HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

TIỂU LUẬN

Học phần: Thực tập cơ sở

Nội dung: Học sâu - khái niệm, các kỹ thuật và ứng dụng

Giảng viên hướng dẫn:	PGS.TS Trần Đình Quế
Họ và tên:	Đặng Quốc Khánh
Mã sinh viên:	B22DCCN444
Lớp hành chính:	D22CQCN12-B

HÀ NỘI – THÁNG 04 NĂM 2025

MỤC LỤC

CHƯƠNG I: TRÍ TUỆ NHÂN TẠO VÀ ỨNG DỤNG

1.1 Các khái niệm cơ bản

1.1.1 Trí tuệ nhân tạo (Artificial Intelligence – AI)

Trí tuệ nhân tạo có thể được mô tả là việc tự động hóa các nhiệm vụ trí tuệ thường do con người thực hiện. Nói cách khác, đó là việc chúng ta tạo ra các hệ thống có khả năng mô phỏng trí thông minh của con người. Trí tuệ nhân tạo do con người tạo ra và nó có thể học hỏi, suy luận và đưa ra quyết định trực tiếp thay cho con người.

Có hai cách tiếp cận AI, đó là AI biểu tượng và học máy.

+ AI biểu tượng: là việc các lập trình viên tự tay xây dựng một bộ quy tắc rõ ràng và đủ lớn để thao tác và bộ quy tắc này được lưu trữ trong các cơ sở dữ liệu rõ ràng. Ví dụ: chương trình cờ vua được lập trình viên lập trình bằng tay. Cách tiếp cận này phù hợp để giải quyết các vấn đề logic mà các quy tắc của nó được xác định rõ ràng.
+ Học máy: khi mô hình AI biểu tượng bất khả thi trong việc tìm ra các quy tắc để giải quyết các vấn đề phức tạp, mơ hồ hơn như phân loại hình ảnh, nhận diện giọng nói hoặc dịch ngôn ngữ tự nhiên, thì học máy là cách tiếp cận mới, thay thế cho AI biểu tượng.

1.1.2 Học máy (Machine Learning - ML)

Học máy được định hình dựa trên ý kiến: “Máy tính về nguyên tắc có thể được chế tạo để mô phỏng mọi khía cạnh của trí thông minh con người.” Học máy thực hiện như sau: máy tính quan sát dữ liệu đầu vào và các câu trả lời tương ứng, rồi tự tìm ra các quy tắc cần thiết. Điều này có nghĩa là: thay vì được lập trình rõ ràng theo các quy tắc như AI biểu tượng, một hệ thống học máy sẽ được huấn luyện dựa trên các tập dữ liệu. Nó được cung cấp nhiều ví dụ liên quan đến một nhiệm vụ, và nó tìm ra cấu trúc thống kê trong các ví dụ này, cuối cùng cho phép hệ thống đưa ra các quy tắc để tự động hóa nhiệm vụ.

Ví dụ: dựa vào tập dữ liệu thu được nhờ chẩn đoán các bệnh nhân từng đến khám tiểu đường, hệ thống sẽ thu thập dữ liệu, phân tích mối tương quan giữa các chỉ số và kết quả, đưa ra quy tắc để đưa ra dự đoán cụ thể cho bệnh nhân mới xem họ có bị tiểu đường hay không dựa trên các chỉ số đã có.

1.1.3 Việc học các quy tắc và biểu diễn chúng từ dữ liệu đầu vào

Để định nghĩa học sâu và hiểu sự khác biệt giữa học sâu với các phương pháp học máy khác, trước tiên chúng ta cần nắm rõ học máy làm gì. Học máy là

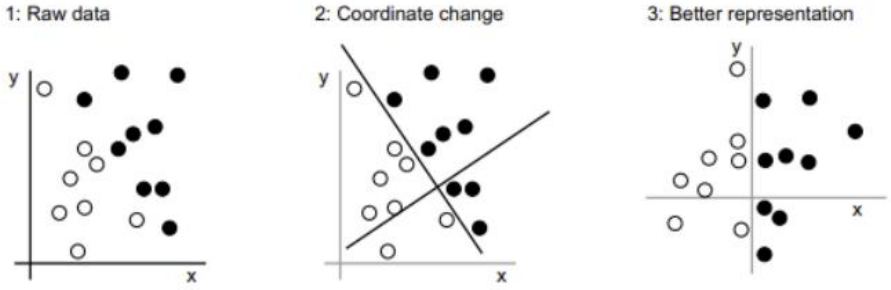
việc khám phá các quy tắc để thực hiện một nhiệm vụ xử lý dữ liệu, dựa trên các ví dụ và kết quả. Vậy để thực hiện học máy, chúng ta cần ba yếu tố:

+ Dữ liệu đầu vào: Ví dụ, nếu nhiệm vụ là nhận diện giọng nói, các dữ liệu này có thể là các tệp âm thanh. Nếu nhiệm vụ là gắn nhãn ảnh, chúng có thể là các bức ảnh.
+ Đầu ra mong đợi - Trong nhiệm vụ nhận diện giọng nói, đó có thể là bản ghi chép được tạo ra từ các tệp âm thanh. Trong nhiệm vụ ảnh, đầu ra mong đợi có thể là các nhãn như "chó", "mèo", v.v.
+ Cách đo lường hiệu quả của thuật toán - Điều này cần thiết để xác định khoảng cách giữa đầu ra hiện tại của thuật toán và đầu ra mong đợi. Số đo này được phản hồi lại để điều chỉnh cách hoạt động của thuật toán. Bước điều chỉnh này chính là cái mà chúng ta gọi là học

Một mô hình học máy biến đổi dữ liệu đầu vào thành các đầu ra có nghĩa, thông qua quá trình "học" từ việc tiếp xúc với các ví dụ đã biết về đầu vào và đầu ra. Do đó, vấn đề cốt lõi trong học máy và học sâu là biến đổi dữ liệu một cách có ý nghĩa, nói cách khác, là việc học các biểu diễn hữu ích của dữ liệu đầu vào mà nó đưa ta gần hơn đến đầu ra mong đợi.

Vậy biểu diễn là gì? Về cơ bản, đó là một cách khác để nhìn nhận dữ liệu để biểu diễn hoặc mã hóa dữ liệu. Chẳng hạn, một bức ảnh màu có thể được mã hóa ở định dạng RGB (đỏ-xanh-lam) hoặc HSV (sắc độ-độ bão hòa-giá trị): đây là hai biểu diễn khác nhau của cùng một dữ liệu. Một số nhiệm vụ khó với biểu diễn này có thể trở nên dễ dàng với biểu diễn khác. Ví dụ, nhiệm vụ "chọn tất cả các pixel màu đỏ trong ảnh" đơn giản hơn ở định dạng RGB (Red – Green - Blue), trong khi "giảm độ bão hòa của ảnh" lại đơn giản hơn ở định dạng HSV. Các mô hình học máy đều xoay quanh việc tìm kiếm các biểu diễn phù hợp cho dữ liệu đầu vào hay còn gọi là những biến đổi của dữ liệu giúp giải quyết nhiệm vụ dễ dàng hơn.

Với những bài toán đơn giản như phân loại các điểm trắng/đen,… chúng ta có thể vẽ được ranh giới giữa các điểm, như bài toán dưới đây:



Tuy nhiên, với những bài toán phức tạp như phân loại hình ảnh, chữ số,… việc biểu diễn không còn đơn giản nữa. Khi đó, chúng ta cần đến học máy. Chúng ta có thể thử tìm kiếm một cách có hệ thống các tập hợp biểu diễn tự động tạo ra từ dữ liệu và các quy tắc dựa trên chúng, rồi xác định cái nào tốt bằng cách dùng phản hồi từ phần trăm kết quả đúng trong một tập dữ liệu. Đó là học máy.

Học, trong bối cảnh học máy, mô tả một quá trình tìm kiếm tự động các biến đổi dữ liệu tạo ra biểu diễn hữu ích của dữ liệu nào đó, được dẫn dắt bởi một tín hiệu phản hồi – những biểu diễn phù hợp với các quy tắc đơn giản hơn để giải quyết nhiệm vụ.

Nói một cách ngắn gọn, học máy là việc tìm kiếm các biểu diễn và quy tắc hữu ích từ dữ liệu đầu vào, trong một không gian khả năng được xác định trước, sử dụng hướng dẫn từ một tín hiệu phản hồi. Ý tưởng đơn giản này cho phép giải quyết một loạt nhiệm vụ trí tuệ đa dạng đáng kinh ngạc, từ nhận diện giọng nói đến lái xe tự động.

1.1.4 “Sâu” trong học sâu

Học sâu là một nhánh cụ thể của học máy: một cách tiếp cận mới để học các biểu diễn từ dữ liệu, nhấn mạnh vào việc học các tầng biểu diễn liên tiếp ngày càng có ý nghĩa hơn.

Chữ “sâu” trong “học sâu” không phải ám chỉ đến bất kỳ sự hiểu biết sâu sắc nào mà phương pháp này đạt được; thay vào đó, nó đại diện cho ý tưởng về các tầng biểu diễn liên tiếp. Số lượng tầng đóng góp vào một mô hình dữ liệu được gọi là độ sâu của mô hình. Nó còn có thể được gọi là học biểu diễn theo tầng hoặc học biểu diễn phân cấp.

Học sâu hiện đại thường liên quan đến hàng chục hoặc thậm chí hàng trăm tầng biểu diễn liên tiếp, và tất cả chúng đều được học một cách tự động thông qua việc tiếp xúc với dữ liệu huấn luyện. Trong khi đó, các phương

pháp học máy khác thường tập trung vào việc học chỉ một hoặc hai tầng biểu diễn của dữ liệu, do đó, chúng đôi khi được gọi là học nông.

Trong học sâu, các biểu diễn theo tầng này được học thông qua các mô hình gọi là mạng nơ-ron, được cấu trúc theo các tầng thực sự chồng lên nhau. Đối với mục đích của chúng ta, học sâu là một thư viện toán học có sẵn để học các biểu diễn từ dữ liệu.

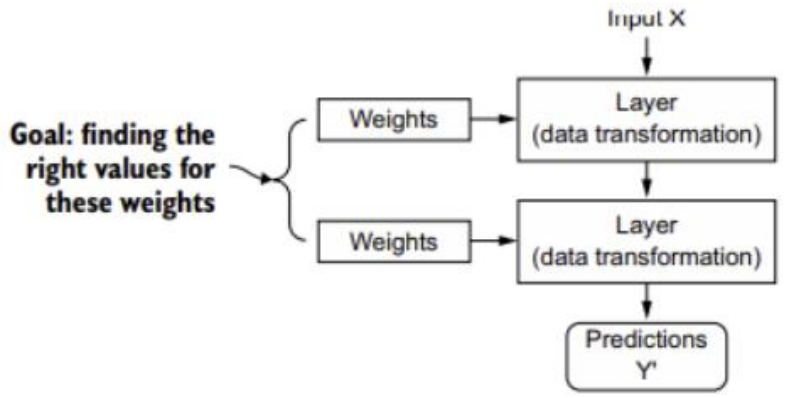
Học sâu như một quá trình chưng cất thông tin nhiều giai đoạn, trong đó thông tin đi qua các bộ lọc liên tiếp và trở nên ngày càng tinh khiết hơn.

1.1.5 Một số thông số cơ bản khác

a) Trọng số (weights)

Mạng nơ-ron sâu là cấu trúc gồm nhiều tầng (layers), mỗi tầng thực hiện một biến đổi dữ liệu. Ví dụ, trong phân loại chữ số, tầng đầu có thể phát hiện cạnh, tầng sau nhận diện hình dạng, và tầng sâu hơn xác định chữ số cụ thể. Các biến đổi này được định nghĩa bởi trọng số (weights), là tập hợp các số lưu trữ thông tin về cách tầng xử lý dữ liệu đầu vào.

Quá trình học trong mạng nơ-ron sâu là tìm tập giá trị trọng số cho tất cả các tầng, sao cho mạng ánh xạ đúng đầu vào ví dụ sang mục tiêu liên quan.

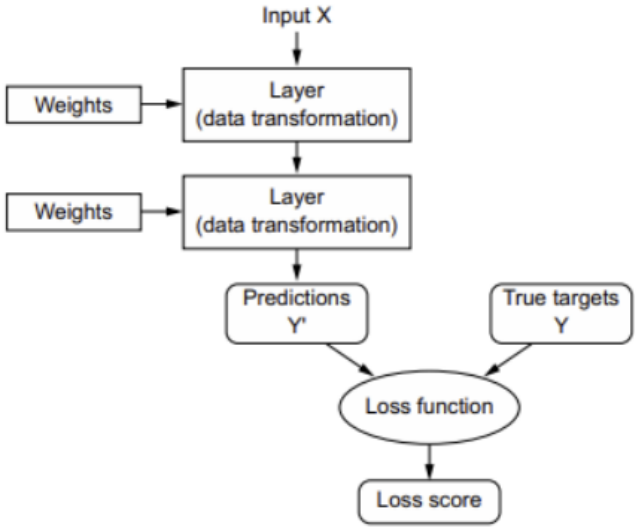


Mô tả hình ảnh: đầu vào X, qua tầng 1 có trọng số W1, sau đó qua tầng 2 có trọng số W2 để ra kết quả Y’.

b) Hàm mất mát (loss function)

Để kiểm soát một thứ gì đó, trước tiên bạn cần phải quan sát được nó. Để kiểm soát đầu ra của một mạng nơ-ron, bạn cần phải đo lường được đầu ra này cách xa bao nhiêu so với kỳ vọng của bạn. Đây là công việc của hàm mất mát của mạng, đôi khi còn được gọi là hàm mục tiêu hoặc hàm chi phí.

Hàm mất mát nhận dự đoán của mạng và mục tiêu đúng, sau đó tính toán điểm số khoảng cách, phản ánh mức độ tốt của mạng trên ví dụ cụ thể.

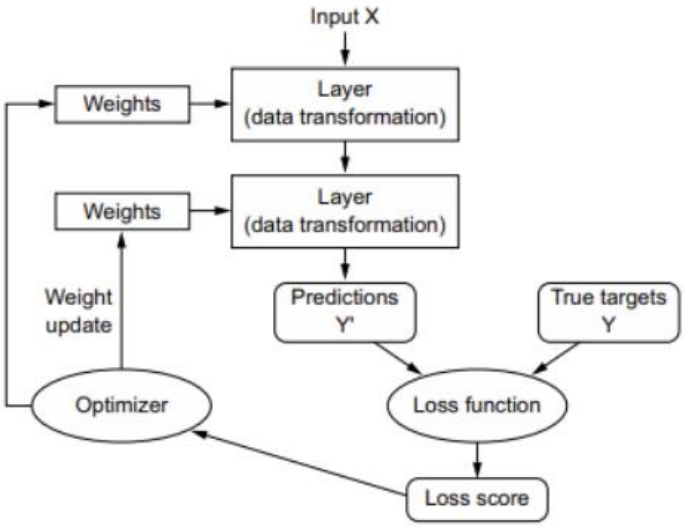


Mô tả hình ảnh: sau khi có dự đoán Y’ ở trên, lấy kết quả chính xác Y để cùng với Y’ đưa vào hàm mất mát (loss function), tính ra điểm mất mát (loss score)

c) Điều chỉnh trọng số nhờ thuật toán Backpropagation

Thủ thuật cơ bản trong học sâu là dùng điểm số mất mát làm tín hiệu phản hồi để điều chỉnh trọng số, theo hướng giảm điểm số mất mát. Việc này do bộ tối ưu hóa thực hiện, sử dụng thuật toán backpropagation, là thuật toán trung tâm trong học sâu.

Backpropagation điều chỉnh trọng số từng chút một, dựa trên đạo hàm của hàm mất mát, để cải thiện dự đoán.



Mô tả hình ảnh: sau khi tính được điểm mất mát, đưa đến bộ tối ưu hóa để cập nhật lại trọng số theo hướng làm giảm điểm mất mát..

d) Vòng lặp huấn luyện và kết quả

Ban đầu, các trọng số của mạng được gán giá trị ngẫu nhiên, nên mạng chỉ thực hiện một loạt các phép biến đổi ngẫu nhiên, do đó điểm số mất mát tương ứng rất cao. Nhưng với mỗi ví dụ được xử lý, các trọng số được điều chỉnh một chút theo hướng đúng, và điểm số mất mát giảm xuống. Đây là vòng lặp huấn luyện.

Thông thường, vòng lặp này lặp lại hàng chục lần trên hàng nghìn ví dụ, cuối cùng tìm ra giá trị trọng số giảm thiểu hàm mất mát. Kết quả là mạng có mất mát tối thiểu, với đầu ra gần nhất có thể với mục tiêu, trở thành mạng đã huấn luyện.

1.1.6 Thành tựu của học sâu

Học sâu đã mang lại những kết quả đáng kinh ngạc trong các nhiệm vụ tri giác và thậm chí cả các nhiệm vụ xử lý ngôn ngữ tự nhiên—những vấn đề liên quan đến các kỹ năng dường như tự nhiên và trực giác đối với con người nhưng từ lâu đã là thách thức lớn đối với máy móc.

Cụ thể, học sâu đã mang lại những bước đột phá sau đây, tất cả đều trong những lĩnh vực vốn khó khăn của học máy:

+ Phân loại hình ảnh gần mức con người

+ Chuyển đổi giọng nói gần mức con người

+ Chuyển đổi chữ viết tay gần mức con người

+ Cải thiện đáng kể dịch máy

+ Cải thiện đáng kể chuyển đổi văn bản thành giọng nói

+ Các trợ lý số như Google Assistant và Amazon Alexa

+ Lái xe tự động gần mức con người

+ Cải thiện nhắm mục tiêu quảng cáo, như được sử dụng bởi Google, Baidu, hoặc Bing

+ Cải thiện kết quả tìm kiếm trên web

+ Khả năng trả lời các câu hỏi bằng ngôn ngữ tự nhiên

+ Chơi cờ vây vượt trội hơn con người

Ngoài ra, học sâu còn giúp con người phát hiện và phân loại bệnh cây trồng trên đồng ruộng chỉ bằng một chiếc điện thoại thông minh đơn giản, hỗ trợ các bác sĩ ung bướu hoặc bác sĩ X-quang trong việc diễn giải dữ liệu hình ảnh y tế, dự đoán các thảm họa tự nhiên như lũ lụt, bão tố, hoặc thậm chí động đất,…

1.**2. Quá trình phát triển** của **học sâu**

1.2.1 Mạng nơ-ron ban đầu

Là bước đệm cho học sâu hiện đại, lấy ý tưởng từ cách bộ não hoạt động.

Tuy nhiên, việc phát triển và ứng dụng thực tế gặp nhiều thách thức, đặc biệt là trong huấn luyện mạng lớn. Điểm đột phá xuất hiện khi thuật toán Backpropagation được tái khám phá, cho phép huấn luyện mạng nhiều tầng bằng tối ưu hóa gradient-descent.

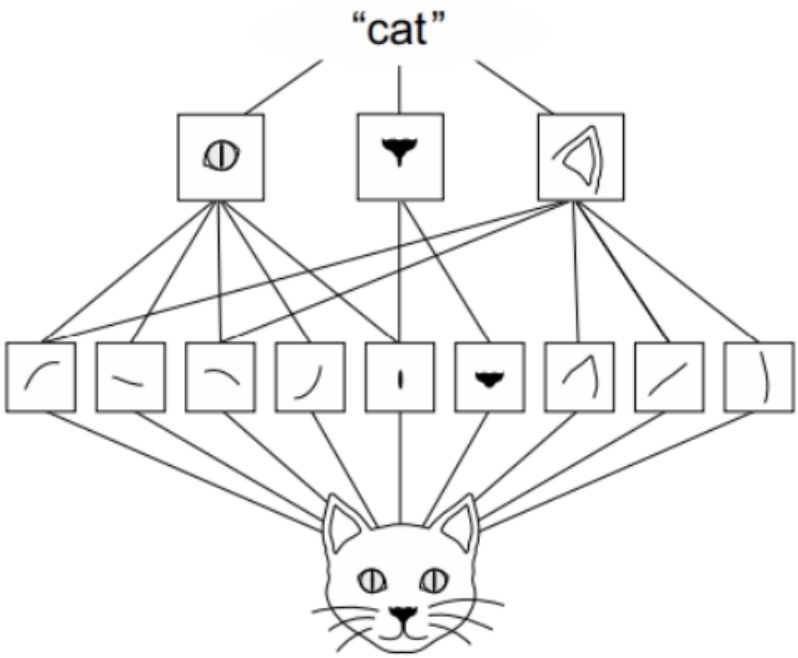
Ứng dụng thực tế đầu tiên thành công của mạng nơ-ron đến từ Bell Labs vào năm 1989, khi Yann LeCun kết hợp các ý tưởng trước đó về mạng nơ-ron tích chập và backpropagation, áp dụng chúng vào vấn đề phân loại chữ số viết tay. Mạng kết quả, được gọi là LeNet, đã được sử dụng để tự động đọc mã ZIP trên phong bì thư.

1.2.2 Mạng nơ-ron hiện đại

Hiện nay, người ta sử dụng ConvNets (convolutional networks). ConvNets là loại mạng nơ-ron chuyên biệt để xử lý dữ liệu dạng lưới, như hình ảnh hoặc chuỗi thời gian. Chúng sử dụng các lớp tích chập (convolutional layers như Conv1D, Conv2D,…) để áp dụng bộ lọc lên dữ liệu đầu vào, học các mẫu

cục bộ như cạnh hoặc góc trong hình ảnh được tạo ra bằng các cửa sổ nhỏ của đầu vào, trượt dần cho đến hết mẫu. Sau đó, các lớp gộp (pooling layers như MaxPooling1D, MaxPooling2D,…) để giảm chiều dữ liệu để giảm tính toán và tránh quá khớp. Cuối cùng, các lớp kết nối đầy đủ (fully connected layers như Dense,…) đưa ra dự đoán cuối cùng, chẳng hạn như phân loại hình ảnh thành "mèo" hoặc "chó".

ConvNets học biểu diễn phân cấp, từ các đặc trưng cơ bản đến phức tạp, giống như cách mắt người nhận diện hình ảnh: bắt đầu từ các cạnh, sau đó là hình dạng, và cuối cùng là toàn bộ đối tượng. Chúng đã cách mạng hóa thị giác máy tính, đạt độ chính xác cao trong các nhiệm vụ như nhận diện khuôn mặt, phân loại hình ảnh, và phát hiện đối tượng. Ngoài ra, ConvNets cũng được áp dụng cho các dữ liệu khác, như xử lý ngôn ngữ tự nhiên (sử dụng các biến thể như TextCNN) hoặc phân tích chuỗi thời gian.



Đây là hình ảnh mô tả cách Convnets học theo sự phân cấp không gian của hình ảnh. Ví dụ, như con mèo, đầu tiên, nó sẽ học những mẫu nhỏ là các đường nét, cạnh của các bộ phận, sau đó, ghép các nét lại thành một bộ phận hoàn chỉnh, cuối cùng là từ các bộ phận, ghép lại thành con mèo.

Các đặc điểm của mạng tích chập ConvNets mang lại cho nó hai tính chất sau:

+ Các mẫu mà chúng học được có tính bất biến với phép dịch chuyển:

  - Nếu Convnets học được 1 mẫu ở góc dưới bên phải của một bức ảnh, nó có thể nhận diện mẫu đó ở bất kỳ đâu: ví dụ, ở góc trên bên trái. Trong khi đó, một mô hình kết nối đầy đủ sẽ phải học lại mẫu đó từ đầu nếu nó xuất hiện ở một vị trí mới.
  - Điều này khiến các Convnets trở nên hiệu quả về mặt dữ liệu khi xử lý hình ảnh (vì thế giới thị giác vốn dĩ có tính bất biến với phép dịch chuyển): chúng cần ít mẫu huấn luyện hơn để học được các biểu diễn có khả năng khái quát hóa tốt.
+ Chúng có thể học theo hệ thống phân cấp không gian của các mẫu:
  - Một tầng tích chập đầu tiên sẽ học các mẫu cục bộ nhỏ như các cạnh (edges), một tầng tích chập thứ hai sẽ học các mẫu lớn hơn được tạo thành từ các đặc trưng của các tầng trước, và cứ tiếp tục như vậy.
  - Điều này cho phép các mạng tích chập học một cách hiệu quả các khái niệm thị giác ngày càng phức tạp và trừu tượng, bởi vì thế giới thị giác vốn dĩ có tính phân cấp không gian

1.2.3 Điều làm học sâu trở nên khác biệt

Lý do chính khiến học sâu (deep learning) phát triển nhanh chóng là vì nó mang lại hiệu suất tốt hơn cho nhiều bài toán. Nhưng đó không phải là lý do duy nhất. Học sâu cũng giúp việc giải quyết vấn đề trở nên dễ dàng hơn, bởi nó tự động hóa hoàn toàn bước quan trọng nhất trong quy trình học máy truyền thống: kỹ thuật đặc trưng (feature engineering).

