

ĐẠI HỌC KHOA HỌC TỰ NHIÊN



BÁO CÁO CUỐI KÌ MÔN PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG THÔNG TIN

KHOA: TOÁN - CƠ - TIN HỌC

NGÀNH: MÁY TÍNH VÀ KHOA HỌC THÔNG TIN

Sinh viên:

Nguyễn Văn Đoàn - 19000412
Tạ Anh Quân - 19000471

Người hướng dẫn:

Trần Hàn Thiện

Mục lục

Lời cảm ơn	1
1 Giới thiệu	2
1.1 Giới thiệu về công ty 2NF	2
1.2 Khái quát đề tài	2
1.3 Công nghệ, công cụ	3
1.3.1 Visual Studio Code	3
1.3.2 Android Studio	3
1.3.3 Github	4
1.3.4 Flutter	5
1.3.5 Ngôn ngữ lập trình Dart	6
1.3.6 Python	7
1.3.7 TFLite	8
1.3.8 Ưu và nhược điểm của từng loại công nghệ sử dụng	9
2 Thiết kế chương trình	11
2.1 Xây dựng giao diện mobile	11
2.1.1 Widget trong Flutter	11
2.1.2 Màn hình chính	12
2.1.3 Màn hình lịch sử dự đoán	13
2.2 Xây dựng mô hình CNN	14
2.2.1 Tập dữ liệu sử dụng	14
2.2.2 Mô hình học sâu VGG16	15
2.2.3 Mô hình học sâu Resnet50	16
2.2.4 Mô hình học sâu InceptionV3	17
2.2.5 Huấn luyện mô hình và kết quả thu được	18
2.3 Tích hợp mô hình CNN vào ứng dụng mobile	19
2.3.1 Chuyển đổi file h5 sang TFLite	19
2.3.2 Tích hợp thư viện TFLite	19
2.3.3 Xây dựng lớp Classifier	19
2.4 Cách hoạt động của chương trình	21
3 Thiết kế sơ đồ UML	22
3.1 Biểu đồ use case	22
3.2 Biểu đồ hoạt động	23
3.2.1 Hoạt động của app	23
3.2.2 Xây dựng và huấn luyện mô hình	23
3.3 Biểu đồ trạng thái	24
3.4 Biểu đồ tuần tự	24
3.5 Class diagram	25
4 Kết quả và hướng phát triển	25
4.1 Kết quả	25
4.2 Hướng phát triển	27
4.3 Bài học kinh nghiệm sau kỳ thực tập	28

5 Phân công công việc	28
6 Tài liệu tham khảo	28

Lời cảm ơn

Đầu tiên chúng em xin cảm ơn đến khoa Toán – Cơ – Tin học trường Đại học Khoa học Tự nhiên – ĐHQGHN đã đưa bộ môn “Phân tích và thiết kế hệ thống thông tin” vào chương trình giảng dạy. Đây là môn học có tính thực tế cao, giúp sinh viên có cơ hội tiếp cận với doanh nghiệp, học hỏi và phát triển bản thân với mục tiêu nghề nghiệp tương lai.

Tiếp đến, chúng em xin gửi lời cảm ơn chân thành nhất đến thầy Vũ Tiến Dũng và cô Nguyễn Thị Bích Thủy đã giúp chúng em kết nối với công ty, liên tục giúp đỡ và hỗ trợ chúng em trong quá trình thực tập để hoàn thành tốt báo cáo.

Đặc biệt, chúng em cũng xin gửi lời cảm ơn chân thành nhất đến các anh chị trong CÔNG TY 2NF đã nhiệt tình hướng dẫn, tạo điều kiện thuận lợi cho chúng em trong suốt quá trình thực tập. Chúng em xin giành lời cảm ơn đặc biệt tới anh Trần Hàn Thiện và anh Nguyễn Quý Minh là những người đã trực tiếp hướng dẫn nhiệt tình và giải đáp các thắc mắc cho chúng em trong suốt kỳ thực tập vừa qua. Quá trình thực tập tuy ngắn nhưng nhờ sự giúp đỡ của các anh và mọi người khác trong công ty chúng em đã học được rất nhiều điều, không chỉ những kiến thức về lập trình, về kĩ thuật, công nghệ mà còn được biết tới những kĩ năng cần có khi làm việc trong một môi trường thực tế như kĩ năng giao tiếp, làm việc nhóm,... những phẩm chất cần có của một nhân viên công ty

Cuối cùng, đề tài “ Xây dựng ứng dụng di động nhận diện viêm phổi thông qua ảnh X quang” được hoàn thành dựa trên sự nỗ lực của tất cả các thành viên trong nhóm. Mặc dù đã có nhiều cố gắng nhưng bài báo cáo khó tránh khỏi những thiếu sót, kính mong được sự thông cảm và chỉ bảo của thầy cô và các bạn.

Xin trân trọng cảm ơn!

1 Giới thiệu

1.1 Giới thiệu về công ty 2NF

2NF là công ty phần mềm chuyên cung cấp giải pháp công nghệ thông tin và các sản phẩm phần mềm cho các doanh nghiệp Nhật Bản, Hàn Quốc, Úc, ...

Địa chỉ: Tầng 9 tòa nhà 3A, ngõ 82 Duy Tân, Dịch Vọng, Cầu Giấy, Hà Nội. Hiện tại 2NF group gồm có 2 công ty: 2NF Software có trụ sở tại Hà Nội và 2NF có trụ sở tại Tokyo với tổng nhân sự là 100 nhân viên.

Công ty đã phát triển cho khách hàng một số dịch vụ:

- Mạng xã hội
- Hệ thống phần mềm quản lý doanh nghiệp, nhà hàng: quản lý sản xuất quản lý chuỗi nhà hàng.
- Hệ thống quản lý nông nghiệp.
- Hệ thống quản lý học tập, thi.
- Hệ thống thương mại điện tử và các hệ thống khác theo nhu cầu khách hàng.

Môi trường làm việc tại công ty tốt, thoải mái trang thiết bị đầy đủ, các anh chị trong công ty nhiệt tình, gần gũi, hàng tuần có anh leader tổ chức một buổi seminar để chia sẻ kinh nghiệm trong ngành cũng như trong cuộc sống,...

1.2 Khái quát đề tài

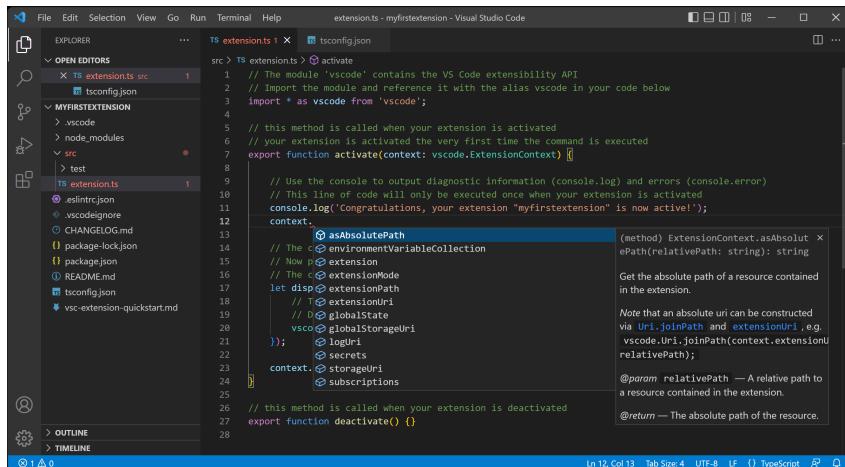
Mặc dù hiện nay đã có nhiều tiến bộ trong lĩnh vực chuẩn đoán và điều trị, viêm phổi vẫn là một trong những bệnh lý gây ra tình trạng phức tạp trong việc chăm sóc sức khỏe và tài chính cho người bệnh và gia đình, đặc biệt khi Covid-19 vẫn còn diễn biến phức tạp. Theo số liệu của UNICEF và WHO, viêm phổi đã giết 2 triệu trẻ em mỗi năm, nhiều hơn số bệnh nhân tử vong do AIDS, sốt rét cộng lại. Tại Việt Nam, hàng năm có khoảng 2.9 triệu lượt trẻ mắc và hơn 4000 trẻ em tử vong mỗi năm do viêm phổi, do vậy nước ta được xem là 1 trong 15 quốc gia đối diện với hiểm họa do bệnh viêm phổi nhiều nhất thế giới. Do đó, nhóm đã lên ý tưởng xây dựng một mobile app có chức năng xác định viêm phổi bằng ảnh X quang nhằm giúp bác sĩ trong quá trình chuẩn đoán bệnh.

1.3 Công nghệ, công cụ

Các công nghệ, công cụ được sử dụng trong suốt thời gian thực tập: Thời gian đầu, mentor đã hướng dẫn thực tập sinh tìm hiểu về các công cụ sẽ giúp ích trong công việc sau này. Một số công cụ đó là: ngôn ngữ lập trình Dart, Framework Flutter, Python - lập trình ứng dụng, Skype - ứng dụng trao đổi trực tuyến, Android Studio, Visual code - trình soạn thảo mã nguồn.

1.3.1 Visual Studio Code

Visual Studio Code là trình chỉnh sửa mã nguồn nhẹ nhưng mạnh mẽ chạy trên máy tính và có sẵn cho Windows, macOS và Linux. Nó đi kèm với hỗ trợ tích hợp cho JavaScript, TypeScript và Nodejs, đồng thời có hệ sinh thái mở rộng phong phú cho các ngôn ngữ và thời gian chạy khác (chẳng hạn như C++, C#, Java, Python, PHP, Go, .NET). Các tính năng bao gồm hỗ trợ gỡ lỗi, đánh dấu cú pháp, hoàn thành mã thông minh, đoạn mã, tái cấu trúc mã và Git nhúng. Người dùng có thể thay đổi chủ đề, phím tắt, tùy chọn và cài đặt tiện ích mở rộng thêm chức năng.



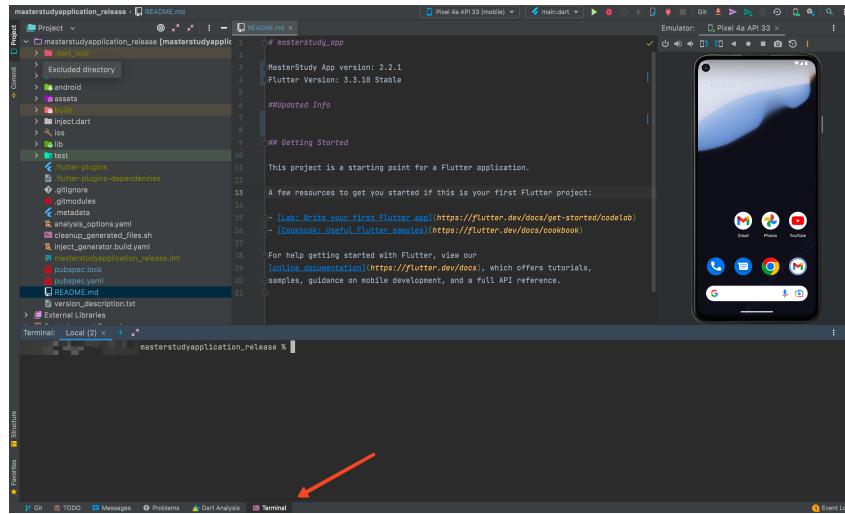
Hình 1: Ảnh minh họa Vscode

1.3.2 Android Studio

Android Studio là Môi trường phát triển tích hợp (IDE) chính thức để phát triển ứng dụng Android. Nhờ có công cụ cho nhà phát triển và trình soạn thảo mã mạnh mẽ của IntelliJ IDEA, Android Studio cung cấp thêm nhiều tính năng giúp bạn nâng cao năng suất khi xây dựng ứng dụng Android, chẳng hạn như:

- Một hệ thống xây dựng linh hoạt dựa trên Gradle.
- Một trình mô phỏng nhanh và nhiều tính năng.

