

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC BÁCH KHOA**  
**Khoa Khoa học - Kỹ thuật Máy tính**



**MÔN: HỆ THỐNG THÔNG MINH**

---

**ĐỀ TÀI: NHẬN DẠNG CHỮ SỐ VIẾT TAY**

---

GVHD: PGS.TS. Quấn Thành Thơ

SV thực hiện: Tô Duy Hưng – 1810198  
Lê Đức Huy – 1810166  
Đỗ Lê Quang Trung – 1811304

**TP. HỒ CHÍ MINH, THÁNG 04/2021**

## Mục lục

<b>1</b>	<b>Giới thiệu mạng nơ-ron tích chập (Convolutional Neural Network)</b>	<b>2</b>
1.1	Lịch sử mạng nơ-ron tích chập . . . . .	2
1.2	Đặc tính cơ bản của mạng nơ-ron tích chập . . . . .	2
<b>2</b>	<b>Cách thức hoạt động của mạng nơ-ron tích chập</b>	<b>3</b>
2.1	Phép tích chập (Convolution) . . . . .	3
2.2	Stride . . . . .	4
2.3	Padding . . . . .	5
2.4	Phép gộp (Pooling) . . . . .	6
2.5	Ý nghĩa của ma trận kết quả . . . . .	6
2.6	Bản đồ thuộc tính (Feature map) . . . . .	6
<b>3</b>	<b>Lớp phi tuyến tính</b>	<b>7</b>
<b>4</b>	<b>Lớp tổng hợp</b>	<b>7</b>
<b>5</b>	<b>Lớp kết nối đầy đủ</b>	<b>8</b>
<b>6</b>	<b>Mạng nơ-ron tích chập dưới góc nhìn của một mạng nơ-ron nhân tạo</b>	<b>8</b>
<b>7</b>	<b>Triển khai một mạng CNN</b>	<b>9</b>
<b>8</b>	<b>Case study: sử dụng CNN cho bài toán nhận diện chữ số viết tay (Handwriting Recognition)</b>	<b>11</b>
<b>9</b>	<b>Kết quả</b>	<b>14</b>
9.1	Nhận diện số từ 0 đến 9 . . . . .	14
9.2	Những trường hợp viết số không cụ thể . . . . .	15

# 1 Giới thiệu mạng nơ-ron tích chập (Convolutional Neural Network)

## 1.1 Lịch sử mạng nơ-ron tích chập

Mạng Nơ-ron tích chập (Convolutional Neural Network, còn được viết tắt là CNN hay ConvNet) là một trong những nền tảng quan trọng của chuyên ngành Thị giác máy tính (Computer Vision) nói riêng, hay của ngành Học sâu (Deep learning) nói chung. CNN được chuyên dùng cho các bài toán liên quan đến hình ảnh như phân loại, phân tích hình ảnh (image classification) hay nhận diện khuôn mặt (facial recognition).

Không nhiều người biết rằng thiết kế của mạng nơ-ron tích chập tuân theo quá trình xử lý thị giác trong các sinh vật sống. Lấy cảm hứng từ nghiên cứu của Hubel và Wiesel trong những năm 1950 và 1960 về các tế bào thần kinh phản ứng riêng với các kích thích thị giác ở mèo và khỉ, một khái niệm tên là "Neocognitron" đã được Kunihiko Fukushima giới thiệu vào năm 1980. Đây được xem là mạng nơ-ron tích chập đầu tiên xuất hiện trên thế giới. "Neocognitron" giới thiệu hai loại tầng cơ bản: một là tầng S-cells được sử dụng để trích xuất các thuộc tính cục bộ (local feature) của hình ảnh, tương đương với simple cell (dùng để xử lý các đường thẳng cơ bản); hai là C-cells được dùng để giảm thiểu các sai sót khi kết hợp nhiều thuộc tính cục bộ với nhau (tương đương với complex cell). Các thuộc tính cục bộ được tích hợp dần và phân loại ở các tầng sâu hơn. Từ đó máy tính sẽ nhận dạng được các đặc điểm khác nhau của hình ảnh trong quá trình học và có thể phân loại được hình ảnh dựa trên các bộ lọc (filter) khác nhau.

Từ đó, mạng nơ-ron tích chập ngày càng được phát triển và cho đến thời điểm hiện tại, khi nhắc về khái niệm này, nhiều người trong chúng ta sẽ nghĩ ngay đến hệ thống nhận dạng chữ viết tay cho zipcode của nhà nghiên cứu Yann LeCun. Ông ấy cùng các cộng sự đã sử dụng Lan truyền ngược (Backpropagation) để tìm hiểu các hệ số nhân tích chập trực tiếp từ hình ảnh của các số viết tay. Do đó, việc học được thực hiện hoàn toàn tự động, hoạt động tốt hơn so với thiết kế hệ số thủ công và phù hợp với một loạt các vấn đề nhận dạng hình ảnh và các loại hình ảnh. Cách tiếp cận này đã trở thành nền tảng của Thị giác máy tính hiện đại.

Có thể thấy, mặc dù CNN được phát minh vào những năm 1980, nhưng bước đột phá của chúng chỉ thực sự bắt đầu vào những năm 2000 với sự phát triển của ngành công nghiệp Trò chơi điện tử. Các card đồ họa ngày càng được nâng cấp để nâng cao trải nghiệm chơi game cho người dùng, mà cốt lõi chính là việc tăng cường khả năng tính toán của các bộ xử lý đồ họa (GPU- Graphic Processing Units). Thoạt nghe thì có vẻ CNN không dính dáng tới GPU lắm vì chúng thuộc 2 chuyên ngành khác biệt, nhưng thực tế chúng đều hoạt động dựa trên các thao tác với ma trận (matrix manipulation). Hình ảnh là một ma trận các điểm ảnh (pixel), còn video thì ghép nối nhiều hình ảnh khác nhau trong một khoảng thời gian nhất định (60 frame per second chẳng hạn) để tạo nên những chuyển động. Do tính chất này, GPU được cải thiện sao cho việc tính toán các ma trận pixel được diễn ra nhanh nhất có thể, điều này vô tình khiến các mạng nơ-ron nhân tạo được hưởng lợi rất nhiều. Theo đánh giá của các nhà khoa học thì hiện nay việc triển khai các mô hình mạng CNN trên GPU đạt tốc độ nhanh hơn 20 lần so với việc triển khai tương đương trên CPU.

## 1.2 Đặc tính cơ bản của mạng nơ-ron tích chập

Vậy tại sao mạng nơ-ron tích chập lại được sử dụng phổ biến cho các bài toán phân tích hình ảnh? Câu trả lời đến từ một bản chất tự nhiên của CNN, đó là khả năng rút trích thuộc tính ẩn (latent feature extraction) cực kỳ tốt.

Trái ngược với mô hình mạng nơ-ron truyền thẳng (Feedforward Neural Network) - mô hình mà ở đó các tầng kết nối đầy đủ trực tiếp với nhau thông qua một trọng số  $w$  (weighted vector) - thì ở CNN, các tầng liên kết được với nhau thông qua cơ chế tích chập (convolution). Tầng tiếp theo là kết quả tích chập từ tầng trước đó, nhờ vậy mà ta có được các kết nối cục bộ. Nghĩa là mỗi nơ-ron ở tầng tiếp theo sinh ra từ bộ lọc (filter) áp đặt lên một vùng ảnh cục bộ của nơ-ron tầng trước đó. Mỗi tầng như vậy được áp đặt các bộ lọc khác nhau, thông thường có vài trăm đến vài nghìn bộ như vậy. Một số tầng khác như tổng hợp (pooling) hay lấy mẫu con (subsampling) dùng để chắt lọc lại các thông tin hữu ích hơn (loại bỏ các thông tin nhiễu).

Trong suốt quá trình huấn luyện, CNN sẽ tự động học được các thông số cho các bộ lọc. Ví dụ trong tác vụ phân lớp ảnh, CNN sẽ cố gắng tìm ra thông số tối ưu cho các bộ lọc tương ứng theo thứ tự **raw pixel**  $\rightarrow$  **cạnh (edges)**  $\rightarrow$  **hình dạng (shapes)**  $\rightarrow$  **khuôn mặt (facial)**  $\rightarrow$  **các đặc trưng mức cao (high-level features)**. Còn lại tầng cuối cùng thì được dùng để phân lớp ảnh.

