



# 遷移學習 (Transfer Learning)

李宏毅&教研處

# 「版權聲明頁」

**本投影片已經獲得作者授權台灣人工智慧學校得以使用於教學用途，如需取得重製權以及公開傳輸權需要透過台灣人工智慧學校取得著作人同意；如果需要修改本投影片著作，則需要取得改作權；另外，如果有需要以光碟或紙本等實體的方式傳播，則需要取得人工智慧學校散佈權。**

## 課程內容

---

- 理論講授 [投影片下載](#) (PDF)
- 理論講授 [影片播放列表](#) (YouTube)
- 今日課程 [投影片下載](#) (PDF)

## 主要課程

1. Model Fine-tuning
2. 遷移學習實作 in TensorFlow

## 延伸閱讀(Optional)

1. Multitask Learning
2. Domain-adversarial training
3. Zero-shot Learning
4. Self-taught learning

# 前言

---

遷移學習是一個相當大的主題，學員在課程及日後較有可能會接觸到的是 Model Fine-tuning，故今日的課程以 Model Fine-tuning 為主。其他遷移學習的主題，學員可視時間許可或興趣，另行延伸閱讀。

- 教研處

# 建議進度

---

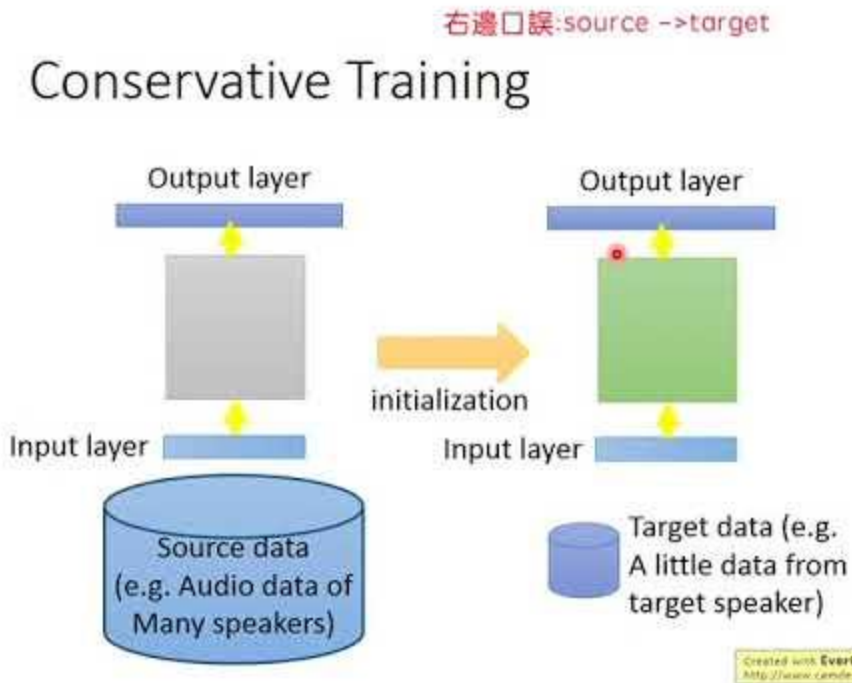
## 上午

09:30 - 10:30	理論講授 - Model Fine-tuning
10:30 - 12:00	實作 - 前置作業、Load Data

## 下午

14:00 - 15:00	實作 - 建立靜態圖 (graph)
15:00 - 16:00	實作 - 載入模型參數
	實作 - 實際執行模型訓練
	實作 - 模型測試
16:00 -	實作 - 練習

# 理論講授 - Model Fine-tuning



# 程式實作 - 遷移學習 in TensorFlow

---

- [影片播放列表](#)

# 實作 - 簡介

## 任務的相似性

- 相似的任務通常具有相似的特徵。如影像辨識在最底層的特徵都是由線條等圖案組成
- 以影像辨識為例，如果借用別人訓練好的模型，就不用從低階特徵開始學習。加速模型學習、提昇表現





# 經過 DNN / CNN 實作課後...

---

## 相信你已學會

- Tensorflow 基本語法及概念
- 用 Tensorflow 搭建深度學習模型

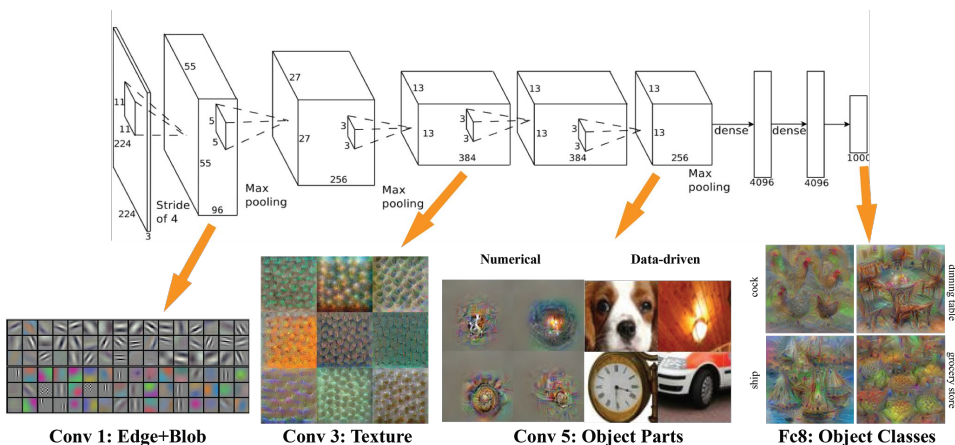
## 但實際上路後...

- 每次都從無到有搭建、訓練模型，太耗時耗神了！
- 我的訓練資料太少怎麼辦？
- 怎麼提升模型的表現？

遷移學習是你的好夥伴！！！！

# 任務的相似性

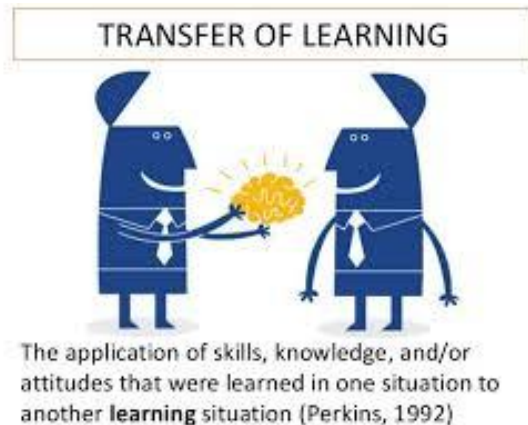
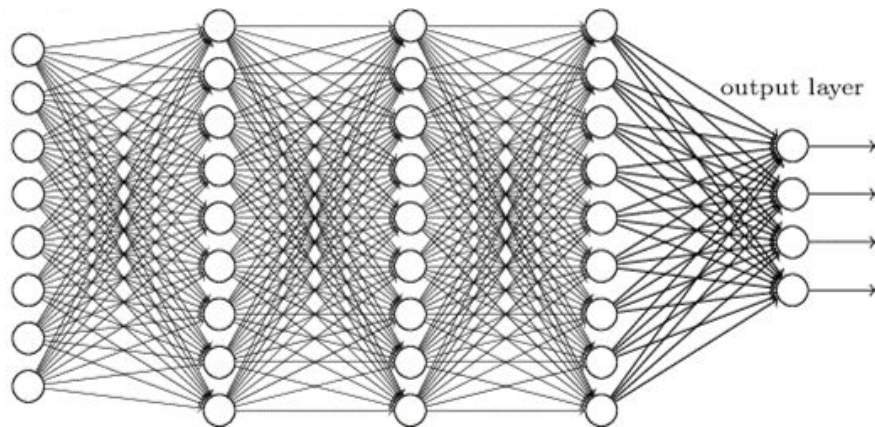
- 相似的任務通常具有相似的特徵。如影像辨識在最底層的特徵都是由線條等圖案組成
- 以影像辨識為例，如果借用別人訓練好的模型，就不用從低階特徵開始學習。加速模型學習、提昇表現



# 遷移學習

借用經驗

- 深度學習模型學習到的特徵都藏在參數組合裡面
- 把別人訓練好模型 (pre-trained model) 的參數複製過來，當作是我們模型的參數



# 實務常用的遷移學習

---

## 1. Fine-tune

複製模型的參數當作初始參數，再用自己的訓練資料進行微調  
(借用的參數會被改變)

