

BỘ NÔNG NGHIỆP VÀ MÔI TRƯỜNG
PHÂN HIỆU TRƯỜNG ĐẠI HỌC THỦY LỢI
BỘ MÔN CÔNG NGHỆ THÔNG TIN



ĐỒ ÁN KHAI PHÁ DỮ LIỆU

TÊN ĐỀ TÀI :

ỨNG DỤNG HỌC SÂU DỰ ĐOÁN ẢNH SỨC KHỎE RAU XANH

Giảng viên hướng dẫn: Th.S Vũ Thị Hạnh

Lớp: S25-64CNTT

Sinh viên thực hiện	Mã sinh viên
Nguyễn Thành Tâm	2251068248
Dương Hữu Tín	2251068265
Ngô Văn Sửu	2251068244

TP. Hồ Chí Minh, ngày 28 tháng 10 năm 2025

LỜI MỞ ĐẦU

Trong bối cảnh nền nông nghiệp đang tiến tới tự động hóa và thông minh hóa (Smart Agriculture), việc quản lý dịch hại và sức khỏe cây trồng đóng vai trò quyết định đến năng suất và chất lượng sản phẩm. Đối với các loại rau củ, việc phát hiện sớm các dấu hiệu bệnh, hư hỏng hoặc thiếu hụt dinh dưỡng là cực kỳ cần thiết để đưa ra các biện pháp can thiệp kịp thời, từ đó tối ưu hóa chi phí sản xuất và giảm thiểu tổn thất kinh tế.

Đồ án "Ứng dụng Học sâu Dự đoán Ảnh Sức khỏe Rau Xanh" được thực hiện nhằm ứng dụng các tiến bộ của lĩnh vực Học Sâu (Deep Learning) để xây dựng một hệ thống phân loại rau tự động. Chúng em tập trung vào việc nghiên cứu và so sánh hiệu suất của hai kiến trúc mạng nơ-ron tích chập (CNN) nổi tiếng: ResNet50 (Residual Network 50-layer) - đại diện cho các mô hình mạnh mẽ, và MobileNetV2 - đại diện cho các mô hình nhẹ, tối ưu cho thiết bị di động. Đặc biệt, kỹ thuật Học Chuyển giao (Transfer Learning) được sử dụng để tận dụng các trọng số đã huấn luyện sẵn, giúp cả hai mô hình đạt được hiệu suất cao mặc dù sử dụng dữ liệu hình ảnh thu thập từ internet có giới hạn.

Toàn bộ quá trình huấn luyện và đánh giá mô hình được thực hiện trên nền tảng Google Colab để tận dụng tài nguyên GPU miễn phí. Cuối cùng, mô hình dự đoán được triển khai thành một ứng dụng web đơn giản, cho phép người dùng dễ dàng tải lên hình ảnh rau và nhận kết quả phân loại sức khỏe ngay lập tức.

Báo cáo này sẽ trình bày chi tiết các bước từ thu thập dữ liệu, xử lý tiền, xây dựng và huấn luyện mô hình, đến đánh giá hiệu năng và triển khai sản phẩm ứng dụng. Với sự hướng dẫn tận tình của cô Vũ Thị Hạnh, nhóm chúng em đã hoàn thành bài báo cáo này. Chúng em đã cố gắng vận dụng những kiến thức đã học để thực hiện và chắc chắn không tránh khỏi những sai sót. Chúng em rất mong nhận được sự thông cảm và góp ý của cô. Nhóm em xin chân thành cảm ơn cô.

MỤC LỤC

DANH MỤC HÌNH ẢNH	3
CHƯƠNG 1. GIỚI THIỆU TỔNG QUAN	4
1.1. Đặt vấn đề và lý do chọn đề tài	4
1.2. Mục tiêu nghiên cứu	4
1.3. Phạm vi nghiên cứu	5
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT	5
2.1. Tổng quan về khai phá dữ liệu (Data Mining)	5
2.2. Học máy và học sâu (Deep Learning)	7
2.3. Các kiến trúc CNN chính và học chuyển giao (Transfer Learning)	8
2.4. Các công cụ triển khai chuyên biệt	10
CHƯƠNG 3. DỮ LIỆU VÀ TIỀN XỬ LÝ	11
3.1. Nguồn dữ liệu	11
3.2. Tiền xử lý dữ liệu	12
CHƯƠNG 4. XÂY DỰNG VÀ HUẤN LUYỆN MÔ HÌNH	25
4.1. ResNet50	25
4.2. MobileNetV2	32
CHƯƠNG 5. KẾT QUẢ VÀ ĐÁNH GIÁ	40
5.1. ResNet50	40
5.2. MobileNetV2	43
5.3. Đánh giá ResNet50 và MobileNetV2	46
CHƯƠNG 6. TRIỂN KHAI ỨNG DỤNG WEB	48
CHƯƠNG 7. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	50
TÀI LIỆU THAM KHẢO	52

DANH MỤC HÌNH ẢNH

Hình 2.1: CRISP - DM	6
Hình 3.1: Kiểm tra số lượng ảnh	14
Hình 3.2: Cấu hình tăng cường data.....	16
Hình 3.3: Tính toán trọng số lớp	17
Hình 3.4: Trích xuất đặc trưng màu và kết cấu.....	19
Hình 3.5: Biểu đồ phân cụm trực quan bằng PCA.....	21
Hình 3.6: Cấu hình tham số mô hình MobileNetV2.....	22
Hình 3.7: Cấu hình tăng cường data.....	23
Hình 4.1: Xây dựng mô hình ResNet50.....	25
Hình 4.2: Biên dịch mô hình và cấu hình callbacks.....	27
Hình 4.3: Huấn luyện giai đoạn 1	29
Hình 4.4: Huấn luyện giai đoạn 2	31
Hình 4.5: Xây dựng mô hình MobileNetV2.....	33
Hình 4.6: Biên dịch mô hình và cấu hình callbacks.....	35
Hình 4.7: Huấn luyện giai đoạn 1	37
Hình 4.8: Huấn luyện giai đoạn 2	38
Hình 5.1: Đồ thị Acc, Loss của mô hình ResNet50.....	40
Hình 5.2: Báo cáo phân loại trên tập test.....	41
Hình 5.3: Ma trận nhầm lẫn.....	42
Hình 5.4: Đồ thị Acc, Loss của mô hình MobileNetV2.....	43
Hình 5.5: Đánh giá Tổng hợp (MobileNetV2 - Tập Validation.....	44
Hình 5.6: Báo cáo phân loại chi tiết	45
Hình 5.7: Ma trận nhầm lẫn.....	46
Hình 5.8: Biểu đồ so sánh hiệu suất 2 mô hình.....	46

CHƯƠNG 1. GIỚI THIỆU TỔNG QUAN

1.1. Đặt vấn đề và lý do chọn đề tài

Nông nghiệp là ngành kinh tế cốt lõi, nhưng năng suất và chất lượng cây trồng luôn bị đe dọa bởi các yếu tố gây bệnh, thiếu hụt dinh dưỡng hoặc hư hỏng. Việc phát hiện sớm và chính xác tình trạng sức khỏe của rau củ quả là vô cùng quan trọng, giúp nông dân đưa ra các biện pháp can thiệp kịp thời, từ đó giảm thiểu thiệt hại mùa màng và tối ưu hóa chi phí sản xuất.

Trong phương pháp truyền thống, việc kiểm tra sức khỏe cây trồng thường dựa vào quan sát bằng mắt thường của người làm vườn hoặc chuyên gia, phương pháp này tốn kém thời gian, chi phí, và dễ bị sai sót do yếu tố chủ quan. Sự phát triển vượt bậc của Học Sâu (Deep Learning), đặc biệt là Mạng Nơ-ron Tích chập (CNN), đã mở ra giải pháp tự động, nhanh chóng và chính xác hơn cho bài toán phân loại hình ảnh.

Đề án này tập trung vào việc nghiên cứu và so sánh hiệu suất giữa hai kiến trúc CNN tiên tiến:

- ResNet50 (Residual Network 50-layer): Đại diện cho các mô hình mạnh mẽ, độ chính xác cao.
- MobileNetV2: Đại diện cho các mô hình nhẹ, tối ưu về tốc độ và kích thước, lý tưởng cho việc triển khai trên thiết bị di động (Mobile Deployment).

Việc sử dụng kỹ thuật Học Chuyển giao (Transfer Learning) sẽ giúp cả hai mô hình đạt được độ chính xác cao ngay cả với nguồn dữ liệu hình ảnh rau xà lách thu thập từ internet có giới hạn. Mục tiêu là xác định mô hình nào (ResNet50 hay MobileNetV2) mang lại sự cân bằng tốt nhất giữa độ chính xác và tốc độ tính toán cho ứng dụng thực tế.

1.2. Mục tiêu nghiên cứu

Mục tiêu chính của đề án này là:

- Xây dựng mô hình phân loại: Phát triển và tinh chỉnh (Fine-tune) mô hình ResNet50 và MobileNetV2 trên nền tảng Google Colab để phân loại tình trạng sức khỏe của rau (ví dụ: Khỏe mạnh, Bị bệnh A, Hư hỏng) từ hình ảnh thu thập được.
- Đánh giá hiệu năng: Đánh giá chính xác hiệu suất của mô hình thông qua các chỉ số tiêu chuẩn (Accuracy, Precision, Recall, F1-Score) trên tập dữ liệu kiểm thử độc lập.
- Triển khai ứng dụng: Phát triển một giao diện web trực quan, thân thiện với người dùng, cho phép người dùng tải lên hình ảnh và nhận kết quả dự đoán sức khỏe rau ngay lập tức.

1.3. Phạm vi nghiên cứu

- Dữ liệu: Tập trung vào dữ liệu hình ảnh rau, được thu thập từ internet và được gán nhãn thủ công hoặc dựa trên nguồn dữ liệu có sẵn.
- Mô hình: Sử dụng kiến trúc ResNet50 và MobileNetV2 làm mô hình chính.
- Công cụ: Giới hạn trong việc sử dụng Google Colab cho việc huấn luyện và ReactJS cho việc triển khai giao diện web.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

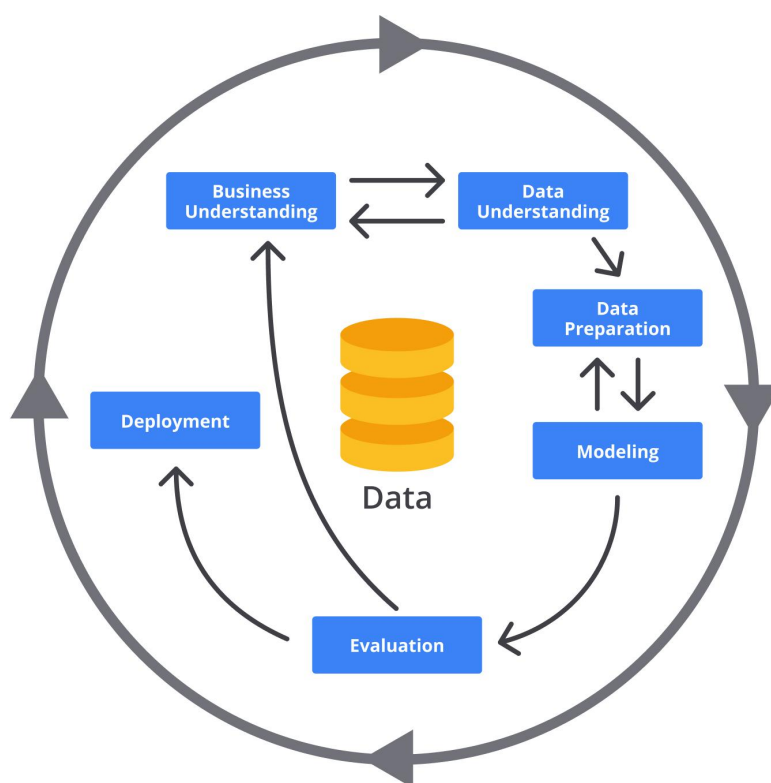
2.1. Tổng quan về khai phá dữ liệu (Data Mining)

2.1.1. Khái niệm khai phá dữ liệu

Khai phá Dữ liệu (Data Mining) là quá trình khám phá các mẫu (patterns) có ý nghĩa, hữu ích, và tiềm năng từ khối lượng lớn dữ liệu. Đây là một bước quan trọng trong quy trình khám phá tri thức từ dữ liệu (Knowledge Discovery in Databases - KDD). Mục tiêu cuối cùng là biến dữ liệu thô thành thông tin có giá trị, hỗ trợ cho việc ra quyết định.

Các tác vụ chính của Data Mining bao gồm: Phân loại (Classification), Hồi quy (Regression), Phân cụm (Clustering), và Luật kết hợp (Association Rules). Trong đồ án này, chúng ta tập trung vào tác vụ Phân loại, nơi mô hình được huấn luyện để gán nhãn cho các mẫu dữ liệu mới (hình ảnh).

2.1.2. Các giai đoạn chính (CRISP -DM)



Hình 2.1: CRISP - DM

Mô hình quy trình chuẩn ngành chéo cho khai phá dữ liệu (CRISP-DM - Cross-Industry Standard Process for Data Mining) là khuôn khổ phổ biến nhất để hướng dẫn các dự án Data Mining. Quy trình này bao gồm sáu giai đoạn lặp đi lặp lại:

1. Hiểu biết về Nghiệp vụ (Business Understanding): Xác định mục tiêu của dự án (nhận diện sức khỏe rau), yêu cầu và tiêu chí thành công.

