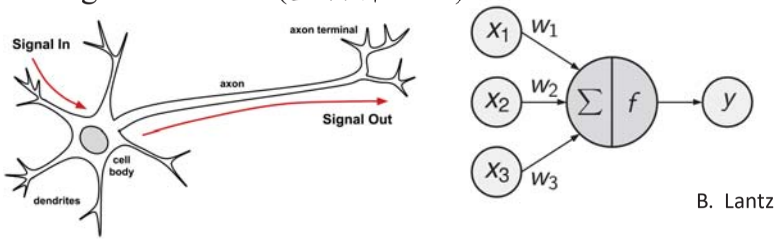
# Lecture 7 Artificial Neural Networks (人工類神經網路)

- 7.1 Artificial Neurons (類神經)
- 7.2 Training (訓練) neural networks with backpropagation (反向傳播)
- 7.3 Modern approaches (現代方法)
- 許志華



# 7.1 Artificial Neurons (類神經)

- 1943 by the neurophysiologist (神經生理學家) Warren McCulloch and the mathematician (數學家) Walter Pitts
- Biological Neurons (生物神經元)

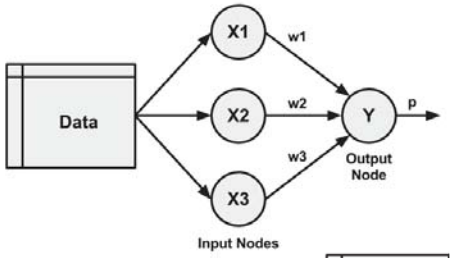


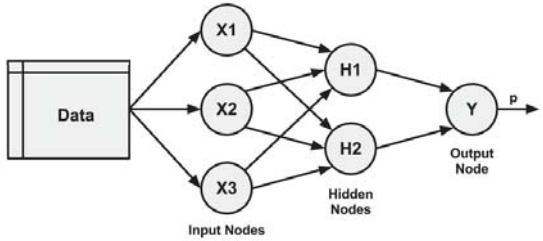

- A directed network diagram (有向網絡圖): The dendrites (樹狀突) *x* variables (變數), weights (加權), and the output signal (輸出信號) *y* variable



# 7.1.1 Network topology (網路拓撲) (1): The number of layers

- Single-layer (單層) network
- Multilayer (多層) network: Fully connected (完全連接), feedforward networks (前饋網路)

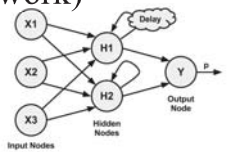


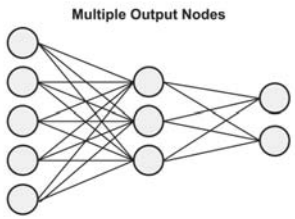



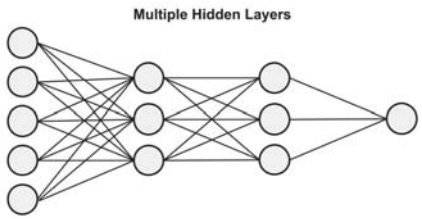



# Network topology (拓撲) (2): The direction of information travel

- Feedforward networks
  - multiple outcomes or multiple hidden layers (隱藏層)
    - hidden layers ≥ 2: deep learning (深度學習)
- recurrent (重現) network (or feedback (回饋) network)

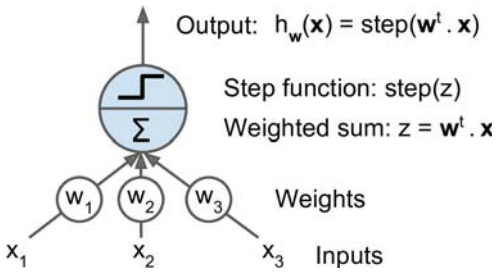


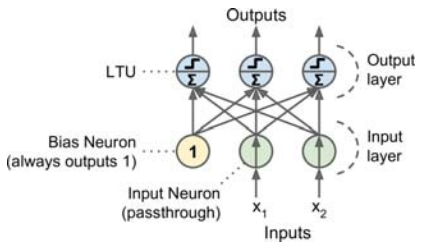



# 7.1.2 The Perceptron (感知器)

- invented in 1957 by Frank Rosenblatt
- linear threshold unit (LTU, 線性閾值單位): the inputs and output are now numbers
- used for simple linear binary classification (二元分類), 3 different binary classes

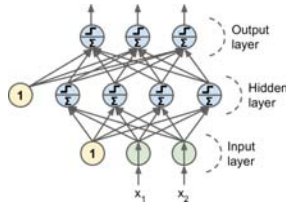


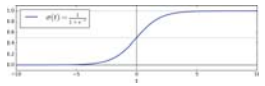



# Multi-Layer Perceptron (多層感知器)

- D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Learning representations (表徵) by back-propagating (反向傳播) errors, *Nature*, pp. 533-536, 1986.



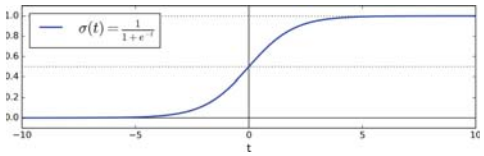

  - Logistic (邏輯斯) function $f(z) = \frac{1}{1+e^{-z}}$ instead of step function
  - range from 0 to 1, differentiable (可微分) (for training)





# Logistic (邏輯斯) function

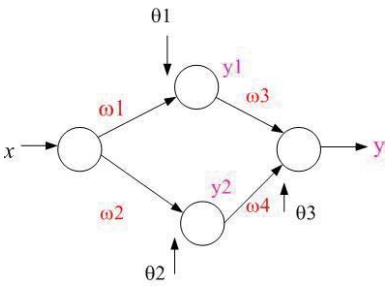
- $f(z) = \frac{1}{1+e^{-z}}$
- $f''(z) = \frac{e^z(1-e^z)}{(e^z+1)^3}$
- $f''(z) > 0$ for z < 0, $f''(z) < 0$ for z > 0
  - Neither convex (凸) nor concave (凹)
  - Bad for optimization (最佳化)



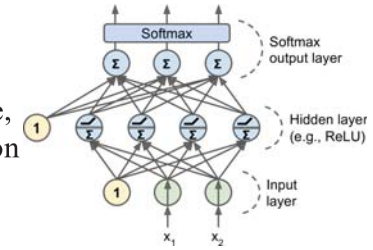



# Nonlinear decision boundary (非線性決策邊界)

- Neural network
  - $y_1 = \frac{1}{1+e^{-(w_1x+\theta_1)}} = f_1(x)$, $y_2 = \frac{1}{1+e^{-(w_2x+\theta_2)}} = f_2(x)$
  - $y = \frac{1}{1+e^{-(w_3y_1+w_4y_2+\theta_3)}} = \frac{e^{(w_3y_1+w_4y_2+\theta_3)}}{e^{(w_3y_1+w_4y_2+\theta_3)}+1} = f(y_1, y_2)$
  - Function of function: composite function (複合函數)
- Linear regreesion 酒的價格
  ≈ –0.4504 + 0.6014 成長期平均溫度 – 0.003958 成時雨量 + 0.001043 冬季雨量
- 邏輯斯迴歸:
  Linear decision function = $\theta x + \theta_0$



# 7.1.3 Applications (應用)




- For regression (迴歸):
  - Output layer (輸出層): single node, mean squared error for loss function
- For classification (分類):
  - Output layer (輸出層)
    - 2: each output corresponding to (對應於) a different binary class, e.g., spam/ham, urgent/not-urgent
    - Many: a shared *softmax* function (lecture 6-42)
      - classes 0 through 9 for digit image classification



# Applications in business (商業中的應用)

- Michal Tkac and Robert Verner, Artificial neural networks in business: Two decades of research (二十年的研究), *Applied Soft Computing*, Volume 38, January 2016, Pages 788-804
  - 412 suitable articles (合適的文章)
  - Auditing and accounting (審計和會計, 14), Costs monitoring (成本監測, 7), Credit scoring (信用評分, 36), Customers metrics (客戶指標, 24), Decision support (決策支援, 37), Derivatives (衍生性金融商品, 28), Exchange & interest rates (匯率和利率, 27), Financial analysis (財務分析, 35), Financial distress & bankruptcy (財務困境和破產, 75), Fraud analysis (詐欺分析, 12), Inflation (通貨膨脹, 6), Marketing (行銷, 18), Reviews of applications (9), Sales (銷售, 11), Shares & bonds (股票和債券, 73)



# 7.1.4 Universal function approximator (萬能函數近似器)

- (Hornik et al. 1989, Cybenko, 1989) A neural network with at least one hidden layer (隱藏層) of sufficiently (足夠地) many neurons is a universal function approximator.
  - Approximate (近似) any continuous (連續的) function to an arbitrary (任意的) precision (精確) over a finite interval.
  - Taylor series (泰勒級數) by polynomials (多項式): Interval of convergence (收斂區間) $(-\infty, \infty)$

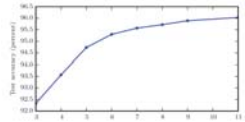
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

http://hobbieroth.blogspot.tw/2015/05/taylors-series.html



# Why deep if 1 layer is an universal function approximator ?

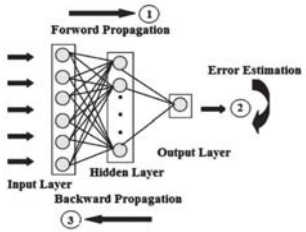
- I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, MIT Press, 2016, sec 6.4. (available online)
  - In many circumstances (情況), using deeper models can reduce the number of units required to represent the desired function and can reduce the amount of generalization (推論) error.
  - Figure 6.6: deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses.



- **李宏毅,** Machine Learning and having it deep and structured (2018,Spring) , Theory 1 - Why Deep Structure?

# 7.2 Training (訓練) neural networks with backpropagation (反向傳播)

- The backpropagation algorithm (演算法) iterates (重複) through many cycles (循環) of two processes:
  - A forward (向前的) phase
  - A backward (向後的) phase
- Animation at https://cdn-images-1.medium.com/max/1000/1*0hf4gLbc-2V5RMXBhluJ_A.gif
- Stopping criterion (停止條件)

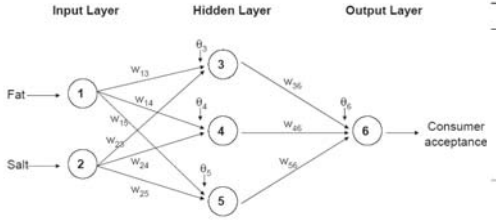



https://www.researchgate.net/profile/Morteza_Esfandyari/publication/241741756/figure/fig2/AS:298577172680729@1448197753779/Figure-2-Back-propagation-multilayer-ANN-with-one-hidden-layer.png



# Example

- G. Shmueli, N. R. Patel and P. C. Bruce, *Data Mining for Business Intelligence* (商業智慧), 2010.
- Using fat (脂肪) & salt (鹽) content to predict consumer acceptance (消費者接受) of cheese (乳酪)
- Observation (觀察)

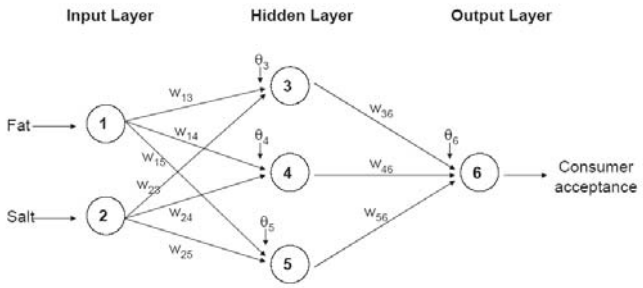



| *Obs.* | *Fat Score* | *Salt Score* | *Acceptance* |
|---|---|---|---|
| 1 | 0.2 | 0.9 | 1 |
| 2 | 0.1 | 0.1 | 0 |
| 3 | 0.2 | 0.4 | 0 |
| 4 | 0.2 | 0.5 | 0 |
| 5 | 0.4 | 0.5 | 1 |
| 6 | 0.3 | 0.8 | 1 |



# 7.2.1 Forward phase (向前的階段) (1)

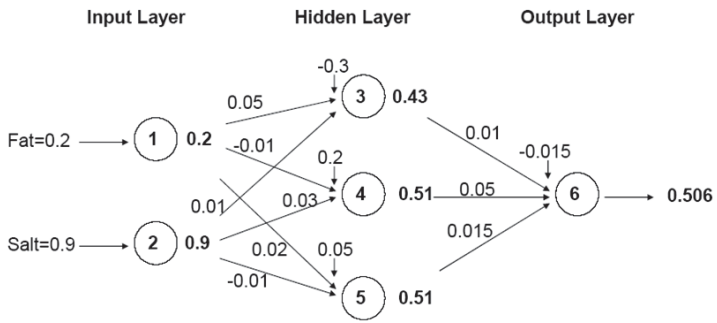
- 從輸入到輸出
- 隨機初始化： $\theta_3 = -0.3, w_{1,3} = 0.05, w_{2,3} = 0.01$
- Observation (觀察) 1, fat 0.2, salt 0.9 $\Longrightarrow$ $\text{Output}_3 = \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-[-0.3+(0.05)(0.2)+(0.01)(0.9)]}} = 0.43$






# Forward (向前的) phase (2)

- $\text{Output}_6 = \frac{1}{1+e^{-[-0.015+(0.01)(0.43)+(0.05)(0.51)+(0.015)(0.51)]}} = 0.506 > 0.5$, classified (分類) as class (類別) 1



# 7.2.2 Backward phase (向後的階段)

$$\min \frac{1}{2m}\sum_{s=1}^{m}(y_s - \hat{y}_s)^2$$

- The network's output signal (輸出信號) $\hat{y}_s$ from the forward phase (向前的階段)
- True target value (真正的目標值) $y_s$ in the training data
- Multiple outputs $m$, now $m = 1$

- An error is propagated (傳遞) backwards in the network to modify (修正) the connection (連接) weights between neurons and reduce future errors.
  - 從輸出往回修正
  - How to minimize a non-convex function? Gradient algorithm (梯度演算法)



# backpropagation (反向傳播) (1)

- Output from output node 6:

$$\hat{y}_6 = \frac{1}{1+e^{-[\theta_6 + w_{36}y_3 + w_{46}y_4 + w_{56}y_5]}} \equiv \frac{1}{1+e^{-x}} = (1+e^{-x})^{-1}$$

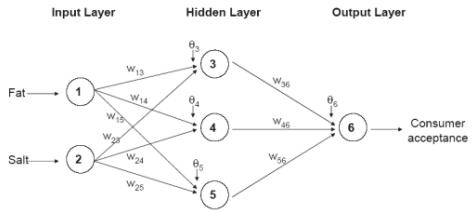
- Objective function (目標函數) $f(\theta, w) = \frac{1}{2}(y - \hat{y}_6)^2$
- chain rule (連鎖律) $\frac{\partial f}{\partial \theta_6} = \frac{2}{2}(y - \hat{y}_6)\frac{\partial (y - \hat{y}_6)}{\partial \theta_6} = (y - \hat{y}_6)(-1)(-1)$

