NCU 演算法 Midterm Project

Mechanical parts (Bolt, Nut, Washer, Pin) Recognition

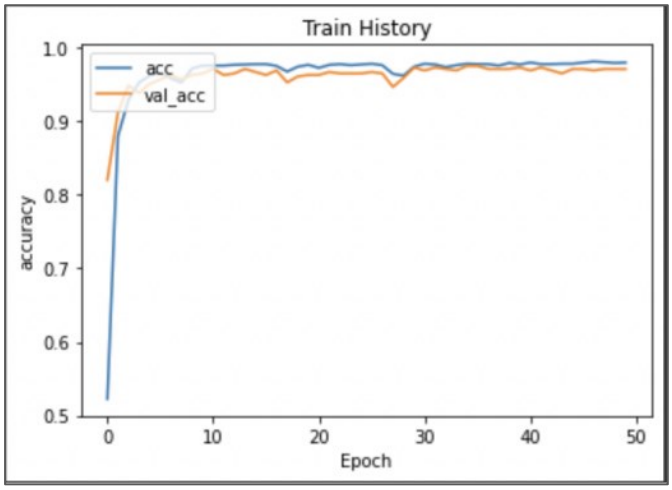
資管三 A 109403019 鄒翔宇

COLAB LINKS: click me

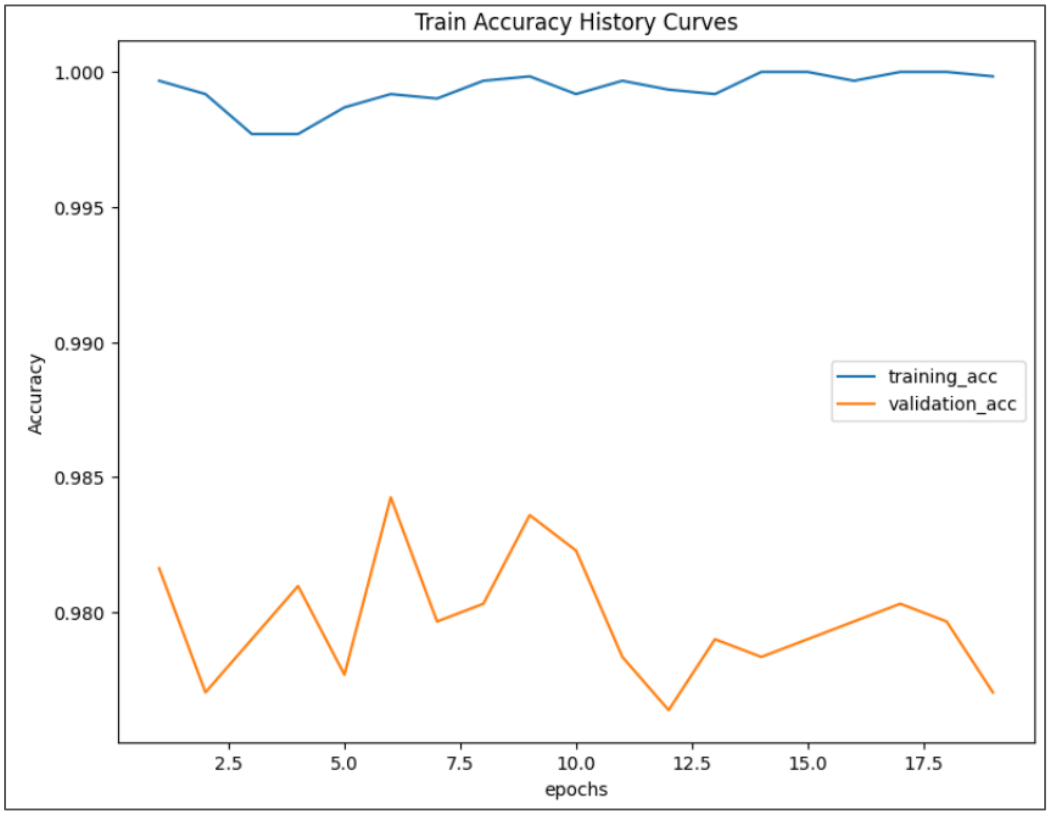
一、 工件辨識準確率及損失率

1. Train accuracy history 圖

Ex :

