**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NÔNG LÂM TP HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

TIỂU LUẬN TỐT NGHIỆP

**NGHIÊN CỨU MÔ HÌNH HỌC MÁY NHẬN DẠNG LOẠI BỆNH TRÊN CÂY LÚA Ở GIAI ĐOẠN ĐẺ NHÁNH**

**Ngành : Công nghệ thông tin**

**Niên khoá : 2021-2025**

**Lớp : DH21DTC**

**Sinh viên thực hiện : Kiều Thị Ngọc Qúy**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NÔNG LÂM TP HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

TIỂU LUẬN TỐT NGHIỆP

**NGHIÊN CỨU MÔ HÌNH HỌC MÁY NHẬN DẠNG LOẠI BỆNH TRÊN CÂY LÚA Ở GIAI ĐOẠN ĐẺ NHÁNH**

**CÁN BỘ HƯỚNG DẪN SINH VIÊN THỰC HIỆN**

TS. Nguyễn Thị Phương Trâm Kiều Thị Ngọc Qúy (MSSV: 21130504)

TP. HỒ CHÍ MINH, tháng 4 năm 2024

# DANH SÁCH CHỮ VIẾT TẮT

# DANH MỤC HÌNH ẢNH

# DANH MỤC BẢNG

# TÓM TẮT

NÊU MỤC TIÊU NGHIÊN CỨU TRONG GIAI ĐOẠN. PHƯƠNG PHÁP SỬ DỤNG TRONG NGHIÊN CỨU. KẾT QUẢ CỦA NGHIÊN CỨU, ĐÓNG GÓP CỦA NGHIÊN CỨU (NẾU CÓ). THU HẸP PHẠM VI NGHIÊN CỨU CỦA ĐÊ TÀI.

**MỤC LỤC**

# MỞ ĐẦU

## LÝ DO CHỌN ĐỀ TÀI

Lúa là cây trồng chủ lực ở Việt Nam, đóng vai trò quan trọng trong an ninh lương thực và kinh tế. Theo thống kê của FAO (2023), Việt Nam sản xuất khoảng 43 triệu tấn lúa mỗi năm, là một trong những nước xuất khẩu gạo hàng đầu thế giới [1]. Tuy nhiên, các bệnh trên cây lúa, đặc biệt trong giai đoạn đẻ nhánh – thời kỳ cây phát triển chồi và lá non, có thể làm giảm năng suất từ 20-50% nếu không được phát hiện và xử lý kịp thời [2]. Các bệnh phổ biến như bạc lá, đạo ôn lá và khô vằn gây thiệt hại lớn, nhưng việc chẩn đoán chủ yếu dựa vào quan sát thủ công, tốn thời gian và kém chính xác.

Sự phát triển của học máy, đặc biệt là các mô hình học sâu như mạng nơ-ron tích chập (CNN), đã mở ra cơ hội để tự động hóa việc nhận dạng bệnh trên cây lúa, giúp cải thiện độ chính xác và tiết kiệm chi phí. Ở Việt Nam, nghiên cứu về nhận dạng bệnh lúa trong giai đoạn đẻ nhánh bằng học máy còn hạn chế, chủ yếu tập trung vào các phương pháp truyền thống. Vì vậy, đề tài này tập trung vào việc áp dụng các mô hình học sâu (DenseNet121, InceptionV3, MobileNetV2, ResNet50) để nhận dạng các bệnh trên lá lúa trong giai đoạn đẻ nhánh, đồng thời so sánh với các phương pháp học máy truyền thống như Random Forest, SVM và kNN, nhằm đề xuất giải pháp hiệu quả cho nông dân.

## MỤC TIÊU VÀ PHẠM VI NGHIÊN CỨU

### Mục tiêu nghiên cứu

* Áp dụng các mô hình học sâu (DenseNet121, InceptionV3, MobileNetV2, ResNet50) để phân loại các bệnh trên lá lúa trong giai đoạn đẻ nhánh, sử dụng tập dữ liệu Rice Leaf Diseases Detection.
* So sánh hiệu suất của các mô hình học sâu với các phương pháp truyền thống (Random Forest, SVM, kNN).
* Phân tích ưu nhược điểm của từng phương pháp trong việc nhận dạng bệnh lúa.

### Phạm vi nghiên cứu

* Tập dữ liệu: Sử dụng tập dữ liệu Rice Leaf Diseases Detection từ Kaggle, giới hạn vào 6 lớp (Bacterial Leaf Blight, Leaf Blast, Leaf Scald, Brown Spot, Sheath Blight, Healthy Rice Leaf), tương ứng với các bệnh phổ biến trong giai đoạn đẻ nhánh [4].
* Mô hình: Áp dụng các mô hình học sâu (DenseNet121, InceptionV3, MobileNetV2, ResNet50) và các mô hình truyền thống (Random Forest, SVM, kNN).
* Ứng dụng: Kết quả nghiên cứu tập trung vào nhận dạng bệnh trong giai đoạn đẻ nhánh, không bao gồm các giai đoạn sinh trưởng khác.

## Ý NGHĨA KHOA HỌC VÀ THỰC TIỄN CỦA ĐỀ TÀI

### Ý nghĩa khoa học

* Ứng dụng học sâu vào nhận dạng bệnh trên cây lúa, góp phần làm phong phú nghiên cứu về AI trong nông nghiệp tại Việt Nam.
* Đánh giá hiệu quả của các mô hình học sâu và truyền thống, cung cấp cơ sở cho các nghiên cứu tiếp theo về tối ưu hóa thuật toán.

### Ý nghĩa thực tiễn:

* Hỗ trợ nông dân phát hiện sớm các bệnh trên cây lúa, giảm thiệt hại năng suất và chi phí quản lý.
* Đề xuất giải pháp tự động hóa chẩn đoán bệnh, dễ triển khai trên các thiết bị đơn giản, phù hợp với nông dân ở vùng nông thôn Việt Nam.

# TỔNG QUAN ĐỀ TÀI

## Các công trình nghiên cứu liên quan

Trong những năm gần đây, trí tuệ nhân tạo (AI) và học máy (Machine Learning) đã trở thành công cụ quan trọng trong nông nghiệp, đặc biệt trong việc nhận dạng và quản lý bệnh trên cây trồng. Nhiều nghiên cứu liên quan đến nhận dạng bệnh trên cây lúa đã đạt được những kết quả đáng chú ý. Chẳng hạn, nghiên cứu của Lu và các cộng sự (2017) đã sử dụng mạng nơ-ron tích chập (CNN) để phân loại các bệnh trên lá lúa, đạt độ chính xác lên đến 95% trên tập dữ liệu tự xây dựng [5]. Tương tự, Zhang và các cộng sự (2019) đã áp dụng các mô hình học sâu như VGG16 và ResNet50 để nhận dạng bệnh đạo ôn và bạc lá trên cây lúa, cho thấy hiệu quả vượt trội so với các phương pháp truyền thống như SVM và Random Forest, với độ chính xác cao hơn trên các tập dữ liệu thử nghiệm [6].

Tuy nhiên, phần lớn các nghiên cứu này tập trung vào dữ liệu tổng quát, không phân tích chi tiết theo từng giai đoạn sinh trưởng cụ thể của cây lúa, chẳng hạn giai đoạn đẻ nhánh – một thời kỳ quan trọng ảnh hưởng lớn đến năng suất cây trồng. Tại Việt Nam, mặc dù đã có nhiều nghiên cứu về bệnh trên cây lúa, nhưng các công trình tập trung vào nhận dạng bệnh trong giai đoạn đẻ nhánh bằng học máy còn rất hạn chế. Theo nghiên cứu của Trần và các cộng sự (2020), các phương pháp hiện tại chủ yếu dựa vào quan sát thủ công hoặc phân tích trong phòng thí nghiệm, đòi hỏi nhiều thời gian và không thể áp dụng đại trà cho nông dân [3]. Điều này tạo cơ hội để áp dụng các mô hình học sâu nhằm cải thiện hiệu quả chẩn đoán bệnh trên cây lúa, đặc biệt trong giai đoạn đẻ nhánh.

## Những vấn đề còn tồn tại

Mặc dù đã có nhiều tiến bộ trong việc ứng dụng AI vào nhận dạng bệnh cây trồng, vẫn còn một số hạn chế:

* **Thiếu hụt dữ liệu hình ảnh trong giai đoạn đẻ nhánh:** Dữ liệu hình ảnh về bệnh trên cây lúa, đặc biệt trong giai đoạn đẻ nhánh, còn khan hiếm và không đồng nhất. Ví dụ, tập dữ liệu công khai như Rice Leaf Diseases Detection trên Kaggle cung cấp nhiều hình ảnh về các loại bệnh trên lá lúa, nhưng không phải tất cả đều phù hợp với giai đoạn đẻ nhánh hoặc có đặc điểm hình thái rõ ràng để nhận diện bệnh một cách chính xác [4].
* **Sự tương đồng về hình thái giữa các bệnh:** Một số bệnh trên lá lúa, chẳng hạn như đốm nâu và đốm nâu hẹp, có hình thái rất giống nhau, dễ gây nhầm lẫn cho các mô hình học máy, dẫn đến giảm độ chính xác trong phân loại. Theo nghiên cứu của Ou (1985), các bệnh này có đặc điểm tương đồng về màu sắc và hình dạng tổn thương, đòi hỏi kỹ thuật nhận dạng tinh vi hơn [7].
* **Yêu cầu chuyên môn cao trong xác thực dữ liệu:** Việc xác định và xác thực dữ liệu hình ảnh bệnh trên cây lúa đòi hỏi kiến thức chuyên môn sâu về bệnh học cây trồng, trong khi các nghiên cứu tại Việt Nam còn thiếu các công cụ tự động hóa để hỗ trợ nông dân nhận diện bệnh sớm, khiến việc ứng dụng thực tế gặp khó khăn [3].
* **Yêu cầu tài nguyên tính toán lớn:** Các mô hình học sâu như ResNet50 hay InceptionV3 thường cần tài nguyên tính toán mạnh mẽ để huấn luyện và triển khai, điều này gây khó khăn khi áp dụng trên các thiết bị di động có cấu hình thấp, vốn rất cần thiết cho nông dân ở vùng nông thôn Việt Nam [6].

## Nội dung nghiên cứu

Nghiên cứu tập trung vào bài toán nhận dạng và phân loại các bệnh trên lá lúa trong giai đoạn đẻ nhánh, sử dụng các mô hình học sâu và so sánh với các phương pháp học máy truyền thống nhằm giải quyết các hạn chế trên.

* **Tìm hiểu bài toán nhận dạng bệnh trên cây lúa trong giai đoạn đẻ nhánh:** Phân tích đặc điểm của các bệnh phổ biến trong giai đoạn này như bạc lá, đạo ôn lá, cháy lá, đốm nâu và khô vằn, đồng thời xác định các đặc trưng hình thái đặc thù của từng bệnh để cải thiện độ chính xác khi thực hiện bài toán nhận dạng.
* **Nghiên cứu các mô hình học sâu:** Tập trung vào các mô hình học sâu như DenseNet121, InceptionV3, MobileNetV2 và ResNet50, phân tích ưu và nhược điểm của chúng trong việc phân loại hình ảnh bệnh trên lá lúa, nhằm lựa chọn mô hình phù hợp nhất cho bài toán này.
* **Xử lý và sử dụng tập dữ liệu:** Sử dụng tập dữ liệu Rice Leaf Diseases Detection từ Kaggle, giới hạn vào 6 lớp bệnh gồm Bacterial Leaf Blight (bạc lá), Leaf Blast (đạo ôn lá), Leaf Scald (cháy lá), Brown Spot (đốm nâu), Sheath Blight (khô vằn) và Healthy Rice Leaf (lá khỏe) [4].
* **Tăng cường dữ liệu và tối ưu hóa mô hình:** Áp dụng các kỹ thuật tiền xử lý dữ liệu như data augmentation (tăng cường dữ liệu) để tạo thêm hình ảnh mới, giúp mô hình học tốt hơn với dữ liệu hạn chế, đồng thời tinh chỉnh các siêu tham số như tốc độ học (learning rate) để tối ưu hóa hiệu suất của mô hình.
* **Đánh giá và so sánh hiệu suất:** Đánh giá hiệu quả của các mô hình dựa trên các chỉ số như độ chính xác (Accuracy), độ chính xác từng lớp (Precision), độ bao phủ (Recall) và chỉ số F1-score, từ đó so sánh hiệu suất giữa các mô hình học sâu và các phương pháp học máy truyền thống để tìm ra giải pháp tối ưu nhất cho bài toán nhận dạng bệnh trên cây lúa trong giai đoạn đẻ nhánh.

