Lab3 - Tuning the Training Process

Advisor：Tsai, Chia-Chi

TA：楊宗軒

Outline

1. Lab3 task

1. ResNet18

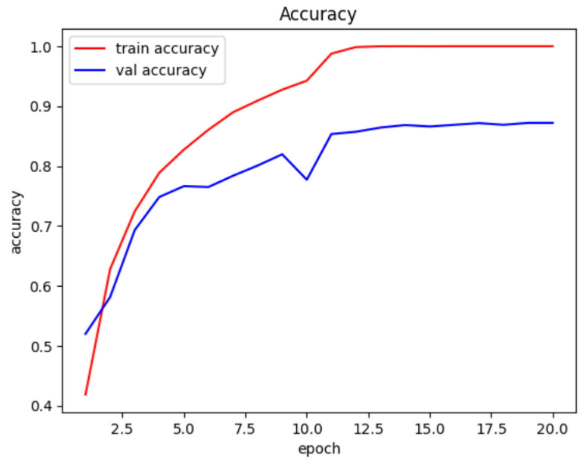
1. CIFAR-10

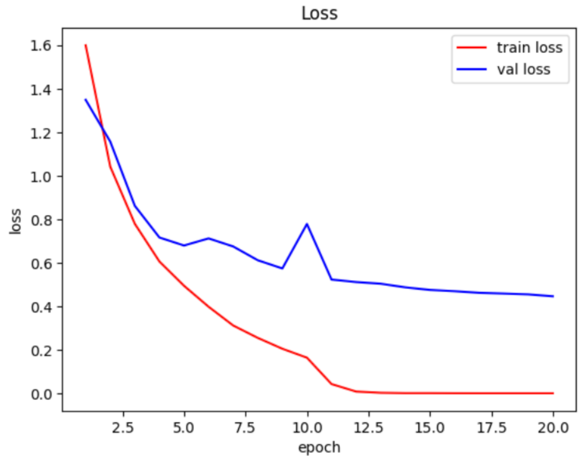
Tasks

- Use Resnet18 to train on CIFAR-10
- Using Pytorch
- Experiment on the following and compare the result with baseline
  - Input image normalization (10%)
  - Data augmentation (10%)
  - Different base learning rate and update strategy (10%)
  - Different batch size (10%)
  - Complete resnet18.py(20%)
- Print test loss and test acc
- Plot train-loss, val-loss, train-acc, val-acc
- Accuracy(>90%拿滿，>80%才有基本分) (10%)
- Report (30%)

Report your result

- Print test loss and test acc
- Plot train-loss, val-loss, train-acc, val-acc






```
Epoch: 14
learning rate:  0.025
Train loss: 0.003 | Train acc: 1.000
Val loss: 0.488 | Val acc: 0.868

Epoch: 15
learning rate:  0.025
Train loss: 0.003 | Train acc: 1.000
Val loss: 0.476 | Val acc: 0.866

Epoch: 16
learning rate:  0.025
Train loss: 0.002 | Train acc: 1.000
Val loss: 0.471 | Val acc: 0.869

Epoch: 17
learning rate:  0.025
Train loss: 0.002 | Train acc: 1.000
Val loss: 0.463 | Val acc: 0.872

Epoch: 18
learning rate:  0.025
Train loss: 0.002 | Train acc: 1.000
Val loss: 0.460 | Val acc: 0.869

Epoch: 19
learning rate:  0.025
Train loss: 0.002 | Train acc: 1.000
Val loss: 0.456 | Val acc: 0.872

Epoch: 20
learning rate:  0.0125
Train loss: 0.002 | Train acc: 1.000
Val loss: 0.447 | Val acc: 0.872
Test loss: 0.465 | Test acc: 0.866
```

Image normalization

- min/max normalization: 縮到0~1或-1~1之間，通常是input range已知的情況可用，output = input / 255

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- transforms.ToTensor(): range(0, 255) -> range(0.0, 1.0)

Image normalization

- Standardization: 將sampled dataset的mean和std轉換成接近於0和1，以此減少偏差，避免被某部分資料支配，通常是用在Input range未知的情況，以採樣的方式來取得。

