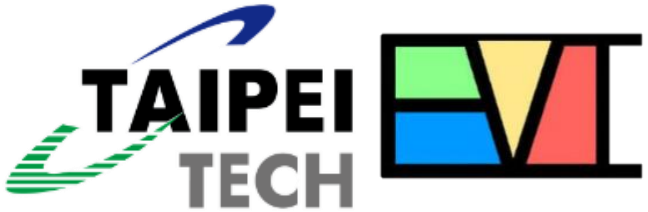


Embedded Vision Intelligent Laboratory

嵌入式智慧影像分析與實境界面
Fall 2021

Instructor：Yen-Lin Chen(陳彥霖), Ph.D.

Professor

Dept. Computer Science and Information Engineering

National Taipei University of Technology

Lecture 2

影像識別深度學習網路模型及 Yolo v4

使用深度學習模型辨識物件

行車辨識

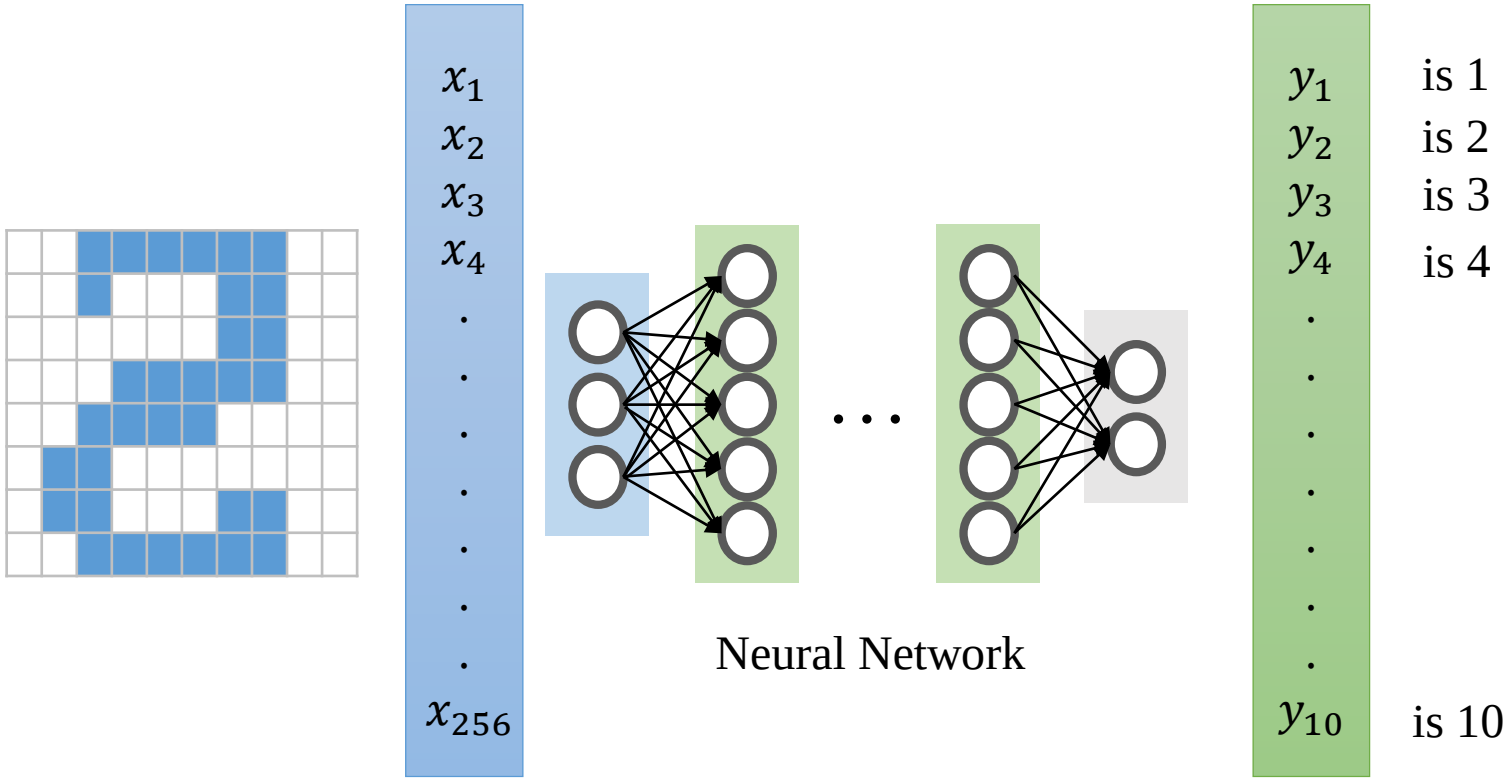
- 在科技發達的現代，交通工具發展日新月異，也增加交通事故的發生。因此為了行車安全，政府、車商甚至是學術單位紛紛投入於路測裝置、行車紀錄以及車輛辨識的領域。
- 早期，車輛辨識是以影像處理的方式萃取特徵進行比對來抓取影像中的車輛位置，但其辨識較慢、辨識成功率也有一定瓶頸。
- 隨著機器學習演算法的開發與相對應的硬體發展，目前的機器學習模型甚至可達到real time（30 fps以上）的辨識速度，且辨識成功率較以往影像處理的方式更高。
- 在目前機器學習影像辨識的成熟發展下，已有開發出車輛軌跡的預測系統、車輛上的路面物件及燈號號誌辨識系統等。

智慧工廠

- 隨著工業4.0產業轉型，許多工廠從原本的工人產線走向機器自動化，人力資源在新興工廠中所佔的成本比例越來越少，傳統工廠也紛紛向智慧工廠的目標推進。
- 為了達到機器自動化，工廠需要有得以辨識物件的軟硬體設備，如產品辨識、晶片錯誤辨識等等，來協助產線分辨產品，進而淘汰有瑕疵的產品。
- 使用機器學習的影像辨識演算法，能夠讓產線的生產速度提高，也能降低傳統工廠的人力資源，大幅提高整體效率。

基礎類神經網路原理解說

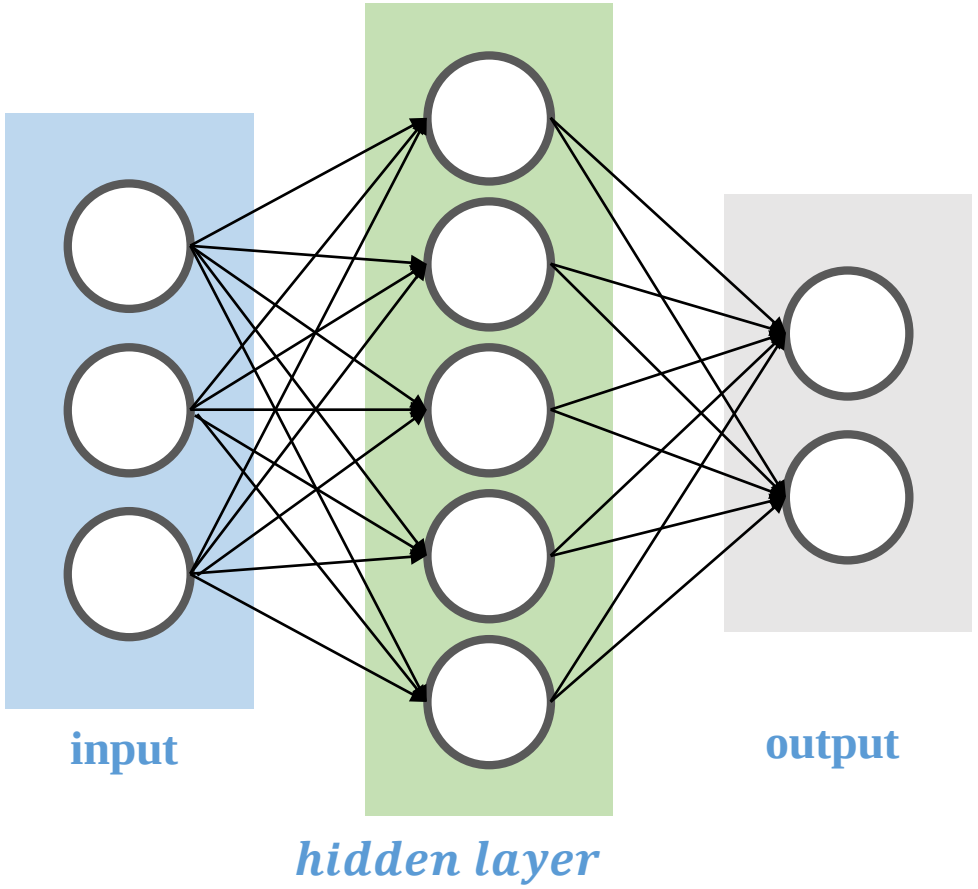
A Brief Example of Neural Networks



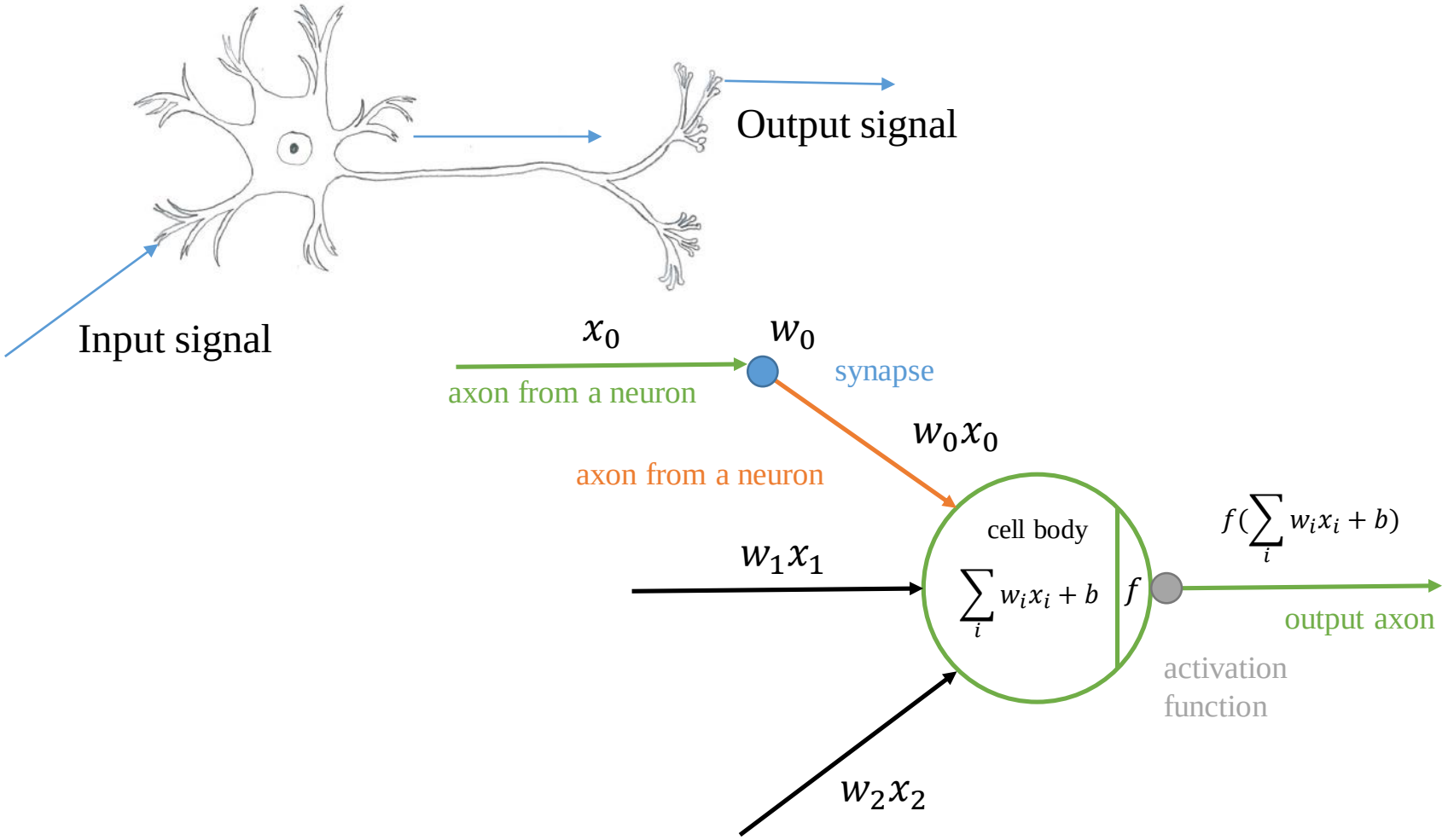
Input: 256-dim vector

Output: 10-dim vector

Basic Artificial Neural Network

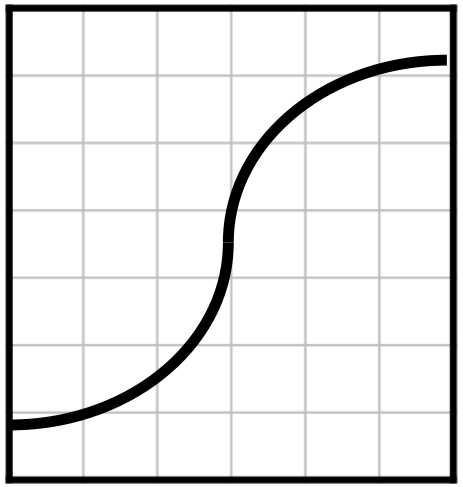


Neuron Design

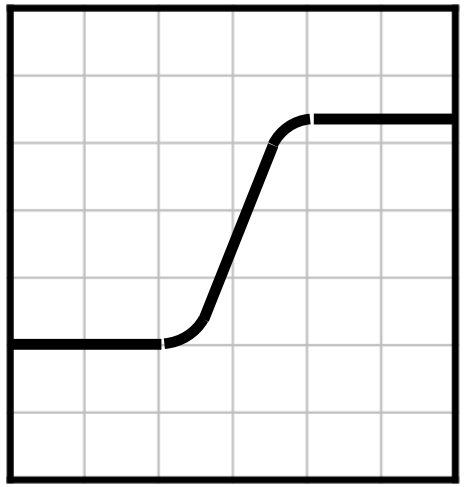


Activation Function for Neurons

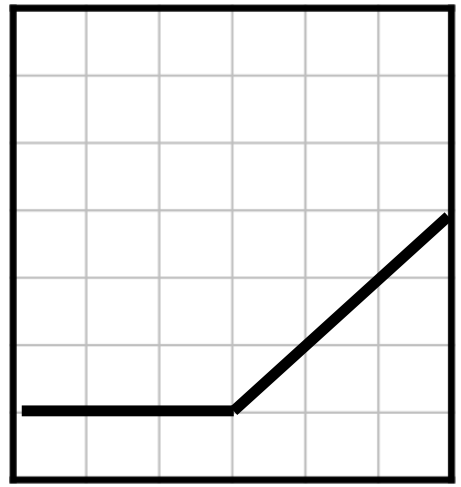
graphs



Sigmoid



tanh



ReLU

equations

$$\sigma(x) = \frac{1}{(1 + e^{-x})}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = 2\sigma(2x) - 1$$

