MỤC LỤC

[DANH MỤC HÌNH ẢNH 3](#_Toc8889046)

[DANH MỤC TỪ VIẾT TẮT 4](#_Toc8889047)

[PHẦN MỞ ĐẦU 5](#_Toc8889048)

[1. Tính cấp thiết của đề tài 5](#_Toc8889049)

[2. Mục tiêu và giới hạn của đề tài 5](#_Toc8889050)

[3. Các kết quả chính của đồ án 6](#_Toc8889051)

[4. Bố cục đồ án 6](#_Toc8889052)

[CHƯƠNG 1: ĐẶT VẤN ĐỀ 7](#_Toc8889053)

[1.1. ĐỊNH NGHĨA BÀI TOÁN PHÂN LOẠI ẢNH NHẬN DẠNG TƯ THẾ NGƯỜI: ĐỨNG, NGỒI VÀ NẰM 7](#_Toc8889054)

[1.2. TẠI SAO CHỌN CNN ĐỂ GIẢI QUYẾT BÀI TOÁN PHÂN LOẠI ẢNH 7](#_Toc8889055)

[1.2.1. Vấn đề của mạng nơ-ron truyền thống và giải pháp cho bài toán xử lý ảnh 7](#_Toc8889056)

[1.2.2. So sánh CNN với các phương pháp phân loại ảnh khác 9](#_Toc8889057)

[CHƯƠNG 2: TỔNG QUAN LÝ THUYẾT 10](#_Toc8889058)

[2.1. MẠNG NƠ-RON NHÂN TẠO VÀ MẠNG NƠ-RON LAN TRUYỀN NGƯỢC 10](#_Toc8889059)

[2.1.1. Giới thiệu mạng Nơ-ron nhân tạo 10](#_Toc8889060)

[2.1.2. Một số kiểu mạng NN 11](#_Toc8889061)

[2.1.3. Mạng Nơ-ron lan truyền ngược MLP 13](#_Toc8889062)

[2.2. CONVOLUTIONAL NEURAL NETWORKS 21](#_Toc8889063)

[2.2.1. Kiến trúc mạng CNN 21](#_Toc8889064)

[2.2.2. Huấn luyện mạng CNN 28](#_Toc8889065)

[2.2.3. Một số thuật toán tối ưu khác 33](#_Toc8889066)

[CHƯƠNG 3: ỨNG DỤNG MẠNG CNN TRONG BÀI TOÁN PHÂN LOẠI ẢNH TƯ THẾ NGƯỜI 37](#_Toc8889067)

[3.1. KERAS 37](#_Toc8889068)

[3.1.1. Lý do chọn Keras để xử lý bài toán phân loại 37](#_Toc8889069)

[3.1.2. Cấu trúc của Keras 39](#_Toc8889070)

[3.1.3. Tổng quan về các Layer 39](#_Toc8889071)

[3.1.4. Các hàm thường sử dụng 43](#_Toc8889072)

[3.2. CÁC BƯỚC THỰC HIỆN MẠNG CNN TRONG BÀI TOÁN PHÂN LOẠI ẢNH TƯ THẾ NGƯỜI 44](#_Toc8889073)

[CHƯƠNG 4: KẾT QUẢ THỰC NGHIỆM 47](#_Toc8889074)

[4.1. CÀI ĐẶT KERAS, TENSORFLOW 47](#_Toc8889075)

[4.2. THỰC NGHIỆM 49](#_Toc8889076)

[KẾT LUẬN 63](#_Toc8889077)

[TÀI LIỆU THAM KHẢO 64](#_Toc8889078)

DANH MỤC HÌNH ẢNH

[Hình 2.1. Cấu tạo một Neural 10](#_Toc9118110)

[Hình 2.2. Mạng tự kết hợp 12](#_Toc9118111)

[Hình 2.3. Mạng kết hợp khác kiểu 12](#_Toc9118112)

[Hình 2.4. Mạng truyền thẳng 13](#_Toc9118113)

[Hình 2.5. Mạng phản hồi 13](#_Toc9118114)

[Hình 2.6. Mạng neural lan truyền ngược MLP 14](#_Toc9118115)

[Hình 2.7. Mô hình mạng nơ-ron lan truyền ngược 17](#_Toc9118116)

[Hình 2.8. Kiến trúc của CNN (Nguồn: www.mathworks.com) 22](#_Toc9118117)

[Hình 2.9 Kiến trúc của CNN (Nguồn: www.chegg.com) 23](#_Toc9118118)

[Hình 2.10. Cửa sổ trượt (Nguồn: read01.com) 24](#_Toc9118119)

[Hình 2.11. Đồ thị hàm ReLU (Nguồn: medium.com) 25](#_Toc9118120)

[Hình 2.12. Max pooling kernel 2x2, stride = 2 (Nguồn: Wikipedia) 26](#_Toc9118121)

[Hình 2.13. Cấu trúc của lớp fully-connected (a), (b) và data dependency (c) (Nguồn: www.researchgate.net) 28](#_Toc9118122)

[Hình 2.14. So sánh Gradient Descent với các hiện tượng vật lý (Nguồn: machinelearningcoban.com) 32](#_Toc9118123)

DANH MỤC TỪ VIẾT TẮT

|  |  |
| --- | --- |
| NN | Neural Networks – Mạng nơ-ron nhân tạo |
| CNN | Convolutional Neural Networks – Mạng nơ-ron tích chập |
| ReLU | Rectified Linear Unit – Tinh chỉnh đơn vị tuyến tính |
| GD | Gradient Descent |
| SGD | Stochastic Gradient Descent |
| FC | Fully-Connected – Kết nối đầy đủ |
| NAG | Nesterov accelerated gradient |

PHẦN MỞ ĐẦU

1. Tính cấp thiết của đề tài

Xã hội ngày càng phát triển, đời sống được nâng cao nhưng kéo theo đó là con người càng bận rộn, các tệ nạn xã hội cũng theo đó mà tăng lên với những mánh khóe, thủ đoạn vô cùng tinh vi. Tình hình an ninh ngày một trở nên phức tạp, trộm cắp hoành hành trở thành mối lo ngại cho xã hội, các tổ chức, cũng như mỗi gia đình và mỗi người dân. Do đó vấn đề giám sát an ninh ngày càng trở nên vô cùng quan trọng. Đi kèm với việc lắp đặt các thiết bị camera sử dụng cho mục đích giám sát thì cần phải có một hệ thống xử lý hình ảnh, video nhằm phân tích các đối tượng, các hành vi và sự kiện.

Cùng với sự phát triển mạnh mẽ về khả năng tính toán của các thế hệ máy tính hiện đại cũng như sự bùng nổ về dữ liệu thông qua mạng lưới Internet trải rộng, ta đã chứng kiến nhiều sự đột phá trong lĩnh vực Học máy, đặc biệt là lĩnh vực Thị giác máy tính. Sự quay lại và phát triển vượt bậc của các phương pháp Học sâu đã giúp Thị giác máy tính đạt được những thành tựu đáng kể trong nhận dạng ảnh. Việc xử lý hình ảnh trở nên đơn giản hơn rất nhiều, điều này đóng vai trò rất quan trọng trong hệ thống giám sát an ninh được sử dụng rất phổ biến ngày nay.

Đề tài “Ứng dụng mạng CNN trong bài toán phân loại ảnh – nhận dạng tư thế người” đã được đưa ra nhằm xây dựng một hệ thống phân loại ba tư thế: đứng, ngồi và nằm từ những bức ảnh người – một bước quan trọng trong hệ thống giám sát an ninh.

2. Mục tiêu và giới hạn của đề tài

Do những hạn chế về kiến thức và thời gian thực hiện đề tài cũng như giới hạn về trang thiết bị, Đồ án của em chỉ mới thực hiện được các công việc như sau:

* Tìm hiểu về mạng Nơ-ron tích chập.
* Ứng dụng mạng Nơ-ron tích chập, xây dựng chương trình phân loại ảnh 3 tư thế người: đứng, ngồi và nằm.
* Chương trình được viết bằng ngôn ngữ Python, sử dụng tập dữ liệu đầu vào ……

3. Các kết quả chính của đồ án

4. Bố cục đồ án

Đồ án được bố cục thành các phần như sau:

**Phần mở đầu**: Giới thiệu lý do chọn đề tài, mục tiêu, giới hạn và kết quả thực hiện được.

**Chương 1**: Định nghĩa bài toán và lựa chọn phương pháp giải quyết bài toán.

**Chương 2**: Cơ sở lý thuyết: Mạng Nơ-ron nhân tạo và mạng Nơ-ron tích chập.

**Chương 3**: Ứng dụng mạng CNN trong bài toán phân loại ảnh tư thế người.

**Chương 4**: Kết quả thực nghiệm, so sánh hiệu suất các mô hình.

**Kết luận**: Tổng kết các nội dung đã trình bày trong đồ án, đề xuất các phương hướng nghiên cứu tiếp theo để tiếp tục cải thiện chất lượng nhận dạng của hệ thống.

CHƯƠNG 1: ĐẶT VẤN ĐỀ

1.1. ĐỊNH NGHĨA BÀI TOÁN PHÂN LOẠI ẢNH NHẬN DẠNG TƯ THẾ NGƯỜI: ĐỨNG, NGỒI VÀ NẰM

Hiện nay, các kỹ thuật phân tích video thông minh (IVA) đang ngày càng được áp dụng rộng rãi trong đời sống thực tế, đặc biệt là trong lĩnh vực giám sát an ninh. Trong quá trình ghi hình, các thông tin liên quan đến đến người (vận tốc, vị trí, tọa độ khung của người, v.v..) và không phải người (đồ vật bị bỏ quên, ô tô, động vật, v. v..) sẽ được trích xuất và lưu giữ. Dựa vào các thông tin này, hệ thống phân tích video thông minh sẽ sử dụng các thông tin thích hợp cho từng bài toán cụ thể, chẳng hạn như: sử dụng thông tin liên quan đến người di chuyển để phân tích hành động khả nghi của đối tượng chuyển động trong khu vực bị giám sát hay là xác định đồ vật bị bỏ quên tại trung tâm công cộng dựa vào các thông tin liên quan đến đồ vật có trong khung hình video.

Trong các loại thông tin như đề cập ở trên, thông tin liên quan đến tư thế người, bao gồm đứng, ngồi hoặc nằm là những thông tin hữu ích. Các thông tin này có thể kết hợp với các thông tin liên quan đến vận tốc để phân loại ra các hành động của người. Từ đó, có thể sử dụng những phân tích này áp dụng cho việc giải quyết các bài toán nhận dạng hành vi, sự kiện và hành động. Để có thể xác định tư thế người chuyển động trong khung hình, một trong các cách hiệu quả hiện nay đó là ứng dụng mạng CNN để phân loại tư thế người.

1.2. TẠI SAO CHỌN CNN ĐỂ GIẢI QUYẾT BÀI TOÁN PHÂN LOẠI ẢNH

1.2.1. Vấn đề của mạng nơ-ron truyền thống và giải pháp cho bài toán xử lý ảnh

Nền tảng của mạng nơ-ron tích chập (Convolutional Neural Networks – CNN) là mạng nơ-ron truyền thống (được dùng với nhiều tên gọi như MLP – Multilayer Perceptron, Regular Neural Net, fully-connected nets) mà chúng ta sẽ đi vào tìm hiểu kỹ hơn ở phần tiếp theo, hoạt động bằng cách nhận input (là một vector) và biến đổi vector này thông qua các hidden layer. Mỗi hidden layer được tạo bởi một tập các nơ-ron (mỗi nơ-ron trong layer được kết nối với toàn bộ các nơ-ron của layer liền trước nó), và điểm đặc biệt là các nơ-ron thuộc cùng một layer hoạt động hoàn toàn biệt lập với nhau (các trọng số gắn với các nơ-ron này được sử dụng độc lập và hoàn toàn không liên quan đến nhau).

Tuy nhiên hiệu suất của mạng nơ-ron truyền thống lại không tốt đối với vấn đề xử lý ảnh. Chẳng hạn với một bức ảnh có kích thước nhỏ: 32x32x3 (rộng 32 pixel, cao 32 pixel, 3 kênh màu – Red-Green-Blue) thì số lượng feature của input sẽ là: 32x32x3 = 3072, do đó ở hidden layer đầu tiên, ngay sau input layer mỗi nơ-ron sẽ phải có lượng trọng số (weights) là 3072, vẫn kiểm soát được, tuy nhiên nếu với ảnh lớn hơn một chút chỉ vào khoảng 200x200x3 thì mỗi nơ-ron đã cần tới 120,000 weights. Chính vì vậy nên khi sử dụng mạng nơ-ron truyền thống trong xử lý ảnh, kích cỡ của ảnh lớn khiến cho số lượng tham số (trọng số) được sử dụng trong mạng tăng lên chóng mặt, khi số lượng trọng số lớn sẽ khiến chi phí tính toán lớn và dễ dẫn đến quá tải (overfitting).

Khi ta nhìn vào một bức ảnh, bản thân các chi tiết trong ảnh giúp ta định hình được nội dung trong bức ảnh, nhưng nếu ta nhận được từng chi tiết riêng lẻ, và phải hình dung toàn bộ nội dung bức ảnh thì công việc trở nên khó khăn hơn. Vấn đề các nơ-ron trên cùng 1 layer hoạt động độc lập với nhau hay chính là việc không sử dụng chung các trọng số khiến cho việc trích xuất những đặc trưng trên input cũng gặp khó khăn hơn và yêu cầu nhiều nơ-ron hơn, nhiều layer hơn dẫn tới chi phí tính toán tăng và dễ vướng vào quá tải. Vì vậy các mạng nơ-ron mới liên tiếp ra đời, cố gắng giải quyết vấn đề này. Cụ thể trong các mạng nơ-ron mới trọng số được dùng chung giữa các nơ-ron theo nhiều cách, giúp giảm số lượng trọng số, giảm chi phí tính toán, nhưng vẫn đem lại kết quả tốt.

Điển hình trong số đó là mạng neural tích chập (Convolutional Neural Networks – CNN). Cũng giống như mạng nơ-ron truyền thống, CNN hoạt động theo phương thức nhận input và biến đổi input thông qua các layer, tuy nhiên điểm khác biệt nằm ở cấu trúc của input và cấu trúc bên trong 1 layer.

1.2.2. So sánh CNN với các phương pháp phân loại ảnh khác

Kết quả từ một bài viết của tác giả Akshat Maheshwari về một thực nghiệm phân loại văn bản sử dụng *Mạng nơ-ron tích chập* (CNN), *Mạng nơ-ron hồi quy* (Recurrent Neural Network – RNN) và *Mạng chú ý phân cấp* (Hierarchical Attention Network – HAN), sử dụng 3 tập dữ liệu với các kích thước khác nhau, kết quả thu được cho ta những nhận xét như sau:

* CNN đã đạt được độ chính xác khá tốt với tính nhất quán cao, còn RNN và HAN đã được độ chính xác cao nhưng không nhất quán trong toàn bộ tập dữ liệu.
* RNN là kiến trúc tệ nhất để thực hiện các kịch bản có sẵn.
* CNN vượt trội hơn về thời gian training, tuy nhiên HAN có thể hoạt động tốt hơn CNN và RNN nếu ta có một tập dữ liệu lớn.
* HAN đã thực hiện rất tốt khi có một bộ training lớn, tuy nhiên HAN không làm được tốt như vậy với bộ dữ liệu training rất ít.
* Khi bộ dữ liệu training ít thì CNN đã đạt đc độ chính xác xác thực tốt nhất.

Trong một ví dụ khác, kết quả thực nghiệm trên 10 tập dữ liệu biểu hiện gen MEG từ kho Kent Ridge (Jinyan & Huiqing, 2002), GEO (Edgar et al., 2002) cho thấy rằng khả năng phân loại của mạng nơ-ron tích chập CNNs đạt được độ chính xác tương đương với mô hình *Máy véc-tơ hỗ trợ* (Support Vector Machines – SVM), *Rừng ngẫu nhiên*(Random Forest – RF) và tốt hơn so với *k láng giềng gần* (k Nearest Neighbors –kNN), *Cây quyết định*(Decision Trees – DT), Adaboost và Bagging.