**Lý do lựa chọn 6 lớp dữ liệu**

Tập dữ liệu Rice Leaf Diseases Detection bao gồm 9 lớp, nhưng nghiên cứu này chọn 6 lớp (Bacterial Leaf Blight, Leaf Blast, Leaf Scald, Brown Spot, Sheath Blight, Healthy Rice Leaf) với 7.537 hình ảnh, dựa trên các tiêu chí sau:

* **Phổ biến trong giai đoạn đẻ nhánh**: Các bệnh như đạo ôn lá, khô vằn và bạc lá là những bệnh chính trong giai đoạn đẻ nhánh, có thể gây thiệt hại năng suất 20-50% [2]. Đốm nâu và cháy lá cũng thường xuất hiện trên lá non trong giai đoạn này [7].
* **Phù hợp với giai đoạn đẻ nhánh**: Các bệnh được chọn đều ảnh hưởng đến lá hoặc bẹ lá trong giai đoạn đẻ nhánh, trong khi lớp lá khỏe mạnh giúp mô hình nhận diện trạng thái không bệnh [2].
* **Giảm nhầm lẫn và độ phức tạp**: Các lớp bị loại (Narrow Brown Leaf Spot, Neck Blast, Rice Hispa) ít liên quan đến giai đoạn đẻ nhánh hoặc dễ gây nhầm lẫn. Ví dụ, Neck Blast chủ yếu xuất hiện ở giai đoạn trỗ, còn Rice Hispa là tổn thương do côn trùng [8].

# PHƯƠNG PHÁP VÀ NỘI DUNG NGHIÊN CỨU

## Cơ sở lý thuyết

### Tập dữ liệu Rice Leaf Diseases Detection [4]

Trong nghiên cứu này, tập dữ liệu Rice Leaf Diseases Detection được lấy từ Kaggle, bao gồm 9.790 hình ảnh lá lúa chụp bằng máy ảnh kỹ thuật số độ phân giải cao trong điều kiện ánh sáng tự nhiên. Các hình ảnh được thu thập từ các cánh đồng lúa, đảm bảo tính chân thực và rõ nét, phù hợp để phân tích và nhận dạng bệnh bằng các mô hình học sâu. Trong nghiên cứu này, tập dữ liệu được giới hạn vào 6 lớp (Bacterial Leaf Blight, Leaf Blast, Leaf Scald, Brown Spot, Sheath Blight, Healthy Rice Leaf) với tổng cộng 7.537 hình ảnh, được chọn dựa trên tính phổ biến, đặc điểm hình thái rõ ràng và sự phù hợp với giai đoạn đẻ nhánh. Có 6 lớp được sử dụng để phân loại, cụ thể như sau:

* **Bacterial Leaf Blight** (Bạc lá) (1.197 hình ảnh): Bệnh bạc lá do vi khuẩn Xanthomonas oryzae gây ra, là một trong những bệnh nghiêm trọng trên cây lúa, đặc biệt trong giai đoạn đẻ nhánh ở vùng nhiệt đới ẩm ướt. Bệnh biểu hiện qua các vết thấm nước trên mép lá, sau đó chuyển thành màu xám trắng hoặc vàng nhạt, với viền rõ ràng. Các vết này có thể lan rộng, gây héo lá và giảm khả năng quang hợp. Theo Ou (1985), bệnh bạc lá thường xuất hiện trên lá non, làm giảm số lượng chồi hữu hiệu nếu không được kiểm soát [7].
* **Leaf Blast** (Đạo ôn lá) (1.748 hình ảnh): Bệnh đạo ôn lá do nấm Magnaporthe oryzae gây ra, là bệnh phá hoại nhất trên lúa đặc biệt trong giai đoạn đẻ nhánh khi lá non dễ bị nhiễm. Ban đầu, bệnh xuất hiện dưới dạng các vết thấm nước nhỏ, sau đó mở rộng thành các vết đạo ôn lớn màu xám hoặc trắng, đôi khi có viền nâu. Các vết này có thể gây chết lá, ảnh hưởng nghiêm trọng đến sự phát triển của cây. IRRI (2020) ghi nhận rằng bệnh đạo ôn lá có thể làm giảm năng suất lên đến 50% trong điều kiện thuận lợi [2].
* **Leaf Scald** (Cháy lá) (1.332 hình ảnh): Bệnh cháy lá, do vi khuẩn Rhynchosporium oryzae gây ra, thường xuất hiện trên lá trong giai đoạn đẻ nhánh, đặc biệt ở vùng khí hậu ẩm. Bệnh được nhận diện bởi các vết thương kéo dài, màu vàng nhạt hoặc xám, thường xuất hiện ở mép lá. Các vết này có thể làm giảm diện tích lá quang hợp, ảnh hưởng đến sự phát triển chồi. Theo Mew (1993), bệnh cháy lá phổ biến ở các vùng nhiệt đới và có thể gây thiệt hại đáng kể nếu cây bị stress [9].
* **Brown Spot** (Đốm nâu) (1.546 hình ảnh): Bệnh đốm nâu do nấm Cochliobolus miyabeanus gây ra, thường xuất hiện trong giai đoạn đẻ nhánh, đặc biệt khi cây lúa thiếu dinh dưỡng hoặc gặp điều kiện ẩm ướt. Bệnh đặc trưng bởi các vết đốm nhỏ, màu nâu sẫm với quầng vàng xung quanh, có thể lan rộng và gây tổn thương mô lá. Các vết đốm này làm giảm khả năng quang hợp và sức khỏe cây. Ou (1985) chỉ ra rằng bệnh đốm nâu là một trong những bệnh phổ biến ở các vùng trồng lúa châu Á [7].
* **Sheath Blight** (Khô vằn) (1.629 hình ảnh): Bệnh khô vằn do nấm Rhizoctonia solani gây ra, ảnh hưởng chủ yếu đến bẹ lá và thân trong giai đoạn đẻ nhánh. Bệnh biểu hiện qua các vết xám trắng với viền nâu trên bẹ lá, có thể lan lên lá và gây héo. Trong trường hợp nghiêm trọng, bệnh dẫn đến đổ ngã, làm giảm năng suất. Theo IRRI (2020), khô vằn là một trong những bệnh quan trọng ở các vùng trồng lúa nhiệt đới, đặc biệt trong điều kiện ẩm cao [2].
* **Healthy Rice Leaf** (Lá lúa khỏe mạnh) (1.085 hình ảnh): Lớp lá lúa khỏe mạnh bao gồm các hình ảnh lá không bị ảnh hưởng bởi bệnh, có màu xanh đồng đều, không có vết đốm, vết thấm nước, hay dấu hiệu héo. Đây là lớp chuẩn so sánh, giúp mô hình nhận diện các đặc điểm của lá lúa trong trạng thái bình thường ở giai đoạn đẻ nhánh. Sự hiện diện của lớp này rất quan trọng để phân biệt giữa trạng thái khỏe mạnh và các trạng thái bệnh lý, đảm bảo độ chính xác của mô hình [10].

### Các công nghệ và thư viện sử dụng

Bài nghiên cứu sử dụng các công nghệ và thư viện sau để triển khai, huấn luyện và đánh giá các mô hình nhận dạng bệnh trên lá lúa trong giai đoạn đẻ nhánh. Dưới đây là định nghĩa và các tính năng chính của từng công nghệ và thư viện:

**Keras**: Là một API mã nguồn mở, cấp cao, được viết bằng Python, dùng để xây dựng và huấn luyện các mô hình học sâu. Nó cung cấp giao diện đơn giản, tích hợp với TensorFlow để hỗ trợ phát triển nhanh các mạng nơ-ron [11]. Cung cấp các mô hình học sâu như DenseNet121, InceptionV3, MobileNetV2, ResNet50 để phân loại hình ảnh. Hỗ trợ tiền xử lý hình ảnh, bao gồm tăng cường dữ liệu (data augmentation) như xoay, lật, hoặc thay đổi độ sáng. Cho phép tùy chỉnh kiến trúc mạng, huấn luyện mô hình trên GPU và lưu trữ trọng số.

**Scikit-learn**: Là một thư viện học máy mã nguồn mở, được viết bằng Python, tích hợp nhiều thuật toán học máy truyền thống và các công cụ xử lý dữ liệu [12]. Hỗ trợ các thuật toán như Random Forest, SVM, kNN để phân loại và đánh giá mô hình. Cung cấp các công cụ tiền xử lý (chuẩn hóa, chia tập dữ liệu), trích xuất đặc trưng và tinh chỉnh siêu tham số. Tích hợp các chỉ số đánh giá như Accuracy, Precision, Recall, F1-score.

**OpenCV** (Open Source Computer Vision Library): Là một thư viện mã nguồn mở, được viết bằng C++ và có giao diện Python, chuyên về xử lý hình ảnh và thị giác máy tính [13]. Hỗ trợ đọc, ghi và xử lý hình ảnh (thay đổi kích thước, chuyển đổi kênh màu, chuẩn hóa). Cung cấp các kỹ thuật tăng cường dữ liệu và trích xuất đặc trưng hình ảnh. Tối ưu hóa hiệu suất xử lý trên CPU và GPU.

**Google Drive**: Là một dịch vụ lưu trữ đám mây của Google, cho phép lưu trữ, chia sẻ và truy cập dữ liệu từ mọi thiết bị có kết nối internet [14]. Lưu trữ tập dữ liệu, mô hình đã huấn luyện và kết quả thí nghiệm một cách an toàn. Hỗ trợ chia sẻ dữ liệu dễ dàng và tích hợp trực tiếp với Google Colab. Đảm bảo truy cập nhanh vào dữ liệu từ môi trường đám mây.

**Google Colab**: Là một nền tảng tính toán trực tuyến miễn phí, cung cấp môi trường Jupyter Notebook để viết và chạy mã Python, tích hợp tài nguyên GPU/TPU [15]. Hỗ trợ chạy các thư viện như Keras, Scikit-learn và OpenCV để xây dựng mô hình học sâu. Tích hợp với Google Drive để truy cập dữ liệu và lưu kết quả. Cho phép thực hiện các thí nghiệm mà không cần máy tính cấu hình cao.

**Transfer Learning** (Học chuyển giao): Là kỹ thuật học máy, trong đó mô hình đã được huấn luyện trước trên tập dữ liệu lớn (như ImageNet) được tái sử dụng cho một bài toán mới với tập dữ liệu nhỏ hơn [16]. Tận dụng các trọng số đã huấn luyện để giảm thời gian và chi phí tính toán. Cải thiện hiệu suất trên tập dữ liệu nhỏ, như Rice Leaf Diseases Detection. Phù hợp với các mô hình phức tạp như DenseNet121, ResNet50. Trong nghiên cứu này, transfer learning được áp dụng để sử dụng các mô hình đã huấn luyện trước, sau đó tinh chỉnh cho bài toán nhận dạng bệnh lúa.

**Fine-tune** (Tinh chỉnh): Là quá trình điều chỉnh một mô hình đã được huấn luyện trước bằng cách huấn luyện lại các lớp cuối trên tập dữ liệu cụ thể, nhằm tối ưu hóa cho bài toán mới [17]. Tùy chỉnh mô hình để phù hợp với số lớp đầu ra (6 lớp bệnh lúa trong nghiên cứu này). Cải thiện độ chính xác bằng cách học các đặc trưng đặc thù của tập dữ liệu. Kết hợp giữ lại kiến thức chung và học thêm kiến thức mới. Trong nghiên cứu này, fine-tune được sử dụng để tinh chỉnh các mô hình học sâu trên tập dữ liệu Rice Leaf Diseases Detection, đảm bảo hiệu quả trong phân loại bệnh lúa.

## Mô hình CNN

### Định nghĩa CNN

Convolutional Neural Network (Mạng Nơ-ron Tích Chập) là mạng nơ-ron nhân tạo thường được sử dụng trong xử lý dữ liệu ảnh hoặc dữ liệu có cấu trúc lưới nhờ khả năng nhận diện và phân tích các đặc điểm quan trọng trong ảnh.

### Cấu trúc mạng CNN

Cấu trúc của mạng CNN bao gồm nhiều lớp được sắp xếp tuần tự để xử lý dữ liệu từ đầu vào đến đầu ra, thường được chia thành hai giai đoạn chính: trích xuất đặc trưng và phân loại. Các thành phần chính của CNN bao gồm lớp đầu vào, tích chập, kích hoạt, gộp, chuẩn hóa, dropout, kết nối đầy đủ và đầu ra; chúng được thiết kế để học các đặc trưng từ hình ảnh và đưa ra dự đoán phân loại ví dụ như nhận diện bệnh trên lá lúa trong bài nghiên cứu này.