- Tính năng Live Edit (Chỉnh sửa trực tiếp) để cập nhật các thành phần kết hợp trong trình mô phỏng và thiết bị thực theo thời gian thực.
- Tích hợp sẵn tính năng hỗ trợ Google Cloud Platform, giúp dễ dàng tích hợp Google Cloud Messaging và App Engine.



Hình 2: Ảnh minh họa Android Studio

1.3.3 Github

GitHub là một dịch vụ nổi tiếng cung cấp kho lưu trữ mã nguồn Git cho các dự án phần mềm. Github có đầy đủ những tính năng của Git, ngoài ra nó còn bổ sung những tính năng về social để các developer tương tác với nhau. Flutter bao gồm hai phần quan trọng:

- SDK (Bộ công cụ phát triển phần mềm): Là một tập hợp các công cụ giúp bạn phát triển các ứng dụng của mình.
- Framework (Thư viện giao diện người dùng dựa trên các tiện ích con): Tập hợp các phần giao diện người dùng mà lập trình viên có thể tái sử dụng (nút, đầu vào văn bản, thanh trượt, v.v.), từ đó có thể cá nhân hóa cho nhu cầu của riêng mình.



Hình 3: Github

1.3.4 Flutter

Flutter là một khung nguồn mở do Google phát triển và hỗ trợ. Các nhà phát triển frontend và fullstack sử dụng Flutter để xây dựng giao diện người dùng (UI) của ứng dụng cho nhiều nền tảng chỉ với một nền mã duy nhất. Tại thời điểm ra mắt vào năm 2018, Flutter chủ yếu hỗ trợ phát triển ứng dụng di động. Hiện nay, Flutter hỗ trợ phát triển ứng dụng trên sáu nền tảng: iOS, Android, web, Windows, MacOS và Linux.

Flutter sử dụng ngôn ngữ lập trình nguồn mở Dart, ngôn ngữ này cũng do Google phát triển. Dart được tối ưu hóa để xây dựng UI và nhiều điểm mạnh của Dart được sử dụng trong Flutter.

Trong Flutter, các nhà phát triển sử dụng các widget để xây dựng bộ cục UI. Điều này có nghĩa là mọi thứ mà người dùng nhìn thấy trên màn hình, từ cửa sổ và bảng điều khiển đến các nút và văn bản, đều được tạo ra từ các widget.

Các widget Flutter được thiết kế để các nhà phát triển có thể dễ dàng tùy chỉnh chúng. Flutter đạt được điều này thông qua cách tiếp cận thành phần. Điều này có nghĩa là hầu hết các widget được tạo thành từ các widget nhỏ hơn và các widget cơ bản nhất đều có những mục đích cụ thể. Điều này cho phép các nhà phát triển kết hợp hoặc chỉnh sửa các widget để tạo ra những widget mới.

Flutter kết xuất các widget bằng công cụ đồ họa của riêng mình thay vì dựa vào các widget tích hợp sẵn của nền tảng. Theo cách này, người dùng sẽ trải nghiệm giao diện tương tự trong ứng dụng Flutter trên các nền tảng. Cách tiếp cận này cũng mang lại sự linh hoạt cho các nhà phát triển vì một số widget Flutter có thể thực hiện các chức năng mà những widget theo nền tảng không thể thực hiện được.

Flutter cũng giúp việc sử dụng các widget do cộng đồng phát triển trở

nên dễ dàng. Kiến trúc của Flutter hỗ trợ tạo ra nhiều thư viện widget và Flutter khuyến khích cộng đồng xây dựng và duy trì các thư viện widget mới.



Hình 4: Ảnh minh họa Flutter

1.3.5 Ngôn ngữ lập trình Dart

Dart là ngôn ngữ lập trình cho Flutter- bộ công cụ giao diện người dùng của Google để xây dựng các ứng dụng Mobile, Web và Desktop app đẹp, được biên dịch nguyên bản từ một cơ sở mã code duy nhất.

Để code các chương trình bằng ngôn ngữ Dart, lập trình viên có thể sử dụng một IDE ví dụ như IntelliJ, Android Studio, Visual Studio Code, ... thậm chí đơn giản nhất là có thể vào trình duyệt web và mở trang dartpad.dev để chạy các mã code của mình.

Sự nổi bật của Dart so với các loại ngôn ngữ khác:

- Dart là một ngôn ngữ lập trình do Google phát triển. Dart là một static type language nên nó là AOT (Ahead of Time) giống với các ngôn ngữ C, C++, Java, ... compile xong rồi mới chạy. Nhưng nó cũng là JIT (Just in Time) giống như các dynamic type language khác như Python, JavaScript,
- Khi phát triển ứng dụng thì Dart sử dụng JIT để hỗ trợ hot load nhanh chóng, còn khi build release thì dùng AOT để tối ưu hiệu năng như một native code. Ngoài ra Dart cũng hướng tới việc trở thành một ngôn ngữ có thể chạy trên nhiều platform khác nhau, nó cũng có máy ảo làm nhiệm vụ dịch source code sang bytecode như Java.

Cấu trúc dữ liệu của ngôn ngữ lập trình Dart:

- Cấu trúc dữ liệu liệt kê enum: biểu diễn một tập hợp cố định các hằng số.
- Cấu trúc dữ liệu Collection (Danh sách – Mảng – List): Trong Dart danh sách cũng là mảng được định nghĩa từ lớp generic List, chứa một tập hợp dữ liệu, xác định từ phần tử thứ 0, truy cập đến mảng (danh sách) dùng ký hiệu [] chứa chỉ số phần tử.

- Cấu trúc dữ liệu Map: là kiểu tập hợp dữ liệu mà mỗi phần tử biểu diễn theo cặp key: value.
- Cấu trúc tập hợp – Set: là 1 danh sách các phần tử chỉ xuất hiện 1 lần.
- Cấu trúc hàng đợi - Queue: dạng danh sách tuân theo nguyên tắc FIFO (First in first out).
- Cấu trúc ngăn xếp – Stack: dạng danh sách tuân theo nguyên tắc LIFO (Last in first out).
- Ngoài ra còn một số cấu trúc khai triển từ Collection như: HashMap, HashSet, LinkedList,..



Hình 5: Dart

1.3.6 Python

Python là một ngôn ngữ lập trình thông dịch, dễ học và hiệu quả, đã trở thành một trong những ngôn ngữ phổ biến nhất trong lĩnh vực lập trình và phân tích dữ liệu. Với cú pháp đơn giản, cộng đồng phát triển mạnh mẽ và các thư viện phong phú, Python đã trở thành công cụ không thể thiếu trong lĩnh vực học máy.

Python cung cấp các thư viện và framework mạnh mẽ như NumPy, Pandas, Scikit-learn và TensorFlow, giúp xây dựng, huấn luyện và triển khai các mô hình học máy một cách dễ dàng. NumPy cung cấp các cấu trúc dữ liệu và hàm tính toán hiệu năng cao, Pandas cho phép xử lý và phân tích dữ liệu một cách linh hoạt, trong khi Scikit-learn cung cấp các thuật toán học máy cơ bản và công cụ đánh giá mô hình.

Python cũng là ngôn ngữ chủ đạo trong TensorFlow, một thư viện học sâu mạnh mẽ được sử dụng rộng rãi trong lĩnh vực học máy. TensorFlow cung cấp khả năng xây dựng và huấn luyện các mạng neural phức tạp, từ mạng neural cơ bản đến mô hình học sâu như CNN, RNN và Transformer.

Ứng dụng của Python trong học máy là vô cùng đa dạng. Python được sử dụng để phân tích và xử lý dữ liệu, xây dựng mô hình dự đoán, phân loại và

gom cụm dữ liệu, nhận dạng và xử lý hình ảnh, và thậm chí trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP). Các công cụ và thư viện hỗ trợ của Python giúp việc phát triển và triển khai các ứng dụng học máy trở nên dễ dàng và hiệu quả.

Với sự kết hợp giữa tính linh hoạt, hiệu suất và sự hỗ trợ mạnh mẽ từ cộng đồng, Python đã chứng tỏ mình là ngôn ngữ lập trình ưu việt trong lĩnh vực học máy. Việc sử dụng Python không chỉ giúp giảm thời gian và công sức trong quá trình phát triển mô hình, mà còn mở ra nhiều cơ hội để khám phá và áp dụng học máy trong các lĩnh vực khác nhau.



Hình 6: Dart

1.3.7 TFLite

TFLite (TensorFlow Lite) là một framework nhẹ và tối ưu hóa của TensorFlow, được thiết kế đặc biệt để chạy các mô hình học máy trên các thiết bị có tài nguyên hạn chế, như điện thoại di động, thiết bị IoT và các thiết bị nhúng.

TFLite giúp giảm kích thước của mô hình học máy và tăng tốc độ chạy bằng cách sử dụng các kỹ thuật nén và tối ưu hóa. Mô hình được chuyển đổi sang định dạng tflite, một định dạng nhị phân nhẹ nhàng và dễ triển khai trên các thiết bị di động. Điều này cho phép các mô hình được tích hợp trực tiếp vào các ứng dụng di động mà không gây tốn kém cho tài nguyên hệ thống.

TFLite cung cấp một tập hợp các công cụ và thư viện để chuyển đổi, tối ưu hóa và triển khai mô hình học máy trên các thiết bị di động. Nó cũng hỗ trợ các tính năng như phân phối tài nguyên, tăng cường hiệu suất và chạy trực tiếp trên chip thông minh.

Với TFLite, các nhà phát triển có thể xây dựng các ứng dụng học máy tiên tiến trên các thiết bị di động, cho phép xử lý dữ liệu và đưa ra dự đoán nhanh chóng và hiệu quả. Điều này mở ra nhiều cơ hội trong lĩnh vực trí tuệ nhân tạo và học máy trên các nền tảng di động, từ ứng dụng di động thông minh đến các thiết bị nhúng nhỏ gọn.



