Hơn nữa là dữ liệu tương tự nhau nên

là ConvNet của pretrained model cũng lấy ra các đặc điểm ở dữ liệu của chúng ta. Do đó nên dùng feature extractor.

Dữ liệu bạn có lớn và tương tự với dữ liệu ở pretrained model. Giờ có nhiều dữ liệu ta không sợ overfitting do đó nên dùng fine-tuning.

Dữ liệu bạn có nhỏ nhưng khác với dữ liệu ở pretrained model. Vì dữ liệu nhỏ nên ta lên dùng feature extractor để tránh overfitting. Tuy nhiên do dữ liệu ta có và dữ liệu ở pretrained model khác nhau, nên không nên dùng feature extractor với toàn bộ ConvNet của pretrained model mà chỉ dùng các layer đầu. Lý do là vì các layer ở phía trước sẽ học các đặc điểm chung chung hơn (cạnh, góc,…), còn các layer phía sau trong ConvNet sẽ học các đặc điểm cụ thể hơn trong dataset (ví dụ mắt, mũi,..).

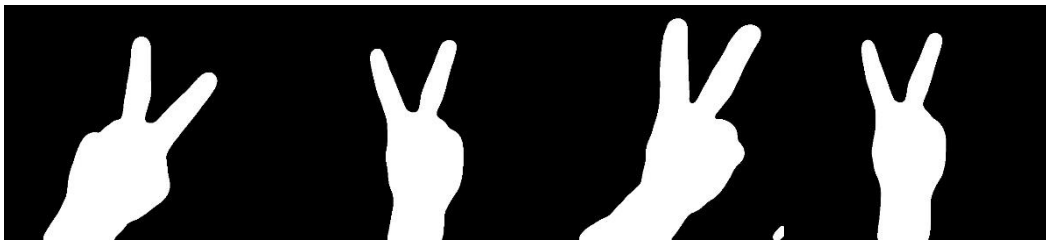
Dữ liệu bạn có lớn và khác với dữ liệu ở pretrained model. Ta có thể train model từ đầu, tuy nhiên sẽ tốt hơn nếu ta khởi tạo các giá trị weight của model với giá trị của pretrained model và sau đó train bình thường.

CHƯƠNG 3: TRIỂN KHAI VÀ XÂY DỰNG

3.1. Dữ liệu ảnh đầu vào

- Xử lý dữ liệu

Trong đồ án này sử dụng tệp data với hơn 2000 ảnh để train và 300 ảnh để test bao gồm hình ảnh của các bàn tay mô tả kí hiệu có thể xảy ra trong quá trình thực hiệc việc giao tiếp của người bị khuyết tật.



Khi bước vào quá trình training thì cần thu được dữ liệu ảnh và label (nhãn dãn) của ảnh từ thư mục data.

```
# Ham duuyet thu muc anh dung de train
def walk_file_tree(image_path):
    X_data = []
    y_data = []
    for directory, subdirectories, files in os.walk(image_path):
        for file in files:
            if not file.startswith('.'):
                path = os.path.join(directory, file)
                gesture_name = gestures[file[0:2]]
                print(gesture_name)
                print(gestures_map[gesture_name])
                y_data.append(gestures_map[gesture_name])
                X_data.append(process_image(path))

            else:
                continue

    X_data, y_data = process_data(X_data, y_data)
    return X_data, y_data
```

Tại hàm “walk_file_tree” sẽ thu thập được dữ liệu cần từ tệp data trong đó

+ X_data: sẽ là mảng chứa dữ liệu ảnh được lấy ra từ vùng chứa file data sau đó sẽ được resize ở hàm process_imgae

+ Y_data: sẽ chứa label (nhãn dán) của từng ảnh trong file data

Đối với việc lấy label, hàm tạo một biến “gesture_name” chứ 2 kí tự đầu tiên của tên file ảnh

Sau đó dựa trên 2 kí tự này, dữ liệu sẽ được ánh xạ tương ứng với ảnh đó thông qua từ từ điển "gestures_map" đã được khai báo ở trên. Từ đó sẽ tạo ra được label và thêm vào mảng Y_data

Thông qua giai đoạn này dữ liệu tên ảnh và label tương ứng sẽ được thêm vào hai mảng đã được quy định.

- Xử lý ảnh

Trong bài toán nhận dạng bàn tay, xử lý ảnh nằm trong giai đoạn tiền xử lý. Đây là một trong những vấn đề cần được nghiên cứu nhằm cải thiện chất lượng nhận dạng. Ở đồ án này sử dụng thư viện PIL để có thể truy xuất đến các thông tin của ảnh đồng thời sử dụng thư viện numpy để lưu trữ ảnh được xử lý.

Mục tiêu của giai đoạn tiền xử lý hướng tới đó là:

- Giảm đi các yếu tố làm bể ảnh, giảm thiểu sực phức tạp và ảnh hưởng của ảnh cũng như ảnh hưởng của màu sắc.

- Đưa ảnh về kích thước chuẩn phù hợp với model sử dụng.

- Giảm đi các yếu tố ảnh hưởng tới ảnh như các điểm nhiễu, ảnh hướng ánh sáng tối hay vệt mờ không liên quan trong quá trình thu nhập đầu vào.

