111 Summer Workshop

ML / DL / ANN / CNN

7/25 (一) 15:30-17:30

AIM Lab 李偉華

Machine Learning Intro.

Outline

- **Machine Learning Intro. (about 0.5 hour)**
  - Supervised - Linear Regression
  - Unsupervised - Clustering
  - Unsupervised - PCA

Why Machine Learning ?

- 我們也許可以告訴電腦規則(Rules)... 用if else 判斷式來打遍天下。
- 但有些任務很困難，我們很難歸納出一個通則並告訴程式如何解決任務。也就是為什麼需要學習。

if 有眼睛和耳朵 return 狗
else return 麵包

狗

if 有眼睛和耳朵 return 狗
else return 麵包

麵包

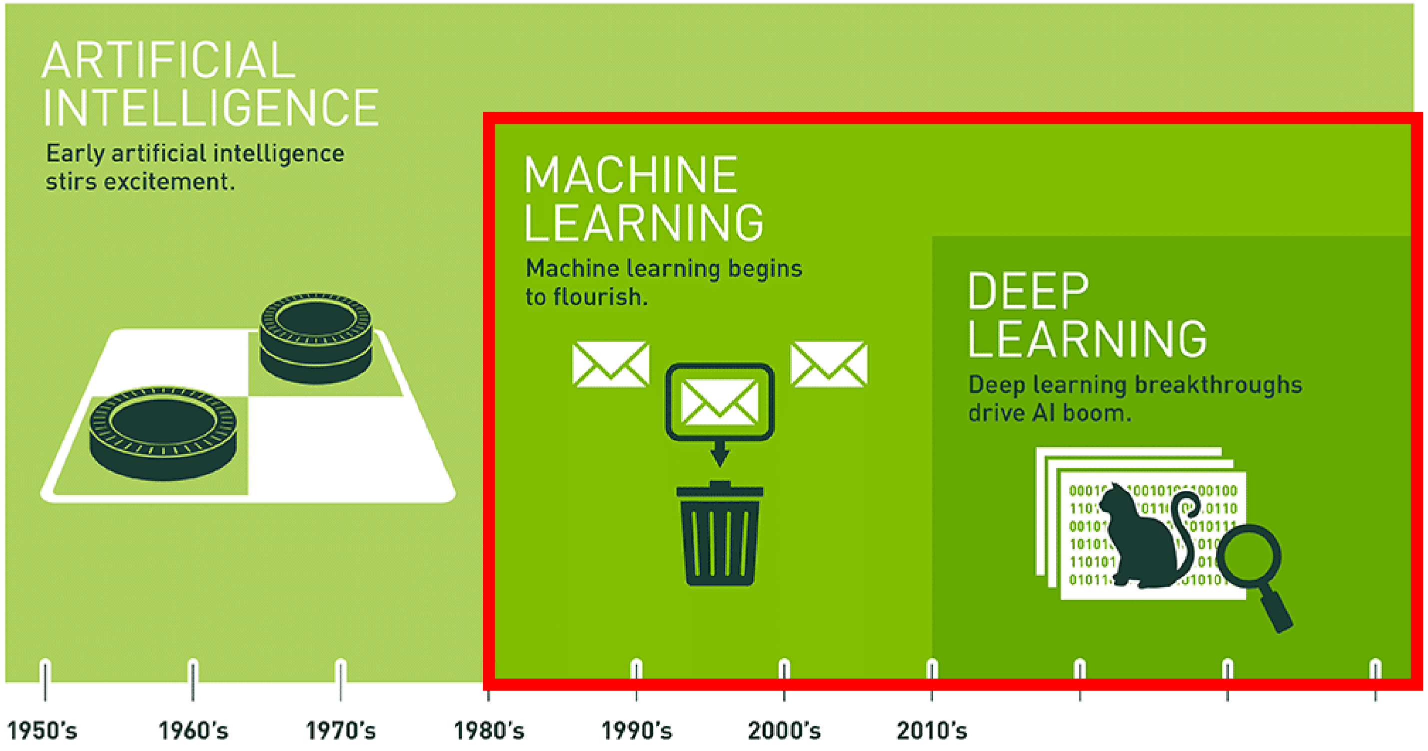
if 有眼睛和耳朵 return 狗
else return 麵包

麵包 ?

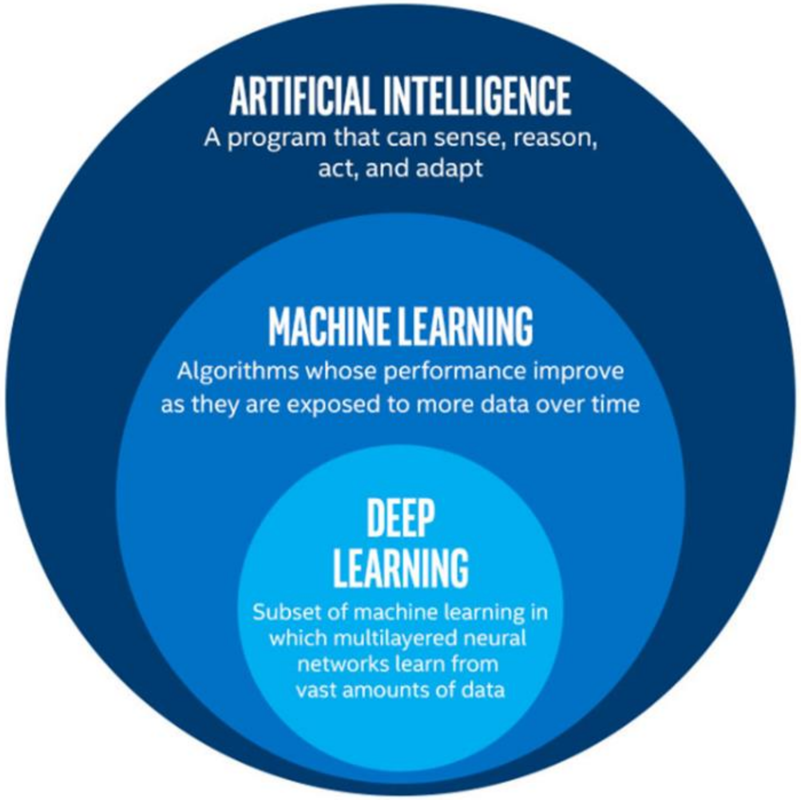
if 有眼睛和耳朵 return 狗
else return 麵包

狗 ?

AI & ML & DL



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.



AI ：計算機模仿人類思考進而模擬人類的能力/行為。

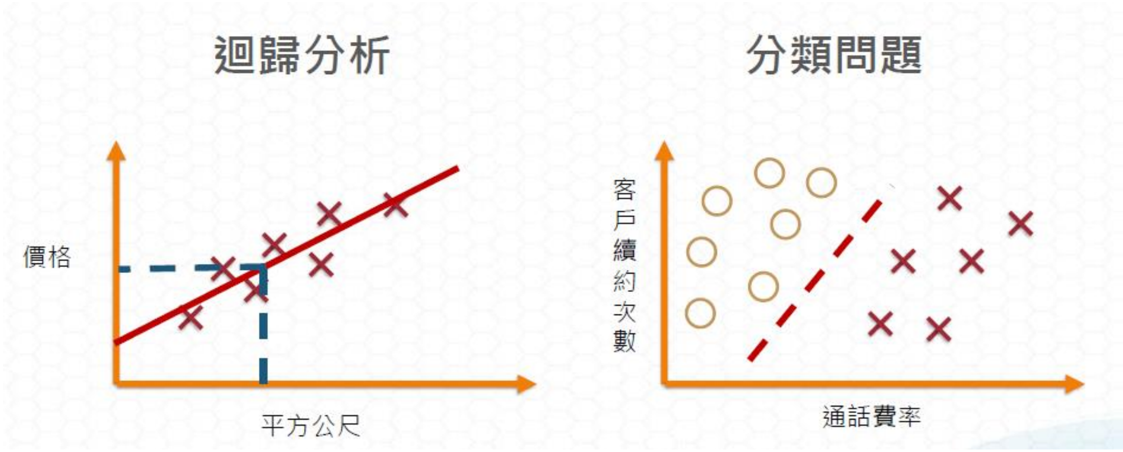
ML：從資料中學習模型。

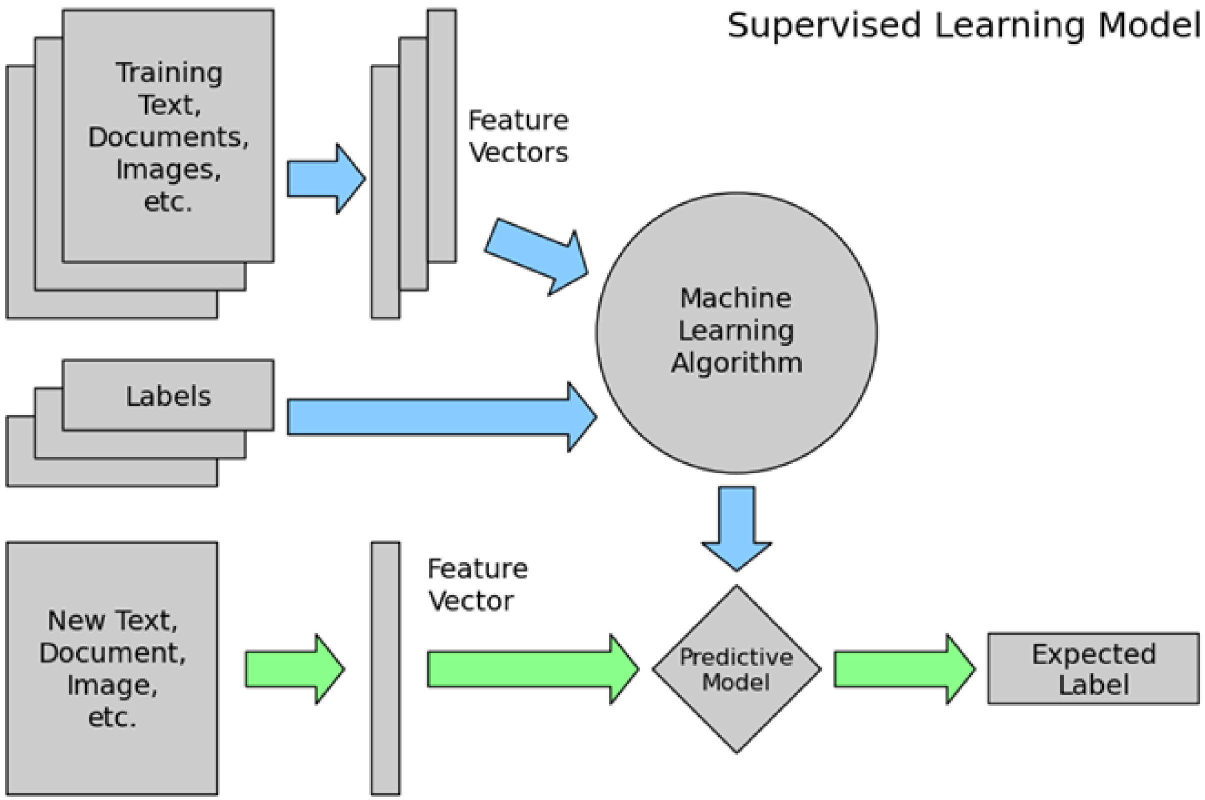
DL ：利用多層的非線性學習資料表徵。

Different Type of ML

監督式學習 Supervised Learning

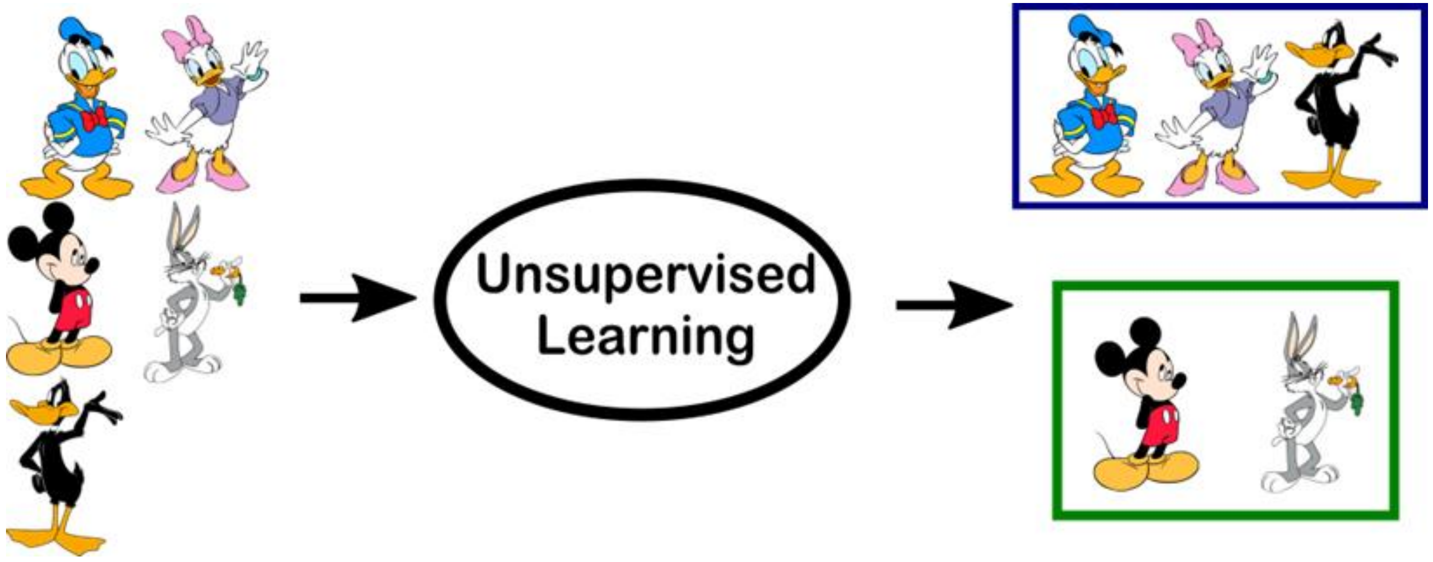
從被標記的訓練數據集 (Training data) 中學習，並導出模型，而這個模型可以對未來的數據做預測。





非監督式學習 Unsupervised Learning

又稱作無監督式學習，這類學習和監督式學習相反，是指訓練資料中並**沒有任何已標籤的資料**，因此模型必須從數據資料集本身找出一定的規則，產生某種能給出結果的模式。



半監督式學習 Semi-Supervised Learning

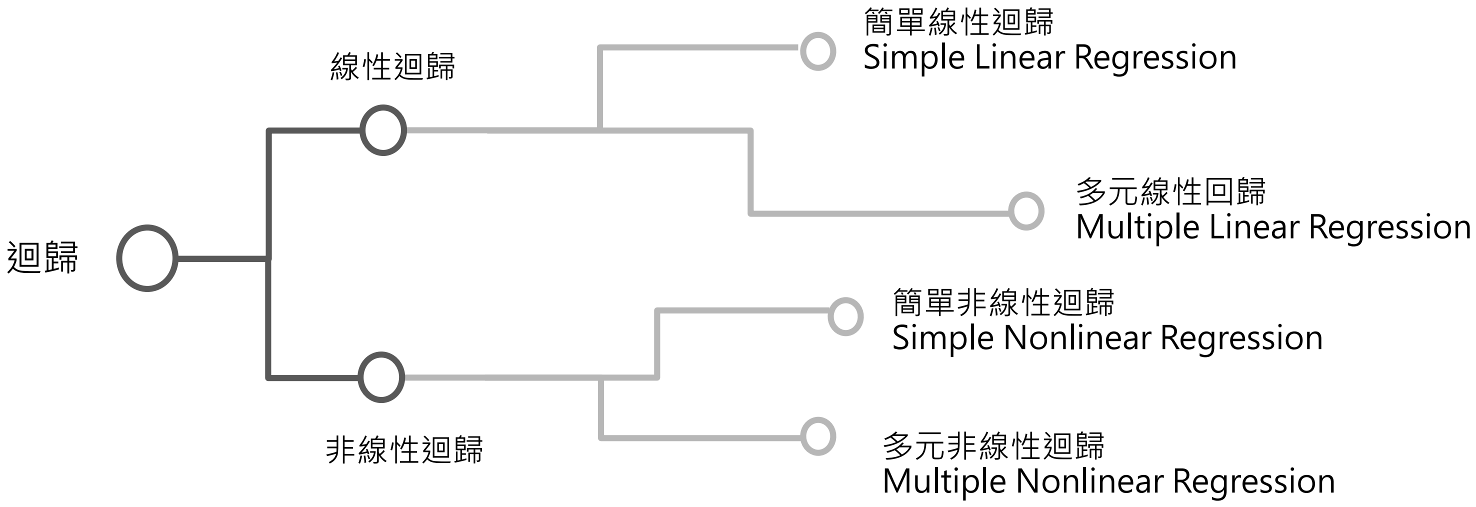
半監督式學習可以說是介於監督式學習和非監督式學習之間，訓練資料會有一群有標記過的資料，以及一群沒有標籤的資料，且有標記的資料往往會比沒有標記的資料要少很多。 如果到很極端的只有一點點資料，就會稱作 Few-Shot Learning.