Mạng nơ-ron tích chập có tính bất biến và tính kết hợp cục bộ (Location Invariance and Compositionality). Với cùng một đối tượng, nếu đối tượng này được chiếu theo các góc độ khác nhau như phép dịch chuyển (translation), phép quay (rotation) và phép co giãn (scaling) thì độ chính xác của thuật toán sẽ bị

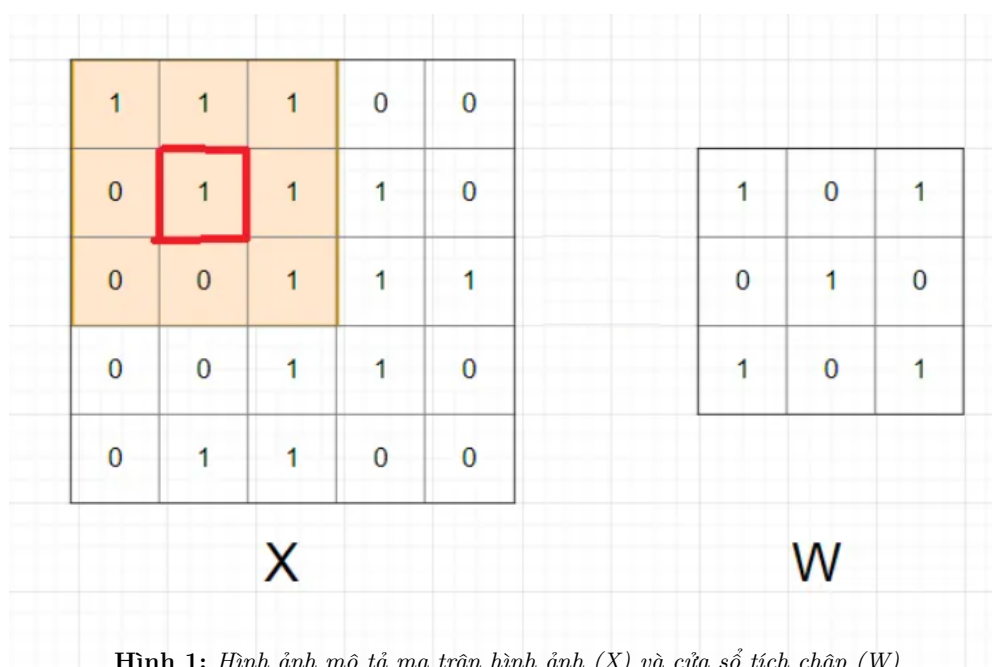
ảnh hưởng đáng kể. Tầng tổng hợp (pooling layer) sẽ cho ta tính bất biến đối với các phép biến hình đã nêu trên. Tính kết hợp cục bộ cho ta các cấp độ biểu diễn thông tin từ mức độ thấp đến mức độ cao và trừu tượng hơn thông qua việc tích chập từ các bộ lọc. Đó là lý do tại sao CNN cho ra mô hình với độ chính xác rất cao. Cách tiếp cận này cũng giống như cách con người nhận biết các vật thể trong tự nhiên. Lấy một ví dụ cụ thể về việc so sánh một con chó và một con mèo, thì cả con người và CNN đều có thể biết cách phân biệt nhờ vào các đặc trưng từ mức độ thấp (có 4 chân, có đuôi) đến mức độ cao (dáng đi, hình thể, màu lông).

## 2 Cách thức hoạt động của mạng nơ-ron tích chập

### 2.1 Phép tích chập (Convolution)

Đúng như cái tên của CNN, phép tính tích chập (convolution) chính là đặc trưng mạng nơ-ron này. Để có thể dễ dàng nắm bắt được ý tưởng của phép tích chập, ta hãy cùng phân tích ví dụ sau đây.

Xét một hình ảnh được biểu diễn dưới dạng ma trận  $X$  kích thước  $5 \times 5$  với các phần tử là chữ số 0 hoặc 1 và một cửa sổ tích chập  $W$  (cửa sổ trượt) là ma trận  $3 \times 3$  như hình 1.



**Hình 1:** Hình ảnh mô tả ma trận hình ảnh ( $X$ ) và cửa sổ tích chập ( $W$ )

Ta có cách tính một phần tử của ma trận kết quả từ phép tích chập như sau:

- Nhân tích chập cửa sổ trượt  $W$  với ma trận con  $3 \times 3$  tại vị trí đầu tiên của ma trận hình ảnh  $X$  - ma trận được tô màu cam trong Hình 1. Phép nhân tích chập được thực hiện bằng cách nhân từng phần tử của ma trận hình ảnh con với phần tử tại vị trí tương ứng của cửa sổ trượt sau đó cộng các kết quả lại. Kết quả cuối cùng chính là giá trị của phần tử đầu tiên trong ma trận kết quả  $Y$ .

Ví dụ nếu gọi  $x_{i,j}$ ,  $y_{i,j}$  và  $w_{i,j}$  lần lượt là các phần tử tại vị trí hàng  $i$  cột  $j$  của các ma trận  $X$ ,  $Y$ ,  $W$  thì  $y_{1,1} = x_{1,1} * w_{1,1} + x_{1,2} * w_{1,2} + x_{1,3} * w_{1,3} + x_{2,1} * w_{2,1} + x_{2,2} * w_{2,2} + x_{2,3} * w_{2,3} + x_{3,1} * w_{3,1} + x_{3,2} * w_{3,2} + x_{3,3} * w_{3,3} = 1 * 1 + 1 * 0 + 1 * 1 + 0 * 0 + 1 * 1 + 1 * 0 + 0 * 1 + 0 * 0 + 1 * 1 = 4$

- Tiếp đó, trượt cửa sổ tích chập sang phải của ma trận hình ảnh stride đơn vị cột (khái niệm về stride sẽ được trình bày sau ở mục 2.2, trong ví dụ này ta xét trường hợp đơn giản nhất là stride bằng 1) và tiếp tục nhân tích chập để tìm được phần tử thứ 2 trong ma trận kết quả. Làm liên tục cho đến khi không thể trượt sang phải thêm được nữa, ta có được các phần tử của hàng đầu tiên trong ma trận kết quả tích chập. Kết quả minh họa ở Hình 2 và Hình 3.
- Để tính tiếp các hàng tiếp theo cho ma trận kết quả, ta dời cửa sổ trượt về vị trí đầu tiên - ma trận được tô màu cam trong Hình 1 - và trượt cửa sổ xuống stride đơn vị hàng. Tiếp tục lặp lại quá trình nhân tích chập và trượt cửa sổ. Ma trận kết quả sau khi hoàn thành quá trình tích chập sẽ có kích thước  $3 \times 3$  như hình 4.

1	1 <sub>x1</sub>	1 <sub>x0</sub>	0 <sub>x1</sub>	0
0	1 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	0
0	0 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	1
0	0	1	1	0
0	1	1	0	0

4	3	

Hình 2: Cách tính phần tử hàng 1 cột 2 của Y

1	1	1 <sub>x1</sub>	0 <sub>x0</sub>	0 <sub>x1</sub>
0	1	1 <sub>x0</sub>	1 <sub>x1</sub>	0 <sub>x0</sub>
0	0	1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>
0	0	1	1	0
0	1	1	0	0

4	3	4

Hình 3: Cách tính phần tử hàng 1 cột 3 của Y

1	1	1	0	0
0	1	1	1	0
0	0	1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>
0	0	1 <sub>x0</sub>	1 <sub>x1</sub>	0 <sub>x0</sub>
0	1	1 <sub>x1</sub>	0 <sub>x0</sub>	0 <sub>x1</sub>

4	3	4
2	4	3
2	3	4

Hình 4: Hình ảnh mô tả ma trận kết quả Y đầy đủ

Tổng quát hóa ta rút ra công thức sau

$$y_{ij} = \sum_{k=i}^{k+m-1} \sum_{l=j}^{l+m-1} (x_{k,l} * w_{k-i+1,l-j+1})$$




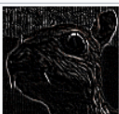

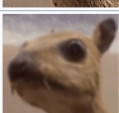
Khi đó, kích thước của ma trận Y là  $(m-k+1) * (n-k+1)$  với  $m * n$  là kích thước của ma trận ảnh đầu vào X. Áp dụng ngay cho ví dụ trên, ta có thể tính được kích thước của ma trận Y là  $(5 - 3 + 1) * (5 - 3 + 1) = 3 * 3$  mà không cần phải thực hiện chi tiết từng bước tích chập.

Như vậy, ta đã có thể nắm được quá trình nhân tích chập diễn ra như thế nào. Để thực hành làm quen với phép tích chập, hãy thử tiếp tục nhân tích chập ma trận hình ảnh ban đầu cho các cửa sổ trượt đặc biệt trong Hình 5.

