Video Compression

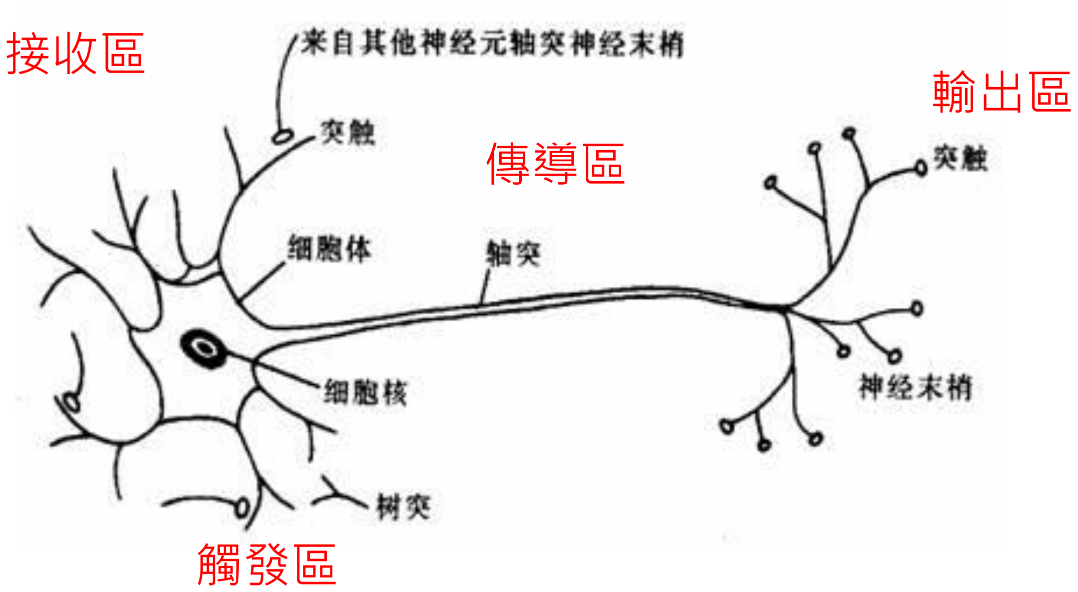
INSTRUCTOR: YAN-TSUNG PENG

DEPT. OF COMPUTER SCIENCE, NCCU

Introduction to Deep Learning

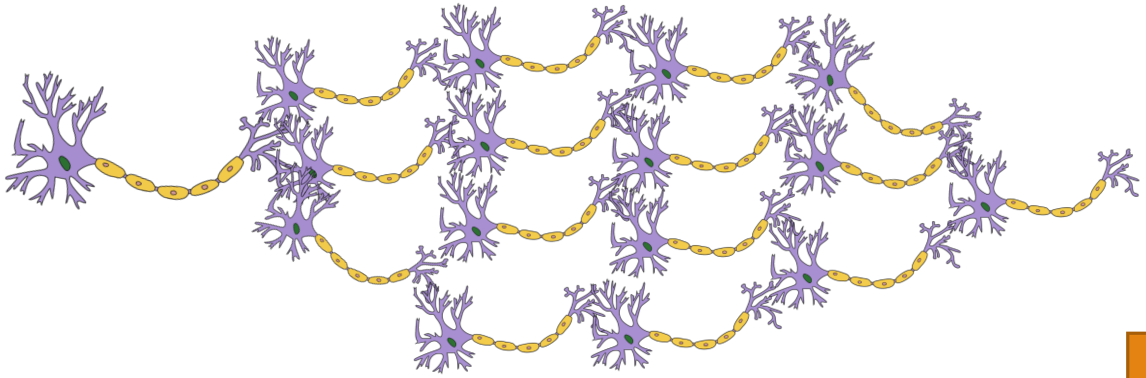
Neural Network

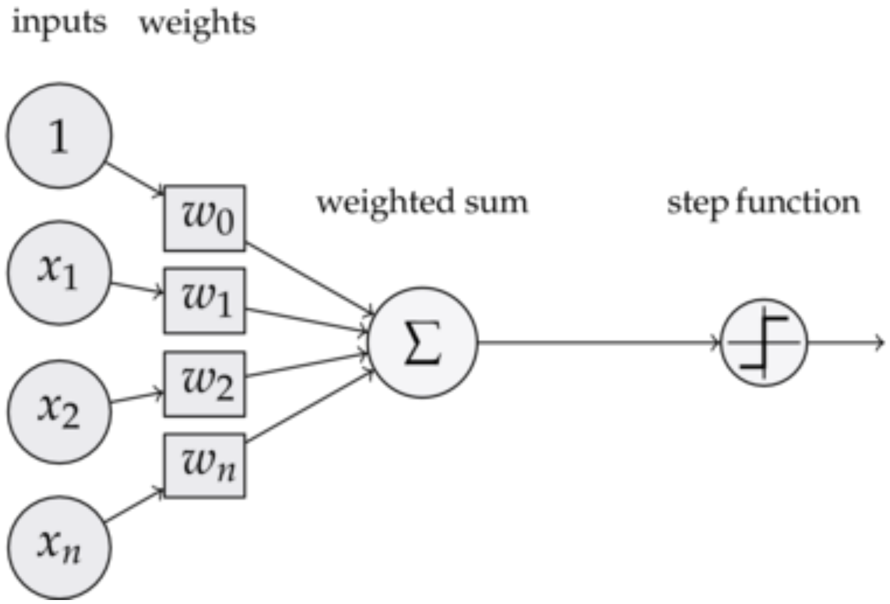
- Biological Neurons



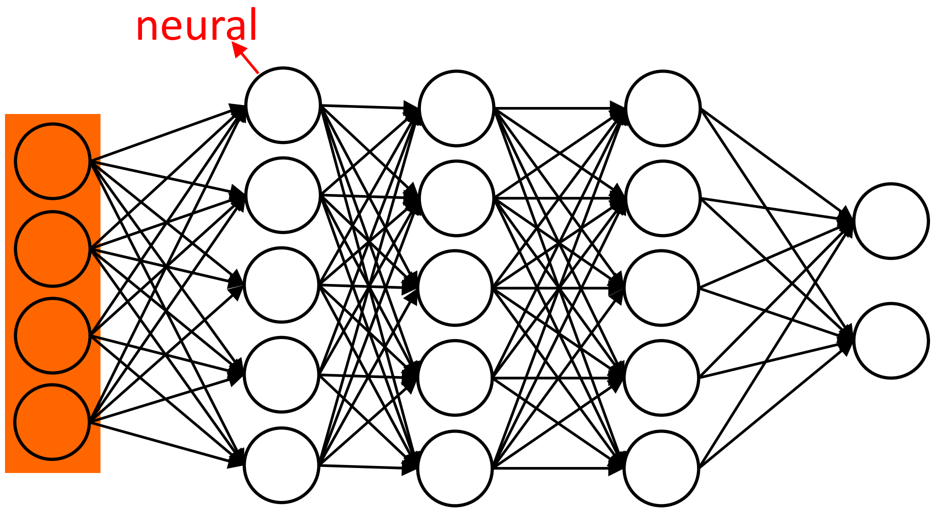
A single neuron can be regarded as a machine with only two states: "yes" when it is activated, and "no" when it is not activated.