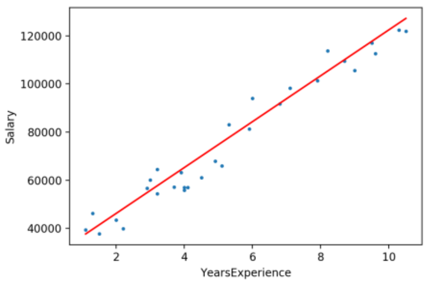
- Transductive Learning : **已經見過測試資料**
  - 你的 Unlabeled data 就是你的測試資料，會參考 testing data 的 feature，但是不使用 testing data 的 label。
- Inductive Learning : **沒有見過測試資料**
  - 我們不考慮測試資料，也就是我們完全不知道測試資料是什麼。

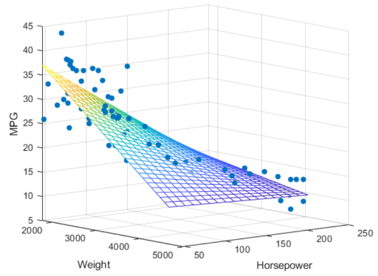
其他學習方式 Others

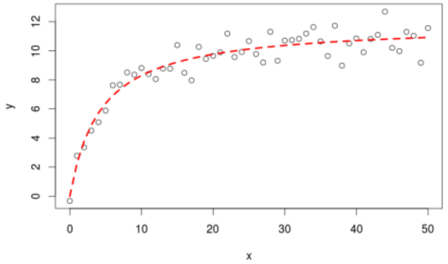
- Reinforcement Learning (強化學習) :
  - 電腦透過與一個動態環境不斷重複地互動，來學習正確地執行一項任務。這種嘗試錯誤(trial-and-error)的學習方法，使電腦在沒有人類干預、沒有被寫入明確的執行任務程式下，就能夠做出一系列的決策。
- Life-long Learning or Continuous Learning (持續學習) :
  - 試圖要解決災難性遺忘的問題 (Catastrophic Forgetting Problem)。
- Active Learning (主動學習):
  - 通過一定的演算法查詢最有用的未標記樣本，並交由專家進行標記，然後用查詢到的樣本訓練分類模型來提高模型的精確度。
- Meta Learning (元學習):
  - 試圖解決訓練資料與測試資料的 Domain Gap ，衍生出來的學習方式，在面臨新的樣本時會再去微調模型。

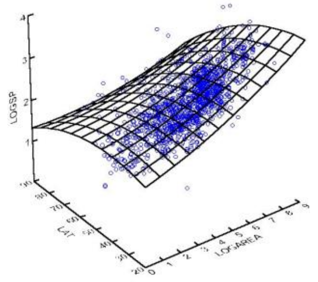
Supervised - Regression











自變數

Independent (a) 獨立的

代表不會被其他數影響

應變數

dependent (a) 相依的

代表會被其他數影響

舉例來說:

自變數

地段、坪數、裝潢、交通、生活機能

可能會影響 房價

應變數

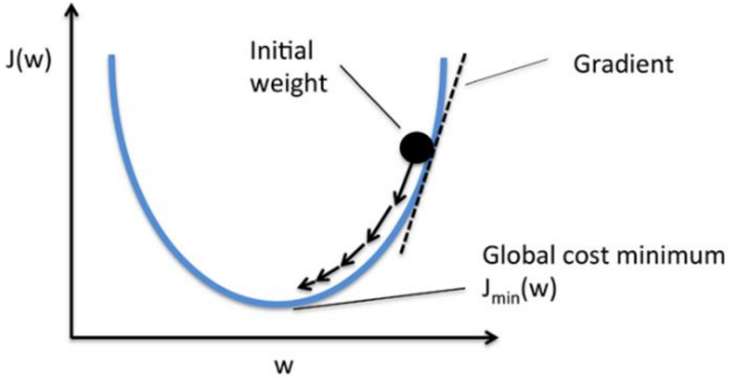
迴歸其實是在找自變數和應變數的關係

梯度下降法線性迴歸

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train, y_train)
y_test = lr.predict(x_test)
```

- 這邊介紹使用 梯度下降 (Gradient Descent) 的方式 來做簡單線性迴歸
  - 設定: 損失函數、學習率、Epoch數
  - 目標要找到一條線: $y = \beta_0 + \beta_1 x$
  - 假設損失函數為: L2 Loss Function (最小平方法) : $\mathrm{Loss}(\hat{\beta}_0, \hat{\beta}_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
  - Learning Rate
  - 這條線能使損失越小越好

```
# 開始訓練回歸模型
for i in range(epochs):
    y_predicted = a*x + b

    # 計算微分值
    d_a = (-2/n) * sum(x * (y - y_predicted))
    d_b = (-2/n) * sum(y - y_predicted)

    # 更新參數
    a = a-learning_rate*d_a
    b = b-learning_rate*d_b
```



$$\mathrm{Loss}(\hat{\beta}_0, \hat{\beta}_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\hat{y} = \beta_0 + \beta_1 x$$

$$\mathrm{Loss}(\hat{\beta}_0, \hat{\beta}_1) = \frac{1}{n}\sum(y_i - (\beta_0 + \beta_1 x))^2$$

$$\frac{dLoss(\widehat{\beta_0}, \widehat{\beta_1})}{d\beta_0} = -\frac{2}{n}\sum(y_i - (\beta_0 + \beta_1 x))$$

$$\frac{dLoss(\widehat{\beta_0}, \widehat{\beta_1})}{d\beta_1} = -\frac{2}{n}\sum(y_i - (\beta_0 + \beta_1 x)) * \beta_1$$

Unsupervised - Clustering

- 經典的分群演算法: K-means

```
from sklearn.cluster import KMeans

km = Kmeans(n_clusters = 5)

km.fit(df)
```

Means : 平均，此演算法跟很多(群)的平均有關

K-means

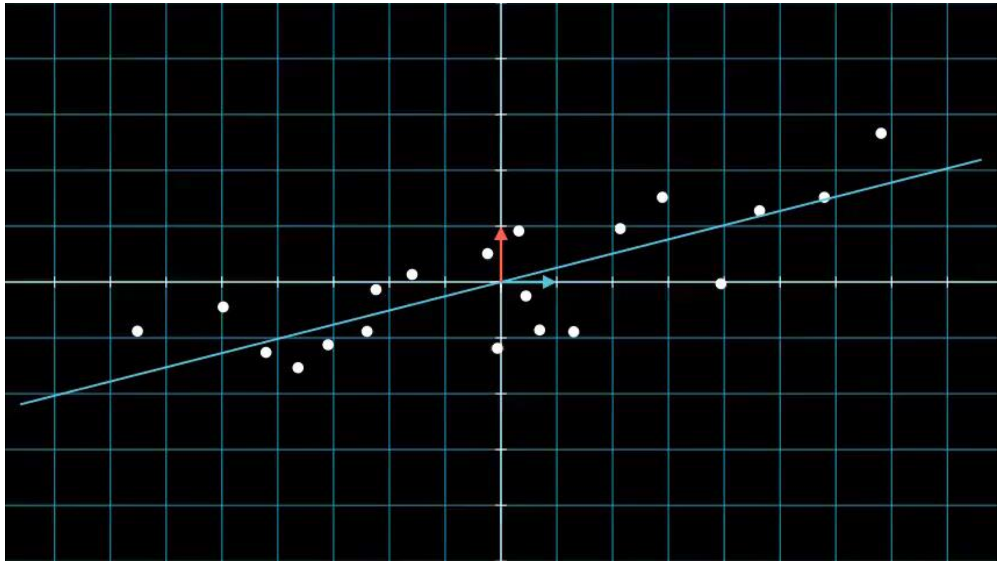
K : 常數，表示我們想要分成K群

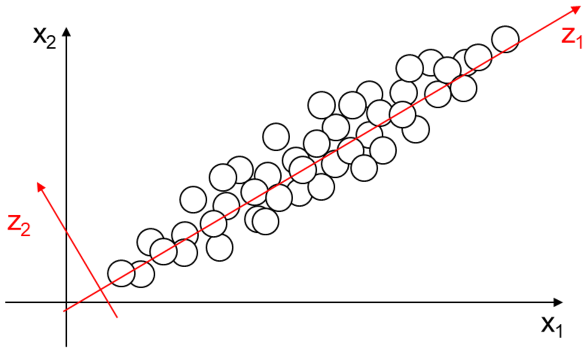
1. 需要設定要分成幾群，k = 群的數量。
2. 隨機分配 k 個樣本，作為群心。
3. 每個樣本開始和這 k 個群心計算距離。
4. 將每個樣本分配給最近的群心。
5. 根據新的樣本群，來更新群心。
6. 重複3~5步驟直到不再有變動 (收斂)

Unsupervised - Dimensionality Reduction

PCA 先備知識:
- 線性組合 Linear Combination
- 線性轉換 Linear Transformation
- 特徵向量/值 Eigenvectors 與 Eigenvalues
- 拉格朗日乘數 Lagrange Multiplier

- 有很多方法可以做降維，這邊介紹最經典的主成分分析 (Principal Component Analysis, PCA)
- 當資料的特徵、變數維度很高，可透過降維的方法，只取出最有用的特徵，減少分析的難度、避免過擬合，並提高資料的可視化
- 將原本的特徵空間以線性轉換到另一個由正交向量組成的座標系統，保留訊息量(變異數)較大主成分，忽略訊息量較少的成分

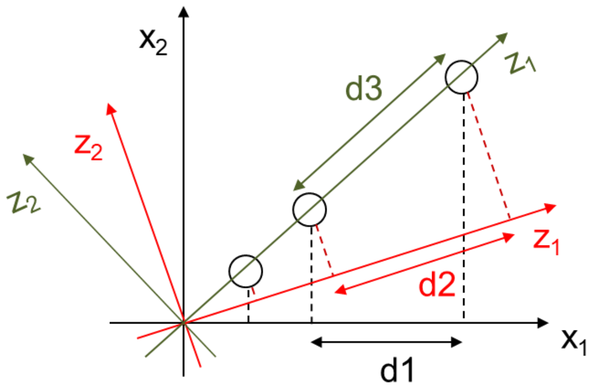




轉換到新的座標後，要能保留資料的最大差異性(最大化資料變異數)

	S1	S2	S3	S4	S5	S6	S7	S8
特徵一	1	2	3	4	5	6	7	8
特徵二	2	4	7	8	10	11	14	16

Unsupervised - Dimensionality Reduction



要保留最大訊息量 > 資料間距離要最遠 > 變異數要大

- 須將 d 維空間的樣本轉換到 d’ 維度, 其中 $d' \leq d$
- 原樣本 $X = (x_1, x_2, \ldots x_m) \in R^{d*m}$
- 投影所需要用到的轉移矩陣 $W = (w^1, w^2, \ldots, w^{d'}) \in R^{d*d'}$
- 投影後得到的樣本

$$Z = (z_1, z_2 \ldots, z_m) \in R^{d'*m}$$

$$z = Wx$$

```
from sklearn.decomposition import PCA
pca = PCA(n_component=0.5)
data = pca.fit_transform([2,8,4,5,9,3],[6,3,0,8,7,1],[5,4,9,1,8,2])
```

Unsupervised - Dimensionality Reduction

樣本數

$X = (x_1, x_2, \dots x_m) \in R^{d*m}$

x_1 是特徵向量

$$x_1 = \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ \dots \\ x_{1d} \end{bmatrix}$$

轉移矩陣

$$W = \begin{bmatrix} w_{11}\ w_{12} \dots w_{1d} \\ w_{21}\ w_{22} \dots w_{2d} \\ w_{31}\ w_{32} \dots w_{3d} \\ \dots \\ w_{d'1}\ w_{d'2} \dots w_{d'd} \end{bmatrix} = \begin{bmatrix} (w^1) \\ (w^2) \\ (w^3) \\ \dots \\ (w^{d'}) \end{bmatrix} \in R^{d'*d}$$

W 是一種轉移矩陣
將 d 維轉換到 d'維

$$\begin{bmatrix} z_{11} \\ z_{12} \\ z_{13} \\ \dots \\ z_{1d'} \end{bmatrix} = \begin{bmatrix} w_{11}\ w_{12} \dots w_{1d} \\ w_{21}\ w_{22} \dots w_{2d} \\ w_{31}\ w_{32} \dots w_{3d} \\ \dots \\ w_{d'1}\ w_{d'2} \dots w_{d'd} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ \dots \\ x_{1d} \end{bmatrix}$$

$$z = Wx$$

樣本數

$Z = (z_1, z_2 \dots, z_m) \in R^{d'*m}$

$$z_1 = \begin{bmatrix} z_{11} \\ z_{12} \\ z_{13} \\ \dots \\ z_{1d'} \end{bmatrix}$$

Unsupervised - Dimensionality Reduction

$$z_1 = w^1 x$$

$$\overline{z_1} = \frac{1}{n}\sum z_1 = \frac{1}{n}\sum w^1 x = w^1 \frac{1}{n}\sum x = w^1 \overline{\mathrm{x}}$$

$$var(z_1) = \sum(z_1 - \overline{z_1})^2 = \sum(w^1(x - \overline{\mathrm{x}}))^2$$

因為: $(ab)^2 = (a^T b)^2 = a^T \underbrace{b a^T}_{\text{scaler}} b = a^T b (a^T b)^T = a^T b b^T a$

所以: $var(z_1) = \sum (w^1)^T (x - \overline{\mathrm{x}})(x - \overline{\mathrm{x}})^T w^1$

$$= (w^1)^T \left(\sum (x - \overline{\mathrm{x}})(x - \overline{\mathrm{x}})^T\right) w^1$$

$$= \underline{(w^1)^T cov(x) w^1}$$

轉移矩陣

$$W = \begin{bmatrix} w_{11} \; w_{12} \ldots w_{1d} \\ w_{21} \; w_{22} \ldots w_{2d} \\ w_{31} \; w_{32} \ldots w_{3d} \\ \ldots \\ w_{d'1} \; w_{d'2} \ldots w_{d'd} \end{bmatrix} = \begin{bmatrix} (w^1) \\ (w^2) \\ (w^3) \\ \ldots \\ (w^{d'}) \end{bmatrix} \in R^{d' * d}$$

$$\begin{bmatrix} z_{11} \\ z_{12} \\ z_{13} \\ \ldots \\ z_{1d'} \end{bmatrix} = \begin{bmatrix} w_{11} \; w_{12} \ldots w_{1d} \\ w_{21} \; w_{22} \ldots w_{2d} \\ w_{31} \; w_{32} \ldots w_{3d} \\ \ldots \\ w_{d'1} \; w_{d'2} \ldots w_{d'd} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ \ldots \\ x_{1d} \end{bmatrix}$$

我們希望每個樣本中的第一維特徵變異數越大越好

我們如果要轉換成d' 維，就需要知道 $w^1 \sim w^{d'}$

Unsupervised - Dimensionality Reduction

我們要找 w^1 極大化 $(w^1)^T\, cov(x)\, w^1$

會使用 Lagrange Multiplier : $L(w^1,\lambda) = (w^1)^T cov(x)(w^1) - \lambda((w^1)^T(w^1) - I)$

將目標函數轉換成拉格朗日方程式，原本的最佳化問題變成要找能讓拉格朗日方程式最大化的 w^1 跟 λ

$$\left.\begin{array}{l} \dfrac{dL(w^1,\lambda)}{dw_1^1} = 0 \\ \dfrac{dL(w^1,\lambda)}{dw_2^1} = 0 \\ \dfrac{dL(w^1,\lambda)}{dw_3^1} = 0 \\ \vdots \\ \dfrac{dL(w^1,\lambda)}{dw_n^1} = 0 \end{array}\right\} \quad 2cov(x)w^1 - 2\lambda w^1 = 0 \;\Rightarrow\; (cov(x) - \lambda I)w^1 = 0$$

$$\Rightarrow\; cov(x)w^1 = \lambda w^1$$

特徵向量 (eigenvector, W)　　特徵值 (eigenvalue,λ)

$\dfrac{dL(w^1,\lambda)}{d\lambda} = 0$ >>> $|w^1| = I$

$(w^1)^T\, cov(x)\, w^1 = (w^1)^T \lambda w^1 = \lambda\, (w^1)^T w^1 = \lambda$

Unsupervised - Dimensionality Reduction

目標:極大化 $(w^1)^T\, cov(x)\, w^1$

其中 $cov(x)w^1 = \alpha\, w^1$

$(w^1)^T\, cov(x)\, w^1 = (w^1)^T \alpha\, w^1 = \alpha\, (w^1)^T w^1 = \alpha$

結論: 共變異數矩陣的最大的特徵值對應的特徵向量，

就是可以使得轉換後變異數最大的轉移矩陣 w^1

演算法流程

輸入：

原始樣本資料集 $X = (x_1, x_2, \ldots, x_m) \in R^{d\times m}$

決定投影後空間維度 d' , $d' \leq d$

過程：

對所有樣本進行中心化運算： $\tilde{x} = \sum_i x_i = 0$

計算所有樣本的共變異數矩陣：$\sum_m X_m X_m^T$

對共變異數矩陣求特徵值 λ

取最大的 d' 個特徵值 $\lambda_1, \lambda_2, \ldots, \lambda_{d'}$ ，

求出相對應的特徵向量 $W = (w^1, w^2, \ldots, w^{d'}) \in R^{d\times d'}$

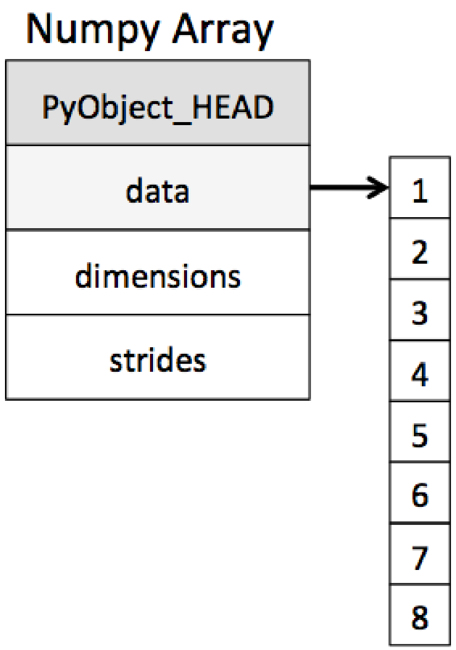
求出新樣本資料集 $Z = (z_1, z_2, \ldots, z_m) \in R^{d'\times m}$

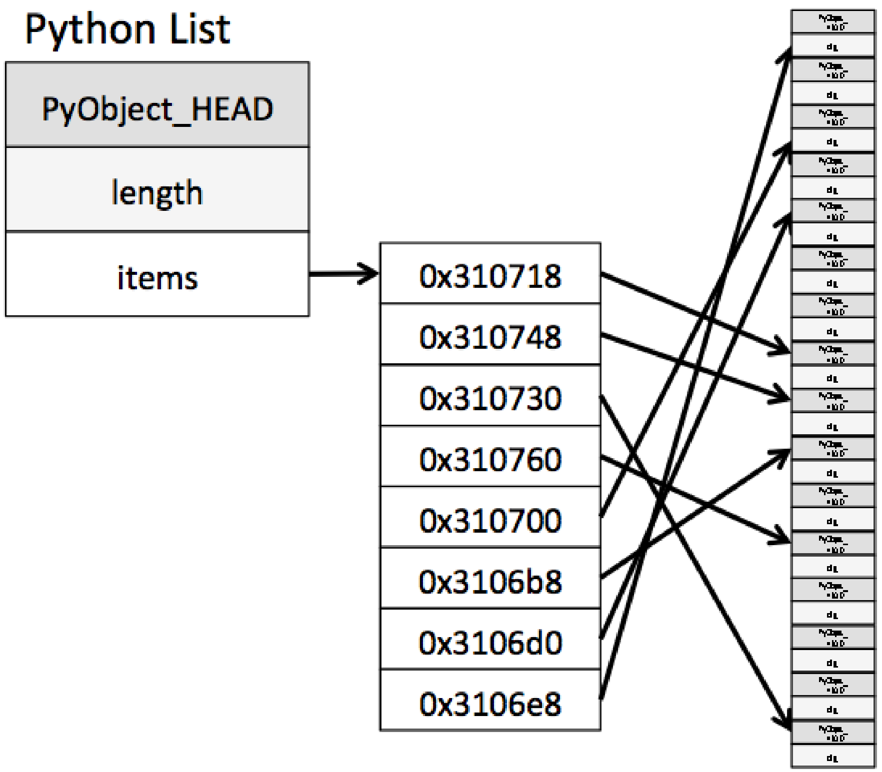
輸出：

轉換矩陣 $W = (w^1, w^2, \ldots, w^{d'}) \in R^{d\times d'}$

新樣本資料集 $Z = (z_1, z_2, \ldots, z_m) \in R^{d'\times m}$

Numpy Array v.s List





Numpy Array Operator