*Hình 2.1: Cấu trúc cơ bản của CNN*

* **Input Layer** (Lớp đầu vào): Nhận hình ảnh với kích thước cố định, được biểu diễn dưới dạng (W, H, C), trong đó W là chiều rộng, H là chiều cao, C là số kênh màu (thường là 3 với ảnh RGB hoặc 1 với ảnh xám). Một số kích thước đầu vào phổ biến bao gồm 224x224x3, 299x299x3, 32x32x3, 28x28x1, tùy thuộc vào đặc trưng của bài toán và mô hình sử dụng. Lớp này đóng vai trò chuẩn hóa dữ liệu đầu vào để đảm bảo tính nhất quán trong quá trình xử lý.
* **Convolution Layer** (Lớp tích chập): Là thành phần cốt lõi của CNN, chịu trách nhiệm trích xuất đặc trưng từ hình ảnh đầu vào chẳng hạn như cạnh, góc, kết cấu. Lớp này sử dụng các bộ lọc kernel là ma trận nhỏ, thường có kích thước 3x3 hoặc 5x5 để trượt trên hình ảnh và thực hiện phép tích chập, tạo ra bản đồ đặc trưng (feature map). Quá trình trượt được điều chỉnh bởi hai siêu tham số:
  + **Stride** (bước nhảy): Là số pixel mà bộ lọc di chuyển sau mỗi bước, thường là 1 hoặc 2.
  + **Padding** (đệm): Được thêm vào viền ảnh để xử lý các cạnh, đặc biệt khi số pixel không chia hết cho kích thước bộ lọc, giúp giữ kích thước đầu ra không bị giảm quá nhiều.
* **Activation Layer** (Lớp kích hoạt): Được đặt sau lớp tích chập để tăng tính phi tuyến, giúp mạng học được các đặc trưng phức tạp hơn. Các hàm kích hoạt phổ biến bao gồm ReLU (Rectified Linear Unit), Sigmoid và Tanh. Trong đó, ReLU thường được sử dụng vì tính đơn giản và hiệu quả; chuyển các giá trị âm về 0 và giữ nguyên giá trị dương; tăng tốc độ hội tụ của mô hình. Lớp kích hoạt cũng có thể được áp dụng sau lớp kết nối đầy đủ (Fully Connected Layer - Dense Layer) để tối ưu hóa đầu ra.
* **Pooling Layer** (Lớp gộp): Được sử dụng để giảm kích thước không gian của bản đồ đặc trưng từ các lớp tích chập, giúp giảm số lượng tham số, tăng khả năng tổng quát hóa và tránh hiện tượng quá khớp. Lớp này sử dụng một cửa sổ (window), thường có kích thước *k* x *k* (ví dụ: 2x2) để trượt trên bản đồ đặc trưng và thực hiện phép tính giảm chiều. Các loại gộp phổ biến bao gồm:
  + **Max Pooling**: Lấy giá trị lớn nhất trong mỗi vùng cửa sổ, giữ lại các đặc trưng nổi bật. Ví dụ, với cửa sổ 2x2 và stride 2, một bản đồ đặc trưng 224x224 sẽ giảm xuống 112x112.
  + **Average Pooling**: Tính giá trị trung bình trong mỗi vùng cửa sổ, giữ lại thông tin tổng quát.
  + **L2 Pooling**: Tính tổng bình phương các giá trị trong cửa sổ, sau đó lấy căn bậc hai, ít phổ biến hơn nhưng có thể được sử dụng để cân bằng giữa max và average pooling.
  + **Global Max Pooling**: Đặt kích thước cửa sổ bằng kích thước đầu vào, lấy giá trị lớn nhất toàn cục trên mỗi bản đồ đặc trưng. Ví dụ, với đầu vào 7x7x512, global max pooling tạo ra vector 1x1x512, giữ lại giá trị nổi bật nhất từ mỗi kênh.
  + **Global Average Pooling**: Tương tự Global Max Pooling, nhưng lấy giá trị trung bình toàn cục, tạo ra vector 1 chiều với thông tin tổng quát hơn.
* **Batch Normalization/ Layer Normalization** (Lớp chuẩn hóa): Được sử dụng để ổn định quá trình huấn luyện và tăng tốc độ hội tụ của mô hình. Có hai loại chuẩn hóa chính bao gồm:
  + **Batch Normalization:** Chuẩn hóa dữ liệu bằng cách điều chỉnh các giá trị trong mỗi lớp ẩn sao cho chúng có giá trị trung bình gần bằng 0 và độ lệch chuẩn gần bằng 1; làm giảm hiện tượng vanishing gradient (độ lớn của gradient giảm xuống quá thấp, làm chậm quá trình cập nhật trọng số [18]) hoặc exploding gradient (sự gia tăng đột ngột của gradient, dẫn đến việc cập nhật trọng số không ổn định [18]). Điều này giúp huấn luyện ổn định hơn, đặc biệt là với DNN.
  + **Layer Normalization:** Chuẩn hóa dữ liệu theo từng nơ-ron, không phụ thuộc vào kích thước batch, thường được sử dụng trong các mô hình như RNN, nhưng cũng có thể áp dụng trong CNN.
* **Dropout Layer** (Lớp Dropout): Được sử dụng để giảm “overfitting” (nguy cơ quá khớp) trong quá trình huấn luyện bằng cách ngắt ngẫu nhiên một số nơ-ron với dropout rate cho trước, thường từ 0.2 đến 0.5. Dropout buộc mô hình phải học các đặc trưng từ các tổ hợp nơ-ron khác nhau, làm tăng khả năng tổng quát hóa trên dữ liệu mới; giúp mô hình không bị phụ thuộc quá nhiều vào các đặc trưng cụ thể của tập huấn luyện và đảm bảo hiệu quả trên các dữ liệu thực tiễn như trong bài toán phân loại hình ảnh.
* **Fully Connected Layer** **- Dense Layer** (Lớp kết nối đầy đủ): Thường được đặt ở giai đoạn cuối của mạng, nó kết nối với tất cả các nơ-ron của các lớp trước đó để tổng hợp các đặc trưng đã trích xuất để đưa ra quyết định.
* **Output Layer** (Lớp đầu ra): Trả về kết quả cuối cùng của mô hình, tùy vào loại bài toán mà sẽ cho ra kết quả khác nhau, gồm:
  + **Phân loại nhị phân**: Đầu ra gồm 1 nơ-ron với hàm kích hoạt sigmoid, trả về giá trị trong khoảng [0, 1], biểu thị xác suất của một trong hai lớp.
  + **Phân loại đa lớp**: Đầu ra gồm *n* nơ-ron, tương ứng với *n* lớp cần phân loại, sử dụng hàm softmax để tạo ra xác suất cho mỗi lớp với tổng xác suất bằng 1.
  + **Hồi quy**: Đầu ra là một nơ-ron, trả về giá trị liên tục, không cần hàm kích hoạt.

### Mô hình Resnet50 [19]

*Hình 2.2: Cấu trúc mô hình ResNet50*

ResNet50 là một kiến trúc mạng nơ-ron tích chập (CNN) thuộc họ ResNet (Residual Network), được phát triển để giải quyết vấn đề suy giảm hiệu suất trong các mạng nơ-ron sâu (DNN). Họ ResNet bao gồm nhiều biến thể với độ sâu khác nhau như ResNet18, ResNet34, ResNet101, trong đó ResNet50 là một biến thể có độ sâu trung bình.

Vấn đề chính mà ResNet nhắm đến là khi mạng nơ-ron trở nên quá sâu, độ chính xác đạt đến mức bão hòa và sau đó giảm nhanh chóng. Sự suy giảm này không phải do overfitting, mà do khó khăn trong việc tối ưu hóa quá trình huấn luyện, đặc biệt là vấn đề vanishing gradient.

ResNet50 bao gồm 50 tầng, bao gồm 48 lớp tích chập, 1 lớp gộp cực đại và 1 lớp gộp trung bình trước khi đi qua lớp kết nối đầy đủ. Với lớp đầu vào là hình ảnh kích thước 224x224x3 (3 kênh màu RGB). Tiếp theo là tiền xử lý với lớp tích chập 7x7 và lớp gộp cực đại để giảm kích thước và chiết xuất đặc trưng cơ bản, đồng thời giữ lại các thông tin quan trọng cho các khối tiếp theo. Khối dư (Residual Block) mỗi khối dư bao gồm tích chập 1x1 (giảm số kênh để giảm chi phí tính toán), tích chập 3x3 (học các đặc trưng không gian), tích chập 1x1 (khôi phục số kênh); khối này sử dụng Batch Normalization để chuẩn hóa đầu ra của các lớp tích chập để ổn định huấn luyện; sử dụng hàm kích hoạt ReLU để thêm tính phi tuyến tính, giúp mô hình học các đặc trưng phức tạp; với Residual Connection, trong đầu vào của khối được cộng trực tiếp với khối đầu ra của khối trước khi đi qua ReLU, theo công thức *y = F(x) + x* , giúp mạng tập trung học các đặc trưng mới mà không làm mất đặc trưng từ các lớp trước và đồng thời giúp làm giảm vấn đề biến mất gradient. Cuối cùng là lớp cuối, bao gồm lớp gộp trung bình toàn cục làm giảm kích thước xuống 1x1x2048, sau đó là lớp kết nối đầy đủ để dự đoán lớp.

Resnet50 có ưu điểm là có khả năng học tốt trên DNN và làm giảm vấn đề biến mất gradient, nhược điểm là nặng đồng thời là cần nhiều tài nguyên để huấn luyện, không phù hợp cho thiết bị di động.

### Mô hình MobileNetV2 [20]

*Hình 2.3: Cấu trúc mô hình MobileNetV2*

MobileNetV2 là một kiến trúc mạng nơ-ron tích chập (CNN) thuộc họ MobileNet, được phát triển để giải quyết vấn đề chi phí tính toán cao trong các mạng nơ-ron sâu (DNN) khi triển khai trên thiết bị di động. Họ MobileNet bao gồm nhiều biến thể như MobileNetV1, MobileNetV3, trong đó MobileNetV2 là một biến thể cải tiến với hiệu suất cao hơn.

Vấn đề chính mà MobileNetV2 nhắm đến là các mạng nơ-ron sâu thường nặng và tiêu tốn nhiều tài nguyên, không phù hợp cho các thiết bị di động có bộ nhớ và năng lượng hạn chế. MobileNetV2 sử dụng tích chập tách biệt chiều sâu và khối nghịch đảo dư để giảm chi phí tính toán.

MobileNetV2 bao gồm 53 tầng, với 17 khối nghịch đảo dư (inverted residual blocks), xen kẽ với các tầng tích chập và gộp. Với lớp đầu vào là hình ảnh kích thước 224x224x3 (3 kênh màu RGB). Tiếp theo là tiền xử lý với lớp tích chập 3x3 để giảm kích thước và chiết xuất đặc trưng cơ bản, đồng thời giữ lại các thông tin quan trọng cho các khối tiếp theo. Khối nghịch đảo dư (Inverted Residual Block) mỗi khối bao gồm tích chập 1x1 (mở rộng số kênh, thường gấp 6 lần), tích chập chiều sâu 3x3 (học đặc trưng không gian, áp dụng bộ lọc 3x3 riêng cho từng kênh), tích chập 1x1 (thu hẹp số kênh, không dùng ReLU để tránh mất thông tin); khối này không sử dụng Batch Normalization hay ReLU ở bước cuối để giữ thông tin; với kết nối tắt, trong đó đầu vào của khối được cộng trực tiếp với đầu ra nếu số kênh khớp, giúp tăng hiệu quả học. Cuối cùng là lớp cuối, bao gồm lớp tích chập 1x1 (tăng số kênh lên 1280), lớp gộp trung bình toàn cục làm giảm kích thước xuống 1x1x1280, sau đó là lớp kết nối đầy đủ để dự đoán lớp.

MobileNetV2 có ưu điểm là nhẹ và nhanh, phù hợp cho thiết bị di động, nhược điểm là hiệu suất thấp trên dữ liệu phức tạp và học đặc trưng yếu.

### Mô hình InceptionV3 [21]

*Hình 2.4: Cấu trúc mô hình InceptionV3*

InceptionV3 là một kiến trúc mạng nơ-ron tích chập (CNN) thuộc họ Inception (hay GoogLeNet), được phát triển để giải quyết vấn đề chi phí tính toán cao và hiệu suất trên dữ liệu phức tạp trong các mạng nơ-ron sâu (DNN). Họ Inception bao gồm nhiều biến thể như InceptionV1, InceptionV2, trong đó InceptionV3 là một biến thể cải tiến với hiệu quả cao hơn.

Vấn đề chính mà InceptionV3 nhắm đến là làm thế nào để chiết xuất đặc trưng ở nhiều tỷ lệ khác nhau mà không làm tăng quá nhiều chi phí tính toán, đồng thời cải thiện hiệu suất trên các tập dữ liệu lớn như ImageNet. InceptionV3 sử dụng các mô-đun Inception để xử lý nhiều kích thước bộ lọc cùng lúc.

InceptionV3 bao gồm 48 tầng, với các mô-đun Inception là thành phần chính, xen kẽ với các tầng gộp. Với lớp đầu vào là hình ảnh kích thước 299x299x3 (3 kênh màu RGB). Tiếp theo là tiền xử lý với các tầng tích chập 3x3 và gộp cực đại để giảm kích thước và chiết xuất đặc trưng cơ bản, đồng thời giữ lại các thông tin quan trọng cho các khối tiếp theo. Mô-đun Inception mỗi mô-đun gồm 4 nhánh xử lý song song: nhánh 1 với tích chập 1x1 (chiết xuất đặc trưng đơn giản), nhánh 2 với tích chập 1x1 và tích chập 3x3 (học đặc trưng lớn hơn), nhánh 3 với tích chập 1x1 và hai tích chập 3x3 (thay cho 5x5, học đặc trưng phức tạp), nhánh 4 với gộp cực đại 3x3 và tích chập 1x1 (giữ thông tin không gian); các nhánh này được nối (concatenate) trên chiều kênh để tạo đầu ra. Cuối cùng là lớp cuối, bao gồm lớp gộp trung bình toàn cục làm giảm kích thước xuống 1x1x2048, sau đó là lớp kết nối đầy đủ để dự đoán lớp.

