# MỞ ĐẦU

## 1. Bối cảnh & Ý nghĩa của Đề tài

Trong thập kỷ qua, trí tuệ nhân tạo (AI) và học sâu (Deep Learning) đã có những bước tiến đột phá, mang lại nhiều ứng dụng quan trọng trong lĩnh vực y học. Đặc biệt, việc áp dụng AI vào chẩn đoán hình ảnh y khoa đã tạo ra những phương pháp hỗ trợ bác sĩ trong việc phát hiện bệnh với độ chính xác cao hơn và nhanh chóng hơn.

Trong chẩn đoán các bệnh về phổi, ảnh X-ray là công cụ quan trọng giúp phát hiện các bất thường trong mô phổi. Tuy nhiên, việc phân tích ảnh X-ray đòi hỏi chuyên môn cao và phụ thuộc vào kinh nghiệm của bác sĩ. Hơn nữa, yếu tố con người có thể dẫn đến sai sót do mệt mỏi hoặc thiếu kinh nghiệm trong các trường hợp hiếm gặp. Những hạn chế này tạo động lực cho việc nghiên cứu và triển khai các mô hình AI nhằm hỗ trợ chẩn đoán chính xác và nhất quán hơn.

Việc ứng dụng AI không chỉ giúp giảm thiểu sai sót trong chẩn đoán mà còn có thể góp phần tối ưu hóa quy trình làm việc trong bệnh viện, giảm tải áp lực cho đội ngũ y tế và đảm bảo bệnh nhân được điều trị kịp thời hơn. Hơn nữa, AI có khả năng học hỏi và cải tiến liên tục nhờ dữ liệu đầu vào lớn, giúp cải thiện chất lượng chẩn đoán theo thời gian.

**Ý nghĩa và tầm quan trọng của ứng dụng học máy trong chẩn đoán y tế**

Việc tích hợp học máy vào quy trình chẩn đoán hình ảnh y khoa mang lại nhiều lợi ích vượt trội:

1. **Cải thiện độ chính xác trong chẩn đoán**

* Các mô hình AI có thể phát hiện các đặc trưng quan trọng trong ảnh X-ray mà con người có thể bỏ sót.
* Những mô hình học sâu được huấn luyện trên lượng dữ liệu lớn có thể đạt được độ chính xác ngang bằng hoặc thậm chí cao hơn so với bác sĩ lâm sàng trong một số trường hợp.
* AI có thể giúp chuẩn hóa chẩn đoán giữa các bác sĩ, giảm sự khác biệt trong kết quả do yếu tố con người.

2. **Tăng tốc độ xử lý và tối ưu hóa nguồn lực y tế**

* AI có thể phân tích một lượng lớn ảnh X-ray trong thời gian ngắn, giúp giảm tải công việc cho bác sĩ.
* Điều này đặc biệt hữu ích trong các bệnh viện quá tải hoặc các khu vực thiếu hụt chuyên gia y tế.
* AI có thể hoạt động 24/7 mà không bị ảnh hưởng bởi yếu tố mệt mỏi hay áp lực công việc.

3. **Phát hiện bệnh sớm và giám sát liên tục**

* Các hệ thống AI có thể nhận diện các dấu hiệu bất thường ngay cả khi chúng chưa rõ ràng đối với bác sĩ.
* Việc chẩn đoán sớm giúp cải thiện hiệu quả điều trị và giảm nguy cơ biến chứng.
* Hệ thống AI có thể hỗ trợ bác sĩ trong việc theo dõi tiến triển bệnh của bệnh nhân theo thời gian.

4. **Hỗ trợ quyết định lâm sàng**

* + AI có thể cung cấp các phân tích dựa trên dữ liệu lịch sử và hỗ trợ bác sĩ trong việc đưa ra quyết định điều trị.
  + Tuy không thay thế vai trò của bác sĩ, AI đóng vai trò như một công cụ hỗ trợ, giúp giảm thiểu sai sót và cải thiện chất lượng chăm sóc bệnh nhân.
  + Hệ thống AI có thể tích hợp với các nền tảng hồ sơ bệnh án điện tử để cung cấp đánh giá toàn diện hơn về tình trạng sức khỏe bệnh nhân.

Với những lợi ích kể trên, việc nghiên cứu và triển khai ứng dụng AI trong phân tích ảnh X-ray không chỉ mang tính cấp thiết mà còn mở ra nhiều cơ hội mới trong lĩnh vực y tế hiện đại.

## 2. Mục tiêu & Phạm vi Nghiên cứu

Đề tài này tập trung vào việc phát triển một hệ thống chẩn đoán viêm phổi từ ảnh X-ray bằng cách ứng dụng các mô hình học sâu. Các mục tiêu chính bao gồm:

1. **Huấn luyện và đánh giá mô hình**

* Sử dụng các mô hình pretrained như DenseNet121, ResNet50, EfficientNet B0 và một mô hình CNN tùy chỉnh để thực hiện phân loại ảnh X-ray.
* So sánh hiệu năng của các mô hình dựa trên độ chính xác, độ nhạy và độ đặc hiệu.
* Lựa chọn mô hình tối ưu để triển khai trong ứng dụng thực tế.
* Tối ưu hóa mô hình để đạt hiệu suất cao nhất mà vẫn đảm bảo tốc độ xử lý nhanh.

2. **Xây dựng ứng dụng web**

* Phát triển giao diện web cho phép người dùng tải ảnh X-ray lên hệ thống.
* Tích hợp mô hình AI để dự đoán và hiển thị kết quả trên giao diện web.
* Đảm bảo hệ thống hoạt động ổn định, dễ sử dụng và có tốc độ xử lý nhanh.
* Cải thiện UI/UX để mang lại trải nghiệm tốt nhất cho người dùng, bao gồm các bác sĩ và bệnh nhân.

3. **Xây dựng cơ sở dữ liệu lưu trữ lịch sử dự đoán**

* Thiết kế database để lưu trữ thông tin về các lần chẩn đoán, bao gồm ảnh đầu vào, kết quả dự đoán, thời gian thực hiện.
* Hỗ trợ truy xuất dữ liệu để phục vụ nghiên cứu và cải tiến mô hình trong tương lai.
* Đảm bảo tính bảo mật và quyền riêng tư của dữ liệu bệnh nhân khi lưu trữ và xử lý thông tin.

**Phạm vi nghiên cứu**

* **Dữ liệu đầu vào**: Chỉ sử dụng dataset Chest X-ray Images có sẵn chia làm 2 tập “Train” và “Test”, bao gồm ảnh đã được gán nhãn "Normal" và "Pneumonia".
* **Phương pháp tiếp cận**: Tập trung vào bài toán phân loại nhị phân (Normal/Pneumonia) mà không mở rộng sang các bệnh lý phổi khác.
* **Hệ thống triển khai**: Xây dựng mô hình AI trên nền tảng Python, sử dụng các thư viện phổ biến như TensorFlow, PyTorch và Flask để triển khai ứng dụng web.
* **Giới hạn**: Không thực hiện chẩn đoán đa bệnh lý, không thay thế bác sĩ trong quyết định lâm sàng, chỉ đóng vai trò hỗ trợ chuyên môn.
* **Môi trường thử nghiệm**: Chạy mô hình trên GPU (Kết hợp nền tảng Google Colab) để tăng tốc độ huấn luyện và đánh giá hiệu suất trên tập dữ liệu thử nghiệm trước khi triển khai thực tế.

Với những mục tiêu và phạm vi nghiên cứu trên, đề tài hướng tới việc phát triển một hệ thống chẩn đoán AI có tính ứng dụng cao, giúp hỗ trợ bác sĩ trong công tác chẩn đoán viêm phổi và góp phần nâng cao chất lượng dịch vụ y tế.

# CHƯƠNG I. TỔNG QUAN

## 1. Tổng quan về Dataset & Bệnh Pneumonia

Dataset "Labeled Chest X-Ray Images" được cung cấp trên Kaggle bởi Tolga Dincer là một nguồn tài liệu quý giá cho nghiên cứu về phân loại ảnh y khoa, đặc biệt trong việc phát hiện viêm phổi. Các thông tin chi tiết về dataset bao gồm:

* **Nguồn gốc**:  
  Dữ liệu được thu thập và gán nhãn bởi các chuyên gia y tế, theo nguồn trích dẫn các hình ảnh (trước - sau) được chọn từ nhóm các bệnh nhân nhi khoa từ một đến năm tuổi theo hồi cứu báo cáo từ Trung tâm Y tế Phụ nữ và Trẻ em Quảng Châu, Quảng Châu, Trung Quốc.
* **Cấu trúc dữ liệu**:
  + Dataset chứa gần **6000 file ảnh** định dạng **JPEG**.
  + Dữ liệu được chia thành hai tập chính: **Train** và **Test**.
  + Mỗi tập được tổ chức thành hai thư mục con:
    - **Normal**: Gồm các ảnh X-ray của phổi bình thường, không có dấu hiệu viêm phổi.
    - **Pneumonia**: Gồm các ảnh X-ray của bệnh nhân bị viêm phổi, đã được gán nhãn rõ ràng.
* **Đặc điểm nổi bật**:
  + **Chất lượng hình ảnh**: Các ảnh X-ray có độ phân giải cao và thường được xử lý sơ bộ để tối ưu hóa chất lượng dữ liệu, giúp các mô hình học sâu dễ dàng trích xuất đặc trưng.
  + **Tính đa dạng**: Ảnh được thu thập từ nhiều nguồn y tế khác nhau, giúp dataset có tính tổng quát cao khi áp dụng cho các trường hợp lâm sàng khác nhau.
  + **Phân chia rõ ràng**: Việc tổ chức dữ liệu theo cấu trúc thư mục (Normal/Pneumonia) tạo điều kiện thuận lợi cho quá trình huấn luyện và đánh giá các mô hình phân loại.

**Bệnh Pneumonia**

Viêm phổi (Pneumonia) là một bệnh nhiễm trùng phổi do vi khuẩn, virus hoặc nấm gây ra, ảnh hưởng trực tiếp đến hệ hô hấp. Dưới đây là các thông tin cơ bản về bệnh:

* **Triệu chứng người mắc bệnh**:
  + **Sốt cao**: Bệnh nhân thường có đợt sốt cao, kèm theo ớn lạnh.
  + **Ho có đờm**: Ho dai dẳng, với đờm có thể có màu vàng hoặc xanh, biểu hiện của phản ứng viêm.
  + **Khó thở**: Do viêm phổi làm giảm khả năng trao đổi khí, gây khó thở, đặc biệt khi vận động.
  + **Đau ngực**: Đau nhói, đặc biệt khi hít thở sâu, là dấu hiệu của viêm nhiễm mô phổi.
  + **Mệt mỏi**: Sự suy giảm năng lượng và cảm giác mệt mỏi kéo dài là biểu hiện của tình trạng nhiễm trùng.
* **Ảnh hưởng của bệnh tới sức khỏe và cuộc sống**:
  + **Tác động sức khỏe**: Nếu không được chẩn đoán và điều trị kịp thời, viêm phổi có thể dẫn đến suy hô hấp, nhiễm trùng toàn thân và các biến chứng nguy hiểm khác.
  + **Chất lượng cuộc sống**: Bệnh có thể kéo dài thời gian nghỉ ngơi và ảnh hưởng tiêu cực đến khả năng thực hiện các hoạt động hàng ngày, đặc biệt ở người cao tuổi, trẻ em và những người có hệ miễn dịch yếu.
  + **Nguy cơ tử vong**: Trong những trường hợp nặng, đặc biệt là ở các nhóm nguy cơ, viêm phổi có thể dẫn đến tử vong do các biến chứng liên quan.
* **Hình thái trên hình chụp X-ray**:
  + **Ở bệnh nhân mắc viêm phổi**:
    - Xuất hiện các vùng mờ bất thường (infiltrates) thể hiện tình trạng viêm nhiễm.
    - Sự hiện diện của các vùng **consolidation** (tăng mật độ mô phổi) cho thấy sự tích tụ chất dịch hoặc tế bào viêm.
    - Các dấu hiệu không đối xứng, với các vùng tổn thương không đồng đều trên hai phổi.
  + **Ở người không mắc viêm phổi**:
    - Hình ảnh X-ray cho thấy phổi có cấu trúc rõ ràng, đồng nhất với không có các vùng mờ hoặc cô đặc bất thường.
    - Các đường viền của phổi được hiển thị rõ ràng, phản ánh một trạng thái sức khỏe bình thường.
* **Vai trò của chẩn đoán hình ảnh X-ray**:
  + **Phương pháp chẩn đoán hiệu quả**: X-ray là phương pháp chẩn đoán nhanh, không xâm lấn và có chi phí thấp, được áp dụng rộng rãi trong các cơ sở y tế.
  + **Phát hiện sớm bệnh và theo dõi tiến triển**: Nhờ khả năng phát hiện sớm các dấu hiệu viêm nhiễm, X-ray hỗ trợ bác sĩ can thiệp kịp thời, từ đó cải thiện kết quả điều trị và giảm nguy cơ biến chứng.
  + **Cung cấp dữ liệu cho nghiên cứu**: Hình ảnh chất lượng cao từ X-ray là nguồn dữ liệu quý báu cho các nghiên cứu về AI và học máy, giúp phát triển các mô hình chẩn đoán tự động với độ chính xác ngày càng được nâng cao.