+ Resize ảnh

- Ảnh cần được đổi về kích thước chuẩn 224x224 nhằm đem lại các hiệu quả và yêu cầu bao gồm:

- Mô hình VGG16 yeu cầu ảnh 224x225 để có thể thực hiện việc training ảnh được chính xác

- Kích thước ảnh nhỏ và là tiêu chuẩn cho nhiều mạng noron sử dụng để tiến hành train. Ngoài ra kích thước 224x224 cũng giúp việc sử dụng các trọng số hay thay đổi trở nên dễ dàng hơn

- Khắc phục các lỗi có thể xảy ra như sự mất mát dữ liệu của ảnh và giảm sự phức tạp của ảnh để có thể trainning dễ dàng hơn

Sau khi thông qua bước

- Sử dụng thư viện PIL để truy cập vào ảnh được sử dụng để training rồi đưa vào numpy để lưu ảnh thông qua lệnh:

```
# Ham xu ly anh resize ve 224x224 va chuyen ve numpy array
def process_image(path):
    img = Image.open(path)
```

```
    img = img.resize((imageSize, imageSize))
    img = np.array(img)
    return img
```

+ Nhị phân ảnh

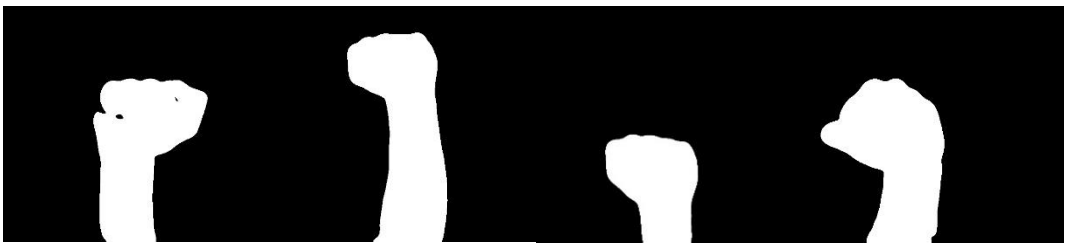
Ảnh nhị phân là ảnh đen trắng chỉ có 2 giá trị 0 và 255 (miền số nguyên) hoặc 0 và 1 (miền số thực)

Việc đưa ảnh về nhị phân sẽ đem lại nhiều lợi ích bao gồm:

Phân đoạn đối tượng (segmentation) nhằm phân tách đối tượng và vùng quan tâm khỏi nền ảnh

Giảm nhiễu hình ảnh thông qua việc đưa về màu trắng hoặc đen tuỳ thuộc vào vùng ảnh

Giảm chiều dữ liệu đồng thời giảm kích thước dữ liệu và độ phức tạp của thuật toán



Trong đồ án này thì dữ liệu ảnh sẽ được đưa về miền số thực từ 0 đến 1 bằng cách chia cho 255 và sau đó sử dụng phương pháp “one-hot encoding” để chuyển label thành dạng dị phân

```
# Xu ly du lieu dau vao
def process_data(X_data, y_data):
    X_data = np.array(X_data, dtype = 'float32')
    if rgb:
        pass
    else:
        X_data = np.stack((X_data,)*3, axis=-1)
    X_data /= 255
    y_data = np.array(y_data)
    y_data = to_categorical(y_data)
    return X_data, y_data
```

Phương pháp one-hot encoding:

Mã hoá one-hot là 1 trong những cách phổ biến để đưa dữ liệu về dạng số. Trong cách mã hoá này thì một “dictionary” (từ điển) cần được xây dựng để chứa tất cả các giá trị khả dĩ của từng dữ liệu hạng mục. Sau đó mỗi giá trị hạng mục sẽ được mã hóa

bằng một vector nhị phân với toàn bộ các phần tử bằng 0 trừ một phần tử bằng 1 tương ứng với vị trí của giá trị hạng mục đó trong từ điển.

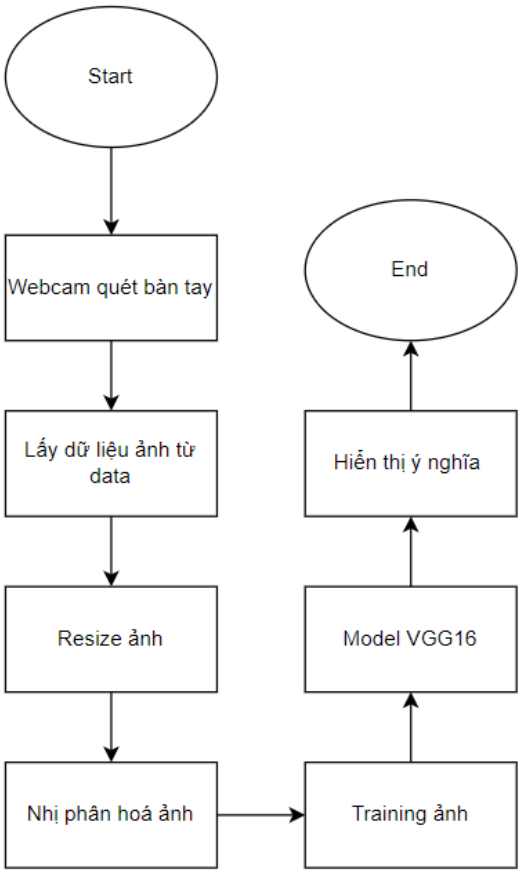
Trong đồ án này sử dụng thư viện Keras để thực hiện việc one-hot enconding cho tệp dữ liệu ảnh

```
y_data = np.array(y_data)
y_data = to_categorical(y_data)
```

Hàm "to_categorical()" thuộc thư viện Keras sẽ thực hiện việc chuyển đổi "y_data" bằng việc tạo ra một mảng 2D với khích thước bào gồm số mẫu và số lớp, trong đó chỉ có một phần từ bằng 1 tại vị trí tương ứng với lớp của mỗi mẫu, các phần từ còn lại bằng 0.

