

**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC BÁCH KHOA**  
**KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH**

—o0o—



**Luận văn tốt nghiệp**

## **Phát hiện kẹt xe từ camera hành trình**

**GVHD:** TS.Dương Ngọc Hiếu

**Nhóm sinh viên thực hiện:**

Trương Ngọc Anh    1410141

Lê Nguyễn Minh Trí    1414207

# Mục lục

<b>1</b>	<b>Tổng quan về hệ thống phân loại ảnh giao thông kết hợp với lưu trữ dữ liệu lớn</b>	<b>12</b>
1.1	Giới thiệu hệ thống phân loại ảnh giao thông kết hợp với dữ liệu lớn . . . . .	13
1.1.1	Các công trình nghiên cứu . . . . .	13
1.2	Kết luận chương . . . . .	14
<b>2</b>	<b>Mạng neural và Mạng neural tích chập (Convolutional Neural Networks CNN</b>	<b>15</b>
2.1	Mạng neural (Neural Network) . . . . .	15
2.1.1	Mạng neural ( Neural Network ) . . . . .	15
2.1.2	Cấu trúc mạng neural . . . . .	16
2.1.3	Mô hình Feedforward Neural Network . . . . .	20
2.2	Mạng neural tích chập (Convolutional neural network) . . . . .	24
2.2.1	Giới thiệu mạng neural tích chập . . . . .	24
2.2.2	Mô hình mạng neural tích chập . . . . .	24
<b>3</b>	<b>Apache Hadoop</b>	<b>32</b>

3.1	Tổng quan về hệ thống Apache Hadoop . . . . .	32
3.2	Hadoop distributed file system - HDFS . . . . .	33
3.2.1	Thiết kế . . . . .	33
3.2.2	Các khái niệm trong HDFS . . . . .	33
3.2.3	HIVE . . . . .	35
3.3	Tại sao sử dụng Apache Hadoop trong đề tài . . . . .	37
<b>4</b>	<b>Phương pháp giải quyết</b>	<b>38</b>
4.1	Phân tích . . . . .	38
4.1.1	Mô tả tập dữ liệu . . . . .	38
4.1.2	Các vấn đề về lưu trữ . . . . .	39
4.2	Giải pháp đề xuất . . . . .	40
4.2.1	Kiến trúc mạng googleNet . . . . .	40
4.2.2	Cấu trúc hệ thống lưu trữ . . . . .	43
<b>5</b>	<b>Hiện thực mô hình loại ảnh giao thông kết hợp Apache Hadoop</b>	<b>44</b>
5.1	Tổng quan các bước . . . . .	44
5.2	Các bước chuẩn bị . . . . .	45
5.2.1	Chuẩn bị môi trường hiện thực . . . . .	45
5.3	Các bước hiện thực . . . . .	47
5.3.1	Cài đặt Apache Hadoop . . . . .	47
5.3.2	Cài đặt HIVE . . . . .	56
5.3.3	Xử lý dữ liệu . . . . .	58
5.3.4	Tạo các bottlenecks . . . . .	59
5.3.5	Huấn luyện . . . . .	59

5.3.6	Chạy Huấn luyện bằng cách kết hợp tensorflow với hệ thống HDFS . . . . .	60
<b>6</b>	<b>Kết quả huấn luyện và đánh giá</b>	<b>61</b>
6.1	Kết quả huấn luyện . . . . .	61
6.2	Sử dụng mô hình phân loại ảnh mới . . . . .	63
<b>7</b>	<b>Kết luận</b>	<b>69</b>
7.1	Các kết quả . . . . .	69
7.2	Hướng phát triển . . . . .	70

# Danh sách hình vẽ

2.1	Tế bào thần kinh neuron sinh học [1]	16
2.2	Cấu trúc cơ bản mạng neuron[2]	17
2.3	Đồ thị hàm step	19
2.4	Đồ thị hàm sigmoid	19
2.5	Đồ thị hàm Tanh	20
2.6	Ví dụ kiến trúc mạng feedforward [3]	21
2.7	MLP	22
2.8	Convolutional layer [4]	26
2.9	Convolutional layer [4]	26
2.10	Mảng các giá trị của bộ lọc [4]	27
2.11	ảnh đầu vào [5]	28
2.12	phép toán tích chập [5]	28
2.13	phép toán tích chập [5]	29
2.14	kết quả tầng tích chập [5]	29
2.15	max-pooling 2 x 2 [6]	30
2.16	Ví dụ tầng tổng hợp [4]	31
4.1	Ảnh mẫu trong phân lớp thông thoáng	39
4.2	Ảnh mẫu trong phân lớp ùn tắc	39

4.3	GoogLeNet [7]	41
4.4	inception module [7]	42
4.5	naive inception module [7]	42
5.1	Mô hình phân loại ảnh giao thông kết hợp Hadoop	45
5.2	Kiểm tra Hadoop service trên master	54
5.3	Kiểm tra Hadoop service trên worker(slave)	54
5.4	Kiểm tra Hadoop service trên worker(slave1	55
5.5	Liệt kê thư mục dữ liệu trên HDFS	56
5.6	Nội dung thư viện LD_LIBRARY_PATH	60
5.7	Câu lệnh thực thi huấn luyện mô hình	60
6.1	Đồ thị biểu diễn độ chính xác (accuracy)	62
6.2	Đồ thị biểu diễn độ sai số (cross entropy)	63
6.3	Ảnh kiểm thử 1 - kết quả	64
6.4	Ảnh kiểm thử 2 - kết quả	65
6.5	Ảnh kiểm thử 3 - kết quả	66
6.6	Ảnh kiểm thử 4 - kết quả	67

# Danh sách bảng

5.1	Các chiều dữ liệu. . . . .	46
5.2	Một vài kiểu dữ liệu. . . . .	47

# Lời cảm ơn

Trước khi vào phần giới thiệu nội dung đề tài luận văn, nhóm sinh viên tụi em xin gửi lời cảm ơn chân thành và sự tri ân sâu sắc đối với thầy **Dương Ngọc Hiếu** đã giúp đỡ, hướng dẫn tận tình cho tụi em về những khái niệm căn bản và phương hướng tiếp cận thông qua việc phân tích ảnh được dùng trên công nghệ Deep Learning, giúp tụi em chỉnh sửa những thiếu sót trong quá trình nghiên cứu và thực hiện đề tài này. Ngoài ra, tụi em cũng xin chân thành cảm ơn anh **Nguyễn Thanh Trông** đã hướng dẫn, giúp đỡ tụi em về việc ứng dụng & kết hợp Deep Learning với Big Data, bổ sung những thiếu sót, sai sót của tụi em trong quá trình thực hiện đề tài. Đồng thời, do những kinh nghiệm thực tiễn và phương hướng tiếp cận về công nghệ Deep Learning & Big Data còn nhiều hạn chế nên bài luận văn không thể tránh khỏi những thiếu sót, tụi em rất mong sẽ nhận được ý kiến đóng góp từ phía Thầy, Cô để tụi em học hỏi thêm được nhiều kinh nghiệm và sẽ hoàn thành tốt đề tài luận văn này hơn nữa.

Tụi em xin chân thành cảm ơn !



# Lời mở đầu

## Vấn đề

Từ những năm gần đây, tình trạng ùn tắc giao thông ở Tp.Hồ Chí Minh đã không dừng lại ở diễn biến phức tạp mà lại còn gia tăng hơn trước. Cụ thể, trên các tuyến đường hiện nay, tình trạng kẹt xe không chỉ xảy ra ở giờ cao điểm mà còn ở các khung giờ khác. Nguyên nhân dẫn đến sự việc trên, một phần ảnh hưởng bởi điều kiện thời tiết, một phần do các công trình cải tạo hạ tầng, khi thi công lấn chiếm mặt đường. Nhưng phần lớn là do mật độ xe cộ ngày một đông dần, dẫn đến việc ùn ứ, ùn tắc, di chuyển chậm,... Từ những nguyên nhân đó, nhóm sinh viên chúng em đã xây dựng mô hình hệ thống phát hiện kẹt xe từ camera hành trình (cụ thể là camera hành trình xe buýt), với mong muốn có thể góp phần giải quyết được một phần nhỏ tình trạng giao thông hiện nay ở Tp. Hồ Chí Minh nói riêng cũng như ở Việt Nam nói chung.

## Động lực nghiên cứu và tính cấp thiết của đề tài

Trong bối cảnh mà các công nghệ xử lý trở phát triển mạnh như hiện nay. Chúng ta có thể áp dụng các công nghệ học máy và deep learning trong nhiều lĩnh vực khác nhau từ y tế, nông nghiệp, kinh tế tài chính... Kèm theo đó là lượng dữ liệu tăng đột biến. Sẽ thật tốt nếu có một hệ thống vừa có thể lưu trữ dữ liệu lớn mà vừa có khả năng áp dụng công nghệ học máy, học sâu nói trên để hỗ trợ trong việc xử lý các vấn đề un tắt giao thông.

Khái niệm dữ liệu lớn, học máy, học sâu nhiều năm gần đây. Khi chính công ty lớn điển hình là Google, hay Tesla cũng đang phát triển tạo ra các hệ thống hỗ trợ giao thông thông minh. Các nước phát triển với các đặc tính dân số đông, mật độ giao thông phức tạp như Trung Quốc cũng đã ráo riết xây dựng các hệ thống camera giám sát kết hợp với học sâu để tiến hành nhận diện ảnh giao thông..v.v.

Đối với trong nước, hiện nay nhà nước cũng đã bắt đầu tiến hành đầu tư các hạng mục giao thông, thực hiện việc trang bị các camera giám sát tại một số tuyến đường trọng tâm nhằm để theo dõi tình trạng giao thông. Nhưng để áp dụng lĩnh vực học sâu hay thậm chí là xử lý ảnh vào để sử dụng thì vẫn còn khan hiếm.

Như vậy, một hệ thống có thể lưu trữ kết hợp với lĩnh vực học sâu là thật sự cần thiết. Không phải chỉ cần thiết cho lĩnh vực giao thông, đây có thể cân nhắc để trở thành một framework để ứng dụng để kết hợp các lĩnh vực khác nhau trong cuộc sống. Đây xứng đáng là một đề nghiên cứu và phát triển. Với sự ra đời các mạng học sâu và các framework lưu trữ dữ liệu

lớn như Apache Hadoop sẽ hỗ trợ cho việc nghiên cứu và phát triển hệ thống một cách dễ tiếp cận hơn. Chúng em đã nghiên cứu và lựa chọn ra mô hình mạng thích hợp cũng như xây dựng hệ thống mô phỏng lưu trữ dữ liệu lớn mà cho rằng có khả năng áp dụng vào thực tiễn.

## Mục tiêu luận văn

Với các cơ sở thực tiễn trên, luận văn này đặt ra các mục tiêu như sau:

- Lựa chọn, áp dụng mô hình mạng phân loại ảnh giao thông.
- Xây dựng hệ thống mô phỏng lưu trữ dữ liệu lớn.
- Kết hợp mạng học sâu với hệ thống lưu trữ dữ liệu lớn.

## Tính khả quan

Để đánh giá về tính khả quan trong đề tài này, chúng ta cần phân tích một số đặc điểm. Thứ nhất là về tập ảnh giao thông, tập ảnh giao thông tốt hay không tùy thuộc vào nhiều yếu tố như nguồn gốc tập ảnh hay cơ cấu hạ tầng camera giao thông được đầu tư như thế nào. Đối với thành phố Hồ Chí Minh nói riêng, chúng ta vẫn còn đang xây dựng hệ thống cơ sở hạ tầng giám sát nên chưa thể dựa vào nguồn camera giám sát để lấy dữ liệu. Vì thế, chúng em sẽ sử dụng tập ảnh từ camera hành trình trên các xe buýt để thực hiện xây dựng mô hình. Khi đã có tập dữ liệu đủ tốt, chúng ta có thể sử dụng mô hình được xây dựng trên đề tài này và có thể được áp dụng vào hệ thống thực tiễn.

## Cấu trúc luận văn

- **Chương 1: Tổng quan về hệ thống phân loại ảnh giao thông kết hợp với lưu trữ dữ liệu lớn.** Giới thiệu sơ lược về vấn đề đặt ra trong luận văn, các nghiên cứu đã có trên thế giới và hướng tập trung nghiên cứu.
- **Chương 2: Mạng neural và Mạng neural tích chập (Convolutional Neural Networks CNN.** Cơ sở lý thuyết về mạng học sâu.
- **Chương 3: Apache Hadoop.** Tổng quát framework lưu trữ dữ liệu lớn Apache Hadoop.
- **Chương 4: Phương pháp giải quyết.** Đề xuất giải pháp cho đề tài.
- **Chương 5: Hiện thực mô hình loại ảnh giao thông kết hợp Apache Hadoop.** Hiện thực giải pháp đã chọn cho hệ thống phân loại. Bao gồm nội dung về cài đặt và huấn luyện mạng học sâu đã chọn.
- **Chương 6: Kết quả huấn luyện và đánh giá.** Kết quả và các đánh giá về kết quả đạt được.
- **Chương 7: Kết luận.**

# Chương 1

## Tổng quan về hệ thống phân loại ảnh giao thông kết hợp với lưu trữ dữ liệu lớn

Bài toán hệ thống phân loại ảnh giao thông kết hợp lưu trữ dữ liệu lớn là một vấn đề khó thuộc lĩnh vực học sâu (deep learning) và liên quan đến vấn đề lưu trữ dữ liệu lớn. Bài toán đặt ra nhiều thách thức, để tìm được câu trả lời phù hợp. Chương này chúng ta tổng quan về các mạng học sâu cũng như các vấn đề dữ liệu lớn để thấy được các nghiên cứu đã có.