Đối với bài toán phân loại văn bản, tập dữ liệu đầu vào không hẳn là ảnh thuần tuý. Tuy nghiên ở mỗi thuật toán chúng ta đều phải có bước tiền xử lý dữ liệu đầu vào về các dạng đặc trưng còn mô hình toán đào tạo thì không thay đổi. Do đó có thể tham khảo những ví dụ này để đánh giá mức độ chính xác của các thuật toán. Với một tập dữ liệu ảnh tư thế người không quá lớn, em lựa chọn CNN để xây dựng bài toán cho tốc độ xử lý nhanh nhưng vẫn đảm bảo kết quả có độ chính xác cao.

CHƯƠNG 2: TỔNG QUAN LÝ THUYẾT

2.1. MẠNG NƠ-RON NHÂN TẠO VÀ MẠNG NƠ-RON LAN TRUYỀN NGƯỢC

2.1.1. Giới thiệu mạng Nơ-ron nhân tạo

***Định nghĩa:*** Mạng nơron nhân tạo, Artificial Neural Network (ANN) hay Neural Network (NN) là một mô hình xử lý thông tin phỏng theo cách thức xử lý thông tin của các hệ nơ-ron sinh học. Nó được tạo nên từ một số lượng lớn các phần tử (nơ-ron) kết nối với nhau thông qua các liên kết (trọng số liên kết) làm việc như một thể thống nhất để giải quyết một vấn đề cụ thể nào đó. Một mạng nơ-ron nhân tạo được cấu hình cho một ứng dụng cụ thể (nhận dạng mẫu, phân loại dữ liệu,...) thông qua một quá trình học từ tập các mẫu huấn luyện. Về bản chất học chính là quá trình hiệu chỉnh trọng số liên kết giữa các nơ-ron.

Neural Network đang trở thành một công cụ rất mạnh mẽ mang lại hiệu quả tốt nhất cho nhiều bài toán khó như nhận dạng ảnh, giọng nói hay xử lý ngôn ngữ tự nhiên.

***Cấu trúc mạng neural nhân tạo:***



Hình 2.1. Cấu tạo một Neural

Các thành phần cơ bản của một nơron nhân tạo bao gồm:

* Tập các đầu vào: Là các tín hiệu vào (input signals) của nơron, các tín hiệu này thường được đưa vào dưới dạng một vector N chiều.
* Tập các liên kết: Mỗi liên kết được thể hiện bởi một trọng số liên kết Synaptic weight. Trọng số liên kết giữa tín hiệu vào thứ j với nơron k thường được kí hiệu là wkj. Thông thường, các trọng số này được khởi tạo một cách ngẫu nhiên ở thời điểm khởi tạo mạng và được cập nhật liên tục trong quá trình học mạng.
* Bộ tổng (Summing function): Thường dùng để tính tổng của tích các đầu vào với trọng số liên kết của nó.
* Ngưỡng (còn gọi là một độ lệch - bias): Ngưỡng này thường được đưa vào như một thành phần của hàm truyền.
* Hàm truyền (Transfer function): Hàm này được dùng để giới hạn phạm vi đầu ra của mỗi nơron. Nó nhận đầu vào là kết quả của hàm tổng và ngưỡng.
* Đầu ra: Là tín hiệu đầu ra của một nơron, với mỗi nơron sẽ có tối đa là một đầu ra.

Xét về mặt toán học, cấu trúc của một nơron k, được mô tả bằng cặp biểu thức sau:



và 

Trong đó: x1, x2, ..., xp: là các tín hiệu vào; (wk1, wk2,..., wkp) là các trọng số liên kết của nơron thứ k; uk là hàm tổng; bk là một ngưỡng; f là hàm truyền và yk là tín hiệu đầu ra của nơron.

Như vậy nơron nhân tạo nhận các tín hiệu đầu vào, xử lý (nhân các tín hiệu này với trọng số liên kết, tính tổng các tích thu được rồi gửi kết quả tới hàm truyền), và cho một tín hiệu đầu ra ( là kết quả của hàm truyền).

2.1.2. Một số kiểu mạng NN

Cách thức kết nối các nơron trong mạng xác định kiến trúc (topology) của mạng. Các nơron trong mạng có thể kết nối đầy đủ (fully connected) tức là mỗi nơron đều được kết nối với tất cả các nơron khác, hoặc kết nối cục bộ (partially connected) chẳng hạn chỉ kết nối giữa các nơron trong các tầng khác nhau. Người ta chia ra hai loại kiến trúc mạng chính:

* Tự kết hợp (autoassociative): là mạng có các nơron đầu vào cũng là các nơron đầu ra. Mạng Hopfield là một kiểu mạng tự kết hợp.



Hình 2.2. Mạng tự kết hợp

* Kết hợp khác kiểu (heteroassociative): là mạng có tập nơron đầu vào và đầu ra riêng biệt. Perceptron, các mạng Perceptron nhiều tầng (MLP: MultiLayer Perceptron), mạng Kohonen, … thuộc loại này.



Hình 2.3. Mạng kết hợp khác kiểu

Ngoài ra tùy thuộc vào mạng có các kết nối ngược (feedback connections) từ các nơron đầu ra tới các nơron đầu vào hay không, người ta chia ra làm 2 loại kiến trúc mạng.

* Kiến trúc truyền thẳng (feedforward architechture): là kiểu kiến trúc mạng không có các kết nối ngược trở lại từ các nơron đầu ra về các nơron đầu vào; mạng không lưu lại các giá trị output trước và các trạng thái kích hoạt của nơron. Các mạng nơron truyền thẳng cho phép tín hiệu di chuyển theo một đường duy nhất; từ đầu vào tới đầu ra, đầu ra của một tầng bất kì sẽ không ảnh hưởng tới tầng đó. Các mạng kiểu Perceptron là mạng truyền thẳng.



Hình 2.4. Mạng truyền thẳng

* Kiến trúc phản hồi (*Feedback architecture*): là kiểu kiến trúc mạng có các kết nối từ nơron đầu ra tới nơron đầu vào. Mạng lưu lại các trạng thái trước đó, và trạng thái tiếp theo không chỉ phụ thuộc vào các tín hiệu đầu vào mà còn phụ thuộc vào các trạng thái trước đó của mạng. Mạng Hopfield thuộc loại này.



Hình 2.5. Mạng phản hồi

2.1.3. Mạng Nơ-ron lan truyền ngược MLP

*a. Kiến trúc mạng MLP*

Mô hình mạng nơron được sử dụng rộng rãi nhất là mô hình mạng nhiều tầng truyền thẳng (MLP: Multi Layer Perceptron). Một mạng MLP tổng quát là mạng có n (n ≥ 2) tầng (thông thường tầng đầu vào không được tính đến): trong đó gồm một tầng đầu ra (tầng thứ n) và (n-1) tầng ẩn.



Hình 2.6. Mạng neural lan truyền ngược MLP

Kiến trúc của một mạng MLP tổng quát có thể mô tả như sau:

* Đầu vào là các vector (x1, x2, ..., xp) trong không gian p chiều, đầu ra là các vector (y1, y2, ..., yq) trong không gian q chiều. Đối với các bài toán phân loại, p chính là kích thước của mẫu đầu vào, q chính là số lớp cần phân loại. Xét ví dụ trong bài toán nhận dạng chữ số: với mỗi mẫu ta lưu tọa độ (x, y) của 8 điểm trên chữ số đó, và nhiệm vụ của mạng là phân loại các mẫu này vào một trong 10 lớp tương ứng với 10 chữ số 0, 1, …, 9. Khi đó p là kích thước mẫu và bằng 8 x 2 = 16; q là số lớp và bằng 10.
* Mỗi nơron thuộc tầng sau liên kết với tất cả các nơron thuộc tầng liền trước nó.
* Đầu ra của nơron tầng trước là đầu vào của nơron thuộc tầng liền sau nó.

Hoạt động của mạng MLP như sau: tại tầng đầu vào các nơron nhận tín hiệu vào xử lý (tính tổng trọng số, gửi tới hàm truyền) rồi cho ra kết quả (là kết quả của hàm truyền); kết quả này sẽ được truyền tới các nơron thuộc tầng ẩn thứ nhất; các nơron tại đây tiếp nhận như là tín hiệu đầu vào, xử lý và gửi kết quả đến tầng ẩn thứ 2;…; quá trình tiếp tục cho đến khi các nơron thuộc tầng ra cho kết quả.

Một số kết quả đã được chứng minh:

* Bất kì một hàm Boolean nào cũng có thể biểu diễn được bởi một mạng MLP 2 tầng trong đó các nơron sử dụng hàm truyền sigmoid.
* Tất cả các hàm liên tục đều có thể xấp xỉ bởi một mạng MLP 2 tầng sử dụng hàm truyền sigmoid cho các nơron tầng ẩn và hàm truyền tuyến tính cho các nơron tầng ra với sai số nhỏ tùy ý.
* Mọi hàm bất kỳ đều có thể hàm truyền sigmoid cho các nơron nơron tầng ra.

*b. Huấn luyện mạng MLP:*

***Khái niệm****:* Học là quá trình thay đổi hành vi của các vật theo một cáchnào đó làm cho chúng có thể thực hiện tốt hơn trong tương lai.

Một mạng nơron được huyấn luyện sao cho với một tập các vector đầu vào X, mạng có khả năng tạo ra tập các vector đầu ra mong muốn Y của nó. Tập X được sử dụng cho huấn luyện mạng được gọi là tập huấn luyện (*training set*). Các phần tử x thuộc X được gọi là các mẫu huấn luyện (*training example*). Quá trình huấn luyện bản chất là sự thay đổi các trọng số liên kết của mạng. Trong quá trình này, các trọng số của mạng sẽ hội tụ dần tới các giá trị sao cho với mỗi vector đầu vào x từ tập huấn luyện, mạng sẽ cho ra vector đầu ra y như mong muốn

Có ba phương pháp học phổ biến là học có giám sát (*supervised learning*), học không giám sát (*unsupervised learning*) và học tăng cường (*Reinforcement* *learning*):

* **Học có giám sát**: Là quá trình học có sựtham gia giám sát của một “thầygiáo”. Cũng giống như việc ta dạy một em nhỏ các chữ cái. Ta đưa ra một chữ “a” và bảo với em đó rằng đây là chữ “a”. Việc này được thực hiện trên tất cả các mẫu chữ cái. Sau đó khi kiểm tra ta sẽ đưa ra một chữ cái bất kì (có thể viết hơi khác đi) và hỏi em đó đây là chữ gì?

Như vậy với học có giám sát, số lớp cần phân loại đã được biết trước. Nhiệm vụ của thuật toán là phải xác định được một cách thức phân lớp sao cho với mỗi vector đầu vào sẽ được phân loại chính xác vào lớp của nó.

* **Học không giám sát**: Là việc học không cần có bất kỳmột sựgiám sát nào.

Trong bài toán học không giám sát, tập dữ liệu huấn luyện được cho dưới dạng: D = {(x1, x2, ..., xN)}, với (x1, x2, ..., xN) là vector đặc trưng của mẫu huấn luyện. Nhiệm vụ của thuật toán là phải phân chia tập dữ liệu D thành các nhóm con, mỗi nhóm chứa các vector đầu vào có đặc trưng giống nhau.

Như vậy với học không giám sát, số lớp phân loại chưa được biết trước, và tùy theo tiêu chuẩn đánh giá *độ tương tự* giữa các mẫu mà ta có thể có các lớp phân loại khác nhau.

* **Học tăng cường**: đôi khi còn được gọi là học thưởng-phạt (*reward-penalty learning*), là sựtổhợp của cả hai mô hình trên. Phương pháp này cụthểnhư sau: với vector đầu vào, quan sát vector đầu ra do mạng tính được. Nếu kết quả được xem là “tốt” thì mạng sẽ được thưởng theo nghĩa tăng các trọng số kết nối lên; ngược lại mạng sẽ bị phạt, các trọng số kết nối không thích hợp sẽ được giảm xuống. Do đó học tăng cường là học theo nhà phê bình (*critic*), ngược với học có giám sát là học theo thầy giáo (*teacher*).

*c. Học có giám sát trong các mạng nơron*

Học có giám sát có thể được xem như việc xấp xỉ một ánh xạ: X→ Y, trong đó X là tập các vấn đề và Y là tập các lời giải tương ứng cho vấn đề đó. Các mẫu (x, y) với x = (x1, x2, . . ., xn) ∈ X, y = (yl, y2, . . ., ym) ∈ Y được cho trước. Học có giám sát trong các mạng nơron thường được thực hiện theo các bước sau:

* **B1**: Xây dựng cấu trúc thích hợp cho mạng nơron, chẳng hạn có (n + 1)nơron vào (n nơron cho biến vào và 1 nơron cho ngưỡng x0), m nơron đầu ra, và khởi tạo các trọng số liên kết của mạng.
* **B2**: Đưa một vector x trong tập mẫu huấn luyện X vào mạng
* **B3**: Tính vector đầu ra o của mạng
* **B4**: So sánh vector đầu ra mong muốn y (là kết quả được cho trong tậphuấn luyện) với vector đầu ra o do mạng tạo ra; nếu có thể thì đánh giá lỗi.
* **B5**: Hiệu chỉnh các trọng sốliên kết theo một cách nào đó sao cho ởlầntiếp theo khi đưa vector x vào mạng, vector đầu ra o sẽ giống với y hơn.
* **B6**: Nếu cần, lặp lại các bước từ 2 đến 5 cho tới khi mạng đạt tới trạngthái hội tụ. Việc đánh giá lỗi có thể thực hiện theo nhiều cách, cách dùng nhiều nhất là sử dụng lỗi tức thời: Err = (**o** - **y**), hoặc Err = |**o** - **y**|; lỗi trung bình bình phương (MSE: mean-square error): Err = (**o**- **y**)2/2;

Có hai loại lỗi trong đánh giá một mạng nơron. Thứ nhất, gọi là lỗi rõ ràng (*apparent error*), đánh giá khả năng xấp xỉ các mẫu huấn luyện của một mạng đã được huấn luyện. Thứ hai, gọi là lỗi kiểm tra (*test error*), đánh giá khả năng tổng quá hóa của một mạng đã được huấn luyện, tức khả năng phản ứng với các vector đầu vào mới. Để đánh giá lỗi kiểm tra chúng ta phải biết đầu ra mong muốn cho các mẫu kiểm tra.

Thuật toán tổng quát ở trên cho học có giám sát trong các mạng nơron có nhiều cài đặt khác nhau, sự khác nhau chủ yếu là cách các trọng số liên kết được thay đổi trong suốt thời gian học. Trong đó tiêu biểu nhất là thuật toán lan truyền ngược.

*d. Mạng lan truyền ngược <Back Propagation Network>*

***Mô hình mạng:***

Mạng nơ-ron lan truyền ngược có mô hình như sau:



Hình 2.7. Mô hình mạng nơ-ron lan truyền ngược

Mạng có 3 lớp:

* Lớp vào (Input Layer) – số node vào là số thuộc tính của đối tượng cần phân lớp.
* Lớp ra (Output Layer) – Số node ra là số đặc điểm cần hướng tới của đối tượng (giá trị ra cần hướng đến – học có giám sát).
* Lớp ẩn (Hidden Layer) – Số node ẩn thường là không xác định trước, nó thường là do kinh nghiệm của người thiết kế mạng, nếu số node ẩn quá nhiều mạng sẽ cồng kềnh, quá trình học sẽ chậm, còn nếu số node ẩn quá ít làm mạng học không chính xác.

Các neural ở các lớp trong thường được kết nối đầy đủ với tất cả các neural lớp ngoài, trên mỗi đường kết nối giữa 2 neural ở 2 lớp khác nhau có 1 trọng số mạng (weight). Các trọng số này đóng vai trò là các giá trị ẩn số mà mạng cần phải tìm ra (học) sao cho với các giá trị đầu vào, thông qua mạng ta nhận được kết quả xấp xỉ với đầu ra mong muốn tương ứng của mẫu học.

