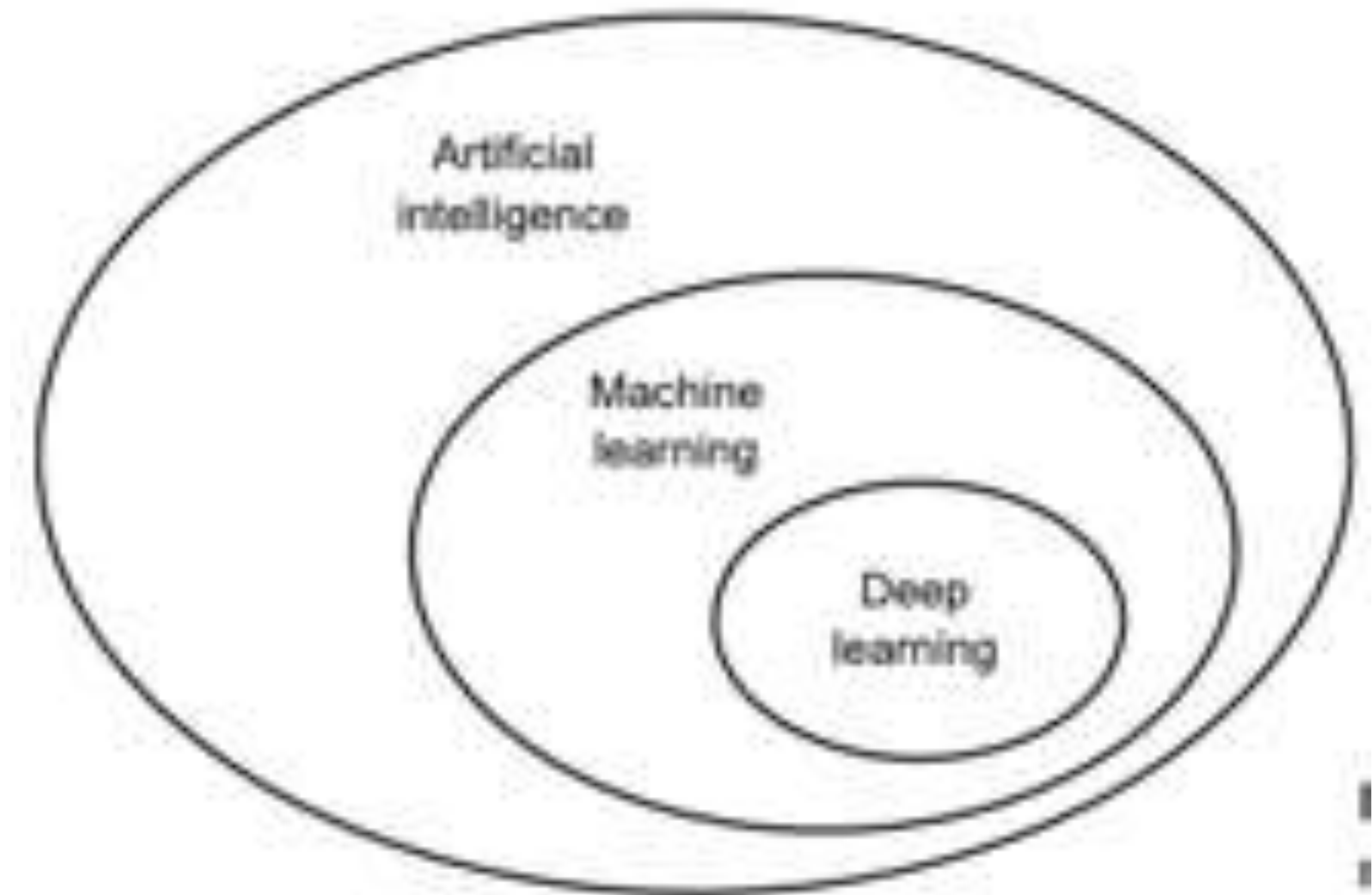
Trí tuệ tính toán và ứng dụng

PGS.TS. Trần Văn Lăng

Deep Learning

- Quan hệ giữa Artificial Intelligent - AI, Machine Learning – ML và Deep Learning – DL như hình bên dưới.
- Để hiểu được sự khác nhau giữa DL với các thuật toán ML, vấn đề sau đây cần phải xác định, trong ML có 3 yếu tố phải chỉ định rõ, đó là:



  - tập dữ liệu đầu vào (*data points*),
  - tập một số ví dụ về kết quả mong đợi và
  - cách thức đánh giá kết quả của thuật toán cũng như kết quả mong đợi.

- Như ta biết: AI cũng đã có cả một quá trình hình thành và phát triển. Từ ban đầu với việc xây dựng các cơ sở tri thức (*knowledge base*)
- Nhưng rồi vẫn cứ đi vào bế tắc vì cơ sở tri thức này cũng chỉ quẫn quanh trong việc lấy tri thức từ dữ liệu mà không có cách ***tự tiếp nhận thêm tri thức***.
- ML đã khắc phục khuyết điểm đó để phát triển, giúp cho AI phát triển vượt trội trong thời gian gần đây.
- Trong hệ thống ML, việc thu thập dữ liệu, kiểm tra dữ liệu; ngay cả những vấn đề về hệ thống (*cyber structure*) cũng góp phần quan trọng.

- Đặc biệt trong đó là làm sao để có thể trích xuất được các đặc trưng của dữ liệu (***feature extraction***).
- Nên trích xuất dữ liệu có ý nghĩa quyết định về sự thành công của một hệ thống AI.
- Khó khăn khi dataset không có label, thì việc trích xuất càng khó cho ra kết quả gọi là chính xác (*hoặc nhẹ nhàng hơn - đó là kết quả không sai*), do việc huấn luyện lúc này theo xu hướng huấn luyện không giám sát (*unsupervised learning*).
- Deep Learning nhằm tạo ra ứng dụng mà máy tính tự đưa ra những kết quả phức tạp theo nhu cầu của thế giới thực từ các kết quả đơn giản hơn.

- Công việc này được lặp lui lặp tới nhiều lần như kiểu con người đang tự học.
- Cụ thể, kết quả Z được hình thành từ các kết quả $A_1, A_2, \ldots, A_n$ đơn giản hơn; rồi những kết quả $A_i, \forall i = \overline{1,n}$ được hình thành từ các kết quả đơn giản hơn là $B_{i1}, B_{i2}, \ldots, B_{im}$. Và rồi cứ thế mà tiếp tục để hình thành nên kết quả Z.
- Như đã trình bày trong những phần trước, để giải quyết vấn đề huấn luyện (*hay là học*) trong ML thì mạng lưới thần kinh nhân tạo (ANN) có vai trò rất lớn.

- Cũng cả một thời gian dài, việc dự báo chỉ đi theo một chiều đó là từ các đặc trưng đầu vào ở tầng input, qua một hàm kích hoạt cho ra những giá trị ở tầng hidden, và cứ tiếp tục một số tầng hidden với đầu vào là kết quả đầu ra qua hàm kích hoạt của tầng hidden trước đó. Đến một lúc nào đó thì có kết quả ở tầng output.
- Qua giữa những năm 70 của thiên niên kỷ 19, hình thành nên khái niệm lan truyền ngược (*back propagation*), từ đó các trọng số trong quá trình tính toán đi theo chiều ngược từ tầng output đến tầng input được cập nhật. Và rồi cứ thể lặp lui lặp tới để tạo ra các trọng số (*gọi là huấn luyện*) làm sao cho kết quả càng thực tế hơn.

- Thuật toán Gradient Descent ra đời cũng lâu lắm rồi nhằm để tìm giá trị để hàm số đạt cực tiểu (*nói chung là đạt cực trị*) đã được áp dụng khá thành công trong việc tìm các trọng số này vì đặc điểm đơn giản của nó.
- Như vậy sự lan truyền ngược đã giải quyết thành công về sự huấn luyện.
- Tuy nhiên, khi số đặc trưng nhiều (*số neuron của tầng input lớn*), và số data point cũng nhiều, số tầng hidden cũng như số neuron của những tầng này lớn thì ANN trở nên gần như không khả thi.
- Kể từ khi những kỹ sư của Hãng NVIDIA làm cho card đồ hoạ có khả năng xử lý các phép tính số học, GPU ra đời. ML thông qua ANN để phát triển DL một cách nhanh chóng. Thêm vào đó Big Data là cơ hội để DL thoả sức khai thác.

CNN và RNN

- Như vậy, cốt lõi của DL đó là ANN. Tuy nhiên, với sự tiến hoá từ ANN đã hình thành nên các khái niệm như CNN và RNN.
- CNN (*Convolutional Neural Network*) là mạng lưới thần kinh tích chập. CNN còn được viết là ConvNet.
  - Ban đầu CNN là nền tảng để tạo ra các ứng dụng AI trong lĩnh vực thị giác máy tính (Computer Vision). Dần dần sau đó có thể dùng cho các lĩnh vực khác khi dữ liệu được số hoá thành các vector, hay matrix hay tensor.
- Còn RNN (*Recurrent Neural Network*), là mạng lưới thần kinh tái diễn
  - RNN dùng cho bài toán liên quan đến dữ liệu mà có sự liên kết với nhau như những chuỗi (*sequence*) sự kiện.

