作業 1 - ARTCNN

資管三_109403502_楊珮綾

colab 連結： 3
Test accuracy: 3
撰寫過程與截圖: 3
一、 資料集下載(未更動) 3
二、 讀入封包(未更動) 4
三、 取得資料集 4
四、 資料前處理 7
五、 建立模型 12
六、 制定訓練計畫 13
七、 評估模型(未更動) 13
八、 做預測 15
心得: 18

- **colab 連結：**
  https://colab.research.google.com/drive/1VF8IO8PIsijVAuIN7Ke_oa9ih1iPbRln?usp=sharing

- **Test accuracy:**

  Test accuracy: **0.4323353171348572**

```
7/7 [==============================] - 2s 25ms/step - loss: 2.1921 - accuracy: 0.4323
Test loss: 2.1921138763427734
Test accuracy: 0.4323353171348572
```

- **撰寫過程與截圖:**

  一、資料集下載(未更動)

```
import random
import os

# 大家盡量先把資料保存在本地端，然後要訓練時用本地端上傳做訓練
# 以節省學術網路資源，避免 IP 被封鎖

if not os.path.isfile("./train.zip"):
        !wget -O train.zip "http://140.115.83.111/files/art/train.zip"
        !wget -O test.zip "http://140.115.83.111/files/art/test.zip"
        !unzip train.zip
        !unzip test.zip
else:
        !echo "檔案已存在"
```

```
串流輸出內容已截斷至最後 5000 行。
  inflating: train_resized/Joan_Miro_27.jpg
  inflating: train_resized/Joan_Miro_28.jpg
  inflating: train_resized/Joan_Miro_29.jpg
  inflating: train_resized/Joan_Miro_3.jpg
  inflating: train_resized/Joan_Miro_30.jpg
  inflating: train_resized/Joan_Miro_31.jpg
  inflating: train_resized/Joan_Miro_32.jpg
  inflating: train_resized/Joan_Miro_33.jpg
  inflating: train_resized/Joan_Miro_34.jpg
  inflating: train_resized/Joan_Miro_35.jpg
  inflating: train_resized/Joan_Miro_36.jpg
  inflating: train_resized/Joan_Miro_37.jpg
  inflating: train_resized/Joan_Miro_39.jpg
  inflating: train_resized/Joan_Miro_4.jpg
  inflating: train_resized/Joan_Miro_40.jpg
  inflating: train_resized/Joan_Miro_42.jpg
  inflating: train_resized/Joan_Miro_43.jpg
  inflating: train_resized/Joan_Miro_44.jpg
  inflating: train_resized/Joan_Miro_45.jpg
  inflating: train_resized/Joan_Miro_46.jpg
  inflating: train_resized/Joan_Miro_47.jpg
  inflating: train_resized/Joan_Miro_48.jpg
  inflating: train_resized/Joan_Miro_49.jpg
  inflating: train_resized/Joan_Miro_5.jpg
  inflating: train_resized/Joan_Miro_50.jpg
  inflating: train_resized/Joan_Miro_51.jpg
  inflating: train_resized/Joan_Miro_52.jpg
  inflating: train_resized/Joan_Miro_55.jpg
```

二、讀入封包(未更動)

```
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
```

三、取得資料集

1. 檢視 artist.csv (未更動)

```
train_dir = "./train_resized/"
test_dir = "./test_resized/"
artists = pd.read_csv("./artists.csv")
num_classes = artists.shape[0]    #回傳行數
print("Number of artists : ", num_classes)
artists.head()    #前五筆資料
```

```
Number of artists :  50
```

	id	name	years	genre	nationality	bio	wikipedia	paintings
0	0	Amedeo Modigliani	1884 - 1920	Expressionism	Italian	Amedeo Clemente Modigliani (Italian pronunciat...	http://en.wikipedia.org/wiki/Amedeo_Modigliani	193
1	1	Vasiliy Kandinskiy	1866 - 1944	Expressionism,Abstractionism	Russian	Wassily Wassilyevich Kandinsky (Russian: ?a?и?...	http://en.wikipedia.org/wiki/Wassily_Kandinsky	88
2	2	Diego Rivera	1886 - 1957	Social Realism,Muralism	Mexican	Diego María de la Concepción Juan Nepomuceno E...	http://en.wikipedia.org/wiki/Diego_Rivera	70
3	3	Claude Monet	1840 - 1926	Impressionism	French	Oscar-Claude Monet (; French: [klod m?n?]; 14 ...	http://en.wikipedia.org/wiki/Claude_Monet	73
4	4	Rene Magritte	1898 - 1967	Surrealism,Impressionism	Belgian	René François Ghislain Magritte (French: [??ne...	http://en.wikipedia.org/wiki/René_Magritte	194