$(1+e^{-x})^{-2}\frac{\partial e^{-x}}{\partial \theta_6} = (y - \hat{y}_6)(1+e^{-x})^{-2}e^{-x}(-1)$

$$= -(y - \hat{y}_6)\frac{1}{1+e^{-x}}\frac{e^{-x}}{1+e^{-x}}$$

$$= -(y - \hat{y}_6)\hat{y}_6(1 - \hat{y}_6) \equiv -g_6,$$

$$\frac{\partial f}{\partial w_{36}} = -g_6\hat{y}_3$$

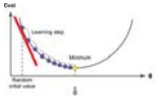





# backpropagation (反向傳播) (2)

- Negative gradient (負梯度) (for minimization)

$$g_k = \hat{y}_k(1 - \hat{y}_k)(y_k - \hat{y}_k)$$



  - Output node $\hat{y}_k = 0.506$ from slide 16:
    - $g_k = 0.506(1 - 0.506)(1 - 0.506) = 0.123$. (slide 14)
    - output 1 (on slide 14 for acceptance), but predict 0.506 (不足), so move upward (↑)
  - If real output 0, predict 0.506 (超過)
    - $g_k = (0.506)(1 - 0.506)(0 - 0.506) = -0.126$
    - move downward (↓)
- The weights are updated $\theta_j^{new} = \theta_j^{old} + ag_j$



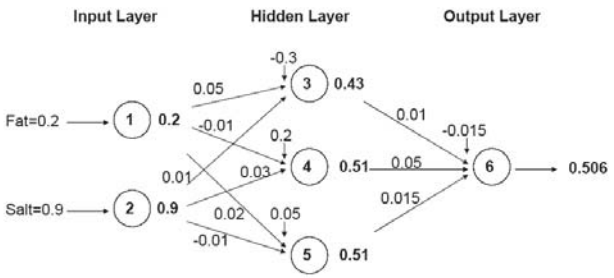
# backpropagation (反向傳播) (3)

- Case updating (案例更新): Pick $a = 0.5$ now

$$\theta_j^{new} = \theta_j^{old} + ag_j, w_{i,j}^{new} = w_{i,j}^{old} + ag_j\hat{y}_3$$

$$\theta_6 = -0.015 + (0.5)(0.123) = 0.047\ (\uparrow)$$

$$w_{36} = 0.01 + (0.5)(0.123)(0.43) = 0.036\ (\uparrow)$$



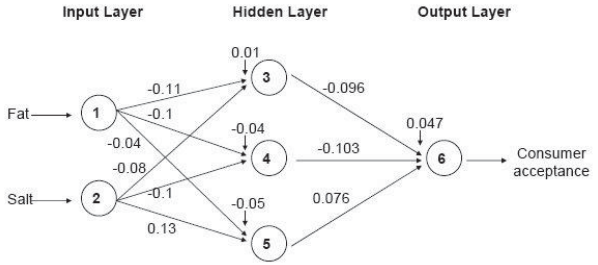
# 7.2.3 Stopping criterion (停止條件)

- Common criteria to stop the updating (更新)
  - When weights (權重) $w(t)$ change very little from one iteration (迭代) to the next.
    $$\|w(t+1) - w(t)\| < \varepsilon$$
  - When the misclassification (分錯) rate reaches a required threshold (閥值)
  - When a limit on runs (執行次數) is reached
- 儲存所有的循環：記憶體不夠。
  - 解方：使用兩組資料儲存。遞迴前，取代之



# 7.2.4 Training Results

- Fat/Salt Example: Final Weights




- Prediction: Forward (向前的) phase

| Obs. | Fat Score | Salt Score | Acceptance |
|---|---|---|---|
| 1 | 0.2 | 0.9 | 1 |
| 2 | 0.1 | 0.1 | 0 |
| 3 | 0.2 | 0.4 | 0 |
| 4 | 0.2 | 0.5 | 0 |
| 5 | 0.4 | 0.5 | 1 |
| 6 | 0.3 | 0.8 | 1 |



# Final Classifications

- Accuracy 50%: Did not do very well!
- Why? 6 observations (觀察) is not enough to train model

| Row Id. | Predicted Class | Actual Class | Prob. for 1 (success) | fat | salt |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0.498658971 | 0.2 | 0.9 |
| 2 | 0 | 0 | 0.497477278 | 0.1 | 0.1 |
| 3 | 0 | 0 | 0.497954285 | 0.2 | 0.4 |
| 4 | 0 | 0 | 0.498202273 | 0.4 | 0.5 |
| 5 | 0 | 1 | 0.49800783 | 0.3 | 0.4 |
| 6 | 0 | 1 | 0.498571499 | 0.3 | 0.8 |



# 7.2.5 Data Center Optimization (資料中心最佳化) at Google

- Jim Gao, Google, 2014.
- PUE值 (Power Usage Effectiveness, 電源使用效率)
  - https://www.digitimes.com.tw/tw/dt/n/shwnws.asp?id=0000449893_so178fcdlg9o48291a9wa&ct=1
  - 計算資料中心節能省電的標準，「資料中心的總用電量」除以「資訊中心內IT設備的總用電量」
  - PUE值越低，代表機房在環境監控、空調冷卻、燈光等週邊設施的用電量更少
  - Historical PUE values at Google

# difficult to understand and optimize energy efficiency (能源效率)

- possible operating configurations (可能的操作配置)
- nonlinear interdependencies (非線性相互依賴)
  - a simple change to the cold aisle temperature setpoint (走道溫度設定點) will produce load variations (負載變化) in the cooling infrastructure (冷卻基礎設施) (chillers, cooling towers, heat exchangers, pumps, etc.),
  - which in turn cause nonlinear changes in equipment efficiency.
  - Ambient (周圍) weather conditions and equipment controls will also impact the resulting DC efficiency.
  - Using standard formulas (標準公式) for predictive modeling often produces large errors because they fail to capture such complex interdependencies



# The neural network features

1. Total server IT load [kW]
2. Total Campus Core Network Room (CCNR) IT load [kW]
3. Total number of process water pumps (PWP) running
4. Mean PWP variable frequency drive (VFD) speed [%]
5. Total number of condenser water pumps (CWP) running
6. Mean CWP variable frequency drive (VFD) speed [%]
7. Total number of cooling towers (冷卻塔) running
8. Mean cooling tower leaving water temperature (LWT) setpoint [F]
9. Total number of chillers running
10. Total number of drycoolers running
11. Total number of chilled water injection pumps running
12. Mean chilled water injection pump setpoint temperature [F]
13. Mean heat exchanger approach temperature [F]
14. Outside air wet bulb (WB) temperature [F]
15. Outside air dry bulb (DB) temperature [F]
16. Outside air enthalpy [kJ/kg]
17. Outside air relative humidity (RH) [%]
18. Outdoor wind speed (室外風速) [mph]
19. Outdoor wind direction [deg]



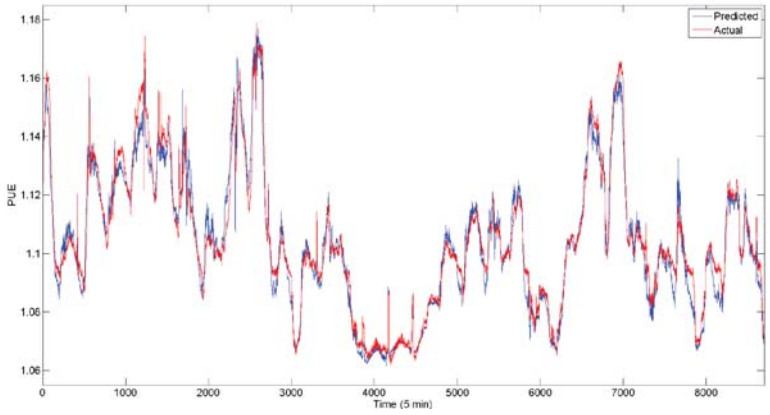
# neural network

- 5 hidden layers (隱藏層), 50 nodes per hidden layer
- 0.001 as the regularization parameter (正規化參數)
- 19 normalized input variables (標準化輸入變數)
- one normalized output variable (the DC PUE)
- 184,435 time samples at 5 minute resolution (解析度) (approximately 2 years of operational data).
  - 70% of the dataset is used for training with the remaining 30% used for cross validation and testing.
  - The chronological order (按時間順序排列) of the dataset is randomly shuffled (隨機洗牌) before splitting to avoid biasing (偏誤) the training and testing sets on newer or older data.



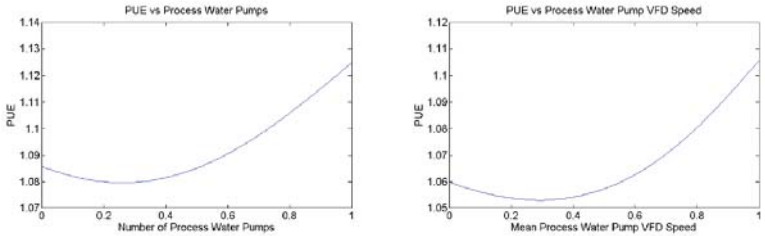
# Predictive Accuracy (預測準確性)

- mean absolute error (平均絕對誤差) of 0.004 and standard deviation (標準差) of 0.005 on the test dataset
- one of Google's DCs over one month during the summer



# Sensitivity Analysis (靈敏度分析)

- isolate for the effects of specific variables (特定變數) by linearly varying one input at a time while holding all others constant (保持所有其他不變)
- All test results have been verified empirically (憑經驗驗證)
- process water pumps (水泵ㄅㄥˋ): 標準化參數




https://intra.ece.ucr.edu/~sren/project/water/index.html



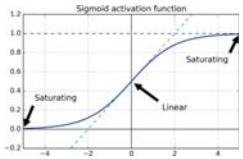
# Application Examples

- Example 3: DC Plant Configuration Optimization (工廠配置最佳化)
  - In this study, a planned upgrade (升級) to the electrical infrastructure (電氣基礎設施) required rerouting (改道) ~40% of the server traffic for several days for worker safety.
  - This required corresponding changes in the operational lineup to meet the reduced IT load while maintaining DC efficiency.
  - Through a combination of PUE simulations and local expertise (模擬和當地專業知識), the team selected a new set of operational parameters (操作參數) that reduced the PUE by ~0.02 compared to the previous configuration.



# 7.3 Modern approaches (現代方法)

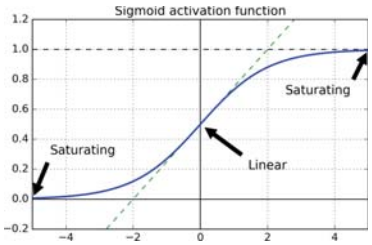
- Vanishing/Exploding Gradients Problems (梯度消失/爆炸問題)
  - Sepp Hochreiter: 1991, diploma thesis (學位論文)
  - gradients often get smaller and smaller as the algorithm progresses down to the lower layers (較低層).
  - the Gradient Descent (梯度下降法) update leaves the lower layer connection weights (權重) virtually unchanged (未改變), and training never converges (收斂) to a good solution.
  - $\theta_j^{new} = \theta_j^{old} + a g_j \Rightarrow \theta_j^{new} \approx \theta_j^{old}$






# Exploding gradients problem (梯度爆炸問題)

- *exploding gradients* problem: The gradients can grow bigger and bigger and the algorithm diverges.
  - recurrent neural networks (遞歸神經網路)
- Y. Bengio, P. Simard, and P. Frasconi, Learning long-term dependencies (長期依賴) with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 1994, 5(2), 157–166.



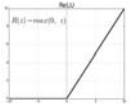
# Renewal (復活) around 1990s

- a huge quantity of data available to train neural networks, and outperform other ML techniques
- tremendous (極大的) increase in computing power: Moore's Law, GPU
- training algorithms have been improved
- local optima rare or close to the global optimum (全域最佳解)
  - Yann LeCun, Yoshua Bengio & Geoffrey Hinton, Deep learning, Nature 521, 436–444, 2015. (In practice, poor local minima are rarely a problem with large networks.)
- funding and progress: talent



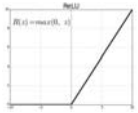
# 7.3.1 Initialization (初始化)

- Xavier Glorot and Yoshua Bengio, *PMLR*, 2010.
  - Logistic activation function (激勵函數) and random initialization: Variance of the outputs of each layer >> variance of its inputs.
  - *n*inputs and *n*outputs: Number of input and output connections (連結) for the layer whose weights are being initialized
  - Normal distribution with mean 0 and standard deviation (標準差) $\sigma = \sqrt{\frac{2}{n_{inputs}+n_{outputs}}}$ ($< 1$)
- He, et al., for the ReLU, *ICCV*, 2015 (next)
  - Normal distribution with mean 0 and standard deviation $\sigma = \sqrt{2}\sqrt{\frac{2}{n_{inputs}+n_{outputs}}}$

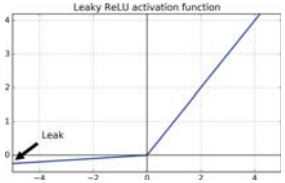




# 7.3.2 Nonsaturating (非飽和) Activation Functions



- ReLU activation function (激勵函數)
  - Does not saturate (飽和) for positive values and quite fast to compute
  - *dying* (死亡) *ReLUs:* During training, some neurons effectively die. Output = 0 if $x < 0$
- *leaky* (漏) *ReLU*: LeakyReLU$\alpha(z) = \max(\alpha z, z)$
  - $\alpha = 0.01 \sim 0.2$
  - If z > 0, LeakyReLU$\alpha(z)$ = z
  - If z < 0, LeakyReLU$\alpha(z) = \alpha z$

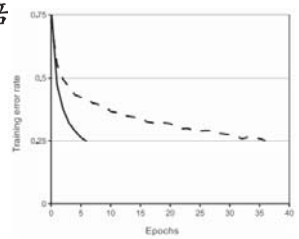



https://ml4a.github.io/images/figures/relu.png



# Performance of ReLUs

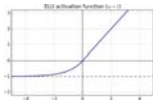
- Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks (深度卷積神經網絡), *Advances in Neural Information Processing Systems* 25, 2012.
  - AlexNet
- A four-layer convolutional neural network with ReLUs (solid line) reaches a 25% training error rate (訓練錯誤率) on CIFAR-10 six times faster (快六倍 network with tanh neurons (dashed line).
  - $g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



# Exponential (指數) linear unit (ELU)

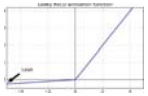
- D. Clevert, T. Unterthiner, S. Hochreiter, Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), *ICLR*, 2016

$$ELU(z) = \begin{cases} \alpha[\exp(z) - 1] & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$



  - Outperformed (勝過) all the ReLU variants in their experiments

$$ELU'(z) = \begin{cases} \alpha \exp(z) & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$