Artificial Neuron



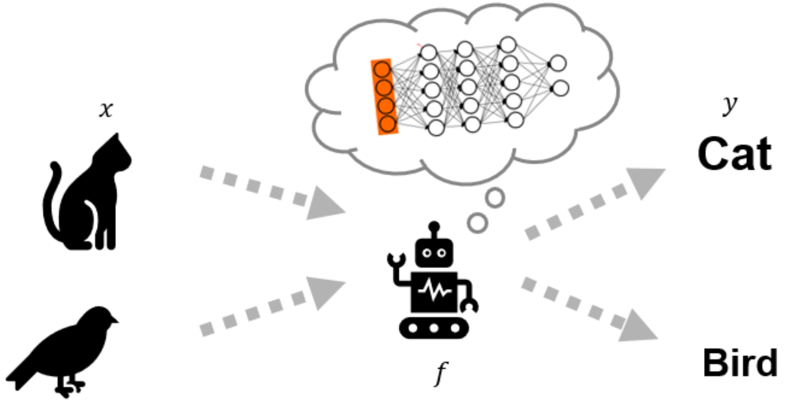


Fully Connected Layer



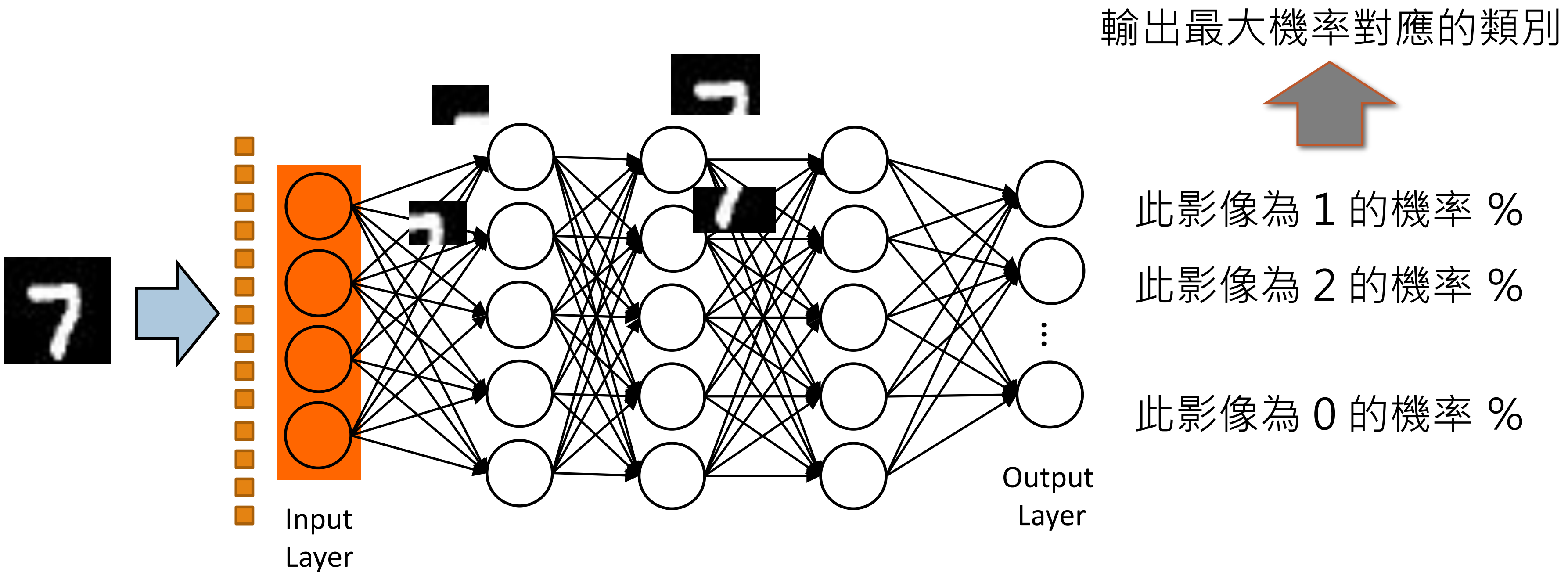
讓電腦模擬人類處理事情的方式

$x , f \rightarrow y$



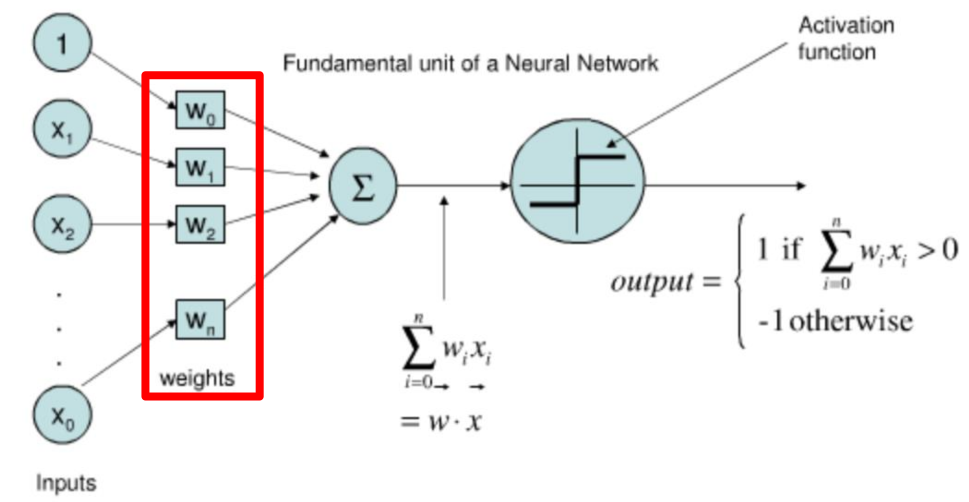
$y = f(x)$

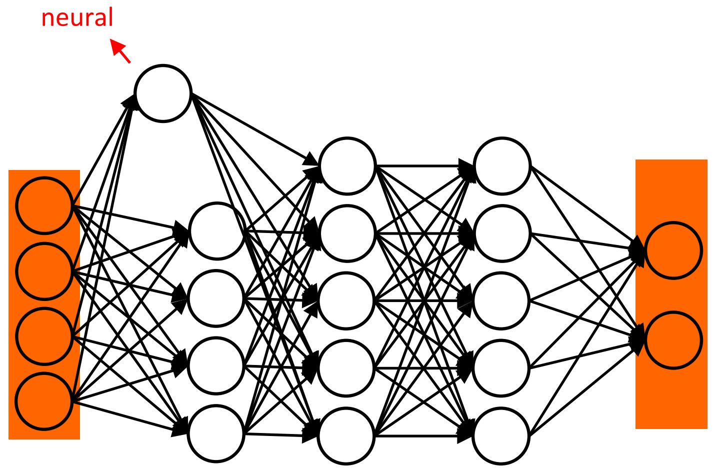
Example – Digit Recognition



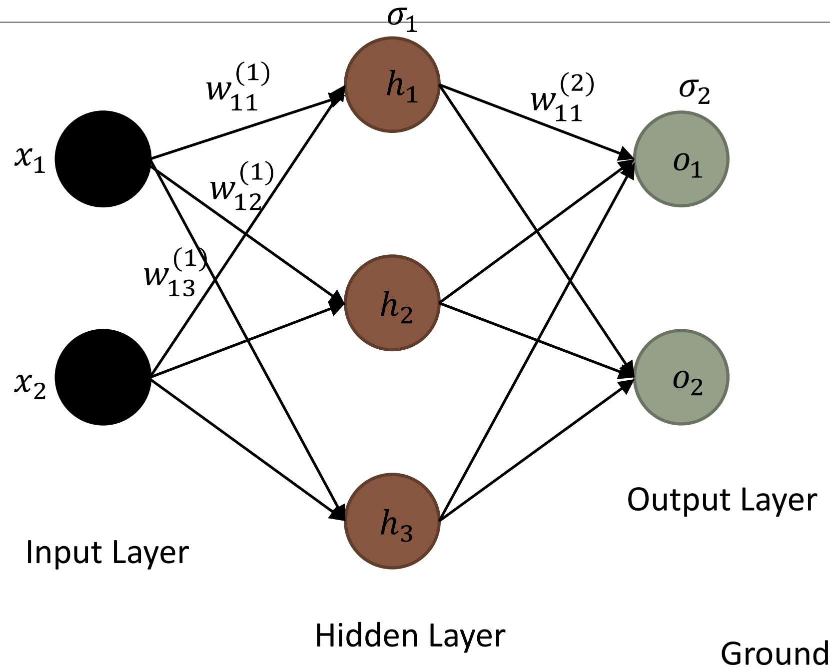
Training Models

- Backpropagation Algorithm
  - It uses the chain rule to compute gradients and applies gradient descent to update the model parameters.





Neural Networks



$$h_1 = \sigma_1(w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2)$$

$$h_2 = \sigma_1(w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2)$$

$$h_3 = \sigma_1(w_{13}^{(1)} x_1 + w_{23}^{(1)} x_2)$$

$$o_1 = \sigma_1(w_{11}^{(2)} h_1 + w_{21}^{(2)} h_2 + w_{31}^{(2)} h_3)$$

$$o_2 = \sigma_2(w_{12}^{(2)} h_1 + w_{22}^{(2)} h_2 + w_{32}^{(2)} h_3)$$

<u>Gradient Descent</u>

1. $\dfrac{\partial L}{\partial o_i} \to \dfrac{\partial o_i}{\partial w_{ij}^{(2)}}$
2. $\dfrac{\partial L}{\partial o_i} \to \dfrac{\partial o_i}{\partial h_i} \to \dfrac{\partial h_i}{\partial w_{ij}^{(1)}}$

Ground-truth: (y_1, y_2)

Regression

$$L(x_1, x_2) = \sum_{i=1}^{2} (y_i - o_i)^2$$

Classification

$$L(x_1, x_2) = -\sum_{i=1}^{2} y_i \log o_i + (1 - y_i) \log(1 - o_i)$$

Training Neural Networks

- Activation Function

k_{th} *layer*

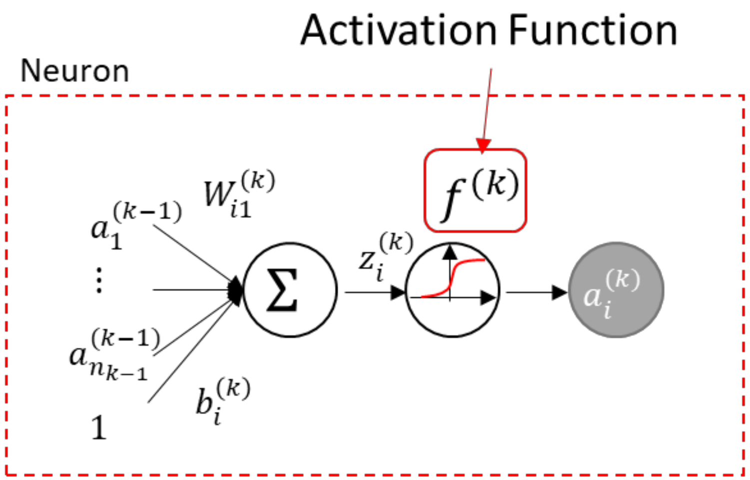
$$a_i^{(k)} = f^{(k)}\left(z_i^{(k)}\right)$$

i_{th} neuron

$$= f^{(k)}\left(\sum_{j=1}^{n_{k-1}} W_{ij}^{(k)} a_j^{(k-1)} + b_i^{(k)}\right)$$

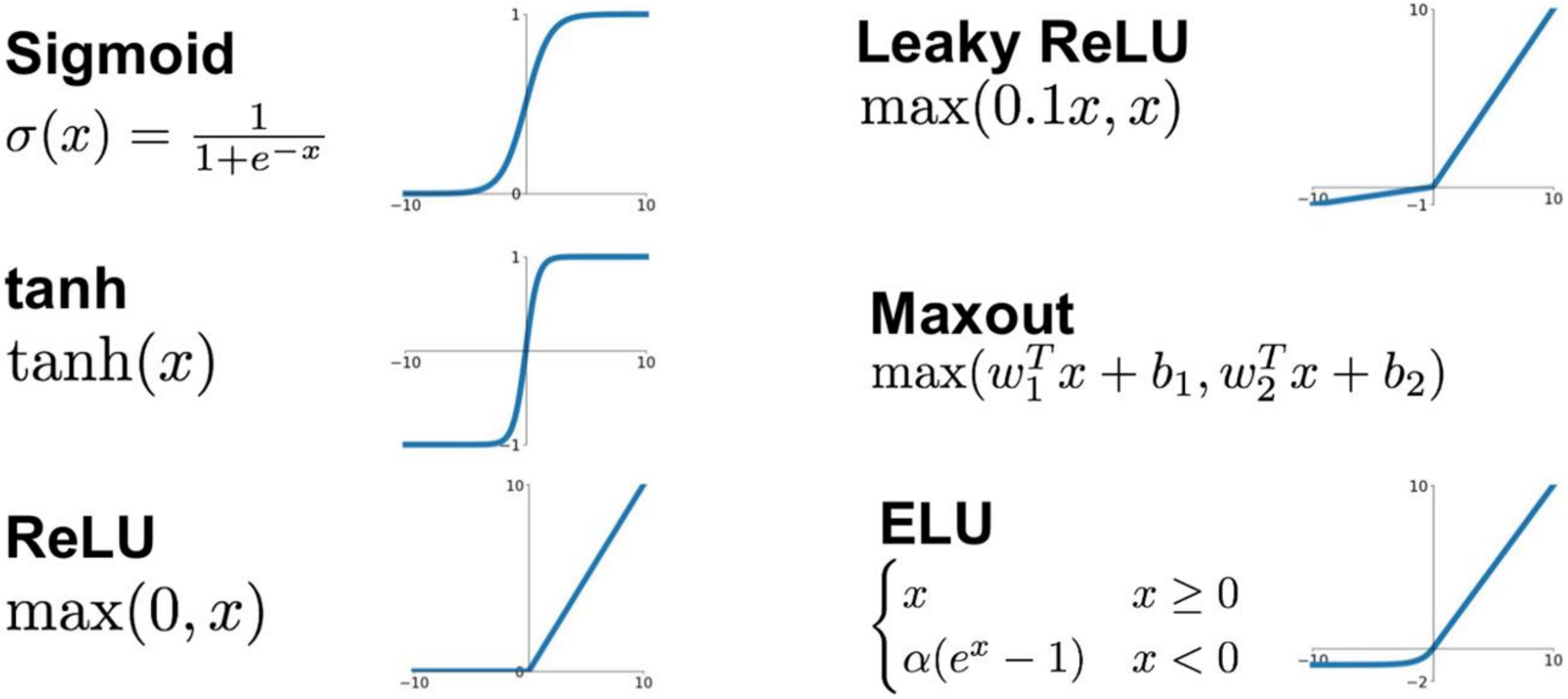
There are n_{k-1} neurons in the previous layer

$$= f^{(k)}\left(\bar{W}_i^{(k)T} \bar{a}^{(k-1)}\right)$$

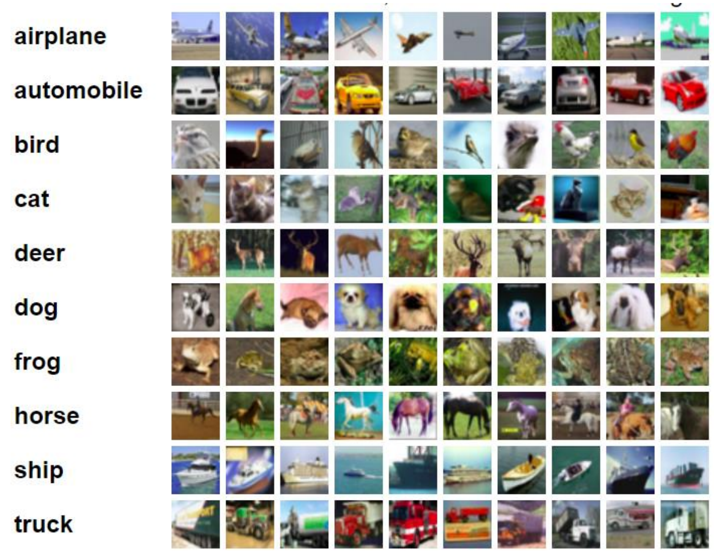


Ex: 1_{st} *layer*

Activation Function



Accuracy on CIFAR10 with Different Activation Functions



CIFAR10

	ReLU	LReLU	PReLU	ELU	SELU	GELU	Swish
ResNet	93.8	94.2	94.1	94.1	93	94.3	94.7
WResNet	95.3	95.6	95.1	94.1	93.2	95.5	95.5
DenseNet	94.8	94.7	94.5	94.4	93.9	94.8	94.8

- Simple choice: ReLU
- Performance fine tuning - LReLU / ELU / GELU
- Sigmoid or tanh are not bad, still useful when used appropriately

Ramachandran et al, "Searching for activation functions," ICLR Workshop 2018

Stochastic Gradient Descent

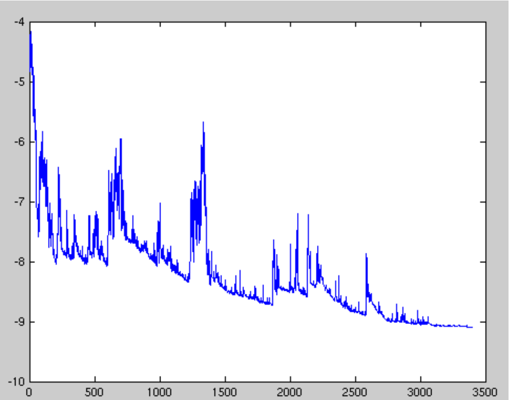
$while \left\|\nabla f\left(\boldsymbol{\theta}; \boldsymbol{X}_{\boldsymbol{i}:\boldsymbol{i}+\boldsymbol{n}-\mathbf{1},:}, \boldsymbol{Y}_{\boldsymbol{i}:\boldsymbol{i}+\boldsymbol{n}-\mathbf{1}}\right)\right\| > \varepsilon\ do$