$$ReLU(x) = \max(0, x)$$

ranges

$$0 < \sigma(x) < 1$$

$$-1 < \tanh(x) < 1$$

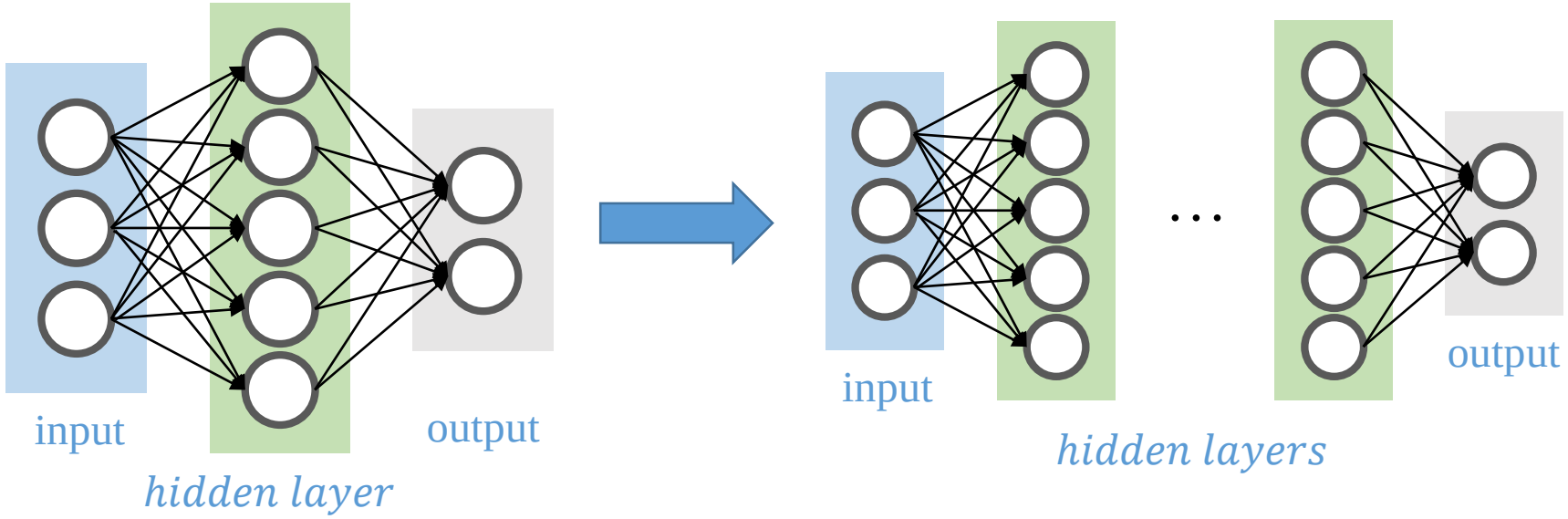
$$0 < ReLU(x) < 1$$

How to Let NNs Work Better

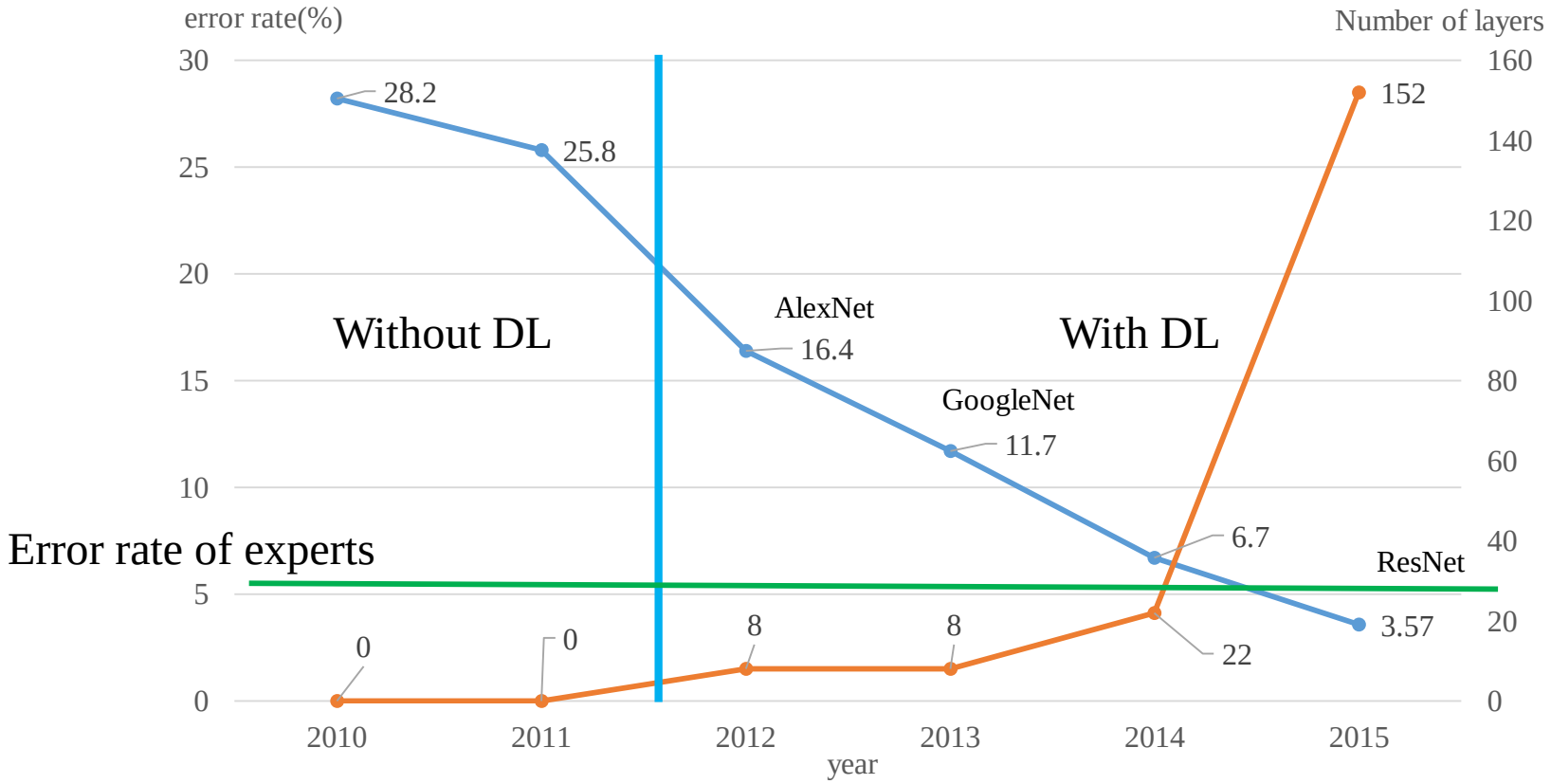
- Something you need to consider first
  - What activation function should I use
  - How many neurons do I need
  - How many layers should I need
- More advanced
  - What kind of network structure should I use (CNN, RNN, GAN, etc.)

Deep Neural Network

- Deep: more hidden layers



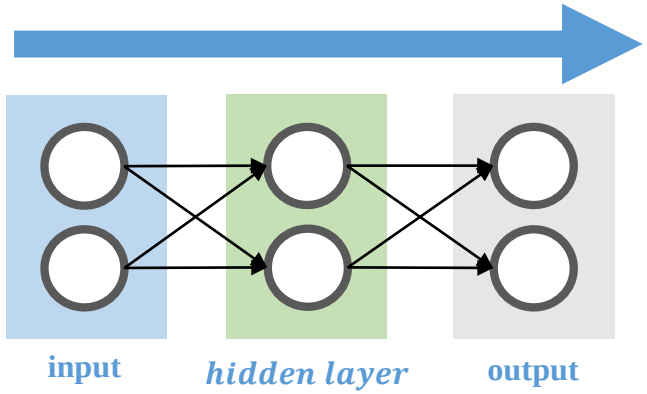
The Deeper the Better?



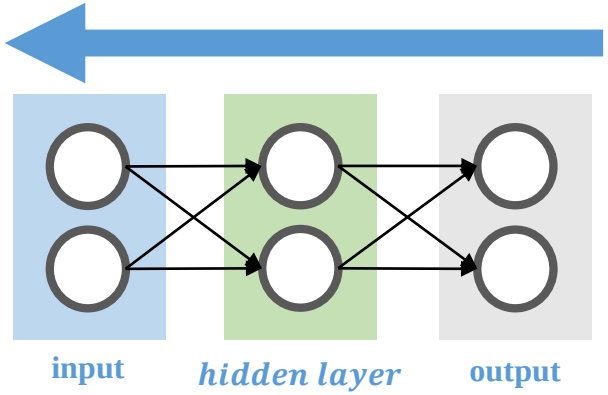
Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei.
ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

How to Train the NNs

- Input some examples
- Calculate the output
  - Forward propagation

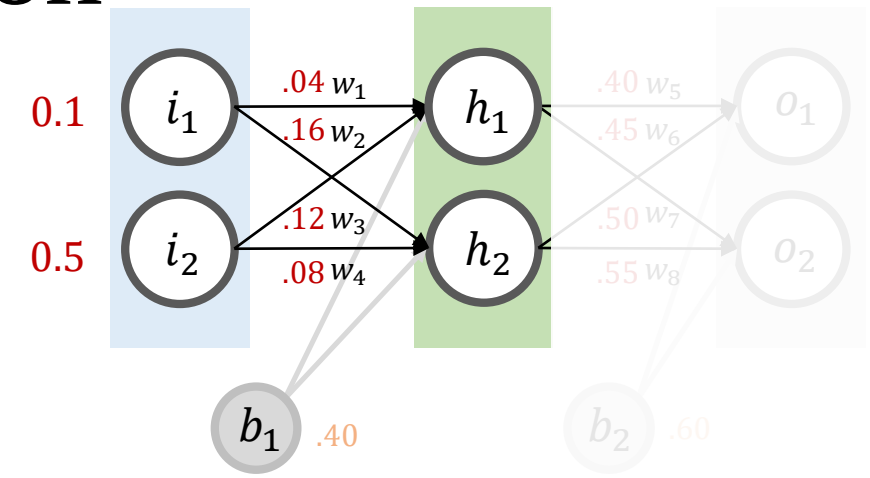


- Measure the errors between the outputs and answers
- Update the weights in NN
  - Back propagation



Forward Propagation

$$\sum_i w_i x_i + b$$



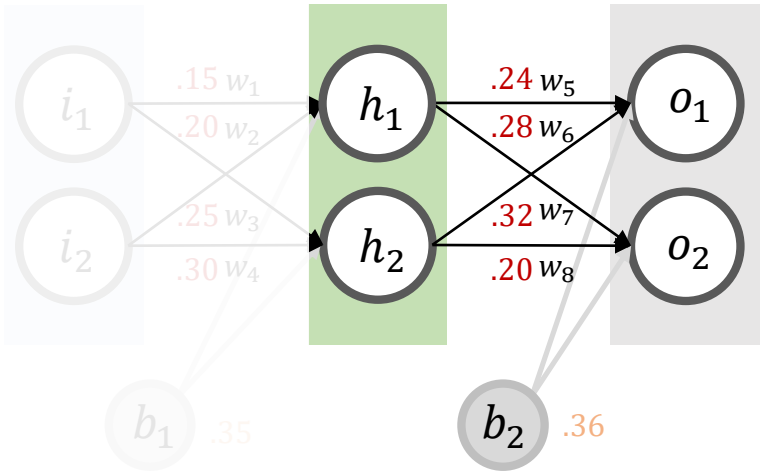
$$net_{h_1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$
$$= 0.04 * 0.1 + 0.12 * 0.5 + 0.40 * 1$$
$$= 0.464$$

$$out_{h_1} = \frac{1}{1 + e^{-net_{h_1}}} = \frac{1}{1 + e^{-0.464}}$$
$$= 0.613962657$$

$$out_{h_2} = 0.611114647$$

Forward Propagation

$$\sum_i w_i x_i + b$$



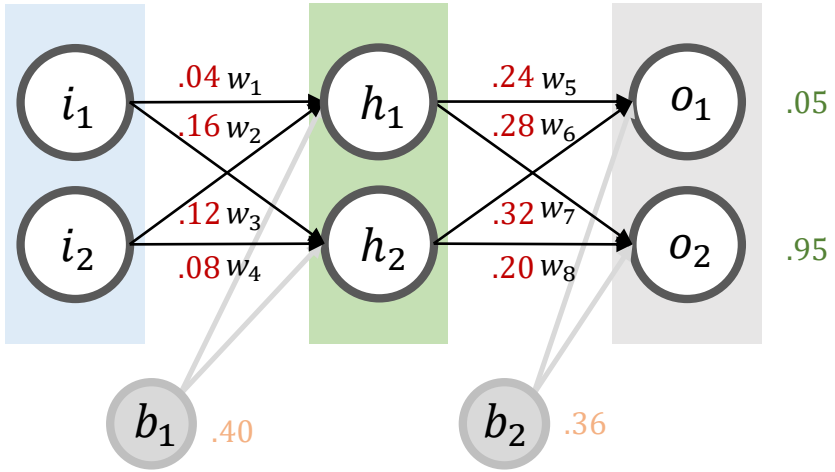
$$net_{o_1} = w_5 * out_{h_1} + w_6 * out_{h_2} + b_2 * 1$$
$$= 0.24 * 0.613962657 + 0.32 * 0.611114647 + 0.36 * 1$$
$$= 0.702907725$$

$$out_{o_1} = \frac{1}{1 + e^{-net_{o_1}}} = \frac{1}{1 + e^{-0.702907725}}$$
$$= 0.668832137$$

$$out_{o_2} = 0.657941101$$

The Errors of Outputs

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$



$$E_{o_1} = \frac{1}{2}(target - output)^2$$
$$= \frac{1}{2}(0.05 - 0.668832137)^2$$
$$= 0.191476607$$