## 1. Layer transform

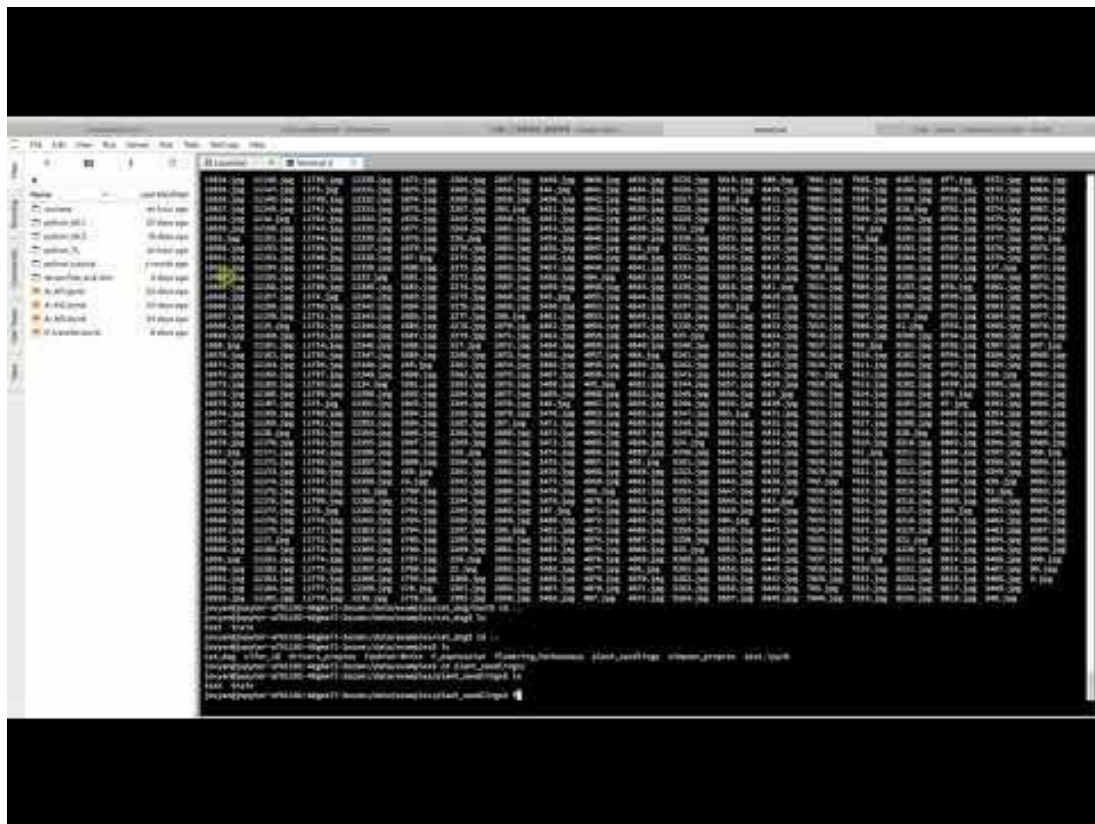
只借用模型的部分神經層參數，並在訓練的過程中凍結該層參數，  
不更新

## 小提醒

借用別人的 pre-trained model 時，記得模型架構，要跟對方的一樣喔！

接下來我們就用一個樣板程式碼，  
帶大家了解如何進行 model fine-tuning 吧

# 實作 - 手把手遷移學習





# 手把手遷移學習

帶大家認識從資料前處理到 pre-trained model 加載參數的流程。

# 手把手後你將學到...

---

- 訓練、測試模型的完整過程: 影像前處理、載入資料、建立 Tensorflow 靜態圖到模型訓練、測試
- 使用 TF Slim 的模型, 不用自己慢慢搭建
- 如何載入 pre-trained model 的參數
- Tensorflow 資源管理機制: `tf.GraphKeys`



# 樣板程式碼

---

## 簡介

- 任務目標: 訓練模型做影像二元分類
- 訓練資料: [Dogs vs. Cats \(kaggle\)](#)
- 遷移模型: 使用 [ImageNet](#) 預訓練的 [ResNet\\_v2\\_50](#) 模型 (對 ImageNet 和 Restnet 不熟悉的話, 可以參考連結)

## 打開程式碼

- 在 hub 中新增 terminal
- 輸入“ `cp -r courses/python_TL ~/` ”複製課程資料夾
- 打開 `tf_transfer_learning.ipynb`

# 樣板程式碼流程

## 前置作業

- 生成資料檔案路徑清單
- 載入 packages
- 定義影像前處理
- 定義 model dict & callbacks

## Load data

- 載入 train, val, test.csv
- data generator

## Tensorflow- 建立靜態圖 (graph)

- graph, session
- optimizer
- placeholder
- resnet\_v2\_50 model
- loss, update

## Tensorflow- 載入模型參數

- tf.train.saver
- var\_list
- saver.restore

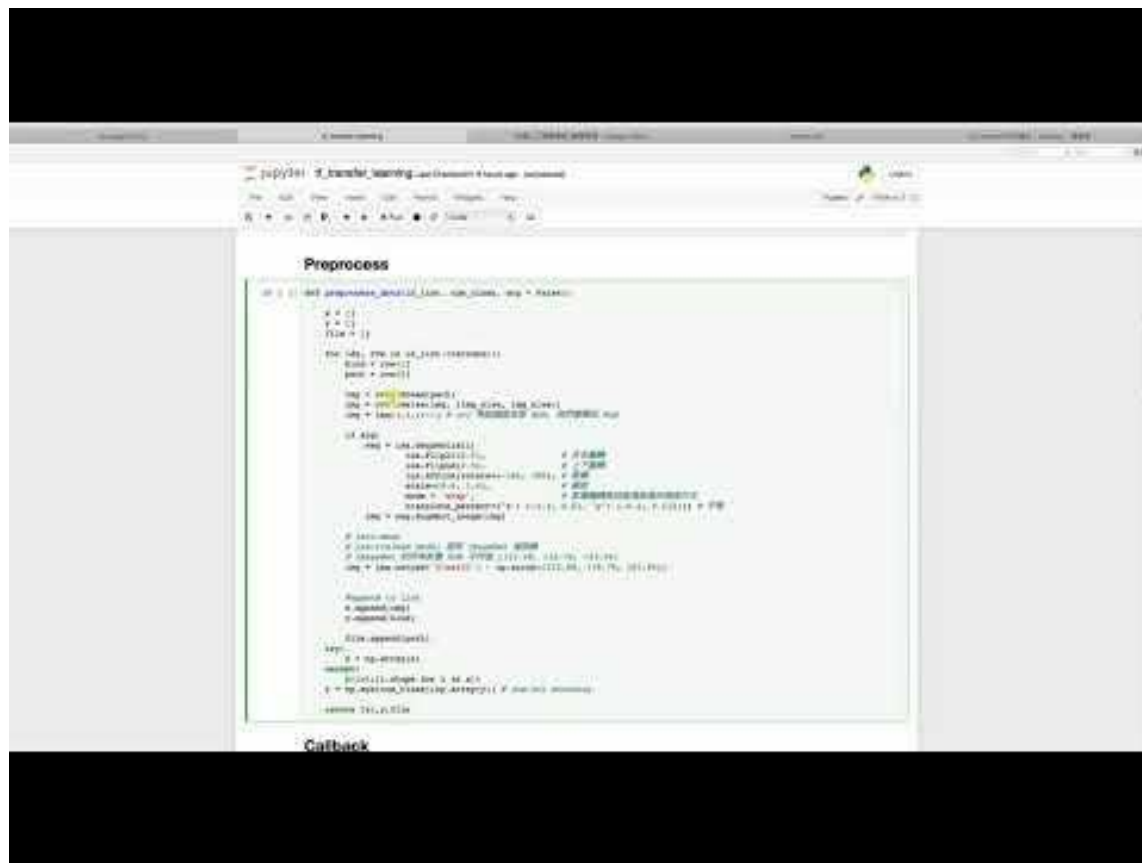
## Tensorflow- 實際執行模型訓練

- 執行 loss, update 操作 (train)
- 執行 loss 操作 (val)

## Tensorflow- 模型測試

- saver.restore
- 執行 pred\_softmax 操作
- accuracy

# 實作 - 前置作業、Load Data





# 前置作業

# 樣板程式碼流程

## 前置作業

- 生成資料檔案路徑清單
- 載入 packages
- 定義影像前處理
- 定義 model dict & callbacks

## Load data

- 載入 train, val, test.csv
- data generator

## Tensorflow- 建立靜態圖 (graph)

- graph, session
- optimizer
- placeholder
- resnet\_v2\_50 model
- loss, update

## Tensorflow- 載入模型參數

- tf.train.saver
- var\_list
- saver.restore

## Tensorflow- 實際執行模型訓練

- 執行 loss, update 操作 (train)
- 執行 loss 操作 (val)

## Tensorflow- 模型測試

- saver.restore
- 執行 pred\_softmax 操作
- accuracy

# 前置作業

## 生成資料檔案路徑清單

- 將資料切成 train、val、test
- 用 csv 檔紀錄檔案位置和對應類別 (狗=1, 貓=0)
- 已經帮大家生好資料清單了, 在 data\_list/cat\_dog/k\_fold/



