ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA TOÁN - CƠ - TIN HỌC

BÁO CÁO CUỐI KÌ

Đề tài: Nhận diện bệnh viêm phổi từ ảnh chụp X-Ray dựa trên mô hình phân loại hình ảnh dựa trên Spark

---

Môn học: Quản trị dữ liệu lớn

Sinh viên thực hiện:

Nguyễn Quỳnh Anh (21002119)

Cao Thị Hoài Thương(21002177)

Tôn Nữ Mai Khanh (21000410)

Giáo viên hướng dẫn:

Ngày 11 tháng 6 năm 2024

Mục lục

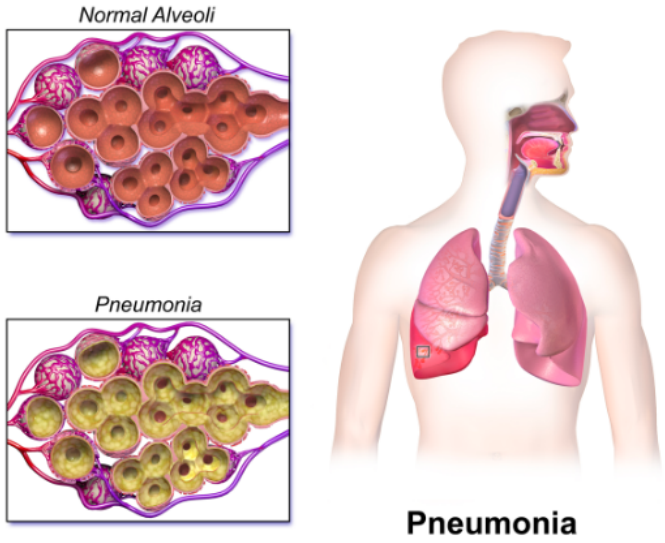
1 Abstract

X-Quang là một trong những kỹ thuật chẩn đoán y tế phổ biến và tiết kiệm chi phí nhất hiện nay. Kể từ khi được Rontgen phát hiện vào năm 1895, X-Quang đã được sử dụng rộng rãi trong các xét nghiệm y tế do nhiều đặc tính mong muốn, chủ yếu là do khả năng xuyên qua các vật chất ít đặc như da nhưng không xuyên qua xương, cho phép thu được hình ảnh rõ ràng trên giấy ảnh. Phân loại chính xác hình ảnh X-quang là một nhiệm vụ quan trọng vì nó là tiền đề cho việc điều trị trong tương lai.

Trong báo cáo này, bộ dữ liệu được sử dụng là bộ dữ liệu ảnh chụp X-quang phần ngực, bao gồm 5.232 hình ảnh X-Quang. Nhóm trình bày các phương pháp xây dựng mô hình, trong đó nhấn mạnh ở các mô hình sử dụng trên Apache Spark để phân loại chính xác một hình ảnh X-quang cho sẵn. Độ chính xác của bộ phân loại dùng thư viện MLlib và BigDL thể hiện được kết quả khả quan hơn, với tỷ lệ dự đoán đúng lần lượt là 75.85% và 84.38%.

2 Introduction

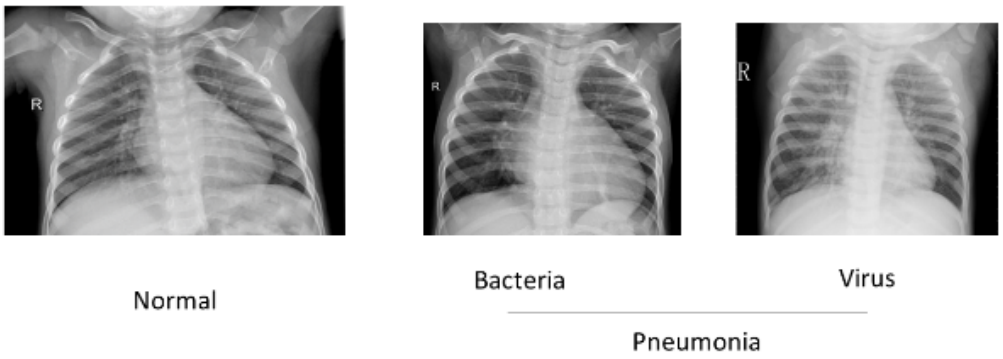
Viêm phổi là một căn bệnh nguy hiểm, lây nhiễm vào phổi làm ảnh hưởng đến quá trình hô hấp bình thường của con người. Theo số liệu thống kê, viêm phổi có thể xảy ra ở trẻ em và người lớn, ảnh hưởng đến khoảng 450 triệu người mỗi năm trên toàn thế giới, và cũng là một trong những nguyên nhân gây tử vong hàng đầu. 2,5 triệu người chết vì viêm phổi vào năm 2019 trong đó gần 1/3 số nạn nhân là trẻ em dưới 5 tuổi. Mức độ viêm phổi có thể là từ nhẹ đến nặng, thậm chí có thể cướp đi cả tính mạng của người bệnh. Các triệu chứng bệnh bao gồm ho, sốt, khó thở, hụt hơi, đau ngực và thở nhanh do các túi khí hoặc phế nang phổi chứa đầy chất mủ (hình 1).



Hình 1: Triệu chứng bệnh

Do đó, việc chẩn đoán nhanh và chính xác rất quan trọng để xác định liệu pháp điều trị kịp thời, hiệu quả cho bệnh nhân. Với công nghệ ngày càng phát triển, ngày càng có nhiều biện pháp được phát triển, trong đó các phương pháp dựa trên X quang là phổ biến và hữu ích nhất. Các kỹ thuật X quang chẩn đoán bệnh phổi bao gồm chụp X-quang ngực, chụp cắt lớp vi tính (CT) và chụp cộng hưởng từ (MRI), trong đó chụp X-quang ngực là hiệu quả và tiết kiệm nhất vì nó sẵn có và đơn giản và không độc hại với người bệnh.

Đồng thời chụp X-quang ngực thường được sử dụng để giúp phân biệt giữa các loại viêm phổi khác nhau (Hình 2) hoặc để theo dõi các bệnh nhân thuộc nhóm nguy cơ có triệu chứng dai dẳng. Tuy nhiên, để có một đội ngũ chuyên gia chẩn đoán bệnh viêm phổi qua hình ảnh X-quang là khó vì nguồn nhân lực, cơ sở vật chất chưa đáp ứng được, nhất là ở các quốc gia chưa phát triển. Vì vậy, việc chẩn đoán bệnh viêm phổi bằng các phương pháp truyền thống rất tốn thời gian, kinh tế và khó có thể chẩn đoán bệnh nhân có bị viêm phổi hay không bằng quy trình chuẩn. [6]



Hình 2: Ảnh chụp X Quang của người bình thường và người đang có bệnh viêm phổi (do vi khuẩn hoặc vi-rút)

Vì vậy, chúng em tìm hiểu và đề xuất phương pháp sử dụng Big data và Deep learning vì đây đều là những lĩnh vực đang phát triển nhanh chóng. Chúng em mong muốn tạo ra một mô hình học máy có giám sát để phân loại thành công ảnh chụp X-quang ngực ở trẻ em để phát hiện bệnh viêm phổi. Để đạt được mục tiêu này, chúng em đã tiếp cận bài toán theo 3 cách khác nhau với:

1. Thư viện TensorFlow Keras.
2. Thư viện MLlib từ Pyspark.
3. Thư viện BigDL từ DLlib.

3 Literature Review

Ảnh chụp X-quang của bệnh nhân được dữ liệu y tế thu thập bao gồm cả hình ảnh phổi bị viêm và phổi bình thường. Mô hình CNN được huấn luyện để có thể trích xuất đặc trưng cho các hình ảnh trong tập dữ liệu. Trong giai đoạn này, các hình ảnh trong truy vấn sẽ được phân loại bằng cách gán một trong hai nhãn P hoặc N (cũng có thể gán 1 hoặc 0). Cụ thể, P và 1 dành cho hình ảnh bệnh nhân bị viêm phổi, N và 0 dành cho trường hợp bình thường. Nhiều trình phân loại khác nhau có thể được sử dụng như Rừng ngẫu nhiên, Hồi quy logistic, v.v.
Một số nghiên cứu đã phát triển mô hình Mạng thần kinh chuyển đổi (CNN) để phát hiện các bệnh về lồng ngực từ hình ảnh X-quang ngực bằng nhiều bộ dữ liệu khác nhau. Rubin và cộng sự. đã sử dụng bộ dữ liệu MIMIC-CXR và đạt được AUC (là chỉ số được tính toán dựa trên đường cong ROC (receiving operating curve) nhằm đánh giá khả năng phân loại của mô hình tốt như thế nào,

càng gần 1 thì càng tốt) trung bình là 0,72 và 0,688 cho các chế độ xem X-quang khác nhau. Họ chia tập dữ liệu của mình thành các tập huấn luyện, kiểm tra và xác thực, đồng thời sử dụng tính năng tăng cường dữ liệu và chuẩn hóa pixel để nâng cao hiệu suất.

Trong một nghiên cứu khác, Pranaya và các cộng sự đã tìm hiểu sử dụng Mạng thần kinh chuyển đổi (CNN) để phát hiện bệnh viêm phổi trong chụp X quang ngực. CNN được xây dựng từ đầu để tự động xác định các trường hợp viêm phổi trong hình ảnh X-quang ngực (CXR) bằng cách trích xuất các đặc điểm hình ảnh thông qua các lớp ReLU tích chập và các max pooling layers. Các lớp tích chập sử dụng các bộ lọc để trích xuất các đặc điểm hình ảnh cụ thể, trong khi các lớp kích hoạt ReLU nâng cao quá trình học tập. Các pooling layers thu nhỏ dữ liệu đầu vào để giảm kích thước không gian và số lượng tham số. Fully connected layers đã thiết lập kết nối giữa các neuron để phân loại bằng Softmax hoặc Máy vectơ hỗ trợ (SVM).

Quá trình tiền xử lý bao gồm tải hình ảnh CXR, xử lý các giá trị null và thay đổi kích thước hình ảnh thành kích thước tiêu chuẩn phù hợp với mô hình. Tập dữ liệu được chia thành các tập huấn luyện, kiểm tra và xác nhận. Dữ liệu huấn luyện được sử dụng để huấn luyện thuật toán CNN, kiểm tra dữ liệu để xác minh kết quả và xác thực để đánh giá hiệu suất mô hình trên dữ liệu mới. Việc tinh chỉnh mô hình CNN được thực hiện bằng cách tối ưu hóa các tham số của nó. Mô hình CNN gán trọng số tương đối cho các đặc điểm hình ảnh khác nhau, phân biệt giữa các đối tượng khác nhau một cách tuyệt vời. Khả năng xây dựng các biểu diễn bên trong của hình ảnh hai chiều đã tạo điều kiện thuận lợi cho việc tìm hiểu vị trí và tỷ lệ, giúp nó có hiệu quả cao đối với các tác vụ như nhận dạng hình ảnh, phát hiện đối tượng và phân đoạn.

4 Theoretical Foundation

4.1 Mô hình mạng nơ-ron Keras

Mạng thần kinh Keras là một loại mô hình học sâu được triển khai bằng thư viện Keras, hiện được tích hợp vào TensorFlow. Keras đơn giản hóa việc tạo và đào tạo mạng lưới thần kinh. Nó cung cấp API cấp cao để xây dựng và định cấu hình mạng lưới thần kinh, cho phép các nhà phát triển xác định các lớp, kết nối và quy trình đào tạo một cách dễ dàng, khiến nó trở thành lựa chọn phổ biến để phát triển các mô hình học sâu.

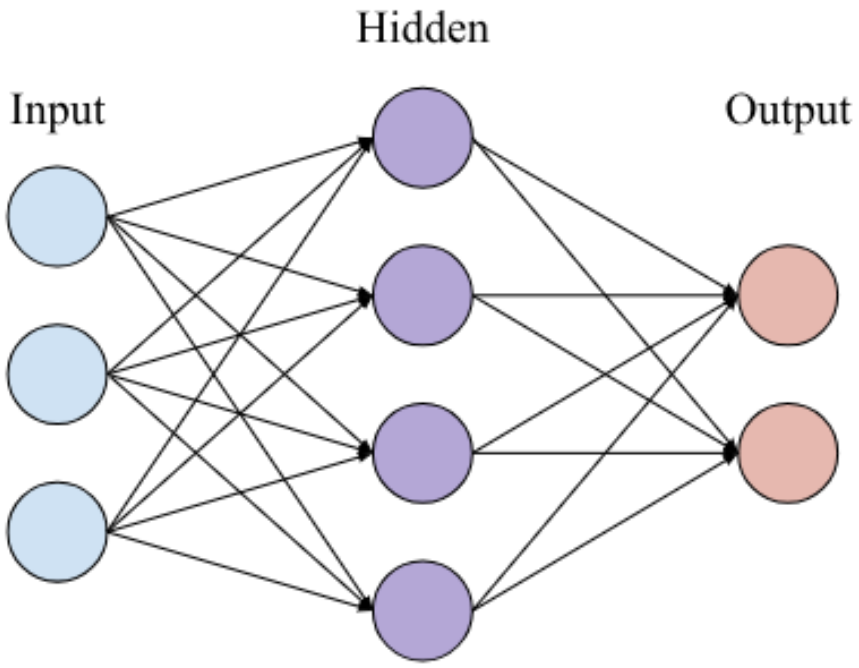
4.1.1 Mạng neuron Nhân tạo (ANNs) và mạng neuron tích chập (CNNs)

- Mạng neuron Nhân tạo (ANNs) là các hệ thống xử lý tính toán mà được lấy cảm hứng mạnh mẽ từ cách các hệ thống thần kinh sinh học hoạt động, chẳng hạn như não người. ANNs chủ yếu bao gồm một số lượng lớn các nút tính toán được kết nối chặt chẽ (được gọi là neuron), chúng hoạt động cùng nhau theo cách phân tán để học từ đầu vào nhằm tối ưu hóa đầu ra cuối cùng.

  – Lớp nhập (input layer) cung cấp cho mạng các số liệu cần thiết. Số lượng neuron trong lớp nhập tương ứng với số lượng thông số đầu vào được cung cấp cho mạng và các thông số đầu vào này được giả thiết ở dạng vector.