  - slower to compute for exponential function



# scaled (縮放) exponential linear units (SELUs)

- G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, Self-Normalizing Neural Networks, NIPS, 2017 (102-page paper)
  - Proof by Banach fixed point theorem: $f(x) = x$
  - 讓每一層的訊號平均值為 0、標準差為 1，避免 梯度消失/爆炸問題，可以訓練深度的神經網路。
- $ELU(z) = \begin{cases} \alpha[\exp(z) - 1] & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$
- $SELU(z) = \begin{cases} \lambda\alpha[\exp(z) - 1] & \text{if } z < 0 \\ \lambda z & \text{if } z \geq 0 \end{cases}$
  - Scale $\lambda$ =1.0507009873554804934193349852946 > 1
  - $\alpha$= 1.6732632423543772848170429916717



# 7.3.3 Batch Normalization (批次標準化)

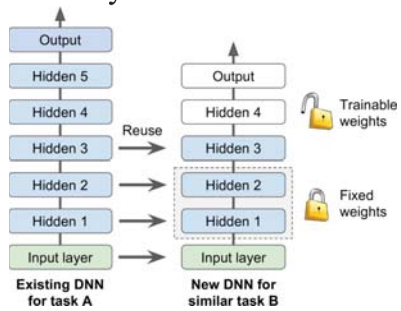
- He initialization (初始化) reduces the vanishing/exploding gradients problems at the beginning of training, it doesn't guarantee that they won't come back during training.
- S. Ioffe and C. Szegedy, Batch Normalization, *ICML*, 2015.
- Steps (1)(2): Evaluate the mean $\mu_B$ and standard deviation (標準差) $\sigma_B$ of the inputs over the current mini-batch (小批量)
- Step (3) zero-center and normalize the inputs $\hat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B{}^2 + \epsilon}}$:
  - Smoothing term (平滑項) $\epsilon$, typically $10^{-3}$
- Step (4) output of the BN operation: a scaled and shifted version of the inputs. $z^{(i)} = \gamma\hat{x}^{(i)} + \beta$
- Cost:
  - computational complexity (計算複雜度)
  - 2 hyperparameters (超參數) to tune: $\gamma, \beta$



# 7.3.4 Reusing Pretrained Layers (重用預訓練的圖層)

- *transfer learning* (遷移學習): Try to find an existing neural network that accomplishes a similar task (完成類似的任務) to the one you are trying to tackle, then just reuse (再使用) the lower layers of this network
  - speed up (加快) training considerably
  - also require much less training data
- Model Zoos:

github.com/tensorflow/models



# 7.3.5 Faster Optimizers (更快的最佳化器)

- using a faster optimizer than the regular Gradient Descent
  - Tijmen Tieleman and Geoffrey Hinton, RMSProp, Cousera, 2012
  - D. Kingma and J. Ba, Adam: A Method for Stochastic Optimization (隨機最佳化), *ICLR*, 2015.
    - Adam: adaptive moment estimation (適應性矩估計)
  - Sashank J. Reddi, Satyen Kale, Sanjiv Kumar, On the Convergence (收斂) of Adam and Beyond, *ICLR*, 2018. (Best Paper Award)
- Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow



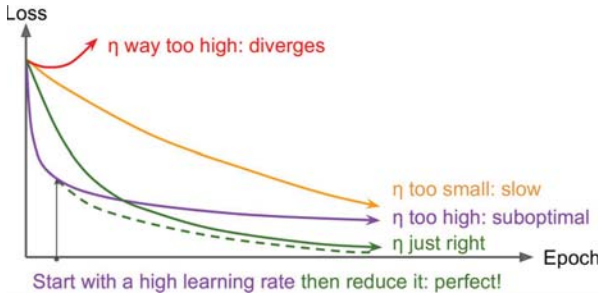
# (1) Momentum (動量) optimization

- B. Polyak, Some methods of speeding up the convergence of iteration methods (疊代法), 1964.
- Gradient Descent: $\theta \leftarrow \theta - \eta \nabla_\theta J(\theta)$
- Momentum algorithm
  1) $m \leftarrow \beta m + \eta \nabla_\theta J(\theta)$
  2) $\theta \leftarrow \theta - m$
- If the gradient remains constant, the terminal velocity (終極速度) (i.e., the maximum size of the weight updates) $\frac{\eta \nabla_\theta J(\theta)}{1-\beta}$
  - If $\beta = 0.9$, 10 times faster than Gradient Descent



# Learning Rate Scheduling

- For Momentum (動量) optimization
- Find a fairly good learning rate by training your network several times during just a few epochs using various learning rates and comparing the learning curves.
- The ideal learning rate will learn quickly and converge to good solution



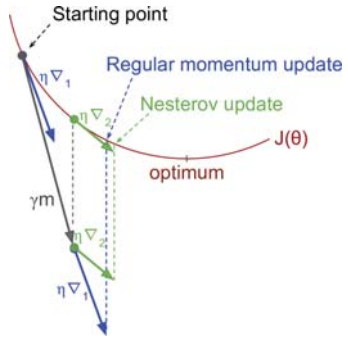



# Learning schedules

- Predetermined piecewise constant learning rate: set the learning rate to $\eta 0 = 0.1$ at first, then to $\eta 1 = 0.001$ after 50 epochs.
- Performance scheduling: Measure the validation error every $N$ steps and reduce the learning rate by a factor of $\lambda$ when the error stops dropping.
- Exponential scheduling: $\eta(t) = \eta 0\ 10^{(-t/r)}$
- Power scheduling: $\eta(t) = \eta 0\ (1 + t/r)^{(-c)}$. The hyperparameter $c$ is typically set to 1.

# (2) Nesterov Accelerated Gradient

- Yurii Nesterov, A Method for Unconstrained Convex Minimization Problem with the Rate of Convergence O(1/k2), 1983.
- Nesterov Accelerated gradient algorithm
  1) $m \leftarrow \beta m + \eta \nabla_\theta J(\theta + \beta m)$
  2) $\theta \leftarrow \theta - m$
- Nesterov update slightly closer to the optimum.
- significantly faster than regular Momentum optimization






# (3) AdaGrad (1)

- John Duchi, Elad Hazan, Yoram Singer, Adaptive Subgradient (適應次梯度) Methods for Online Learning and Stochastic Optimization, *JMLR*, 2011
- (1) $s \leftarrow s + \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
  - $\otimes$ : element-wise multiplication. $[1\ 2] \otimes [1\ 2] = [1\ 4]$
  - Equivalent: $s_i \leftarrow s_i + \left(\frac{\partial J(\theta)}{\partial \theta_i}\right)^2$ for each element $s_i$ of the vector $s$
  - If the cost function is steep (陡峭的) along the ith dimension, then $s_i$ will get larger and larger at each iteration.



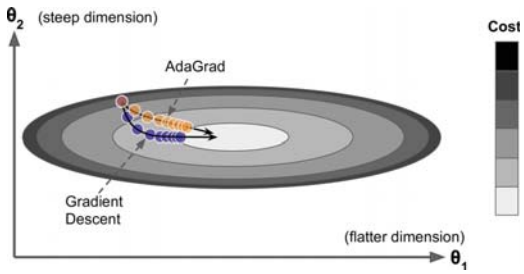
# AdaGrad (2)

- (2) $\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{s + \epsilon}$
  - Equivalent: $\theta_i \leftarrow \theta_i - \eta \frac{\partial J(\theta)}{\partial \theta_i} \frac{1}{\sqrt{s_i + \epsilon}}$
  - almost identical to Gradient Descent, but the gradient vector is scaled down by a factor of $\sqrt{s + \epsilon}$
  - $\oslash$: element-wise division
  - $\epsilon$: a smoothing term to avoid division by zero, typically set to $10^{-10}$



# Performance of AdaGrad

- This algorithm decays the learning rate, but faster for steep dimensions than for dimensions with gentler slopes. This is called an *adaptive learning rate*.
- It helps point the resulting updates more directly toward the global optimum
- stops too early



# (4) RMSProp

- Tijmen Tieleman and Geoffrey Hinton in 2012
- RMSProp
  - (1) $s \leftarrow \beta s + (1-\beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
    - AdaGrad: $s \leftarrow s + \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
  - (2) $\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{s+\epsilon}$
- decay rate $\beta$ is typically set to 0.9
  - exponential decay (next)
  - accumulating only the gradients from the most recent iterations (as opposed to all the gradients since the beginning of training)



# (5) Adam Optimization

- D. Kingma and J. Ba, Adam: A Method for Stochastic Optimization, ICLR, 2015.
- Adam: *adaptive moment estimation*
- (1) $m \leftarrow \beta_1 m + (1-\beta_1)\nabla_\theta J(\theta)$
  - Momentum optimization: keeps track of an exponentially decaying average of past gradients, $m_2 = \beta m_1 + \eta \nabla_\theta J(\theta_1), m_3 = \beta m_2 + \eta \nabla_\theta J(\theta_2) = \beta^2 m_1 + \beta\eta \nabla_\theta J(\theta_1) + \eta \nabla_\theta J(\theta_2), m_4 = \cdots$



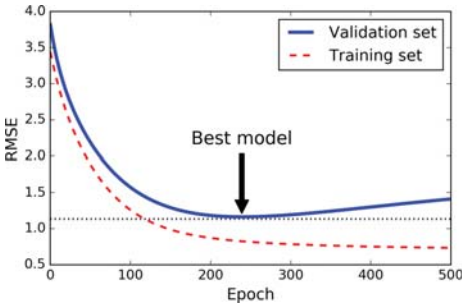
# Adam Algorithm

- (2) $s \leftarrow \beta_2 s + (1-\beta_2)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
  - RMSProp it keeps track of an exponentially decaying average of past squared gradients, $s \leftarrow \beta s + (1-\beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
- (3) $m \leftarrow \frac{m}{1-{\beta_1}^T}$
- (4) $s \leftarrow \frac{s}{1-{\beta_2}^T}$
- (5)$\theta \leftarrow \theta - \eta m \oslash \sqrt{s+\epsilon}$
- $T$ represents the iteration number (starting at 1).



# 7.3.6 Avoiding Overfitting (過度擬合) Through Regularization (正規化)

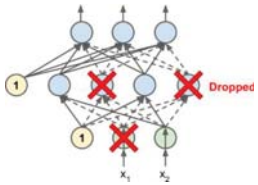
- 10.4.1 Early Stopping: just interrupt training when its performance on the validation set starts dropping




- 10.4.2 ℓ1 and ℓ2 Regularization

# Dropout (丟棄法)

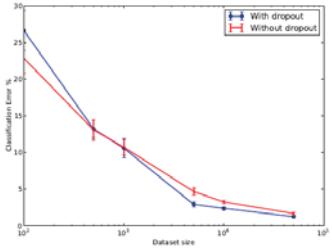
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *JMLR*, 2014.
- at every training step, every neuron a probability *p* dropped out, but it may be active during the next step.
- hyperparameter *p: dropout rate*, and typically set to 50%.
- got a 1–2% accuracy boost (準確性提升) simply by adding dropout

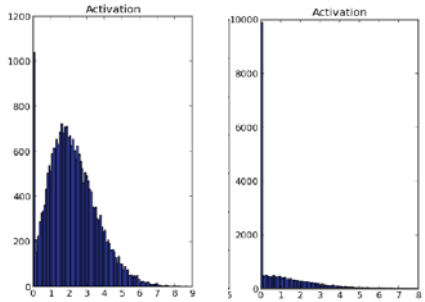





# Data set size and Effect on Sparsity

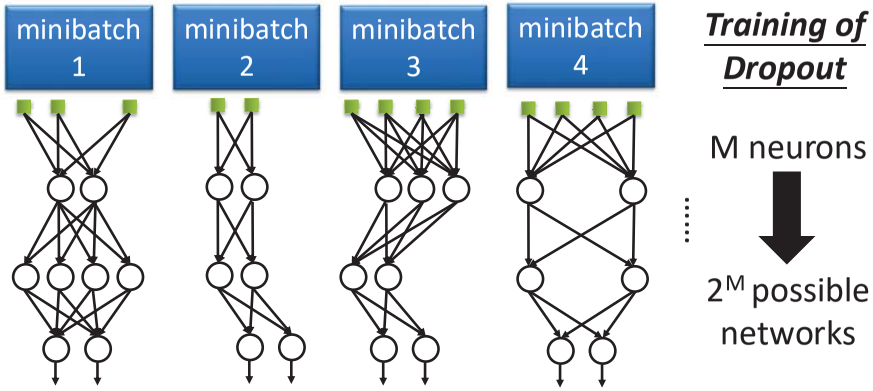
- classification experiments (分類實驗) on MNIST and vary the amount of data given to the network
  - data sets of size 100, 500, 1K, 5K, 10K and 50K chosen randomly
- ReLUs were used for both models.
- (left) Without dropout
  - a large fraction of units have high activation
- (right) Dropout with p = 0.5




# Dropout is a kind of ensemble learning (整體學習)

- 李宏毅, Deep Learning Tutorial

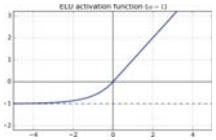



- Using one mini-batch to train one network
- Warde-Farley et al, An empirical analysis of dropout in piecewise linear networks, 2014

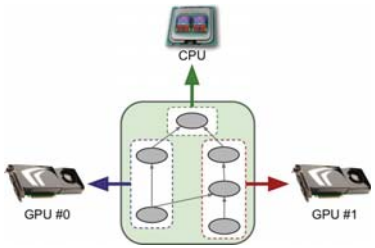


# 7.3.7 Practical Guidelines (實用指南)

- Default configuration
  - Initialization: He initialization (初始化)
  - Activation function: ELU
  - Normalization: Batch Normalization (批次標準化)
  - Regularization: Dropout
  - Optimizer: Adam
  - Learning rate schedule: None



- Chapter 12 Distributing (分散) TensorFlow Across Devices and Servers in Geron book



# Lecture 8 Keras and examples (例子)

- 8.1 Keras
- 8.2 MNIST
- 8.3 Predicting house prices (預測房價)
- 8.4 More information (更多資訊)
- Chollet, Deep Learning (深度學習) with Python
- 許志華



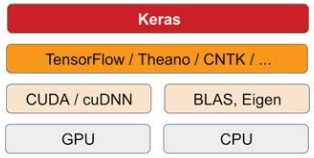
# 8.1 Keras

- key features (主要特點)
  - allows the same code to run seamlessly (無縫) on CPU or GPU.
  - a user-friendly (方便使用的) API that makes it easy to quickly prototype (雛型) deep-learning models.
  - built-in support for convolutional networks (for computer vision), recurrent networks (遞歸網路) (for sequence processing), and any combination of both.
  - It supports arbitrary network architectures (架構): multi-input or multi-output (多輸入多輸出) models, layer sharing (圖層共享), model sharing, and so on.



# Keras, TensorFlow, Theano, and CNTK

- Chollet, *Deep Learning with Python*
- Theano by the MILA lab, Université de Montréal
- TensorFlow by Google
- CNTK by Microsoft




- CPU: TensorFlow is itself wrapping (包) a low-level library (低階程式庫) for tensor operations (張量運算) called Eigen.
- GPU: TensorFlow wraps a library of well-optimized deep-learning operations called the NVIDIA CUDA Deep Neural Network library (cuDNN).



