

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH

KHOA CƠ KHÍ CHẾ TẠO MÁY

BỘ MÔN CƠ ĐIỆN TỬ



HCMUTE

BÁO CÁO CUỐI KÌ

MÔN TRÍ TUỆ NHÂN TẠO

**NHẬN DIỆN CÁC ĐIỂM LANDMARK
TRÊN KHUÔN MẶT ỨNG DỤNG CNN**

GVHD: PGS. TS Nguyễn Trường Thịnh

MHP: ARIN337629_06

Họ và tên: Nguyễn Trọng Nhân

MSSV: 19146065

Năm học: Học kì II 2021 - 2022

Thành phố Hồ Chí Minh, tháng 6 năm 2022

MỤC LỤC

DANH MỤC HÌNH ẢNH.....	iii
DANH MỤC BẢNG.....	iv
DANH MỤC CÁC TỪ VIẾT TẮT.....	v
CHƯƠNG 1. TỔNG QUAN.....	1
1.1. Facial Landmark là gì?.....	2
1.2. Lý do chọn đề tài	2
1.3. Mục tiêu nghiên cứu.....	3
1.4. Phương pháp nghiên cứu.....	3
1.5. Nội dung báo cáo.....	3
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT.....	4
2.1. Thuật toán CNN - Convolutional Neural Network	4
2.1.1. Convolutional Neural Network là gì	4
2.1.2. Cấu trúc của mạng CNN	6
2.2. Thư viện OpenCv	9
2.3. Thư viện Pytorch	10
2.4. Thư viện Pillow	12
2.5. Công cụ xây dựng Web app - Streamlit	12
CHƯƠNG 3. ỨNG DỤNG	14
3.1. Xây dựng mô hình – Training model	14
3.1.1. Datasets	14
3.1.2. Xây dựng mô hình.....	14
3.2. Xây dựng Web App với Streamlit.....	43
CHƯƠNG 4. KẾT LUẬN.....	44
4.1. Kết quả đạt được	44
4.1.1. Mô hình chuẩn đoán.....	44

4.1.2. WebApp	45
4.2. Nhược điểm	45
4.3. Hướng phát triển.....	46
TÀI LIỆU THAM KHẢO	47
PHỤ LỤC	48

DANH MỤC HÌNH ẢNH

Hình 1.1. Giới thiệu về Facial Landmark.....	1
Hình 1.2. Ví dụ về các điểm Landmark trên khuôn mặt người	2
Hình 2.1. Ví dụ về Convolutional	5
Hình 2.2. Hình ảnh trắng đen được số hóa.....	6
Hình 2.3. Hình ảnh được Convoled feature	6
Hình 2.4. Cấu trúc mạng CNN.....	7
Hình 2.5. Ma trận 28x28	8
Hình 4.1. Hình dự đoán 1	44
Hình 4.2. Hình dự đoán 2	45

DANH MỤC BẢNG

Bảng 3.1. Datasets	14
---------------------------------	----

DANH MỤC CÁC TỪ VIẾT TẮT

Viết tắt	Ý nghĩa
AI	<u>A</u> rtificial <u>I</u> ntelligent
CNN	<u>C</u> onvolutional <u>N</u> eural <u>N</u> etwork

CHƯƠNG 1. TỔNG QUAN

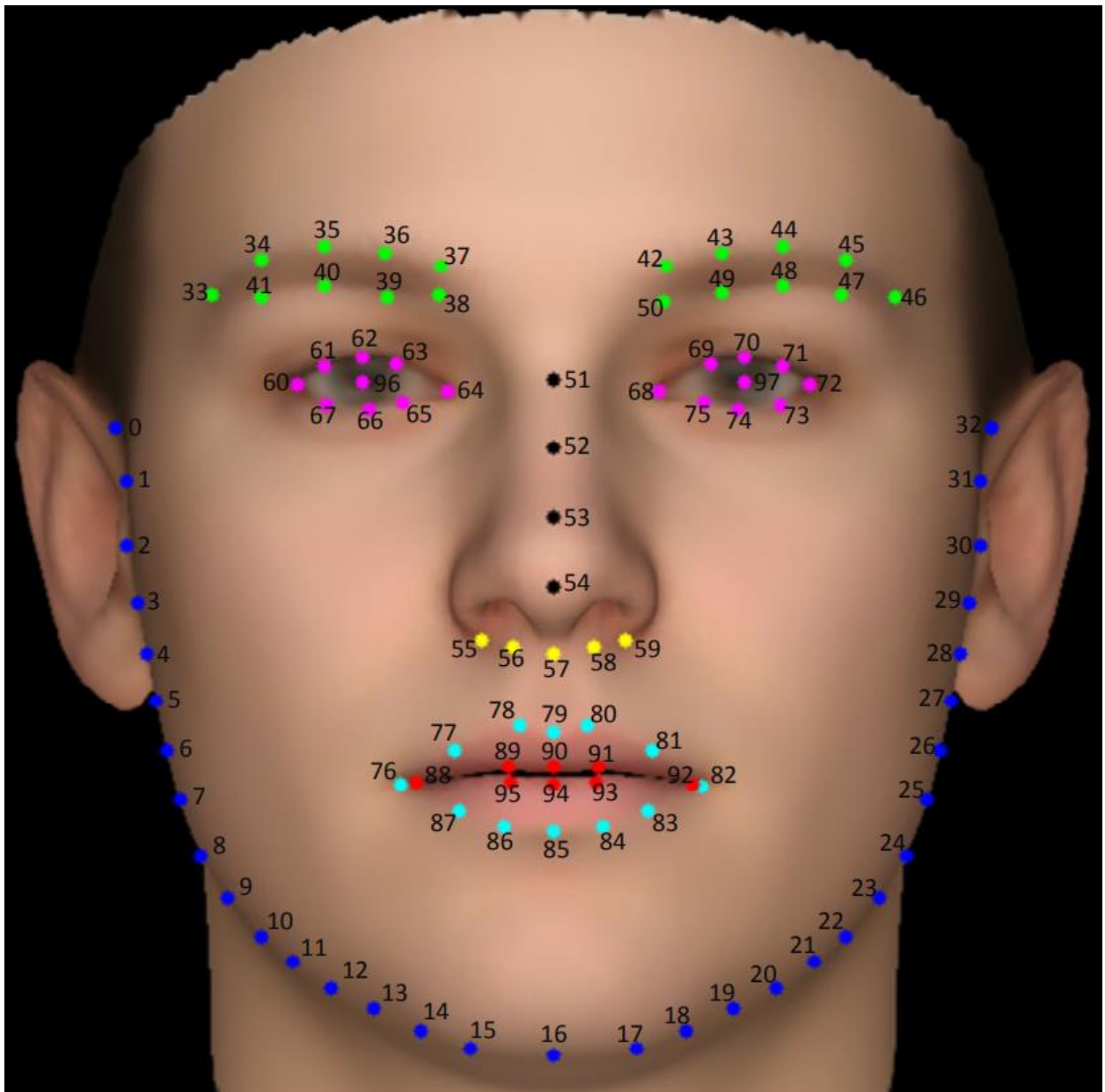
1.1. Facial Landmark là gì?



Hình 1.1. Giới thiệu về Facial Landmark

Facial Landmark được định nghĩa là các điểm mốc trên khuôn mặt người mà kết quả của bài toán này chính là dự đoán ra các điểm chính tạo nên hình dạng của một đối tượng, cụ thể là khuôn mặt người trong một bức ảnh. Facial Landmark là đầu vào hay còn gọi là bài toán tiên quyết cần phải giải quyết của nhiều bài toán khác như dự đoán tư thế đầu, hoán đổi khuôn mặt, phát hiện nháy mắt, phát hiện cảm xúc, xoay chỉnh cấu trúc khuôn mặt hay dễ hình dung nhất là việc xây dựng FaceID, các Filter trong các phần mềm chụp và chỉnh sửa ảnh,...

Facial Landmark Detection được mô tả dễ hiểu bằng cách cho một hình ảnh với đối tượng chính là khuôn mặt người nhìn trực diện hoặc nghiêng vào ống kính để tiến hành nhận dạng. Mục tiêu chính là dự đoán hình dạng hoặc cố gắng định vị được điểm quan trọng theo chính hình dạng đó. Từ đó có thể phát hiện được điểm tạo nên cấu trúc khuôn mặt dựa trên các phương pháp nhận dạng. Sau đó sẽ tiến hành đánh dấu lại các điểm đó, lưu và tạo thành hình ảnh mới với tỷ lệ trùng khớp phụ thuộc vào khả năng làm việc của phương pháp nhận dạng đó.



Hình 1.2. Ví dụ về các điểm Landmark trên khuôn mặt người

1.2. Lý do chọn đề tài

Việc khai thác và sử dụng Facial Landmark hiện nay đang dần trở nên vô cùng phổ biến và ứng dụng rộng rãi trong nhiều lĩnh vực có sử dụng đến yếu tố xử lý hình ảnh. Quá trình nghiên cứu và tiến hành thực nghiệm nhận dạng các điểm mốc trên khuôn mặt nhằm nâng cao chất lượng, cải tiến chất lượng của các đề tài có quy mô hay yêu cầu lớn hơn. Bởi như đã đề cập ở mục trên 1.1, Facial Landmark Detection là bài toán đầu tiên và tiên quyết, nếu muốn tiếp cận được với các đề tài cao hơn, bài toán phức tạp hơn, việc nắm vững bài toán này sẽ thúc đẩy khả năng làm việc với các mô hình bài toán lớn hơn. Hướng tới việc nghiên cứu chuyên sâu về xử lý ảnh được dễ dàng hơn.

Quá trình giải quyết bài toán này được thực hiện bởi trí tuệ nhân tạo thông qua các thư viện hỗ trợ Deeplearning, có cơ hội thực hành chuyên sâu với thuật toán Mạng Nơ-ron tích tụ (CNN), được thực hiện thử nghiệm trên Google Colaboratory nhằm tiết kiệm tài nguyên máy tính, làm cho quá trình học tập và nghiên cứu về AI được dễ dàng hơn,... Chính vì lý do trên nên đề xuất được thực hiện đề tài:

“NHẬN DIỆN CÁC ĐIỂM LANDMARK TRÊN KHUÔN MẶT ỨNG DỤNG CNN”

1.3. Mục tiêu nghiên cứu

- Sử dụng CNN như một phương pháp nghiên cứu tìm ra các điểm Facial Landmarks
- Thiết kế Web Apps để nhận diện các điểm mốc trên khuôn mặt bằng hình ảnh.

1.4. Phương pháp nghiên cứu

- Đưa ra các lý thuyết về phương pháp nhận diện khuôn mặt, điểm trên khuôn mặt
- Dựa vào các đề tài nghiên cứu có sẵn trên Internet tìm ra các phương pháp tối ưu
- Từ những phương pháp chọn lọc, tạo ra các Models nhận dạng dựa trên Deep Learning, sử dụng chúng để nhận dạng thực tế.

1.5. Nội dung báo cáo

Bài báo cáo này sẽ trình bày tập trung chủ yếu thuật toán tạo ra một Model dựa trên CNN bằng cách tối ưu hóa, chỉ khai thác các yếu tố cần thiết nhằm tạo ra một phương pháp có khả năng nhận dạng ở mức độ chính xác cao nhất. Sau đó tiến hành đưa vào Web App để thuận tiện cho quá trình kiểm tra, nghiệm thu.

Bài báo cáo này được xây dựng dựa trên 4 chương, bao gồm:

CHƯƠNG 1: TỔNG QUAN

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

CHƯƠNG 3: ỨNG DỤNG

CHƯƠNG 4: KẾT LUẬN

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. Thuật toán CNN - Convolutional Neural Network

2.1.1. Convolutional Neural Network là gì

Convolutional Neural Network (CNN hoặc ConvNet) được tạm dịch ra thành “Mạng nơ ron tích chập” là một trong những mô hình Deep Learning mà trong đó sử dụng nhiều thuật toán thu được mô hình dữ liệu trừu tượng hóa ở mức nâng cao bằng việc sử dụng nhiều lớp xử lý cấu trúc, trong đó đối tượng làm việc chủ yếu là hình ảnh. Bằng cách phân tích hình ảnh, CNN là một lớp của mạng nơ ron sâu. Nó giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao và trực quan hóa kết quả.

