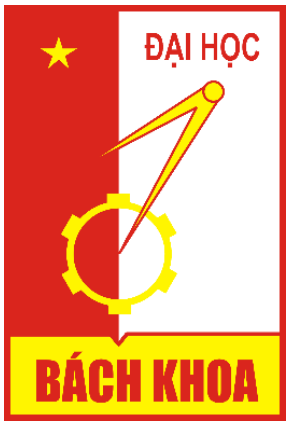
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

----- ☙ 🕮 ❧ -----



PROJECT II

Đề tài: **Tìm hiểu và xây dựng mô hình phân loại biển báo giao thông**

Giảng viên: **ThS. Vũ Đức Vượng**

Sinh viên thực hiện: Hoàng Hà My 20207644

Hà Nội, năm 2023

MỤC LỤC

MỞ ĐẦU

Xe ô tô tự lái (hay còn gọi là xe tự hành) là phương tiện có khả năng cảm nhận môi trường xung quanh và di chuyển an toàn mà không cần người lái hoặc cần sự can thiệp rất ít của con người. Loại phương này là sự kết hợp của rất nhiều công nghệ có khả năng cảm biến môi trường xung quanh như radar, lidar, GPS, AI (trí tuệ nhân tạo),… Theo hiệp hội kỹ sư ô tô Mỹ (SAE) hiện xác định 6 cấp độ khác nhau dành cho ô tô. Cấp độ 0 (no automation) sẽ cần con người điều khiển xe hoàn toàn theo phương pháp thủ công trong khi đó cấp độ 5 (full automation) sẽ cho phép xe vận hành hoàn toàn tự động.

Biển báo giao thông là những biển hiệu, chỉ dẫn trên đường thể hiện những thông tin về giao thông, mục đích cơ bản là giúp cho những người tham gia giao thông chấp hành luật giao thông một cách chính xác và an toàn nhất.

Như vậy, để nghiên cứu và xây dựng xe ô tô tự lái, rất cần thiết phải xây dựng mô hình giúp phương tiện nhận diện được biển báo giao thông, từ đó đưa ra những quyết định vận hành phù hợp.

CHƯƠNG I. TỔNG QUAN ĐỀ TÀI

I.1. Thông tin tổng quan

I.1.1. Giới thiệu đề tài

Ở đề tài này, chúng ta sẽ đi tìm hiểu và xây dựng mô hình xử lý bài toán phân loại biển báo giao thông với đầu vào (input) là các ảnh biển báo giao thông, kết quả đầu ra (output) là tên biển báo đó là biển báo gì. Ví dụ : cấm rẽ trái, cấm quay đầu, giới hạn tốc độ,…

I.1.2. Mục tiêu đề tài

Mục tiêu cụ thể của đề tài được trình bày trong bảng dưới đây theo nguyên tắc SMART [6] :

Tính cụ thể (Specific)	Tìm hiểu về học sâu và mô hình mạng nơ-ron nhân tạo, áp dụng thuật toán trên tập dữ liệu đã thu thập, đánh giá kết quả của thuật toán từ đó đưa ra kế hoạch cải tiến.
Tính đo lường (Measurable)	Thu thập hình ảnh và thông tin của biển báo giao thông để đưa vào quá trình huấn luyện, sau đó thực hiện tiền xử lý hình ảnh để kết quả dự đoán cao hơn 85%.
Tính khả thi (Achievable)	Ứng dụng được xây dựng và có thể triển khai như một phần của hệ thống tự lái. Ứng dụng được xây dựng và có thể triển khai như một phần của hệ thống tự lái.
Tính thực tế (Realistic)	Phạm vi nghiên cứu của đề tài phù hợp với trình độ của sinh viên thực hiện cũng như kết quả mà đề tài mang lại phù hợp với tình hình thực tế hiện
Tính thời hạn (Timely)	Đề tài được hoàn thành đúng tiến độ theo thời gian của kì học mà người học thực hiện đề tài. Đề tài được hoàn thành đúng tiến độ theo thời gian của kì học mà người học thực hiện đề tài.

I.1.3. Phương pháp thực hiện đề tài

Đề tài này sử dụng phương pháp phân tích và tổng kết kinh nghiệm. Cụ thể là từ những công trình nghiên cứu liên quan đến đề tài và sự hỗ trợ từ các thư viện học máy để đề xuất một cách tiếp cận trong giải quyết vấn đề đặt ra.

Nội dung nghiên cứu về lý thuyết sẽ tập trung giới thiệu học sâu và mô hình sẽ sử dụng ở mức tổng quan. Sau đó, người thực hiện thực hiện huấn luyện mô hình và kiểm thử hệ thống phân loại biển báo giao thông.

I.1.4. Đối tượng nghiên cứu

Đối tượng nghiên cứu của đề tài là phương pháp và ứng dụng phân loạibiển báo giao thông qua ảnh chụp. Do tính có thời hạn và năng lực của người thực hiện, đề tài sẽ được huấn luyện trên bộ dữ liệu biển báo giao thông German Traffic Sign (GTSRB) có sẵn. Mô hình xây dựng có thể sử dụng được với hệ thống biển báo giao thông tại Việt Nam khi có bộ dữ liệu về biển báo giao thông Việt Nam.

CHƯƠNG II. LÝ THUYẾT TỔNG QUAN

II.1. Biểu diễn ảnh

II.1.1. Hệ màu RGB

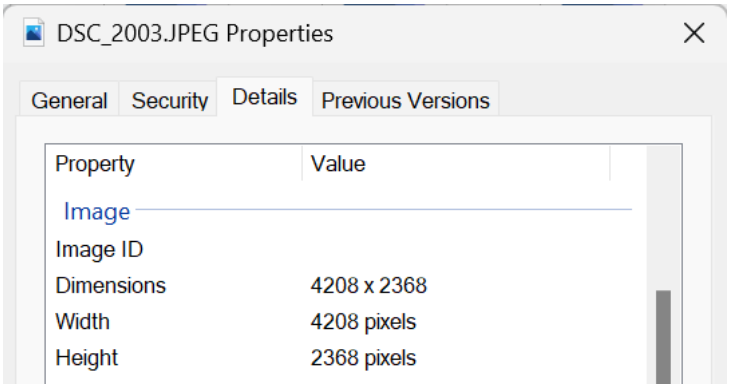
Ba màu chính của ánh sáng khi tách ra từ lăng kính là đỏ (Red), xanh lục (Green), xanh lam (Blue). Khi trộn ba màu trên theo tỉ lệ nhất định sẽ tạo ra các màu khác nhau.

H1. Mô hình màu RGB (Wikipedia)

Với mỗi bộ ba số nguyên trong khoảng [0, 255] sẽ tạo thành một màu khác nhau. Như vậy, số màu có thể tạo ra khi sử dụng hệ màu RGB là: 16777216 màu.

II.1.2. Ma trận hóa ảnh màu

Khi xem thông tin chi tiết một ảnh màu được lưu trong máy tính, giả sử:



H2. Ví dụ phần mô tả ảnh trong máy tính

Ta sẽ thấy chiều dài ảnh là 4208 pixels, chiều rộng ảnh là 2368 pixels, kích thước bức ảnh là 4208 * 2368. Pixel (hay điểm ảnh) là một điểm vật lý trong một hình ảnh raster, hoặc một khối màu rất nhỏ và là đơn vị cơ bản nhất để tạo nên một bức ảnh kỹ thuật số.

Với bức ảnh trên ta có thể biểu diễn dưới dạng một ma trận có 2368 hàng và 4208 cột.

$$\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,4208} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,4208} \\ \cdots & \cdots & \cdots & \cdots \\ w_{2368,1} & w_{2368,2} & \cdots & w_{2368,4208} \end{bmatrix} \textit{trong đó, mỗi phần tử } w_{i,j} \textit{ là một pixel.}$$

Tuy nhiên, để biểu diễn một màu ta cần 3 thông số (r, g, b) nên pixel sẽ có dạng véc – tơ $w_{i,j} = (r_{i,j}, g_{i,j}, b_{i,j})$.

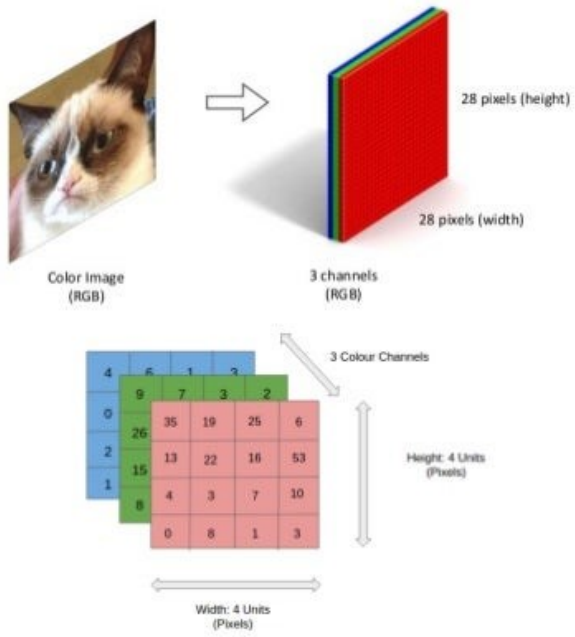
Để tiện lưu trữ và xử lý, ta sẽ không lưu ảnh màu bằng một ma trận với mỗi phần tử là các véc – tơ ba chiều mà sẽ tách mỗi giá trị (r, g, b) trong một pixel thành một ma trận riêng.

$$\begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,4208} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,4208} \\ \cdots & \cdots & \cdots & \cdots \\ r_{2368,1} & r_{2368,2} & \cdots & r_{2368,4208} \end{bmatrix} \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,4208} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,4208} \\ \cdots & \cdots & \cdots & \cdots \\ g_{2368,1} & g_{2368,2} & \cdots & g_{2368,4208} \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,4208} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,4208} \\ \cdots & \cdots & \cdots & \cdots \\ b_{2368,1} & b_{2368,2} & \cdots & b_{2368,4208} \end{bmatrix}$$

R G B

Mỗi ma trận được tách ra được gọi là 1 channel nên ảnh màu gồm 3 channel: channel red, channel green, channel blue.

Tensor là dữ liệu nhiều hơn hai chiều, ví dụ như biểu diễn ảnh màu trên máy tính sẽ được biểu diễn dưới dạng tensor 3 chiều kích thước 600*800*3 do có 3 ma trận (channel) red, green và blue kích thước 600*800.



