

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**  
**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**



**BÁO CÁO BÀI TẬP LỚN**  
**HỌC PHẦN TRÍ TUỆ NHÂN TẠO**  
**ĐỀ TÀI**  
**ỨNG DỤNG CNN TRONG PHÁT HIỆN VÀ DỰ ĐOÁN**  
**VẾT NÚT TRÊN BỀ MẶT ĐÁ TỰ NHIÊN**

**Giáo viên hướng dẫn: Mai Thanh Hồng**

**Lớp: 2024IT6094003      Khóa: K18**

**Nhóm sinh viên thực hiện: Nhóm 3**

**Mã SV: 2023604008      Họ và tên: Nguyễn Huy Hoàng**

**Mã SV: 2023600954      Họ và tên: Nguyễn Quốc Khánh**

**Mã SV: 2023600730      Họ và tên: Vũ Văn Thanh**

**Mã SV: 2023604859      Họ và tên: Phạm Tất Tụ**

**Hà Nội, năm 2025**

## **LỜI CẢM ƠN**

Trong thời gian làm bài tập lớn, chúng em xin gửi lời cảm ơn chân thành đến cô Mai Thanh Hồng, người đã tận tình hướng dẫn chúng em trong quá trình làm báo cáo, cũng như giúp chúng em định hướng trong việc xác định các vấn đề, yêu cầu và hướng xử lý cho đề tài.

Chúng em cũng xin chân thành cảm ơn các thầy cô giáo Trường Công Nghệ Thông tin và Truyền Thông thuộc Trường Đại Học Công Nghiệp Hà Nội đã dạy dỗ cho chúng em kiến thức về các môn đại cương cũng như các môn chuyên ngành, giúp chúng em có được cơ sở lý thuyết vững vàng và tạo điều kiện giúp đỡ chúng em trong suốt quá trình học tập.

Tuy nhiên, vì thời gian có hạn cũng như trình độ hiểu biết của bản thân còn nhiều hạn chế. Cho nên trong báo cáo không thể tránh khỏi những thiếu sót, chúng em rất mong nhận được những ý kiến đóng góp, chỉ bảo của các thầy, cô giáo trong Khoa, Trường để chúng em có thể rút ra được kinh nghiệm, kiến thức cần thiết để hoàn thiện thêm bài báo cáo và phục vụ cho các công việc sau này.

**Chúng em xin chân thành cảm ơn!**

# PHIẾU HỌC TẬP CÁ NHÂN/NHÓM

## I. Thông tin chung

1. Tên lớp: 20242IT6094003 Khóa:18

2. Họ và tên sinh viên: Phạm Tất Tụ

3. Tên nhóm : Nhóm 3

Họ và tên thành viên trong nhóm:

Họ và tên	Mã sinh viên
Nguyễn Huy Hoàng	2023604008
Nguyễn Quốc Khánh	2023600954
Vũ Văn Thanh	2023600730
Phạm Tất Tụ	2023604859

## II. Nội dung học tập

**1. Tên chủ đề:** Ứng dụng CNN trong phát hiện vết nứt trên bề mặt đá tự nhiên

### 2. Hoạt động của sinh viên

“ *Hoạt động/Nội dung 1:* Tìm hiểu lý thuyết về mạng CNN

Mục tiêu/chuẩn đầu ra: Sinh viên sẽ hiểu được các kiến thức cơ bản về mạng nơ ron tích chập (CNN), bao gồm cấu trúc, nguyên lý hoạt động và các ứng dụng trong nhận diện hình ảnh.

“ *Hoạt động/Nội dung 2:* Thu thập và xử lý data

Mục tiêu/chuẩn đầu ra: Sinh viên sẽ có khả năng thu thập, xử lý và chuẩn hóa dữ liệu hình ảnh crack-rock/non-crack-rock từ các nguồn khác nhau, tạo ra tập dữ liệu sạch và đồng nhất để phục vụ cho quá trình huấn luyện mô hình.

“ *Hoạt động/Nội dung 3:* Huấn luyện , đánh giá mô hình CNN, từ đó đưa vào sử dụng thực tế

Mục tiêu/chuẩn đầu ra: Sinh viên sẽ có khả năng sử dụng các thư viện như TensorFlow hoặc PyTorch, YOLO, Open-CV để xây dựng, huấn luyện, đánh giá mô hình CNN trên tập dữ liệu logo, đồng thời phân tích kết quả để cải thiện mô hình. Từ đó đưa mô hình vào sử dụng thực tế.

### **3. Sản phẩm nghiên cứu**

Một hệ thống nhận diện logo thương hiệu hoàn chỉnh, bao gồm:

- Mô hình CNN đã được huấn luyện và tối ưu hóa.
- Tập dữ liệu đã được chuẩn hóa và gán nhãn cho từng logo thương hiệu.
- Tài liệu báo cáo mô tả quy trình thực hiện, kết quả đánh giá và những khó khăn gặp phải trong quá trình phát triển hệ thống.

### **III. Nhiệm vụ học tập**

1. Hoàn thành Tiểu luận, Bài tập lớn, Đồ án/Dự án theo đúng thời gian quy định (từ ngày 07/04/2025 đến ngày 30/05/2025)
2. Báo cáo sản phẩm nghiên cứu theo chủ đề được giao trước giảng viên và những sinh viên khác

### **IV. Học liệu thực hiện Tiểu luận, Bài tập lớn, Đồ án/Dự án**

#### **1. Tài liệu học tập:**

- [1] Trần Hùng Cường, Nguyễn Phương Nga, Giáo trình Trí tuệ nhân tạo, 2014
- [2]. Joshi, Practeck, Artificial intelligence with python, 2017
- [3]. AI VIET NAM – COURSE 2022 [PDF], Basic CNN - Exercise, 11/12/2022
- [4]. AI VIET NAM – COURSE 2022 Basic CNN - Tutorial, 11/12/2022
- [5]. Afshine Amidi, CS230 - Convolutional Neural Network, 10/2019

#### **2. Phương tiện, nguyên liệu thực hiện Tiểu luận, Bài tập lớn, Đồ án/Dự án (nếu có).....**

# KẾ HOẠCH THỰC HIỆN TIỂU LUẬN, BÀI TẬP LỚN, ĐỒ ÁN/DỰ ÁN

Tên lớp : 20242IT6094003      Khóa :18

Họ và tên sinh viên : Phạm Tất Tụ

Tên nhóm : 3

Tên chủ đề : Ứng dụng CNN trong phát hiện vết nứt trên bề mặt đá tự nhiên

Tuần	Người thực hiện	Nội dung công việc	Phương pháp thực hiện
1	Cả nhóm	Đưa ra lý do chọn đề tài	Dựa vào kiến thức tìm hiểu và hướng dẫn viết BM từ giảng viên
	Nguyễn Huy Hoàng	Viết biên bản cuộc họp	Trao đổi trong nhóm và ghi lại nội dung thảo luận
	Phạm Tất Tụ	- Làm Phiếu hoạt động nhóm cá nhân/nhóm  - Tìm hiểu mục đích của đề tài đề ra phương pháp nghiên cứu	Dựa vào mẫu phiếu được phát và tài liệu tham khảo từ giảng viên
	Nguyễn Quốc Khánh	-Làm phiếu hoạt động nhóm theo tuần  -Tìm hiểu về phạm vi nghiên cứu ,đối tượng nghiên cứu và tiềm năng phát triển	Tham khảo đề tài tương tự và tài liệu môn học

	Vũ Văn Thanh	-Tìm hiểu phạm vi nghiên cứu, đối tượng nghiên cứu và tiềm năng phát triển	Tra cứu sách, internet và tài liệu giảng viên cung cấp
2	Nguyễn Huy Hoàng	-Tổng hợp nội dung ra thành văn bản word -Hỗ trợ nội dung cho các thành viên trong nhóm	Dựa vào nội dung từ các thành viên gửi và chỉnh sửa, định dạng
	Phạm Tất Tụ	-Tìm hiểu ứng dụng thực tế	Tra cứu tài liệu, ví dụ thực tiễn trên internet
	Nguyễn Quốc Khánh	-Tìm hiểu nội dung kiến trúc cơ bản	Học qua slide bài giảng và tài liệu học tập
	Vũ Văn Thanh	-Tìm hiểu tổng quan và khái niệm cơ bản của CNN	Tham khảo sách, internet và tài liệu giảng viên cung cấp
3	Nguyễn Huy Hoàng	Trình bày nội dung về lớp làm phẳng và lớp kết nối hoàn toàn	Tham khảo giáo trình, bài giảng, tìm hiểu thêm từ internet và tài liệu học thuật.
	Phạm Tất Tụ	-Viết Phiếu học tập cá nhân/nhóm -Tìm hiểu về lớp đầu ra và đưa ra kỹ thuật hỗ trợ	Đọc tài liệu học tập, làm việc nhóm để thảo luận và viết phiếu học tập.

	Nguyễn Quốc Khánh	-Viết hoạt động đạt được của tuần 1 vào phiếu Kế hoạch làm việc nhóm -Tìm hiểu về lớp kích hoạt và lớp pooling	Tổng hợp hoạt động nhóm, tìm hiểu tài liệu và ví dụ minh họa về lớp kích hoạt và pooling.
	Vũ Văn Thanh	-Tìm hiểu về lớp đầu vào và lớp tích chập	Tham khảo tài liệu giảng dạy, video minh họa và đọc sách chuyên sâu về mạng nơ-ron tích chập.
4	Nguyễn Huy Hoàng	- tìm hiểu về thuật toán argmax	Tự nghiên cứu tài liệu, ghi chú nội dung chính
	Phạm Tất Tụ	-tìm hiểu về thuật toán argmax -Viết phiếu học tập cá nhân nhóm	Tự học qua tài liệu, trao đổi với nhóm
	Nguyễn Quốc Khánh	-tìm hiểu về thuật toán softmax -Viết hoạt động đạt được của tuần 1 vào phiếu Kế hoạch làm việc nhóm	Tìm tài liệu tham khảo, thảo luận nhóm, tổng hợp vào phiếu
	Vũ Văn Thanh	-tìm hiểu về thuật toán softmax	Tự học, chia sẻ kết quả trong nhóm

5	Nguyễn Huy Hoàng	-hỗ trợ trình bày báo cáo	sử dụng word để định dạng báo cáo
	Phạm Tất Tụ	-Viết phiếu học tập cá nhân nhóm	Dựa trên nhật ký hoạt động nhóm và phản hồi của các thành viên
	Nguyễn Quốc Khánh	-Viết hoạt động đạt được của tuần 2,3,4 vào phiếu Kế hoạch làm việc nhóm	Dựa trên nhật ký hoạt động nhóm và phản hồi của các thành viên
	Vũ Văn Thanh	-hỗ trợ trình bày báo cáo	sử dụng word để định dạng báo cáo
6	Nguyễn Huy Hoàng	-Trình bày word và và đầy các file nội dung và file code	Tổ chức folder Drive theo danh mục rõ ràng; kiểm tra định dạng và nội dung trước khi upload.
	Phạm Tất Tụ	-Viết Phiếu học tập cá nhân/nhóm	Dựa trên mẫu có sẵn; tổng hợp hoạt động và tự đánh giá mức độ hoàn thành.
	Nguyễn Quốc Khánh	-Viết hoạt động đạt được của tuần 5,6 vào phiếu Kế hoạch làm việc nhóm	Dựa trên nhật ký hoạt động nhóm và phản hồi của các thành viên
	Vũ Văn Thanh	-Viết tiến độ hoạt động nhóm	Tổng hợp từ khóa từ tài liệu chính; cập nhật bảng tiến độ trong file nhóm.



7	Nguyễn Huy Hoàng	-Trình bày word và và đẩy các file nội dung và file code, dữ liệu test lên drive nhóm	Tổ chức folder Drive theo danh mục rõ ràng; kiểm tra định dạng và nội dung trước khi upload.
	Phạm Tất Tụ	-Viết Phiếu học tập cá nhân/nhóm	Dựa trên mẫu có sẵn; tổng hợp hoạt động và tự đánh giá mức độ hoàn thành.
	Nguyễn Quốc Khánh	-Viết hoạt động đạt được của tuần vào phiếu Kế hoạch làm việc nhóm  -Tìm các phân key word để cả nhóm học	Dựa trên nhật ký hoạt động nhóm và phản hồi của các thành viên.
	Vũ Văn Thanh	-Viết tiếp tiến độ hoạt động nhóm	Tổng hợp từ khóa từ tài liệu chính; cập nhật bảng tiến độ trong file nhóm.