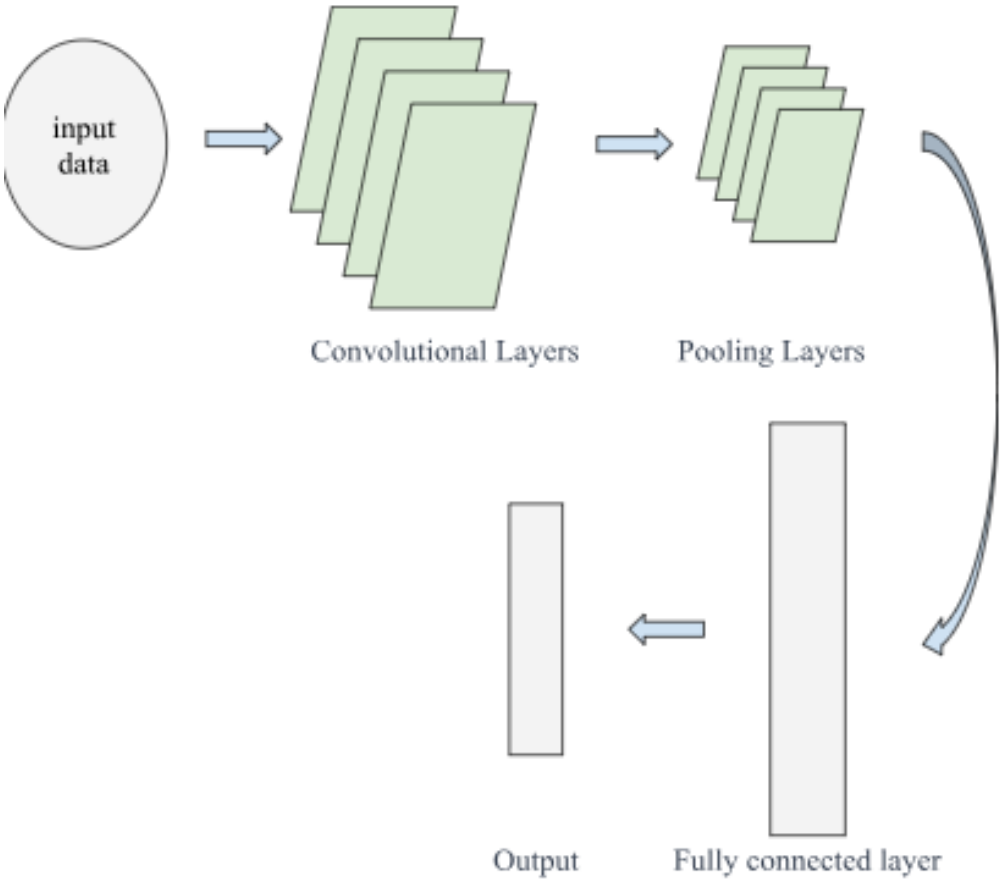
  – Lớp ẩn (hidden layer) chứa các neuron ẩn giúp kết nối giá trị đầu vào đến giá trị đầu ra. Một mạng neuron có thể có một hoặc nhiều lớp ẩn chịu trách nhiệm chính cho việc xử lý các neuron của lớp nhập và đưa các thông tin đến neuron của lớp xuất. Các neuron này thích ứng với việc phân loại và nhận diện mối liên hệ giữa thông số đầu vào và thông số đầu ra.

  – Lớp xuất (output layer) chứa các neuron đầu ra nhằm chuyển thông tin đầu ra của các tính toán từ ANN đến người dùng. Một ANN có thể được xây dựng để có nhiều thông số đầu ra. Cấu trúc cơ bản của một ANN có thể được mô hình hóa như hiển thị trong hình 3



Hình 3: Cấu trúc ANN

- Mạng thần kinh chuyển đổi (CNN) là một lớp mạng thần kinh chuyên biệt được thiết kế chủ yếu để xử lý dữ liệu giống như lưới có cấu trúc, chẳng hạn như hình ảnh. CNN tương tự như ANN có một lớp đầu vào và một lớp đầu ra với nhiều lớp ẩn ở giữa. Chúng đã trở thành nền tảng cho các nhiệm vụ thị giác máy tính khác nhau nhờ khả năng nắm bắt các phân cấp không gian trong dữ liệu. Mạng CNN tập hợp các lớp Convolution chồng lên nhau và sử dụng các hàm nonlinear activation như ReLU để kích hoạt các trọng số trong các node. Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo (hình 4)



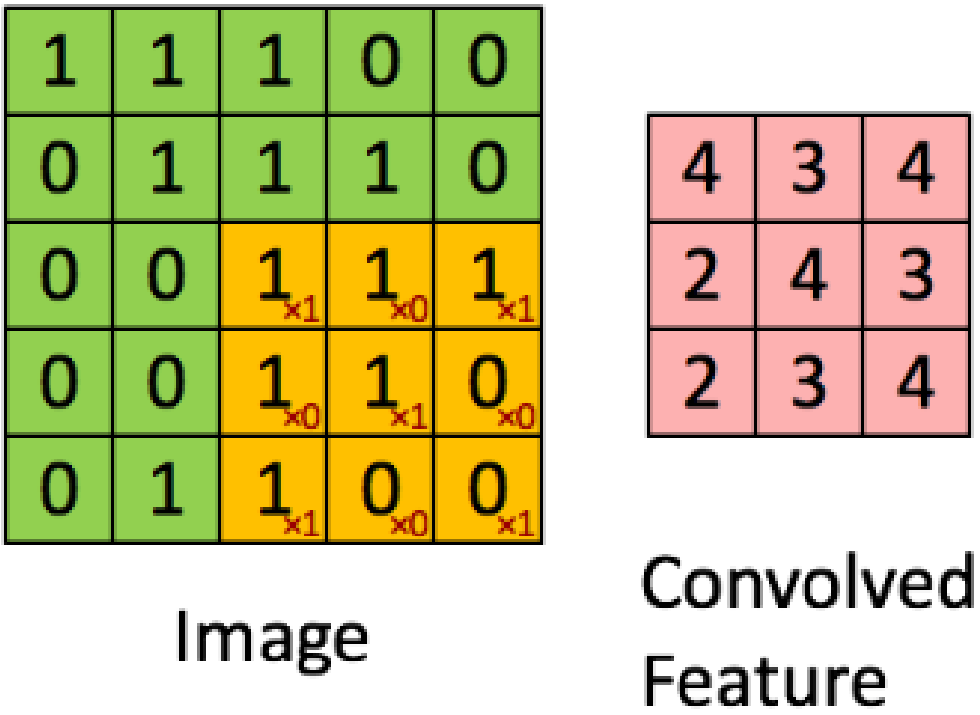
Hình 4: kiến trúc CNN

4.1.2 Lớp tích chập

Các lớp tích chập thường được xếp chồng lên nhau trong CNN, tạo thành hệ thống phân cấp trong đó các lớp trước đó học các tính năng đơn giản như các cạnh hoặc kết cấu, trong khi các lớp sâu hơn tìm hiểu các tính năng trừu tượng và phức tạp hơn. Tích chập cho phép phát hiện các tính năng bất kể vị trí của chúng trong đầu vào.

Các lớp tích chập đóng một vai trò quan trọng trong cách CNN hoạt động. Tích chập duy trì mối quan hệ giữa các pixel bằng cách tìm hiểu các tính năng hình ảnh bằng cách sử dụng các ô vuông nhỏ của dữ liệu đầu vào. Nó là 1 phép toán có 2 đầu vào như ma trận hình ảnh và 1 bộ lọc hoặc hạt nhân (kernel).

Trong hình ảnh ví dụ 5, ma trận bên trái là một hình ảnh trắng đen được số hoá. Ma trận có kích thước 5x5 và mỗi điểm ảnh có giá trị 1 hoặc 0 là giao điểm của dòng và cột. Convolution hay tích chập là nhân từng phần tử trong ma trận 3. Kernel, filter là một tra trận có kích thước nhỏ như trong ví dụ trên là 3x3.



Hình 5: Ví dụ về lớp tích chập

Convolution hay tích chập là nhân từng phần tử bên trong ma trận 3x3 với ma trận bên trái. Kết quả được một ma trận gọi là Convolved feature được sinh ra từ việc nhân ma trận Filter với ma trận ảnh 5x5 bên trái.

Mỗi hình ảnh đầu vào đều có những đặc điểm nhất định và những đặc điểm này được ghi lại trong lớp chập. Sự kết hợp của 1 hình ảnh với các bộ lọc khác nhau có thể thực hiện các hoạt động như phát hiện cạnh, làm mờ và làm sắc nét bằng cách áp dụng các bộ lọc.

Nói chung, các lớp tích chập sẽ tích chập hình ảnh đầu vào để thu được kết quả sau đó được chuyển sang các lớp tiếp theo.

- Phép tích chập:

  Theo toán học, tích chập là phép toán tuyến tính, cho ra kết quả là một hàm bằng việc tính toán dựa trên hai hàm đã có (f và g).

  Ví dụ: đối với phép lọc ảnh, phép tích chập giữa ma trận lọc và ảnh, cho ra kết quả ảnh đã được xóa nhiễu (làm mờ).

  Công thức tích chập giữa hàm ảnh $f(x, y)$ và bộ lọc $k(x, y)$ (kích thước mn):
  $k(x, y) \times f(x, y) = \sum_{u=-m/2}^{m/2} \sum_{v=-n/2}^{n/2} k(u, v) f(x - u, y - v)$

Thành phần không thể thiếu của phép tích chập là ma trận kernel (bộ lọc). Điểm neo (anchor point) của kernel sẽ quyết định vùng ma trận tương ứng trên ảnh để tích chập, thông thường anchor point được chọn là tâm của kernel. Giá trị mỗi phần tử trên kernel được xem như là hệ số tổ hợp với lần lượt từng giá trị độ xám của điểm ảnh trong vùng tương ứng với kernel.

4.1.3 ReLU

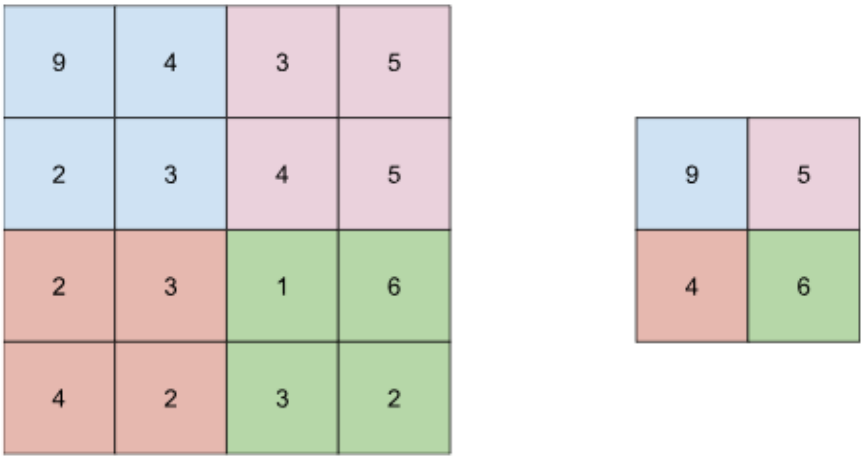
ReLU (Rectified Linear Unit) là 1 hàm phi tuyến.

- Với đầu vào là Feature Maps từ Convolution Layer
- Với đầu ra là:
  $f(x) = max(0, x)$

Với cách tính toán là áp dụng hàm ReLU cho mỗi giá trị trong feature map, nếu giá trị là âm, chuyển thành 0; nếu là dương, giữ nguyên giá trị.
Mục đích chính của việc lọc bỏ các giá trị bằng 0 của hàm ReLU trong mạng CNN là tạo ra tính phi tuyến tính và kích hoạt các đặc trưng quan trọng trong dữ liệu. Các giá trị âm trong đầu ra của lớp tích chập thường đại diện cho các đặc trưng không quan trọng hoặc nhiễu. Bằng cách chuyển chúng thành 0, hàm ReLU loại bỏ các đặc trưng không quan trọng này và tập trung vào các đặc trưng quan trọng hơn. Điều này giúp giảm khả năng overfitting và cải thiện khả năng tổng quát hóa của mô hình.

4.1.4 Pooling Layer

Sau hàm kích hoạt, thông thường chúng ta sử dụng tầng pooling. Một số loại pooling layer phổ biến như là max pooling, average pooling, và trong bài toán này chúng em sử dụng max pooling. Chức năng chính của tầng này là giảm chiều của tầng trước đó. Với một pooling có kích thước 2x2, cần phải trượt filter 2x2 này trên những vùng ảnh có kích thước tương tự rồi sau đó tính max, hay average cho vùng ảnh đó.
Ý tưởng đằng sau tầng pooling là vị trí tuyệt đối của những đặc trưng trong không gian ảnh không còn cần cần thiết, thay vào đó vị trí tương đối giữa các đặc trưng đã đủ để phân loại đối tượng. Hơn giảm tầng pooling có khả năng giảm chiều cực kì nhiều, làm hạn chế overfit, và giảm thời gian huấn luyện tốt.

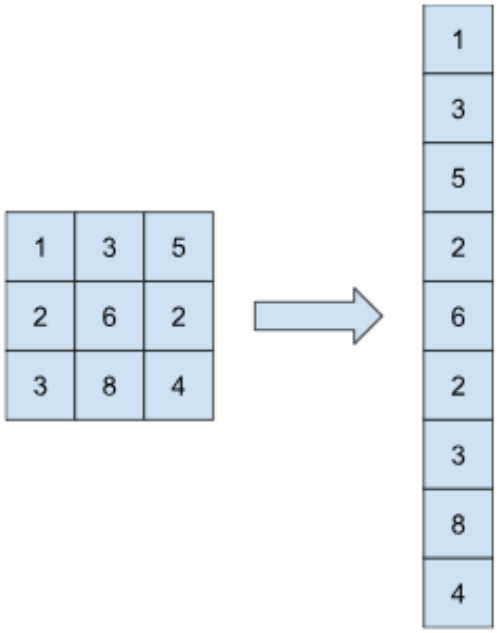


Hình 6: Ví dụ cho tính toán pooling layer

Ví dụ 6: Max pooling sẽ chọn giá trị lớn nhất từ mỗi vùng nhỏ của đặc trưng sau lớp ReLU, giữ lại các đặc điểm quan trọng.

4.1.5 Flatten Layer

Sau khi có được đầu ra từ lớp tích chập, chúng ta cần chuyển đổi tensor 3D này thành một vector 1 chiều để có thể sử dụng cho các layer fully connected (Dense). Flatten layer chính làm công việc này bằng cách "làm phẳng" tensor, biến đổi nó từ (số chiều, chiều rộng, chiều cao) thành một vector 1 chiều.



Hình 7: Sau khi chuyển đổi thu được vector 1 chiều

4.1.6 Fully Connected Layer

Tầng cuối cùng của mô hình CNN trong bài toán phân loại ảnh là tầng fully connected layer. Tầng này có chức năng chuyển ma trận đặc trưng ở tầng trước thành vector chứa xác suất của các đối tượng cần được dự đoán.

Softmax Layer: chuyển đổi đầu ra của mô hình thành một phân phối xác suất.

Lớp Softmax thực hiện hai công việc chính:

- Chuyển đổi đầu vào thành một phân phối xác suất: Lớp Softmax áp dụng hàm exponential (mũ) lên đầu vào để tăng giá trị của các node và sau đó chuẩn hóa chúng để tạo thành một phân phối xác suất, trong đó tổng các giá trị đầu ra bằng 1.

- Tạo ra xác suất dự báo: Khi được áp dụng cho đầu ra của một mạng nơ-ron đa lớp, lớp Softmax biến đổi các giá trị đầu ra thành xác suất dự báo cho các lớp khác nhau. Các giá trị đầu ra cao hơn sẽ có xác suất cao hơn và được coi là lớp được dự báo.