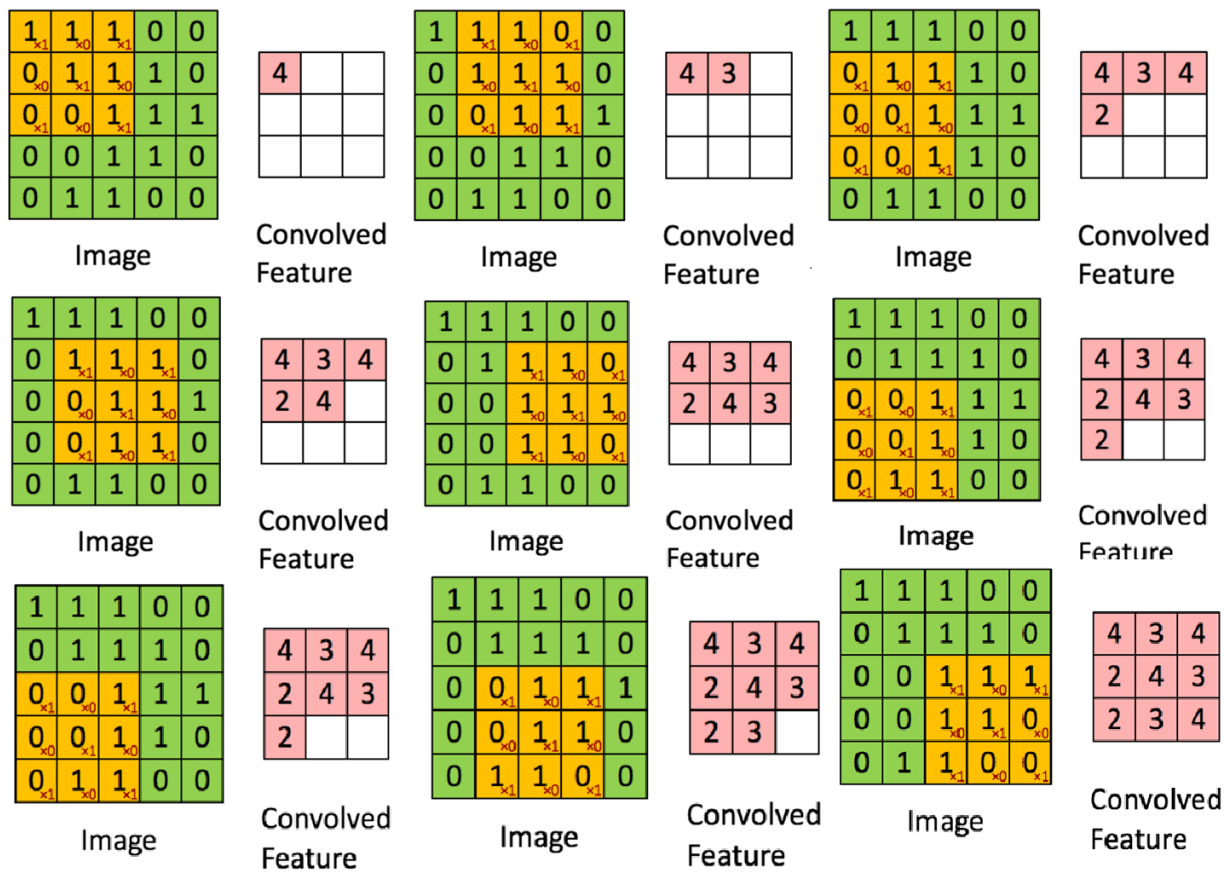
Lấy ví dụ cơ bản, ta có thể sử dụng lớp mạng này trong các ứng dụng như nhận dạng hình ảnh, nhận dạng khuôn mặt, hay thuật toán được ứng dụng nhiều vào quảng cáo tự động trên các nền tảng mạng xã hội như Facebook, Instagram, Google, Youtube,...

Convolutional Neural Network là hệ thống mạng nhận đầu vào là một mảng hai chiều và hoạt động trực tiếp trên hình ảnh thay vì tập trung trích xuất tính năng thường thấy ở các mạng nơ ron khác. Do đó, để tìm hiểu CNN là gì cần tập trung vào một số thuật ngữ như sau:

Feature được dịch theo nghĩa tiếng Việt là *đặc điểm*. Khi sử dụng thuật toán CNN so sánh hình ảnh theo từng mảnh, mỗi mảnh đó được gọi là Feature.

Mỗi Feature được xem như một hình ảnh mini hay gọi là những mảng hai chiều nhỏ. Các Feature được khớp với những khía cạnh chung trong bức ảnh đó. Nghĩa là Feature sẽ tương ứng với khía cạnh nào đó của hình ảnh và chúng sẽ khớp lại với nhau.

Convolutional được hiểu là *tích chập*. Xét về cơ bản, khi xem một hình ảnh mới, thuật toán CNN sẽ không nhận biết được nó ở vị trí nào, các Feature sẽ khớp với nhau ở đâu? Chính vì vậy, Convolutional sẽ thử chúng với tất cả các vị trí khác nhau và tạo thành một bộ lọc gọi là Filter. Quá trình này được thực hiện thông qua phần toán nơ ron tích chập. Convolutional được hiểu như một cửa sổ trượt (Sliding Windows) trên một ma trận



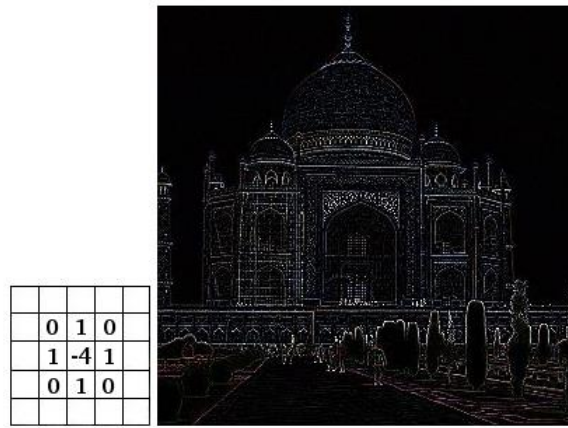
Hình 2.1. Ví dụ về Convolutional

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.

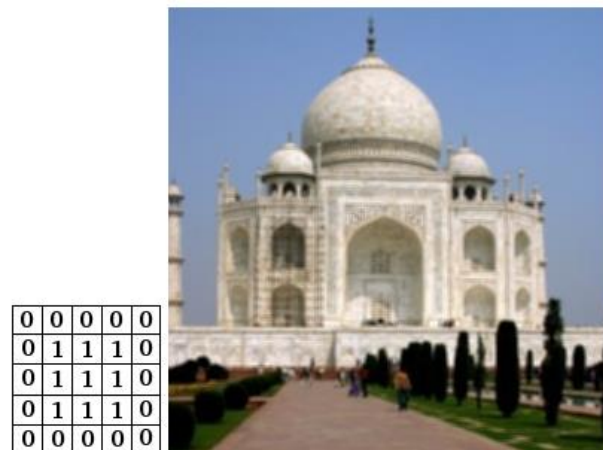
Trong hình ảnh ví dụ Hình 2.2, ma trận bên trái là một hình ảnh trắng đen được số hóa. Ma trận có kích thước 5×5 và mỗi điểm ảnh có giá trị 1 hoặc 0 là giao điểm của dòng và cột.

Convolution hay tích chập là nhân từng phần tử trong ma trận 3. 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.

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 Convoled feature được sinh ra từ việc nhân ma trận Filter với ma trận ảnh 5×5 bên trái.



Hình 2.2. Hình ảnh trắng đen được số hóa



Hình 2.3. Hình ảnh được Convoled feature

2.1.2. Cấu trúc của 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.

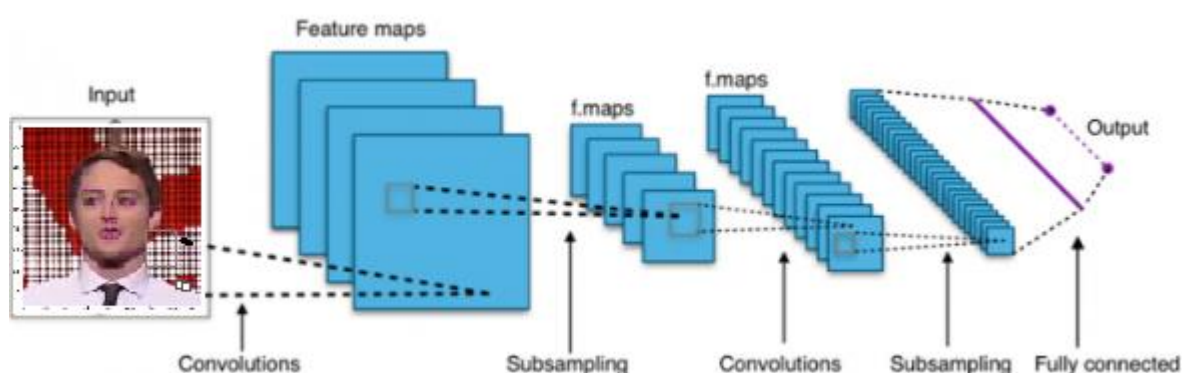
Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo. Trong mô hình mạng truyền ngược (feedforward neural network) thì mỗi neural đầu vào (input node) cho mỗi neural đầu ra trong các lớp tiếp theo.

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 (training) 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 2.4. 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 góc 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.

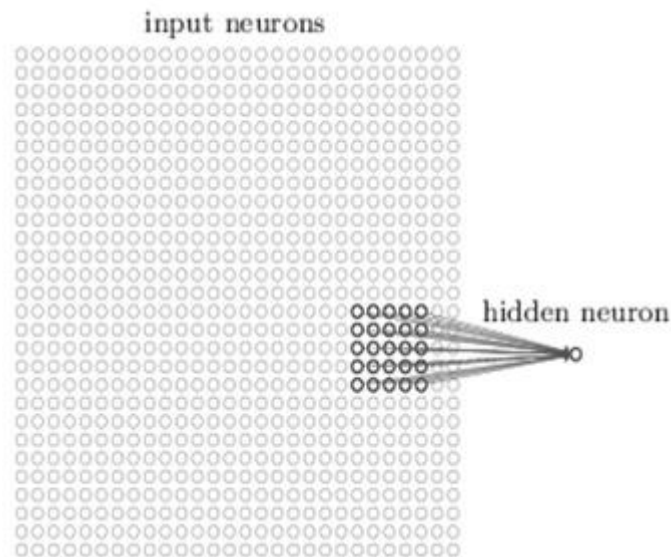
Mạng CNN sử dụng 3 ý tưởng cơ bản:

- **Các trường tiếp nhận cục bộ** (local receptive field)
- **Trọng số chia sẻ** (shared weights)
- **Tổng hợp** (pooling).

2.1.2.1. 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 đây gọi là một trường tiếp nhận cục bộ.



Hình 2.5. Ma trận 28×28

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.

2.1.2.2. Trọng số chia sẻ (shared weight and bias)

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. 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.

2.1.2.3. Lớp tổng hợp (pooling layer)

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

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.

2.2. Thư viện OpenCv

OpenCV là tên viết tắt của open source computer vision library – có thể được hiểu là một thư viện nguồn mở cho máy tính. Cụ thể hơn OpenCV là kho lưu trữ các mã nguồn mở được dùng để xử lý hình ảnh, phát triển các ứng dụng đồ họa trong thời gian thực.

OpenCV cho phép cải thiện tốc độ của CPU khi thực hiện các hoạt động real time. Nó còn cung cấp một số lượng lớn các mã xử lý phục vụ cho quy trình của thị giác máy tính hay các learning machine khác.

OpenCV có cấu trúc module, tức là nó bao gồm cả những thư viện liên kết tĩnh lẫn thư viện liên kết động. Nắm rõ các module của OpenCV sẽ giúp bạn đọc hoàn toàn thấu hiểu OpenCV là gì.

- Core functionality (core): Module này sở hữu cơ chế rất nhỏ gọn. Nó được dùng để định hình các cấu trúc của cơ sở dữ liệu cơ bản, bao gồm cả những mảng đa chiều. Ngoài ra nó còn xác định các chức năng của những module đi kèm khác nữa.
- Image Processing (imgproc): Đây là module được dùng cho quá trình xử lý hình ảnh. Nó cho phép người dùng thực hiện các hoạt động như lọc hình ảnh tuyến tính và phi tuyến, thực hiện phép biến hình, thay đổi không gian màu, xây dựng biểu đồ và rất nhiều thao tác khác liên quan.

- Video Analysis (video): Giống như tên gọi của nó, module này cho phép phân tích các video. Kết quả được trả về bao gồm các ước tính chuyển động, thực hiện tách nền và các phép toán theo dõi vật thể.
- Camera Calibration and 3D Reconstruction (calib3d): Module này cung cấp các thuật toán hình học đa chiều cơ bản và hiệu chuẩn máy ảnh single và stereo. Ngoài ra nó còn đưa ra các dự đoán kiểu dáng của đối tượng và sử dụng thuật toán thư tín âm thanh nổi cùng các yếu tố tái tạo 3D.
- 2D Features Framework (features2d): Module này giúp phát hiện các tính năng nổi trội của bộ nhận diện, bộ truy xuất thông số và thông số đối chọi
- Ngoài ra còn có rất nhiều module khác với đa dạng tính năng, ví dụ như: FLANN, Google test wrapper...

OpenCV được cho là một phần mềm đa nhiệm. Nó được ứng dụng trong rất nhiều trường hợp khác nhau. Ví dụ, ta sẽ nói về các phần mềm định vị, bản đồ nói chung. Hẳn rằng trong chúng ta ai cũng đã có ít nhất một lần cần sử dụng đến các map online đúng không. Bạn sử dụng các map để tìm đường, tra cứu tình hình giao thông hoặc đơn giản là xem xét các hình ảnh thực tế của địa điểm cần đến. Những lúc như vậy, OpenCV đóng vai trò là nhà cung cấp dữ liệu hình ảnh cho các app về Map này. OpenCV sẽ đem đến cho người dùng hình ảnh về đường phố hay các căn nhà, con người xung quanh địa điểm được chỉ định.