	檔案路徑	類別
1	data/cat_dog/test/dog.11275.jpg	1
2	data/cat_dog/test/dog.4161.jpg	1
3	data/cat_dog/test/dog.5938.jpg	1
4	data/cat_dog/test/dog.9130.jpg	1
5	data/cat_dog/test/dog.11165.jpg	1
6	data/cat_dog/test/dog.8642.jpg	1
7	data/cat_dog/test/dog.7348.jpg	1
8	data/cat_dog/test/dog.2806.jpg	1
9	data/cat_dog/test/dog.5304.jpg	1
10	data/cat_dog/test/dog.1574.jpg	1
11	data/cat_dog/test/dog.4250.jpg	1
12	data/cat_dog/test/dog.6090.jpg	1
13	data/cat_dog/test/dog.8287.jpg	1
14	data/cat_dog/test/dog.6706.jpg	1

# 前置作業

## 載入 packages

---

```
'''Basic package'''
import os
# 告訴系統要第幾張卡被看到。 Ex. 硬體總共有8張顯卡，以下設定只讓系統看到第1張顯卡
# 若沒設定，則 Tensorflow 在運行時，預設會把所有卡都佔用
# 要看裝置內顯卡數量及目前狀態的話，請在終端機內輸入 "nvidia-smi"
# 若你的裝置只有一張顯卡可以使用，可以忽略此設定
os.environ["CUDA_VISIBLE_DEVICES"] = "0"

import queue
import cv2          #影像處理
import scipy.misc   #影像處理
import numpy as np
import pandas as pd
from tqdm import tqdm_notebook as tqdm #進度條
import matplotlib.pyplot as plt #繪圖

# 自定義 library
from generator import data_generators
from callbacks import *
```

# 前置作業

載入 packages

---

```
'''Tensorflow package'''
import tensorflow as tf
import tensorflow.contrib.slim as slim
import tensorflow.contrib.slim.nets as slimNet

'''Data augmentation package'''
from imgaug import augmenters as iaa
import imgaug as ia
```

## TF Slim

- 高階的函式庫
- 定義一些常用的神經網路層、經典深度學習模型、損失函數、評估指標等...
- 快速搭建模型
- 讓你的程式碼變得優雅簡潔
- [更詳細的說明](#)
- [官方 github](#)



# 前置作業

影像前處理技巧: 數據增強

## 為什麼要做數據增強？

- 真實世界的 testing data 不一定長得跟訓練資料一樣
- 避免模型 overfitting
- 轉換我們手上的 training data 來增加多樣性, 以應付未來各種可能會出現的樣子

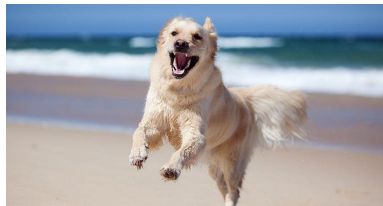
## 常用數據增強方法

- 翻轉 (flip)
- 縮放 (zoom)
- 平移 (shift)

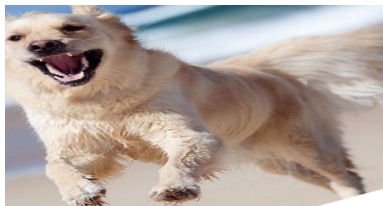
## 參考資料

- [深度學習中的 Data Augmentation 方法及代碼實現](#)
- [The Effectiveness of Data Augmentation in Image Classification using Deep Learning](#)

訓練模型的影像 ...



實際測試的時候卻碰到 ...



# 前置作業

## 影像前處理

```
def preprocess_data(id_list, num_class, aug = False):
```

```
    x = []
    y = []
    file = []
```

```
    for idx, row in id_list.iterrows():
        kind = row[1]
        path = row[0]
```

```
        img = cv2.imread(path)
        img = cv2.resize(img, (img_size, img_size))
        img = img[:, :, :-1]
```

匯入影像, 並縮放到指定大小  
可改用其他如 PIL 匯入影像

```
        if aug:
            seq = iaa.Sequential([
                iaa.Fliplr(0.5),          # 左右翻轉
                iaa.Flipud(0.5),         # 上下翻轉
                iaa.Affine(rotate=(-180, 180), # 旋轉
                           scale=(0.6, 1.4), # 縮放
                           mode = 'wrap',    # 影像翻轉造成區塊缺值的補值方式
                           translate_percent={ "x": (-0.2, 0.2), "y": (-0.2, 0.2)})) # 平移
            img = seq.augment_image(img)
```

數據增強, 增加訓練數據的  
複雜度

```
        # zero-mean
        # pre-trained model 使用 ImageNet 做訓練
        # ImageNet 的所有影像 RGB 平均值 [123.68, 116.78, 103.94]
        img = img.astype('float32') - np.array([123.68, 116.78, 103.94])
```

[Why do we normalize images by subtracting the dataset's image mean and not the current image mean in deep learning?](#)

```
        #append to list
        x.append(img)
        y.append(kind)
```

```
        file.append(path)
```

```
    try:
        x = np.array(x)
    except:
        print([i.shape for i in x])
    y = np.eye(num_class)[np.array(y)] # one-hot encoding
```

```
    return [x], y, file
```

# 前置作業

## model dict

- 將 model 有關的資訊和操作，用字典 (dictionary) 包起來，集中管理
- 以存取 train\_loss 為例，只要呼叫 model\_dict['history']['train\_loss'] 就可以儲存或查看 train loss

```
model_dict = {  
    'model_name' : model_name,  
    'reduce_lr' : ReduceLROnPlateau(lr=1.7e-4, factor=0.5, patience=3),  
    'earlystop' : EarlyStopping(min_delta = 1e-4, patience= 10),  
    'checkpoint' : Model_checkpoint(os.path.join('model', model_name)),  
    'train_batch_log' : History(['loss']),  
    'val_batch_log' : History(['loss']),  
    'history' : {  
        'train_loss':[],  
        'val_loss':[]  
    },  
    'testing' : {  
        'y_true' : [],  
        'y_pred' : [],  
        'files' : []  
    }  
}
```

定義在 callbacks.py  
裡面

# 前置作業

## callbacks

---

- 大家在 "實戰演練 DNN & CNN" 課程中遇過, 應該不陌生

```
callback_dict = {
    'on_session_begin':[], # start of a session
    'on_batch_begin':[], # start of a training batch
    'on_batch_end':[], # end of a training batch
    'on_epoch_begin':[], # start of a epoch
    'on_epoch_end':[
        model_dict['reduce_lr'],
        model_dict['earlystop'],
        model_dict['checkpoint']
    ], # end of a epoch
    'on_session_end':[] # end of a session
}
callback_manager = Run_collected_functions(callback_dict)
```



Load data

# 樣板程式碼流程

## 前置作業

- 生成資料檔案路徑清單
- 載入 packages
- 定義影像前處理
- 定義 model dict & callbacks

## Load data

- 載入 train, val, test.csv
- data generator

## Tensorflow- 建立靜態圖 (graph)

- graph, session
- optimizer
- placeholder
- resnet\_v2\_50 model
- loss, update

## Tensorflow- 載入模型參數

- tf.train.saver
- var\_list
- saver.restore

## Tensorflow- 實際執行模型訓練

- 執行 loss, update 操作 (train)
- 執行 loss 操作 (val)

## Tensorflow- 模型測試

- saver.restore
- 執行 pred\_softmax 操作
- accuracy

# Load data

## load data list

---

```
#load train/test set
test_dog = pd.read_csv(os.path.join("data_list/cat_dog/k_fold", "test_dog_" + fold + ".csv"))
test_cat = pd.read_csv(os.path.join("data_list/cat_dog/k_fold", "test_cat_" + fold + ".csv"))

train_dog = pd.read_csv(os.path.join("data_list/cat_dog/k_fold", "train_dog_" + fold + ".csv"))
train_cat = pd.read_csv(os.path.join("data_list/cat_dog/k_fold", "train_cat_" + fold + ".csv"))

val_dog = pd.read_csv(os.path.join("data_list/cat_dog/k_fold", "val_dog_" + fold + ".csv"))
val_cat = pd.read_csv(os.path.join("data_list/cat_dog/k_fold", "val_cat_" + fold + ".csv"))
```

- 載入 data list, 待會要放入 generator, 讓 generator 去相對應的路徑讀檔案進來
- cat\_dog data set 被切成五份 (5 fold), 這裡只用第一份做訓練 (fold = 0)