Khái niệm tích chập

- Tích chập (*convolution*) là khái niệm quan trọng và cơ bản trong xử lý và phân tích tín hiệu số
- Nếu biết được đáp ứng xung (*impulse response*) của hệ thống, khi đó với một convolution có thể xây dựng đầu ra của hệ thống đối với bất kỳ tín hiệu đầu vào nào.
- Vấn đề đặt ra là bằng cách nào mà chỉ với một đáp ứng xung của hệ thống thì có thể xác định được đầu ra với tín hiệu đầu vào cho trước.

- Về mặt toán học, với $x(n)$ là tín hiệu đầu vào, $k(n)$ là đáp ứng xung và $y(n)$ là đầu ra, convolution (ký hiệu phép toán $\otimes$) trong miền rời rạc được định nghĩa $y(n) = x(n) \otimes k(n) = \sum_{m=-\infty}^{+\infty} x(m)k(n-m)$
- Ý nghĩa được thể hiện qua ví dụ, chẳng hạn với hàm Dirac Delta

$$\delta(x) = \begin{cases} 1 \text{ nếu } x = 0 \\ 0 \text{ nếu } x \neq 0 \end{cases}$$

- Ta có thể viết

$$x(n) = (x \otimes \delta)(n) = \sum_{m=-\infty}^{+\infty} x(m)\delta(n-m)$$

- Ở đây giá trị đầu ra $x(n)$ (*cũng chính là tín hiệu đầu vào* $x(n)$) được trượt qua các xung $\delta(n)$ tương ứng.
- Nếu $\delta(n)$ là một hàm tổng quát hơn, chẳng hạn $k(n)$ thì đầu ra $y(n)$ như là sự chập lại của các tín hiệu vào xung quanh nó.

- Một ví dụ khác, cho các tín hiệu đầu vào

$$x(n) = \begin{cases} n+3, \text{ nếu } n = 0,1,2 \\ 0, \text{ với các } n \text{ còn lại} \end{cases}$$

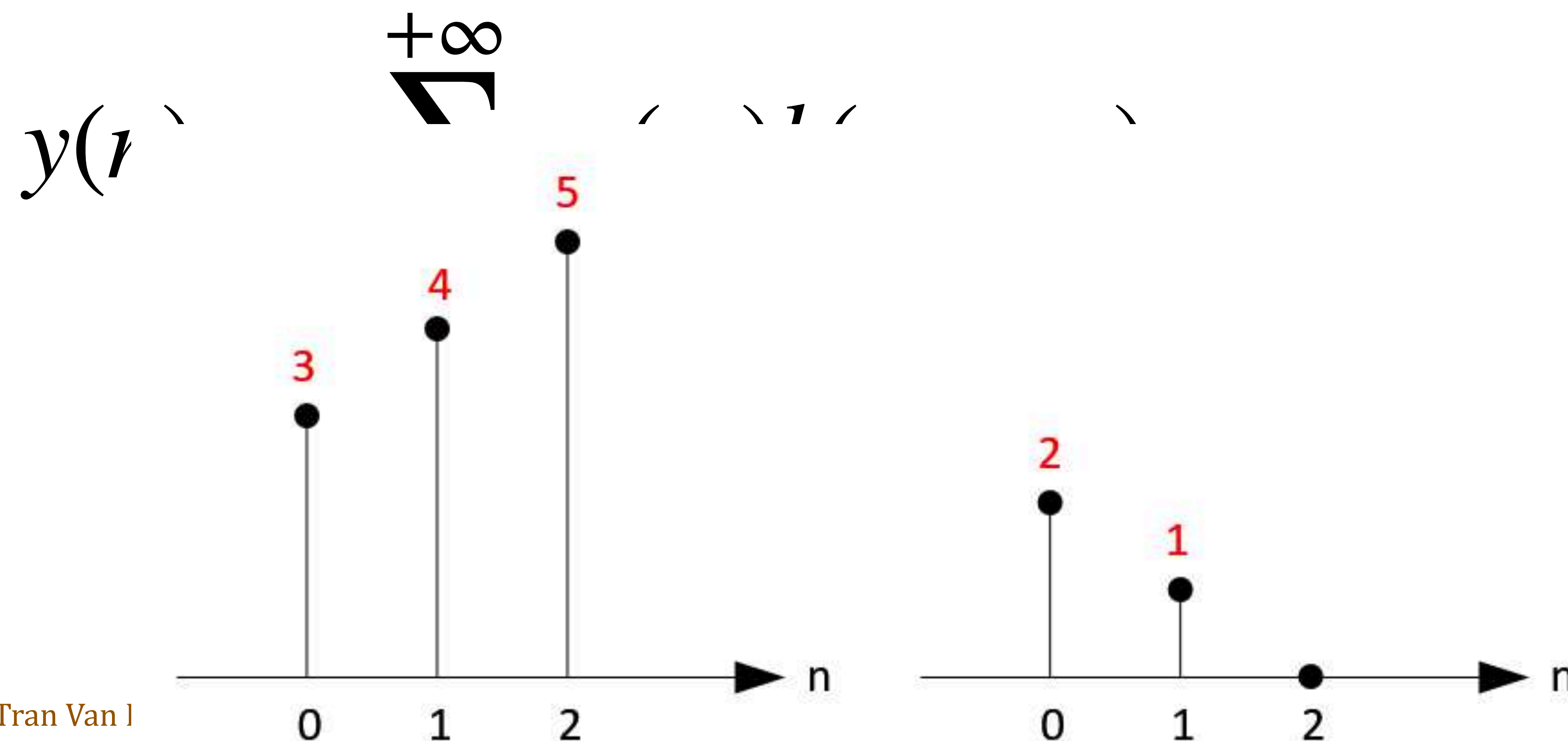
- Và đáp ứng xung là hàm

$$k(n) = \begin{cases} 2-n, \ \text{ nếu } n = 0,1 \\ 0, \text{ ngược lại} \end{cases}$$

- Khi đó đầu ra qua phép tích chập là

$+\infty$

$y($




```python
def x(n):
    if n==0 or n==1 or n== 2:
        return n+3
    else:
        return 0
```

```python
def k(n):
    if n==0 or n==1:
        return 2-n
    else:
        return 0
```

```python
INFTY = 100000 # Giả sử 100000 là số vô cùng
def y(n):
    s = 0
    for m in range(-INFTY,INFTY):
        s += x(m)*k(n-m)
    return s
```

```python
for n in range(10):
    print( "y[%2d] = %d" % (n,y(n)) )
```

```
y[ 0] = 6
y[ 1] = 11
y[ 2] = 14
y[ 3] = 5
y[ 4] = 0
y[ 5] = 0
y[ 6] = 0
y[ 7] = 0
y[ 8] = 0
y[ 9] = 0
```

- Trong miền liên tục, hai hàm f và g tích chập với nhau cho ra một hàm:

$$(f \otimes g)(x) = f(x) \otimes g(x) = \int_{-\infty}^{+\infty} f(t)g(x-t)dt = \int_{-\infty}^{+\infty} f(x-t)g(t)dt$$

- Tích chập này có tính giao hoán, nên trong miền rời rạc có thể viết

$$(f \otimes g)(n) = \sum_{m=-\infty}^{+\infty} f(m)g(n-m) = \sum_{m=-\infty}^{+\infty} f(n-m)g(m)$$

- Tích chập 2 chiều là sự mở rộng của tích chập 1 chiều bằng cách tích chập cả chiều ngang và chiều dọc trong không gian 2 chiều.