Sau quá trình này dữ liệu ảnh đã sẵn sàng để training.

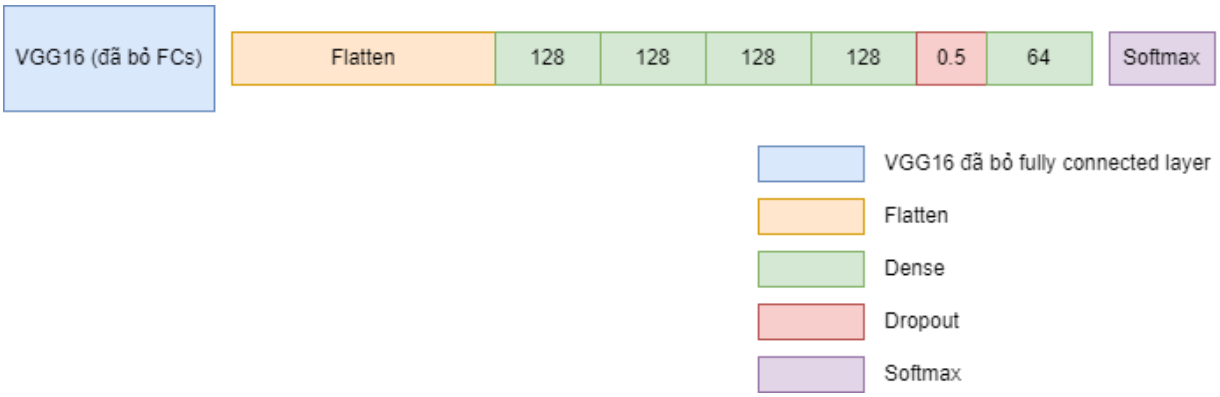
3.2. Lưu đồ giải thuật AI



Hình 13 Lưu đồ giải thuật

3.3. Xây dựng model và train model

Ở nội dung này, nhóm sẽ đề xuất một mô hình theo các phân tích ở trên và sử dụng phương pháp học chuyển giao với VGG16 để xây dựng mô hình.



Hình 14 : Cấu trúc model xây dựng bằng phương pháp Transfer learning với VGG16

Vì các layer trong ConvNet của VGG16 đã được train trước đó rồi nên ta sẽ không train lại trên các layer này nữa, ta sẽ đóng băng các layer này lại.

```
# Dong bang cac lop duoi, chi train lop ben tren minh them vao
for layer in base_model.layers:
    layer.trainable = False
```

Lớp Faltten dùng để làm phẳng thành một vetor và Dense thể hiện một fully connected layer, tức là toàn bộ các unit của layer trước được nối với toàn bộ các unit của layer hiện tại. Cuối cùng lớp Softmax, thực ra đây là lớp Dense nhưng hàm kích hoạt activation của nó là Softmax dùng để thể hiện xác suất để input đầu vào rơi vào nhãn tương ứng.

```
# Them cac lop ben tren
x = base_model.output
x = Flatten()(x)
x = Dense(128, activation='relu', name='fc1')(x)
x = Dense(128, activation='relu', name='fc2')(x)
x = Dense(128, activation='relu', name='fc2a')(x)
x = Dense(128, activation='relu', name='fc3')(x)
x = Dropout(0.5)(x)
x = Dense(64, activation='relu', name='fc4')(x)
```

Trong quá trình training chúng ta sẽ tạo một CheckPoint để theo dõi và chọn model nào có tham số accuracy cao nhất để lưu lại và sử dụng predict cho tập test về sau.

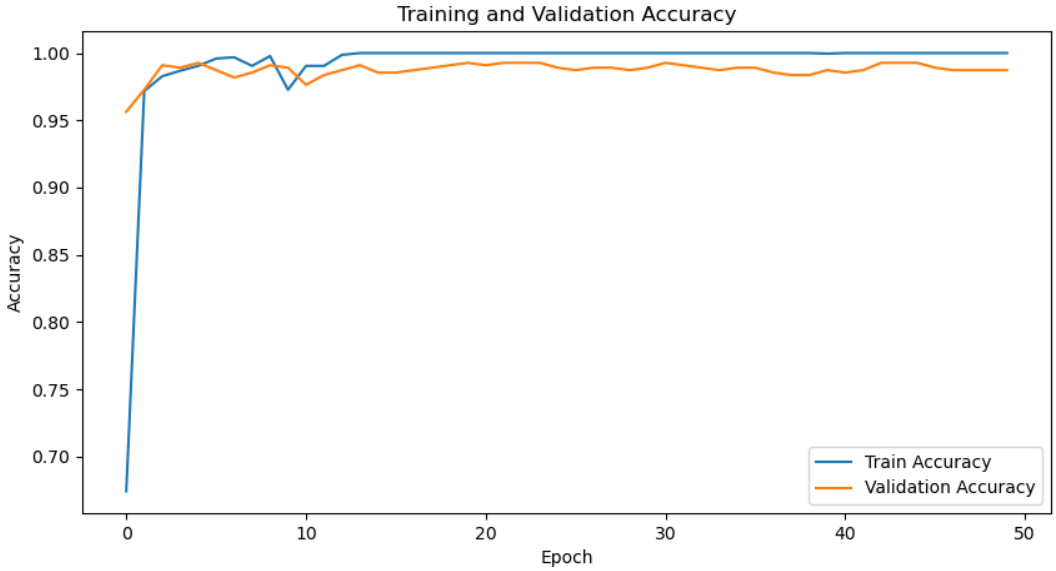
```
# Dat cac checkpoint de luu lai model tot nhat
model_checkpoint = ModelCheckpoint(filepath=models_path,
save_best_only=True)
early_stopping = EarlyStopping(monitor='val_acc',
                               min_delta=0,
                               patience=10,
                               verbose=1,
                               mode='auto',
```

```
                                    restore_best_weights=True)
```

Sau đó ta sẽ cho mô hình tiến hành train 50 ephocs và đánh giá độ chính xác của mô hình.