# Load data

## data generator

```
generators = data_generators(batch_size, [[train_dog, val_dog, test_dog], [train_cat, val_cat, test_cat]], 2, preprocess_data)
```

```
generators.load_val() # validation data 不多且每個 epoch 的資料固定，可以預先載入記憶體
```

```
generators.start_train_threads() # 開啟訓練集的執行緒（支援多執行緒，但為了不造成伺服器負擔，預設只開一個執行緒）
```

- data\_generators 使用多執行緒的方式，在前一個 batch data 讀進來訓練的同時，讀下一個 batch data
- 減少檔案讀取時間，加快模型訓練速度
- 定義在 generator.py
- 為了降低伺服器負擔，預設只開一個執行緒





# 實作 - 建立靜態圖

台灣人工智慧學院\_課程學習

File Edit View Insert Format Style Arrange Tools Help All contents except in Chinese

## Tensorflow- 建立靜態圖

### resnet v2\_50 model

使用和 pre-trained 一樣的模型架構

- pre-trained model: TF Slim- resnet\_v2\_50 訓練後的參數記錄
- 我們要建立和 pre-trained 一樣的容器 (模型架構), 才能順利把參數記錄還原回來


pre-trained model 訓練好之後, 參數命名和對應的數值, 都被記錄在這四個檔案當中

- ☐ checkpoint
- ☐ pretrained\_20160828-000000.ckpt.data-000000
- ☐ pretrained\_20160828-000000.ckpt.index
- ☐ pretrained\_20160828-000000.ckpt.meta

The diagram illustrates the ResNet v2\_50 architecture. It starts with an input layer, followed by four blocks of residual units. Block 1 has 2 units, Block 2 has 3 units, Block 3 has 5 units, and Block 4 has 3 units. Each block is followed by a residual connection. The final output is a fully connected layer. The diagram is labeled with 'Block 1', 'Block 2', 'Block 3', 'Block 4', and 'Fully connect'.

28

Click to add speaker notes



Tensorflow- 建立靜態圖

# 樣板程式碼流程

## 前置作業

- 生成資料檔案路徑清單
- 載入 packages
- 定義影像前處理
- 定義 model dict & callbacks

## Load data

- 載入 train, val, test.csv
- data generator

## Tensorflow- 建立靜態圖 (graph)

- graph, session
- optimizer
- placeholder
- resnet\_v2\_50 model
- loss, update

## Tensorflow- 載入模型參數

- tf.train.saver
- var\_list
- saver.restore

## Tensorflow- 實際執行模型訓練

- 執行 loss, update 操作 (train)
- 執行 loss 操作 (val)

## Tensorflow- 模型測試

- saver.restore
- 執行 pred\_softmax 操作
- accuracy

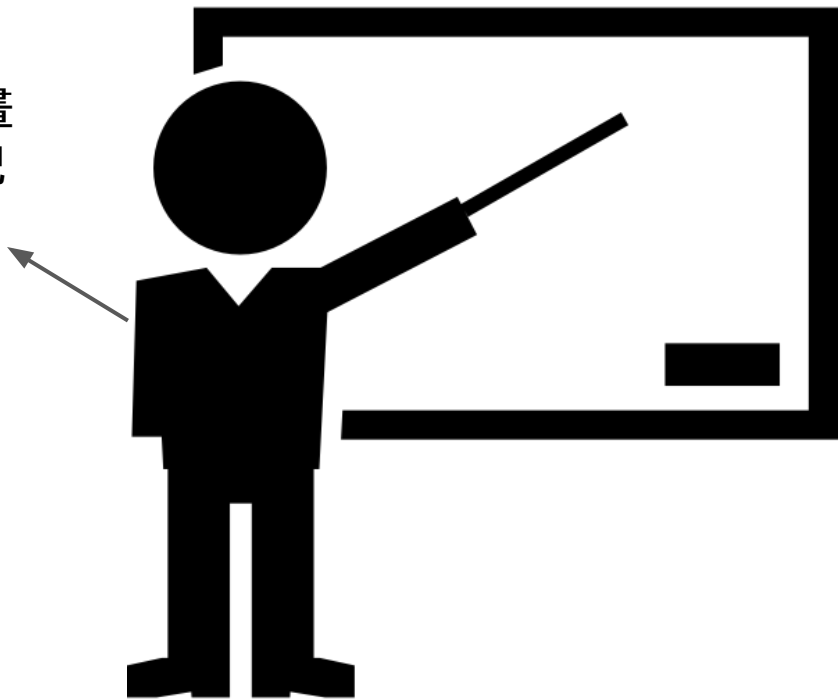
# Tensorflow- 建立靜態圖

## Graph & Session

---

- 大家在 "實戰演練 DNN & CNN" 課程中遇過, 應該不陌生

Session 就像計畫  
執行者, 實際分配  
資源、執行計算



Graph 就像計畫圖  
一樣, 定義計算的  
流程

# Tensorflow- 建立靜態圖

## Optimizer

---

你可以選擇...

- SGD
- Adagrad
- RMSprop
- Adam
- 深度學習優化方法比較

```
#### optimizer ####  
lr = tf.placeholder(tf.float32, shape=[])  
optimizer = tf.train.AdamOptimizer(lr)
```

Learning rate 會在訓練的過程中不斷改變，所以用 placeholder 建立一個 lr 變數

# Tensorflow- 建立靜態圖

## Placeholder

---

- 定義執行階段的傳入值, 同時指定型態 (dtype) 跟大小 (shape)

```
#### placeholder ####
input_img = tf.placeholder(dtype=tf.float32, shape=(None, img_size, img_size, 3))
y_true = tf.placeholder(dtype=tf.float32, shape=(None, num_class))
is_training = tf.placeholder(dtype=tf.bool, shape=[])
```

Diagram illustrating the shape parameters in the TensorFlow placeholder code:

- batch size**: Points to the `None` value in the first dimension of `input_img`'s shape.
- image height**: Points to the `img_size` value in the second dimension of `input_img`'s shape.
- image width**: Points to the `img_size` value in the third dimension of `input_img`'s shape.
- image channels**: Points to the `3` value in the fourth dimension of `input_img`'s shape.

指定 shape 中某個維度為 `None`, 代表那個維度會依實際輸入自己決定  
`None` 通常放在第一個維度表示 batch size 是可變的

# Tensorflow- 建立靜態圖

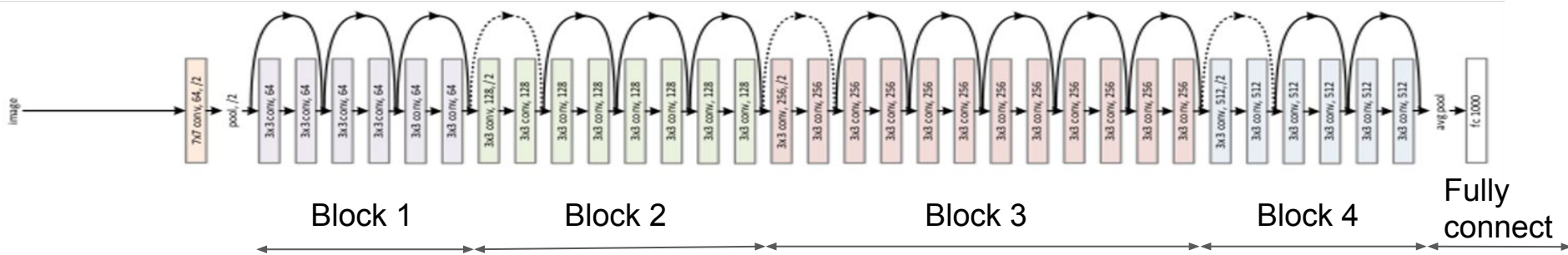
resnet\_v2\_50 model

## 使用和 pre-trained 一樣模型架構

- pre-trained model: TF Slim- resnet\_v2\_50 訓練後的參數紀錄
- 我們要建立和 pre-trained 一樣的容器 (模型架構), 才能順利把參數紀錄還原回來

*pre-trained model* 訓練好之後, 參數命名和對應的數值, 都被記錄在這四個檔案當中

<input type="checkbox"/>	..
<input type="checkbox"/>	checkpoint
<input type="checkbox"/>	pretrain_0.ckpt.data-00000-of-00001
<input type="checkbox"/>	pretrain_0.ckpt.index
<input type="checkbox"/>	pretrain_0.ckpt.meta



