

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN**



BÁO CÁO THỰC TẬP CHUYÊN MÔN

**ĐỀ TÀI: NGHIÊN CỨU VỀ THUẬT TOÁN CNN VÀ ỨNG DỤNG TRONG
VIỆC NHẬN DIỆN VĂN BẢN TỪ HÌNH ẢNH**

Giảng viên hướng dẫn: TRẦN THỊ DUNG

Sinh viên thực hiện: NGUYỄN TRUNG KIÊN

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá : 58

Tp. Hồ Chí Minh, tháng 8 năm 2020

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN**



BÁO CÁO THỰC TẬP CHUYÊN MÔN

**ĐỀ TÀI: NGHIÊN CỨU VỀ THUẬT TOÁN CNN VÀ ỨNG DỤNG TRONG
VIỆC NHẬN DIỆN VĂN BẢN TỪ HÌNH ẢNH**

Giảng viên hướng dẫn: TRẦN THỊ DUNG

Sinh viên thực hiện: NGUYỄN TRUNG KIÊN

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá : 58

Tp. Hồ Chí Minh, tháng 8 năm 2020

MỤC LỤC

DANH SÁCH HÌNH ẢNH	1
DANH SÁCH BẢNG BIỂU	2
NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN	3
CHƯƠNG 1: MỞ ĐẦU.....	4
1.1. Tổng quan về xử lý ảnh và nhận dạng văn bản	4
1.2. Mục tiêu nghiên cứu	5
1.3. Đối tượng và phạm vi nghiên cứu	5
1.3.1. Đối tượng	5
1.3.2. Phạm vi nghiên cứu	5
1.4. Ý nghĩa của đề tài.....	5
1.5. Cấu trúc báo cáo thực tập chuyên môn	5
CHƯƠNG 2: TÌM HIỂU VỀ XỬ LÝ ẢNH VÀ NHẬN DẠNG VĂN BẢN TỪ HÌNH ẢNH.....	6
2.1. Tổng quan về nhận dạng hình ảnh.....	6
2.1.1. Giới thiệu chung.....	6
2.1.2. Các bước trong quá trình xử lý ảnh	6
2.2. Một số thuật ngữ	7
2.2.1. Hệ màu RGB	7
2.2.2. Pixel	7
2.2.3. Ảnh màu	7
2.2.4. Tensor	8
2.2.5. Ảnh xám.....	8
2.2.6. Chuyển từ ảnh màu sang ảnh xám	9
2.3. Sơ lược về convolution.....	9
2.3.1. Convolutional (tích chập) là gì?	9
2.3.2. Phương pháp tích trập trong xử lý ảnh.....	10
2.3.3. Một số kernel	11
2.4. Xử lý ảnh và các phương pháp xử lý ảnh	13
2.4.1. Xử lý ảnh là gì?	13
2.4.2. Các phương pháp xử lý ảnh	14
2.5. Các phương pháp nhận dạng văn bản từ hình ảnh	15

CHƯƠNG 3: GIỚI THIỆU VỀ NN – NEURAL NETWORK.....	16
3.1. Tổng quan về mạng nơ-ron (Neural network - NN)	16
<i>3.1.1. Neural network là gì?</i>	16
<i>3.1.2. Hoạt động của các nơ-ron</i>	16
3.2. Mô hình neural network	16
<i>3.2.1. Mô hình logistic regression</i>	16
<i>3.2.2. Mô hình tổng quát</i>	17
3.3. Feedforward	20
CHƯƠNG 4: TỔNG QUAN VỀ CNN – CONVOLUTION NEURAL NETWORK VÀ	
BÀI TOÁN NHẬN DẠNG ẢNH	22
4.1. Tổng quan về mạng nơ-ron tích chập (CNN)	22
4.2. Cấu trúc của CNN	22
<i>4.2.1. Convolution layer</i>	23
<i>4.2.2. Pooling layer</i>	23
<i>4.2.3. Fully connected layer</i>	25
<i>4.2.4. Mô hình CNN</i>	25
CHƯƠNG 5: NHẬN DẠNG VĂN BẢN VỚI CNN.....	26
5.1. Đặt vấn đề bài toán	26
5.2. Mô hình tổng quát	26
5.3. Xây dựng mạng nơ-ron nhận dạng ký tự	27
<i>5.3.1. Xây dựng mạng nơ-ron</i>	27
5.4. Xây dựng chương trình thử nghiệm	29
<i>5.4.1. Giới thiệu Python</i>	29
<i>5.4.2. Tổng quan về thư viện OpenCV và Tesseract</i>	29
<i>5.4.3. Xây dựng code mẫu</i>	30
5.5. Kết quả thu được	37
5.6. Hạn chế	38
CHƯƠNG 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	39
TÀI LIỆU THAM KHẢO	40

DANH SÁCH HÌNH ẢNH

Hình 2.1. Các bước trong quá trình xử lý ảnh.....	6
Hình 2.2. Mạng Deep Learning gồm nhiều Convolution layer, Pooling layer và Full connection layer.....	9
Hình 2.3. Một ví dụ cho phép tích trập	10
Hình 2.4. Ví dụ cho cửa sổ trượt.....	10
Hình 2.5. Ảnh gốc	11
Hình 2.6. Box Blur	11
Hình 2.7. Gaussian Blur 3x3	11
Hình 2.8. Gaussian Blur 5x5	12
Hình 2.9. Sharpen.....	12
Hình 2.10. Edge detection	13
Hình 2.11. Ví dụ nhị phân từ ảnh xám bằng thresholding	14
Hình 2.12. Median filter.....	15
Hình 3.1. Mô hình logistic regression.....	16
Hình 3.2. Mô hình neural network	17
Hình 3.3. Ví dụ mô hình tổng quát.....	18
Hình 3.4. feedforward từ input layer đến output layer.....	21
Hình 4.1. Pooling đối với ảnh màu (tensor 3 chiều)	24
Hình 4.2. Pooling đối với ảnh xám (ma trận 2 chiều).....	24
Hình 4.3. Fully connected layer	25
Hình 4.4. Cấu trúc mô hình CNN	25
Hình 5.1. Ảnh đầu vào	26
Hình 5.2. Hình chưa chấp nhận được.....	26
Hình 5.3. Hình chấp nhận được	26
Hình 5.4. Mô hình CNN trong nhận dạng số viết tay	27
Hình 5.5. Minh họa fully connection	29

Hình 5.6. Code main.py	30, 31, 32, 33
Hình 5.7. Code model.py	33
Hình 5.8. Code mẫu.....	34, 35, 36, 37
Hình 5.9. Input đầu vào	38
Hình 5.10. Các kết quả qua các filter	38

DANH SÁCH BẢNG BIỂU

Bảng 5.1. Kết quả thu được với các filter được tô đậm là chấp nhận được.....	37
--	----

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp. Hồ Chí Minh, ngày tháng năm
Giáo viên hướng dẫn

Trần Thị Dung

CHƯƠNG 1: MỞ ĐẦU

1.1. Tổng quan về xử lý ảnh và nhận dạng văn bản

Nhận dạng là lĩnh vực được các nhà khoa học rất quan tâm vì tính ứng dụng của nó trong cuộc sống hiện nay, có nhiều lĩnh vực nhận dạng như nhận dạng tín hiệu, nhận dạng giọng nói, nhận dạng hình ảnh,... Vấn đề nhận dạng văn bản từ hình ảnh là một vấn đề thú vị đối với các nhà nghiên cứu.

Văn bản từ hình ảnh có thể được ứng dụng trong các cơ quan, nhà máy, xí nghiệp, trường học. Điển hình như hệ thống nhận diện biển số xe của các nhà giữ xe, nhận diện biển số để phạt nguội khi vi phạm giao thông. Và điển hình nhất là ứng dụng trong việc dịch bằng máy ảnh của ông lớn Google.

Để nhận dạng văn bản có nhiều phương pháp xử lý và kỹ thuật khác nhau như: logic mờ, giải thuật di truyền, mô hình xác suất thống kê, mô hình mạng nơ-ron.

Hiện nay, với sự phát triển không ngừng của khoa học máy tính, cũng như sự bùng nổ trí tuệ nhân tạo, phương pháp Deep Learning ra đời cũng dùng để phục vụ trong việc nhận dạng và xử lý ảnh. Deep Learning là thuật toán nằm trong nhóm các thuật toán Machine Learning để phát triển trí tuệ nhân tạo, Deep Learning dựa trên ý tưởng từ não bộ tới việc tiếp thu nhiều tầng biểu đạt, cả cụ thể lẫn trừu tượng, qua đó làm rõ dữ liệu. Deep Learning được ứng dụng trong nhận diện hình ảnh, nhận diện giọng nói, xử lý ngôn ngữ tự nhiên. Hiện nay rất nhiều các bài toán nhận dạng sử dụng Deep Learning để giải quyết do Deep Learning có thể giải quyết các bài toán với số lượng lớn, kích thước đầu vào lớn với hiệu năng cũng như độ chính xác vượt trội so với các phương pháp phân lớp truyền thống. Các thuật toán Deep Learning phổ biến như: Deep Boltzmann Machines (DBM), Deep Belief Networks (DBN), Convolutional Neural Network (CNN) ,...

Vì những lý do trên, tôi chọn nghiên cứu đề tài: **“Nghiên cứu về thuật toán CNN và ứng dụng trong việc nhận dạng văn bản từ hình ảnh”**

1.2. Mục tiêu nghiên cứu

Mục tiêu chính của đề tài này là sử dụng kỹ thuật Convolutional Neural Network (CNN) để xây dựng chương trình nhận dạng chữ viết tay

1.3. Đối tượng và phạm vi nghiên cứu

1.3.1. Đối tượng

- Các văn bản từ hình ảnh.
- Cơ sở lý thuyết về nhận dạng ảnh.
- Các phương pháp, giải thuật về nhận dạng.
- Thuật toán Convolutional Neural Network (CNN).

1.3.2. Phạm vi nghiên cứu

- Nghiên cứu kỹ thuật xử lý ảnh

1.4. Ý nghĩa của đề tài

Về khoa học: Đề tài sẽ mang ý nghĩa cung cấp tài liệu về mặt lý thuyết để làm rõ về các phương pháp và kỹ thuật nhận diện văn bản từ hình ảnh.

Về thực tiễn: Đề tài góp phần hỗ trợ trong việc phát triển các ứng dụng liên quan đến việc xử lý ảnh không chỉ nằm trong nhận diện chữ viết như nhận diện đồ vật,...