OpenCV còn được dùng để khởi tạo ra những hình ảnh 3 chiều phức tạp. Hoạt động này rất được yêu thích, nhất là trong thời đại trí tuệ nhân tạo AI phát triển như thế này. OpenCV thường được viết bởi ngôn ngữ Python, nhờ các câu lệnh ngắn gọn cùng thuộc tính đơn giản, Python giúp cho quá trình phát triển phần mềm OpenCV diễn ra dễ dàng hơn. Sử dụng ngôn ngữ Python sẽ là biện pháp tốt nhất cho những người không mạnh mẽ lập trình. Điểm trừ của Python là vì có cấu tạo quá đơn giản nên một số tính năng cần sự phức tạp sẽ bị hạn chế.

Đối với các công nghệ hiện đại, OpenCV cũng là một yếu tố không thể thiếu. Tất cả những ứng dụng công nghệ như robot, xe tự lái, bảng cảm ứng thông minh... đều có sự góp mặt của OpenCV trong khâu xử lý hình ảnh. Ví dụ gần gũi nhất trong cuộc sống có thể kể đến hệ thống mở khóa điện thoại bằng cách nhận diện khuôn mặt người dùng.

2.3. Thư viện Pytorch

Pytorch chính là một framework hỗ trợ Deep Learning được phát triển bởi Facebook. (Bên cạnh Amazon, Google hay Apple, Facebook được biết đến là đơn vị công nghệ đầu tư rất nhiều nguồn lực cho việc phát triển trí tuệ nhân tạo).

Những lợi ích Pytorch đem lại:

- **Mã nguồn mở:** như đã chia sẻ ở trên, nhờ sử dụng mã nguồn mở đã tạo nên một cộng đồng rất lớn với nguồn tài nguyên “chất lượng” và “số lượng”.
- **Khả năng xử lý đồ họa:** như Numpy đồng thời có kiểm soát CPU & GPU rõ ràng.
- Tập hợp nhiều Pythonic trong tự nhiên.
- Dễ dàng xử lý khi gặp bug.
- Có **TouchScript** được xem là một tập hợp con của Python. Tập hợp này giúp triển khai các ứng dụng vào quy mô sản xuất từ đó mở rộng quy mô. Đồng thời khi nói đến việc xây dựng các nguyên mẫu với tốc độ nhanh, **sử dụng Pytorch** được ưu tiên hơn so với Tensorflow vì nó nhẹ hơn.
- Các hàm, **cú pháp cơ bản trong Pytorch** giúp xử lý các **bài toán về AI** nhanh chóng.

Các tính năng chính của PyTorch

- **Giao diện thân thiện** - PyTorch cung cấp API dễ sử dụng; do đó nó được coi là rất đơn giản để vận hành và chạy trên Python. Việc thực thi mã trong khuôn khổ này khá dễ dàng.
- **Sử dụng Python** - Thư viện này được coi là Pythonic tích hợp trơn tru với ngăn xếp khoa học dữ liệu Python. Do đó, nó có thể tận dụng tất cả các dịch vụ và chức năng được cung cấp bởi môi trường Python.
- **Đồ thị tính toán** - PyTorch cung cấp một nền tảng tuyệt vời cung cấp đồ thị tính toán động. Do đó người dùng có thể thay đổi chúng trong thời gian chạy. Điều này rất hữu ích khi nhà phát triển không biết cần bao nhiêu bộ nhớ để tạo mô hình mạng nơ-ron.

PyTorch được biết đến với ba cấp độ trừu tượng như được đưa ra dưới đây

- Tensor – Mảng n-chiều bắt buộc chạy trên GPU.
- Variable – Nút trong đồ thị tính toán. Lưu trữ dữ liệu và gradient.
- Module – Lớp mạng nơ-ron sẽ lưu trữ trạng thái hoặc trọng số có thể học được.

2.4. Thư viện Pillow

Pillow là một fork từ thư viện PIL của Python được sử dụng để xử lý hình ảnh. So với PIL thì Pillow được cập nhật thường xuyên và đánh giá cao hơn. (PIL đã không được cập nhật từ năm 2009). Pillow là bản nâng cấp của PIL và là một lựa chọn đáng tin cậy nếu bạn đang có một dự án cần phải làm việc với nhiều hình ảnh.

Những khả năng của Pillow:

- Khi sử dụng Pillow, bạn không chỉ có thể mở và lưu hình ảnh, mà còn có thể xử lý đặc điểm của hình ảnh, chẳng hạn như màu sắc, độ mờ, độ sáng, tối, ...
- Pillow hỗ trợ xử lý nhiều tệp hình ảnh khác nhau như PDF, WebP, PCX, PNG, JPEG, GIF, PSD, WebP, PCX, GIF, IM, EPS, ICO, BMP, và còn nhiều hơn thế.
- Có thể dễ dàng tạo ra thumbnails cho hình ảnh, những ảnh thu nhỏ này mang hầu hết đặc điểm của hình ảnh và gần như không khác gì ảnh gốc ngoại trừ chúng được thu nhỏ lại.
- Hỗ trợ một bộ sưu tập các bộ lọc hình ảnh như – FIND_EDGES, DETAIL, SMOOTH, BLUR, CONTOUR, SHARPEN, SMOOTH_MORE,...

2.5. Streamlit - A faster way to build and share data apps

Streamlit là một open-source Python lib, nó giúp ta dễ dàng tạo một web app cho MachineLearning và Data Science. Đối với một người có rất ít kiến thức về HTML, CSS, JavaScript thì đây có lẽ là công cụ rất phù hợp với mình để demo các sản phẩm AI. Chúng ta có thể test cục bộ được, sau đó muốn deploy lên internet có thể dùng Heroku, Streamlit hay Ngrok đều được.

Streamlit cho phép người dùng sử dụng các lệnh đơn giản, dễ dàng thực hiện. Kết hợp Streamlit và GitHub cho phép một hệ sinh thái vô cùng phong phú và đa dạng ứng dụng hữu ích, từ trang tổng quan đến mạng sâu và hơn thế nữa.

Model sau khi train xong được đưa lên Github, và sử dụng các lệnh của Streamlit để hiển thị thành webapp, điều này khá đơn giản và tiết kiệm thời gian. Link webapp dễ dàng được truy cập bởi những ai có link, đây là một điều khá tiện lợi.

Để cài đặt:

```
pip install streamlit
```

Sau khi cài đặt xong có thể chạy ngay câu lệnh sau, nó đưa chúng ta tới tab với đường link <http://localhost:8501/>

```
streamlit hello
```

Sau khi cài xong bạn có thể chạy ứng dụng với cú pháp sau:

```
streamlit run app.py
```

Để Deploy webapp streamlit chúng ta chỉ cần đưa lên github và liên kết chúng lại, quá trình deploy sẽ xử lý và đưa ra link web để sử dụng.

CHƯƠNG 3. ỨNG DỤNG

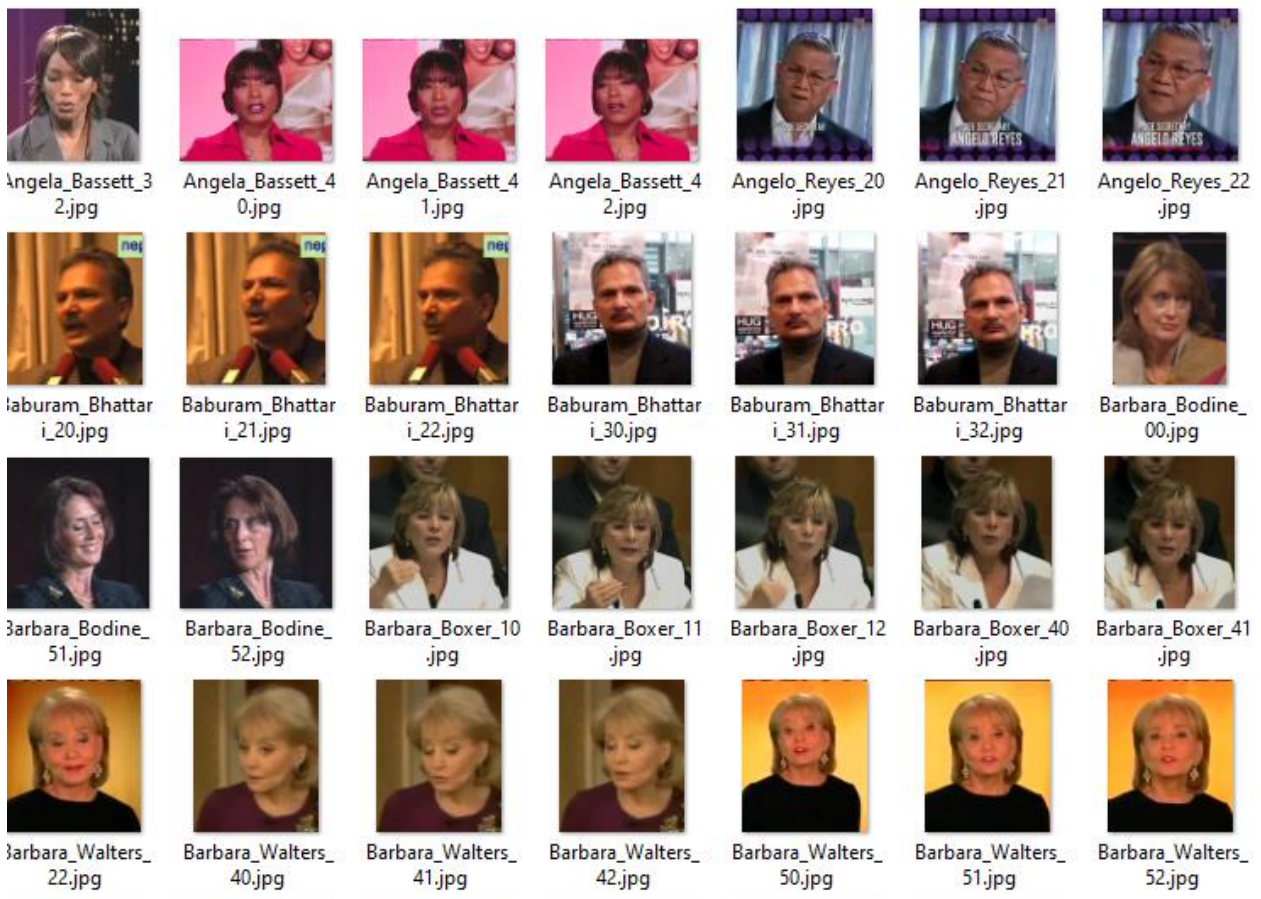
3.1. Xây dựng mô hình – Training model

Tiến hành sử dụng Google Colab để thực hiện đề tài, nhằm tiết kiệm tài nguyên máy tính, do Colab có các chức năng hỗ trợ nâng cao giúp tăng tốc quá trình learning.

3.1.1. Datasets

Trong bài báo cáo này, sử dụng nguồn dữ liệu được tổng hợp nhiều nguồn trên Internet, trong đó sử dụng bộ hình ảnh và các điểm mốc dữ liệu chứa 68 điểm trên mỗi ảnh. Tổng cộng bao gồm:

Số lượng ảnh dùng cho File Test	2308
Số lượng ảnh dùng cho File Train	3462



\ **Hình 3.** Hình ảnh từ dataset

3.1.2. Xây dựng mô hình

Liên kết Google Drive

```
from google.colab import drive
```

```
drive.mount('/content/drive')
```

Khai báo các thư viện cần thiết

```
#Import the required libraries
import glob
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import cv2
from torch.utils.data import Dataset, DataLoader
import torch
from torchvision import transforms, utils
```

Lấy dữ liệu là lớp frame với các điểm mốc

```
[3] key_pts_frame = pd.read_csv('/content/drive/MyDrive/Facial_Landmark_Detection/training_frames_keypoints.csv')

n = 0
image_name = key_pts_frame.iloc[n, 0]
key_pts = key_pts_frame.iloc[n, 1:].to_numpy()
key_pts = key_pts.astype('float').reshape(-1, 2)

print('Image name: ', image_name)
print('Landmarks shape: ', key_pts.shape)
print('First 4 key pts: {}'.format(key_pts[:4]))

Image name: Luis_Fonsi_21.jpg
Landmarks shape: (68, 2)
First 4 key pts: [[ 45.  98.]
 [ 47. 106.]
 [ 49. 110.]
 [ 53. 119.]]
```

Các điểm mốc này, được đánh dấu dựa trên từng ảnh theo các vị trí của 68 điểm mốc trên khuôn mặt tương ứng, sau đó được lưu vào file .csv. Như trên hình ta thử in ra dữ liệu điểm mốc của người tên là Luis_Fonsi trong dataset.

Kiểm tra các giá trị và dataset đã được đưa vào hoàn chỉnh chưa

```
# print out some stats about the data
print('Number of images: ', key_pts_frame.shape[0])

img2 = cv2.imread('/content/drive/MyDrive/Facial_Landmark_Detection/training/Daniel_Radcliffe_00.jpg')
width, height, channels = img2.shape
npixel = img2.size
print("npixel", npixel)
print("width {}, height {}, channels {}".format(width, height, channels))
```

Number of images: 3462
npixel 38784
width 128, height 101, channels 3

Từ đường dẫn đã lưu trữ lấy hình ảnh của Daniel_Radcliffe và show các thông số như tổng số ảnh, pixel và các kích thước.