2. Hiểu biết về Dữ liệu (Data Understanding): Thu thập, khám phá dữ liệu ban đầu (hình ảnh rau), nhận diện vấn đề chất lượng dữ liệu.
3. Chuẩn bị Dữ liệu (Data Preparation): Làm sạch, biến đổi, xây dựng dữ liệu (tiền xử lý ảnh, tăng cường dữ liệu).
4. Mô hình hóa (Modeling): Lựa chọn, xây dựng và huấn luyện mô hình (sử dụng ResNet50 và Transfer Learning).
5. Đánh giá (Evaluation): Đánh giá hiệu suất của mô hình đã huấn luyện (sử dụng các chỉ số như Accuracy, F1-Score).
6. Triển khai (Deployment): Đưa mô hình vào sử dụng thực tế (thông qua ứng dụng web Streamlit).

Trong đồ án này, việc nhận diện sức khỏe rau bằng hình ảnh chính là giai đoạn Mô hình hóa và Triển khai trong khuôn khổ CRISP-DM.

2.2. Học máy và học sâu (Deep Learning)

2.2.1. Phân loại trong học máy

Phân loại (Classification) là một kỹ thuật học có giám sát (Supervised Learning), trong đó mô hình được huấn luyện bằng các cặp dữ liệu đầu vào và nhãn tương ứng. Mục tiêu là học một hàm ánh xạ từ dữ liệu đầu vào (hình ảnh) sang một trong các lớp đầu ra (khỏe mạnh, bệnh A, bệnh B). Đây là tác vụ trọng tâm của đồ án này.

2.2.2. Giới thiệu Mạng Nơ-ron Tích chập (CNN)

Mạng Nơ-ron Tích chập (Convolutional Neural Network - CNN) là một lớp mạng nơ-ron sâu chuyên biệt, được thiết kế đặc biệt để xử lý dữ liệu có cấu trúc lưới, như hình ảnh. Sự ưu việt của CNN so với các mạng nơ-ron truyền thống trong xử lý ảnh đến từ ba loại lớp chính:

- Lớp Tích chập (Convolutional Layer): Sử dụng các bộ lọc (filters/kernels) để quét qua hình ảnh, trích xuất các đặc trưng cục bộ như cạnh, góc, hoặc kết cấu. Tính năng chia sẻ trọng số (Weight Sharing) giúp giảm đáng kể số lượng tham số cần huấn luyện.
- Lớp Kéo xuống (Pooling Layer): Giảm kích thước không gian của biểu diễn đặc trưng, làm cho mô hình ít nhạy cảm hơn với sự thay đổi nhỏ của đối tượng trong ảnh, đồng thời giảm chi phí tính toán.
- Lớp Kết nối đầy đủ (Fully Connected Layer): Sau khi các lớp tích chập trích xuất đặc trưng, lớp này sẽ sử dụng các đặc trưng đó để thực hiện việc phân loại cuối cùng.

CNN đặc biệt phù hợp với đề tài "Ứng dụng học sâu dự đoán ảnh sức khỏe rau xanh" vì nó có khả năng tự động học và trích xuất các đặc trưng trực quan quan trọng (ví dụ: màu sắc của lá, hình dạng đốm bệnh) mà không cần can thiệp thủ công.

2.3. Các kiến trúc CNN chính và học chuyển giao (Transfer Learning)

2.3.1. Kiến trúc ResNet và khối tàn dư (Residual Block)

ResNet (Residual Network) là một kiến trúc CNN đột phá được giới thiệu vào năm 2015 bởi Kaiming He và cộng sự. Trước ResNet, việc làm cho mạng sâu hơn thường dẫn đến vấn đề Gradient Vanishing (độ dốc bị triệt tiêu) và hiện tượng giảm hiệu suất (mô hình sâu hơn lại tệ hơn mô hình nông hơn).

ResNet giải quyết vấn đề này bằng cách đưa vào Khối Tàn dư (Residual Block). Thay vì học trực tiếp hàm mục tiêu $H(x)$, khối tàn dư học một hàm tàn dư $F(x) = H(x) - x$. Đầu ra của khối tàn dư là $F(x) + x$, trong đó x là ánh xạ đồng nhất (Identity Mapping) được cộng thẳng vào đầu ra của lớp tích chập.

Vai trò của Khối Tàn dư: Nó cho phép gradient dễ dàng lan truyền ngược qua các lớp bằng cách sử dụng đường tắt (shortcut connection), giúp huấn luyện

các mạng cực kỳ sâu (như ResNet50, ResNet101) mà không làm suy giảm hiệu suất.

ResNet50 là phiên bản ResNet có 50 lớp sâu, được sử dụng rộng rãi vì nó cung cấp sự cân bằng tuyệt vời giữa độ chính xác cao và tốc độ tính toán hợp lý.

2.3.2. Kiến trúc MobileNetV2 (sự tối ưu cho di động)

MobileNetV2 là một kiến trúc CNN được thiết kế đặc biệt cho các thiết bị di động và các ứng dụng yêu cầu tính toán hiệu quả, được giới thiệu bởi Sandler et al. vào năm 2018. Mục tiêu chính của MobileNetV2 là duy trì độ chính xác cao trong khi giảm đáng kể số lượng tham số và chi phí tính toán.

Kiến trúc này dựa trên hai khái niệm cốt lõi:

Tích chập Sâu và Tách biệt (Depthwise Separable Convolutions): Thay vì thực hiện tích chập $3 \times 3 \times M \times N$ thông thường, MobileNetV2 tách nó thành hai bước:

Depthwise Convolution: Áp dụng một bộ lọc 3×3 duy nhất cho mỗi kênh đầu vào (M).

Pointwise Convolution: Áp dụng tích chập 1×1 để kết hợp các kênh đầu ra.

Kỹ thuật này giúp giảm số lượng phép tính lên đến 8 hoặc 9 lần so với tích chập tiêu chuẩn.

Khối Tàn dư Đảo ngược Tuyến tính (Inverted Residual and Linear Bottlenecks): Không giống như ResNet truyền thống (mở rộng ở giữa), MobileNetV2 sử dụng khối tàn dư đảo ngược: mở rộng kênh ở đầu và thu hẹp ở cuối bằng lớp tuyến tính.

Mục đích: Đảm bảo rằng các thông tin quan trọng được bảo toàn trong biểu diễn có kích thước thấp, tối ưu hóa bộ nhớ và hiệu suất.

2.3.3. Học chuyển giao (Transfer Learning)

Học Chuyển giao (Transfer Learning) là một kỹ thuật trong Deep Learning, nơi mô hình được phát triển cho một tác vụ (Task A) được sử dụng lại làm điểm khởi đầu cho một tác vụ khác (Task B).

Sử dụng Trọng số ImageNet: Trong đề án này, chúng ta sử dụng các trọng số của cả hai kiến trúc ResNet50 và MobileNetV2 đã được huấn luyện trên tập dữ liệu khổng lồ ImageNet (hơn 14 triệu hình ảnh thuộc 1000 lớp). Các lớp ban đầu của các mô hình này đã học được các đặc trưng cơ bản và phổ quát về hình ảnh (như cạnh, vân, kết cấu).

Lợi ích:

- Rút ngắn thời gian huấn luyện: Mô hình đã được khởi tạo tốt, chỉ cần tinh chỉnh nhẹ.
- Cải thiện hiệu suất với dữ liệu hạn chế: Đặc biệt quan trọng đối với dữ liệu thu thập từ internet có thể không đủ lớn hoặc đa dạng như ImageNet.
- Giảm rủi ro quá khớp (Overfitting): Trọng số được lấy từ một tập dữ liệu rất lớn, giúp ổn định quá trình huấn luyện.

Quá trình này thường bao gồm việc đóng băng (freezing) các lớp đầu và chỉ huấn luyện các lớp cuối (Fully Connected Layers) được thêm vào tùy chỉnh cho bài toán phân loại sức khỏe rau của chúng ta.

2.4. Các công cụ triển khai chuyên biệt

2.4.1. Google Colaboratory (Colab)

Google Colab là một dịch vụ điện toán đám mây dựa trên Jupyter Notebook, cho phép người dùng viết và chạy Python.

Lợi ích chính: Cung cấp quyền truy cập miễn phí vào các tài nguyên phần cứng mạnh mẽ, đặc biệt là GPU (Graphics Processing Unit). Đối với các mô hình Deep Learning sâu như ResNet50, việc sử dụng GPU là bắt buộc để giảm thời gian huấn luyện từ vài ngày xuống còn vài giờ hoặc vài chục phút.

Tính năng quan trọng: Cho phép tích hợp liền mạch với Google Drive để lưu trữ dữ liệu và mô hình đã huấn luyện.

2.4.2. Flask (Backend API)

Vai trò: Flask (một micro-framework của Python) được sử dụng để xây dựng API xử lý (Processing API). Flask chịu trách nhiệm tải mô hình (.h5) vào bộ nhớ, nhận ảnh từ Frontend, thực hiện các bước tiền xử lý, gọi hàm `model.predict()`, và xử lý các tính toán phức tạp như Grad-CAM, Contours

2.4.3. ReactJS (Giao diện Người dùng - Frontend)

Vai trò: ReactJS (một thư viện JavaScript) được sử dụng để xây dựng Giao diện Người dùng Động (UI). Nó quản lý việc tải tệp ảnh từ người dùng, gửi yêu cầu dự đoán qua API Flask, và hiển thị trực quan các kết quả trả về, bao gồm kết quả phân loại và các hình ảnh giải thích Grad-CAM.

Sự kết hợp giữa ReactJS (giao diện) và Flask (xử lý mô hình) tạo nên một kiến trúc triển khai chuyên nghiệp, tách biệt rõ ràng trách nhiệm của Frontend và Backend, giúp đồ án có tính mở rộng và bảo trì cao.

CHƯƠNG 3. DỮ LIỆU VÀ TIỀN XỬ LÝ

3.1. Nguồn dữ liệu

Đồ án sử dụng bộ dữ liệu "Lettuce Health Compiled Dataset" được cung cấp trên nền tảng Kaggle. Bộ dữ liệu này đã được tổ chức sẵn cho bài toán phân loại, tập trung vào việc nhận diện sức khỏe của Rau xà lách trong các trạng thái khác nhau.

Link dataset: <https://www.kaggle.com/datasets/ashimstha/lettuce-health-compiled-dataset>

Bộ dữ liệu này được tổ chức theo cấu trúc phân cấp đơn giản, gồm hai thư mục chính tương ứng với hai lớp (binary classification):

- Khỏe mạnh (Healthy): 326 ảnh, hình ảnh rau xà lách được coi là khỏe mạnh, có đủ dinh dưỡng và không có dấu hiệu bệnh tật rõ rệt.

- Không khỏe mạnh (Unhealthy/Stressed): 381 ảnh, hình ảnh tổng hợp của rau xà lách bị căng thẳng, bao gồm nhiều loại bệnh tật (nấm, vi khuẩn, virus) và thiếu hụt dinh dưỡng.

Tổng số lượng hình ảnh là khoảng 700 mẫu. Đây là một quy mô dữ liệu tương đối nhỏ đối với Deep Learning, do đó việc sử dụng Transfer Learning với ResNet50, MobileNetV2 và kỹ thuật Data Augmentation (như đã lên kế hoạch) là vô cùng cần thiết để tránh quá khớp (Overfitting) và đảm bảo tính tổng quát hóa của mô hình.

Ngoài ra để tránh trường hợp quá khớp overfitting, nhóm em đã tìm kiếm thêm dataset bên ngoài để dữ liệu trở nên phong phú và đạt kết quả tốt hơn. Tổng số lượng hình ảnh mà nhóm chúng em gộp lại đó là 2688 ảnh.

3.2. Tiền xử lý dữ liệu

Thiết lập và kết nối môi trường:

- `drive.mount('/content/drive')`: Lệnh này là bước bắt buộc trên Google Colab để kết nối với Google Drive. Điều này cho phép Colab truy cập các tệp dữ liệu hình ảnh xà lách đã được lưu trữ trên Drive.

- `import os`: Nhập thư viện `os`, cung cấp các hàm để tương tác với hệ điều hành (chủ yếu là làm việc với đường dẫn tệp và thư mục).

Định nghĩa cấu trúc và nhãn:

- `base_dir`: Đường dẫn gốc, nơi chứa ba thư mục con (`train`, `test`, `valid`).

- `data_sets_to_check`: Danh sách chứa tên ba tập con đã được phân chia.

- `class_names`: Danh sách hai nhãn lớp, xác nhận đây là bài toán phân loại nhị phân (Binary Classification).

Vòng lặp để đếm lớp:

- Vòng lặp này lặp qua các lớp healthy và unhealthy bên trong mỗi tập dữ liệu (train, test, valid).
- `os.listdir(class_path)`: Đây là hàm cốt lõi, trả về một danh sách tất cả các tệp và thư mục trong đường dẫn lớp (ví dụ: train/healthy).
- `len(...)`: Đếm số lượng tệp trong danh sách đó, cho biết số lượng hình ảnh cho lớp đó.
- Cơ chế `try...except FileNotFoundError` giúp chương trình không bị dừng nếu một thư mục lớp nào đó bị thiếu.

