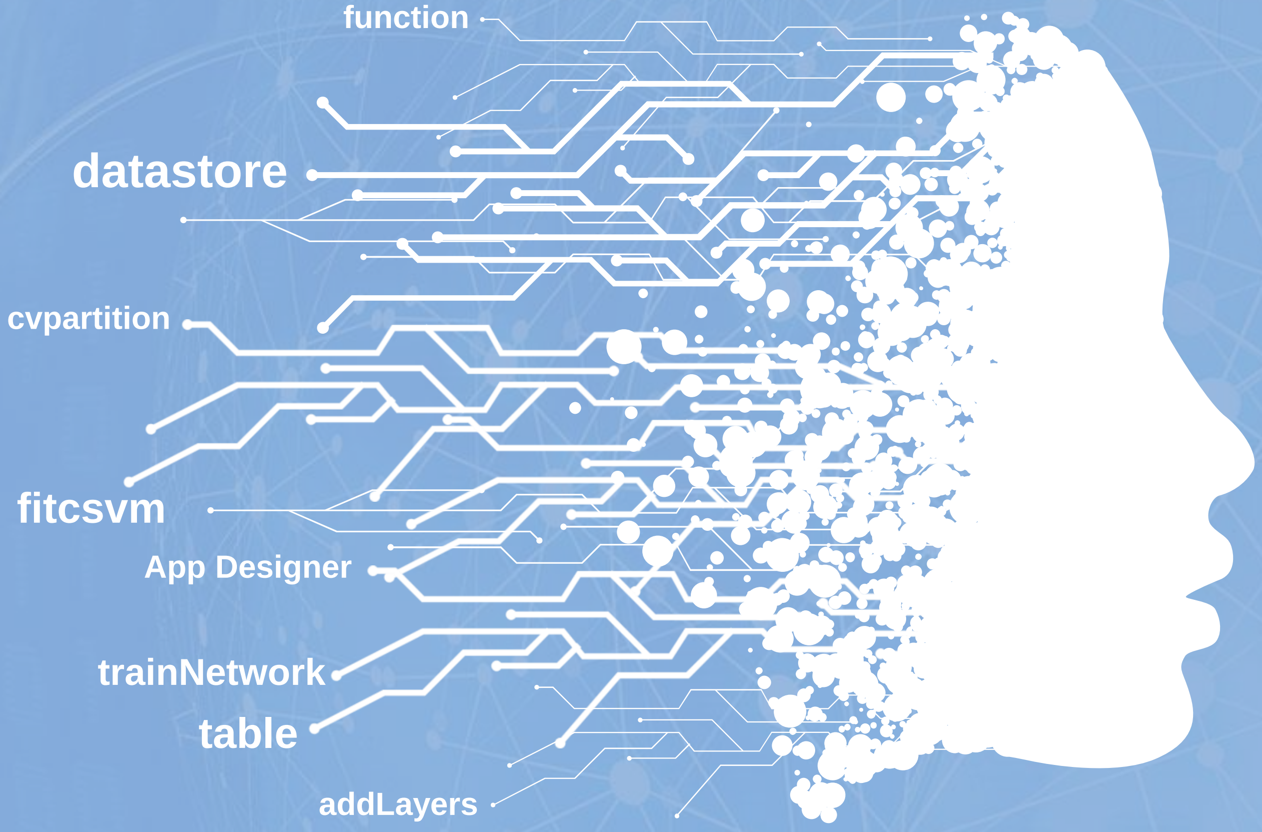


Deep Learning

MATLAB進階程式語言與實作

盧家鋒 Chia-Feng Lu, Ph.D.
Department of Biomedical Imaging and Radiological Sciences, NYCU
alvin4016@nycu.edu.tw

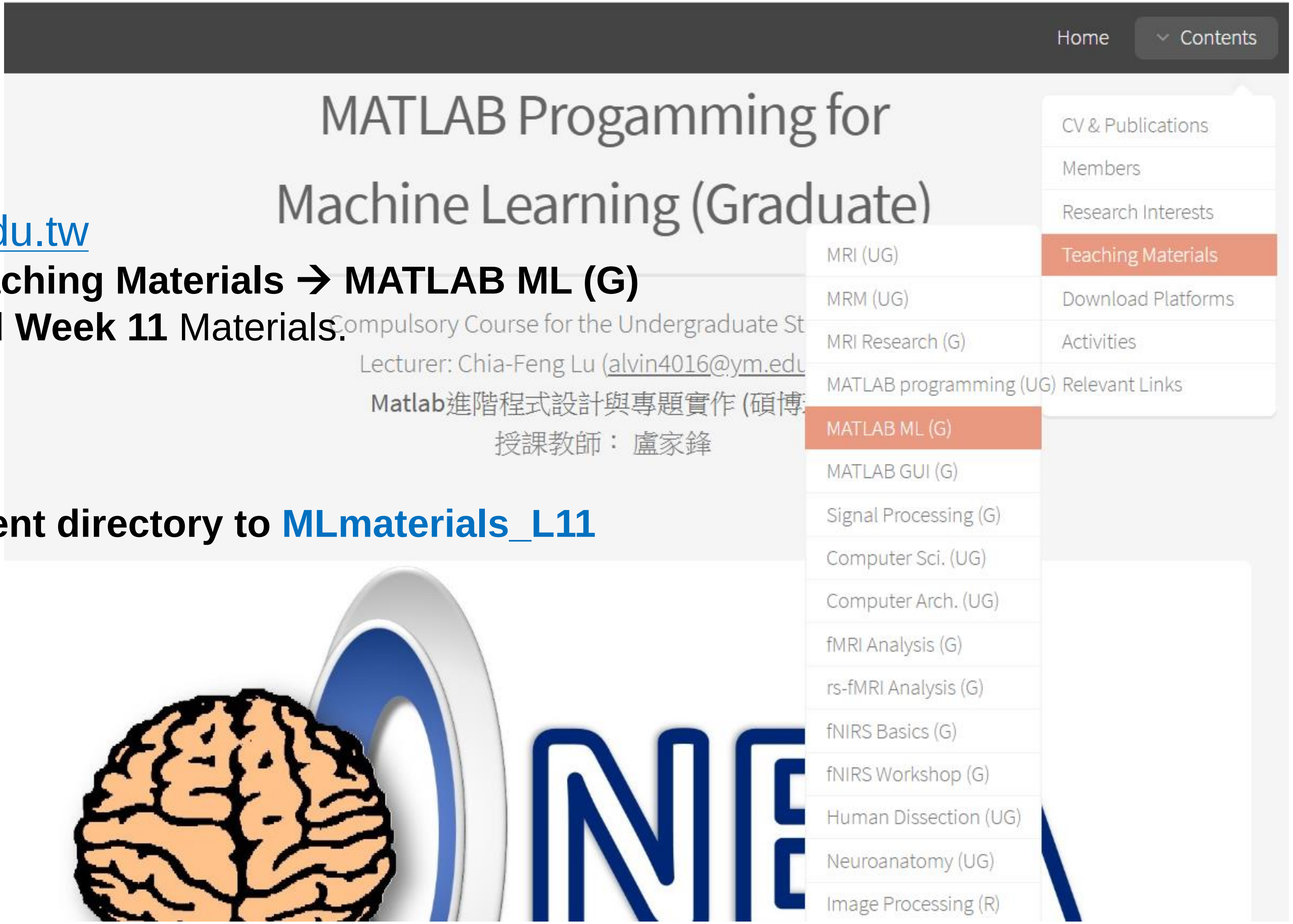
Teaching Materials

cflu.lab.nycu.edu.tw

Contents → Teaching Materials → MATLAB ML (G)

Please download **Week 11** Materials.

Please set current directory to MLmaterials_L11



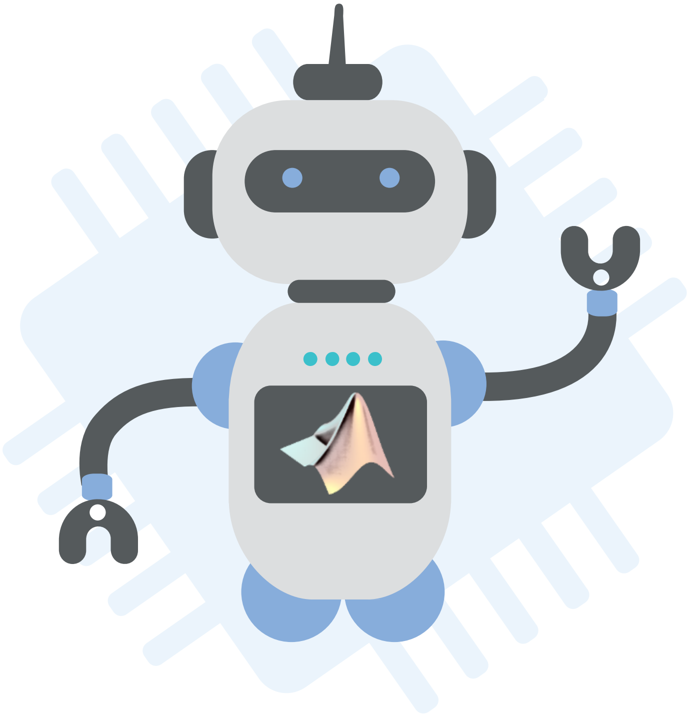
Contents in this Week

01 Deep Learning

ReLU, softmax function, dropout

02 Convolutional Neural Network

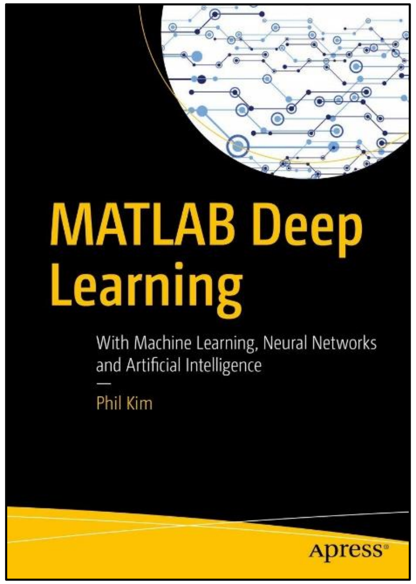
Convolution layer, pooling layer



References

[Textbook 5]

