

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

1	Giới thiệu	7
1.1	Tổng quan	7
1.2	Mục tiêu đề tài	8
2	Mạng neural và Mạng neural tích chập (Convolutional Neural Networks CNN	10
2.1	Mạng neural (Neural Network)	10
2.1.1	Mạng neural (Neural Network)	10
2.1.2	Cấu trúc mạng neural	11
2.1.3	Mô hình Feedforward Neural Network	15
2.2	Mạng neural tích chập (Convolutional neural network)	19
2.2.1	Giới thiệu mạng neural tích chập	19
2.2.2	Mô hình mạng neural tích chập	19
2.2.3	Kiến trúc mạng GoogLeNet)	27
3	Apache Hadoop	30
3.1	Tổng quan về hệ thống Apache Hadoop	30
3.2	Hadoop distributed file system - HDFS	31

3.2.1	Thiết kế	31
3.2.2	Các khái niệm trong HDFS	31
3.3	Tại sao sử dụng Apache Hadoop trong đề tài?	33
4	Mô hình loại ảnh giao thông kết hợp Apache Hadoop	34
4.1	Vấn đề và mục tiêu	34
4.2	Phương pháp transfer learning	34
4.3	Các bước chuẩn bị	35
4.3.1	Tổng quan về tập dữ liệu	35
4.3.2	Môi trường	36
4.4	Các bước hiện thực	38
4.4.1	Cài đặt Apache Hadoop	38
4.4.2	Xử lý dữ liệu	45
4.4.3	Tạo các bottlenecks	46
4.4.4	Huấn luyện	46
5	Kết quả huấn luyện và đánh giá	48
5.1	Kết quả huấn luyện	48
5.2	Sử dụng mô hình phân loại ảnh mới	48
6	Kết luận	53
6.1	Các kết quả	53
6.2	Hướng phát triển	54

Danh sách hình vẽ

2.1	Tế bào thần kinh neuron sinh học	11
2.2	Cấu trúc cơ bản mạng neuron	12
2.3	Đồ thị hàm step	14
2.4	Đồ thị hàm sigmoid	14
2.5	Đồ thị hàm Tanh	15
2.6	Ví dụ kiến trúc mạng feedforward	16
2.7	MLP	17
2.8	Convolutional layer	21
2.9	Convolutional layer	21
2.10	Mảng các giá trị của bộ lọc	22
2.11	ảnh đầu vào	23
2.12	phép toán tích chập	23
2.13	phép toán tích chập	24
2.14	kết quả tầng tích chập	24
2.15	max-pooling 2 x 2	25
2.16	Ví dụ tầng tổng hợp	26
2.17	GoogLeNet	27
2.18	inception module	28

2.19	naive inception module	28
4.1	Kiểm tra Hadoop service trên master	44
4.2	Kiểm tra Hadoop service trên worker(slave)	45
4.3	Kiểm tra Hadoop service trên worker(slave1)	45
5.1	Kết quả huấn luyện	49
5.2	Ảnh kiểm thử 1 - kết quả	50
5.3	Ảnh kiểm thử 2 - kết quả	51
5.4	Ảnh kiểm thử 3 - kết quả	52
5.5	Ảnh kiểm thử 4 - kết quả	52

Danh sách bảng

4.1	Các chiều dữ liệu.	36
4.2	Một vài kiểu dữ liệu.	37

Listings

4.1	Nội dung file <code>/etc/hosts</code>	39
4.2	Cài đặt Java 8	39
4.3	Cài đặt SSH	40
4.4	Kết nối SSH	40

Chương 1

Giới thiệu

1.1 Tổng quan

Với sự phát triển nhanh chóng của nước ta, với cơ sở hạ tầng giao thông còn chưa hoàn thiện nên nạn ùn tắc giao thông là điều không thể tránh khỏi. Các công tác nhận biết và xử lý các chót ùn tắc tương đối chậm và chưa hiệu quả. Nạn ùn tắc giao thông có thể là nguyên nhân chính trong việc kiềm hãm sự phát triển.

Trong những năm gần đây, sự bùng nổ kỹ thuật số và thông tin cho thấy những công trình nghiên cứu mang tính ứng dụng cao trong các lĩnh vực trí tuệ nhân tạo. Điển hình là chủ đề học máy, học sâu, sự phát triển mạnh của hai chủ đề này khiến cho vấn đề phân loại hình ảnh đạt tới một cột mốc mới. Đề tài "Phát hiện kẹt xe qua camera hành trình" kỳ vọng có thể áp dụng mô hình học sâu để nhận biết tình trạng giao thông.

1.2 Mục tiêu đề tài

Đối với đề tài này, giai đoạn đề cương sẽ tập trung tìm hiểu các kiến thức về học sâu, mạng neuron và các khái niệm cơ bản, mong muốn xây dựng được mô hình có khả năng phân loại hình ảnh giao thông.

Các công việc thực hiện:

- Tìm hiểu các kiến thức về học sâu nói chung, mạng neuron nói riêng và ứng dụng.
- Xây dựng được bộ dữ liệu phục vụ cho việc huấn luyện.
- Cấu hình, sử dụng kiến trúc mạng có sẵn để huấn luyện.

CÁC CÂU HỎI CẦN TRẢ LỜI KHI VIẾT CHƯƠNG GIỚI THIỆU.