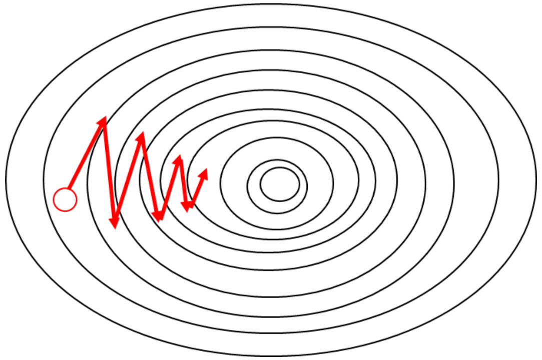
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla f\left(\boldsymbol{\theta}; \boldsymbol{X}_{\boldsymbol{i}:\boldsymbol{i}+\boldsymbol{n}-\mathbf{1},:}, \boldsymbol{Y}_{\boldsymbol{i}:\boldsymbol{i}+\boldsymbol{n}-\mathbf{1}}\right)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$

End while

Batch size = n

Minibatch size=1

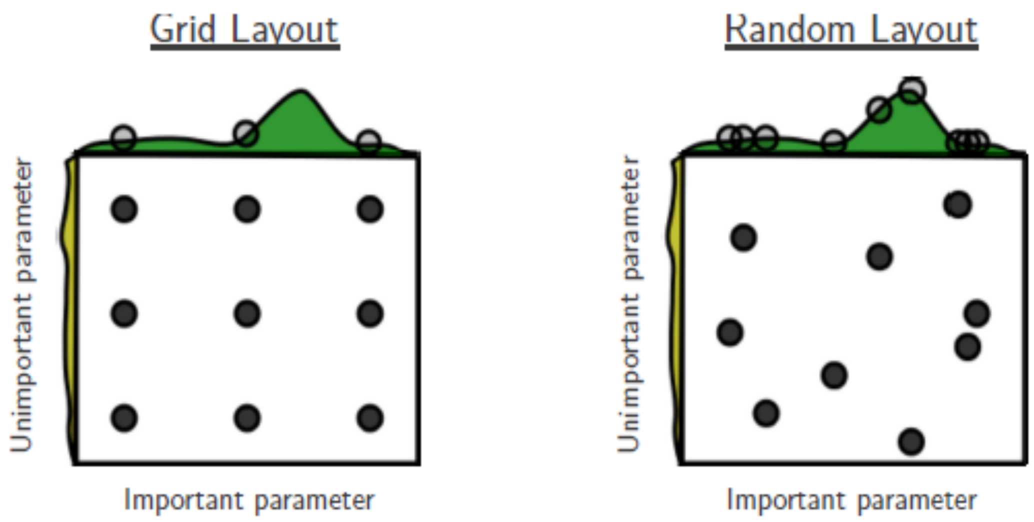




Training process via SGD (Source: Wikipedia)

Choosing Hyperparameters

- Grid Search/Random Search
- Construct a hyperparameter grid



Suggestion:

Random Search

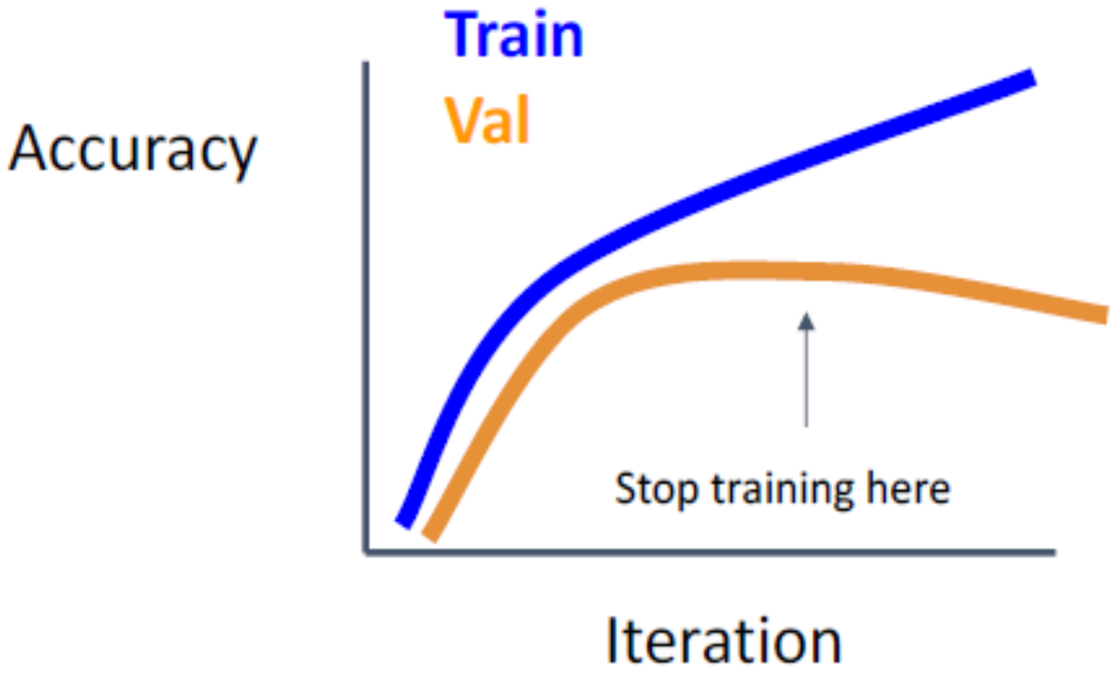
Bergstra and Bengio, “Random Search for Hyper-Parameter Optimization”, JMLR 2012

Suggested Training Steps for Neural Networks

1. Choosing a set of hyperparameters
2. Check the initial loss (does it fit your expectation?)
3. Overfit the small set of data samples
   - Turn off regularization to reach 100% accuracy (~10 minibatches) for debugging
     - Loss not changing -> could be bad initialization or low learning rate
     - Loss NaN or InF -> could be bad initialization or high learning rate
4. Find good learning rate (1e-2, 1e-3, …) on all training data
   - loss should decrease significantly within 100 iterations
5. Constructing a coarse hyperparameter grid
   - Train several models for ~5 epochs on the whole dataset
6. Using fine hyperparameter grid and train longer

