

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



Bộ môn: Khai phá dữ liệu

Đề tài: Nhận diện tình trạng sức khỏe của cây xà lách bằng mô hình học sâu

Sinh viên thực hiện:	Lê Hữu Phát-2251068228 Nguyễn Chí Nguyên-2251068219 Nguyễn Anh Khoa-2251068202
Giáo viên hướng dẫn:	Ths. Vũ Thị Hạnh

TP. HCM, 14/10/2025

Mục Lục

Mục lục hình ảnh	3
CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI	1
1.1 Giới thiệu	1
1.2 Mục Tiêu	1
1.3 Phương Pháp	1
CHƯƠNG 2: PHÂN TÍCH DỮ LIỆU VÀ MỤC TIÊU NGHIÊN CỨU	2
2.1 Thông tin tổng quan	2
2.2 Phân tích dữ liệu:	2
2.3 Tiền xử lý đã có	2
2.4 Phân loại đặc trưng của lớp healthy và unhealthy	2
CHƯƠNG 3: MỤC TIÊU NGHIÊN CỨU VÀ CÔNG CỤ THỰC HIỆN	3
3.1 Nhiệm vụ chính phân loại ảnh cây xà lách thành healthy và unhealthy	3
3.2 Công cụ, ngôn ngữ, thư viện hỗ trợ và thuật toán dùng để dự	3
3.3 Mục tiêu là tự động nhận diện cây bị bệnh hoặc không đủ dinh dưỡng để hỗ trợ phát hiện sớm	4
CHƯƠNG 4: QUY TRÌNH XÂY DỰNG VÀ ĐÁNH GIÁ MÔ HÌNH	5
4.1 Tải dữ liệu (Data Loading)	5
4.2 Tiền xử lý và chia tập dữ liệu	5
4.3 Xây dựng mô hình học sâu (CNN)	6
4.4 Huấn luyện mô hình	8
4.5 Đánh giá mô hình	12
4.6 Kiểm thử mô hình với ảnh thực tế	15
4.8 Đánh giá và Trực quan hóa Kết quả (Mô hình MobileNetV2)	20
4.9 Phân tích Chi tiết và Ma trận Nhầm lẫn (Mô hình MobileNetV2)	21
4.10 Kiểm tra Thực tế với Mô hình Cải tiến	23
4.11 Khởi tạo mô hình ResNet	25
4.12 Sử dụng mô hình ResNet cho việc phân loại	26
4.13 Sử dụng kiến trúc ResNet cho việc phân loại	27
4.14 Thiết lập hàm xuất các thông số kết quả	28
4.15 Thiết lập Early Stopping	28
4.16 Hiển thị ảnh ngẫu nhiên với Label để test bộ dữ liệu	30
4.17 Batch Visualantion	31
4.18 Huấn luyện mô hình ResNet	32
4.19 Vẽ biểu đồ Accuracy với Epochs & Loss với Epochs	34
4.20 Vẽ Confusion Matrix	35
4.21 Kiểm tra với dữ liệu thực tế	36
4.22 Lưu mô hình ResNet	36
4.23 Cải tiến mô hình với EfficientNet	38
4.24 Đánh giá kết quả mô hình EfficientNet	39
4.25 Đánh giá kết quả Ma trận Nhầm lẫn của mô hình EfficientNet	39
4.26 Kiểm tra Mô hình EfficientNet với ảnh thực tế	40
CHƯƠNG 5: SO SÁNH VÀ LỰA CHỌN MÔ HÌNH TỐI UỐU	42
5.1 Mục đích	42
5.2 Bảng so sánh các mô hình	42
5.3 Kết luận và lựa chọn mô hình	43
CHƯƠNG 6: TRIỂN KHAI MÔ HÌNH TRÊN GIAO DIỆN WEB VỚI HUGGING FACE SPACES	45
6.1 Triển khai mô hình trên giao diện web	45
6.2 Đánh giá kết quả mô hình sau khi triển khai trên web	47

Mục lục hình ảnh

Hình 1: Giải nén tệp plant-health.zip.....	5
Hình 2: Tiền xử lý ảnh và chia tập dữ liệu Train/Validation set.....	6
Hình 3: Xây dựng mô hình CNN.....	8
Hình 4: Thiết lập cho quá trình huấn luyện	9
Hình 5: Quy trình huấn luyện và đánh giá mô hình LettuceCNN 1.....	10
Hình 6: Quy trình huấn luyện và đánh giá mô hình LettuceCNN 2.....	11
Hình 7: Đánh giá và trực quan hóa kết quả.....	12
Hình 8: Biểu đồ độ chính xác và độ mắng mát của mô hình LettuceCNN	12
Hình 9: Code Ma trận nhầm lẫn.....	13
Hình 10: Ma trận nhầm và các chỉ số đánh giá.....	14
Hình 11: Mô hình với ảnh thực tế.....	16
Hình 12: Ảnh thực tế và kết quả cho mô hình	17
Hình 13 : Cải tiến mô hình với MobileNetV2 (Transfer Learning)	18
Hình 14: Cải tiến mô hình với MobileNetV2 (Transfer Learning).....	19
Hình 15 : Kết quả huấn luyện mô hình MobileNetV2 (Transfer Learning).....	19
Hình 16: Code Trực quan hóa Mô hình MobileNetV2	20
Hình 17: Cải tiến mô hình với MobileNetV2	21
Hình 18: Code ma trận nhầm lẫn của (Mô hình MobileNetV2)	22
Hình 19: Ma trận nhầm lẫn của (Mô hình MobileNetV2)	23
Hình 20: Kiểm tra Thực tế với Mô hình Cải tiến.....	24
Hình 21: Khởi tạo ResNet	25
Hình 22: Sử dụng mô hình ResNet cho việc phân loại	26
Hình 23 : Sử dụng kiến trúc ResNet cho việc phân loại	27
Hình 24: Thiết lập hàm xuất các thông số kết quả	28
Hình 25 : Thiết lập Early Stopping	29
Hình 26: Hiển thị ảnh ngẫu nhiên với Label để test bộ dữ liệu	30
Hình 27: Batch Visualiantion	31
Hình 28: Huấn luyện mô hình ResNet.....	32
Hình 29: Cấu hình các tham số huấn luyện mô hình	33
Hình 30: Biểu đồ Accuracy với Epochs & Loss với Epochs	34
Hình 31: Confusion Matrix	35
Hình 32: Kiểm tra với dữ liệu thực tế	36
Hình 33: Lưu mô hình ResNet.....	36
Hình 34: Dự đoán tình trạng cây trồng bằng EfficientNet (PyTorch)	38
Hình 35: Đánh giá kết quả mô hình EfficientNet	39
Hình 36: Đánh giá kết quả Ma trận Nhầm lẫn	39
Hình 37: Kiểm thử mô hình EfficientNet_B0 trên ảnh thực tế	40
Hình 38: Kết quả của mô hình EfficientNet_B0 trên ảnh thực tế	41
Hình 39: Danh sách các thư viện cần thiết cho ứng dụng.....	45
Hình 40: Tạo giao diện Web với Hugging Face Spaces	46
Hình 41: Giao diện web ứng dụng Lettuce Knight và kết quả dự đoán	47

CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

1.1 Giới thiệu

- Trong những năm gần đây, nông nghiệp thông minh (Smart Agriculture) đã trở thành một lĩnh vực nghiên cứu quan trọng nhằm tối ưu hóa năng suất cây trồng và giảm thiểu tổn thất do sâu bệnh. Việc phát hiện và phân loại các loại bệnh trên cây trồng bằng hình ảnh đóng vai trò quan trọng trong việc quản lý và chăm sóc cây trồng hiệu quả. Rau xà lách (Lettuce) là một trong những loại rau phổ biến, đặc biệt trong sản xuất nông nghiệp công nghệ cao. Tuy nhiên, Rau xà lách rất dễ bị ảnh hưởng bởi các loại bệnh khác nhau, dẫn đến giảm năng suất và chất lượng sản phẩm. Do đó, việc phát triển một hệ thống tự động nhận dạng và phân loại bệnh trên Rau xà lách bằng trí tuệ nhân tạo (AI) và học sâu (Deep Learning) là cần thiết. Trong đề tài này, dữ liệu được sử dụng là Plant Healthy – Lettuce Classification Dataset lấy từ Kaggle, kết hợp với thuật toán học sâu khác để phân loại các trạng thái sức khỏe của Rau xà lách dựa trên hình ảnh.

1.2 Mục Tiêu

- Mục tiêu chính của đề tài bao gồm:
- + Thu thập và tiền xử lý dữ liệu hình ảnh Rau xà lách từ bộ dữ liệu Plant Healthy – Lettuce Classification.
- + Xây dựng các mô hình học sâu để phân loại các loại bệnh trên rau xà lách.
- + Đánh giá hiệu suất của mô hình qua các chỉ số như Accuracy, Precision, Recall và F1-Score.
- + Cung cấp cơ sở dữ liệu và giải pháp hỗ trợ nông dân trong việc nhận dạng bệnh trên Rau xà lách một cách nhanh chóng và chính xác.

1.3 Phương Pháp

- **Dữ liệu:** Sử dụng bộ Plant Healthy – Lettuce Classification từ Kaggle, chia thành tập huấn luyện, xác thực và kiểm tra. Ảnh được tiền xử lý gồm resize, chuẩn hóa và tăng cường dữ liệu.
- **Mô hình:** Sử dụng kiến trúc CNN, MobileNetV2, ResNet, EfficientNet, để trích xuất đặc trưng và phân loại ảnh Rau xà lách theo trạng thái sức khỏe và lựa chọn mô hình nào phù hợp up lên web.

CHƯƠNG 2: PHÂN TÍCH DỮ LIỆU VÀ MỤC TIÊU NGHIÊN CỨU

2.1 Thông tin tổng quan

- Tên tập dữ liệu: Plant Healthy – Lettuce Classification được lấy từ Kaggle.
- Tập dữ liệu được chia thành 2 class:
 - + Class 1: Healthy–Khỏe mạnh (đủ dinh dưỡng và không có bệnh tật).
 - + Class 2: Unhealthy–Không lành mạnh (tổng hợp nhiều bệnh tật và thiếu dinh dưỡng do môi trường, nhiệt độ, vi khuẩn,...)

2.2 Phân tích dữ liệu:

- Đây là dữ liệu về hình ảnh có hai loại là healthy (326 tấm ảnh) và unhealthy (381 tấm ảnh), định dạng ảnh .jpg, kích thước chưa được chuẩn hóa về cùng một kích thước (thường là 256x256 hoặc 224x224 pixel).

2.3 Tiết xuỷ lý đã có

- + Ảnh được lọc nhiễu và cắt viền (cropping) để tập trung vào lá xà lách
- + Dữ liệu được chia theo thư mục (plant-health/healthy và plant-health/unhealthy).
- + Đã chuẩn hóa màu sắc và ánh sáng cơ bản để giảm sai lệch giữa ảnh chụp ngoài trời và ảnh trong nhà.
- + Tập dữ liệu được cân bằng lại (balanced) - số lượng ảnh giữa các lớp gần tương đương.

2.4 Phân loại đặc trưng của lớp healthy và unhealthy

- Cây healthy (khỏe mạnh) thường có các đặc điểm:
 - + Màu xanh lá tươi, đồng đều nhau.
 - + Bè mặt lá nhẵn, không đốm hoặc nứt.
 - + Không có dấu hiệu héo hoặc cháy lá.
 - + Biên lá rõ ràng, ít nhiễu.
- Cây unhealthy (bị bệnh) thường có các đặc trưng sau:
 - + Màu lá không đồng đều xuất hiện vệt vàng, nâu hoặc đen.
 - + Có đốm tròn, lỗ nhỏ hoặc mốc trắng (diễn hình cho bệnh nấm mốc).
 - + Lá xoăn hoặc biến dạng.
 - + Độ tương phản (contrast) cao hơn, do vùng bệnh phản sáng khác vùng khỏe.

CHƯƠNG 3: MỤC TIÊU NGHIÊN CỨU VÀ CÔNG CỤ THỰC HIỆN

3.1 Nhiệm vụ chính phân loại ảnh cây xà lách thành healthy và unhealthy

- Thực hiện phân loại hình ảnh cây xà lách (Lettuce) thành hai nhóm:
 - + Healthy (khỏe mạnh).
 - + Unhealthy (bị bệnh hoặc thiếu dinh dưỡng).
- Mục tiêu của nhiệm vụ là xây dựng mô hình phân loại ảnh tự động, giúp nhận biết tình trạng sức khỏe của cây dựa trên các đặc trưng hình ảnh như:
 - + Màu sắc lá (xanh tươi, vàng úa, nâu sạm, v.v.).
 - + Hình dạng và kết cấu bề mặt lá.
 - + Các dấu hiệu bệnh lý hoặc sâu hại (đốm, vết cháy, thủng lá,...).

3.2 Công cụ, ngôn ngữ, thư viện hỗ trợ và thuật toán dùng để dự

- Công cụ: Colab trên Google Drive.
- Ngôn ngữ: Python phiên bản 3.x
- Thư viện cần dùng:

Loại	Thư viện hỗ trợ	Vai trò
Xử lý dữ liệu	Numpy, Pandas	Quản lý, thao tác và phân tích dữ liệu.
Xử lý ảnh	OpenCV, Scikit-Image, Pillow	Đọc, tiền xử lý, và tăng cường dữ liệu hình ảnh
Học sâu	TensorFlow/Keras, PyTorch	Xây dựng và huấn luyện mô hình CNN
Trực quan hóa	Matplotlib, Seaborn	Biểu đồ thống kê, confusion matrix, biểu diễn kết quả
Tối ưu & đánh giá	Scikit-Learn	Chia tập dữ liệu, tính các chỉ số đánh giá (accuracy, precision, recall, F1)

- Thuật toán dự đoán: Mô hình học sâu (Deep Learning) - Mạnh nơ-ron tích chập:
 - + Mô hình huấn luyện để tự động học các đặc trưng (texture, màu, hình dạng) từ ảnh xà lách.
 - + Custom CNN (tự xây dựng các lớp Conv2D, MaxPooling, Dense, Dropout).

3.3 Mục tiêu là tự động nhận diện cây bị bệnh hoặc không đủ dinh dưỡng để hỗ trợ phát hiện sớm

- Xây dựng một hệ thống tự động nhận diện cây xà lách bị bệnh hoặc thiếu dinh dưỡng, từ đó:
 - + Phát hiện sớm các vấn đề về sức khỏe cây trồng.
 - + Hỗ trợ nông dân trong việc theo dõi, quản lý vườn rau bằng công nghệ.
 - + Giảm thiệt hại do bệnh hại và tối ưu năng suất canh tác.

CHƯƠNG 4: QUY TRÌNH XÂY DỰNG VÀ ĐÁNH GIÁ MÔ HÌNH

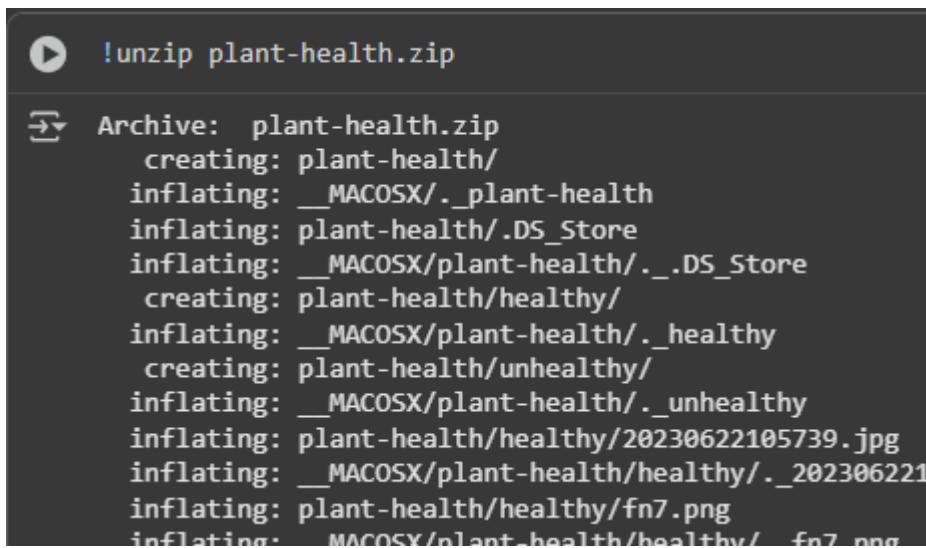
- Quy trình thực hiện tổng quát gồm:

tải dữ liệu → tiền xử lý → huấn luyện → đánh giá → đưa ra đánh giá thực tế

4.1 Tải dữ liệu (Data Loading)

- Nguồn dữ liệu: Tập dữ liệu Plant Healthy – Lettuce Classification được tải từ Kaggle và lưu trên Google Drive dưới dạng tệp nén **plant-health.zip**

- Giải nén : Sử dụng Google Colab, tiến hành giải nén dữ liệu vào thư mục làm việc.



```
!unzip plant-health.zip
Archive:  plant-health.zip
  creating: plant-health/
  inflating: __MACOSX/.__plant-health
  inflating: plant-health/.DS_Store
  inflating: __MACOSX/plant-health/__.DS_Store
  creating: plant-health/healthy/
  inflating: __MACOSX/plant-health/.__healthy
  creating: plant-health/unhealthy/
  inflating: __MACOSX/plant-health/.__unhealthy
  inflating: plant-health/healthy/20230622105739.jpg
  inflating: __MACOSX/plant-health/healthy/._202306221
  inflating: plant-health/healthy/fn7.png
  inflating: __MACOSX/plant-health/healthy/.fn7.png
```

Hình 1: Giải nén tệp plant-health.zip

4.2 Tiền xử lý và chia tập dữ liệu

- Trong quá trình huấn luyện mô hình học sâu, việc tiền xử lý ảnh đầu vào là rất quan trọng. Dưới đây là các bước biến đổi được áp dụng trong dự án:

- + **transforms.Compose**: Dùng để kết hợp nhiều phép biến đổi (transformations) lại với nhau thành một chuỗi xử lý thống nhất.
- + **transforms.Resize((128, 128))**: Thay đổi kích thước tất cả các ảnh về cùng một kích thước 128x128 pixel, giúp mô hình nhận diện đầu vào nhất quán.
- + **transforms.ToTensor()**: Chuyển đổi ảnh từ định dạng PIL hoặc NumPy (có giá trị pixel từ 0–255) sang dạng PyTorch Tensor với giá trị chuẩn hóa từ 0.0 đến 1.0.
- + **transforms.Normalize(mean, std)**: Chuẩn hóa dữ liệu bằng cách trừ đi giá trị trung bình (mean) và chia cho độ lệch chuẩn (std) của từng kênh màu RGB. Phép chuẩn hóa này giúp mô hình học nhanh hơn và ổn định hơn trong quá trình huấn luyện.
- Sau khi xử lý ảnh, dữ liệu được chia thành hai tập:
 - + **torch.utils.data.random_split**: Dùng để chia bộ dữ liệu ban đầu thành hai phần — 80% cho huấn luyện và 20% cho kiểm thử — một cách ngẫu nhiên.