*Ngày tháng năm 2025.*

**XÁC NHẬN CỦA GIẢNG VIÊN**

*(Ký, ghi rõ họ tên)*

Mai Thanh Hồng

## BÁO CÁO HỌC TẬP CÁ NHÂN/NHÓM

Tên lớp : 20242IT6094003    Khóa :18

Họ và tên sinh viên : Nguyễn Huy Hoàng

Tên nhóm : 3

Tên chủ đề : Ứng dụng CNN trong phát hiện vết nứt trên bề mặt đá tự nhiên

<b>Tuần</b>	<b>Người thực hiện</b>	<b>Nội dung công việc</b>	<b>Kết quả đạt được</b>	<b>Kiến nghị với giảng viên hướng dẫn</b>
1	Nguyễn Huy Hoàng	Chốt đề tài: Ứng dụng CNN trong phát hiện vết nứt trên bề mặt đá tự nhiên  Thành lập kênh giao tiếp chung cho nhóm và drive để lưu trữ tài liệu nhóm  Viết biên bản cuộc họp, phiếu hoạt động nhóm cá nhân và nhóm.	Hoàn Thành	
2	Phạm Tất Tụ	Tìm hiểu tổng quan về đề tài	Hoàn Thành	
3	Nguyễn Quốc Khánh	Trình bày tổng quan về mạng CNN  Trình bày ứng dụng thực tế và nội dung kiến trúc cơ bản của CNN	Hoàn Thành	

4	Vũ Văn Thanh	Trình bày các lớp trong mô hình CNN	Hoàn Thành	
5	Nguyễn Huy Hoàng	Trình bày thuật toán Softmax và Argmax	Hoàn Thành	
6	Nguyễn Quốc Khánh, Vũ Văn Thanh	Trình bày về thuật toán Lenet Xây dựng mô hình CNN	Hoàn thành	
7	Phạm Tất Tụ	Hoàn thiện báo cáo	Hoàn thành	

Ngày....tháng.....năm.....  
**XÁC NHẬN CỦA GIẢNG VIÊN**  
*(Ký, ghi rõ họ tên)*

*Mai Thanh Hồng*

## MỤC LỤC

PHIẾU HỌC TẬP CÁ NHÂN/NHÓM .....	2
KẾ HOẠCH THỰC HIỆN TIỂU LUẬN, BÀI TẬP LỚN, ĐỒ ÁN/DỰ ÁN .....	4
BÁO CÁO HỌC TẬP CÁ NHÂN/NHÓM .....	9
DANH SÁCH HÌNH ẢNH .....	0
LỜI MỞ ĐẦU .....	1
CHƯƠNG I: TỔNG QUAN VỀ DỰ ÁN .....	2
1.1. Lý do chọn đề tài .....	2
1.2. Mục tiêu đề tài .....	3
1.3. Phương pháp nghiên cứu .....	4
1.4. Đối tượng và phạm vi nghiên cứu .....	4
1.6. Tiềm năng phát triển .....	5
1.7. Tầm nhìn dài hạn .....	5
1.8. Ngôn ngữ và công nghệ áp dụng .....	5
CHƯƠNG II: TỔNG QUAN VỀ BÀI TOÁN CRACK DETECTION .....	6
2.1. Mô tả bài toán .....	6
2.2. Mô hình áp dụng: Lenet .....	6
2.3. Thách thức đặc thù của dữ liệu đá tự nhiên .....	7
2.4. Mục tiêu của mô hình CNN trong bài .....	7
CHƯƠNG III: CƠ SỞ LÝ THUYẾT VÀ MÔ HÌNH CNN .....	7
3.1. Convolutional Neural Network (CNN) .....	7
3.2. Mô hình LeNet .....	17
CHƯƠNG IV: DỮ LIỆU VÀ TIỀN XỬ LÝ .....	19
4.1 Bộ dữ liệu sử dụng .....	19
4.2. Chia tập & cấu trúc thư mục .....	19
4.3 Tiền xử lý và tăng cường dữ liệu .....	21
4.4 Thống kê sau tiền xử lý .....	21
CHƯƠNG V: THIẾT KẾ VÀ HUẤN LUYỆN MÔ HÌNH .....	22
5.1. Cấu trúc mô hình LeNet .....	22
5.2 Cấu hình huấn luyện .....	24
5.3 Quá trình huấn luyện .....	24
5.4 Đánh giá và nhận xét .....	25
5.5 Nhận xét kết quả .....	27
CHƯƠNG VI: SERVING MODEL .....	27
6.1 Mục tiêu .....	27
6.2 Cấu trúc dự án .....	27
6.3 Tạo API sử dụng FastAPI .....	28

6.4 Tạo mô hình LeNet .....	30
6.5 Triển khai mô hình trên Serve .....	31
CHƯƠNG VII: THIẾT KẾ HỆ THỐNG GIAO DIỆN TRỰC QUAN .....	31
7.1. Công cụ được lựa chọn .....	31
7.2. Các tính năng chính của Streamlit .....	32
7.3. Triển khai ứng dụng .....	32
7.4. Đánh giá và nhận xét hệ thống .....	34
KẾT LUẬN .....	36
TÀI LIỆU THAM KHẢO .....	38

# DANH SÁCH HÌNH ẢNH

Hình 1: Local connected layer .....	8
Hình 2: Shared parameters .....	9
Hình 3: Cấu trúc một mạng CNN .....	9
Hình 4: CONV step .....	10
Hình 5: avg pooling      Hình 6: max pooling .....	11
Hình 7: Fully Connected (FC) .....	11
Hình 8: Bộ lọc .....	12
Hình 9: Stride .....	12
Hình 10: Padding .....	13
Hình 11: receptive field .....	15
Hình 12: Softmax .....	16
Hình 13: Cấu trúc mạng CNN .....	17
Hình 14: Kiến trúc mô hình chi tiết .....	18
Hình 15: Số lượng ảnh trong các tập .....	20
Hình 16: Cấu trúc mô hình LeNet .....	23
Hình 17: Quá trình huấn luyện .....	25
Hình 18: Đồ thị Accuracy và Loss qua các epoch .....	26
Hình 19: Ví dụ Output API doc .....	31
Hình 20: Output FE: Streamlit .....	35

# LỜI MỞ ĐẦU

Đá tự nhiên (granite, marble,...) được ưa chuộng trong xây dựng, trang trí nội - ngoại thất nhờ độ bền, vẻ đẹp vân hạt độc đáo. Tuy nhiên, dưới tác động của tải trọng, môi trường và quy trình gia công, bề mặt đá xuất hiện vết nứt (crack) hoặc đường chảy tinh thể (joint). Nếu không được phát hiện kịp thời, các khuyết tật này làm giảm chất lượng thẩm mỹ, tiềm ẩn nguy cơ mất an toàn kết cấu và gây tổn thất kinh tế trong thi công cũng như bảo trì.

Do đó, ở trong bài viết này, mục đích chính của chúng em sẽ là tìm hiểu và Xây dựng hệ thống CNN phân loại crack/non-crack và dự đoán đặc trưng hình học vết nứt (độ rộng, mật độ) trên ảnh chụp cận cảnh đá tự nhiên.

## **Mục tiêu nghiên cứu:**

- Xây dựng hệ thống CNN phân loại và dự đoán đặc trưng hình học vết nứt ( độ rộng, mật độ) trên ảnh chụp cận cảnh đá tự nhiên.
- Đánh giá định lượng độ chính xác, tốc độ, độ bền mô hình trên bộ dữ liệu thực thu thập tại Việt Nam.

## **Đối tượng và phạm vi:**

- Đối tượng: ảnh RGB độ phân giải  $\geq 1024 \times 768$ px của bề mặt đá tự nhiên
- Điều kiện ghi hình: chiếu sáng tự nhiên/ đèn flash, góc chụp  $0 - 30^\circ$ , khoảng cách 20–60 cm.
- Giới hạn: không xét vi nứt  $< 0,1$  mm

## **Cấu trúc báo cáo:**

Chương 1: Tổng quan về Convolutional Neural Network (CNN)

Chương 2: Tổng quan bài toán Crack-Rock

Chương 2: Dữ liệu & tiền xử lý

Chương 3: Thiết kế & triển khai mô hình

Chương 5: Ứng dụng & triển khai.

Chương 6: Kết luận và Hướng phát triển.

# CHƯƠNG I: TỔNG QUAN VỀ DỰ ÁN

## 1.1. Lý do chọn đề tài

Trí tuệ nhân tạo (AI), đặc biệt là nhánh học sâu (Deep Learning), đã và đang phát triển mạnh mẽ, tạo ra bước đột phá trong nhiều lĩnh vực như y tế, giao thông, công nghiệp sản xuất và giám sát tự động. Trong đó, thị giác máy tính (Computer Vision) – một nhánh quan trọng của AI – cho phép máy móc “nhìn thấy” và hiểu được thế giới hình ảnh tương tự như con người.

Các mạng nơ-ron tích chập (CNN) đóng vai trò trung tâm trong việc trích xuất và học đặc trưng hình ảnh, mang lại hiệu quả vượt trội trong các bài toán như nhận diện khuôn mặt, phát hiện vật thể và phân tích bề mặt vật liệu.

### 1.1.1. Bối cảnh phát triển của AI và ứng dụng học sâu

Trong thực tế sản xuất và xây dựng, đá tự nhiên như granite và marble thường được sử dụng làm vật liệu lát sàn, ốp tường hoặc gia công nội thất. Tuy nhiên, do yếu tố vận chuyển, va chạm hoặc áp lực môi trường, các vết nứt vi mô có thể xuất hiện trên bề mặt đá. Nếu không được phát hiện sớm, các vết nứt này không chỉ làm giảm giá trị thẩm mỹ mà còn ảnh hưởng đến độ bền và an toàn của công trình.

Việc phát triển một hệ thống tự động nhận diện vết nứt giúp cải thiện hiệu suất kiểm tra chất lượng, giảm phụ thuộc vào lao động thủ công và tăng độ chính xác trong đánh giá sản phẩm.

### 1.1.2. Ý nghĩa của bài toán nhận diện vết nứt trên bề mặt đá tự nhiên

Trong thực tế sản xuất và xây dựng, đá tự nhiên như granite và marble thường được sử dụng làm vật liệu lát sàn, ốp tường hoặc gia công nội thất. Tuy nhiên, do yếu tố vận chuyển, va chạm hoặc áp lực môi trường, các vết nứt vi mô có thể xuất hiện trên bề mặt đá. Nếu không được phát hiện sớm, các vết nứt này



không chỉ làm giảm giá trị thẩm mỹ mà còn ảnh hưởng đến độ bền và an toàn của công trình.

Việc phát triển một hệ thống tự động nhận diện vết nứt giúp cải thiện hiệu suất kiểm tra chất lượng, giảm phụ thuộc vào lao động thủ công và tăng độ chính xác trong đánh giá sản phẩm.

### **1.1.2. Hạn chế của phương pháp truyền thống**

Phương pháp kiểm tra truyền thống chủ yếu dựa vào quan sát thủ công bởi kỹ thuật viên có kinh nghiệm.

Tuy nhiên:

- Kết quả phụ thuộc nhiều vào chủ quan người kiểm tra.
- Tốn thời gian và nhân lực khi cần kiểm tra số lượng lớn.
- Khó phát hiện vết nứt nhỏ hoặc vết nứt có hoa văn nền phức tạp.

Do đó, việc áp dụng AI – cụ thể là mô hình CNN – trong phát hiện vết nứt là một hướng tiếp cận hiện đại, hiệu quả và có tính ứng dụng thực tiễn cao.

## **1.2. Mục tiêu đề tài**

### **1.2.1. Mục tiêu tổng quát**

Xây dựng một mô hình học sâu sử dụng mạng CNN để nhận diện vết nứt trên ảnh bề mặt đá tự nhiên, góp phần nâng cao tính chính xác và tự động hóa trong quy trình kiểm định chất lượng.

### **1.2.2. Mục tiêu cụ thể**

- Huấn luyện mô hình CNN (LeNet) phân loại ảnh có/không có vết nứt.
- Tiền xử lý và tăng cường dữ liệu phù hợp với đặc điểm vân đá.
- Đánh giá hiệu quả mô hình qua các chỉ số chính xác (accuracy), độ mất mát
- Xây dựng pipeline đơn giản có thể ứng dụng trong môi trường thực tế.

### **1.3. Phương pháp nghiên cứu**

#### **1.3.1. Phương pháp lý thuyết**

- Tìm hiểu lý thuyết về CNN, đặc biệt là cấu trúc LeNet và VGG.
- Nghiên cứu các bài toán classification và segmentation trong lĩnh vực phát hiện khuyết tật vật liệu.

#### **1.3.2. Phương pháp thực nghiệm**

- Triển khai mô hình CNN bằng thư viện PyTorch.
- Huấn luyện mô hình trên tập dữ liệu ảnh crack/non-crack.
- Đánh giá và điều chỉnh siêu tham số dựa trên kết quả thực nghiệm.

#### **1.3.3. Công cụ hỗ trợ**

- Ngôn ngữ lập trình: Python
- Thư viện: PyTorch, Torchvision, Matplotlib
- Công cụ xử lý ảnh: OpenCV
- Môi trường phát triển: Google Colab hoặc Kaggle

### **1.4. Đối tượng và phạm vi nghiên cứu**

#### **1.4.1. Đối tượng nghiên cứu**

- Ảnh bề mặt đá granite và marble, chụp cận cảnh, có hoặc không có vết nứt.

#### **1.4.2. Phạm vi nghiên cứu**

- Tập trung vào bài toán phân loại nhị phân (crack vs non-crack).
- Không triển khai mô hình phát hiện đối tượng hoặc phân đoạn pixel-level.
- Không xử lý ảnh trong điều kiện khắc nghiệt (ánh sáng quá mạnh, bụi bẩn, nứt vi mô dưới 0.1mm).

### **1.5. Tình hình hiện tại**

Một số nghiên cứu gần đây đã ứng dụng CNN để phát hiện vết nứt trên bê tông, đường nhựa hoặc thép. Tuy nhiên, ứng dụng cụ thể cho đá tự nhiên còn

tương đối ít và thiếu bộ dữ liệu chuyên biệt. Đặc điểm hoa văn phức tạp của đá tự nhiên là thách thức cho mô hình học sâu, và là động lực để nhóm thực hiện đề tài này

### 1.6. Tiềm năng phát triển

- Mở rộng sang bài toán segmentation để xác định chính xác vùng nứt.
- Áp dụng Transfer Learning với mô hình mạnh hơn như ResNet, MobileNet, hoặc EfficientNet.
- Xây dựng giao diện người dùng (app/web) phục vụ kiểm định tại chỗ.
- Tích hợp với thiết bị edge như Jetson Nano để triển khai ngoài thực địa.

### 1.7. Tầm nhìn dài hạn

Hệ thống phát hiện vết nứt có thể trở thành một **công cụ kiểm định chất lượng bán tự động** cho các xưởng gia công đá, nhà máy sản xuất vật liệu xây dựng, hoặc đơn vị tư vấn kỹ thuật xây dựng. Trong tương lai, có thể kết hợp với các cảm biến và hệ thống robot để tạo thành chuỗi kiểm tra hoàn toàn tự động.