Với những thông tin trên, dataset "Labeled Chest X-Ray Images" và các dữ liệu liên quan đến bệnh Pneumonia cung cấp một nền tảng vững chắc cho việc xây dựng và triển khai các mô hình học sâu hỗ trợ chẩn đoán viêm phổi, góp phần nâng cao chất lượng chăm sóc y tế trong thời đại công nghệ số.

## 2. Tổng quan về Các Mô hình Học máy

### 2.1. Các Mô hình Pretrained

Các mô hình pretrained được huấn luyện trên các bộ dữ liệu khổng lồ (như ImageNet) và đã chứng minh khả năng trích xuất đặc trưng mạnh mẽ. Chúng thường được sử dụng làm điểm khởi đầu để tinh chỉnh cho các bài toán cụ thể như phân loại ảnh X-ray. Dưới đây là chi tiết về từng mô hình:

* **DenseNet121**:
  + **Kiến trúc và Đặc điểm**:  
    DenseNet121 được xây dựng dựa trên các "dense block", trong đó mỗi lớp nhận đầu vào là tập hợp tất cả các đặc trưng từ các lớp trước đó. Phương pháp này cho phép tái sử dụng thông tin một cách hiệu quả và giúp giảm thiểu hiện tượng vanishing gradient. Mô hình có tổng cộng 121 lớp, sử dụng “growth rate” để kiểm soát số lượng đặc trưng mới được sinh ra tại mỗi lớp.
  + **Ưu điểm chi tiết**:
    - **Tái sử dụng đặc trưng**: Nhờ các kết nối dày đặc, mô hình giúp giảm số lượng tham số cần thiết so với các kiến trúc truyền thống, giảm thiểu nguy cơ overfitting khi áp dụng cho tập dữ liệu nhỏ.
    - **Khả năng học sâu**: Việc kết nối trực tiếp giữa các lớp giúp thông tin lưu thông một cách mượt mà qua các lớp khác nhau, cải thiện hiệu suất tổng thể.
  + **Nhược điểm chi tiết**:
    - **Yêu cầu bộ nhớ**: Do cấu trúc kết nối phức tạp, mô hình đòi hỏi bộ nhớ lớn trong quá trình huấn luyện và triển khai.
    - **Tinh chỉnh tham số**: Cần điều chỉnh kỹ lưỡng các siêu tham số (như growth rate và số lượng lớp trong mỗi block) để phù hợp tối ưu với dữ liệu X-ray.
* **ResNet50**:
  + **Kiến trúc và Đặc điểm**:  
    ResNet50 được thiết kế với các residual blocks có chứa các “skip connections” cho phép đường truyền gradient không bị suy giảm qua nhiều lớp. Kiến trúc này sử dụng các bottleneck layers, giúp giảm số lượng tính toán cần thiết trong khi vẫn duy trì khả năng biểu diễn các đặc trưng phức tạp.
  + **Ưu điểm chi tiết**:
    - **Khả năng đào tạo các mạng sâu**: Nhờ vào cơ chế residual, ResNet50 có thể huấn luyện các mạng rất sâu mà không gặp phải vấn đề vanishing gradient, điều này rất quan trọng với các tác vụ phân loại hình ảnh phức tạp như X-ray.
    - **Độ ổn định và phổ biến**: Mô hình đã được chứng minh qua nhiều nghiên cứu và ứng dụng thực tế, đảm bảo độ tin cậy cao trong nhiều môi trường.
  + **Nhược điểm chi tiết**:
    - **Tài nguyên tính toán**: Số lượng tham số lớn và kiến trúc sâu đòi hỏi phần cứng mạnh mẽ (như GPU) để thực hiện huấn luyện trong thời gian hợp lý.
    - **Rủi ro overfitting**: Nếu không có đủ dữ liệu đa dạng, ResNet50 có thể dễ bị overfitting, đặc biệt trong các bài toán y khoa với dữ liệu hạn chế.
* **EfficientNet B0**:
  + **Tổng quan về EfficientNet**

EfficientNet là một họ mô hình **Convolutional Neural Network (CNN)** được phát triển bởi Google Brain và công bố vào năm 2019 trong bài báo "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks" ([Tan & Le, 2019](https://arxiv.org/abs/1905.11946)). Mô hình này được thiết kế để cân bằng giữa **độ chính xác** và **hiệu suất tính toán**, giúp giảm số lượng tham số nhưng vẫn đạt độ chính xác cao hơn so với các mô hình CNN truyền thống như ResNet hay Inception.

Điểm nổi bật của EfficientNet là việc áp dụng chiến lược **Compound Scaling**, giúp mở rộng mô hình theo một cách cân đối và tối ưu hơn. Trước đây, các phương pháp mở rộng mô hình thường chỉ tập trung vào một trong ba yếu tố sau:

* + - **Depth Scaling**: Tăng số tầng convolutional.
    - **Width Scaling**: Mở rộng số lượng kênh (filters) trong mỗi tầng.
    - **Resolution Scaling**: Tăng độ phân giải ảnh đầu vào.

EfficientNet kết hợp cả ba yếu tố trên theo một tỷ lệ nhất định, giúp cải thiện hiệu suất mô hình mà không làm tăng chi phí tính toán quá mức.

* + **Giới thiệu về EfficientNet B0**

EfficientNet B0 là phiên bản cơ bản nhất trong họ EfficientNet. Nó được thiết kế bằng cách sử dụng **Neural Architecture Search (NAS)** để tìm ra cấu trúc mạng tối ưu dựa trên tập dữ liệu ImageNet. So với các mô hình CNN phổ biến khác, EfficientNet B0 đạt độ chính xác cao hơn nhưng sử dụng ít tham số hơn. Một số thông số chính của EfficientNet B0:

* + - **Số tham số**: 5.3 triệu (ResNet-50 có khoảng 25 triệu).
    - **Top-1 Accuracy trên ImageNet**: 77.1% (so với 76.2% của ResNet-50).
    - **Kích thước ảnh đầu vào**: 224x224 pixels.
    - **Kiến trúc**: Dựa trên Mobile Inverted Bottleneck Convolution (MBConv) từ MobileNetV2.

Nhờ sự tối ưu về mặt kiến trúc, EfficientNet B0 có thể chạy tốt trên các thiết bị có tài nguyên tính toán hạn chế như smartphone và laptop mà vẫn đảm bảo hiệu suất cao.

* + **Ưu và nhược điểm của EfficientNet B0**

**Ưu điểm**

* + - **Hiệu quả về mặt tính toán**: So với ResNet hoặc VGG, EfficientNet B0 đạt độ chính xác cao hơn trong khi sử dụng ít tham số hơn.
    - **Dễ dàng triển khai trên thiết bị có tài nguyên hạn chế**: Có thể chạy trên CPU hoặc GPU tầm trung.
    - **Khả năng generalize tốt**: Hiệu quả với nhiều tác vụ thị giác máy tính như phân loại ảnh, nhận diện vật thể, và phân đoạn ảnh.

**Nhược điểm**

* + - **Không phù hợp với ảnh độ phân giải cao**: Ảnh y tế thường có kích thước lớn (ví dụ 1024x1024), cần xử lý trước khi đưa vào mô hình.
    - **Cần fine-tune để đạt hiệu suất cao nhất**: EfficientNet B0 đã được huấn luyện trên ImageNet, nhưng để áp dụng vào bài toán cụ thể như dự đoán viêm phổi, cần điều chỉnh một số tham số.

Nhờ vào tính hiệu quả và khả năng mở rộng, EfficientNet B0 đã trở thành một trong những mô hình CNN phổ biến nhất cho các bài toán thị giác máy tính hiện nay.

### 2.2. Mô hình Custom CNN

* **Kiến trúc và Đặc điểm**:  
  Mô hình Custom CNN được thiết kế từ đầu với cấu trúc phù hợp với đặc trưng của ảnh X-ray. Một kiến trúc điển hình có thể bao gồm:
  + **Các lớp Convolutional**: Nhằm trích xuất các đặc trưng không gian từ ảnh, với số lượng bộ lọc (filters) được lựa chọn sao cho phù hợp với độ phức tạp của ảnh X-ray.
  + **Lớp Pooling**: Thường sử dụng max pooling để giảm kích thước không gian của đặc trưng, giúp giảm số lượng tham số và giảm tính phức tạp của mô hình.
  + **Lớp Batch Normalization**: Giúp ổn định quá trình huấn luyện bằng cách chuẩn hóa các đầu ra của các lớp trước đó.
  + **Activation Function**: Sử dụng các hàm kích hoạt như ReLU để tăng khả năng phi tuyến của mô hình.
  + **Lớp Dropout**: Để giảm thiểu overfitting bằng cách ngẫu nhiên bỏ qua một số nút trong quá trình huấn luyện.
  + **Lớp Fully Connected**: Ở phần cuối của mạng, nhằm chuyển đổi các đặc trưng đã trích xuất thành đầu ra cho bài toán phân loại nhị phân (Normal vs. Pneumonia).
* **Ưu điểm chi tiết**:
  + **Tùy chỉnh cao**: Kiến trúc có thể được thiết kế và điều chỉnh theo đặc trưng cụ thể của ảnh X-ray, cho phép mô hình học các đặc trưng riêng biệt từ dữ liệu.
  + **Tối ưu hóa tài nguyên**: Có thể xây dựng một mô hình nhẹ hơn, phù hợp với các môi trường tính toán hạn chế, nếu dữ liệu huấn luyện không quá lớn.
  + **Kiểm soát tham số**: Người phát triển có thể tự do điều chỉnh số lượng lớp, kích thước kernel, và các siêu tham số khác để đạt hiệu suất tối ưu.
* **Nhược điểm chi tiết**:
  + **Phụ thuộc vào thiết kế**: Hiệu suất của mô hình Custom CNN phụ thuộc rất nhiều vào thiết kế kiến trúc ban đầu. Nếu không được thiết kế tốt, mô hình có thể không khai thác hết được thông tin từ dữ liệu.
  + **Quá trình thử nghiệm và tinh chỉnh**: Việc tối ưu hóa mô hình yêu cầu quá trình thử nghiệm và tinh chỉnh liên tục, đòi hỏi nhiều thời gian và kinh nghiệm thực tiễn.
  + **So sánh với pretrained**: Mô hình Custom CNN thường không có lợi thế của việc học từ dữ liệu khổng lồ như các mô hình pretrained, vì vậy, nếu dữ liệu huấn luyện không đủ phong phú, hiệu năng có thể không bằng các mô hình đã được huấn luyện trước.

### 2.3. So sánh tổng quan

* **DenseNet121**:
  + **Ưu điểm**: Tái sử dụng đặc trưng mạnh mẽ, hiệu quả trong việc giảm thiểu overfitting, hiệu suất cao trong các bài toán phức tạp.
  + **Nhược điểm**: Yêu cầu bộ nhớ lớn và cần điều chỉnh tham số cẩn thận cho phù hợp với đặc trưng của ảnh X-ray.
* **ResNet50**:
  + **Ưu điểm**: Khả năng học sâu vượt trội nhờ các residual blocks, ổn định và được kiểm chứng qua nhiều nghiên cứu.
  + **Nhược điểm**: Đòi hỏi tài nguyên tính toán cao, dễ gặp vấn đề overfitting nếu không có dữ liệu đa dạng.
* **EfficientNet B0**:
  + **Ưu điểm**: Hiệu quả tính toán và mở rộng linh hoạt, phù hợp cho ứng dụng thời gian thực.
  + **Nhược điểm**: Cần quá trình tinh chỉnh chi tiết để tối ưu hóa trên tập dữ liệu cụ thể và chưa có nhiều ứng dụng rộng rãi trong y khoa.
* **Custom CNN**:
  + **Ưu điểm**: Linh hoạt trong thiết kế, có thể tối ưu hóa riêng cho dữ liệu X-ray, yêu cầu tài nguyên tính toán thấp hơn nếu được thiết kế hợp lý.
  + **Nhược điểm**: Hiệu suất phụ thuộc vào thiết kế ban đầu, đòi hỏi nhiều công sức trong quá trình thử nghiệm và tối ưu hóa.

## 3. Công nghệ & Công cụ Hỗ trợ

### 3.1. Các công nghệ và công cụ sử dụng

* **Ngôn ngữ lập trình**
  + **Python**: Ngôn ngữ chính được sử dụng cho toàn bộ dự án, từ huấn luyện mô hình cho đến phát triển ứng dụng web.
* **Framework và Thư viện Học Sâu**:
  + **TensorFlow/PyTorch**: Hai framework học sâu được sử dụng để xây dựng, huấn luyện và tinh chỉnh các mô hình như DenseNet121, ResNet50, EfficientNet B0 và mô hình Custom CNN.
  + **Keras**: Sử dụng như một API cấp cao của TensorFlow để đơn giản hóa quá trình xây dựng và huấn luyện mô hình.
* **Thư viện Xử lý Dữ liệu và Hình Ảnh**:
  + **NumPy** và **Pandas**: Dùng để xử lý và phân tích dữ liệu, chuyển đổi và thao tác với mảng số liệu.
  + **OpenCV**, **ImageGenerator**: Dùng để xử lý ảnh, thực hiện các thao tác tiền xử lý như resize, chuyển đổi màu sắc, và data augmentation.
  + **Matplotlib** và **Seaborn**: Dùng để trực quan hóa dữ liệu và kết quả huấn luyện mô hình.
* **Phát triển Ứng dụng Web**:
  + **Flask**: Framework Python dùng để xây dựng backend cho ứng dụng web, tạo API cho việc upload ảnh và dự đoán kết quả.
  + **HTML/CSS/JavaScript/React**: Dùng để xây dựng giao diện người dùng (frontend) thân thiện, dễ sử dụng.
* **Cơ sở Dữ liệu**:
  + **SQL (MySQL)**: Dùng để lưu trữ lịch sử dự đoán, thông tin ảnh và các logs liên quan đến quá trình vận hành của hệ thống.
* **Công cụ Hỗ trợ Phát Triển và Triển Khai**:
  + **Jupyter Notebook**: Dùng để thử nghiệm, trực quan hóa kết quả và phân tích dữ liệu trong quá trình phát triển mô hình.
  + **Git**: Hệ thống quản lý phiên bản giúp theo dõi sự thay đổi của code và hỗ trợ làm việc theo nhóm.

