LỜI CAM ĐOAN

Tôi xin cam đoan: Luận văn thạc sỹ chuyên ngành Khoa học máy tính, tên đề tài "Nghiên cứu hỗ trợ chuẩn đoán bệnh lao dựa vào học máy" là công trình nghiên cứu, tìm hiểu và trình bày do tôi thực hiện dưới sự hướng dẫn khoa học của **PGS.TS. Đỗ Năng Toàn**, Viện Công nghệ Thông tin, Viện Hàn lâm Khoa học và Công nghệ Việt Nam.

Kết quả tìm hiểu, nghiên cứu trong luận văn là hoàn toàn trung thực, không vi phạm bất cứ điều gì trong luật sở hữu trí tuệ và pháp luật Việt Nam. Nếu sai, tôi hoàn toàn chịu trách nhiệm trước pháp luật.

Tất cả các tài liệu, bài báo, khóa luận, công cụ phần mềm của các tác giả khác được sử dụng lại trong luận văn này đều được chỉ dẫn tường minh về tác giả và đều có trong danh mục tài liệu tham khảo.

Thái Nguyên, ngày 30 tháng 6 năm 2022.

Tác giả luận văn

Nguyễn Hữu Khánh

LỜI CẢM ƠN

Tác giả xin chân thành cảm ơn **PGS.TS. Đỗ Năng Toàn**, Viện Công nghệ Thông tin, Viện Hàn lâm Khoa học và Công nghệ Việt Nam, là giáo viên hướng dẫn khoa học đã hướng dẫn tác giả hoàn thành luận văn này, xin được cảm ơn các thầy, cô giáo trường Đại học công nghệ thông tin và truyền thông nơi tác giả theo học và hoàn thành chương trình cao học đã nhiệt tình giảng dạy và giúp đỡ.

Xin cảm ơn Trung tâm Đào tạo Từ xa - Đại học Thái Nguyên nơi tác giả công tác đã tạo mọi điều kiện thuận lợi để tác giả có thời gian, tâm trí để hoàn thành nhiệm vụ nghiên cứu và chương trình học tập.

Và cuối cùng xin cảm ơn gia đình, bạn bè, đồng nghiệp đã động viên, giúp đỡ tác giả trong suốt thời gian học tập, nghiên cứu và hoàn thành luận văn này.

Xin chân thành cảm ơn.

Thái Nguyên, ngày 30 tháng 06 năm 2022

Tác giả luận văn

Nguyễn Hữu Khánh

Danh sách hình vẽ

Mục lục

MỞ ĐẦU

Theo báo cáo của Tổ chức Y tế thế giới (TCYTTG - WHO Report 2020 - Global Tuberculosis Control)[1], mặc dù đã đạt được một số thành tựu đáng kể trong công tác chống lao trong thời gian qua, bệnh lao vẫn đang tiếp tục là một trong các vấn đề sức khoẻ cộng đồng chính trên toàn cầu. TCYTTG ước tính năm 2019 trên toàn cầu có khoảng 10 triệu người hiện mắc lao, một con số đã giảm rất chậm trong những năm gần đây; 8,2% trong số mắc lao có đồng nhiễm HIV. Bệnh lao là nguyên nhân gây tử vong đứng hàng thứ hai trong các bệnh nhiễm trùng với khoảng 1,2 triệu người tử vong do lao và khoảng 208.000 người chết do lao trong số những người nhiễm HIV. Số tử vong này làm cho lao là một trong các bệnh gây tử vong hàng đầu ở nữ giới. WHO đã công bố kết quả của mô hình đánh giá tác động ngắn hạn của đại dịch Covid-19 lên số ca tử vong do lao trong năm 2020. Kết quả cho thấy rằng tử vong do lao có thể tăng đáng kể trong năm 2020 và sẽ ảnh hưởng đến nhóm bệnh nhân lao dễ bị tổn thương nhất, tăng khoảng 200.000 – 400.000 ca tử vong, nếu như các dịch vụ chẩn đoán và phát hiện bệnh nhân trên toàn cầu bị ngưng trệ và giảm từ 25 – 50% trong khoảng 3 tháng. Con số tử vong sẽ tương ứng với mức tử vong toàn cầu do lao vào năm 2015, một bước lùi nghiêm trọng trong quá trình hướng tới mục tiêu của Hội nghị Cấp Cao Liên Hợp Quốc về Lao và Chiến lược thanh toán bệnh lao của WHO.

Về Việt nam, hiện chúng ta vẫn là nước có gánh nặng bệnh lao cao, đứng thứ 11 trong 30 nước có số người bệnh lao cao nhất trên toàn cầu, đồng thời đứng thứ 11 trong số 30 nước có gánh nặng bệnh lao kháng đa thuốc cao nhất thế giới[1]. Bệnh lao vẫn là một trong những bệnh truyền nhiễm phổ biến ở Việt Nam. Hàng năm, ước tính có 17.000 trường hợp tử vong do lao tại Việt Nam, cao hơn gấp hai lần so với con số tử vong do tai nạn giao thông. Mỗi năm ước tính có 180.000 người có bệnh lao hoạt động; 5.000 trường hợp trong số đó được xác định nhiễm lao kháng đa thuốc. Chẩn đoán bệnh lao không thật sự khó trong đa số các trường hợp. Điều đáng chú ý là làm sao chẩn đoán sớm và chẩn đoán đúng để khởi động điều trị sớm nhằm giảm các tổn thương cũng như biến chứng của lao gây ra. Để làm được điều trên, việc ứng dụng công nghệ thông tin vào quá trình chuẩn đoán là thực sự cần thiết, đặc biệt là áp dụng những tiến bộ của học máy để xây dựng lên hệ thống hỗ trợ chuẩn đoán bệnh lao.

Đối tượng nghiên cứu: Ảnh X-quang lồng ngực trong y tế thu nhận bởi các máy chiếu, chụp chuyên dụng.

Phạm vi nghiên cứu: Ảnh đa mức xám chụp phổi thẳng thường quy (tư thế sau - trước), chụp phổi nghiêng thường quy và chụp đỉnh phổi tư thế ưỡn ngực.

Những nội dung nghiên cứu chính: Dự kiến nội dung báo cáo của luận văn gồm: phần mở đầu, 3 chương chính, phần kết luận, tài liệu tham khảo, phụ lục. Bố cục được trình bày như sau:

Phần mở đầu: Nêu lý do chọn đề tài và hướng nghiên cứu chính

Chương 1: Khái quát về CNN và bài toán chuẩn đoán bệnh lao.

Chương 2: Một số mô hình hỗ trợ chuẩn đoán.

Chương 3: Chương trình thử nghiệm.

Mặc dù đã có cố gắng nỗ lực, song luận văn không tránh khỏi những thiếu sót do năng lực và thời gian hạn chế. Em chân thành mong muốn lắng nghe những đóng góp, góp ý của thầy, cô, bạn bè, đồng nghiệp để luận văn được cải thiện tốt hơn.

Em xin chân thành cảm ơn.

CHƯƠNG 1

Khái quát về CNN và bài toán chuẩn đoán bệnh lao.

1.1. Khái quát về CNN

1.1.1. Giới thiệu

Tương tự như việc trẻ em học cách nhận diện đối tượng, chúng ta cần cho thuật toán học rất nhiều hình ảnh trước khi nó có thể đưa ra phân loại cho hình ảnh đầu vào mà nó chưa từng thấy [3].

Hình 1.1: Cách máy tính "nhìn" một hình ảnh.

Máy tính "nhìn" theo cách khác con người. Trong thế giới máy tính chỉ có những con số. Mỗi hình ảnh có thể được biểu diễn dưới dạng mảng 2 chiều những con số được gọi là các pixel.

Mặc dù máy tính nhìn nhận theo cách khác con người, chúng ta vẫn có thể dạy máy tính nhận diện các mẫu như con người. Điều quan trọng là chúng ta cần nghĩ về hình ảnh theo một cách khác đi.

Để dạy thuật toán nhận diện đối tượng trong hình ảnh, ta sử dụng một loại mạng ANN, đó là Convolutional Neural Networks (CNN). Tên của nó

được dựa trên phép tính quan trọng được sử dụng trong mạng- tích chập.

Convolutional Neural Networks (CNN) là một trong những mô hình deep learning phổ biến nhất và có ảnh hưởng nhiều nhất trong cộng đồng Computer Vision. CNN được dùng trong trong nhiều bài toán như nhân dạng ảnh, phân tích video, ảnh MRI, hoặc cho bài các bài của lĩnh vự xử lý ngôn ngữ tự nhiên, và hầu hết đều giải quyết tốt các bài toán này.

Mạng CNN lấy cảm hứng từ não người [6]. Nghiên cứu trong những thập niên 1950 và 1960 của D.H Hubel và T.N Wiesel trên não của động vật đã đề xuất một mô hình mới cho việc cách mà động vật nhìn nhận thế giới. Trong báo cáo, hai ông đã diễn tả 2 loại tế bào neural trong não và cách hoạt động khác nhau: tế bào đơn giản (simple cell – S cell) và tế bào phức tạp (complex cell – C cell).

Các tế bào đơn giản được kích hoạt khi nhận diện các hình dáng đơn dản như đường nằm trong một khu vực cố định và một góc cạnh của nó. Các tế bào phức tạp có vùng tiếp nhận lơn hơn và đầu ra của nó không nhạy cảm với những vị trí cố định trong vùng.

Trong thị giác, vùng tiếp nhận của một neural tương ứng với một vùng trên võng mạc nơi mà sẽ kích hoạt neural tương ứng.

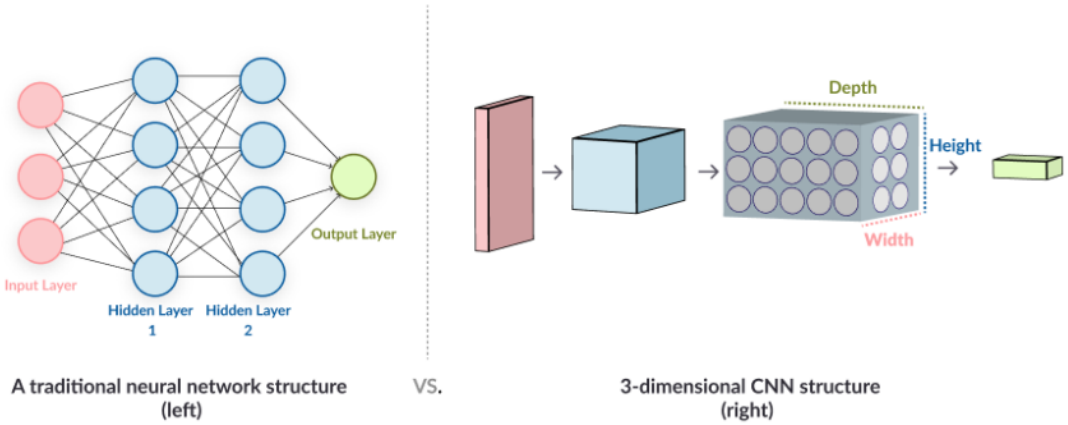
Năm 1980, Fukushima đề xuất mô hình mạng neural có cấp bậc gọi là neocognitron. Mô hình này dựa trên khái niệm về S cell và C cell. Mạn neocognitron có thể nhận diện mẫu dựa trên việc học hình dáng của đối tượng.

Sau đó vào năm 1998, mạng CNN được giới thiệu bởi Bengio, Le Cun, Bottou và Haffner. Mô hình đầu tiên của họ được gọi tên là LeNet-5. Mô hình này có thể nhận diện chữ số viết tay.

1.1.2. Kiến trúc mạng CNN

Mạng CNN có kiến trúc khác với Mạng Neural thông thường. Mạng ANN bình thường chuyển đổi đầu vào thông qua hàng loạt các tầng ẩn. Mỗi tầng là một tập các neural và các tầng được liên kết đầy đủ với các neural ở tầng trước đó. Và ở tầng cuối cùng sẽ là tầng kết quả đại diện cho dự đoán của mạng.

Đầu tiên, mạng CNN được chia thành 3 chiều: rộng, cao, và sâu. Kế đến, các neural trong mạng không liên kết hoàn toàn với toàn bộ neural lớp sau mà chỉ liên kết tới một vùng nhỏ. Cuối cùng, một tầng đầu ra được tối giản thành véc-tơ của giá trị xác suất (hình 1.2).



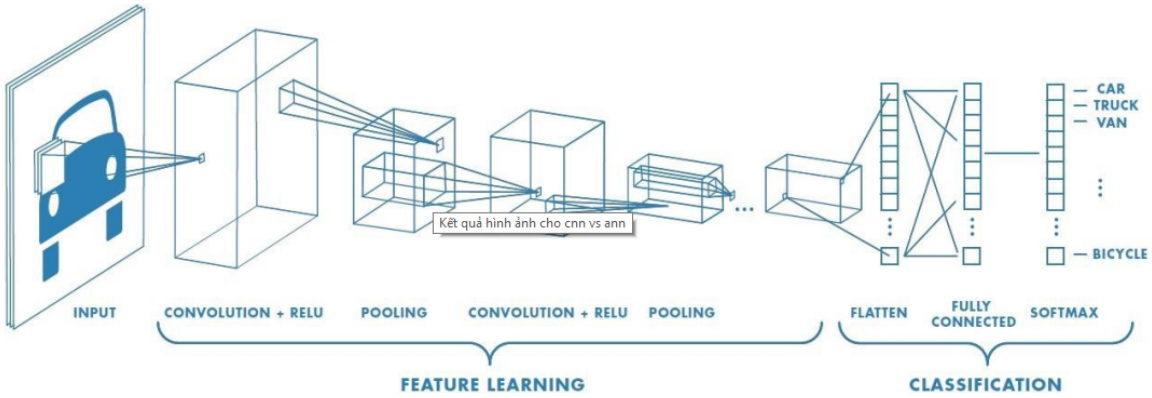
Hình 1.2: Mạng Neural thông thường (trái) và CNN (phải).

Mạng CNN gồm hai thành phần:

Phần tầng ẩn hay phần rút trích đặc trưng: trong phần này, mạng sẽ tiến hành tính toán hàng loạt phép tích chập và phép hợp nhất (pooling) để phát hiện các đặc trưng. Ví dụ: nếu ta có hình ảnh con ngựa vằn, thì trong phần này mạng sẽ nhận diện các sọc vằn, hai tai, và bốn chân của nó.

Phần phân lớp: tại phần này, một lớp với các liên kết đầy đủ sẽ

đóng vai trò như một bộ phân lớp các đặc trưng đã rút trích được trước đó. Tầng này sẽ đưa ra xác suất của một đối tượng trong hình 1.3.