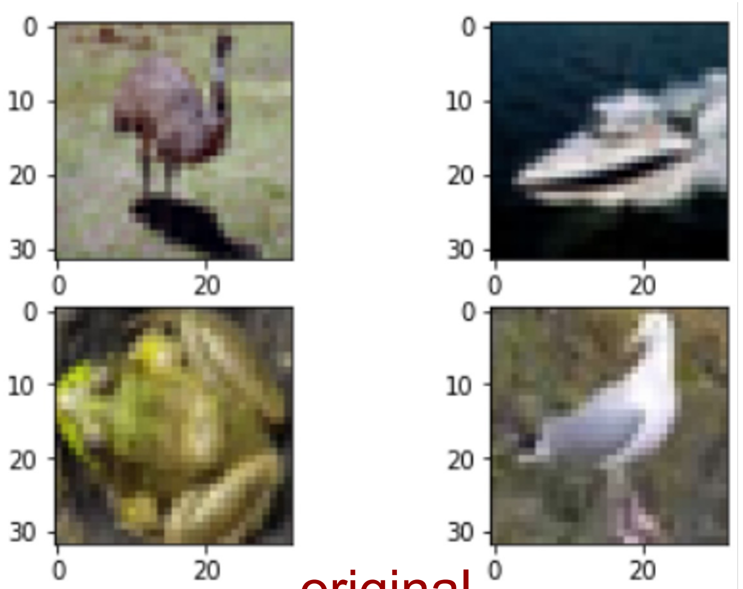
- mean -> 0 ：unbiased的data更有利於model學習

- std -> 1 ：減緩梯度消失和梯度爆炸

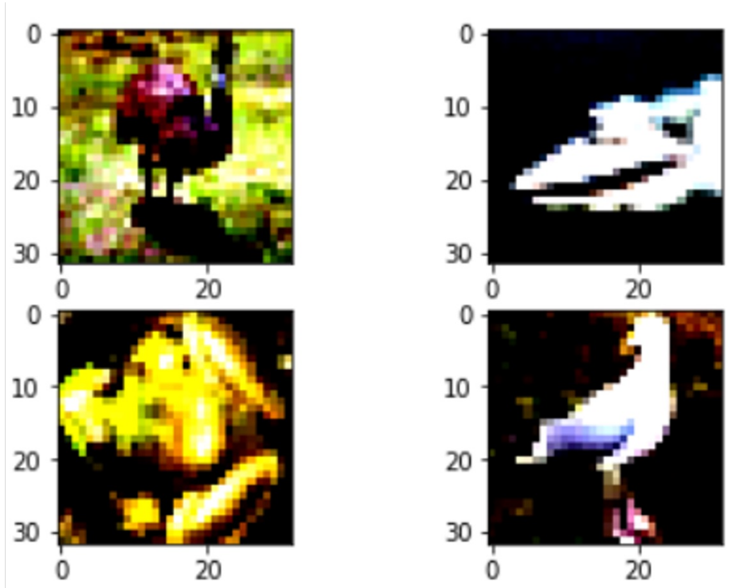
```python
# 計算normalization需要的mean & std
def get_mean_std(dataset, ratio=0.3):
    # Get mean and std by sample ratio
    dataloader = torch.utils.data.DataLoader(dataset, batch_size=int(len(dataset)*ratio), shuffle=True, num_workers=2)

    data = next(iter(dataloader))[0]     # get the first iteration data
    mean = np.mean(data.numpy(), axis=(0,2,3))
    std = np.std(data.numpy(), axis=(0,2,3))
    return mean, std

train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transforms.ToTensor())
test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transforms.ToTensor())

train_mean, train_std = get_mean_std(train_dataset)
test_mean, test_std = train_mean, train_std
print(train_mean, train_std)
print(test_mean, test_std)
```

Data augmentation

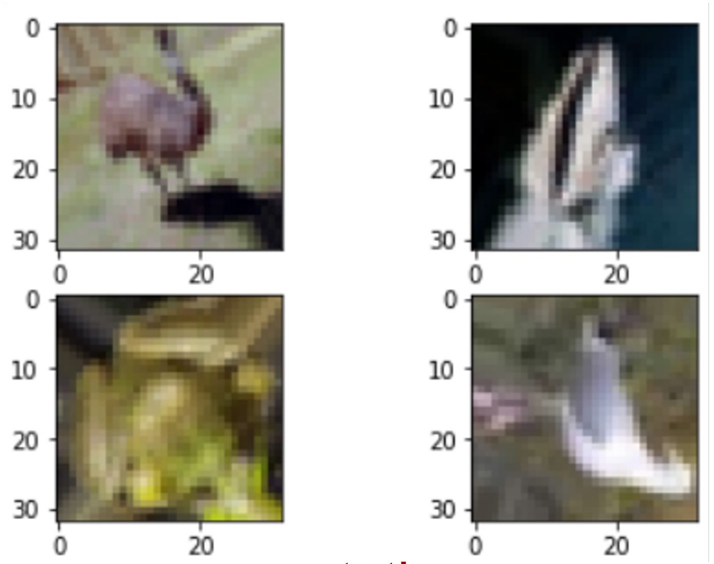
- 為了在training樣本固定的情況下，藉由不改變被辨識物件特性(例如classification種類)，對image做一些改動，來讓training data更多元化
- 將圖片進行旋轉、調整大小、比例尺寸，或改變亮度色溫、翻轉、加入Gaussian noise等處理
- [Transforming and augmenting images — Torchvision main documentation]



original



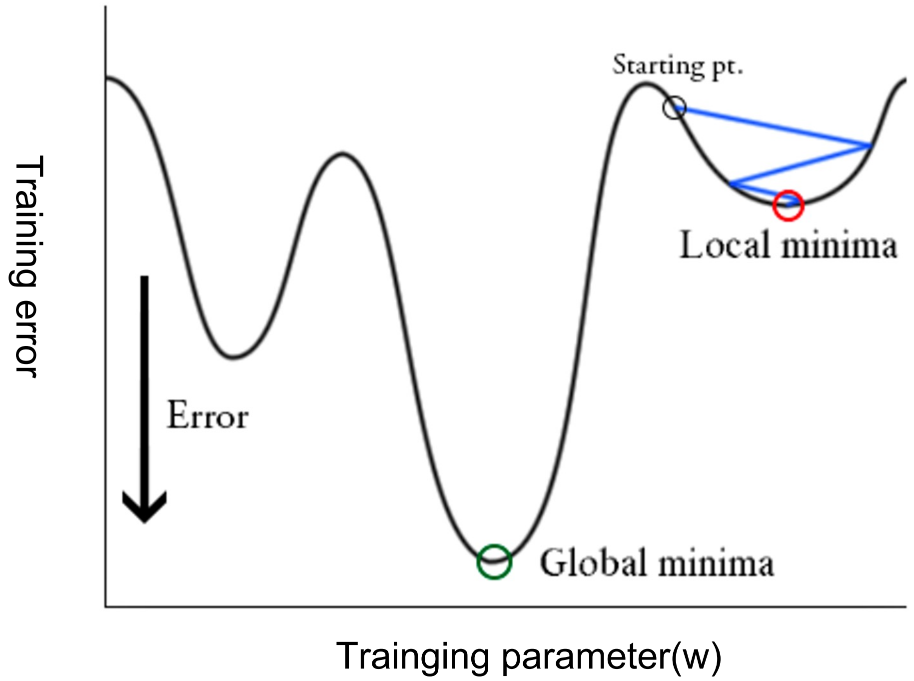
standardization



rotation

Learning rate and update strategy

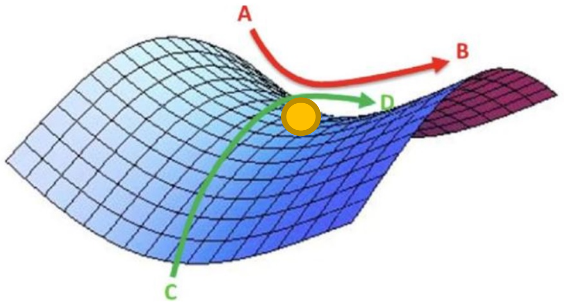
- 訓練model時，若採用固定的learning rate，容易找到local minima而非global minima