## 1.1 Giới thiệu hệ thống phân loại ảnh giao thông kết hợp với dữ liệu lớn

Hệ thống phân loại ảnh giao thông kết hợp với lưu trữ dữ liệu lớn bao gồm hai vấn đề chính đó là sử dụng lĩnh vực học sâu để huấn luyện tập dữ liệu ảnh giao thông với mục đích dự đoán và kiểm soát ùn tắc giao thông. Vấn đề kế tiếp đó chính là hệ thống lưu trữ dữ liệu video từ camera.

Vấn đề kết hợp lĩnh vực học sâu và với hệ thống dữ liệu lớn đang được nhiều nhà nghiên cứu cũng như nhiều công ty lớn đang tìm hiểu.

### 1.1.1 Các công trình nghiên cứu

#### Vấn đề phân loại ảnh

Trước tiên, với lĩnh vực phân loại ảnh, hiện nay đã có rất nhiều nhóm hoặc công ty đã cho ra đời nhiều kiến trúc mạng neuron phức tạp có độ chính xác cao như LeNet, ResNet, AlexNet, GoogleNet. Cùng với sự phát triển của tập dữ liệu ImageNet với 1000 lớp phân loại tổng cộng khoảng 1,2 triệu ảnh thì các kiến trúc mạng phân lớp có kiến trúc ngày càng phức tạp hơn được tạo ra nhằm đáp ứng tối đa nhu cầu phân loại đó.

#### Vấn đề dữ liệu lớn

Google, Oracle hay Amazon, Apache là những người đi đầu trong việc phát triển các dịch vụ cũng như framework lưu trữ dữ liệu lớn như Apache Hadoop, dịch vụ Google Cloud,... Tuy nhiên, một số các dịch vụ thường hay có giá thanh đắt đỏ.

Ngoài ra, hiện nay còn có một số nhóm nghiên cứu, trực hay không trực thuộc các tổ chức doanh nghiệp cũng đã phát triển các hệ thống kết hợp lĩnh vực học máy, học sâu với các kiến trúc lưu trữ dữ liệu lớn như Databricks, Yahoo với TensorflowOnSpark.

## 1.2 Kết luận chương

Đề tài "Nhận dạng ảnh giao thông qua camera hành trình này" được thực hiện với mục đích xây dựng một hệ thống mô phỏng lưu trữ dữ liệu giao thông sử dụng các công cụ lưu trữ dữ liệu lớn và kết nối với mạng học sâu huấn luyện trên tập ảnh giao thông tự xây dựng.

Luận văn sẽ tập trung nghiên cứu về lý thuyết mô hình học sâu, kiến trúc mạng tích chập được sử dụng, thực hiện huấn luyện trên tập ảnh giao thông hai lớp kẹt xe và thông thoáng. Từ đó, kết nối xây dựng và cài đặt hệ thống mô phỏng lưu trữ dữ liệu camera giao thông sử dụng các nền tảng dữ liệu lớn.

## Chương 2

# Mạng neural và Mạng neural tích chập (Convolutional Neural Networks CNN

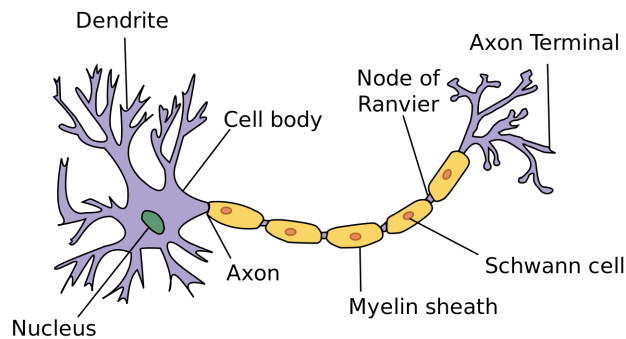
### 2.1 Mạng neural (Neural Network)

#### 2.1.1 Mạng neural ( Neural Network )

Theo khái niệm về sinh học, mạng neural là sự kết nối giữa các tế bào thần kinh neural lại với nhau. Trong lĩnh vực trí tuệ nhân tạo, mạng neural còn được gọi là Artificial Neural Network (ANN) - mạng neural nhân tạo, đây là mô hình xử lý dữ liệu, mô phỏng lại chức năng và cách hoạt động của hệ thống neural sinh học ở con người. Hình [2.1](#) minh họa cấu trúc của tế bào thần kinh neuron.

Mạng neural có gồm nhiều đơn vị kết nối, làm việc như một thể thống





Hình 2.1: Tế bào thần kinh neuron sinh học [1]

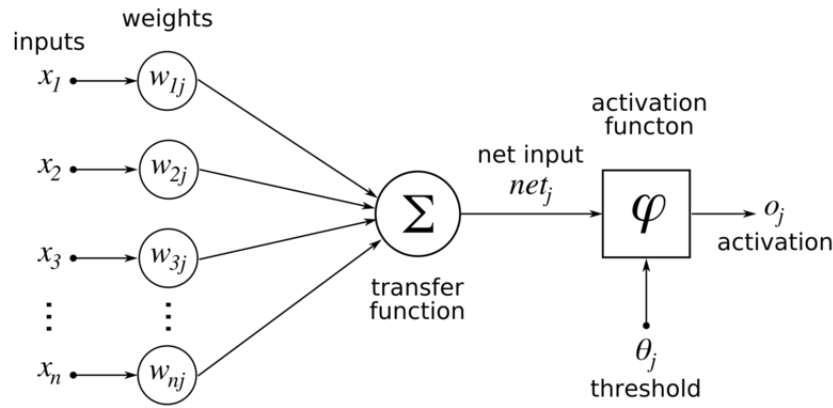
nhất thông qua việc trao đổi thông tin nhờ các liên kết.

### 2.1.2 Cấu trúc mạng neural

Như đã trình bày, các neuron trong một mạng làm việc như một thể thống nhất bằng việc trao đổi thông tin. Thực tế, đây là quá trình điều chỉnh các trọng số được truyền từ input ban đầu kết hợp với các hàm tính toán để có được các thông số trọng số phù hợp nhất. Quá trình này còn được gọi là quá trình học hay huấn luyện. Hình 2.2 mô tả cấu trúc đơn giản nhất của một mạng neuron.

#### Cấu trúc mạng neuron

- **Tập các node:** bao gồm nhiều node, mỗi node là đơn vị nhỏ nhất giữ chức năng xử lý thông tin của mạng.
- **Các tầng:** Các node trên được xếp thành các tầng, các node chung một tầng không thể kết nối nhau. Trong đó tầng input và tầng output là 2 tầng thiết yếu. Tùy vào một số mạng cụ thể có thể có thêm một hay nhiều tầng nằm ở giữa được gọi là tầng ẩn (hidden layer).



Hình 2.2: Cấu trúc cơ bản mạng neuron[2]

- **Tầng input - input layer:** các nối ở tầng này nhận dữ liệu đầu vào và truyền tới các node ở các tầng kế tiếp, trong một số trường hợp còn có chức năng xử lý thông tin.
  - **Tầng ẩn - hidden layer:** một số mạng có thể có thêm tầng ẩn, số lượng tầng ẩn trong một mạng có thể nhiều hơn 1. Có chức năng nhận các giá trị từ từng input hoặc tầng ẩn trước nó, tính toán các giá trị và gửi đến các node ở các tầng ẩn hoặc tầng output tiếp theo đó tùy theo từng mạng cụ thể.
  - **Tầng output - output layer:** nhận giá trị từ tầng trước đó (tầng ẩn hoặc tầng input) để tính toán các giá trị ngõ ra.
- **Các liên kết:** mỗi node trong một tầng truyền thông tin qua các node ở các tầng khác thông qua các liên kết. Các giá trị mà các liên kết này được gán sẽ được gọi là trọng số liên kết (weight). Giá trị trọng số được kết nối vào neuron j với neuron k là  $w_{kj}$ .
  - **Hàm truyền - transfer function:** dùng để tính tổng các tích input

với trọng số liên kết của nó.

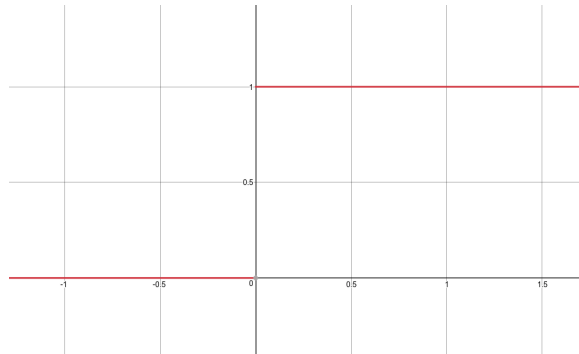
$$\sum input_j * w_{ij}$$

- **Activation function:** dùng để tính toán giá trị input sang giá trị output. Tùy vào mục đích và cụ thể từng loại mạng mà có nhiều loại activation function khác nhau.

Trong các bài toán khác nhau, người có những loại hàm activation như sau.

– *Step function:*

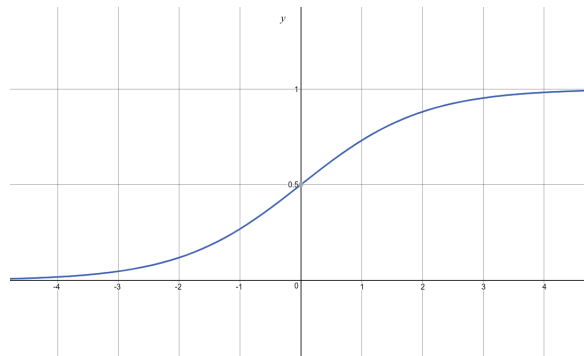
$$\begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}$$



Hình 2.3: Đồ thị hàm step

– *sigmoid function:*

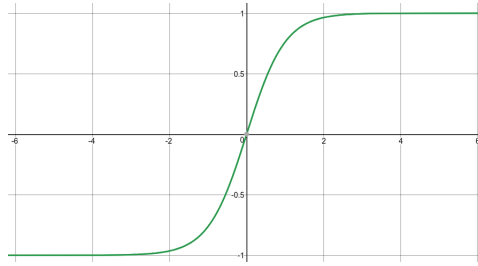
$$A(x) = \frac{1}{1 + e^{-x}}$$



Hình 2.4: Đồ thị hàm sigmoid

– *Tanh function:*

$$\text{Tanh}(x) = \frac{2}{1 + e^{-2x}} - 1$$



Hình 2.5: Đồ thị hàm Tanh

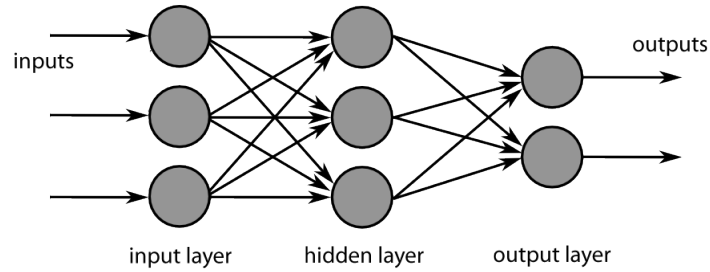
### 2.1.3 Mô hình Feedforward Neural Network

Thời điểm hiện nay, chúng ta có rất nhiều loại mô hình mạng neuron do sự khác nhau về sự kết hợp cũng như về mặt kiến trúc và thuật toán mà mạng đó áp dụng. Trong phần này, chúng ta sẽ tìm hiểu về mô hình mạng Feedforward Neural Network (FFNN), đây là kiến trúc mạng neuron được sử dụng phổ biến trong các bài toán dự báo. Mô hình gồm hai thành phần chính đó là kiến trúc feedforward - mạng truyền thẳng và giải thuật Backpropagation được áp dụng trong mạng.

#### 2.1.3.1 Kiến trúc Feedforward

Đối với mạng feedforward, cấu trúc gồm một tầng input, một tầng output và có thể có nhiều hơn một tầng ẩn nằm giữa hai tầng input và output.

Như hình 2.6, một mạng Feedforward, trong tầng input và output thì số lượng neuron tại mỗi hai tầng này sẽ là cố định tùy theo đặc tính của dữ liệu. Đối với tầng ẩn, số lượng tầng ẩn cũng như số lượng neuron trong mỗi tầng tùy thuộc vào cá nhân thiết kế.