—TỔNG QUAN—

–Trình bày rõ vấn nạn hiện nay về giao thông như thế nào, cơ sở hạ tầng và cách giải quyết hiện nay ra sao.

–Tiếp theo trình bày ngày nay, dữ liệu liên tục được sinh ra ra sao?, với nguồn dữ liệu như vậy thì có lợi ích gì? Big data giúp ích gì cho cuộc sống và các vấn đề xã hội? Các công cụ big data ra sao?

–Nói nói làm sao đó chuyển qua machine learning,deep learning phát triển ra sao, các mô hình học sâu liên tục được đưa ra như thế nào, độ chính xác ra sao, có dễ dàng áp dụng thực tiễn ko?

–Rồi nêu lên quan điểm của bạn như thế nào nếu học sâu kết hợp vs big data có khả thi hay ko. khả năng áp dụng thực tế như nào? Có ai đã áp dụng kết hợp DEep learning và big data hay chưa.?

–Rồi lái tới ý tưởng của đề tài -> đề tài này ra sao, có ứng dụng được không?

tính khả quan của đề tài, hạn chế và ưu điểm của đề tài?

—MỤC TIÊU ĐỀ TÀI— Chém ra vài cái mục tiêu đi...=))))))

Chương 2

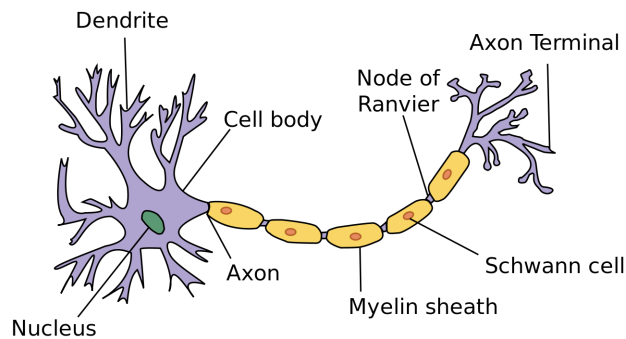
Mạng neural và Mạng neural tích chập (Convolutional Neural Networks CNN

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

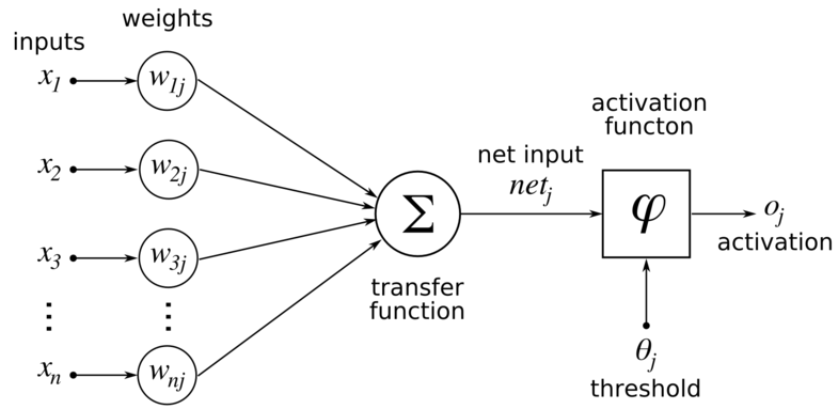
nhất thông qua việc trao đổi thông tin nhờ các liên kết.

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

- **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.

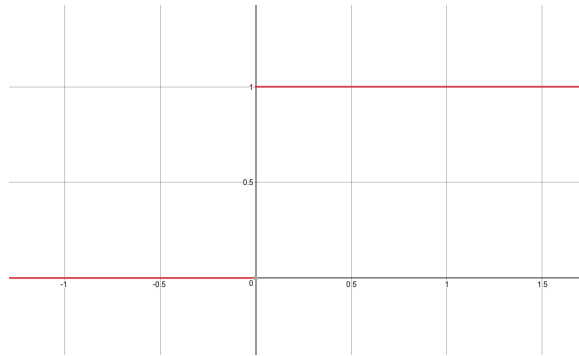
$$\sum input_j * w_{ij}$$

- **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:*

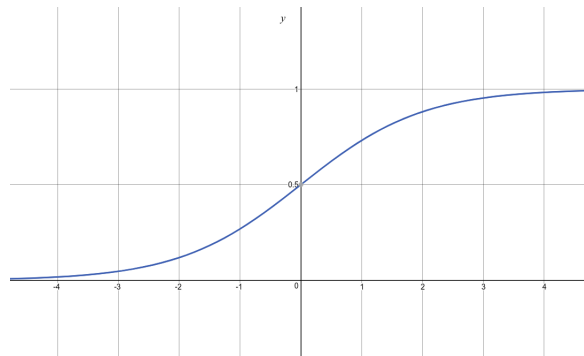
$$\begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}$$



Hình 2.3: Đồ thị hàm step

– *sigmoid function:*

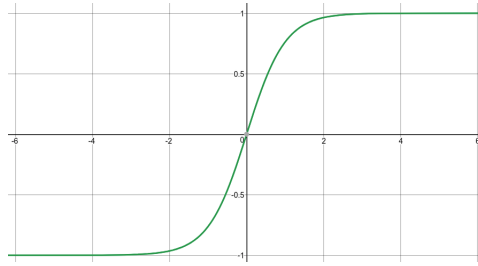
$$A(x) = \frac{1}{1 + e^{-x}}$$



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

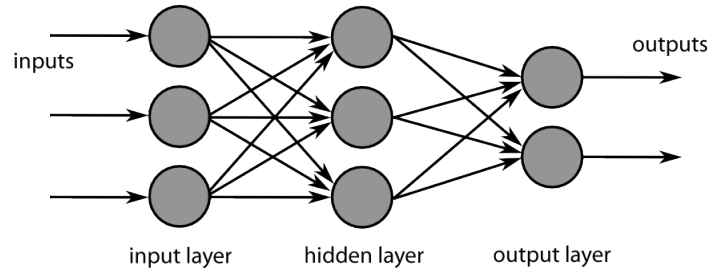
2.1.3 Mô hình Feedforward Neural Network

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.



