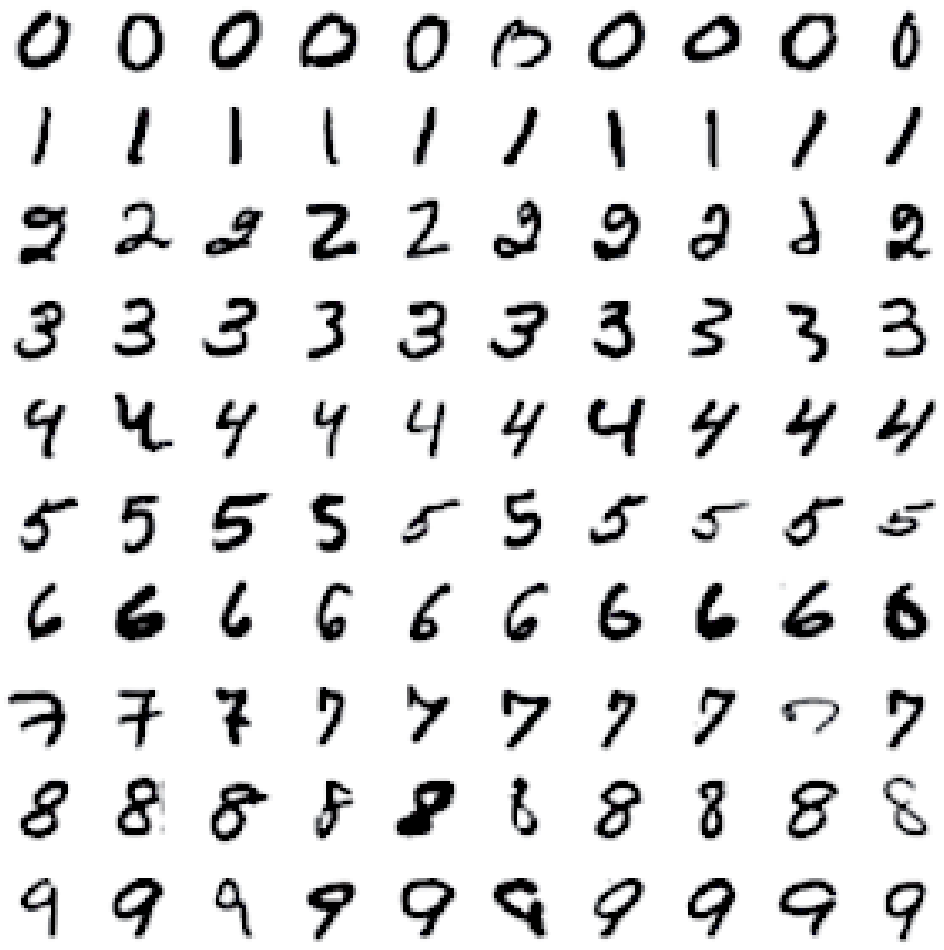
Convolutional Neuron Network

MNIST dataset

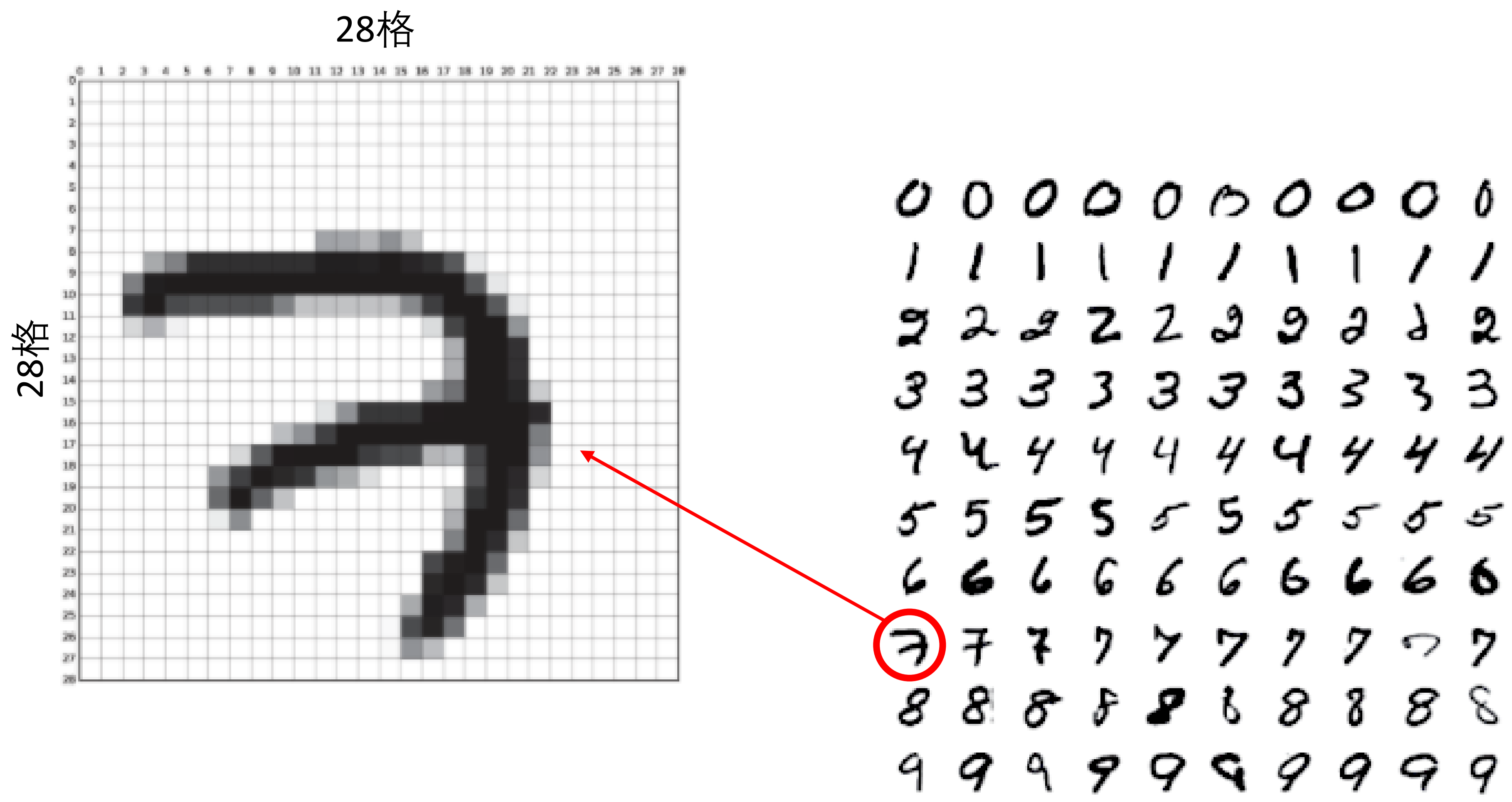
MNIST stands for Modified National Institute of Standards and Technology and is often used as a benchmark in machine learning research.