```
import numpy as np

# initial an 6*6 matrix
na = np.arange(1, 49).reshape(3, 4, 4)

print(na) # show na

print("Show the shape of na: ",na.shape)

print("======")
print(na[0,1,3])
print("======")
print(na[:,:2,:2])
print("======")
print(na[::2,::2,::2])
print("======")
print(na[-3::2,-3::2,-3::2])
```

Channel, Row , Col

```
[[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]
  [13 14 15 16]]

 [[17 18 19 20]
  [21 22 23 24]
  [25 26 27 28]
  [29 30 31 32]]

 [[33 34 35 36]
  [37 38 39 40]
  [41 42 43 44]
  [45 46 47 48]]]
```

起始位置(包含)

間隔

0:0:0

結束位置(不包含)

```
Show the shape of na:  (3, 4, 4)
======
8
======
[[[ 1  2]
  [ 5  6]]

 [[17 18]
  [21 22]]

 [[33 34]
  [37 38]]]
======
[[[ 1  3]
  [ 9 11]]

 [[33 35]
  [41 43]]]
======
[[[ 6  8]
  [14 16]]

 [[38 40]
  [46 48]]]
```

img&np_operate.zip

Numpy Array Operator

```python
import numpy as np

# initial an 6*6 matrix
na = np.arange(1, 49).reshape(3, 4, 4)

print(na) # show na

print("Show the shape of na: ",na.shape)

print("======")
print(na[0,1,3])
print("======")
print(na[:,:2,:2])
print("======")
print(na[::2,::2,::2])
print("======")
print(na[-3::2,-3::2,-3::2])
```

Channel, Row , Col

```
[[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]
  [13 14 15 16]]

 [[17 18 19 20]
  [21 22 23 24]
  [25 26 27 28]
  [29 30 31 32]]

 [[33 34 35 36]
  [37 38 39 40]
  [41 42 43 44]
  [45 46 47 48]]]
```

起始位置(包含)

間隔

0:0:0

結束位置(不包含)

```
Show the shape of na:  (3, 4, 4)
======
8
======
[[[ 1  2]
  [ 5  6]]

 [[17 18]
  [21 22]]

 [[33 34]
  [37 38]]]
======
[[[ 1  3]
  [ 9 11]]

 [[33 35]
  [41 43]]]
======
[[[ 6  8]
  [14 16]]

 [[38 40]
  [46 48]]]
```

img&np_operate.zip

Numpy Array Operator

```
import numpy as np

# initial an 6*6 matrix
na = np.arange(1, 49).reshape(3, 4, 4)

print(na) # show na

print("Show the shape of na: ",na.shape)

print("======")
print(na[0,1,3])
print("======")
print(na[:,:2,:2])
print("======")
print(na[::2,::2,::2])
print("======")
print(na[-3::2,-3::2,-3::2])
```

Channel, Row , Col

```
[[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]
  [13 14 15 16]]

 [[17 18 19 20]
  [21 22 23 24]
  [25 26 27 28]
  [29 30 31 32]]

 [[33 34 35 36]
  [37 38 39 40]
  [41 42 43 44]
  [45 46 47 48]]]
```

起始位置(包含)

間隔

0:0:0

結束位置(不包含)

```
Show the shape of na:  (3, 4, 4)
======
8
======
[[[ 1  2]
  [ 5  6]]

 [[17 18]
  [21 22]]

 [[33 34]
  [37 38]]]
======
[[[ 1  3]
  [ 9 11]]

 [[33 35]
  [41 43]]]
======
[[[ 6  8]
  [14 16]]