### 3.2. Lý do lựa chọn và vai trò của các công nghệ, công cụ.

* **Python**:  
   Python là lựa chọn hàng đầu trong lĩnh vực khoa học dữ liệu và học máy nhờ vào cú pháp dễ học, cộng đồng hỗ trợ rộng lớn và hàng loạt thư viện chuyên dụng. Nó cho phép tích hợp liền mạch giữa các giai đoạn tiền xử lý dữ liệu, huấn luyện mô hình và phát triển ứng dụng.
* **TensorFlow/PyTorch & Keras**:  
   Các framework này được đánh giá cao về khả năng xây dựng và triển khai các mô hình học sâu. TensorFlow và PyTorch có hỗ trợ GPU mạnh mẽ, giúp rút ngắn thời gian huấn luyện. Keras, với giao diện đơn giản, cho phép nhanh chóng thử nghiệm các kiến trúc mô hình mà không phải lo lắng quá nhiều về các chi tiết kỹ thuật phức tạp.
* **NumPy, Pandas, OpenCV & ImageGenerator**:  
   NumPy và Pandas là những thư viện cốt lõi trong xử lý dữ liệu, giúp chuyển đổi và xử lý dữ liệu một cách hiệu quả. OpenCV và Pillow cung cấp các công cụ mạnh mẽ để xử lý ảnh, từ việc chuẩn hóa kích thước ảnh đến tăng cường dữ liệu (data augmentation), điều cần thiết để cải thiện hiệu suất của các mô hình học sâu.
* **Matplotlib & Seaborn**:  
   Các thư viện này hỗ trợ trực quan hóa dữ liệu và kết quả mô hình, giúp phân tích và hiểu rõ hơn về hiệu quả của mô hình qua các biểu đồ, từ đó dễ dàng đưa ra các điều chỉnh cần thiết.
* **Flask và Frontend Technologies (HTML, CSS, JavaScript, React)**:  
   Việc xây dựng ứng dụng web yêu cầu một backend mạnh mẽ để xử lý các yêu cầu từ người dùng và tích hợp mô hình dự đoán. Flask và Django là những framework Python linh hoạt, dễ triển khai và có thể mở rộng. Đối với giao diện người dùng, các công nghệ frontend như HTML, CSS và JavaScript (hoặc React) giúp tạo ra trải nghiệm người dùng thân thiện, cho phép tải ảnh và hiển thị kết quả dự đoán một cách trực quan.
* **Cơ sở Dữ liệu (SQL/NoSQL)**:  
   Việc lưu trữ lịch sử dự đoán và các thông tin liên quan đến ảnh yêu cầu một hệ thống quản lý dữ liệu hiệu quả. Cơ sở dữ liệu SQL (như MySQL hay PostgreSQL) hay NoSQL (như MongoDB) đều cung cấp khả năng lưu trữ, truy vấn và bảo mật dữ liệu tốt, đảm bảo rằng thông tin được lưu trữ một cách có tổ chức và an toàn.
* **Jupyter Notebook, Git & Docker**:  
   Jupyter Notebook là công cụ lý tưởng để thử nghiệm và trực quan hóa dữ liệu, giúp các nhà nghiên cứu nhanh chóng nhận diện các vấn đề trong quá trình huấn luyện mô hình. Git giúp quản lý phiên bản code, hỗ trợ làm việc nhóm và duy trì lịch sử thay đổi của dự án. Docker tạo ra môi trường nhất quán cho việc triển khai ứng dụng, giảm thiểu các vấn đề liên quan đến sự khác biệt giữa môi trường phát triển và môi trường sản xuất.

# CHƯƠNG II. QUY TRÌNH CDIO

## A. Conceive

### 1. Phân tích yêu cầu và nhu cầu ứng dụng

Theo báo cáo bệnh lao toàn cầu năm 2022 của WHO, ước tính có khoảng 10,6 triệu người mắc lao và 1,6 triệu người tử vong do lao trên toàn thế giới. Trong khi đó, viêm phổi được ước tính gây ra hơn 700.000 ca tử vong mỗi năm trên toàn cầu theo UNICEF. Tại Việt Nam, hai bệnh này vẫn là vấn đề y tế nghiêm trọng với số ca mắc và tử vong không nhỏ. Phương pháp chẩn đoán truyền thống dựa vào triệu chứng lâm sàng và chụp X-quang ngực, nhưng việc đọc và phân tích hình ảnh X-quang đòi hỏi chuyên môn cao và có thể gặp sai sót do yếu tố con người. Do đó, nhu cầu phát triển các hệ thống hỗ trợ chẩn đoán tự động dựa trên hình ảnh X-quang ngực ngày càng tăng, nhằm hỗ trợ bác sĩ trong việc phát hiện và chẩn đoán viêm phổi một cách nhanh chóng và chính xác.

### 2. Xác định bài toán: Chẩn đoán Pneumonia từ ảnh X-ray

Bài toán đặt ra là xây dựng một hệ thống sử dụng kỹ thuật học máy, đặc biệt là deep learning với CNN, nhằm hỗ trợ các chuyên gia đưa ra một kết quả nhanh hơn, chính xác hơn,…

Cấu trúc bộ dữ liệu: bao gồm 5.856 hình ảnh X-quang ngực đã được xác thực, được chia thành tập huấn luyện và tập kiểm tra độc lập, với các nhãn tương ứng: bình thường, viêm phổi. Ngoài ra, nhằm tăng tính đa dạng và tính chính xác của dataset chúng em đã thêm 1 số cách chức năng ở bước tiền xử lý dữ liệu làm thay đổi góc nhìn, phóng to hoặc thu nhỏ ảnh.

### 3. Đặt ra mục tiêu và tiêu chí đánh giá

Hiện nay, để hỗ trợ chẩn đoán bệnh phổi hiệu quả, nghiên cứu đã áp dụng các kỹ thuật học sâu, đặc biệt là các mô hình CNN tiêu biểu như Resnet, DenseNet, CNN customers, MobileNet và ResNet50, để phân loại ảnh X-quang phổi. Các mô hình này được áp dụng kỹ thuật học chuyển tiếp tinh chỉnh mô hình sau đó được huấn luyện và so sánh để tìm ra mô hình có độ chính xác cao nhất. Ngoài ra, việc sử dụng kỹ thuật học chuyển giao (transfer learning) cũng giúp cải thiện hiệu suất của mô hình khi dữ liệu huấn luyện hạn chế.

**\* Hướng tiếp cận:**

Với mục đích tiếp cận các chuyên gia lĩnh vực y tế nói riêng và những người không chuyên về IT nói chung, thì chúng em quyết định tạo một trang web với giao diện đơn giản nhằm đơn giản hóa các thao tác.

**\* Tiêu chí đánh giá:**

Để đánh giá hiệu suất của mô hình phân loại, các tiêu chí sau thường được sử dụng:

- Độ chính xác (Accuracy): Tỷ lệ dự đoán đúng trên tổng số dự đoán.

- Độ nhạy (Recall) hoặc Tỷ lệ dương tính thực sự (Sensitivity): Tỷ lệ mẫu dương tính được dự đoán đúng trên tổng số mẫu dương tính thực sự.

- Độ đặc hiệu (Specificity): Tỷ lệ mẫu âm tính được dự đoán đúng trên tổng số mẫu âm tính thực sự.

- Giá trị dự đoán dương tính (Precision): Tỷ lệ mẫu dương tính thực sự trên tổng số mẫu được dự đoán là dương tính.

- Điểm F1 (F1-score): Trung bình điều hòa của Precision và Recall, đặc biệt hữu ích khi dữ liệu không cân bằng.

- Việc sử dụng các tiêu chí này giúp đánh giá toàn diện hiệu suất của mô hình, đảm bảo khả năng phân loại chính xác và đáng tin cậy trong việc chẩn đoán viêm phổi từ ảnh X-quang.

## B. Design

### 1. Thiết kế kiến trúc tổng quan của hệ thống

1. **Kiến trúc tổng quan**

Gồm các thành phần chính :

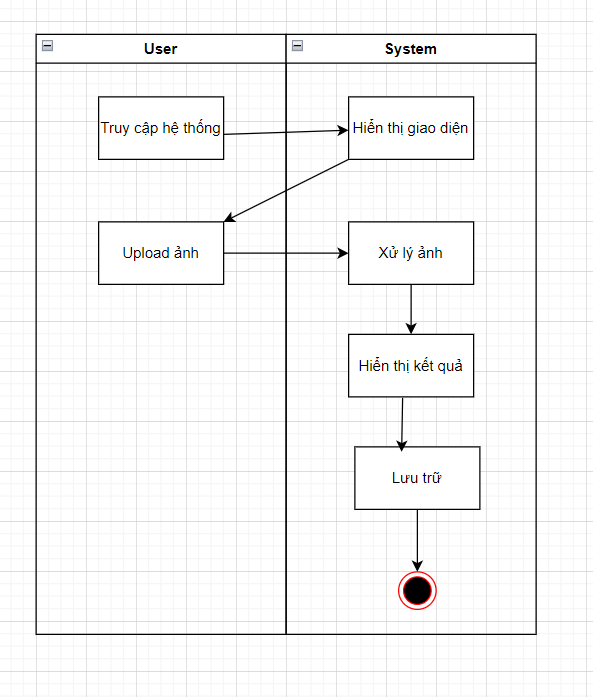
• Frontend (React.js): Giao diện web cho người dùng tải ảnh lên và nhận kết quả.

• Backend (Flask/FastAPI): Xử lý yêu cầu từ frontend, chuyển ảnh vào mô hình AI và lưu thông tin vào cơ sở dữ liệu.

• Model CNN (Keras/TensorFlow): Mô hình đã được huấn luyện để phân loại ảnh thành "Pneumonia" hoặc "Normal".

• Database (MySQL): Lưu thông tin người dùng và lịch sử dự đoán.

1. **Sơ đồ**



### 2. Thiết kế quy trình xử lý dữ liệu: tiền xử lý, phân chia tập dữ liệu (training/validating/testing)

Trong hệ thống nhận diện viêm phổi qua ảnh, quy trình xử lý dữ liệu đóng vai trò quan trọng nhằm đảm bảo dữ liệu đầu vào đạt chất lượng cao, từ đó giúp mô hình học máy (CNN) hoạt động hiệu quả. Quy trình được chia làm hai phần chính: tiền xử lý dữ liệu và phân chia tập dữ liệu.

#### **2.1. Tiền xử lý dữ liệu**

Quá trình tiền xử lý giúp cải thiện chất lượng ảnh, làm nổi bật các đặc trưng cần thiết cho việc nhận diện viêm phổi, giảm thiểu nhiễu và biến đổi không cần thiết trong dữ liệu. Cụ thể:

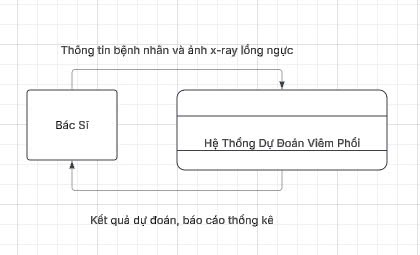
* **Thu thập và định dạng dữ liệu:**
  + **Nguồn dữ liệu:** Ảnh X-quang phổi được thu thập từ các cơ sở y tế, bệnh viện hoặc các nguồn dữ liệu công khai.
  + **Định dạng đồng nhất:** Chuyển đổi tất cả ảnh về cùng định dạng (JPEG) để đảm bảo đồng nhất khi xử lý.
* **Tiền xử lý ảnh:**
  + Cân bằng kích thước ảnh: Tất cả các ảnh được resize về một kích thước chuẩn (ví dụ: 224x224 pixels) nhằm thống nhất đầu vào cho mô hình học máy.
  + Chuẩn hóa cường độ ảnh: Giá trị pixel được chuẩn hóa (chẳng hạn, chia cho 255) để đưa về khoảng giá trị [0, 1], giúp mô hình dễ dàng học được các đặc trưng.
* **Data Augmentation:**
  + Áp dụng các phép biến đổi như xoay ảnh, lật ngang, thay đổi độ sáng, phóng to/thu nhỏ để tạo ra các biến thể của ảnh gốc. Điều này giúp mở rộng tập dữ liệu và giảm hiện tượng overfitting khi huấn luyện mô hình.