***Hoạt động:***

Ta sử dụng một số kí hiệu sau:

* xi: Giá trị đầu vào của neural thứ i
* *yk*: Giá trị đầu ra của neural thứk
* *Vi*j: vector trọng số trên đường kết nối từneural node vào thứi tớineural node ẩn thứ j.
* *W*jk: vector trọng số trên đường kết nối từneural nodeẩn thứj tớineural node ra thứ k.
* dk: Giá trị đầu ra mong muốn của neural nút ra thứ k
* η: tốc độ học (Hệ số học) của mạng.
* f: hàm truyền với: f(x) = 1 / (1 + e-x) - Ở đây chúng ta sử dụng hàm truyền là hàm Sigmoid, giới hạn giá trị đầu ra trong khoảng [0-1].
* Ta sử dụng thêm 1 số ký hiệu sau:
* IIi – Input của node Input thứ i o OIi – Ouput của node Input thứ i
* IHi – Input của node Hidden thứ i o OHi – Output của node Hidden thứ i o IOi – Input của node Output thứ i o OOi – Output của node Output thứ i

Thuật toán lan truyền ngược được mô tả như sau:

**Input**:

* Mạng feed-forward với n đầu vào, m nút ẩn và L đầu ra.
* Hệ số học η
* Sai số học ԑ
* Tập dữ liệu huấn luyện D = {xi - là vector đầu vào, dk - là vector đầu ra mong muốn}.

**Output**: Các vector trọng số sau khi đã được huấn luyện.

***Thuật toán****:*

**Bước 1**: Khởi tạo trọng sốVij, Wjkbởi các giá trịngẫu nhiên nhỏ.

**Vij = Random(-1,1), Wjk = Random(-1,1)**

**Bước 2:** Lan truyền tiến tính toán đầu ra thực tếyk

* Tại node vào thứ i (Input):

**IIi = xi , OIi = IIi**

* Tại node ẩn thứ p (Hidden):

**IHp = ΣOIiVip , OHp = f(IHp) = 1 / (1 + e-IHp)**

* Tại node ra thứ q (Output):

**IOq = ΣOHiWiq , OOq = f(IOq) = 1 / (1 + e-IOq)**

Như vậy giá trị đầu ra thực tế của mạng với bộ trọng số ngẫu nhiên ban đầu là: yk = OOk. Thực tế ta có thể nhận thấy đầu ra mong muốn dk và đầu ra thực tế yk là luôn khác nhau, chúng ta cần phải tính toán độ sai khác này và có phương pháp điều chỉnh các trọng số mạng qua từng bước học sao cho qua mỗi lần học sự sai khác này giảm xuống, quá trình học sẽ dừng khi sự sai khác này đạt đến 1 ngưỡng giới hạn phù hợp nào đó.

**Bước 3:** Đánh giá lỗi học - lỗi trung bình bình phương (MSE: mean-square error):

**E = 1/L\*sqrt(Σ(dk - yk)2)**

Nếu E ≤ ԑ thì dừng học.

**Bước 4**: Lan truyền ngược điều chỉnh trọng số

* Với mỗi nút q thuộc tầng ra, tính đạo hàm ngược sai số thành phần δq theo công thức:

**δq = (dq – yq)yq(1 – yq)**

* Cập nhật các trọng số từ tầng ẩn tới tầng ra Wjk:

**Δwpq *= η*δqOHp**

**Wpq**(New) **= wpq**(Old) **+ Δwpq**

* Với mỗi nút p thuộc tầng ẩn, tính đạo hàm ngược sai số δp theo công thức:

**δp = OHp(1-OHp)Σ(wpk(old).δk), k=1..L**

* Cập nhật các trọng số từ tầng vào tới tầng ẩn Vij:

**Δvip *= η*δpOIi**

**vip**(New) **= vip**(Old) **+ Δvip**

Lặp lại bước 2 cho tới khi thỏa mãn điều kiện kết thúc ở bước 3. Kết quả thu được bộ trọng số chuẩn Vij, Wjk sao cho giá trị đầu ra thực tế và giá trị đầu ra mong muốn là gần giống nhau nhất (Trung bình bình phương lỗi nhỏ nhất)

*e. Một số vấn đề lưu ý trong xây dựng mạng MLP.*

***Xác định kích thước mẫu:***

Không có nguyên tắc nào hướng dẫn kích thước mẫu phải là bao nhiêu đối với một bài toán cho trước. Hai yếu tố quan trọng ảnh hưởng đến kích thước mẫu:

* Dạng hàm đích: khi hàm đích càng phức tạp thì kích thước mẫu cần tăng.
* Nhiễu: khi dữ liệu bị nhiễu (thông tin sai hoặc thiếu thông tin) kích thước mẫu cần tăng.

Đối với mạng truyền thẳng, cho hàm đích có độ phức tạp nhất định, kèm một lượng nhiễu nhất định thì độ chính xác của mô hình luôn có một giới hạn nhất định. Nói cách khác độ chính xác của mô hình là hàm theo kích thước tập mẫu.



*Hình 2.7. Ảnh hưởng của kích thước mẫu*

***Xác định số nơron tầng ẩn:***

Câu hỏi chọn số lượng noron trong tầng ẩn của một mạng MLP thế nào là khó, nó phụ thuộc vào bài toán cụ thể và vào kinh nghiệm của nhà thiết kế mạng. Có nhiều đề nghị cho việc chọn số lượng nơron tầng ẩn h trong một mạng MLP. Chẳng hạn h phải thỏa mãn h>(p-1)/(n+2), trong đó p là sốlượng mẫu huấn luyện và n là số lượng đầu vào của mạng. Càng nhiều nút ẩn trong mạng, thì càng nhiều đặc tính của dữ liệu huấn luyện sẽ được mạng nắm bắt, nhưng thời gian học sẽ càng tăng.

***Vấn đề quá khớp:***

Khái niệm quá khớp: Vấn đề quá khớp xảy ra khi mạng được luyện quá khớp (quá sát) với dữ liệu huấn luyện (kể cả nhiễu), nên nó sẽ trả lời chính xác những gì đã được học, còn những gì không được học thì nó không quan tâm. Như vậy mạng sẽ không có được khả năng tổng quát hóa. Vấn đề quá khớp xảy ra vì mạng có năng lực quá lớn. Có 3 cách để hạn chế bớt năng lực của mạng:

* Hạn chế số nút ẩn
* Ngăn không cho mạng sử dụng các trọng số lớn
* Giới hạn số bước luyện

Khi mạng được luyện, nó chuyển từ các hàm ánh xạ tương đối đơn giản đến các hàm ánh xạ tương đối phức tạp. Nó sẽ đạt được một cấu hình tổng quát hóa tốt nhất tại một điểm nào đó. Sau điểm đó mạng sẽ học để mô hình hóa nhiễu, những gì mạng học được sẽ trở thành quá khớp. Nếu ta phát hiện ra thời điểm mạng đạt đến trạng thái tốt nhất này, ta có thể ngừng tiến trình luyện trước khi hiện tượng quá khớp xảy ra. Ta biết rằng, chỉ có thể để đánh giá mức độ tổng quát hóa của mạng bằng cách kiểm tra mạng trên các mẫu nó không được học. Ta thực hiện như sau: chia mẫu thành tập mẫu huấn luyện và tập mẫu kiểm tra. Luyện mạng với tập mẫu huấn luyện nhưng định kỳ dừng lại và đánh giá sai số trên tập mẫu kiểm tra. Khi sai số trên tập mẫu kiểm tra tăng lên thì quá khớp đã bắt đầu và ta dừng tiến trình luyện.

Chú ý rằng, nếu sai số kiểm tra không hề tăng lên, tức là mạng không có đủ số nút ẩn để quá khớp. Khi đó mạng sẽ không có đủ số nút cần thiết để thực hiện tốt nhất. Do vậy nếu hiện tượng quá khớp không hề xảy ra thì ta cần bắt đầu lại nhưng sử dụng nhiều nút ẩn hơn.

2.2. CONVOLUTIONAL NEURAL NETWORKS

2.2.1. Kiến trúc mạng CNN

Các lớp cơ bản trong một mạng CNN bao gồm: Lớp tích chập (Convolutional), Lớp kích hoạt phi tuyến ReLU (Rectified Linear Unit), Lớp lấy mẫu (Pooling) và Lớp kết nối đầy đủ (Fully-connected), được thay đổi về số lượng và cách sắp xếp để tạo ra các mô hình huấn luyện phù hợp cho từng bài toán khác nhau.



Hình 2.8. Kiến trúc của CNN (Nguồn: [www.mathworks.com](https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html))

2.2.1.1. Convolutional layer

Đây là thành phần quan trọng nhất trong mạng CNN, cũng là nơi thể hiện tư tưởng xây dựng sự liên kết cục bộ thay vì kết nối toàn bộ các điểm ảnh. Các liên kết cục bộ này được tính toán bằng phép tích chập giữa các giá trị điểm ảnh trong một vùng ảnh cục bộ với các bộ lọc (filters) có kích thước nhỏ.

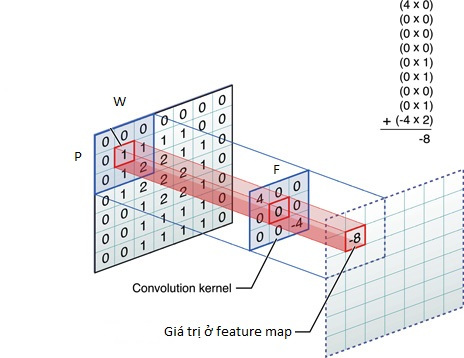
Với input là những bức ảnh, thông tin ta cần xử lý là rời rạc nên giờ ta sẽ xem với hàm số rời rạc tích chập được sử dụng như thế nào. Khi đó ảnh và filter chính là 2 hàm số rời rạc. Số chiều của convolutional chính là số hướng mà hàm filter có thể di chuyển được. Cụ thể như sau:

* Convolutional 1D: Tích chập 1 chiều sẽ chỉ cho phép hàm filter di chuyển theo 1 chiều, (nghĩa là ảnh của ta phải chuyển đổi về dạng vector) chính là độ dài của vector.
* Convolutional 2D: Tích chập 2 chiều sẽ cho phép hàm filter di chuyển theo 2 chiều (theo cả width và height).
* Convolutional 3D: Tương tự ta cũng có tích chập 3 chiều là khi hàm filter có thể di chuyển thêm cả chiều sâu của ảnh nữa.

Ngoài việc lựa chọn filter của ta slide được theo những chiều nào, ta còn có thể kiểm soát được tốc độ trượt của filter. Vì hàm số giờ đây không còn là liên tục nên tốc độ trượt của filter là có thể điều chỉnh.

* Input: Đầu vào của CNN là một bức ảnh được biểu diển bởi ma trận pixel với kích thước: [W x H x D].
* W: chiều rộng.
* H: chiều cao.
* D: Là độ sâu, hay dễ hiểu là số lớp màu của ảnh. Ví dụ ảnh RBG sẽ là 3 lớp ảnh Red, Green, Blue.

Chúng ta có một ma trận là Kernel, khi chiếu nó lên 1 vùng của bức ảnh. Lấy từng phần tử tương ứng của ảnh, nhân nó với phần tử tương ứng của Kernel, sau đó lấy tổng. Mục tiêu của các lớp tích chập là trích chọn các đặc trưng của ảnh đầu vào.



Hình 2.9 Kiến trúc của CNN (Nguồn: [www.chegg.com](http://www.chegg.com))

Dưới đây là một số các khái niệm cơ bản của phần này:

*Filter*, *Kernel* hay *Feature Detector* đều là cách gọi của ma trận tích chập *filter matrix* , ma trận này là 1 ma trận hệ số, và quét toàn bộ bức ảnh, mỗi lần quét thì tính tích chập trên vùng bị quét. Thông thường, ở các lớp đầu tiên của ma trận này sẽ có kích thước là [5x5x3]

*Convolved Feature, Activation Map* hay *Feature Map*là đầu ra của ảnh khi cho bộ lọc chạy hết bức ảnh với phép tích chập.

*Receptive field* là vùng ảnh được chọn để tính tích chập, hay bằng đúng cái kích thước của bộ lọc.

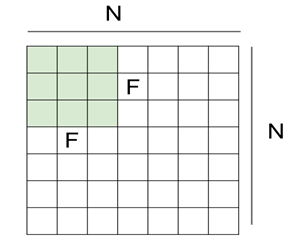
*Depth* là số lượng bộ lọc. Lưu ý: ở đây là số lượng bộ lọc (filter) chứ không phải số lượng kênh màu RBG như ở trên.

*Stride* được hiểu là khoảng cách dịch chuyển của bộ lọc sau mỗi lần tính. Ví dụ khi *stride* = 2. Tức sau khi tính xong tại 1 vùng ảnh, nó sẽ dịch sang phải 2 pixel. Tương tự với việc dịch xuống dưới

*Zero-Padding* là việc thêm các giá trị 0 ở xung quanh biên ảnh, để đảm bảo phép tích chập được thực hiện đủ trên toàn ảnh. Nếu không sẽ xảy ra hiện tượng mất mát dữ liệu, kích thước sẽ giảm nhanh khi qua các layer => cho thông tin không còn chính xác nữa.

* Công thức để tính kích thước cửa sổ trượt (Filter):

Output size:



Hình 2.10. Cửa sổ trượt (Nguồn: read01.com)

VD: N = 7, F =3, P =0:

stride = 1 => (7-3)/1 + 1 = 5

stride = 2 => (7-3)/2 + 1 = 3

stride = 3 => (7-3)/3 + 1 = 2.33 //don’t fit

* Công thức tính kích thước đầu ra của ảnh ứng với mỗi layer:

Giả sử ảnh đầu ra là [W2 x H2 x D2]

Thì:

* W2 = (W1 – F + 2P)/S + 1
* H2 = (H1 – F +2P)/S + 1
* D2 = K

Trong đó:

* [W1 x H1 x D1]: Kích thước ảnh đầu vào
* F: Kích thước của bộ lọc Kernel (NxN)
* S: giá trị Stride
* P: số lượng zero-padding thêm vào viền ảnh
* K: Số lượng bộ lọc (Depth)

2.2.1.2. ReLU Layer

Lớp này được xây dựng với ý nghĩa đảm bảo tính phi tuyến của mô hình huấn luyện sau khi đã thực hiện một loạt các phép tính toán tuyến tính qua các lớp Tích chập. Lớp Kích hoạt phi tuyến nói chung sử dụng các hàm kích hoạt phi tuyến như ReLU hoặc sigmoid, tanh… để giới hạn phạm vi biên độ cho phép của giá trị đầu ra. Trong số các hàm kích hoạt này, hàm ReLU được chọn do cài đặt đơn giản, tốc độ xử lý nhanh mà vẫn đảm bảo được tính toán hiệu quả. Cụ thể, phép tính toán của hàm ReLU chỉ đơn giản là chuyển tất cả các giá trị âm thành giá trị 0.

𝑓(𝑥) = 𝑚𝑎𝑥(0, 𝑥)

Thông thường, lớp ReLU được áp dụng ngay phía sau lớp Tích chập, với đầu ra là một ảnh mới có kích thước giống với ảnh đầu vào, các giá trị điểm ảnh cũng hoàn toàn tương tự trừ các giá trị âm đã bị loại bỏ.



Hình 2.11. Đồ thị hàm ReLU (Nguồn: medium.com)

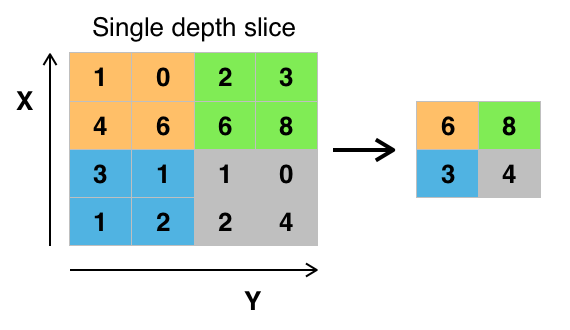
2.2.1.3. Pooling Layer

Một thành phần tính toán chính khác trong mạng CNN là lấy mẫu (Pooling), thường được đặt sau lớp Tích chập và lớp ReLU để làm giảm kích thước kích thước ảnh đầu ra trong khi vẫn giữ được các thông tin quan trọng của ảnh đầu vào. Việc giảm kích thước dữ liệu có tác dụng làm giảm được số lượng tham số cũng như tăng hiệu quả tính toán. Lớp lấy mẫu cũng sử dụng một cửa sổ trượt để quét toàn bộ các vùng trong ảnh tương tự như lớp Tích chập, và thực hiện phép lấy mẫu thay vì phép tích chập – tức là ta sẽ chọn lưu lại một giá trị duy nhất đại diện cho toàn bộ thông tin của vùng ảnh đó.

Điều này rất hữu ích khi ta sử dụng mạng cho ảnh có kích cỡ lớn (dimentional reduction). Tuy nhiên nếu lạm dụng loại layer này cũng có thể khiến dữ liệu đi qua bị mất mát thông tin.