1.5. Cấu trúc báo cáo thực tập chuyên môn

- Chương 1: Mở đầu
- Chương 2: Tìm hiểu về xử lý ảnh và nhận dạng hình ảnh.
- Chương 3: Tổng quan về thuật toán CNN – Convolutional Neural Network.
- Chương 4: Nhận dạng văn bản với CNN
- Chương 5: Kết luận và hướng phát triển

CHƯƠNG 2: TÌM HIỂU VỀ XỬ LÝ ẢNH VÀ NHẬN DẠNG VĂN BẢN TỪ HÌNH ẢNH

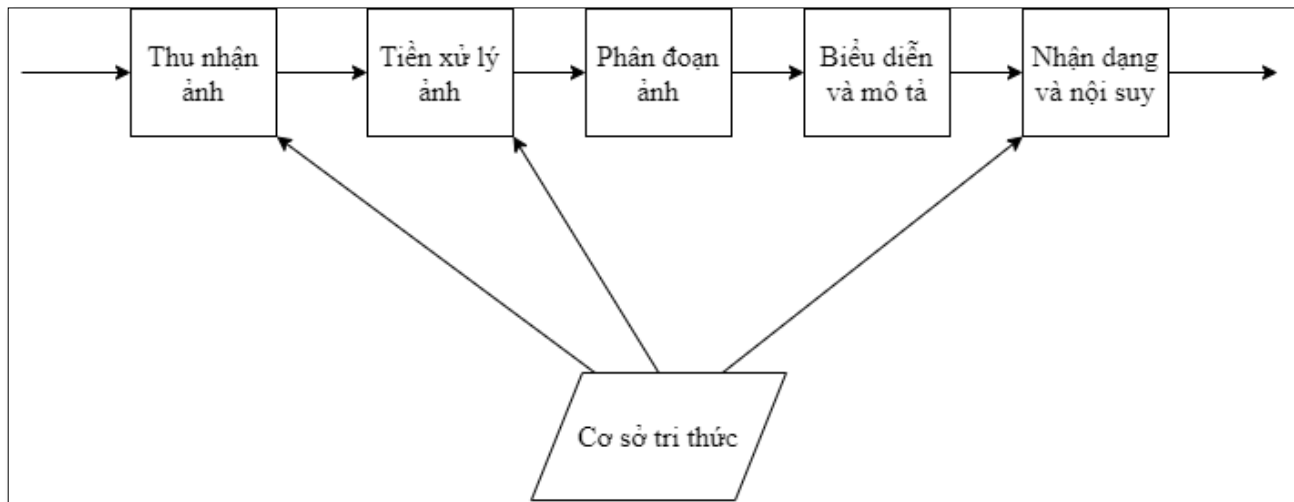
2.1. Tổng quan về nhận dạng hình ảnh

2.1.1. Giới thiệu chung

Hiện nay, vấn đề xử lý văn bản từ hình ảnh có ý nghĩa quan trọng, có nhiều ứng dụng rộng rãi trong đời sống xã hội như nhận dạng biển số xe, ứng dụng dịch bằng máy ảnh, nhận dạng phiếu hàng hóa, truy xuất thông tin từ chứng minh nhân dân,.. Vấn đề nhận dạng hình ảnh nói chung và vấn đề nhận dạng văn bản từ hình ảnh nói riêng là một thách thức lớn đối với các nhà nghiên cứu. Có nhiều phong chữ khác nhau trên mỗi hình ảnh khác nhau đó đó chúng ta không thể xác định cách duy nhất để nhận dạng văn bản một cách chính xác, tin cậy để nhận dạng bất kì văn bản nào là một điều không dễ dàng.

Với mong muốn tìm ra giải pháp để nhận dạng văn bản từ hình ảnh hiệu quả, tôi nghiên cứu kỹ thuật Deep Learning, cụ thể là thuật toán CNN, đây là phương pháp có thể giải quyết các bài toán với số lượng lớn, kích thước đầu vào lớn với hiệu năng cũng như độ chính xác vượt trội so với các phương pháp phân lớp truyền thống.

2.1.2. Các bước trong quá trình xử lý ảnh



Hình 2.1. Các bước trong quá trình xử lý ảnh

2.1.2.1. Thu nhận ảnh

Các thiết bị thu nhận ảnh có hai loại chính ứng với hai loại ảnh thông dụng Raster, Vector và có thể cho ảnh đen trắng hoặc ảnh màu.

2.1.2.2. Tiền xử lý

Sau bộ phận thu nhận ảnh, ảnh có thể bị nhiễu hoặc độ tương phản thấp nên cần đưa vào bộ tiền xử lý để nâng cao chất lượng ảnh.

2.1.2.3. Phân đoạn ảnh

Tách ảnh đầu vào thành các vùng để biểu diễn, phân tích và nhận dạng ảnh.

2.1.2.4. Biểu diễn ảnh

Các vật thể sau khi được phân ddaonj có thể được mô tả dưới dạng chuỗi các điểm và biểu diễn ảnh thường được sử dụng khi ta quan tâm đến đặc tính bên trong của vùng ảnh. Ví dụ: đường cong, hình dạng,... quá trình biểu diễn ảnh là việc biến đổi các số liệu của ảnh thành dạng thích hợp, cần thiết cho việc xử lý bằng máy tính

2.1.2.5. Nhận dạng và nội suy

Là quá trình phân loại vật thể dựa trên cơ sở các chi tiết mô tả vật thể và nhận dạng ảnh là quá trình xác định ảnh và quá trình này thu được bằng cách so sánh với mẫu đã được lưu trữ từ trước. Đối với đề tài này, vật thể và các mẫu vật thể là những chữ số trong văn bản.

2.1.2.6. Cơ sở tri thức

Các quá trình xử lý liệt kê trong hình thức xử lý ảnh được thực hiện dưới sự giám sát và thực hiện dựa trên cơ sở các kiến thức về lĩnh vực xử lý ảnh.

2.2. Một số thuật ngữ

2.2.1. Hệ màu RGB

RGB viết tắt của red – green – blue (đỏ - lục -lam), đây là ba màu cơ bản được tách ra từ lăng kính, tùy theo tỷ lệ trộn có thể tạo ra được nhiều màu khác nhau.

2.2.2. Pixel

Pixel (hay điểm ảnh) là một khối màu rất nhỏ và là đơn vị cơ bản nhất để tạo nên một bức ảnh kỹ thuật số.

2.2.3. Ảnh màu

Có thể hiểu một bức ảnh kỹ thuật số được tạo nên từ rất nhiều pixels, mỗi pixel biểu diễn một màu khác nhau. Ví dụ có một bức ảnh có kích thước 400x300 pixel, ta có thể hiểu như bức ảnh ấy là một ma trận 400x300 với 400 cột và 300 hàng, mỗi ô trong ma

trận tương ứng với một pixel. Vậy ảnh màu là một ma trận các pixel mà mỗi pixel biểu diễn một điểm màu.

$$\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,400} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,400} \\ \vdots & \vdots & \ddots & \vdots \\ w_{300,1} & w_{300,2} & \cdots & w_{300,400} \end{bmatrix}$$

Mỗi điểm màu được biểu diễn bởi ba hệ số (r, g, b). để tiện cho việc xử lý ảnh thì sẽ tách ma trận ảnh thành 3 chanel red, green, blue:

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,400} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,400} \\ \vdots & \vdots & \ddots & \vdots \\ r_{300,1} & r_{300,2} & \cdots & r_{300,400} \end{bmatrix}$$

$$G = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,400} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,400} \\ \vdots & \vdots & \ddots & \vdots \\ g_{300,1} & g_{300,2} & \cdots & g_{300,400} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,400} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,400} \\ \vdots & \vdots & \ddots & \vdots \\ b_{300,1} & b_{300,2} & \cdots & b_{300,400} \end{bmatrix}$$

2.2.4. Tensor

Khi dữ liệu dạng 1 chiều, người ta gọi là vector, mặc định vector viết dưới dạng cột. Khi dữ liệu dạng 2 chiều, người ta gọi là ma trận, kích thước số hàng* số cột. Khi dữ liệu nhiều hơn 2 chiều sẽ được gọi là tensor. Ví dụ dữ liệu ảnh màu kích thước 400*300 trên máy tính sẽ được biểu diễn bởi ma trận 3 chiều 400*300*3 do có 3 ma trận màu red, green, blue chồng lên nhau

2.2.5. Ảnh xám

Tương tự ảnh màu, ảnh xám cũng có thể biểu diễn dưới dạng ma trận 400*300 nhưng thay vì mỗi ô là một vector (r, g, b) như ảnh màu thì ảnh xám mỗi ô có giá trị trong khoảng [0, 255]. Do đó khi biểu diễn ảnh xám trên máy tính chỉ cần một ma trận là

đủ. Giá trị 0 là màu đen, 255 là màu trắng, giá trị pixel càng gần 0 thì càng tối, ngược lại, giá trị pixel càng về 255 thì càng sáng

2.2.6. Chuyển từ ảnh màu sang ảnh xám

Do mỗi pixel trong ảnh màu được biểu diễn bằng 3 giá trị (r, g, b) còn trong ảnh xám thì chỉ cần 1 giá trị x trong khoảng [0, 255] nên chúng ta cần chuyển 3 giá trị (r, g, b) về một giá trị duy nhất. Để làm điều đó ta có công thức:

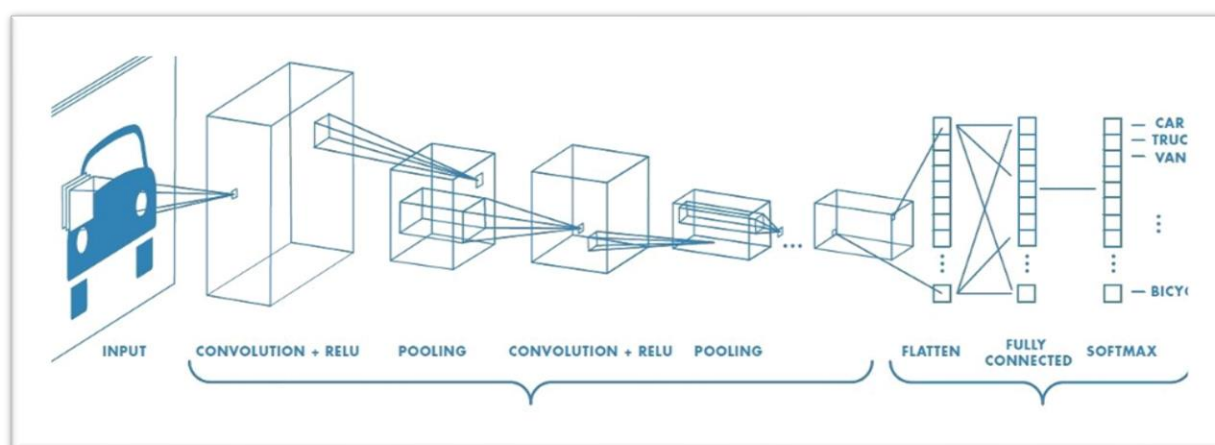
$$x = r * 0.299 + g * 0.587 + b * 0.114$$

Tuy nhiên việc chuyển ngược từ ảnh xám sang ảnh màu là một việc khó vì ta phải tìm r, g, b nên độ chính xác khó có thể cao được