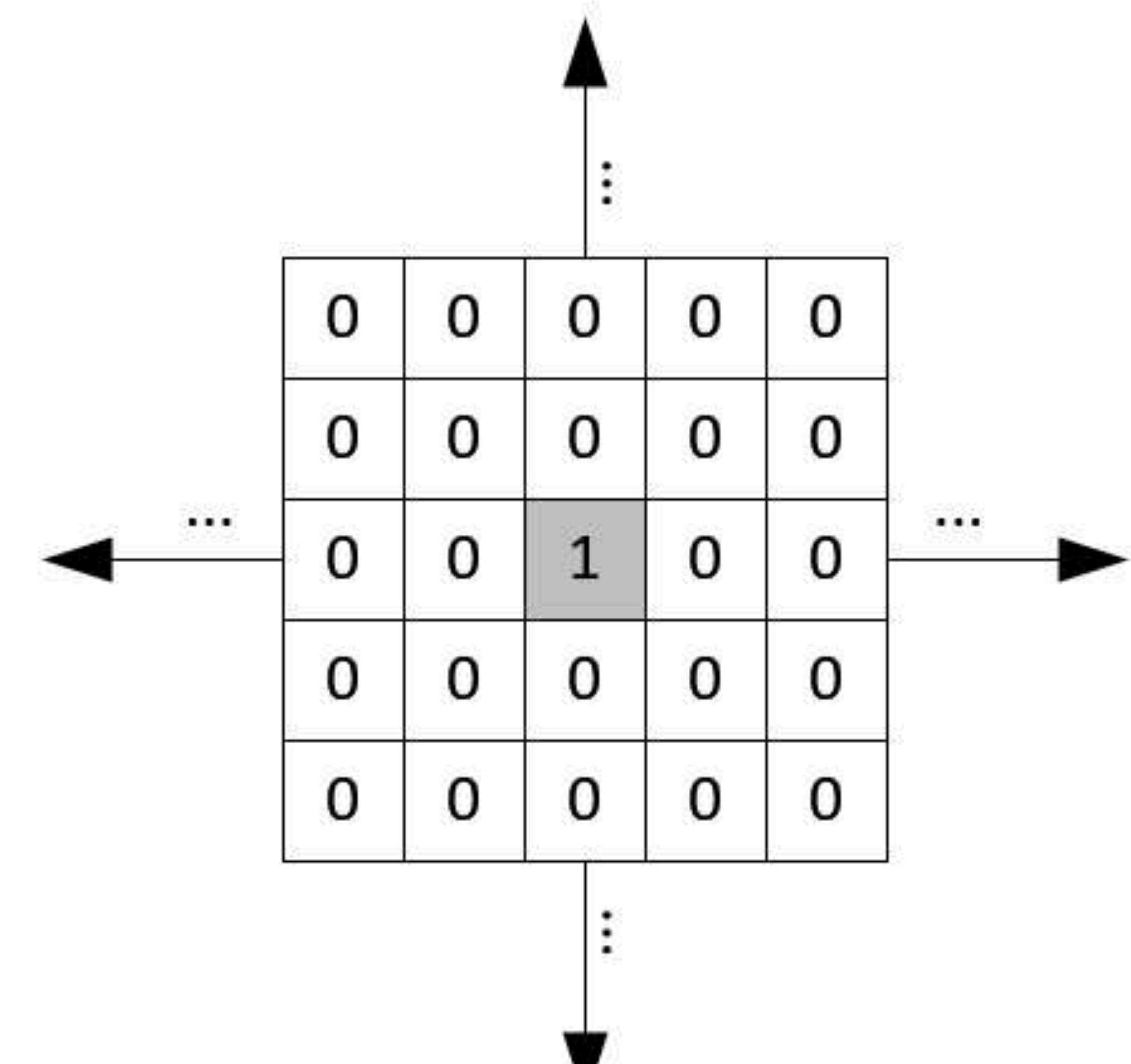
- Tích chập này thường được dùng trong xử lý ảnh chẳng hạn làm mịn ảnh, làm sắc nét (*sharpening*) ảnh, phát hiện cạnh của ảnh.
- Khi đó tích chập của 2 hàm f, g trong không gian rời rạc được viết

$$(f \otimes g)(x,y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u,v)g(x-u,y-v) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(x-u,y-v)g(u,v)$$

- Ý nghĩa của phép toán tích chập này để biết tác động của hàm ảnh $f(x,y)$ bởi bộ lọc (kernel) $g(x,y)$ hay ngược lại.

- Ví dụ với hàm Dirac Delta trong mặt phẳng (như hình) dùng làm hàm xung (impulse function)

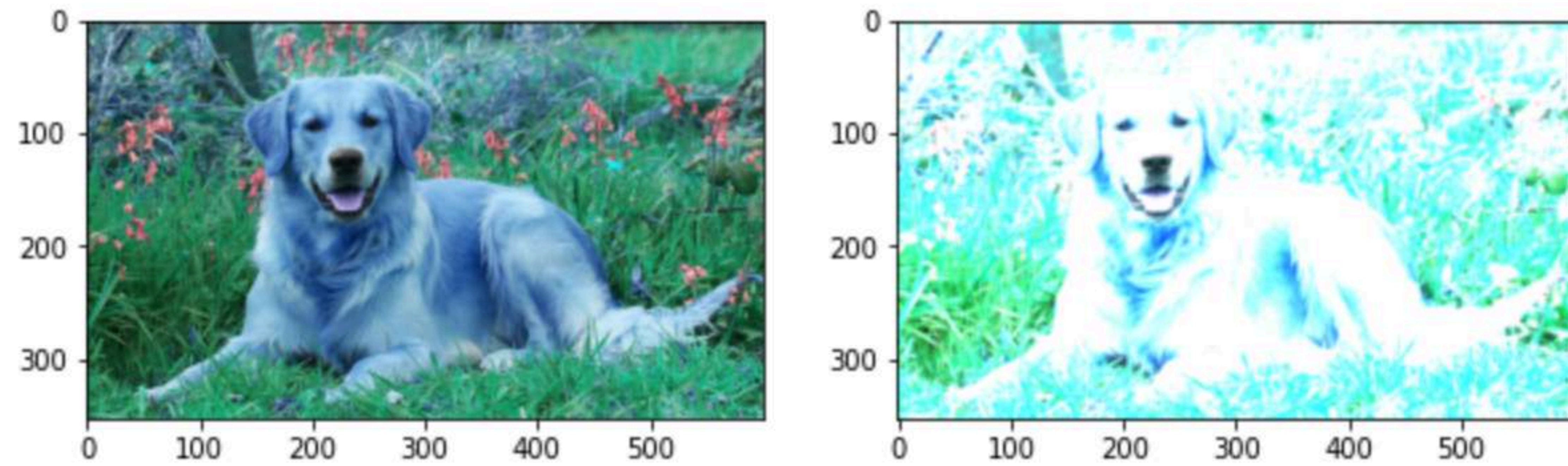
$$\delta(n,m) = \begin{cases} 1, \text{ nếu } n,m = 0 \\ 0, \text{ ngược lại} \end{cases}$$



- Thì

$$x(n,m) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} x(i,j)\delta(n-i,m-j)$$

- Lưu ý kernel (đáp ứng xung) trong trường hợp 2 chiều thì điểm trung tâm phải là điểm gốc. Chẳng hạn, với kích thước kernel là 3, thì điểm trung tâm là $g(0{,}0)$, các chỉ số 2 chiều còn lại là -1, 0, 1.

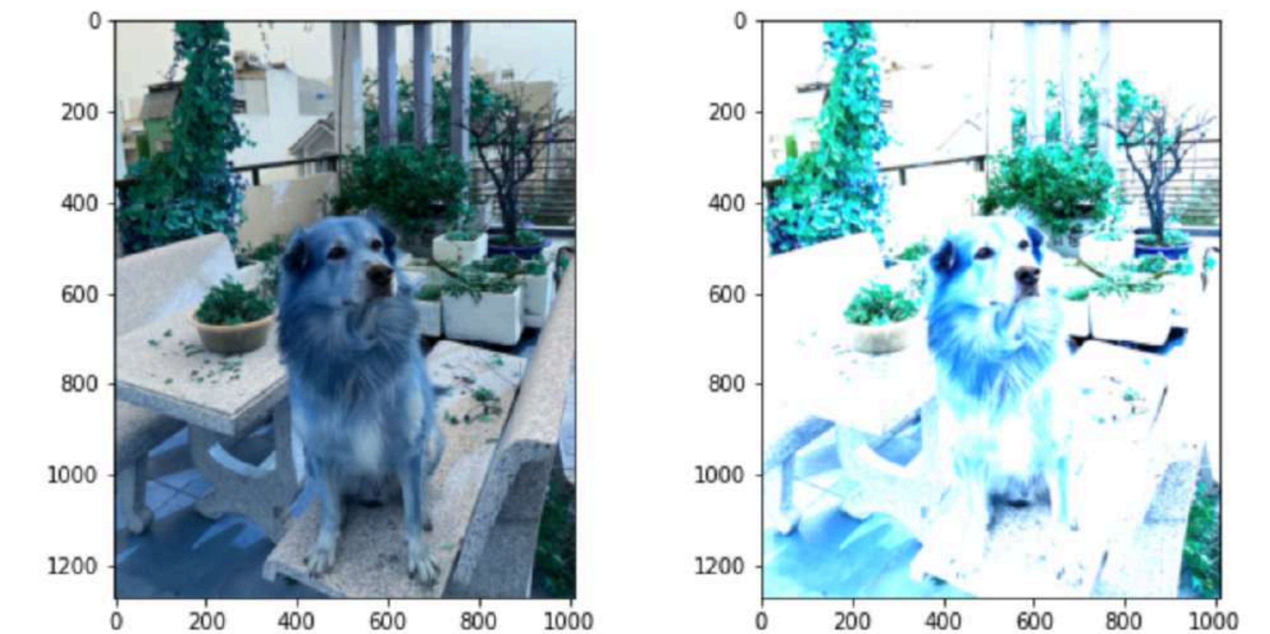


- Ví dụ, để làm mờ ảnh (*blurring*) trong xử lý ảnh người ta thay thế những điểm có độ sáng bất kỳ bằng trung bình độ sáng của những điểm lân cận (*kích thước ma trận liên quan đến độ mờ của ảnh*)
- Khi đó cần phải tính tích chập của ảnh với ma trận tích chập *Kernel* có giá trị

$$Kernel = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- Chương trình viết như sau

