BỘ NÔNG NGHIỆP VÀ MÔI TRƯỜNG

PHÂN HIỆU TRƯỜNG ĐẠI HỌC THỦY LỢI

BỘ MÔN CÔNG NGHỆ THÔNG TIN

TÊN ĐỀ TÀI

NHẬN DIỆN BỆNH TRÊN LÁ CÂY THỰC VẬT

Giảng viên:	Ths. Vũ Thị Hạnh
Sinh viên thực hiện:	2351267280 Đoàn Anh Vũ 2351267274 Nguyễn Hữu Tuấn Phát 2351267259 Phạm Thị Quỳnh Giao
Lớp:	S26-65TTNT

TP. Hồ Chí Minh, ngày … tháng … năm 2025

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

TP. Hồ Chí Minh, ngày … tháng … năm 2025

Chữ ký của giảng viên

LỜI CẢM ƠN

Lời đầu tiên, nhóm em xin bày tỏ lòng biết ơn sâu sắc và chân thành nhất đến quý thầy cô Bộ môn Công nghệ Thông tin – Phân hiệu Trường Đại học Thủy Lợi, những người đã truyền đạt kiến thức và tạo điều kiện thuận lợi nhất cho nhóm trong suốt quá trình học tập và nghiên cứu.

Đặc biệt, nhóm em xin gửi lời cảm ơn chân thành đến cô Vũ Thị Hạnh . Với sự hướng dẫn tận tâm, những định hướng chuyên môn quý báu và sự khích lệ của Thầy, nhóm em đã có thêm động lực để vượt qua những khó khăn về mặt thuật toán và xử lý dữ liệu để hoàn thành đề tài này.

Bên cạnh đó, nhóm cũng xin cảm ơn cộng đồng Kaggle cùng tác giả bộ dữ liệu "New Plant Diseases Dataset" đã cung cấp nguồn tài nguyên quý giá, giúp nhóm có cơ sở thực chứng để triển khai mô hình phân loại bệnh lá cây một cách khách quan nhất.

Mặc dù đã dành nhiều tâm huyết cho bài báo cáo, nhưng do hạn chế về mặt thời gian và kinh nghiệm trong lĩnh vực Machine Learning, bài làm chắc chắn không tránh khỏi những thiếu sót. Nhóm em rất mong nhận được những ý kiến đóng góp từ Thầy để nhóm có cơ hội hoàn thiện kiến thức và phát triển kỹ năng nghiên cứu trong tương lai.

Nhóm em xin chân thành cảm ơn!

Mục Lục

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT 6
DANH MỤC HÌNH ẢNH 7
DANH MỤC BẢNG 8
MỞ ĐẦU 9
A. GIỚI THIỆU ĐỀ TÀI 9
1. Bối cảnh đề tài 9
2. Ý nghĩa và mục tiêu thực hiện 9
B. YÊU CẦU BÀI TOÁN 9
1. Đối tượng và phạm vi nghiên cứu 9
2. Các yêu cầu kỹ thuật cần đạt được 10
C. GIỚI THIỆU TẬP DỮ LIỆU 10
1. Quy mô và cấu trúc dữ liệu 10
2. Tổ chức tập dữ liệu 13
3. Đặc điểm hình ảnh 13
4. Phân tích đặc điểm của từng bệnh 14
Chương 1: Tiền xử lý dữ liệu 20
1. Train_transform 20
2. Valid_transform 21
3. Chia Train/ valid 22
4. Phát hiện và thống kê các ảnh bị quá sáng (over-bright) 23
Chương 2: Tạo DataLoader & cấu hình huấn luyện 25
1. Tạo DataLoader 25
2. cấu hình huấn luyện 25
Chương 3: Định nghĩa mô hình 28
1. Tải MobileNetV3-Large pre-trained trên ImageNet. 28
2. Earrly stopping 28
Chương 4: Huấn luyện mô hình và đánh giá mô hình - MobileNet. 29
1. Định nghãi hàm validate 29

2. Chạy mô hình và Đánh giá. 29
3. Thử nghiệm mô hình với tập Test 32
Chương 5 Huấn luyện mô hình và đánh giá mô hình - EfficientNet. 34
1. Chạy mô hình và Đánh giá. 34
2. Thử nghiệm mô hình với tập Test 36
Chương 6 Huấn luyện mô hình và đánh giá mô hình - RESNET. 38
1. Huấn luyện mô hình và đánh giá 38
2. Thử nghiệm với tập test 40
Chương 7 Đánh giá sơ bộ kết quả 3 mô hình 42
1. Mô hình MobileNetV3-Large 42
2. Mô hình EfficientNet 42
3. Mô hình ResNet-18 42
Định hướng phát triển cho lần báo cáo cuối cùng 43

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

Bảng 1 Danh sách các nhãn 11

Bảng 2 Danh sách các bệnh và dấu hiệu nhận biết 14

MỞ ĐẦU

A. GIỚI THIỆU ĐỀ TÀI

1. Bối cảnh đề tài

Trong khuôn khổ môn học Machine Learning, việc tiếp cận với các bài toán thực tế thông qua các bộ dữ liệu quy chuẩn là yêu cầu trọng tâm để củng cố kiến thức lý thuyết. Đề tài "Phân loại bệnh hại trên lá cây sử dụng bộ dữ liệu New Plant Diseases Dataset" giúp sinh viên rèn luyện kỹ năng giải quyết bài toán thị giác máy tính (Computer Vision) – một trong những lĩnh vực quan trọng nhất của trí tuệ nhân tạo hiện nay.

2. Ý nghĩa và mục tiêu thực hiện

Việc thực hiện đề tài này không chỉ dừng lại ở việc hoàn thành nội dung môn học mà còn mang lại những giá trị cụ thể:

Về kiến thức: Giúp nhóm nắm vững quy trình xử lý dữ liệu hình ảnh quy mô lớn (hơn 87.000 mẫu) và hiểu sâu về cơ chế hoạt động của các thuật toán phân loại.

Về kỹ năng: Thực hành kỹ thuật xây dựng, huấn luyện và tối ưu hóa các mô hình học máy (như mạng CNN) để đạt được độ chính xác cao nhất trên dữ liệu thực tế.

Về thực tiễn: Tiếp cận với bài toán chuyển đổi số trong nông nghiệp, hiểu được cách công nghệ có thể hỗ trợ con người trong việc nhận diện và quản lý dịch bệnh cây trồng một cách tự động.

B. YÊU CẦU BÀI TOÁN

Dựa trên yêu cầu của giảng viên và đặc điểm của bộ dữ liệu được cung cấp, bài toán được xác định cụ thể như sau:

1. Đối tượng và phạm vi nghiên cứu

Dữ liệu đầu vào: Bộ dữ liệu New Plant Diseases Dataset bao gồm 38 lớp (classes) khác nhau, đại diện cho nhiều loại cây trồng (cà chua, khoai tây, táo, nho, ngô...) ở cả hai trạng thái: khỏe mạnh và bị bệnh.

Phạm vi: Tập trung vào bài toán phân loại hình ảnh (Image Classification) dựa trên các đặc trưng về màu sắc, hình dạng và kết cấu của vết bệnh trên lá.

2. Các yêu cầu kỹ thuật cần đạt được

Để đáp ứng tiêu chuẩn của môn học, nhóm cần giải quyết các yêu cầu sau:

Tiền xử lý dữ liệu: Thực hiện chuẩn hóa (Normalization), tái cấu trúc kích thước ảnh và áp dụng các kỹ thuật tăng cường dữ liệu (Data Augmentation) để đảm bảo mô hình có tính tổng quát hóa tốt.

Thiết kế mô hình: Xây dựng cấu trúc mạng Neural phù hợp (ví dụ: CNN hoặc sử dụng các mô hình tiền huấn luyện qua Transfer Learning) để trích xuất đặc trưng hình ảnh hiệu quả.

Tối ưu hóa: Lựa chọn hàm mất mát (Loss function) và bộ tối ưu hóa (Optimizer) phù hợp để cực tiểu hóa sai số trong quá trình huấn luyện.