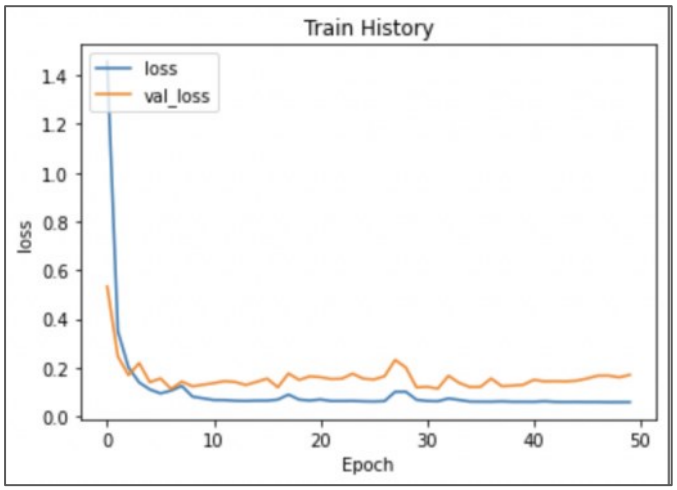


Train accuracy history 圖:



2. Train loss history 圖

Ex :



Train loss history 圖:

3. Test loss ,Test accuracy 截圖:

```
1 score=model.evaluate(test_data,  verbose=0)
2 print('Test  lose:',  score[0])
3 print('Test  Accuracy:',  score[1])
```

```
Test lose: 0.17291387915611267
Test Accuracy: 0.9835957884788513
```

二、 請詳述你如何實作期末專題，包括資料前處理、選擇模型建立、調整參數...等等

1. 導入套件

導入所需套件，因為不是一開始就知道需要那些套件，所以邊做邊加，並增加註解，未來複習時也能夠快速理解每一個套件負責的項目。

```
import os    # os 模組，用於操作文件和目錄
import zipfile    # zipfile 模組，用於讀寫 ZIP 文件
import random    # random 模組，用於生成隨機數
import numpy as np    # numpy 模組，用於數組和矩陣運算

import tensorflow as tf    # tensorflow 模組，用於機器學習和深度學習
from tensorflow import keras    # keras 模組，用於建構模型
from keras import layers    # layers 模組，用於定義深度學習模型的各層
from keras.models import Sequential    # Sequential class, 用於建構網路架構
from keras.layers import Conv2D, BatchNormalization, Activation, MaxPooling2D, Dense, Dropout, Flatten # 各種 Keras 層，用於構建卷積神經網絡
from keras.regularizers import l2    # l2 正則化，用於防止Overfitting
from keras.callbacks import EarlyStopping, ReduceLROnPlateau    # Keras 回調函數，用於在訓練過程中監控模型表現並進行相應調整

from PIL import Image    # PIL 模組，用於處理圖像

from matplotlib import pyplot as plt    # pyplot 模組，用於繪製圖表
```

2. 下載資料集

使用助教提供的方法從 Google Drive 上下載資料集，並解壓縮到指定路徑。

```
!gdown https://drive.google.com/uc?id=1qCHfycy91EyUFzilBvxu8hYVYp0l6jCh  --output /tmp/MidTerm_Dataset.zip
```

```
Downloading...
From: https://drive.google.com/uc?id=1qCHfycy91EyUFzilBvxu8hYVYp0l6jCh
To: /tmp/MidTerm_Dataset.zip
100% 50.4M/50.4M [00:00<00:00, 104MB/s]
```

```
local_zip='/tmp/MidTerm_Dataset.zip'
zip_ref=zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()
```

下載解壓縮後，檢查一下資料集的狀況，訓練集中每個分類各有 1523 張圖片，測試集每個分類各有 381 張。

```
def walk_through_dir(directory_name):
    for dirpaths,dirnames,filenames in os.walk(directory_name):
        print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpaths}'")
```

```
train_dir='../tmp/MidTerm_Dataset/Train'
test_dir='../tmp/MidTerm_Dataset/Test'
```

```
walk_through_dir(train_dir)
```

```
There are 4 directories and 0 images in '../tmp/MidTerm_Dataset/Train'
There are 0 directories and 1523 images in '../tmp/MidTerm_Dataset/Train/washer'
There are 0 directories and 1523 images in '../tmp/MidTerm_Dataset/Train/locatingpin'
There are 0 directories and 1523 images in '../tmp/MidTerm_Dataset/Train/nut'
There are 0 directories and 1523 images in '../tmp/MidTerm_Dataset/Train/bolt'
```

```
walk_through_dir(test_dir)
```

