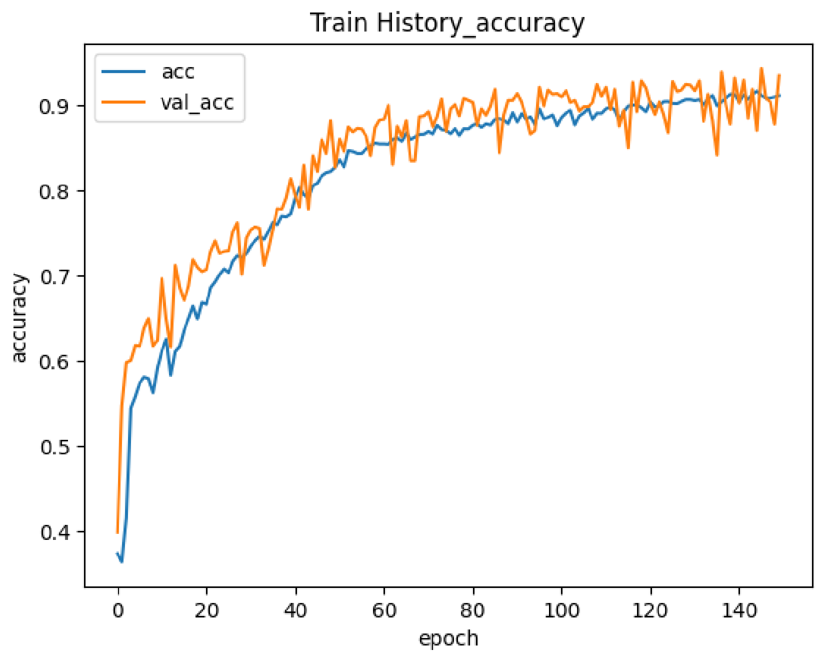
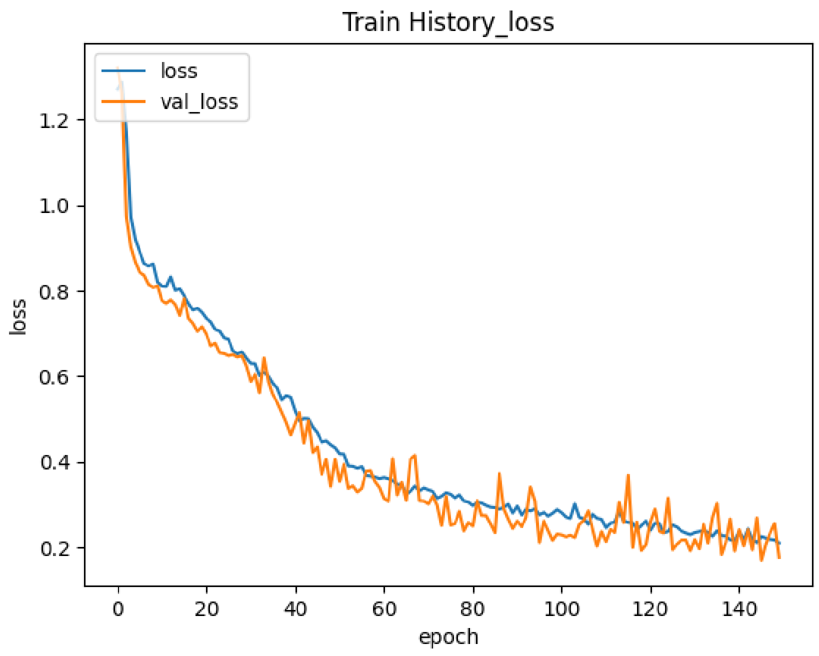
CNN 模型_零件辨識　　　　　　　　　　　　　　資管三 B 109403533 林采璇