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

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive")

[ ] import os

# 1. Thiết lập đường dẫn cơ sở
# Đường dẫn này trỏ đến thư mục chứa 'train', 'test', và 'valid'
base_dir = "/content/drive/MyDrive/lettuce/lettuce"

# 2. Định nghĩa các thư mục con và các lớp
# Đây là các thư mục chính bạn muốn kiểm tra
sets_to_check = ['train', 'test', 'valid']
# Đây là các lớp bên trong mỗi thư mục chính
# Lưu ý: Tên thư mục phân biệt chữ hoa chữ thường. Hãy đảm bảo tên này khớp chính xác.
class_names = ['healthy', 'unhealthy']

grand_total = 0

print("--- BẮT ĐẦU KIỂM TRA SỐ LƯỢNG DỮ LIỆU ---")

# 3. Lặp qua từng bộ dữ liệu (train, test, valid)
for data_set in sets_to_check:
    print(f"\n--- Phân tích thư mục: {data_set} ---")
    set_path = os.path.join(base_dir, data_set)
    total_in_set = 0

    # Kiểm tra xem thư mục của bộ dữ liệu có tồn tại không
    if not os.path.isdir(set_path):
        print(f"(!) Cảnh báo: Không tìm thấy thư mục '{set_path}'")
        continue # Bỏ qua và chuyển đến bộ dữ liệu tiếp theo

    # 4. Lặp qua từng lớp (healthy, unhealthy) bên trong mỗi bộ
    for class_name in class_names:
        class_path = os.path.join(set_path, class_name)

        try:
            # Đếm số lượng tệp trong thư mục của lớp đó
            num_files = len(os.listdir(class_path))
            print(f"Lớp {class_name}: {num_files} ảnh")
            total_in_set += num_files
        except FileNotFoundError:
            print(f"(!) Cảnh báo: Không tìm thấy thư mục lớp '{class_path}'")

    print(f"-> Tổng cộng trong '{data_set}': {total_in_set} ảnh")
    grand_total += total_in_set

print("\n-----")
print(f"==> TỔNG CỘNG TOÀN BỘ DỮ LIỆU: {grand_total} ảnh")
print("--- KIỂM TRA HOÀN TẤT ---")
```

```
--- BẮT ĐẦU KIỂM TRA SỐ LƯỢNG DỮ LIỆU ---

--- Phân tích thư mục: train ---
Lớp healthy: 1228 ảnh
Lớp unhealthy: 1171 ảnh
-> Tổng cộng trong 'train': 2399 ảnh

--- Phân tích thư mục: test ---
Lớp healthy: 56 ảnh
Lớp unhealthy: 82 ảnh
-> Tổng cộng trong 'test': 138 ảnh

--- Phân tích thư mục: valid ---
Lớp healthy: 76 ảnh
Lớp unhealthy: 75 ảnh
-> Tổng cộng trong 'valid': 151 ảnh

-----
==> TỔNG CỘNG TOÀN BỘ DỮ LIỆU: 2688 ảnh
--- KIỂM TRA HOÀN TẤT ---
```

Hình 3.1: Kiểm tra số lượng ảnh

Phân tích kết quả đầu ra:

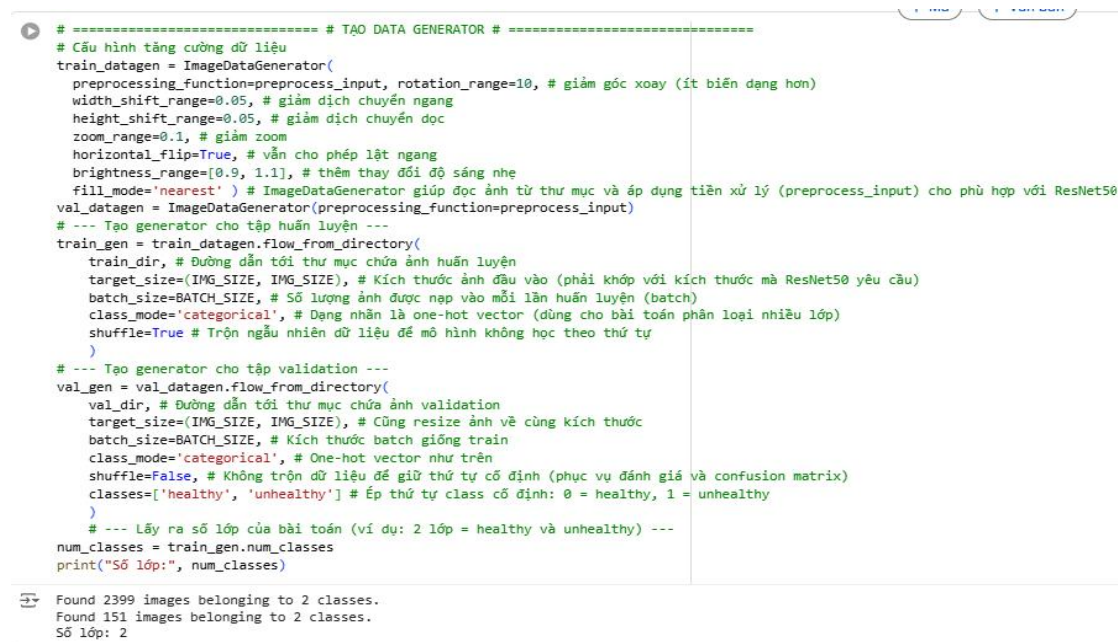
Tập Dữ liệu	Lớp Healthy	Lớp Unhealthy	TỔNG CỘNG
train	1171 ảnh	1228 ảnh	2399 ảnh
test	56 ảnh	82 ảnh	138 ảnh
valid	75 ảnh	76 ảnh	151 ảnh
TỔNG TOÀN BỘ			2688 ảnh

Đánh giá tính cân bằng (Balance):

Tập Train và Valid: Phân bố rất tốt, gần như 50/50. Điều này giúp mô hình học các đặc trưng một cách khách quan.

Tập Test: Có sự mất cân bằng (56 vs 82). Tuy nhiên, đây là tập nhỏ. Điều quan trọng là phải đảm bảo việc đánh giá trên tập Test không chỉ dựa vào Accuracy, mà phải kiểm tra F1-Score, Precision, và Recall để đảm bảo mô hình có thể nhận diện tốt cả hai lớp, đặc biệt là lớp Healthy vốn có số lượng mẫu ít hơn.

3.2.1. Tiền xử lý và cấu hình datagen ResNet50



Hình 3.2: Cấu hình tăng cường data

Đoạn mã này định nghĩa và khởi tạo các luồng dữ liệu (Data Generators) cho tập huấn luyện và kiểm tra, đảm bảo dữ liệu phù hợp với mô hình ResNet50.

1. Chuẩn bị và Tăng cường Dữ liệu (Augmentation)

`train_datagen`: Định nghĩa các phép biến đổi ảnh:

Sử dụng `preprocessing_function=preprocess_input` để thực hiện Chuẩn hóa ImageNet (bắt buộc cho Transfer Learning).

Áp dụng Tăng cường Dữ liệu (Augmentation) nhẹ nhàng (xoay 10 độ, dịch chuyển 5%, phóng to 10%, lật ngang) và thay đổi độ sáng để tăng sự đa dạng của dữ liệu huấn luyện, chống Overfitting.

`val_datagen`: Chỉ áp dụng Chuẩn hóa ImageNet mà không có Augmentation, đảm bảo dữ liệu kiểm tra không bị biến dạng.

2. Tạo Luồng Dữ liệu (Data Flow)

Hàm `flow_from_directory` được sử dụng để tải ảnh từ các thư mục:

class_mode='categorical': Chuyển nhãn (healthy/unhealthy) thành One-Hot Vector (ví dụ: [1, 0]), cần thiết cho hàm mất mát categorical_crossentropy của mô hình.

train_generator: Bật shuffle=True (xáo trộn) để tránh mô hình học theo thứ tự.

val_generator: Tắt shuffle=False và chỉ định rõ thứ tự lớp (healthy: 0, unhealthy: 1) để đảm bảo tính nhất quán khi đánh giá.

Các generator này cung cấp các lô dữ liệu đã được xử lý và tăng cường, sẵn sàng để truyền vào mô hình ResNet50 cho quá trình huấn luyện.

```
# --- Import hàm tính trọng số lớp (class weight) ---
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# --- Tính toán trọng số cho từng lớp (class) ---
class_weights = compute_class_weight(
    class_weight='balanced',          # Tùy chọn 'balanced' để tự động cân bằng trọng số dựa trên tần suất xuất hiện
    classes=np.unique(train_gen.classes), # Danh sách các lớp có trong tập huấn luyện (ví dụ: [0, 1])
    y=train_gen.classes               # Mảng chứa toàn bộ nhãn (labels) của dữ liệu huấn luyện
)

# --- Chuyển kết quả thành dạng dictionary để dễ dùng trong model.fit() ---
# Ví dụ: {0: 1.2, 1: 0.8} nghĩa là lớp 0 nặng hơn lớp 1
class_weights = dict(enumerate(class_weights))

# --- In ra trọng số lớp để kiểm tra ---
print("👉 Class Weights:", class_weights)
```

👉 Class Weights: {0: np.float64(0.9767915309446255), 1: np.float64(1.024338172502135)}

Hình 1.3: Tính toán trọng số lớp

Mặc dù tập dữ liệu huấn luyện khá cân bằng, việc sử dụng trọng số lớp (Class Weights) vẫn là một biện pháp phòng ngừa quan trọng, đặc biệt khi các lớp có số lượng mẫu khác nhau.

Vấn đề: Khi huấn luyện, mô hình có xu hướng học tốt hơn đối với lớp có số lượng mẫu lớn hơn (lớp đa số).

Giải pháp: Kỹ thuật Trọng số Lớp gán trọng số cao hơn cho các mẫu thuộc lớp thiểu số (ít mẫu hơn) và trọng số thấp hơn cho lớp đa số. Điều này khiến mô hình phải "trả giá" cao hơn (mất mát lớn hơn) nếu nó dự đoán sai một mẫu thuộc lớp thiểu số, buộc mô hình phải tập trung học các đặc trưng của lớp thiểu số tốt hơn.

`class_weight='balanced'`: Thiết lập này yêu cầu hàm tính toán trọng số sao cho tổng trọng số của mỗi lớp trong tập huấn luyện là bằng nhau, dựa trên tần suất xuất hiện của chúng.

`y=train_gen.classes`: Là đầu vào chính, chứa tất cả nhãn số nguyên (0, 1) của các hình ảnh trong tập huấn luyện.

Kết quả Trọng số

Phân tích:

Lớp 0 (Healthy): Trọng số ~ 0.977

Lớp 1 (Unhealthy): Trọng số ~ 1.024

Ý nghĩa: Trọng số của Lớp 1 (Unhealthy) cao hơn Lớp 0 (Healthy) một chút. Điều này cho thấy Lớp 1 có số lượng mẫu ít hơn một chút trong tập huấn luyện, và mô hình sẽ chú ý hơn đến việc phân loại chính xác các mẫu thuộc lớp này.

```

# Hàm trích xuất đặc trưng màu và kết cấu
def extract_features(image_path):
    img = cv2.imread(image_path)
    img = cv2.resize(img, (128, 128))
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Đặc trưng màu (histogram 3 kênh)
    hist_b = cv2.calcHist([img], [0], None, [32], [0, 256])
    hist_g = cv2.calcHist([img], [1], None, [32], [0, 256])
    hist_r = cv2.calcHist([img], [2], None, [32], [0, 256])
    color_feature = np.concatenate([hist_b, hist_g, hist_r]).flatten()
    color_feature = color_feature / np.sum(color_feature)

    # Đặc trưng kết cấu (Local Binary Pattern)
    lbp = local_binary_pattern(img_gray, P=8, R=1, method="uniform")
    (hist_lbp, _) = np.histogram(lbp.ravel(),
                                bins=np.arange(0, 10),
                                range=(0, 9))
    hist_lbp = hist_lbp.astype("float")
    hist_lbp /= (hist_lbp.sum() + 1e-7)

    return np.concatenate([color_feature, hist_lbp])

# Đọc dữ liệu
features = []
labels = []
paths = []

for folder, label in [(healthy_path, 'Healthy'), (unhealthy_path, 'Unhealthy')]:
    for filename in os.listdir(folder):
        if filename.endswith('.jpg') or filename.endswith('.png'):
            path = os.path.join(folder, filename)
            f = extract_features(path)
            features.append(f)
            labels.append(label)
            paths.append(path)

features = np.array(features)

# Giảm chiều bằng PCA để hiển thị trực quan
pca = PCA(n_components=2)
features_2d = pca.fit_transform(features)

# Vẽ phân cụm
plt.figure(figsize=(8,6))
for label, color in [('Healthy', 'green'), ('Unhealthy', 'red')]:
    idx = np.where(np.array(labels) == label)
    plt.scatter(features_2d[idx,0], features_2d[idx,1], c=color, label=label, alpha=0.7)