2.3. Sơ lược về convolution

2.3.1. Convolutional (tích chập) là gì?

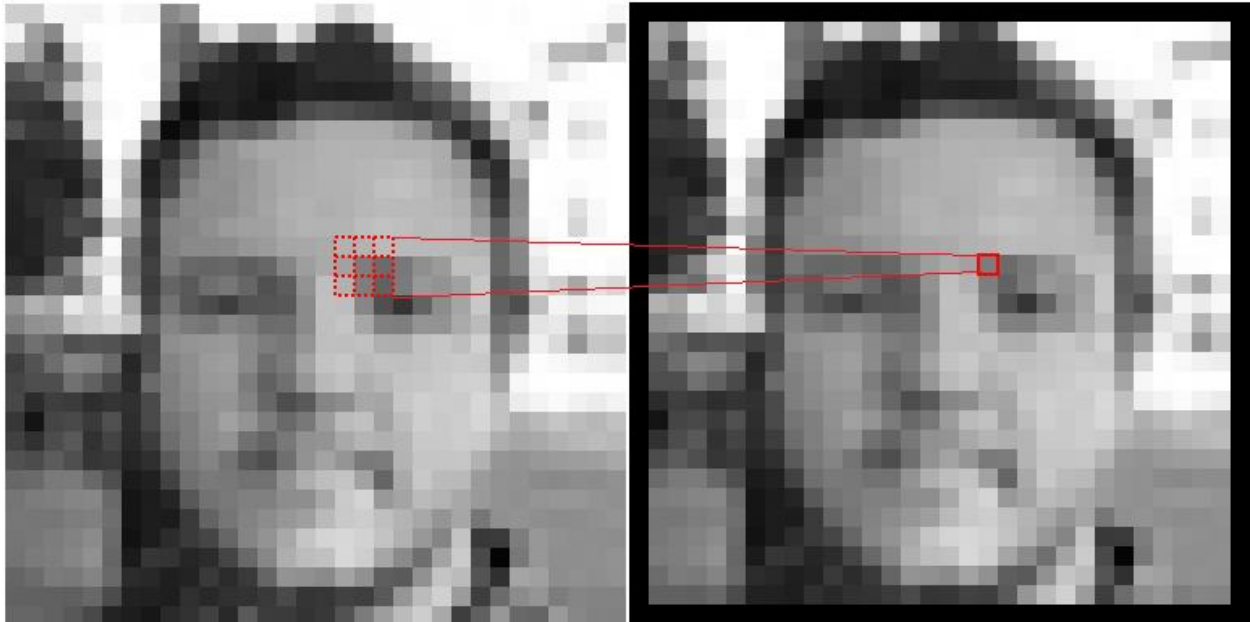
Convolution là một kỹ thuật quan trọng xử lý ảnh. Nó có mặt trong hầu hết các thuật toán làm mờ (Gaussian Blur), hay làm rõ các đường (Edge detector). Có thể hiểu tích chập là một cửa sổ trượt trên một ma trận. Trong nhận dạng ảnh, convolution layer (tạm dịch là lớp tích trập) là một tầng biến đổi ma trận đầu vào để làm rõ và tách ra các đặc tính của hình ảnh mà vẫn bảo toàn tính tương quan không gian giữa đầu ra và đầu vào



Hình 2.2. Mạng Deep Learning gồm nhiều Convolution layer, Pooling layer và Full connection layer

Các convolutional layer có các tham số (kernel) đã được học để tự điều chỉnh lấy ra những thông tin chính xác nhất mà không cần chọn các đặc tính.

2.3.2. Phương pháp tích trập trong xử lý ảnh



Hình 2.3. Một ví dụ cho phép tích trập

Như hình trên tôi xin phép lấy một ví dụ dễ nhìn hơn bằng ma trận 5x5 và có cửa sổ chập là một ma trận 3x3:

image						kernel		
1	0	0	1	0		1	0	0
0	1	1	0	1	X	0	1	1
1	0	1	0	1		1	0	1
1	0	0	1	0				
0	1	1	0	1				

Hình 2.4. Ví dụ cho cửa sổ trượt

Dựa vào hình trên, ta có bên trái là ma trận đầu vào là một bức ảnh đen trắng. Mỗi giá trị của ma trận tương đương với một điểm ảnh, 0 là màu đen, 1 là màu trắng (đối với ảnh nhị phân hóa). Nếu là ảnh xám hóa (Grayscale) thì giá trị từ 0 đến 255.

Bên phải là kernel, filter hay feature detector là một ma trận dùng để biến đổi ma trận đầu vào trong 2 vòng lặp lồng nhau convolution. Từ trái qua phải, từ trên xuống dưới, thực hiện phép nhân từ phần tử ở cùng vị trí ở ma trận cửa sổ với kernel rồi tính tổng giá trị scalar và điền vào ma trận kết quả. Cứ thế cho đến khi chạy hết:

$$\text{Giá trị scalar} = a \cdot b = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$$

2.3.3. Một số kernel

Mỗi kernel khác nhau sẽ cho ra những output khác nhau, sau đây là một số kernel ví dụ cho một ảnh gốc:



Hình 2.5. Ảnh gốc

2.3.3.1. Kernel làm mờ ảnh (Blur)

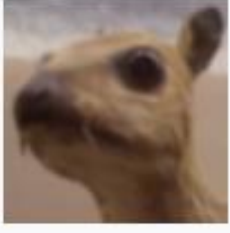
$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
---	--

Hình 2.6. Box Blur

$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
--	--

Hình 2.7. Gaussian Blur 3x3

$\frac{1}{256}$	1	4	6	4	1
	4	16	24	16	4
	6	24	36	24	6
	4	16	24	16	4
	1	4	6	4	1




Hình 2.8. Gaussian Blur 5x5

Ở kernel “Box blur”, giá trị mức xám đầu ra là giá trị trung bình chuẩn của các giá trị ở các điểm ảnh bị bao phủ bởi kernel. Các hệ số của bộ lọc là 1 thay vì là 1/9 là để cho việc tính toán được hiệu quả hơn. Sau khi quá trình tính toán kết thúc, tất cả các giá trị được chia cho 9.

Ở kernel “Gaussian blur 3x3”, các giá trị điểm ảnh được nhân với các hệ số khác nhau, thể hiện sự quan trọng so với các điểm khác. Điểm ảnh ở giữa kernel được nhân với giá trị lớn hơn, thể hiện sự quan trọng lớn hơn trong tính toán giá trị trung bình. Các hệ số khác được coi như là khoảng cách từ điểm đó tới trung tâm kernel. Gaussian blur 5x5 cũng có ý nghĩa như vậy




2.3.3.2. Kernel làm sắc nét (Sharpen)

0	-1	0
-1	5	-1
0	-1	0



Hình 2.9. Sharpen

2.3.3.3. Kernel xác định cạnh (Edge detection)

$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Hình 2.10. Edge detection

2.4. Xử lý ảnh và các phương pháp xử lý ảnh

2.4.1. Xử lý ảnh là gì?

Con người thi nhận thông tin qua các giác quan, trong đó có thị giác đóng vai trò quan trọng nhất. Những năm trở lại đây, với sự phát triển của công nghệ thông tin nói chung và phần cứng nói riêng, việc xử lý ảnh và đồ họa phát triển một cách mạnh mẽ và có nhiều ứng dụng trong cuộc sống. Xử lý ảnh và đồ họa đóng một vai trò quan trọng trong tương tác trí tuệ nhân tạo.

Quá trình xử lý ảnh được xem như là quá trình thao tác ảnh đầu vào nhằm cho ra kết quả như mong muốn. Kết quả đầu ra của một quá trình xử lý ảnh có thể là một ảnh “tốt hơn” hoặc một kết luận.

Ảnh có thể xem là tập hợp của rất nhiều điểm ảnh được xem như là một đặc trưng cường độ sáng được gọi là các pixels.

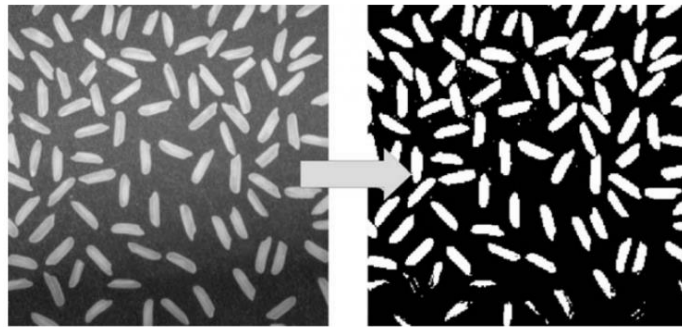
2.4.2. Các phương pháp xử lý ảnh

2.4.2.1. Lọc mịn ảnh (làm mờ - Blur)

Bằng cách thay thế các giá trị của mỗi điểm ảnh bằng mức trung bình của mức cường độ của các điểm ảnh lân cận được xác định bởi mặt nạ lọc (kernel), quá trình này tạo ra một hình ảnh bị giảm chuyển tiếp giữa các điểm ảnh. Ứng dụng rõ ràng nhất của làm mịn là giảm nhiễu.

2.4.2.2. Nhị phân ảnh

Ảnh nhị phân (binary image): giá trị mỗi điểm ảnh là 0 hoặc 1, nghĩa là trắng hoặc đen. Trong thực tế, người ta sử dụng ảnh xám để biểu diễn ảnh nhị phân có 2 giá trị là 0 hoặc 255 tương ứng với 2 giá trị 0 và 1.



Hình 2.11. Ví dụ nhị phân từ ảnh xám bằng thresholding

Nhị phân ảnh là quá trình biến đổi một ảnh xám thành ảnh nhị phân. Ví dụ ta có giá trị cường độ sáng tại một điểm ảnh là $I(x,y)$. $INP(x,y)$ là giá trị cường độ sáng của điểm ảnh trên ảnh nhị phân. Với x và y là tọa độ pixel của ảnh. Để biến đổi ảnh xám thành ảnh nhị phân. Ta so sánh giá trị cường độ điểm sáng tại một điểm ảnh với một ngưỡng nhị phân T , thông thường người ta chọn $T = 128$.

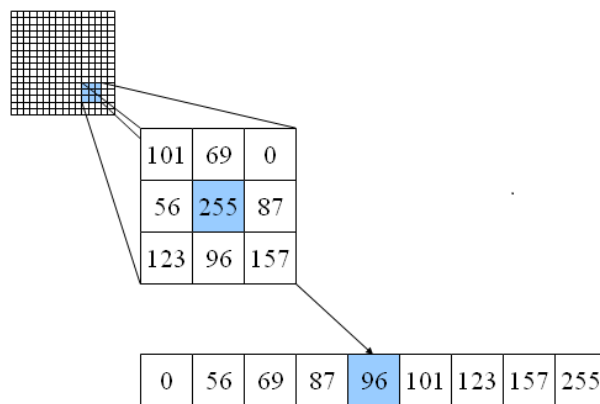
- Nếu $I(x,y) < T$ thì $INP(x,y) = 0$ (0) – màu trắng
- Nếu $I(x,y) > T$ thì $INP(x,y) = 255$ (1) – màu đen

2.4.2.3. Xác định cạnh – Edge detection

Edge Detection là một kỹ thuật xử ảnh được sử dụng để tìm kiếm viền bao của của các đối tượng trong ảnh. Trong xử lý ảnh, việc kiểm tra thực chất là tìm những khu vực bị mất liên tục về độ sáng.

