ĐẠI HỌC BÁCH KHOA HÀ NỘI
KHOA VẬT LÝ KỸ THUẬT

MÔN HỌC: 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

Giảng viên hướng dẫn: ThS. Bùi Ngọc Hà

Hà Nội, Tháng 7 Năm 2025

Mục lục

Mục lục hình ảnh

I. Cơ sở lý thuyết

1. Thư viện Tensorflow

1.1. 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.

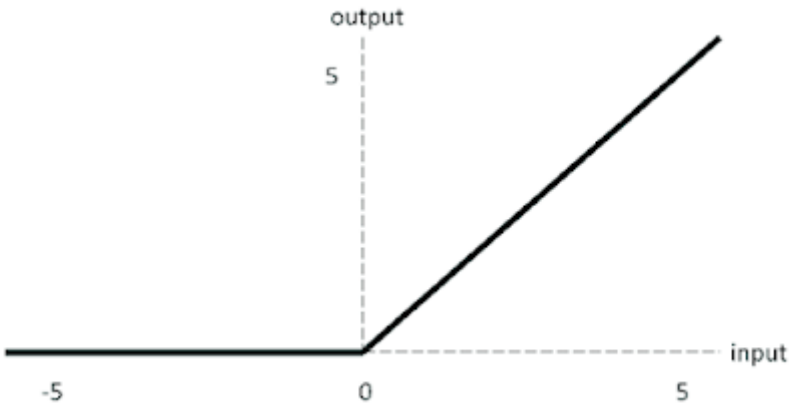
1.2. Cách thức hoạt động

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. 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.

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.

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 một chức năng sang input của một chức năng khác.

2. Hàm kích hoạt RELU



Hình 1. Hàm 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.

3. Hàm mất mát

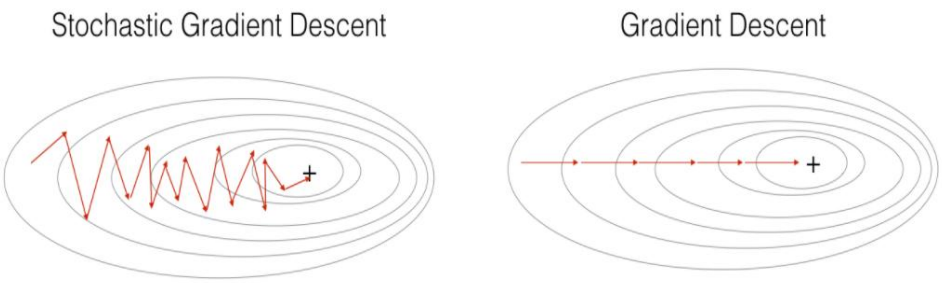
Cross-entropy là hàm loss được sử dụng mặc định cho bài toán phân lớp nhị phân. Nó được thiết kế để sử dụng với bài toán phân loại nhị phân trong đó các giá trị mục tiêu nhận một trong 2 giá trị {0, 1}. Về mặt toán học, nếu như MSE tính khoảng cách giữa 2 đại lượng số thì cross-entropy hiểu nôm na là phương pháp tính khoảng cách giữa 2 phân bố xác suất. Hàm yêu cầu layer đầu ra được gồm 1 node và sử dụng kích hoạt 'sigmoid' để dự đoán xác suất cho đầu ra.

4. Thuật toán tối ưu hóa

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. 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.

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.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.

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.