- **MATLAB Deep Learning With Machine Learning, Neural Networks and Artificial Intelligence, 1st edition, 2017**
  Phil Kim
- **Deep Learning (Ch.5)**
- **Convolutional Neural Network (Ch.6)**



Deep Learning

ReLU, softmax function, dropout

Neural Network: History

- **The deep neural network is the multi-layer neural network** that contains two or more hidden layers.

- Multi-layer neural network took 30 years to solve the problems of **learning rule** of the single-layer neural network, which was eventually solved by the **back-propagation algorithm**.

- The backpropagation training with the **additional hidden layers often resulted in poorer performance**. Deep Learning provided a solution to this problem.

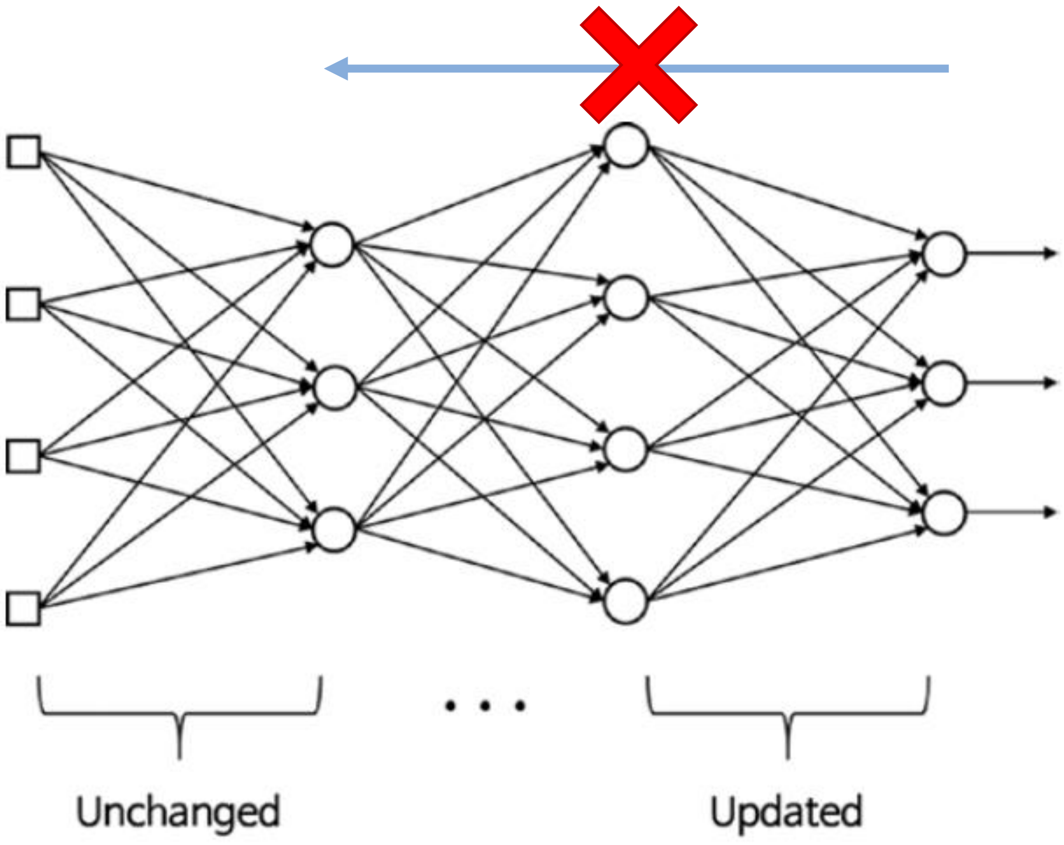
Improvement of Deep Neural Network

- The neural network with deeper layers yielded poorer performance was that the network was not properly trained.

- The backpropagation algorithm experiences three difficulties in training deep neural network：
  - **Vanishing gradient**
  - **Overfitting**
  - **Computational load**

Vanishing Gradient

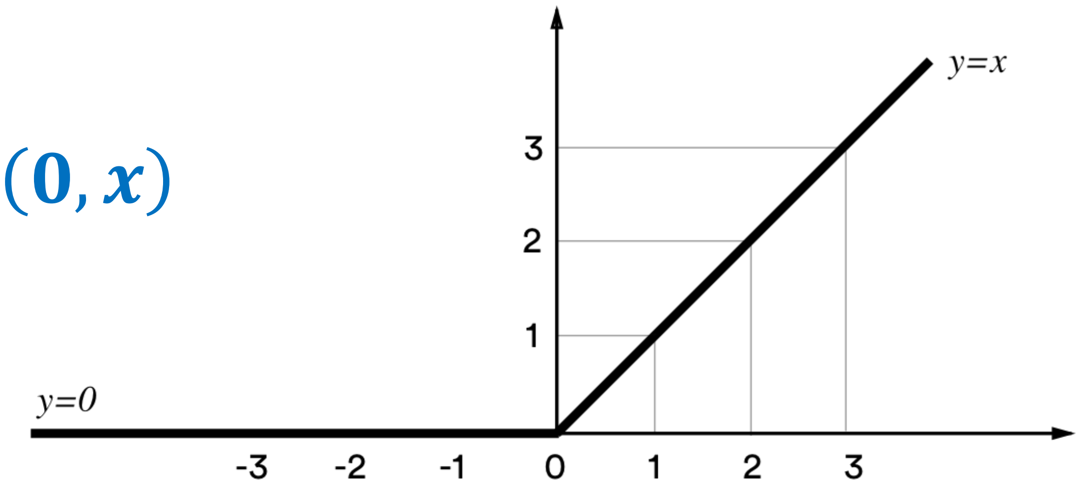
- The vanishing gradient in the training process occurs when the output error is more likely to **fail to reach the farther nodes**.
- As the error hardly reaches the first hidden layer, **the weight cannot be adjusted**.



Vanishing Gradient: ReLU

- A solution to the vanishing gradient is using the **Re**ctified **L**inear **U**nit (**ReLU**) function as the activation function.

$$\varphi(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} = max(0, x)$$



- The sigmoid function limits the node's outputs to the unity, the ReLU function does not exert such limits and better transmit the error than the sigmoid function.

Derivative of ReLU Function

$$\varphi(x) = \begin{cases} x, & x > 0 \\ 0, & x \le 0 \end{cases} = max(0, x)$$

$$\varphi'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \le 0 \end{cases}$$

Generalized delta rule:
(weight update)

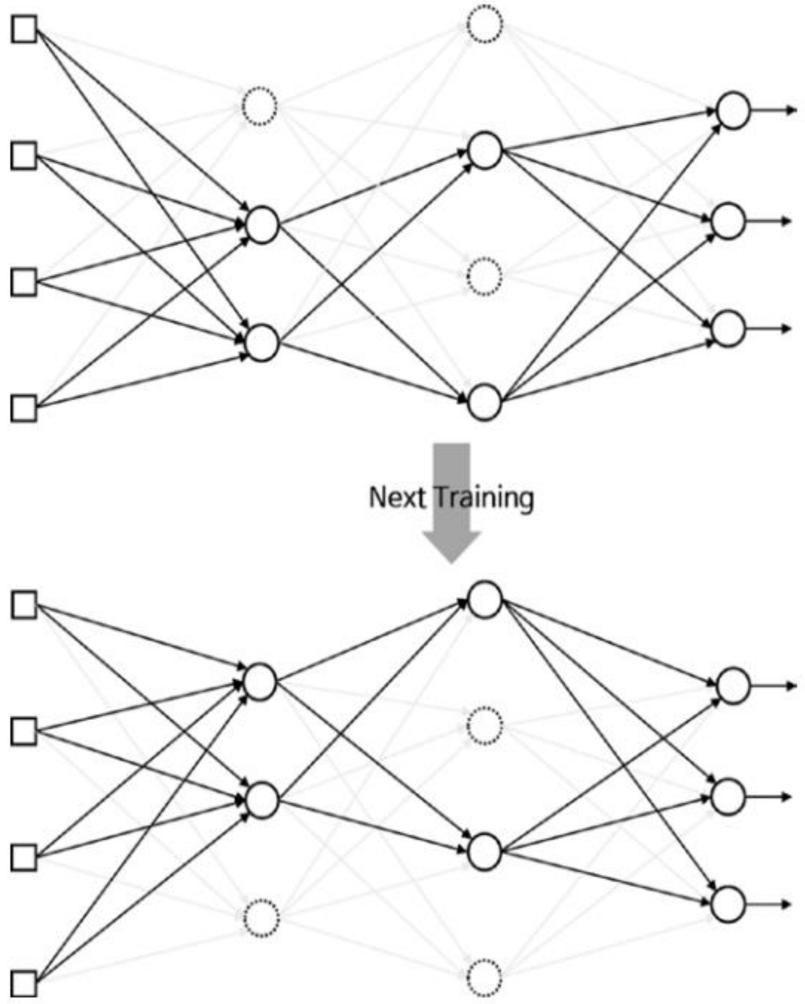
$$w_{ij} \leftarrow w_{ij} + \alpha\delta_i x_j$$

$$\delta_i = \varphi'(v_i)e_i$$

Overfitting: Dropout

- The reason that the deep neural network is especially vulnerable to overfitting is that the model becomes **more complicated as it includes more hidden layers**, and hence more weight.