Thông thường, Pooling layer có nhiều hình thức khác nhau phù hợp cho nhiều bài toán, tuy nhiên Max Pooling là được sử dụng nhiều và phổ biến hơn cả, với ý tưởng cũng rất sát với thực tế: “Giữ lại chi tiết quan trọng” hay hiểu ở trong bài toán này chính giữ lại pixel có giá trị lớn nhất.

Ví dụ: Max pooling với bộ lọc 2x2 và stride = 2. Bộ lọc sẽ chạy dọc ảnh. Và với mỗi vùng ảnh được chọn, sẽ chọn ra 1 giá trị lớn nhất và giữ lại.



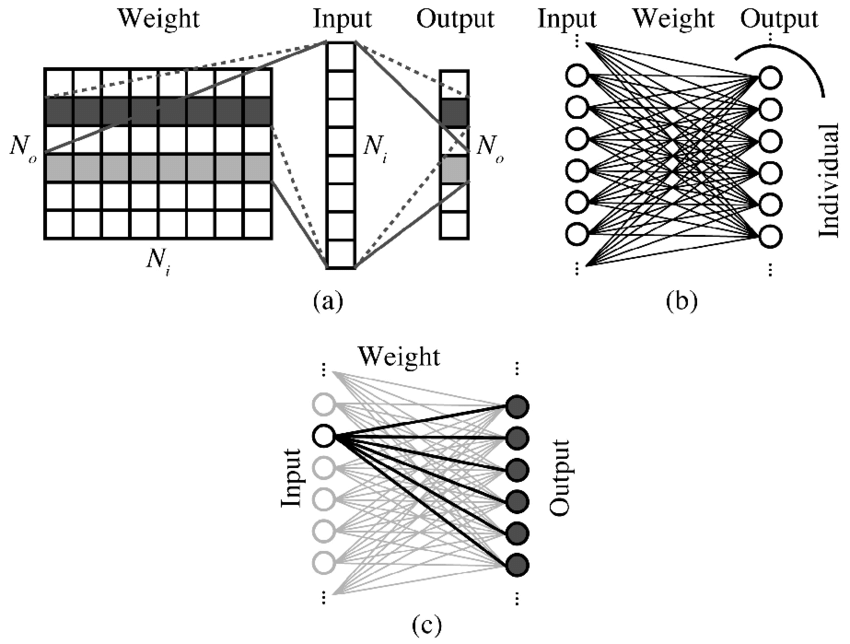
Hình 2.12. Max *pooling kernel 2x2, stride = 2* (Nguồn: Wikipedia)

Thông thường max pooling có kích thước là 2x2 và stride=2. Nếu lấy giá trị quá lớn, thay vì giúp đỡ thì nó lại làm phá vỡ cấu trúc ảnh và mất mát thông tin nghiêm trọng.

2.2.1.4. Fully-Connected Layer

Cũng như khi sử dụng mạng NN truyền thống để xử lý những dữ liệu có dạng ma trận như hình ảnh. Ta cần làm phẳng dữ liệu về dạng vector, sau đó đưa vào NN như bình thường. Hay nói cách khác phần Fully-Connected Layer (FC Layer) chính là một mạng NN được gắn vào phần cuối của CNN. Phần FC-Layer này chính là nơi từ các feature được trích xuất bởi phần tích chập và lấy mẫu tạo ra kết quả cuối cùng (Phân loại (Classification) hoặc Hồi quy (Regression)).

Lớp kết nối đầy đủ này được thiết kế hoàn toàn tương tự như trong mạng nơ-ron truyền thống, tức là tất cả các điểm ảnh được kết nối đầy đủ với các nút trong lớp tiếp theo. So với mạng nơ-ron truyền thống, các ảnh đầu vào của lớp này đã có kích thước được giảm bớt rất nhiều, đồng thời vẫn đảm bảo các thông tin quan trọng cho việc nhận dạng. Do vậy, việc tính toán nhận dạng sử dụng mô hình truyền thẳng đã không còn phức tạp và tốn nhiều thời gian như trong mạng nơ ron truyền thống.



Hình 2.13. Cấu trúc của lớp fully-connected (a), (b) và data dependency (c) (Nguồn: www.researchgate.net)

2.2.2. Huấn luyện mạng CNN

2.2.2.1. Thuật toán Stochastic Gradient Descent (SGD)

a. Gradient Descent cho hàm nhiều biến.

Trước khi đi vào thuật toán chính, chúng ta sẽ tìm hiểu xem gradient descent là gì, và các tính gradient descent cho hàm nhiều biến.

***Gradient Descent:***

Trong Machine Learning nói riêng và Toán Tối Ưu nói chung, chúng ta thường xuyên phải tìm giá trị nhỏ nhất (hoặc đôi khi là lớn nhất) của một hàm số nào đó. Ví dụ như các hàm mất mát trong hai bài [Linear Regression](https://machinelearningcoban.com/2016/12/28/linearregression/) (Hồi quy tuyến tính) và [K-means Clustering](https://machinelearningcoban.com/2017/01/01/kmeans/). Nhìn chung, việc tìm global minimum của các hàm mất mát trong Machine Learning là rất phức tạp, thậm chí là bất khả thi. Thay vào đó, người ta thường cố gắng tìm các điểm local minimum, và ở một mức độ nào đó, coi đó là nghiệm cần tìm của bài toán.

Các điểm local minimum là nghiệm của phương trình đạo hàm bằng 0. Nếu bằng một cách nào đó có thể tìm được toàn bộ (hữu hạn) các điểm cực tiểu, ta chỉ cần thay từng điểm local minimum đó vào hàm số rồi tìm điểm làm cho hàm có giá trị nhỏ nhất. Tuy nhiên, trong hầu hết các trường hợp, việc giải phương trình đạo hàm bằng 0 là bất khả thi. Nguyên nhân có thể đến từ sự phức tạp của dạng của đạo hàm, từ việc các điểm dữ liệu có số chiều lớn, hoặc từ việc có quá nhiều điểm dữ liệu.

Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà chúng ta coi là gần với nghiệm của bài toán, sau đó dùng một phép toán lặp để tiến dần đến điểm cần tìm, tức đến khi đạo hàm gần với 0. Gradient Descent (GD) và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất.

***Gradient Descent cho hàm 1 biến:***

Đầu tiên chúng ta có 2 nhận xét sau:

* Nếu đạo hàm của hàm số tại : thì nằm phía bên phải so với (và ngược lại). Để điểm tiếp theo gần với hơn, chúng ta cần di chuyển về phía bên trái, tức về phía âm. Nói cách khác, chúng ta cần di chuyển ngược dấu với đạo hàm:

Trong đó là một đại lượng ngược dấu với đạo hàm

* càng xa về phía bên phải thì càng lớn hơn 0 (và ngược lại). Vậy lượng di chuyển , một cách trực quan nhất, là tỷ lệ thuận với .

Hai nhận xét trên cho chúng ta một cách cập nhật đơn giản là:

Trong đó  là một số dương được gọi là learning rate (tốc độ học). Dấu trừ thể hiện việc chúng ta đi ngược với đạo hàm (đây cũng chính là lý do mà phương pháp này được gọi là Gradient Descent – descent nghĩa là đi ngược).

Việc lựa chọn learning rate rất quan trọng trong các bài toán thực tế. Việc lựa chọn giá trị này phụ thuộc nhiều vào từng bài toán và phải làm một vài thí nghiệm để chọn ra giá trị tốt nhất. Ngoài ra, tùy vào một số bài toán, GD có thể làm việc hiệu quả hơn bằng cách chọn ra learning rate phù hợp hoặc chọn learning rate khác nhau ở mỗi vòng lặp.

***Gradient Descent cho hàm nhiều biến***

Giả sử ta cần tìm global minimum cho hàm  trong đó  (theta) là một vector, thường được dùng để ký hiệu tập hợp các tham số của một mô hình cần tối ưu (trong Linear Regression thì các tham số chính là hệ số ). Đạo hàm của hàm số đó tại một điểm  bất kỳ được ký hiệu là  (hình tam giác ngược đọc là nabla). Tương tự như hàm 1 biến, thuật toán GD cho hàm nhiều biến cũng bắt đầu bằng một điểm dự đoán , sau đó, ở vòng lặp thứ t, quy tắc cập nhật là:



Hoặc viết dưới dạng đơn giản hơn: 

*Quy tắc cần nhớ:* Luôn luôn đi ngược hướng với đạo hàm.

*Kiểm tra đạo hàm:*

Việc tính đạo hàm của hàm nhiều biến thường khá phức tạp và dễ mắc lỗi, nếu chúng ta tính sai đọa hàm thì thuật toán GD không thể chạy đúng được. Trong thực nghiệm có một cách để chúng ta kiểm tra xem đạo hàm tính được có đúng không. Cách này dựa trên định nghĩa của đạo hàm (cho hàm một biến):



Một cách thường được sử dụng là lấy một giá trị  rất nhỏ, ví dụ , và sử dụng công thức:



Cách tính này được gọi là numerical gradient.

b. Thuật toán Stochastic Gradient Descent.

Trong thuật toán này, tại 1 thời điểm, ta chỉ tính đạo hàm của hàm mất mát dựa trên chỉ một điểm dữ liệu  rồi cập nhật  dựa trên đạo hàm này. Việc này được thực hiện với từng điểm trên toàn bộ dữ liệu, sau đó lặp lại quá trình trên. Thuật toán rất đơn giản này trên thực tế lại làm việc rất hiệu quả.

Mỗi lần duyệt một lượt qua tất cả các điểm trên toàn bộ dữ liệu được gọi là một epoch. Với GD thông thường thì mỗi epoch ứng với 1 lần cập nhật , với SGD thì mỗi epoch ứng với N lần cập nhật  với N là số điểm dữ liệu. Nhìn vào một mặt, việc cập nhật từng điểm một như thế này có thể làm giảm đi tốc độ thực hiện 1 epoch. Nhưng nhìn vào một mặt khác, SGD chỉ yêu cầu một lượng epoch rất nhỏ (thường là 10 cho lần đầu tiên, sau đó khi có dữ liệu mới thì chỉ cần chạy dưới một epoch là đã có nghiệm tốt). Vì vậy SGD phù hợp với các bài toán có lượng cơ sở dữ liệu lớn và các bài toán yêu cầu mô hình thay đổi liên tục, tức online learning.

***Thứ tự lựa chọn điểm dữ liệu***

Một điểm cần lưu ý đó là: sau mỗi epoch, chúng ta cần shuffle (xáo trộn) thứ tự của các dữ liệu để đảm bảo tính ngẫu nhiên. Việc này cũng ảnh hưởng tới hiệu năng của SGD.

Một cách toán học, quy tắc cập nhật của SGD là:



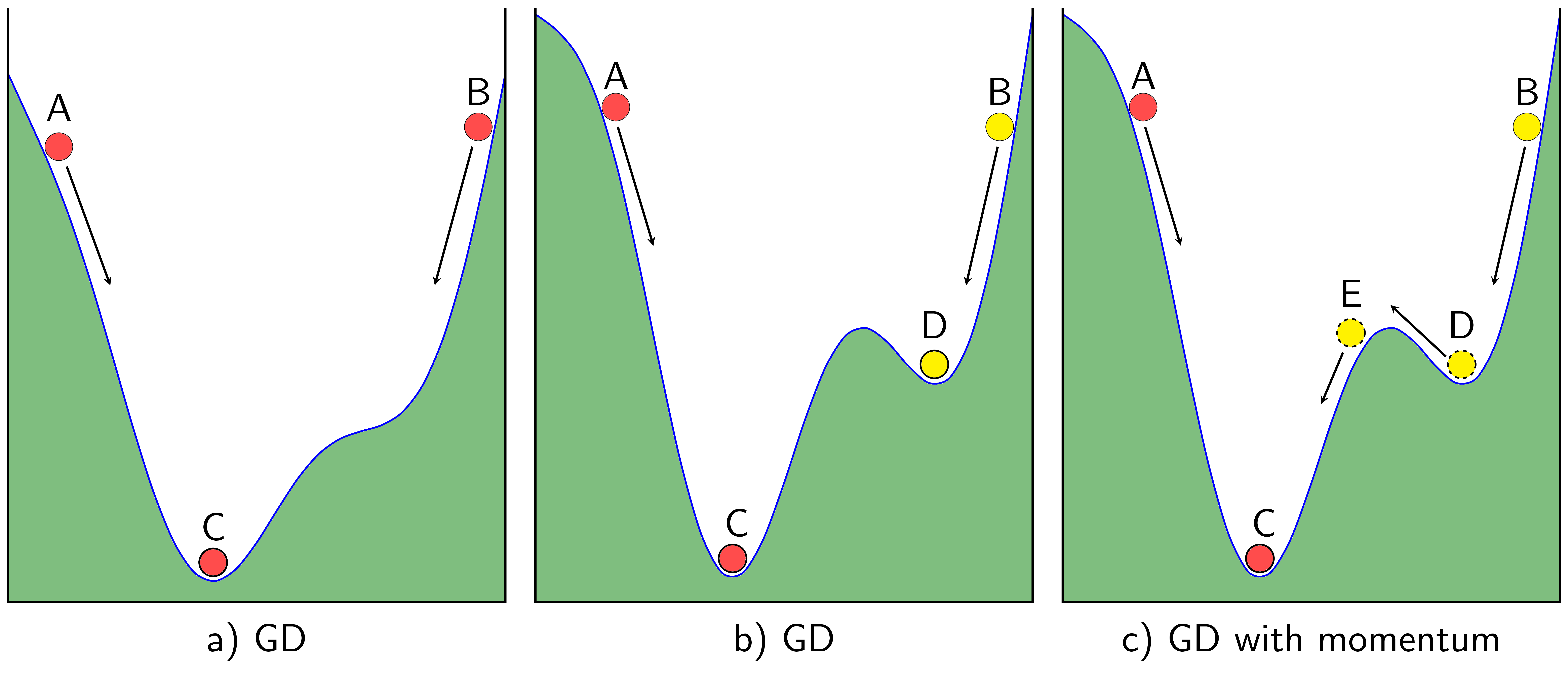
Trong đó là hàm mất mát với chỉ 1 cặp điểm dữ liệu (input, label) là . Chú ý: chúng ta hoàn toàn có thể áp dụng các thuật toán tăng tốc GD như Momentum, AdaGrad,… vào SGD.

2.2.2.2. Thuật toán Stochastic Gradient Descent with Momentum

a. Momentum

***Gradient dưới góc nhìn vật lý***

Thuật toán GD thường được ví với tác dụng của trọng lực lên một hòn bi đặt trên một mặt có dạng như hình một thung lũng giống như hình 13a dưới đây. Bất kể ta đặt hòn bi ở A hay B thì cuối cùng hòn bi cũng sẽ lăn xuống và kết thúc ở vị trí C.



Hình 2.14. So sánh Gradient Descent với các hiện tượng vật lý (Nguồn: machinelearningcoban.com)

Tuy nhiên, nếu như bề mặt có hai đáy thung lũng như Hình 13b thì tùy vào việc đặt bi ở A hay B, vị trí cuối cùng của bi sẽ ở C hoặc D. Điểm D là một điểm local minimum chúng ta không mong muốn.

Nếu suy nghĩ một cách vật lý hơn, vẫn trong Hình 13b, nếu vận tốc ban đầu của bi khi ở điểm B đủ lớn, khi bi lăn đến điểm D, theo đà, bi có thể tiếp tục di chuyển lên dốc phía bên trái của D. Và nếu giả sử vận tốc ban đầu lớn hơn nữa, bi có thể vượt dốc tới điểm E rồi lăn xuống C như trong Hình 13c. Đây chính là điều chúng ta mong muốn. Bạn đọc có thể đặt câu hỏi rằng liệu bi lăn từ A tới C có theo đà lăn tới E rồi tới D không. Xin trả lời rằng điều này khó xảy ra hơn vì nếu so với dốc DE thì dốc CE cao hơn nhiều.

Dựa trên hiện tượng này, một thuật toán được ra đời nhằm khắc phục việc nghiệm của GD rơi vào một điểm local minimum không mong muốn. Thuật toán đó có tên là Momentum (tức theo đà trong tiếng Việt).