 [[38 40]
  [46 48]]]
```

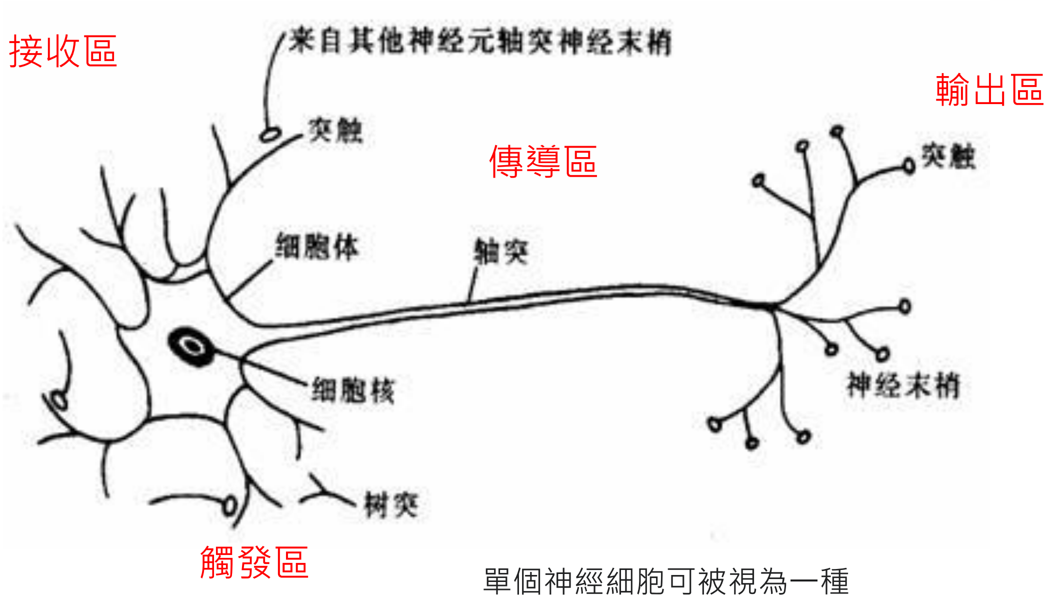
img&np_operate.zip

Deep Learning Intro.

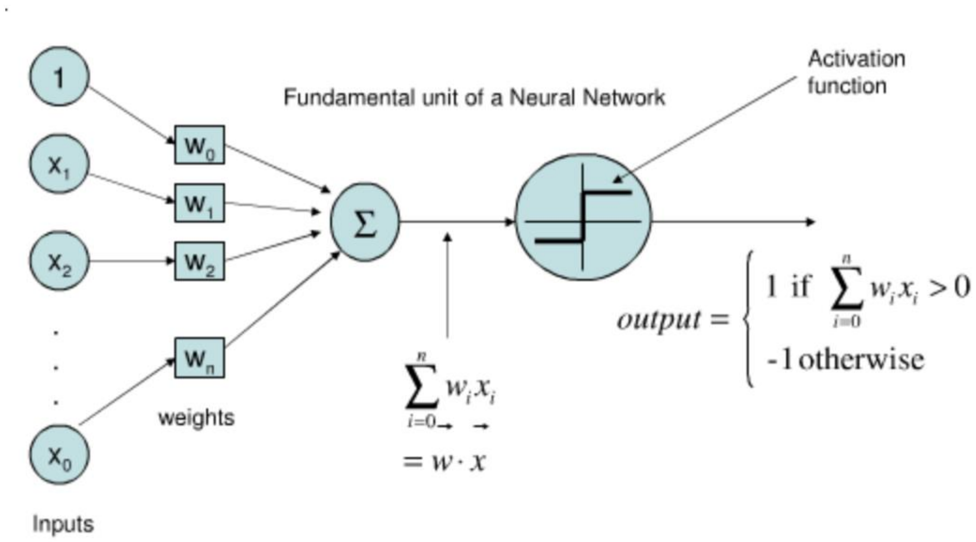
Outline

- **Deep Learning Intro. (about 1.5 hour)**
  - ANN & Back Propagation
  - Computer Vision & CNN
  - Padding & Pooling & Flatten
  - Activation Function
  - Pytorch - Build Network
  - Homework - Build an image classifier
    - Loss Function
    - Dataset & Dataloder
    - Augmentation

ANN



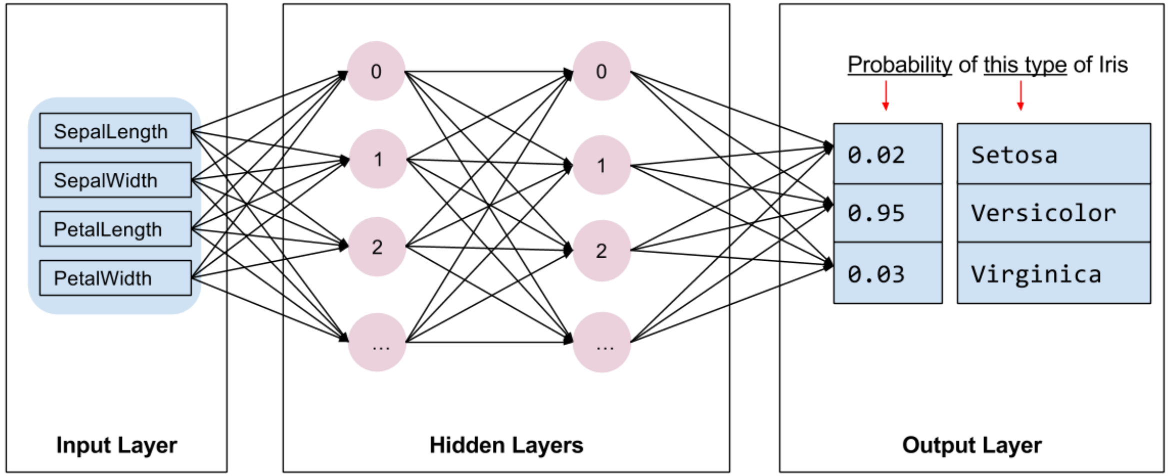
單個神經細胞可被視為一種
只有兩種狀態的機器:
激動時為『是』，而未激動時為『否』。

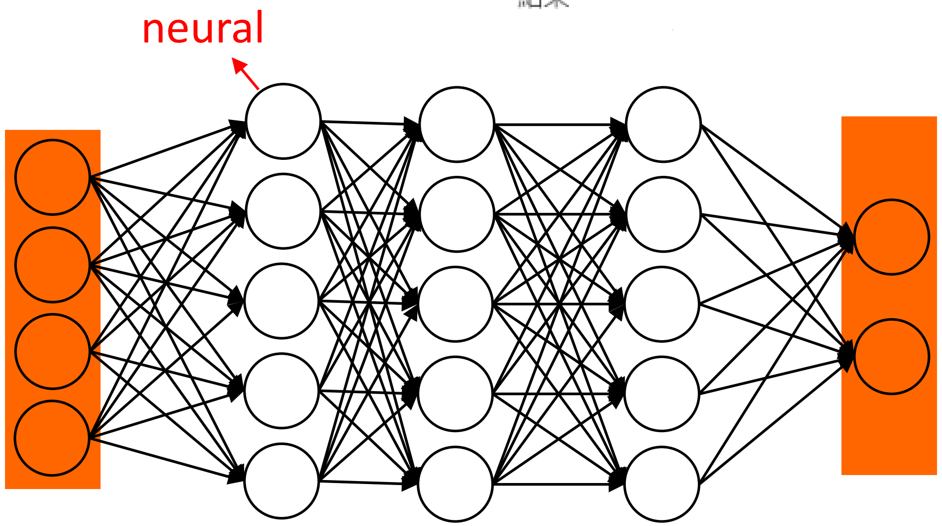


Multi-Layer Perceptron

- 包含輸入層 (Input layer)、隱藏層 (Hidden layer) 和輸出層 (Output layer)
- 深度神經網路泛指 MLP中將隱藏層加深。
- 通常利用 backpropagation 來訓練

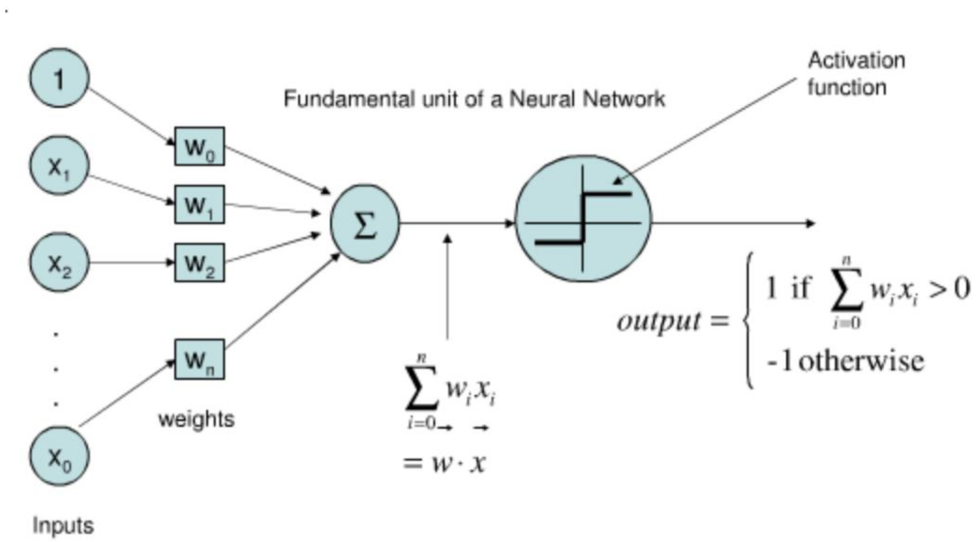
參數初始化
↓
正向傳遞 (Forward)
↓
計算成本
↓
反向傳遞 (Backpropagation)
↓
更新參數
↓
檢查是否結束迭代 —(否)→ 正向傳遞
↓(是)
結束

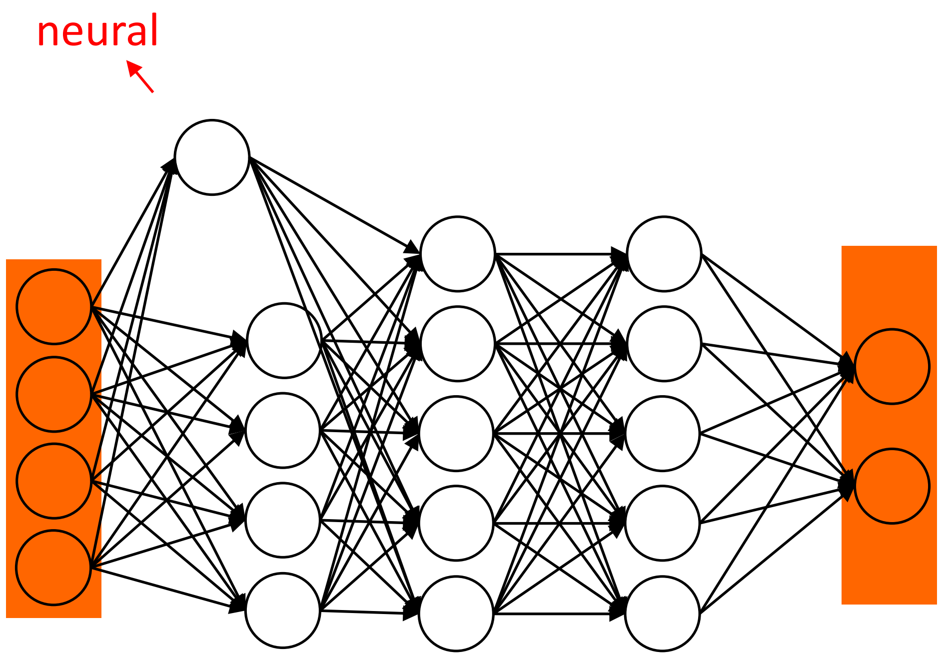




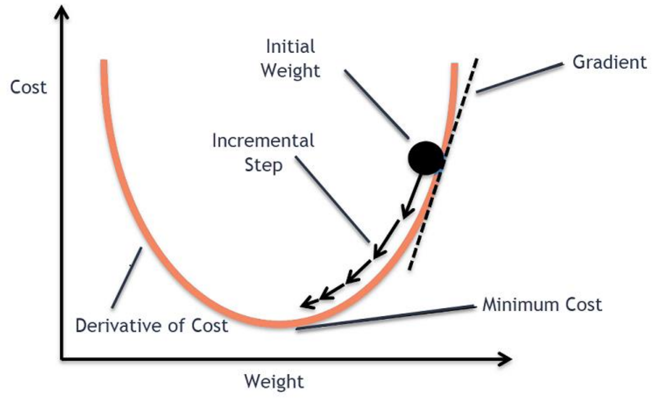
Backpropagation

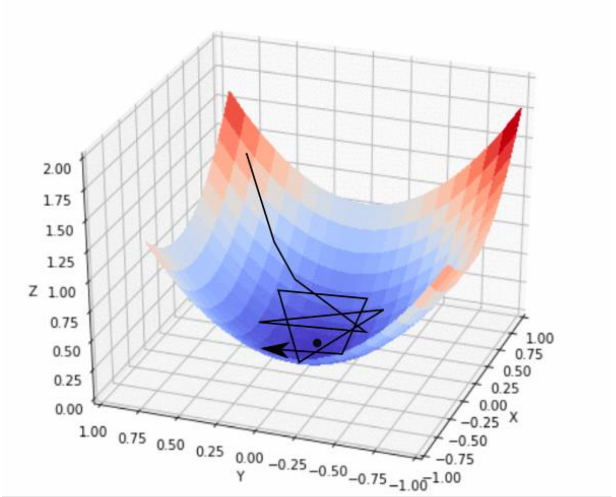
- Compute gradients of expressions based on the chain rule





Backpropagation





<u>Interpreting the gradient</u>

$$\nabla f(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h\nabla f(x)$$

<u>Example</u>

$$f(x) = 3x^2 \to \nabla f(x) = \lim_{h \to 0} \frac{3(x+h)^2 - 3x^2}{h}$$

$$= 3 \lim_{h \to 0} \frac{x^2 + 2xh + h^2 - x^2}{h} = 3 \lim_{h \to 0} 2x + h = 6x$$

$$if\ x = 2, \nabla f(2) = 12$$

$$x + h \to f(x) \text{ will increase by 12h}$$

Backpropagation

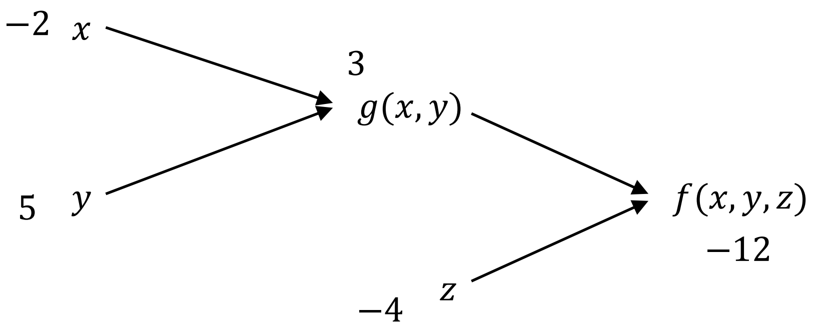
Chain Rule - Example

$$\frac{dg(x,y)}{dx} = 1 \qquad \frac{dg(x,y)}{dy} = 1$$

$$f(x,y,z) = (x+y)z$$

$$let\ g(x,y) = x+y \rightarrow f(x,y,z) = g(x,y)z$$

$$\frac{df(x,y,z)}{dx} = \frac{df(g(x,y),z)}{dg(x,y)}\frac{dg(x,y)}{dx} = z$$



$$\frac{df(x,y,z)}{dy} = \frac{df(g(x,y),z)}{dg(x,y)}\frac{dg(x,y)}{dy} = z$$

$$\frac{df(x,y,z)}{dz} = \frac{df(g(x,y),z)}{dz} = g(x,y)$$

Backpropagation

Chain Rule - Example

$$\frac{dg(x,y)}{dx} = 1 \qquad \frac{dg(x,y)}{dy} = 1$$

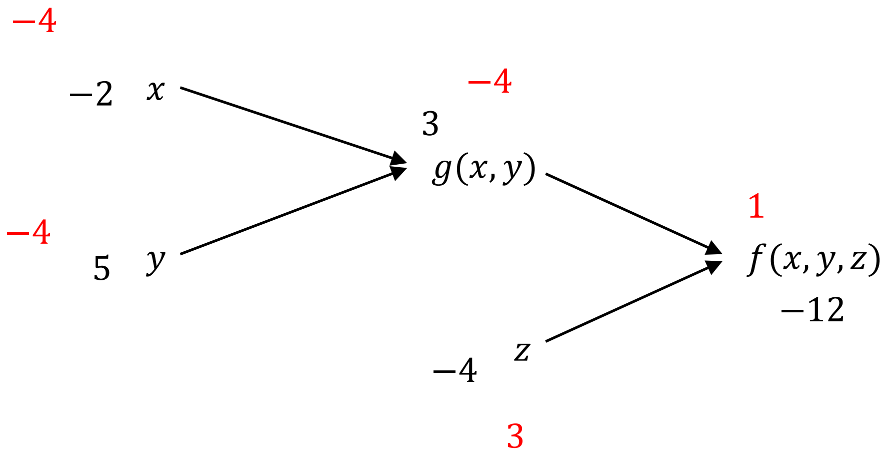