2.4.2.4. Bộ lọc làm mờ trung vị Median Filter

Trung vị (median) là ta lấy ra giá trị trung vị sau khi sắp thứ tự dãy số. Ví dụ: cho dãy median([6, 8, 11, 4, 1]). Sau khi sắp thứ tự dãy median, ta được [1, 4, 6, 8, 11] => số ở giữa dãy đã sắp thứ tự là 6, vậy dãy median có giá trị là 6. Median rất hiệu quả với nhiễu muối tiêu (tức nhiễu các hạt nhỏ trong ảnh).



Hình 2.12. Median filter

2.5. Các phương pháp nhận dạng văn bản từ hình ảnh

Có nhiều phương pháp trong các hệ thống nhận dạng chữ số viết tay, có thể kể đến như: đối sánh mẫu, thống kê, cấu trúc, mạng nơ-ron, SVM. Trong bài báo cáo này tôi chọn dùng mạng nơ-ron (thuật toán CNN trong Deep Learning)

CHƯƠNG 3: GIỚI THIỆU VỀ NN – NEURAL NETWORK

3.1. Tổng quan về mạng nơ-ron (Neural network - NN)

3.1.1. Neural network là gì?

Neural là tính từ của neuron (nơ-ron), network chỉ cấu trúc đồ thị nên có thể hiểu neural network (NN) là một hệ thống cấu trúc đồ thị lấy cấu trúc từ những hoạt động của các nơ-ron trong hệ thần kinh.

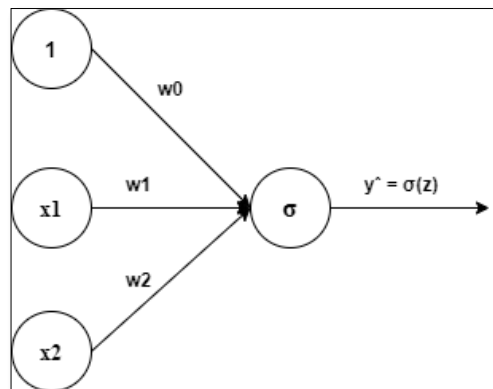
3.1.2. Hoạt động của các nơ-ron

Nơ-ron là các đơn vị cơ bản của hệ thần kinh và là một phần quan trọng nhất của não. Trong não bộ, mỗi nơ-ron có chứa nhân, các tín hiệu vào thông qua sợi nhánh và các tín hiệu ra thông qua sợi trục kết nối với các nơ-ron khác. Có thể hình dung rằng các nơ-ron nhận tín hiệu từ sợi nhánh và truyền qua sợi trục tới các sợi nhánh của các nơ-ron khác. Neural network lấy cảm hứng từ não bộ và cách thức các nơ-ron hoạt động bằng cách mô phỏng qua các thuật toán.

3.2. Mô hình neural network

3.2.1. Mô hình logistic regression

Mô hình logistic regression là mô hình neural network đơn giản nhất chỉ với input layer và output layer:

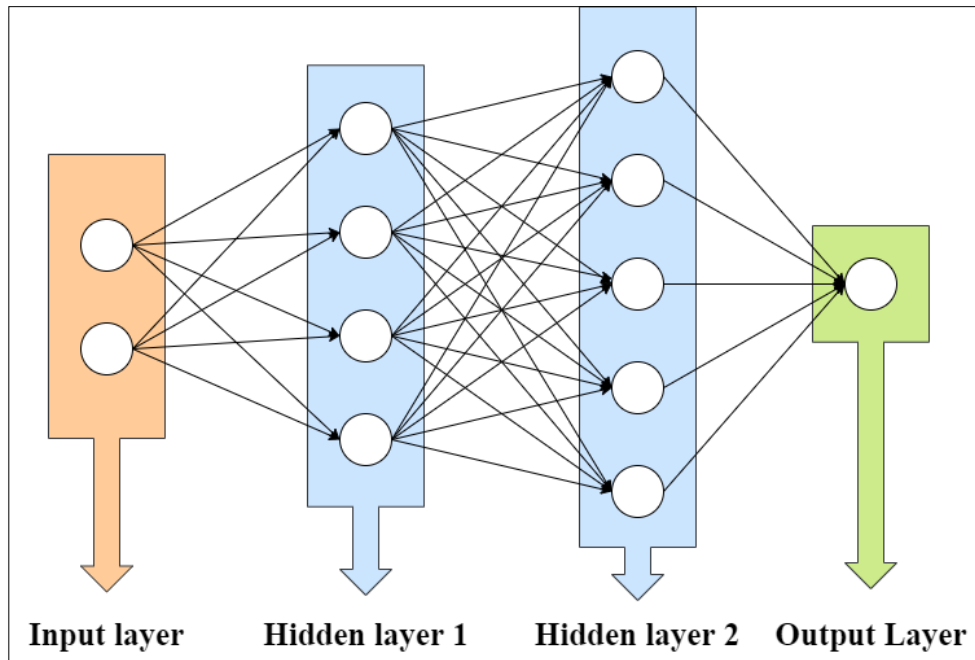


Hình 3.1. Mô hình logistic regression

Trong đó linear: $z = 1 * w_0 + x_1 * w_1 + x_2 * w_2$

Và: $y^ = \sigma(z) = \frac{1}{1 + e^{-z}}$ được gọi là activation function

3.2.2. Mô hình tổng quát



Hình 3.2. Mô hình neural network

Layer đầu tiên là input layer, các layer ở giữa là hidden layer, layer cuối là output layer. Các hình tròn là các node. Mỗi mô hình luôn có 1 input layer và output layer, có thể có hoặc không có hidden layer. Tổng số layer trong mô hình được quy ước là số layer -1 (vì không tính input layer). Ví dụ ở hình trên, có 1 input layer, 2 hidden layer và 1 output layer vậy số layer trong mô hình là 3.

Mỗi một node trong hidden layer và output layer:

- Liên kết với tất cả các node ở layer trước đó với các hệ số w riêng.
- Mỗi node có 1 hệ số bias b riêng. Hệ số w_0 được gọi là hệ số bias.
- Diễn ra 2 bước: tính tổng z và áp dụng activation function

Các ký hiệu:

- Số các node trong hidden layer thứ i là $l^{(i)}$.
- Ma trận $W^{(k)}$ kích thước $l^{(k-1)} * l^{(k)}$ là ma trận hệ số giữa layer $(k-1)$ và layer k , trong đó $w_{ji}^{(k)}$ là hệ số kết nối từ node thứ j của layer $k-1$ đến node thứ i của layer k .

- Vector $b^{(k)}$ kích thước $l^k * 1$ là hệ số bias (w_0) của các node trong layer k, trong đó $b_i^{(k)}$ là bias của node thứ i trong layer k.

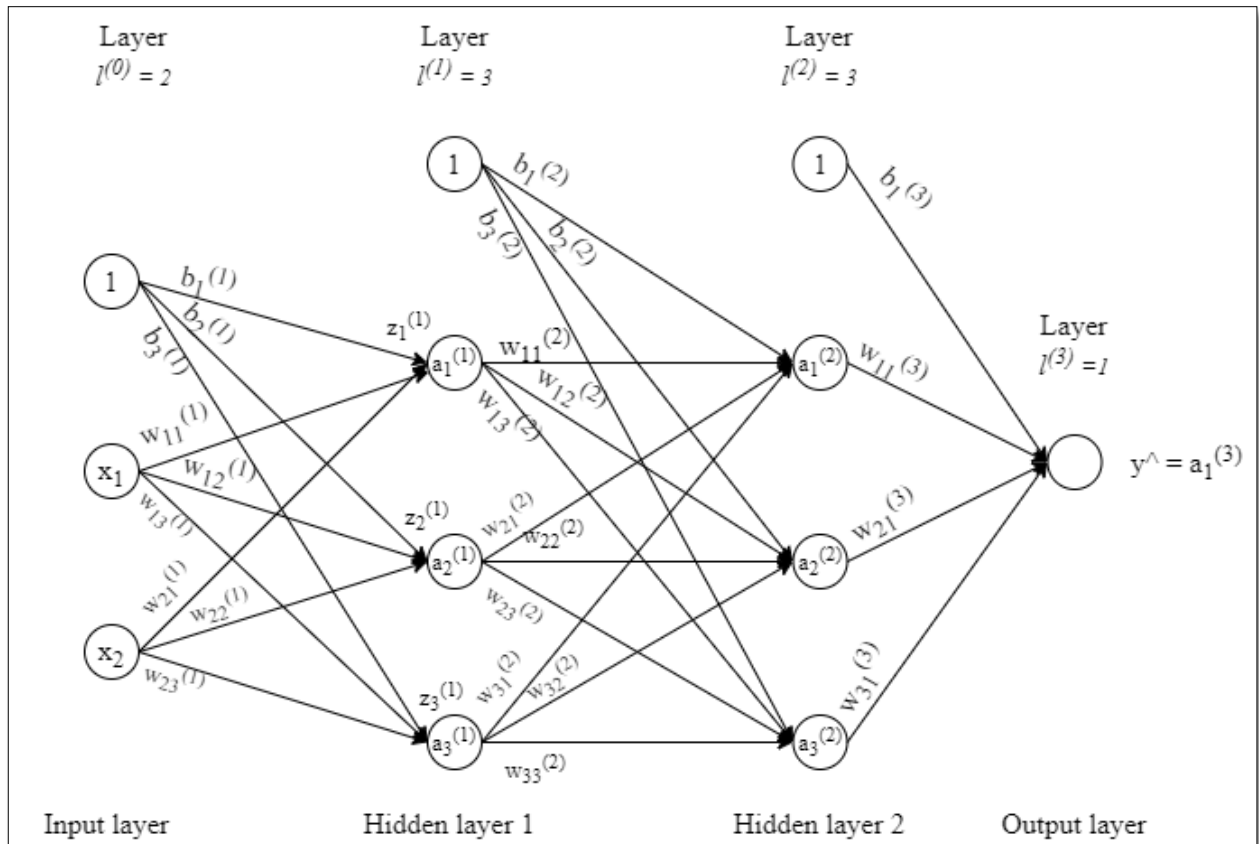
Với node thứ i trong layer l có bias $b_i^{(l)}$ thực hiện 2 bước:

B1. Tính tổng linear: $z_i^{(l)} = \sum_{j=1}^{l^{(l-1)}} a_j^{(l-1)} * w_{ji}^{(l)} + b_i^{(l)}$, là tổng tất cả các node trong layer trước nhân với hệ số w tương ứng, rồi cộng với bias b.

B2. Áp dụng activation function: $a_i^{(l)} = \sigma(z_i^{(l)})$

Vector $z^{(k)}$ kích thước $l^{(k)} * 1$ là giá trị của các node trong layer k sau bước tính tổng linear.

Vector $a^{(k)}$ kích thước $l^{(k)} * 1$ là giá trị của các node trong layer k sau khi áp dụng hàm activation function.