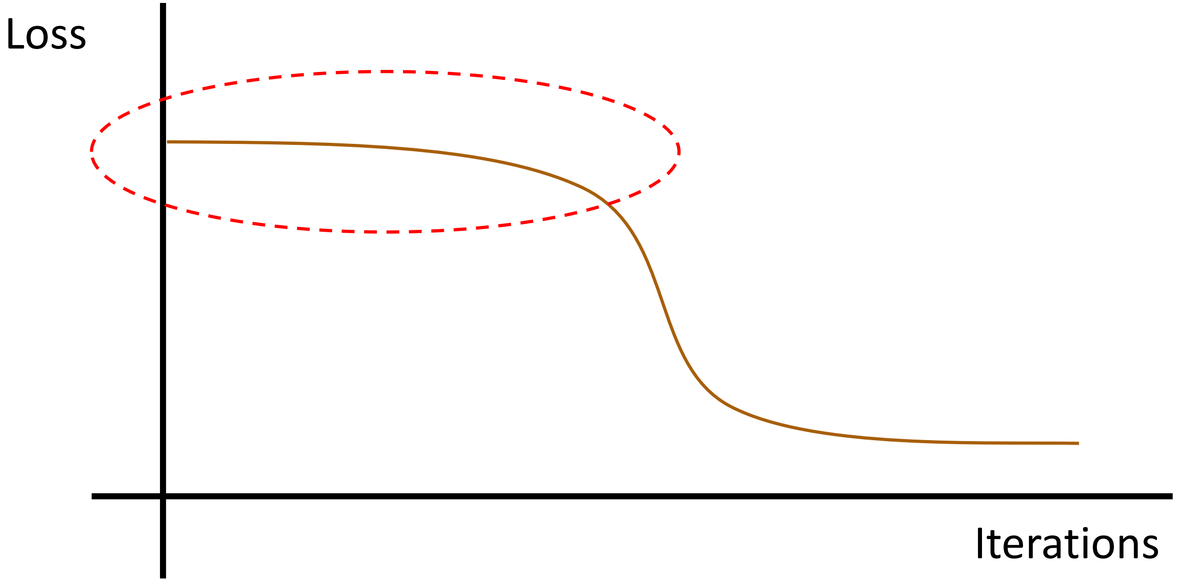
Learning (Loss) Curves

- Stop training the model by checking with the validation accuracy or loss



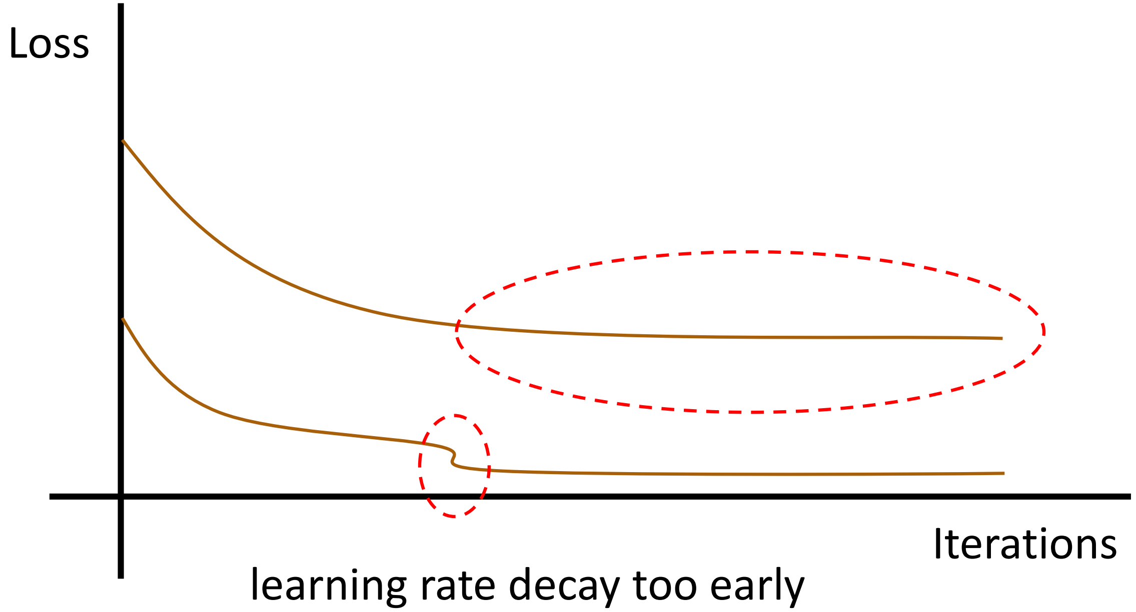
Learning (Loss) Curves

- Check training loss
    - Bad initialization



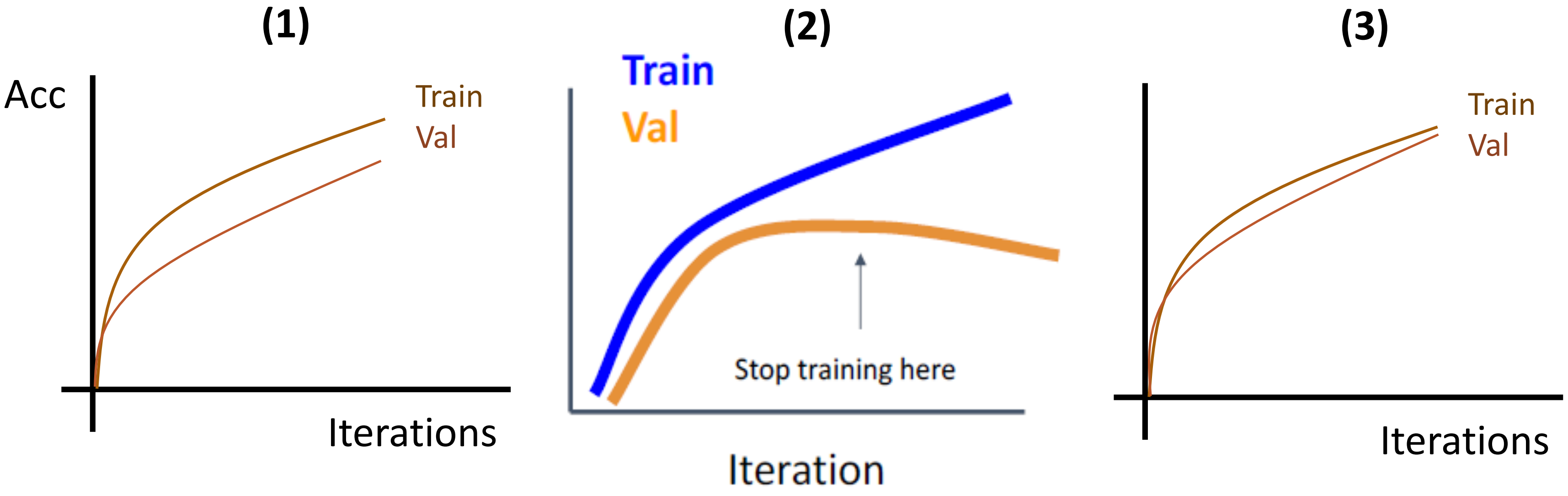
Learning (Loss) Curves

- Flat loss curve
  - Enable learning rate decay



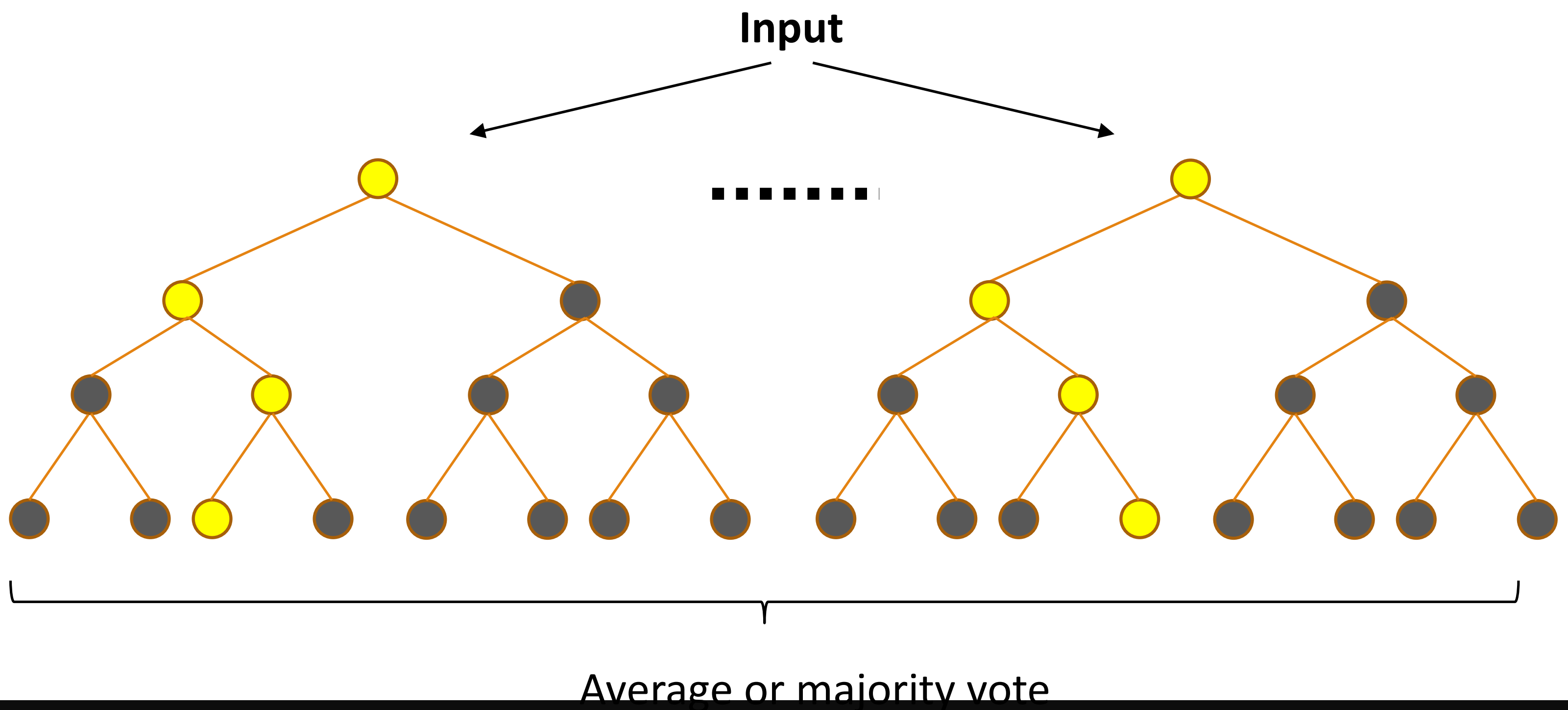
Learning (Loss) Curves

- Check training and validation loss both
  1. Train/val accuracy have a gap but both go up
  2. Overfitting – add regularization/more data/reduce the model size
  3. No gap between train/val – underfitting



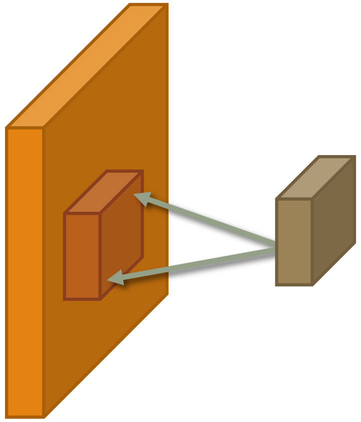
Ensemble Learning

- ❑ Ex: **Random forest**
- ❑ Constructing multiple decision trees for prediction, where all the predictions are averaged (or majority vote) to generate the final result.

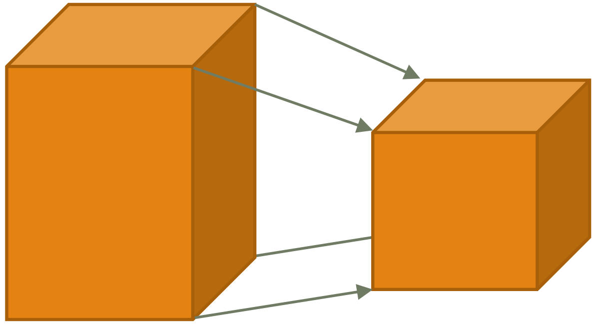


Convolutional Neural Networks

- Convolutional layers + Pooling Layers + Fully-connected layers + normalization layers



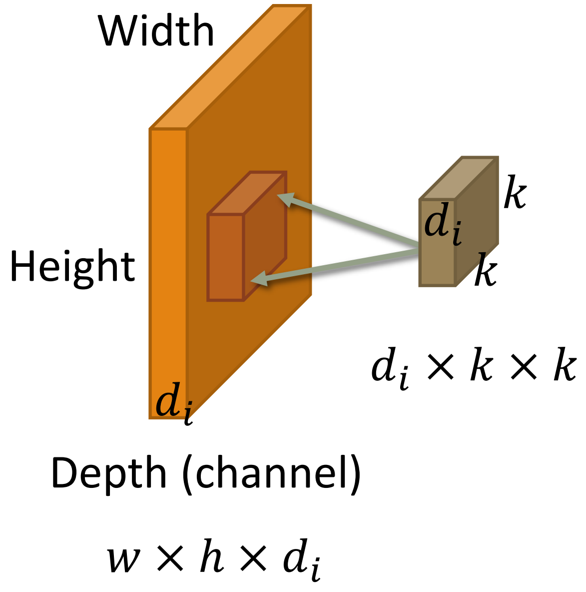
Convolutional layers



Pooling Layers

Convolutional Layers

- Convolutional Layers
  - neurons are in 3D (3D tensor), which are width, height, and depth (channel)



Convolution:
Filtering the image with a convolution kernel, which means calculating a inner product of the kernel and a patch sliding over the image spatially.

Convolution

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	-1	-1
-1	1	-1
-1	-1	1

Kernel 1

Convolution

Stride = 1
Kernel = 3*3

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	-1	-1
-1	1	-1
-1	-1	1

Kernel 1

3			

Convolution

Stride = 1
Kernel = 3*3

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	-1	-1
-1	1	-1
-1	-1	1

Kernel 1

3	-1		

Padding

為每條邊增加 pixels，使經過 filter 後的 feature map 大小與原圖一致。

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

k

0	1	0
1	-1	0
0	0	0

Convolution

2

2

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

k

0	1	0
1	-1	0
0	0	0

Horizontal strides: 2 →

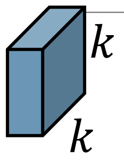
downsample the input by 2 in the horizontal direction

Stride: s

Padding: p

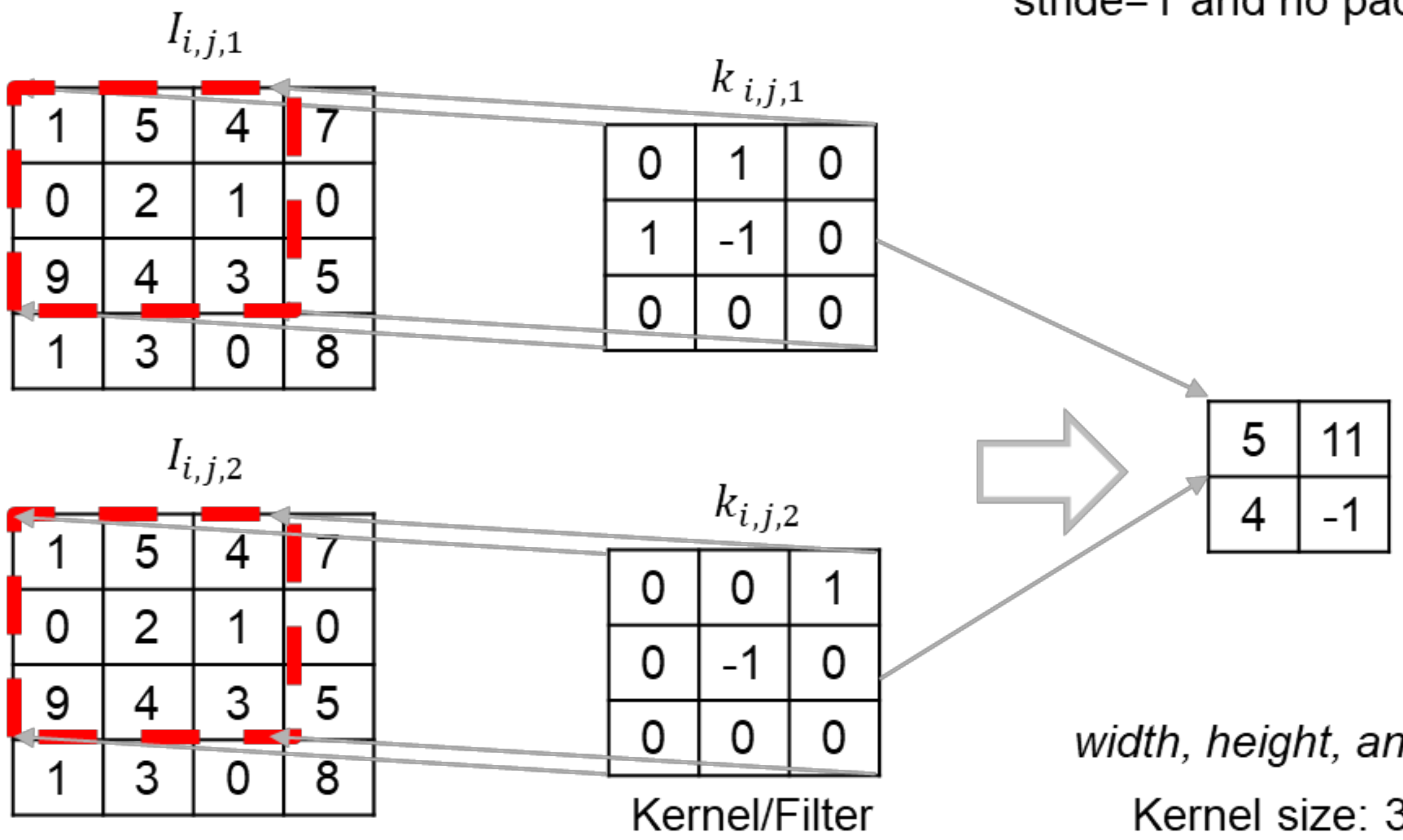
Output: $\left(\left\lfloor\frac{w - k + 2p}{s}\right\rfloor + 1\right)$