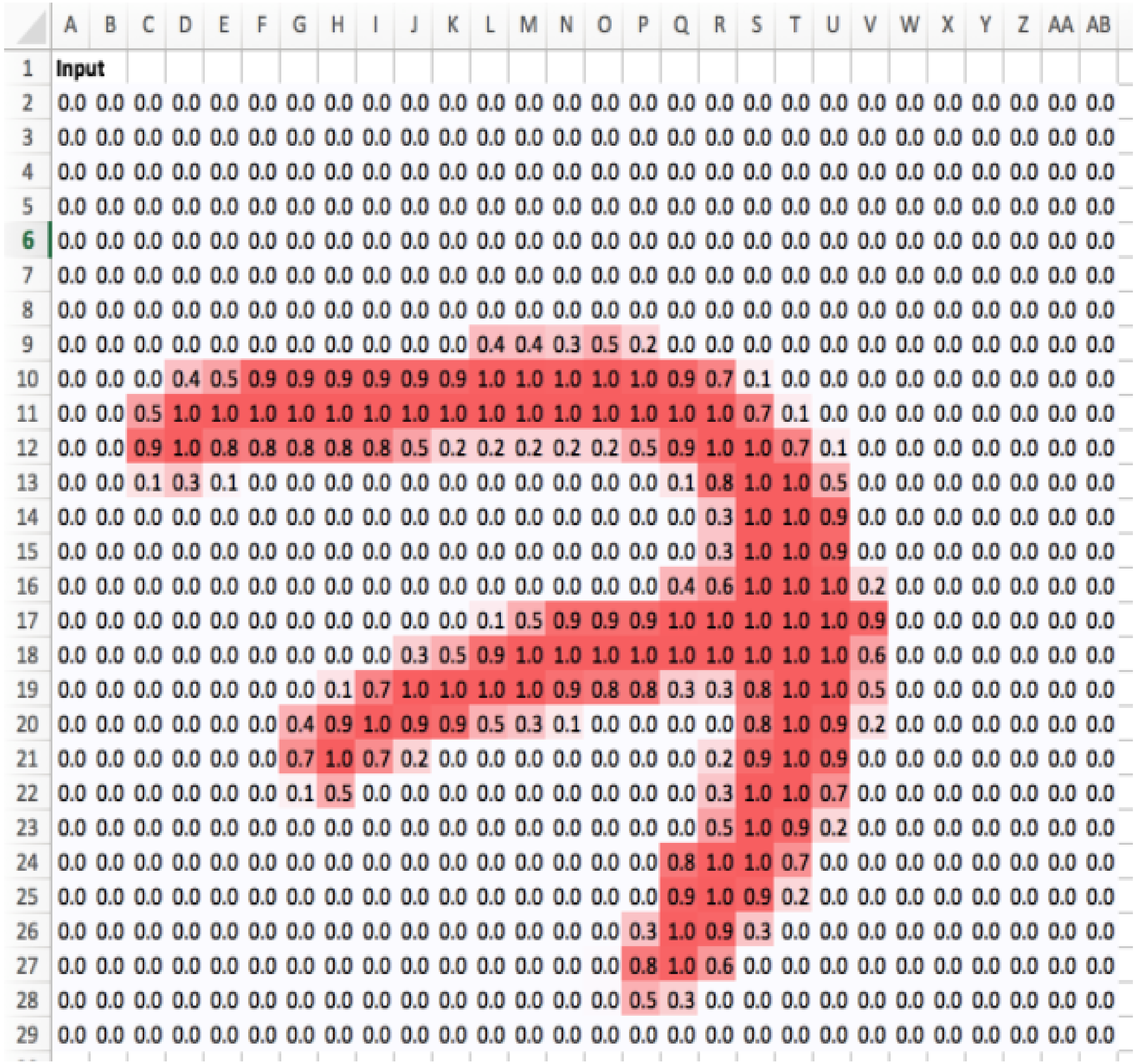
The MNIST dataset consists of a large collection of handwritten digits, commonly used for training various machine learning algorithms. Each image in the dataset is a grayscale image of size **28x28 pixels**, representing a single digit from 0 to 9. MNIST is a popular benchmark dataset in the field of machine learning and computer vision.

It consists of 60,000 training images and 10,000 testing images.

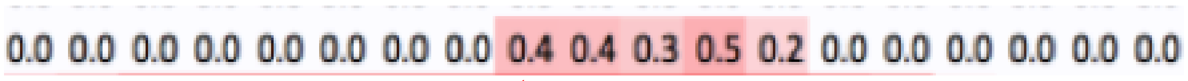


http://yann.lecun.com/exdb/mnist/

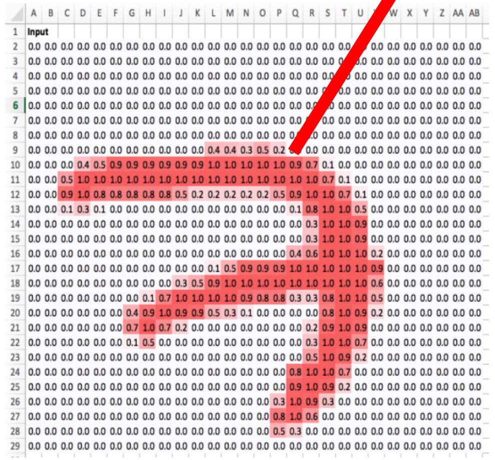

28格
28格



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1	**Input**																											
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.4	0.3	0.5	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.4	0.5	0.9	0.9	0.9	0.9	0.9	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12	0.0	0.0	0.9	1.0	0.8	0.8	0.8	0.8	0.8	0.5	0.2	0.2	0.2	0.2	0.2	0.5	0.9	1.0	1.0	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.1	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.8	1.0	1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.6	1.0	1.0	1.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0
17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.5	0.9	0.9	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0
18	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.5	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0
19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.7	1.0	1.0	1.0	1.0	0.9	0.8	0.8	0.3	0.3	0.8	1.0	1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0
20	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.9	1.0	0.9	0.9	0.5	0.3	0.1	0.0	0.0	0.0	0.0	0.8	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0
21	0.0	0.0	0.0	0.0	0.0	0.0	0.7	1.0	0.7	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.9	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
22	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	1.0	1.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
26	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	0.9	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
27	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	1.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
28	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0



