

TRƯỜNG ĐẠI HỌC BÁCH KHOA – ĐẠI HỌC ĐÀ NẴNG
KHOA ĐIỆN TỬ VIỄN THÔNG

--∞📖∞--



BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH

PHÁT HIỆN VẾT NÚT TRÊN BỀ MẶT TƯỜNG

Giáo viên hướng dẫn: TRẦN THỊ MINH HẠNH

Sinh viên thực hiện: LÊ HẢI ĐĂNG LÂM

TRẦN MINH NGUYỄN

Đà Nẵng, 6/2022

TÓM TẮT

Tên đề tài: PHÁT HIỆN VẾT NÚT MẶT TƯỜNG

Sinh viên thực hiện:

STT	Tên sinh viên	Mã số sinh viên	Lớp
1	Lê Hải Đăng Lâm	106180089	18DT2
2	Trần Minh Nguyên	106180099	18DT2

Các vết nứt là dấu hiệu ban đầu của sự xuống cấp của bất kỳ cơ sở hạ tầng dân dụng nào, xuất hiện do nhiều nguyên nhân khác nhau, chẳng hạn như chuyển dịch kết cấu nền, co rút và giãn nở, pha trộn vật liệu không cân bằng, đất bị trương nở, quá tải, thiên tai, ...

Với những công trình xuất hiện vết nứt thường tiềm ẩn những mối nguy hiểm, vì vậy việc sử dụng máy móc trong quá trình đánh giá, sẽ giúp đảm bảo an toàn, tránh những rủi ro cho con người.

Khoa học ngày càng phát triển, những năm gần đây, “trí tuệ nhân tạo” – AI (Artificial Intelligence) và cụ thể hơn là “học máy” - Machine Learning nổi lên như một bằng chứng của cuộc cách mạng công nghiệp lần thứ tư. Trí tuệ nhân tạo đang len lỏi vào mọi lĩnh vực trong đời sống. Xe tự hành của Google và Tesla, hệ thống gắn thẻ khuôn mặt trong ảnh của Facebook, trợ lý ảo Siri của Apple, hệ thống gợi ý sản phẩm của các sàn thương mại điện tử, hệ thống gợi ý phim của Netflix, máy chơi cờ vây AlphaGo của Google DeepMind,... là một vài trong vô vàn những ứng dụng của AI/Machine Learning.

Là sinh viên chuyên ngành Kỹ thuật máy tính, với đề tài “Phát hiện vết nứt trên bề mặt tường” với mong muốn sử dụng những kiến thức mình học được về Machine Learning kết hợp sử dụng ngôn ngữ lập trình Python để tạo ra một chương trình hỗ trợ cảnh báo và giám sát giúp các nhà thi công sớm nhận biết rủi ro về công trình. Nội dung đồ án gồm có 3 chương:

- Chương 1: Tổng quan về hệ thống phát hiện vết nứt mặt tường
- Chương 2: Cơ sở lý thuyết
- Chương 3: Thiết kế và thực hiện hệ thống

LỜI MỞ ĐẦU

Các vết nứt là dấu hiệu ban đầu của sự xuống cấp của bất kỳ cơ sở hạ tầng dân dụng nào, xuất hiện do nhiều nguyên nhân khác nhau, chẳng hạn như chuyển dịch kết cấu nền, co rút và giãn nở, pha trộn vật liệu không cân bằng, đất bị trương nở, quá tải, thiên tai v.v. Các nhiệm vụ phát hiện vết nứt có thể được thực hiện bằng cách thu thập thông tin theo cách thủ công, tức là kiểm tra và đánh giá cấu trúc trực quan bởi con người hoặc tự động. Các kỹ thuật kiểm tra thủ công tốn nhiều công sức, thời gian, phụ thuộc vào kiểm tra viên và dễ bị ảnh hưởng bởi sự kém nhạy bén của kiểm tra viên. Để khắc phục tất cả các vấn đề liên quan đến đánh giá thủ công, kỹ thuật kiểm tra tự động cung cấp một giải pháp hiệu quả làm giảm tính chủ quan và có thể được sử dụng như một giải pháp thay thế cho mắt người.

Với những công trình xuất hiện vết nứt thường tiềm ẩn những mối nguy hiểm, vì vậy việc sử dụng máy móc trong quá trình đánh giá, sẽ giúp đảm bảo an toàn, tránh những rủi ro cho con người.

Khoa học ngày càng phát triển, những năm gần đây, “trí tuệ nhân tạo” – AI (Artificial Intelligence) và cụ thể hơn là “học máy” - Machine Learning nổi lên như một bằng chứng của cuộc cách mạng công nghiệp lần thứ tư. Trí tuệ nhân tạo đang len lỏi vào mọi lĩnh vực trong đời sống. Xe tự hành của Google và Tesla, hệ thống tự tag khuôn mặt trong ảnh của Facebook, trợ lý ảo Siri của Apple, hệ thống gợi ý sản phẩm của sàn thương mại điện tử, hệ thống gợi ý phim của Netflix, máy chơi cờ vây AlphaGo của Google DeepMind,... là một vài trong vô vàn những ứng dụng của AI/Machine Learning.

Mạng nơ-ron tích chập – CNN (Convolutional Neural Networks) là một trong những mô hình Deep Learning tiên tiến giúp chúng ta hiện thực hóa Machine Learning cũng như chinh phục trí tuệ nhân tạo. CNN phổ biến nhất và có ảnh hưởng nhiều nhất trong cộng đồng “thị giác máy tính” - Computer Vision (một lĩnh vực bao gồm các phương pháp thu nhận, xử lý ảnh kỹ thuật số, phân tích và nhận dạng các hình ảnh). CNN được dùng trong trong nhiều bài toán như nhận dạng ảnh, phân tích video, ảnh MRI, hoặc cho các bài của lĩnh vực xử lý ngôn ngữ tự nhiên và hầu hết đều giải quyết tốt các bài toán này.

Là sinh viên chuyên ngành Kỹ thuật máy tính, với đề tài “Phát hiện vết nứt mặt tường” tôi mong muốn sử dụng những kiến thức mình học được về Machine Learning kết hợp sử dụng ngôn ngữ lập trình Python để tạo ra một chương trình hỗ trợ cảnh báo và giám sát giúp các nhà thi công sớm nhận biết và tránh các rủi ro về công trình.

CHƯƠNG 1: TỔNG QUAN VỀ HỆ THỐNG PHÁT HIỆN VẾT NÚT TRÊN BỀ MẶT TƯỜNG

1.1. Giới thiệu:

Chương 1 sẽ giới thiệu tổng quan về Machine Learning và sơ lược về cách xây dựng hệ thống, đối tượng được phát hiện và phương pháp để thực hiện.

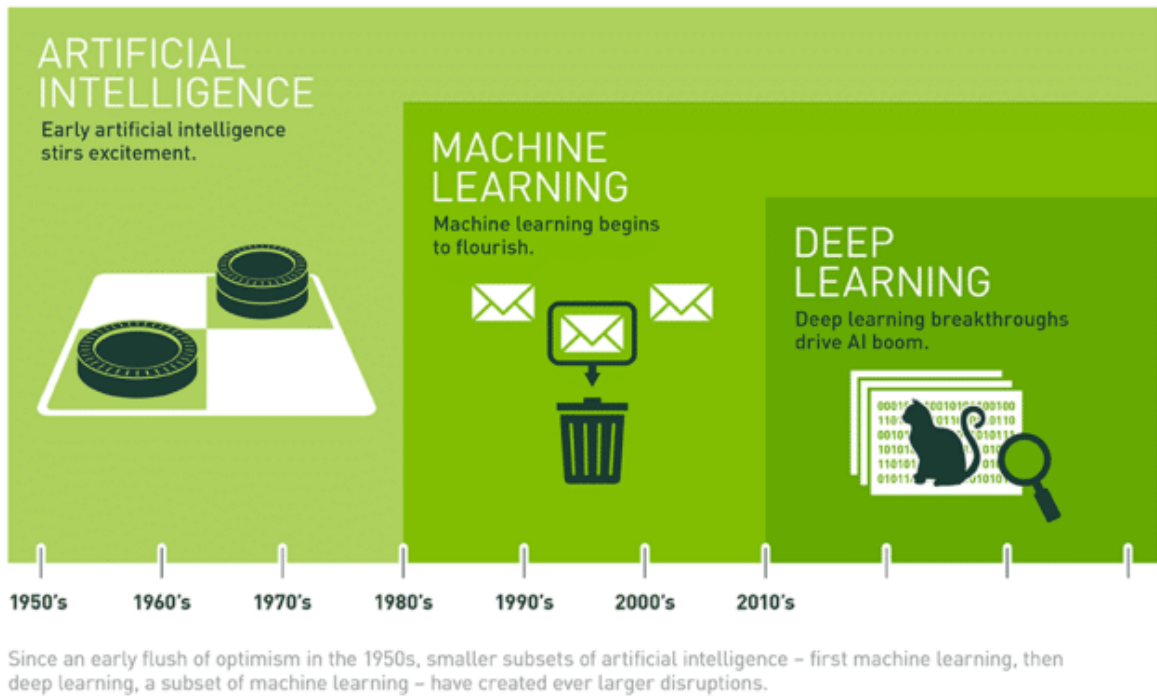
1.2. Giới thiệu về Machine Learning:

Những năm gần đây, AI - Artificial Intelligence (Trí Tuệ Nhân Tạo), và cụ thể hơn là Machine Learning (Học Máy hoặc Máy Học) nổi lên như một bằng chứng của cuộc cách mạng công nghiệp lần thứ tư (1 - động cơ hơi nước, 2 - năng lượng điện, 3 - công nghệ thông tin). Trí Tuệ Nhân Tạo đang len lỏi vào mọi lĩnh vực trong đời sống mà có thể chúng ta không nhận ra. Xe tự hành của Google và Tesla, hệ thống tự tag khuôn mặt trong ảnh của Facebook, trợ lý ảo Siri của Apple, hệ thống gợi ý sản phẩm của Amazon, hệ thống gợi ý phim của Netflix, máy chơi cờ vây AlphaGo của Google DeepMind, ..., chỉ là một vài trong vô vàn những ứng dụng của AI/Machine Learning.

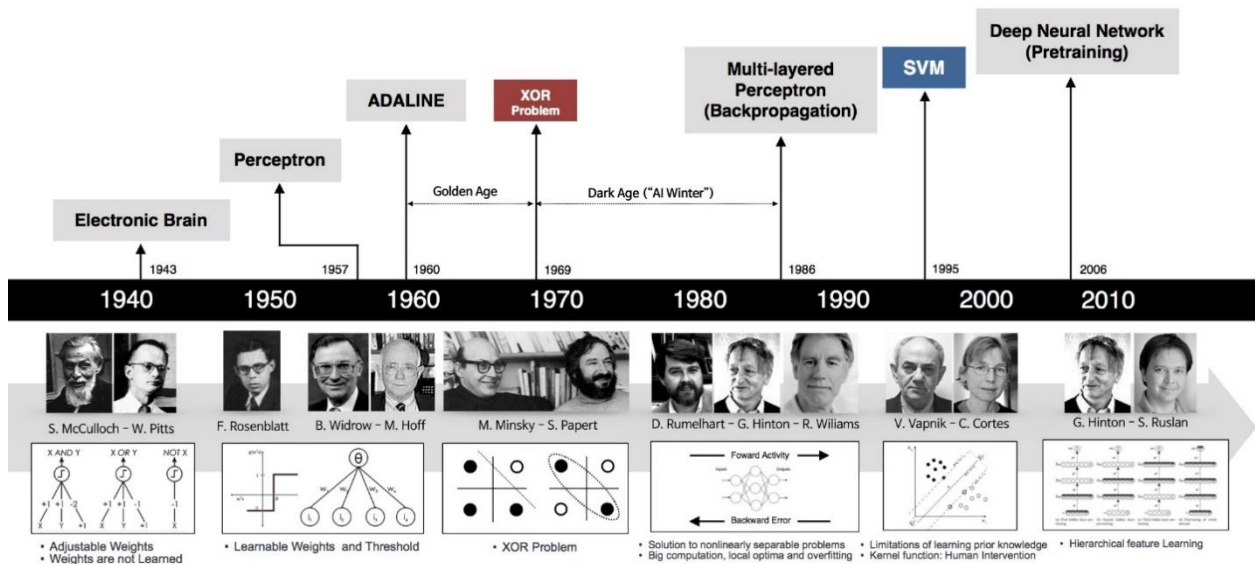
Machine learning (ML) hay máy học là một nhánh của trí tuệ nhân tạo (AI), nó là một lĩnh vực nghiên cứu cho phép máy tính có khả năng cải thiện chính bản thân chúng dựa trên dữ liệu mẫu (training data) hoặc dựa vào kinh nghiệm (những gì đã được học). Machine learning có thể tự dự đoán hoặc đưa ra quyết định mà không cần được lập trình cụ thể.

Bài toán machine learning thường được chia làm hai loại là dự đoán (prediction) và phân loại (classification). Các bài toán dự đoán như dự đoán giá nhà, giá xe... Các bài toán phân loại như nhận diện chữ viết tay, nhận diện đồ vật...

Có thể diễn tả mối quan hệ và sự hình thành của AI, ML, DL qua hình ảnh sau:



Hình 1.1. Mối quan hệ giữa AI, Machine Learning và Deep Learning [1]



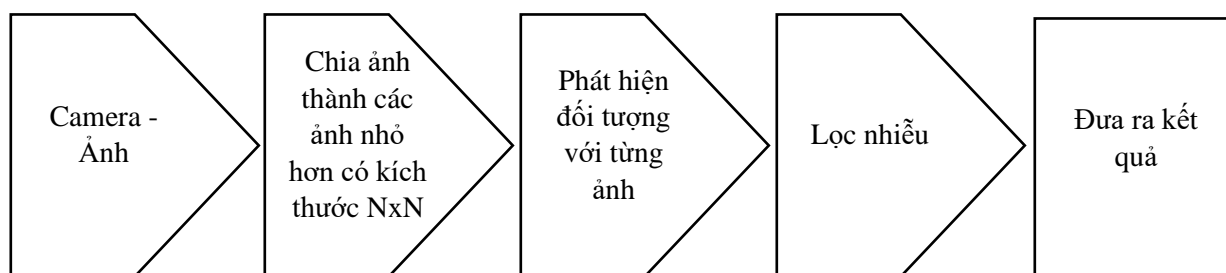
Hình 1.2. Lịch sử phát triển Machine Learning [2]

Qua hình ảnh trên ta có thể thấy được “Trí tuệ nhân tạo” hay AI là lĩnh vực bao quát tất cả và được hình thành từ những năm 1940-1950. Đến ngày nay đã phát triển rất mạnh mẽ trong mọi lĩnh vực và đóng góp một phần không nhỏ trong cuộc sống hằng ngày. Với sự phát triển mạnh mẽ trên thì lần lượt các thuật ngữ mới lần lượt đã được ra đời: “Machine

Learning” vào năm 1980 và “Deep Learning” vào năm 2010, đánh dấu sự phát triển của Trí tuệ nhân tạo ngày nay.