Convolution



$k \times k \times 2$

stride=1 and no padding

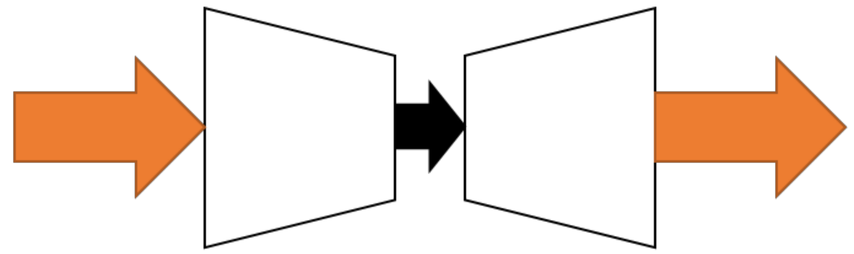


width, height, and depth

Kernel size: 3x3x2

Transposed Convolution

- Why do we need transposed convolution?
  - Up-sampling
  - Encoder-Decoder Structure



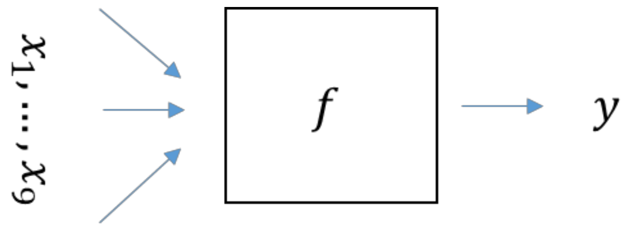
1	5	4	7
0	2	1	0
9	4	3	5
1	3	0	8

0	1	0
1	-1	0
0	0	0

3	5
7	2

Transposed Convolution

- Considering 3x3 convolution is a function, then $f(x_1, \dots, x_9) = y$



- If we would like to go the opposite direction, i.e. $f^{-1}(y) = x_1, \dots, x_9$, how to achieve this?

- Reference: https://github.com/vdumoulin/conv_arithmetic

Transposed Convolution (Deconvolution)

- Convolution Matrix
  - Rearrange the 3x3 kernel to a 4x16 Convolution Matrix

0	1	0
1	-1	0
0	0	0

0	1	0	0	1	-1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	-1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	1	-1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	1	-1	0	0	0	0	0

Each row defines one convolution operation

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

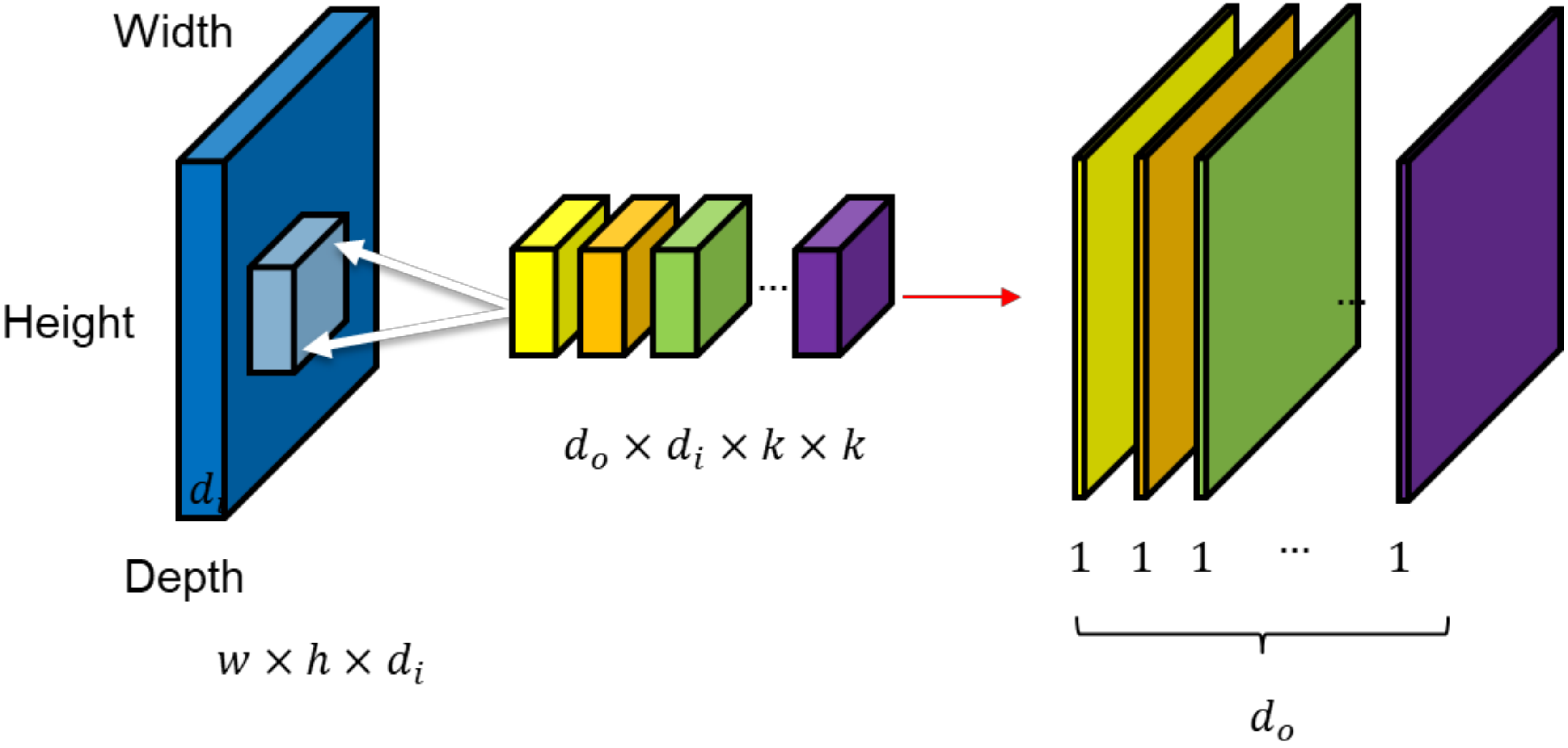
Rearrange the 4x4 input to a 16x1 column vector

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

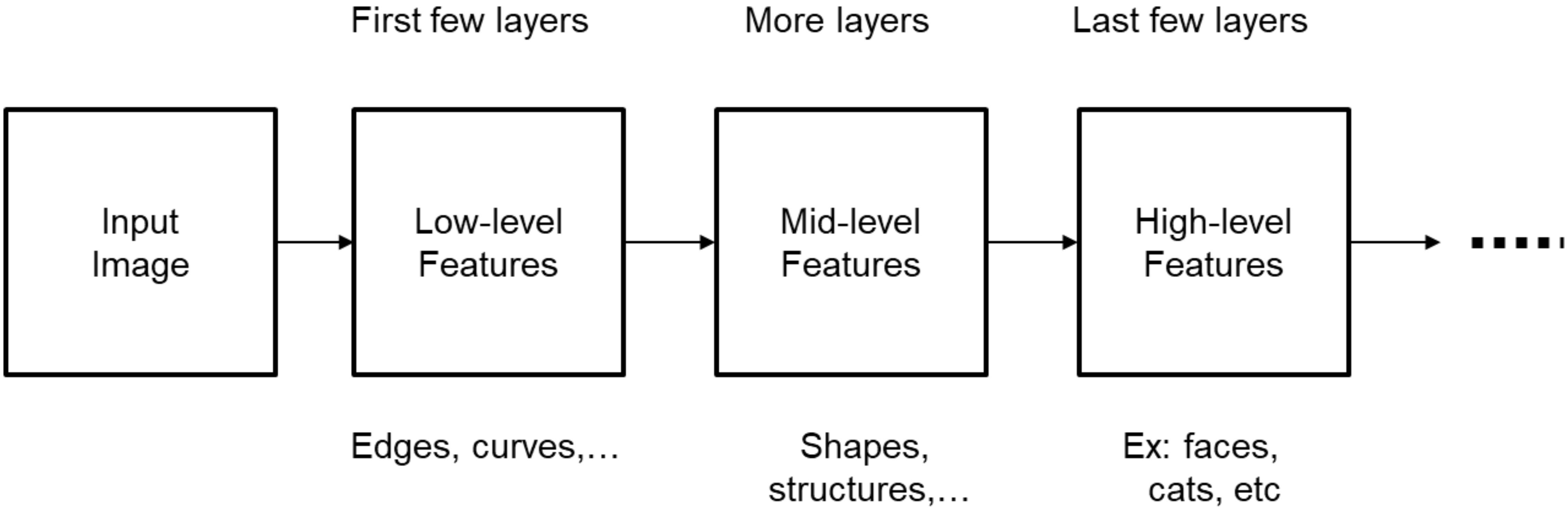
Transposed Convolution (Deconvolution)

- Convolving a 4x4 input with a 3x3 kernel is equivalent to multiplying the 4x16 convolution matrix by the 16x1 input column vector, and the result will be reshaped to a 2x2 matrix from a 4x1 column vector
- So, if you now have a 2x2 matrix, multiplying the transposed convolution matrix (16x4) by the 2x2 matrix, you will have 16x1 column vector (4x4 matrix)
- That is, we now up-sample a 2x2 to a 4x4 matrix

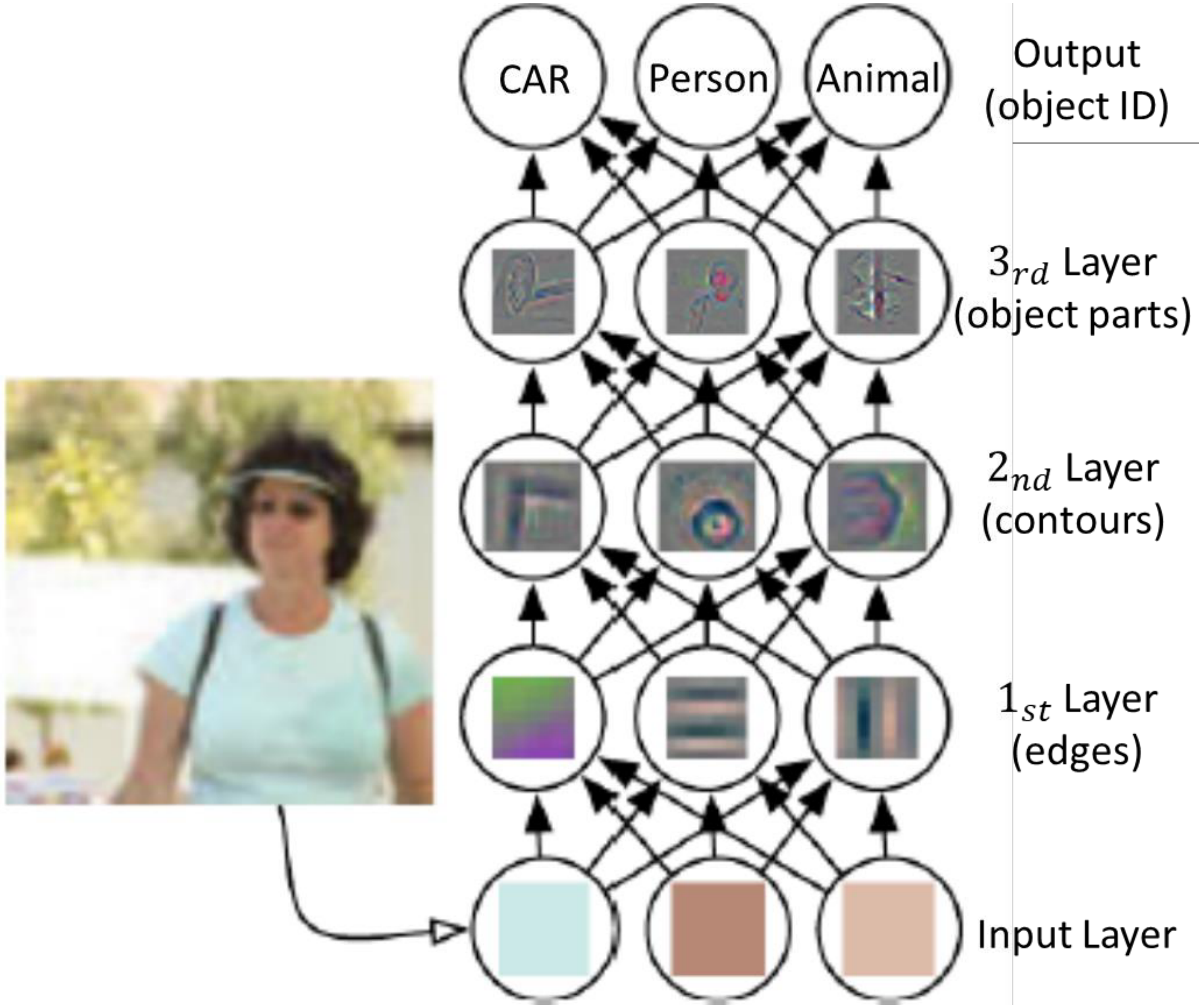
多通道卷積: Convolution



Feature Extraction

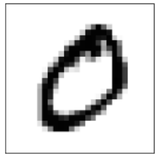


Example


CAR
Person
Animal
Output (object ID)
3_{rd} Layer (object parts)
2_{nd} Layer (contours)
1_{st} Layer (edges)
Input Layer