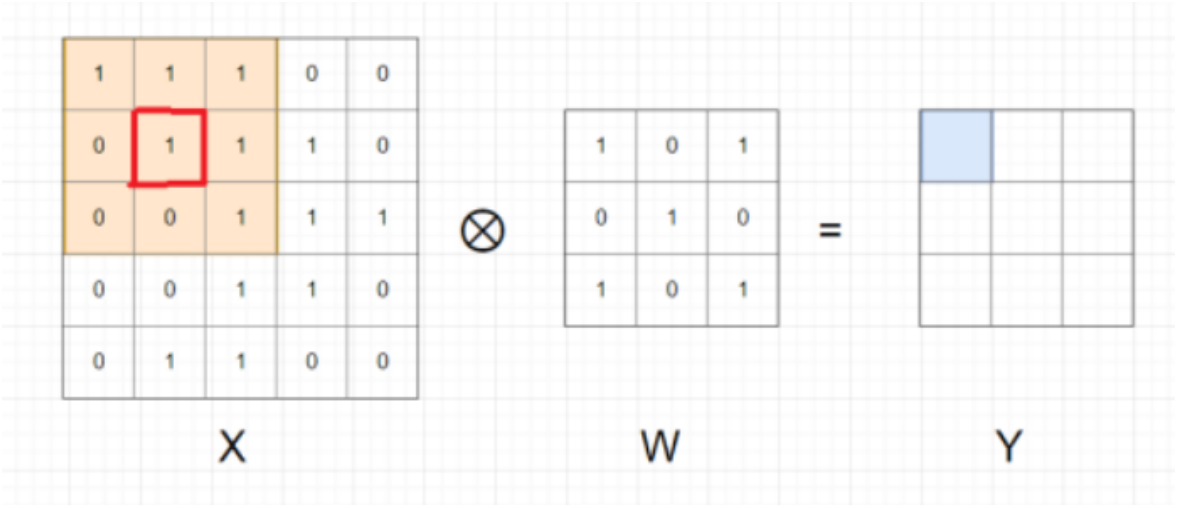
Hình 1.3: Kiến trúc mạng CNN.

1.1.2.1. Trích rút đặc trưng

1.1.2.1.1. Lớp tích chập Tích chập là một khối quan trọng trong CNN. Thuật ngữ tích chập được dựa trên một phép hợp nhất toán học của hai hàm tạo thành hàm thứ ba. Phép toán này kết hợp hai tập thông tin khác nhau.

Trong trường hợp CNN, tích chập được thực hiện trên giá trị đầu vào của dữ liệu và kernel/filter (thuật ngữ này được sử dụng khác nhau tùy tình huống) để tạo ra một bản đồ đặc trưng (feature map).

Ta thực hiện phép tích chập bằng cách trượt kernel/filter theo dữ liệu đầu vào. Tại mỗi vị trí, ta tiến hành phép nhân ma trận và tính tổng các giá trị để đưa vào bản đồ đặc trưng. Thao tác này đã được minh họa cụ thể trong hình 1.4



Hình 1.4: Minh họa phép tích chập.

Trong thực tế, tích chập được thực hiện hiện trên không gian 3 chiều. Vì mỗi hình ảnh được biểu diễn dưới dạng 3 chiều: rộng, cao, và sâu. Chiều sâu ở đây chính là giá trị màu sắc của hình (RGB).

Ta thực hiện phép tích chập trên đầu vào nhiều lần khác nhau. Mỗi lần sử dụng một kernel/filter khác nhau. Kết quả ta sẽ thu được những bản đồ đặc trưng khác nhau. Cuối cùng, ta kết hợp toàn bộ bản đồ đặc trưng này thành kết quả cuối cùng của tầng tích chập.

1.1.2.1.2. Lớp ReLU Tương tự như mạng neural thông thường, ta sử dụng một hàm kích hoạt (activate function) để có đầu ra dưới dạng phi tuyến. Trong trường hợp CNN, đầu ra của phép tích chập sẽ đi qua hàm kích hoạt nào đó ví dụ như hàm tinh chỉnh các đơn vị tuyến tính (Rectified linear units - ReLU).

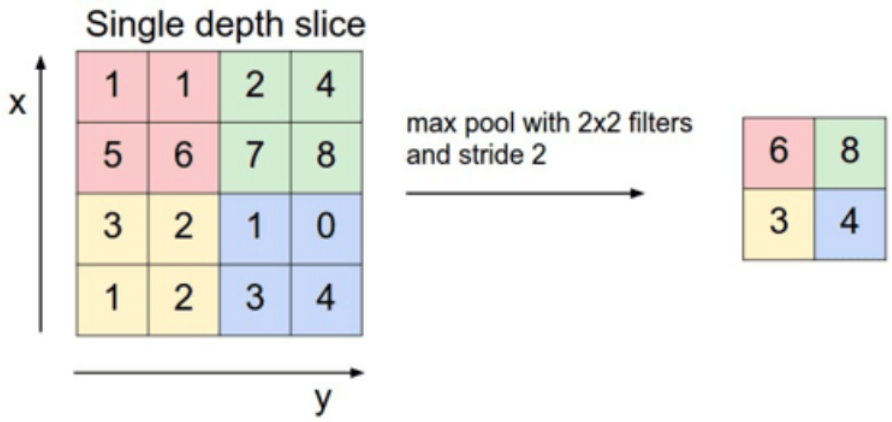
Trong quá trình trượt kernel/filter trên dữ liệu đầu vào, ta sẽ quy định một bước nhảy (stride) với mỗi lần di chuyển. Thông thường ta lựa chọn thường chọn bước nhảy là 1. Nếu kích thước bước nhảy tăng, kernel/filter sẽ có ít ô trùng lắp.

Bởi vì kích thước đầu ra luôn nhỏ hơn đầu vào nên ta cần một phép

xử lí đầu vào để đầu ra không bị co giãn. Đơn giản ta chỉ cần thêm một lề nhỏ vào đầu vào. Một lề (padding) với giá trị 0 sẽ được thêm vào xung quanh đầu vào trước khi thực hiện phép tích chập.

1.1.2.1.3. Lớp pooling Thông thường, sau mỗi tầng tích chập, ta sẽ cho kết quả đi qua một tầng hợp nhất (pooling layer). Mục đích của tầng này là để nhanh chóng giảm số chiều. Việc này giúp giảm thời gian học và hạn chế việc overfitting.

Một phép hợp nhất đơn giản thường được dùng đó là max pooling, phép này lấy giá trị lớn nhất của một vùng để đại diện cho vùng đó. Kích thước của vùng sẽ được xác định trước để giảm kích thước của bản đồ đặc trưng nhanh chóng nhưng vẫn giữ được thông tin cần thiết (hình 1.5).



Hình 1.5: Max pooling kích thước 2×2

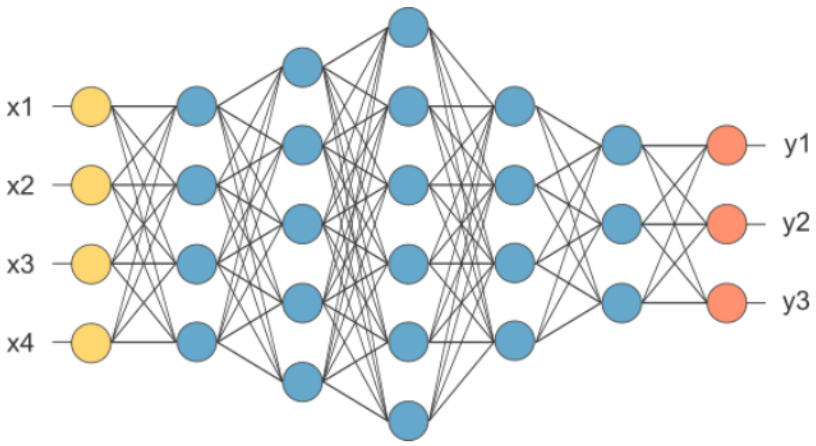
Như vậy, khi thiết kế phần rút trích đặc trưng của mạng CNN, ta cần chú ý đến 4 siêu tham số quan trọng là: Kích thước kernel/filter, Số lượng kernel/filter, Kích thước bước nhảy (stride), Kích thước lề (padding).

1.1.2.2. Phân lớp

Trong phần phân lớp, ta sử dụng một vài tầng với kết nối đầy đủ để xử lí kết quả của phần tích chập. Vì đầu vào của mạng liên kết đầy đủ là 1

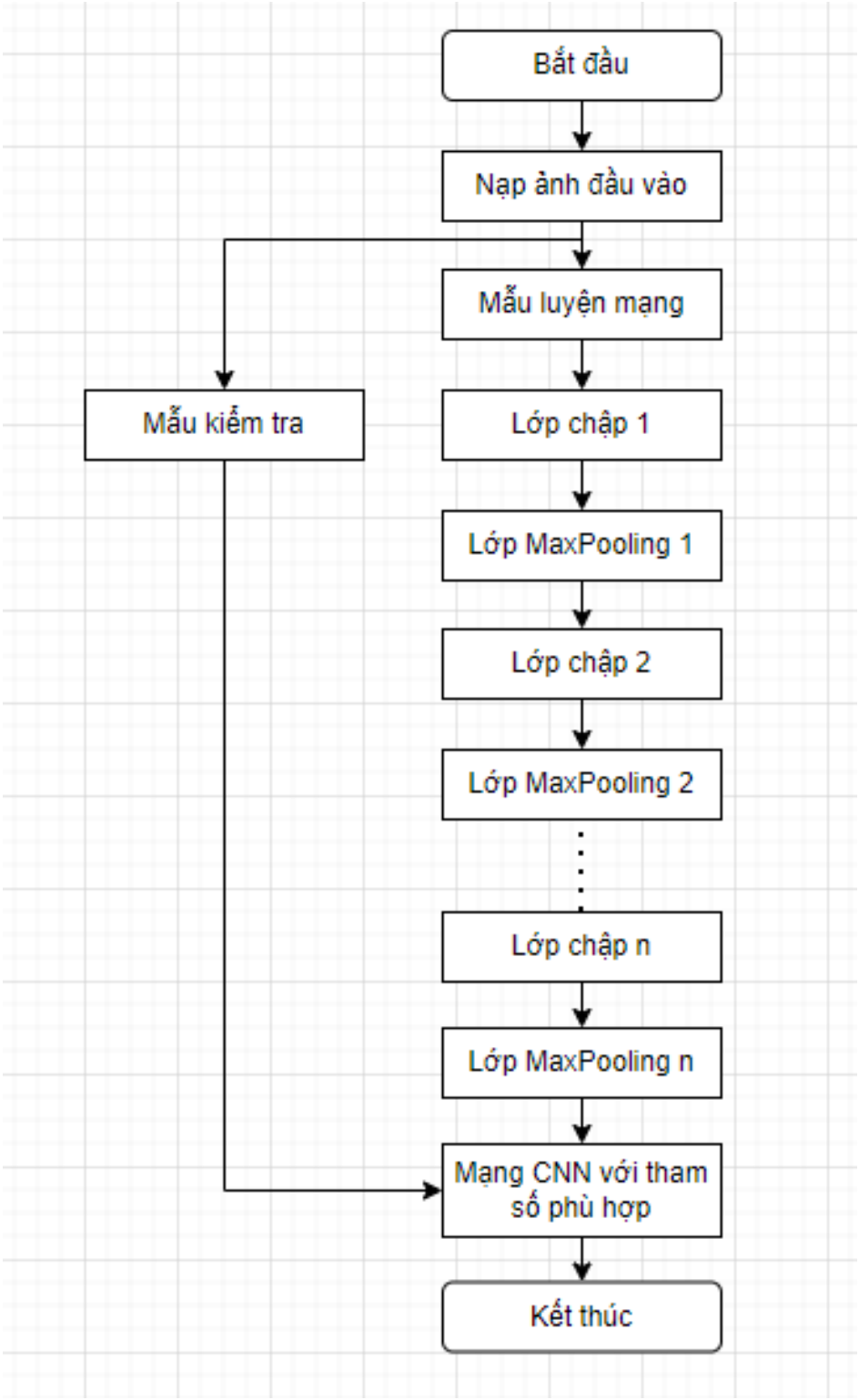
chiều, ta cần làm phẳng đầu vào trước khi phân lớp. Tầng cuối cùng trong mạng CNN là một tầng liên kết đầy đủ, phần này hoạt động tương tự như mạng neural thông thường.

Kết quả thu được cuối cùng cũng sẽ là một véc-tơ với các giá trị xác suất cho việc dự đoán như mạng neural thông thường.



Hình 1.6: Lớp kết nối đầy đủ

1.1.3. Ứng dụng CNN trong phân loại ảnh



Hình 1.7: Max pooling kích thước 2×2

Các bước để thực hiện phân loại hình ảnh dựa trên mạng CNN được mô tả trong Hình 1.7. Đầu tiên, kho dữ liệu ảnh đầu vào được nạp. Ảnh này được chia làm hai phần, một phần dành cho luyện mạng và một phần cho kiểm tra. Trước tiên, ta phải lựa chọn cấu trúc mạng CNN bao gồm số lượng lớp ẩn, các tham số trong mỗi lớp ẩn như kích thước trường tiếp nhận cục bộ, stride, padding. Ảnh luyện mạng sau đó được đưa vào lớp chập 1 để thực hiện tích chập trên ảnh và thực hiện hàm ReLU. Sau đó, kết quả được đưa đến quá trình thực hiện pooling với tham số pooling size phù hợp để giảm kích cỡ ảnh. Ảnh sẽ tiếp tục được đưa thêm qua các lớp tích chập nữa cho đến khi đạt được kết quả mong muốn. Kết quả này được dàn phẳng và đưa vào lớp kết nối đầy đủ. Cuối cùng là quá trình thực hiện các activation function và phân loại ảnh. Quá trình luyện mạng sẽ kết thúc sau khi tổng sai số nhỏ hơn một ngưỡng cho phép hoặc sau một số thế hệ cho trước (điều kiện hội tụ). Kết thúc của quá trình luyện mạng là cấu trúc mạng CNN với các tham số phù hợp. Để kiểm tra, các mẫu ảnh kiểm tra được đưa qua mạng CNN rồi thực hiện đánh giá sai số.

1.1.4. Xây dựng mạng CNN cho phân loại ảnh

Trước tiên, đối với mỗi điểm ảnh trong ảnh đầu vào, ta mã hóa cường độ của điểm ảnh là giá trị của neural tương ứng trong tầng đầu vào.

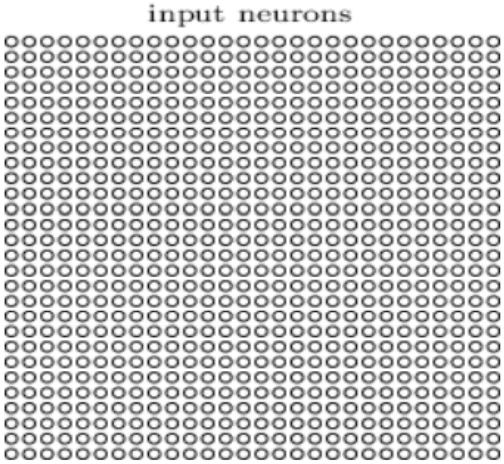
Ví dụ đối với bài toán nhận dạng chữ viết tay từ tập dữ liệu MNIST, mỗi bức ảnh kích thước 28x28 điểm ảnh. Do vậy, mạng có 784 (28x28) neural đầu vào (Hình 1.8). Sau đó ta huấn luyện trọng số (weight) và độ lệch (bias) để đầu ra của mạng như ta mong đợi là xác định chính xác ảnh các chữ số 0, 1, 2...8, 9.