Đánh giá kết quả: * Mô hình phải đạt được độ chính xác (Accuracy) mục tiêu trên tập kiểm thử (Test set).

C. GIỚI THIỆU TẬP DỮ LIỆU

Bộ dữ liệu được sử dụng trong đề tài là "New Plant Diseases Dataset", một phiên bản mở rộng và cải tiến từ bộ dữ liệu gốc PlantVillage. Đây là một trong những nguồn dữ liệu chuẩn và phổ biến nhất hiện nay dành cho việc nghiên cứu nhận diện bệnh cây trồng qua hình ảnh.

1. Quy mô và cấu trúc dữ liệu

Tổng số lượng hình ảnh: Khoảng 87,907 hình ảnh chất lượng cao.

Số lượng lớp (Classes): Bộ dữ liệu bao gồm 38 lớp khác nhau.

Đối tượng cây trồng: Bao gồm nhiều loại cây nông nghiệp quan trọng như: Táo, Nho, Ngô, Khoai tây, Cà chua, Đào, Anh đào, Dâu tây, Cam và Việt quất.

Trạng thái phân loại: Mỗi loại cây được chia thành các trạng thái:

Cây khỏe mạnh (Healthy).

Cây bị bệnh (ví dụ: Bệnh sương mai, bệnh đốm lá, bệnh rỉ sắt, bệnh cháy lá...).

Bảng 1 Danh sách các nhãn

STT	Nhóm Cây Trồng	Tên Lớp Dữ Liệu (Class Name)	Tình trạng / Tên bệnh
1	Cây Táo	Apple___Apple_scab	Bệnh ghẻ táo
2		Apple___Black_rot	Bệnh thối đen
3		Apple___Cedar_apple_rust	Bệnh rỉ sắt táo
4		Apple___healthy	Khỏe mạnh
5	Cây Anh đào	Cherry___Powdery_mildew	Bệnh phấn trắng
6		Cherry___healthy	Khỏe mạnh
7	Cây Ngô (Bắp)	Corn___Cercospora_leaf_spot Gray_leaf_spot	Bệnh đốm lá xám
8		Corn___Common_rust	Bệnh rỉ sắt
9		Corn___Northern_Leaf_Blight	Bệnh cháy lá ngô
10		Corn___healthy	Khỏe mạnh
11	Cây Nho	Grape___Black_rot	Bệnh thối đen
12		Grape___Esca_(Black_Measles)	Bệnh Esca (lốm đốm đen)
13		Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	Bệnh cháy lá
14		Grape___healthy	Khỏe mạnh
15	Cây Cam	Orange___Haunglongbing_(Citrus_greening)	Bệnh vàng lá gân xanh
16	Cây Đào	Peach___Bacterial_spot	Bệnh đốm lá vi

STT	Nhóm Cây Trồng	Tên Lớp Dữ Liệu (Class Name)	Tình trạng / Tên bệnh
			khuẩn
17		Peach___healthy	Khỏe mạnh
18	Cây Ớt chuông	Pepper,_bell___Bacterial_spot	Bệnh đốm lá vi khuẩn
19		Pepper,_bell___healthy	Khỏe mạnh
20	Cây Khoai tây	Potato___Early_blight	Bệnh đốm vòng (sớm)
21		Potato___Late_blight	Bệnh sương mai (muộn)
22		Potato___healthy	Khỏe mạnh
23	Cây Việt quất	Blueberry___healthy	Khỏe mạnh
24	Cây Mâm xôi	Raspberry___healthy	Khỏe mạnh
25	Cây Đậu nành	Soybean___healthy	Khỏe mạnh
26	Cây Bí ngòi	Squash___Powdery_mildew	Bệnh phấn trắng
27	Cây Dâu tây	Strawberry___Leaf_scorch	Bệnh cháy lá dâu tây
28		Strawberry___healthy	Khỏe mạnh
29	Cây Cà chua	Tomato___Bacterial_spot	Bệnh đốm lá vi khuẩn
30		Tomato___Early_blight	Bệnh đốm vòng (sớm)

STT	Nhóm Cây Trồng	Tên Lớp Dữ Liệu (Class Name)	Tình trạng / Tên bệnh
31		Tomato___Late_blight	Bệnh sương mai (muộn)
32		Tomato___Leaf_Mold	Bệnh nấm mốc lá
33		Tomato___Septoria_leaf_spot	Bệnh đốm lá Septoria
34		Tomato___Spider_mites Two-spotted_spider_mite	Nhện đỏ hại lá
35		Tomato___Target_Spot	Bệnh đốm mục tiêu
36		Tomato___Tomato_Yellow_Leaf_Curl_Virus	Virus xoăn vàng lá
37		Tomato___Tomato_mosaic_virus	Virus khảm cà chua
38		Tomato___healthy	Khỏe mạnh

2. Tổ chức tập dữ liệu

Dữ liệu đã được tác giả phân chia sẵn thành các thư mục để phục vụ quá trình huấn luyện mô hình, cụ thể:

Train set (Tập huấn luyện): Chiếm khoảng 80% tổng lượng ảnh (70,295 ảnh). Được sử dụng để mô hình học các đặc trưng hình ảnh của từng loại bệnh.

Validation set (Tập thẩm định): Chiếm khoảng 20% (17,572 ảnh). Được sử dụng để tinh chỉnh các tham số và kiểm tra độ chính xác sau mỗi chu kỳ huấn luyện (epoch).

3. Đặc điểm hình ảnh

Định dạng: Hình ảnh màu (RGB).

Kích thước: Các ảnh gốc có kích thước đồng nhất (256 x 256) , giúp đơn giản hóa quá trình tiền xử lý.

Môi trường chụp: Hình ảnh được chụp trong các điều kiện ánh sáng khác nhau, trên các nền trơn hoặc hậu cảnh tự nhiên, giúp mô hình tăng khả năng nhận diện trong môi trường thực tế.

4. Phân tích đặc điểm của từng bệnh

Data gồn 38 nhãn trong đó có 12 nhãn là lây khỏe mạnh và 26 cây có nhãn là cây bệnh

Dấu hiệu nhận biết 26 loại bệnh:

Bảng 2 Danh sách các bệnh và dấu hiệu nhận biết

STT	Tên tiếng Việt	Tên tiếng Anh	Dấu hiệu nhận biết	Nguyên nhân
1	Bệnh ghẻ táo	Apple Scab	Vết đốm màu nâu hoặc ô liu, bề mặt lá sần sùi, nứt nẻ.	Nấm
2	Bệnh thối đen	Apple Black Rot	Vết đen vòng tròn đồng tâm trên lá, quả thối đen.	Nấm
3	Bệnh rỉ sắt táo	Cedar Apple Rust	Vết vàng cam rực rỡ trên mặt lá, có chấm đỏ nhỏ ở giữa.	Nấm
4	Bệnh phấn trắng sơ-ri	Cherry Powdery Mildew	Lớp bột trắng mịn bao phủ lá, làm lá cuốn lại.	Nấm
5	Đốm lá xám (Ngô)	Gray Leaf Spot	Vết bệnh hình chữ nhật dài, màu xám chạy dọc gân lá.	Nấm
6	Bệnh rỉ sắt ngô	Common Rust	Các nốt mụn nhỏ màu nâu đỏ, chứa đầy bào tử bột.	Nấm
7	Cháy lá ngô	Northern Leaf	Vết bệnh hình thoi lớn, màu xám	Nấm

