ĐẠI HỌC BÁCH KHOA HÀ NỘI
KHOA VẬT LÝ KỸ THUẬT

BÁO CÁO DỰ ÁN CUỐI KÌ

MÔN: PH3460-LẬP TRÌNH ỨNG DỤNG

CHỦ ĐỀ: XÂY DỰNG CHƯƠNG TRÌNH PHÁT HIỆN CHÁY RỪNG TỪ HÌNH ẢNH VỆ TINH.

Danh sách thành viên:

Bùi Công Huy	20210439
Nguyễn Hải Triều	20217445

Gảng viên hướng dẫn: ThS.Bùi Ngọc Hà

Hà Nội, tháng 7 năm 2025

Mục Lục

Mục lục ảnh

Phần 1. Xây dựng mô hình

I. Thư viện Tensorflow

1. Tổng quan

Với sự bùng nổ của lĩnh vực Trí Tuệ Nhân Tạo – A.I. trong thập kỷ vừa qua, machine learning và deep learning rõ ràng cũng phát triển theo cùng. Và ở thời điểm hiện tại, TensorFlow chính là thư viện mã nguồn mở cho machine learning nổi tiếng nhất thế giới, được phát triển bởi các nhà nghiên cứu từ Google. Việc hỗ trợ mạnh mẽ các phép toán học để tính toán trong machine learning và deep learning đã giúp việc tiếp cận các bài toán trở nên đơn giản, nhanh chóng và tiện lợi hơn nhiều.

Các hàm được dựng sẵn trong thư viện cho từng bài toán cho phép TensorFlow xây dựng được nhiều neural network. Nó còn cho phép tính toán song song trên nhiều máy tính khác nhau, thậm chí trên nhiều CPU, GPU trong cùng 1 máy hay tạo ra các dataflow graph – đồ thị luồng dữ liệu để dựng nên các model. Trong lĩnh vực A.I. này, nắm rõ những điều cơ bản của TensorFlow thực sự rất quan trọng.

Được viết bằng C++ và thao tác interface bằng Python nên phần performance của TensorFlow cực kỳ tốt. Đối tượng sử dụng nó cũng đa dạng không kém: từ các nhà nghiên cứu, nhà khoa học dữ liệu và dĩ nhiên không thể thiếu các lập trình viên.

2. Cách thức hoạt động

2.1. Kiến trúc

Kiến trúc TensorFlow hoạt động được chia thành 3 phần:

– Tiền xử lý dữ liệu

– Dựng model

– Train và ước tính model

2.2 Nguyên lý

TensorFlow cho phép các lập trình viên tạo ra dataflow graph, cấu trúc mô tả làm thế nào dữ liệu có thể di chuyển qua 1 biểu đồ, hay 1 sê-ri các node đang xử lý. Mỗi node trong đồ thị đại diện 1 operation toán học, và mỗi kết nối hay edge giữa các node là 1 mảng dữ liệu đa chiều, hay còn được gọi là ‘tensor’.

TensorFlow cung cấp tất cả những điều này cho lập trình viên theo phương thức của ngôn ngữ Python. Vì Python khá dễ học và làm việc, ngoài ra còn cung cấp nhiều cách tiện lợi để ta hiểu được làm thế nào các high-level abstractions có thể kết hợp cùng nhau. Node và tensor trong TensorFlow là các đối tượng Python, và các ứng dụng TensorFlow bản thân chúng cũng là các ứng dụng Python.

Các operation toán học thực sự thì không được thi hành bằng Python. Các thư viện biến đổi có sẵn thông qua TensorFlow được viết bằng các binary C++ hiệu suất cao. Python chỉ điều hướng lưu lượng giữa các phần và cung cấp các high-level abstraction lập trình để nối chúng lại với nhau.

3. Thành phần

3.1. Tensor

Tên của TensorFlow được đưa ra trực tiếp là nhờ vào framework cốt lõi của nó: Tensor. Trong TensorFlow, tất cả các tính toán đều liên quan tới các tensor. 1 tensor là 1 vector hay ma trận của n-chiều không gian đại diện cho tất cả loại dữ liệu. Tất cả giá trị trong 1 tensor chứa đựng loại dữ liệu giống hệt nhau với 1 shape đã biết (hoặc đã biết 1 phần). Shape của dữ liệu chính là chiều của ma trận hay mảng.

1 tensor có thể được bắt nguồn từ dữ liệu input hay kết quả của 1 tính toán. Trong TensorFlow, tất cả các hoạt động được tiến hành bên trong 1 graph – biểu đồ. Biểu đồ là 1 tập hợp tính toán được diễn ra liên tiếp. Mỗi operation được gọi là 1 op node (operation node) và được kết nối với nhau.

Biểu đồ phát thảo các op và kết nối giữa các node. Tuy nhiên, nó không hiển thị các giá trị. Phần edge của các node chính là tensor, 1 cách để nhập operation với dữ liệu.

3.2. Graph

TensorFlow sử dụng framework dạng biểu đồ. Biểu đồ tập hợp và mô tả tất cả các chuỗi tính toán được thực hiện trong quá trình training. Biểu đồ cũng mang rất nhiều lợi thế:

- Nó được làm ra để chạy trên nhiều CPU hay GPU, ngay cả các hệ điều hành trên thiết bị điện thoại.
- Tính di động của biểu đồ cho phép bảo toàn các tính toán để bạn sử dụng ngay hay sau đó. Biểu đồ có thể được lưu lại để thực thi trong tương lai.
- Tất cả tính toán trong biểu đồ được thực hiện bằng cách kết nối các tensor lại với nhau. 1 tensor có 1 node và 1 edge. Node mang operation toán học và sản xuất các output ở đầu cuối. Các edge giải thích mối quan hệ input/output giữa các node.

4. Lợi ích

Lợi ích dễ thấy nhưng quan trọng nhất mà TensorFlow cung cấp cho việc lập trình machine learning chính là abstraction. Thay vì phải đối phó với những tình huống rườm rà từ việc thực hiện triển khai các thuật toán, hay tìm ra cách hợp lý để chuyển output của 1 chức năng sang input của 1 chức năng khác, giờ đây bạn có thể tập trung vào phần logic tổng thể của 1 ứng dụng hơn. TensorFlow sẽ chăm sóc phần còn lại thay cho bạn.

Ngoài ra TensorFlow còn ung cấp các tiện ích bổ sung cho các lập trình viên cần debug cũng như giúp tự suy xét các ứng dụng TensorFlow. Chế độ eager execution cho phép đánh giá và sửa đổi từng operation của biểu đồ 1 cách riêng biệt và minh bạch, thay vì phải dựng toàn bộ biểu đồ dưới dạng 1 đối tượng độc lập vốn khá mơ hồ hay phải đánh giá chung tổng thể. Cuối cùng, 1 tính năng khá độc đáo của TensorFlow là TensorBoard. TensorBoard cho phép bạn quan sát 1 cách trực quan những gì TensorFlow đang làm.

TensorFlow còn có nhiều cải tiến từ sự hậu thuẫn từ các ekíp thương mại hạng A tại Google. Google không những tiếp lửa cho tiến độ nhanh chóng cho sự phát triển đằng sau dự án, mà còn tạo ra nhiều phục vụ độc đáo xung quanh TensorFlow để nó dễ dàng deploy và sử dụng: như silicon TPU mình đã nói ở trên để tăng tốc hiệu suất đám mây Google, 1 online hub cho việc chia sẻ các model được tạo với framework, sự hiện diện của in-browser và gần gũi với mobile của framework, và nhiều hơn thế nữa…

Lưu ý: Trong 1 số công việc training, vài chi tiết về việc triển khai của TensorFlow làm cho nó khó có thể quyết định được hoàn toàn kết quả training model. Đôi khi 1 model được train trên 1 hệ thống này sẽ có thay đổi 1 chút so với 1 model được train trên hệ thống khác, ngay cả khi chúng được cung cấp dữ liệu như nhau. Các nguyên nhân cho điều này cũng xê xích hay 1 số hành vi khi không được xác định khi sử dụng GPU. Điều này nói rằng, các vấn đề đó có thể giải quyết được, và đôi ngũ của TensorFlow cũng đang xem xét việc kiểm soát nhiều hơn để ảnh hưởng đến tính quyết định trong quy trình làm việc.

5. API Keras trong Thư viện Tensorflow

5.1. Tổng quan

– Keras là API cấp cao của platform Tensorflow. Nó cung cấp một giao diện có thể tiếp cận, có hiệu quả cao để giải quyết các vấn đề về Machine Learning (ML), tập trung vào modern deep learning.
– Keras bao gồm từng bước của quy trình công việc machine learning, từ xử lý dữ liệu đến điều chỉnh siêu phân tích đến triển khai. Nó được phát triển với trọng tâm là cho phép thử nghiệm nhanh.
– Với Keras, chúng ta có quyền truy cập đầy đủ vào khả năng mở rộng và khả năng đa nền tảng của TensorFlow.
– Có thể chạy Keras trên POD TPU hoặc các cụm GPU lớn và có thể xuất các mô hình Keras để chạy trong trình duyệt hoặc trên thiết bị di động cũng có thể phục vụ các mô hình Keras thông qua API Web.
– Keras được thiết kế để giảm tải nhận thức bằng cách đạt được các mục tiêu sau:
+ Cung cấp các giao diện đơn giản, nhất quán.

+ Giảm thiểu số lượng hành động cần thiết cho các trường hợp sử dụng phổ biến.
+ Cung cấp thông báo lỗi rõ ràng, có thể hành động.
+ Thực hiện theo nguyên tắc tiết lộ tiến bộ về sự phức tạp: Thật dễ dàng để bắt đầu và có thể hoàn thành quy trình công việc nâng cao bằng cách học.
+ Giúp bạn viết mã ngắn gọn, có thể đọc được.
– Tích hợp hoàn hảo với TensorFlow: Từ TensorFlow 2.0, Keras đã được tích hợp sẵn vào TensorFlow, giúp khai thác sức mạnh của TensorFlow một cách dễ dàng thông qua Keras.

5.2. Submodule Preprocessing trong API Keras

Đây là một module con trong keras, chứa các công cụ và hàm phục vụ cho việc tiền xử lý dữ liệu (preprocessing), đặc biệt là dữ liệu ảnh và chuỗi.

5.3. Submodule Image trong Submodule Preprocessing

Là một module con bên trong preprocessing, chuyên dùng cho các thao tác tiền xử lý dữ liệu ảnh như chuẩn hóa, tăng cường dữ liệu, chuyển đổi định dạng,...

5.4. Class ImageDataGenerator trong API Keras

ImageDataGenerator là một tiện ích mạnh mẽ giúp tải, tiền xử lý và tăng cường dữ liệu ảnh theo thời gian thực trong quá trình huấn luyện hoặc dự đoán mô hình.

Nó cho phép dễ dàng áp dụng các phép biến đổi như chuẩn hóa, xoay, lật, phóng to/thu nhỏ ảnh, giúp cải thiện khả năng tổng quát và độ bền của mô hình học máy.

Khi sử dụng ImageDataGenerator, có thể quản lý hiệu quả các bộ dữ liệu lớn và thực hiện tăng cường dữ liệu ngay trong quá trình huấn luyện, giảm nhu cầu lưu trữ nhiều bản sao ảnh đã biến đổi trên ổ đĩa.

Điều này đặc biệt hữu ích trong các bài toán học sâu liên quan đến phân loại ảnh hoặc nhận diện đối tượng.

5.5. Phương thức flow_from_directory () trong Class ImageDataGenerator

flow_from_directory là một phương thức được cung cấp bởi lớp ImageDataGenerator của Keras trong TensorFlow.

Phương thức này dùng để tải và tiền xử lý ảnh trực tiếp từ một thư mục trên ổ đĩa một cách hiệu quả.

Thư mục này cần được tổ chức sao cho mỗi thư mục con đại diện cho một lớp khác nhau, và các ảnh của từng lớp sẽ được lưu trong các thư mục con tương ứng.

Khi gọi flow_from_directory, phương thức sẽ tự động gán nhãn dựa trên tên các thư mục con, thay đổi kích thước ảnh về kích thước mong muốn và tạo ra các lô (batch) ảnh cùng với nhãn để phục vụ cho quá trình huấn luyện hoặc kiểm tra.

Cách tiếp cận này giúp tiết kiệm bộ nhớ vì ảnh được tải theo từng lô thay vì tải toàn bộ cùng lúc, rất phù hợp khi làm việc với các bộ dữ liệu lớn.

Ngoài ra, phương thức này còn có thể áp dụng các phép tăng cường dữ liệu và tiền xử lý theo cấu hình của ImageDataGenerator.

flow_from_directory thường được sử dụng trong các dự án phân loại ảnh để đơn giản hóa quá trình chuẩn bị dữ liệu cho các mô hình học sâu.

5.6. Class Sequential model trong API Keras

5.6.1. Cấu trúc

– Sequential Class được thiết kế để xây dựng các mô hình theo từng lớp một cách tuần tự. Mỗi lớp trong mô hình Sequential được thêm vào theo thứ tự, tạo thành một chuỗi các lớp.

– Các lớp cơ bản trong Sequential Class bao gồm:

+ Dense: Lớp kết nối đầy đủ (fully connected layer).

+ Conv2D: Lớp tích chập 2D (dành cho xử lý hình ảnh).

+ LSTM: Lớp LSTM (dành cho xử lý chuỗi thời gian).

+ Dropout: Lớp dropout để ngăn chặn overfitting.

5.6.2. Cú pháp

Để tạo một mô hình Sequential, bạn sử dụng cú pháp sau:

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(64, activation='relu', input_shape=(784,)),
    Dense(10, activation='softmax')
])
```

Hình 1. Cú pháp mô hình Sequential

Trong ví dụ trên, chúng ta tạo một mô hình với hai lớp Dense. Lớp đầu tiên có 64 neuron và sử dụng hàm kích hoạt ReLU, trong khi lớp thứ hai có 10 neuron và sử dụng hàm kích hoạt softmax.

II. Pillow

1. Tổng quan

Python Pillow cho phép thao tác với hình ảnh và thực hiện các tác vụ xử lý hình ảnh cơ bản.

Là một folk của Python Imaging Library (PIL), Pillow hỗ trợ các định dạng hình ảnh như JPEG, PNG, và nhiều hơn nữa, cho phép đọc, chỉnh sửa và lưu hình ảnh.

Với Python Pillow, có thể cắt, thay đổi kích thước, xoay và áp dụng các bộ lọc cho hình ảnh, làm cho nó trở thành một công cụ đa năng để thao tác hình ảnh.

Pillow thường được sử dụng cho các nhiệm vụ xử lý hình ảnh cấp cao và công việc thăm dò.

Mặc dù không phải là thư viện nhanh nhất, nhưng nó cung cấp một đường cong học tập nhẹ nhàng và một bộ tính năng toàn diện cho nhu cầu xử lý hình ảnh cơ bản đến trung gian.

Có thể nâng cao khả năng của nó bằng cách tích hợp nó với Numpy cho các thao tác cấp độ pixel và tạo hình ảnh động.

Thư viện Python Pillow là một folk của một thư viện cũ tên là PIL.

PIL là viết tắt của Python Imaging Library và đó là thư viện ban đầu cho phép Python đối phó với hình ảnh.

PIL đã bị ngưng vào năm 2011 và chỉ hỗ trợ Python 2.

Để sử dụng mô tả riêng của các nhà phát triển, Pillow là folk thân thiện từ PIL giữ cho thư viện tồn tại và bao gồm hỗ trợ cho Python 3.

Có nhiều hơn một module trong Python để xử lý hình ảnh và thực hiện xử lý hình ảnh.

Nếu muốn xử lý hình ảnh trực tiếp bằng cách thao tác với các pixel của chúng, thì bạn có thể sử dụng Numpy và Scipy.

Các thư viện phổ biến khác để xử lý hình ảnh là OpenCV, Scikit-Image và Mahotas.

Một số thư viện này nhanh hơn và mạnh hơn Pillow.

Tuy nhiên, Pillow vẫn là một công cụ quan trọng để xử lý hình ảnh.

Nó cung cấp các tính năng xử lý hình ảnh tương tự như các tính năng được tìm thấy trong phần mềm xử lý hình ảnh như Photoshop.

Pillow thường là tùy chọn ưa thích cho các nhiệm vụ xử lý hình ảnh cấp cao không yêu cầu chuyên môn xử lý hình ảnh nâng cao hơn.

Nó cũng thường được sử dụng cho công việc khám phá khi xử lý hình ảnh.

Pillow cũng có lợi thế là được cộng đồng Python sử dụng rộng rãi và nó không có đường cong học tập dốc giống như một số thư viện xử lý hình ảnh khác.

2. Module ImageFile

Module ImageFile cung cấp các chức năng hỗ trợ cho hình ảnh mở và lưu các chức năng.

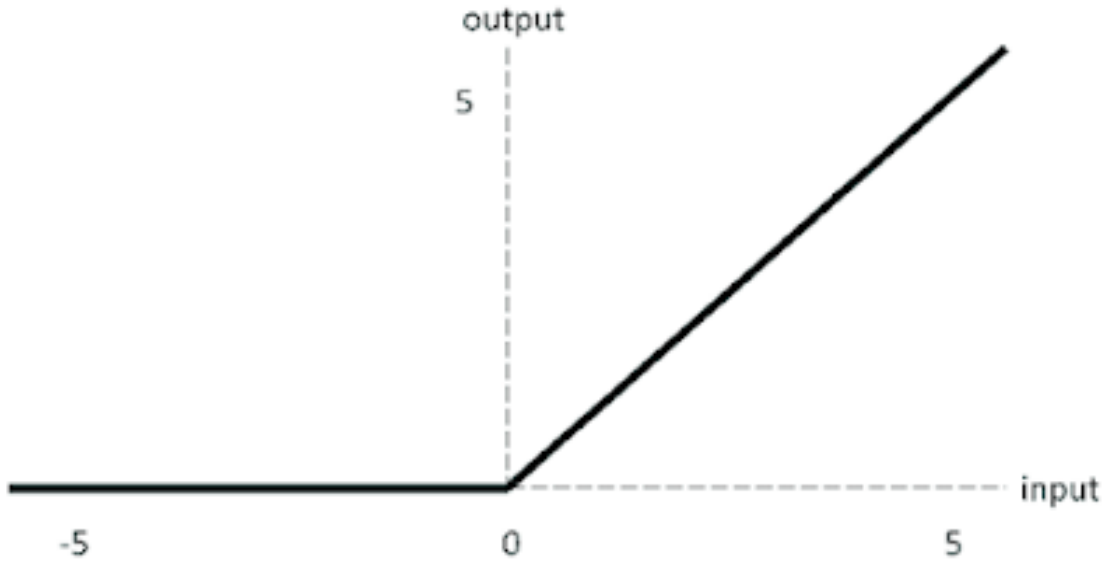
Module ImageFile cung cấp các chức năng hỗ trợ đọc và ghi tệp ảnh, đặc biệt hữu ích khi xử lý dữ liệu ảnh có thể bị thiếu hoặc bị hỏng một phần.

III. Hàm kích hoạt phi tuyến tính (nonlinear activation)

1. ReLu

Công thức hàm Relu như sau:

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



Hình 2. Đồ thị Relu

– Ưu điểm của hàm Relu:

+ Tính đơn giản của nó và nó đã được chứng minh là giúp tăng tốc quá trình training.

+ Không bị chặn như hàm Sigmoid hay Tanh nên nó không phải là nguyên nhân gây ra hiện tượng vanishing gradient.

– Nhược điểm:

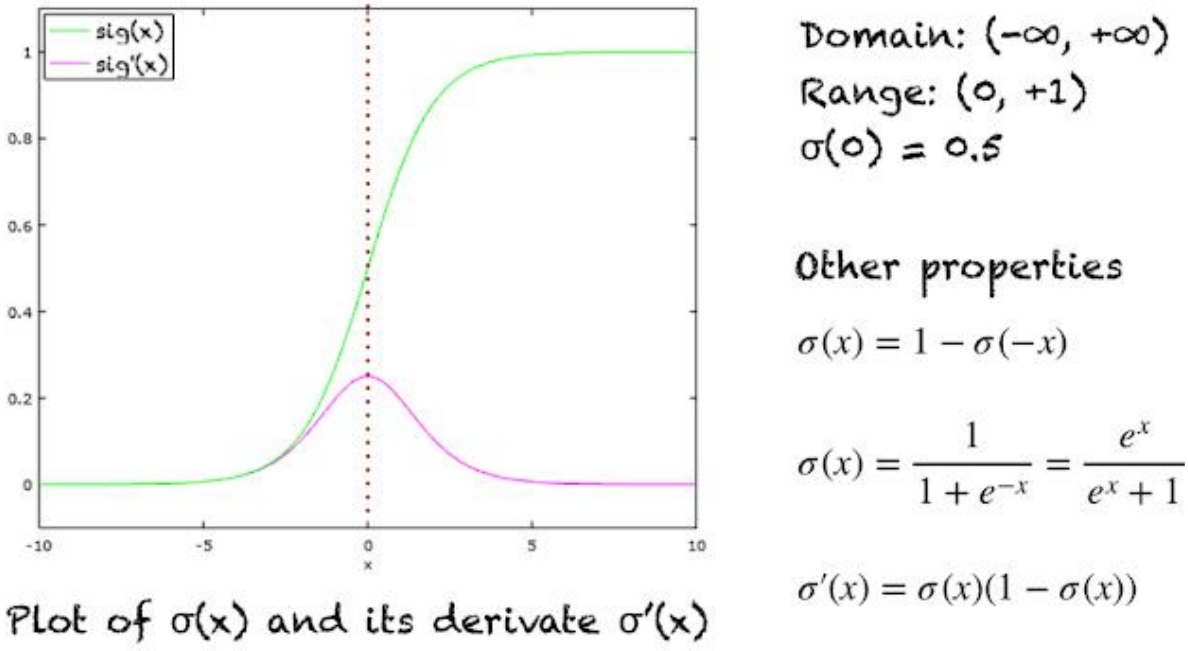
+ Tại những điểm có giá trị âm thì giá trị của Relu sẽ bằng 0 (dying relu) và theo lý thuyết nó sẽ không có đạo hàm tại các điểm 0 nhưng thực tế thì người ta thường bổ sung thêm đạo hàm của relu tại 0 bằng 0 và bằng thực nghiệm người ta cũng thấy rằng xác suất để input relu rơi vào điểm 0 là rất nhỏ.

+ Và do nó ko được chặn trên nên cũng có một nhược điểm là gây ra hiện tượng exploding gradient nhưng thường sẽ relu sẽ hoạt động tốt trong thực tế.

2. Sigmoid

Hàm sigmoid có dạng đường cong "S" khá đẹp. Đây là một hàm liên tục, khả vi và bị chặn trong khoảng (0, 1). Công thức của hàm sigmoid như sau:

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



Hình 3. Đồ thị Hàm Sigmoid

Đây là một hàm được ưa chuộng trong quá khứ với đặc điểm có tính đạo hàm tại mọi điểm nhưng hiện nay nó ít được sử dụng hơn do một số lí do như giá trị đạo hàm của nó bị chặn trong khoảng (0, 0.25) do đó nó dễ gây hiện tượng vanishing gradient.

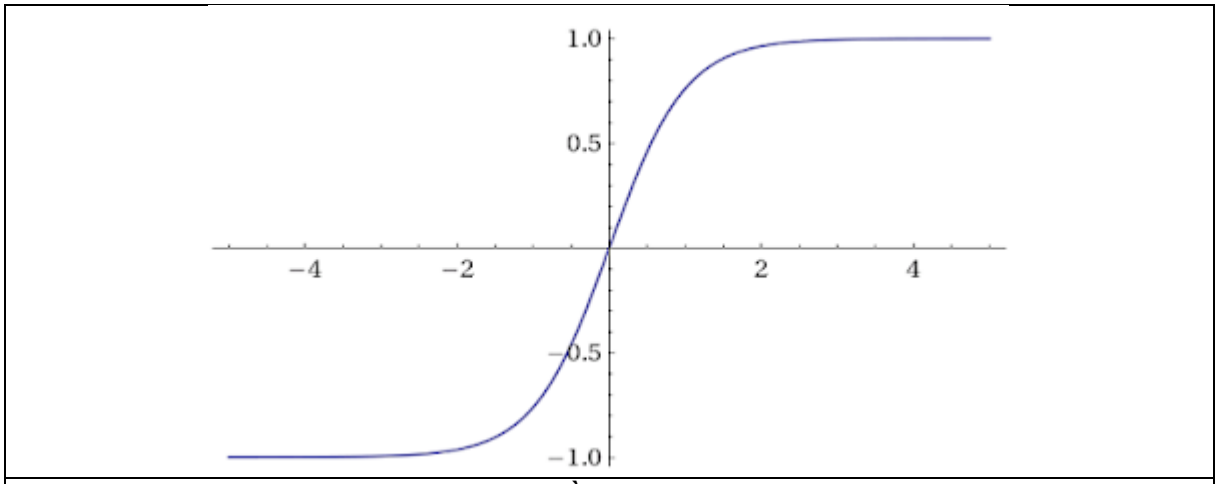
Ngoài ra việc sử dụng hàm mũ khiến cho việc tính toán trở nên lâu hơn.

Nói chung hàm sigmoid thường được sử dụng trong các bài toán hồi quy logic hoặc sử dụng trong các cơ chế attention như CBAM, SEblock, ...

3. Tanh

Hàm tanh thì có hình dáng tương tự như hàm sigmoid nhưng nó khác với hàm sigmoid là nó có tính đối xứng qua gốc tọa độ và cũng có các tính chất tương tự như hàm sigmoid. Công thức của hàm Tanh như sau:

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



Hình 4. Đồ thị Hàm Tanh

IV. Hàm mất mát (Binary Cross-Entropy Loss)

Cross-entropy là hàm loss được sử dụng mặc định cho bài toán phân lớp nhị phân.

Nó được thiết kế để sử dụng với bài toán phân loại nhị phân trong đó các giá trị mục tiêu nhận một trong 2 giá trị {0, 1}.

Về mặt toán học, nếu như MSE tính khoảng cách giữa 2 đại lượng số thì cross-entropy hiểu nôm na là phương pháp tính khoảng cách giữa 2 phân bố xác suất.

$$H(\mathbf{p}, \mathbf{q}) = \mathbf{E}_{\mathbf{p}}[-\log \mathbf{q}]$$

Với p và q là rời rạc (như y - nhãn thật sự và y^ - nhãn dự đoán) trong bài toán, công thức này được viết dưới dạng:

$$H(\mathbf{p}, \mathbf{q}) = -\sum_{i=1}^{C} p_i \log q_i \quad (1)$$

Trong đó C là số lượng các class cần phân lớp, trong bài toán binary classification thì C = 2.
Cross-entropy được cung cấp trong Keras bằng cách thiết lập tham số loss='binary_crossentropy 'khi compile mô hình.

```python
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
```

Hàm yêu cầu layer đầu ra được gồm 1 node và sử dụng kích hoạt 'sigmoid' để dự đoán xác suất cho đầu ra.

