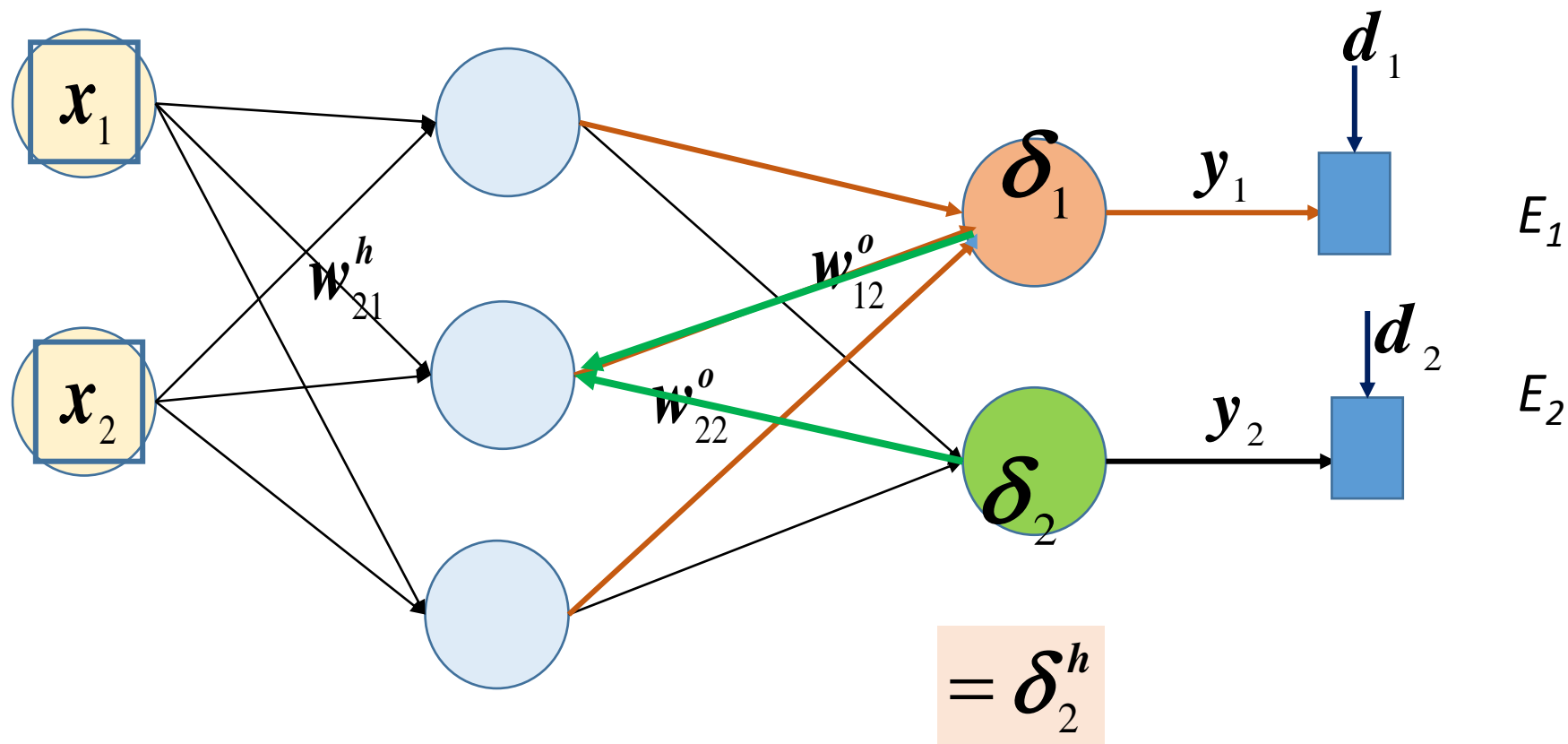


Deep Learning: Introduction

Deep Learning: When network goes deeper, problem arises

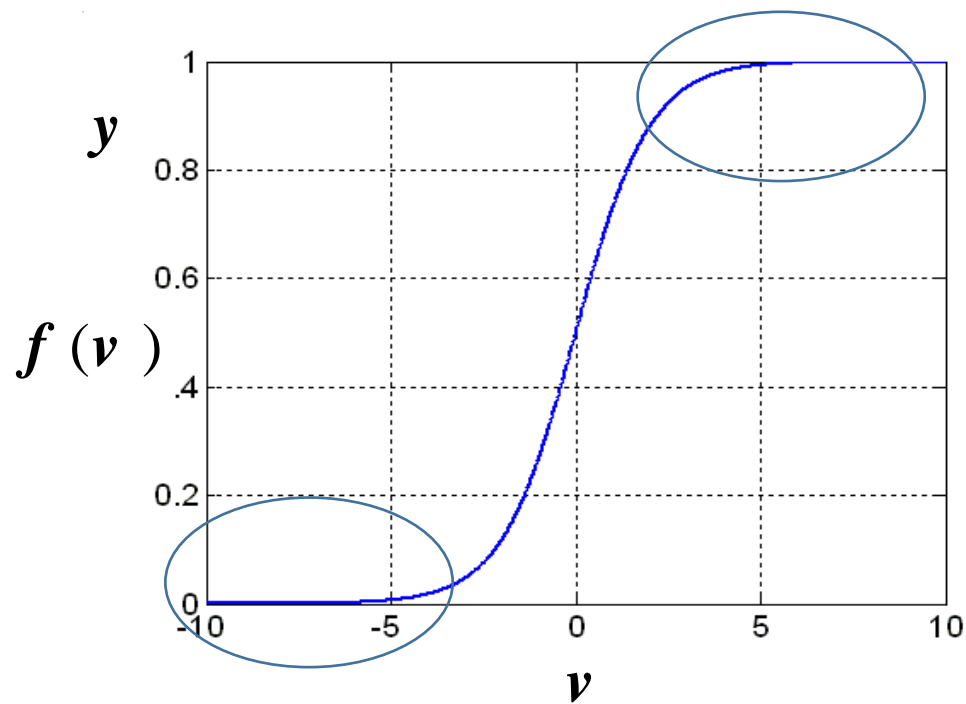
- Becomes hard to converge
 - activation function
- Takes large computing cost
 - Deep network implies large size of network, more computing required
 - Convolution mask
- Tends to over-fitting, reducing generalization capability
 - Large number of parameters, uses large training data
 -



$$\frac{\partial E}{\partial w_{21}^h} = \{ \delta_1 w_{12}^o + \delta_2 w_{22}^o \} f'(v_2^h) x_1$$

$$= \delta_2^h x_1$$

$$w_{21}^h(n+1) = w_{21}^h(n) - \eta \delta_2^h x_1$$



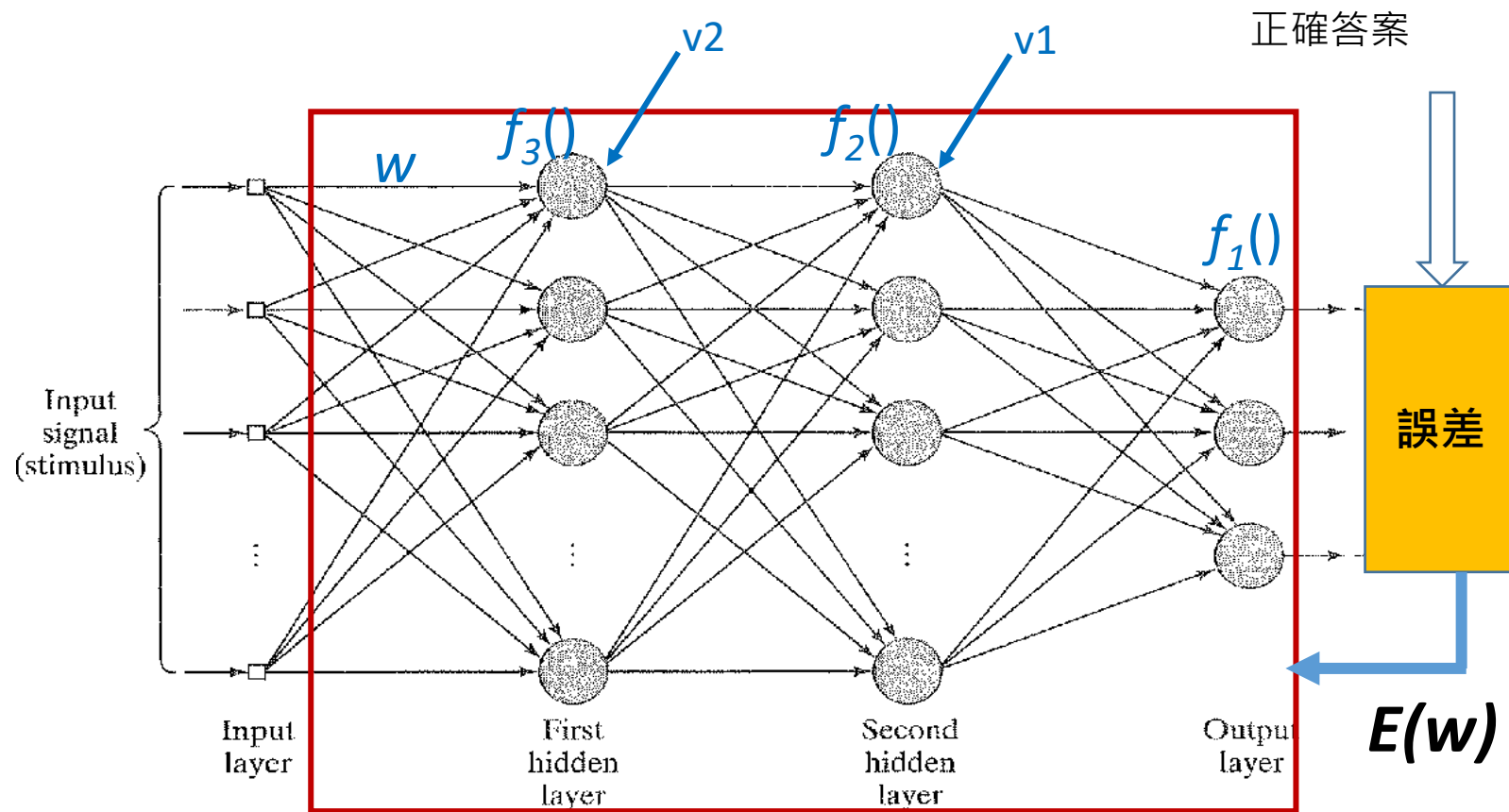
$$E = \sum_{k=1}^{\phi} (e_k(n))^2$$

$$= \sum_{k=1}^{\phi} (d_k - y_k)^2$$

$$y = f(v) = \frac{1}{1 + \exp(-av)}$$

$$f'(v) = af(v) [1 - f(v)]$$

避免梯度消失



利用chain rule

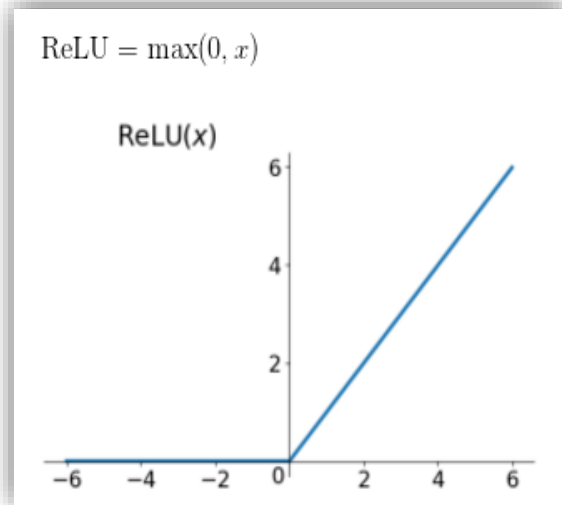
$$\frac{\partial E(w)}{\partial w} = \frac{\partial E(w)}{\partial v_1} \frac{\partial v_1}{\partial v_2} \frac{\partial v_2}{\partial w}$$