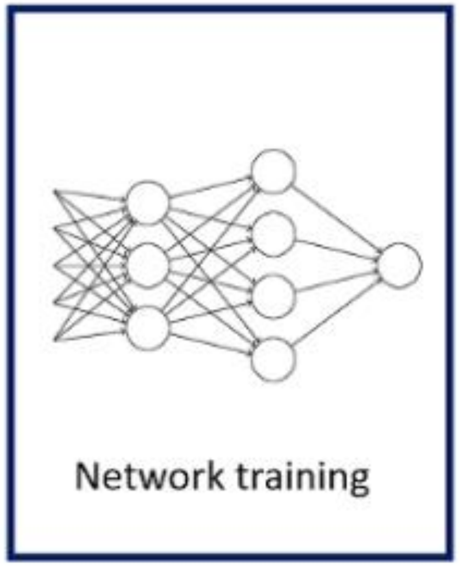
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.4	0.3	0.5	0.2	0.0	0.0	0.0	0.0	0.0	0.0



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1	Input																											
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	[illegible]	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	[illegible]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.4	0.3	0.5	0.2	[illegible]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.4	0.5	0.9	0.9	0.9	0.9	0.9	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12	0.0	0.0	0.9	1.0	0.8	0.8	0.8	0.8	0.8	0.5	0.2	0.2	0.2	0.2	0.2	0.5	0.9	1.0	1.0	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.1	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.8	1.0	1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.6	1.0	1.0	1.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0
17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.5	0.9	0.9	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0
18	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.5	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0
19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.7	1.0	1.0	1.0	1.0	0.9	0.8	0.8	0.3	0.3	0.8	1.0	1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0
20	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.9	1.0	0.9	0.9	0.5	0.3	0.1	0.0	0.0	0.0	0.0	0.8	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0
21	0.0	0.0	0.0	0.0	0.0	0.0	0.7	1.0	0.7	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.9	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
22	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	1.0	1.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
26	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	0.9	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
27	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	1.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
28	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Flatten

.
.
.
0
0
0
0
0.4
0.4
0.3
0.5
0.2
.
.
.



0
1
2
3
4
5
6
7
8
9

Flatten

28x28 = 784 features

Convolution

Convolution in the context of neural networks refers to the mathematical operation of combining two functions to produce a third function that represents how one function modifies the shape of the other. In the context of Convolutional Neural Networks (CNNs), convolution involves applying a **filter (also known as a kernel)** to an input image or feature map to **extract various features**.

卷積（Convolution）是一種數學運算，用於從輸入數據中提取特徵。卷積通常涉及將一個稱為卷積核或濾波器的小矩陣應用於輸入數據的不同區域，然後將它們的乘積總和起來以產生輸出特徵圖。

1	2	3	1	1
4	5	6	1	1
7	8	9	2	3
1	1	1	1	1
1	2	1	0	1
1	3	4	2	1
1	1	3	4	4

⋮

28x28

......

0	1	0
1	1	1
0	1	0

convolutional kernel

1x0 + 2x1 + 3x0
4x1 + 5x1 + 6x1
7x0 + 8x1 + 9x0
= 25

25				

Feature map

Types of Convolutional kernels

1. **Edge detectors/邊緣檢測器**: Used for detecting edges and contours in images. Common edge detectors include Sobel, Prewitt, and Scharr kernels.
2. **Blur kernels/模糊核**: Also known as smoothing kernels, used to reduce noise and details in images to achieve a blurred effect. Common blur kernels include Gaussian and mean kernels.
3. **Sharpening kernels/增強核**: Used to enhance edges and details in images to improve sharpness. Sharpening kernels are typically a sharpened version to make details in the image more prominent.
4. **Depthwise convolution kernels/深度卷積核**: Used in depthwise separable convolutions to independently process each input channel, reducing computational cost and improving efficiency.
5. **Pooling kernels/分類核**: Typically used in conjunction with convolutional kernels for reducing the size of the input feature maps, thereby reducing computational cost and increasing translation invariance.
6. **Transpose convolution kernels/轉置卷積核**: Also known as deconvolution kernels, used for upsampling operations to increase the size of feature maps.

Edge detectors/邊緣檢測器

pip install opencv-python

```
import cv2
import numpy as np

# 讀取圖像
image = cv2.imread('input_image.jpg', cv2.IMREAD_GRAYSCALE)

# 檢測邊緣
sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)

# 組合X和Y方向的邊緣檢測結果
sobel_combined = cv2.addWeighted(sobel_x, 0.5, sobel_y, 0.5, 0)

# 顯示結果
cv2.imshow('Original Image', image)
cv2.imshow('Sobel X', sobel_x)
cv2.imshow('Sobel Y', sobel_y)
cv2.imshow('Sobel Combined', sobel_combined)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Blur kernels/模糊核