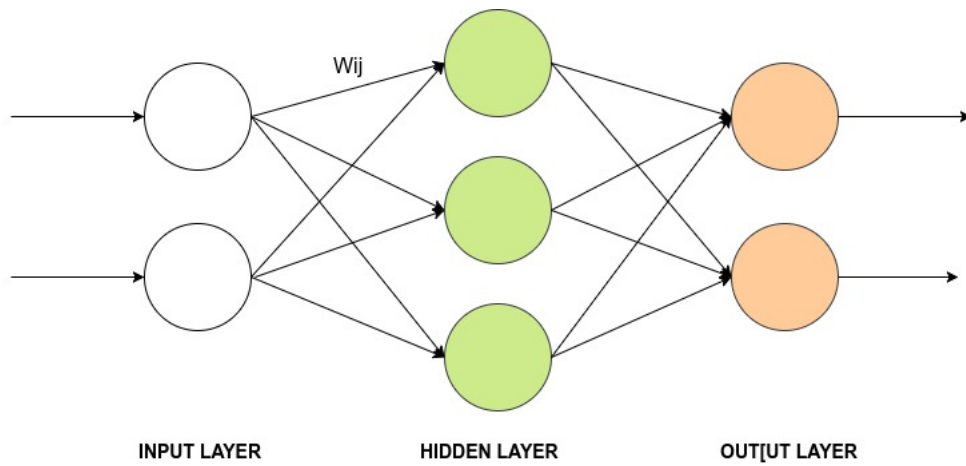
Hình 2.6: Ví dụ kiến trúc mạng feedforward [3]

### 2.1.3.2 Giải thuật Backpropagation

Ở nội dung này, chúng ta sẽ không đi sâu vào kỹ thuật xử lý các phép tính [8] đạo hàm mà chỉ trình bày giải thuật lan truyền ngược một cách đơn giản nhất.

Các ký hiệu và hàm được dùng trong trình bày giải thuật:

- $W_{ij}$ : là trọng số nối node thứ  $i$  tới node  $j$  ở layer kế tiếp.
- $I_j$ : là đầu vào tại node thứ  $j$ .
- $O_j$ : là kết quả xuất tại node thứ  $j$ .
- $\theta_j$ : bias tại node  $j$ .
- $l$ : tốc độ học của mạng (learning rate).
- $Err_j$ : giá trị lỗi tại node thứ  $j$ .
- Activation function được dùng trong nội dung này là hàm Sigmoid như mục trên



Hình 2.7: MLP

Nội dung thuật toán.

**Input:**

- Mạng feed forward với  $n$  input,  $m$  node ở tầng ẩn, và  $p$  output.
- Hệ số học hay tốc độ học  $l$ .
- Tập dữ liệu huấn luyện  $D$ .
- Sai số học  $\epsilon$ .

**Output:** Vector các trọng số mới sau khi huấn luyện.

**Nội dung thuật toán:**

- **Bước 1:** Khởi tạo ngẫu nhiên các giá trị trọng số  $W_{ij}$ .
- **Bước 2:** Tính toán các giá trị đầu vào  $I_j$  và đầu ra  $O_j$ .
  - Tại node  $i$  ở tầng input:

$$I_i = x_i, O_i = I_i$$

- Tại node  $j$  ở tầng khác:

$$I_j = \sum_i W_{ij} O_i + \theta_j$$

$$O_j = \frac{1}{1 + e^{-I_j}}$$

- **Bước 3:** Tính toán lỗi trung bình và đánh giá.

- Tại node thuộc tầng output:

$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

- Tại node thuộc tầng ẩn:

$$Err_j = O_j(1 - O_j) \sum_k Err_k W_{jk}$$

Với  $Err_k, W_{jk}$  là giá trị lỗi tại node  $k$  ở tầng tiếp theo và giá trị trọng số của node  $j$  đến  $k$ .

Thuật toán sẽ dừng lại khi  $Err_k \leq \epsilon$

- **Bước 4:** Cập nhật các trọng số và độ lệch

$$W_{ij} = W_{ij} + (l)Err_j O_i$$

$$\theta_j = \theta_j + (l)Err_j$$

Thuật toán sẽ tiếp tục lặp lại bước 2 cho đến khi thỏa điều kiện dừng và cho ra các tập trọng số và độ lệch tốt nhất.



## **2.2 Mạng neural tích chập (Convolutional neural network)**

### **2.2.1 Giới thiệu mạng neural tích chập**

Trong vài năm trở lại đây, chúng ta thấy được sự nở rộ của các hệ thống thông minh từ các công ty công nghệ lớn trên thế giới. Các chức năng nhận dạng, phân loại hay dự đoán được áp dụng rộng rãi vào các lĩnh vực thương mại, vận tải..v..v.

Mô hình Deep learning được sử dụng phổ biến và phát triển giúp các hệ thống thông minh có độ chính xác cao ngày nay chính là Convolutional Neural Networks(CNN) - mạng neuron tích chập. Trong các nội dung tới, chúng ta sẽ tìm hiểu các khái niệm, kiến trúc, cũng như ứng dụng của CNN trong lĩnh vực phân loại ảnh.

### **2.2.2 Mô hình mạng neural tích chập**

#### **2.2.2.1 Input và output**

Phân loại ảnh là công đoạn chuyển hóa từ một đầu là một hình ảnh và kết quả là một nhãn ứng với hình ảnh đó hoặc là các xác suất mà hệ thống dự đoán dựa trên đặc điểm của ảnh. Với con người, công việc nhận diện này được hình thành từ khi mới sinh ra, chúng ta có thể đưa ra kết quả của một hình ảnh bất kỳ mà không chút khó khăn. Nhưng máy tính thì không đơn giản như vậy, đầu vào và kết quả phải được đưa về dạng kỹ thuật số mà máy có thể hiểu được.

Khi một máy tính nhận vào một ảnh, nó sẽ thấy một mảng các giá trị

pixel tùy thuộc vào kích thước và độ phân giải của ảnh[9]. Ví dụ, một ảnh màu có kích thước 224 x 224 pixel thì máy tính sẽ thấy hình ảnh này dưới dạng một mảng có kích thước 224 x 224 x 3, giá trị 3 do thuộc tính ảnh màu(RGB) mà có được, giá trị này sẽ là 1 nếu đây là ảnh trắng đen.

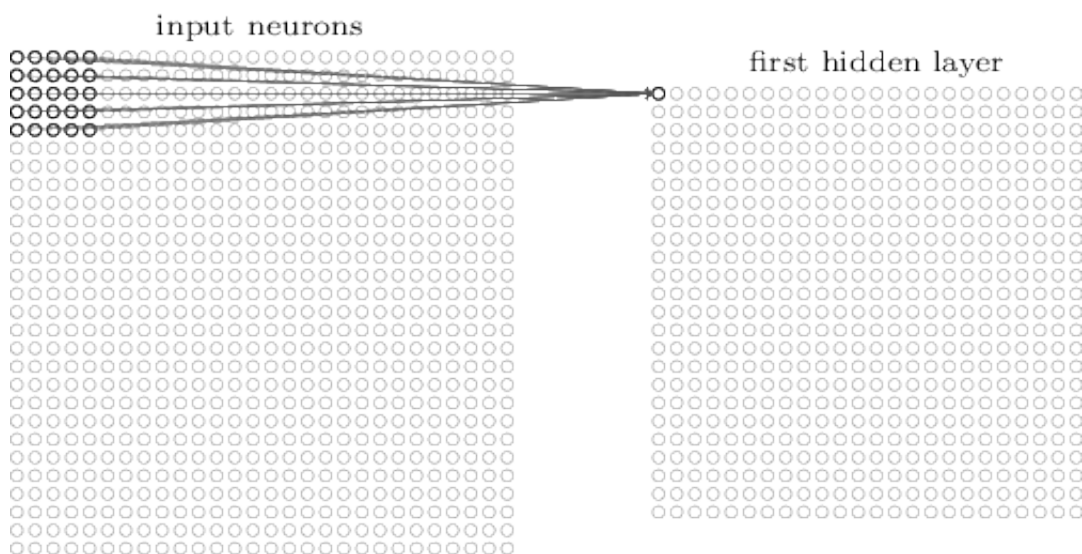
Đối với output, đây là một mảng các giá trị xác suất, mảng giá trị này cũng tùy thuộc vào số lượng nhãn(lớp) cần dự đoán. Ví dụ, (0.90 cho xe ô tô, 0.1 cho xe tải).

#### 2.2.2.2 Convolutional layer

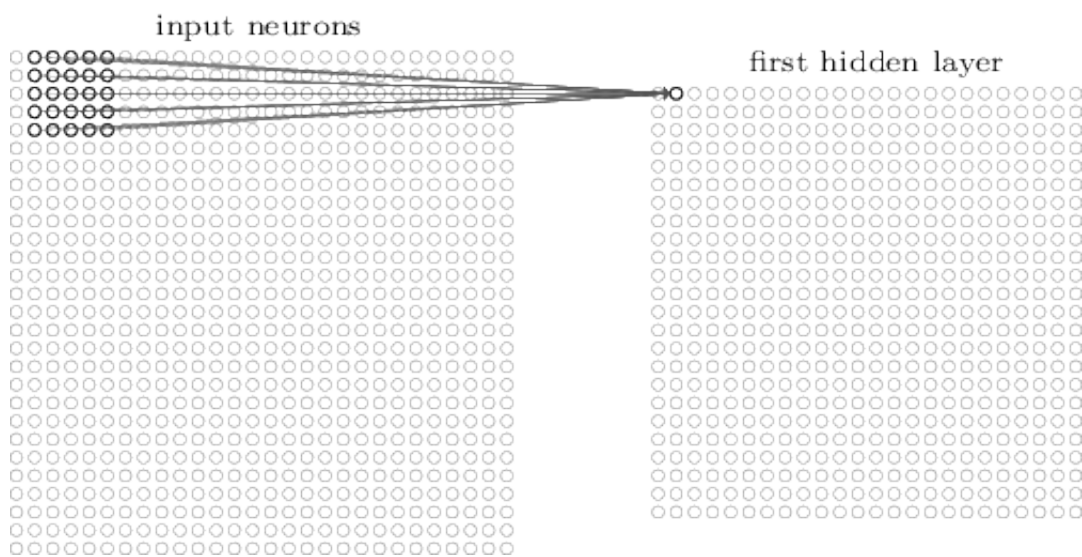
Tầng đầu tiên của một mạng CNN luôn luôn là tầng tích chập (convolutional layer)[10]. Như đã biết đầu vào (input) là một mảng các giá trị pixel. Trong trường hợp cụ thể để dễ hình dung ta chọn input là mảng các giá trị pixel có kích thước 32 x 32 x 3 với 32 x 32 là chiều dài và chiều rộng của tấm hình và 3 là giá trị RGB khi là ảnh màu. Đối với input trên có nghĩa là sẽ có một ma trận có kích thước 32 x 32 pixel mỗi pixel sẽ chứa 3 giá trị mà mỗi giá trị đó lần lượt biểu diễn cho giá trị của 3 màu sắc trên máy tính là đỏ(RED), lục(GREEN) và lam(BLUE).

Tạm thời bỏ qua giá trị RGB để đi vào cách hoạt động của tầng tích chập này. Cách đơn giản để giải thích cách hoạt động của tầng tích chập là tưởng tượng sẽ có một khuôn sẽ trượt từ phía trên bên trái cho đến hết tấm ảnh[11]. Với kích thước ảnh là 32 x 32 như trên, chọn kích thước ô trượt ví dụ là 5 x 5. Ô trượt có kích thước 5 x 5 sẽ trượt lần lượt qua cả input ảnh, ô trượt này được gọi là kernel hay filter(bộ lọc). Bộ lọc là một mảng các giá trị trọng số. Một điểm ghi chú là chiều sâu của bộ lọc sẽ bằng với chiều sâu của ảnh, với input 32 x 32 x 3 thì bộ lọc cũng sẽ có 5 x 5 x 3. Hình 2.8 và

2.9 minh họa cách kernel trượt trên input.

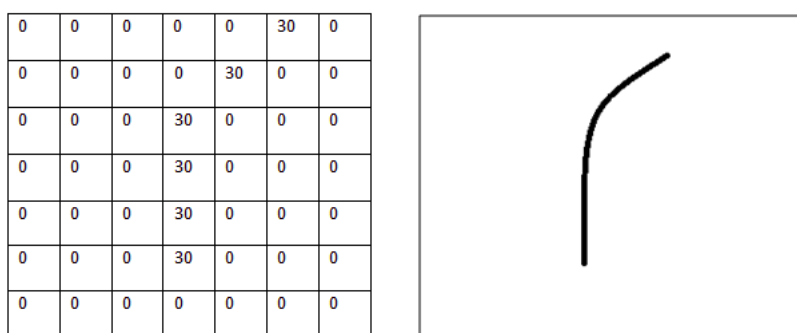


Hình 2.8: Convolutional layer [4]



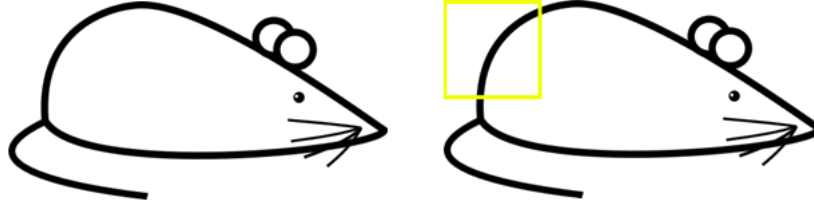
Hình 2.9: Convolutional layer [4]

Bây giờ ta sẽ đi vào việc mà tăng tích chập thực sự làm với những phép tính. Như đã trình bày rằng mỗi bộ lọc sẽ là một mảng các giá trị pixel, công dụng của mảng giá trị này nhằm mục đích phát hiện các đặc tính của mỗi vùng input mà filter trượt qua. Các đặc tính ở đây có thể là đường thẳng, đường cong, màu đơn giản. Ví dụ ta có một filter có kích thước là  $7 \times 7 \times 3$  dùng để phát hiện một dạng đường cong như hình 2.10.