3.4. Đánh giá mô hình:

Hình 15 Biểu đồ Training and Validation Loss của model xây dựng bằng phương pháp Transfer learning với VGG16



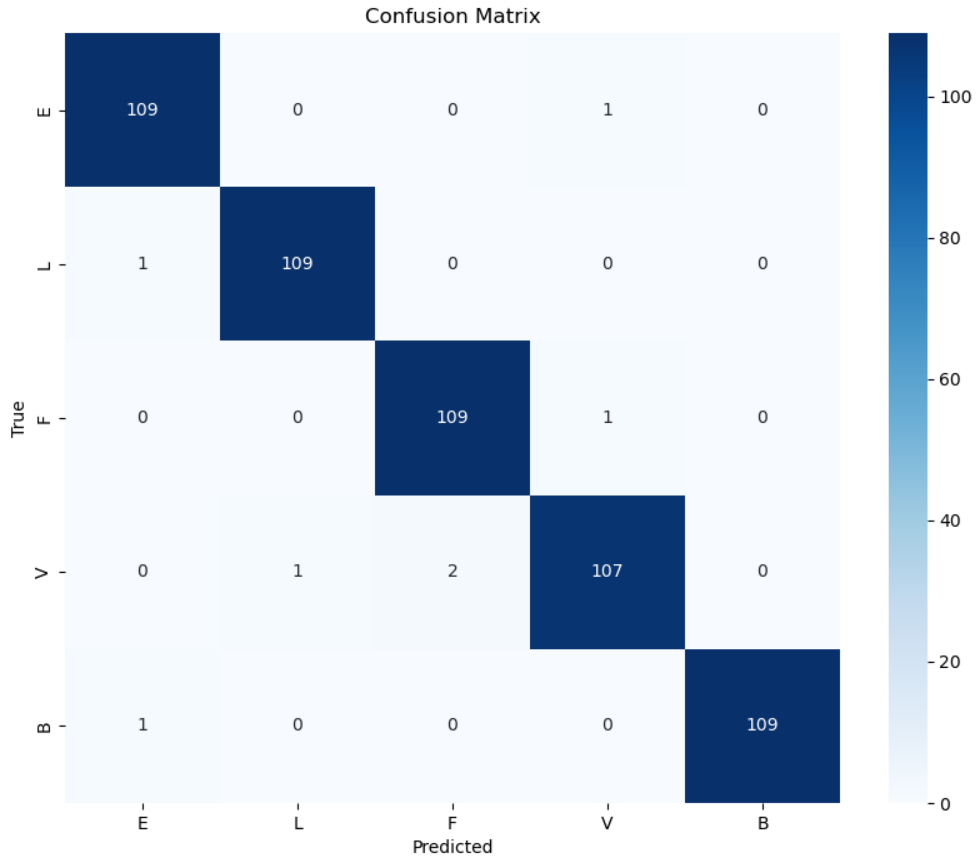
Hình 16 Biểu đồ Training và Validation Accuracy của model xây dựng bằng phương pháp Transfer learning với VGG16

Cách đơn giản và hay được sử dụng để đánh giá xem mô hình sử dụng có hiệu quả không là accuracy (độ chính xác). Cách đánh giá này đơn giản tính tỉ lệ giữa số điểm được dự đoán đúng và tổng số điểm trong tập dữ liệu kiểm thử. Theo kết quả accuracy

của mô hình là ~100%. Nhưng vẫn cần phải đánh giá thêm chỉ số Loss để có thể tránh trường hợp underfitting và overfitting.

Loss là khoảng cách giữa vector nhãn thực và vector nhãn model predict ra , model dự đoán càng lệch so với giá trị thực thì Loss (độ sai) càng to và ngược lại, nếu dự đoán càng sát với giá trị thực thì Loss (độ sai) sẽ nhỏ dần về 0.

Ở biểu đồ trên , khi số epoch tăng lên thì cả Train Loss và Validation Loss đều giảm, Train Accuracy và Val Accuracy đều tăng. Điều đó là phù hợp để tiếp tục sử dụng mô hình.



Hình 17 Confusion matrix của model xây dựng bằng phương pháp Transfer learning với VGG16

Confusion matrix thể hiện có bao nhiêu điểm dữ liệu thực sự thuộc vào một class, và được dự đoán là rơi vào một class.

Một mô hình tốt sẽ cho một confusion matrix có các phần tử trên đường chéo chính có giá trị lớn, các phần tử còn lại có giá trị nhỏ. Nói cách khác, khi biểu diễn bằng màu sắc, đường chéo có màu càng đậm so với phần còn lại sẽ càng tốt.