The total error for the neural network is the sum of these errors:

$$E_{total} = E_{o_1} + E_{o_2} = 0.191476607 + 0.042649200 = 0.234125807$$

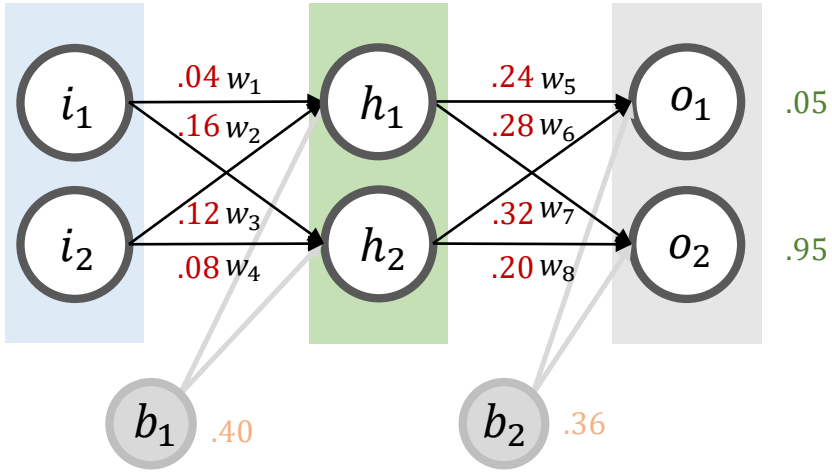
Updating Weights

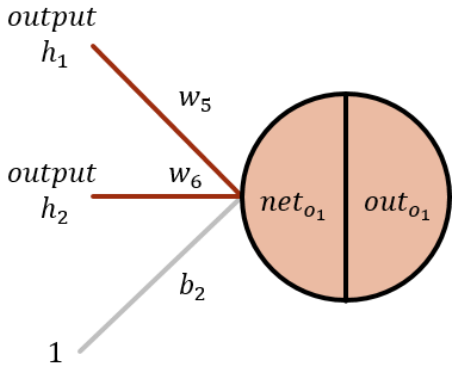
- The only layer with the answer is the output layer
  - The only layer we can know the errors
- We need to update the weights from the output layer to hidden layers

- Solution: Back-propagation

Backward Propagation to Update w5

- Output layer





$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

$$E_{total} = \frac{1}{2}(target - out_{o_1})^2 + \frac{1}{2}(target - out_{o_2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o_1}} = 2 * \frac{1}{2}\left(target_{o_1} - out_{o_1}\right)^{2-1} * (-1) + 0$$

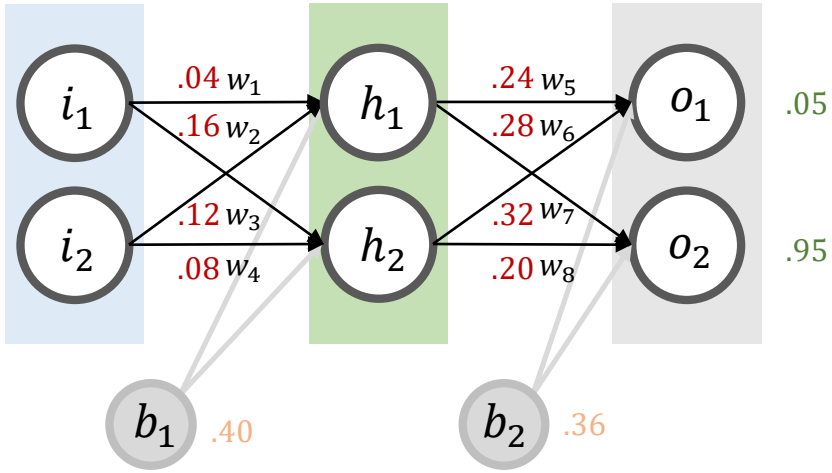
$$= -\left(target_{o_1} - out_{o_1}\right)$$

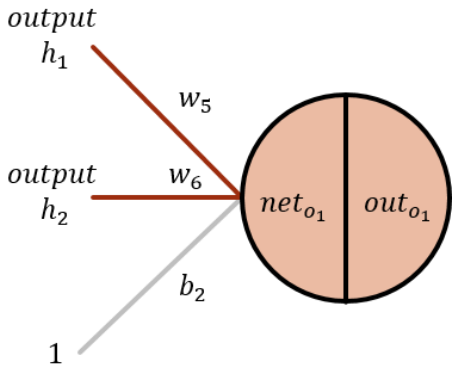
$$= -(0.05 - 0.668832137)$$

$$= 0.618832137$$

Backward Propagation to Update w5

- Output layer





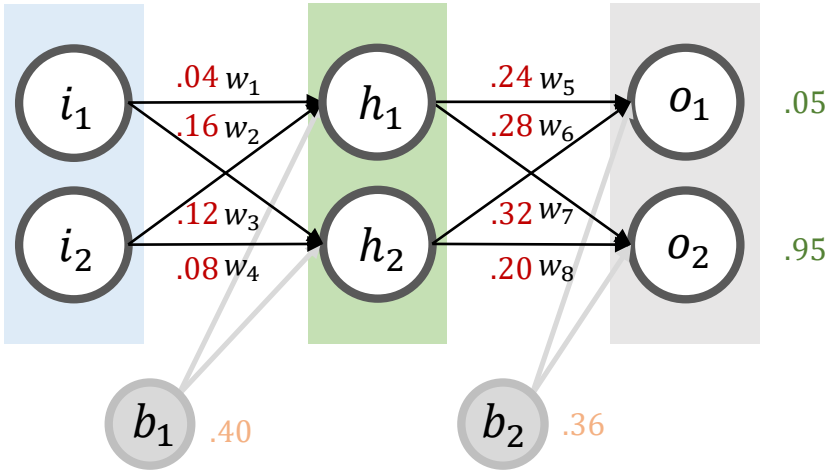
$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

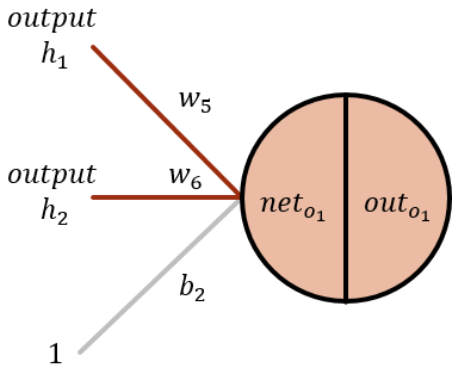
$$out_{o_1} = \frac{1}{1 + e^{-net_{o_1}}}$$

$$\frac{\partial out_{o_1}}{\partial net_{o_1}} = out_{o_1}(1 - out_{o_1})$$
$$= 0.668832137(1 - 0.668832137)$$
$$= 0.337664274$$

Backward Propagation to Update w5

- Output layer





$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

$$net_{o_1} = w_5 * out_{h_1} + w_6 * out_{h_2} + b_2 * 1$$

$$\frac{\partial net_{o_1}}{\partial w_5} = 1 * out_{h_1} * w_5^{1-1} + 0 + 0$$
$$= out_{h_1}$$
$$= 0.613962657$$

Backward Propagation to Update w5

- Output layer

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.618832137 * 0.337664274 * 0.613962657 = 0.128292105$$

Learning rate

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5}$$

$$= 0.15 - 0.5 * 0.128292105$$

$$= 0.0858539475$$

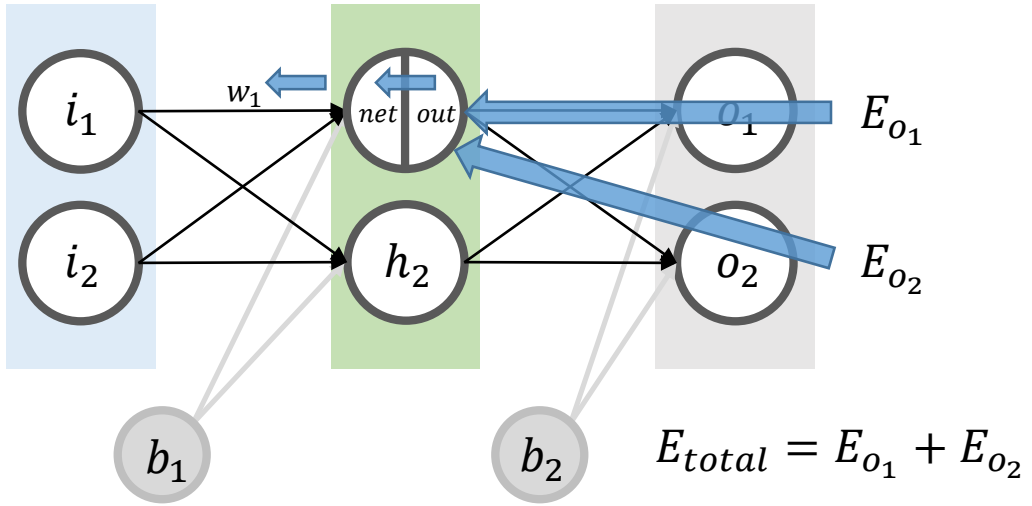
$$w_6^+ = 0.238117666$$

$$w_7^+ = 0.300177638$$

$$w_8^+ = 0.300084039$$

Backward Propagation to Update w1

- Hidden layer



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$

Backward Propagation to Update w1

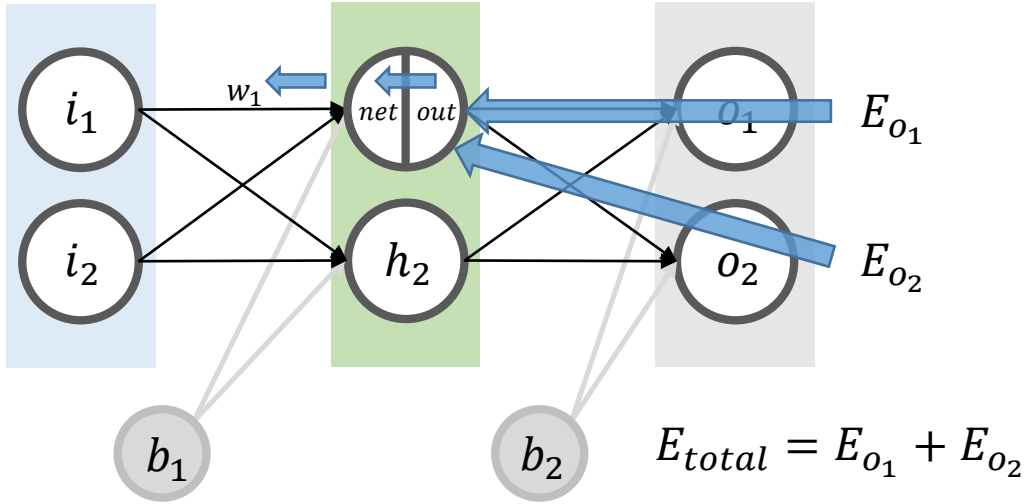
- Hidden layer

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial net_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial net_{o_1}} = \frac{\partial E_{o_1}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}}$$



$$\frac{\partial E_{o_1}}{\partial net_{o_1}} = \frac{\partial E_{o_1}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} = 0.618832137 * 0.337664274 = 0.208957504$$

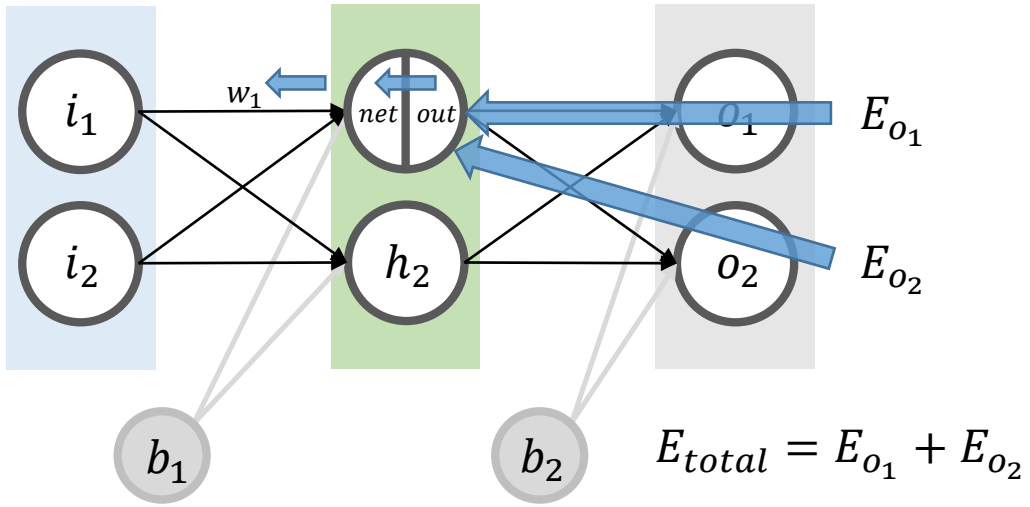
Backward Propagation to Update w1

- Hidden layer

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial net_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial out_{h_1}}$$



$$net_{o_1} = w_5 * out_{h_1} + w_6 * out_{h_2} + b_2 * 1$$

$$\frac{\partial net_{o_1}}{\partial out_{h_1}} = w_5 = 0.24$$

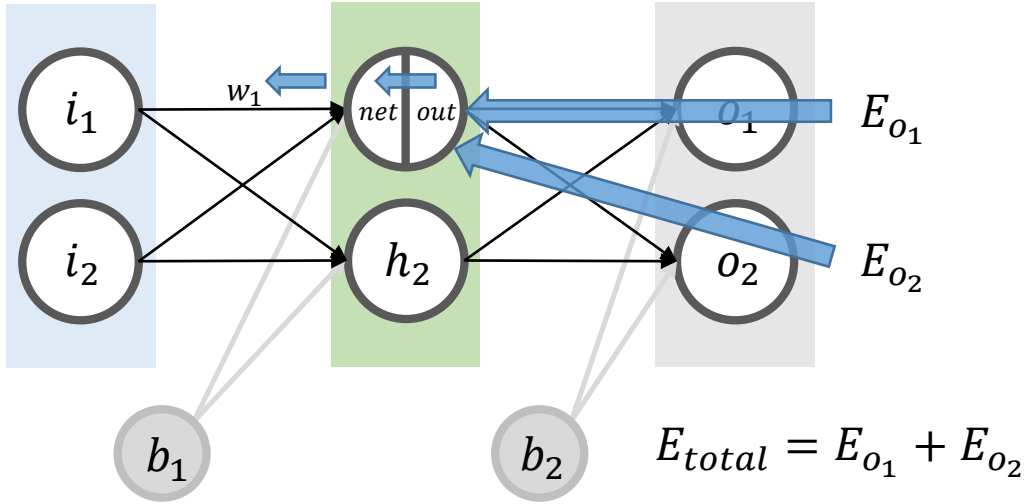
$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial out_{h_1}} = 0.208957504 * 0.24 = 0.050149801$$