1.3. Giới thiệu hệ thống phát hiện vết nứt:

Nhiệm vụ của đề tài là nhận biết được các vết nứt trên mặt tường dựa vào xử lý ảnh và các thuật toán xử lý. Bản chất của quá trình là phân chia nhỏ đầu vào thành các đầu vào nhỏ hơn, phát hiện đối tượng trong các đầu đó, kết hợp với những thuật toán xử lý ảnh để loại bỏ nhiễu không mong muốn. Quá trình được diễn ra như sau:



Hình 1.3. Quá trình thực hiện

1.4. Mục tiêu đề tài:

- + Tìm hiểu các thuật toán phát hiện vết nứt
- + Nghiên cứu về mạng Nơ-ron tích chập (CNN) – mạng nơ-ron được sử dụng nhiều trong lĩnh vực thị giác máy tính
- + Tìm hiểu mô hình ResNet, đặc biệt là ResNet-50
- + Nâng cao khả năng thiết kế và lập trình bằng ngôn ngữ lập trình Python
- + Rèn luyện khả năng tự nghiên cứu, tìm hiểu tài liệu
- + Huấn luyện mô hình đạt được độ chính xác tốt (>90%)

1.5. Các nghiên cứu về đề tài:

Dự án	Tác giả	Dataset	Mô hình	Kết quả
A comparison of deep convolutional neural networks for image-based detection of concrete surface cracks [3]	Marek Słoński	SDNET	VGG16	85%
Comparison of deep convolutional neural networks and edge detectors	Sattar Dorafshan1a ,	SDNET	AlexNet	97%

for image-based crack detection in concrete [4]	Robert J. Thomasb , Marc Maguire			
Deep Learning for Detecting Building Defects Using Convolutional Neural Networks [5]	Husein Perez, Joseph H. M. Tah and Amir Mosavi	SDNET	VGG16 ResNet50	97.83% 96.23%
Performance Evaluation of Deep CNN-Based Crack Detection and Localization Techniques for Concrete Structures [6]	Luqman Ali 1 , Fady Alnajjar	CCIC	VGG16 VGG19 ResNet50	98.7% 96% 99.4%

1.6. Kết luận chương:

Ở chương 1, chúng tôi đã giới thiệu khái quát về Trí tuệ nhân tạo và các dự án liên quan cũng như trình bày tổng quát về các kỹ thuật, thuật toán cơ bản có trong hệ thống. Bên cạnh đó tôi còn nêu rõ các ưu nhược điểm của từng kỹ thuật để có thể lựa chọn ra kỹ thuật thích hợp phục vụ dự án. Từ những yêu cầu trên thì cần phải có kiến thức lý thuyết cần thiết để áp dụng vào quá trình thực hiện, vì thế chương tiếp theo sẽ trình bày về các lý thuyết liên quan sẽ được áp dụng.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Giới thiệu chương:

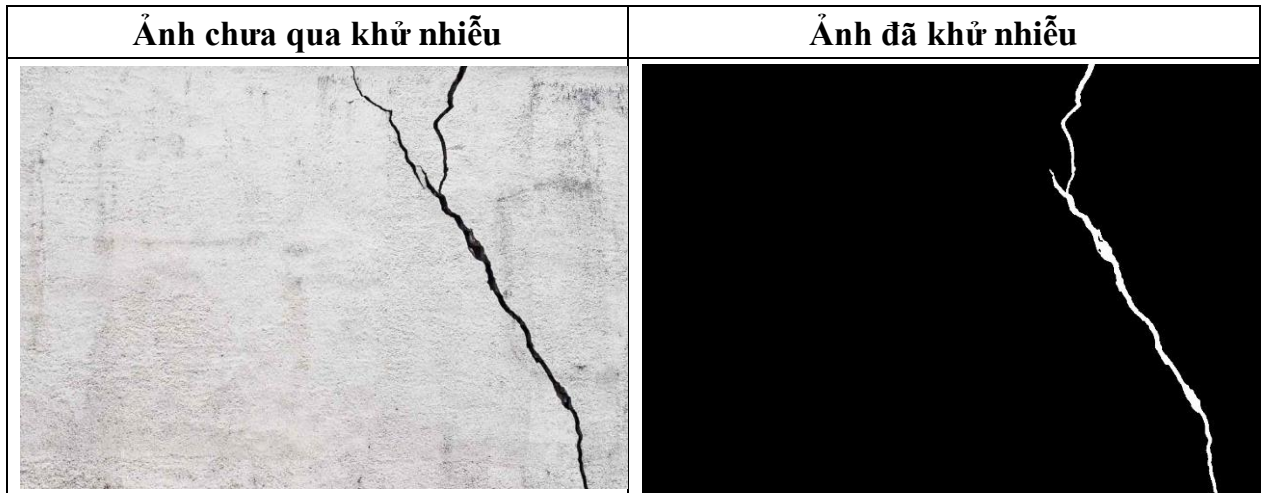
Chương 2 sẽ giới thiệu tổng quát về mạng nơ-ron tích chập và đi sâu vào tìm hiểu các phân lớp có trong đó, bên cạnh đó sẽ đi vào tìm hiểu về lý thuyết của mô hình ResNet-50 để có cơ sở cho việc xây dựng mô hình nhận dạng phía sau.

2.2. Phát hiện vết nứt sử dụng thuật xử lý ảnh:

2.2.1. Các bước thực hiện:

- + Chuyển ảnh qua xám
- + Làm mờ
- + Đảo màu
- + Sử dụng hàm lọc trung vị để lọc nhiễu salt-and-pepper
- + Lọc nhiễu

2.2.2. Kết quả:





2.2.3. Kết luận:

Từ kết quả đạt được ta thấy thuận xử lý sử dụng chi phí thấp nhưng kết quả đạt được không cao, vì các trường hợp đầu vào khác nhau phải sử dụng các công thức xử lý ảnh khác nhau nên không thể sử dụng rộng rãi mà chỉ có thể được sử dụng trong trường hợp cố định. Với phương pháp sử dụng mạng nơ-ron được cho là phương pháp tối ưu.

2.3. Giới thiệu chung mạng nơ-ron tích chập (CNN):

Mạng nơ-ron tích chập (Convolutional Neural Network), còn được gọi là CNN hay ConvNet, là một thuật toán Deep Learning có thể lấy hình ảnh đầu vào, gán độ quan trọng (các trọng số - weights và độ lệch - bias có thể học được) cho các đối tượng, đặc trưng khác nhau trong hình ảnh và có thể phân biệt được từng đối tượng, đặc trưng này với nhau. Công việc tiền xử lý được yêu cầu cho mạng nơ-ron tích chập thì ít hơn nhiều so với các thuật toán phân loại khác.

Kiến trúc của nơ-ron tích chập tương tự như mô hình kết nối của các nơ-ron trong bộ não con người và được lấy cảm hứng từ hệ thống vỏ thị giác trong bộ não (visual cortex).

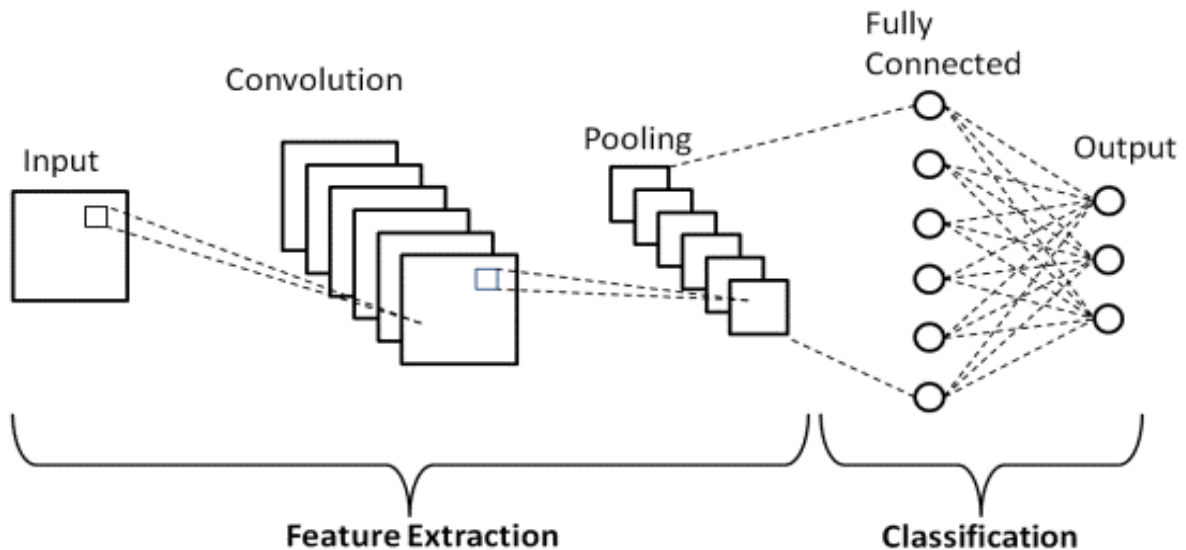
Các nơ-ron riêng lẻ chỉ phản ứng với các kích thích trong một khu vực hạn chế của trường thị giác được gọi là Trường tiếp nhận (Receptive Field). Một tập hợp các trường như vậy chồng lên nhau để bao phủ toàn bộ khu vực thị giác.

Mạng nơ-ron nhân tạo là một chuỗi các thuật toán được sử dụng để tìm ra mối quan hệ của một tập dữ liệu thông qua cơ chế vận hành của bộ não sinh học. Mạng nơ-ron nhân tạo thường được huấn luyện qua một tập dữ liệu chuẩn cho trước, từ đó có thể đúc rút được kiến thức từ tập dữ liệu huấn luyện, và áp dụng với các tập dữ liệu khác với độ chính xác cao.

Các phương pháp sử dụng để huấn luyện mạng nơ-ron nhân tạo ngày càng tối ưu hơn về mặt tính toán và phục vụ cho nhiều mục đích khác nhau. Hiện nay, kiến trúc mạng nơ-ron ngày càng được hoàn thiện cho nhiều nhiệm vụ, trong đó mạng nơ-ron tích chập được chú ý rất nhiều vì tính hiệu quả trong thị giác máy tính. Mạng nơ-ron tích chập với các cải tiến góp phần giảm thời gian tính toán và tăng độ chính xác hứa hẹn sẽ là một trong những phương pháp được áp dụng rất nhiều vào thực tế trong tương lai.

2.4. Cấu trúc mạng nơ-ron tích chập:

Mạng nơ-ron tích chập (CNN) là một tập hợp các lớp Convolution chồng lên nhau và sử dụng các hàm kích hoạt phi tuyến như ReLU để kích hoạt các trọng số trong các node. Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo.

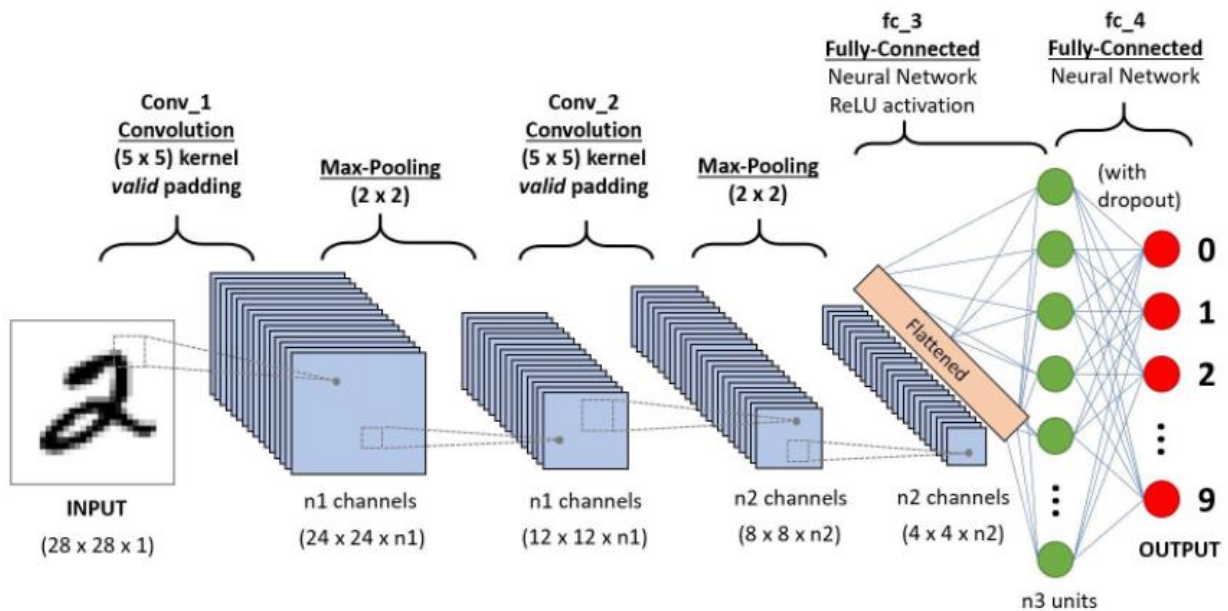


Hình 2.1. Cấu trúc mạng nơ-ron tích chập [7]

Các lớp cơ bản trong một mạng nơ-ron tích chập bao gồm:

- Lớp tích chập
- Lớp kích hoạt phi tuyến
- Lớp lấy mẫu
- Lớp kết nối đầy đủ

Các lớp này được thay đổi về số lượng và cách sắp xếp để tạo ra các mô hình huấn luyện phù hợp cho từng bài toán khác nhau.