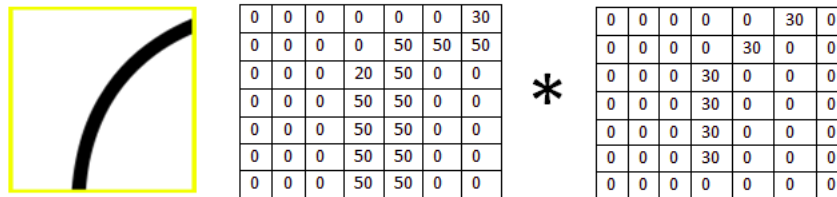


Hình 2.10: Mảng các giá trị của bộ lọc [4]

Khi bộ lọc trên trượt đến vùng được đánh dấu vàng có dạng đường cong giống với bộ lọc như hình 2.11. Lúc đó phép toán tích chập sẽ được thực hiện như hình 2.12 với ma trận bên trái chính là giá trị pixel của vùng được đánh dấu trên ảnh mà bộ lọc trượt tới, ma trận bên phải chính là bộ lọc được sử dụng hiện tại.



Hình 2.11: ảnh đầu vào [5]



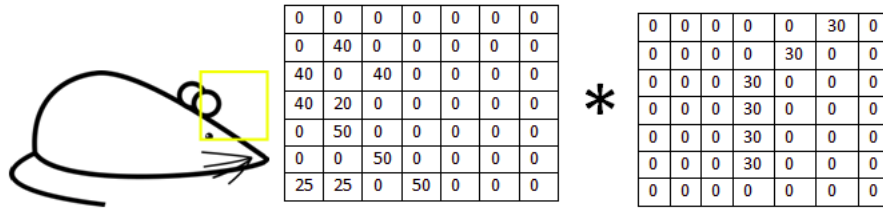
Hình 2.12: phép toán tích chập [5]

Kết quả của phép tính trên sẽ có kết quả như sau:

$$(50 * 30) + (50 * 30) + (50 * 30) + (20 * 30) + (50 * 30) = 6600$$

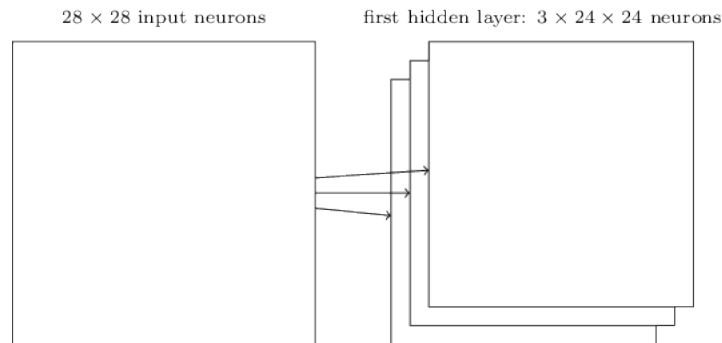
Đây là một con số rất lớn, thông thường nếu bộ lọc trượt tới một vùng mà vùng đó có hình dạng tương tự như bộ lọc thì kết quả khi thực hiện phép tính là một con số rất lớn. Ngược lại, kết quả sẽ ra rất nhỏ hoặc bằng 0. Ví dụ là hình ảnh 2.13 kết quả sẽ là 0 khi thực hiện phép tính

Trên đây là mô tả đơn giản về tầng tích chập, trên thực tế, chúng ta có thể có nhiều bộ lọc để phát hiện các khuôn mẫu, hình dạng khác nhau trong ảnh đầu vào. Kết quả xuất của tầng tích chập thứ nhất còn được gọi là bản



Hình 2.13: phép toán tích chập [5]

đồ đặc tính (feature map) và có thể có nhiều feature map cho một input sau khi hoàn thành tầng tích chập. Như 2.14 biểu diễn một input kích thước như hình sau khi qua tầng tích chập và sử dụng tập các bộ lọc 5 x 5 cho ra tập 3 feature map mà mỗi cái nhận diện được một khuôn dạng khác nhau xuất hiện trong input.

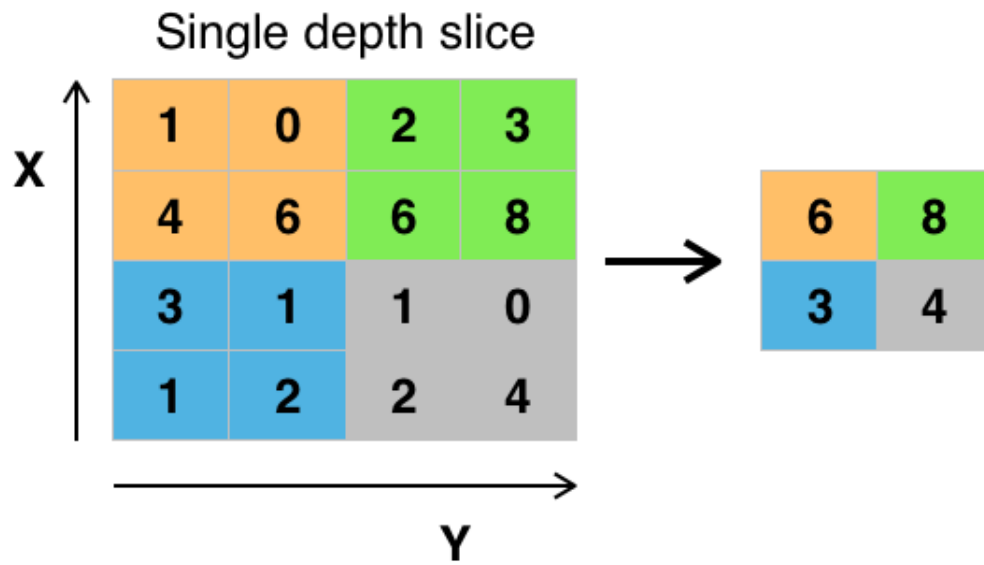


Hình 2.14: kết quả tầng tích chập [5]

### 2.2.2.3 Pooling Layer

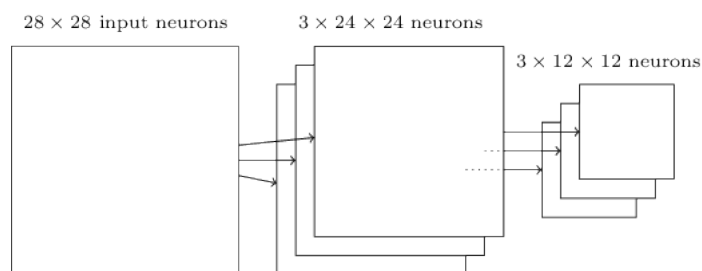
Sau khi qua một hoặc vài tầng tích chập, CNN sẽ chứa các tầng tổng hợp (Pooling Layer) ngay sau đó. Ở đây, tầng tổng hợp sẽ đơn giản hóa thông tin được lấy từ đầu ra từ tầng tích chập trước đó. Có nhiều kiểu tổng hợp

khác nhau đối với tầng này, nhưng max-pooling là phép tổng hợp phổ biến được sử dụng. Hình 2.15 biểu diễn một ví dụ phép max-pooling với một bộ lọc có kích thước là  $2 \times 2$ . Giá trị lớn nhất trong mỗi vùng được trượt qua sẽ được chọn làm kết quả xuất ra.



Hình 2.15: max-pooling  $2 \times 2$  [6]

Công dụng của phép pooling này giúp giảm đi kích thước của tập miêu tả đặc trưng từ đó cũng làm cho số lượng tham số và tính toán giảm theo. Và do chúng ta có thể có nhiều feature map từ tầng tích chập nên phép tổng hợp cũng sẽ được áp dụng độc lập cho mỗi feature map. Nếu có 3 feature map thì sẽ có 3 phép tổng hợp trong trường hợp này là max-pooling.



Hình 2.16: Ví dụ tầng tổng hợp [4]

#### 2.2.2.4 Fully Connected Layer

Sau khi thông tin input đã qua các tầng tích chập và tổng hợp, các giá trị dữ liệu sẽ đi đến tầng fully connected để xuất ra kết quả. Kết quả tại tầng fully connected là một vector có kích thước bằng với số lớp(nhãn) mà bài toán cần dự đoán. Như bài toán phân loại ảnh giao thông ùn tắc và thông thoáng thì lúc này số nhãn cần dự đoán và kích thước vector tại tầng này sẽ bằng 2. Giá trị của các phần tử trong vector sẽ là giá trị xác suất của mỗi nhãn mà mạng dự đoán,  $[0.8, 0.2]$  sẽ biểu diễn 80% ảnh này thuộc lớp 1 và 20% ảnh thuộc lớp thứ 2. Về cơ bản, cách kết nối ở tầng này giống như cách kết nối neuron giữa các tầng với nhau ở mạng neuron ở mục trước. Khi đó, tất cả neuron ở tầng pooling sẽ kết nối với từng neuron trong tầng cuối.



## Chương 3

# Apache Hadoop

### 3.1 Tổng quan về hệ thống Apache Hadoop

Apache Hadoop là một dự án phát triển phần mềm nhằm cung cấp một nền tảng phân tán, có thể mở rộng linh hoạt và có độ tin cậy cao.

Ngoài ra Hadoop còn được xem là một thư viện hay framework cho phép xử lý phân tán khối lượng lớn dữ liệu trên nhiều cụm máy tính bằng các mô hình lập trình. Framework này được thiết kế với mục đích có khả năng mở rộng từ một máy chủ đơn lẻ lên đến rất nhiều trạm làm việc mà mỗi máy trạm có khả năng tính toán và lưu trữ cục bộ.

## 3.2 Hadoop distributed file system - HDFS

### 3.2.1 Thiết kế

HDFS là một hệ thống file nhằm lưu trữ một lượng rất rất lớn (Lớn ở đây theo nghĩa có thể là hàng trăm megabytes, gigabytes, hoặc terabytes) với cơ chế streaming data access trên những thiết bị phổ thông <sup>1</sup>. Đây là cơ chế phân luồng dữ liệu trong HDFS với mục đích ghi một lần chạy nhiều lần. Điển hình là dữ liệu sẽ được sinh và sao chép từ nguồn và sau đó có rất nhiều tiến trình phân tích khác nhau thực thi dữ liệu trên. Mỗi hoạt động phân tích sẽ thực thi liên quan đến một phần nào đó khác nhau trên cả một tập dữ liệu trên.

Từ các thiết bị phổ thông ở đây là những những thiết bị phần cứng máy tính, HDFS không yêu cầu một phần cứng đắt tiền hay độ tin cậy cao. Mà nó được thiết kế để chạy trên những cụm máy tính từ nhiều nhà cung ứng khác nhau. Vì thế mà xác suất để một node (đơn vị phần cứng) gặp lỗi và thất bại là rất lớn, đặc biệt là những cụm có hàng ngàn máy trạm. Với đặc tính đó, HDFS được thiết kế sao cho không có sự gián đoạn nào được phát hiện ở người dùng khi mà việc một số lượng node gặp lỗi giữa chừng.