Hình 3.3. Ví dụ mô hình tổng quát

Mô hình trên là một ví dụ của neural network gồm 3 layer. Input layer có 2 node, hidden layer 1 và 2 đều có 3 node, output layer có 1 node. Mỗi một node trong hidden layer và output layer đều có bias nên trong các layer trước của nó (ở ví dụ này là input layer và 2 hidden layer đều có thêm một node để tính bias)

Ở hình 3.3, xét hidden layer 1, ta có:

Node thứ 1:

- $z_1^{(1)} = x_1 * w_{11}^{(1)} + x_2 * w_{21}^{(1)} + b_1^{(1)}$
- $a_1^{(1)} = \sigma(z_1^{(1)})$

Node thứ 2:

- $z_2^{(1)} = x_1 * w_{12}^{(1)} + x_2 * w_{22}^{(1)} + b_2^{(1)}$
- $a_2^{(1)} = \sigma(z_2^{(1)})$

Node thứ 3:

- $z_3^{(1)} = x_1 * w_{13}^{(1)} + x_2 * w_{23}^{(1)} + b_3^{(1)}$
- $a_3^{(1)} = \sigma(z_3^{(1)})$

Xét hidden layer 2, ta có:

Node thứ 1:

- $z_1^{(2)} = a_1^{(1)} * w_{11}^{(2)} + a_2^{(1)} * w_{21}^{(2)} + a_3^{(1)} * w_{31}^{(2)} + b_1^{(2)}$
- $a_1^{(2)} = \sigma(z_1^{(2)})$

Node thứ 2:

- $z_2^{(2)} = a_1^{(1)} * w_{12}^{(2)} + a_2^{(1)} * w_{22}^{(2)} + a_3^{(1)} * w_{32}^{(2)} + b_2^{(2)}$
- $a_2^{(2)} = \sigma(z_2^{(2)})$

Node thứ 3:

- $z_3^{(2)} = a_1^{(1)} * w_{13}^{(2)} + a_2^{(1)} * w_{23}^{(2)} + a_3^{(1)} * w_{33}^{(2)} + b_3^{(2)}$
- $a_3^{(2)} = \sigma(z_3^{(2)})$

Xét output layer, ta có:

- $z_1^{(3)} = a_1^{(2)} * w_{11}^{(3)} + a_2^{(2)} * w_{21}^{(3)} + a_3^{(2)} * w_{31}^{(3)} + b_1^{(3)}$
- $y^{\wedge} = a_1^{(3)} = \sigma(z_1^{(3)})$

3.3. Feedforward

Để nhất quán ký hiệu, gọi input layer là $a^{(0)} (= x)$ kích thước là $2*1$, $z^{(1)}$ là ma trận của hidden layer 1, $z^{(2)}$ là ma trận của hidden layer 2, $z^{(3)}$ là ma trận của output layer. Sử dụng hình 3.3 ở trên làm ví dụ, ta có:

$$z^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix} = \begin{bmatrix} a_1^{(0)} * w_{11}^{(1)} + a_2^{(0)} * w_{21}^{(1)} + b_1^{(1)} \\ a_1^{(0)} * w_{12}^{(1)} + a_2^{(0)} * w_{22}^{(1)} + b_2^{(1)} \\ a_1^{(0)} * w_{13}^{(1)} + a_2^{(0)} * w_{23}^{(1)} + b_3^{(1)} \end{bmatrix}$$

$$= (W^{(1)})^T * a^{(0)} + b^{(1)}$$

Và ta có: $a^{(1)} = \sigma(z^{(1)})$

Trong đó:

$$W^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix} \Rightarrow (W^{(1)})^T = \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} \end{bmatrix}$$

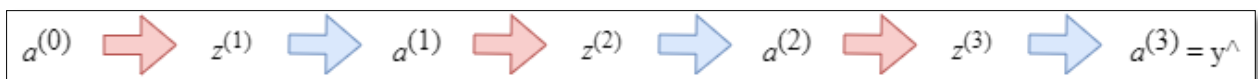
$$a^{(0)} (= x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ hay } \begin{bmatrix} a_1^{(0)} \\ a_2^{(0)} \end{bmatrix}$$

$$b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}$$

Tương tự, ta có:

$$\begin{aligned}z^{(2)} &= (W^{(2)})^T * a^{(1)} + b^{(2)} \\a^{(2)} &= \sigma(z^{(2)}) \\z^{(3)} &= (W^{(3)})^T * a^{(2)} + b^{(3)} \\a^{(3)} &= \sigma(z^{(3)})\end{aligned}$$

Từ đó ta có feedforward như sau:



Hình 3.4. feedforward từ input layer đến output layer

CHƯƠNG 4: TỔNG QUAN VỀ CNN – CONVOLUTION NEURAL NETWORK VÀ BÀI TOÁN NHẬN DẠNG ẢNH

4.1. Tổng quan về mạng nơ-ron tích chập (CNN)

Convolutional Neural Network (CNN – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning. Nó giúp cho chúng ta xây dựng được hệ thống thông minh với độ chính xác khá cao. Như hệ thống xử lý ảnh lớn như Facebook, Google, Amazon,... đã đưa vào sản phẩm của mình những chức năng thông minh như nhận diện khuôn mặt người dùng, phát triển xe tự lái, drone giao hàng tự động. Đáng chú ý là sự phát triển của FaceID – nhận diện mở khóa bằng khuôn mặt do các “ông lớn” trong lĩnh vực di động phát triển như Iphone, Huawei,...

CNN được sử dụng nhiều trong các bài toán nhận dạng các vật thể trong ảnh. Để tìm hiểu tại sao thuật toán này được sử dụng rộng rãi, chúng ta hãy cùng tìm hiểu thuật toán này

4.2. Cấu trúc của CNN

Mạng CNN là một tập hợp bao gồm các lớp của Convolution chồng lên nhau và sử dụng các hàm để kích hoạt các trọng số trong các node như ReLU (Tinh chỉnh các đơn vị tuyến tính) để tạo ra thông tin trừu tượng hơn. Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo. Lớp tiếp theo là kết quả của lớp trước đó, nhờ vậy mà ta có được các kết nối cục bộ. Như vậy, mỗi nơ-ron ở lớp kế tiếp sinh ra từ kết quả của kernel áp đặt lên một vùng ảnh cục bộ của nơ-ron trước đó.

Trong mô hình Feedforward Neural Network (mạng nơ-ron truyền thẳng), các layer kết nối trực tiếp với nhau thông qua một trọng số w (weighted vector). Các layer này còn được gọi là có kết nối đầy đủ (fully connected layer) hay affine layer.

Mỗi lớp như vậy được đặt lên các kernel khác nhau, thông thường có vài trăm đến vài nghìn kernel như vậy. Một số lớp khác như pooling layer dùng để chắt lọc lại các thông tin hữu ích hơn. Pooling bao gồm việc duyệt từng bước trong một ô vuông cửa sổ dọc trên một hình ảnh và lấy giá trị lớn nhất từ cửa sổ ở mỗi bước. Sau khi pooling, một hình ảnh sẽ có khoảng một phần tư số điểm ảnh ban đầu. Một layer pooling là hoạt động

thực hiện pooling trên một hình ảnh hoặc một tập các hình ảnh. Đầu ra sẽ có cùng số lượng hình ảnh, nhưng mỗi cái sẽ có điểm ảnh ít hơn. Điều này cũng rất hữu ích trong việc quản lý tải trọng tính toán.

Trong suốt quá trình huấn luyện, CNN sẽ tự động học được các thông số cho các filter. Ví dụ trong tác vụ phân lớp ảnh, CNN sẽ cố gắng tìm ra thông số tối ưu cho các filter tương ứng thứ tự raw pixel > edges > shapes > facial > high-level features. Layer cuối cùng được dùng để phân lớp ảnh.

4.2.1. Convolution layer

Giả sử input của 1 convolution layer tổng quát là một tensor $H * W * D$. Kernel có kích thước $F * F * D$, số ô trượt: stride S và viền padding P . Convolutional layer áp dụng K kernel. Thì output của layer là tensor có kích thước:

$$\left(\frac{H - F + 2P}{S} + 1\right) * \left(\frac{W - F + 2P}{S} + 1\right) * D$$

Lưu ý:

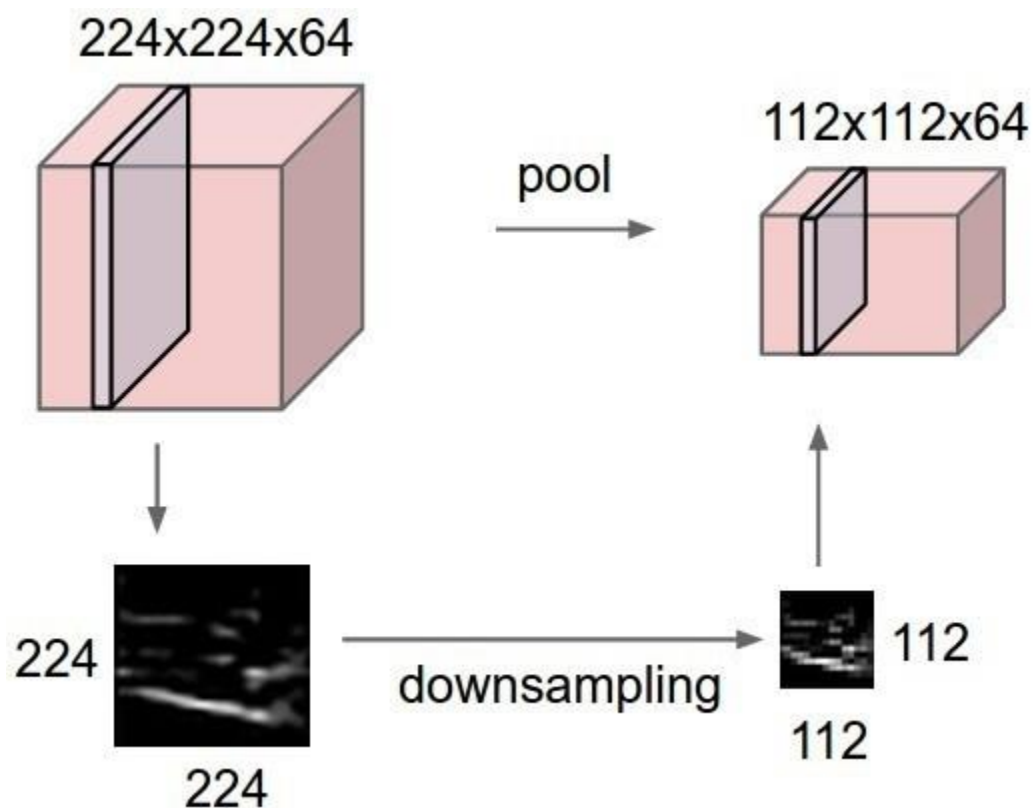
- Output của convolutional layer sẽ qua hàm activation function (xem lại phần 3.2.1) trước khi trở thành input của convolutional layer tiếp theo.
- Do mỗi kernel có kích thước $F * F * D$ và có 1 hệ số bias, nên tổng tham số của kernel là $F * F * D + 1$. Mà convolutional layer áp dụng K kernel vậy nên tổng số tham số có trong layer này là $K * (F * F * D + 1)$
- Ở trên là công thức ví dụ đối với một tensor 3 chiều, tương tự đối với ma trận 2 chiều với tham số D không có.

4.2.2. Pooling layer

Pooling layer thường được dùng giữa các convolutional layer để giảm kích thước dữ liệu nhưng vẫn giữ được thuộc tính quan trọng. Kích thước dữ liệu giảm nhằm làm giảm việc tính toán.