一、　工件辨識準確率及損失率

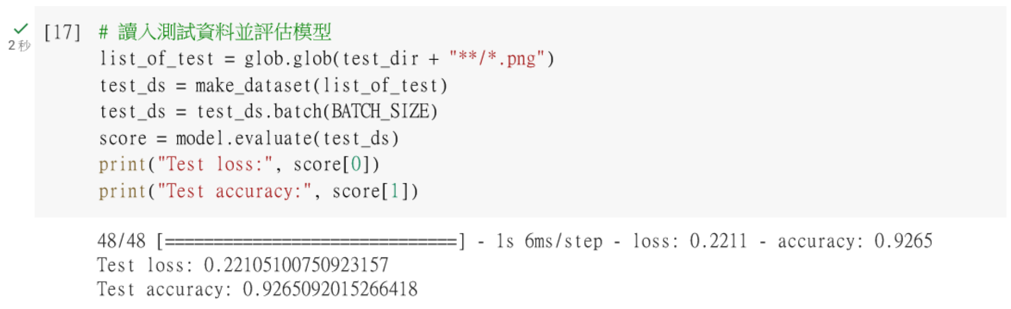
(1)　Train accuracy history 圖



(2)　Train loss history 圖



(3)　Test loss ,Test accuracy 截圖:



```
[17] # 讀入測試資料並評估模型
     list_of_test = glob.glob(test_dir + "**/*.png")
     test_ds = make_dataset(list_of_test)
     test_ds = test_ds.batch(BATCH_SIZE)
     score = model.evaluate(test_ds)
     print("Test loss:", score[0])
     print("Test accuracy:", score[1])
```

```
48/48 [==============================] - 1s 6ms/step - loss: 0.2211 - accuracy: 0.9265
Test loss: 0.22105100750923157
Test accuracy: 0.9265092015266418
```

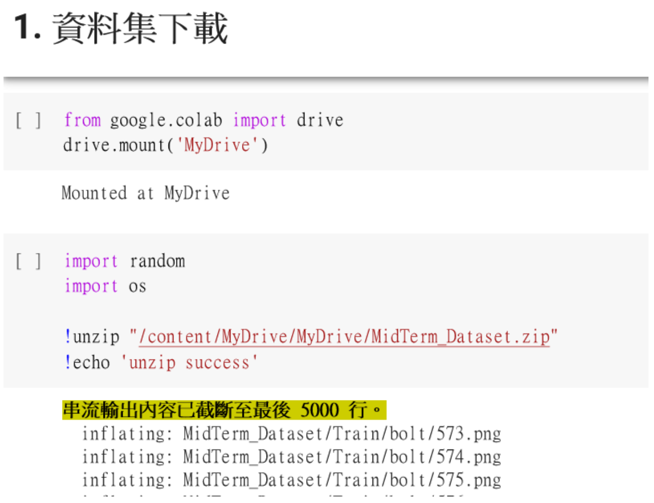
二、　詳述專題：包括資料前處理、選擇模型建立、調整參數等

(一)　前言：

剛開始的前幾次在建模時，一不小心下手太重，疊了太多層（捲積、池化、正規、Dropout 的排列組合），各種參數也設的不知輕重，結果就把 google colab 的系統 RAM 和 GPU RAM 玩到超過上限，帳號也被鎖了。後來，便直接簡化模型，幾次調整之後，發現訓練的狀況還行，也就是繳交的這份模型。

(二)　正文：以下為整個專案程式的實作過程

(1)　資料集下載：為了方便每次資料集的載入，將原始 zip 存於雲端硬碟，每次訓練有需要便存取。



1. 資料集下載

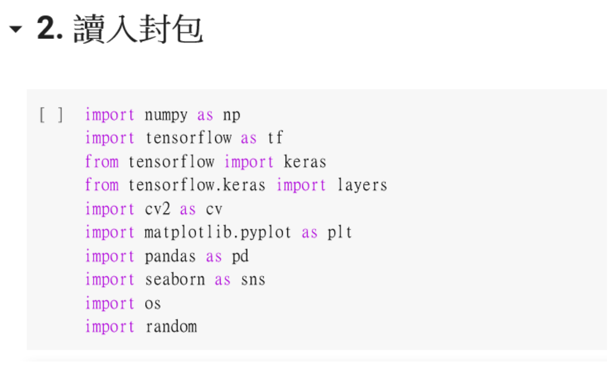
```
[ ] from google.colab import drive
    drive.mount('MyDrive')
```

```
Mounted at MyDrive
```

```
[ ] import random
    import os

    !unzip "/content/MyDrive/MyDrive/MidTerm_Dataset.zip"
    !echo 'unzip success'
```

```
串流輸出內容已截斷至最後 5000 行。
  inflating: MidTerm_Dataset/Train/bolt/573.png
  inflating: MidTerm_Dataset/Train/bolt/574.png
  inflating: MidTerm_Dataset/Train/bolt/575.png
```