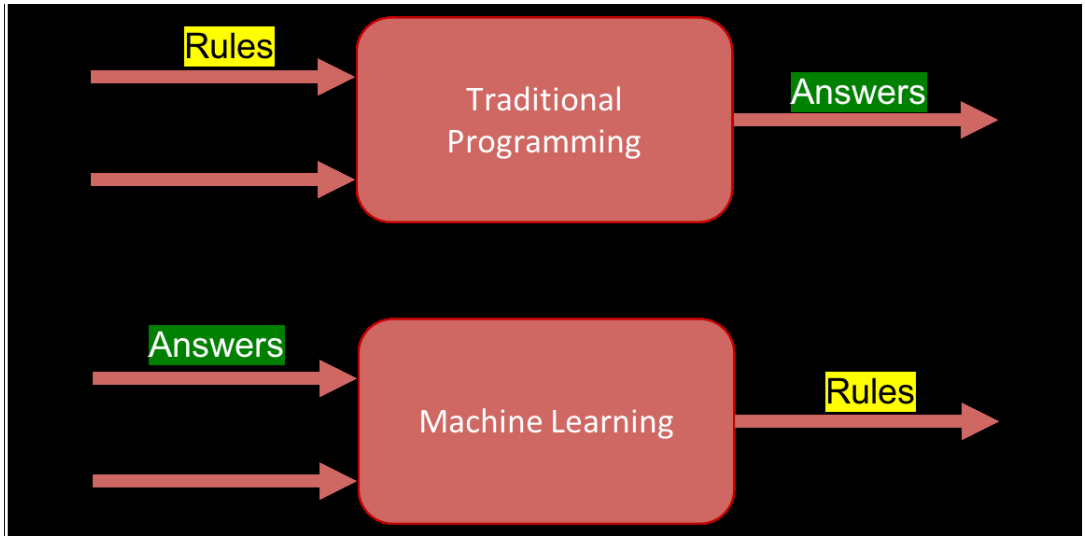
H3. Ví dụ biểu diễn ảnh màu dưới dạng tensor
(source: https://www.slideshare.net/BertonEarnshaw/a-brief-survey-of-tensors)

II.2. Tổng quan về Học máy, học sâu và Mạng nơ-ron nhân tạo

II.2.1. Học máy là gì?

Theo Wikipedia, học máy (*Machine Learning*) là một lĩnh vực của trí tuệ nhân tạo liên quan đến việc nghiên cứu và xây dựng các kĩ thuật cho phép các hệ thống "học" tự động từ dữ liệu để giải quyết những vấn đề cụ thể. Các thuật toán học máy xây dựng một mô hình dựa trên dữ liệu mẫu, được gọi là dữ liệu huấn luyện, để đưa ra dự đoán hoặc quyết định mà không cần được lập trình chi tiết về việc đưa ra dự đoán hoặc quyết định này. Ví dụ như các máy có thể "học" cách phân loại thư điện tử xem có phải thư rác (spam) hay không và tự động xếp thư vào thư mục tương ứng.

Chương trình học máy và chương trình lập trình truyền thống có sự khác biệt lớn. Với các chương trình lập trình truyền thống, máy tính nhận các nguyên tắc, điều kiện để thực hiện nhiệm vụ, từ đó đưa ra câu trả lời. Ngược lại, đối với học máy, chương trình dựa vào dữ liệu và quan sát để tìm ra được các nguyên tắc này.



H4. Sự khác biệt giữa chương trình truyền thống và chương trình học máy (Machine Learning vs. Traditional Programming Paradigm - https://datalya.com)

II.2.2. Phân loại học máy

Dựa trên tiêu chí về vấn đề, nhiệm vụ cần giải quyết của bài toán, học máy được chia thành ba loại:

- Hồi quy (Regression): giải quyết bài toán dự đoạn giá trị một đại lượng nào đó dựa vào giá trị của các đại lượng tương quan, ví dụ bài toán dự đoán số lượng sản phẩm bán ra của một cửa hàng dựa trên các yếu tố như số lượng quảng cáo đã chạy, mức giảm giá được áp dụng, mùa vụ hiện tại (ví dụ: mùa hè, mùa đông), và đánh giá khách hàng,…
- Phân lớp (Classification): giải quyết các bài toán nhận dạng xem một đối tượng thuộc nào trong các lớp đã đưa ra, ví dụ bài toán phân loại hoa diên vĩ (iris) dựa trên thông số về chiều dài đài hoa (SepalLength), chiều rộng đài hoa (SepalWidth), chiều dài cánh hoa (PetalLength) và chiều rộng cánh hoa (PetalWidth).
- Phân cụm (Clustering): ý tưởng cơ bản giống với bài toán phân lớp, nhưng với bài toán phân cụm, các cụm chưa được xác định trước và thuật toán phải tự khám phá và phân cụm dữ liệu, ví dụ bài toán phân loại khách hàng của một cửa hàng thông qua số lượng đơn hàng, tổng giá trị

mua sắm, tần suất mua sắm, loại sản phẩm mua, và thời gian giữa các đơn hàng của mỗi người.

Dựa trên cách huấn luyện máy học, học máy được chia thành thành 4 loại chính:

- Học có giám sát (Supervised learning): là thuật toán dự đoán đầu ra của một dữ liệu mới dựa trên các cặp dữ liệu đầu vào – đầu ra đã biết từ trước (input – outcome / data - label). Đây là nhóm phổ biến nhất trong các thuật toán học máy. Ví dụ bài toán nhận diện biển báo giao thông, ta có ảnh của hàng chục nghìn bức ảnh biển báo giao thông được chụp trong điều kiện ánh sáng và nền khác nhau. Ta đưa vào trong một thuật toán những bức ảnh này và cho biết bức ảnh tương ứng với biển báo nào. Sau đó, khi đã hoàn thành huấn luyện, mô hình nhận được một bức ảnh nó chưa từng thấy bao giờ, nó sẽ dự đoán bức ảnh đó là biển báo nào.
- Học không giám sát (Unsupervised learning): là thuật toán mà không nhận được kết quả đầu ra hay nhãn mà chỉ có dữ liệu đầu vào, thuật toán unsupervised learning sẽ dựa vào cấu trúc của dữ liệu để thực hiện một công việc nào đó. Ví dụ bài toán phát hiện gian lận tín dụng bằng học không giám sát, chúng ta không có dữ liệu được gán nhãn (gian lận hoặc không gian lận) từ trước, với dữ liệu đầu vào bao gồm các giao dịch tín dụng, mỗi giao dịch đi kèm với các thông tin như số tiền, địa điểm, thời gian, loại giao dịch, và các biến khác liên quan và mục tiêu là phát hiện các mẫu không bình thường trong dữ liệu.
- Học bán giám sát (Semi – supervised learning): thuật toán sử dụng với các bài toán khi chúng ta có một lượng lớn dữ liệu nhưng chỉ có một phần trong chúng được gán nhãn trước. Ví dụ bài toán phân loại văn bản, ta có một tập dữ liệu văn bản lớn, chỉ một số lượng nhỏ văn bản được gán nhãn về chủ đề. Mục tiêu là xây dựng một mô hình có thể tự động phân loại các văn bản không được gán nhãn.
- Học tăng cường (Reinforcement learning): là các bài toán giúp cho một hệ thống tự động xác định hành vi dựa trên hoàn cảnh để đạt được lợi ích

cao nhất. Hiện tại, thuật toán này chủ yếu được áp dụng vào lý thuyết trò chơi (game theory). Ví dụ đào tạo một robot để chơi cờ vua và ngày càng trở nên thông minh trong quá trình chơi. Bắt đầu từ một trạng thái ngẫu nhiên trên bàn cờ, robot thực hiện các nước đi dựa trên chiến lược ngẫu nhiên ban đầu. Sau mỗi nước đi, robot nhận được điểm (reward) dựa trên hiệu suất của nó. Nếu nó thực hiện nước đi tốt (ví dụ, chiến thắng hoặc giữa trận), nó nhận được reward tích cực; ngược lại, nếu nó thực hiện nước đi kém, nó nhận được reward tiêu cực.

II.2.3. Hàm mất mát và thuật toán Gradient descent

Về bản chất học máy là quá trình mà với mỗi dữ liệu đầu vào trong quá trình huấn luyện, thuật toán sẽ thay đổi các tham số bên trong để mô hình có thể "dự đoán" tốt hơn ở tương lai.

Hàm mất mát (loss function) là một hàm số đo lường sự chênh lệch giữa giá trị dự đoán của mô hình và giá trị thực tế của dữ liệu đầu ra. Mục tiêu của việc huấn luyện mô hình là làm cho giá trị của hàm mất mát này là nhỏ nhất có thể. Hàm mất mát đóng vai trò quan trọng trong quá trình tối ưu hóa mô hình bằng cách sử dụng các phương pháp như gradient descent.

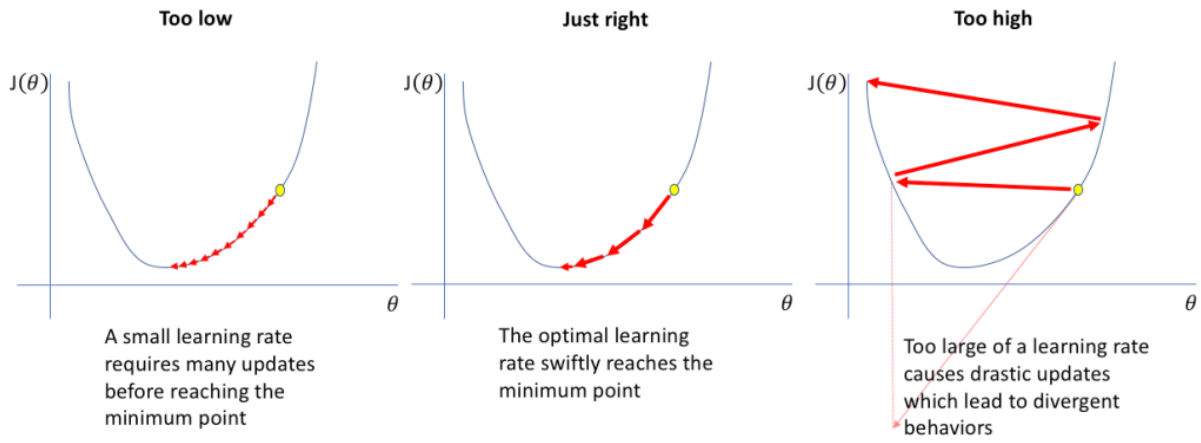
Gradient descent là thuật toán tìm giá trị nhỏ nhất của một hàm số f(x) dựa trên đạo hàm. Thuật toán gồm các bước:

- Bước 1: Khởi tạo giá trị $x = x_0$ tùy ý.
- Bước 2: Gán $x = x - learning\_rate * f'(x)$ (với learning_rate là hằng số không âm, ví dụ learning_rate = 0.002).
- Bước 3: Tính lại f(x): nếu f(x) đủ nhỏ thì dừng lại, ngược lại tiếp tục bước 2.

Thuật toán hoạt động rất tốt trong trường hợp không thể tìm giá trị nhỏ nhất bằng đại số tuyến tính.