### 3.2.2 Các khái niệm trong HDFS

- **Block(sector):** trong các đĩa cứng thông thường, đây là các phần nhỏ cuối cùng được chia ra để chứa dữ liệu, và đây cũng là đơn vị nhỏ nhất mà một lần đọc và ghi dữ liệu được hiện thực. Thông thường, block ở các đĩa cứng có dung lượng nhỏ chỉ tầm 512 byte. Ở HDFS,

---

<sup>1</sup>các máy tính hoặc máy trạm

block thường có dung lượng lớn hơn rất nhiều - mặc định là 128MB. Do dung lượng của một block trên đĩa lớn như vậy nên các file tập tin trong HDFS nhiều khi có dung lượng và được lưu trữ nằm trong cả một block (trong HDFS). Nguyên nhân của việc block trong HDFS có dung lượng lớn như vậy là nhằm việc tối thiểu hóa chi phí tìm kiếm [12]

- **Namenodes và Datanodes:** một HDFS gồm nhiều node, trong đó có hai loại chính đó là một master node (Namenode) - máy chủ và nhiều datanodes (những thiết bị nô lệ). Namenode là chương trình chỉ đạo các datanodes thực hiện các nhiệm vụ I/O ở mức thấp. Còn datanodes được sự điều phối của Namenode, lưu trữ và thường xuyên báo cáo với master các khối(block) mà nó hiện đang lưu trữ. Ngoài ra, các datanodes còn giao tiếp với nhau để sao lưu các khối dữ liệu dự phòng.
- **jobTracker:** Trình nền JobTracker là một liên lạc giữa ứng dụng của bạn và Hadoop. Một khi bạn gửi mã nguồn của bạn tới các cụm (cluster), JobTracker sẽ quyết định kế hoạch thực hiện bằng cách xác định những tập tin nào sẽ xử lý, các nút được giao các nhiệm vụ khác nhau, và theo dõi tất cả các nhiệm vụ khi chúng đang chạy. Nếu một nhiệm vụ (task) thất bại (fail), JobTracker sẽ tự động chạy lại nhiệm vụ đó, có thể trên một node khác, cho đến một giới hạn nào đó được định sẵn của việc thử lại này. Chỉ có một JobTracker trên một cụm Hadoop. Nó thường chạy trên một máy chủ như là một nút master của cluster.
- **TaskTraker:** Như với các trình nền lưu trữ, các trình nền tính toán cũng phải tuân theo kiến trúc master/slave: JobTracker là giám sát

tổng việc thực hiện chung của một công việc MapReduce và các task-Tracker quản lý việc thực hiện các nhiệm vụ riêng trên mỗi node slave. Mỗi TaskTracker chịu trách nhiệm thực hiện các task riêng mà các JobTracker giao cho. Mặc dù có một TaskTracker duy nhất cho một node slave, mỗi TaskTracker có thể sinh ra nhiều JVM để xử lý các nhiệm vụ Map hoặc Reduce song song. Một trong những trách nhiệm của các TaskTracker là liên tục liên lạc với JobTracker. Nếu JobTracker không nhận được nhịp đập từ một TaskTracker trong vòng một lượng thời gian đã quy định, nó sẽ cho rằng TaskTracker đã bị treo (cached) và sẽ gửi lại nhiệm vụ tương ứng cho các nút khác trong cluster. Cấu trúc liên kết này có một node Master là trình nền NameNode và JobTracker và một node đơn với SNN trong trường hợp node Master bị lỗi. Đối với các cụm nhỏ, thì SNN có thể thường chú trong một node slave. Mặt khác, đối với các cụm lớn, phân tách NameNode và JobTracker thành hai máy riêng. Các máy slave, mỗi máy chỉ lưu trữ một DataNode và Tasktracker, để chạy các nhiệm vụ trên cùng một node nơi lưu dữ liệu của chúng

### **3.2.3 HIVE**

Hive là hạ tầng kho dữ liệu cho Hadoop. Nhiệm vụ chính là cung cấp sự tổng hợp dữ liệu, truy vấn và phân tích. Nó hỗ trợ phân tích các tập dữ liệu lớn được lưu trong HDFS của Hadoop cũng như trên Amazon S3. Điểm hay của HIVE là hỗ trợ truy xuất giống SQL đến dữ liệu có cấu trúc, được biết với tên HiveSQL (hoặc HQL) cũng như phân tích big data với MapReduce. Hive không được xây dựng để hồi đáp nhanh các câu truy vấn nhưng nó được

xây dựng cho các ứng dụng khai thác dữ liệu (data mining). Các ứng dụng khai thác dữ liệu có thể mất nhiều phút đến nhiều giờ để phân tích dữ liệu và HIVE được dùng chủ yếu.

### 3.2.3.1 Cách tổ chức của HIVE

Dữ liệu được tổ chức thành 3 định dạng trong HIVE.

**Tables:** Chúng rất tương tự như bảng (tables) trong RDBMS và chứa các dòng (rows). Hive chỉ được xếp lớp trên HDFS, do đó tables được ánh xạ trực tiếp vào các thư mục của hệ thống tập tin. Nó cũng hỗ trợ các tables được lưu trên các hệ thống tập tin khác.

**Partitions:** Hive tables có thể có nhiều hơn 1 partition. Chúng được ánh xạ với các thư mục con và các hệ thống tập tin.

**Buckets:** Trong Hive, dữ liệu có thể được chia thành các buckets. Buckets được lưu trữ như các tập tin trong partition trong hệ thống tập tin.

Hive cũng có metastore để lưu tất cả metadata. Nó là cơ sở dữ liệu quan hệ chứa thông tin khác nhau liên quan đến Hive Schema (column types, owners, key-value data, statistics,...). Chúng ta có thể dùng MySQL cho việc này.

### 3.2.3.2 HIVESQL

Ngôn ngữ truy vấn Hive cung cấp các toán tử cơ bản giống SQL. Đây là một số tác vụ mà HQL có thể làm dễ dàng.

- Tạo và quản lý tables và partitions.
- Hỗ trợ các toán tử Relational, Arithmetic và Logical khác nhau.

- Evaluate functions
- Tải về nội dung 1 table từ thư mục cục bộ hoặc kết quả của câu truy vấn đến thư mục HDFS.
- Đây là ví dụ truy vấn HQL:

```
SELECT upper(name), salesprice
FROM sales ;
SELECT category , count(1)
FROM products
GROUP BY category ;
```

### 3.3 Tại sao sử dụng Apache Hadoop trong đề tài

Hiện nay trên các xe buýt đều được trang bị camera hành trình, các video này ghi lại hình ảnh trên các con đường. Các hình ảnh này được gửi về hệ thống một cách liên tục hàng ngày và hàng giờ. Do kích thước dữ liệu tăng lên đáng kể cần một hệ thống lưu trữ có khả năng mở rộng linh động, vì thế hệ thống HDFS do Apache Hadoop hỗ trợ là công cụ hợp lý. Ngoài ra hệ thống HDFS còn có thể kết hợp với các thư viện hỗ trợ học sâu như Tensorflow nên ta có thể áp dụng các mô hình trong lĩnh vực machine learning nói chung và deep learning nói riêng để xây dựng một hệ thống phân loại ảnh. Khi kết hợp các thành phần trên, ta có thể tinh chỉnh tập dữ liệu sao cho phù hợp với bài toán của thể của từng trường hợp cũng như các lĩnh vực khác mà không nhất thiết là lĩnh vực giao thông.

## Chương 4

# Phương pháp giải quyết

Ở chương này, chúng ta sẽ phân tích các vấn đề cụ thể trong đề tài và đưa ra lựa chọn giải quyết cần thiết.

### 4.1 Phân tích

#### 4.1.1 Mô tả tập dữ liệu

Một trong những vấn đề cần được quan tâm nhất trong đề tài này chính là đặc điểm của tập dữ liệu. Sau một khoảng thời gian truy xuất video lấy từ camera hành trình, chúng ta có được khoảng hơn 14000 ảnh về giao thông. Theo như đã tìm hiểu và so sánh, có thể thấy đây không phải là một tập ảnh lớn nếu so với tập dữ liệu ImageNet gồm khoảng 14 triệu ảnh thuộc 1000 lớp khác nhau. Ngoài ra, các ảnh thu được từ camera hành trình thật chất không phải là bộ ảnh có chất lượng tốt.

Toàn bộ số ảnh trên được thực hiện gán nhãn dựa trên hai nhãn chính là kẹt và thông thoáng để phục vụ cho việc huấn luyện. Dưới đây là một số

hình ảnh được trích ra từ tập ảnh. Với hai hình đầu được lấy ra từ lớp thông thoáng, hai hình sau được lấy ra từ lớp ùn tắc.



(a) Ảnh mẫu 1

(b) Ảnh mẫu 2

Hình 4.1: Ảnh mẫu trong phân lớp thông thoáng



(a) Ảnh mẫu 3

(b) Ảnh mẫu 4

Hình 4.2: Ảnh mẫu trong phân lớp ùn tắc

#### 4.1.2 Các vấn đề về lưu trữ

Trong đề tài này, ta cần thực hiện mô phỏng lại một hệ thống lưu trữ các dữ liệu ghi nhận từ xe buýt đang hoạt động. Các dữ liệu này bao gồm



các video được ghi lại từ các camera hành trình trên xe buýt và dữ liệu GPS mà xe buýt đó gửi về liên tục.

## 4.2 Giải pháp đề xuất

### 4.2.1 Kiến trúc mạng googleNet

Với tập dữ liệu có kích thước không lớn việc xây dựng một mạng neuron tích chập để huấn luyện sẽ dẫn đến kết quả phân loại không tốt và có thể tốn kém thời gian nếu không có cấu hình đủ mạnh để thực hiện. Chính vì vậy, việc sử dụng phương pháp transfer learning là lựa chọn cần thiết.

#### 4.2.1.1 Phương pháp transfer learning

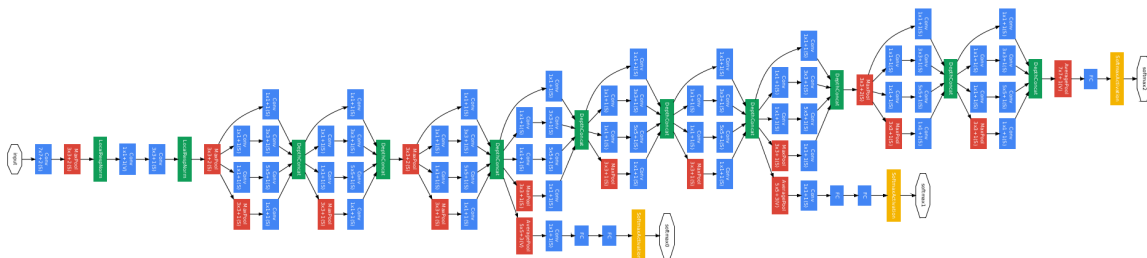
Ở các nội dung trước, ta đã trình bày cấu trúc của một mạng neuron tích chập bao gồm rất nhiều layers. Các layers đứng trước bao gồm nhiều tầng tích chập kết hợp với các hàm activation phi tuyến tính và các tầng tổng hợp. Tầng cuối cùng là một tầng kết nối đầy đủ (Fully Connected Layer) và thường là một hàm softmax, số lượng node (unit) ở tầng này bằng với số lượng lớp(label) cần phân loại. Vì thế ta đầu ra ở layer trước tầng cuối cùng là các vector đặc trưng (feature vector). Việc huấn luyện sử dụng mô hình gồm nhiều lớp như trên với một tập dữ liệu lớn khoảng hơn 1 triệu ảnh cũng tốn rất nhiều thời gian. Mà ở đây, chúng ta không có tập dữ liệu lớn như vậy và cũng không có thời gian quá nhiều. Vì thế, ta sử dụng một phương pháp là Transfer learning, sử dụng lại những mô hình sẵn có để áp dụng vào tập dữ liệu của chúng ta.

Để hiểu được phương pháp này, chúng ta nhìn lại, toàn bộ các lớp trong

một mạng CNN được coi là bộ Feature Extractor. Dựa trên nhận xét rằng các bức ảnh đều có những đặc tính giống nhau nào đó, với cơ sở dữ liệu khác, ta cũng có thể sử dụng phần Feature Extractor này để tạo ra các feature vectors. Sau đó, ta thay output layer cũng bằng một Softmax Regression (hoặc multi-class SVM) nhưng với số lượng units bằng với số lượng class ở bộ cơ sở dữ liệu mới. Ta chỉ cần train layer cuối cùng này. Kinh nghiệm thực tế của tôi cho thấy, việc làm này đã tăng kết quả phân lớp lên rất nhiều.

#### 4.2.1.2 Kiến trúc mạng GoogLeNet

Đây là kiến trúc mạng tích chập với 22 tầng. GoogLeNet còn là quán quân của ILSVRC 2014 [7]. Mạng googLeNet có cấu trúc mạng nằm trong mạng, có 9 tầng mà mỗi tầng là một inception module. Theo tài liệu cho biết, việc áp dụng inception module giúp làm giảm đáng kể số lượng tham số tính toán giúp giải quyết vấn đề về tài nguyên. 4.3 minh họa cho cấu trúc của mạng googLeNet.

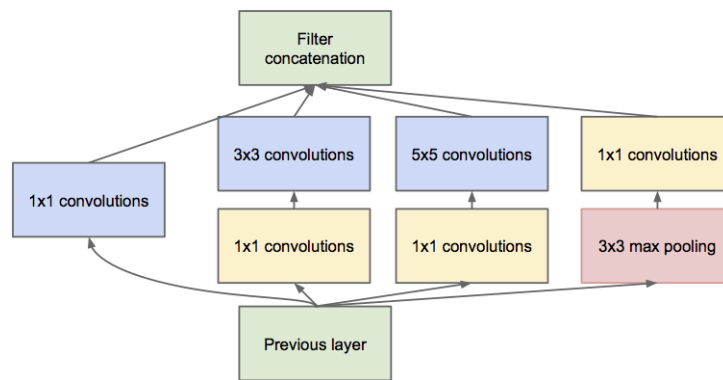


Hình 4.3: GoogLeNet [7]

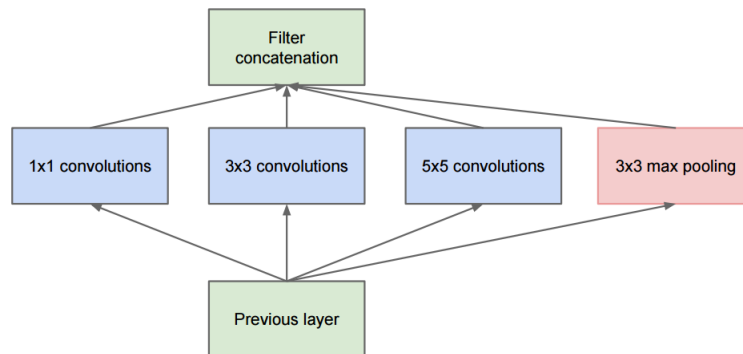
#### Inception module

Với minh họa kiến trúc của mạng googLeNet, ta sẽ thấy các layer là một khối

mạng nhỏ nằm bên trong. Đây là các inception module. Đối với các mạng tích chập thông thường, khi một tập dữ liệu bắt đầu đi vào một tầng thì sẽ chỉ có hai sự lựa chọn đó chính là tầng tích chập hoặc tầng tổng hợp, nhưng với googLeNet sẽ có tập input sẽ đi vào một lớp module tại đó sẽ các phương thức tích chập và pooling sẽ được tính toán một cách song song và độc lập với nhau [7].