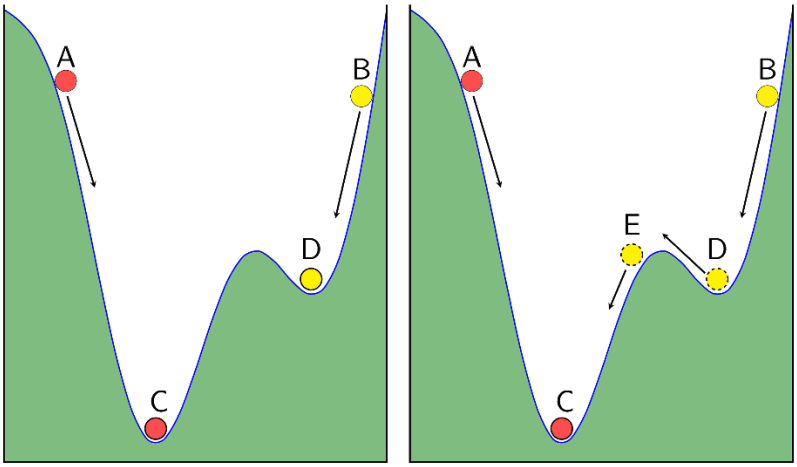


Hình 2. So sánh giữa hai thuật toán tối ưu

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 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.

5. Momentum

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

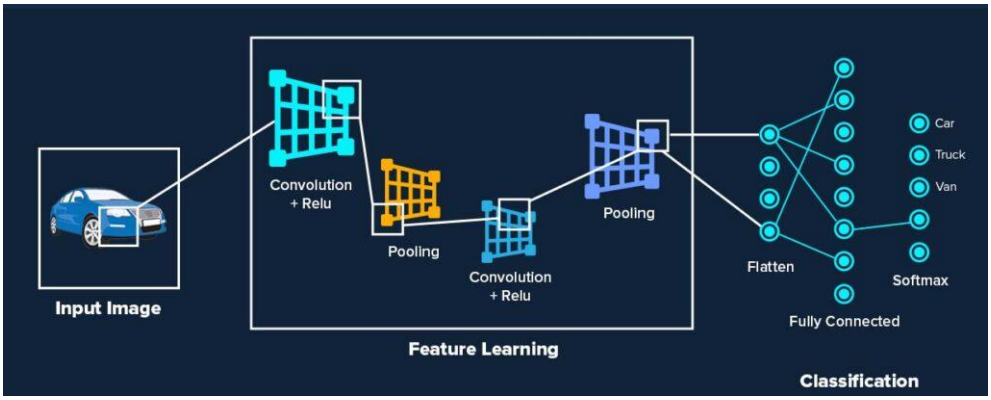


Hình 3. Mô tả cách hoạt động của Momentum

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. 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.

6. Mô hình CNN(Convolutional Neural Networks)

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.



Hình 4: Quy trình xử lý của mạng CNN

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.

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.

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.).

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.

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. 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.

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

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.

7. Đánh giá mô hình

7.1. Accuracy và loss

Trong thực tế train model, người ta hay plot loss và accuracy lên trên biểu đồ để trực quan hóa 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.

7.2. Ma trận nhầm lẫn

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.

7.3. 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. 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: 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. 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.

7.4. F1-score và đường cong ROC

F1-score, là harmonic mean của precision và recall. F1-score có giá trị nằm trong nửa khoảng (0,1]. F1 càng cao, bộ phân lớp càng tốt. Khi cả recall và precision đều bằng 1 (tốt nhất có thể), F1=1.

Đường cong ROC là hình ảnh trực quan về hiệu suất của mô hình trên tất cả các ngưỡng. 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. 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.

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.