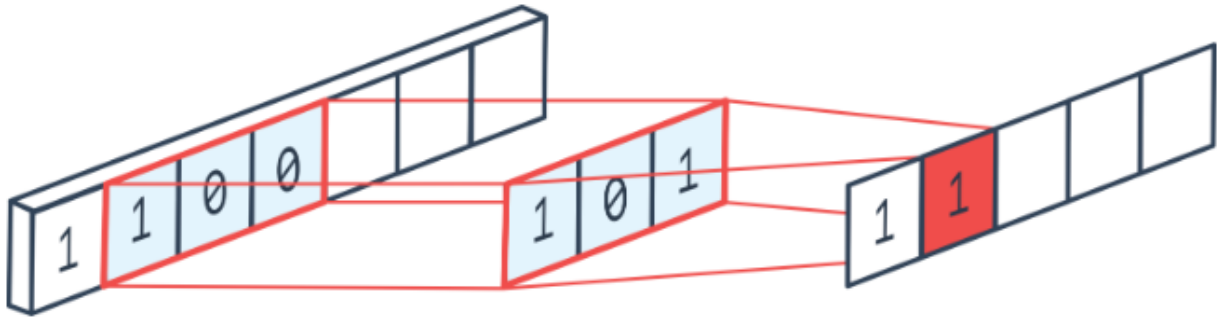
Hình 2.2. Sơ đồ khối phân lớp mô hình CNN [8]

2.4.1. Lớp tích chập:

Tích chập là lớp đầu tiên để trích xuất các tính năng từ hình ảnh đầu vào. Tích chập duy trì mối quan hệ giữa các pixel bằng cách tìm hiểu các tính năng hình ảnh bằng cách sử dụng các ô vuông nhỏ của dữ liệu đầu vào. Nó là một phép toán có hai đầu vào như ma trận hình ảnh và một bộ lọc hoặc hạt nhân.

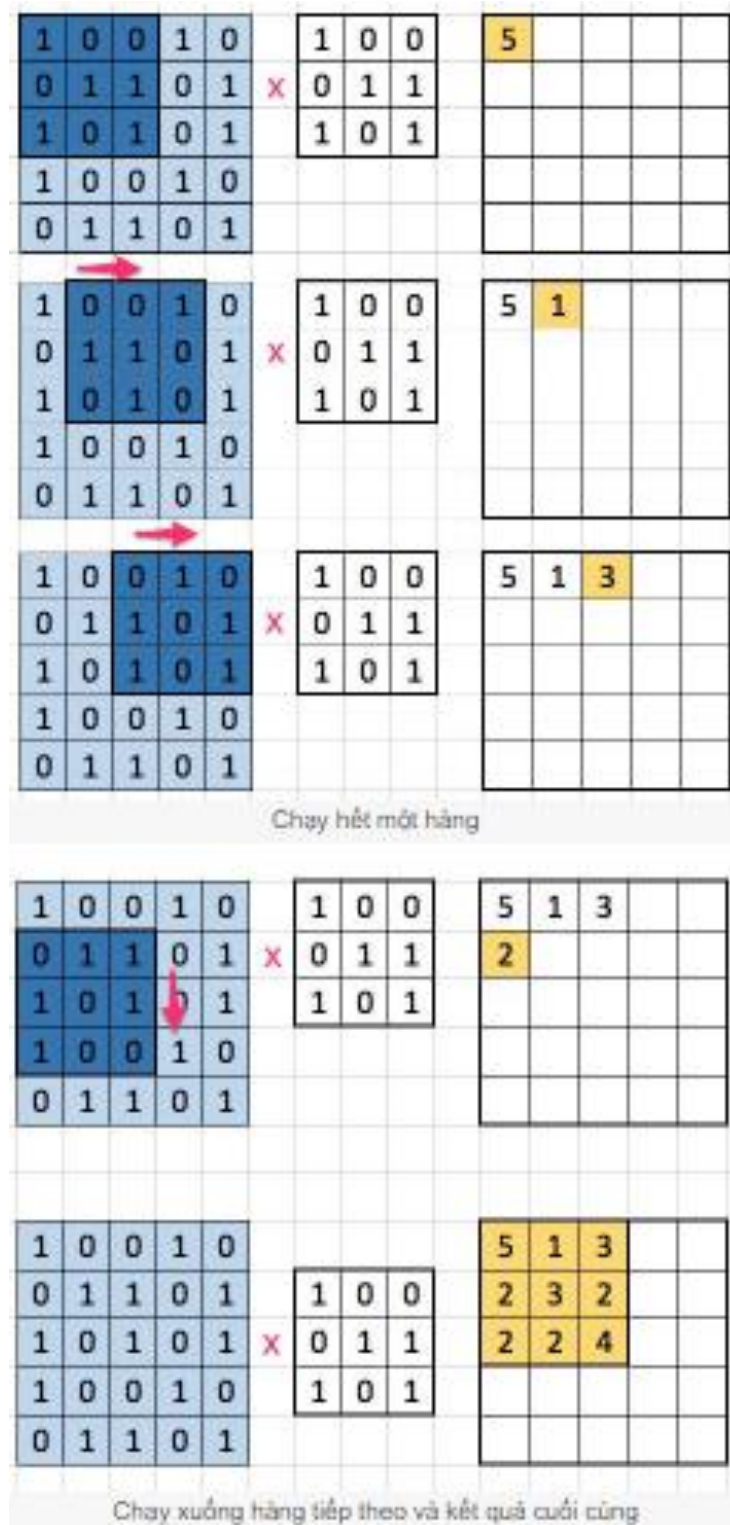
Số chiều của phép chập chính là số chiều mà hàm lọc có thể di chuyển được, cụ thể như sau:

- Convolution 1D: Phép chập 1 chiều cho phép hàm lọc di chuyển dọc theo chiều dài của ma trận.



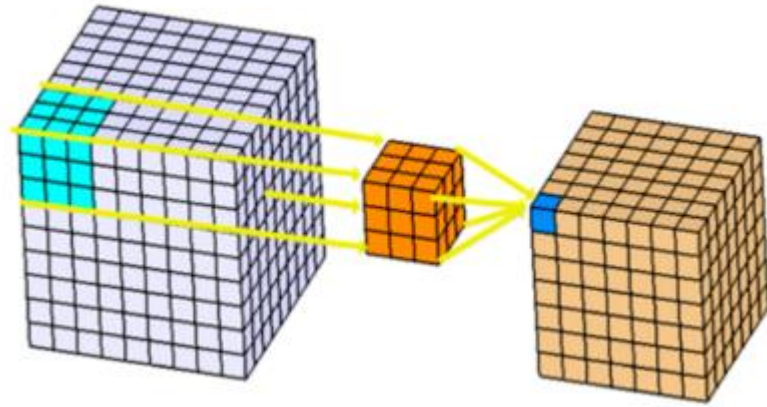
Hình 2.3. Phép chập một chiều với kích thước 1x3 [9]

- Convolution 2D: Hàm lọc trong phép chập 2 chiều này sẽ di chuyển theo 2 chiều (chiều rộng và chiều cao) của ma trận ảnh.



Hình 2.4. Phép chập hai chiều với kích thước 3x3 [10]

- Convolution 3D: Hàm lọc trong phép chập 3 chiều này sẽ di chuyển theo 3 chiều rộng, chiều cao và chiều sâu của ma trận ảnh.

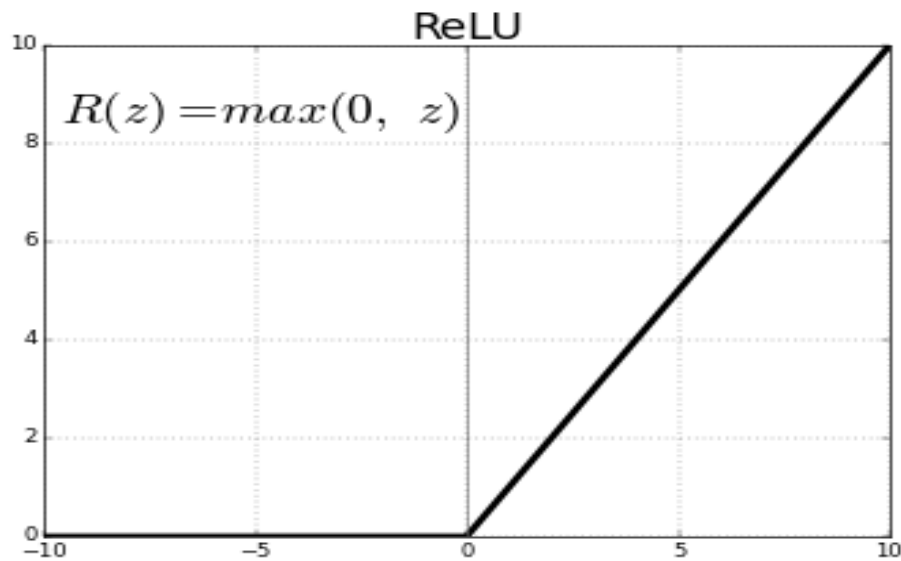


Hình 2.5. Phép chập ba chiều với kích thước $3 \times 3 \times 3$ [11]

2.4.2. Lớp kích hoạt phi tuyến:

Hàm kích hoạt có tác dụng mô phỏng các neuron có tỷ lệ truyền xung qua axon. Trong activation function thì nó còn có hàm nghĩa là: ReLU, Tanh, Sigmoid, ... Hiện nay, hàm ReLU được dùng phổ biến và vô cùng thông dụng.

Hàm ReLU đơn giản sẽ lọc các giá trị nhỏ hơn không và chỉ cho các giá trị lớn hơn không đi qua.



Hình 2.6. Hàm kích hoạt ReLU [12]

Những đặc điểm vượt trội của hàm kích hoạt ReLU so với các hàm kích hoạt khác:

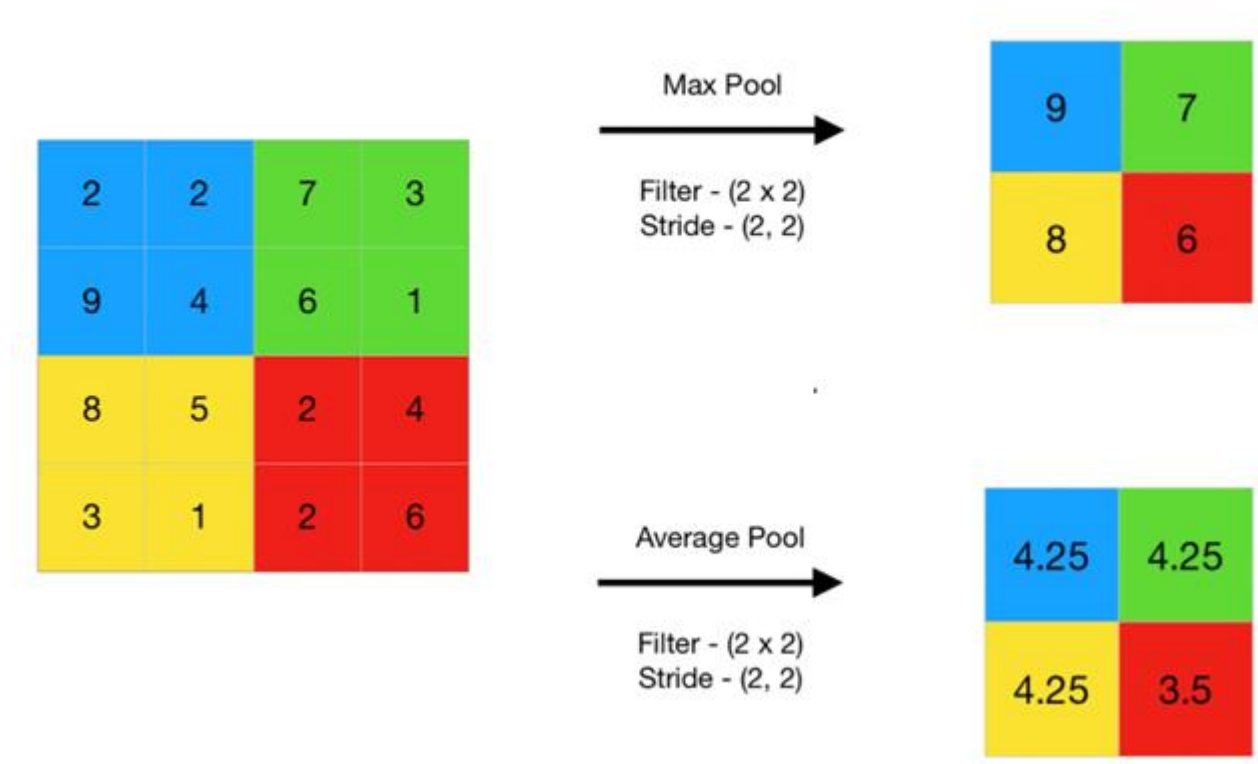
- Tốc độ hội tụ nhanh, không bị bão hòa ở hai đầu.

- Tính toán nhanh.

2.4.3. Lớp lấy mẫu:

Lớp lấy mẫu thường được sử dụng ngay sau lớp tích chập để giảm bớt số lượng tham số khi hình ảnh quá lớn nhưng vẫn giữ lại những thông tin quan trọng cho quá trình tính toán sau này.

Lớp lấy mẫu có nhiều loại khác nhau: Max pooling, Average pooling, Sum pooling.



Hình 2.7. Các loại lấy mẫu [13]

Tương tự như Lớp tích chập, Lớp lấy mẫu sử dụng một ma trận với chức năng tương ứng để quét toàn bộ các vùng trong ma trận ảnh đầu vào và thực hiện phép lấy mẫu. Cuối cùng ta thu được một giá trị duy nhất đại diện cho toàn bộ thông tin của vùng ảnh đó.

2.4.4. Lớp kết nối đầy đủ:

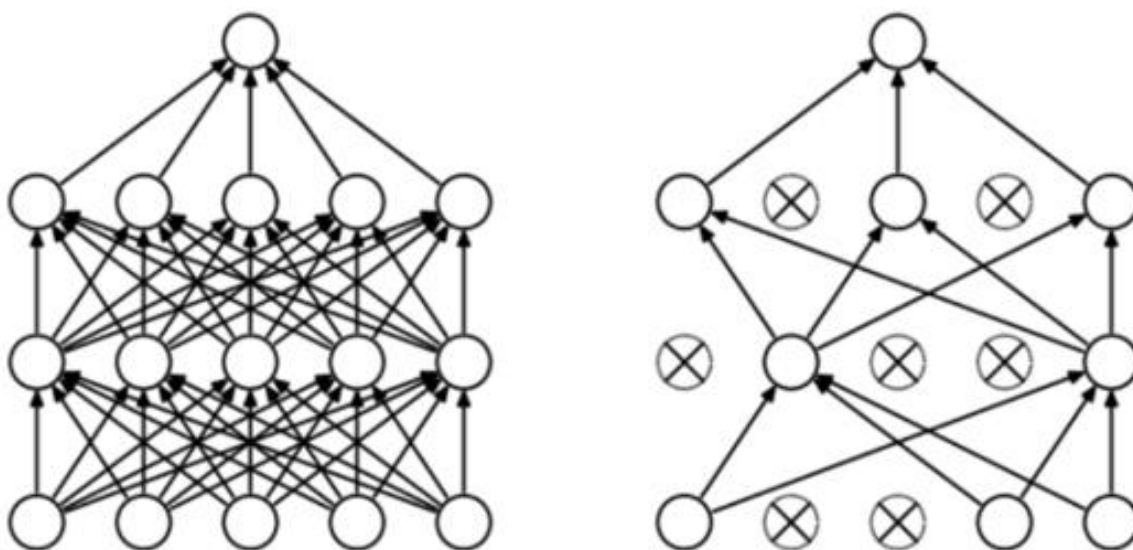
Lớp kết nối đầy đủ là tầng cuối cùng của mô hình mạng nơ-ron tích chập. Tầng này có chức năng chuyển ma trận đặc trưng ở tầng trước thành vectơ chứa xác suất của các đối tượng cần được dự đoán.