Mạng tích chập sử dụng 3 ý tưởng cơ bản: các trường tiếp nhận cục bộ (local receptive field), trọng số chia sẻ (shared weights) và tổng hợp

(pooling). Chúng ta hãy xem xét lần lượt từng ý tưởng.

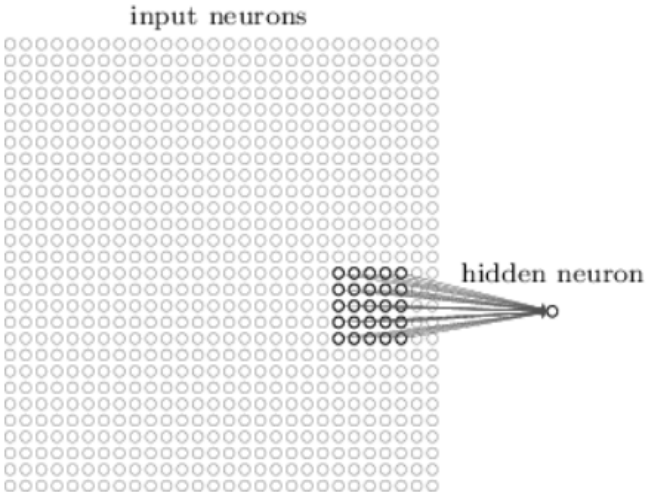
1.1.4.1. Trường tiếp nhận cục bộ (Local receptive fields)

Như thường lệ chúng ta sẽ kết nối các điểm ảnh đầu vào cho các neural ở tầng ẩn. Nhưng chúng ta sẽ không kết nối mỗi điểm ảnh đầu vào cho mỗi neural ẩn. Thay vào đó, chúng ta chỉ kết nối trong phạm vi nhỏ, các vùng cục bộ của bức ảnh.



Hình 1.8: Lớp input gồm 28x28 neural cho nhận dạng chữ từ tập dữ liệu MNIST

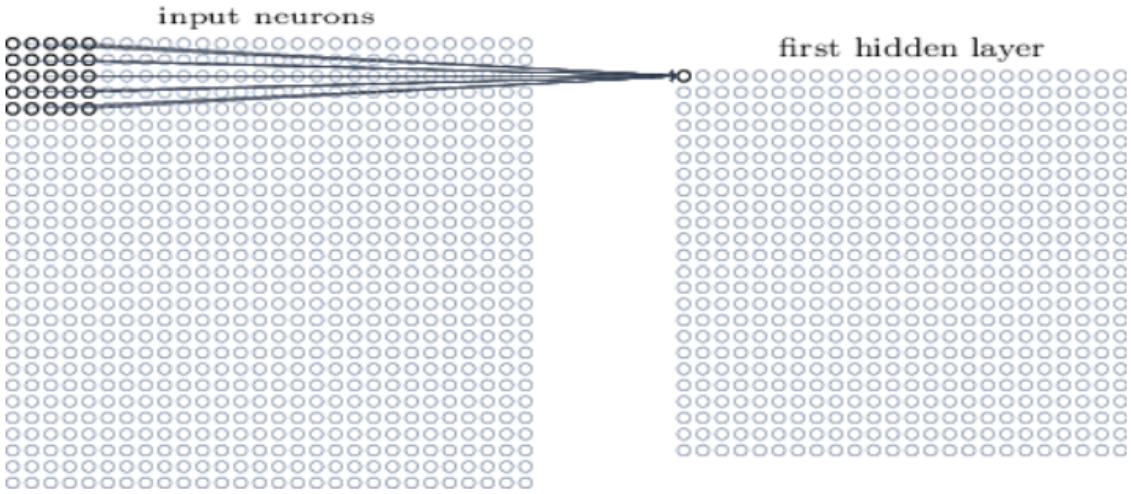
Để được chính xác hơn, mỗi neural trong lớp ẩn đầu tiên sẽ được kết nối với một vùng nhỏ của các neural đầu vào, ví dụ, một vùng 5×5, tương ứng với 25 điểm ảnh đầu vào. Vì vậy, đối với một neural ẩn cụ thể, chúng ta có thể có các kết nối như Hình 1.9 sau:



Hình 1.9: Kết nối vùng 5x5 neural input với neural lớp ẩn

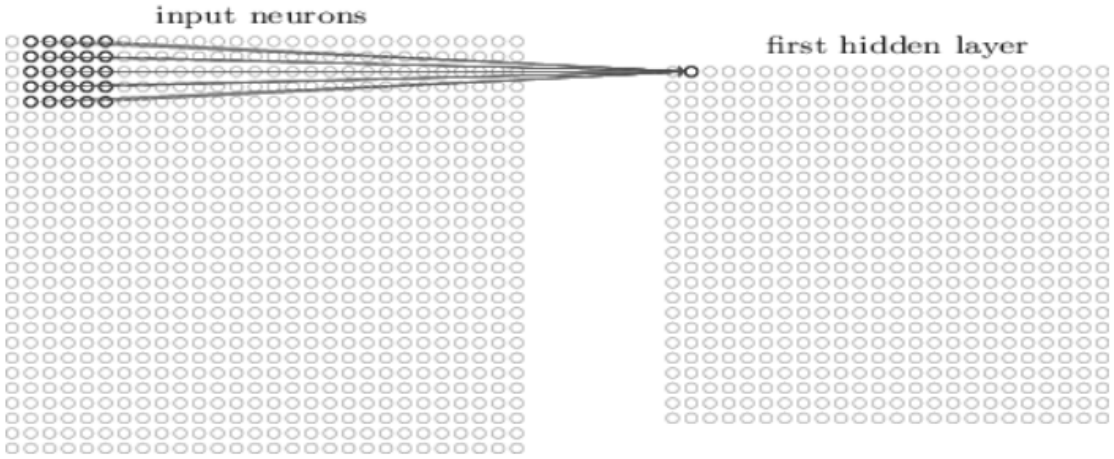
Vùng đó trong bức ảnh đầu vào được gọi là vùng tiếp nhận cục bộ cho neural ẩn. Đó là một cửa sổ nhỏ trên các điểm ảnh đầu vào. Mỗi kết nối sẽ học một trọng số và neural ẩn cũng sẽ học một độ lệch (overall bias). Ta có thể hiểu rằng, neural lớp ẩn cụ thể học để phân tích trường tiếp nhận cục bộ cụ thể của nó.

Sau đó chúng ta trượt trường tiếp nhận cục bộ trên toàn bộ bức ảnh. Đối với mỗi trường tiếp nhận cục bộ, có một neural ẩn khác trong tầng ẩn đầu tiên. Để minh họa điều này một cách cụ thể, chúng ta hãy bắt đầu với một trường tiếp nhận cục bộ ở góc trên bên trái (Hình 1.10):



Hình 1.10: Vị trí bắt đầu của trường tiếp nhận cục bộ

Sau đó, chúng ta trượt trường tiếp nhận cục bộ trên bởi một điểm ảnh bên phải (tức là bằng một neuron), để kết nối với một neural ẩn thứ hai (Hình 1.11):

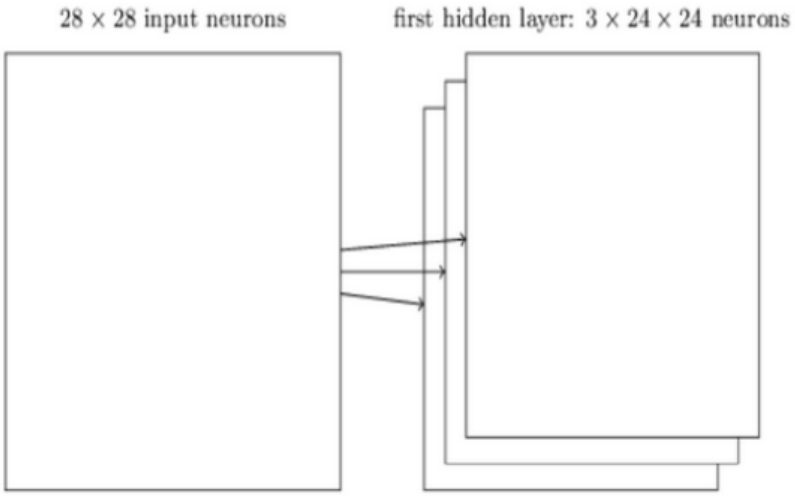


Hình 1.11: Vị trí thứ 2 của trường tiếp nhận cục bộ và neural lớp ẩn

Cứ như vậy, ta sẽ xây dựng các lớp ẩn đầu tiên. Lưu ý rằng nếu chúng ta có một ảnh đầu vào 28×28 và 5×5 trường tiếp nhận cục bộ thì ta sẽ có 24×24 neural trong lớp ẩn. Có được điều này là do chúng ta chỉ có thể di chuyển các trường tiếp nhận cục bộ ngang qua 23 neural (hoặc

xuống dưới 23 neuron), trước khi chạm với phía bên phải (hoặc dưới) của ảnh đầu vào.

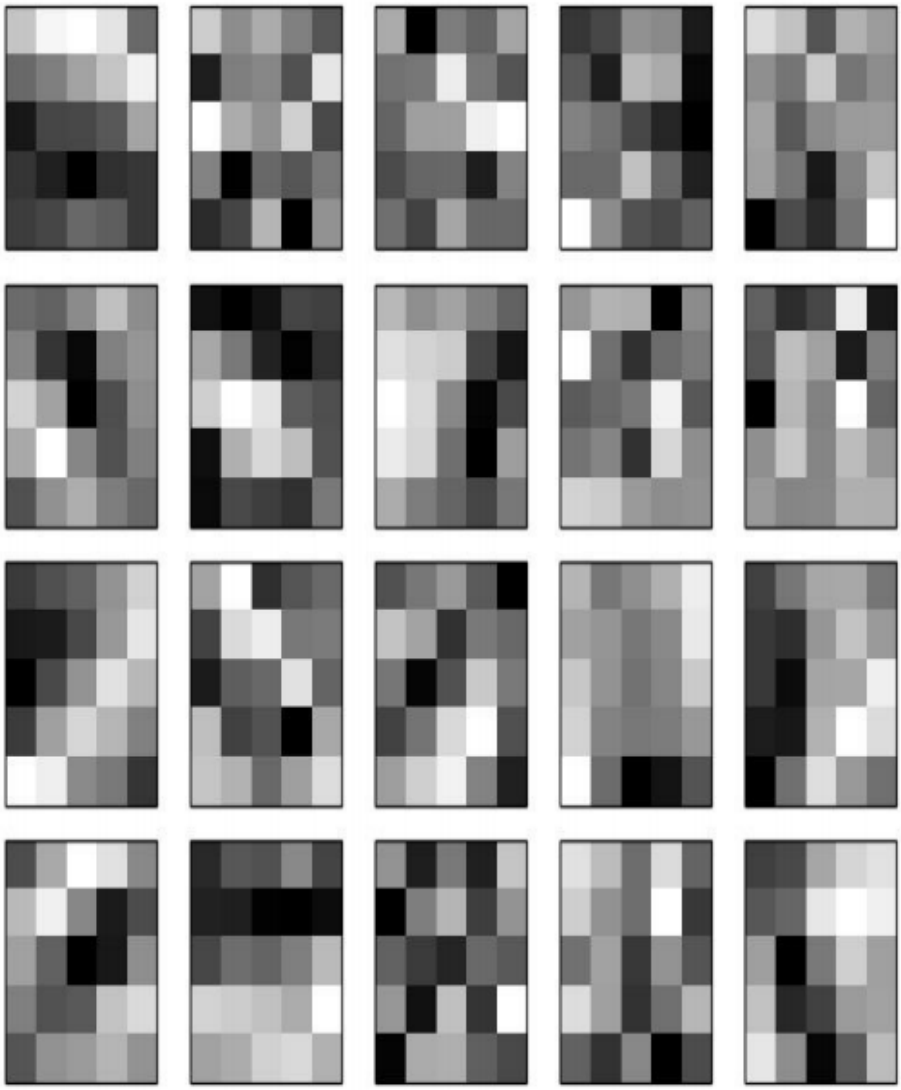
Với bài toán nhận dạng ảnh người ta thường gọi ma trận lớp ẩn đầu vào là feature map, trọng số xác định các đặc trưng là shared weight và độ lệch xác định một feature map là shared bias. Như vậy đơn giản nhất là qua các bước trên chúng ta chỉ có 1 feature map. Tuy nhiên trong nhận dạng ảnh chúng ta cần nhiều hơn một feature map.



Hình 1.12: Trường tiếp nhận cục bộ với ba bản đồ đặc trưng

Trong ví dụ ở Hình 1.12, có 3 bản đồ đặc trưng. Mỗi bản đồ đặc trưng được xác định bởi một tập 5×5 trọng số chia sẻ, và một độ lệch chia sẻ duy nhất. Kết quả là các mạng có thể phát hiện 3 loại đặc trưng khác nhau, với mỗi đặc trưng được phát hiện trên toàn bộ ảnh.

Trong thực tế mạng CNN có thể sử dụng nhiều bản đồ đặc trưng hơn. Một trong những mạng chập đầu tiên là LeNet-5, sử dụng 6 bản đồ đặc trưng, mỗi bản đồ được liên kết đến một trường tiếp nhận cục bộ 5×5, để phát hiện các kí tự MNIST. Vì vậy, các ví dụ minh họa ở trên là thực sự khá gần LeNet-5. Trong một số nghiên cứu gần đây sử dụng lớp tích chập với 20 và 40 bản đồ đặc trưng.



Hình 1.13: Trường tiếp nhận cục bộ với 20 bản đồ đặc trưng

Trên đây (hình 1.13) là 20 ảnh tương ứng với 20 bản đồ đặc trưng khác nhau (hay còn gọi là bộ lọc, hay là nhân). Mỗi bản đồ được thể hiện là một hình khối kích thước 5×5, tương ứng với 5×5 trọng số trong trường tiếp nhận cục bộ. Khối trắng có nghĩa là một trọng số nhỏ hơn, vì vậy các bản đồ đặc trưng đáp ứng ít hơn để tương ứng với điểm ảnh đầu vào. Khối sẫm màu hơn có nghĩa là trọng số lớn hơn, do đó, các bản đồ đặc trưng đáp ứng nhiều hơn với các điểm ảnh đầu vào tương ứng.

Có thể thấy rằng, trường tiếp nhận cục bộ thích hợp cho việc phân tách

dữ liệu ảnh, giúp chọn ra những vùng ảnh có giá trị nhất cho việc đánh giá phân lớp.

1.1.4.2. Trọng số chia sẻ và độ lệch (Shared weights and biases)

Đầu tiên, các trọng số cho mỗi filter (kernel) phải giống nhau. Tất cả các neural trong lớp ẩn đầu sẽ phát hiện chính xác feature tương tự chỉ ở các vị trí khác nhau trong hình ảnh đầu vào. Chúng ta gọi việc map từ input layer sang hidden layer là một feature map. Ta cần tìm ra mối quan hệ giữa số lượng Feature map với số lượng tham số.