### 1.8. Ngôn ngữ và công nghệ áp dụng

Trong khuôn khổ đề tài, nhóm lựa chọn Python 3.11 làm ngôn ngữ lập trình chính nhờ tính linh hoạt, hệ sinh thái thư viện phong phú và khả năng tái lập thí nghiệm cao. Phần cốt lõi của mô hình được xây dựng trên khung PyTorch 2.x, kết hợp Torchvision để quản lý dữ liệu hình ảnh và các tiện ích tiền xử lý. Mô hình học máy áp dụng là Convolutional Neural Network – LeNet, một kiến trúc CNN kinh điển, gọn nhẹ nhưng đủ năng lực khái quát hoá đặc trưng vết nứt trên ảnh nhỏ.

Đối với khâu tiền xử lý và thao tác ảnh, nhóm sử dụng song song hai thư viện Pillow (PIL) và OpenCV-Python: PIL phục vụ thao tác IO cơ bản, trong khi OpenCV hỗ trợ các phép biến đổi hình học và tăng cường dữ liệu ở tốc độ cao.

Quá trình phát triển, huấn luyện và thử nghiệm được thực hiện chủ yếu trên Google Colab khai thác GPU Tesla T4 miễn phí—kết hợp Jupyter Notebook nhằm ghi chép và chia sẻ kết quả một cách minh bạch. Cuối cùng, các biểu đồ diễn tiến hàm mất mát, độ chính xác cũng như confusion matrix được trực quan hóa bằng Matplotlib và Seaborn, giúp phân tích quá trình hội tụ và chất lượng mô hình một cách trực quan, thuyết phục.

## CHƯƠNG II: TỔNG QUAN VỀ BÀI TOÁN CRACK DETECTION

### 2.1. Mô tả bài toán

Phát hiện vết nứt (crack detection) là một bài toán thị giác máy tính có vai trò quan trọng trong kiểm định chất lượng bề mặt vật liệu, đặc biệt là đá tự nhiên (như đá granite, marble...). Mục tiêu là xác định xem **ảnh đầu vào có chứa vết nứt hay không**, dựa trên đặc trưng hình ảnh.

Trong bài tập lớn này, chúng em tập trung vào **nhiệm vụ phân loại nhị phân (binary classification)**, tức là:

- **Input:** ảnh cắt từ mặt đá
- **Output:** trả về dự đoán có vết nứt hay không

Điều khó ở đây là hoa văn đá vốn đã loang lổ, dễ gây nhầm lẫn với vết nứt thật. Thêm vào đó, ảnh chụp thực tế có lúc sáng gắt, lúc tối, góc chụp lệch..., nên mô hình phải đủ “khôn” để không bị đánh lừa bởi những chi tiết nhiễu đó. Chúng em dùng mạng CNN (kiến trúc LeNet) cùng các kỹ thuật tiền xử lý, tăng cường dữ liệu để mô hình học cách phân biệt vết nứt thật với vân đá tự nhiên.

### 2.2. Mô hình áp dụng: Lenet

Hai kiến trúc CNN được áp dụng trong bài toán này là **LeNet-5**.

**LeNet-5:** là mạng nơ-ron tích chập đầu tiên có cấu trúc rõ ràng, gồm 2 lớp convolution, 2 lớp pooling và 1 mạng fully connected. Ưu điểm: đơn giản, dễ huấn luyện, phù hợp với ảnh nhỏ ( $32 \times 32$  hoặc  $64 \times 64$ ).

### 2.3. Thách thức đặc thù của dữ liệu đá tự nhiên

**Mẫu vân đá phức tạp:** dễ gây nhầm lẫn giữa vết nứt thật và hoa văn tự nhiên.

**Tương phản thấp:** crack có màu gần giống nền, khó phát hiện bằng mắt thường.

**Nhiều ảnh chụp:** ảnh có thể bị ánh sáng chiếu xiên, bị bóng hoặc phản sáng.

Do đó, cần có bước **tiền xử lý ảnh phù hợp** và lựa chọn mô hình CNN có khả năng phân biệt tốt đặc trưng hình học của vết nứt.

### 2.4. Mục tiêu của mô hình CNN trong bài

- Huấn luyện mô hình LeNet đơn giản để phân loại crack/non-crack.
- Tối ưu hoá độ chính xác qua các siêu tham số (learning rate, batch size...)
- Đánh giá hiệu quả mô hình trên tập test (accuracy, loss).

## CHƯƠNG III: CƠ SỞ LÝ THUYẾT VÀ MÔ HÌNH CNN

### 3.1. Convolutional Neural Network (CNN)

#### 3.1.1. Tổng quan Convolutional Neural Network (CNN)

##### 3.1.1.1. Normal Neural Network trên images

Mỗi image có hàng ngàn pixels, mỗi pixel được xem như 1 feature, vậy nếu như một bức ảnh có kích thước  $1000 * 1000$ , thì sẽ có 1.000.000 features. Với normal feed-forward neural networks, mỗi layer là full-connected với previous input layer.

Trong noral feed-forward neural network, với mỗi layer, có 1.000.000 pixels, mỗi pixel lại kết nối full-connected với 1.000.000 pixels ở layer trước, tức sẽ có 1012 tham số. Đây là còn số quá lớn để có thể tính được vào thời điểm đó, bởi vì với mô hình có nhiều tham số, sẽ dễ bị overfitted và cần lượng lớn data cho việc training, ngoài ra còn cần nhiều memory và năng lực tính toán cho việc training và prediction.

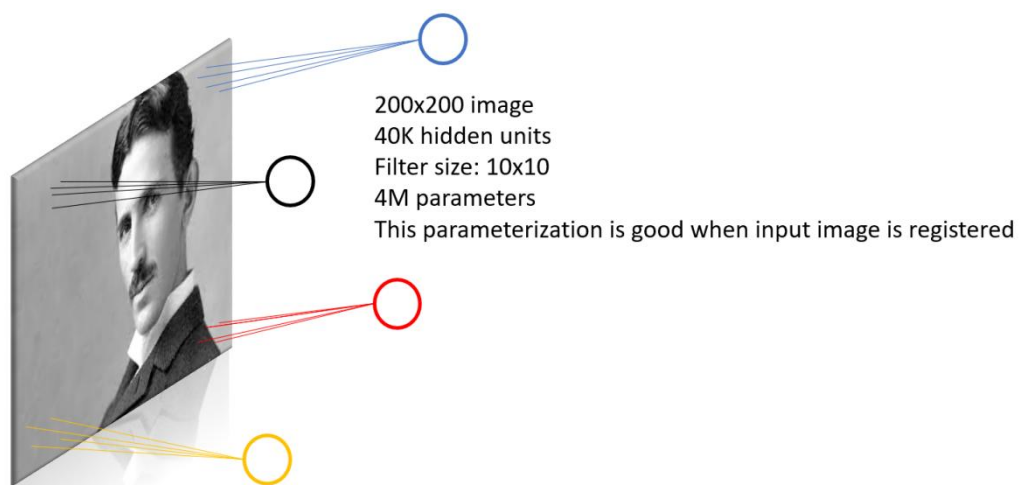
### 3.1.1.2. CNN hoạt động ra sao?

Có 2 đặc tính của image hình thành nên cách hoạt động của CNN trên image, đó là **feature localization** và **feature independence of location**.

Feature localization: mỗi pixel hoặc feature có liên quan với các pixel quanh nó.

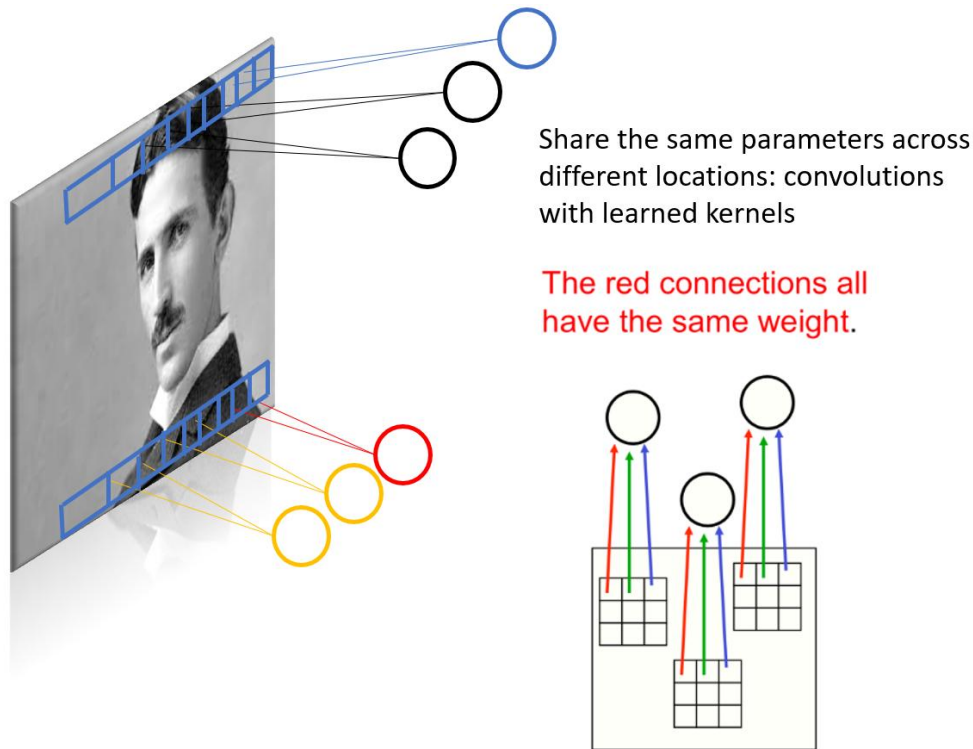
Feature Independence of location: mỗi feature dù nó có nằm ở đâu trong bức ảnh, thì nó vẫn mang giá trị của feature đó. CNN xử lý vấn đề có quá nhiều tham số với Shared parameters (feature independence of location) của Locally connected networks (feature loclization), được gọi là Convolution Net.

**Locally connected layer:** Trong hidden layer đầu tiên, mỗi node sẽ kết nối tới một cụm nhỏ pixels của input image chứ không phải toàn bộ image, gọi là small portion. Theo cách này, ta sẽ có ít kết nối hơn, vì thế ít tham số hơn giữa input và hidden layer đầu tiên.



*Hình 1: Local connected layer*

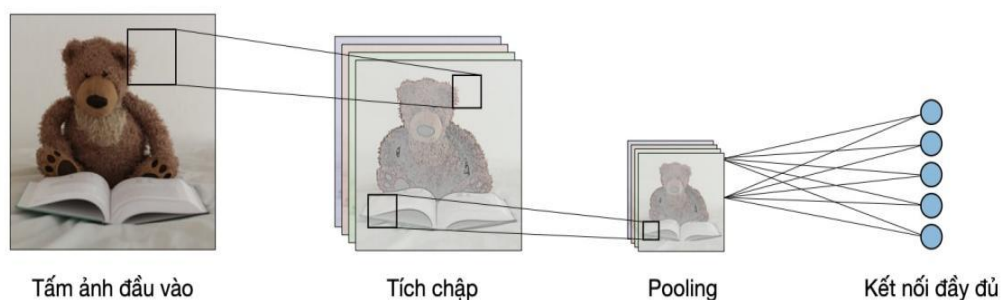
**Shared parameters:** Có những khu vực, mà việc tìm ra feature là giống nhau về cách làm, vì vậy ta có thể dùng chung bộ parameter, trong hình trên là phía phải bên trên và phía trái bên dưới. Tức ta chia sẻ bộ parameter giữa những vị trí khác nhau trong bức ảnh.



Hình 2: Shared parameters

### 3.1.2. Kiến trúc của một mạng CNN

Kiến trúc truyền thống của một mạng CNN - Mạng neural tích chập (Convolutional neural network), còn được biết đến với cái tên CNNs, là một dạng mạng neural được cấu thành bởi các tầng sau:



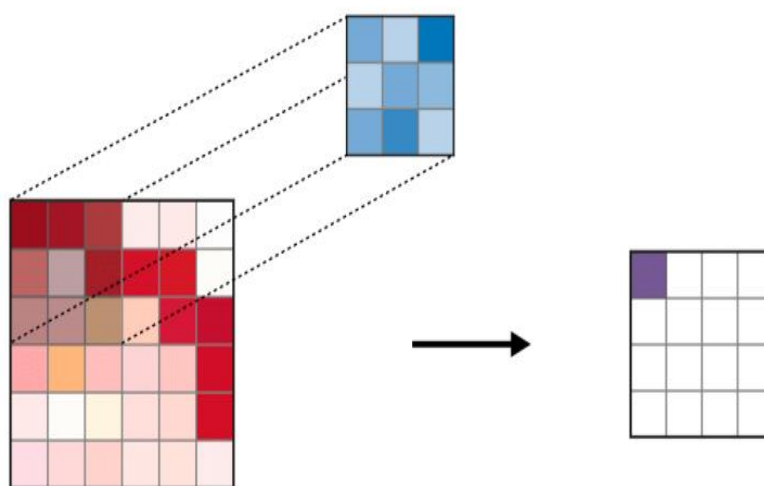
Hình 3: Cấu trúc một mạng CNN

Tầng tích chập và tầng pooling có thể được hiệu chỉnh theo các siêu tham số (hyperparameters) được mô tả ở những phần tiếp theo

### 3.1.2.1. Các kiểu tầng

#### 3.1.2.1.1. Tầng tích chập (CONV)

Tầng tích chập (CONV) sử dụng các bộ lọc để thực hiện phép tích chập khi đưa chúng đi qua đầu vào  $I$  theo các chiều của nó. Các siêu tham số của các bộ lọc này bao gồm kích thước và bộ lọc  $F$  và độ trượt (stride)  $S$ . Kết quả đầu ra  $O$  được gọi là feature map hay activation map.