Hình 7: TFLite

1.3.8 Ưu và nhược điểm của từng loại công nghệ sử dụng

Ưu và nhược điểm của từng loại công nghệ sử dụng

Công nghệ	Ưu điểm	Nhược điểm
Ngôn ngữ lập trình Dart	Dễ học, tài liệu có sẵn, hiệu suất cao, cú pháp rõ ràng, có thể viết mã trên web (DartPad). Có thể hoạt động trên bất kỳ hệ điều hành hay trình duyệt nào.	Khó tìm các giải pháp cho các vấn đề, ít được sử dụng trên thị trường, ít gói hỗ trợ.
Ngôn ngữ lập trình Python	Dễ học, có nhiều thư viện hỗ trợ cho việc xây dựng các mô hình học máy, trí tuệ nhân tạo. Có nhiều tài liệu tham khảo, dễ dàng fix bug.	Tốc độ xử lý, tính toán không cao.
Framework Flutter	Chạy nhanh, có thể xem kết quả ngay lập tức khi thay đổi code, tùy chỉnh toàn bộ và kết xuất nhanh. Áp dụng được cho web, hỗ trợ trên cả Android Studio và Vs Code.	Nhiều tính năng chưa hỗ trợ, chiếm nhiều dung lượng, ứng dụng flutter khá nặng, giao diện không giống 100% bản gốc.

Skype	Kết nối dễ dàng với nhau (đa quốc gia), tính năng chia sẻ màn hình dễ dàng, nhiều chế độ hiển thị. Chia sẻ tài liệu, ghi lại cuộc họp dễ dàng, cho phép thực hiện các cuộc gọi thoại bằng hình ảnh với bất kỳ ai trên thế giới miễn có tài khoản Skype.	Một số tính năng mất phí, mất kết nối, chất lượng âm thanh dựa trên đường truyền, chưa xử lý tiếng ồn xung quanh triệt để khi trò chuyện.
Android Studio	Là công cụ tốt nhất để xây dựng ứng dụng di động, chỉnh sửa mã thông minh, giả lập nhanh, giàu tính năng, có thể kiểm tra các khung của mã, hệ thống giao diện tự do tùy chỉnh.	Yêu cầu hệ thống cao, chiếm nhiều bộ nhớ, độ trễ cao, trình giả lập chậm, sử dụng Ram chậm, cài đặt rất lâu với máy yếu.
VS Code	Nhỏ, rất nhiều tiện ích mở rộng, tùy chỉnh giao diện, soạn thảo rất tốt, dễ dàng, thân thiện với người dùng, có thể sử dụng hầu hết các ngôn ngữ.	Chạy máy ảo khá chậm, một số tiện ích mở rộng hay gấp sự cố, cài càng nhiều tiện ích càng tốn tài nguyên.
TFLite	Hiệu suất tốt trên các thiết bị di động, kích thước nhỏ gọn, tiết kiệm không gian lưu trữ và giảm tải cho tài nguyên hệ thống, dễ tích hợp vào ứng dụng di động.	TFLite bị hạn chế khả năng hỗ trợ cho một số tác vụ học máy phức tạp khác như xử lý ngôn ngữ tự nhiên hay mô hình có kích thước lớn và cấu trúc phức tạp hơn, TFLite còn làm giảm độ chính xác của mô hình hoặc giới hạn khả năng.

2 Thiết kế chương trình

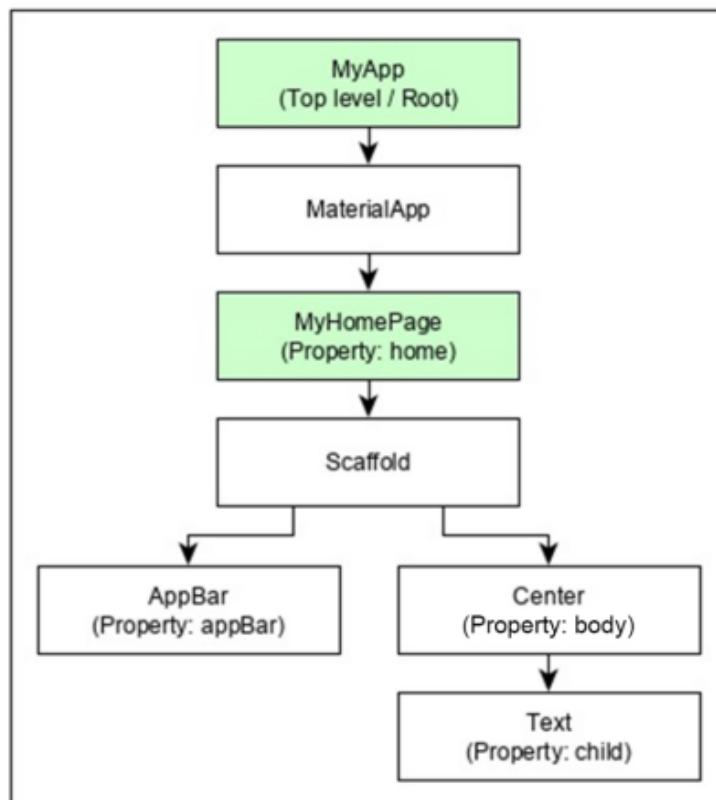
2.1 Xây dựng giao diện mobile

2.1.1 Widget trong Flutter

Mọi khái niệm về giao diện trong Flutter được biểu thị là Widget(tiện ích). Widget là thành phần giao diện cơ bản nhất tạo nên toàn bộ giao diện của ứng dụng.

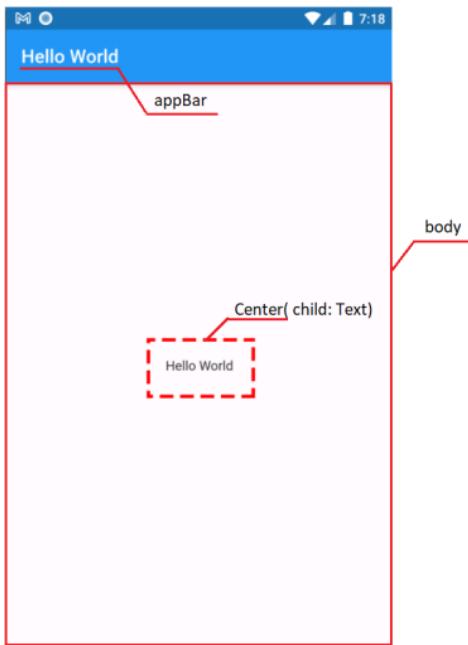
Trong Flutter, bản thân ứng dụng chính là 1 widget. Mỗi ứng dụng chính là một top-level widget và nó bao gồm một hoặc nhiều widget con, mỗi widget này lại bao gồm một hoặc nhiều widget con khác. Nhờ sự kết hợp linh hoạt này lập trình viên có thể tạo ra bất kỳ ứng dụng phức tạp nào.

Khái quát cấu trúc widget của ứng dụng Hello world thông qua sơ đồ bên dưới:



Hình 8: Cấu trúc Widget của ứng dụng HelloWorld

Cấu trúc trên có giao diện như sau:



Hình 9: Giao diện của ứng dụng HelloWorld

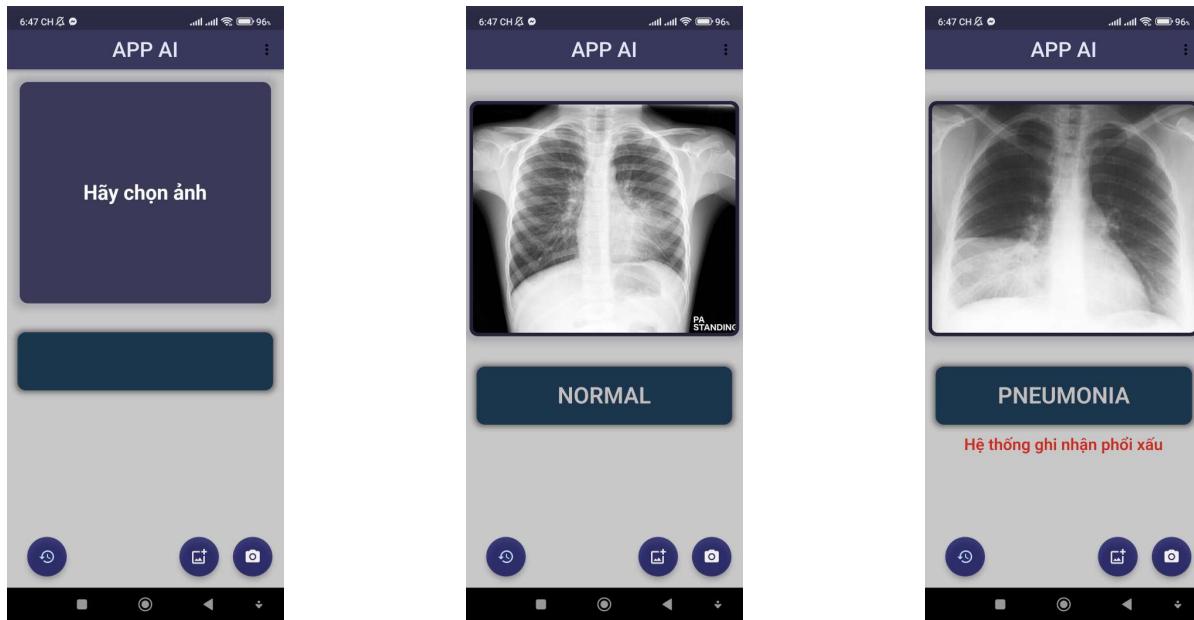
Giải thích cấu trúc:

- MyApp là một widget được tạo ra bằng widget gốc của Flutter, MaterialApp.
- MaterialApp có các thuộc tính của màn hình home và mô tả giao diện người dùng, nó được tạo ra bởi widget MyHomePage.
- MyHomePage được tạo bởi widget gốc của flutter là Scaffold.
- Scaffold có 2 thuộc tính là body và appBar.
- Phần body chứa giao diện chính còn appBar chứa phần header của ứng dụng.
- Widget Center có thuộc tính Child, chứa phần nội dung là 1 widget Text (chứa dòng chữ Hello World).

2.1.2 Màn hình chính

Màn hình chính có giao diện như hình số 10a. Màn hình gồm có 1 container để hiển thị ảnh người dùng chọn hoặc ảnh chụp thông qua camera, 1 container để hiển thị kết quả dự đoán của app. Ở dưới góc phải màn hình là 3 button, có chức năng lắc lượt từ trái qua phải là xem lại lịch sử dự đoán của app, chọn ảnh từ thư viện ảnh trong máy và chụp ảnh trực tiếp từ camera.

Sau khi tiến hành chọn ảnh, app sẽ tiến hành dự đoán và trả về kết quả. Nếu kết quả trả về là phổi của bệnh nhân bình thường thì màn hình sẽ thể hiện như hình 10b, còn nếu chương trình phát hiện ra bệnh nhân bị viêm phổi thì sẽ thể hiện giống như hình 10c.