STT	Tên tiếng Việt	Tên tiếng Anh	Dấu hiệu nhận biết	Nguyên nhân
		Blight	bạc hoặc nâu xám.	
8	Bệnh thối đen nho	Grape Black Rot	Vết đốm nhỏ màu nâu sáng, có viền đen xung quanh.	Nấm
9	Bệnh Esca (Lốm đốm)	Grape Esca	Vết cháy vàng cam giữa các gân lá tạo hình "vằn hổ".	Nấm
10	Bệnh cháy lá nho	Grape Leaf Blight	Vết bệnh màu nâu không đều, lá khô dần và rụng.	Nấm
11	Vàng lá gân xanh	Citrus Greening	Lá vàng loang lổ, gân lá sưng xanh, quả nhỏ biến dạng.	Vi khuẩn
12	Đốm lá vi khuẩn (Đào)	Peach Bacterial Spot	Vết đốm nhỏ sũng nước, sau đó khô và tạo thành lỗ thủng.	Vi khuẩn
13	Đốm vi khuẩn (Ớt)	Bell Pepper Bacterial Spot	Vết thương nhỏ màu xanh vàng, sau thâm đen sần sùi.	Vi khuẩn
14	Bệnh đốm vòng (Sớm)	Early Blight	Vết đen có các vòng tròn đồng tâm như bia bắn (táo/cà chua).	Nấm
15	Bệnh sương mai	Late Blight	Vết thâm sũng nước, mặt dưới lá có lớp nấm trắng mịn.	Nấm
16	Bệnh phấn trắng (Bí)	Squash Powdery Mildew	Các mảng trắng như bụi phấn phủ kín bề mặt lá.	Nấm
17	Bệnh cháy lá dâu tây	Strawberry Leaf Scorch	Vết đốm màu tím sẫm, sau lan rộng thành mảng cháy khô.	Nấm
18	Đốm vi khuẩn	Tomato Bacterial	Vết đen nhỏ li ti, có quầng vàng	Vi khuẩn

STT	Tên tiếng Việt	Tên tiếng Anh	Dấu hiệu nhận biết	Nguyên nhân
	(Cà chua)	Spot	xung quanh đốm.	
19	Bệnh nấm mốc lá	Tomato Leaf Mold	Mặt trên lá có vết vàng, mặt dưới có lớp nấm màu ô liu.	Nấm
20	Đốm lá Septoria	Tomato Septoria Leaf Spot	Vết đốm tròn nhỏ, tâm màu xám trắng, viền nâu sẫm.	Nấm
21	Nhện đỏ hại lá	Tomato Spider Mites	Các đốm trắng li ti (vết chích), lá chuyển vàng xám.	Côn trùng
22	Bệnh đốm mục tiêu	Tomato Target Spot	Vết đốm nâu tròn có nhiều vòng tròn đồng tâm rõ rệt.	Nấm
23	Virus xoăn vàng lá	Yellow Leaf Curl Virus	Lá nhỏ lại, xoăn tít, mép lá vàng và cuộn hướng lên.	Virus
24	Virus khảm cà chua	Tomato Mosaic Virus	Lá có màu xanh đậm và nhạt đan xen như hình khảm.	Virus
25	Bệnh phấn trắng (Nho)	Grape Powdery Mildew	Các vết mốc trắng mỏng xuất hiện trên cả hai mặt lá.	Nấm
26	Bệnh đốm mục tiêu (Ngô)	Corn Target Spot	Vết đốm tròn màu nâu có vòng đồng tâm.	Nấm

Tổng hợp triệu chứng của lá cây thực vật theo nguyên nhân

Bệnh do nấm phổ biến với triệu chứng rõ ràng (vết tròn, chấm nhỏ, mốc trắng/vàng, cháy ở mép) sẽ dễ tách đặc trưng cho mô hình học

hỏi và nhận diện.

Hình 4.1 Apple Black Rot

Hình 4.2 Grape Black Rot

Hình 4.3 Corn Northern Leaf Blight

Bệnh do vi khuẩn có triệu chứng là vệt nâu, viền vàng đặc trưng cũng dễ tách đặc trưng cho mô hình học hỏi và nhận diện

Hình 4.4 Tomato___Bacterial_spot

Hình 4.5 Peach___Bacterial_spot

Bệnh do côn trùng có triệu chứng là chấm vàng li ti, có lớp phủ như mạng tơ, lá khô

Hình 4.6 Tomato___Spider_mites Two-spotted_spider_mite

Bệnh do virus riêng biệt có triệu chứng là lá quăn, vệt vàng, lá bị biến dạng.

Hình 4.7 Tomato___Tomato_mosaic_virus

Chương 1: Tiền xử lý dữ liệu

1. Train_transform

```python
train_transform = transforms.Compose([
    transforms.Resize((256,256)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

✓ 0.0s Python

1.1. transforms.Resize((256, 256))

Thay đổi kích thước tất cả ảnh về đúng 256x256 pixel.

Lý do: MobileNetV3 (và hầu hết các mô hình CNN hiện đại) yêu cầu input cố định. Thường resize lên 256 trước, sau đó có thể crop ngẫu nhiên (nhưng ở đây không có crop).

1.2. transforms.RandomHorizontalFlip()

Lật ngang ảnh ngẫu nhiên với xác suất 50%.

Lợi ích: Tăng dữ liệu, giúp mô hình không phụ thuộc vào vị trí trái/phải (rất hữu ích với ảnh tự nhiên như động vật, đồ vật...).

1.3. transforms.RandomRotation(10)

Xoay ngẫu nhiên ảnh trong khoảng ±10 độ.

Giúp mô hình học được đặc trưng bất kể góc chụp hơi lệch – tăng tính robust.

1.4. transforms.ColorJitter(brightness=0.2, contrast=0.2)

Ngẫu nhiên thay đổi độ sáng (brightness) và độ tương phản (contrast) trong khoảng ±20%.

Không thay đổi hue/saturation ở đây giúp tránh làm màu sắc bị méo quá mức.

Rất hữu ích khi dữ liệu có ảnh chụp ở điều kiện ánh sáng khác nhau (như bạn từng kiểm tra ảnh quá sáng).

1.5. transforms.ToTensor()

Chuyển ảnh từ định dạng PIL Image (hoặc numpy) sang PyTorch Tensor.

Đồng thời chuẩn hóa giá trị pixel từ [0, 255] → [0.0, 1.0].

1.6. transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

Chuẩn hóa từng kênh màu (R, G, B) bằng công thức: (pixel - mean) / std

Giá trị mean và std này là thống kê chính thức của ImageNet (tập dữ liệu mà MobileNetV3 được pre-trained).

Lý do bắt buộc dùng đúng giá trị này:

- Mô hình pre-trained đã học trên dữ liệu được normalize như vậy.
- Nếu không normalize đúng → hiệu suất giảm mạnh (accuracy có thể tụt 10-20%).

2. Valid_transform

```python
val_transform = transforms.Compose([
    transforms.Resize((256,256)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

✓ 0.0s Python

2.1. transforms.Resize((256, 256))

Thay đổi kích thước tất cả ảnh về đúng 256×256 pixel.

Đảm bảo input có kích thước thống nhất, phù hợp với mô hình.

2.2. transforms.ToTensor()

Chuyển ảnh từ định dạng PIL Image sang PyTorch Tensor.

Đồng thời chuẩn hóa giá trị pixel từ [0, 255] → [0.0, 1.0].

2.3. transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

Chuẩn hóa từng kênh R, G, B theo công thức (pixel - mean) / std.

Đây chính là giá trị mean và std của ImageNet – tập dữ liệu mà MobileNetV3-Large được pre-trained.

Bắt buộc phải dùng đúng giá trị này để mô hình hoạt động đúng như kỳ vọng.

3. Chia Train/ valid

```
from torch.utils.data import random_split
full_ds = ImageFolder(train_path, transform=train_transform)
train_size = int(0.8 * len(full_ds))
val_size = len(full_ds) - train_size
train_ds, valid_ds = random_split(full_ds, [train_size, val_size], generator=torch.Generator().manual_se

valid_ds.dataset.transform = val_transform
test_ds = ImageFolder(test_path, transform=val_transform)
```

✓ 0.2s Python

3.1. full_ds = ImageFolder(train_path, transform=train_transform)

Dùng torchvision.datasets.ImageFolder để tự động tạo dataset từ cấu trúc thư mục:

Áp dụng train_transform (có augmentation: flip, rotation, color jitter...) cho toàn bộ dữ liệu train gốc.