2. 只取出名字與畫的數量，把名字用下底線連起來(未更動)

```
artists = artists.loc[:, ["name", "paintings"]]    #用index的標籤來取出資料
artists["name"] = artists["name"].str.split(" ").apply(lambda parts: "_".join(parts))
artists.head()
```

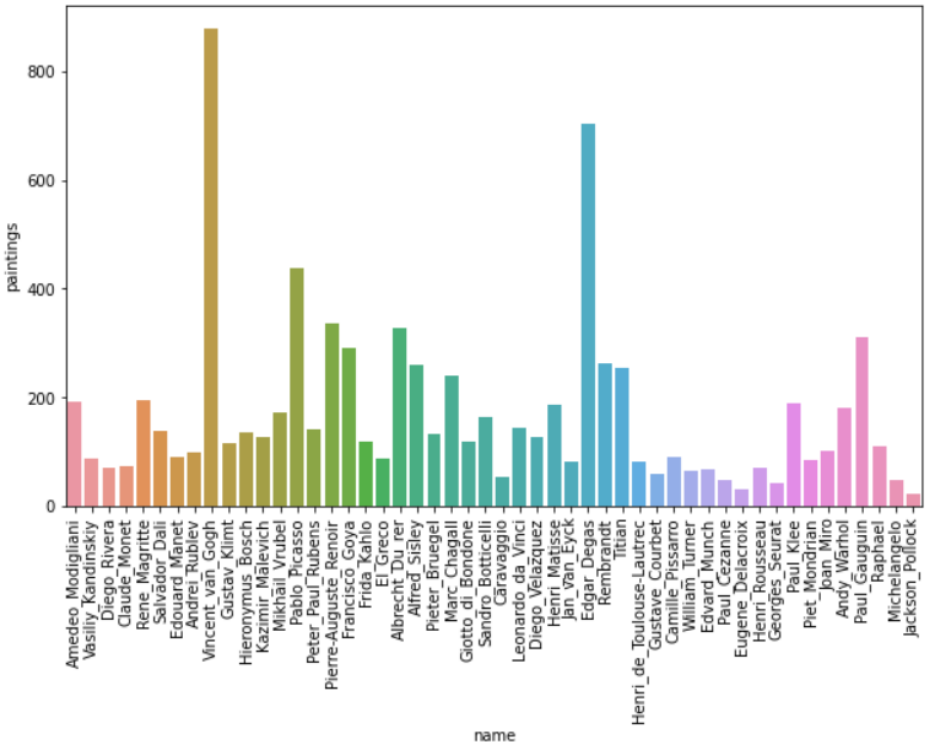
	name	paintings
0	Amedeo_Modigliani	193
1	Vasiliy_Kandinskiy	88
2	Diego_Rivera	70
3	Claude_Monet	73
4	Rene_Magritte	194

3. 計算各個畫家畫作數量

在 barplot = sns.barplot(x=artists.name, y=artists.paintings)加上 x、y，才不會有 error

```
%matplotlib inline
plt.figure(figsize=(10, 6)) #a為圖形的寬， b為圖形的高，單位為英寸
barplot = sns.barplot(x=artists.name, y=artists.paintings)
for item in barplot.get_xticklabels():
        item.set_rotation(90)

print("可以看到每個畫家之間的畫作數量很不平均，這會影響到模型的訓練。")
print("最多畫作為 : ", artists.paintings.max(), " 最少畫作為 : ", artists.paintings.min())
```

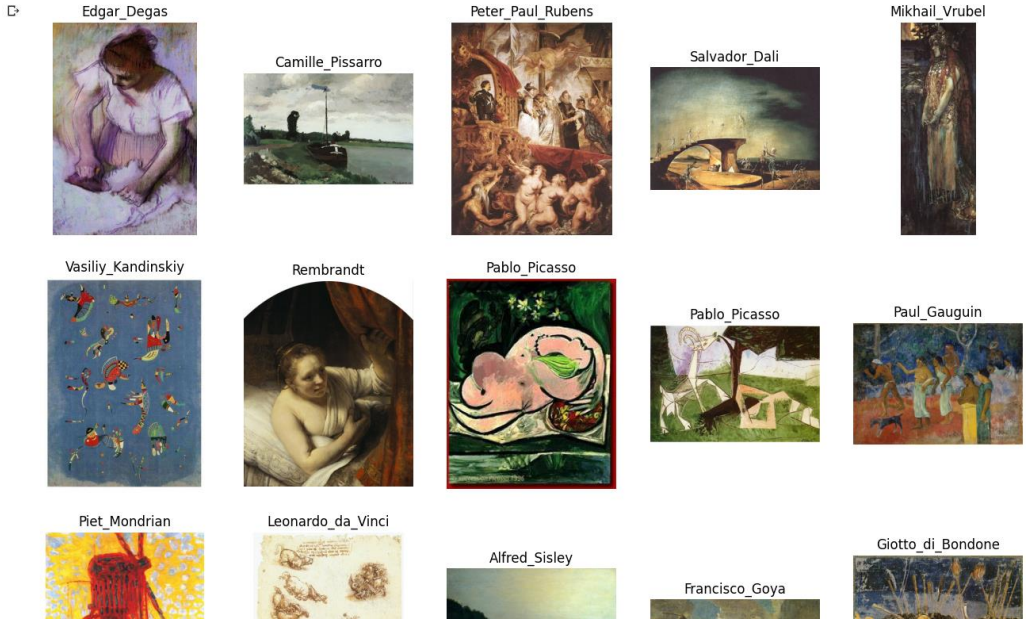
```
可以看到每個畫家之間的畫作數量很不平均，這會影響到模型的訓練。
最多畫作為 : 877 最少畫作為 : 24
```