```
import cv2

# 讀取圖像
image = cv2.imread('input_image.jpg')

# 定義模糊核大小
kernel_size = (5, 5) # 這裡使用一個5x5的模糊核

# 應用均值模糊
blurred_image = cv2.blur(image, kernel_size)

# 顯示原始圖像和模糊後的圖像
cv2.imshow('Original Image', image)
cv2.imshow('Blurred Image', blurred_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Sharpening kernels/增強核

```
import cv2
import numpy as np

def sharpening_kernel(image, kernel_size=(3, 3), strength=1.0):
# Define the sharpening kernel
kernel = np.array([[-1, -1, -1],
[-1, 9, -1],
[-1, -1, -1]]) * strength
# Apply the kernel to the image
sharpened_image = cv2.filter2D(image, -1, kernel)
return sharpened_image

# Load an example image
image = cv2.imread('example_image.jpg')

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply sharpening kernel to the grayscale image
sharpened_image = sharpening_kernel(gray_image, strength=1.5)

# Display the original and sharpened images
cv2.imshow('Original Image', gray_image)
cv2.imshow('Sharpened Image', sharpened_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Depthwise convolution kernels/深度卷積核

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import transforms
from PIL import Image
import matplotlib.pyplot as plt

# 讀入圖像
image = Image.open("example_image.jpg")

# 轉換圖像為 PyTorch tensor 格式
preprocess = transforms.Compose([
transforms.ToTensor(),
])
image_tensor = preprocess(image).unsqueeze(0)

# 定義深度卷積模型
class DepthwiseConvModel(nn.Module):
def __init__(self):
super(DepthwiseConvModel, self).__init__()
# 使用深度卷積進行特徵提取
self.depthwise_conv = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3,
groups=3, padding=1)

def forward(self, x):
x = self.depthwise_conv(x)
return x
```

```
# 初始化模型
model = DepthwiseConvModel()

# 執行深度卷積
with torch.no_grad():
output = model(image_tensor)

# 將輸出轉換回圖像格式
output_image = transforms.ToPILImage()(output.squeeze(0))

# 顯示原始圖像和深度卷積後的圖像
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image)
plt.axis('off')
plt.subplot(1, 2, 2)
plt.title('Depthwise Convolution Output')
plt.imshow(output_image)
plt.axis('off')
plt.show()
```

Pooling kernels/分類核

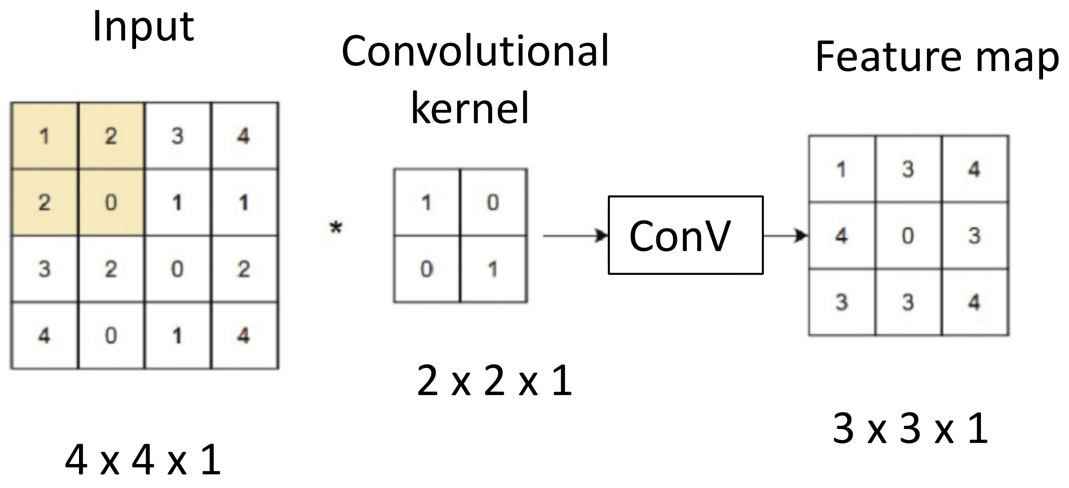
```
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from PIL import Image

# 讀入圖像
image = Image.open("example_image.jpg")

# 轉換圖像為PyTorch Tensor
transform = transforms.Compose([
transforms.Resize((224, 224)), # 將圖像大小調整為模型的輸入尺寸
transforms.ToTensor() # 將圖像轉換為Tensor
])
image = transform(image).unsqueeze(0) # 添加一個batch維度

# 定義池化層
pooling_layer = nn.MaxPool2d(kernel_size=2, stride=2)

# 執行池化操作
output = pooling_layer(image)

# 輸出池化後的圖像尺寸
print("輸入圖像尺寸:", image.shape)
print("池化後的圖像尺寸:", output.shape)
```