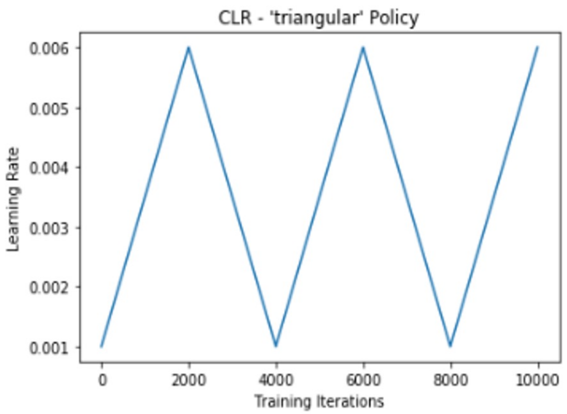


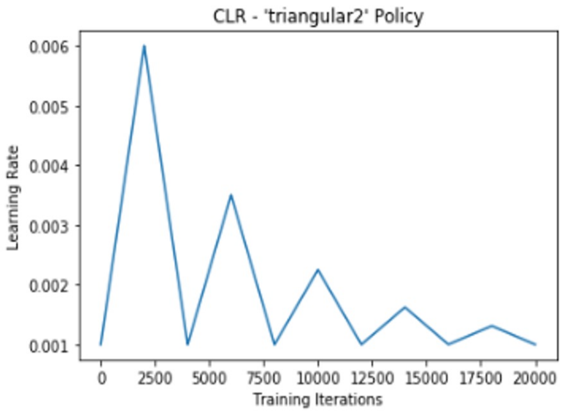
Learning rate and update strategy

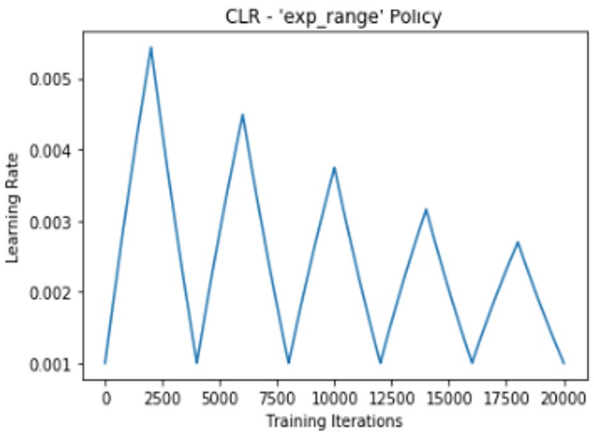
- Learning Rate Decay : 通常在訓練一定epoch後，會對學習率進行衰減，從而讓model收斂得更好，但不斷的縮小learning rate也有缺點(陷入saddle point)

- 所以也有人使用Cyclical Learning Rates: 設定學習率的上下限後，讓學習率在一定範圍內衰降或增加，在遇到saddle point不會卡住









Batch size

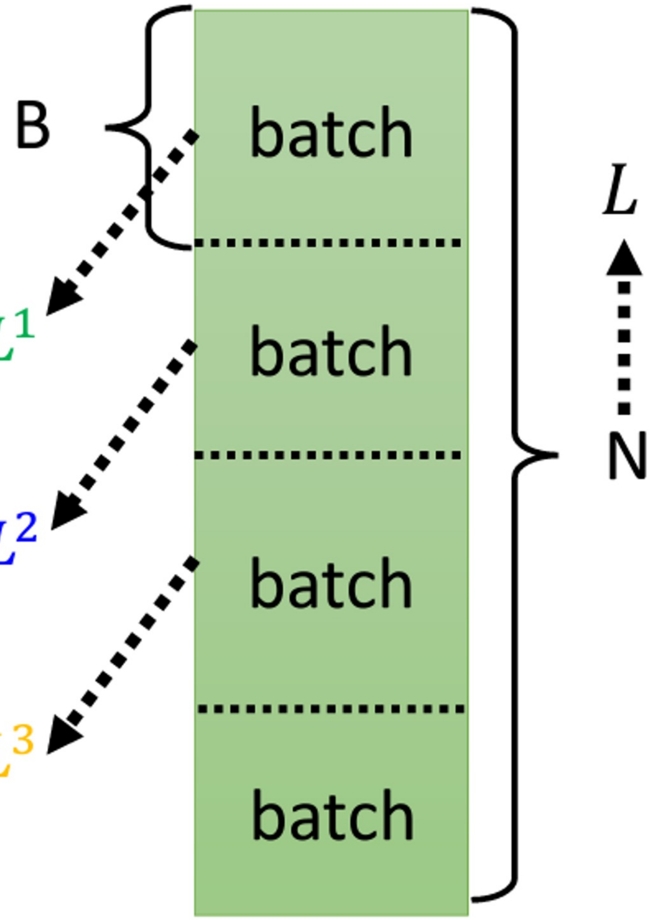
$$\boldsymbol{\theta}^* = arg \min_{\boldsymbol{\theta}} L$$

- (Randomly) Pick initial values $\boldsymbol{\theta}^0$
- Compute gradient $\boldsymbol{g}^0 = \nabla L^1(\boldsymbol{\theta}^0)$ L^1

  update $\boldsymbol{\theta}^1 \leftarrow \boldsymbol{\theta}^0 - \eta \boldsymbol{g}^0$
- Compute gradient $\boldsymbol{g}^1 = \nabla L^2(\boldsymbol{\theta}^1)$ L^2

  update $\boldsymbol{\theta}^2 \leftarrow \boldsymbol{\theta}^1 - \eta \boldsymbol{g}^1$
- Compute gradient $\boldsymbol{g}^3 = \nabla L^3(\boldsymbol{\theta}^2)$ L^3

  update $\boldsymbol{\theta}^3 \leftarrow \boldsymbol{\theta}^2 - \eta \boldsymbol{g}^3$



1 **epoch** = see all the batches once ➡ **Shuffle** after each epoch

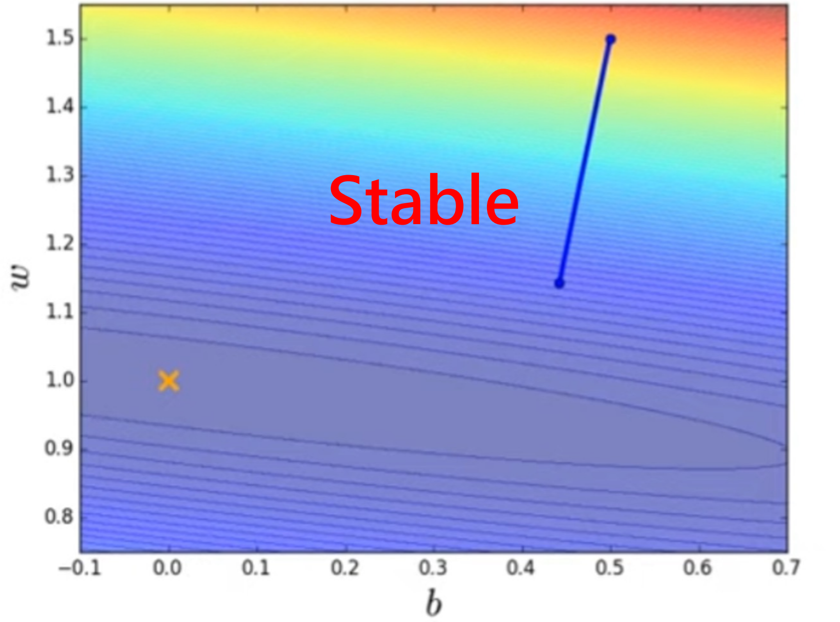
Large batch v.s. small batch

Consider 20 example(N=20)