$$f(x,y,z) = (x+y)z$$

$$let\ g(x,y) = x + y \rightarrow f(x,y,z) = g(x,y)z$$

$$\frac{df(x,y,z)}{dx} = \frac{df(g(x,y),z)}{dg(x,y)}\frac{dg(x,y)}{dx} = z$$

$$\frac{df(x,y,z)}{dy} = \frac{df(g(x,y),z)}{dg(x,y)}\frac{dg(x,y)}{dy} = z$$

$$\frac{df(x,y,z)}{dz} = \frac{df(g(x,y),z)}{dz} = g(x,y)$$



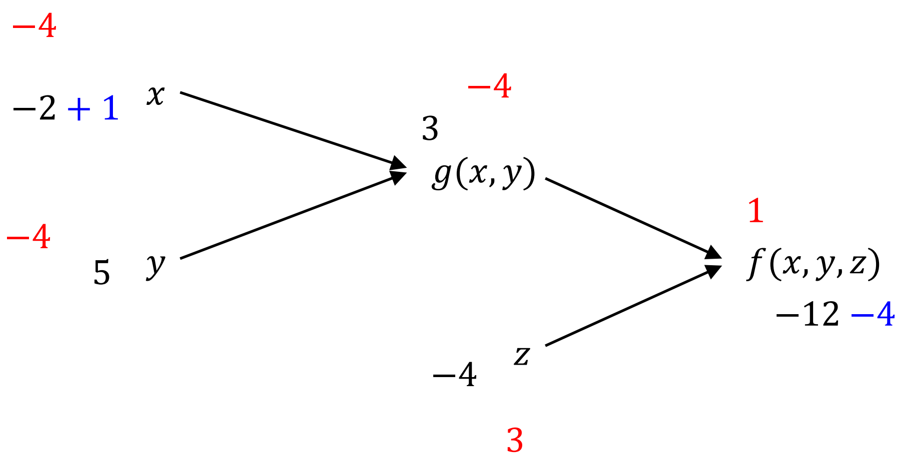
在 x = -2, y=5, z=-4 的情況下，Gradient 值

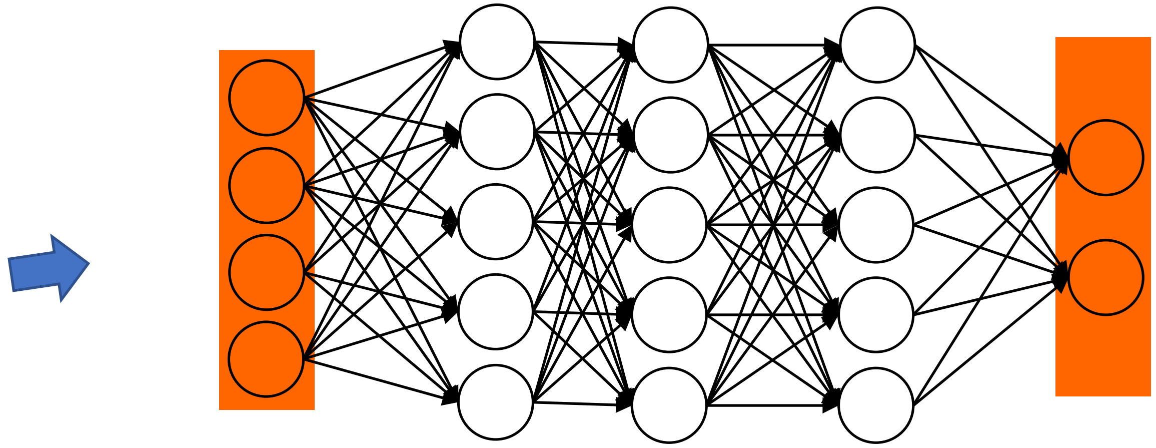
Backpropagation

Chain Rule - Example

$$f(x, y, z) = (x + y)z$$

$$let\ g(x, y) = x + y \rightarrow f(x, y, z) = g(x, y)z$$

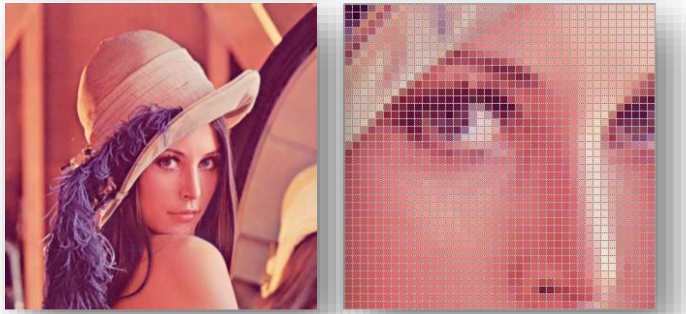




在 x = -2, y=5, z=-4 的情況下，Gradient 值

Computer Vision

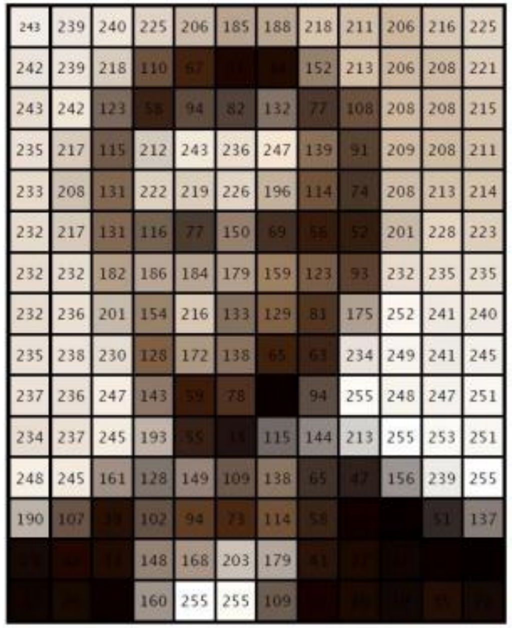
- What is an image?



Red plane

Green plane

Blue plane



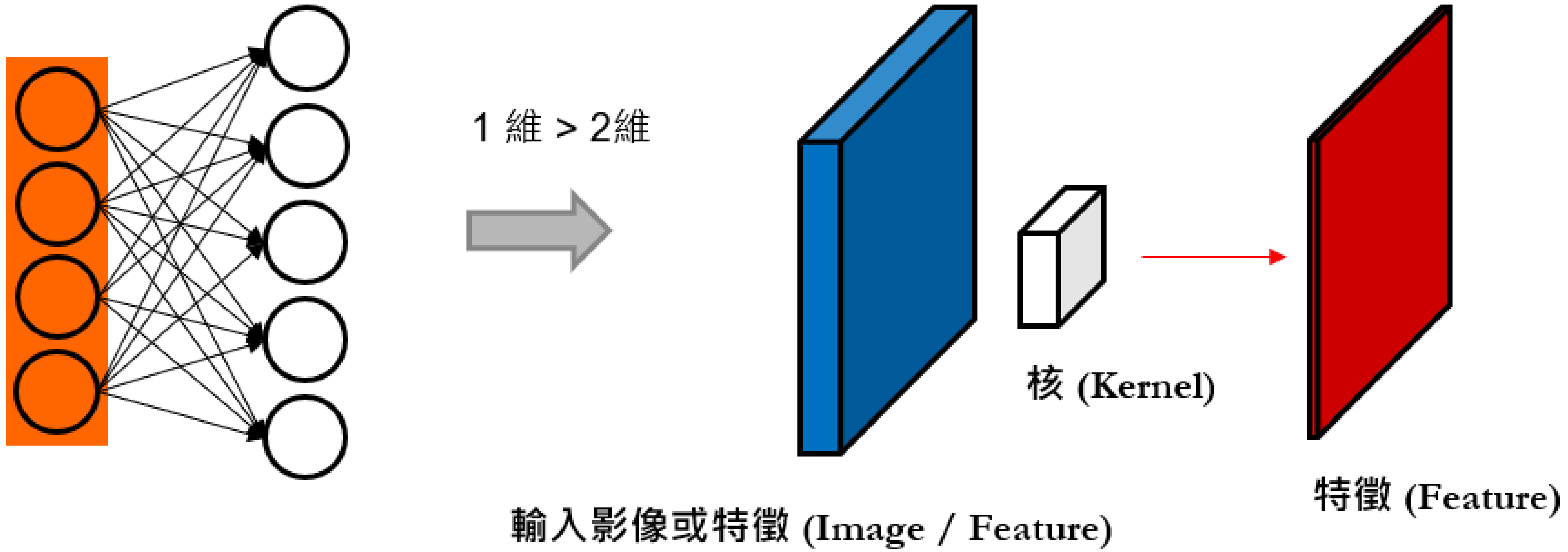
0 ~ 255 (8 bits)
3 channel

深度學習在影像的應用

近年來深度學習在影像上有很多有用的應用，可以分成以下幾個等級。

電腦視覺 Computer (Machine) Vision	High Level	Image → Detection/Recognition
影像分析 Image Analysis	↑	Image → Features/Characteristics
影像處理 Image Processing	Low Level	Image → Image

Convolution



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

Kernel 1

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

Kernel 2

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

Kernel 1

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

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

Convolution

Stride = 1
Kernel = 3*3

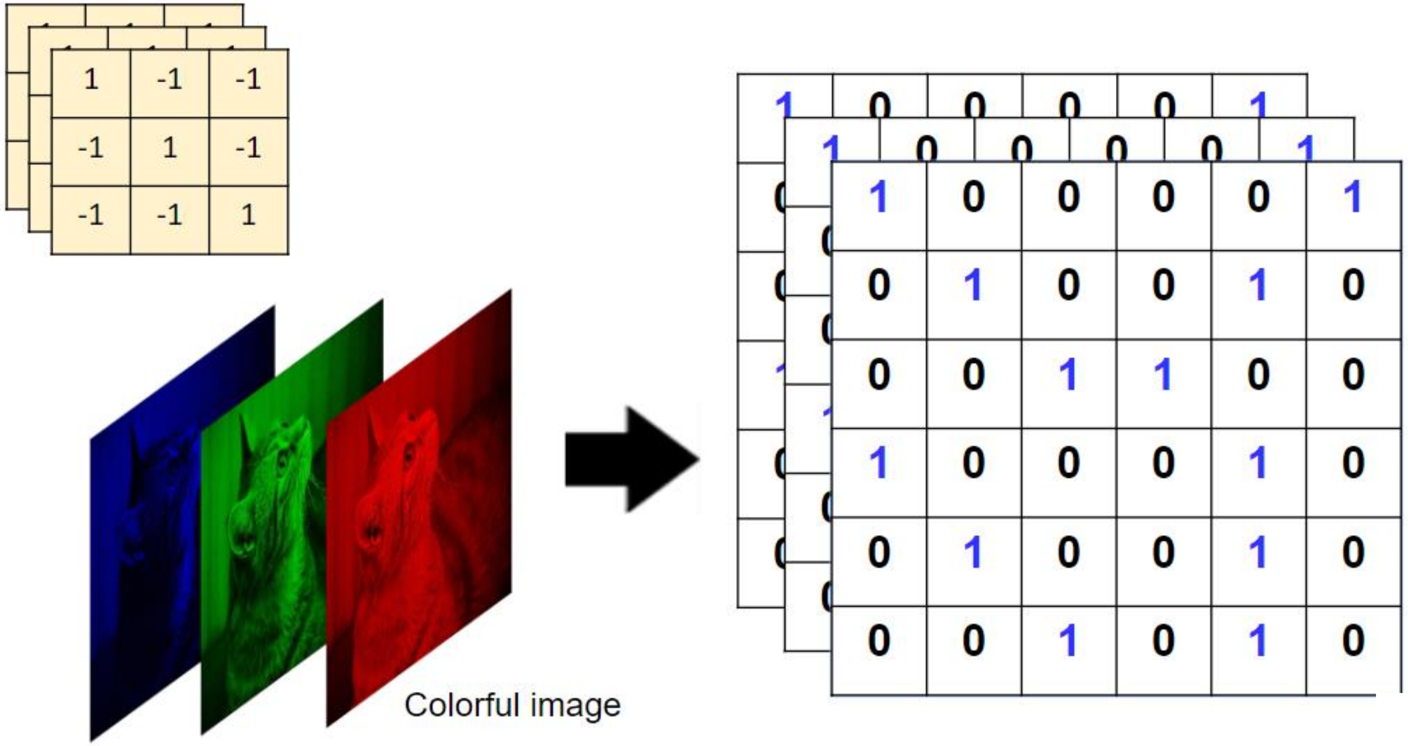
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

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

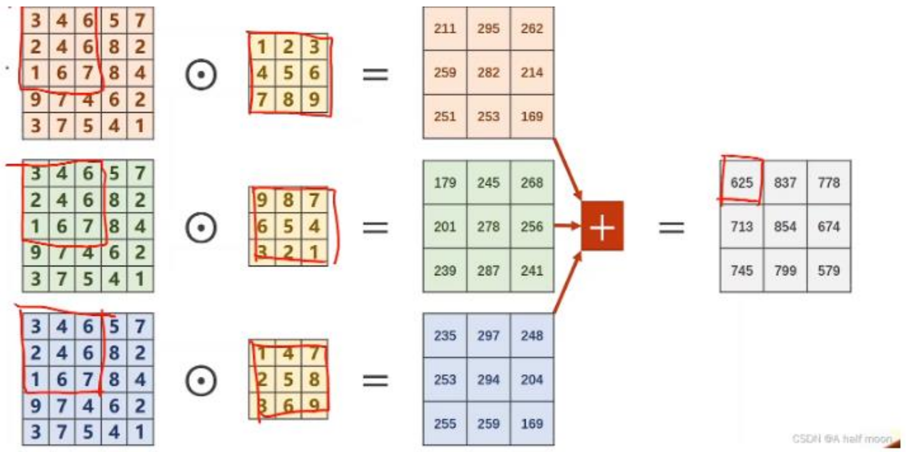
Kernel 2

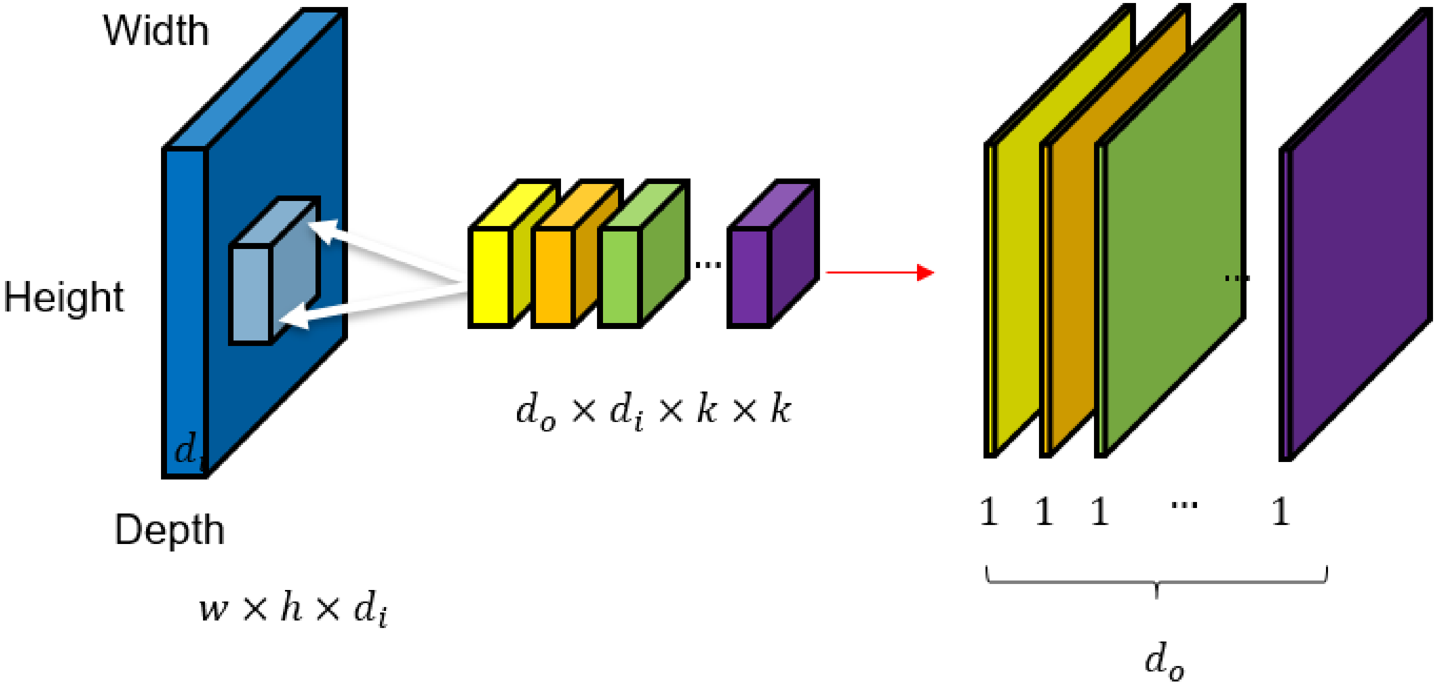
-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	-4	3

Colorful image - Convolution



多通道卷積 (Multi-Channel Convolution)

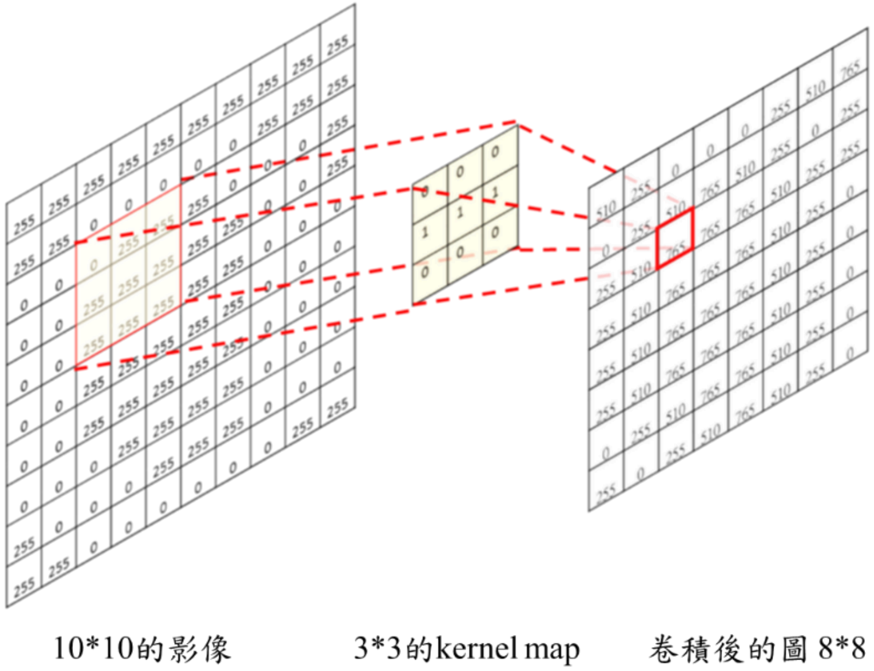


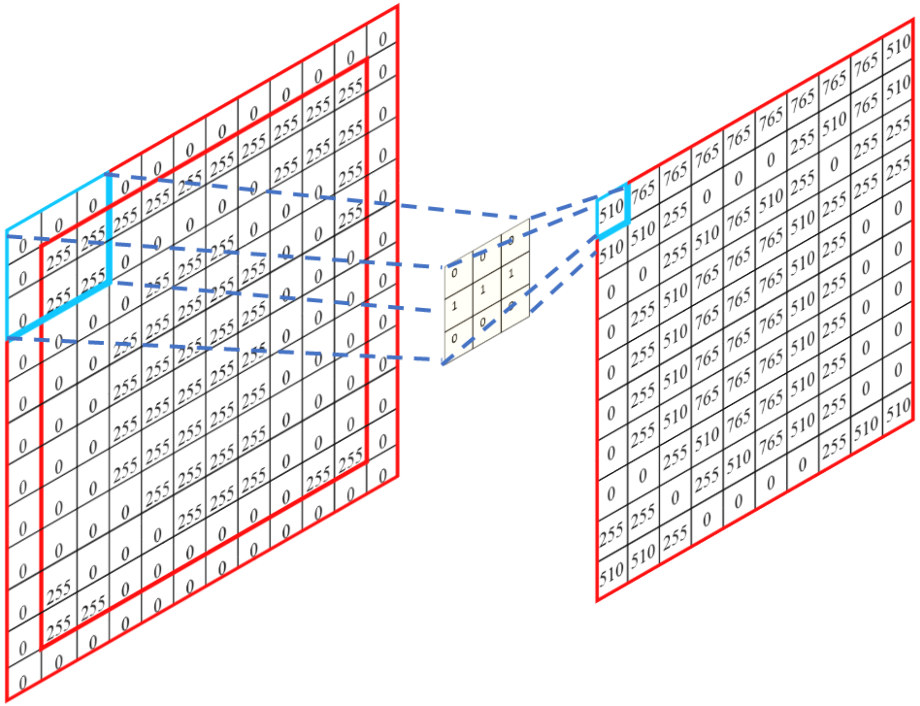


https://blog.csdn.net/weixin_44064434/article/details/123647168

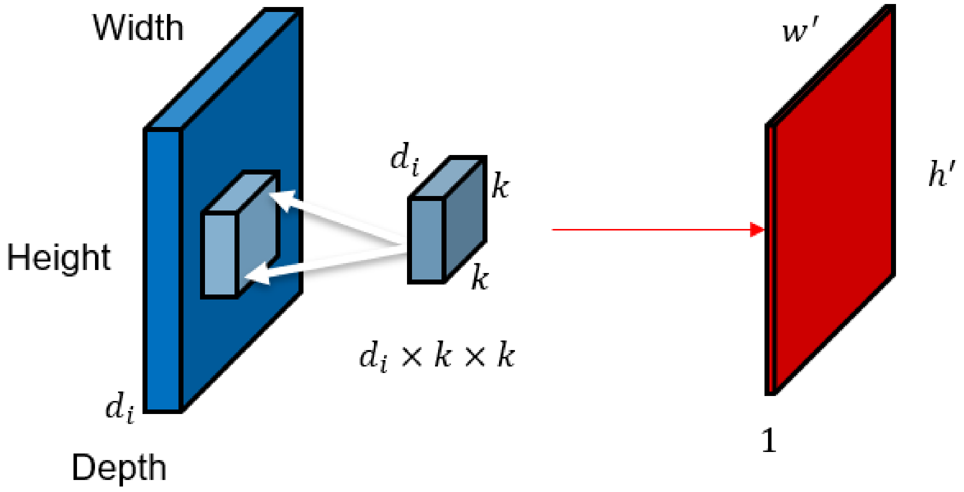
Padding

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





Convolution