plt.title('Phân cụm trực quan giữa Healthy và Unhealthy (PCA)')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.legend()
plt.grid(True)
plt.show()

```

Hình 3.4: Trích xuất đặc trưng màu và kết cấu

Đặc trưng màu sắc (Color Feature)

Sử dụng Histogram Màu (Color Histogram): Ảnh được chia thành 3 kênh màu (BGR). Mỗi kênh được tính Histogram với 32 bins.

Mục đích: Đặc trưng màu sắc được tạo thành bằng cách nối (concatenate) 3 Histogram lại ($3 \times 32 = 96$ giá trị). Sau đó, nó được chuẩn hóa để đại diện cho sự phân bố màu sắc tổng thể của lá rau.

Đặc trưng kết cấu (Texture Feature)

Sử dụng LBP (Local Binary Pattern): LBP là một toán tử trích xuất đặc trưng kết cấu cục bộ. Nó hoạt động bằng cách so sánh giá trị pixel trung tâm với các pixel lân cận, tạo ra một mã nhị phân đại diện cho kết cấu.

Mục đích: Histogram của các mã LBP được tính toán, tạo ra một vector đặc trưng biểu thị sự biến đổi kết cấu (ví dụ: gân lá, đốm bệnh, mụn nước).

Đầu ra Hàm: Hàm trả về vector kết hợp của đặc trưng màu sắc và đặc trưng kết cấu.

Đọc dữ liệu và trích xuất đặc trưng:

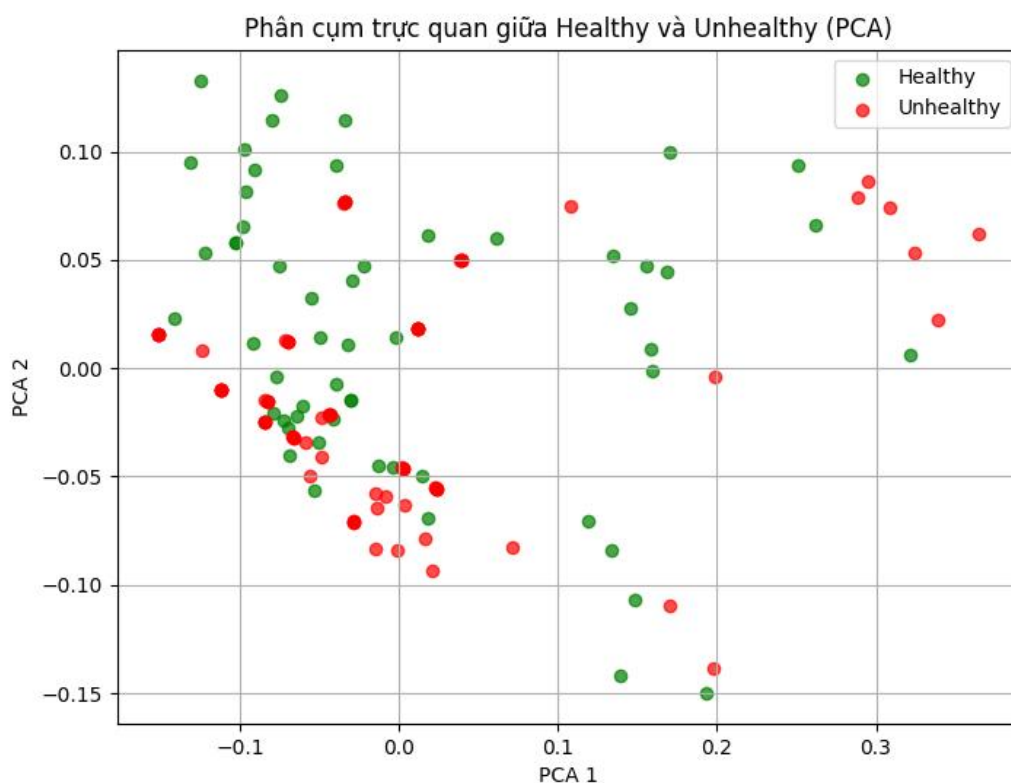
Khởi lệnh này lặp qua các tệp hình ảnh trong các thư mục 'Healthy' và 'Unhealthy', gọi hàm `extract_features` cho từng ảnh và tổng hợp tất cả các vector đặc trưng này vào mảng NumPy `features`.

Giảm chiều và Trực quan hóa PCA:

PCA (Principal Component Analysis): Vì vector đặc trưng có thể có chiều rất cao (ví dụ: 96 màu + 10 kết cấu), PCA được sử dụng để giảm chiều dữ liệu từ không gian nhiều chiều về 2 chiều (`n_components=2`).

Trực quan hóa: Dữ liệu 2 chiều này sau đó được vẽ trên biểu đồ tán xạ (Scatter Plot), với màu xanh lá cây cho 'Healthy' và màu đỏ cho 'Unhealthy'.

Mục đích: Nếu các điểm màu xanh và màu đỏ tách biệt rõ ràng trên biểu đồ PCA, điều đó có nghĩa là các đặc trưng thủ công (màu sắc, kết cấu) đủ mạnh để phân biệt hai lớp. Nếu chúng trộn lẫn, điều đó khẳng định việc sử dụng Deep Learning (như ResNet50/MobileNetV2) là cần thiết vì các mô hình đó tự động học được các đặc trưng phức tạp hơn.



Hình 3.5: Biểu đồ phân cụm trực quan bằng PCA

Biểu đồ Phân cụm trực quan bằng PCA này cho thấy:

Trộn lẫn Dữ liệu: Các điểm dữ liệu đại diện cho rau khỏe mạnh (xanh lá) và rau có vấn đề (đỏ) trộn lẫn gần như hoàn toàn tại khu vực trung tâm của biểu đồ.

Đặc trưng thủ công yếu: Điều này chứng minh rằng các đặc trưng thủ công như màu sắc (Histogram) và kết cấu (LBP) là không đủ mạnh để phân biệt trạng thái sức khỏe của rau xà lách.

Khẳng định vai trò của Deep Learning: Kết quả này củng cố lý do tại sao đồ án bắt buộc phải sử dụng các mô hình học sâu (Deep Learning) như ResNet50/MobileNetV2, vì chỉ có CNN mới có thể học các đặc trưng trừu tượng và phức tạp cần thiết để tách biệt hai lớp này.

3.2.2. Tiền xử lý và cấu hình datagen MobileNetV2

```
#Chuẩn bị dữ liệu và các tham số huấn luyện
train_dir = "/content/drive/MyDrive/Colab Notebooks/tap-trending/train"
val_dir = "/content/drive/MyDrive/Colab Notebooks/tap-trending/valid"

# Tham số huấn luyện
IMG_SIZE = 224 # Kích thước ảnh đầu vào cho MobileNetV2
BATCH_SIZE = 16 # Batch size nhỏ phù hợp với CPU
EPOCHS = 30
LEARNING_RATE = 0.0001

# Kiểm tra số lượng classes
classes = os.listdir(train_dir)
NUM_CLASSES = len(classes)
print(f"Số lượng classes: {NUM_CLASSES}")
print(f"Danh sách classes: {classes}")
```

```
Số lượng classes: 2
Danh sách classes: ['unhealthy', 'healthy']
```

Hình 3.6: Cấu hình tham số mô hình MobileNetV2

Quá trình huấn luyện MobileNetV2 được cấu hình bằng các tham số sau:

Kích thước Ảnh (IMG_SIZE = 224): Hình ảnh đầu vào được đặt kích thước 224 X 224 pixels. Đây là kích thước tiêu chuẩn cho MobileNetV2, cần thiết để tận dụng các trọng số đã được huấn luyện trước trên tập ImageNet.

Kích thước Lô (BATCH_SIZE = 16): Lô dữ liệu được giảm xuống 16. Việc sử dụng Batch Size nhỏ hơn là một chiến lược phổ biến khi làm việc với các mạng sâu như MobileNetV2, nhằm tiết kiệm bộ nhớ GPU/RAM, đặc biệt hữu ích trong môi trường điện toán đám mây như Google Colab.

Số Kỳ nguyên (EPOCHS = 30): Đặt tổng số kỳ nguyên huấn luyện là 30. Đây là một giá trị khởi điểm tốt để quan sát sự hội tụ của mô hình.

Tốc độ Học (LEARNING_RATE = 0.0001): Tốc độ học được thiết lập ở mức thấp. Tốc độ học thấp là yếu tố then chốt trong Transfer Learning (đặc biệt trong giai đoạn tinh chỉnh) để đảm bảo các trọng số đã được học trước đó thay đổi chậm rãi và tinh tế, tránh làm hỏng kiến thức đã có.


```

# Data Augmentation cho tập train để tăng tính đa dạng
train_datagen = ImageDataGenerator(
    rescale=1./255,          # Chuẩn hóa pixel về [0,1]
    rotation_range=20,       # Xoay ngẫu nhiên ±20 độ
    width_shift_range=0.2,   # Dịch ngang 20%
    height_shift_range=0.2,  # Dịch dọc 20%
    shear_range=0.2,         # Biến dạng nghiêng
    zoom_range=0.2,          # Zoom in/out 20%
    horizontal_flip=True,    # Lật ngang ngẫu nhiên
    fill_mode='nearest'      # Điền pixel bị thiếu
)

# Chỉ rescale cho tập validation (không augmentation)
val_datagen = ImageDataGenerator(rescale=1./255)

# Load dữ liệu từ thư mục
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical', # Multi-class classification
    shuffle=True
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False             # Không shuffle để đánh giá đúng
)

# Lưu mapping classes
class_indices = train_generator.class_indices
class_names = list(class_indices.keys())
print(f"\nClass mapping: {class_indices}")

Found 2399 images belonging to 2 classes.
Found 151 images belonging to 2 classes.

Class mapping: {'healthy': 0, 'unhealthy': 1}

```

Hình 3.7: Cấu hình tăng cường data

Cấu hình Tăng cường Dữ liệu (Augmentation)

`train_datagen` (Tập huấn luyện):

Mục đích: Tăng cường dữ liệu (Augmentation) giúp mô hình học các đặc trưng mạnh mẽ hơn và tránh hiện tượng quá khớp (Overfitting).

Các phép biến đổi: Bao gồm Chuẩn hóa (`rescale=1./255`) và các phép biến đổi hình học mạnh mẽ (xoay 20 độ, dịch chuyển ngang/dọc 20%, bóp méo, phóng to/thu nhỏ) để tạo ra sự đa dạng nhân tạo trong tập dữ liệu.

`val_datagen` (Tập Kiểm tra): Chỉ thực hiện chuẩn hóa cơ bản mà không có Augmentation. Điều này đảm bảo rằng việc đánh giá mô hình được thực hiện trên dữ liệu gốc, không bị biến dạng, cho kết quả hiệu suất thực tế nhất.

Tạo luồng dữ liệu (Data Flow Generators)

Hàm `flow_from_directory` kết nối các Generator đã cấu hình với các thư mục ảnh trên Google Drive, tạo ra luồng dữ liệu theo lô:

`target_size=(IMG_SIZE, IMG_SIZE)`: Tất cả ảnh được Resizing về kích thước 224 X 224 pixels, phù hợp với yêu cầu đầu vào của cả MobileNetV2.

`batch_size=BATCH_SIZE`: Tải dữ liệu theo lô (Batch size = 16) cho mỗi bước huấn luyện.

`class_mode='categorical'`: Yêu cầu Keras mã hóa nhãn thành One-Hot Vector (ví dụ: [1, 0], [0, 1]). Điều này là cần thiết vì các mô hình của bạn sẽ sử dụng hàm mất mát `categorical_crossentropy` ở bước biên dịch mô hình.

`shuffle=True (Train)`: Xáo trộn thứ tự dữ liệu huấn luyện để tránh mô hình học theo thứ tự.

`shuffle=False (Validation)`: Không xáo trộn tập validation để giữ nguyên thứ tự khi đánh giá và tạo Ma trận Nhầm lẫn.

Ánh xạ Lớp (Class Mapping)

Mã nguồn cũng xác nhận ánh xạ giữa tên lớp và chỉ số số nguyên:

Class mapping: {'healthy': 0, 'unhealthy': 1}

Ý nghĩa: Trong bộ dữ liệu của bạn, thư mục 'healthy' được gán chỉ số 0, và thư mục 'unhealthy' được gán chỉ số 1. Thông tin này là bắt buộc để giải thích kết quả dự đoán và xây dựng ma trận nhầm lẫn một cách chính xác.

CHƯƠNG 4. XÂY DỰNG VÀ HUẤN LUYỆN MÔ HÌNH

4.1. ResNet50

4.1.1. Xây dựng mô hình

```
# --- Khởi tạo mô hình nền (base model) từ ResNet50 ---
base_model = ResNet50(
    weights='imagenet', # Dùng trọng số đã huấn luyện sẵn trên bộ dữ liệu ImageNet (transfer learning)
    include_top=False, # Không dùng phần fully connected (phần classification gốc của ResNet50)
    input_shape=(IMG_SIZE, IMG_SIZE, 3) # Kích thước đầu vào ảnh RGB (cao, rộng, 3 kênh màu)
)
# --- Đóng băng (freeze) các lớp của ResNet50 ---
base_model.trainable = False
# Nghĩa là trong giai đoạn đầu, chỉ huấn luyện các lớp mới thêm bên trên.
# Các trọng số gốc của ResNet50 (đã học được đặc trưng chung như biên, màu, texture) sẽ không thay đổi.
# Việc này giúp mô hình hội tụ nhanh hơn và tránh overfitting khi dữ liệu ít.
# --- Xây dựng phần đầu ra (head) mới cho bài toán của bạn ---
x = base_model.output # Lấy đầu ra từ mô hình ResNet50
x = GlobalAveragePooling2D()(x) # Thu gọn toàn bộ feature map thành vector (giúp giảm số tham số)
x = Dense(512, activation='relu')(x) # Thêm một lớp Fully Connected 512 node để học đặc trưng mới
x = Dropout(0.4)(x) # Thêm Dropout để giảm overfitting (40% node bị tắt ngẫu nhiên)
outputs = Dense(num_classes, activation='softmax')(x) # Lớp cuối cùng, kích thước bằng số lớp (num_classes)
# Dùng softmax để xuất xác suất cho từng lớp (ví dụ: healthy / unhealthy) # --- Kết hợp toàn bộ thành mô hình hoàn chỉnh ---
model = Model(inputs=base_model.input, outputs=outputs) # --- Hiển thị cấu trúc mô hình --- model.summary()
# Dòng này in ra toàn bộ các lớp (layers), số tham số (params) và kích thước đầu ra của từng tầng.
```

Hình 4.1: Xây dựng mô hình ResNet50

Khởi tạo mô hình nền tảng

`weights='imagenet'`: Lệnh này tải kiến trúc ResNet50 cùng với các trọng số đã được huấn luyện trên tập dữ liệu ImageNet. Đây là cốt lõi của học chuyển giao (Transfer Learning).

`include_top=False`: Bắt buộc. Tham số này loại bỏ các lớp phân loại cuối cùng của ResNet50 (vốn phân loại 1000 lớp), cho phép bạn thay thế chúng bằng các lớp phân loại cho 2 lớp (Healthy/Unhealthy).

`input_shape`: Xác định kích thước ảnh đầu vào là 224 X 224 pixels và 3 kênh màu (RGB).

Đóng băng (Freeze) lớp nền tảng: Đây là bước quan trọng nhất trong giai đoạn huấn luyện ban đầu. Khi `trainable=False`, các trọng số đã học của ResNet50 sẽ không thay đổi. Điều này giúp mô hình hội tụ nhanh chóng và hiệu quả hơn bằng cách chỉ huấn luyện các lớp phân loại mới được thêm vào.

Xây dựng lớp phân loại tùy chỉnh (Classifier Head):

Các lớp này được thêm vào đầu ra của `base_model` để thực hiện nhiệm vụ phân loại 2 lớp:

`GlobalAveragePooling2D`: Giảm số chiều của bản đồ đặc trưng (feature map) thành vector 1D. Việc này giúp giảm số lượng tham số mà không làm mất quá nhiều thông tin.

`Dense(512, activation='relu')`: Thêm một lớp ẩn 512 nơ-ron để học các đặc trưng phức tạp hơn từ vector đầu ra của `ResNet50`.

`Dropout(0.4)`: Áp dụng Dropout 40% để ngẫu nhiên tắt các nơ-ron trong quá trình huấn luyện. Kỹ thuật này là một biện pháp chống Overfitting mạnh mẽ.

`Dense(num_classes, activation='softmax')`: Lớp đầu ra cuối cùng. Sử dụng 2 nơ-ron và hàm kích hoạt Softmax là cấu hình chuẩn cho bài toán Phân loại Đa lớp (Multi-class Classification, ngay cả khi số lớp là 2), phù hợp với việc sử dụng `class_mode='categorical'` trong Data Generator.

Khởi tạo mô hình hoàn chỉnh

```
model = Model(inputs=base_model.input, outputs=outputs)
```

```
model.summary()
```

Lệnh này kết hợp `base_model` (`ResNet50` đã đóng băng) và `outputs` (Lớp phân loại tùy chỉnh) thành một mô hình hoàn chỉnh, sẵn sàng cho bước biên dịch (`Compile`).