# Tensorflow- 建立靜態圖

resnet\_v2\_50 model

Google 幫大家用 ImageNet 訓練好一些模型摟

Model	TF-Slim File	Checkpoint	Top-1 Accuracy	Top-5 Accuracy
Inception V1	Code	<a href="#">inception_v1_2016_08_28.tar.gz</a>	69.8	89.6
Inception V2	Code	<a href="#">inception_v2_2016_08_28.tar.gz</a>	73.9	91.8
Inception V3	Code	<a href="#">inception_v3_2016_08_28.tar.gz</a>	78.0	93.9
Inception V4	Code	<a href="#">inception_v4_2016_09_09.tar.gz</a>	80.2	95.2
Inception-ResNet-v2	Code	<a href="#">inception_resnet_v2_2016_08_30.tar.gz</a>	80.4	95.3
ResNet V1 50	Code	<a href="#">resnet_v1_50_2016_08_28.tar.gz</a>	75.2	92.2
ResNet V1 101	Code	<a href="#">resnet_v1_101_2016_08_28.tar.gz</a>	76.4	92.9
ResNet V1 152	Code	<a href="#">resnet_v1_152_2016_08_28.tar.gz</a>	76.8	93.2
ResNet V2 50^	Code	<a href="#">resnet_v2_50_2017_04_14.tar.gz</a>	75.6	92.8
ResNet V2 101^	Code	<a href="#">resnet_v2_101_2017_04_14.tar.gz</a>	77.0	93.7
ResNet V2 152^	Code	<a href="#">resnet_v2_152_2017_04_14.tar.gz</a>	77.8	94.1
ResNet V2 200	Code	TBA	79.9*	95.2*
VGG 16	Code	<a href="#">vgg_16_2016_08_28.tar.gz</a>	71.5	89.8
VGG 19	Code	<a href="#">vgg_19_2016_08_28.tar.gz</a>	71.1	89.8
MobileNet_v1_1.0_224	Code	<a href="#">mobilenet_v1_1.0_224_2017_06_14.tar.gz</a>	70.7	89.5
MobileNet_v1_0.50_160	Code	<a href="#">mobilenet_v1_0.50_160_2017_06_14.tar.gz</a>	59.9	82.5
MobileNet_v1_0.25_128	Code	<a href="#">mobilenet_v1_0.25_128_2017_06_14.tar.gz</a>	41.3	66.2
NASNet-A_Mobile_224#	Code	<a href="#">nasnet-a_mobile_04_10_2017.tar.gz</a>	74.0	91.6
NASNet-A_Large_331#	Code	<a href="#">nasnet-a_large_04_10_2017.tar.gz</a>	82.7	96.2

→ 我選這個

你也可以挑一個自己喜歡的 :)



# Tensorflow- 建立靜態圖

resnet\_v2\_50 model

---

## 使用 TF Slim 裡面的模型

- 直接使用別人寫好的模型架構，不用自己慢慢刻
- slim.arg\_scope: 可以一次設定所有架構中特定層的某個超參數，不用一層一層設定，詳細用法可以參考[其他大大的解說](#)

```
'''Tensorflow package'''
import tensorflow as tf
import tensorflow.contrib.slim as slim
import tensorflow.contrib.slim.nets as slimNet

with slim.arg_scope(slimNet.resnet_utils.resnet_arg_scope(batch_norm_decay=0.99)):
    _, layers_dict = slimNet.resnet_v2.resnet_v2_50(input_img, global_pool=False, is_training=is_training)
    conv_output = layers_dict['resnet_v2_50/block4']

with tf.variable_scope('CLASS_1'):
    pred = classifier(conv_output)
    pred_softmax = tf.nn.softmax(pred)
```

# Tensorflow- 建立靜態圖

resnet\_v2\_50 model

## 模型架構怎麼刻？(optional)

- 想了解 TF Slim 中的 resnet 如何架構起來，可以參考 `tf_model / resnet_v2.py`
- 其他 TF Slim 的經典模型架構 (alexnet、inception、vgg...) 刻法，可以研究[官方github](#)
- 大家可以組個讀書會，一起看 paper 、刻模型

The image shows a screenshot of the TensorFlow Slim repository. On the left, a file explorer displays the directory structure, with `tf_model` highlighted. An arrow points from `tf_model` to the `resnet_v2.py` file in the middle pane. The right pane shows the commit history for the `resnet_v2.py` file, listing various commits and their descriptions.

Commit	Description
tensorflow-gardener	Add decay/epsilon/updates_collections parameters to Inception v3 arg...
BUILD	Remove deprecated is_training from resnet_arg_scope.
alexnet.py	Remove so many more hourglass imports
alexnet_test.py	Fix the dlopen contrib test hack by making a pywrap_tensorflow modul
inception.py	Remove so many more hourglass imports
inception_v1.py	Switch tf.concat_v2 references in third_party/tensorflow to tf.concat.
inception_v1_test.py	Fix the dlopen contrib test hack by making a pywrap_tensorflow modul
inception_v2.py	Expanding the comments so that users know that logits shape and spa
inception_v2_test.py	Fix the dlopen contrib test hack by making a pywrap_tensorflow modul
inception_v3.py	Add decay/epsilon/updates_collections parameters to Inception v3 arg.

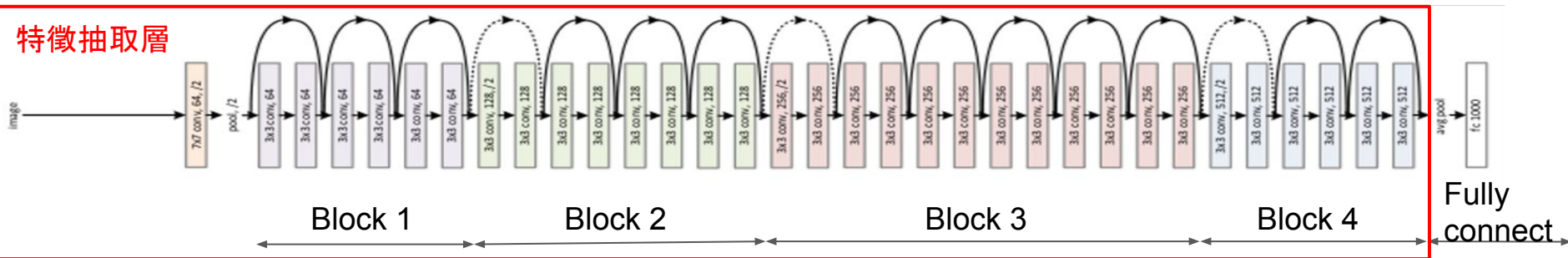
# Tensorflow- 建立靜態圖

resnet\_v2\_50 model

## 使用 TF Slim 裡面的模型

- 每個資料集最後分類的類別不一定一樣，但前面都是抽取特徵的過程
- 只取影像特徵抽取的部分 (input ~ block 4)
- 根據自己的任務，另外接分類層進行訓練

特徵抽取層



```
with slim.arg_scope(slimNet.resnet_utils.resnet_arg_scope(batch_norm_decay=0.99)):  
_, layers_dict = slimNet.resnet_v2.resnet_v2_50(input_img, global_pool=False, is_training=is_training)  
conv_output = layers_dict['resnet_v2_50/block4']
```

```
with tf.variable_scope('CLASS_1'):  
pred = classifier(conv_output)  
pred_softmax = tf.nn.softmax(pred)
```

# Tensorflow- 建立靜態圖

loss & update

---

## 計算 loss

- 做分類問題時會把預測值再過一個 softmax 層，讓所有類別的預測值加起來是 1 (模仿機率的樣子)
- 經過 softmax 後再透過 cross\_entropy 計算 loss (預測跟實際的差距)
- 使用 `tf.losses.softmax_cross_entropy(y_true, pred)` 同時做兩件事情

```
#### loss ####  
loss = tf.losses.softmax_cross_entropy(y_true, pred)
```

# Tensorflow- 建立靜態圖

loss & update

---

## update 模型參數

- `update_ops`: 只要使用到內建 BN 層, 就需透過 `tf.GraphKeys.UPDATE_OPS` 裡面的更新器, 才會更新到 BN 層的 mean & variance (`tf.GraphKeys` 後面會說明)
- `tf.control_dependencies`: 將數個操作流程有順序的綁在一起。
- 實際在 session 在執行 update 時, 會先執行 `update_ops` 再執行 `update`

```
#### udpate ####  
update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)  
with tf.control_dependencies(update_ops):  
    update = optimizer.minimize(loss)
```

# Tensorflow- 建立靜態圖

其他

---

- var\_list: 所有可被訓練的參數清單, 等會載入 pre-trained model 參數會用到
- saver: 儲存、載入模型
- init: 初始化我們定義好的 variable

```
#### other ####  
var_list = tf.trainable_variables() # 與 tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES) 相同  
saver = tf.train.Saver() # 處理模型儲存、載入  
init = tf.global_variables_initializer()
```

# Tensorflow- 資源管理機制

tf.GraphKeys

## 集合 (Collection)

- 全局的儲存機制來管理不同的資源 (張量、變量、ops)
- 使用 `tf.get_collection()` 可以獲得相對應的資源
- 下面列出一些 `tf` 自動維護的集合, 更多的集合請參考[官方網站](#)