Backward Propagation to Update w1

- Hidden layer

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial net_{h_1}}$$



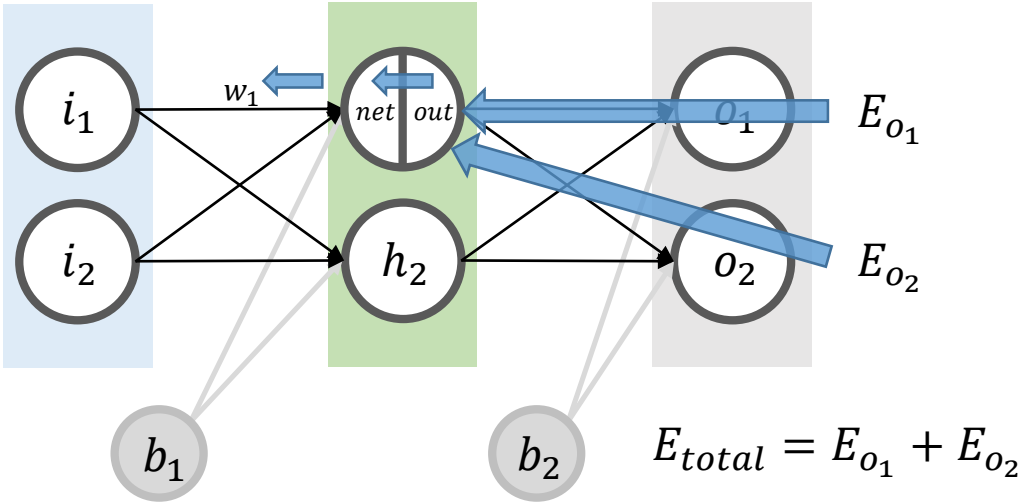
$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = 0.050149801, \qquad \frac{\partial E_{o_2}}{\partial out_{h_1}} = -0.018404176$$

$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial net_{h_1}} = 0.050149801 + (-0.018404176\ ) = 0.031745625$$

Backward Propagation to Update w1

- Hidden layer

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$



$$out_{h_1} = \frac{1}{1 + e^{-net_{h_1}}}$$

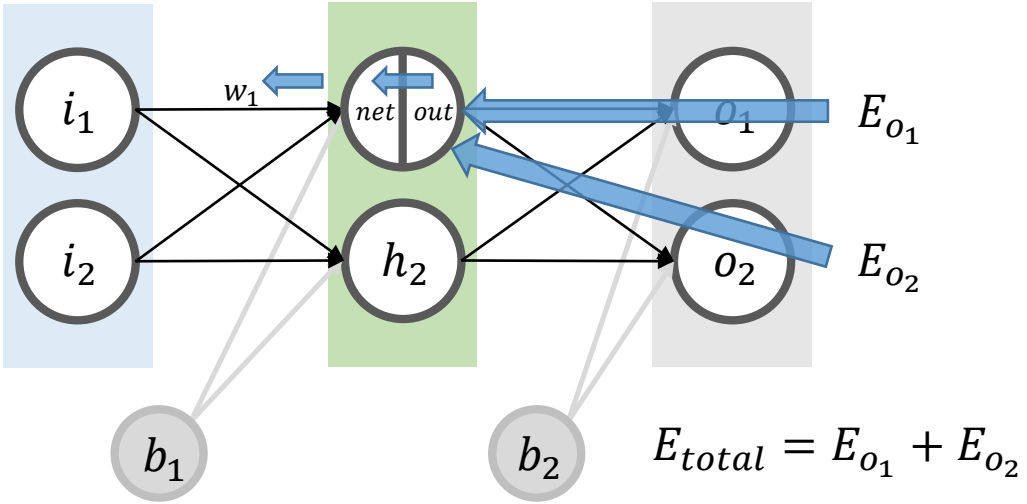
$$\frac{\partial out_{h_1}}{\partial net_{h_1}} = out_{h_1}(1 - out_{h_1}) = 0.613962657 * (1 - 0.613962657\ )$$

$$= 0.237012513$$

Backward Propagation to Update w1

- Hidden layer

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$



$$net_{h_1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h_1}}{\partial w_1} = i_1 = 0.1$$

Backward Propagation to Update w1

- Hidden layer

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.031745625 * 0.237012513 * 0.1 = 0.000752411$$

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1}$$
$$= 0.04 - 0.5 * 0.000752411$$
$$= 0.039623795$$

$$w_2^+ = 0.118118973$$

$$w_3^+ = 0.159635010$$

$$w_4^+ = 0.078175051$$

簡介

- 每次丟了東西，我們都希望有一種方法能快速定位出失物。現在，目標檢測演算法或許能做到。目標檢測的用途遍佈多個行業，從安防監控，到智慧城市中的即時交通監測。簡單來說，這些技術背後都是強大的深度學習演算法。
- 卷積神經網絡 (Convolutional Neural Network) 在電腦視覺中佔有非常重要的地位，在影像辨識上的精準度可以達到超過人類的水準，是目前深度學習發展的一大主力
- 本次課程將要來介紹 CNN 的經典模型 LeNet、AlexNet、VGG，以及由CNN本身擴展的RCNN、Fast RCNN和Faster RCNN等神經網路。

深度學習框架

Deep Learning Library: PyTorch

- PyTorch是一個Open Source的Python機器學習庫，底層由C++實現，應用於人工智慧領域，如自然語言處理等。它最初由Facebook的人工智慧研究團隊開發。
- PyTorch主要有兩大特徵：
  - 類似於NumPy的張量計算，可使用GPU加速
  - 基於帶自動微分系統的深度神經網路
- PyTorch包括torch.nn、torch.optim等子模組。

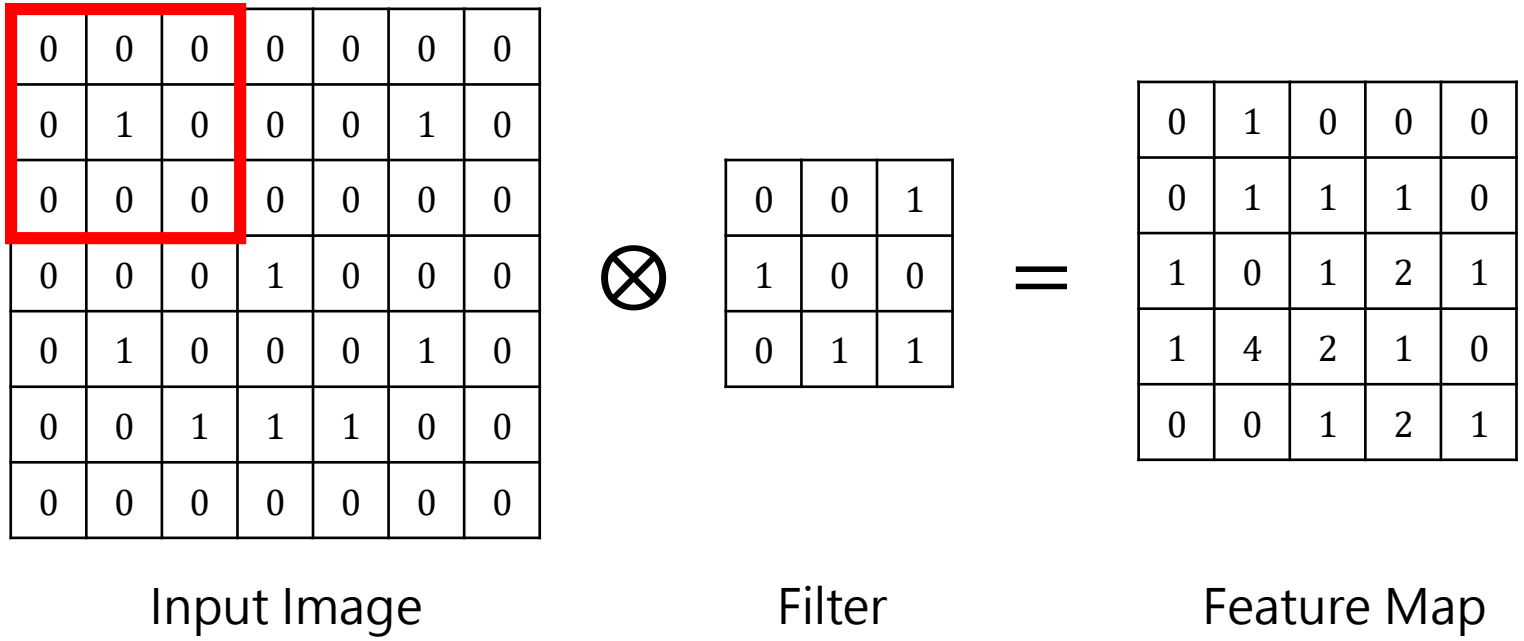
Deep Learning Library: Tensorflow

- TensorFlow是一個Open Source軟體庫，用於各種感知和語言理解任務的機器學習。目前被50個團隊用於研究和生產許多Google商業產品，如語音辨識、Gmail、Google 相簿和搜尋等，其中許多產品曾使用過其前任軟體DistBelief。
- TensorFlow最初由Google大腦團隊開發，用於Google的研究和生產，於2015年11月9日在Apache Open Source License 2.0下發布。

卷積神經網絡 CNN 經典模型

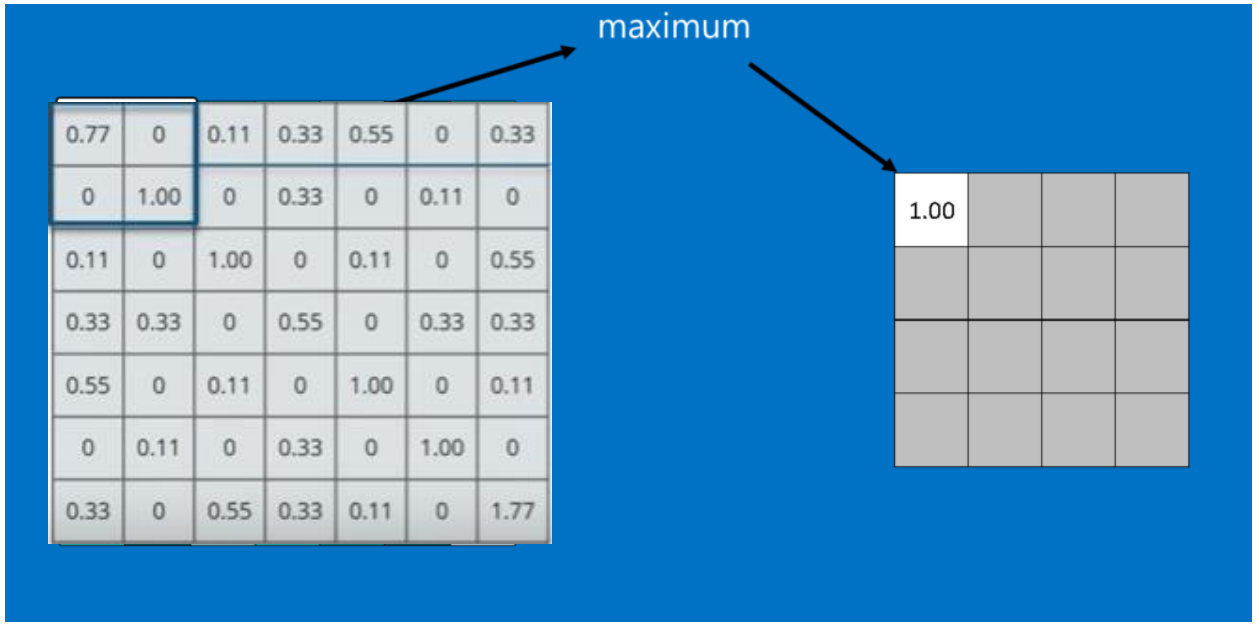
Convolution layer

- Convolution means **multiplication** and **add**
- Main purpose is to extract feature



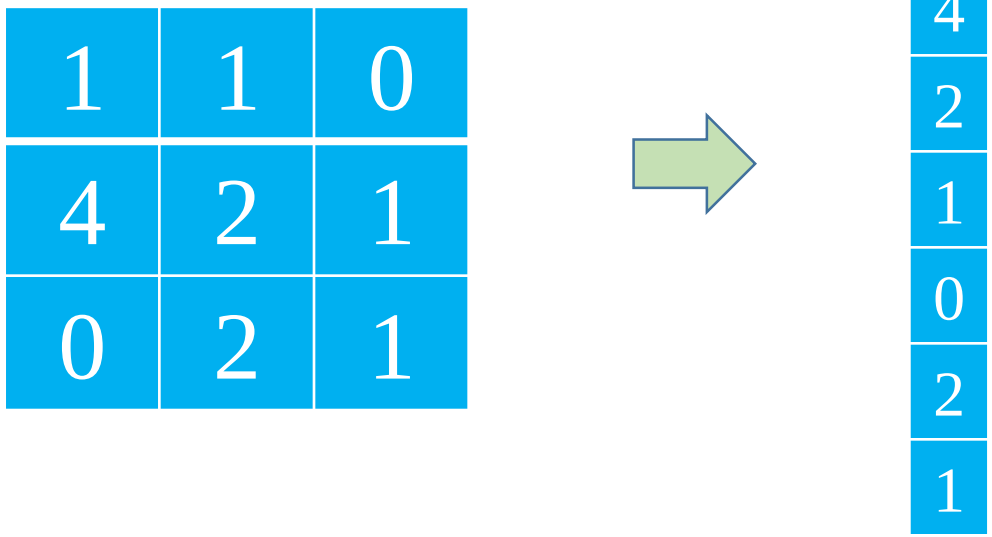
Pooling layer

- Max Pooling: get the max value in the window
- Other types of pooling methods: Max Pooling, Average Pooling, ROI Pooling



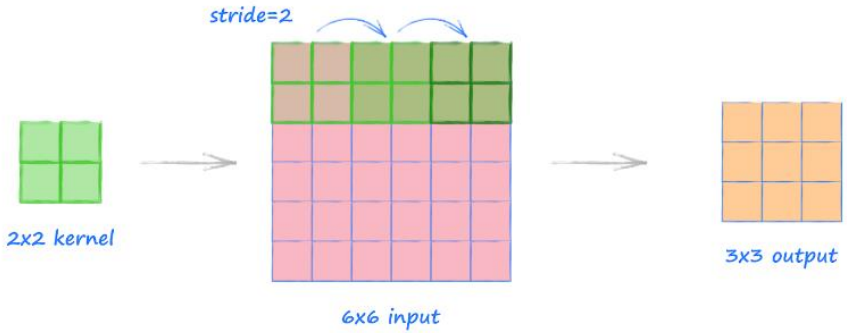
Fully Connected layer

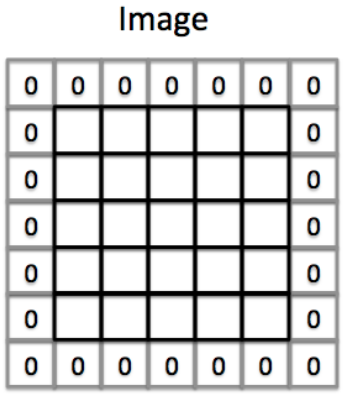
- Sometimes we use a term “flatten” to mean the conversion from 3-D structure to 1-D
- Fully-connected layer (called dense in keras) is similar to traditional neural networks
- May contain lots of weights