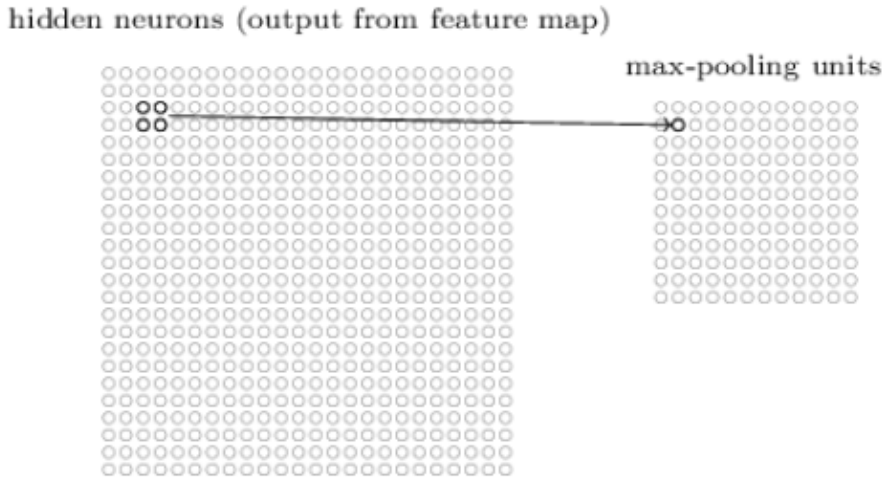
Chúng ta thấy mỗi fearture map cần $25 = 5\times5$ shared weight và 1 shared bias. Như vậy mỗi feature map cần $5\times5+1 = 26$ tham số. Như vậy nếu có 10 feature map thì có $10\times26 = 260$ tham số. Chúng ta xét lại nếu layer đầu tiên có kết nối đầy đủ nghĩa là chúng ta có 28×28=784 neural đầu vào như vậy ta chỉ có 30 neural ẩn. Như vậy ta cần 28x28x30 shared weight và 30 shared bias. Tổng số tham số là 28x28x30+30 tham số lớn hơn nhiều so với CNN. Ví dụ vừa rồi chỉ mô tả để thấy được sự ước lượng số lượng tham số chứ chúng ta không so sánh được trực tiếp vì 2 mô hình khác nhau. Nhưng điều chắc chắn là nếu mô hình có số lượng tham số ít hơn thì nó sẽ chạy nhanh hơn.

1.1.4.3. Lớp chứa hay lớp tổng hợp (Pooling layer)

Ngoài các lớp tích chập vừa mô tả, mạng neural tích chập cũng chứa các lớp pooling. Lớp pooling thường được sử dụng ngay sau lớp tích chập. Những gì các lớp pooling làm là đơn giản hóa các thông tin ở đầu ra từ các lớp tích chập.

Ví dụ, mỗi đơn vị trong lớp pooling có thể thu gọn một vùng 2×2 neural trong lớp trước. Một thủ tục pooling phổ biến là max-pooling.

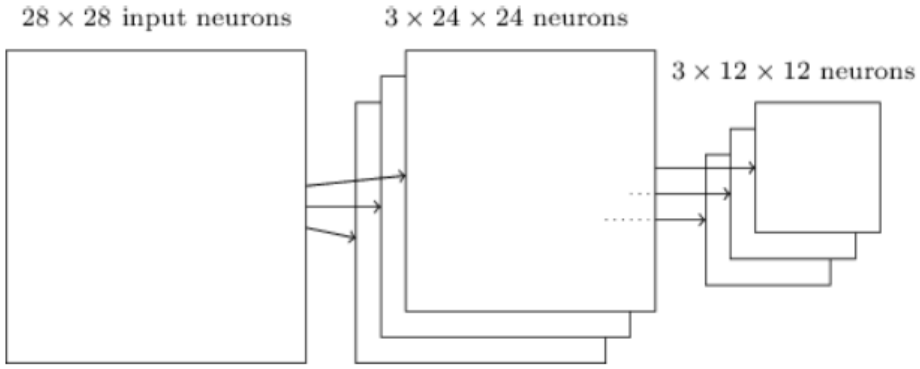
Trong maxpooling, một đơn vị pooling chỉ đơn giản là kết quả đầu ra kích hoạt giá trị lớn nhất trong vùng đầu vào 2×2, như minh họa trong hình 1.14 sau:



Hình 1.14: Trường tiếp nhận cục bộ với 20 bản đồ đặc trưng

Lưu ý rằng bởi vì chúng ta có 24×24 neural đầu ra từ các lớp tích chập, sau khi pooling chúng ta có 12×12 neuron.

Như đã đề cập ở trên, lớp tích chập thường có nhiều hơn một bản đồ đặc trưng. Chúng ta áp dụng max-pooling cho mỗi bản đồ đặc trưng riêng biệt. Vì vậy, nếu có ba bản đồ đặc trưng, các lớp tích chập và max-pooling sẽ kết hợp như hình 1.15 sau:



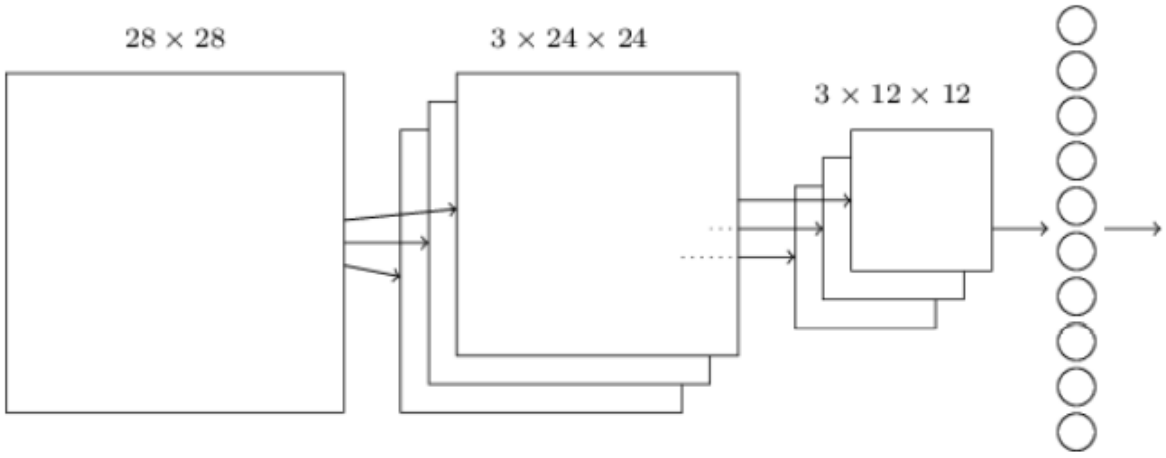
Hình 1.15: Trường tiếp nhận cục bộ với 20 bản đồ đặc trưng

Chúng ta có thể hiểu max-pooling như là một cách cho mạng để hỏi

xem một đặc trưng nhất được tìm thấy ở bất cứ đâu trong một khu vực của ảnh. Sau đó nó bỏ đi những thông tin định vị chính xác. Trực giác là một khi một đặc trưng đã được tìm thấy, vị trí chính xác của nó là không quan trọng như vị trí thô của nó so với các đặc trưng khác. Một lợi ích lớn là có rất nhiều tính năng gộp ít hơn (fewer pooled features), và vì vậy điều này sẽ giúp giảm số lượng các tham số cần thiết trong các lớp sau.

Max-pooling không phải là kỹ thuật duy nhất được sử dụng để pooling. Một phương pháp phổ biến khác được gọi là L2 pooling. Ở đây, thay vì lấy giá trị kích hoạt tối đa (maximum activation) của một vùng 2×2 neuron, chúng ta lấy căn bậc hai của tổng các bình phương của kích hoạt trong vùng 2×2. Trong khi các chi tiết thì khác nhau, nhưng về trực giác thì tương tự như max-pooling: L2 pooling là một cách để cô đọng thông tin từ các lớp tích chập. Trong thực tế, cả hai kỹ thuật đã được sử dụng rộng rãi. Và đôi khi người ta sử dụng các loại pooling khác.

Như vậy, chúng ta có thể đặt tất cả những ý tưởng lại với nhau để tạo thành một mạng tích chập hoàn chỉnh. Nó tương tự như kiến trúc chúng ta phân tích ở trên, nhưng có thêm một lớp 10 neural đầu ra, tương ứng với 10 giá trị có thể cho các số MNIST ('0', '1', '2', v.v...) (hình 1.16):



Hình 1.16: Một kiến trúc mạng CNN cho nhận dạng chữ viết từ dữ liệu MNIST

Mạng bắt đầu với 28 × 28 neural đầu vào, được sử dụng để mã hóa các cường độ điểm ảnh cho ảnh MNIST. Sau đó là một lớp tích chập sử dụng 5 × 5 trường tiếp nhận cục bộ và 3 bản đồ đặc trưng. Kết quả là một lớp 3 × 24 × 24 neural lớp ẩn. Bước tiếp theo là một lớp max-pooling, áp dụng cho 2 × 2 vùng qua 3 bản đồ đặc trưng (feauture maps). Kết quả là một lớp 3 × 12 × 12 neural đặc trưng ở tầng ẩn.

Lớp cuối cùng của các kết nối trong mạng là một lớp đầy đủ kết nối. Lớp này nối mọi neural từ lớp max-pooled tới mọi neural của tầng ra.

1.1.4.4. Cách chọn tham số cho CNN

Hiệu quả hoạt động của mạng CNN phụ thuộc rất nhiều vào việc lựa chọn các tham số sau:

- Số các convolution layer: càng nhiều các convolution layer thì performance càng được cải thiện. Sau khoảng 3 hoặc 4 layer, các tác động được giảm một cách đáng kể.
- Filter size: thường filter theo size 5×5 hoặc 3×3
- Pooling size: thường là 2×2 hoặc 4×4 cho ảnh đầu vào lớn

Trong thực tế, tùy vào ứng dụng cụ thể mà ta chọn các tham số khác nhau. Thông thường ta sẽ thực hiện nhiều lần việc train test để chọn ra được param tốt nhất (Phương pháp thử sai).

1.2. Bài toán chuẩn đoán bệnh lao

1.2.1. Các dữ liệu để chuẩn đoán bệnh lao

Để chuẩn đoán được bệnh lao cần dựa vào rất nhiều dữ liệu [2] như:

- **Lâm sàng**

- **Toàn thân:** Sốt nhẹ về chiều, ra mồ hôi đêm, chán ăn, mệt mỏi, gầy sút cân.
- **Cơ năng:** Ho, khạc đờm, ho ra máu, đau ngực, khó thở.
- **Thực thể:** Nghe phổi có thể có tiếng bệnh lý (ran ẩm, ran nổ,....).

• **Cận lâm sàng**

- Nhuộm soi đờm trực tiếp tìm AFB.
- Xét nghiệm Xpert MTB/RIF (nếu có thể).
- Nuôi cấy tìm vi khuẩn lao.
- X-Quang phổi thường quy.

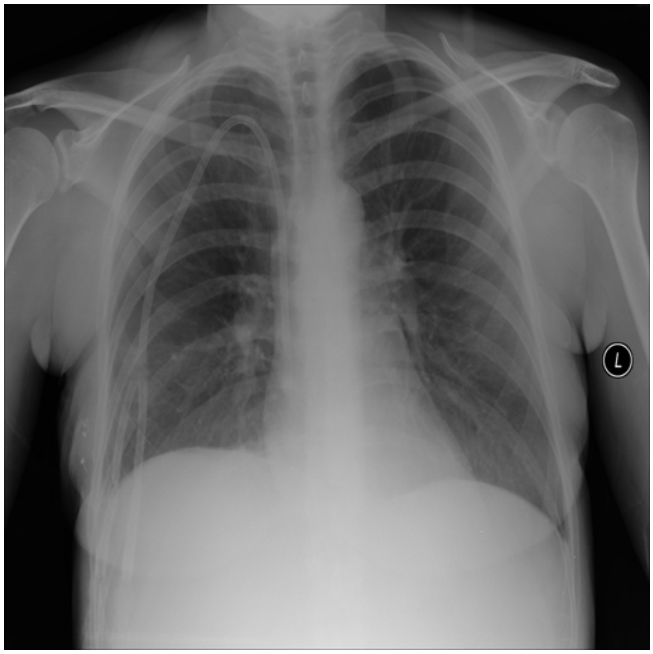
Tuy vậy, do mục tiêu, đối tượng nghiên cứu, phạm vi nghiên cứu, nên luận văn chỉ tập trung nghiên cứu hỗ trợ chuẩn đoán bệnh lao qua các hình ảnh x-quang phổi thường quy dựa vào học máy, học sâu.

1.2.2. Mô tả bài toán

Như đã đề cập ở trên, luận văn chỉ tập trung nghiên cứu hỗ trợ chuẩn đoán bệnh lao qua các hình ảnh x-quang phổi thường quy dựa vào học máy, học sâu nên bài toán này chính là bài toán phân loại ảnh trong Thị giác Máy - Computer Vision.

Bài toán phân loại hình ảnh của luận văn là một trong những nhiệm vụ phổ biến trong Computer Vision. Mục tiêu chính của bài toán này đó chính là phân loại một hình ảnh đầu vào (input) thành một nhãn (label) đầu ra (output).

Với bài toán phân loại ảnh của luận văn, ta có tập dữ liệu ảnh chụp x-quang phổi đã được gán nhãn làm hình ảnh đầu vào (input). Hình ảnh có định dạng PNG như hình 1.17.



Hình 1.17: Ảnh chụp x-quang của người không có lao

Tập dữ liệu trên được một nhóm các nhà nghiên cứu từ Đại học Qatar, Doha, và Đại học Dhaka, Bangladesh cùng với các cộng tác viên của họ từ Malaysia phối hợp với các bác sĩ từ Hamad Medical Corporation và Bangladesh đã tạo ra một cơ sở dữ liệu về hình ảnh X-quang phổi cho người có lao cùng với hình ảnh người không có lao [5]. Dữ liệu trên được cung cấp miễn phí tại https://www.kaggle.com/datasets/tawsifurrahman/tuberculosis-tb-chest-xray-dataset.

Từ tập dữ liệu trên, ta sẽ tiến hành huấn luyện các mô hình học sâu của luận văn. Sau khi hoàn thành quá trình huấn luyện, ta có thể để các mô hình đã được huấn luyện kể trên thực hiện nhiêm vụ phân loại, dự đoán về khả năng ảnh x-quang phổi được đưa vào là của người có lao hay không, đây là nhãn (label) đầu ra (output) mong muốn cho bài toán của luận văn, dựa vào nhãn đầu ra ta sẽ kết luận xem người có ảnh chụp x-quang đó có bị lao hay tổn thương phổi không.

CHƯƠNG 2
Một số mô hình hỗ trợ chuẩn đoán.