Lớp kết nối đầy đủ này được thiết kế dựa vào các tính chất cơ bản của một mạng nơ-ron thuần túy. Các điểm ảnh đầu ra của lớp này lại là đầu vào của lớp tiếp theo. Tuy nhiên việc giảm bớt kích thước ở những lớp trước mà vẫn giữ được những đặc trưng cần thiết cho việc nhận dạng đã giúp cho việc tính toán trong các mô hình mạng nơ-ron truyền thẳng không còn quá phức tạp và tốn thời gian như mạng nơ-ron truyền thống.

2.4.5. Dropout:

Dropout là một kỹ thuật được sử dụng nhằm tránh lỗi quá khớp (over-fitting). Khi sử dụng full connected layer, các neural sẽ phụ thuộc nhiều lần nhau trong suốt quá trình huấn luyện, điều này làm giảm sức mạng cho mỗi neural và dẫn đến bị over-fitting tập train.

Kỹ thuật Dropout sẽ bỏ qua một vài unit trong suốt quá trình huấn luyện trong mô hình, những unit bị bỏ qua được lựa chọn ngẫu nhiên. Tại mỗi giai đoạn huấn luyện, mỗi node có xác suất bị bỏ qua là $1-p$ và xác suất được chọn là p .



Hình 2.8. Mạng nơ-ron thông thường (trái) và mạng nơ-ron áp dụng Dropout (phải) [14]

Kỹ thuật dropout được thực hiện như sau:

Trong pha huấn luyện: với mỗi hidden layer, với mỗi training sample, với mỗi lần lặp, chọn ngẫu nhiên p phần trăm số node và bỏ qua nó (bỏ qua luôn hàm kích hoạt cho các node bị bỏ qua).

Trong pha kiểm tra: Sử dụng toàn bộ hàm kích hoạt, nhưng giảm chúng với tỷ lệ p (do chúng ta bị mất đi $p\%$ hàm kích hoạt trong quá trình huấn luyện).

2.4.6. Softmax:

Softmax là một cách ràng buộc đầu ra của các mạng nơ-ron phải có tổng bằng 1. Qua đó, các giá trị đầu ra của hàm softmax có thể được coi như là một phân phối xác suất của các biến đầu ra. Nó rất hữu ích trong bài toán phân loại đa lớp. Kết quả đầu ra của hàm softmax sẽ có tổng bằng 1. Để làm được điều này hàm softmax sẽ chuyển đổi giá trị đầu ra của mạng nơ-ron bằng cách chia cho tổng giá trị. Lúc này đầu ra có thể coi là một vector của xác suất dự đoán của các class.

2.4.7. Cross Entropy:

Mục đích của việc huấn luyện mạng nơ-ron là tìm ra một tập hợp tham số tối ưu cho bài toán. Việc đó tương ứng với các thao tác tính toán và cập nhật trọng số sao cho cực tiểu hóa hàm lỗi - loss function. Cross-entropy là một loss function, sử dụng để so sánh khoảng cách giữa giá trị đầu ra của softmax và onehot encoding.

CROSS - ENTROPY

The diagram illustrates the Cross-Entropy loss function. It shows two vertical vectors: S (Softmax output) and L (One-hot encoding). S has values $[0.7, 0.2, 0.1]$ and L has values $[1.0, 0.0, 0.0]$. Arrows point from these vectors to the formula $D(S, L) = -\sum_i L_i \log(S_i)$.

Hình 2.9. Công thức Loss function cross-entropy [15]

2.4.8. Epoch – Batch size – Iterations:

Số epoch là số lần duyệt qua hết các dữ liệu trong tập huấn luyện trong quá trình huấn luyện. Khi dữ liệu quá lớn, chúng ta không thể đưa hết mỗi lần tất cả tập dữ liệu vào để huấn luyện được, vì bạn cần một siêu máy tính có lượng RAM và GPU RAM cực lớn để

lưu trữ toàn bộ hình ảnh trên, điều này là bất khả thi đối với người dùng bình thường, phòng lab nhỏ. Buộc lòng chúng ta phải chia nhỏ tập dữ liệu ra thành các batch.

Batch size là số lượng mẫu dữ liệu trong một batch. Batch size thường được chọn là số mũ của 2 ví dụ 16, 32, 64, 128 để CPU/GPU tính toán tốt hơn.

Iterations là số lượng batches cần để hoàn thành 1 epoch.

Ví dụ tập huấn luyện có 32.000 dữ liệu, batch size là 32, khi đó iterations sẽ là $32.000 / 32 = 1.000$ để có thể duyệt qua dữ liệu hoàn thành 1 epoch.

2.4.9. Kỹ thuật tăng cường dữ liệu

Trong Deep learning thì vấn đề dữ liệu có vai trò rất quan trọng ảnh hưởng lớn đến mô hình huấn luyện của mô hình. Chính vì vậy việc có ít dữ liệu khiến cho việc huấn luyện model khó tạo ra được kết quả tốt trong việc dự đoán. Do đó kỹ thuật tăng cường dữ liệu (data augmentation) phục vụ cho việc nếu có ít dữ liệu, thì vẫn có thể tạo ra được nhiều dữ liệu hơn dựa trên những dữ liệu đã có. Có nhiều phương pháp được sử dụng trong việc tăng cường dữ liệu từ cơ sở dữ liệu sẵn có như lật theo chiều dọc, chiều ngang, cắt ngẫu nhiên một phần bức ảnh, chuyển đổi màu của bức ảnh bằng cách thêm giá trị vào 3 kênh màu RGB, thêm nhiễu vào ảnh như nhiễu ngẫu nhiên, nhiễu có mẫu, nhiễu cộng, nhiễu nhân, nhiễu do nén ảnh, nhiễu mờ do chụp không lấy nét, nhiễu mờ do chuyển động. . .

Tùy vào bộ cơ sở dữ liệu mà có cách thức riêng để tăng cường cơ sở dữ liệu để đạt được kết quả tốt nhất. Bản thân mỗi cách tăng cường dữ liệu lại có các yếu tố điều khiển riêng, ví dụ như tần suất sử dụng, góc xoay cho mỗi lần, số lần phóng to, thu nhỏ, . . . Cách tăng cường dữ liệu cho từng giai đoạn huấn luyện trên mỗi epoch có thể khác nhau, epoch đầu ở với tốc độ học lớn có thể khác với các epoch cuối với tốc độ học nhỏ.

2.4.10. Một số cấu trúc CNN:

+ LeNet (1998): Đây là mô hình CNN thành công đầu tiên, đặc biệt là trong việc nhận dạng chữ số, ký tự trong văn bản. Được phát triển bởi Yann Lecun vào cuối những năm 90.

+ AlexNet (2012): Được phát triển bởi Alex Krizhevsky, Ilya Sutskever và Geoff Hinton. Lần đầu được giới thiệu vào năm 2012 với cấu trúc khá tương tự như LeNet nhưng với số lượng neuron, filter và layer lớn hơn, tăng tốc độ tính toán lên 6 lần. Được coi là mạng neural đầu tiên phổ biến rộng rãi khả năng của CNNs.

+ GoogLeNet (2014): Là mạng CNNs tốt nhất năm 2014 được phát triển bởi Szegedy từ Google. Với một số thay đổi như giảm thiểu số lượng tham số trong AlexNet từ 60 triệu xuống còn 4 triệu, sử dụng Average Pooling thay cho Fully Connected layer.

+ ResNet (2015): Là mạng CNNs tốt nhất năm 2015. Bỏ qua Fully Connected layer ở cuối mạng, và sử dụng “special skip connection” và “batch normalization”.

+ DenseNet (2016): một trong những network mới nhất cho visual object recognition. Nó cũng gần giống ResNet nhưng có một vài điểm khác biệt. Densenet có cấu trúc gồm các dense block và các transition layers.

2.5. Mô hình mạng ResNet-50:

2.5.1. Giới thiệu chung về mô hình ResNet:

ResNet (viết tắt của residual network), là mạng học sâu nhận được quan tâm từ những năm 2012 sau cuộc thi LSVRC2012 và trở nên phổ biến trong lĩnh vực thị giác máy. ResNet khiến cho việc huấn luyện hàng trăm thậm chí hàng nghìn lớp của mạng nơ-ron trở nên khả thi và hiệu quả.

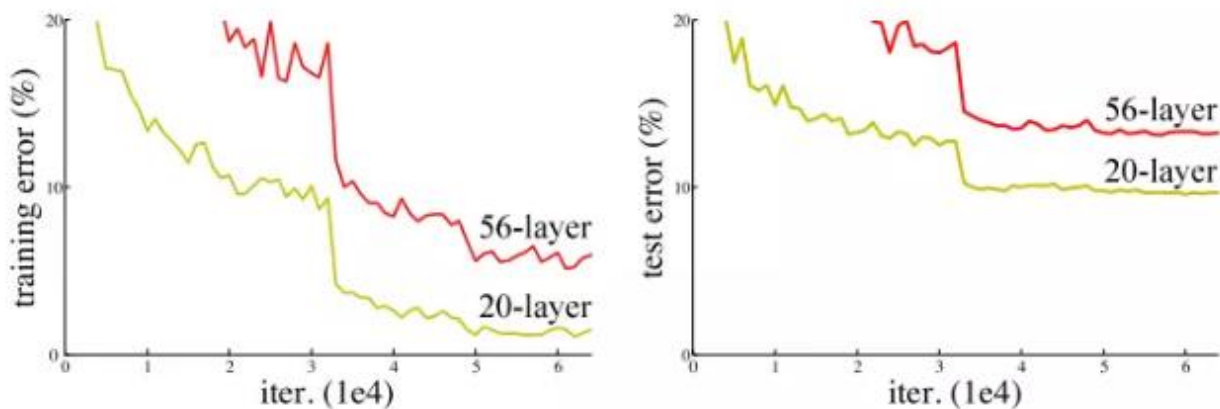
Nhờ khả năng biểu diễn mạnh mẽ của ResNet, hiệu suất của nhiều ứng dụng thị giác máy, không chỉ các ứng dụng phân loại hình ảnh được tăng cường. Một số ví dụ có thể kể đến là các ứng dụng phát hiện đồ vật và nhận dạng khuôn mặt.

Theo định lý gần đúng phổ quát, về mặt kiến trúc, một mạng nơ-ron truyền thẳng có khả năng xấp xỉ mọi hàm với dữ liệu huấn luyện được cung cấp, miễn là không vượt quá sức chứa của nó. Tuy nhiên, xấp xỉ tốt dữ liệu không phải là mục tiêu duy nhất, chúng ta cần một mô hình có khả năng tổng quát hóa dữ liệu. Đó là lý do các kiến trúc sâu trở thành xu hướng của cộng đồng nghiên cứu.

Kể từ AlexNet, các kiến trúc CNN ngày càng sâu hơn. Trong khi AlexNet chỉ có 5 lớp tích chập, mạng VGG và GoogLeNet có đến 19 và 22 lớp tương ứng.

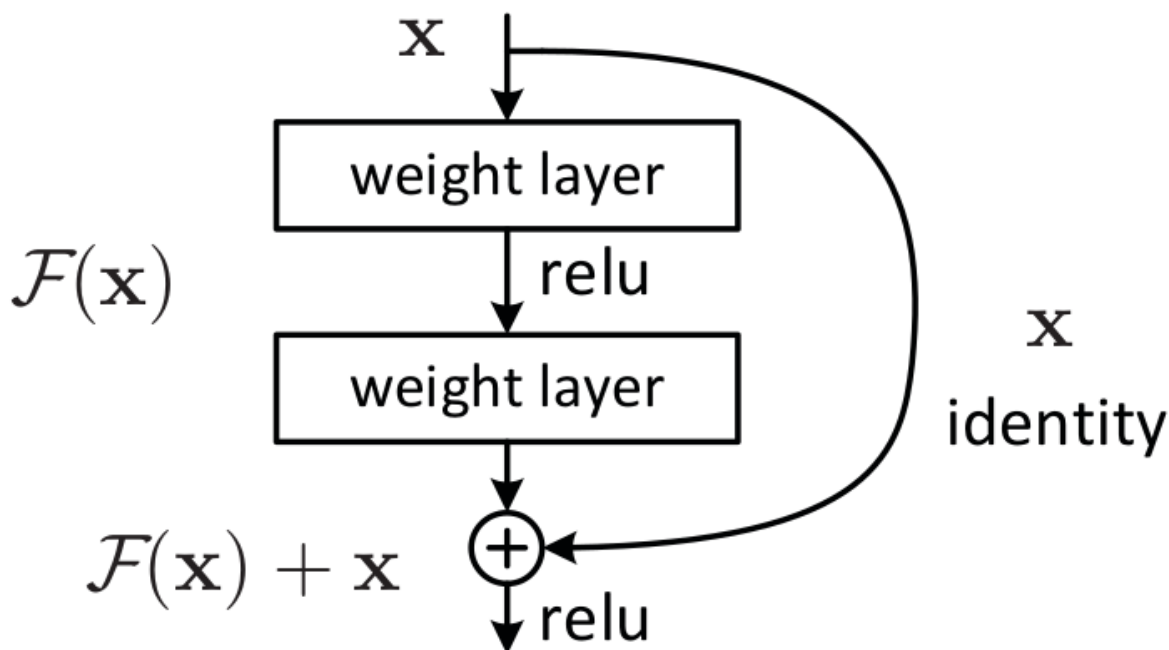
Tuy nhiên, tăng độ sâu mạng không chỉ đơn giản là xếp chồng các lớp lại với nhau. Mạng sâu rất khó huấn luyện vì vấn đề vanishing gradient – vì độ dốc được truyền ngược trở lại các lớp trước đó, phép nhân lặp đi lặp lại có thể làm cho độ dốc cực nhỏ. Kết quả là, hiệu suất của mạng bị bão hòa hoặc giảm hiệu quả nhanh chóng.

Vanishing gradient là vấn đề xảy ra khi huấn luyện các mạng nơ ron nhiều lớp. Khi huấn luyện, giá trị đạo hàm là thông tin phản hồi của quá trình lan truyền ngược. Giá trị này trở nên vô cùng nhỏ tại các lớp nơ ron đầu tiên khiến cho việc cập nhật trọng số mạng không thể xảy ra.