Batch size = N(Full batch)

Update after seeing

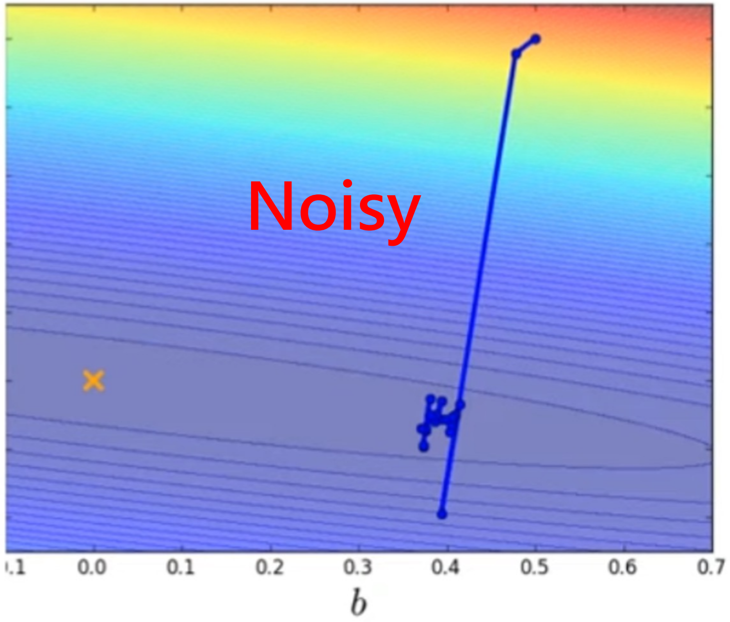
all 20 examples



Batch size = 1

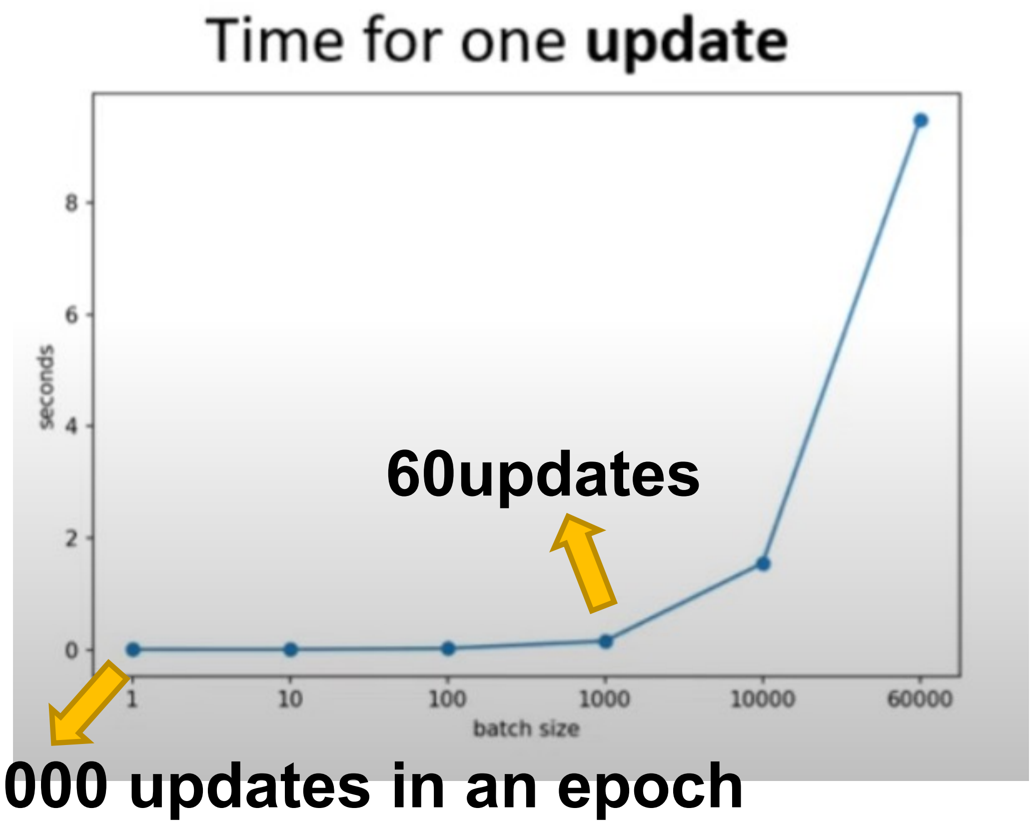
Update for each example

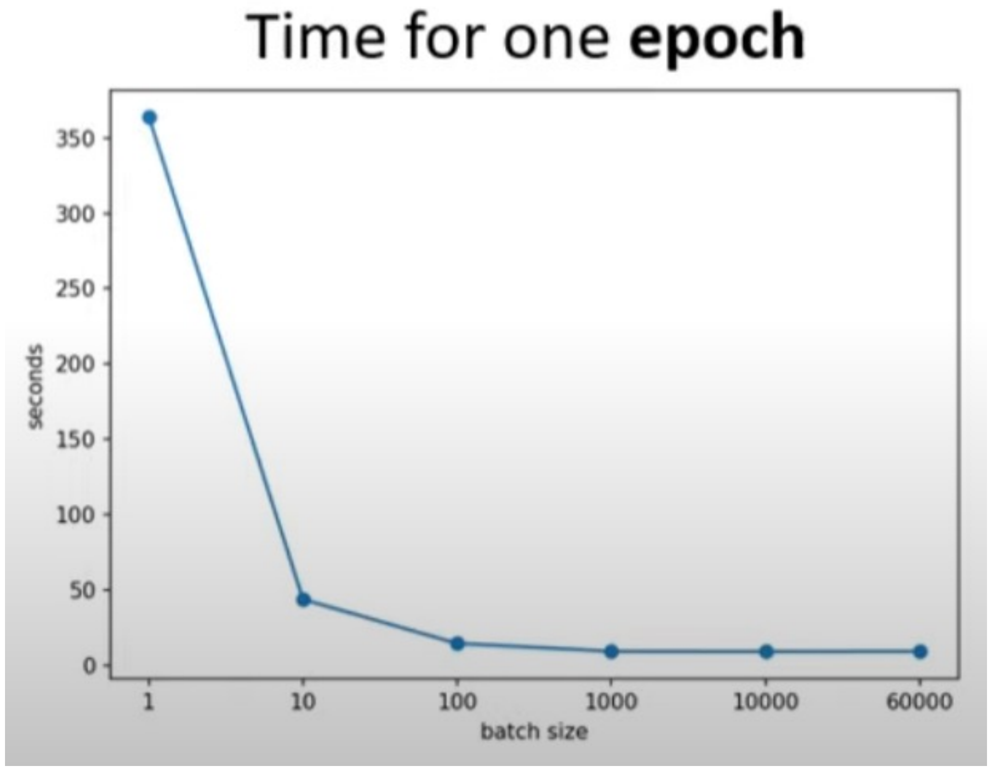