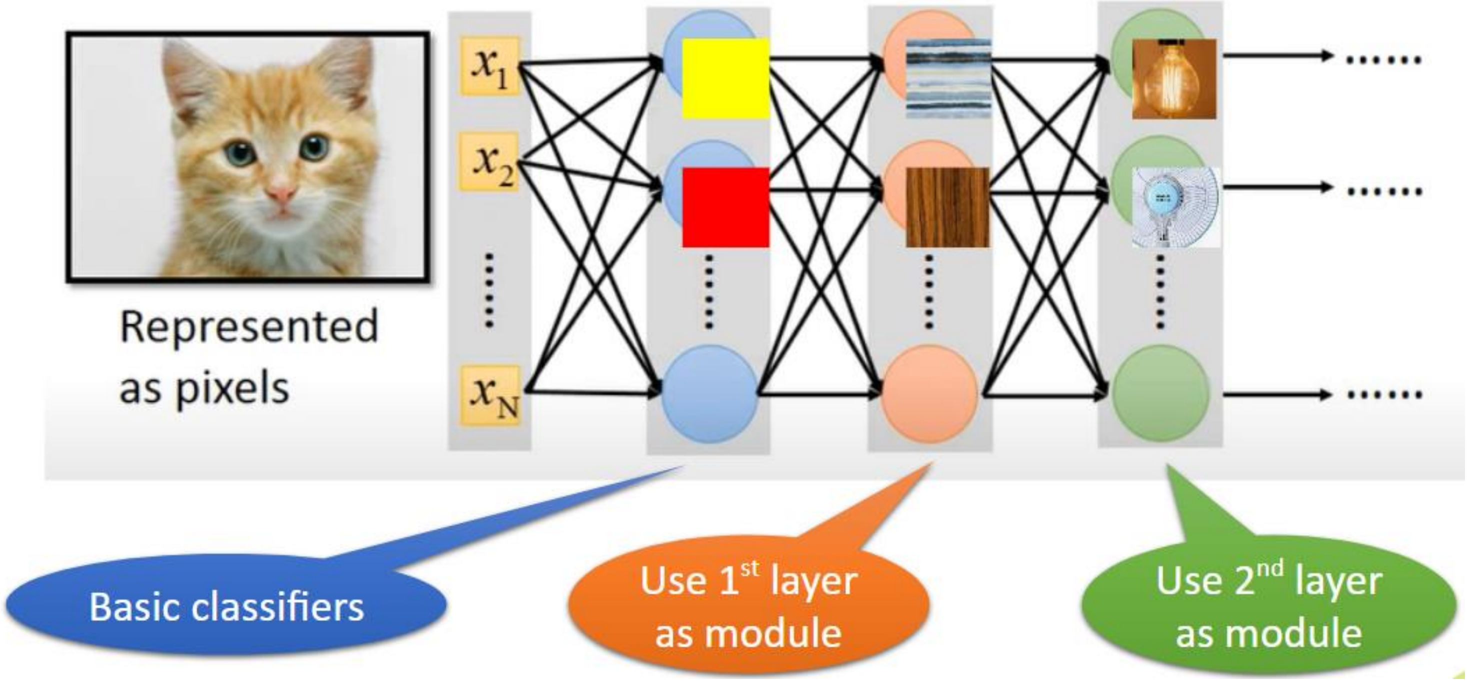
Stride=1, Kernel Size =k, pad = $\left\lfloor \frac{k}{2} \right\rfloor$

$$w' = \begin{cases} w - 2\left\lfloor \frac{k}{2} \right\rfloor, & \text{w/o padding;} \\ w, & \text{w padding.} \end{cases}$$

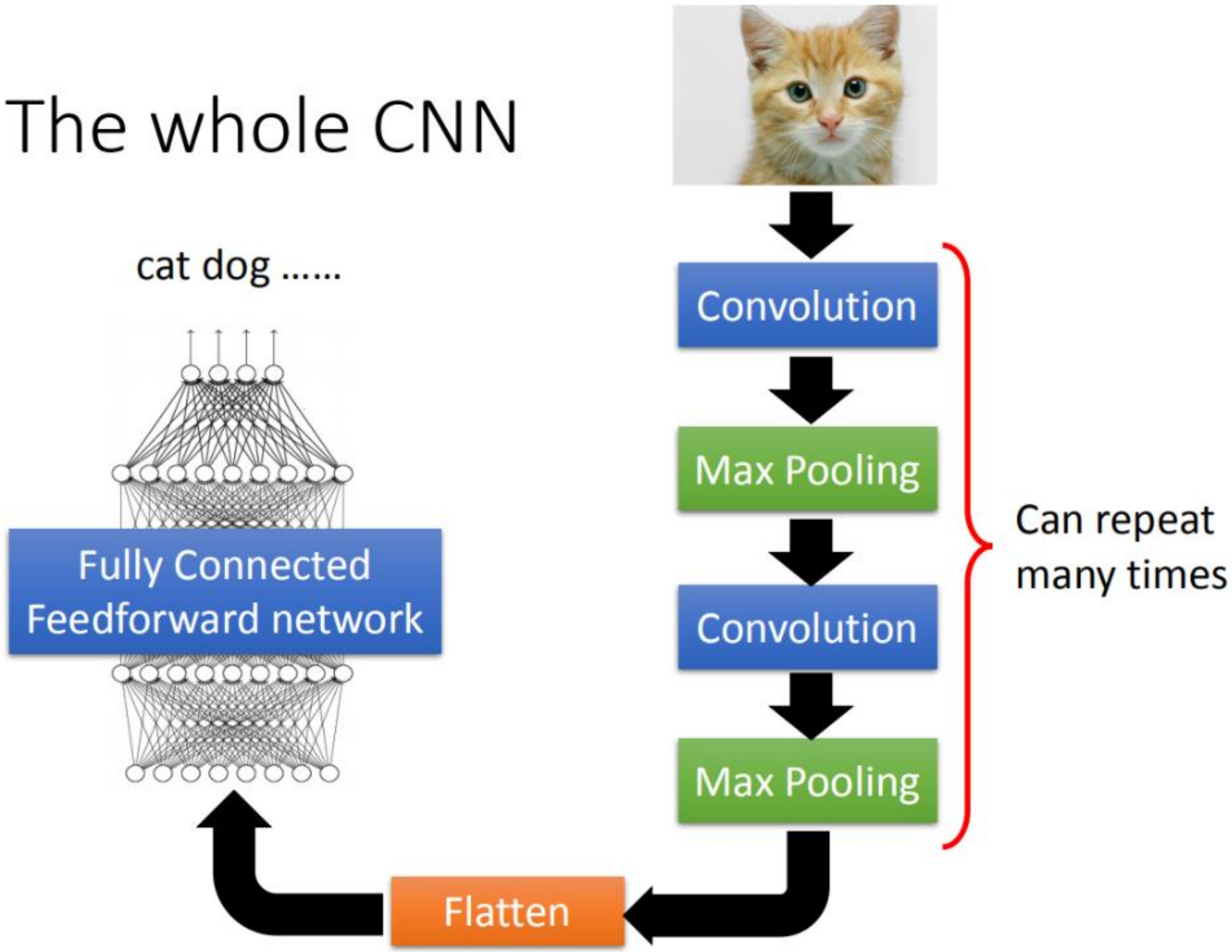
$$h' = \begin{cases} h - 2\left\lfloor \frac{k}{2} \right\rfloor, & \text{w/o padding;} \\ h, & \text{w padding.} \end{cases}$$

$$w' = floor\left(\frac{W + 2 \times pad - ks}{S}\right) + 1$$

Convolution Neural Network



The whole CNN



Max Pooling

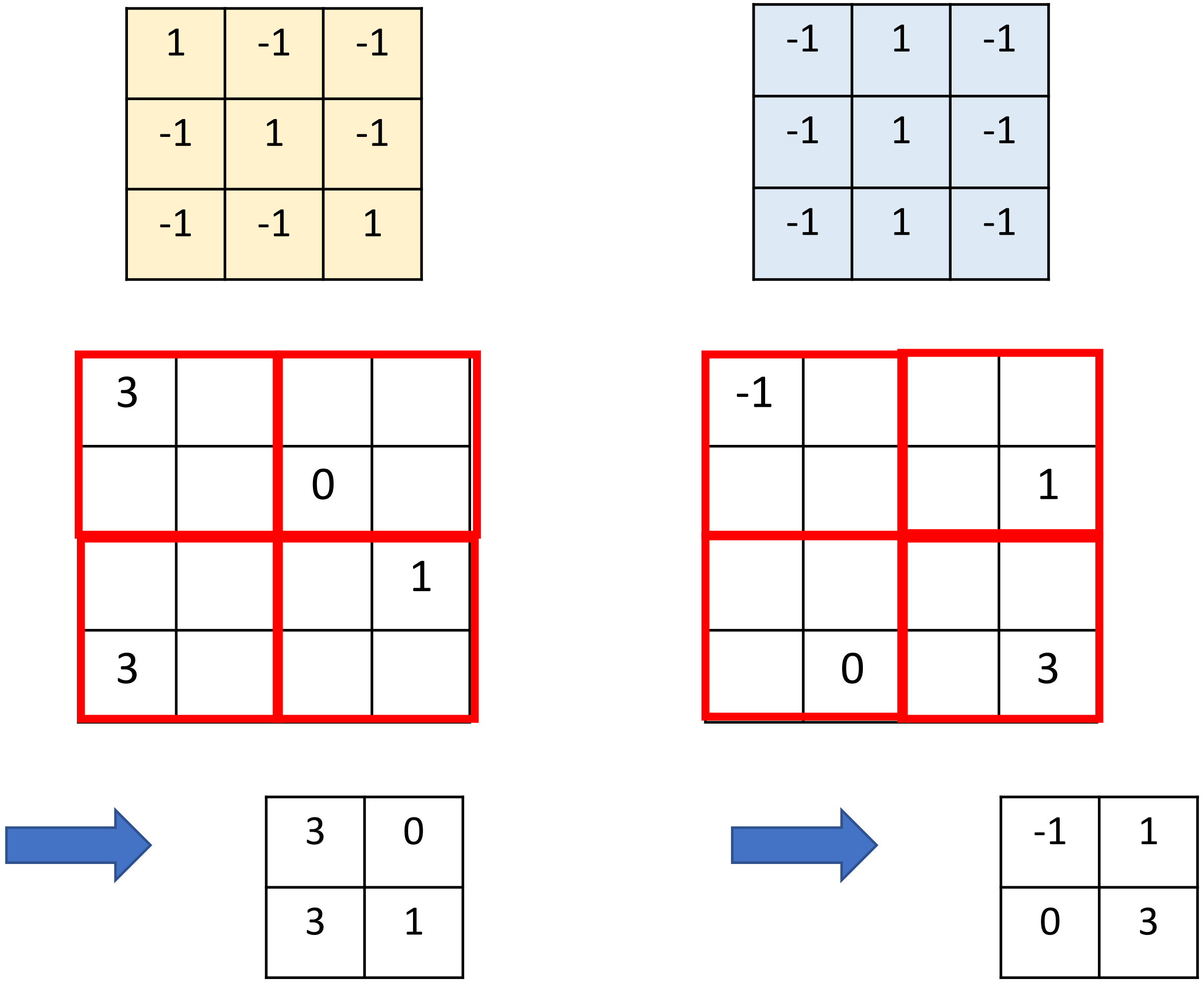
1	-1	-1
-1	1	-1
-1	-1	1

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

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

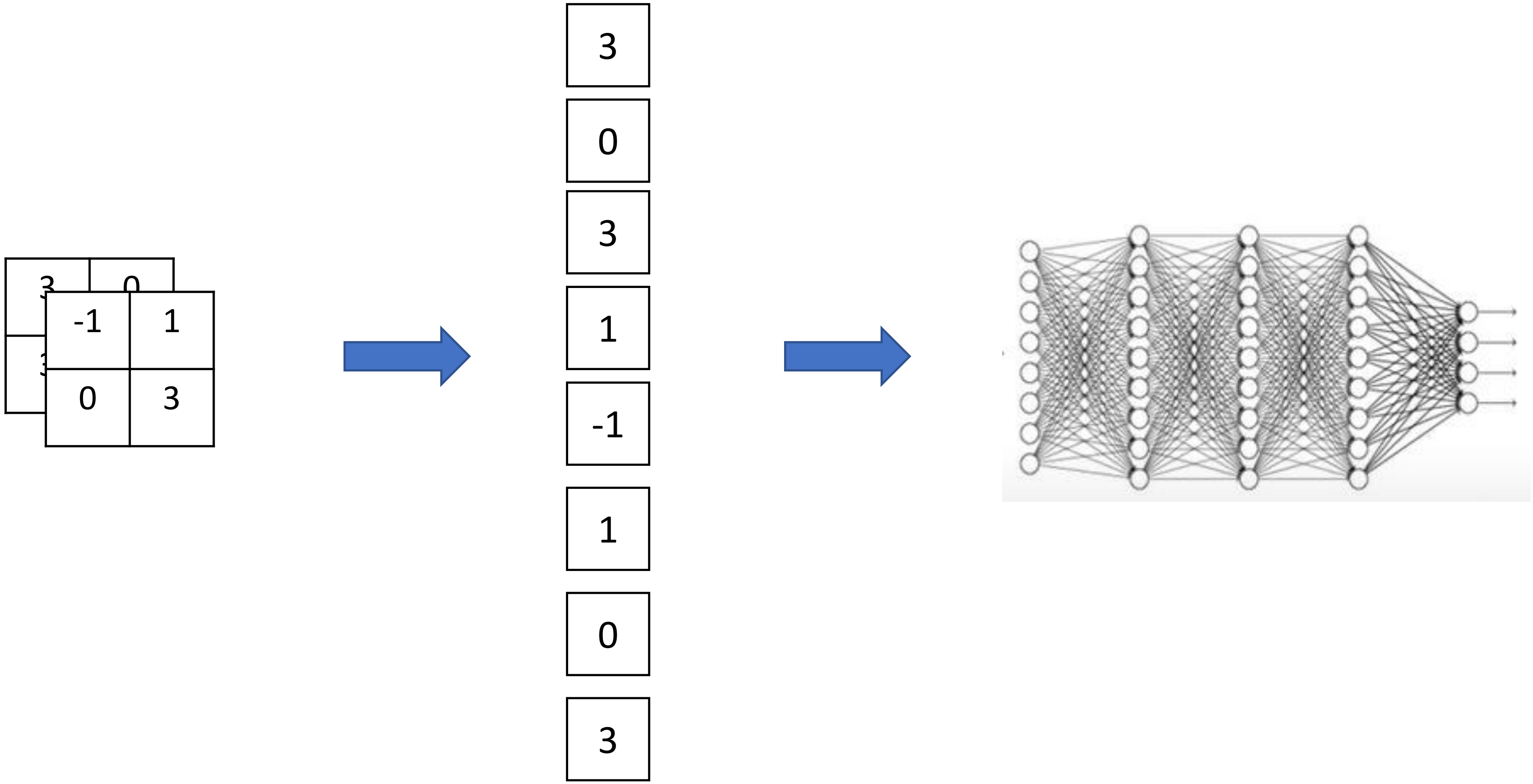
-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	-4	3

Max Pooling

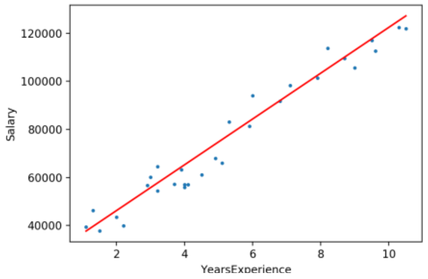


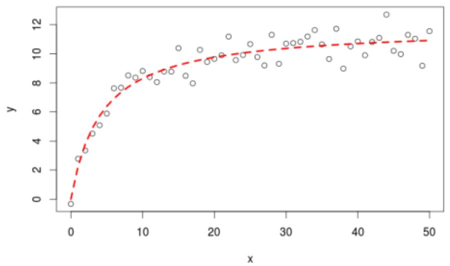
- Average Pooling
- Median Pooling
- Min Pooling

Flatten



激活函數 (Activation function)

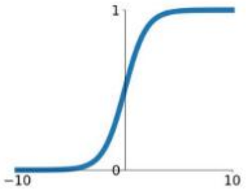




目的：做非線性轉換

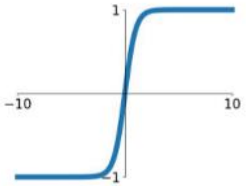
Sigmoid

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



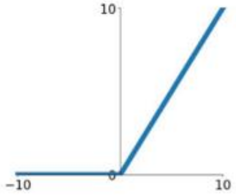
tanh

$\tanh(x)$



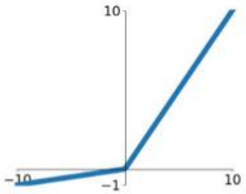
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

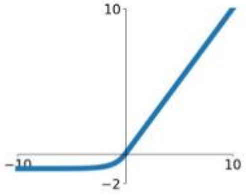


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

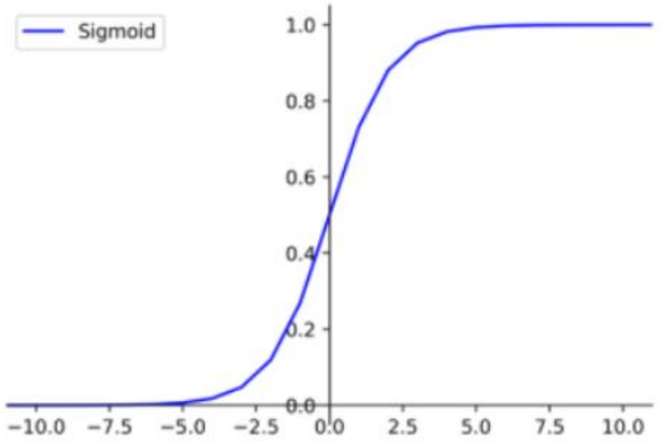


Sigmoid

- 常被用於二分類問題的網路模型中，會輸出範圍介於 [0, 1] (大於0.5 & 小於0.5)
- 是 gradient-based method，所以 activation 是可微的函數比較好計算

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

$$\Rightarrow \sigma'(x) = \sigma(x)\left(1-\sigma(x)\right) = \frac{1}{1+e^{-x}} * \left(1-\frac{1}{1+e^{-x}}\right)$$



- 指數運算較為耗時
- 當 X 很大或很小，會趨近於0 > 造成梯度消失 (gradient vanishing)

Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

- 將一組向量映射為每個向量當中的元素，都位於 (0, 1) 之間。
- 每個分類的機率分佈，這個向量的所有元素相加總和應為 1。
- Softmax 通常加在最後一層

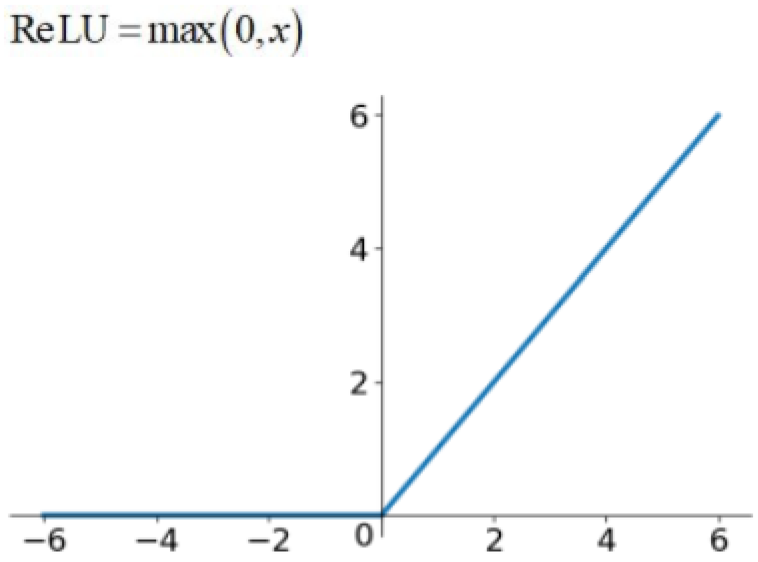
```python
import numpy as np