(a) Màn hình chính khi chưa chọn ảnh

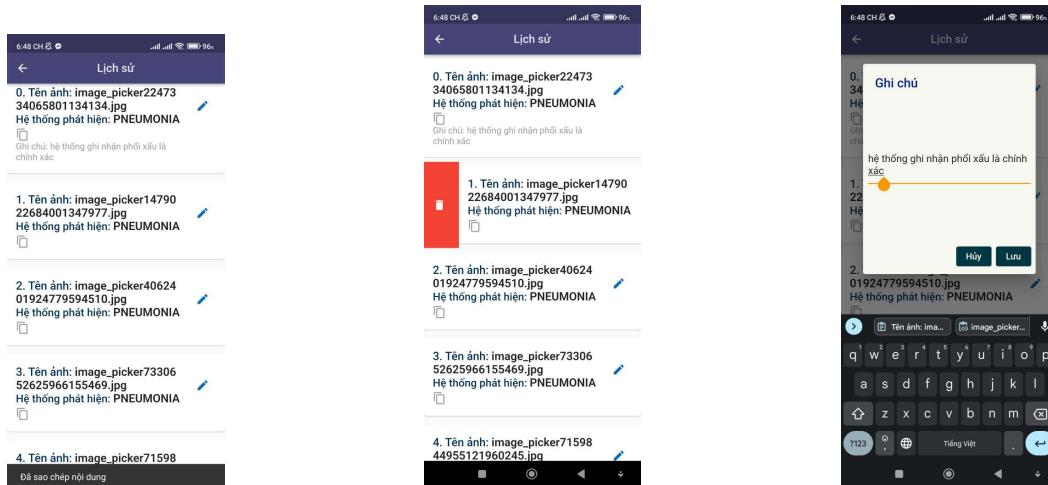
(b) Màn hình khi trả về kết quả phổi tốt

(c) Màn hình khi trả về kết quả phổi xấu

Hình 10: Màn hình chính

2.1.3 Màn hình lịch sử dự đoán

Màn hình lịch sử dự đoán gồm các chức năng hiển thị tên bức ảnh đã dự đoán, nhãn dự đoán. Màn hình lịch sử dự đoán có giao diện được thể hiện ở hình 11a. Nếu như người dùng muốn xóa lịch sử dự đoán, họ chỉ cần vuốt ô lịch sử đó sang bên phải giống như hình 11b. Còn nếu như người dùng muốn thêm ghi chú cho bất kỳ dự đoán nào trong quá khứ thì chỉ cần bấm vào biểu tượng cái bút và chương trình sẽ phản hồi lại giống như hình 11c.



(a) Màn hình lịch sử dự đoán

(b) Màn hình khi muốn xóa lịch sử dự đoán

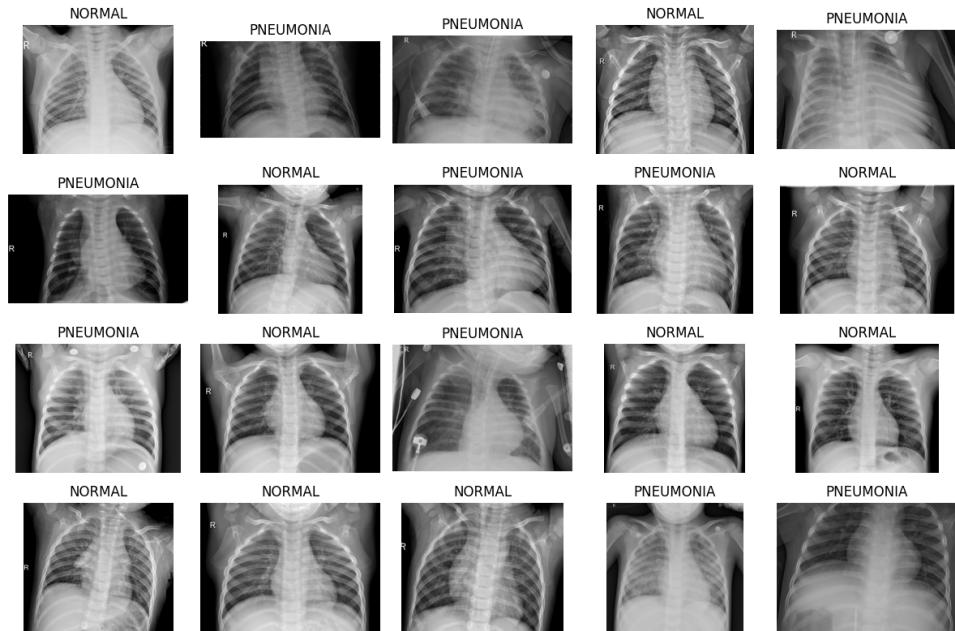
(c) Màn hình ghi chú lịch sử dự đoán

Hình 11: Màn hình lịch sử dự đoán

2.2 Xây dựng mô hình CNN

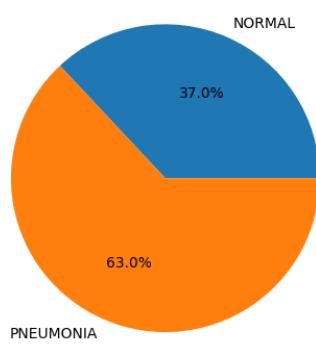
2.2.1 Tập dữ liệu sử dụng

Tập dữ liệu được bạn em sử dụng là bộ ảnh Chest [Beginer] Chest X-ray Classification trên Kaggle. Bộ ảnh gồm có 13791 ảnh và gồm 2 nhãn normal và pneumonia.

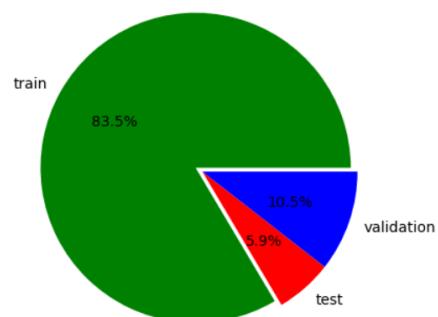


Hình 12: Ảnh X quang phổi trong tập dữ liệu

Trong đó ảnh mang nhãn normal chiếm 37% tổng số ảnh còn ảnh mang nhãn pneumonia chiếm 63%. Bộ ảnh được chia làm 3 tập train, test và validation, mỗi tập ảnh trên chiếm 83.5%, 5.9% và 10.5% tổng số ảnh.



Hình 13: Phân bố nhãn trong tập ảnh



Hình 14: Số lượng ảnh trong các tập train, test, validation

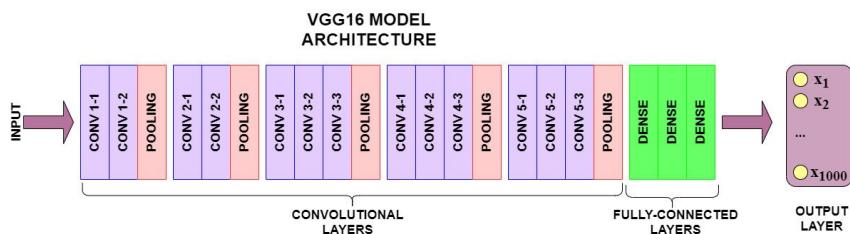
2.2.2 Mô hình học sâu VGG16

Mô hình VGG16 là một trong những mô hình nổi tiếng trong lĩnh vực học sâu và nhận dạng hình ảnh. Nó được phát triển bởi nhóm nghiên cứu tại Đại học Oxford và có tên gọi là VGG (Visual Geometry Group) dựa trên tên của nhóm nghiên cứu.

VGG16 có một kiến trúc mạng nơ-ron sâu với 16 tầng. Kiến trúc này tập trung vào việc sử dụng các lớp tích chập có kích thước nhỏ 3×3 liên tiếp nhau, đẩy mức độ phân tách của các đặc trưng lên cao và giúp nâng cao độ chính xác của mô hình. Nó đã chứng minh khả năng rất tốt trong việc nhận dạng và phân loại hình ảnh trên các tập dữ liệu lớn như ImageNet.

Kiến trúc của VGG16 bao gồm 13 lớp tích chập (Convolutional Layers) và 3 lớp kết nối đầy đủ (Fully Connected Layers). Các lớp tích chập được nhóm lại thành các khối, trong đó có 5 khối có cùng cấu trúc, gồm 2 lớp tích chập và sau đó là một lớp gộp (Pooling Layer) để giảm kích thước đầu ra. Các lớp tích chập đều sử dụng hàm kích hoạt ReLU và sau đó là một lớp chuẩn hóa theo độ sâu (Normalization Layer).

Cuối cùng, sau khi các lớp tích chập đã trích xuất được các đặc trưng từ hình ảnh, các lớp kết nối đầy đủ được sử dụng để phân loại hình ảnh vào các nhãn tương ứng. VGG16 thường được sử dụng trong các bài toán phân loại hình ảnh, nhận dạng đối tượng và trích xuất đặc trưng.



Hình 15: Mô hình VGG16

Áp dụng vào bài toán xử lý ảnh viêm phổi, bạn em sử dụng mô hình Vgg16 được xây dựng sẵn của thư viện Keras với trọng số đã được lấy từ bộ dữ liệu Imagenet. Mô hình nhận ảnh đầu vào có kích thước $(224, 224, 3)$ và sử dụng phép max pooling.

Sau đó, dòng lệnh **model.trainable = False** được sử dụng để đóng băng (freeze) các trọng số của mô hình, điều này đảm bảo rằng chỉ có các lớp mới được thêm vào sau này sẽ được huấn luyện. Ảnh đầu vào sẽ qua 16 tầng lớp tích chập, ở mỗi tầng tích chập, ảnh sẽ đi qua lớp max pooling để giảm độ phức tạp tính toán và số lượng tham số của mô hình cũng như là một biểu diễn tổng quát của đặc trưng bằng cách chọn giá trị lớn nhất.

Ảnh sau khi đi qua 16 tầng tích chập sẽ được dát mỏng thành ma trận 1 chiều bằng hàm Flatten() trước khi được đưa vào lớp Fully Connected. Lớp

này sẽ bao gồm các hàm GlobalAveragePooling2D và Dense.

Lớp GlobalAveragePooling2D được sử dụng để thực hiện phép pooling trung bình trên toàn bộ ảnh đầu vào, có tác dụng giảm chiều dữ liệu. Nó tính giá trị trung bình của các đặc trưng trên mỗi kênh và tạo ra một vector đặc trưng có kích thước là (1, 1, 512).

Sau khi đi qua lớp GlobalAveragePooling2D, ảnh sẽ đi qua lớp dense với hàm kích hoạt ReLU. Nó sẽ thực hiện phép ánh xạ tuyến tính bằng cách tính tổng trọng số của các đầu vào và hệ số điều chỉnh (bias). Sau đó, hàm kích hoạt ReLU được áp dụng để tạo ra đầu ra cuối cùng của lớp này.

Ảnh sau khi đi qua lớp dense thì sẽ đi qua lớp dense cuối cùng có 2 nơ-ron và sử dụng hàm kích hoạt sigmoid. Nó nhận đầu vào từ lớp trước đó và thực hiện phép ánh xạ tuyến tính. Hàm kích hoạt sigmoid được áp dụng để chuyển đổi giá trị đầu ra thành một xác suất trong khoảng từ 0 đến 1, cho biết xác suất dự đoán của mô hình cho các lớp đích. Kết quả của lớp này là một vector có kích thước 2, biểu thị xác suất dự đoán cho hai lớp khác nhau.

2.2.3 Mô hình học sâu Resnet50

Mô hình ResNet50 là một mô hình nổi tiếng trong lĩnh vực học sâu và nhận dạng hình ảnh. Nó được phát triển bởi nhóm nghiên cứu tại Microsoft Research và có tên gọi là ResNet (Residual Network).