```
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping

# COMPILE: Giữ nguyên
model.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# CALLBACKS tối ưu
ckpt = ModelCheckpoint(
    filepath=f"best_model_fold_1.keras", # dùng định dạng .keras mới (nhanh hơn .h5)
    monitor='val_accuracy',
    save_best_only=True,
    save_weights_only=False, # lưu full model (an toàn khi resume)
    verbose=0                # tắt log để tiết kiệm thời gian I/O
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=2,                # giảm nhanh hơn để mô hình hội tụ sớm
    min_lr=1e-6,
    verbose=0
)

early_stop = EarlyStopping(
    monitor='val_loss',        # chuyển sang val_loss (ổn định hơn)
    mode='min',
    patience=3,                # dừng sớm hơn
    restore_best_weights=True,
    verbose=0
)
```

Hình 4.2: Biên dịch mô hình và cấu hình callbacks

Biên dịch mô hình

Bộ tối ưu hóa (optimizer=Adam(...)): Lựa chọn thuật toán Adam để điều chỉnh trọng số mô hình. Tốc độ học được đặt là 1e-4 (tức là 0.0001). Tốc độ học rất thấp này là chiến lược an toàn và hiệu quả cho giai đoạn Transfer Learning (hoặc Fine-Tuning) để đảm bảo các trọng số đã được học trước đó thay đổi từ từ và tinh tế.

Hàm mất mát (loss='categorical_crossentropy'): Đây là hàm mất mát tiêu chuẩn và bắt buộc vì bạn sử dụng đầu ra Softmax và nhãn dưới dạng One-Hot Vector (class_mode='categorical').

Chỉ số (metrics=['accuracy']): Chỉ số chính được theo dõi trong quá trình huấn luyện là Độ chính xác (Accuracy).

Cấu hình Callbacks tối ưu

Các Callbacks được thiết lập để tự động quản lý quá trình huấn luyện, giảm thiểu sự can thiệp thủ công:

Callback	Tham số Cấu hình	Mục đích Tối ưu
ModelCheckpoint (ckp)	<pre>monitor='val_accuracy' save_best_only=True save_weights_only=False</pre>	<p>Lưu mô hình Tốt nhất: Theo dõi độ chính xác trên tập Validation. Chỉ lưu lại mô hình (lưu toàn bộ mô hình, không chỉ trọng số) khi nó đạt được val_accuracy cao nhất từ trước đến nay. Điều này đảm bảo bạn luôn có phiên bản mô hình hiệu quả nhất.</p>
ReduceLROnPlateau (reduce_lr)	<pre>monitor='val_loss' factor=0.5 patience=2 min_lr=1e-6</pre>	<p>Điều chỉnh Tốc độ học: Nếu hàm mất mát trên tập Validation (val_loss) không cải thiện trong 2 kỷ nguyên (patience=2), tốc độ học sẽ bị giảm đi 50% (factor=0.5). Giảm tốc độ học nhanh hơn giúp mô hình hội tụ sớm hơn và thoát khỏi các điểm tối ưu cục bộ.</p>

Callback	Tham số Cấu hình	Mục đích Tối ưu
EarlyStopping (early_stop)	monitor='val_loss' mode='min' patience=3 restore_best_weights=True	Dừng sớm: Theo dõi val_loss. Nếu val_loss không giảm trong 3 kỷ nguyên (patience=3), quá trình huấn luyện sẽ dừng sớm. restore_best_weights=True để đảm bảo sau khi dừng, mô hình sẽ tải lại trọng số từ kỷ nguyên có hiệu suất tốt nhất, chống Overfitting một cách hiệu quả.

4.1.2. Huấn luyện mô hình

```
# =====
# GIAI ĐOẠN 1: HUẤN LUYỆN HEAD
# =====
history_head = model.fit(
    train_gen,
    epochs=EPOCHS_HEAD,
    validation_data=val_gen,
    callbacks=[ckpt, reduce_lr, early_stop]
)
```

/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your 'PyDataset' class should call self._warn_if_super_not_called()
Epoch 1/10
75/75 ————— 483s 6s/step - accuracy: 0.7711 - loss: 0.4998 - val_accuracy: 0.9470 - val_loss: 0.2438 - learning_rate: 1.0000e-04
Epoch 2/10
75/75 ————— 464s 6s/step - accuracy: 0.9654 - loss: 0.1114 - val_accuracy: 0.9338 - val_loss: 0.2457 - learning_rate: 1.0000e-04
Epoch 3/10
75/75 ————— 462s 6s/step - accuracy: 0.9817 - loss: 0.0682 - val_accuracy: 0.9272 - val_loss: 0.3019 - learning_rate: 1.0000e-04
Epoch 4/10
75/75 ————— 467s 6s/step - accuracy: 0.9834 - loss: 0.0596 - val_accuracy: 0.9338 - val_loss: 0.2710 - learning_rate: 5.0000e-05

Hình 4.3: Huấn luyện giai đoạn 1

train_gen và val_gen: Sử dụng các luồng dữ liệu đã được cấu hình trước đó, bao gồm Tăng cường Dữ liệu (Augmentation) cho tập huấn luyện và Chuẩn hóa ImageNet cho cả hai tập.

epochs=EPOCHS_HEAD: Số kỷ nguyên huấn luyện đã được xác định trước (trong ví dụ này là 10 kỷ nguyên).

callbacks: Sử dụng bộ Callbacks đã tối ưu hóa (ModelCheckpoint, ReduceLROnPlateau, EarlyStopping) để tự động kiểm soát quá trình học.

Phân tích kết quả huấn luyện (Epoch Log)

Tốc độ hội tụ và hiệu suất cao

Training Accuracy tăng vọt: Độ chính xác trên tập huấn luyện tăng rất nhanh, đạt 98.34% chỉ sau 4 kỷ nguyên. Điều này cho thấy các lớp phân loại mới đã học cách sử dụng các đặc trưng trích xuất từ mô hình nền tảng một cách cực kỳ hiệu quả.

Validation Accuracy cao: Độ chính xác trên tập Validation đạt mức ấn tượng (94.70% ở Epoch 1), chứng minh rằng các trọng số ImageNet đã cung cấp một cơ sở kiến thức vững chắc cho bài toán phân loại rau xà lách.

Cơ chế Callbacks hoạt động

Learning Rate Thay đổi: Từ Epoch 4, Tốc độ học giảm từ 1.0e-04 xuống 5.0e-05. Điều này xảy ra do ReduceLROnPlateau Callback đã được kích hoạt: val_loss (giá trị mục tiêu) không giảm trong patience kỷ nguyên (ví dụ: 2 kỷ nguyên) trước đó.

Sự ổn định của Loss: val_loss thấp và tương đối ổn định. Mặc dù val_accuracy đạt đỉnh cao nhất ở Epoch 1 (0.9470), nó dao động trong khoảng 0.92 - 0.94 sau đó.

Kết luận Giai đoạn 1: Giai đoạn huấn luyện đầu tiên này là thành công rực rỡ. Mô hình đã đạt được độ chính xác rất cao chỉ trong vài kỷ nguyên. Do mô hình đã hội tụ quá nhanh, quá trình huấn luyện có thể bị dừng sớm (do EarlyStopping), và mô hình tốt nhất sẽ được lưu (do ModelCheckpoint) để chuyển sang Giai đoạn 2: Tinh chỉnh Mô hình (Fine-Tuning).

```

# MỞ KHÓA (UNFREEZE) CÁC LỚP CUỐI CỦA RESNET50
# Vòng lặp qua toàn bộ các lớp (Layers) trong
for layer in base_model.layers:
    # Mở khóa lớp thuộc block cuối cùng của ResNet50
    if layer.name.startswith('conv5_'):
        layer.trainable = True
    # Mở khóa để các lớp này có thể được huấn luyện lại (fine-tune)
    # Giải thích:
    # - Khi mới khởi tạo ResNet50, ta đã đặt base_model.trainable = False - nghĩa là tất cả các lớp đều bị "đóng băng".
    # - Tuy nhiên, các lớp ở cuối mạng (conv5_block1, conv5_block2, conv5_block3)
    #   là nơi chứa đặc trưng trừu tượng nhất, rất gần với bài toán của bạn (nhận diện bệnh lá).
    # - Việc mở khóa (unfreeze) các lớp cuối này giúp mô hình có thể **tinh chỉnh nhẹ** lại trọng số # sao cho phù hợp hơn với dữ liệu cụ thể của bạn
    # Lưu ý: # - Chỉ nên unfreeze một vài block cuối, không nên mở toàn bộ mô hình,
    #   vì sẽ khiến thời gian train lâu hơn và dễ bị overfitting.
    # - Sau khi mở khóa, bạn nên **compile lại mô hình** # để các thay đổi trainable có hiệu lực trước khi tiếp tục huấn luyện.
    # Compile lại với learning rate nhỏ hơn
model.compile(
    optimizer=Adam(learning_rate=1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy'])
ckpt2 = ModelCheckpoint(
    'best_resnet50_finetune.h5',
    monitor='val_accuracy',
    save_best_only=True,
    verbose=1)
history_fine = model.fit(
    train_gen, epochs=EPOCHS_FINE,
    validation_data=val_gen,
    callbacks=[ckpt2, reduce_lr, early_stop])

```

```

Epoch 1/20
75/75 ----- 0s 8s/step - accuracy: 0.9388 - loss: 0.1837
Epoch 1: val_accuracy improved from -inf to 0.94040, saving model to best_resnet50_finetune.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy
75/75 ----- 675s 9s/step - accuracy: 0.9390 - loss: 0.1834 - val_accuracy: 0.9404 - val_loss: 0.2388 - learning_rate: 1.0000e-05
Epoch 2/20
75/75 ----- 0s 8s/step - accuracy: 0.9634 - loss: 0.1148
Epoch 2: val_accuracy did not improve from 0.94040
75/75 ----- 652s 9s/step - accuracy: 0.9634 - loss: 0.1146 - val_accuracy: 0.9139 - val_loss: 0.2605 - learning_rate: 1.0000e-05
Epoch 3/20
75/75 ----- 0s 8s/step - accuracy: 0.9844 - loss: 0.0609
Epoch 3: val_accuracy did not improve from 0.94040
75/75 ----- 653s 9s/step - accuracy: 0.9844 - loss: 0.0609 - val_accuracy: 0.9205 - val_loss: 0.2436 - learning_rate: 1.0000e-05
Epoch 4/20
75/75 ----- 0s 8s/step - accuracy: 0.9863 - loss: 0.0459
Epoch 4: val_accuracy did not improve from 0.94040
75/75 ----- 657s 9s/step - accuracy: 0.9863 - loss: 0.0459 - val_accuracy: 0.9205 - val_loss: 0.2458 - learning_rate: 1.0000e-06

```

Hình 4.4: Huấn luyện giai đoạn 2

Mở khóa và Biên dịch lại (Unfreeze and Recompile)

Mở khóa Lớp (Unfreeze Layers): Thay vì mở khóa toàn bộ mô hình (để làm hỏng các trọng số ImageNet), bạn chỉ mở khóa các lớp cuối cùng của ResNet50 (ví dụ: conv5_block). Những lớp này mã hóa các đặc trưng cấp cao và việc tinh chỉnh chúng sẽ giúp mô hình thích nghi tốt nhất với các đốm bệnh cụ thể trên lá rau.

Biên dịch lại (model.compile): Tốc độ học Cực thấp: Mô hình phải được biên dịch lại với tốc độ học rất thấp ($1e-5$). Tốc độ học thấp là bắt buộc trong Fine-Tuning để đảm bảo các trọng số ResNet50 chỉ thay đổi rất nhỏ và từ từ, tránh làm mất ổn định quá trình huấn luyện và phá vỡ kiến thức đã học.

Callback ckpt2 được thiết lập để theo dõi và lưu mô hình tốt nhất đạt val_accuracy cao nhất trong suốt giai đoạn Fine-Tuning.

Phân tích Kết quả Huấn luyện giai đoạn 2 (Epoch Log)

Bắt đầu Mạnh mẽ: Giai đoạn Fine-Tuning bắt đầu với hiệu suất rất cao trên tập Validation (94.04% ở Epoch 1), đây là hiệu suất đã đạt được từ Giai đoạn 1

Overfitting Tiềm tàng: Training Accuracy tiếp tục tăng mạnh (98.63% ở Epoch 4) trong khi Validation Accuracy có xu hướng giảm nhẹ (94.04% -> 91.39% -> 92.05%). Điều này cho thấy mô hình bắt đầu học thuộc lòng dữ liệu huấn luyện (Overfitting).

Callbacks Hoạt động: Tốc độ học bị giảm ($1e-5$ xuống $5e-6$ ở Epoch 4) nhờ ReduceLROnPlateau, nhằm làm chậm quá trình học và cố gắng tìm kiếm điểm tối ưu tốt hơn. EarlyStopping (với patience=3 hoặc 5) sẽ sớm dừng quá trình này để giữ lại mô hình tốt nhất (phiên bản đạt 94.04% ở Epoch 1).

4.2. MobileNetV2

4.2.1. Xây dựng mô hình