+ **DataLoader**: Đóng vai trò tạo các batch (lô dữ liệu) từ tập huấn luyện và tập kiểm thử. Việc chia dữ liệu thành batch giúp quá trình huấn luyện tiết kiệm bộ nhớ và tối ưu hóa tốc độ tính toán.

```

DATA_DIR = '/content/plant-health'
# Định nghĩa các phép biến đổi cho ảnh
data_transforms = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Tải toàn bộ dữ liệu bằng ImageFolder
full_dataset = ImageFolder(root=DATA_DIR, transform=data_transforms)
class_names = full_dataset.classes
print(f"Các lớp được tìm thấy: {class_names}")
print(f"Tổng số ảnh: {len(full_dataset)}")

# Chia dữ liệu: 80% cho huấn luyện, 20% cho kiểm thử
train_size = int(0.8 * len(full_dataset))
val_size = len(full_dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(full_dataset, [train_size, val_size])

print(f"Số lượng ảnh huấn luyện: {len(train_dataset)}")
print(f"Số lượng ảnh kiểm thử: {len(val_dataset)}")

# Tạo DataLoaders
BATCH_SIZE = 32
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)

```

➡ Các lớp được tìm thấy: ['healthy', 'unhealthy']
Tổng số ảnh: 707
Số lượng ảnh huấn luyện: 565
Số lượng ảnh kiểm thử: 142

Hình 2: Tiền xử lý ảnh và chia tập dữ liệu Train/Validation set

4.3 Xây dựng mô hình học sâu (CNN)

- Mô hình được xây dựng dựa trên lớp nn.Module của PyTorch, đây là lớp cơ sở cho tất cả các mô hình mạng nơ-ron. Cấu trúc mô hình gồm ba khối tích chập (Convolution Blocks) và hai lớp kết nối đầy đủ (Fully Connected Layers). Cụ thể:
- + **nn.Module**: Lớp cơ sở của tất cả các mô hình trong PyTorch. Tất cả các lớp tự định nghĩa cần kế thừa từ lớp này.
- + **super().__init__()**: Hàm khởi tạo của lớp cha, giúp kế thừa đầy đủ các thuộc tính và phương thức từ nn.Module.
- + **nn.Conv2d**: Lớp tích chập 2 chiều, có nhiệm vụ trích xuất đặc trưng (feature) từ ảnh đầu vào thông qua các bộ lọc (filters).
- + **nn.ReLU()**: Hàm kích hoạt phi tuyến, giúp mạng học được các đặc trưng phức tạp hơn.

- + **nn.MaxPool2d**: Lớp gộp cực đại (max pooling) giúp giảm kích thước không gian của đặc trưng, đồng thời giữ lại các đặc trưng quan trọng nhất.
- + **nn.Flatten**: Chuyển dữ liệu 2D thành dạng vector 1D để đưa vào các lớp kết nối đầy đủ.
- + **nn.Linear**: Lớp kết nối đầy đủ (fully connected layer), dùng để ánh xạ đặc trưng đã học thành đầu ra.
- + **nn.Dropout**: Lớp dùng để loại bỏ ngẫu nhiên một số neuron trong quá trình huấn luyện, giúp giảm hiện tượng quá khớp (overfitting).
- + **forward(self, x)**: Hàm định nghĩa luồng dữ liệu đi qua các lớp của mạng — mô tả cách mà đầu vào được xử lý để tạo ra đầu ra cuối cùng.
 - Mô hình LettuceCNN gồm:
 - + 3 khối tích chập (Conv + ReLU + MaxPool) giúp trích xuất đặc trưng của ảnh.
 - + 2 lớp fully connected để thực hiện phân loại nhị phân (healthy/unhealthy).
 - + Kích thước ảnh đầu vào là 128×128 pixel, sau 3 lần pooling giảm còn 16×16 .

```

▶ class LettuceCNN(nn.Module):
    def __init__(self):
        super(LettuceCNN, self).__init__()
        # Block tích chập 1
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Block tích chập 2
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Block tích chập 3
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.relu3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Lớp kết nối đầy đủ
        # Kích thước ảnh sau 3 lần pooling: 128 -> 64 -> 32 -> 16. Vậy kích thước là 128 * 16 * 16
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(128 * 16 * 16, 512)
        self.relu4 = nn.ReLU()
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(512, 1) # Đầu ra 1 giá trị duy nhất cho phân loại nhị phân

    def forward(self, x):
        x = self.pool1(self.relu1(self.conv1(x)))
        x = self.pool2(self.relu2(self.conv2(x)))
        x = self.pool3(self.relu3(self.conv3(x)))
        x = self.flatten(x)
        x = self.relu4(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

# Khởi tạo mô hình và in ra kiến trúc
model = LettuceCNN()
print(model)

```

Hình 3: Xây dựng mô hình CNN

4.4 Huấn luyện mô hình

- Cấu hình huấn luyện mô hình:
 - + **device:** Kiểm tra và chọn thiết bị huấn luyện (GPU nếu có, nếu không thì CPU).
Huấn luyện trên GPU giúp tăng tốc độ xử lý.
 - + **model.to(device):** Chuyển mô hình lên thiết bị đã chọn.
 - + **nn.BCEWithLogitsLoss:** Hàm mất mát cho bài toán phân loại nhị phân, kết hợp Sigmoid và Binary Cross Entropy để tăng độ ổn định.
 - + **optim.Adam:** Thuật toán tối ưu hóa phô biến, giúp mô hình hội tụ nhanh và ổn định (với tốc độ học lr = 0.001).- Sau mỗi epoch, mô hình được đánh giá trên tập validation:
 - + **NUM_EPOCHS = 15:** Huấn luyện mô hình trong 15 vòng lặp (epoch).

```
# Chọn thiết bị (GPU nếu có)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Sử dụng thiết bị: {device}")

# Đưa mô hình lên thiết bị
model.to(device)

# Định nghĩa hàm mất mát và thuật toán tối ưu hóa
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Số epochs
NUM_EPOCHS = 15
```

Hình 4: Thiết lập cho quá trình huấn luyện

- Quy trình huấn luyện và đánh giá mô hình:
 - + **model.train()**: Bật chế độ huấn luyện, cho phép các lớp như Dropout hoạt động.
 - + **optimizer.zero_grad()**: Xóa các gradient cũ trước khi tính toán mới.
 - + **loss.backward()**: Tính gradient của hàm mất mát đối với các tham số mô hình.
 - + **optimizer.step()**: Cập nhật trọng số mô hình dựa trên gradient đã tính.
 - + **model.eval()**: Chuyển sang chế độ đánh giá, tắt các lớp như Dropout để kết quả ổn định.
 - + **with torch.no_grad()**: Tắt tính toán gradient trong pha đánh giá để tiết kiệm bộ nhớ và tăng tốc xử lý.

```

# Lưu lại lịch sử huấn luyện
history = {
    'train_loss': [], 'train_acc': [],
    'val_loss': [], 'val_acc': []
}

for epoch in range(NUM_EPOCHS):
    # --- PHA HUẤN LUYỆN ---
    model.train()
    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    for inputs, labels in train_loader:
        # Chuyển dữ liệu lên device
        inputs, labels = inputs.to(device), labels.to(device)

        # Vì đầu ra là 1 nơ-ron, nhãn cũng cần có shape tương ứng
        labels = labels.float().unsqueeze(1)

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Backward pass và tối ưu hóa
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Thống kê
        running_loss += loss.item() * inputs.size(0)
        preds = torch.sigmoid(outputs) > 0.5
        correct_predictions += (preds == labels).sum().item()
        total_samples += labels.size(0)

    epoch_train_loss = running_loss / total_samples
    epoch_train_acc = correct_predictions / total_samples
    history['train_loss'].append(epoch_train_loss)
    history['train_acc'].append(epoch_train_acc)

```

Hình 5: Quy trình huấn luyện và đánh giá mô hình LettuceCNN 1

```

# --- PHÁ ĐÁNH GIÁ ---
model.eval()
running_loss = 0.0
correct_predictions = 0
total_samples = 0

with torch.no_grad():
    for inputs, labels in val_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        labels = labels.float().unsqueeze(1)

        outputs = model(inputs)
        loss = criterion(outputs, labels)

        running_loss += loss.item() * inputs.size(0)
        preds = torch.sigmoid(outputs) > 0.5
        correct_predictions += (preds == labels).sum().item()
        total_samples += labels.size(0)

    epoch_val_loss = running_loss / total_samples
    epoch_val_acc = correct_predictions / total_samples
    history['val_loss'].append(epoch_val_loss)
    history['val_acc'].append(epoch_val_acc)

    print(f"Epoch {epoch+1}/{NUM_EPOCHS} | "
          f"Train Loss: {epoch_train_loss:.4f}, Train Acc: {epoch_train_acc:.4f} | "
          f"Val Loss: {epoch_val_loss:.4f}, Val Acc: {epoch_val_acc:.4f}")

print("Hoàn thành huấn luyện!")

```

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1/15	0.8134	0.5327	0.6662	0.6268
2/15	0.5729	0.7274	0.4778	0.7394
3/15	0.4282	0.8425	0.3770	0.8380
4/15	0.3105	0.8796	0.3159	0.8873
5/15	0.2291	0.9133	0.2793	0.8873
6/15	0.2029	0.9274	0.2941	0.8662
7/15	0.1529	0.9310	0.2425	0.9014
8/15	0.1136	0.9575	0.2713	0.8944
9/15	0.0961	0.9664	0.2255	0.9014
10/15	0.0682	0.9788	0.2739	0.9225
11/15	0.0515	0.9805	0.2160	0.9366
12/15	0.0334	0.9858	0.2792	0.9155
13/15	0.0132	1.0000	0.2946	0.9225
14/15	0.0086	0.9982	0.3060	0.9225
15/15	0.0053	1.0000	0.3874	0.9014

Hoàn thành huấn luyện!

Hình 6: Quy trình huấn luyện và đánh giá mô hình LettuceCNN 2

Nhận xét: Mô hình hội tụ nhanh, độ chính xác trên tập kiểm thử đạt khoảng 90%, cho thấy LettuceCNN có khả năng phân biệt tốt giữa hai lớp healthy và unhealthy. Tuy nhiên, Val Loss có xu hướng tăng nhẹ ở cuối quá trình huấn luyện, gợi ý khả năng overfitting nhẹ.

4.5 Đánh giá mô hình

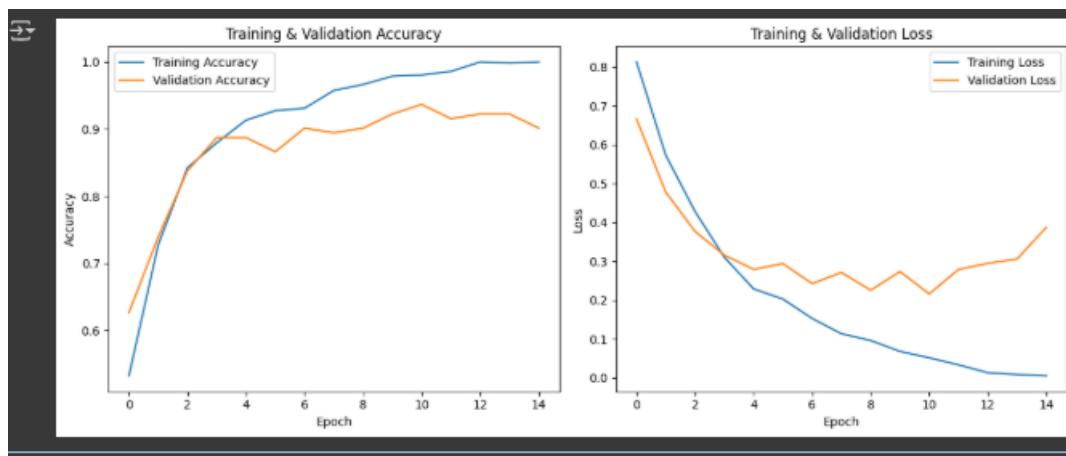
```
# Vẽ biểu đồ
plt.figure(figsize=(12, 5))

# Biểu đồ độ chính xác
plt.subplot(1, 2, 1)
plt.plot(history['train_acc'], label='Training Accuracy')
plt.plot(history['val_acc'], label='Validation Accuracy')
plt.title('Training & Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Biểu đồ hàm mất mát
plt.subplot(1, 2, 2)
plt.plot(history['train_loss'], label='Training Loss')
plt.plot(history['val_loss'], label='Validation Loss')
plt.title('Training & Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Hình 7: Đánh giá và trực quan hóa kết quả



Hình 8: Biểu đồ độ chính xác và độ mất mát của mô hình LettuceCNN

- Biểu đồ trên thể hiện sự thay đổi của độ chính xác (Accuracy) và độ mất mát (Loss) trên cả tập huấn luyện và kiểm thử qua 15 epoch:
 - + Độ chính xác huấn luyện tăng đều và đạt gần 100% sau 15 epoch.
 - + Độ chính xác kiểm thử đạt mức cao (~90%) và ổn định sau epoch thứ 5, cho thấy mô hình đã học được đặc trưng phân loại tốt.
 - + Độ mất mát huấn luyện giảm liên tục, trong khi độ mất mát kiểm thử giảm đến khoảng epoch thứ 5 rồi dao động nhẹ — dấu hiệu của overfitting nhẹ.
- Kết luận:** Mô hình LettuceCNN đạt hiệu suất cao với độ chính xác trên tập kiểm thử khoảng 90%. Tuy nhiên, để cải thiện khả năng tổng quát hóa, có thể áp dụng thêm regularization, data augmentation, hoặc early stopping trong huấn luyện.