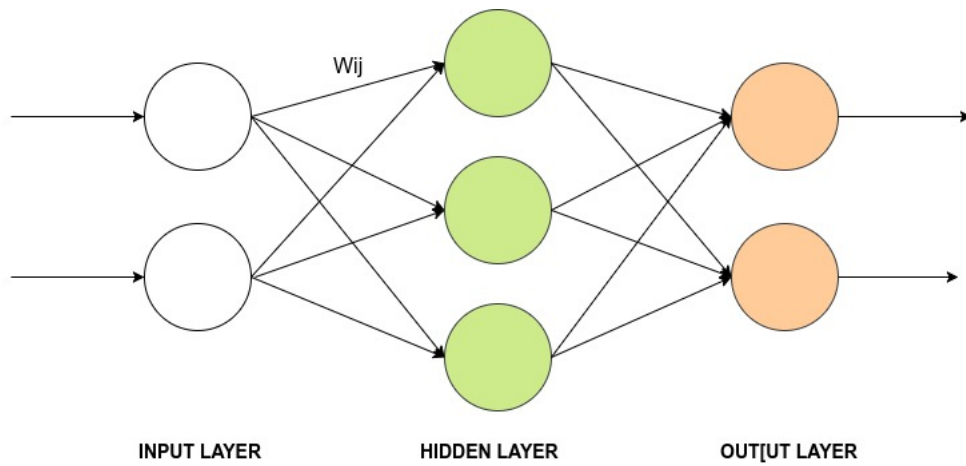
Hình 2.6: Ví dụ kiến trúc mạng feedforward

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[2] 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 .
- θ_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 ϵ .

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[3]. 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.

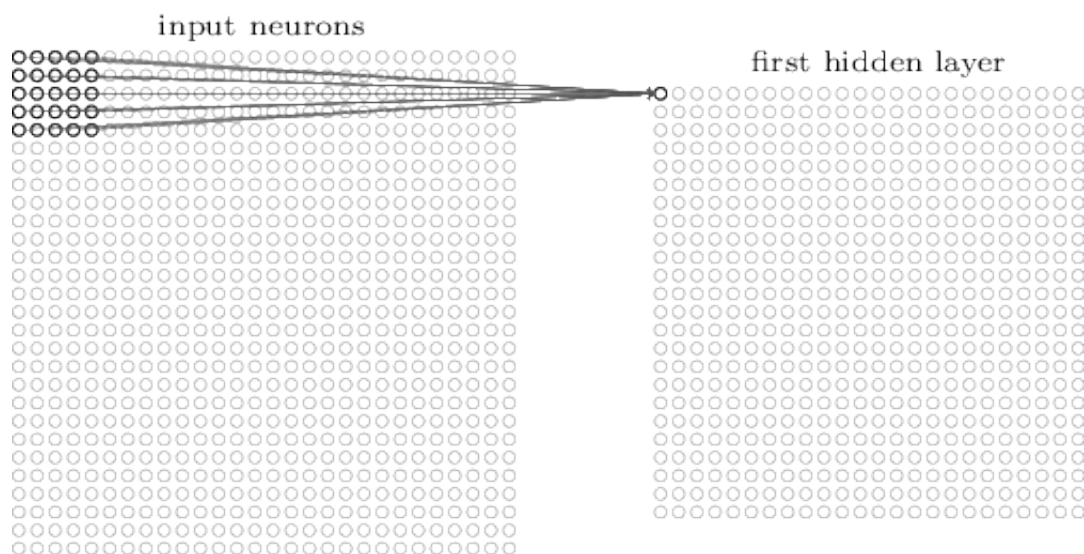
Đối với output, đây là một mảng các giá trị xác suất, mảng giá trị này cũng tùy thuộc vào số lượng nhãn(lớp) cần dự đoán. Ví dụ, (0.90 cho xe ô tô, 0.1 cho xe tải).