***Gradient Descent với Momentum***

*Để biểu diễn momentum bằng toán học thì chúng ta phải làm thế nào?*

Trong GD, chúng ta cần tính lượng thay đổi ở thời điểm tt để cập nhật vị trí mới cho nghiệm (tức hòn bi). Nếu chúng ta coi đại lượng này như vận tốc  trong vật lý, vị trí mới của hòn bi sẽ là . Dấu trừ thể hiện việc phải di chuyển ngược với đạo hàm. Công việc của chúng ta bây giờ là tính đại lượng  sao cho nó vừa mang thông tin của độ dốc (tức đạo hàm), vừa mang thông tin của đà, tức vận tốc trước đó  (chúng ta coi như vận tốc ban đầu ). Một cách đơn giản nhất, ta có thể cộng (có trọng số) hai đại lượng này lại:



Trong đó γ thường được chọn là một giá trị khoảng 0.9, là vận tốc tại thời điểm trước đó,  chính là độ dốc của điểm trước đó. Sau đó vị trí mới của hòn bi được xác định như sau:



Thuật toán đơn giản này tỏ ra rất hiệu quả trong các bài toán thực tế (trong không gian nhiều chiều, cách tính toán cũng hoàn tòan tương tự).

b. Thuật toán Stochastic Gradient Descent with Momentum

Một cách tự nhiên, áp dụng Momentum vào thuật toán Stochastic Gradient Descent. Ta có:



Mỗi lần sẽ cập nhật  với:

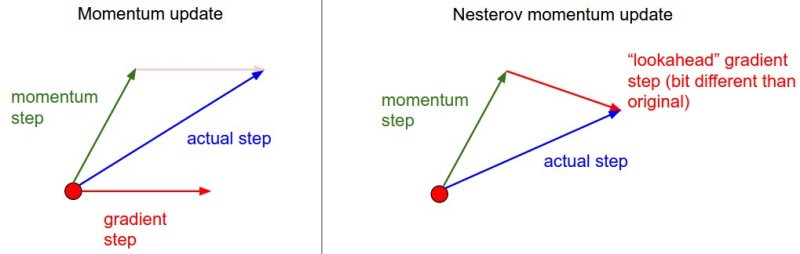


2.2.3. Một số thuật toán tối ưu khác

***Nesterov accelerated gradient***

Nesterov accelerated gradient (Nesterov, 1983, 2004) là một biến thể nhỏ của kỹ  
thuật momentum cho phép gradients thoát các điểm local minimum, khiến thuật toán hội tụ nhanh hơn.

Ý tưởng cơ bản là dự đoán hướng đi trong tương lai, tức nhìn trước một bước. Cách thức này có 1 ít khác biệt so với momentum update, với momentum update ta tính toán đạo hàm tại vị trí hiện hành rồi sau đó làm 1 cú nhảy tới dựa trên vector momentum trước đó, với Nesterov momentum thay vì tính toán đạo hàm tại điểm hiện hành, chúng ta dựa vào vector momentum cũ để tính toán vị trí sắp tới, rồi sau đó mới dùng đạo hàm tại vị trí mới để correct lại. Theo dõi hình dưới đây:



Ý tưởng của Nesterov accelerated gradient. (Nguồn: [CS231n Stanford: Convolutional Neural Networks for Visual Recognition](https://cs231n.github.io/neural-networks-3/))

* Với momentum thông thường: lượng thay đổi là tổng của hai vector: momentum vector và gradient ở thời điểm hiện tại.
* Với Nesterove momentum: lượng thay đổi là tổng của hai vector: momentum vector và gradient ở thời điểm được xấp xỉ là điểm tiếp theo.

Công thức cập nhật của NAG được cho như sau:

***Adagrad***

Adagrad (Duchi, 2011) cho phép tinh chỉnh các gradients khác nhau bằng  
việc có learning rate riêng biệt *η*. Cho mỗi giá trị của *f, f’, i*

Không giống như các cách thức trước, learning rate hầu như giống nhau cho quá trình learning, adagrad coi learning rate cũng là một tham số. Nó tạo các update lớn với các dữ liệu khác biệt nhiều và các update nhỏ cho các dữ liệu ít khác biệt.

Adagrad chia learning rate với tổng bình phương của lịch sử biến thiên (đạo hàm). Quy luật cập nhật như sau:

Trong đó:

ϵ là hệ số để tránh lỗi chia cho 0, default là 1e−8

là một ma trận chéo nơi mà mỗi phần tử (i, i) là bình phương của đạo hàm vector tham số tại thời điểm t

Một lợi ích dễ thấy của Adagrad là tránh việc điều chỉnh learning rate bằng tay, thường sẽ để default là 0.01 và thuật toán sau đó sẽ tự động điều chỉnh.

Còn điểm yếu của Adagrad là tổng bình phương biến thiên sẽ lớn dần theo thời gian cho đến khi nó làm learning rate cực kỳ nhỏ, làm việc training trở nên đóng băng.

***RMSprop***

RMSprop (Hinton, 2012) là một kỹ thuật điều chỉnh learning rate có tính thích nghi, nó chưa từng được công bố, và được đề xuất bởi Geoff Hinton trong bài giảng trên Coursera.

Vì Adagrad thêm vào gradient từ epoch đầu tiên, các trọng số bị buộc phải giảm  
đơn điệu. Điều này có thể được giải quyết thông qua kỹ thuật Adadelta:  
với *γ* là một tham số mới, thường là 0*.*9. Công thức Adadelta theo Adagrad như  
sau  
*η* được thiết lập một lần và cho tất cả (thường là 10*−*3).

***Adadelta***

Adadelta (Zeiler, 2012) là một phần mở rộng của RMSprop, Adadelta sinh ra để làm giảm nhược điểm của Adagrad (việc làm thay đổi learning rate theo tính đơn điệu giảm). Nó giới hạn sự tích lũy của độ biến thiên tới một giới hạn nhất định.

Để làm được điều trên nó đưa ra khái niệm *running average* phụ thuộc vào trung bình trước và độ dốc hiện tại.

và quy luật cập nhật mới như sau:

***Adam***

Adam (Kingma & Ba, 2014) thường là sự lựa chọn mặc định đối với một số thuật toán deep learning. Giống với Adadelta và RMSprop, nó duy trì trung bình bình phương độ dốc (slope) quá khứ vt và cũng đồng thời duy trì trung bình độ dốc quá khứ mt, giống momentum.

, ,

với *β*1 và *β*2 thường là 0*.*9 và 0*.*999. Nhưng sức mạnh và tính chắc chắn của  
Adam nằm ở việc nó làm cho toàn bộ quá trình học tập phụ thuộc vào yếu tố  
precise. Để tránh những vấn đề về số trong các bước đầu tiên, các vector này  
được thay đổi tỷ lệ:

,

trước khi vào các quy tắc cập nhật:

Đây là kỹ thuật tối ưu hoá được sử dụng trong bài báo cáo này, cùng với độ giảm  
learning rate:

*α*0 được xác định bởi cross-validation, và *η*0 thường bắt đầu trong khoảng 10*−*3 *−*  
10*−*2.

Trong khi momentum giống như một quả cầu lao xuống dốc, thì Adam lại giống như một quả cầu rất nặng và có ma sát (friction), nhờ vậy nó dễ dàng vượt qua local minimum và đạt tới điểm tối ưu nhất (flat minimum).

CHƯƠNG 3: ỨNG DỤNG MẠNG CNN TRONG BÀI TOÁN PHÂN LOẠI ẢNH TƯ THẾ NGƯỜI

* 1. **KERAS**
     1. **Lý do chọn Keras để xử lý bài toán phân loại**

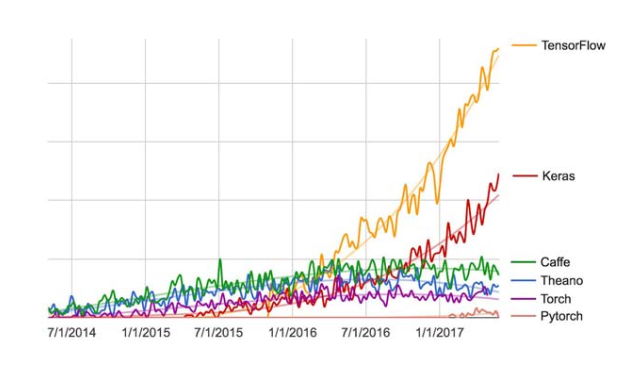
Hiện nay có rất nhiều các thư viện hỗ trợ deep-learning. Trong đó có các thư viện nổi tiếng được xây dựng bởi các hãng công nghệ lớn tiêu biểu như: Google (Keras, TensorFlow), Facebook (Caffe2, Pytorch), Microsoft(CNTK), Amazon(Mxnet)…

*Vậy lựa chọn thư viện nào là tốt nhất, phù hợp nhất với vấn để chúng ta đang giải quyết?*

Nhìn chung có một số đặc điểm mà thư viện deep-learing nên có:

* Hỗ trợ tính toán với CPU và GPU, và các hệ thống phấn tán.
* Hỗ trợ các ngôn ngữ lập trình phổ biến: C++/ Python/ Java.
* Chạy được trên nhiều hệ điều hành.
* Thời gian từ ý tưởng đến xây dựng mô hình là ngắn.
* Có khả năng giúp người lập trình can thiệp sâu vào mô hình và tạo ra các mô hình phức tạp.

Một số thống kê về tìm kiếm các framwork deep-learning trên google



Ta thấy được Tensorflow và Keras được mọi người tìm kiếm nhiều nhất. Theo thống kê tính đến năm 2017 Keras có hơn 200.000 người dùng, từ các nhà nghiêm cứu, kỹ sư, ở cả các công ty Startup và công ty lớn cho đến sinh viên hay những người có sở thích về học máy. Keras được sử dụng tại Google, Netflix, Uber, CERN, Yelp, Square, và hàng trăm công ty khởi nghiệp đang làm việc trên một loạt các vấn đề. Keras cũng là một khung phổ biến trên Kaggle, trang web về các cuộc thi deep-learning nơi mà hâu hết các chiến thắng đều thuộc về mô hình sử dụng Keras.

Tóm lại, với việc phổ biến, hỗ trợ mạnh mẽ này, chúng ta sẽ sử dụng Keras để giải quyết bài toán phân loại ảnh tư thế người trong đồ án này. Việc tìm hiểu kĩ về nó cũng sẽ giúp ích cho chúng ta sau này.

* *Vậy Keras là gì ?*

Keras là một framework deep-learning cho Python được phát triển bởi Francois Chollet, một kỹ sư nghiên cứu deep-learning tại Google. Keras được coi là một thư viện ‘high-level’ với phần ‘low-level’, (còn được gọi là backend) có thể là TensorFlow, CNTK, hoặc Theano.

Keras cung cấp thuận tiện để xác định và đào tạo cho hầu hết các loại mô hình học sâu.

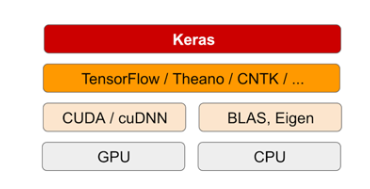
Keras bao gồm các tính năng sau:

* Cho phép cùng một mã chạy liền mạch trên CPU và GPU
* Có API cho phép người sử dụng dễ dàng nhanh chóng tạo ra các mô hình cho việc học sâu
* Tích hợp sẵn mạng Convolutional Network (cho computer vision), Recurrent Network (cho sequence processing), hoặc bất kỳ sự kết hợp nào cho cả hai.
* Hỗ trợ các kiến trúc mạng tùy ý multi-input or multi-output models,
* layer sharing, model sharing. Điều này khiến Keras phù hợp với bất kỳ mô hình học sâu nào

Keras được phân phối theo giấy phép MIT license, điều đó nghĩa là nó là mã nguồn mở và miễn phí trong cộng đồng. Cho đến thời điểm hiện tại (2019), Keras tương thích với python 2.7-3.6.

* + 1. **Cấu trúc của Keras**

Về cấu trúc của Keras, nó là một thư viện mô hình, cung cấp các khối xây dựng cấp cao để phát triển mô hình học sâu. Nó là API để gọi tới các thư viện ở lớp dưới.



Ta thấy được nó hỗ trợ hầu hết các thư viện thông dụng, nổi tiếng. Trong đồ án này chúng ta sẽ sử dụng Keras/ TensorFlow, một lần nữa vì sự phổ biến và mạnh mẽ của nó.

Về cấu trúc, để xây dựng mô hình mạng, Keras cung cấp cho chúng ta 2 cách để xây dựng model. Đó là xây dựng theo:

* Sequential

Mô hình tuần tự, tạo model bằng các layer ghép tuần tự với nhau.

* Model Class with Functional API

Sử dụng phương pháp này cho phép chúng ta tạo ra các complex neural networks

* + 1. **Tổng quan về các Layer**

Các layer trong Keras có thể dễ dàng ghép lại với nhau để tạo thành các mô hình mạng. Chúng chính là các layer mà ta đã tìm hiểu trong chương 2. Phần này sẽ giới thiệu và đi vào chi tiết các layer, điều khiến cho Keras lại mạnh mẽ và được sử dụng nhiều như vậy, vì có rất nhiều lớp trong Keras nên sẽ không thể tìm hiểu hết được. Việc đó cũng vượt quá đồ án này, nên dưới đây sẽ chỉ tìm hiểu về những layer sẽ sử dụng, và thông dụng.

Tất cả các lớp trong Keras đều có một số phương thức chung, các phương thức này sẽ cung cấp các thông tin cần thiết về mô hình lớp mạng.

layer.get\_weights() : trả về trọng số của lớp dưới dạng một danh sách các mảng Numpy.

layer.set\_weights(weights): đặt trọng số của lớp từ danh sách các mảng Numpy (có cùng hình dạng với đầu ra của get\_weights).

layer.get\_config(): trả về một từ điển chứa cấu hình của lớp. Lớp có thể được xác nhận lại từ cấu hình của nó thông qua

Nếu một lớp có một nút duy nhất (nghĩa là nếu đó không phải là một lớp dùng chung), bạn có thể nhận được tenxơ đầu vào, tenxơ đầu ra, hình dạng đầu vào và hình dạng đầu ra thông qua:

|  |
| --- |
| layer.input layer.output layer.input\_shape layer.output\_shape |

Nếu lớp có nhiều nút, có thể sử dụng các phương thức sau:

|  |
| --- |
| layer.get\_input\_at(node\_index) layer.get\_output\_at(node\_index) layer.get\_input\_shape\_at(node\_index) layer.get\_output\_shape\_at(node\_index) |

**Dense**

|  |
| --- |
| keras.layers.Dense(units, activation=None, use\_bias=True, kernel\_initializer='glorot\_uniform', bias\_initializer='zeros', kernel\_regularizer=None, bias\_regularizer=None, activity\_regularizer=None, kernel\_constraint=None, bias\_constraint=None) |

Đây chính là layer neural network như chúng ta đã nói trong chương 2.

Dense thực hiện thao tác: output = activation(dot(input, kernel) + bias) trong đó hàm activation kích hoạt phần tử được truyền dưới dạng activation đối số, kernel là ma trận trọng số được tạo bởi lớp và bias là một vectơ sai lệch được tạo bởi lớp (chỉ áp dụng nếu use\_bias là True).

units: số nguyên dương, chiều của output đầu ra

activation:dùng để chọn activation

use\_bias: boolean, sử dụng hay không vector sai lệch

kernel\_initializer: khởi tạo cho ma trận trọng số kernel

bias\_initializer: khởi tạo cho vector bias

**Dropout**

|  |
| --- |
| keras.layers.Dropout(rate, noise\_shape=None, seed=None) |

Áp dụng cho đầu vào. Dropout bao gồm việc thiết lập ngẫu nhiên một phần rate của các đơn vị đầu vào thành 0 tại mỗi lần cập nhật trong thời gian đào tạo, điều này giúp tránh overfitting.

rate: Tỷ lệ dropout

noise\_shape: Tensor 1D, được nhân với đầu vào để tạo ra sự ngẫu nhiên. Nếu đầu vào có dạng (batch\_size, timesteps, features) thì noise\_shape có dạng noise\_shape=(batch\_size, 1, features)

seed: một số nguyên Python để làm số ngẫu nhiên

**Flatten**

|  |
| --- |
| keras.layers.Flatten(data\_format=None) |

Chuyển dữ liệu đầu vào sang dạng 1D tensors, thường dùng để fully connection

**Input**

|  |
| --- |
| keras.engine.input\_layer.Input() |

Được sử dụng để khởi tạo 1 tensor Keras

**Reshape**

|  |
| --- |
| keras.layers.Reshape(target\_shape) |

Định hình lại đầu ra cho 1 đối tượng.