Các kỹ thuật học máy trước đây (học nông) chỉ bao gồm việc biến đổi dữ liệu đầu vào thành một hoặc hai không gian biểu diễn liên tiếp, thường thông qua các phép biến đổi đơn giản như SVM hoặc cây quyết định. Nhưng các biểu diễn cần thiết cho các bài toán phức tạp thường không thể đạt được bằng những kỹ thuật này. Do đó, con người phải nỗ lực rất nhiều để làm cho dữ liệu đầu vào dễ xử lý hơn. Họ phải tự tay thiết kế các tầng biểu diễn cho dữ liệu của mình. Quá trình này được gọi là kỹ thuật đặc trưng. Ngược lại, học sâu tự động hóa hoàn toàn bước này: với học sâu, bạn học tất cả các đặc trưng trong một lần duy nhất thay vì phải tự thiết kế chúng. Điều này đã đơn giản hóa đáng kể quy trình học máy, thường thay thế các pipeline nhiều giai đoạn phức tạp bằng một mô hình học sâu đơn giản, end-to-end.

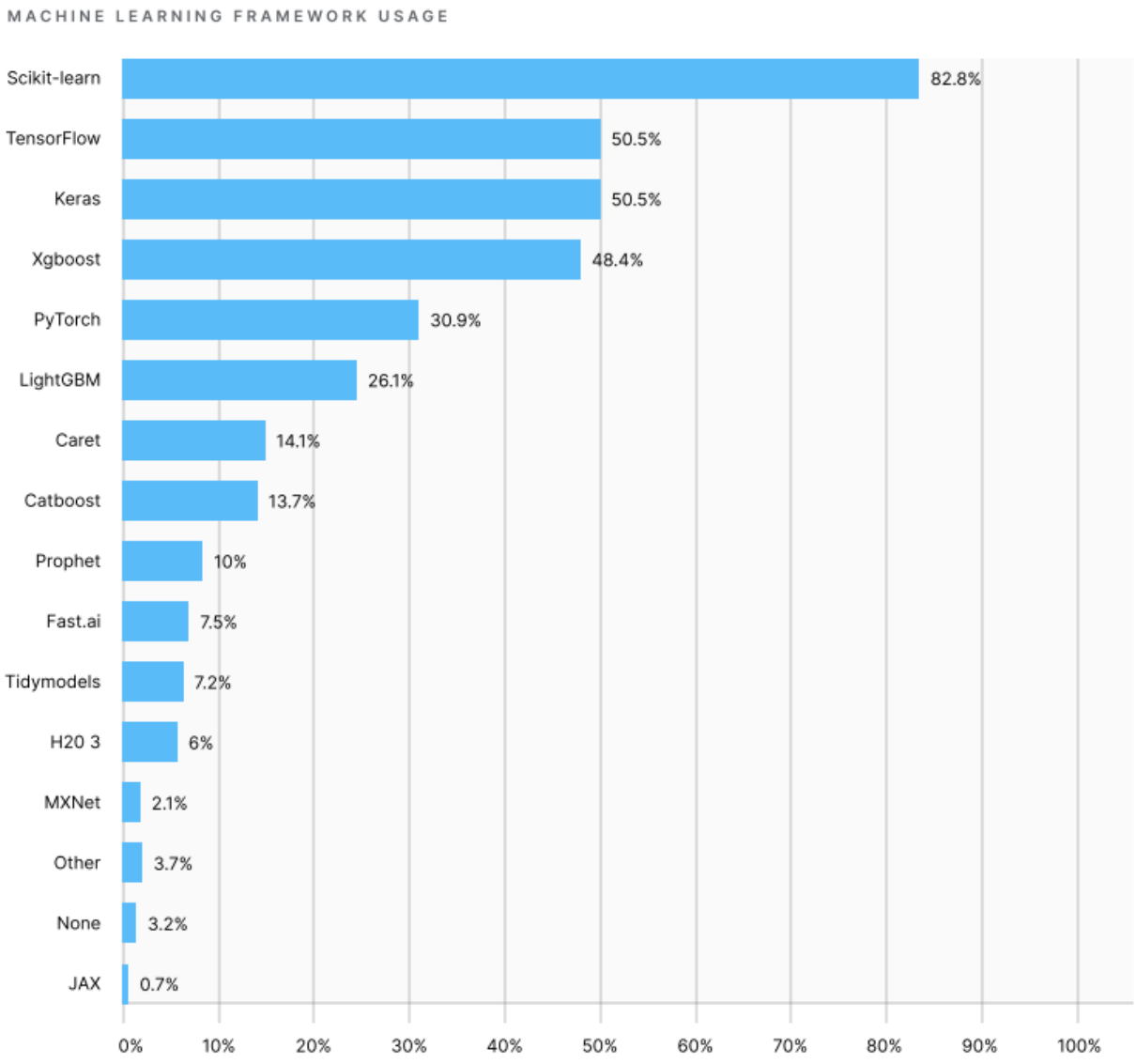
Nếu vấn đề cốt lõi là có nhiều tầng biểu diễn liên tiếp, liệu các phương pháp học nông có thể được áp dụng lặp đi lặp lại để mô phỏng hiệu ứng của học sâu không? Thực tế, việc áp dụng liên tiếp các phương pháp học chỉ làm hiệu

quả giảm nhanh, bởi vì tầng biểu diễn đầu tiên tối ưu trong một mô hình ba tầng không phải là tầng đầu tiên tối ưu trong một mô hình một hoặc hai tầng. Điều mang tính cách mạng của học sâu là nó cho phép một mô hình học tất cả các tầng biểu diễn cùng lúc, thay vì học từng tầng một cách tham lam (greedy). Với việc học đặc trưng chung, mỗi khi mô hình điều chỉnh một đặc trưng nội bộ, tất cả các đặc trưng khác phụ thuộc vào nó sẽ tự động thích nghi với thay đổi mà không cần sự can thiệp của con người. Mọi thứ được giám sát bởi một tín hiệu phản hồi duy nhất: mọi thay đổi trong mô hình đều phục vụ mục tiêu cuối cùng. Điều này mạnh mẽ hơn nhiều so với việc xếp chồng các mô hình học nông một cách tham lam, bởi vì nó cho phép học các biểu diễn phức tạp, trừu tượng bằng cách chia nhỏ chúng thành một chuỗi dài các không gian trung gian (tầng); mỗi không gian chỉ là một phép biến đổi đơn giản từ không gian trước đó.

Đây là hai đặc điểm cốt lõi của cách học sâu học từ dữ liệu: cách thức từng bước, tầng-bởi-tầng mà các biểu diễn ngày càng phức tạp được phát triển, và việc các biểu diễn trung gian này được học cùng lúc, mỗi tầng được cập nhật để đáp ứng cả nhu cầu biểu diễn của tầng trên và tầng dưới. Cùng nhau, hai đặc tính này đã khiến học sâu thành công vượt trội hơn nhiều so với các phương pháp học máy trước đây.

1.2.4 Tổng quan về học máy hiện đại

Chúng ta cùng xem xét thống kế sau đây của Kaggle, cho thấy tỷ lệ sử dụng của các framework phần mềm học máy khác nhau.



Từ năm 2016 đến 2020, toàn bộ ngành học máy và khoa học dữ liệu đã bị chi phối bởi hai phương pháp: học sâu và cây tăng cường gradient. Cụ thể, cây tăng cường gradient được sử dụng cho các bài toán có dữ liệu cấu trúc, thông qua Scikit-learn, XGBoots hoặc LightGBM, trong khi học sâu được sử dụng cho các bài toán nhận thức (perceptual problems) như phân loại hình ảnh, sử dụng Keras, thường kết hợp với framework mẹ của nó là TensorFlow.

=> Điểm chung của các công cụ này là chúng đều là các thư viện Python: Python là ngôn ngữ được sử dụng rộng rãi nhất cho học máy và khoa học dữ liệu.

1.3 Những yếu tố ảnh hưởng đến sự phát triển của học sâu

Học sâu chỉ thực sự bùng nổ sau 2012, vậy tại sao lại như vậy, và điều gì đã xảy ra trong thời gian này? Thực tế, có ba yếu tố kỹ thuật đang thúc đẩy sự tiến bộ trong học máy:

+ Phần cứng (Hardware)
+ Tập dữ liệu và chuẩn đánh giá (Datasets and benchmarks)
+ Tiến bộ thuật toán (Algorithmic advances)

Vậy tại sao lại cần ba yếu tố này? Lý do vì lĩnh vực này được dẫn dắt bởi các kết quả thực nghiệm hơn là lý thuyết, các tiến bộ thuật toán chỉ có thể xảy ra khi có dữ liệu và phần cứng phù hợp để thử nghiệm các ý tưởng mới (hoặc mở rộng quy mô các ý tưởng cũ), chứ không đơn thuần là sử dụng bút và giấy để nghiên cứu như toán học hay vật lý. Nó là 1 ngành khoa học kỹ thuật.

1.3.1 Phần cứng (Hardware)

Từ năm 1990 đến 2010, các CPU thông thường đã nhanh hơn khoảng 5.000 lần. Kết quả là, ngày nay bạn có thể chạy các mô hình học sâu nhỏ trên laptop của mình, điều không thể thực hiện được cách đây 25 năm.

Nhưng các mô hình học sâu điển hình dùng trong thị giác máy tính hoặc nhận diện giọng nói yêu cầu sức mạnh tính toán lớn hơn nhiều so với khả năng của laptop. Trong suốt những năm 2000, các công ty như NVIDIA và AMD đã đầu tư hàng tỷ đô-la để phát triển các chip nhanh, song song hóa cao (hay còn gọi là GPU) nhằm phục vụ đồ họa cho các trò chơi điện tử - những siêu máy tính giá rẻ, chuyên dụng được thiết kế để dựng các cảnh 3D phức tạp trên màn hình của bạn theo thời gian thực. Một số lượng nhỏ GPU bắt đầu thay thế các cụm CPU lớn trong các ứng dụng có khả năng song song hóa cao, bắt đầu từ mô phỏng vật lý.

Điều xảy ra là thị trường trò chơi điện tử đã tài trợ cho siêu máy tính cho thế hệ tiếp theo của các ứng dụng trí tuệ nhân tạo. Đôi khi, những điều lớn lao bắt đầu từ trò chơi. Ngày nay, NVIDIA Titan RTX, một GPU có giá 2.500 USD vào cuối năm 2019, có thể cung cấp hiệu suất tối đa 16 teraFLOPS ở độ chính xác đơn (16 nghìn tỷ phép toán float32 mỗi giây). Đó là sức mạnh tính toán gấp khoảng 500 lần so với siêu máy tính nhanh nhất thế giới năm 1990, Intel Touchstone Delta. Trên Titan RTX, chỉ mất vài giờ để huấn luyện một mô hình ImageNet tương tự như loại đã giành chiến thắng trong cuộc thi ILSVRC vào khoảng năm 2012 hoặc 2013. Trong khi đó, các công ty lớn huấn luyện mô hình học sâu trên cụm hàng trăm GPU.

Hơn nữa, ngành học sâu đã vượt qua GPU và đang đầu tư vào các chip chuyên dụng, hiệu quả hơn cho học sâu. Vào năm 2016, tại hội nghị I/O 22 hàng năm, Google đã công bố dự án Tensor Processing Unit (TPU): một thiết kế chip mới được phát triển từ đầu để chạy mạng nơ-ron sâu nhanh hơn đáng kể và tiết kiệm năng lượng hơn nhiều so với GPU hàng đầu. Đến năm 2020, phiên bản thứ ba của thẻ TPU đạt hiệu suất 420 teraFLOPS. Đó là sức mạnh tính toán gấp 10.000 lần so với Intel Touchstone Delta năm 1990.

1.3.2 Dữ liệu (Data)

Trí tuệ nhân tạo (AI) đôi khi được ca ngợi là cuộc cách mạng công nghiệp mới. Nếu học sâu là động cơ hơi nước của cuộc cách mạng này, thì dữ liệu là than đá: nguyên liệu thô cung cấp năng lượng cho các máy thông minh của chúng ta. Về dữ liệu, ngoài sự tiến bộ vượt bậc trong phần cứng lưu trữ trong 20 năm qua (theo định luật Moore), yếu tố thay đổi cuộc chơi là sự bùng nổ của internet, giúp việc thu thập và phân phối các tập dữ liệu lớn cho học máy trở nên khả thi. Ngày nay, các công ty lớn làm việc với các tập dữ liệu hình ảnh, video và ngôn ngữ tự nhiên mà không thể thu thập được nếu không có internet. Các thẻ hình ảnh do người dùng tạo trên Flickr, ví dụ, đã là một kho báu dữ liệu cho thị giác máy tính. Video trên YouTube cũng vậy. Và Wikipedia là một tập dữ liệu quan trọng cho xử lý ngôn ngữ tự nhiên.

Nếu có một tập dữ liệu đã làm chất xúc tác cho sự phát triển của học sâu, thì đó là tập dữ liệu ImageNet, bao gồm 1,4 triệu hình ảnh được gắn nhãn thủ công với 1.000 danh mục hình ảnh (mỗi hình ảnh một danh mục). Nhưng điều làm ImageNet đặc biệt không chỉ là kích thước lớn, mà còn là cuộc thi hàng năm liên quan đến nó. Việc có các chuẩn đánh giá chung mà các nhà nghiên cứu cạnh tranh để vượt qua đã giúp rất nhiều cho sự phát triển của học sâu, bằng cách làm nổi bật thành công của nó so với các phương pháp học máy cổ điển.

1.3.3 Thuật toán (Algorithms)

Ngoài phần cứng và dữ liệu, cho đến cuối những năm 2000, chúng ta vẫn thiếu một cách đáng tin cậy để huấn luyện các mạng nơ-ron rất sâu. Kết quả là, các mạng nơ-ron vẫn còn khá nông, chỉ sử dụng một hoặc hai tầng biểu diễn; do đó, chúng không thể vượt trội hơn các phương pháp học nông như SVM và random forests. Vấn đề cốt lõi là lan truyền gradient qua các tầng sâu. Tín hiệu phản hồi dùng để huấn luyện mạng nơ-ron sẽ mờ dần khi số lượng tầng tăng lên.

Điều này đã thay đổi vào khoảng năm 2009–2010 với sự xuất hiện của một số cải tiến thuật toán đơn giản nhưng quan trọng, cho phép lan truyền gradient tốt hơn:

+ Hàm kích hoạt tốt hơn cho các tầng nơ-ron.
+ Phương pháp khởi tạo trọng số tốt hơn, bắt đầu với việc huấn luyện trước từng tầng (layer-wise pretraining), sau đó nhanh chóng bị bỏ qua.
+ Phương pháp tối ưu hóa tốt hơn, như RMSProp và Adam.

Chỉ khi những cải tiến này bắt đầu cho phép huấn luyện các mô hình với 10 tầng trở lên, học sâu mới bắt đầu tỏa sáng.

Cuối cùng, vào các năm 2014, 2015 và 2016, các phương pháp tiên tiến hơn để cải thiện lan truyền gradient đã được phát hiện, như chuẩn hóa hàng loạt (batch normalization), kết nối dư (residual connections), và tích chập phân tách theo chiều sâu (depthwise separable convolutions).

Ngày nay, chúng ta có thể huấn luyện các mô hình sâu tùy ý từ đầu. Điều này đã mở ra việc sử dụng các mô hình cực lớn, có sức mạnh biểu diễn đáng kể—tức là mã hóa các không gian giả thuyết rất phong phú. Khả năng mở rộng cực lớn này là một trong những đặc điểm nổi bật của học sâu hiện đại. Các kiến trúc mô hình quy mô lớn, với hàng chục tầng và hàng chục triệu tham số, đã mang lại những tiến bộ quan trọng trong cả thị giác máy tính (ví dụ: các kiến trúc như ResNet, Inception, hoặc Xception) và xử lý ngôn ngữ tự nhiên (ví dụ: các kiến trúc dựa trên Transformer lớn như BERT, GPT-3, hoặc XLNet).

1.3.4 Một vài yếu tố khác

Học sâu bùng nổ sau 2012 nhờ làn sóng đầu tư lớn từ ngành công nghiệp (tăng từ dưới 1 tỷ USD năm 2011 lên 16 tỷ USD năm 2017), sự dân chủ hóa công cụ (như Keras, TensorFlow, Theano giúp dễ tiếp cận hơn), và các đặc tính vượt trội: tính đơn giản (loại bỏ feature engineering), khả năng mở rộng (song song hóa trên GPU/TPU), và tính linh hoạt (tái sử dụng mô hình). Học sâu là một cuộc cách mạng AI lâu dài. Đến năm 2021, học sâu đã qua giai đoạn bùng nổ ban đầu, nhưng vẫn còn tiềm năng lớn trong tương lai.

CHƯƠNG II: CÁC KỸ THUẬT CẦN BIẾT TRONG HỌC SÂU

2.1 Biểu diễn dữ liệu bằng tensor

Tất cả các hệ thống học máy hiện tại đều sử dụng tensor làm cấu trúc dữ liệu cơ bản của chúng. Tensor là nền tảng của lĩnh vực này - nền tảng đến mức TensorFlow được đặt tên theo chúng. Vậy tensor là gì?

Về cốt lõi, tensor là một thùng chứa dữ liệu, thường là dữ liệu số. Vì vậy, nó là một thùng chứa cho các con số. Ví dụ: với ma trận, nó là tensor bậc 2: tensor là một khái quát hóa của ma trận cho một số chiều bất kỳ.

2.1.1 Tensor bậc n

+ Tensor bậc 0: hay còn gọi là vô hướng, ví dụ như 1 số thực
+ Tensor bậc 1: là vector, biểu diễn trong NumPy ở dạng np.array, ví dụ: x = np.array([1,2,3,4,5]), là tensor bậc 1, và là vector 5 chiều (do x có 5 phần tử)
+ Tensor bậc 2: là ma trận
+ Tensor bậc 3 và cao hơn (n chiều): tương tự ma trận 3 và n chiều.

2.1.2 Các thông số cơ bản của tensor

Giả sử, chúng ta đang dùng tensor ts. Các thuộc tính cơ bản của tensor bao gồm:

+ Số trục, đại diện cho bậc của tensor, có thể dùng thuộc tính ndim để lấy ra số trục của tensor đó. Ví dụ: print(ts.ndim)
+ Hình dạng, đại diện cho số mẫu và cách biểu diễn mẫu đó (sẽ tìm hiểu rõ hơn ở sau). Có thể dùng thuộc tính shape để lấy ra hình dạng của tensor đó. Ví dụ: print(ts.shape)
+ Kiểu dữ liệu: là kiểu dữ liệu mà dùng để biểu diễn cho mỗi điểm của dữ liệu, có thể dùng thuộc tính dtype để lấy ra kiểu dữ liệu của tensor đó. Ví dụ:print(ts.dtype). Dtype có thể là int32, int64, float64,…

2.1.3 Các lô dữ liệu

Mô hình học sâu không xử lý toàn bộ tập dữ liệu cùng một lúc; thay vào đó, chúng chia dữ liệu thành các lô nhỏ

Ví dụ: 1 lô lấy 128 ảnh, thì lô 1 là train_images[0:128], lô 2 là [128:256], lô thứ n là [128*n : 128*(n+1)]

Trục đầu tiên (trục 0, vì chỉ số bắt đầu từ 0) trong tất cả các tensor dữ liệu mà bạn sẽ gặp trong học sâu được gọi là trục mẫu

2.1.4 Một số ví dụ về tensor trong thực tế

+ Vector dữ liệu: tensor bậc 2, chứa các số mẫu và đặc trưng (ví dụ: tensor có 20 mẫu và mỗi mẫu có 5 giá trị biểu diễn nó, thì tensor có shape là (20,5))
+ Dữ liệu chuỗi thời gian, dữ liệu tuần tự: tensor bậc 3, gồm số mẫu, số bước nhảy thời gian và số đặc trưng
+ Hình ảnh: tensor bậc 4: gồm số mẫu. chiều cao, chiều rộng, số kênh (số kênh là cách biểu diễn màu: ví dụ thang độ xám, giá trị từ 0-255 thì có 1 kênh, màu RGB có 3 kênh (giá trị red ,giá trị green, giá trị blue), màu RGB thêm độ trong suốt thì có 4 kênh). Ví dụ: dữ liệu gồm 10000 ảnh, mỗi ảnh có kích thước 32x32, và dùng kênh màu RGB, thì tensor biểu diễn sẽ có shape là (10000,32,32,3)
+ Video: tensor bậc 5, giống hình ảnh, nhưng có thêm giá trị số khung hình/ giây (FPS)

2.1.5 Dữ liệu vector

Đây là dạng dữ liệu phổ biến nhất. Trong một tập dữ liệu, mỗi điểm dữ liệu đơn lẻ có thể được mã hóa thành một vector, và do đó, một lô dữ liệu sẽ được mã hóa thành một tensor bậc 2 (tức là một mảng các vector), trong đó trục đầu tiên là trục mẫu (samples axis) và trục thứ hai là trục đặc trưng (features axis).

Ví dụ: Một tập dữ liệu bảo hiểm về con người, trong đó chúng ta xem xét tuổi, giới tính và thu nhập của mỗi người. Mỗi người có thể được biểu diễn dưới dạng một vector gồm 3 giá trị, và do đó, toàn bộ tập dữ liệu gồm 100.000 người có thể được lưu trữ trong một tensor bậc 2 có shape là (100000, 3).

2.1.6 Dữ liệu chuỗi thời gian/ tuần tự

Khi làm việc với dữ liệu là thời gian hoặc dữ liệu mang tính tuần tự, việc lưu trữ dữ liệu đó trong một tensor bậc 3 với một trục thời gian rõ ràng là hợp lý. Mỗi mẫu (sample) có thể được mã hóa dưới dạng một tensor bậc 2, và do đó, một lô dữ liệu sẽ được mã hóa thành một tensor bậc 3.

Theo quy ước, trục thời gian luôn là trục thứ hai (trục có chỉ số 1). Ví dụ: Một tập dữ liệu về giá cổ phiếu: Cứ mỗi phút, chúng ta lưu trữ giá hiện tại của cổ phiếu, giá cao nhất trong phút vừa qua, và giá thấp nhất trong phút vừa qua. Như vậy, mỗi phút được mã hóa thành một vector 3 chiều, một ngày giao 32 dịch đầy đủ được mã hóa thành một ma trận có hình dạng (390, 3) (có 390 phút trong một ngày giao dịch), và dữ liệu của 250 ngày có thể được lưu trữ trong

một tensor bậc 3 có hình dạng (250, 390, 3). Ở đây, mỗi mẫu sẽ là dữ liệu của một ngày.

2.1.7 Dữ liệu hình ảnh

Hình ảnh thường có ba chiều: chiều cao, chiều rộng và độ sâu màu (color depth). Mặc dù các hình ảnh thang độ xám (như các chữ số trong MNIST) chỉ có một kênh màu và do đó có thể được lưu trữ trong tensor bậc 2, nhưng theo quy ước, tensor hình ảnh luôn là tensor bậc 3, với một kênh màu một chiều cho các hình ảnh thang độ xám. Một lô gồm 128 hình ảnh thang độ xám có kích thước 256 × 256 có thể được lưu trữ trong một tensor có hình dạng (128, 256, 256, 1), và một lô gồm 128 hình ảnh màu có thể được lưu trữ trong một tensor có hình dạng (128, 256, 256, 3).

Có hai quy ước về hình dạng của tensor hình ảnh: quy ước channels-last (được sử dụng chuẩn trong TensorFlow) và quy ước channels-first (ngày càng ít được ưa chuộng).

Quy ước channels-last đặt trục độ sâu màu ở cuối: (số mẫu, chiều cao, chiều rộng, độ sâu màu). Trong khi đó, quy ước channels-first đặt trục độ sâu màu ngay sau trục lô: (số mẫu, độ sâu màu, chiều cao, chiều rộng). Với quy ước channels-first, các ví dụ ở trên sẽ trở thành (128, 1, 256, 256) và (128, 3, 256, 256). API của Keras hỗ trợ cả hai định dạng này

2.1.8 Dữ liệu video

Dữ liệu video là một trong số ít loại dữ liệu thực tế mà bạn sẽ cần sử dụng tensor bậc 5. Một video có thể được hiểu là một chuỗi các khung hình, mỗi khung hình là một hình ảnh màu. Vì mỗi khung hình có thể được lưu trữ trong một tensor bậc 3 (chiều cao, chiều rộng, độ sâu màu), một chuỗi các khung hình có thể được lưu trữ trong một tensor bậc 4 (số khung hình, chiều cao, chiều rộng, độ sâu màu), và do đó, một lô gồm nhiều video khác nhau có thể được lưu trữ trong một tensor bậc 5 có hình dạng (số mẫu, số khung hình, chiều cao, chiều rộng, độ sâu màu)

Ví dụ, một đoạn video YouTube dài 60 giây, kích thước 144 × 256, được lấy mẫu với tốc độ 4 khung hình mỗi giây sẽ có 240 khung hình. Một lô gồm 4 đoạn video như vậy sẽ được lưu trữ trong một tensor có hình dạng (4, 240, 144, 256, 3). Đó là tổng cộng 106.168.320 giá trị! Nếu kiểu dữ liệu (dtype) của tensor là float32, mỗi giá trị sẽ được lưu trữ trong 32 bit, vì vậy tensor này sẽ chiếm 405 MB. Các video bạn gặp trong đời thực nhẹ hơn nhiều, vì chúng

không được lưu trữ ở định dạng float32, và chúng thường được nén rất nhiều (chẳng hạn như ở định dạng MPEG).