3.2. Chia train/validation (80/20)

train_size = int(0.8 * len(full_ds)): 80% làm train.

val_size = còn lại: 20% làm validation.

random_split(...): Chia ngẫu nhiên dataset thành 2 phần con.

generator=torch.Generator().manual_seed(42): → Đặt seed cố định → mỗi lần chạy code đều chia giống nhau → kết quả huấn luyện và đánh giá có thể tái lập (reproducible).

3.3 valid_ds.dataset.transform = val_transform

Đây là bước rất quan trọng và dễ bị bỏ sót

random_split tạo ra các Subset, nhưng tất cả chúng đều chia sẻ cùng một dataset gốc (full_ds).

Ban đầu, full_ds dùng train_transform → nếu không sửa, valid_ds cũng sẽ bị augmentation (flip, xoay ngẫu nhiên...) → đánh giá sai lệch.

Dòng này ghi đè transform chỉ cho phần validation thành val_transform (chỉ resize + normalize, không augmentation) → đánh giá chính xác, ổn định.

3.4 test_ds = ImageFolder(test_path, transform=val_transform)

Tạo dataset riêng cho tập test (thư mục độc lập).

Dùng cùng val_transform → tiền xử lý giống validation (không augmentation).

4. Phát hiện và thống kê các ảnh bị quá sáng (over-bright)

```
def is_over_bright(img_path, thresh=0.85):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
    v = hsv[:, :, 2]/ 255.0
    mean_v = v.mean()
    return mean_v > thresh, mean_v


from tqdm import tqdm
over_stat = []
over_count = Counter()

for img_path, label in tqdm(full_ds.samples):
    is_over, mean_v = is_over_bright(img_path)
    if is_over:
        class_name = class_names[label]
        over_count[class_name] += 1
    over_stat.append((mean_v))

df_over = pd.DataFrame.from_dict({
    "Class": list(over_count.keys()),
    "Total Images": [class_count[c] for c in over_count],
    "Over Bright Images": [over_count[c] for c in over_count],
})

df_over['Ratio'] = df_over['Over Bright Images'] / df_over['Total Images']
df_over = df_over.sort_values(by="Ratio", ascending=False)

df_over.head()
```

Hình 4.1 Phát hiện và thống kê các ảnh bị quá sáng_1

100%|██████████| 70295/70295 [00:49<00:00, 1426.37it/s]

	Class	Total Images	Over Bright Images	Ratio
9	Tomato___Tomato_mosaic_virus	1790	295	0.164804
6	Raspberry___healthy	1781	283	0.158899
3	Peach___healthy	1728	239	0.138310
0	Apple___Cedar_apple_rust	1760	206	0.117045
4	Pepper,_bell___healthy	1988	200	0.100604

Hình 4.1 Phát hiện và thống kê các ảnh bị quá sáng_2

Tỉ lệ ảnh bị over bright khoảng gần 2%, rất thấp và không ảnh hưởng quá nhiều tới mô hình

Chương 2: Tạo DataLoader & cấu hình huấn luyện

1. Tạo DataLoader

```
train_load = DataLoader(
    train_ds,
    batch_size=64,
    shuffle=True,
    num_workers=8,
    pin_memory=True,
)

valid_load = DataLoader(
    valid_ds,
    batch_size=64,
    shuffle=False,
    num_workers=8,
    pin_memory=True,
```

Hình 1.Tạo DataLoader

1.1 train_load giúp tạo dòng chảy dữ liệu cho tập huấn luyện.

Mỗi lần lấy 64 mẫu (ảnh + nhãn) cùng lúc để đưa vào mô hình

Có sự xáo trộn dữ liệu ở mỗi epoch giúp huấn luyện tốt hơn

Với num_workers = 8 có nghĩa là sử dụng 8 processes con để tải dữ liệu song song giúp tăng tốc độ tải

Ghim bộ nhớ (pinned memory = True) trên RAM giúp chuyển dữ liệu sang GPU nhanh hơn.

1.2. Valid_load

Không cần xáo trộn vì tập validation chỉ dùng để đánh giá, không dùng để học. Giữ nguyên thứ tự giúp metrics (accuracy, loss) giữa các epoch dễ tái lập và so sánh.

2. cấu hình huấn luyện

```
criterion = torch.nn.CrossEntropyLoss(label_smoothing=0.1)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
scaler = torch.cuda.amp.GradScaler()
torch.backends.cudnn.benchmark = True
torch.set_float32_matmul_precision("high")
```

Python

Hình 2. cấu hình huấn luyện

2.1. criterion = torch.nn.CrossEntropyLoss(label_smoothing=0.1)

Định nghĩa hàm mất mát là CrossEntropyLoss đây là hàm phổ biến nhất cho bài toán phân loại nhiều lớp với label_smoothing=0.1.

Thay vì nhãn đúng là 1.0 và các nhãn sai là 0.0 (hard label), nó làm "mềm" nhãn một chút: nhãn đúng thành 0.9, và phân bổ 0.1/(num_classes-1) cho các nhãn sai.

Lợi ích: Giảm overconfidence của mô hình, giúp mô hình tổng quát hóa tốt hơn (generalize better), đặc biệt hữu ích khi dữ liệu có nhiễu hoặc lớp không cân bằng.

Giá trị 0.1 là lựa chọn phổ biến và hiệu quả trong nhiều paper gần đây (ví dụ ResNet, ViT, ConvNeXt...).

2.2. optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

Khởi tạo bộ tối ưu hóa Adam cho mô hình.

model.parameters(): Tất cả các tham số (weights & biases) của mô hình sẽ được cập nhật.

lr=1e-3 (tức 0.001): Learning rate ban đầu là 0.001 – giá trị khởi đầu chuẩn khi dùng Adam cho các mô hình CNN phân loại ảnh (như ResNet, EfficientNet...).

2.3. scaler = torch.cuda.amp.GradScaler()

Khởi tạo GradScaler để hỗ trợ Automatic Mixed Precision (AMP) – huấn luyện với độ chính xác hỗn hợp (half-precision, float16).

Tăng tốc độ huấn luyện đáng kể (thường nhanh gấp 2-3 lần).

Tiết kiệm bộ nhớ GPU (VRAM) từ đó có thể dùng batch_size lớn hơn hoặc mô hình to hơn.

Độ chính xác cuối cùng gần như không suy giảm (thậm chí đôi khi tốt hơn).

Scaler sẽ tự động xử lý việc scale gradient để tránh underflow khi dùng float16.

2.4 torch.backends.cudnn.benchmark = True

Bật chế độ benchmark của cuDNN (thư viện tối ưu CUDA cho deep learning).

cuDNN sẽ tự động thử nhiều thuật toán convolution khác nhau ở lần chạy đầu tiên và chọn cái nhanh nhất phù hợp với input size hiện tại.

Tăng tốc độ forward/backward đáng kể (thường 10-30%).

2.5 torch.set_float32_matmul_precision("high")

Cấu hình độ chính xác cho các phép nhân ma trận (matrix multiplication) ở kiểu float32.

Giá trị "high": Cho phép sử dụng Tensor Cores trên GPU NVIDIA (Ampere trở lên như RTX 30xx, 40xx, A100...) để thực hiện matmul float32 nhanh hơn mà vẫn giữ độ chính xác gần như đầy đủ.

Tăng tốc độ huấn luyện thêm (đặc biệt trên GPU mới), thường nhanh hơn 20-50% mà không làm giảm chất lượng mô hình.

Nếu đặt "highest" thì chính xác tuyệt đối nhưng chậm hơn, "medium" thì nhanh hơn nhưng có thể giảm nhẹ độ chính xác.

Chương 3: Định nghĩa mô hình

1. Tải MobileNetV3-Large pre-trained trên ImageNet.

```python
model = models.mobilenet_v3_large(weights="IMAGENET1K_V1")
model.classifier[3] = torch.nn.Linear(
    model.classifier[3].in_features, 38
)
model = model.to(device)
```

Hình 1 Tải MobileNetV3-Large pre-trained trên ImageNet.

Giữ nguyên toàn bộ thân mạng (feature extractor), rất mạnh nhờ pre-training.

Thay lớp đầu ra cuối cùng từ 1000 lớp thành 38 lớp để phù hợp với bài toán.