ResNet50 có một kiến trúc mạng nơ-ron sâu với tổng cộng 50 tầng. Mô hình này đặc biệt vì sử dụng khối cơ bản gọi là "bottleneck block" để giải quyết vấn đề mất mát thông tin và xấp xỉ hàm nhận dạng. Các khối "bottleneck block" có cấu trúc phức tạp hơn so với các mô hình truyền thống, nhưng lại cho phép mô hình học được các đặc trưng phức tạp hơn và sâu hơn.

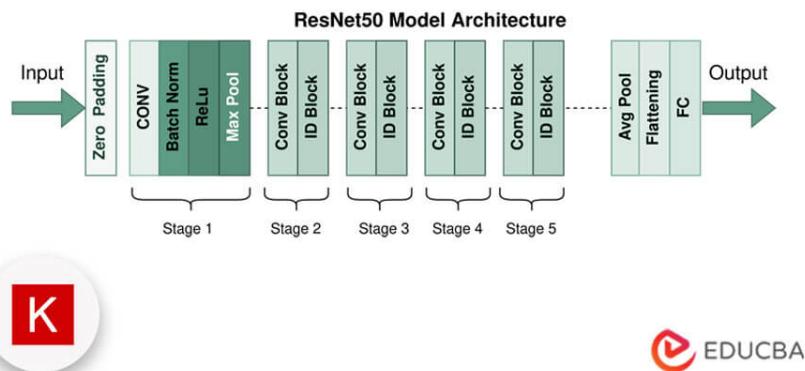
Kiến trúc của ResNet50 bao gồm nhiều khối "bottleneck block", trong đó mỗi khối gồm nhiều lớp tích chập và kết nối đầy đủ. Mỗi khối cung cấp một đường đi trực tiếp (identity shortcut) để truyền thông tin từ lớp đầu vào đến lớp đầu ra, giúp giảm mất mát thông tin và đảm bảo rằng thông tin quan trọng không bị mất đi qua quá trình lan truyền ngược.

Mô hình ResNet50 thường được sử dụng trong các bài toán phân loại hình ảnh, nhận dạng đối tượng và trích xuất đặc trưng. Nó đã chứng minh khả năng rất tốt trong việc xử lý các tập dữ liệu lớn và phức tạp như ImageNet.

Áp dụng vào bài toán nhận diện ảnh viêm phổi thông qua ảnh X quang, mô hình Resnet50 được xây dựng bằng mô hình được dựng sẵn của thư viện Keras với các trọng số được tải từ bộ dữ liệu Imagenet. Mô hình này được định nghĩa với đầu vào có kích thước (224, 224, 3) và sử dụng phép pooling max.

Giống với mô hình Vgg16, các trọng số của mô hình sẽ được đóng băng

Keras ResNet⁵⁰



Hình 16: Mô hình Resnet

bằng lệnh '**model.trainable = False**', điều này đảm bảo rằng chỉ có các lớp mới được thêm vào sau này sẽ được huấn luyện.

Ảnh sau khi đi qua 50 tầng của mạng Resnet50 sẽ được dát mỏng thành ma trận 1 chiều bằng hàm Flatten() trước khi được đưa vào lớp Fully Connected. Lớp này sẽ bao gồm các hàm GlobalAveragePooling2D và Dense. Cấu trúc của lớp fully connected ở mô hình Resnet50 được xây dựng giống với lớp Fully Connected được xây dựng cho mô hình Vgg16 ở phía trên. Điểm khác biệt duy nhất giữa lớp Fully Connected của mô hình Resnet50 so với Vgg16 đó là tham số của các lớp GlobalAveragePooling và Dense. Do mô hình Resnet50 sau khi đi qua lớp Flatten sẽ trả về một vector có kích thước là (1,1,2048) so với (1,1,512) ở mô hình Vgg16. Vậy nên tham số đầu vào ở lớp GlobalAveragePooling của mô hình Resnet50 sẽ là 2048 thay vì 512 như ở mô hình Vgg16 và tương tự đối với lớp Dense có hàm kích hoạt là Relu.

2.2.4 Mô hình học sâu InceptionV3

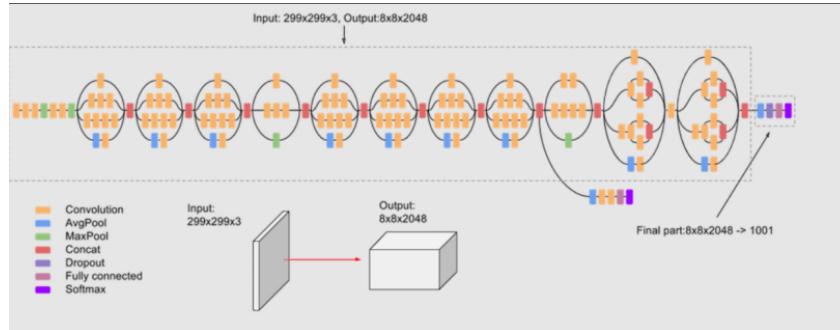
Mô hình InceptionV3 là một trong những mô hình nổi tiếng và mạnh mẽ trong lĩnh vực nhận dạng hình ảnh, được phát triển bởi nhóm nghiên cứu Google trong dự án Inception. Với kiến trúc đa tầng và một số tính năng độc đáo, InceptionV3 đã chứng minh khả năng đáng kinh ngạc trong việc phân loại hình ảnh và trích xuất đặc trưng.

Kiến trúc cơ bản của InceptionV3 được xây dựng trên ý tưởng sử dụng các mô-đun Inception, nơi các bộ lọc và kích thước khác nhau được áp dụng song song trên đầu vào. Các mô-đun Inception được thiết kế để tận dụng thông tin từ các tầng trước đó và tạo ra một lượng lớn các đặc trưng trung gian. Điều này giúp mô hình tạo ra một biểu diễn phức tạp và đa dạng của dữ liệu hình ảnh.

Kiến trúc InceptionV3 sử dụng các mô-đun Inception với các kích thước bộ lọc khác nhau như 1x1, 3x3 và 5x5 để trích xuất thông tin từ các tầng

trước đó. Đồng thời, nó cũng áp dụng kỹ thuật giảm chiều dữ liệu bằng cách sử dụng các tầng giảm kích thước như max pooling và average pooling. Điều này giúp giảm kích thước dữ liệu và tăng tính hiệu quả tính toán của mô hình.

Một điểm đáng chú ý khác của InceptionV3 là việc sử dụng kỹ thuật giảm overfitting như Regularization và Dropout. Điều này giúp mô hình trở nên ổn định hơn và giảm khả năng bị overfitting khi huấn luyện trên các tập dữ liệu lớn.



Hình 17: Mô hình InceptionV3

Áp dụng vào bài toán nhận diện viêm phổi thông qua ảnh X quang, nhóm đã tiến hành xây dựng mô hình InceptionV3 bằng mô hình được dựng sẵn trong thư viện Keras với trọng số từ bộ dữ liệu Imagenet. Mô hình nhận ảnh có kích thước đầu vào là (224,224,3) và có sử dụng phép max pooling. Trong trường hợp này, các trọng số lấy từ tập dữ liệu Imagenet sẽ được vô hiệu hóa nhằm đảm bảo rằng chỉ có các lớp mới được thêm vào sau này sẽ được huấn luyện.

Sau khi đi qua các mô-đun Inception, ảnh sẽ được là phẳng và chuyển tới lớp Fully Connected với các thành phần chính là các hàm GlobalAveragePooling2D và Dense. Cấu trúc của lớp Fully Connected của mô hình InceptionV3 được xây dựng giống với lớp Fully Connected của mô hình Resnet. Các tham số truyền vào các hàm GlobalAveragePooling với hàm Dense đều không thay đổi do cả hai mô hình này đề cho ra kết quả là một vector có kích thước là (1, 1, 2048) sau khi qua làm Flatten. Do đó, nhóm đã tiến hành xây dựng lớp Fully Connected của mô hình InceptionV3 giống với mô hình Resnet50.

2.2.5 Huấn luyện mô hình và kết quả thu được

Sau khi chia tập ảnh thành 3 tập con là train, test, và validation, bọn em tiến hành huấn luyện cả 3 mô hình học máy Vgg16, Resnet50 và InceptionV3 trên máy laptop cá nhân. Cấu hình máy tính mà nhóm bọn em sử dụng có bộ xử lý Intel i59300H và card đồ họa rời GTX1050 4GB (Mobile). Bọn em tiến hành huấn luyện lần lượt 3 mô hình trên 50 epochs. Thời gian tiến hành huấn luyện mỗi mô hình trên máy tính cá nhân mất khoảng 4 tiếng.

2.3 Tích hợp mô hình CNN vào ứng dụng mobile

Sau khi tiến hành huấn luyện mô hình và thu được kết quả như mong muốn, nhóm tiến hành lưu lại kết quả mô hình vào file h5 và sau đó sử dụng thư viện TFLiteConverter để tiến hành chuyển đổi mô hình sang dạng TFLite.

2.3.1 Chuyển đổi file h5 sang TFLite

Để tiến hành chuyển đổi file h5 sang dạng TFLite, nhóm chúng em sử dụng thư viện có sẵn của Tensorflow để tiến hành chuyển đổi. Quá trình chuyển đổi được minh họa bằng ví dụ dưới đây:

```
1 model_h5 = tf.keras.models.load_model('assets/model_lung_2.h5')
2
3 converter = tf.lite.TFLiteConverter.from_keras_model(model_h5)
4 tflite_model = converter.convert()
5
6 with open('assets/model_lung_2.tflite', 'wb') as f:
7     f.write(tflite_model)
```

2.3.2 Tích hợp thư viện TFLite

Để tiến hành sử dụng được mô hình học máy trong ứng dụng mobile, cần phải tải hai thư viện sau:

- tflite_flutter: cho phép người dùng truy cập vào thư viện TensorFlow Lite. Khi gọi các phương thức của tflite_flutter, nó sẽ gọi phương thức tương ứng của SDK TensorFlow Lite gốc.
- tflite_flutter_helper: Cho phép thao tác với đầu vào và đầu ra của TensorFlow. Ví dụ, nó chuyển đổi dữ liệu hình ảnh thành cấu trúc tensor. Điều này giúp giảm công sức cần thiết để tạo ra logic tiền xử lý và hậu xử lý cho mô hình của bạn.

Để dùng được các thư viện trên, cần phải tiến hành khai báo chúng ở trong file **pubspec.yaml** như sau:

```
1 tflite_flutter: ^0.9.0
2 tflite_flutter_helper: ^0.3.1
```

vspace0.2cm Sau đó chạy lệnh **flutter pub get** để lấy các package từ trong thư viện.