CNN Kernel

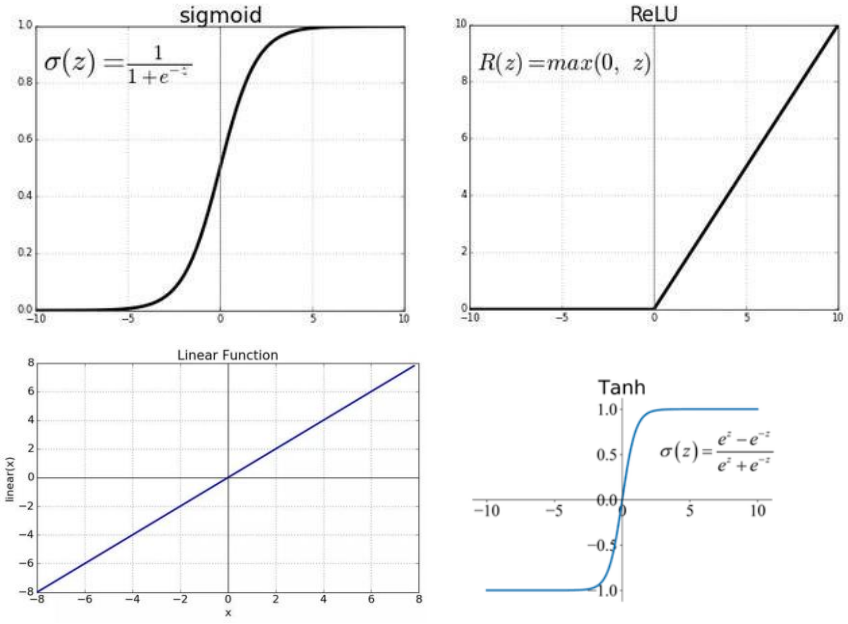
- Window(kernel, filter)
  - A filter that is used to extract the features from the images
- Stride
  - Defines the step size of the kernel when sliding through the image
- Padding
  - A term relevant to convolutional neural networks as it refers to the amount of pixels added to an image when it is being processed by the kernel of a CNN





CNN Activation Function

- Activation Function
  - A function that is added into an artificial neural network in order to help the network learn complex patterns in the data
  - Sigmoid(Used for two-class logistic)
  - ReLU
  - Linear
  - Softmax(Used for multi-class logistic)
  - Tanh



CNN Training

- Batch size
  - A hyperparameter that defines the number of samples to work through before updating the internal model parameters
- Dropout
  - Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel
- Learning Rate
  - The learning rate controls how quickly the model is adapted to the problem.
  - Smaller learning rates require more epochs given the smaller changes made to the weights each update
  - larger learning rates result in rapid changes and require fewer training epochs
- Epoch
  - The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset

LeNet (1998)

- LeNet 是由 Yann LeCun 團隊提出的網路架構，是卷積神經網路的始祖。其架構由兩個卷積層、池化層、全連接層以及最後一層 Gaussian 連接層所組成，早期用來辨識手寫數字影像
- 由下圖可以看到 LeNet 的網路架構共有七層：卷積層 (Convolutions, C1)、池化層 (Subsampling, S2)、卷積層 (C3)、池化層 (S4)、全連接卷積層 (C5)、全連接層 (F6)、Gaussian 連接層 (output)

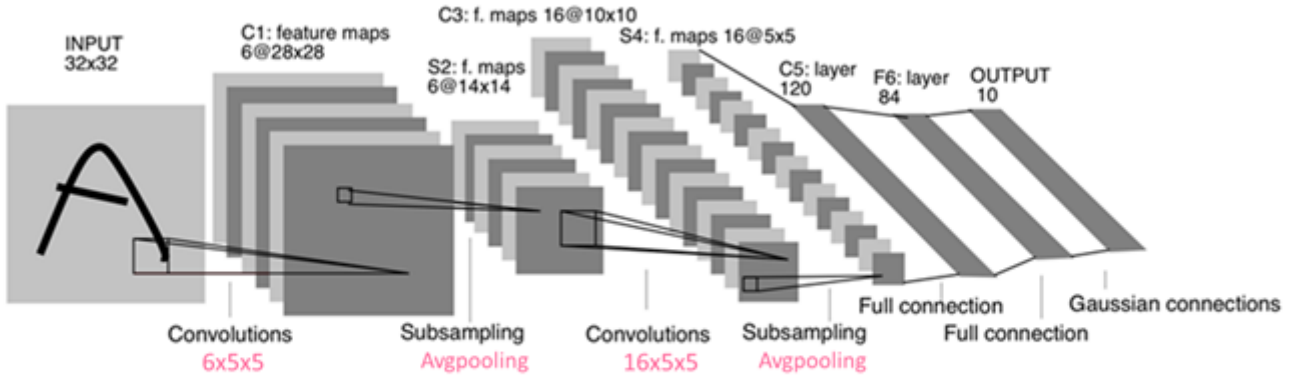


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet

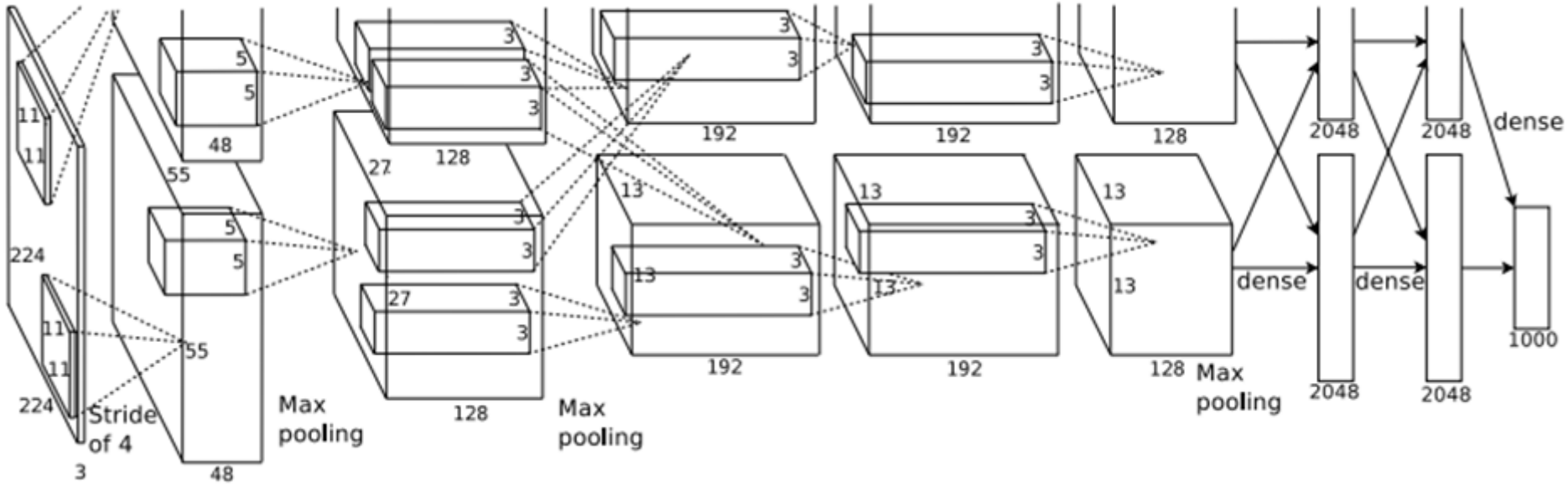
- 輸入層是一個 32x32 的圖片，而 Filter size 皆為 5x5，第一個 Filter 與第二個 Filter 的輸出通道分別為 6、16，並且皆使用 Sigmoid 作為activate function。
- 池化層的window為 2x2，stride 為 2，使用平均池化進行採樣。最後的全連接層的神經元數量分別是 120、84 個。
- 最後一層輸出層是 Gaussian連接層，採用 RBF 函數（徑向歐式距離函數），計算輸入向量和參數向量之間的歐式距離。因為 LeNet 應用於辨識手寫影像，數字為0~9，所以輸出層為 10 個神經元

AlexNet (2012)

- Alex Krizhevsky 於 2012 年提出卷積神經網路 AlexNet，並在同年的 ImageNet LSVRC 競賽中奪得了冠軍 (Top-5 錯誤率為 15.3%)，並且準確率遠超過第二名 (Top-5 錯誤率為 26.2%)，造成了很大的轟動，至此 CNN 開始廣泛被研究
- 在 AlexNet 出現之前，深度學習沉寂了一段時間。這是因為 LeNet 在早期的小資料集上可以取得好的成績，但在更大的資料集的表現卻不如人意，因此許多人改鑽研其他的機器學習方法。而 AlexNet 的出現，可以說是時代的分水嶺，正式開啟了 CNN 的時代

AlexNet

- AlexNet 的架構有八層，共使用五個卷積層、三個全連接層，相較於 LeNet 模型更深，下圖是 AlexNet 的網路架構



AlexNet

- AlexNet 為何如此成功? 以下來介紹 AlexNet的特點
- 1. 模型架構：
  - 第一層到第五層是卷積層，其中第一、第二和第五個卷積層後使用池化層，並且採用大小為 3x3、stride 為2 的 Maxpooling。比 LeNet 使用的平均池化，更能保留重要的特徵，並且因為 stride < size (2<3)，因此pooling 可以重疊 (overlap)，能重複檢視特徵，避免重要的特徵被捨棄掉，同時避免overfitting和的問題。第六到第八層則是全連接層
- 2. 輸入層尺寸：
  - 將輸入層變大，可以輸入224x224 尺寸的彩色圖片

AlexNet

- 3. 卷積核尺寸（kernel size）：
  - 由於輸入層變大，所以將第一層的kernel 設定為 11x11、stride 為4，擴大感受野(Receptive field)，並且也使用較大的步長（stride）提取特徵；第二層的 kernel 則使用 5x5，然後之後的 kernel 皆使用 3x3，stride 都設定為1
- 4. activate function(激勵函數)：
  - 將 LeNet 使用的 Sigmoid 改為 ReLU，可以避免因為神經網路層數過深或是梯度過小，[illegible]ent）的問題

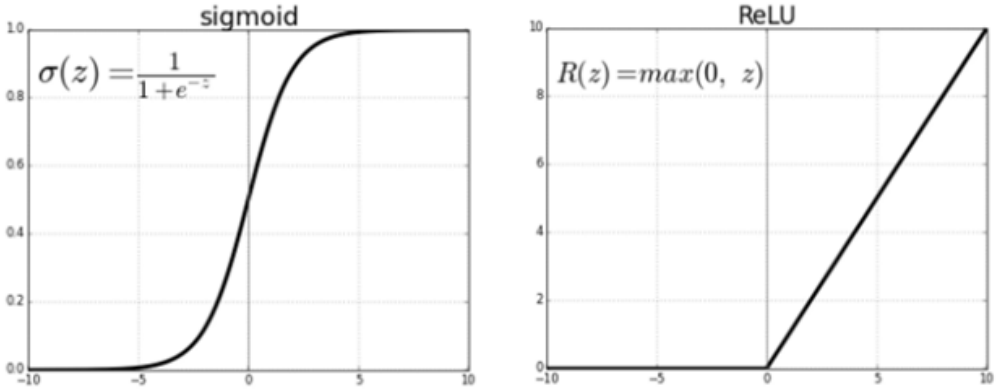
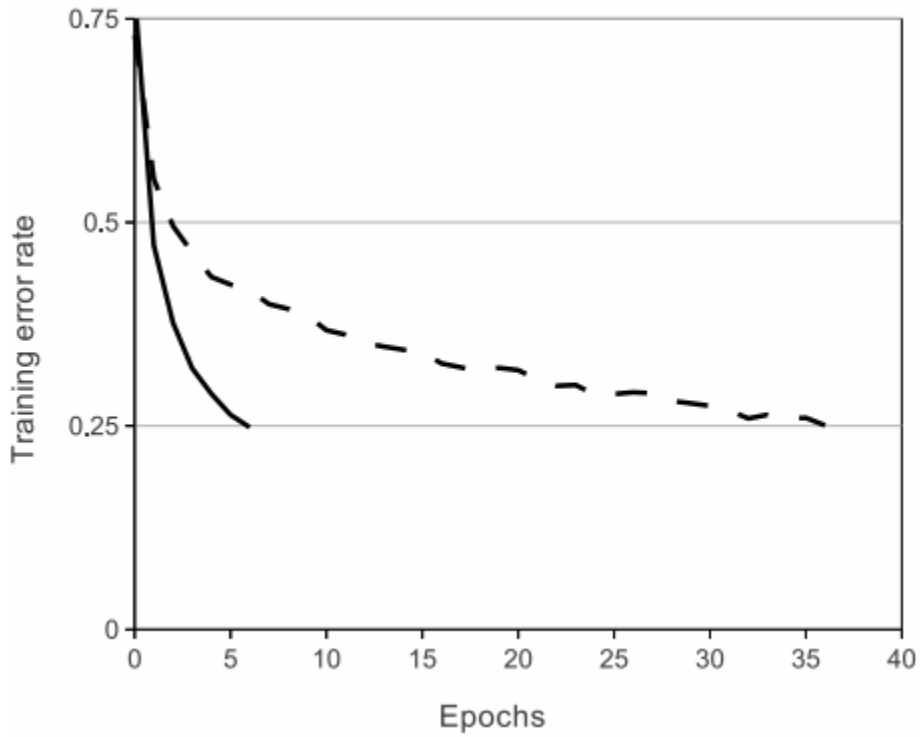


Fig: ReLU v/s Logistic Sigmoid

AlexNet

- 並且相較於早期使用的 Sigmoid/Tanh，ReLU 收斂速度較快。下圖是論文比較 ReLU 和 Tanh 的收斂速度



AlexNet

- 5. 降低 overfitting 的方法：採用了 Dropout 以及資料擴增 (Data augmentation)
  - Dropout：指在每一次訓練時，隨機選取神經元使其不參與訓練，如此一來可以讓模型不過度依賴某些特徵，進而增強模型的泛化能力 (generalization)，AlexNet 在第六、第七層全連接層中加入了 Dropout ，設定為 0.5
  - Data augmentation：為了提升演算法的準確率，增加訓練資料集是一個很有效的方式。資料擴增就是從既有的訓練資料集中利用一些變換去生成一些新的資料，以快速地擴充訓練資料