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("./goldenretriever.jpg")
ker = np.array(
    [
        [1, 1, 1],
        [1, 1, 1],
        [1, 1, 1]
    ])
ker = ker/sum(ker)
out = cv2.filter2D(img,-1,kernel=ker)
_,axs = plt.subplots(1,2,figsize=(10,5))
axs[0].imshow(img)
axs[1].imshow(out)
plt.show()
```



Xử lý ảnh

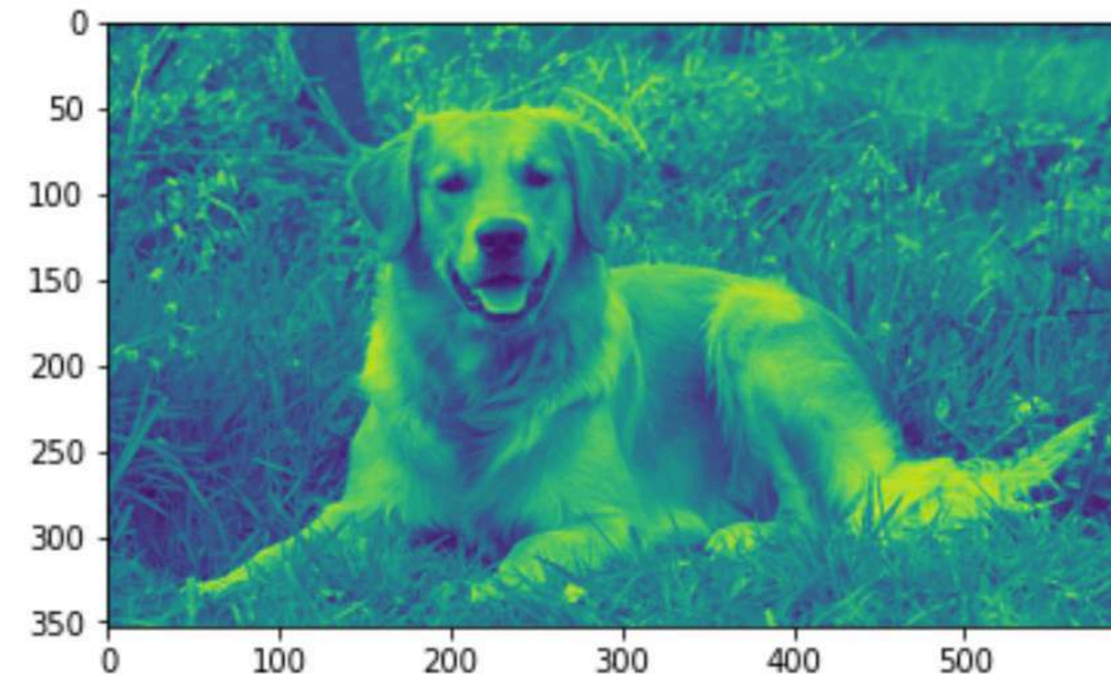
- Một ảnh có kích thước $w \times h$ pixel, mà mỗi pixel là một màu được tạo ra từ 3 giá trị Red, Green, Blue.
- Nên một ảnh được ký hiệu dưới dạng 1 ma trận

$$P = (p_{ij}) = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1h} \\ p_{21} & p_{22} & \cdots & p_{2h} \\ \vdots & \vdots & & \vdots \\ p_{w1} & p_{w2} & \cdots & p_{wh} \end{bmatrix}$$

- Trong đó $p_{ij} = (r_{ij}, g_{ij}, b_{ij})$ được tách thành 3 ma trận có cùng kích thước tương ứng là Red, Green, Blue.

- Khi đó một ảnh màu $w \times h$ thay vì ở dạng một ma trận như trên sẽ được đưa về một tensor $w \times h \times 3$ ứng với 3 màu Red, Green, Blue.
- Với ảnh xám, do mỗi pixel chỉ có 1 giá trị thuộc $[0,255]$, trong đó giá trị tiến về 0 thì càng tối và tiến về 255 thì càng sáng; nên chỉ cần một ma trận $w \times h$ có thể lưu trữ một ảnh xám.
- Trong xử lý ảnh, để chuyển một ảnh màu sang ảnh xám, người ta dùng công thức

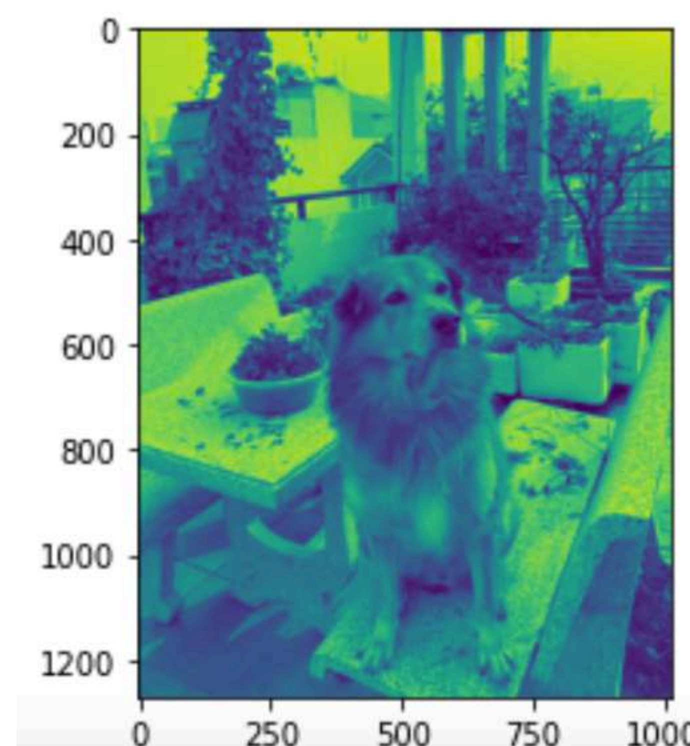
$$p_{ij} = 0.299 r_{ij} + 0.587 g_{ij} + 0.114 b_{ij}$$



- Vì sao như vậy.
  - Hỏi chuyên gia về xử lý ảnh hoặc hỏi Google, hoặc xây dựng một ANN để tìm 3 trọng số này
- Để viết chương trình với

```
img = cv2.imread("../dataset/Mickey.jpg")
imgray = np.dot( img[...,:3],[0.299,0.587,0.144] )
plt.imshow(imgray)
plt.show()
```