惡化梯度消失

長久以來網路無法走太深

Activation function

ReLU



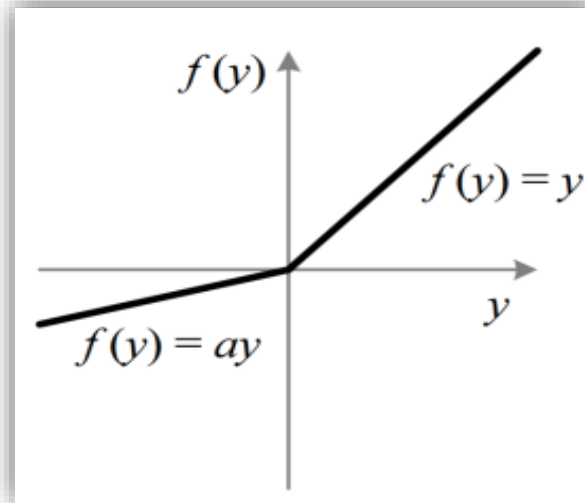
Pros:

- Compute fast (only judge more than the 0)
- Convergence faster than sigmoid and tanh

Cons:

- Dead ReLU Problem

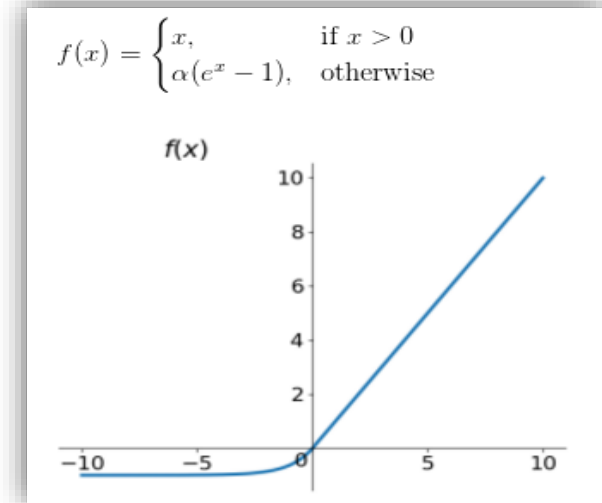
Leaky ReLU



Pros:

- Compute fast (only judge more than the 0)
- Convergence faster than sigmoid and tanh
- Solve Dead ReLU Problem

ELU (Exponential Linear Units)



Pros:

- Compute fast (only judge more than the 0)
- Convergence faster than sigmoid and tanh
- Solve Dead ReLU Problem

Convolution arithmetic for deep learning

input

kernel

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3 ₀	2 ₁	1 ₂	0
0	0 ₂	1 ₂	3 ₀	1
3	1 ₀	2 ₁	2 ₂	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2 ₀	1 ₁	0 ₂
0	0	1 ₂	3 ₂	1 ₀
3	1	2 ₀	2 ₁	3 ₂
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0 ₀	0 ₁	1 ₂	3	1
3 ₂	1 ₂	2 ₀	2	3
2 ₀	0 ₁	0 ₂	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0 ₀	1 ₁	3 ₂	1
3	1 ₂	2 ₂	2 ₀	3
2	0 ₀	0 ₁	2 ₂	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0	1 ₀	3 ₁	1 ₂
3	1	2 ₂	2 ₂	3 ₀
2	0	0 ₀	2 ₁	2 ₂
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0	1	3	1
3 ₀	1 ₁	2 ₂	2	3
2 ₂	0 ₂	0 ₀	2	2
2 ₀	0 ₁	0 ₂	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

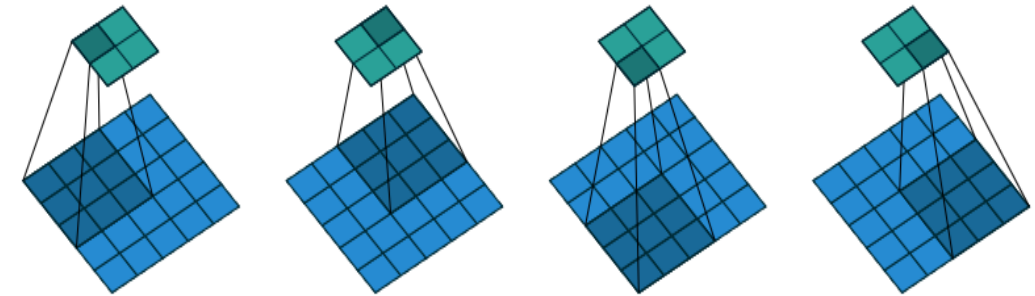
3	3	2	1	0
0	0	1	3	1
3	1 ₀	2 ₁	2 ₂	3
2	0 ₂	0 ₂	2 ₀	2
2	0 ₀	0 ₁	0 ₂	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0	1	3	1
3	1	2 ₀	2 ₁	3 ₂
2	0	0 ₂	2 ₂	2 ₀
2	0	0 ₀	0 ₁	1 ₂

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

- computing a local area of data, instead of all the data
- A linear transformation that preserves this notion of ordering
- It is sparse and reuses parameters

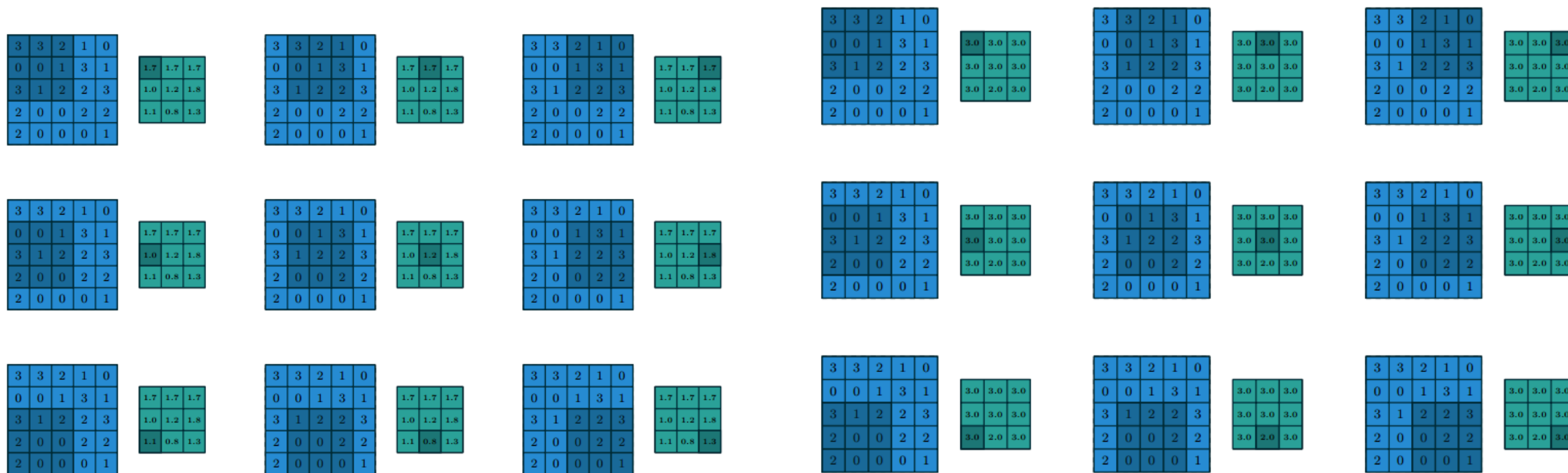


2D convolution (kernel size 3*3 over 5*5input)

Convolution arithmetic for deep learning

- Pooling-- reduce the size of feature maps

- Max pooling: kernel 中找最大值者
- Average pooling: kernel 中平均

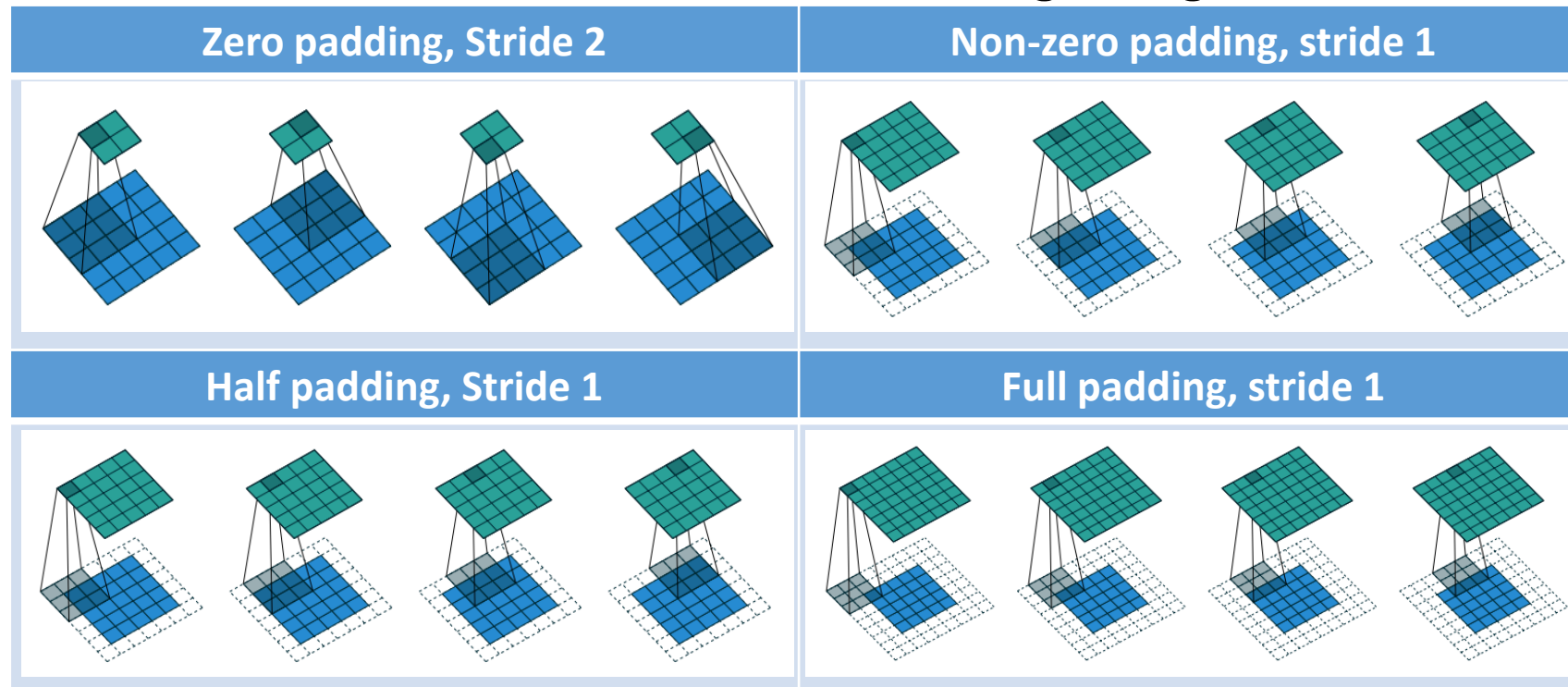


Average pooling

Max pooling

Convolution arithmetic for deep learning

- Stride
 - distance between two consecutive positions of the kernel
- Padding
 - number of zeros concatenated at the beginning and at the end of an axis



Source: [A guide to convolution arithmetic for deep learning](#)

Convolution arithmetic for deep learning

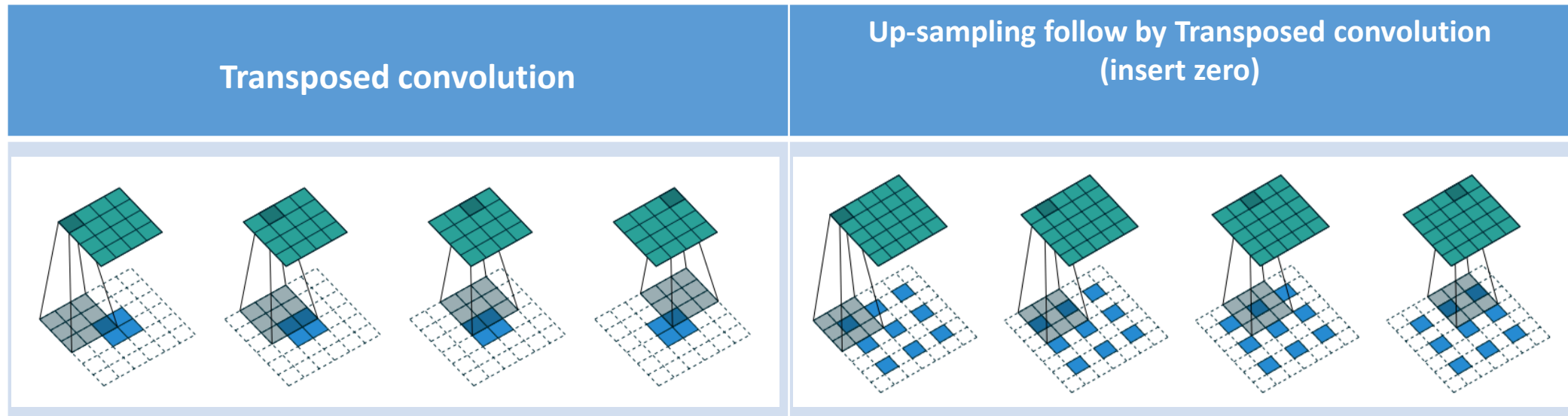
- Transposed convolution
 - from something that has the shape of the output of some convolution to something that has the shape of its input
 - convolution could be represented as a sparse matrix $C(4*4_{\text{input}}, 3*3_{\text{kernel}})$