2.1.9 Các phép toán tensor

Mọi biến đổi trong mạng nơ-ron sâu đều có thể được rút gọn thành các phép toán tensor (như cộng, nhân, tích vô hướng).

Ví dụ tầng Dense trong Keras (keras.layers.Dense(512, activation = "relu")) là một hàm:

+ Nhận đầu vào là một ma trận (input).
+ Thực hiện: output = relu(dot(input, W) + b):
  - dot(input, W): Tích vô hướng giữa đầu vào và ma trận trọng số W.
  - + b: Cộng vector độ lệch b.
  - relu(): Hàm kích hoạt ReLU (max(x, 0) : x>0 trả về x, x<0 trả về 0)

a) Các phép toán theo từng phần tử

Phép toán ReLU và phép cộng là các phép toán theo từng phần tử: các phép toán được áp dụng một cách độc lập cho từng phần tử trong các tensor được xem xét. Điều này có nghĩa là các phép toán này rất phù hợp để triển khai song song hóa ở quy mô lớn. Nếu bạn muốn viết một triển khai Python đơn giản cho một phép toán theo từng phần tử, bạn sẽ sử dụng vòng lặp for, như trong triển khai đơn giản sau của phép toán ReLU theo từng phần tử:

```python
def naive_relu(x):
    assert len(x.shape) == 2  # x là một tensor bậc 2 NumPy
    x = x.copy()  # Tránh ghi đè lên tensor đầu vào
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] = max(x[i, j], 0) #phép relu
    return x
```

Tương tự đối với phép cộng:

```python
def naive_add(x, y):
    assert len(x.shape) == 2  # x và y là các tensor bậc 2 NumPy
    assert x.shape == y.shape
    x = x.copy()  # Tránh ghi đè lên tensor đầu vào
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] += y[i, j] # Phép cộng
    return x
```

Theo cùng nguyên tắc, bạn có thể thực hiện phép nhân theo từng phần tử, phép trừ, v.v.

Trong thực tế, khi làm việc với các mảng NumPy, các phép toán này có sẵn dưới dạng các hàm NumPy được tối ưu hóa tốt, và chúng giao phó công việc nặng nhọc cho một triển khai Basic Linear Algebra Subprograms (BLAS). BLAS là các routine thao tác tensor cấp thấp, song song hóa cao, hiệu quả, thường được triển khai bằng Fortran hoặc C.

Vì vậy, trong NumPy, bạn có thể thực hiện phép toán theo từng phần tử sau đây, và nó sẽ cực kỳ nhanh:

```
z = x + y  # Phép cộng theo từng phần tử
z = np.maximum(z, 0.)  # Phép ReLU theo từng phần tử

```

Tất nhiên, kết quả hàm có sẵn sẽ đúng và nhanh hơn hàm tự xây vì nó đã được tối ưu.

b) Phép phát sóng (broadcasting)

Với phép cộng 2 tensor đơn giản ở trên, chúng chỉ hỗ trợ phép cộng giữa các tensor bậc 2 có hình dạng giống hệt nhau. Nhưng nếu chúng ta muốn cộng một tensor bậc 2 với 1 vector, thì chúng ta cần dùng phép phát song.

Khi có thể và không có sự mơ hồ, tensor nhỏ hơn sẽ được phát sóng (broadcast) để khớp với hình dạng của tensor lớn hơn. Broadcasting bao gồm hai bước:

+ Bước 1: Các trục (còn gọi là trục phát sóng - broadcast axes) được thêm vào tensor nhỏ hơn để khớp với số trục của tensor lớn hơn.
+ Bước 2: Tensor nhỏ hơn được lặp lại dọc theo các trục mới này để khớp với toàn bộ hình dạng của tensor lớn hơn.

Ví dụ: Xét X có hình dạng (32, 10) và Y có hình dạng (10,):

+ Đầu tiên, chúng ta thêm một trục đầu tiên rỗng vào y, khiến hình dạng của nó trở thành (1, 10)
+ Sau đó, chúng ta lặp lại y 32 lần dọc theo trục mới này, để cuối cùng ta có một tensor Y với hình dạng (32, 10)

```python
import numpy as np
X = np.random.random((32, 10)) #random 1 ma trận có shape = 32,10
Y = np.random.random((10,)) # random vector có shape = 10
print("Shape của X và Y là:")
print(X.shape, Y.shape)
print()

# Thêm trục rỗng vào Y
Y = np.expand_dims(y, axis=0)
print("Shape của Y sau khi thêm 1 trục rỗng")
print(Y.shape)
print("Thu được Y có shape là (1,10)")
print()

#Lặp lại 32 lần y dọc theo trục 0
Y = np.concatenate([Y] * 32, axis=0)
print("Shape của Y sau 32 lần lặp")
print(Y.shape)
```

✓ 0.0s Python

```
Shape của X và Y là:
(32, 10) (10,)

Shape của Y sau khi thêm 1 trục rỗng
(1, 10)
Thu được Y có shape là (1,10)

Shape của Y sau 32 lần lặp
(32, 10)
```

+ Cuối cùng, tiến hành cộng X và Y, vì chúng có cùng hình dạng.

Thực tế, việc lặp lại này chỉ diễn ra ở mức độ thuật toán chứ không diễn ra ở cấp độ bộ nhớ, nghĩa là chúng ta chỉ việc thục hiện luôn tất cả các phép toán mà không cần thực hiện việc chuẩn hóa vector Y như ở trên

Ví dụ sau áp dụng phép toán maximum (relu) theo từng phần tử cho hai tensor có hình dạng khác nhau thông qua broadcasting:

```python
import numpy as np
x = np.random.random((64, 3, 32, 10))
y = np.random.random((32, 10))
z = np.maximum(x, y)
print(x.shape, y.shape, z.shape)
```

✓ 0.0s Python

```
(64, 3, 32, 10) (32, 10) (64, 3, 32, 10)
```

Dễ thấy shape của z sẽ giống shape của x, mặc dù shape y không thay đổi

c) Tích vô hướng với tensor

Tích tensor, hay còn gọi là tích vô hướng (dot product) (không nên nhầm lẫn với tích theo từng phần tử, toán tử *), là một trong những phép toán tensor phổ biến và hữu ích nhất.

Trong NumPy, tích tensor được thực hiện bằng hàm np.dot (vì ký hiệu toán học cho tích tensor thường là một dấu chấm), tuy nhiên shape của x, y phải tương thích (ví dụ giống như ma trận n x k thì phải nhân với ma trận dạng k x m chứ không thể nào là j x m)

Ví dụ nhân 2 vector: (tương tự với nhân 2 ma trận,.. giống như trong toán học)

```
x = np.random.random((32,))  # x và y là các vector NumPy
y = np.random.random((32,))

#kết quả thu được là 1 số, cách nhân giống với nhân 2 vector thông thường
z = np.dot(x, y)
print(z)
```

✓ 0.0s Python

```
7.893155968560009
```

Lưu ý: nếu ndim > 1 thì dot(x,y) khác dot(y,x)

d) Phép reshape (định hình lại)

Định dạng lại một tensor có nghĩa là sắp xếp lại các hàng và cột của nó để khớp với một hình dạng mục tiêu. Tất nhiên, tensor sau khi định dạng lại sẽ có cùng tổng số hệ số như tensor ban đầu. Định dạng lại được hiểu rõ nhất thông qua các ví dụ đơn giản:

```
import numpy as np

# Tạo ma trận x ban đầu
x = np.array([[0., 1.],
              [2., 3.],
              [4., 5.]])
print("Hình dạng ban đầu của x:", x.shape)  # (3, 2)

# Định dạng lại thành (6, 1)
x = x.reshape((6, 1))
print("x sau khi định dạng lại thành (6, 1):\n", x)

# Định dạng lại thành (2, 3)
x = x.reshape((2, 3))
print("x sau khi định dạng lại thành (2, 3):\n", x)

# Chuyển vị ma trận x
x = np.transpose(x)
print("Hình dạng của x sau khi chuyển vị:", x.shape)  #
```

✓ 0.0s Python

```
Hình dạng ban đầu của x: (3, 2)
x sau khi định dạng lại thành (6, 1):
 [[0.]
 [1.]
 [2.]
 [3.]
 [4.]
 [5.]]
x sau khi định dạng lại thành (2, 3):
 [[0. 1. 2.]
 [3. 4. 5.]]
Hình dạng của x sau khi chuyển vị: (3, 2)
```

2.2 Cách xây dựng một mô hình học sâu bằng các API của Keras

Để xây dựng một mô hình hoàn chỉnh, chúng ta cần trải qua nhiều bước: tìm dữ liệu, chuẩn hóa, xây các tầng, sau đó chuyển sang mô hình bằng lệnh compile, sau đó gọi phương thức fit để truyền dữ liệu vào huấn luyện để sử dụng nó.

2.2.1 Tìm và phân chia dữ liệu

a) Các tập dữ liệu

Bộ dữ liệu được sử dụng bao gồm 3 tập dữ liệu: tập huấn luyện (train), tập kiểm định (validation), tập kiểm tra (test):

+ Tập huấn luyện: là đầu vào của mô hình, được truyền vào mô hình để máy học và huấn luyện
+ Tập kiểm định: sử dụng song song với tập huấn luyện, do trong quá trình học, máy có thể "học vẹt" trên tập huấn luyện, do đó gây ra hiện tượng kết quả trên tập huấn luyện sẽ chính xác hơn nhiều so với tập kiểm tra (hay còn gọi là overfitting – hiện tượng quá khớp). Việc sử dụng tập kiểm định nhằm kiểm tra song song với tập huấn luyện trong quá trình huấn luyện để dễ dàng xác định xem khi nào mô hình bắt đầu xảy ra hiện tượng quá khớp.
+ Tập kiểm tra: dùng để kiểm tra độ chính xác của mô hình.

b) Tiền xử lý dữ liệu

Dữ liệu được đưa vào có thể là bất cứ thứ gì: ảnh, video, văn bản, âm thanh, chuỗi thời gian,… Để máy tính có thể hiểu các dữ liệu này, cần chuyển nó thành các tensor thông qua các hàm có sẵn trong thư viện utils của Keras

c) Hiện tượng overfitting

Là hiện tượng : mô hình học phải những mẫu lạ, không khái quát, làm cho tỉ lệ chính xác trên tập kiểm tra giảm dù tỉ lệ chính xác trên tập huấn luyện cao

Nguyên nhân:

+ Đầu vào không hợp lệ: ví dụ đầu vào phải là hình ảnh về số nhưng lại đưa hình ảnh về xe ô tô
+ Dữ liệu bị nhiễu
+ Đầu vào không khái quát: ví dụ: đầu vào là giọng địa phương, so với giọng phổ thông
+ Mẫu bị gán sai nhãn: ví dụ: đầu vào là ảnh cây, nhưng lại bị gán nhãn là cột đèn
+ Mối quan hệ giả: ví dụ: một từ ngữ mang ý nghĩa tích cực 54%, tiêu cực 46% sẽ nhận diện là tích cực

2.2.2 Khối xây dựng cơ bản của học sâu: Tầng (lớp Layers)

Cấu trúc dữ liệu cơ bản nhất trong mạng nơ-ron chính là tầng (layer). Nói một cách dễ hiểu, tầng là một "bộ xử lý dữ liệu" nhận vào một hoặc nhiều tensor và cho ra một hoặc nhiều tensor khác. Một số tầng không có trạng thái (stateless), nhưng thường thì tầng sẽ có trạng thái: đó là trọng số (weights) của tầng, gồm một hoặc nhiều tensor được học qua quá trình hạ gradient ngẫu nhiên. Trọng số này chính là "kiến thức" mà mạng nơ-ron tích lũy được.

Mỗi loại tầng sẽ phù hợp với các định dạng tensor và kiểu xử lý dữ liệu khác nhau, ví dụ:

+ Dữ liệu vector đơn giản, được lưu trong tensor bậc 2 có dạng (số mẫu, số đặc trưng), thường được xử lý bởi các tầng kết nối đầy đủ (densely connected layers), hay còn gọi là tầng Dense trong Keras.
+ Dữ liệu chuỗi (sequence data), được lưu trong tensor bậc 3 có dạng (số mẫu, số bước thời gian, số đặc trưng), thường được xử lý bởi các tầng tích chập 1D (Conv1D).
+ Dữ liệu hình ảnh, được lưu trong tensor bậc 4, thường được xử lý bởi các tầng tích chập 2D (Conv2D).

Việc xây dựng mô hình học sâu trong Keras chỉ đơn giản là ghép các tầng tương thích với nhau để tạo thành một dòng xử lý dữ liệu hữu ích.

Cách sử dụng một tầng có sẵn trong Keras:

$$layer1 = layers.tên\_tầng(số\ nơron, activation =' tên\ hàm\ kích\ hoạt')$$

Cách tạo mô hình từ các tầng:

```
model = keras.Sequential([
    layer1 = layers.tên_tầng (số_nơ_ron, activation = "hàm_kích_hoạt")
    ...
    layern = layers.tên_tầng (số_nơ_ron, activation = "hàm_kích_hoạt")
])
```

Ngoài ra, có thể phân loại các tầng thành tầng ẩn, tầng đầu ra,… như ví dụ sau:

```
# đầu vào
inputs = keras.Input(shape=(28, 28, 1))
# các tầng ẩn
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
# đầu ra
# tầng đầu ra
outputs = layers.Dense(10, activation="softmax")(x)
# tạo mô hình
model = keras.Model(inputs=inputs, outputs=outputs)
```

2.2.3 Các hàm kích hoạt cơ bản

a) Hàm kích hoạt ReLU

+ Công thức: $f(x) = \max(x, 0)$
+ Tác dụng: Chuyển các giá trị âm thành 0, giữ nguyên các giá trị dương. Nó giúp mô hình hội tụ nhanh hơn, giảm vấn đề gradient biến mất (vanishing gradient)
+ Ứng dụng: Sử dụng trong các tầng ẩn của mô hình, trong quá trình học của máy, nhằm học các mẫu phức tạp trong dữ liệu (như các góc cạnh, màu sắc của ảnh hoặc xu hướng dữ liệu của bảng)
  – Dùng trong xử lý ảnh: để học các góc, cạnh, màu sắc và vật thể
  – Dùng để dự đoán xu hướng tối ưu
  – Dùng để xử lý văn bản
+ Cách sử dụng: $activation = 'relu'$
+ Biến thể:
  – Leaky ReLU: thay vì x<0 thì x=0 như ReLU, thì với biến thể này, sẽ lấy giá trị αx với α nhỏ khoảng 0.01, cho phép đầu gradient âm nhỏ
  – ELU: cho ra đầu âm mượt mà hơn Leaky ReLU

b) Hàm kích hoạt Sigmoid

+ Công thức: $f(x) = \frac{1}{1+ e^{-x}}$
+ Tác dụng: ánh xạ giá trị vào khoảng (0,1), biến nó thành giá trị xác suất
+ Ứng dụng: dùng trong tầng đầu ra của bài toán phân loại nhị phân, kết quả đầu ra sẽ là xác suất của từng nhãn, lớn hơn 0.5 sẽ lấy nhãn 1, nhỏ hơn 0.5 sẽ lấy nhãn 0.
+ Cách sử dụng: $activation = 'sigmoid'$

c) Hàm kích hoạt Softmax

+ Công thức: $f(x_i) = \frac{e^{x_i}}{\sum_j^K (e^{x_j})}$
+ Tác dụng: Chuyển các giá trị thành xác suất, có tổng bằng 1, phù hợp cho phân loại đa lớp, lớp nào có xác suất cao nhất thì lớp đó sẽ được gán nhãn cho đầu vào.
+ Ứng dụng: dùng trong tầng đầu ra của bài toán phân loại đa lớp
+ Cách sử dụng: $activation = 'softmax'$

d) Hàm kích hoạt Tanh

+ Công thức: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
+ Tác dụng: ánh xạ giá trị vào khoảng (-1,1), giúp dữ liệu cân bằng quanh 0
+ Ứng dụng: sử dụng trong các tầng ẩn, đặc biệt là trong việc xử lý dữ liệu tuần tự như chuỗi thời gian, xử lý ngôn ngữ tự nhiên hay nhận diện giọng nói
+ Cách sử dụng: $activation = 'tanh'$

e) Hàm kích hoạt Linear

+ Công thức: $f(x) = x$
+ Tác dụng: không biến đổi dữ liệu
+ Ứng dụng: trong các bài toán hồi quy
+ Cách sử dụng: không gọi hàm activation hoặc $activation = 'linear'$

2.2.4 Một số loại tầng hay sử dụng trong Keras

a) Tầng kết nối đầy đủ - Dense

+ Đây là tầng kết nối đầy đủ, nghĩa là mỗi nơ ron trong tầng Dense đều được kết nối với tất cả các nơ ron ở tầng ngay kề trước nó.
+ Phù hợp cho dữ liệu ở dạng vector, không phù hợp với dữ liệu có cấu trúc không gian
+ Ứng dụng:

- Sử dụng ở tầng ẩn khi làm việc với dữ liệu dạng bảng, ví dụ như dữ liệu kết quả bệnh dựa trên các triệu chứng
- Sử dụng ở tầng đầu ra, ví dụ: với số nơ ron là 1, hàm kích hoạt sigmoid nếu dùng bài toán phân loại nhị phân, số nơ ron là số nhãn, hàm kích hoạt softmax nếu bài toán phân loại đa nhãn

b) Tầng tích chập 1D – Conv1D

+ Đây là tầng tích chập 1 chiều trong mạng nơ ron, được thiết kế để trích xuất các đặc trưng từ dữ liệu 1 chiều, hoạt động bằng cách dùng một cửa sổ tích chập trượt qua dữ liệu đầu vào theo một chiều, thường là chiều theo tuần tự thời gian/chuỗi
+ Phù hợp cho dữ liệu ở dạng tensor bậc 3, mang tính tuần tự
+ Ứng dụng;
  - Sử dụng ở tầng ẩn khi làm việc với các chuỗi thời gian, tín hiệu âm thanh hay ngôn ngữ tự nhiên

c) Tầng tích chập 2D – Conv2D

+ Tương tự như tầng tích chập 1 chiều, khác biệt ở chỗ nó làm việc với dữ liệu 2 chiều, hay còn gọi là dữ liệu được không gian hóa
+ Phù hợp với dữ liệu ở dạng tensor bậc 4, mang tính không gian
+ Ứng dụng:
  - Sử dụng ở tầng ẩn khi làm việc với hình ảnh

d) Tầng MaxPooling1D

+ Thường sử dụng cùng với Conv1D
+ Giúp giảm nhanh shape của tensor dữ liệu, từ đó giúp tập trung vào các đặc trưng quan trọng (ví dụ: đỉnh cao nhất của cửa sổ, hay những từ quan trọng nhất trong câu (do sử dụng max))
+ Đi cùng với Conv1D: Conv1D sẽ học những chi tiết từ dữ liệu, MaxPooling1D sẽ khái quát hóa dữ liệu nhanh chóng

e) Tầng MaxPooling2D

+ Tương tự như MaxPooling1D, nhưng đi kèm cùng với tầng Conv2D

f) Tầng làm phẳng – Flatten

+ Chuyển từ dữ liệu nhiều chiều về dữ liệu 1 chiều dạng vector
+ Giúp kết nối với tầng Dense

+ Lựa chọn thay thế: GlobalMaxPoolingND/ GlobalAveragePoolingND với N tương ứng với tầng các Conv và MaxPooling sử dụng trong mô hình

2.2.5 Chuyển từ các layers sang mô hình

Sau khi tạo các tầng, cần phải chọn thêm:

+ Hàm mất mát (loss)
+ Bộ tối ưu hóa (optimizer)
+ Chỉ số đo khi huấn luyện mô hình (metric)

Cách compile model, giả sử mô hình có sẵn lưu ở trong biến model

$$model.compile(optimizer = '\ bộ\ tối\ ưu', loss = '\ hàm\ mất\ mát', metrics = [metrics1, metrics2])$$

2.2.6 Một số bộ tối ưu hóa hay dùng

a) SGD: hạ gradient ngẫu nhiên

+ Chọn một phần nhỏ mẫu huấn luyện, sau tính gradient trên lô dữ liệu đã chọn và cập nhật tham số trên đó
+ Có ưu điểm như tốc độ tính toán nhanh, đơn giản, dễ triển khai, do tính ngẫu nhiên nên không dính phải những mẫu cục bộ
+ Không phù hợp với các bài toán tối ưu, ví dụ như tối ưu trải nghiệm khách hàng, vì nó sẽ chọn ngẫu nhiên một số khách hàng và tập trung vào tối ưu trải nghiệm của một số khách hàng đó và bỏ qua những khác hàng còn lại.

b) RMSprop:

+ Ưu việt hơn so với SGD ở chỗ: SGL có learning rate cố định, còn trong RMSProp thì learning rate được điều chỉnh dựa trên độ lớn của gradient
+ Ưu điểm: đơn giản, hiệu quả với mạng nơ ron sâu

c) Adam

+ Kết hợp ưu điểm của cả RMSProp và SGD
+ Thường sẽ được sử dụng nhiều, gần như mặc định trong các framework học sâu
+ Ưu điểm: ít cần điều chỉnh tham số
+ Nhược điểm: một số trường hợp có thể dẫn đến việc hội tụ tối ưu không tốt

2.2.7 Một số hàm mất mát

+ CategoricalCrossentropy/SparseCategoricalCrossentropy: sử dụng cho các bài toán phân loại đa lớp (categorical). SparseCategorical sẽ biểu diễn trực tiếp ở dạng số nguyên thay vì mã hóa one-hot, do đó nó tiết kiệm dữ liệu hơn, nhưng nhìn chung, kết quả của 2 hàm này là như nhau
+ BinaryCrossentropy : sử dụng cho bài toán phân loại nhị phân (binary), chỉ có 2 nhãn
+ MSE (Mean Square Error): đo lường giá trị trung bình bình phương sai lệch giữa giá trị dự đoán và giá trị thực tế, từ đó loại ra các giá trị ngoại lai (có khác biệt lớn so với trung bình). Hàm này sử dụng trong các bài toán dự báo xu hướng, giá nhà, thời tiết với dữ liệu liên tục
+ MAE (Mean Absolute Error): tương tự MSE nhưng giảm ảnh hưởng của các giá trị ngoại lai (outliers)

2.2.8 Một số metrics phổ biến

+ Accuracy: độ chính xác
+ Precision: tỉ lệ đúng cho lớp dương trên tổng số dự đoán là dương: $\frac{True\ Positvie}{True\ Positve+False\ Positve}$, tuy nhiên nó bỏ sót giá trị False Negative (nhãn là dương nhưng dự đoán âm)
+ Recall: tỷ lệ mẫu đúng (True) trên tổng số nhãn dương thực sự $\frac{True\ Positive}{True\ Positive+False\ Negative}$
+ F1-Scoree: tổng hợp trung bình của cả precision và Recall

$$F1-Score = 2\frac{Precision\ .\ Recall}{Precision + Recall}$$

2.2.9 Lưu ý trong chọn các thông số

Cần lưu ý việc chọn các thông số là rất quan trọng. Mạng nơ-ron sẽ tìm mọi cách để giảm giá trị mất mát, nên nếu hàm mất mát không thực sự phản ánh đúng mục tiêu thành công của bài toán, mạng có thể làm những điều mà bạn không mong muốn, như trong ví dụ về tối ưu trải nghiệm khách hàng ở 2.2.6

Với những hàm mất mát cơ bản, việc lựa chọn sao cho phù hợp đã được trình bày ở trên, dựa theo các tiêu chí và mô hình sử dụng, có thể tham khảo để chọn lựa.