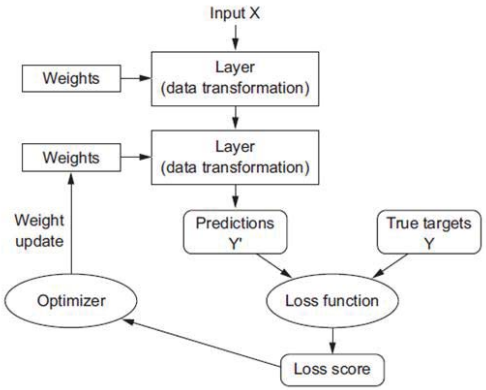
# Testing Keras

- `import keras`
- `keras.__version__` # input _ _ (no spacing)
- `Using TensorFlow backend.`
- `'2.1.4'`

- appendix B Running Jupyter notebooks on an EC2 GPU instance
  - The original (and up-to-date) version of this guide can be found at https://blog.keras.io.
  - Amazon EC2: Amazon Elastic Compute Cloud (亞馬遜彈性計算雲)

# 8.1.1 Training a neural network (訓練神經網路)

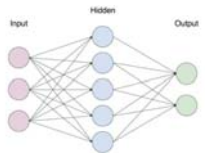
- The input data and corresponding targets (相對應的目標): 監督學習
- Layers (層): Combined into a network (or model) that maps your inputs to your targets
- The loss function (損失函數): the feedback signal (反饋信號) used for learning
- The optimizer (最佳化器): determines how learning proceeds






# Layers (層): Building blocks (基石) of deep learning (深度學習)

```
from keras import models
from keras import layers
model = models.Sequential # 模型：循序的
model.add(layers.Dense(5, input_shape=(3,)))
# input_shape 輸入形狀 3
# A dense layer (稠密層) with 5 output units
# dense: fully-connected (完全連接)
model.add(layers.Dense(2))
```




# Second layer: No input shape argument (輸入形狀參數)

# automatically inferred (自動推斷) its input shape as being the output shape (輸出形狀) of the layer that came before

https://hackernoon.com/artificial-neural-network-a843ff870338



# 8.2 MNIST

- MNIST dataset: a set of 70,000 small images of digits handwritten (手寫數字) by high school students and employees (員工) of the US Census Bureau (人口普查局)
  - 60,000 training images, 10,000 test images
  - assembled by the National Institute of Standards and Technology (國家標準與技術研究所) (the NIST in MNIST) in the 1980s.
  - M: Modified (with high school students)
- Python: lec08 MNIST-GPU
- `from keras.datasets import mnist`
- `(train_images, train_labels), (test_images, test_labels) = mnist.load_data() # 標籤 {0, 1, …,9}`



# MNIST dataset in NumPy arrays (陣列)

- `train_images.shape, train_images.dtype`
- `((60000, 28, 28), dtype('uint8'))`
- # 60000 個 28×28 影像
- # dtype: data type (資料類型)
- # unit8: unsigned integer (無符號整數) 8 bits
- `test_images.shape, test_images.dtype`
- `((10000, 28, 28), dtype('uint8'))`
- each grayscale image (灰階影像) 28x28 pixels (像素) = 784 features (特徵)
  - 每像素 8 位元，2^8 = 256
  - one pixel's intensity (強度): from 0 (black 黑色) to 255 (white 白色)

# Tensors (張量)

- TensorFlow
- data stored in multidimensional (多維) NumPy arrays (陣列)
- A tensor is a container (容器) for data
- A tensor is defined by three key attributes (關鍵屬性)
  - ((10000, 28, 28), dtype('uint8')) # 10000 個 28×28 影像
  - Number of axes (軸): A 3D tensor has three axes, and a matrix has two axes. The tensor's ndim in Python libraries
  - Shape (形狀): Integers that describes how many dimensions (維度) the tensor has along each axis. Now 10000, 28, 28
  - Data type (dtype, 資料類型): float32 (浮點), uint8 (unsigned integer 無符號整數), float64, and so on.



# Classification problem (分類問題)

- 監督學習
- 10 categories or classes (類別): {0, 1, 2, ..., 9}
- A label (標籤): The class associated with a specific sample (樣本): Now 5
- `import matplotlib.pyplot as plt`
- `plt.imshow(train_images[0,:,:], cmap = plt.cm.gray)`
- `# 第 0 個影像`
- `plt.show()`
- `train_images[0,7,:] # 第 8 個列 (row) ，28 個元素`
- `array([ 0, 0, 0, 0, 0, 0, 0, 49, 238, 253, 253, 253, 253, 253, 253, 253, 251, 93, 82, 82, 56, 39, 0, 0, 0, 0, 0], dtype=uint8)`
- 0 (black 黑色) to 255 (white 白色)



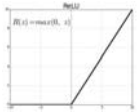
# 8.2.1 Data preprocessing and Training (資料預處理和訓練)

- `train_images1 = train_images.reshape((60000, 28 * 28))`
  `# train_images (60000, 28, 28)`
- `# reshape (重塑) , 28 * 28 圖片變行向量`
- `train_images1 = train_images1.astype('float32') / 255`
  `# as type (作為類型), [0, 1] 標準化`

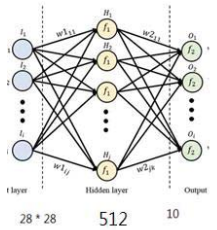
- `from keras.utils import to_categorical # 類別`
- `test_labels_categ = to_categorical(test_labels)`
- `train_labels_categ = to_categorical(train_labels)`
- `train_labels[0]`
- `5`
- `train_labels_categ[0]`
- `[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]`
- `# 0, 1, 2, 3, 4, 5, 6, 7, 8, 9`



# Build the neural network



- `from keras import models`
- `from keras import layers`
- `network = models.Sequential() # 循序`
- `# two Dense layers, which are densely-connected (also called "fully-connected" (完全連接))`
- `network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))`
- `# 每個 28 * 28 點都分別連到 512 點`
- `network.add(layers.Dense(10, activation='softmax'))`
- `# 10 個輸出 {0, 1, 2, …, 9}`
- `# 6.4 Softmax Regression (迴歸)`




https://www.researchgate.net/figure/Structure-of-a-multi-input-multi-output-neural-network_fig2_316738074

# Compiling the model (編譯模型)

```
network.compile(optimizer='rmsprop', # slide 7.41
                loss='categorical_crossentropy',
                metrics=['accuracy'])
# keras.io/optimizers/
# default lr = 0.001, (lr: learning rate (學習率))
# categorical_crossentropy: 類別的交叉熵
# accuracy (準確性): the fraction (分數) of the images
that were correctly classified (正確分類)
```

- Or

```
from keras import optimizers
network.compile(optimizer =
                optimizers.RMSprop(lr=0.001),
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```



# Train our network

- `network.fit(train_images1, train_labels_categ, epochs=5, batch_size=128, validation_data =(test_images1, test_labels_categ)) # GPU`
  - 5 epochs: 5 iterations over all samples (樣本) in the `train_images1` (X) and `train_labels_categ` tensors (y)
  - mini-batches (小批量) of 128 samples, 60000/ 128 = 468.75
- 1/60000 (training samples) = 16.667e-06, 1 μs (microsecond) = $10^{-6}$ s

```
Epoch 1/5
60000/60000 [==============================] - 2s 27us/step - loss: 0.2582 - acc: 0.9250
Epoch 2/5
60000/60000 [==============================] - 1s 23us/step - loss: 0.1040 - acc: 0.9681
Epoch 3/5
60000/60000 [==============================] - 1s 23us/step - loss: 0.0681 - acc: 0.9793
Epoch 4/5
60000/60000 [==============================] - 1s 22us/step - loss: 0.0493 - acc: 0.9851
Epoch 5/5
60000/60000 [==============================] - 1s 22us/step - loss: 0.0368 - acc: 0.9889
```



# Comparison of GPU and CPU speeds

- 1 μs = $10^{-6}$ s, (CPU / GPU time) 142 / 22 ≈ 6.45
- GPU: NVIDIA GTX 1070, RAM 24450 MB, Win10 64 bits

```
Epoch 1/5
60000/60000 [==============================] - 2s 27us/step - loss: 0.2582 - acc: 0.9250
Epoch 2/5
60000/60000 [==============================] - 1s 23us/step - loss: 0.1040 - acc: 0.9681
Epoch 3/5
60000/60000 [==============================] - 1s 23us/step - loss: 0.0681 - acc: 0.9793
Epoch 4/5
60000/60000 [==============================] - 1s 22us/step - loss: 0.0493 - acc: 0.9851
Epoch 5/5
60000/60000 [==============================] - 1s 22us/step - loss: 0.0368 - acc: 0.9889
```

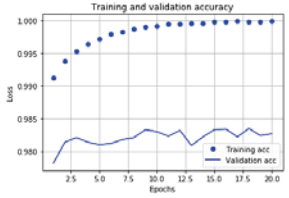
- CPU: Intel i3-2350m 2.3 ghz, Ram 4 gb, Win7 64 bits

```
Epoch 1/5
60000/60000 [==============================] - 11s 191us/step - loss: 0.2567 - acc: 0.9259
Epoch 2/5
60000/60000 [==============================] - 9s 146us/step - loss: 0.1042 - acc: 0.9695
Epoch 3/5
60000/60000 [==============================] - 9s 149us/step - loss: 0.0693 - acc: 0.9790
Epoch 4/5
60000/60000 [==============================] - 9s 142us/step - loss: 0.0494 - acc: 0.9852
Epoch 5/5
60000/60000 [==============================] - 9s 148us/step - loss: 0.0378 - acc: 0.9886
```



# 8.2.2 Performance (性能)

- Hidden nodes: 512, epochs = 20
- the training loss ↓ and accuracy (準確度) ↑ with every epoch
- Epoch 20
  - loss: 7.3601e-04, acc: 0.9999
  - val_loss: 0.1054, val_acc: 0.9827
- Training accuracy (準確度) 0.9999
  - 60000 samples (樣本), 59993 correct
  - 60000 – 59993 = 7 errors
- val_acc: 0.9827
  - 10000 samples, 9827 correct
  - 10000 – 9827 = 173 errors

# Using a trained network to generate predictions on new data

- `network.predict_classes(test_images1) # 10000類別`

```
array([7, 2, 1, ..., 4, 5, 6], dtype=int64)
```

- `network.predict(test_images1)[0,:] #第 0 個機率`

```
array([4.5772174e-23, 5.8948574e-26, 6.4225793e-18,
8.5518960e-14,5.1778876e-31, 1.0735723e-22,
1.0159417e-37, 1.0000000e+00,
4.2359035e-21, 4.1163427e-18], dtype=float32)
```

- `network.predict(test_images1)[9999,:] #第9999個機率`

```
array([1.9884683e-26, 0.0000000e+00, 1.1246934e-25,
1.7475806e-28,1.1125332e-23, 1.6612716e-23,
1.0000000e+00, 2.1005899e-37,
1.5144515e-32, 3.1339862e-33], dtype=float32)
```



# Confusion matrix (混淆矩陣)

- `import pandas as pd`
- `prediction = network.predict_classes(test_images1)`
- `pd.crosstab(test_labels, prediction,`
  `rownames=['label'], colnames=['predict'])`

- 對角線正確
- 其他錯誤：173
- val_acc: 0.9827
  - 10000 samples
  - 9827 correct
  - 173 errors

| predict | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| label | | | | | | | | | | |
| 0 | 971 | 0 | 0 | 1 | 1 | 1 | 3 | 1 | 2 | 0 |
| 1 | 1 | 1124 | 1 | 2 | 0 | 1 | 2 | 1 | 2 | 1 |
| 2 | 5 | 1 | 1008 | 2 | 3 | 0 | 2 | 5 | 5 | 1 |
| 3 | 1 | 0 | 1 | 995 | 0 | 4 | 0 | 3 | 3 | 3 |
| 4 | 1 | 0 | 1 | 1 | 969 | 0 | 4 | 2 | 0 | 4 |
| 5 | 2 | 0 | 0 | 5 | 1 | 877 | 3 | 0 | 3 | 1 |
| 6 | 4 | 3 | 1 | 1 | 4 | 3 | 942 | 0 | 0 | 0 |
| 7 | 0 | 4 | 7 | 1 | 2 | 0 | 0 | 1006 | 3 | 5 |
| 8 | 1 | 0 | 4 | 2 | 6 | 2 | 2 | 2 | 950 | 5 |
| 9 | 1 | 2 | 0 | 1 | 8 | 3 | 0 | 6 | 3 | 985 |



# Error analysis (錯誤分析) of 0

| predict | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| label | | | | | | | | | | |
| 0 | 971 | 0 | 0 | 1 | 1 | 1 | 3 | 1 | 2 | 0 |