## 常見集合

集合名稱	集合內容	使用場景
<code>tf.GraphKeys.GLOBAL_VARIABLES</code>	所有變量	保存TensorFlow模型
<code>tf.GraphKeys.TRAINABLE_VARIABLES</code>	可學習的變量 (一般指神經網路中的參數)	模型訓練
<code>tf.GraphKeys.SUMMARIES</code>	日志生成相關的張量	TensorFlow計算可視化 (TensorBoard)
<code>tf.GraphKeys.QUEUE_RUNNERS</code>	處理輸入的QueueRunner	輸入處理
<code>tf.GraphKeys.MOVING_AVERAGE_VARIABLES</code>	所有計算了滑動平均 值的變量	計算變量的滑動平均 值

# Tensorflow- 資源管理機制

tf.GraphKeys

## tf.GraphKeys.GLOBAL\_VARIABLES

- 定義 variable 的時候, tf 會自動把所有的 variable 記錄到清單

```
tf.reset_default_graph() # clean graph
# Declare the input node
with tf.name_scope('input'):
    x_input = tf.placeholder(shape = (None,x_train.shape[1]),
                             name = 'x_input',
                             dtype=tf.float32)
    y_out = tf.placeholder(shape = (None, y_train.shape[1]),
                           name = 'y_label',
                           dtype=tf.float32)

with tf.variable_scope('hidden_layer'):
    w1 = tf.get_variable('weight1',
                         shape= [x_train.shape[1], 25],
                         dtype=tf.float32,
                         initializer=tf.truncated_normal_initializer(stddev=0.1))
    b1 = tf.Variable(tf.zeros(shape=[25]))
    x_h1 = tf.nn.relu(tf.add(tf.matmul(x_input, w1), b1))

with tf.variable_scope('output_layer'):
    w2 = tf.Variable(tf.truncated_normal(shape=[25, y_train.shape[1]],stddev=0.1),
                     dtype = tf.float32)
    b2 = tf.get_variable('bias2',
                         shape=[y_train.shape[1]],
                         dtype=tf.float32,
                         initializer=tf.constant_initializer(0.0))
    output = tf.add(tf.matmul(x_h1, w2), b2)

with tf.name_scope('cross_entropy'):
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=output, labels=y_out))

with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(tf.nn.softmax(output),1), tf.argmax(y_out,1))
    compute_acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

with tf.name_scope('train'):
    train_step = tf.train.AdamOptimizer(learning_rate=FLAGS.lr).minimize(loss)
```

tf.GraphKeys.GLOBAL\_VARIABLES

<tf.Variable 'hidden\_layer/weight1:0' shape=(64, 25) dtype=float32\_ref>,  
<tf.Variable 'hidden\_layer/Variable:0' shape=(25,) dtype=float32\_ref>,  
<tf.Variable 'hidden\_layer/weight1/Adam:0' shape=(64, 25)  
dtype=float32\_ref>... 太多列不下

程式碼來源

tensorflow\_and\_dnn / 02\_tensorflow\_basis /  
01\_tf\_basic.ipynb



# Tensorflow- 資源管理機制

tf.GraphKeys

## tf.GraphKeys.TRAINABLE\_VARIABLES

- 定義 variable 的時候, tf 會自動把可被 optimizer 更新的 variable 記錄到清單

```
tf.reset_default_graph() # clean graph
# Declare the input node
with tf.name_scope('input'):
    x_input = tf.placeholder(shape = (None,x_train.shape[1]),
                             name = 'x_input',
                             dtype=tf.float32)
    y_out = tf.placeholder(shape = (None, y_train.shape[1]),
                           name = 'y_label',
                           dtype=tf.float32)

with tf.variable_scope('hidden_layer'):
    w1 = tf.get_variable('weight1',
                        shape=[x_train.shape[1], 25],
                        dtype=tf.float32,
                        initializer=tf.truncated_normal_initializer(stddev=0.1))
    b1 = tf.Variable(tf.zeros(shape=[25]))
    x_h1 = tf.nn.relu(tf.add(tf.matmul(x_input, w1), b1))

with tf.variable_scope('output_layer'):
    w2 = tf.Variable(tf.truncated_normal(shape=[25, y_train.shape[1]],stddev=0.1),
                    dtype = tf.float32)
    b2 = tf.get_variable('bias2',
                        shape=[y_train.shape[1]],
                        dtype=tf.float32,
                        initializer=tf.constant_initializer(0.0))
    output = tf.add(tf.matmul(x_h1, w2), b2)

with tf.name_scope('cross_entropy'):
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=output, labels=y_out))

with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(tf.nn.softmax(output),1), tf.argmax(y_out,1))
    compute_acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

with tf.name_scope('train'):
    train_step = tf.train.AdamOptimizer(learning_rate=FLAGS.lr).minimize(loss)
```

tf.GraphKeys.TRAINABLE\_VARIABLES

參數名稱

<tf.Variable 'hidden\_layer/weight1:0' shape=(64, 25) dtype=float32\_ref>  
<tf.Variable 'hidden\_layer/Variable:0' shape=(25,) dtype=float32\_ref>  
<tf.Variable 'output\_layer/Variable:0' shape=(25, 10) dtype=float32\_ref>  
<tf.Variable 'output\_layer/bias2:0' shape=(10,) dtype=float32\_ref>

程式碼來源

tensorflow\_and\_dnn / 02\_tensorflow\_basis /  
01\_tf\_basic.ipynb

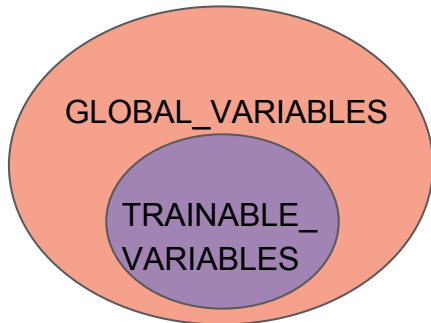
# Tensorflow- 資源管理機制

tf.GraphKeys

## TRAINABLE\_VARIABLES VS GLOBAL\_VARIABLES

tf.GraphKeys.TRAINABLE\_VARIABLES

```
<tf.Variable 'hidden_layer/weight1:0' shape=(64, 25) dtype=float32_ref>  
<tf.Variable 'hidden_layer/Variable:0' shape=(25,) dtype=float32_ref>  
<tf.Variable 'output_layer/Variable:0' shape=(25, 10) dtype=float32_ref>  
<tf.Variable 'output_layer/bias2:0' shape=(10,) dtype=float32_ref>
```



tf.GraphKeys.GLOBAL\_VARIABLES

```
<tf.Variable 'hidden_layer/weight1:0' shape=(64, 25)  
dtype=float32_ref>,  
<tf.Variable 'hidden_layer/Variable:0' shape=(25,) dtype=float32_ref>,  
<tf.Variable 'hidden_layer/weight1/Adam:0' shape=(64, 25)  
dtype=float32_ref>,  
<tf.Variable 'hidden_layer/weight1/Adam_1:0' shape=(64, 25)  
dtype=float32_ref>,  
<tf.Variable 'hidden_layer/Variable/Adam:0' shape=(25,) dtype=float32_ref>,  
<tf.Variable 'hidden_layer/Variable/Adam_1:0' shape=(25,) dtype=float32_ref>,  
<tf.Variable 'output_layer/Variable:0' shape=(25, 10) dtype=float32_ref>,  
<tf.Variable 'output_layer/bias2:0' shape=(10,) dtype=float32_ref>,  
<tf.Variable 'output_layer/Variable/Adam:0' shape=(25, 10) dtype=float32_ref>,  
<tf.Variable 'output_layer/Variable/Adam_1:0' shape=(25, 10) dtype=float32_ref>,
```

# Tensorflow- 資源管理機制

tf.GraphKeys


---

幫參數取一個好名字很重要呀！

- 方便參數管理
- 要對模型內特定參數進行操作的時候 (設定哪些可以被更新、取值), 需要指名參數名稱





A vertical bar on the left side of the text, composed of several colored segments: light blue, pink, orange, green, and grey.