2.2.10 Phương thức fit()

Công thức: $model.fit(tham\ số\ 1 = giá\ trị\ 1, \ldots., tham\ số\ n = giá\ trị\ n)$

Các tham số truyền vào phương thức fit

+ x: đầu vào, là tensor, ví dụ: x = train_data
+ y: nhãn ứng với đầu vào, là mảng, ví dụ: y = train_label
+ Có thể gộp thành train_dataset, lúc này không cần khai báo y, chỉ cần khai báo x = train_dataset
+ batch_size: kích thước lô, 1 số nguyên, ví dụ batch_size = 32
+ epochs: số vòng lặp huấn luyện, ví dụ epochs = 10
+ callbacks: chủ yếu sử dụng 2 đối tượng ModelCheckpoint (lưu mô hình tốt nhất) và EarlyStopping (dừng sớm). Thường sẽ định nghĩa callback riêng và sau đó gọi callbacks = callback
+ x_val, y_val: giống như tập train, nhưng ở đây áp dụng cho tập validation, có thể gộp thành validation_data
+ class_weight: gán trọng số cho các lớp để xử lý dữ liệu không cân bằng

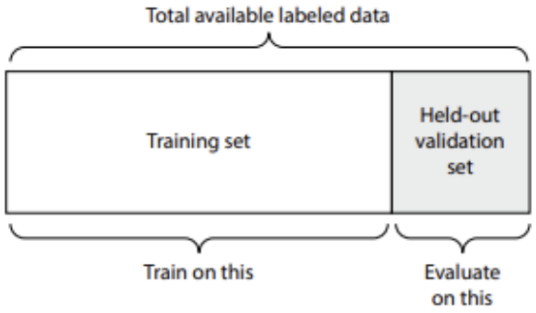
2.3 Một số cách để xử lý overfitting

2.3.1 Chia dữ liệu huấn luyện thành tập train và validation

Nguyên lý chung: sử dụng một tập kiểm định song song với tập kiểm tra. Tập kiểm định này sẽ không ảnh hưởng đến quá trình huấn luyện mô hình, tuy nhiên chúng ta có thể theo dõi kết quả thông qua tập kiểm định này để xem độ chênh lệch về độ chính xác giữa chúng, từ đó điều chỉnh các siêu tham số như batch_size, số epochs, learning rate,… hay điều chỉnh lại cấu trúc mô hình cho phù hợp

a) Chia tập huấn luyện đơn giản

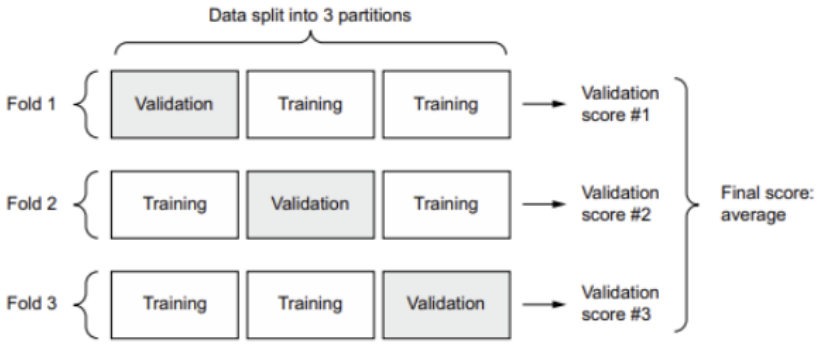
+ Chỉ đơn giản là chia tập huấn luyện thành 2 tập training set và validation test theo tỉ lệ bạn muốn, ví dụ 8:2



+ Phù hợp với các bộ dữ liệu lớn có nhiều dữ liệu

b) Chia tập huấn luyện theo kiểu K-fold

+ Chia tập huấn luyện thành k phần như nhau để thực hiện huấn luyện k lần, lần 1 lấy phần 1 để kiểm định, còn lại để huấn luyện, lần 2 lấy phần 2 kiểm định, còn lại huấn luyện,… lần n lấy phần n kiểm định, còn lại huấn luyện), thực hiện đủ k lần



+ Phù hợp với lượng dữ liệu huấn luyện ít vì bảo đảm tất cả dữ liệu đều tham gia vào huấn luyện và kiểm tra.

2.3.2 Chỉnh sửa các siêu tham số

Có thể chỉnh sửa các siêu tham số để mô hình đem lại kết quả tốt hơn

a) Learning rate

Là siêu tham số ảnh hưởng đến việc cập nhật trọng số của các bộ tối ưu:

+ Learning rate cao: Cập nhật trọng số nhanh, học nhanh hơn nhưng dễ vượt qua điểm tối ưu, gây không hội tụ
+ Learning rate thấp: Cập nhật mượt mà, dễ hội tụ đến tối ưu cục bộ nhưng sẽ tốn nhiều thời gian, chi phí hơn
+ Một số giá trị hay dùng: 1e-5, 1e-4, 1e-3, 0.01, 0.1

b) Batch size

Là số lượng mẫu dữ liệu được xử lý trong một lần lan truyền để cập nhật trọng số.

+ Batch size lớn: Tăng tốc huấn luyện vì ít cập nhật trọng số hơn mỗi epoch, tuy nhiên nó yêu cầu nhiều bộ nhớ, dễ gây ra tối ưu cục bộ
+ Batch size nhỏ: yêu cầu ít bộ nhớ hơn, tổng quát hóa hơn nhưng huấn luyện chậm hơn do phải cập nhật nhiều trọng số mỗi epoch và làm cho gradient không ổn định

2.3.3 Data augmentation

Do bộ dữ liệu sử dụng để huấn luyện mô hình có kích thước khá nhỏ, do đó có thể làm cho độ chính xác không cao do mô hình không thể tổng quát hóa mà chỉ như học thuộc lòng. Có càng nhiều dữ liệu thì mô hình càng có thể học chi tiết hơn, tiếp xúc với mọi khía cạnh của dữ liệu, do đó ít gây ra overfitting. Sử dụng data augmentation là một giải pháp để làm điều này

Data augmentation biến đổi dữ liệu cũ theo cách khác nhau (như xoay 1 góc, zoom to, thu nhỏ, dịch ảnh theo chiều ngang/dọc, lật ảnh, lấy ý) mà vẫn giữ nguyên nội dung chính để mô hình không thấy một hình ảnh giống nhau 2 lần, giúp việc tổng quát hóa tốt hơn.

Ví dụ, với dữ liệu ảnh, ta có những tầng phục vụ tăng cường dữ liệu sau:

+ RandomFlip: giúp lật ảnh theo chiều ngang hoặc chiều dọc, thường dùng với 2 tham số horizontal và vertical (ngang và dọc). Ví dụ: layers.RandomFlip("horizontal")
+ RandomRotation(x): quay ảnh ngẫu nhiên trong phạm vi -x * 360 độ đến x * 360 độ
+ RandomZoom(x): phóng to hoặc thu nhỏ từ 0 đến x lần
+ RandomTranslation: dịch ảnh theo chiều ngang hoặc dọc
+ RandomContrast(x): thay đổi độ tương phản từ -x đến x lần
+ Random Brightness(x): thay đổi độ sáng từ -x đến x lần

2.3.4 Chuẩn hóa theo lô (BatchNormalization)

Chuẩn hóa (normalization) là tất cả các phương pháp nhằm làm cho các mẫu dữ liệu khác nhau mà một mô hình học máy nhìn thấy trở nên giống nhau hơn, điều này giúp mô hình học và khái quát hóa tốt trên dữ liệu mới. Hình thức chuẩn hóa dữ liệu phổ biến nhất là lấy giá trị gốc trừ cho giá trị trung bình, sau đó chia cho độ lệch chuẩn (giả định dữ liệu tuân theo phân phối chuẩn). Tuy nhiên, mặc dù dữ liệu đầu vào đã chuẩn hóa, nhưng liệu sau mỗi tầng biến đổi, liệu đầu ra còn đúng với dạng chuẩn hay không? Chúng ta chưa có cơ sở gì để khẳng định điều đó.

Chuẩn hóa theo batch (batch normalization) làm chính điều đó. Chúng ta sẽ sử dụng tầng BatchNormalization trong Keras. Nó có thể chuẩn hóa dữ liệu một cách thích nghi ngay cả khi giá trị trung bình và phương sai thay đổi theo thời gian trong quá trình huấn luyện. Trong quá trình huấn luyện, nó sử dụng giá trị trung bình và phương sai của batch dữ liệu hiện tại để chuẩn hóa các mẫu, và trong quá trình suy luận (khi có thể không có sẵn một batch dữ liệu đại

diện đủ lớn), nó sử dụng trung bình động lũy thừa của giá trị trung bình và phương sai theo batch của dữ liệu đã thấy trong quá trình huấn luyện.

Tầng BatchNormalization có thể được sử dụng sau bất kỳ tầng nào—Dense, Conv2D,… Tuy nhiên, nên dùng BatchNormalization trước khi dùng hàm kích hoạt, như sau:

```
# chưa sử dụng hàm kích hoạt vội
x = layers.Conv2D(32, 3, use_bias=False)(x)
# dùng chuẩn hóa
x = layers.BatchNormalization()(x)
# sau đó mới dùng hàm kích hoạt
x = layers.Activation("relu")(x)
```

2.3.5 Dropout

+ Ngẫu nhiên bỏ qua (drop out) (đặt về 0) một số đặc trưng đầu ra của tầng trong quá trình huấn luyện.
+ Ví dụ: Một tầng trả về vector [0.2, 0.5, 1.3, 0.8, 1.1], sau khi áp dụng dropout (tỷ lệ 0.5), vector có thể là [0, 0.5, 1.3, 0, 1.1]
+ Tỷ lệ đặc trưng bị bỏ qua thường từ 0.2 đến 0.5
+ Sử dụng, giả sử tỉ lệ bỏ là 0.5: x = layers.Dropout(0.5)(x)

2.3.6 Regularization

+ Regularization là quá trình thêm một hàm phạt (vào hàm mất mát (loss function) hoặc áp dụng các kỹ thuật để hạn chế độ phức tạp của mô hình, giúp mô hình không học các chi tiết nhiễu (noise) trong dữ liệu huấn luyện mà tập trung vào các đặc trưng tổng quát, có khả năng khái quát hóa tốt trên dữ liệu mới.
+ Thường được dùng trong các mô hình nhỏ
+ Gồm 2 loại Regularization: L1 và L2
  - Regularization L1: Chi phí tỉ lệ với tổng giá trị tuyệt đối của trọng số : $\sum |w_i|$
  - Regularization L2: Chi phí tỉ lệ với tổng bình phương của các trọng số : $\sum w_i^2$
+ Code minh họa cách sử dụng:

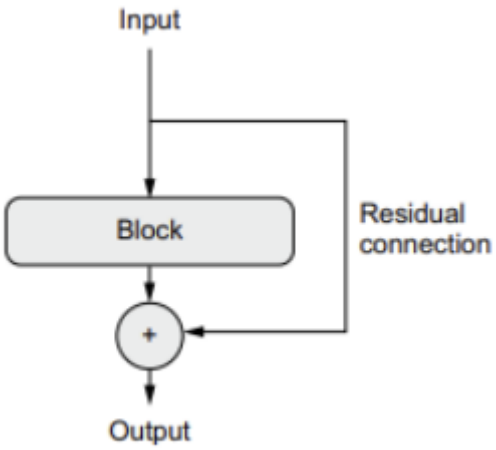
```
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation='relu',
                 kernel_regularizer=regularizers.l2(0.01)),
    layers.Dense(64, activation='relu',
                 kernel_regularizer=regularizers.l1(0.01)),
    layers.Dense(10, activation='softmax')
])
```

+ Áp dụng: L2 thường được sử dụng nhiều hơn bằng cách chọn trọng số nhỏ, làm mô hình mượt mà hơn, phân phối tầm quan trọng đều hơn giữa các đặc trưng, nhưng không loại bỏ hoàn toàn đặc trưng nào.
+ Ưu điểm:
  - Dễ triển khai, được tích hợp sẵn trong các framework, thư viện như TensorFlow, Keras
  - L2 thường hiệu quả hơn trong các mô hình DL vì nó không làm mất quá nhiều thông tin
  - L1 hữu ích khi cần giảm số lượng đặc trưng hoặc tham số
+ Nhược điểm:
  - Cần điều chỉnh siêu tham số λ, nếu chọn sai có thể làm mô hình thiếu khớp.
  - L1 có thể gây ra gradient không ổn định trong một số trường hợp

2.3.7 Residual connection – kết nối dư

Khi thông tin lan truyền qua các tầng, mỗi tầng sẽ đưa vào trong thông tin một lượng nhiễu nhất định. Nếu mô hình quá sâu, nhiều tầng, thì nhiễu có thể lấn át thông tin gradient, và lan truyền ngược ngừng hoạt động. Mô hình của bạn sẽ không huấn luyện được gì cả, gây ra vấn đề gradient biến mất (vanishing gradients)

Để khắc phục vấn đề này, chỉ cần buộc mỗi hàm trong chuỗi phải không phá hủy - tức là giữ lại một phiên bản không nhiễu của thông tin chứa trong đầu vào trước đó. Cách dễ nhất để thực hiện điều này là sử dụng một kết nối dư (residual connection): chỉ cần cộng đầu vào của một tầng hoặc một khối tầng trở lại vào đầu ra của nó để giữ thông tin không bị phá hủy, giúp gradient lan truyền không nhiễu qua mạng sâu.



Ví dụ về xây dựng hàm residual block trong xây dựng mạng tích chập CNN:

```
def residual_block(x, filters, pooling=False):
    # đầu vào có x, nên tạo 1 residual = x
    residual = x
    # x: 32,32,filter, tầng đầu xử lý đơn giản, tầng 2 để học chi tiết hơn
    x = layers.Conv2D(filters, 3, activation="relu", padding="same")(x)
    x = layers.Conv2D(filters, 3, activation="relu", padding="same")(x)
    # nếu có dùng maxpooling
    if pooling:
        # x: 15,15,filter
        x = layers.MaxPooling2D(2, padding="same")(x)
        # residual: 15,15,64 (có strides=2)
        residual = layers.Conv2D(filters, 1, strides=2)(residual)
    # TH còn lại, xét nếu residual và x không cùng số filter
    elif filters != residual.shape[-1]:
        # residual: 30,30,filter
      residual = layers.Conv2D(filters, 1)(residual)
    # cộng 2 tensor cùng shape là được
    x = layers.add([x, residual])
    # trả về
    return x
```

2.3.8 Callbacks

Là hàm được tự động gọi sau mỗi epochs nhằm kiểm tra, giám sát hiệu suất, tối ưu hóa huấn luyện, lưu lại kết quả quan trọng. Có hai loại callbacks phổ biến được sử dụng: ModelCheckpoint và EarlyStopping

Khi sử dụng, có thể định nghĩa sẵn các callbacks, và truyền vào phương thức fit mảng các hàm callbacks đã được định nghĩa sẵn

a) ModelCheckpoint – lưu lại mô hình tốt nhất

+ Filepath: đường dẫn mong muốn lưu file
+ Save_best_only = ‘True’: chỉ lưu mô hình tốt nhất

+ Monitor : tiêu chí chọn mô hình tốt nhất, thường là val_loss hoặc val_accuracy
+ Save_weights_only = ‘True’: chỉ lưu trọng số, nhẹ hơn lưu cả mô hình

b) EarlyStopping

Giúp tự động dừng huấn luyện khi giá trị val_loss và val_accuracy của tập kiểm định không cải thiện sau một số epochs được chỉ định

+ Monitor: tiêu chí đánh giá mô hình, thường là val_loss hoặc val_accuracy
+ Patience: độ kiên nhẫn, nghĩa là sau số lần bằng với độ kiên nhẫn mà kết quả trên tập validation không cải thiện thì sẽ phải dừng huấn luyện
+ Restore_best_weights= ‘True’: chọn lại trọng số tốt nhất để lưu

c) Minh họa:

```
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

checkpoint = ModelCheckpoint(
    filepath='best_model.keras',
    monitor='val_loss',
    save_best_only=True
)

history = model.fit(
    x_train, y_train,
    epochs=20,
    batch_size=32,
    validation_data=(x_val, y_val),
    callbacks=[early_stopping, checkpoint]
)
```

CHƯƠNG III: ỨNG DỤNG HỌC SÂU CHO NHẬN DIỆN TÌNH TRẠNG BỆNH THÔNG QUA ẢNH X-QUANG (X-RAY) TRONG LĨNH VỰC Y TẾ

Việc chọn chủ đề ứng dụng học sâu để nhận diện tình trạng bệnh thông qua ảnh X-quang xuất phát từ nhu cầu thực tiễn trong lĩnh vực y tế, nơi mà việc chẩn đoán nhanh và chính xác các bệnh về phổi như viêm phổi, COVID-19, hay lao phổi có vai trò quan trọng trong việc cải thiện tỷ lệ sống và giảm áp lực cho hệ thống y tế. Ảnh X-quang là một phương pháp chẩn đoán hình ảnh phổ biến, không xâm lấn, và chi phí thấp, nhưng việc phân tích thủ công bởi bác sĩ thường tốn thời gian và phụ thuộc vào kinh nghiệm

Học sâu, đặc biệt là mạng tích chập CNN, có khả năng tự động trích xuất đặc trưng từ ảnh X-quang, hỗ trợ bác sĩ trong việc phát hiện bệnh một cách nhanh chóng, chính xác, và giảm thiểu sai sót do yếu tố con người

Chủ đề này không chỉ có ý nghĩa trong việc nâng cao hiệu quả chẩn đoán y tế mà còn góp phần thúc đẩy ứng dụng công nghệ trí tuệ nhân tạo vào các lĩnh vực chăm sóc sức khỏe, đặc biệt tại các khu vực thiếu hụt bác sĩ chuyên môn

Tuy nhiên, việc ứng dụng học sâu trong lĩnh vực này chỉ có ý nghĩa hỗ trợ các bác sĩ trong việc chẩn đoán, không thể nào thay thế vai trò thực sự của bác sĩ trong lĩnh vực y tế

3.1 Các bộ dữ liệu được sử dụng

Ở đây, chúng ta sẽ ứng dụng mạng tích chập CNN trong phân loại hình ảnh với các lĩnh vực y tế, abc, abc ứng với các bộ dữ liệu sau

+ Chest X-Ray Images (Pneumonia): bộ dữ liệu về phân loại viêm phổi hoặc bình thường trên Kaggle. Bộ dữ liệu gồm 5863 ảnh, với 2 nhãn normal (bình thường) và pneumonia (bị viêm phổi) Đường dẫn: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia
  - Nhận xét về bộ dữ liệu: Độ chênh lệch giữa số ảnh về viêm phổi và bình thường chênh nhau rất lớn (do đặc thù, nên số ảnh viêm phổi sẽ nhiều hơn, cụ thể là gần 3 lần), vì vậy, cần sử dụng kỹ thuật class_weight để xử lý chênh lệch giữa các lớp.
  - Bộ dữ liệu đã được chia sẵn làm 3 tập: train (huấn luyện), val (kiểm định), test (kiểm tra). Số ảnh trong tập kiểm định (val) là khá ít (16 ảnh, 8 ảnh cho mỗi nhãn), do đó, chúng ta sẽ cắt bớt một vài ảnh từ tập test và tập train sang để huấn luyện

- Bộ dữ liệu ảnh viêm phổi được lưu ở dạng màu grayscale (thang màu xám), do đó không cần xây dựng mô hình quá phức tạp.

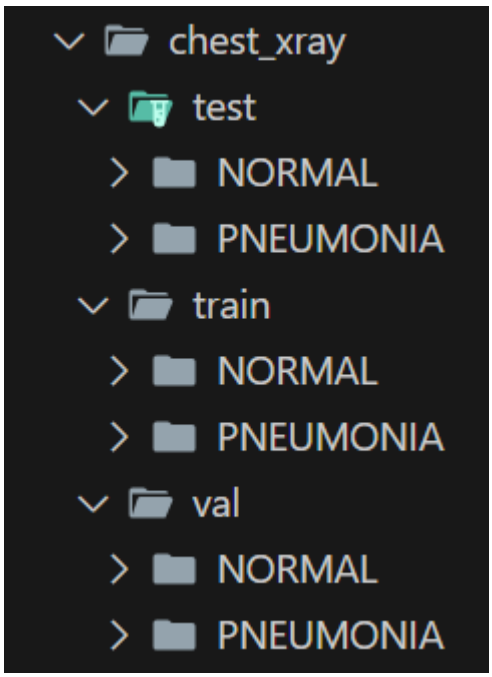
+ COVID19_Pneumonia_Normal_Chest_Xray_PA_Dataset: bộ dữ liệu về phân loại bệnh covid-viêm phổi – bình thường, gồm khoảng 2313 ảnh cho mỗi bệnh. Đường dẫn đến bộ dữ liệu trên Kaggle như sau: https://www.kaggle.com/datasets/amanullahasraf/covid19-pneumonia-normal-chest-xray-pa-dataset
  - Độ chênh lệch dữ liệu giữa số ảnh gần như bằng 0 giữa các bệnh
  - Bộ dữ liệu gồn 3 tập: covid, pneumonia, normal, chưa được chia làm các tập dữ liệu riêng biệt val, train, test, và các ảnh lưu ở cả 3 chế độ grayscale, RGB, RGBA
  - Xử lý: do mô hình dùng chung, nên chỉ lấy 2 tập là covid và pneumonia để làm dữ liệu, chuyển hết các ảnh về dạng màu grayscale và chia thành 3 phần val, test, train theo tỉ lệ 1000:1000: còn lại (tầm 300 ảnh cho tập val)
+ Tuberculosis Chest X-ray Database: bộ dữ liệu về phân loại bệnh lao phổi và phổi bình thường thông qua chụp X-quang, gồm 3500 ảnh X-quang phổi bình thường và 700 ảnh về bệnh lao phổi. Đường dẫn đến bộ dữ liệu: https://www.kaggle.com/datasets/tawsifurrahman/tuberculosis-tb-chest-xray-dataset
  - Độ chênh lệch dữ liệu là khá lớn (gấp 5 lần)
  - Bộ dữ liệu gồm 2 tập: Normal và Tuberculosis, chưa được chia thành các tập train, val, test
  - Xử lý: chuyển tất cả các ảnh về grayscale, lấy 700 ảnh mỗi loại chia vào 3 tập: train, val, test theo tỉ lệ 300, 100, 300
  - Thử nghiệm kết quả với bộ dữ liệu rất nhỏ
+ So sánh 3 bộ dữ liệu với nhau, theo thứ tự 1,2,3, sau khi đã phân chia thành các tập train, test, val

STT	Tập train	Tập val	Tập test	Nhận xét
1	5216	55	585	- Tập train: 1314 Normal – 3875 Pneumonia: độ chênh lệch giữa dữ liệu ứng với 2 nhãn là khá lớn, gấp 3 lần - Tập val có khá ít dữ liệu, do đó, đôi khi các giá trị loss và accuracy trên tập val không thực sự đại diện cho sự thật

				- Đây là bộ dữ liệu không quá bé, cỡ vừa nhưng có độ chênh lệch giữa các lớp khá lớn
2	2000	600	2000	- Tập train: tỉ lệ dữ liệu giữa các lớp là 1:1, không có chênh lệch - Tỉ lệ dữ liệu giữa 3 tập dữ liệu của cả 2 bộ dữ liệu đều khá hợp lý - Bộ dữ liệu 2: là bộ dữ liệu nhỏ - Bộ dữ liệu 3: là bộ dữ liệu rất nhỏ
3	600	200	600	

3.2 Huấn luyện mô hình với bộ dữ liệu Chest X-Ray Images (Pneumonia)

Bước 1: Tải dữ liệu từ đường link ở trên. Dữ liệu sau khi được tải về sẽ có cấu trúc thư mục như sau:



Bước 2: Import các thư viện cần thiết, tạo đường dẫn

```
import tensorflow as tf
from tensorflow.keras.utils import image_dataset_from_directory
from pathlib import Path
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
```

Bước 3: Định nghĩa đường dẫn đến thư mục chứa dữ liệu và chuyển đổi các ảnh thành các tensor thông qua hàm image_dataset_from_directory trong thư viện utils của Keras:

+ Tạo 3 tập dữ liệu huấn luyện (train_dataset), kiểm định (validation_dataset), kiểm tra (test_dataset) từ 3 thư mục con train, val, test
+ Chọn kích thước ảnh là 150 pixel * 150 pixel
+ Kích thước lô: 64 ảnh/lô
+ Chế độ màu: do ảnh X-quang nên dùng thang máu độ xám: grayscale

```python
import tensorflow as tf
from tensorflow.keras.utils import image_dataset_from_directory
from pathlib import Path
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt

# Định nghĩa đường dẫn đến thư mục dữ liệu
base_dir = Path(r"C:\Users\ASUS\Desktop\TTCS\chest_xray")

# Tạo tập dữ liệu huấn luyện
train_dataset = image_dataset_from_directory(
    base_dir / "train",
    image_size=(150, 150),
    batch_size=64,
    color_mode="grayscale"
)

# Tạo tập dữ liệu kiểm tra (validation)
validation_dataset = image_dataset_from_directory(
    base_dir / "val",
    image_size=(150, 150),
    batch_size=64,
    color_mode="grayscale"
)

# Tạo tập dữ liệu thử nghiệm (test)
test_dataset = image_dataset_from_directory(
    base_dir / "test",
    image_size=(150, 150),
    batch_size=64,
    color_mode="grayscale"
)

```

Bước 4: Sử dụng tăng cường dữ liệu (data augmentation), do số lượng ảnh không quá nhiều, việc sử dụng data augmentation sẽ giúp đa dạng hóa dữ liệu huấn luyện hơn mà không bị trùng lặp với dữ liệu cũ