The size of Feature map

input feature

$$h = hi - hf + 1$$

$$w = wi - wf + 1$$



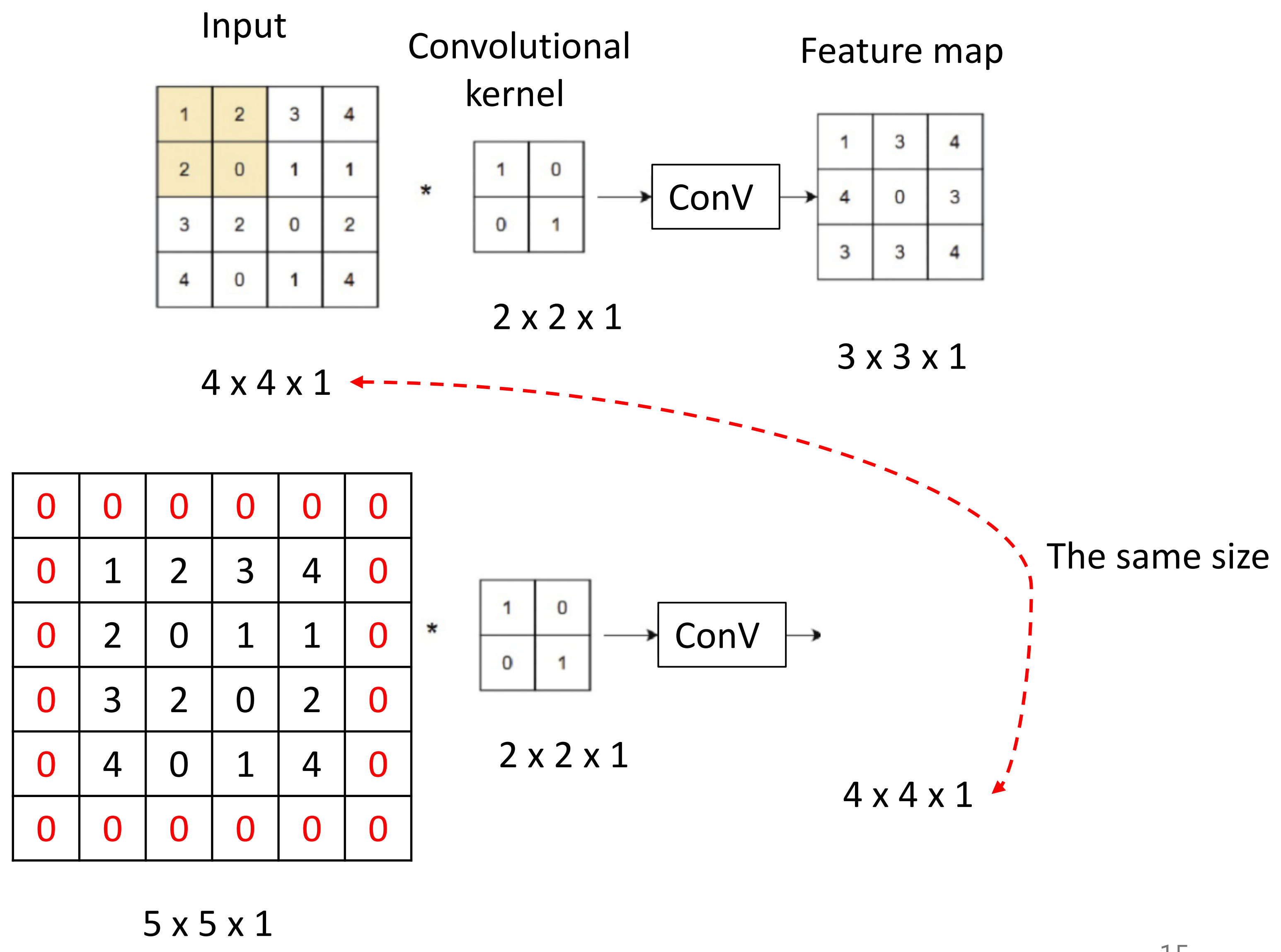
Padding

$$h = hi - hf + 1$$

$$w = wi - wf + 1$$

$$h = hi - hf + 2 * P + 1$$

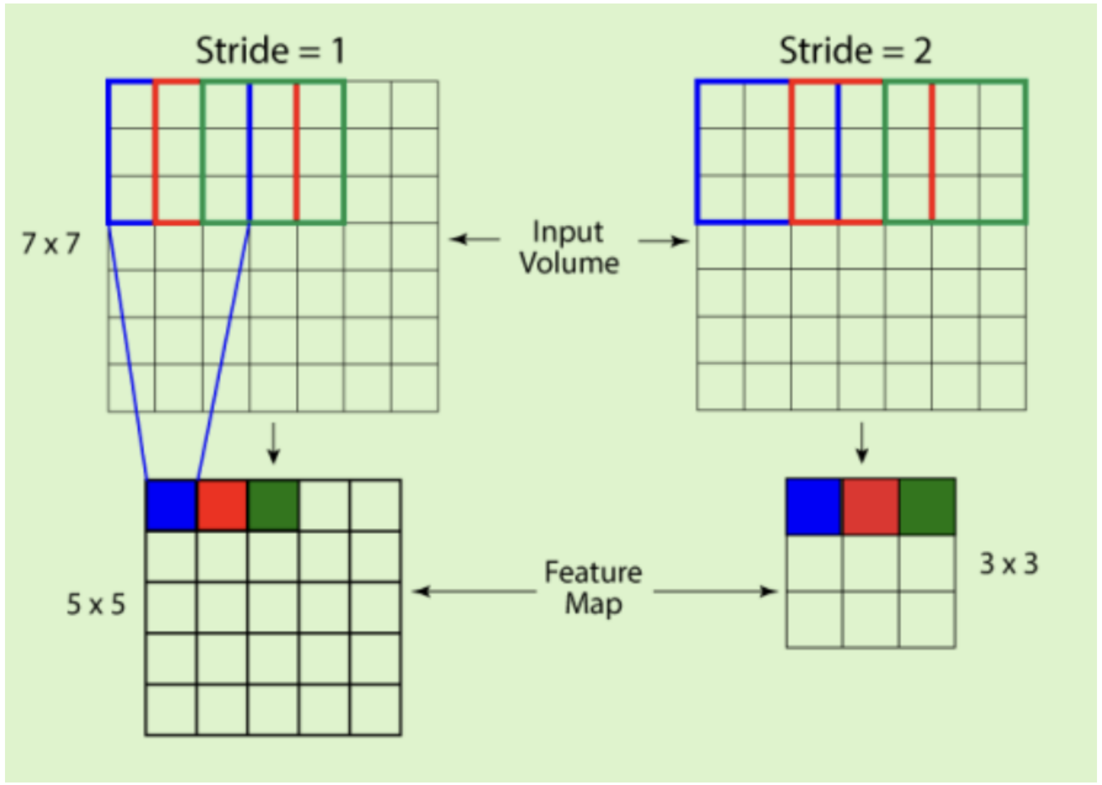
$$w = wi - wf + 2 * P + 1$$



Stride

- The "stride" in CNN, or Convolutional Neural Network, refers to the number of pixels by which the filter/kernel is slid over the input image. It determines how much the filter moves between each application of the filter to the input volume. In simpler terms, it controls the step size of the filter as it moves across the input image.
- A larger stride value means the filter skips more pixels as it moves, resulting in a smaller output volume spatially. Conversely, a smaller stride value means the filter moves more slowly, resulting in a larger output volume spatially.

Stride 是指在應用卷積核 (convolutional kernel) 進行過濾時，在輸入資料上移動的步長。這個步長可以控制著卷積操作過程中，卷積核滑動的間距。當 Stride 越大時，輸出特徵圖 (output feature map) 的尺寸會越小；反之，當 Stride 越小，輸出特徵圖的尺寸會越大。Stride 用於控制著特徵提取的密度和輸出尺寸的調整。