InceptionV3 có ưu điểm là hiệu quả cao trên dữ liệu lớn và học đặc trưng đa dạng, nhược điểm là kiến trúc phức tạp và khó triển khai trên thiết bị yếu.

### Mô hình DenseNet121 [22]

*Hình 2.5: Cấu trúc mô hình DenseNet121*

DenseNet121 là một kiến trúc CNN thuộc họ DenseNet (Densely Connected Convolutional Networks), được thiết kế để cải thiện hiệu quả của mạng nơ-ron bằng cách tăng cường luồng thông tin giữa các tầng. Họ DenseNet có nhiều biến thể như DenseNet169, DenseNet201, trong đó DenseNet121 là phiên bản có độ sâu trung bình.

Vấn đề chính mà DenseNet giải quyết là sự mất mát thông tin trong các mạng sâu, cũng như số lượng tham số lớn. DenseNet sử dụng kết nối dày đặc (dense connectivity) để tái sử dụng đặc trưng và giảm số tham số.

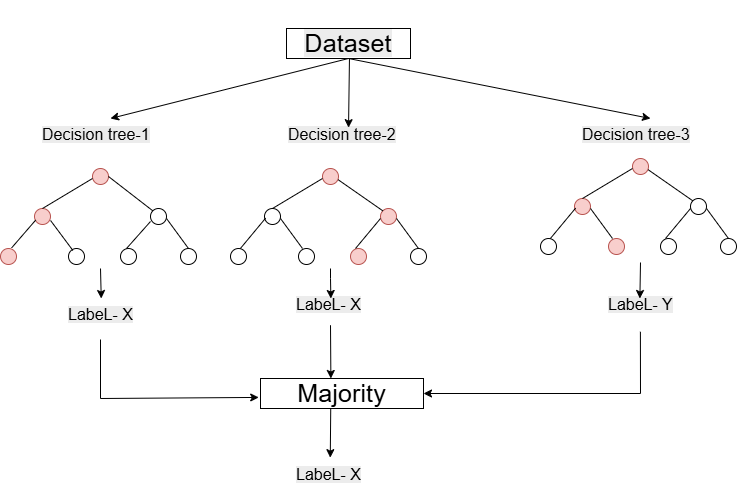
DenseNet121 bao gồm 121 tầng, chia thành 4 khối dày đặc (dense blocks), xen kẽ với các tầng chuyển tiếp (transition layers). Với lớp đầu vào là hình ảnh kích thước 224x224x3 (3 kênh màu RGB). Tiếp theo là tiền xử lý với lớp tích chập 7x7 và lớp gộp cực đại để giảm kích thước và chiết xuất đặc trưng cơ bản, đồng thời giữ lại các thông tin quan trọng cho các khối tiếp theo. Khối dày đặc (Dense Block) mỗi tầng trong khối nhận đầu vào từ tất cả các tầng trước đó thông qua phép nối (concatenation); mỗi tầng bao gồm tích chập 1x1 (giảm số kênh để giảm chi phí tính toán), tích chập 3x3 (học các đặc trưng không gian); khối này sử dụng Batch Normalization để chuẩn hóa đầu ra của các lớp tích chập để ổn định huấn luyện; sử dụng hàm kích hoạt ReLU để thêm tính phi tuyến tính, giúp mô hình học các đặc trưng phức tạp; với kết nối dày đặc, giúp tái sử dụng đặc trưng từ các tầng trước, giảm số tham số và tăng hiệu quả học. Tầng chuyển tiếp (Transition Layer) bao gồm tích chập 1x1 (giảm số kênh) và gộp trung bình (2x2, stride 2) để giảm kích thước không gian. Cuối cùng là lớp cuối, bao gồm lớp gộp trung bình toàn cục làm giảm kích thước xuống 1x1x1024, sau đó là lớp kết nối đầy đủ để dự đoán lớp.

DenseNet121 có ưu điểm là ít tham số hơn ResNet và học hiệu quả nhờ tái sử dụng đặc trưng, nhược điểm là cần nhiều bộ nhớ và tốc độ chậm trên phần cứng yếu.

## Random Forest [23]

### Định nghĩa Random Forest

Random Forest (Rừng Ngẫu Nhiên) là một mô hình phân loại bao gồm nhiều cây quyết định, được xây dựng trên các tập con khác nhau của tập dữ liệu ban đầu, và sử dụng phương pháp trung bình để cải thiện độ chính xác dự đoán. RF có thể áp dụng cho cả các vấn đề phân loại và hồi quy. Trong khi một cây quyết định đơn lẻ chỉ kiểm thử một cây, RF sử dụng nhiều cây quyết định trong giai đoạn kiểm thử, làm cho nó trở thành một lựa chọn ưu việt hơn so với cây quyết định đơn lẻ.



*Hình 2.6: Cấu trúc Random Forest*

### Cách hoạt động của thuật toán

**Bước 1:** Chuẩn bị dữ liệu và tạo các tập con

* Chuẩn bị tập dữ liệu lớn, ví dụ như thông tin khách hàng bao gồm tuổi, thu nhập và nhãn “có mua máy tính” hoặc “không mua”.
* Random Forest sẽ lấy ngẫu nhiên một số mẫu từ tập dữ liệu này để tạo ra nhiều tập con. Mỗi tập con có số lượng mẫu bằng tập dữ liệu gốc nhưng các mẫu được chọn ngẫu nhiên và có thể lặp lại.

**Bước 2:** Xây dựng từng cây quyết định

* Với mỗi tập con vừa tạo, Random Forest sẽ xây dựng một cây quyết định. Cây quyết định giống như một sơ đồ câu hỏi, ví dụ “tuổi có lơn hơn 30 không?”, ếu có thì đi tiếp, nếu không thì rẽ nhánh khác.
* Để tránh các cây giống nhau, tại mỗi câu hỏi, thuật toán chỉ chọn ngẫu nhiên một số đặc trưng (ví dụ: chỉ chọn tuổi và thu nhập, không chọn học vấn) để tìm câu hỏi tốt nhất.
* Cây sẽ tiếp tục phát triển bằng cách đặt các câu hỏi cho đến khi không thể chia nhỏ thêm, ví dụ khi tất cả khách hàng trong nhánh đều “có mua” hoặc “không mua”.

**Bước 3:** Để từng cây đưa ra dự đoán

* Sau khi xây dựng xong, ví dụ bạn có 100 cây, bạn sẽ dùng chúng để dự đoán cho một khách hàng mới, chẳng hạn một người 25 tuổi, thu nhập trung bình. Mỗi cây sẽ đưa ra dự đoán riêng.
* Với bài toán phân loại, một cây có thể nói “có mua”, cây khác có thể nói “không mua”.

**Bước 4:** Kết hợp dự đoán từ tất cả các cây

* Random Forest sẽ tổng hợp kết quả từ tất cả các cây. Nếu bạn đang làm bài toán phân loại, nó sẽ đếm xem có bao nhiêu cây chọn “có mua” và bao nhiêu cây chọn “không mua”.

**Bước 5:** Kiểm tra độ chính xác của mô hình

* Để xem mô hình có tốt không, Random Forest sẽ dùng các mẫu OOB (những khách hàng không được chọn trong tập con của mỗi cây) làm tập kiểm tra. Mỗi cây sẽ dự đoán trên các mẫu OOB mà nó không dùng để học, sau đó tổng hợp kết quả để xem mô hình dự đoán đúng bao nhiêu lần.

### Ưu và nhược điểm

**Ưu điểm:**

* **Dự đoán chính xác và ít bị ảnh hưởng bởi dữ liệu nhiễu**: Random Forest sử dụng nhiều cây quyết định để đưa ra dự đoán và kết quả cuối cùng được tổng hợp từ tất cả các cây thông qua majority voting hoặc lấy trung bình. Điều này giúp giảm thiểu tác động của các điểm dữ liệu bất thường (nhiễu).
* **Hoạt động tốt trên dữ liệu lớn và phức tạp**: Vì Random Forest chia tập dữ liệu thành nhiều tập con và xây dựng nhiều cây quyết định, nó có thể xử lý dữ liệu có nhiều đặc trưng (features) và mẫu (samples) mà không bị quá tải.

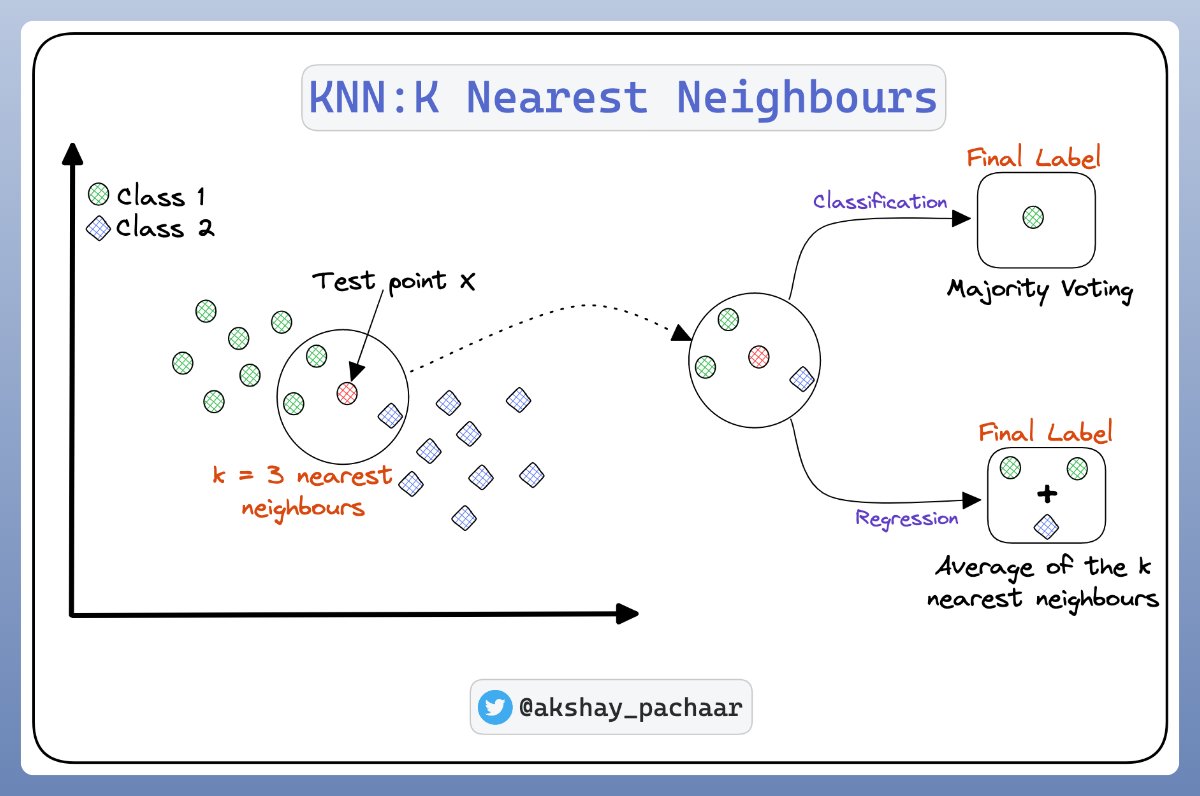
**Nhược điểm:**

* **Tốn nhiều thời gian và tài nguyên để xử lý:** Random Forest phải xây dựng và lưu trữ nhiều cây quyết định, mỗi cây lại cần xử lý một tập dữ liệu con. Điều này đòi hỏi nhiều thời gian tính toán và bộ nhớ.
* **Khó giải thích kết quả:** Vì Random Forest kết hợp dự đoán từ nhiều cây, khó để hiểu tại sao mô hình đưa ra một kết quả cụ thể.

## K-Nearest Neighbors [24]

### Định nghĩa của K-Nearest Neighbors (kNN)

KNN (K-Nearest Neighbors - Hàng xóm gần nhất) là một thuật toán học máy đơn giản, thuộc nhóm phương pháp học lười (lazy learning), được phát triển để giải quyết các bài toán phân loại và hồi quy dựa trên sự tương đồng giữa các điểm dữ liệu. KNN không có các biến thể cụ thể, nhưng có thể thay đổi số lượng hàng xóm gần nhất (gọi là *k*) để điều chỉnh cách hoạt động của mô hình.



*Hình 2.6: Cấu trúc kNN*

### Cách hoạt động của thuật toán

**Bước 1:** Chuẩn bị và lưu trữ dữ liệu huấn luyện

* Chuẩn bị một tập dữ liệu đã được phân loại sẵn, ví dụ như thông tin khách hàng bao gồm tuổi, thu nhập, học vấn và nhãn “có mua máy tính” hoặc “không mua”.
* kNN sẽ lưu trữ toàn bộ tập dữ liệu này để sử dụng sau này. Mỗi khách hàng được xem như một điểm trong không gian, với các đặc điểm như tuổi và thu nhập là tọa độ của điểm đó.