Update 20 times in an epoch



Large batch v.s. small batch

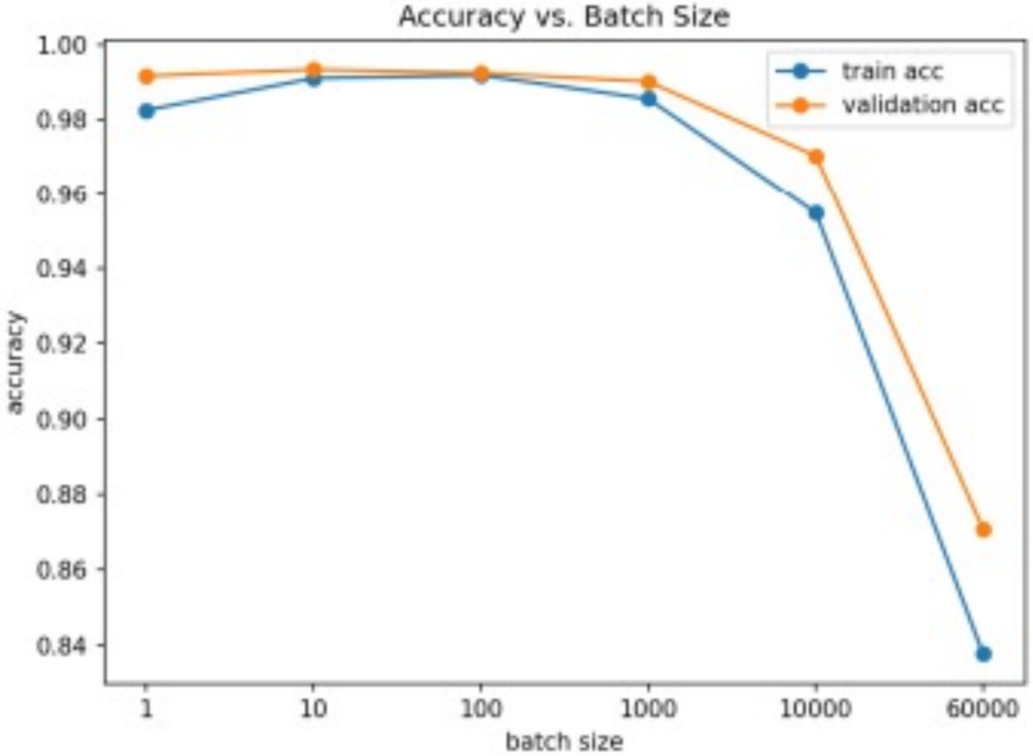
- Larger batch size doesn’t require longer time to compute gradient(update)
- Smaller batch requires longer time for one epoch(考慮平行計算)