Từ mạng CNN cơ bản người ta có thể tạo ra rất nhiều mô hình khác nhau, từ những mạng neural cơ bản 1 đến 2 layer đến 100 layer. Khi thêm nhiều layer hơn thì theo lý thuyết độ chính xác phải cao hơn, nhưng thực tế lại không phải độ chính xác không tăng thậm chí là có lúc lại giảm. Về kernel ta có 3x3, 5x5 hay thậm chí 7x7, vậy thì dùng kernel nào tốt, càng nhỏ liệu có càng chính xác? Vì vậy, chương này ta sẽ tìm hiểu một số mô hình nổi tiếng của CNN, các cài đặt, cấu hình của chúng và hướng áp dụng vào bài toán chuẩn đoán bệnh lao của luận văn.

2.1. VGG-16

2.1.1. VGG16 là gì

VGG là viết tắt của Visual Geometry Group; nó là một kiến trúc CNN sâu tiêu chuẩn với nhiều lớp. Kiến trúc VGG là cơ sở của các mô hình nhận dạng đối tượng mang tính đột phá. Được phát triển như một mạng nơ-ron sâu, VGGNet cũng vượt qua các đường cơ sở về nhiều tác vụ và bộ dữ liệu ngoài ImageNet. Hơn nữa, bây giờ nó vẫn là một trong những kiến trúc nhận dạng hình ảnh phổ biến nhất.

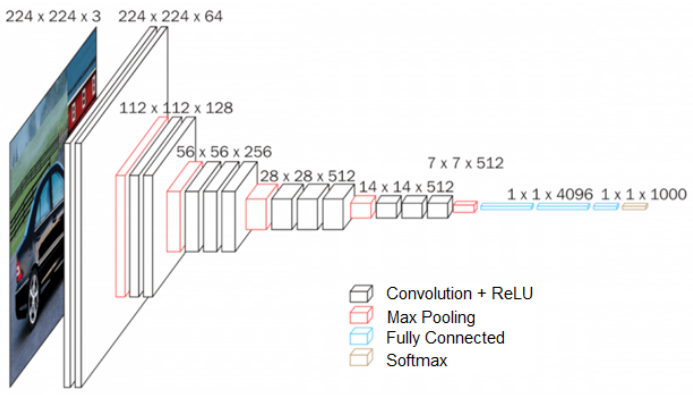
Karen Simonyan và Andrew Zisserman [7] đã đề xuất ý tưởng về mạng VGG vào năm 2013 và gửi mô hình thực tế dựa trên ý tưởng này trong ImageNet Challenge 2014. Họ gọi nó là VGG theo tên bộ phận của Visual Geometry Group tại Đại học Oxford nơi họ làm việc.

Mô hình VGG16, hoặc VGGNet, là một mạng nơ-ron phức hợp hỗ trợ 16 lớp. Mô hình VGG16 đạt được độ chính xác gần như 92,7% trong bài kiểm tra top 5 trong ImageNet (ImageNet là một tập dữ liệu bao gồm

hơn 14 triệu hình ảnh thuộc gần 1000 lớp), nó có thể phân loại hình ảnh thành 1000 loại đối tượng, bao gồm bàn phím, động vật, bút chì, chuột, v.v. Ngoài ra, mô hình có kích thước đầu vào hình ảnh là 224 x 224. Nó thay thế các bộ lọc kích thước hạt nhân lớn bằng một số bộ lọc kích thước hạt nhân 3×3 lần lượt.

2.1.2. Kiến trúc VGG16

Trong tất cả các cấu hình, VGG16 được xác định là mô hình hoạt động tốt nhất trên tập dữ liệu ImageNet. Hãy xem lại kiến trúc thực tế của cấu hình này (Hình 2.1).



Hình 2.1: Kiến trúc VGG16.

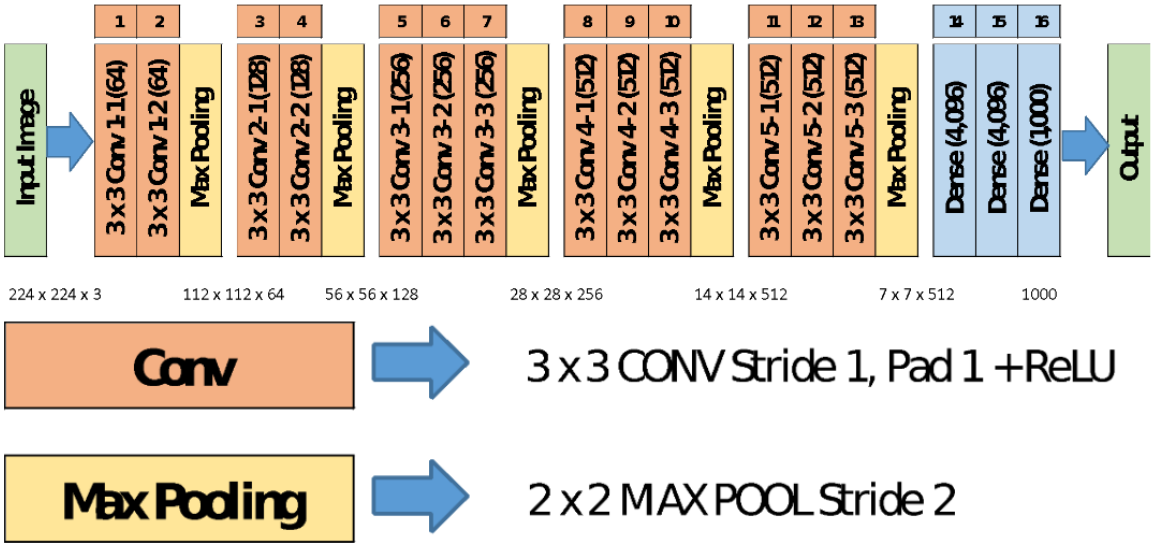
Đầu vào cho bất kỳ cấu hình mạng nào được coi là hình ảnh có kích thước cố định 224 x 224 với ba kênh R, G và B. Quá trình xử lý trước duy nhất được thực hiện là chuẩn hóa các giá trị RGB cho mỗi pixel. Điều này đạt được bằng cách trừ đi giá trị trung bình cho mỗi pixel.

Hình ảnh được chuyển qua ngăn xếp đầu tiên gồm 2 lớp tích chập có kích thước tiếp nhận rất nhỏ là 3 x 3, tiếp theo là kích hoạt ReLU. Mỗi lớp trong số hai lớp này chứa 64 bộ lọc. Stride được cố định ở 1 pixel và padding là 1 pixel. Cấu hình này bảo toàn độ phân giải không gian và kích thước của bản đồ kích hoạt đầu ra giống với kích thước hình ảnh đầu vào.

Các bản đồ kích hoạt sau đó được chuyển qua tổng hợp tối đa không gian trên cửa sổ 2 x 2 pixel, với stride là 2 pixel. Điều này làm giảm một nửa kích thước của các lần kích hoạt. Do đó, kích thước của các kích hoạt ở cuối ngăn xếp đầu tiên là 112 x 112 x 64.

Các kích hoạt sau đó chảy qua ngăn xếp thứ hai tương tự, nhưng với 128 bộ lọc so với 64 bộ lọc trong ngăn xếp thứ nhất. Do đó, kích thước sau ngăn xếp thứ hai trở thành 56 x 56 x 128. Tiếp theo là ngăn xếp thứ ba với ba lớp chập và một lớp tổng hợp tối đa. Số lượng bộ lọc được áp dụng ở đây là 256, làm cho kích thước đầu ra của ngăn xếp là 28 x 28 x 256. Tiếp theo là hai ngăn xếp gồm ba lớp chập, với mỗi ngăn chứa 512 bộ lọc. Đầu ra ở cuối cả hai ngăn xếp này sẽ là 7 x 7 x 512.

Các chồng lớp chập trùng được theo sau bởi ba lớp được kết nối hoàn chỉnh với một lớp làm phẳng ở giữa. Hai lớp đầu tiên có 4.096 tế bào thần kinh mỗi lớp và lớp được kết nối đầy đủ cuối cùng đóng vai trò là lớp đầu ra và có 1.000 tế bào thần kinh tương ứng với 1.000 lớp có thể có cho tập dữ liệu ImageNet. Tiếp theo là lớp đầu ra là lớp kích hoạt Softmax được sử dụng để phân loại (Hình 2.2).



Hình 2.2: Chi tiết kiến trúc VGG16.

2.1.3. Cài đặt VGG16

```
def make_vgg16_model(input_shape):
inputs = keras.Input(shape=input_shape)
x = data_augmentation(inputs)
x = Rescaling(1.0 / 255)(x)
#Block 1
x = Conv2D(filters=start_step, kernel_size=(3, 3), padding='same', activation='relu',
    name='conv1_1')(x)
x = Conv2D(filters=start_step, kernel_size=(3, 3), padding='same', activation='relu',
    name='conv1_2')(x)
x = MaxPooling2D(pool_size=(2,2), strides=(2,2), name='max_pooling2d_1')(x)

#Block 2
x = Conv2D(filters=(start_step*2), kernel_size=(3, 3), padding='same', activation='relu',
    name='conv2_1')(x)
x = Conv2D(filters=(start_step*2), kernel_size=(3, 3), padding='same', activation='relu',
    name='conv2_2')(x)
x = MaxPooling2D(pool_size=(2,2), strides=(2,2), name='max_pooling2d_2')(x)

#Block 3
x = Conv2D(filters=(start_step*4), kernel_size=(3, 3), padding='same', activation='relu',
    name='conv3_1')(x)
x = Conv2D(filters=(start_step*4), kernel_size=(3, 3), padding='same', activation='relu',
    name='conv3_2')(x)
x = Conv2D(filters=(start_step*4), kernel_size=(3, 3), padding='same', activation='relu',
    name='conv3_3')(x)
x = MaxPooling2D(pool_size=(2,2), strides=(2,2), name='max_pooling2d_3')(x)

#Block 4
x = Conv2D(filters=(start_step*8), kernel_size=(3, 3), padding='same', activation='relu',
    name='conv4_1')(x)
x = Conv2D(filters=(start_step*8), kernel_size=(3, 3), padding='same', activation='relu',
    name='conv4_2')(x)
```

```
x = Conv2D(filters=(start_step*8), kernel_size=(3, 3), padding='same', activation='relu',
    name='conv4_3')(x)
x = MaxPooling2D(pool_size=(2,2), strides=(2,2), name='max_pooling2d_4')(x)

#Block 5
x = Conv2D(filters=(start_step*8), kernel_size=(3, 3), padding='same', activation='relu',
    name='conv5_1')(x)
x = Conv2D(filters=(start_step*8), kernel_size=(3, 3), padding='same', activation='relu',
    name='conv5_2')(x)
x = Conv2D(filters=(start_step*8), kernel_size=(3, 3), padding='same', activation='relu',
    name='conv5_3')(x)
x = MaxPooling2D(pool_size=(2,2), strides=(2,2), name='max_pooling2d_5')(x)

#Flatten and FC
x = Flatten(name='flatten')(x)
x = Dense((start_step*8), activation='relu', name='fc_1')(x)
x = Dropout(0.5, name='dropout_1')(x)
x = Dense((start_step*4), activation='relu', name='fc_2')(x)
x = Dropout(0.5, name='dropout_2')(x)
outputs = Dense(1, activation='sigmoid', name='output')(x)
return keras.Model(inputs,outputs)
```

2.1.4. Ưu nhược điểm của VGG16

Ưu điểm:

- VGG đã mang đến một sự cải tiến lớn về độ chính xác và cải thiện cả về tốc độ. Điều này chủ yếu là do cải thiện độ sâu của mô hình.
- Sự gia tăng số lượng các lớp với các hạt nhân nhỏ hơn làm tăng tính phi tuyến tính, điều này luôn luôn là một điều tích cực trong học sâu.
- VGG mang theo nhiều kiến trúc khác nhau được xây dựng dựa trên khái niệm tương tự. Điều này cung cấp cho chúng ta nhiều lựa chọn hơn về kiến trúc nào có thể phù hợp nhất với ứng dụng của chúng ta.

Nhược điểm:

- Một vấn đề của VGG liên quan đến giới hạn tính toán của máy tính cũng khiến cho việc huấn luyện không hiệu quả khi số lượng hidden layers lớn lên. Vấn đề này có tên là vanishing gradient.
- Số lượng bộ lọc mà chúng ta có thể sử dụng tăng gấp đôi trên mỗi bước hoặc qua mỗi ngăn xếp của lớp tích chập. Đây là một nguyên

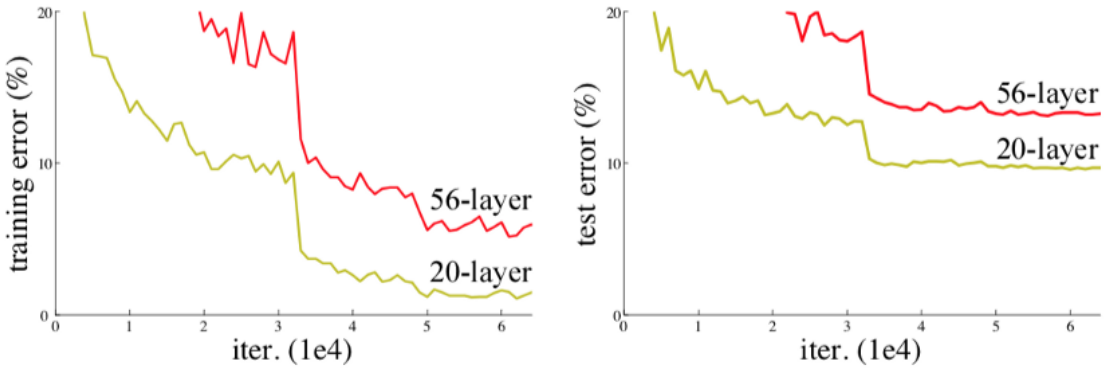
tắc chính được sử dụng để thiết kế kiến trúc của mạng VGG16. Một trong những nhược điểm quan trọng của mạng VGG16 là nó là một mạng khổng lồ, có nghĩa là cần nhiều thời gian hơn để đào tạo các tham số của nó.

2.2. ResNet

2.2.1. Resnet là gì

ResNet (Residual Network) được giới thiệu đến công chúng vào năm 2015 và thậm chí đã giành được vị trí thứ 1 trong cuộc thi ILSVRC 2015 với tỉ lệ lỗi top 5 chỉ 3.57%. Không những thế nó còn đứng vị trí đầu tiên trong cuộc thi ILSVRC and COCO 2015 với ImageNet Detection, ImageNet localization, Coco detection và Coco segmentation.Hiện tại thì có rất nhiều biến thể của kiến trúc ResNet với số lớp khác nhau như ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152,...Với tên là ResNet theo sau là một số chỉ kiến trúc ResNet với số lớp nhất định.