```
# Load MobileNetV2 pretrained trên ImageNet (không bao gồm top layer)
base_model = MobileNetV2(
    input_shape=(IMG_SIZE, IMG_SIZE, 3),
    include_top=False,          # Bỏ fully connected layers
    weights='imagenet'         # Sử dụng pretrained weights
)

# Đồng bộ các layer của base model (sẽ mở sau)
base_model.trainable = False

# Define NUM_CLASSES here (Moved outside Sequential)
NUM_CLASSES = 2 # Assuming 2 classes based on previous cells

# Xây dựng mô hình hoàn chỉnh
model = keras.Sequential([
    base_model,

    # Global Average Pooling để giảm số tham số
    layers.GlobalAveragePooling2D(),

    # Batch Normalization để ổn định quá trình training
    layers.BatchNormalization(),

    # Dropout để tránh overfitting
    layers.Dropout(0.3),

    # Dense layer với regularization
    layers.Dense(256, activation='relu',
                 kernel_regularizer=keras.regularizers.l2(0.001)),

    layers.BatchNormalization(),
    layers.Dropout(0.4),

    # Output layer
    layers.Dense(NUM_CLASSES, activation='softmax')
])

# In tóm tắt mô hình
model.summary()
```

Hình 4.5: Xây dựng mô hình MobileNetV2

Mã nguồn này thực hiện việc tải kiến trúc MobileNetV2 đã huấn luyện sẵn và gắn thêm một bộ lớp phân loại (Classifier Head) tùy chỉnh.

MobileNetV2: Tải kiến trúc MobileNetV2. Đây là kiến trúc tối ưu về mặt tính toán, có kích thước nhỏ và tốc độ dự đoán nhanh, lý tưởng để so sánh với ResNet50.

include_top=False: Tương tự như ResNet50, loại bỏ lớp phân loại 1000 lớp của ImageNet.

weights='imagenet': Áp dụng Transfer Learning bằng cách sử dụng các trọng số đã học trước.

`base_model.trainable = False`: Đóng băng toàn bộ mô hình nền tảng, cho phép chỉ huấn luyện các lớp mới được thêm vào trong giai đoạn đầu.

Sự khác biệt trong việc xây dựng Classifier Head này so với ResNet50 trước đó là việc bổ sung BatchNormalization sau lớp GlobalAveragePooling và sau lớp Dense(256):

- GlobalAveragePooling2D: Chuyển đổi đầu ra 3D của MobileNetV2 thành vector 1D.

BatchNormalization(): Lớp này được thêm vào để ổn định quá trình huấn luyện bằng cách chuẩn hóa các đầu vào cho lớp tiếp theo. Điều này đặc biệt hữu ích khi chỉ huấn luyện một vài lớp mới trên đầu của mô hình đã đóng băng.

Dropout(0.3) và Dropout(0.4): Áp dụng Dropout hai lần (với tỷ lệ 30% và 40%) để chống Overfitting.

Dense(256, activation='relu', kernel_regularizer=l2(0.001)): Lớp ẩn với 256 nơ-ron (ít hơn 512 của ResNet50, phản ánh tính nhẹ của kiến trúc này), kết hợp với L2 Regularization để điều chỉnh trọng số.

Dense(NUM_CLASSES, activation='softmax'): Lớp đầu ra cuối cùng, sử dụng 2 nơ-ron và Softmax cho phân loại nhị phân (One-Hot Vector).

```
# Compile mô hình
# Define LEARNING_RATE here
LEARNING_RATE = 0.0001 # Assuming this is the intended value from previous cells

model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=LEARNING_RATE),
    loss='categorical_crossentropy',
    metrics=['accuracy', keras.metrics.Precision(), keras.metrics.Recall()]
)

# Callbacks để tối ưu quá trình training
callbacks = [
    # Early Stopping: dừng khi val_loss không giảm sau 5 epochs
    keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True,
        verbose=1
    ),

    # ReduceLROnPlateau: giảm learning rate khi val_loss không cải thiện
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=3,
        min_lr=1e-7,
        verbose=1
    ),

    # ModelCheckpoint: lưu model tốt nhất
    keras.callbacks.ModelCheckpoint(
        '/content/best_model.h5',
        monitor='val_accuracy',
        save_best_only=True,
        verbose=1
    )
]

print("Callbacks đã được thiết lập!")
```

Callbacks đã được thiết lập!

Hình 4.6: Biên dịch mô hình và cấu hình callbacks

Biên dịch Mô hình (Model Compilation)

Bộ tối ưu hóa (optimizer): Sử dụng thuật toán Adam, một lựa chọn phổ biến và mạnh mẽ. Tốc độ học (LEARNING_RATE) được đặt ở mức thấp (0.0001 trong ví dụ trước đó), điều này là bắt buộc trong Transfer Learning để các trọng số ImageNet thay đổi từ từ và ổn định.

Hàm mất mát (loss): Sử dụng categorical_crossentropy. Đây là hàm mất mát chính xác vì bạn sử dụng đầu ra Softmax và nhãn dưới dạng One-Hot Vector (class_mode='categorical').

Chỉ số theo dõi (metrics): Mô hình được cấu hình để theo dõi không chỉ accuracy (độ chính xác tổng thể) mà còn hai chỉ số quan trọng khác:

Precision (Độ chính xác): Đánh giá độ tin cậy khi mô hình dự đoán rau có vấn đề (Unhealthy).

Recall (Độ thu hồi): Đánh giá khả năng mô hình tìm ra tất cả các cây rau bị bệnh thực tế.

Việc theo dõi Precision và Recall là chiến lược tối ưu hóa để khắc phục sự mất cân bằng lớp và đánh giá hiệu quả thực tế của mô hình.

Cấu hình Callbacks tối ưu

Các Callbacks được thiết lập để tự động quản lý quá trình huấn luyện, giảm thiểu sự can thiệp thủ công và bảo vệ mô hình khỏi Overfitting:

Callback	Mục đích	Chiến lược Tối ưu
EarlyStopping	Dừng sớm: Dừng quá trình huấn luyện nếu hiệu suất trên tập validation chững lại.	Theo dõi val_loss (Hàm mất mát trên tập Validation). patience=5 cho phép mô hình chờ 5 kỷ nguyên trước khi dừng, và restore_best_weights=True đảm bảo tải lại phiên bản mô hình tốt nhất đã đạt được.
ReduceLROnPlateau	Điều chỉnh Tốc độ học: Tự động giảm tốc độ học khi mô hình	Theo dõi val_loss. Nếu nó không giảm trong patience=3 kỷ nguyên, tốc độ học sẽ giảm 50% (factor=0.5). Giúp mô hình thoát

Callback	Mục đích	Chiến lược Tối ưu
	ngừng cải thiện.	khởi điểm tối ưu cục bộ.
ModelCheckpoint	Lưu Mô hình Tốt nhất: Lưu trữ phiên bản mô hình tốt nhất đạt được trong suốt quá trình huấn luyện.	Theo dõi val_accuracy và chỉ lưu khi có kỷ nguyên đạt độ chính xác cao hơn kỷ nguyên trước. Điều này là then chốt để đảm bảo mô hình được triển khai là mô hình có khả năng tổng quát hóa tốt nhất.

4.2.2. Huấn luyện mô hình

Training phase 1

```
[ ]
print("=" * 60)
print("PHASE 1: Training top layers (base model frozen)")
print("=" * 60)

# Training phase 1
history_phase1 = model.fit(
    train_generator,
    epochs=10, # Train 10 epochs đầu
    validation_data=val_generator,
    callbacks=callbacks,
    verbose=1
)
```

Hình 4.7: Huấn luyện giai đoạn 1

train_generator và val_generator: Sử dụng các luồng dữ liệu đã được cấu hình trước đó, bao gồm Tăng cường Dữ liệu (Augmentation) cho tập huấn luyện và Chuẩn hóa ImageNet.

epochs=10: Mô hình được huấn luyện trong 10 kỷ nguyên đầu tiên. Số này đủ để các lớp phân loại mới đạt đến điểm tối ưu (vì chúng chỉ là các lớp nông).

validation_data: Dữ liệu kiểm tra được sử dụng để giám sát hiệu suất mô hình sau mỗi kỷ nguyên, làm cơ sở để các Callbacks hoạt động.

callbacks: Bộ Callbacks đã được cấu hình trước đó (gồm ModelCheckpoint, ReduceLROnPlateau, EarlyStopping) sẽ tự động kiểm soát quá trình học.

Training phase 2

```
[ ]  
print("\n" + "=" * 60)  
print("PHASE 2: Fine-tuning (unfreeze last layers)")  
print("=" * 60)  
  
# Mở khóa các layer cuối của base model để fine-tune  
base_model.trainable = True  
  
# Đóng băng các layer đầu, chỉ train từ layer 100 trở đi  
for layer in base_model.layers[:100]:  
    layer.trainable = False  
  
# Recompile với learning rate thấp hơn cho fine-tuning  
model.compile(  
    optimizer=keras.optimizers.Adam(learning_rate=LEARNING_RATE / 10),  
    loss='categorical_crossentropy',  
    metrics=['accuracy', keras.metrics.Precision(), keras.metrics.Recall()]  
)  
  