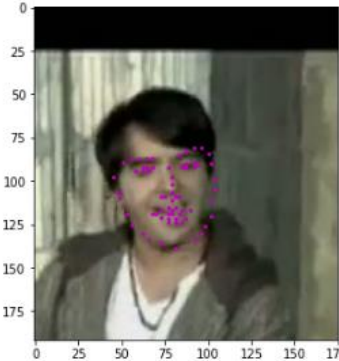
Tạo hàm kết hợp giữa dữ liệu các facial landmark và hình ảnh tương ứng

```
def show_keypoints(image, key_pts):
    """Show image with keypoints"""
    plt.imshow(image)
    plt.scatter(key_pts[:, 0], key_pts[:, 1], s=20, marker='.', c='m')
```

```
# Display a few different types of images by changing the index n

# select an image by index in our data frame
n = 0
image_name = key_pts_frame.iloc[n, 0]
key_pts = key_pts_frame.iloc[n, 1:].to_numpy()
key_pts = key_pts.astype('float').reshape(-1, 2)

plt.figure(figsize=(5, 5))
show_keypoints(mping.imread(os.path.join('/content/drive/MyDrive/Facial_Landmark_Detection/training', image_name)), key_pts)
plt.show()
```



Dựa vào hàm kết hợp Show_keypoints(), so sánh kết quả ta thấy được dữ liệu các điểm mốc trên khuôn mặt được ghép chính xác với ảnh được hiển thị

Hàm gọi và chỉnh sửa trên toàn bộ dataset

```
[ ] class FacialKeypointsDataset(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, csv_file, root_dir, transform=None):
        """
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """
        self.key_pts_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

    def __len__(self):
        return len(self.key_pts_frame)

    def __getitem__(self, idx):
        image_name = os.path.join(self.root_dir,
                                   self.key_pts_frame.iloc[idx, 0])

        image = mpimg.imread(image_name)

        # if image has an alpha color channel, get rid of it
        if(image.shape[2] == 4):
            image = image[:, :, 0:3]

        key_pts = self.key_pts_frame.iloc[idx, 1:].to_numpy()
        key_pts = key_pts.astype('float').reshape(-1, 2)
        sample = {'image': image, 'keypoints': key_pts}

        if self.transform:
            sample = self.transform(sample)

        return sample
```

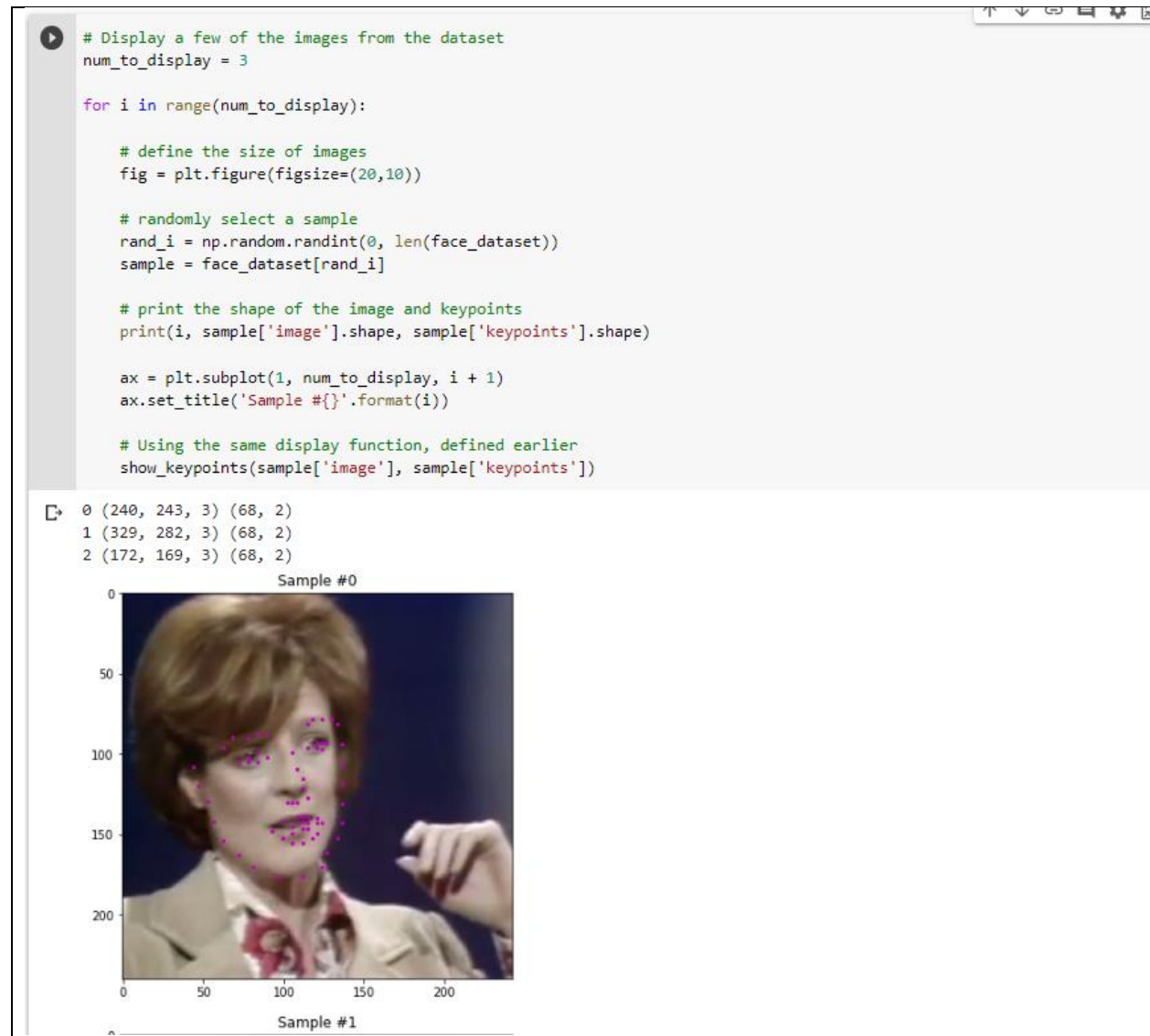
Cấu trúc lại toàn bộ dataset trong file training

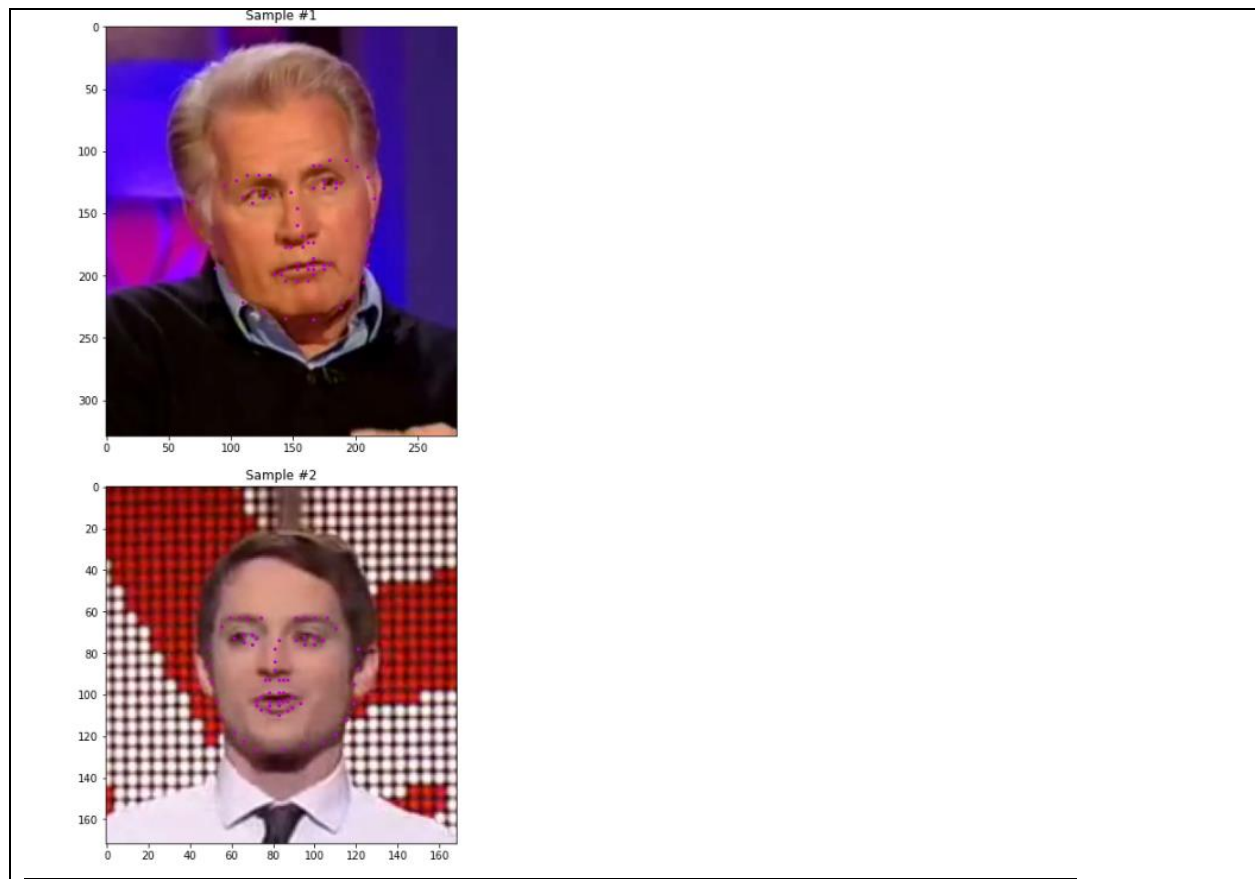
```
▶ # Construct the dataset
face_dataset = FacialKeypointsDataset(csv_file='/content/drive/MyDrive/Facial_Landmark_Detection/training_frames_keypoints.csv'
                                       root_dir='/content/drive/MyDrive/Facial_Landmark_Detection/training')

# print some stats about the dataset
print('Length of dataset: ', len(face_dataset))

☐ Length of dataset: 3462
```

Hiển thị một số hình ảnh từ dataset sau khi đã kết hợp dữ liệu điểm ảnh





Hiệu chỉnh lại màu của hình ảnh thành grayscale

```
# tranforms

class Normalize(object):
    """Convert a color image to grayscale and normalize the color range to [0,1]."""

    def __call__(self, sample):
        image, key_pts = sample['image'], sample['keypoints']

        image_copy = np.copy(image)
        key_pts_copy = np.copy(key_pts)

        # convert image to grayscale
        image_copy = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

        # scale color range from [0, 255] to [0, 1]
        image_copy = image_copy/255.0

        # scale keypoints to be centered around 0 with a range of [-1, 1]
        # mean = 100, sqrt = 50, so, pts should be (pts - 100)/50
        key_pts_copy = (key_pts_copy - 100)/50.0

        return {'image': image_copy, 'keypoints': key_pts_copy}
```

Chỉnh sửa lại kích thước hình ảnh

```
class Rescale(object):
    """Rescale the image in a sample to a given size.

    Args:
        output_size (tuple or int): Desired output size. If tuple, output is
            matched to output_size. If int, smaller of image edges is matched
            to output_size keeping aspect ratio the same.
    """

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        self.output_size = output_size

    def __call__(self, sample):
        image, key_pts = sample['image'], sample['keypoints']

        h, w = image.shape[:2]
        if isinstance(self.output_size, int):
            if h > w:
                new_h, new_w = self.output_size * h / w, self.output_size
            else:
                new_h, new_w = self.output_size, self.output_size * w / h
        else:
            new_h, new_w = self.output_size

        new_h, new_w = int(new_h), int(new_w)

        img = cv2.resize(image, (new_w, new_h))

        # scale the pts, too
        key_pts = key_pts * [new_w / w, new_h / h]

        return {'image': img, 'keypoints': key_pts}
```

Cắt ngẫu nhiên hình ảnh để đồng bộ kích thước của toàn bộ hình ảnh

```
class Rescale(object):
    """Rescale the image in a sample to a given size.

    Args:
        output_size (tuple or int): Desired output size. If tuple, output is
            matched to output_size. If int, smaller of image edges is matched
            to output_size keeping aspect ratio the same.
    """

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        self.output_size = output_size

    def __call__(self, sample):
        image, key_pts = sample['image'], sample['keypoints']

        h, w = image.shape[:2]
        if isinstance(self.output_size, int):
            if h > w:
                new_h, new_w = self.output_size * h / w, self.output_size
            else:
                new_h, new_w = self.output_size, self.output_size * w / h
        else:
            new_h, new_w = self.output_size

        new_h, new_w = int(new_h), int(new_w)

        img = cv2.resize(image, (new_w, new_h))

        # scale the pts, too
        key_pts = key_pts * [new_w / w, new_h / h]

        return {'image': img, 'keypoints': key_pts}
```