inputs = np.array([1, 4, 9, 7, 5])

def softmax(inputs):
    return np.exp(inputs)/sum(np.exp(inputs))

outputs = softmax(inputs)
for n in range(len(outputs)):
    print('{} -> {}'.format(inputs[n], outputs[n]))
```

```
1 -> 0.00028901145493871657
4 -> 0.005804950249395781
9 -> 0.8615310049461178
7 -> 0.11659554257150641
5 -> 0.015779490778041354
```

Rectified Linear Unit (ReLU)

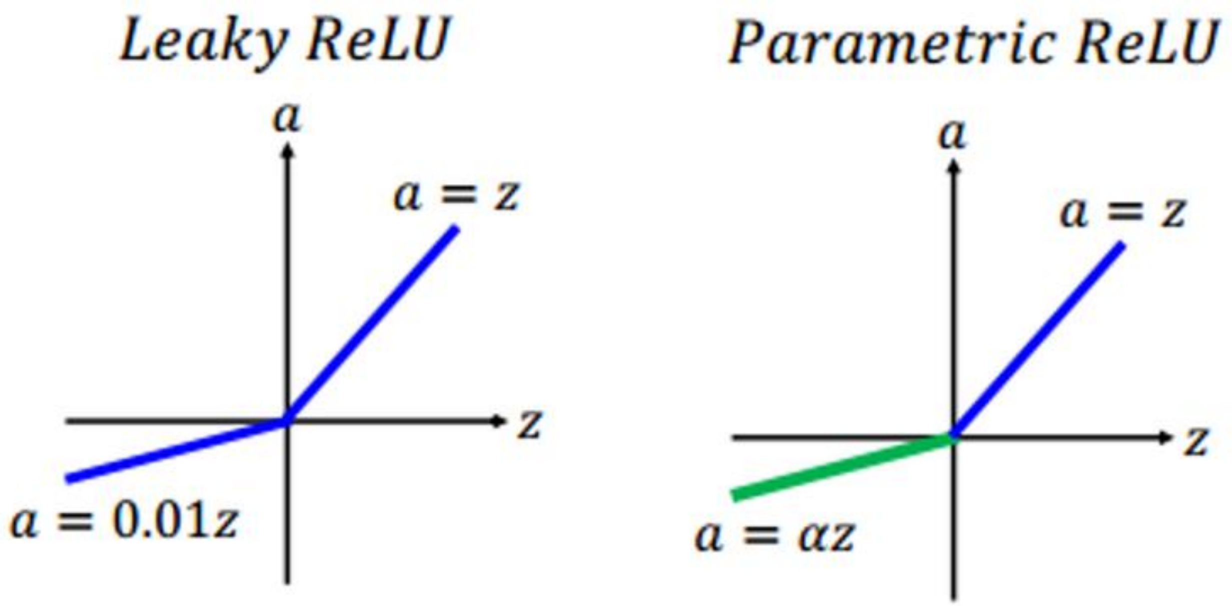
$$f(x) = \max(0, x)$$

- 計算速度相當快
- 收斂速度快
- 通常用在隱藏層



- 當某個神經元輸出為0後，就難以再度輸出 (後面都會是0)

ReLU Variant



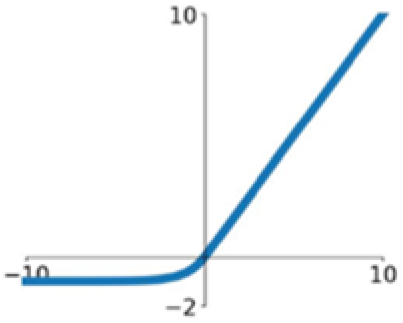
解決了X<0的區段梯度消失的問題

ELU

- 也解決 Dead ReLU 問題，輸出的均值接近於0

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Pytorch - Build Network

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

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()

        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)

    def forward(self, x):

        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)

        return x

model = Net()
output = model(input_content)
```

https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

Pytorch - Build Network

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

class Net2(nn.Module):
    def __init__(self):
        super(Net2, self).__init__()
        self.features = nn.Sequential(
                        nn.Conv2d(3, 6, 5),
                        nn.ReLU(),
                        nn.MaxPool2d(2,2),
                        nn.Conv2d(6, 16, 5),
                        nn.ReLU(),
                        nn.MaxPool2d(2,2)
        )
    def forward(self, x):
        x = self.features(x)
        return x

net2 = Net2()
print(net2)
```

nn.Sequential

一個有序的容器，NN會有序地被執行

Pytorch - Build Network

```python
class UNet(nn.Module):
    def __init__(self, layer_size=5, input_channels=3, upsampling_mode='nearest'):
        super().__init__()
        self.freeze_enc_bn = False
        self.upsampling_mode = upsampling_mode
        self.layer_size = layer_size
        self.enc_1 = PCBActiv(input_channels, 64, bn=False, sample='down-7')
        self.enc_2 = PCBActiv(64, 128, sample='down-5')
        self.enc_3 = PCBActiv(128, 256, sample='down-5')
        self.enc_4 = PCBActiv(256, 512, sample='down-3')
        for i in range(4, self.layer_size):
            name = 'enc_{:d}'.format(i + 1)
            setattr(self, name, PCBActiv(512, 512, sample='down-3'))

        for i in range(4, self.layer_size):
            name = 'dec_{:d}'.format(i + 1)
            setattr(self, name, PCBActiv(512 + 512, 512, activ='leaky'))
        self.dec_4 = PCBActiv(512 + 256, 256, activ='leaky')
        self.dec_3 = PCBActiv(256 + 128, 128, activ='leaky')
        self.dec_2 = PCBActiv(128 + 64, 64, activ='leaky')
        self.dec_1 = PCBActiv(64 + input_channels, input_channels,
                              bn=False, activ=None, conv_bias=True)
```

```python
def forward(self, input):
    h_dict = {}  # for the output of enc_N
    h_dict['h_0']= input
    h_key_prev = 'h_0'
    for i in range(1, self.layer_size + 1):
        l_key = 'enc_{:d}'.format(i)
        h_key = 'h_{:d}'.format(i)
        h_dict[h_key] = getattr(self, l_key)(
            h_dict[h_key_prev])
        h_key_prev = h_key

    h_key = 'h_{:d}'.format(self.layer_size)
    h = h_dict[h_key]

    # concat upsampled output of h_enc_N-1 and dec_N+1, then do dec_N
    # (exception)
    #                           input        dec_2          dec_1
    #                           h_enc_7      h_enc_8        dec_8

    for i in range(self.layer_size, 0, -1):
        enc_h_key = 'h_{:d}'.format(i - 1)
        dec_l_key = 'dec_{:d}'.format(i)
        h = F.interpolate(h, scale_factor=2, mode=self.upsampling_mode)
        h = torch.cat([h, h_dict[enc_h_key]], dim=1)
        h= getattr(self, dec_l_key)(h)

    return h
```

setattr & getattr 的靈活運用

讓模型更有彈性

Conv2d - Usage

Conv2d

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*) [SOURCE]

Convolution Layer

torch.nn.Conv1d : Input [N, C, W] # move kernel in 1D

torch.nn.Conv2d : Input [N, C, W, H] # move kernel in 2D

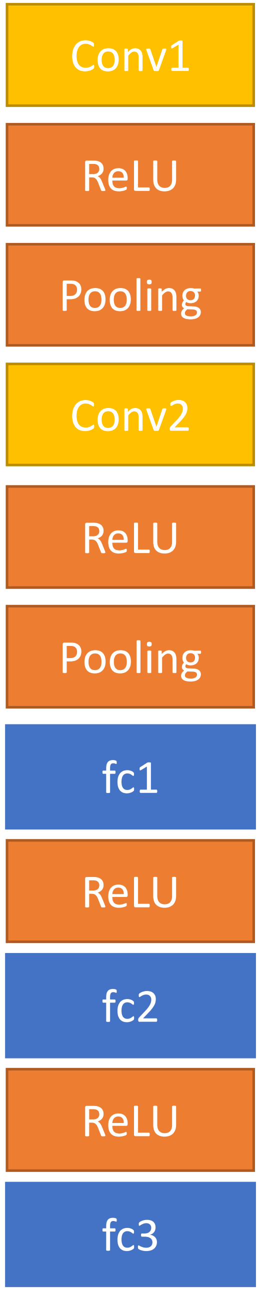
torch.nn.Conv3d : Input [N, C, D, W, H] # move kernel in 3D

Input:16 → output: 33

```
>>> # With square kernels and equal stride
>>> m = nn.Conv2d(16, 33, 3, stride=2)
>>> # non-square kernels and unequal stride and with padding
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
>>> # non-square kernels and unequal stride and with padding and dilation
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2), dilation=(3, 1))
>>> input = torch.randn(20, 16, 50, 100)
>>> output = m(input)
```

Input : 1*32*32

[C, H, W]: 1*32*32 > 6*28*28



$$w' = \begin{cases} w - 2\left\lfloor \frac{k}{2} \right\rfloor, & \text{w/o padding;} \\ w, & \text{w padding.} \end{cases}$$

Define modules param.

Build Network

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


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
print(net)
```

Input : 1*32*32

[C, H, W]: 1*32*32 > 6*28*28

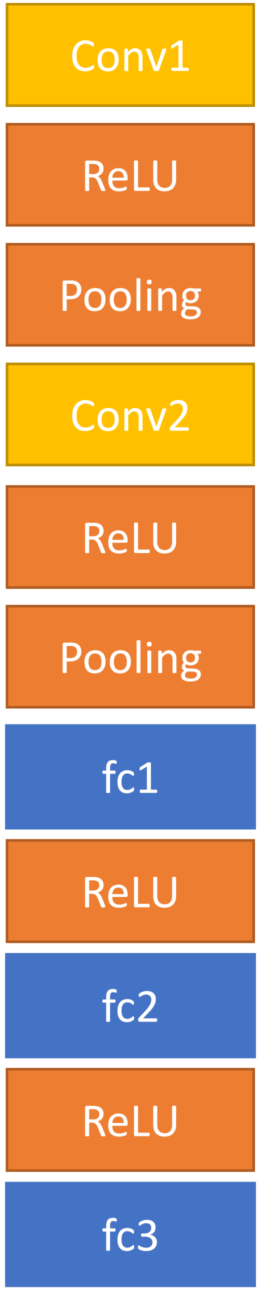
[C, H, W]: 6*28*28

$$w' = \begin{cases} w - 2\left\lfloor \frac{k}{2} \right\rfloor, & \text{w/o padding;} \\ w, & \text{w padding.} \end{cases}$$

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


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
print(net)
```

Define modules param.

Build Network

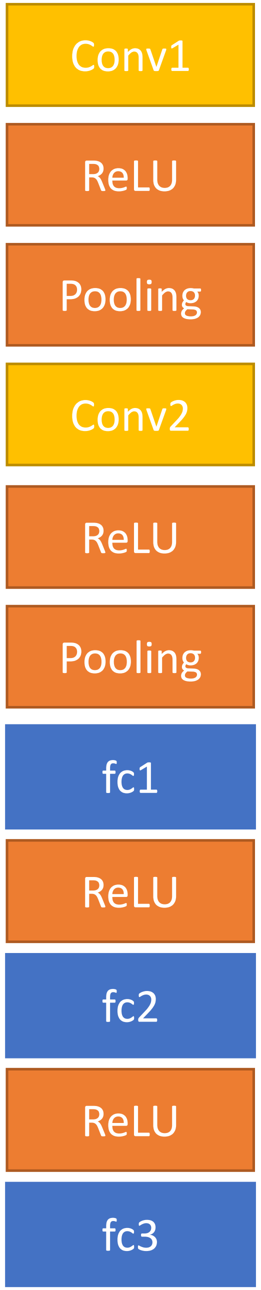


Input : 1*32*32

[C, H, W]: 1*32*32 > 6*28*28

[C, H, W]: 6*28*28

[C, H, W]: 6*28*28 > 6*14*14



$$w' = \begin{cases} w - 2\left\lfloor \frac{k}{2} \right\rfloor, & \text{w/o padding;} \\ w, & \text{w padding.} \end{cases}$$

Define modules param.

Build Network

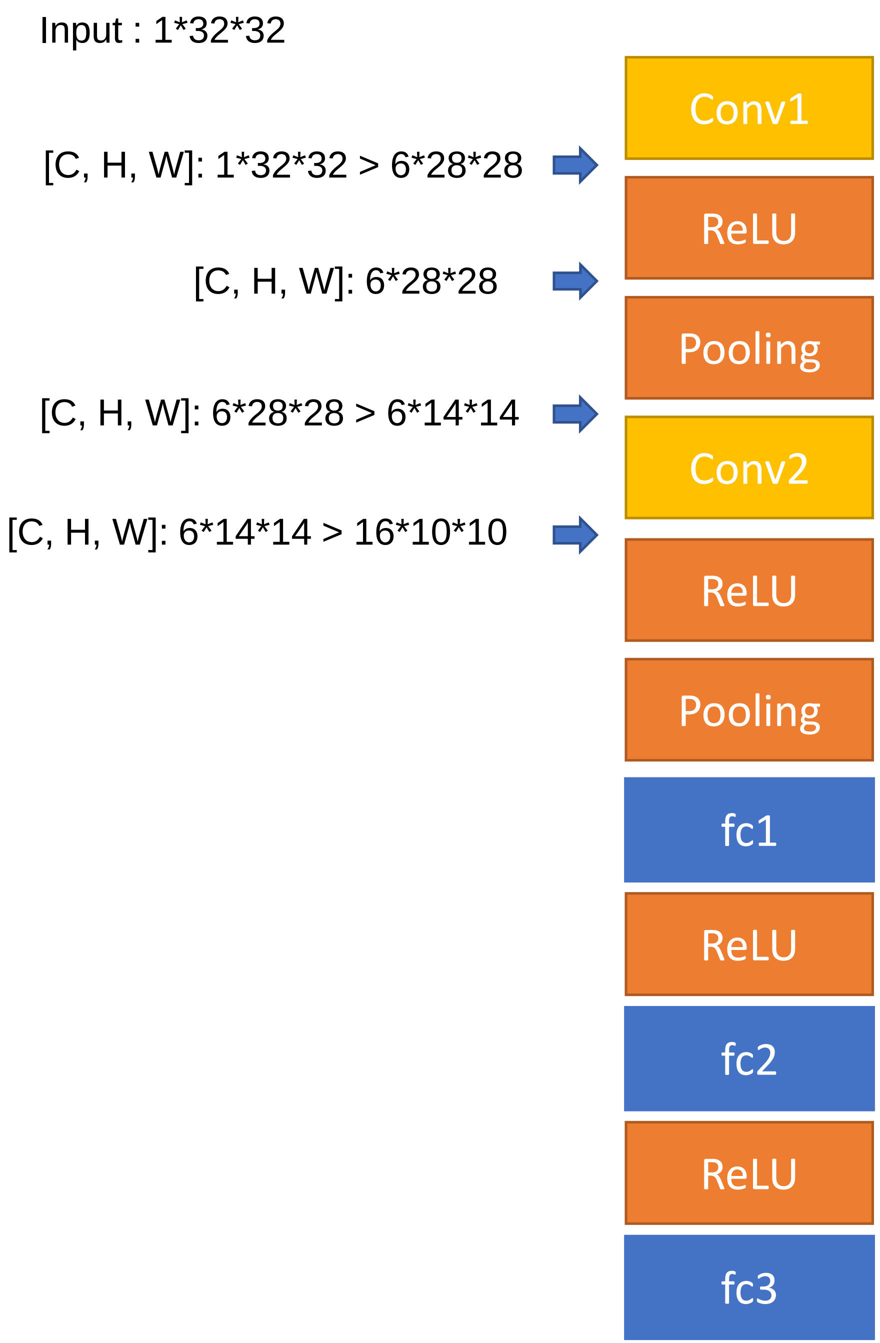
```
import torch
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
print(net)
```

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


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
print(net)
```

Define modules param.

Build Network

$$w' = \begin{cases} w - 2\left\lfloor \frac{k}{2} \right\rfloor, & \text{w/o padding;} \\ w, & \text{w padding.} \end{cases}$$

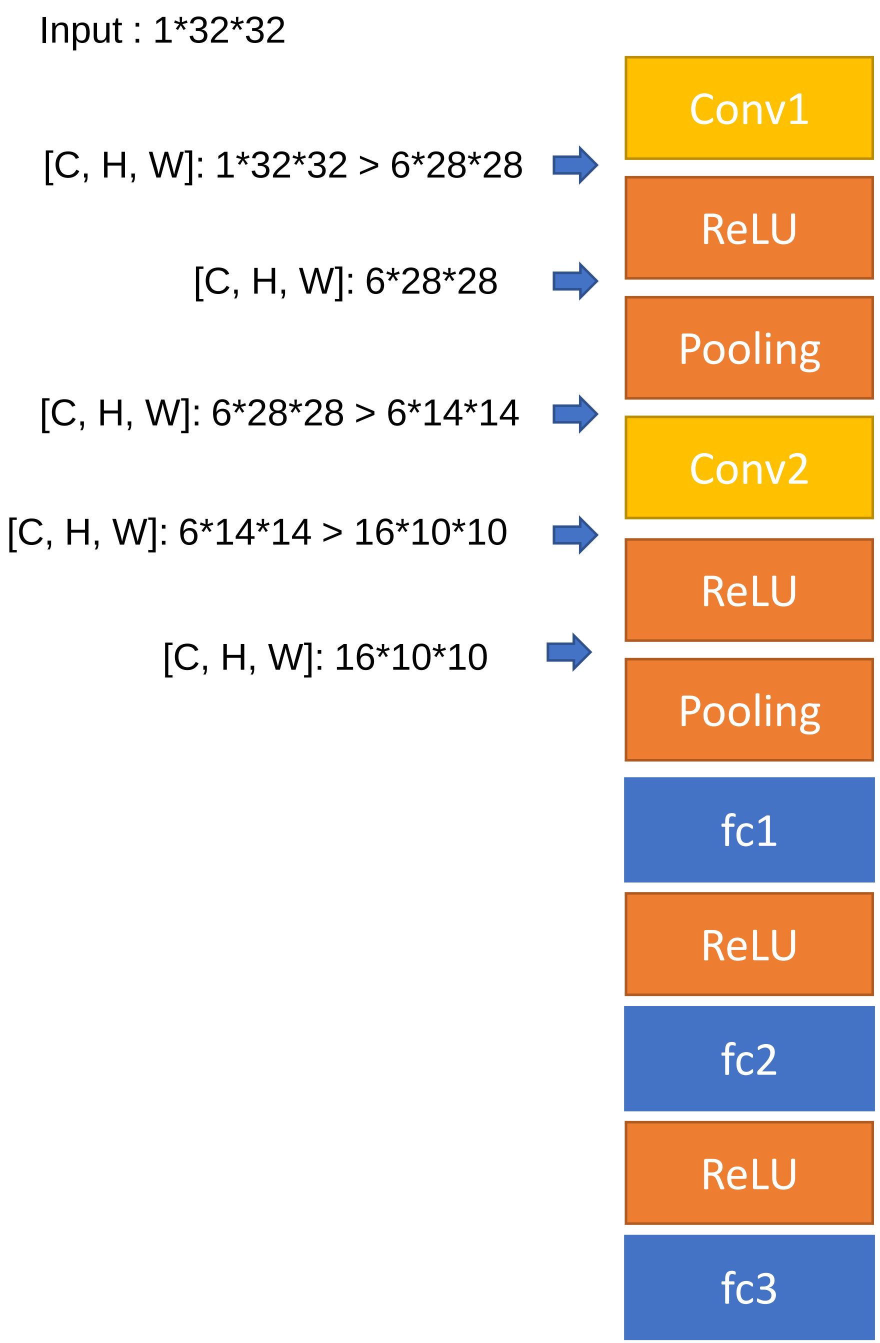


```python
import torch
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
print(net)
```

Define modules param.