(2) 讀入封包：把訓練過程所需要的封包讀入。

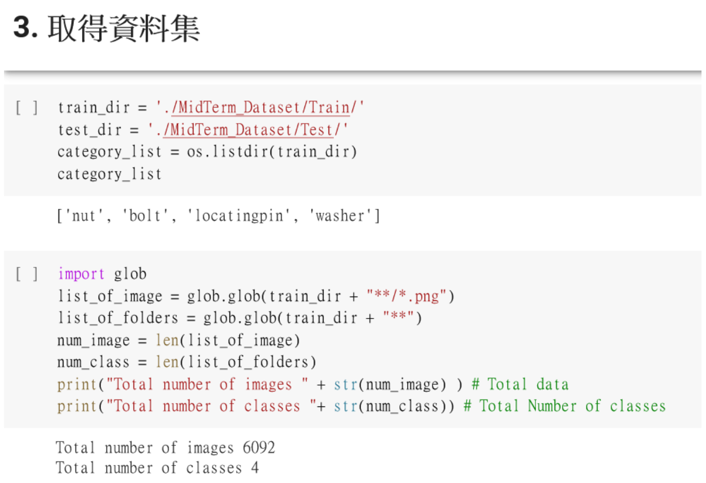


▾ **2.** 讀入封包

```
[ ]  import numpy as np
     import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras import layers
     import cv2 as cv
     import matplotlib.pyplot as plt
     import pandas as pd
     import seaborn as sns
     import os
     import random
```

(3) 取得資料集：

① 原始資料夾內有分屬兩個資料夾：Train 和 Test，兩個資料夾底下又再各自有分屬四種零件的資料夾，印出確認資料夾內容。

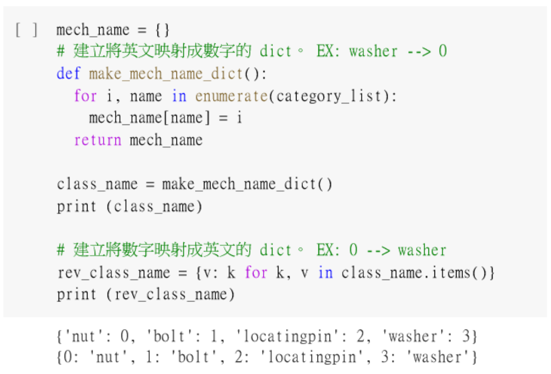


3. 取得資料集

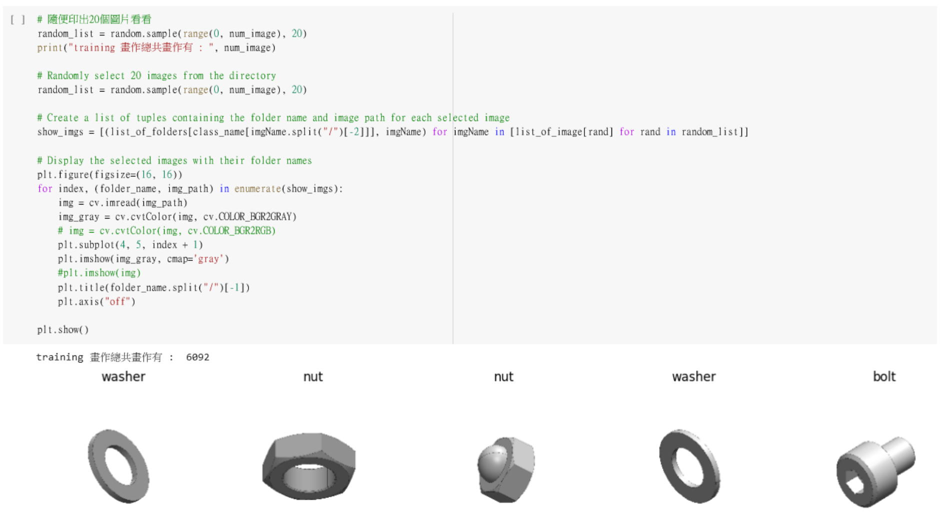
```
[ ]  train_dir = './MidTerm_Dataset/Train/'
     test_dir = './MidTerm_Dataset/Test/'
     category_list = os.listdir(train_dir)
     category_list

     ['nut', 'bolt', 'locatingpin', 'washer']
```

```
[ ]  import glob
     list_of_image = glob.glob(train_dir + "**/*.png")
     list_of_folders = glob.glob(train_dir + "**")
     num_image = len(list_of_image)
     num_class = len(list_of_folders)
     print("Total number of images " + str(num_image) ) # Total data
     print("Total number of classes "+ str(num_class)) # Total Number of classes

     Total number of images 6092
     Total number of classes 4
```