4.2 Mô hình sử dụng PySpark từ thư viện MLlib

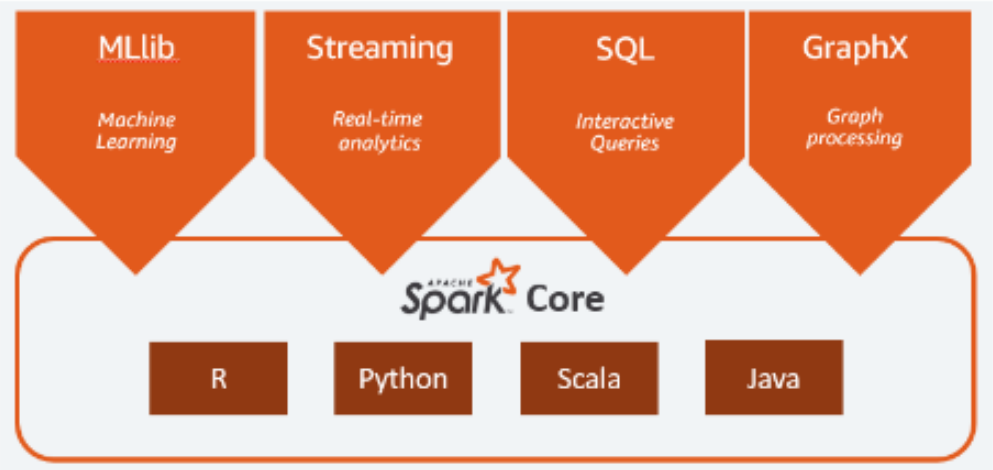
4.2.1 Apache Spark

Apache Spark ra đời năm 2009, là một hệ thống xử lý phân tán nguồn mở được sử dụng cho các công việc liên quan đến dữ liệu lớn. Hệ thống này sử dụng khả năng ghi vào bộ nhớ đệm nằm trong bộ nhớ và thực thi truy vấn tối ưu hóa nhằm giúp truy vấn phân tích nhanh dữ liệu có kích thước bất kỳ.

Apache Spark có rất nhiều ưu điểm khiến nó trở thành một trong những dự án hoạt động tích cực nhất trong hệ sinh thái Hadoop. Apache Spark cung cấp một loạt các dịch vụ đa dạng và mạnh mẽ. Xử lý dữ liệu là một trong những lợi ích hàng đầu của Spark, cho phép xử lý cả theo lô và thời gian thực đem lại hiệu suất cao. Tốc độ chạy các chương trình Spark nhanh hơn tới 100 lần so với Hadoop MapReduce trong bộ nhớ hoặc nhanh hơn 10 lần trên đĩa.

Thêm vào đó, Spark có tính tương thích cao, có khả năng kết hợp với mọi nguồn dữ liệu và định dạng tệp được hỗ trợ bởi cụm Hadoop, mở ra khả năng tiếp cận và xử lý dữ liệu từ nhiều nguồn khác nhau. Hơn nữa, Spark hỗ trợ nhiều ngôn ngữ lập trình như Java, Scala, Python và R, tạo điều kiện thuận lợi cho các nhà phân tích dữ liệu sử dụng ngôn ngữ mà họ ưa thích. Ngoài ra, phân tích thời gian thực cũng là một lợi thế nổi bật khác của Apache Spark, cho phép xử lý dữ liệu đến từ các luồng sự kiện thời gian thực với tốc độ hàng triệu sự kiện mỗi giây. Ví dụ như dữ liệu từ Facebook hoặc các hoạt động chia sẻ, đăng bài liên tục trên mạng xã hội. Spark có thể làm điều này một cách hiệu quả, khẳng định sức mạnh của nó trong việc xử lý trực tiếp dữ liệu thời gian thực.

Apache Spark cung cấp một bộ thư viện phong phú bao gồm Spark SQL để truy vấn dữ liệu có cấu trúc, Spark Streaming để xử lý dữ liệu theo thời gian thực, MLlib để học máy và GraphX để xử lý đồ thị.



Hình 8: Tổng quan về Apache Spark

Một số công cụ của PySpark phục vụ cho việc huấn luyện mô hình:

- ParamGridBuilder: Đây là một công cụ trong PySpark cho phép bạn xây dựng một lưới các giá trị siêu tham số để thử nghiệm. Trong trường hợp này, nó sẽ được sử dụng để xây dựng các giá trị siêu tham số để điều chỉnh mô hình, ví dụ như số lượng cây trong Random Forest hay hệ số trong Logistic Regression.

- CrossValidator: Là một công cụ trong PySpark giúp thực hiện quá trình xác thực chéo. Nó chia tập dữ liệu thành các fold, và sau đó huấn luyện và đánh giá mô hình trên mỗi fold với các giá trị siêu tham số khác nhau. Kết quả là mô hình tốt nhất dựa trên các giá trị siêu tham số đã chọn.

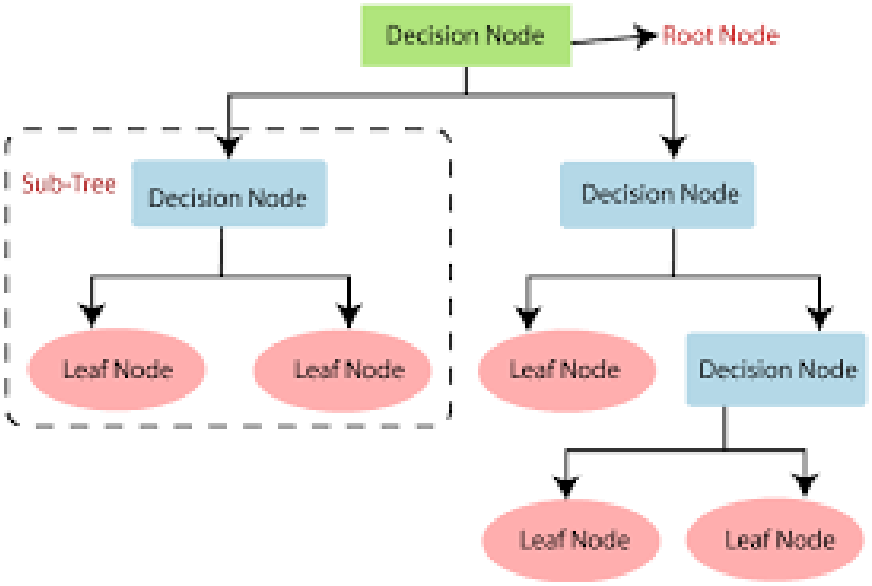
4.2.2 MLlib

[1] Spark bao gồm MLlib, một thư viện thuật toán để thực hiện máy học trên dữ liệu ở quy mô lớn. Các nhà khoa học dữ liệu có thể đào tạo các mô hình học máy bằng cách sử dụng R hoặc Python trên bất kỳ nguồn dữ liệu Hadoop nào, được lưu bằng MLlib và được nhập vào một quy trình chạy trên Java hoặc Scala. Spark cho phép tính toán tương tác nhanh chạy trong bộ nhớ và được thiết kế để hỗ trợ học máy nhanh. Các thuật toán bao gồm khả năng phân loại, hồi quy, phân cụm, lọc cộng tác và khai thác mẫu. Theo các so sánh benchmark Spark MLlib nhanh hơn 9 lần so với phiên bản chạy trên Hadoop (Apache Mahout).

1. Decision Tree[4]:

Decision Tree là một kiểu mô hình dự báo (predictive model), nghĩa là một ánh xạ từ các quan sát về một sự vật/hiện tượng tới các kết luận về giá trị mục tiêu của sự vật/hiện tượng. Decision Tree có cấu trúc hình cây và là một sự tượng trưng của một phương thức quyết định cho việc xác định lớp các sự kiện đã cho. Mỗi nút của cây chỉ ra một tên lớp hoặc một phép thử cụ thể, phép thử này chia không gian các dữ liệu tại nút đó thành các kết quả có thể đạt được của phép thử. Mỗi tập con được chia ra là không gian con của các dữ liệu được tương ứng với vấn đề con của sự phân loại. Sự phân chia này thông qua một cây con tương ứng. Quá trình xây dựng thuật toán decision tree có thể xem như là một chiến thuật chia để trị cho sự phân loại đối tượng. Một cây quyết định có thể mô tả bằng các khái niệm nút và đường nối các nút trong cây. Mỗi nút của cây quyết định có thể là:

- Nút lá (leaf node) hay còn gọi là nút trả lời (answer node), nó biểu thị cho một lớp các trường hợp (bản ghi), nhãn của nó là tên của lớp.
- Nút không phải là lá (non-leaf node) hay còn gọi là nút trong (inner node), nút này xác định một phép thử thuộc tính (attribute test), nhãn của nút này có tên của thuộc tính và sẽ có một nhánh (hay đường đi) nối nút này đến cây con (subtree) ứng với mỗi kết quả có thể có của phép thử. Nhãn của nhánh này chính là giá trị của thuộc tính đó. Nút không phải lá nằm trên cùng là nút gốc (root node).



Hình 9: Cấu trúc Decision Tree

Một cây quyết định sử dụng để phân loại dữ liệu bằng cách bắt đầu đi từ nút gốc của cây và

đi xuyên qua cây theo các nhánh cho tới khi gặp nút lá, khi đó ta sẽ được lớp của dữ kiện đang xét. Giả sử chúng ta có một ví dụ minh họa một cây quyết định để xác định khả năng bị bệnh viêm phổi. Chúng ta có một bệnh nhân, hãy thử dùng cây quyết định này để dự đoán khả năng bị bệnh của người đó. Câu hỏi đầu tiên là: "Tuổi của bệnh nhân trong khoảng nào?". Giả sử bệnh nhân đó 50 tuổi, chúng ta đi tiếp sang nhánh ngoài cùng bên phải. Câu hỏi tiếp theo là: "Người đó có hút thuốc không?". Giả sử câu trả lời là có. Điều này có nghĩa là chúng ta sẽ đi theo nhánh bên phải. Khi đó, vì chúng ta đã đi đến nút lá, chúng ta có thể dự đoán rằng người này có nguy cơ cao bị bệnh viêm phổi.

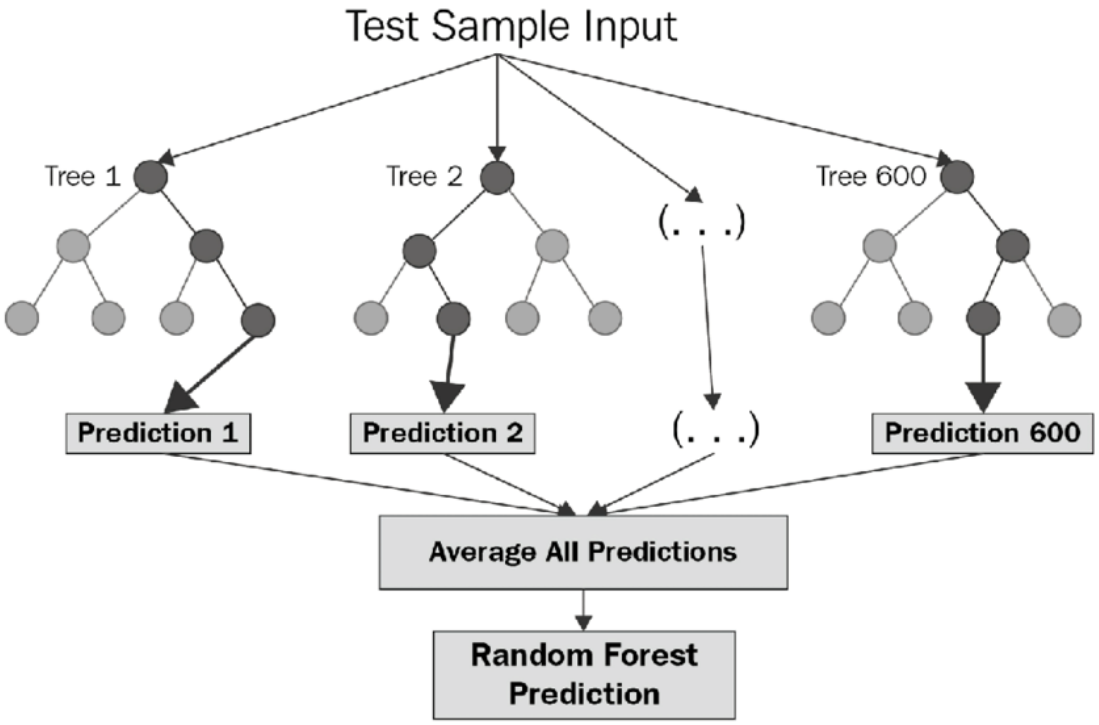
Dù có độ chính xác khá cao nhưng cây quyết định tồn tại những hạn chế lớn đó là:

- Dễ xảy ra quá khớp nếu số lượng các đặc trưng để hỏi lớn. Khi độ sâu của cây quyết định không bị giới hạn thì có thể tạo ra những node lá chỉ có một vài quan sát. Những kết luận dự báo từ chúng thường chỉ đúng trên tập huấn luyện mà không đúng trên tập kiểm tra.
- Trong tình huống bộ dữ liệu có số lượng biến lớn. Một cây quyết định có độ sâu giới hạn (để giảm thiểu quá khớp) thường bỏ sót những biến quan trọng.

Nếu như sức mạnh của một cây quyết định là yếu thì hợp sức của nhiều cây quyết định sẽ trở nên mạnh mẽ hơn. Ý tưởng của sự hợp sức đã hình thành nên mô hình rừng cây (Random Forest).

2. Random Forest[3]:

   Mô hình rừng cây được huấn luyện dựa trên sự phối hợp giữa luật kết hợp (ensembling) và quá trình lấy mẫu tái lặp (bootstrapping). Cụ thể thuật toán này tạo ra nhiều cây quyết định mà mỗi cây quyết định được huấn luyện dựa trên nhiều mẫu con khác nhau và kết quả dự báo là bầu cử (voting) từ toàn bộ những cây quyết định. Như vậy một kết quả dự báo được tổng hợp từ nhiều mô hình nên kết quả của chúng sẽ không bị chệch. Đồng thời kết hợp kết quả dự báo từ nhiều mô hình sẽ có phương sai nhỏ hơn so với chỉ một mô hình. Điều này giúp cho mô hình khắc phục được hiện tượng quá khớp. Tính ngẫu nhiên là một yếu tố quan trọng để đảm bảo rằng quần thể cây đa dạng. Một trong những lợi thế lớn nhất của rừng ngẫu nhiên là tính linh hoạt của nó. Nó có thể được sử dụng cho cả nhiệm vụ hồi quy và phân loại.



Mô hình rừng cây sẽ áp dụng cả hai phương pháp học kết hợp (ensemble learning) và lấy mẫu tái lập (bootstrapping).

Kết quả dự báo từ mô hình rừng cây là sự kết hợp của nhiều cây quyết định nên chúng tận dụng được trí thông minh đám đông và giúp cải thiện độ chính xác so với chỉ sử dụng một mô hình cây quyết định.

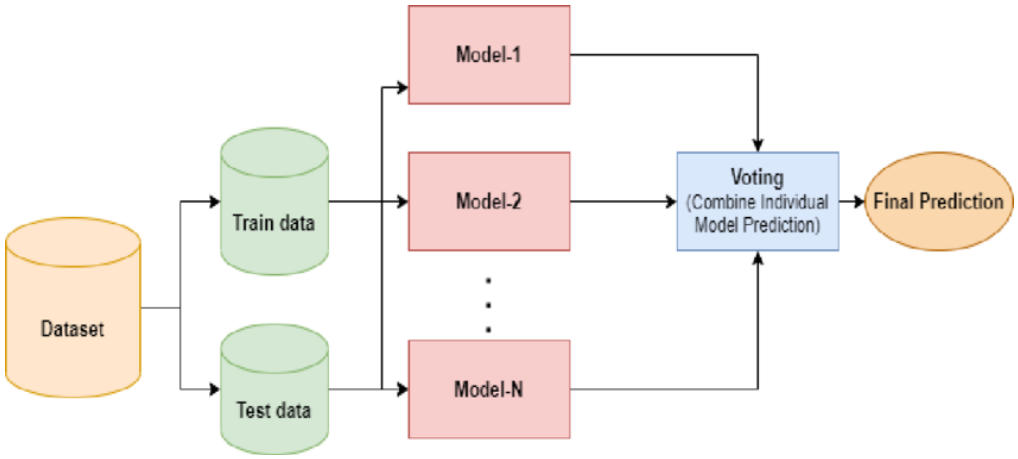
Nếu như mô hình cây quyết định thường bị nhạy cảm với dữ liệu ngoại lai (outlier) thì mô hình rừng cây được huấn luyện trên nhiều tập dữ liệu con khác nhau, trong đó có những tập được loại bỏ dữ liệu ngoại lai, điều này giúp cho mô hình ít bị nhạy cảm với dữ liệu ngoại lai hơn.

