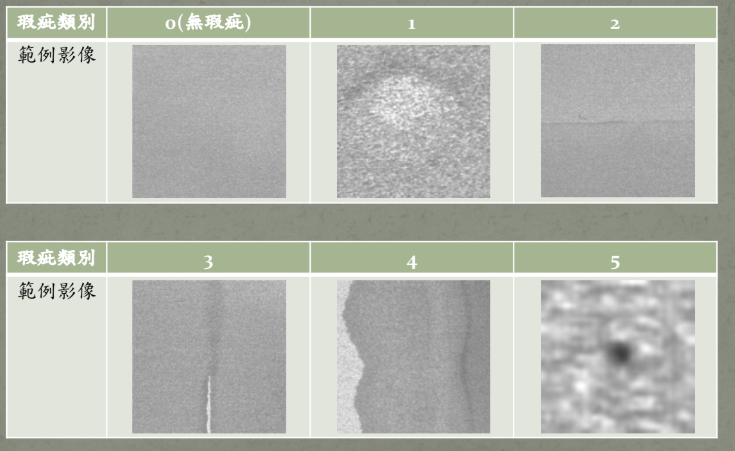
瑕疵檢測 呂旻翰 周恒生

# 介紹

在工業產線上,產品檢測是品質管理重要的一環,近年來深度學習更是大量運用在工業產品線的視覺檢測上。

利用深度學習的基礎實現瑕疵檢測系統 ,

本組Deep Learning所使用網路的網路為CNN

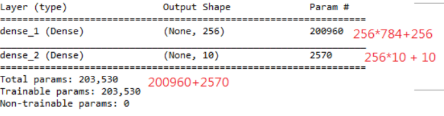
訓練資料共6種類別,其中包含一類正常資料與5類瑕疵資料

目標: 正確識別測試影像的瑕疵類別(包含無瑕疵影像)

# Rlated work

## Keras MNIST 手寫數字辨識

使用來自MNIST的資料庫，僅需將label轉成one-hot格式就可以用來訓練，訓練模組如圖:



而我們因為需要使用本地的資料庫，因此必須用pyhton自己寫出資料的走訪與存入，並注意會不會OOM。

# 方法

本段方法著重在本組嘗試各種的輸入方式，因為512\*152的圖片太大，無法一次讀入到記憶體中進行訓練。

## 使用iter容器去批次跑訓練

* 僅弄完輸入的部分
* 有批次跑過但因為批次能跑的量太少(太多會OOM)所以建立的模型在自己訓練的時候準確率就不高
* 放棄這個方法

## 把圖片縮小:外部處理篇

* 理由：太大的圖片放不進 RAM

使用 格式工廠將圖片的解析度縮小至 28 x 28

* 壓縮成效:一張圖片從181KB變成1.81KB

因此可以直接將全部圖片整個讀入記憶體做訓練了

28\*28的準確率: 只有0.9178791

研究csv人工對照的結果

會發現在類別0跟類別5

最容易判斷錯誤

可能是縮得太小導致圖片模糊

為此在下篇用128\*128去跑

## 把圖片縮小:內部處理篇

* 理由：太大的圖片放不進 RAM
* 使用 Numpy 在讀圖片的同時用PIL的Image.resize將圖片的解析度縮小至 128 x 128
* 本來是每調一次參數就重新跑整個程式，但是這樣都把時間浪費在讀檔，太憨，所以之後都直接利用 Spyder 能儲存和匯入的變數功能直接匯入，勝過Jupyter Notebook