2.3.3 Xây dựng lớp Classifier

Lớp Classifier có nhiệm vụ sau:

- Đọc các nhãn từ file và tích hợp mô hình học máy
- Tiền xử lý ảnh
- Tiến hành dự đoán

- Xử lí output từ Tensorflow

Đầu tiên, quá trình đọc file nhãn với mô hình được thực hiện trong Flutter như sau:

```
1  Future<void> loadLabels() async {
2      labels = await FileUtil.loadLabels(_labelsFileName);
3      if (labels.length == _labelsLength) {
4          print('Labels loaded successfully');
5      } else {
6          print('Unable to load labels');
7      }
8  }
9
10 Future<void> loadModel() async {
11     try {
12         interpreter =
13             await Interpreter.fromAsset(modelName, options: _interpreterOptions);
14         print('Interpreter Created Successfully');
15
16         _inputShape = interpreter.getInputTensor(0).shape;
17         _outputShape = interpreter.getOutputTensor(0).shape;
18         _inputType = interpreter.getInputTensor(0).type;
19         _outputType = interpreter.getOutputTensor(0).type;
20
21         _outputBuffer = TensorBuffer.createFixedSize(_outputShape, _outputType);
22         _probabilityProcessor =
23             TensorProcessorBuilder().add(postProcessNormalizeOp).build();
24     } catch (e) {
25         print('Unable to create interpreter, Caught Exception: ${e.toString()}');
26     }
27 }
```

Đối với bước tiền xử lí ảnh, nhóm tiến hành thu nhỏ ảnh theo kích thước bé nhất theo chiều dài hoặc chiều rộng của ảnh.

```
1  TensorImage _preProcess() {
2      int cropSize = min(_inputImage.height, _inputImage.width);
3      return ImageProcessorBuilder()
4          .add(ResizeWithCropOrPadOp(cropSize, cropSize))
5          .add(ResizeOp(
6              _inputShape[1], _inputShape[2], ResizeMethod.NEAREST_NEIGHBOUR))
7          .add(preProcessNormalizeOp)
8          .build()
9          .process(_inputImage);
10 }
```

Và cuối cùng là xây dựng hàm dự đoán cho lớp Classifier

```
1  Category predict(Image image) {
2      final pres = DateTime.now().millisecondsSinceEpoch;
3      _inputImage = TensorImage(_inputType);
4      _inputImage.loadImage(image);
5      _inputImage = _preProcess();
6      final pre = DateTime.now().millisecondsSinceEpoch - pres;
7
8      print('Time to load image: $pre ms');
9
10     final runs = DateTime.now().millisecondsSinceEpoch;
11     interpreter.run(_inputImage.buffer, _outputBuffer.getBuffer());
12     final run = DateTime.now().millisecondsSinceEpoch - runs;
13
14     print('Time to run inference: $run ms');
15
16     Map<String, double> labeledProb = TensorLabel.fromList(
17         labels, _probabilityProcessor.process(_outputBuffer))
18         .getMapWithFloatValue();
19     final pred = getTopProbability(labeledProb);
20
21     return Category(pred.key, pred.value);
22 }
```

Sau các bước trên, Flutter đã kết nối thành công với TFLite và có thể sử dụng mô hình TensorFlow Lite trong ứng dụng.

2.4 Cách hoạt động của chương trình

App sẽ tiến hành dự đoán bệnh, người dùng sẽ tiến hành chọn ảnh từ 1 trong 2 nguồn khác nhau đó là chụp ảnh từ camera của điện thoại hoặc là chọn ảnh từ trong thư viện ảnh của máy.

Để lấy được những dữ liệu từ các nguồn trên nhóm đã sử dụng các API và thư viện trong Flutter như dart:io, image_picker, gallery_saver và shared_preferences. Đầu tiên, nhóm đã import các thư viện này vào mã nguồn của ứng dụng. Sau đó, khai báo và khởi tạo các biến cần thiết để thực hiện quá trình chọn và xử lý ảnh. Sử dụng lớp con ClassifierQuant để khởi tạo đối tượng _classifier l. Điều này cho phép nhóm sử dụng mô hình đã được lưu trước đó để dự đoán nhãn trên ảnh.

Tiếp theo, nhóm có tạo 1 biến image để lưu trữ đường dẫn tới ảnh được chọn từ Camera hoặc thư viện. Tạo biến imageWidget là một đối tượng Image được sử dụng để hiển thị ảnh đã chọn trên giao diện người dùng. Nhóm đã định nghĩa hai phương thức getImage() và captureImage() để lấy ảnh từ thư viện và Camera tương ứng. Nhóm sử dụng ImagePicker để lấy ảnh từ các nguồn này và sau đó gán giá trị cho biến image và imageWidget. Đồng thời, nhóm đánh dấu imageSelected là true để biểu thị rằng đã có ảnh được chọn. Sau đó, nhóm gọi phương thức predict() để thực hiện quá trình dự đoán trên ảnh đã chọn.

Tạo 1 phương thức saveToHistory() được sử dụng để lưu trữ thông tin về ảnh đã dự đoán vào lịch sử ==> sử dụng SharedPreferences để lưu trữ danh sách các ảnh đã dự đoán. Phương thức này lấy thông tin về tên ảnh, nhãn và thời gian dự đoán, sau đó thêm vào danh sách lịch sử. Cuối cùng, nhóm có phương thức predict(), nơi nhóm thực hiện quá trình dự đoán trên ảnh đã chọn. Đầu tiên, nhóm kiểm tra xem có ảnh được chọn và biến image có giá trị không null hay không. Nếu có, nhóm sử dụng thư viện image để đọc và giải mã ảnh từ đường dẫn image, sau đó gọi phương thức predict() trên đối tượng classifier để dự đoán nhãn. Kết quả dự đoán được lưu vào biến category. Nếu category không null, nhóm lưu thông tin về ảnh, nhãn và thời gian vào lịch sử bằng cách gọi phương thức saveToHistory() và lưu ảnh vào bộ nhớ thiết bị.

Để hiển thị lại dữ liệu đã dự đoán. Bước đầu tiên là truy xuất Shared-Preferences để lấy danh sách imageHistory, đại diện cho lịch sử ảnh đã được dự đoán trong ứng dụng.

Sau khi có được danh sách imageHistory, nhóm tiếp tục lặp qua từng mục trong danh sách này. Trong mỗi lần lặp, nhóm lấy danh sách ghi chú tương ứng từ bộ nhớ, được lưu trữ dưới tên imageNotes_index, với index là chỉ

mục hiện tại trong danh sách imageHistory. Danh sách ghi chú này đại diện cho các ghi chú liên quan đến mỗi ảnh trong lịch sử.

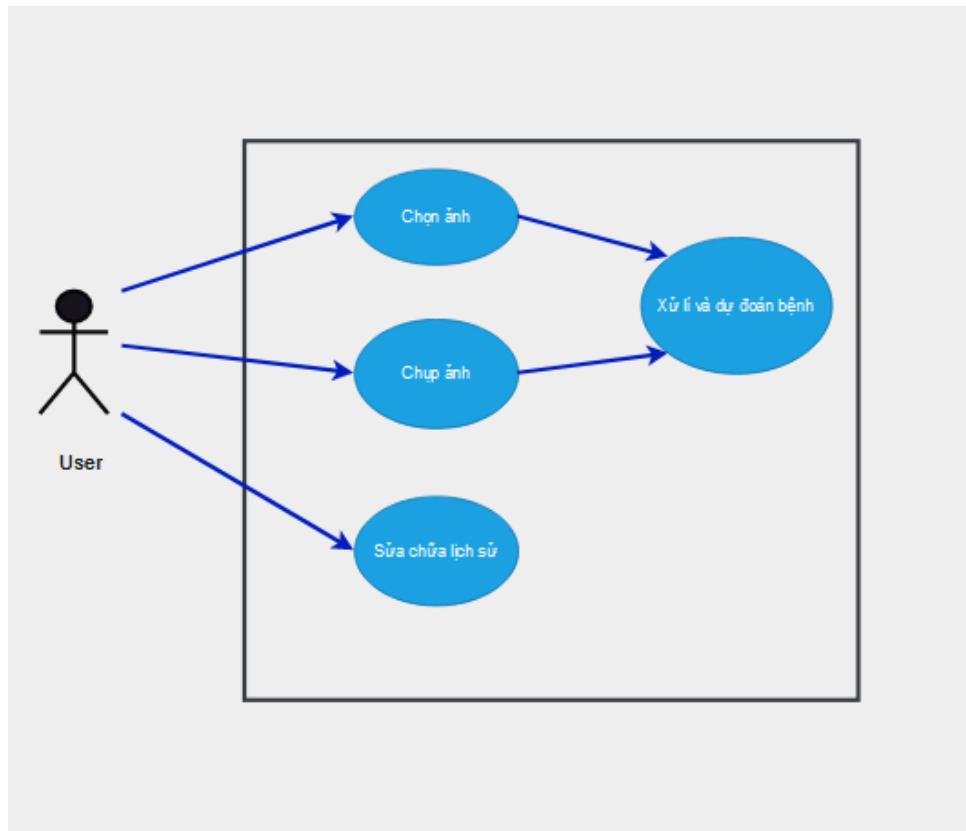
Sau khi hoàn tất quá trình lặp, nhóm cập nhật trạng thái của lớp _HistoryScreenState bằng cách gán giá trị mới cho các biến imageHistory và imageNotes. Điều này đảm bảo rằng dữ liệu lịch sử ảnh đã được tải vào ứng dụng và sẵn sàng để hiển thị cho người dùng.

Với dữ liệu lịch sử ảnh đã tải, người dùng có thể thực hiện các hoạt động như xem danh sách lịch sử ảnh đã dự đoán và ghi chú liên quan. Họ cũng có thể thực hiện các hành động như xóa một mục trong lịch sử. Khi một mục được xóa, nhóm cập nhật lại danh sách lịch sử ảnh và danh sách ghi chú, và lưu các thay đổi vào bộ nhớ để giữ cho dữ liệu được cập nhật.

Với việc cung cấp thư viện lưu và load dữ liệu local, ứng dụng có thể hiển thị danh sách lịch sử ảnh đã dự đoán và các ghi chú liên quan. Người dùng có thể thực hiện các hoạt động quản lý và tương tác với các mục trong lịch sử, đảm bảo rằng dữ liệu lịch sử ảnh luôn được cập nhật và sẵn sàng sử dụng.