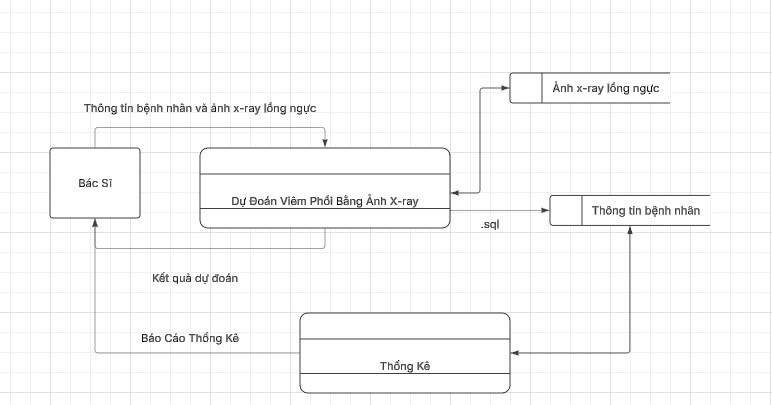
#### **2.2. Phân chia tập dữ liệu**

Để đảm bảo hiệu quả huấn luyện và đánh giá mô hình, dữ liệu cần được phân chia thành các tập riêng biệt:

**Chia theo tỷ lệ:**

* + **Tập huấn luyện (Training Set):** Chiếm khoảng 70-80% dữ liệu, dùng để huấn luyện mô hình.
  + **Tập xác thực (Validation Set):** Chiếm khoảng 10-15% dữ liệu, dùng để điều chỉnh siêu tham số và kiểm tra hiệu quả của mô hình trong quá trình huấn luyện.
  + **Tập kiểm tra (Test Set):** Chiếm khoảng 10-15% dữ liệu, dùng để đánh giá hiệu suất cuối cùng của mô hình sau khi huấn luyện xong.

**2.3 Sơ đồ DFD:**

**

**

### 3. Thiết kế giao diện ứng dụng website (UI/UX)

* File app.py (Back-end):

|  |
| --- |
| *from flask import Flask, request, render\_template, jsonify*  *from PIL import Image*  *import numpy as np*  *from tensorflow import keras*  *import mysql.connector*  *app = Flask(\_\_name\_\_)*  *model = keras.models.load\_model('densenet121\_model\_finetune.keras')*  *db\_config = {*  *'host': 'localhost', # Change if using a remote server*  *'user': 'root', # Replace with your MySQL username*  *'password': '123456', # Replace with your MySQL password*  *'database': 'pneunomia'*  *}*  *def get\_prediction\_text(pred):*  *"""*  *Hàm chuyển kết quả dự đoán (0 hoặc 1) thành văn bản.*  *0: Normal, 1: Pneumonia*  *"""*  *if pred == 0:*  *return "Normal"*  *elif pred == 1:*  *return "Pneumonia"*  *else:*  *return "Invalid Prediction"*  *@app.route('/')*  *def index():*  *return render\_template('index.html')*  *@app.route('/predict', methods=['POST'])*  *def predict():*  *try:*  *ID = request.form.get("ID")*  *name = request.form.get("name")*  *surname = request.form.get("surname")*  *gioi\_tinh = request.form.get("gioi\_tinh")*  *do\_tuoi = int(request.form.get("do\_tuoi"))*  *phone\_number = request.form.get("phone\_number")*  *city = request.form.get("city")*  *# Kiểm tra file ảnh được gửi lên qua request.files*  *if 'image' not in request.files:*  *return render\_template('index.html', result="Không tìm thấy file ảnh trong yêu cầu.")*  *file = request.files['image']*  *if file.filename == "":*  *return render\_template('index.html', result="Không có file ảnh được chọn.")*  *# Mở file ảnh bằng Pillow*  *image = Image.open(file)*  *# Nếu ảnh không ở chế độ RGB*  *if image.mode != "RGB":*  *image = image.convert("RGB")*  *# Resize ảnh theo kích thước mà model yêu cầu (ví dụ: 224x224)*  *image = image.resize((224, 224))*    *# Chuyển ảnh sang mảng NumPy và chuẩn hóa về khoảng [0,1]*  *image\_array = np.array(image).astype('float32') / 255.0*  *# Thêm dimension cho batch: shape (1, 224, 224, 3)*  *image\_array = np.expand\_dims(image\_array, axis=0)*  *# Dự đoán với model*  *# Giả sử model trả về xác suất cho lớp "Pneumonia" ở vị trí đầu tiên*  *y\_pred\_probs = model.predict(image\_array)*  *# Áp dụng threshold 0.5 để chuyển đổi xác suất thành nhãn 0 hoặc 1*  *pred\_class = 1 if y\_pred\_probs[0][0] >= 0.5 else 0*  *# Chuyển kết quả dự đoán thành văn bản*  *result\_text = get\_prediction\_text(pred\_class)*  *image.save(f"D:/cdio3/images/Predict\_{result\_text}\_{ID}.png")*  *conn = mysql.connector.connect(*  *host="localhost",*  *user="root",*  *password="123456",*  *database="pneunomia"*  *)*  *cursor = conn.cursor()*  *sql = """INSERT INTO Users (ID,name, surname, gioi\_tinh, do\_tuoi, phone\_number, city, prediction\_label)*  *VALUES (%s,%s, %s,%s, %s, %s, %s, %s)"""*  *values = (ID,name, surname, gioi\_tinh, do\_tuoi, phone\_number, city, result\_text)*  *cursor.execute(sql, values)*  *conn.commit()*  *cursor.close()*  *conn.close()*  *except Exception as e:*  *return jsonify({"result": f"Có lỗi xảy ra: {e}"})*  *# Trả về kết quả dưới dạng JSON để frontend có thể xử lý*  *return render\_template('index.html', result=result\_text)*  *if \_\_name\_\_ == '\_\_main\_\_':*  *app.run(debug=True)* |

### 4. Thiết kế hệ thống lưu trữ lịch sử dự đoán (Database)

**-** Tạo database

|  |
| --- |
| *USE pneunomia;*  *CREATE TABLE users (*  *ID VARCHAR(10) PRIMARY KEY,*  *name VARCHAR(50),*  *surname VARCHAR(50),*  *gioi\_tinh boolean;*  *do\_tuoi INT;*  *phone\_number VARCHAR(15),*  *city VARCHAR(50),*  *prediction\_label VARCHAR(10),*  *created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP*  *);*  *show tables;*  *desc users;* |

## C. Implement

## 1. DenseNet121:

### 1. Chuẩn bị dữ liệu:

Tiền xử lý và phân chia dữ liệu:

|  |
| --- |
| *main\_path = "/content/drive/MyDrive/chest\_xray"*  *train\_path = os.path.join(main\_path,"train")*  *test\_path=os.path.join(main\_path,"test")*  *train\_normal = glob.glob(train\_path+"/NORMAL/\*.jpeg")*  *train\_pneumonia = glob.glob(train\_path+"/PNEUMONIA/\*.jpeg")*  *test\_normal = glob.glob(test\_path+"/NORMAL/\*.jpeg")*  *test\_pneumonia = glob.glob(test\_path+"/PNEUMONIA/\*.jpeg")*  *train\_list = [x for x in train\_normal]*  *train\_list.extend([x for x in train\_pneumonia])*  *df\_train = pd.DataFrame(np.concatenate([['Normal']\*len(train\_normal) , ['Pneumonia']\*len(train\_pneumonia)]), columns = ['class'])*  *df\_train['image'] = [x for x in train\_list]*  *test\_list = [x for x in test\_normal]*  *test\_list.extend([x for x in test\_pneumonia])*  *df\_test = pd.DataFrame(np.concatenate([['Normal']\*len(test\_normal) , ['Pneumonia']\*len(test\_pneumonia)]), columns = ['class'])*  *df\_test['image'] = [x for x in test\_list]*  *train\_df, val\_df = train\_test\_split(df\_train, test\_size = 0.20, random\_state = SEED, stratify = df\_train['class'])*  *train\_datagen = ImageDataGenerator(*  *rescale=1/255.,*  *zoom\_range = 0.1,*  *rotation\_range = 0.1,*  *width\_shift\_range = 0.1,*  *height\_shift\_range = 0.1)*  *val\_datagen = ImageDataGenerator(rescale=1/255.)*  *ds\_train = train\_datagen.flow\_from\_dataframe(*  *train\_df,*  *x\_col = 'image',*  *y\_col = 'class',*  *target\_size = (IMG\_SIZE, IMG\_SIZE),*  *class\_mode = 'binary',*  *batch\_size = BATCH,*  *seed = SEED)*  *ds\_val = val\_datagen.flow\_from\_dataframe(*  *val\_df,*  *x\_col = 'image',*  *y\_col = 'class',*  *target\_size = (IMG\_SIZE, IMG\_SIZE),*  *class\_mode = 'binary',*  *batch\_size = BATCH,*  *seed = SEED)*  *ds\_test = val\_datagen.flow\_from\_dataframe(*  *df\_test,*  *x\_col = 'image',*  *y\_col = 'class',*  *target\_size = (IMG\_SIZE, IMG\_SIZE),*  *class\_mode = 'binary',*  *batch\_size = 1,*  *shuffle = False)* |

### 2.Huấn luyện mô hình:

**Bước 1:Khởi tạo mô hình Pretrained:**

Sử dụng DenseNet121 với trọng số từ ImageNet, loại bỏ phần fully-connected cuối cùng (include\_top=False) để phù hợp với bài toán phân loại nhị phân.

|  |
| --- |
| *from tensorflow.keras.applications import DenseNet121*  *base\_model = DenseNet121(weights='imagenet', include\_top=False, input\_shape=(IMG\_SIZE, IMG\_SIZE, 3)) #IMG\_SIZE =224*  *base\_model.trainable = False* |

* + *Mục đích:* Khai thác các đặc trưng đã học từ tập dữ liệu lớn ImageNet, giúp tăng hiệu quả nhận diện mà không cần huấn luyện lại từ đầu.

·**Xây dựng kiến trúc mô hình:**

|  |
| --- |
| *from tensorflow.keras import layers, models, optimizers*  *model = models.Sequential([*  *base\_model,*  *layers.GlobalAveragePooling2D(),*  *layers.Dense(256, activation='relu'),*  *layers.Dropout(0.5),*  *layers.Dense(1, activation='sigmoid')])* |

Tạo mô hình Sequential với các lớp bổ sung sau base model:

* + **GlobalAveragePooling2D:** Rút trích các đặc trưng tổng hợp từ feature maps.
  + **Dense Layer (256 neurons, ReLU):** Giúp học các đặc trưng phức tạp hơn từ kết quả pooling.
  + **Dropout (0.5):** Giảm hiện tượng overfitting bằng cách ngẫu nhiên bỏ qua một số neuron trong quá trình huấn luyện.
  + **Dense Layer (1 neuron, Sigmoid):** Lớp đầu ra cho bài toán phân loại nhị phân (viêm phổi và không viêm phổi).

**Bước 2: Quy trình huấn luyện ban đầu:**

**a.Compile mô hình:**

|  |
| --- |
| *model.compile(optimizer=optimizers.Adam(learning\_rate=1e-4),*  *loss='binary\_crossentropy',*  *metrics=['Recall'])* |

Sử dụng optimizer Adam với learning rate 1e-4 và loss function là binary\_crossentropy cho bài toán phân loại nhị phân. Ngoài ra, metric được theo dõi là Recall, nhằm đảm bảo mô hình không bỏ sót các ca bệnh dương tính.

**b.Huấn luyện mô hình:** Huấn luyện trên tập dữ liệu huấn luyện (ds\_train) và xác nhận trên tập dữ liệu validation (ds\_val) trong 50 epochs.

|  |
| --- |
| *history = model.fit(*  *ds\_train,*  *epochs=50,*  *validation\_data=ds\_val)* |

**c. Fine-tuning mô hình:**

·**Mở khóa các lớp:**

**S**au quá trình huấn luyện ban đầu, mở khóa (unfreeze) 71 layer cuối cùng của mô hình để cho phép huấn luyện thêm, từ đó tinh chỉnh các trọng số của mô hình Pretrained cho phù hợp với bài toán cụ thể.

|  |
| --- |
| *for layer in model.layers[-71:]:*  *layer.trainable = True* |

·**Giảm learning rate:**Để tránh làm mất đi các trọng số đã được học từ trước và đảm bảo quá trình fine-tuning ổn định, giảm learning rate xuống 1e-5.

|  |
| --- |
| *model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=1e-5),*  *loss='binary\_crossentropy',*  *metrics=['Recall'])* |

**d**· **Tiếp tục huấn luyện (Fine-tuning):**Huấn luyện thêm 10 epochs với các callback như early stopping và checkpoint để dừng huấn luyện sớm khi không cải thiện và lưu lại mô hình tốt nhất.

|  |
| --- |
| *from tensorflow.keras.callbacks import EarlyStopping*  *early\_stopping = EarlyStopping(*  *monitor='val\_loss',*  *patience=5,*  *restore\_best\_weights=True)*  *from tensorflow.keras.callbacks import ModelCheckpoint*  *checkpoint = ModelCheckpoint(*  *'/content/drive/MyDrive/chest\_xray/densenet121\_model\_finetune.keras',*  *monitor='val\_loss',*  *save\_best\_only=True,*  *mode='min'*  *)*  *model.fit(ds\_train,*  *epochs=10,*  *validation\_data=ds\_val,*  *callbacks=[early\_stopping, checkpoint])* |