AlexNet

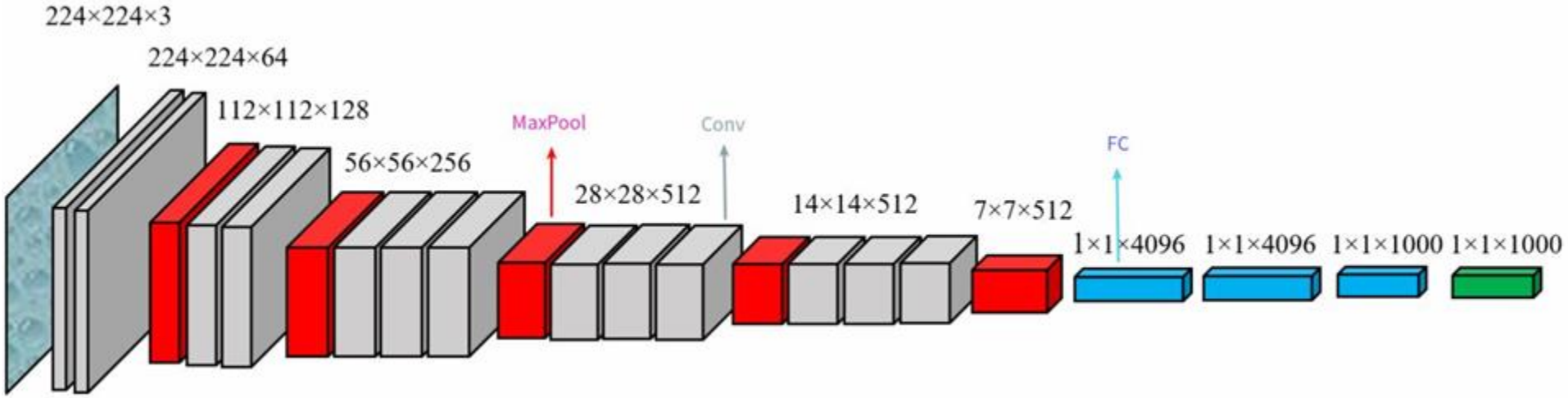
- AlexNet 的處理方式有兩種：
  - 隨機裁減 將 256x256 的圖片進行隨機裁減至 224x224，然後進行水平翻轉，這樣的做法可以增加 2*[(256 - 224)**2] = 2048 倍
  - 對 RGB 通道做主成分分析 (PCA)，接著使用高斯擾動，對顏色、光照做變換

AlexNet

- 6. 使用 GPU：
  - AlexNet 使用了兩塊 GTX580 GPU 做訓練，由於單個 GTX580 GPU 只有 3 GB 的記憶體，無法負荷如此大量的運算，因此將神經網路分布於兩塊 GPU 上平行運算，大幅的加快了訓練速度

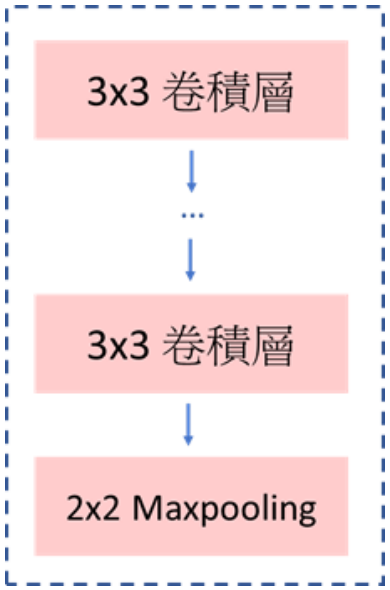
VGG (2014)

- VGGNet 在 2014 年由牛津大學 Visual Geometry Group 提出，並在 ImageNet LSVRC 的分類競賽中獲得了第二名（第一名是 GooLeNet）
- VGGNet 比起 AlexNet 採用更深層的網路，特點是重複採用同一組基礎模組，並改用小卷積核替代 AlexNet 裡的中大型卷積核，其架構由 n 個 VGG Block 與3個全連接層所組成



VGG

- VGG Block：
  - VGG Block 的構造就是由不同數量（數量為超參數）的 3x3 卷積層（kernel size=3x3, stride=1, padding=”same”），以及 2x2 的 Maxpooling 組成（pool size=2, stride=2）
  - padding=”same” 會保持 feature map 的尺寸跟原輸入一樣

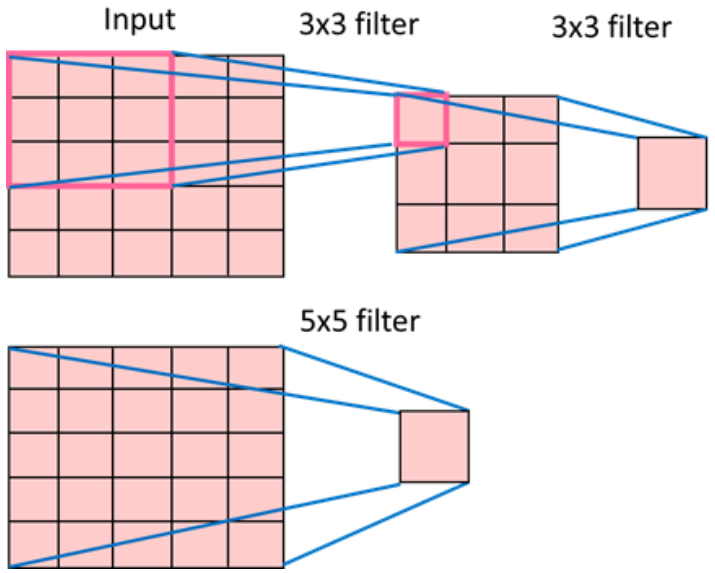


VGG

- 採用小卷積核代替大卷積核
  - VGG 使用多個小卷積核 3x3 來代替 AlexNet 的中大型卷積核 (5x5)，這樣的作法可以達到相同的感受野(Receptive field)，同時減少參數量

VGG

- 由下圖可以看出使用兩個 3x3 卷積層可以達到與 5x5 的卷積層相同的感受野(Receptive field)，5x5 的卷積層的訓練參數量為 5x5=25，而 3x3 的卷積層所需要的參數量更少，只需要 3x3x2=18 個參數量（不考慮 bias）



VGG

- VGGNet 有許多不同的結構，例如 VGG11, VGG13, VGG16, VGG19，其差異在於網路的層數（卷積層數量與全連接層數量）。常見的 VGGNet 指的是 VGG16，其架構使用了五個卷積層與三個全連接層，其中前兩個卷積層內含 2 個基礎模組、後三個卷積層內含 3 個基礎模組，總共為 2x2 + 3x3 + 3 = 16 層網路層數

VGG

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

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					

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

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

RCNN、Fast RCNN、Faster RCNN

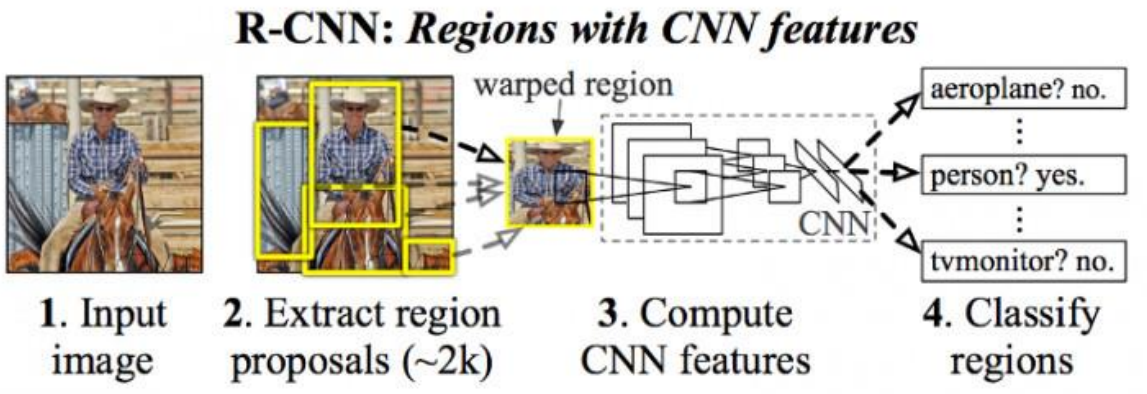
R-CNN-概述

- 卷積神經網路(CNN)經過很多改良與演化，逐漸從CNN發展到R-CNN到Fast R-CNN再到Faster R-CNN。當中運用各種不同的手段或方法來提升它的速度。傳統CNN是對整張影像做卷積以提取特徵，在辨識新的影像時亦是如此。如下圖，人的肉眼可清楚辨識出影像中有一台汽車和一個人，但傳統CNN會對沒有目標物件的區域(如下圖框1和框2的區塊)進行檢測，這很明顯的是在浪費時間。

R-CNN-概述

- 所以Ross Girshick，Jeff Donahue，Trevor Darrell，Jitendra Malik，在論文中提出一種叫做R-CNN的算法，意思是帶區域的卷積網絡，或者說帶區域的CNN。這個演算法嘗試選出一些區域，在這些區域上運行卷積網絡分類器是有意義的，所以這裡不再針對每個滑動窗運行檢測算法，而是只選擇一些ROI區域，在少數ROI區域上執行卷積網絡分類器
- R-CNN將深度學習引入檢測領域後，一舉將PASCAL VOC資料集上的辨識率從35.1%提升到53.7%。

R-CNN-流程



- R-CNN演算法的流程如下：
  1. 輸入影像。
  2. 候選區域選擇，Region Proposal是一類傳統的區域提取方法，可以視作使用長寬不同的sliding window， Proposal 部分採用Selective Search，核心概念在於依據紋理、色彩、大小等特徵進行分組。
  3. 對每個候選區域，使用深度網路CNN提取特徵，使用池化、卷積等操作，得到固定維度的輸出。
  4. 分類與邊框回歸（bounding-box regression）：
     1. 將特徵送入每一類的SVM 分類器，判別是否屬於該類。
     2. 訓練一個線性迴歸模型，為每個辨識到的物體生成更精確的邊界框。

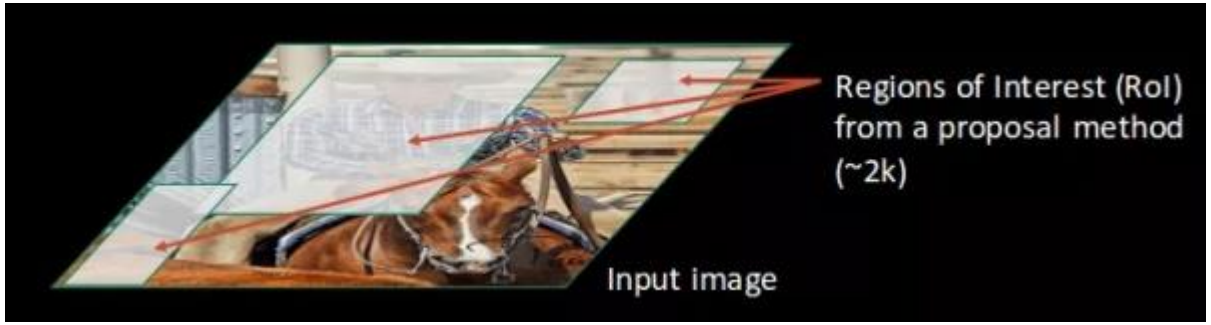
R-CNN-ROI選取步驟

- 首先將一張圖片作為輸入。
- 之後，它會生成最初的sub-分割，將圖片分成多個區域。
- 基於顏色、結構、尺寸、形狀，將相似的區域合併成更大的區域。
- 最後，生成最終的目標物體位置（Region of Interest）。