Hình 2.10. Mối tương quan giữa độ sâu và hiệu suất mạng [16]

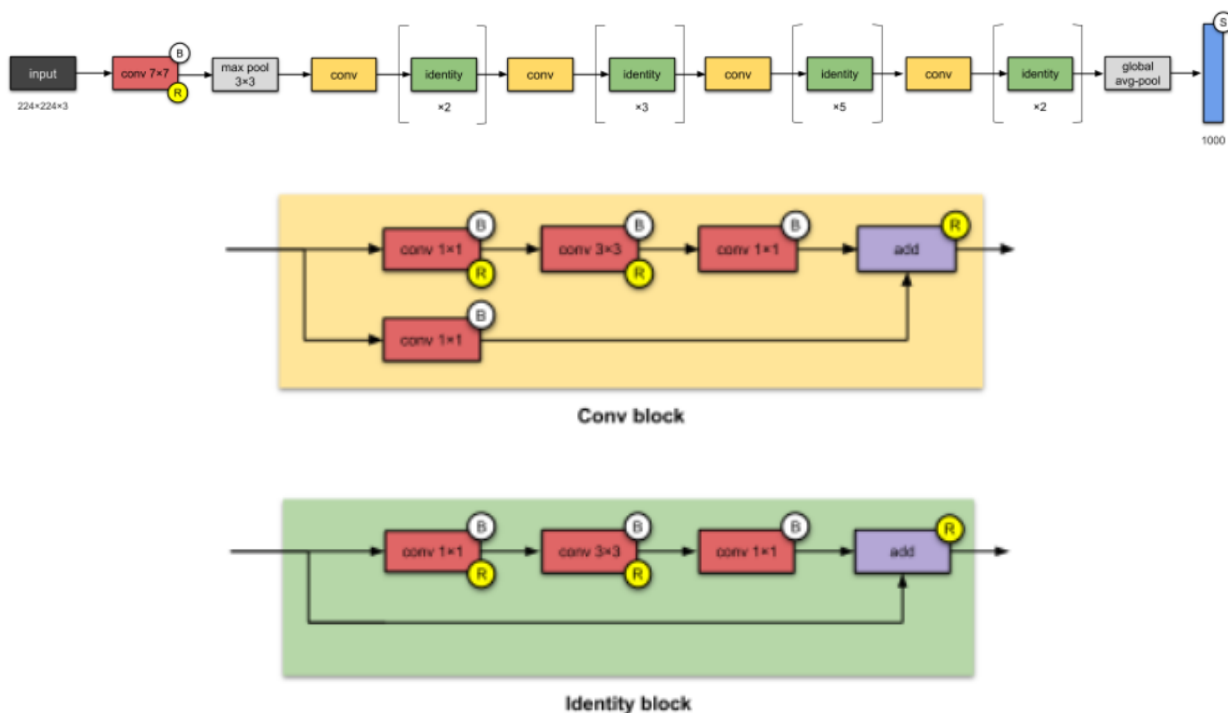
Ý tưởng chính của ResNet là sử dụng kết nối “tắt” đồng nhất để xuyên qua một hay nhiều lớp. Một khối như vậy được gọi là một residual block như trong hình sau:



Hình 2.11. Kết nối “tắt” đồng nhất [17]

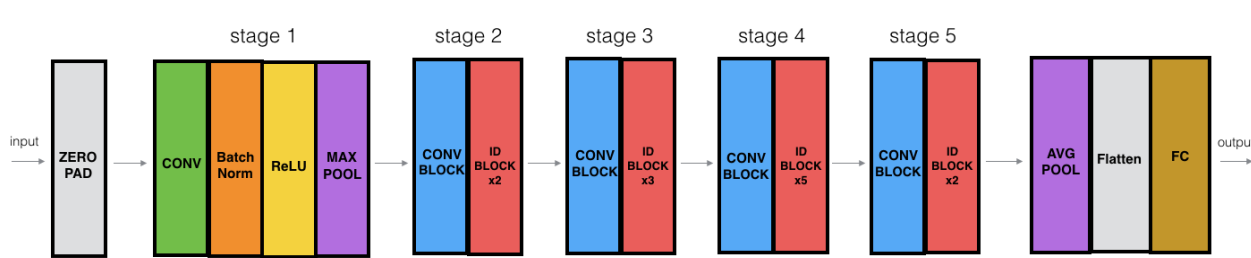
ResNet có khối tích chập (Convolutional Block, chính là Conv block trong hình) sử dụng bộ lọc kích thước 3x3 giống với của InceptionNet. Khối tích chập bao gồm 2 nhánh tích chập trong đó một nhánh áp dụng tích chập 1x1 trước khi cộng trực tiếp vào nhánh còn lại.

Khối xác định (Identity block) thì không áp dụng tích chập 1x1 mà cộng trực tiếp giá trị của nhánh đó vào nhánh còn lại.



Hình 2.12. Conv block (giữa) và Identity block (dưới) [18]

2.5.2. Kiến trúc mạng ResNet-50:



Hình 2.13. Kiến trúc tóm tắt của mạng ResNet-50 [19]

Kiến trúc của ResNet-50 bao gồm 50 layer:

- Zero-padding (3x3)

- Stage 1: có 1 layer
 - Tích chập với 64 filter có kích thước 7x7, với độ trượt (stride) 2x2
 - Batch normalization: chuẩn hóa các feature về trạng thái zero-mean với độ lệch chuẩn 1
 - Hàm kích hoạt phi tuyến ReLU
 - MaxPooling 3x3
- Stage 2: có 9 layer
 - Convolution block (khối tích chập) sử dụng 3 filter: 1x1 conv, 64; 3x3 conv, 64; 1x1 conv, 256
 - 2 identity block, với mỗi khối sử dụng 3 filter: 1x1 conv, 64; 3x3 conv, 64; 1x1 conv, 256
- Stage 3: có 12 layer
 - Convolution block (khối tích chập) sử dụng 3 filter: 1x1 conv, 128; 3x3 conv, 128; 1x1 conv, 512
 - 3 identity block, với mỗi khối sử dụng 3 filter: 1x1 conv, 128; 3x3 conv, 128; 1x1 conv, 512
- Stage 4: có 18 layer
 - Convolution block (khối tích chập) sử dụng 3 filter: 1x1 conv, 256; 3x3 conv, 256; 1x1 conv, 1024
 - 5 identity block, với mỗi khối sử dụng 3 filter: 1x1 conv, 256; 3x3 conv, 256; 1x1 conv, 1024
- Stage 5: 9 layer
 - Convolution block (khối tích chập) sử dụng 3 filter: 1x1 conv, 512; 3x3 conv, 512; 1x1 conv, 2048
 - 2 identity block, với mỗi khối sử dụng 3 filter: 1x1 conv, 512; 3x3 conv, 512; 1x1 conv, 2048
- AveragePooling, Flatten, lớp kết nối đầy đủ (fully connected): có 1 layer

2.6. Kết luận chương:

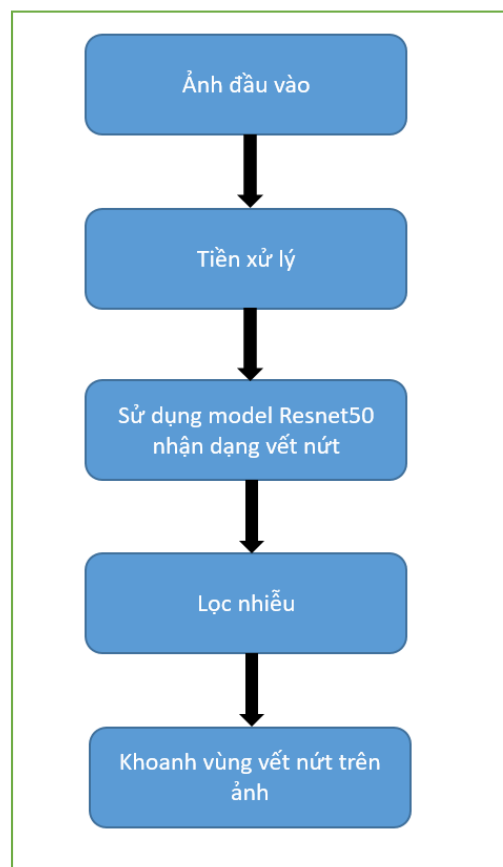
Chương 2 đã nêu rõ được về cấu trúc mạng nơ-ron tích chập và các thành phần có trong đó và chức năng chính của từng phân lớp. Bên cạnh đó chương này cũng đã giới thiệu về mô hình mạng ResNet, đặc biệt là mô hình và kiến trúc mạng ResNet-50.

CHƯƠNG 3: THIẾT KẾ VÀ THỰC HIỆN HỆ THỐNG

3.1. Giới thiệu chương 3:

Dựa trên các kiến thức đã tìm hiểu và được học, trong chương 3 này chúng tôi sẽ trình bày các bước xây dựng cơ sở dữ liệu, quá trình huấn luyện mô hình và áp dụng trên thực tế.

3.2. Sơ đồ hệ thống:



Hình 3.1. Sơ đồ hệ thống

3.3. Tạo dữ liệu huấn luyện:

Để xây dựng một hệ thống phát hiện vết nứt mặt tường cần có bộ dữ liệu lớn và đa dạng về các vết nứt và vết không nứt có trên bề mặt tường. Ở đây chúng tôi sử dụng bộ dữ liệu CCIC được lấy từ trang web Mendeley Data - Crack Detection, do Çağlar Fırat Özgenel đóng góp. Bộ dữ liệu chứa hình ảnh của các bề tường khác nhau có và không có vết nứt. Dữ

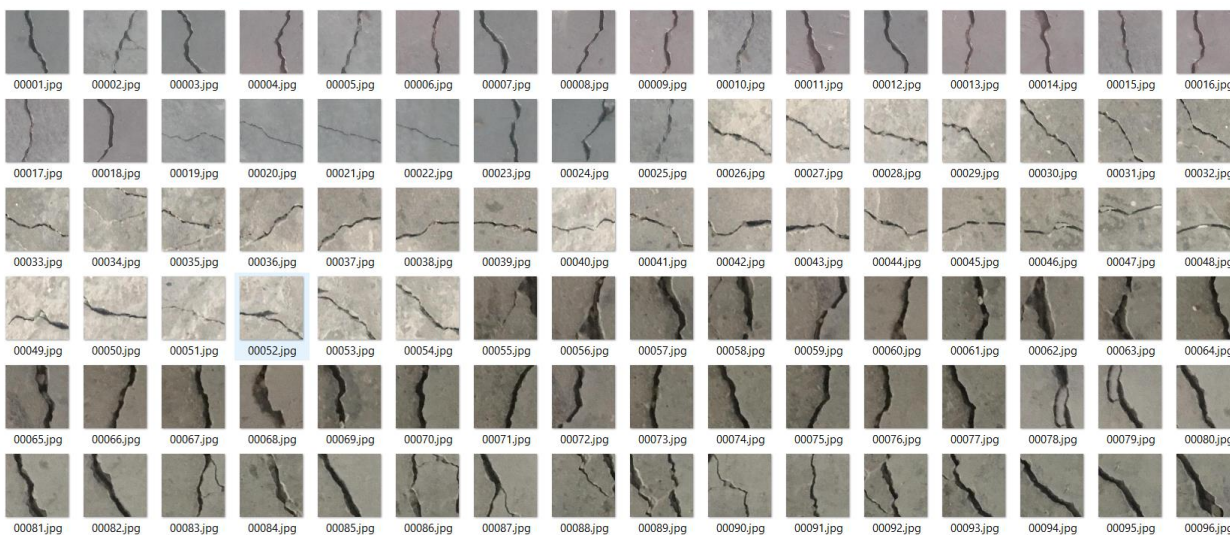
liệu hình ảnh được chia thành hai dạng âm (không có vết nứt) và dương (có vết nứt) trong thư mục riêng biệt để phân loại ảnh. Mỗi lớp có 20000 hình ảnh với tổng số 40000 hình ảnh với 227 x 227 pixel với các kênh RGB.

- Dữ liệu ảnh không nứt:



Hình 3.2. Dữ liệu ảnh không có vết nứt

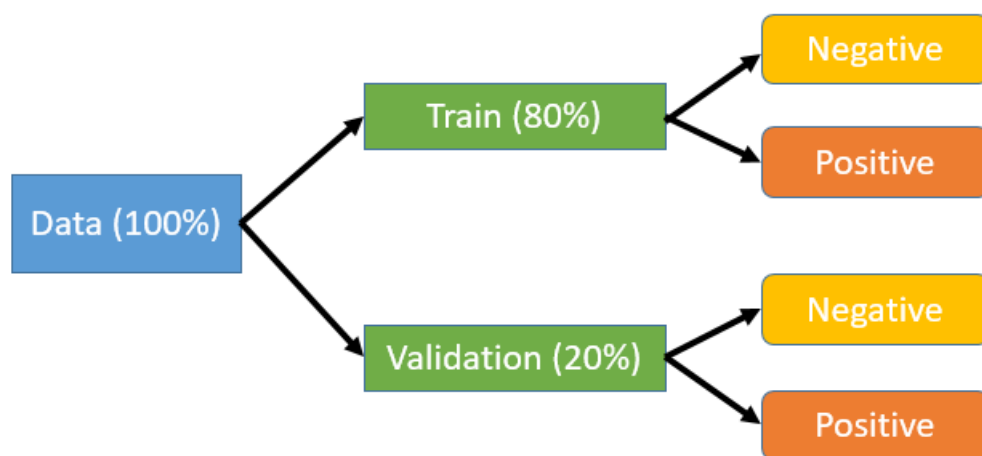
- Dữ liệu ảnh có nứt:



Hình 3.3. Dữ liệu ảnh có vết nứt

3.4. Tạo cơ sở dữ liệu:

Sau khi có được dữ liệu mẫu gồm 2 thư mục Negative (ảnh không nứt) và Positive (ảnh nứt) . Chúng tôi tiến hành chia bộ dữ liệu như hình vẽ:



Hình 3.4. Phân chia dữ liệu

Với 10.000 ảnh Negative mẫu ta chia thành 8.000 ảnh để huấn luyện và 2.000 ảnh để đánh giá mô hình, tương tự với 10.000 ảnh Positive mẫu ta cũng chia thành 8000 ảnh để huấn luyện và 2.000 ảnh để đánh giá mô hình.