+ Với mọi loại hình ảnh đều có thể sử dụng các tầng như RandomFlip, RandomRotation, RandomZoom
+ Với dữ liệu X-quang, việc sử dụng thêm RandomContrast để điều chỉnh độ tương phản và RandomBrightness để điều chỉnh độ sáng khá hữu ích, vì nó sẽ làm rõ hơn sự khác biệt, tương phản giữa vùng trắng, mờ đục (dấu hiệu viêm phổi) so với vùng trong suốt (bình thường)

```
# Tăng cường dữ liệu
data_augmentation = keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.3),
    layers.RandomContrast(0.2),
    layers.RandomBrightness(0.2),
])
```

Bước 5: Xây dựng model

+ Do ở bước 3 đã chọn kích thước ảnh là 150 pixel * 150 pixel và kênh màu là grayscale, nên input đầu vào sẽ là (150,150,1)
+ Áp dụng tầng tăng cường dữ liệu đã xây ở trên
+ Độ xám biểu diễn số từ 0-255, do đó dùng tầng Rescaling để chuẩn hóa về dạng giá trị trong khoảng (0,1)
+ Sử dụng 3 tầng kết hợp giữa Conv2D và MaxPooling2D để máy có thể học được những chi tiết của ảnh
+ Sử dụng regularization: tránh mô hình có thể học quá mức trên tập train, làm giảm overfitting
+ Sử dụng tầng Flatten để làm phẳng dữ liệu về dạng 1 chiều để đưa vào tầng Dense phía sau tạo đầu ra
+ Sau đó sử dụng Dropout(0.5) để giúp giảm tình trạng overfitting (quá khớp) của mô hình
+ Vì đầu ra là nhị phân (2 nhãn) do đó sử dụng tầng Dense với số nơ ron là 1 và hàm kích hoạt là sigmoid để đưa ra xác suất của từng nhãn đối với hình ảnh được đưa vào
+ Lưu ý khi xây dựng model: phải thử nghiệm nhiều cách xây dựng model khác nhau để tìm ra mô hình cho ra kết quả có độ chính xác cao nhất, chứ không phải lúc nào mô hình phức tạp cũng sẽ cho ra kết quả chính xác hơn các mô hình đơn giản. Dựa trên thực tế xây dựng và huấn luyện mô hình, có thể kết luận như sau:
    - Kỹ thuật tăng cường dữ liệu, regularization và Dropout là có ích, giúp tăng độ chính xác, do đó sử dụng nó trong mô hình này là hợp lý
    - Các kỹ thuật như kết nối dư, chuẩn hóa theo lô, hay sử dụng tầng SeparableConv2D tuy nâng cao và nghe có vẻ phức tạp và theo lý thuyết sẽ hiệu quả hơn, nhưng khi áp dụng lại làm cho mô hình bị overfitting nghiêm trọng, do đó không sử dụng chúng trong mô hình này (độ chính xác trên tập train lên tới 98% nhưng trên tập validation và tập kiểm tra chỉ dừng ở mức tối đa 75%), so với

không sử dụng đem lại độ chính xác cao nhất lên tới hơn 88% (xem kết quả ở dưới)

```
# Xây dựng mô hình
inputs = keras.Input(shape=(150, 150, 1))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(inputs)

# Tầng 1
x = layers.Conv2D(filters=8, kernel_size=3, activation='relu',
                  kernel_regularizer=keras.regularizers.l2(0.01))(x)
x = layers.MaxPooling2D()(x)
# Tầng 2
x = layers.Conv2D(filters=16, kernel_size=3, activation='relu',
                  kernel_regularizer=keras.regularizers.l2(0.01))(x)
x = layers.MaxPooling2D()(x)

# Tầng 3
x = layers.Conv2D(filters=32, kernel_size=3, activation='relu',
                  kernel_regularizer=keras.regularizers.l2(0.01))(x)
x = layers.MaxPooling2D()(x)

# Làm phẳng và thêm Dense
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(64, activation="relu")(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Bước 6: Biên dịch mô hình, chọn 3 tham số cho mô hình đó là hàm mất mát, bộ tối ưu hóa và các metrics

+ Hàm mất mát: do bài toán phân loại nhị phân, nên chọn hàm mất mát là binary_crossentropy
+ Bộ tối ưu Adam đem lại điểm mạnh của cả gradient descent và rmsprop, learning rate thấp để mô hình học chậm hơn nhưng nó ổn định hơn và tránh việc bị vượt qua điểm tối ưu so với sử dụng learning rate có giá trị cao
+ Metrics: chúng ta cần xem xét mô hình dựa trên độ chính xác của dự đoán, do đó chọn accuracy

```
# Biên dịch mô hình
model.compile(loss="binary_crossentropy",
              optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
              metrics=["accuracy"])
```

Bước 7: Sử dụng phương thức fit để huấn luyện mô hình

+ Định nghĩa callbacks: để lưu lại mô hình tốt nhất trong quá trình huấn luyện vào file best_model_pneumonia.keras dựa trên điều kiện là giá trị hàm mất mát của tập kiểm định (không phải của tập train vì hiện tượng

overfitting làm cho độ chính xác cao hơn và giá trị hàm mất mát thấp hơn trên tập train so với tập validation và test.

+ Định nghĩa class_weight: do mô hình chứa nhiều ảnh viêm phổi hơn ảnh bình thường 3 lần, do đó sẽ đặt class_weight để cân bằng lại dữ liệu (với lớp Normal – 0 và Pneunomia – 1 sẽ có tỉ lệ 3 : 1) (việc gán lớp 0-1 là dựa vào thứ thự bảng chữ cái của từ bắt đầu, nên N ứng với 0, còn P ứng với 1)
+ Truyền vào mô hình các tập huấn luyện, tập kiểm dịnh và huấn huyện qua 20 epochs, sau đó truyền vào callbacks và class_weight

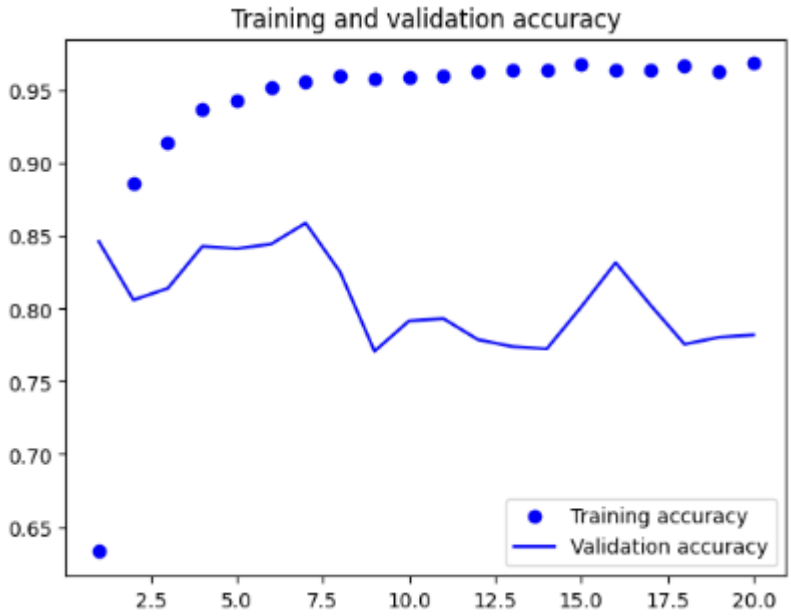
```
# Callbacks
callbacks = [
    keras.callbacks.ModelCheckpoint(filepath="best_model_pneumonia.keras",
                                    save_best_only=True,
                                    monitor="val_loss")
]

# Thêm class_weight để xử lý chênh lệch lớp
class_weight = {0: 3.0, 1: 1.0}

# Huấn luyện mô hình
history = model.fit(train_dataset,
                    epochs=20,
                    validation_data=validation_dataset,
                    callbacks=callbacks
                    , class_weight=class_weight)
```

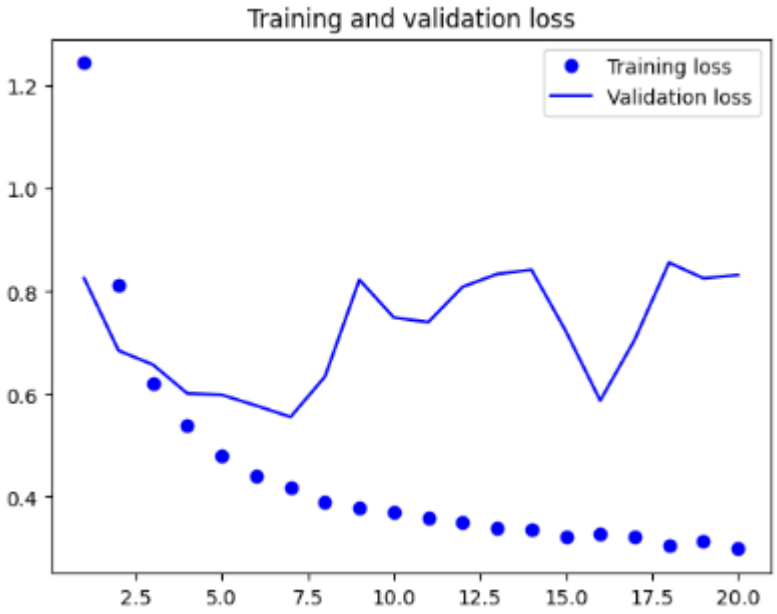
+ Sau khi mô hình đã fit (huấn luyện) xong với dữ liệu được từ tập huấn luyện và kiểm định, ta đã có mô hình tốt nhất và có thể theo dõi kết quả theo từng epoch như hình dưới đây:

```
Epoch 1/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 17s 145ms/step - accuracy: 0.5026 - loss: 1.3228 - val_accuracy: 0.8462 - val_loss: 0.8256
Epoch 2/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 11s 135ms/step - accuracy: 0.8788 - loss: 0.8926 - val_accuracy: 0.8061 - val_loss: 0.6848
Epoch 3/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 11s 135ms/step - accuracy: 0.9110 - loss: 0.6439 - val_accuracy: 0.8141 - val_loss: 0.6574
Epoch 4/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 12s 137ms/step - accuracy: 0.9393 - loss: 0.5475 - val_accuracy: 0.8429 - val_loss: 0.6013
Epoch 5/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 11s 137ms/step - accuracy: 0.9442 - loss: 0.4743 - val_accuracy: 0.8413 - val_loss: 0.5987
Epoch 6/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 10s 116ms/step - accuracy: 0.9535 - loss: 0.4355 - val_accuracy: 0.8446 - val_loss: 0.5777
Epoch 7/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 10s 114ms/step - accuracy: 0.9601 - loss: 0.4099 - val_accuracy: 0.8590 - val_loss: 0.5556
Epoch 8/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 9s 112ms/step - accuracy: 0.9606 - loss: 0.3859 - val_accuracy: 0.8253 - val_loss: 0.6343
Epoch 9/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 10s 117ms/step - accuracy: 0.9565 - loss: 0.3734 - val_accuracy: 0.7708 - val_loss: 0.8226
Epoch 10/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 10s 114ms/step - accuracy: 0.9575 - loss: 0.3764 - val_accuracy: 0.7917 - val_loss: 0.7493
Epoch 11/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 10s 120ms/step - accuracy: 0.9639 - loss: 0.3476 - val_accuracy: 0.7933 - val_loss: 0.7401
Epoch 12/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 10s 121ms/step - accuracy: 0.9673 - loss: 0.3349 - val_accuracy: 0.7788 - val_loss: 0.8088
...
82/82 ━━━━━━━━━━━━━━━━━━━━ 11s 127ms/step - accuracy: 0.9711 - loss: 0.2886 - val_accuracy: 0.7821 - val_loss: 0.8319
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 0.7888 - loss: 0.8293
```

Bước 8: Sơ đồ hóa để nhận xét kết quả, sự chênh lệch về độ chính xác và giá trị hàm mất mát giữa tập train và tập validation

+ Từ bước này sẽ chỉ là kiểm tra và chạy thử kết quả
+ Tuy mô hình vẫn overfitting, nhưng kết quả trên tập kiểm định cũng không quá thấp (tầm 86-87% tối đa).
+ Kết quả này có thể coi là khá ổn





Bước 9: Xem độ chính xác trên tập kiểm tra

+ Kết quả: 88,21%, khá ổn so với một mô hình đơn giản

+ Tuy nhiên, hiện tượng overfitting xảy ra khá sớm, có thể do tập validation quá ít dữ liệu cho nên không phản ánh chính xác được hết kết quả thực sự

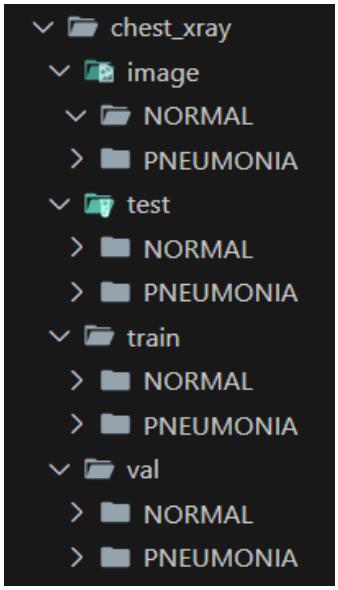
```python
import tensorflow as tf
from tensorflow.keras.utils import image_dataset_from_directory
from pathlib import Path
import matplotlib.pyplot as plt
from tensorflow.keras import config

# Định nghĩa đường dẫn đến thư mục dữ liệu
base_dir = Path(r"C:\Users\ASUS\Desktop\TTCS\chest_xray")

# Tạo tập dữ liệu thử nghiệm (test)
test_dataset = image_dataset_from_directory(
    base_dir / "test",
    image_size=(150, 150),
    batch_size=64,
    color_mode="grayscale",
    shuffle=False
)

# Tải mô hình đã lưu
model = tf.keras.models.load_model("best_model_pneumonia.keras")

# Đánh giá accuracy trên tập test
test_loss, test_accuracy = model.evaluate(test_dataset)
print(f"Test accuracy: {test_accuracy:.4f}")
print(f"Test loss: {test_loss:.4f}")
```

✓ 0.8s

```
Found 585 files belonging to 2 classes.
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8661 - loss: 0.6115
Test accuracy: 0.8821
Test loss: 0.5527
```

Bước 10: Kiểm tra kết quả

+ Để thuận tiện, việc kiểm tra sẽ diễn ra trên hàng loạt các ảnh
+ Tạo thêm 1 folder image, trong folder đó có 2 folder con tương ứng với ảnh bình thường và ảnh bị viêm phổi, sau đó đưa các hình ảnh tìm được tương ứng vào 2 folder con. In ra kết quả của 10 ảnh bất kỳ cùng với nhãn và dự đoán
  – Thư mục mới



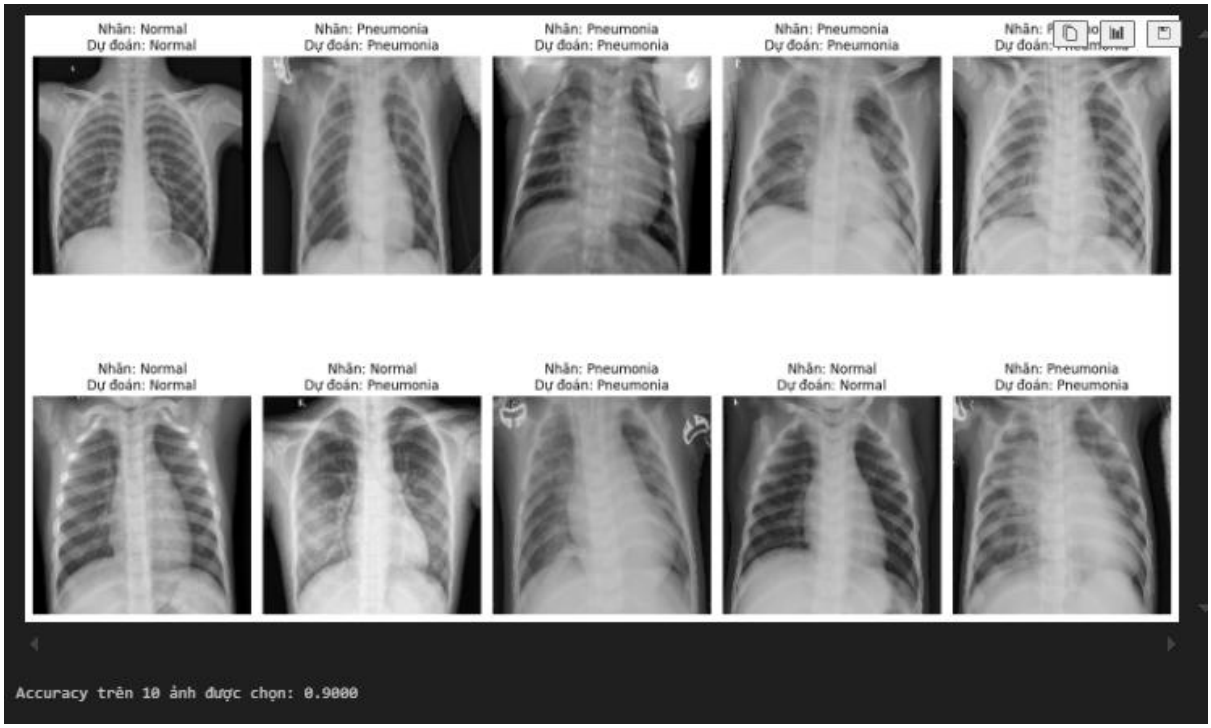
– Code triển khai ứng dụng:

```
import tensorflow as tf
from tensorflow.keras.utils import image_dataset_from_directory
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt

# Định nghĩa đường dẫn đến thư mục dữ liệu
base_dir = Path(r"C:\Users\ASUS\Desktop\TTCS\chest_xray")

# Tạo tập dữ liệu từ các ảnh thêm vào
test_dataset = image_dataset_from_directory(
    base_dir / "image",
    image_size=(150, 150),
    batch_size=64,
    color_mode="grayscale",
    shuffle=False  # Không xáo trộn để giữ thứ tự
)
```

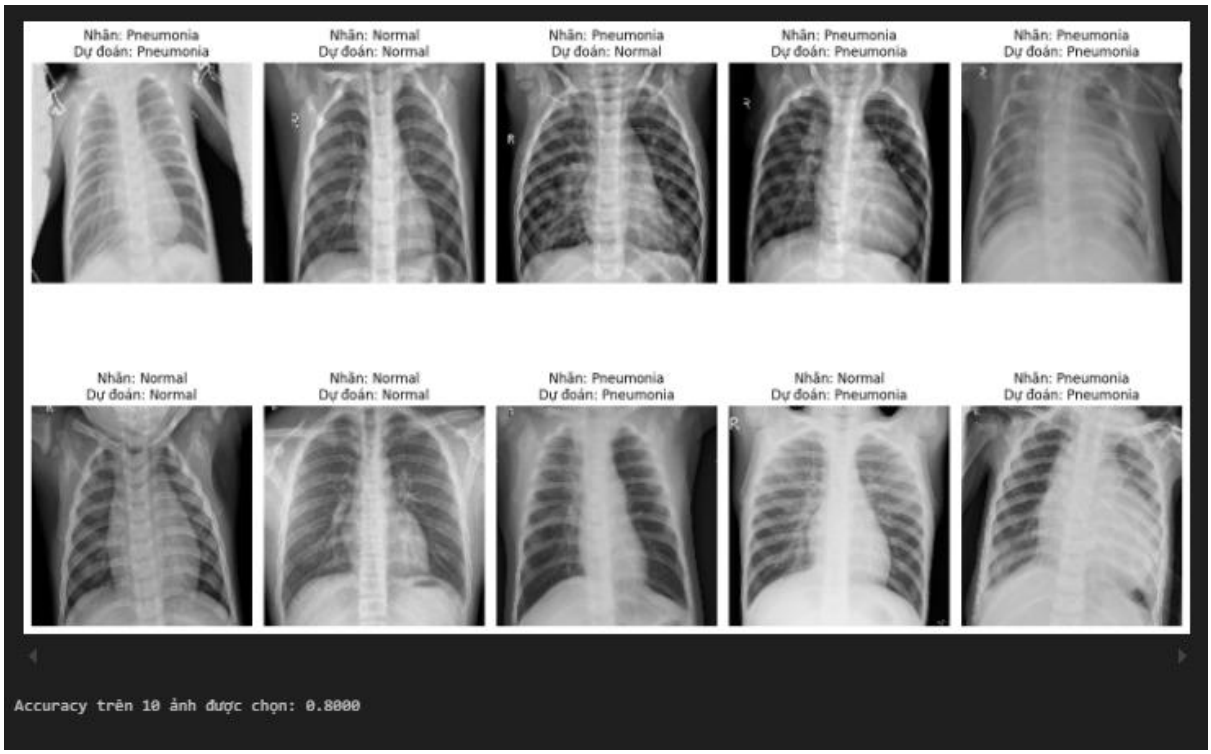
```python
# Tải mô hình đã lưu
model = tf.keras.models.load_model("best_model_pneumonia.keras")
# Lấy toàn bộ ảnh và nhãn từ tập image
test_images = []
test_labels = []
for images, labels in test_dataset.unbatch():
    test_images.append(images.numpy())
    test_labels.append(labels.numpy())
# Chuyển danh sách thành mảng numpy
test_images = np.array(test_images)
test_labels = np.array(test_labels)
# Chọn ngẫu nhiên 10 chỉ số ảnh
num_images_to_display = 10
random_indices = np.random.choice(len(test_images), num_images_to_display, replace=False)
# Lấy 10 ảnh và nhãn tương ứng
selected_images = test_images[random_indices]
selected_labels = test_labels[random_indices]
# Dự đoán trên 10 ảnh được chọn
predictions = model.predict(selected_images)
predicted_labels = (predictions > 0.5).astype(int).flatten()
# Hiển thị 10 ảnh cùng với nhãn thực tế và dự đoán
class_names = ["Normal", "Pneumonia"]
plt.figure(figsize=(15, 10))
for i in range(num_images_to_display):
    plt.subplot(2, 5, i + 1)  # Sắp xếp thành lưới 2x5
    plt.imshow(selected_images[i].squeeze(), cmap='gray')  # Hiển thị ảnh grayscale
    true_label = class_names[selected_labels[i]]
    pred_label = class_names[predicted_labels[i]]
    plt.title(f"Nhãn: {true_label}\nDự đoán: {pred_label}")
    plt.axis('off')  # Ẩn trục
plt.tight_layout()
plt.show()
# Tính và in accuracy trên 10 ảnh được chọn
accuracy = np.mean(selected_labels == predicted_labels)
print(f"Accuracy trên 10 ảnh được chọn: {accuracy:.4f}")
```

+ Chạy code và xem kết quả:

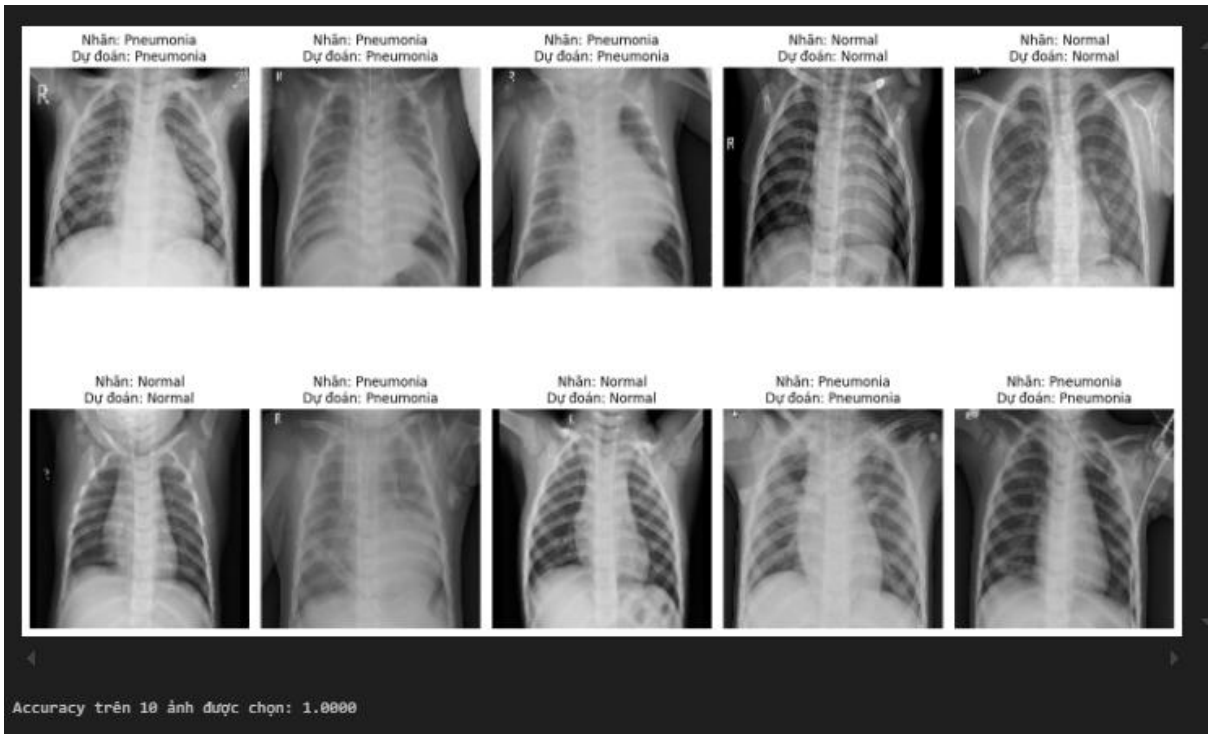
Dự đoán lần 1: Độ chính xác: 9/10



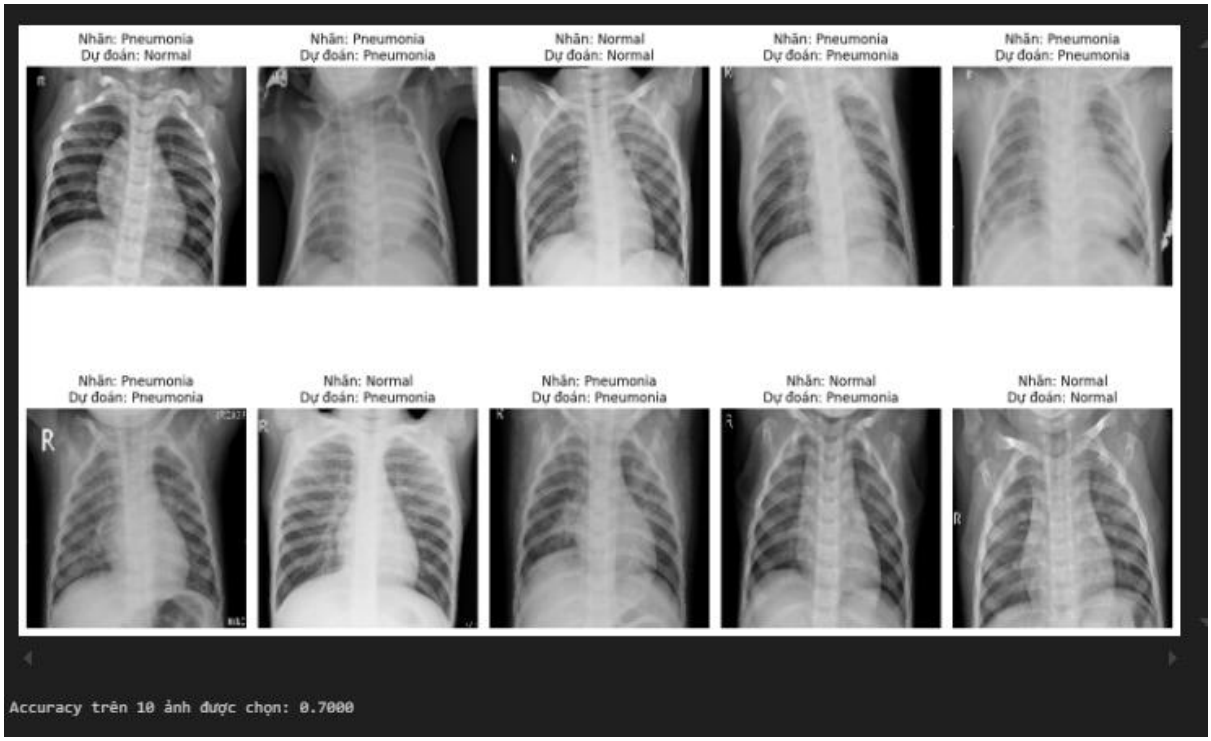
Dự đoán lần 2: Độ chính xác: 8/10



Dự đoán lần 3: Độ chính xác: 10/10

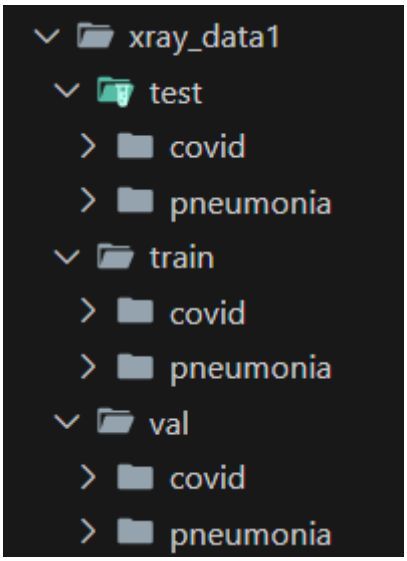


Dự đoán lần 4: Độ chính xác: 7/10



3.3 Huấn luyện mô hình với bộ dữ liệu COVID19 Pneumonia Normal Chest Xray PA_Dataset

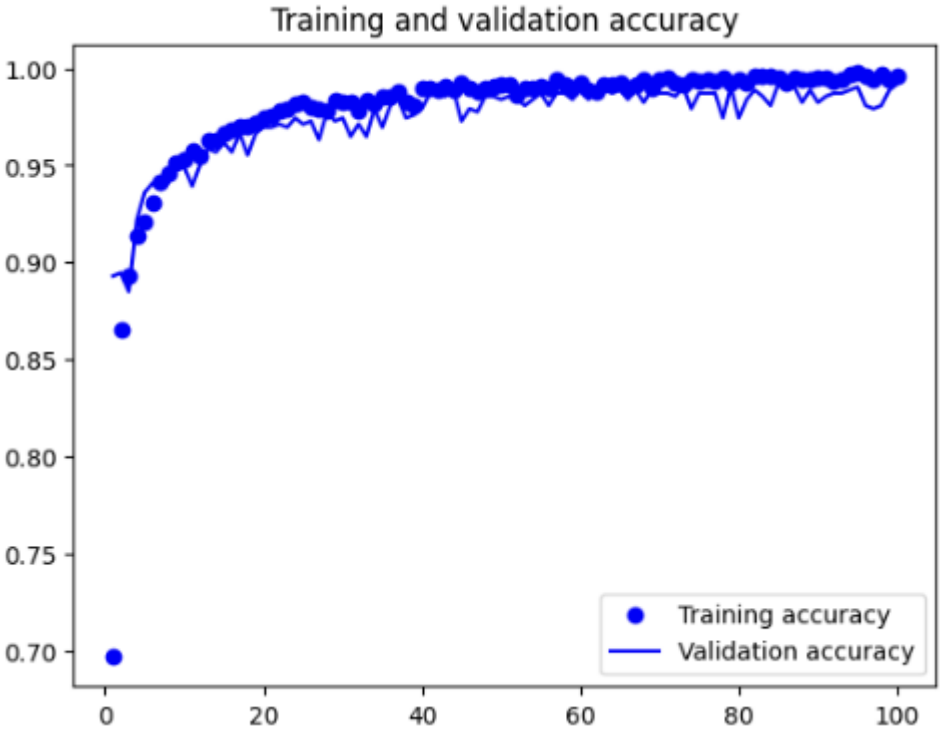
+ Do đã xử lý dữ liệu ở trên và sử dụng chung model, nên ở đây, ta đi xem xét kết quả, chỉ thay đổi 2 điểm: bỏ class_weight do dữ liệu đã cân bằng và model tốt nhất sẽ lưu ở file “best_model_covid.keras”
+ Sau khi huấn luyện mô hình, ta nhận thấy kết quả trên mô hình này khá tốt (do tập val đã đủ nhiều), ít bị overfitting, do đó ta tăng lên 100 epoch để xem chi tiết hơn
+ Độ chính xác của mô hình là 89.89%, cao hơn tầm 1% so với với bộ dữ liệu đầu tiên
+ Ảnh cấu trúc thư mục dữ liệu:

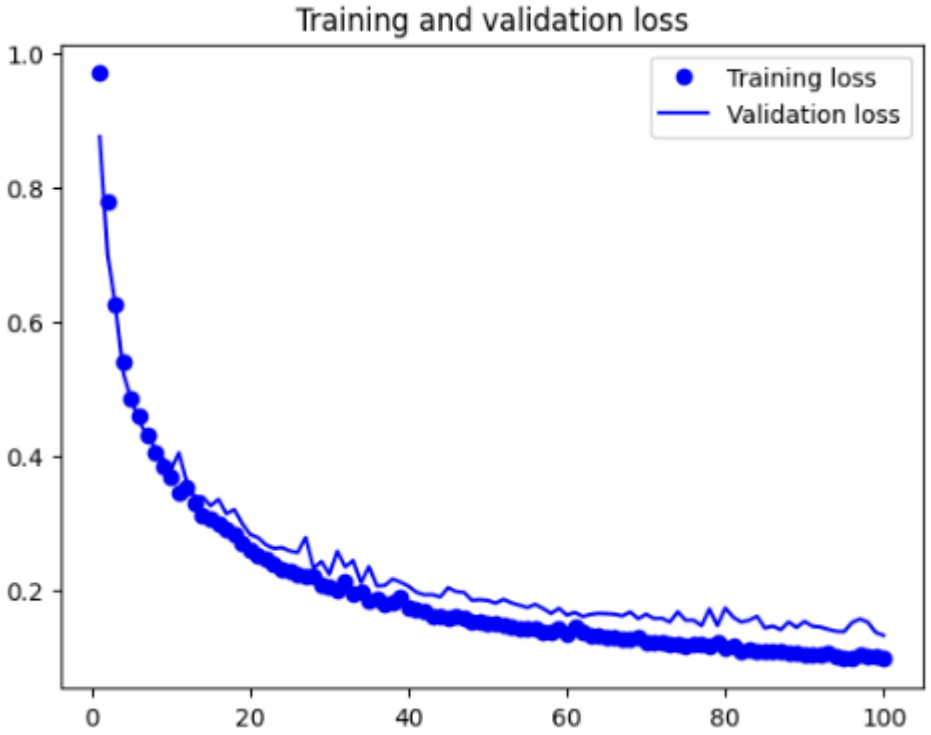


+ Ảnh quá trình chạy các epochs

```
Epoch 1/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 142ms/step - accuracy: 0.6267 - loss: 0.9992 - val_accuracy: 0.8930 - val_loss: 0.8757
Epoch 2/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 135ms/step - accuracy: 0.8540 - loss: 0.8239 - val_accuracy: 0.8946 - val_loss: 0.6993
Epoch 3/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 145ms/step - accuracy: 0.8886 - loss: 0.6516 - val_accuracy: 0.8850 - val_loss: 0.6227
Epoch 4/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 147ms/step - accuracy: 0.9053 - loss: 0.5604 - val_accuracy: 0.9217 - val_loss: 0.5224
Epoch 5/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 154ms/step - accuracy: 0.9314 - loss: 0.4882 - val_accuracy: 0.9361 - val_loss: 0.4842
Epoch 6/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 152ms/step - accuracy: 0.9270 - loss: 0.4680 - val_accuracy: 0.9409 - val_loss: 0.4515
Epoch 7/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 156ms/step - accuracy: 0.9420 - loss: 0.4384 - val_accuracy: 0.9377 - val_loss: 0.4310
Epoch 8/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 162ms/step - accuracy: 0.9458 - loss: 0.4091 - val_accuracy: 0.9409 - val_loss: 0.4122
Epoch 9/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 157ms/step - accuracy: 0.9518 - loss: 0.3840 - val_accuracy: 0.9489 - val_loss: 0.3967
Epoch 10/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 167ms/step - accuracy: 0.9537 - loss: 0.3693 - val_accuracy: 0.9489 - val_loss: 0.3801
Epoch 11/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 165ms/step - accuracy: 0.9634 - loss: 0.3442 - val_accuracy: 0.9393 - val_loss: 0.4051
Epoch 12/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 5s 164ms/step - accuracy: 0.9504 - loss: 0.3636 - val_accuracy: 0.9505 - val_loss: 0.3613
...
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 227ms/step - accuracy: 0.9973 - loss: 0.1010 - val_accuracy: 0.9920 - val_loss: 0.1332
31/31 ━━━━━━━━━━━━━━━━━━━━ 4s 136ms/step - accuracy: 0.9000 - loss: 0.7802
```

+ Sơ đồ hóa bằng matplotlib





+ Kiểm tra độ chính xác trên tập test với mô hình tốt nhất

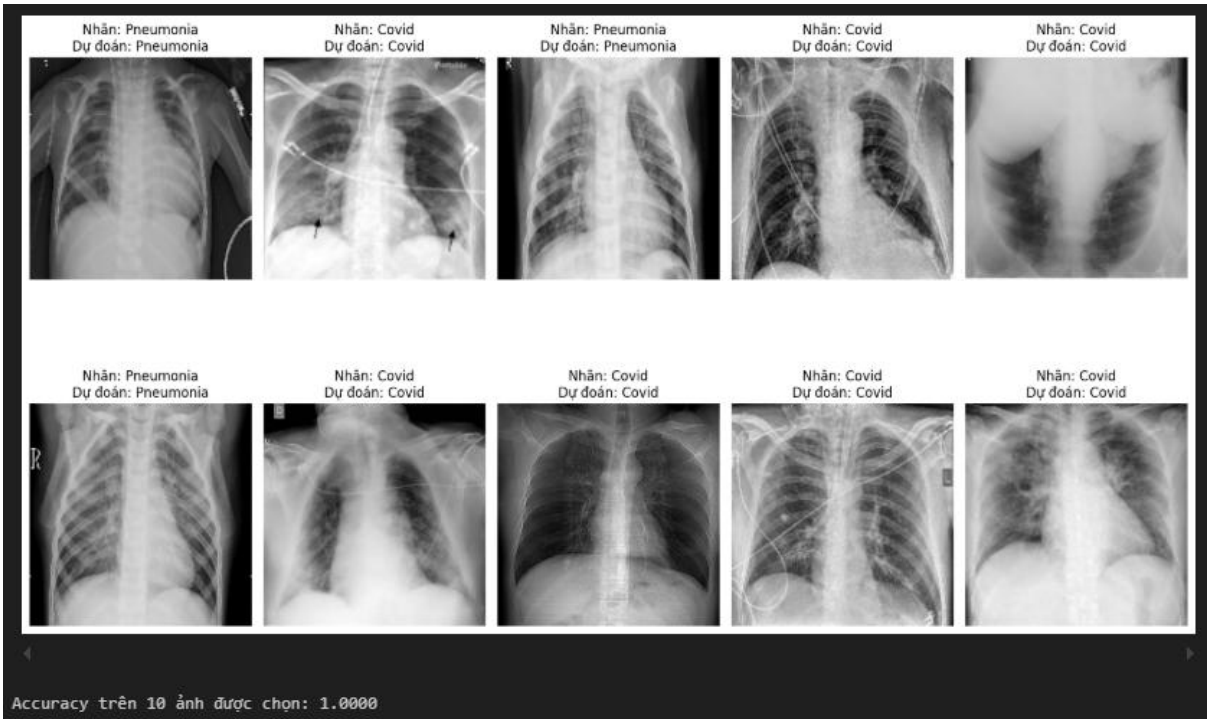
```python
import tensorflow as tf
from tensorflow.keras.utils import image_dataset_from_directory
from pathlib import Path
import matplotlib.pyplot as plt
from tensorflow.keras import config

# Định nghĩa đường dẫn đến thư mục dữ liệu
base_dir = Path(r"C:\Users\ASUS\Desktop\TTCS\xray_data1")

# Tạo tập dữ liệu thử nghiệm (test)
test_dataset = image_dataset_from_directory(
    base_dir / "test",
    image_size=(150, 150),
    batch_size=64,
    color_mode="grayscale",
    shuffle=False
)

# Tải mô hình đã lưu
model = tf.keras.models.load_model("best_model_covid.keras")

# Đánh giá accuracy trên tập test
test_loss, test_accuracy = model.evaluate(test_dataset)
print(f"Test accuracy: {test_accuracy:.4f}")
print(f"Test loss: {test_loss:.4f}")
```

✓ 20.9s

```
Found 1979 files belonging to 2 classes.
31/31 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.9313 - loss: 0.5571
Test accuracy: 0.8989
Test loss: 0.7913
```

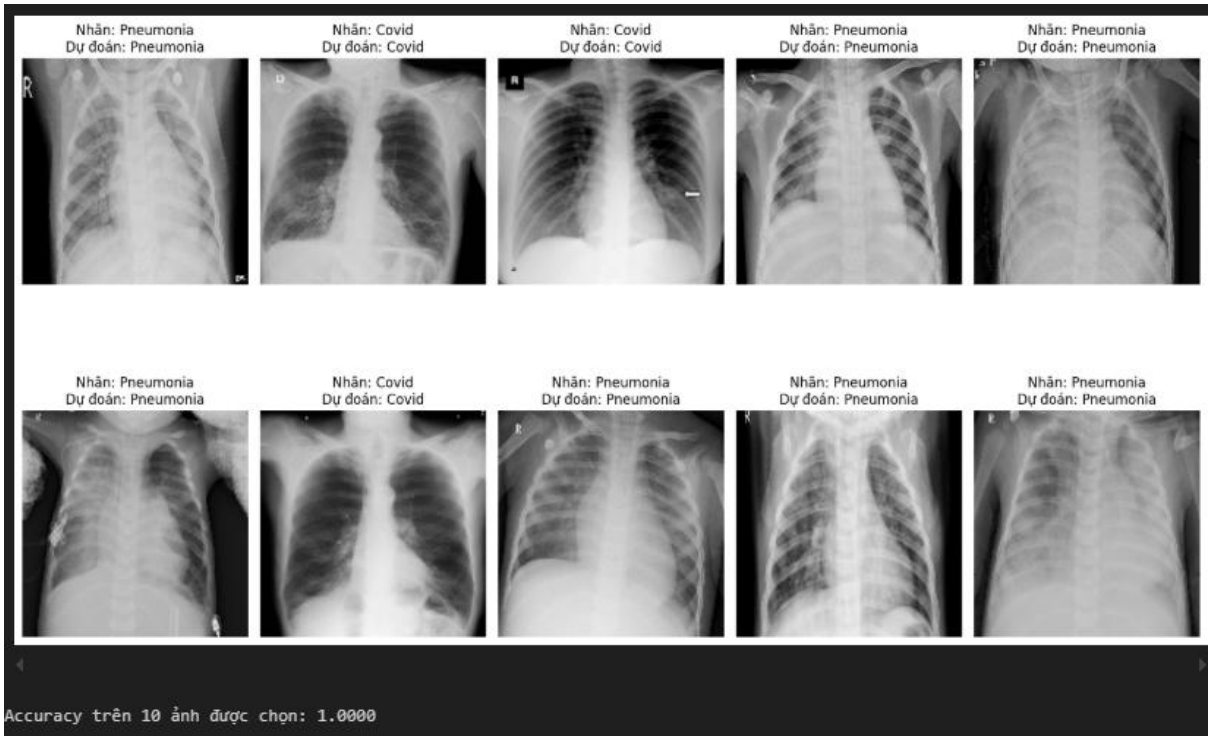
+ Triển khai ứng dụng như ở trên:

Dự đoán lần 1: độ chính xác 10/10

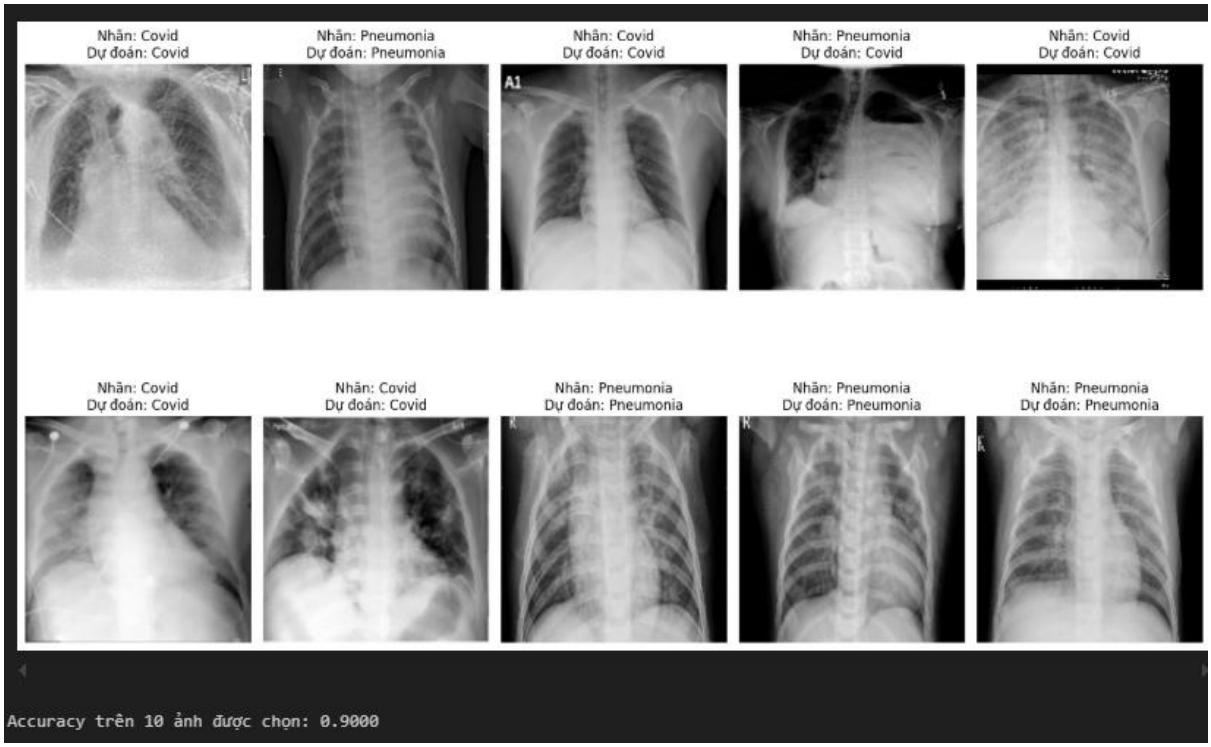