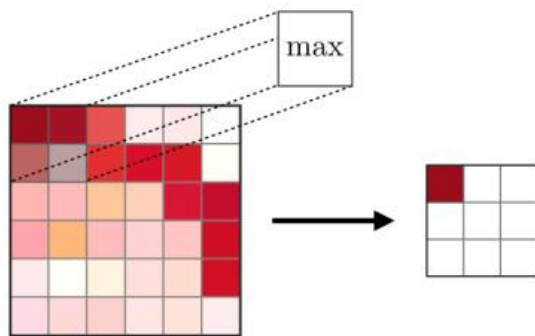
Hình 4: CONV step

*Lưu ý: Bước tích chập cũng có thể được khái quát hóa cả với trường hợp một chiều (1D) và ba chiều (3D).*

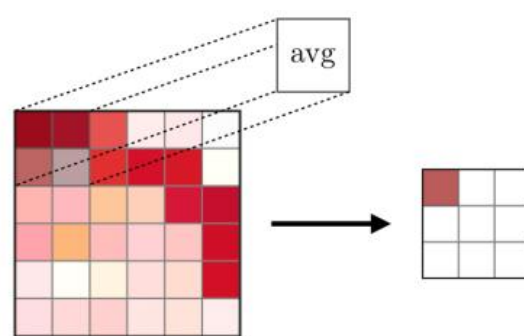
#### 3.1.2.1.2. Pooling (POOL)

Tầng pooling (POOL) là một phép downsampling, thường được sử dụng sau tầng tích chập, giúp tăng tính bất biến không gian. Cụ thể, max pooling và average pooling là những dạng pooling đặc biệt, mà tương ứng là trong đó giá trị lớn nhất và giá trị trung bình được lấy ra.





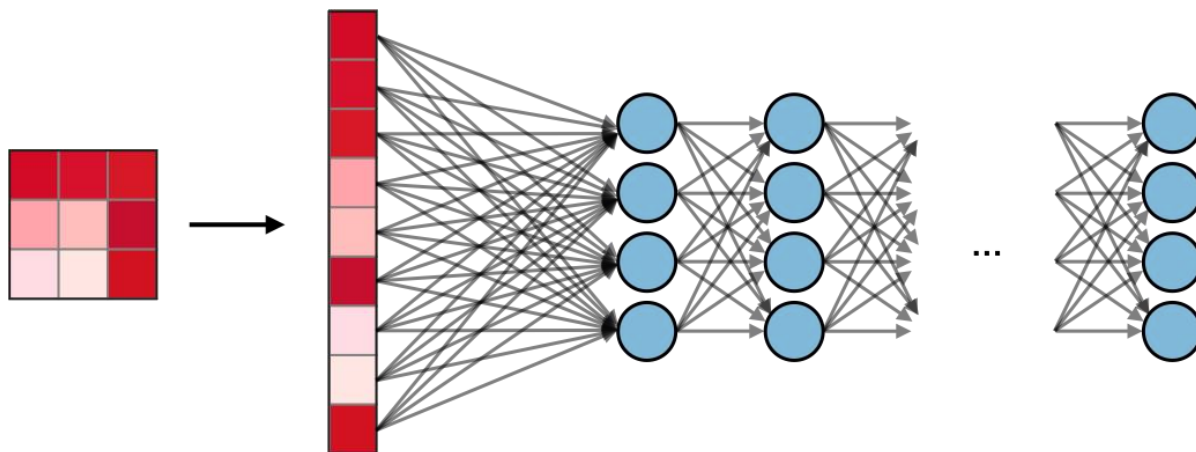
Hình 5: avg pooling



Hình 6: max pooling

### 3.1.2.1.3. Fully Connected (FC)

Tầng kết nối đầy đủ (FC) nhận đầu vào là các dữ liệu đã được làm phẳng, mà mỗi đầu vào đó được kết nối đến tất cả neuron. Trong mô hình mạng CNNs, các tầng kết nối đầy đủ thường được tìm thấy ở cuối mạng và được dùng để tối ưu hóa mục tiêu của mạng ví dụ như độ chính xác của lớp.



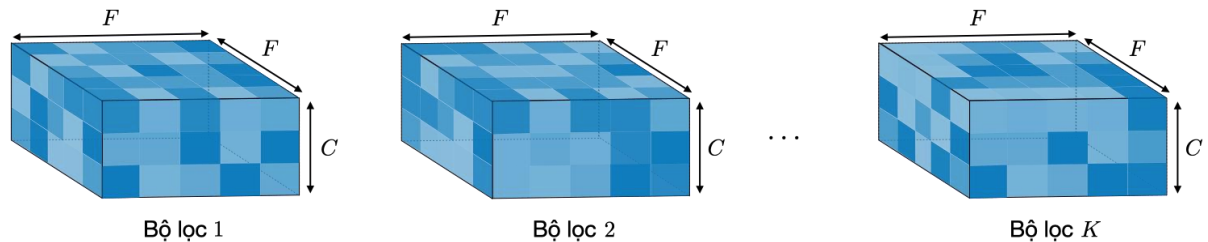
Hình 7: Fully Connected (FC)

### 3.1.3. Các siêu tham số của bộ lọc

Tầng tích chập chứa các bộ lọc mà rất quan trọng cho ta khi biết ý nghĩa đằng sau các siêu tham số của chúng.

### 3.1.3.1. Các chiều của một bộ lọc

Một bộ lọc kích thước  $F \times F$  áp dụng lên đầu vào chứa  $C$  kênh thì có kích thước tổng thể là  $F \times F \times C$  thực hiện phép tích chập trên đầu vào kích thước  $I \times I \times C$  và cho ra một feature map (hay còn gọi là activation map) có kích thước  $O \times O \times 1$

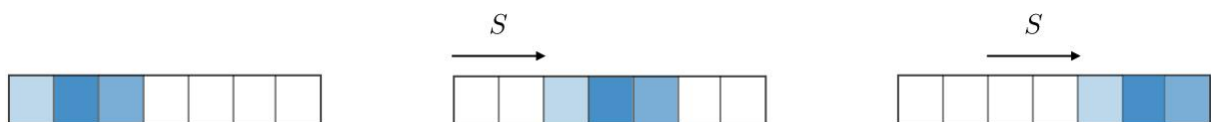


Hình 8: Bộ lọc

*Lưu ý: Việc áp dụng  $K$  bộ lọc có kích thước  $F \times F$  cho ra một feature map có kích thước  $O \times O \times K$ .*

### 3.1.3.2. Stride

Đối với phép tích chập hoặc phép pooling, độ trượt SS ký hiệu số pixel mà cửa sổ sẽ di chuyển sau mỗi lần thực hiện phép tính.



Hình 9: Stride

### 3.1.3.3. Zero-padding

Zero-padding là tên gọi của quá trình thêm PP số không vào các biên của đầu vào. Giá trị này có thể được lựa chọn thủ công hoặc một cách tự động bằng một trong ba những phương pháp như Valid, Same, Full

**Valid:**

- Không sử dụng padding
- Bỏ phép tích chập cuối nếu số chiều không khớp

**Same:**

- Sử dụng padding để làm cho feature map có kích thước  $\lceil \frac{I}{S} \rceil$
- Kích thước đầu ra thuận lợi về mặt toán học
- Còn được gọi là ‘half’ padding

### Full

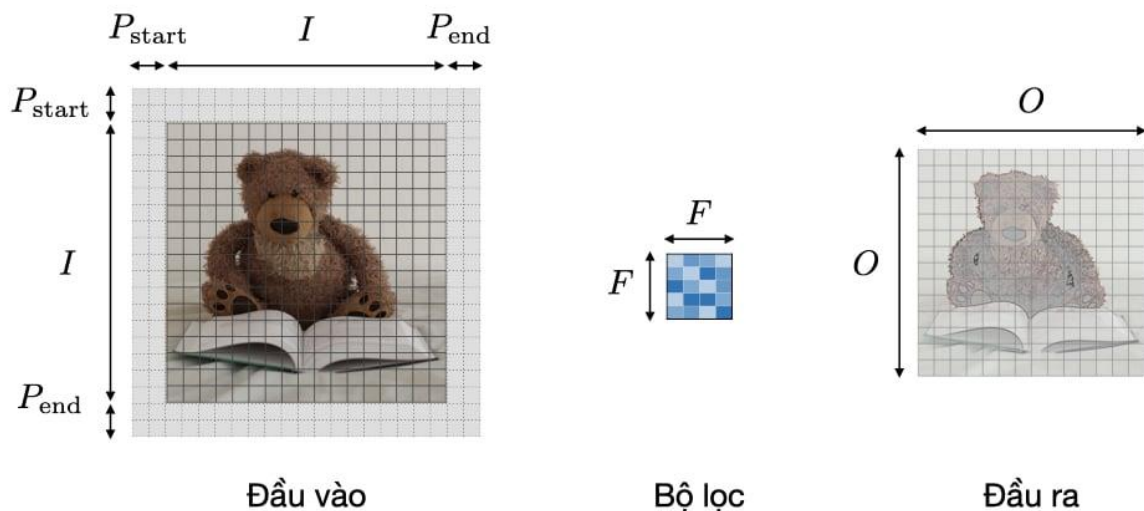
- Padding tối đa sao cho các phép tích chập có thể được sử dụng tại các rìa của đầu vào
- Bộ lọc ‘thấy’ được đầu vào từ đầu đến cuối

### 3.1.4. Điều chỉnh siêu tham số

#### 3.1.4.1. Tính tương thích của tham số trong mô hình tích chập

Bằng cách ký hiệu  $I$  là độ dài kích thước đầu vào,  $F$  là độ dài của bộ lọc,  $P$  là số lượng zero padding,  $S$  là độ trượt, ta có thể tính được độ dài  $O$  của feature map theo một chiều bằng công thức:

$$O = \frac{I - F + P_{start} + P_{end}}{S} + 1$$

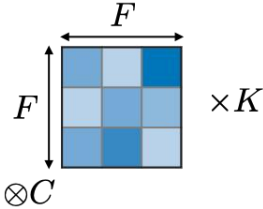
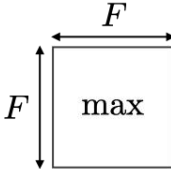
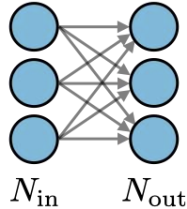


Hình 10: Padding

**Lưu ý:** Trong một số trường hợp,  $P_{start}=P_{end} \triangleq P$ , ta có thể thay thế  $P_{start}+P_{end}$  bằng  $2P$  trong công thức trên.

### 3.1.4.2. Hiểu về độ phức tạp của mô hình

Để đánh giá độ phức tạp của một mô hình, cách hữu hiệu là xác định số tham số mà mô hình đó sẽ có. Trong một tầng của mạng neural tích chập, nó sẽ được tính toán như sau:

	CONV	POOL	FC
Minh Họa			
Kích thước đầu vào	$I \times I \times C$	$I \times I \times C$	$N_{in}$
Kích thước đầu ra	$O \times O \times K$	$O \times O \times C$	$N_{out}$
Số lượng tham số	$(F \times F \times C + 1) \cdot K$	0	$(N_{in} + 1) \times N_{out}$
Lưu ý	<ul style="list-style-type: none"> <li>- Một tham số bias với mỗi bộ lọc</li> <li>- Trong đa số trường hợp, <math>S &lt; F</math></li> <li>- Một lựa chọn phổ biến cho <math>K</math> là <math>2C</math></li> </ul>	<ul style="list-style-type: none"> <li>- Phép pooling được áp dụng lên từng kênh (channel-wise)</li> <li>- Trong đa số trường hợp, <math>S = F</math></li> </ul>	<ul style="list-style-type: none"> <li>- Đầu vào được làm phẳng</li> <li>- Mỗi neuron có một tham số bias</li> <li>- Số neuron trong một tầng FC phụ thuộc vào ràng buộc kết cấu</li> </ul>

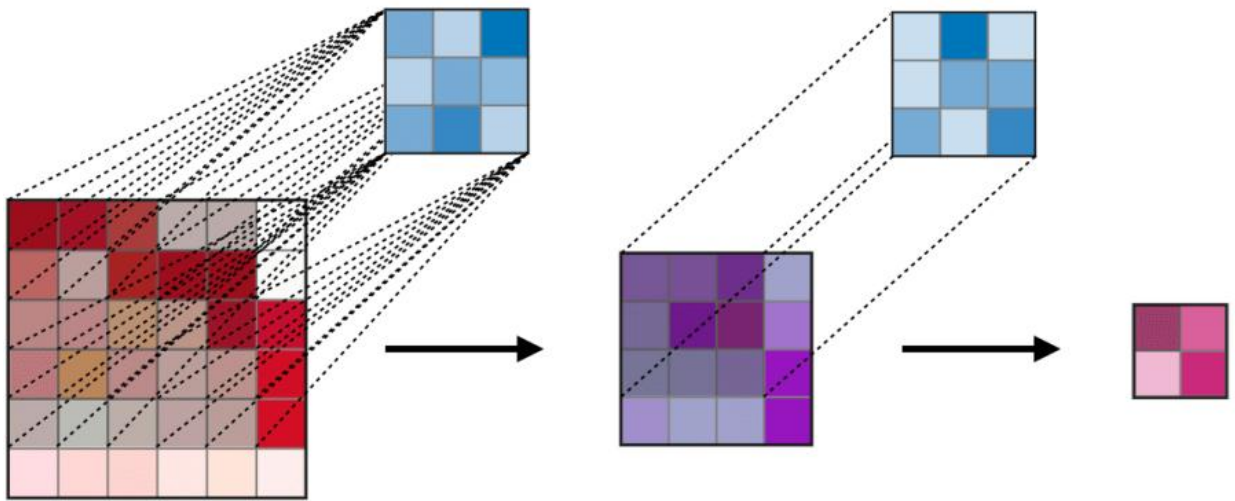
### 3.1.4.3. Trường Thụ Cảm (receptive field)