4. 隨機讀取畫作來看看(未更動)

```
img_list = os.listdir(train_dir) #返回指定的文件夾包含的文件或文件夾的名字的列表
total_len = len(img_list)
random_list = random.sample(range(0, total_len), 20)
print("training 畫作總共畫作有 : ", total_len)

show_imgs = [img_list[rand] for rand in random_list]

plt.figure(figsize=(16, 16))
for index, imgName in enumerate(show_imgs): #enumerate() 函式來同時輸出索引與元素
        img_path = train_dir + imgName
        img = cv.imread(img_path) #讀取圖片檔
        img = cv.cvtColor(img, cv.COLOR_BGR2RGB) #將影像的色彩模型從將BGR格式轉換成RGB格式
        plt.subplot(4, 5, index + 1)
        plt.imshow(img)
        plt.axis("off")
        plt.title("_".join(imgName.split("_")[:-1]))
```

```
training 畫作總共畫作有 : 7520
```



四、資料前處理

1. make_author_dict()：建立將英文映射成數字的 dict。
   將 csv 中 id、name 欄位的資料叫出，並且 name 加上底線，再將 id、name 放入 dict 中。(dict 結構為[name:id])

2. rev_author_dict()：建立將數字映射成英文的 dict。
   將 csv 中 id、name 欄位的資料叫出，並且 name 加上底線，再將 id、name 放入 dict 中。(dict 結構為[id:name])

```
# 請建立將英文映射成數字的 dict。EX: Van_Gogh --> 0
def make_author_dict():
        ##########
        # todo #
        ##########
        author_dict={}
        artists = pd.read_csv("./artists.csv")
        id_data = list(artists.loc[:, 'id'])
        print(id_data)
        artists.loc[:, 'name']
        name_data = list(artists["name"].str.split(" ").apply(lambda parts: "_".join(parts)))
        print(name_data)
        for i in range(len(id_data)):
            author_dict[name_data[i]]=id_data[i]
        print(author_dict)
        return author_dict



class_name = make_author_dict()
def rev_author_dict():
        ##########
        # todo #
        ##########
        re_author_dict={}
        artists = pd.read_csv("./artists.csv")
        id_data = list(artists.loc[:, 'id'])
        artists.loc[:, 'name']
        name_data = list(artists["name"].str.split(" ").apply(lambda parts: "_".join(parts)))
        for i in range(len(id_data)):
            re_author_dict[id_data[i]]=name_data[i]
        print(re_author_dict)
        return re_author_dict
# 請建立將數字映射成英文的 dict。 EX: 0 --> Van_Gogh
rev_class_name = rev_author_dict()
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
['Amedeo_Modigliani', 'Vasiliy_Kandinskiy', 'Diego_Rivera', 'Claude_Monet', 'Rene_Magritte', 'Salvador_Dali', 'Edouard_Manet', 'Andrei_Rublev', 'Vincent_van_Gogh'
{'Amedeo_Modigliani': 0, 'Vasiliy_Kandinskiy': 1, 'Diego_Rivera': 2, 'Claude_Monet': 3, 'Rene_Magritte': 4, 'Salvador_Dali': 5, 'Edouard_Manet': 6, 'Andrei_Rublev
{0: 'Amedeo_Modigliani', 1: 'Vasiliy_Kandinskiy', 2: 'Diego_Rivera', 3: 'Claude_Monet', 4: 'Rene_Magritte', 5: 'Salvador_Dali', 6: 'Edouard_Manet', 7: 'Andrei_Rub
```