**Convolutional Layer**

|  |
| --- |
| keras.layers.Conv2D(filters, kernel\_size, strides=(1, 1), padding='valid', data\_format=None, dilation\_rate=(1, 1), activation=None, use\_bias=True, kernel\_initializer='glorot\_uniform', bias\_initializer='zeros', kernel\_regularizer=None, bias\_regularizer=None, activity\_regularizer=None, kernel\_constraint=None, bias\_constraint=None) |

filters: số nguyên, là số lượng bộ lọc cho lớp này.

kernel\_size: số nguyên, chỉ định chiều cao và chiều rộng của cửa sổ tích chập 2D, có thể là một số nguyên duy nhất để chỉ định cùng một giá trị cho chiều rộng và chiều cao.

strides: chỉ định bước nhảy, trượt của cửa sổ tích chập theo chiều cao và chiều rộng.

padding: same là dùng padding, còn valid là không dùng

data\_format: string, “channels\_last” hoặc "channels\_first" . Thứ tự của các kích thước trong đầu vào. "channels\_last"tương ứng với đầu vào có hình dạng (batch, height, width, channels) trong khi "channels\_first" tương ứng với đầu vào có hình dạng (batch, channels, height, width). Nếu không đặt thì mặc định của nó sẽ là “chanels\_last”.

activation: để sử dụng activation, keras hỗ trợ các hàm activation thường dùng như:softmax, elu, selu, softplus, softsign, relu, tanh, sigmoid, …

**Polling Layer**

|  |
| --- |
| keras.layers.MaxPooling2D(pool\_size=(2, 2), strides=None, padding='valid', data\_format=None) |

Được sử dụng để lấy feature nổi bật (dùng max) và làm giảm thông số khi training

pool\_size: size cửa sổ dùng trong poling.

strides: số nguyên, bộ 2 số nguyên hoặc Không có. Bước tiến giá trị. Nếu không, nó sẽ mặc định pool\_size.

padding : có giá trị là "valid"hoặc "same"(không phân biệt chữ hoa chữ thường).

data\_format :kiểu string, một trong channels\_last (mặc định) hoặc là channels\_first. Thứ tự của các kích thước trong đầu vào. channels\_last tương ứng với đầu vào có hình dạng(batch, height, width, channels) trong khi channels\_first tương ứng với đầu vào có hình dạng (batch, channels, height, width). Nó mặc định là image\_data\_format giá trị được tìm thấy trong tệp cấu hình Keras của bạn tại ~/.keras/keras.json. Nếu bạn không bao giờ đặt nó, thì đó sẽ là "channels\_last".

* + 1. **Các hàm thường sử dụng**

ImageDataGenerator tạo thêm data bằng cách scale,rotation…

Các hàm loss functions thường dùng :

* mean\_squared\_error thường dùng trong regression tính theo eculic
* mean\_absolute\_error tính theo trị tuyệt đối
* categorical\_crossentropy dùng trong classifier nhiều class
* binary\_crossentropy dùng trong classifier 2 class
* kullback\_leibler\_divergence dùng để tính loss giữa phân phối thực tế và thực nghiệm

metrics nó là thước đo để ta đánh giá accuracy của model.

* binary\_accuracy nếu y\_true==y\_pre thì trả về 1 ngược lại 0,dùng cho 2 class
* categorical\_accuracy tương tự binary\_accuracy nhưng cho nhiều class

optimizers dùng để chọn thuật toán training.

* SGD Stochastic gradient descent optimizer
* RMSprop RMSProp optimizer
* Adam Adam optimizer

activations để chọn activation function

* linear như trong linear regression
* softmax dùng trong multi classifier
* relu max(0,x) dùng trong các layer cnn,rnn để giảm chi phí tính toán
* tanh range (-1,1)
* Sigmoid range (0,1) dùng nhiều trong binary class

Callbacks : khi model chúng ta lớn có khi training thì gặp sự cố ta muốn lưu lại model để chạy lại thì callback giúp t làm điều này.

* ModelCheckpoint lưu lại model sau mỗi epoch
* EarlyStopping stop training khi training ko cải thiện model
* ReduceLROnPlateau giảm learning mỗi khi metrics ko được cải thiện
  1. **CÁC BƯỚC THỰC HIỆN MẠNG CNN TRONG BÀI TOÁN PHÂN LOẠI ẢNH TƯ THẾ NGƯỜI**

Để xây dựng một bộ phân loại tư thế người dựa trên mạng CNN thì cần thực hiện các bước sau:

**Bước 1:** **Chuẩn bị tập mẫu cho việc huấn luyện và kiểm tra mô hình**

Tập dữ liệu ảnh bao gồm 2 phần: 1 phần dành cho việc huấn luyện mạng CNN và 1 phần dành cho việc kiểm tra và đánh giá mô hình mạng đã huấn luyện. Mỗi 1 phần bao gồm 3 lớp tương ứng với 3 tư thế: đứng, ngồi và nằm. Để đểm bảo việc đánh giá mô hình được chính xác thì các ảnh nằm trong phần dữ liệu kiểm tra không được giống với các ảnh nằm trong phần dữ liệu huấn luyện.

**Bước 2: Xây dựng mô hình mạng CNN**

Dựa trên lí thuyết liên quan đến mạng CNN, cấu trúc mạng CNN bao gồm các lớp Convolution, ReLU, Pooling và Fully-Connected sẽ được xây dựng và thử nghiệm.

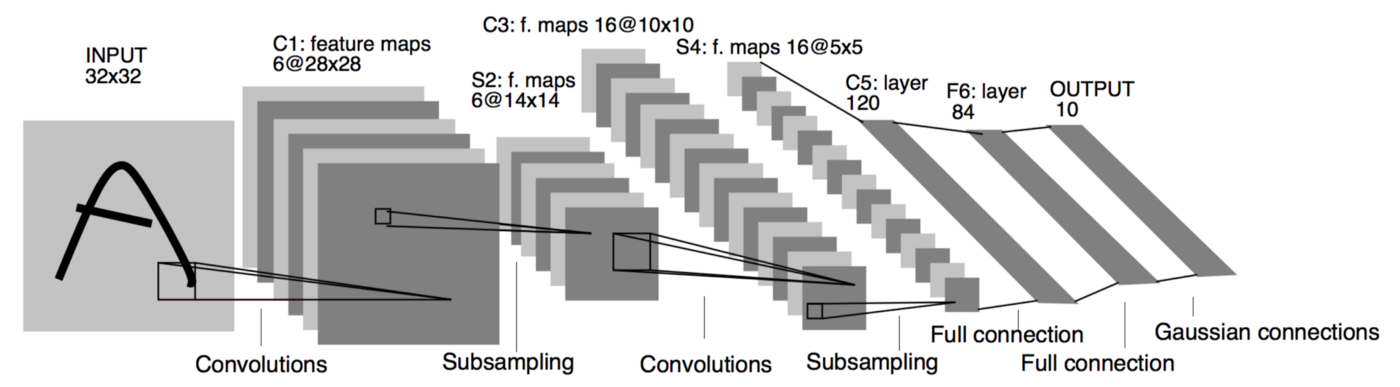
Dưới đây là một số cấu trúc mạng CNN phổ biến đã được xây dựng và sử dụng trong thực tế:

**Một số cấu trúc mạng CNN**

**Mạng LeNet5**

LeNet5 là một cấu trúc mạng CNN, được tạo ra từ năm 1998 bởi Yann LeCunn (Giám đốc nghiên cứu về AI của Facebook) cùng với Léon Bottou, Yoshua Bengio và Patrick Haffner để nhận diện chữ viết tay. Mô hình này là cơ sở thiết kế cho các mạng sau này.

*Cấu trúc mạng LeNet5:*



*Ảnh gốc được xuất bản trong [LeCun et al., 1998]*

Kiến trúc LeNet-5 bao gồm hai tập hợp các lớp gộp và trung bình, tiếp theo là một lớp chập phẳng, sau đó là hai lớp được kết nối đầy đủ và cuối cùng là một bộ phân loại softmax.

Theo tiêu chuẩn hiện đại, LeNet-5 là một mạng rất đơn giản. Nó chỉ có 7 lớp, trong đó có 3 lớp tích chập (C1, C3 và C5), 2 lớp lấy mẫu phụ (gộp) (S2 và S4) và 1 lớp được kết nối đầy đủ (F6), cuối cùng là lớp đầu ra. Các lớp tích chập sử dụng bộ lọc kích thước 5x5 với bước nhảy bằng 1. Lớp lấy mẫu phụ là 2x2. Hàm kích hoạt tanh được sử dụng trên toàn mạng.

Có một số kiến trúc thú vị đã được thực hiện trong LeNet5 không phổ biến trong học sâu hiện đại. Đầu tiên, các nhân (kernel) tích chập riêng lẻ trong lớp C3 không sử dụng tất cả các tính năng (feature) được tạo bởi lớp S2, điều này rất bất thường theo tiêu chuẩn ngày nay. Có thể giải thích cho điều đó là làm cho mạng đòi hỏi tính toán ít hơn. Lý do khác là điều này có ý nghĩa hoàn hảo: nếu các nhân khác nhau nhận được đầu vào khác nhau, chúng sẽ học các mô hình khác nhau.

Thứ hai, lớp đầu ra sử dụng 10 nơ-ron chức năng cơ sở bán kính Euclide tính toán khoảng cách L2 giữa vectơ đầu vào của kích thước 84 và **các vectơ có trọng số được xác định trước** theo cùng một chiều. Số 84 xuất phát từ thực tế là về cơ bản các trọng số đại diện cho mặt nạ nhị phân 7x12, một cho mỗi chữ số. Điều này buộc mạng phải chuyển đổi hình ảnh đầu vào thành một biểu diễn bên trong sẽ làm cho đầu ra của lớp F6 càng gần càng tốt với trọng số được mã hóa bằng tay của 10 nơ ron của lớp đầu ra.

LeNet-5 có thể đạt được tỷ lệ lỗi dưới 1% trên tập dữ liệu MNIST, rất gần với trạng thái hiện đại (được tạo ra bởi một nhóm gồm ba mạng LeNet-4 được tăng cường).

**Mạng riêng:**

Mạng gồm các lớp Con2D + MaxPolling2D được xếp nối tiếp nhau, 3 lần liên tiếp. Sau đó đi qua 1 lớp Flatten, rồi đến lớp Dropout với rate = 0.5, Lớp Dense có đầu ra là 512 rồi cuối cùng là lớp Dense với đầu ra là số lượng classes cần phân loại. Mạng sử dụng thuật toán SGD để tối ưu hàm loss.

**Bước 3: Huấn luyện mô hình mạng CNN**

Các thông số cũng như thuật toán huấn luyện mô hình mạng CNN được thiết lập trước khi triển khi qua trình học cho mạng trên tập dữ liệu huấn luyện đã được chuẩn bị tại bước 1.

**Bước 4: Lưu các mô hình mạng đã được huấn luyện**

Trong quá trình huấn luyện mạng, các mô hình đã được huấn luyện sau *n* epochs (hoặc iterations) sẽ được lưu lại và sử dụng cho việc kiểm tra và đánh giá chất lượng mô hình mạng đã được huấn luyện tại bước 5.

**Bước 5: Kiểm tra và đánh giá chất lượng mô hình mạng đã được huấn luyện trên tập kiểm tra**

Tại bước này, các mô hình đã lưu tại bước 4 được kiểm tra trên tập dữ liệu kiểm tra đã chuẩn bị tại bước 1.

**Bước 6: Hiệu chỉnh các tham số của mô hình mạng**

Dựa vào các kết quả kiểm tra của các mô hình đã kiểm nghiệm tại bước 5, các thông số liên quan đến cấu trúc mạng và quá trình học sẽ được thay đổi 1 cách thực nghiệm.

**Bước 7: Lặp lại bước 3**

Sau khi đã thay đổi các tham số tại bước 6, thực hiện lại quá trình huấn luyện mạng. Mục đích lặp lại quá trình huấn luyện cho đến hiệu chỉnh là để tìm ra bộ các tham số tối ưu mô hình mạng bằng cách thử nghiệm nhiều lần.

CHƯƠNG 4: KẾT QUẢ THỰC NGHIỆM

Chương này sẽ thực hiện triển khai mô hình, huấn luyện và đánh giá mạng tạo ra.

* 1. **CÀI ĐẶT KERAS, TENSORFLOW**

Để tiến hành cài đặt Keras và tensorflow cần phải cài đặt python, và pip (thay thế pip có thể sử dụng anacoda hoặc các phần mềm khác để tải các gói Keras về)

Cài đặt python bằng cách vào trang chủ<https://www.python.org/> tải phiên bản phù hợp về, việc cài đặt được thực hiện dễ dàng và trực quan. Sau khi cài đặt xong

Kiểm tra phiên bản python và pip

|  |
| --- |
| $ python –version  $ pip –version |

Cài đặt tensorflow

|  |
| --- |
| $ pip install tensorflow |

Cài đặt Keras

|  |
| --- |
| $ pip install keras |

Keras sẽ tự động thêm tensorflow làm thư viện để gọi

Kiểm tra xem đã cài đặt thành công

**Cài đặt Jupyter Notebook**

Jupyter notebook là một công cụ tuyệt vời để chúng ta code và chạy thử các mạng trong keras mà chúng ta xây dựng, nó cho chúng ta thấy rõ được việc chạy từng lệnh như thế nào. Công cụ này được sử dụng rộng rãi trong khoa học dữ liệu và cộng đồng machine learning. Link download ở (<https://jupyter.org>) , sau khi cài đặt chúng ta có thể chỉnh sửa trong trình duyệt của mình.

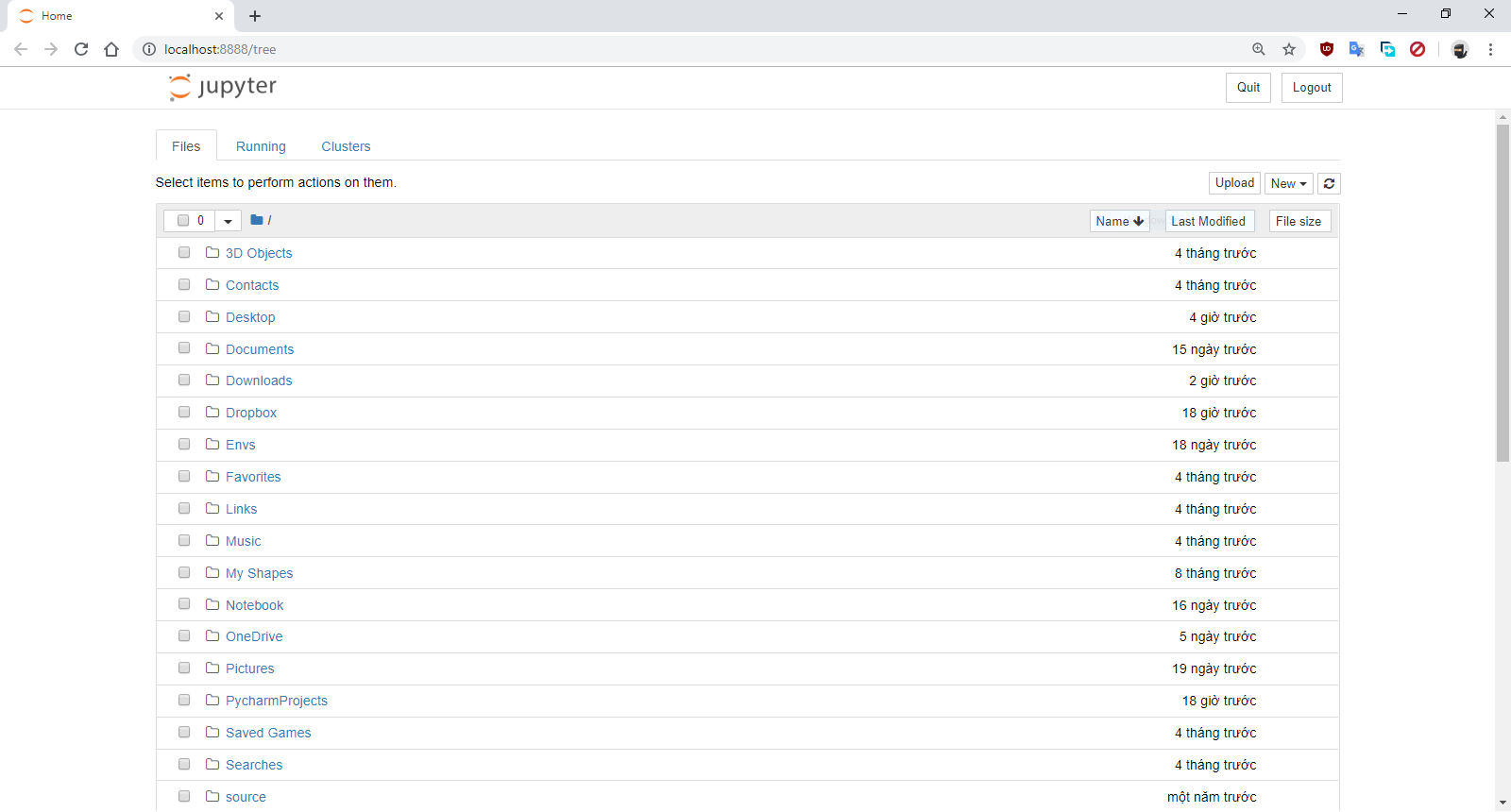
Tương tự như cài đặt tensorflow và keras, việc cài đặt jupyter notebook cũng rất đơn giản.