**Bước 2:** So sánh dữ liệu mới với dữ liệu cũ

* Khi có một khách hàng mới, ví dụ một người 25 tuổi, thu nhập trung bình, là sinh viên, kNN sẽ so sánh người này với từng người trong tập dữ liệu cũ.
* Thuật toán sẽ đo khoảng cách giữa khách hàng mới và từng khách hàng cũ, giống như cách bạn đo khoảng cách giữa hai điểm trên bản đồ, dựa trên các đặc điểm như tuổi, thu nhập, học vấn. Người nào càng giống khách hàng mới (khoảng cách nhỏ) thì được coi là “hàng xóm gần”.

**Bước 3:** Chọn số lượng hàng xóm gần nhất

* Sau khi đo khoảng cách, kNN sẽ sắp xếp tất cả khách hàng cũ từ gần đến xa so với khách hàng mới, rồi chọn một số lượng hàng xóm gần nhất, gọi là *k*.
* Ví dụ: nếu bạn chọn *k* = 3, kNN sẽ lấy 3 người có đặc điểm giống khách hàng mới nhất.

**Bước 4:** Dự đoán dựa trên hàng xóm

* kNN sẽ xem 3 người gần nhất này để dự đoán cho khách hàng mới. Nếu bạn đang làm bài toán phân loại, nó sẽ đếm xem trong 3 người, bao nhiêu người “có mua” và bao nhiêu người “không mua”.
* Ví dụ: nếu người 1 và người 2 “có mua”, còn người 3 “không mua”, thì kNN sẽ dự đoán khách hàng mới “có mua” vì đa số (2/3) hàng xóm đã mua.

**Bước 5:** Kiểm tra và điều chỉnh để dự đoán tốt hơn

* Để dự đoán chính xác hơn, có thể ưu tiên những người gần hơn bằng cách cho họ ảnh hưởng lớn hơn đến kết quả.
* Cũng có thể thử thay đổi *k* để xem giá trị nào cho kết quả tốt nhất, vì nếu *k* quá nhỏ dự đoán có thể bị ảnh hưởng bởi nhiễu (noise), nếu *k* quá lớn thì có thể đã bỏ qua các đặc điểm quan trọng.

### Ưu và nhược điểm

**Ưu điểm:**

* **Đơn giản và dễ triển khai:** kNN không cần phải xây dựng mô hình phức tạp trước, chỉ cần lưu trữ dữ liệu và so sánh khi dự đoán.
* **Linh hoạt cho cả phân loại và hồi quy:** kNN có thể được dùng cho nhiều bài toán khác nhau.

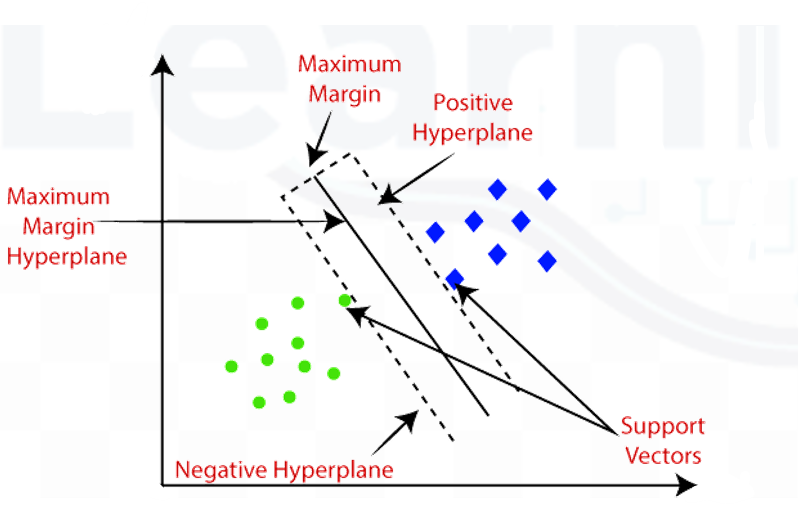
**Nhược điểm:**

* **Chậm khi tập dữ liệu lớn:** Như ví dụ đã được trình bày ở trên thì kNN phải so sánh khách hàng mới với tất cả khách hàng cũ để tìm hàng xóm gần nhất, điều này rất tốn thời gian nếu danh sách khách hàng dài.
* **Dễ bị ảnh hưởng bởi nhiễu trong dữ liệu:** Nếu danh sách khách hàng có thông tin sai lệch.

## Support Vector Machine [25]

### Định nghĩa Support Vector Machine (SVM)

SVM (Support Vector Machine) là một thuật toán học máy mạnh mẽ, thuộc nhóm các phương pháp phân loại và hồi quy dựa trên lề (margin-based methods), được phát triển để giải quyết vấn đề phân tách các lớp dữ liệu một cách hiệu quả.



*Hình 2.6: Cấu trúc SVM*

### Cách hoạt động của thuật toán

**Bước 1:** Chuẩn bị dữ liệu và tìm đường phân tách

* Chuẩn bị một tập dữ liệu đã được phân loại sẵn, ví dụ như thông tin khách hàng bao gồm tuổi, thu nhập và nhãn “có mua máy tính” hoặc “không mua”.
* SVM sẽ cố gắng tìm một đường thẳng (nếu dữ liệu có 2 đặc trưng) hoặc một mặt phẳng (nếu có nhiều đặc trưng hơn) để chia tách hai nhóm khách hàng, ví dụ khách hàng “có mua” nằm bên trái và “không mua” nằm bên phải.

**Bước 2:** Làm cho khoảng cách giữa các nhóm lớn nhất

* Sau khi tìm đường phân tách, SVM sẽ chọn đường có khoảng cách lớn nhất đến các khách hàng gần nhất của mỗi nhóm, gọi là support vectors.

**Bước 3:** Xử lý khi dữ liệu có nhiễu

* Trong thực tế, dữ liệu thường không hoàn hảo, có thể có một số khách hàng bị lẫn lộn, ví dụ một người “có mua” lại nằm giữa nhóm “không mua”. SVM sẽ cho phép một số lỗi như vậy xảy ra, nhưng sẽ cố gắng hạn chế chúng bằng cách phạt các lỗi này, đồng thời vẫn giữ lề lớn nhất có thể.
* Có thể điều chỉnh mức độ nghiêm khắc của việc phạt lỗi, ví dụ nếu muốn ít lỗi hơn, SVM sẽ cố gắng hơn nhưng lề có thể nhỏ lại.

**Bước 4:** Xử lý dữ liệu không phân tách thẳng

* Nếu dữ liệu không thể chia tách bằng đường thẳng, ví dụ khách hàng “có mua” nằm trong một vòng tròn, còn “không mua” nằm ngoài vòng tròn, SVM sẽ dùng một kỹ thuật gọi là kernel trick. Kỹ thuật này giống như việc biến đổi dữ liệu để đưa nó lên một không gian mới, nơi mà dữ liệu có thể được phân tách dễ dàng hơn, ví dụ bằng cách dùng RBF để biến đổi dữ liệu mà không cần vẽ đường thẳng.
* Các loại kernel của SVM gồm:
  + **Linear kernel:** Sử dụng cho dữ liệu tuyến tính.
  + **Polynomial kernel:** Sử dụng cho dữ liệu phức tạp, tạo không gian đa chiều.
  + **RBF** (Radial Basis Function): Phổ biến cho dữ liệu phi tuyến tính.
  + **Sigmoid kernel:** Tương tự như hàm kích hoạt trong mạng neural.

**Bước 5:** Dự đoán và xử lý bài toán đa lớp

* Sau khi tìm được đường phân tách, SVM sẽ dùng nó để dự đoán cho khách hàng mới, ví dụ xem người này nằm ở phía nào của đường phân tách để quyết định “có mua” hay “không mua”.
* Nếu có nhiều hơn hai nhóm, ví dụ “mua máy tính”, “mua điện thoại”, “không mua”, SVM sẽ chia bài toán thành nhiều bài toán nhỏ, mỗi bài toán phân biệt một nhóm với các nhóm còn lại, sau đó chọn nhóm có khả năng cao nhất.

### Ưu và nhược điểm

**Ưu điểm:**

* **Hiệu quả trên dữ liệu phức tạp và số chiều cao:** SVM hoạt động rất tốt khi dữ liệu có nhiều đặc trưng, ví dụ như dữ liệu hình ảnh hoặc dữ liệu gen với hàng nghìn đặc trưng.
* **Linh hoạt với dữ liệu không tuyến tính nhờ kernel trick**: SVM có thể xử lý dữ liệu không phân tách thẳng bằng cách dùng thủ thuật hạt nhân để biến đổi dữ liệu.

**Nhược điểm:**

* Tốn nhiều thời gian để xử lý trên dữ liệu lớn: SVM cần tính toán để tìm đường phân tách tối ưu và quá trình này rất phức tạp nếu danh sách khách hàng dài.
* Khó chọn tham số phù hợp: SVM có nhiều tham số cần điều chỉnh, ví dụ mức độ chấp nhận lỗi hoặc tham số của RBF. Nếu chọn sai, SVM có thể dự đoán không tốt.

## Các Chỉ Số Đánh Giá Hiệu Suất Mô Hình

Để đánh giá hiệu suất của các mô hình học máy (Random Forest, KNN, SVM) và các mô hình học sâu (InceptionV3, DenseNet121, MobileNetV2, ResNet50) trong bài toán nhận dạng bệnh trên cây lúa ở giai đoạn đẻ nhánh, nghiên cứu sử dụng các chỉ số đo lường phổ biến được tính toán từ ma trận nhầm lẫn (confusion matrix) và các biểu đồ trực quan hóa.

### Ma trận nhầm lẫn và các chỉ số đánh giá

**Ma trận nhầm lẫn (Confusion Matrix)**

Ma trận nhầm lẫn là một công cụ quan trọng để đánh giá hiệu suất của các mô hình phân loại, đặc biệt trong bài toán đa lớp như nhận dạng bệnh trên cây lúa. Ma trận này biểu diễn mối quan hệ giữa các giá trị thực tế và giá trị dự đoán của mô hình, bao gồm các thành phần sau:

* **TP (True Positive):** Số lượng mẫu thuộc lớp dương (ví dụ: lá lúa bị bệnh bạc lá) được mô hình dự đoán đúng là dương.
* **TN (True Negative):** Số lượng mẫu thuộc lớp âm (ví dụ: lá lúa khỏe) được mô hình dự đoán đúng là âm.
* **FP (False Positive):** Số lượng mẫu thuộc lớp âm nhưng bị mô hình dự đoán sai là dương.
* **FN (False Negative):** Số lượng mẫu thuộc lớp dương nhưng bị mô hình dự đoán sai là âm.



Hình 1: Confusion Matrix

**Các chỉ số đánh giá**

Dựa trên ma trận nhầm lẫn, các chỉ số sau được sử dụng để đánh giá hiệu suất của mô hình:

* **Accuracy (Độ chính xác):** Đo lường tỷ lệ các dự đoán đúng (bao gồm cả dương và âm) trên tổng số mẫu dữ liệu. Với công thức sau:
* **Precision (Độ chính xác từng lớp)**: Đo lường tỷ lệ dự đoán dương đúng trên tổng số dự đoán dương của mô hình. Chỉ số này đặc biệt quan trọng khi cần giảm thiểu số lượng dự đoán dương sai (FP). Với công thức sau:
* **Recall (Độ bao phủ)**: Đo lường tỷ lệ dự đoán dương đúng trên tổng số mẫu thực tế thuộc lớp dương. Với công thức sau:
* **F1-Score**: Là thước đo trung bình hài hòa giữa Precision và Recall, thể hiện sự cân bằng giữa hai chỉ số này. Với công thức sau:
* **Đường cong ROC** **(Receiver Operating Characteristic):** được vẽ bằng cách biểu diễn tỷ lệ dương thực tế (Recall) so với tỷ lệ dương sai (False Positive Rate): ().
* **AUC (Area Under Curve):** Là diện tích dưới đường cong ROC. AUC đo lường khả năng phân biệt giữa các lớp của mô hình. AUC nằm trong khoảng từ 0 đến 1, và mô hình được xem là tốt khi AUC đạt từ 0.8 trở lên, cho thấy khả năng phân biệt tốt giữa các lớp bệnh và lá khỏe [26].

### Biểu đồ huấn luyện và đánh giá quá trình học

Ngoài các chỉ số trên, nghiên cứu sử dụng các biểu đồ huấn luyện để trực quan hóa hiệu suất của các mô hình học sâu trong quá trình huấn luyện.