3. get_label()：取出 label 並轉成數字
   先把放進的 pic_name 加上底線處理，再從 author_dict 叫對應的數字
4. get_path()：將路徑合併
   把 dir 加上 pic_name 即是路徑
5. make_paths_label()：將 preprocess 完成的 path、label 用 for 迴圈放入 paths 和 labels，再將 labels 轉成 onehot
   先把檔案丟進 get_label()、get_path()，再透過 for 放入個別陣列。
   透過 keras.utils.to_categorical(labels,num_classes)把 labels 陣列 轉成 onehot 陣列

```
def get_label(pic_name):
        # 請取出 label 並轉成數字
        # EX: Claude_Monet_1.jpg -> Claude_Monet -> 1
        ###########
        # todo #
        ###########
        pic_name="_".join(pic_name.split("_")[:-1])
        imgName_label=class_name[pic_name]
        return imgName_label

#print(get_label("Claude_Monet_1_1.jpg"))

def get_path(dir, pic_name):
        # 請將路徑合併
        # EX: ./train_resized/ + Claude_Monet_1.jpg => ./train_resized/Claude_Monet_1.jpg
        ###########
        # todo #
        ###########
        imgPath=dir + pic_name
        return imgPath
#print(get_path("./train_resized/","Claude_Monet_1.jpg"))

def make_paths_label(dir):
        img_list = os.listdir(dir)
        paths = []
        labels = []
        onehot_labels = []
        # 將preprocess完成的 path、label 用 for 迴圈放入 paths 和 labels
        ###########
        # todo #
        ###########
        for i in range(len(img_list)):
            labels.append(get_label(img_list[i]))
            paths.append(get_path(dir,img_list[i]))
        # 將 labels 轉成 onehot
        # todo
        train_label = keras.utils.to_categorical(labels,num_classes)
        onehot_labels=train_label
        return paths , onehot_labels

#make_paths_label("./train_resized/")
```

6. 查看一下結果

加入 np.set_printoptions(threshold=np.inf)，讓 onehot_labels 輸出時能夠完整呈現。

```
# 來查看一下
np.set_printoptions(threshold=np.inf)

paths, onehot_labels = make_paths_label(train_dir)

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

```
paths :
./train_resized/Sandro_Botticelli_154.jpg
./train_resized/Vincent_van_Gogh_695.jpg
./train_resized/Kazimir_Malevich_45.jpg
./train_resized/Edouard_Manet_86.jpg
./train_resized/Kazimir_Malevich_69.jpg
--------------------
labels :
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
```

```
# 轉成 tensorflow dataset 格式，變成路徑 tensor
# 這個只是 from_tensor_slices 範例
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)
```

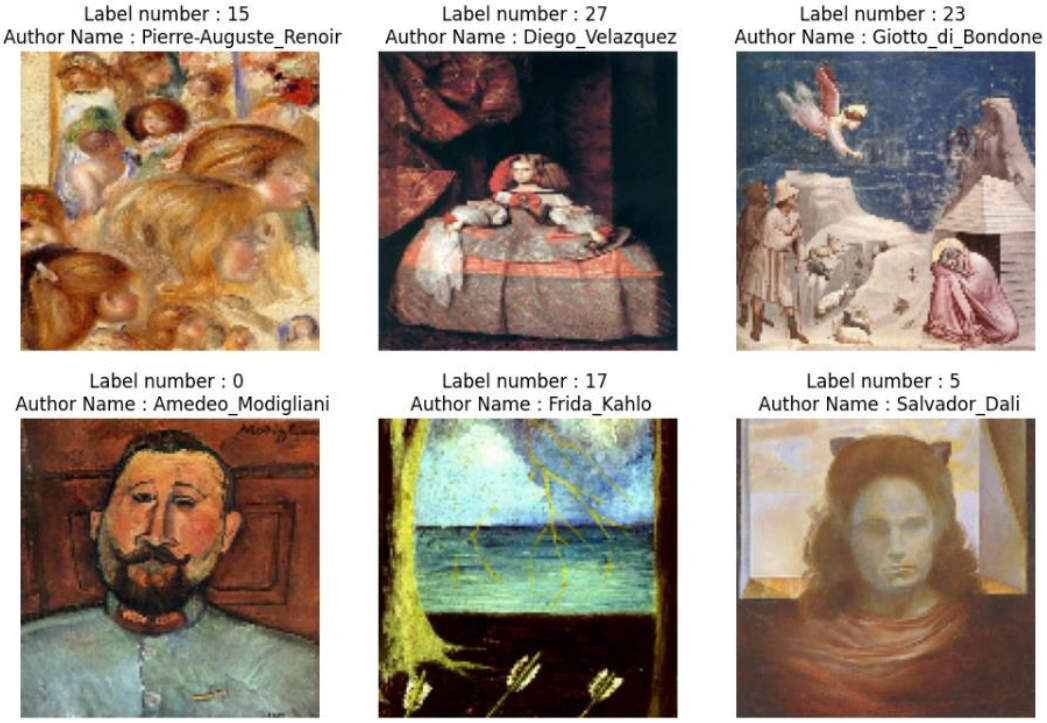
```
turn to tensor
tf.Tensor(b'./train_resized/Sandro_Botticelli_154.jpg', shape=(), dtype=string)
tf.Tensor(b'./train_resized/Vincent_van_Gogh_695.jpg', shape=(), dtype=string)
tf.Tensor(b'./train_resized/Kazimir_Malevich_45.jpg', shape=(), dtype=string)
tf.Tensor(b'./train_resized/Edouard_Manet_86.jpg', shape=(), dtype=string)
tf.Tensor(b'./train_resized/Kazimir_Malevich_69.jpg', shape=(), dtype=string)
```

7. 統一圖片大小
8. 把每張圖片正規化，映射到 [0,1] 之間
9. 把資料集打散(shuffle) (未更動)