- Dropout: **train only some of the randomly selected nodes**
- Use massive training data is very helpful to reduce potential bias.



Dropout

- ym contains zeros for as many elements as the ratio.
- ym contains **1 / (1 - ratio)** for the other elements to compensate for the loss of output due to the dropped elements.

$$y_1 = \begin{bmatrix} 0.5356 \\ 0.9537 \\ 0.5442 \\ 0.0821 \\ 0.3663 \\ 0.8509 \end{bmatrix} \qquad ym = \begin{bmatrix} 2 \\ 2 \\ 0 \\ 0 \\ 0 \\ 2 \end{bmatrix}$$

Ex: ratio = 0.5, 1/(1-ratio) = 2

```
function ym = Dropout(y, ratio)
  [m, n] = size(y);
  ym     = zeros(m, n);

  num     = round(m*n*(1-ratio));
  idx     = randperm(m*n, num);
  ym(idx) = 1 / (1-ratio);
end
```

$$y_1 * ym = \begin{bmatrix} 1.0712 \\ 1.9075 \\ 0 \\ 0 \\ 0 \\ 1.7017 \end{bmatrix}$$

Computational Load

- The number of weights increases geometrically with the number of hidden layers, thus requiring more training data. This ultimately requires more calculations to be made.

- This trouble has been relieved to a considerable extent by the introduction of high-performance hardware, such as **GPU**, and algorithms, such as **batch normalization**.

	Sample 1	**Sample 2**	**Sample 3**	**Sample 4**
$\mathbf{x_1}$	0.68	0.80	0.57	0.34
$\mathbf{x_2}$	238.4	128.9	451.6	781.2

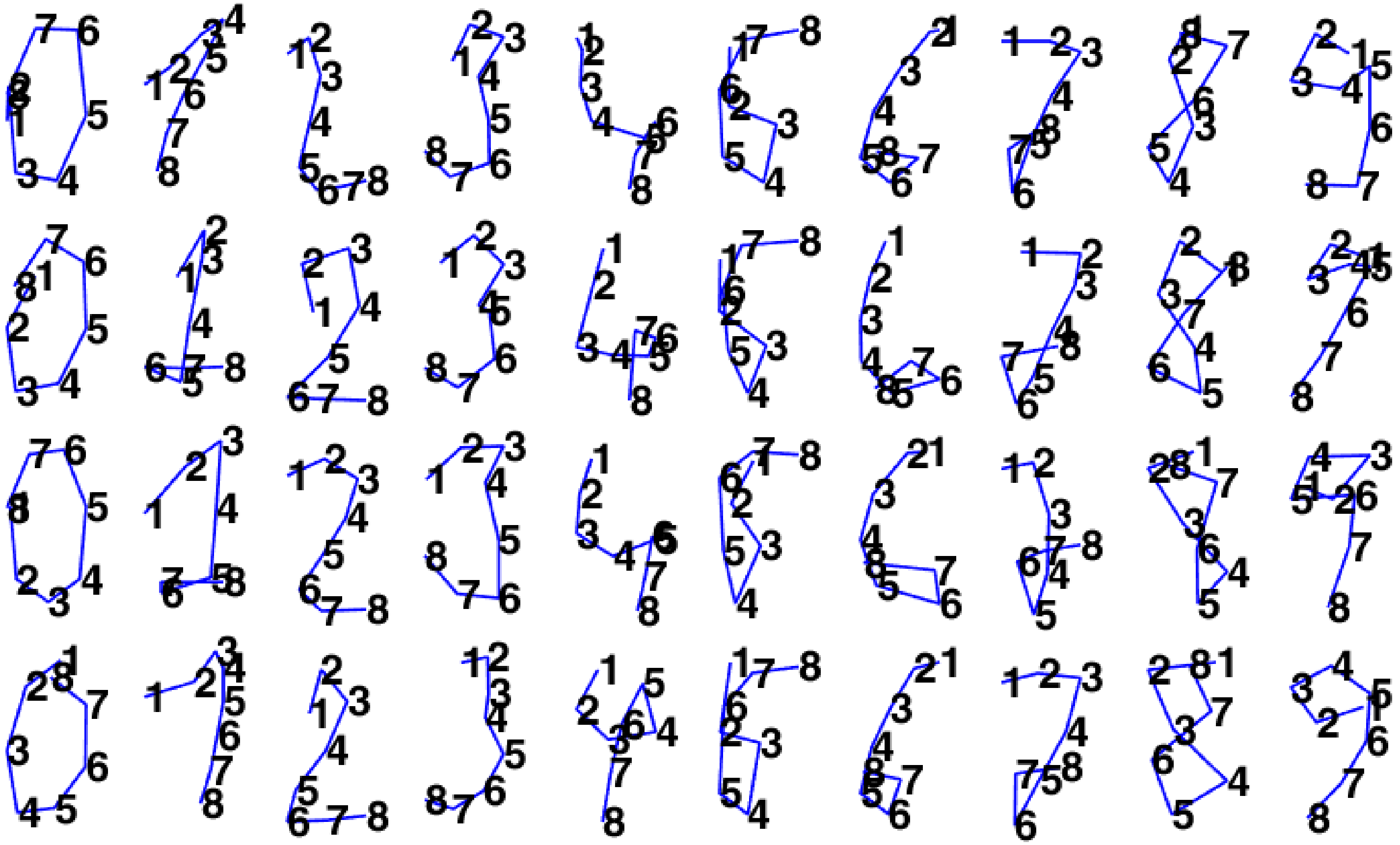
Exercise:
Pen-Based Recognition of Handwritten Digits

- **E. Alpaydin, Fevzi. Alimoglu (July 1998)**

MLmaterials_L11\DeepLearning\pendigits.names, pendigits.tra, pendigits.tes

- A digit database by collecting 250 samples from 44 writers. The samples written by 30 writers are used for training (7494 samples), and the digits written by the other 14 are used for writer independent testing (3498 samples).

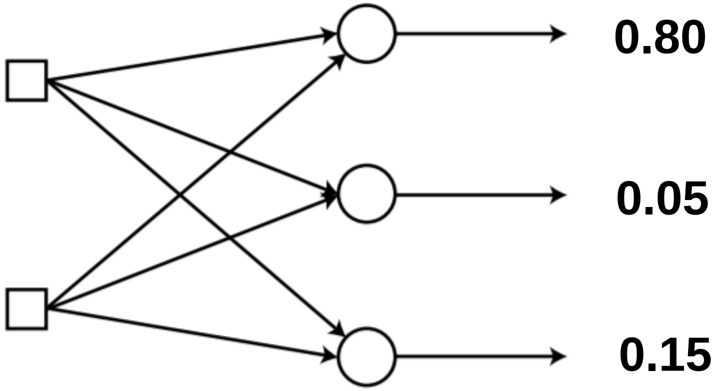
- The (x, y) coordinate information was recorded. The input vector size is 2*8, two times the number of 8 resampled points for each digit.



