

人工智慧與機器學習 作業一

資管二 A 109403524 洪祥銘

[Colab 連結](#)

過程

取得資料集：

把原始的 zip 檔案下載到 local 再上傳到我自己的雲端上，並且 mount 上資料夾、再進行解壓縮

```
from google.colab import drive
drive.mount('/hw1_v2')
```

資料前處理：

建立一個用來存 author 的 dict，在 make_AuthorDict function 內把 author 的 name 跟數字 map 起來，再將 dict 反向從數字 map 回 name，存在 rev_class_name 內

```
author_Dict = dict()

def make_AuthorDict():
    index = 0
    for i in artists["name"]:
        author_Dict[i] = index
        index = index + 1
    return author_Dict

class_name = make_AuthorDict()

# 請建立將數字映成英文的dict。 EX: 0 -> Van_Gogh
rev_class_name = {v: k for k, v in author_Dict.items()}
```

get_label 用 repartition 把名字 (label) 從檔案名中最後一個底線提取出來

get_path 則把 dir 的路徑跟檔名合併 return

make_paths_label 將經過處理的 label、path 存到 list 內，再用 np.eye 把資料轉乘 one_hot format

```
def get_label(picName):
    # 請取出label並轉成數字
    # EX: Claude_Monet_1.jpg -> Claude_Monet -> 1
    # 先切最後一個底線前的字串
    picName = picName.rpartition('_')[0]
    # 對應到數字
    return author_Dict[picName]

def get_path(dir, picName):
    # 請將路徑合併
    # EX: ./train_resized/ + Claude_Monet_1.jpg => ./train_resized/Claude_Monet_1.jpg
    path = dir + picName
    return path

def make_paths_label(dir):
    img_list = os.listdir(dir)
    paths = []
    labels = []

    # 將preprocess完成的path、label用for迴圈放入paths和labels
    for path in img_list:
        labels.append(get_label(path))
        paths.append(get_path(dir, path))
        onehot_labels = np.eye(50)[labels]

    # 將labels轉成onehot
    return paths, onehot_labels
```

以下是有嘗試過但沒有在最後最佳 MODEL 出現的前處理

這個我稱之為【宇宙無敵爆裂暴力 underSample 法】，原本的目標是要將超過兩百筆作品的畫家隨機把該畫家的作品砍到剩兩百（砍的時候若重複就跳過，不強硬砍到兩百、保留隨機性），但是在今天課後與老師討教過得知這樣會失真，確實我最佳結果的模型當時並沒有去抑制作品數量，因此暴力刪減應該是不好的。

```
def underSampling():
    for i in range(0, 675):
        rand = random.randint(0,876)
        target_path = train_dir + 'Vincent_van_Gogh_' + str(rand) + '.jpg'
        if os.path.exists(target_path):
            os.remove(target_path)
            print('1 success')

    for i in range(0, 500):
        rand = random.randint(0,701)
        target_path = train_dir + 'Edgar_Degas_' + str(rand) + '.jpg'
        if os.path.exists(target_path):
            os.remove(target_path)
            print('2 success')
```

這個我稱之為【徒手造輪子克難 overSample 法】，把原本資料低於 150 筆的透過隨機翻轉、放大補充到自身數量四倍（或是等於 150），目標是希望能達成資料增強的效果。在繳交出來的 colab 檔案中沒有使用該方法，但在其他的測試中訓練完成時 Test Accuracy 也能有 0.64，略低於完全不做的情况，但相差一點點而已，我不是很確定是否有起到效果。跟老師討教完後，聽起來應該是有用，但有待更嚴謹的驗證。

```
# overSampling
def overSampling():
    for i in range(0, 50):
        print(i)
        rand_bond = artists.paintings[i]-1
        while artists.paintings[i] < 150 and artists.paintings[i] < rand_bond*4 :
            rand_num = random.randint(0, rand_bond)
            rand_path = train_dir + artists.name[i] + '_' + str(rand_num) + '.jpg'
            des_path = train_dir + artists.name[i] + '_' + str(artists.paintings[i]+1) + '.jpg'
            print(des_path)
            if os.path.exists(rand_path):
                shutil.copyfile(rand_path, des_path)
                image = Image.open(des_path)
                rand_trans = random.randint(0,2)
                if rand_trans % 3 == 0:
                    image = image.transpose(Image.FLIP_LEFT_RIGHT)
                elif rand_trans % 3 == 1:
                    image = image.transpose(Image.FLIP_TOP_BOTTOM)
                elif rand_trans % 3 == 2:
                    image = image.transpose(Image.ROTATE_90)

                image.save(des_path)
                artists.paintings[i]+=1

# overSampling()
```

接著設定圖片長寬、shuffle_buffer size，以及對圖片做正則化、轉成 0-1 格式，將 label 與圖片本身的 tensor map 起來，然後打散

```
# 決定你輸入模型的圖片長寬
# shuffle buffer size
IMG_WIDTH = 256
IMG_HEIGHT = 256
IMG_SIZE = None
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)/255

    # 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
    img = tf.image.resize(img, [IMG_WIDTH, IMG_HEIGHT])

    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)
    return full_ds

full_ds = make_dataset(train_dir)
```