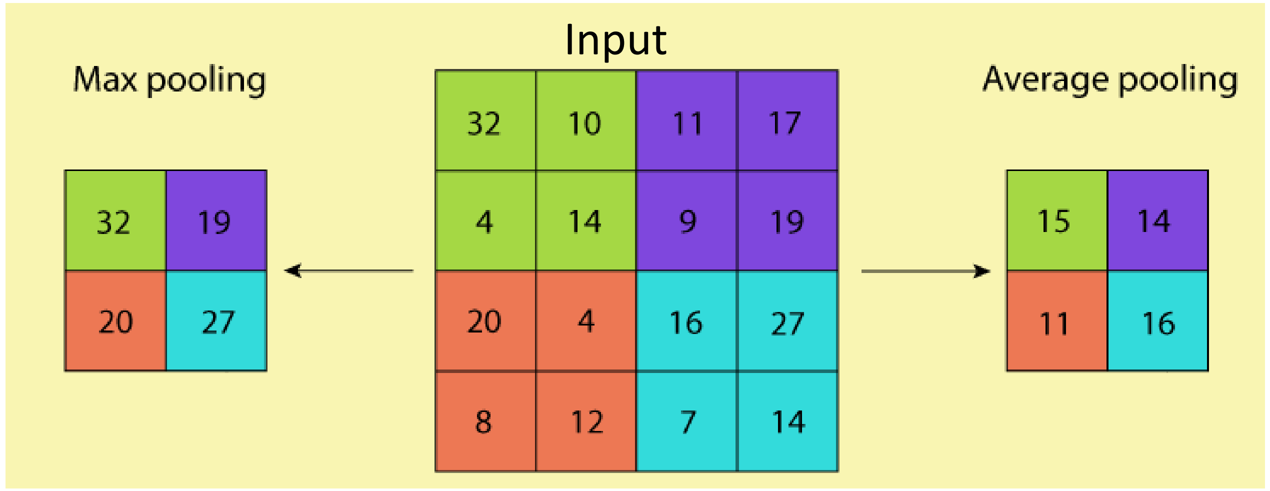


e from:
://developersbreach.com/convolution-neural-network-deep-learning/

Pooling

Pooling in a Convolutional Neural Network (CNN) is a technique used to **reduce the spatial dimensions** (width and height) of the input volume while retaining the most important information. It helps in controlling overfitting and reducing computational complexity.

There are different types of pooling layers, such as max pooling and average pooling.



gure from:
tps://developersbreach.com/convolution-neural-network-deep-learning/

Size of feature map

input feature

$h = hi - hf + 1$

$w = wi - wf + 1$

Padding

$h = hi - hf + 2 * P + 1$

$w = wi - wf + 2 * P + 1$

Stride

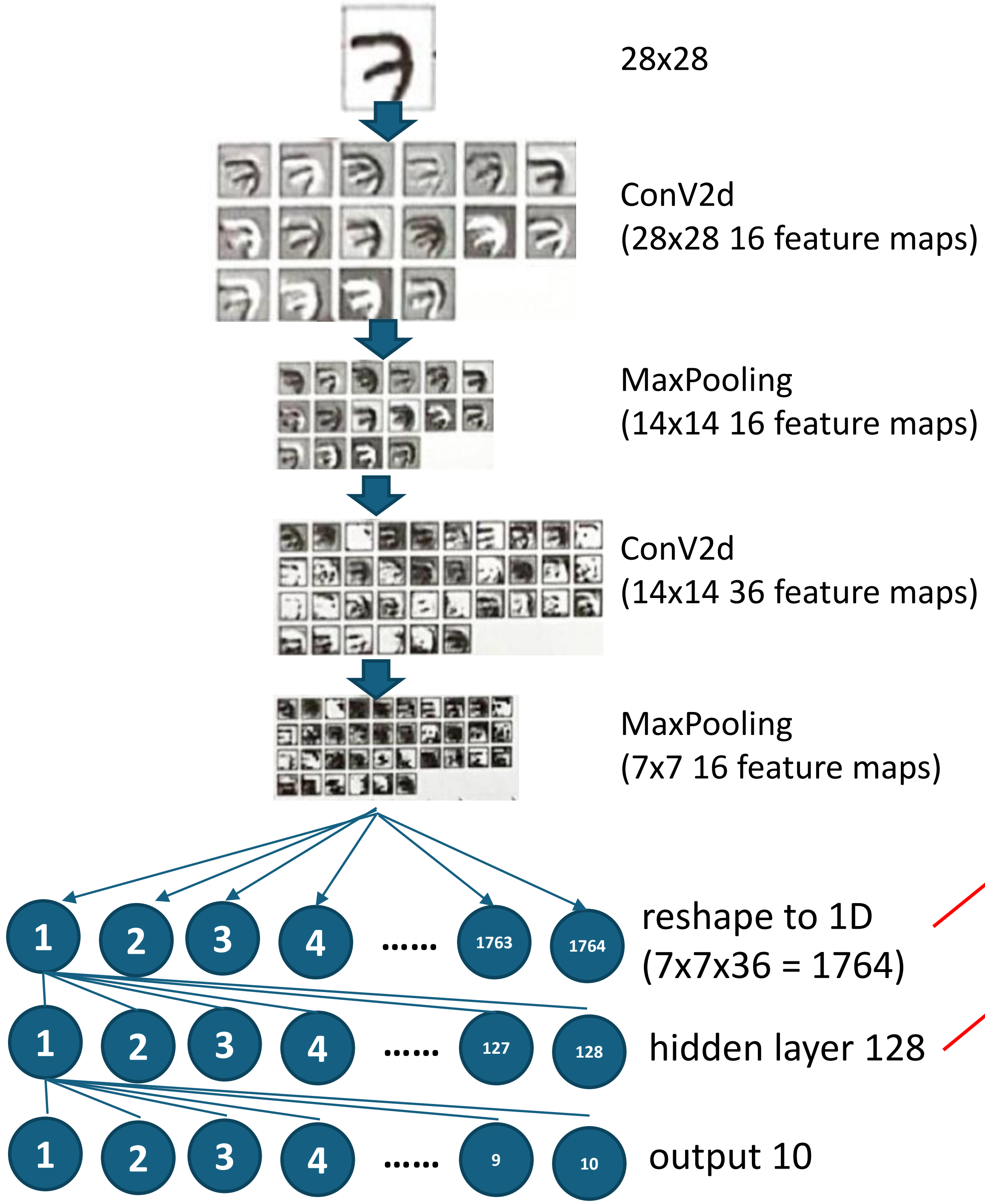
$h = (hi - hf + 2 * P) / S + 1$

$w = (wi - wf + 2 * P) / S + 1$

32 x 32 x 1

2 x 2 x 1

h = (32-2+2)/2 + 1 = 17
w = (32-2+2)/2 + 1 = 17



Flattening: Before passing the output of the convolutional and pooling layers to the FC layers, the feature maps are flattened into a one-dimensional vector. This flattening operation converts the spatial information into a linear representation.

Fully Connected Layers: The flattened feature vector is then passed through one or more FC layers. Each neuron in these layers is connected to every neuron in the previous and subsequent layers. The FC layers perform non-linear transformations on the input data, allowing the network to learn complex patterns and relationships.