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

- convolution whose forward and backward passes are computed by multiplying with C and C^T
- defines a transposed convolution whose forward and backward passes are computed by multiplying with C^T and $(C^T)^T = C$ respectively

Convolution arithmetic for deep learning

- Transposed convolution
 - Transposed convolution with interpolation up-sampling



Source: [A guide to convolution arithmetic for deep learning](#)

Bernoulli Distribution PDF:

$$f_X(x) = p^x (1 - p)^{1-x} = \begin{cases} p & \text{if } x = 1, \\ q & \text{if } x = 0. \end{cases}$$

If we have many data $i=1$ to N , with y_i , Using Maximum Likelihood estimation

$$\mathcal{L}(\theta | \mathbf{y}) = \log \prod_{i=1}^N p^{y_i} (1 - p)^{1-y_i}$$

$$= \sum_{i=1}^N \log(p^{y_i} (1 - p)^{1-y_i})$$

$$= \sum_{i=1}^N \{y_i \log(p) + (1 - y_i) \log(1 - p)\}$$

找個最佳的 p ，使得現在手上出現1及0
組合的機率最大

To find the maximum Likelihood →

所以 p 是要找的機率， y 是現在手上出現1及0的 training pattern ground truth

$$- \mathcal{L}(\theta | \mathbf{y}) = - \sum_{i=1}^N \{y_i \log(p) + (1 - y_i) \log(1 - p)\} \text{ is minimized}$$

Cross Entropy Loss function

Binary classification:

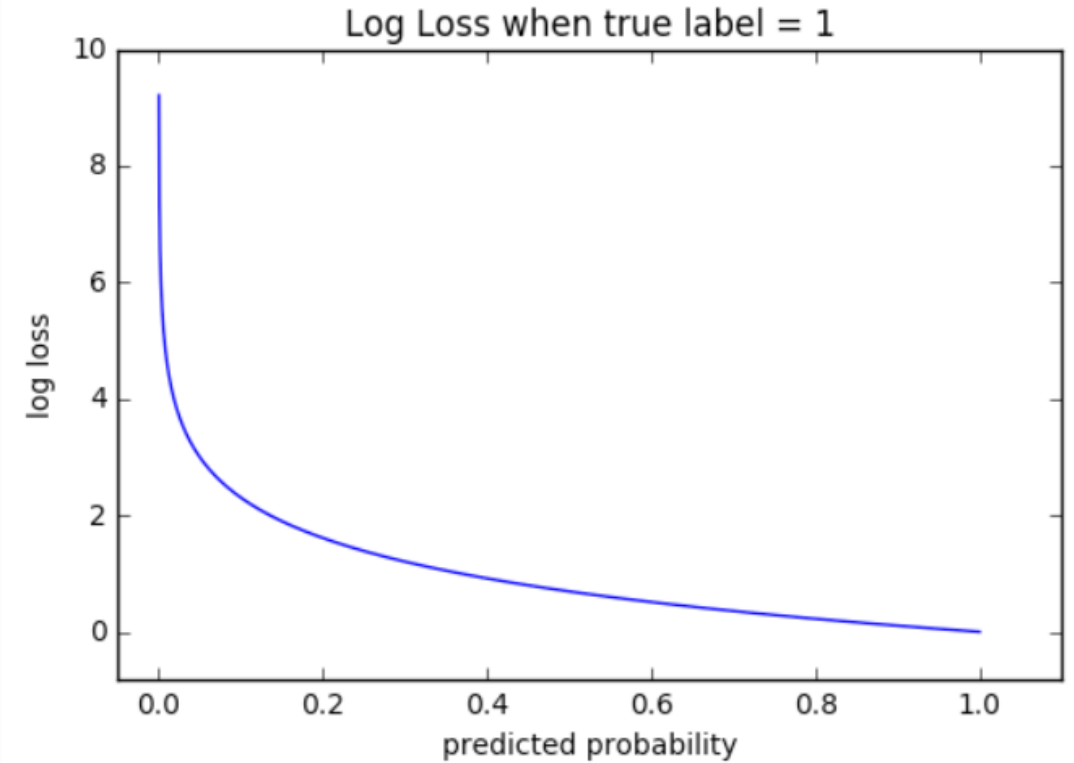
$$-(y \log(p) + (1 - y) \log(1 - p))$$

Multi-classes classification:

$$\text{Cross Entropy loss : } H(y, \hat{y}) = - \sum_{i=1}^{Class} y_i \log(\hat{y}_i)$$

Ground true label

Output from network



Let \hat{y} be the output of the network, y be the desired output

$$\text{By } E = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

$$\frac{\partial E}{\partial \hat{y}} = -y \frac{1}{\hat{y}} + (1 - y) \frac{1}{1 - \hat{y}}$$

If $f()$ is Sigmoid

$$f'(v) = af(v) [1 - f(v)]$$

$$\frac{\partial \hat{y}}{\partial v} = a\hat{y}(1 - \hat{y})$$

In output layer: $\frac{\partial E(w)}{\partial w} = \frac{\partial E(w)}{\partial y} \frac{\partial y}{\partial v} \frac{\partial v}{\partial w}$

$$\frac{\partial E}{\partial v} = \left(-\frac{y}{\hat{y}} + \frac{(1 - y)}{1 - \hat{y}}\right) \hat{y}(1 - \hat{y}) = \underline{y - \hat{y}}$$

不受 Sigmoid 函式微分影響

Softmax function in the output layer for representing probability:

$$f(v_i) = \frac{e^{v_i}}{\sum_{k=1}^m e^{v_k}}$$




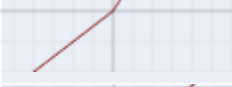


$$\begin{aligned} \frac{\partial f(v_i)}{\partial v_l} &= - \frac{e^{v_i} e^{v_l}}{\left(\sum_{k=1}^m e^{v_k}\right)^2} \\ &= \frac{e^{v_i}}{\sum_{k=1}^m e^{v_k}} \left(-\frac{e^{v_l}}{\sum_{k=1}^m e^{v_k}}\right) = \hat{y}_i (-\hat{y}_l) \end{aligned}$$

$$\begin{aligned} \frac{\partial f(v_i)}{\partial v_i} &= \frac{e^{v_i}}{\sum_{k=1}^m e^{v_k}} - \frac{e^{v_i} e^{v_i}}{\left(\sum_{k=1}^m e^{v_k}\right)^2} \\ &= \frac{e^{v_i}}{\sum_{k=1}^m e^{v_k}} \left(1 - \frac{e^{v_i}}{\sum_{k=1}^m e^{v_k}}\right) = \hat{y}_i (1 - \hat{y}_i) \end{aligned}$$



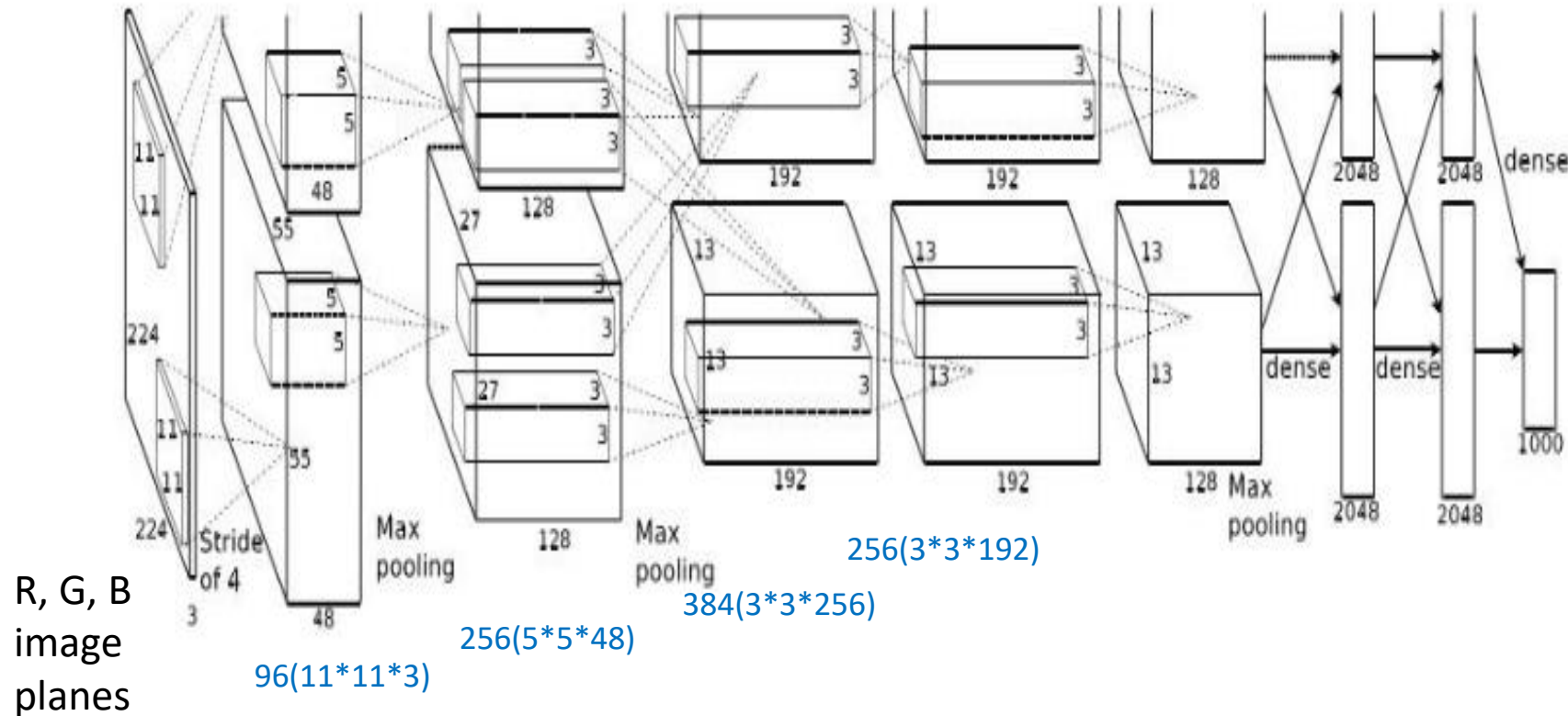
$$\frac{\partial f(v_i)}{\partial v_l} = \hat{y}_i (\delta_{il} - \hat{y}_l)$$

More activation functions

Name	Plot	Equation	Derivative	Range
Logistic (Sigmoid)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0,1)$
Tanh		$f(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
ReLU		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Leak ReLU		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
ELU		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\alpha, \infty)$
Softsign		$f(x) = \frac{x}{1 + x }$	$f'(x) = \frac{x}{(1 + x)^2}$	$(-1,1)$
Softmax		$f\left(\vec{x}\right)_i = \frac{e^{x_i}}{\sum_{k=1}^K e^{x_k}}$ for $i = 1 \dots K$	$f'\left(\vec{x}\right)_i = f\left(\vec{x}\right)_i (\delta_{ij} - f\left(\vec{x}\right)_j)$	$(0,1)$

AlexNet

- 5 convolutional layers and 2 fully connected layers
- Max pooling layer after first, second and fifth layers
- Use ReLU as activation function



AlexNet

- Trained the 1.3 million high-resolution images in the LSVRC-2010(ImageNet Large Scale Visual Recognition Competition)
- GPU implementation of convolutional nets



Model	Top-1	Top-5
<i>Sparse coding [2]</i>	47.1%	28.2%
<i>SIFT + FVs [24]</i>	45.7%	25.7%
CNN	37.5%	17.0%

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

- 11, 13, 16 and 19 layers implementation
- 3*3 and 1*1 kernel sizes for deep model