Multi-Class Classification

- One-hot encoding

$\{5, 7, \boldsymbol{Class\ 1}\}$	⇨	$\{5, 7, \mathbf{1, 0, 0}\}$
$\{9, 8, \boldsymbol{Class\ 3}\}$		$\{9, 8, \mathbf{0, 0, 1}\}$
$\{2, 4, \boldsymbol{Class\ 2}\}$		$\{2, 4, \mathbf{0, 1, 0}\}$
$\cdots$		$\cdots$
$\{6, 5, \boldsymbol{Class\ 3}\}$		$\{6, 5, \mathbf{0, 0, 1}\}$

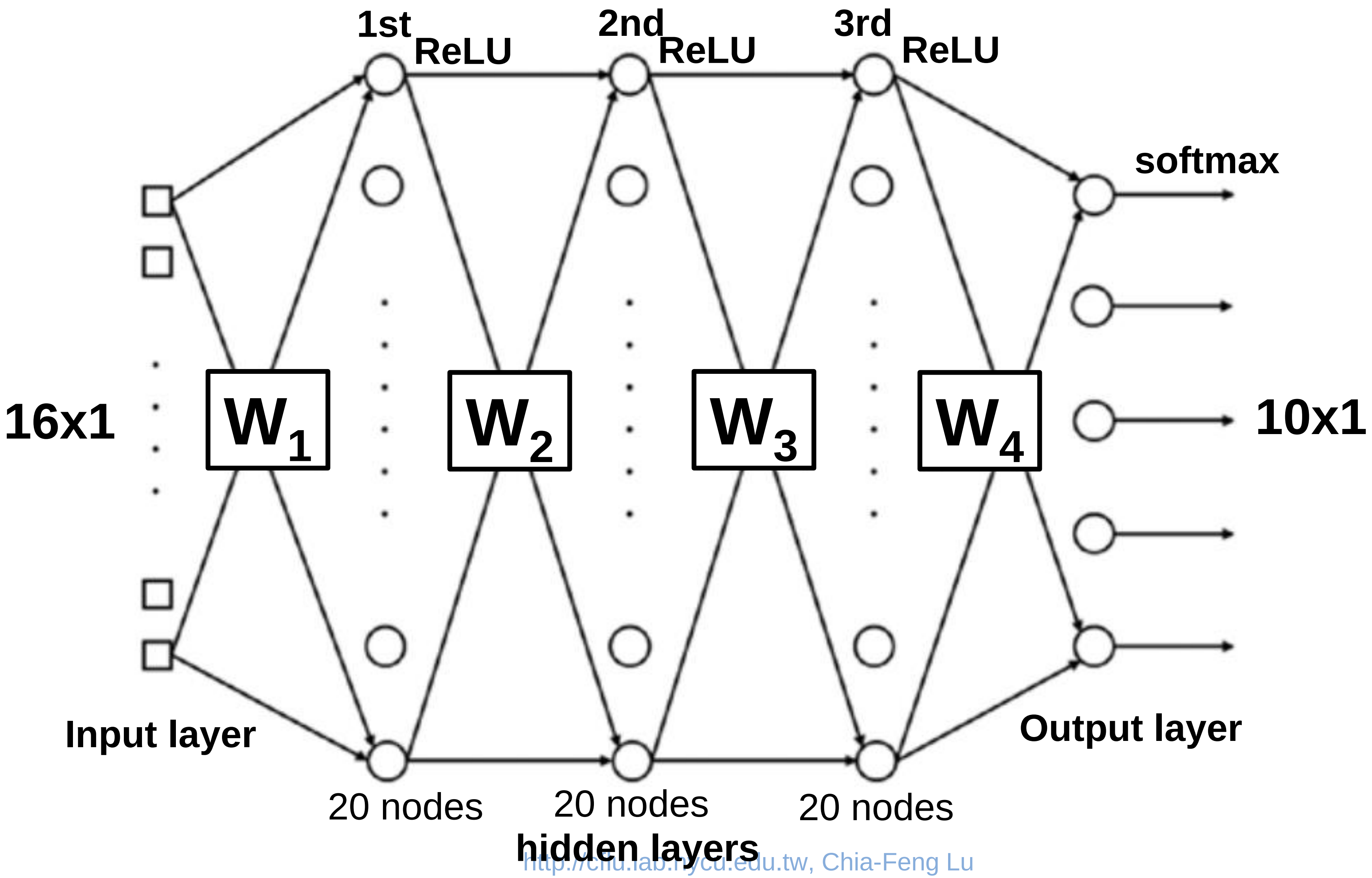


- Softmax function as the output activation function

$$y_i = \varphi(v_i) = \frac{e^{v_i}}{e^{v_1} + e^{v_2} + e^{v_3} + \cdots + e^{v_M}} = \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}}$$

v_i is the weighted sum of the *i*-th output node.
M is the number of output nodes.
$\varphi(v_1) + \varphi(v_2) + \varphi(v_3) + \cdots + \varphi(v_M) = 1$

Deep Learning Architecture



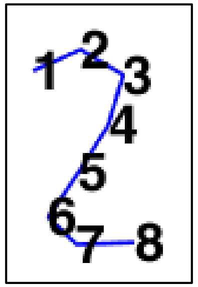
Exercise:
Pen-Based Recognition of Handwritten Digits

- **MLmaterials_L11\DeepLearning**
  - **DL_pendigits.m**
  - **ReLU.m**
  - **Softmax.m**

- load('pendigits.tra')

Pendigits: 7494 x 17 double

(2*8 coordinates)+ 1 response



Test dataset

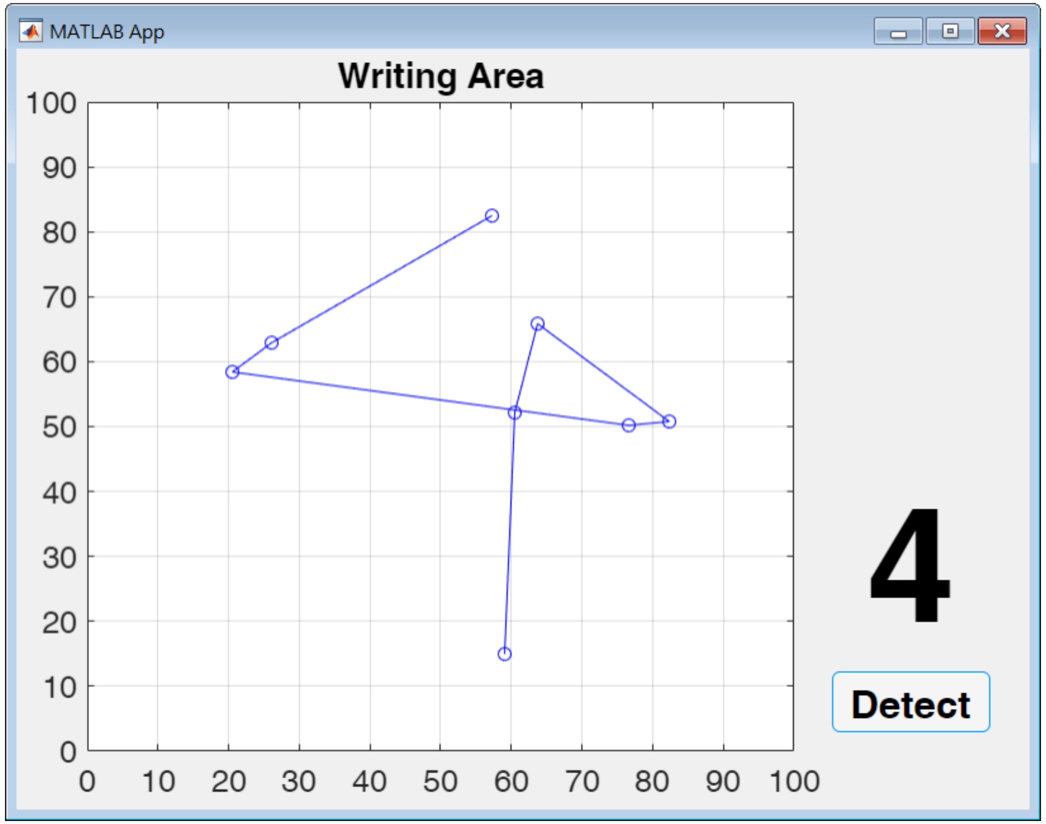
Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	**355** 10.1%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	100% 0.0%
2	**0** 0.0%	**355** 10.1%	**2** 0.1%	**1** 0.0%	**2** 0.1%	**3** 0.1%	**1** 0.0%	**8** 0.2%	**0** 0.0%	**2** 0.1%	94.9% 5.1%
3	**0** 0.0%	**2** 0.1%	**360** 10.3%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**1** 0.0%	**0** 0.0%	**0** 0.0%	99.2% 0.8%
4	**0** 0.0%	**0** 0.0%	**0** 0.0%	**333** 9.5%	**0** 0.0%	**6** 0.2%	**0** 0.0%	**4** 0.1%	**0** 0.0%	**1** 0.0%	96.8% 3.2%
5	**1** 0.0%	**1** 0.0%	**2** 0.1%	**0** 0.0%	**352** 10.1%	**0** 0.0%	**0** 0.0%	**9** 0.3%	**0** 0.0%	**2** 0.1%	95.9% 4.1%
6	**0** 0.0%	**3** 0.1%	**0** 0.0%	**0** 0.0%	**5** 0.1%	**322** 9.2%	**3** 0.1%	**0** 0.0%	**1** 0.0%	**0** 0.0%	96.4% 3.6%
7	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**3** 0.1%	**0** 0.0%	**332** 9.5%	**0** 0.0%	**0** 0.0%	**0** 0.0%	99.1% 0.9%
8	**0** 0.0%	**3** 0.1%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**1** 0.0%	**0** 0.0%	**339** 9.7%	**0** 0.0%	**3** 0.1%	98.0% 2.0%
9	**7** 0.2%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**0** 0.0%	**3** 0.1%	**334** 9.5%	**1** 0.0%	96.8% 3.2%
10	**0** 0.0%	**0** 0.0%	**0** 0.0%	**2** 0.1%	**2** 0.1%	**3** 0.1%	**0** 0.0%	**0** 0.0%	**1** 0.0%	**327** 9.3%	97.6% 2.4%
	97.8% 2.2%	97.5% 2.5%	98.9% 1.1%	99.1% 0.9%	96.7% 3.3%	96.1% 3.9%	98.8% 1.2%	93.1% 6.9%	99.4% 0.6%	97.3% 2.7%	**97.5%** **2.5%**

Exercise:
Pen-Based Recognition of Handwritten Digits

- Try the Recognition App
  - **MLmaterials_L11\DeepLearning\DL_pendigits_App.mlapp**

Test on your own handwritten digits .