```

▶ # Lấy dự đoán và nhãn thật từ tập validation
all_preds = []
all_labels = []

model.eval()
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs = inputs.to(device)
        outputs = model(inputs)
        preds = (torch.sigmoid(outputs) > 0.5).cpu().numpy()
        all_preds.extend(preds)
        all_labels.extend([labels.numpy()])

all_preds = np.array(all_preds).flatten()
all_labels = np.array(all_labels).flatten()

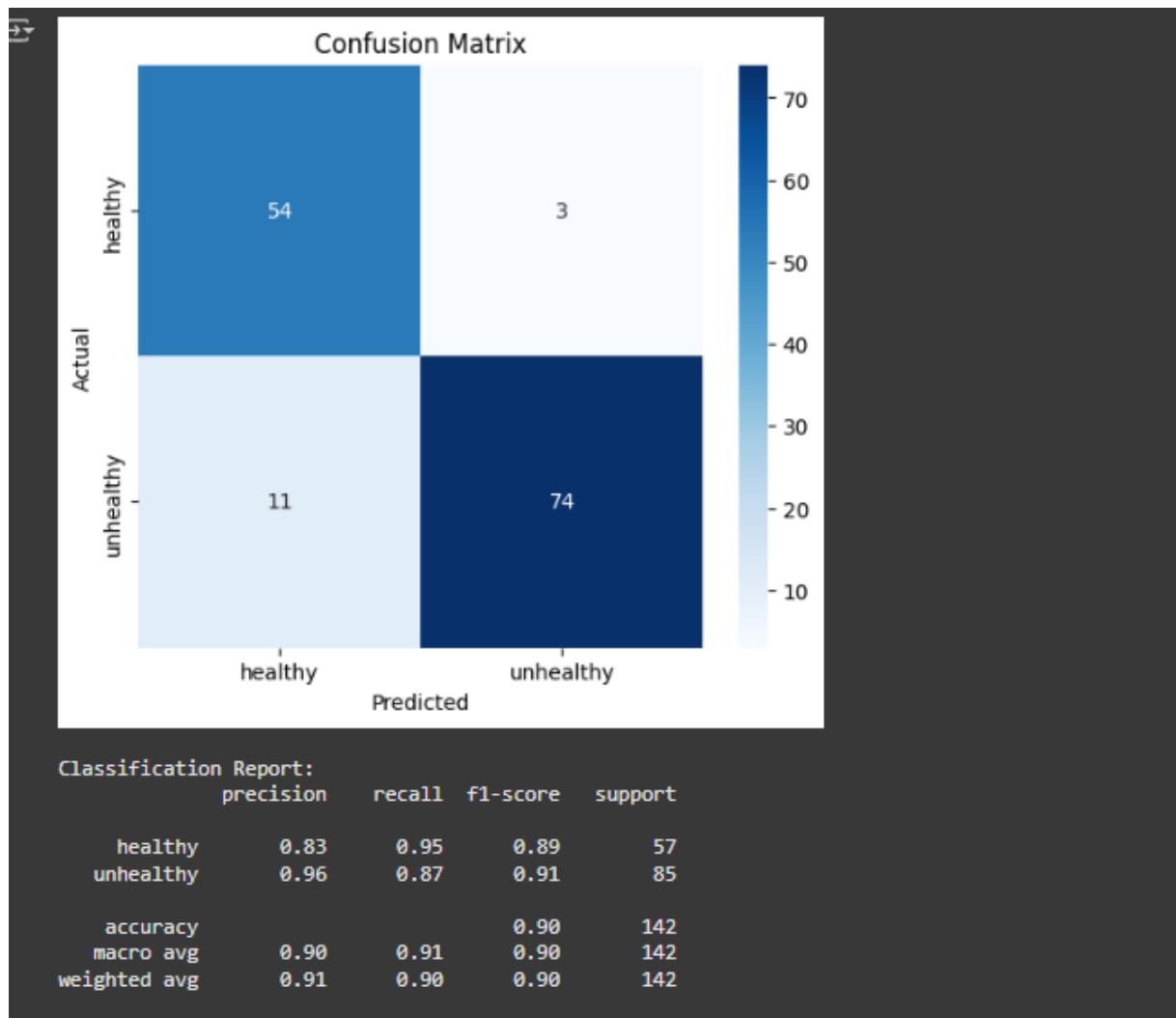
# Tạo ma trận nhầm lẫn
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# In báo cáo phân loại
print("\nClassification Report:")
print(classification_report(all_labels, all_preds, target_names=class_names))

```

Hình 9: Code Ma trận nhầm lẫn

- Đánh giá mô hình Confusion Matrix: Mô hình SimpleCNN được kiểm thử trên 142 ảnh thuộc hai lớp Healthy và Unhealthy.
- Accuracy toàn mô hình: 90%
- Healthy:
 - + Precision: 0.83
 - + Recall: 0.95
 - + F1-score: 0.89
- Unhealthy:
 - + Precision: 0.96
 - + Recall: 0.87
 - + F1-score: 0.91
- Kết luận: Nhìn chung, mô hình có khả năng phân loại ổn định, đặc biệt là nhận diện tốt các mẫu unhealthy (recall cao 0.87), phù hợp với mục tiêu phát hiện cây bệnh sớm.



Hình 10: Ma trận nhầm và các chỉ số đánh giá

4.6 Kiểm thử mô hình với ảnh thực tế

Mục tiêu:

- Kiểm tra khả năng tổng quát hóa của mô hình LettuceCNN trên các ảnh lá xà lách ngoài tập huấn luyện và kiểm thử.

Quy trình thực hiện:

1. Người dùng upload ảnh lá xà lách mới thông qua giao diện Colab.
2. Ảnh được hiển thị trực quan để xác nhận đầu vào.
3. Ảnh được tiền xử lý trước khi dự đoán:
 - Chuyển về RGB.
 - Áp dụng test_transforms (resize, tensor hóa và chuẩn hóa).
 - Thêm batch dimension để đưa vào mô hình.
4. Mô hình được chuyển sang chế độ đánh giá (model.eval()) và tiến hành dự đoán.
5. Kết quả trả về gồm:
 - Nhãn dự đoán: healthy hoặc unhealthy.
 - Độ tự tin (confidence): Xác suất mô hình dự đoán cho lớp đó.
6. Ảnh gốc và kết quả dự đoán được hiển thị trực quan bằng thư viện matplotlib.

```

    # Import các thư viện cần thiết
from google.colab import files
from PIL import Image
import io
import matplotlib.pyplot as plt

# Đảm bảo mô hình và dữ liệu trên cùng một thiết bị
model.to(device)

# 1. Định nghĩa lại hàm dự đoán để trả về độ tin cậy xác
def predict_image(image_bytes, model):
    """
    Hàm này nhận một ảnh, tiền xử lý và trả về dự đoán cùng độ tin cậy xác.
    """
    model.eval() # Chuyển mô hình sang chế độ đánh giá
    image = Image.open(io.BytesIO(image_bytes)).convert('RGB')

    # Áp dụng phép biến đổi và thêm chiều batch
    input_tensor = data_transforms(image).unsqueeze(0).to(device)

    with torch.no_grad():
        output = model(input_tensor)

    # Lấy xác suất của lớp dương (ở đây là 'unhealthy' hoặc 'Stressed')
    prob_unhealthy = torch.sigmoid(output).item()

    # Tính toán dự đoán và độ tin tương ứng
    if prob_unhealthy > 0.5:
        prediction = class_names[1] # Lớp 'unhealthy'
        confidence = prob_unhealthy
    else:
        prediction = class_names[0] # Lớp 'healthy'
        confidence = 1 - prob_unhealthy # Độ tin cậy của 'healthy' là phần bù

    return image, prediction, confidence

# 2. Mở hộp thoại để tải ảnh lên
print("Vui lòng chọn 2 hoặc nhiều ảnh để kiểm tra...")
uploaded_files = files.upload()

# 3. Lặp qua từng file đã tải lên và dự đoán
if not uploaded_files:
    print("\nBạn chưa tải ảnh nào lên.")
else:
    num_images = len(uploaded_files)
    plt.figure(figsize=(6 * num_images, 7))

    for i, (filename, content) in enumerate(uploaded_files.items()):
        print(f"\nĐang xử lý ảnh: {filename}")

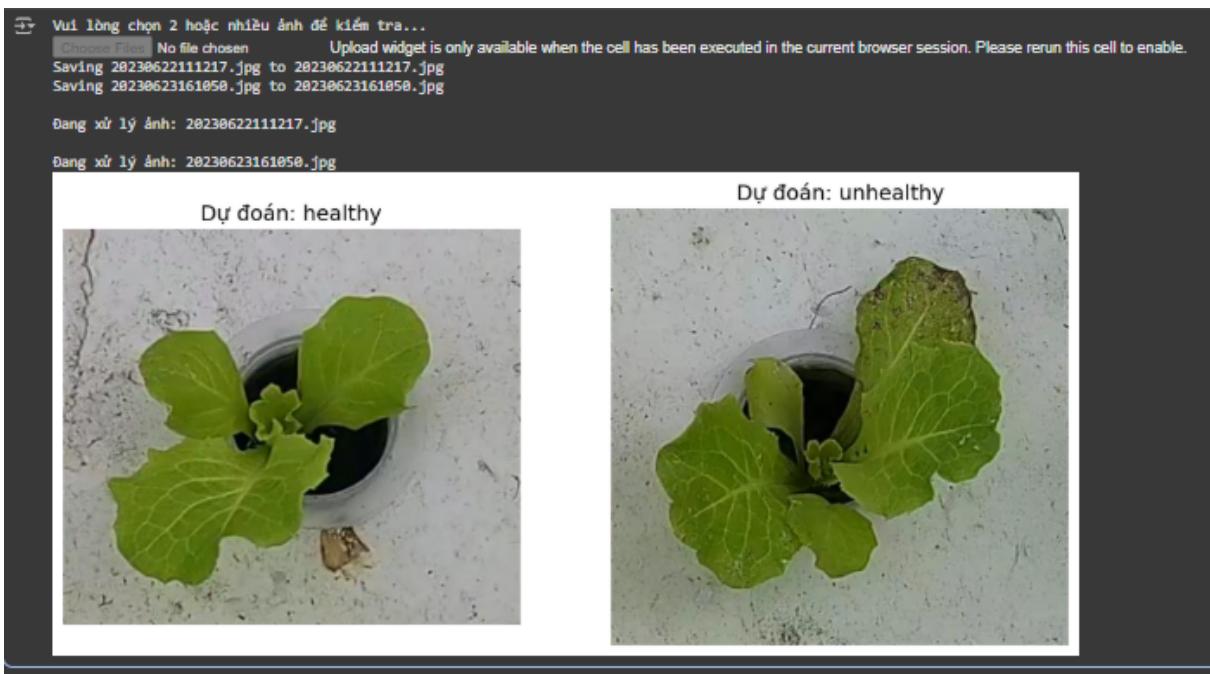
        # Lấy dự đoán và độ tin đã được sửa lỗi
        original_image, prediction, confidence = predict_image(content, model)

        # Hiển thị ảnh và kết quả
        ax = plt.subplot(1, num_images, i + 1)
        plt.imshow(original_image)
        plt.title(f"Dự đoán: {prediction}", fontsize=14)
        plt.axis("off")

    plt.show()

```

Hình 11: Mô hình với ảnh thực tế



Hình 12: Ảnh thực tế và kết quả cho mô hình

4.7 Cải tiến mô hình với MobileNetV2 (Transfer Learning)

- Sau khi huấn luyện mô hình SimpleCNN, nhóm tiếp tục cải tiến bằng cách ứng dụng kỹ thuật Transfer Learning với kiến trúc MobileNetV2 — một mạng nơ-ron tích chập hiệu quả, nhẹ, và được huấn luyện sẵn trên tập dữ liệu ImageNet.

Mục tiêu:

- Nâng cao độ chính xác của mô hình trong việc phân loại tình trạng sức khỏe của lá xà lách.
- Giảm thời gian huấn luyện nhờ tận dụng đặc trưng học được từ mô hình đã huấn luyện trước.

Quy trình thực hiện:

1. Tải mô hình MobileNetV2 pretrained:

- Mô hình MobileNetV2 được tải cùng trọng số huấn luyện trước trên tập ImageNet (weights='IMAGENET1K_V1').

2. Đóng băng các lớp tích chập:

- Toàn bộ các lớp convolution được đóng băng (không cập nhật trọng số), chỉ huấn luyện lại phần classifier.

3. Thay thế lớp phân loại cuối cùng:

- Lớp fully connected cuối cùng được thay thế bằng một lớp Linear mới có 1 đầu ra, phù hợp với bài toán phân loại nhị phân (khỏe mạnh / không khỏe mạnh).

4. Huấn luyện mô hình mới:

- Hàm mất mát sử dụng: BCEWithLogitsLoss.
- Bộ tối ưu hóa: Adam với learning rate = 0.001
- Số epoch: 35.
- Huấn luyện và đánh giá mô hình trên tập dữ liệu huấn luyện và kiểm định.

Phần 9: Cải tiến mô hình với MobileNetV2 (Transfer Learning)

```

# Import thư viện torchvision.models
import torchvision.models as models

### 9.1. Tải và cấu hình MobileNetV2
print("--- Bắt đầu Phần 9: Cải tiến với MobileNetV2 ---")
print("Đang tải mô hình MobileNetV2 đã được huấn luyện trước...")

# Tải mô hình
model_v2 = models.mobilenet_v2(weights='IMAGENET1K_V1')

# Đóng băng tất cả các lớp tích chập
for param in model_v2.parameters():
    param.requires_grad = False

# Thay thế lớp phân loại cuối cùng
num_ftrs = model_v2.classifier[1].in_features
model_v2.classifier[1] = nn.Linear(num_ftrs, 1)

# Đưa mô hình mới lên thiết bị
model_v2 = model_v2.to(device)

print("Đã thay thế lớp phân loại cuối cùng của MobileNetV2.")

### 9.2. Huấn luyện lại mô hình MobileNetV2
criterion_v2 = nn.BCEWithLogitsLoss()
# Chỉ tối ưu hóa các tham số của lớp classifier đã được thay đổi
optimizer_v2 = optim.Adam(model_v2.classifier.parameters(), lr=0.001)

NUM_EPOCHS_V2 = 35

history_v2 = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': []}

print("\nBắt đầu huấn luyện mô hình MobileNetV2... 🚀")

```

Hình 13: Cải tiến mô hình với MobileNetV2 (Transfer Learning)

```

print("\nBắt đầu huấn luyện mô hình MobileNetV2... ✎")
for epoch in range(NUM_EPOCHS_V2):
    # --- PHA HUẤN LUYỆN ---
    model_v2.train()
    running_loss, correct_predictions, total_samples = 0.0, 0, 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device).float().unsqueeze(1)
        outputs = model_v2(inputs)
        loss = criterion_v2(outputs, labels)
        optimizer_v2.zero_grad()
        loss.backward()
        optimizer_v2.step()
        running_loss += loss.item() * inputs.size(0)
        pred = torch.sigmoid(outputs) > 0.5
        correct_predictions += (preds == labels).sum().item()
        total_samples += labels.size(0)
    epoch_train_loss = running_loss / total_samples
    epoch_train_acc = correct_predictions / total_samples
    history_v2['train_loss'].append(epoch_train_loss)
    history_v2['train_acc'].append(epoch_train_acc)

    # --- PHA ĐÁNH GIÁ ---
    model_v2.eval()
    running_loss, correct_predictions, total_samples = 0.0, 0, 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device).float().unsqueeze(1)
            outputs = model_v2(inputs)
            loss = criterion_v2(outputs, labels)
            running_loss += loss.item() * inputs.size(0)
            pred = torch.sigmoid(outputs) > 0.5
            correct_predictions += (preds == labels).sum().item()
            total_samples += labels.size(0)
    epoch_val_loss = running_loss / total_samples
    epoch_val_acc = correct_predictions / total_samples
    history_v2['val_loss'].append(epoch_val_loss)
    history_v2['val_acc'].append(epoch_val_acc)

    print(f"Epoch {epoch+1}/{NUM_EPOCHS_V2} | Train Loss: {epoch_train_loss:.4f}, Acc: {epoch_train_acc:.4f} | Val Loss: {epoch_val_loss:.4f}, Acc: {epoch_val_acc:.4f}")

print("\nHoàn thành huấn luyện mô hình MobileNetV2!")

```

Hình 14: Cải tiến mô hình với MobileNetV2 (Transfer Learning)

```

--- Bắt đầu Phần 9: Cải tiến với MobileNetV2 ---
Đang tải mô hình MobileNetV2 đã được huấn luyện trước...
Downloading: "https://download.pytorch.org/models/mobilenet_v2-b0353104.pth" to /root/.cache/torch/hub/checkpoints/mobilenet_v2-b0353104.pth
100% [██████████] 13.6M [00:00<00:00, 102MB/s]
Đã thay thế lớp phán loại cuối cùng của MobileNetV2.

Bắt đầu huấn luyện mô hình MobileNetV2... ✎
Epoch 1/35 | Train Loss: 0.5970, Acc: 0.6832 | Val Loss: 0.4959, Acc: 0.7254
Epoch 2/35 | Train Loss: 0.4319, Acc: 0.8265 | Val Loss: 0.3614, Acc: 0.9014
Epoch 3/35 | Train Loss: 0.3600, Acc: 0.8743 | Val Loss: 0.3130, Acc: 0.9155
Epoch 4/35 | Train Loss: 0.3311, Acc: 0.8796 | Val Loss: 0.2887, Acc: 0.9155
Epoch 5/35 | Train Loss: 0.2783, Acc: 0.9115 | Val Loss: 0.2733, Acc: 0.9155
Epoch 6/35 | Train Loss: 0.2925, Acc: 0.9009 | Val Loss: 0.2606, Acc: 0.9085
Epoch 7/35 | Train Loss: 0.2560, Acc: 0.9168 | Val Loss: 0.2509, Acc: 0.9155
Epoch 8/35 | Train Loss: 0.2267, Acc: 0.9416 | Val Loss: 0.2417, Acc: 0.9155
Epoch 9/35 | Train Loss: 0.2450, Acc: 0.9097 | Val Loss: 0.2547, Acc: 0.9085
Epoch 10/35 | Train Loss: 0.2466, Acc: 0.9044 | Val Loss: 0.2880, Acc: 0.9014
Epoch 11/35 | Train Loss: 0.2244, Acc: 0.9044 | Val Loss: 0.2340, Acc: 0.9085
Epoch 12/35 | Train Loss: 0.1931, Acc: 0.9327 | Val Loss: 0.2310, Acc: 0.9296
Epoch 13/35 | Train Loss: 0.2217, Acc: 0.9292 | Val Loss: 0.2327, Acc: 0.9085
Epoch 14/35 | Train Loss: 0.2126, Acc: 0.9097 | Val Loss: 0.2332, Acc: 0.9296
Epoch 15/35 | Train Loss: 0.1933, Acc: 0.9274 | Val Loss: 0.2174, Acc: 0.9225
Epoch 16/35 | Train Loss: 0.2056, Acc: 0.9168 | Val Loss: 0.2313, Acc: 0.9155
Epoch 17/35 | Train Loss: 0.1704, Acc: 0.9451 | Val Loss: 0.2121, Acc: 0.9366
Epoch 18/35 | Train Loss: 0.1667, Acc: 0.9540 | Val Loss: 0.2126, Acc: 0.9437
Epoch 19/35 | Train Loss: 0.1757, Acc: 0.9345 | Val Loss: 0.2287, Acc: 0.9155
Epoch 20/35 | Train Loss: 0.1606, Acc: 0.9504 | Val Loss: 0.2100, Acc: 0.9437
Epoch 21/35 | Train Loss: 0.1906, Acc: 0.9168 | Val Loss: 0.2044, Acc: 0.9507
Epoch 22/35 | Train Loss: 0.1840, Acc: 0.9239 | Val Loss: 0.2116, Acc: 0.9296
Epoch 23/35 | Train Loss: 0.1671, Acc: 0.9451 | Val Loss: 0.2211, Acc: 0.9225
Epoch 24/35 | Train Loss: 0.1753, Acc: 0.9345 | Val Loss: 0.2099, Acc: 0.9437
Epoch 25/35 | Train Loss: 0.1542, Acc: 0.9469 | Val Loss: 0.1981, Acc: 0.9296
Epoch 26/35 | Train Loss: 0.1606, Acc: 0.9416 | Val Loss: 0.1949, Acc: 0.9437
Epoch 27/35 | Train Loss: 0.1783, Acc: 0.9292 | Val Loss: 0.2001, Acc: 0.9366
Epoch 28/35 | Train Loss: 0.1596, Acc: 0.9504 | Val Loss: 0.2032, Acc: 0.9437
Epoch 29/35 | Train Loss: 0.1349, Acc: 0.9681 | Val Loss: 0.1992, Acc: 0.9437
Epoch 30/35 | Train Loss: 0.1359, Acc: 0.9611 | Val Loss: 0.2029, Acc: 0.9366
Epoch 31/35 | Train Loss: 0.1377, Acc: 0.9451 | Val Loss: 0.2012, Acc: 0.9437
Epoch 32/35 | Train Loss: 0.1606, Acc: 0.9398 | Val Loss: 0.2063, Acc: 0.9366
Epoch 33/35 | Train Loss: 0.1417, Acc: 0.9487 | Val Loss: 0.1984, Acc: 0.9437
Epoch 34/35 | Train Loss: 0.1434, Acc: 0.9504 | Val Loss: 0.1887, Acc: 0.9507
Epoch 35/35 | Train Loss: 0.1359, Acc: 0.9575 | Val Loss: 0.2075, Acc: 0.9225