|  |
| --- |
| $ pip install jupyter |

Sau khi cài đặt xong, khởi động jupyter trong cmd trong windows (nếu sử dụng linux thì cũng tương tự trong terminal). Điều hướng đến thư mục làm việc và chạy

|  |
| --- |
| $ jupyter notebook |

Khởi động lên sẽ thấy như sau:



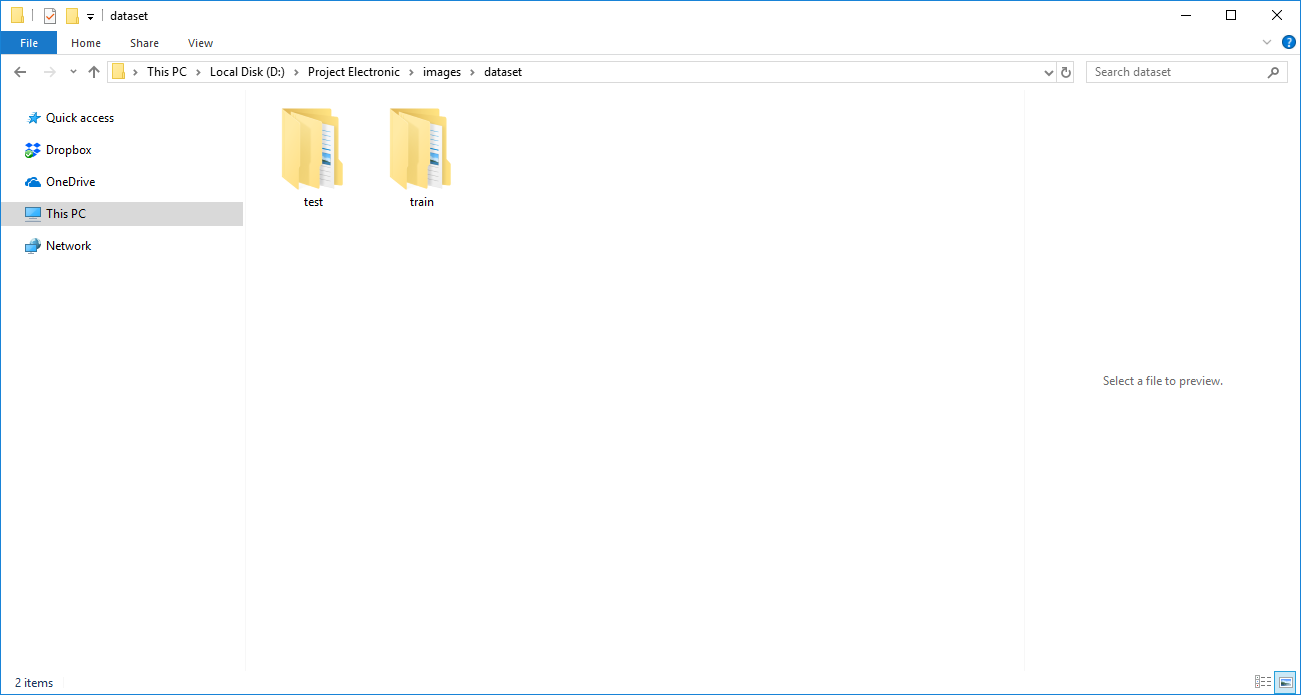
Để bắt đầu làm việc với Jupyter chọn **New** ⭢ **Python3**.

* 1. **THỰC NGHIỆM**

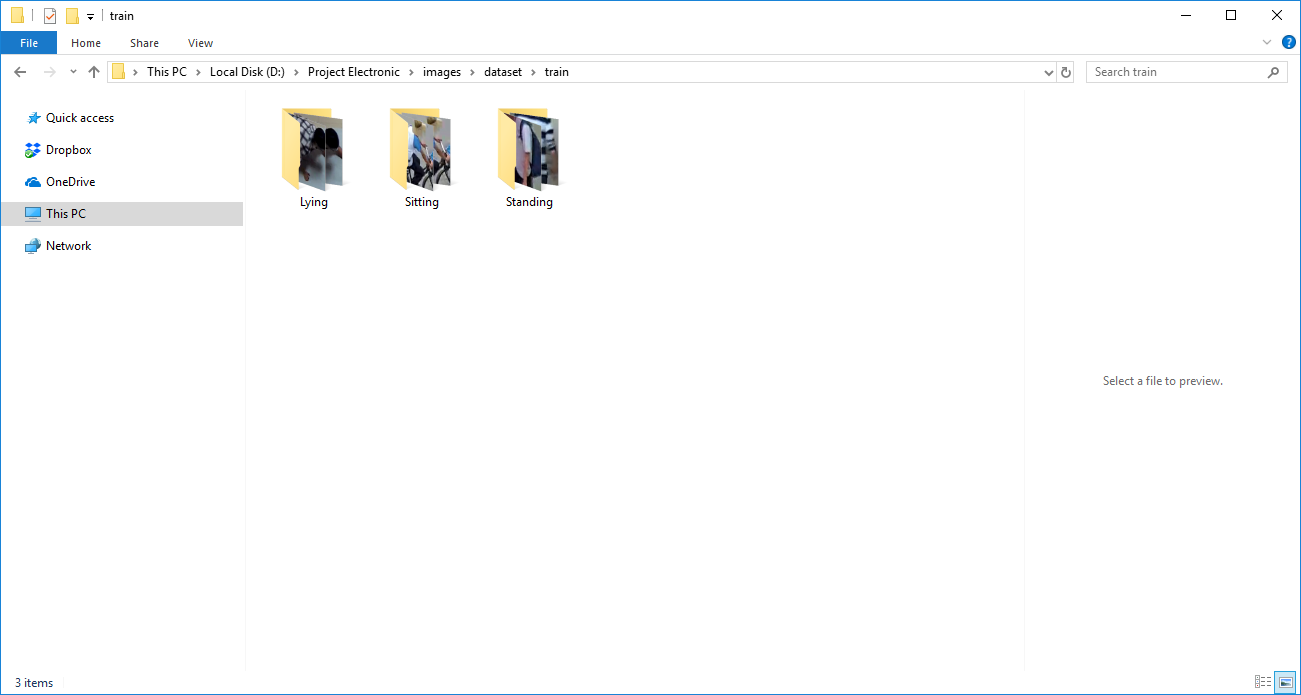
Thực hiện lần lượt các bước như trong chương 3 mô tả.

**Bước 1: Tập mẫu huấn luyện và kiểm tra**

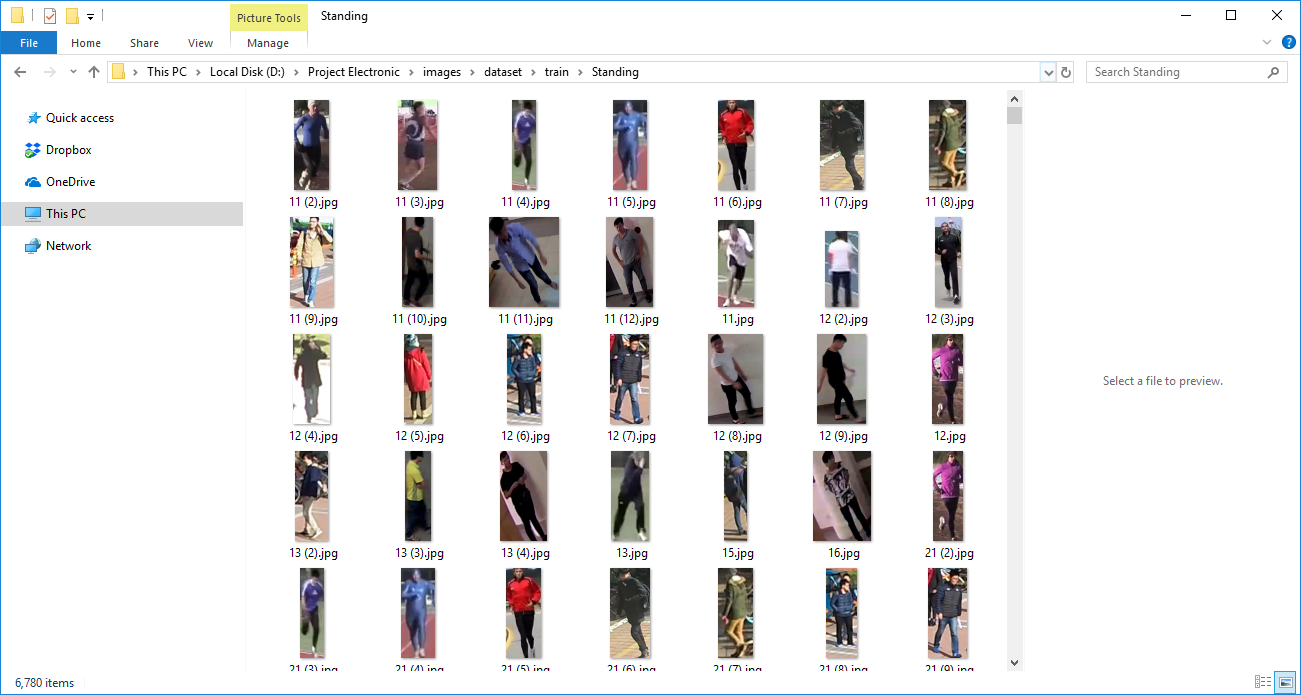
Dưới đây là tập dữ liệu thường gặp trong thực tế, đã được phân chia rõ ràng thành các folder tương ứng.

****

Trong mỗi thư mục train và test đều có 3 thư mục nhỏ, tương ứng với 3 tư thế người cần huấn luyện cho mạng



Trong mỗi folder sẽ chứa các ảnh tương ứng về tư thế đó



Tuy nhiên tập dữ liệu này có số lượng ảnh không đồng đều nhau, cụ thể

train:

* Lying: 8767 ảnh
* Sitting: 6588 ảnh
* Standing: 6780 ảnh

test:

* Lying: 1224 ảnh
* Sitting: 1637 ảnh
* Standing: 1560 ảnh

Đây là việc rất thường gặp trong thực tế, vì số lượng ảnh thu thập được là không bằng nhau. Nên dưới đây sẽ viết 1 chương trình để đưa dữ liệu ở các folder thống nhất với nhau, để tạo nên sự khách quan cho mạng sẽ huấn luyện.

Để dự án này được rõ ràng dễ chỉnh sửa và quản lý, đồ án này sẽ đặt tất cả các tệp, thư mục dữ liệu vào trong 1 folder duy nhất.

Tạo folder cho project:

|  |
| --- |
| $ mkdir HumanPosture $ cd HumanPosture |

// Create a folder for dataset

|  |
| --- |
| $ mkdir dataset $ mkdir dataset\train $ mkdir dataset\test $ mkdir dataset\train\Lying $ mkdir dataset\train\Sitting $ mkdir dataset\train\Standing $ mkdir dataset\test\Lying $ mkdir dataset\test\Sitting $ mkdir dataset\test\Standing |

Sau khi đã tạo thư mục dự án, cũng như các thư mục cần thiết cho tập dữ liệu, chạy jupyter notebook để thực hiện code với giao diện trực quan và dễ dàng.

Trong cmd gõ:

|  |
| --- |
| $ jupyter notebook |

Cửa sổ hiện lên, chọn new ⭢ python3 , rename tên file python thành copy\_data

trong file mới này sẽ viết chương trình copy dữ liệu từ thư mục gốc sang thư mục dự án và đưa dữ liệu về chung 1 chuẩn, cụ thể ở đây là sẽ copy 6000 ảnh cho mỗi thư mục con của tập train, và 1000 ảnh cho mỗi thư mục con tập test.

|  |
| --- |
| import os, shutil, glob def copy\_image(src, dist, number):     count = 0     for jpgfile in glob.iglob(os.path.join(src, "\*.jpg")):         count += 1         if count > number:             print("copied", number, "image from", src, "to ",dist)             break         else:             shutil.copy(jpgfile, dist) |

|  |
| --- |
| def copy\_image\_to\_dist(srcPaths, distPaths, number):     for path in os.listdir(distPaths):         src = os.path.join(srcPaths, path)         dist = os.path.join(distPaths, path)         copy\_image(src, dist, number)         print('------------') |

|  |
| --- |
| srcPaths\_train = "D:/Project Electronic/images/dataset/train" distPaths\_train = "./dataset/train"  copy\_image\_to\_dist(srcPaths\_train, distPaths\_train, 6000) |

Tiến hành tương tự để copy 1000 ảnh vào thư mục test

**Bước 2: Xây dựng mô hình mạng CNN**

Mạng LeNet5:

Tạo 1 file riêng để mô hình mạng được tách riêng ra dễ kiểm soát, đặt tên file là LeNet5.py, với phương pháp này chúng ta có thể sử dụng để xây dựng nhiều mạng khác nhau và tiến hành thử nghiệm trên từng mạng để tìm được mạng tốt nhất cho bài toán đang giải quyết.

Xây dựng mô hình mạng LeNet5 bằng Keras như sau:

|  |
| --- |
| import keras  from keras.models import Sequential from keras.layers import Conv2D from keras.layers import MaxPooling2D from keras.layers import Flatten from keras.layers import Dense   model = Sequential() #Layer 1 #Conv Layer 1 model.add(Conv2D(filters = 6,                   kernel\_size = 5,                   strides = 1,                   activation = 'relu',                   input\_shape = (32,32,1))) #Pooling layer 1 model.add(MaxPooling2D(pool\_size = 2, strides = 2)) #Layer 2 #Conv Layer 2 model.add(Conv2D(filters = 16,                   kernel\_size = 5,                  strides = 1,                  activation = 'relu',                  input\_shape = (14,14,6))) #Pooling Layer 2 model.add(MaxPooling2D(pool\_size = 2, strides = 2)) #Flatten model.add(Flatten()) #Layer 3 #Fully connected layer 1 model.add(Dense(units = 120, activation = 'relu')) #Layer 4 #Fully connected layer 2 model.add(Dense(units = 84, activation = 'relu')) #Layer 5 #Output Layer model.add(Dense(units = 10, activation = 'softmax')) model.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy']) |

Mạng riêng:

Tạo 1 file mới, rename thành smallCNN

|  |
| --- |
| import keras  from keras import layers from keras import models from keras import optimizers from keras.models import Sequential from keras.layers import Conv2D from keras.layers import MaxPooling2D from keras.layers import Flatten from keras.layers import Dense  class smallCNN:     @staticmethod     def build(width, height, depth, classes):         model = models.Sequential()         model.add(layers.Conv2D(32, (3, 3), activation='relu',                 input\_shape=(height, width, depth)))         model.add(layers.MaxPooling2D((2, 2)))         model.add(layers.Conv2D(64, (3, 3), activation='relu'))         model.add(layers.MaxPooling2D((2, 2)))         model.add(layers.Conv2D(128, (3, 3), activation='relu'))         model.add(layers.MaxPooling2D((2, 2)))         model.add(layers.Conv2D(128, (3, 3), activation='relu'))         model.add(layers.MaxPooling2D((2, 2)))         model.add(layers.Flatten())         model.add(layers.Dropout(0.5))         model.add(layers.Dense(512, activation='relu'))         model.add(layers.Dense(classes, activation='softmax'))          model.compile(loss='categorical\_crossentropy',         optimizer=optimizers.SGD(lr=0.01),         metrics=['accuracy'])          return model |

**Bước 3 & 4: Huấn luyện mô hình mạng và lưu mô hình mạng**

Trong jupyter notebook tạo 1 file mới, rename thành train\_and\_test. Đây chính là nơi mà chúng ta viết code training mạng và test trên tập dữ liệu.