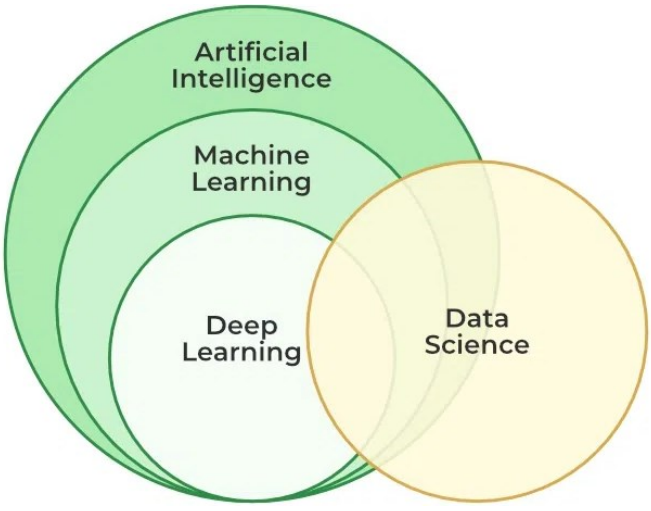
Vấn đề chọn hệ số learning_rate vô cùng quan trọng, có thể xảy ra 3 trường hợp:

- Nếu chọn learning_rate nhỏ: mỗi lần hàm số giảm rất ít nên mất rất nhiều lần thực hiện bước 2 để hàm số đạt giá trị nhỏ nhất.
- Nếu chọn learning_rate hợp lý: sau một số lần lặp bước 2 vừa phải thì hàm sẽ đạt giá trị đủ nhỏ.
- Nếu chọn learning_rate quá lớn: sẽ gây hiện tượng overshoot và không bao giờ đạt được giá trị nhỏ nhất của hàm.



H5. Đồ thị biểu diễn 3 trường hợp chọn learning_rate

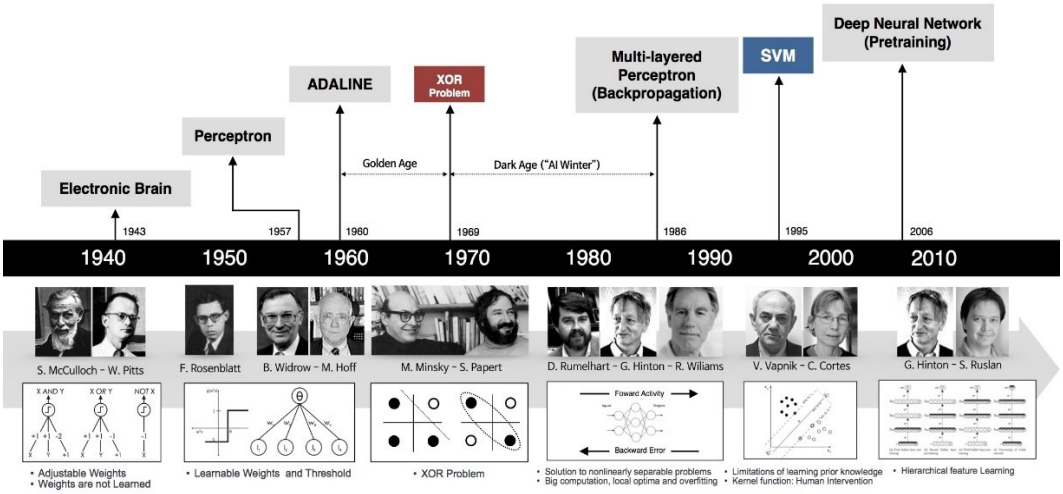
II.2.4. Định nghĩa học sâu



H6. Mối quan hệ giữa AI, ML, DS và DL

Học sâu *(Deep Learning)* là một phương pháp học máy *(Machine Learning)* trong lĩnh vực trí tuệ nhân tạo *(Artificial Intelligence)* – ở đó các máy tính sẽ học và cải thiện chính nó thông qua các thuật toán. Deep Learning được xây dựng dựa trên các khái niệm phức tạp hơn rất nhiều, chủ yếu hoạt động với các mạng nơ-ron nhân tạo (Artificial Neural Networks – ANNs) được lấy cảm hứng từ khả năng tư duy và suy nghĩ của bộ não con người. Trong những năm gần đây, học sâu đã đạt được nhiều thành công quan trọng trong nhiều lĩnh vực bao gồm nhận dạng hình ảnh, xử lý ngôn ngữ tự nhiên, nhận dạng giọng nói và hệ thống gợi ý. Một số kiến trúc học sâu phổ biến trong đó có Mạng nơ-ron tích chập (CNNs), Mạng nơ-ron hồi quy (RNNs), …

II.2.5. Lược sử học sâu



H7. Lịch sử Deep Learning

Thời gian gần đây, học sâu xuất hiện phổ biến như một cuộc cách mạng bùng nổ không chỉ với giới công nghệ mà còn len lỏi vào từng bài toán thực tế trong cuộc sống. Trên thực tế, những dấu mốc lịch sử đầu tiên của học sâu đã được xây dựng từ những năm 1940.

Lịch sử của học sâu bắt đầu từ năm **1943** khi Warren McCulloch và Walter Pitts tạo ra một mô hình máy tính dựa trên các mạng thần kinh mô phỏng hoạt động của não bộ con người. Mạng nơ-ron này có khả năng rất hạn chế và không có cơ chế học hỏi. Năm **1957**, Frank Rosenblatt công bố bài báo *"Perceptron: Thuật toán để học có giám sát các phân loại nhị phân"*. Điều này truyền cảm hứng cho cuộc cách mạng trong nghiên cứu mạng nơ-ron nông trong nhiều năm sau.

Mặc dù thuật toán này mang lại nhiều kỳ vọng, nó nhanh chóng được chứng minh không thể giải quyết những bài toán đơn giản. **Năm 1969,** sự sụp đổ của Perceptron, Marvin Minsky và Seymour Papert xuất bản cuốn sách "Perceptrons" trong đó họ chỉ ra rằng thuật toán perceptron của Rosenblatt không thể giải quyết các chức năng phức tạp như phép tính XOR. Sự thất bại này khiến cho các nghiên cứu về perceptron bị gián đoạn gần 20 năm. Thời kỳ này còn được gọi là **Mùa đông AI thứ nhất (The First AI winter)**.

Năm **1986**, Terry Sejnowski tạo ra NetTalk, một mạng nơ-ron học cách phát âm văn bản tiếng Anh bằng cách hiển thị văn bản dưới dạng đầu vào và khớp phiên âm để so sánh. Cũng trong năm này, Geoffrey Hinton, Rumelhart và Williams chỉ ra việc triển khai thành công quá trình lan truyền ngược trong mạng nơ-ron (backpropagation).

Năm **1989**, mạng CNN (Convolutional Neural Network – Mạng nơ-ron tích chập) sử dụng backpropagation, Yann LeCun sử dụng phương pháp lan truyền ngược để huấn luyện mạng nơ-ron phức hợp để nhận dạng các chữ số viết tay. Đây là một thời điểm đột phá vì nó đặt nền tảng của tầm nhìn máy tính hiện đại sử dụng học sâu.

Năm **1991**, vấn đề về Vanishing Gradient xuất hiện, Sepp Hochreiter xác định vấn đề về Vanishing Gradient có thể làm cho việc học của mạng nơron sâu trở nên cực kỳ chậm và gần như không thực tế. Vấn đề này được phát hiện khiến cho mạng nơ-ron tiếp tục rơi vào thời kỳ băng giá **(Mùa đông AI thứ hai)**.

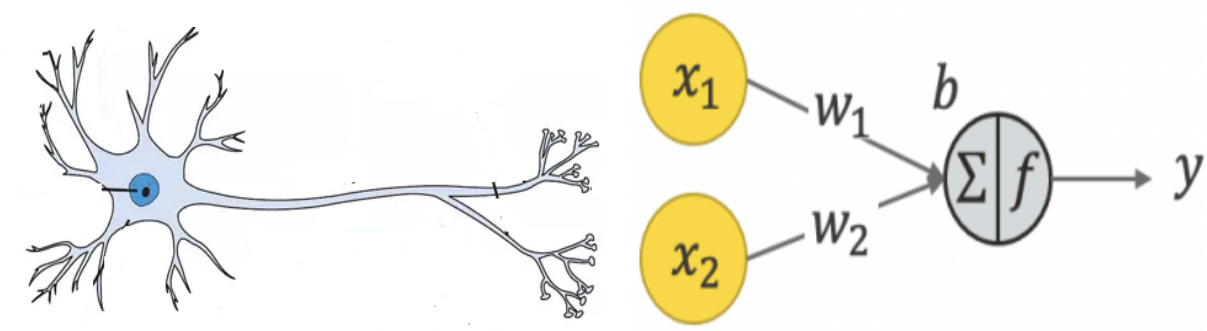
Vào thời điểm **những năm 1990 và đầu những năm 2000**, mạng nơ-ron dần được thay thế bởi support vector machines - SVM. SVMs có ưu điểm là bài toán tối ưu để tìm các tham số của nó là một bài toán lồi - có nhiều các thuật toán tối ưu hiệu quả giúp tìm nghiệm của nó.

Năm **2006**, xuất hiện Deep Belief Network, Geoffrey Hinton, Ruslan Salakhutdinov, Osindero và The xuất bản bài báo "A fast learning algorithm for deep belief nets", trong đó họ xếp chồng nhiều RBM lại với nhau thành từng lớp và gọi chúng là Deep Belief Networks. Quá trình đào tạo hiệu quả hơn nhiều đối với lượng dữ liệu

lớn. Cũng kể từ năm này, mạng nơron với nhiều lớp ẩn được đổi tên thành học sâu (deep learning).

II.2.6. Artificial Neural Network

Qua việc tìm hiểu lược sử học sâu, ta dễ dàng nhận thấy mạng nơ-ron nhân tạo (Artificial Neural Network) là cốt lõi của hệ thống học sâu. Mạng nơ-ron nhân tạo được xây dựng dựa trên các nguyên tắc về cấu tạo và hoạt động của các tế bào nơ-ron trong não bộ con người. Nơ-ron là đơn vị cơ bản cấu tạo hệ thống thần kinh và là phần quan trọng nhất của não. Não chúng ta gồm khoảng 10 triệu nơ-ron và mỗi nơ-ron liên kết với 10.000 nơ-ron khác.



H8. Nơ-ron nhân tạo mô phỏng nơ-ron sinh học

Mạng nơ-ron nhân tạo bao gồm nhiều lớp (layer) khác nhau, độ "sâu" của mạng được thể hiện ở số lượng lớp trong mạng đó. Trong mỗi lớp có các nút mạng (node, nơ-ron) và được liên kết với các lớp liền kề khác. Trong học máy, nơ-ron được định nghĩa là một hàm toán học nhận vào một hay nhiều giá trị đầu vào được nhân với các trọng số (weight). Trọng số là giá trị thể hiện của mỗi kết nối giữa hai nút mạng, trọng số này càng lớn thì kết nối này càng quan trọng đối với mạng.