```
1 img = cv2.imread("../dataset/Mickey.jpg")
2 imgray = np.dot( img[...,:3],[0.299,0.587,0.144] )
3 plt.imshow(imgray)
4 plt.show()
```

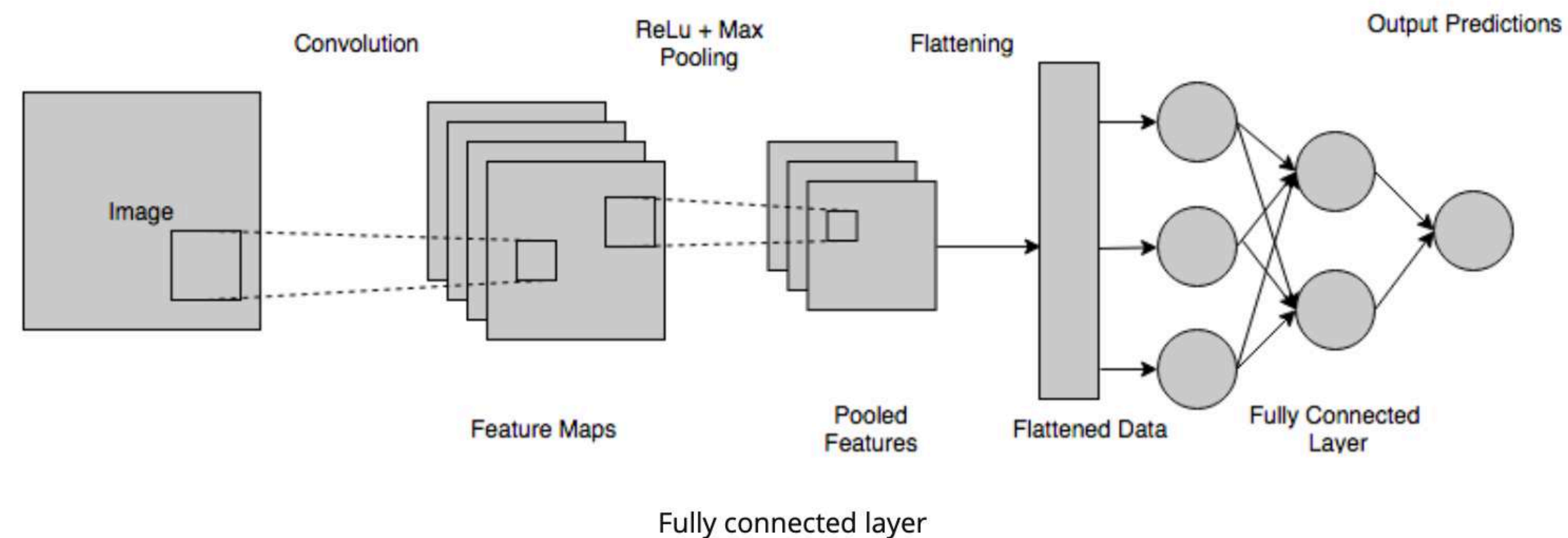



Mạng lưới thần kinh tích chập

- Mạng lưới thần kinh tích chập (CNN) trước hết nó là một ANN mà trong đó mỗi tầng hidden được thay bằng tầng **convolution** (*convolutional layer*) và tầng **pooling** một cách lần lượt để rồi chuyển đến tầng **fully connected** trước khi đưa ra tầng Output.
- Nên có thể tóm gọn: **Input Layer → Convolutional Layer →Pooling Layer → . . . →Convolutional Layer→Pooling Layer→Fully Connected Layer →Output Layer**
- Lưu ý, sau mỗi tầng Hidden (*bao gồm convolution và pooling*), hàm kích hoạt được sử dụng để tạo dữ liệu đầu ra cho tầng tiếp theo.

Mạng lưới thần kinh tích chập

- Tầng pooling thường là tầng để tổng hợp lấy ra đặc trưng tốt nhất (*max pooling*)
- Do kết quả của tầng pooling có thể là một tensor, nên cần phải dát phẳng ra (*flattening*) để trở thành một vector trước khi đưa ra tầng output, qua đó bảo đảm các neuron được kết nối theo kiểu của ANN,
- Chính vì vậy có tầng mang tên fully Connection để làm công việc flattening này nhằm biến một tensor trở thành một vector.



Fully connected layer

- **Quay trở lại** với tầng convolution, vì sao phép toán tích chập lại đóng vai trò quan trọng trong việc xác định các đặc trưng.
- Trước hết, do CNN xuất phát ban đầu từ lĩnh vực thị giác máy tính (*computer vision*), nên việc nhận dạng hình ảnh đưa vào là hình gì, hoặc những gì có trong hình đó là một nhu cầu.

- Để làm được điều này, phải coi tất cả các pixel tạo nên ảnh như là những đặc trưng thô ban đầu (*raw feature*), hay là thuộc tính của dataset
- Nên với ảnh màu, thì số lượng pixel rất lớn, dẫn đến số đặc trưng này rất lớn; từ đó số neuron của tầng input vô cùng lớn.
- Ngay cả với một ảnh đen trắng kích thước $w \times h$ số lượng pixel cũng đã lớn, nên có số lượng neuron của tầng input cũng đã quá lớn so với một tầng input thông thường.
- Với ảnh màu có kích thước w pixel chiều rộng và h pixel chiều cao, thì kích thước của mỗi data point là $w \times h \times 3$

- Khi đó số neuron ở tầng input là $w \times h \times 3$
- Do cần huấn luyện để có được nhiều đặc trưng, vì vậy cần phải có những tầng tích chập.
- Với mỗi kernel khác nhau ta sẽ huấn luyện để được những đặc trưng khác nhau của ảnh, nên trong mỗi tầng convolution cần phải dùng nhiều kernel để học được nhiều thuộc tính của ảnh.
- Cũng với dataset là ảnh màu, nên do đầu vào của tầng input là một tensor $w \times h \times 3$, từ đó một kernel cũng phải là tensor dạng $k \times k \times 3$, trong đó $k = 3,5,7,9,\ldots$

Ví dụ minh hoạ

- Giả sử có 2 ảnh xám là hình của chữ L và chữ T được lưu trữ trong vùng 9×9 pixel; vấn đề đặt ra là cho chương trình máy tính học hình dạng của 2 chữ này.
- Ở đây, do ảnh xám nên mỗi data point có kích thước $9 \times 9 \times 1$
- Cũng giả sử thêm, tại mỗi pixel có giá trị là 1 nếu đó là nét chữ, còn -1 không phải là net chữ. Nên các feature của mỗi chữ là một ảnh nhỏ trong đó có nét đặc trưng tạo nên chữ đó. Chẳng hạn chữ L có một nét gồm 2 cạnh vuông góc với nhau tạo nên 1 góc gần 90^0; còn chữ T có nét gồm 2 cạnh vuông góc với nhau nhưng có 2 góc gần 90^0.

- Từ đây sử dụng ma trận kernel để lọc trong phép tính tích chập.
- Qua phép toán tích chập (3×3), kích thước ma trận còn là 7×7
- Điều này xãy ra là do thực hiện các phép tính tích chập trên ma trận, nên với ma trận ảnh đầu vào $w \times h$ phần tử, sau khi thực hiện qua hàm xung (*ma trận kernel* 3×3) thì kích thước của ma trận ảnh đầu ra chỉ còn $(w-2) \times (h-2)$ phần tử
- Tương tự như vậy, với ma trận kernel K là ma trận 5×5 thì ma trận đầu ra chỉ còn $(w-4) \times (h-4)$.

Padding, Stride

Lưu ý

- Tổng quát, nếu ma trận K là ma trận kích thước $k \times k$, thì ma trận ảnh đầu ra qua phép tích chập chỉ còn $(w - k + 1) \times (h - k + 1)$ phần tử.
- Để bảo toàn kích thước ma trận ảnh qua phép tích chập, cần phải đệm thêm (***padding***) một số phần tử có giá trị bằng 0 ở viền ngoài của ma trận ảnh.
- Ngoài ra, nếu cần ma trận ảnh đầu ra nhỏ hơn, thay vì mỗi lần tính tích chập cho một phần tử của ma trận đầu vào tiếp tục nhảy qua (***stride***) phần tử kế tiếp với bước nhảy là 1; ta dùng bước nhảy là 2 hoặc 3, ... để bỏ qua những phần từ kề nhau.

- Tổng quát, với kernel $k \times k$, padding là p, stride là s. Thì ma trận ảnh ban đầu có kích thước $w \times h$, ma trận ảnh qua phép tính tích chập kích thước là $\lfloor \frac{w-k+2p}{s} + 1 \rfloor \times \lfloor \frac{h-k+2p}{s} + 1 \rfloor$ phần tử
- Khi đó với K kernel (K bộ lọc), sau khi qua tầng convolution thì đầu vào của tầng pooling là một tensor có kích thước $\lfloor \frac{w-k+2p}{s} + 1 \rfloor \times \lfloor \frac{h-k+2p}{s} + 1 \rfloor \times K$ sau khi qua hàm kích hoạt.

- Giả sử một kernel có kích thước $k \times k \times d$ và một độ lệch (bias) số tham số của một kernel là $k \times k \times d + 1$. Nên với K kernel có số tham số là $K(k \times k \times d + 1)$
- Chẳng hạn, với ảnh màu (red, green, blue) và kernel có kích thước 5×5, thì
    - một kernel là một khối kích thước $5 \times 5 \times 3 = 75$.
    - thêm độ lệch nên kích thước của một bộ lọc là $75 + 1$
- Nên với 10 bộ lọc ($K = 10$) thì số tham số của bộ lọc là $76 \times 10 = 760$

Ví dụ

- Giả sử một ảnh màu có 64×64 pixel. Vấn đề đặt ra là xây dựng một hệ thống DL để nhận ra ảnh đưa vào là gì. Như vậy
  - Ảnh có kích thước $64 \times 64 \times 3$
- Giả sử kiến trúc của CNN này như sau:
  - Tầng tích chập đầu tiên có 10 bộ lọc kích thước $k = 3$ cho mỗi channel, giả sử độ trượt (stride) là $s = 1$, và không bổ sung thêm viền ($p = 0$).
    - Kết quả cho ra $\left\lfloor \frac{64-3+0}{1} + 1 \right\rfloor \times \left\lfloor \frac{64-3+0}{1} + 1 \right\rfloor = 62 \times 62$
    - Số tham số ứng với 3 channel là $62 \times 62 \times 3$

- Tầng max pooling kích thước 2×2, $s = 2$ và $p = 0$ nên kích thước giảm đi một nữa, còn số channel giữ nguyên là 3. Ta có kích thước là $31 \times 31 \times 3 \times 10$

- Tầng tích chập thứ hai có 20 bộ lọc, $k = 5, s = 2, p = 0$
    - Tương tự kích thước đầu ra
      $$\left\lfloor \frac{31 - 5 + 0}{2} + 1 \right\rfloor \times \left\lfloor \frac{31 - 5 + 0}{2} + 1 \right\rfloor = 14 \times 14$$
    - Số tham số $14 \times 14 \times 3$

- Tầng max pooling (2,2) $s = 2$ và $p = 0$. Kích thước là $7 \times 7 \times 3 \times 20$
- Tầng flatten để trãi thành vector có thành phần $7 \times 7 \times 20 = 980$
- Chẳng hạn để nhận ra hình có phải là chó hay không, ta chỉ cần qua hàm Sigmoid để có kết quả, nên tầng Output chỉ cần 1 neuron, sau đó qua hàm sigmoid để nhận biết.

Ví dụ

- Theo dataset MINST, dữ liệu là một ảnh xám kích thước 28×28, nên để sử dụng ta coi đây là ảnh với 1 channel có kích thước $28 \times 28 \times 1$.
- Xây dựng một CNN có cấu trúc như sau:
  - Tầng input với $28 \times 28 \times 1$ neuron
  - Tầng convolution 1: có 32 kernel, $s = 1$, $p = 0$, $k = 3$
    - Có $\left\lfloor \frac{28-3+0}{1} + 1 \right\rfloor \times \left\lfloor \frac{28-3+0}{1} + 1 \right\rfloor = 26 \times 26$
    - Qua tầng max pooling (2,2), với $s = 2$, $p = 0$: còn lại $13 \times 13 \times 32$

- Tầng convolution 2: có 32 kernel, $s = 1$, $p = 0$, $k = 3$
  - Có $\left\lfloor \frac{13-3+0}{1} + 1 \right\rfloor \times \left\lfloor \frac{13-3+0}{1} + 1 \right\rfloor = 11 \times 11$
  - Qua tầng max pooling (2,2), với $s = 2$, $p = 0$: còn lại $5 \times 5 \times 32$
- Tầng flatten để tạo ra fully connection có 800 thành phần

Thư viện Keras

Linear Regression

- Hiện nay có nhiều thư viện Python để hiện thực ứng dụng DL. Ở đây chúng ta dùng Keras cũng là sản phẩm của Google (Lưu ý rằng, khi sử dụng Keras cần có TensorFlow)
- Trước hết ta thử dùng hồi quy tuyến tính với Keras
- Giả sử hàm xấp xỉ cần tìm là $\varphi(x, w_0, w_1, w_2, w_3) = w_0 + w_1x_1 + w_2x_2 + w_3x_3$
- Gói thư viện cần chuẩn bị
-

```
import numpy as np
from tensorflow.keras import Sequential,optimizers
from tensorflow.keras.layers import Dense, Activation
```

- Giả sử x_1, x_2, x_3 là điểm của 3 môn học Toán, Văn và Anh văn. Trong đó Toán hệ số 3, Văn hệ số 3 còn Anh văn có hệ số 1.
- Tạo dữ liệu ngẫu nhiên để thử nghiệm bằng cách tạo mảng $y_{train} = 3x_1 + 2x_2 + x_3$ với $X_{train} = (x_1, x_2, x_3)$ trong đó $x_1, x_2, x_3 \in [0,10]$

```
N = 2000
X = np.random.rand(N,3)*10
y = 3*X[:,0] + 2*X[:,1] + X[:,2]
```

- Dành ra 1/4 dữ liệu để kiểm định

```
NN = 500
X_train = np.split( X,[N-NN] )[0]
X_valid = np.split( X,[N-NN] )[1]
y_train = np.split( y,[N-NN] )[0]
y_valid = np.split( y,[N-NN] )[1]
```

- Dùng lớp Sequential() để tạo các tầng theo trình tự thực hiện của phương thức add().