```python
# 決定你輸入模型的圖片長寬
IMG_WIDTH = 128
IMG_HEIGHT = 128
IMG_SIZE = (IMG_WIDTH, IMG_HEIGHT)
# shuffle buffer size
SHUFFLE_BUFFER = 1000


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)

    # 請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
    ##########
    # todo #
    ##########
    # resize the image to IMG_HEIGHT x IMG_WIDTH
    img = tf.image.resize(img, [IMG_HEIGHT, IMG_WIDTH])

    # normalize pixel values to [0, 1]
    img /= 255.0
    return img
#plt.imshow(get_image("./train_resized/Albrecht_Du_rer_3.jpg"))

# 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=False)
    return full_ds


full_ds = make_dataset(train_dir)
```

10. 取出 Tensor 圖片 (未更動)

```
# 取出 Tensor 圖片來看看
plt.figure(figsize=(12, 8))
for index, (img, label) in enumerate(full_ds.take(6)):
        l = np.argmax(label.numpy())
        plt.subplot(2, 3, index + 1)
        plt.imshow(img)
        plt.title("Label number : {} \n Author Name : {}".format(l, rev_class_name[l]))
        plt.axis("off")
```



11. 拆分成 training_data、validation_data（未更動）
12. 添加 batch：設定為 128

```
# 切割成 training data 與 validation data
train_len = int(0.8 * total_len)
val_len = total_len - 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的大小通常選擇在32-256之間
# todo
BATCH_SIZE =128   #每次訓練時會使用128筆資料進行梯度下降

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

```
train size :  6016  val size :  1504
```

13. 查看添加後 batch 的維度

```
# 查看添加batch後的維度
trainiter = iter(train_ds) #將 train_ds 資料集轉換成 iterator 物件，方便取出其中的資料。
x, y = trainiter.next()
print("training image batch shape : ", x.shape) #(batch_size, image_width, image_height, channels)
print("training label batch shape : ", y.shape) #(batch_size, num_classes)
```

```
training image batch shape :  (128, 128, 128, 3)
training label batch shape :  (128, 50)
```

五、建立模型

經過多次測試發現 batch 為 128、照片大小為 128X128 時，做這樣準確率較高。

```
input_shape = (128,128,3)
# 自訂你的 model
##########
# todo #
##########
model = keras.Sequential([
        keras.Input(shape=input_shape),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
                layers.MaxPooling2D(pool_size=(2, 2)),
                layers.MaxPooling2D(pool_size=(2, 2)),
                layers.MaxPooling2D(pool_size=(2, 2)),
                layers.Dropout(0.2),
                layers.Conv2D(128, kernel_size=(3, 3), activation="relu"),
                layers.MaxPooling2D(pool_size=(2, 2)),
                layers.MaxPooling2D(pool_size=(2, 2)),
                layers.Dropout(0.2),
                layers.Flatten(),
                layers.Dense(128, activation="relu"),
                layers.Dense(128, activation="relu"),
                layers.Dropout(0.5),
                layers.Dense(num_classes, activation="softmax"),
])


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 64)      1792

 max_pooling2d (MaxPooling2D  (None, 63, 63, 64)       0
 )

 max_pooling2d_1 (MaxPooling  (None, 31, 31, 64)       0
 2D)

 max_pooling2d_2 (MaxPooling  (None, 15, 15, 64)       0
 2D)

 dropout (Dropout)           (None, 15, 15, 64)        0

 conv2d_1 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 max_pooling2d_4 (MaxPooling  (None, 3, 3, 128)        0
 2D)

 dropout_1 (Dropout)         (None, 3, 3, 128)         0

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 128)               147584

 dense_1 (Dense)             (None, 128)               16512

 dropout_2 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 50)                6450

=================================================================
Total params: 246,194
Trainable params: 246,194
Non-trainable params: 0
_________________________________________________________________
```