```
for i in range(len(test_labels)):
    if (test_labels[i] != prediction[i]) & (test_labels[i] == 0):
```

- Predict 3 
- 4  5 
- 6   
- 7 (index 6597) 

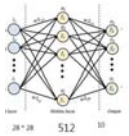
array([2.8e-07, 9e-34, 6.6e-13, 1.5e-07, 2e-32, 4.7e-16, 5e-35, 9.99e-01,3e-20, 4e-06])

- 8  



# 8.2.3 Overfitting (過度配適) and Underfitting (低度擬合)

- Slide 16: Hidden nodes 512, epochs = 20, acc: 0.9999, val_acc: 0.9827
  - Gap 0.9999 – 0.9827 = 0.0172




- `layers.Dense 128`
- `acc: 0.9981, val_acc: 0.9801, gap 0.0180`
- `Layers.Dense 64`
- `Acc: 0.9929, val_acc: 0.9749, gap 0.0180`
- `layers.Dense 16`
- `acc: 0.9603, val_acc: 0.9535, gap 0.0068`
- `layers.Dense 4`
- `acc: 0.8653, val_acc: 0.8681, gap 0.0028`

# Adding regularization (正規化)

- Without regularization: `layers.Dense 128`
- `acc: 0.9981, val_acc: 0.9801, gap: 0.0180`

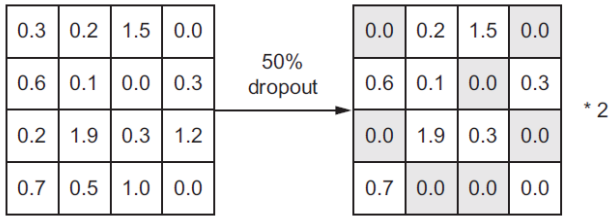
- `from keras import regularizers`
- `network.add(layers.Dense(128, kernel_regularizer = regularizers.l1(0.001), activation='relu', input_shape=(28 * 28,))) # first layer`
- `acc: 0.9564, val_acc: 0.9557, gap: 0.0007`

- Other:
  - keras.regularizers.l2(0.1)
  - keras.regularizers.l1_l2(l1 = 0.01, l2 = 0.01)



# Adding dropout (丟棄法) (1)

- Dropout applied to an activation matrix at training time, with rescaling (縮放) happening during training.
- At test time, the activation matrix is unchanged.






# Adding dropout (丟棄法) (2)

- Hidden nodes 512, acc: 0.9999, val_acc: 0.9827, gap 0.0172
- `layers.Dense(512, kernel_regularizer = regularizers.l1(0.001)`
- `acc: 0.9581, val_acc: 0.9537, gap 0.0044`
- `network.add(layers.Dropout(0.5))`
- `acc: 0.9895, val_acc: 0.9836, gap 0.0059`

- `layers.Dense(512, kernel_regularizer = regularizers.l1(0.001)`
- `model.add(layers.Dropout(0.5))`
- `acc: 0.9171, val_acc: 0.9474`
- `Google combine dropout and regularization`



# 8.2.4 Further experiments (進一步實驗)

- Try using 2 hidden or three hidden layers, and see how doing so affects validation and test accuracy (驗證和測試準確性)
- Try using layers with more hidden units or fewer hidden units.
- Try using the mse loss function instead of crossentropy.
- Try using the tanh activation instead of relu.
- Different dropout probability
- Different keras.regularizers.l2(0.1) and keras.regularizers.l1_l2(l1 = 0.01, l2 = 0.01)
- Different optimizers with different learning rate (學習率)
- https://keras.io/models/model/

# 8.3 Predicting house prices (預測房價)

- Python: lec08 Predicting house prices
- The Boston Housing Price dataset in the mid-1970s
  - X: 13 個, Per capita crime rate (人均犯罪率)….
  - y: The prices are typically between $10,000 and $50,000
    - Units: 1000, so train_targets: [15.2, 42.3,…
- a regression example (迴歸的例子)
- only 506 data points
  - split between 404 training samples (訓練樣本) and 102 test samples



# feature-wise normalization (特徵標準化) for the test data

- `train_data.mean(axis=0) # 13 個 X`
- `array([3.74511057e+00, 1.14801980e+01, 1.11044307e+01, 6.18811881e-02, 5.57355941e-01, 6.26708168e+00, 6.90106436e+01, 3.74027079e+00, 9.44059406e+00, 4.05898515e+02, 1.84759901e+01, 3.54783168e+02,1.27408168e+01])`
- `mean = train_data.mean(axis=0)`
- `train_data -= mean`
- `test_data -= mean`
- `std = train_data.std(axis=0) # 標準差`
- `train_data /= std`
- `test_data /= std`



# Building your network

- Because only 404 samples are available, you'll use a very small network with two hidden layers, each with 64 units.
  - one way to mitigate overfitting (減輕過度配適)

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu',
          input_shape=(train_data.shape[1],)))
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(1))
#  no activation function for scalar regression
model.compile(optimizer='rmsprop',
              loss='mse', metrics=['mae'])
```

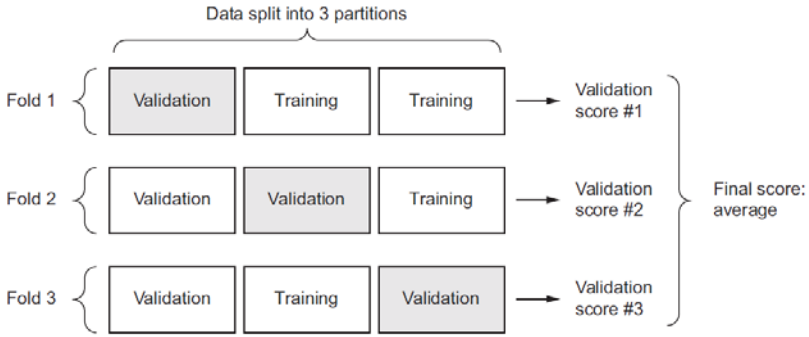
# mse loss function—mean squared error (均方誤差)

# mean absolute error (MAE, 平均絕對誤差): 0.5 predictions off by $500 on average

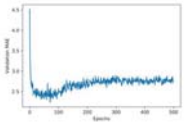


# Validating your approach using K-fold validation

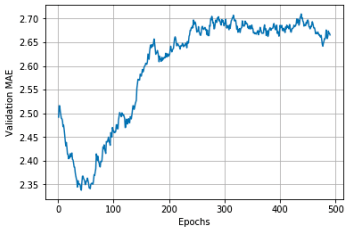
- 3-fold cross-validation (交叉驗證)



# Validation MAE by epoch



- difficult to see: scaling and high variance
- do the following:
  - Omit the first 10 data points: different scale
  - Replace each point with an exponential moving average (指數移動平均) of the previous points: previous * factor + point * (1 – factor), factor = 0.9
- MAE stops improving significantly after 80 epochs
- `test_mae_score`
- 2.6750272208569097
- still off by about $2,675.






# 8.4 More information (更多資訊)

- Auto-Keras
  - https://autokeras.com/
  - https://arxiv.org/pdf/1806.10282.pdf
  - https://autokeras.com/start/ (7 行程式)
- Auto-Sklearn
  - http://automl.github.io/auto-sklearn/stable/



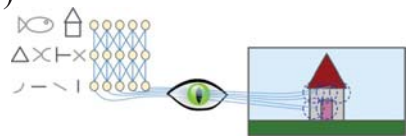
# Lecture 9 Deep learning for computer vision (計算機視覺)

- 9.1 Architecture (結構) of the Visual Cortex (視覺皮質)
- 9.2 convolutional neural networks (convnets) (卷積神經網路)
- 9.3 Training a convnet (CNN) from scratch
- 9.4 Using a pretrained convnet
- 9.5 Visualizing (視覺化) what convnets learn
- 9.6 Applications
- Chollet, Deep Learning (深度學習) with Python
- Geron, Hands-On Machine Learning With Scikit-Learn and Tensorflow
- 許志華



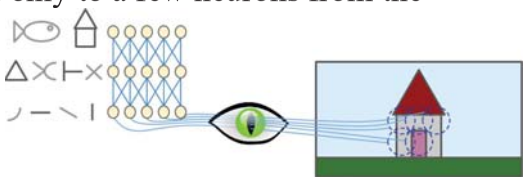
# 9.1 The Architecture (結構) of the Visual Cortex (視覺皮質)

- D.H. Hubel and T. Wiesel: experiments (實驗) on cats in 1958, Nobel Prize (諾貝爾獎) in Physiology (生理學) or Medicine in 1981
- Many neurons in the visual cortex have a small *local receptive field* (區域接受區)
  - react (反應) only to visual stimuli (刺激) located in a limited region (有限的區域) of the visual field
  - the local receptive fields of 5 neurons are represented by dashed circles (虛線圓圈)



# Local receptive field (區域接受區)

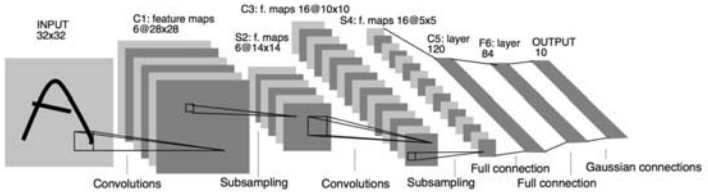
- The receptive fields of different neurons may overlap (重疊), and together they tile (鋪) the whole visual field.
- Some neurons react (反應) only to images of horizontal lines (水平線), or lines with different orientations (排列方向)
- Some neurons have larger receptive fields, and they react to more complex patterns (複雜的模式) that are combinations (組合) of the lower-level patterns. (each neuron connected only to a few neurons from the previous layer)





# 9.2 convolutional neural networks (convnets) (卷積神經網路)

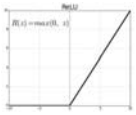
- Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-Based Learning Applied to Document Recognition (文件識別), *Proceedings of the IEEE*, Nov 1998. (LeNet-5 architecture)




  - convolutional layers (卷積層)
  - Subsampling (部分取樣): Pooling Layer (池化層)
  - the final layer (最後一層) outputs estimated class probabilities (輸出估計的類別機率)



# MNIST Example in Keras



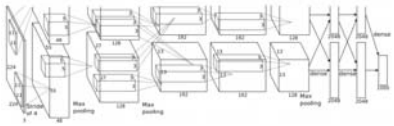
- Python: lec09-1 MNIST
- `model = models.Sequential()`
- `model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))` #卷積層， activation function (激勵函數)
- `model.add(layers.MaxPooling2D((2, 2)))` # 池化層
- `model.add(layers.Conv2D(64, (3, 3), activation='relu'))`
- `model.add(layers.MaxPooling2D((2, 2)))`
- `model.add(layers.Conv2D(64, (3, 3), activation='relu'))`



# ILSVRC ImageNet challenge (挑戰)

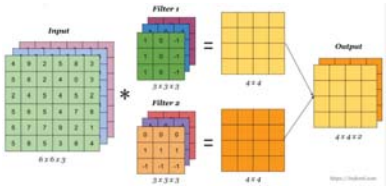
- a subset (子集) of ImageNet with roughly (大約) 1000 images in each of 1000 categories (類別)
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, ImageNet Classification with Deep Convolutional Neural Networks, *NIPS*, 2012. (AlexNet, 2 GPUs)
- The top-five error rate (前5個錯誤率): the system's top 5 predictions (預測) did not include the correct answer (正確答案).
- fell to barely (僅僅) over 3% in just five years…
- Human: 5 – 10%

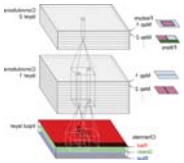
| Model | Top-1 | Top-5 |
|---|---|---|
| *Sparse coding [2]* | *47.1%* | *28.2%* |
| *SIFT + FVs [24]* | *45.7%* | *25.7%* |
| CNN | **37.5%** | **17.0%** |



# 9.2.1 Convolutional Layer (卷積層)

- Input images composed of (包含) multiple sublayers (多個子層)
  - Grayscale images (灰階影像): one channel (通道)
  - Three *channels*: red, green, and blue (RGB)
- Convolutions operate over 3D tensors, called *feature maps* (特徵地圖)
  - two spatial axes (空間軸): *height* (高) and *width* (寬度)
  - a *depth* axis (also called the *channels* axis).



https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/



# AlphaGo

- David Silver, Aja Huang (黃士傑), et al., Mastering the game of Go with deep neural networks and tree search (樹搜尋), *Nature*, 2016.
- Difficult: $3^{361}$ states. (棋盤 $19 \times 19 = 361$)
  - 暴力法不可行
- Neural network architecture (to approximate the policy)
  - Input: $19 \times 19 \times 48$
  - 48 feature planes
    - Stone colour: 3, Player stone, opponent stone, empty
    - Self-atari size: 8

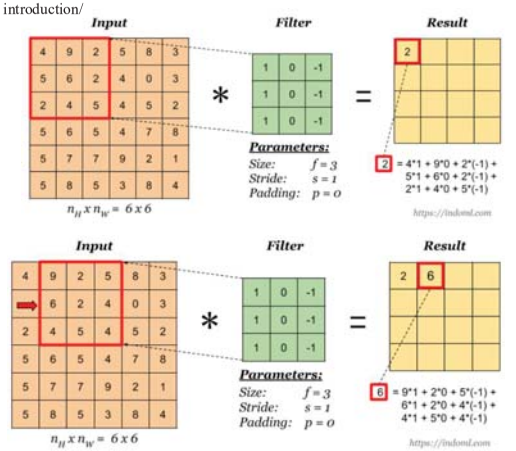
https://projectsplinter.com/faq/atari/



# Convolution Operation (卷積運算)

- http://cs231n.github.io/convolutional-networks/#conv
  - Animation (動畫)
  - Stanford


https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/



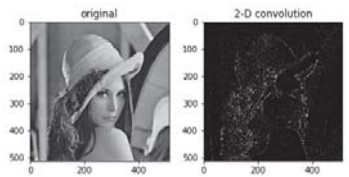

- Filter: 訓練得知
- 傳統 人找的
- 輸入 $6 \times 6$，輸出 $4 \times 4$
- 少 2

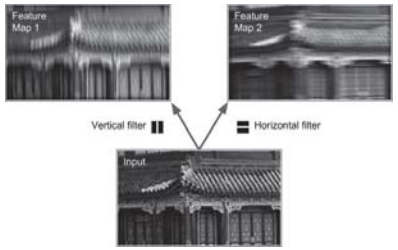


# Filters (過濾器) or convolution kernel (卷積核)

- edge detector

```
[[1, 1, 1]
[1, -8, 1]
[1, 1, 1]]
```



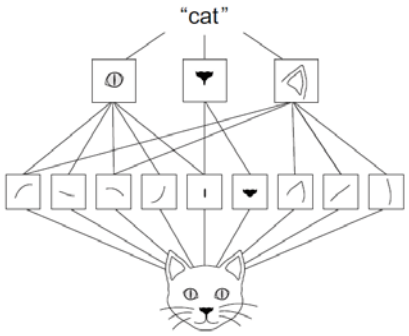




- A $7 \times 7$ matrix full of 0s (black) except for the central column (中間行) with 1s (white).
  - Ignore (忽略) everything in their receptive field (接受區) except for the central vertical line (垂線)
- The second: black square with a horizontal (水平的) white line in the middle

# key characteristic (關鍵特徵)

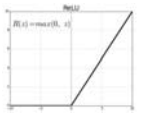
- The patterns they learn are translation invariant (學習的模式是平移不變的)
- They can learn spatial hierarchies (空間層次結構) of patterns

https://stats.stackexchange.com/questions/208936/what-is-translation-invariance-in-computer-vision-and-convolutional-neural-netwo

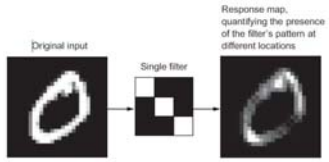



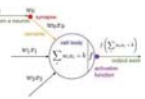



# MNIST Example



- `model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))`
- Input: a feature map (特徵地圖) of size (28, 28, 1)
- # MNIST 28 * 28，灰階 1
- the first convolution layer (卷積層)
  - filters (過濾器): 32 (自選的)
  - kernel_size (核大小): (3, 3) (自選)
    - 訓練得知其係數
  - Output: Response map of size (26, 26, 32)
    - 28 – 2 = 26
    - compute 32 filters over its input
  - Parameters (參數): 32 * (3 * 3 + 1) = 320 (the +1 the bias term)



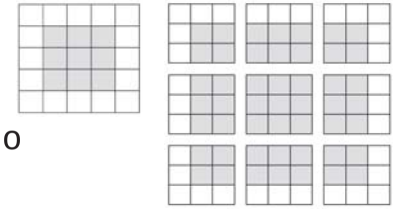



http://cs231n.github.io/neural-networks-1/

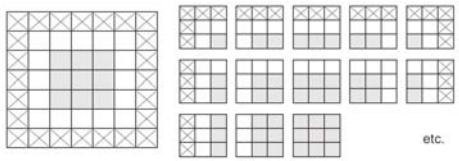


# 9.2.2 border effect (邊緣效應) and zero padding (補零)

- Valid locations of 3 x 3 patches (補釘) in a 5 x 5 input feature map (特徵地圖)
  - `padding='valid' # no`