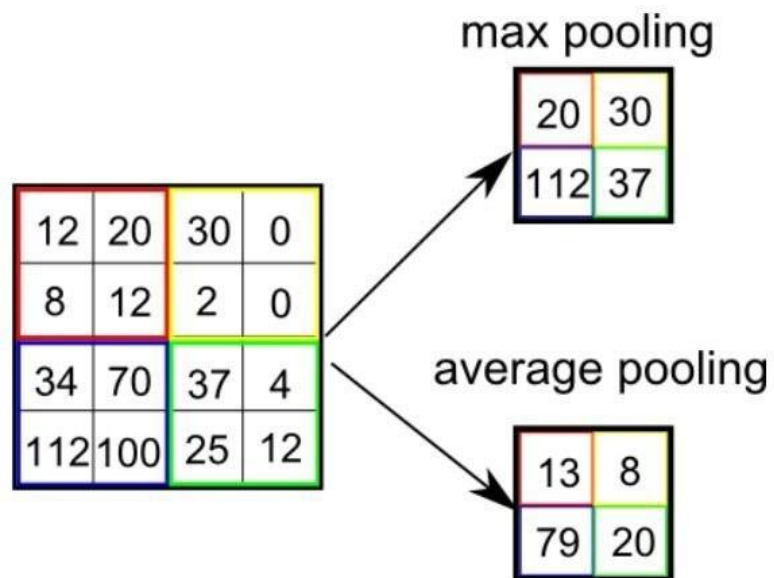
Ví dụ ta có một pooling kích thước $K * K$. Input của pooling layer có kích thước $H * W * D$, ta tách ra làm D ma trận kích thước $H * W$. Với mỗi ma trận, trên vùng kích thước $K * K$ trên ma trận, ta tìm ra giá trị lớn nhất hoặc giá trị trung bình của dữ liệu rồi viết vào ma trận kết quả. Hầu hết pooling layer có kích thước $2 * 2$, số ô trượt 2 và viền 0.

Các quy tắc về output cũng tính như convolution layer với số ô trượt và viền tương ứng. khi đó, output của ma trận sẽ giảm đi một nửa, chiều sâu D thì vẫn giữ nguyên.



Hình 4.1. Pooling đối với ảnh màu (tensor 3 chiều)

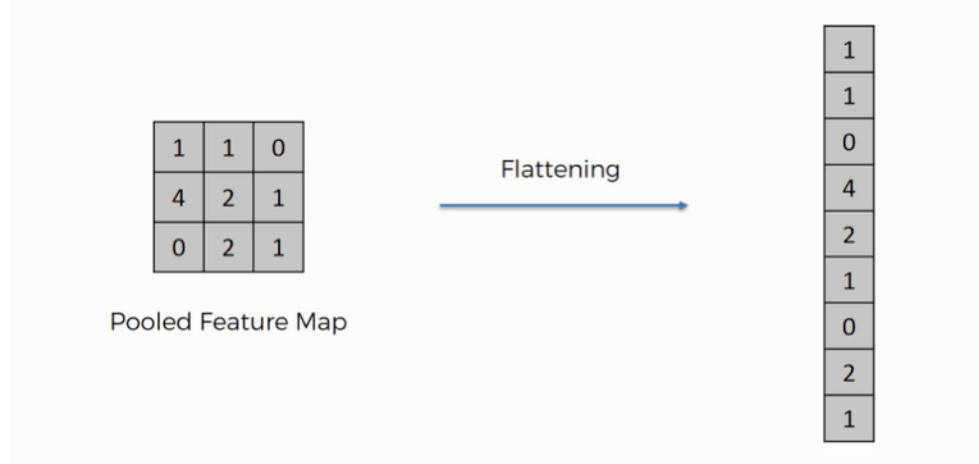
Đối với ảnh xám thì do chỉ là ma trận 2 chiều nên pooling sẽ đơn giản hơn



Hình 4.2. Pooling đối với ảnh xám (ma trận 2 chiều)

4.2.3. Fully connected layer

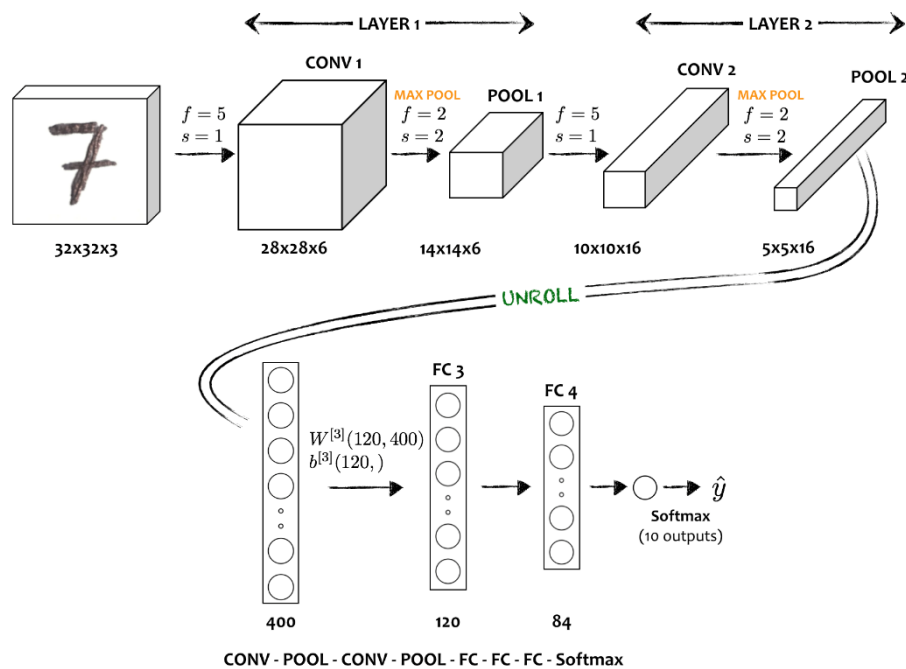
Sau khi ảnh được truyền qua các convolutional layer và pooling layer thì model đã học được tương đối các đặc điểm của ảnh thì tensor output của layer cuối cùng có kích thước $H*W*D$, sẽ được chuyển về 1 vector kích thước $H*W*D$ sau đó ta dùng các fully connected layer để kết hợp các đặc điểm của ảnh để ra đc output của model



Hình 4.3. Fully connected layer

4.2.4. Mô hình CNN

Kết hợp các phần trên, ta có được cấu trúc cơ bản của một mô hình mạng lưới nơ-ron tích chập: Ảnh đầu vào >> Convolutional layer + Pooling layer >> Fully connected layer >> output đầu ra.



Hình 4.4. Cấu trúc mô hình CNN

CHƯƠNG 5: NHẬN DẠNG VĂN BẢN VỚI CNN

5.1. Đặt vấn đề bài toán

Mục đích được đặt ra của bài toán là cho một ảnh đầu vào, dùng các phép biến đổi xử lý ảnh và nhận dạng ra được chữ cái trong ảnh đó. Ví dụ cho ảnh đầu vào

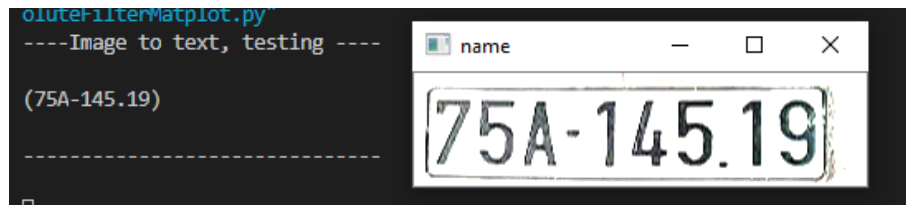


Hình 5.1. Ảnh đầu vào

Thì input phải nhận ra được các ký tự trong ảnh, sau đây là một ví dụ về output chấp nhận được và chưa chấp nhận được:



Hình 5.2. Hình chưa chấp nhận được



Hình 5.3. Hình chấp nhận được

5.2. Mô hình tổng quát

Trong mô hình Feedforward Neural Network (Mạng nơ-ron truyền ngược) có nhắc ở phần 3.3, các layer kết nối trực tiếp với nhau thông qua một trọng số w . Các layer này còn được gọi là có kết nối đầy đủ (fully connected layer).

Trong mô hình CNN thì ngược lại. Các layer liên kết được với nhau thông qua cơ chế convolution. Layer tiếp theo là kết quả của convolution từ layer trước đó, nhờ vậy mà ta có được kết nối cục bộ. Nghĩa là mỗi nơ-ron ở layer tiếp theo sinh ra từ filter áp đặt lên một vùng ảnh cục bộ của nơ-ron layer trước đó.

Mỗi layer như vậy được áp đặt qua mỗi filter khác nhau, một số layer khác như pooling layer sẽ chắt lọc thông tin hoặc giảm độ phức tạp của bài toán bằng cách lược bỏ đi những phần nhiễu.

Trong suốt quá trình huấn luyện, CNN sẽ tự động học được các thông số cho các filter.

Bước 1: Nhận ảnh đầu vào, đi qua tiền xử lý

Bước 2: Chuyển đổi ảnh sang ma trận hoặc tensor

Bước 3: Xây dựng mạng nơ-ron

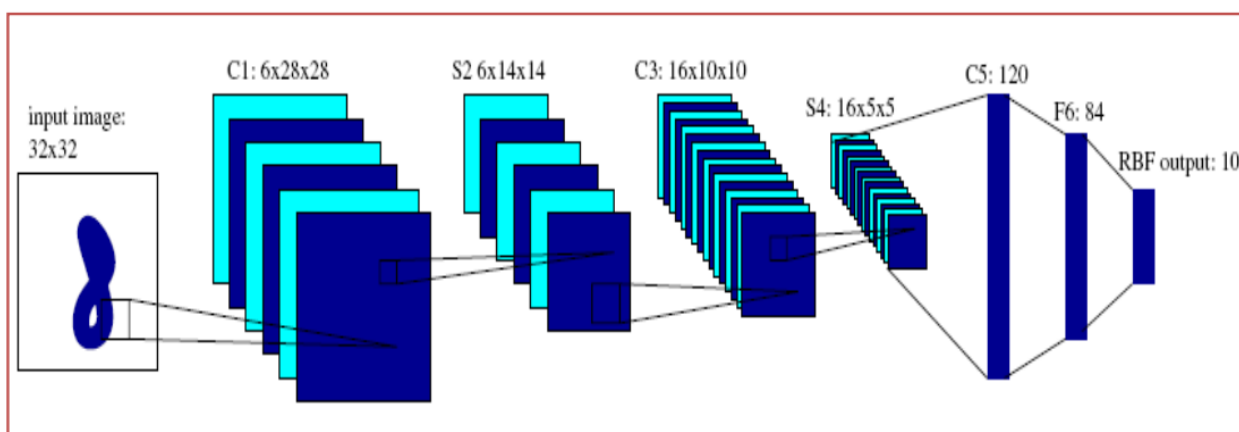
Bước 4: Kết quả

5.3. Xây dựng mạng nơ-ron nhận dạng ký tự

5.3.1. Xây dựng mạng nơ-ron

Sau khi tách được các ký tự từ ảnh, ta cần nhận dạng các ký tự từ ảnh rồi chuyển chúng sang text. Để nhận dạng được các ký tự có rất nhiều phương pháp nhận dạng, có thể là KNN, SVM, mạng nơ-ron lan truyền ngược,... Ở đây chúng ta sử dụng mạng neural lan tích chập trong nhận dạng ký tự số nguyên do mạng nơ-ron tích chập có độ chính xác cao, và hiệu năng tốc độ xử lý tính toán nhanh hơn các mạng trước đó.