Trường thụ cảm (receptive field) tại tầng  $k$  là vùng được ký hiệu  $R_k \times R_k$  của đầu vào mà những pixel của activation map thứ  $k$  có thể "nhìn thấy". Bằng cách gọi  $F_j$  là kích thước bộ lọc của tầng  $j$  và  $S_j$  là giá trị độ trượt

của tầng  $i$  và để thuận tiện, ta mặc định  $S_0=1$ , trường thụ cảm của tầng được tính toán bằng công thức:

$$R_k = 1 + \sum_{j=1}^k (F_j - 1) \prod_{i=0}^{j-1} S_i$$

Trong ví dụ bên dưới, ta có  $F_1 = F_2 = 3$  và  $S_1 = S_2 = 1$ , nên cho ra được  $R_2 = 5$

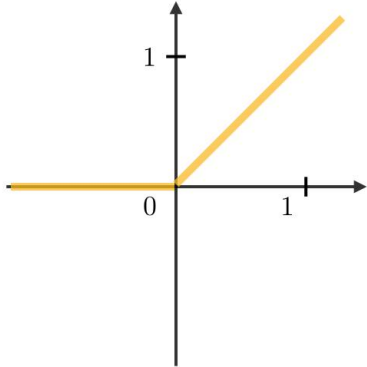
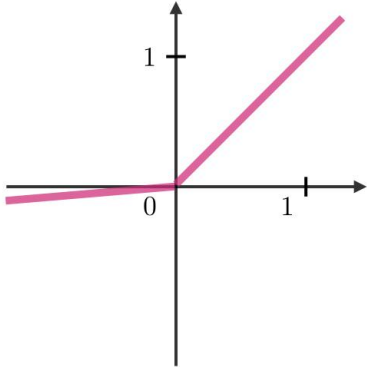
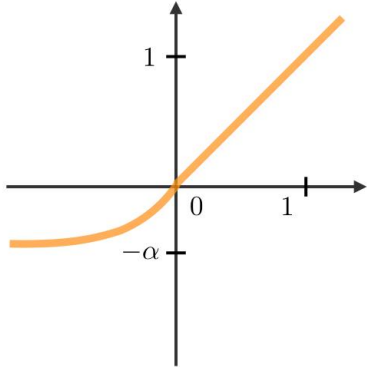


Hình 11: receptive field

### 3.1.5. Các hàm kích hoạt thường gặp (activation function)

#### 3.1.5.1. Rectified Linear Unit (ReLU)

Tầng rectified linear unit (ReLU) là một hàm kích hoạt  $g$  được sử dụng trên tất cả các thành phần. Mục đích của nó là tăng tính phi tuyến tính cho mạng. Những biến thể khác của ReLU được tổng hợp ở bảng dưới:

ReLU	Leaky ReLU	ELU
$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ với $\epsilon \ll 1$	$g(z) = \max(\alpha(ez - 1), z)$ với $\alpha \ll 1$
		
Độ phức tạp phi tuyến tính có thể thông dịch được về mặt sinh học	Gán vấn đề ReLU chết cho những giá trị âm	Khả vi tại mọi nơi

### 3.1.5.2. Softmax

Bước softmax có thể được coi là một hàm logistic tổng quát lấy đầu vào là một vector chứa các b giá trị  $x \in R^n$  và cho ra là một vector gồm các xác suất  $p \in R^n$  thông qua một hàm softmax ở cuối kiến trúc. Nó được định nghĩa như sau:

$$p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{với} \quad p_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

Hình 12: Softmax

## 3.2. Mô hình LeNet

### 3.2.1 Giới thiệu tổng quan

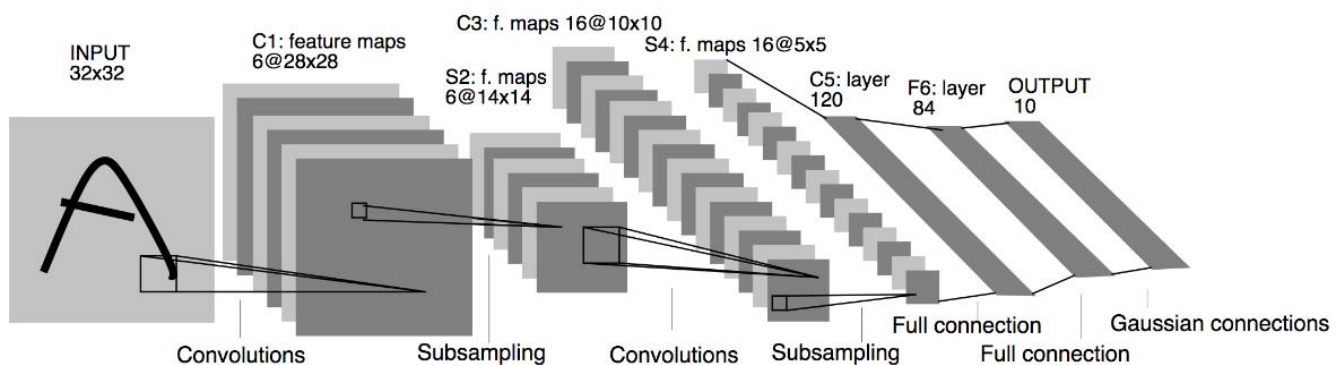
**LeNet-5** là một trong những kiến trúc mạng nơ-ron tích chập (CNN) đầu tiên và nổi bật nhất, được đề xuất bởi **Yann LeCun** vào năm 1998. Mô hình này ban đầu được thiết kế để nhận diện chữ số viết tay trong bộ dữ liệu MNIST. Với cấu trúc đơn giản nhưng hiệu quả, LeNet vẫn là lựa chọn phù hợp để áp dụng cho các bài toán phân loại ảnh cơ bản, đặc biệt là trong môi trường giáo dục và huấn luyện mô hình cơ bản.

Trong dự án này, nhóm lựa chọn LeNet vì các lý do:

- Cấu trúc nhẹ, dễ huấn luyện trên máy không có GPU.
- Đủ khả năng học được các đặc trưng hình ảnh đơn giản như vết nứt (crack).
- Dễ dàng triển khai bằng thư viện TensorFlow/Keras hoặc PyTorch.

### 3.2.2 Kiến trúc chi tiết của mô hình LeNet

- Mô hình LeNet gốc gồm **7 tầng** chính, bao gồm:
  - + 3 tầng **Convolutional**
  - + 2 tầng **Subsampling (Avg Pooling)**
  - + 2 tầng **Fully Connected (Dense)**
- Khi xây dựng kiến trúc, dùng 2 loại layers mang ý nghĩa chính là Convolution layers và Subsampling layers.



Hình 13: Cấu trúc mạng CNN

Kiến trúc mô hình chi tiết:

Layer	Layer type	Feature Map	Size	Kernel	Stride	Activation	Padding
Input	Image	1	32x32	-	-	-	
C1	Convolution (Conv2D)	6	28x28	5x5	1	relu	no
S2	Sub sampling (AvgPool2D)	6	14x14	2x2	2	-	same
C3	Convolution (Conv2D)	16	10x10	5x5	1	relu	no
S4	Sub sampling (AvgPool2D)	16	5x5	2x2	2	-	same
C5	Convolution	120	1x1	5x5	1	relu	
F6	Fully connected	-	84	-	-	relu	
Output	Fully connected	-	10	-	-	softmax	

Hình 14: Kiến trúc mô hình chi tiết

### 3.2.3 Mô hình hóa bằng mã (Keras)

```

lenet_model = tf.keras.models.Sequential([
    1     tf.keras.layers.Conv2D(filters=6, kernel_size=(5, 5),
    2     activation='relu', input_shape=(img_size, img_size, 3)),
    3
    4     # S2: Average Pooling layer
    5     tf.keras.layers.AvgPool2D(pool_size=(2, 2), strides=2),
    6
    7     # C3: Conv layer with 16 filters, kernel size 5x5
    8     tf.keras.layers.Conv2D(filters=16, kernel_size=(5, 5),
    9     activation='relu'),
    10
    11    # S4: Average Pooling layer
    12    tf.keras.layers.AvgPool2D(pool_size=(2, 2), strides=2),
    13
    14    # Flatten before going into dense layers
    15    tf.keras.layers.Flatten(),
    16
    17    # C5: Fully connected layer with 120 units
    18    tf.keras.layers.Dense(units=120, activation='relu'),
    19
    20    # F6: Fully connected layer with 84 units
    21    tf.keras.layers.Dense(units=84, activation='relu'),
    22
    23    # Output layer: 1 unit (e.g., for binary classification)
    24    tf.keras.layers.Dense(units=1, activation='sigmoid') # or 'softmax'
    25    for multi-class
    26 ])

lenet_model.summary()

```



### 3.2.4. Ưu điểm và hạn chế

Ưu điểm:

- Cấu trúc đơn giản, dễ triển khai và huấn luyện.
- Số lượng tham số vừa phải, thích hợp với máy không có GPU.
- Phù hợp cho bài toán phân loại nhị phân trên ảnh nhỏ.

Hạn chế:

- Không thích hợp với ảnh độ phân giải cao hoặc đặc trưng phức tạp.
- Chưa tích hợp cơ chế regularization hiện đại như Dropout, BatchNorm.
- Dễ overfit nếu dữ liệu không được tăng cường tốt.

### 3.2.5 Ứng dụng trong dự án

Trong dự án này, mô hình LeNet được sử dụng để phân loại ảnh cắt từ bề mặt đá tự nhiên thành 2 lớp: crack và non-crack. Dữ liệu đầu vào được resize về  $32 \times 32$  pixels và chuẩn hoá trước khi đưa vào mô hình.

Việc huấn luyện được thực hiện trên tập dữ liệu hơn 3.000 ảnh đã được gán nhãn, sử dụng hàm mất mát CrossEntropy và thuật toán tối ưu Adam.

## CHƯƠNG IV: DỮ LIỆU VÀ TIỀN XỬ LÝ

### 4.1 Bộ dữ liệu sử dụng

Nhóm sử dụng tập dữ liệu Concrete Crack Dataset, được chia sẵn thành ba thư mục train, val, test, với tổng cộng 40.000 ảnh thuộc hai lớp:

- crack (có vết nứt)
- no\_crack (không có vết nứt)

Lấy data: Data được lên một repo github. Thao tác dưới đây để lấy data:

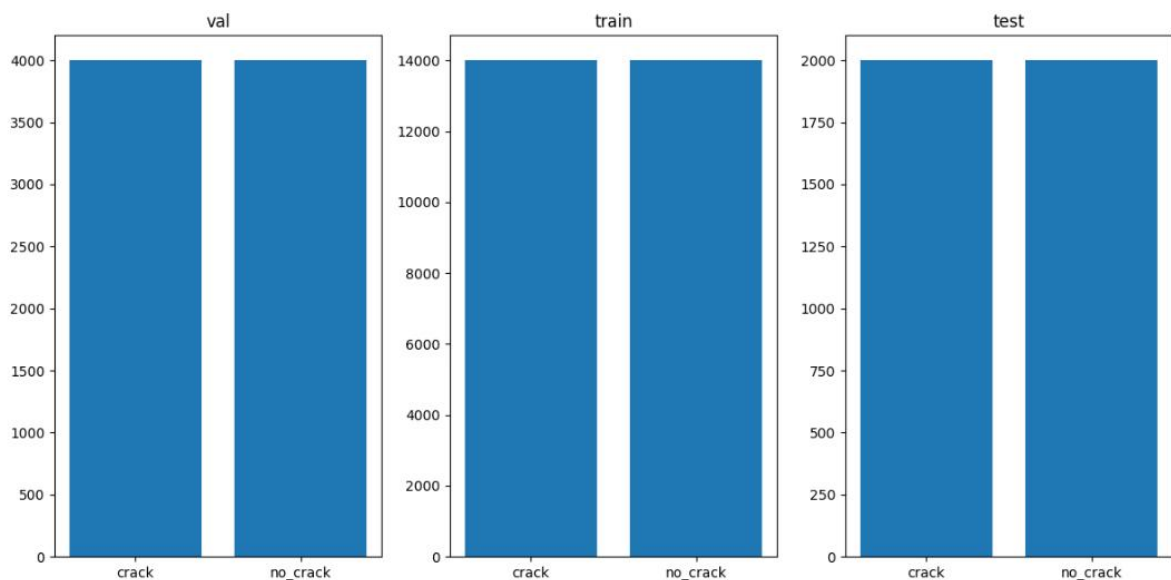
```
1 filepath = 'temp_concrete_crack'
2 Repo.clone_from('https://github.com/bimewok/Concrete-Crack-Image-Classifier', filepath)
```

### 4.2. Chia tập & cấu trúc thư mục

Tập dữ liệu	<i>crack</i>	<i>no_crack</i>	Tổng	Tỉ lệ
Train	14 000	14 000	28 000	70 %
Validation	4 000	4 000	8 000	20 %
Test	2 000	2 000	4 000	10 %

Tất cả ảnh gốc là tệp JPEG, ba kênh RGB, độ phân giải  $227 \times 227$  px. Sau khi tải về, nhóm rà soát, loại bỏ 37 ảnh hỏng (đọc lỗi) rồi tổ chức lại thư mục cho thuận tiện huấn luyện.

```
1 def show_labels(data_dir):
2     data = os.listdir(data_dir)
3     fig, ax = plt.subplots(1, len(data), figsize=(12,6))
4     for idx in range(len(data)):
5         sub_dir = os.path.join(data_dir, data[idx])
6         labels = os.listdir(sub_dir)
7         list_data = []
8         for label in labels:
9             image_files = list(paths.list_images(os.path.join(sub_dir,
10 label)))
11             list_data.append(len(image_files))
12         ax[idx].bar(labels, list_data)
13         ax[idx].set_title(data[idx])
14         # ax[idx].axis('off')
15     plt.tight_layout()
16     plt.show()
17 show_labels(base_dir)
```



Hình 15: Số lượng ảnh trong các tập

Cấu trúc này tương thích trực tiếp với phương thức `flow_from_directory()` của **Keras ImageDataGenerator**, giúp gán nhãn tự động dựa vào tên thư mục.