Using smaller masks
The first using 1*1 kernel size



Simplifying networks to afford deeper

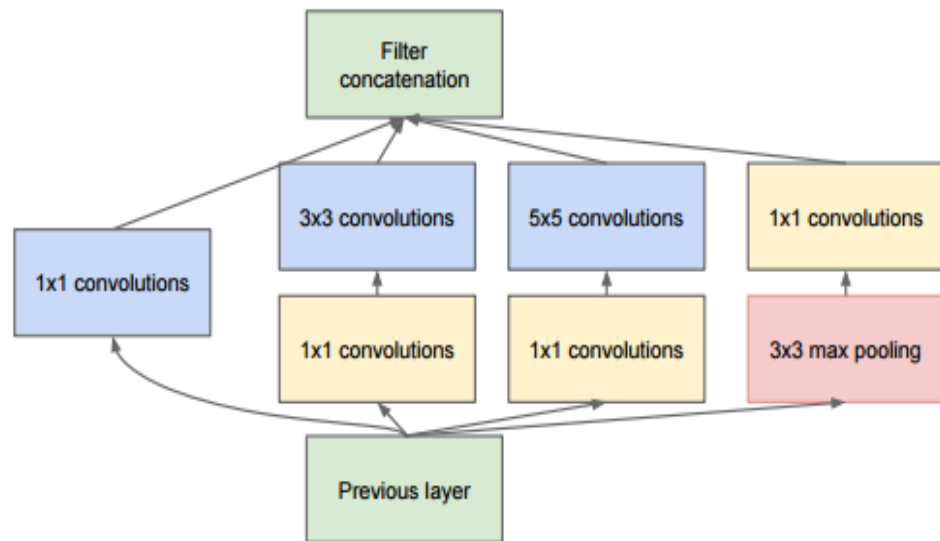
Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Source: [Very Deep Convolutional Networks for Large-Scale Image Recognition](#) (ICLR 2014)

GoogleNet

- Inception
- compute various convolution kernel sizes in same network
- 1*1 convolutions for reducing the number of maps

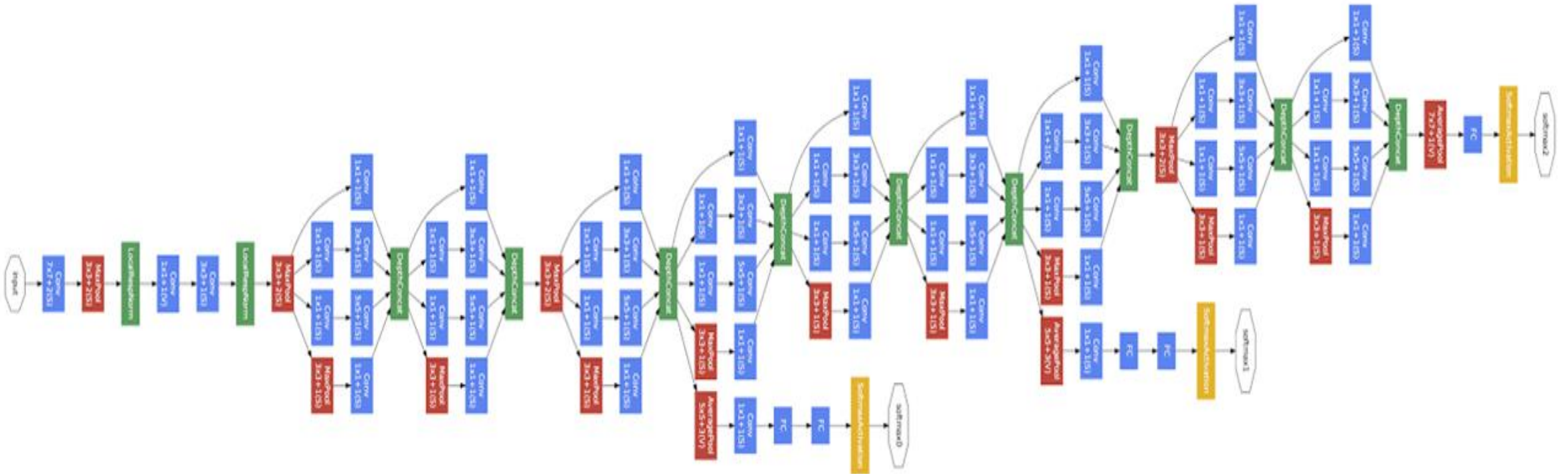


Inception model

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

GoogleNet

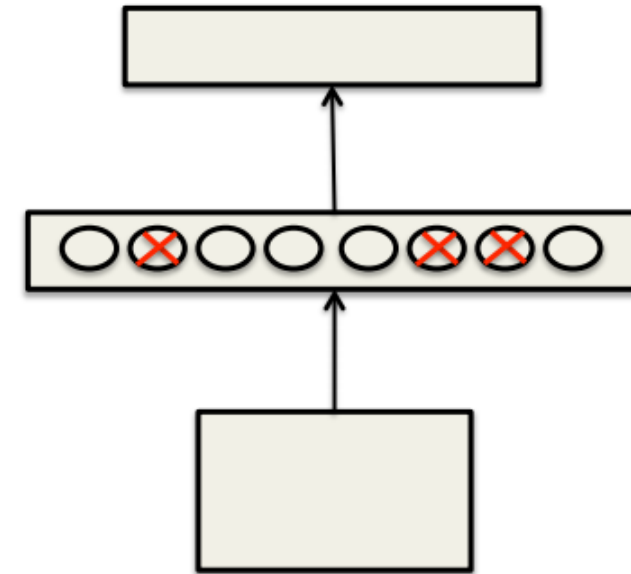
- Architecture
 - Based on inception model



Source: [Going deeper with convolutions](#) (CVPR 2015)

Dropout and Pruning

- An efficient way to average many large neural nets
 - In training time : Randomly omit each hidden unit with specified probability (P%)
 - In test time : Use all of the hidden units, but to multiply (1-P%)



Batch normalization

- Renormalization
 - Normalize the output within a range of $[0,1]$
 - Normalize the data after convolution, before activation function

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

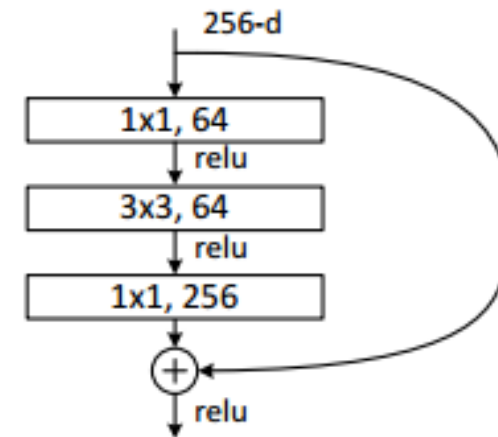
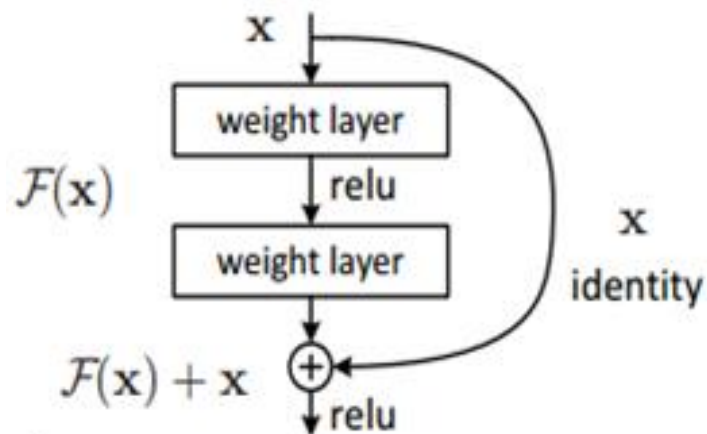
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \quad // \text{ scale and shift}$$

ResNet

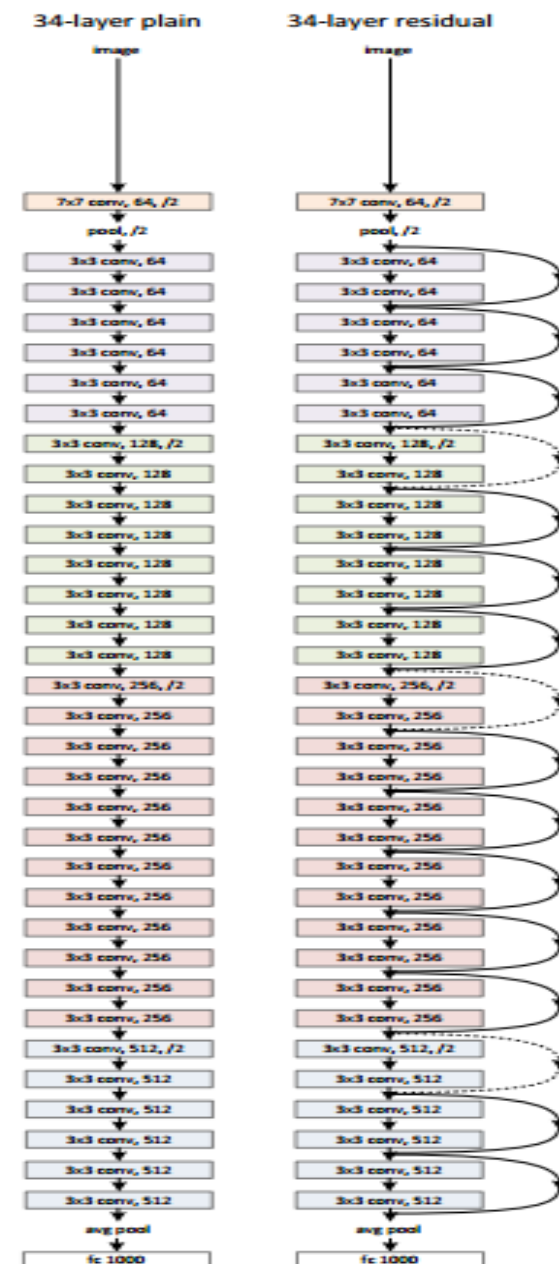
- Residual Net
 - Residual unit and bottleneck
 - 3*3 and 1*1 kernel size convolutions
 - Max pooling replaced by strides
 - Average pooling after last convolution layer
 - Use batch normalization before activation function
- Residual Unit
- bottleneck



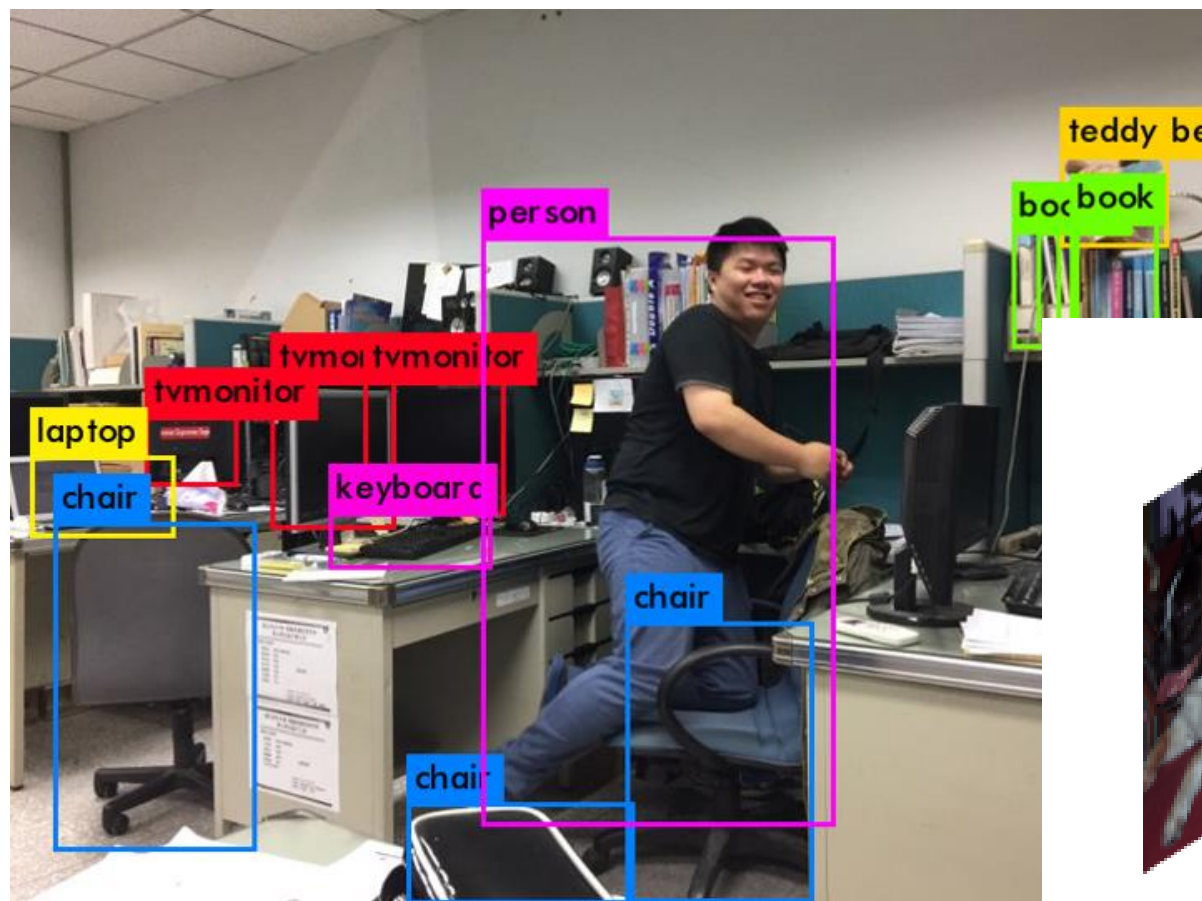
ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Source: [Deep Residual Learning for Image Recognition](#) (CVPR 2016)