```
model = Sequential()
```

- Dense() được dùng để chỉ một *Fully Connected Layer*, (ANN này không có tầng hidden) và số neuron của tầng output là 1; còn input_shape=(3,) chỉ hình dạng của dữ liệu ở tầng input.

```
model.add( Dense(1,input_shape=(3,)) )
```

  - Chẳng hạn với ảnh màu có kích thước 64×64 pixel, thì

```
input_shape=(64,64,3)
```

- Activation() để chỉ hàm tác động, ở dây dùng hàm linear

```
model.add( Activation('linear') )
```

- Để tối ưu hàm mất mát, sử dụng stochastic gradient descent với hệ số $\lambda = 0.01$. Hàm mất mát dùng sai số trung bình phương (*Mean Squared Error*).

```
model.compile(loss='mse',optimizer=optimizers.SGD(learning_rate=0.01))
```

LinearRegression-Keras.py

- Sau khi xây dựng được mô hình và chỉ ra phương pháp tính hàm mất mát, bước tiếp theo huấn luyện mô hình: với 1500 dữ liệu, ta chia thành 10 lô (batch), mỗi lô là 150 dữ liệu. Như vậy cần 10 lần huấn luyện (epoch) cho một lô mới huấn luyện xong 1500 dữ liệu.

```
model.fit( X_train,y_train,epochs=10 )
```

- Bản chất việc huấn luyện là tìm ra trọng số (*và cả độ lệch*)

```
w = model.get_weights()
```

LinearRegression-Keras.py

- Sau đó sử dụng các trọng số này để tính kết quả dựa trên dữ liệu kiểm định

```
y_pred = w[0][0]*X_valid[:,0] + w[0][1]*X_valid[:,1] +
             w[0][2]*X_valid[:,2] + w[1]
```

- Từ đây so với kết quả kiểm định để đánh giá sai số

```
from sklearn.metrics import mean_absolute_error
print( "Mean Absolute Error: %10.8f" % mean_absolute_error(y_pred,y_valid) )
# Hoặc dùng
from tensorflow.keras.losses import MeanAbsoluteError
print( "Mean Absolute Error: %10.8f" % MeanAbsoluteError()(y_valid,
y_pred).numpy() )
```

Logistic Regression với Keras

- Nếu dùng hồi quy Logistic, thì hàm tác động phải là hàm sigmoid, và hàm mất mát phải là xác suất nên dùng cross_entropy để tính giá trị mất mát thay vì bình phương tối thiểu

```
from keras import losses
model.add(Activation('sigmoid'))
model.compile( loss=losses.binary_crossentropy,optimizer=
optimizers.SGD(lr=0.05) )
```

ANN với Keras

- Sử dụng dataset là các chữ số viết tay tại http://yann.lecun.com/exdb/mnist/
- Bao gồm 10 class, 60.000 ảnh cho training, 10.000 ảnh cho test, mỗi ảnh có kích thước 28×28 và là các ảnh xám nên chỉ có 1 channel (thay vì 3 channel)
- Có thể download trước, hoặc sử dụng trực tiếp từ gói dataset của Keras

```
from keras.datasets import mnist
(X_train,y_train), (X_test,y_test) = mnist.load_data()
```

THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

Four files are available on this site:

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:   test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:   test set labels (4542 bytes)
```

- Trong 60.000 data point này ta chỉ dùng 55.000 data point để huấn luyện, còn 5.000 data point để kiểm định (validation data). Và vẫn giữ nguyên cho dữ liệu test

```
X_valid, y_valid = X_train[55000:60000,:], y_train[55000:60000]
X_train, y_train = X_train[:55000,:], y_train[:55000]
print( 'X_train shape:',X_train.shape )
print( 'y_train shape:',y_train.shape )
print( 'X_valid shape:',X_valid.shape )
print( 'X_test  shape :',X_test.shape )
```

- Các thư viện cần thiết

```
from keras.layers import Dense, Flatten
from keras.models import Sequential
from keras.utils import to_categorical
from keras import metrics, losses, optimizers
import matplotlib.pyplot as plt
import numpy as np
```

- Chuẩn hoá dữ liệu để đưa về giá trị thuộc [0,1] (dạng không thứ nguyên)

```
X_train = X_train/255
X_valid = X_valid/255
X_test = X_test/255
```

- Do có 10 chữ số nên, đồng thời thực hiện việc one-hot coding, nhằm để có sự phân loại rành mạch giữa các nhãn.
- Chẳng hạn, trước khi xử lý, y_train[9] có giá trị là 3, sau khi xử lý có giá trị là [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]

```
NUM_CLASSES = 10
y_train = to_categorical( y_train,NUM_CLASSES )
y_valid = to_categorical( y_valid,NUM_CLASSES )
y_test = to_categorical( y_test,NUM_CLASSES )
```

- Sau đó xây dựng một cấu trúc ANN 4-tầng:
  - Tầng Input với số neuron là $28 \times 28 = 784$ được trải dài thành một vector
  - Tầng Hidden thứ nhất có 128 neuron có hàm các động là ReLU.
  - Tầng Hidden thứ hai với 256 neuron có hàm các động là ReLU.
  - Tầng Hidden thứ ba với 512 neuron có hàm các động là ReLU.
  - Tầng Output có 10 neuron với một softmax Layer để phân lớp

```
model = Sequential()
model.add( Flatten(input_shape=(28,28)) )
model.add( Dense(128,activation='relu') )
model.add( Dense(256,activation='relu') )
model.add( Dense(512,activation='relu') )
model.add( Dense(NUM_CLASSES,activation='softmax') )
```

```
1 print( H.history['val_accuracy'] )
```

```
[0.9772999882698059, 0.980400025844574]
```

```
1 score = model.evaluate( X_test,y_test )
2 print('Mất mát: %.4f'% score[0])
3 print('Độ chính xác %.4f'% score[1])
```

```
10000/10000 [==============================] - 0s 41us/step
Mất mát: 0.0727
Độ chính xác 0.9796
```

MLP-Keras.py

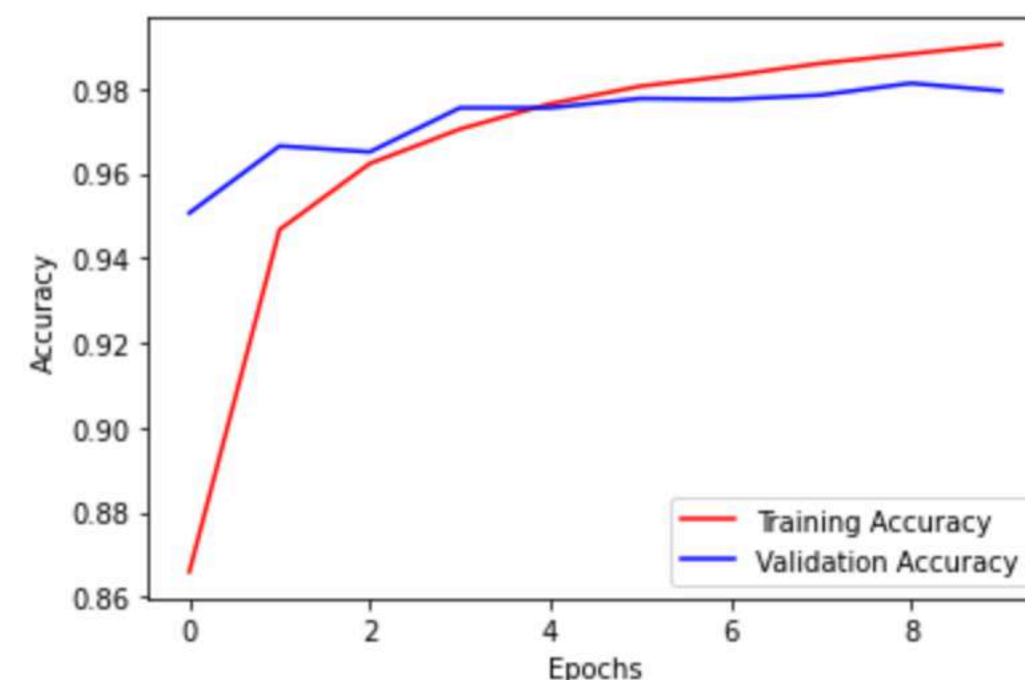
- Để rồi huấn luyện (dùng verbose=0 để không hiển thị những gì đang làm)

```
model.compile(loss=losses.categorical_crossentropy,optimizer=optimizers.SGD(lr=0.1),metrics=['accuracy'])
M = model.fit( X_train,y_train,validation_data=(X_valid,y_valid),epochs=100 )
```