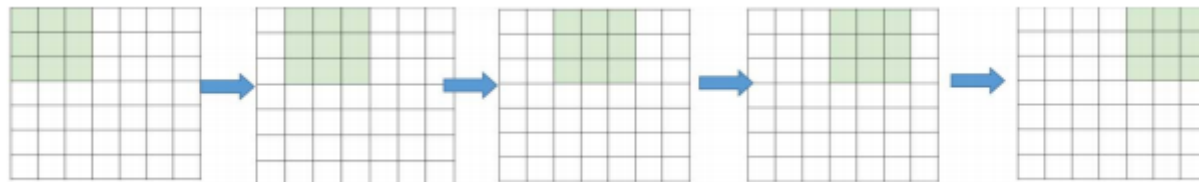
Những ma trận từ các cửa sổ trượt trên còn được gọi là bộ lọc (filter) bởi vì chúng hoạt động như các bộ lọc trong kỹ thuật xử lý ảnh, có thể được thiết lập để phát hiện các cạnh trong hình ảnh hay làm mờ hình ảnh,... Chính vì vậy, để mạng nơ-ron hoạt động hiệu quả hơn, ta có thể thêm vào nhiều tầng phía sau tầng đầu vào, mỗi tầng có thể được áp dụng các bộ lọc khác nhau. Nhờ đó, nó có thể trích xuất được nhiều đặc tính quan trọng từ các hình ảnh đầu vào.

## 2.2 Stride

Trong thực tế, lớp tích chập có nhiều thiết lập khác nhau cung cấp các khả năng để giảm số lượng các tham số, đồng thời làm giảm các tác dụng phụ, một trong số đó là *stride*. Trong các ví dụ được đề cập ở trên, ta giả sử ma trận tích chập trượt lần lượt trên hình ảnh đầu vào, mỗi lần di chuyển 1 đơn vị. Ta có thể thay đổi độ dời của ma trận tích chập mỗi lần di chuyển bằng cách thay đổi *stride*. Ta có thể thấy trong Hình 6, với ma trận đầu vào có kích thước 7x7, nếu ta di chuyển ma trận tích hợp 1 đơn vị mỗi lần cho đến hết thì ma trận kết quả sẽ có kích thước 5x5. Nếu ta thay đổi *stride*=2, tức là di chuyển ma trận tích hợp 2 đơn vị mỗi lần thì ma trận kết quả sẽ chỉ có kích thước 3x3.

Tác vụ (operation)	Kernel $w$	Ảnh kết quả
Ảnh gốc	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Nhận diện cạnh	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Làm nét ảnh	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Làm mờ ảnh	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Hình 5: Một số bộ lọc đặc biệt



Hình 6: Sử dụng  $stride=1$  với ma trận tích chập

Tổng quát hóa, với ma trận đầu vào có kích thước  $n * n$ , ma trận tích hợp có kích thước  $f * f$  và giá trị  $stride$  là  $s$  thì ma trận kết quả  $O$  sẽ có kích thước là:

$$(\lfloor \frac{n-f}{s} \rfloor + 1) * (\lfloor \frac{n-f}{s} \rfloor + 1)$$

## 2.3 Padding

Bởi vì sau mỗi lần thực hiện phép tích chập, kích thước ma trận đầu ra  $Y$  đều nhỏ kích thước của ma trận đầu vào  $X$  (do  $m - k + 1 < m$ ) nên một trong những hạn chế của ma trận tích chập là có thể làm mất đi thông tin quan trọng tồn tại trên đường viền của hình ảnh. Để khắc phục hạn chế này, một giải pháp đơn giản đã được đề ra là sử dụng kỹ thuật *zero padding*. Kỹ thuật này cho phép thêm vào các số 0 bên ngoài ma trận đầu vào nhằm tạo ra đường viền ảo cho hình ảnh, giúp ngăn chặn việc kích thước đầu ra giảm dần khi tới các lớp sâu hơn trong mạng nơ-ron, cho phép triển khai các mạng học sâu phức tạp với nhiều tầng tích chập.

Hình là một ví dụ minh họa trực quan cho trường hợp  $padding = 1$ . Ma trận  $X$  trước khi thực hiện phép tích chập đã được thêm 1 viền số 0 bên ngoài, nhờ đó kích thước ma trận  $Y$  thu được sau khi tích chập với kernel có giá trị là  $(7 - 3 + 1) * (7 - 3 + 1) = 5 * 5$ , bằng với kích thước ma trận  $X$  ban đầu. Tổng quát hóa với  $padding = p$ , ta sẽ thêm  $p$  vector 0 vào mỗi phía của ma trận. Kết hợp với giả thiết ma trận đầu vào có kích thước  $n * n$ , tích chập với kernel kích thước  $f * f$  và giá trị  $stride = s$  thì ma trận kết quả  $Y$  sẽ có kích thước:

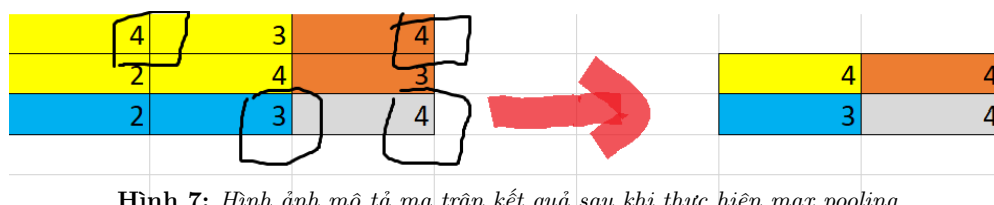
$$(\lfloor \frac{n-f+2p}{s} \rfloor + 1) * (\lfloor \frac{n-f+2p}{s} \rfloor + 1)$$

## 2.4 Phép gộp (Pooling)

Sau khi có được các ma trận kết quả với kích thước  $3 \times 3$  từ phép tính tích chập. Ta tiếp tục thực hiện phép gộp (pooling) để tiếp tục thu nhỏ ma trận hình ảnh. Có nhiều cách thực hiện phép gộp khác nhau như: lấy giá trị lớn nhất (max pooling), lấy giá trị trung bình (average pooling) hay lấy giá trị tổng (sum pooling). Trong thực tế, max pooling có khả năng khử nhiễu tốt hơn các loại pooling khác, vì vậy ta sẽ sử dụng max pooling để làm ví dụ minh họa cho phép gộp. Xét ma trận kết quả được tính từ cửa sổ tích gộp đầu tiên. Ta thực hiện phép gộp max pooling như sau:

- Chia ma trận kết quả ra thành các ma trận con (grid) có kích thước  $2 \times 2$ . Lưu ý là do ma trận kết quả có kích thước là  $3 \times 3$  nên các ma trận con nằm ở cuối hàng và cuối cột chỉ có kích thước  $1 \times 2$  hoặc  $2 \times 1$ .
- Giá trị lớn nhất trong mỗi ma trận con (khoanh vuông màu đen) chính là phần tử của ma trận gộp mới. Lúc này ma trận kết quả sau khi gộp chỉ còn có kích thước  $2 \times 2$ , giảm đáng kể so với ma trận  $5 \times 5$  ban đầu.

Hình 7 mô tả quá trình thực hiện phép gộp này. Và như vậy, ta đã nắm bắt được ý tưởng của phép gộp (pooling) trong mạng nơ-ron tích chập.



Hình 7: Hình ảnh mô tả ma trận kết quả sau khi thực hiện max pooling

## 2.5 Ý nghĩa của ma trận kết quả

Sau khi đã thực hiện phép tích chập (convolution) và phép gộp (pooling), ta có được ma trận kết quả kích thước  $2 \times 2$  là ma trận sau dấu mũi tên trong Hình 7. Giả sử nếu có thêm 3 cửa sổ trượt nữa, thực hiện tương tự các phép tích chập và phép gộp như đã nêu trên, ta sẽ có 4 ma trận kết quả.

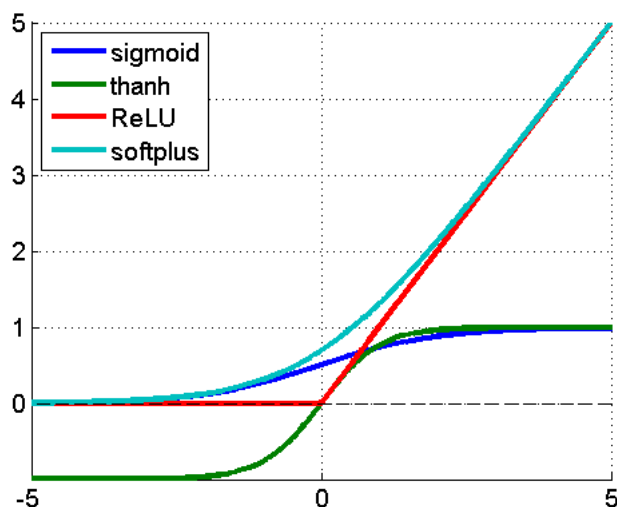
Chọn ra giá trị lớn nhất trong 4 ma trận trên, giả sử là  $k$ . Khi đó, vì nhận thấy rằng giá trị tối đa có thể có được của mỗi phần tử chỉ có thể là  $k$  nên ta sẽ thực hiện thêm một biến đổi cho 4 ma trận này bằng cách chuyển các giá trị  $k$  thành dấu x, các giá trị nhỏ hơn  $k$  đều được thay bằng 0. Sau phép biến đổi này, ta có được 4 ma trận mới.