```
Accuracy trên 10 ảnh được chọn: 1.0000
```

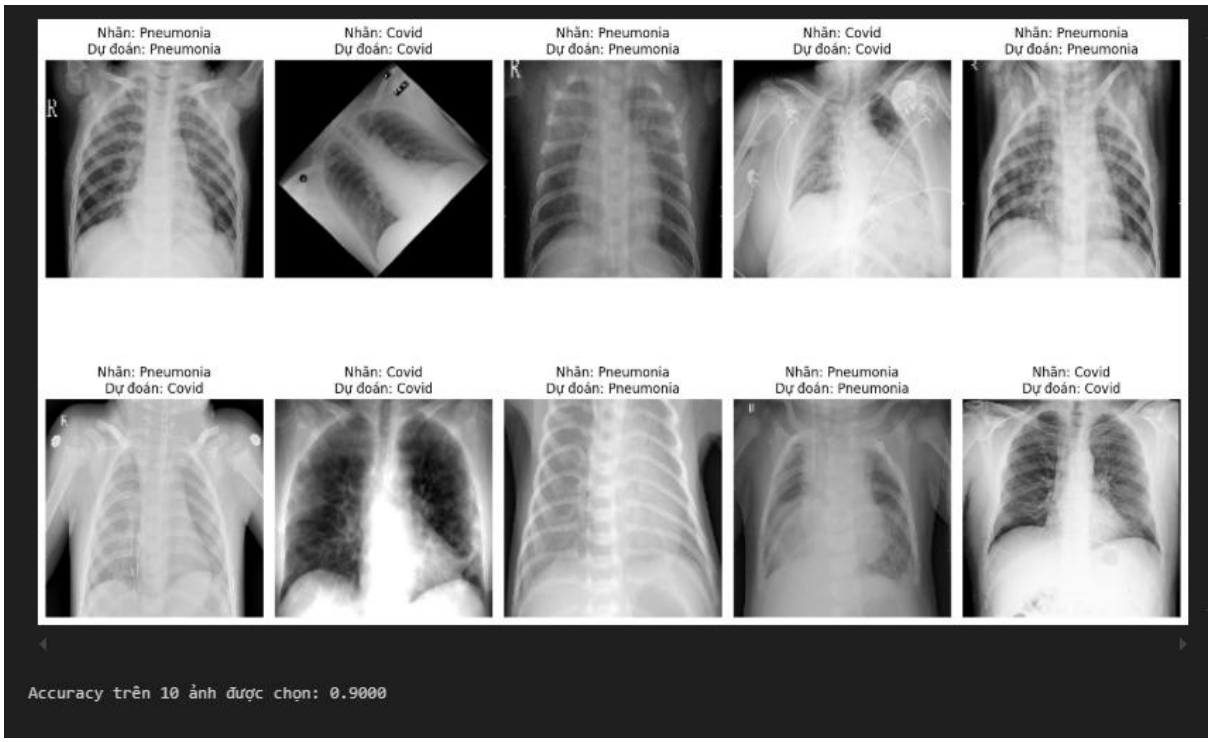
Dự đoán lần 2: độ chính xác 10/10



Dự đoán lần 3: độ chính xác 9/10

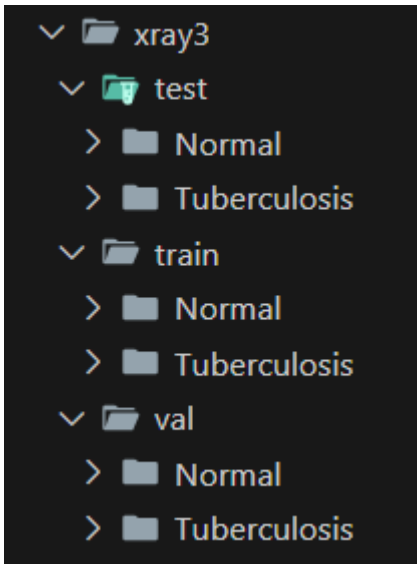


Dự đoán lần 4: độ chính xác 9/10



3.4 Huấn luyện mô hình với bộ dữ liệu Tuberculosis Chest X-ray Database

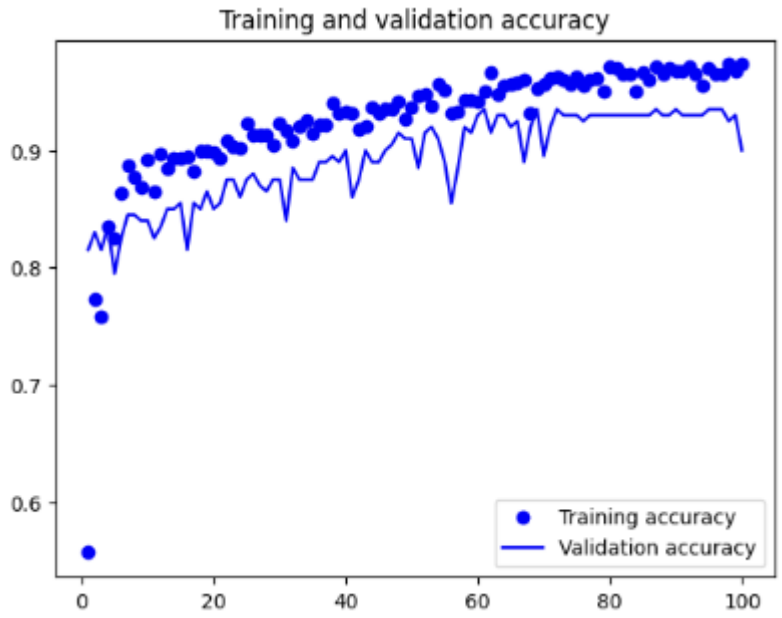
+ Model được lưu lại khi làm việc với tập dữ liệu này là best_model_xray3.keras
+ Độ chính xác trên tập kiểm tra đạt 88,83%, ngang bằng so với bộ dữ liệu đầu tiên. Lý do mà độ chính xác của nó thấp hơn so với sử dụng bộ dữ liệu thứ 2 là vì số dữ liệu để học khá ít, chỉ 300 ảnh cho mỗi loại
+ Cấu trúc thư mục

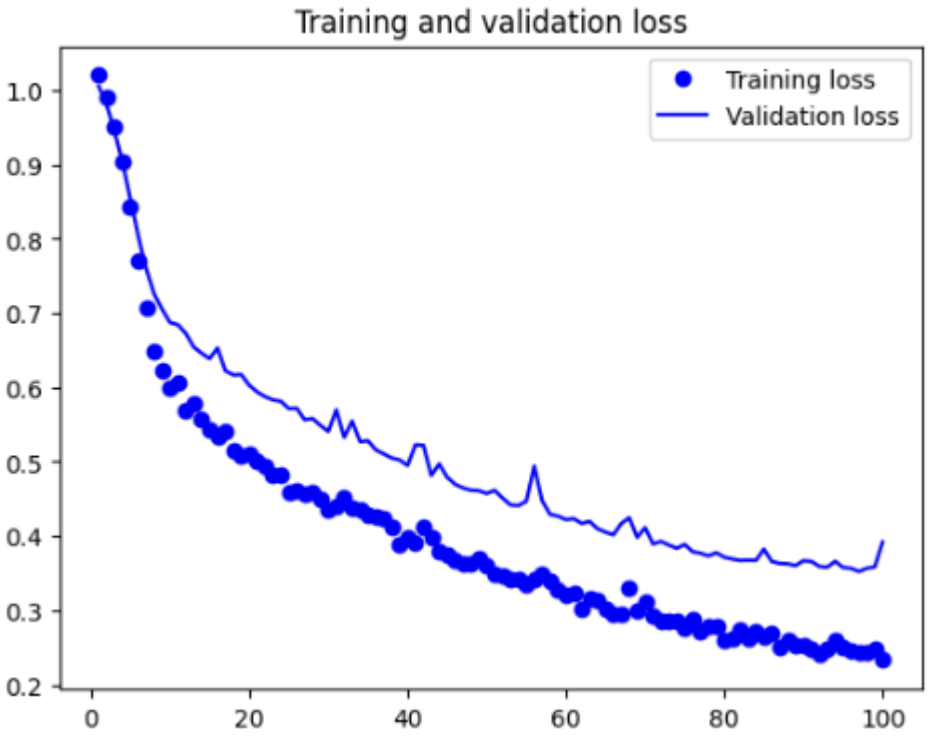


+ Quan sát độ chính xác theo từng epochs

```
Epoch 1/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 3s 132ms/step - accuracy: 0.5186 - loss: 1.0250 - val_accuracy: 0.8150 - val_loss: 1.0055
Epoch 2/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 0.7665 - loss: 0.9945 - val_accuracy: 0.8300 - val_loss: 0.9803
Epoch 3/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 79ms/step - accuracy: 0.7585 - loss: 0.9600 - val_accuracy: 0.8150 - val_loss: 0.9446
Epoch 4/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 77ms/step - accuracy: 0.8346 - loss: 0.9143 - val_accuracy: 0.8350 - val_loss: 0.9033
Epoch 5/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 78ms/step - accuracy: 0.8350 - loss: 0.8531 - val_accuracy: 0.7950 - val_loss: 0.8538
Epoch 6/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 77ms/step - accuracy: 0.8552 - loss: 0.7832 - val_accuracy: 0.8250 - val_loss: 0.8033
Epoch 7/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 79ms/step - accuracy: 0.8923 - loss: 0.7063 - val_accuracy: 0.8450 - val_loss: 0.7594
Epoch 8/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 77ms/step - accuracy: 0.8678 - loss: 0.6576 - val_accuracy: 0.8450 - val_loss: 0.7259
Epoch 9/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 0.8609 - loss: 0.6449 - val_accuracy: 0.8400 - val_loss: 0.7053
Epoch 10/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 81ms/step - accuracy: 0.8851 - loss: 0.5955 - val_accuracy: 0.8400 - val_loss: 0.6875
Epoch 11/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 0.8623 - loss: 0.5959 - val_accuracy: 0.8250 - val_loss: 0.6844
Epoch 12/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 81ms/step - accuracy: 0.8999 - loss: 0.5496 - val_accuracy: 0.8350 - val_loss: 0.6726
...
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 80ms/step - accuracy: 0.9713 - loss: 0.2380 - val_accuracy: 0.9000 - val_loss: 0.3919
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 75ms/step - accuracy: 0.8575 - loss: 0.5045
Test accuracy: 0.8617
Test loss: 0.5053
```

+ Sơ đồ hóa bằng matplotlib





+ Kiểm tra độ chính xác trên tập kiểm tra

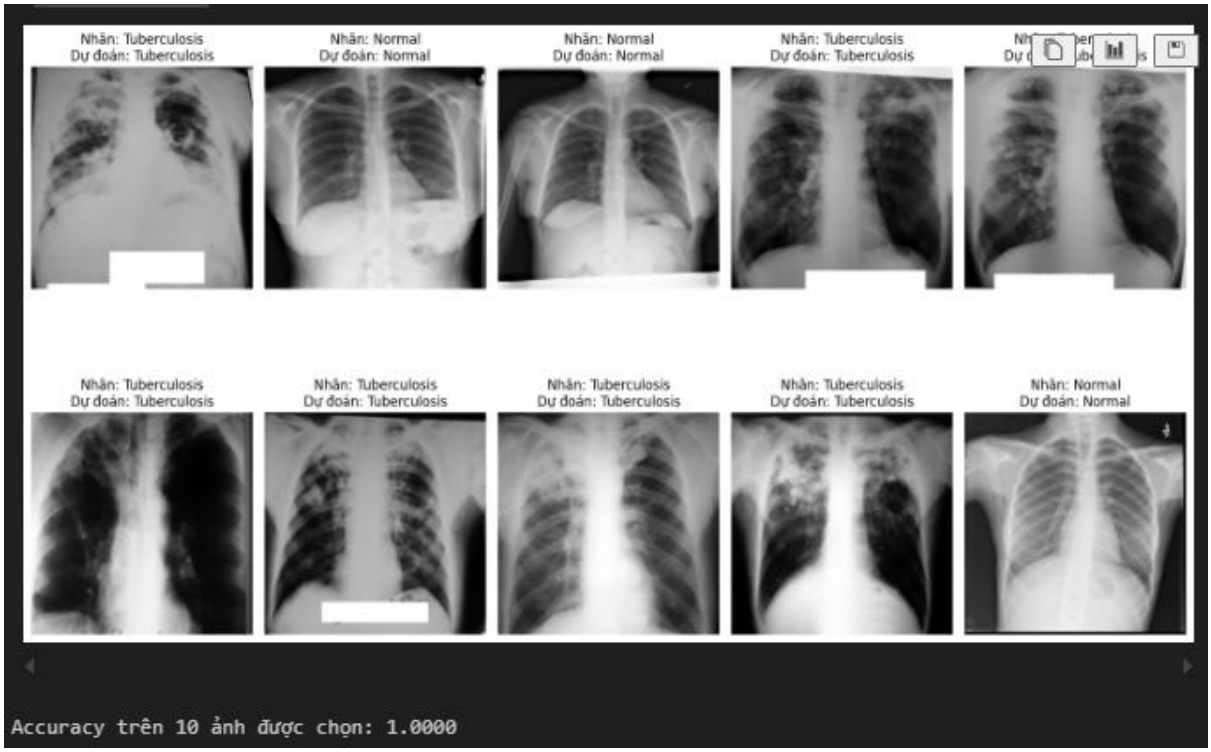
```python
import tensorflow as tf
from tensorflow.keras.utils import image_dataset_from_directory
from pathlib import Path
import matplotlib.pyplot as plt
from tensorflow.keras import config

# Định nghĩa đường dẫn đến thư mục dữ liệu
base_dir = Path(r"C:\Users\ASUS\Desktop\TTCS\xray3")

# Tạo tập dữ liệu thử nghiệm (test)
test_dataset = image_dataset_from_directory(
    base_dir / "test",
    image_size=(150, 150),
    batch_size=64,
    color_mode="grayscale",
    shuffle=False
)

# Tải mô hình đã lưu
model = tf.keras.models.load_model("best_model_xray3.keras")

# Đánh giá accuracy trên tập test
test_loss, test_accuracy = model.evaluate(test_dataset)
print(f"Test accuracy: {test_accuracy:.4f}")
print(f"Test loss: {test_loss:.4f}")
```

✓ 1.2s

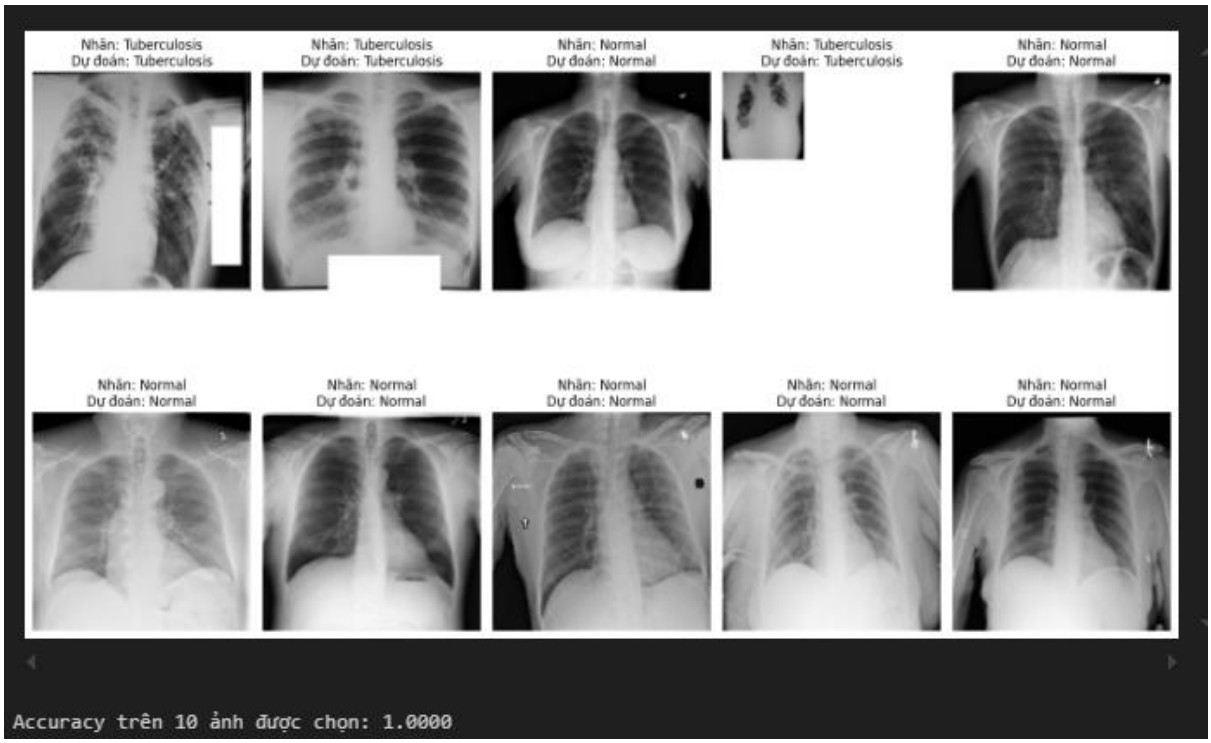
```
Found 600 files belonging to 2 classes.
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.8957 - loss: 0.4215
Test accuracy: 0.8883
Test loss: 0.4933
```

+ Triển khai ứng dụng

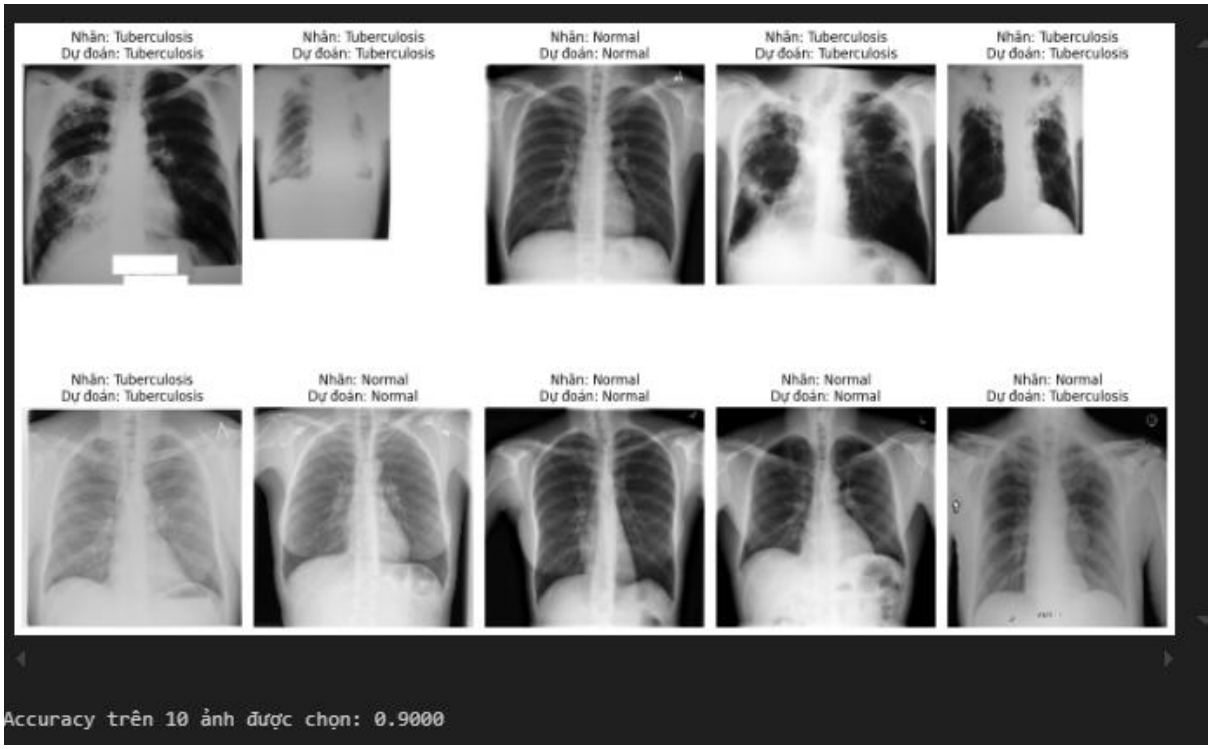
Lần 1: độ chính xác 10/10



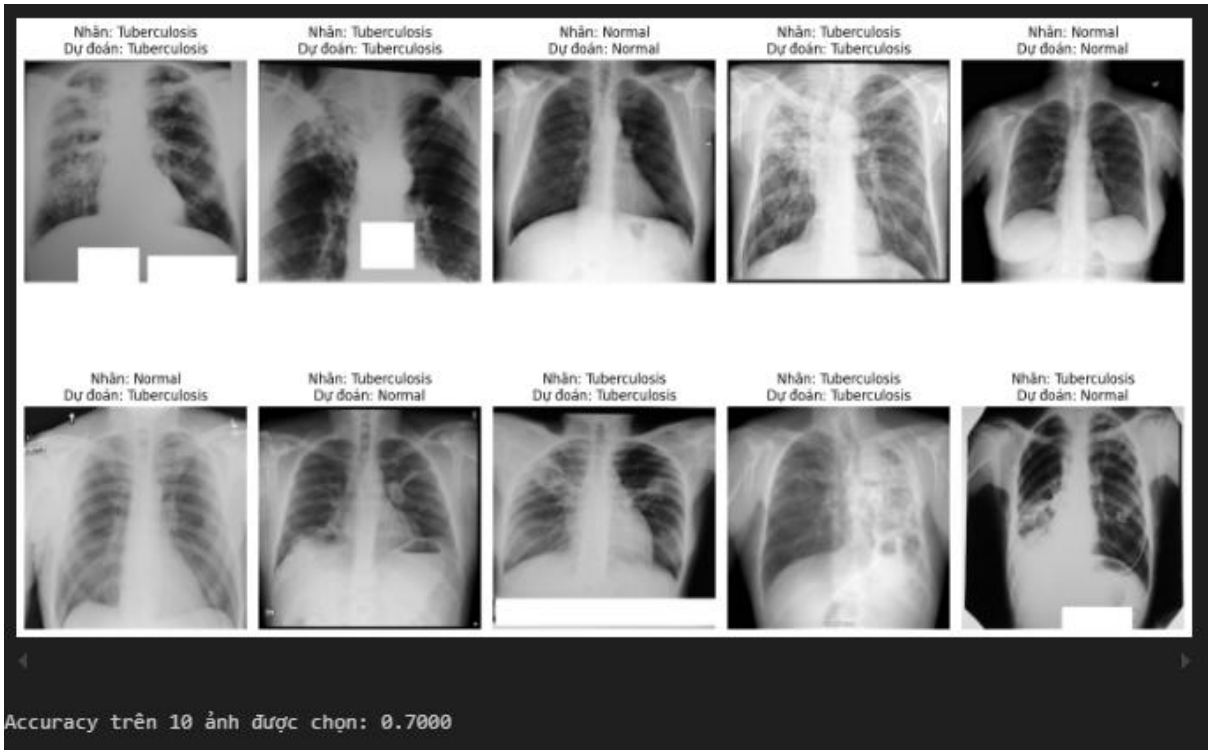
Lần 2: độ chính xác 10/10



Lần 3: độ chính xác 9/10

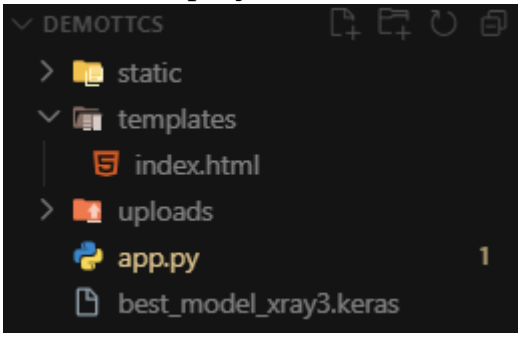


Lần 4: độ chính xác: 7/10

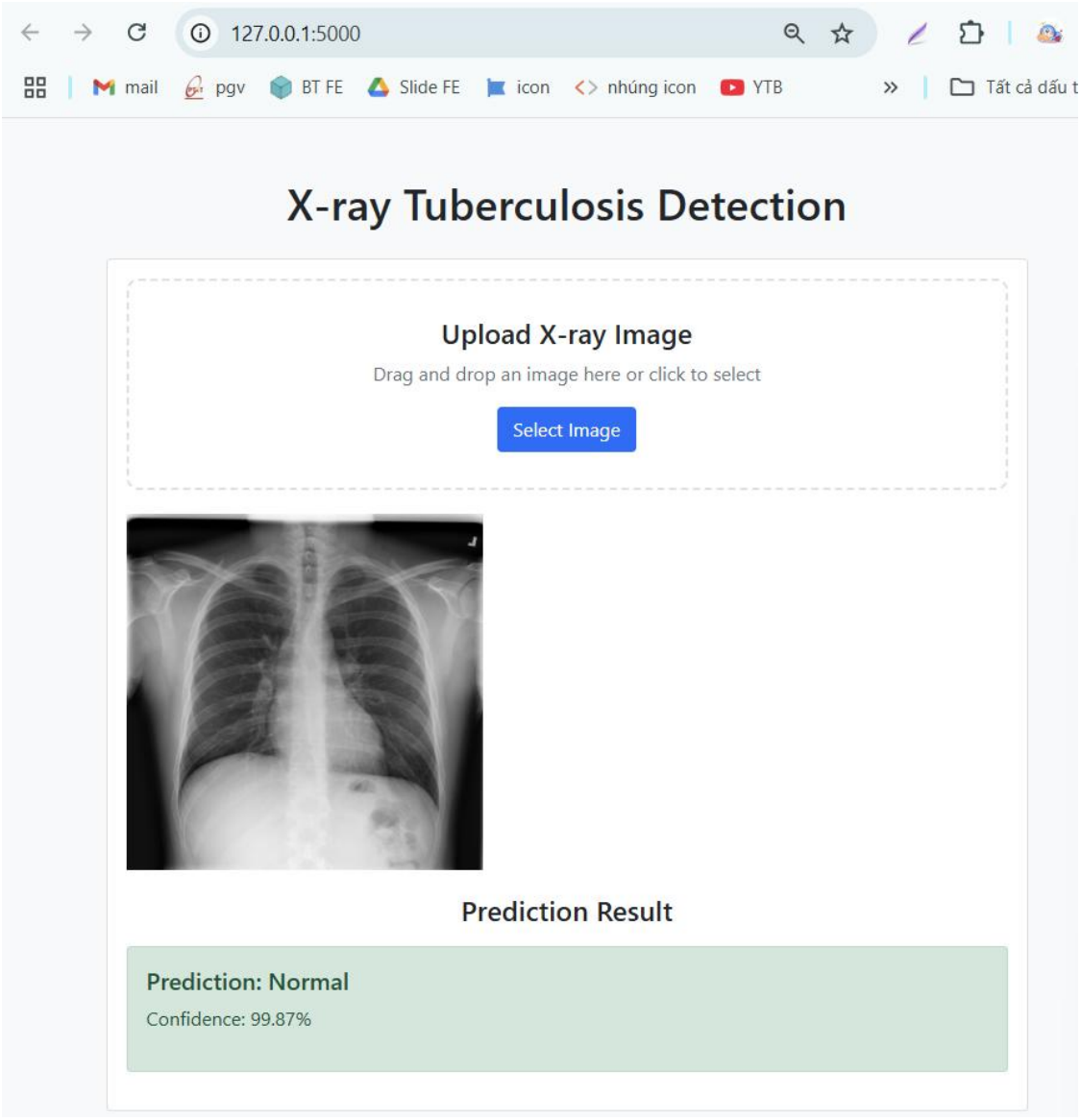


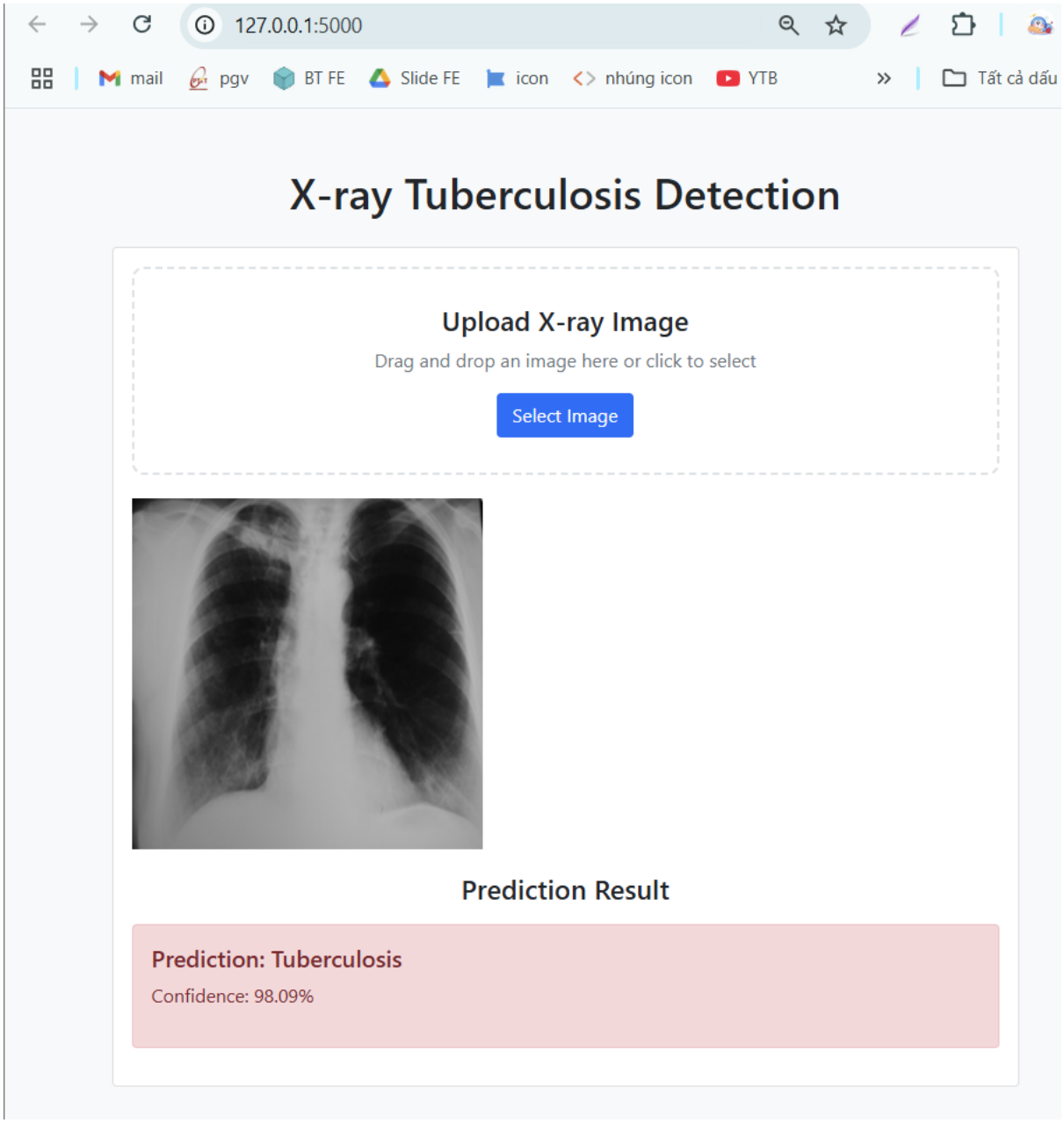
Với ví dụ này, em sẽ triển khai một ứng dụng web với Flask sử dụng model đã huấn luyện (làm tương tự với 2 model ở phần trên), kết quả thực hiện như sau:

+ Bước 1: tạo project



+ Bước 2: Thực hiện code 2 file app.py và index.html
  - app.py: https://ideone.com/NXHIAi
  - index.html: https://ideone.com/W4WhI8
+ Bước 3: Xem kết quả





3.5 Nhận xét chung

3.5.1 Hiệu suất tổng thể đối với các bộ dữ liệu

Các mô hình được huấn luyện trên ba bộ dữ liệu (Chest X-Ray Images (Pneumonia), COVID19_Pneumonia_Normal_Chest_Xray_PA_Dataset, và Tuberculosis Chest X-ray Database) cho thấy độ chính xác trung bình trên tập kiểm tra dao động từ 88% đến 90%. Cụ thể:

+ Bộ dữ liệu 1 (Chest X-Ray Images): Đạt độ chính xác 88,21%.
+ Bộ dữ liệu 2 (COVID19_Pneumonia_Normal_Chest_Xray_PA_Dataset): Đạt độ chính xác cao nhất, 89,89%.

+ Bộ dữ liệu 3 (Tuberculosis Chest X-ray Database): Đạt độ chính xác 88,83%.

Mức độ chính xác này cho thấy mô hình CNN được xây dựng có khả năng phân loại khá tốt các bệnh lý phổi thông qua ảnh X-quang, mặc dù kiến trúc mô hình khá đơn giản (chủ yếu sử dụng các tầng Conv2D, MaxPooling2D, và Dense). Tuy nhiên, sự khác biệt nhỏ về độ chính xác giữa các bộ dữ liệu phản ánh ảnh hưởng của kích thước dữ liệu và độ cân bằng giữa các lớp.

3.5.2 Hiệu quả của các kỹ thuật chống quá khớp

+ Data Augmentation: Giúp tăng sự đa dạng của dữ liệu huấn luyện, đặc biệt hiệu quả với bộ dữ liệu 2 và 3, nơi dữ liệu gốc không quá lớn. Các biến đổi như RandomFlip, RandomRotation, RandomContrast, và RandomBrightness phù hợp với ảnh X-quang, vì chúng làm nổi bật các đặc trưng quan trọng (như vùng mờ đục của viêm phổi hoặc lao phổi).
+ Dropout (0.5): Giảm sự phụ thuộc của mô hình vào một số nơ-ron cụ thể, giúp cải thiện hiệu suất trên tập kiểm tra, đặc biệt với bộ dữ liệu 1 và 3
+ L2 Regularization: Làm mượt mô hình, giảm giá trị các trọng số lớn, góp phần giảm quá khớp trên cả ba bộ dữ liệu.
+ Việc không sử dụng các kỹ thuật nâng cao như BatchNormalization hay Residual Connection là hợp lý, vì chúng có thể làm mô hình phức tạp hơn và gây quá khớp, đặc biệt với dữ liệu nhỏ (như đã thử nghiệm: độ chính xác trên tập validation chỉ đạt 75% khi áp dụng BatchNormalization, so với 88% khi không dùng).

3.6 Kết luận và hướng cải thiện

Việc ứng dụng học sâu, đặc biệt là mạng tích chập CNN, trong phân loại ảnh X-quang đã thể hiện tiềm năng lớn trong lĩnh vực y tế. Các bộ dữ liệu được phân tích cho thấy khả năng nhận diện các bệnh như viêm phổi, COVID-19, và lao phổi với độ chính xác cao, hỗ trợ bác sĩ trong việc đưa ra quyết định nhanh chóng và giảm thiểu sai sót.

Ý nghĩa của học sâu trong y tế không chỉ nằm ở việc cải thiện hiệu quả chẩn đoán mà còn ở khả năng mở rộng ứng dụng sang các lĩnh vực khác như phát hiện ung thư, bệnh tim mạch, hay các bệnh hiếm gặp thông qua hình ảnh y khoa. Tuy nhiên, các bộ dữ liệu hiện tại vẫn tồn tại hạn chế như độ chênh lệch giữa các lớp, kích thước dữ liệu nhỏ, hoặc tập validation không đủ đại diện.

Một số hướng cải thiện sau này:

+ Thu thập và xây dựng các bộ dữ liệu lớn hơn, cân bằng hơn giữa các lớp để cải thiện hiệu suất mô hình.

+ Phát triển các hệ thống hỗ trợ chẩn đoán tích hợp, có khả năng giải thích kết quả (explainable AI) để tăng độ tin cậy và hỗ trợ bác sĩ tốt hơn.
+ Ứng dụng học sâu vào các loại hình ảnh y khoa khác (MRI, CT) hoặc các bệnh lý phức tạp hơn, đồng thời thử nghiệm triển khai thực tế tại các bệnh viện để đánh giá hiệu quả trong môi trường thực.

CHƯƠNG IV: TỔNG HỢP CÁC THAY ĐỔI SO VỚI BẢN TIỂU LUẬN CŨ

4.1 Chỉnh sửa các lỗi chính tả

+ Đã sửa các lỗi chính tả trong bài tiểu luận cũ: (leaning rate thành learning rate, GRB thành RGB,...)
+ Đã chỉnh sửa trong việc sử dụng tiếng Anh và tiếng Việt lẫn nhau
+ Việc thêm các trích dẫn lý thuyết là không cần thiết, vì trong phạm vi môn học đều đang sử dụng kiến thức và dữ liệu từ sách "Deep learning with Python – Second Edition" của Francois Chollet

4.2 Chỉnh sửa độ chi tiết trong lý thuyết

4.2.1 Phần Regularization

1. Bổ sung lý thuyết: Regularization là quá trình thêm một hàm phạt (vào hàm mất mát (loss function) hoặc áp dụng các kỹ thuật để hạn chế độ phức tạp của mô hình, giúp mô hình không học các chi tiết nhiễu (noise) trong dữ liệu huấn luyện mà tập trung vào các đặc trưng tổng quát, có khả năng khái quát hóa tốt trên dữ liệu mới.
2. Bổ sung ưu – nhược điểm của kỹ thuật này:

+ Ưu điểm:
  – Dễ triển khai, được tích hợp sẵn trong các framework, thư viện như TensorFlow, Keras
  – L2 thường hiệu quả hơn trong các mô hình DL vì nó không làm mất quá nhiều thông tin
  – L1 hữu ích khi cần giảm số lượng đặc trưng hoặc tham số
+ Nhược điểm:
  – Cần điều chỉnh siêu tham số λ, nếu chọn sai có thể làm mô hình thiếu khớp.
  – L1 có thể gây ra gradient không ổn định trong một số trường hợp

3. Bổ sung code minh họa cho kỹ thuật này