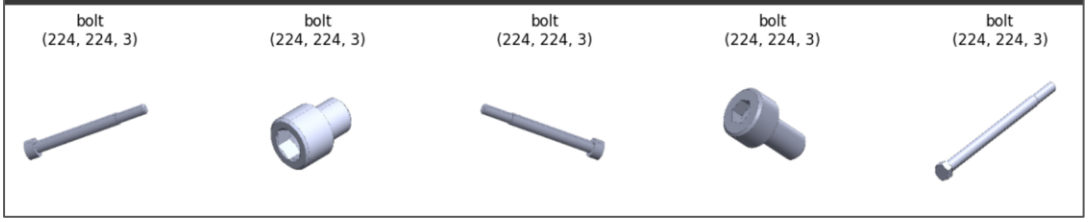
```
There are 4 directories and 0 images in '../tmp/MidTerm_Dataset/Test'
There are 0 directories and 381 images in '../tmp/MidTerm_Dataset/Test/washer'
There are 0 directories and 381 images in '../tmp/MidTerm_Dataset/Test/locatingpin'
There are 0 directories and 381 images in '../tmp/MidTerm_Dataset/Test/nut'
There are 0 directories and 381 images in '../tmp/MidTerm_Dataset/Test/bolt'
```

3. 視覺化資料集

隨機選取資料集中的圖片，除了了解圖片內容外也順便確認一下圖片有沒有問題。

```
#  從指定路徑下隨機顯示total_number張圖片
def  plot_random_image(target_dir,  total_number=5):

    #  隨機取得total_number張圖片
    target_class=random.choice(os.listdir(target_dir))
    target_folder=os.path.join(target_dir,  target_class)
    random_image=random.sample(os.listdir(target_folder),  5)

    #  顯示total_number張圖片
    plt.figure(figsize=(16,  5))
    row  =  int(total_number  **  (1/2))
    for  i  in  range(total_number):
        plt.subplot(row,  5,  i+1)
        img=tf.io.read_file(os.path.join(target_folder,  random_image[i]))

        img=tf.io.decode_image(img)
        plt.imshow(img)
        plt.title(f'{target_class}\n{img.shape}')
        plt.axis(False)
```

```
plot_random_image(train_dir)
```



4. 資料前處理

這部分我使用 tf.keras.preprocessing.image_dataset_from_directory 這項套件分別對訓練資料以及測試資料進行資料預處理。其中比較需要設定的是 label_mode 以及 shuffle，label_mode 要設定成”categorical”。另外將訓練集打亂，減少樣本之間的相關性，這樣做可以訓練的效果比較好，而測試集是提供評估模型效果的資料，並不會參與模型訓練的過程，因此不需要做打亂的動作。

```python
image_size = (224, 224) # 依照資料集中所有的圖片大小做設定

train_data=tf.keras.preprocessing.image_dataset_from_directory(
  directory=train_dir,
  label_mode="categorical",
  image_size=image_size,
  color_mode="rgb", # Default value: "rgb"
  batch_size=32,  # Default value: 32
  shuffle=True, # 打亂資料，減少樣本之間的相關性，可以獲得比較好的效果，預設為 True
)
class_names=train_data.class_names
num_classes=len(class_names)


test_data=tf.keras.preprocessing.image_dataset_from_directory(
  directory=test_dir,
  label_mode="categorical",
  image_size=image_size,
  color_mode="rgb", # Default value: "rgb"
  batch_size=32,  # Default value: 32
  shuffle=False, # 不打亂資料
)