3.5. Xây dựng và chọn lựa mô hình học máy:

3.5.1. Mô hình ResNet-50:

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 32, 32]	9,408
BatchNorm2d-2	[-1, 64, 32, 32]	128
ReLU-3	[-1, 64, 32, 32]	0
MaxPool2d-4	[-1, 64, 16, 16]	0
Conv2d-5	[-1, 64, 16, 16]	4,096
BatchNorm2d-6	[-1, 64, 16, 16]	128
ReLU-7	[-1, 64, 16, 16]	0
Conv2d-8	[-1, 64, 16, 16]	36,864
BatchNorm2d-9	[-1, 64, 16, 16]	128
ReLU-10	[-1, 64, 16, 16]	0
Conv2d-11	[-1, 256, 16, 16]	16,384
BatchNorm2d-12	[-1, 256, 16, 16]	512
Conv2d-13	[-1, 256, 16, 16]	16,384
BatchNorm2d-14	[-1, 256, 16, 16]	512
ReLU-15	[-1, 256, 16, 16]	0
Bottleneck-16	[-1, 256, 16, 16]	0
Conv2d-17	[-1, 64, 16, 16]	16,384
BatchNorm2d-18	[-1, 64, 16, 16]	128
ReLU-19	[-1, 64, 16, 16]	0
Conv2d-20	[-1, 64, 16, 16]	36,864
BatchNorm2d-21	[-1, 64, 16, 16]	128
ReLU-22	[-1, 64, 16, 16]	0
Conv2d-23	[-1, 256, 16, 16]	16,384
BatchNorm2d-24	[-1, 256, 16, 16]	512
ReLU-25	[-1, 256, 16, 16]	0
Bottleneck-26	[-1, 256, 16, 16]	0
Conv2d-27	[-1, 64, 16, 16]	16,384
BatchNorm2d-28	[-1, 64, 16, 16]	128
ReLU-29	[-1, 64, 16, 16]	0
Conv2d-30	[-1, 64, 16, 16]	36,864
BatchNorm2d-31	[-1, 64, 16, 16]	128
ReLU-32	[-1, 64, 16, 16]	0
Conv2d-33	[-1, 256, 16, 16]	16,384
BatchNorm2d-34	[-1, 256, 16, 16]	512
ReLU-35	[-1, 256, 16, 16]	0
Bottleneck-36	[-1, 256, 16, 16]	0
Conv2d-37	[-1, 128, 16, 16]	32,768
BatchNorm2d-38	[-1, 128, 16, 16]	256
ReLU-39	[-1, 128, 16, 16]	0

Conv2d-40	[-1, 128, 8, 8]	147,456
BatchNorm2d-41	[-1, 128, 8, 8]	256
ReLU-42	[-1, 128, 8, 8]	0
Conv2d-43	[-1, 512, 8, 8]	65,536
BatchNorm2d-44	[-1, 512, 8, 8]	1,024
Conv2d-45	[-1, 512, 8, 8]	131,072
BatchNorm2d-46	[-1, 512, 8, 8]	1,024
ReLU-47	[-1, 512, 8, 8]	0
Bottleneck-48	[-1, 512, 8, 8]	0
Conv2d-49	[-1, 128, 8, 8]	65,536
BatchNorm2d-50	[-1, 128, 8, 8]	256
ReLU-51	[-1, 128, 8, 8]	0
Conv2d-52	[-1, 128, 8, 8]	147,456
BatchNorm2d-53	[-1, 128, 8, 8]	256
ReLU-54	[-1, 128, 8, 8]	0
Conv2d-55	[-1, 512, 8, 8]	65,536
BatchNorm2d-56	[-1, 512, 8, 8]	1,024
ReLU-57	[-1, 512, 8, 8]	0
Bottleneck-58	[-1, 512, 8, 8]	0
Conv2d-59	[-1, 128, 8, 8]	65,536
BatchNorm2d-60	[-1, 128, 8, 8]	256
ReLU-61	[-1, 128, 8, 8]	0
Conv2d-62	[-1, 128, 8, 8]	147,456
BatchNorm2d-63	[-1, 128, 8, 8]	256
ReLU-64	[-1, 128, 8, 8]	0
Conv2d-65	[-1, 512, 8, 8]	65,536
BatchNorm2d-66	[-1, 512, 8, 8]	1,024
ReLU-67	[-1, 512, 8, 8]	0
Bottleneck-68	[-1, 512, 8, 8]	0
Conv2d-69	[-1, 128, 8, 8]	65,536
BatchNorm2d-70	[-1, 128, 8, 8]	256
ReLU-71	[-1, 128, 8, 8]	0
Conv2d-72	[-1, 128, 8, 8]	147,456
BatchNorm2d-73	[-1, 128, 8, 8]	256
ReLU-74	[-1, 128, 8, 8]	0
Conv2d-75	[-1, 512, 8, 8]	65,536
BatchNorm2d-76	[-1, 512, 8, 8]	1,024
ReLU-77	[-1, 512, 8, 8]	0
Bottleneck-78	[-1, 512, 8, 8]	0
Conv2d-79	[-1, 256, 8, 8]	131,072
BatchNorm2d-80	[-1, 256, 8, 8]	512
ReLU-81	[-1, 256, 8, 8]	0
Conv2d-82	[-1, 256, 4, 4]	589,824
BatchNorm2d-83	[-1, 256, 4, 4]	512
ReLU-84	[-1, 256, 4, 4]	0

Conv2d-85	[-1, 1024, 4, 4]	262,144
BatchNorm2d-86	[-1, 1024, 4, 4]	2,048
Conv2d-87	[-1, 1024, 4, 4]	524,288
BatchNorm2d-88	[-1, 1024, 4, 4]	2,048
ReLU-89	[-1, 1024, 4, 4]	0
Bottleneck-90	[-1, 1024, 4, 4]	0
Conv2d-91	[-1, 256, 4, 4]	262,144
BatchNorm2d-92	[-1, 256, 4, 4]	512
ReLU-93	[-1, 256, 4, 4]	0
Conv2d-94	[-1, 256, 4, 4]	589,824
BatchNorm2d-95	[-1, 256, 4, 4]	512
ReLU-96	[-1, 256, 4, 4]	0
Conv2d-97	[-1, 1024, 4, 4]	262,144
BatchNorm2d-98	[-1, 1024, 4, 4]	2,048
ReLU-99	[-1, 1024, 4, 4]	0
Bottleneck-100	[-1, 1024, 4, 4]	0
Conv2d-101	[-1, 256, 4, 4]	262,144
BatchNorm2d-102	[-1, 256, 4, 4]	512
ReLU-103	[-1, 256, 4, 4]	0
Conv2d-104	[-1, 256, 4, 4]	589,824
BatchNorm2d-105	[-1, 256, 4, 4]	512
ReLU-106	[-1, 256, 4, 4]	0
Conv2d-107	[-1, 1024, 4, 4]	262,144
BatchNorm2d-108	[-1, 1024, 4, 4]	2,048
ReLU-109	[-1, 1024, 4, 4]	0
Bottleneck-110	[-1, 1024, 4, 4]	0
Conv2d-111	[-1, 256, 4, 4]	262,144
BatchNorm2d-112	[-1, 256, 4, 4]	512
ReLU-113	[-1, 256, 4, 4]	0
Conv2d-114	[-1, 256, 4, 4]	589,824
BatchNorm2d-115	[-1, 256, 4, 4]	512
ReLU-116	[-1, 256, 4, 4]	0
Conv2d-117	[-1, 1024, 4, 4]	262,144
BatchNorm2d-118	[-1, 1024, 4, 4]	2,048
ReLU-119	[-1, 1024, 4, 4]	0
Bottleneck-120	[-1, 1024, 4, 4]	0
Conv2d-121	[-1, 256, 4, 4]	262,144
BatchNorm2d-122	[-1, 256, 4, 4]	512
ReLU-123	[-1, 256, 4, 4]	0
Conv2d-124	[-1, 256, 4, 4]	589,824
BatchNorm2d-125	[-1, 256, 4, 4]	512
ReLU-126	[-1, 256, 4, 4]	0

Conv2d-127	[-1, 1024, 4, 4]	262,144
BatchNorm2d-128	[-1, 1024, 4, 4]	2,048
ReLU-129	[-1, 1024, 4, 4]	0
Bottleneck-130	[-1, 1024, 4, 4]	0
Conv2d-131	[-1, 256, 4, 4]	262,144
BatchNorm2d-132	[-1, 256, 4, 4]	512
ReLU-133	[-1, 256, 4, 4]	0
Conv2d-134	[-1, 256, 4, 4]	589,824
BatchNorm2d-135	[-1, 256, 4, 4]	512
ReLU-136	[-1, 256, 4, 4]	0
Conv2d-137	[-1, 1024, 4, 4]	262,144
BatchNorm2d-138	[-1, 1024, 4, 4]	2,048
ReLU-139	[-1, 1024, 4, 4]	0
Bottleneck-140	[-1, 1024, 4, 4]	0
Conv2d-141	[-1, 512, 4, 4]	524,288
BatchNorm2d-142	[-1, 512, 4, 4]	1,024
ReLU-143	[-1, 512, 4, 4]	0
Conv2d-144	[-1, 512, 2, 2]	2,359,296
BatchNorm2d-145	[-1, 512, 2, 2]	1,024
ReLU-146	[-1, 512, 2, 2]	0
Conv2d-147	[-1, 2048, 2, 2]	1,048,576
BatchNorm2d-148	[-1, 2048, 2, 2]	4,096
Conv2d-149	[-1, 2048, 2, 2]	2,097,152
BatchNorm2d-150	[-1, 2048, 2, 2]	4,096
ReLU-151	[-1, 2048, 2, 2]	0
Bottleneck-152	[-1, 2048, 2, 2]	0
Conv2d-153	[-1, 512, 2, 2]	1,048,576
BatchNorm2d-154	[-1, 512, 2, 2]	1,024
ReLU-155	[-1, 512, 2, 2]	0
Conv2d-156	[-1, 512, 2, 2]	2,359,296
BatchNorm2d-157	[-1, 512, 2, 2]	1,024
ReLU-158	[-1, 512, 2, 2]	0
Conv2d-159	[-1, 2048, 2, 2]	1,048,576
BatchNorm2d-160	[-1, 2048, 2, 2]	4,096
ReLU-161	[-1, 2048, 2, 2]	0
Bottleneck-162	[-1, 2048, 2, 2]	0
Conv2d-163	[-1, 512, 2, 2]	1,048,576
BatchNorm2d-164	[-1, 512, 2, 2]	1,024
ReLU-165	[-1, 512, 2, 2]	0
Conv2d-166	[-1, 512, 2, 2]	2,359,296
BatchNorm2d-167	[-1, 512, 2, 2]	1,024
ReLU-168	[-1, 512, 2, 2]	0
Conv2d-169	[-1, 2048, 2, 2]	1,048,576
BatchNorm2d-170	[-1, 2048, 2, 2]	4,096
ReLU-171	[-1, 2048, 2, 2]	0
Bottleneck-172	[-1, 2048, 2, 2]	0
AdaptiveAvgPool2d-173	[-1, 2048, 1, 1]	0

Linear-174	[-1, 128]	262,272
ReLU-175	[-1, 128]	0
Dropout-176	[-1, 128]	0
Linear-177	[-1, 2]	258

```

=====
Total params: 23,770,562
Trainable params: 262,530
Non-trainable params: 23,508,032
-----
Input size (MB): 0.05
Forward/backward pass size (MB): 23.41
Params size (MB): 90.68
Estimated Total Size (MB): 114.13
-----

```

Hình 3.5. Kiến trúc lớp Convolutional Layers

3.5.2. Quá trình huấn luyện:

Huấn luyện mô hình là quá trình tiếp theo khi đã thu thập đủ lượng dữ liệu cần có. Việc huấn luyện yêu cầu máy phải sử dụng bộ xử lý GPU để hoạt động vì thế tôi đã quyết định sử dụng Google Colab để thực hiện huấn luyện cho mô hình.

Google Colab là một sản phẩm từ Google Research, nó cho phép chạy các dòng code python thông qua trình duyệt, đặc biệt phù hợp với Data analysis, Machine learning và giáo dục. Colab không cần yêu cầu cài đặt hay cấu hình máy tính, mọi thứ có thể chạy thông qua trình duyệt, bạn có thể sử dụng tài nguyên máy tính từ CPU tốc độ cao và cả GPUs và cả TPUs đều được cung cấp cho người dùng.

Colab cung cấp nhiều loại GPU, thường là Nvidia K80s, T4s, P4s and P100s, tuy nhiên người dùng không thể chọn loại GPU trong Colab, GPU trong Colab thay đổi theo thời gian. Vì là dịch vụ miễn phí, nên Colab sẽ có những thứ tự ưu tiên trong việc sử dụng tài nguyên hệ thống, cũng như giới hạn thời gian sử dụng, thời gian sử dụng tối đa lên tới 12 giờ.

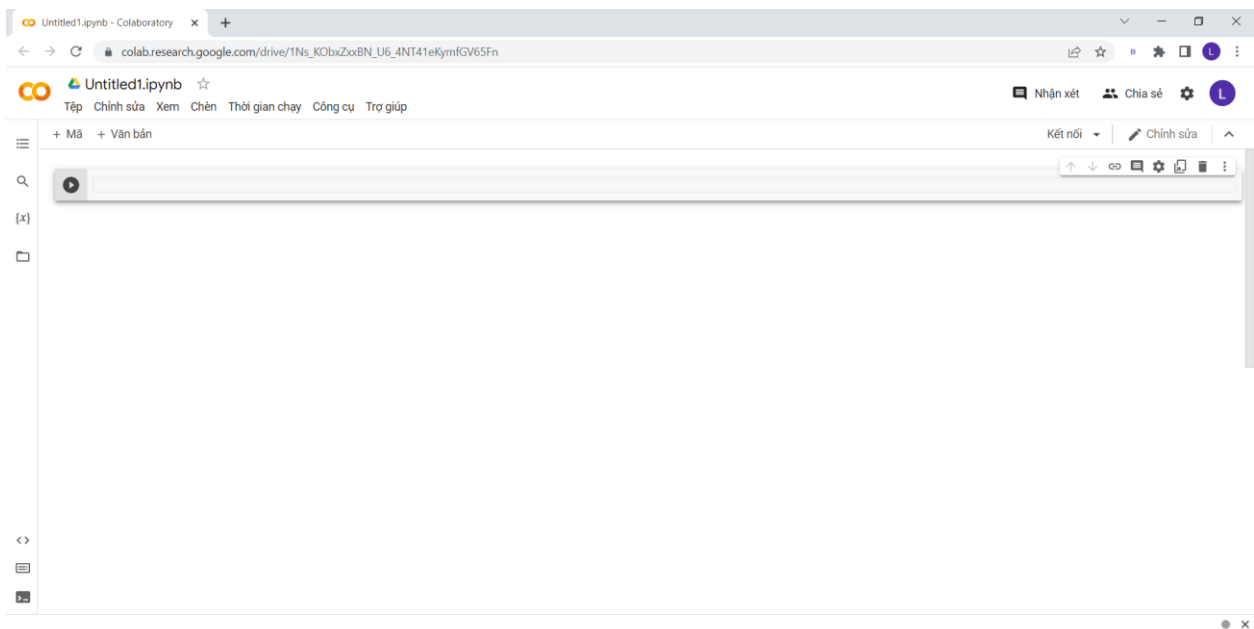
Một số thông số của Google Colab:

- Hệ điều hành: Linux
- RAM GPU: 12GB GDDR5 VRAM, hỗ trợ CUDA
- Bộ nhớ: 359GB
- Thời gian sử dụng: 12 giờ

- Ưu điểm của Colab:

- Cấu hình chip mạnh mẽ
- RAM lớn: 12GB
- GPU lớn, cung cấp 12 - 13 GB VRAM, có hỗ trợ CUDA

- Nhược điểm: Đây là môi trường ảo do Google cung cấp để thực hành nên sẽ khó khăn cho các bài toán lớn. Trên thực tế Colab giới hạn sau 12 giờ sẽ tự động ngắt các kết nối và dừng các hoạt động của hệ thống và xóa toàn bộ dữ liệu.