Chuyển đổi mảng hình ảnh đưa vào Tensors

```
class ToTensor(object):
    """Convert ndarrays in sample to Tensors."""

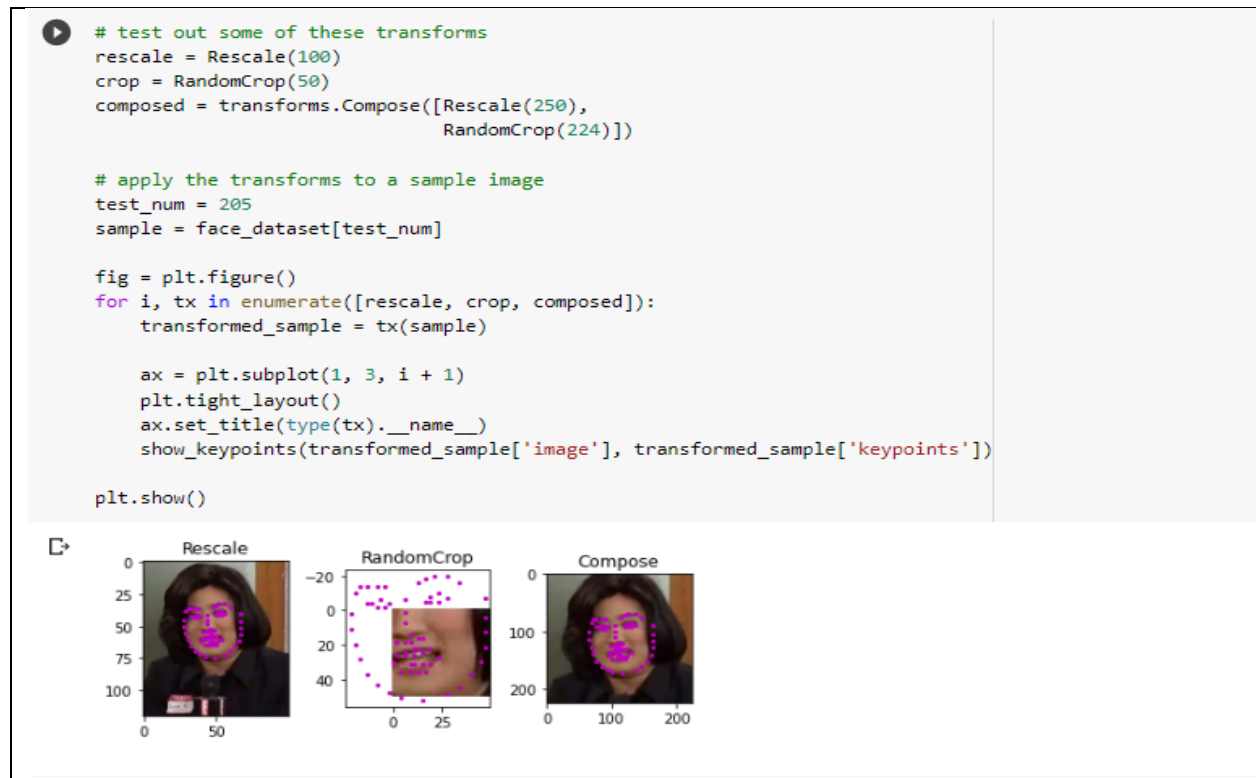
    def __call__(self, sample):
        image, key_pts = sample['image'], sample['keypoints']

        # if image has no grayscale color channel, add one
        if len(image.shape) == 2:
            # add that third color dim
            image = image.reshape(image.shape[0], image.shape[1], 1)

        # swap color axis because
        # numpy image: H x W x C
        # torch image: C x H x W
        image = image.transpose((2, 0, 1))

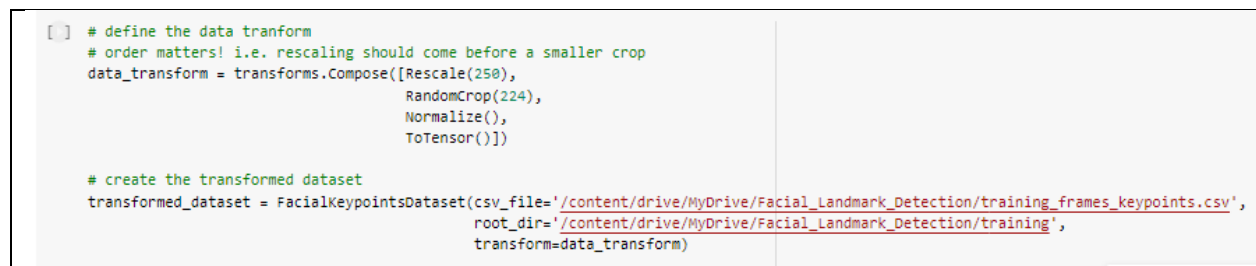
        return {'image': torch.from_numpy(image),
                'keypoints': torch.from_numpy(key_pts)}
```

Kiểm tra một số hình ảnh random



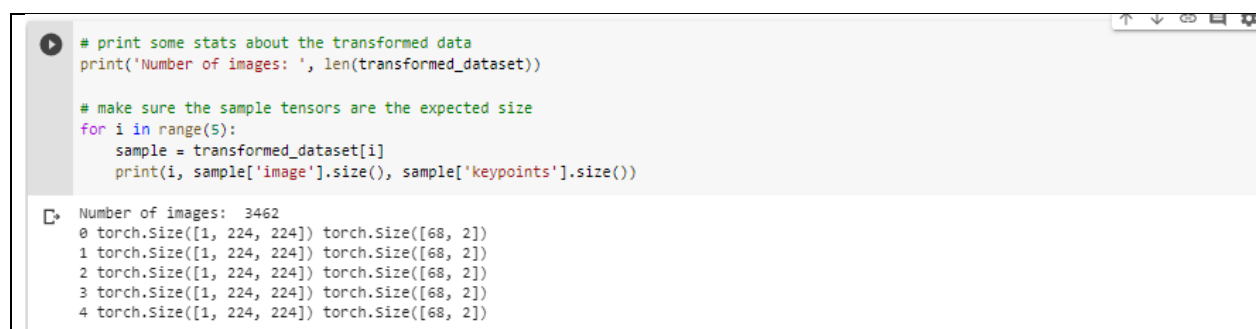
Kiểm tra hình ảnh ngẫu nhiên sau khi trải qua ba quá trình hiệu chỉnh thu được kết quả như trên.

Định nghĩa lại cấu trúc chuyển đổi data



Dựa vào các hàm đã viết, lựa chọn các thông số tối ưu sau đó tạo ra dataset đã được hiệu chỉnh toàn bộ sau đó lưu lại.

Kiểm tra lại một số trạng thái của data



Load các file chứa các hàm hỗ trợ quá trình training

```
from google.colab import files
src = list(files.upload().values())[0]
open('workspace_utils.py', 'wb').write(src)

[ ] from google.colab import files
src = list(files.upload().values())[0]
open('models.py', 'wb').write(src)

[ ] from google.colab import files
src = list(files.upload().values())[0]
open('data_load.py', 'wb').write(src)
```

Trong đó nội dung của các file lần lượt như sau:

“**workspace_utils.py**”: Có nhiệm vụ giữ cho trình duyệt luôn chạy tránh trường hợp dừng do lỗi kết nối với Drive.

```
import signal

from contextlib import contextmanager

import requests

DELAY = INTERVAL = 4 * 60 # interval time in seconds
MIN_DELAY = MIN_INTERVAL = 2 * 60
KEEPALIVE_URL = "https://nebula.udacity.com/api/v1/remote/keep-alive"
TOKEN_URL =
"http://metadata.google.internal/computeMetadata/v1/instance/attributes/keep_alive_token"
TOKEN_HEADERS = {"Metadata-Flavor": "Google"}

def _request_handler(headers):
    def _handler(signum, frame):
        requests.request("POST", KEEPALIVE_URL, headers=headers)
    return _handler

@contextmanager
def active_session(delay=DELAY, interval=INTERVAL):
    """
    Example:

    from workspace_utils import active_session
```

```

with active_session():
    # do long-running work here.
    """
    token = requests.request("GET", TOKEN_URL, headers=TOKEN_HEADERS).text
    headers = {'Authorization': "STAR " + token}
    delay = max(delay, MIN_DELAY)
    interval = max(interval, MIN_INTERVAL)
    original_handler = signal.getsignal(signal.SIGALRM)
    try:
        signal.signal(signal.SIGALRM, _request_handler(headers))
        signal.setitimer(signal.ITIMER_REAL, delay, interval)
        yield
    finally:
        signal.signal(signal.SIGALRM, original_handler)
        signal.setitimer(signal.ITIMER_REAL, 0)

def keep_awake(iterable, delay=DELAY, interval=INTERVAL):
    """
    Example:

    from workspace_utils import keep_awake

    for i in keep_awake(range(5)):
        # do iteration with lots of work here
    """
    with active_session(delay, interval): yield from iterable

```

“**models.py**”: Có nhiệm vụ định nghĩa cấu trúc của một mạng CNN tiêu chuẩn

```

## Define the convolutional neural network architecture

import torch
import torch.nn as nn
import torch.nn.functional as F
# can use the below import should you choose to initialize the weights of your
Net
import torch.nn.init as I

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()

        ## Define all the layers of this CNN, the only requirements are:
        ## 1. This network takes in a square (same width and height), grayscale
        image as input

```

```

    ## 2. It ends with a linear layer that represents the keypoints
    ## Last layer output 136 values, 2 for each of the 68 keypoint (x, y)
pairs

    # 1 input image channel (grayscale), 32 output channels/feature maps,
5x5 square convolution kernel

    ## Shape of a Convolutional Layer
    # K - out_channels : the number of filters in the convolutional layer
    # F - kernel_size
    # S - the stride of the convolution
    # P - the padding
    # W - the width/height (square) of the previous layer

    # Since there are F*F*D weights per filter
    # The total number of weights in the convolutional layer is K*F*F*D

    # 224 by 224 pixels

    ## self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size,
stride=1, padding=0)
    # output size = (W-F)/S + 1 = (224-5)/1 + 1 = 220
    # the output Tensor for one image, will have the dimensions: (1, 220,
220)
    # after one pool layer, this becomes (10, 13, 13)
    self.conv1 = nn.Conv2d(1, 32, 5)

    # maxpool layer
    # pool with kernel_size=2, stride=2
    self.pool = nn.MaxPool2d(2, 2)

    # 220/2 = 110
    # output size = (W-F)/S + 1 = (110-3)/1 + 1 = 108
    # the output Tensor for one image, will have the dimensions: (32, 110,
110)
    self.conv2 = nn.Conv2d(32, 64, 3)

    # output size = (W-F)/S + 1 = (54-3)/1 + 1 = 52
    # the output Tensor for one image, will have the dimensions: (64, 54,
54)
    self.conv3 = nn.Conv2d(64, 128, 3)

    # output size = (W-F)/S + 1 = (26-3)/1 + 1 = 24
    # the output Tensor for one image, will have the dimensions: (128, 26,
26)
    self.conv4 = nn.Conv2d(128, 256, 3)

    # output size = (W-F)/S + 1 = (12-3)/1 + 1 = 10
    # the output Tensor for one image, will have the dimensions: (256, 12,
12)

```

```

self.conv5 = nn.Conv2d(256, 512, 1)

# output size = (W-F)/S + 1 = (6-1)/1 + 1 = 6
# the output Tensor for one image, will have the dimensions: (512, 6,
6)

# Fully-connected (linear) layers
self.fc1 = nn.Linear(512*6*6, 1024)
self.fc2 = nn.Linear(1024, 512)
self.fc3 = nn.Linear(512, 68*2)

# Dropout
self.dropout = nn.Dropout(p=0.25)

def forward(self, x):
    ## Define the feedforward behavior of this model
    ## x is the input image and, as an example, here you may choose to
include a pool/conv step:

    # 5 conv/relu + pool layers
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = self.pool(F.relu(self.conv3(x)))
    x = self.pool(F.relu(self.conv4(x)))
    x = self.pool(F.relu(self.conv5(x)))

    # Prep for linear layer / Flatten
    x = x.view(x.size(0), -1)

    # linear layers with dropout in between
    x = F.relu(self.fc1(x))
    x = self.dropout(x)
    x = F.relu(self.fc2(x))
    x = self.dropout(x)
    x = self.fc3(x)

    return x

```

“**data_load.py**”: Có nhiệm vụ tổng hợp các hàm đã được viết ở trên bao gồm chỉnh liên kết giữa dataset và file .csv, hiệu chỉnh các yếu tố cần thiết chuẩn hóa dataset

```

import glob
import os
import torch
from torch.utils.data import Dataset, DataLoader
import numpy as np

```



```

import matplotlib.image as mpimg
import pandas as pd
import cv2

class FacialKeypointsDataset(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, csv_file, root_dir, transform=None):
        """
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """
        self.key_pts_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

    def __len__(self):
        return len(self.key_pts_frame)

    def __getitem__(self, idx):
        image_name = os.path.join(self.root_dir,
                                   self.key_pts_frame.iloc[idx, 0])

        image = mpimg.imread(image_name)

        # if image has an alpha color channel, get rid of it
        if(image.shape[2] == 4):
            image = image[:, :, 0:3]

        key_pts = self.key_pts_frame.iloc[idx, 1:].to_numpy()
        key_pts = key_pts.astype('float').reshape(-1, 2)
        sample = {'image': image, 'keypoints': key_pts}

        if self.transform:
            sample = self.transform(sample)

        return sample

# tranforms

class Normalize(object):
    """Convert a color image to grayscale and normalize the color range to
    [0,1]."""