```

5. 模型建立

這個部分主要參考助教提供的投影片建立 model，另外使用 BatchNormalization() 添加批量標準化層，用於加速訓練並提高模型效能。於全連接層，並使用 L2 正則化防止 Overfitting。

```python
input_shape = (224, 224, 3)

def create_model(input_shape, num_classes):

    model = keras.Sequential(
        [

            Conv2D(filters=32, kernel_size=(3, 3), activation="relu", input_shape=input_shape), # 卷積層，提取特徵
            BatchNormalization(), # 批量標準化層，用於加速訓練並提高模型性能
            MaxPooling2D(pool_size=(2, 2)), # 最大池化層，保留主要特徵

            Conv2D(filters=64, kernel_size=(3, 3), activation="relu"),
            BatchNormalization(),
            MaxPooling2D(pool_size=(2, 2)),

            Conv2D(filters=128, kernel_size=(3, 3), activation="relu"),
            BatchNormalization(),
            MaxPooling2D(pool_size=(2, 2)),

            Flatten(),    # 展平層，將卷積層的特徵圖展平為一維向量
            Dense(256,activation="relu", kernel_regularizer=l2(0.001)), # 全連接層，使用L2正則化以防止過擬合
            Dropout(0.5), # Dropout層，減少過擬合，隨機丟棄一定比例的神經元
            Dense(num_classes, activation='softmax')      # 輸出層，使用Softmax激活函數，對應各個類別的概率
        ]
    )

    return model

model=create_model(input_shape, num_classes)
model.summary()
```

使用 model.summary()查看模型架構。

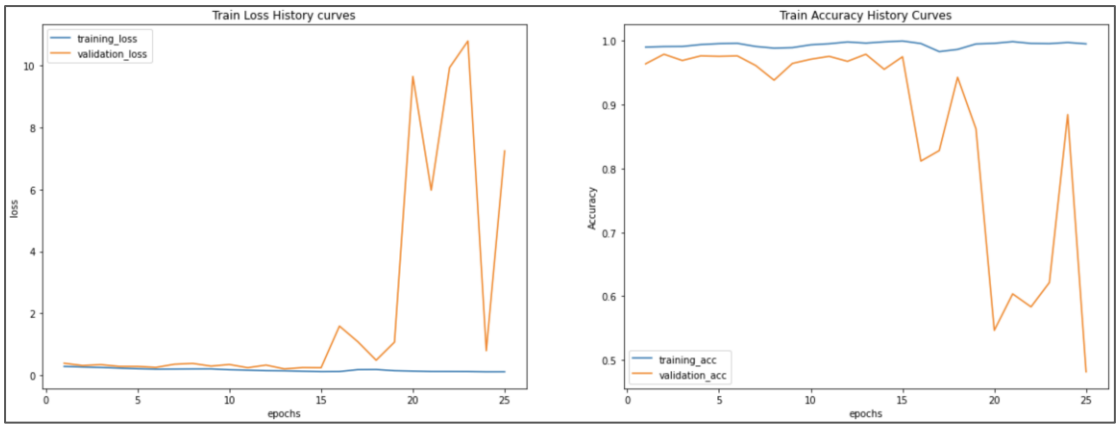
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 222, 222, 32)      896

batch_normalization_6 (Batc  (None, 222, 222, 32)      128
hNormalization)

max_pooling2d_6 (MaxPooling  (None, 111, 111, 32)      0
2D)

conv2d_7 (Conv2D)            (None, 109, 109, 64)      18496

batch_normalization_7 (Batc  (None, 109, 109, 64)      256
hNormalization)

max_pooling2d_7 (MaxPooling  (None, 54, 54, 64)        0
2D)

conv2d_8 (Conv2D)            (None, 52, 52, 128)       73856

batch_normalization_8 (Batc  (None, 52, 52, 128)       512
hNormalization)

max_pooling2d_8 (MaxPooling  (None, 26, 26, 128)       0
2D)

flatten_2 (Flatten)          (None, 86528)             0

dense_4 (Dense)              (None, 256)               22151424

dropout_2 (Dropout)          (None, 256)               0

dense_5 (Dense)              (None, 4)                 1028

=================================================================
Total params: 22,246,596
Trainable params: 22,246,148
Non-trainable params: 448
_________________________________________________________________
```

6. 調整參數 & 訓練過程

參數設定上 BATCH_SIZE 我一開始都是設為 32。EPOCHS 在一開始摸索模型時先設 5-10，大概看一下效果如何，還好 Colab GPU 加速跑起來蠻快的，所以很快就摸到自己蠻滿意的模型，Test Accuracy 來到 0.97 左右。