## 資料前處理

* 用os.walk去跑完所有目錄
* 圖片存入np.array
* 然後設label
* 將label轉成one-hot格式

## 加速 model.fit 的運算效率

* 一開始用 CPU 硬跑，使用率 95% 卻 50 ms/step
* 後來發現 TensorFlow 有提供 **GPU 版**，搭配 NVIDIA 推出的 CUDA，可以運用顯卡的**平行運算**能力大幅減少運算時間，達到 4 ms/step
* 雖然顯卡比 CPU 快很多，但是顯卡只有 8 GB 記憶體，一旦某些參數設定過大就會跳 **Out Of Memory**，為此**折騰**很多次
* **batch\_size** 調多大就有多快，但是發現調大會變很快的時候已經為時已晚

## 前 80 次的上傳做了什麼調整：

* 捲積層：
  + 疊 0、1、2層
  + 調 filter 數量
  + 調 kernel size
  + Dropout 層的比例
* 隱藏層：
  + 神經元的數量
  + 兩層隱藏層
  + Dropout 層的比例
* 簡單來講就是一直湊組合
* optimizer
* model.fit：

batch\_size 的大小

epoch 的次數

validation\_split 的比例

* 但是，有幾次是**用完全一模一樣的參數和設定**去 fit 模型，每個模型的分差最大竟然可以差到 10 分
* **盲點**：完全沒有觀察以 epoch 為 x 軸的圖表去評斷模型的好壞

## 前 81 次到 120 次做了什麼調整：

* 只好繼續問 Google，發現大部分的人在討論模型的優化時，不只是瞎試，還會**根據 acc 和 val\_acc 隨 epoch 遞增的走向圖表**是否**同時正常地成長和有無 Overfit、Underfit 來判斷一個模型的好壞**
* **解決不了的癥結點：**不管怎麼調 validation\_split 的比例，一開始的 epoch，val\_acc 全部都是 0，到一定的 epoch 之後 val\_acc 會直接從 0 衝到 1，之後的幾個 epoch，val\_acc 會在 0 跟 1 之間大震盪，再之後 val\_acc 就幾乎都大於 0.98，Google 沒有人有跟我們一樣的狀況，唯一比較接近的狀況是 val\_acc 永遠是 1，有人給出的答案是**訓練的資料對模型來說太簡單了**，所以我們也認為瑕疵檢測的資料也是這個原因才會有震盪的現象

## 模型

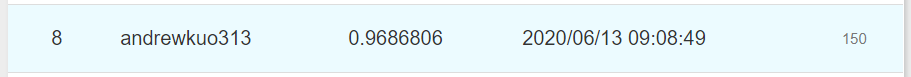
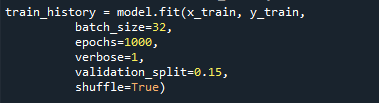
## 

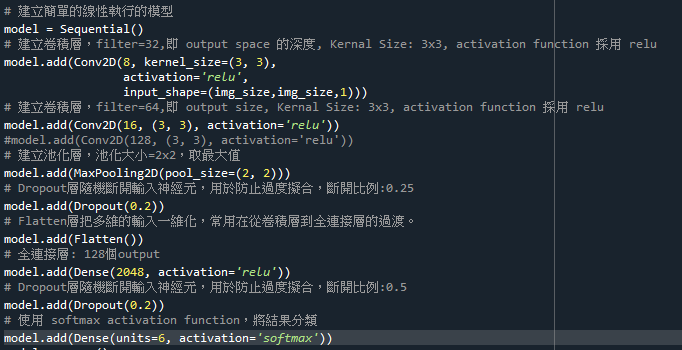
## Csv生成

* 用了csv.writer跟pandas的data.dropna()（去除空行）
* 再把輸出的label貼到test.csv中

# 實驗結果跟結論

## 最高分：

總共上傳資料的次數：159 次最高分的參數設定：



## 模型的 acc 完全不上升的原因推測：

* validation\_split 的比例過大
* 沒有捲積層
* 捲積層只有一層但 filter 數量太小
* 隱藏層的神經元數量太少
* Dropout 的比例太大

## 分數跑到 0.90 分以下的原因推測：

* Epoch 次數過少
* 隱藏層的神經元數量太少