Chuyển mô hình lên GPU để huấn luyện/inference nhanh.

2. Earrly stopping

```python
#early stop
class EarlyStopping:
    def __init__(self, patience=4, min_delta=1e-4):
        self.patience = patience
        self.min_delta = min_delta
        self.best_loss = float("inf")
        self.counter = 0

    def step(self, val_loss):
        if val_loss < self.best_loss - self.min_delta:
            self.best_loss = val_loss
            self.counter = 0
            return False
        else:
            self.counter += 1
            return self.counter >= self.patience
```

Python

Hình 2 Earrly stopping

patience=4 Cho phép mô hình "không cải thiện" liên tiếp tối đa 4 epoch trước khi dừng. Giá trị phổ biến: 3–10, tùy bài toán (bạn đang dùng 4 → khá hợp lý).

min_delta=1e-4 (0.0001) Chỉ coi là cải thiện nếu validation loss giảm ít nhất 0.0001 so với best trước đó. Tránh dừng sớm do dao động nhỏ ngẫu nhiên.

best_loss = float("inf") Ban đầu chưa có loss nào, nên đặt giá trị rất lớn để epoch đầu luôn được coi là "best".

counter Đếm số epoch liên tiếp mà loss không cải thiện đủ.

Chương 4: Huấn luyện mô hình và đánh giá mô hình - MobileNet.

1. Định nghãi hàm validate

```python
@torch.no_grad()

def validate(model, val_loader, criterion, device):
    model.eval()
    total_loss = 0.0
    correct = 0
    total = 0

    for x, y in val_loader:
        x = x.to(device, non_blocking=True)
        y = y.to(device, non_blocking=True)

        out = model(x)
        loss = criterion(out, y)

        total_loss += loss.item()
        preds = out.argmax(dim=1)
        correct += (preds == y).sum().item()
        total += y.size(0)

    avg_loss = total_loss / len(val_loader)
    acc = correct / total

    return avg_loss, acc
```

Hình 1 hàm validate

@torch.no_grad(): Bắt buộc khi evaluate → không tính gradient → nhanh hơn, ít tốn VRAM.

model.eval(): Quan trọng – chuyển chế độ để Dropout tắt và BatchNorm dùng thống kê đã học thay vì tính mới.

Không dùng Mixed Precision (autocast/scaler): Vì chỉ forward, không cần backprop → đơn giản và an toàn hơn.

preds = out.argmax(dim=1): Lấy chỉ số lớp có logit cao nhất → dự đoán cuối cùng.

(preds == y).sum().item(): Đếm số mẫu đúng trong batch (vì y và preds là tensor).

Trả về hai giá trị: avg_loss (validation loss) và acc (accuracy) – dùng để theo dõi và quyết định early stopping/lưu model.

2. Chạy mô hình và Đánh giá.

2.1 Mô hình

```python
early_stop = EarlyStopping(patience=5)
history = {
    "train_loss": [],
    "val_loss": [],
    "val_acc": [],
    "lr": []
}
best_val_loss = float("inf")
for epoch in range(15):
    model.train()
    total_loss = 0.0
    for x, y in train_load:
        x = x.to(device, non_blocking=True)
        y = y.to(device, non_blocking=True)

        optimizer.zero_grad(set_to_none=True)

        with autocast():
            out = model(x)
            loss = criterion(out, y)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        total_loss += loss.item()
    train_loss = total_loss / len(train_load)
    val_loss, val_acc = validate(
        model, valid_load, criterion, device
    )
    lr = optimizer.param_groups[0]["lr"]
    history["train_loss"].append(train_loss)
```

Pythor

Hình 2.1 mô hình MobileNet_1

```python
    history["train_loss"].append(train_loss)
    history["val_loss"].append(val_loss)
    history["val_acc"].append(val_acc)
    history["lr"].append(lr)
    print(
        f"Epoch [{epoch+1}/15] | "
        f"Train Loss: {train_loss:.4f} | "
        f"Val Loss: {val_loss:.4f} | "
        f"Val Acc: {val_acc*100:.2f}% | "
        f"LR: {lr}"
    )
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save({
            "epoch": epoch,
            "model_state_dict": model.state_dict(),
            "optimizer_state_dict": optimizer.state_dict(),
            "val_loss": val_loss,
            "val_acc": val_acc,
        }, "model_v3.pth")

    if early_stop.step(val_loss):
        print("early access")
        break

    if epoch == 7:
        for param_group in optimizer.param_groups:
            param_group["lr"] = 1e-4
```

Python

Hình 2.1 mô hình MobileNet_2

Tổng số epoch: 15.

Trong mỗi epoch phần huấn luyện (training phase):

- vòng lặp qua train_loader, mixed precision, scaler.step, tính train_loss.
- Phần đánh giá (validation phase): gọi hàm validate() (model.eval() + no_grad) để tính val_loss và val_acc trên valid_loader.
- Lưu lịch sử (history dict): train_loss, val_loss, val_acc, lr.
- In log chi tiết sau mỗi epoch.
- Lưu model tốt nhất nếu val_loss cải thiện (lưu cả model_state_dict và optimizer_state_dict).
- Kiểm tra early stopping dựa trên val_loss.
- Giảm learning rate xuống 1e-4 bắt đầu từ epoch 8.

2.2 Kết quả huấn luyện:

```
Epoch [1/15] | Train Loss: 0.8070 | Val Loss: 0.7622 | Val Acc: 98.33% | LR: 0.001
Epoch [2/15] | Train Loss: 0.7212 | Val Loss: 0.7594 | Val Acc: 97.39% | LR: 0.001
Epoch [3/15] | Train Loss: 0.7100 | Val Loss: 0.7748 | Val Acc: 96.96% | LR: 0.001
Epoch [4/15] | Train Loss: 0.7063 | Val Loss: 0.7126 | Val Acc: 98.88% | LR: 0.001
Epoch [5/15] | Train Loss: 0.7040 | Val Loss: 0.7300 | Val Acc: 98.58% | LR: 0.001
Epoch [6/15] | Train Loss: 0.7002 | Val Loss: 0.7348 | Val Acc: 98.17% | LR: 0.001
Epoch [7/15] | Train Loss: 0.7003 | Val Loss: 0.7012 | Val Acc: 99.08% | LR: 0.001
Epoch [8/15] | Train Loss: 0.6972 | Val Loss: 0.7474 | Val Acc: 97.99% | LR: 0.001
Epoch [9/15] | Train Loss: 0.6791 | Val Loss: 0.6774 | Val Acc: 99.80% | LR: 0.0001
Epoch [10/15] | Train Loss: 0.6759 | Val Loss: 0.6768 | Val Acc: 99.84% | LR: 0.0001
Epoch [11/15] | Train Loss: 0.6752 | Val Loss: 0.6762 | Val Acc: 99.84% | LR: 0.0001
Epoch [12/15] | Train Loss: 0.6746 | Val Loss: 0.6754 | Val Acc: 99.88% | LR: 0.0001
Epoch [13/15] | Train Loss: 0.6743 | Val Loss: 0.6756 | Val Acc: 99.86% | LR: 0.0001
Epoch [14/15] | Train Loss: 0.6740 | Val Loss: 0.6759 | Val Acc: 99.87% | LR: 0.0001
Epoch [15/15] | Train Loss: 0.6737 | Val Loss: 0.6773 | Val Acc: 99.82% | LR: 0.0001
```

Hình 2.2 Kết quả mô hình MobileNet

Train Loss: Giảm đều từ ~0.8070 (epoch 1) → ~0.6737 (epoch 15) → mô hình đang học tốt trên tập train.

Val Loss: Giảm từ ~0.7622 → ~0.6773 → cải thiện liên tục, không tăng ngược lại.

Val Accuracy: Tăng từ ~98.33% → 99.82% (cao nhất ở epoch cuối) → gần như hoàn hảo.

Learning Rate: Giảm đúng kế hoạch từ 0.001 → 0.0001 từ epoch 9 → giúp fine-tuning giai đoạn cuối.