2.2.2.2 Convolutional layer

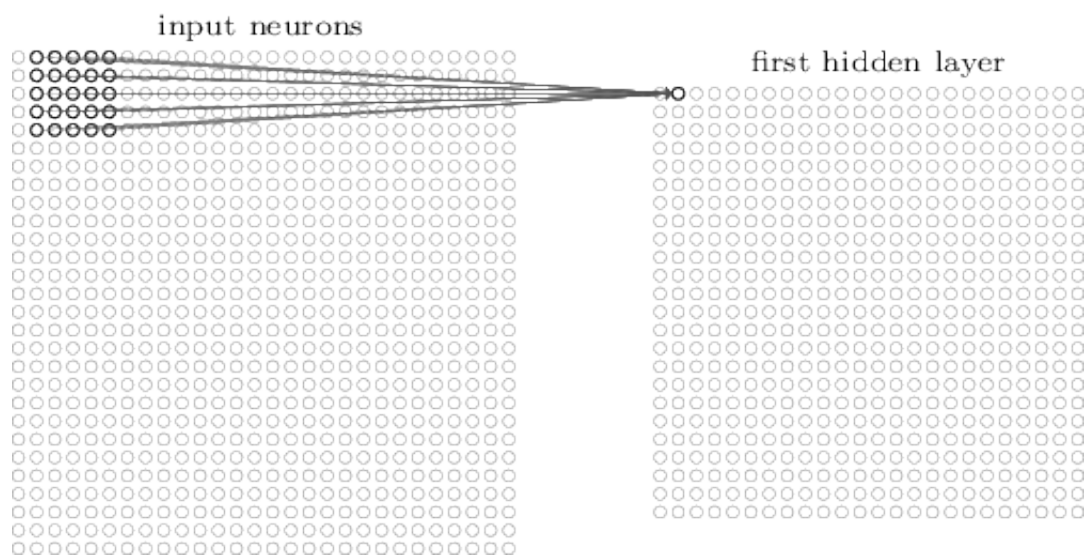
Tầng đầu tiên của một mạng CNN luôn luôn là tầng tích chập (convolutional layer)[4]. Như đã biết đầu vào (input) là một mảng các giá trị pixel. Trong trường hợp cụ thể để dễ hình dung ta chọn input là mảng các giá trị pixel có kích thước 32 x 32 x 3 với 32 x 32 là chiều dài và chiều rộng của tấm hình 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[5]. 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.

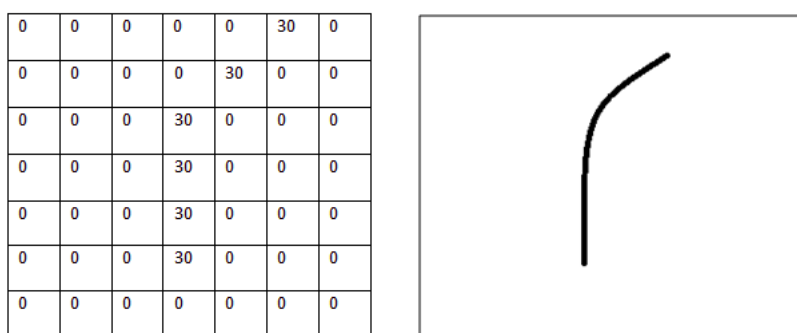


Hình 2.8: Convolutional layer



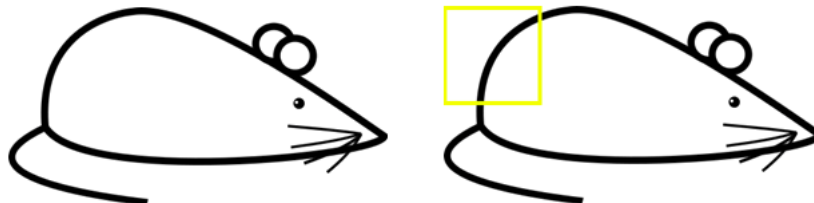
Hình 2.9: Convolutional layer

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.

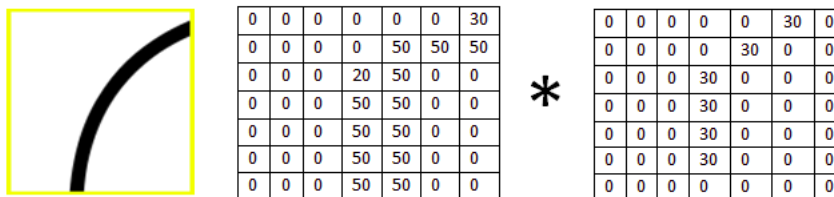


Hình 2.10: Mảng các giá trị của bộ lọc

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



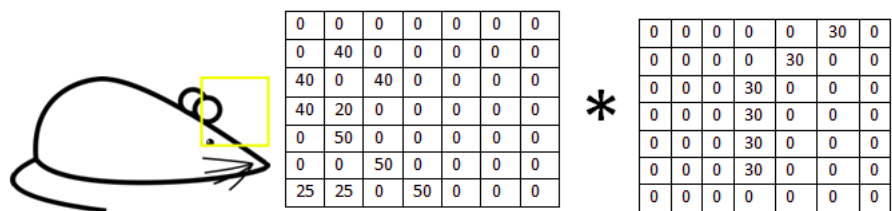
Hình 2.12: phép toán tích chập

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

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

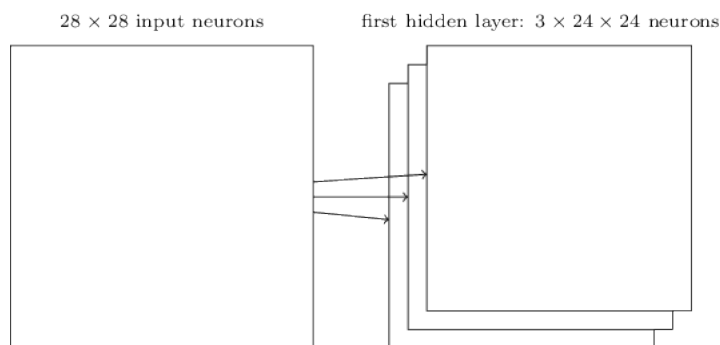
Đây là một con số rất lớn, thông thường nếu bộ lọc trượt tới một vùng mà vùng đó có hình dạng tương tự như bộ lọc thì kết quả khi thực hiện phép tính là một con số rất lớn. Ngược lại, kết quả sẽ ra rất nhỏ hoặc bằng 0. Ví dụ là hình ảnh [2.13](#) kết quả sẽ là 0 khi thực hiện phép tính

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

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.