- Kết quả đánh giá trên tập kiểm thử có độ mất mát và độ chính xác

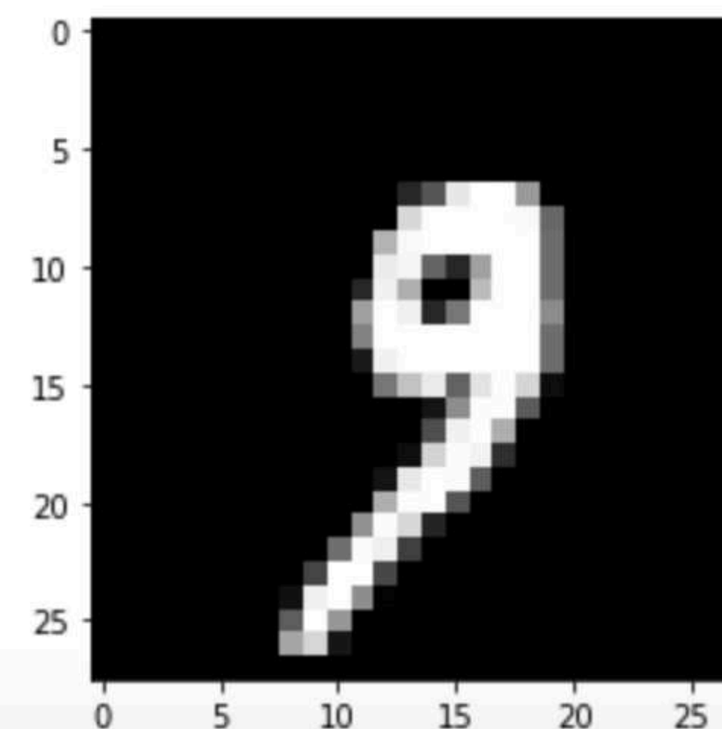
```
score = model.evaluate( X_test,y_test,verbose=0 ) # Để không xuất ra những gì đang xử lý
print('Mất mát: %.4f'% score[0])
print('Độ chính xác %.4f'% score[1])
```

```
1 N = np.arange(0,EPOCHS)
2 plt.plot( N,H.history["accuracy"],color="r",label="Training Accuracy" )
3 plt.plot( N,H.history["val_accuracy"],color="b",label="Validation Accuracy" )
4 plt.xlabel( "Epochs" )
5 plt.ylabel( "Accuracy" )
6 plt.legend()
7 plt.show()
```




```
1 ifig = int(input("Kiểm tra hình thứ (từ 0 đến 9999): "))
2 plt.imshow( X_test[ifig], cmap='gray' )
3 y_pred = model.predict( X_test )
4 print( 'Hình này được dự đoán là số:',np.argmax(y_pred[ifig]) )
5 print( 'Với xác suất là %.5f:' % y_pred[ifig,np.argmax(y_pred[ifig])] )
```

```
Kiểm tra hình thứ (từ 0 đến 9999): 1000
Hình này được dự đoán là số: 9
Với xác suất là 0.99929:
```



- Hoặc kiểm tra bằng hình ảnh

```
ifig = int(input("Kiểm tra hình thứ (từ 0 đến 9999): "))
plt.imshow( X_test[ifig], cmap='gray' )
y_pred = model.predict( X_test )
print( 'Hình này được dự đoán là số:',np.argmax(y_pred[ifig]) )
print( 'Với xác suất là %.5f:' % y_pred[ifig,np.argmax(y_pred[ifig])] )
```

Lưu ý

Về softmax

- Khi dự đoán ta có thể cho biết kết quả dự đoán là gì. Tuy nhiên, trong một số trường hợp cần phải chỉ ra khả năng mà kết quả đó xãy ra.
- Trong trường hợp này cần đến hàm kích hoạt là hàm Softmax
- Gọi $z_i, \forall i = \overline{1,N_O}$ là các kết quả ở tầng Output, thì hàm kích hoạt Softmax

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N_O} e^{z_j}}, \forall i = \overline{1,N_O}$$

- Từ đây dễ dàng suy ra $\sum_{i=1}^{N_O} \sigma(z_i) = 1$ và $0 < \sigma(z_i) < 1, \forall i = \overline{1,N_O}$

Hiện thực CNN với Keras

- Trước hết, để dễ so sánh ta dùng dataset là kiểu chữ số viết tay như trong ANN.
- Lưu ý, giả sử dataset có số lượng data point cho một lần huấn luyện là $N_I = 55000$, thì với ảnh xám kích thước 28×28 ta có dữ liệu cần thực hiện cho tầng Convolution đầu tiên là tensor $N_I \times 28 \times 28 \times 1$
- Nên cần định dạng lại dữ liệu để sử dụng

```
X_train = X_train.reshape( X_train.shape[0],28,28,1 )
X_valid = X_valid.reshape( X_valid.shape[0],28,28,1 )
X_test  = X_test.reshape(  X_test.shape[0],28,28,1 )
```

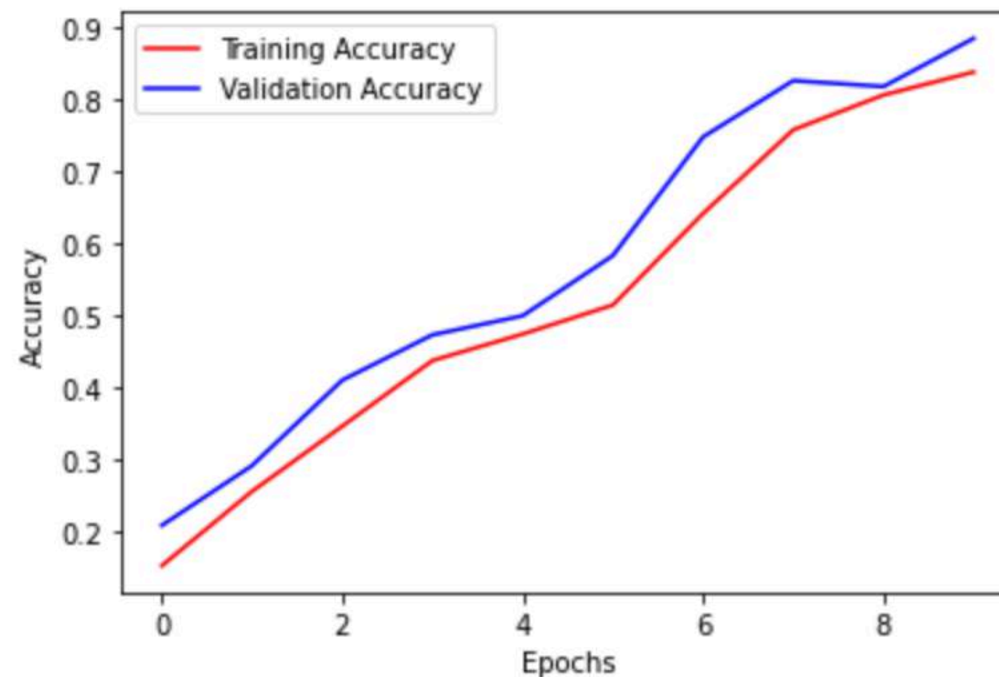
- Khi đó, có thể xây dựng một CNN gồm các tầng như sau:

```
model = Sequential()
model.add( Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)) )
model.add( MaxPooling2D(pool_size=(2,2)) )
model.add( Conv2D(32,(3,3),activation='relu') )
model.add( MaxPooling2D(pool_size=(2,2)) )
model.add( Flatten() )
model.add( Dense(128,activation='relu') )
model.add( Dense(256,activation='relu') )
model.add( Dense(512,activation='relu') )
model.add( Dense(NUM_CLASSES,activation='softmax') )
```

- Ngoài ra epoch là số lần duyệt qua hết tập huấn luyện.
- Giả sử tập huấn luyện gồm 55.000 data point, batch-size là 100 có nghĩa là mỗi lần cập nhật trọng số, ta dùng 100 data. Suy ra cần 55.000/100 = 550 lần lặp để duyệt qua hết tập huấn luyện (hoàn thành 1 epochs).
- Huấn luyện với thuật toán huấn luyện tối ưu có thể là Dùng để chọn thuật toán huấn luyện có thể là SGD, RMSprop, hoặc Adam. Còn hàm mất mát khi 2 lớp dùng binary_crossentropy, còn nhiều lớp phải dùng categorical_crossentropy.