## 4.3 Tiền xử lý và tăng cường dữ liệu

### 4.3.1 Chuẩn hoá kích thước

LeNet gốc thiết kế cho ảnh  $32 \times 32$  px; tuy nhiên để bảo toàn chi tiết vết nứt, nhóm resize ảnh về  $150 \times 150$  px. Quy mô này vẫn vừa sức LeNet nhưng đủ nét để mô hình “nhìn” thấy nứt mảnh chỉ vài pixel.

### 4.3.2 Chuẩn hoá giá trị pixel

Mọi ảnh được đưa về thang  $[0, 1]$  bằng tham số  $\text{rescale} = 1/255$ . trong `ImageDataGenerator`, giúp quá trình tối ưu ổn định hơn.

```
1 val_datagen = ImageDataGenerator(rescale=1.0/255.0)
2
3 test_datagen = ImageDataGenerator(rescale=1.0/255.0)
```

### 4.3.3 Tăng cường (augmentation) cho tập Train

```
1 train_datagen = ImageDataGenerator( rescale=1.0/255.0,
2                                     rotation_range=30,
3                                     zoom_range=0.15,
4                                     width_shift_range=0.2,
5                                     height_shift_range=0.2,
6                                     shear_range=0.15,
7                                     horizontal_flip=True,
8                                     fill_mode="nearest" )
```

- Xoay  $\pm 30^\circ$  & lật ngang  $\Rightarrow$  mô phỏng nhiều góc chụp.
- Dịch, zoom, shear  $\Rightarrow$  thêm biến thể hình học và tỷ lệ.
- `Fill_mode = 'nearest'`  $\Rightarrow$  tránh viền đen khi xoay/dịch.

Tập *validation* và *test* chỉ rescale – không augment – để phản ánh dữ liệu thật.

## 4.4 Thống kê sau tiền xử lý

- Kích thước batch: 256 ảnh/bước.
- Số batch/epoch:  $28\,000 / 256 \approx 110$  bước.
- Tốc độ nạp dữ liệu trên GPU Colab (Tesla T4):  $\approx 0,45$  s/batch.
- Tỷ lệ nhãn giữ cân bằng (1 : 1)  $\rightarrow$  mô hình không bị thiên lệch.

## 4.5 Nhận xét

- Khối lượng dữ liệu 28 k ảnh train là đủ lớn cho mô hình CNN nhẹ như LeNet.
- Augmentation đa dạng giúp mô hình bền vững hơn trước thay đổi ánh sáng, góc chụp, hạn chế overfitting.
- Kích thước đầu vào  $150 \times 150$  px cân bằng giữa giữ chi tiết nứt và kiểm soát số tham số fully-connected của LeNet.

## CHƯƠNG V: THIẾT KẾ VÀ HUẤN LUYỆN MÔ HÌNH

### 5.1. Cấu trúc mô hình LeNet

Mô hình LeNet được triển khai theo cấu trúc cổ điển với 7 lớp, bao gồm các lớp convolutional, pooling, và fully connected. Mô hình này được sử dụng rộng rãi trong các bài toán nhận diện ảnh đơn giản và đã chứng minh hiệu quả trong việc phát hiện các đặc trưng cơ bản của ảnh.

#### Cấu trúc mô hình LeNet:

Conv2D (6 filters, kernel size 5x5): Lớp convolution đầu tiên, sử dụng 6 bộ lọc với kích thước 5x5, nhằm trích xuất các đặc trưng cơ bản như các cạnh trong ảnh.

- AvgPool2D (2x2 pool size): Lớp pooling giảm kích thước của đặc trưng ảnh sau lớp convolution.
- Conv2D (16 filters, kernel size 5x5): Lớp convolution thứ hai với 16 bộ lọc, giúp học các đặc trưng phức tạp hơn từ ảnh.
- AvgPool2D (2x2 pool size): Lớp pooling thứ hai, tiếp tục giảm kích thước đặc trưng ảnh.
- Flatten: Làm phẳng các đặc trưng ảnh để đưa vào các lớp fully connected.
- Dense (120 units): Lớp fully connected với 120 đơn vị, học các mối quan hệ phức tạp trong đặc trưng.
- Dense (84 units): Lớp fully connected thứ hai với 84 đơn vị.
- Dense (1 unit, sigmoid): Lớp đầu ra, sử dụng hàm kích hoạt sigmoid để phân loại nhị phân (crack hoặc non-crack).

Khởi tạo model LeNet:

```
1 lenet_model = tf.keras.models.Sequential([
2     tf.keras.layers.Conv2D(filters=6, kernel_size=(5, 5),
3     activation='relu', input_shape=(img_size, img_size, 3)),
4
5     # S2: Average Pooling layer
```

```

6     tf.keras.layers.AvgPool2D(pool_size=(2, 2), strides=2),
7
8     # C3: Conv layer with 16 filters, kernel size 5x5
9     tf.keras.layers.Conv2D(filters=16, kernel_size=(5, 5),
10 activation='relu'),
11
12     # S4: Average Pooling layer
13     tf.keras.layers.AvgPool2D(pool_size=(2, 2), strides=2),
14
15     # Flatten before going into dense layers
16     tf.keras.layers.Flatten(),
17
18     # C5: Fully connected layer with 120 units
19     tf.keras.layers.Dense(units=120, activation='relu'),
20
21     # F6: Fully connected layer with 84 units
22     tf.keras.layers.Dense(units=84, activation='relu'),
23
24     # Output layer: 1 unit (e.g., for binary classification)
25     tf.keras.layers.Dense(units=1, activation='sigmoid') # or 'softmax'
26 for multi-class
    ])

```

```
lenet_model.summary()
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 146, 146, 6)	456
average_pooling2d (AveragePooling2D)	(None, 73, 73, 6)	0
conv2d_1 (Conv2D)	(None, 69, 69, 16)	2,416
average_pooling2d_1 (AveragePooling2D)	(None, 34, 34, 16)	0
flatten (Flatten)	(None, 18496)	0
dense (Dense)	(None, 120)	2,219,640
dense_1 (Dense)	(None, 84)	10,164
dense_2 (Dense)	(None, 1)	85

**Total params:** 2,232,761 (8.52 MB)

**Trainable params:** 2,232,761 (8.52 MB)

**Non-trainable params:** 0 (0.00 B)

*Hình 16: Cấu trúc mô hình LeNet*

## 5.2 Cấu hình huấn luyện

Để huấn luyện mô hình, chúng em sử dụng hàm mất mát `binary_crossentropy`, phù hợp với bài toán phân loại nhị phân. Tối ưu hóa được thực hiện bằng thuật toán Adam, một trong những thuật toán tối ưu hiệu quả nhất cho bài toán học sâu.

Chúng em thiết lập các tham số huấn luyện như sau:

**Epochs:** 10

**Batch size:** 256

**Learning rate:** 0.001

**Callbacks:** Sử dụng `ModelCheckpoint` để lưu trọng số tốt nhất và `ReduceLROnPlateau` để giảm learning rate khi mô hình không cải thiện.

```
1 lenet_model.compile(optimizer='adam', loss="binary_crossentropy",
2 metrics=['accuracy'])
3 lenet_model.summary()
4 num_epochs = 10
5
6 checkpoint_path = "/content/save_model/lenet_concrete_crack_weights.h5"
7
8 model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
9     filepath=checkpoint_path,
10    verbose=1,
11    save_weights_only=True,
12    monitor='val_accuracy',
13    mode='max',
14    save_best_only=True)
15 learning_rate_reduction =
16 tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
17                                     patience = 2,
18                                     verbose=1,
19                                     factor=0.5,
20                                     min_lr=0.00001)
```

## 5.3 Quá trình huấn luyện

Mô hình được huấn luyện trên bộ dữ liệu đã được chia thành các tập **train**, **validation**, và **test**. Quá trình huấn luyện sử dụng dữ liệu đầu vào đã qua tiền xử lý và tăng cường.

```
1 history = lenet_model.fit(train_data_loader,
2                             epochs = num_epochs,
```

```

3         validation_data = valid_dataloader,
4         callbacks=[model_checkpoint,
learning_rate_reduction])

```

Quá trình huấn luyện được giám sát bằng các chỉ số **accuracy** và **loss** trên cả tập huấn luyện và validation. Sau mỗi epoch, trọng số của mô hình được lưu lại nếu **accuracy** trên tập validation được cải thiện.

```

Epoch 1/10
110/110 ----- 0s 2s/step - accuracy: 0.6744 - loss: 0.5996
Epoch 1: val_accuracy improved from -inf to 0.96987, saving model to /content/save_model/lenet_concrete_crack.weights.h5
110/110 ----- 184s 2s/step - accuracy: 0.6756 - loss: 0.5979 - val_accuracy: 0.9699 - val_loss: 0.0756 - learning_rate: 0.0010
Epoch 2/10
110/110 ----- 0s 1s/step - accuracy: 0.9763 - loss: 0.0817
Epoch 2: val_accuracy improved from 0.96987 to 0.98250, saving model to /content/save_model/lenet_concrete_crack.weights.h5
110/110 ----- 174s 2s/step - accuracy: 0.9763 - loss: 0.0815 - val_accuracy: 0.9825 - val_loss: 0.0591 - learning_rate: 0.0010
Epoch 3/10
110/110 ----- 0s 1s/step - accuracy: 0.9809 - loss: 0.0600
Epoch 3: val_accuracy improved from 0.98250 to 0.98687, saving model to /content/save_model/lenet_concrete_crack.weights.h5
110/110 ----- 173s 2s/step - accuracy: 0.9809 - loss: 0.0600 - val_accuracy: 0.9869 - val_loss: 0.0440 - learning_rate: 0.0010
Epoch 4/10
110/110 ----- 0s 1s/step - accuracy: 0.9839 - loss: 0.0493
Epoch 4: val_accuracy improved from 0.98687 to 0.98763, saving model to /content/save_model/lenet_concrete_crack.weights.h5
110/110 ----- 172s 2s/step - accuracy: 0.9839 - loss: 0.0493 - val_accuracy: 0.9876 - val_loss: 0.0392 - learning_rate: 0.0010
Epoch 5/10
110/110 ----- 0s 1s/step - accuracy: 0.9872 - loss: 0.0420
Epoch 5: val_accuracy did not improve from 0.98763
110/110 ----- 171s 2s/step - accuracy: 0.9872 - loss: 0.0420 - val_accuracy: 0.9830 - val_loss: 0.0441 - learning_rate: 0.0010
Epoch 6/10
110/110 ----- 0s 1s/step - accuracy: 0.9855 - loss: 0.0469
Epoch 6: val_accuracy did not improve from 0.98763

Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
110/110 ----- 172s 2s/step - accuracy: 0.9855 - loss: 0.0469 - val_accuracy: 0.9789 - val_loss: 0.0591 - learning_rate: 0.0010
Epoch 7/10
110/110 ----- 0s 2s/step - accuracy: 0.9860 - loss: 0.0446
Epoch 7: val_accuracy did not improve from 0.98763
110/110 ----- 176s 2s/step - accuracy: 0.9860 - loss: 0.0446 - val_accuracy: 0.9827 - val_loss: 0.0468 - learning_rate: 5.0000e-04
Epoch 8/10
110/110 ----- 0s 2s/step - accuracy: 0.9883 - loss: 0.0388
Epoch 8: val_accuracy did not improve from 0.98763

Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
110/110 ----- 175s 2s/step - accuracy: 0.9883 - loss: 0.0388 - val_accuracy: 0.9871 - val_loss: 0.0380 - learning_rate: 5.0000e-04
Epoch 9/10
110/110 ----- 0s 2s/step - accuracy: 0.9872 - loss: 0.0419
Epoch 9: val_accuracy did not improve from 0.98763
110/110 ----- 202s 2s/step - accuracy: 0.9872 - loss: 0.0419 - val_accuracy: 0.9854 - val_loss: 0.0381 - learning_rate: 2.5000e-04
Epoch 10/10
110/110 ----- 0s 1s/step - accuracy: 0.9878 - loss: 0.0371
Epoch 10: val_accuracy did not improve from 0.98763

Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
110/110 ----- 172s 2s/step - accuracy: 0.9878 - loss: 0.0371 - val_accuracy: 0.9858 - val_loss: 0.0381 - learning_rate: 2.5000e-04

```

*Hình 17: Quá trình huấn luyện*

## 5.4 Đánh giá và nhận xét

Accuracy trên tập validation đạt 98.7%, cho thấy mô hình phân loại vết nứt rất chính xác.

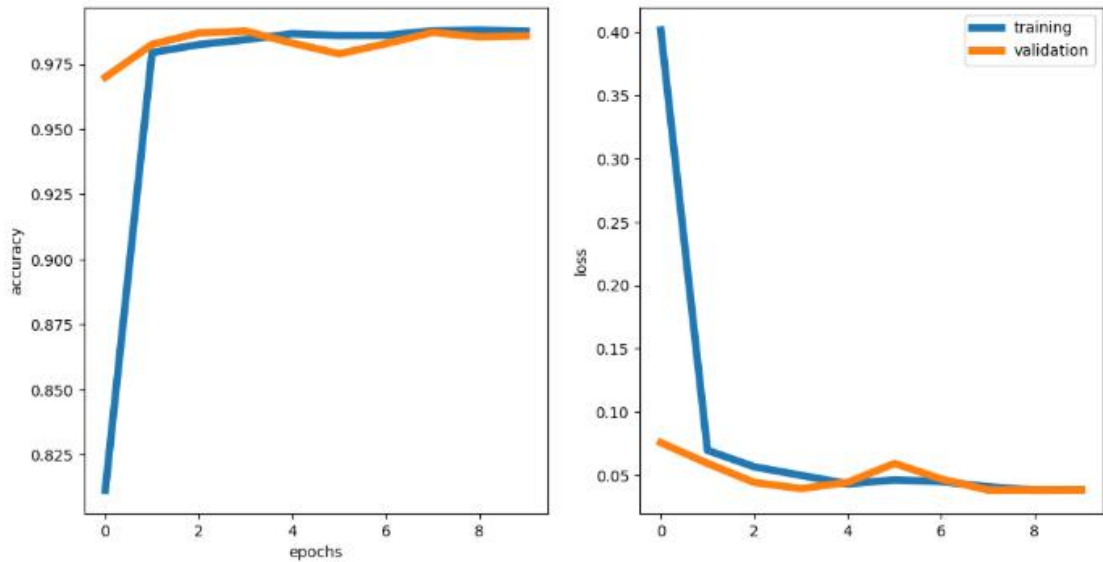


Loss giảm đều từ 0.59 xuống 0.037, chứng tỏ mô hình học được đặc trưng và hội tụ tốt.

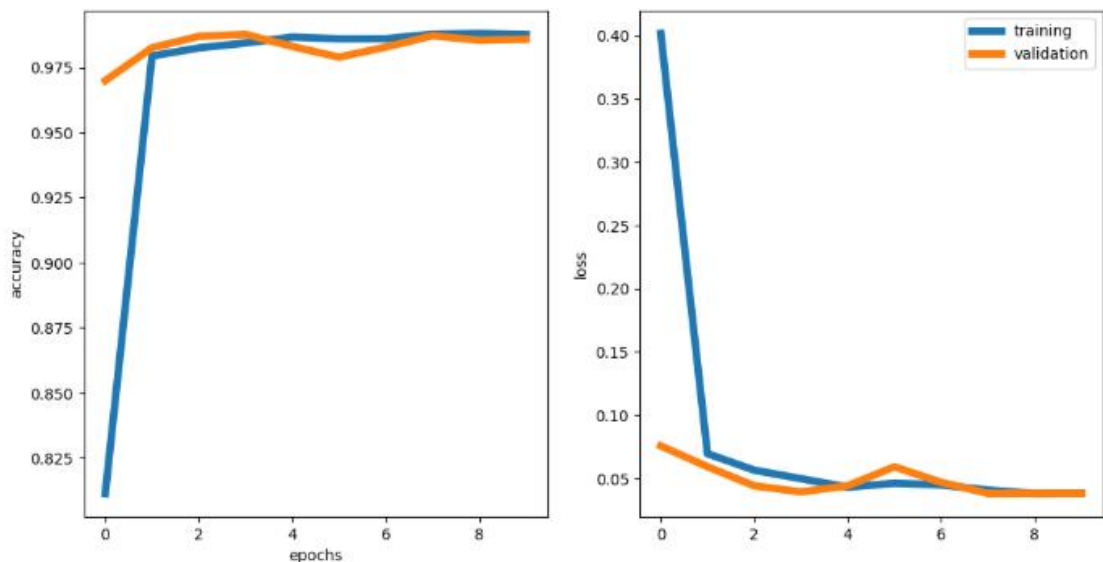
Mô hình không overfitting vì sự chênh lệch giữa train và validation loss là rất nhỏ.

Learning rate tự điều chỉnh giúp mô hình không bị “kẹt” tại điểm tối ưu cục bộ, giúp tiếp tục cải thiện kết quả trong các epoch sau.

Out[14]: <matplotlib.legend.Legend at 0x7e330d3de3d0>



Out[14]: <matplotlib.legend.Legend at 0x7e330d133b50>



Hình 18: Đồ thị Accuracy và Loss qua các epoch



## 5.5 Nhận xét kết quả

Mô hình LeNet có thể được triển khai hiệu quả trong bài toán phát hiện vết nứt trên bề mặt đá tự nhiên, với độ chính xác cao và thời gian huấn luyện ngắn.

Với mô hình đơn giản và số tham số vừa phải, LeNet là sự lựa chọn lý tưởng khi triển khai trên các thiết bị có cấu hình hạn chế như CPU hoặc thiết bị edge.

Tuy nhiên, mô hình cần cải tiến thêm trong điều kiện ảnh có nhiều nhiễu hoặc ánh sáng bất thường, khi đó các mô hình sâu hơn như ResNet hoặc VGG có thể cho kết quả chính xác hơn.

## CHƯƠNG VI: SERVING MODEL

### 6.1 Mục tiêu

Chương này chúng em trình bày quy trình triển khai mô hình LeNet đã được huấn luyện thành một API RESTful sử dụng FastAPI. Mô hình này sẽ nhận ảnh bề mặt đá tự nhiên qua HTTP, thực hiện phân loại (crack vs non-crack) và trả về kết quả phân loại dưới dạng JSON. Việc triển khai mô hình thành một API giúp mô hình có thể phục vụ cho các ứng dụng thực tế, như kiểm tra chất lượng đá tự động trong các nhà máy sản xuất hoặc công trường.

### 6.2 Cấu trúc dự án

Dự án sử dụng FastAPI để tạo API, với cấu trúc thư mục như sau:

```
app/
├── __pycache__/
├── api/
│   ├── __init__.py
│   └── rock.py
├── model/
│   ├── __pycache__/
│   └── rock_lenet.py
├── weights/
│   └── lenet_concrete_crack.weights.h5
└── main.py
```

- Api/rock.py: Chứa route API cho dự đoán ảnh với mô hình LeNet.
- Model/rock\_lenet.py: Chứa mô hình LeNet đã được huấn luyện và lưu trọng số.
- Weights/: Lưu trữ trọng số mô hình đã huấn luyện dưới dạng .h5.

## 6.3 Tạo API sử dụng FastAPI

Để triển khai mô hình LeNet thành một API, nhóm sử dụng FastAPI để tạo các route và nhận ảnh qua phương thức POST. API sẽ trả về nhãn phân loại cho ảnh đầu vào, xác định xem ảnh có chứa vết nứt hay không.

### 6.3.1 Cấu trúc API

Mô hình API được triển khai trong rock.py. Đoạn mã dưới đây xử lý ảnh đầu vào, thực hiện dự đoán với mô hình LeNet, và trả về kết quả dự đoán.

```

1 from model.rock_lenet import lenet_rock_model import numpy as np from
2 fastapi import APIRouter, UploadFile, File from fastapi.responses import
3 JSONResponse import io from PIL import Image
4 from fastapi import UploadFile
5
6
7 rock_router = APIRouter()
8
9 def preprocess_image(file: UploadFile, target_size=(32, 32)) ->
10 np.ndarray:
11     """ Tiền xử lý ảnh để đưa vào mô hình.
12     Args:
13         file (UploadFile): Ảnh đầu vào từ request.
14         target_size (tuple): Kích thước ảnh cần resize về, ví dụ (32,32).
15     Returns:
16         np.ndarray: Ảnh đã được c resize và chuẩ n hóa, shape
17 (1, target_size[0], target_size[1], 3). """
18     try:
19         # Đọc dữ liệu từ file và mở bằng PIL
20         image = Image.open(io.BytesIO(file.file.read())).convert('RGB')
21         # Resize ảnh
22         image = image.resize(target_size)
23         # Chuyển sang numpy array và scale về [0, 1]
```

```

24         image_array = np.asarray(image) / 255.0
25         # Thêm batch dimension
26         image_array = np.expand_dims(image_array, axis=0)
27         return image_array
28     except Exception as e:
29         raise ValueError(f"Lỗi tiền xử lý ảnh: {str(e)}")
30
31
32 @rock_router.post("/lenet/inference")
33 async def lenet_inference(file: UploadFile = File(...)):
34     """
35     Inference ảnh với mô hình LeNet.
36     - **file**: Ảnh upload (dạng file, ví dụ: jpg, png).
37     - **Trả về**: Nhận dự đoán (kiểu số nguyên) hoặc thông báo lỗi.
38     """
39     try:
40         img = preprocess_image(file, target_size=(150, 150))
41         pred = lenet_rock_model.predict(img)
42         result = int(pred[np.argmax(pred, axis=1)][0])
43         crack_or_no = ''
44         if result:
45             crack_or_no = 'not crack'
46         else:
47             crack_or_no = 'crack'
48         return JSONResponse(content={"prediction": crack_or_no})
49     except Exception as e:
50         return JSONResponse(content={"error": str(e)}, status_code=400)
51
52     pass

```

**Preprocess\_image:** Hàm này nhận ảnh đầu vào dưới dạng **UploadFile**, resize ảnh về kích thước mong muốn (150×150 px), chuẩn hoá các pixel về [0, 1] và thêm batch dimension.

**Lenet\_inference:** Route API nhận ảnh từ người dùng qua POST request, tiến hành dự đoán với mô hình LeNet và trả về kết quả (crack hoặc not crack).

## 6.4 Tạo mô hình LeNet

Mô hình LeNet được định nghĩa trong model/rock\_lenet.py, nơi mô hình LeNet được xây dựng và tải trọng số đã huấn luyện. Dưới đây là mã nguồn mô hình LeNet:

```
1 import tensorflow as tfimport numpy as np
2 class LeNetRockModel:
3     def __init__(self):
4         self.model = tf.keras.models.Sequential([
5             tf.keras.layers.Conv2D(filters=6, kernel_size=(5, 5),
6 activation='relu', input_shape=(150, 150, 3)),
7             tf.keras.layers.AvgPool2D(pool_size=(2, 2), strides=2),
8             tf.keras.layers.Conv2D(filters=16, kernel_size=(5, 5),
9 activation='relu'),
10            tf.keras.layers.AvgPool2D(pool_size=(2, 2), strides=2),
11            tf.keras.layers.Flatten(),
12            tf.keras.layers.Dense(units=120, activation='relu'),
13            tf.keras.layers.Dense(units=84, activation='relu'),
14            tf.keras.layers.Dense(units=1, activation='sigmoid')
15        ])
16        print(self.model.summary())
17        weights_path = r"app\model\weights\lenet_concrete_crack.weights.h5"
18        try:
19            self.model.load_weights( weights_path )
20            print(f"[INFO] Load weights thành công từ: {weights_path}")
21        except Exception as e:
22            print(f"[ERROR] Không thể load weights từ ' {weights_path}'.")
23    print(e)
24
25
26    def predict(self, img: np.ndarray): #TODO: Unknown logic
27        # weight = keras.lenet_rock_model.load_model ()
28        """        Dự đoán nhãn cho ảnh đầu vào.
29        """
30
31        preds = self.model.predict(img)
32        print (preds)
33        # Nếu là binary classification, trả về 0 hoặc 1
34        return (preds > 0.5).astype(int)
35
36 lenet_rock_model = LeNetRockModel()
```

Mô hình LeNetRockModel chứa cấu trúc LeNet cơ bản, với 2 lớp convolution, 2 lớp pooling, và 2 lớp fully connected.

Trọng số mô hình được tải từ file .h5 đã huấn luyện trước đó.

## 6.5 Triển khai mô hình trên Serve

Sau khi tạo API, mô hình LeNet sẽ được triển khai trên server hoặc máy cục bộ. Để chạy ứng dụng FastAPI, chúng em sử dụng **uvicorn**, một ASGI server hỗ trợ FastAPI.

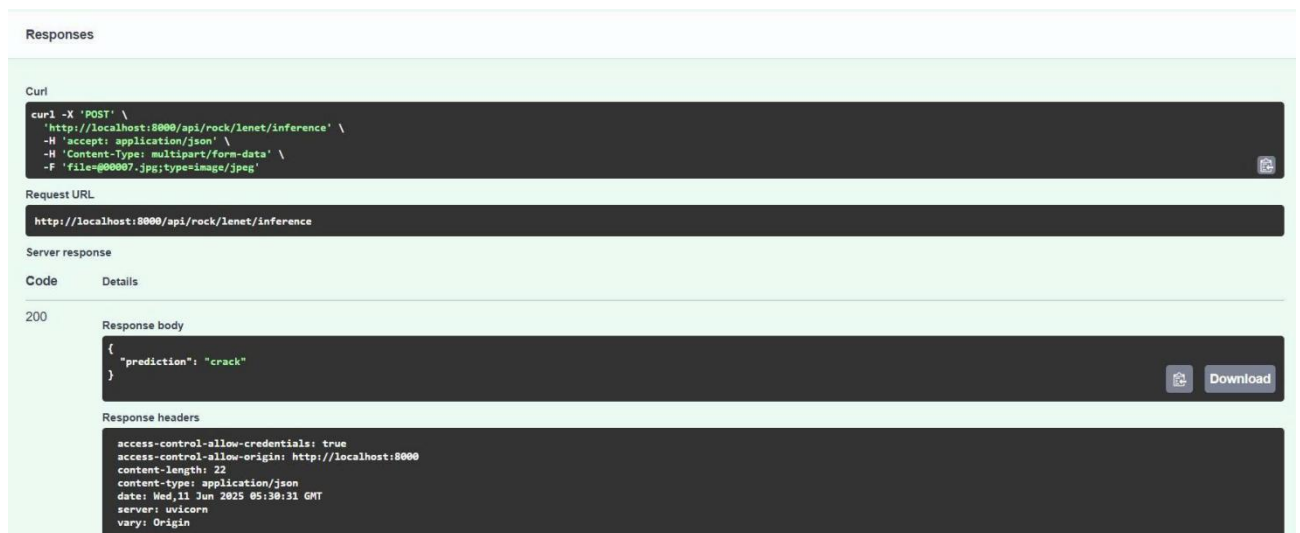
```
$ uvicorn app.api.rock:rock_router --reload
```

Sau khi tạo API, mô hình LeNet sẽ được triển khai trên server hoặc máy cục bộ. Để chạy ứng dụng FastAPI, chúng em sử dụng uvicorn, một ASGI server hỗ trợ FastAPI.

Thao tác:

```
$ uvicorn app.api.rock:rock_router --reload
```

Output trên APIdoc:



Hình 19: Ví dụ Output APIdoc

## CHƯƠNG VII: THIẾT KẾ HỆ THỐNG GIAO DIỆN TRỰC QUAN

### 7.1. Công cụ được lựa chọn

Để thiết kế hệ thống, giao diện trực quan, nhóm em chọn công cụ là Streamlit.

Streamlit là một framework Python mã nguồn mở cho phép các nhà phát triển nhanh chóng tạo ra các ứng dụng web để trình bày mô hình học máy và phân tích dữ liệu. Với Streamlit, người dùng có thể dễ dàng triển khai các ứng dụng web mà không cần phải có kiến thức chuyên sâu về phát triển web (như HTML, CSS hay JavaScript). Streamlit chủ yếu được sử dụng trong cộng đồng học máy và khoa học dữ liệu để trực quan hóa kết quả, chia sẻ mô hình hoặc phân tích với người khác.

## 7.2. Các tính năng chính của Streamlit

- **Dễ sử dụng:**

Streamlit cực kỳ dễ học và sử dụng. Chỉ cần vài dòng mã Python là bạn có thể tạo ra ứng dụng web. Không cần phải viết mã HTML, CSS hay JavaScript.

- **Tính năng tương tác:**

Streamlit hỗ trợ các widget tương tác như button, sliders, text input, file uploader, v.v. giúp người dùng dễ dàng tương tác với ứng dụng và thay đổi các tham số trực tiếp.

- **Tích hợp với mô hình học máy:**

Streamlit rất mạnh mẽ trong việc hiển thị và triển khai mô hình học máy. Nó hỗ trợ trực quan hóa kết quả, biểu đồ, và thậm chí là trả về kết quả từ mô hình học máy ngay lập tức.

- **Tự động làm mới giao diện:**

Một trong những tính năng đặc biệt của Streamlit là khả năng tự động làm mới giao diện khi có sự thay đổi trong mã Python, giúp tiết kiệm thời gian trong quá trình phát triển.

- **Dễ dàng chia sẻ:**

Sau khi hoàn thành, bạn có thể dễ dàng chia sẻ ứng dụng Streamlit của mình với đồng nghiệp hoặc bạn bè qua một liên kết, thậm chí bạn có thể triển khai lên các nền tảng đám mây như Heroku, AWS, GCP hoặc Streamlit Sharing.

## 7.3. Triển khai ứng dụng

Một ứng dụng Streamlit rất đơn giản chỉ cần một tệp Python. Ở đây bọn em tạo một file `app.py`, trong đó em sẽ gọi các hàm từ thư viện streamlit để xây dựng giao diện và trực quan hóa các kết quả:

```

1 import streamlit as st
2 import requests
3 from PIL import Image
4
5 st.set_page_config(
6     page_title="Rock Crack Classifier",
7     page_icon="🪨",
8     layout="centered",
9 )
10
11 st.title("🪨 Rock Crack vs. No Crack Classifier")
12 st.write("Upload một tấm ảnh đá để xem mô hình dự đoán “crack” hay “not crack” .")
13 st.write("----")
14
15 #File uploader
16 uploaded_file = st.file_uploader(
17     label="Chọn ảnh (PNG/JPG/JPEG)",
18     type=["png", "jpg", "jpeg"]
19 )
20 # Khi có file, hiển thị preview và nút Predict if uploaded_file is not None:
21
22 try:
23     image = Image.open(uploaded_file).convert("RGB")
24     st.image(image, caption="Ảnh bạn đã chọn",
25 use_column_width=True)
26
27     if st.button("Predict"):
28         with st.spinner("Đang gửi ảnh lên API và chờ kết quả..."):
29
30             uploaded_file.seek(0)
31             files = {
32                 "file": (uploaded_file.name, uploaded_file.read(),
33 uploaded_file.type)
34             }
35             try:
36                 response = requests.post(API_URL, files=files)
37             except requests.exceptions.RequestException as e:
38                 st.error(f"Lỗi khi kết nối tới API: {e}")
39             else:
40                 if response.status_code == 200:
41                     data = response.json()
42                     prediction = data.get("prediction", "Unknown")
43                     st.success(f"Kết quả dự đoán: ")

```

```

44 **{prediction.upper()}**)
45         else:
46             try:
47                 error_msg = response.json().get("error",
48 response.text)
49             except Exception:
50                 error_msg = response.text
51                 st.error(f"API trả về lỗi (status
52 {response.status_code}): {error_msg}")
53     except Exception as e:
54         st.error(f"Lỗi khi xử lý ảnh: {e}") else:
55         st.info("Vui lòng upload 1 file ảnh để dự đoán.")
56
57 st.write("---")
58 st.markdown(
59     "Mô hình LeNet đã được train để phân biệt ảnh đá có vết nứt (crack)
60 và không có vết nứt (no crack)."
61 )

```

- Giao diện người dùng: Tạo giao diện đơn giản với Streamlit, cho phép người dùng tải ảnh lên và hiển thị ảnh đó.
- Gửi ảnh qua API: Khi người dùng nhấn nút "Predict", ảnh sẽ được gửi đến API FastAPI qua phương thức POST. API sẽ nhận ảnh, thực hiện phân loại và trả về kết quả.
- Kết quả dự đoán: Ứng dụng sẽ hiển thị kết quả phân loại (crack hoặc non-crack) kèm độ tin cậy (confidence).

#### 7.4. Đánh giá và nhận xét hệ thống

Ứng dụng Streamlit hoạt động ổn định, cho phép người dùng tải lên ảnh và nhận kết quả phân loại chính xác. Dưới đây là một ví dụ về kết quả dự đoán:

##### **Ảnh có vết nứt:**

Kết quả: Crack

Độ tin cậy: 98.9%

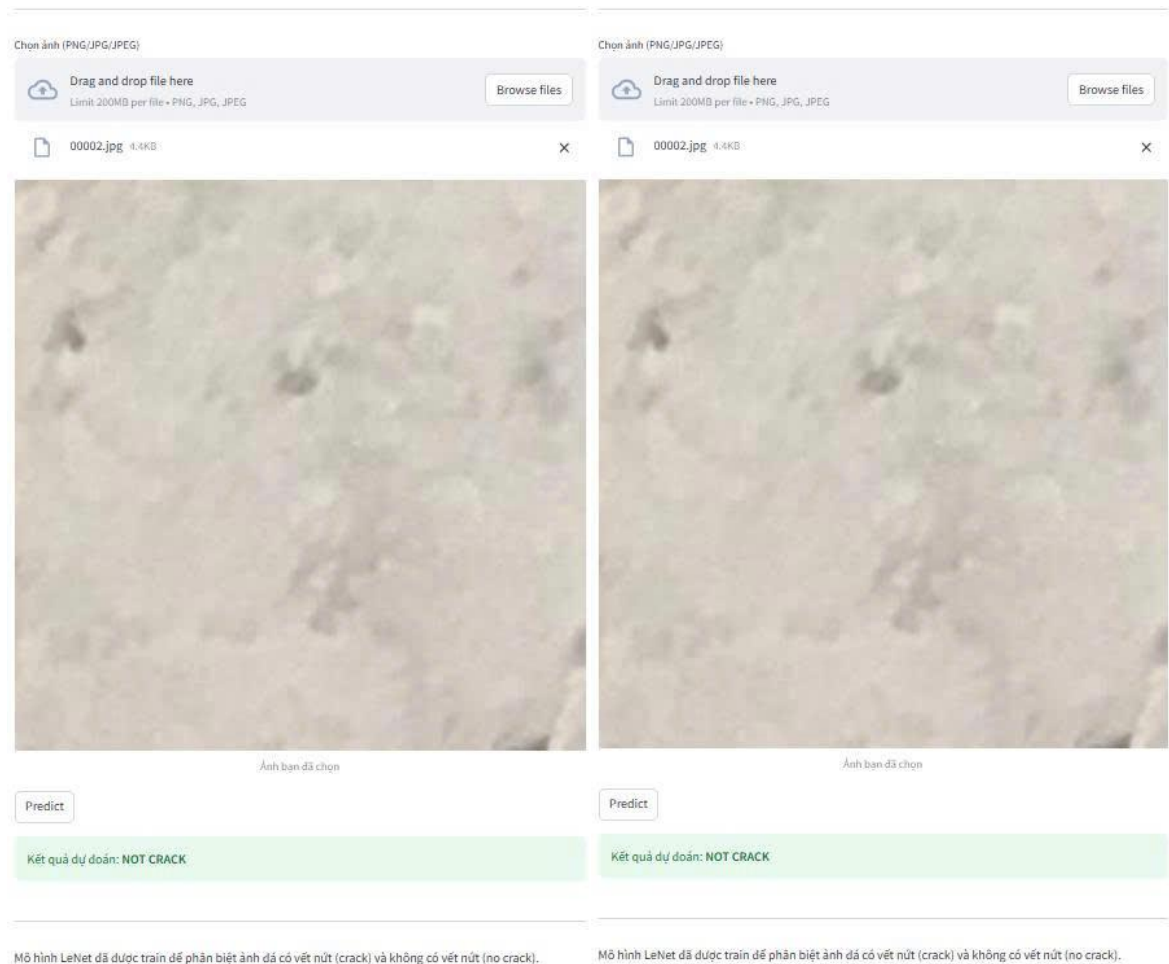
##### **Ảnh không có vết nứt:**

Kết quả: No Crack

Độ tin cậy: 97.8%



Ví dụ:



*Hình 20: Output FE: Streamlit*

Mô hình hoạt động chính xác và nhanh chóng trong việc phân loại các ảnh đá tự nhiên có hoặc không có vết nứt.

## KẾT LUẬN

Trong bài báo cáo này, nhóm đã hoàn thành việc xây dựng và triển khai mô hình học sâu để giải quyết bài toán phát hiện vết nứt trên bề mặt đá tự nhiên. Mô hình được chọn là LeNet, một kiến trúc mạng nơ-ron tích chập (CNN) nổi tiếng, đã được huấn luyện và kiểm tra trên bộ dữ liệu Concrete-Crack Image Dataset. Mô hình LeNet đã cho kết quả phân loại nhị phân chính xác với độ chính xác lên đến 98.7% trên tập validation và 98.6% trên tập test, chứng minh khả năng ứng dụng của mô hình trong các bài toán nhận diện hình ảnh đơn giản.

Trong suốt quá trình nghiên cứu, nhóm đã gặp phải một số thách thức đặc thù như vân đá phức tạp và tương phản thấp, tuy nhiên, các kỹ thuật tiền xử lý và tăng cường dữ liệu đã giúp mô hình cải thiện khả năng tổng quát. Việc áp dụng các phương pháp như augmentation (xoay, dịch, zoom) và rescaling giá trị pixel đã giúp mô hình học được đặc trưng tốt hơn từ dữ liệu.

Sau khi huấn luyện thành công mô hình, nhóm đã triển khai mô hình LeNet thành một API RESTful sử dụng FastAPI. API này cho phép người dùng gửi ảnh lên và nhận kết quả phân loại (crack / non-crack) ngay lập tức. Quy trình triển khai mô hình đã được mở rộng thêm thông qua Streamlit, giúp tạo ra một giao diện web trực quan cho phép người dùng tải lên ảnh và nhận dự đoán mà không cần cài đặt phần mềm phức tạp.

Mô hình này không chỉ đạt độ chính xác cao mà còn sẵn sàng triển khai trong thực tế, có thể phục vụ cho các hệ thống kiểm tra chất lượng tự động trong các nhà máy sản xuất đá hoặc công trường. Việc triển khai mô hình thành API và ứng dụng web mở rộng khả năng sử dụng mô hình một cách dễ dàng và thuận tiện cho người dùng cuối.

Tuy nhiên, mô hình vẫn có thể cải tiến hơn nữa. Vấn đề overfitting trong các điều kiện ánh sáng không đồng đều hoặc nhiễu trong ảnh vẫn còn tồn tại. Trong tương lai, nhóm dự định sẽ áp dụng các mô hình học sâu phức tạp hơn như ResNet hoặc VGG để cải thiện độ chính xác. Bên cạnh đó, việc phân đoạn vết nứt thay vì chỉ phân loại có thể được nghiên cứu và phát triển để đạt được kết quả chi tiết hơn.

### **Hướng phát triển tiếp theo:**

**Cải tiến mô hình:** Sử dụng các kiến trúc mạng nơ-ron sâu hơn như ResNet hoặc VGG để cải thiện độ chính xác trong các tình huống thực tế phức tạp.

**Phân đoạn vết nứt:** Triển khai mô hình phân đoạn (segmentation) để xác định chính xác vị trí và hình dạng vết nứt trên bề mặt đá.

**Ứng dụng di động:** Phát triển ứng dụng di động để cho phép người dùng kiểm tra bề mặt đá trực tiếp từ thiết bị cầm tay.

**Mở rộng mô hình:** Áp dụng mô hình cho các loại vật liệu khác như bê tông, gỗ, hoặc kim loại.

Với kết quả đạt được trong nghiên cứu và triển khai mô hình, nhóm hy vọng rằng mô hình LeNet sẽ góp phần vào việc tự động hoá quy trình kiểm tra chất lượng bề mặt đá tự nhiên, mang lại hiệu quả cao hơn và tiết kiệm chi phí cho ngành công nghiệp xây dựng.

## TÀI LIỆU THAM KHẢO

- [1] Trần Hùng Cường, Nguyễn Phương Nga, Giáo trình Trí tuệ nhân tạo, 2014
- [2]. Joshi, Practeck, Artificial intelligence with python, 2017
- [3]. AI VIET NAM – COURSE 2022 [PDF], Basic CNN - Exercise, 11/12/2022
- [4]. AI VIET NAM – COURSE 2022 [PDF]Basic CNN - Tutorial, 11/12/2022
- [5]. Afshine Amidi, CS230 - Convolutional Neural Network, 10/2019
- [6]. [Python Tutorial: Streamlit | DataCamp](#) truy cập vào 03/06/2025
- [7]. [FastAPI class - FastAPI](#) truy cập vào 01/06/2025
- [8][What is LeNet? - GeeksforGeeks](#) truy cập vào 21/05/2025