Không kích hoạt Early Stopping (patience=5) → vì val_loss vẫn cải thiện dần.

2.3 Đánh giá tổng thể:

Không có overfitting:

- Train loss luôn cao hơn val loss một chút (ví dụ epoch cuối: 0.6737 vs 0.6773) → hiện tượng bình thường và tốt khi dùng data augmentation mạnh ở train (làm dữ liệu train "khó" hơn).
- Val acc cao và ổn định, không giảm → mô hình học được đặc trưng chung, không học nhiễu.

Không underfitting: Loss giảm đều, accuracy cao → mô hình đủ mạnh (MobileNetV3 pre-trained + fine-tuning tốt).

Accuracy quá cao cần kiểm tra lại xem có data leakage không (ảnh trùng giữa train/val?) hoặc dataset quá dễ.

3. Thử nghiệm mô hình với tập Test

```python
from PIL import Image
from pathlib import Path

checkpoint = torch.load("model_v3.pth", map_location=device)
model.load_state_dict(checkpoint["model_state_dict"])
model.eval()

test_root = Path(test_path)

img_paths = [
    p for p in test_root.rglob("*")
    if p.is_file() and p.suffix.lower() in ".jpg"
]

if not img_paths:
    raise FileNotFoundError(f"No image files found under: {test_root.resolve()}")

for p in sorted(img_paths):
    img = Image.open(p).convert("RGB")
    x = val_transform(img).unsqueeze(0).to(device)

    with torch.no_grad():
        out = model(x)
        pred = int(out.argmax(dim=1).item())

    print(p.name, full_ds.classes[pred])
```

Python

Hình 3 Thử nghiệm với tập Test của mô hình MobileNet

```
AppleCedarRust3.JPG Apple___Cedar_apple_rust
AppleCedarRust4.JPG Apple___Cedar_apple_rust
AppleScab1.JPG Apple___Apple_scab
AppleScab2.JPG Apple___Apple_scab
AppleScab3.JPG Apple___Apple_scab
CornCommonRust1.JPG Corn_(maize)___Common_rust_
CornCommonRust2.JPG Corn_(maize)___Common_rust_
CornCommonRust3.JPG Corn_(maize)___Common_rust_
PotatoEarlyBlight1.JPG Potato___Early_blight
PotatoEarlyBlight2.JPG Potato___Early_blight
PotatoEarlyBlight3.JPG Potato___Early_blight
PotatoEarlyBlight4.JPG Potato___Early_blight
PotatoEarlyBlight5.JPG Potato___Early_blight
PotatoHealthy1.JPG Potato___healthy
PotatoHealthy2.JPG Potato___healthy
TomatoEarlyBlight1.JPG Tomato___Early_blight
TomatoEarlyBlight2.JPG Tomato___Early_blight
TomatoEarlyBlight3.JPG Tomato___Early_blight
TomatoEarlyBlight4.JPG Tomato___Early_blight
TomatoEarlyBlight5.JPG Tomato___Early_blight
TomatoEarlyBlight6.JPG Tomato___Early_blight
TomatoHealthy1.JPG Tomato___healthy
TomatoHealthy2.JPG Tomato___healthy
TomatoHealthy3.JPG Tomato___healthy
TomatoHealthy4.JPG Tomato___healthy
TomatoYellowCurlVirus1.JPG Tomato___Tomato_Yellow_Leaf_Curl_Virus
TomatoYellowCurlVirus2.JPG Tomato   Tomato Yellow Leaf Curl Virus
```

Hình 3 Kết quả thử nghiệm với tập Test của mô hình MobileNet

Kết quả: Mô hình đạt độ chính xác 100% trên các mẫu test được kiểm tra.

Chương 5 Huấn luyện mô hình và đánh giá mô hình - EfficientNet.

1. Chạy mô hình và Đánh giá.

1.1. Mô hình

```
early_stop = EarlyStopping(patience=5)
history = {
    "train_loss": [],
    "val_loss": [],
    "val_acc": [],
    "lr": []
}
best_val_loss = float("inf")
for epoch in range(15):
    model.train()
    total_loss = 0.0
    for x, y in train_load:
        x = x.to(device, non_blocking=True)
        y = y.to(device, non_blocking=True)
        optimizer.zero_grad(set_to_none=True)
        with autocast():
            out = model(x)
            loss = criterion(out, y)
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        total_loss += loss.item()
    train_loss = total_loss / len(train_load)
    val_loss, val_acc = validate(
        model, valid_load, criterion, device
```

Hình 1.1 Mô hình EfficientNet_1

```
    lr = optimizer.param_groups[0]["lr"]
    history["train_loss"].append(train_loss)
    history["val_loss"].append(val_loss)
    history["val_acc"].append(val_acc)
    history["lr"].append(lr)
    print(
        f"Epoch [{epoch+1}/15] | "
        f"Train Loss: {train_loss:.4f} | "
        f"Val Loss: {val_loss:.4f} | "
        f"Val Acc: {val_acc*100:.2f}% | "
        f"LR: {lr}"
    )
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save({
            "epoch": epoch,
            "model_state_dict": model.state_dict(),
            "optimizer_state_dict": optimizer.state_dict(),
            "val_loss": val_loss,
            "val_acc": val_acc,
        }, "model_v3.pth")
    if early_stop.step(val_loss):
        print("early access")
        break
```

Hình 1.1 Mô hình EfficientNet_2

Tổng số epoch: 15.

Trong mỗi epoch gồm hai giai đoạn chính:

Phần huấn luyện (training phase): Vòng lặp qua train_loader: chuyển dữ liệu lên GPU (non_blocking), xóa gradient (zero_grad(set_to_none=True)), sử dụng Mixed Precision (autocast), tính loss, thực hiện backward với GradScaler, cập nhật trọng số và scaler, tính trung bình train_loss.

Phần đánh giá (validation phase): Chuyển mô hình sang model.eval() + torch.no_grad(), gọi hàm validate() để tính val_loss và val_acc trên valid_loader.

Các bước sau mỗi epoch:

- Lưu các chỉ số (train_loss, val_loss, val_acc, lr) vào history.
- In log chi tiết (epoch, losses, accuracy, lr).
- Nếu val_loss tốt hơn kỷ lục → lưu checkpoint (model + optimizer state).
- Kiểm tra early stopping dựa trên val_loss (patience=5).
- Giảm learning rate xuống 1e-4 từ epoch 8 trở đi.

Quy trình này giúp mô hình huấn luyện hiệu quả, giám sát chặt chẽ và dừng đúng lúc khi đã hội tụ.

1.2. Kết quả huấn luyện

```
Epoch [1/15] | Train Loss: 0.6743 | Val Loss: 0.6737 | Val Acc: 99.91% | LR: 0.0001
Epoch [2/15] | Train Loss: 0.6741 | Val Loss: 0.6737 | Val Acc: 99.92% | LR: 0.0001
Epoch [3/15] | Train Loss: 0.6738 | Val Loss: 0.6736 | Val Acc: 99.94% | LR: 0.0001
Epoch [4/15] | Train Loss: 0.6738 | Val Loss: 0.6732 | Val Acc: 99.94% | LR: 0.0001
Epoch [5/15] | Train Loss: 0.6738 | Val Loss: 0.6733 | Val Acc: 99.93% | LR: 0.0001
Epoch [6/15] | Train Loss: 0.6734 | Val Loss: 0.6728 | Val Acc: 99.94% | LR: 0.0001
Epoch [7/15] | Train Loss: 0.6735 | Val Loss: 0.6747 | Val Acc: 99.89% | LR: 0.0001
Epoch [8/15] | Train Loss: 0.6734 | Val Loss: 0.6729 | Val Acc: 99.96% | LR: 0.0001
Epoch [9/15] | Train Loss: 0.6734 | Val Loss: 0.6737 | Val Acc: 99.92% | LR: 0.0001
Epoch [10/15] | Train Loss: 0.6731 | Val Loss: 0.6735 | Val Acc: 99.93% | LR: 0.0001
Epoch [11/15] | Train Loss: 0.6734 | Val Loss: 0.6738 | Val Acc: 99.92% | LR: 0.0001
early access
```

Hình 1.2 Kết quả huấn luyện Mô hình EfficientNet

Train Loss: Giảm từ ~0.674 (epoch 1) → ~0.6734 (epoch 11) → rất ổn định ở mức thấp.