### 3. Đánh giá hiệu năng:

Accuracy: 0.9679

Recall: 0.9872

Precision: 0.9625

F1-score: 0.9747

## 2.EfficientNet B0:

### 1.Chuẩn bị dữ liệu

|  |
| --- |
| *import numpy as np*  *import os*  *import tensorflow as tf*  *from tensorflow.keras import layers, models*  *from tensorflow.keras.applications.efficientnet import EfficientNetB0, preprocess\_input*  *from sklearn.metrics import confusion\_matrix, classification\_report*  *import matplotlib.pyplot as plt*  *import seaborn as sns*    *base\_dir = "/content/drive/MyDrive/Colab Notebooks/chest\_xray"*  *train\_dir = os.path.join(base\_dir, "train")*  *test\_dir = os.path.join(base\_dir, "test")img\_size = (224, 224) # Kích thước phù hợp với EfficientNetB0*  *num\_classes = 2 # Số lớp nhãn*  *# Check if the dataset is balanced*  *def get\_class\_counts(base\_dir):*  *counts = {}*  *for dir in os.listdir(base\_dir):*  *counts[dir] = len(os.listdir(os.path.join(base\_dir, dir)))*  *return counts*    *class\_counts = get\_class\_counts(train\_dir)*  *print("Class counts:", class\_counts)* |

- Hàm ***get\_class\_counts*** lặp qua các thư mục con (NORMAL và PNEUMONIA) để đếm số lượng ảnh trong mỗi lớp, giúp đánh giá xem dữ liệu có cân bằng hay không.

|  |
| --- |
| *# Data augmentation and preprocessing functions*  *data\_augmentation = tf.keras.Sequential([*  *layers.RandomRotation(0.2),*  *layers.RandomTranslation(0.2, 0.2),*  *layers.RandomZoom(0.2),*  *layers.RandomFlip("horizontal")*  *])*    *def augment\_and\_preprocess(image, label): # Chỉ apply cho tập train*  *image = tf.cast(image, tf.float32)*  *image = data\_augmentation(image)*  *image = preprocess\_input(image)*  *return image, label*    *def preprocess\_only(image, label): # Chỉ apply cho tập test*  *image = tf.cast(image, tf.float32)*  *image = preprocess\_input(image)*  *return image, label*    *# Data loading*  *train\_ds = tf.keras.utils.image\_dataset\_from\_directory(*  *train\_dir,*  *labels="inferred",*  *label\_mode="binary",*  *batch\_size=32,*  *image\_size=img\_size,*  *shuffle=True, # Bật xáo trộn*  *seed=123,*  *color\_mode="rgb" # Mode color RGB (X-ray chỉ là ảnh đen trắng)*  *)*    *test\_ds = tf.keras.utils.image\_dataset\_from\_directory(*  *test\_dir,*  *labels="inferred",*  *label\_mode="binary",*  *batch\_size=32,*  *image\_size=img\_size,*  *seed=123,*  *color\_mode="rgb"*  *)*  *# Caching and prefetching*  *AUTOTUNE = tf.data.AUTOTUNE*  *train\_ds = train\_ds.cache().prefetch(buffer\_size=AUTOTUNE)*  *test\_ds = test\_ds.cache().prefetch(buffer\_size=AUTOTUNE)*  *# Mapping dữ liệu*  *train\_ds\_aug = train\_ds.map(augment\_and\_preprocess, num\_parallel\_calls=AUTOTUNE)*  *test\_ds\_pre = test\_ds.map(preprocess\_only, num\_parallel\_calls=AUTOTUNE)* |

- Tạo một pipeline ***data augmentation*** tăng cường dữ liệu với các phép biến đổi ngẫu nhiên (xoay, dịch, zoom, lật) nhằm làm phong phú tập dữ liệu và giúp mô hình học được các đặc trưng đa dạng.

- Hàm tiền xử lý ***augment\_and\_preprocess***: Chuyển đổi kiểu dữ liệu, áp dụng các phép tăng cường, và chuẩn hóa ảnh (cho tập train) theo chuẩn của EfficientNet B0 (thông qua preprocess\_input).

- Hàm tiền xử lý ***preprocess\_only***: Được dùng cho tập test để chỉ chuẩn hóa mà không áp dụng tăng cường dữ liệu.

- Tạo các dataset cho huấn luyện và kiểm tra từ cấu trúc thư mục, tự động gán nhãn theo tên thư mục

- Caching và Prefetching: Giúp tăng tốc độ đọc dữ liệu và giảm thời gian chờ của quá trình huấn luyện.

- Map: Áp dụng các hàm ***augment\_and\_preprocess*** cho tập huấn luyện và ***preprocess\_only*** cho tập kiểm tra, đảm bảo mỗi ảnh được xử lý đúng cách.

### 2.Xây dựng Mô hình EfficientNet B0

|  |
| --- |
| *# Model building*  *def build\_efficientnetb0\_model(activation="sigmoid", learning\_rate=1e-3):*  *base\_model = EfficientNetB0(*  *include\_top=False,*  *weights="imagenet",*  *input\_shape=(img\_size[0], img\_size[1], 3)*  *)*  *base\_model.trainable = False*  *x = base\_model.output*    *# Global AveragePooling2D layer*  *x = layers.GlobalAveragePooling2D()(x)*  *x = layers.BatchNormalization()(x)*  *x = layers.Dropout(0.5)(x)*  *outputs = layers.Dense(1, activation="sigmoid")(x)*  *loss = "binary\_crossentropy"*    *model = models.Model(inputs=base\_model.input, outputs=outputs)*  *model.compile(*  *optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),*  *loss=loss,*  *metrics=["accuracy"]*  *)*  *return model, base\_model* |

- Base Model: Sử dụng pre-trained model *EfficientNetB0* từ keras với *include\_top=False* để loại bỏ lớp phân loại gốc của *ImageNet*, giữ lại phần trích xuất đặc trưng.

- Đóng băng trọng số: *base\_model.trainable = False* giữ các trọng số đã học trên ImageNet khi huấn luyện phần đầu tiên.

- Xây dựng phần head: Thêm *Global Average Pooling, Batch Normalization* và *Dropout* để tránh overfitting, sau đó thêm lớp Dense cuối cùng với activation *sigmoid* (với binary\_crossentropy) cho bài toán phân loại nhị phân.

- Biên dịch mô hình: Sử dụng *Adam optimizer* với learning rate đã định, loss và metric.

### 3.Huấn luyện và Đánh giá Mô hình ban đầu

|  |
| --- |
| *early\_stopping = tf.keras.callbacks.EarlyStopping(*  *monitor='val\_loss', # Đánh giá bằng val\_loss*  *patience=5, # Dừng nếu 5 epochs liên tiếp không cải thiện*  *mode='min',*  *restore\_best\_weights=True, # Trả về weight của epoch train tốt nhất*  *verbose=1*  *)*  *def train\_and\_evaluate(model, train\_data, test\_data, epochs=20):*  *history = model.fit(*  *train\_data,*  *epochs=epochs,*  *validation\_data=test\_data,*  *callbacks=[early\_stopping ]*  *)*    *plt.figure(figsize=(12, 5))*  *plt.subplot(1, 2, 1)*  *plt.plot(history.history['loss'], label='Train Loss')*  *plt.plot(history.history['val\_loss'], label='Val Loss')*  *plt.title('Loss')*  *plt.legend()*    *plt.subplot(1, 2, 2)*  *plt.plot(history.history['accuracy'], label='Train Accuracy')*  *plt.plot(history.history['val\_accuracy'], label='Val Accuracy')*  *plt.title('Accuracy')*  *plt.legend()*  *plt.show()*    *# Compute evaluation metrics*  *y\_true = []*  *y\_pred = []*    *for images, labels in test\_data:*  *preds = model.predict(images)*  *preds = (preds > 0.5).astype(int)*  *y\_true.extend(labels.numpy().flatten())*  *y\_pred.extend(preds.flatten())*    *# Confusion matrix and classification report*  *cm = confusion\_matrix(y\_true, y\_pred)*  *print("Confusion Matrix:")*  *print(cm)*  *plt.figure(figsize=(6,5))*  *sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')*  *plt.xlabel('Predicted')*  *plt.ylabel('True')*  *plt.title('Confusion Matrix')*  *plt.show()*    *print("\nClassification Report:")*  *report = classification\_report(y\_true, y\_pred)*  *print(report)*    *return history* |

- Đặt ***early\_stopping***  nếu 5 epochs liên tiếp val\_loss không giảm xuống đẻ tránh huấn luyện quá lâu.

- Thực hiện ***train\_and\_evaluate*** : Huấn luyện mô hình với dữ liệu đã tăng cường, lưu *history* và sau đó vẽ đồ thị các chỉ số huấn luyện và đánh giá mô hình trên tập test bằng cách tính *confusion matrix* và in ra báo cáo phân loại.

|  |
| --- |
| *model\_b0\_sigmoid\_lr\_low, base\_model = build\_efficientnetb0\_model(activation="sigmoid", learning\_rate=1e-3)*  *history\_b0\_sigmoid\_lr\_low = train\_and\_evaluate(model\_b0\_sigmoid\_lr\_low, train\_ds\_aug, test\_ds\_pre, epochs=15)* |

- Khởi tạo model B0 và trả về *model\_bo\_sigmoid\_lr\_low* để *train\_and\_evaluate* và *base\_model* để sử dụng fine-tune sau khi train phần head.

**Bước 4: Fine-tuning Mô hình**

|  |
| --- |
| *base\_model.trainable = True*  *# Xác định số tầng để unfreeze*  *fine\_tune\_at = int(len(base\_model.layers) \* 0.8)*  *# Đóng băng lại tất cả các lớp trước fine\_tune\_at, mở khóa phần còn lại*  *for layer in base\_model.layers[:fine\_tune\_at]:*  *layer.trainable = False*  *for layer in base\_model.layers[fine\_tune\_at:]:*  *layer.trainable = True*    *# Compile lại mô hình với learning rate nhỏ hơn*  *model\_b0\_sigmoid\_lr\_low.compile(*  *optimizer=tf.keras.optimizers.Adam(learning\_rate=1e-5),*  *loss="binary\_crossentropy",*  *metrics=["accuracy"]*  *)*    *lr\_scheduler = tf.keras.callbacks.ReduceLROnPlateau(*  *monitor="val\_loss",*  *factor=0.5,*  *patience=3,*  *min\_lr=1e-6*  *)* |

- Mở khóa các tầng của base mode: Sau giai đoạn huấn luyện ban đầu, mở khóa các tầng trong phần cuối của base model (20% cuối cùng) để fine-tune các trọng số, giúp mô hình học thêm các đặc trưng phù hợp với dữ liệu X-ray.

- Tái biên dịch mô hình với Learning Rate thấp hơn:

+ Việc giảm learning rate (từ 1e-3 xuống 1e-5) khi fine-tuning giúp tránh việc cập nhật quá mạnh đối với các trọng số đã được tinh chỉnh.

+ Callback *ReduceLROnPlateau* tự động giảm 50% *learning rate* mỗi lần nếu không thấy cải thiện về *val\_loss* sau 3 epochs liên tục nhưng không được thấp hơn 1e-6 nhằm tránh việc học quá chậm hoặc dừng hẳn.

|  |
| --- |
| *history\_finetune = model\_b0\_sigmoid\_lr\_low.fit(*  *train\_ds\_aug,*  *epochs=30,*  *callbacks=[early\_stopping, lr\_scheduler],*  *validation\_data=test\_ds\_pre,*  *initial\_epoch=5*  *)* |

- Tiếp tục huấn luyện đến 30 epochs (bắt đầu từ epoch 5 bởi vì quá trình train trước đó gặp early\_stopping tại epoch 5) với các tầng được mở khóa để fine-tune mô hình.