Mô hình CNN được xây dựng để nhận dạng các ký tự như sau:



Hình 5.4. Mô hình CNN trong nhận dạng số viết tay

Ảnh đầu vào là một bức ảnh thô kích thước 32*32 pixel. Chúng ta sử dụng tensor có kích thước 5*5*6 (hoặc 6 ma trận chập kích thước 5*5) cho ra một tensor (hoặc 6 ma

trận) ảnh đặc trưng sau khi chập lần 1 đó là các ma trận ánh xạ đặc trưng ở tầng chập thứ nhất. Mỗi ma trận ánh xạ đặc trưng này có kích thước $28*28$ (tính theo công thức ở phần 4.2.1). Tức là ảnh gốc ban đầu được phân tích theo 6 chiều đặc trưng khác nhau với ma trận chập $5*5$.

Do kích thước ảnh đặc trưng ở tầng 1 có kích thước $28*28$ còn lớn, nên bước tiếp theo chúng ta thực hiện phép giảm số chiều ở ma trận đặc trưng (max pooling – xem lại phần 4.2.2). Như vậy, với 6 ma trận đặc trưng ban đầu kích thước $28*28$ ở tầng 1, ta tạo được 6 ma trận kích thước $14*14$ ở tầng pooling layer (tầng pooling 2).

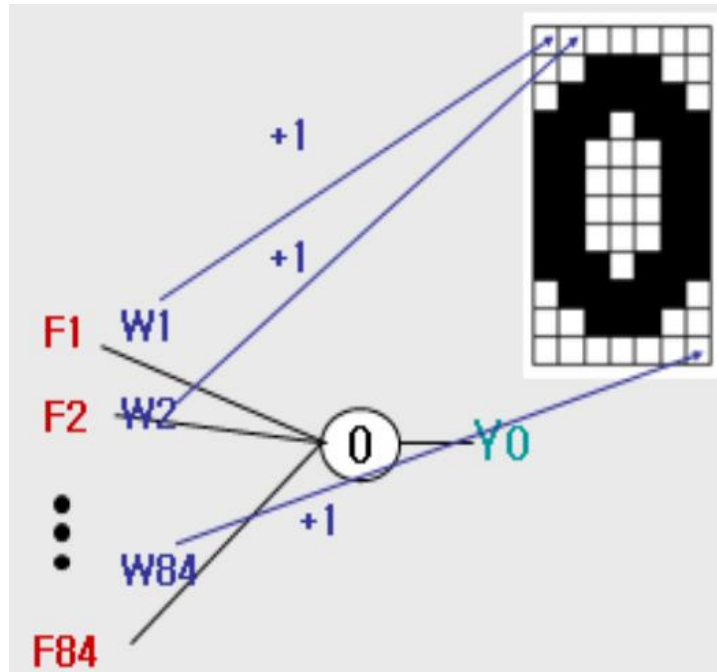
Tiếp tục sử dụng 16 ma trận chập kích thước $5*5$ chập với các ma trận tầng pooling layer ta được 16 ma trận ánh xạ đặc trưng kích thước $10*10$ ở tầng chập thứ 3.

Do kích thước ảnh còn lớn ở tầng chập thứ 3 có kích thước $14*14$, ta tiếp tục thực hiện max pooling. Kết quả với 16 ma trận đặc trưng kích thước $10*10$ ở tầng chập thứ 3 ta tạo được 16 ma trận kích thước $5*5$ ở tầng pooling thứ 4.

Tiếp tục sử dụng 120 ma trận chập kích thước $5*5$ chập với các ma trận ở tầng pooling thứ 4 ta được 120 ma trận ánh xạ đặc trưng kích thước $1*1$ ở tầng chập thứ 5.

Do các đặc trưng ở tầng chập thứ 5 là các đặc điểm đặc trưng $1*1$, cho nên ta không thực hiện phép toán max pooling lần nữa. Tiếp theo ta sử dụng phép toán max để giảm kích thước ở tầng chập thứ 5 do tầng chập thứ 5 có tới 120 node, ta dùng hàm max để giảm kích thước ma trận xuống còn 84 node ở tầng thứ 6.

Mỗi khối ở các tầng ta sử dụng hàm activation function (xem lại ở phần 3.2.1) để tính toán giá trị ra của các node mạng. Với 84 node ở tầng thứ 6, ở đây sử dụng mô hình mạng nơ-ron truyền thẳng với các kết nối fully connection, với 10 output:



Hình 5.5. Minh họa fully connection

5.4. Xây dựng chương trình thử nghiệm

5.4.1. Giới thiệu Python

Hiện nay, xử lý ảnh đang là một lĩnh vực mà rất nhiều người quan tâm, nghiên cứu. Nhờ vào sự phát triển mạnh mẽ của Machine Learning - một lĩnh vực nhỏ của Khoa Học Máy Tính, nó có khả năng tự học hỏi dựa trên dữ liệu đưa vào mà không cần phải được lập trình cụ thể, xử lý ảnh đã và đang được ứng dụng vào nhiều lĩnh vực trong cuộc sống: y tế (X Ray Imaging, PET scan,...), thị giác máy tính (giúp máy tính có thể hiểu, nhận biết đồ vật như con người), các công nghệ nhận dạng (vân tay, khuôn mặt,...),...

5.4.2. Tổng quan về thư viện OpenCV và Tesseract

OpenCV là một thư viện mã nguồn mở hàng đầu cho thị giác máy tính xử lý ảnh và máy học, và các tính năng tăng tốc GPU trong hoạt động thời gian thực.

Tesseract là thư viện OCR nổi tiếng do độ chính xác cao hơn hẳn các thư viện khác. Tesseract có thể chạy độc lập hoặc tích hợp với OpenCV đều được. Nếu chạy độc lập thì Tesseract sử dụng thư viện leptonica để đọc hình ảnh.

5.4.3. Xây dựng code mẫu

5.4.3.1. Code dùng python học nhận dạng

Chúng ta dùng 2 file code python để code trong đó: **main.py** dùng để code thực hiện huấn luyện mô hình, **model.py** dùng để chứa mô hình network được dùng trong **main.py**.

```
import torch
import matplotlib.pyplot as plt
import numpy as np
import torchvision

from torch import nn, optim
from torchvision import datasets, transforms

batch_size = 32 #lượng ảnh dùng trong 1 batch
learning_rate = 0.01 #tỉ lệ học được
num_epochs = 20 #số lần sử dụng toàn bộ dữ liệu để train

#dataloader
#load package dataset.MNIST có trong torchvision để tải về bộ dữ
#liệu vào mục data

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST("data", train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.137, ), (0.3081, ))
                   ])
    ),
    batch_size=batch_size,
    shuffle=True
)

val_loader = torch.utils.data.DataLoader(
    datasets.MNIST("data", train=False, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.137, ), (0.3081, )),
                   ])),
    batch_size=batch_size,
    shuffle=False
)
```

```

# #hiển thị dữ liệu
# def imshow(img, mean, std):
#     img = img / std + mean # unnormalize
#     npimg = img.numpy()
#     plt.imshow(np.transpose(npimg, (1, 2, 0)))
#     plt.show()

# dataiter = iter(train_loader)
# images, labels = dataiter.next()
# imshow(torchvision.utils.make_grid(images), 0.1307, 0.3081)
# print(labels)

#get device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

#model bằng class simple
from model import SimpleModel
model = SimpleModel().to(device)

#tính toán tổn thất
criterion = nn.CrossEntropyLoss()
#optimizer
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

#training
num_steps = len(train_loader)

for epoch in range(num_epochs):
    #----training----
    #set model to training

    model.train()

    total_loss = 0

    for i, (images, labels) in enumerate(train_loader):
        images, labels = images.to(device), labels.to(device)

        #zero gradients để xóa hết gradient của batch cũ trước khi
        #gradient batch hiện tại

```

```

optimizer.zero_grad()

#forward
outputs = model(images)

#tính toán tổn thất
loss = criterion(outputs, labels)

#backward
loss.backward()
optimizer.step()

total_loss += loss.item()

#print log
if (i+1)&100 ==0:
    print("epoch {}/{} - step: {}/{} - loss: {:.4f}".format
at(
        epoch, num_epochs, i, num_steps, total_loss/(i+1)
    ))
#----Kiểm định----
#set model to evaluating
model.eval()

val_losses = 0

with torch.no_grad():
    correct =0
    total =0
    for _, (images,labels) in enumerate(val_loader):
        images, labels = images.to(device), labels.to(device)

        outputs = model(images)

        _, predicted = torch.max(outputs,1)

        loss = criterion(outputs,labels)

        val_losses += loss.item()

        total += labels.size(0)
        correct += (predicted == labels).sum().item()

```

```

        print("epoch {} - accuracy: {} - validation loss: {:.4f}"
              .format(
                  epoch, correct / total, val_losses / (len(val_loader)
                ))

```

Hình 5.6. code trong main.py

```

from torch import nn

#mô hình mạng dùng bên main.py
class SimpleModel(nn.Module): #class kế thừa từ nn.Module
    def __init__(self, num_classes=10):
        super(SimpleModel, self).__init__()

        #layer conv gồm 3 bước:
        # - convolution
        # - hàm activation funcion
        # - max pooling
        self.conv1 = nn.Sequential(
            #Bước 1: Convolution input là 1 chanel, output là
            32 chanel

            #,kernel 5*5 viền = 2, trượt 1 ô
            nn.Conv2d(1, 32, kernel_size=5, padding=2, stride
            =1),

            #hàm actiovation function
            nn.ReLU(),
            #max pooling size =2, trượt 2
            nn.MaxPool2d(kernel_size=2, stride=2))

        #layer Fully connected
        self.fc = nn.Linear(14 * 14 * 32, num_classes)

    def forward(self, x):
        out = self.conv1(x)
        out = out.view(out.size(0), -1)
        out = self.fc(out)

        return out