```python
model.add(Dense(1, activation='sigmoid'))
```

V. Thuật Toán tối ưu hóa (Optimizer)

1. Tổng quan

Về cơ bản, thuật toán tối ưu là cơ sở để xây dựng mô hình neural network với mục đích "học " được các features (hay pattern) của dữ liệu đầu vào, từ đó có thể tìm 1 cặp weights và bias phù hợp để tối ưu hóa model.

2. Các thuật toán tối ưu

2.1. Gradient Descent (GD)

Trong các bài toán tối ưu, chúng ta thường tìm giá trị nhỏ nhất của 1 hàm số nào đó, mà hàm số đạt giá trị nhỏ nhất khi đạo hàm bằng 0. Nhưng đối với các hàm số nhiều biến thì đạo hàm rất phức tạp, thậm chí là bất khả thi. Nên thay vào đó người ta tìm điểm gần với điểm cực tiểu nhất và xem đó là nghiệm bài toán. Gradient Descent dịch ra tiếng Việt là giảm dần độ dốc, nên hướng tiếp cận ở đây là chọn 1 nghiệm ngẫu nhiên cứ sau mỗi vòng lặp (hay epoch) thì cho nó tiến dần đến điểm cần tìm.

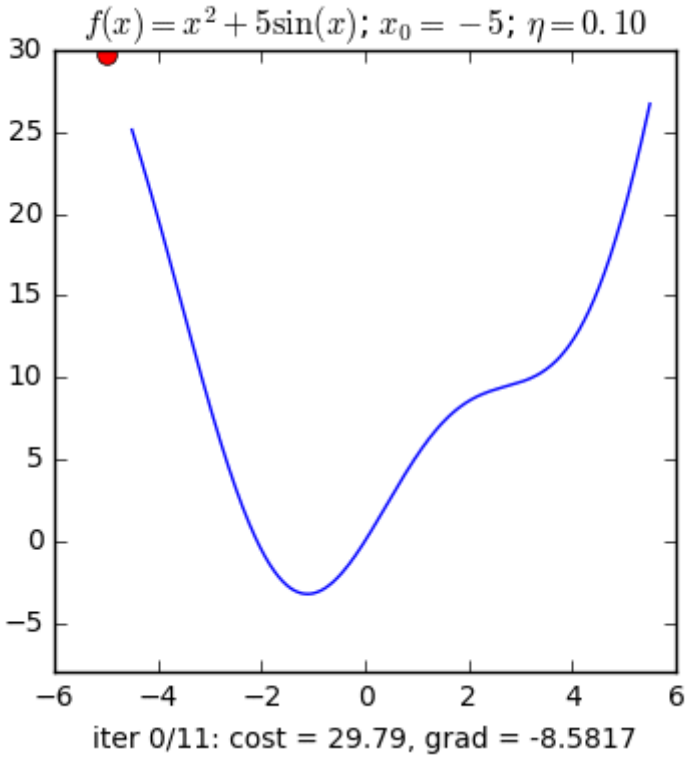
Công thức: xnew = xold - learningrate. gradient(x)

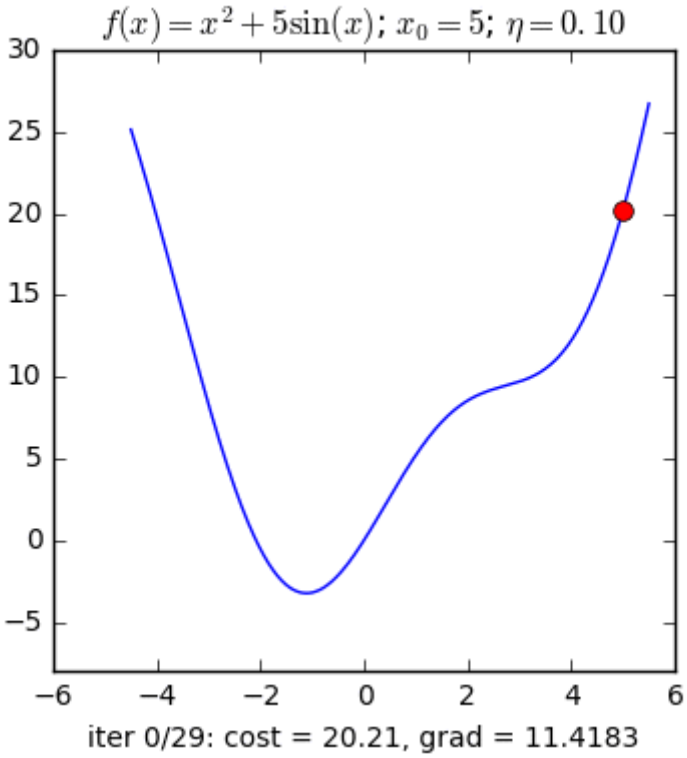
Công thức trên được xây dựng để cập nhật lại nghiệm sau mỗi vòng lặp. Dấu '-' trừ ở đây ám chỉ ngược hướng đạo hàm.

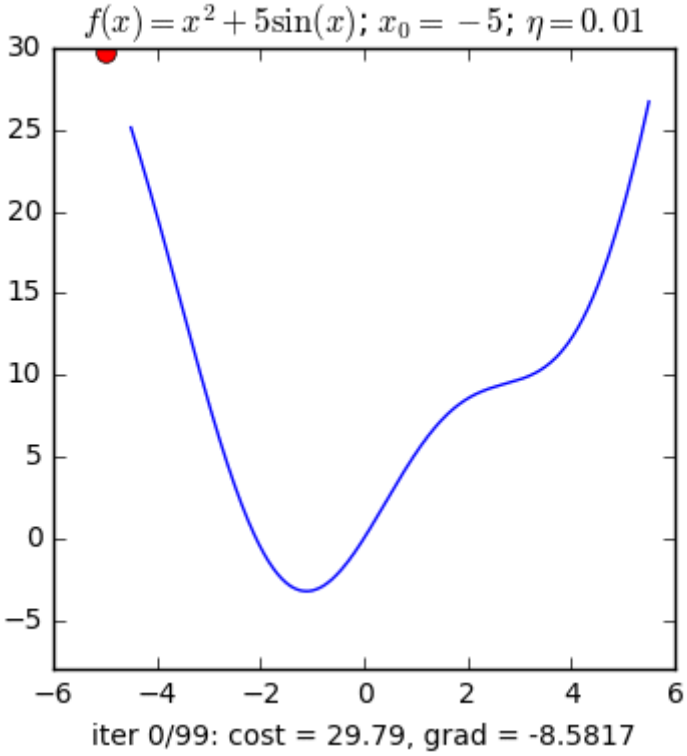
Ví dụ như đối với hàm f(x)= 2x +5sin(x) như hình dưới thì f'(x) =2x + 5cos(x) với x_old =-4 thì f'(-4) <0 => x_new > x_old nên nghiệm sẽ di chuyển về bên phải tiến gần tới điểm cực tiểu.

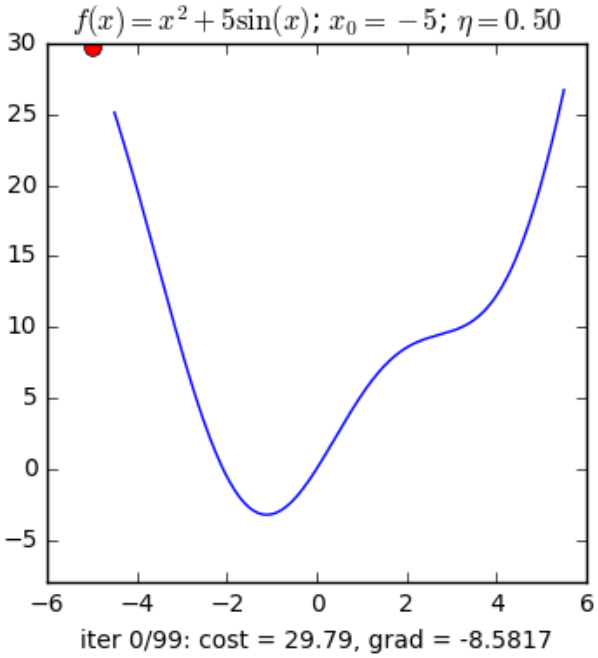
ngược lại với x_old =4 thì f'(4) >0 => x_new <x_old nên nghiệm sẽ di chuyển về bên trái tiến gần tới điểm cực tiểu.

2.1.1. Gradient cho hàm 1 biến



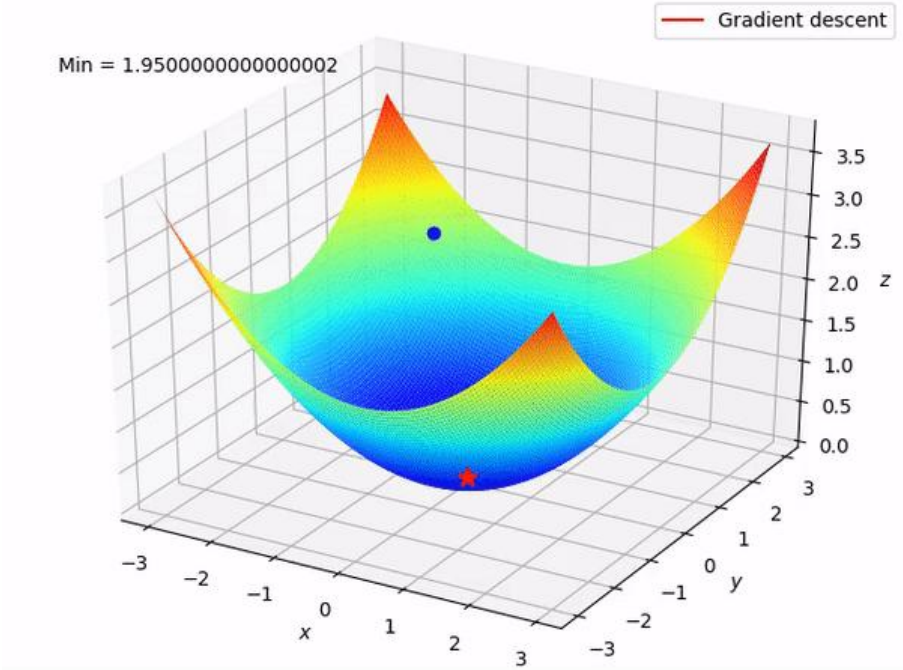

$f(x) = x^2 + 5\sin(x);\ x_0 = 5;\ \eta = 0.10$
30
25
20
15
10
5
0
−5
−6
−4
−2
0
2
4
6
iter 0/29: cost = 20.21, grad = 11.4183


$f(x) = x^2 + 5\sin(x);\ x_0 = -5;\ \eta = 0.01$
30
25
20
15
10
5
0
−5
−6
−4
−2
0
2
4
6
iter 0/99: cost = 29.79, grad = -8.5817



Hình 5. Gradient cho hàm 1 biến

2.1.2. Gradient descent cho hàm nhiều biến



Hình 6. Gradient cho hàm nhiều biến

2.1.3. Ưu điểm

Thuật toán gradient descent cơ bản, dễ hiểu.

Thuật toán đã giải quyết được vấn đề tối ưu model neural network bằng cách cập nhật trọng số sau mỗi vòng lặp.

2.1.4. Nhược điểm

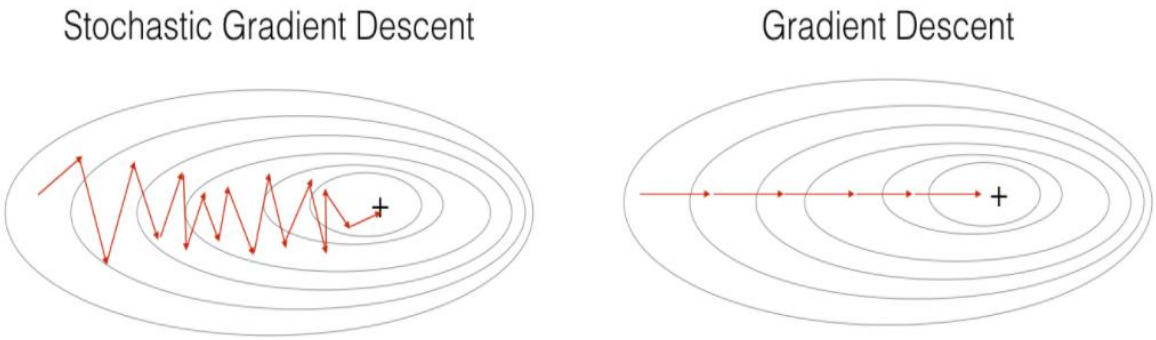
Vì đơn giản nên thuật toán Gradient Descent còn nhiều hạn chế như phụ thuộc vào nghiệm khởi tạo ban đầu và learning rate.

Ví dụ 1 hàm số có 2 global minimum thì tùy thuộc vào 2 điểm khởi tạo ban đầu sẽ cho ra 2 nghiệm cuối cùng khác nhau.

Tốc độ học quá lớn sẽ khiến cho thuật toán không hội tụ, quanh quẩn bên đích vì bước nhảy quá lớn; hoặc tốc độ học nhỏ ảnh hưởng đến tốc độ training

2.2. Stochastic Gradient Descent (SGD)

Stochastic là 1 biến thể của Gradient Descent. Thay vì sau mỗi epoch chúng ta sẽ cập nhật trọng số (Weight) 1 lần thì trong mỗi epoch có N điểm dữ liệu chúng ta sẽ cập nhật trọng số N lần. Nhìn vào 1 mặt, SGD sẽ làm giảm đi tốc độ của 1 epoch. Tuy nhiên nhìn theo 1 hướng khác, SGD sẽ hội tụ rất nhanh chỉ sau vài epoch. Công thức SGD cũng tương tự như GD nhưng thực hiện trên từng điểm dữ liệu.



Hình 7. So sánh giữa 2 thuật toán tối ưu

Nhìn vào 2 hình trên, ta thấy SGD có đường đi khá là zig zắc, không mượt như GD. Dễ hiểu điều đó vì 1 điểm dữ liệu không thể đại diện cho toàn bộ dữ liệu. Ở đây, GD có hạn chế đối với cơ sở dữ liệu lớn (vài triệu dữ liệu) thì việc tính toán đạo hàm trên toàn bộ dữ liệu qua mỗi vòng lặp trở nên cồng kềnh. Bên cạnh đó GD không phù hợp với online learning. Online learning là khi dữ liệu cập nhật liên tục (ví dụ như thêm người dùng đăng kí) thì mỗi lần thêm dữ liệu ta phải tính lại đạo hàm trên toàn bộ dữ liệu => thời gian tính toán lâu, thuật toán không online nữa. Vì thế SGD ra đời để giải quyết vấn đề đó, vì mỗi lần thêm dữ liệu mới vào chỉ cần cập nhật trên 1 điểm dữ liệu đó thôi, phù hợp với online learning.

Một ví dụ minh hoạ: có 10.000 điểm dữ liệu thì chỉ sau 3 epoch ta đã có được nghiệm tốt, còn với GD ta phải dùng tới 90 epoch để đạt được kết quả đó.

2.2.1. Ưu điểm

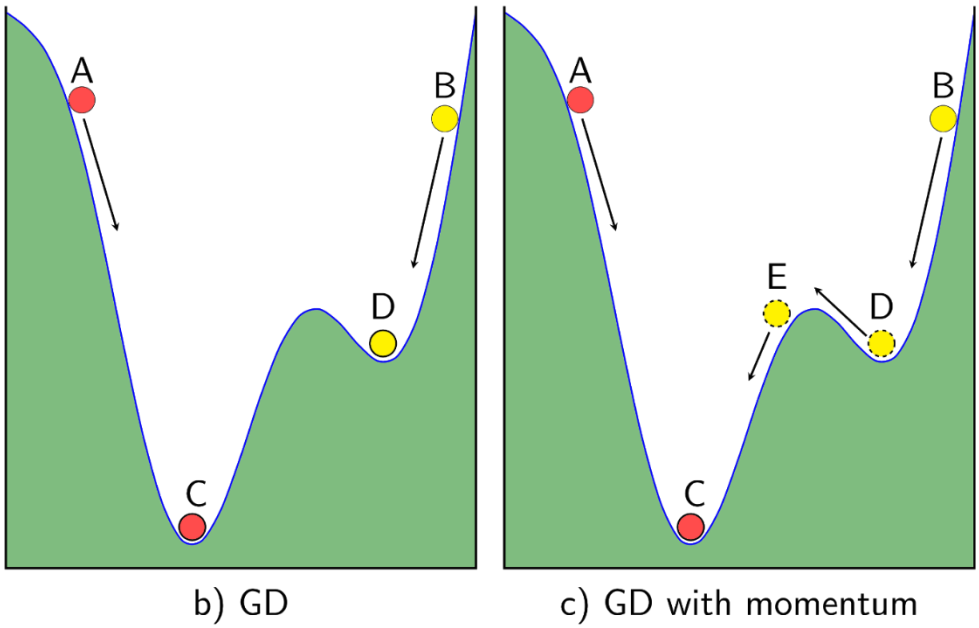
Thuật toán giải quyết được đối với cơ sở dữ liệu lớn mà GD không làm được. Thuật toán tối ưu này hiên nay vẫn hay được sử dụng.

2.2.2. Nhược điểm

Thuật toán vẫn chưa giải quyết được 2 nhược điểm lớn của gradient descent (learning rate, điểm dữ liệu ban đầu). Vì vậy ta phải kết hợp SGD với 1 số thuật toán khác như: Momentum, AdaGrad...Các thuật toán này sẽ được trình bày ở phần sau.

2.3. Momentum

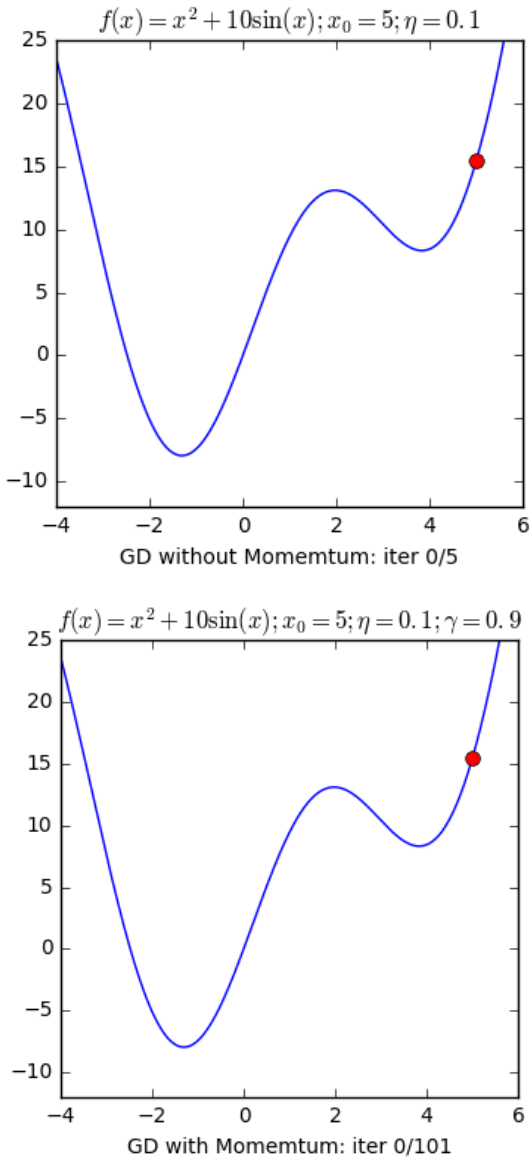
Để khắc phục các hạn chế trên của thuật toán Gradient Descent người ta dùng gradient descent with momentum.



Hình 8. Mô tả cách hoạt động của Momentum

Để giải thích được Gradient with Momentum thì trước tiên ta nên nhìn dưới góc độ vật lí: Như hình b phía trên, nếu ta thả 2 viên bi tại 2 điểm khác nhau A và B thì viên bị A sẽ trượt xuống điểm C còn viên bi B sẽ trượt xuống điểm D, nhưng ta lại không mong muốn viên bi B sẽ dừng ở điểm D (local minimum) mà sẽ tiếp tục lăn tới điểm C (global minimum). Để thực hiện được điều đó ta phải cấp cho viên bi B 1 vận tốc ban đầu đủ lớn để nó có thể vượt qua điểm E tới điểm C. Dựa vào ý tưởng này người ta xây dựng nên thuật toán Momentum (tức là theo đà tiến tới).

Nhìn dưới góc độ toán học, ta có công thức Momentum:
xnew = xold - (gama. v + learningrate. gradient)
Trong đó:
xnew: tọa độ mới
xod: tọa độ cũ
gama: parameter, thường =0.9
learningrate: tốc độ học



Hình 9. Gratient hàm 1 biến với Momentum

Qua 2 ví dụ minh họa trên của hàm f(x) = x.2 + 10sin(x), ta thấy GD without momentum sẽ hội tụ sau 5 vòng lặp nhưng không phải là global minimum. Nhưng GD with momentum dù mất nhiều vòng lặp nhưng nghiệm tiến tới global minimum, qua hình ta thấy nó sẽ vượt tốc tiến tới điểm global minimum và dao động qua lại quanh điểm đó trước khai dừng lại.

2.3.1. Ưu điểm

Thuật toán tối ưu giải quyết được vấn đề: Gradient Descent không tiến được tới điểm global minimum mà chỉ dừng lại ở local minimum.

2.3.2. Nhược điểm

Tuy momentum giúp hòn bi vượt dốc tiến tới điểm đích, tuy nhiên khi tới gần đích, nó vẫn mất khá nhiều thời gian giao động qua lại trước khi dừng hẳn, điều này được giải thích vì viên bi có đà.

2.4. Adagrad

Không giống như các thuật toán trước đó thì learning rate hầu như giống nhau trong quá trình training (learning rate là hằng số), Adagrad coi learning rate là 1 tham số. Tức là Adagrad sẽ cho learning rate biến thiên sau mỗi thời điểm t.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

Trong đó:

n: hằng số

gt: gradient tại thời điểm t

ϵ: hệ số tránh lỗi (chia cho mẫu bằng 0)

G: là ma trận chéo mà mỗi phần tử trên đường chéo (i,i) là bình phương của đạo hàm vectơ tham số tại thời điểm t.

2.4.1. Ưu điểm

Một lợi ích dễ thấy của Adagrad là tránh việc điều chỉnh learning rate bằng tay, chỉ cần để tốc độ học default là 0.01 thì thuật toán sẽ tự động điều chỉnh.

2.4.2. Nhược điểm

Yếu điểm của Adagrad là tổng bình phương biến thiên sẽ lớn dần theo thời gian cho đến khi nó làm tốc độ học cực kì nhỏ, làm việc training trở nên đóng băng.

2.5. RMSprop

RMSprop giải quyết vấn đề tỷ lệ học giảm dần của Adagrad bằng cách chia tỷ lệ học cho trung bình của bình phương gradient.

$$E[g^2]_t = 0{,}9E[g^2]_{t\text{-}1} + 0{,}1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

2.5.1. Ưu điểm

Ưu điểm rõ nhất của RMSprop là giải quyết được vấn đề tốc độ học giảm dần của Adagrad (vấn đề tốc độ học giảm dần theo thời gian sẽ khiến việc training chậm dần, có thể dẫn tới bị đóng băng)

2.5.2. Nhược điểm

Thuật toán RMSprop có thể cho kết quả nghiệm chỉ là local minimum chứ không đạt được global minimum như Momentum. Vì vậy người ta sẽ kết hợp cả 2 thuật toán Momentum với RMSprop cho ra 1 thuật toán tối ưu Adam. Chúng ta sẽ trình bày nó trong phần sau.

2.6. Adam

Như đã nói ở trên Adam là sự kết hợp của Momentum và RMSprop. Nếu giải thích theo hiện tượng vật lí thì Momentum giống như 1 quả cầu lao xuống dốc, còn Adam như 1 quả cầu rất nặng có ma sát, vì vậy nó dễ dàng vượt qua local minimum tới global minimum và khi tới global minimum nó không mất nhiều thời gian dao động qua lại quanh đích vì nó có ma sát nên dễ dừng lại hơn.