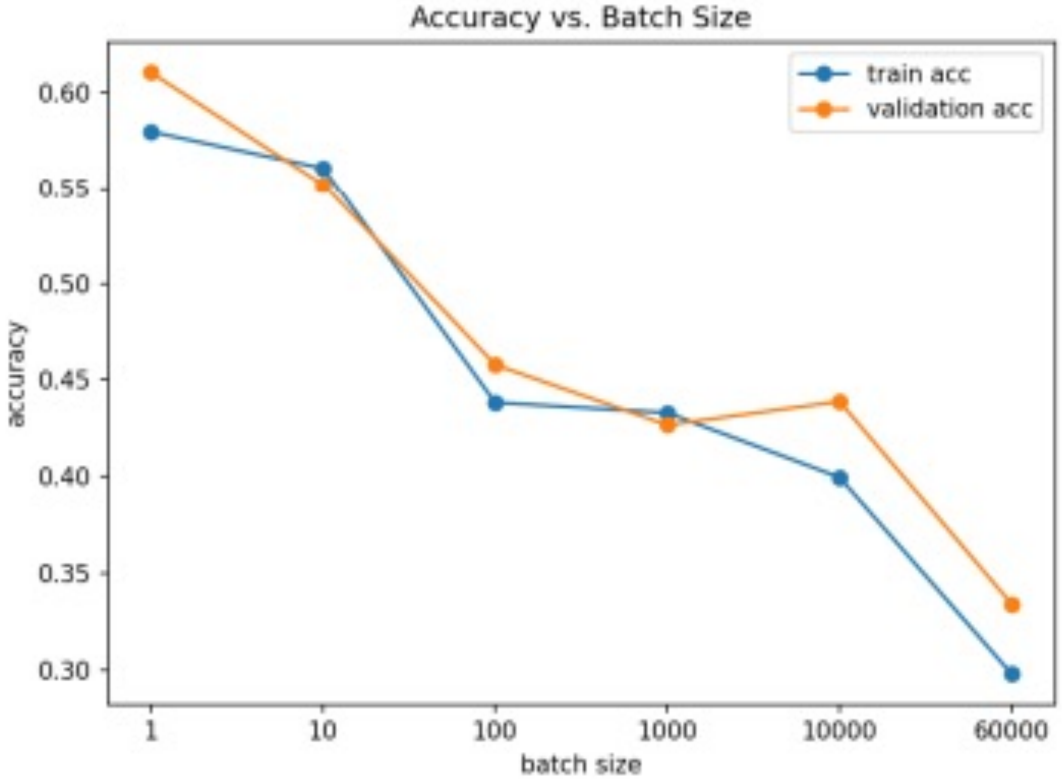


Large batch v.s. small batch

MNIST



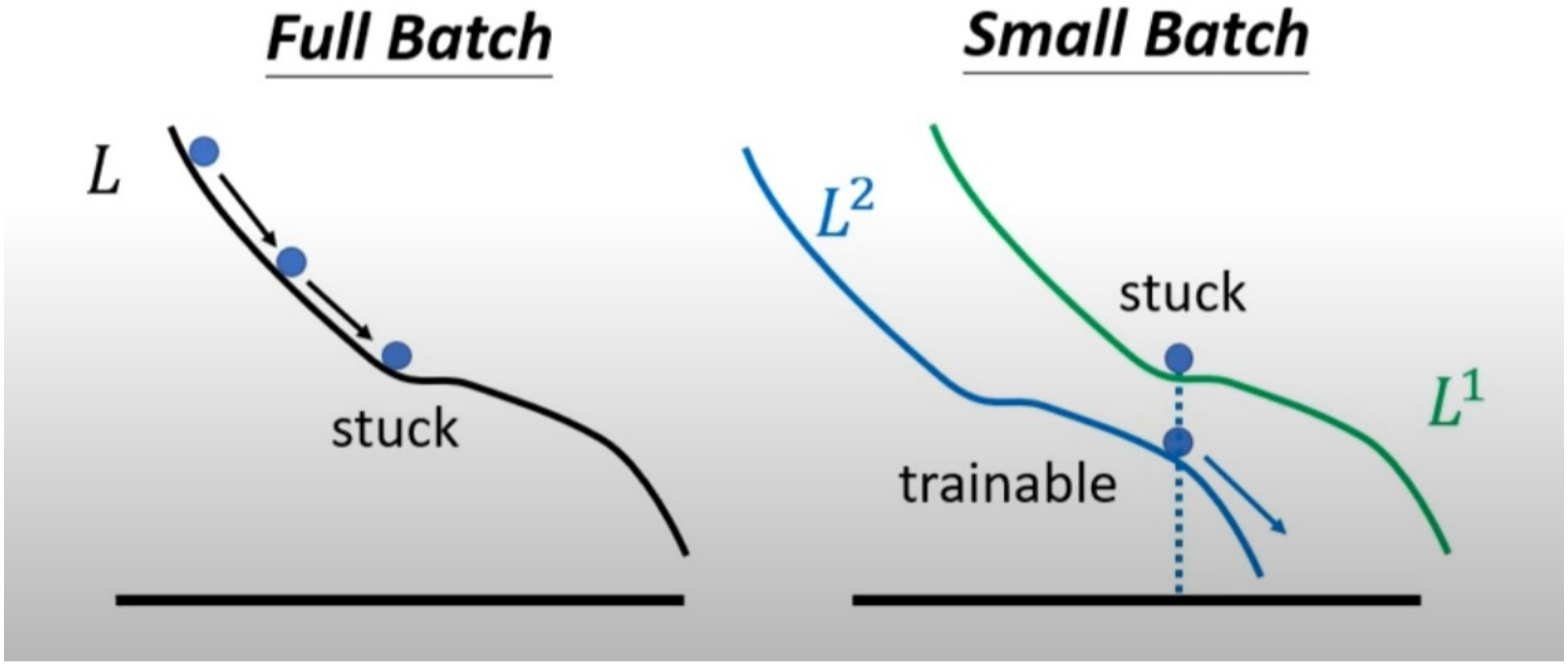
CIFAR-10



- Smaller batch size has better performance

Large batch v.s. small batch

- Smaller batch size has better performance
- “Noisy” update is better for training



Batch size

	Small	Large
Speed for one update (no parallel)	Faster	Slower
Speed for one update (with parallel)	Same	Same (not too large)
Time for one epoch	Slower	Faster WIN
Gradient	Noisy	Stable
Optimization	Better WIN	Worse
Generalization	Better WIN	Worse

Batch size is a hyperparameter you have to decide

參考: https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/small-gradient-v7.pdf

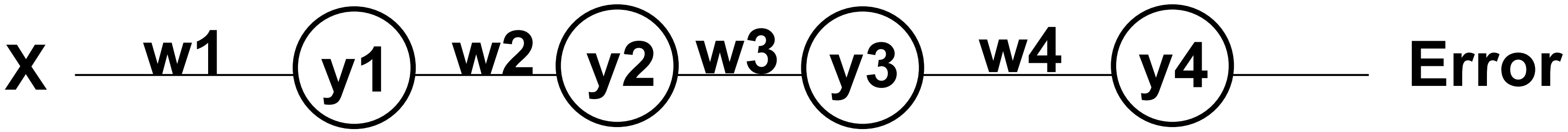
Outline

1. Lab3 task
2. ResNet18
3. CIFAR-10

ResNet

- 為了取得更多或更深層的特徵，我們會採用越來越多層的CNN模型作為解決方法，五層結果可能會比一層的結果好，七層可能比五層結果好，那麼為了提高模型的表現，我們可以無限一直往上加層數嗎?
- 當我們層數一直往上疊加時，會發現反而越深層的網路表現會比較差，這可能發生了梯度消失的問題。

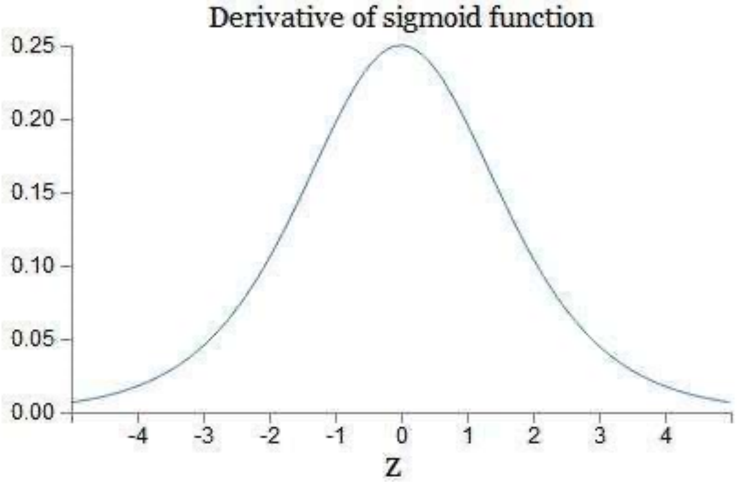
Vanishing gradient problem



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial y_4}\frac{\partial y_4}{\partial z_4}\frac{\partial z_4}{\partial x_4}\frac{\partial x_4}{\partial z_3}\frac{\partial z_3}{\partial x_3}\frac{\partial x_3}{\partial z_2}\frac{\partial z_2}{\partial x_2}\frac{\partial x_2}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

$$= \frac{\partial E_{total}}{\partial y_4}\sigma'(z_4)\,w_4\,\sigma'(z_3)\,w_3\,\sigma'(z_2)\,w_2\,\sigma'(z_1)\,x_1$$