Sau khi biến đổi thì các ma trận kết quả sẽ thể hiện được sự phân bố mẫu (pattern) của các cửa sổ trượt nhỏ trên ma trận hình ảnh ban đầu. Đó là lý do vì sao các cửa sổ trượt này được gọi là các bộ lọc (filter). Nhiệm vụ của chúng chính là tìm ra các đặc trưng của hình ảnh. Ở các lớp cao hơn, các bộ lọc đóng các vai trò khác nhau tùy theo yêu cầu của bài toán như bộ phát hiện cạnh (edge detector), bộ phát hiện nhiễu (noise detector),...

## 2.6 Bản đồ thuộc tính (Feature map)

Nếu ghép 4 ma trận kết quả đã nêu trên lại với nhau, ta được một hình ảnh gọi là bản đồ thuộc tính (feature map), được trích xuất sau quá trình tích chập của mạng CNN. Bản đồ thuộc tính là một đặc trưng nổi bật của mạng nơ-ron tích chập, bởi nó thể hiện được các thuộc tính tiềm ẩn (latent features) của hình ảnh đầu vào.

Cần lưu ý rằng bản đồ thuộc tính được sinh ra dựa trên các bộ lọc đã được định nghĩa trước (hay chính là các cửa sổ tích chập trong ví dụ trên). Do đó, điều quan trọng chính là làm sao định nghĩa được các bộ lọc này một cách phù hợp để áp dụng hiệu quả cho những bài toán cụ thể. Ví dụ như với bài toán nhận diện khuôn mặt người, bộ lọc sẽ là các bộ nhận diện mắt, mũi, miệng. Còn đối với bài toán nhận diện vạch kẻ đường, mạng nơ-ron cần học được các bộ lọc nhận diện đường thẳng,... Vấn đề này sẽ được bàn đến trong phần tiếp theo, khi ta cần sử dụng cơ chế học của mạng nơ-ron nhân tạo thông thường để có thể học được bộ lọc.



Hình 8: Các hàm phi tuyến tính phổ biến

### 3 Lớp phi tuyến tính

Lớp tiếp theo sau lớp tích chập mà ta sẽ giới thiệu là lớp phi tuyến tính. Lớp phi tuyến tính có thể được sử dụng để điều chỉnh hoặc cắt bỏ các giá trị trong ma trận đầu ra. Lớp này được áp dụng để bảo hòa hay giới hạn các giá trị được tạo ra trong ma trận đầu ra.

Trong suốt nhiều năm, các hàm phi tuyến tính được sử dụng nhiều nhất là sigmoid và tanh. Tuy nhiên, hiện tại hàm ReLU (Rectified Linear Unit) được sử dụng phổ biến hơn bởi những lý do sau:

- ReLU có định nghĩa đơn giản về cả hàm số lẫn đạo hàm.

$$ReLU(x) = \max(0, x)$$

$$\frac{d}{dx} ReLU(x) = \begin{cases} 1, & \text{nếu } x > 0 \\ 0, & \text{nếu } x \leq 0 \end{cases}$$

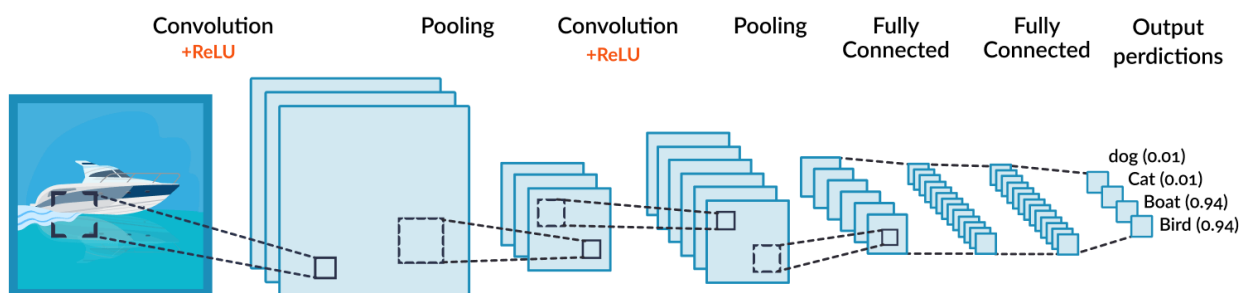
- Các hàm bảo hòa như sigmoid gây ra các vấn đề trong việc truyền ngược. Khi mạng thần kinh được thiết kế sâu hơn, các tín hiệu độ dốc sẽ bị "biến mất", không có ý nghĩa trong quá trình huấn luyện mạng thần kinh. Điều này xảy ra do giá trị độ dốc của các hàm đó rất gần bằng 0 hầu như ở khắp mọi nơi trừ vùng gần giá trị trung bình của hàm. Tuy nhiên, ReLU có một độ dốc không đổi cho đầu vào dương và bằng 1.
- ReLU tạo ra các thể diện thưa hơn, bởi vì các giá trị âm đều được chuyển hoàn toàn về 0. Ngược lại, các hàm sigmoid và tanh luôn có kết quả khác 0 về độ dốc, có thể gây bất lợi cho việc huấn luyện mạng thần kinh.

### 4 Lớp tổng hợp

Lớp tổng hợp thường được sử dụng giữa các lớp tích chập để đơn giản hóa thông tin đầu vào nhằm làm giảm số lượng neuron ở các lớp tiếp theo trong khi vẫn giữ lại được các đặc tính quan trọng của dữ liệu đầu vào. Điều này giúp làm giảm độ phức tạp và khối lượng tính toán của mạng thần kinh đồng thời vẫn bảo toàn được hiệu suất tính toán và độ chính xác của nó. Trong lĩnh vực xử lý ảnh, việc này tương tự với giảm độ phân giải của hình ảnh.

Các phương pháp tổng hợp được dùng phổ biến hiện nay là tổng hợp trung bình (average pooling) và tổng hợp cực đại (max pooling). Phương pháp tổng hợp cực đại chia hình ảnh ban đầu thành nhiều vùng nhỏ hơn có cùng kích thước và chọn giá trị lớn nhất trong vùng đó làm giá trị đầu ra của 1 neuron trong lớp tiếp theo. Các kích thước vùng thường được dùng để tổng hợp là 2x2 và 4x4. Ta có thể thấy trong Hình





Hình 9: Lớp kết nối đầy đủ

7, khi sử dụng lớp tổng hợp cực đại với kích thước vùng là  $2 \times 2$ , nó sẽ chọn giá trị cực đại trong mỗi vùng (các vùng xanh vàng, cam, xám, xanh dương) và tổng hợp thành ma trận kết quả, ma trận kết quả đã giảm kích thước đi so với ma trận đầu vào.

Việc sử dụng các lớp tổng hợp sẽ làm thay đổi vị trí tương đối của các điểm ảnh trong hình ảnh, do đó sẽ không bảo toàn được thông tin ban đầu. Vì vậy, các lớp tổng hợp chỉ nên được sử dụng khi cần trích xuất và bảo toàn các thông tin quan trọng của hình ảnh. Ngoài ra, ta có thể sử dụng các lớp tổng hợp với *stride* khác nhau để tăng độ hiệu quả cho việc trích xuất các đặc tính của dữ liệu đầu vào.

## 5 Lớp kết nối đầy đủ

Lớp kết nối đầy đủ được xây dựng tương tự như cách sắp xếp các neuron trên một lớp trong các mạng thần kinh nhân tạo cổ điển. Trong đó, mỗi neuron ở lớp sau được kết nối tới tất cả các neuron ở lớp trước như trong Hình 9. Đây là những lớp rất quan trọng đối với quá trình huấn luyện mạng thần kinh tích chập, giúp mạng thần kinh "học hỏi" được các đặc tính quan trọng của hình ảnh và đưa ra các kết quả dự đoán chính xác nhất.