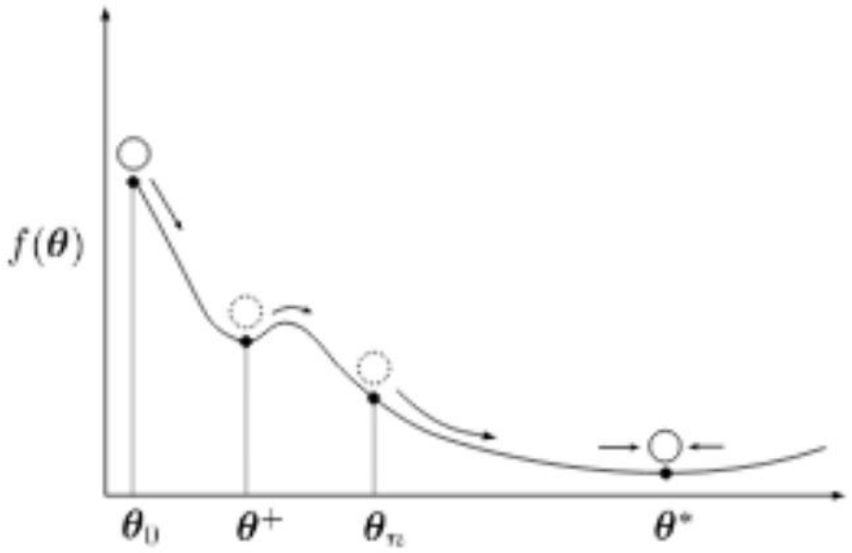


Figure 2: Heavy Ball with Friction, where the ball with mass overshoots the local minimum θ^+ and settles at the flat minimum θ^*.

Hình 10. Cách hoạt động của thuật toán

Công thức:

$$
\begin{aligned}
\boldsymbol{g}_n &\longleftarrow \nabla f(\boldsymbol{\theta}_{n-1}) \\
\boldsymbol{m}_n &\longleftarrow (\beta_1/(1-\beta_1^n))\,\boldsymbol{m}_{n-1} + ((1-\beta_1)/(1-\beta_1^n))\,\boldsymbol{g}_n \\
\boldsymbol{v}_n &\longleftarrow (\beta_2/(1-\beta_2^n))\,\boldsymbol{v}_{n-1} + ((1-\beta_2)/(1-\beta_2^n))\,\boldsymbol{g}_n \odot \boldsymbol{g}_n \\
\boldsymbol{\theta}_n &\longleftarrow \boldsymbol{\theta}_{n-1} - a\,\boldsymbol{m}_n/(\sqrt{\boldsymbol{v}_n} + \epsilon)\,,
\end{aligned}
$$

VI. Mô hình CNN (Convolutional Neural Networks)

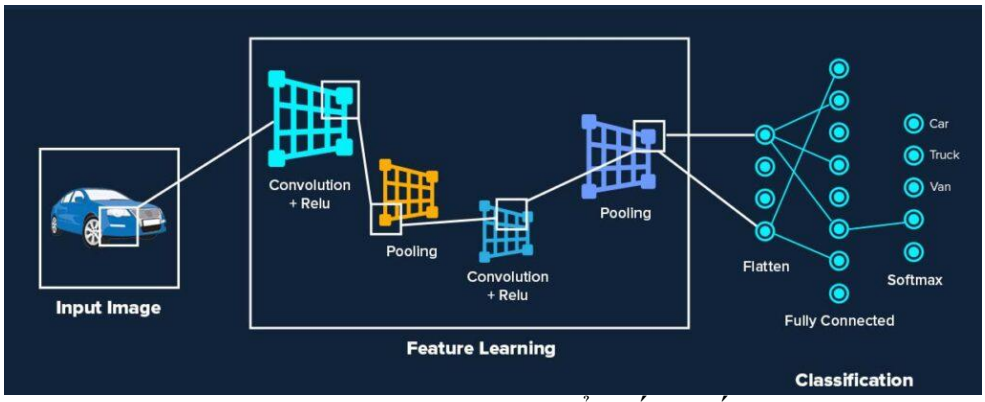
1. Tổng quan

Convolutional Neural Networks (viết tắt là CNN) là một mô hình học sâu (Deep Learning) được thiết kế chuyên biệt để xử lý dữ liệu hình ảnh và thị giác máy tính. CNN hoạt động dựa trên nguyên lý của mạng nơ-ron truyền thống, nhưng điểm khác biệt chính là khả năng tự động trích xuất đặc trưng mà không cần sự can thiệp thủ công từ con người. Nhờ đó, CNN trở thành công cụ có khả năng nhận diện vật thể, phân loại hình ảnh và xử lý video rất hiệu quả, mạnh mẽ.

Trước khi Convolutional Neural Networks ra đời, các phương pháp nhận diện hình ảnh yêu cầu con người phải thực hiện trích xuất đặc trưng bằng tay - một quá trình rất phức tạp và tốn nhiều thời gian. CNN đã thay thế quy trình này bằng cách sử dụng các phép toán của đại số tuyến tính, đặc biệt là tích chập (convolution) và nhân ma trận (matrix multiplication), để tự động nhận diện các đặc điểm trong hình ảnh. Thông qua nhiều lớp xử lý, CNN có thể phát hiện từ các đặc trưng đơn giản như cạnh, góc, màu sắc cho đến những chi tiết phức tạp hơn như hình dạng, kết cấu và toàn bộ đối tượng trong ảnh.

Với khả năng mở rộng và ứng dụng rộng rãi, CNN đang được sử dụng trong nhiều lĩnh vực như thị giác máy tính, chẩn đoán y khoa (phân tích ảnh y tế), nhận diện khuôn mặt trong an ninh và cả xe tự lái. Tuy nhiên, một hạn chế lớn của CNN là yêu cầu tài nguyên tính toán mạnh mẽ (GPU, TPU) để xử lý lượng dữ liệu khổng lồ trong quá trình huấn luyện.

Bên cạnh CNN truyền thống, thế giới cũng đang phát triển các loại convolutional neural networks mới. Một hướng nghiên cứu mới đang nổi lên là mạng nơ-ron tích chập lượng tử (Quantum Convolutional Neural Networks). QCNN sử dụng cơ học lượng tử để tăng tốc độ tính toán và tối ưu hóa hiệu suất trong các bài toán phức tạp. Mặc dù còn đang trong giai đoạn nghiên cứu, nhưng QCNN hứa hẹn sẽ mở ra nhiều đột phá mới trong trí tuệ nhân tạo AI và thị giác máy tính.



Hình 11. Quy trình phổ biến nhất

2. Các lớp cơ bản và nguyên lý hoạt động

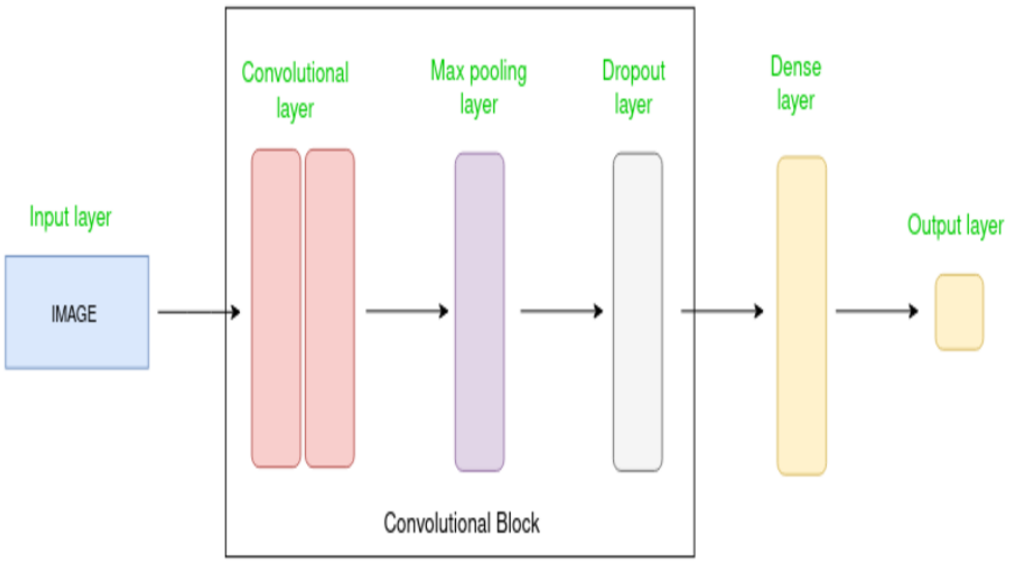
2.1. Convolution layer

Convolutional layer là lớp tích chập là thành phần quan trọng nhất của CNN, chịu trách nhiệm trích xuất các đặc trưng từ dữ liệu đầu vào. Lớp này sử dụng một bộ lọc (kernel) - một ma trận nhỏ có kích thước phổ biến như 3x3 hoặc 5x5 - quét qua từng vùng nhỏ của hình ảnh và thực hiện phép nhân tích chập (convolution) giữa các giá trị pixel với trọng số của bộ lọc. Kết quả của quá trình này tạo thành bản đồ đặc trưng (feature map), giúp mô hình phát hiện các đặc điểm như cạnh, góc, màu sắc hoặc kết cấu trong ảnh.

Các tham số quan trọng của lớp tích chập bao gồm: Số lượng bộ lọc, Stride (bước di chuyển của bộ lọc) và Padding (giữ kích thước ảnh). Trong đó:

+ Stride xác định khoảng cách di chuyển của kernel trên ảnh đầu vào theo cả chiều ngang (trái sang phải) và chiều dọc (trên xuống dưới).
+ Padding là quá trình thêm giá trị vào viền ảnh để kiểm soát kích thước feature map, bảo vệ thông tin viền ảnh khi thực hiện tích chập.

Sau mỗi phép tích chập, Convolutional Neural Networks thường áp dụng hàm kích hoạt ReLU (Rectified Linear Unit) để loại bỏ giá trị âm, tăng tính phi tuyến và giúp mô hình học hiệu quả hơn.



Hình 12. Qúa trình thứ tự các layer diễn ra

2.2. Pooling layer

Sau khi trích xuất đặc trưng qua lớp tích chập, Convolutional Neural Networks sử dụng Pooling Layer để giảm kích thước feature map, từ đó giảm số lượng tham số, tăng hiệu suất tính toán và tránh hiện tượng overfitting (mô hình học quá kỹ vào dữ liệu huấn luyện, nhưng lại hoạt động kém khi gặp dữ liệu mới). Pooling hoạt động bằng cách áp dụng một bộ lọc nhỏ (thường là 2x2 hoặc 3x3) để lấy giá trị đại diện cho mỗi vùng quét, giúp giữ lại những thông tin quan trọng nhất.

Có hai phương pháp pooling phổ biến: Max Pooling và Average Pooling.

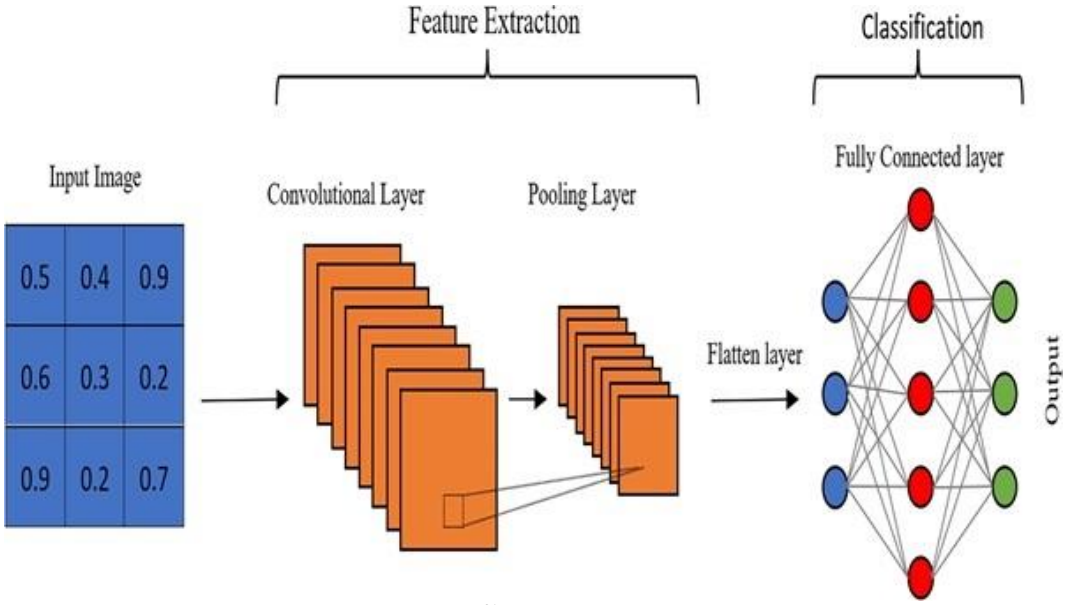
+ Trong Max Pooling, giá trị lớn nhất trong vùng quét sẽ được giữ lại, giúp mô hình tập trung vào những đặc trưng nổi bật nhất.
+ Average Pooling tính trung bình các giá trị trong vùng quét, giúp tổng hợp thông tin thay vì chỉ giữ giá trị lớn nhất như Max Pooling.

Mặc dù pooling làm mất đi một số thông tin, nhưng đổi lại, nó giúp mô hình hoạt động hiệu quả hơn, giảm thiểu độ phức tạp và cải thiện khả năng tổng quát hóa đối với dữ liệu mới.

2.3. Fully connected layer

Fully connected layer là lớp kết nối đầy đủ nằm ở cuối mạng Convolutional Neural Networks, đóng vai trò tổng hợp tất cả các đặc trưng đã trích xuất và thực hiện nhiệm vụ phân loại hình ảnh. Ở lớp này, mỗi nơ-ron được kết nối với toàn bộ nơ-ron ở lớp trước, tạo nên một mạng lưới liên kết chặt chẽ. Các giá trị từ feature map trước đó sẽ được chuyển thành một vector một chiều, một chuỗi dài duy nhất và đưa vào lớp fully connected để xử lý. Quá trình này được gọi là Làm phẳng Flattening.

Tiếp đó, CNN sử dụng các hàm kích hoạt phi tuyến như Softmax hoặc Sigmoid để tính toán xác suất cho từng lớp đầu ra. Điều này giúp cho mô hình đưa ra quyết định cuối cùng, chẳng hạn như phân loại hình ảnh thành các nhóm khác nhau (ví dụ: chó, mèo, ô tô, v.v.).



Hình 13. Qúa trình diễn ra Fully Connected Layer

2.4. Additional layers

Bên cạnh ba lớp chính, CNN có thể bao gồm một số lớp bổ sung để tối ưu hiệu suất và độ chính xác của mô hình.

+ Lớp kích hoạt (Activation Layer) giúp tăng khả năng học và tạo tính phi tuyến cho mô hình bằng cách áp dụng các hàm phi tuyến như ReLU, Sigmoid hoặc Tanh.
+ Lớp dropout (Dropout Layer) là một kỹ thuật quan trọng để giảm overfitting, bằng cách tạm thời loại bỏ ngẫu nhiên một số nơ-ron trong quá trình huấn luyện, giúp mô hình trở nên linh hoạt và tổng quát hơn.
+ Ngoài ra, lớp chuẩn hóa (Batch Normalization Layer) cũng thường được sử dụng để tăng tốc độ huấn luyện và giúp mô hình ổn định hơn bằng cách chuẩn hóa dữ liệu giữa các lớp.

3. Cấu trúc mạng neural tích chập

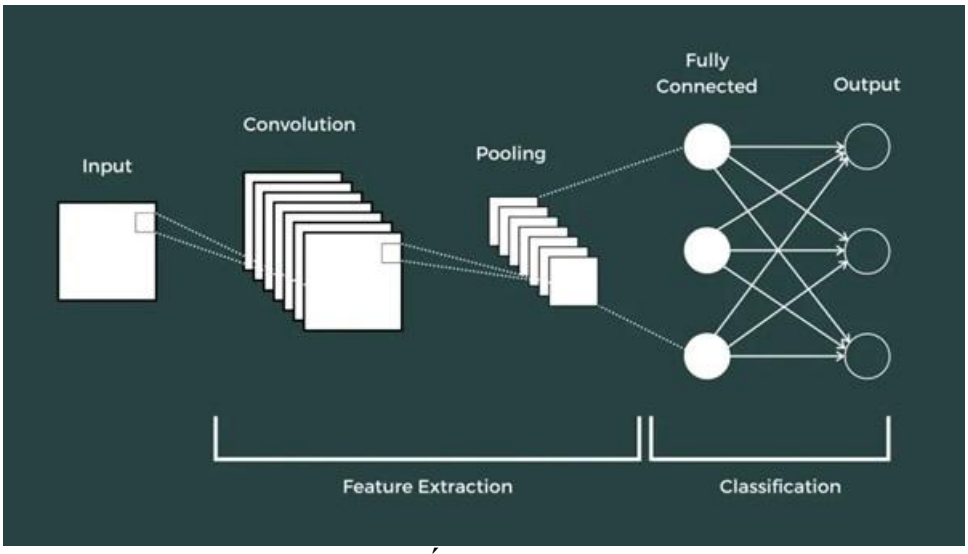
Mạng nơ-ron tích chập có cấu trúc rất đặc biệt, giúp mô hình tự động trích xuất đặc trưng từ dữ liệu hình ảnh mà không cần sự can thiệp thủ công.

Cơ bản, CNN được xây dựng bằng cách chồng nhiều lớp tích chập (Convolutional Layers), xen kẽ với lớp pooling (Pooling Layers) và kết thúc bằng lớp kết nối đầy đủ (Fully Connected Layers), giúp xử lý và phân loại hình ảnh. Ngoài ra, các hàm kích hoạt phi tuyến (như ReLU, Tanh) được sử dụng để tăng khả năng biểu diễn của mô hình. Mỗi lớp trong CNN đảm nhận một nhiệm vụ cụ thể, từ trích xuất đặc trưng đến giảm kích thước và cuối cùng là phân loại hình ảnh.

Cấu trúc cơ bản của CNN có thể chia thành ba phần chính: trường tiếp nhận cục bộ (Local Receptive Field), trọng số chia sẻ (Shared Weights and Bias) và lớp tổng hợp (Pooling Layer):

+ Trường tiếp nhận cục bộ - Local receptive field: Là nơi dữ liệu đầu vào được phân tách thành từng vùng nhỏ để xử lý. Thay vì kết nối toàn bộ các điểm ảnh (pixels) của một hình ảnh với tất cả các nơ-ron ở lớp tiếp theo, CNN chỉ kết nối từng vùng nhỏ (local receptive field) với một nơ-ron nhất định. Như vậy mỗi nơ-ron sẽ chỉ nhận diện đặc trưng của một khu vực cục bộ. Điều này giúp mô hình có khả năng nhận diện các đặc trưng quan trọng theo từng khu vực nhỏ trong ảnh trước khi tổng hợp thành thông tin lớn hơn ở các lớp tiếp theo.