```

```

def __call__(self, sample):
    image, key_pts = sample['image'], sample['keypoints']

    image_copy = np.copy(image)
    key_pts_copy = np.copy(key_pts)

    # convert image to grayscale
    image_copy = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

    # scale color range from [0, 255] to [0, 1]
    image_copy = image_copy/255.0

    # scale keypoints to be centered around 0 with a range of [-1, 1]
    # mean = 100, sqrt = 50, so, pts should be (pts - 100)/50
    key_pts_copy = (key_pts_copy - 100)/50.0

    return {'image': image_copy, 'keypoints': key_pts_copy}

class Rescale(object):
    """Rescale the image in a sample to a given size.
    Args:
        output_size (tuple or int): Desired output size. If tuple, output is
            matched to output_size. If int, smaller of image edges is matched
            to output_size keeping aspect ratio the same.
    """

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        self.output_size = output_size

    def __call__(self, sample):
        image, key_pts = sample['image'], sample['keypoints']

        h, w = image.shape[:2]
        if isinstance(self.output_size, int):
            if h > w:
                new_h, new_w = self.output_size * h / w, self.output_size
            else:
                new_h, new_w = self.output_size, self.output_size * w / h
        else:
            new_h, new_w = self.output_size

        new_h, new_w = int(new_h), int(new_w)

        img = cv2.resize(image, (new_w, new_h))

        # scale the pts, too

```

```

        key_pts = key_pts * [new_w / w, new_h / h]

        return {'image': img, 'keypoints': key_pts}

class RandomCrop(object):
    """Crop randomly the image in a sample.

    Args:
        output_size (tuple or int): Desired output size. If int, square crop
            is made.
    """

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        if isinstance(output_size, int):
            self.output_size = (output_size, output_size)
        else:
            assert len(output_size) == 2
            self.output_size = output_size

    def __call__(self, sample):
        image, key_pts = sample['image'], sample['keypoints']

        h, w = image.shape[:2]
        new_h, new_w = self.output_size

        top = np.random.randint(0, h - new_h)
        left = np.random.randint(0, w - new_w)

        image = image[top: top + new_h,
                       left: left + new_w]

        key_pts = key_pts - [left, top]

        return {'image': image, 'keypoints': key_pts}

class ToTensor(object):
    """Convert ndarrays in sample to Tensors."""

    def __call__(self, sample):
        image, key_pts = sample['image'], sample['keypoints']

        # if image has no grayscale color channel, add one
        if len(image.shape) == 2:
            # add that third color dim
            image = image.reshape(image.shape[0], image.shape[1], 1)

        # swap color axis because

```

```

# numpy image: H x W x C
# torch image: C X H X W
image = image.transpose((2, 0, 1))

return {'image': torch.from_numpy(image),
        'keypoints': torch.from_numpy(key_pts)}

```

Duy trì khả năng training của máy trong một thời gian dài

```

[ ] # import the usual resources
import matplotlib.pyplot as plt
import numpy as np

# import utilities to keep workspaces alive during model training
from workspace_utils import active_session

# watch for any changes in model.py, if it changes, re-load it automatically
%load_ext autoreload
%autoreload 2

```

Tạo Net trong file “models.py”

```

[ ] ## Define the Net in models.py

import torch
import torch.nn as nn
import torch.nn.functional as F

## Once you've define the network, you can instantiate it
# one example conv layer has been provided for you
from models import Net

net = Net()
print(net)

Net(
  (conv1): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
  (conv5): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1))
  (fc1): Linear(in_features=18432, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=136, bias=True)
  (dropout): Dropout(p=0.25, inplace=False)
)

```

Lưu data sau khi chuyển đổi đưa vào file “model.py”

```
# create the transformed dataset
transformed_dataset = FacialKeypointsDataset(csv_file='/content/drive/MyDrive/Facial_Landmark_Detection/training_frames_keypoints.csv',
                                             root_dir='/content/drive/MyDrive/Facial_Landmark_Detection/training',
                                             transform=data_transform)

print('Number of images: ', len(transformed_dataset))

# iterate through the transformed dataset and print some stats about the first few samples
for i in range(4):
    sample = transformed_dataset[i]
    print(i, sample['image'].size(), sample['keypoints'].size())
```

Number of images: 3462
0 torch.Size([1, 224, 224]) torch.Size([68, 2])
1 torch.Size([1, 224, 224]) torch.Size([68, 2])
2 torch.Size([1, 224, 224]) torch.Size([68, 2])
3 torch.Size([1, 224, 224]) torch.Size([68, 2])

Load data từ file training vào batch_size

```
# load training data in batches
batch_size = 10

train_loader = DataLoader(transformed_dataset,
                           batch_size=batch_size,
                           shuffle=True,
                           num_workers=4)
```

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarning: This DataLoader will create 4 worker processes with os.setpgrp() calling os.setpgid(0, 0) and then setting the process group to os.getpid(). It should be noted that changing the group has marked support for the POSIX realtime API deprecated, and that this code has been tested thoroughly on Linux with PyTorch version 1.1.0 and above. If you're running on a different platform, you may want to verify the relevant behavior of the setpgid() function on your platform. (Triggered internally at /usr/lib/python3.7/torch/_utils.py:565.)

Sử dụng hình ảnh từ file “test” để đưa vào batch_size

```
# load in the test data, using the dataset class
# AND apply the data_transform you defined above

# create the test dataset
test_dataset = FacialKeypointsDataset(csv_file='/content/drive/MyDrive/Facial_Landmark_Detection/test_frames_keypoints.csv',
                                       root_dir='/content/drive/MyDrive/Facial_Landmark_Detection/test',
                                       transform=data_transform)

# load test data in batches
batch_size = 10

test_loader = DataLoader(test_dataset,
                         batch_size=batch_size,
                         shuffle=True,
                         num_workers=4)
```

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarning: This DataLoader will create 4 worker processes with os.setpgrp() calling os.setpgid(0, 0) and then setting the process group to os.getpid(). It should be noted that changing the group has marked support for the POSIX realtime API deprecated, and that this code has been tested thoroughly on Linux with PyTorch version 1.1.0 and above. If you're running on a different platform, you may want to verify the relevant behavior of the setpgid() function on your platform. (Triggered internally at /usr/lib/python3.7/torch/_utils.py:565.)

Test thử model dựa trên hình ảnh từ file “test”

```
[ ] # test the model on a batch of test images

def net_sample_output():

    # iterate through the test dataset
    for i, sample in enumerate(test_loader):

        # get sample data: images and ground truth keypoints
        images = sample['image']
        key_pts = sample['keypoints']

        # convert images to FloatTensors
        images = images.type(torch.FloatTensor)

        # forward pass to get net output
        output_pts = net(images)

        # reshape to batch_size x 68 x 2 pts
        output_pts = output_pts.view(output_pts.size()[0], 68, -1)

        # break after first image is tested
        if i == 0:
            return images, output_pts, key_pts
```

Gọi các hàm đã viết sẵn ở trên, trả về các giá trị hình ảnh test, điểm dự đoán

```
[ ] # call the above function
# returns: test images, test predicted keypoints, test ground truth keypoints
test_images, test_outputs, gt_pts = net_sample_output()

# print out the dimensions of the data to see if they make sense
print(test_images.data.size())
print(test_outputs.data.size())
print(gt_pts.size())
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarning: This DataLoader will create
  cpuset_checked))
torch.Size([10, 1, 224, 224])
torch.Size([10, 68, 2])
torch.Size([10, 68, 2])
```

```
def show_all_keypoints(image, predicted_key_pts, gt_pts=None):
    """Show image with predicted keypoints"""

    # image is grayscale
    plt.imshow(image, cmap='gray')
    plt.scatter(predicted_key_pts[:, 0], predicted_key_pts[:, 1], s=20, marker='.', c='m')
    # plot ground truth points as green pts
    if gt_pts is not None:
        plt.scatter(gt_pts[:, 0], gt_pts[:, 1], s=20, marker='.', c='g')
```

Kiểm tra bằng cách cho hiển thị điểm tiên đoán và điểm được đánh dấu sẵn

```
# visualize the output
# by default this shows a batch of 10 images
def visualize_output(test_images, test_outputs, gt_pts=None, batch_size=10):

    for i in range(batch_size):
        plt.figure(figsize=(20,10))
        ax = plt.subplot(1, batch_size, i+1)

        # un-transform the image data
        image = test_images[i].data # get the image from it's Variable wrapper
        image = image.numpy() # convert to numpy array from a Tensor
        image = np.transpose(image, (1, 2, 0)) # transpose to go from torch to numpy image

        # un-transform the predicted key_pts data
        predicted_key_pts = test_outputs[i].data
        predicted_key_pts = predicted_key_pts.numpy()
        # undo normalization of keypoints
        predicted_key_pts = predicted_key_pts*50.0+100

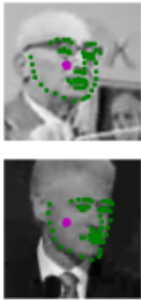
        # plot ground truth points for comparison, if they exist
        ground_truth_pts = None
        if gt_pts is not None:
            ground_truth_pts = gt_pts[i]
            ground_truth_pts = ground_truth_pts*50.0+100

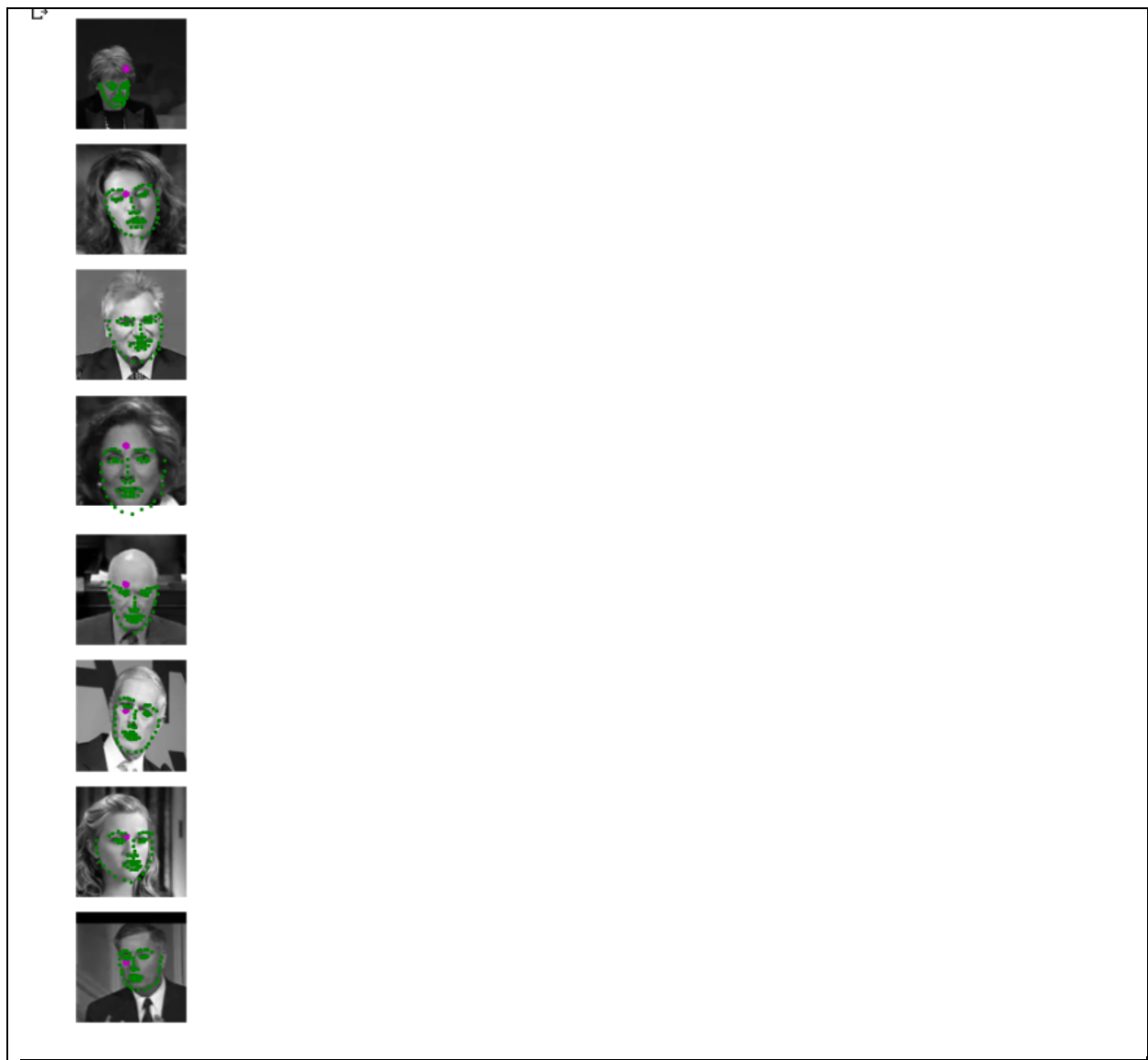
        # call show_all_keypoints
        show_all_keypoints(np.squeeze(image), predicted_key_pts, ground_truth_pts)

        plt.axis('off')

    plt.show()

# call it
visualize_output(test_images, test_outputs, gt_pts)
```





Trong đó các điểm màu xanh là các điểm được chấm sẵn từ trước còn các điểm màu hồng là các điểm tiên đoán.