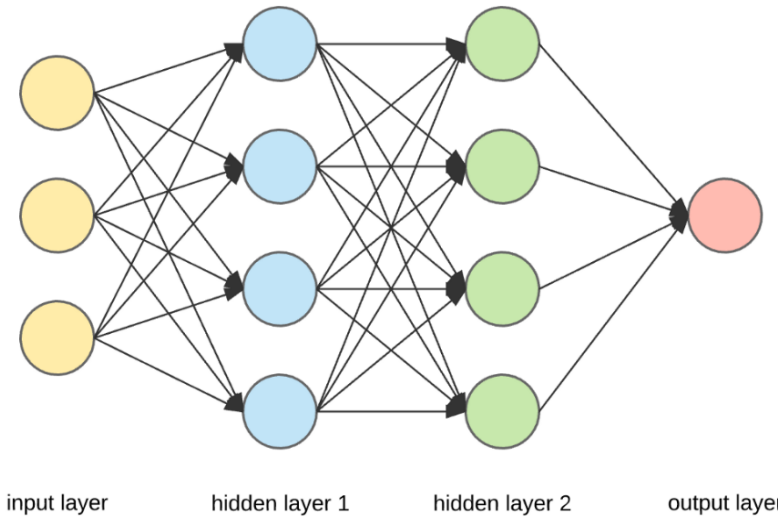
Nơ-ron được định nghĩa với công thức sau :

$$\boldsymbol{y = f(\sum x_i w_i + b)}$$

Trong đó :

- x_i là giá trị đại diện cho dữ liệu đầu vào (input) của nơ-ron.
- w_i là trọng số (weight).
- b là một tham số bổ sung cho nơ-ron, dùng để điều chỉnh giá trị đầu ra của nơ-ron (bias).
- Hàm f được gọi là hàm kích hoạt (activation function).

Lớp đầu tiên của mạng nơ-ron nhân tạo (input layer) nhận thông tin đầu vào từ nguồn bên ngoài và chuyển nó đến lớp ẩn (hidden layer), một mạng có thể có một hay nhiều lớp ẩn. Mỗi nơ-ron ở lớp ẩn nhận thông tin từ những nơ-ron ở lớp ngay trước đó, tính toán tổng trọng số, sau đó chuyển tiếp cho các nơ-ron ở lớp tiếp theo. Mỗi nơ-ron sẽ có một hàm kích có nhiệm vụ chuẩn hóa đầu ra từ nơ-ron này. Cuối cùng, kết quả sẽ được trả về ở layer cuối cùng (output layer). Trong thực tế, nhiều bài toán yêu cầu đầu ra gồm nhiều hơn một giá trị. Do đó ta thường xây dựng những mạng nơ-ron với lớp đầu ra gồm nhiều nút mạng.

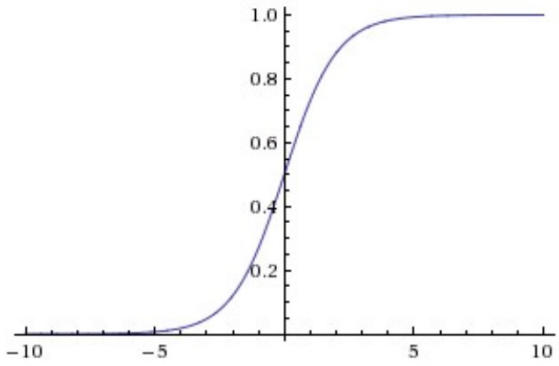


H9. Ví dụ cấu trúc của một ANN có 2 lớp ẩn với đầu ra có một giá trị

Quá trình trên được gọi là Feedforward (truyền thẳng). Dữ liệu từ tập mẫu huấn luyện được đưa vào mạng nơ-ron nhân tạo, sau đó kết quả đầu ra được so sánh với kết quả thực tế. Sự sai khác sẽ được sử dụng để điều chỉnh trọng số của các nút mạng. Việc thay đổi trọng số cho phù hợp sẽ được tính toán thông qua thuật toán Backpropagation (lan truyền ngược).

II.2.7. Một số hàm kích hoạt thông dụng

a. Hàm sigmoid



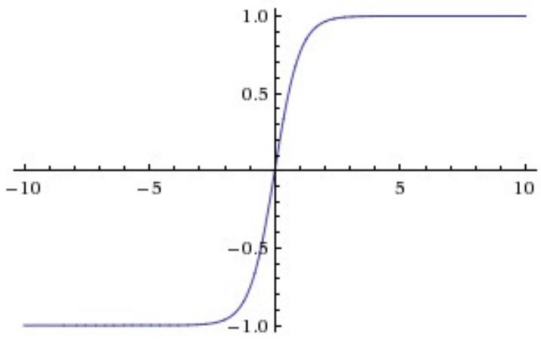
H10. Đồ thị hàm sigmoid (https://cs231n.github.io/neural-networks-1/)

Công thức toán học của hàm sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Hàm số nhận đầu vào là một số thực sau đó "nén" lại thành đầu ra có giá trị trong khoảng [0,1] và có thể được xem như xác suất trong một số bài toán. Tuy nhiên hiện nay hàm này rất ít được dùng vì xuất hiện một số nhược điểm. Ví dụ như hàm sigmoid làm bão hòa và triệt tiêu gradient: khi đầu vào có giá trị tuyệt đối rất lớn, gradient của hàm số này sẽ rất gần với 0. Điều này đồng nghĩa với việc các hệ số tương ứng với node đang xét sẽ gần như không được cập nhật.

b. Hàm tanh



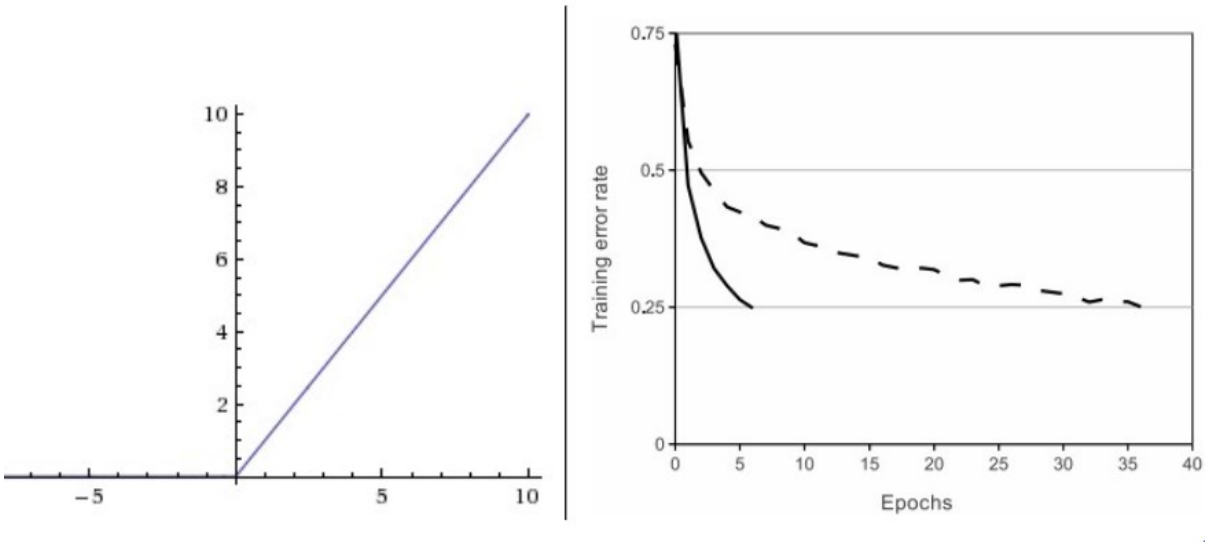
H11. Đồ thị hàm tanh (https://cs231n.github.io/neural-networks-1/)

Công thức toán học của hàm tanh:

$$\tanh(x) = 2\sigma(2x) - 1$$

Hàm tanh có nhược điểm tương tự như hàm sigmoid, xuất hiện vấn đề bão hòa và triệt tiêu đạo hàm khi đầu vào là rất lớn.

c. Hàm ReLU



H12. Đồ thị hàm ReLU và tốc độ hội tụ khi so sánh với hàm tanh. (https://cs231n.github.io/neural-networks-1/)

Hàm ReLU trở nên phổ biến trong những năm gần đây, công thức toán học của hàm:

$$f(x) = max(0, x)$$

Ưu điểm của hàm ReLU:

- Theo Krizhevsky et al., hàm ReLU được chứng minh có tốc độ tăng đáng kể cho việc huấn luyện các mạng học sâu. Điều này do dạng tuyến tính và không bão hòa của hàm.
- So với các hàm tanh/sigmoid liên quan đến các phép toán tốn kém (hàm mũ,...), hàm ReLU được triển khai bằng cách đơn giản.

Nhược điểm của hàm ReLU:

- Với các nút mạng có giá trị nhỏ hơn 0, sau khi áp dụng hàm ReLU sẽ thành 0, không còn ý nghĩa với lớp tiếp theo và các hệ số tương ứng từ nút đó cũng không được cập nhật với gradient descent. Hiện tượng này được gọi là "dying ReLU".

d. Hàm Leaky ReLU

Hàm Leaky ReLU là một biến thể của hàm ReLU để khắc phục vấn đề "dying ReLU". Công thức toán học của hàm Leaky ReLU:

$$f(x) = \begin{cases} 0.01x \text{ với } x < 0 \\ x \text{ với } x \geq 0 \end{cases}$$

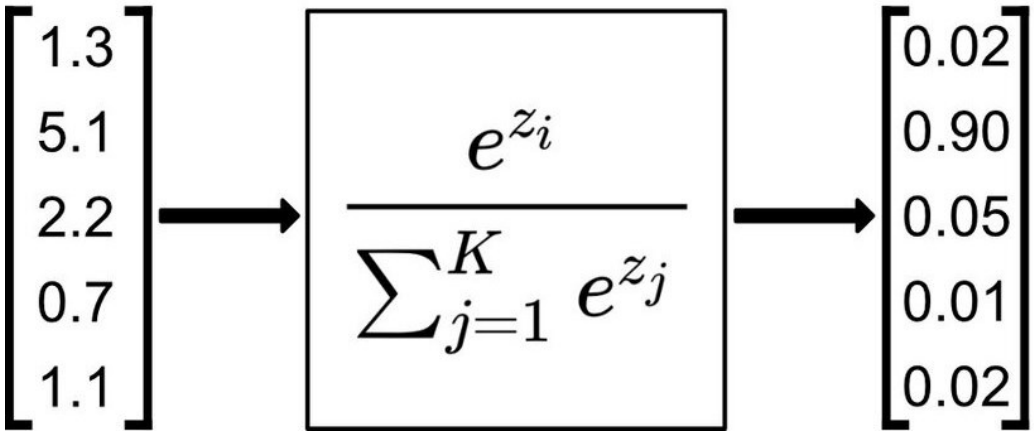
Hàm Leaky ReLU thừa hưởng ưu điểm của hàm ReLU và giải quyết được nhược điểm của hàm ReLU bằng cách để một độ dốc nhỏ cho các giá trị âm thay vì để giá trị là 0.

e. Hàm Softmax

Softmax hay là hàm trung bình mũ, tính toán xác suất xuất hiện của một sự kiện. Nói tổng quát, hàm trung bình mũ sẽ đưa ra khả năng hiện diện của một lớp trong tổng số toàn bộ các lớp của bài toán. Sau khi tính toán, tổng xác suất sẽ bằng 1, mỗi xác suất nằm trong khoảng (0;1].

Hàm softmax là một hàm kích hoạt thường được sử dụng trong các lớp đầu ra của mạng nơ-ron, đặc biệt là trong bài toán phân loại nhiều lớp. Hàm này chuyển đầu ra của một mạng nơ-ron thành một phân phối xác suất qua các lớp.