② 接著延續上面取的的資料夾資料建立字典，使零件名稱和數字間相互映射(EX: washer --> 0；0 --> washer)，並嘗試印出訓練集圖片。



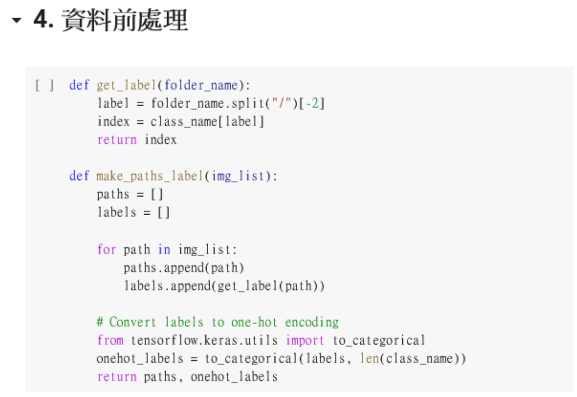
```
[ ]  mech_name = {}
     # 建立將英文映射成數字的 dict。 EX: washer --> 0
     def make_mech_name_dict():
       for i, name in enumerate(category_list):
         mech_name[name] = i
       return mech_name

     class_name = make_mech_name_dict()
     print (class_name)

     # 建立將數字映射成英文的 dict。 EX: 0 --> washer
     rev_class_name = {v: k for k, v in class_name.items()}
     print (rev_class_name)

     {'nut': 0, 'bolt': 1, 'locatingpin': 2, 'washer': 3}
     {0: 'nut', 1: 'bolt', 2: 'locatingpin', 3: 'washer'}
```



```
[ ]  # 隨便印出20個圖片看看
     random_list = random.sample(range(0, num_image), 20)
     print("training 畫作總共畫作有 : ", num_image)

     # Randomly select 20 images from the directory
     random_list = random.sample(range(0, num_image), 20)

     # Create a list of tuples containing the folder name and image path for each selected image
     show_imgs = [(list_of_folders[class_name[imgName.split("/")[-2]]], imgName) for imgName in [list_of_image[rand] for rand in random_list]]

     # Display the selected images with their folder names
     plt.figure(figsize=(16, 16))
     for index, (folder_name, img_path) in enumerate(show_imgs):
         img = cv.imread(img_path)
         img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
         # img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
         plt.subplot(4, 5, index + 1)
         plt.imshow(img_gray, cmap='gray')
         #plt.imshow(img)
         plt.title(folder_name.split("/")[-1])
         plt.axis("off")

     plt.show()

     training 畫作總共畫作有 :  6092
```

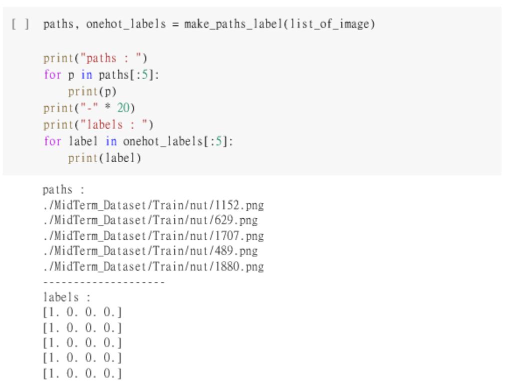
(4)　資料前處理

①　把標籤轉成 one-hot



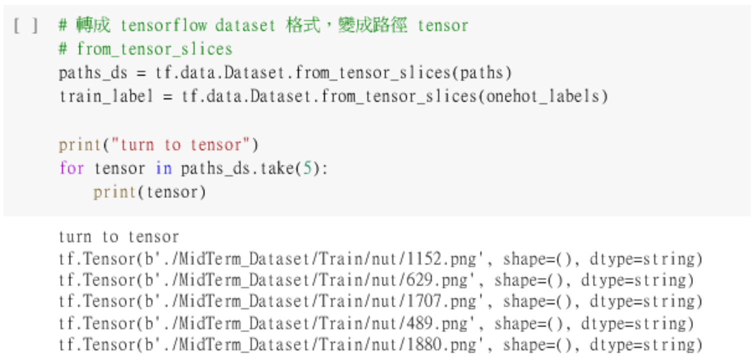
4. 資料前處理

```
[ ]  def get_label(folder_name):
         label = folder_name.split("/")[-2]
         index = class_name[label]
         return index

     def make_paths_label(img_list):
         paths = []
         labels = []

         for path in img_list:
             paths.append(path)
             labels.append(get_label(path))

         # Convert labels to one-hot encoding
         from tensorflow.keras.utils import to_categorical
         onehot_labels = to_categorical(labels, len(class_name))
         return paths, onehot_labels
```



```
[ ]  paths, onehot_labels = make_paths_label(list_of_image)

     print("paths : ")
     for p in paths[:5]:
         print(p)
     print("-" * 20)
     print("labels : ")
     for label in onehot_labels[:5]:
         print(label)

     paths :
     ./MidTerm_Dataset/Train/nut/1152.png
     ./MidTerm_Dataset/Train/nut/629.png
     ./MidTerm_Dataset/Train/nut/1707.png
     ./MidTerm_Dataset/Train/nut/489.png
     ./MidTerm_Dataset/Train/nut/1880.png
     --------------------
     labels :
     [1. 0. 0. 0.]
     [1. 0. 0. 0.]
     [1. 0. 0. 0.]
     [1. 0. 0. 0.]
     [1. 0. 0. 0.]
```