Object detection (Yolo V2)



Segmentation

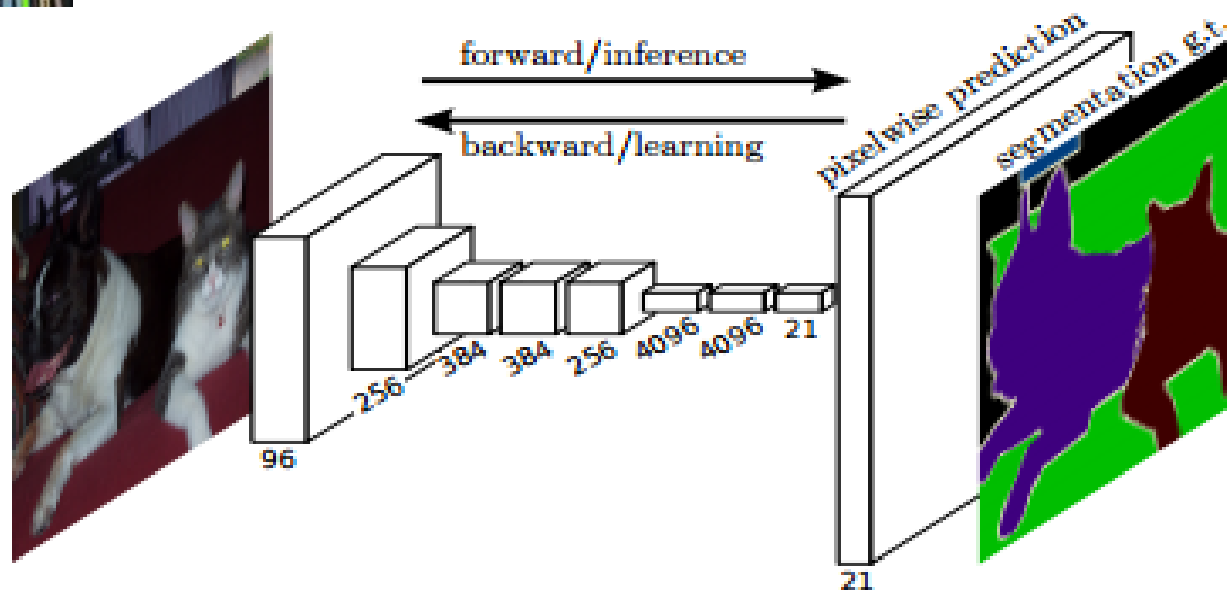


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

Outline

- Machine Learning to Deep learning
- Convolution arithmetic for deep learning
- Some classical deep learning models
 - AlexNet
 - VGG
 - GoogleNet
 - ResNet
- Some applications
 - Segmentation
 - Object detection
 - Generate Data

Segmentation

- Fully Convolutional Network(FCN)
 - Uses convolutions and transposed convolutions
 - Fully-connected layers are replaced by transposed convolutions
 - Pixel-wise prediction

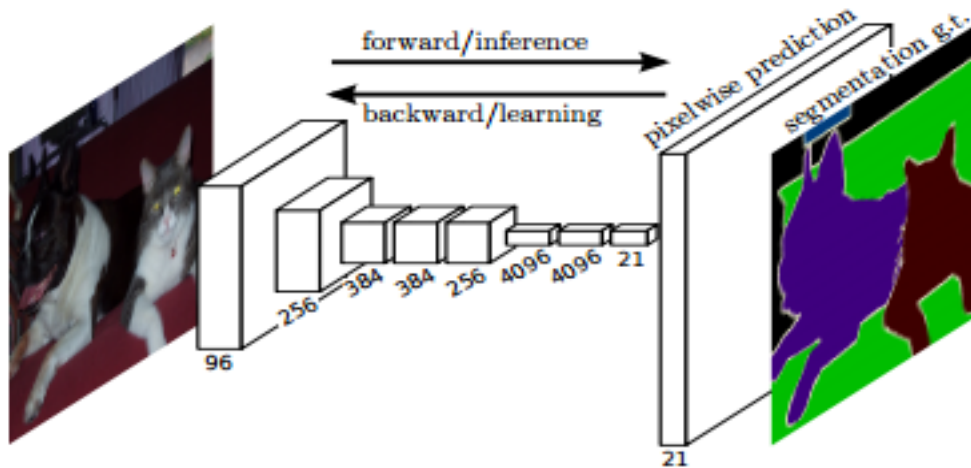


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

Source : [Fully Convolutional Networks for Semantic Segmentation](#)(CVPR 2015)

Segmentation

- Feature Pyramid Network(FPN)
 - Feature hierarchy
 - Bottom-up pathway
 - Feedforward computation of convolutional network
 - Top-down pathway
 - Hallucinates higher resolution features by up-sampling spatially coarser
 - lateral connections
 - Merged with the corresponding bottom-up map by element-wise addition

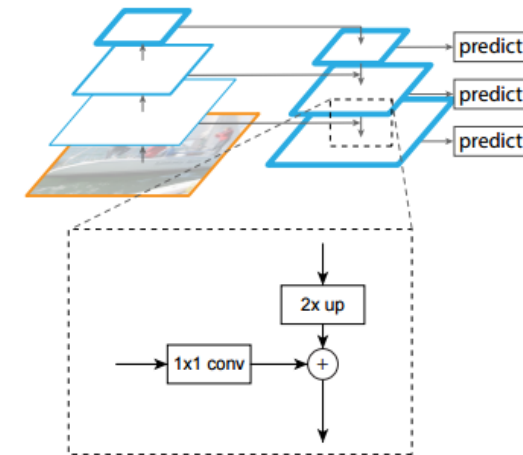
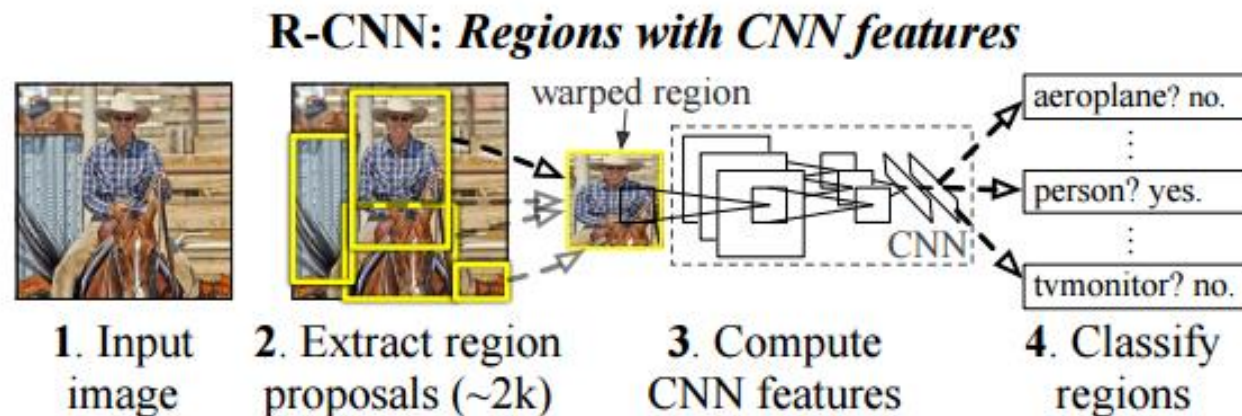


Figure 3. A building block illustrating the lateral connection and the top-down pathway, merged by addition.

Source : [Feature Pyramid Networks for Object Detection](#) (CVPR 2016)

Object detection

- R-CNN(Region proposals CNN)
 - Selective search for finding input $\sim 20K$ candidates
 - CNN for learning feature
 - SVM for classification
 - greedy non-maximum suppression for merging region



Source: [Rich feature hierarchies for accurate object detection and semantic segmentation](#) (CVPR 2014)

RCNN (Regional CNN)

Region Proposal:

Selective search (一種hierarchical similarity merging)從影像提取2000個左右的候選區域

候選區域resize 成固定大小，輸入到CNN，提取CNN最後convolution 層的輸出作為特徵

提取到的CNN特徵輸入到SVM(每個類別用一個SVM做二分類)進行分類

SPP (Spatial Pyramid Pooling)

RCNN 須(1)進行warping 使ROI 固定大小，(2)每個candidate region 運算一次CNN

SPP: 在CNN 之後加一個spatial pyramid pooling
對整張影像計算一次CNN，提取candidate regions 的特徵，放入SPP

SPP 對每一個即使不同大小的candidate region，切成固定格數，計算每個格子特徵，達到相同feature 長度

Object detection



RCNN

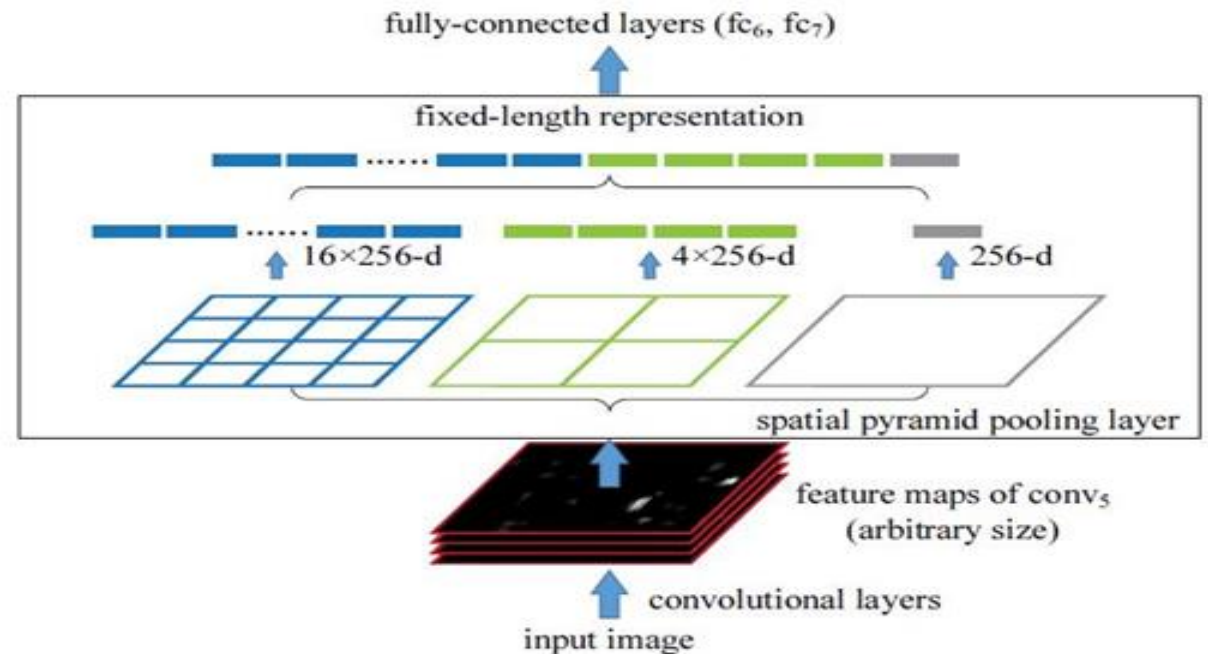


SPPNet



- SPPNet(Spatial Pyramid Pooling)