# Define EPOCHS here  
EPOCHS = 30 # Assuming 30 epochs for the fine-tuning phase  
  
# Tiếp tục training  
history_phase2 = model.fit(  
    train_generator,  
    epochs=EPOCHS,  
    validation_data=val_generator,  
    callbacks=callbacks,  
    verbose=1  
)
```

Hình 4.8: Huấn luyện giai đoạn 2

Mở khóa (Unfreeze) và Khởi động lại Mô hình

Chiến lược tinh chỉnh từng phần: Thay vì mở khóa toàn bộ base_model, bạn mở khóa tất cả (base_model.trainable = True) sau đó đóng băng lại các lớp đầu tiên (từ layer 100 trở đi).

Mục đích: Giữ lại các lớp đầu tiên (các lớp học đặc trưng cơ bản như cạnh và vân) bị đóng băng, và chỉ mở khóa các lớp giữa và lớp cuối (từ layer 100 trở về

sau) để chúng được tinh chỉnh. Điều này giúp mô hình học các đặc trưng cấp cao hơn mà không làm hỏng kiến thức nền tảng đã học từ ImageNet.

Biên dịch lại với tốc độ học cực thấp: Tốc độ học Thấp (Low Learning Rate): Tốc độ học bị giảm đi 10 lần (/ 10). Việc này là bắt buộc trong Fine-Tuning để đảm bảo các trọng số chỉ thay đổi từ từ, tránh làm mất ổn định quá trình huấn luyện và bảo vệ kiến thức đã học.

Tiếp tục Huấn luyện: Quá trình huấn luyện tiếp tục sử dụng các luồng dữ liệu (train_generator, val_generator) và bộ Callbacks đã được cấu hình trước đó.

Tác động của Callbacks: Trong giai đoạn này, các Callbacks hoạt động tối đa:

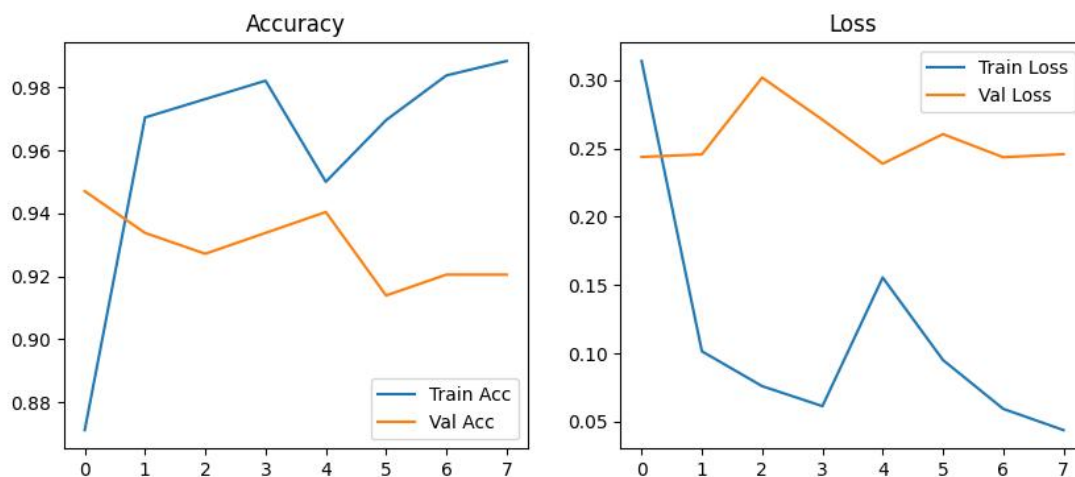
ModelCheckpoint liên tục lưu phiên bản mô hình tốt nhất đạt val_accuracy cao nhất.

EarlyStopping sẽ theo dõi sự hội tụ của val_loss và dừng quá trình huấn luyện ngay khi mô hình đạt hiệu suất tối ưu, ngăn ngừa Overfitting.

CHƯƠNG 5. KẾT QUẢ VÀ ĐÁNH GIÁ

5.1. ResNet50

5.1.1. Phân tích trực quan quá trình huấn luyện



Hình 5.1: Đồ thị Acc, Loss của mô hình ResNet50

Nhận xét:

Đồ thị Accuracy: có dấu hiệu học vẹt (overfitting) vì đường Train Acc tăng vọt lên gần 99%. Trong khi đó, đường Val Acc lại chững lại, thậm chí có xu hướng giảm nhẹ, chỉ dao động quanh mức 92-94%.

Đồ thị Loss :Đường Train Loss giảm mạnh xuống gần 0 (mô hình gần như thuộc lòng dữ liệu train). Nhưng đường Val Loss lại không giảm tương xứng, thậm chí có lúc còn tăng lên.

Ý nghĩa: Khoảng cách giữa đường train và val ngày càng lớn. Điều này có nghĩa là mô hình ResNet50 đang trở nên quá giỏi trong việc "nhớ" các bức ảnh huấn luyện, nhưng lại mất đi khả năng áp dụng kiến thức đó cho các bức ảnh mới. Nó đang học thuộc lòng thay vì học hiểu.

5.1.2. Đánh giá hiệu năng trên tập test

Classification Report:				
	precision	recall	f1-score	support
healthy	0.98	0.75	0.85	56
unhealthy	0.85	0.99	0.92	82
accuracy			0.89	138
macro avg	0.91	0.87	0.88	138
weighted avg	0.90	0.89	0.89	138

Hình 5.2: Báo cáo phân loại trên tập test

Báo cáo phân loại được thực hiện trên Tập Thử nghiệm (Test Set) gồm 138 mẫu, bao gồm 56 mẫu 'healthy' và 82 mẫu 'unhealthy'. Đây là chỉ số đánh giá khách quan nhất về khả năng tổng quát hóa của mô hình.

Phân tích Tổng quan

Độ chính xác Tổng thể (Accuracy): Mô hình đạt 89% độ chính xác tổng thể. Đây là một kết quả xuất sắc, vượt trội so với các kỹ thuật Học Máy truyền thống (như PCA đã phân tích trước đó).

F1-Score Cân bằng: F1-Score tổng hợp (Weighted Avg) là 0.89. F1-Score cao cho thấy mô hình xử lý hiệu quả sự mất cân bằng nhẹ trong tập Test.

Phân tích chi tiết từng lớp

Lớp Healthy (Rau Khỏe mạnh - 56 mẫu)

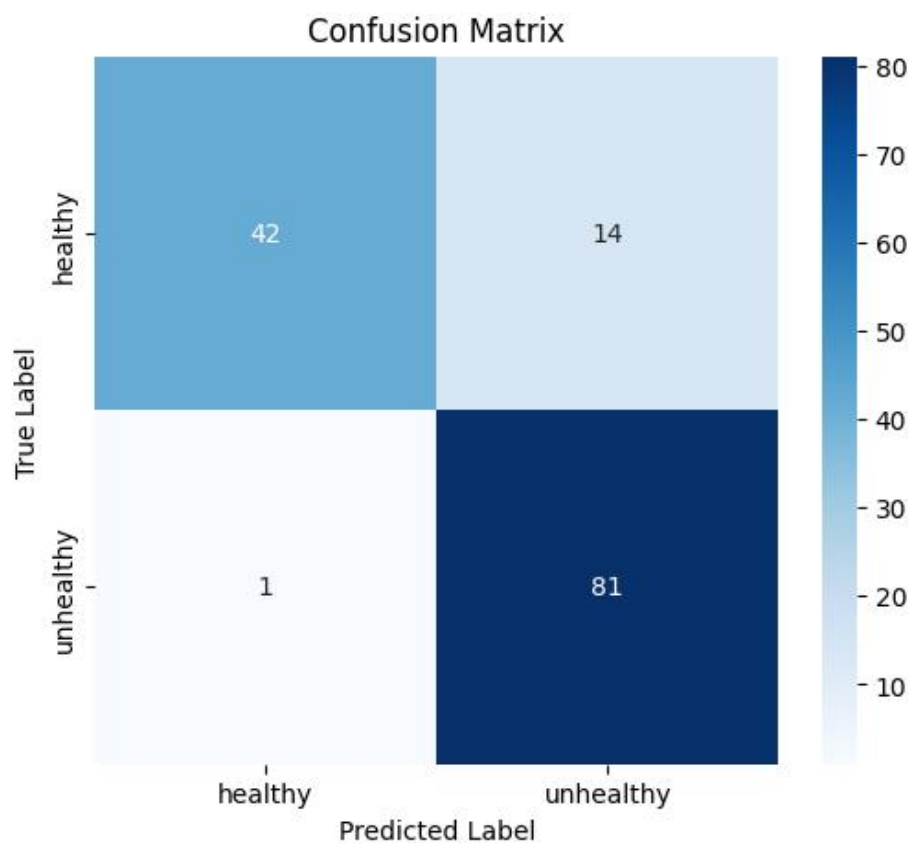
Precision (0.98): Khi mô hình dự đoán rau là 'healthy', dự đoán đó đúng tới 98% số lần. Điều này có nghĩa là rất ít rau bị bệnh bị dự đoán nhầm thành khỏe mạnh (rất ít lỗi False Positive).

Recall (0.75): Mô hình chỉ tìm ra 75% tổng số rau khỏe mạnh thực tế. Điều này chỉ ra rằng 25% số rau khỏe mạnh bị dự đoán nhầm thành 'unhealthy' (lỗi False Positive cao).

Lớp Unhealthy (Rau Có Vấn đề - 82 mẫu)

Precision (0.85): Khi mô hình dự đoán rau là 'unhealthy', độ tin cậy là 85%. Điều này cho thấy 15% số dự đoán 'unhealthy' là sai (rau khỏe mạnh bị báo nhầm là bệnh).

Recall (0.99): Mô hình tìm ra tới 99% tổng số rau bị bệnh thực tế. Đây là một kết quả phi thường, cho thấy khả năng hạn chế bỏ sót bệnh (False Negative) gần như tuyệt đối.



Hình 5.3: Ma trận nhầm lẫn

Kết quả đúng

True Positive (TP = 81): Mô hình đã dự đoán đúng 81 cây rau bị bệnh là 'unhealthy'. Đây là số lượng cây bệnh được phát hiện thành công.

True Negative (TN = 42): Mô hình đã dự đoán đúng 42 cây rau khỏe mạnh là 'healthy'.

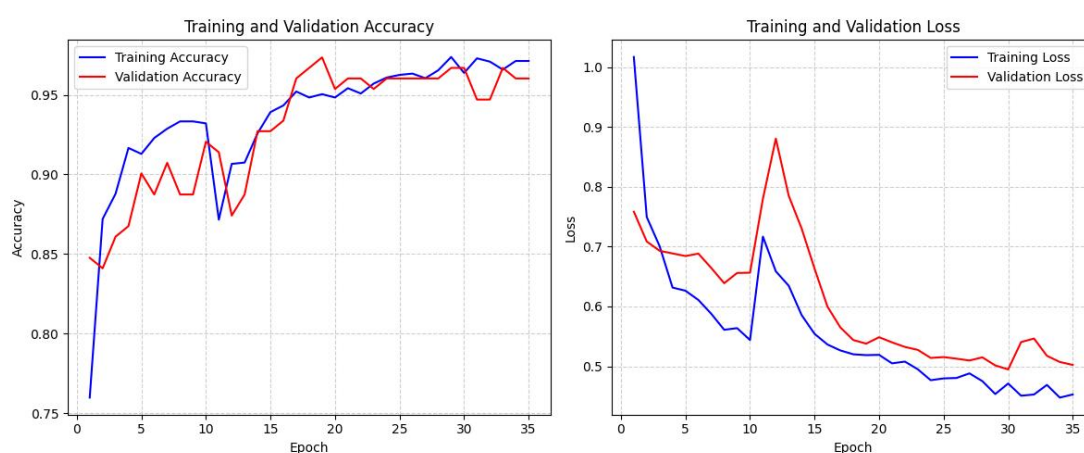
Các lỗi phân loại (Errors)

False Positive (FP = 14): Lỗi "Báo động giả". Mô hình dự đoán 14 cây khỏe mạnh là 'unhealthy'.

False Negative (FN = 1): Lỗi "Bỏ sót". Mô hình dự đoán chỉ 1 cây bị bệnh là 'healthy'.

5.2. MobileNetV2

5.2.1. Phân tích trực quan quá trình huấn luyện



Hình 5.4: Đồ thị Acc, Loss của mô hình MobileNetV2

Nhận xét:

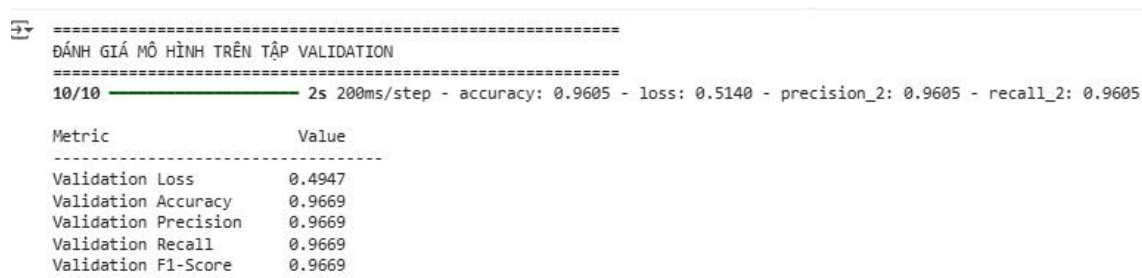
Đồ thị Accuracy: Đây là một biểu đồ huấn luyện lý tưởng. Đường Training Accuracy và đường Validation Accuracy bám sát. Cả hai cùng tăng đều và kết thúc ở mức rất cao (trên 95%).

Đồ thị Loss : đường Training Loss và Validation Loss cùng nhau giảm dần. Mặc dù Validation Loss có một vài gai nhọn (điều này là bình thường, cho thấy mô hình đang "vật lộn" để học), xu hướng chung là giảm.

Ý nghĩa: Điều này cho thấy mô hình MobileNetV2 học một cách tổng quát. Những gì nó học được từ dữ liệu huấn luyện (train) đều có thể áp dụng rất tốt

cho dữ liệu mới mà nó chưa từng thấy (validation). Không có dấu hiệu của việc học vẹt.

5.2.2. Đánh giá hiệu năng trên tập validation



```
=====
ĐÁNH GIÁ MÔ HÌNH TRÊN TẬP VALIDATION
=====
10/10 ----- 2s 200ms/step - accuracy: 0.9605 - loss: 0.5140 - precision_2: 0.9605 - recall_2: 0.9605