②　再透過上一步的 one-hot，把它轉成 Tensorflow dataset 格式

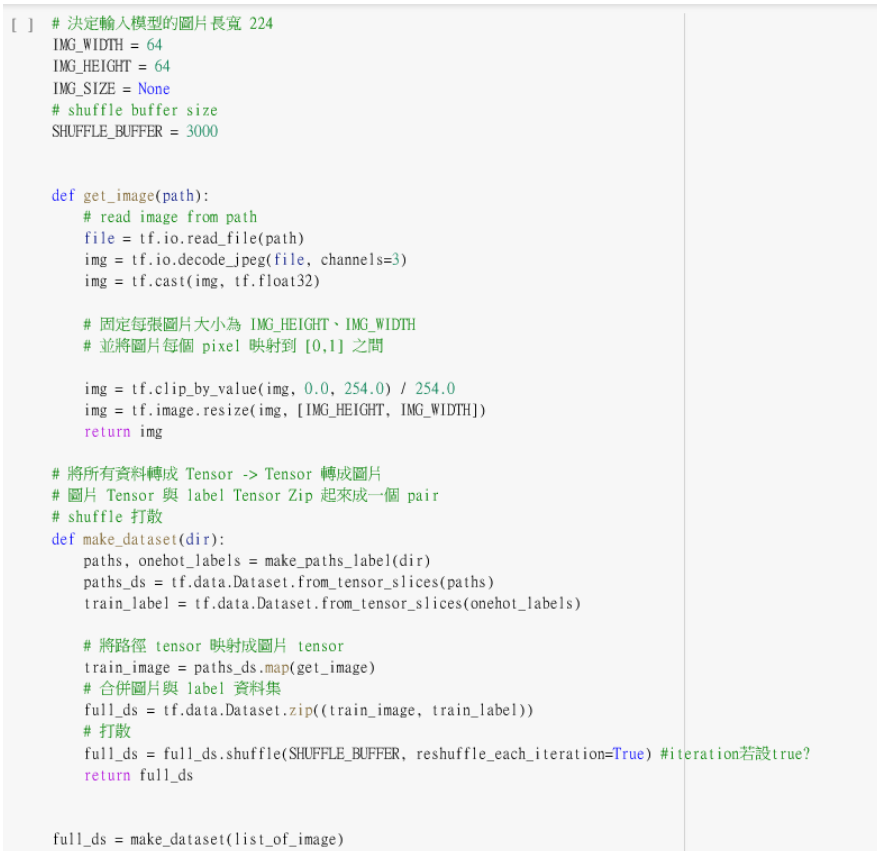


```
[ ]  # 轉成 tensorflow dataset 格式，變成路徑 tensor
     # from_tensor_slices
     paths_ds = tf.data.Dataset.from_tensor_slices(paths)
     train_label = tf.data.Dataset.from_tensor_slices(onehot_labels)

     print("turn to tensor")
     for tensor in paths_ds.take(5):
         print(tensor)

     turn to tensor
     tf.Tensor(b'./MidTerm_Dataset/Train/nut/1152.png', shape=(), dtype=string)
     tf.Tensor(b'./MidTerm_Dataset/Train/nut/629.png', shape=(), dtype=string)
     tf.Tensor(b'./MidTerm_Dataset/Train/nut/1707.png', shape=(), dtype=string)
     tf.Tensor(b'./MidTerm_Dataset/Train/nut/489.png', shape=(), dtype=string)
     tf.Tensor(b'./MidTerm_Dataset/Train/nut/1880.png', shape=(), dtype=string)
```

③ 訓練用的圖形處理

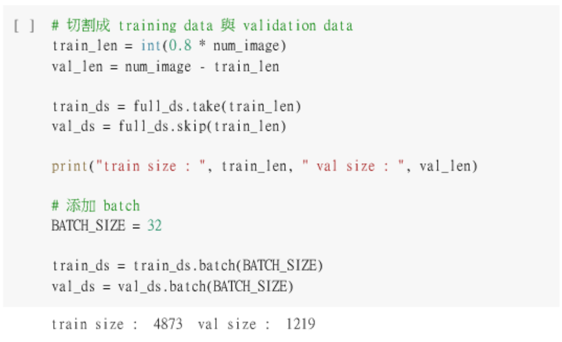
(包含 image 大小、shuffle buffer 相關的處理)