Hình 3.6. Môi trường làm việc trên Colab

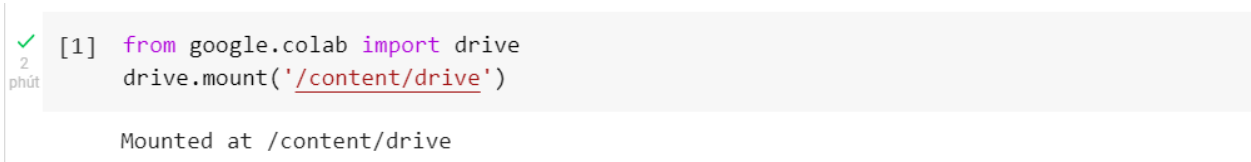
Quá trình huấn luyện mô hình mạng ResNet-50 được thực hiện qua các bước:

- Bước 1: Kết nối GoogleColab với Drive
- Bước 2: Tải dữ liệu chuẩn bị huấn luyện và đánh giá lên Drive
- Bước 3: Cài các thư viện cần thiết
- Bước 4: Đọc dữ liệu huấn luyện và đánh giá từ Google Drive
- Bước 5: Tăng cường và chuyển đổi dữ liệu
- Bước 6: Tải file Pretrained model
- Bước 7: Thực hiện huấn luyện

- Bước 8: Kết thúc quá trình huấn luyện và tải file weights về máy tính

3.5.2.1. Kết nối GoogleColab với Drive:

Bởi vì khi thực hiện trên Colab các file tải lên hay là file hình thành sẽ bị mất đi khi quá trình hoạt động dừng lại hay kết thúc. Để thuận tiện cho việc lưu trữ và giữ thông tin một cách an toàn và có thể sử dụng một cách tốt nhất thì ta sẽ tải lên Drive sau đó kết nối Drive với Colab để có thể sử dụng được tất cả các file có trong Drive.



```
[1] from google.colab import drive
    drive.mount('/content/drive')

Mounted at /content/drive
```

Hình 3.7. Dòng lệnh kết nối với driver

Sau khi đã kết nối với Drive thì ta có thể dễ dàng nhìn thấy các file có trong Drive và có thể mở chúng hay là sử dụng chúng.

3.5.2.2. Tải dữ liệu chuẩn bị huấn luyện và đánh giá lên Drive:

Tiến hành tải bộ dữ liệu đã chuẩn bị lên Google Drive đã kết nối với Google Colab



Hình 3.8. Dữ liệu sau khi tải lên driver

3.5.2.3. Cài các thư viện cần thiết:

Tiến hành cài các thư viện hỗ trợ sử dụng trong dự án

```
0 giây
import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torch.utils.data import sampler
from torch.optim import lr_scheduler
from torchvision import datasets, models, transforms
import numpy as np
import os
from PIL import Image
import time
import copy
import random
import cv2
import re
import shutil
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

Hình 3.9. Các thư viện sử dụng trong dự án

3.5.2.4. Đọc dữ liệu huấn luyện và đánh giá từ Google Drive:

```
positive_train = "/content/drive/MyDrive/colab_doan/Dataset/train/Positive"
positive_val = "/content/drive/MyDrive/colab_doan/Dataset/val/Positive"
negative_train = "/content/drive/MyDrive/colab_doan/Dataset/train/Negative"
negative_val = "/content/drive/MyDrive/colab_doan/Dataset/val/Negative"
```

Hình 3.10. Đường dẫn tới dữ liệu

3.5.2.5. Tăng cường và chuyển đổi dữ liệu:

```
[ ] ## Define data augmentation and transforms
chosen_transforms = {'train': transforms.Compose([
    transforms.RandomResizedCrop(size=227),
    transforms.RandomRotation(degrees=10),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.ColorJitter(brightness=0.15, contrast=0.15),
    transforms.ToTensor(),
    transforms.Normalize(mean_nums, std_nums)
]), 'val': transforms.Compose([
    transforms.Resize(227),
    transforms.CenterCrop(227),
    transforms.ToTensor(),
    transforms.Normalize(mean_nums, std_nums)
])
}
```

Hình 3.11. Tăng cường và chuyển đổi dữ liệu

3.5.2.6. Tải file Pretrained model:

```
## Load pretrained model
# residual networks. These are a special type of neural networks in contrast to fully connected networks.
# this solves the problem of vanishing gradients.
resnet50 = models.resnet50(pretrained=True)

# Freeze model parameters
for param in resnet50.parameters():
    param.requires_grad = False

## Change the final layer of the resnet model
# Change the final layer of ResNet50 Model for Transfer Learning
fc_inputs = resnet50.fc.in_features

resnet50.fc = nn.Sequential(
    nn.Linear(fc_inputs, 128),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(128, 2)
)

# Linear layer is an alias for fully connected layer.
# Convert model to be used on GPU
resnet50 = resnet50.to(device)

from torchsummary import summary
print(summary(resnet50, (3, 227, 227)))
```

Hình 3.12. Tải file Pretrained model

3.5.2.7. Huấn luyện và lưu model:

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=10):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                scheduler.step()
                model.train() # Set model to training mode
            else:
                model.eval() # Set model to evaluate mode

            current_loss = 0.0
            current_corrects = 0

            # Here's where the training happens
            print('Iterating through data...')

            # training starts...
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # We need to zero the gradients, don't forget it
                optimizer.zero_grad()

                # Time to carry out the forward training pass
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    # this computes the class that the model belongs to...
                    _, preds = torch.max(outputs, 1)
                    # compute the loss here...
                    # we compute the difference between true label and the predicted output...
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward() # this is the back-propagation step. we
                        # minimize our model loss and optimize model in this phase
                        optimizer.step()
                        # optimizer updates the model parameters...

                # We want variables to hold the loss statistics
                # combine our model losses and compute the wrongly classified examples
                current_loss += loss.item() * inputs.size(0)
                current_corrects += torch.sum(preds == labels.data)

            # compute the loss and accuracy for each of our epoch
            epoch_loss = current_loss / dataset_sizes[phase]
            epoch_acc = current_corrects.double() / dataset_sizes[phase]

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # Make a copy of the model if the accuracy on the validation set has improved
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()
```

```
base_model = train_model(resnet50, criterion, optimizer , exp_lr_scheduler, num_epochs=6)
visualize_model(base_model)
plt.show()
torch.save(base_model, '/content/drive/MyDrive/colab_doan/model/model_resnet50')
# an epoch is the total time during which one training step completes. In other words, it is the total
# time that our model takes to process the input data once...
```

Hình 3.13. Code huấn luyện model

Các tham số sử dụng trong quá trình train:

- Epochs: 6
- Batch size: 256

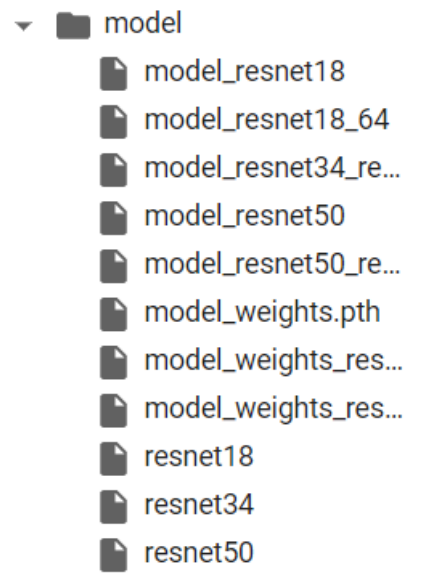
3.5.2.8. Kết thúc huấn luyện:

Ta nhận được kết quả huấn luyện model ResNet-50

```
Training complete in 79m 41s
Best val Acc: 0.964357
```

Hình 3.14. Kết quả huấn luyện

và model sau quá trình huấn luyện



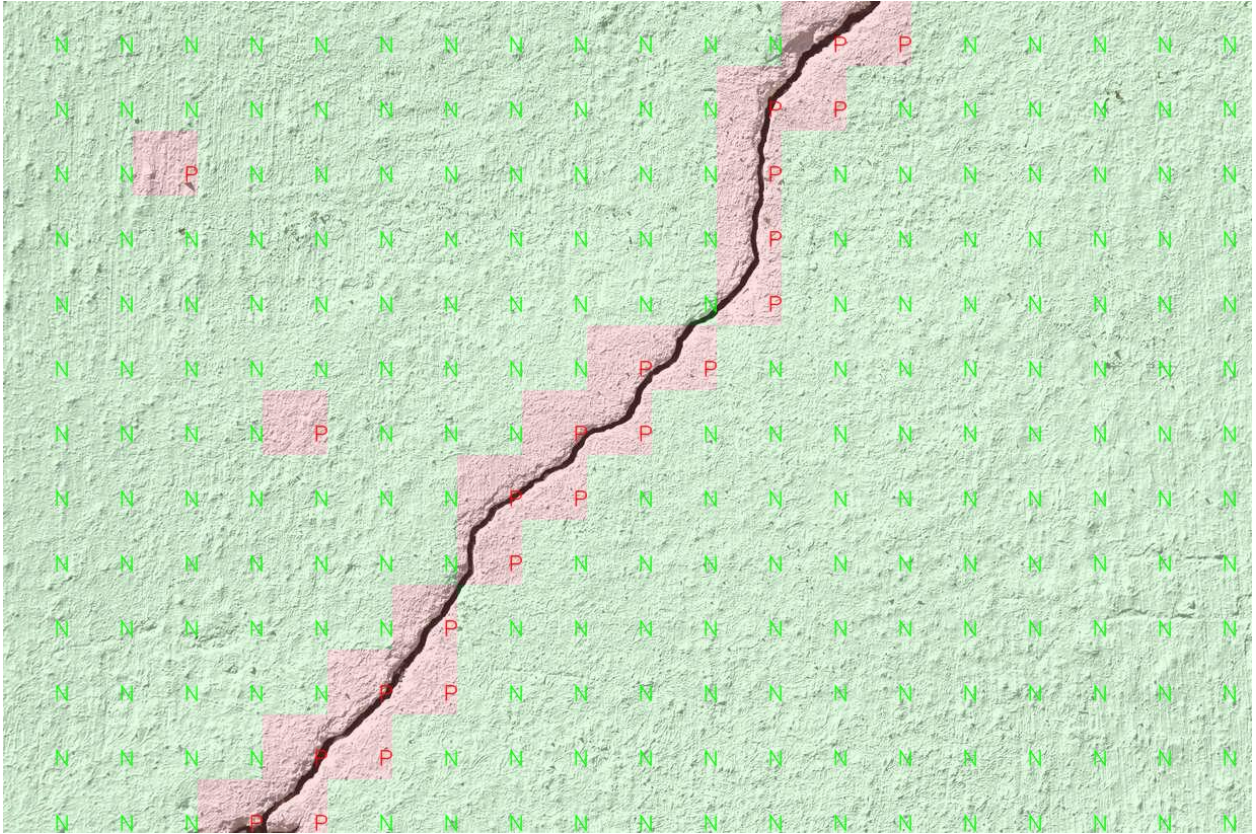
Hình 3.15. Model được lưu sau khi huấn luyện

3.6. Học nhiễu

+ Sử dụng thuật toán Connected – Component

+ Cách bước thực hiện:

+ Sau khi có kết quả từ quá trình đánh giá ảnh với model đã huấn luyện ta tạo ma trận 0 bằng với số ô ta xét trên hình. Gán các ô mà mô hình đánh giá là có vết nứt bằng 255.