Output Layer

Convolutional layer

```
torch.nn.Conv2d(
    in_channels,          # 輸入通道數
    out_channels,         # 輸出通道數（卷積核數）
    kernel_size,          # 卷積核大小
    stride=1,             # 步幅大小
    padding=0,            # 填充大小
)
```

```
import torch.nn as nn
import torch.nn.functional as F

class CNN_network(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 18, kernel_size=3, stride=1, padding=1)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return x
```

Add one Convolutional layer
Activation function: relu

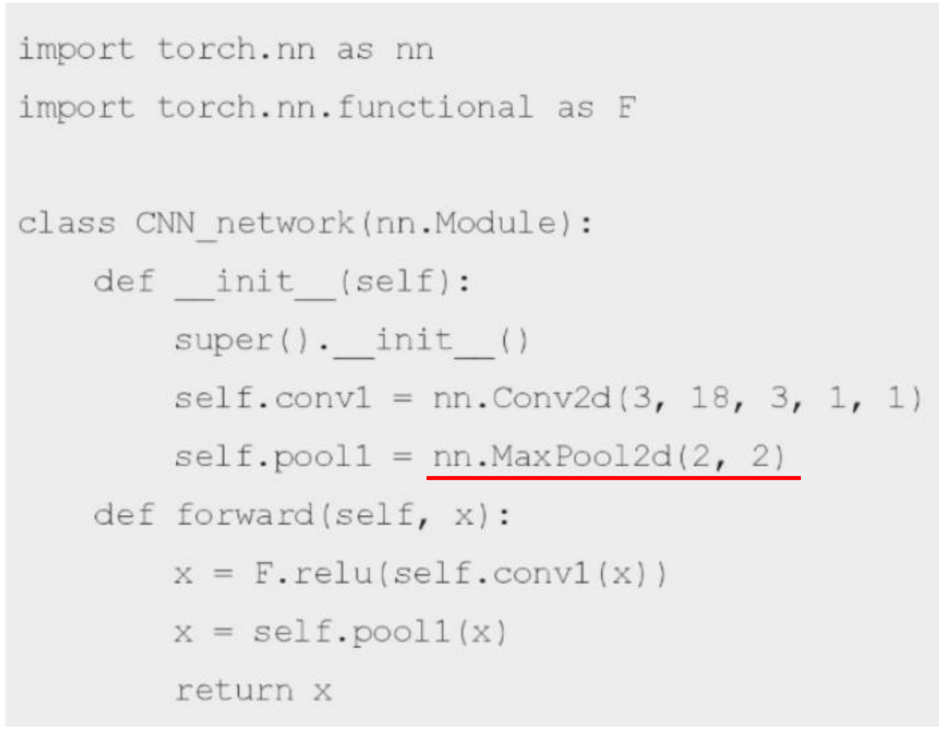
Pooling

```
torch.nn.MaxPool2d(
    kernel_size,     # 池化窗格大小
    stride,          # 步幅，預設：kernel_size
)
```

```
torch.nn.AvgPool2d(
    kernel_size,     # 池化窗格大小
    stride,          # 步幅，預設：kernel_size
)
```

```
import torch.nn as nn
import torch.nn.functional as F

class CNN_network(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 18, 3, 1, 1)
        self.pool1 = nn.MaxPool2d(2, 2)
    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        return x
```

Pooling



```
import torch.nn as nn
import torch.nn.functional as F

class CNN_network(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 18, 3, 1, 1)
        self.pool1 = nn.MaxPool2d(2, 2)
    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        return x
```

- **`self.conv1 = nn.Conv2d(3, 18, 3, 1, 1)`** creates a convolutional layer named conv1 with 3 input channels, 18 output channels, a kernel size of 3×3, a stride of 1, and a padding of 1.
- **`self.pool1 = nn.MaxPool2d(2, 2)`** creates a max pooling layer named pool1 with a kernel size of 2×2 and a stride of 2.

  2. **`self.conv1 = nn.Conv2d(3, 18, 3, 1, 1)`**：定義了一個卷積層 **`conv1`**，它有3個輸入通道，產生18個輸出通道，卷積核大小為3×3，步幅為1，並且在卷積操作時使用了填充，填充的大小為1。
  3. **`self.pool1 = nn.MaxPool2d(2, 2)`**：定義了一個池化層 **`pool1`**，使用最大池化方法，池化核大小為2×2，步幅為2。

The forward method defines the forward pass of the network.

- **`x = F.relu(self.conv1(x))`** applies the convolution operation using conv1 on input x and then applies the ReLU activation function.
- **`x = self.pool1(x)`** applies max pooling using pool1 on the result of the convolution.
- Finally, it returns the output x.

  1. **`x = F.relu(self.conv1(x))`**：將輸入 x 通過卷積層 conv1 進行卷積操作，然後通過 relu 激活函數進行激活。
  2. **`x = self.pool1(x)`**：將經過 relu 激活後的輸出 x 通過池化層 pool1 進行最大池化操作。
  3. **`return x`**：返回經過卷積和池化後的特徵圖 x。

Fully Connected Layer

```
import torch.nn as nn
import torch.nn.functional as F

class CNN_network(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 18, 3, 1, 1)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.linear1 = nn.Linear(16*16*18, 64)
        self.linear2 = nn.Linear(64, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = x.view(-1, 16*16*18)
        x = F.relu(self.linear1(x))
        x = F.log_softmax(self.linear2(x), dim = 1)
        return x
```

- `` `self.linear1 = nn.Linear(16*16*18, 64)` `` defines a fully connected layer `` `linear1` `` with input size of 16×16×18 (which is inferred from the previous layer), and output size of 64.
- `` `self.linear2 = nn.Linear(64, 10)` `` defines another fully connected layer `` `linear2` `` with input size of 64 and output size of 10.

  `` `self.linear1 = nn.Linear(16*16*18, 64)` ``：定義了一個全連接層 `` `linear1` ``，將池化後的特徵圖展平為一維，並將其映射到64個神經元。
  `` `self.linear2 = nn.Linear(64, 10)` ``：定義了另一個全連接層 `` `linear2` ``，將上一層的64個神經元映射到最終輸出的10個神經元（假設執行的是一個分類任務）。

- `` `x = x.view(-1, 16*16*18)` `` reshapes the tensor `` `x` `` to fit into the fully connected layer.
- `` `x = F.relu(self.linear1(x))` `` applies a ReLU activation to the output of the first linear layer.
- `` `x = F.log_softmax(self.linear2(x), dim=1)` `` applies a softmax activation along the second linear layer's output to get the final output probabilities.