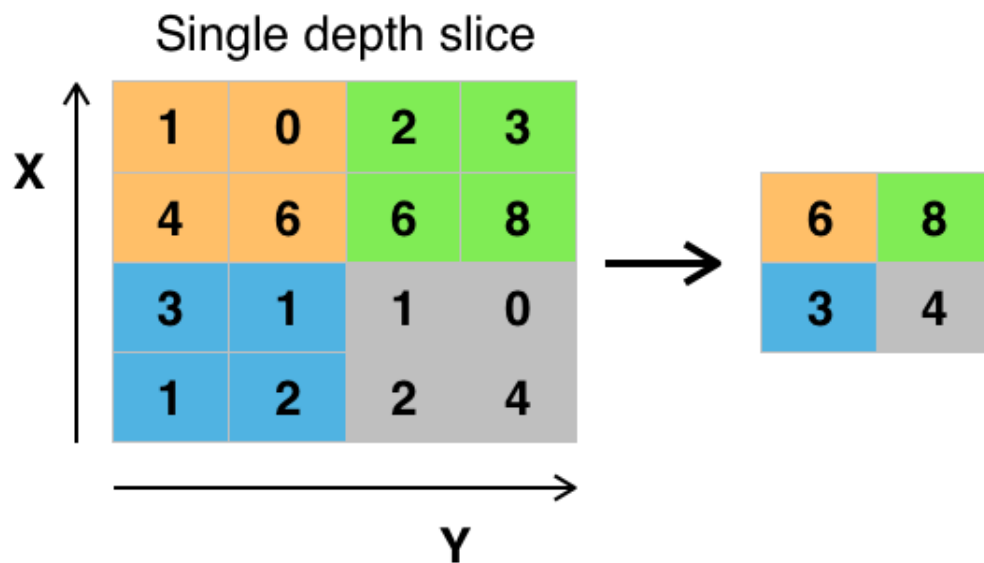


Hình 2.14: kết quả tầng tích chập

2.2.2.3 Pooling Layer

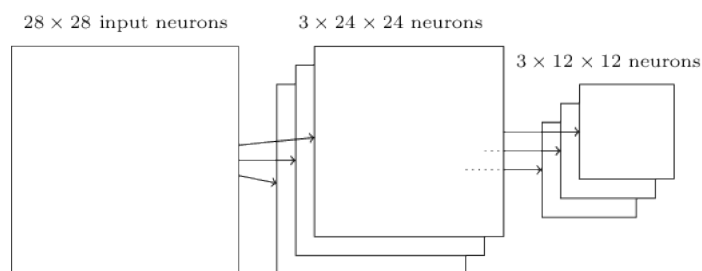
Sau khi qua một hoặc vài tầng tích chập, CNN sẽ chứa các tầng tổng hợp (Pooling Layer) ngay sau đó. Ở đây, tầng tổng hợp sẽ đơn giản hóa thông tin được lấy từ đầu ra từ tầng tích chập trước đó. Có nhiều kiểu tổng hợp

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 2×2 . Giá trị lớn nhất trong mỗi vùng được trượt qua sẽ được chọn làm kết quả xuất ra.



Hình 2.15: max-pooling 2×2

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

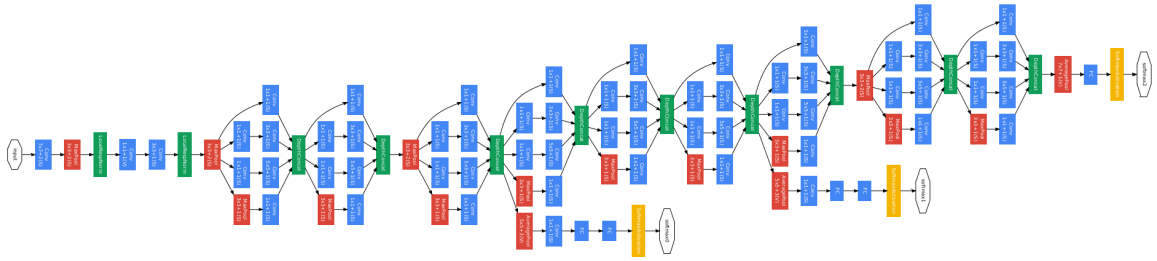
2.2.2.4 Fully Connected Layer

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.

2.2.3 Kiến trúc mạng GoogLeNet)

2.2.3.1 Ý tưởng

Đây là kiến trúc mạng tích chập với 22 tầng. GoogLeNet còn là quán quân của ILSVRC 2014 [9]. 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. 2.17 minh họa cho cấu trúc của mạng googLeNet.