Các kiến trúc mạng trước khi Resnet ra đời như Alexnet, VGG được coi là các mạng nơ ron thuần (plain network). Đối với các mạng nơ ron thuần, Kaiming He[8] đã thí nghiệm và đưa ra kết luận khi tăng số lượng layer của mạng từ 20 lên 56 thì lỗi trên tập huấn luyện và trên tập kiểm tra của mạng 56 layer đều cao hơn so với mạng 20 layer(hình 2.3).

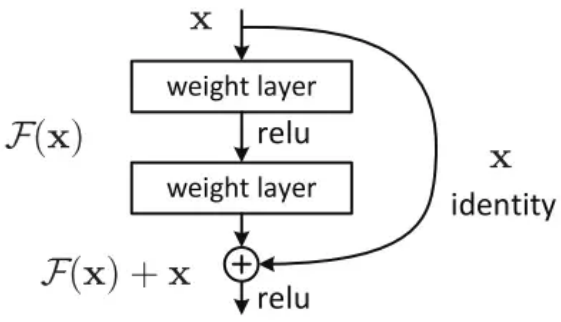


Hình 2.3: Lỗi huấn luyện và lỗi kiểm tra với mạng 20 layer và 56 layer.

ResNet đưa ra là sử dụng kết nối "tắt" đồng nhất để xuyên qua một hay nhiều lớp. Một khối như vậy được gọi là một Residual Block(hình 2.4). Ý tưởng chính của phương pháp này thực ra rất đơn giản, Resnet thực hiện residual mapping để copy thông tin từ các layer nông shallow layer trước đó đến các layer sâu hơn. Chúng ta giả sử output của shallow layer là x. Trong quá trình forward của mạng nó được đưa qua một phép biến đổi tuyến tính $F(x)$. Chúng ta giả sử output của phép biến đổi tuyền tính này là $H(x)$. Một residual (phần dư) giữa deep layer và shallow layer là

$$\mathcal{F}(\mathbf{x}; W_i) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$$

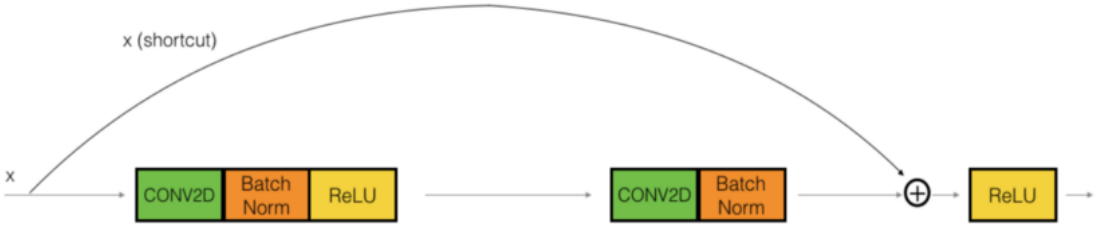
Trong đó, W_i là các tham số của mô hình CNN với phép biến đổi F và nó được tối ưu trong quá trình huấn luyện.



Hình 2.4: Residual block.

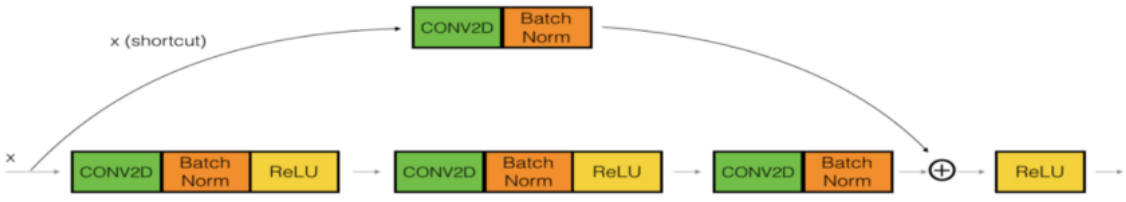
Việc thêm vào các residual block vào trong kiến trúc mạng deep learning có hai cách tuỳ thuộc vào từng trường hợp cụ thể.

- **identity mapping** trong trường hợp này residual mapping đơn giản là việc cộng trực tiếp x vào đầu ra của các stacked block $F(x)$. Đây là một cách sử dụng khá phổ biến trong thiết kế mạng ResNet nếu như input activation có cùng số chiều với output activation.



Hình 2.5: Identity mapping.

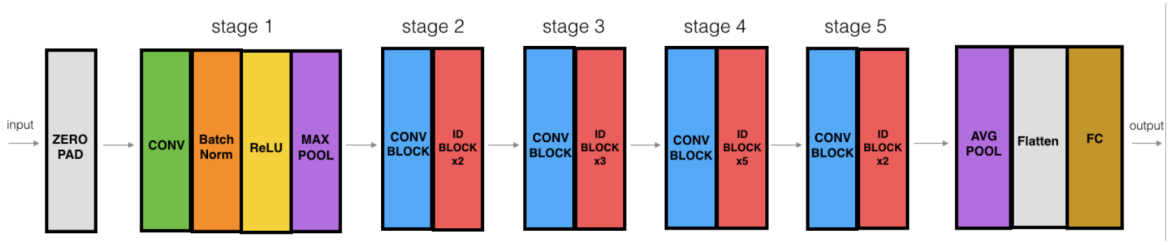
- **Convolutional block** một trường hợp khác là thay vì cộng trực tiếp giá trị của input activation chúng ta sẽ đưa qua một convolution transformation. Trường hợp này có thể được thực hiện trong trường hợp input activation và output activation có số chiều khác nhau. Lúc này đầu ra được xác định như sau $y = F(x; Wi) + \text{Conv}(x)$.



Hình 2.6: Convolutional block.

2.2.2. Kiến trúc của Resnet

Hình 2.7 dưới đây mô tả chi tiết kiến trúc mạng nơ ron ResNet



Hình 2.7: Kiến trúc mạng ResNet.

"ID BLOCK" trong hình trên là viết tắt của từ Identity block và ID BLOCK x3 nghĩa là có 3 khối Identity block chồng lên nhau. Nội dung hình 2.7 như sau:

- Zero-padding : Input với (3,3)
- Stage 1 : Tích chập (Conv1) với 64 filters với shape(7,7), sử dụng stride (2,2). BatchNorm, MaxPooling (3,3).
- Stage 2 : Convolutiontal block sử dụng 3 filter với size 64x64x256, f=3, s=1. Có 2 Identity blocks với filter size 64x64x256, f=3.
- Stage 3 : Convolutional sử dụng 3 filter size 128x128x512, f=3,s=2. Có 3 Identity blocks với filter size 128x128x512, f=3.
- Stage 4 : Convolutional sử dụng 3 filter size 256x256x1024, f=3,s=2. Có 5 Identity blocks với filter size 256x256x1024, f=3.
- Stage 5 :Convolutional sử dụng 3 filter size 512x512x2048, f=3,s=2. Có 2 Identity blocks với filter size 512x512x2048, f=3.
- The 2D Average Pooling : sử dụng với kích thước (2,2).
- The Flatten.
- Fully Connected (Dense) : sử dụng softmax activation.

2.2.3. Cài đặt Resnet

```
def ResNet50(input_shape = (512, 512, 1), classes = 2):

    # Define the input as a tensor with shape input_shape
    X_input = Input(input_shape)


    # Zero-Padding
    X = ZeroPadding2D((3, 3))(X_input)

    # Stage 1
    X = Conv2D(64, (7, 7), strides = (2, 2), name = 'conv1', kernel_initializer =
        glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis = 3, name = 'bn_conv1')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

    # Stage 2
    X = convolutional_block(X, f = 3, filters = [64, 64, 256], stage = 2, block='a', s = 1)
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')


    # Stage 3
    X = convolutional_block(X, f = 3, filters = [128, 128, 512], stage = 3, block='a', s = 2)
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')

    # Stage 4
    X = convolutional_block(X, f = 3, filters = [256, 256, 1024], stage = 4, block='a', s = 2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')

    # Stage 5
    X = convolutional_block(X, f = 3, filters = [512, 512, 2048], stage = 5, block='a', s = 2)
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')

    # AVGPOOL . Use "X = AveragePooling2D(...)(X)"
    X = AveragePooling2D()(X)

    # output layer
    X = Flatten()(X)
    X = Dense(classes, activation='softmax', name='fc' + str(classes), kernel_initializer =
        glorot_uniform(seed=0))(X)


    # Create model
    model = Model(inputs = X_input, outputs = X, name='ResNet50')

    return model
```

2.2.4. Ưu nhược điểm của ResNet

Ưu điểm:

- Kiến trúc ResNet không cần phải kích hoạt tất cả các nơ-ron trong mọi epoch (một epoch được tính là khi chúng ta đưa tất cả dữ liệu trong tập train vào mạng neural network 1 lần). Điều này làm giảm đáng kể thời gian đào tạo và cải thiện độ chính xác. Khi một đặc trưng đã được học, nó sẽ không cố gắng học lại mà tập trung vào việc học các đặc trưng mới hơn. Một cách tiếp cận rất thông minh đã cải thiện đáng kể hiệu suất đào tạo mô hình.
- ResNets giải quyết được khá tốt vấn đề Vanishing Gradient của các mạng CNN thuần.
- Có thể đào tạo dễ dàng các mạng với số lớp rất lớn mà không làm tăng tỷ lệ đào tạo lỗi.

Nhược điểm: [10]

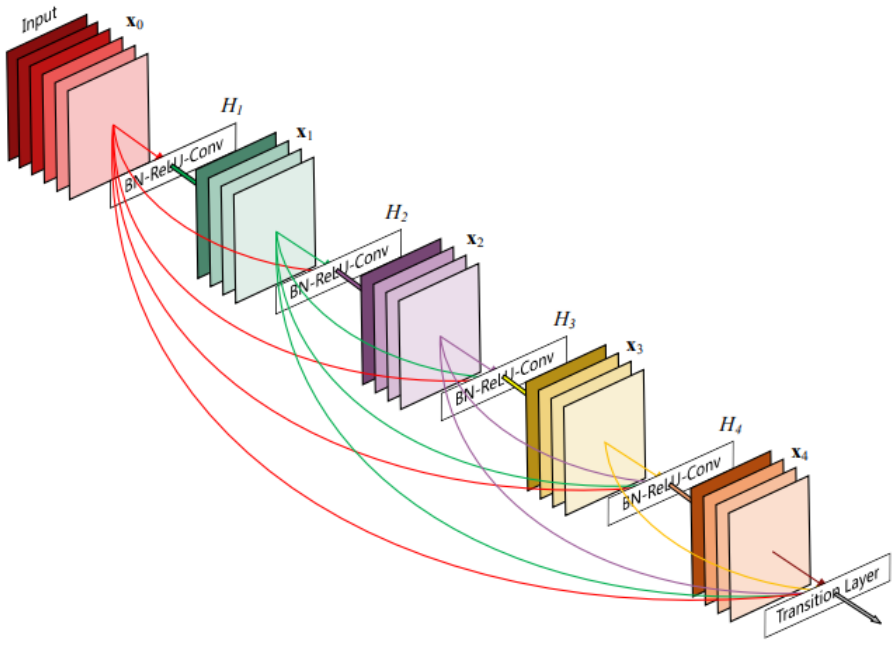
- Đối với mạng sâu hơn, việc phát hiện lỗi trở nên khó khăn.
- Nếu mạng quá nông, việc đào tạo có thể rất kém hiệu quả.

2.3. DenseNet

2.3.1. DenseNet là gì

DenseNet - Dense Convolutional Network (Mạng Tích chập Kết nối Dày đặc) - là một trong những biến thể mở rộng của Resnet và là một kiến trúc mạng,trong đó mỗi lớp được kết nối trực tiếp với mỗi lớp khác nhau theo kiểu chuyển tiếp (trong mỗi khối dense block). Đối với mỗi lớp, các bản đồ đặc trưng (feature map) của tất cả các lớp ở phần trước được coi là các đầu vào riêng biệt và ở đó các bản đồ tính năng lại tiếp tục làm đầu

vào cho tất cả các lớp tiếp theo. Cấu trúc mạng này mang lại độ chính xác "state of the art" trên CIFAR. Trên bộ dữ liệu ILSVRC 2012 (ImageNet), DensetNet đạt được độ chính xác tương tự như ResNet, nhưng sử dụng ít hơn một nửa số lượng tham số. DenseNet có cấu trúc gồm các dense block và các transaction layers. Trong kiến trúc CNN truyền thống, nếu có L Layer thì có L connection, trong densenet sẽ có L(L+1)/2 connection[4].



Hình 2.8: Kiến trúc DenseNet.

Xét một hình ảnh x_0 duy nhất được truyền qua một mạng tích chập. Mạng bao gồm L lớp, mỗi lớp thực hiện một phép biến đổi phi tuyến tính $H_l(\cdot)$, trong đó l là chỉ mục các lớp. $H_l(\cdot)$ có thể là một hàm tổng hợp của các hoạt động như Batch Normalization (BN), lớp ReLU, lớp Pooling, hoặc lớp Convolution (Conv). DenseNet biểu thị đầu ra của lớp thứ l như là x_l.

Kết nối dày đặc - Dense connectivity Để cải thiện hơn nữa luồng thông tin giữa các lớp, DenseNet đề xuất một mô hình kết nối mà từ bất kỳ lớp nào cũng có thể kết nối đến tất cả các lớp tiếp theo. Hình 2.8 minh

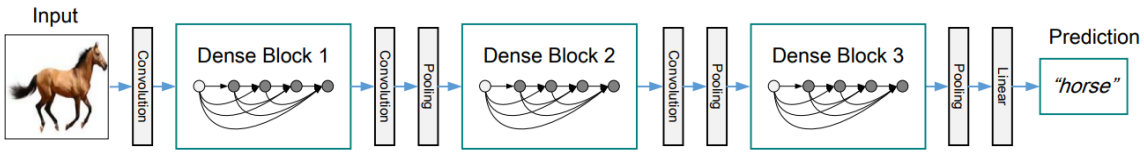
họa bố trí kiến trúc của DenseNet. Dễ dàng nhìn thấy, lớp thứ l nhận được các bản đồ đặc trưng của tất cả các lớp trước đó, $x_0, x_1, x_2, ..., x_{l-1}$), làm đầu vào:

$$x_l = H_l([x_0, x_1, ..., x_{l-1}]) \tag{2.3.1}$$

với $[x_0, x_1, ..., xl - 1]$ đề cập đến việc nối các trích chọn đặc trưng được tạo thành trong các lớp $0, 1, ..., l - 1$. Do khả năng kết nối dày đặc của nó, Gao Huang[9] gọi kiến trúc mạng này là Mạng kết nối dày đặc (DenseNet). Để dễ thực hiện, DenseNet nối nhiều đầu vào của $H_l(\cdot)$ trong phương trình 2.3.1 thành một tensor duy nhất.

Hàm tổng hợp - Composite function Ta định nghĩa $H_l(\cdot)$ là một hàm tổng hợp của ba hoạt động liên tiếp: Batch Normalization (BN), tiếp theo là một hàm tinh chỉnh các đơn vị tuyến tính (ReLU) và một tích chập 3 $\times$ 3 (Conv).