  `` `x = x.view(-1, 16*16*18)` ``：將池化後的特徵圖 x 展平成一維，以便餵入全連接層。
  `` `x = F.relu(self.linear1(x))` ``：將展平後的特徵向量 x 通過全連接層 linear1，然後通過 relu 激活函數進行激活。
  `` `x = F.log_softmax(self.linear2(x), dim=1)` ``：將經過 linear1 的結果 x 通過全連接層 linear2，然後使用 log_softmax 函數計算輸出，dim=1 表示計算輸出的維度在第一軸上（通常用於多類別分類問題）。

Dropout

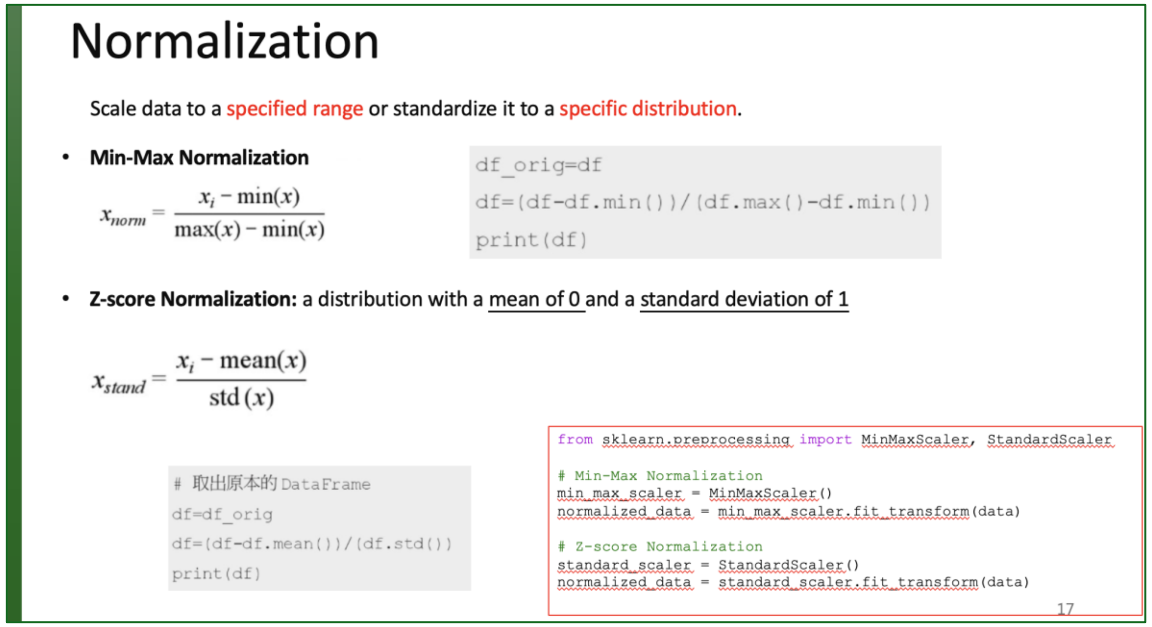
- Dropout is a regularization technique used during training to prevent overfitting.
- Dropout randomly sets a fraction of input units to zero during each update pass. This means that the information flow is temporarily removed from some neurons, forcing the network to learn more robust features.

```
torch.nn.Dropout(
    p=0.5,      # 丟棄機率
)
```

```
import torch.nn as nn
import torch.nn.functional as F

class CNN_network(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 18, 3, 1, 1)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.dropout1 = nn.Dropout(0.4) drop  40% neurons
        self.linear1 = nn.Linear(16*16*18, 64)
        self.linear2 = nn.Linear(64, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = x.view(-1, 16*16*18)
        x = self.dropout1(x)
        x = F.relu(self.linear1(x))
        x = F.log_softmax(self.linear2(x), dim = 1)
        return x
```

BatchNorm/批次標準化

- Batch Normalization (BatchNorm) is a technique used to normalize the inputs of each layer.
- Batch Normalization normalizes the inputs of each mini-batch, aiming to make the mean of the inputs close to zero and the standard deviation close to one.
- This helps prevent some input data from being too large or too small, thereby alleviating the problem of gradient vanishing and making neural networks easier to train.



Normalization

Scale data to a specified range or standardize it to a specific distribution.

- **Min-Max Normalization**

$$x_{norm} = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

```
df_orig=df
df=(df-df.min())/(df.max()-df.min())
print(df)
```

- **Z-score Normalization:** a distribution with a mean of 0 and a standard deviation of 1

$$x_{stand} = \frac{x_i - \text{mean}(x)}{\text{std}\,(x)}$$

```
# 取出原本的DataFrame
df=df_orig
df=(df-df.mean())/(df.std())
print(df)
```

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Min-Max Normalization
min_max_scaler = MinMaxScaler()
normalized_data = min_max_scaler.fit_transform(data)

# Z-score Normalization
standard_scaler = StandardScaler()
normalized_data = standard_scaler.fit_transform(data)
```

17

BatchNorm/批次標準化

```
nn.BatchNorm2d(
    num_features,  # 通道數
)
```

```
self.batch1 = nn.BatchNorm2d(18)
```

此指令定義了有 18 個通道數的 BatchNorm 層。

Multiple channels of feature maps are typically generated, with each channel corresponding to different features.
When applying Batch Normalization, the input of each channel is normalized independently.

18 feature maps ➔ 18 channels of BatchNorm

```
import torch.nn as nn
import torch.nn.functional as F

class CNN_network(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 18, 3, 1, 1)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.dropout1 = nn.Dropout(0.4)
        self.batch1 = nn.BatchNorm2d(18)
        self.linear1 = nn.Linear(16*16*18, 64)
        self.linear2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.batch1(x)  # BatchNorm層需放在relu()之前
        x = F.relu(x)
        x = self.pool1(x)
        x = self.dropout(x)
        x = x.view(-1, 16*16*18)
        x = F.relu(self.linear1(x))
        x = F.log_softmax(self.linear2(x), dim = 1)
        return x
```

Exercise: MNIST

1. Create a MNIST prediction model without convolutional computation and compare it with the CNN version.
2. Improve the accuracy of the CNN version

Submission requirements:

1. source code (MNIST_noCNN.py, MNIST_CNN.py) of (1) & (2)
2. PDF documents

   Explaining your strategy of (2).

   Show the outputs
3. Upload to e-learning before 4/12 14:10