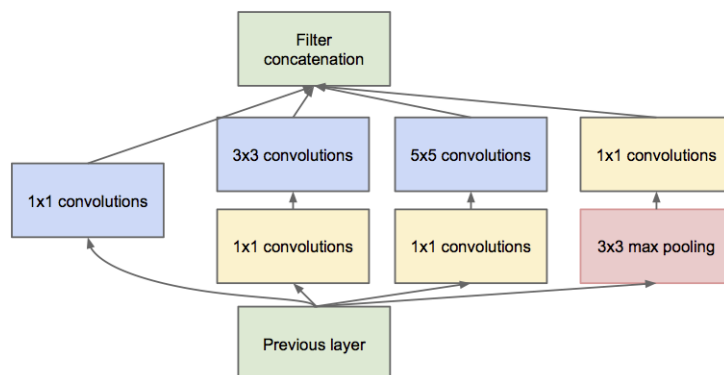


Hình 2.17: GoogLeNet

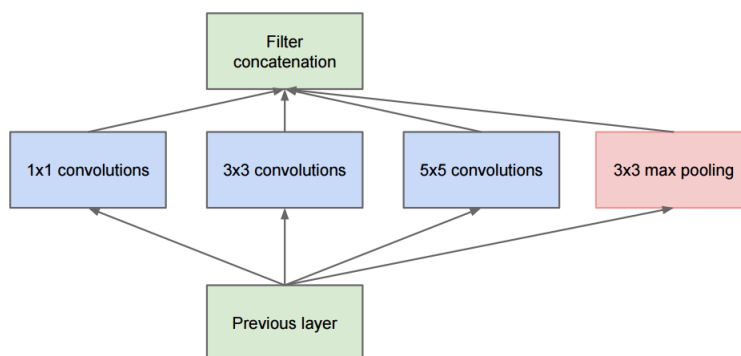
2.2.3.2 Inception module

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 [9].



Hình 2.18: inception module



Hình 2.19: naive inception module

Hình 2.18 miêu tả cấu trúc của một module trong mạng. Trong khi đó 2.19 là một ý tưởng ban đầu mà tác giả đã nghĩ tới. Ở 2.18 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[8], điều này giúp cho khối lượng tham số

phải tính toán ở phép toán tích chập với bộ lọc 3×3 và 5×5 sẽ được giảm đi một cách đáng kể.

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 ¹. 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,

¹các 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

3.3 Tại sao sử dụng Apache Hadoop trong đề tài?

Chương 4

Mô hình loại ảnh giao thông kết hợp Apache Hadoop

4.1 Vấn đề và mục tiêu

Đối với vấn đề phát hiện kẹt xe qua hình ảnh camera, chúng ta sẽ sử dụng mạng neuron tích chập với tập ảnh huấn luyện được trích xuất từ những đoạn video do camera hành trình xe buýt ghi lại trong thời gian hoạt động. Ngoài ra, chúng ta còn sử dụng mô hình googLeNet để áp dụng vào vấn đề và phương pháp transfer learning (sẽ được trình bày trong mục tiếp theo).

4.2 Phương pháp transfer learning

Ngày nay, với sự phát triển của Deep learning do nguồn dữ liệu to lớn và các máy tính ngày càng cải tiến về khả năng tính toán khiến cho kết quả chính xác của các bài toán phân loại ngày càng cao. Như mô hình mạng

googLeNet có rất nhiều tầng khiến cho việc huấn luyện tốn kém thời gian. Thay vì như vậy chúng ta áp dụng phương pháp transfer learning.

Transfer learning là công đoạn lấy model đã được huấn luyện từ một tập dữ liệu khác [6] và tích hợp lại với tập dữ liệu có đang có. Việc này là khả thi do mô hình đã huấn luyện trước đó sử dụng tập ảnh khổng lồ giúp cho mô hình học được loạt những đặc tính thường thấy trong cùng một ảnh. Có thể thấy tất cả các ảnh đều có những đặc tính cơ bản giống nhau, khi muốn huấn luyện lại cho tập ảnh của chính mình chúng ta chỉ cần thay tầng cuối cùng của mạng bằng tập dữ liệu của mình. Mô hình sẽ tự điều chỉnh lại các trọng số cũng như độ lệch từ các giá trị từ mô hình đã huấn luyện với tập dữ liệu mới mà không cần làm lại từ đầu.

4.3 Các bước chuẩn bị

4.3.1 Tổng quan về tập 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 1077 hình ảnh định dạng JPG.
- Thư mục **thong**: chứa các hình ảnh được cho là giao thông trong tình trạng thông thoáng. Bao gồm 2000 hình ảnh định dạng JPG.

4.3.2 Môi trường

Bộ phân loại ảnh giao hông 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[11] 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 4.1: 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 4.2: Một vài kiểu dữ liệu.

4.4 Các bước hiện thực

4.4.1 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.

4.4.1.1 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](#)

4.4.1.2 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 4.1: 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 4.2: 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 4.3: 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 4.4: 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>
<property>
<name>yarn.resourcemanager.resource-tracker.address</name>
<value>master:8025</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

```

4.4.1.3 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
```

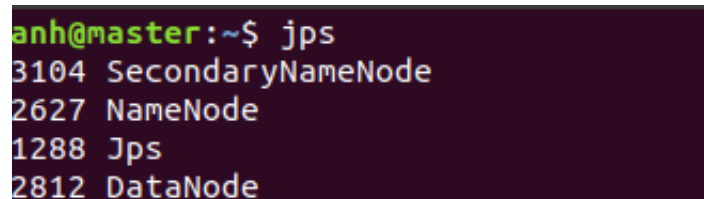
Sau các lệnh này được thực thi xong thì các máy datanodes đã được cài đặt hadoop thành công

- **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 4.1: Kiểm tra Hadoop service trên master

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

Hình 4.2: Kiểm tra Hadoop service trên worker(slave)

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

Hình 4.3: Kiểm tra Hadoop service trên worker(slave1)

4.4.2 Xử lý dữ liệu

Các hình ảnh được trình bày, dữ liệu được lưu vào các thư mục có chứa các tên mô tả cho đặt 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

4.4.3 Tạo các bottlenecks

Bottlenecks[10] 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.