- Similar with R-CNN
- Fix drawbacks of R-CNN
 - Different input sizes instead of fixed input sizes
- Spatial Pyramid Pooling
 - Divides feature maps into fixed size blocks
 - Choose the **max one** (pooling) in each block
 - Same dimension into fully-connected layers



Spatial Pyramid Pooling model

Fast RCNN

RCNN 須(1)進行warping 使ROI 固定大小，(2)每個candidate region 運算一次CNN

SPP: 在CNN 之後加一個spatial pyramid pooling
對整張影像計算一次CNN，提取candidate region 的特徵，放入SPP



改善RCNN成為：

整張圖進行CNN，在最後的CNN對每一個ROI 加ROI pooling layer成為固定feature 長度
直接加softmax 進行分類

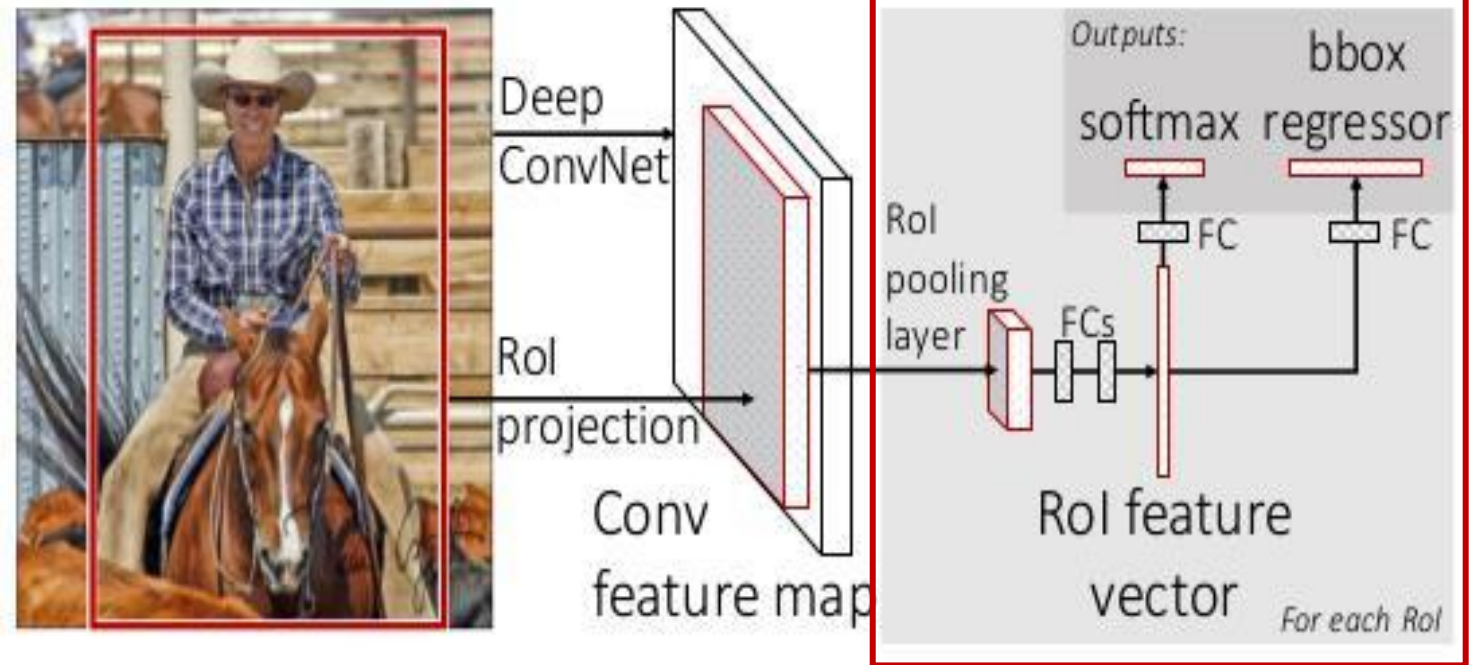
Regressor 直接加入於convolution layer 之後

Object detection

- Fast R-CNN
 - Based on Spatial Pyramid Pooling
 - Improve performance of R-CNN and SPPNet
 - SVM is replaced by fully-connected layers
 - Integrates learning features and classification into a model
 - ROI pooling output feature vector and coordinates input fully-connected layers

Fast R-CNN architecture

Source: [Fast R-CNN](#) (CVPR 2015)



➡ But we still need region proposal, to find out the candidate regions

Faster RCNN

移除selective search 找尋所有candidate regions的步驟

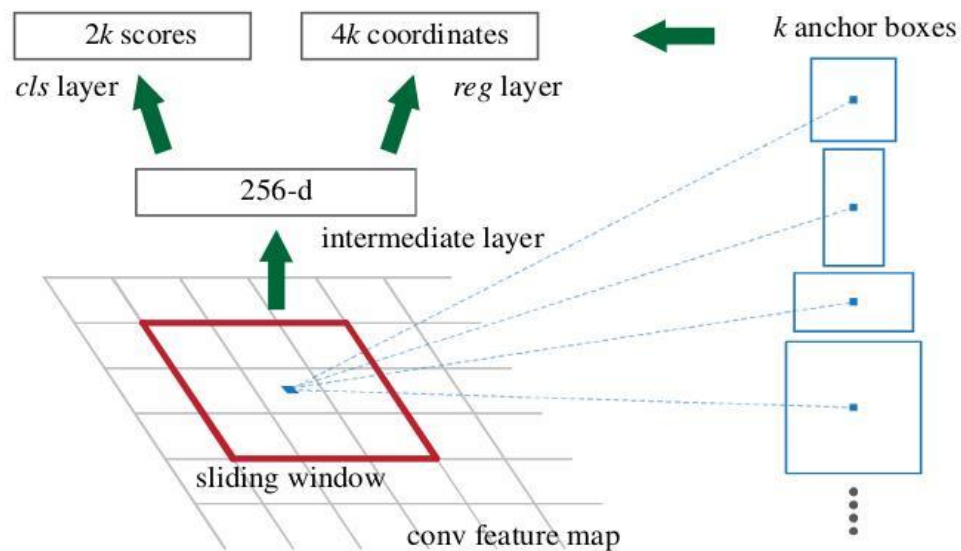
加入region proposal network整合到network中，並加入anchor box 設計成數種固定大小boxes來找candidate regions

整張圖進行CNN，在最後的CNN對每一個ROI 加ROI pooling layer成為固定feature 長度
直接加softmax 進行分類

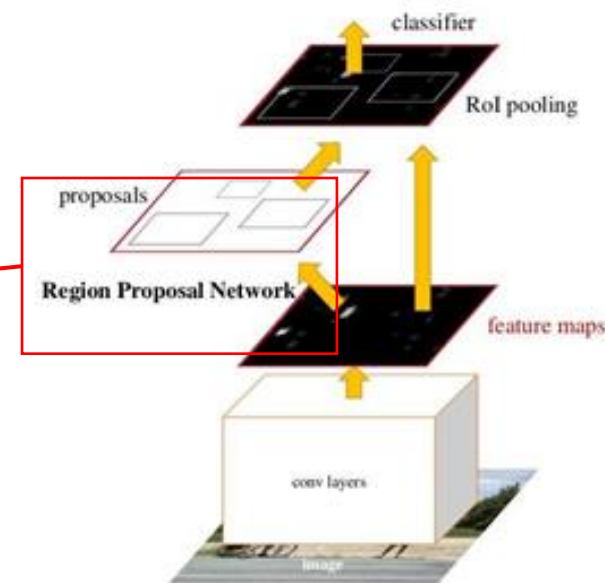
Regressor 直接加入於convolution layer 之後

Object detection

- Faster R-CNN
 - Based on Fast R-CNN
 - Improve performance of Fast R-CNN
 - Selective Search is replaced by Region Proposal Network(RPN)
 - Add RPN between the last convolution layer and ROI pooling layer



RPN architecture

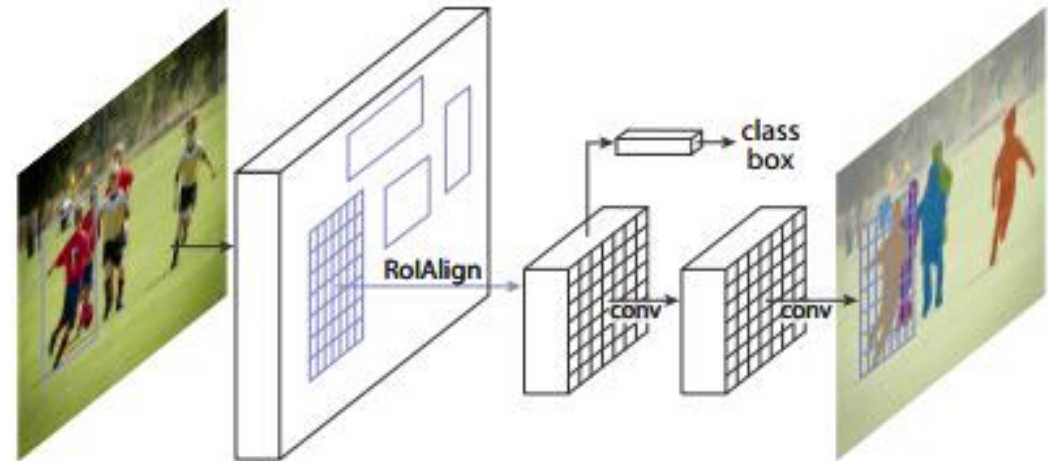


Faster R-CNN architecture

Source: [Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks](#) (CVPR 2016)

Object detection

- Mask R-CNN
 - Based on Faster R-CNN
 - Add a branch for segmentation
 - Fully Convolutional Network(FCN)
 - Feature Pyramid Network(FPN)



Mask R-CNN architecture

以上RCNN系列做法逐一找尋candidate boxes非常耗時間

YOLO born was born to solve the problem

- YOLO was born in 2015
- YOLO v2 was born in 2016 and published in 2017 CVPR
- YOLO v3 was born in March 2018

YOLO v1

屬那個類別才算

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2$$

Bounding box

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2$$

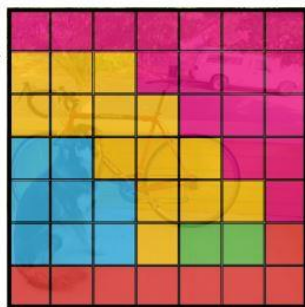
Confidence

$$+ \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Classification

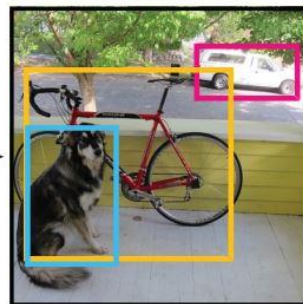
Loss Function

$$B \begin{cases} x, y, w, h \\ \Pr(\text{Object}) * IOU_{\text{pred}}^{\text{truth}} \end{cases}$$