- img_width 和 img _height 原本是設定圖片原始長寬的 224，但為了跑模型時，不要消耗太多 google colab 資源，就暫且減小了。
- Shuffle_Buffer 有看到相關資料寫說設定成資料集大小最好，代表打亂程度越大。但設定成資料集大小(6092)，會直接超越 google colab 能執行的 RAM 上限，故粗略砍半設定。



```
[ ] # 決定輸入模型的圖片長寬 224
    IMG_WIDTH = 64
    IMG_HEIGHT = 64
    IMG_SIZE = None
    # shuffle buffer size
    SHUFFLE_BUFFER = 3000


    def get_image(path):
        # read image from path
        file = tf.io.read_file(path)
        img = tf.io.decode_jpeg(file, channels=3)
        img = tf.cast(img, tf.float32)

        # 固定每張圖片大小為 IMG_HEIGHT、IMG_WIDTH
        # 並將圖片每個 pixel 映射到 [0,1] 之間

        img = tf.clip_by_value(img, 0.0, 254.0) / 254.0
        img = tf.image.resize(img, [IMG_HEIGHT, IMG_WIDTH])
        return img

    # 將所有資料轉成 Tensor -> Tensor 轉成圖片
    # 圖片 Tensor 與 label Tensor Zip 起來成一個 pair
    # shuffle 打散
    def make_dataset(dir):
        paths, onehot_labels = make_paths_label(dir)
        paths_ds = tf.data.Dataset.from_tensor_slices(paths)
        train_label = tf.data.Dataset.from_tensor_slices(onehot_labels)

        # 將路徑 tensor 映射成圖片 tensor
        train_image = paths_ds.map(get_image)
        # 合併圖片與 label 資料集
        full_ds = tf.data.Dataset.zip((train_image, train_label))
        # 打散
        full_ds = full_ds.shuffle(SHUFFLE_BUFFER, reshuffle_each_iteration=True) #iteration若設true?
        return full_ds


    full_ds = make_dataset(list_of_image)
```

④ 切割成 training data 與 validation data (8:2 拆分)

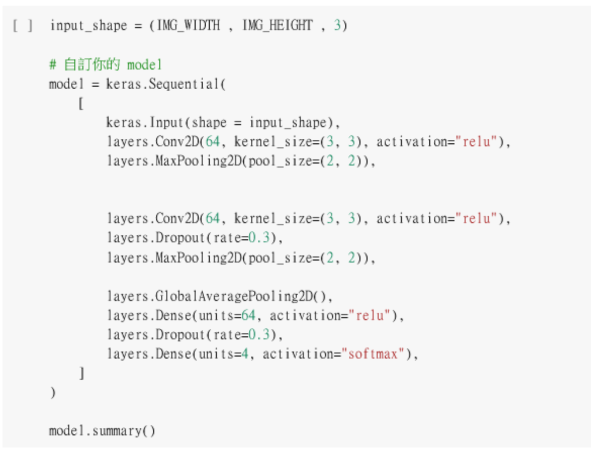


```
[ ] # 切割成 training data 與 validation data
    train_len = int(0.8 * num_image)
    val_len = num_image - train_len

    train_ds = full_ds.take(train_len)
    val_ds = full_ds.skip(train_len)

    print("train size : ", train_len, " val size : ", val_len)

    # 添加 batch
    BATCH_SIZE = 32

    train_ds = train_ds.batch(BATCH_SIZE)
    val_ds = val_ds.batch(BATCH_SIZE)

    train size :  4873  val size :  1219
```

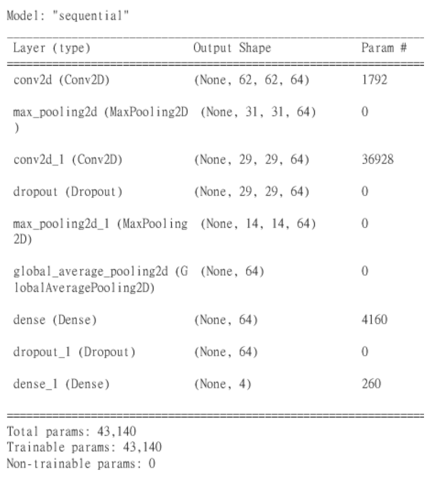
(5)　模型建立：

就如同我前言所言，簡化成較為單純的模型，兩次捲積，兩次池化，以及適當的 dropout。

5. 建立模型



```
[ ]  input_shape = (IMG_WIDTH , IMG_HEIGHT , 3)

     # 自訂你的 model
     model = keras.Sequential(
         [
             keras.Input(shape = input_shape),
             layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
             layers.MaxPooling2D(pool_size=(2, 2)),


             layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
             layers.Dropout(rate=0.3),
             layers.MaxPooling2D(pool_size=(2, 2)),

             layers.GlobalAveragePooling2D(),
             layers.Dense(units=64, activation="relu"),
             layers.Dropout(rate=0.3),
             layers.Dense(units=4, activation="softmax"),
         ]
     )

     model.summary()
```