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$



H13. Ví dụ áp dụng hàm softmax

(https://botpenguin.com/glossary/softmax-function)

II.2.8. Backpropagation

Backpropagation (lan truyền ngược) là một thuật toán quan trọng trong quá trình huấn luyện mạng nơ-ron, đặc biệt là trong các mô hình sâu. Quá trình này giúp mô hình "học" từ dữ liệu bằng cách điều chỉnh trọng số của các liên kết giữa các nơ-ron.

Lan truyền ngược bắt đầu sau khi đưa dữ liệu qua mạng nơ-ron để tạo ra dự đoán. Khi đó, giá trị dự đoán được so sánh với giá trị thực tế thông qua một hàm mất mát. Đạo hàm của hàm mất mát đối với từng trọng số trong mạng được tính toán sử dụng quy tắc chuỗi (chain rule). Đạo hàm của hàm mất mát được truyền ngược qua mạng, giúp đánh giá cách mỗi trọng số ảnh hưởng đến sự sai lệch giữa đầu ra dự đoán và giá trị thực tế.

Sau đó, thuật toán sử dụng các giá trị đạo hàm này để cập nhật trọng số của mạng, thường thông qua một phương pháp tối ưu hóa như gradient descent. Quá trình lặp này tiếp tục cho đến khi mạng đạt được sự hội tụ, tức là giảm thiểu mức độ sai lệch giữa dự đoán và thực tế đến mức tối ưu.

Backpropagation là một trong những cơ sở của học sâu và đã đóng góp đáng kể vào thành công của nhiều ứng dụng hiện đại như nhận diện hình ảnh, ngôn ngữ tự nhiên, và nhiều lĩnh vực khác.

II.3. Mạng nơ-ron tích chập

II.3.1. Phép tính tích chập (Convolutional operation)

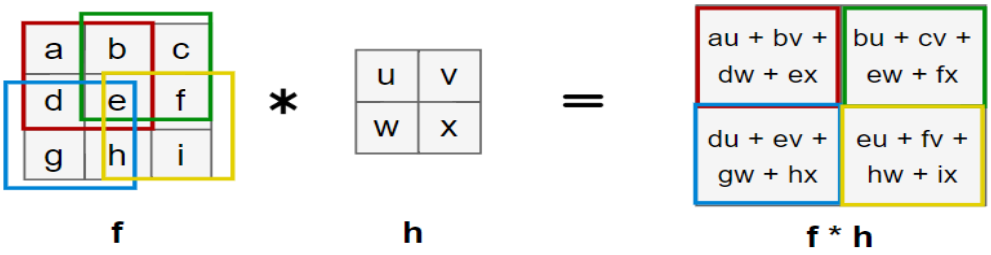
a. Công thức

Tích chập là phép tính toán học áp dụng trên hai hàm, với kết quả đầu ra là hàm thứ ba thường là sự sửa đổi của một trong các hàm ban đầu. Trong xử lý ảnh, phép tính chập có thể được biểu diễn như sau:

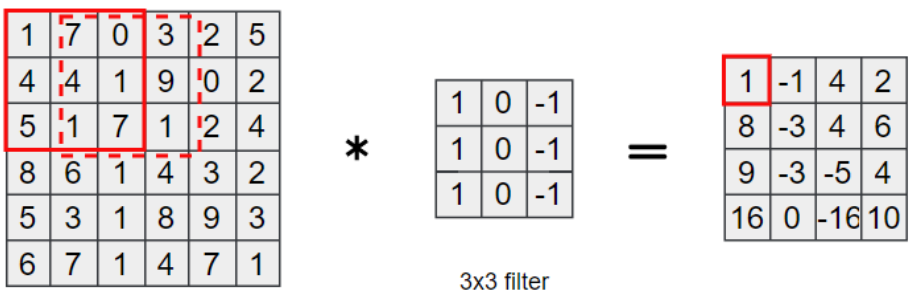
$$(f * h)(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x-i, y-j) \times h(i,j)$$

trong đó f là ma trận ảnh đầu vào có kích thước *n x n*, h là ma trận tích chập (mask/ filter) và k là bậc của ma trận h. Ma trận tích chập thường là ma trận vuông với bậc lẻ. Ma trận đầu ra sẽ có kích thước *(n – k + 1) x (n – k + 1)*.

Ví dụ 1: ma trận f có kích thước 3x3, ma trận h có kích thước 2x2, ma trận đầu ra $f * h$ có kích thước 2x2.



Ví dụ 2: ma trận f có kích thước 6x6, ma trận h có kích thước 3x3, ma trận đầu ra có kích thước 4x4.



b. Padding

Với công thức như trên, khi ta thực hiện phép tính tích chập thì đều thu được ma trận đầu ra có kích thước nhỏ hơn ma trận đầu vào. Muốn ma trận đầu ra có kích thước bằng với kích thước ma trận đầu vào, ta sẽ cần thêm phần đệm (padding), bằng cách thêm các giá trị 0 ở viền ngoài của ma trận đầu vào. Padding = p nghĩa là thêm p vector 0 vào mỗi phía của ma trận.

0	0	0	0	0	0	0	0
0	1	7	0	3	2	5	0
0	4	4	1	9	0	2	0
0	5	1	7	1	2	4	0
0	8	6	1	4	3	2	0
0	5	3	1	8	9	3	0
0	6	7	1	4	7	1	0
0	0	0	0	0	0	0	0

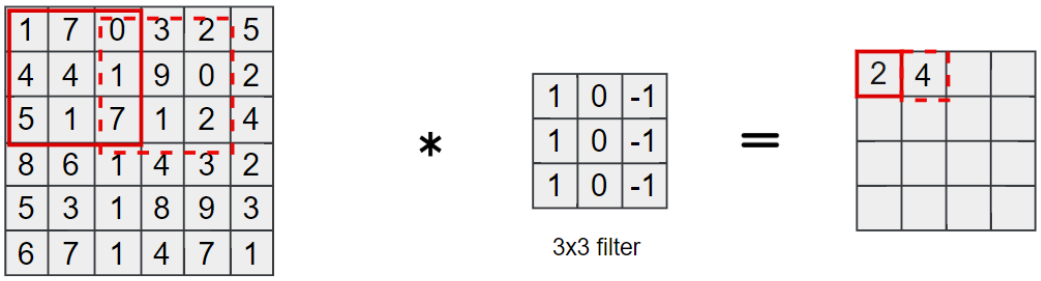
Ví dụ: padding = 1

Phần đệm có tác dụng chống mất thông tin của pixel ở viền ảnh khi thực hiện phép tính tích chập. Ma trận đầu ra có kích thước $(n - k + 1 + 2p) \times (n - k + 1 + 2p)$.

c. Strided

Trong phép tính tích chập thông thường, bước nhảy (stride) thường được thiết lập là 1, có nghĩa là mỗi lần áp dụng phép tính toán, ma trận trượt qua chỉ một bước. Ngược lại, khi chúng ta sử dụng strided convolution với một stride được đặt là s, điều này sẽ điều chỉnh cách ma trận trượt di chuyển, thực hiện mỗi lần tính toán với một bước nhảy là s. Điều này có nghĩa là, thay vì trượt qua mỗi pixel một cách liên tiếp, với stride là s, chúng ta sẽ bỏ qua s-1 pixel giữa các lần tính toán. Điều này có thể giảm kích thước của ma trận kết quả và giảm độ phức tạp của mô hình.

Stride trong phép tính tích chập là một yếu tố quan trọng có thể ảnh hưởng đến độ chính xác của mô hình và cũng có thể được điều chỉnh tùy thuộc vào yêu cầu cụ thể của bài toán.

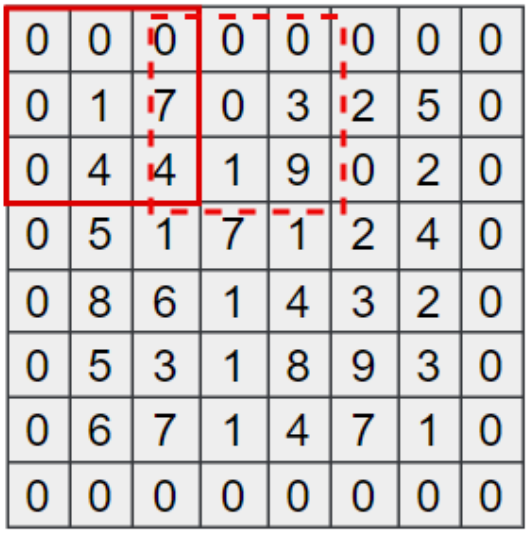


Ví dụ với strided s = 2

Sử dụng stride giúp tăng tốc tính toán do giảm số lượng phép tính cần thực hiện, bên cạnh đó phương pháp này giúp giảm chiều của dữ liệu mà vẫn giữ được các đặc trưng quan trọng. Ma trận đầu ra có kích thước $(\frac{n-k}{s} + 1)\ x\ (\frac{n-k}{s} + 1)$.

d. Kết hợp padding và stride

Thực tế khi sử dụng phép tích chập để xây dựng mạng nơ – ron tích chập, ta cần kết hợp cả hai phương pháp padding và stride để tận dụng được hết các lợi ích mà nó mang lại.

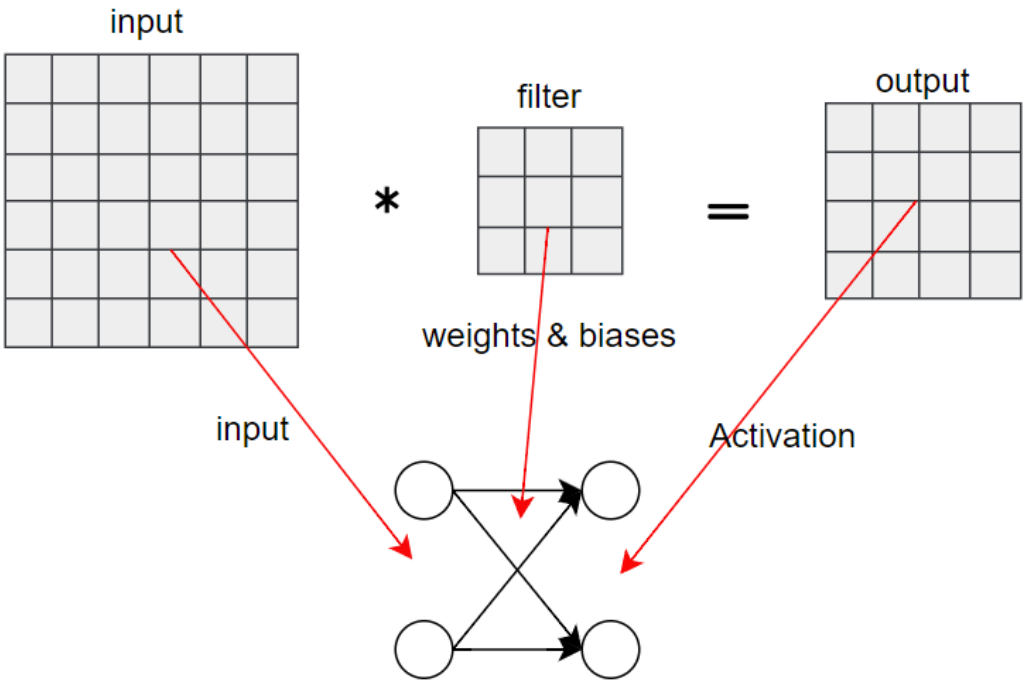


Ví dụ với p = 1, s = 2

Ma trận đầu ra sẽ có kích thước $(\frac{n-k+2p}{s}+1)\,x\,(\frac{n-k+2p}{s}+1)$. Trong trường hợp $(n-k+2p)$ không chia hết cho s, ta sẽ làm tròn xuống.