R-CNN-執行步驟

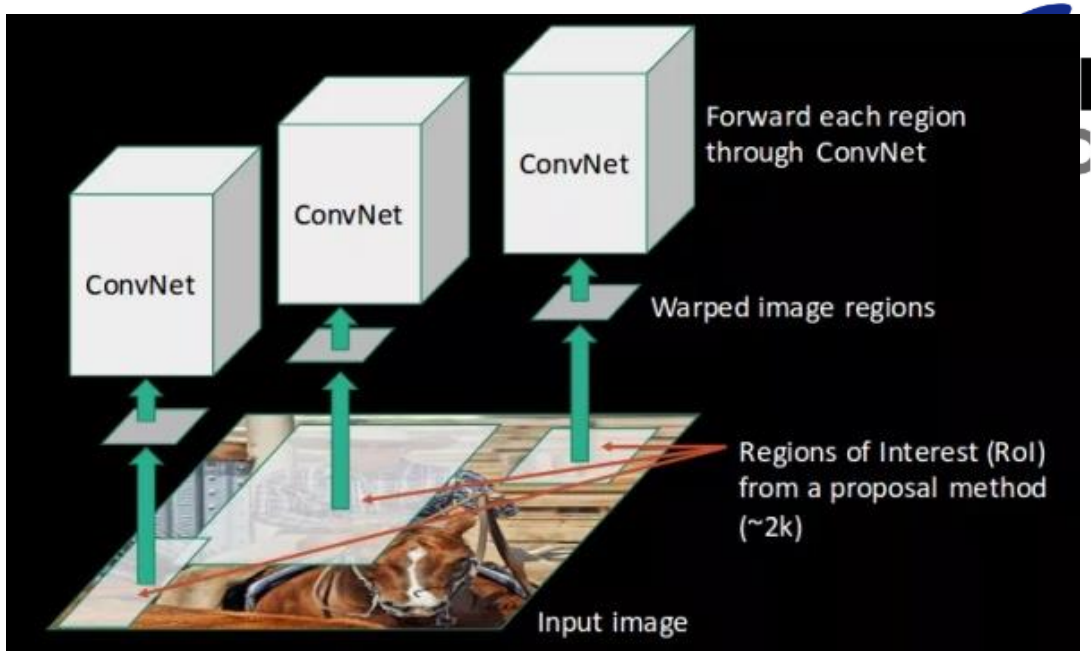
- 輸入一張圖片

- 尋找ROI

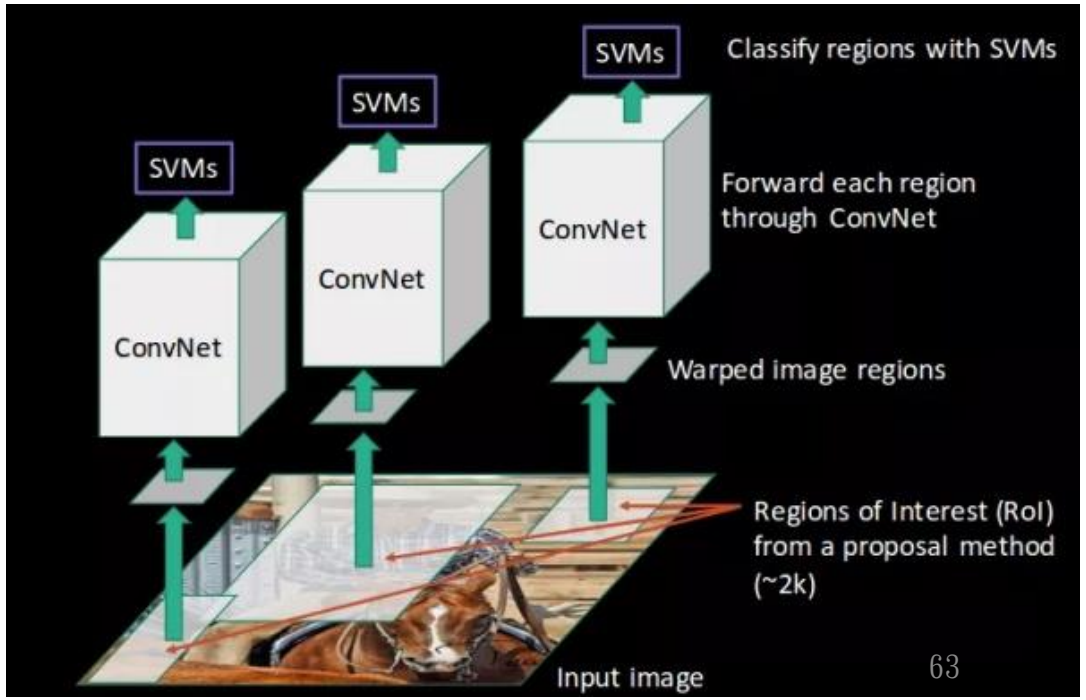


R-CNN-執行步驟

- 將這些區域輸入到CNN中

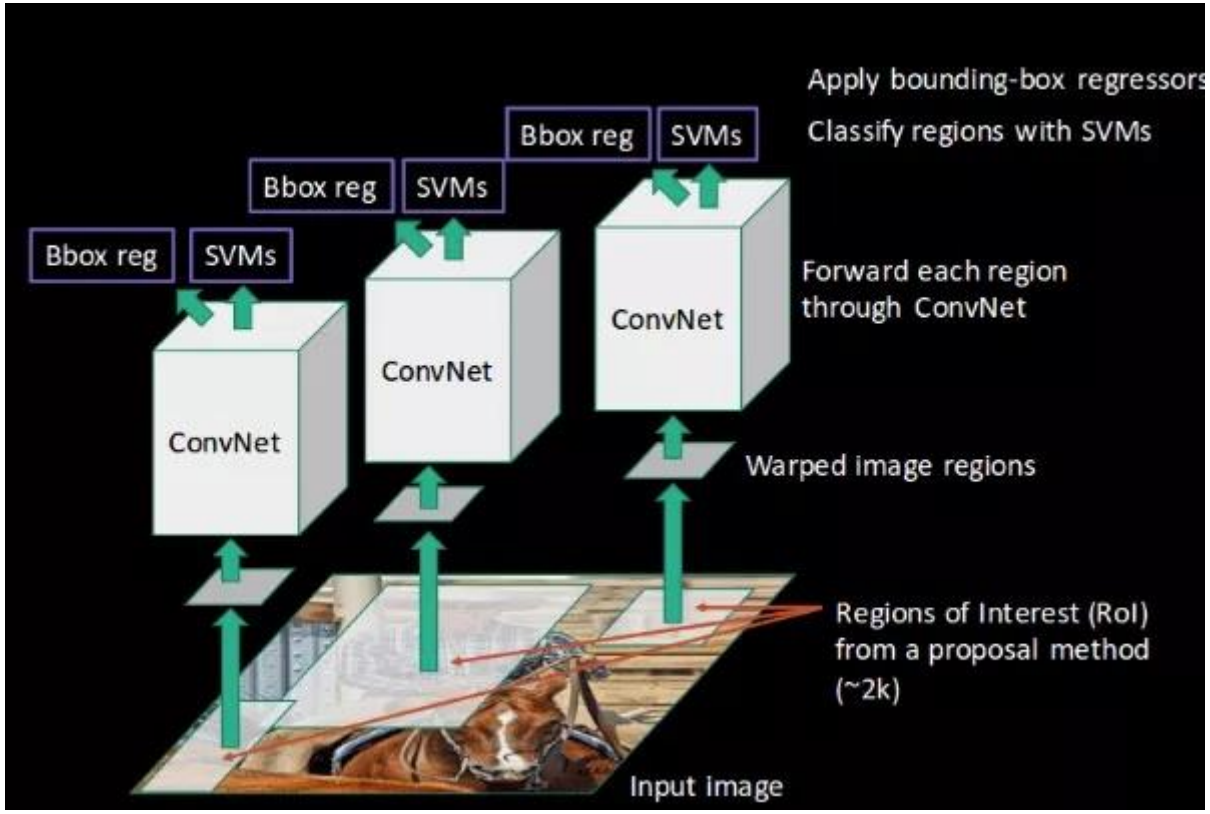


- CNN為每個區域萃取特徵，利用SVM將這些區域分成不同類別



R-CNN-執行步驟

- 用邊界框迴歸預測每個區域的邊界框位置



R-CNN-缺點

- 訓練一個RCNN模型非常昂貴，並且步驟較多：
  - 根據選擇性搜尋，要對每張圖片提取2000個單獨區域。
  - 用CNN提取每個區域的特徵。假設我們有N張圖片，那麼CNN特徵就是N*2000。
  - 用RCNN進行目標檢測的整個過程有三個模型：
    - 用於特徵提取的CNN。
    - 用於目標物體辨別的線性SVM分類器。
    - 調整邊界框的迴歸模型。
- 有鑑於以上種種原因，使得整體運行效率大幅降低，為了解決這個問題，產生了下一代的方法 - Fast RCNN。

Fast R-CNN-概述

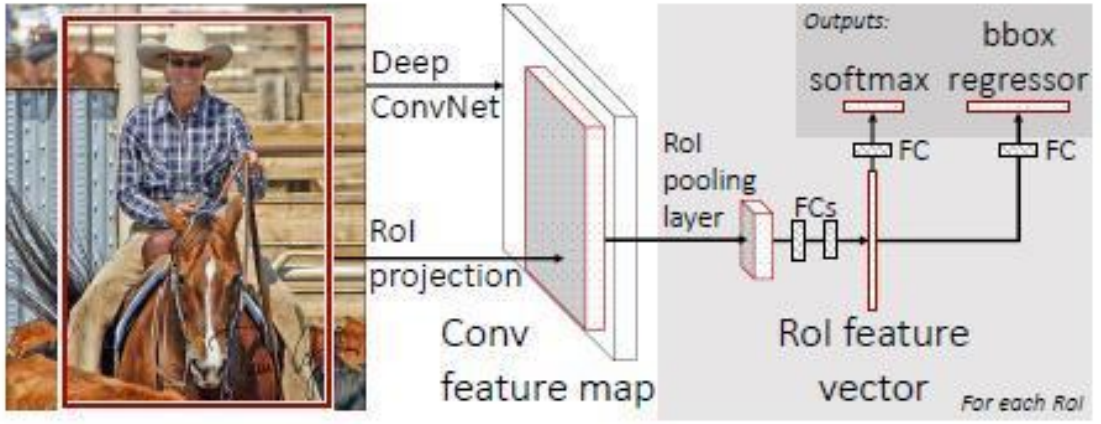
- 繼2014年的R-CNN推出之後，Ross Girshick在2015年推出Fast R-CNN，構思精巧，流程更為緊湊，大幅提升了目標檢測的速度。
- Fast R-CNN和R-CNN相比，訓練時間從84小時減少到9.5小時，測試時間從47秒減少到0.32秒，並且在PASCAL VOC 2007上測試的準確率相差無幾，約在66%-67%之間。

		R-CNN	Fast R-CNN
Faster!	Training Time:	84 hours	**9.5 hours**
	(Speedup)	1x	**8.8x**
FASTER!	Test time per image	47 seconds	**0.32 seconds**
	(Speedup)	1x	**146x**

Fast R-CNN-做法概述

- RCNN的作者Ross Girshick提出了一種想法，在每張照片上只執行一次CNN，然後找到一種方法在2000個區域中進行計算。
  在Fast RCNN中，我們將圖片輸入到CNN中，會相應地生成傳統特徵對映。利用這些對映，就能提取出ROI感興趣區域。之後，我們使用一個ROI池化層將所有提出的區域重新修正到合適的尺寸，以輸入到全連接層中。

Fast R-CNN-流程

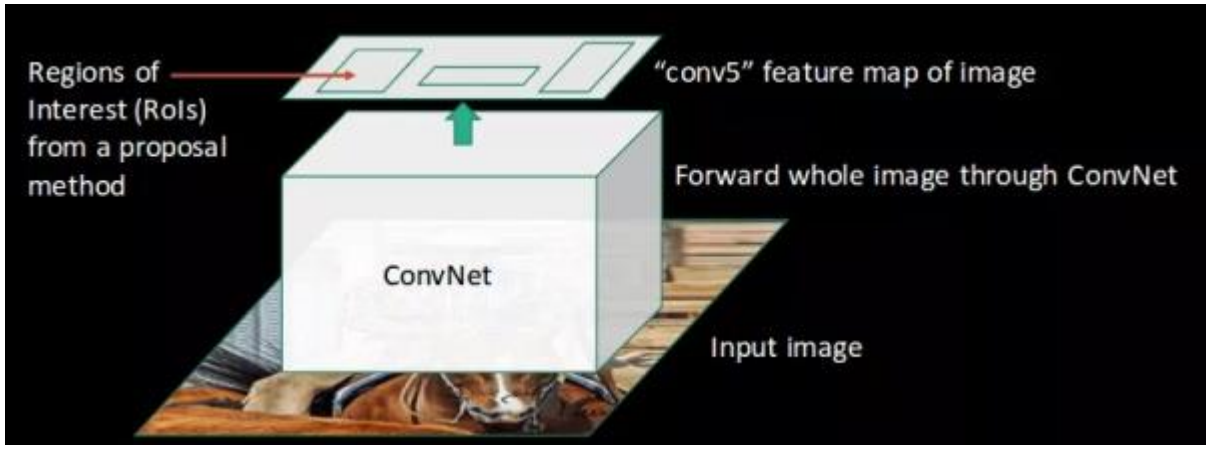


- Fast R-CNN演算法的流程如下：
  1. 輸入影像。
  2. 候選區域選擇，找到ROI所在。
  3. 對每個候選區域，使用深度網路CNN提取特徵，使用池化、卷積等操作，得到固定維度的輸出(加入一個ROI池化層，確定每個區域尺寸相同)。
  4. 分類與邊框回歸（bounding-box regression）：
     1. 使用softmax分類器進行分類(softmax加在模型的頂層)(softmax與SVM有著不同的loss function，同時將分類行為改為機率分布的判斷)。
     2. 在模型中加入線性迴歸層(使用multi-task loss函式)以取得邊界框。