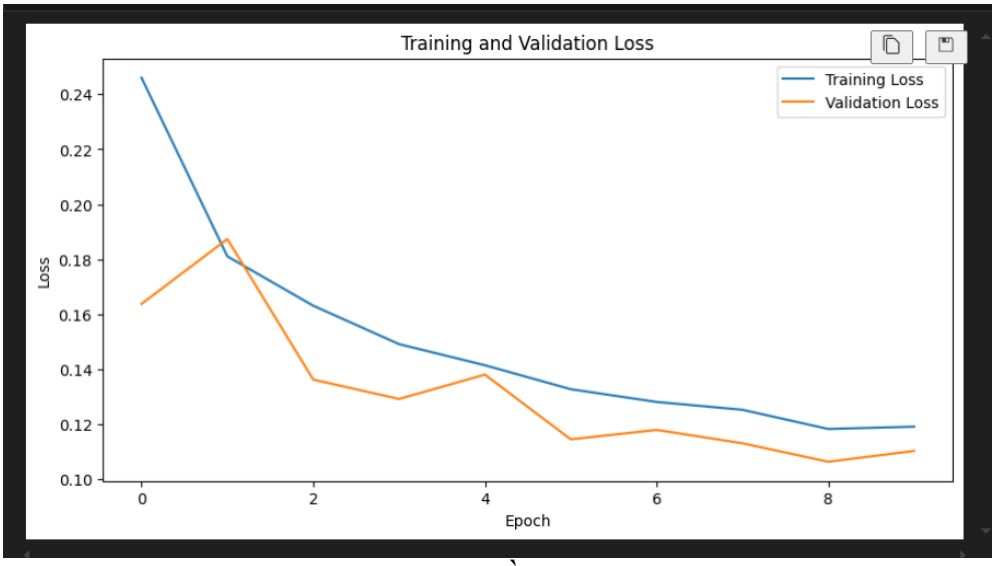
II. Kết quả và thảo luận

1. Accuracy và loss

Hình 4. Đồ thị acccuracy và loss

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.

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.



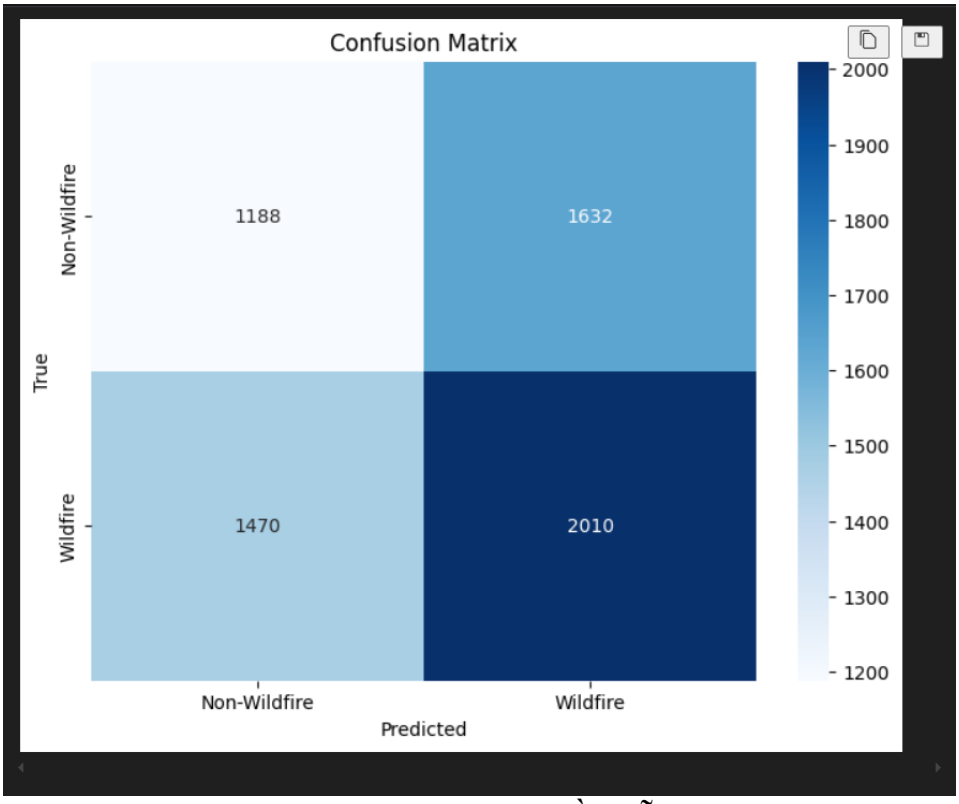
Hình 5. Đồ thị loss

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.

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. Ma trận nhầm lẫn

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. 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.



Hình 6. Ma trận nhầm lẫn

3. Kết quả thực tế

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.

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