六、制定訓練計畫

設定訓練 27 次

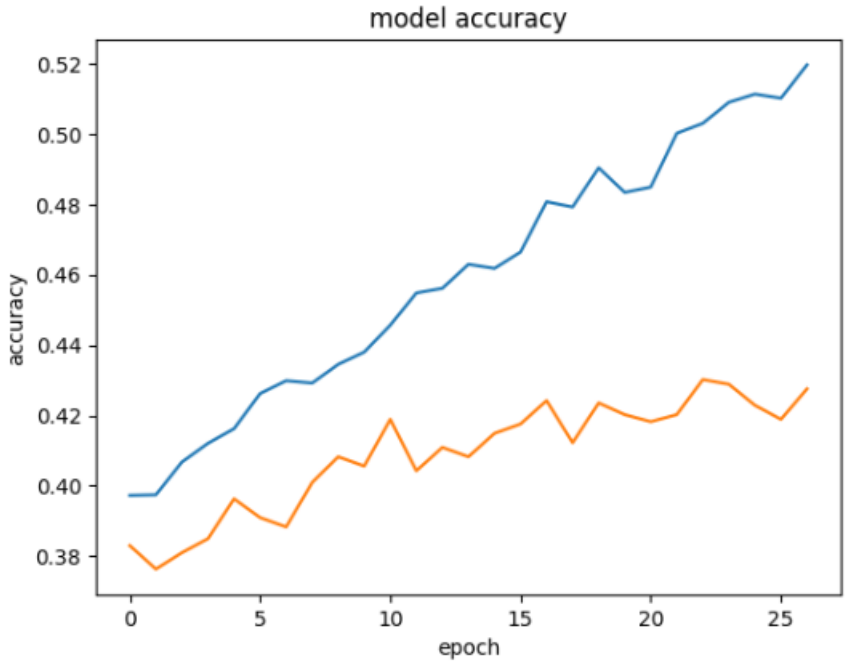
```
# todo
batch_size = 128
EPOCHS = 27

###########
# 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)
```

```
Epoch 1/27
47/47 [==============================] - 66s 1s/step - loss: 2.1994 - accuracy: 0.3973 - val_loss: 2.3305 - val_accuracy: 0.3830
Epoch 2/27
47/47 [==============================] - 43s 858ms/step - loss: 2.1387 - accuracy: 0.3974 - val_loss: 2.3325 - val_accuracy: 0.3763
Epoch 3/27
47/47 [==============================] - 42s 853ms/step - loss: 2.1198 - accuracy: 0.4067 - val_loss: 2.3274 - val_accuracy: 0.3810
Epoch 4/27
47/47 [==============================] - 42s 851ms/step - loss: 2.1074 - accuracy: 0.4121 - val_loss: 2.3288 - val_accuracy: 0.3850
Epoch 5/27
47/47 [==============================] - 42s 856ms/step - loss: 2.0794 - accuracy: 0.4162 - val_loss: 2.2772 - val_accuracy: 0.3963
Epoch 6/27
47/47 [==============================] - 42s 850ms/step - loss: 2.0494 - accuracy: 0.4262 - val_loss: 2.2954 - val_accuracy: 0.3910
Epoch 7/27
47/47 [==============================] - 44s 860ms/step - loss: 2.0393 - accuracy: 0.4299 - val_loss: 2.2657 - val_accuracy: 0.3883
Epoch 8/27
47/47 [==============================] - 44s 883ms/step - loss: 2.0234 - accuracy: 0.4292 - val_loss: 2.2541 - val_accuracy: 0.4009
Epoch 9/27
47/47 [==============================] - 42s 841ms/step - loss: 1.9987 - accuracy: 0.4345 - val_loss: 2.2702 - val_accuracy: 0.4082
Epoch 10/27
47/47 [==============================] - 43s 866ms/step - loss: 1.9949 - accuracy: 0.4380 - val_loss: 2.2033 - val_accuracy: 0.4056
Epoch 11/27
47/47 [==============================] - 42s 846ms/step - loss: 1.9590 - accuracy: 0.4456 - val_loss: 2.1600 - val_accuracy: 0.4189
Epoch 12/27
47/47 [==============================] - 42s 856ms/step - loss: 1.9314 - accuracy: 0.4548 - val_loss: 2.2332 - val_accuracy: 0.4043
Epoch 13/27
47/47 [==============================] - 63s 1s/step - loss: 1.9018 - accuracy: 0.4561 - val_loss: 2.1886 - val_accuracy: 0.4109
Epoch 14/27
47/47 [==============================] - 43s 836ms/step - loss: 1.9049 - accuracy: 0.4629 - val_loss: 2.2018 - val_accuracy: 0.4082
Epoch 15/27
47/47 [==============================] - 41s 836ms/step - loss: 1.8910 - accuracy: 0.4618 - val_loss: 2.1889 - val_accuracy: 0.4149
Epoch 16/27
47/47 [==============================] - 42s 847ms/step - loss: 1.8829 - accuracy: 0.4664 - val_loss: 2.1307 - val_accuracy: 0.4176
Epoch 17/27
47/47 [==============================] - 42s 849ms/step - loss: 1.8289 - accuracy: 0.4807 - val_loss: 2.1488 - val_accuracy: 0.4242
Epoch 18/27
47/47 [==============================] - 42s 838ms/step - loss: 1.8047 - accuracy: 0.4792 - val_loss: 2.1464 - val_accuracy: 0.4122
Epoch 19/27
47/47 [==============================] - 43s 845ms/step - loss: 1.7864 - accuracy: 0.4904 - val_loss: 2.1287 - val_accuracy: 0.4235
Epoch 20/27
47/47 [==============================] - 42s 843ms/step - loss: 1.7922 - accuracy: 0.4834 - val_loss: 2.1507 - val_accuracy: 0.4202
Epoch 21/27
47/47 [==============================] - 43s 865ms/step - loss: 1.7968 - accuracy: 0.4849 - val_loss: 2.1420 - val_accuracy: 0.4182
Epoch 22/27
47/47 [==============================] - 42s 843ms/step - loss: 1.7452 - accuracy: 0.5002 - val_loss: 2.1563 - val_accuracy: 0.4202
Epoch 23/27
47/47 [==============================] - 43s 839ms/step - loss: 1.7342 - accuracy: 0.5030 - val_loss: 2.1597 - val_accuracy: 0.4302
Epoch 24/27
47/47 [==============================] - 43s 860ms/step - loss: 1.7166 - accuracy: 0.5090 - val_loss: 2.1351 - val_accuracy: 0.4289
```