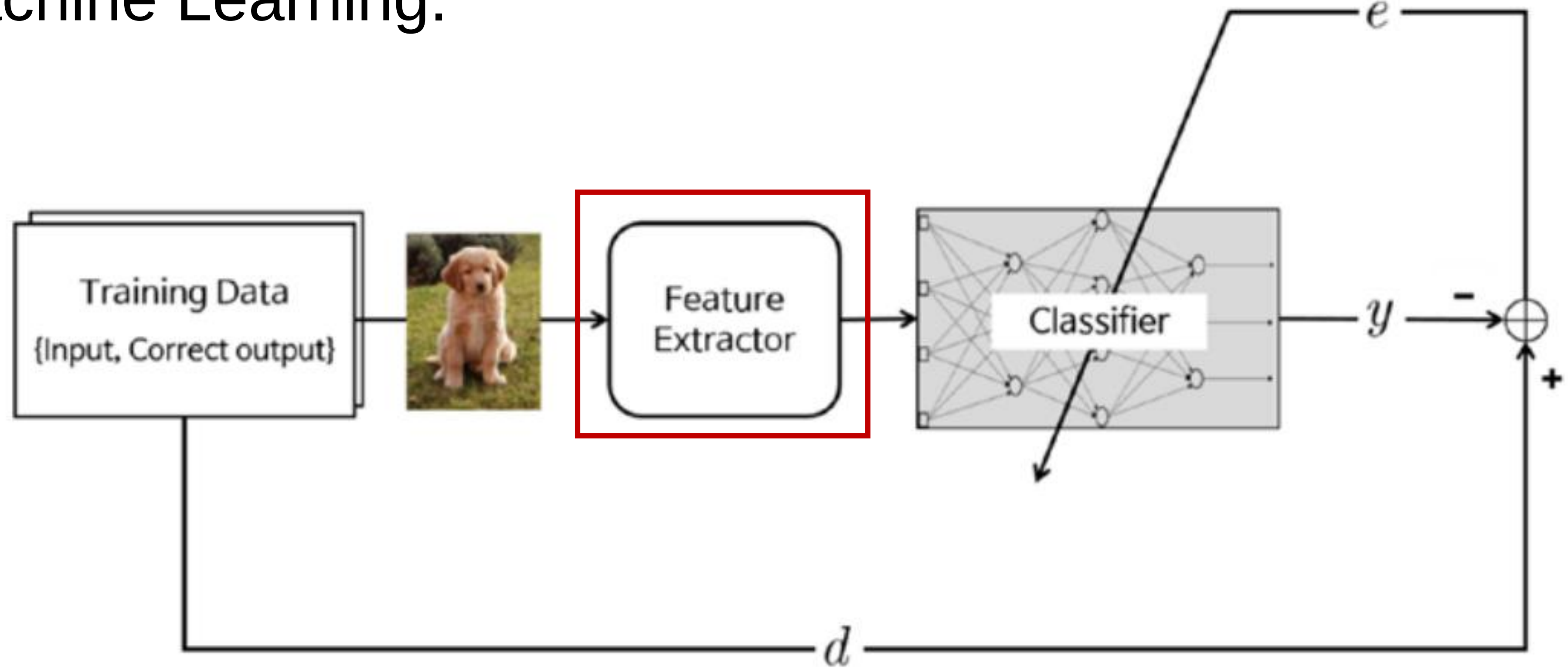


2 Convolutional Neural Network

Convolution layer, pooling layer

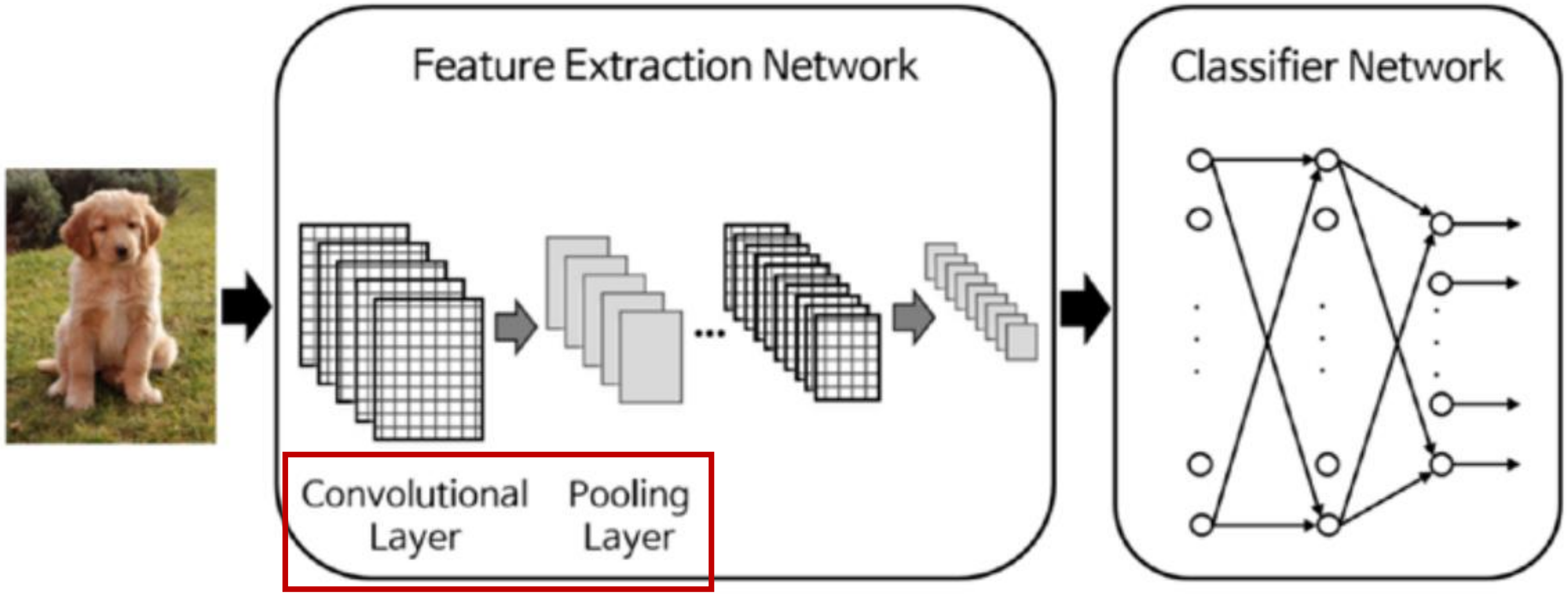
Convolutional Neural Network (ConvNet/CNN)

- Before ConvNet (or CNN), the feature extractor has been designed by experts of specific areas, and was independent of Machine Learning.



Convolutional Neural Network (ConvNet/CNN)

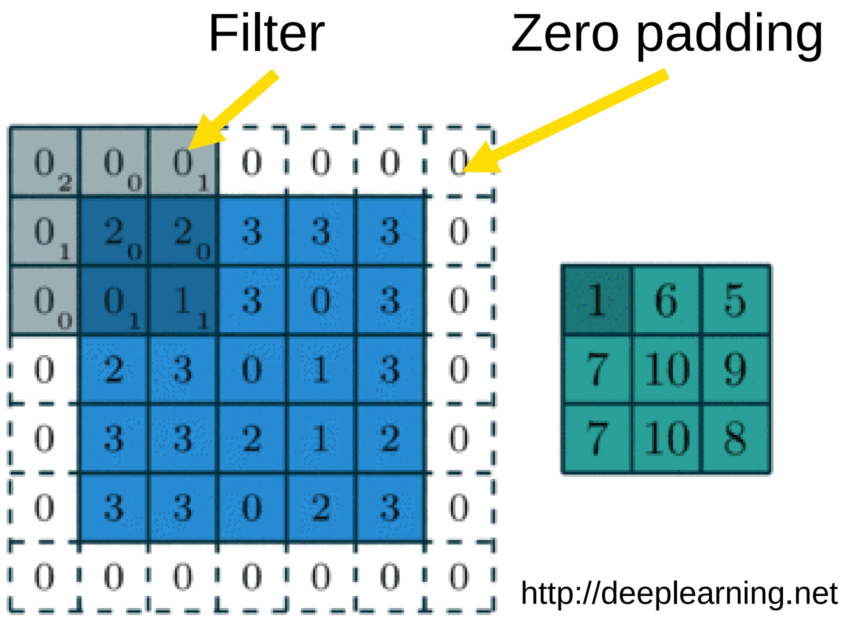
- ConvNet (or CNN) includes the feature extractor in the training process rather than designing it manually.