+ Trọng số chia sẻ: Giúp giảm đáng kể số lượng tham số cần huấn luyện trong mô hình. Trong một lớp tích chập, các bộ lọc (filters) được quét trên toàn bộ hình ảnh theo cơ chế tích chập (convolution), với mỗi bộ lọc đảm nhận nhiệm vụ phát hiện một đặc trưng cụ thể, chẳng hạn như cạnh, góc, màu sắc hoặc hình dạng. Việc sử dụng chung các trọng số giữa các vùng trong ảnh giúp CNN có thể học được các mẫu đặc trưng một cách hiệu quả và tiết kiệm tài nguyên tính toán.
+ Lớp tổng hợp (Pooling Layer): Đóng vai trò giảm kích thước dữ liệu đầu ra, giúp tối ưu mô hình và tránh dư thừa thông tin. Pooling giúp mô hình trở nên bất biến với các thay đổi như dịch chuyển, co giãn hoặc xoay hình ảnh, đảm bảo CNN vẫn có thể nhận diện đúng đối tượng dù chúng xuất hiện ở các vị trí khác nhau trong ảnh. Có hai loại pooling phổ biến: Max Pooling, lấy giá trị lớn nhất trong vùng quét để giữ lại đặc trưng quan trọng nhất, và Average Pooling, tính giá trị trung bình của các điểm trong vùng quét để làm mịn dữ liệu.

Với cơ chế kết hợp giữa các lớp này, Convolutional Neural Networks có khả năng xây dựng biểu diễn trừu tượng từ cấp độ thấp đến cao, từ các đặc điểm đơn giản như cạnh và góc cho đến các hình dạng phức tạp hơn như khuôn mặt hoặc vật thể. Bên cạnh đó, trong suốt quá trình huấn luyện, CNN tự động điều chỉnh các bộ lọc và trọng số thông qua backpropagation, và thuật toán tối ưu hóa, giúp mô hình học cách trích xuất đặc trưng và nhận diện hình ảnh một cách hiệu quả.



Hình 14. Cấu trúc mạng CNN

4. Lợi ích của mangh neural tích chập

– Khả năng xử lý với độ chính xác cao trong thị giác máy tính

– Tự động trích xuất đặc trưng, giảm công sức xử lý thủ công

– Tái sử dụng mô hình với Transfer Learning

– Hiệu suất cao nhờ chia sẻ tham số

– Khả năng tổng quát hóa cao, giảm overfitting

5. Cách chọn các tham số cho CNN

5.1. Số lượng lớp tích chập (Convolution Layer)

Số lượng convolution layer ảnh hưởng trực tiếp đến khả năng học của mô hình. Các lớp tích chập đầu tiên sẽ nhận diện những đặc trưng cơ bản như cạnh, góc, trong khi các lớp sâu hơn sẽ xử lý thông tin phức tạp hơn như hình dạng và kết cấu. Nếu mô hình quá nông (ít lớp), nó có thể không đủ mạnh để trích xuất đầy đủ các đặc trưng cần thiết, dẫn đến hiệu suất thấp. Ngược lại, nếu mô hình quá sâu (nhiều lớp), nó có thể dễ gặp phải vanishing gradient, làm giảm khả năng học hiệu quả.

Vì vậy, khi xem xét số lượng lớp tích chập trong Convolutional Neural Networks cần lưu ý:

+ Với các bài toán nhận diện ảnh thông thường, chỉ cần 3 - 5 lớp tích chập là đủ để đạt kết quả tốt.

+ Đối với các mô hình lớn hơn như ResNet hoặc VGG, số lượng lớp có thể lên tới hàng chục, thậm chí hàng trăm lớp, nhưng cần có kỹ thuật skip connection hoặc batch normalization để giúp mô hình hội tụ tốt hơn.

5.2. Kích thước bộ lọc (Filter Size)

Bộ lọc (kernel) là thành phần chính trong quá trình tích chập, giúp mô hình trích xuất đặc trưng từ ảnh. Bộ lọc nhỏ (3x3) thường được sử dụng vì nó có khả năng nắm bắt chi tiết tốt hơn mà không làm tăng quá nhiều số lượng tham số. Trong khi đó, bộ lọc lớn hơn (5x5 hoặc 7x7) có thể giúp nhận diện các đặc trưng rộng hơn, nhưng đồng thời làm tăng số lượng tham số và độ phức tạp tính toán.

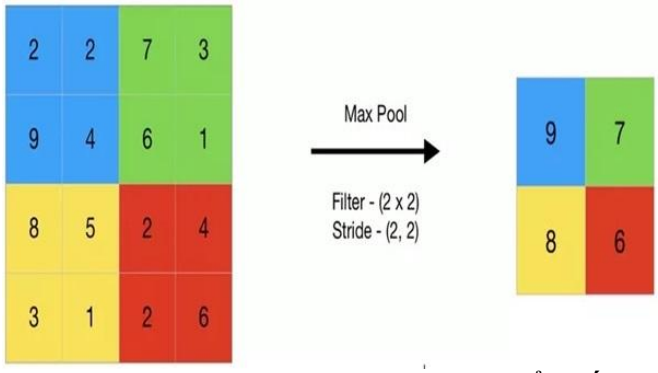
Một số cách lựa chọn kích thước bộ học phù hợp trong Convolutional Neural Networks bao gồm:

+ Bộ lọc 3x3 là lựa chọn phổ biến nhất vì nó cân bằng giữa hiệu suất và tốc độ tính toán.
+ Bộ lọc 5x5 có thể được sử dụng khi cần phát hiện các đặc trưng lớn hơn, nhưng thường chỉ được áp dụng ở các lớp đầu tiên của mô hình.
+ Nếu muốn giảm tải tính toán nhưng vẫn giữ được khả năng trích xuất đặc trưng, có thể sử dụng nhiều bộ lọc 3x3 liên tiếp thay vì một bộ lọc lớn hơn

5.3. Kích thước Pooling Layer

Pooling Layer giúp giảm kích thước đầu ra của các lớp tích chập, giảm số lượng tham số và tăng khả năng tổng quát hóa của mô hình. Trong đó, Max Pooling thường được sử dụng nhiều hơn so với Average Pooling vì nó giữ lại giá trị nổi bật nhất trong vùng quét, giúp mô hình tập trung vào các đặc trưng quan trọng. Kích thước pooling phổ biến nhất là 2x2, giúp giảm kích thước đầu ra xuống một nửa mà vẫn giữ lại thông tin quan trọng

+ Sử dụng Max Pooling với kích thước 2x2 cho các bài toán nhận diện ảnh thông thường.
+ Với ảnh có độ phân giải cao hoặc khi cần giảm nhanh kích thước feature map, có thể thử pooling 4x4, nhưng cần kiểm tra xem có mất quá nhiều thông tin hay không.



Hình 15. Kich thước poolling phổ biến

5.4. Số lần huấn luyện (Epochs) và chia tập dữ liệu (Train/Test Split)

Số lần huấn luyện (epochs) ảnh hưởng đến khả năng mô hình học từ dữ liệu. Nếu số epochs quá ít, mô hình có thể chưa học được đủ đặc trưng quan trọng và dẫn đến underfitting. Ngược lại, nếu số epochs quá nhiều, mô hình có thể học quá sâu vào dữ liệu huấn luyện và dẫn đến overfitting.

Do đó, nên tham khảo các tham số sau:

+ Bắt đầu với 10-50 epochs, sau đó theo dõi độ lỗi (loss) và độ chính xác (accuracy) trên tập validation để điều chỉnh.
+ Nếu thấy mô hình đạt độ chính xác cao trên tập huấn luyện nhưng thấp trên tập kiểm tra, có thể sử dụng kỹ thuật early stopping để ngừng huấn luyện khi mô hình bắt đầu có dấu hiệu overfitting.
+ Chia tập dữ liệu thành 80% train - 20% test hoặc 70% train - 15% validation - 15% test để đảm bảo đánh giá mô hình chính xác nhất.

6. Thách thức và hạn chế của mạng neural tích chập

– Yêu cầu tài nguyên tính toán lớn:
– Cần lượng dữ liệu lớn để huấn luyện
– Độ phức tạp cao, khó huấn luyện
– Khó giải thích và kiểm soát kết quả
– Dễ gặp vấn đề overfitting nếu không kiểm soát tốt

7. Một số mô hình CNN phổ biến

– LeNet
– AlexNet
– Resnet
– GoogleNet
– VGG

8. Dataset

https://www.kaggle.com/datasets/abdelghaniaaba/wildfire-prediction-dataset

VII. Đánh giá mô hình

1. Accuracy and Loss

1.1. Accracy

Dịch Tiếng Anh ra thôi! Accuracy = Sự chính xác.

Ví dụ chúng ta đưa vào model 50 cái ảnh gồm cả chó và mèo vào một model phân loại. Sau khi chạy xong model thì thấy rằng có 30 ảnh được nhận diện đúng còn lại 20 ảnh nhận sai bét tè lè nhè.

Vậy ta có:

$$Accuracy = \frac{30}{50} = 0.6$$

Hay một cách tổng quát ta có:

$$Accuracy = \frac{n}{N}$$

Trong đó:
n: Số sample dự đoán đúng
N: Tổng số sample đưa vào dự đoán

Ta có thể tạm hiểu rằng model có accuracy càng cao thì càng tốt vì sẽ dự đoán được chính xác nhiều nhất (tạm hiểu nhé, trong các chương sau sẽ có các tham số khác nữa như Precision, Recall….)

1.2. Loss

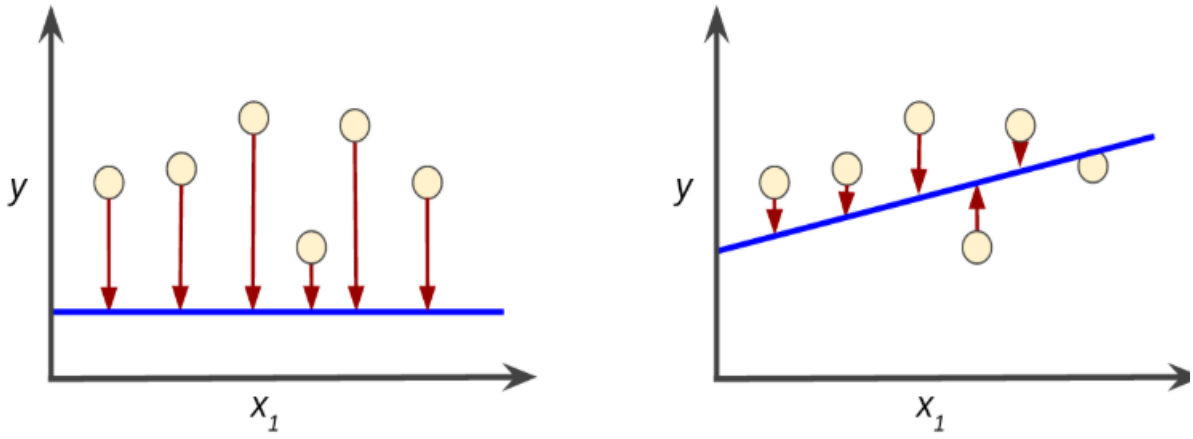
Loss thường là một số thực không âm (trừ một số trường hợp loss là cosin proximity) thể hiện sự chênh lệch giữa hai đại lượng: nhãn thật của dữ liệu và nhãn của dữ liệu do model của chúng ta predic ra.

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

Hai phương án có thể có cùng Accuracy nhưng Loss sẽ có thể khác nhau nhé. Nhiều bạn rất hay nhầm lẫn và đánh đồng hai khái niệm này. Ví dụ như với bài toán chó mèo đi, giả sử dữ liệu predict chỉ có duy nhất 1 ảnh CHÓ (vector nhãn thực là [1 , 0], probability là chó là 1 và mèo là 0, tạm gọi là y) và:

+ Phương án 01: Dự đoán ra probability của một ảnh đầu vào như sau 0.8 là chó và 0.2 là mèo (vector predict $\hat{y}$ = [0.8, 0.2]) -> Ta quyết định ảnh này là CHÓ và dự đoán đúng -> Accuracy = 1.
+ Phương án 02: Vector predict $\hat{y}$ = [0.6, 0.4] -> Ta cũng vẫn quyết định đây là CHÓ vì probability là chó vẫn cao hơn và -> Accuracy = 1

Tuy nhiên nhìn bằng mắt thường ta cũng thấy rõ ràng Phương án 1 tốt hơn vì dự đoán sát hơn với vector nhãn thực. Trong thực tế chúng ta sẽ sử dụng một số hàm loss khác nhau tùy vào bài toán cụ thể như: Hinge Loss, Cross Entropy Loss…nhưng tóm lại bạn cứ hiểu nôm na là Loss là hàm số trả về một số thực không âm để chỉ ra model của ta đang dự đoán sát (loss nhỏ) hay xa (loss to) so với nhãn thực của dữ liệu.
Hình bên trái loss khá lớn vì khoảng cách giữa đường dự đoán (xanh blue) và các điểm dữ liệu là khá xa. Hình bên phải thì ngon lành hơn nhiều



Hình 16. Đồ thị loss

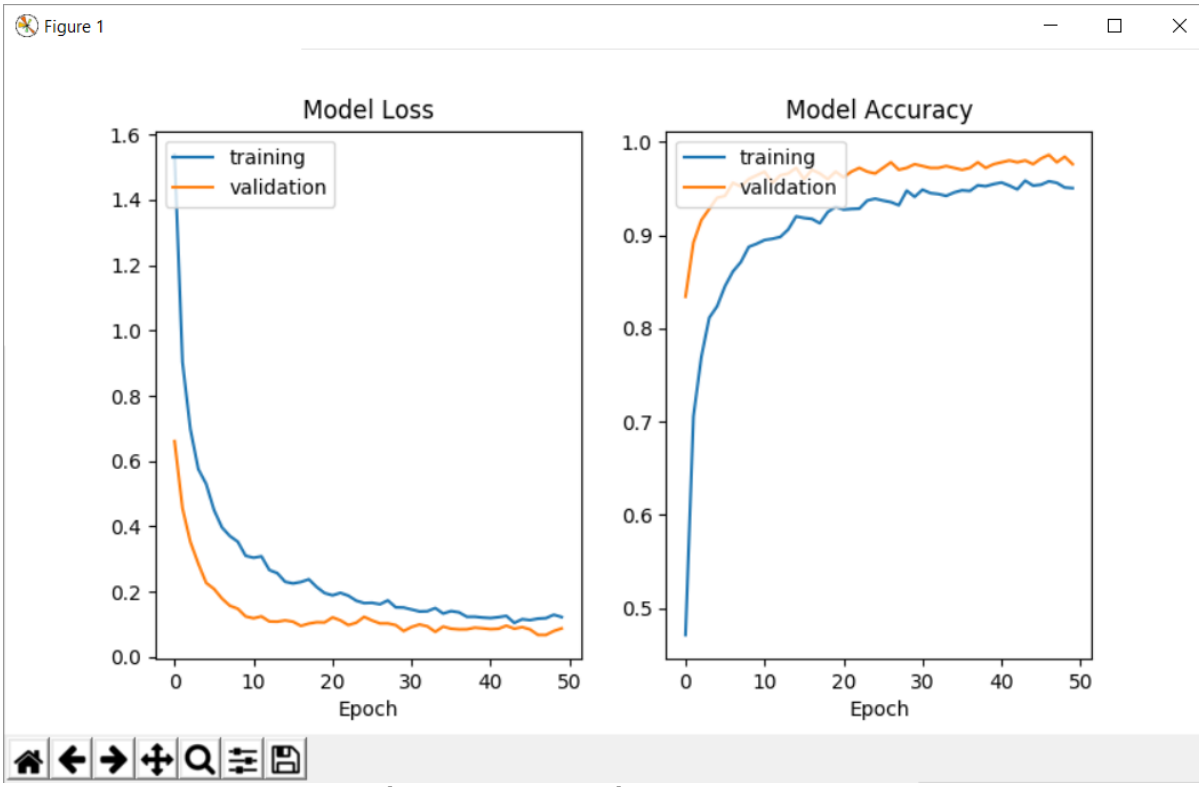
Ví dụ về công thức hàm loss Cross Entropy được dùng khá nhiều trong các bài toán Deep Learning:

$$\text{Loss} = -\sum_{i=1}^{n}\sum_{j=1}^{m} y_{i,j}\log_e(p_{i,j})$$

Trong đó:
n: Số lượng điểm dữ liệu
m: Số lượng class
$y_{i,j}$: Đây là nhãn thật của dữ liệu. $y_{i,j}$=1 nếu điểm dữ liệu i thuộc class j và =0 nếu ngược lại.
$p_{i,j}$: Đay là nhãn dự đoán ra. $p_{i,j}$ thể hiện probability mà model của bạn dự đoán điểm dữ liệu i thuộc class j.

1.3. Đồ thị chính xác và đồ thị mất mát

Trong thực tế train model, người ta hay plot loss và accuracy lên trên biểu đồ để trực quan hóa và dựa vào kinh nghiệm sẽ "ngắm" ra được vấn đề của model.



Đồ thị Loss và Đồ thị chính xác

Ví dụ như hình trên, khi số epoch tăng lên thì cả Train Loss và Validation Loss đều giảm, Train Acc và Val Acc đều tăng. Điều đó là phù hợp. Nhưng nếu như trong quá trình plot ra, đến một epoch nào đó tự nhiên train loss vẫn giảm và val loss bắt đầu tăng lên thì model đã bắt đầu Overfit rồi đó.

2. Ma trận Confusion (Confusion matrix)

Cách tính sử dụng accuracy như ở trên chỉ cho chúng ta biết được bao nhiêu phần trăm lượng dữ liệu được phân loại đúng mà không chỉ ra được cụ thể mỗi loại được phân loại như thế nào, lớp nào được phân loại đúng nhiều nhất, và dữ liệu thuộc lớp nào thường bị phân loại nhầm vào lớp khác. Để có thể đánh giá được các giá trị này, chúng ta sử dụng một ma trận được gọi là confusion matrix.

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

Với các bài toán với nhiều lớp dữ liệu, cách biểu diễn bằng màu này rất hữu ích. Các ô màu đậm thể hiện các giá trị cao. Một mô hình tốt sẽ cho một confusion matrix có các phần tử trên đường chéo chính có giá trị lớn, các phần tử còn lại có giá trị nhỏ. Nói cách khác, khi biểu diễn bằng màu sắc, đường chéo có màu càng đậm so với phần còn lại sẽ càng tốt.

2.1. True/False Positive/Negative

Cách đánh giá này thường được áp dụng cho các bài toán phân lớp có hai lớp dữ liệu. Cụ thể hơn, trong hai lớp dữ liệu này có một lớp nghiêm trọng hơn lớp kia và cần được dự đoán chính xác. Ví dụ, trong bài toán xác định có bệnh ung thư hay không thì việc không bị sót (miss) quan trọng hơn là việc chẩn đoán nhầm âm tính thành dương tính. Trong bài toán xác định có mìn dưới lòng đất hay không thì việc bỏ sót nghiêm trọng hơn việc báo động nhầm rất nhiều. Hay trong bài toán lọc email rác thì việc cho nhầm email quan trọng vào thùng rác nghiêm trọng hơn việc xác định một email rác là email thường.

Trong những bài toán này, người ta thường định nghĩa lớp dữ liệu quan trọng hơn cần được xác định đúng là lớp Positive (P-dương tính), lớp còn lại được gọi là Negative (N-âm tính). Ta định nghĩa True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN) dựa trên confusion matrix chưa chuẩn hoá như sau:

	Predicted as Positive	Predicted as Negative
Actual: Positive	True Positive (TP)	False Negative (FN)
Actual: Negative	False Positive (FP)	True Negative (TN)

Hình 17. Phân bố đồ thị ma trận

Người ta thường quan tâm đến TPR, FNR, FPR, TNR (R - Rate) dựa trên normalized confusion matrix như sau:

	Predicted as Positive	Predicted as Negative
Actual: Positive	TPR = TP/(TP + FN)	FNR = FN/(TP + FN)
Actual: Negative	FPR = FP/(FP + TN)	TNR = TN/(FP + TN)

Hình 18. Phân bố các thông số tốc độ

False Positive Rate còn được gọi là False Alarm Rate (tỉ lệ báo động nhầm), False Negative Rate còn được gọi là Miss Detection Rate (tỉ lệ bỏ sót). Trong bài toán dò mìn, thà báo nhầm còn hơn bỏ sót, tức là ta có thể chấp nhận False Alarm Rate cao để đạt được Miss Detection Rate thấp.

Chú ý:

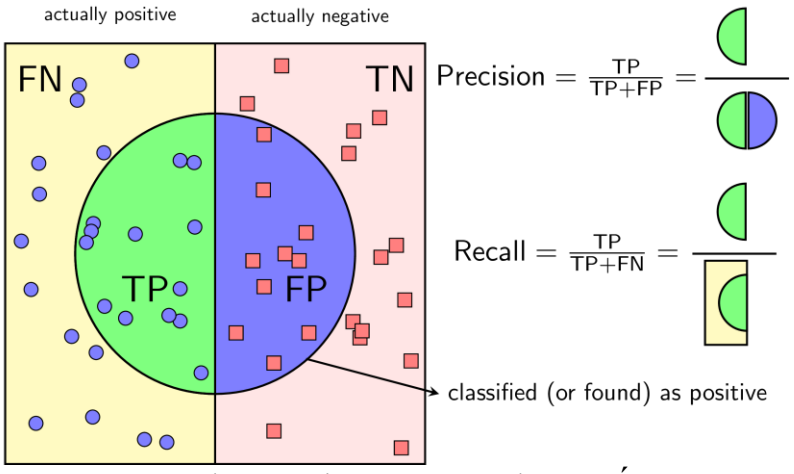
+ Việc biết một cột của confusion matrix này sẽ suy ra được cột còn lại vì tổng các hàng luôn bằng 1 và chỉ có hai lớp dữ liệu.
+ Với các bài toán có nhiều lớp dữ liệu, ta có thể xây dựng bảng True/False Positive/Negative cho mỗi lớp nếu coi lớp đó là lớp Positive, các lớp còn lại gộp chung thành lớp Negative, giống như cách làm trong one-vs-rest.

2.2. Precision và recall

Với bài toán phân loại mà tập dữ liệu của các lớp là chênh lệch nhau rất nhiều, có một phép đó hiệu quả thường được sử dụng là Precision-Recall.

Trước hết xét bài toán phân loại nhị phân. Ta cũng coi một trong hai lớp là positive, lớp còn lại là negative.

Xét hình dưới đây:



Hình 19. Vị trí của các thông số

Với một cách xác định một lớp là positive, Precision được định nghĩa là tỉ lệ số điểm true positive trong số những điểm được phân loại là positive (TP + FP).

Recall được định nghĩa là tỉ lệ số điểm true positive trong số những điểm thực sự là positive (TP + FN).

Một cách toán học, Precison và Recall là hai phân số có tử số bằng nhau nhưng mẫu số khác nhau:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Có thể nhận thấy rằng TPR và Recall là hai đại lượng bằng nhau. Ngoài ra, cả Precision và Recall đều là các số không âm nhỏ hơn hoặc bằng một.

Precision cao đồng nghĩa với việc độ chính xác của các điểm tìm được là cao. Recall cao đồng nghĩa với việc True Positive Rate cao, tức tỉ lệ bỏ sót các điểm thực sự positive là thấp.

Khi Precision = 1, mọi điểm tìm được đều thực sự là positive, tức không có điểm negative nào lẫn vào kết quả. Tuy nhiên, Precision = 1 không đảm bảo mô hình là tốt. Nếu một mô hình chỉ tìm được đúng một điểm positive mà nó chắc chắn nhất thì không thể gọi nó là một mô hình tốt.

Khi Recall = 1, mọi điểm positive đều được tìm thấy. Tuy nhiên, đại lượng này lại không đo liệu có bao nhiêu điểm negative bị lẫn trong đó. Nếu mô hình phân loại mọi điểm là positive thì chắc chắn Recall = 1, tuy nhiên dễ nhận ra đây là một mô hình cực tồi.

Một mô hình phân lớp tốt là mô hình có cả Precision và Recall đều cao, tức càng gần một càng tốt. Có hai cách đo chất lượng của bộ phân lớp dựa vào Precision và Reall: Precision-Recall curve và F-score.

2.3. F1-score

F1-score, là harmonic mean của precision và recall (giả sử rằng hai đại lượng này khác không):

$$\frac{2}{F_1} = \frac{1}{\text{precision}} + \frac{1}{\text{recall}} \text{ hay } F_1 = 2\frac{1}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2\frac{\text{precion} \cdot recall}{\text{precision} + \text{recall}}$$

F−1-score có giá trị nằm trong nửa khoảng (0,1]. F_1 càng cao, bộ phân lớp càng tốt. Khi cả recall và precision đều bằng 1 (tốt nhất có thể), F_1=1. Khi cả recall và precision đều thấp, ví dụ bằng 0.1, F_1=0.1. Dưới đây là một vài ví dụ về F1

precision	recall	F_1
1	1	1
0.1	0.1	0.1
0.5	0.5	0.5
1	0.1	0.182
0.3	0.8	0.36

Như vậy, một bộ phân lớp với precision = recall = 0.5 tốt hơn một bộ phân lớp khác với precision = 0.3, recall = 0.8 theo cách đo này.

Trường hợp tổng quát của F_1 score là F_β score:

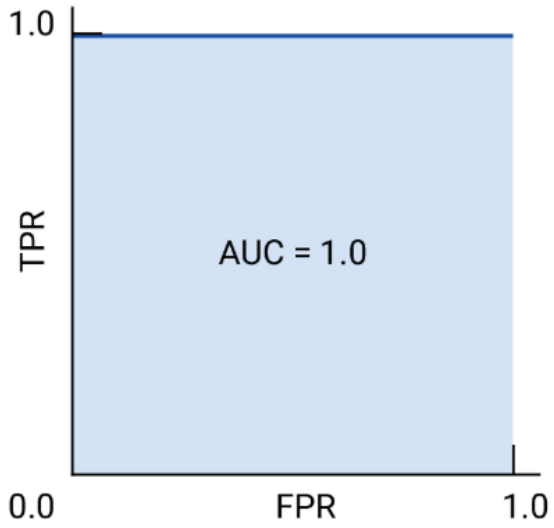
$$F_\beta = (1 + \beta^2) \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

F_1 chính là một trường hợp đặc biệt của F_β khi $\beta=1$. Khi $\beta>1$, recall được coi trọng hơn precision, khi $\beta<1$, precision được coi trọng hơn. Hai đại lượng β thường được sử dụng là $\beta=2$ và $\beta=0.5$.

2.4. Đường cong ROC

Đường cong ROC là hình ảnh trực quan về hiệu suất của mô hình trên tất cả các ngưỡng. Phiên bản dài của tên, đặc tính hoạt động của bộ thu, là một phần còn lại của công nghệ phát hiện radar trong Thế chiến II.

Đường cong ROC được vẽ bằng cách tính tỷ lệ dương tính thực (TPR) và tỷ lệ dương tính giả (FPR) ở mọi ngưỡng có thể (trong thực tế, ở các khoảng thời gian đã chọn), sau đó lập biểu đồ TPR trên FPR. Một mô hình hoàn hảo, ở một ngưỡng nào đó có TPR là 1.0 và FPR là 0.0, có thể được biểu thị bằng một điểm tại (0, 1) nếu tất cả các ngưỡng khác bị bỏ qua hoặc bằng cách sau:

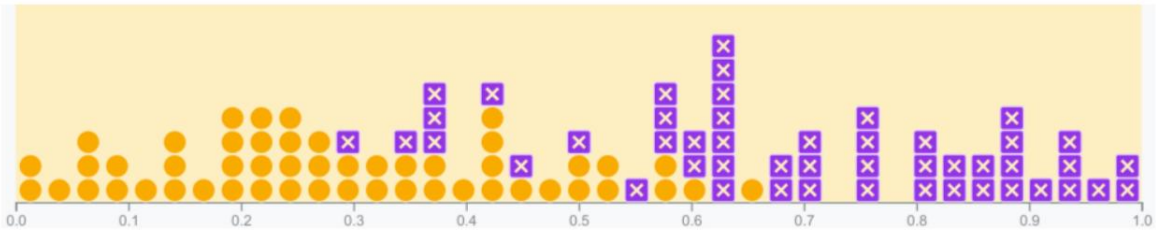


Hình 20. ROC và AUC của một mô hình hoàn hảo giả định

2.4.1. Diện tích dưới đường cong (AUC)

Diện tích dưới đường cong ROC (AUC) đại diện cho xác suất mô hình, nếu được cung cấp một ví dụ dương tính và âm tính được chọn ngẫu nhiên, sẽ xếp hạng dương tính cao hơn âm tính.

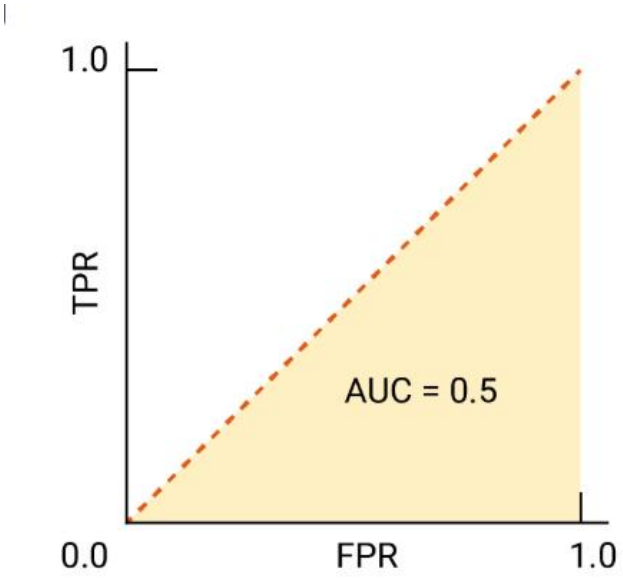