- Sử dụng cùng các callbacks để giám sát quá trình huấn luyện.

|  |
| --- |
| *model\_save\_path = "/content/drive/MyDrive/Colab Notebooks/best\_model.keras"*  *model\_b0\_sigmoid\_lr\_low.save(model\_save\_path)*  *print(f"Model saved to: {model\_save\_path}")* |

- Thực hiện lưu model sau fine-tune và vẽ báo cáo *confusion matrix, classification report.*

Kết quả cho ra:

*accuracy: 0.9539 - loss: 0.1283 -* ***val\_accuracy: 0.9263*** *-* ***val\_loss: 0.1958*** *- learning\_rate: 1e-5*

## 3.ResNet50:

### 1.Chuẩn bị dữ liệu

* + Tiền xử lý dataset: chuẩn hóa, tăng cường dữ liệu (data augmentation)
  + Phân chia tập dữ liệu:
* **Tập huấn luyện (Train set):** Dùng để huấn luyện mô hình
* **Tập kiểm tra(Validation set):** Dùng để điều chỉnh hyperparameter
* **Tập kiểm chứng(Test set):** Dùng để đánh giá hiệu suất của mô hình

### 2.Huấn luyện mô hình

* Cài đặt và cấu hình các mô hình pretrained:
* **ResNet50** là một mô hình mạng nơ-ron tích chập (CNN) có 50 lớp, được huấn luyện trước trên bộ dữ liệu ImageNet. Mô hình này giúp tận dụng các đặc trưng đã học từ hàng triệu hình ảnh, thay vì huấn luyện lại từ đầu.

|  |
| --- |
| *base\_model = ResNet50(weights="imagenet", include\_top=False, input\_shape=(224, 224, 3))*  *for layer in base\_model.layers[:100]: #Freeze 100 layer đầu*  *layer.trainable = False*  *for layer in base\_model.layers[100:]: # Fine-tune từ layer 100 trở đi*  *layer.trainable = True* |

* Fine-tune:
* Đóng băng 100 lớp đầu tiên:
* Nhận diện các đăng trung cơ bản (đường nét, góc cạnh, màu sắc)
* Giữ nguyên các đặc trưng và không cần huấn luyện lại
* Fine-tune từ lớp thứ 100 trở đi:
* Các lớp sâu sẽ chưa thông tin đặc trưng cụ thể hơn
* Việc fine-tune giúp mô hình học đặc điểm cụ thể của bài toán thay vì sử dụng đặc trưng tổng quát từ ImageNet
* Xây dựng mô hình:

|  |
| --- |
| *model = models.Sequential([*  *base\_model,*  *layers.GlobalAveragePooling2D(),*  *layers.Dense(256, activation='relu'),*  *layers.Dropout(0.5),*  *layers.Dense(1, activation='sigmoid')*  *])* |

* + GlobalAveragePooling2D()
* Global Average Pooling tính giá trị trung bình của từng kênh đặc trưng
* Giảm số lượng tham số và hạn chế overfitting
  + Dense(256,activation=’relu’)
* Lớp Dense với 256 neuron và hàm ReLU(Rectified Linear Unit)
* Tăng cường khả năng học của mô hình
  + Dropout(0.5)
* Dropout ngẫu nhiên vô hiệu hóa 50% số neuron trong quá trình huấn luyện
* Hạn chế overfitting vì mô hình sẽ ít phụ thuộc vào 1 số neuron cụ thể
  + Dense(1,activation=’sigmoid’)
* Vì đây là bài toán phân loại, đầu ra sẽ chỉ có giá trị 0 hoặc 1
* Nếu giá trị >0.5, mô hình sẽ dự đoán 1(dương tính), ngược lại là 0 (âm tính)

|  |
| --- |
| *model.compile(optimizer=optimizers.Adam(learning\_rate=1e-4),*  *loss='binary\_crossentropy',*  *metrics=['Recall'])* |

* + Tham số huấn luyện:
* **Optimizer**: Adam với learning rate=1e-4 (Giúp điều chỉnh tốc độ học theo từng tham số)
* **Loss function**: binary\_crossentropy (Phù hợp với các bài toán nhị phân)
* **Metrics**: Recall
* Được tính bằng số mẫu dương tính dự đoán đúng / Tổng số mẫu dương thực tế (Nhằm tính tỉ lệ số mẫu dương bị bỏ sót)
  + Quá trình triển khai mô hình

|  |
| --- |
| *early\_stopping = EarlyStopping(*  *monitor="val\_loss",*  *patience=5,*  *restore\_best\_weights=True)*  *reduce\_lr = ReduceLROnPlateau(*  *monitor="val\_loss",*  *factor=0.5,*  *patience=4,*  *min\_lr=2e-7,*  *verbose=1)*  *checkpoint = ModelCheckpoint(*  *"/content/drive/MyDrive/best\_model.keras",*  *monitor="val\_loss",*  *save\_best\_only=True,*  *mode="min")*  *callbacks = [early\_stopping,reduce\_lr,checkpoint]*  *history = model.fit(*  *ds\_train,*  *epochs=20,*  *validation\_data=ds\_val,*  *callbacks = callbacks)* |

* **EarlyStopping**: Dừng sớm nếu validation loss không cải thiện sau 5 epoch, đồng thời khôi phục trọng sô tốt nhất. Điều này giúp giảm overfitting
* **ReduceLROnPlateau:** Nếu validation loss không cải thiện sau 4 epoch, learning rate sẽ giảm 50% nhằm tối ưu quá trình học. Giá trị tối thiểu được đặt là 2e-7
* **ModelCheckPoint:** Lưu lại mô hình có validation loss thấp nhất, nhằm đảm bảo sử dụng mô hình tối ưu trong quá trình kiểm tra

### 3.Đánh giá hiệu năng của mô hình

* **Accuracy**: 91%
* Độ chính xác thấp hơn khi phân loại lớp ‘Pneumonia’(87%) và cao khi phân loại lơp ‘Normal’(98%), mô hình đôi khi dự đoán sai dương tính (False Positve)
* Recall cao hơn ở lớp ‘Pneumonia’(99%) nhưng khá thấp ở lớp ‘Normal’(76%), mô hình có xu hướng dự đoán dương tính cao hơn
* F1-score ở lớp ‘Pneumonia’(93%) cao hơn ‘Normal’(86%), cho thấy mô hình phân loại dương tính tốt hơn so với âm tính

## 4.CNN custom

### 1.Chuẩn bị dữ liệu

|  |
| --- |
| *def get\_data\_loaders(data\_dir, batch\_size=32, num\_workers=4):*  *train\_transform = transforms.Compose([*  *transforms.Resize((224, 224)),*  *transforms.RandomHorizontalFlip(),*  *transforms.RandomRotation(10),*  *transforms.ToTensor(),*  *transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])*  *])*  *test\_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])*  *])*  *train\_dataset = datasets.ImageFolder(root=f'{data\_dir}/train', transform=train\_transform)*  *test\_dataset = datasets.ImageFolder(root=f'{data\_dir}/test', transform=test\_transform)*  *train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True, num\_workers=num\_workers)*  *test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False, num\_workers=num\_workers)*  *return train\_loader, test\_loader* |

* *Resize ảnh về kích thước 224x224.*
* *Lật ngang ngẫu nhiên (RandomHorizontalFlip).*
* *Xoay ngẫu nhiên trong khoảng ±10 độ (RandomRotation).*
* *Chuyển đổi ảnh thành tensor (ToTensor).*
* *Chuẩn hóa giá trị pixel dựa trên trung bình (mean) và độ lệch chuẩn (std) của tập ImageNet.*
* *Không có augmentation, chỉ resize và chuẩn hóa.*
* *datasets.ImageFolder:* Dùng để tải dữ liệu ảnh từ thư mục.
* *transform=train\_transform*: Áp dụng augmentation cho tập train.
* *transform=test\_transform*: Áp dụng chuẩn hóa cho tập test.
* *shuffle=True* → Xáo trộn dữ liệu trong tập train để tránh overfitting.
* *shuffle=False* → Không xáo trộn tập test để giữ nguyên thứ tự dữ liệu.

### 2.Xây dựng mô hình CNN

|  |
| --- |
| *class CustomCNN(nn.Module):*  *def \_\_init\_\_(self):*  *super(CustomCNN, self).\_\_init\_\_()*  *self.conv\_layers = nn.Sequential(*  *nn.Conv2d(in\_channels=3, out\_channels=32, kernel\_size=3, padding=1),*  *nn.ReLU(),*  *nn.MaxPool2d(2, 2),*  *nn.Conv2d(in\_channels=32, out\_channels=64, kernel\_size=3, padding=1),*  *nn.ReLU(),*  *nn.MaxPool2d(2, 2),*  *nn.Conv2d(in\_channels=64, out\_channels=128, kernel\_size=3, padding=1),*  *nn.ReLU(),*  *nn.MaxPool2d(2, 2),*  *nn.Dropout(0.5)*  *)*  *self.fc\_layers = nn.Sequential(*  *nn.Linear(128 \* 28 \* 28, 256),*  *nn.ReLU(),*  *nn.Dropout(0.5),*  *nn.Linear(256, 2),*  *nn.Softmax(dim=1)*  *)*  *def forward(self, x):*  *x = self.conv\_layers(x)*  *x = x.view(-1, 128 \* 28 \* 28)*  *x = self.fc\_layers(x)*  *return x* |

* Tạo một lớp CNN tùy chỉnh kế thừa từ *nn.Module.*
* *Conv2d(3, 32, 3, padding=1):* Lớp tích chập với 3 kênh đầu vào (RGB), 32 kênh đầu ra, *kernel size = 3.*
* *MaxPool2d(2,2):* Giảm kích thước ảnh xuống một nửa.
* *ReLU():* Hàm kích hoạt để đưa giá trị âm về 0.
* *Dropout(0.5):* Ngăn overfitting bằng cách bỏ ngẫu nhiên 50% neurons.
* *Linear(128 \* 28 \* 28, 256):* Kết nối toàn phần từ feature map thành vector có 256 phần tử.
* *Softmax(dim=1):* Biến output thành xác suất của 2 lớp.
* *x.view(-1, 128 \* 28 \* 28):* Chuyển từ tensor 3D thành 1D để đưa vào mạng FC.

**Khởi tạo mô hình**

|  |
| --- |
| *def init\_model(device):*  *model = CustomCNN().to(device)*  *return model* |

* Tạo mô hình và chuyển nó lên GPU nếu có.

**Huấn luyện mô hình**

|  |
| --- |
| *def train(model, train\_loader, criterion, optimizer, epoch):*  *model.train()*  *total = 0*  *correct = 0*  *for batch\_idx, (data, target) in enumerate(train\_loader):*  *data, target = data.to(device), target.to(device)*  *optimizer.zero\_grad()*  *output = model(data)*  *loss = criterion(output, target)*  *loss.backward()*  *optimizer.step()*  *\_, predicted = torch.max(output.data, 1)*  *total += target.size(0)*  *correct += (predicted == target).sum().item()*  *accuracy = 100 \* correct / total*  *print(f"Train Accuracy at Epoch {epoch}: {accuracy:.2f}%")*  *return accuracy* |

* *model.train(): Chuyển model sang chế độ huấn luyện.*
* *optimizer.zero\_grad()*: Xóa gradient trước khi tính toán lại.
* *loss.backward()*: Tính gradient.
* *optimizer.step()*: Cập nhật trọng số.

### 3.Đánh giá mô hình

|  |
| --- |
| *def evaluate(model, test\_loader, criterion):*  *model.eval()*  *test\_loss = 0*  *correct = 0*  *with torch.no\_grad():*  *for data, target in test\_loader:*  *data, target = data.to(device), target.to(device)*  *output = model(data)*  *test\_loss += criterion(output, target).item()*  *pred = output.argmax(dim=1, keepdim=True)*  *correct += pred.eq(target.view\_as(pred)).sum().item()*  *test\_loss /= len(test\_loader.dataset)*  *accuracy = 100. \* correct / len(test\_loader.dataset)*  *print(f"Test Set: Average loss: {test\_loss:.4f}, Accuracy: {correct}/{len(test\_loader.dataset)} ({accuracy:.2f}%)")*  *return accuracy* |

* model.eval(): Chuyển model sang chế độ đánh giá.
* with torch.no\_grad(): Tắt tính toán gradient để tiết kiệm bộ nhớ.
* Accuracy: 0.90
* Recall: 0.82
* Precision: 0.99
* F1-score: 0.90

**3. Triển khai ứng dụng website**

* **Kiến trúc ứng dụng**

Ứng dụng web được chia thành ba phần chính:

* Frontend: Giao diện web (HTML, CSS) cho phép người dùng tải ảnh lên và xem kết quả dự đoán.
* Backend: Xử lý logic của ứng dụng, sử dụng Flask để nhận yêu cầu, xử lý ảnh, thực hiện dự đoán và lưu kết quả vào cơ sở dữ liệu.
* API: Cung cấp endpoint /predict để nhận ảnh từ người dùng, xử lý và trả về kết quả dự đoán.
* **Tích hợp model dự đoán vào backend**
* Mô hình densenet121\_model\_finetune.keras được tải vào Flask.
* Ảnh được tải lên sẽ được tiền xử lý: chuyển đổi sang RGB, resize về 224x224, chuẩn hóa dữ liệu về khoảng [0,1].
* Model dự đoán và trả về xác suất, kết quả được phân loại thành hai nhóm: "Normal" và "Pneumonia".
* Kết quả dự đoán được lưu vào cơ sở dữ liệu MySQL cùng với thông tin bệnh nhân.
* **Xây dựng giao diện**
* Một giao diện đơn giản được xây dựng bằng Flask (index.html).
* Người dùng có thể tải lên ảnh X-quang phổi.
* Kết quả dự đoán hiển thị trực tiếp trên trang sau khi xử lý.
* Ảnh được lưu trữ cùng với nhãn dự đoán.
* **Code triển khai**

Phần backend được viết bằng Flask, kết nối với MySQL để lưu trữ thông tin bệnh nhân và kết quả dự đoán.