Build Network

$$w' = \begin{cases} w - 2\left\lfloor \frac{k}{2} \right\rfloor, & \text{w/o padding;} \\ w, & \text{w padding.} \end{cases}$$



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


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
```

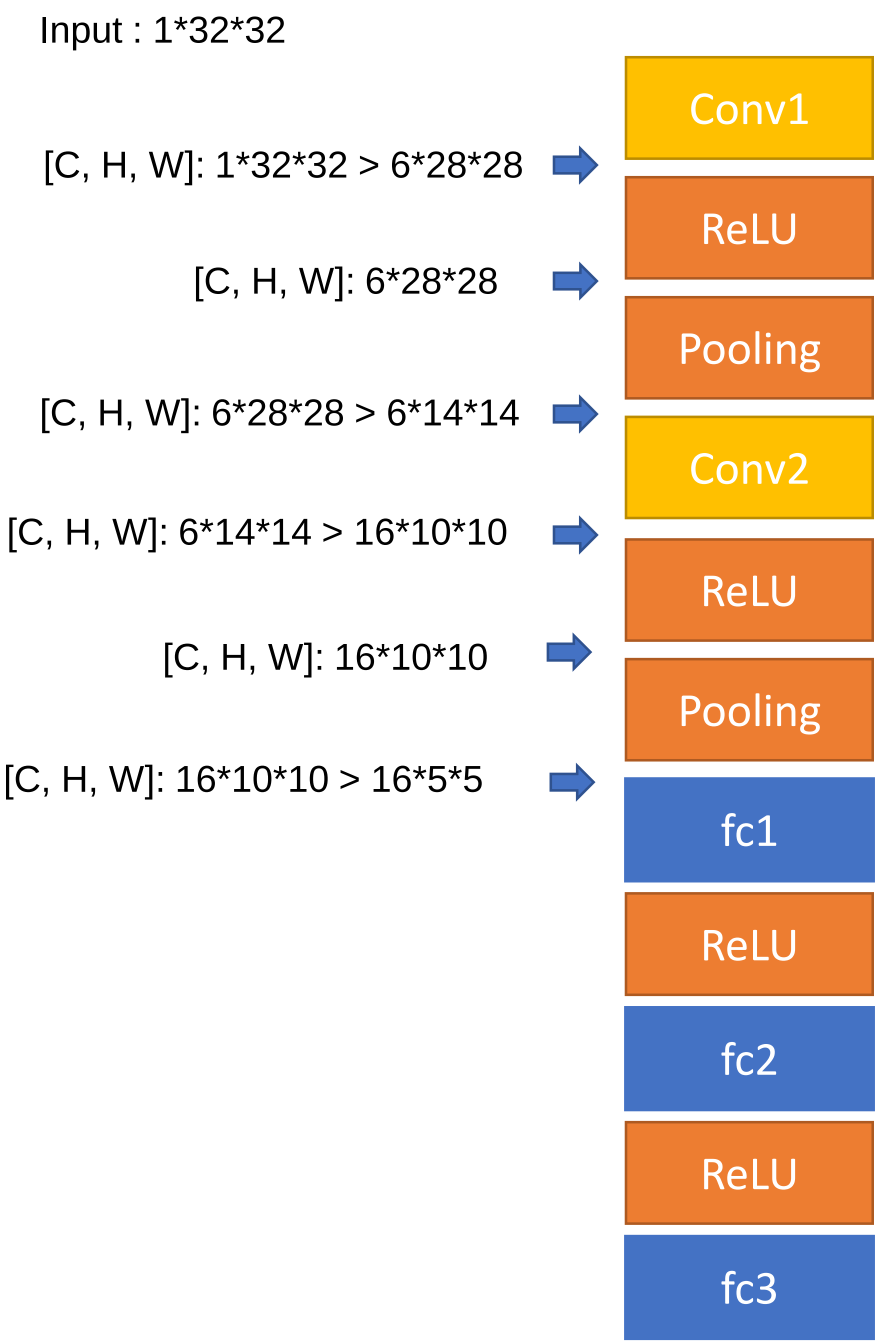
$$w' = \begin{cases} w - 2\left\lfloor \frac{k}{2} \right\rfloor, & \text{w/o padding;} \\ w, & \text{w padding.} \end{cases}$$

Define modules param.

```python
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
```

Build Network

```python
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
print(net)
```

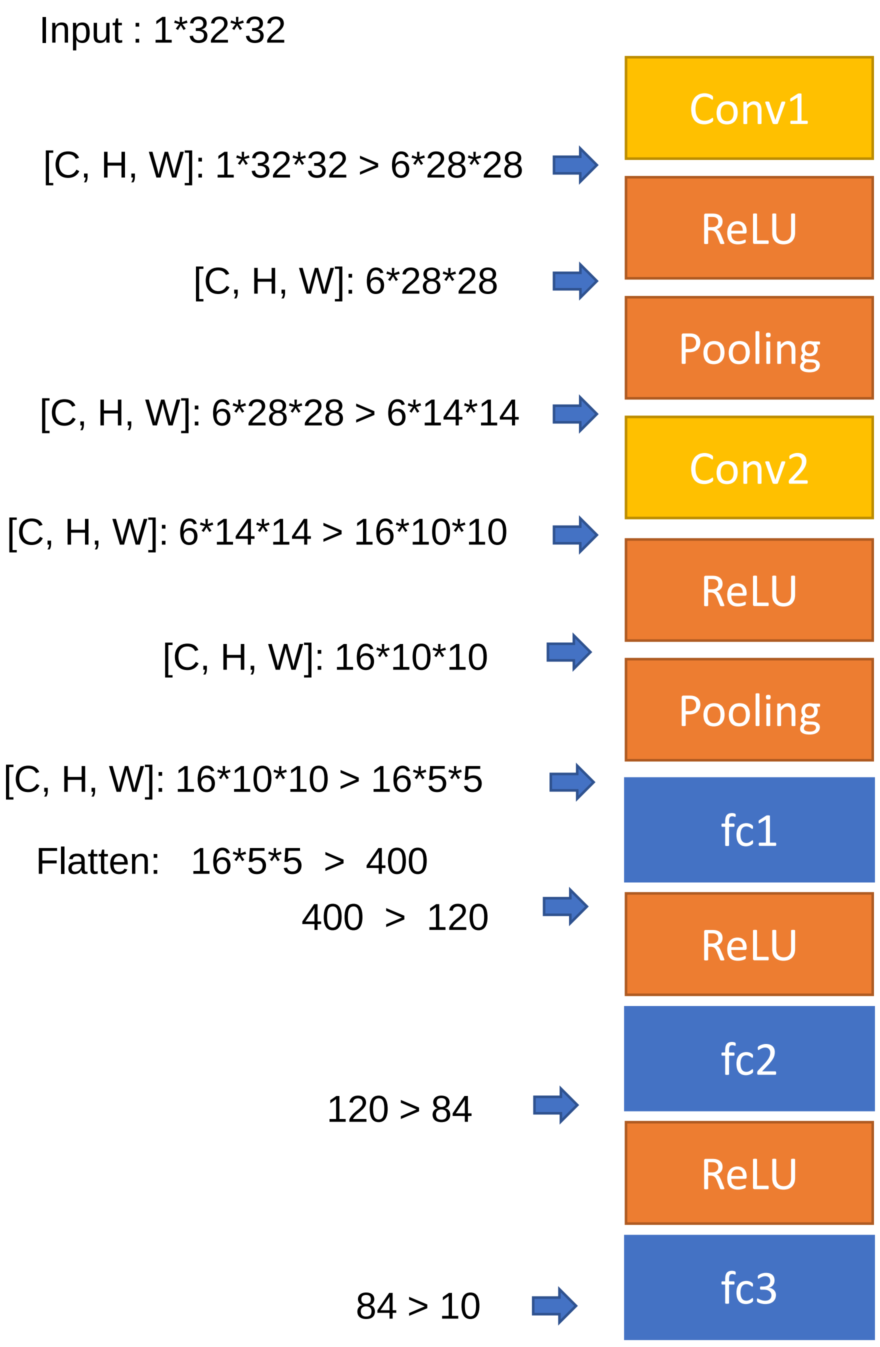


```python
import torch
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
print(net)
```

Define modules param.

Build Network

$$w' = \begin{cases} w - 2\left\lfloor \frac{k}{2} \right\rfloor, & \text{w/o padding;} \\ w, & \text{w padding.} \end{cases}$$



Load Data

- Dataset, Dataloader

https://pytorch.org/docs/stable/data.html

```python
import torch
from torch.utils.data import Dataset, DataLoader
```
Python

```python
class Dataset(Dataset):
    def __init__(self):
        self.data = torch.tensor([[1,1,1,1],[2,2,2,2],[3,3,3,3],[4,4,4,4]])
        self.label = torch.tensor([1, 2, 3, 4])

    def __getitem__(self,index):
        return self.data[index],self.label[index]

    def __len__(self):
        return len(self.data)
```
Python

```python
dataset = Dataset()
dataloader = DataLoader(dataset=dataset,
                        batch_size=2, shuffle=True)
```
Python

```python
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
for i,(data,label) in enumerate(dataloader):
    data = data.to(device)
    label = label.to(device)
    print(data,label)
```

```
cpu
tensor([[1, 1, 1, 1],
        [3, 3, 3, 3]]) tensor([1, 3])
tensor([[2, 2, 2, 2],
        [4, 4, 4, 4]]) tensor([2, 4])
```

Data Augmentation

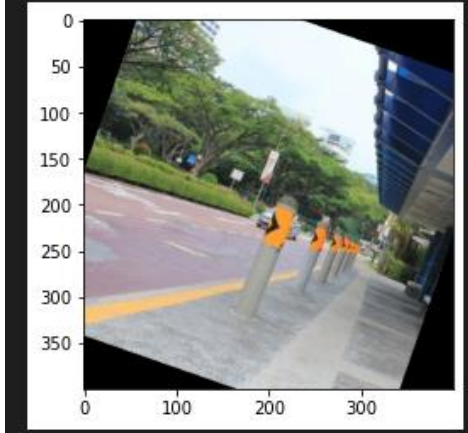
```python
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import functional as F
import torchvision.transforms as transforms
from PIL import Image as Image
import matplotlib.pyplot as plt
import os
```

```python
class Dataset(Dataset):
    def __init__(self, dir, transform=None):
        self.dir = dir
        self.image_list = os.listdir(dir)
        print(self.image_list)
        self.transform = transform
    def __len__(self):
        return len(self.image_list)

    def __getitem__(self,index):
        image = Image.open(os.path.join(self.dir,self.image_list[index]))
        if self.transform:
            image = self.transform(image)
        else:
            image = F.to_tensor(image)

        return image
```

```python
my_transform = transforms.Compose([
    transforms.RandomCrop(400),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(20),
    transforms.ToTensor()
])
dataset = Dataset(dir="./img",transform=my_transform)
dataloader = DataLoader(dataset=dataset,
                        batch_size=1, shuffle=True)
```

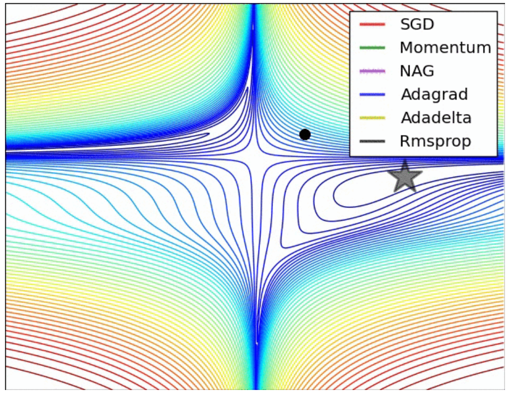
```
['(1).jpg', '(2).jpg', '(3).jpg']
```

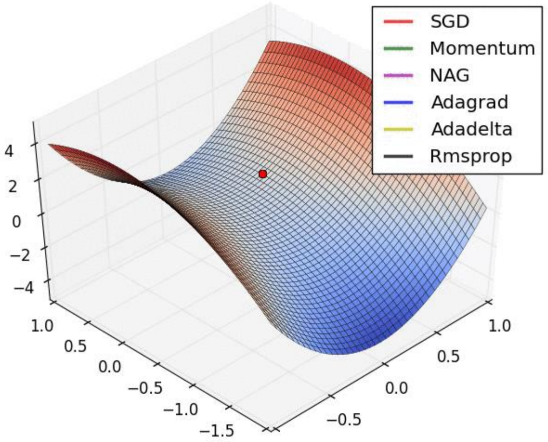
```python
for epoch in range(5):
    for i,(image) in enumerate(dataloader):
        plt.imshow(image[0].permute(1, 2, 0))
```



https://pytorch.org/vision/stable/transforms.html

Simple Training Example






```
model = Net()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
scheduler = lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)
criterion = torch.nn.MSELoss()
dataset = ImagesDataset(path_to_images)
data_loader = torch.utils.data.DataLoader(train_dataset, batch_size=10)

train = True
for epoch in range(epochs):
    if train:
        lr_scheduler.step()

    for inputs, labels in data_loader:
        inputs = Variable(to_gpu(inputs))
        labels = Variable(to_gpu(labels))
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        if train:
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

    if not train:
        save_best_model(epoch_validation_accuracy)
```

- Optimizer & Scheduler (adjust Learning Rate while training): https://pytorch.org/docs/stable/optim.html
- Loss Function: https://pytorch.org/docs/stable/nn.html#loss-functions
- How to Save Model : https://pytorch.org/tutorials/beginner/saving_loading_models.html

Homework

Build your Rain Streak classifier

What you have ?

Training Set:

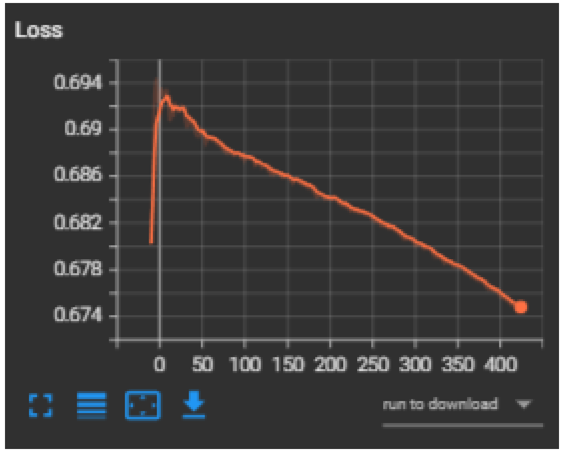
- Heavy Rain Streak: 80 pics
- Light Rain Streak: 80 pics

Test Set:

- Rain Streak: 40 pics

Requirements:

- Report the **Accuracy** of your Classifier.
- Check Your Loss with **tensorboard** (screen-shot)



補充資料:影像讀檔

- Import cv2
- from PIL import Image as Image
- import torch
- import torchvision.transforms.functional as F

```
print("===== Tensor Operate ======")

print("The Shape of cv2 image: ",cv_image.shape)
cv_image_tensor = F.to_tensor(cv_image) # H W C > C H W
print("After <to_tensor> operate, Tensor shape is:", cv_image_tensor.shape)
cv_image_tensor = cv_image_tensor.permute(1,2,0) # C H W > H W C
print("After <permute> operate, Tensor shape is",cv_image_tensor.shape)


# move our tnesor to GPU and back to cpu
cv_image_tensor = cv_image_tensor.to(device)
print("After <to device> operate, Show where the tensor is: ",cv_image_tensor.device)
cv_image_tensor = cv_image_tensor.cpu()
print("After <cpu()> operate, Show where the tensor is: ",cv_image_tensor.device)

# convert our tensor to numpy array
print("Show the type of tensor: ",type(cv_image_tensor))
cv_image = cv_image_tensor.numpy() # this will work only when our tensor on cpu
print("After <numpy()> operate, Show where the cv_image is: ",type(cv_image))
```

img&np_operate.zip