Different AI Systems

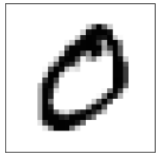
- Rule-based systems

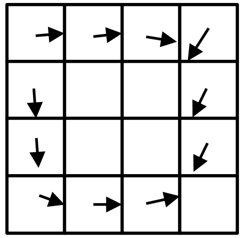


A circle-shaped ring in the center of the image

Zero

- Classic machine learning





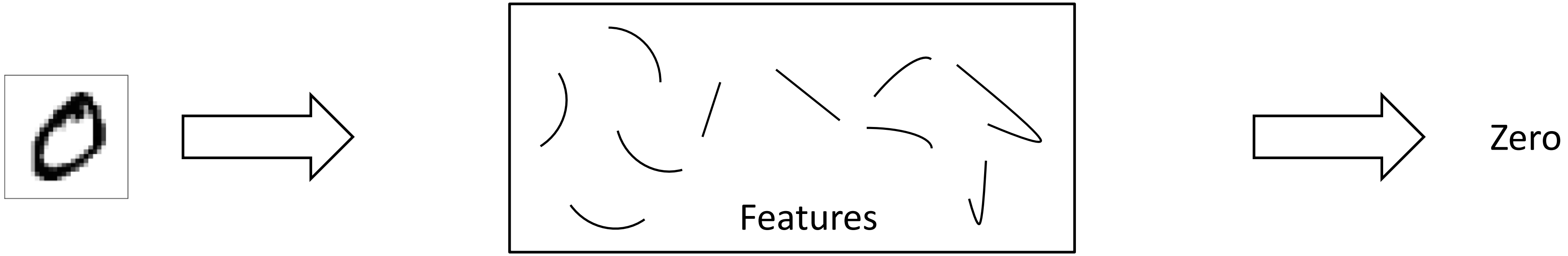
Directions

Magnitudes

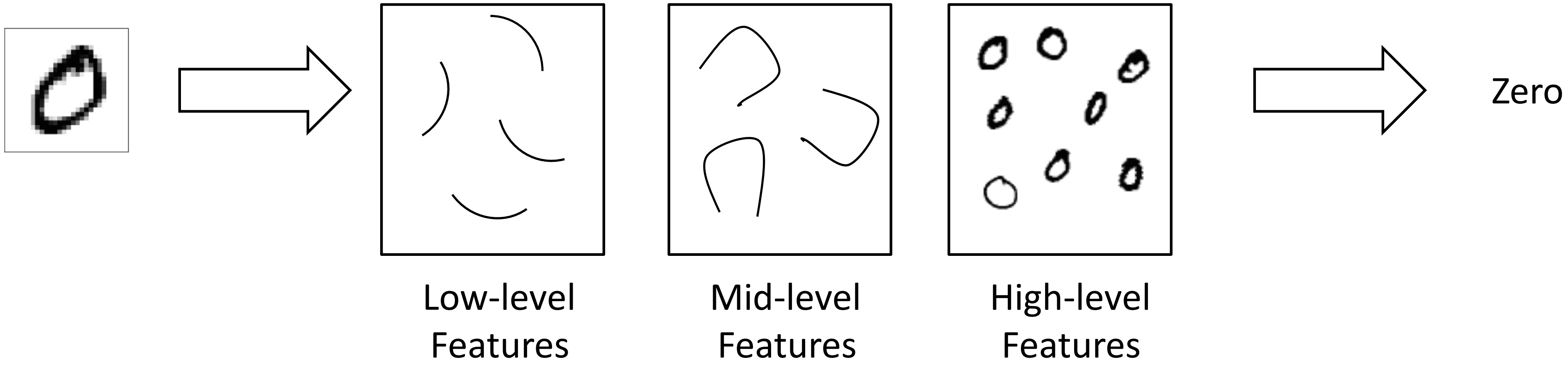
Zero

Different AI systems

- Representation learning

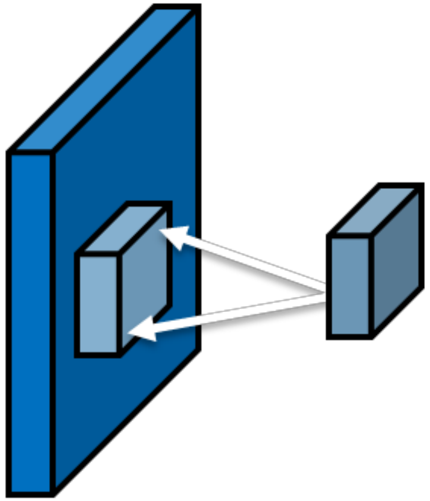


- Representation learning – Deep learning

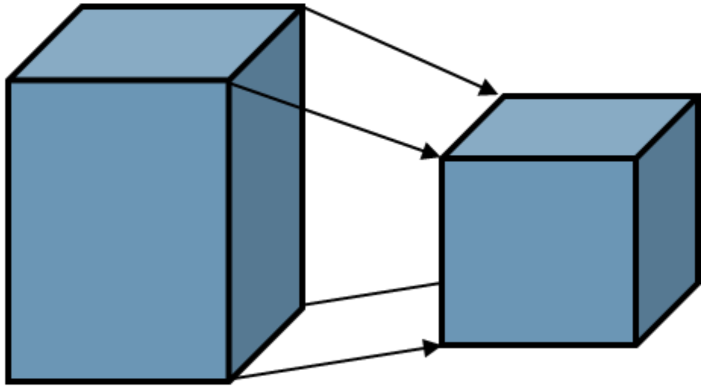


Convolution Neural Network

- 卷積層　Convolution Layers
- 池化層　Pooling Layers
- 攤平　Flatten
- 全連接層　Fully-connected Layers