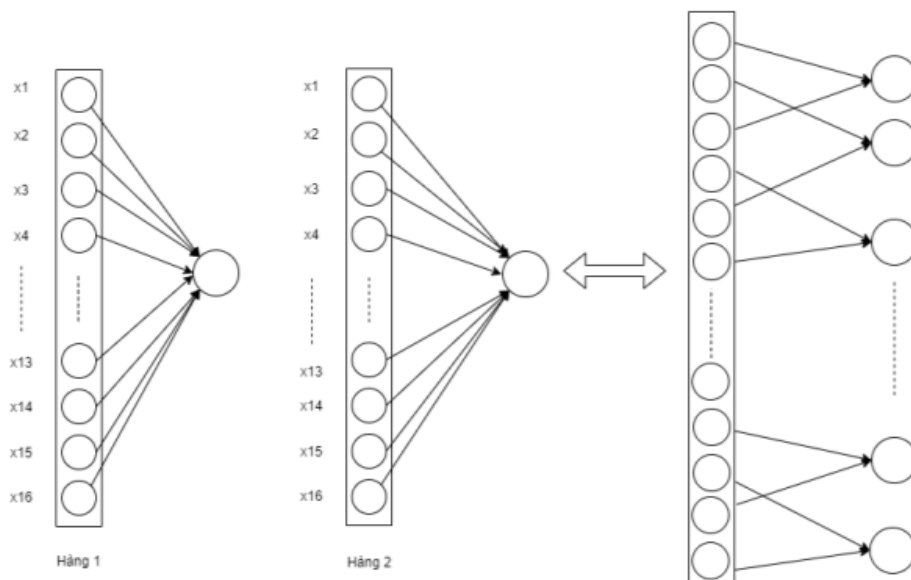
Hạn chế chính của lớp kết nối đầy đủ là nó chứa nhiều tham số và yêu cầu nhiều thao tác tính toán phức tạp trong quá trình huấn luyện mạng thần kinh. Chúng ta có thể loại bỏ một phần các kết nối nhằm tăng tốc độ huấn luyện mà vẫn bảo toàn được hiệu suất và độ chính xác của mạng thần kinh bằng kỹ thuật loại bỏ (dropout).

## 6 Mạng nơ-ron tích chập dưới góc nhìn của một mạng nơ-ron nhân tạo

Trong một mạng nơ-ron nhân tạo thông thường thì một mạng nơ-ron được cấu thành bởi các nút nơ-ron khác nhau nối tiếp nhau và qua quá trình xử lý thông tin sẽ tạo ra các nút nơ-ron ở tầng tiếp theo. Tương tự như vậy với mạng nơ-ron tích chập, ví dụ có một hình ảnh được biểu diễn dưới dạng ma trận đầu vào với kích thước  $10 \times 10$  và một cửa sổ tích chập là ma trận  $4 \times 4$  thì quá trình nhân tích chập tại vị trí đầu trên ma trận đầu vào  $10 \times 10$  sẽ tạo ra được một phần tử của ma trận mới và đó cũng tương đương với một nút nơ-ron được tạo ra sau khi thực hiện một phép biến đổi. Nút nơ-ron mới sẽ nối với 16 điểm trên ma trận đầu vào.

Hình 10 mô tả một ma trận đầu vào cùng với một cửa sổ và khi thực hiện phép tích chập trên một vị trí trên ma trận đầu vào sẽ tạo ra được một nút nơ-ron. Khi cho cửa sổ trượt qua hàng thứ hai sẽ thu được một nơ-ron khác và khi ta cho ma trận filter này trượt hết trên ma trận đầu vào thì sẽ tạo ra được 49 nút nơ-ron khác nhau và mỗi nút nối với một ma trận  $16$  đơn vị.

Một đặc điểm quan trọng của mạng nơ-ron tích chập là cơ chế chia sẻ trọng số (shared weights). Có nghĩa là các trọng số trên mỗi bộ lọc phải giống nhau và các nơ-ron trong lớp ẩn đầu sẽ phát hiện chính xác điểm tương tự chỉ ở các vị trí khác nhau trong dữ liệu đầu vào. Việc làm này sẽ làm giảm tối đa số lượng các tham số (parameters), mỗi bản đồ đặc trưng sẽ giúp phát hiện thêm một vài đặc trưng khác. Với một ma trận hình ảnh đầu vào kích thước  $10 \times 10$  như ở trên và 4 bộ lọc có ma trận kích thước  $4 \times 4$  thì mỗi bản đồ thuộc tính cần  $4 \times 4 = 16$  trọng số và số nơ-ron được tạo ra ở lớp thứ hai là 49. Như vậy nếu có 4 bản đồ thuộc tính thì có  $4 \times 16 = 64$  tham số. Với một mạng nơ-ron có kết nối đầy đủ thì chúng ta sẽ có  $10 \times 10 \times 49 = 4900$  trọng số. Từ kết quả cho thấy sử dụng lớp tích chập sẽ cần số lượng tham số ít hơn nhiều lần so với lớp kết nối đầy đủ nhưng vẫn có thể rút ra các đặc trưng một cách hiệu quả.



**Hình 10:** Hình ảnh mô tả phân tích chập một bộ lọc tạo ra các nơ-ron

Một khả năng khác của mạng nơ-ron tích chập là số tham số không phụ thuộc vào kích thước của đầu vào. Với những ma trận đầu vào có kích thước khác nhau và thông qua quá trình học theo phương pháp nơ-ron tích chập sẽ rút ra những thuộc tính ẩn mà ta có thể khó nhận thấy. Xét một ví dụ chúng ta có 10 bộ lọc và mỗi một bộ lọc sẽ là một ma trận kích thước  $3 \times 3 \times 3$  và có một giá trị sai lệch (bias) là 1, chúng ta cần tính xem có bao nhiêu parameter sẽ được tạo ra từ việc sử dụng mạng nơ-ron tích chập? Để giải được bài toán này thì cần tính số lượng parameters cần dùng mỗi bộ lọc rồi từ đó tính được kết quả.

- Số lượng tham số cho mỗi bộ lọc là  $3 \times 3 \times 3 = 27$
- Tổng số tham số cho mỗi bộ lọc là  $27 + 1 = 28$
- Tổng số tham số cho 10 bộ lọc là  $28 \times 10 = 280$

Như vậy từ ví dụ này thì cho dù dữ liệu đầu vào là bao nhiêu thì số lượng tham số được tạo ra cũng là 280, do đó số tham số có được từ mạng nơ-ron tích chập này không phụ thuộc vào kích thước đầu vào.

## 7 Triển khai một mạng CNN

Một mạng CNN cơ bản bao gồm 3 bộ phận chính: Tầng tích chập (Convolution), tầng tổng hợp (Pooling) và tầng cuối cùng là tầng kết nối đầy đủ (Fully Connected).

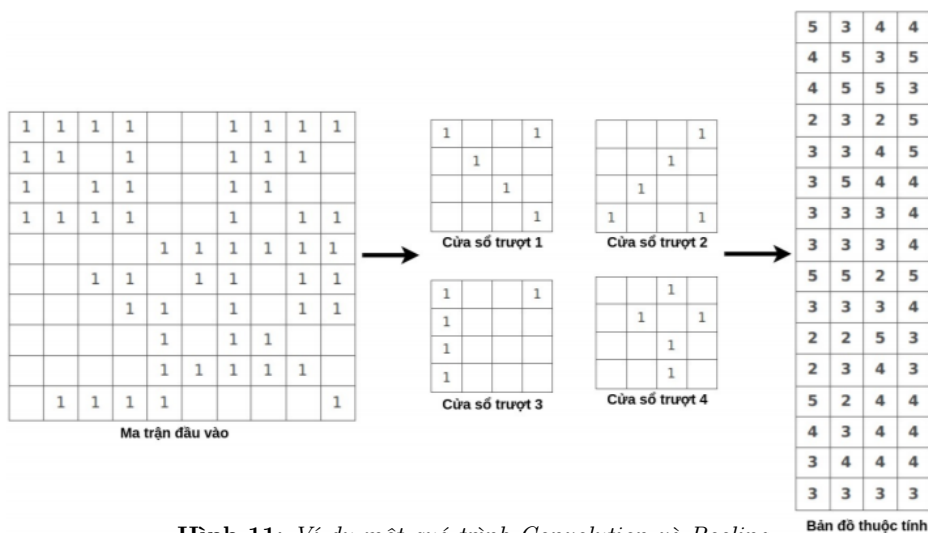
Tầng tích chập là tầng quan trọng nhất và cũng là tầng đầu tiên của mô hình CNN, có chức năng chính là phát hiện các đặc trưng cụ thể của input ban đầu (input ở đây có thể là một bức ảnh cây cối). Tầng này có các bộ phận chính là một ma trận đầu vào, bản đồ thuộc tính (Feature map) và các cửa sổ trượt (Convolution Filter). Tầng này bao gồm nhiều bản đồ thuộc tính đã được trích xuất ra các đặc tính cụ thể, mỗi bản đồ thuộc tính được tạo ra bằng cách quét input ban đầu qua cửa sổ trượt.