3.5. Đọc dữ liệu thời gian thực từ camera và sử dụng mô hình để đọc ngôn ngữ tay

Sau khi có được model thì đến giai đoạn xây dựng lên cách để phần mềm có thể lấy được ảnh của bàn tay con người, trong đồ án này sẽ viết lên đoạn code để phần mềm có thể sử dụng webcam của chính máy tính đang chạy phần mềm:

a) Tải model vào phần mềm và xây dựng hàm dự đoán

+ Thực hiện việc load model vào sau khi đã training:

```
# Load model tu file da train
model = load_model('models/mymodel.h5')
```

+ Xây dựng hàm predict để xem cử chỉ bàn tay có ý nghĩa gì :

```
# Ham de predict xem la ky tu gi
def predict_rgb_image_vgg(image):
    image = np.array(image, dtype='float32')
    image /= 255
    pred_array = model.predict(image)
    print(f'pred_array: {pred_array}')
    result = gesture_names[np.argmax(pred_array)]
    print(f'Result: {result}')
    print(max(pred_array[0]))
    score = float("%0.2f" % (max(pred_array[0]) * 100))
    print(result)
    return result, score
```

Tại đây, ảnh sẽ được đưa về miền số nguyên và tạo thành ma trận nhờ vào thư viện numpy bằng lệnh "np.array". Đồng thời lấy được tên cử chỉ (thêm vào "result") tương ứng và độ chính xác cao nhất khi dự đoán (thêm vào "score").

b) Truy cập được camera của máy tính bằng thư viện OpenCV:

```
# Camera
camera = cv2.VideoCapture(0)
camera.set(10,200)
camera.set(cv2.CAP_PROP_AUTO_EXPOSURE, 0.01)
```

Tiếp theo webcam cũng cần xử lý ảnh đầu vào trong quá trình thu thập ảnh từ người dùng:

+ Lấy vùng nhận được hình ảnh được giới hạn trong 1 ô vuông

```
# Ve khung hinh chu nhat vung detection region
    cv2.rectangle(frame, (int(cap_region_x_begin * frame.shape[1]), 0),
```

```
                    (frame.shape[1], int(cap_region_y_end *
frame.shape[0])), (255, 0, 0), 2)
```

+ Xoá đi nền để lấy ảnh hình bàn tay

```
# Tach nen
        img = remove_background(frame)

        # Lay vung detection
        img = img[0:int(cap_region_y_end * frame.shape[0]),
              int(cap_region_x_begin *frame.shape[1]):frame.shape[1]]#
clip the ROI
```

+ Chuyển ảnh thành hình đen trắng

```
# Chuyen ve den trang
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray, (blurValue, blurValue), 0)
```

c) Đưa dữ liệu ảnh bàn tay có được vào hàm dự đoán "predict_rgb_image_ vgg" xem cử chỉ bàn tay có ý nghĩa gì

```
cv2.imshow('thresh', cv2.resize(thresh, dsize=None, fx=0.5, fy=0.5))

        if
(np.count_nonzero(thresh)/(thresh.shape[0]*thresh.shape[0])>0.2):
            # Neu nhu ve duoc hinh ban tay
            if (thresh is not None):
                # Dua vao mang de predict
                target = np.stack((thresh,) * 3, axis=-1)
                target = cv2.resize(target, (224, 224))
                target = target.reshape(1, 224, 224, 3)
                prediction, score = predict_rgb_image_vgg(target)

                # Neu probality > nguong du doan thi hien thi
                print(score,prediction)
                if (score>=predThreshold):
                    cv2.putText(frame, "Sign:" + prediction, (20, 150),
cv2.FONT_HERSHEY_SIMPLEX, 3,
                                (0, 0, 255), 10, lineType=cv2.LINE_AA)
    thresh = None
```

Trong đây ảnh lấy được sẽ được coi là biến "target" và sẽ được resize lại về 224x224 để phù hợp với model đang sử dụng

Đồng thời lấy được kết quả ý nghĩa cử chỉ từ biến "prediction" và độ chính xác "scorre" sau khi đưa "target" vào hàm dự đoán và in ra màn hình.

d) Xử lý các phim bấm thêm của webcam

Đối với webcam này sẽ có các nút bấm được quy định như sau:

- Nhấn phím B để thu nhận nền.

- Nhấn Q để thoát
- R để lấy lại nền (nếu kết quả nhận ko chính xác).

Đầu tiên sử dụng OpenCV để tạo ra 1 biến đại diện cho nút bấm của người dùng ”k = cv2.waitKey(10)”

Tương ứng với từng chức năng của nút bấm:

Đối với phím B:

```
elif k == ord('b'):
        bgModel = cv2.createBackgroundSubtractorMOG2(0, bgSubThreshold)

        isBgCaptured = 1
        cv2.putText(frame, "Background captured", (20, 150),
cv2.FONT_HERSHEY_SIMPLEX, 3,
                    (0, 0, 255), 10, lineType=cv2.LINE_AA)
        time.sleep(2)
        print('Background captured')
```

Đối với phim Q: sẽ break vòng lặp “ while camera.isOpened()” và thực hiện việc tắt camera.

```
if k == ord('q'):  # Bam q de thoat
        break
```

Đối với phím R:

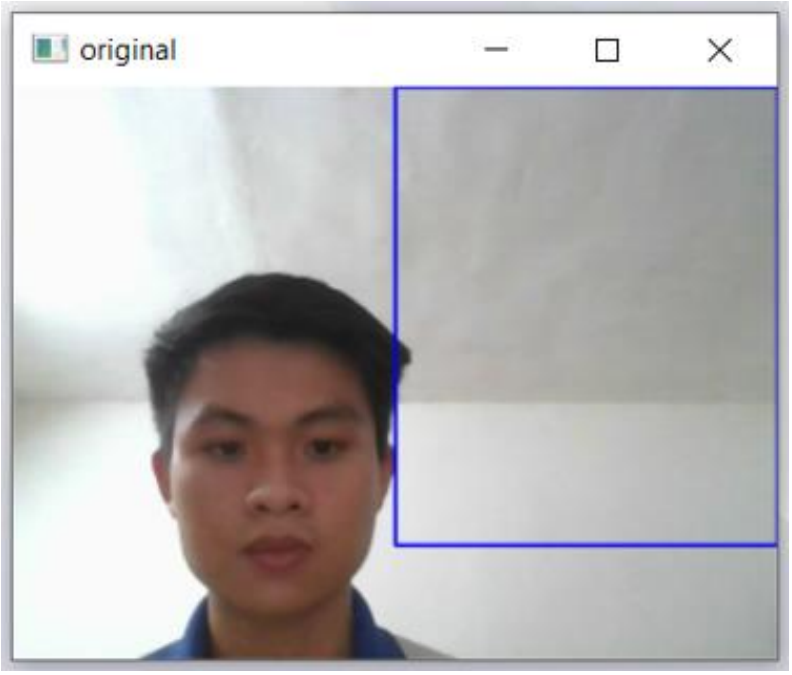
```
 elif k == ord('r'):

        bgModel = None
        isBgCaptured = 0
        cv2.putText(frame, "Background reset", (20, 150),
cv2.FONT_HERSHEY_SIMPLEX, 3,
                    (0, 0, 255),10,lineType=cv2.LINE_AA)
        print('Background reset')
        time.sleep(1)
```

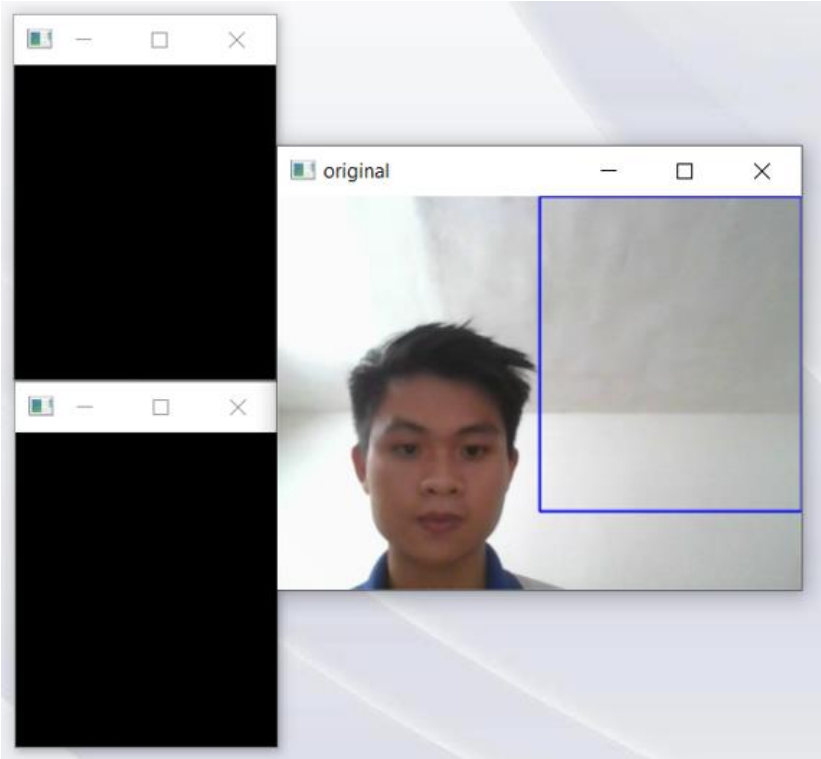
Qua đây sản phẩm đã có thể tuy cập và sử dụng webcam nhằm lấy được cử chỉ của người dùng và hiển thị ý nghĩa lên màn hình sử dụng.

CHƯƠNG 4: KẾT QUẢ

Model đã thông qua ảnh realtime từ camera máy tính nhận diện được các chữ cái trong Ngôn ngữ ký hiệu Mỹ (ASL) là E, L, F, V, B.