**4. Xây dựng và tích hợp Database**

• Thiết kế database lưu trữ lịch sử các lần dự đoán và kết quả (schema, bảng dữ liệu)

• Tích hợp database với website: lưu trữ và truy xuất lịch sử dự đoán

• Kiểm thử và triển khai hệ thống

• Thực hiện kiểm thử từng module: unit test, integration test, system test

• Đánh giá hiệu năng và độ ổn định của toàn hệ thống

• Triển khai hệ thống trên môi trường thực tế (local server/ cloud)

## D. Operate

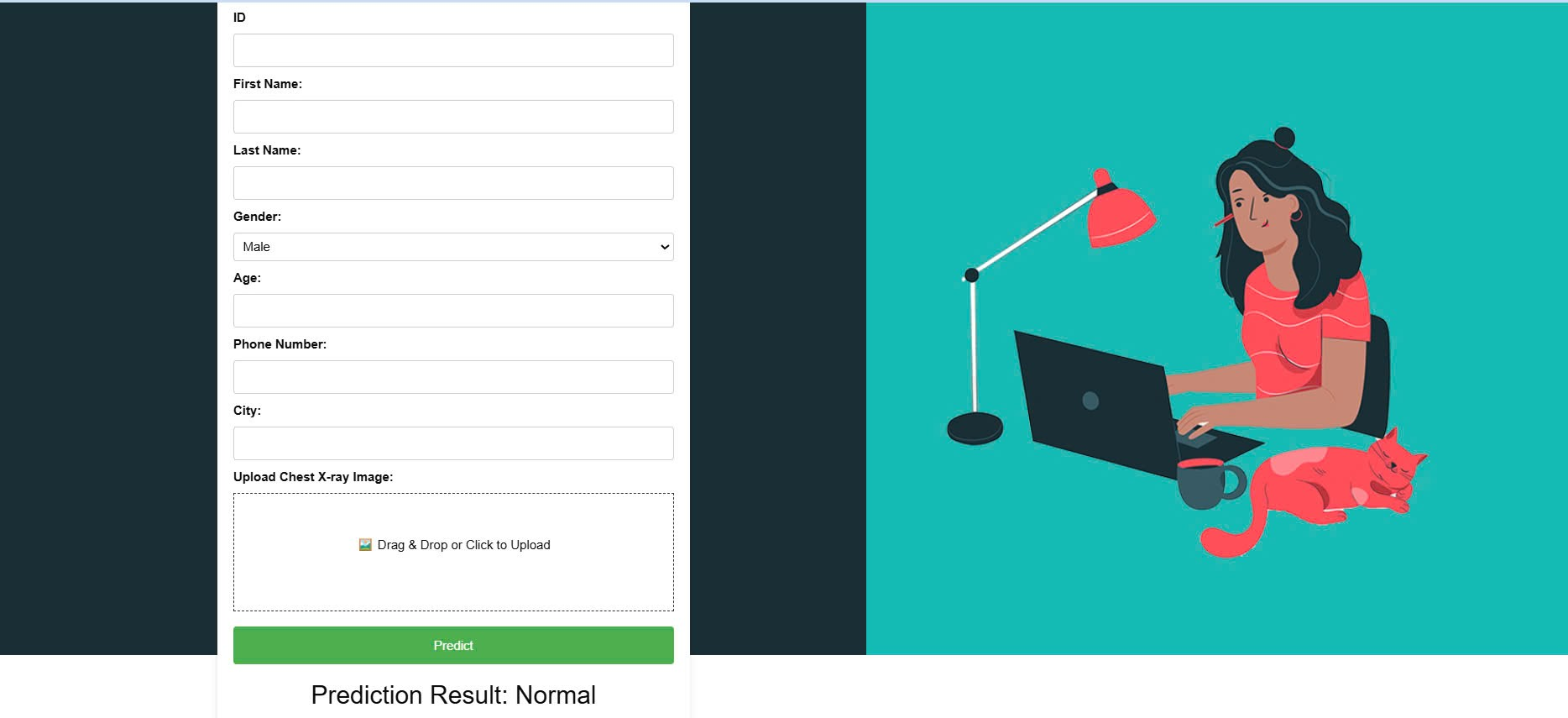
**1. Hướng dẫn sử dụng ứng dụng website**

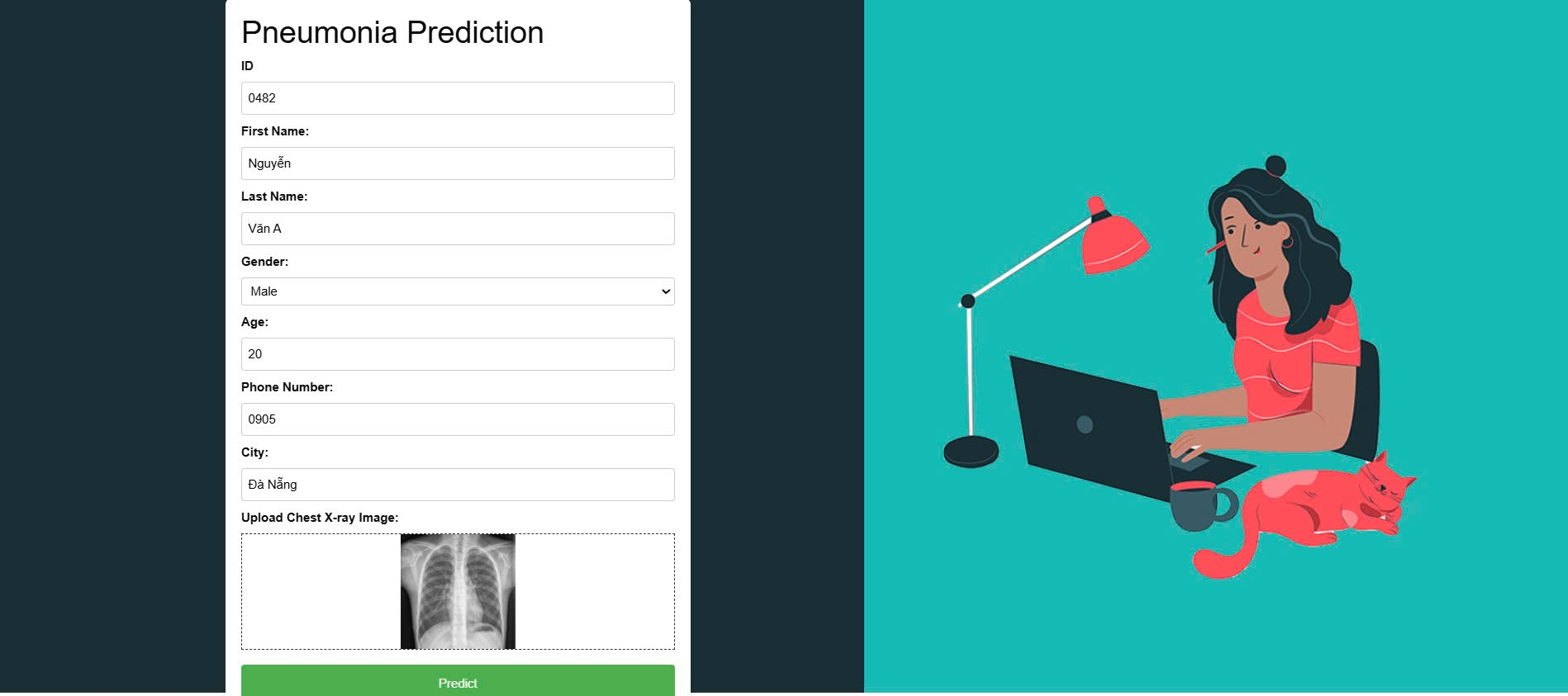
**B1:** Điền thông tin.

**B2:** Upload ảnh (có thể kéo thả)

**B3:** Nhấn nút predict.

**B4:** Đọc kết quả trả về.

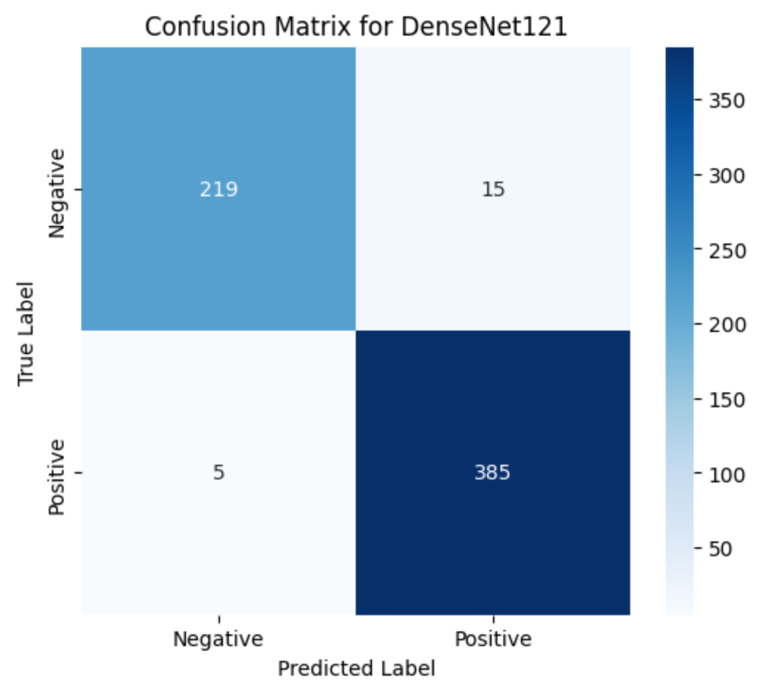
****

****

# CHƯƠNG III. KẾT QUẢ VÀ ĐÁNH GIÁ

## 1. Kết quả huấn luyện và so sánh các mô hình

### 1.1. DenseNet121

****

**Ưu điểm:**

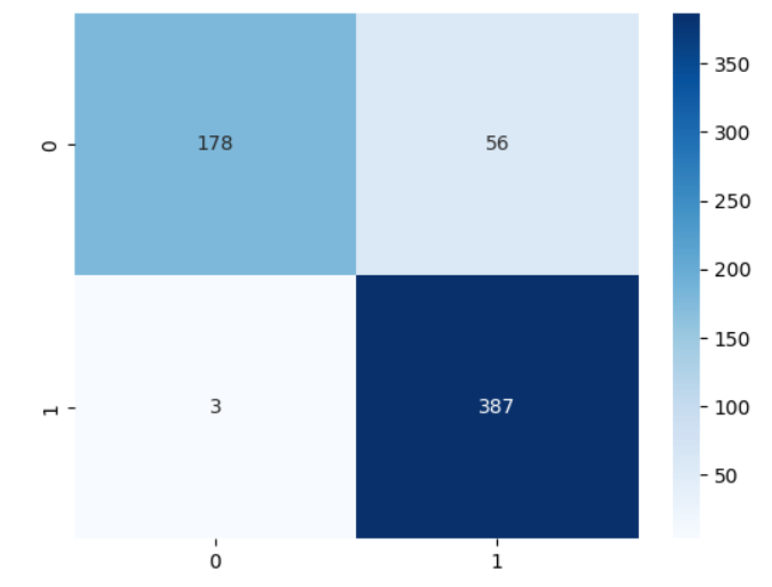
**Hiệu suất cao:**

* Accuracy ~96.79%, Precision ~96.25%, Recall ~98.72%, F1-Score ~97.50%.
* Các chỉ số này cho thấy mô hình có khả năng phân loại rất tốt, giảm thiểu cả false positive và false negative.
* Kiến trúc dense connections:
* Tận dụng thông tin từ nhiều tầng, giúp trích xuất đặc trưng hiệu quả và cải thiện quá trình huấn luyện.

**Nhược điểm:**

* Kiến trúc phức tạp có thể đòi hỏi bộ nhớ và thời gian tính toán cao, ảnh hưởng đến tốc độ huấn luyện và inference.
* Nếu dữ liệu không đủ đa dạng, mô hình phức tạp có thể dễ gặp vấn đề overfitting, đòi hỏi kỹ thuật regularization và augmentation tốt.

### 1.2. ResNet50



**Ưu điểm:**

* Recall rất cao (~99%)
* Mô hình phát hiện được hầu hết các ca bệnh, rất quan trọng trong các ứng dụng y tế nhằm không bỏ sót ca bệnh nghiêm trọng.
* Kiến trúc residual:
* Giúp giảm thiểu vấn đề gradient biến mất, cải thiện khả năng huấn luyện của các mô hình sâu.

**Nhược điểm:**

* Precision và Accuracy thấp hơn:
* Precision chỉ khoảng ~87% và Accuracy ~90.55%, cho thấy có khả năng tạo ra nhiều false positives, dẫn đến chẩn đoán sai lệch.
* Hiệu suất tổng thể không bằng DenseNet121:
* Mặc dù có Recall cao nhưng các chỉ số khác không đạt mức tối ưu, cần cân nhắc khi áp dụng trong các ứng dụng yêu cầu độ chính xác cao.

### 1.3. EfficientNetB0

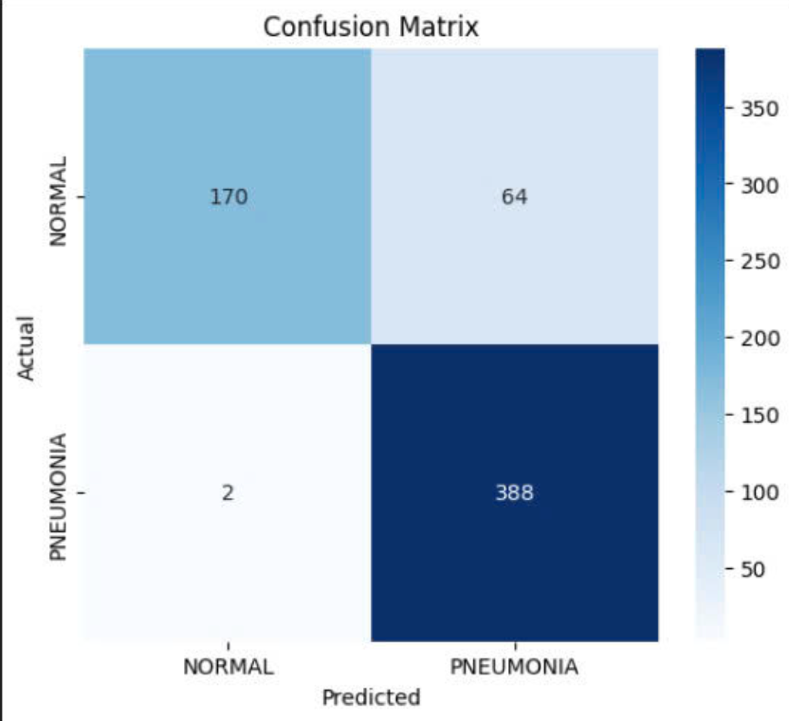
**Ưu điểm:**

* Hiệu quả tính toán:
* Kiến trúc được thiết kế để tối ưu hóa số lượng tham số và chi phí tính toán, thích hợp cho các hệ thống có hạn chế về tài nguyên.
* Cân bằng các chỉ số:
* Accuracy ~92.63%, Precision ~90.31%, Recall ~98.72%, F1-Score ~94.29% cho thấy mô hình có sự cân đối tốt giữa các chỉ số đánh giá.