```
Test lose: 0.1755744218826294
Test Accuracy: 0.9737532734870911
```

但當我嘗試把 epochs 調高(>25)，想藉由訓練更多次來提升準確度，訓練次數來到 15 左右就開始出現 validation(test) loss 大暴增，準確率直接爆降



問了一下 ChatGPT(4.0)並查了一下資料。有許多可能都會導致 loss 大爆噴，如學習率過高、資料集中的雜訊或異常值、資料擴增過度、過擬合、隨機性。資料集以及資料預處裡的部分我想問題不大，而 Overfitting 的狀況實在非常明顯，因為在訓練資料集的表現很好，但驗證資料集的結果是慘不忍睹。處理 Overfitting 的方式有許多種，雖然降低 Epochs 可以得到比較好的效果，但我總覺得有更好的方式。

我主要從減少模型架構、降低學習率、新增早停回調下手。首先，先將模型減少一輪三層。接著將學習率(learning_rate)從原先的 0.0001 下降至 0.00001，並且使用自適應學習率優化器 Adam。另外添加早停回調，只要模型出現 overfitting 或是 loss 不降、accuracy 不升等模型沒有進步狀況就提前結束訓練。最後，可能因為學習速率比較低的關係，模型收斂速度比較慢，所以我將 BATCH_SIZE 改為 64，然後 EPOCHS 設 20，再次開始訓練!

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.00001)

# Compile model
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])


# 添加早停回調防止過擬合
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Hyperparameters
EPOCHS = 20
BATCH_SIZE = 64

history_of_model = model.fit(train_data,epochs=EPOCHS, batch_size=BATCH_SIZE, validation_data=test_data, validation_split=0.1, callbacks=[early_stopping])
```

這次確實有發生早停回調的情況，也得出了更好的模型。

```
Epoch 16/20
191/191 [==============================] - 16s 82ms/step - loss: 0.0931 - accuracy: 0.9997 - val_loss: 0.1903 - val_accuracy: 0.9797
Epoch 17/20
191/191 [==============================] - 16s 85ms/step - loss: 0.0901 - accuracy: 1.0000 - val_loss: 0.1799 - val_accuracy: 0.9803
Epoch 18/20
191/191 [==============================] - 18s 92ms/step - loss: 0.0881 - accuracy: 1.0000 - val_loss: 0.1834 - val_accuracy: 0.9797
Epoch 19/20
191/191 [==============================] - 20s 102ms/step - loss: 0.0872 - accuracy: 0.9998 - val_loss: 0.1899 - val_accuracy: 0.9770
```

7. 評估模型

評估模型跟模型建立以集調整參數的部分是一直來回的，調了好久終於練出滿意的結果~這邊使用 model.evaluate()去評估模型的成效。雖然比起開頭 0.97 只有多 1%的準確率，但尋找提升準確率以及降低 loss 方法的這個學習過程，讓

我覺得這 1%還是有滿滿的成就感。不只機器在學習，我也在學習。

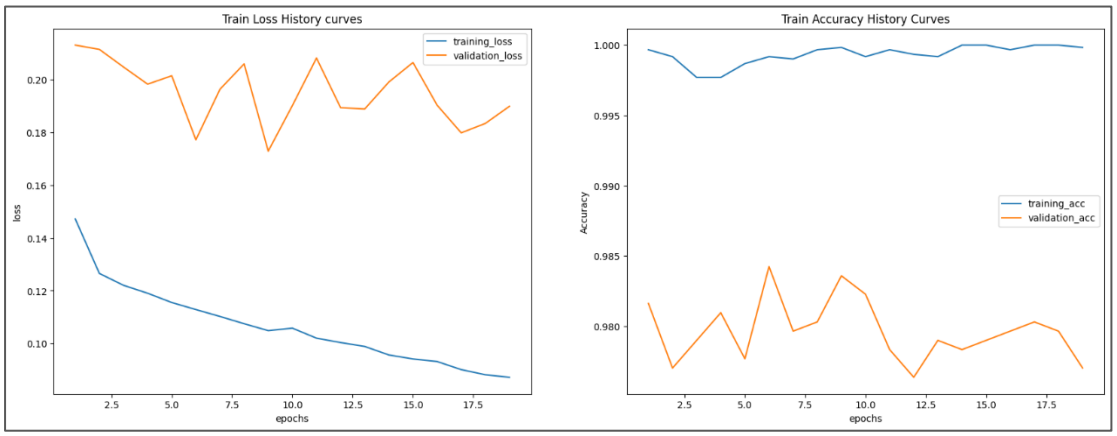
```
score=model.evaluate(test_data,  verbose=0)
print('Test  lose:',  score[0])
print('Test  Accuracy:',  score[1])
```

```
Test lose: 0.17291387915611267
Test Accuracy: 0.9835957884788513
```