Tiến hành training cho model

```
▶ ## Define the loss and optimization
import torch.optim as optim

criterion = nn.MSELoss()
optimizer = optim.Adam(net.parameters(), lr = 0.001)

[ ] def train_net(n_epochs):

    # prepare the net for training
    net.train()
    training_loss = []

    for epoch in range(n_epochs): # loop over the dataset multiple times

        running_loss = 0.0

        # train on batches of data, assumes you already have train_loader
        for batch_i, data in enumerate(train_loader):
            # get the input images and their corresponding labels
            images = data['image']
            key_pts = data['keypoints']

            # flatten pts
            key_pts = key_pts.view(key_pts.size(0), -1)

            # convert variables to floats for regression loss
            key_pts = key_pts.type(torch.FloatTensor)
            images = images.type(torch.FloatTensor)

            # forward pass to get outputs
            output_pts = net(images)

            # calculate the loss between predicted and target keypoints
            loss = criterion(output_pts, key_pts)

            # zero the parameter (weight) gradients
            optimizer.zero_grad()

            # backward pass to calculate the weight gradients
            loss.backward()

            # update the weights
            optimizer.step()
```

```

        # update the weights
        optimizer.step()

        # print loss statistics
        running_loss += loss.item()
        if batch_i % 10 == 9:    # print every 10 batches
            print('Epoch: {}, Batch: {}, Avg. Loss: {}'.format(epoch + 1, batch_i+1, running_loss/10))
            running_loss = 0.0
        training_loss.append(running_loss)

    print('Finished Training')
    return training_loss

```

```

[ ] # train your network
n_epochs = 20 # start small, and increase when you've decided on your model structure and hyperparams

# this is a Workspaces-specific context manager to keep the connection
# alive while training your model, not part of pytorch
with active_session():
    training_loss = train_net(n_epochs)
# visualize the loss as the network trained
plt.figure()
plt.semilogy(training_loss)
plt.grid()
plt.xlabel('Epoch')
plt.ylabel('Loss');

```

```

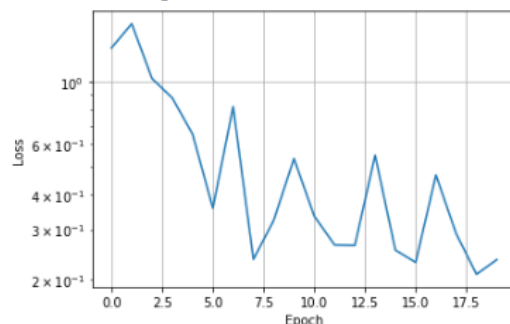
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarning: This DataLoader will create 4 worker
cpuset_checked))
Epoch: 1, Batch: 10, Avg. Loss: 0.5423878163099289
Epoch: 1, Batch: 20, Avg. Loss: 0.33685930669307707
Epoch: 1, Batch: 30, Avg. Loss: 0.2515428841114044
Epoch: 1, Batch: 40, Avg. Loss: 0.19551145955920218
Epoch: 1, Batch: 50, Avg. Loss: 0.17550639137625695
Epoch: 1, Batch: 60, Avg. Loss: 0.20681693106889726
Epoch: 1, Batch: 70, Avg. Loss: 0.17810240238904954
Epoch: 1, Batch: 80, Avg. Loss: 0.20534133464097976
Epoch: 1, Batch: 90, Avg. Loss: 0.16638070791959764
Epoch: 1, Batch: 100, Avg. Loss: 0.20531949549913406
Epoch: 1, Batch: 110, Avg. Loss: 0.21487596854567528
Epoch: 1, Batch: 120, Avg. Loss: 0.23099478036165239
Epoch: 1, Batch: 130, Avg. Loss: 0.3048130556941032
Epoch: 1, Batch: 140, Avg. Loss: 0.17162472903728485
Epoch: 1, Batch: 150, Avg. Loss: 0.19590983390808106
Epoch: 1, Batch: 160, Avg. Loss: 0.20824026241898536
Epoch: 1, Batch: 170, Avg. Loss: 0.22408104464411735
Epoch: 1, Batch: 180, Avg. Loss: 0.20407070157603345

```

```

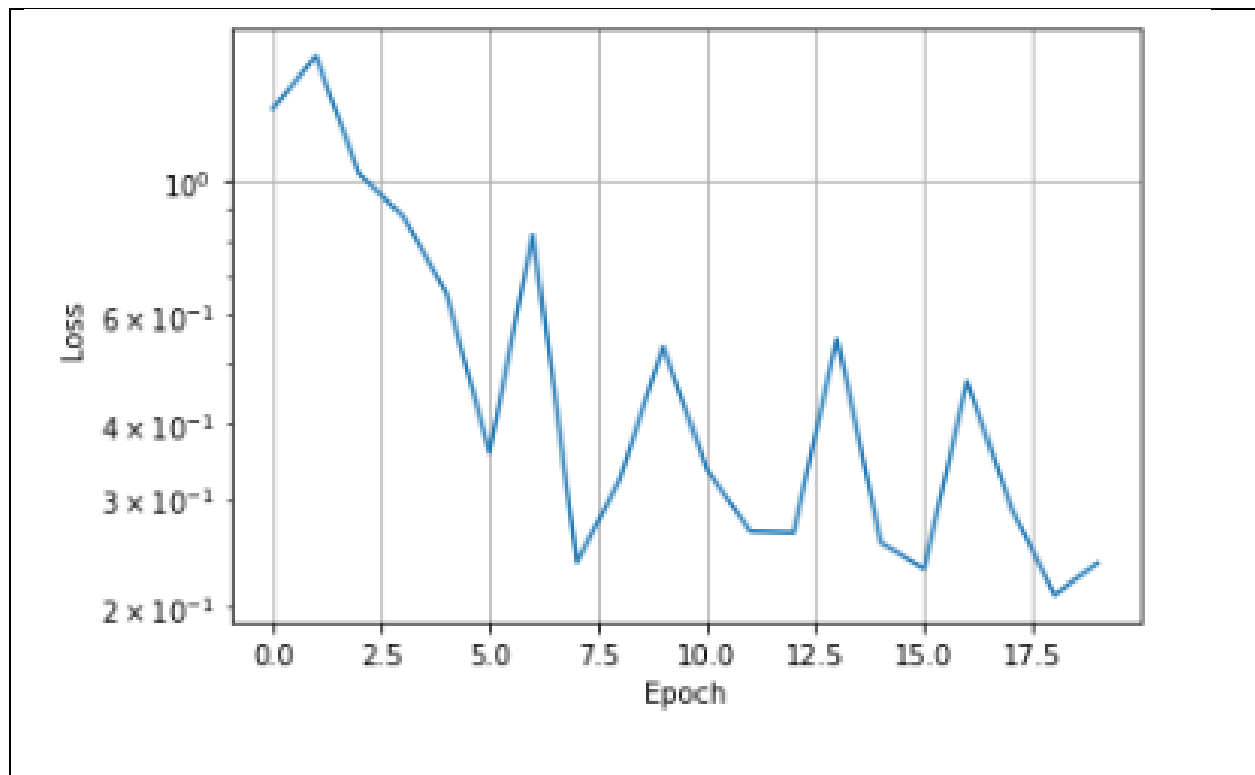
Epoch: 20, Batch: 270, Avg. Loss: 0.02866520769894123
Epoch: 20, Batch: 280, Avg. Loss: 0.03137787189334631
Epoch: 20, Batch: 290, Avg. Loss: 0.03339725993573665
Epoch: 20, Batch: 300, Avg. Loss: 0.03538747988641262
Epoch: 20, Batch: 310, Avg. Loss: 0.041439698450267314
Epoch: 20, Batch: 320, Avg. Loss: 0.031641971133649346
Epoch: 20, Batch: 330, Avg. Loss: 0.024384322762489318
Epoch: 20, Batch: 340, Avg. Loss: 0.02973825577646494
Finished Training

```



Sau 20 lần học, in đồ thị thể hiện giá trị loss giảm dần qua từng lần học. Kết quả model đạt giá trị chính xác 98%.

Đồ thị giá trị loss



Một số hình ảnh so sánh kết quả giữa model sau khi train và giá trị thực tế

```
# get a sample of test data again
test_images, test_outputs, gt_pts = net_sample_output()

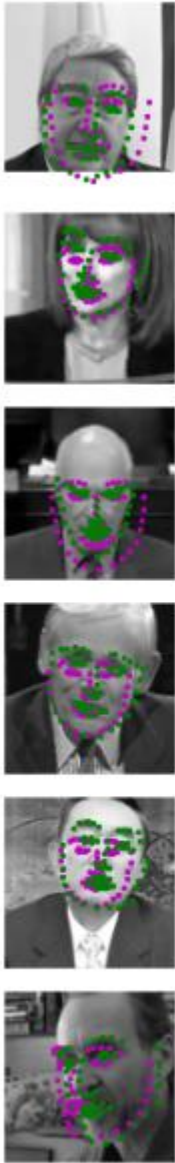
print(test_images.data.size())
print(test_outputs.data.size())
print(gt_pts.size())
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarning:
  cpuset_checked))
torch.Size([10, 1, 224, 224])
torch.Size([10, 68, 2])
torch.Size([10, 68, 2])
```

```
## visualize test output
# you can use the same function as before, by un-commenting the line below:

visualize_output(test_images, test_outputs, gt_pts)
```





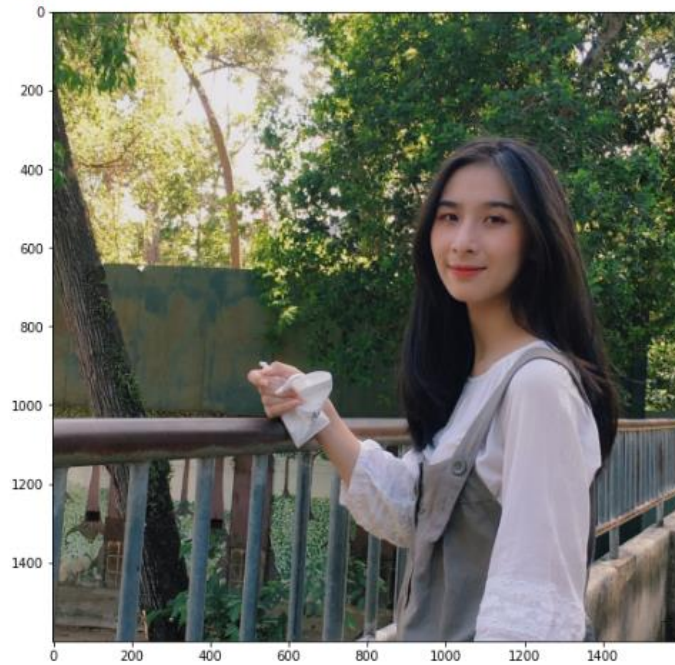
Trong đó các điểm màu xanh là các điểm được chấm sẵn từ trước còn các điểm màu hồng là các điểm tiên đoán.

Test thử hình ảnh thực tế khi sử dụng model sau khi train

```
[ ] ## load in color image for face detection
image = cv2.imread('/content/drive/MyDrive/Facial_Landmark_Detection/image_test/beauty_1648987254064.JPG')
# switch red and blue color channels
# --> by default OpenCV assumes BLUE comes first, not RED as in many images
gray = image.copy()
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# plot the image
fig = plt.figure(figsize=(9,9))
plt.imshow(image)
```

<matplotlib.image.AxesImage at 0x7f16807b0350>



Dùng haarcascade để xác định vị trí khuôn mặt



Tiến hành nhận diện các điểm dựa trên model đã train

```
▶ image_copy = np.copy(image)
#Including a padding to extract face as HAAR classifier's bounding box, crops sections of the face

PADDING = 100
images, keypoints = [], []

# loop over the detected faces from your haar cascade
for (x,y,w,h) in faces:

    # Select the region of interest that is the face in the image
    roi = image_copy[y-PADDING:y+h+PADDING, x-PADDING:x+w+PADDING]

    ## Convert the face region from RGB to grayscale
    roi = cv2.cvtColor(roi, cv2.COLOR_RGB2GRAY)

    ## Normalize the grayscale image so that its color range falls in [0,1] instead of [0,255]
    roi = (roi / 255.).astype(np.float32)

    ## Rescale the detected face to be the expected square size for your CNN (224x224, suggested)
    roi = cv2.resize(roi, (224, 224))
    images.append(roi)

    ## Reshape the numpy image shape (H x W x C) into a torch image shape (C x H x W)
    if len(roi.shape) == 2:
        roi = np.expand_dims(roi, axis=0)
    else:
        roi = np.rollaxis(roi, 2, 0)

    # Match the convolution dimensions
    roi = np.expand_dims(roi, axis=0)

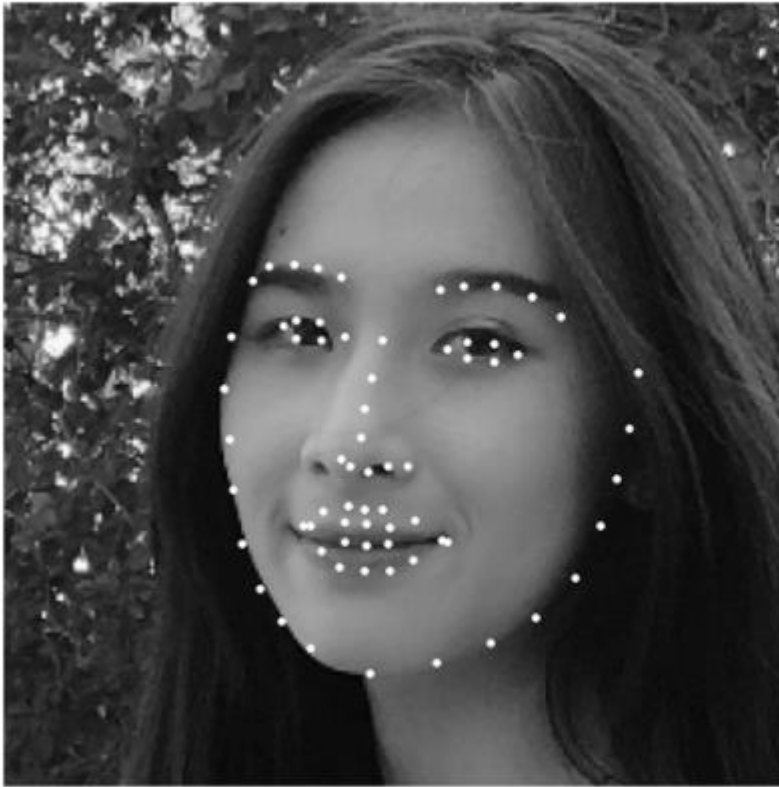
    ## Make facial keypoint predictions using your loaded, trained network
    # Forward pass
    roi = torch.from_numpy(roi).type(torch.FloatTensor)
    output_pts = net.forward(roi)

    output_pts = output_pts.view(output_pts.size()[0], 68, -1)
    keypoints.append(output_pts[0])

## Display each detected face and the corresponding keypoints
show_all_keypoints(images, keypoints)
```



```
## Display each detected face and the corresponding keypoints  
show_all_keypoints(images, keypoints)
```