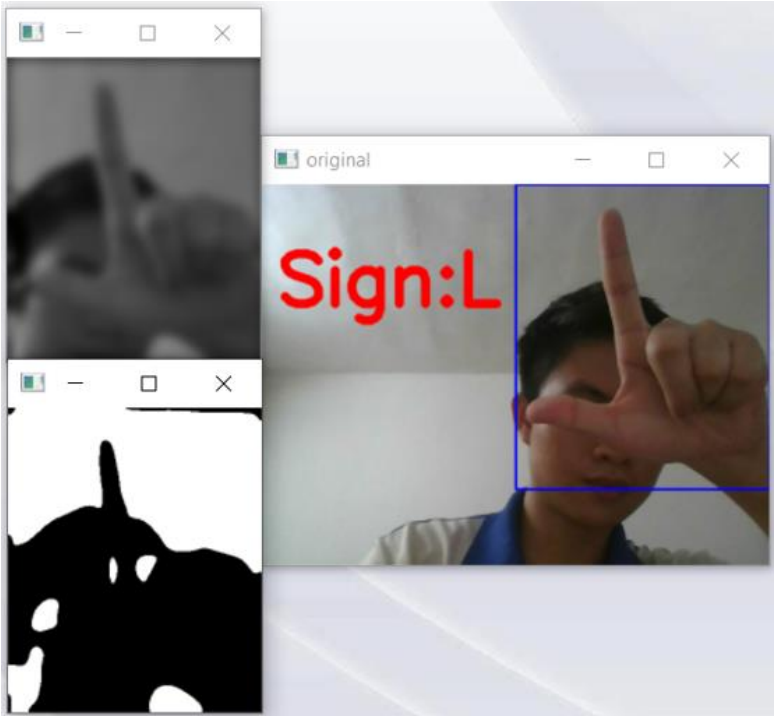


Hình 18 Giao diện khi khởi động



Hình 19 Giao diện nhận diện khi nhấn phím B

Hình 20 Nhận diện chữ V trong ASL với vùng nhận diện rõ ràng



Hình 21 Nhận diện chữ L trong ASL với vùng nhận diện không rõ ràng

CHƯƠNG 5: KẾT LUẬN – HƯỚNG PHÁT TRIỂN

Sau nghiên cứu và thực hiện đồ án, nhóm đã hoàn thành được phần có sự tích hợp trí tuệ nhân tạo AI nhằm tạo ra được những kết quả sau quá trình thực hiện:

Sử dụng được code để có ảnh lấy được từ webcam.

Sử dụng dược AI để có thể trainning được tập data thu nhập được.

Sử dụng model được tạo thành sau quá trình training để có thể xử lý ảnh từ webcam.

Phần mềm chạy ổn định, tốc độ xử lý hay train ảnh còn hơi chậm. Hiển thị các kí tự rõ rành, chính xác nhưng vẫn có sự đơ, delay.

Phần mềm dễ sử dụng và đáp ứng nhu càu thực tiễn cũng như các yêu cầu cơ bản về sử dụng phần mềm của người dùng.

Nhóm dự kiến sẽ phát triển để có thể nhận diện hết tất cả các chữ trong bảng chữ cái tiếng Anh và các câu nói bằng hai tay cả tiếng Anh lẫn tiếng Việt.

TÀI LIỆU THAM KHẢO

[1] https://vietnammoi.vn/hoc-ngon-ngu-ky-hieu-de-hay-kho-159138.htm

[2] https://vi.wikipedia.org/wiki/Ng%C3%B4n_ng%E1%BB%AF_k%C3%BD_hi%E1%BB%87u

[3] https://www.geeksforgeeks.org/python-keras-keras-utils-to_categorical/

[4] https://viblo.asia/p/thao-tac-voi-process-vyDZO6kdKwj

[5] https://www.elcom.com.vn/6-ung-dung-tri-tue-nhan-tao-ai-mang-tinh-cach-mang-trong-nganh-giao-thong-1663563867

[6] https://arxiv.org/abs/1409.1556?fbclid=IwAR3Bljc7LXQwwd64sQODYCrQV_WrvedR2fDvFIf2oMb8nJhP_lO7o-OpFVo

SOURCE CODE

Code trainning data:

```
import os
import warnings
import cv2
import keras
import matplotlib.pyplot as plt
import matplotlib.style as style
import numpy as np
import pandas as pd
from PIL import Image
from keras import models, layers, optimizers
from keras.applications import VGG16
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.layers import Dense, Dropout, Flatten
from keras.models import Model
from keras.preprocessing import image as image_utils
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True



# Dinh nghia cac bien


gestures = {'L_': 'L',
'fi': 'E',
'ok': 'F',
```

```
'pe': 'V',
'pa': 'B'
}

gestures_map = {'E': 0,
'L': 1,
'F': 2,
'V': 3,
'B': 4
}



gesture_names = {0: 'E',
1: 'L',
2: 'F',
3: 'V',
4: 'B'}



image_path = 'data'
models_path = 'models/saved_model.hdf5'
rgb = False
imageSize = 224



# Ham xu ly anh resize ve 224x224 va chuyen ve numpy array
def process_image(path):
img = Image.open(path)
img = img.resize((imageSize, imageSize))
img = np.array(img)
```

```
return img

# Xu ly du lieu dau vao
def process_data(X_data, y_data):
X_data = np.array(X_data, dtype = 'float32')
if rgb:
pass
else:
X_data = np.stack((X_data,)*3, axis=-1)
X_data /= 255
y_data = np.array(y_data)
y_data = to_categorical(y_data)
return X_data, y_data

# Ham duuyet thu muc anh dung de train
def walk_file_tree(image_path):
X_data = []
y_data = []
for directory, subdirectories, files in os.walk(image_path):
for file in files:
if not file.startswith('.'):
path = os.path.join(directory, file)
gesture_name = gestures[file[0:2]]
print(gesture_name)
print(gestures_map[gesture_name])
y_data.append(gestures_map[gesture_name])
X_data.append(process_image(path))

else:
continue
```

```
    X_data, y_data = process_data(X_data, y_data)
    return X_data, y_data




    # Load du lieu vao X va Y
    X_data, y_data = walk_file_tree(image_path)

    # Phan chia du lieu train va test theo ty le 80/20
    X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size = 0.2,
random_state=12, stratify=y_data)

    # Dat cac checkpoint de luu lai model tot nhat
    model_checkpoint = ModelCheckpoint(filepath=models_path,
save_best_only=True)
    early_stopping = EarlyStopping(monitor='val_acc',
    min_delta=0,
    patience=10,
    verbose=1,
    mode='auto',
    restore_best_weights=True)

    # Khoi tao model
    model1 = VGG16(weights='imagenet', include_top=False,
input_shape=(imageSize, imageSize, 3))
    optimizer1 = optimizers.Adam()
    base_model = model1
```

```
    # Them cac lop ben tren
    x = base_model.output
    x = Flatten()(x)
    x = Dense(128, activation='relu', name='fc1')(x)
    x = Dense(128, activation='relu', name='fc2')(x)
    x = Dense(128, activation='relu', name='fc2a')(x)
    x = Dense(128, activation='relu', name='fc3')(x)
    x = Dropout(0.5)(x)
    x = Dense(64, activation='relu', name='fc4')(x)

    predictions = Dense(5, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)

    # Dong bang cac lop duoi, chi train lop ben tren minh them vao
    for layer in base_model.layers:
    layer.trainable = False

    model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    model.fit(X_train, y_train, epochs=50, batch_size=64, validation_data=(X_test,
y_test), verbose=1,
    callbacks=[early_stopping, model_checkpoint])

    # Luu model da train ra file
    model.save('models/mymodel.h5')
```