Hình 4.4: inception module [7]



Hình 4.5: naive inception module [7]

Hình 4.4 miêu tả cấu trúc của một module trong mạng. Trong khi đó

4.5 là một ý tưởng ban đầu mà tác giả đã nghĩ tới. Ở 4.4 chúng ta thấy trước khi thực hiện các phép tích chập với các filter 3 x 3 và 5 x 5, input đều được xử lý qua phép tích chập với filter 1 x 1. Các bộ lọc 1 x 1 có tác dụng làm giảm đi chiều của các input[13], điều này giúp cho khối lượng tham số phải tính toán ở phép toán tích chập với bộ lọc 3 x 3 và 5 x 5 sẽ được giảm đi một cách đáng kể.

### 4.2.2 Cấu trúc hệ thống lưu trữ

Để đáp ứng cho nhu cầu lưu trữ dữ liệu video và dữ liệu GPS này chúng ta có thể sử dụng Apache Hadoop kết hợp với HIVE.

Cấu trúc của hệ thống mô phỏng này sẽ bao gồm ba máy lưu trữ và một máy làm nhiệm vụ là máy chủ (master) để điều khiển hoạt động của các máy còn lại. Các video sẽ được lưu trữ theo từng thư mục, mỗi thư mục chứa dữ liệu các video trong một ngày đó.

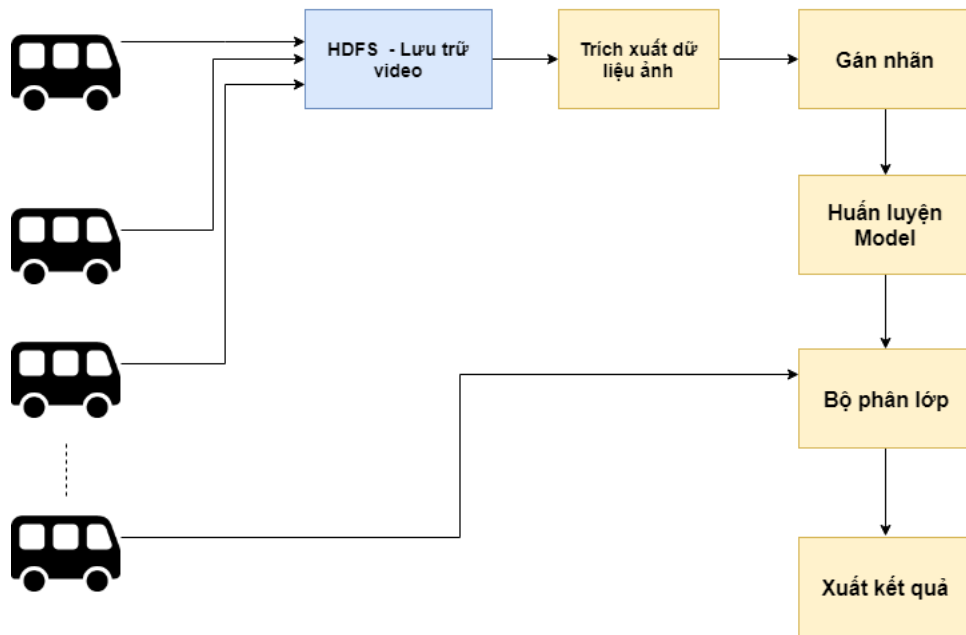
Đối với HIVE, các dữ liệu GPS sẽ được lưu trữ trên một bảng dữ liệu của HIVE, bảng này bao gồm nhiều trường mô tả dữ liệu hoạt động của xe buýt, trong đó có các trường dữ liệu chính để chúng ta truy xuất như sau: Longitude đây là kinh độ của xe buýt, Latitude đây là vĩ độ của xe buýt, tracktime là dữ liệu thời gian có định dạng (YYYY-MM-DD hh-mm-ss). Như vậy, với một mẫu tin được lưu, sẽ thể hiện được thời gian cụ thể cũng như vị trí GPS của xe buýt tại thời gian đó.

## Chương 5

# Hiện thực mô hình loại ảnh giao thông kết hợp Apache Hadoop

### 5.1 Tổng quan các bước

Hệ thống phân loại ảnh kết hợp Apache Hadoop và mô hình học sâu Deeplearning sẽ được thực hiện dựa trên sơ đồ trên. Hiện nay, các xe buýt đều được trang bị camera hành trình và hệ thống GPS. Các video được ghi lại sẽ được lưu tại một nơi. Do số lượng của các video được gửi về liên tục và hàng ngày và giờ, nên việc sử dụng hệ thống lưu trữ HDFS của Apache Hadoop là lựa chọn hợp lý. Tiếp đến là việc truy xuất ảnh từ video và gán nhãn ảnh phục vụ cho việc huấn luyện mô hình học sâu. Đây cũng là công đoạn tốn kém chi phí nhất. Sau cùng là việc huấn luyện tập dữ liệu đã xây dựng.



Hình 5.1: Mô hình phân loại ảnh giao thông kết hợp Hadoop

## 5.2 Các bước chuẩn bị

### 5.2.1 Chuẩn bị môi trường hiện thực

Bộ phân loại ảnh giao thông được huấn luyện trên nền tảng hệ điều hành Linux, ngôn ngữ Python phiên bản 3.6 kết hợp với thư viện Tensorflow mã nguồn mở chuyên được sử dụng cho những mô hình học sâu.

Tensorflow[14] là một thư viện học sâu mã nguồn mở được Google phát triển. Thư viện này đã thu hút được sự chú ý lớn từ cộng đồng Deep-learning. Tensorflow cho phép chạy các thuật toán machine learning trên nhiều GPU, có nhiều module được dựng sẵn giúp cho việc xây dựng và thực thi mô hình đơn giản hơn.

**Các khái niệm:**

- **Tensor:** đây là cấu trúc dữ liệu được sử dụng hoàn toàn trong Tensorflow. Hay nói cách khác, tất cả dữ liệu đều biểu diễn dưới dạng tensor. Đơn giản, tensor là một mảng gồm n chiều hay list kèm theo một số thuộc tính khác.
- **Rank:** còn được gọi là số chiều của dữ liệu

Rank	Đơn vị số	Ví dụ
0	Scalar	$s = 123$
1	Vector	$s = [0.8, 0.1, 0.1]$
2	Matrix	$s = [[1,2,3], [4,5,6], [7,8,9]]$
3	3-Tensor	$s = [ [ [1], [2], [3] ], [ [4], [5], [6] ], [ [7], [8], [9] ] ]$
n	n-Tensor	n chiều dữ liệu...

Bảng 5.1: Các chiều dữ liệu.

- **Shape:** biểu diễn chiều của tensor. Ví dụ,  $t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$  có shape là  $[3, 3]$ ,  $t = [[[1], [2], [3]], [[4], [5], [6]], [[7], [8], [9]]]$  có shape là  $[1, 3, 3], \dots$
- **Type:** là các kiểu dữ liệu được sử dụng trong Tensorflow. Một vài kiểu dữ liệu cơ bản như.

Data type	Python code	Mô tả
<i>DT-FLOAT</i>	tf.float32	32 bits floating point.
<i>DT-DOUBLE</i>	tf.float64	64 bits floating point.
<i>DT-INT16</i>	tf.int16	16 bits signed integer.
<i>DT-INT32</i>	tf.int32	32 bits signed integer.
<i>DT-INT64</i>	tf.int64	64 bits signed integer.
...	...	...

Bảng 5.2: Một vài kiểu dữ liệu.

## 5.3 Các bước hiện thực

### 5.3.1 Cài đặt Apache Hadoop

Trong phần hướng dẫn này chúng ta sẽ biết được cách cài đặt một cụm gồm nhiều máy Hadoop. Thông qua nhiều bước, ta sẽ biết cách cấu hình cũng như khởi tạo và kết thúc hoạt động của một cụm máy Hadoop.

#### 5.3.1.1 Cấu hình cần thiết để cài đặt một cụm 3 máy hadoop

Một cụm Hadoop có thể rất nhiều máy, nhưng để có được cái nhìn tổng quan và vì thời gian của đề tài nên chúng ta sẽ tiến hành cài Hadoop trên 3 máy tính. Việc bổ sung thêm máy trạm vào cụm sẽ được hướng dẫn ở nội dung cuối phần này.

- **Số lượng máy tính:** 3 máy tính.
- **Hệ điều hành:** sử dụng hệ điều hành Ubuntu phiên bản 14.04 hoặc 16.04 hoặc cao hơn.



- **Phiên bản Apache Hadoop:** Gói cài đặt [Hadoop phiên bản 3.1.0](#)

### 5.3.1.2 Cài đặt Hadoop trên máy Master(Namenode)

Mục này được thực hiện chỉ riêng trên máy được chọn làm Master.

#### 1. Các vấn đề trước khi cài đặt Hadoop:

- **Thêm địa chỉ IP đầu vào vào file */etc/hosts***

Listing 5.1: Nội dung file */etc/hosts*

```
127.0.0.1          localhost
127.0.1.1          master
192.168.1.50       master
192.168.1.61       slave
192.168.1.73       slave1
```

Chúng ta bỏ qua 2 dòng đầu của file và chú ý tới 3 dòng tiếp theo. Bên cột trái, đó chính là lần lượt địa chỉ IP của 3 máy sẽ cài Hadoop. 192.168.1.50 là máy được chọn làm master và từ *master* bên cột phải là hostname của máy đó. Tương tự 2 dòng cuối, đó là IP của 2 máy được chọn làm datanode và hostname lần lượt là *slave*, *slave1*.

- **Cài đặt Java:** hiện nay, Java được cài ở đây sẽ có phiên bản là Java 8. Thực hiện lần lượt các lệnh sau trên terminal để cài đặt Java 8.

Listing 5.2: Cài đặt Java 8

```
sudo add-apt-repository ppa:webupd8team/java
```

```
sudo apt-get update
```

```
sudo apt-get install oracle-java8-installer
```

- **Cấu hình SSH:** Thực hiện lần lượt các lệnh sau để cài cũng như cấu hình kết nối SSH máy master tới các máy datanodes.

Listing 5.3: Cài đặt SSH

```
sudo apt-get install openssh-server openssh-client  
ssh-keygen -t rsa -P ""
```

Sao chép nội dung file `.ssh/id_rsa.pub` của máy master đến file `.ssh/authorized_keys` của máy master cũng như các máy datanodes. Sau đó kiểm tra kết nối đã thành công hay chưa bằng lệnh.

Listing 5.4: Kết nối SSH

```
ssh slave  
ssh slave1
```

## 2. Cài đặt Hadoop

- **Tải gói cài đặt:** Tải gói cài đặt Hadoop theo link phía dưới  
<http://hadoop.apache.org/releases.html>
- **Giải nén gói cài đặt:** Giải nén đến thư mục `/usr/local` bằng câu lệnh  

```
tar xzf hadoop-3.1.0.tar.gz -C /usr/local/
```
- **Chỉnh sửa `/.bashrc`:** Thêm các biến môi trường cài đặt Hadoop vào file `/.bashrc` như nội dung sau.

```

export HADOOP_PREFIX="/usr/local/hadoop-3.1.0"
export PATH=$PATH:$HADOOP_PREFIX/bin
export PATH=$PATH:$HADOOP_PREFIX/sbin
export HADOOP_MAPRED_HOME=${HADOOP_PREFIX}
export HADOOP_COMMON_HOME=${HADOOP_PREFIX}
export HADOOP_HDFS_HOME=${HADOOP_PREFIX}
export YARN_HOME=${HADOOP_PREFIX}

```

Sau khi lưu nội dung chỉnh sửa, thực thi `source ~/.bashrc` để các biến hệ thống thiết lập các biến môi trường.