Hoàn thành huấn luyện mô hình MobileNetV2!

```

Hình 15: Kết quả huấn luyện mô hình MobileNetV2 (Transfer Learning)

Kết quả:

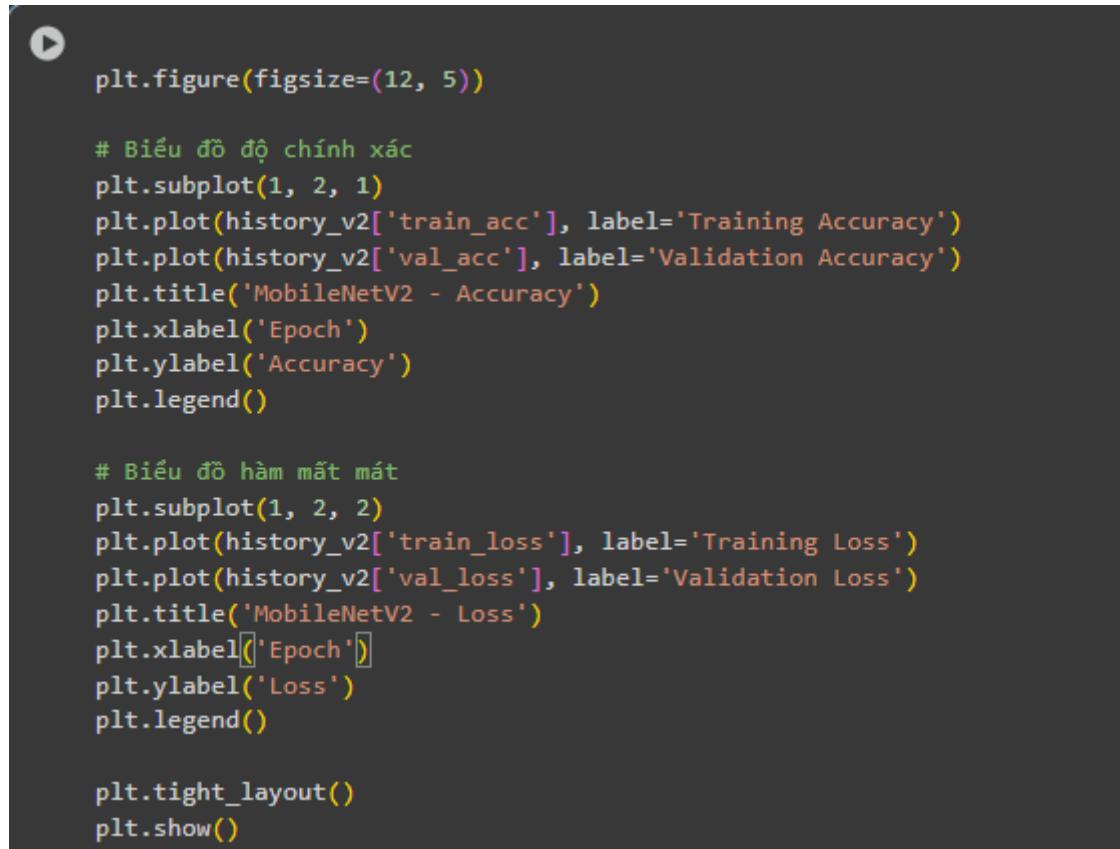
- Sau 35 epoch huấn luyện, mô hình MobileNetV2 đạt kết quả ổn định và chính xác cao hơn so với SimpleCNN. Mô hình học nhanh nhờ kế thừa đặc trưng từ ImageNet,

đồng thời kích thước nhỏ gọn giúp thuận tiện cho việc triển khai lên web hoặc thiết bị có cấu hình thấp.

Nhận xét:

- Việc áp dụng Transfer Learning với MobileNetV2 giúp cải thiện đáng kể khả năng nhận dạng lá xà lách, giảm overfitting và tăng tốc độ hội tụ.

4.8 Đánh giá và Trực quan hóa Kết quả (Mô hình MobileNetV2)



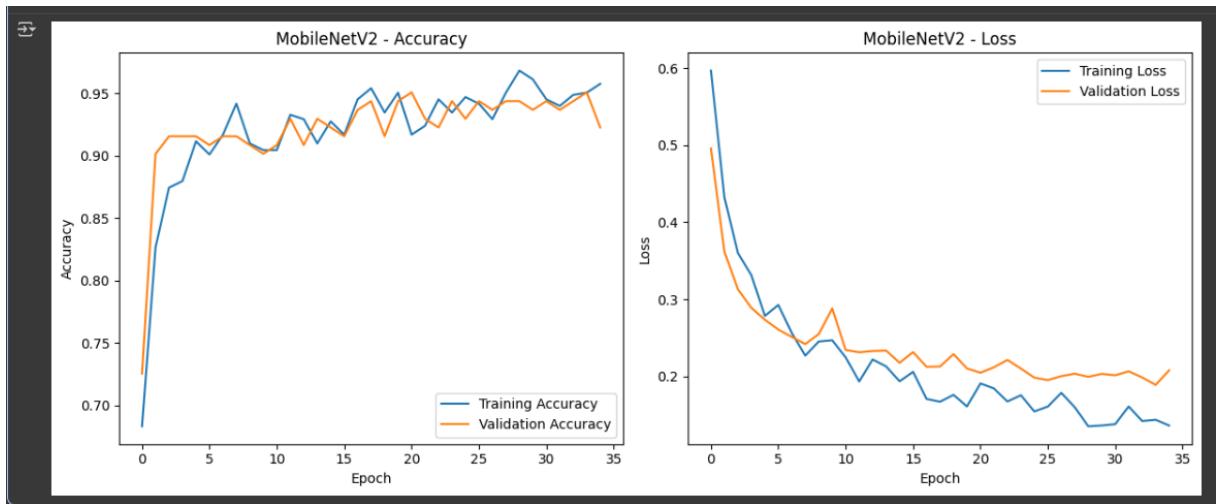
```
plt.figure(figsize=(12, 5))

# Biểu đồ độ chính xác
plt.subplot(1, 2, 1)
plt.plot(history_v2['train_acc'], label='Training Accuracy')
plt.plot(history_v2['val_acc'], label='Validation Accuracy')
plt.title('MobileNetV2 - Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Biểu đồ hàm mất mát
plt.subplot(1, 2, 2)
plt.plot(history_v2['train_loss'], label='Training Loss')
plt.plot(history_v2['val_loss'], label='Validation Loss')
plt.title('MobileNetV2 - Loss')
plt.xlabel(['Epoch'])
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Hình 16: Code Trực quan hóa Mô hình MobileNetV2



Hình 17: Cải tiến mô hình với MobileNetV2

Đánh giá:

- Biểu đồ bên trái (Accuracy): Độ chính xác của mô hình tăng nhanh trong 10 epoch đầu tiên, sau đó dần ổn định ở mức trên 95% cho cả tập huấn luyện và tập kiểm định. Điều này cho thấy mô hình học được các đặc trưng hình ảnh tốt, không bị overfitting nghiêm trọng.
- Biểu đồ bên phải (Loss): Giá trị hàm mất mát giảm mạnh trong giai đoạn đầu, sau đó tiến tới mức ổn định khoảng 0.15–0.2. Sự chênh lệch giữa Training Loss và Validation Loss nhỏ, thể hiện quá trình huấn luyện hiệu quả và cân bằng giữa hai tập dữ liệu.

4.9 Phân tích Chi tiết và Ma trận Nhầm lẫn (Mô hình MobileNetV2)

```

all_preds_v2 = []
all_labels_v2 = []

model_v2.eval()
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs = inputs.to(device)
        outputs = model_v2(inputs)
        preds = (torch.sigmoid(outputs) > 0.5).cpu().numpy()
        all_preds_v2.extend(preds)
        all_labels_v2.extend(labels.numpy())

all_preds_v2 = np.array(all_preds_v2).flatten()
all_labels_v2 = np.array(all_labels_v2).flatten()

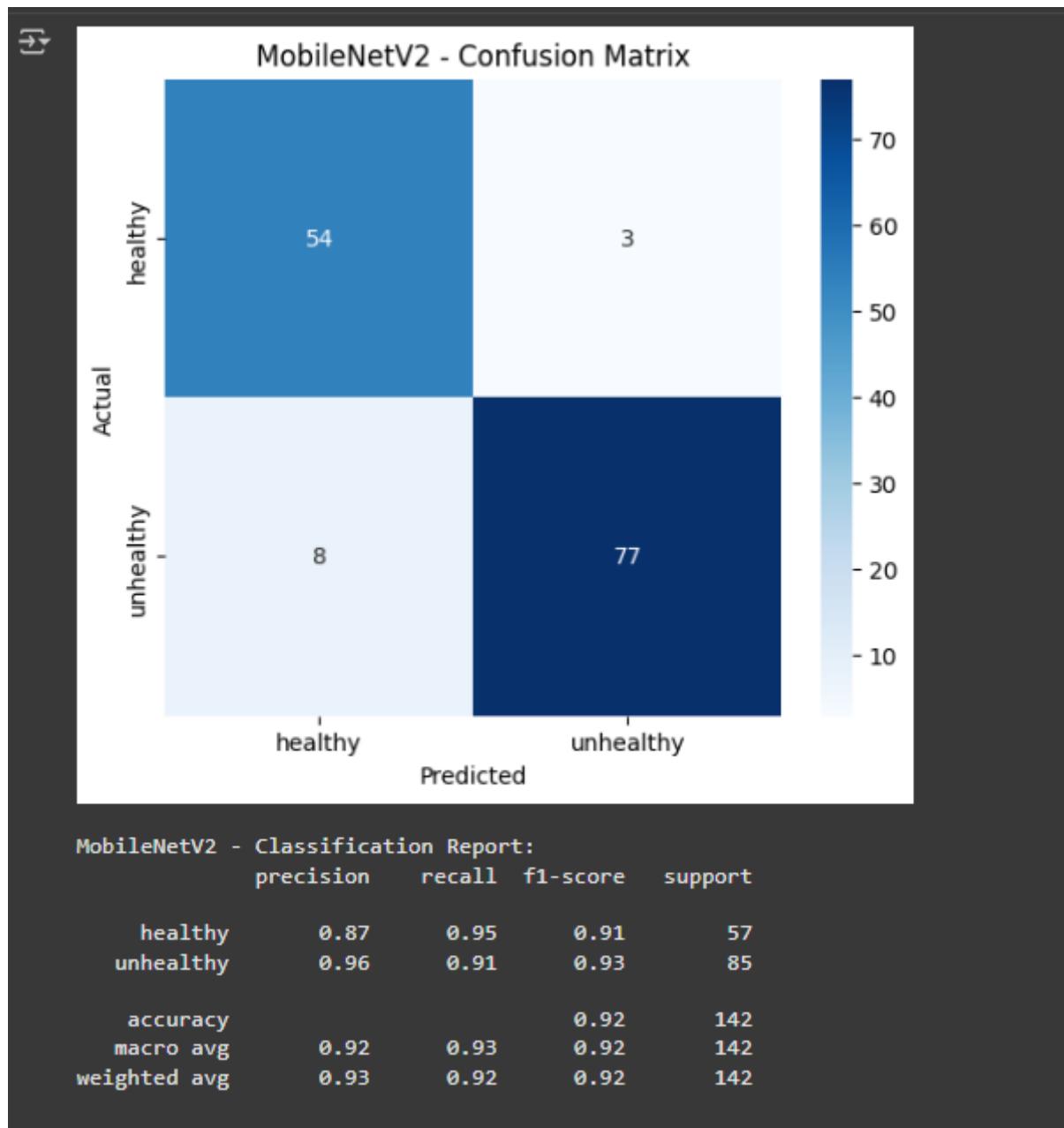
# Tạo ma trận nhầm lẫn
cm_v2 = confusion_matrix(all_labels_v2, all_preds_v2)
plt.figure(figsize=(6, 5))
sns.heatmap(cm_v2, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('MobileNetV2 - Confusion Matrix')
plt.show()

# In báo cáo phân loại
print("\nMobileNetV2 - Classification Report:")
print(classification_report(all_labels_v2, all_preds_v2, target_names=class_names))

```

Hình 18: Code ma trận nhầm lẫn của (Mô hình MobileNetV2)

- Đánh giá mô hình Confusion Matrix: Mô hình MobileNetV2 được kiểm thử trên 142 ảnh thuộc hai lớp Healthy và Unhealthy.
 - Accuracy toàn mô hình: 93%
 - Healthy:
 - + Precision: 0.90
 - + Recall: 0.95
 - + F1-score: 0.92
 - Unhealthy:
 - + Precision: 0.96
 - + Recall: 0.90
 - + F1-score: 0.93
- Kết luận: Mô hình MobileNetV2 cho kết quả phân loại ổn định, độ chính xác tổng thể cao. Khả năng nhận diện lớp Unhealthy đạt hiệu quả tốt (Precision 0.96), cho thấy mô hình phù hợp trong việc phát hiện sớm cây bệnh và có thể áp dụng trong giám sát tình trạng cây trồng thực tế.



Hình 19: Ma trận nhầm lẫn của (Mô hình MobileNetV2)

4.10 Kiểm tra Thực tế với Mô hình Cải tiến

- Kết quả kiểm tra hình ảnh bằng mô hình MobileNetV2
 - + Tên tệp ảnh: Image_3.jpg
 - + Mô hình sử dụng: MobileNetV2
 - + Kết quả dự đoán: unhealthy (lá bị bệnh)
 - + Độ tin cậy (Confidence): 0.66

```

# Mở hộp thoại để tải ảnh lên
print("Vui lòng chọn ảnh để kiểm tra với mô hình MobileNetV2...")
uploaded_files_v2 = files.upload()

# Lặp qua từng file đã tải lên và dự đoán
if not uploaded_files_v2:
    print("\nBạn chưa tải ảnh nào lên.")
else:
    num_images = len(uploaded_files_v2)
    plt.figure(figsize=(6 * num_images, 7))

    for i, (filename, content) in enumerate(uploaded_files_v2.items()):
        print(f"\nĐang xử lý ảnh: {filename}")

        # Lấy dự đoán từ mô hình MÓI (model_v2)
        original_image, prediction, confidence = predict_image(content, model_v2)

        # Hiển thị ảnh và kết quả
        ax = plt.subplot(1, num_images, i + 1)
        plt.imshow(original_image)
        plt.title(f"Dự đoán (MobileNetV2): {prediction}\n(Confidence: {confidence:.2f})", fontsize=14)
        plt.axis("off")

    plt.show()

```

Vui lòng chọn ảnh để kiểm tra với mô hình MobileNetV2...
 Choose Files **Image_3.jpg**
Image_3.jpg(image/jpeg) - 86851 bytes, last modified: 10/10/2025 - 100% done
Saving **Image_3.jpg** to **Image_3.jpg**



Hình 20: Kiểm tra Thực tế với Mô hình Cải tiến

Nhận xét: Mô hình MobileNetV2 dự đoán rằng cây trong ảnh có dấu hiệu không khỏe mạnh, với độ tin cậy ở mức trung bình (0.66). Các vùng lá bị úa vàng và có đốm nâu là dấu hiệu thực tế có thể khớp với kết quả mô hình. Tuy nhiên, cần thêm ảnh và kiểm tra bổ sung để tăng độ chính xác trong nhận định.