```
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation='relu',
                 kernel_regularizer=regularizers.l2(0.01)),
    layers.Dense(64, activation='relu',
                 kernel_regularizer=regularizers.l1(0.01)),
    layers.Dense(10, activation='softmax')
])
```

4.2.2 Phần Callbacks

+ Bổ sung code minh họa:

```
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

checkpoint = ModelCheckpoint(
    filepath='best_model.keras',
    monitor='val_loss',
    save_best_only=True
)

history = model.fit(
    x_train, y_train,
    epochs=20,
    batch_size=32,
    validation_data=(x_val, y_val),
    callbacks=[early_stopping, checkpoint]
)
```

4.3 Chỉnh sửa phần thực hành

4.3.1 Bổ sung lý cho chọn chủ đề và ý nghĩa

Việc chọn chủ đề ứng dụng học sâu để nhận diện tình trạng bệnh thông qua ảnh X-quang xuất phát từ nhu cầu thực tiễn trong lĩnh vực y tế, nơi mà việc chẩn đoán nhanh và chính xác các bệnh về phổi như viêm phổi, COVID-19, hay lao phổi có vai trò quan trọng trong việc cải thiện tỷ lệ sống và giảm áp lực cho hệ thống y tế. Ảnh X-quang là một phương pháp chẩn đoán hình ảnh phổ biến, không xâm lấn, và chi phí thấp, nhưng việc phân tích thủ công bởi bác sĩ thường tốn thời gian và phụ thuộc vào kinh nghiệm

Học sâu, đặc biệt là mạng tích chập CNN, có khả năng tự động trích xuất đặc trưng từ ảnh X-quang, hỗ trợ bác sĩ trong việc phát hiện bệnh một cách nhanh chóng, chính xác, và giảm thiểu sai sót do yếu tố con người

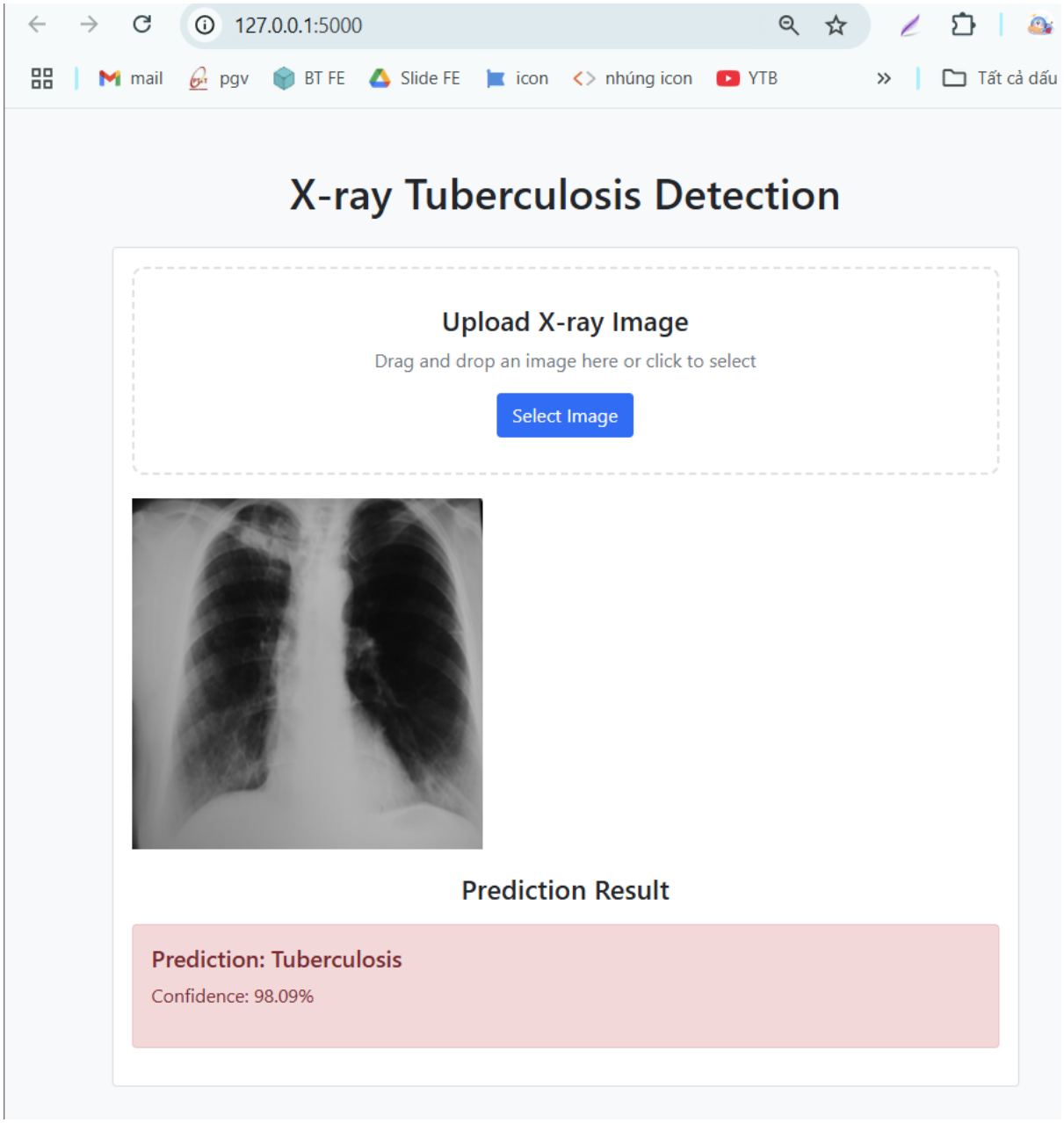
Chủ đề này không chỉ có ý nghĩa trong việc nâng cao hiệu quả chẩn đoán y tế mà còn góp phần thúc đẩy ứng dụng công nghệ trí tuệ nhân tạo vào các lĩnh vực chăm sóc sức khỏe, đặc biệt tại các khu vực thiếu hụt bác sĩ chuyên môn

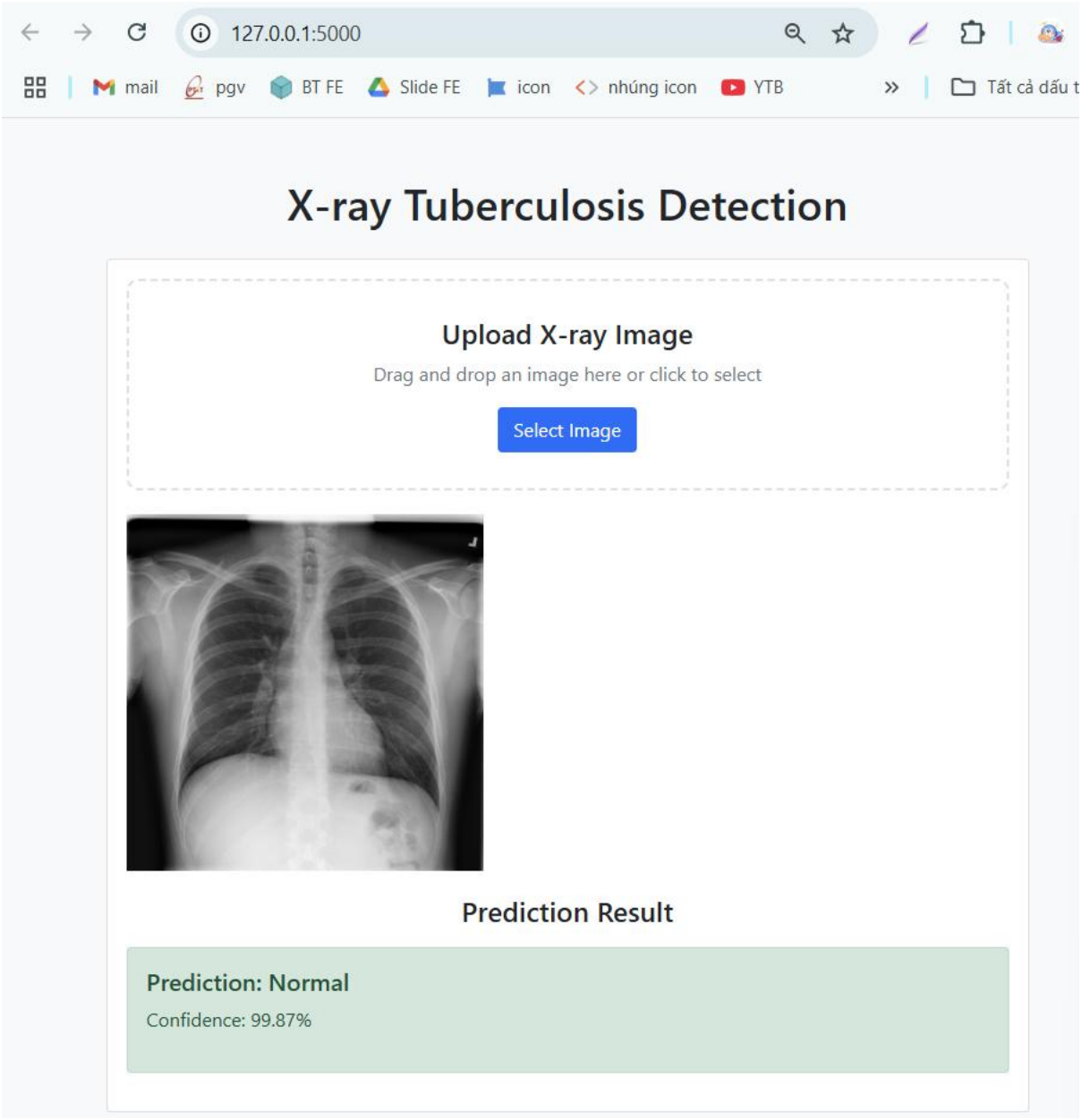
Tuy nhiên, việc ứng dụng học sâu trong lĩnh vực này chỉ có ý nghĩa hỗ trợ các bác sĩ trong việc chẩn đoán, không thể nào thay thế vai trò thực sự của bác sĩ trong lĩnh vực y tế

4.3.2 Bổ sung bảng nhận xét, so sánh giữa 3 bộ dữ liệu

STT	Tập train	Tập val	Tập test	Nhận xét
1	5216	55	585	- Tập train: 1314 Normal – 3875 Pneumonia: độ chênh lệch giữa dữ liệu ứng với 2 nhãn là khá lớn, gấp 3 lần - Tập val có khá ít dữ liệu, do đó, đôi khi các giá trị loss và accuracy trên tập val không thực sự đại diện cho sự thật - Đây là bộ dữ liệu không quá bé, cỡ vừa nhưng có độ chênh lệch giữa các lớp khá lớn
2	2000	600	2000	- Tập train: tỉ lệ dữ liệu giữa các lớp là 1:1, không có chênh lệch - Tỉ lệ dữ liệu giữa 3 tập dữ liệu của cả 2 bộ dữ liệu đều khá hợp lý - Bộ dữ liệu 2: là bộ dữ liệu nhỏ - Bộ dữ liệu 3: là bộ dữ liệu rất nhỏ
3	600	200	600	

4.3.3 Bổ sung phần triển khai ứng dụng cụ thể hơn





4.3.4 Bổ sung nhận xét chi tiết hơn

a) Hiệu suất tổng thể đối với các bộ dữ liệu

Các mô hình được huấn luyện trên ba bộ dữ liệu (Chest X-Ray Images (Pneumonia), COVID19_Pneumonia_Normal_Chest_Xray_PA_Dataset, và Tuberculosis Chest X-ray Database) cho thấy độ chính xác trung bình trên tập kiểm tra dao động từ 88% đến 90%. Cụ thể:

+ Bộ dữ liệu 1 (Chest X-Ray Images): Đạt độ chính xác 88,21%.
+ Bộ dữ liệu 2 (COVID19_Pneumonia_Normal_Chest_Xray_PA_Dataset): Đạt độ chính xác cao nhất, 89,89%.

+ Bộ dữ liệu 3 (Tuberculosis Chest X-ray Database): Đạt độ chính xác 88,83%.

Mức độ chính xác này cho thấy mô hình CNN được xây dựng có khả năng phân loại khá tốt các bệnh lý phổi thông qua ảnh X-quang, mặc dù kiến trúc mô hình khá đơn giản (chủ yếu sử dụng các tầng Conv2D, MaxPooling2D, và Dense). Tuy nhiên, sự khác biệt nhỏ về độ chính xác giữa các bộ dữ liệu phản ánh ảnh hưởng của kích thước dữ liệu và độ cân bằng giữa các lớp.

b) Hiệu quả của các kỹ thuật chống quá khớp

+ Data Augmentation: Giúp tăng sự đa dạng của dữ liệu huấn luyện, đặc biệt hiệu quả với bộ dữ liệu 2 và 3, nơi dữ liệu gốc không quá lớn. Các biến đổi như RandomFlip, RandomRotation, RandomContrast, và RandomBrightness phù hợp với ảnh X-quang, vì chúng làm nổi bật các đặc trưng quan trọng (như vùng mờ đục của viêm phổi hoặc lao phổi).
+ Dropout (0.5): Giảm sự phụ thuộc của mô hình vào một số nơ-ron cụ thể, giúp cải thiện hiệu suất trên tập kiểm tra, đặc biệt với bộ dữ liệu 1 và 3
+ L2 Regularization: Làm mượt mô hình, giảm giá trị các trọng số lớn, góp phần giảm quá khớp trên cả ba bộ dữ liệu.
+ Việc không sử dụng các kỹ thuật nâng cao như BatchNormalization hay Residual Connection là hợp lý, vì chúng có thể làm mô hình phức tạp hơn và gây quá khớp, đặc biệt với dữ liệu nhỏ (như đã thử nghiệm: độ chính xác trên tập validation chỉ đạt 75% khi áp dụng BatchNormalization, so với 88% khi không dùng).

4.3.5 Bổ sung phần kết luận và hướng cải thiện sau này

Việc ứng dụng học sâu, đặc biệt là mạng tích chập CNN, trong phân loại ảnh X-quang đã thể hiện tiềm năng lớn trong lĩnh vực y tế. Các bộ dữ liệu được phân tích cho thấy khả năng nhận diện các bệnh như viêm phổi, COVID-19, và lao phổi với độ chính xác cao, hỗ trợ bác sĩ trong việc đưa ra quyết định nhanh chóng và giảm thiểu sai sót.

Ý nghĩa của học sâu trong y tế không chỉ nằm ở việc cải thiện hiệu quả chẩn đoán mà còn ở khả năng mở rộng ứng dụng sang các lĩnh vực khác như phát hiện ung thư, bệnh tim mạch, hay các bệnh hiếm gặp thông qua hình ảnh y khoa. Tuy nhiên, các bộ dữ liệu hiện tại vẫn tồn tại hạn chế như độ chênh lệch giữa các lớp, kích thước dữ liệu nhỏ, hoặc tập validation không đủ đại diện.

Một số hướng cải thiện sau này:

+ Thu thập và xây dựng các bộ dữ liệu lớn hơn, cân bằng hơn giữa các lớp để cải thiện hiệu suất mô hình.

+ Phát triển các hệ thống hỗ trợ chẩn đoán tích hợp, có khả năng giải thích kết quả (explainable AI) để tăng độ tin cậy và hỗ trợ bác sĩ tốt hơn.
+ Ứng dụng học sâu vào các loại hình ảnh y khoa khác (MRI, CT) hoặc các bệnh lý phức tạp hơn, đồng thời thử nghiệm triển khai thực tế tại các bệnh viện để đánh giá hiệu quả trong môi trường thực.