Code webcam

```
    #! /usr/bin/env python3

    import copy
```

```
import cv2
import numpy as np
from keras.models import load_model
import time

# Cac khai bao bien
prediction = ''
score = 0
bgModel = None

gesture_names = {0: 'E',
1: 'L',
2: 'F',
3: 'V',
4: 'B'}

# Load model tu file da train
model = load_model('models/mymodel.h5')

# Ham de predict xem la ky tu gi
def predict_rgb_image_vgg(image):
image = np.array(image, dtype='float32')
image /= 255
pred_array = model.predict(image)
print(f'pred_array: {pred_array}')
result = gesture_names[np.argmax(pred_array)]
print(f'Result: {result}')
print(max(pred_array[0]))
score = float("%0.2f" % (max(pred_array[0]) * 100))
print(result)
return result, score
```

```
# Ham xoa nen khoi anh
def remove_background(frame):
fgmask = bgModel.apply(frame, learningRate=learningRate)
kernel = np.ones((3, 3), np.uint8)
fgmask = cv2.erode(fgmask, kernel, iterations=1)
res = cv2.bitwise_and(frame, frame, mask=fgmask)
return res



# Khai bao kich thuoc vung detection region
cap_region_x_begin = 0.5
cap_region_y_end = 0.8

# Cac thong so lay threshold
threshold = 60
blurValue = 41
bgSubThreshold = 50#50
learningRate = 0

# Nguong du doan ky tu
predThreshold= 95

isBgCaptured = 0 # Bien luu tru da capture background chua

# Camera
camera = cv2.VideoCapture(0)
camera.set(10,200)
camera.set(cv2.CAP_PROP_AUTO_EXPOSURE, 0.01)

while camera.isOpened():
# Doc anh tu webcam
ret, frame = camera.read()
# Lam min anh
```

```
        frame = cv2.bilateralFilter(frame, 5, 50, 100)
        # Lat ngang anh
        frame = cv2.flip(frame, 1)

        # Ve khung hinh chu nhat vung detection region
        cv2.rectangle(frame, (int(cap_region_x_begin * frame.shape[1]), 0),
        (frame.shape[1], int(cap_region_y_end * frame.shape[0])), (255, 0, 0), 2)

        # Neu ca capture dc nen
        if isBgCaptured == 1:
        # Tach nen
        img = remove_background(frame)

        # Lay vung detection
        img = img[0:int(cap_region_y_end * frame.shape[0]),
        int(cap_region_x_begin * frame.shape[1]):frame.shape[1]] # clip the ROI



        # Chuyen ve den trang
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray, (blurValue, blurValue), 0)

        cv2.imshow('original1', cv2.resize(blur, dsize=None, fx=0.5, fy=0.5))

        ret, thresh = cv2.threshold(blur, threshold, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

        cv2.imshow('thresh', cv2.resize(thresh, dsize=None, fx=0.5, fy=0.5))

        if (np.count_nonzero(thresh)/(thresh.shape[0]*thresh.shape[0])>0.2):
        # Neu nhu ve duoc hinh ban tay
        if (thresh is not None):
```

```
        # Dua vao mang de predict
        target = np.stack((thresh,) * 3, axis=-1)
        target = cv2.resize(target, (224, 224))
        target = target.reshape(1, 224, 224, 3)
        prediction, score = predict_rgb_image_vgg(target)

        # Neu probality > nguong du doan thi hien thi
        print(score,prediction)
        if (score>=predThreshold):
        cv2.putText(frame, "Sign:" + prediction, (20, 150),
cv2.FONT_HERSHEY_SIMPLEX, 3,
        (0, 0, 255), 10, lineType=cv2.LINE_AA)
        thresh = None

        # Xu ly phim bam
        k = cv2.waitKey(10)
        if k == ord('q'): # Bam q de thoat
        break
        elif k == ord('b'):
        bgModel = cv2.createBackgroundSubtractorMOG2(0, bgSubThreshold)

        isBgCaptured = 1
        cv2.putText(frame, "Background captured", (20, 150),
cv2.FONT_HERSHEY_SIMPLEX, 3,
        (0, 0, 255), 10, lineType=cv2.LINE_AA)
        time.sleep(2)
        print('Background captured')

        elif k == ord('r'):

        bgModel = None
        isBgCaptured = 0
```

```
        cv2.putText(frame, "Background reset", (20, 150), cv2.FONT_HERSHEY_SIMPLEX,
3,
        (0, 0, 255),10,lineType=cv2.LINE_AA)
        print('Background reset')
        time.sleep(1)



        cv2.imshow('original', cv2.resize(frame, dsize=None, fx=0.5, fy=0.5))



        cv2.destroyAllWindows()
        camera.release()
```