3 Thiết kế sơ đồ UML

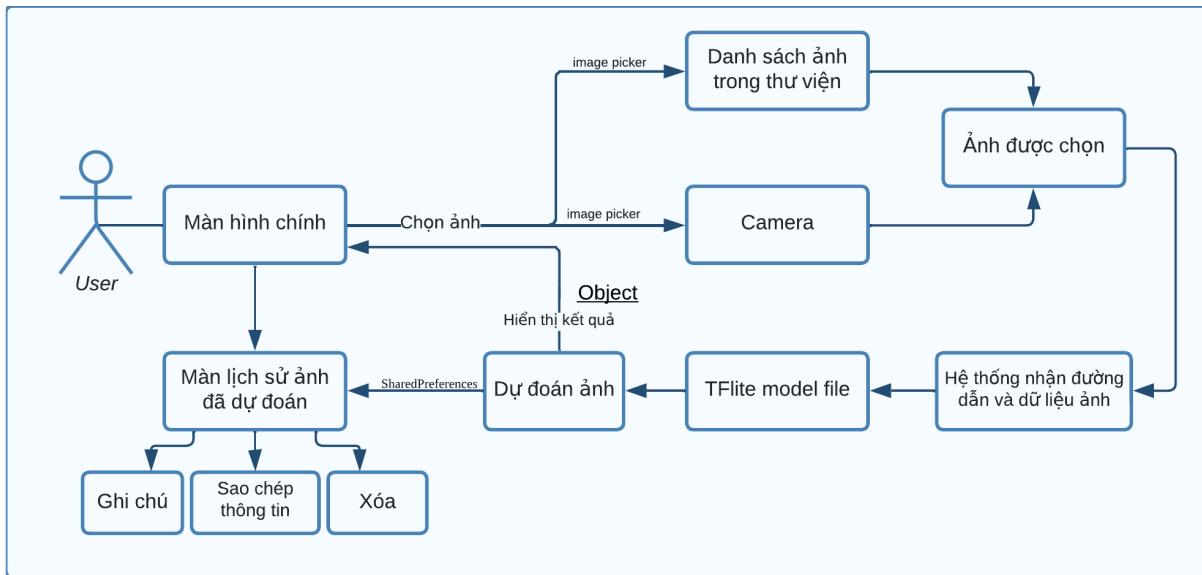
3.1 Biểu đồ use case



Hình 18: Sơ đồ use case

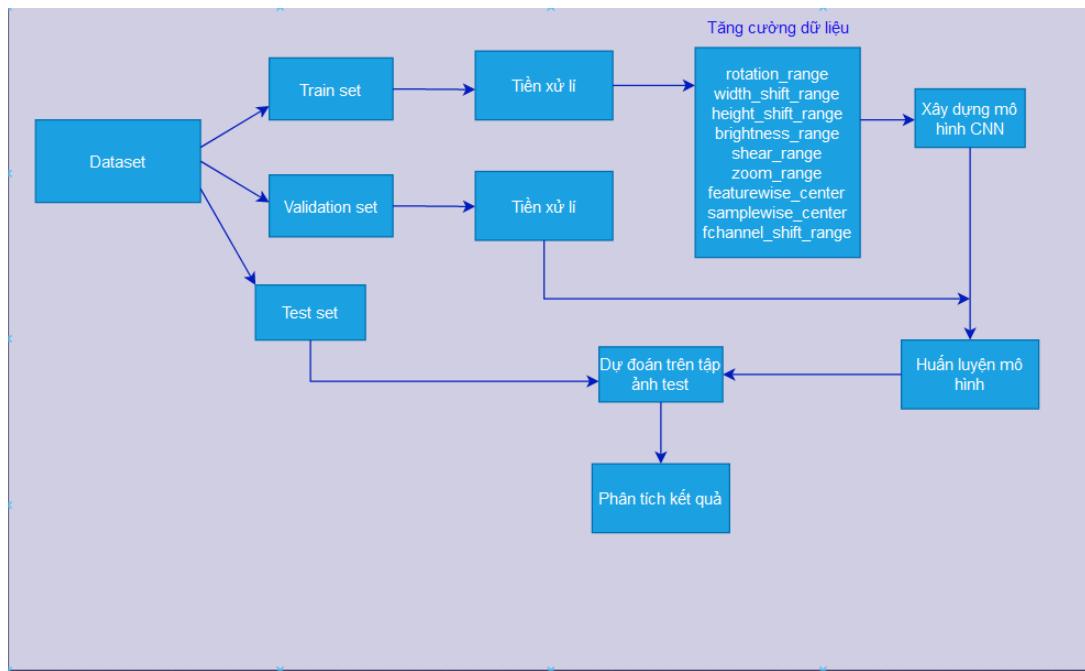
3.2 Biểu đồ hoạt động

3.2.1 Hoạt động của app



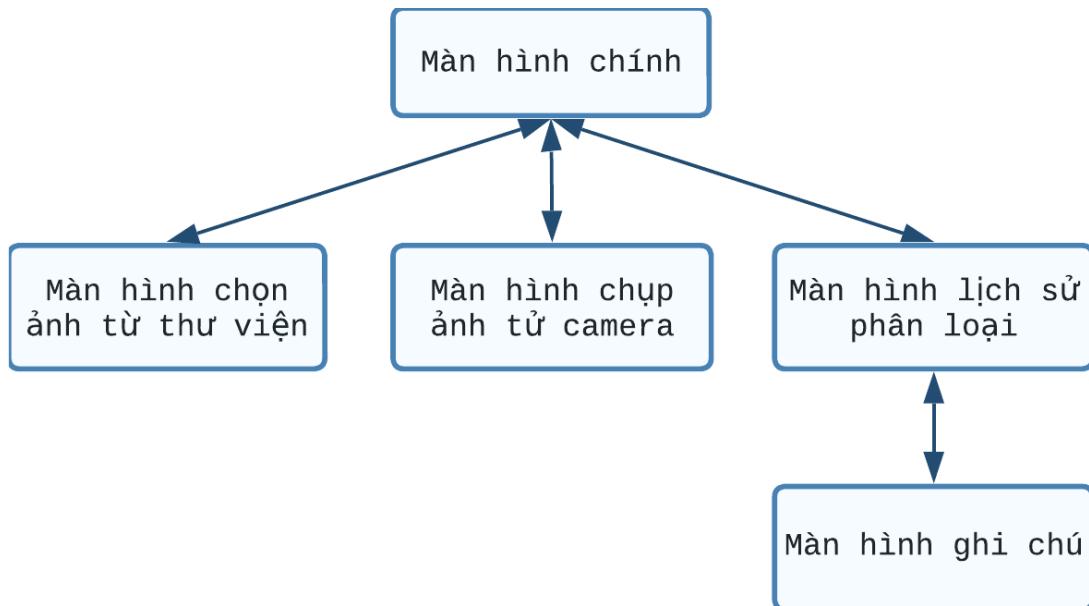
Hình 19: Sơ đồ hoạt động của app

3.2.2 Xây dựng và huấn luyện mô hình



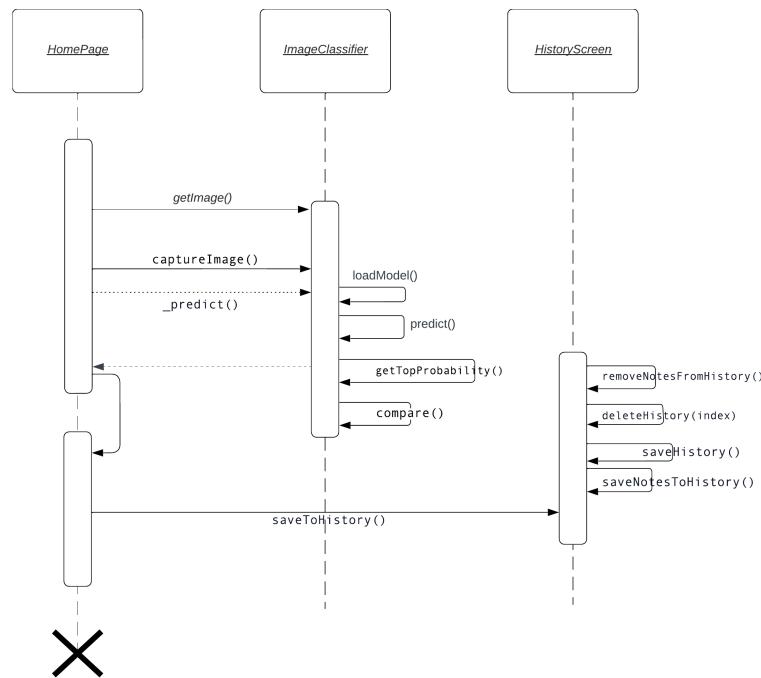
Hình 20: Sơ đồ xây dựng và huấn luyện mô hình CNN

3.3 Biểu đồ trạng thái



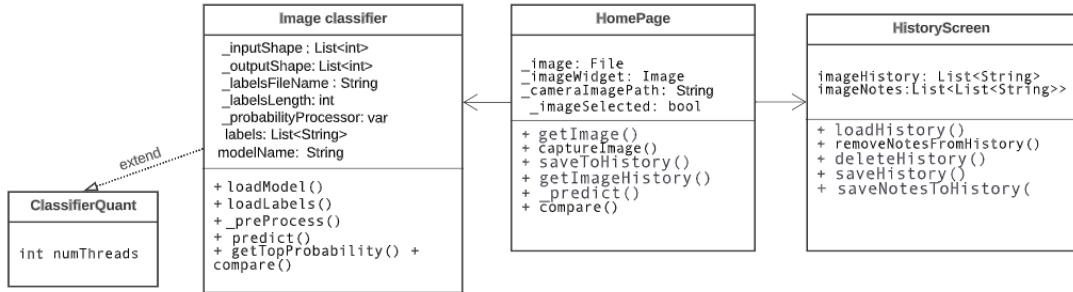
Hình 21: Sơ đồ trạng thái của ứng dụng

3.4 Biểu đồ tuần tự



Hình 22: Biểu đồ tuần tự

3.5 Class diagram

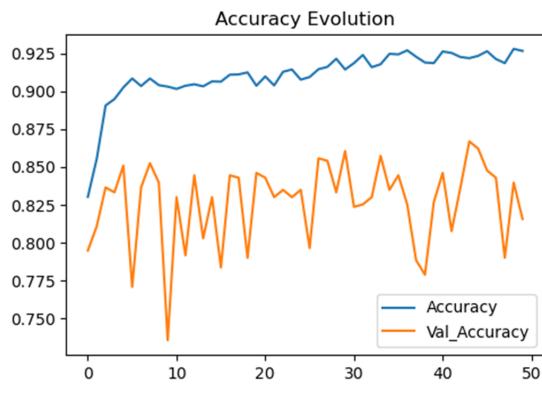


Hình 23: Class diagram của ứng dụng

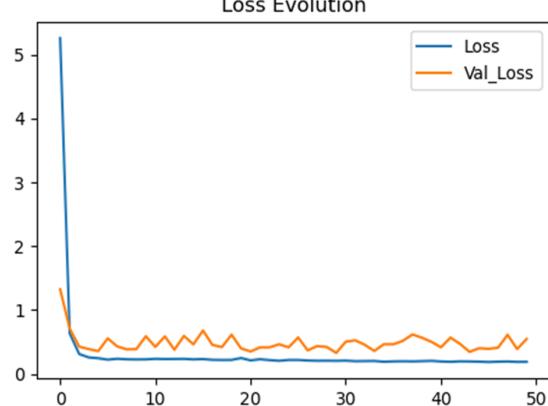
4 Kết quả và hướng phát triển

4.1 Kết quả

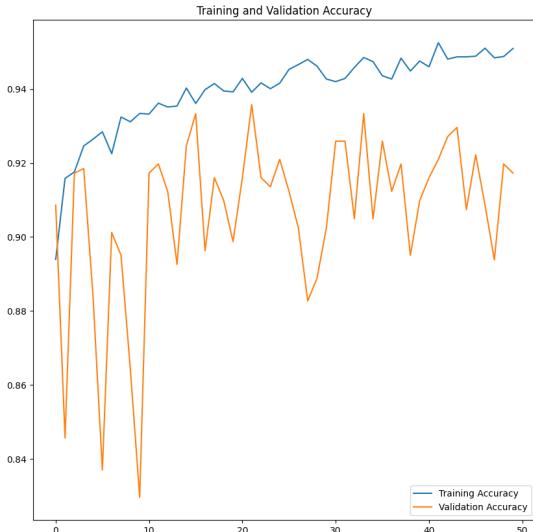
Dưới đây là sơ đồ mô tả độ chính xác và độ mất mát trong quá trình huấn luyện và kiểm tra của 3 mô hình học sâu là Vgg16, Resnet50 và InceptionV3 cho tập ảnh Chest [Beginer] Chest X-ray Classification.