- **Cấu hình file `hadoop-env.sh`:** cấu hình biến `JAVA_HOME`

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle/
```

Thay đổi đường dẫn `JAVA_HOME` nếu cài đặt Java 8 ở một nơi khác.

- **Cấu hình file `core-site.xml`:** thêm các dòng sau

```

<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://master:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/home/'your\_user'/hdata</value>
  </property>
</configuration>

```

- **Cấu hình file hdfs-site.xml:** thêm các nội dung như sau

```
<configuration>
<property>
    <name>dfs.replication</name>
    <value>2</value>
</property>
</configuration>
```

- **Cấu hình file mapred-site.xml:** thêm các nội dung như sau

```
<configuration>
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
</configuration>
```

- **Cấu hình yarn-set.xml:** thêm các nội dung như sau

```
<configuration>
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
<property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>master:8030</value>
</property>
```

```

<property>
    <name>yarn.resourcemanager.address</name>
    <value>master:8040</value>
</property>
</configuration>

```

- **Cấu hình file workers:** thêm hostname của 3 máy trạm mà đã đặt ở file */etc/hosts*

```

master
slave
slave1

```

### 5.3.1.3 Cài đặt Hadoop trên các datanodes

#### 1. Các cấu hình chuẩn bị:

- **Thêm địa chỉ IP vào file */etc/hosts*:** thực hiện giống như đã làm ở máy master
- **Cài đặt Java 8**

#### 2. Cài đặt Hadoop cho tất cả máy datanodes: thao tác này sẽ được thực hiện trên terminal của máy master.

- **sử dụng SSH để gửi file cài đặt:** thực hiện lần lượt các lệnh sau để gửi file cài đặt hadoop tới các máy datanodes

```

scp /usr/local/hadoop-3.1.0 slave:/usr/local/hadoop-3.1.0
scp /usr/local/hadoop-3.1.0 slave1:/usr/local/hadoop-3.1.0

```

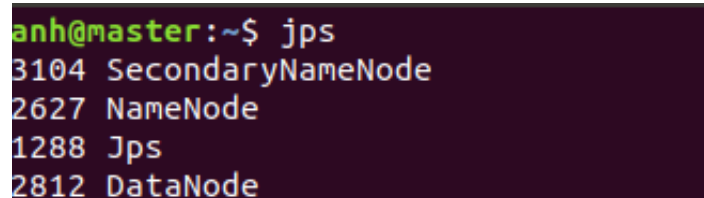
Sau các lệnh này được thực thi xong thì các máy datanodes đã được cài đặt hadoop thành công

- **Cấu hình file `/.bashrc`:** chúng ta có thể copy nội dung từ file `/.bashrc` của master đến các máy datanodes.

3. **Khởi động cụm máy Hadoop:** chạy các lệnh sau để khởi động Hadoop

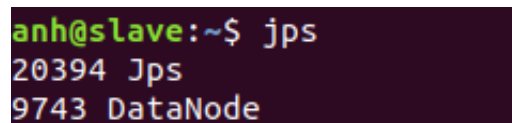
- `bin/hdfs namenode -format`
- `sbin/start-dfs.sh`

4. **Kiểm tra xem Hadoop service:** lần lượt thực thi lệnh `jps` từ terminal của các máy master và datanode để kiểm tra, nếu kết quả như hình dưới đây thì các service hadoop đang hoạt động.



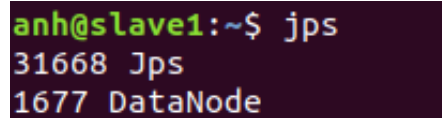
```
anh@master:~$ jps
3104 SecondaryNameNode
2627 NameNode
1288 Jps
2812 DataNode
```

Hình 5.2: Kiểm tra Hadoop service trên master



```
anh@slave:~$ jps
20394 Jps
9743 DataNode
```

Hình 5.3: Kiểm tra Hadoop service trên worker(slave)



```
anh@slave1:~$ jps
31668 Jps
1677 DataNode
```

Hình 5.4: Kiểm tra Hadoop service trên worker(slave1)

#### 5.3.1.4 Thêm một datanodes vào một cụm Hadoop đang hoạt động

Trong quá trình sử dụng, việc muốn mở rộng cụm máy Hadoop là điều có thể xảy ra. Chúng ta có thể dễ dàng thêm một datanodes và một cụm máy Hadoop đang hoạt động theo các bước sau.

1. **Cài đặt Java 8:** đảm bảo rằng máy được thêm vào cụm phải được cài đặt Java 8 như đã trình bày ở phía trên.
2. **Thêm địa chỉ IP vào /etc/hosts:** thêm địa chỉ IP của máy datanode mới vào file /etc/hosts của máy master và các máy datanodes đã có cũng như datanode mới.
3. **Cấu hình SSH:** copy nội dung file /.ssh/id\_rsa.pub vào máy datanodes mới giống như đã làm với các máy datanode đã trước.
4. **Cài đặt Hadoop:** chúng ta cũng sử dụng SSH để gửi thư mục cài Hadoop từ máy master sang máy datanode mới như đã thực hiện.
5. **Cấu hình file workers:** thêm hostname của máy datanode mới vào file workers trên máy master.
6. **Khởi động datanode:** thực hiện file *start-dfs.sh* trong thư mục sbin trên máy master để tiến hành khởi động datanode mới vào cụm máy.

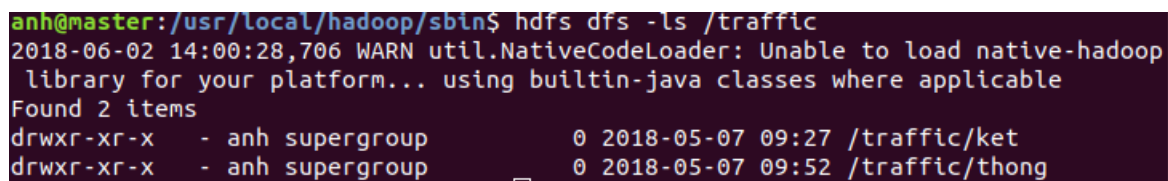


### 5.3.1.5 Lưu trữ dữ liệu ảnh trên HDFS

Để đưa dữ liệu lên HDFS, ta sử dụng câu lệnh sau.

```
hdfs dfs -put ~/Document/traffic/ hdfs://master:9000/
```

Với `/Document/traffic` là đường dẫn đến thư mục lưu dữ liệu ảnh, và `hdfs://master:9000/` là đường dẫn đến vị trí lưu của HDFS. Để xem kết quả put ta thực hiện lệnh sau để liệt kê các thư mục dữ liệu đã được lưu trên HDFS.



```
anh@master:/usr/local/hadoop/sbin$ hdfs dfs -ls /traffic
2018-06-02 14:00:28,706 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
Found 2 items
drwxr-xr-x - anh supergroup          0 2018-05-07 09:27 /traffic/ket
drwxr-xr-x - anh supergroup          0 2018-05-07 09:52 /traffic/thong
```

Hình 5.5: Liệt kê thư mục dữ liệu trên HDFS

## 5.3.2 Cài đặt HIVE

Để cài đặt HIVE và lưu trữ dữ liệu GPS, ta thực hiện các bước sau đây.

### 5.3.2.1 Tải gói cài đặt HIVE

Tải gói cài đặt HIVE ở với đường dẫn sau <http://mirror.downloadvn.com/apache/hive/>, ta sẽ sử dụng phiên bản 2.2.0.

### 5.3.2.2 Cài đặt HIVE

1. **Giải nén:** giải nén gói cài đặt và copy thư mục có được vào vị trí thư mục muốn cài chẳng hạn như `/usr/local/hive`.

2. **Cấu hình biến môi trường:** thêm vào file */.bashrc* nội dung như sau

```
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:/usr/local/Hadoop/lib/*:.
export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:.
```

Thực hiện lệnh *source ~/.bashrc* để thực thi cấu hình.

#### 5.3.2.3 Tải và cài đặt Apache Derby

Làm theo các lệnh hướng dẫn để tải à cài đặt Apache Derby

1. **Tải Apache Derby:** vào và tải ở địa chỉ sau *http://archive.apache.org/dist/db/derby/* với phiên bản 10.14.2.0
2. **Giải nén:** giải nén vào thư mục */usr/local/derby*
3. **Cấu hình biến môi trường cho Derby:** Thêm nội dung file */.bashrc* như sau.

```
export DERBY_HOME=/usr/local/derby
export PATH=$PATH:$DERBY_HOME/bin
```

#### 5.3.2.4 dữ liệu GPS

Dữ liệu GPS là tập dữ liệu lưu vị trí GPS của một xe buýt trong một thời điểm nhất định. Dữ liệu này được lưu bằng bảng trong HIVE. Cấu trúc của bảng dữ liệu bao gồm độ, vĩ độ, thời điểm theo định dạng DD/MM/YYYY hh:mm:ss.

### 5.3.3 Xử lý dữ liệu

Để xây dựng mô hình phân loại hình ảnh, chúng ta cần phải có một tập huấn luyện đủ tốt. Ở đây, các hình ảnh được trích xuất từ camera hành trình từ các tuyến xe buýt.

Cấu trúc tổ chức tập dữ liệu gồm 2 thư mục chính:

- Thư mục **ket**: chứa các hình ảnh được cho là giao thông trong tình trạng ùn tắc. Bao gồm khoảng 7500 hình ảnh định dạng JPG.
- Thư mục **thong**: chứa các hình ảnh được cho là giao thông trong tình trạng thông thoáng. Bao gồm 7700 hình ảnh định dạng JPG.

Các hình ảnh được trình bày và lưu trữ vào các thư mục có chứa các tên mô tả cho đặc tính của những hình ảnh đó. Với bộ dữ liệu trên, chương trình tạo một kiểu dữ liệu dictionary với khóa chính là giá trị biểu diễn cho tên thư mục và cũng là tên class cần phân loại, value chính là đường dẫn các file ảnh tương ứng.

Để sử dụng được trong mô hình mạng, các hình ảnh sẽ được mã hóa sang một định dạng mới nhờ các phương thức hỗ trợ có sẵn trong thư viện Tensorflow. Sau khi được mã hóa, kết quả chính là các tensor có thông số shape như sau  $[299, 299, 3]$ , với hai vị trí đầu tiên chính là kích thước của hình ảnh cũng như của tensor, giá trị 3 biểu diễn cho độ sâu (ảnh màu).

Sau cùng, bộ dữ liệu đã được mã hóa được chia thành 3 tập con sử dụng với 3 mục đích khác nhau: tập huấn luyện (Training set), tập validation để tránh vấn đề overfit trong quá trình huấn luyện và tập kiểm thử dùng để kiểm tra độ chính xác của mô hình sau khi huấn luyện hoàn tất. Riêng tập huấn luyện sẽ được dùng để tạo ra các bottlenecks

### 5.3.4 Tạo các bottlenecks

Bottlenecks[15] là một từ được dùng để chỉ tầng (layer) nằm ngay trước fully-connected layer. Với kiến trúc mạng googLeNet, tầng này đã được huấn luyện tập dữ liệu trước đó nên có được kết quả đủ tốt để phân biệt được đặc tính của mỗi lớp(class) yêu cầu. Có nghĩa ở bước này chúng ta sẽ tạo ra một bản tóm tắt các giá trị trọng số đủ tốt cho mỗi ảnh input. Tầng cuối cùng của kiến trúc mạng sẽ sử dụng các giá trị bottlenecks này để huấn luyện và điều chỉnh để phân loại các lớp mới. Điều này nhờ vào việc mạng đã được huấn luyện bởi tập dữ liệu gồm 1000 lớp khác nhau của ImageNet giúp cho việc phát hiện các mẫu đặc tính trở nên dễ dàng hơn.

### 5.3.5 Huấn luyện

Sau khi hoàn tất tạo các giá trị bottleneck, việc thực hiện thực hiện cấu hình mạng và huấn luyện bắt đầu. Tổng số bước huấn luyện sẽ được cài đặt mặc định là 4000 bước, tuy nhiên có thể thay đổi lại tùy theo tình huống. Mỗi bước huấn luyện sẽ chọn ra 100 dữ liệu ngẫu nhiên <sup>1</sup> để đưa vào tầng cuối cùng <sup>2</sup> để dự đoán lớp, lớp dự đoán sẽ được so sánh với các lớp thực tế để mạng điều chỉnh và cập nhật các giá trị trọng số thông qua cơ chế lan truyền ngược như đã trình bày ở chương trước. Do phép toán được thực hiện trên tập huấn luyện nên sẽ gây ra vấn đề overfit, vì thế mà tập validation sẽ được sử dụng để đo lại giá trị sai lệch và độ chính xác. Nếu độ chính xác tại tập huấn luyện cao nhưng tại tập validation không thay đổi hoặc thấp thì chứng tỏ mô hình mạng gặp phải vấn đề overfit và việc huấn luyện tiếp tục

---

<sup>1</sup>Tập dữ liệu lúc này là những bottlenecks

<sup>2</sup>Tầng fully-connected với softmax là activation function

không còn có ích.

### 5.3.6 Chạy Huấn luyện bằng cách kết hợp tensorflow với hệ thống HDFS

#### 5.3.6.1 Chuẩn bị

Ở các nội dung trên, chúng ta đã thực hiện việc cài đặt Apache Hadoop cũng như các biến môi trường cần thiết, trong đó có hai biến môi trường cần thiết để chạy thực thi mô hình trên là **JAVA\_HOME** và vị trí cài đặt Hadoop **HADOOP\_HDFS\_HOME**. Bây giờ chúng ta sẽ cần thêm một biến môi trường cho hệ thống nữa.

- **LD\_LIBRARY\_PATH**: để thêm biến này, chúng ta cần thêm vào file `/.bashrc` nội dung như sau.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${JAVA_HOME}/jre/lib/amd64/server
```

Hình 5.6: Nội dung thư viện LD\_LIBRARY\_PATH

#### 5.3.6.2 Chạy mô hình huấn luyện.

Chúng ta thực hiện câu lệnh sau đây để tiến hành chạy mô hình huấn luyện kết nối với tập dữ liệu ở HDFS.

```
CLASSPATH=$((${HADOOP_HDFS_HOME}/bin/hadoop classpath --glob) python model.py \
--image_dir=hdfs://master:9000/traffic/
```