撰寫 function 顯示訓練過程的優化曲線

```
def  plot_loss_curves(history):
    train_loss=history.history['loss']
    val_loss=history.history['val_loss']

    train_accuracy=history.history['accuracy']
    val_accuracy=history.history['val_accuracy']

    epochs=range(1,len(history.history['loss'])+1)
    plt.figure(figsize=(20,7))
    #  plot  loss  data
    plt.subplot(1,2,1)
    plt.plot(epochs,train_loss,label="training_loss")
    plt.plot(epochs,val_loss,label="validation_loss")
    plt.title("Train  Loss  History  curves")
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.legend()
    #  plt.show()

    #  plot  accuracy  data
    plt.subplot(1,2,2)
    plt.plot(epochs,train_accuracy,label="training_acc")
    plt.plot(epochs,val_accuracy,label="validation_acc")
    plt.title("Train  Accuracy  History  Curves")
    plt.xlabel('epochs')
    plt.ylabel('Accuracy')
    plt.legend()
```

```
plot_loss_curves(history_of_model)
```



8. 使用模型進行預測

雖然模型表現看起來還不錯，但只有看到數字跟曲線還是不太夠。所以就撰寫

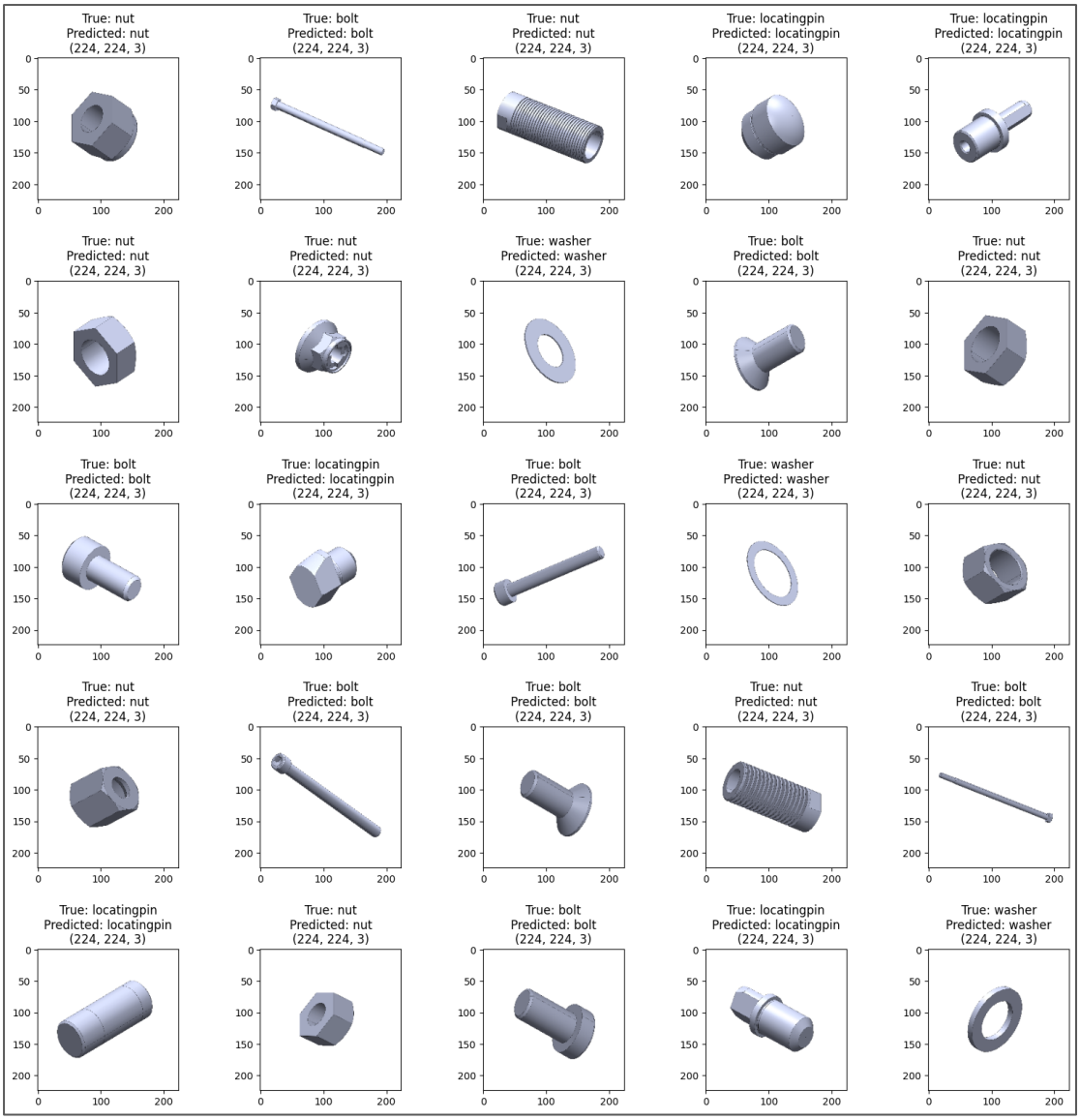
了一個 function 實際使用模型進行預測並顯示比對結果!

```python
def predict_random_mechanical_parts(target_dir, model, total_number=25):
    # 隨機取得total_number張圖片
    random_images = []
    for _ in range(total_number):
        target_class = random.choice(os.listdir(target_dir))
        target_folder = os.path.join(target_dir, target_class)
        random_image = random.choice(os.listdir(target_folder))
        random_images.append((target_class, random_image))

    # 顯示total_number圖片
    plt.figure(figsize=(16, 16))
    row = int(total_number ** (1/2))
    count_correct = 0
    for i, (target_class, random_image) in enumerate(random_images):
        plt.subplot(row, 5, i + 1)
        img = tf.io.read_file(os.path.join(target_dir, target_class, random_image))

        img = tf.io.decode_image(img)
        plt.imshow(img)

        # 對圖片進行預處理，使其適合輸入到模型中
        preprocessed_img = tf.expand_dims(img, axis=0)

        # 進行預測
        pred = model.predict(preprocessed_img)[0]    # 回傳針對不同class的預測值
        predicted_class_index = np.argmax(pred)
        predicted_class_name = class_names[predicted_class_index]
        if target_class == predicted_class_name:
            count_correct = count_correct + 1
        plt.title(f'True: {target_class}\nPredicted: {predicted_class_name}\n{img.shape}')

    plt.tight_layout(pad=2.0)
    print('Prediction Accuracy: ' + str(round((count_correct / total_number * 100), 2)) + '%')
```