4.4.4 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 ¹ để đưa vào tầng cuối cùng ² 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 chúng ta mô hình mạng gặp phải vấn đề overfit và việc huấn luyện tiếp tục

¹Tập dữ liệu lúc này là những bottlenecks

²Tầng fully-connected với softmax là activation function

không còn có ích.

Chương 5

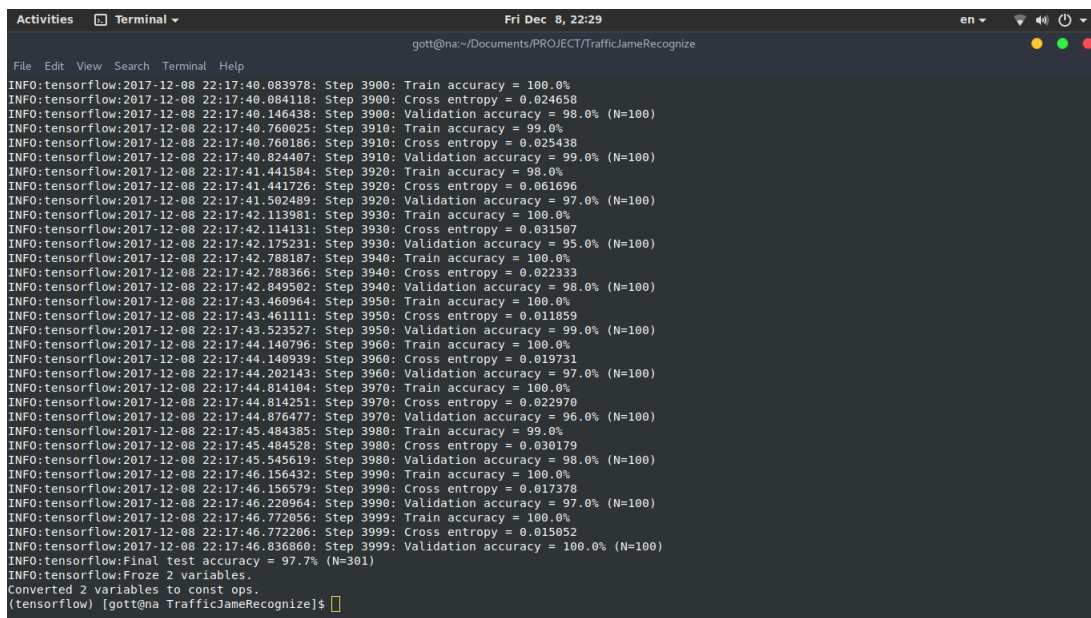
Kết quả huấn luyện và đánh giá

5.1 Kết quả huấn luyện

Với các thông số cà mạng đã cấu hình ở chương trước, [5.1](#) cho thấy kết quả huấn sau khi huấn luyện mạng có kết quả tương đối tốt với giá trị $cross_entropy = 0.015052$ và độ chính xác ở tập kiểm thử là 97.7%. Không những thế việc áp dụng kỹ thuật transfer learning giúp tối 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.