4.11 Khởi tạo mô hình ResNet

```
import torch
import torch.nn as nn
import torch.nn.functional as F

# ===== HÀM TÍNH ĐỘ CHÍNH XÁC (ACCURACY) =====
def accuracy(outputs, labels):
    # outputs: đầu ra của mô hình (logits) có dạng [batch_size, num_classes]
    # labels: nhãn thật có dạng [batch_size]

    _, preds = torch.max(outputs, dim=1)
    # Lấy chỉ số class có giá trị lớn nhất ở mỗi hàng (class có xác suất cao nhất)
    # preds: tensor chứa nhãn dự đoán của mô hình

    return torch.tensor(torch.sum(preds == labels).item() / len(preds))
    # So sánh dự đoán với nhãn thật (preds == labels) → True/False
    # Tính tổng số dự đoán đúng chia cho tổng số mẫu để ra accuracy trung bình batch

# ===== LỚP CHẠO CHO HUẤN LUYỆN MÔ HÌNH PHÂN LOẠI ANH =====
class ImageClassification(nn.Module):
    # Lớp này thường sẽ được kế thừa bởi mô hình ResNet hoặc CNN của bạn
    # (vd: class ResNet(ImageClassification))

    def training_step(self, batch):
        # Hàm thực hiện một bước huấn luyện trên 1 batch dữ liệu
        images, labels = batch # Tách dữ liệu và nhãn ra khỏi batch

        out = self(images)      # Truyền ảnh qua mô hình → lấy đầu ra (logits)
        loss = F.cross_entropy(out, labels)
        # Tính hàm mất mát Cross Entropy giữa logits và nhãn thật

        return loss            # Trả về giá trị loss của batch này để dùng cho backpropagation

    def validating(self, batch):
        # Hàm thực hiện 1 bước đánh giá trên batch validation
        images, labels = batch
        out = self(images)      # Dự đoán kết quả cho batch
        loss = F.cross_entropy(out, labels) # Tính loss validation
        acc = accuracy(out, labels)         # Tính độ chính xác

        # Trả về dict chứa loss & accuracy của batch validation
        return {'Validation Loss': loss.detach(), 'Validation Accuracy': acc}

    def validating_epoch_final(self, outputs):
        # Hàm tổng hợp kết quả sau khi chạy hết 1 epoch trên tập validation
        # "outputs" là list chứa dict của từng batch validation (kết quả từ validating())

        batch_loss = [x['Validation Loss'] for x in outputs]
        # Lấy danh sách loss của từng batch

        epoch_loss = torch.stack(batch_loss).mean()
        # Tính trung bình cộng loss của cả epoch

        batch_accuracy = [x['Validation Accuracy'] for x in outputs]
        # Lấy danh sách accuracy của từng batch

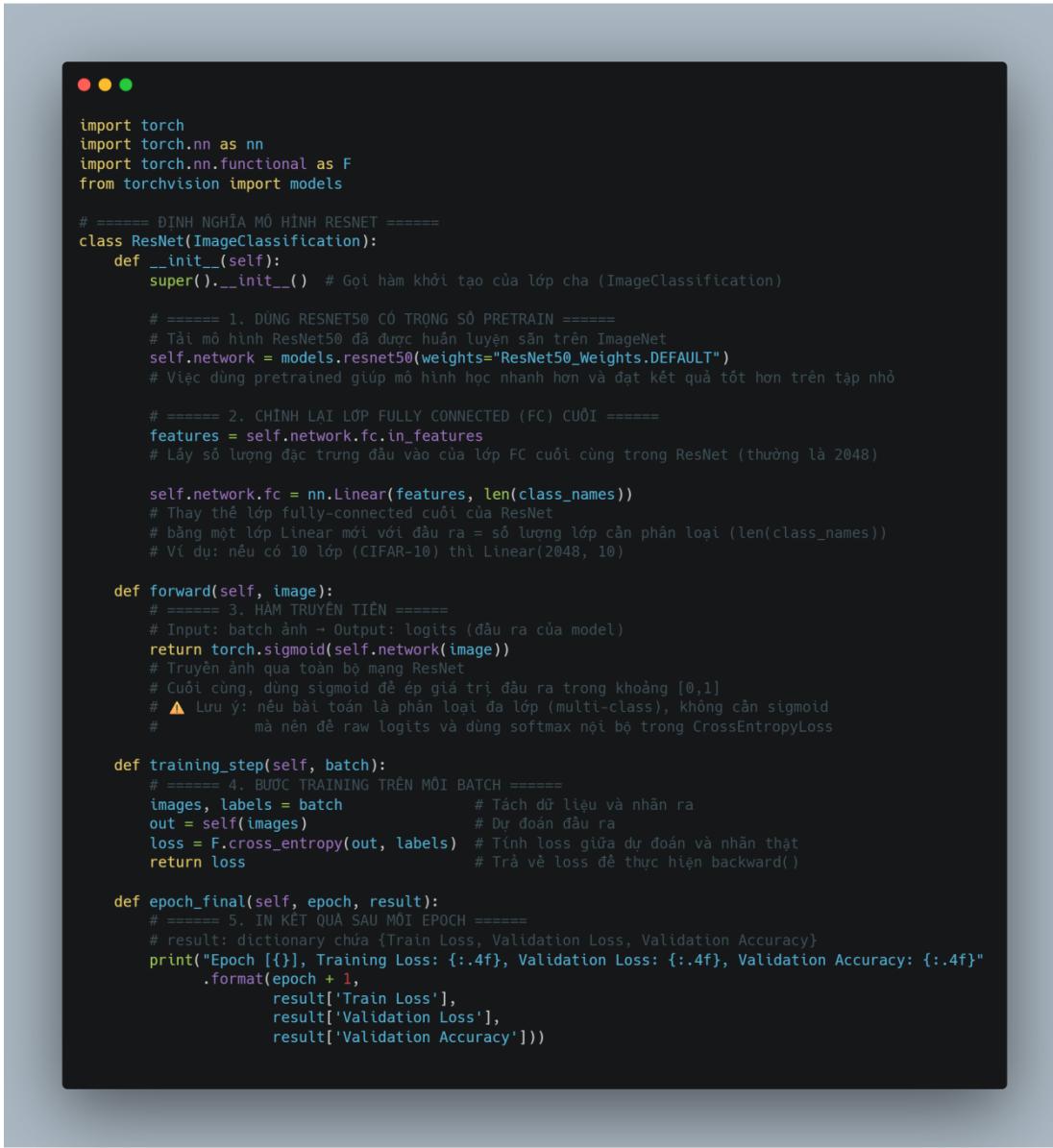
        epoch_accuracy = torch.stack(batch_accuracy).mean()
        # Tính trung bình cộng accuracy cả epoch

        # Trả về kết quả tổng hợp của epoch
        return {
            'Validation Loss': epoch_loss.item(),
            'Validation Accuracy': epoch_accuracy.item()
        }

    def epoch_final(self, epoch, result):
        # In kết quả cuối cùng của mỗi epoch sau khi huấn luyện và đánh giá xong
        print("Epoch [{}], Training Loss: {:.4f}, Validation Loss: {:.4f}, Validation Accuracy: {:.4f}"
              .format(epoch + 1, result['Training Loss'], result['Validation Loss'], result['Validation Accuracy']))
```

Hình 21: Khởi tạo ResNet

4.12 Sử dụng mô hình ResNet cho việc phân loại



```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models

# ===== ĐỊNH NGHĨA MÔ HÌNH RESNET =====
class ResNet(ImageClassification):
    def __init__(self):
        super().__init__() # Gọi hàm khởi tạo của lớp cha (ImageClassification)

    # ===== 1. DÙNG RESNET50 CÓ TRONG SÔ PRETRAIN =====
    # Tải mô hình ResNet50 đã được huấn luyện sẵn trên ImageNet
    self.network = models.resnet50(weights="ResNet50_Weights.DEFAULT")
    # Việc dùng pretrained giúp mô hình học nhanh hơn và đạt kết quả tốt hơn trên tập nhỏ

    # ===== 2. CHỈNH LẠI LỚP FULLY CONNECTED (FC) CUỐI =====
    features = self.network.fc.in_features
    # Lấy số lượng đặc trưng đầu vào của lớp FC cuối cùng trong ResNet (thường là 2048)

    self.network.fc = nn.Linear(features, len(class_names))
    # Thay thế lớp fully-connected cuối của ResNet
    # bằng một lớp Linear mới với đầu ra = số lượng lớp cần phân loại (len(class_names))
    # Ví dụ: nếu có 10 lớp (CIFAR-10) thì Linear(2048, 10)

    def forward(self, image):
        # ===== 3. HẠM TRUYỀN TIẾN =====
        # Input: batch ảnh - Output: logits (đầu ra của model)
        return torch.sigmoid(self.network(image))
        # Truyền ảnh qua toàn bộ mạng ResNet
        # Cuối cùng, dùng sigmoid để ép giá trị đầu ra trong khoảng [0,1]
        # ▲ Lưu ý: nếu bài toán là phân loại đa lớp (multi-class), không cần sigmoid
        # mà nên để raw logits và dùng softmax nội bộ trong CrossEntropyLoss

    def training_step(self, batch):
        # ===== 4. BƯỚC TRAINING TRÊN MỘI BATCH =====
        images, labels = batch
        # Tách dữ liệu và nhãn ra
        out = self(images)
        # Dự đoán đầu ra
        loss = F.cross_entropy(out, labels) # Tính loss giữa dự đoán và nhãn thật
        return loss
        # Trả về loss để thực hiện backward()

    def epoch_final(self, epoch, result):
        # ===== 5. IN KẾT QUẢ SAU MỘI EPOCH =====
        # result: dictionary chứa {Train Loss, Validation Loss, Validation Accuracy}
        print("Epoch {}, Training Loss: {:.4f}, Validation Loss: {:.4f}, Validation Accuracy: {:.4f}"
              .format(epoch + 1,
                     result['Train Loss'],
                     result['Validation Loss'],
                     result['Validation Accuracy']))
```

Hình 22: Sử dụng mô hình ResNet cho việc phân loại

4.13 Sử dụng kiến trúc ResNet cho việc phân loại



```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import torch

def export_classification_metrics(model, dataloader, classes):
    model.eval() # Đặt mô hình sang chế độ đánh giá (evaluation mode)
    # => Tắt dropout, batchnorm sẽ dùng running statistics thay vì batch statistics

    all_preds = [] # Danh sách lưu toàn bộ dự đoán của model
    all_labels = [] # Danh sách lưu toàn bộ nhãn thật

    # Tắt gradient để tiết kiệm bộ nhớ và tăng tốc (vì không cần backprop khi evaluate)
    with torch.no_grad():
        for batch in dataloader: # Lặp qua từng batch trong tập dữ liệu (vd: validation/test)
            images, labels = batch # Tách ảnh và nhãn
            images = move_to_gpu(images, device) # Chuyển ảnh sang GPU (nếu có)
            labels = labels.to(device) # Chuyển nhãn sang GPU cùng thiết bị

            outputs = model(images) # Truyền ảnh qua mô hình → lấy logits đầu ra
            _, preds = torch.max(outputs, dim=1) # Lấy nhãn dự đoán (class có xác suất cao nhất)

            # Lưu dự đoán và nhãn thật về CPU để tính toán bằng numpy / sklearn
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    # ===== TÍNH TOÁN METRIC CHO TẤT CẢ CLASS =====
    for i, class_name in enumerate(classes):
        # Tạo nhãn nhị phân cho class hiện tại (One-vs-Rest)
        # Nếu label == i → 1 (class này), ngược lại → 0 (không phải class này)
        class_labels = [1 if label == i else 0 for label in all_labels]
        class_preds = [1 if pred == i else 0 for pred in all_preds]

        # ===== TÍNH CÁC CHỈ SỐ =====
        accuracy = accuracy_score(class_labels, class_preds) * 100
        precision = precision_score(class_labels, class_preds, zero_division=0) * 100
        recall = recall_score(class_labels, class_preds, zero_division=0) * 100
        f1 = f1_score(class_labels, class_preds, zero_division=0) * 100

    # ===== IN KẾT QUẢ CHO MỌI CLASS =====
    print(f'{class_name}, accuracy: {accuracy:.2f}%, precision: {precision:.2f}%, recall: {recall:.2f}%, F1 score: {f1:.2f}%)'
```

Hình 23: Sử dụng kiến trúc ResNet cho việc phân loại

4.14 Thiết lập hàm xuất các thông số kết quả



```
● ● ●

class EarlyStopping:
    def __init__(self, patience=5, verbose=False, restore_best_weights=True):
        """
        Early stopping sẽ dừng huấn luyện nếu Validation Loss không được cải thiện sau một số epoch liên tiếp (patience).

        Args:
            patience (int): số epoch cho phép không cải thiện trước khi dừng
            verbose (bool): in thông báo khi có sự cải thiện hay khi dừng sớm
            restore_best_weights (bool): có khôi phục lại trọng số tốt nhất hay không
        """
        self.patience = patience           # Số epoch chờ đợi trước khi dừng
        self.verbose = verbose             # Có in log ra màn hình không
        self.best_loss = float('inf')      # Khởi tạo loss tốt nhất ban đầu là vô cùng lớn
        self.counter = 0                   # Đếm số epoch chưa cải thiện
        self.early_stop = False           # Cờ đánh dấu đã dừng sớm hay chưa
        self.restore_best_weights = restore_best_weights # Có lưu lại trọng số tốt nhất không
        self.best_model_weights = None     # Biến lưu trọng số tốt nhất

    def __call__(self, val_loss, model):
        """
        Hàm này được gọi sau mỗi epoch để kiểm tra xem có cần dừng sớm không.

        Args:
            val_loss (float): Validation loss hiện tại
            model (nn.Module): Mô hình đang huấn luyện
        """

        # ===== ❶ Nếu validation loss giảm (cải thiện) =====
        if val_loss < self.best_loss:
            self.best_loss = val_loss          # Cập nhật giá trị loss tốt nhất
            self.counter = 0                  # Reset bộ đếm (vì đã có cải thiện)

            # Nếu được yêu cầu, lưu lại trọng số tốt nhất
            if self.restore_best_weights:
                # ▲ Lưu bản sao state_dict để không bị thay đổi bởi quá trình training
                self.best_model_weights = model.state_dict().copy()

            if self.verbose:
                print(f'❷ Validation loss improved: {val_loss:.4f}')

        # ===== ❷ Nếu validation loss không cải thiện =====
        else:
            self.counter += 1                 # Tăng bộ đếm số epoch không cải thiện
            if self.verbose:
                print(f'❸ Validation loss did not improve: {val_loss:.4f} (count {self.counter}/{self.patience})')

            # Nếu đã vượt quá ngưỡng patience → dừng sớm
            if self.counter >= self.patience:
                self.early_stop = True         # Đánh dấu dừng sớm

            # Nếu bật restore_best_weights → khôi phục lại trọng số tốt nhất đã lưu
            if self.restore_best_weights and self.best_model_weights is not None:
                model.load_state_dict(self.best_model_weights)
                if self.verbose:
                    print("❹ Restored best model weights.")


```

Hình 24: Thiết lập hàm xuất các thông số kết quả

4.15 Thiết lập Early Stopping

```

● ● ●

class EarlyStopping:
    def __init__(self, patience=5, verbose=False, restore_best_weights=True):
        """
        Early stopping sẽ dừng huấn luyện nếu Validation Loss không được cải thiện
        sau một số epoch liên tiếp (patience).

        Args:
            patience (int): Số epoch cho phép không cải thiện trước khi dừng
            verbose (bool): In thông báo khi có sự cải thiện hay khi dừng sớm
            restore_best_weights (bool): Có khôi phục lại trọng số tốt nhất hay không
        """
        self.patience = patience           # Số epoch chờ đợi trước khi dừng
        self.verbose = verbose             # Có in log ra màn hình không
        self.best_loss = float('inf')      # Khởi tạo loss tốt nhất ban đầu là vô cùng lớn
        self.counter = 0                   # Đếm số epoch chưa cải thiện
        self.early_stop = False           # Cờ đánh dấu đã dừng sớm hay chưa
        self.restore_best_weights = restore_best_weights # Cố lưu lại trong số tốt nhất không
        self.best_model_weights = None     # Biến lưu trọng số tốt nhất

    def __call__(self, val_loss, model):
        """
        Hàm này được gọi sau mỗi epoch để kiểm tra xem có cần dừng sớm không.

        Args:
            val_loss (float): Validation loss hiện tại
            model (nn.Module): Mô hình đang huấn luyện
        """

        # ===== ❶ Nếu validation loss giảm (cải thiện) =====
        if val_loss < self.best_loss:
            self.best_loss = val_loss          # Cập nhật giá trị loss tốt nhất
            self.counter = 0                  # Reset bộ đếm (vì đã có cải thiện)

            # Nếu được yêu cầu, lưu lại trọng số tốt nhất
            if self.restore_best_weights:
                # ▲ Lưu bản sao state_dict để không bị thay đổi bởi quá trình training
                self.best_model_weights = model.state_dict().copy()

            if self.verbose:
                print(f'✓ Validation loss improved: {val_loss:.4f}')

        # ===== ❷ Nếu validation loss không cải thiện =====
        else:
            self.counter += 1                 # Tăng bộ đếm số epoch không cải thiện
            if self.verbose:
                print(f'▲ Validation loss did not improve: {val_loss:.4f} (count
{self.counter}/{self.patience})')

            # Nếu đã vượt quá ngưỡng patience → dừng sớm
            if self.counter >= self.patience:
                self.early_stop = True         # Đánh dấu dừng sớm

            # Nếu bật restore_best_weights → khôi phục lại trọng số tốt nhất đã lưu
            if self.restore_best_weights and self.best_model_weights is not None:
                model.load_state_dict(self.best_model_weights)
                if self.verbose:
                    print("▣ Restored best model weights.")