Fast R-CNN-執行步驟

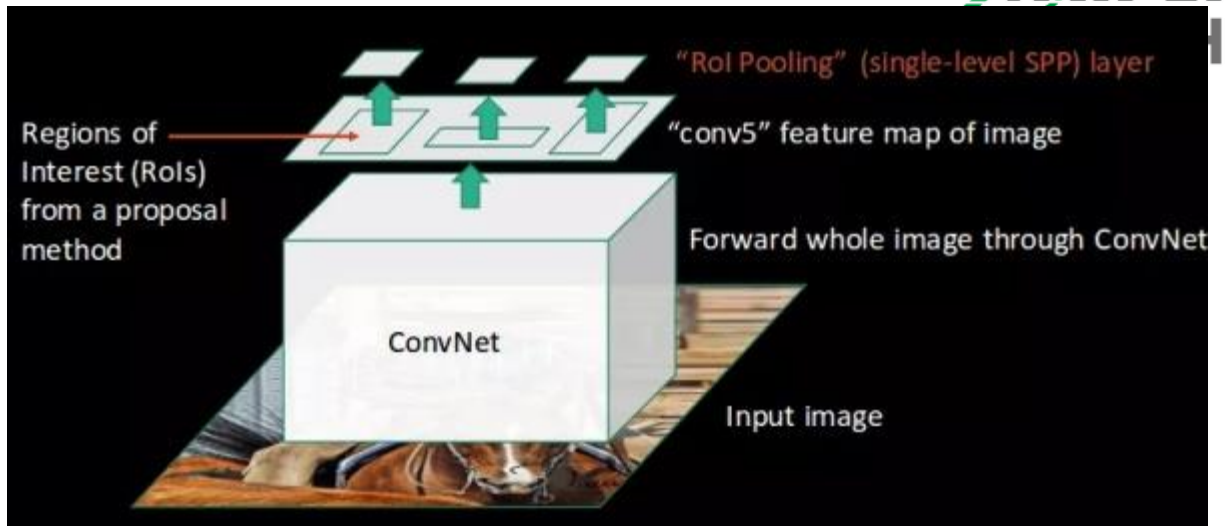
- 輸入一張圖片

- 影像被傳遞到卷積網路中，以獲得感興趣區域

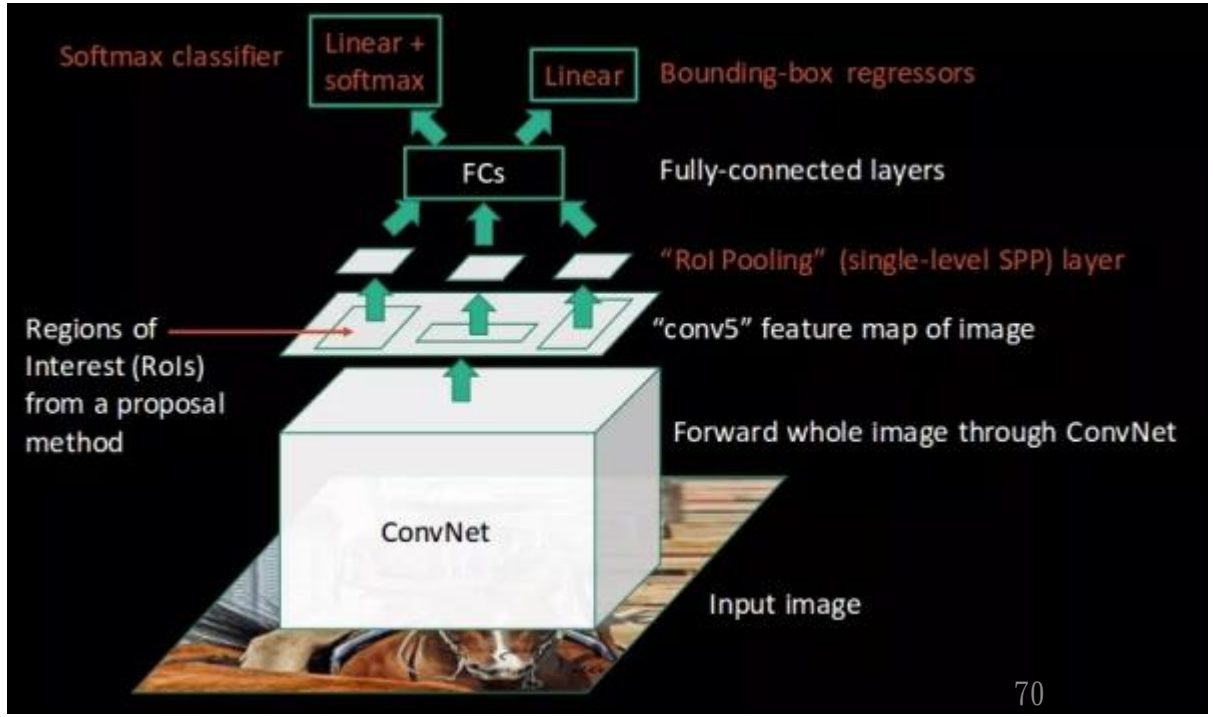


Fast R-CNN-執行步驟

- 在區域上應用RoI池化層，保證每個區域的尺寸相同



- 將這些區域傳遞到一個全連接層中進行分類，softmax和線性迴歸層同時返回邊界框



Fast R-CNN-與R-CNN的主要差異

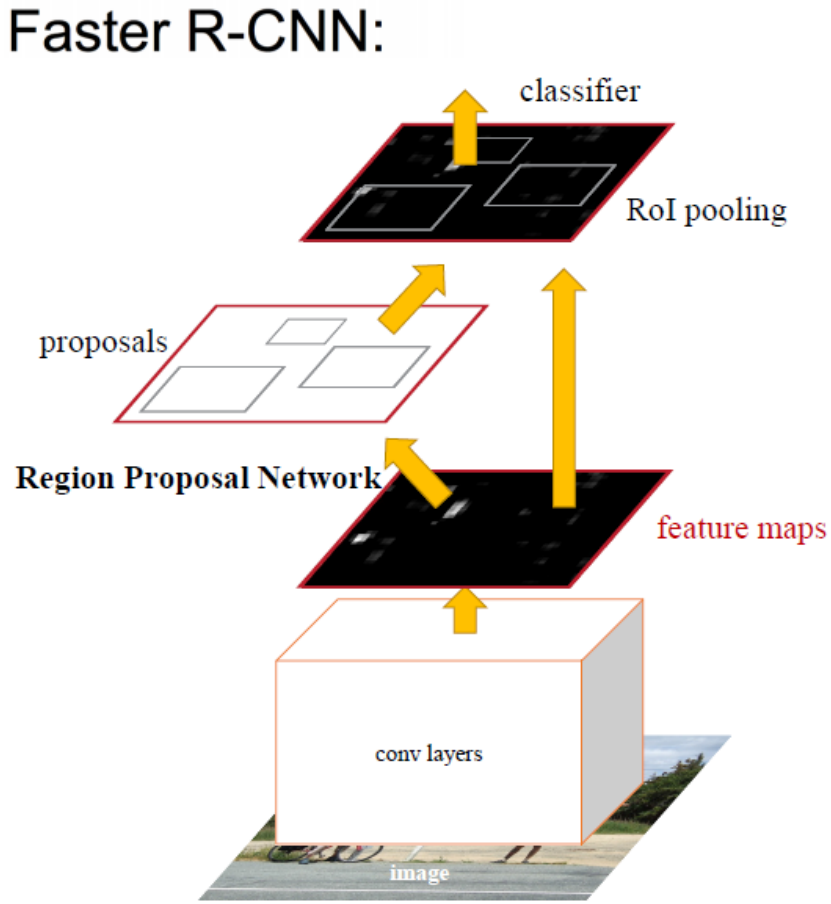
- 卷積不再是對每個region proposal進行，而是直接對整張影象，這樣減少了很多重複計算。原來R-CNN是對每個region proposal分別做卷積，因為一張影象中有2000左右的region proposal，肯定相互之間的重疊率很高，因此產生重複計算。
- 用ROI pooling進行特徵的尺寸變換，因為全連接層的輸入要求尺寸大小一樣，因此不能直接把region proposal作為輸入。
- 將regressor放進網路一起訓練，每個類別對應一個regressor，同時用softmax代替原來的SVM分類器(同時使得R-CNN的三個模型減少為只需要一個模型)。

Fast R-CNN-缺點

- 與R-CNN相同，同樣用的是Selective Search作為尋找感興趣區域的，這一過程通常較慢。與RCNN不同的是，透過縮減模型以及降低對重疊部分的重複計算，使得Fast RCNN處理一張圖片時間縮短至只需要約2~3秒。但是在大型真實資料集上，這種速度仍然不夠理想。

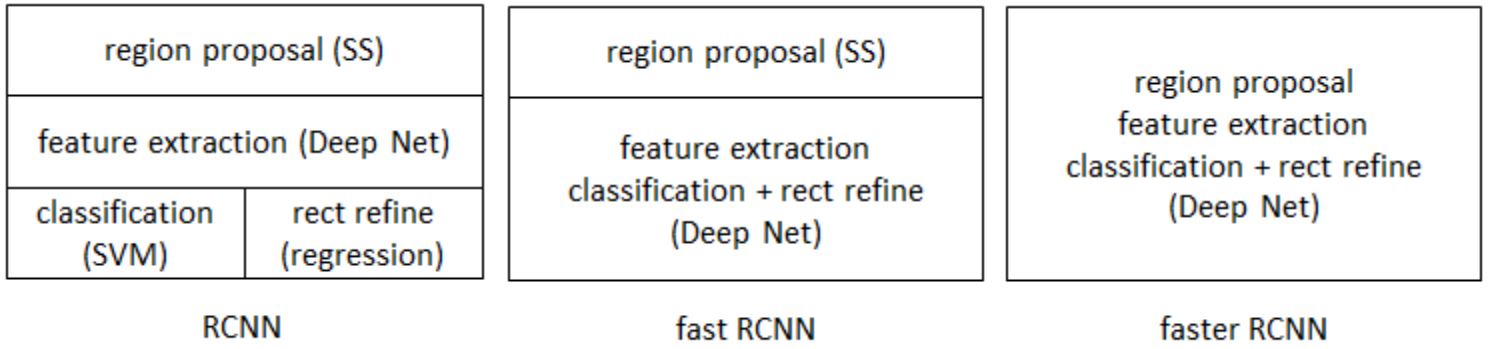
Faster R-CNN-概述

- 繼2014年推出R-CNN，2015年推出Fast R-CNN之後，目標檢測界的領軍人物Ross Girshick團隊在2015年又推出一力作：Faster R-CNN，使簡單網路目標檢測速度達到17fps，在PASCAL VOC上準確率為59.9%，複雜網路達到5fps，準確率78.8%。



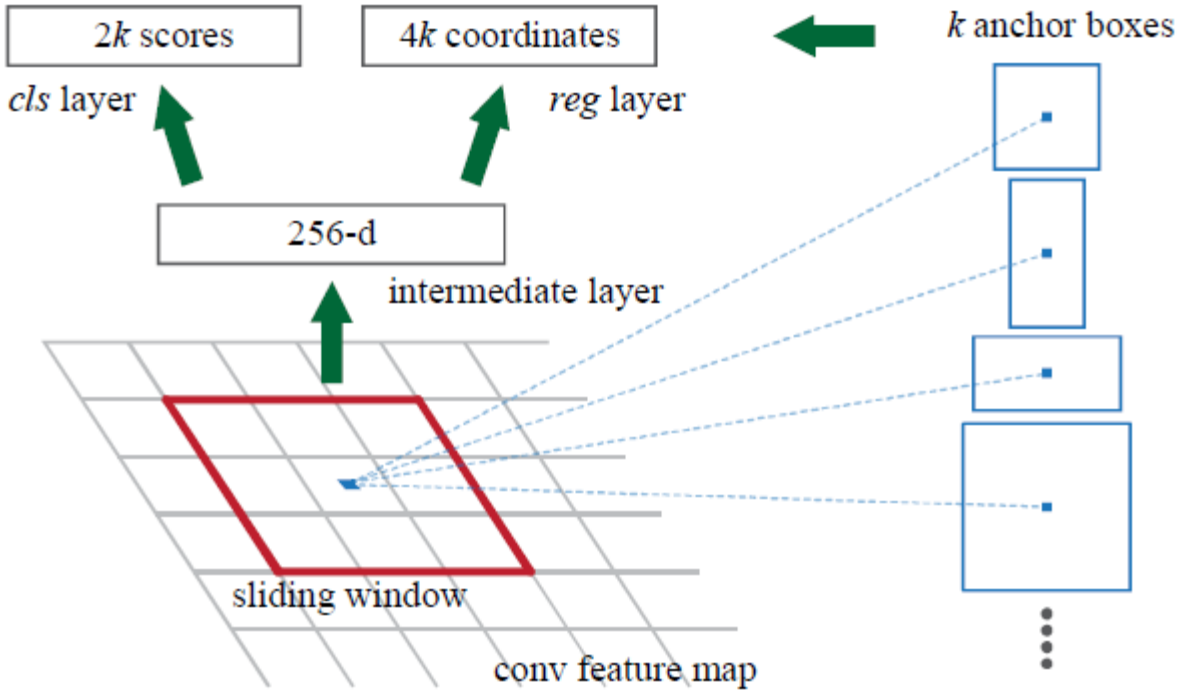
Faster R-CNN-概述

- 在Fast R-CNN還存在著瓶頸問題：Selective Search（選擇性搜尋）。要找出所有的候選框，這個也非常耗時。那我們有沒有一個更加高效的方法來求出這些候選框呢？
- 在Faster R-CNN中加入一個提取邊緣的神經網路，也就說**找候選框的工作也交給神經網路來做了**。這樣，目標檢測的四個基本步驟（候選區域生成，特徵提取，分類，位置精修）終於被統一到一個深度網路框架之內。如下圖所示。



Faster R-CNN-概述

- Faster R-CNN可以簡單地看成是「RPN+ Fast R-CNN」的模型，用區域生成網路（Region Proposal Network，簡稱RPN）來代替Fast R-CNN中的Selective Search（選擇性搜尋）方法。
- RPN架構如右圖所示。



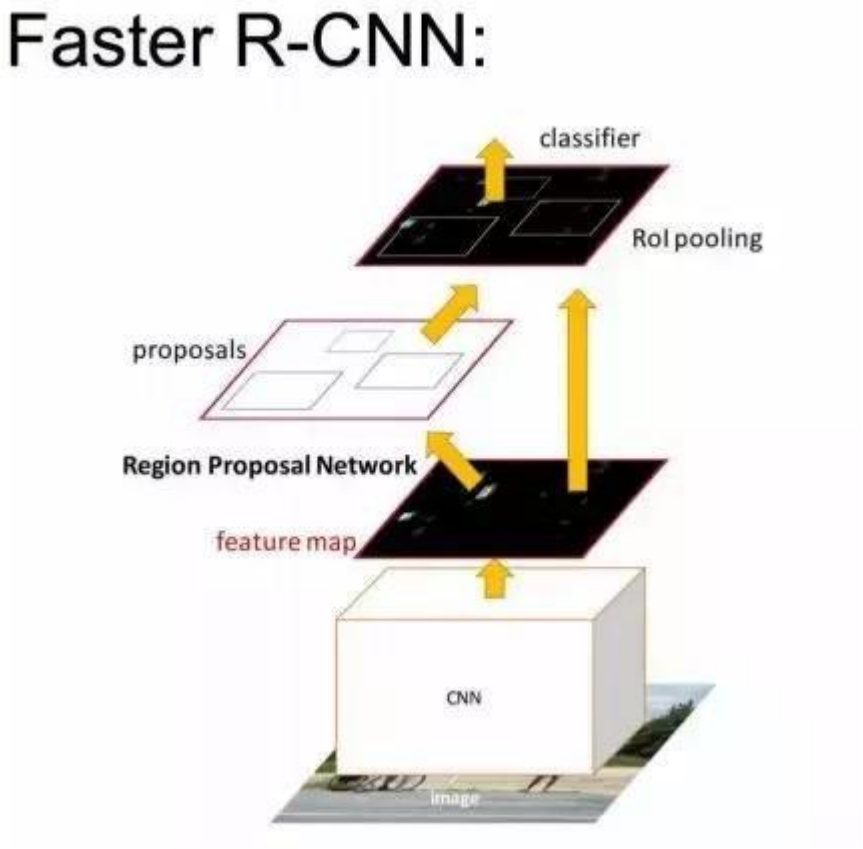
Region Proposal Network (RPN)

RPN(Region Proposal Network)概述

- RPN的工作步驟如下：
  - 在feature map（特徵圖）上滑動視窗
  - 建一個神經網路用於物體分類+框位置的迴歸
  - 滑動視窗的位置提供了物體的大體位置資訊
  - 框的迴歸提供了框更精確的位置

Faster R-CNN-流程

- Faster R-CNN演算法的流程如下：
  1. 輸入影像。
  2. 在特徵對映上應用Region Proposal Networ
     ，返回object proposals和相應
  3. 應用Rol池化層，將所有proposals修正到同樣尺寸。
  4. 最後，將proposals傳遞到全連接層，生成目標物體的邊界框。



Faster R-CNN-與Fast R-CNN的主要差異

- 加入Region Proposal Network的動作，意味著也將找到候選框的工作也交給神經網絡來做了。這個動作大幅降低了R-CNN以及Fast R-CNN面臨的最大問題，也就是使用Selective Search尋找ROI的行為會大幅拖慢整體運作的情況。

從CNN到R-CNN到Fast R-CNN再到Faster R-CNN……

1. RCNN 解決的是,“為什麼不用CNN做classification呢？” 用SelectiveSearch去選框，CNN取特徵，SVM分類。BoundingBox迴歸。
2. Fast-RCNN 解決的是，“為什麼不一起輸出bounding box和label呢？”
3. Faster-RCNN 解決的是，“為什麼還要用selective search呢？為什麼不用CNN做特徵提取呢？” 鑑於神經網路的強大的feature extraction能力，可以將目標檢測的任務放到NN上面來做，於是出現了RPN（region proposal network）

R-CNN、Fast R-CNN、Faster R-CNN的比較

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	**0.2 seconds**
(Speedup)	1x	25x	**250x**
mAP (VOC 2007)	66.0	**66.9**	**66.9**

R-CNN、Fast R-CNN、Faster R-CNN的比較

	使用方法	缺點	改進
R-CNN	Selective Search提取 Region Proposal CNN提取特徵 SVM分類 Bounding Box迴歸	訓練步驟繁瑣 訓練、測試速度慢 訓練占空間	從DPM HSC的34.3%直接提升到了66%（mAP） 引入RP CNN
FAST R-CNN	Selective Search提取 Region Proposal CNN提取特徵 softmax分類 multi-task loss函式邊框迴歸	用SS提取RP(慢) 無法滿足實時應用，沒有真正實現端到端訓練測試 利用了GPU，但是區域建議方法是在CPU上實現的	由66.9%提升到70% 每張影像耗時約為2s
FASTER R-CNN	region proposal network提取 Region Proposal CNN提取特徵 softmax分類 multi-task loss函式邊框迴歸	還是無法達到即時檢測目標 獲取region proposal，再對每個proposal分類計算量還是比較大。	提高了檢測精度和速度 真正實現端到端的目標檢測框架 生成建議框僅需約10ms