因為我們初始化的權重通常是在0附近的小數，
w2*w3*w4會很小，導致w1的梯度消失



若我們activation function是使用sigmoid，
因為simoid導數的閾值是(0,0.25)，導致神經網絡在反向傳播的時候，
其梯度越來越小，最後甚至根本無法訓練

使用Batch Normalization或改用ReLU可以解決

Degradation

- 但當深度逐漸增加，我們發現56層的神經網路反而比20層網路結果還差。這樣的結果並非來自於 overfitting和Vanishing gradient problem，而是因為深度增加連帶著使得 training error 增加所導致的退化問題，以至於深層的特徵丟失了淺層特徵的原始模樣

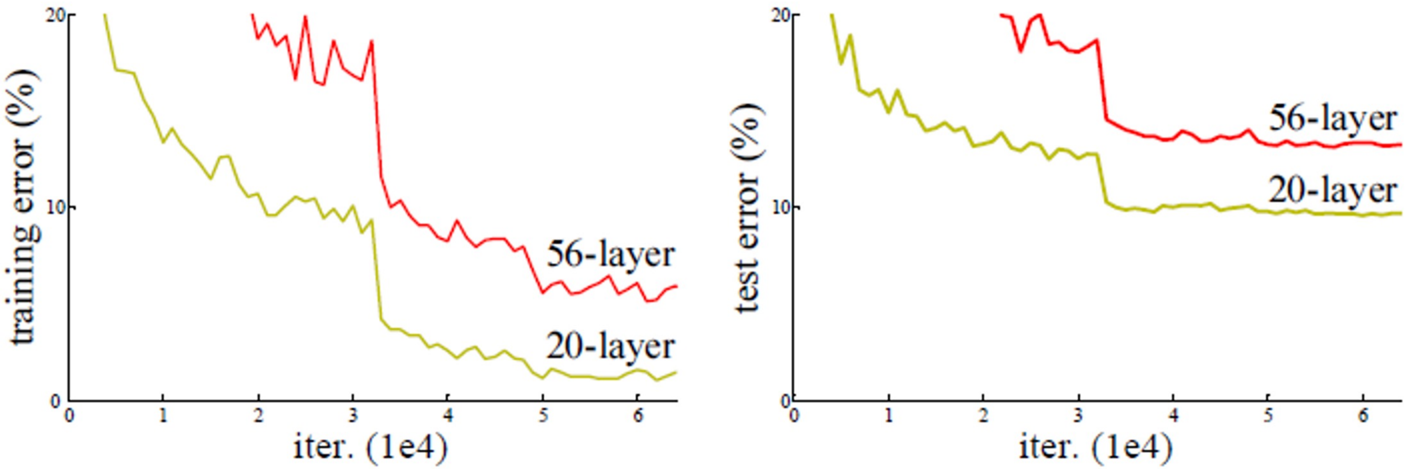
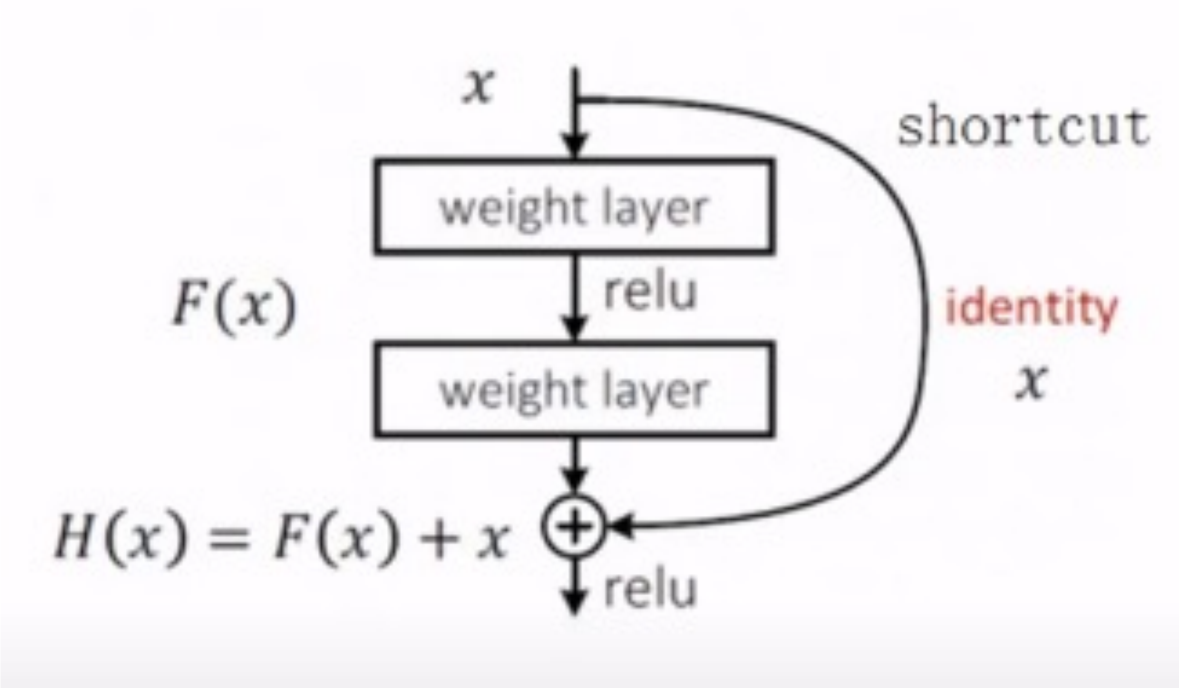