Bên cạnh đó, các bộ dữ liệu được sử dụng từ những cây quyết định đều xuất phát từ dữ liệu huấn luyện nên quy luật học được giữa các cây quyết định sẽ gần tương tự như nhau và tổng hợp kết quả giữa chúng không có xu hướng bị chệch.

(a) Ensemble model:

Mô hình hóa tập hợp là một quá trình trong đó nhiều mô hình được tạo ra để dự đoán kết quả, bằng cách sử dụng nhiều thuật toán mô hình hóa khác nhau hoặc sử dụng các bộ dữ liệu huấn luyện khác nhau. Mô hình tập hợp sau đó tổng hợp dự đoán của từng

mô hình cơ sở và đưa ra dự đoán cuối cùng. Động lực của việc sử dụng các mô hình tập hợp là để giảm sai số tổng quát của dự đoán. Miễn là các mô hình cơ sở đa dạng và độc lập, sai số dự đoán của mô hình sẽ giảm khi sử dụng phương pháp tổng hợp.

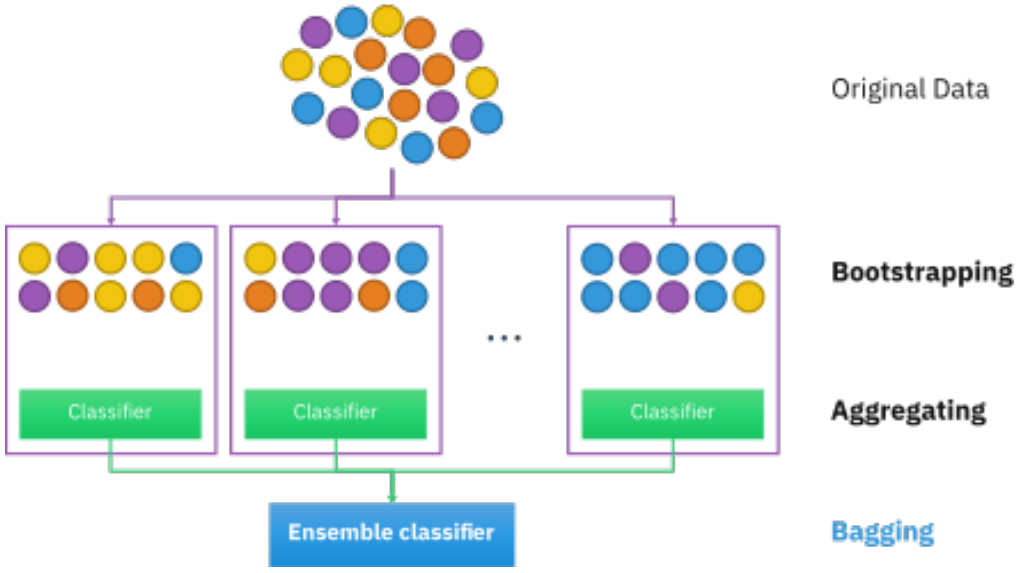


Thông thường những kết quả từ mô hình kết hợp sẽ tốt hơn so với chỉ sử dụng một mô hình bởi chúng ta đang vận dụng trí thông minh đám đông. Mặc dù mô hình tập hợp có nhiều mô hình cơ sở trong mô hình nhưng nó hoạt động và thực hiện như một mô hình duy nhất.

(b) Bagging:

Bagging (Bootstrap aggregating) là tổng hợp các bootstrap sử dụng cách tiếp cận xây dựng mỗi bộ phân loại một cách độc lập với nhau, sau đó sử dụng phương pháp bỏ phiếu để chọn ra kết quả cuối cùng của bộ kết hợp. Tức là mỗi bộ phân loại cơ bản sẽ được xây dựng độc lập với các bộ phân loại khác bằng cách thay đổi tập dữ liệu huấn luyện đầu vào, thay đổi các đặc trưng trong tập huấn luyện. Bagging tạo ra các bộ phân loại từ các tập mẫu con có lặp từ tập mẫu ban đầu (sử dụng bootstrap lấy mẫu có hoàn lại) và một thuật toán học máy, mỗi tập mẫu sẽ tạo ra một bộ phân loại cơ bản.

Các bộ phân loại sẽ được kết hợp bằng phương pháp bỏ phiếu theo số đông. Tức là khi có một mẫu cần được phân loại, mỗi bộ phân loại sẽ cho ra một kết quả. Tức là dự đoán của từng cây riêng lẻ được coi là một phiếu bầu. Và kết quả nào xuất hiện nhiều nhất/trung bình các kết quả sẽ được lấy làm kết quả của bộ kết hợp.



4.3 BigDL[2]

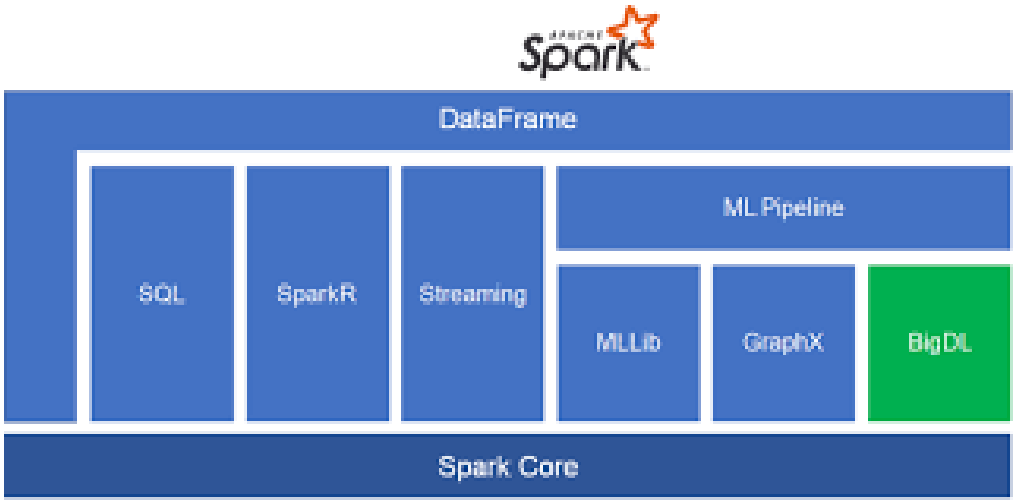
(a) Analytics Zoo:

Analytics Zoo cung cấp nền tảng AI và phân tích thống nhất giúp kết hợp liền mạch các chương trình Spark, TensorFlow. Nó được thiết kế để đơn giản hóa và đẩy nhanh quá trình phát triển, triển khai và mở rộng quy mô ứng dụng deep learning và big data.

Với trọng tâm là thu hẹp khoảng cách giữa xử lý dữ liệu lớn truyền thống và khả năng AI hiện đại, Analytics Zoo cung cấp API cấp cao tích hợp liền mạch Spark, TensorFlow, Keras và các thư viện phổ biến khác. Tính linh hoạt của nó cho phép người dùng thực hiện phân tích từ đầu đến cuối, từ nhập và xử lý trước dữ liệu đến phát triển và triển khai mô hình trên quy mô lớn.

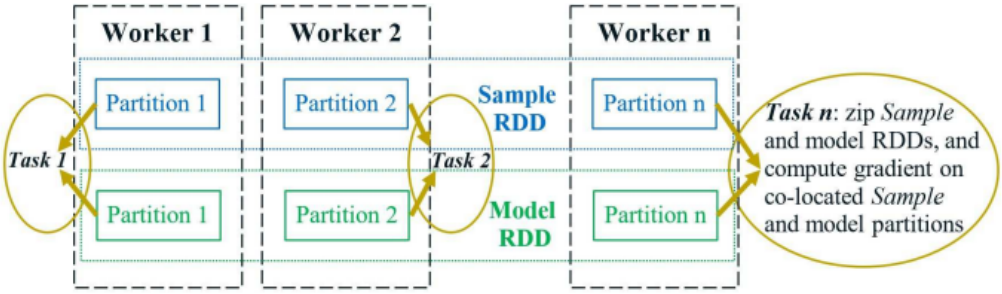
(b) BigDL:

BigDL là một khung học sâu phân tán dành cho các nền tảng và quy trình làm việc dữ liệu lớn. Nó được triển khai như một thư viện trên Apache Spark và cho phép người dùng viết các ứng dụng deep learning quy mô lớn của họ (bao gồm đào tạo mô hình, chỉnh sửa và kết luận) dưới dạng các chương trình Spark tiêu chuẩn, có thể chạy trực tiếp trên dữ liệu lớn hiện có (Hadoop hoặc cụm Spark). BigDL cung cấp hỗ trợ toàn diện cho các công nghệ học sâu (hoạt động mạng lưới thần kinh, các lớp, tổn thất và tối ưu hóa); đặc biệt, người dùng có thể chạy trực tiếp các mô hình hiện có được xác định trong các khung khác (chẳng hạn như TensorFlow, Keras,...) trên Spark theo kiểu phân tán.

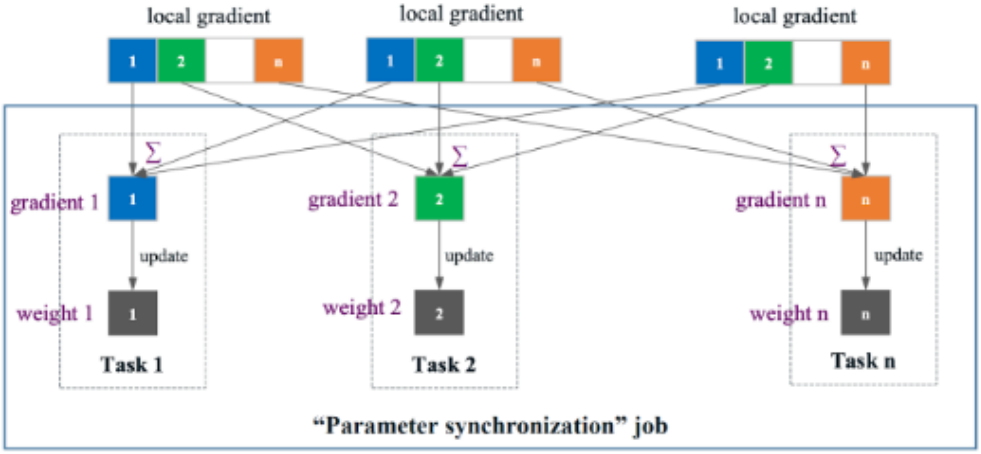


Hình 10: BigDL là một thư viện thuộc Spark bên cạnh MLlib

BigDL cũng cung cấp khả năng tích hợp liền mạch các công nghệ deep learning vào hệ sinh thái dữ liệu lớn. Chương trình BigDL không chỉ có thể tương tác trực tiếp với các thành phần khác nhau trong khung Spark (ví dụ: DataFrames, Spark Streaming, v.v.), nó còn có thể chạy trực tiếp trong nhiều loại framework (chẳng hạn như Apache Kafka, v.v.). Kể từ nguồn mở đầu tiên vào ngày 30 tháng 12 năm 2016, BigDL đã cho phép nhiều người dùng trong cộng đồng xây dựng các ứng dụng học sâu của họ (ví dụ: phát hiện đối tượng, phát hiện gian lận, v.v.) trên Spark và nền tảng dữ liệu lớn. BigDL mô hình hóa dữ liệu huấn luyện dưới dạng RDD của các mẫu, được phân vùng tự động và có khả năng được lưu vào bộ nhớ trên cụm Spark. Ngoài ra, để triển khai đào tạo song song dữ liệu, BigDL cũng xây dựng RDD gồm các mô hình, mỗi mô hình là bản sao của mô hình mạng thần kinh ban đầu. Mô hình và RDD mẫu được phân vùng và đặt cùng vị trí trên toàn cụm. Do đó, trong mỗi lần lặp lại của quá trình đào tạo mô hình, một công việc Spark "tiến lùi mô hình" duy nhất có thể áp dụng toán tử zip chức năng cho các phân vùng của mô hình và RDD mẫu, đồng thời tính toán độ dốc song song cho từng bản sao mô hình (sử dụng một lô nhỏ dữ liệu trong phân vùng mẫu cùng vị trí), như được minh họa trong hình 12. Đồng bộ hóa tham số là một hoạt động quan trọng về hiệu suất để đào tạo song song dữ liệu (về tốc độ và khả năng mở rộng). Mỗi gradient cục bộ (được tính toán bởi một tác vụ trong công việc "tiến lùi mô hình") được chia đều thành N phân vùng; sau đó mỗi tác vụ n trong công việc "đồng bộ hóa tham số" sẽ tổng hợp các gradient cục bộ này và cập nhật trọng số cho phân vùng thứ n.



Hình 11: Quy trình của mô hình spark forward-backward, tính toán song song đạo hàm địa phương của từng bản sao



Hình 12: Quá trình đồng bộ hoá thông số của BigDL

Lợi ích chính của việc sử dụng BigDL và Analytics Zoo để phát triển AI là khả năng tích hợp liền mạch với Apache Spark, cho phép các nhà phát triển tận dụng khả năng xử lý dữ liệu mạnh mẽ của Spark trong quy trình AI của họ. Ngoài ra, Model Zoo và các mô hình tích hợp trong Analytics Zoo cung cấp nhiều mô hình được đào tạo trước và các trường hợp sử dụng tham chiếu, giúp các nhà phát triển bắt đầu và đẩy nhanh quá trình phát triển AI dễ dàng hơn.

5 Methodology

Sử dụng kiến thức cơ sở lý thuyết như trên, nhóm em thực hiện xây dựng mô hình nhằm đưa ra dự đoán của ảnh chụp X-quang bất kỳ.

5.1 Mô tả tập dữ liệu

Báo cáo nhóm em sử dụng một bộ dữ liệu hình ảnh X-quang ngực nhi khoa từ một nghiên cứu được viết bởi Kermany và đồng nghiệp (Kermany et.al, 2018).

Về đặc điểm của bộ dữ liệu, dữ liệu được thu thấp từ một nhóm bệnh nhân nhi độ tuổi từ 1 đến 5 được chọn từ Trung tâm Y tế Phụ nữ và Trẻ em Guangzhou ở Guangzhou. Tất cả các ảnh X-quang hướng trước - hướng sau được chụp như một phần của điều trị y tế thường xuyên, và chỉ những bức ảnh đáp ứng tiêu chuẩn kiểm soát chất lượng được lựa chọn.

Các hình ảnh được kiểm tra và chẩn đoán bởi hai bác sĩ chuyên gia và được chấp thuận bởi một bác sĩ chuyên gia thứ ba để đảm bảo độ chính xác. Các chẩn đoán được sử dụng để đưa ra quyết định chuyển giới, bao gồm "chuyển giới khẩn cấp" cho viêm phổi nhiễm trùng, "chăm sóc hỗ trợ" cho viêm phổi virus, và "quan sát" chỉ cho tình trạng bình thường. Tổng số ảnh X-quang:

Tổng cộng có 5.232 hình ảnh X-quang phần ngực, trong đó bao gồm 3.883 ảnh của người bệnh viêm phổi (2.538 viêm phổi do vi khuẩn và 1.345 viêm phổi do virus) và 1.349 ảnh của người bình thường. Dữ liệu được thu thập và gán nhãn từ 5.856 bệnh nhân để huấn luyện hệ thống trí tuệ nhân tạo (AI).