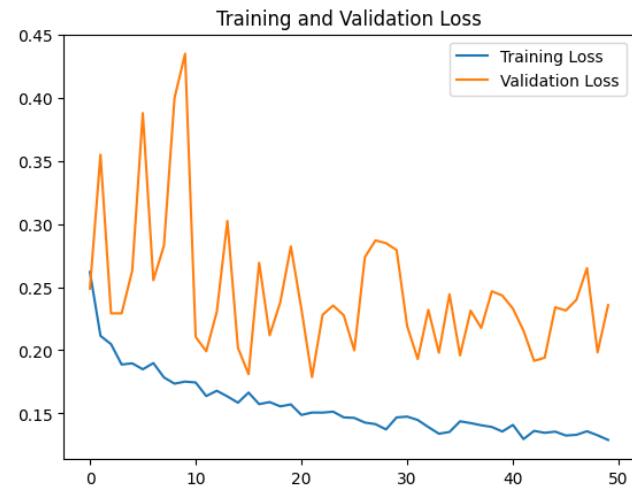
Hình 24: Độ chính xác của mô hình Inception



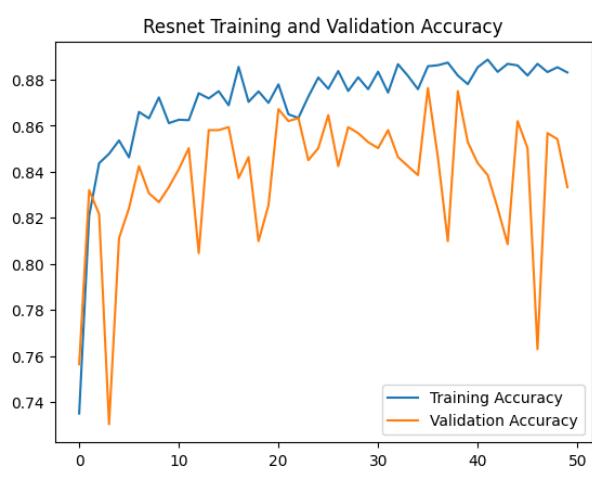
Hình 25: Hàm loss của mô hình Inception



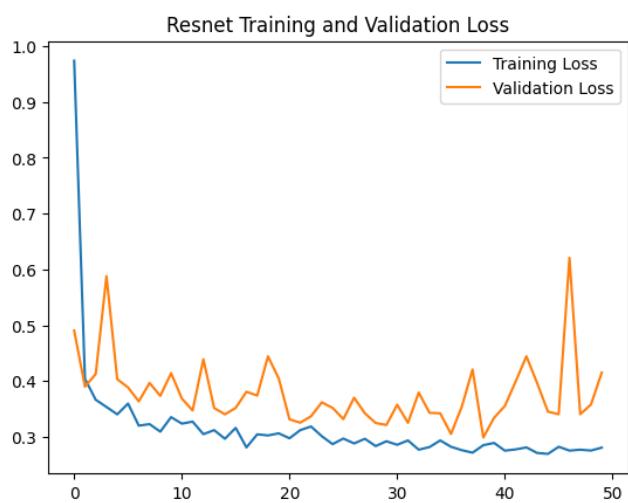
Hình 26: Độ chính xác của mô hình Vgg16



Hình 27: Hàm loss của mô hình Vgg16



Hình 28: Độ chính xác của mô hình Resnet50



Hình 29: Hàm loss của mô hình Resnet50

Nhìn vào kết quả trên, ta có thể nhận thấy mô hình Vgg16 trả về kết quả có độ chính xác cao nhất (xấp xỉ 92%) so với các mô hình còn lại là InceptionV3 (xấp xỉ 85%) và Resnet (xấp xỉ 86%). Tuy nhiên, thang đo bằng độ chính xác của mô hình đôi khi không phản ánh đúng mức độ chính xác của mô hình. Do đó, nhóm chúng em đã sử dụng thêm thang đo là Precision, Recall và F1 score để tiến hành so sánh độ chính xác của mô hình. Kết quả Precision, Recall và F1 score mà nhóm bạn em tính toán được sẽ được mô tả ở bảng dưới đây:

Điểm số	VGG16	InceptionV3	Resnet50
Precision score	0.5644	0.5602	0.5334
Recall score	0.5152	0.5349	0.5271
F1 score	0.5387	0.5473	0.5303

Như bảng số liệu trên, ta có thể thấy được mô hình VGG16 cho thấy sự cân bằng tương đối giữa Precision và Recall. Precision là tỷ lệ giữa số lượng dự đoán đúng và tổng số lượng dự đoán positive. Recall là tỷ lệ giữa số lượng dự đoán đúng và tổng số lượng ground truth positive. Điểm F1-score là một trung bình điều hòa giữa Precision và Recall, giúp đánh giá sự kết hợp hiệu quả của cả hai chỉ số trên. Trong trường hợp này, mô hình VGG16 đạt F1-score là 0.5387, cho thấy mức độ khá trong việc kết hợp Precision và Recall.

Mô hình InceptionV3 cũng thể hiện sự cân bằng tương đối giữa Precision và Recall. Tuy nhiên, F1-score của mô hình InceptionV3 là 0.5473, cao hơn so với mô hình VGG16. Điều này cho thấy mô hình InceptionV3 có khả năng kết hợp Precision và Recall một cách hiệu quả hơn, đạt độ chính xác tổng thể tốt hơn trong việc phân loại.

Mô hình ResNet cũng đạt được một mức độ cân bằng tương đối giữa Precision và Recall. Tuy nhiên, F1-score của mô hình ResNet là 0.5303, thấp hơn so với cả hai mô hình trước đó. Điều này cho thấy mô hình ResNet có khả năng kết hợp Precision và Recall chưa tốt hơn so với hai mô hình kia, và có thể cần được tinh chỉnh để đạt được hiệu suất tốt hơn trong việc phân loại.

Dựa trên kết quả trên, nhóm quyết định chọn mô hình InceptionV3 do mô hình InceptionV3 vì nó đạt được sự cân bằng tương đối giữa độ chính xác và các chỉ số Precision, Recall và F1-score. Mô hình này có độ chính xác 85% và F1-score là 0.5473, cao hơn so với mô hình VGG16 và ResNet50. Điều này cho thấy mô hình InceptionV3 có khả năng kết hợp Precision và Recall một cách hiệu quả, đồng thời đạt được độ chính xác tổng thể tốt hơn.

4.2 Hướng phát triển

Mặc dù nhóm đã cố gắng tìm hiểu cũng như là tra cứu các tài liệu chuyên ngành cũng như là nghiệp vụ nhưng ứng dụng phát hiện và phân loại viêm phổi thông qua ảnh X quang còn nhiều hạn chế. Kết quả mà nhóm thu được chưa cao, mô hình đang bị overfit.

Trong tương lai, nhóm dự kiến sẽ tiến hành huấn luyện lại mô hình và sẽ áp dụng các kỹ thuật tiền xử lý ảnh như Gaussian Blur. Đồng thời nhóm sẽ tiến hành phân chia lại tập dữ liệu, giảm số lượng ảnh ở tập train và tăng số lượng ảnh ở tập test và validation. Ngoài ra nhóm cũng sẽ bổ sung thêm các tính năng cho ứng dụng như có khả năng zoom ảnh, đổi với mô hình học sâu, ngoài cải thiện độ chính xác của mô hình, nhóm cũng sẽ tiến hành cải thiện mô hình giúp mô hình có thể phân loại được các loại bệnh liên quan tới phổi như là viêm phổi virus, viêm phổi do nấm, xác định các tổn thương phổi do Covid,...

Nhóm cũng rất mong nhận được sự góp ý của thầy cô để nhóm có thể hoàn thiện sản phẩm.

4.3 Bài học kinh nghiệm sau kỳ thực tập

Qua kỳ thực tập trên, nhóm chúng em rút ra được một số bài học sau:

- Nắm bắt một số khái niệm cơ bản và phương pháp xây dựng ứng dụng Flutter.
- Phát triển khả năng thiết kế giao diện, phân tích trải nghiệm của người sử dụng.
- Nắm được cách thức làm việc trong môi trường công ty phần mềm.
- Bài học về kỹ năng: luôn có sự chuẩn bị kỹ càng, tập trung vào vấn đề, quản lý thời gian. Đặc biệt phải có sự chỉn chu và thích nghi tốt.
- Kinh nghiệm về thái độ làm việc: phải có tính tự giác, tính kỷ luật cao, tinh thần làm việc tích cực, ham học hỏi và phải biết sửa sai.

5 Phân công công việc

Họ tên	Công việc
Nguyễn Văn Đoàn	Thiết Kế UI và xây dựng tính năng cho app: xây dựng giao diện và chức năng cho việc chụp ảnh từ camera, thư viện máy; lưu trữ ảnh chụp và sử dụng nó làm đầu vào cho quá trình phân loại; lưu trữ và quản lý lịch sử chuẩn đoán trong cơ sở dữ liệu hoặc lưu trữ dưới dạng danh sách các mục, hiển thị thông tin lịch sử chuẩn đoán và cho phép người dùng tương tác với các mục trong lịch sử.
Tạ Anh Quân	Xây dựng và tiến hành huấn luyện mô hình học sâu trên dữ liệu đã chuẩn bị, đánh giá, điều chỉnh mô hình để đạt được hiệu suất tốt trên tập kiểm tra, sử dụng mô hình TFLite đã nhúng để phân loại ảnh viêm phổi. Xử lý kết quả phân loại và hiển thị cho người dùng.

6 Tài liệu tham khảo

- [1] Flutter. (n.d.). Flutter documentation.
- [2] Dart. (n.d.). Tutorials.
- [3] Flutter. (n.d.). Data and backend.

- [4] *Das, O.*(2010). Internship report: Android application development for GPS-based location tracker & NITR attendance management system (Report No. 10-11-01). Stanford University.
- [5] *Vignaraj, J.* (2021). Image classifier [Source code].
- [6] *Caldarie, T.* (2020). Flutter_tflite_audio [Source code].
- [7] TensorFlow Lite. (n.d.). Tflite 0.9.0 [Source code].
- [8] *Tsang, S.-H.* (2018, October 2). Review: Inception-v3 - 1st runner up (Image classification) in ILSVRC 2015 [Blog post].
- [9] *Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z.* (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).
- [10] TensorFlow Insights. (2021, December 28). TensorFlow insights - part 6: Custom model - Inception V3 [Blog post]. Medium.
- [11] *Zhang, Y., & Yang, Q.* (2018). A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 31(6), 1075-1089. Document
- [12] *He, K., Zhang, X., Ren, S., & Sun, J.* (2015). Deep residual learning for image recognition. arXiv:1512.03385 [cs]. Document