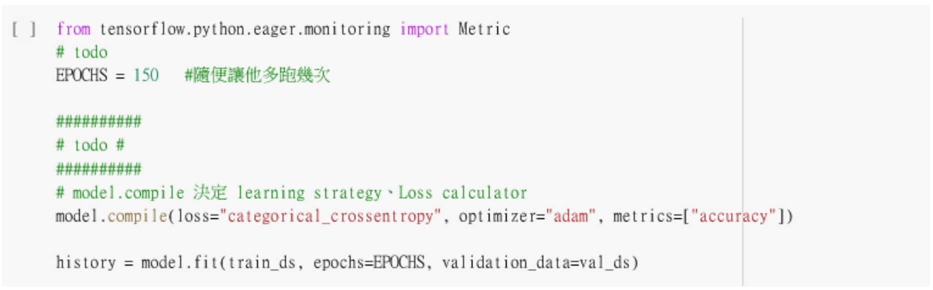


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 31, 31, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 29, 29, 64)        36928

 dropout (Dropout)           (None, 29, 29, 64)        0

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 global_average_pooling2d (G  (None, 64)               0
 lobalAveragePooling2D)

 dense (Dense)               (None, 64)                4160

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 43,140
Trainable params: 43,140
Non-trainable params: 0
```

(6)　制定訓練計畫：

其實前幾次跑訓練計畫時，害怕會 overfitting，所以就 Epoch=10 次為單位，疊加著跑，並輸出相應的 Test loss、Test accuracy 觀察。後來發現累加起來也有跑到 Epoch=150，就索性跑一次 Epoch 設定是 150 的。

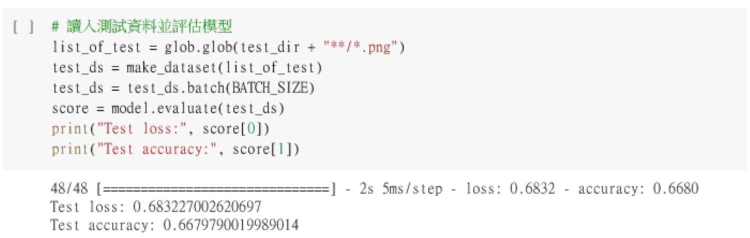
6. 制定訓練計畫



```
[ ]  from tensorflow.python.eager.monitoring import Metric
     # todo
     EPOCHS = 150    #隨便讓他多跑幾次

     ##########
     # todo #
     ##########
     # model.compile 決定 learning strategy、Loss calculator
     model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

     history = model.fit(train_ds, epochs=EPOCHS, validation_data=val_ds)
```

[前幾次的紀錄]

- Epoch=30 (Test loss: 0.68；Test accuracy:0.66)



```
[ ]  # 讀入測試資料並評估模型
     list_of_test = glob.glob(test_dir + "**/*.png")
     test_ds = make_dataset(list_of_test)
     test_ds = test_ds.batch(BATCH_SIZE)
     score = model.evaluate(test_ds)
     print("Test loss:", score[0])
     print("Test accuracy:", score[1])

     48/48 [==============================] - 2s 5ms/step - loss: 0.6832 - accuracy: 0.6680
     Test loss: 0.6832270026206970
     Test accuracy: 0.6679790019989014
```

- Epoch=50（Test loss: 0.46；Test accuracy:0.80）

```
[ ]  # 讀入測試資料並評估模型
     list_of_test = glob.glob(test_dir + "**/*.png")
     test_ds = make_dataset(list_of_test)
     test_ds = test_ds.batch(BATCH_SIZE)
     score = model.evaluate(test_ds)
     print("Test loss:", score[0])
     print("Test accuracy:", score[1])
```

```
48/48 [==============================] - 1s 5ms/step - loss: 0.4655 - accuracy: 0.8064
Test loss: 0.4654691219329834
Test accuracy: 0.806430459022522
```

- Epoch=130（Test loss: 0.24；Test accuracy:0.90）

```
[ ]  # 讀入測試資料並評估模型
     list_of_test = glob.glob(test_dir + "**/*.png")
     test_ds = make_dataset(list_of_test)
     test_ds = test_ds.batch(BATCH_SIZE)
     score = model.evaluate(test_ds)
     print("Test loss:", score[0])
     print("Test accuracy:", score[1])
```