Tầng tổng hợp có mục đích làm giảm số lượng thông số mà ta phải tính toán, điều này giúp chúng ta giảm thời gian tính toán. Có 2 phương pháp tổng hợp thường gặp nhất là phép tổng hợp lớn nhất (Max pooling) và phép tổng hợp trung bình (Average Pooling).

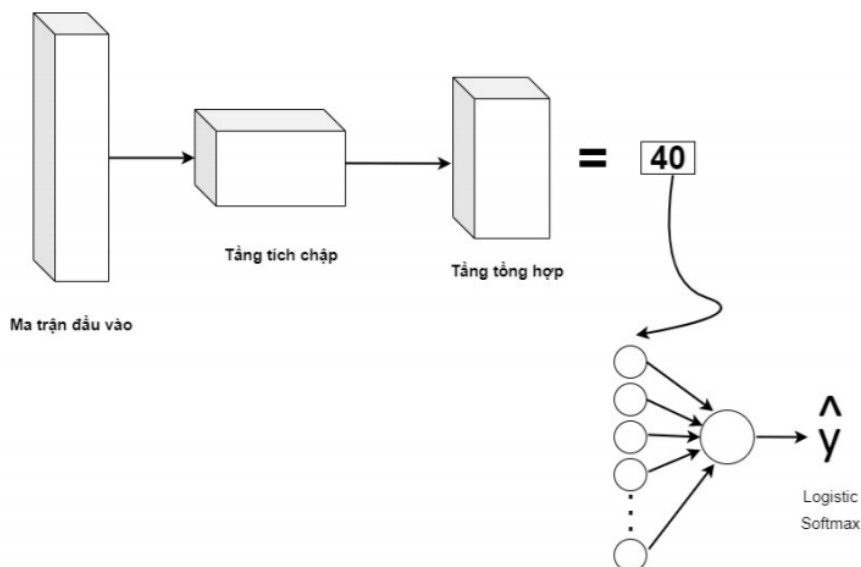
Cuối cùng tầng kết nối đầy đủ sẽ kết hợp các đặc điểm để ra được output của model. Để dễ hình dung, ta sẽ mô tả mạng quá trình Convolution và Pooling ở ví dụ trước đó qua hình 11 dưới đây.

Hình 11 ở trên biểu diễn ví dụ một ma trận đầu vào có kích thước  $10 \times 10$ , 4 cửa sổ trượt, mỗi cửa sổ trượt có kích thước  $4 \times 4$  và một bản đồ thuộc tính kết quả có kích thước  $4 \times 16$  và một bản đồ thuộc tính kết quả có kích thước  $4 \times 16$ . Cách tính toán bao gồm nhân ma trận đầu vào với các cửa sổ trượt, sau đó thực hiện Pooling để ra được kết quả bản đồ thuộc tính đã được trình bày ở phần trước.

Đây là ví dụ quá trình một ma trận ban đầu được thực hiện Convolution và Pooling để thu được một bản đồ thuộc tính, và quá trình này có thể lặp đi lặp lại nhiều lần nếu cần thiết. Có nghĩa là, bản đồ thuộc



Hình 11: Ví dụ một quá trình Convolution và Pooling



Hình 12: Ví dụ một mạng CNN

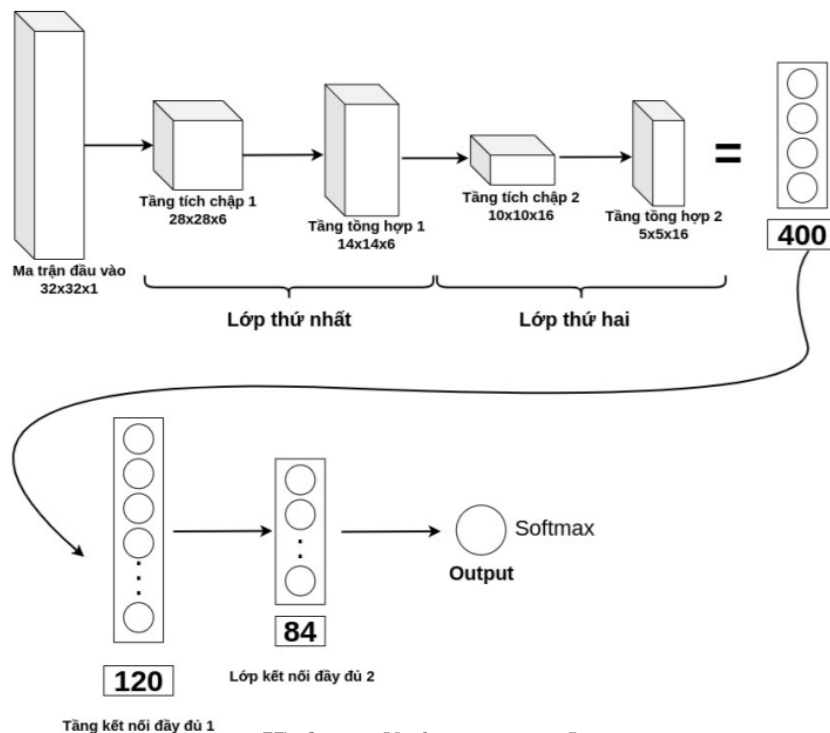
tính  $4 \times 16$  ta có được trong ví dụ trên, có thể tiếp tục thực hiện qua các cửa sổ trượt khác, và sẽ thu được một bản đồ thuộc tính khác với cách tính toán tương tự. Quá trình này có thể lặp lại rất nhiều lần, và đến tầng cuối cùng thì ta sẽ học qua tầng Kết nối đầy đủ. Chúng ta cần chú ý rằng quá trình Convolution và Pooling phải được thực hiện ít nhất một lần và 2 tầng này luôn đi kèm với nhau, nhưng luôn cần một tầng kết nối đầy đủ sau cùng, điều này tương đương với việc triển khai một mạng Nơ-ron Network đơn giản cho một bài toán cần học.

Nhưng tại sao tầng cuối phải là fully connected? Ta có thể thấy thực ra việc học CNN là nó học các trọng số của các cửa sổ trượt, nó cần học và cập nhật lại các trọng số này. Tương tự như những bài toán trước đó, cần phải có bài toán cho mạng CNN học. Việc học này sẽ được thực hiện ở tầng fully connected sau cùng, tương tự như một mạng nơ-ron cơ bản trình bày ở ví dụ trước.

Hình 12 thể hiện ví dụ một mạng CNN, ta có một ma trận đầu vào, một lớp tích chập và một lớp tổng hợp, ta ví dụ 40 là kết quả tính toán có được sau khi qua lớp tổng hợp (một vector 40 chiều), và vector 40 chiều này sẽ được đưa vào lớp kết nối đầy đủ, sử dụng hàm logistic hoặc Softmax tùy theo mục đích, để thu được output sau cùng.

Và với kiến trúc mạng này, ta có thể ráp nhiều tầng CNN lại với nhau. Ta cũng chú ý rằng tầng kết nối đầy đủ có thể nhiều hơn 1 tầng để tăng khả năng học của bài toán cần triển khai, nhưng không nên quá nhiều vì sẽ làm mất đặc trưng của bài toán cần học, vì nếu quá nhiều sẽ dẫn đến việc tăng số lượng thông số đầu vào. Nếu số tầng fully connected quá nhiều thì chỉ riêng thông số của các tầng này cũng đã hơn các tầng trước đó, và điều này làm mất đi một ưu điểm của CNN - giảm số tham số cần học của một mạng học sâu.

Mạng LeNet5 là một trong những mạng CNN lâu đời và nổi tiếng nhất, được Yann LeCun phát triển vào những năm 1998. Cấu trúc của mạng LeNet5 gồm 2 lớp Convolution và Maxpooling, 2 lớp Fully Connected và output là hàm Softmax. Hình 13 thể hiện ví dụ một mạng Lenet5.