e. Sự tương đồng so với mạng nơ – ron nhân tạo

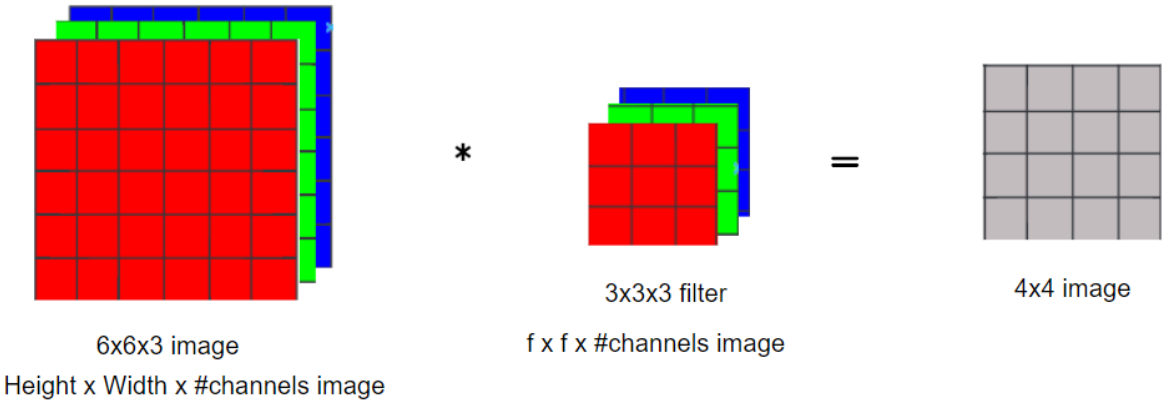
Sự tương đồng giữa các thành phần của phép tích chập so với các thành phần trong mạng nơ – ron nhân tạo được thể hiện trong sơ đồ dưới đây:



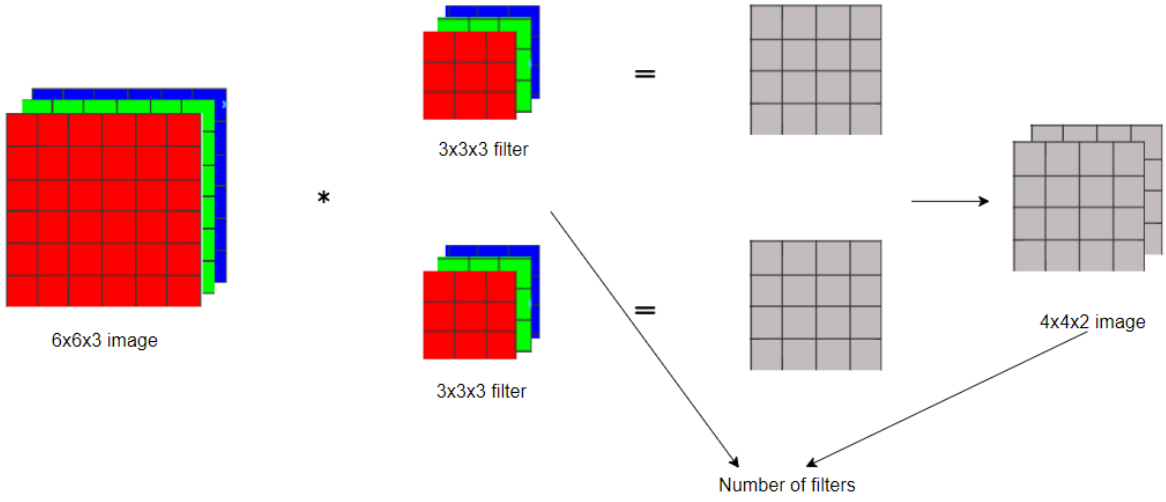
II.3.2. Phép tích chập cho ảnh màu

Ở những ví dụ trên, ta đang xét đầu vào là một ma trận tương ứng với một bức ảnh xám. Tuy nhiên, hiện nay ta cần làm việc nhiều hơn với các bức ảnh màu.

Theo phần II.1 đã xem xét ở trên, với một bức ảnh màu được biểu diễn theo hệ màu RGB ta sẽ cần 3 channel tương ứng với 3 ma trận. Biểu diễn một phép tích chập cơ bản như sau:



Filter cũng phải có chiều sâu bằng với chiều sâu của ảnh đầu vào. Ví dụ trên ta chỉ sử dụng một filter nên kết quả output vẫn là ảnh 2 chiều vì phép tích chập không được triển khai trên chiều sâu. Tuy nhiên như vậy sẽ dẫn đến đầu vào của phép tích chập tiếp theo chỉ là ảnh 2 chiều. Để bức ảnh đầu ra cuối cùng có chiều sâu, ta đi tính toán phép tích chập với nhiều filter. Một bức ảnh thực tế có nhiều góc cạnh đặc trưng cần phát hiện, do đó, ta sẽ sử dụng nhiều filter, mỗi filter có các giá trị khác nhau.



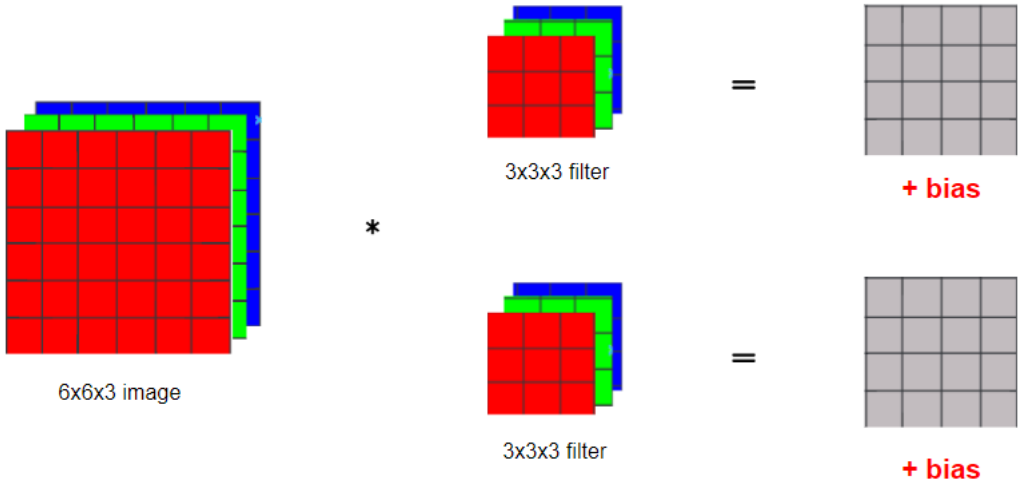
Ma trận đầu ra sẽ có kích thước $(n - f + 1)\ x\ (n - f + 1)\ x\ n'_c$. Trong đó n'_c là số lượng filter. Nếu sử dụng phương pháp padding hoặc strided thì kích thước ma trận có sự thay đổi tương ứng như đã xem xét ở những phần trên.

II.3.3. Các loại lớp (layer) trong mạng nơ – ron tích chập

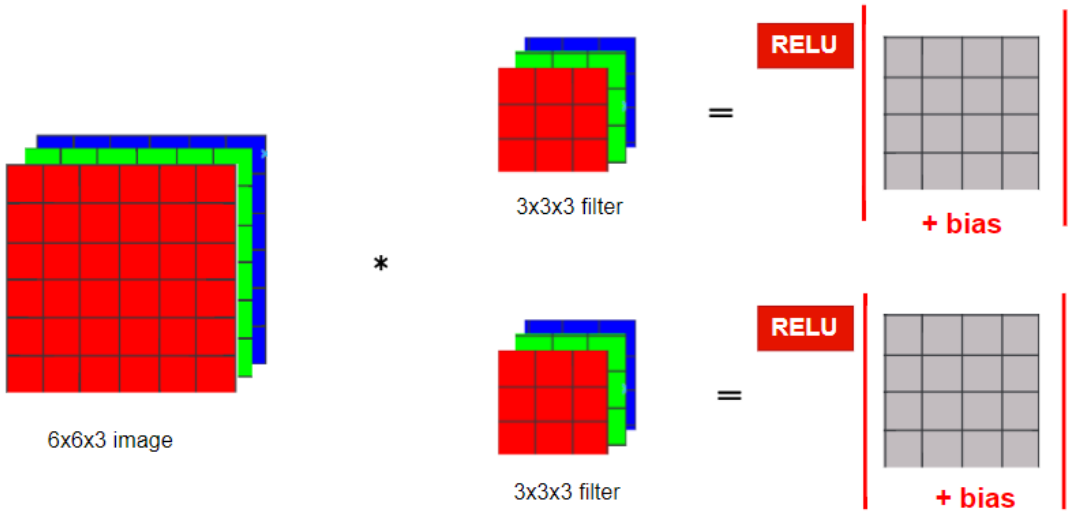
Trong mạng nơ – ron tích chập, đầu ra của lớp trước sẽ thành đầu vào của lớp tiếp theo. Ta đi tìm hiểu các loại lớp được sử dụng để xây dựng mạng.

a. Lớp tích chập cơ bản (Convolution layer)

Sau khi thực hiện phép tích chập với một ảnh đầu vào, để hoàn thiện một lớp tích chập ta cần thêm các bias và tính toán kết quả qua các hàm kích hoạt.



H14. Sau khi thực hiện tích chập sẽ cộng thêm bias



H15. Sau đó, áp dụng hàm kích hoạt (ví dụ với hàm ReLU)

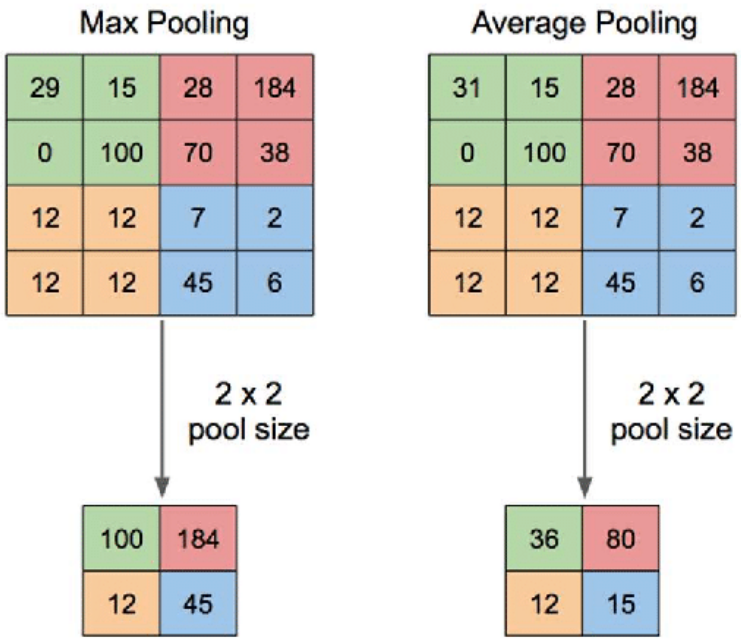
Để đưa các ma trận hoặc các tensor qua hàm kích hoạt, ta tính toán với từng giá trị của ô trong ma trận, tensor rồi ghi lại kết quả vào đúng ô đó.

b. Lớp tổng hợp (Pooling layer)

Lớp tổng hợp thường được dùng xen giữa các lớp tích chập nhằm giảm kích thước dữ liệu đầu vào nhưng vẫn giữ được các thuộc tính quan trọng. Kích thước dữ liệu giảm giúp ích rất nhiều trong việc tăng tốc độ tính toán của mô hình.