Tensorflow- 載入模型參數

# 樣板程式碼流程

## 前置作業

- 生成資料檔案路徑清單
- 載入 packages
- 定義影像前處理
- 定義 model dict & callbacks

## Load data

- 載入 train, val, test.csv
- data generator

## Tensorflow- 建立靜態圖 (graph)

- graph, session
- optimizer
- placeholder
- resnet\_v2\_50 model
- loss, update

## Tensorflow- 載入模型參數

- tf.train.saver
- var\_list
- saver.restore

## Tensorflow- 實際執行模型訓練

- 執行 loss, update 操作 (train)
- 執行 loss 操作 (val)

## Tensorflow- 模型測試

- saver.restore
- 執行 pred\_softmax 操作
- accuracy

# Tensorflow- 載入模型參數

`sess.run(init)`

---

## 初始化模型

- 初始化 variable 裡面的值
- 將所有 variable 加入相對應的 `tf.GraphKeys`
- 不管有沒有要載入 pre-trained model 都需要初始化模型
- [更詳細請參考官方說明](#)

### Initializing variables

---

Before you can use a variable, it must be initialized. If you are programming in the low-level TensorFlow API (that is, you are explicitly creating your own graphs and sessions), you must explicitly initialize the variables. Most high-level frameworks such as `tf.contrib.slim`, `tf.estimator.Estimator` and `Keras` automatically initialize variables for you before training a model.

Explicit initialization is otherwise useful because it allows you not to rerun potentially expensive initializers when reloading a model from a checkpoint as well as allowing determinism when randomly-initialized variables are shared in a distributed setting.

To initialize all trainable variables in one go, before training starts, call `tf.global_variables_initializer()`. This function returns a single operation responsible for initializing all variables in the `tf.GraphKeys.GLOBAL_VARIABLES` collection. Running this operation initializes all variables. For example:

# Tensorflow- 載入模型參數

Load weights from pre-trained model

- 用參數清單告訴 saver 待會要從 pre-trained model 載入哪些參數
- 模型參數名稱要和 pre-trained model 參數名稱一樣才能順利載入

```
#### load weights from pre-train model ####  
saver_restore = tf.train.Saver([v for v in var_list if 'resnet_v2_50' in v.name])  
saver_restore.restore(sess, 'tf_ckpt/resnet_v2_50.ckpt')
```

resnet\_v2\_50/conv1/weights:0  
resnet\_v2\_50/conv1/biases:0  
resnet\_v2\_50/block1/unit\_1/bottleneck\_v2/preact/gamma:0  
resnet\_v2\_50/block1/unit\_1/bottleneck\_v2/preact/beta:0  
resnet\_v2\_50/block1/unit\_1/bottleneck\_v2/shortcut/weights:0  
resnet\_v2\_50/block1/unit\_1/bottleneck\_v2/shortcut/biases:0  
resnet\_v2\_50/block1/unit\_1/bottleneck\_v2/conv1/weights:0  
resnet\_v2\_50/block1/unit\_1/bottleneck\_v2/conv1/BatchNorm/gamma:0  
...

想知道模型架構的參數怎麼命名的, 可以參考 `tf_model / resnet_v2.py`



# 實作 - 實際執行模型訓練

台灣人工智慧學院 課程學習

File Edit View Insert Format Style Window Tools Help (82 shortcuts listed at 22:16)

## Tensorflow- 實際執行模型訓練

### loss, update

- graph 中定義的計算流程都會被連結起來
- 以計算 loss 為例
- 執行 `sess.run(loss, feed_dict={input_img: x[0], y_true: y, is_training: True})`
- 會連鎖執行下圖紅色流程和藍色流程
- Tensorflow 的 data flow 概念

```
def train_step(sess, input_img, y_true, is_training):  
    # 1. 將 input_img 和 y_true 喂入模型  
    feed_dict = {input_img: input_img, y_true: y_true, is_training: is_training}  
    # 2. 執行 sess.run 計算 loss 和 update  
    loss, update = sess.run([loss, update], feed_dict=feed_dict)  
    return loss, update
```

Training stage

Graph

Click to add equation editor

48



# Tensorflow- 實際執行模型訓練

# 樣板程式碼流程

## 前置作業

- 生成資料檔案路徑清單
- 載入 packages
- 定義影像前處理
- 定義 model dict & callbacks

## Load data

- 載入 train, val, test.csv
- data generator

## Tensorflow- 建立靜態圖 (graph)

- graph, session
- optimizer
- placeholder
- resnet\_v2\_50 model
- loss, update

## Tensorflow- 載入模型參數

- tf.train.saver
- var\_list
- saver.restore

## Tensorflow- 實際執行模型訓練

- 執行 loss, update 操作 (train)
- 執行 loss 操作 (val)

## Tensorflow- 模型測試

- saver.restore
- 執行 pred\_softmax 操作
- accuracy

# Tensorflow- 實際執行模型訓練

loss, update

- graph 中定義的計算流程都會被連結起來
- 以計算 loss 為例
- 執行 `sess.run(loss, feed_dict={input_img: x[0], y_true: y, is_training: True})`
- 會連鎖執行下圖紅色流程和藍色流程
- Tensorflow 的 data flow 概念

```
epoch_bar = tqdm(range(nb_epoch), desc="epoch", unit="epoch")
for epoch in epoch_bar:
```

Training stage

```
    ## train ##
    train_batch_bar = tqdm(range(n_batch), desc="train_batch", unit="batch", leave=False)

    for batch in train_batch_bar:
        x, y = generators.train_queue.get()

        _, train_loss = sess.run([update, loss], feed_dict={input_img: x[0], y_true: y,
                                                             is_training: True, lr: model_dict['reduce']
                                                             model_dict['train_batch_log'].push({'loss': train_loss})

    model_dict['history']['train_loss'].append(model_dict['train_batch_log'].avg_value('loss'))
    model_dict['train_batch_log'].reset()
```

Graph

```
main_graph = tf.Graph()
sess = tf.Session(graph=main_graph)
with main_graph.as_default():
    ### optimizer ###
    lr = tf.placeholder(tf.float32, shape=[])
    optimizer = tf.train.AdamOptimizer(lr)

    ### placeholder ###
    1 input_img = tf.placeholder(dtype=tf.float32, shape=(None, img_size, img_size, 3))
    2 y_true = tf.placeholder(dtype=tf.float32, shape=(None, 2))
    3 is_training = tf.placeholder(dtype=tf.bool, shape=[])

    4 with slim.arg_scope(slimNet.resnet_utils.resnet_arg_scope(batch_norm_decay=0.99)):
        layers_dict = slimNet.resnet_v2.resnet_v2_50(input_img, global_pool=False, is_training=is_training)
        conv_output = layers_dict['resnet_v2_50/block4']

    5 with tf.variable_scope('CLASS_1'):
        pred = classifier(conv_output, num_class)
        pred_softmax = tf.nn.softmax(pred)

    ### loss ###
    loss = tf.losses.softmax_cross_entropy(y_true, pred)

    ### update ###
    update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS) #使用內建的 batch normalization layer, 必須執行
    with tf.control_dependencies(update_ops): #tf.GraphKeys.UPDATE_OPS 才會更新到 BN 層的 mean, variance
        update = optimizer.minimize(loss)

    ### other ###
    var_list = tf.trainable_variables() # 與 tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES) 相同
    saver = tf.train.Saver() # 處理模型儲存、載入
    init = tf.global_variables_initializer()
```

# 實作 - 模型測試



The screenshot shows a Jupyter Notebook interface with a sidebar on the left containing a table of contents. The main area displays a notebook titled "Tensorflow- 模型測試" with the subtitle "load model, inference, accuracy".

**Tensorflow- 模型測試**  
load model, inference, accuracy

- 把訓練過程儲存的模型載入
- `pred_softmax` 得到預測結果
- 計算 `accuracy` (ML 課程有教過嚕)
- 在程式最後加上 `%%javascript Jupyter.notebook.session.delete();`
- 把 `notebook kernel` 關閉
- 主要是為了釋放顯卡記憶體

```
%%writefile test.py
import tensorflow as tf
import numpy as np
import sys
import os

# Load the saved model
model_path = os.path.join('models', 'model.pb')
with tf.Session() as sess:
    graph_def = tf.GraphDef()
    graph_def.FromString(open(model_path, 'rb').read())
    sess.graph.CopyFrom(graph_def)

    # Load the test data
    test_data_path = os.path.join('data', 'test_data.npy')
    test_data = np.load(test_data_path)

    # Run the inference
    pred_softmax = sess.run('pred_softmax', {
        'input': test_data
    })

    # Calculate the accuracy
    accuracy = np.mean(pred_softmax == test_data)

    print('Accuracy: %f' % accuracy)
```

At the bottom of the notebook, there is a cell with the following code:

```
%%javascript Jupyter.notebook.session.delete();
```



# Tensorflow- 模型測試

# 樣板程式碼流程

## 前置作業

- 生成資料檔案路徑清單
- 載入 packages
- 定義影像前處理
- 定義 model dict & callbacks

## Load data

- 載入 train, val, test.csv
- data generator

## Tensorflow- 建立靜態圖 (graph)

- graph, session
- optimizer
- placeholder
- resnet\_v2\_50 model
- loss, update

## Tensorflow- 載入模型參數

- tf.train.saver
- var\_list
- saver.restore

## Tensorflow- 實際執行模型訓練

- 執行 loss, update 操作 (train)
- 執行 loss 操作 (val)