**Biểu đồ độ chính xác và độ mất mát**: Sử dụng thư viện Matplotlib để vẽ biểu đồ theo dõi độ chính xác (accuracy) và độ mất mát (loss) trên tập huấn luyện (train) và tập xác thực (validation) qua các epoch. Biểu đồ này giúp trực quan hóa quá trình huấn luyện, từ đó phát hiện các vấn đề như học quá mức (overfitting) hoặc học chưa đủ (underfitting). Ví dụ, nếu độ chính xác trên tập huấn luyện tăng cao nhưng trên tập xác thực giảm, điều này cho thấy mô hình đang học quá mức và không tổng quát hóa tốt trên dữ liệu mới [27].

Các chỉ số và biểu đồ trên sẽ được áp dụng để đánh giá hiệu suất của 7 mô hình (Random Forest, KNN, SVM, InceptionV3, DenseNet121, MobileNetV2, ResNet50) trong bài toán nhận dạng bệnh trên cây lúa ở giai đoạn đẻ nhánh. Việc sử dụng các chỉ số đa dạng giúp đảm bảo đánh giá toàn diện, từ đó lựa chọn mô hình tối ưu nhất cho ứng dụng thực tế.

# GIẢI PHÁP CHO BÀI TOÁN

## Phát biểu bài toán

Cây lúa là một trong những cây trồng quan trọng nhất tại Việt Nam, không chỉ đóng vai trò là nguồn lương thực chính mà còn là sản phẩm xuất khẩu chủ lực, góp phần tạo việc làm và sinh kế cho hàng triệu nông dân. Tuy nhiên, trong quá trình phát triển, cây lúa thường bị ảnh hưởng bởi nhiều loại bệnh khác nhau, đặc biệt trong giai đoạn đẻ nhánh – một giai đoạn quan trọng quyết định năng suất và chất lượng hạt lúa. Các bệnh phổ biến trong giai đoạn này bao gồm bệnh bạc lá (Bacterial Leaf Blight), đạo ôn lá (Leaf Blast), cháy lá (Leaf Scald), đốm nâu (Brown Spot), và khô vằn (Sheath Blight).

Bài toán đặt ra là xây dựng một mô hình học máy có khả năng nhận dạng và phân loại chính xác các loại bệnh trên lá lúa trong giai đoạn đẻ nhánh thông qua hình ảnh, sử dụng tập dữ liệu hình ảnh được thu thập từ nhiều quốc gia trên Kaggle (Rice Leaf Diseases Detection Dataset). Nghiên cứu sẽ tập trung vào việc áp dụng các mô hình học sâu (deep learning) như DenseNet121, InceptionV3, MobileNetV2, ResNet50 và so sánh hiệu suất của chúng với các mô hình học máy truyền thống như Random Forest, SVM, và KNN. Kết quả từ nghiên cứu không chỉ giúp nông dân phát hiện sớm các loại bệnh trên cây lúa một cách nhanh chóng và chính xác, mà còn hỗ trợ đưa ra các biện pháp xử lý kịp thời, từ đó nâng cao năng suất, chất lượng sản phẩm nông nghiệp, và thúc đẩy ứng dụng công nghệ trí tuệ nhân tạo (AI) trong nông nghiệp hiện đại.

## Giải pháp cụ thể để nhận dạng bệnh

Để giải quyết bài toán nhận dạng bệnh trên lá lúa trong giai đoạn đẻ nhánh, nghiên cứu đề xuất một quy trình gồm các bước từ tiền xử lý dữ liệu, xây dựng mô hình, đến huấn luyện và tối ưu hóa mô hình. Quy trình này được thiết kế để tận dụng tối đa các mô hình học máy và học sâu, đồng thời so sánh hiệu suất giữa chúng.

### Sơ đồ luồng hiện thực bài toán

Quy trình hiện thực bài toán nhận dạng bệnh trên lá lúa bao gồm các bước sau:

**Bước 1: Thu thập và chuẩn bị dữ liệu**

Sử dụng tập dữ liệu hình ảnh lá lúa từ Rice Leaf Diseases Detection Dataset trên Kaggle, bao gồm 6 lớp: Bacterial Leaf Blight (bạc lá), Leaf Blast (đạo ôn lá), Leaf Scald (cháy lá), Brown Spot (đốm nâu), Sheath Blight (khô vằn), và Healthy Rice Leaf (lá khỏe).

Với Dữ liệu được lưu trữ trên Google Drive, sử dụng thư viện glob để quét và lấy danh sách tất cả các tệp hình ảnh “.jpg” trong từng thư mục con, sau đó tạo danh sách “image\_links” chứa đường dẫn tới từng hình ảnh và danh sách “labels” chứa nhãn tương ứng, cuối cùng kiểm tra số lượng hình ảnh mỗi lớp bằng cách vẽ biểu đồ cột (data['labels'].value\_counts().plot.bar()) để đảm bảo dữ liệu không bị mất cân bằng nghiêm trọng.

**Bước 2: Tiền xử lý dữ liệu**

Chuẩn hóa và tăng cường dữ liệu hình ảnh để đảm bảo mô hình học được các đặc trưng đa dạng và tránh học quá mức (overfitting).

**Tải và chuẩn hóa hình ảnh**: Sử dụng hàm “load\_and\_resize\_image” để xử lý từng hình ảnh.

* Tải hình ảnh bằng “cv2.imread" từ đường dẫn trong image\_links.
* Thay đổi kích thước hình ảnh thành 224x224 (cho DenseNet121, MobileNetV2, ResNet50) hoặc 229x229 (cho InceptionV3) bằng “cv2.resize” để phù hợp với đầu vào của các mô hình học sâu.
* Chuyển đổi định dạng màu từ BGR (định dạng mặc định của OpenCV) sang RGB bằng “cv2.cvtColor”, vì các mô hình học sâu yêu cầu định dạng RGB.
* Chuẩn hóa giá trị pixel bằng cách chia toàn bộ giá trị pixel cho 255, đưa giá trị từ [0, 255] về khoảng [0, 1].

**Tăng cường dữ liệu** (Data Augmentation): Sử dụng ImageDataGenerator từ TensorFlow để tăng cường dữ liệu cho tập huấn luyện.

Áp dụng các biến đổi ngẫu nhiên như xoay hình ảnh tối đa 15 độ (rotation\_range=15), dịch chuyển ngang/dọc tối đa 10% (width\_shift\_range=0.1, height\_shift\_range=0.1), biến đổi góc cắt tối đa 10% (shear\_range=0.1), phóng to hoặc thu nhỏ tối đa 10% (zoom\_range=0.1), và lật ngang hình ảnh (horizontal\_flip=True).

Phương thức điền pixel bị thiếu sau biến đổi là fill\_mode='nearest', nghĩa là lấy giá trị từ pixel gần nhất để điền vào vùng trống.

Tập xác thực và kiểm tra không áp dụng tăng cường dữ liệu, chỉ chuẩn hóa bằng hàm preprocess\_input của từng mô hình (ví dụ: chuẩn hóa theo ImageNet).

**Tạo generator dữ liệu**: Sử dụng flow\_from\_dataframe để tạo các luồng dữ liệu (data generators).

* “train\_generator” cho tập huấn luyện, với tăng cường dữ liệu, kích thước lô (batch size) 32 và đầu ra dạng categorical (one-hot encoding).
* “val\_generator” cho tập xác thực và test\_generator cho tập kiểm tra, không tăng cường dữ liệu, chỉ chuẩn hóa, với cùng kích thước lô và đầu ra dạng categorical.

**Bước 3: Chia tập dữ liệu**

Chia dữ liệu thành 3 tập: tập huấn luyện (train) (70%), tập xác thực (validation) (15%) và tập kiểm tra (test) (15%) để huấn luyện và đánh giá mô hình.

**Chia lần 1:** Sử dụng “train\_test\_split” từ Scikit-learn để chia tập dữ liệu ban đầu thành tập huấn luyện + xác thực (85%) và tập kiểm tra (15%). Tham số “stratify=labels” được sử dụng để đảm bảo các lớp bệnh được phân bố đồng đều, ví dụ: nếu lớp Healthy Rice Leaf có 1000 hình ảnh, thì tập kiểm tra sẽ có khoảng 150 hình ảnh từ lớp này (15%).

**Chia lần 2:** Từ tập huấn luyện + xác thực (85%), tiếp tục chia thành tập huấn luyện (70% tổng dữ liệu) và tập xác thực (15% tổng dữ liệu) bằng “train\_test\_split” với “test\_size=0.176” (vì 0.176 \* 85% = 15%). Tham số “stratify” cũng được sử dụng để giữ tỷ lệ các lớp bệnh đồng đều.

Sử dụng “LabelEncoder” để chuyển các nhãn từ dạng văn bản (ví dụ: "Sheath\_Blight") thành số (0 đến 5).

Sau đó, nhãn của tập huấn luyện và xác thực được chuyển thành dạng one-hot encoding bằng “to\_categorical” (ví dụ: nhãn 0 thành [1, 0, 0, 0, 0, 0]) để phù hợp với đầu ra của mô hình học sâu. Nhãn của tập kiểm tra được giữ ở dạng số để đánh giá.

**Bước 4: Xây dựng mô hình**

**Mô hình CNN**

Khởi tạo mô hình cơ sở: Sử dụng các mô hình học sâu từ TensorFlow và Keras: DenseNet121, InceptionV3, MobileNetV2, và ResNet50. Mỗi mô hình được khởi tạo với “include\_top=False” (bỏ tầng đầu ra mặc định của ImageNet) và trọng số đã huấn luyện trước trên ImageNet (weights='imagenet').

Thêm các tầng tùy chỉnh:

Tầng GlobalAveragePooling2D để giảm chiều dữ liệu từ không gian 3D (ví dụ: 7x7x2048 với ResNet50) thành vector 1D (2048).

Tầng BatchNormalization để chuẩn hóa đầu ra, giúp huấn luyện ổn định hơn.

Tầng Dropout (tỷ lệ 0.5-0.6) để ngẫu nhiên bỏ qua một số nơ-ron, giảm nguy cơ học quá mức.

Tầng Dense với 64 hoặc 128 đơn vị, hàm kích hoạt ReLU và L2 regularization (trong ResNet50, MobileNetV2, InceptionV3) để phạt các trọng số lớn, giảm học quá mức.

* Tầng Ouput Dense với 6 đơn vị, hàm kích hoạt softmax để phân loại 6 lớp bệnh.

Biên dịch mô hình: Sử dụng bộ tối ưu Adam với tốc độ học ban đầu 0.001, hàm mất mát “categorical\_crossentropy” (phù hợp với bài toán phân loại đa lớp) và chỉ số đo lường là “accuracy”.

**Mô hình truyền thống**

Sử dụng RandomForestClassifier, SVC và KNeighborsClassifier từ Scikit-learn để xây dựng các mô hình. Các mô hình này phân loại dựa trên dữ liệu hình ảnh đã được tiền xử lý và chuyển đổi thành đặc trưng phù hợp.

**Bước 5: Huấn luyện và tối ưu mô hình**

Huấn luyện mô hình trên tập huấn luyện, tối ưu hóa hiệu suất bằng các kỹ thuật như điều chỉnh siêu tham số (Hyperparameter), sử dụng kỹ thuật dừng sớm (early stopping) và giảm tốc độ học (learning rate scheduling).

**Huấn luyện ban đầu**

Các mô hình CNN được huấn luyện trong 5 epoch đầu tiên trên “train\_generator” với dữ liệu xác thực từ “val\_generator”.

Sử dụng bộ tối ưu Adam với tốc độ học 0.001, hàm mất mát “categorical\_crossentropy” và chỉ số đo lường accuracy.

Các tầng của mô hình cơ sở (base model) được đóng băng (layer.trainable = False) để giữ trọng số đã huấn luyện trước trên ImageNet, chỉ huấn luyện các tầng mới thêm vào như Global Average Pooling, Dense và Dropout.

**Tinh chỉnh mô hình** **(fine-tuning)**

Sau 5 epoch đầu, mở khóa một số tầng cuối của mô hình cơ sở: 10 tầng cuối với DenseNet121 và ResNet50, 20 tầng cuối với MobileNetV2, 50 tầng cuối với InceptionV3.

Biên dịch lại mô hình với tốc độ học nhỏ hơn (0.0001) để tránh thay đổi trọng số quá lớn, vẫn sử dụng Adam và “categorical\_crossentropy”.

Huấn luyện thêm 20 epoch với các kỹ thuật tối ưu hóa:

* **Early Stopping:** Dừng huấn luyện nếu độ mất mát trên tập xác thực (val\_loss) không cải thiện sau 3-5 epoch, đồng thời khôi phục trọng số tốt nhất (restore\_best\_weights=True).
* **Learning Rate Scheduler:** Sử dụng “ReduceLROnPlateau” (với DenseNet121, ResNet50, InceptionV3) để giảm tốc độ học nếu độ mất mát không cải thiện sau 2 epoch hoặc Cosine Annealing (với MobileNetV2) để điều chỉnh tốc độ học theo hàm cosin qua các epoch.

Các mô hình Random Forest, SVM và KNN được huấn luyện trên tập huấn luyện, với dữ liệu hình ảnh đã được tiền xử lý và chuyển đổi thành vector đặc trưng. Các tham số như số cây trong Random Forest, tham số phạt *C* trong SVM và hàng xóm *k* trong kNN.