Mô hình hoàn hảo ở trên, chứa một hình vuông có các cạnh dài 1, có diện tích dưới đường cong (AUC) là 1.0. Điều này có nghĩa là có 100% khả năng mô hình sẽ xếp hạng chính xác một ví dụ dương tính được chọn ngẫu nhiên cao hơn một ví dụ âm tính được chọn ngẫu nhiên. Nói cách khác, khi xem xét mức độ phân tán của các điểm dữ liệu bên dưới, AUC cho biết xác suất mô hình sẽ đặt một hình vuông được chọn ngẫu nhiên ở bên phải một hình tròn được chọn ngẫu nhiên, không phụ thuộc vào vị trí đặt ngưỡng.



Nói một cách cụ thể hơn, một bộ phân loại thư rác có AUC là 1.0 luôn gán cho một email rác ngẫu nhiên có nhiều khả năng là thư rác hơn một email hợp lệ ngẫu nhiên. Việc phân loại thực tế của từng email phụ thuộc vào ngưỡng mà bạn chọn.

Đối với một bộ phân loại nhị phân, một mô hình hoạt động chính xác như các dự đoán ngẫu nhiên hoặc tung đồng xu có ROC là một đường chéo từ (0,0) đến (1,1). AUC là 0,5, thể hiện xác suất 50% để xếp hạng chính xác một ví dụ ngẫu nhiên về ví dụ tích cực và ví dụ tiêu cực.

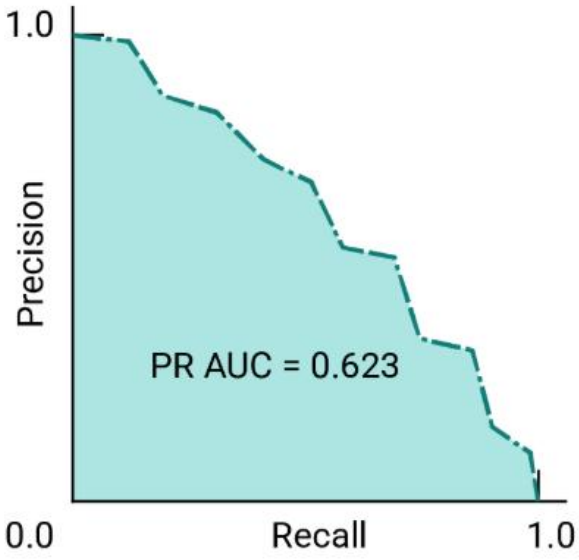
Trong ví dụ về trình phân loại thư rác, trình phân loại thư rác có AUC là 0,5 chỉ gán xác suất cao hơn một nửa cho một email rác ngẫu nhiên so với một email hợp lệ ngẫu nhiên.



Hình 21. ROC và AUC của các lần đoán hoàn toàn ngẫu nhiên

2.4.2. Đường cong độ chính xác-độ hồi quy

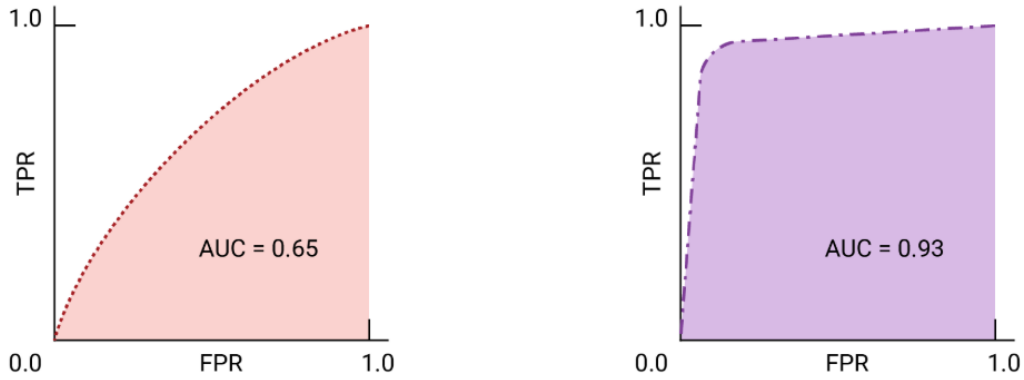
AUC và ROC hoạt động hiệu quả khi so sánh các mô hình khi tập dữ liệu được cân bằng gần như giữa các lớp. Khi tập dữ liệu không cân bằng, các đường cong độ chính xác-độ hồi quy (PRC) và diện tích bên dưới các đường cong đó có thể cung cấp hình ảnh trực quan so sánh hiệu suất mô hình tốt hơn. Đường cong độ chính xác-độ hồi quy được tạo bằng cách vẽ độ chính xác trên trục y và độ hồi quy trên trục x trên tất cả các ngưỡng.



Hình 22. ROC và AUC của đường cong độ chính xác- độ hồi quy

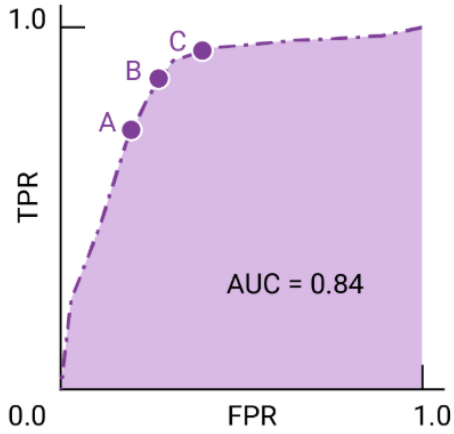
2.4.3. AUC và ROC để chọn mô hình và ngưỡng

AUC là một chỉ số hữu ích để so sánh hiệu suất của hai mô hình khác nhau, miễn là tập dữ liệu được cân bằng gần như nhau. Mô hình có diện tích lớn hơn dưới đường cong thường là mô hình tốt hơn.



Hình 23. ROC và AUC của hai mô hình giả định. Đường cong ở bên phải, với AUC lớn hơn, thể hiện mô hình tốt hơn trong hai mô hình.

Các điểm trên đường cong ROC gần nhất với (0,1) thể hiện phạm vi ngưỡng hoạt động hiệu quả nhất cho mô hình nhất định. Như đã thảo luận trong các phần Ngưỡng, Ma trận nhầm lẫn và Chọn chỉ số và đánh đổi, ngưỡng mà bạn chọn phụ thuộc vào chỉ số quan trọng nhất đối với trường hợp sử dụng cụ thể. Hãy xem xét các điểm A, B và C trong sơ đồ sau, mỗi điểm đại diện cho một ngưỡng:



Hình 24. Ba điểm được gắn nhãn đại diện cho các ngưỡng.

Nếu kết quả dương tính giả (cảnh báo giả) gây ra tổn thất lớn, bạn nên chọn một ngưỡng có FPR thấp hơn, chẳng hạn như ngưỡng tại điểm A, ngay cả khi TPR giảm. Ngược lại, nếu kết quả dương tính giả có chi phí thấp và kết quả âm tính giả (bỏ lỡ kết quả dương tính thực) có chi phí cao, thì ngưỡng cho điểm C (tối đa hoá TPR) có thể được ưu tiên. Nếu chi phí gần như tương đương, điểm B có thể mang lại sự cân bằng tốt nhất giữa TPR và FPR.

Phần 2. Chương trình sử dụng mô hình

1. Tkinter

1.1. Tổng quan

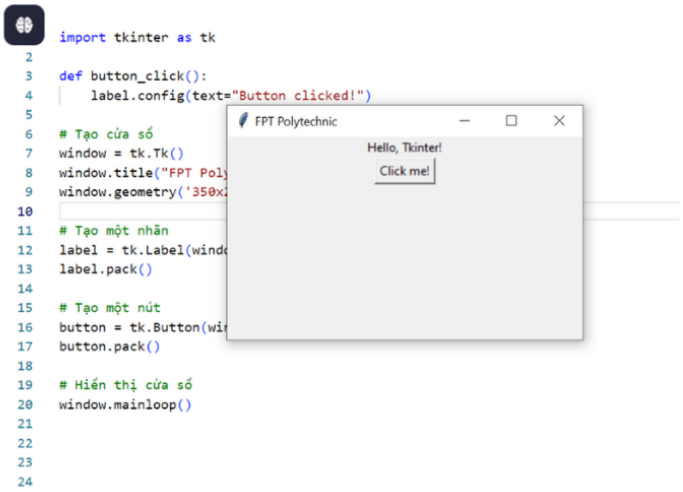
Tkinter là một thư viện lập trình giao diện người dùng đồ họa (GUI - Graphical User Interface) được tích hợp sẵn trong ngôn ngữ lập trình Python. Nó sẽ giúp người dùng tương tác với chương trình thông qua các thành phần đồ họa như nút bấm, ô nhập liệu, nhãn văn bản...

Dựa trên thư viện Tk (viết bằng C), Tkinter mang đến một cách tiếp cận dễ dàng cho cả người mới bắt đầu và lập trình viên có kinh nghiệm, cho phép họ xây dựng các ứng dụng với giao diện người dùng từ đơn giản đến phức tạp mà không cần phải nắm vững các khái niệm đồ họa phức tạp. Với cú pháp dễ hiểu và khả năng tích hợp liền mạch với Python, Tkinter là một lựa chọn lý tưởng để nhanh chóng tạo ra các prototype và ứng dụng GUI hoàn chỉnh.

1.2. Các thành phần giao diện

Tkinter hỗ trợ một loạt các thành phần giao diện người dùng (widgets), cho phép bạn tạo ra các ứng dụng đa dạng và phong phú. Dưới đây là một số thành phần phổ biến:

1.2.1. Widgets cơ bản



Hình 25. Giao diện khi dùng tkinter

Label: Hiển thị văn bản tĩnh hoặc hình ảnh.
Button: Nút bấm, kích hoạt một hàm khi được nhấn.
Entry: Ô nhập liệu văn bản đơn dòng.
Text: Vùng nhập liệu văn bản đa dòng, hỗ trợ định dạng.
Checkbutton: Hộp kiểm, cho phép người dùng chọn hoặc bỏ chọn một tùy chọn.
Radiobutton: Nút radio, cho phép người dùng chọn một trong nhiều tùy chọn.
Listbox: Danh sách cho phép người dùng chọn một hoặc nhiều mục.
Scrollbar: Thanh cuộn, dùng để điều hướng trong các widgets như Listbox và Text.
Frame: Khung chứa các widgets khác, giúp tổ chức giao diện.
Menu: Menu, cung cấp các tùy chọn cho người dùng.
Canvas: Vùng vẽ, cho phép vẽ đồ họa và hình ảnh phức tạp.
Scale: Thanh trượt, cho phép người dùng chọn một giá trị trong một phạm vi.
Spinbox: Hộp quay số, cho phép người dùng chọn một giá trị từ một danh sách.
ProgressBar: Thanh tiến trình, hiển thị tiến độ của một tác vụ.
Message: Hiển thị văn bản nhiều dòng với khả năng tự động xuống dòng.

1.2.2. Widgets nâng cao

Ngoài các widgets cơ bản trên, bạn có thể tìm thấy hoặc tạo ra các widgets nâng cao hơn, cung cấp chức năng phức tạp hơn, chẳng hạn như:

+ Treeview: Hiển thị dữ liệu theo cấu trúc cây.
+ Notebook/Tab: Tạo các tab để tổ chức giao diện.
+ PanedWindow: Cho phép chia cửa sổ thành nhiều vùng có thể thay đổi kích thước.
+ Dialog boxes: Hộp thoại cho các tương tác người dùng đặc biệt (ví dụ: hộp thoại xác nhận, hộp thoại nhập liệu).

1.3. Quản lý hình học

Quản lý hình học (Geometry Management) trong Tkinter là cách bạn sắp xếp và định vị các widgets (thành phần giao diện người dùng) trên cửa sổ ứng dụng. Tkinter Python cung cấp ba phương pháp chính để quản lý hình học: pack (), grid (), và place (). Mỗi phương pháp có ưu điểm và nhược điểm riêng, phù hợp với các trường hợp sử dụng khác nhau.

1.3.1. pack ()

Cách thức hoạt động: pack () sắp xếp các widgets theo chiều dọc hoặc ngang, một cách đơn giản và trực quan. Nó đặt các widgets vào một vùng chứa (thường là cửa sổ chính hoặc một Frame) và tự động điều chỉnh kích thước của chúng để phù hợp.

Ưu điểm: Dễ sử dụng, nhanh chóng cho các bố cục đơn giản.

Nhược điểm: Khó kiểm soát chính xác vị trí và kích thước của các widgets, đặc biệt trong các bố cục phức tạp. Việc thay đổi thứ tự pack () có thể ảnh hưởng đến toàn bộ bố cục.

Thuộc tính chính:

+ side: Xác định vị trí (TOP, BOTTOM, LEFT, RIGHT).
+ fill: Xác định xem widget có được fill toàn bộ không gian hay không (X, Y, BOTH, NONE).
+ expand: Xác định xem widget có được phép mở rộng khi cửa sổ thay đổi kích thước hay không (True, False).
+ padx, pady: Khoảng cách giữa widget và cạnh của vùng chứa theo chiều ngang và dọc.
+ ipadx, ipady: Khoảng cách giữa nội dung widget và cạnh của widget.

1.3.2. grid ()

Cách thức hoạt động: grid () sắp xếp các widgets theo một lưới (grid) có hàng và cột. Bạn chỉ định vị trí của mỗi widget bằng số hàng và số cột.

Ưu điểm: Cho phép kiểm soát chính xác vị trí và kích thước của các widgets, phù hợp với các bố cục phức tạp. Dễ dàng quản lý bố cục khi thêm hoặc xóa widgets.

Nhược điểm: Cách dùng có thể phức tạp hơn pack () đối với các bố cục đơn giản.

Thuộc tính chính:

+ row, column: Chỉ định hàng và cột cho widget.
+ rowspan, columnspan: Chỉ định số hàng và cột mà widget chiếm.
+ sticky: Xác định cách widget được căn chỉnh trong ô lưới (N, S, E, W, NW, NE, SW, SE).
+ padx, pady: Khoảng cách giữa widget và cạnh của ô lưới.

1.3.3. place ()

Cách thức hoạt động: place () cho phép bạn chỉ định vị trí và kích thước chính xác của mỗi widget bằng tọa độ pixel hoặc phần trăm của cửa sổ.

Ưu điểm: Cho phép kiểm soát hoàn toàn vị trí và kích thước của các widgets.

Nhược điểm: Khó quản lý và bảo trì khi có nhiều widgets, đặc biệt khi cửa sổ thay đổi kích thước. Không tự động điều chỉnh vị trí khi thêm hoặc xóa widgets.

Thuộc tính chính:

+ x, y: Tọa độ x và y của góc trên bên trái của widget.
+ width, height: Chiều rộng và chiều cao của widget.
+ relx, rely: Tọa độ tương đối (phần trăm) của widget so với cửa sổ.
+ anchor: Xác định điểm neo của widget (N, S, E, W, CENTER).