3.2. Xây dựng webapp với Streamlit

Streamlit là một open-source Python lib, nó giúp ta dễ dàng tạo một web app cho MachineLearning và Data Science. Đối với một người có rất ít kiến thức về HTML, CSS, JavaScript thì đây có lẽ là công cụ rất phù hợp với mình để demo các sản phẩm AI. Chúng ta có thể test cục bộ được, sau đó muốn deploy lên internet có thể dùng Heroku, Streamlit hay Ngrok đều được.

Trong đề tài này, mô hình sau khi đã train được lưu dưới dạng file .pt, model sẽ được load lên streamlit, tiến hành xử lý hình ảnh theo các thông số đầu vào như đã training ở trên và đưa ra dự đoán, webapp sẽ được deploy thông qua trang chủ Streamlit để dễ dàng chia sẻ.

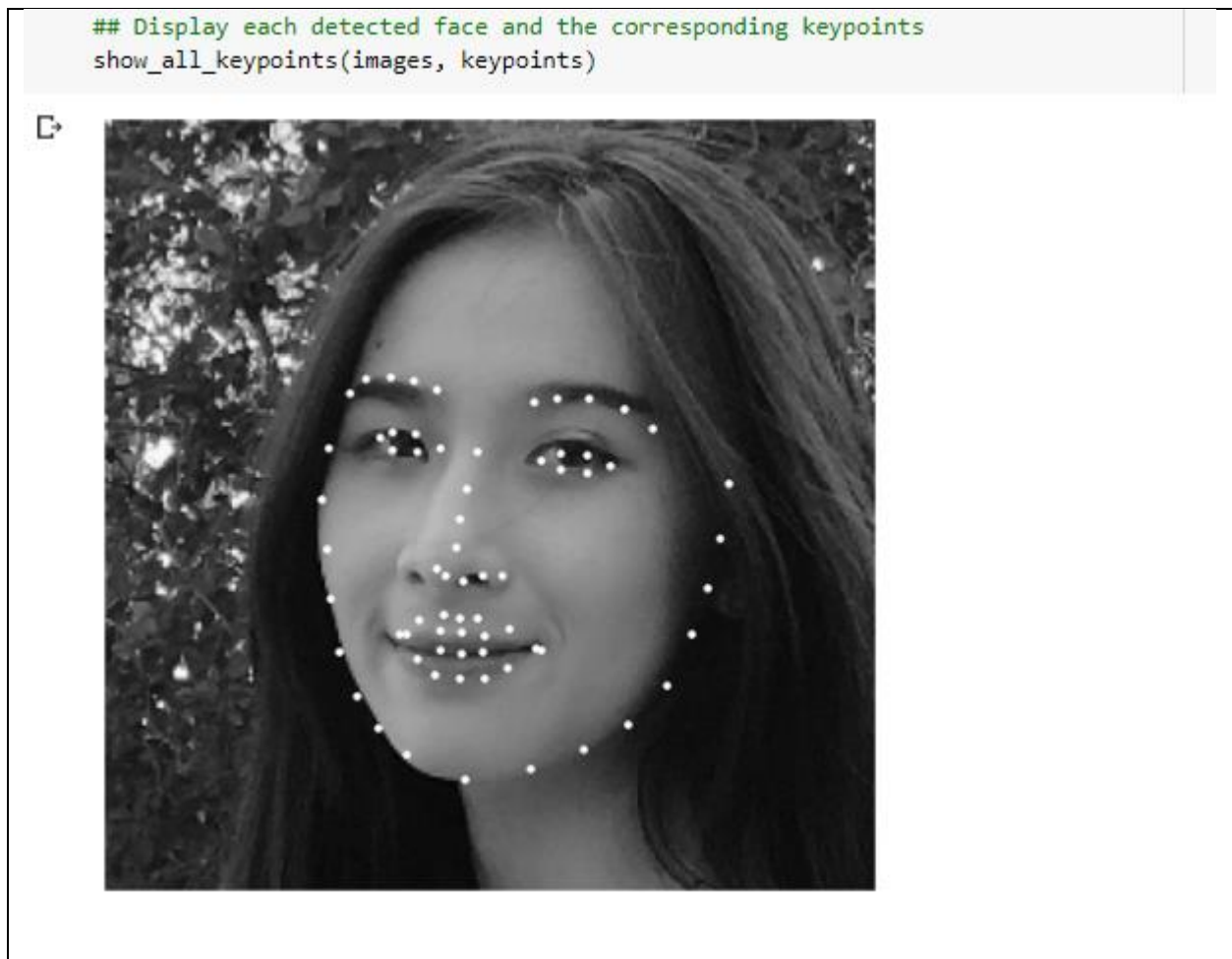
CHƯƠNG 4. KẾT LUẬN

4.1. Kết quả đạt được

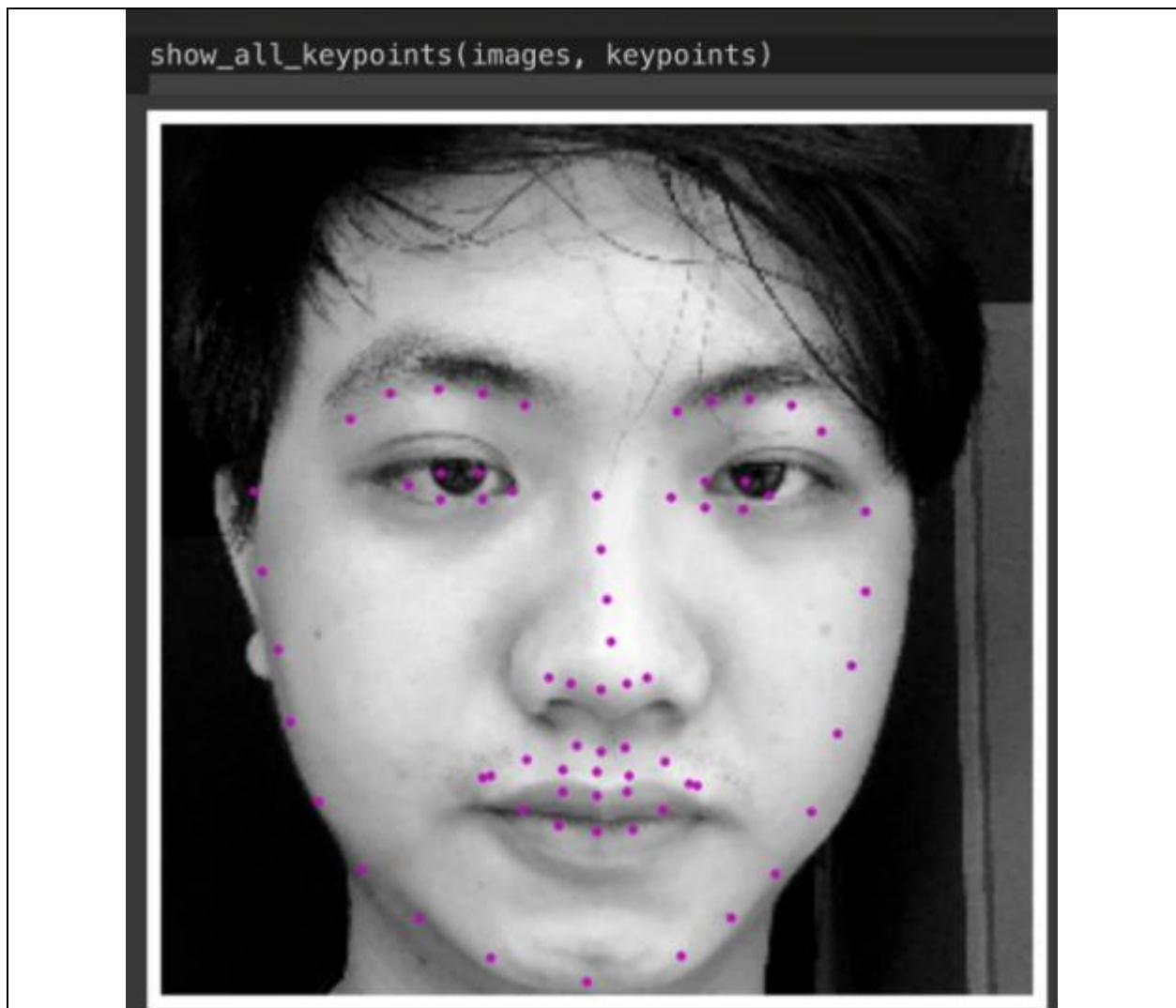
4.1.1. Mô hình chuẩn đoán

Mô hình CNN được xây dựng trong việc xác định các điểm mốc với độ chính xác 98%, cho ra kết quả chuẩn đoán chính xác các vị trí điểm mốc trên hình ảnh một khuôn mặt ngẫu nhiên

Sử dụng Colab để chạy dự đoán, chúng ta có một số kết quả sau:



Hình 4.1. Hình dự đoán 1



Hình 4.2. Hình dự đoán 2

4.1.2. WebApp

4.2. Nhược điểm

Mô hình tạo ra tuy đã nhận diện được khuôn mặt và tạo ra được các điểm mốc cho khuôn mặt khá chính xác nhưng vẫn chưa đạt được giá trị tiệm cận tuyệt đối như những mô hình huấn luyện chuẩn khác

Mô hình chưa kết hợp với chạy thời gian thực nên tính ứng dụng chưa cao.

4.3. Hướng phát triển

Trong tương lai, hướng phát triển trực tiếp sẽ là định hình và nghiên cứu thêm khả năng ứng dụng của việc xác định các điểm Facial Landmark. Tiến hành huấn luyện lại cho model và áp dụng các thuật toán tăng cường nhằm tối ưu tốt nhất có thể cho các dự án sau.

TÀI LIỆU THAM KHẢO

Tiếng Việt

- [1]. Mì AI, Xây dựng web app, triển khai model, Link truy cập: <https://miai.vn/2021/12/29/xay-dung-web-app-trien-khai-model-trong-1-phut-voi-streamlit-mi-ai/>
- [2]. Trần Văn Huy, Introduction to Streamlit, Link truy cập: <https://huytranvan2010.github.io/Introduction-to-streamlit/>
- [3]. TOPDev, Thuật toán CNN – Convolutional Neural Network, Link truy cập: <https://topdev.vn/blog/thuat-toan-cnn-convolutional-neural-network/>

Tiếng Anh

- [1]. Streamlit, A faster way to build and share data apps, Available: <https://streamlit.io/>
- [2]. DenseNet: Better CNN model than ResNet. Available: <http://www.programmersonsought.com/article/7780717554/>
- [3]. Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in Proceedings of 2010 IEEE international symposium on circuits and systems, 2010, pp. 253-256.
- [4]. ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks. Available: <https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/>
- [5]. H. Jung, B. Kim, I. Lee, J. Lee, and J. Kang, "Classification of lung nodules in CT scans using three-dimensional deep convolutional neural networks with a checkpoint ensemble method," *BMC medical imaging*, vol. 18, p. 48, 2018.
- [6]. S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, pp. 1345-1359, 2009.

PHỤ LỤC

1. Dataset: https://drive.google.com/drive/folders/1o7P8ScLxg9ztMuMaUcv4OuAA_zwkMgsZ?usp=sharing
2. Link Github: <https://github.com/nhanguyene/Facial-Landmark-Detection-Project>
3. Link Youtube: <https://www.youtube.com/channel/UC60sjnrmHd7dq5aOiweal1Q/featured>