Tầng hợp nhất - Pooling layers Thao tác nối được sử dụng trong Phương trình 2.3.1 không khả thi khi kích thước của bản đồ đối tượng thay đổi. Tuy nhiên, một phần thiết yếu của mạng tích chập là các lớp lấy mẫu xuống làm thay đổi kích thước của bản đồ đối tượng. Để tạo điều kiện thuận lợi cho việc giảm tần số lấy mẫu trong kiến trúc, DenseNet chia mạng thành nhiều khối dày đặc được kết nối với nhau(hình 2.9). DensetNet đề cập đến các lớp giữa các khối là các lớp chuyển tiếp, các lớp này thực hiện tích chập và tổng hợp. Các lớp chuyển tiếp được sử dụng trong các cấu trúc DenseNet bao gồm lớp chuẩn hóa hàng loạt và lớp tích chập 1 $\times$ 1, tiếp theo là lớp gộp trung bình 2 $\times$ 2.



Hình 2.9: Một DenseNet sâu với ba khối dày đặc.

Tỉ lệ phát triển - Growth rate Nếu mỗi hàm H_l tạo ra k bản đồ đặc trưng, thì lớp thứ l có $k_0 + kl - 1$ bản đồ đặc trưng đầu vào, trong đó k_0 là số kênh trong lớp đầu vào. Một điểm khác biệt quan trọng giữa DenseNet và các kiến trúc mạng trước đó là DenseNet có thể có các lớp rất hẹp, ví dụ: $k = 12$. Ta gọi siêu tham số k là tỉ lệ phát triển của mạng. Tỉ lệ phát triển quy định lượng thông tin mới mà mỗi lớp đóng góp vào trạng thái toàn cục. Trạng thái toàn cục, sau khi được lưu trữ, có thể được truy cập từ mọi nơi trong mạng mà không cần phải sao chép nó từ lớp này sang lớp khác - không giống như trong các kiến trúc mạng truyền thống.

Các lớp nút cổ chai - Bottleneck layers Mặc dù mỗi lớp chỉ tạo ra k bản đồ đặc trưng đầu ra, nhưng nó thường có nhiều đầu vào hơn. Người ta lưu ý rằng tích chập 1×1 có thể được đưa vào làm lớp nút cổ chai trước mỗi tích chập 3×3 để giảm số lượng bản đồ đặc trưng đầu vào và từ đó để cải thiện hiệu quả tính toán. Thiết kế này đặc biệt hiệu quả đối với DenseNet.

Độ nén - Compression Để cải thiện hơn nữa độ nhỏ gọn của mô hình, chúng ta có thể giảm số lượng bản đồ đặc trưng ở các lớp chuyển tiếp. Nếu một khối dày đặc chứa m bản đồ đặc trưng, để lớp chuyển tiếp sau tạo ra θm bản đồ đặc trưng đầu ra, trong đó $0 < \theta \leq 1$ được gọi là hệ số nén. Khi $\theta = 1$, số lượng bản đồ đối tượng trên các lớp chuyển tiếp không thay đổi.

2.3.2. Kiến trúc của DenseNet

Từ hình 2.9, có thể nhận thấy rằng DenseNets được chia thành nhiều khối dày đặc DenseBlock. Các kiến trúc khác nhau của DenseNets đã được tóm tắt trong bài báo "Densely Connected Convolutional Networks"[9].

Mỗi kiến trúc bao gồm bốn DenseBlock với số lượng lớp khác nhau. Ví dụ, DenseNet-121 có [6,12,24,16] lớp trong bốn khối dày đặc trong khi DenseNet-169 có [6, 12, 32, 32] lớp. Chúng ta có thể thấy rằng phần đầu tiên của kiến trúc DenseNet bao gồm Lớp chuyển đổi 2 bước 7x7, tiếp theo là lớp MaxPooling 3x3 bước-2. Và khối dày đặc thứ tư được theo sau bởi một Lớp phân loại chấp nhận các bản đồ đặc trưng của tất cả các lớp của mạng để thực hiện phân loại.

Ngoài ra, các phép toán tích chập bên trong mỗi kiến trúc là các lớp Cổ chai. Điều này có nghĩa là Conv 1x1 làm giảm số lượng kênh trong đầu vào và Conv 3x3 thực hiện hoạt động tích chập trên phiên bản đã biến đổi của đầu vào với số lượng kênh giảm hơn so với đầu vào.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Hình 2.10: Các kiến trúc DenseNet.

2.3.3. Cài đặt DenseNet

```
def DenseNet(input_shape=(512, 512, 1),nb_dense_block=4, growth_rate=32, nb_filter=64,
    reduction=0.0, dropout_rate=0.0, weight_decay=1e-4, classes=1000, weights_path=None):

   eps = 1.1e-5

   compression = 1.0 - reduction

   global concat_axis
   concat_axis = 3
   img_input = Input(shape=input_shape, name='data')

   # From architecture for ImageNet (Table 1 in the paper)
   nb_filter = 64
   nb_layers = [6,12,24,16] # For DenseNet-121

   # Initial convolution
   x = ZeroPadding2D((3, 3), name='conv1_zeropadding')(img_input)
   x = Convolution2D(nb_filter, 7, 7, subsample=(2, 2), name='conv1', bias=False)(x)
   x = batch_normalization(epsilon=eps, axis=concat_axis, name='conv1_bn')(x)
   x = Activation('relu', name='relu1')(x)
   x = ZeroPadding2D((1, 1), name='pool1_zeropadding')(x)
   x = MaxPooling2D((3, 3), strides=(2, 2), name='pool1')(x)

   # Add dense blocks
   for block_idx in range(nb_dense_block - 1):
   stage = block_idx+2
   x, nb_filter = dense_block(x, stage, nb_layers[block_idx], nb_filter, growth_rate,
       dropout_rate=dropout_rate, weight_decay=weight_decay)

   # Add transition_block
   x = transition_block(x, stage, nb_filter, compression=compression,
       dropout_rate=dropout_rate, weight_decay=weight_decay)
   nb_filter = int(nb_filter * compression)

   final_stage = stage + 1
   x, nb_filter = dense_block(x, final_stage, nb_layers[-1], nb_filter, growth_rate,
       dropout_rate=dropout_rate, weight_decay=weight_decay)

   x = batch_normalization(epsilon=eps, axis=concat_axis,
       name='conv'+str(final_stage)+'_blk_bn')(x)
   x = Activation('relu', name='relu'+str(final_stage)+'_blk')(x)
   x = GlobalAveragePooling2D(name='pool'+str(final_stage))(x)

   x = Dense(classes, name='fc6')(x)
   x = Activation('softmax', name='prob')(x)

   model = Model(img_input, x, name='densenet')

   if weights_path is not None:
   model.load_weights(weights_path)

   return model


def conv_block(x, stage, branch, nb_filter, dropout_rate=None, weight_decay=1e-4):
   eps = 1.1e-5
   conv_name_base = 'conv' + str(stage) + '_' + str(branch)
   relu_name_base = 'relu' + str(stage) + '_' + str(branch)
```

```
    # 1x1 Convolution (Bottleneck layer)
    inter_channel = nb_filter * 4
    x = batch_normalization(epsilon=eps, axis=concat_axis, name=conv_name_base+'_x1_bn')(x)
    x = Activation('relu', name=relu_name_base+'_x1')(x)
    x = Convolution2D(inter_channel, 1, 1, name=conv_name_base+'_x1', bias=False)(x)

    if dropout_rate:
    x = Dropout(dropout_rate)(x)

    # 3x3 Convolution
    x = batch_normalization(epsilon=eps, axis=concat_axis, name=conv_name_base+'_x2_bn')(x)
    x = Activation('relu', name=relu_name_base+'_x2')(x)
    x = ZeroPadding2D((1, 1), name=conv_name_base+'_x2_zeropadding')(x)
    x = Convolution2D(nb_filter, 3, 3, name=conv_name_base+'_x2', bias=False)(x)

    if dropout_rate:
    x = Dropout(dropout_rate)(x)

    return x


def transition_block(x, stage, nb_filter, compression=1.0, dropout_rate=None,
     weight_decay=1E-4):
    eps = 1.1e-5
    conv_name_base = 'conv' + str(stage) + '_blk'
    relu_name_base = 'relu' + str(stage) + '_blk'
    pool_name_base = 'pool' + str(stage)

    x = batch_normalization(epsilon=eps, axis=concat_axis, name=conv_name_base+'_bn')(x)
    x = Activation('relu', name=relu_name_base)(x)
    x = Convolution2D(int(nb_filter * compression), 1, 1, name=conv_name_base, bias=False)(x)

    if dropout_rate:
    x = Dropout(dropout_rate)(x)

    x = AveragePooling2D((2, 2), strides=(2, 2), name=pool_name_base)(x)

    return x


def dense_block(x, stage, nb_layers, nb_filter, growth_rate, dropout_rate=None,
     weight_decay=1e-4, grow_nb_filters=True):
    concat_feat = x

    for i in range(nb_layers):
    branch = i+1
    x = conv_block(concat_feat, stage, branch, growth_rate, dropout_rate, weight_decay)
    concat_feat = merging([concat_feat, x], mode='concat', concat_axis=concat_axis,
         name='concat_'+str(stage)+'_'+str(branch))

    if grow_nb_filters:
    nb_filter += growth_rate

    return concat_feat, nb_filter
```

2.3.4. Ưu nhược điểm của DenseNet

Ưu điểm:

- Giải quyết khá tốt vấn đề vanishing-gradient của các mạng CNN thuần.
- Cải thiện sự truyền tải đặc trưng giữa các lớp cả về phía trước cũng như phía sau.
- Giảm đáng kể số lượng tham số.
- Khuyến khích sử dụng lại các đặc trưng.

Nhược điểm:

- Kết nối quá mức không chỉ làm giảm hiệu suất tính toán và hiệu quả tham số của mạng mà còn làm cho các mạng dễ bị overfitting. [11]

CHƯƠNG 3

Chương trình thử nghiệm.

3.1. Phân tích yêu cầu bài toán

Như đã nói tại Chương 1, bài toán phân loại hình ảnh của luận văn là một trong những nhiệm vụ phổ biến trong Computer Vision. Mục tiêu chính của bài toán này đó chính là phân loại một hình ảnh đầu vào (input) thành một nhãn (label) đầu ra (output).

Với bài toán phân loại ảnh của luận văn, ta có tập dữ liệu ảnh chụp x-quang phổi đã được gán nhãn làm hình ảnh đầu vào (input). Hình ảnh có định dạng PNG như hình 1.17.

Tập dữ liệu trên được một nhóm các nhà nghiên cứu từ Đại học Qatar, Doha, và Đại học Dhaka, Bangladesh cùng với các cộng tác viên của họ từ Malaysia phối hợp với các bác sĩ từ Hamad Medical Corporation và Bangladesh đã tạo ra một cơ sở dữ liệu về hình ảnh X-quang phổi cho người có lao cùng với hình ảnh người không có lao [5]. Dữ liệu trên được cung cấp miễn phí tại https://www.kaggle.com/datasets/tawsifurrahman/tuberculosis-tb-chest-xray-dataset. Thông tin cụ thể về tập dữ liệu như sau:

- Tổng số 4200 ảnh đã được phân loại thành hai thư mục Normal (bình thường) và Tuberculosis (có lao).
- Số ảnh người mắc lao: 700 ảnh.
- Số ảnh người không mắc lao: 3500 ảnh.
- Kích thước ảnh: 512 x 512 (pixel)
- Định dạng ảnh: PNG.
- Tập tin Normal.metadata.xlsx tổng hợp thông tin về các file ảnh của

người không mắc lao.

- Tập tin Tuberculosis.metadata.xlsx tổng hợp thông tin về các file ảnh của người mắc lao.

Mục tiêu mong muốn của bài toán là việc có thể dự đoán về khả năng ảnh x-quang phổi được đưa vào là của người có lao hay không, đây là nhãn (label) đầu ra (output) mong muốn, dựa vào nhãn đầu ra này ta sẽ kết luận xem người có ảnh chụp x-quang đó có bị lao hay tổn thương phổi không. Để thực hiện được mục tiêu trên, ta cần thực hiện hai pha:

- **Học:** Lặp lại nhiều lần việc lần lượt đưa tất cả ảnh đầu vào để huấn luyện mô hình, trích chọn ra các đặc trưng của ảnh rồi lưu lại. Quá trình này nên dừng lại khi độ chính xác đạt mức tối thiểu mà ta đặt ra hoặc độ chính xác tăng lên không đáng kể sau một số lượt huấn luyện nhất định.
- **Phân loại:** Dùng các đặc trưng mà mô hình thu được tại pha Học, đưa ảnh chụp x-quang đầu vào để tiến hành so sánh với các đặc trưng đó, kết thúc quá trình bằng một giá trị dự đoán khả năng mắc lao của người có ảnh x-quang được đưa vào.

Để đảm bảo tính chính xác, khách quan cho khả năng dự đoán của mô hình đã đào tạo, ta cần tách dữ liệu cho hai pha này phải khác nhau. Ta có thể tách riêng 100 ảnh trong tập dữ liệu bao gồm cả ảnh x-quang của người có lao và không có lao là một tập dữ liệu mới gọi là dữ liệu xác thực để sử dụng cho pha Phân loại. Còn lại 4100 ảnh x-quang của tập dữ liệu gốc ta gọi là dữ liệu huấn luyện và sử dụng cho pha Học.