1.3.4. Bí quyết chọn phương pháp phù hợp:

Bố cục đơn giản: pack () là lựa chọn tốt nhất vì sự đơn giản và dễ sử dụng.

Bố cục phức tạp, có cấu trúc: grid () là lựa chọn tốt nhất vì khả năng kiểm soát chính xác và dễ bảo trì.

Vị trí và kích thước chính xác, không quan tâm đến sự thay đổi kích thước cửa sổ thì place () là lựa chọn tốt nhất.

1.4. Các phiên bản python phù hợp

Nói chung, Tkinter hoạt động tốt với các phiên bản Python từ 2.7 trở lên đến các phiên bản Python 3 hiện tại. Bạn sẽ không gặp vấn đề gì khi sử dụng Tkinter với Python 3.7, 3.8, 3.9, 3.10, 3.11 và các phiên bản mới hơn.

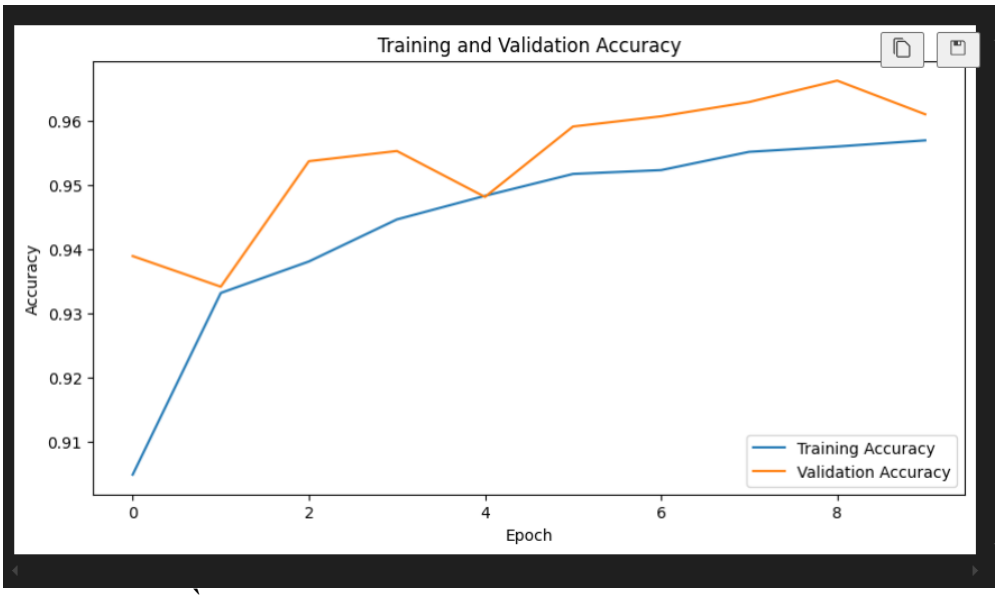
2. Numpy

Numpy là một thư viện Python được sử dụng để làm việc với các mảng. Nó cũng có các chức năng để làm việc trong miền đại số tuyến tính, biến đổi Fourier và ma trận. Numpy được tạo ra vào năm 2005 bởi Travis Oliphant. Đây là một dự án nguồn mở và có thể sử dụng nó một cách tự do. Numpy là viết tắt của Numerical Python.

Phần 3. Kết quả

1. Đồ thị chính xác và mất mát

1.1. Đồ thị chính xác



Hình 26. Đồ thị chính xác của 1 tệp dữ liệu training và xác nhận

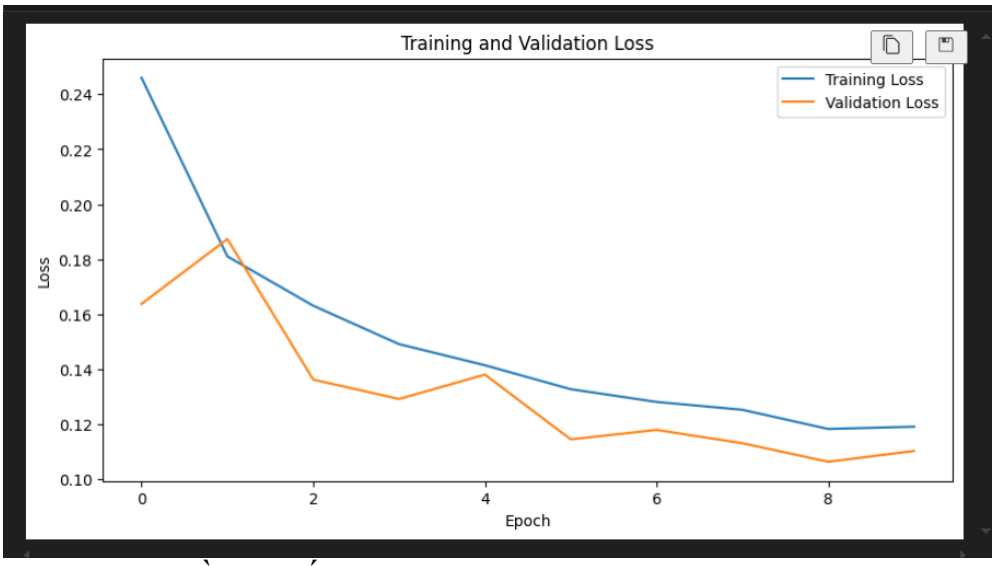
– Cả hai đường accuracy (huấn luyện và kiểm tra) đều tăng dần qua các epoch, cho thấy mô hình đang học hiệu quả và cải thiện dần độ chính xác.

– Đường validation accuracy luôn cao hơn hoặc xấp xỉ so với training accuracy ở hầu hết các epoch. Đây là hiện tượng khá hiếm, thường xảy ra khi dữ liệu kiểm tra dễ hơn dữ liệu huấn luyện hoặc mô hình có sử dụng các kỹ thuật regularization mạnh.
– Đường validation accuracy có một vài điểm dao động nhẹ nhưng nhìn chung khá ổn định và duy trì ở mức cao (trên 0.95).
– Không có dấu hiệu rõ rệt của hiện tượng overfitting (quá khớp), vì accuracy trên tập kiểm tra không giảm khi accuracy trên tập huấn luyện tăng.

Kết luận:

Mô hình đang học tốt, có khả năng tổng quát hóa cao trên dữ liệu kiểm tra. Tuy nhiên, nên kiểm tra lại chất lượng và độ đa dạng của dữ liệu validation để đảm bảo đánh giá mô hình một cách khách quan và chính xác nhất.

1.2. Đồ thị mất mát



Hình 27. Đồ thị mất mát trên 2 tập dữ liệu đào tạo và xác nhận

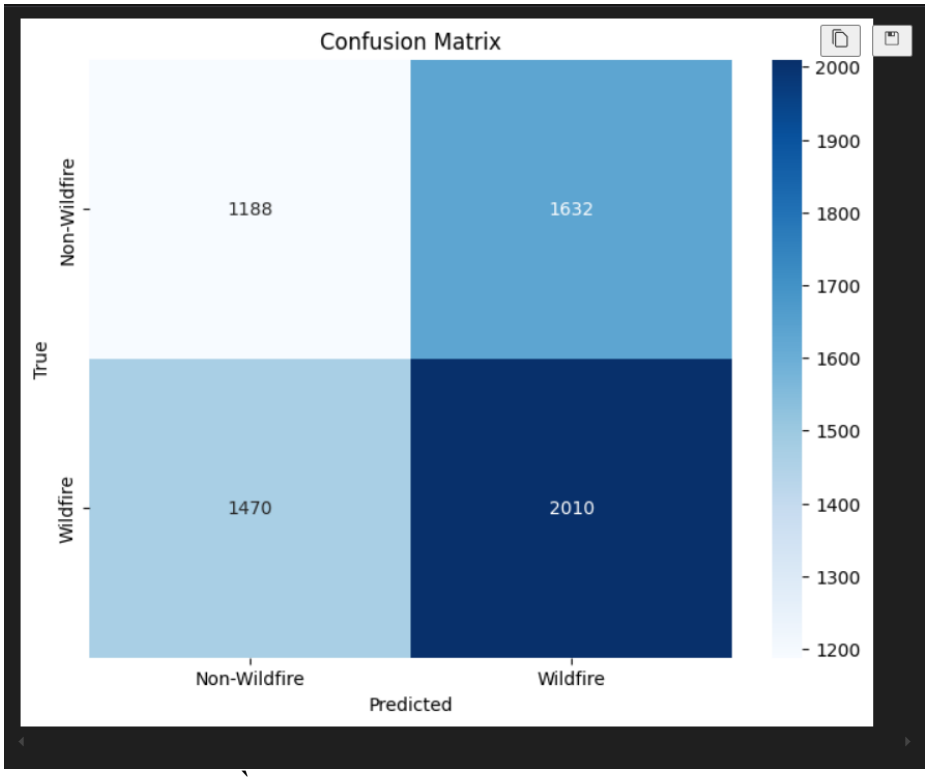
– Cả hai đường loss (huấn luyện và kiểm tra) đều giảm dần qua các epoch, cho thấy mô hình đang học tốt và quá trình tối ưu diễn ra hiệu quả.
– Đường validation loss luôn thấp hơn so với training loss ở hầu hết các epoch. Đây là hiện tượng khá hiếm, thường xảy ra khi dữ liệu validation dễ hơn dữ liệu huấn luyện hoặc có sử dụng các kỹ thuật regularization mạnh.
– Đường validation loss có một vài điểm dao động nhẹ nhưng nhìn chung khá ổn định và tiếp tục giảm về cuối quá trình huấn luyện.

– Không có dấu hiệu rõ rệt của hiện tượng overfitting (quá khớp) hoặc underfitting (chưa học đủ), vì cả hai đường đều giảm và không có khoảng cách lớn giữa chúng.

Kết luận:

Mô hình đang học hiệu quả, không bị quá khớp, và có khả năng tổng quát hóa tốt trên dữ liệu kiểm tra. Tuy nhiên, nên kiểm tra lại chất lượng và độ đa dạng của dữ liệu validation để đảm bảo đánh giá mô hình một cách khách quan nhất.

2. Biểu đồ Confusion matrix



Hình 28. Đồ thị confusion matrix của phân loại 2 lớp

– Số lượng dự đoán đúng cho lớp Wildfire (2010) và non-wildfire (1188) đều khá lớn, tuy nhiên số lượng dự đoán sai cũng không nhỏ.

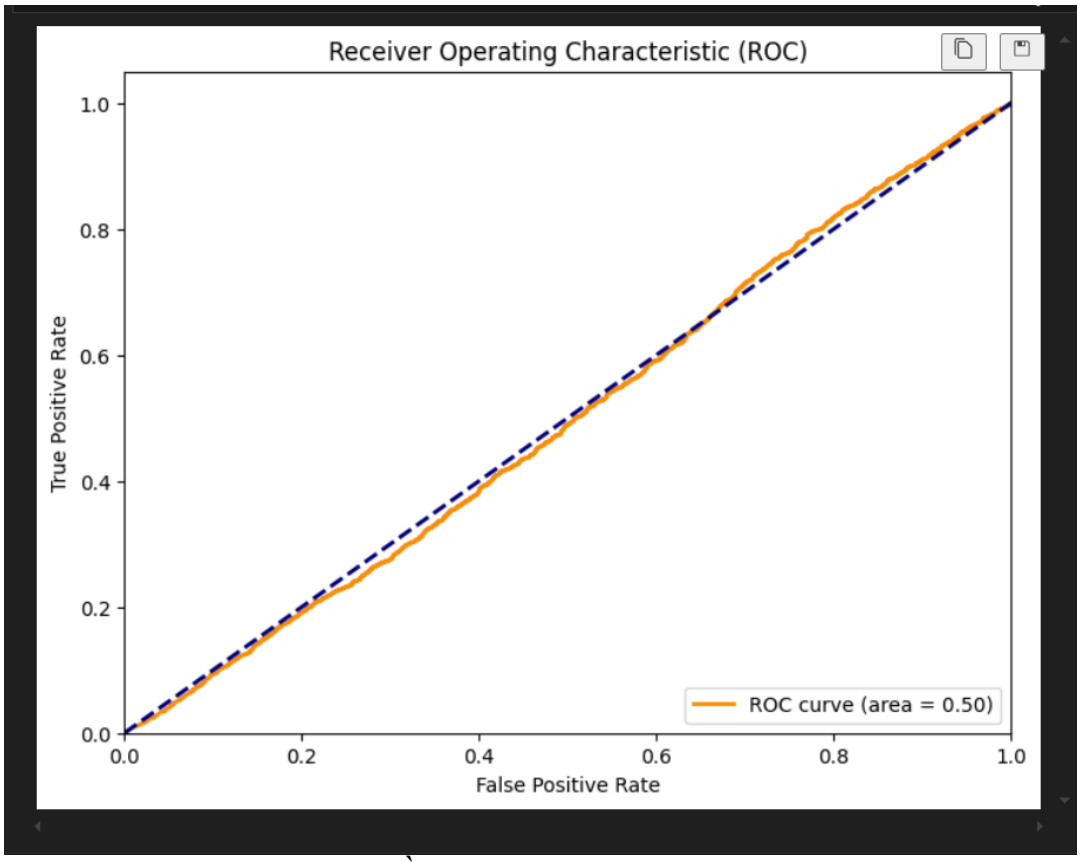
– Mô hình dự đoán nhầm từ non-wildfire sang Wildfire (1632 trường hợp) và từ Wildfire sang non-wildfire (1470 trường hợp) khá nhiều, cho thấy mô hình còn nhầm lẫn giữa hai lớp.

– Tổng số trường hợp dự đoán đúng (1188 + 2010 = 3198) thấp hơn tổng số trường hợp dự đoán sai (1632 + 1470 = 3102), tức là tỷ lệ dự đoán đúng và sai gần như tương đương.

Kết luận:

Mô hình có khả năng nhận diện cả hai lớp nhưng vẫn còn nhầm lẫn khá lớn giữa Wildfire và non-wildfire. Điều này cho thấy mô hình chưa thực sự phân biệt tốt hai lớp, cần cải thiện thêm, có thể bằng cách cân bằng lại dữ liệu, tăng cường dữ liệu huấn luyện, hoặc điều chỉnh lại kiến trúc và tham số mô hình.

3. Đường cong ROC



Hình 29. Đồ thị ROC và AUC của model

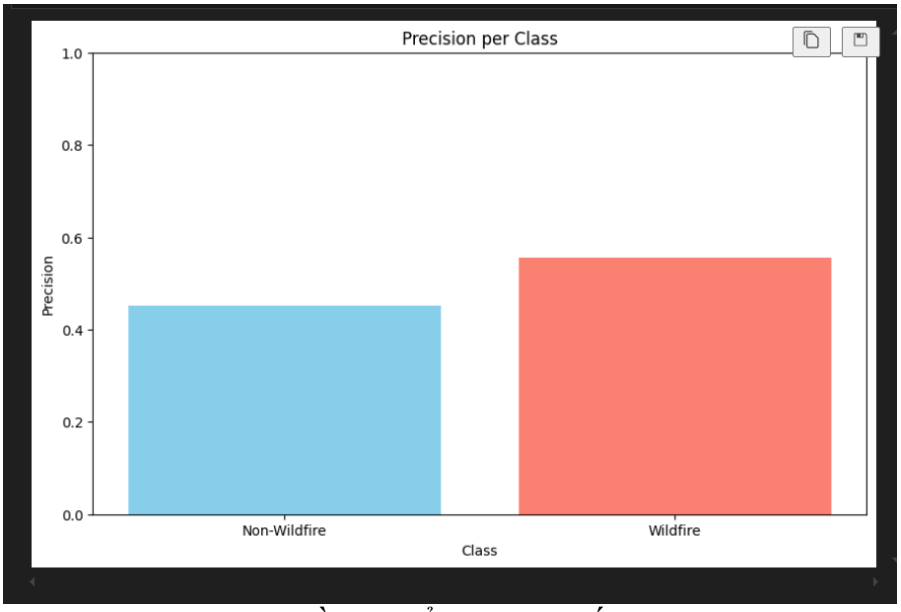
– Đường cong ROC (màu cam) gần như trùng với đường chéo tham chiếu (đường đứt nét màu xanh), với diện tích dưới đường cong (AUC) là 0.50.
– Giá trị AUC = 0.50 cho thấy mô hình không có khả năng phân biệt giữa hai lớp (Wildfire và non-wildfire), hiệu suất tương đương với việc dự đoán ngẫu nhiên.
– Đường ROC không vượt lên trên đường chéo, chứng tỏ mô hình không học được đặc trưng nào hữu ích từ dữ liệu để phân loại.

Kết luận:

Mô hình hiện tại hoàn toàn không có khả năng phân biệt giữa các trường hợp cháy rừng và không cháy rừng nên kiểm tra lại dữ liệu, quá trình huấn luyện, kiến trúc mô hình hoặc các tham số huấn luyện để cải thiện hiệu suất.

4. Đồ thị của precision và recall

4.1. Precision



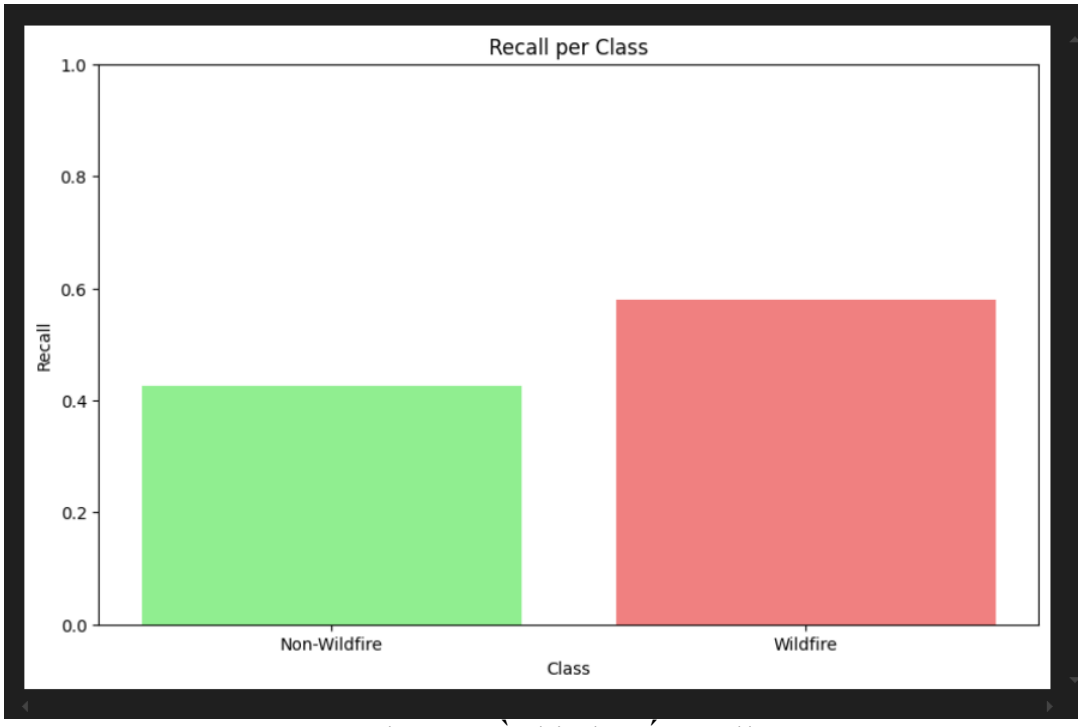
Hình 30. Đồ thị thể hiện chỉ số precision

– Precision (độ chính xác) của lớp Wildfire cao hơn đáng kể so với lớp Non-Wildfire. Điều này cho thấy khi mô hình dự đoán một trường hợp là cháy rừng (Wildfire), xác suất dự đoán đúng cao hơn so với khi dự đoán là không cháy rừng (Non-Wildfire).
– Precision của lớp Non-Wildfire chỉ ở mức trung bình (khoảng 0.45), nghĩa là trong số các dự đoán là Non-Wildfire, chỉ có khoảng 45% là đúng thực sự.
– Precision của lớp Wildfire đạt khoảng 0.56, cao hơn lớp Non-Wildfire nhưng vẫn chưa thực sự tốt, cho thấy mô hình vẫn còn nhầm lẫn một số trường hợp không cháy rừng thành cháy rừng.

Kết luận:

Mô hình dự đoán các trường hợp cháy rừng chính xác hơn so với không cháy rừng, nhưng độ chính xác của cả hai lớp đều chưa cao. Nên xem xét cân bằng lại dữ liệu, cải thiện chất lượng dữ liệu huấn luyện hoặc điều chỉnh mô hình để nâng cao precision, đặc biệt là cho lớp non-wildfire.

4.2. Recall



Hình 31. Đồ thị chỉ số recall

– Recall (độ bao phủ) của lớp Wildfire cao hơn đáng kể so với lớp Non-Wildfire. Điều này cho thấy mô hình nhận diện được nhiều trường hợp cháy rừng hơn so với các trường hợp không cháy rừng.
– Recall của lớp Non-Wildfire chỉ ở mức trung bình (khoảng 0.43), nghĩa là mô hình bỏ sót khá nhiều trường hợp không cháy rừng (nhiều mẫu thực tế Non-Wildfire không được nhận diện đúng).
– Recall của lớp Wildfire đạt khoảng 0.58, cao hơn lớp Non-Wildfire nhưng vẫn chưa thực sự tốt, cho thấy vẫn còn một tỷ lệ nhất định các trường hợp cháy rừng bị bỏ sót.

Kết luận:

Mô hình có khả năng nhận diện cháy rừng tốt hơn không cháy rừng, nhưng hiệu suất tổng thể vẫn còn hạn chế, đặc biệt là với lớp Non-Wildfire nên cân nhắc cải thiện dữ liệu huấn luyện, cân bằng lại các lớp hoặc điều chỉnh mô hình để tăng recall, nhất là cho lớp non-wildfire, nhằm giảm thiểu các trường hợp bị bỏ sót.

5. Nhận diện cháy rừng

Tài liệu tham khảo

[1] https://topdev.vn/blog/tensorflow-la-gi/
[2] https://viblo.asia/p/tai-sao-lai-su-dung-activation-function-trong-neural-network-MG24BwweJz3#_34-relu-8
[3] https://viblo.asia/p/kham-pha-cross-entropy-bi-quyet-giai-ma-ham-mat-mat-trong-deep-learning-PwlVmb8845Z
[4] https://ndquy.github.io/posts/loss-function-p2/#binary-cross-entropy-loss
[5] https://viblo.asia/p/optimizer-hieu-sau-ve-cac-thuat-toan-toi-uu-gdsgdadam-Qbq5QQ9E5D8#_6-adam-7
[6] https://vnptai.io/vi/blog/detail/transfer-learning-la-gi
[7] https://vnptai.io/vi/blog/detail/convolutional-neural-networks-la-gi#section-4-pooling-layer
[8] https://topdev.vn/blog/thuat-toan-cnn-convolutional-neural-network/
[9] https://websitehcm.com/sequential-class-trong-keras/#Kien_truc_cua_Sequential_Class
[10] https://www.miai.vn/2020/06/12/oanh-gia-model-ai-theo-cach-mi-an-lien-chuong-1-loss-va-accuracy/
[11] https://machinelearningcoban.com/2017/08/31/evaluation/
[12] https://machinelearningcoban.com/2017/08/31/evaluation/#-confusion-matrix
[13] https://miai.vn/2020/06/16/oanh-gia-model-ai-theo-cach-mi-an-lien-chuong-2-precision-recall-va-f-score/
[14] https://phamdinhkhanh.github.io/2020/08/13/ModelMetric.html#6-trade-off-gi%E1%BB%AFa-precision-v%C3%A0-recall
[15] https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=vi