```
48/48 [==============================] - 1s 4ms/step - loss: 0.2454 - accuracy: 0.9062
Test loss: 0.2454147487878794
Test accuracy: 0.9061679840087891
```

（7） 評定模型（Test loss: 0.2210；Test accuracy:0.9265）

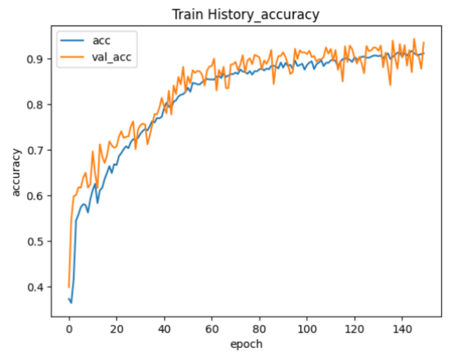
```
[ ]  # 讀入測試資料並評估模型
     list_of_test = glob.glob(test_dir + "**/*.png")
     test_ds = make_dataset(list_of_test)
     test_ds = test_ds.batch(BATCH_SIZE)
     score = model.evaluate(test_ds)
     print("Test loss:", score[0])
     print("Test accuracy:", score[1])
```

```
48/48 [==============================] - 1s 6ms/step - loss: 0.2211 - accuracy: 0.9265
Test loss: 0.22105100750923157
Test accuracy: 0.9265092015266418
```

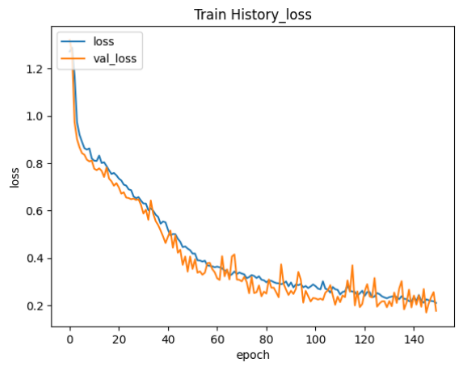
7. 評估模型

```
[ ]  print(history.history.keys())

     plt.plot(history.history["accuracy"])
     plt.plot(history.history["val_accuracy"])
     plt.title("Train History_accuracy")
     plt.ylabel("accuracy")
     plt.xlabel("epoch")
     plt.legend(['acc','val_acc'], loc='upper left')
     plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

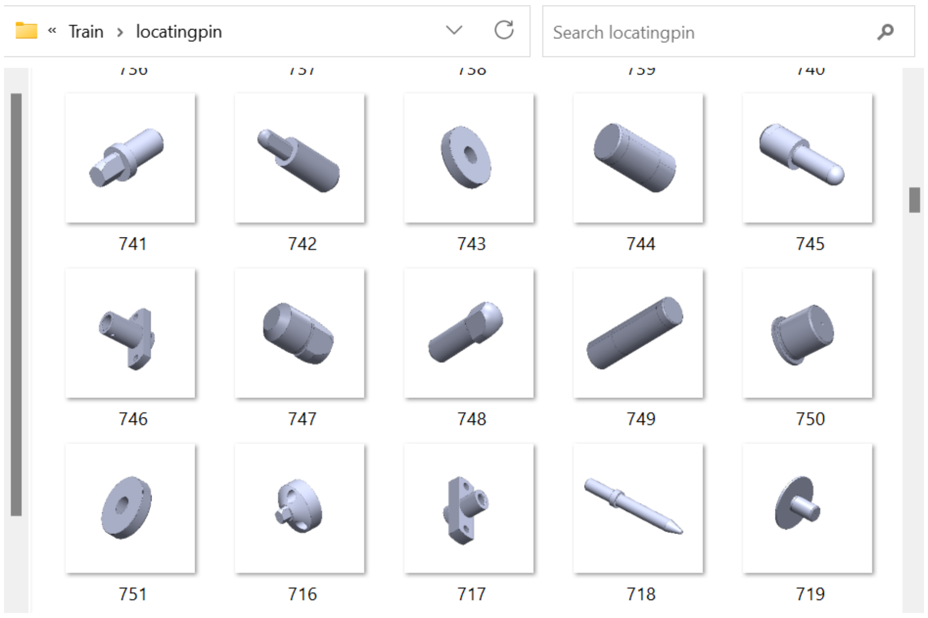


```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("Train History_loss")
plt.ylabel("loss")
plt.xlabel("epoch")
plt.legend(['loss','val_loss'], loc='upper left')
plt.show()
```



(三) 結論：

最終訓練出來的 Test Accuracy 和 Test Lost 其實比我預想中的好，因為在檢查訓練資料集的時候有發現，location pin 的訓練資料集並不像其他零件那麼的整齊，明顯有混到其他零件的樣子(ex. 下圖的 743 和 751 應該是 washer?)。



再者，目前未包含圖片方向調整等的資料前處理方式，若加入，或許準確度能再更進一步提升。