Hình 13: Ví dụ một mạng Lenet-5

Một số thông tin về kiến trúc mạng LeNet5 như sau [18]:

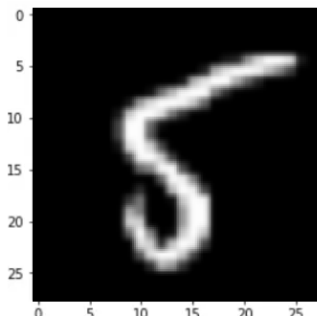
- Đầu vào là một ma trận có kích thước  $32 \times 32 \times 1$ , nghĩa là một tấm ảnh có chiều dài 32 pixel, chiều rộng ảnh là 32 pixel và số lượng kênh ảnh là 1 (ảnh đen trắng).
- Lớp thứ nhất bao gồm:
  - Tầng tích chập thứ 1: là một ma trận  $5 \times 5 \times 3$ , tốc độ stride là 1, đi qua 6 cửa sổ trượt có kích thước  $5 \times 5 \times 1$ , và output của nó là 1 ma trận có kích thước  $28 \times 28 \times 6$ .
  - Tầng tổng hợp thứ 1: Kích thước cửa sổ trượt là 1 ma trận  $2 \times 2$ , tốc độ stride là 2, số lượng cửa sổ trượt là 16, và output của nó là 1 ma trận có kích thước  $14 \times 14 \times 6$ .
- Lớp thứ hai bao gồm:
  - Tầng tích chập thứ 2: là một ma trận  $5 \times 5 \times 6$ , tốc độ stride là 1, đi qua 16 cửa sổ trượt và output của nó là 1 ma trận có kích thước  $10 \times 10 \times 16$ .
  - Tầng tổng hợp thứ 1: Kích thước cửa sổ trượt là 1 ma trận  $2 \times 2$ , tốc độ stride là 2 và output của nó là 1 ma trận có kích thước  $5 \times 5 \times 16$ .
- Số nút Output của lớp thứ 2 sẽ là  $5 \times 5 \times 16 = 400$ .
- Số nút Output của tầng kết nối đầy đủ thứ 1 sẽ là 120.
- Số nút Output của tầng kết nối đầy đủ thứ 2 sẽ là 84.

Với kiến trúc như thế này, thì việc áp dụng vào từng bài toán cụ thể sẽ được trình bày ở phần sau.

## 8 Case study: sử dụng CNN cho bài toán nhận diện chữ số viết tay (Handwriting Recognition)

Từ những lý do trên, nhóm chúng em quyết định sẽ làm việc với mạng tích chập để giải quyết bài toán phân loại chữ số viết tay từ tập dữ liệu MNIST. MNIST là bộ cơ sở dữ liệu phổ biến về ảnh các chữ số viết

tay từ 0 đến 9 bao gồm 2 tập con: tập huấn luyện gồm 60000 ảnh và tập kiểm tra gồm 10000 ảnh. Trong đó mỗi ảnh dữ liệu là một ảnh xám có kích thước 28x28 như Hình 14 và mục tiêu của bài toán là dự đoán con số viết tay trong bức ảnh đó.



**Hình 14:** Dữ liệu đầu tiên trong tập dữ liệu MNIST

Nhóm lựa chọn sử dụng thư viện keras hỗ trợ cho quá trình huấn luyện mô hình của mình. Giới thiệu một chút về keras, thì đây là một framework mã nguồn mở về Học sâu được viết bằng ngôn ngữ Python, với các đặc điểm sau:

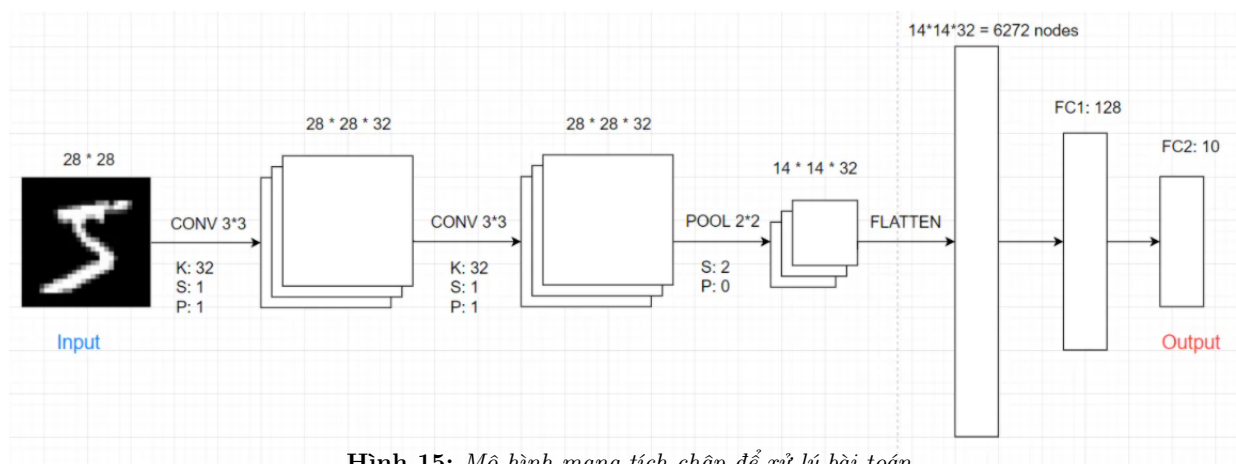
- Việc định nghĩa các tầng (layer), hàm kích hoạt (activation function), hàm mất mát (loss function) thực hiện rất đơn giản.
- Chỉ cần định nghĩa mô hình (model) và hàm mất mát, keras sẽ mặc định hỗ trợ người dùng phần Lan truyền ngược (backpropagation).
- Việc tính toán được thực hiện tối ưu trên cả CPU và GPU.

Vì dữ liệu đầu vào cho mô hình mạng tích chập là 1 tensor 4 chiều (N, W, H, D), cụ thể trong bài toán này là ảnh xám với  $W = H = 28$ ,  $D = 1$  và N là số lượng ảnh cho mỗi lần huấn luyện, nên trước tiên ta cần reshape lại dữ liệu ảnh có kích thước (N,28,28) thành kích thước (N,28,28,1) để giống kích thước mà keras yêu cầu.

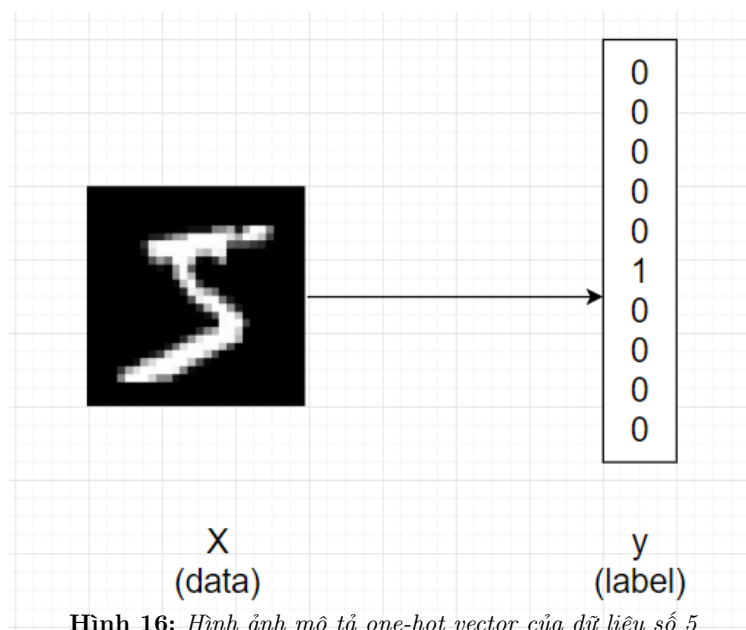
Sau đó, ta xây dựng mô hình cho bài toán theo mạng tích chập như Hình 15: từ ảnh đầu vào (input image) đi qua tầng tích chập (convolutional layer), tiếp đến là tầng tổng hợp (pooling layer), và cuối cùng là làm phẳng (flatten) trước khi đưa qua tầng kết nối đầy đủ (fully connected layer) để trả về kết quả đầu ra (output).

Chi tiết hơn, khi định nghĩa mô hình bằng cách sử dụng hàm Sequential() - một hàm được hỗ trợ bởi thư viện keras để ta có thể tự sắp xếp các tầng lên nhau trong mô hình mạng tích chập theo ý của mình. Dữ liệu đầu vào lần lượt đi qua các tầng sau

1. Tầng tích chập với 32 kernel kích thước 3\*3 dùng hàm kích hoạt là hàm ReLU
2. Tầng tổng hợp
3. Tầng làm phẳng để chuyển kiểu dữ liệu từ tensor sang vector. Từ tensor có kích thước (14, 14, 32), số nút (node) ta có sau khi làm phẳng là  $14 * 14 * 32 = 6272$  nút.



**Hình 15:** Mô hình mạng tích chập để xử lý bài toán



Hình 16: Hình ảnh mô tả one-hot vector của dữ liệu số 5

4. Tầng kết nối đầy đủ (FC) với 128 nút và dùng hàm kích hoạt là hàm sigmoid

Vì mỗi ảnh sẽ thuộc 1 lớp (class) từ 0 đến 9 nên ta sẽ có tổng cộng là 10 lớp. Chính vì vậy mà tầng đầu ra sẽ có 10 nút (node) chứa 10 giá trị thực tương ứng với ảnh là số từ 0 đến 9. Gọi  $z_k$  và  $a_k$  lần lượt là giá trị thực trong nút và xác suất một tấm ảnh có giá trị là  $k$ , ta có các công thức tính xác suất cơ bản như sau:

$$a_0 = \frac{e^{z_0}}{e^{z_0} + e^{z_1} + \dots + e^{z_9}}$$

$$a_1 = \frac{e^{z_1}}{e^{z_0} + e^{z_1} + \dots + e^{z_9}}$$

$$\dots$$

$$a_9 = \frac{e^{z_9}}{e^{z_0} + e^{z_1} + \dots + e^{z_9}}$$

Tổng quát hóa thành hàm kích hoạt, ta có công thức:  $a_k = \frac{e^{z_k}}{\sum_{i=1}^{10} e^{z_i}}$ . Đây cũng chính là công thức của hàm kích hoạt softmax.