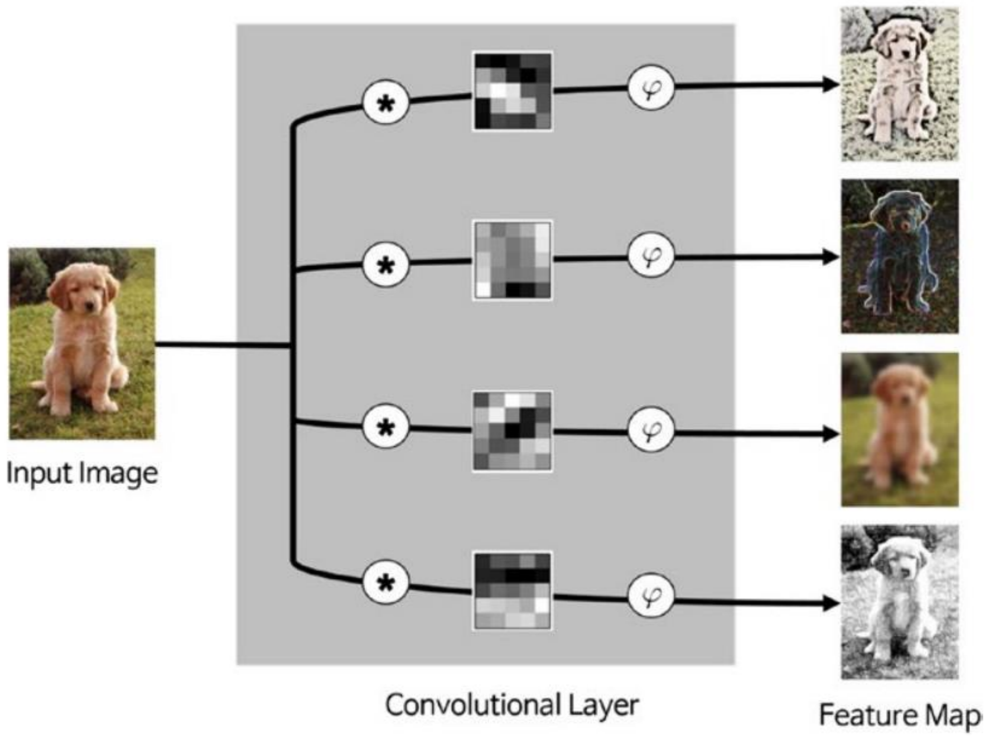
Convolution Layer

Convolution: https://cflu.lab.nycu.edu.tw/CFLu_course_matlabimage.html, Week 4

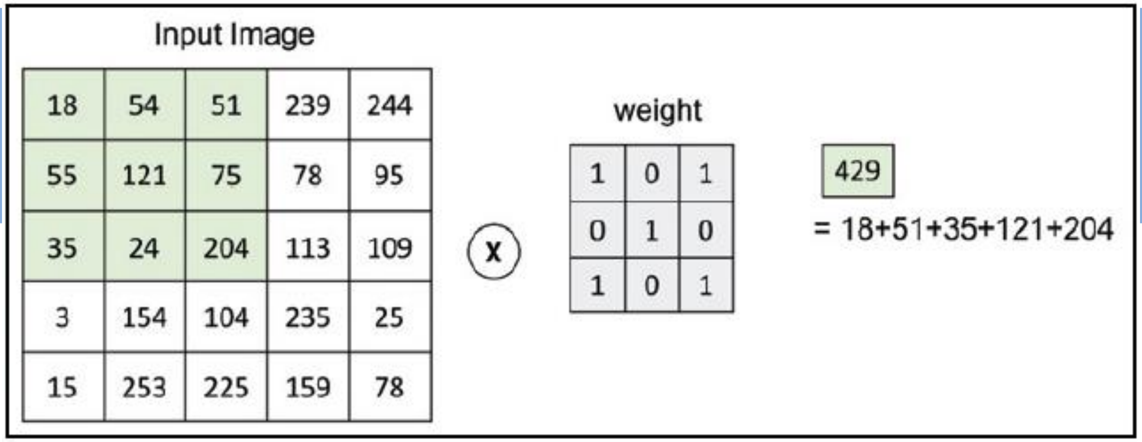
- Kernel size: filter size(F)
- Stride: sliding length of filter per step(S)
- Padding: control the output feature maps' size