Val Loss:

- Epoch 1–10: ~0.6737 (gần như không đổi)
- Epoch 11: 0.6738 (tăng nhẹ)

Val Accuracy:

- Dao động nhẹ quanh ~99.9% từ epoch 1 đến epoch 10
- Epoch 11: 99.92%

Early Stopping: Kích hoạt ở epoch 11 (sau epoch 10), dừng sớm vì val_loss không cải thiện thêm.

Learning Rate: Vẫn là 0.001 (chưa thấy giảm lr trong log này, có thể bạn chưa áp dụng hoặc đặt ở epoch muộn hơn).

2. Thử nghiệm mô hình với tập Test

```python
from PIL import Image
from pathlib import Path

checkpoint = torch.load("model_v3.pth", map_location=device)
model.load_state_dict(checkpoint["model_state_dict"])
model.eval()

test_root = Path(test_path)

img_paths = [
    p for p in test_root.rglob("*")
    if p.is_file() and p.suffix.lower() in ".jpg"
]

if not img_paths:
    raise FileNotFoundError(f"No image files found under: {test_root.resolve()}")

for p in sorted(img_paths):
    img = Image.open(p).convert("RGB")
    x = val_transform(img).unsqueeze(0).to(device)

    with torch.no_grad():
        out = model(x)
        pred = int(out.argmax(dim=1).item())

    print(p.name, full_ds.classes[pred])
```

Pythor

Hình 2 Thử nghiệm với tập test của Mô hình EfficientNet

```
AppleCedarRust1.JPG Apple___Cedar_apple_rust
AppleCedarRust2.JPG Apple___Cedar_apple_rust
AppleCedarRust3.JPG Apple___Cedar_apple_rust
AppleCedarRust4.JPG Apple___Cedar_apple_rust
AppleScab1.JPG Apple___Apple_scab
AppleScab2.JPG Apple___Apple_scab
AppleScab3.JPG Apple___Apple_scab
CornCommonRust1.JPG Corn_(maize)___Common_rust_
CornCommonRust2.JPG Corn_(maize)___Common_rust_
CornCommonRust3.JPG Corn_(maize)___Common_rust_
PotatoEarlyBlight1.JPG Potato___Early_blight
PotatoEarlyBlight2.JPG Potato___Early_blight
PotatoEarlyBlight3.JPG Potato___Early_blight
PotatoEarlyBlight4.JPG Potato___Early_blight
PotatoEarlyBlight5.JPG Potato___Early_blight
PotatoHealthy1.JPG Potato___healthy
PotatoHealthy2.JPG Potato___healthy
TomatoEarlyBlight1.JPG Tomato___Early_blight
TomatoEarlyBlight2.JPG Tomato___Early_blight
TomatoEarlyBlight3.JPG Tomato___Early_blight
TomatoEarlyBlight4.JPG Tomato___Early_blight
TomatoEarlyBlight5.JPG Tomato___Early_blight
TomatoEarlyBlight6.JPG Tomato___Early_blight
TomatoHealthy1.JPG Tomato___healthy
TomatoHealthy2.JPG Tomato___healthy
TomatoHealthy3.JPG Tomato___healthy
TomatoHealthy4.JPG Tomato___healthy
TomatoYellowCurlVirus1.JPG Tomato___Tomato_Yellow_Leaf_Curl_Virus
TomatoYellowCurlVirus2.JPG Tomato___Tomato_Yellow_Leaf_Curl_Virus
TomatoYellowCurlVirus3.JPG Tomato___Tomato_Yellow_Leaf_Curl_Virus
```

Hình 2 Kết quả thử nghiệm với tập test của Mô hình EfficientNet

Kết quả: Mô hình đạt độ chính xác 100% trên các mẫu test được kiểm tra.

Chương 6 Huấn luyện mô hình và đánh giá mô hình - RESNET.

1. Huấn luyện mô hình và đánh giá

1.1 Huấn luyện mô hình

```
scaler = torch.amp.GradScaler('cuda', enabled=(device.type == "cuda"))

best_val_loss = float("inf")

for epoch in range(epochs):
    model.train()
    running_loss = 0.0

    for x, y in train_loader:
        x = x.to(device, non_blocking=True)
        y = y.to(device, non_blocking=True)

        optimizer.zero_grad(set_to_none=True)

        # Cập nhật cách dùng autocast mới: torch.amp.autocast
        with torch.amp.autocast('cuda', enabled=(device.type == "cuda")):
            out = model(x)
            loss = criterion(out, y)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        running_loss += loss.item()
    # Tính toán loss trung bình của tập Train
    train_loss = running_loss / len(train_loader)

    # Đánh giá trên tập Validation (sử dụng hàm validate đã định nghĩa)
    val_loss, val_acc = validate(model, val_loader, criterion, device)
```

Hình 1.1 Huấn luyện mô hình ResNet_1

```
    # In kết quả sau mỗi Epoch
    print(
        f"Epoch [{epoch+1}/{epochs}] | "
        f"Train Loss: {train_loss:.4f} | "
        f"Val Loss: {val_loss:.4f} | "
        f"Val Acc: {val_acc*100:.2f}%"
    )

    # Lưu lại model có kết quả tốt nhất trên tập Val
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save(model.state_dict(), "resnet18_best.pth")
        print(f"--> Đã lưu model tốt nhất tại Epoch {epoch+1}")

    # Chiến lược giảm Learning Rate thủ công tại epoch số 8 (index 7)
    if epoch == 7:
        for g in optimizer.param_groups:
            g["lr"] = 1e-4
        print("--> Đã hạ Learning Rate xuống 1e-4")
```

Hình 1.1 Huấn luyện mô hình ResNet_2

Khởi tạo:

- Tạo GradScaler hỗ trợ Mixed Precision chỉ khi chạy trên GPU (enabled nếu device là "cuda").
- Khởi tạo best_val_loss = float("inf") để theo dõi model tốt nhất.

Trong mỗi epoch:

Phần huấn luyện (training phase):

- Chuyển mô hình sang model.train().
- Duyệt train_loader: chuyển dữ liệu lên device (non_blocking=True), xóa gradient (zero_grad(set_to_none=True)).
- Sử dụng torch.amp.autocast (cập nhật cách gọi mới) để forward, tính loss bằng criterion.
- Thực hiện backward với scaler.scale(loss), cập nhật trọng số (scaler.step(optimizer)) và cập nhật scaler.
- Cộng dồn loss để tính train_loss trung bình.

Phần đánh giá (validation phase):

- Gọi hàm validate() để tính val_loss và val_acc trên valid_loader.

Sau mỗi epoch:

- In log chi tiết: epoch, train_loss, val_loss, val_acc (%).
- Nếu val_loss tốt hơn kỷ lục → cập nhật best_val_loss và lưu model tốt nhất (resnet18_best.pth).
- Giảm learning rate xuống 1e-4 bắt đầu từ epoch 8 (sau epoch 7).

Quy trình này sử dụng Mixed Precision hiện đại, tối ưu bộ nhớ và tốc độ, đồng thời giám sát chặt chẽ để lưu model hiệu suất cao nhất.