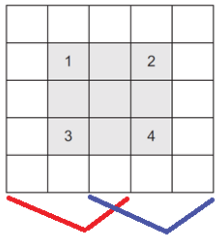
- *Padding:* If you want to get an output feature map with the same spatial dimensions (空間維度) as the input.
  - `padding='same'`

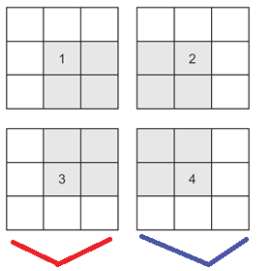


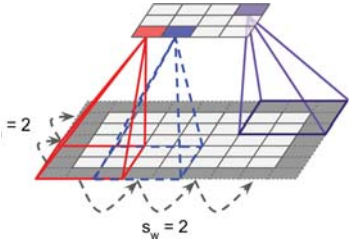



# stride (闊步)

- https://keras.io/layers/convolutional/
  - other parameters (參數) in `layers.Conv2D`
- Stride 2: The width and height of the feature map are downsampled (縮減採樣) by a factor (因子) of 2
  - Reducing dimensionality (維度)

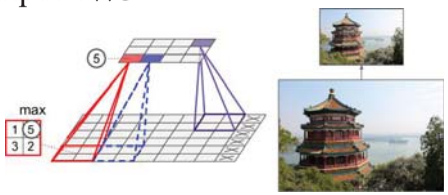


# 9.2.3 Pooling Layer (池化層)

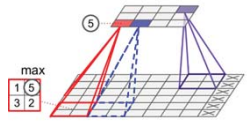
- Subsample (部分取樣) the input image:
  - Goal: reduce the computational load (計算負荷), the memory usage (記憶體使用量), and the number of parameters (參數)
  - thereby limiting the risk (限制風險) of overfitting
- Max pooling layer
  - 2 × 2 pooling kernel, stride 2, no padding
  - feature map 9 × 6, then output 4 × 3





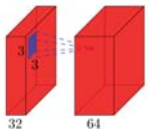

# 9.2.4 MNIST Example

- `model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))`
- # Output: a feature map of size (26, 26, 32), 32自選
- `model.add(layers.MaxPooling2D((2, 2)))`
- # Output Shape: 13, 13, 32
- `# https://keras.io/layers/pooling/`
- `# MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid')`
- `# strides = None: default to pool_size`
- `# no padding`






# MNIST Example



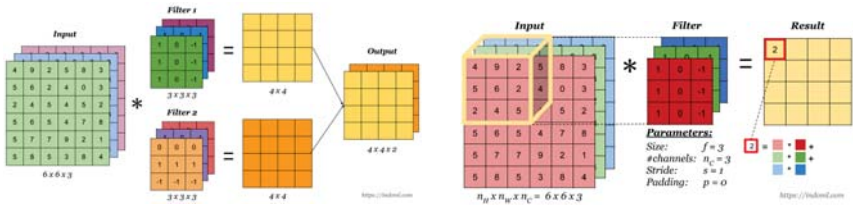

- `model.add(layers.Conv2D(64, (3, 3), activation='relu'))`
- Output: a feature map of size (11, 11, 64), 13 – 2 = 11
- Parameters (參數): 32 * 64 * 3 * 3 + 64 (bias) = 18496
- `model.add(layers.MaxPooling2D((2, 2)))`
- Output: a feature map of size (5, 5, 64)
- `model.add(layers.Conv2D(64, (3, 3), activation='relu'))`
- Output: a feature map of size (3, 3, 64)
- Parameters: 64 * 64 * 3 * 3 + 64 = 36928
- # 3 `Conv2D, 2 MaxPooling2D`

https://i.stack.imgur.com/2r4XG.png



# Computing the output of a neuron in a convolutional layer

- https://indoml.com/2018/03/07/student – notes – convolutional – neural – networks – cnn – introduction/



# Adding a classifier on top of the convnet

- `model.add(layers.Flatten())`
- # Output Shape: 576 # from 3 * 3 * 64
- `model.add(layers.Dense(64, activation='relu'))`
- # Parameters (參數): 576 * 64 + 64 = 36928
- `model.add(layers.Dense(10, activation = 'softmax')) # MNIST 10 個輸出`
- # Parameters: 64 * 10 + 10 = 650



# evaluate the model on the test data

- `model.fit(train_images, train_labels, epochs=5, batch_size=64) # labels (標籤)`
- `# MNIST 監督學習 分類`
- `test_loss, test_acc = model.evaluate(test_images, test_labels)`
- `test_acc`
- 0.99129999999999996

- the densely connected network: test accuracy of 97.8%
  - Error rate (錯誤率): 1 – 97.8% = 2.2%
- the basic convnet has a test accuracy of 99.1%
  - Decreased the error rate by ($\frac{2.2-0.9}{2.2} \approx$) 59%



# 9.2.5 Why downsample (縮減採樣) feature maps (特徵地圖) this way?

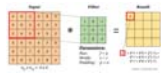
- Why not remove the max-pooling layers and keep fairly large (相當大的) feature maps all the way up?
- `model_no_max_pool.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))`
- `model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))`
- `model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))`
- `model_no_max_pool.summary()`

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 22, 22, 64)        36928
=================================================================
Total params: 55,744
```



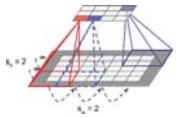
# What's wrong with this setup?

- It isn't conducive (不利於) to learning a spatial hierarchy (空間層次結構) of features (特徵).
  - The 3 × 3 windows in the third layer (第三層) will only contain information coming from 7 × 7 windows in the initial input.
- The final feature map has 22 × 22 × 64 = 30,976 total coefficients (係數) per sample.

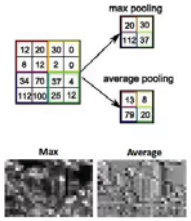


  - Comparison: a feature map of size (3, 3, 64) before
  - flatten it to stick a Dense layer of size 512 on top, that layer would have (30,976 × 512 ≈) 15.8 million parameters.
  - This is far too large for such a small model and would result in intense overfitting.

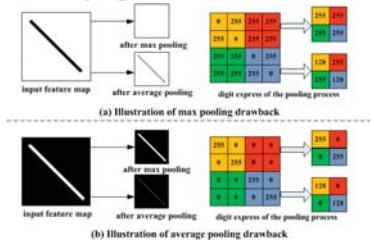
# Other approaches to achieve such downsampling (縮減採樣)



- use strides in the prior convolution layer
- use average (平均) pooling instead of max pooling
  - (12 + 20 + 8 + 12) / 4 = 13
  - more informative to look at the *maximal presence* of different features than at their *average presence*.




https://www.quora.com/What-is-the-benefit-of-using-average-pooling-rather-than-max-pooling




https://www.researchgate.net/figure/Toy-example-illustrating-the-drawbacks-of-max-pooling-and-average-pooling_fig2_300020038



# 9.3 Training a convnet (CNN) from scratch (從頭開始) on a small dataset (小數據集)

- This dataset contains 25,000 images of dogs and cats (12,500 from each class).
  - medium-resolution color (中等解析度的彩色) JPEGs
- create a new dataset containing 4,000 pictures of cats and dogs (2,000 cats, 2,000 dogs), 3 subsets (子集)
  - 2,000 pictures for training (訓練)
  - 1,000 for validation (驗證)
  - 1,000 for testing (測試)
- Objective: Classify (分類) images as dogs or cats



# Why 3 sets?

- You train on the training data (訓練資料) and evaluate your model on the validation (驗證) data.
- Once your model is ready for prime time (準備好黃金時間), you test it one final time on the test (測試) data.
- Why not have two sets: a training set and a test set?
  - Tuning the configuration (調整配置) of the model based on its performance on the validation set can quickly result in *overfitting to the validation set*, even though your model is never directly trained on it.



# Results

- Training a small convnet on the 2,000 training samples, without any regularization (正規化)
  - a baseline (基線) for classification accuracy of 71%.
  - overfitting.
- introduce *data augmentation* (資料擴增): a powerful technique for mitigating overfitting in computer vision.
  - Reach an accuracy of 82%.
- feature extraction (特徵萃取) with a pretrained network (accuracy of 90% to 96%)
- fine-tuning a pretrained network (final accuracy of 97%)

# 9.3.1 Building your network (建立網路)

- Python: lec09-2 convnets-dog-cat
  - 課堂中從此開始執行
  - 前面是從 25,000 中，取出 4,000 張影像
- `model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))`
- inputs of size 150 × 150 (arbitrary choice 任意選擇)
- `model.add(layers.MaxPooling2D((2, 2)))`
- # 150 – 2 = 148, size of feature map: 148 / 2 = 74
- `model.add(layers.Conv2D(64, (3, 3), activation='relu'))`
- `model.add(layers.MaxPooling2D((2, 2)))`
- `# size of feature map: 72 / 2 = 36`



# Building your network

- `# size of feature map: 36`
- `model.add(layers.Conv2D(128, (3, 3), activation='relu'))`
- `model.add(layers.MaxPooling2D((2, 2)))`
- `# size of feature map: 34 / 2 = 17`
- `model.add(layers.Conv2D(128, (3, 3), activation='relu'))`
- `model.add(layers.MaxPooling2D((2, 2)))`
- `# size of feature map: 15 / 2 = 7 (from 148)`
- `# depth of feature map: from 32 to 128`
- `#4 Conv2D, 4 MaxPooling2D`
- `model.add(layers.Flatten())`
- `model.add(layers.Dense(512,activation='relu'))`
- `model.add(layers.Dense(1,activation='sigmoid'))`



# 9.3.2 Data preprocessing (資料前處理) and training (訓練)

- Currently, the data sits on a drive as JPEG files, so the steps for getting it into the network are roughly as follows:
  1) Read the picture files.
  2) Decode the JPEG content to RGB grids of pixels (像素網格)
  3) Convert these into floating-point tensors (浮點張量)
  4) Rescale (重新調整) the pixel values (between 0 and 255) to the [0, 1] interval
- `from keras.preprocessing.image import ImageDataGenerator`
- # All images will be rescaled by 1./255
- `train_datagen = ImageDataGenerator(rescale=1./255)`



# Using **ImageDataGenerator** to read images from directories (目錄)

- ```
  train_generator = train_datagen.flow_from_directory(
          # This is the target directory (目標目錄)
          train_dir,
          # All images will be resized to 150x150
          target_size=(150, 150),
          batch_size=20, # 批量大小
          # binary_crossentropy loss, binary labels
          class_mode='binary') #交叉熵
  ```
- ```
  validation_generator = test_datagen.flow_from_directory(
          validation_dir,
          target_size=(150, 150),
          batch_size=20,
          class_mode='binary')
  ```

# A Python generator (產生器)

- train_generator = train_datagen.flow_from_directory(
  train_dir,
  target_size=(150, 150),
  batch_size = 20, # 批量大小
  class_mode='binary') #交叉熵

- for data_batch, labels_batch in train_generator:
  print('data batch shape:', data_batch.shape)
  print('labels batch shape:',labels_batch.shape)
  break
- data batch shape: (20, 150, 150, 3) # 3: RGB images
- labels batch shape: (20,)



# Fitting the model using a batch generator (批量產生器)

- history = model.fit_generator(
  train_generator,
  steps_per_epoch=100, # 2000 images / 20 samples per batch
  epochs=30, # 訓練 30 大回合
  validation_data = validation_generator,
  validation_steps = 50) # 1000 images / 20

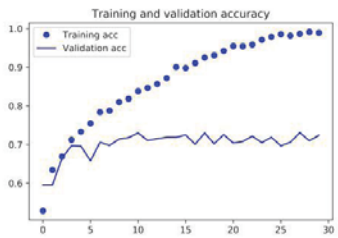
```
Epoch 1/30
100/100 [==============================] - 7s 74ms/step - loss: 0.6901 - acc: 0.5270 - val_loss: 0.6740 - val_acc: 0.6040
Epoch 2/30
100/100 [==============================] - 7s 67ms/step - loss: 0.6513 - acc: 0.6250 - val_loss: 0.6440 - val_acc: 0.6150
```

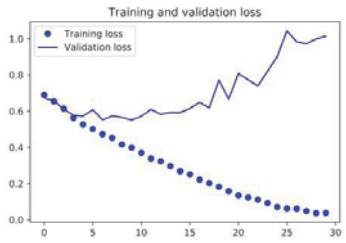
- # save your model after training
- model.save(' lec09_cats_and_dogs_small_1.h5')



# loss and accuracy (損失和準確性)

- 30 epochs
- The training accuracy (訓練準確性) increases linearly (線性增加) over time, until it reaches nearly 100%, whereas the validation (驗證) accuracy stalls (陷入) at 70–72%.
- Overfit



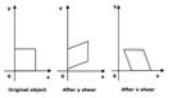


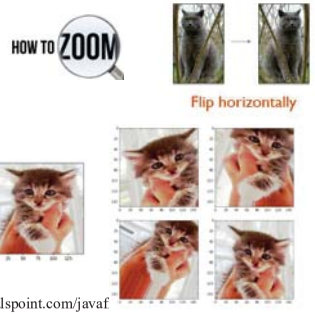

# 9.3.3 Using data augmentation (資料擴增)

- Overfitting is caused by having too few samples to learn from, rendering (使得) you unable (無法) to train a model that can generalize to new data.
- Data augmentation takes the approach of generating more training data (產生更多的訓練資料) from existing training samples, by *augmenting* the samples via a number of random transformations (隨機變換) that yield believable-looking (可信的樣子) images.

# Setting up a data augmentation configuration via ImageDataGenerator