3.2. Phân tích lựa chọn công cụ

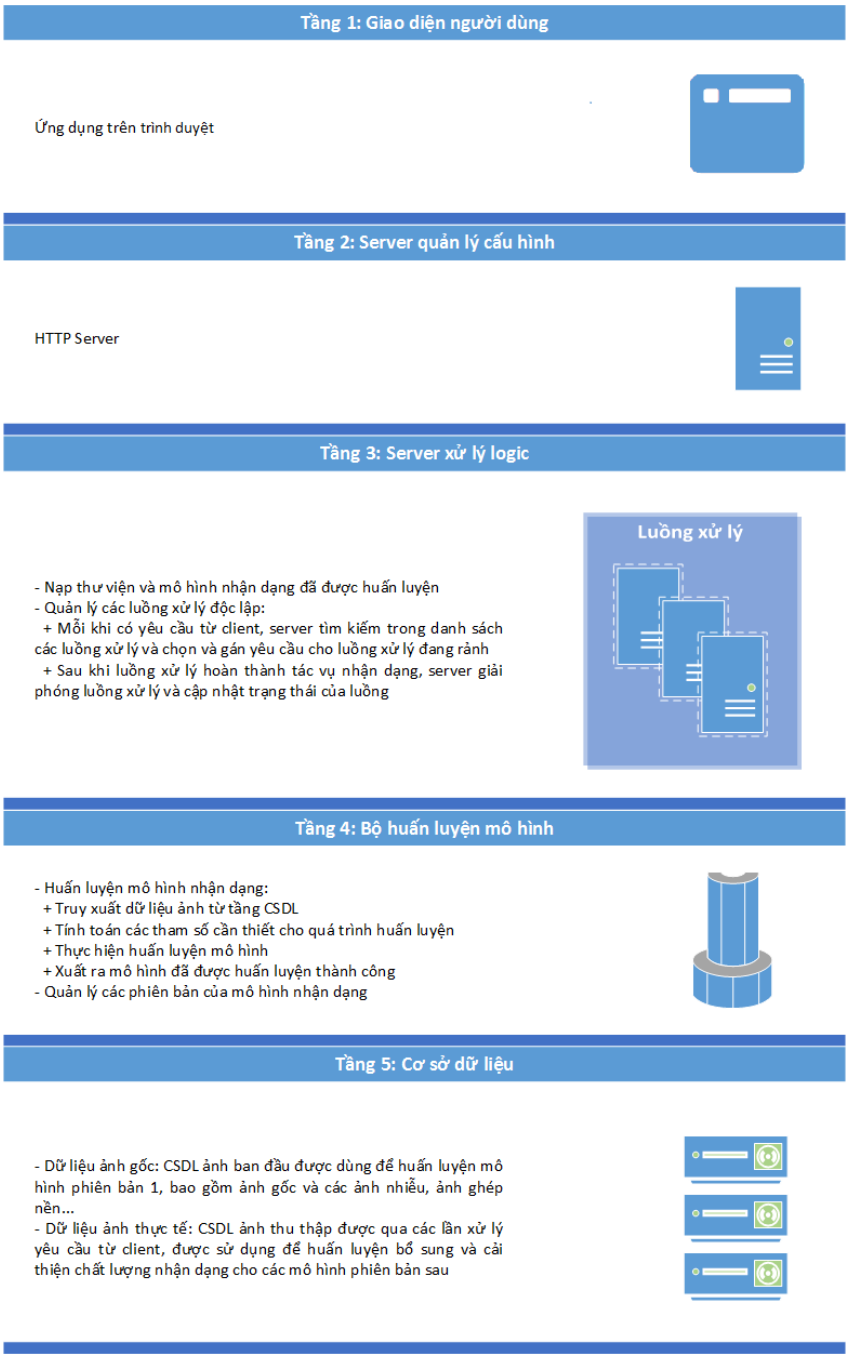
3.2.1. Lựa chọn mô hình đã hệ thống hóa

3.2.2. Phân tích mô hình hệ thống

Hệ thống chương trình thử nghiệm Phần mềm chuẩn đoán bệnh lao được thiết kế theo kiến trúc Client/Server năm tầng (hình 3.1), trong đó:

- **Tầng thứ nhất** là tầng giao diện người phân loại, cụ thể là ứng dụng client trên trình duyệt, quản lý tương tác người phân loại với ứng dụng như chọn ảnh gửi lên server... và hiển thị kết quả phân loại do server gửi về.
- **Tầng thứ hai** là tầng server quản lý cấu hình hệ thống, ví dụ cấu hình giao thức gửi/nhận dữ liệu với client, cụ thể giao thức được sử dụng trong hệ thống là giao thức HTTP.
- **Tầng thứ ba** là tầng server thực hiện logic xử lý các yêu cầu từ client, như quản lý và phân phối các luồng xử lý độc lập, đảm bảo hiệu năng và chất lượng tính toán phân loại cho nhiều client trong cùng một thời điểm.
- **Tầng thứ tư** là tầng đảm nhiệm xây dựng, tinh chỉnh và quản lý các phiên bản mô hình phân loại cho hệ thống, với bộ ảnh huấn luyện được lấy từ tầng quản lý dữ liệu bên dưới.
- **Tầng cuối cùng** là tầng quản lý dữ liệu, bao gồm CSDL ảnh phục vụ cho việc huấn luyện mô hình, CSDL ảnh đã xử lý từ các client nhằm mục đích bổ sung sự đa dạng của CSDL ảnh và cải thiện độ chính xác của mô hình phân loại. Các bộ ảnh trên được lưu tách biệt để thuận tiện cho việc quản lý và đánh giá độ chính xác của các phiên bản mô

hình huấn luyện cũng như mức độ ảnh hưởng của bộ ảnh huấn luyện lên chất lượng mô hình.



Hình 3.1: Kiến trúc Client/Server n tầng.

Luồng hoạt động chính của hệ thống được thể hiện trong hình 3.2, trong đó các bước thực hiện của server và client từ lúc khởi động ban đầu tới lúc kết thúc như sau:

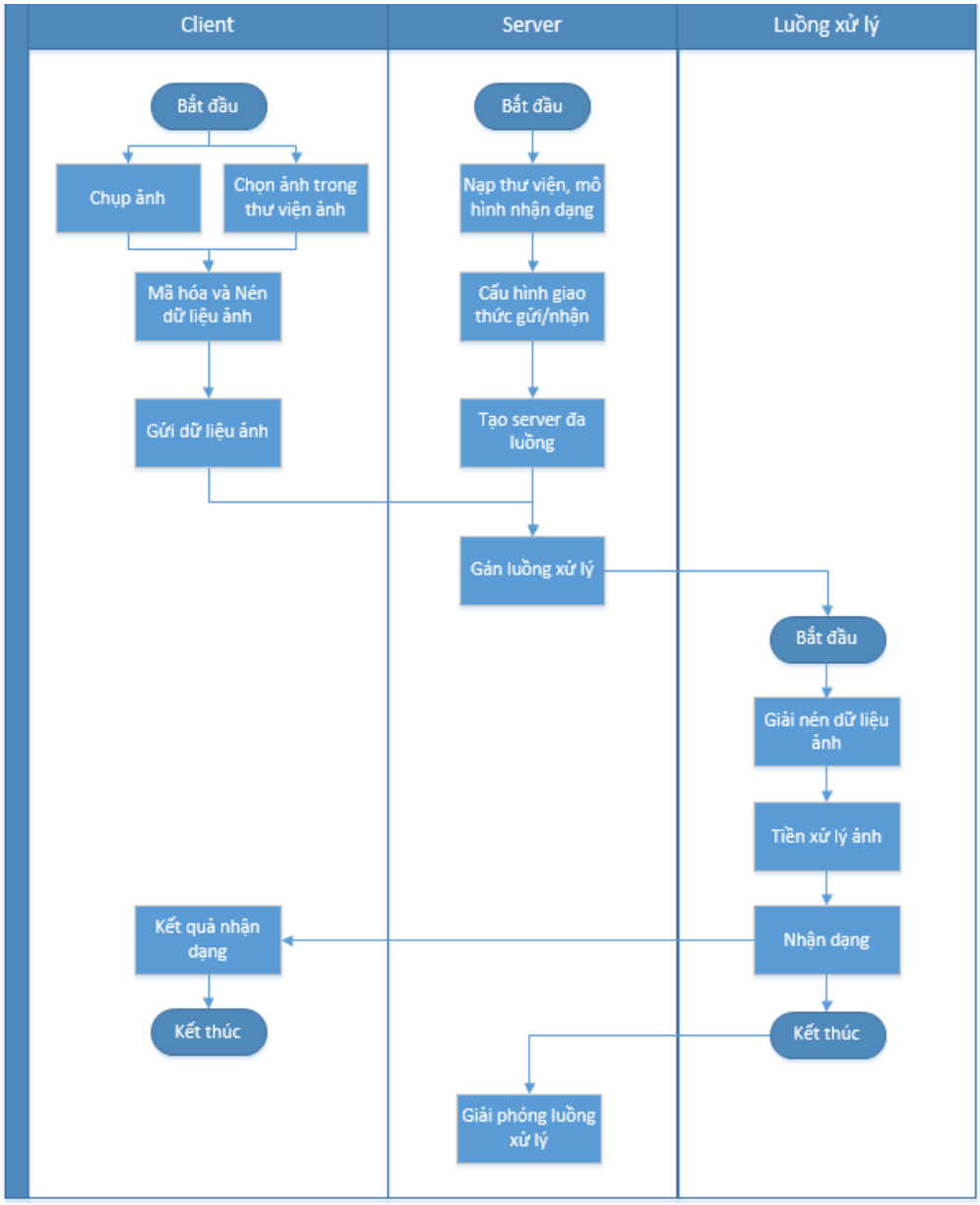
- Client (ứng dụng trên trình duyệt)

  1. Người phân loại khởi động ứng dụng/website.
  2. Người phân loại thực hiện chọn ảnh chụp x-quang trước đó được lưu trong máy tính để gửi.
  3. Ảnh chụp được mã hóa, nén lại và gửi tới máy chủ.
  4. Ứng dụng đợi nhận kết quả phân loại từ máy chủ gửi về và hiển thị cho người phân loại

- Chương trình Server

  1. Chương trình được khởi động và nạp các thư viện cần thiết.
  2. Chương trình nạp mô hình phân loại đã được huấn luyện trước đó.
  3. Giao thức gửi, nhận dữ liệu giữa ứng dụng phía client và chương trình server được cấu hình.
  4. Một loại các luồng xử lý được khởi tạo, đặt trạng thái ban đầu là trạng thái sẵn sàng.
  5. Khi có ứng dụng client kết nối tới, chương trình kiểm tra trong danh sách các luồng xử lý và chọn một luồng đang ở trạng thái sẵn sàng để nhận và tính toán dữ liệu do client gửi tới.
  6. Trong luồng xử lý:
     - Bắt đầu quá trình tính toán phân loại, cờ trạng thái là "bận".
     - Thực hiện giải nén dữ liệu thành dữ liệu ảnh gốc.
     - Sử dụng mô hình đã nạp để phân loại ảnh.
     - Trả kết quả phân loại về cho ứng dụng client.
     - Kết thúc quá trình tính toán.
  7. Khi luồng xử lý đã hoàn thành quá trình tính toán phân loại, chương trình giải phóng luồng xử lý bằng cách cập nhật lại trạng

thái hiện tại của luồng.



Hình 3.2: Luồng hoạt động chính của hệ thống.

3.3. Một số kết quả chương trình

3.3.1. Một số giao diện chính của chương trình

3.3.1.1. Giao diện quản trị

3.3.1.2. Giao diện người dùng

3.3.2. Một số chức năng chính của chương trình

3.3.3. Một số một số ca phân loại được thực hiện bởi chương trình

Kết luận

Phân loại ảnh số là một lĩnh vực nghiên cứu hấp dẫn vì có thể áp dụng trong rất nhiều bài toán thực tế. Đây cũng là một bài toán phức tạp nhưng không quá khó để giải quyết nếu ta biết ứng dụng các thành tựu nghiên cứu trong các lĩnh vực như xử lý ảnh số, trí tuệ nhân tạo... Trong đó, việc ứng dụng thành quả của Deep learning mà trong đó đặc biệt là các mô hình của mạng CNN cho ta các kết quả thực sự ấn tượng.

Kết quả đã đạt được

Sau một thời gian tìm hiểu nghiên cứu, luận văn đã trình bày được các vấn đề sau:

- Trình bày khái quát về CNN và bài toán chẩn đoán bệnh lao
- Hệ thống hóa một số mô hình học sâu hỗ trợ chẩn đoán
- Cài đặt thử nghiệm một trong các mô hình đã được hệ thống hóa

Hướng hoàn thiện và phát triển tiếp theo:

Chương trình tuy đã đảm bảo được những chức năng chính yếu nhất của luận văn, nhưng để áp dụng vào thực tế thì vẫn chưa thể được. Lý do chính cho việc này là do sự khác biệt giữa nguồn ảnh đầu vào. Như đã trình bày ở Chương 3, nguồn ảnh đầu vào của bài toán luận văn là ảnh định dạng PNG, tuy nhiên, thực tế nguồn ảnh x-quang y tế được chụp

qua các thiết bị thu nhận ảnh y tế (CT, MRI...) hầu hết lại ở định dạng DICOM. Việc không đồng nhất về định dạng ảnh khiến cho chương trình hiện nay chưa thể đưa vào sử dụng trong thực tế.

Một vấn đề khác khi nghiên cứu bài toán của luận văn với bộ dữ liệu do Tawsifur Rahman và cộng sự [5] cung cấp là chất lượng ảnh đầu vào không đồng đều. Hầu hết ảnh trong bộ dữ liệu đều có chất lượng tốt, sắc nét, rõ ràng. Nhưng cũng có một vài ảnh mờ, không thực sự rõ nét. Ảnh chất lưởng kém hơn ít nhiều cũng sẽ ảnh hưởng đến độ chính xác của chương trình. Việc nâng cao chất lượng ảnh đầu vào cho chương trình nói riêng và cả lĩnh vực Thị giác Máy - Computer Vision - nói chung là vấn đề rất quan trọng.

Từ những vấn đề nêu trên, tác giả đề xuất hướng phát triển tiếp theo là hoàn thiện thêm các chức năng liên quan đến nâng cao chất lượng ảnh đầu vào, chức năng kết nối với thiết bị thu nhận ảnh y tế (CT, MRI...) để hoàn thiện chương trình có thể ứng dụng vào thực tiễn.

Tài liệu tham khảo

[1] WHO *Global tuberculosis report 2020*, báo cáo tại https://www.who.int/publications/i/item/9789240013131, 2020

[2] Bộ Y tế *Chuẩn đoán bệnh lao*, https://healthvietnam.vn/thu-vien/tai-lieu-tieng-viet/ho-hap/chan-doan-benh-lao, 2015

[3] Nguyễn Thanh Tuấn *Deep learning cơ bản (Chưa xuất bản)*, https://drive.google.com/file/d/1lNjzISABdoc7SRq8tg-xkCRRZRABPCKi/view, 2019

[4] Võ Thị Một, Võ Duy Nguyên, Nguyễn Tấn Trần Minh Khang *Trích chọn đặc trưng và phân loại ảnh x-quang phổi*, TNU Journal of Science and Technology 226(07): 182 - 189 PDF, 2021

[5] Tawsifur Rahman, Amith Khandakar, Muhammad A. Kadir, Khandaker R. Islam, Khandaker F. Islam, Zaid B. Mahbub, Mohamed Arselene Ayari, Muhammad E. H. Chowdhury *Reliable Tuberculosis Detection using Chest X-ray with Deep Learning, Segmentation and Visualization*, IEEE Access - Vol. 8, 2020

[6] Grace W. Lindsay *Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future*, arXiv:2001.07092 [q-bio.NC] (PDF), 2001

[7] Karen Simonyan, Andrew Zisserman *Very deep convolutional networks for large-scale image recognition*, ICLR 2015 (PDF), 2015

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun *Deep Residual Learning for Image Recognition*, arXiv:1512.03385 (PDF), 2015

[9] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger *Densely Connected Convolutional Networks*, arXiv:1608.06993 (PDF), 2016

[10] Cedric Okinda, Innocent Nyalala, Tchalla Korohou, Celestine Okinda, Jintao Wang, Tracy Achieng, Patrick Wamalwa, Tai Mang, Mingxia Shen *A review on computer vision systems in monitoring of poultry: A welfare perspective*, Artificial Intelligence in Agriculture - Volume 4 - Pages 184-208, 2020

[11] Wenqi Liu, Kun Zeng *SparseNet: A Sparse DenseNet for Image Classification*, arXiv:1804.05340 (PDF), 2018