1.2. Đánh giá mô hình

```
Epoch [1/15] | Train Loss: 0.6740 | Val Loss: 0.6766 | Val Acc: 99.84%
--> Đã lưu model tốt nhất tại Epoch 1
Epoch [2/15] | Train Loss: 0.6736 | Val Loss: 0.6764 | Val Acc: 99.86%
--> Đã lưu model tốt nhất tại Epoch 2
Epoch [3/15] | Train Loss: 0.6736 | Val Loss: 0.6771 | Val Acc: 99.84%
Epoch [4/15] | Train Loss: 0.6734 | Val Loss: 0.6765 | Val Acc: 99.86%
Epoch [5/15] | Train Loss: 0.6731 | Val Loss: 0.6779 | Val Acc: 99.77%
Epoch [6/15] | Train Loss: 0.6728 | Val Loss: 0.6764 | Val Acc: 99.86%
Epoch [7/15] | Train Loss: 0.6728 | Val Loss: 0.6757 | Val Acc: 99.87%
--> Đã lưu model tốt nhất tại Epoch 7
Epoch [8/15] | Train Loss: 0.6727 | Val Loss: 0.6761 | Val Acc: 99.87%
--> Đã hạ Learning Rate xuống 1e-4
Epoch [9/15] | Train Loss: 0.6726 | Val Loss: 0.6754 | Val Acc: 99.90%
--> Đã lưu model tốt nhất tại Epoch 9
Epoch [10/15] | Train Loss: 0.6723 | Val Loss: 0.6761 | Val Acc: 99.83%
Epoch [11/15] | Train Loss: 0.6725 | Val Loss: 0.6762 | Val Acc: 99.84%
Epoch [12/15] | Train Loss: 0.6724 | Val Loss: 0.6756 | Val Acc: 99.89%
Epoch [13/15] | Train Loss: 0.6723 | Val Loss: 0.6753 | Val Acc: 99.88%
--> Đã lưu model tốt nhất tại Epoch 13
Epoch [14/15] | Train Loss: 0.6721 | Val Loss: 0.6754 | Val Acc: 99.87%
Epoch [15/15] | Train Loss: 0.6725 | Val Loss: 0.6756 | Val Acc: 99.86%
```

Hình 1.2 Kết quả huấn luyện mô hình ResNet

Train Loss: Giảm đều từ 0.6740 (epoch 1) → 0.6725 (epoch 15) → hội tụ ổn định.

Val Loss: Giảm từ 0.6766 (epoch 1) → thấp nhất 0.6753 (epoch 13) → dao động nhẹ ở các epoch cuối.

Val Accuracy:

- Tăng nhanh từ 99.84% (epoch 1) → đạt cao nhất 99.90% (epoch 9).
- Duy trì ổn định quanh 99.8–99.9% đến cuối.

Early Stopping: Không kích hoạt → mô hình vẫn cải thiện nhẹ đến epoch cuối.

Learning Rate: Giảm xuống 1e-4 từ epoch 8 → hỗ trợ fine-tuning hiệu quả.

Đánh giá tổng thể: Mô hình hội tụ nhanh ngay từ epoch đầu, đạt validation accuracy tối đa 99.90% và duy trì ổn định cao. Checkpoint tốt nhất được cập nhật nhiều lần (epoch 1, 2, 7, 9, 13), chứng tỏ hiệu suất tăng dần đều mà không overfit.

2. Thử nghiệm với tập test

```python
from PIL import Image
from pathlib import Path

model.load_state_dict(torch.load("./resnet18_best.pth", map_location=device))
model.eval()

# test_path = "./test"
test_root = Path(test_path)

img_paths = [p for p in test_root.rglob("*") if p.suffix.lower() in [".jpg", ".png"]]

for p in img_paths:
    img = Image.open(p).convert("RGB")
    x = val_transform(img).unsqueeze(0).to(device)

    with torch.no_grad():
        out = model(x)
        pred = out.argmax(dim=1).item()

    print(p.name, ds_for_train.classes[pred])
```

Hình 2 Thử nghiệm với tập test của Mô hình ResNet

```
PotatoHealthy2.JPG Potato___healthy
PotatoEarlyBlight5.JPG Potato___Early_blight
TomatoHealthy3.JPG Tomato___healthy
AppleScab1.JPG Apple___Apple_scab
PotatoEarlyBlight1.JPG Potato___Early_blight
PotatoEarlyBlight2.JPG Potato___Early_blight
TomatoEarlyBlight3.JPG Tomato___Early_blight
TomatoHealthy2.JPG Tomato___healthy
TomatoEarlyBlight6.JPG Tomato___Early_blight
TomatoEarlyBlight5.JPG Tomato___Early_blight
TomatoHealthy4.JPG Tomato___healthy
TomatoYellowCurlVirus4.JPG Tomato___Tomato_Yellow_Leaf_Curl_Virus
AppleScab2.JPG Apple___Apple_scab
AppleScab3.JPG Apple___Apple_scab
CornCommonRust2.JPG Corn_(maize)___Common_rust_
CornCommonRust3.JPG Corn_(maize)___Common_rust_
PotatoEarlyBlight4.JPG Potato___Early_blight
TomatoEarlyBlight4.JPG Tomato___Early_blight
PotatoEarlyBlight3.JPG Potato___Early_blight
AppleCedarRust4.JPG Apple___Cedar_apple_rust
TomatoYellowCurlVirus1.JPG Tomato___Tomato_Yellow_Leaf_Curl_Virus
TomatoEarlyBlight1.JPG Tomato___Early_blight
TomatoHealthy1.JPG Tomato___healthy
TomatoYellowCurlVirus2.JPG Tomato___Tomato_Yellow_Leaf_Curl_Virus
```

Hình 2 Kết quả thử nghiệm với tập test của Mô hình ResNet

Mô hình dự đoán chính xác 100% trên tất cả các mẫu test, bao gồm các trường hợp khó như bệnh ở giai đoạn sớm và trạng thái khỏe mạnh:

Chương 7 Đánh giá sơ bộ kết quả 3 mô hình

Cả ba mô hình đều đạt được hiệu suất cực kỳ ấn tượng trên bộ dữ liệu New Plant Diseases Dataset. Kết quả chi tiết như sau:

1. Mô hình MobileNetV3-Large

Hiệu suất: Đạt độ chính xác trên tập validation lên đến 99.82%.

Đặc điểm huấn luyện: Train loss giảm đều và thấp hơn val loss một chút, cho thấy mô hình không bị overfit nhờ áp dụng data augmentation mạnh.

Ưu điểm: Tốc độ huấn luyện và suy luận nhanh, phù hợp cho các thiết bị di động hoặc môi trường có tài nguyên hạn chế.

2. Mô hình EfficientNet

Hiệu suất: Đạt độ chính xác cực cao, dao động quanh mức 99.92%.

Đặc điểm huấn luyện: Hội tụ rất nhanh ngay từ những epoch đầu tiên. Kích hoạt Early Stopping ở epoch 11 do val_loss không còn cải thiện thêm.

Ưu điểm: Khả năng trích xuất đặc trưng rất mạnh mẽ và ổn định.

3. Mô hình ResNet-18

Hiệu suất: Đạt độ chính xác cao nhất ở mức 99.90%.

Đặc điểm huấn luyện: Quá trình huấn luyện ổn định, checkpoint tốt nhất được cập nhật liên tục qua các giai đoạn.

Ưu điểm: Cấu trúc mạng khối (residual blocks) giúp giải quyết tốt vấn đề triệt tiêu gradient, mang lại kết quả dự đoán chính xác 100% trên các mẫu thử nghiệm (test set).

Nhận xét chung: Cả 3 mô hình đều dự đoán chính xác 100% trên các mẫu test được kiểm tra. Tuy nhiên, cần lưu ý kiểm tra lại hiện tượng Data Leakage (rò rỉ dữ liệu) hoặc độ đa dạng của tập Test vì tỉ lệ chính xác tuyệt đối thường rất hiếm gặp trong thực tế.

Định hướng phát triển cho lần báo cáo cuối cùng

1. Tăng cường mô tả:

- RandomResizedCrop (mô phỏng zoom/cắt lá)
- Gaussian blur / motion blur nhẹ (mô phỏng rung tay)
- RandomPerspective / affine (mô phỏng góc chụp)
- RandomErasing/Cutout (mô phỏng lá bị che, bóng, lỗ sâu)
- Aug theo "ánh sáng thực": thay đổi gamma, white balance

2. Test với tập dữ liệu được thu thập thực tế bằng điện thoại cũ, camera không chất lượng (nông dân thường chỉ dùng smartphone không có camera quá tốt)

3. Thiết kế, tối ưu mô hình cho inference hướng tới phát triển cho ứng dụng di động