```
datagen = ImageDataGenerator(
    # a value in degrees (0–180), a range within which to
    # randomly rotate pictures (旋轉圖片)
    rotation_range=40,
    # ranges (as a fraction of total width or
    # height) within which to randomly translate
    # pictures vertically or horizontally.
    width_shift_range=0.2,
    height_shift_range=0.2,
    # shearing transformations (剪切變換)
    shear_range=0.2,
    zoom_range=0.2, #縮放
    horizontal_flip=True, # 水平翻轉
    # fill in newly created pixels
    fill_mode='nearest')
```






https://www.youtube.com/watch?v=USNR5ckU9Es
https://www.cnblogs.com/mengnan/p/9489716.html

https://www.tutorialspoint.com/javaf



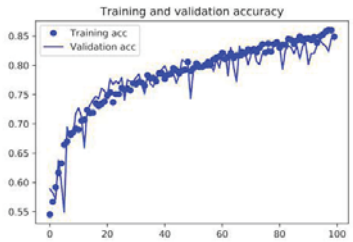
# Training the convnet using data-augmentation (資料擴增) generators

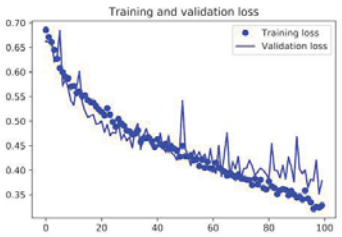
- `train_datagen = ImageDataGenerator(rescale=1./255, …)`
- `test_datagen = ImageDataGenerator(rescale=1./255)`
- `# the validation data shouldn't be augmented!`
- `batch_size=32`
- `steps_per_epoch=100`
- Per epoch
  - 2000 training images
  - pick 32 samples per batch
  - Do data-augmentation
  - 100 iterations (so (32 * 100 =) 3200 images in total)



# data augmentation and dropout (資料擴增和丟棄法)

- `model.add(layers.Flatten())`
- `model.add(layers.Dropout(0.5)) # fight overfitting`
- `model.add(layers.Dense(512, activation='relu'))`
- The validation curves (驗證曲線) are closely tracking (密切追踪) training curves
- the 100th epoch: acc: 0.8609, val_acc: 0.8344






# 9.4 Using a pretrained convent (預訓練的卷積神經網路)

- A *pretrained network* is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task (影像分類任務).
  - consider a large convnet trained on the ImageNet dataset
    - 1.4 million labeled images and 1,000 different classes
    - classes are mostly animals and everyday objects
  - then repurpose this trained network for something as remote as identifying furniture (識別家具) items in images.
- two ways to use a pretrained network: *feature extraction* (特徵抽取) and *fine-tuning* (微調)

# use the VGG16 architecture (架構)

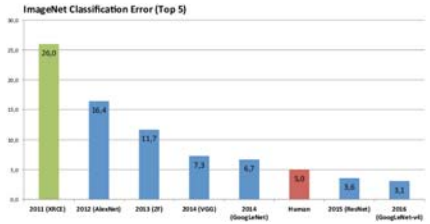
- Karen Simonyan and Andrew Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv, 2014. (in ICLR, 2015)
- depth to 16–19 weight layers
- a simple and widely used convnet architecture for ImageNet
  - ImageNet Challenge 2014 submission: the first and the second places in the localisation and classification tracks respectively (University of Oxford)
  - a special case of object detection (物體偵測): a single object bounding box (邊界框) should be predicted for each of the top-5 classes


https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/



# ImageNet

- O. Russakovsky, et al., ImageNet Large Scale Visual Recognition Challenge (大規模視覺識別挑戰), *International Journal of Computer Vision*, 2015.
- S. Bianco, et al., Benchmark Analysis of Representative Deep Neural Network Architectures, IEEE Access, 2018
- green bar: the best computer vision approach
- AlexNet: 8 layers
- VGG:19
- ResNet: 52
- GoogLeNet: 22




https://www.researchgate.net/figure/Winner-results-of-the-ImageNet-large-scale-visual-recognition-challenge-LSVRC-of-the_fig7_324476862



# Training ResNet-50 on ImageNet

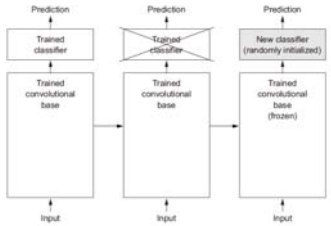
- Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda, Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes, NIPS'17 Workshop: Deep Learning at Supercomputer Scale (超級電腦等級)
  - Training Procedure for Large Minibatches
  - 90 epochs in 15 minutes with 1024 NVIDIA Tesla P100 GPUs (Nikola Tesla：交流供電系統)
  - top-1 accuracy

| Team | Hardware | Software | Minibatch size | Time | Accuracy |
|---|---|---|---|---|---|
| He *et al.* [5] | Tesla P100 × 8 | Caffe | 256 | 29 hr | 75.3 % |
| Goyal *et al.* [4] | Tesla P100 × 256 | Caffe2 | 8,192 | 1 hr | 76.3 % |
| Codreanu *et al.* [3] | KNL 7250 × 720 | Intel Caffe | 11,520 | 62 min | 75.0 % |
| You *et al.* [10] | Xeon 8160 × 1600 | Intel Caffe | 16,000 | 31 min | 75.3 % |
| This work | Tesla P100 × 1024 | Chainer | 32,768 | 15 min | 74.9 % |



# 9.4.1 Feature extraction (特徵萃取)

- Use the representations learned by a previous network to extract interesting features from new samples.
- Swapping (交換) classifiers while keeping the same convolutional base
- Why? the feature maps of a convnet are presence maps (存在地圖) of generic concepts (一般概念) over a picture, which is likely to be useful (有用) regardless of the computer-vision problem at hand



# Instantiating the VGG16 convolutional base (實例化VGG16卷積基)

- Python: lec09-3-using-a-pretrained-convnet
- `from keras.applications import VGG16`
- `conv_base = VGG16(weights='imagenet',`
  `# pre-training on ImageNet`
  `include_top=False,`
  `# include (or not) the densely connected (密集連接) classifier`
  `input_shape=(150, 150, 3))`
  `# purely optional, can process inputs of any size`
- available as part of `keras.applications`
  - `Xception, Inception V3, ResNet50, VGG16, VGG19, MobileNet, MobileNetV2, InceptionResNetV2, DenseNet, NASNet`



`conv_base.summary()`

- Total params: 14,714,688
- The final feature map has shape (4, 4, 512).
- That's the feature on top of which you'll stick a densely connected (密集連接) classifier.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | (None, 150, 150, 3) | 0 |
| block1_conv1 (Conv2D) | (None, 150, 150, 64) | 1792 |
| block1_conv2 (Conv2D) | (None, 150, 150, 64) | 36928 |
| block1_pool (MaxPooling2D) | (None, 75, 75, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 75, 75, 128) | 73856 |
| block2_conv2 (Conv2D) | (None, 75, 75, 128) | 147584 |
| block2_pool (MaxPooling2D) | (None, 37, 37, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 37, 37, 256) | 295168 |
| block3_conv2 (Conv2D) | (None, 37, 37, 256) | 590080 |
| block3_conv3 (Conv2D) | (None, 37, 37, 256) | 590080 |
| block3_pool (MaxPooling2D) | (None, 18, 18, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 18, 18, 512) | 1180160 |
| block4_conv2 (Conv2D) | (None, 18, 18, 512) | 2359808 |
| block4_conv3 (Conv2D) | (None, 18, 18, 512) | 2359808 |
| block4_pool (MaxPooling2D) | (None, 9, 9, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 9, 9, 512) | 2359808 |
| block5_conv2 (Conv2D) | (None, 9, 9, 512) | 2359808 |
| block5_conv3 (Conv2D) | (None, 9, 9, 512) | 2359808 |
| block5_pool (MaxPooling2D) | (None, 4, 4, 512) | 0 |



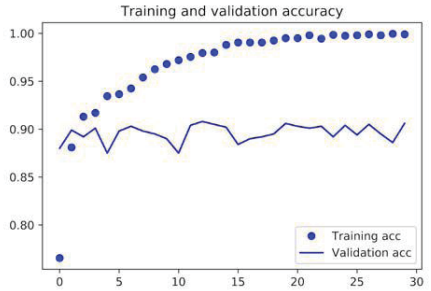
# 9.4.1.1 Fast feature extraction without data augmentation

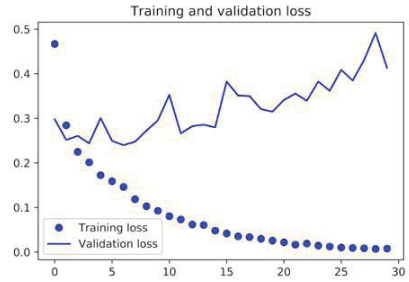
- You'll start by running instances of the previously introduced ImageDataGenerator to extract images as Numpy arrays as well as their labels (標籤).
- extract features from these images by calling the `predict` method of the `conv_base` model:
  `features_batch = conv_base.predict(inputs_batch)#20`
  - The extracted features are currently of shape (samples, 4, 4, 512).
- You'll feed them to a densely connected classifier (密集連接的分類器), so first you must flatten (弄平) them to (samples, 8192).
- then define your densely connected classifier (note the use of dropout for regularization) and train it on the data and labels that you just recorded.



# Results

- Training is very fast, because you only have to deal with two Dense layers—an epoch takes less than one second even on CPU.
- a validation accuracy (驗證準確性) of about 90%



# 9.4.1.2 Feature extraction with data augmentation

- `model = models.Sequential()`
- `model.add(conv_base)`
- `model.add(layers.Flatten())`
- `model.add(layers.Dense(256, activation='relu'))`
- `model.add(layers.Dense(1, activation='sigmoid'))`
- `model.summary()`

```
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
```



# Freezing a layer (凍結一層) or set of layers

- prevent their weights from being updated during training
- `print(len(model.trainable_weights))`
- `30 # number of trainable weights before freezing the conv base (13), (13 + 2) * 2 = 30`
- `# two per layer (the main weight matrix and the bias vector)`
- `conv_base.trainable = False`
- `print(len(model.trainable_weights))`
- `4`
- `model.add(layers.Dense(256, activation='relu'))`
- `model.add(layers.Dense(1, activation='sigmoid'))`



# Results

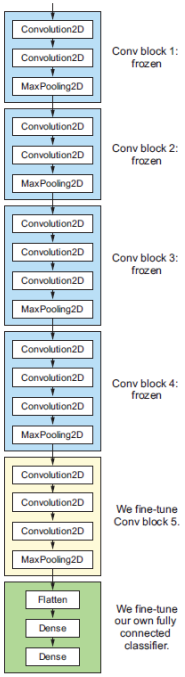
- a validation accuracy (驗證準確性) of about 96%.
  - from scratch: 70%
  - data augmentation (資料擴增): 82%
  - Fast feature extraction without data augmentation: 90%
  - Human: 95%





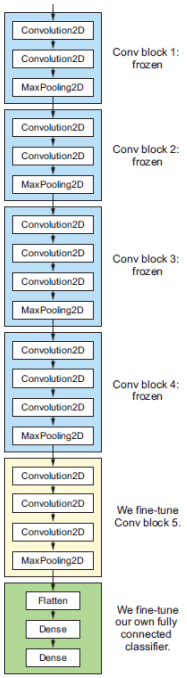
# 9.4.2 Fine-tuning (微調)

- Fine-tuning the last convolutional block of the VGG16 network
- steps for fine-tuning a network are as follow:
  1) Add your custom network on top of an already-trained base network.
  2) Freeze (凍結) the base network.
  3) Train the part you added.
  4) Unfreeze some layers in the base network.
  5) Jointly train both these layers and the part you added.



# (4) Unfreeze Conv block 5
# (5) Jointly train

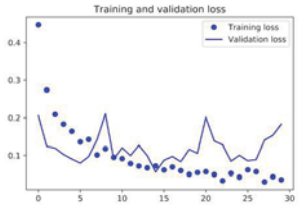
- `conv_base.trainable = True`
- `set_trainable = False`
- 
```
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```
- RMSProp optimizer
  - using a very low learning rate 1e-5
  - limit the magnitude of the modifications (限制修改的幅度) you make to the representations of the three layers you're fine-tuning



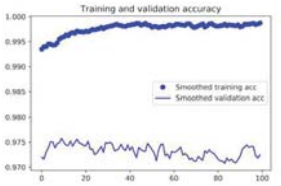



# validation accuracy

- a validation accuracy of about 97%.
  - Feature extraction with data augmentation 96%


- smooth by exponential moving averages (指數移動平均)







# Test accuracy

- 
```
test_generator = test_datagen.flow_from_directory(
      test_dir,  target_size=(150, 150),
      batch_size=20, class_mode='binary')
```
- `test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)`
- `print('test acc:', test_acc)` # 0.967999992371
- By using modern deep-learning techniques, you managed to reach this result using only a small fraction (小部分) of the training data available (about 10%).
  - a huge difference (巨大的差異) between being able to train on 20,000 samples compared to 2,000 samples!



# 9.4.3 Epistemology of Deep Learning (深度學習的知識論)

- Yann LeCun, seminar talk at the Institute for Advanced Study, 2019/2/22
- Engineering science: inventing new artifacts (發明新文物)
- Natural science: discovery (發現), study and explain phenomena (解釋現象)
- Theory often follows invention (跟隨發明)
  - Steam engine (蒸汽機) [1695-1715], thermodynamics (熱力學) [1824 -]
  - Computer [1941 – 1945], computer science [1950-1960]
  - Teletype (電傳) [1906], Information Theory (資訊理論) [1948]

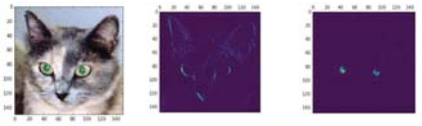
# 9.5 Visualizing (視覺化) what convnets learn

- It's often said that deep-learning models are "black boxes" (黑盒子)
  - learning representations (學習表示) that are difficult to extract and present in a human-readable form (人類可讀的形式)
- Since 2013, a wide array of techniques have been developed for visualizing and interpreting (視覺化和解釋) these representations.
- Python: lec09-4-visualizing-what-convnets-learn



# 9.5.1 Visualizing intermediate activations (視覺化中間激勵函數)

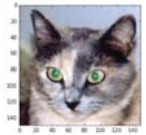
- Display the feature maps that are output by various convolution and pooling layers in a network, given a certain input
  - the output of a layer is often called its *activation*, the output of the activation function
- This gives a view into how an input is decomposed into (分解成) the different filters learned by the network.