Xử lý dữ liệu đầu vào nhằm mục đích đưa dữ liệu về dạng tensor phù hợp với input đầu vào của Keras, tức là đưa về dạng 3D tensor (height, width, color\_depth)

|  |
| --- |
| #data processing  IMAGE\_DIMS = (96, 96, 3) imagePaths = sorted(list(paths.list\_images('dataset\\train'))) random.seed(42) random.shuffle(imagePaths)  # initialize the data and the labels  data = [] labels = [] for imagePath in imagePaths:     image = cv2.imread(imagePath)     image = cv2.resize(image, (IMAGE\_DIMS[1], IMAGE\_DIMS[0]))     image = img\_to\_array(image)     data.append(image)     label = imagePath.split(os.path.sep)[-2]     labels.append(label)  # scale the raw pixel intensities to the range [0, 1] data = np.array(data, dtype="float") / 255.0 print("[INFO] data matrix: {:.2f}MB".format(     data.nbytes / (1024 \* 1000.0))) labels = np.array(labels) # binarize the labels lb = LabelBinarizer() labels = lb.fit\_transform(labels) |

Chia dữ liệu thành 2 phần: 1 phần để train và 1 phần để validition

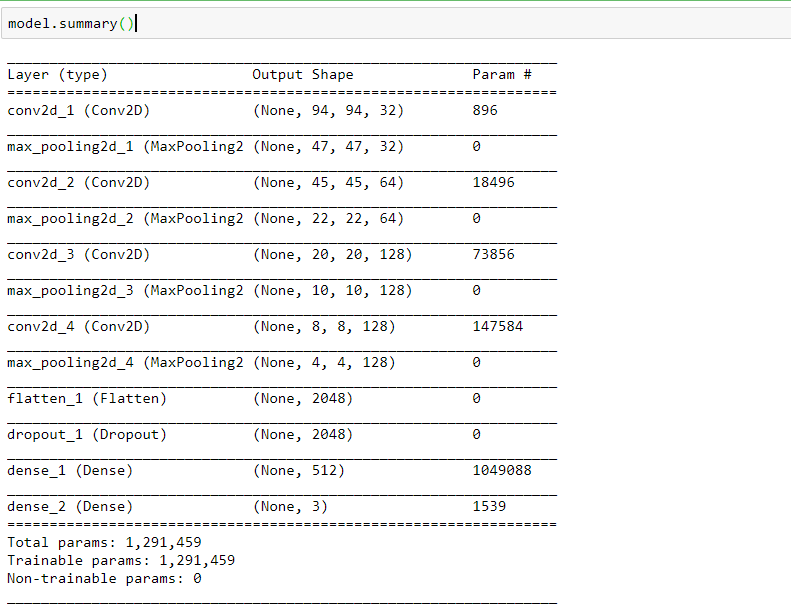
|  |
| --- |
| # partition the data into training and validition(test\_size) (trainX, valX, trainY, valY) = train\_test\_split(data, labels, test\_size=0.2, random\_state=42) |

Load mạng đã tạo ra ở trên, để chuẩn bị tiến hành training mạng

|  |
| --- |
| from smallCNN import smallCNN  num\_class = len(lb.classes\_) model = smallCNN.build(IMAGE\_DIMS[1], IMAGE\_DIMS[0], IMAGE\_DIMS[2], num\_class) |

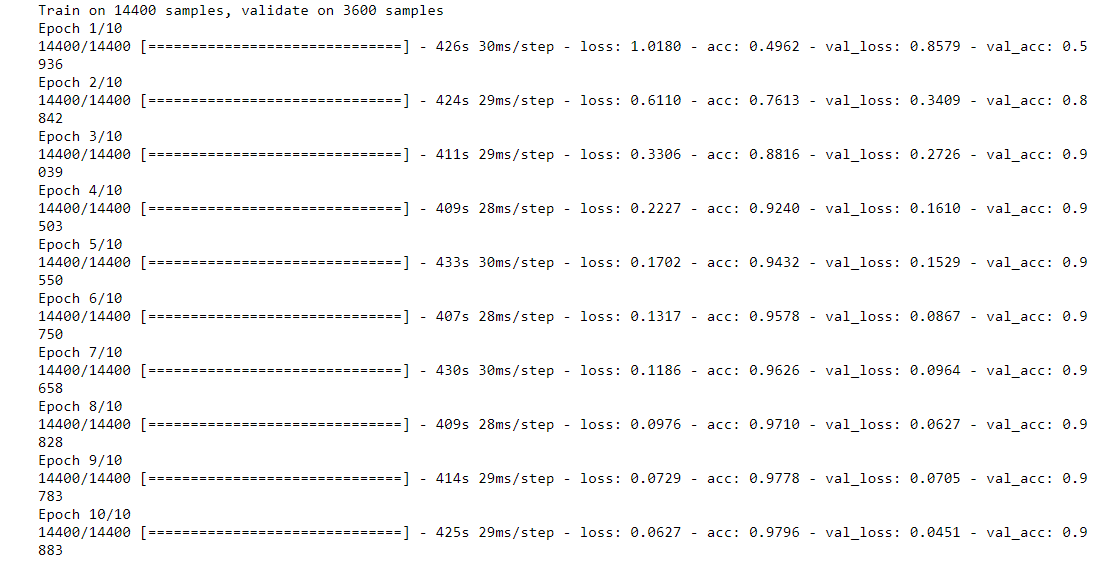
Duyệt lại thông số của mạng

|  |
| --- |
| model.summary() |



Huấn luyện mạng và lưu mô hình mạng theo từng epoch

|  |
| --- |
| #save model with every epoch and show it by TensorBoard  from tensorflow.keras.callbacks import TensorBoard import keras filepath="weights-improvement-{epoch:02d}-{val\_acc:.2f}.hdf5" callbacks\_list = [     keras.callbacks.TensorBoard(     log\_dir='my\_log\_dir',     histogram\_freq=1,     ),     keras.callbacks.ModelCheckpoint(filepath, save\_weights\_only=False) ]  #start training network  BS = 32 EPOCHS = 10 H = model.fit(trainX, trainY,               batch\_size=BS,               validation\_data=(valX, valY),               epochs=EPOCHS,                callbacks=callbacks\_list) |



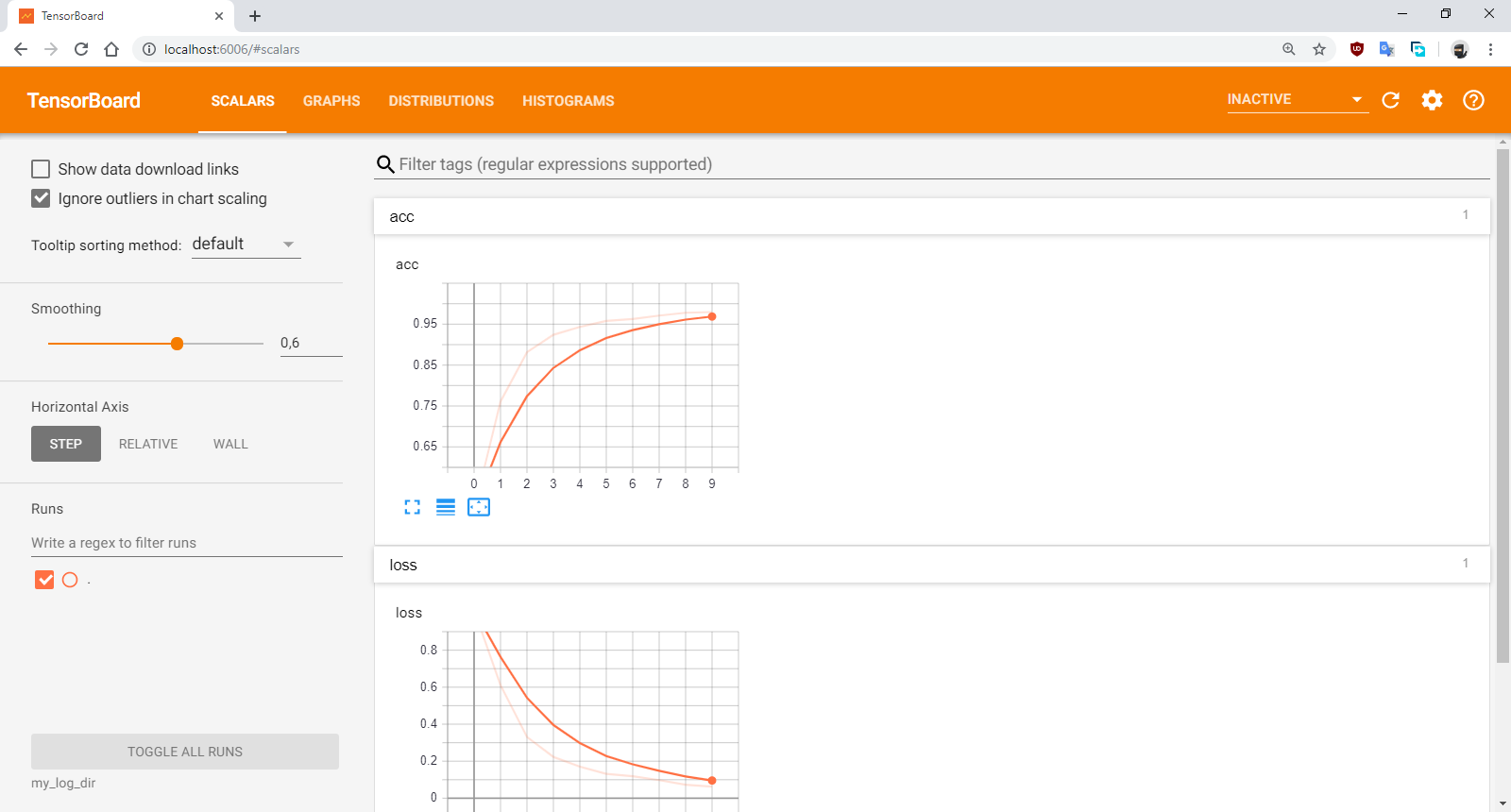
**Bước 5: Kiểm tra đánh giá mô hình mạng**

Ở trên chúng ta đã cấu hình sử dụng hàm callbacks để lưu lại mô hình sau mỗi epoch, và lưu lại giá trị val\_acc, val\_loss, acc, loss trong thư mục my\_log\_dir. Đây là 1 công cụ trực quan rất tuyệt vời của Keras giúp đánh giá mô hình nhanh và chính xác.

Khởi động Tensorboard ở cmd:

|  |
| --- |
| $ tensorboard --logdir=my\_log\_dir |

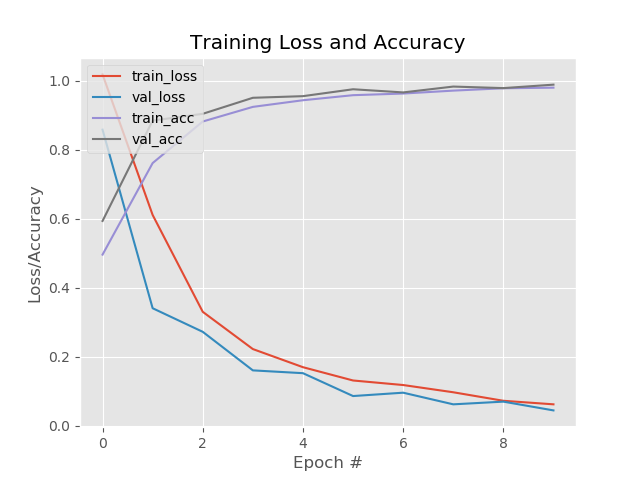
Điều hướng đến trang localhost://6006



Hoặc có thể dùng thư viện matplotlib để hiển thị giá trị acc, loss:

|  |
| --- |
| # plot the training loss and accuracy plt.style.use("ggplot") plt.figure() N = EPOCHS plt.plot(np.arange(0, N), H.history["loss"], label="train\_loss") plt.plot(np.arange(0, N), H.history["val\_loss"], label="val\_loss") plt.plot(np.arange(0, N), H.history["acc"], label="train\_acc") plt.plot(np.arange(0, N), H.history["val\_acc"], label="val\_acc") plt.title("Training Loss and Accuracy") plt.xlabel("Epoch #") plt.ylabel("Loss/Accuracy") plt.legend(loc="upper left") plt.savefig("plot") |

Thể hiện trên hình dưới



**Đánh giá mô hình dựa trên dữ liệu test**

Trước hết đưa dữ liệu test về dạng chuẩn:

|  |
| --- |
| data\_test = [] labels\_test = [] imageTestPaths = sorted(list(paths.list\_images("dataset\\test"))) for imagePath in imageTestPaths:     image = cv2.imread(imagePath)     image = cv2.resize(image, (IMAGE\_DIMS[1], IMAGE\_DIMS[0]))     image = img\_to\_array(image)     data\_test.append(image)     label = imagePath.split(os.path.sep)[-2]     labels\_test.append(label)  data\_test = np.array(data\_test, dtype="float") / 255.0 label\_tests = np.array(labels\_test) print("[INFO] data matrix: {:.2f}MB".format(     data\_test.nbytes / (1024 \* 1000.0)))  # binarize the labels lb = LabelBinarizer() labels\_test = lb.fit\_transform(labels\_test) |

Đánh giá mô hình trên dữ liệu tập test (loss, acc)

|  |
| --- |
| test\_loss, test\_acc = model.evaluate(data\_test, labels\_test) |

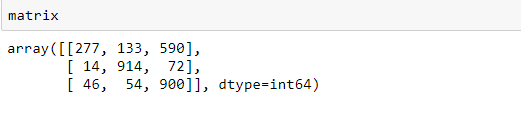
Kết quả chạy:



Đánh giá mô hình bằng confusion matrix:

|  |
| --- |
| from sklearn.metrics import confusion\_matrix #predict on data  predict = model.predict(data\_test) matrix = confusion\_matrix(labels\_test.argmax(axis=1), predict.argmax(axis=1)) |

Kết quả:



**Bước 6: Hiệu chỉnh tham số, mô hình của mạng**

KẾT LUẬN

- Những kết luận về kết quả nghiên cứu của đồ án; trong đó, chú ý trình bày những điểm chính sau đây:

- Nhắc lại mục tiêu của đề tài.

- Tóm lược những nội dung đã làm được của đề tài.

- Một số đề xuất kiến nghị khi áp dụng đề tài.

- Hạn chế của đề tài.

TÀI LIỆU THAM KHẢO

1. *https://dominhhai.github.io/vi/2018/04/nn-intro/*
2. [*https://vi.wikipedia.org/wiki/M%E1%BA%A1ng\_n%C6%A1-ron*](https://vi.wikipedia.org/wiki/M%E1%BA%A1ng_n%C6%A1-ron)
3. *https://viblo.asia/p/su-dung-cnn-trong-bai-toan-nhan-dang-mat-nguoi-phan-1-eW65GoOP5DO*
4. *https://viblo.asia/p/ung-dung-convolutional-neural-network-trong-bai-toan-phan-loai-anh-4dbZNg8ylYM*
5. *https://labs.septeni-technology.jp/technote/ml-18-convolution-neural-network-part-1/*
6. *https://labs.septeni-technology.jp/technote/ml-19-convolution-neural-network-part-2/*
7. *https://labs.septeni-technology.jp/technote/ml-20-convolution-neural-network-part-3/*
8. *https://ereka.vn/post/chia-se-ve-mang-noron-tich-chap-convolutional-neural-networks-or-convnets-52790224348847566*
9. [*https://machinelearningcoban.com/2017/01/12/gradientdescent*](https://machinelearningcoban.com/2017/01/12/gradientdescent)
10. [*https://machinelearningcoban.com/2017/01/16/gradientdescent2*](https://machinelearningcoban.com/2017/01/16/gradientdescent2)
11. *https://journals.sagepub.com/doi/full/10.5772/62163*
12. *https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f*
13. *https://medium.com/@pechyonkin/key-deep-learning-architectures-lenet-5-6fc3c59e6f4*
14. *Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, Gradient-Based Learning Applied to Document Recognition, 1998.*
15. *https://engmrk.com/lenet-5-a-classic-cnn-architecture/*