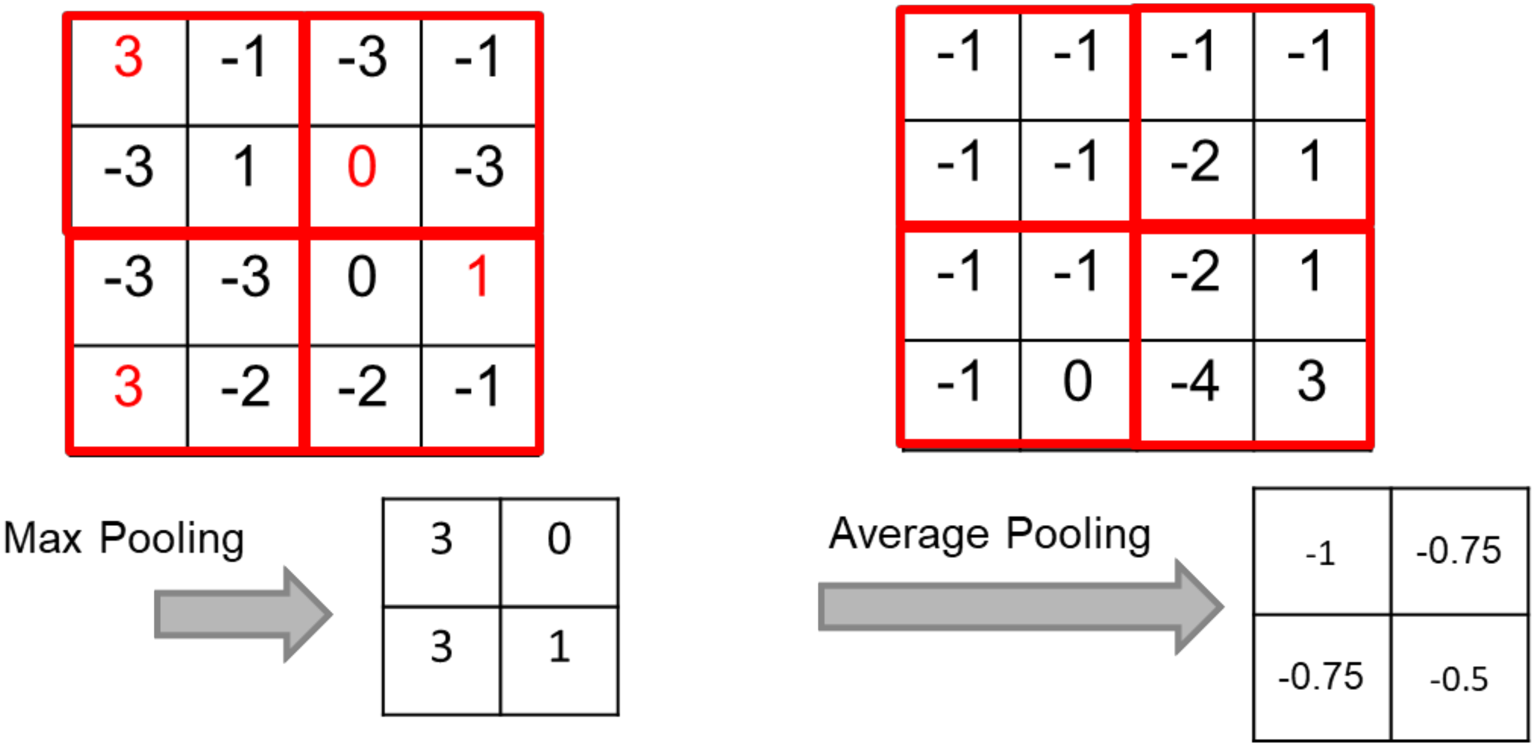
卷積層
Convolution layers



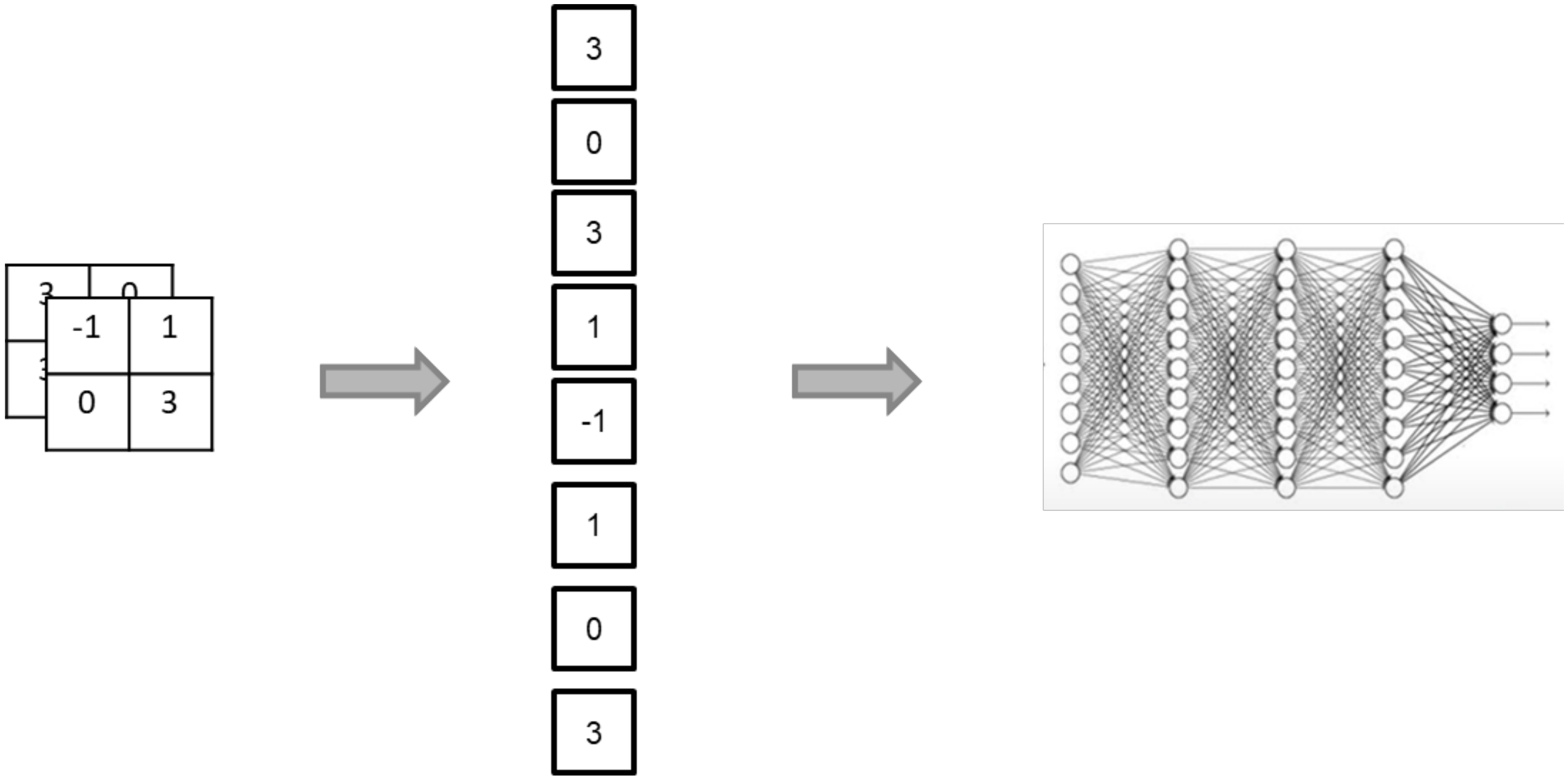
池化層
Pooling Layers

Pooling Layer

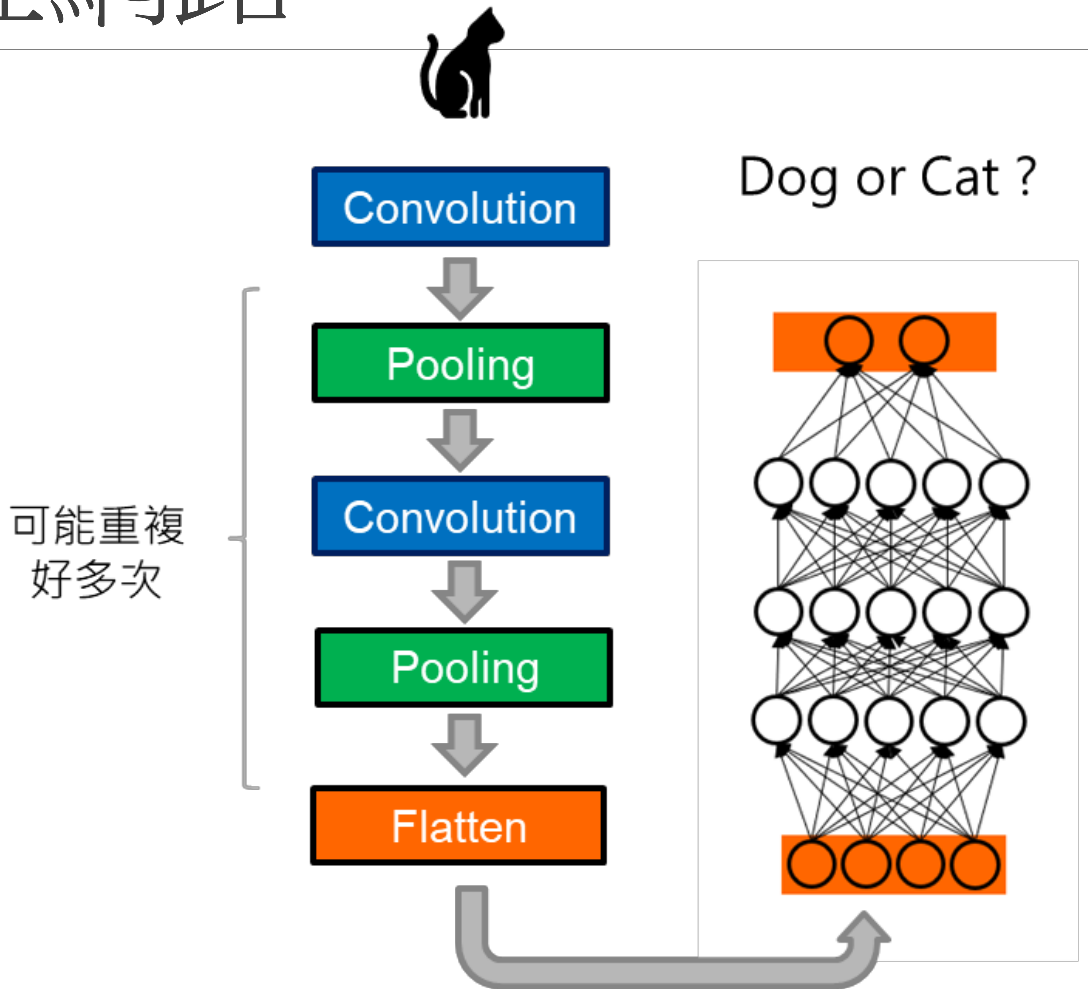
- Purpose: Reduce the number of parameters and computational complexity of the network.



Flattening



卷積神經網路


Convolution
Pooling
Convolution
Pooling
Flatten
可能重複
好多次
Dog or Cat ?

Common Architectures

- Commonly Used Deep Learning Architectures
  - VGG
  - UNet
  - ResNet
  - Transformer

VGG

- proposed by K. Simonyan and A. Zisserman in 2014

Layer	Ch	Rec F
Image	3	
2Conv+Max P	64	3x3
2Conv+Max P	128	3x3
3Conv+Max P	256	3x3
3Conv+Max P	512	3x3
3Conv+Max P	512	3x3
3FC	-	

VGG16 Architecture

Activation: Relu

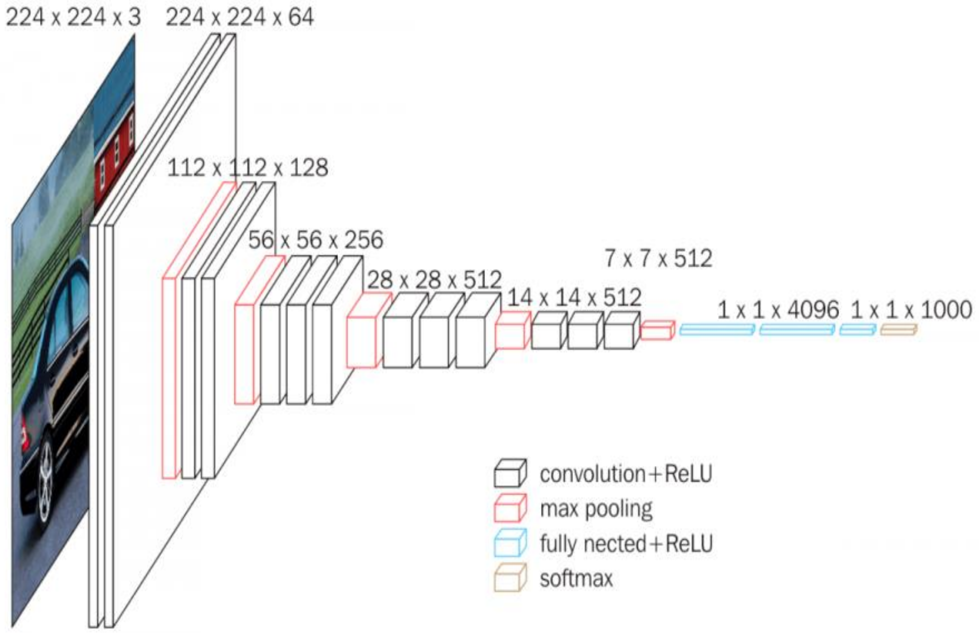
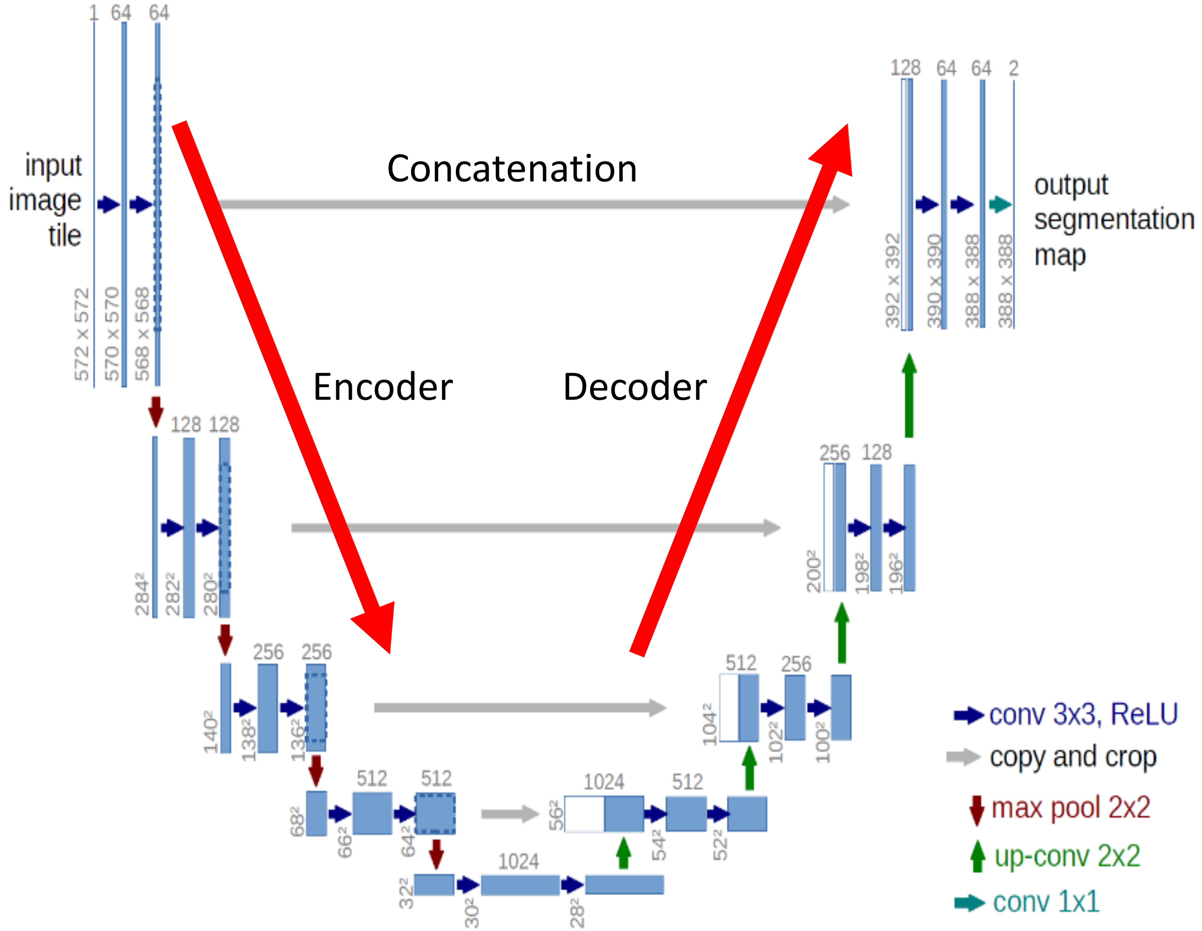