**Bước 6: Đánh giá hiệu suất**

Các mô hình được đánh giá trên tập kiểm tra (test\_generator) bằng cách dự đoán nhãn và so sánh với nhãn thực tế.

Dự đoán: Sử dụng “model.predict” để dự đoán trên tập kiểm tra, lấy nhãn có xác suất cao nhất bằng np.argmax. Nhãn thực tế được lấy từ “test\_generator.classes”.

Sử dụng thư viện Scikit-learn để tính các chỉ số Accuracy, Precision, Recall và F1-Score. Vẽ ma trận nhầm lẫn bằng “confusion\_matrix” và hiển thị bằng sns.heatmap. vẽ đường cong ROC và tính AUC bằng roc\_curve và auc, hiển thị bằng Matplotlib, với mỗi lớp có một đường cong riêng.

Sau đó vẽ biểu đồ độ chính xác và độ mất mát qua các epoch bằng plt.plot.

*Hình 3.1: Sơ đồ luồng minh họa hiện thực bài toán*

### Giải pháp tiền xử lý dữ liệu

Tiền xử lý dữ liệu là bước quan trọng để đảm bảo chất lượng hình ảnh đầu vào phù hợp với các mô hình học máy và học sâu. Các bước tiền xử lý được thực hiện như sau:

**Tải và chuẩn hóa hình ảnh**: Hình ảnh lá lúa được tải từ tập dữ liệu và chuẩn hóa bằng cách thay đổi kích thước thành 224x224 (cho DenseNet121, MobileNetV2, ResNet50) hoặc 229x229 (cho InceptionV3), chuyển đổi định dạng màu từ BGR sang RGB, và chuẩn hóa giá trị pixel về khoảng [0, 1] bằng cách chia cho 255. Điều này giúp hình ảnh đồng nhất và phù hợp với yêu cầu đầu vào của các mô hình học sâu.

**Tăng cường dữ liệu** (Data Augmentation): Để tăng số lượng và đa dạng dữ liệu, đồng thời giảm nguy cơ học quá mức, nghiên cứu áp dụng các kỹ thuật tăng cường dữ liệu trên tập huấn luyện, bao gồm: xoay ngẫu nhiên (rotation range 15 độ), dịch chuyển ngang và dọc (width/height shift range 0.1), biến đổi góc cắt (shear range 0.1), phóng to/thu nhỏ (zoom range 0.1), và lật ngang (horizontal flip). Các kỹ thuật này giúp mô hình học được các đặc trưng đa dạng hơn từ hình ảnh, ví dụ hình ảnh lá lúa bị bệnh có thể xuất hiện ở nhiều góc độ hoặc điều kiện ánh sáng khác nhau [10].

**Chia tập dữ liệu**: Dữ liệu được chia thành 3 tập với tỷ lệ 70% cho tập huấn luyện, 15% cho tập xác thực, và 15% cho tập kiểm tra, đảm bảo các lớp bệnh được phân bố đồng đều (stratified split) để tránh mất cân bằng dữ liệu.

**Mã hóa nhãn** (Label Encoding): Các nhãn lớp (ví dụ: "Bacterial Leaf Blight", "Healthy Rice Leaf") được mã hóa thành số (0 đến 5) bằng LabelEncoder, sau đó chuyển thành dạng one-hot encoding (ví dụ: [1, 0, 0, 0, 0, 0]) để phù hợp với đầu ra của mô hình học sâu

### Xây dựng mô hình

**Mô hình CNN**

**ResNet50**: Sử dụng kiến trúc ResNet50 với các khối dư (residual blocks) và kết nối tắt (skip connections), giúp giảm vấn đề biến mất gradient. Mô hình được thêm các tầng Global Average Pooling, Dense, và BatchNormalization để cải thiện hiệu suất.

**MobileNetV2**: Sử dụng kiến trúc MobileNetV2 với các khối nghịch đảo dư (inverted residual blocks) và tích chập tách biệt chiều sâu, tối ưu cho hiệu suất trên thiết bị di động. Mô hình được bổ sung các tầng BatchNormalization và Dropout để tăng độ ổn định.

**InceptionV3**: Sử dụng kiến trúc InceptionV3 với các mô-đun Inception xử lý song song, giúp chiết xuất đặc trưng ở nhiều tỷ lệ khác nhau. Mô hình được thêm các tầng Global Average Pooling, Dense, và Dropout để giảm học quá mức.

**DenseNet121**: Sử dụng kiến trúc DenseNet121 với các khối dày đặc (dense blocks) và kết nối trực tiếp giữa các tầng, giúp giảm số lượng tham số và tăng hiệu quả học đặc trưng. Mô hình được xây dựng với tầng Global Average Pooling, các tầng Dense với hàm kích hoạt ReLU, và tầng đầu ra softmax cho 6 lớp bệnh.

**Mô hình truyền thống**

**Random Forest:** Sử dụng thuật toán Random Forest với nhiều cây quyết định, tổng hợp kết quả qua majority voting để phân loại bệnh.

**KNN:** Sử dụng thuật toán kNN để phân loại bệnh dựa trên các hàng xóm gần nhất của hình ảnh mới, với nhãn được chọn qua majority voting.

**SVM:** Sử dụng thuật toán SVM để tìm siêu phẳng phân tách tối ưu giữa các lớp bệnh, với khả năng xử lý dữ liệu không tuyến tính nhờ kernel trick.

### Huấn luyện và tối ưu mô hình

**Mô hình CNN**

Huấn luyện ban đầu: Các mô hình học sâu được huấn luyện với tập huấn luyện trong 5 epoch ban đầu, sử dụng bộ tối ưu Adam với tốc độ học (learning rate) 0.001. Các tầng của mô hình cơ sở (base model) được đóng băng (frozen) để giữ lại trọng số đã huấn luyện trước (pre-trained) trên ImageNet, chỉ huấn luyện các tầng mới thêm vào như Global Average Pooling và Dense.

Tinh chỉnh mô hình (Fine-tuning): Sau 5 epoch đầu, các tầng cuối của mô hình cơ sở (10 tầng cuối với DenseNet121 và ResNet50, 20 tầng cuối với MobileNetV2, 50 tầng cuối với InceptionV3) được mở khóa để huấn luyện thêm. Tốc độ học được giảm xuống 0.0001 để tránh thay đổi trọng số quá lớn, đồng thời sử dụng các kỹ thuật tối ưu như:

* **Early Stopping:** Dừng huấn luyện nếu độ mất mát trên tập xác thực (val\_loss) không cải thiện sau 3-5 epoch, giúp tránh overfitting.
* **Learning Rate Scheduler:** Với DenseNet121, ResNet50 và InceptionV3, sử dụng “ReduceLROnPlateau” để giảm tốc độ học nếu độ mất mát không cải thiện sau 2 epoch. Với MobileNetV2 sử dụng “Cosine Annealing Learning Rate Schedule”r để điều chỉnh tốc độ học một cách mượt mà qua các epoch.

Tối ưu thêm với các tầng bổ sung: Các mô hình học sâu được thêm các tầng như BatchNormalization để ổn định huấn luyện, Dropout (0.5-0.6) và L2 regularization (trong ResNet50, MobileNetV2, InceptionV3) để giảm hiện tượng overfitting bằng cách phạt các trọng số lớn.

**Mô hình truyền thống:** Các mô hình Random Forest, SVM và KNN được huấn luyện trên tập huấn luyện, với dữ liệu hình ảnh đã được tiền xử lý, với các tham số được điều chỉnh để đạt hiệu suất tốt nhất.

## Hiện thực giải pháp

### Môi trường triển khai thực nghiệm

Môi trường thực nghiệm được triển khai trên Google Colab, một nền tảng tính toán đám mây miễn phí, hỗ trợ GPU để tăng tốc huấn luyện các mô hình học sâu.

Môi trường lập trình: Sử dụng Python với các thư viện chính như TensorFlow và Keras để xây dựng và huấn luyện các mô hình học sâu (DenseNet121, InceptionV3, MobileNetV2, ResNet50), và Scikit-learn để triển khai các mô hình học máy truyền thống (Random Forest, SVM, KNN).

Lưu trữ dữ liệu: Tập dữ liệu Rice Leaf Diseases Detection được lưu trữ trên Google Drive tại thư mục “/content/gdrive/MyDrive/tieuLuan/dataset/Rice\_Leaf\_AUG”

Thư viện hỗ trợ: Sử dụng các thư viện bổ trợ như OpenCV để tải và xử lý hình ảnh, Matplotlib và Seaborn để trực quan hóa kết quả (biểu đồ độ chính xác, độ mất mát, ma trận nhầm lẫn, đường cong ROC) và NumPy, Pandas để quản lý và xử lý dữ liệu.

Cấu hình huấn luyện: Các mô hình được huấn luyện với kích thước lô (batch size) là 32 (16 với MobileNetV2 để cập nhật trọng số mượt mà hơn), số epoch tối đa là 20 trong giai đoạn tinh chỉnh và sử dụng các kỹ thuật như Early Stopping, Learning Rate Scheduler để tối ưu hóa quá trình huấn luyện.

Lưu kết quả: Các mô hình sau khi huấn luyện được lưu dưới định dạng Keras

Bảng 3.2: Hyperparameter của từng mô hình CNN

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Hyperparameter** | **ResNet50** | **InceptionV3** | **DenseNet121** | **MobileNetV2** |
| **Số lớp trainable** | 10 | 50 | 10 | 20 |
| **Learning Rate**  **(trước/fine-tune)** | 0.001/ 0.0001 | | | 0.001/ 0.0005 |
| **Batch Size** | 32 | 32 | 32 | 16 |
| **Epochs** | 25 | | | |
| **Optimizer** | Adam | | | |
| **Dropout** | 0.5 | 0.5 | 0.5 | 0.6 |
| **L2\_Regularization** | 0.01 | 0.001 |  | 0.01 |
| **Learning Rate Scheduler** | ReduceLROnPlateau | | | Cosine Annealing |
| **Early Stopping (Patience)** | 3 | 5 | 3 | 5 |
| **Accuracy (Train)** | 0.9921 | 0.9860 | 0.8936 | 0.9947 |
| **Accuracy (Val)** | 0.9937 | 0.9930 | 0.9734 | 0.9930 |
| **Thời gian huấn luyện/ Epoch** | 1443s-1508s | 1184s-1222s | 1272s-1487s | 492s -526s |

**Ý nghĩa của các Hyperparameter**

**Số lớp trainable:** Số tầng được huấn luyện trong giai đoạn tinh chỉnh, ảnh hưởng đến khả năng học đặc trưng mới của mô hình.

**Learning Rate:** Tốc độ cập nhật trọng số, lớn thì học nhanh nhưng dễ bỏ qua điểm tối ưu, nhỏ thì học chậm và chính xác hơn.

**Batch Size:** Số mẫu xử lý mỗi lần cập nhật trọng số, nhỏ thì cập nhật mượt mà nhưng chậm, lớn thì nhanh nhưng có thể bỏ qua chi tiết.

**Số Epochs:** Số lần mô hình lặp qua dữ liệu huấn luyện, nhiều thì học kỹ nhưng dễ overfitting.

**Optimizer:** Thuật toán cập nhật trọng số, Adam giúp mô hình hội tụ nhanh và ổn định.

**Dropout:** Tỷ lệ bỏ qua nơ-ron ngẫu nhiên để giảm overfitting, giúp mô hình không phụ thuộc quá nhiều vào một số nơ-ron cụ thể.

**L2 Regularization:** Phạt trọng số lớn để giảm overfitting; hệ số lớn phạt nặng hơn, làm mô hình đơn giản hơn.

**Learning Rate Scheduler:** Điều chỉnh tốc độ học trong huấn luyện, giúp mô hình hội tụ tốt hơn khi gần điểm tối ưu.

**Early Stopping:** Dừng huấn luyện nếu không cải thiện, tránh overfitting, patience là số epoch chờ trước khi dừng.

**Accuracy (Train/Val):** Độ chính xác trên tập huấn luyện hoặc tập xác thực, chênh lệch lớn giữa hai tập có thể chỉ ra overfitting.

**Thời gian huấn luyện (Epochs):** Tổng thời gian thực tế để huấn luyện mô hình qua các epoch, phụ thuộc vào batch size và phần cứng.

### Kết quả thực nghiệm

Kết quả thực nghiệm với phương pháp fine-tune các mô hình CNN gồm ResNet50, InceptionV3, DenseNet121, MobileNetV2 và các phương pháp truyền thống như Random Forest, kNN, SVM.