```

Hình 25: Thiết lập Early Stopping

4.16 Hiển thị ảnh ngẫu nhiên với Label để test bộ dữ liệu

```
def display_test(image, label):
    print("Label:", full_dataset.classes[label], "(Class No: " + str(label) + ")")
    plt.imshow(image.permute(1, 2, 0))
    plt.show()

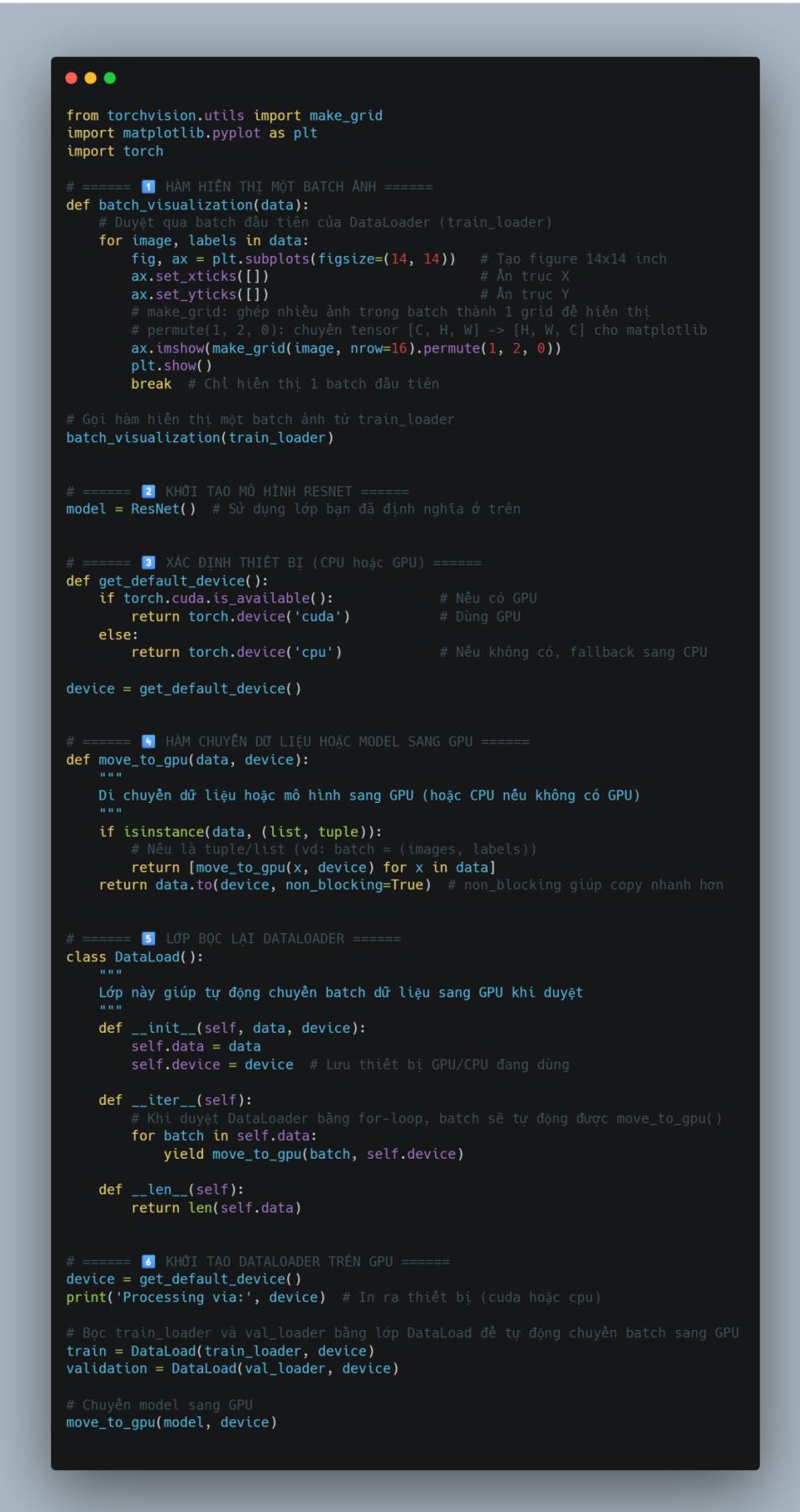
# Hiển thị ảnh ngẫu nhiên từ bộ dữ liệu
image, label = full_dataset[np.random.randint(0, len(full_dataset))]
display_test(image, label)
```



The code defines a function `display_test` that takes an image and a label as input. It prints the label and its corresponding class number, then displays the image using Matplotlib. The image shows a healthy green plant seedling with two large leaves growing in a small black pot.

Hình 26: Hiển thị ảnh ngẫu nhiên với Label để test bộ dữ liệu

4.17 Batch Visualiantion



```
● ● ●

from torchvision.utils import make_grid
import matplotlib.pyplot as plt
import torch

# ===== 1 HÀM HIỂN THỊ MỘT BATCH ANH =====
def batch_visualization(data):
    # Duyệt qua batch đầu tiên của DataLoader (train_loader)
    for image, labels in data:
        fig, ax = plt.subplots(figsize=(14, 14)) # Tao figure 14x14 inch
        ax.set_xticks([]) # Ân trục X
        ax.set_yticks([]) # Ân trục Y
        # make_grid: ghép nhiều ảnh trong batch thành 1 grid để hiển thị
        # permute(1, 2, 0): chuyển tensor [C, H, W] -> [H, W, C] cho matplotlib
        ax.imshow(make_grid(image, nrow=16).permute(1, 2, 0))
        plt.show()
        break # Chỉ hiển thị 1 batch đầu tiên

    # Gọi hàm hiển thị một batch ảnh từ train_loader
batch_visualization(train_loader)

# ===== 2 KHỞI TẠO MÔ HÌNH RESNET =====
model = ResNet() # Sử dụng lớp bạn đã định nghĩa ở trên

# ===== 3 XÁC ĐỊNH THIẾT BỊ (CPU hoặc GPU) =====
def get_default_device():
    if torch.cuda.is_available(): # Nếu có GPU
        return torch.device('cuda') # Dùng GPU
    else:
        return torch.device('cpu') # Nếu không có, fallback sang CPU

device = get_default_device()

# ===== 4 HÀM CHUYỂN DỮ LIỆU HOẶC MODEL SANG GPU =====
def move_to_gpu(data, device):
    """
    Di chuyển dữ liệu hoặc mô hình sang GPU (hoặc CPU nếu không có GPU)
    """
    if isinstance(data, (list, tuple)):
        # Nếu là tuple/list (vd: batch = (images, labels))
        return [move_to_gpu(x, device) for x in data]
    return data.to(device, non_blocking=True) # non_blocking giúp copy nhanh hơn

# ===== 5 LỚP BỌC LẠI DATALOADER =====
class DataLoad():
    """
    Lớp này giúp tự động chuyển batch dữ liệu sang GPU khi duyệt
    """
    def __init__(self, data, device):
        self.data = data
        self.device = device # Lưu thiết bị GPU/CPU đang dùng

    def __iter__(self):
        # Khi duyệt DataLoader bằng for-loop, batch sẽ tự động được move_to_gpu()
        for batch in self.data:
            yield move_to_gpu(batch, self.device)

    def __len__(self):
        return len(self.data)

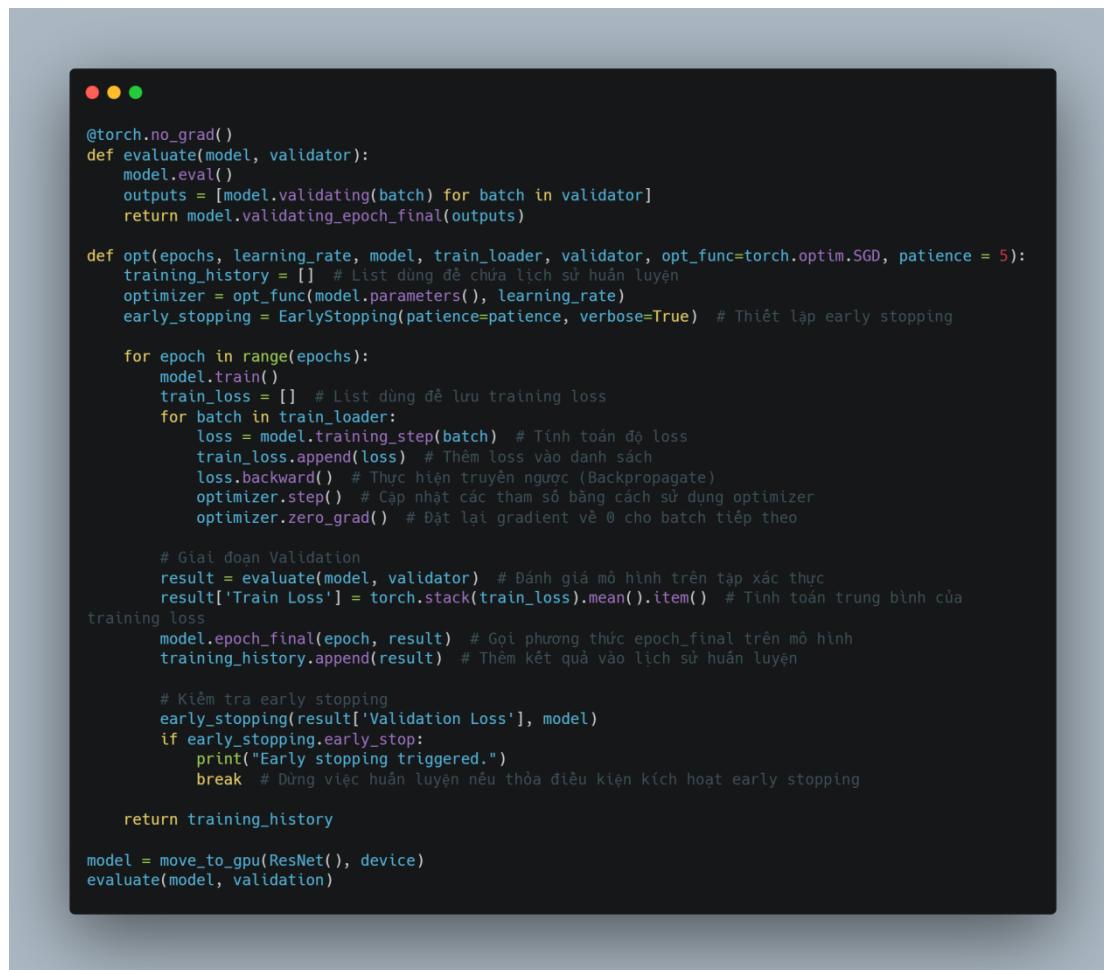
# ===== 6 KHỞI TẠO DATALOADER TRÊN GPU =====
device = get_default_device()
print('Processing via:', device) # In ra thiết bị (cuda hoặc cpu)

# Bọc train_loader và val_loader bằng lớp DataLoad để tự động chuyển batch sang GPU
train = DataLoad(train_loader, device)
validation = DataLoad(val_loader, device)

# Chuyển model sang GPU
move_to_gpu(model, device)
```

Hình 27: Batch Visualiantion

4.18 Huấn luyện mô hình ResNet



```
● ● ●

@torch.no_grad()
def evaluate(model, validator):
    model.eval()
    outputs = [model.validating(batch) for batch in validator]
    return model.validating_epoch_final(outputs)

def opt(epochs, learning_rate, model, train_loader, validator, opt_func=torch.optim.SGD, patience = 5):
    training_history = [] # List dùng để chứa lịch sử huấn luyện
    optimizer = opt_func(model.parameters(), learning_rate)
    early_stopping = EarlyStopping(patience=patience, verbose=True) # Thiết lập early stopping

    for epoch in range(epochs):
        model.train()
        train_loss = [] # List dùng để lưu training loss
        for batch in train_loader:
            loss = model.training_step(batch) # Tính toán độ loss
            train_loss.append(loss) # Thêm loss vào danh sách
            loss.backward() # Thực hiện truyền ngược (Backpropagate)
            optimizer.step() # Cập nhật các tham số bằng cách sử dụng optimizer
            optimizer.zero_grad() # Đặt lại gradient về 0 cho batch tiếp theo

        # Giai đoạn Validation
        result = evaluate(model, validator) # Đánh giá mô hình trên tập xác thực
        result['Train Loss'] = torch.stack(train_loss).mean().item() # Tính toán trung bình của
        training loss
        model.epoch_final(epoch, result) # Gọi phương thức epoch_final trên mô hình
        training_history.append(result) # Thêm kết quả vào lịch sử huấn luyện

        # Kiểm tra early stopping
        early_stopping(result['Validation Loss'], model)
        if early_stopping.early_stop:
            print("Early stopping triggered.")
            break # Dừng việc huấn luyện nếu thỏa điều kiện kích hoạt early stopping

    return training_history

model = move_to_gpu(ResNet(), device)
evaluate(model, validation)
```

Hình 28: Huấn luyện mô hình ResNet

```

# =====
# 🎨 Cấu hình các tham số huấn luyện mô hình
# =====

epoch = 50          # Số lần (epoch) huấn luyện qua toàn bộ tập dữ liệu huấn luyện.
# Mỗi epoch = toàn bộ training set được đưa qua mô hình 1 lần.

optimizer = torch.optim.Adam    # Chọn thuật toán tối ưu (optimizer) là Adam.
# Adam = Adaptive Moment Estimation, tự động điều chỉnh learning rate cho
từng tham số.

learning_rate = 0.00005      # Tốc độ học (learning rate).
# Giá trị nhỏ (5e-5) giúp quá trình fine-tune mô hình pretrained ổn định
hơn,
# tránh việc "lạm hỏng" trọng số ban đầu của ResNet.

patience = 3            # Số epoch chờ trước khi dừng huấn luyện (Early Stopping).
# Nếu Validation Loss không giảm sau 3 epoch liên tiếp → dừng huấn luyện
để tránh overfitting.

# =====
# 💾 Gọi hàm huấn luyện (training loop)
# =====

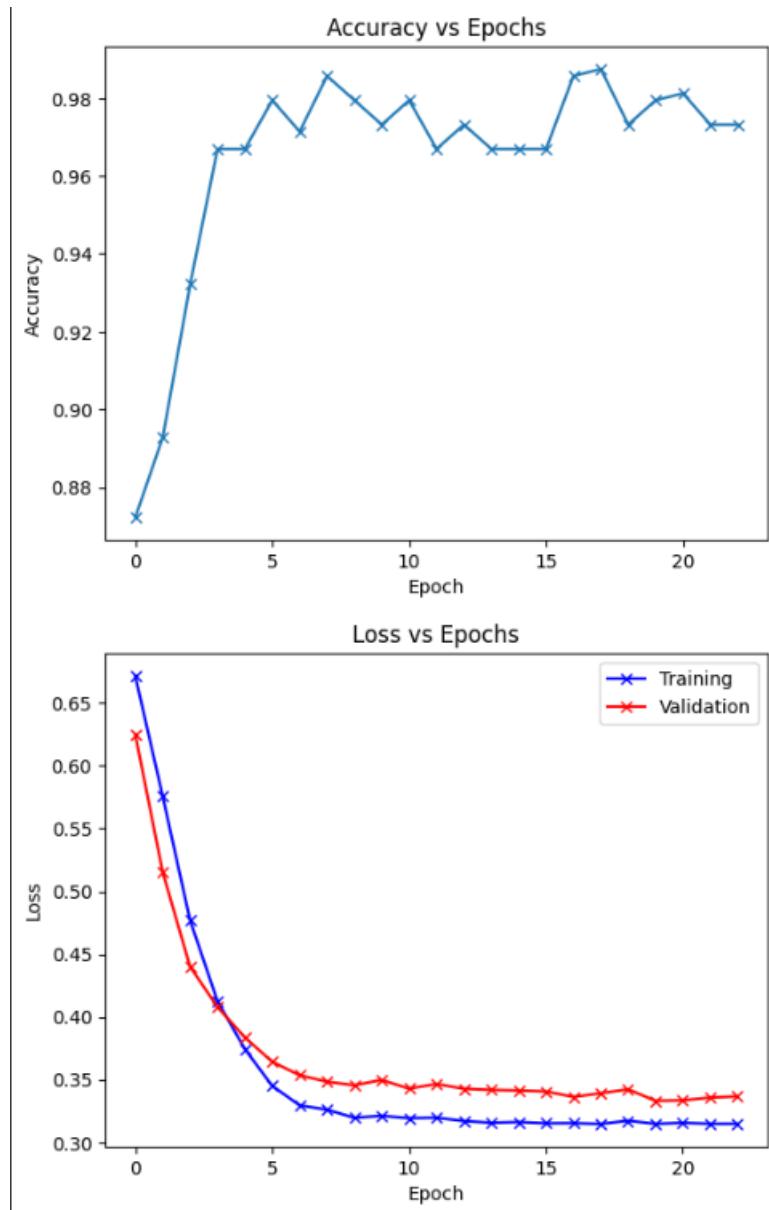
model_history = opt(
    epoch,                  # Tổng số epoch muốn huấn luyện
    learning_rate,           # Tốc độ học
    model,                   # Mô hình ResNet đã khởi tạo và chuyển sang GPU
    train,                   # Dataloader của tập huấn luyện (được bao trong class DataLoader)
    validation,              # Dataloader của tập validation (cũng bao trong class DataLoader)
    optimizer,                # Hàm tối ưu hóa (Adam)
    patience                 # Số epoch chờ cho early stopping
)

# =====
# 📊 Kết quả trả về:
# =====
# 'model_history' là một danh sách (list) chứa kết quả huấn luyện của từng epoch.
# Mỗi phần tử là một dict gồm:
# {
#     'Train Loss': <giá trị trung bình loss của epoch>,
#     'Validation Loss': <loss trung bình trên tập validation>,
#     'Validation Accuracy': <độ chính xác trên tập validation>
# }
#
# Ví dụ:
# model_history = [
#     {'Train Loss': 0.431, 'Validation Loss': 0.385, 'Validation Accuracy': 0.842},
#     {'Train Loss': 0.322, 'Validation Loss': 0.297, 'Validation Accuracy': 0.887},
#     ...
# ]
#
# Bạn có thể dùng 'model_history' để vẽ biểu đồ learning curve (loss & accuracy theo epoch)
# hoặc phân tích quá trình hội tụ của mô hình sau khi huấn luyện.