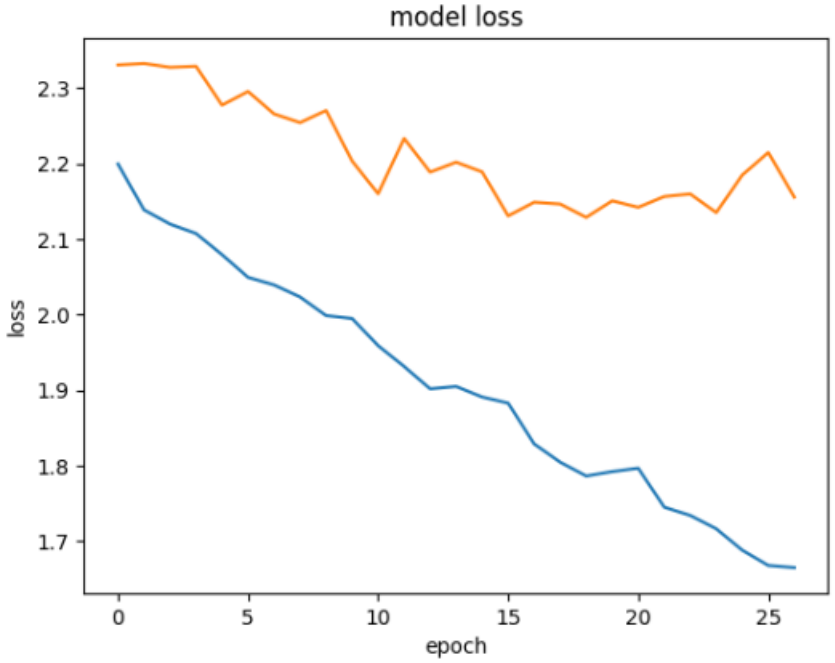
七、評估模型(未更動)

```
print(history.history.keys())

plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
plt.title("model  accuracy")
plt.ylabel("accuracy")
plt.xlabel("epoch")
plt.show()
```

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




```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("model  loss")
plt.ylabel("loss")
plt.xlabel("epoch")

plt.show()
```



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

```
7/7 [==============================] - 2s 25ms/step - loss: 2.1921 - accuracy: 0.4323
Test loss: 2.1921138763427734
Test accuracy: 0.4323353171348572
```