## Tensorflow- 模型測試

- saver.restore
- 執行 pred\_softmax 操作
- accuracy

# Tensorflow- 模型測試

load model, inference, accuracy

- 把訓練過程儲存的模型載入
- pred\_softmax 得到預測結果
- 計算 accuracy (ML 課程有教過唷)
- 在程式最後加上 %%javascript Jupyter.notebook.session.delete();
- 把 notebook kernel 關閉
- 主要是為了釋放顯卡記憶體

```
''' load model 最好的結果'''  
saver.restore(sess, os.path.join('model', model_name) + '.ckpt')
```

```
for x, y, file in generators.get_test_data():  
    y_pred = sess.run(pred_softmax, feed_dict={input_img: x[0], is_training: False})  
  
    model_dict['testing']['y_true'].extend(y[:, 1].tolist())  
    model_dict['testing']['y_pred'].extend(y_pred[:, 1].tolist())  
    model_dict['testing']['files'].extend(file)
```

```
df = pd.DataFrame({"files":model_dict['testing']['files'],  
                  "y_true":model_dict['testing']['y_true'],  
                  "y_pred":model_dict['testing']['y_pred']})
```

```
from sklearn.metrics import accuracy_score  
  
accuracy = accuracy_score(df['y_true'], df['y_pred'].round())  
print('Accuracy on testing set is {}'.format(accuracy))
```

```
%%javascript  
Jupyter.notebook.session.delete();
```



# 練習 1 - Plant Seedlings Classification

---

- 將模板的資料集換成 [Plant seedlings](#), 分辨影像中的植物種類
- data 在 /data/examples/plant\_seedlings 裡面
- 生成 train、val、test 資料清單, 放到 data\_list/plant\_seedlings 資料夾  
提示: 使用 [os.listdir\(\)](#) 或 [os.walk\(\)](#) 取得 data 路徑
- 因為 kaggle 的 test data 沒有正確答案, 所以只能用 plant\_seedlings/train 去切資料
- 記得修改樣板中的 Config、Load data
- Testing 改用 sklearn.metrics 其他驗證方法 (請參考 ML 實作課)

## 練習 2 - Plant Seedlings Classification

- Layer transform: 試著凍結 block 1 & block 2 的參數, 在訓練的過程中不更新。  
提示 1: 觀察 *resnet* 參數命名  
提示 2: 在 *graph* 的 *optimizer.minimize(loss)* 多設定一個 *var\_list*
- 李宏毅 layer transform- 14:53 ~ 25:24

### Model Fine-tuning

One-shot learning: only a few examples in target domain

- Task description
  - Target data:  $(x^t, y^t)$  ← Very little
  - Source data:  $(x^s, y^s)$  ← A large amount
- Example: (supervised) speaker adaption
  - Target data: audio data and its transcriptions of specific user
  - Source data: audio data and transcriptions from many speakers
- Idea: training a model by source data, then fine-tune the model by target data
  - Challenge: only limited target data, so be careful about overfitting

## 額外練習 1 - flowering herbaceous (optional)

---

- 再換一個 ImageNet 裡面其中兩個類別的 data set 試試看
- data 在 /data/examples/flowering\_herbaceous 裡面
- 一樣的流程再做一遍
- 觀察 train test 結果有沒有什麼異常的地方

## 額外練習 2 - 換一個 pre-trained model (optional)

---

- 換換其他的 pre-trained model (上網找找有沒有其他人訓練好的模型, 或者自己訓練好的)
- 記得模型架構要跟 pre-trained model 一樣喔

# 今天的課程聽不懂怎麼辦？

---

沒關係，換莫凡講給你聽：)

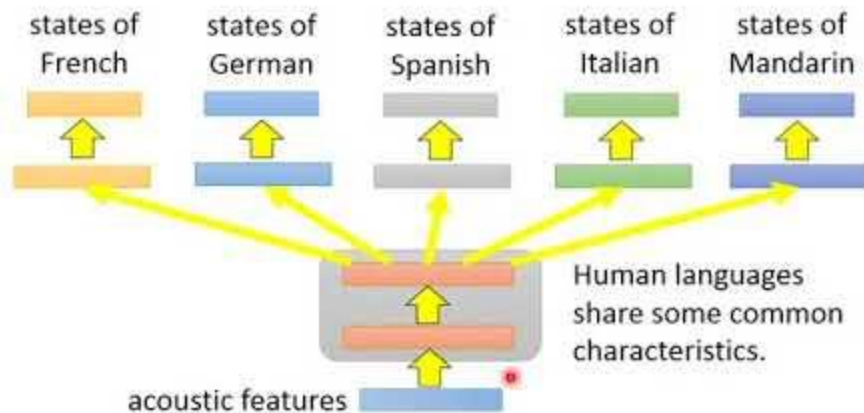


# 理論講授 - Multitask Learning

延伸閱讀

---

## Multitask Learning - Multilingual Speech Recognition

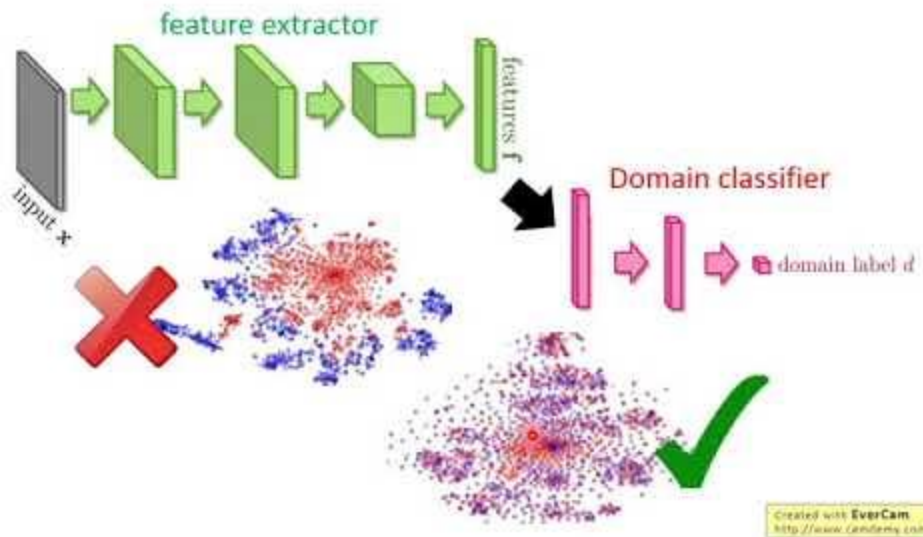


# 理論講授 - Domain-adversarial training

延伸閱讀

---

## Domain-adversarial training



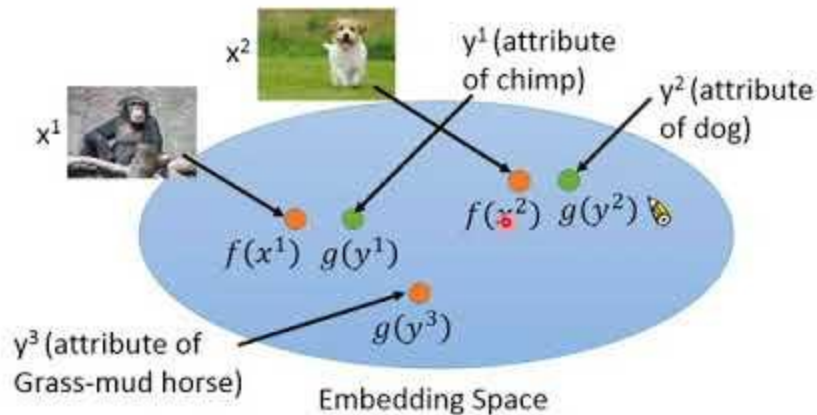
# 理論講授 - Zero-shot Learning

延伸閱讀

$f(*)$  and  $g(*)$  can be NN.

## Zero-shot Learning

- Attribute embedding





# 理論講授 - Self-taught learning

延伸閱讀

## Transfer Learning - Overview

		Source Data (not directly related to the task)	
		labelled	unlabeled
Target Data	labelled	<p>Fine-tuning</p> <p>Multitask Learning</p>	<p>Self-taught learning</p> <p>Rajat Raina , Alexis Battle , Honglak Lee , Benjamin Packer , Andrew Y. Ng, Self-taught learning: transfer learning from unlabeled data, ICML, 2007</p>
	unlabeled	<p>Domain-adversarial training</p> <p>Zero-shot learning</p>	<p>Self-taught Clustering</p> <p>Wenyuan Dai, Qiang Yang, Gui-Rong Xue, Yong Yu, "Self-taught clustering", ICML 2008</p>