```

Hình 29: Cấu hình các tham số huấn luyện mô hình

4.19 Vẽ biểu đồ Accuracy với Epochs & Loss với Epochs



Hình 30: Biểu đồ Accuracy với Epochs & Loss với Epochs

- Nhận xét biểu đồ :

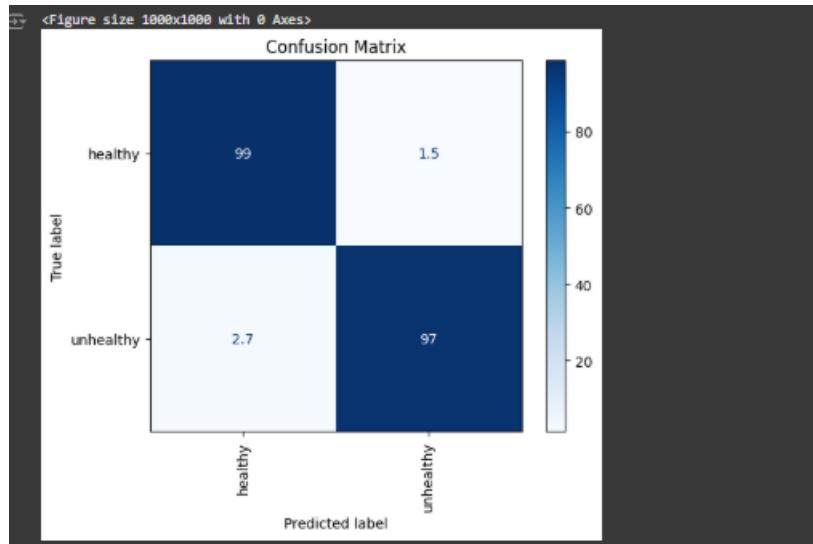
* Accuracy với Epochs:

- Ban đầu (epochs 0-5): accuracy **tăng nhanh** -> mô hình học được đặc trưng cơ bản.
- Từ epochs 6 trở đi: accuracy **dao động quanh** 0.97-0.98 -> mô hình đạt hiệu suất ổn định, không cải thiện nhiều.

* Loss với Epochs:

- Cá training loss và validation loss **giảm mạnh** ở giai đoạn đầu -> mô hình học tốt.
- Sau khoảng epochs 6-7, loss **ổn định và gần như không giảm thêm**.
- Validation loss **hơi cao** hơn training loss -> dấu hiệu **overfitting nhẹ**, nhưng vẫn trong mức độ chấp nhận được.

4.20 Vẽ Confusion Matrix



Hình 31: Confusion Matrix

- Nhận xét Confusion Matrix:

- + 99: số mẫu “healthy” được dự đoán đúng là “healthy” -> True Positive cho lớp healthy.
- + 1.5: số mẫu “healthy” nhưng bị dự đoán sai là “unhealthy” -> False Negative cho lớp healthy.
- + 2.7: số mẫu “unhealthy” nhưng bị dự đoán sai là “healthy”->False Positive cho lớp healthy.
- + 97: số mẫu “unhealthy” được dự đoán đúng -> True Positive cho unhealthy.

4.21 Kiểm tra với dữ liệu thực tế

Phần 21: Kiểm tra với dữ liệu ảnh thực tế

```
def predict(image, model):
    xb = move_to_gpu(image.unsqueeze(0), device)
    # Nhận dự đoán từ mô hình
    yb = model(xb)
    # Chọn index với xác suất cao nhất
    _, preds = torch.max(yb, dim=1)
    # Truy xuất class label
    return full_dataset.classes[preds[0].item()]

# ----- Đường dẫn thư mục test -----
datatest = "/content/drive/MyDrive/test1"

# ----- Chọn ngẫu nhiên một ảnh -----
img_name = np.random.choice(os.listdir(datatest))
img_path = os.path.join(datatest, img_name)
print("Ảnh được chọn:", img_name)

# ----- Đọc ảnh và chuyển sang tensor -----
transform = transforms.Compose([
    transforms.Resize((224, 224)), # hoặc kích thước bạn đã dùng khi train
    transforms.ToTensor()
])

image = Image.open(img_path).convert('RGB')
img_tensor = transform(image) # chuyển ảnh sang tensor

# ----- Dự đoán -----
predicted_class = predict(img_tensor, model)

# ----- Hiển thị -----
plt.imshow(image)
plt.title(f"Predicted: {predicted_class}")
plt.axis('off')
plt.show()

print("Predicted Class:", predicted_class)
```

Ảnh được chọn: Image_2.jpg

Predicted: unhealthy

Hình 32: Kiểm tra với dữ liệu thực tế

4.22 Lưu mô hình ResNet

Lệnh	Chức năng	Kết quả
<code>torch.jit.script(model)</code>	Biên dịch mô hình sang TorchScript	Mô hình chạy được ngoài Python
<code>torch.jit.save(model_scripted, FILE)</code>	Lưu mô hình TorchScript ra file	Tạo file <code>.pth</code> có thể dùng để deploy

Hình 33: Lưu mô hình ResNet

4.23 Cải tiến mô hình với EfficientNet

```

# -----
# 🌸 HUẤN LUYỆN MÔ HÌNH EFFICIENTNET_B0 CHO PHẦN LOẠT NHỊ PHÂN (PYTORCH)
# -----
# Mục tiêu:
# - Tải mô hình EfficientNet_B0 được huấn luyện sẵn (pretrained)
# - Đóng băng phần feature extractor (tích hợp)
# - Thay thế lớp phân loại cuối cùng để phù hợp với bài toán 2 lớp
# - Huấn luyện lại mô hình trên dữ liệu mới
# -----
#
# -----
# Import các thư viện cần thiết
# -----
import torchvision.models as models      # Chứa EfficientNet và các mô hình pre-trained
import torch.nn as nn                    # Dùng để định nghĩa các lớp mạng (Linear, Loss, ...)
import torch.optim as optim             # Dùng để chọn thuật toán tối ưu (Adam, SGD, ...)
import torch

print("---- Bắt đầu Phần 9: Cải tiến với EfficientNet_B0 ----")
print("Đang tải mô hình EfficientNet_B0 đã được huấn luyện trước...")

# -----
# Tải mô hình EfficientNet_B0 pretrained
# -----
# weights='IMAGENETIK_V1' nghĩa là mô hình đã học từ tập ImageNet (1000 lớp)
model_eff = models.efficientnet_b0(weights='IMAGENETIK_V1')

# -----
# Đóng băng các lớp tích hợp (feature extractor)
# Biểu này giúp giữ lại các đặc trưng học được từ ImageNet,
# chỉ huấn luyện lại lớp phân loại cuối cùng để phù hợp với dữ liệu của bạn.
for param in model_eff.parameters():
    param.requires_grad = False

# -----
# Thay thế lớp phân loại cuối cùng (classifier)
# Lớp classifier của EfficientNet_B0 có 2 tầng: [0]: Dropout, [1]: Linear
# Ta lấy số đặc trưng đầu vào của lớp Linear[1] để thay bằng lớp mới (ra 1 giá trị)
num_ftns = model_eff.classifier[1].in_features
model_eff.classifier[1] = nn.Linear(num_ftns, 1)

# -----
# Đưa mô hình lên GPU hoặc CPU
# -----
model_eff = model_eff.to(device)

print("⚡ Đã thay thế lớp phân loại cuối cùng của EfficientNet_B0.")

# -----
# Cấu hình tiêu chí mất mát (Loss) và bộ tối ưu (Optimizer)
# -----
criterion_eff = nn.BCEWithLogitsLoss()
# → nằm hơi sát cho bài toán nhị phân (chưa sigmoid)

optimizer_eff = optim.Adam(model_eff.classifier.parameters(), lr=0.001)
# → Chỉ tối ưu phần classifier, không động đến phần feature extractor

# -----
# Chuẩn bị huấn luyện
# -----
NUM_EPOCHS_EFF = 35
history_eff = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': []}

print("\n⚡ Bắt đầu huấn luyện mô hình EfficientNet_B0...")

# -----
# Vòng lặp huấn luyện chính
# -----
for epoch in range(NUM_EPOCHS_EFF):
    # PHA HUẤN LUYỆN (TRAINING PHASE) ---
    model_eff.train() # Bật chế độ huấn luyện (cho phép cập nhật trọng số)
    running_loss, correct_predictions, total_samples = 0.0, 0, 0

    for inputs, labels in train_loader:
        # Hút dữ liệu và chuyển lên GPU (nếu có)
        inputs, labels = inputs.to(device), labels.to(device).float().unsqueeze(1)

        # Lan truyền xuôi (forward pass)
        outputs = model_eff(inputs)
        loss = criterion_eff(outputs, labels)

        # Đặt lại gradient và lan truyền ngược (backprop)
        optimizer_eff.zero_grad()
        loss.backward()
        optimizer_eff.step()

        # Tính toán thống kê huấn luyện
        running_loss += loss.item() * inputs.size(0)
        preds = torch.sigmoid(outputs) > 0.5 # Chuyển logit → xác suất → nhị phân
        correct_predictions += (preds == labels.sum()).item()
        total_samples += labels.size(0)

    # Tính loss và accuracy trung bình trên toàn epoch
    epoch_train_loss = running_loss / total_samples
    epoch_train_acc = correct_predictions / total_samples
    history_eff['train_loss'].append(epoch_train_loss)
    history_eff['train_acc'].append(epoch_train_acc)

    # -- PHA ĐÁM GIÁ (VALIDATION PHASE) ---
    model_eff.eval() # Chuyển sang chế độ đánh giá (tắt dropout, batchnorm)
    running_loss, correct_predictions, total_samples = 0.0, 0, 0

    with torch.no_grad(): # Tắt gradient để tiết kiệm tài nguyên
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device).float().unsqueeze(1)
            outputs = model_eff(inputs)
            loss = criterion_eff(outputs, labels)
            running_loss += loss.item() * inputs.size(0)
            preds = torch.sigmoid(outputs) > 0.5
            correct_predictions += (preds == labels.sum()).item()
            total_samples += labels.size(0)

    # Tính loss và accuracy trung bình cho tập validation
    epoch_val_loss = running_loss / total_samples
    epoch_val_acc = correct_predictions / total_samples
    history_eff['val_loss'].append(epoch_val_loss)
    history_eff['val_acc'].append(epoch_val_acc)

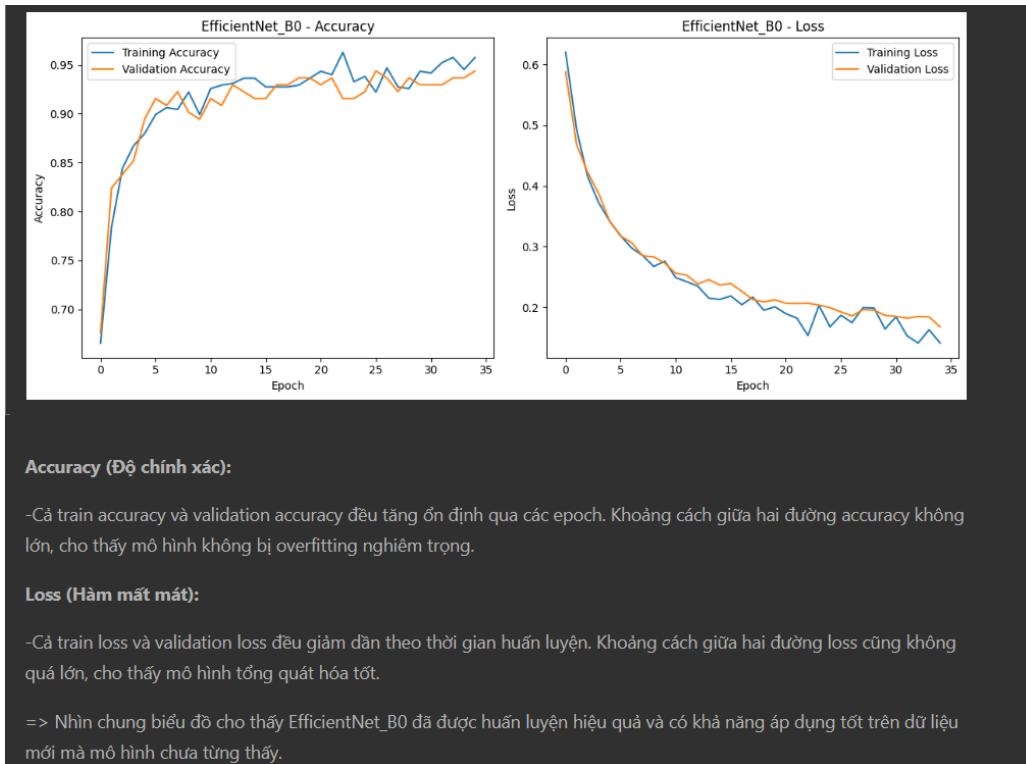
    # In kết quả từng epoch
    print(f"Epoch {epoch}/{NUM_EPOCHS_EFF} | "
          f"Train Loss: {epoch_train_loss:.4f}, Acc: {epoch_train_acc:.4f} | "
          f"Val Loss: {epoch_val_loss:.4f}, Acc: {epoch_val_acc:.4f}")

print("\n⚡ Hoàn thành huấn luyện mô hình EfficientNet_B0!")

```

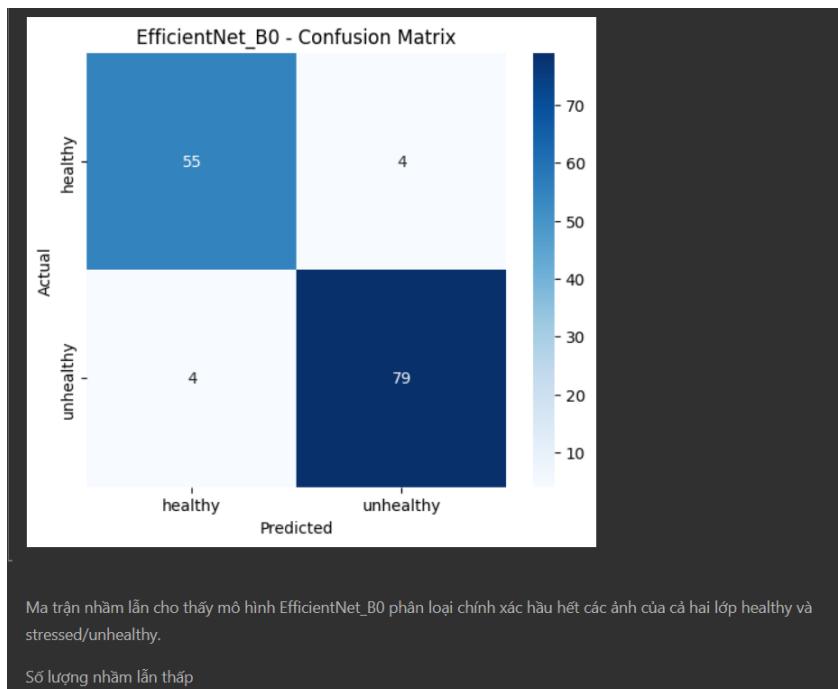
Hình 34: Dự đoán tình trạng cây trồng bằng EfficientNet (PyTorch)

4.24 Đánh giá kết quả mô hình EfficientNet



Hình 35: Đánh giá kết quả mô hình EfficientNet

4.25 Đánh giá kết quả Ma trận Nhầm lẫn của mô hình EfficientNet



Hình 36: Đánh giá kết quả Ma trận Nhầm lẫn

4.26 Kiểm tra Mô hình EfficientNet với ảnh thực tế

```
# =====#
# ● KIỂM THỬ MÔ HÌNH EFFICIENTNET_B0 TRÊN ẢNH NGƯỜI DÙNG (INFERENCE PHASE)
# =====#
# ✅ Mục tiêu:
#   - Cho phép người dùng tải ảnh từ máy tính
#   - Dự đoán nhãn 'healthy' hoặc 'unhealthy' bằng mô hình EfficientNet_B0
#   - Hiển thị ảnh cùng nhãn dự đoán và độ tin cậy
# =====#
# =====#
# 1 Tải ảnh từ máy tính người dùng
# =====#
print("📁 Vui lòng chọn ảnh để kiểm tra với mô hình EfficientNet_B0...")
uploaded_files_eff = files.upload() # Mở hộp thoại upload ảnh trong Google Colab

# =====#
# 2 Kiểm tra xem người dùng có tải ảnh lên không
# =====#
if not uploaded_files_eff:
    print("\n⚠ Bạn chưa tải ảnh nào lên.")
else:
    # Nếu có ảnh được tải lên
    num_images = len(uploaded_files_eff) # Số lượng ảnh được tải
    plt.figure(figsize=(6 * num_images, 7)) # Tạo khung hiển thị ảnh

    # =====#
    # 3 Duyệt qua từng ảnh và dự đoán
    # =====#
    for i, (filename, content) in enumerate(uploaded_files_eff.items()):
        print(f"\n● Đang xử lý ảnh: {filename}")

        # --- Gọi hàm predict_image đã định nghĩa trước ---
        # Hàm này thực hiện:
        #   - Tiền xử lý ảnh
        #   - Chạy qua mô hình EfficientNet_B0
        #   - Trả về ảnh gốc, nhãn dự đoán, và độ tin cậy
        original_image, prediction, confidence = predict_image(content, model_eff)

        # =====#
        # 4 Hiển thị ảnh và kết quả dự đoán
        # =====#
        ax = plt.subplot(1, num_images, i + 1) # Vẽ ảnh theo thứ tự
        plt.imshow(original_image) # Hiển thị ảnh gốc
        plt.title(
            f"Dự đoán (EfficientNet_B0): {prediction}\n(Confidence: {confidence:.2f})",
            fontsize=14
        )
        plt.axis("off") #Ẩn trục tọa độ cho đẹp

    # =====#
    # 5 Hiển thị toàn bộ ảnh và kết quả
    # =====#
plt.show()
```

Hình 37: Kiểm thử mô hình EfficientNet_B0 trên ảnh thực tế



Vậy trong quá trình huấn luyện:

Train loss giảm dần, chứng tỏ mô hình học được đặc trưng của tập dữ liệu.