**ResNet50**

*Hình 3.5: Biểu đồ theo dõi hiệu suất mô hình ResNet50*

*Hình 3.6: Confusion Matrix của mô hình ResNet50*

*Hình 3.7: Biểu đồ ROC và chỉ số AUC các class của mô hình InceptionV3*

**InceptionV3**

*Hình 3.8: Biểu đồ theo dõi hiệu suất mô hình InceptionV3*

*Hình 3.9: Confusion Matrix của mô hình InceptionV3*

*Hình 3.10: Biểu đồ ROC và chỉ số AUC từng lớp của mô hình ResNet50*

**DenseNet121**

*Hình 3.11: Biểu đồ theo dõi hiệu suất mô hình DenseNet121*

*Hình 3.12: Confusion Matrix của mô hình DenseNet121*

*Hình 3.13: Biểu đồ ROC và chỉ số AUC từng lớp của mô hình InceptionV3*

**MobileNetV2**

*Hình 3.14: Biểu đồ theo dõi hiệu suất mô hình MobileNetV2*

*Hình 3.15: Confusion Matrix của mô hình MobileNetV2*

*Hình 3.16: Biểu đồ ROC và chỉ số AUC từng lớp của mô hình MobileNetV2* **Random Forest**

*Hình 3.17: Confusion Matrix của Random Forest*

*Hình 3.18: Biểu đồ ROC và chỉ số AUC từng lớp của Random Forest* **SVM**

*Hình 3.19: Confusion Matrix của SVM*

*Hình 3.20: Biểu đồ ROC và chỉ số AUC từng lớp của SVM* **kNN**

*Hình 3.21: Confusion Matrix của kNN*

*Hình 3.22: Biểu đồ ROC và chỉ số AUC từng lớp của kNN*

Đánh giá mô hình dụng bằng các chỉ số đánh giá như Accuracy, Precision, Recall, F1 – Score.

Bảng 3.3: Bảng thống kê kết quả của từng mô hình trên 5 lần chạy

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mô hình** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **RestNet50** | 0,9278 | 0,9298 | 0,9279 | 0,9283 |
| **InceptionV3** | 0,9296 | 0,9303 | 0,9296 | 0,9297 |
| **DenseNet121** | 0,8172 | 0,8204 | 0,8172 | 0,8164 |
| **MobileNetV2** | **0,9887** | **0,9889** | **0,9887** | **0,9888** |
| **SVM** | 0,9813 | 0,9815 | 0,9813 | 0,9813 |
| **kNN** | 0,9667 | 0,9673 | 0,9667 | 0,9668 |
| **Random Forest** | 0,9855 | 0,9856 | 0,9855 | 0,9855 |

### Phân tích, đánh giá và nhận xét kết quả

**Phân tích và đánh giá kết quả dựa trên các chỉ số đánh giá và thời gian chạy:**

Dựa trên các chỉ số Accuracy, Precision, Recall và F1-Score của các mô hình, có thể rút ra một số nhận định quan trọng về hiệu suất và mức độ phù hợp của từng phương pháp trong bài toán phân loại hình ảnh.

* Các mô hình học sâu (Deep Learning) như MobileNetV2, ResNet50, InceptionV3 và DenseNet121 đều đạt được hiệu quả rất cao với các chỉ số vượt mức 96%, trong đó **MobileNetV2** nổi bật nhất với độ chính xác gần như tuyệt đối (Accuracy: 98.87%), cùng với các chỉ số Precision, Recall và F1 đều tiệm cận mức tối đa. Điều này cho thấy mô hình có khả năng nhận diện mẫu rất chính xác và đồng đều trên mọi khía cạnh đánh giá.
* ResNet50 cũng thể hiện hiệu suất vượt trội (Accuracy: 98.55%) và chỉ thấp hơn một chút so với MobileNetV2. Tuy nhiên, thời gian huấn luyện mỗi epoch của ResNet50 khá cao (1443s - 1508s), cho thấy yêu cầu tài nguyên tính toán lớn hơn. Trong khi đó, MobileNetV2 chỉ cần khoảng 492s - 526s mỗi epoch, cho thấy đây là một mô hình cực kỳ tối ưu về hiệu suất tính toán lẫn độ chính xác, rất phù hợp với các hệ thống có giới hạn về phần cứng.
* InceptionV3 và DenseNet121 cũng đạt được kết quả rất khả quan, lần lượt là 98.13% và 96.67% về Accuracy. Mặc dù thời gian huấn luyện giữa hai mô hình này khá tương đương, DenseNet121 có lợi thế về độ ổn định với các chỉ số Precision và F1 gần nhau, phù hợp cho các ứng dụng yêu cầu độ cân bằng giữa tài nguyên và hiệu năng.
* Ở nhóm mô hình truyền thống, kNN và SVM có kết quả tốt nhưng không thể cạnh tranh với các mô hình học sâu. SVM đạt Accuracy 92.96%, với độ chính xác và khả năng phát hiện mẫu rất đồng đều. kNN cũng có kết quả khá tương tự (Accuracy: 92.78%). Tuy nhiên, hai mô hình này không tận dụng được đặc trưng không gian của dữ liệu hình ảnh như các mô hình CNN, do đó hiệu quả thực tế có thể bị giới hạn khi áp dụng vào dữ liệu có độ phức tạp cao hơn.
* Ngược lại, Random Forest cho kết quả thấp hơn đáng kể (Accuracy: 81.72%), chứng tỏ sự hạn chế rõ rệt trong việc xử lý dữ liệu hình ảnh ở định dạng gốc. Đây là mô hình phù hợp hơn cho dữ liệu dạng bảng hơn là dữ liệu không gian như ảnh.

**Phân tích và đánh giá kết quả của mô hình CNN dựa trên Confusion Matrix**

Các mô hình CNN (DenseNet121, InceptionV3, MobileNetV2, ResNet50) nhìn chung đạt hiệu quả phân loại cao trên các lớp như Healthy Rice Leaf, Leaf Blast và Brown Spot, với phần lớn mẫu được nhận diện đúng theo nhãn thực tế. Tuy nhiên, vẫn tồn tại một số khó khăn khi phân loại các lớp bệnh có đặc điểm hình thái gần giống nhau.

Cụ thể, lớp Leaf Scald thường bị nhầm lẫn với Bacterial Leaf Blight và Leaf Blast ở một số trường hợp. Điều này có thể do cả ba loại bệnh đều tạo ra các vùng đốm hoặc vết loang trên lá với màu sắc và hình dạng tương tự nhau, đặc biệt ở giai đoạn sớm khi các vết cháy của Leaf Scald có thể trông giống các vệt loang màu xám trắng của Bacterial Leaf Blight hoặc các đốm xám của Leaf Blast. Dù vậy, trong học máy, sự khác biệt về độ đậm nhạt của màu sắc (vết loang của Leaf Scald thường đậm hơn so với Bacterial Leaf Blight) hỗ trợ mô hình phân biệt tốt hơn, nhưng vẫn cần dữ liệu phong phú hơn để giảm thiểu nhầm lẫn.

Bên cạnh đó, lớp Sheath Blight và Brown Spot cũng ghi nhận một số trường hợp nhầm lẫn. Nguyên nhân có thể là do sự tương đồng về đặc trưng hình ảnh ở giai đoạn đầu, khi các vùng nhiễm khuẩn của Sheath Blight chưa lan rộng và có thể bị nhầm với các đốm nâu đặc trưng của Brown Spot. Những yếu tố như chất lượng hình ảnh không đồng đều (ánh sáng yếu, độ phân giải thấp) cũng góp phần làm tăng khả năng nhầm lẫn giữa các lớp này.

**Phân tích và đánh giá kết quả dựa trên biểu đồ ROC và chỉ số AUC**

Dựa vào biểu đồ ROC và chỉ số AUC, các mô hình học sâu như MobileNetV2, ResNet50 và InceptionV3 cho thấy hiệu suất vượt trội, đạt AUC 1.00 ở hầu hết các lớp như Bacterial Leaf Blight, Brown Spot, Healthy Rice Leaf, Leaf Blast và Leaf Scald, nhưng có nhầm lẫn nhỏ ở lớp Sheath Blight (AUC 0.99), do lớp này dễ bị nhầm với Leaf Blast hoặc Leaf Scald vì hình thái tương đồng ở giai đoạn đầu. DenseNet121 cũng đạt AUC 1.00 ở phần lớn lớp, nhưng giảm xuống 0.99 ở Bacterial Leaf Blight, cho thấy khó khăn khi phân biệt lớp này với các lớp tương tự như Leaf Blast. Trong khi đó, các mô hình truyền thống như SVM đạt AUC 0.99 đồng đều cho tất cả lớp, Random Forest có AUC dao động từ 0.94 đến 1.00 (thấp nhất ở Sheath Blight) và KNN cũng tương tự (AUC từ 0.94 đến 1.00), phản ánh nhầm lẫn nhiều hơn giữa các lớp có đặc trưng giống nhau như Bacterial Leaf Blight và Sheath Blight,

# KẾT LUẬN VÀ KIẾN NGHỊ

## Kết luận

## Kiến nghị

# TÀI LIỆU THAM KHẢO

[1] Food and Agriculture Organization of the United Nations, “World Food and Agriculture Statistical Yearbook 2023,” 2023. Accessed: May 1, 2025. [Online]. Available: https://www.fao.org/documents/card/en/c/cc8166en

[2] International Rice Research Institute (IRRI), “Rice Diseases: Identification and Management,” 2020. Accessed: May 1, 2025. [Online]. Available: https://www.irri.org/resources/rice-diseases

[3] T. T. Tran, H. V. Nguyen, and Q. L. Pham, “Quản lý bệnh trên cây lúa tại Việt Nam,” Tạp chí Khoa học Nông nghiệp Việt Nam, vol. 18, no. 3, pp. 205–214, 2020.

[4] Kaggle, “Rice Leaf Diseases Detection Dataset,” 2023. Accessed: May 1, 2025. [Online]. Available: https://www.kaggle.com/datasets/loki4514/rice-leaf-diseases-detection/data

[5] Y. Lu, S. Yi, N. Zeng, Y. Liu, and Y. Zhang, “Deep Learning for Rice Disease Detection,” Computers and Electronics in Agriculture, vol. 140, pp. 321–329, Aug. 2017.

[6] X. Zhang, Y. Qiao, F. Meng, C. Fan, and M. Zhang, “Convolutional Neural Networks for Rice Disease Classification,” Precision Agriculture, vol. 20, no. 5, pp. 1047–1060, Oct. 2019.

[7] S. H. Ou, Rice Diseases, 2nd ed. Kew, Surrey: Commonwealth Mycological Institute, 1985.

[8] E. A. Heinrichs, Biology and Management of Rice Insects. Los Baños, Philippines: International Rice Research Institute, 1994.

[9] T. W. Mew, “Bacterial Blight of Rice,” in Plant Pathology in Asia, H. S. Chaube, Ed. New Delhi: Oxford & IBH Publishing Co., 1993, pp. 123–140.

[10] Z. Liu, L. Zhang, and Y. Yang, “Deep Learning for Plant Disease Detection,” Computers and Electronics in Agriculture, vol. 175, pp. 105–115, Aug. 2020.

[11] Keras Team, “Keras: The Python Deep Learning API,” 2023. Accessed: May 1, 2025. [Online]. Available: https://keras.io/

[12] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, Oct. 2011.

[13] G. Bradski and A. Kaehler, “OpenCV,” Dr. Dobb’s Journal of Software Tools, vol. 25, no. 11, pp. 120–126, Nov. 2000.

[14] Google, “Google Drive: Cloud Storage for Work and Home,” 2023. Accessed: May 1, 2025. [Online]. Available: https://www.google.com/drive/

[15] Google, “Google Colaboratory: Write and Execute Python Code in Your Browser,” 2023. Accessed: May 1, 2025. [Online]. Available: https://colab.research.google.com/

[16] S. J. Pan and Q. Yang, “A Survey on Transfer Learning,” IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[17] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How Transferable Are Features in Deep Neural Networks?” in Advances in Neural Information Processing Systems (NIPS), vol. 27, pp. 3320–3328, 2014.

[18] vMixGPT, “Vanishing Gradient và Exploding Gradient là gì? Làm sao để ngăn chặn chúng,” 2025. Accessed: May 30, 2025. [Online]. Available: https://vmixgpt.com/vanishing-gradient-va-exploding-gradient-la-gi-lam-sao-e-ngan-chan-chung/

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778, 2016. [Online]. Available: https://www.cv-foundation.org/openaccess/content\_cvpr\_2016/papers/He\_Deep\_Residual\_Learning\_CVPR\_2016\_paper.pdf

[20] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4510–4520, 2018. [Online].

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2818–2826, 2016. [Online].

[22] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2261–2269, 2017. [Online].

[23] L. Breiman, “Random Forests,” Machine Learning, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: https://link.springer.com/article/10.1023/A:1010933404324

[24] T. M. Cover and P. E. Hart, “Nearest Neighbor Pattern Classification,” IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21–27, 1967. [Online]. Available: https://ieeexplore.ieee.org/document/1053964

[25] C. Cortes and V. Vapnik, “Support-Vector Networks,” Machine Learning, vol. 20, no. 3, pp. 273–297, 1995. [Online].

[26] Fawcett, T., “An introduction to ROC analysis,” Pattern Recognition Letters, vol. 27, no. 8, pp. 861–874, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016786550500303X

[27] Goodfellow, I., Bengio, Y., & Courville, A., Deep Learning, MIT Press, 2016.

# PHỤ LỤC