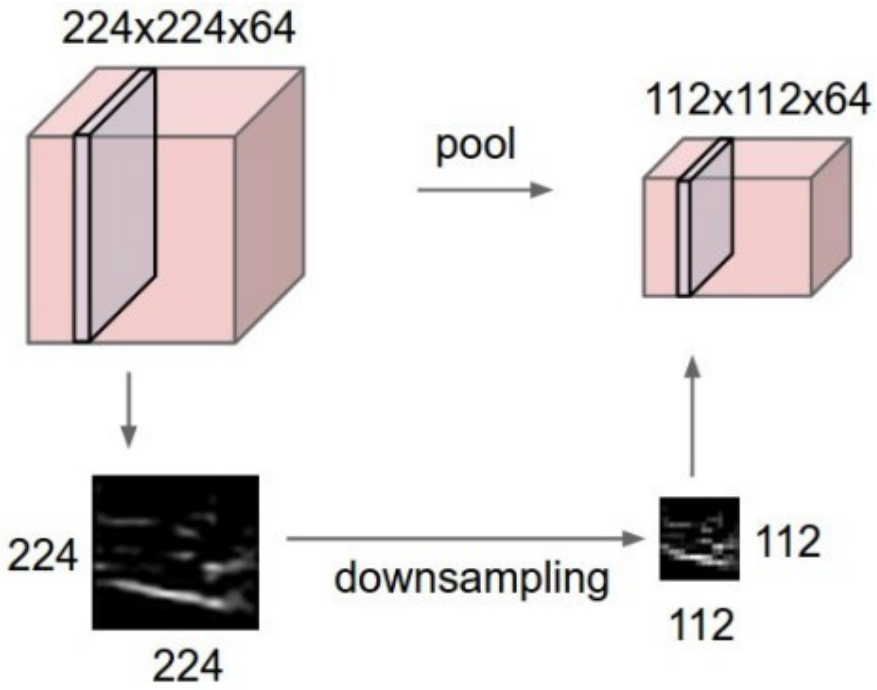
Hai loại lớp tổng hợp được sử dụng phổ biến nhất là Max pooling và Average pooling.

Giả sử ta có một pooling kích thước *k x k.* Đầu vào của pooling layer có kích thước n_H *x* n_W *x* n_C *(chiều dài x chiều rộng x chiều sâu),* ta tách tensor ra làm n_C ma trận kích thước n_H *x* n_W. Với mỗi ma trận, trên vùng kích thước *k x k* ta tìm giá trị lớn nhất hoặc giá trị trung bình của dữ liệu (tương ứng với Max pooling và Average pooling). Quy tắc về padding và strided được áp dụng như phép tính tích chập trên ảnh.



H16. Ví dụ cơ bản về 2 loại pooling layer (sử dụng bước nhảy s = 1)

Max pooling được sử dụng thường xuyên hơn trong mạng nơ – ron tích chập, và thực tế thường lựa chọn pooling layer với kích thước *2 x 2,* bước nhảy *s = 2,* padding p = 0. Khi đó đầu ra sẽ có chiều dài và chiều rộng giảm đi một nửa, chiều sâu được giữ nguyên.



H17. Sử dụng pooling layer 2x2

(http://cs231n.github.io/convolutional-networks/)

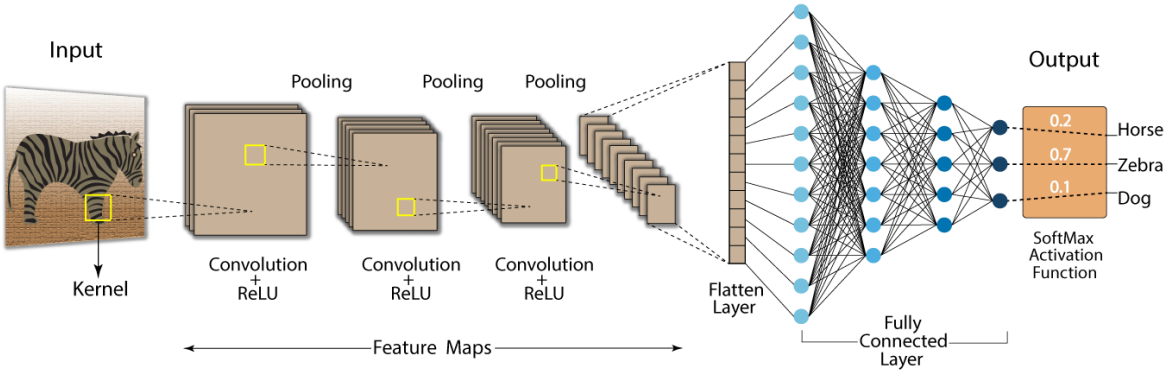
c. Làm phẳng (Flatten layer)

Sau khi thông tin được trích xuất từ các lớp tích chập, chúng ta cần làm phẳng (flatten) nó thành một vector một chiều. Flatten layer thực hiện công việc này. Việc sử dụng Flatten layer chính là cách để chuyển đổi từ mạng nơ – ron tích chập sang mạng nơ – ron nhân tạo và cho phép áp dụng các kiến trúc mạng nơ-ron tiêu biểu để thực hiện các nhiệm vụ phân loại hoặc dự đoán.

d. Lớp kết nối hoàn toàn (Fully connected layer)

Sau khi được làm phẳng, vector này sẽ được đưa vào các lớp fully connected để thực hiện quyết định cuối cùng. Các lớp Dense thường được sử dụng ở cuối mạng để thực hiện phân loại. Mỗi node trong lớp fully connected kết nối với tất cả các node trong lớp trước đó, do đó nó được gọi là fully connected layer.

Tensor đầu ra của layer cuối cùng, kích thước $n_H \; x \; n_W \; x \; n_C$ sẽ được chuyển về 1 vector kích thước bằng tích $(n_H \; x \; n_W \; x \; n_C)$. Sau đó ta dùng các fully connected layer để kết hợp các đặc điểm của ảnh để có được đầu ra của mô hình.



H18. Ví dụ về một mô hình mạng nơ – ron tích chập dự đoạn hình ảnh động vật (https://www.analyticsvidhya.com/blog/2022/03/basics-of-cnn-in-deep-learning/)

CHƯƠNG III. XÂY DỰNG VÀ KIỂM THỬ

III.1. Thông tin tập dữ liệu huấn luyện

Để xây dựng một mô hình học máy, học sâu hoàn chỉnh, thành phần không thể thiếu đó là tập dữ liệu.

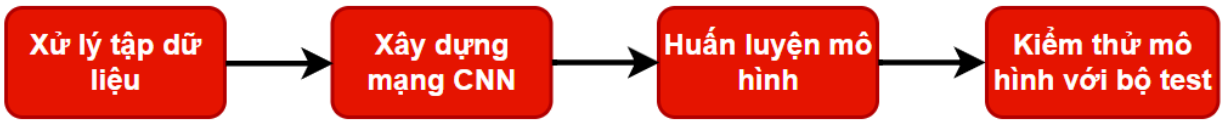
Về dữ liệu, em sử dụng bộ dữ liệu biển báo giao thông German Traffic Sign (GTSRB) có sẵn trên Kaggle – nền tảng trực tuyến cho cộng đồng Machine Learning và Khoa học dữ liệu. Bộ dữ liệu được chia sẵn thành ba 3 thư mục: meta, train, test. Thư mục train bao gồm 43 thư mục con ứng với 43 loại biển báo khác nhau với gần 40.000 ảnh. Thư mục test với hơn 12.000 ảnh đa dạng biển báo, riêng biệt với ảnh ở thư mục train để thử nghiệm mô hình. Các hình ảnh có điều kiện ánh sáng và hình nền ảnh khác nhau.

H19. Một số hình ảnh trong bộ dữ liệu GTSRB

III.2. Xây dựng mô hình

Sau khi tìm hiểu lý thuyết tổng quan như phần trên của báo cáo, em thấy mô hình mạng nơ – ron tích chập rất phù hợp để giải quyết bài toán đặt ra của đề tài là phân loại biển báo giao thông từ các bức ảnh chụp.

Để xây dựng mô hình phân loại biển báo giao thông, ta sẽ đi thực hiện các bước như sau:



III.2.1. Xử lý tập dữ liệu

Tập dữ liệu được lưu trữ dưới thư mục GTSRB trong project.

Trước tiên, đọc ảnh từ thư mục train, tiền xử lý ảnh bằng cách thiết lập kích thước của tất cả ảnh thành *30 x 30 x 3* và chuyển đổi ảnh thành mảng NumPy. Tiếp theo, lưu mảng các ảnh trong danh sách 'data', lưu nhãn tương ứng của ảnh trong danh sách 'labels', nhãn được lưu theo số thứ tự của tập [0, 42]. Đoạn mã:

```
data = []
labels = []
cur_path = os.getcwd()

#Xu ly tap du lieu
for i in range(43):
    path = os.path.join(cur_path, 'GTSRB\\train', str(i))
    images = os.listdir(path)

    for x in images:
        try:
            image = Image.open(path + '\\' + x)
            image = image.resize((30, 30))
            image = np.array(image)
            data.append(image)
            labels.append(i)
        except:
            print("Lỗi không load được ảnh")
data = np.array(data)
labels = np.array(labels)
```

Sau đó, phân chia tập dữ liệu thành 2 tập dữ liệu huấn luyện và dữ liệu kiểm thử với tỉ lệ tương ứng 80% – 20%.