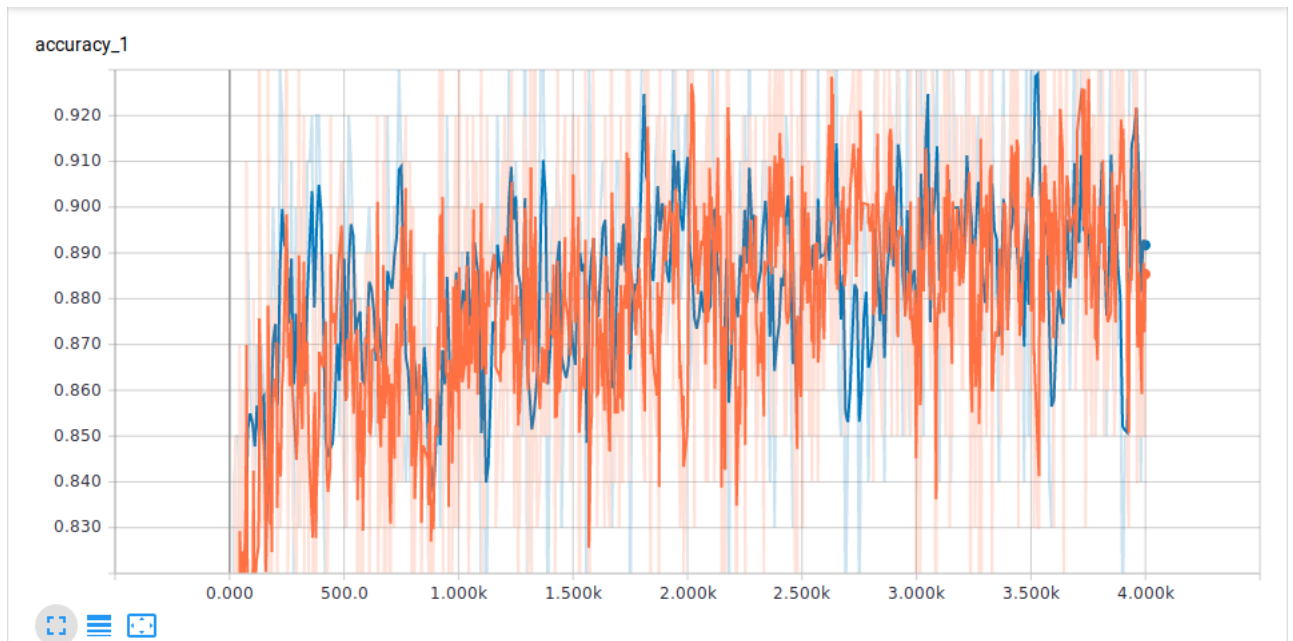
Hình 5.7: Câu lệnh thực thi huấn luyện mô hình

## Chương 6

# Kết quả huấn luyện và đánh giá

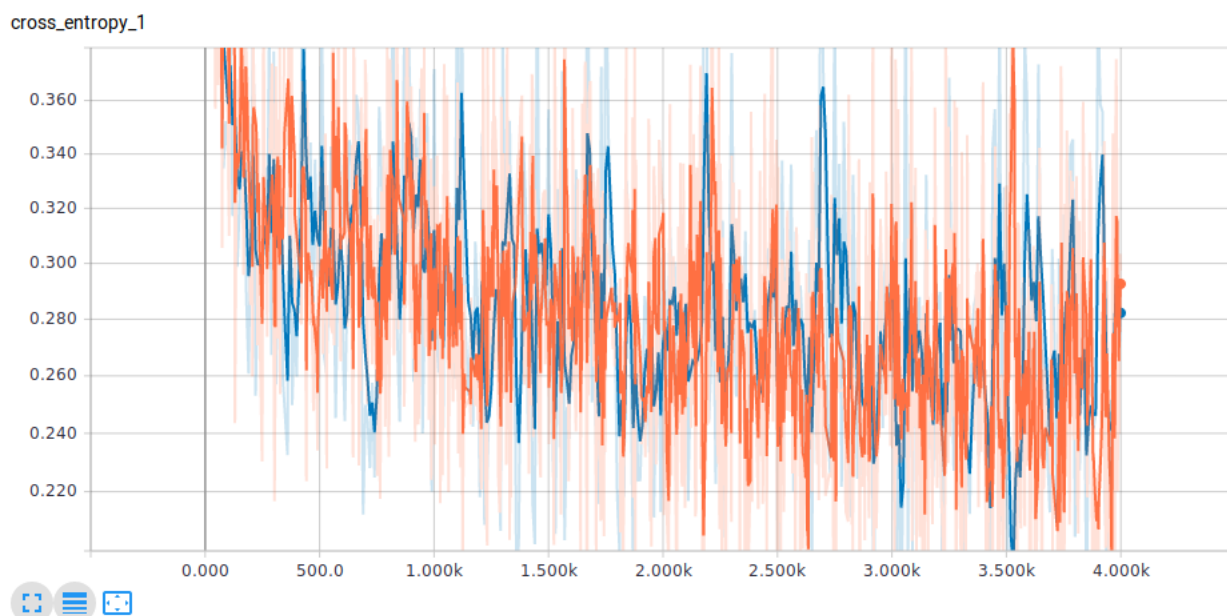
### 6.1 Kết quả huấn luyện

Với các thông số mạng đã cấu hình ở chương trước, việc huấn luyện mạng mang lại kết quả tương đối tốt với giá trị  $cross\_entropy = 0.29$  và độ chính xác ở tập kiểm thử là 88.7%. Không những thế việc áp dụng kỹ thuật transfer learning giúp tiết kiệm thời gian huấn luyện gấp nhiều lần. Cụ thể, việc xây dựng mô hình thủ công và huấn luyện lại từ đầu đối với máy tính không hỗ trợ card GPU sẽ mất thời gian là 8 tiếng nhưng đối với việc áp dụng mô hình inception model với kỹ thuật transfer learning thì chỉ mất từ 45 đến 60 phút để hoàn thành.



Hình 6.1: Đồ thị biểu diễn độ chính xác (accuracy)

Đồ thị 6.1 biểu diễn chi tiết độ chính xác trong quá trình huấn luyện với đường màu cam biểu diễn cho quá trình huấn luyện và đường màu xanh lam biểu diễn cho kiểm tra trên tập Validation. Tương tự với 6.2.



Hình 6.2: Đồ thị biểu diễn độ sai số (cross entropy)

## 6.2 Sử dụng mô hình phân loại ảnh mới

Sử dụng mô hình để phân loại một số hình ảnh khác chưa được phân loại. Hình 6.3 có thể nhận biết bằng mắt thường là hình ảnh của những con đường đang trong tình trạng ùn tắc. Khi đưa ảnh vào mô hình, ta thu được kết quả phân loại với hai giá trị *ket* và *thong* lần lượt là 0.9910778 và 0.0089. Các con số này mang ý nghĩa bộ phân lớp cho rằng tám ảnh trên có khoảng 99% là thuộc về lớp kẹt xe và xấp xỉ 1% là thuộc về lớp thông thoáng.





Hình 6.3: Ảnh kiểm thử 1 - kết quả

Tương tự đối với hình 6.4, cũng là một hình ảnh thể hiện tình trạng giao thông đang bị ùn ứ, khó khăn trong việc di chuyển. Mô hình đã đưa ra kết quả phân loại với hai giá trị *ket* và *thong* lần lượt là 0.9616194 và 0.038380623. Tức là 96 % thuộc về lớp kẹt xe, 3.8% là thuộc về lớp thông thoáng



ket 0.9616194  
thong 0.038380623

Hình 6.4: Ảnh kiểm thử 2 - kết quả

Đối với hai hình ảnh tiếp theo được phân biệt vào loại đường thông thoáng một cách dễ dàng. Mô hình dự đoán xác suất hai hình ảnh dưới đây thuộc lớp thông thoáng lần lượt là.

Ở hình 6.5, khi nhìn vào camera ta có thể nhận biết được vào thời điểm này, các phương tiện trên đường rất ít, di chuyển rất dễ dàng, không xảy ra tình trạng kẹt xe hay ùn tắc nào. Mô hình cũng đã đưa ra kết quả với hai giá trị *ket* và *thong* lần lượt là 0.005655429 và 0.9943446. Tức là 99.4 % thuộc về lớp thông thoáng, xấp xỉ 0.6 % thuộc về lớp kẹt xe.



Hình 6.5: Ảnh kiểm thử 3 - kết quả

Tương tự với hình 6.6, ta nhận thấy trong hình ảnh này, các phương tiện đang cách xe buýt một khoảng tương đối xa, mặt đường ở giữa khá thông thoáng để di chuyển. Từ đó, mô hình đã đưa ra kết quả với hai giá trị *ket*

và *thong* lần lượt là 0.10274496 và 0.89725506. Tức là 89.7 % thuộc về lớp thông thoáng, và xấp xỉ 10.3 % thuộc về lớp kẹt xe.



Hình 6.6: Ảnh kiểm thử 4 - kết quả

Như vậy, kết quả kiểm thử đối với một số hình ảnh chưa được phân loại của mô hình đã huấn luyện cho kết quả khá chính xác. Tuy nhiên, đối với vấn đề phân loại giao thông thì không chỉ có hai trường hợp ùn tắc hay thông thoáng mà còn tồn tại nhiều trường hợp hơn. Như tình huống đông xe nhưng di chuyển chậm, trên thực tế đây là tình huống không phải ùn tắc nhưng ảnh chụp gần giống với ảnh ùn tắc. Cần chú ý vấn đề này khi lựa chọn, phân loại ảnh huấn luyện, hoặc để giải quyết tốt hơn cần phải tăng số lượng lớp(nhãn) cần phân loại lên thành 3 hay lớn hơn thay vì 2 như ban đầu.

# Chương 7

## Kết luận

### 7.1 Các kết quả

Luận văn này giúp nghiên cứu, tìm hiểu về mạng học sâu cũng như việc phân loại ảnh giao thông. Các kết quả đã đạt được đáp ứng được các mục tiêu đã đặt ra.

- Xây dựng được mô hình HDFS sử dụng Apache Hadoop có thể lưu trữ dữ liệu video tập dữ liệu ảnh
- Xây dựng được bộ dữ liệu phục vụ cho việc huấn luyện mô hình phân biệt 2 loại ảnh giao thông. Số lượng ảnh thu thập được trung bình mỗi lớp khoảng 1000 cho đến 2000 ảnh.
- Huấn luyện thành công mô hình học sâu GoogleNet sử dụng tập dữ liệu được lưu trữ tư hệ thống HDFS.
- Xác định được vị trí và cho kết quả tình hình giao thông.

## 7.2 Hướng phát triển

Sau khi đạt được kết quả huấn luyện khá tốt, hướng phát triển của đề tài trong tương lai như sau:

1. Tiếp tục xem xét việc xác định các lớp ảnh giả mạo thông trong tương lai giúp phát hiện nhiều loại hình như ùn tắc, thông thoáng, đông xe di chuyển chậm,.v.v. giúp cụ thể hóa tình trạng giao thông.
2. Xây dựng trang web giúp nhận biết kẹt xe.
3. Có thể sử dụng nền tảng đã xây dựng để áp dụng cho nhiều bài toán phân loại ảnh khác.

Hệ thống tiềm năng trên có thể giúp dân cư sinh sống ở các khu đô thị cũng như ban quản lý nắm bắt tình hình giao thông để phân luồng di chuyển và khắc phục một cách nhanh nhất.

# Tài liệu tham khảo

- [1] Q. J. at English Wikipedia, “Neuron image.”
- [2] Chrislb, “Perceptron image.”
- [3] Chrislb, “Multilayer neural network.”
- [4] M. Nielsen, “Convolutional layer,”
- [5] S. Raval, “Image for convolution operation.”
- [6] Aphex34, “Image for max pooling.”
- [7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *CoRR*, vol. abs/1409.4842, 2014.
- [8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ch. 6,9. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [9] A. Karpathy, “ CS231n: Convolutional Neural Networks for Visual Recognition,” 2016.
- [10] A. Karpathy, “ CS231n: Convolutional Neural Networks for Visual Recognition,” 2016.



- [11] A. Karpathy, “ CS231n: Convolutional Neural Networks for Visual Recognition,” 2016.
- [12] T. White, *Hadoop: The Definitive Guide*, ch. 3, pp. 43–54. O’reilly Media.
- [13] M. Lin, Q. Chen, and S. Yan, “Network in network,” *CoRR*, vol. abs/1312.4400, 2013.
- [14] tensorflow.org, “Tensorflow programmer guide,”
- [15] tensorflow.org, “ Image retraining,”
- [16] R. Cadène, N. Thome, and M. Cord, “Master’s thesis : Deep learning for visual recognition,” *CoRR*, vol. abs/1610.05567, 2016.
- [17] A. Karpathy, “ CS231n: Convolutional Neural Networks for Visual Recognition,” 2016.
- [18] A. Karpathy, “ CS231n: Convolutional Neural Networks for Visual Recognition,” 2016.
- [19] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” *CoRR*, vol. abs/1311.2901, 2013.
- [20] Y. Gao and J. Dong, “A Deep Convolutional Network for Traffic Congestion Classification,” pp. 1–11.