Figure from: https://neurohive.io/en/popular-networks/vgg16/

U-Net

Encoder-Decoder Architecture

No-padding



ResNet

Learn residuals instead of the whole thing

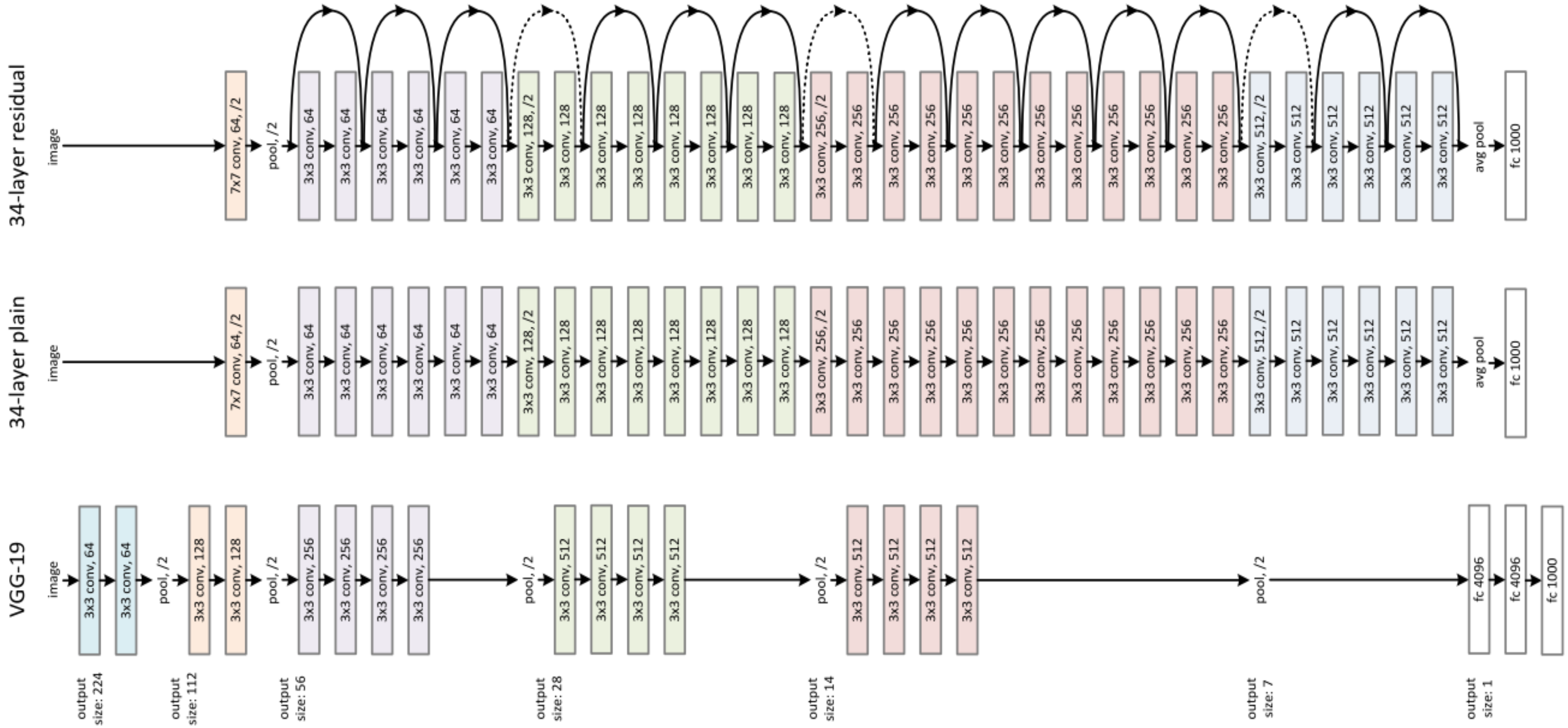
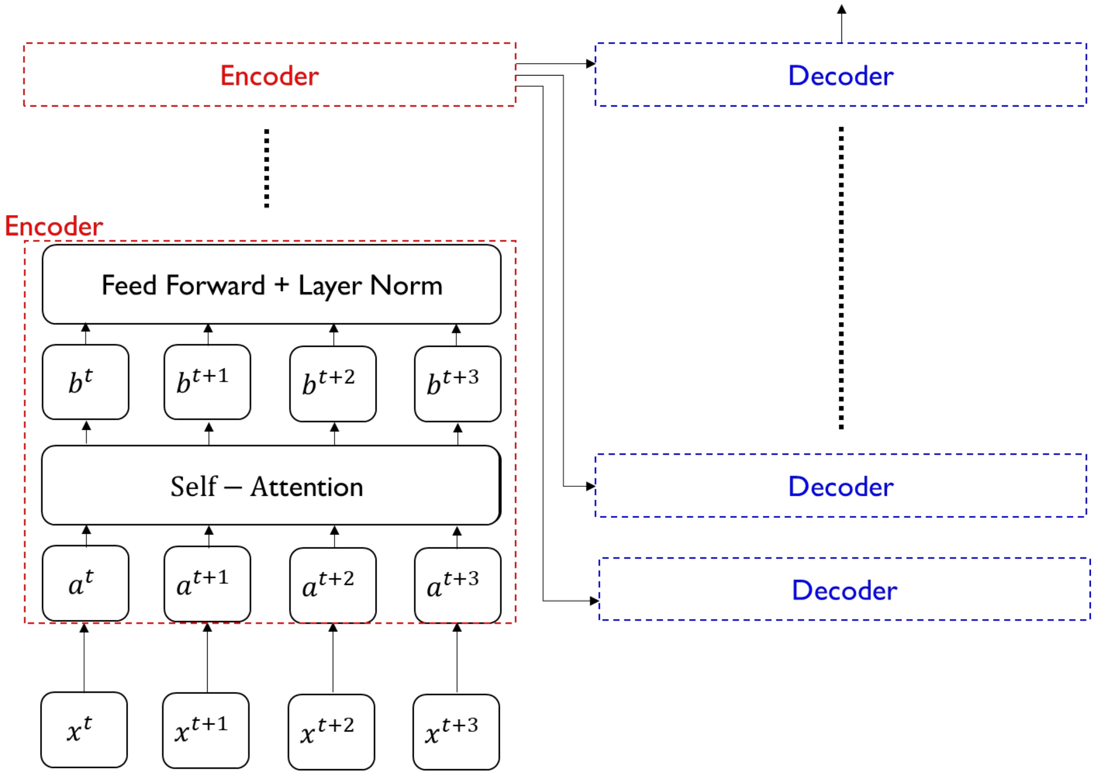
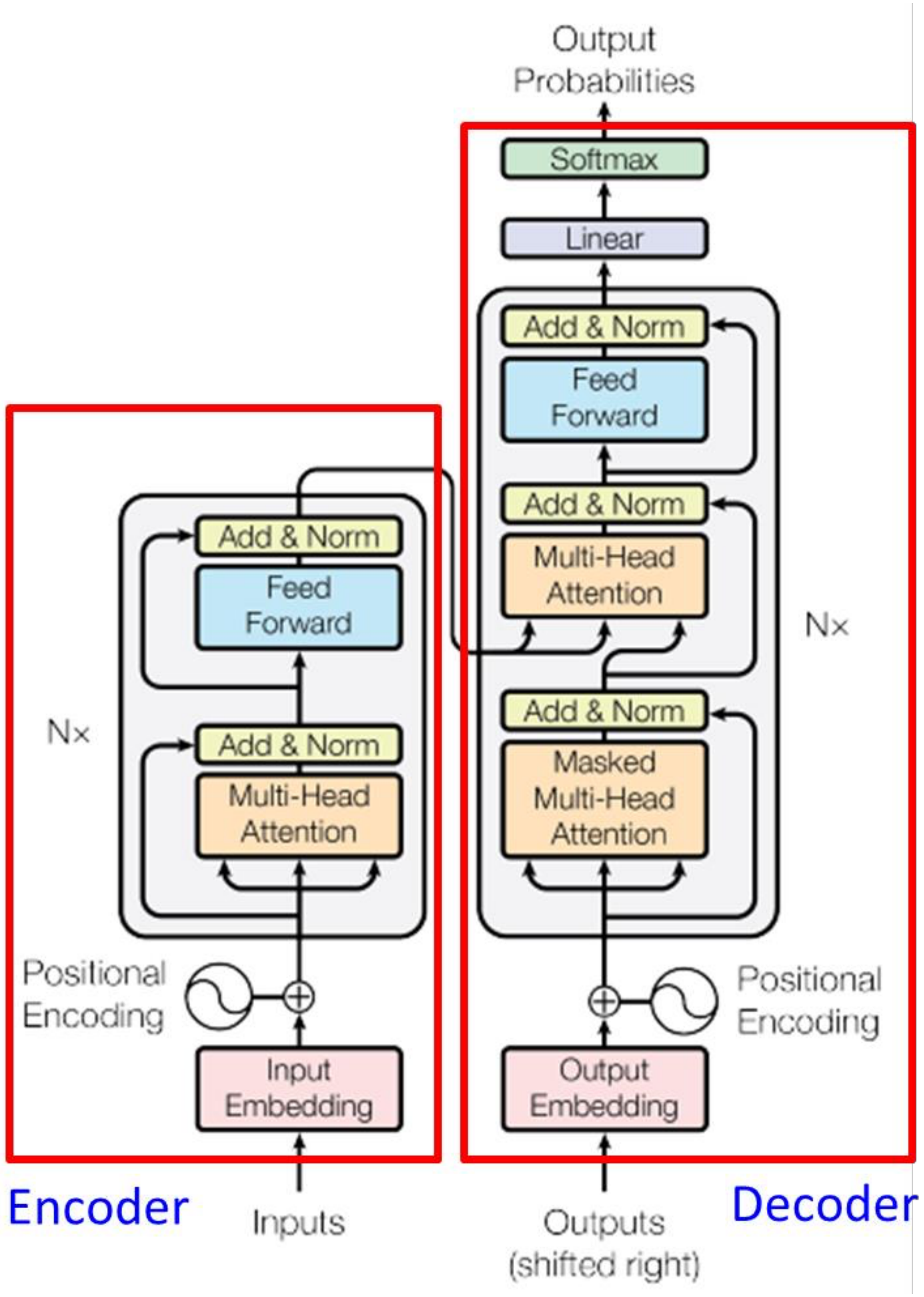


Figure from: https://arxiv.org/abs/1512.03385

Transformer

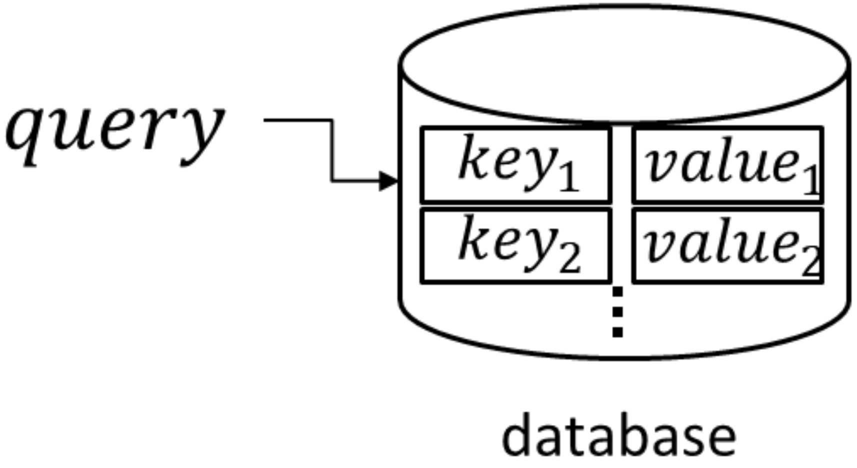




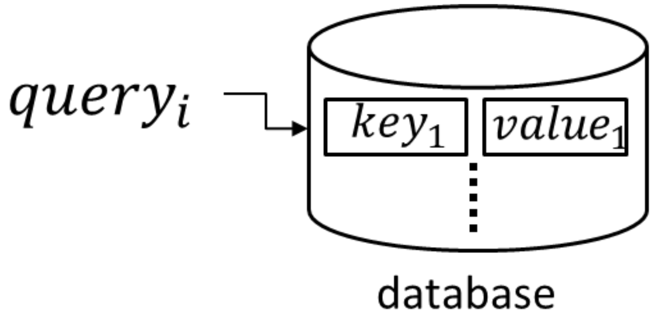
Attention Mechanism

- Imitating information retrieval
  - retrieving v for q based on k in a database
    - value v
    - query q
    - key k

$$Att\,(q,k,v) = \sum_{\forall i} sim\,(q,k_i) \times v_i$$



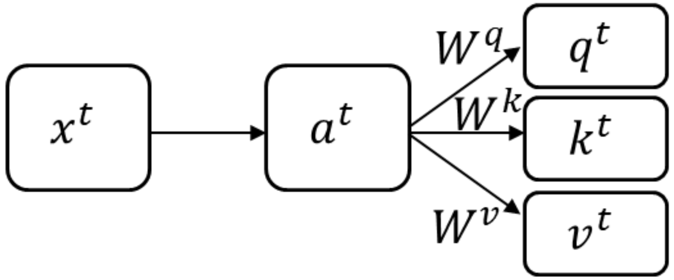
Attention is all you need



q: query (to match others)

k: key (to be matched)

v: information to be extracted



$$q^t = W^q a^t$$

$$k^t = W^k a^t$$

$$v^t = W^v a^t$$

Positional Encoding

- ❑ Goal: considering the order of the words in the input sequence
- ❑ A specific vector added to each input embedding, helping determine the position/distance of/between each word