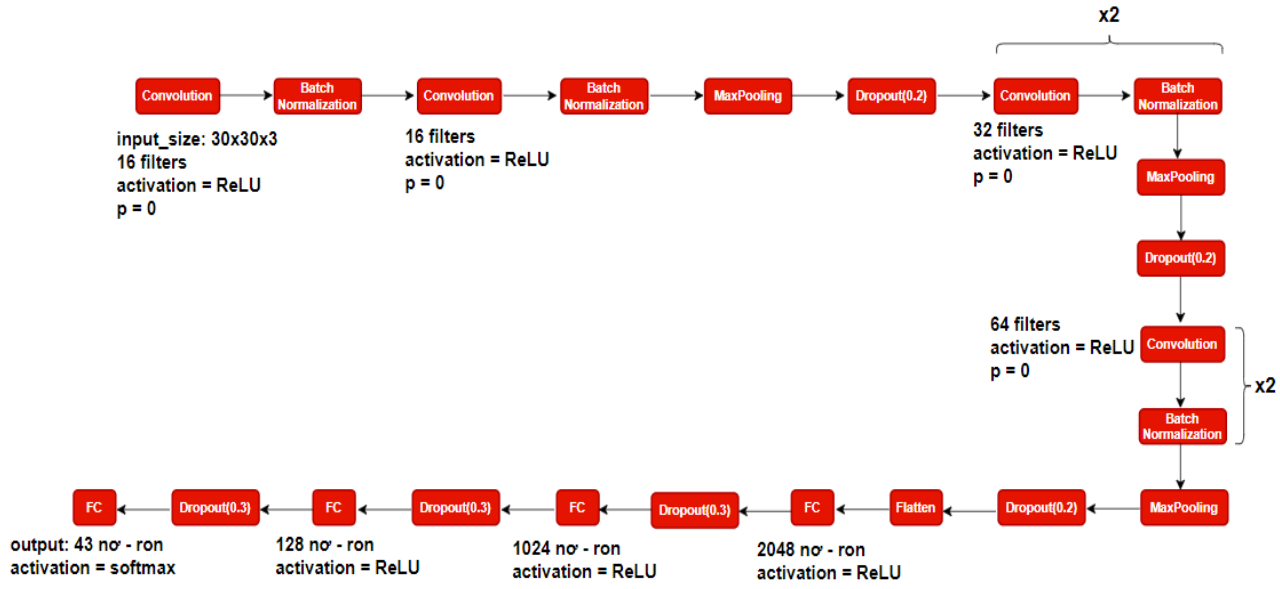
Cuối cùng, mã hóa các nhãn trong y_train và y_test theo phương pháp One – hot.

```
#Phan chia du lieu 80-20
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

#Ma hoa cac nhan
y_train = to_categorical(y_train, num_classes: 43)
y_test = to_categorical(y_test, num_classes: 43)
```

III.2.2. Xây dựng mạng nơ – ron tích chập

Qua thực nghiệm và kinh nghiệm cá nhân, em xây dựng mạng nơ – ron tích chập như sau:



III.2.3. Huấn luyện mô hình

Mô hình có sử dụng thêm dropout để tránh quá mức. Xây dựng mô hình với đoạn mã sau:

```
#Xay dung mo hinh
filter_size = (3,3)
pool_size = (2,2)

model = Sequential()
model.add(Conv2D( filters: 16, filter_size, activation='relu', input_shape=X_train.shape[1:], padding='same'))
model.add(BatchNormalization())
model.add(Conv2D( filters: 16, filter_size, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(rate=0.2))
model.add(Conv2D( filters: 32, filter_size, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D( filters: 32, filter_size, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(rate=0.2))
model.add(Conv2D( filters: 64, filter_size, activation='relu', padding='same'))
model.add(BatchNormalization())
```

```
model.add(Conv2D( filters: 64, filter_size, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(rate=0.2))
model.add(Flatten())
model.add(Dense( units: 2048, activation='relu'))
model.add(Dropout(rate=0.3))
model.add(Dense( units: 1024, activation='relu'))
model.add(Dropout(rate=0.3))
model.add(Dense( units: 128, activation='relu'))
model.add(Dropout(rate=0.3))
model.add(Dense( units: 43, activation='softmax'))

#compile mo hinh
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Sau đó, thực hiện huấn luyện mô hình và lưu lại mô hình dưới tên "traffic_sign_20207644.h5" để dùng sau này.

```
#compile mo hinh
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

#huan luyen mo hinh
history = model.fit(X_train, y_train, batch_size=16, epochs=10, validation_data=(X_test, y_test))
model.save("traffic_sign_20207644.h5")
```

Quá trình huấn luyện diễn ra:

```
1961/1961 [==============================] - 121s 59ms/step - loss: 1.7344 - accuracy: 0.4728 - val_loss: 0.4855 - val_accuracy: 0.8322
Epoch 2/10
1961/1961 [==============================] - 118s 60ms/step - loss: 0.5221 - accuracy: 0.8373 - val_loss: 0.1858 - val_accuracy: 0.9450
Epoch 3/10
1961/1961 [==============================] - 116s 59ms/step - loss: 0.3139 - accuracy: 0.9161 - val_loss: 0.1179 - val_accuracy: 0.9642
Epoch 4/10
1961/1961 [==============================] - 116s 59ms/step - loss: 0.2497 - accuracy: 0.9365 - val_loss: 0.0779 - val_accuracy: 0.9799
Epoch 5/10
1961/1961 [==============================] - 117s 59ms/step - loss: 0.2084 - accuracy: 0.9490 - val_loss: 0.0733 - val_accuracy: 0.9809
Epoch 6/10
1961/1961 [==============================] - 115s 58ms/step - loss: 0.1669 - accuracy: 0.9601 - val_loss: 0.0456 - val_accuracy: 0.9878
Epoch 7/10
1961/1961 [==============================] - 115s 58ms/step - loss: 0.1779 - accuracy: 0.9620 - val_loss: 0.0715 - val_accuracy: 0.9858
Epoch 8/10
1961/1961 [==============================] - 113s 58ms/step - loss: 0.1382 - accuracy: 0.9704 - val_loss: 0.0370 - val_accuracy: 0.9923
Epoch 9/10
1961/1961 [==============================] - 123s 63ms/step - loss: 0.1449 - accuracy: 0.9689 - val_loss: 0.0402 - val_accuracy: 0.9906
Epoch 10/10
1961/1961 [==============================] - 121s 62ms/step - loss: 0.1341 - accuracy: 0.9728 - val_loss: 0.0230 - val_accuracy: 0.9944
```

III.2.4. Kiểm thử mô hình

Ta kiểm thử mô hình với tập dữ liệu test

```python
y_test = pd.read_csv('GTSRB/Test.csv')

labels = y_test["ClassId"].values
imgs = y_test["Path"].values
model = load_model('traffic_sign_20207644.h5')
data=[]

for img in imgs:
    image = Image.open('GTSRB/' + img)
    image = image.resize((30,30))
    data.append(np.array(image))

X_test=np.array(data)
pred = np.argmax(model.predict(X_test), axis=-1)

#Tinh accuracy
from sklearn.metrics import accuracy_score
print(accuracy_score(labels, pred))
```

Được kết quả tỉ lệ mô hình phân loại đúng hơn 96%:

```
395/395 [==============================] - 7s 16ms/step
0.9619160728424386
```

III.3. Demo

Em sẽ xây dựng một ứng dụng với giao diện đơn giản để tiện cho việc demo mô hình. Đầu tiên, sử dụng mô hình đã huấn luyện và lưu lại ở trên:

```
#dung model da xay dung
model = load_model('traffic_sign_20207644.h5')
```

Tạo dictionary với 43 nhãn là tên biển báo giao thông:

```
#cac nhan
classes = { 1:'Tốc độ tối đa cho phép (20km/h)',
            2:'Tốc độ tối đa cho phép (30km/h)',
            3:'Tốc độ tối đa cho phép (50km/h)',
            4:'Tốc độ tối đa cho phép (60km/h)',
            5:'Tốc độ tối đa cho phép (70km/h)',
            6:'Tốc độ tối đa cho phép (80km/h)',
            7:'Hết hạn chế tốc độ tối đa (80km/h)',
            8:'Tốc độ tối đa cho phép (100km/h)',
            9:'Tốc độ tối đa cho phép (120km/h)',
            10:'Cấm vượt',
            11:'Cấm ô tô tải vượt',
            12:'Giao nhau Với đường không ưu tiên',
            13:'Đường ưu tiên',
            14:'Giao nhau với đường ưu tiên',
            15:'Dừng lại',
            16:'Đường cấm',
            17:'Cấm xe tải',
            18:'Cấm đi ngược chiều',
            19:'Nguy hiểm khác',
            20:'Chỗ ngoặt nguy hiểm vòng bên trái',
            21:'Chỗ ngoặt nguy hiểm vòng bên phải',
            22:'Nhiều chỗ ngoặt nguy hiểm liên tiếp',
            23:'Đường không bằng phẳng',
```

Xây dựng giao diện cơ bản:

```
#xay dung giao dien
top=tk.Tk()
top.geometry('800x600')
top.title('Traffic sign recognition')
top.configure(background='#CDCDCD')
label=Label(top,background='#CDCDCD', font=('arial',15,'bold'))
sign_image = Label(top)
```

```
def show_classify_button(file_path):
    classify_b=Button(top,text="Phân loại",command=lambda: classify(file_path),padx=10,pady=5)
    classify_b.configure(background='#364156', foreground='white',font=('arial',10,'bold'))
    classify_b.place(relx=0.79,rely=0.46)
```

```
upload=Button(top,text="Thêm ảnh",command=upload_image,padx=10,pady=5)
upload.configure(background='#364156', foreground='white',font=('arial',10,'bold'))
upload.pack(side=BOTTOM,pady=50)
sign_image.pack(side=BOTTOM,expand=True)
label.pack(side=BOTTOM,expand=True)
heading = Label(top, text="Phân loại biển báo giao thông",pady=20, font=('arial',20,'bold'))
heading.configure(background='#CDCDCD',foreground='#364156')
heading.pack()
top.mainloop()
```

Xây dựng hàm sử dụng mô hình đã xây dựng để trả về kết quả tên biển báo ứng với nhãn được trả về:

```
def classify(file_path):
    global label_packed
    image = Image.open(file_path)
    image = image.resize((30,30))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)
    pred = model.predict_classes([image])[0]
    sign = classes[pred+1]
    print(sign)
    label.configure(foreground='#011638', text=sign)
```

Hàm cho phép người dùng thêm ảnh để thực hiện phân loại:

```
def upload_image():
    try:
        file_path=filedialog.askopenfilename()
        uploaded=Image.open(file_path)
        uploaded.thumbnail(((top.winfo_width()/2.25),(top.winfo_height()/2.25)))
        im=ImageTk.PhotoImage(uploaded)
        sign_image.configure(image=im)
        sign_image.image=im
        label.configure(text='')
        show_classify_button(file_path)
    except:
        pass
```

Giao diện có dạng như sau:

KẾT LUẬN

Qua thời gian nghiên cứu thực hiện, em đã nắm bắt được lý thuyết tổng quan về học máy, học sâu, các khái niệm về mạng nơ – ron nhân tạo và xây dựng được mô hình phân loại biển báo giao thông với độ chính xác lên tới 96%.

Tuy nhiên, do thời gian và hiểu biết của em còn hạn chế nên hiện nay chương trình chỉ dừng lại ở mức phân loại được với hình ảnh biển báo giao thông, hướng phát triển tiếp của đề tài là thêm tính năng phát hiện biển báo giao thông từ ảnh và video để áp dụng vào các hệ thống xe tự lái thực tế. Đồng thời, đồ án chắc chắn cũng không tránh khỏi những thiếu sót, nên em rất mong nhận được ý kiến đóng góp chỉ dẫn từ thầy.

Em xin chân thành cảm ơn thầy!

TÀI LIỆU THAM KHẢO

[1] Wikipedia

[2] CS231n: Deep Learning for Computer Vision (Stanford - Spring 2023)

[3] datalya.com

[4] slideshare.net

[5] analyticsvidhya.com

…