$$\Pr(\text{Class}_i | \text{Object})$$

$$S \times S \times (B * 5 + C)$$

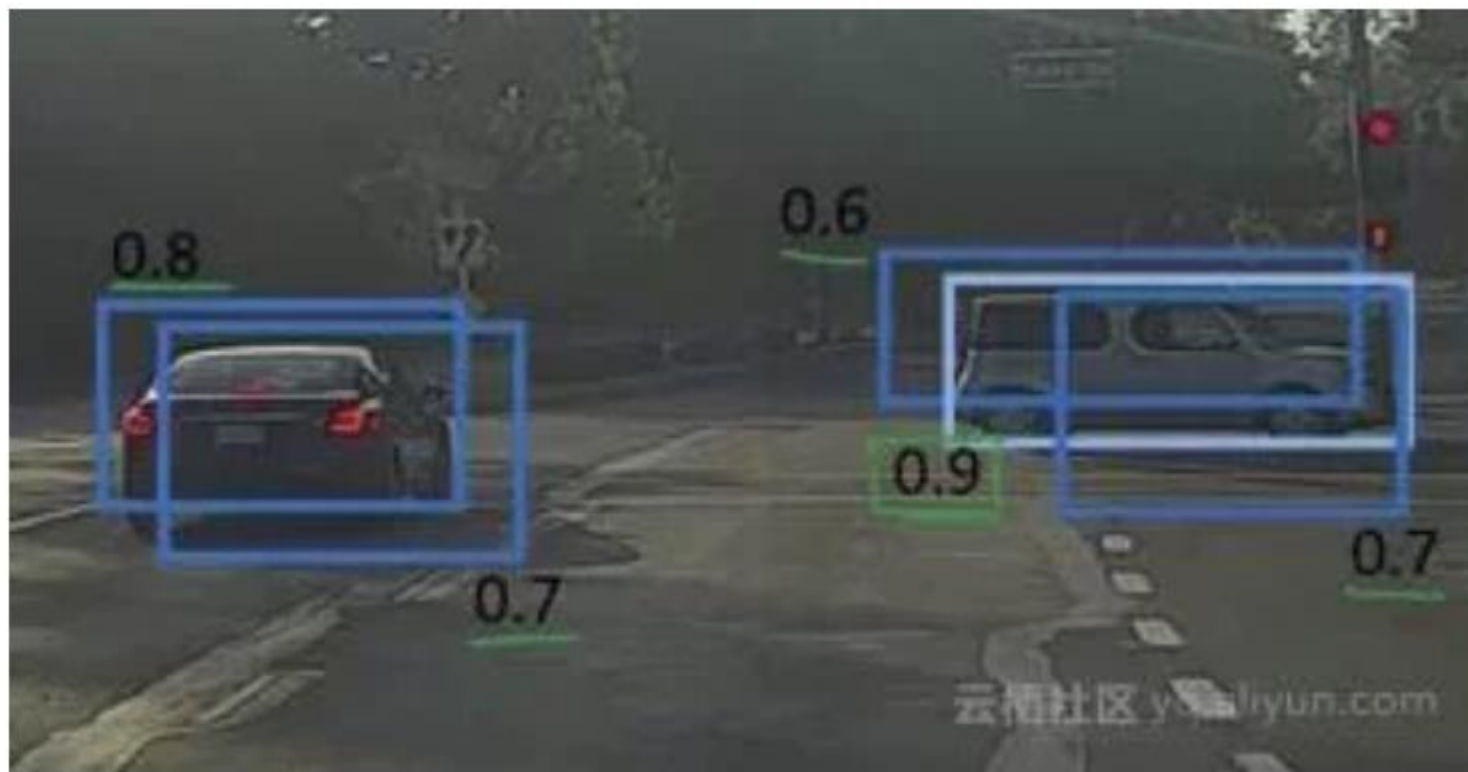


$$\Pr(\text{Class}_i | \text{Object}) * \Pr(\text{Object}) * IOU_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * IOU_{\text{pred}}^{\text{truth}}$$

可能造成數個格子均預測此物體，而有多個bounding boxes 重複

For overlap boxes,

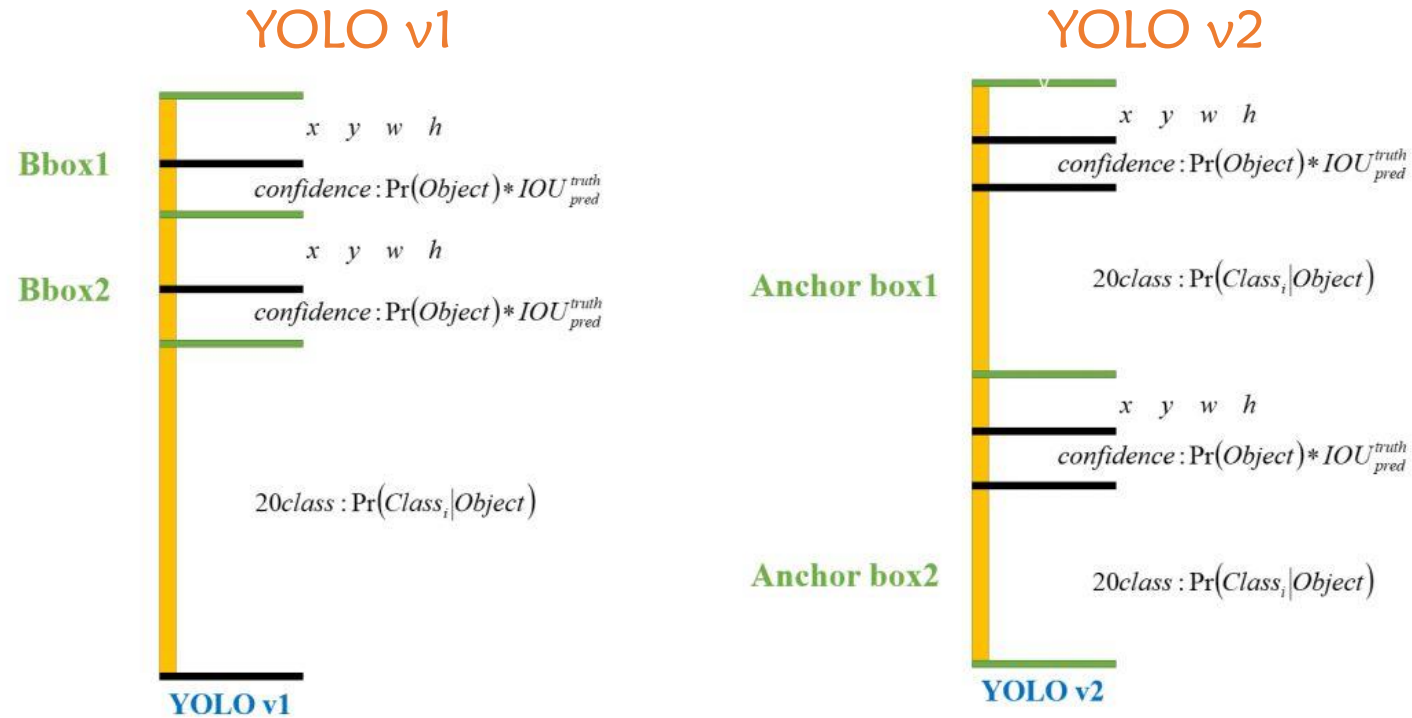
➔ Non Maximum Suppression to select one candidate



不能有重疊物體

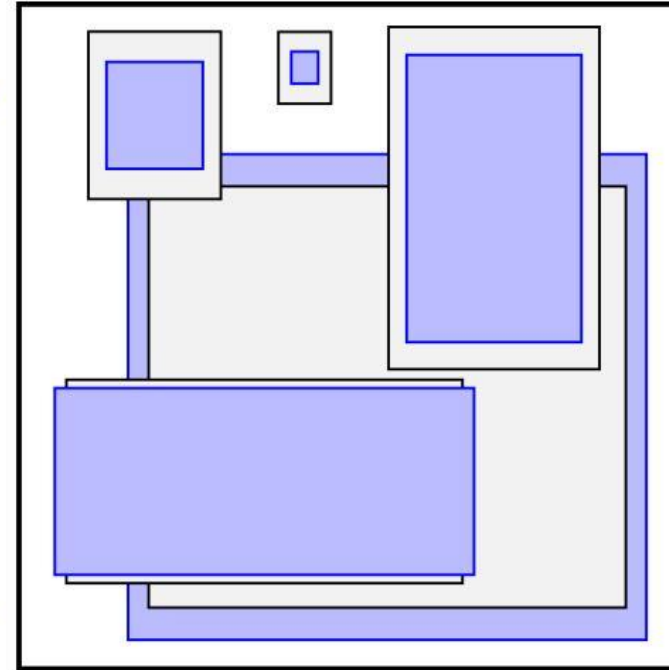
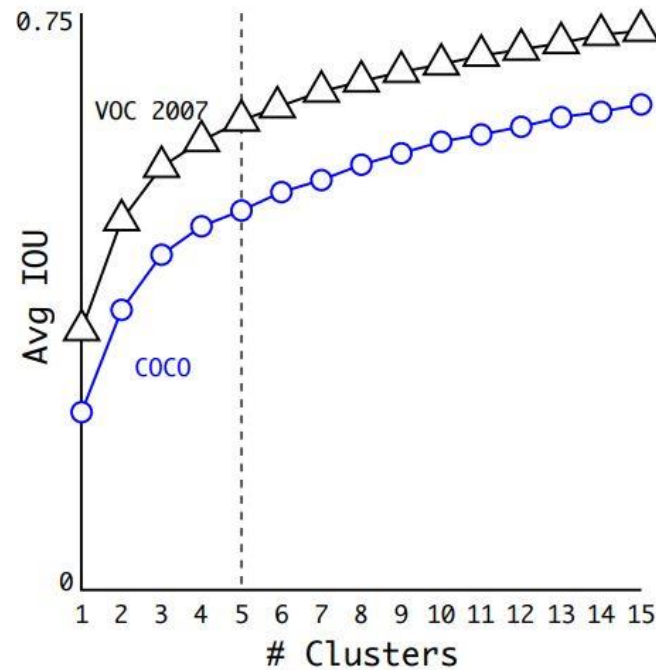
YOLO v2

- Using anchor box to replace bounding box



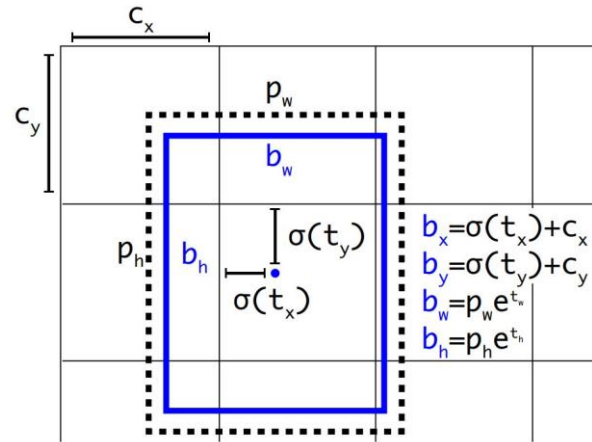
YOLO v2

- Using K-mean to decide how many kind of anchor box size



YOLO v2

- Learning to adjust the anchor box to become a bounding box



$$\Pr(Object) * IOU(b, Object) = \sigma(t_o)$$

YOLO v2

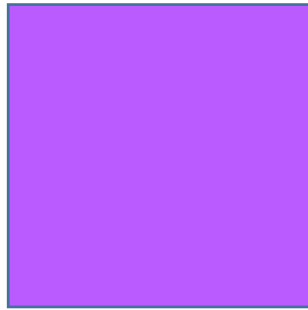
- Remove the fully connected layer to finish multi-scale training

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	78.6	40

YOLO v3

- YOLOv3 predicts an objectness score for each bounding box using logistic regression