**Nhược điểm:**

* Accuracy thấp hơn DenseNet121:
* Mặc dù hiệu quả nhưng có thể không đạt được mức độ chính xác tối ưu như DenseNet121 trong một số bài toán đặc thù.
* Khả năng mở rộng:
* Với phiên bản B0, mô hình đã đạt được sự cân bằng giữa hiệu năng và tài nguyên, nhưng có thể cần phiên bản nâng cấp khi dữ liệu hoặc bài toán trở nên phức tạp hơn.

### 1.4. Custom CNN

****

**Ưu điểm:**

* Tùy chỉnh theo bài toán:
* Có thể thiết kế kiến trúc phù hợp nhất với tập dữ liệu và yêu cầu cụ thể của ứng dụng.
* Chỉ số Recall ~99% và F1-Score ~95% cho thấy khả năng phân biệt tốt giữa các lớp bệnh.
* Linh hoạt trong thiết kế:
* Cho phép tối ưu hóa các tầng, hàm kích hoạt, và các tham số khác theo nhu cầu cụ thể của ứng dụng.

**Nhược điểm:**

* Khả năng tổng quát hóa:
* Mô hình custom có thể không ổn định khi áp dụng cho các tập dữ liệu khác nếu chưa được thiết kế và điều chỉnh cẩn thận.
* Yêu cầu tối ưu và kiểm thử kỹ lưỡng:
* Việc thiết kế và tinh chỉnh mô hình từ đầu đòi hỏi thời gian và công sức đáng kể để đạt được hiệu suất tối ưu như các mô hình chuẩn (DenseNet, ResNet, EfficientNet).

**Tổng kết**

* **DenseNet121** nổi trội về độ chính xác và khả năng trích xuất đặc trưng, nhưng đòi hỏi tài nguyên tính toán lớn.
* **ResNet50** có ưu thế ở Recall cao, phù hợp với các ứng dụng y tế cần phát hiện tối đa các ca bệnh, nhưng có thể gặp vấn đề với độ chính xác tổng thể.
* **EfficientNetB0** mang lại hiệu quả tính toán cao và sự cân bằng tốt giữa các chỉ số, thích hợp cho các ứng dụng yêu cầu tính toán nhẹ.
* **Custom CNN** cho phép tùy chỉnh linh hoạt và đạt được các chỉ số tốt, nhưng cần chú trọng đến khả năng tổng quát hóa và yêu cầu tối ưu hóa kỹ lưỡng.

## 2. Hiệu quả của model được triển khai trong ứng dụng:

### 2.1. Xây dựng mô hình DenseNet121:

* **Thành công trong việc triển khai mô hình:** Ứng dụng đã triển khai thành công mô hình DenseNet121 với các thông số đánh giá (ví dụ: độ chính xác, độ nhạy, độ đặc hiệu, F1-score, …)
* **Hiệu quả dự đoán:** Các chỉ số đánh giá cho thấy mô hình có khả năng nhận diện viêm phổi với mức độ chính xác cao, góp phần hỗ trợ các bác sĩ trong việc chẩn đoán sớm bệnh.

### 2.2. Quản lý dữ liệu:

* Lưu trữ thông tin: Thông tin bệnh nhân và kết quả dự đoán được lưu trữ đầy đủ trong cơ sở dữ liệu, đảm bảo an toàn và dễ dàng truy xuất khi cần.
* Quản lý hình ảnh: Đường dẫn của ảnh được lưu trên ổ đĩa, giúp việc liên kết giữa dữ liệu bệnh nhân và hình ảnh thực hiện được một cách trực quan và hiệu quả, hỗ trợ quá trình kiểm tra và đánh giá kết quả dự đoán.

### 2.3. Giao diện website:

* Tính thân thiện và dễ sử dụng: Giao diện website được thiết kế trực quan, đáp ứng đầy đủ các chức năng cần thiết trong quá trình thao tác, từ nhập liệu bệnh nhân, tải lên ảnh chụp, đến hiển thị kết quả dự đoán.
* Trải nghiệm người dùng: Các tính năng giao diện được tối ưu hóa giúp người dùng dễ dàng thao tác mà không gặp phải rào cản kỹ thuật, tạo điều kiện thuận lợi cho việc sử dụng ứng dụng trong môi trường lâm sàng.

### 2.4. Hiệu năng backend:

* Tốc độ xử lý chậm: Mặc dù các chức năng đã được tích hợp đầy đủ, nhưng backend vẫn gặp vấn đề về tốc độ nhận và phản hồi dữ liệu, gây ảnh hưởng không nhỏ đến trải nghiệm người dùng.
* Hướng cải tiến: Cần thực hiện tối ưu hóa mã nguồn, nâng cấp hạ tầng hoặc áp dụng các công nghệ mới để cải thiện tốc độ xử lý backend, đảm bảo thời gian phản hồi nhanh chóng trong quá trình vận hành ứng dụng.

**Kết luận:**

Ứng dụng nhận diện viêm phổi đã đạt được nhiều thành tựu quan trọng về mô hình học sâu, quản lý dữ liệu và giao diện người dùng. Tuy nhiên, để nâng cao hiệu quả sử dụng trong môi trường thực tế, cần tập trung cải thiện tốc độ xử lý của backend. Việc khắc phục các hạn chế này sẽ góp phần làm tăng tính khả dụng và hiệu quả của toàn bộ hệ thống.

## 3. Đánh giá tổng thể dự án: thành công và hạn chế

### 3.1. Thành công:

Hiệu suất mô hình đạt mức mong đợi:

* Các mô hình nhận diện viêm phổi sau quá trình huấn luyện và fine-tuning đã cho ra các chỉ số đo lường ấn tượng (Accuracy, Precision, Recall, F1-score), cho thấy khả năng phân biệt chính xác giữa các trường hợp viêm phổi và bình thường.
* Việc kết hợp mô hình Pretrained và Fine-tuning đã giúp tận dụng tối đa các đặc trưng học được từ tập dữ liệu lớn (ImageNet) và thích nghi tốt với bài toán chuyên biệt.
* Quy trình tiền xử lý và phân chia dữ liệu hiệu quả:
* Quy trình tiền xử lý (chuẩn hóa, tăng cường dữ liệu) và phân chia dữ liệu (training, validation, test) đã tạo ra một tập dữ liệu đồng nhất, giúp quá trình huấn luyện diễn ra ổn định và giảm hiện tượng overfitting.
* Tích hợp và triển khai thành công trong ứng dụng:
* Việc xây dựng database để lưu trữ lịch sử dự đoán và tích hợp vào website đã cho phép người dùng truy xuất thông tin dễ dàng, đồng thời cung cấp dữ liệu cần thiết để đánh giá hiệu năng hệ thống trong môi trường thực tế.
* Kiểm thử chi tiết các module (unit test, integration test, system test) đã giúp đảm bảo tính ổn định và đáp ứng hiệu quả của toàn hệ thống.

### 3.2. Hạn chế:

**Đa dạng dữ liệu:**

Tập dữ liệu hiện có có thể chưa đủ đa dạng về nguồn gốc và điều kiện chụp ảnh (ví dụ: các ảnh có chất lượng thấp, nhiễu cao) dẫn đến hạn chế trong khả năng khái quát của mô hình.

**Yêu cầu tính toán và tài nguyên:**

Quá trình huấn luyện và fine-tuning mô hình, đặc biệt với các mô hình Pretrained, đòi hỏi tài nguyên tính toán khá lớn. Điều này có thể gây khó khăn khi triển khai trên các hệ thống có hạn chế về phần cứng.

**Khả năng tối ưu hóa tham số:**

Mặc dù các chỉ số đo lường đã cho thấy hiệu suất tốt, nhưng vẫn có thể tối ưu thêm các tham số (learning rate, số epoch, batch size) để nâng cao hiệu quả và giảm thiểu quá trình huấn luyện quá lâu.

Tích hợp và bảo trì hệ thống:

Quá trình tích hợp mô hình vào ứng dụng và thiết lập database đòi hỏi sự theo dõi liên tục về hiệu năng và độ ổn định. Một số vấn đề về đồng bộ dữ liệu hoặc bảo mật có thể phát sinh khi triển khai trong môi trường thực tế.

### 3.3. Đề xuất cải tiến trong tương lai:

**Mở rộng tập dữ liệu:**

Thu thập thêm dữ liệu từ nhiều nguồn khác nhau, bao gồm cả các ảnh có điều kiện chụp khác nhau, để tăng tính đa dạng và khả năng khái quát của mô hình.

**Tối ưu hóa mô hình:**

Nghiên cứu và áp dụng các thuật toán tối ưu hóa mới, cũng như thử nghiệm các kiến trúc mô hình khác để tìm ra giải pháp hiệu quả nhất.

**Nâng cao khả năng triển khai:**

Cải tiến giao diện và tích hợp hệ thống giám sát, logging để theo dõi hiệu năng và phát hiện sớm các sự cố trong quá trình vận hành.

**Phát triển thêm chức năng:**

Mở rộng các tính năng của ứng dụng như báo cáo chi tiết kết quả dự đoán, hỗ trợ người dùng theo dõi lịch sử và cung cấp các giải pháp can thiệp kịp thời.

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 1.Tổng kết dự án

* Sử dụng 4 mô hình CNN: **DenseNet121, ResNet50, EfficientNet B0, Custom CNN** để phân loại bệnh **Pneumonia**.
* **DenseNet121** cho kết quả tối ưu nhất, đạt precision cao hơn so với các mô hình còn lại.
* Các mô hình đều đạt hiệu suất tốt nhưng vẫn có một số trường hợp phân loại sai cần cải thiện.
* Ứng dụng có tiềm năng hỗ trợ bác sĩ trong chẩn đoán bệnh **Pneumonia**.

## 2. Các bài học kinh nghiệm rút ra

* **Chất lượng dữ liệu đầu vào** ảnh hưởng đáng kể đến độ chính xác của mô hình.
* **Fine-tuning** mô hình giúp cải thiện hiệu suất đáng kể, đặc biệt với mô hình DenseNet121.
* **Data Augmentation** giúp tăng độ chính xác bằng cách cải thiện khả năng tổng quát hóa.
* **Overfitting** có thể xảy ra nếu không tối ưu dropout và batch normalization đúng cách.
* **Thời gian huấn luyện** và yêu cầu phần cứng khác nhau giữa các mô hình: **EfficientNet B0** có hiệu suất tốt với thời gian huấn luyện ngắn, trong khi **ResNet50** cần nhiều tài nguyên hơn.
* **Kết hợp nhiều mô hình** có thể giúp cải thiện độ chính xác nhưng cần tối ưu hóa ensemble learning.

## 3. Hướng phát triển và cải tiến trong tương lai

* **Mở rộng bộ dữ liệu**: Thu thập thêm dữ liệu từ nhiều nguồn khác nhau để cải thiện khả năng tổng quát hóa.
* **Cải tiến mô hình**:
  + Tiếp tục fine-tune mô hình **DenseNet121** để giảm lỗi phân loại.
  + Thử nghiệm **Hybrid models** hoặc **Ensemble Learning** kết hợp các mô hình để tăng độ chính xác.
* **Tạo hệ thống lưu trữ dữ liệu**:
  + **Cơ sở dữ liệu (MySQL, MongoDB, PostgreSQL)** để lưu trữ kết quả dự đoán và lịch sử bệnh nhân.
  + **Xây dựng API backend** để kết nối mô hình với giao diện người dùng.
* **Triển khai web ứng dụng**:
  + Xây dựng giao diện web thân thiện cho người dùng tải ảnh X-ray và nhận kết quả.
  + Triển khai mô hình trên **cloud** để dễ dàng truy cập từ xa.
* **Cải thiện explainability**:
  + Ứng dụng **Grad-CAM** để trực quan hóa vùng ảnh mà mô hình sử dụng để đưa ra quyết định.
* **Thử nghiệm các kỹ thuật mới**:
  + Kiểm tra các **Vision Transformer (ViT)** hoặc **Swin Transformer** để so sánh hiệu suất với CNN.
  + Áp dụng **self-supervised learning** để tận dụng dữ liệu chưa gán nhãn.

# TÀI LIỆU THAM KHẢO

1. [Image classification via fine-tuning with EfficientNet - Keras](https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/)

2. [Complete Architectural Details of all EfficientNet Models](https://medium.com/towards-data-science/complete-architectural-details-of-all-efficientnet-models-5fd5b736142)

3. [EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks Mingxing Tan, Quoc V. Le - Cornell University](https://arxiv.org/abs/1905.11946)

3. [Image classfication via fine-tuning with DenseNet - Keras](https://keras.io/api/applications/densenet/)