效果看起來很不錯，完工!

```python
predict_random_mechanical_parts(target_dir=test_dir, model=model)
```

```
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 22ms/step
Prediction Accuracy: 100.0%
```


True: nut
Predicted: nut
(224, 224, 3)
True: bolt
Predicted: bolt
(224, 224, 3)
True: nut
Predicted: nut
(224, 224, 3)
True: locatingpin
Predicted: locatingpin
(224, 224, 3)
True: locatingpin
Predicted: locatingpin
(224, 224, 3)
True: nut
Predicted: nut
(224, 224, 3)
True: nut
Predicted: nut
(224, 224, 3)
True: washer
Predicted: washer
(224, 224, 3)
True: bolt
Predicted: bolt
(224, 224, 3)
True: nut
Predicted: nut
(224, 224, 3)
True: bolt
Predicted: bolt
(224, 224, 3)
True: locatingpin
Predicted: locatingpin
(224, 224, 3)
True: bolt
Predicted: bolt
(224, 224, 3)
True: washer
Predicted: washer
(224, 224, 3)
True: nut
Predicted: nut
(224, 224, 3)
True: nut
Predicted: nut
(224, 224, 3)
True: bolt
Predicted: bolt
(224, 224, 3)
True: bolt
Predicted: bolt
(224, 224, 3)
True: nut
Predicted: nut
(224, 224, 3)
True: bolt
Predicted: bolt
(224, 224, 3)
True: locatingpin
Predicted: locatingpin
(224, 224, 3)
True: nut
Predicted: nut
(224, 224, 3)
True: bolt
Predicted: bolt
(224, 224, 3)
True: locatingpin
Predicted: locatingpin
(224, 224, 3)
True: washer
Predicted: washer
(224, 224, 3)

三、 參考資料

1. co-lab 上傳文件 - csdn
2. Keras API reference
3. Image data preprocessing