Score=1



Bounding box
overlap ground
truth

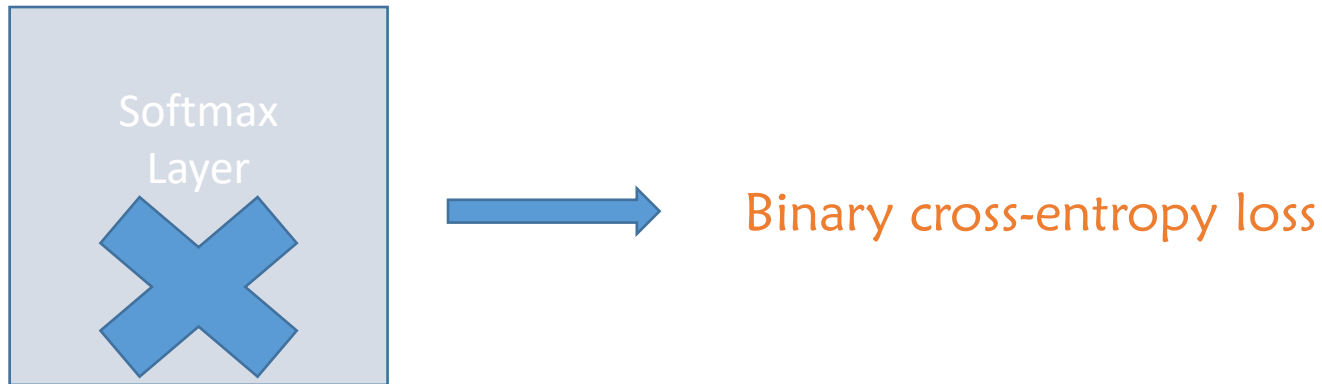
Score too low, be ignored



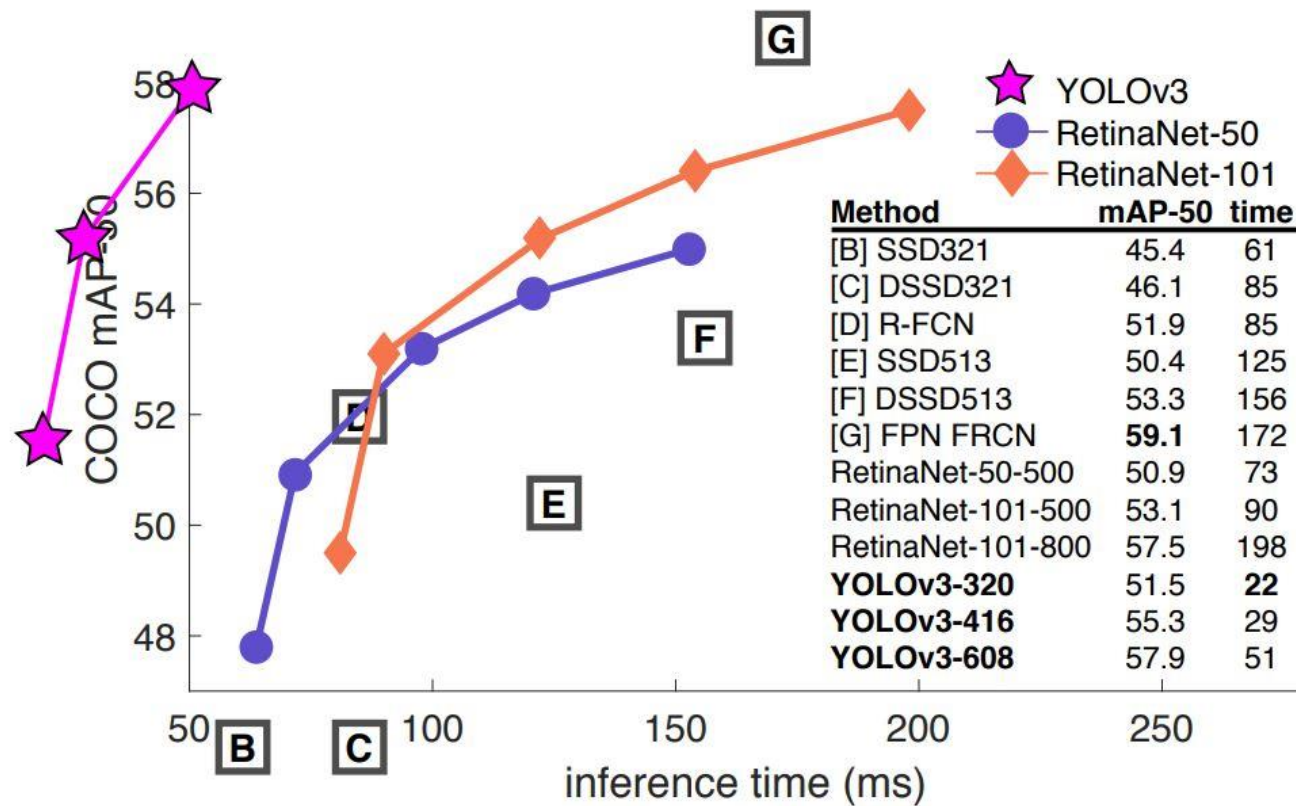
Bounding box
can't overlap
ground truth

YOLO v3

- Take out softmax, using binary cross-entropy loss for the class predictions

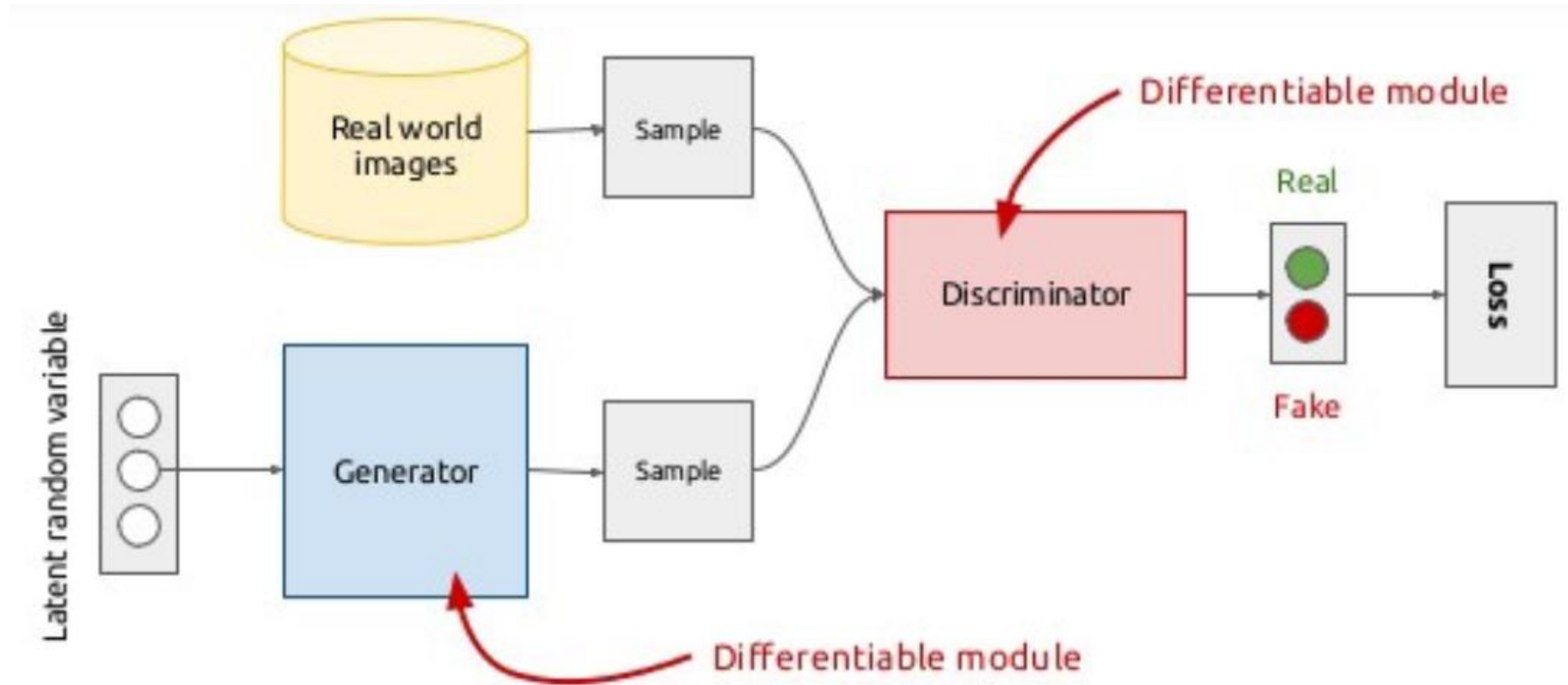


YOLO v3



Generative Adversarial Network (GAN)

- Two minimax-player game
 - Generative model $G(z)$
 - Discriminate model $D(x)$



- Loss function

$$\min_G \max_D V(D, G)$$

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

Generate Data

- Generative Adversarial Network(GAN)

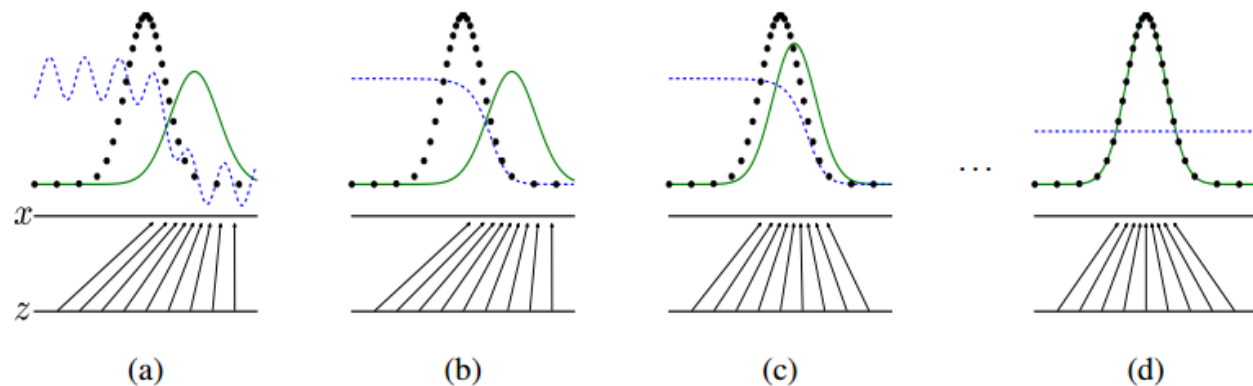
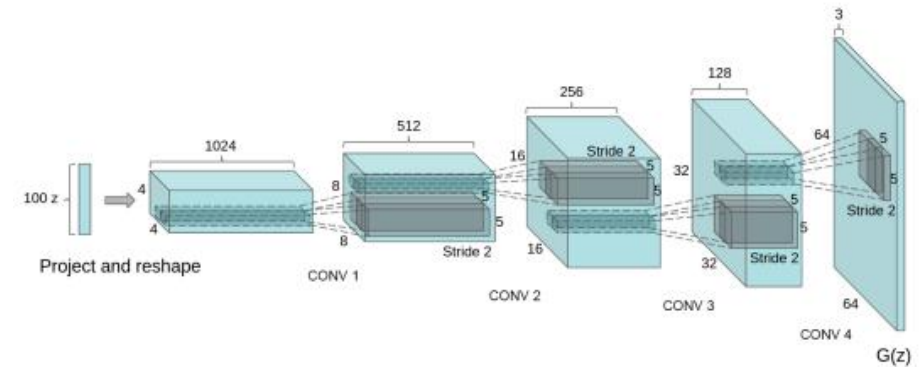


Figure 1: Generative adversarial nets are trained by simultaneously updating the discriminative distribution (D , blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) p_x from those of the generative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of x . The upward arrows show how the mapping $x = G(z)$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g . (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}$. (c) After an update to G , gradient of D has guided $G(z)$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(x) = \frac{1}{2}$.

Generate Data

- Deep Convolutional Generative Adversarial Network(GAN)
 - Replace any pooling layers with
 - Stride convolutions (discriminator)
 - Transposed convolutions (generator).
 - batch normalize
 - activation
 - Generator : ReLU (output :Tanh)
 - Discriminator : leaky ReLU(output is sigmoid)



Generator model

Reference

- Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).
- Activation function – Wikipedia -https://en.wikipedia.org/wiki/Activation_function
- A. Krizhevsky, I. Sutskever, G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *NIPS*, 2012.
- Karen Simonyan, Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *ICLR*, 2015.
- Szegedy, Christian, et al. "Going deeper with convolutions." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015.
- Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167* (2015).
- He, K., Zhang, X., Ren, S., Sun, J, "Deep Residual Learning for Image Recognition," *CVPR*, 2016.
- Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik UC Berkeley, "Rich feature hierarchies for accurate object detection and semantic segmentation" *CVPR*, 2014.
- J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, A. W. M. Smeulders, "Selective Search for Object Recognition" *IJCV*, 2013.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition" *ECCV*, 2014.

Reference

- R. Girshick, "*Fast R-CNN*," in IEEE International Conference on Computer Vision (ICCV), 2015.
- . S. Ren, K. He, R. Girshick, and J. Sun, "*Faster R-CNN: Towards real-time object detection with region proposal networks*," In Advance in Neural Information Processing Systems (NIPS), 2015.
- M. D. Zeiler and R. Fergus, "*Visualizing and understanding convolutional neural networks*," in European Conference on Computer Vision (ECCV), 2014.
- K. He, G. Gkioxari, P. Dollár, and R. Girshick, "*Mask R-CNN*." arXiv preprint arXiv:1703.06870, 2017.
- T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "*Feature pyramid networks for object detection*," arXiv:1612.03144, 2016.
- S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "*Aggregated residual transformations for deep neural networks*," arXiv:1611.05431, 2016.
- Jonathan Long, Evan Shelhamer, Trevor Darrel, "Fully Convolutional Networks for Semantic Segmentation", CVPR, 2015.
- Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.
- Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." *arXiv preprint arXiv:1511.06434* (2015).