5.2 Sử dụng mô hình phân loại ảnh mới

Sử dụng mô hình để phân loại một số hình ảnh khác chưa được phân loại. [5.2](#) và [5.3](#) có thể nhận biết bằng mắt thường đây là hình ảnh của những



```
Activities Terminal Fri Dec 8, 22:29
gott@na:~/Documents/PROJECT/TrafficJameRecognize

File Edit View Search Terminal Help

INFO:tensorflow:2017-12-08 22:17:40.083978: Step 3900: Train accuracy = 100.0%
INFO:tensorflow:2017-12-08 22:17:40.084118: Step 3900: Cross entropy = 0.024658
INFO:tensorflow:2017-12-08 22:17:40.146438: Step 3900: Validation accuracy = 98.0% (N=100)
INFO:tensorflow:2017-12-08 22:17:40.760025: Step 3910: Train accuracy = 99.0%
INFO:tensorflow:2017-12-08 22:17:40.760186: Step 3910: Cross entropy = 0.025438
INFO:tensorflow:2017-12-08 22:17:40.824407: Step 3910: Validation accuracy = 99.0% (N=100)
INFO:tensorflow:2017-12-08 22:17:41.441504: Step 3920: Train accuracy = 99.0%
INFO:tensorflow:2017-12-08 22:17:41.441726: Step 3920: Cross entropy = 0.061696
INFO:tensorflow:2017-12-08 22:17:41.502489: Step 3920: Validation accuracy = 97.0% (N=100)
INFO:tensorflow:2017-12-08 22:17:42.113981: Step 3930: Train accuracy = 100.0%
INFO:tensorflow:2017-12-08 22:17:42.114131: Step 3930: Cross entropy = 0.031507
INFO:tensorflow:2017-12-08 22:17:42.175231: Step 3930: Validation accuracy = 95.0% (N=100)
INFO:tensorflow:2017-12-08 22:17:42.788187: Step 3940: Train accuracy = 100.0%
INFO:tensorflow:2017-12-08 22:17:42.788366: Step 3940: Cross entropy = 0.022333
INFO:tensorflow:2017-12-08 22:17:42.849502: Step 3940: Validation accuracy = 98.0% (N=100)
INFO:tensorflow:2017-12-08 22:17:43.460964: Step 3950: Train accuracy = 100.0%
INFO:tensorflow:2017-12-08 22:17:43.461111: Step 3950: Cross entropy = 0.011859
INFO:tensorflow:2017-12-08 22:17:43.523527: Step 3950: Validation accuracy = 99.0% (N=100)
INFO:tensorflow:2017-12-08 22:17:44.140796: Step 3960: Train accuracy = 100.0%
INFO:tensorflow:2017-12-08 22:17:44.140939: Step 3960: Cross entropy = 0.019731
INFO:tensorflow:2017-12-08 22:17:44.202143: Step 3960: Validation accuracy = 97.0% (N=100)
INFO:tensorflow:2017-12-08 22:17:44.814104: Step 3970: Train accuracy = 100.0%
INFO:tensorflow:2017-12-08 22:17:44.814251: Step 3970: Cross entropy = 0.022970
INFO:tensorflow:2017-12-08 22:17:44.876477: Step 3970: Validation accuracy = 96.0% (N=100)
INFO:tensorflow:2017-12-08 22:17:45.484385: Step 3980: Train accuracy = 99.0%
INFO:tensorflow:2017-12-08 22:17:45.484528: Step 3980: Cross entropy = 0.030179
INFO:tensorflow:2017-12-08 22:17:45.545619: Step 3980: Validation accuracy = 98.0% (N=100)
INFO:tensorflow:2017-12-08 22:17:46.156432: Step 3990: Train accuracy = 100.0%
INFO:tensorflow:2017-12-08 22:17:46.156579: Step 3990: Cross entropy = 0.017378
INFO:tensorflow:2017-12-08 22:17:46.220964: Step 3990: Validation accuracy = 97.0% (N=100)
INFO:tensorflow:2017-12-08 22:17:46.772056: Step 3999: Train accuracy = 100.0%
INFO:tensorflow:2017-12-08 22:17:46.772206: Step 3999: Cross entropy = 0.015052
INFO:tensorflow:2017-12-08 22:17:46.836860: Step 3999: Validation accuracy = 100.0% (N=100)
INFO:tensorflow:Final test accuracy = 97.7% (N=301)
INFO:tensorflow:Froze 2 variables.
Converted 2 variables to const ops.
(tensorflow) [gott@na TrafficJameRecognize]$
```

Hình 5.1: Kết quả huấn luyện

con đường đang trong tình trạng ùn tắc. Kết quả dự đoán cho lớp cả hai hình ảnh này là chính xác theo lớp kẹt xe.



ket 0.997227
thong 0.00277315

Hình 5.2: Ảnh kiểm thử 1 - 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.



Hình 5.3: Ảnh kiểm thử 2 - 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.



thong 0.719465
ket 0.280535

Hình 5.4: Ảnh kiểm thử 3 - kết quả



thong 0.725881
ket 0.274119

Hình 5.5: Ảnh kiểm thử 4 - kết quả

Chương 6

Kết luận

6.1 Các kết quả

Đề cương 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 ở chương 1.

- 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.
- Cài đặt, cấu hình các thông số cũng như mô hình mạng đã được huấn luyện trước, ứng dụng vào vấn đề phân loại ảnh giao thông.
- Kết quả huấn luyện và kiểm thử đối với mạng googLeNet thu được khá tốt.

6.2 Hướng phát triển

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

- **Bước 1.** Tiếp tục xem xét việc xác định các lớp ảnh giao 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.
- **Bước 2.** Xây dựng ứng dụng di động nhận biết kẹt xe.
- **Bước 3.** Phát triển Web-service nhận thông tin hình ảnh từ các camera kết nối, phối hợp với mô hình đã huấn luyện để tiến hành phân loại giao thông.

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] Rémi Cadène, Nicolas Thome, and Matthieu Cord. Master's Thesis : Deep Learning for Visual Recognition. 2016.
- [2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. Chapter 6:Deep Feedforward Networks.
- [3] Andrej Karpathy. CS231n: Convolutional Neural Networks for Visual Recognition. CNN overview, 2016.
- [4] Andrej Karpathy. CS231n: Convolutional Neural Networks for Visual Recognition. convolutional layers, 2016.
- [5] Andrej Karpathy. CS231n: Convolutional Neural Networks for Visual Recognition. CNN architecture, 2016.
- [6] Andrej Karpathy. CS231n: Convolutional Neural Networks for Visual Recognition. transfer learning, 2016.
- [7] Andrej Karpathy. CS231n: Convolutional Neural Networks for Visual Recognition. module 1 - Neural Networks and module 2 - Convolutional Neural Network, 2016.

- [8] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. 2013.
- [9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed and Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. 2014.
- [10] tensorflow.org. Image retraining.
- [11] tensorflow.org. Tensorflow programmer guide.
- [12] Tom White. *Hadoop: The Definitive Guide*. oreilly.
- [13] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. 2013.