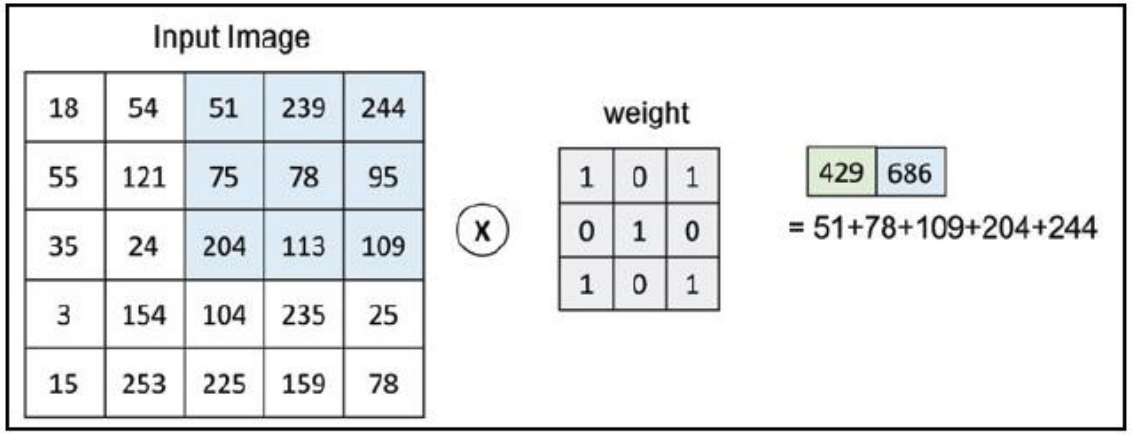


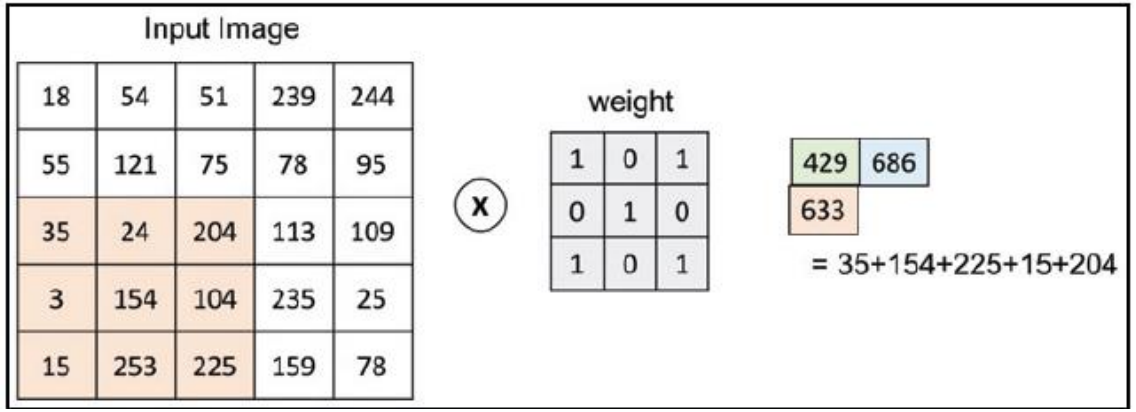
http://deeplearning.net

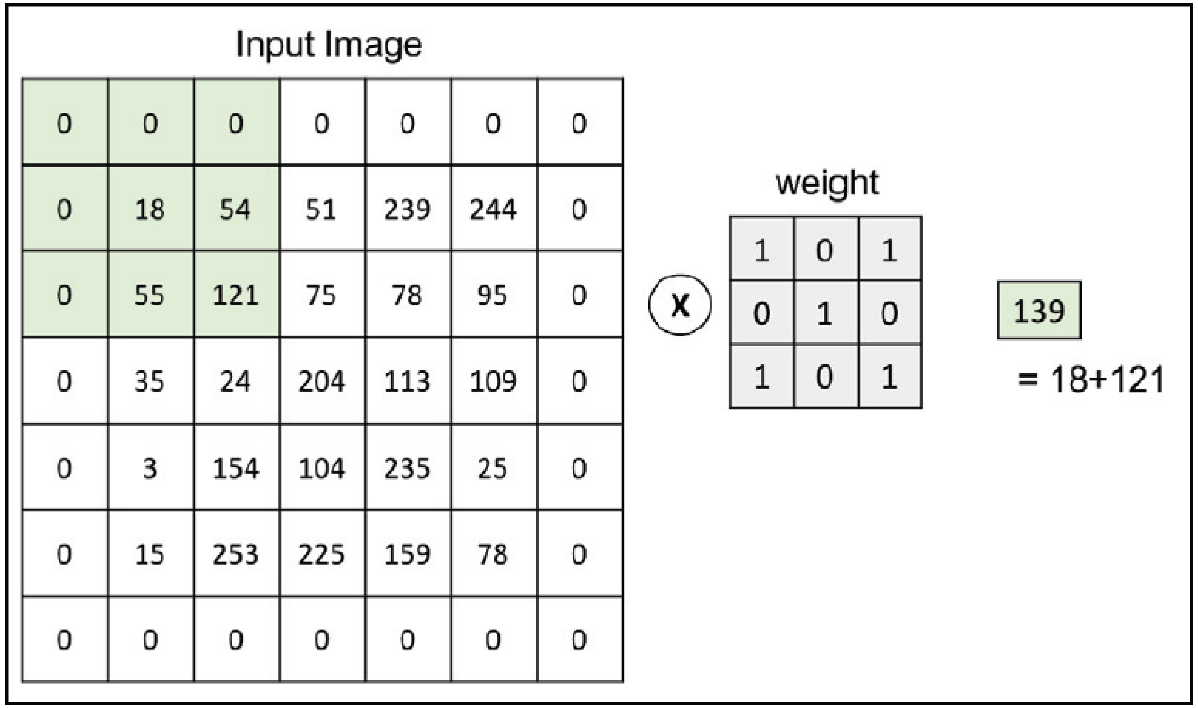


Convolution





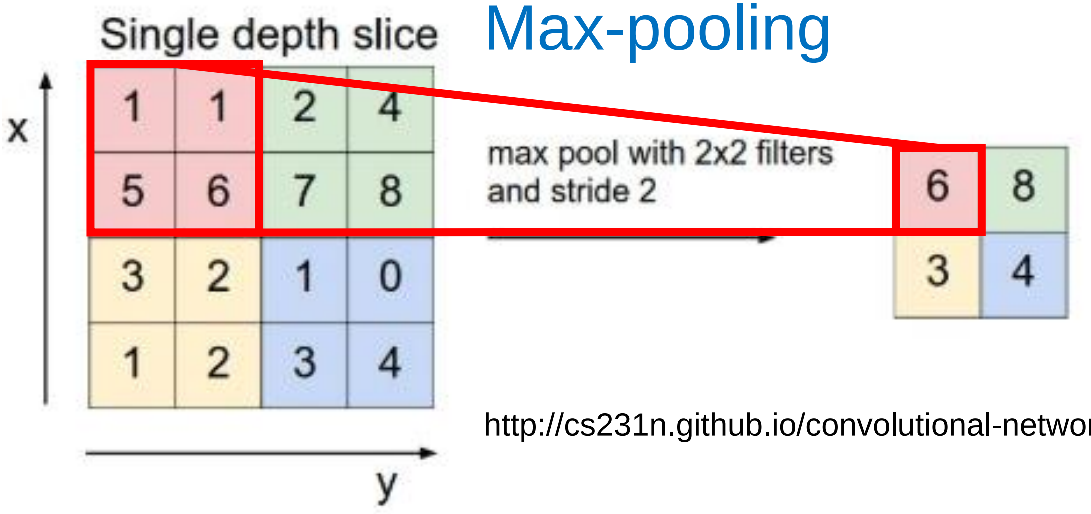




Pooling Layer

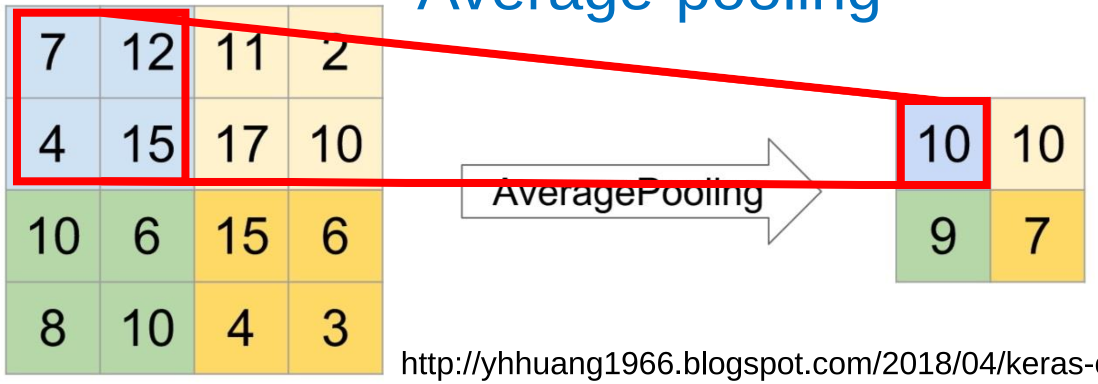
- Kernel size: pooling kernel size
- Stride: usually equal to kernel size
- Padding: control the output feature maps' size

Max-pooling



http://cs231n.github.io/convolutional-networks/

Average-pooling



http://yhhuang1966.blogspot.com/2018/04/keras-cnn.html

Pooling Layer

- The pooling layer compensates for eccentric and tilted objects to some extent. For example, the pooling layer can improve the recognition of a cat, which may be off-center in the input image.
- As the pooling process reduces the image size, it is highly beneficial for relieving the computational load and preventing overfitting.

Exercise: MNIST database

- 28-by-28 pixel black-and-white images

MLmaterials_L11\CNN\MnistDatabase.mat

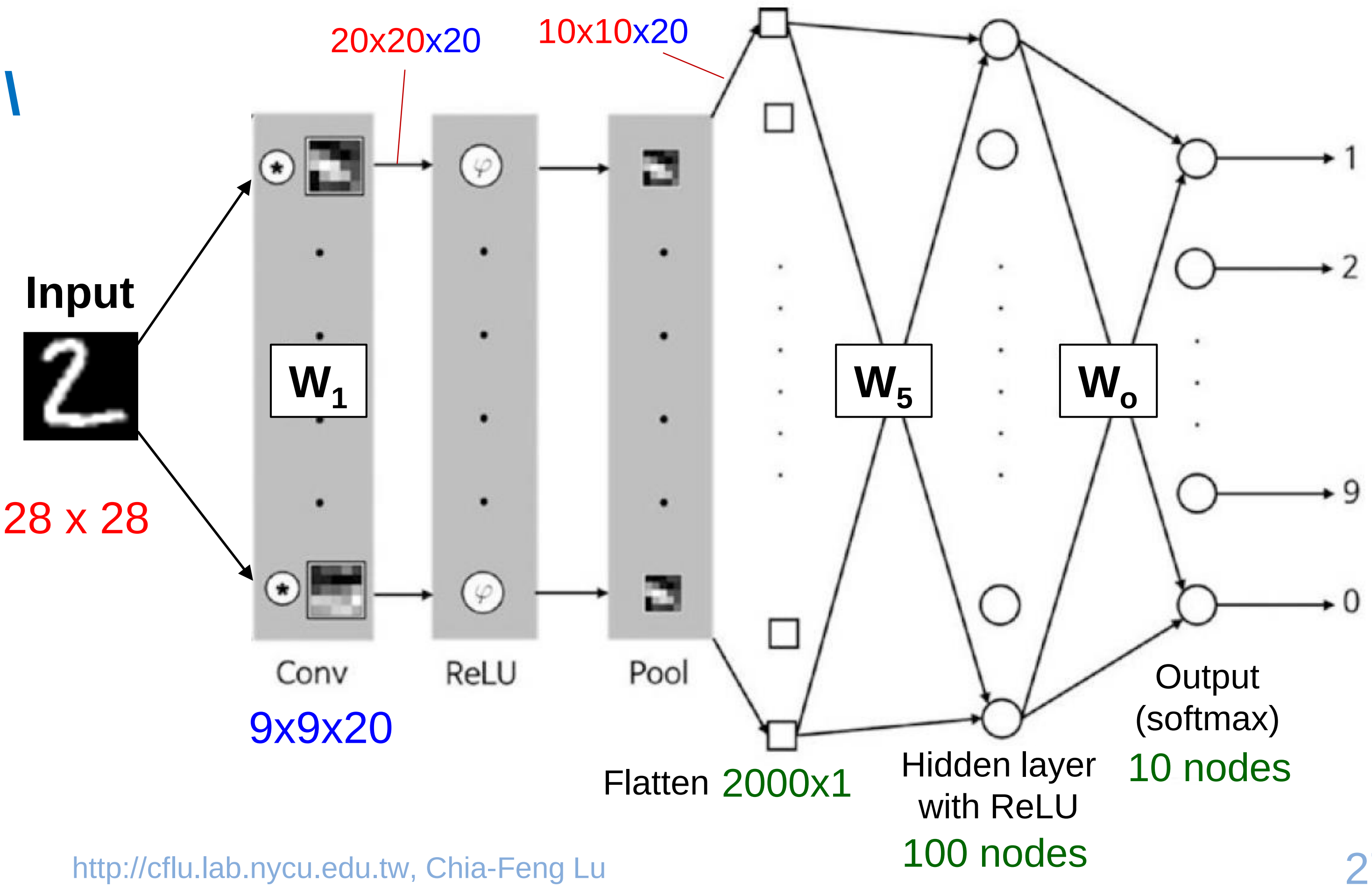
- 10,000 images of handwritten numbers. In general, 8,000 images are used for training, and the remaining 2,000 images are used for the validation test.