# Visualizing intermediate activations (視覺化中間激勵函數)

- `from keras.models import load_model`
- `model = load_model('cats_and_dogs_small_2.h5')`
- get an input image—a picture of a cat, not part of the images the network was trained on

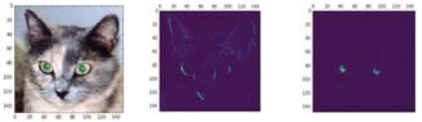


- Instantiating a model from an input tensor and a list of output tensors
- `from keras import models`
- `layer_outputs = [layer.output for layer in model.layers[:8]]`
- # Extracts the outputs of the top eight layers
- `activation_model = models.Model(inputs=model.input, outputs=layer_outputs)`



# Running the model in predict mode

- `activations = activation_model.predict(img_tensor)`
- the activation of the first convolution layer for the cat image input
- `first_layer_activation = activations[0]`
- `print(first_layer_activation.shape)`
- `(1, 148, 148, 32)` # a 148 × 148 feature map with 32 channels
- `plt.matshow(first_layer_activation[0, :, :, 4], cmap='viridis')`
- `plt.matshow(first_layer_activation[0, :, :, 7], cmap='viridis')`

Every channel of every layer activation on the test cat picture

- 4 convolution layers
  - `model.add(layers.Conv2D(32, (3, 3),`
  - `model.add(layers.Conv2D(64, (3, 3),`
  - `model.add(layers.Conv2D(128, (3, 3),`
  - `model.add(layers.Conv2D(128, (3, 3),`
- Every channel of every layer activation on the test cat picture



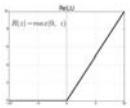
# a few things to note here

- The first layer acts as a collection of various edge detectors (各種邊緣檢測器).
- As you go higher, the activations become increasingly abstract and less visually interpretable. They begin to encode higher-level concepts (編碼更高級別的概念) such as "cat ear" and "cat eye."
- The sparsity of the activations increases with the depth of the layer

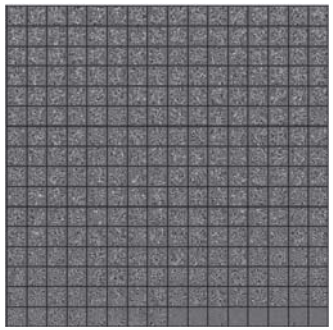


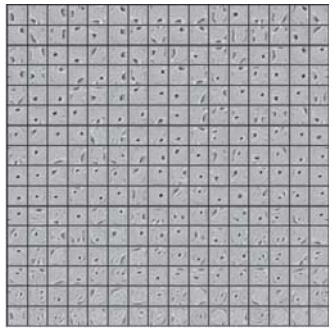
# Features learned on MNIST

- N. Srivastava, et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *JMLR*, 2014.
- Features learned on MNIST with one hidden layer autoencoders having 256 rectified linear units



- (a) Without dropout (b) Dropout with p = 0.5

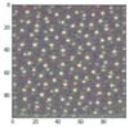






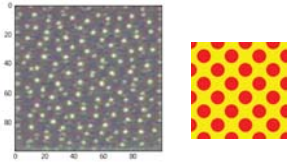
# 9.5.2 Visualizing convnet filters (視覺化 convent 的過濾器)

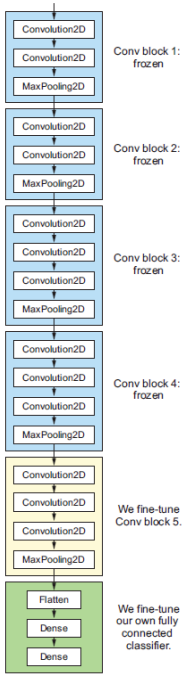
- display the visual pattern that each filter is meant to respond to
  - This can be done with *gradient ascent* (梯度上升) *in input space*: applying *gradient descent* to the value of the input image of a convnet so as to *maximize* the response of a specific filter, starting from a blank input image.
  - The resulting input image will be one that the chosen filter is maximally responsive to (最大程度地響應).



# One filter

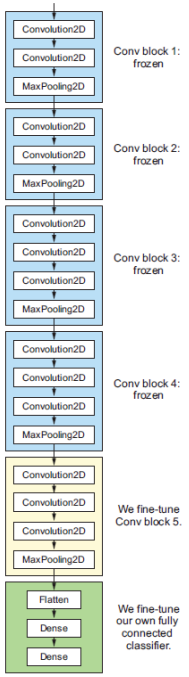
- VGG16 network pretrained on ImageNet
  - filter 0 in layer block3_conv1 is responsive to a polka-dot pattern (波爾卡圓點圖案)
- Polka dots are commonly seen on children'sclothing, toys, furniture, ceramics, and Eastern European folk art (東歐民間藝術) but they appear in a wide array of contexts.





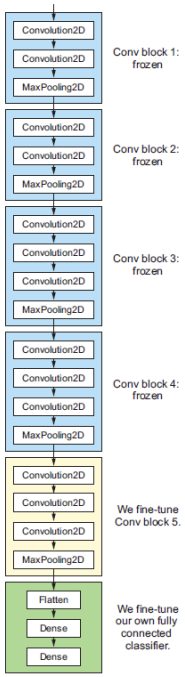

# Filter patterns for layer block1_conv1

- look at the first 64 filters in each layer
- some black margins (黑色邊緣) between each filter pattern
- encode simple directional edges and colors



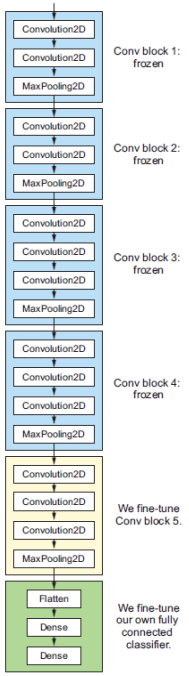

# Filter patterns for layer block2_conv1

- encode simple textures (紋理) made from combinations of edges and colors



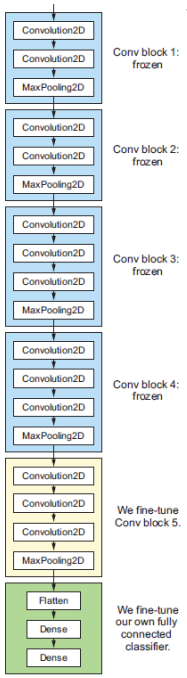

# Filter patterns for layer block3_conv1

- The filters in higher layers begin to resemble textures found in natural images: feathers, eyes, leaves, and so on.



# Filter patterns for layer block4_conv1

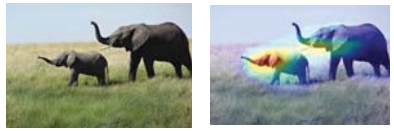
- The filters in higher layers begin to resemble textures found in natural images: feathers, eyes, leaves, and so on.






# 9.5.3 Visualizing heatmaps of class activation (視覺化激勵類別的熱圖)

- produce heatmaps of class activation over input images.
  - A class activation heatmap is a 2D grid of scores associated with a specific output class, computed for every location in any input image, indicating how important (多麼重要) each location is with respect to (關於) the class under consideration.
- R. R. Selvaraju, at al., Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, arXiv, 2017.
  - class activation map (CAM)





# two African elephants (非洲大象)

- VGG16 network pretrained on ImageNet
- `preds = model.predict(x)`
- `print('Predicted:', decode_predictions(preds, top=3)[0])`
- `Predicted:', [(u'n02504458', u'African_elephant', 0.92546833), # with 92.5% probability`
- `(u'n01871265', u'tusker', 0.070257246),`
- `(u'n02504013', u'Indian_elephant', 0.0042589349)]`

https://www.shutterstock.com/zh-Hant/image-photo/dika-tusker-512692519?src=bCVAIbW6h7YwlEEuoSgIQw-1-0

https://www.worldwildlife.org/species/indian-elephant



# To visualize via the Grad-CAM process

- African elephant class activation heatmap over the test picture
- use OpenCV to generate an image that superimposes (疊加) the original image on the heatmap you just obtained
- `[(u'n02504458', u'African_elephant', 0.92546833), # with 92.5% probability`
- `(u'n02504013', u'Indian_elephant', 0.0042589349)]`

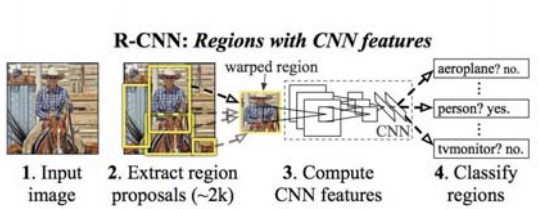
# 9.6 Applications (應用)

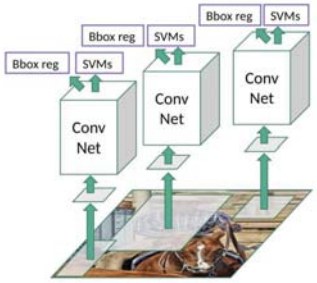
- 9.6.1 Deep Learning for Object Detection (物體檢測的深度學習)
- 9.6.2 Arrhythmia (心律不整)
- 9.6.3 皮膚癌
- 9.6.4 糖尿病視網膜病變
- scholar.google.com.tw
  - Convolutional Neural Network manufacturing, marketing, …



# 9.6.1 Object Detection Algorithms (物體檢測演算法)

- Rohith Gandhi, R-CNN, Fast R-CNN, Faster R-CNN, YOLO—Object Detection Algorithms, Towards Data Science, Jul 10, 2018
- R-CNN, or Region-based Convolutional Neural Network (基於區域的卷積神經網路), 2013.
- self-driving cars



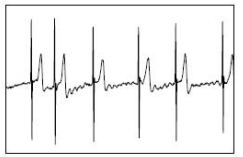
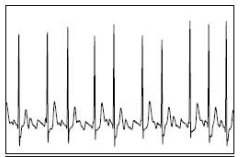
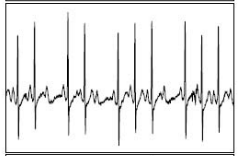






# 9.6.2 Arrhythmia (心律不整)

- https://kknews.cc/education/kz8k2aq.html
- 如果患者疑似有心律不齊症狀，通常會去醫院做個心電圖。但是，在醫院做的幾分鐘或者十幾分鐘心電圖，往往發現不了問題，這時候醫生就可能會給患者一個可穿戴的心電圖監測器，要戴兩周。
- 兩周，就要產生幾百小時的心電圖數據，醫生要一秒一秒地檢查，從中找出患者是否出現了有問題的心律不齊。
- 更難的是，還得把有問題的心律不齊和安全無害的心律不齊區分開，它們在心電圖上看起來非常像。



| Class | Description | Example | Train + Val Patients | Test Patients |
|---|---|---|---|---|
| AFIB | Atrial Fibrillation |  | 4638 | 44 |
| AFL | Atrial Flutter |  | 3805 | 20 |
| AVB_TYPE2 | Second degree AV Block Type 2 (Mobitz II) |  | 1905 | 28 |
| BIGEMINY | Ventricular Bigeminy |  | 2855 | 22 |

# Arrhythmia (心律不整)

- P. Rajpurkar, A.Y. Hannun, M. Haghpanahi, C. Bourn, A.Y. Ng, Cardiologist-Level (心臟病專家級) Arrhythmia Detection with Convolutional Neural Networks, arXiv:1707.01836, 2017

- Problem Formulation: The ECG (心電圖) arrhythmia detection task is a sequence-to-sequence task
  - input an ECG signal $X = [x_1, \ldots, x_k]$
  - outputs a sequence of labels $r = [r_1, \ldots, r_n]$; each $r_i$ can take on one of m different rhythm classes (類別)
- Train a 34-layer convolutional neural network



# Training

- We collect and annotate a dataset of 64,121 ECG (心電圖) records from 29,163 patients.
- The ECG data is sampled at a frequency (取樣頻率) of 200 Hz and is collected from a single-lead, noninvasive and continuous monitoring device (連續監測裝置) called the Zio Patch which has a wear period up to 14 days.
- Each ECG record in the training set is 30 seconds long and can contain more than one rhythm type.
- Each record is annotated (註釋) by a clinical (臨床) ECG expert: the expert highlights segments of the signal and marks it as corresponding to one of the 14 rhythm classes.

https://www.irhythmtech.com/professionals/why-zio



# Testing (測試)

- https://kknews.cc/education/kz8k2aq.html
- We collect a test set of 336 records from 328 unique patients (患者).
- For the test set, ground truth annotations (基本真相註釋) for each record were obtained by a committee of three board certified cardiologists (認證的心臟科醫生); there are three committees responsible for different splits of the test set.
- The cardiologists discussed each individual record as a group and came to a consensus labeling (共識標籤).
- For each record in the test set we also collect 6 individual annotations from cardiologists not participating in the group.
- This is used to assess performance of the model compared to an individual (個別的) cardiologist.



# Model vs. Cardiologist (心臟科醫生) Performance

- (查準率) precision = $\frac{TP}{TP+FP}$ (猜對)
- (查全率) recall = $\frac{TP}{TP+FN}$ (找到)
- *Harmonic mean* (調和平均數) $F_1 = \frac{2}{\frac{1}{precesion}+\frac{1}{recall}}$

| | Seq | | Set | |
|---|---|---|---|---|
| | Model | Cardiol. | Model | Cardiol. |
| Class-level F1 Score | | | | |
| AFIB | **0.604** | 0.515 | **0.667** | 0.544 |
| AFL | **0.687** | 0.635 | **0.679** | 0.646 |
| AVB_TYPE2 | **0.689** | 0.535 | **0.656** | 0.529 |
| BIGEMINY | **0.897** | 0.837 | **0.870** | 0.849 |
| CHB | **0.843** | 0.701 | **0.852** | 0.685 |
| EAR | **0.519** | 0.476 | **0.571** | 0.529 |
| IVR | **0.761** | 0.632 | **0.774** | 0.720 |
| JUNCTIONAL | 0.670 | **0.684** | **0.783** | 0.674 |
| NOISE | **0.823** | 0.768 | **0.704** | 0.689 |
| SINUS | **0.879** | 0.847 | **0.939** | 0.907 |
| SVT | **0.477** | 0.449 | **0.658** | 0.556 |
| TRIGEMINY | **0.908** | 0.843 | **0.870** | 0.816 |
| VT | 0.506 | **0.566** | 0.694 | **0.769** |
| WENCKEBACH | **0.709** | 0.593 | **0.806** | 0.736 |
| Aggregate Results | | | | |
| Precision (PPV) | **0.800** | 0.723 | **0.809** | 0.763 |
| Recall (Sensitivity) | **0.784** | 0.724 | **0.827** | 0.744 |
| F1 | **0.776** | 0.719 | **0.809** | 0.751 |

# 9.6.3 皮膚癌

- A. Esteva, B. Kuprel, R.A. Novoa, J. Ko, S.M. Swetter, Helen M. Blau & S. Thrun, Dermatologist-level classification of skin cancer with deep neural networks, Nature, February 2017.
- 129,450 clinical images (臨床影像)
- 2,032 different diseases (不同的疾病)
- binary classification use cases
  - keratinocyte carcinomas (角質形成細胞癌) versus benign seborrheic keratoses: identification of the most common cancers
  - malignant melanomas (惡性黑色素瘤) versus benign nevi: the identification of the deadliest skin cancer (最致命的皮膚癌)



# Performance

- The CNN achieves performance on par with (與...相提並論) all tested experts across both tasks, demonstrating an artificial intelligence (人工智慧) capable of classifying skin cancer (皮膚癌) with a level of competence comparable to dermatologists (能力可比皮膚科醫生)
  - kknews.cc/tech/k8e94ab.html
  - 在 21 位經過認證的皮膚科醫生的監督下



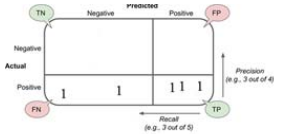
# 9.6.4 糖尿病視網膜病變

- V. Gulshan, et al., Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy (糖尿病性視網膜病變) in Retinal Fundus Photographs (視網膜眼底照片). JAMA (美國醫學會雜誌). 2016 Dec 13;316(22):2402-2410.
- Use a deep convolutional neural network
- Validation Set Performance for All-Cause Referable Diabetic Retinopathy in the EyePACS-1 Data Set (9946 Images) (next)



# Performance of the algorithm (black curve) and ophthalmologists (眼科醫師) (colored circles)

- (敏感度) sensitivity $= \frac{TP}{TP+FN}$ (找到): 大好




- (特異性) specificity $= \frac{TN}{TN+FP}$
  - 1 – spec: 小好