Về cấu trúc thư mục, bộ dữ liệu được tổ chức thành ba thư mục: train, test, và val. Mỗi thư mục này chứa hai thư mục con liên quan đến 5.863 hình ảnh .jpeg từ bệnh nhân bình thường (nằm trong thư mục con NORMAL) và bệnh nhân mắc bệnh viêm phổi (nằm trong thư mục con PNEUMONIA). Bộ dữ liệu này chủ yếu được sử dụng để huấn luyện mô hình trí tuệ nhân tạo để phân loại giữa các trường hợp viêm phổi và tình trạng bình thường dựa trên hình ảnh X-quang ngực.

5.2 Quá trình tiền xử lý dữ liệu

5.2.1 Phân tách dữ liệu

Dữ liệu đã được chia thành các tập con đào tạo, kiểm tra và xác thực khi được lấy từ API kaggle. Do đó, không cần phân tách train, test thêm nữa và chúng ta có thể tiến hành nhập tập dữ liệu.

Đầu tiên, các tham số về chiều cao và chiều rộng sẽ được xác định (128 x 128) để thay đổi kích thước hình ảnh. Ngoài ra, kích thước lô - là số lượng mẫu sẽ được truyền qua mạng - được xác định là 32.

Sau đo, sử dụng hàm image_dataset_from_directory từ mô-đun xử lý trước của Keras để nhập dữ liệu hình ảnh từ các thư mục đã định nghĩa. Các tập (train_df), (val_df), và (test_df) được

tạo ra, mỗi tập bao gồm các batch chứa hình ảnh và nhãn tương ứng.

```
#Define image sizes
img_height = 128
img_width = 128
batch_size = 32


#import images into dataframes
train_df = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    color_mode = 'grayscale',
    image_size = (img_height,img_width),
    batch_size = batch_size
)

val_df = tf.keras.preprocessing.image_dataset_from_directory(
    val_dir,
    color_mode = 'grayscale',
    image_size = (img_height,img_width),
    batch_size = batch_size
)

test_df = tf.keras.preprocessing.image_dataset_from_directory(
    test_dir,
    color_mode = 'grayscale',
    image_size = (img_height,img_width),
    batch_size = batch_size
)
```

```
Found 5216 files belonging to 2 classes.
Found 16 files belonging to 2 classes.
Found 624 files belonging to 2 classes.
```

Ta thu được chỉ có hai lớp khác nhau được xác định trong các bộ dữ liệu, một cho nhóm bình thường và một cho viêm phổi.

5.2.2 Trực quan hóa dữ liệu

Sau bước tiền xử lý, tất cả hình ảnh có cùng kích thước (128 x 128). Để hiểu rõ hơn về tập dữ liệu, ta thể hiện các thông số trong 3 tập đã cho, nhóm nhận thấy rằng tập dữ liệu không được cân bằng giữa nhãn NORMAL và PNEUMONIA, cụ thể là:

- Tập train chứa 1341 nhãn NORMAL, 3875 nhãn PNEUMONIA.
- Tập valid chứa 234 nhãn NORMAL, 390 nhãn PNEUMONIA.
- Tập test chứa 8 nhãn NORMAL, 8 nhãn PNEUMONIA.

Sự mất cân bằng này có thể gây ra vấn đề khi huấn luyện các mô hình, đặc biệt là khi sử dụng các chỉ số như độ chính xác hoặc diện tích dưới đường cong (AUC) để đánh giá mô hình. Không chỉ vậy, tập dữ liệu huấn luyện lớn hơn đáng kể so với hai tập dữ liệu còn lại.
Sự chênh lệch này có thể ảnh hưởng đến quá trình đào tạo và đánh giá mô hình, và cần được xem xét khi lựa chọn các phương pháp đánh giá hiệu suất và khi xử lý dữ liệu để giảm thiểu ảnh hưởng của mất cân bằng.

5.3 Mô hình mạng nơ-ron Keras[5]

Nhóm sẽ sử dụng mô-đun xử lý trước thử nghiệm của lớp keras như là một giai đoạn đầu tiên của mô hình để chuẩn hóa giá trị đầu vào hình ảnh về một phạm vi chuẩn hóa mới từ 0 (tương đương với ánh sáng tối nhất trong bất kỳ kênh màu RGB nào) đến 1 (tương đương với ánh sáng tối đa trong bất kỳ kênh màu RGB nào).
Mô hình này bao gồm ba khối tích chập với mỗi khối được theo sau bởi một lớp max pool. Sau đó là một lớp fully connected với 128 đơn vị trên đỉnh, được kích hoạt bởi hàm kích hoạt ReLU. Tiếp theo là một lớp Dense khác được kích hoạt bởi hàm Softmax để chuẩn hóa đầu ra của mạng thành một phân phối xác suất trên 2 lớp tiềm năng.

```
1    model = tf.keras.Sequential([
2    layers.experimental.preprocessing.Rescaling(1./255),
3    layers.Conv2D(32, 3, activation='relu'),
4    layers.MaxPooling2D(),
5    layers.Conv2D(32, 3, activation='relu'),
6    layers.MaxPooling2D(),
```

```
    layers.Conv2D(32, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(2, activation='softmax')
])
```

Tiếp đến, sử dụng trình tối ưu hóa Adam (vì nó có thời gian tính toán nhanh hơn - yêu cầu cho tập huấn luyện lớn của chúng ta - và yêu cầu ít tham số điều chỉnh hơn) và một hàm mất mát categorical crossentropy thưa thớt (sparse categorical crossentropy). Hàm này sẽ tạo ra một chỉ số danh mục của nhãn nguyên có khả năng tương ứng cao nhất, và được khuyến nghị khi tập dữ liệu lớn. Chúng ta sẽ sử dụng độ chính xác (accuracy) như một độ đo đánh giá dễ hiểu cho mô hình sơ bộ, mặc dù có thể không phải là phương pháp đúng đắn nhất vì các nhãn không cân bằng trong tập dữ liệu huấn luyện.

```
model.compile(
    optimizer='adam',
    loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)
```

Bước tiếp theo là huấn luyện mô hình với tập dữ liệu huấn luyện của chúng ta, và để thực hiện điều này, chúng ta đã sử dụng hàm model.fit(). Ngoài ra, mô hình được đánh giá sau mỗi epoch trong quá trình huấn luyện bằng cách sử dụng tập dữ liệu xác nhận được cung cấp bởi Kaggle. Tuy nhiên, điều này có thể không phải là lựa chọn tối ưu, vì tập dữ liệu xác nhận thực sự nhỏ so với tập dữ liệu huấn luyện khi chỉ chiếm 0.3% tổng số mẫu. Một sự phân chia 80-10-10 (huấn luyện-xác nhận-kiểm tra) có thể là lựa chọn tốt hơn.

```
epochs = 10
history= model.fit(
                train_df,
                validation_data=val_df,
                epochs=epochs
            )
```

5.4 Mô hình sử dụng PySpark từ thư viện MLlib

5.4.1 Chuẩn bị môi trường và dữ liệu

Nhóm sẽ sử dụng thư viện MLlib của PySpark xây dựng một mô hình có khả năng mở rộng hơn. Để thực hiện điều này, nhóm chọn mô hình Random Forest, sử dụng phương pháp cross-validation để tìm ra các siêu tham số tốt nhất.

Vì ta sẽ thực hiện trên môi trường sử dụng PySpark nên phải nhập dữ liệu vào dataframe của Spark và phân vùng cho mỗi 3 tập dữ liệu (train-validation-test). Sau đó, tạo một hàm định nghĩa Người dùng (UDF) trong Spark để chuyển đổi hình ảnh thành một vector pixel của một kênh duy nhất và kích thước 64x64. Ngoài ra, dữ liệu sẽ được tỷ lệ bằng cách chia mỗi pixel cho 255.

```
from pyspark.ml.linalg import Vectors, VectorUDT
from pyspark.sql.functions import udf

@F.udf(VectorUDT())
def image_to_vector_udf(imagepath: StringType):
  img = cv2.imread(imagepath) #read the image
  img = cv2.resize(img, (32, 32)) #resize
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #change color scale to gray
  img = img/255.0
  img = np.reshape(img, (32,32,1)).flatten().tolist()
  #flat_list = [item for sublist in img for item in sublist]
  return Vectors.dense(img)


# Aplicar la UDF a la columna de imagen
train_df = train_df.withColumn("features", image_to_vector_udf(train_df["path"])
    )
test_df = test_df.withColumn("features", image_to_vector_udf(test_df["path"]))
val_df = val_df.withColumn("features", image_to_vector_udf(val_df["path"]))


list_to_vector_udf = udf(lambda l: Vectors.dense(l), VectorUDT())

train_df = train_df.select(
    train_df["label"],
```

```
    list_to_vector_udf(train_df["features"]).alias("features")
)

test_df = test_df.select(
    test_df["label"],
    list_to_vector_udf(test_df["features"]).alias("features")
)

val_df = val_df.select(
    val_df["label"],
    list_to_vector_udf(val_df["features"]).alias("features")
)
```

5.4.2 Áp dụng Random Forest

Trong báo cáo này, chúng ta đang làm việc với một tập dữ liệu huấn luyện lớn có hơn 5 nghìn hình ảnh, và chỉ định 4 folds cho quá trình cross-validation. Phương pháp cross-validation này sẽ chia tập dữ liệu train thành hai phần: một phần được sử dụng để huấn luyện mô hình và phần còn lại để xác nhận nó. Quá trình này sẽ được thực hiện 4 lần, mỗi lần cho một fold khác nhau. Cách tiếp cận này tốt hơn để đảm bảo mô hình và siêu tham số tốt nhất hơn so với việc sử dụng trực tiếp tập dữ liệu xác nhận, vì nó chỉ chứa 16 mẫu.

Để thực hiện điều này, ParamGridBuilder và CrossValidator sẽ được sử dụng để điều chỉnh mô hình. Sau đó, để đánh giá mô hình và chọn ra mô hình tốt nhất, F1-score sẽ được sử dụng làm độ đo. F1-score là một độ đo hiệu suất dựa trên sự cân bằng giữa precision và recall, và nó thường hoạt động tốt hơn trên các tập dữ liệu mất cân bằng.

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

estimator = RandomForestClassifier(labelCol='label')

evaluator = MulticlassClassificationEvaluator(labelCol='label')

```

```
rfc = estimator
params = ParamGridBuilder() \
    .addGrid(rfc.numTrees, [ 20, 50, 100 ]) \
    .addGrid(rfc.maxDepth, [ 3, 5, 7 ]) \
    .build()

crossval = CrossValidator(estimator=estimator,
                          estimatorParamMaps=params,
                          evaluator=evaluator,
                          numFolds=4)

model_rf = crossval.fit(train_df)
```

5.5 Mô hình BigDL từ thư viện DLlib

Nhóm quyết định sử dụng cùng một kiến trúc CNN (Convolutional Neural Network) mà chúng ta đã sử dụng trong phần triển khai mô hình NN của Keras cho bài toán hiện tại. Do đó, mô hình bao gồm 3 khối tích chập với mỗi khối có một lớp max pool. Tất cả đều được kích hoạt bởi hàm ReLU. Ngoài ra, có một lớp fully connected dense layer với 128 đơn vị ở trên cùng được kích hoạt bởi hàm ReLU, và một lớp dense khác sử dụng hàm kích hoạt Softmax để chuẩn hóa đầu ra của mạng thành một phân phối xác suất qua 2 lớp tiềm năng.

```
model = Sequential()
model.add(Reshape((64, 64, 3), input_shape=(64, 64, 3)))
model.add(Conv2D(32, (3, 3), activation="relu", name="conv1"))
model.add(MaxPooling2D())
model.add(Conv2D(32, (3, 3), activation="relu", name="conv2"))
model.add(MaxPooling2D())
model.add(Conv2D(32, (3, 3), activation="relu", name="conv3"))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(128, activation="relu", name="fc1"))
model.add(Dense(2, activation="softmax", name="fc2"))
```

Tương tự như phần 5.3, chúng ta sẽ sử dụng trình tối ưu hóa Adam, mặc dù đây không phải là tối ưu nhất cho dữ liệu này - như đã được giải thích trong phần trước - nhưng chúng ta sẽ sử dụng độ

chính xác làm độ đo cho tập validation để nhất quán với mô hình CNN trước đó của chúng ta được xây dựng trên Keras.

Ở bước huấn luyện mô hình, nhóm chọn epochs là 30 vì epochs càng lớn sẽ cho ra kết quả càng tốt.