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

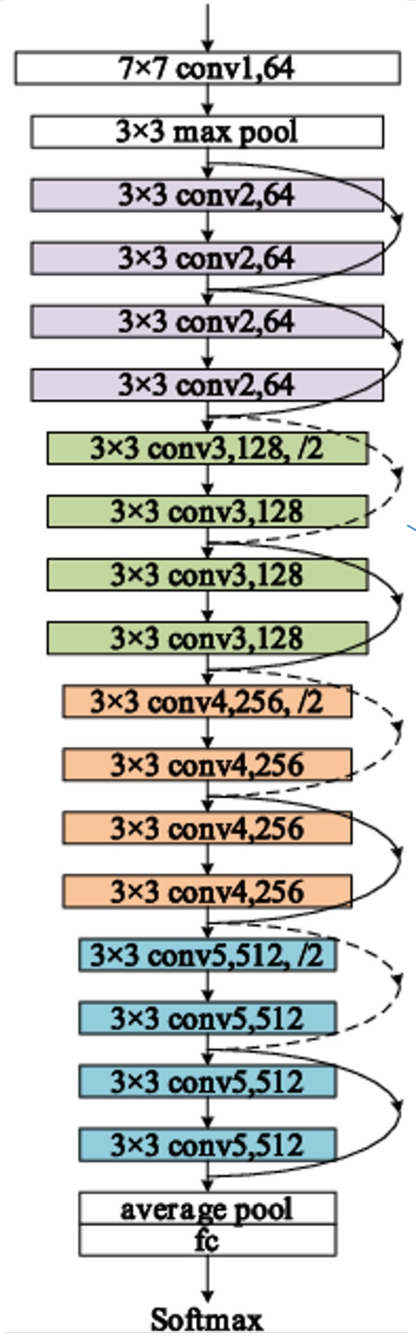
ResNet-18

- 用Deep residual Network來處理degradation，這樣做能在網路層加深後，正確率至少不會變的更差。



輸入是x
學到的特徵是H(x)
Residual = H(x) – x
F(x) = H(x) – x
原本學習是這樣: x → H(x)
已經知道 F(x) = H(x) – x
所以學習也可以這樣寫：x → F(x) + x
輸入→特徵
變成：輸入→ 輸入 + 殘差

ResNet18



實線表示維度相同
計算方式為H(x)=F(x)+x

虛線表示維度不同
計算方式為H(x)=F(x)+Wx
其中W是1*1的卷積，調整x的維度

[ResNet paper]

ResNet18-implement

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

[ResNet paper]

ResNet18-implement

```python
    # make layers
    #self.layer1 = ...
    #self.layer2 = ...
    #self.layer3 = ...
    #self.layer4 = ...
    #self.fc = ...

#This function is primarily used to repeat the same residual block
def make_layer(self, block, channels, num_blocks, stride):
```

[ResNet paper](#)

Outline

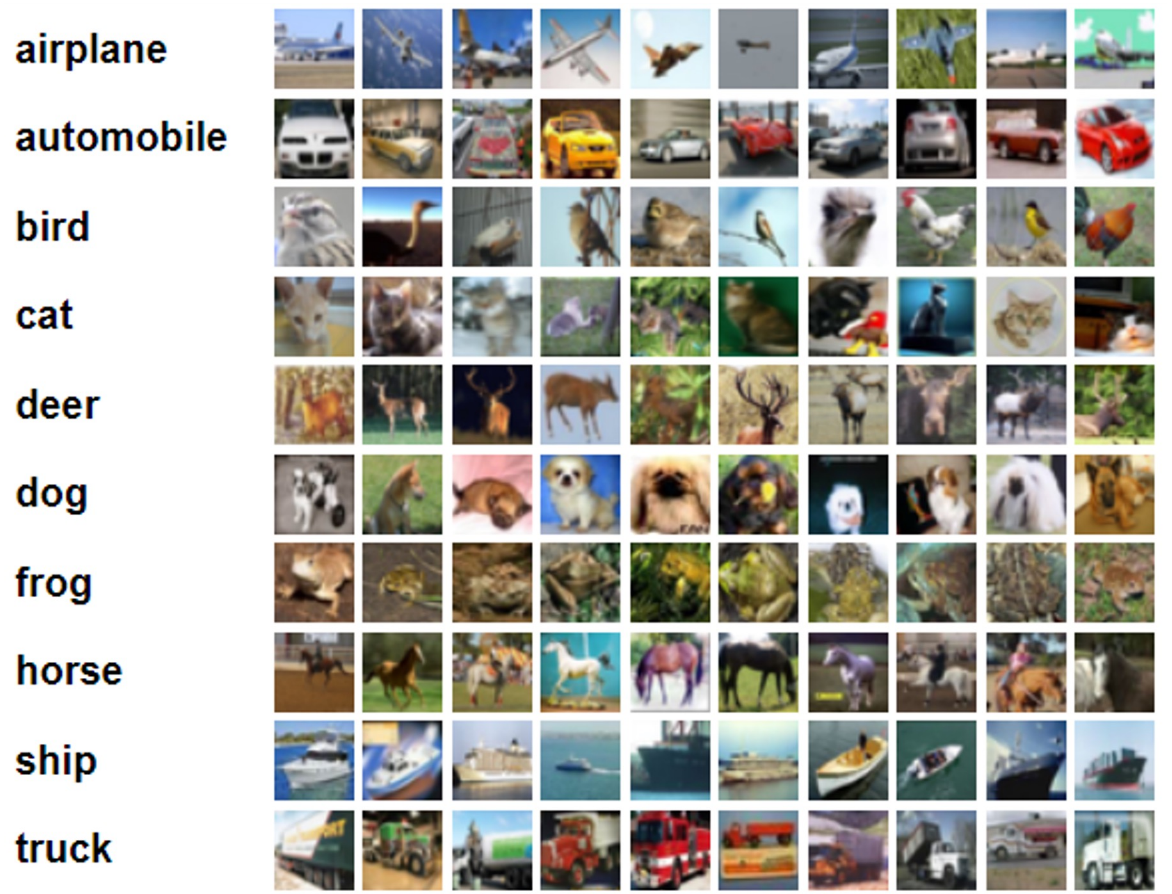
1. Lab3 task

1. ResNet18

1. CIFAR-10

CIFAR-10

- CIFAR-10 consists of 32x32 colour images in 10 classes
- 50000 training images + 10000 test images



Upload to moodle

- 學號_lab3.zip
  - 學號_lab3.ipynb
    - IPython notenook 須包含程式碼跟結果
  - resnet18.py
    - 上傳完成的Resnet18
  - 學號_lab3.pdf
    - 說明不同tuning方式的原理及如何實作
    - 說明並比較不同tuning方式如何造成影響
    - 截圖並說明各項結果(包含training/val的accuracy和loss曲線圖 & test accuracy和loss結果)
    - 如何搭建Resnet18
    - 實作過程中遇到的困難及你後來是如何解決的

END

Advisor：Tsai, Chia-Chi