```

Hình 5.7. Code trong model.py

5.4.3.2. Code nhận dạng chữ số từ hình ảnh

```
from PIL import Image
from pathlib import Path
import matplotlib.pyplot as plt
import pytesseract as ptr
import cv2
import numpy as np
import os

#thêm thư viện tesseract
ptr.pytesseract.tesseract_cmd = r'C:\Program Files (x86)\Tesseract-OCR\tesseract.exe'
img_path = 'testSoXe_cropped.png'

#xám hóa ảnh đầu vào
img = cv2.imread(img_path, cv2.COLOR_BGR2GRAY)

identity = np.array((
    [0, 0, 0],
    [0, 1, 0],
    [0, 0, 0]))

edge = np.array((
    [-1, -1, -1],
    [-1, 8, -1],
    [-1, -1, -1]))

boxblur = (1.0 / 9) * np.array(
    [[1, 1, 1],
     [1, 1, 1],
     [1, 1, 1]])

gaussian = (1.0 / 16) * np.array(
    [[1, 2, 1],
     [2, 4, 2],
     [1, 2, 1]])

emboss = np.array(
    [[-2, -1, 0],
     [-1, 1, 1],
     [0, 1, 2]])
```

```

square = np.array(
    [[ 0,  2,  0],
     [-2, -1,  2],
     [ 0, -2,  0]])

smallBlur = np.ones((7, 7), dtype="float") * (1.0 / (7 * 7))
largeBlur = np.ones((21, 21), dtype="float") * (1.0 / (21 * 2
1))

# construct a sharpening filter
sharpen = np.array((
    [0, -1, 0],
    [-1, 5, -1],
    [0, -1, 0]))

laplacian = (1.0 / 16) * np.array(
    [[ 0,  0, -1,  0,  0],
     [ 0, -1, -2, -1,  0],
     [-1, -2, 16, -2, -1],
     [ 0, -1, -2, -1,  0],
     [ 0,  0, -1,  0,  0]])

sobelLeft = np.array((
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]))

sobelRight = np.array((
    [1, 0, -1],
    [2, 0, -2],
    [1, 0, -1]))

sobelTop = np.array((
    [-1, -2, -1],
    [ 0,  0,  0],
    [ 1,  2,  1]))

sobelBottom = np.array((
    [ 1,  2,  1],
    [ 0,  0,  0],
    [-1, -2, -1]))

```



```

filters = [
    ("Identity", identity), #0
    ("Edge", edge), #1
    ("Box Blur", boxblur), #2
    ("Square", square), #3
    ("Gaussian", gaussian), #4
    ("Emboss", emboss), #5
    ("Small blur", smallBlur), #6
    ("Large blur", largeBlur), #7
    ("Sharpen", sharpen), #8
    ("Laplacian", laplacian), #9
    ('Sobel Left', sobelLeft), #10
    ('Sobel Right', sobelRight), #11
    ('Sobel Top', sobelTop), #12
    ('Sobel Bottom', sobelBottom) #13
]

##-----show all filters-----
# fig = plt.figure(figsize=(12, 8))
# fig.subplots_adjust(hspace=0.3, wspace=0.1)

# for i, filter in enumerate(filters):
#     axes = fig.add_subplot(3, 5, i+1)
#     axes.set(title=filter[0])
#     axes.grid(False)
#     axes.set_xticks([])
#     axes.set_yticks([])
#     img_out = cv2.filter2D(img, 0, filter[1])
#     axes.imshow(img_out, cmap='gray', vmin=0, vmax=255)
#     print("-----Filter: "+filter[0]+"-----\n")
#     print(ptr.image_to_string(img_out))
#     print("\n-----")
# plt.show()

##-----show từng filter-----
# def show(i, filter, name):
#     img_out = cv2.filter2D(img, 0, filter[1])
#     name = filter[0]
#     print("-----Filter: "+name+"-----\n")

```

```

#     print(ptr.image_to_string(img_out))
#     print("\n-----")

#     cv2.imshow(name, img_out)
#     cv2.waitKey(0)

# for i, filter in enumerate(filters):
#     name = filter[0]
#     show(i,filter, name)

#qua các filter
img_out = cv2.filter2D(img, 0,emboss[1])
img_out = cv2.filter2D(img_out, 0,gaussian[1])
img_out = cv2.filter2D(img_out, 0,sharpen[1])
img_out = cv2.filter2D(img_out, 0,boxblur[1])
img_out = cv2.filter2D(img_out, 0,edge[1])

print("----Image to text, testing ----\n")
print(ptr.image_to_string(img_out))
print("\n-----\n")
cv2.imshow("name", img_out)
cv2.waitKey(0)

```

Hình 5.8. Code mẫu nhận diện ảnh dùng pytesseract

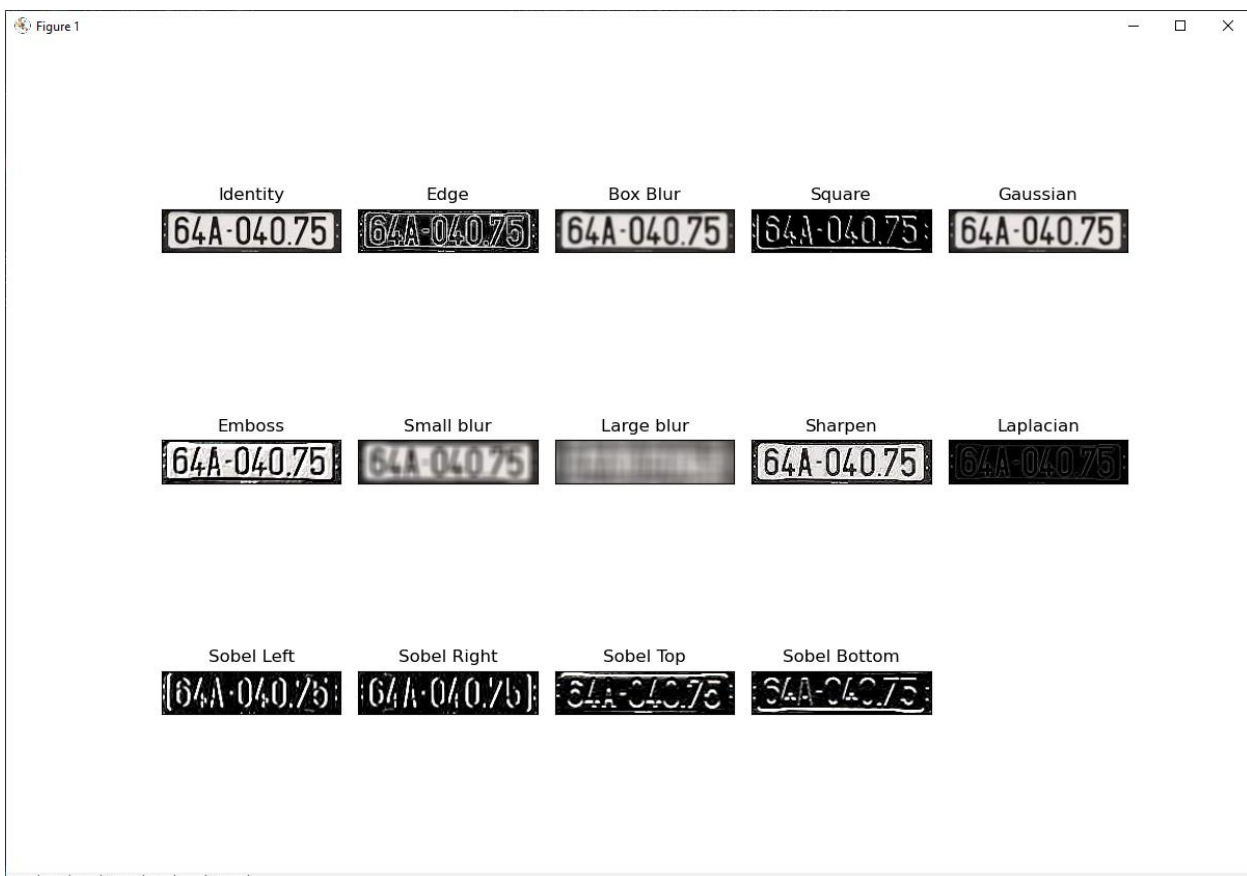
5.5. Kết quả thu được

Bảng 5.1. Kết quả thu được với các filter được tô đậm là các filter chấp nhận được

STT	TÊN FILTER	KẾT QUẢ NHẬN ĐƯỢC	KẾT QUẢ MONG MUỐN
1	Identity	164A-040.75 f	64A-040.75
2	Edge		
3	Box blur	64A-040.75	
4	Square		
5	Gaussian	64A-040.75	
6	Emboss	'64A-040.758	
7	Small blur	[BLA OLO 751	
8	Large blur		
9	Sharpen	64A-040.75	
10	Laplacian		
11	Sobel left	{841-040.75:	
12	Sobel right	040.75}	
13	Sobel top		
14	Sobel bottom		



Hình 5.9. Input đầu vào



Hình 5.10. Các kết quả qua các filter

5.6. Hạn chế

Tuy có nhiều filter khiến kết quả đúng ngay, không cần qua thêm nhiều filter nhưng đối với các ảnh khác nhau phải có các bước qua các filter khác nhau nhằm loại nhiễu và thu được kết quả tốt nhất.

CHƯƠNG 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Công nghệ thông tin ngày càng phát triển, kèm thêm sự bùng nổ của trí tuệ nhân tạo của cách mạng công nghiệp lần thứ 4, mọi việc đều có thể số hóa để giảm bớt gánh nặng công sức của con người. Việc nhận dạng ký tự từ hình ảnh không chỉ giúp chúng ta trong việc chuyển các văn bản từ hình ảnh đơn thuần, mà nó còn giúp trong các việc như giữ xe, các camera sẽ lấy thông tin biển số xe rồi lưu trực tiếp vào cơ sở dữ liệu. Hoặc trong vấn đề an ninh, việc đọc được văn bản từ hình ảnh giúp đọc được biển số xe, giúp các cán bộ công an phạt nguội, minh bạch, rõ ràng. Còn trong việc học, việc ứng dụng nhận dạng văn bản từ hình ảnh giúp học sinh có thể chụp ảnh rồi chuyển sang văn bản một cách dễ dàng hơn.

Về mở rộng hơn, việc trích xuất văn bản từ hình ảnh cũng có thể giúp chúng ta giải toán nhanh hơn. Thử nghĩ xem, chúng ta chỉ cần dùng điện thoại, chụp ảnh một bài toán khó và chỉ trong vài giây, ta đã có được kết quả, thậm chí còn có thể có được cả lời giải. Con người chúng ta là một tạo vật phi thường, không ngừng phát triển, chỉ cần chúng ta có ý tưởng thì sẽ có thể thành sự thật nếu chúng ta nỗ lực không ngừng. Vì thế, hướng mở rộng trong việc nhận dạng văn bản từ hình ảnh cực kì rộng lớn, như chuyển đổi file PDF thành file Word, dịch văn bản trực tiếp bằng cách sử dụng camera, ... Nhận diện văn bản nói riêng và nhận diện hình ảnh nói chung có thể giúp chúng ta rất nhiều.

TÀI LIỆU THAM KHẢO

- http://lib.uet.vnu.edu.vn/bitstream/123456789/918/1/K20_KTPM_Le_Thi_T_hu_Hang_luanvan.pdf
- <https://drive.google.com/file/d/1UQQ5L2KqCgy5P7Z2hmiuzeCQg4Oqqexu/view>
- <https://minhng.info/tutorials/opencv-anh-nhi-phan.html>
- <https://www.slideshare.net/elLeonNo1/gio-trnh-x-l-nh>
- <https://techmaster.vn/posts/35477/convolution-xu-ly-anh-qua-vi-du-python-thuc-te>
- <https://topdev.vn/blog/thuat-toan-cnn-convolutional-neural-network/>
- <https://deeplearning.vn/post/nhan-dien-chu-viet-pytorch/>