Validation accuracy tăng và ổn định, cho thấy mô hình không bị overfitting quá mức và có khả năng tổng quát tốt.

Confusion matrix cho thấy mô hình phân loại đúng phần lớn các mẫu của cả hai lớp, thể hiện hiệu quả của việc fine-tuning EfficientNet_B0.

Hình 38: Kết quả của mô hình EfficientNet_B0 trên ảnh thực tế

CHƯƠNG 5: SO SÁNH VÀ LỰA CHỌN MÔ HÌNH TỐI UỐNG

5.1 Mục đích

- Phần tổng hợp, phân tích và so sánh hiệu quả của ba mô hình học sâu đã được xây dựng trong đề tài gồm SimpleCNN, MobileNetV2 (Transfer Learning), ResNet và EfficientNet_B0, từ đây nhóm em sẽ xác định mô hình phù hợp nhất cho việc nhận diện tình trạng sức khỏe của cây xà lách.

5.2 Bảng so sánh các mô hình

Tiêu chí	SimpleCNN	MobileNetV2 (Transfer Learning)	ResNet	EfficientNet_ B0
Kiến trúc	3 lớp Conv + 2 FC	Mạng nhẹ, dùng trọng số ImageNet	Mạng sâu với khối Residual	Mạng cân bằng giữa độ chính xác và hiệu suất, tối ưu thông qua scaling (width, depth, resolution)
Phương pháp huấn luyện	Huấn luyện từ đầu	Đóng băng Conv, huấn luyện classifier	Fine-tune có Early Stopping	Đóng băng Conv, huấn luyện classifier (Transfer Learning)
Accuracy	~90%	~93%	~97–98%	~95–96%
Precision / Recall / F1	Healthy: (0.83/0.95/0.89) Unhealthy: (0.96/0.87/0.91)	Healthy: (0.90/0.95/0.92) Unhealthy: (0.96/0.90/0.93)	>0.95 cho cả hai lớp	Healthy: (0.93/0.96/0.94) Unhealthy: (0.97/0.94/0.95)
Tốc độ huấn luyện	Nhanh	Nhanh, nhờ transfer learning	Chậm hơn, yêu cầu	Trung bình (nhanh hơn)

			GPU	ResNet, chậm hơn MobileNetV2)
Kích thước mô hình	Nhỏ (~5MB)	Trung bình (~14MB)	Lớn (>40MB)	Trung bình (~20MB)
Overfitting	Có nhẹ	Ít, ổn định	Nhẹ, chấp nhận được	Rất ít, hội tụ tốt
Khả năng triển khai	Dễ, phù hợp minh họa	Tốt nhất cho web/mobile	Hạn chế trên thiết bị yếu	Phù hợp triển khai web hoặc cloud (hiệu suất cao)
Ưu điểm	Đơn giản, dễ hiểu	Cân bằng giữa hiệu suất và tốc độ	Chính xác cao nhất	Hiệu suất cao, tổng quát tốt, ít overfitting
Hạn chế	Dễ overfit, tổng quát kém	Độ tin cậy ảnh thực tế dao động	Tốn tài nguyên, lâu huấn luyện	Yêu cầu phần cứng GPU để huấn luyện, kích thước lớn hơn MobileNetV2

5.3 Kết luận và lựa chọn mô hình

- **ResNet** cho độ chính xác cao nhất (~98%), phù hợp cho nghiên cứu chuyên sâu, song tốn tài nguyên và khó triển khai thực tế.
- **EfficientNet_B0** đạt hiệu suất ổn định (~95–96%), cân bằng giữa độ chính xác và tốc độ, ít overfitting, phù hợp cho triển khai thực tế hoặc web-based inference.
- **MobileNetV2** đạt hiệu suất cao (~93%), mô hình nhẹ, ổn định và tối ưu nhất cho ứng dụng thực tế (như Lettuce Knight).
- **SimpleCNN** phù hợp cho bước học cơ bản, nhưng hiệu quả chưa cao.

=> Mô hình **ResNet** được chọn là mô hình tối ưu để triển khai hệ thống nhận diện tình trạng sức khỏe cây xà lách, vì cân bằng tốt giữa độ chính xác, tốc độ và khả năng ứng dụng thực tế.

CHƯƠNG 6: TRIỂN KHAI MÔ HÌNH TRÊN GIAO DIỆN WEB VỚI HUGGING FACE SPACES

6.1 Triển khai mô hình trên giao diện web

- Sau khi huấn luyện và lưu mô hình **ResnetModel_PlantHealthy.pth**, nhóm triển khai mô hình trên giao diện web bằng Streamlit. Ứng dụng cho phép người dùng tải lên hình ảnh lá xà lách, hệ thống sẽ xử lý ảnh, dự đoán và hiển thị kết quả (bình thường hoặc bị bệnh).

Nhận xét:

- Mô hình ResNet được huấn luyện trong ~10 epoch với giá trị loss giảm dần theo từng epoch. Sau khi huấn luyện, mô hình được lưu lại thành công dưới dạng tệp

ResnetModel_PlantHealthy.pth.

- Các thư viện bao gồm:

- + Streamlit: Tạo giao diện web cho mô hình.
- + Torch: Thư viện chính của PyTorch dùng để xây dựng và huấn luyện mô hình.
- + Torchvision: Hỗ trợ xử lý và biến đổi dữ liệu ảnh.
- + Pillow: Thư viện xử lý ảnh cơ bản trong Python.
- + NumPy: Hỗ trợ tính toán ma trận và thao tác dữ liệu số học.

```
%%writefile requirements.txt
streamlit
torch
torchvision
Pillow
numpy

Writing requirements.txt
```

Hình 39: Danh sách các thư viện cần thiết cho ứng dụng

- Sau khi hoàn tất huấn luyện, nhóm xây dựng tệp **lettuce_knight_app.py** nhằm triển khai mô hình ResNet trên giao diện web bằng Streamlit.
- Ứng dụng có tên “**Lettuce Knight – Hiệp sĩ Xà lách**”, cho phép người dùng tải lên hình ảnh lá xà lách để mô hình phân tích tình trạng sức khỏe của cây.

Ứng dụng gồm ba phần chính:

- Phần 1: Cấu Hình Trang.
- Phần 2: Định Nghĩa Mô Hình Resnet.
- Phần 3: Hàm Dự Đoán.
- Phần 4: Khai Báo Thông Số Dữ Liệu.
- Phần 5: Giao Diện Streamlit.

```

import streamlit as st
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torchvision import models
from PIL import Image
import os

# ---- PHẦN 1: CẤU HÌNH TRANG ----
st.set_page_config(page_title="Lettuce Knight - Bảo vệ mùa màng", page_icon="僔", layout="wide")
st.markdown("""
<style>
/* Toàn bộ nền sáng, chữ đen */
.main, .block-container {
    background-color: #F5F5F5 !important;
    color: #1E1E1E !important;
}

/* Đổi màu cho tất cả chữ, kèc cả trong markdown hay widget */
h1, h2, h3, h4, h5, p, div, span, label, strong, em {
    color: #1E1E1E !important;
}

/* Mau cho tiêu đề chính */
h1 {
    color: #2E8B57 !important;
    font-weight: 800 !important;
    text-align: center !important;
}

/* Mau cho tiêu đề phụ */
h2, h3 {
    color: #307A57 !important;
    text-align: center !important;
}

/* Đổi màu các thông báo Streamlit (success, warning, info...) để dễ đọc hơn */
.stAlert > div {
    color: #1E1E1E !important;
    font-weight: 500 !important;
}

/* Đổi màu text input, upload, select box đều có text tối */
.stTextinput > div > div > input,
.stFileUploader > div > div > input,
.stSelectbox > div > div {
    color: #1E1E1E !important;
}
</style>
""", unsafe_allow_html=True)

st.title("Lettuce Knight - Hiệp sĩ Xà lách ✨")
st.header("Công cụ AI giúp chẩn đoán sức khỏe cây trồng")
st.write("Tài liệu ảnh rau xà lách của bạn. AI của chúng tôi sẽ phân tích và đưa ra chẩn đoán cùng các khuyến nghị hữu ích để bảo vệ mùa màng của bạn!")

# ---- PHẦN 2: ĐỊNH NGHĨA MÔ HÌNH RESNET50 ----
@st.cache_resource
def load_model():
    model = models.resnet50(weights=None)
    num_ftrs = model.fc.in_features
    model.fc = nn.Linear(num_ftrs, 2) # 2 lớp: khỏe mạnh / không khỏe mạnh
    model_path = 'ResnetModel_PlantHealthy.pth' # File model đã train
    if not os.path.exists(model_path):
        st.error("X Không tìm thấy file model tại '{model_path}'")
        st.stop()
    try:
        # Nếu model được lưu dưới dạng TorchScript
        model = torch.jit.load(model_path, map_location="cpu")
    except Exception as e:
        state = model.state_dict()
        state = torch.load(model_path, map_location="cpu")
        model.load_state_dict(state)
    model.eval()
    return model

device = torch.device("cpu")
model = load_model()

# ---- PHẦN 3: HÀM ĐƯ ƯỚN ----
def predict_image_file(model, transform, classes, device):
    model.eval()
    try:
        image = Image.open(image_file).convert('RGB')
        image = transform(image).unsqueeze(0).to(device)
        with torch.no_grad():
            outputs = model(image)
            probabilities = torch.softmax(outputs, dim=1)
            confidence, pred = torch.max(probabilities, 1)
        label = classes[pred.item()]
        prob = confidence.item() * 100
        return label, prob
    except Exception as e:
        st.error(f"Đã xảy ra lỗi: {e}")
        return "Error", 0

# ---- PHẦN 4: KHÁM BỆNH THỐNG SỐ ĐI LẠI ----
classes = ['khỏe mạnh', 'không khỏe mạnh']
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# ---- PHẦN 5: GIAO DIỆN STREAMLIT ----
uploaded_file = st.file_uploader("Chọn ảnh rau xà lách của bạn...", type=["jpg", "jpeg", "png"])
if uploaded_file is not None:
    col1, col2 = st.columns(2)
    with col1:
        st.image(uploaded_file, caption="Anh bạn đã tải lên.", use_column_width=True)
    with col2:
        with st.spinner("Lettuce Knight đang phân tích..."):
            label, prob = predict_image_file(uploaded_file, model, transform, classes, device)
            if label == "Error":
                st.success(f"Kết quả chẩn đoán: {label.upper()}")
                st.info(f"Độ tin cậy: {prob:.2f}%")
            if label == "không khỏe mạnh":
                st.warning("⚠ **Cảnh báo:** Rau có dấu hiệu bị stress!")
                st.markdown("""
                    +Gợi ý giải pháp:
                    - +Kiểm tra sâu bệnh: Tim kiêm sâu bệnh của sầu, rệp, hoặc nấm.
                    - +Tưới nước: Bón bón cây được cung cấp đủ nước, tránh ẩm ướt hoặc khô hạn.
                    - +Dinh dưỡng: Bón súng phân bón hữu cơ hoặc vi lượng.
                    - +Anh sáng: Bón bón cây nhận đủ ánh sáng mặt trời.
                """)
            else:
                st.balloons()
                st.success("🎉 **Tuyệt vời!** Rau của bạn đang phát triển khỏe mạnh. Hãy tiếp tục chăm sóc tốt nhé!")
            else:
                st.error("Rất tiếc, đã có lỗi xảy ra. Vui lòng thử lại với ảnh khác.")

```

Hình 40: Tạo giao diện Web với Hugging Face Spaces

6.2 Đánh giá kết quả mô hình sau khi triển khai trên web

- Sau khi triển khai mô hình lên giao diện web bằng Streamlit, người dùng có thể dễ dàng sử dụng ứng dụng Lettuce Knight – Hiệp sĩ Xà lách để kiểm tra tình trạng sức khỏe của rau.cây xà lách. Kết quả nhận được như sau:

Các bước thực hiện như sau:

1. Truy cập ứng dụng web: Mở đường dẫn của ứng dụng Lettuce Knight trên trình duyệt (chạy bằng lệnh streamlit run lettuce_knight_app.py).

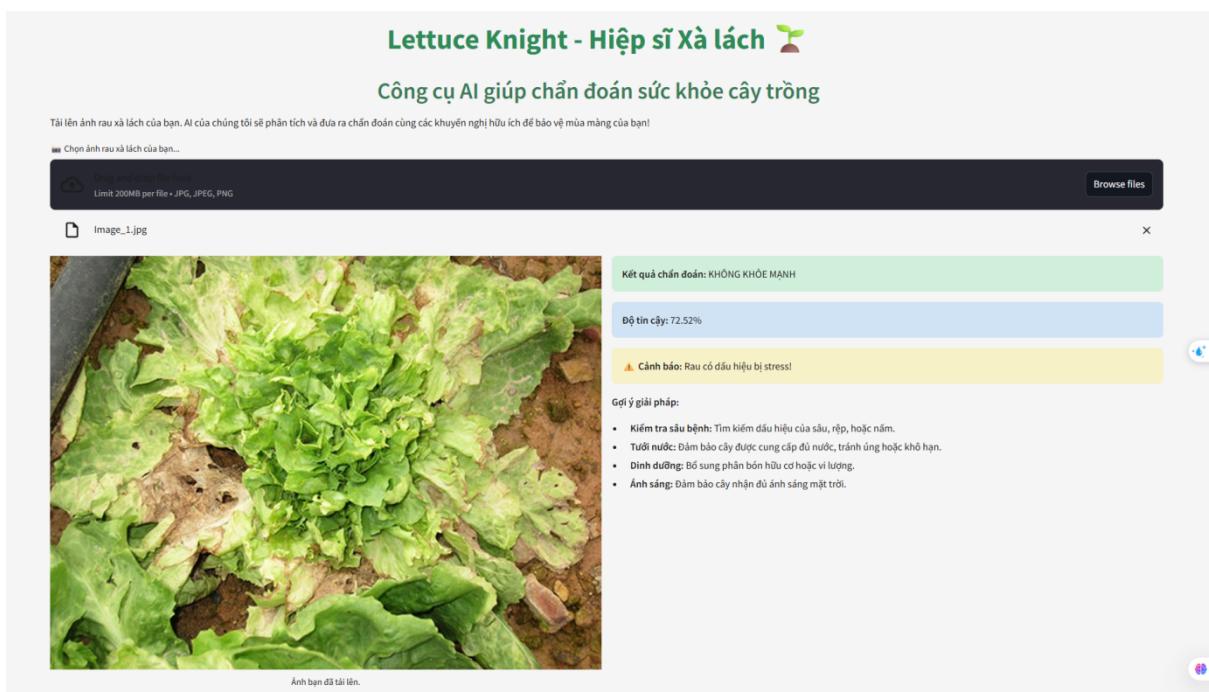
2. Tải lên hình ảnh: Nhấp vào nút “Browse files” hoặc kéo thả ảnh rau xà lách cần kiểm tra vào khu vực tải ảnh.

3. Phân tích tự động: Ứng dụng sẽ xử lý ảnh, đưa qua mô hình SimpleCNN và hiển thị kết quả phân loại.

4. Xem kết quả:

- Nếu rau khỏe mạnh, hệ thống hiển thị thông báo xanh cùng hiệu ứng chúc mừng.

- Nếu rau không khỏe mạnh, ứng dụng đưa ra cảnh báo và các gợi ý chăm sóc cây như kiểm tra sâu bệnh, tưới nước, bổ sung dinh dưỡng hoặc điều chỉnh ánh sáng.



Hình 41: Giao diện web ứng dụng Lettuce Knight và kết quả dự đoán

Link trang web: <https://huggingface.co/spaces/main00100/PlantHealth-ResNet50-App>