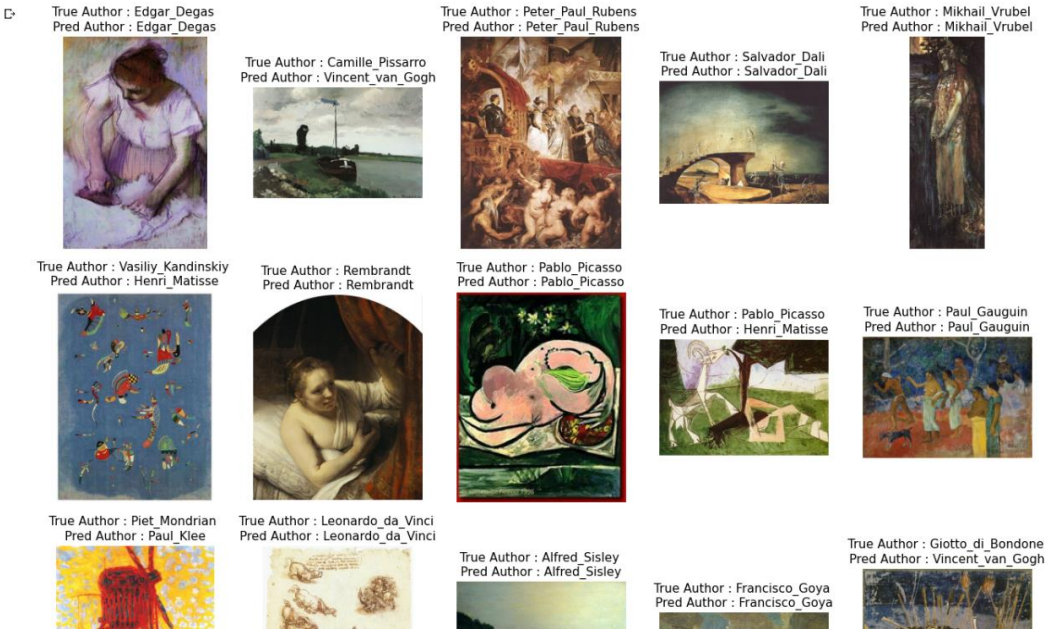
八、做預測

1. 把前面讀取的圖片拿來丟入模型做預測

np.expand_dims(img, 0)：expand img dimension
model.predict(img)：丟入模型
np.argmax(predictions[0])：取出 softmax 後 (50,) 取最大值的 index 作為辨識結果

```python
def predict_author(img):
    # 寫個單圖片模型預測 function
    # input : opencv img (height,width,3)
    # output : 某個作家名字 E.g. Claude_Monet
    #
    # 參考步驟:
    # 1. expand img dimension (height,width,3) -> (1,height,width,3)
    # 2. 丟入模型 model.predict
    # 3. 取出 softmax 後 (50,) 取最大值的 index 作為辨識結果
    # 4. 將辨識結果轉為畫作家名字

    author_name = ""
    ##########
    # todo #
    ##########

    # expand img dimension (height,width,3) -> (1,height,width,3)
    img = np.expand_dims(img, 0)
    # 丟入模型 model.predict
    predictions = model.predict(img)
    # 取出 softmax 後 (50,) 取最大值的 index 作為辨識結果
    predicted_index = np.argmax(predictions[0])

    # 將辨識結果轉為畫作家名字
    author_name = rev_class_name[predicted_index]
    return author_name
```

```
plt.figure(figsize=(16, 16))
for index, imgName in enumerate(show_imgs):
        img_path = train_dir + imgName
        img = cv.imread(img_path)
        img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        plt.subplot(4, 5, index + 1)
        plt.axis("off")
        plt.imshow(img)
        img = cv.resize(img, (IMG_WIDTH, IMG_HEIGHT))
        img = img / 255.0
        plt.title(
                "True Author : {} \nPred Author : {}".format(
                        "_".join(imgName.split("_")[:-1]), predict_author(img)
                ),
                size=11,
        )
```



2. 用自己的照片試試

Predict author : Edgar_Degas

```
from google.colab import files


def upload_img():
        uploaded = files.upload()
        img_name = list(uploaded.keys())[0]
        img = cv.imread(img_name)
        img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        plt.imshow(img)
        img = cv.resize(img, (IMG_WIDTH, IMG_HEIGHT))
        img = img / 255.0
        return img


def eval():
        img = upload_img()
        plt.title("predict author : {}".format(predict_author(img)))
        plt.axis("off")
        plt.show()
```

```
# 自己上傳一張圖片來試試看
# Demo 圖片來自:
# Interview with Cyberpunk 2077 "ponpon shit" producer Yuki Kawamura (https://block.fm/news/cyberpunk2077_uscracks_ENG)
eval()
```

```
選擇檔案 測試照片.jpg
• 測試照片.jpg(image/jpeg) - 92179 bytes, last modified: 2023/4/10 - 100% done
Saving 測試照片.jpg to 測試照片 (1).jpg
1/1 [==============================] - 0s 19ms/step
```

- **心得:**

  這是我第一次做這樣的深度學習這樣的作業，由於我自己也不會 python，所以在開始動手 coding 之前，就先花了許多時間把範例的程式碼逐行查詢、寫註解，先了解大概在幹嘛再開始下手，雖然做到後面還是有點矇。

  大致上測試會影響準確率的訓練模型、各項數值(照片長寬、batch size、epochs)的部分一開始都是亂試的，不太確定什麼應該要大、什麼要小…直到到後來才慢慢地抓到一點感覺。

  實際從查資料到 coding 完大該花了 2-3 天，之後的時間都在測試…

  但是一天內試沒幾次又跑出說用量用完了!!!被限制了!!!可惡…