```
model.fit(
    X_train, Y_train,
    epochs=30,
    batch_size=50,
    validation_data=(X_val, Y_val)
)
```

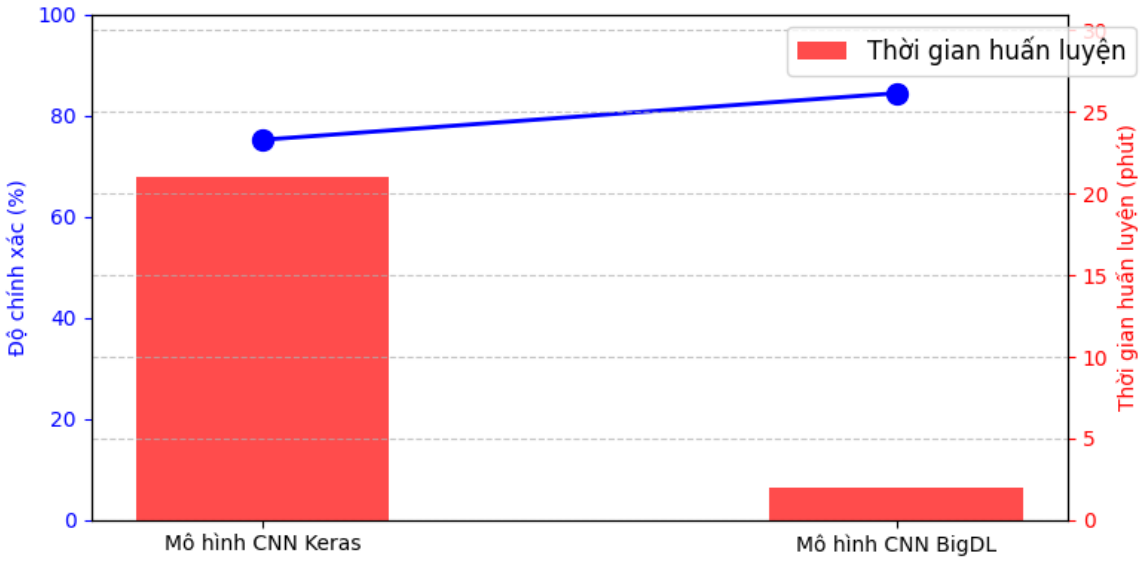
6 Experiment

Với bài toán đặt ra, chúng em đã triển khai 3 mô hình học máy, đó là: sử dụng Mạng thần kinh chuyển đổi (CNN), và mô hình rừng ngẫu nhiên nhằm thực hiện phân loại hình ảnh để dự đoán bệnh viêm phổi.

Thực hiện đánh giá trên tập dữ liệu test gồm 642 mẫu dữ liệu, trong đó có 234 mẫu có nhãn "Normal" và 390 mẫu có nhãn "Preumonia".

Dựa trên kết quả, có thể nhận thấy rằng các nhãn trong tập dữ liệu huấn luyện Kaggle không cân bằng so với các tập dữ liệu khác (1341 trường hợp liên quan đến chụp X-quang bình thường so với 3875 trường hợp hình ảnh mô tả bệnh viêm phổi trong tập dữ liệu huấn luyện). Về so sánh giữa các mẫu (chạy trên Colab host T4 GPU):

- Mô hình CNN của Keras thu được độ chính xác 75,16% và được huấn luyện trong khoảng 21 phút.
- Mô hình CNN được triển khai bằng BigDL đạt độ chính xác 84,38% và hoàn thành quá trình huấn luyện trong thời gian dưới 2 phút.



Hình 13: Đánh giá hai mô hình CNN

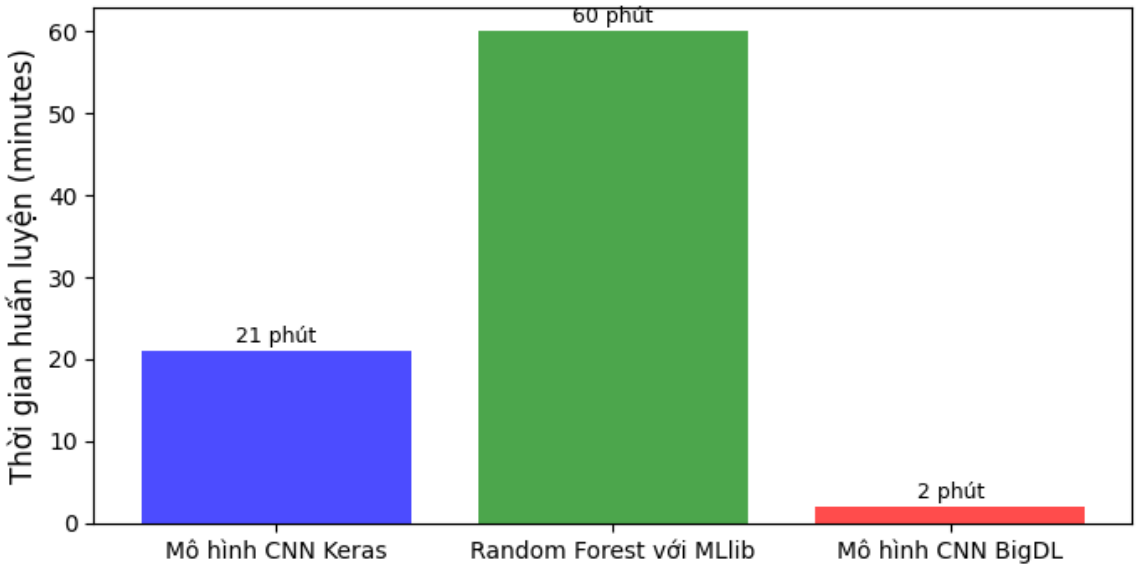
Bởi vì tập dữ liệu nhóm sử dụng không cân bằng nên F1-Score sẽ được sử dụng để đánh giá phân loại đa lớp MLlib. F1-Score có giá trị nằm trong nửa khoảng (0, 1]. Cụ thể, điểm càng cao thì mô hình càng tốt trong khi 0 biểu thị hiệu suất kém nhất.

F1-Score là một phép tính đơn giản liên quan đến việc thu hồi và độ chính xác tổng thể của mô hình và nó có thể được định nghĩa như sau:

$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$

Random Forest được triển khai với MLib đã thu được điểm F1 là 75,85% và đạt được thời gian huấn luyện trong khoảng 1 giờ.

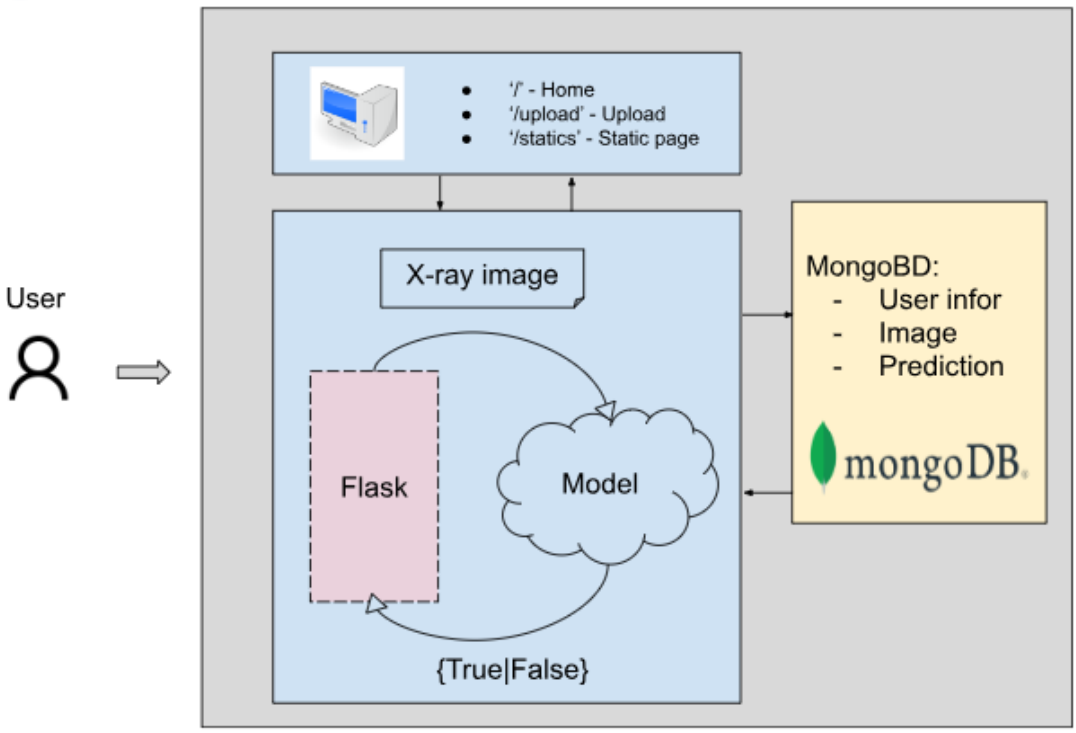
So sánh thời gian huấn luyện cho 3 mô hình:



Hình 14: Biểu đồ so sánh thời gian huấn luyện

7 Application and user interface

Để giúp người dùng thao tác dễ dàng, phục vụ cho mục đích y tế chẩn đoán bệnh, nhóm em đã xây dựng giao diện cho phép tải lên một hình ảnh X-Ray cùng thông tin bệnh nhân tương ứng và đưa ra dự đoán bệnh đối với ảnh đó, sau đó thực hiện lưu trữ thông tin bệnh vào MongoDB. 15



Hình 15: Sơ đồ API

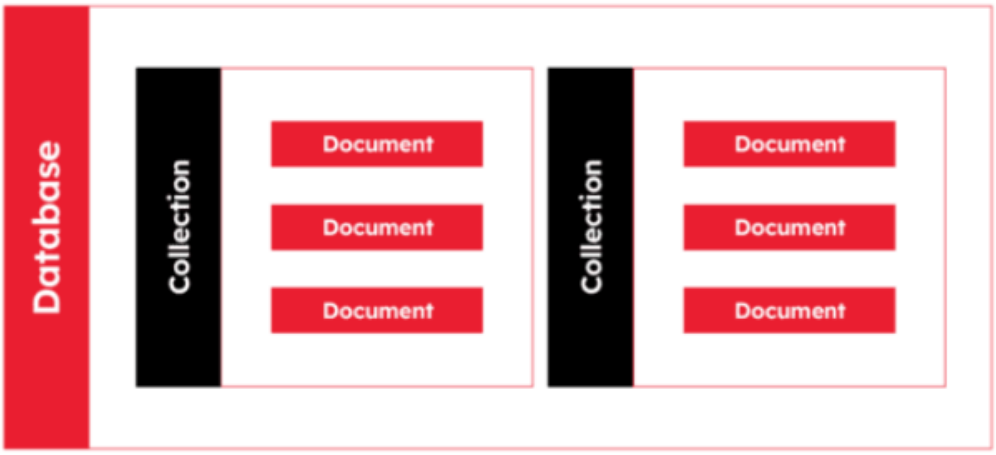
7.1 Cơ sở lý thuyết

7.1.1 Lưu trữ dữ liệu sử dụng MongoDB

Nhóm quyết định chọn MongoDB để lưu trữ dữ liệu bởi vì việc ghi chú lại bệnh án của bệnh nhân rất cần thiết để theo dõi liệu trình cho một người và đồng thời cũng phục vụ cho công việc nghiên cứu khảo sát bệnh của người dân trong khu vực. Bên cạnh đó, dữ liệu người dùng nhập vào của một bệnh viện tăng theo thời gian nên cần được lưu trữ chúng trên một cơ sở dữ liệu "khổng lồ" như MongoDB. Điểm lợi thế khi sử dụng MongoDB là dễ dàng tích hợp một lượng lớn dữ liệu đa dạng, cung cấp dữ liệu cho các ứng dụng hiệu suất cao, hỗ trợ các ứng dụng đám mây lai và đa đám mây.

MongoDB là một máy chủ cơ sở dữ liệu và dữ liệu được lưu trữ trong các cơ sở dữ liệu này. Thay vi cac ban ghi, trương dư liêu như trong SQL, MongoDB sư dung lưu trư dư liêu dươi dang JSON nên tư đo môi Collection se sơ hưu nhưng kich thươc va cac document riêng. Vi vây, no co thê sư dung lưu trư cac dư liêu lơn co kich thươc va đô phưc tap đa dang (Big Data). Thay vì sử dụng các table và row như trong cơ sở dữ liệu quan hệ, vì là cơ sở dữ liệu NoSQL, MongoDB được tạo thành từ collection và document. Document được tạo thành từ các cặp khóa và giá trị (là đơn vị

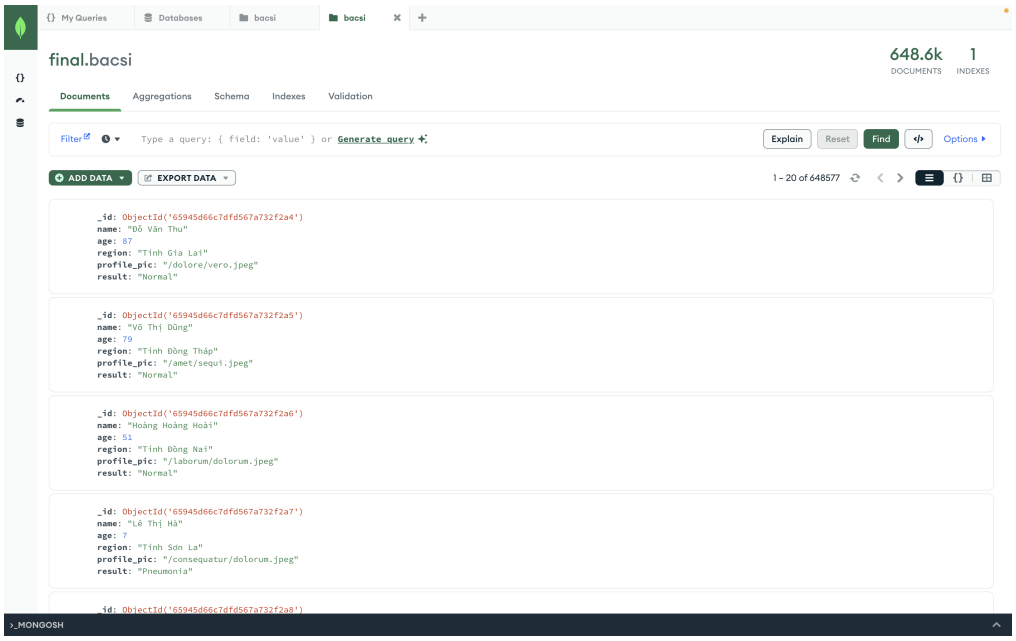
dữ liệu cơ bản của MongoDB). Còn collection, tương đương với table trong SQL, là nơi chứa các bộ document. Mối liên hệ giữa các document với các collection trong một cơ sở dữ liệu được thể hiện qua hình 16



Hình 16: Cấu trúc MongoDB

Trong ứng dụng giao diện dự đoán bệnh viêm phổi, nhóm đã tạo một cơ sở dữ liệu tên **final**, cơ sở dữ liệu này có chứa một collection **bacsi** chứa các document, được tạo bằng cách sử dụng các field (trường). Các field là các cặp khóa-giá trị trong document, giống như các column trong cơ sở dữ liệu quan hệ. Giá trị của các field có thể là bất kỳ loại dữ liệu BSON nào như double, string, boolean, v.v. Ở đây, mỗi document đại diện cho một bệnh nhân có các thông tin:

- Tên
- Tuổi
- Địa chỉ
- Ảnh X-quang
- Dự đoán đưa ra tương ứng (NORMAL/PREUNOMIA)



Hình 17: Giao diện của MongoDB chứa dữ liệu

MongoDB hoạt động với 2 layer: Layer ứng dụng và layer dữ liệu. Tại layer ứng dụng, người dùng thao tác bằng cách nhập tên, tuổi, địa chỉ ở phần bên trái, tại ô bên phải ta sẽ tải hình ảnh lên tại đây. Lúc này phần back-end chứa một máy chủ được sử dụng để thực hiện logic phía máy chủ và cũng chứa trình điều khiển hoặc MongoDB shell để tương tác với máy chủ MongoDB với sự trợ giúp của truy vấn

Sau đó, các truy vấn này được gửi đến máy chủ MongoDB thuộc Layer Dữ liệu. Bây giờ, máy chủ MongoDB nhận các truy vấn và chuyển các truy vấn đã nhận tới công cụ lưu trữ vì bản thân máy chủ MongoDB không trực tiếp đọc hoặc ghi dữ liệu vào tệp hoặc đĩa hoặc bộ nhớ. Sau khi chuyển các truy vấn nhận được tới bộ máy lưu trữ, bộ máy lưu trữ chịu trách nhiệm đọc hoặc ghi dữ liệu trong tệp hoặc bộ nhớ.

7.1.2 Kết nối FE với BE và cơ sở dữ liệu (sử dụng FLask)

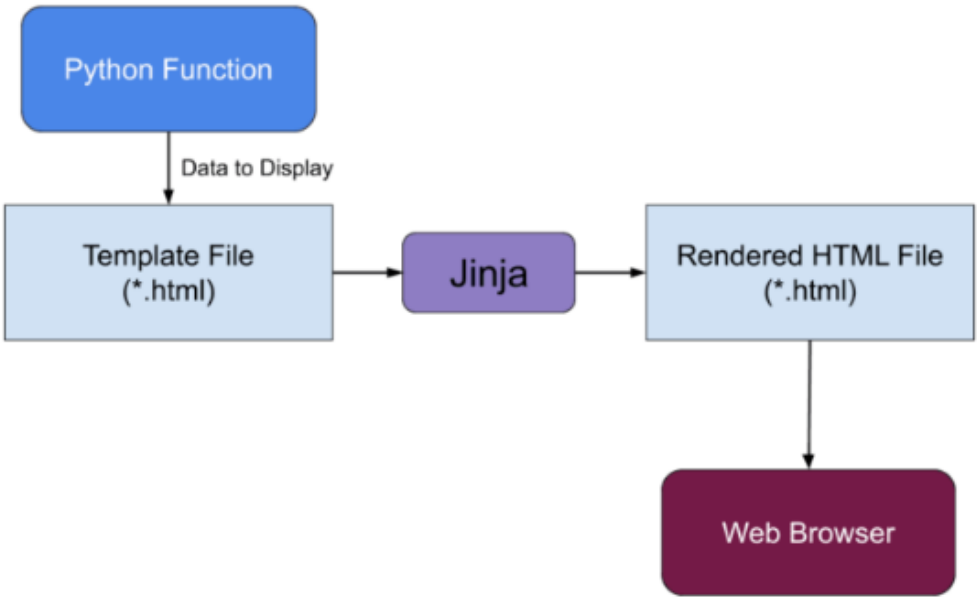
Flask là một micro web framework được viết bằng Python, không yêu cầu tool hay thư viện cụ thể nào. “Micro” không có nghĩa là thiếu chức năng mà “micro” theo triết lý thiết kế là cung cấp một lõi chức năng “súc tích” nhất cho ứng dụng web nhưng người dùng có thể mở rộng bất cứ lúc nào. Flask luôn hỗ trợ các thành phần tiện ích mở rộng cho ứng dụng như tích hợp cơ sở dữ liệu, xác thực biểu mẫu, xử lý upload, các công nghệ xác thực, template, email, RESTful..., chỉ khác là khi nào bạn muốn thì bạn mới đưa vào thôi. Người dùng có thể tập trung xây dựng web application

ngay từ đầu trong một khoảng thời gian rất ngắn và có thể phát triển quy mô của ứng dụng tùy theo yêu cầu.
Ưu điểm của Flask:

- Dễ kết nối: Sau khi cài đặt Python, để cài đặt Flask chỉ cần bạn gõ lệnh: 'pip install Flask'
- Scalable – Mở rộng: Chính nguyên lý và áp dụng microframework đã giúp cho Flask Python có thể dễ dàng mở rộng nếu có nhu cầu từ Web Application. Do phần core chạy độc lập và ít phụ thuộc, nên cho dù mở rộng ở mức quy mô thì web application sử dụng Flask Python vẫn đáp ứng được.
- Flexible – Linh hoạt: Sự linh hoạt là tính năng cốt lõi và cũng là ưu điểm của Flask Python. Chính vì giữ cho core và các thành phần khác đơn giản nên ít bị phụ thuộc vào nhau. Sự đơn giản này giúp cho anh em có thể chuyển hướng web application theo business owner dễ dàng hơn.
- Easy to negotiate – Dễ trao đổi: Về mặt cốt lõi thì phần core của Flask Python rất dễ hiểu với anh em developers. Phần triển khai cũng vô cùng nhanh chóng. Anh em chỉ cần tập trung viết code, viết cho chính xác cái thứ mà web application mình cần làm là được.

Nhược điểm của Flask: Nhược điểm của Flask Python cũng bắt đầu từ chính ưu điểm của nó. Do phần core được giữ cho thật sự đơn giản nên trong quá trình phát triển nếu cần gì thêm sẽ phải cài riêng. Web Application cần nhiều thì sẽ tốn thời gian để cài cho đủ các thành phần.

1. Front-end: Nói là độc lập nhưng Flask Python vẫn có phụ thuộc vào 2 plugins là:
    - Werkzeug là WSGI utility library: WSGI về cơ bản là một giao thức giúp web application python có thể dễ dàng giao tiếp với server.
    - jinja2 là template engine: Template là một layout được thiết kế các khung web có sẵn, ta chỉ cần thêm nội dung chính của nó vào, và nhờ các template ta mới tiết kiệm thời gian trong việc phát triển website. Jinja2 là một ngôn ngữ tạo template cung cấp cho các lập trình viên Python, được tạo ra dựa trên ý tưởng của Django template. Jinja2 được sử dụng để tạo HTML, XML hoặc các định dạng file khác dựa trên nguyên tắc kết hợp các dữ liệu vào các vị trí đã được đánh dấu trong văn bản.



Hình 18: Cấu truc Front-end

2. Back-end:

   Để kết nối cơ sở dữ liệu, nhóm em sử dụng phần mở rộng Flask của MongoDB: MongoEngine. Có thể sử dụng MongoEngine một cách độc lập mà không cần dựa vào Flask nhưng có thể sử dụng kết hợp với Flask.

   Để sử dụng MongoEngine trong Flask, trước tiên chúng ta cần định cấu hình thông tin của MongoDB trong Flask trước khi khởi tạo MongoEngine với máy chủ của mình để kết nối cơ sở dữ liệu và máy chủ, có thể nói bằng mã:

   Cài đặt : pip install flask_mongoengine