Metric          Value
-----
Validation Loss    0.4947
Validation Accuracy 0.9669
Validation Precision 0.9669
Validation Recall   0.9669
Validation F1-Score 0.9669
=====
```

Hình 5.5: Đánh giá Tổng hợp (MobileNetV2 - Tập Validation)

Phân tích Độ chính xác và Hàm mất mát

Độ chính xác (Validation Accuracy): Mô hình MobileNetV2 đạt 96.69% độ chính xác trên tập Validation. Đây là một kết quả cực kỳ ấn tượng, thậm chí còn cao hơn so với độ chính xác Validation cao nhất của ResNet50 (khoảng 94.7% ở Giai đoạn 1).

Hàm mất mát (Validation Loss): Giá trị 0.4947 là tương đối thấp, cho thấy mô hình đã học được các đặc trưng tốt và ổn định.

Đánh giá Hiệu suất Cân bằng (Precision, Recall, F1-Score)

Điểm nổi bật của kết quả này là sự đồng nhất hoàn hảo giữa các chỉ số:

Validation Precision: 0.9669

Validation Recall: 0.9669

Validation F1-Score: 0.9669

Ý nghĩa: Sự bằng nhau của các chỉ số này ($P = R = F1$) cho thấy mô hình MobileNetV2 đã đạt được sự cân bằng gần như tuyệt đối giữa khả năng tránh báo động giả (Precision) và khả năng phát hiện bệnh (Recall) trên tập

validation. Điều này xảy ra khi mô hình có tỷ lệ False Positive và False Negative rất thấp và gần bằng nhau.

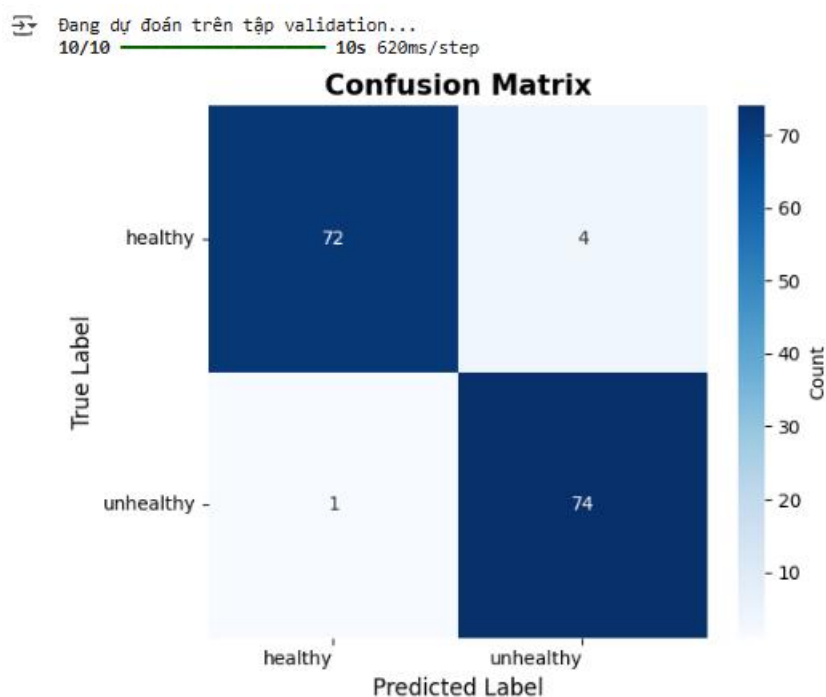
BÁO CÁO PHÂN LOẠI CHI TIẾT				
	precision	recall	f1-score	support
healthy	0.9863	0.9474	0.9664	76
unhealthy	0.9487	0.9867	0.9673	75
accuracy			0.9669	151
macro avg	0.9675	0.9670	0.9669	151
weighted avg	0.9676	0.9669	0.9669	151

Hình 5.6: Báo cáo phân loại chi tiết

Kết quả này là báo cáo phân loại chi tiết của mô hình MobileNetV2 trên tập Validation (151 mẫu).

Độ chính xác Tổng thể (Accuracy): Mô hình đạt 96.69% độ chính xác. Đây là một kết quả xuất sắc trên tập Validation.

F1-Score Cân bằng: F1-Score tổng hợp (Weighted Avg) đạt 0.9669. Chỉ số này khẳng định hiệu suất đồng đều của mô hình trên cả hai lớp.



Hình 5.7: Ma trận nhầm lẫn

Các kết quả chính xác (TP và TN)

True Positive (TP = 74): Mô hình đã dự đoán đúng 74 cây rau bị bệnh là 'unhealthy'.

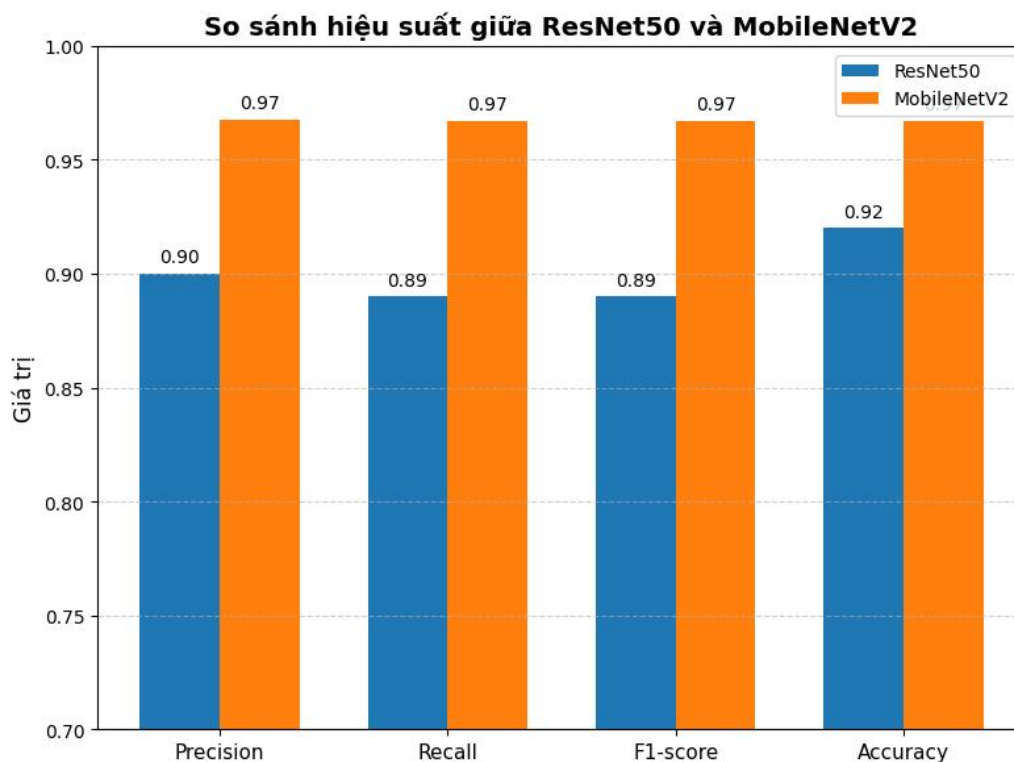
True Negative (TN = 72): Mô hình đã dự đoán đúng 72 cây rau khỏe mạnh là 'healthy'.

Các Lỗi Phân loại (FP và FN)

False Positive (FP = 4): Lỗi "Báo động giả". Chỉ 4 cây khỏe mạnh bị dự đoán nhầm thành 'unhealthy'. Đây là một lỗi rất thấp.

False Negative (FN = 1): Lỗi "Bỏ sót". Chỉ 1 cây bị bệnh bị dự đoán nhầm là 'healthy'. Tỷ lệ bỏ sót này cực kỳ thấp, ngang bằng với ResNet50.

5.3. Đánh giá RestNet50 và MobileNetV2



Hình 5.8: Biểu đồ so sánh hiệu suất 2 mô hình

Nhận xét 2 mô hình:

MobileNetV2 vượt trội hơn ResNet50 trên tất cả các chỉ số đánh giá.

Về Độ chính xác (Accuracy): MobileNetV2 đạt 97%, cao hơn đáng kể so với 90% của ResNet50.

Về các chỉ số chi tiết: Cả Precision, Recall, và F1-score của MobileNetV2 đều đạt mức 0.97 rất cao và cân bằng, trong khi ResNet50 chỉ đạt từ 0.89 đến 0.90.

Kết luận: Dựa trên biểu đồ so sánh này, MobileNetV2 là mô hình vượt trội hơn rõ rệt và cho kết quả tốt hơn hẳn.

CHƯƠNG 6. TRIỂN KHAI ỨNG DỤNG WEB

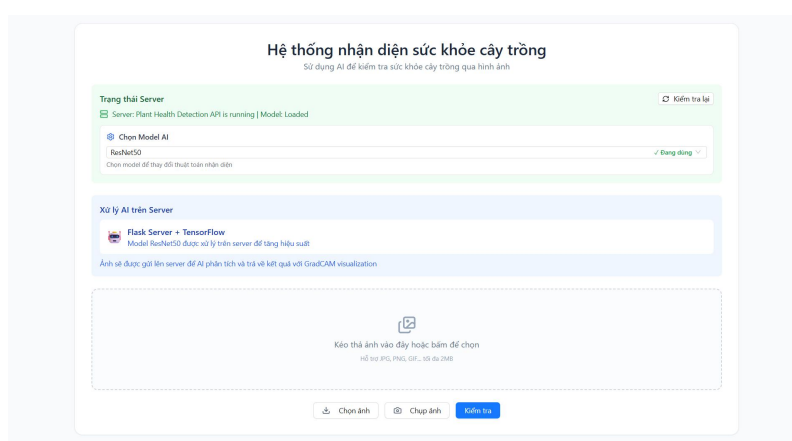
Link : <https://laptrinhvienweb.netlify.app/>

Giai đoạn cuối cùng là biến mô hình đã huấn luyện thành một ứng dụng web dễ sử dụng, cho phép người dùng tải lên hình ảnh và nhận dự đoán sức khỏe rau ngay lập tức.

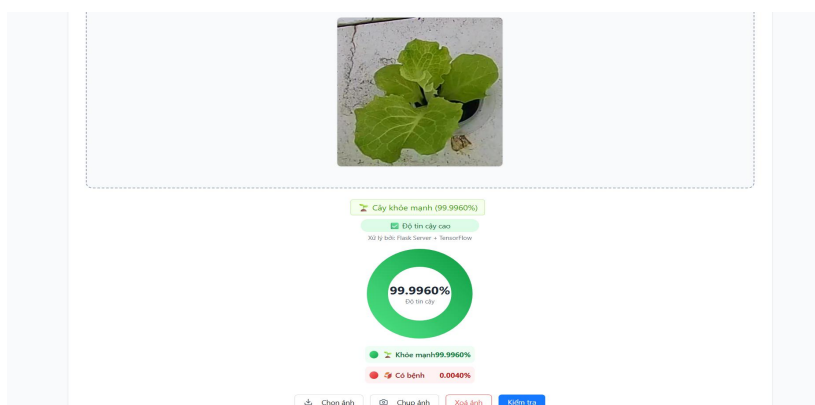
Frontend (Client): Xây dựng bằng ReactJS, chịu trách nhiệm cho giao diện người dùng và gửi ảnh.

Backend (Server/API): Xây dựng bằng Flask, chịu trách nhiệm tải mô hình, xử lý tính toán Deep Learning, và Grad-CAM.

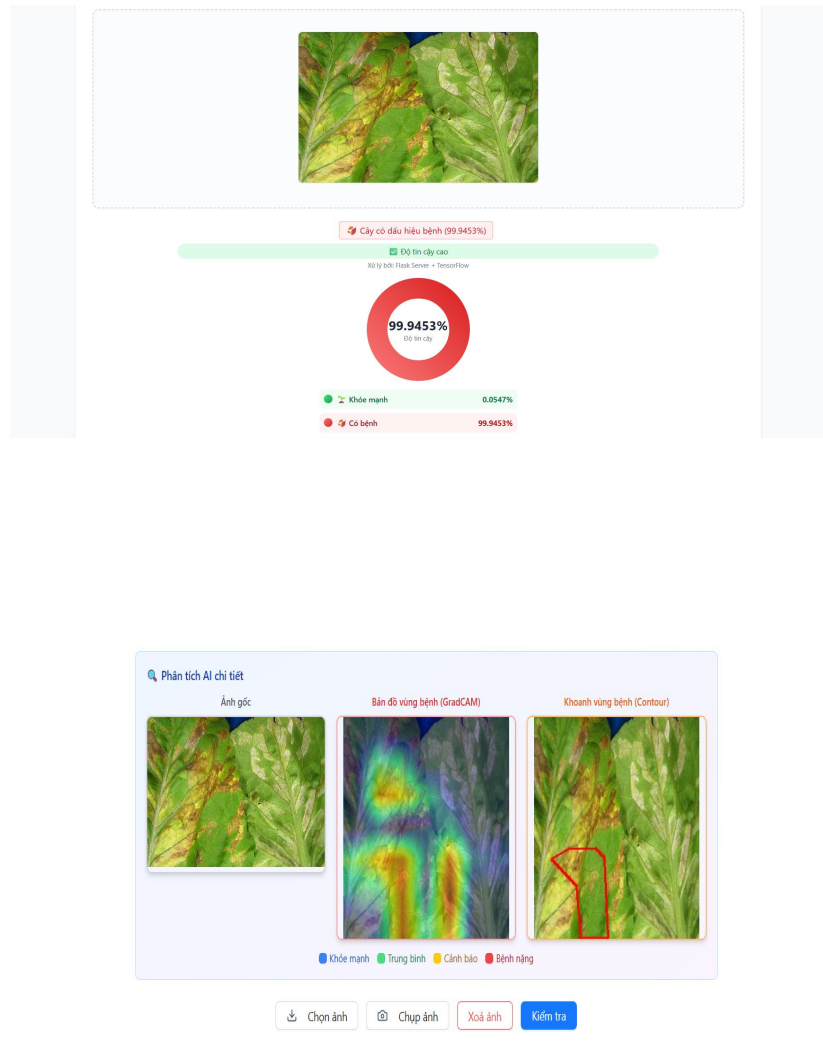
Giao diện ban đầu



Giao diện khi dự đoán ảnh 'Healthy'



Giao diện khi dự đoán ảnh ‘Unhealthy’



CHƯƠNG 7. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Đồ án đã hoàn thành tất cả các mục tiêu đã đặt ra ban đầu:

Xây dựng thành công mô hình Deep Learning ResNet50, MobileNetV2 bằng kỹ thuật Transfer Learning trên nền tảng Google Colab.

Ứng dụng web đã được triển khai bằng ReactJS, cung cấp một giao diện trực quan và dễ sử dụng cho người dùng cuối.

Hạn chế của Đồ án

Mặc dù đạt được kết quả tích cực, đồ án vẫn còn một số hạn chế:

Tính đa dạng của dữ liệu: Dữ liệu chỉ tập trung vào rau xà lách và được thu thập từ internet/tổng hợp. Hiệu suất mô hình có thể giảm khi đối mặt với các điều kiện ánh sáng, góc chụp, hoặc nền khác biệt trong môi trường thực tế (nông trại).

Phân loại nhị phân: Mô hình chỉ phân biệt được rau 'Healthy' và rau 'Unhealthy' tổng thể, chưa thể phân biệt các loại bệnh cụ thể (ví dụ: Downy Mildew, Tip Burn, thiếu chất).

Hướng phát triển tương lai

Để nâng cao giá trị thực tiễn và mở rộng phạm vi ứng dụng của đồ án "Ứng dụng học sâu dự đoán ảnh sức khỏe rau xanh", các hướng nghiên cứu và phát triển trong tương lai có thể tập trung vào những điểm sau:

Mở rộng phân loại đa lớp (Multi-class Classification):

Hiện tại mô hình chỉ phân loại nhị phân (Healthy/Unhealthy). Hướng phát triển tiếp theo là thu thập và gán nhãn chi tiết hơn (ví dụ: Downy Mildew, Tip Burn, thiếu Kali, thiếu đạm).

Việc này đòi hỏi mô hình không chỉ nói "Có vấn đề" mà còn chẩn đoán chính xác "Vấn đề là gì", giúp người dùng đưa ra phương pháp can thiệp chuyên biệt và hiệu quả hơn.

Tích hợp hệ thống giám sát thời gian thực (Real-time Monitoring):

Chuyển từ ứng dụng dự đoán ảnh tĩnh sang xử lý luồng video hoặc tích hợp với các hệ thống camera giám sát IoT.

Ứng dụng có thể được triển khai trên các thiết bị tại nông trại (ví dụ: máy tính nhúng) để liên tục theo dõi sức khỏe cây trồng và đưa ra cảnh báo ngay khi phát hiện bệnh lây lan, thay vì chỉ chẩn đoán thủ công từng lá rau.

TÀI LIỆU THAM KHẢO

1. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778.
2. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 4510–4520.
3. Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 618–626.
4. Abadi, M., et al. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Mở truy cập tại: <https://www.tensorflow.org/>
5. Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.
6. React Documentation. (2024). React – A JavaScript library for building user interfaces. Mở truy cập tại: <https://react.dev/>.
7. Stha, A. (2024). Healthy vs. Stressed Lettuce Image Dataset. Kaggle. Mở truy cập tại: <https://www.kaggle.com/datasets/ashimstha/lettuce-health-compiled-dataset>