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

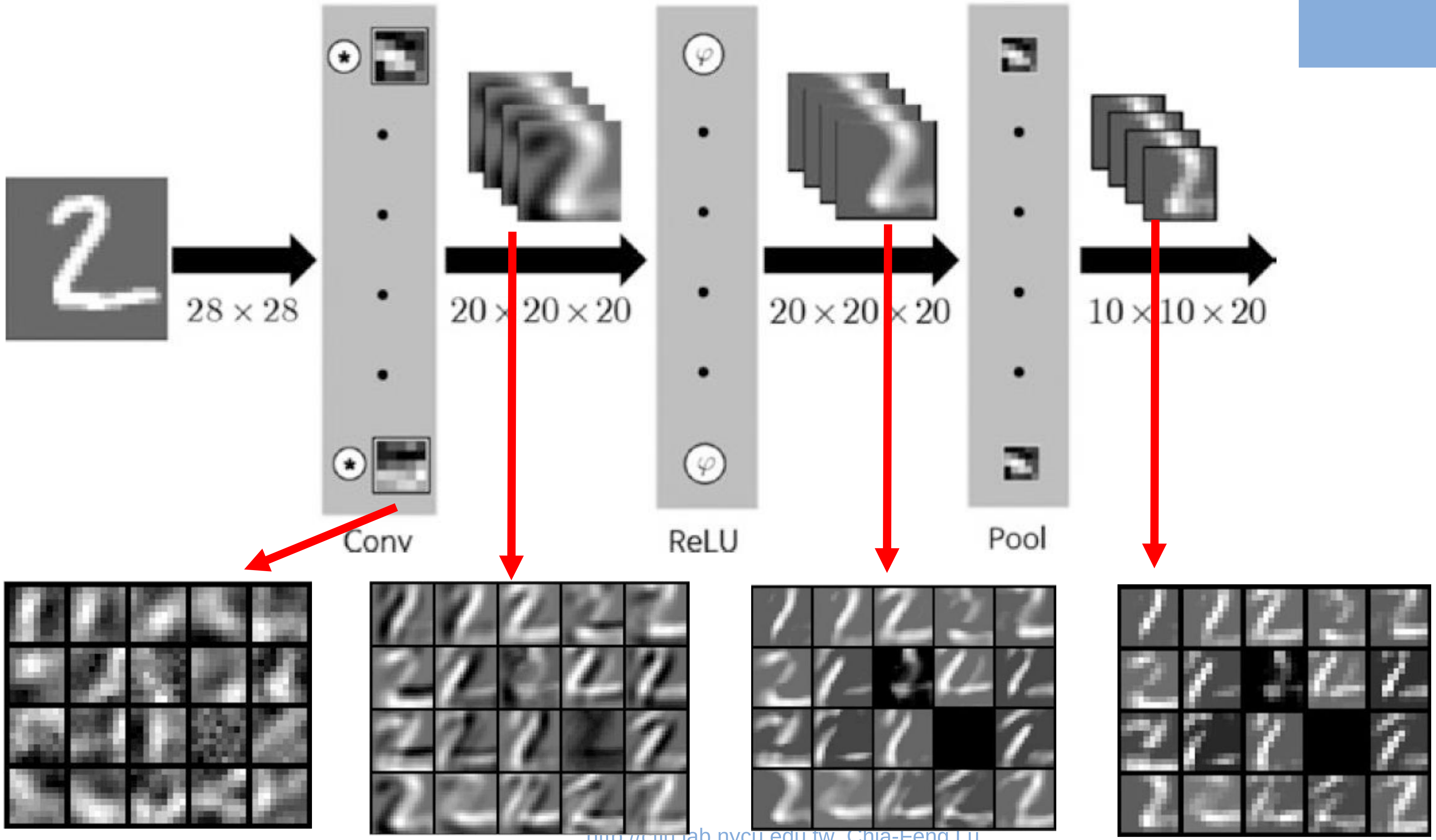
Exercise: MNIST database

- Architecture of convolutional neural network (CNN)

- **MLmaterials_L11\CNN**
  - **TestMnistConv.m**
  - **MnistConv.m**
  - **Conv.m**
  - **ReLU.m**
  - **Pool.m**
  - **Softmax.m**



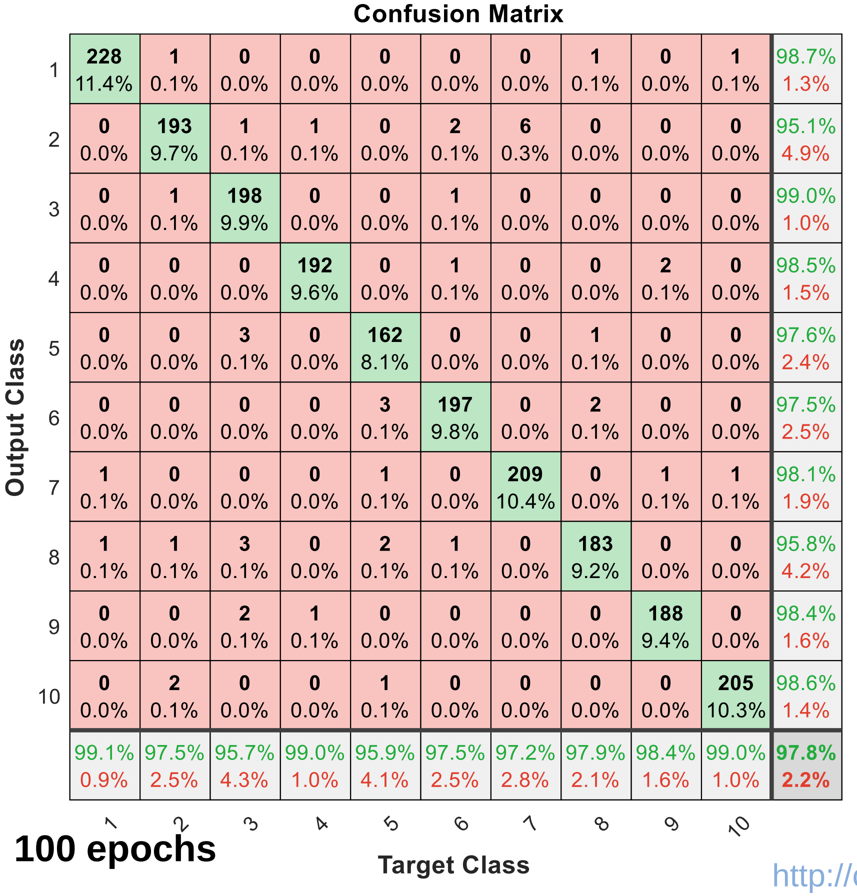
- **MLmaterials_L11\CNN\PlotFeatures.m**



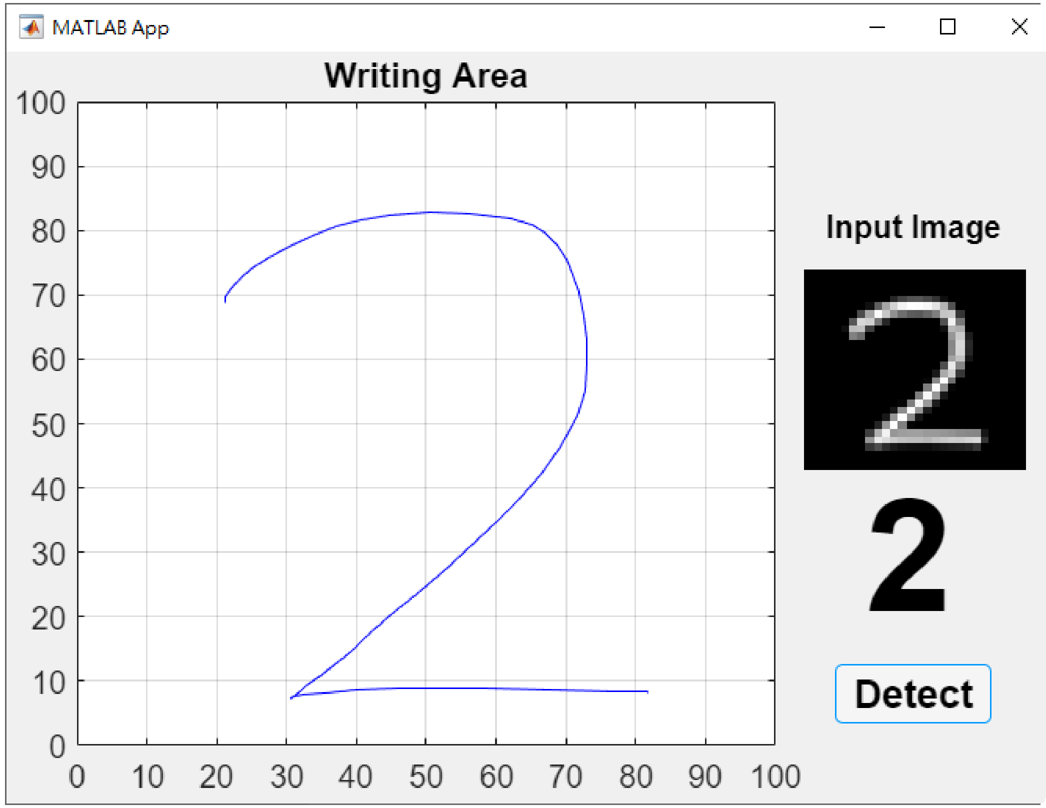
Exercise: MNIST database

- Try the Recognition App
  - **MLmaterials_L11\CNN\Mnist_CNN_App.mlapp**

Test on your own handwritten digits .



100 epochs



THE END

Contact:

盧家鋒 alvin4016@nycu.edu.tw