Khi đã định nghĩa mô hình và hàm kích hoạt, việc cuối cùng ta cần làm đó là định nghĩa hàm mất mát (loss function). Nhưng trước hết, ta phải dùng phương pháp one-hot encoding - một phương pháp phổ biến để mã hóa các dữ liệu rời rạc (categorical data) - nhằm chuyển đổi nhãn của ảnh từ giá trị số nguyên sang vector cùng kích thước với đầu ra của mô hình. Khi đó nhãn của một dữ liệu số  $i$  sẽ được mã hóa thành vector  $v$  có kích thước  $10 \times 1$  với  $v_{i+1} = 1$  và các giá trị còn lại bằng 0. Hình 16 là ví dụ minh họa mô tả one-hot vector của dữ liệu số 5.

Giờ ta có cả giá trị thật (label) dạng one-hot encoding và giá trị dự đoán ở tầng đầu ra sau hàm softmax cùng kích thước  $10 \times 1$ . Ta cần định nghĩa hàm mất mát để đánh giá độ tốt của mô hình.

Thực tế, để đánh giá một dữ liệu là số  $i$  thì ta càng mong muốn rằng  $a_i$  càng gần 1 càng tốt còn các giá trị  $a$  khác thì ngược lại, càng gần 0 thì càng tốt. Vì như thế cũng đồng nghĩa là mô hình của ta dự đoán đúng được ảnh đầu vào là ảnh số  $i$ . Từ ý tưởng này, giả sử  $\hat{a}_i$  lần lượt là giá trị dự đoán cho giá trị thực  $a_i$  ta định nghĩa hàm mất mát như sau:

$$L = - \sum_{i=0}^9 (a_i * \log(\hat{a}_i))$$

Qua đó, ta thấy rằng:

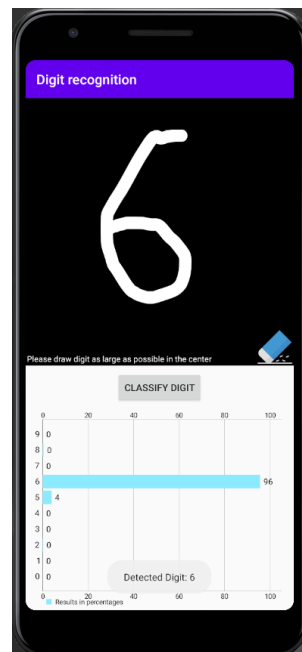
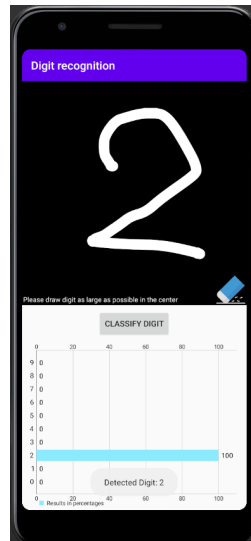
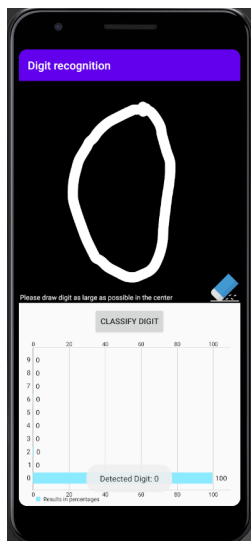
- Hàm  $L$  giảm dần từ 0 đến 1
- Khi model dự đoán gần 1, tức giá trị dự đoán gần với giá trị thực  $a_i$  thì  $L$  nhỏ, xấp xỉ 0
- Khi model dự đoán gần 0, tức giá trị dự đoán ngược lại giá trị thực  $a_i$  thì  $L$  rất lớn, tiến tới dương vô cùng.

Vậy nên có thể kết luận rằng hàm L nhỏ khi giá trị mô hình dự đoán gần với giá trị thực và rất lớn khi mô hình dự đoán sai, hay nói cách khác L càng nhỏ thì model dự đoán càng gần với giá trị chính xác. Khi đó bài toán tìm mô hình của chúng ta sẽ trở thành bài toán tìm giá trị nhỏ nhất của L.

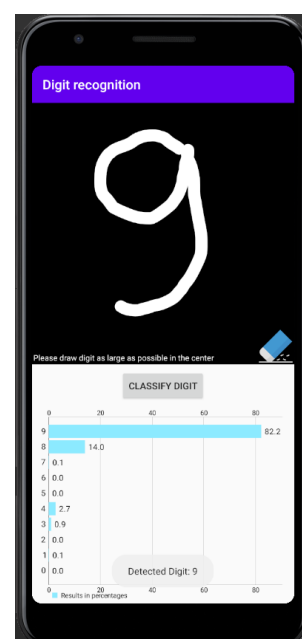
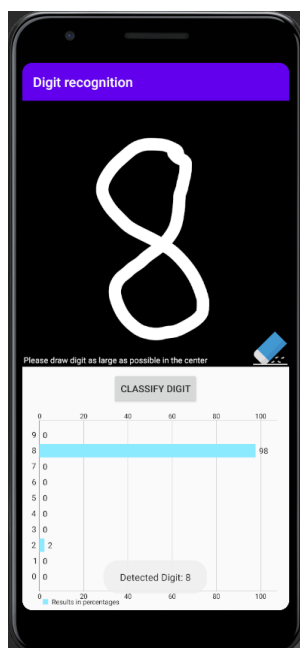
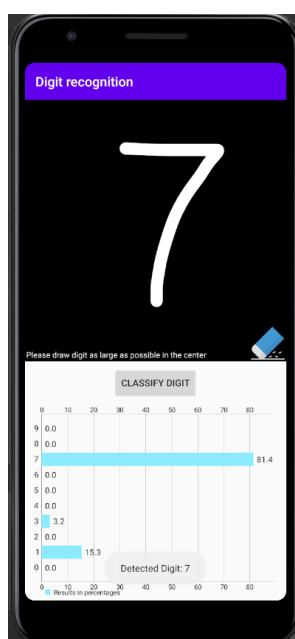
Hàm mất mát L được định nghĩa như trên trong keras gọi là “categorical\_crossentropy”.

## 9 Kết quả

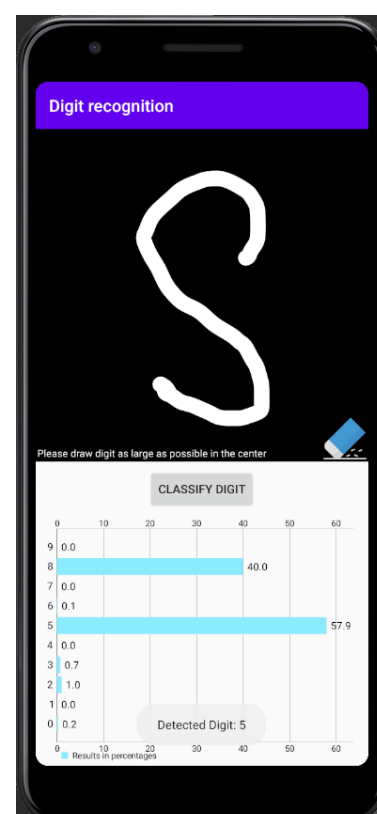
### 9.1 Nhận diện số từ 0 đến 9







## 9.2 Những trường hợp viết số không cụ thể



Ta thấy, với 2 trường hợp trên, kết quả dự đoán sẽ không chính xác, tỉ lệ dự đoán được phân bố ra các số khác nhau.

Tóm lại, độ chính xác của ứng dụng nhận dạng chữ số viết tay sẽ phụ thuộc rất nhiều vào cách người dùng vẽ. Nếu người dùng viết càng rõ ràng, độ chính xác sẽ càng cao. Nếu người dùng viết nguệch ngoạc, không thể hiện số cụ thể, kết quả dự đoán sẽ không chính xác.





## Tài liệu tham khảo

- [1] O. Abdel-Hamid, L. Deng, and D. Yu, “Exploring convolutional neural network structures and optimization techniques for speech recognition.,” in Interspeech, vol. 11, pp. 73–5, 2013.
- [2] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” arXivpreprint arXiv:1603.07285, 2016.