```
from flask_mongoengine import MongoEngine
app.config['MONGODB_SETTINGS'] = {
    'db': 'final',
    'host':     localhost    ,
      port  ': 27017
}
db = MongoEngine()
db.init_app(app)
```

   Sau khi định cấu hình thông tin mongodb, tạo mô hình dữ liệu bằng MongoEngine.

```
class bacsi(db.Document):
```

```
    name = db.StringField()
    age = db.StringField()
    region = db.StringField()
    profile_pic = db.StringField()
    result = db.StringField()
```

3. Vẽ biểu đồ:

   Matplotlib là một thư viện vẽ biểu đồ 2D chung cho ngôn ngữ lập trình Python. Nó cung cấp các công cụ mạnh mẽ để tạo ra các loại biểu đồ phong phú như đường, cột, biểu đồ tròn và nhiều loại biểu đồ khác. Matplotlib hoạt động trên khái niệm của các đối tượng giống như các đối tượng trong MATLAB, nơi mỗi yếu tố của biểu đồ được kiểm soát bởi đối tượng. Mô hình đối tượng trong Matplotlib bao gồm các thành phần chính như Figure, Axes, và các yếu tố như Line, Bar, Pie, v.v. Các hàm trong Matplotlib cho phép tùy chỉnh nhiều khía cạnh của biểu đồ, bao gồm cả kích thước, màu sắc, kiểu đường, v.v.

7.2 Triển khai giao diện

7.2.1 Xử lý ảnh đầu vào

Đầu tiên ta xử lý ảnh đầu vào, đưa ảnh về kích thước mong muốn là (128 x 128), sau đó tải mô hình được train trước (bidl.h5) bằng hàm load_model của Keras. Cần một bước tiền xử lý hình ảnh, ta sẽ đọc hình ảnh bằng OpenCV, sau đó chuyển đổi hình ảnh từ định dạng BGR sang RGB. Hình ảnh được thay đổi kích thước để phù hợp với kích thước đã chỉ định (64, 64) bằng cv2.resize. Tiếp đến chuẩn hóa các giá trị pixel thành phạm vi [0, 1]. Hình ảnh được định hình lại để phù hợp với hình dạng đầu vào mong đợi của mô hình ((1, 64, 64, 3)). Cuối cùng trả về hình ảnh được xử lý trước hoặc None nếu có vấn đề với việc đọc hoặc thay đổi kích thước hình ảnh.

7.2.2 Dựng API

1. API Home:

   Endpoint: /

   Method: GET và POST

   Chức năng: Trả về trang chính (home) của ứng dụng web hoặc xử lý yêu cầu POST nếu có.

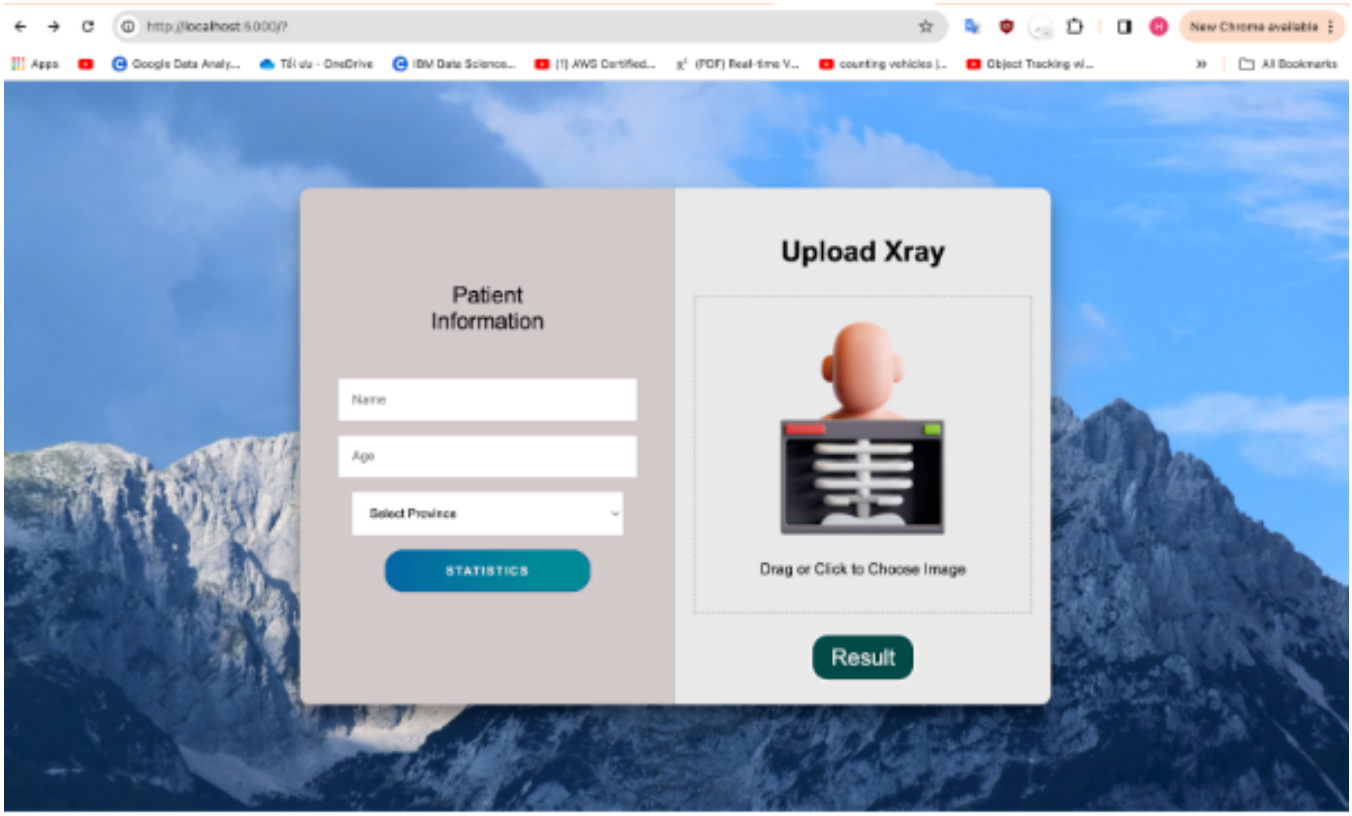
Nếu có yêu cầu POST, nó kiểm tra xem có tệp hình ảnh được tải lên không và thực hiện xử lý trước khi chuyển hướng đến trang kết quả.

2. API Upload File:
   Endpoint: /upload
   Method: GET và POST
   Chức năng: Xử lý yêu cầu tải lên hình ảnh. Nếu có yêu cầu POST, nó kiểm tra xem tệp hình ảnh đã được tải lên hay chưa, sau đó thực hiện tiền xử lý và dự đoán kết quả bằng mô hình đã được đào tạo. Sau đó, lưu thông tin người dùng và kết quả vào MongoDB trước khi hiển thị kết quả trên trang web.

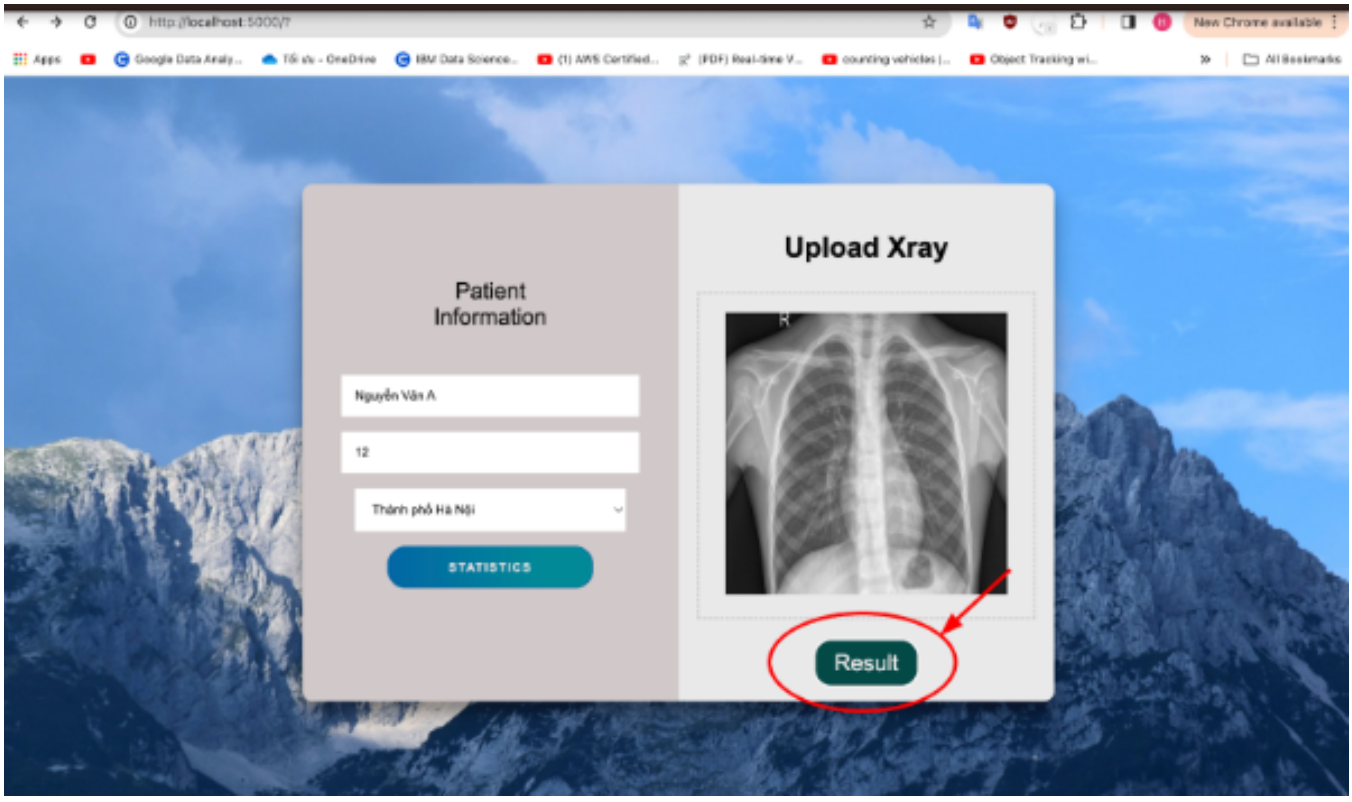
3. API Statistics:
   Endpoint: /statistics
   Method: GET
   Chức năng: Truy xuất dữ liệu từ MongoDB, thống kê và tổng hợp thông tin. Tạo biểu đồ và biểu đồ tròn thể hiện thông tin thống kê về số lượng trường hợp bệnh nhân theo khu vực và nhóm tuổi. Kết quả được nhúng vào trang web và hiển thị thông qua biểu đồ cột ghép và biểu đồ tròn.

7.3 Giao diện và ứng dụng

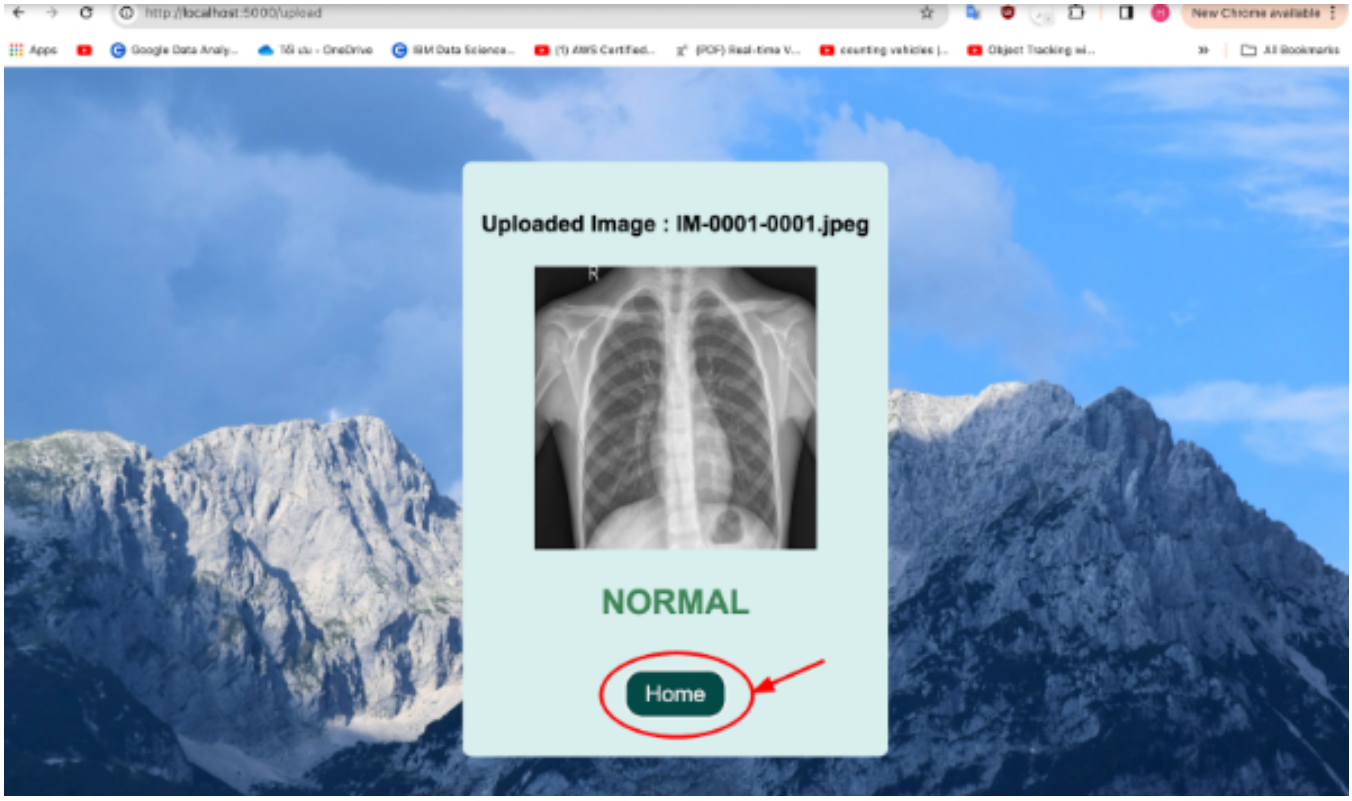
Giao diện cho phép người dùng nhập vào họ tên, tuổi của bệnh nhân và tải lên ảnh X-ray tương ứng. Tiếp theo, bấm vào nút "Submit", kết quả hiện ra trên màn hình là dự đoán "normal" (bệnh nhân được mô hình dự đoán không bệnh) hoặc "pneumonia" (bệnh nhân được mô hình dự đoán có bệnh viêm phổi). Sau khi hoàn tất tải lên và cho ra dự đoán thành công, dữ liệu về tên, tuổi, dự đoán bệnh, và hình ảnh tương ứng sẽ được đưa vào một phần mềm cơ sở dữ liệu mã nguồn mở NoSQL, được thiết kế hướng theo đối tượng và hỗ trợ trên đa nền tảng MongoDB.