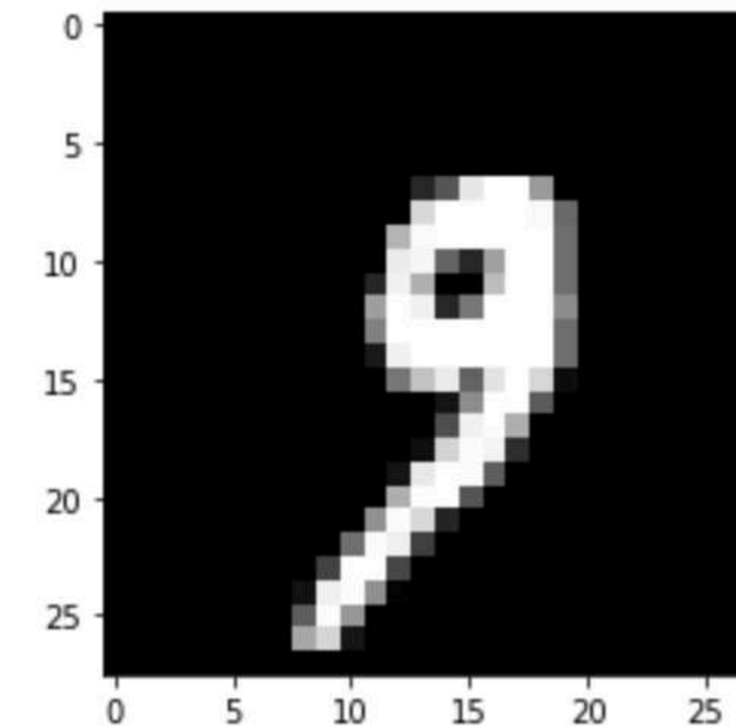
```
model.compile( loss=losses.categorical_crossentropy,optimizer='sgd',metrics=['accuracy'] )
H = model.fit( X_train,y_train,validation_data=(X_valid,y_valid),epochs=2,batch_size=100 )
```

```
1 N = np.arange(0,EPOCHS)
2 plt.plot( N,H.history["accuracy"],color="r",label="Training Accuracy" )
3 plt.plot( N,H.history["val_accuracy"],color="b",label="Validation Accuracy" )
4 plt.xlabel( "Epochs" )
5 plt.ylabel( "Accuracy" )
6 plt.legend()
7 plt.show()
```




```
1 ifig = int(input("Kiểm tra hình thứ (từ 0 đến 9999): "))
2 plt.imshow( X_test[ifig].reshape(28,28), cmap='gray' )
3 y_pred = model.predict( X_test )
4 print( 'Hình này được dự đoán là số:',np.argmax(y_pred[ifig]) )
5 print( 'Với xác suất là %.5f:' % y_pred[ifig,np.argmax(y_pred[ifig])] )
```

```
Kiểm tra hình thứ (từ 0 đến 9999): 1000
Hình này được dự đoán là số: 9
Với xác suất là 0.99998:
```



- Hoặc kiểm tra bằng hình ảnh

```
ifig = int(input("Kiểm tra hình thứ (từ 0 đến 9999): "))
plt.imshow( X_test[ifig].reshape(28,28),cmap='gray' )
y_pred = model.predict( X_test )
print( 'Hình này được dự đoán là số:',np.argmax(y_pred[ifig]) )
print( 'Với xác suất là %.5f:' % y_pred[ifig,np.argmax(y_pred[ifig])] )
```

CNN-Keras.py

- Lưu ý thêm, khi cần coi cấu trúc của CNN đã hiện thực:

```
print( "Cấu trúc của CNN" )
model.summary()
```

```
Cấu trúc của CNN
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 11, 11, 32)        9248
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 5, 5, 32)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_21 (Dense)             (None, 128)               102528
_________________________________________________________________
dense_22 (Dense)             (None, 256)               33024
_________________________________________________________________
dense_23 (Dense)             (None, 512)               131584
_________________________________________________________________
dense_24 (Dense)             (None, 10)                5130
=================================================================
Total params: 281,834
Trainable params: 281,834
Non-trainable params: 0
_________________________________________________________________
```

- Ngoài ra, để trực quan hoá cấu trúc CNN, cần install thêm

  brew install graphviz

  pip intall pydot

- Sau đó thực thi

```
model = Sequential()
model.add( Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)) )
model.add( MaxPooling2D(pool_size=(2,2)) )
model.add( Conv2D(32,(3,3),activation='relu') )
```

Mạng Tế bào thần kinh tái diễn

Giới thiệu

- Mạng Tế bào thần kinh tái diễn (RNN-Recurrent Neuron Network) với ý tưởng sử dụng thêm biến để lưu lại thông tin từ những bước xử lý trước đó, từ đó đưa ra dự đoán cho bước tiếp theo
- Nên RNN thường được sử dụng cho những dataset có dạng liên kết với nhau. Chẳng hạn,
    - Cần nhận dạng hành vi của người qua những cử động, khi đó video gồm nhiều frame hình mà mỗi frame hình là một hình ảnh và hình ảnh này được liên kết với nhau tạo nên một chuỗi các hình ảnh.
    - Hoặc, với các dữ liệu y khoa khi cần theo dõi cả một dãy những kết quả khám bệnh của một người ở những lần khám bệnh trước đó.

- Hoặc trong xử lý ngôn ngữ tự nhiên (NLP-Natural Language Processing), việc hiểu nghĩa hoàn chỉnh một câu cần có cả ngữ cảnh (những câu trước và sau đó) đóng vai trò quan trọng.
- Ngoài việc dịch ra, với NLP người ta còn tập trung vào việc sinh văn bản (tập làm văn); đây là một lĩnh vực rất phát triển hiện nay.
- Trong kinh tế, chẳng hạn để dự đoán chứng khoán của một công ty thì những dữ liệu trước đó hoặc những kết quả hoạt động tài chính theo thời gian là một yếu tố quan trọng.

- Cũng chính vì đặc điểm này nên RNN còn được hiểu theo nghĩa là Mạng tế bào thần kinh hồi quy.

Cấu trúc

- Trong ANN, một bước tính toán bao gồm với giá trị đầu vào là x_t, tính $h_t = Wx_t$, sau đó thông qua hàm tác động $\varphi()$, chẳng hạn hàm $sigmod()$ (hoặc $tanh()$ hay $reLU()$) để tìm giá trị đầu ra $y_t = \varphi(h_t)$.
- Nhưng RNN có một số bổ sung thêm ở bước $h_t = f(W_h h_{t-1} + Wx_t)$ được tính toán dựa trên bước trước đó, rồi tính $y_t = g(h_t)$. Ở đây $f()$ thường là hàm tanh() hay reLU(); còn $g()$ là hàm sigmoid() hay softmax().
- Trong RNN, bước tính ở tầng Hidden này được gọi là trạng thái (State), nên thay vì h_t người ta hay ký hiệu là s_t với $s_t = f(W_s s_{t-1} + Wx_t)$, trong đó giá trị ban đầu s_o được cho bởi một giá trị khá nhỏ nào đó.

Minh hoạ

- Giả sử có 24 frame hình trong một cảnh video, cần biết kết quả đầu ra là gì.
- Cấu trúc RNN như sau:
    - Gọi x_t là một frame hình (là một vector được mô tả là $N \times 1$)
    - Một vector trạng thái s_t kích thước $M \times 1$
        - $s_o = 0, s_t = f(Wx_t + W_s s_{t-1})$ với f là hàm kích hoạt, thông thường là hàm tanH hay reLU
        - W có kích thước là $M \times N$, W_s có kích thước là $M \times M$
    - Vector đầu ra $y_t = g(W_k s_t)$ kích thước là $K \times 1$ (trong đó W_k kích thước $K \times M$)
- Khi đó kết quả sau cùng là $g(W_k s_{24})$ với g là hàm sigmoid() hay softmax() để phân loại (do frame hình 24 là hình sau cùng của một cảnh video)

- Qua ví dụ minh hoạ cho thấy RNN coi dữ liệu đầu vào là một chuỗi liên tục, nối tiếp nhau theo thời gian. Tại thời điểm t, với dữ liệu đầu vào x_t cho ra kết quả output y_t
- Tuy nhiên, khác với ANN thông thường, y_t lại được sử dụng là đầu vào để tính kết quả đầu ra cho thời điểm $t + 1$. Từ đó cho phép RNN lưu trữ và truyền thông tin đến thời điểm tiếp theo.
- RNN sử dụng 3 ma trận trọng số cho 2 quá trình tính toán, trong đó
  - Quá trình thứ nhất: W_s kết hợp với trạng thái trước đó và W kết hợp với đầu vào từ đó tính ra trạng thái hiện tại.
  - Quá trình thứ hai: W_k kết hợp trạng thái hiện tại tính đầu ra.

- Lưu ý với ANN, đầu ra của một tầng (Layer) là đầu vào của tầng tiếp theo. Tuy nhiên, cũng có những khác nhau giữa RNN và ANN như sau:
  - ANN sử dụng các ma trận trọng số (*W*) khác nhau cho từng tầng. Mạng có bao nhiêu tầng thì có bấy nhiêu ma trận trọng số.
  - RNN chỉ sử dụng một mạng Neuron duy nhất (thường là 1 layer) để tính giá trị output của từng timestep. Do đó các outputs khi trở thành input sẽ được nhân với cùng một weights matrix (ở đây là W_R như trong Hình 2). Đây cũng chính là lý do tại sao có từ Recurrent trong tên của RNN. Recurrent có nghĩa là mô hình sẽ thực hiện các phép tính toán giống hệt nhau cho từng phần tử của chuỗi dữ liệu đầu vào và kết quả output sẽ phụ thuộc vào kết quả của các tính toán ở phần trước.