Hình 3.16. Ảnh sau khi đánh giá với mô hình chưa qua Connected Component

```

[[ 0  0  0  0  0  0  0  0  0  0  0  0 255 255  0  0  0  0
  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 255 255  0  0  0  0  0
  0  0]
 [ 0  0 255  0  0  0  0  0  0  0  0 255  0  0  0  0  0  0
  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 255  0  0  0  0  0  0
  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 255  0  0  0  0  0  0
  0  0]
 [ 0  0  0  0  0  0  0  0  0 255 255  0  0  0  0  0  0  0
  0  0]
 [ 0  0  0  0 255  0  0  0 255 255  0  0  0  0  0  0  0  0
  0  0]
 [ 0  0  0  0  0  0  0 255 255  0  0  0  0  0  0  0  0  0
  0  0]
 [ 0  0  0  0  0  0  0 255  0  0  0  0  0  0  0  0  0  0
  0  0]
 [ 0  0  0  0  0  0 255  0  0  0  0  0  0  0  0  0  0  0
  0  0]
 [ 0  0  0  0  0 255 255  0  0  0  0  0  0  0  0  0  0  0
  0  0]
 [ 0  0  0  0 255 255  0  0  0  0  0  0  0  0  0  0  0  0
  0  0]
 [ 0  0  0 255 255  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0]]

```

Hình 3.17. Ma trận mô phỏng lại vị trí vết nứt

+ Sử dụng `connectedComponentsWithStats` kết quả trả về của `Statistics` bao gồm: Vị trí bắt đầu, vị trí kết thúc và diện tích của các vùng riêng biệt

```

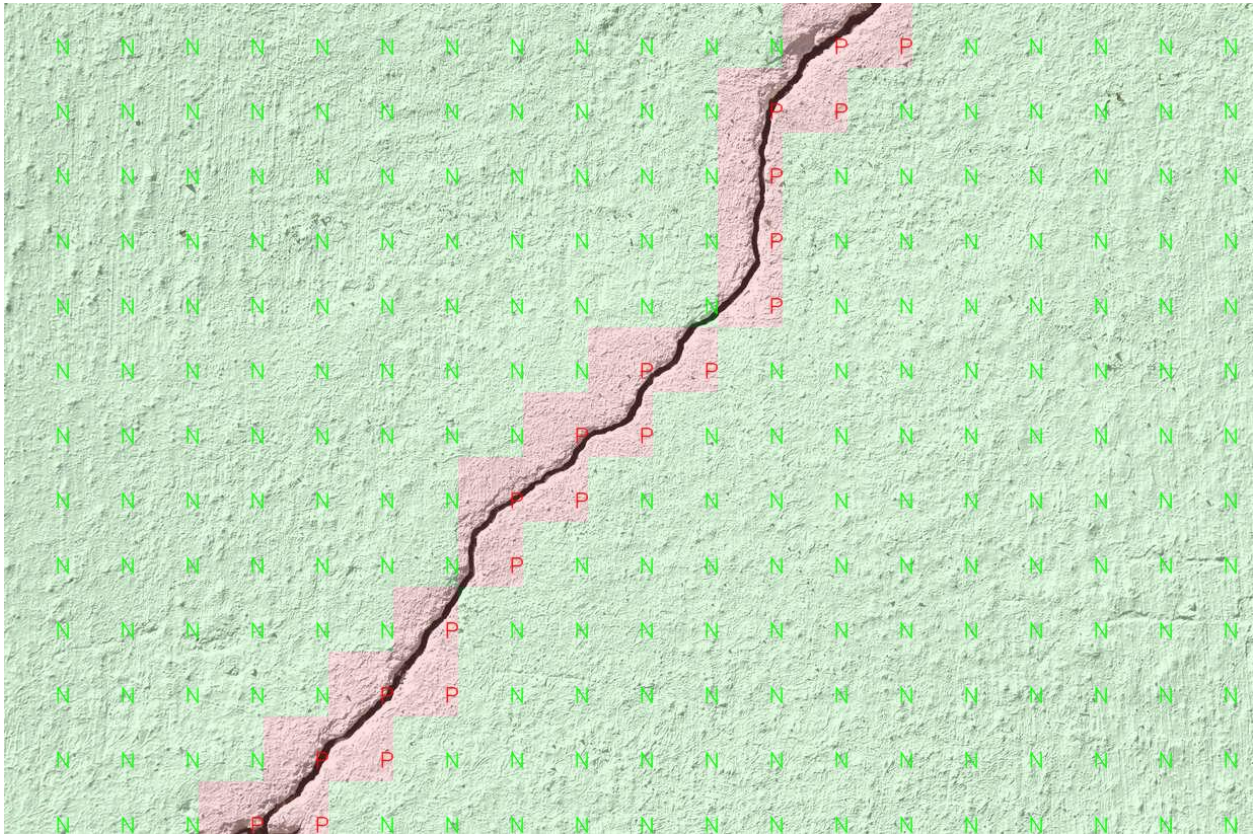
[[ 0  0 20 13 237]
 [ 3  0 11 13 21]
 [ 2  2  1  1  1]
 [ 4  6  1  1  1]]

```

Hình 3.18. Kết quả trả về của `Statistics` (`connectedComponentsWithStats`)

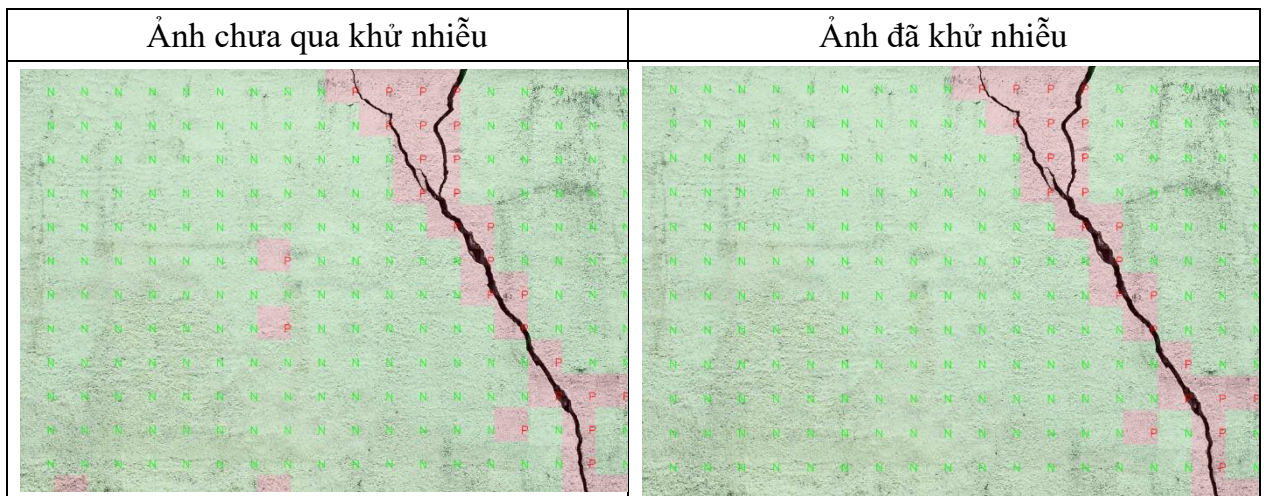
+ Chọn diện tích thích hợp để loại bỏ vùng nhiễu

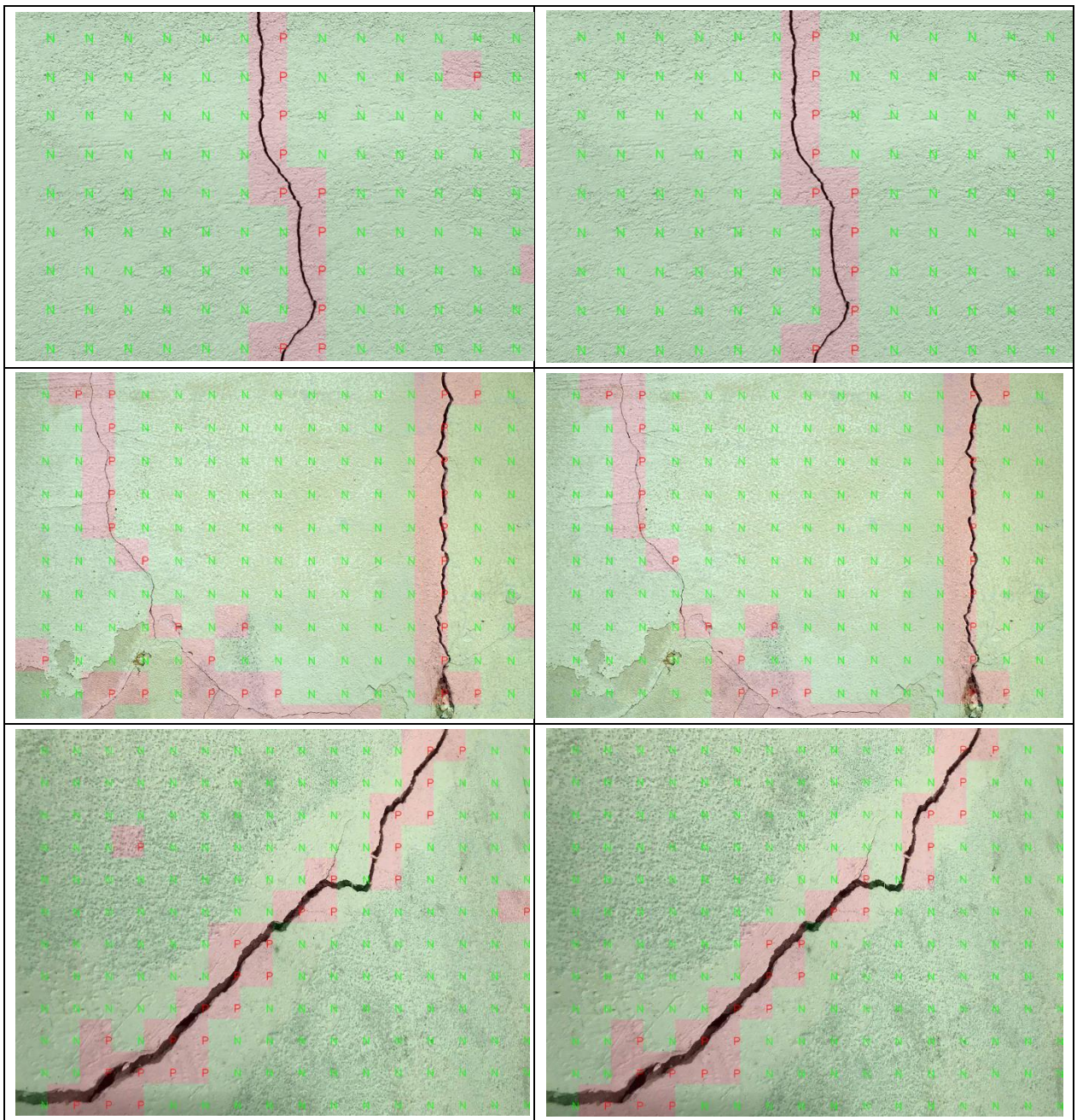
+ Dựa trên ma trận vẽ lại ảnh vết nứt

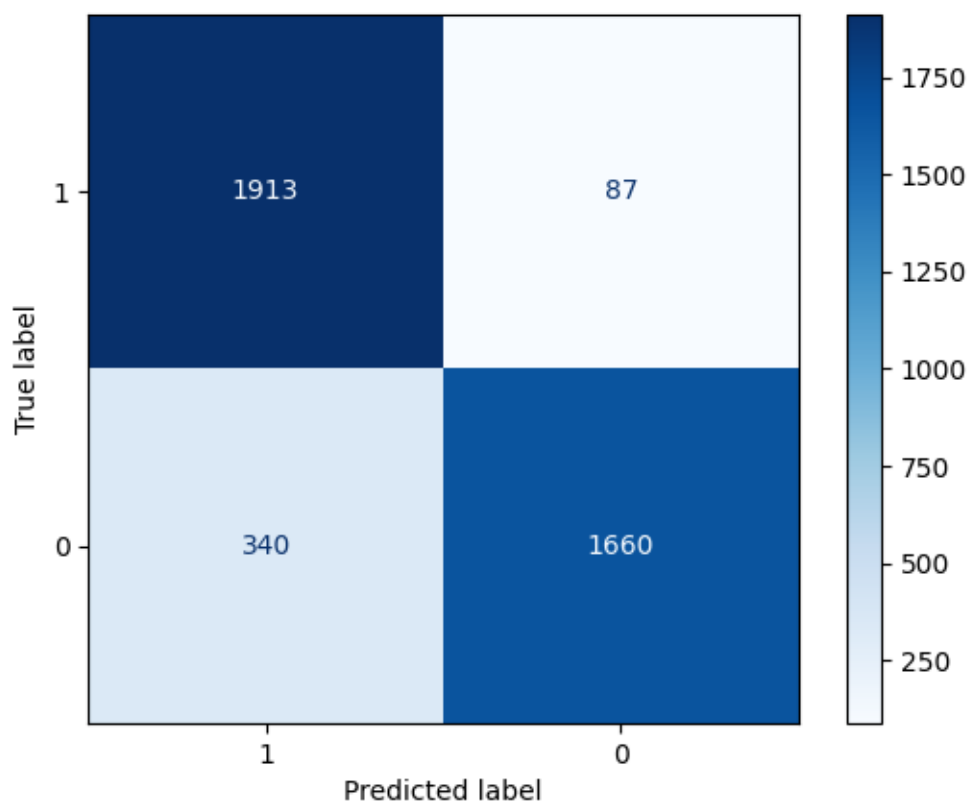


Hình 3.19. Kết quả sau khi khử nhiễu

3.7. Kết quả thu được khi thử trên nhiều hình khác:







Hình 3.20. Ma trận nhầm lẫn

Dựa vào ma trận nhầm lẫn cho ta thấy rằng độ chính xác phân bố theo đường chéo nên mô hình dự đoán khá tốt, đa số mô hình dự đoán được các hành động trong các nhãn, một số hành động nhận dạng nhầm lẫn, nhưng tổng quan về độ chính xác khá ổn.

Độ nhận dạng trung bình: 89.3%

3.9. Kết luận chương 3:

Trong chương 3, đã trình bày cụ thể các bước của quá trình huấn luyện từ cơ bản nhất là chuẩn bị cơ sở dữ liệu cho đến các bước cuối cùng là thực hiện huấn luyện trên Colab và tải mô hình được huấn luyện về máy tính; cũng như việc áp dụng mô hình và xử lý ảnh vào thực tế.

TÀI LIỆU THAM KHẢO

- [1]. [The Difference Between AI, Machine Learning, and Deep Learning? | NVIDIA Blog](#)
- [2]. [Take you into the “past life and this life” of neural network | Develop Paper](#)
- [3]. Ying, X. An Overview of Overfitting and its Solutions. J. Phys. Conf. Ser. 2019, 1168, 022022
- [4]. Dorafshan, S.; Thomas, R.J.; Maguire, M. Comparison of deep convolutional neural networks and edge detectors for image-based crack detection in concrete. Construct. Build. Mater. 2018, 186, 1031–1045
- [5]. [\[1908.04392\] Deep Learning for Detecting Building Defects Using Convolutional Neural Networks \(arxiv.org\)](#)
- [6]. [Sensors | Free Full-Text | Performance Evaluation of Deep CNN-Based Crack Detection and Localization Techniques for Concrete Structures \(mdpi.com\)](#)
- [7]. [Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network | upGrad blog](#)
- [8]. [Convolutional Neural Network Architecture | CNN Architecture \(analyticsvidhya.com\)](#)
- [9]. [1D Convolution \(peltarion.com\)](#)
- [10]. [Convolution - Tích chập giải thích bằng code thực tế \(techmaster.vn\)](#)
- [11]. [3D Convolutions : Understanding + Use Case | Kaggle](#)
- [12]. [Activation Functions in Neural Networks | by SAGAR SHARMA | Towards Data Science](#)
- [13]. [CNN | Introduction to Pooling Layer - GeeksforGeeks](#)
- [14]. Dropout: A Simple Way to Prevent Neural Networks from Overfitting - Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov
- [15]. [Understand Cross Entropy Loss in Minutes | by Uniqtech | Data Science Bootcamp | Medium](#)
- [16], [17]. [Understanding ResNet50 architecture \(opengenius.org\)](#)
- [18]. [Khoa học dữ liệu \(phamdinhhkhanh.github.io\)](#)

[19]. [Understanding and Coding a ResNet in Keras | by Priya Dwivedi | Towards Data Science](#)