Trang chủ của giao diện:



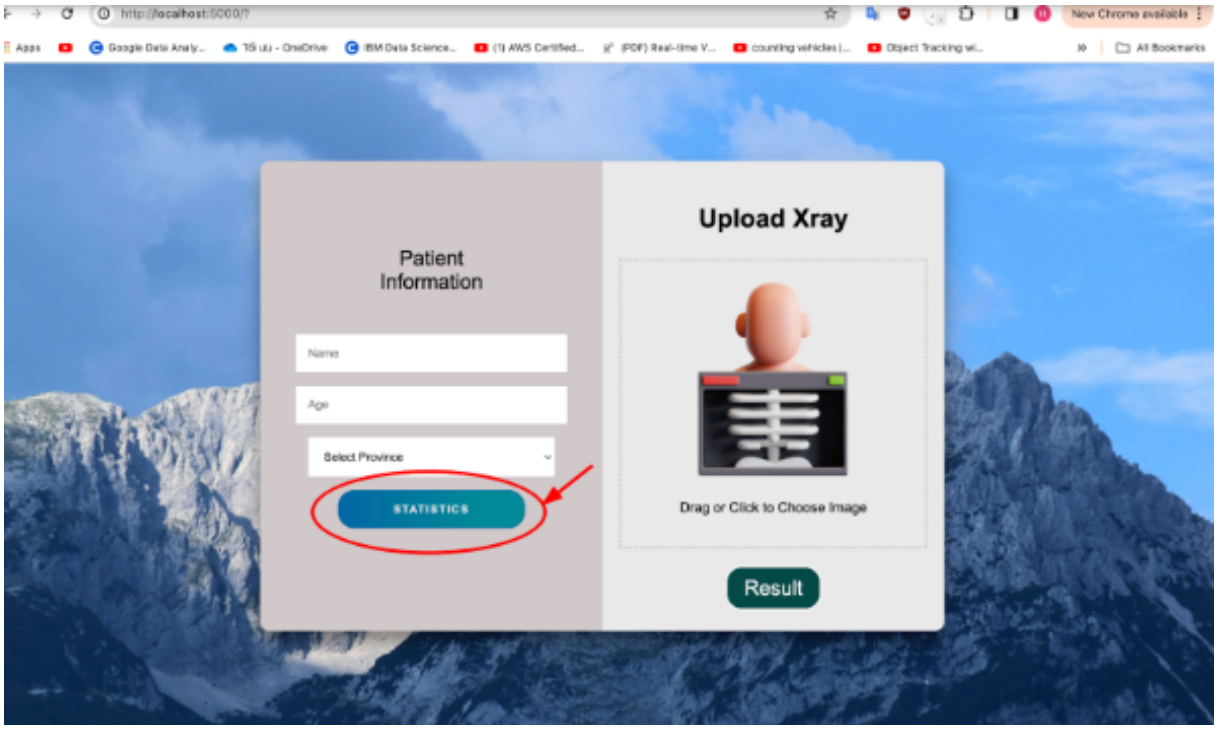
Sau khi nhập thông tin và upload ảnh:



Hiện kết quả normal / pneumonia và ấn Home để trả về trang chủ:

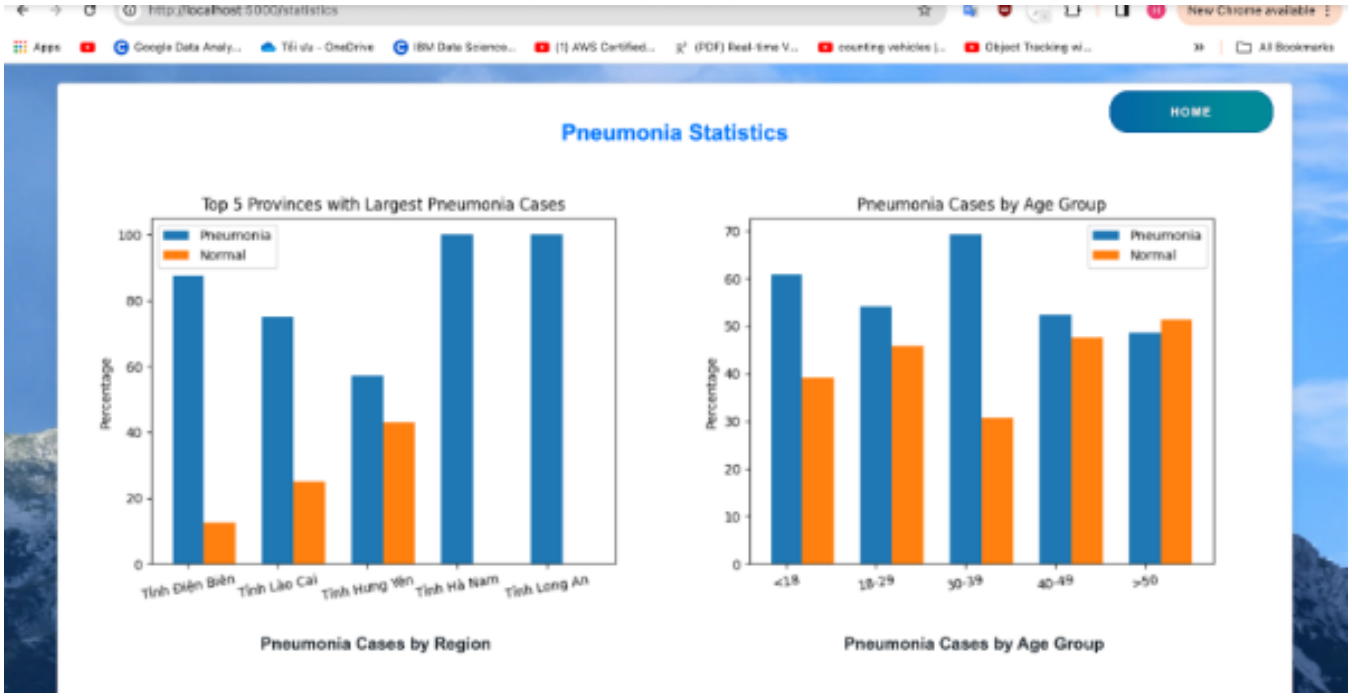


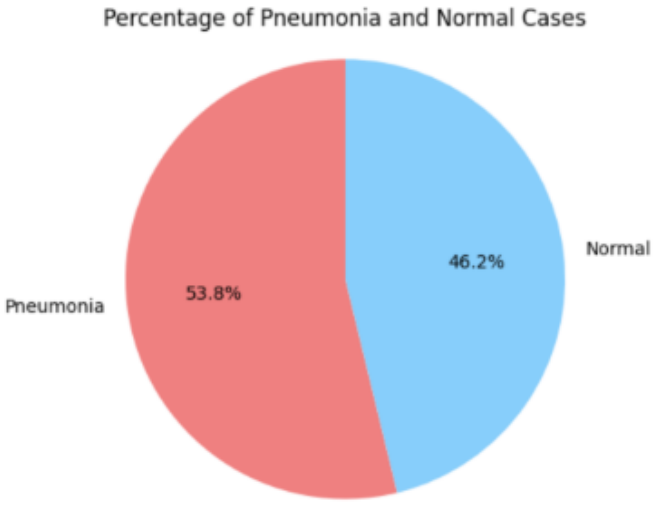
Xem số liệu thống kê:



Biểu đồ: gồm 3 biểu đồ thống kê:

- Top 5 tỉnh bị nhiễm bệnh cao nhất.
- Thống kê tỷ lệ nhiễm bệnh theo từng nhóm tuổi.
- Thống kê tỷ lệ chênh lệch giữa nhiễm bệnh và không nhiễm trên tổng số liệu.





Percentage of Pneumonia and Normal Cases

8 Conclusion

Đọc ảnh y khoa từ lâu đã trở thành một phần quan trọng của chăm sóc y tế và đã phát triển cùng với các tiến bộ gần đây trong các phương pháp quét. Những kỹ thuật hình ảnh này hiện giờ cho phép nhận biết những thay đổi dù là nhỏ nhất trong các biến vật lý và có thể hỗ trợ các chuyên gia y tế có kiến thức và người thực hành gặp khó khăn trong các nhiệm vụ nhận diện này thông qua việc sử dụng Trí tuệ Nhân tạo. Tuy nhiên, những công nghệ Trí tuệ Nhân tạo này vẫn đang được cải thiện, và sự chuyển đổi sang các công cụ Trí tuệ Nhân tạo đang là xu thế bởi sự tối ưu và hỗ trợ mạnh mẽ của nó.

Qua báo cáo đề tài nhận diện bệnh viêm phổi từ ảnh chụp X-Ray dựa trên mô hình phân loại hình ảnh dựa trên Spark, nhóm chúng em mong muốn thực hiện xây dựng mô hình học máy có khả năng đưa ra dự đoán tối ưu về mặt thời gian thực thi chương trình, tiết kiệm chi phí tính toán và có độ chính xác cao.

Qua thí nghiệm so sánh trên tập dữ liệu gồm hơn 5000 ảnh, mô hình CNN sử dụng BigDL cho ra kết quả dự đoán đúng nhất và thời gian chạy nhanh gấp 10 lần mô hình CNN sử dụng Keras và nhanh gấp 30 lần mô hình Random Forest sử dụng thư viện MLlib.

Vấn đề lưu trữ lịch sử dự đoán bệnh án của các bệnh nhân cũng được giải quyết nhờ MongoDB, nhóm đã thực hiện lưu trữ được hơn 650.000 mẫu dữ liệu trên MongoDB, từ đó vấn đề cập nhật khảo sát bệnh trên các tỉnh thành cũng được đảm bảo.

Tài liệu

[1] Mehdi Assefi, Ehsun Behravesh, Guangchi Liu, and Ahmad P. Tafti. Big data machine learning using apache spark mllib. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 3492–3498, 2017.

[2] Jason Jinquan Dai, Yiheng Wang, Xin Qiu, Ding Ding, Yao Zhang, Yanzhang Wang, Xianyan Jia, Cherry Li Zhang, Yan Wan, Zhichao Li, Jiao Wang, Shengsheng Huang, Zhongyuan Wu, Yang Wang, Yuhao Yang, Bowen She, Dongjie Shi, Qiaoling Lu, Kai Huang, and Guoqiong Song. Bigdl: A distributed deep learning framework for big data. *Proceedings of the ACM Symposium on Cloud Computing*, 2018.

[3] A. S. More and Dipti P. Rana. Review of random forest classification techniques to resolve data imbalance. In *2017 1st International Conference on Intelligent Systems and Information Management (ICISIM)*, pages 72–78, 2017.

[4] Arundhati Navada, Aamir Nizam Ansari, Siddharth Patil, and Balwant A. Sonkamble. Overview of use of decision tree algorithms in machine learning. In *2011 IEEE Control and System Graduate Research Colloquium*, pages 37–42, 2011.

[5] Shanay Shah, Heeket Mehta, and Pankaj Sonawane. Pneumonia detection using convolutional neural networks. In *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 933–939, 2020.

[6] Shuhan Tan, Boris Ivanovic, Xinshuo Weng, Marco Pavone, and Philipp Kraehenbuehl. Language conditioned traffic generation, 2023.

A Phụ lục

- Trong dự án bài tập lớn, nhóm thực hiện đánh giá mô hình trên Colab host T4 GPU.
- Source code được public trên GitHub