設定 batch，我設定為 32，個人在訓練時發現如果將 batch size 設大於 32 會 over-fitting

```
# 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
# todo
batch_size = 32

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

Model 設計

簡而言之就是 Conv2D、maxpooling 疊疊樂，同時搭配 batchNormalization 加速收斂把 dropout 調高抑制訓練避免過度擬合。我覺得最關鍵的在此，剛開始 ACC 都一直上不去，索性就放棄認真啃資料，有幸讀到 [ithome 鐵人賽系列好文](#)，裡面提到 GlobalAvgPooling 將 CNN Layer 轉 Dense 效果比 flatten 好，就改用 GAP 了。

另外本來有在建立 model 時套了幾層增強資料用的 layer，但實做效果不佳，也就放棄了。本想參考 VGG16 的做法，但考量到訓練時間有點喪心病狂，就沒納入考量了。

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

    layers.Conv2D(128, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.BatchNormalization(),

    layers.Conv2D(256, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.BatchNormalization(),

    layers.Conv2D(512, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.BatchNormalization(),

    layers.Conv2D(1024, kernel_size=(3, 3), activation='relu'),
    layers.BatchNormalization(),

    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.3),
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.2),
    layers.Dense(num_classes, activation='softmax'),
])
model.summary()
```

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

# 資料增強
data_augmentation = keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.25),
    layers.experimental.preprocessing.RandomZoom(0.2),
])
```

制定訓練計畫：

起初訓練計畫只有一個，但我感覺收斂的狀況不好，Loss 與 Accuracy 到最後都還抖來抖去，啃資料得出的結論有可能是學習率還是太大，Loss 在 local minimum 跳來跳去，我就將訓練過程拆成兩部分（也有實驗過三部分），先跑 50 次看得出來的圖長怎樣，在將學習率手動調低，但看起來影響不大，感覺是一開始就要決定好比較實在，不過就相同的準確率，我這樣似乎有訓練得比較快，對我而言不乏是一種嘗試。

另外我的 optimizer 使用的是酷酷的、又新又潮的 [ranger 遊俠哥](#)（連結是 repo），跑起來的效果比 adam 好（ACC 從 adam 的 0.589 > ranger 的 0.677），我也不知道施了什麼魔法，跟老師討論時他說他也沒用過，看來我是先驅了（誤）。

```
] # todo
epochs = 50

# model.compile 決定learning strategy、Loss caculator
import tensorflow_addons as tfa
from keras import optimizers

radam = tfa.optimizers.RectifiedAdam(0.0002)
ranger = tfa.optimizers.Lookahead(radam, sync_period=6, slow_step_size=0.5)

model.compile(loss = "categorical_crossentropy", optimizer = ranger, metrics = ["accuracy"])

history = model.fit(train_ds, batch_size=batch_size, epochs=epochs, validation_data=val_ds, callbacks=[
    keras.callbacks.ModelCheckpoint("art_model_cool_best/model_{epoch:02d}", save_best_only=True)
])

[ ] epochs = 100

radam = tfa.optimizers.RectifiedAdam(0.00005) # 1/2
ranger = tfa.optimizers.Lookahead(radam, sync_period=6, slow_step_size=0.5)

model.compile(loss = "categorical_crossentropy", optimizer = ranger, metrics = ["accuracy"])

history = model.fit(train_ds, batch_size=batch_size, epochs=epochs, initial_epoch=50, validation_data=val_ds, callbacks=[
    keras.callbacks.ModelCheckpoint("art_model_cool_best/model_{epoch:02d}", save_best_only=True)
])
```

執行結果：

```
# 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_best.evaluate(test_ds)
print("Test loss:", score[0])
print("Test accuracy:", score[1])

27/27 [=====] - 3s 31ms/step - loss: 1.8709 - accuracy: 0.6778
Test loss: 1.870876669883728
Test accuracy: 0.6778442859649658
```

看到這個數字都快哭出來，想說都課金了，一定要來個猛的，不眠不休終於弄出來這個頂到肺的準確率，滿足虛榮心 XD

預測函式：

```
[ ] def predictAuthor(img):
    # 寫個單片模型預測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值(50.) 取最大值的index作為辨識結果
    # 4. 將辨識結果轉為畫家名字

    img = np.expand_dims(img, axis=0)
    authorNo = np.argmax(model.predict([img]), axis=-1)[0]
    authorName = rev_class_name[authorNo]

    return authorName

[ ] plt.figure(figsize=(16, 16))
for index, imgName in enumerate(show_imgs):
    imgpath = train_dir + imgName
    img = cv.imread(imgpath)
    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 : {} \n Pred Author : {}".format("_".join(imgName.split("_")[:-1]), predictAuthor(img)), size=11)
```

與結果：


```
[ ] 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(predictAuthor(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()

toast.png
• toast.png(image/png) - 97247 bytes, last modified: 2022/4/18 - 100% done
Saving toast.png to toast (1).png
predict author: Pablo_Picasso
```



心得：

首先就是爽到 XX，問了去年的學長姊跟今年的同學，估計很難比我的還頂，真的花了很多時間做這份功課（周末都沒啥睡討拍），也得到一些大佬開示，這邊我超感激。但還有一點就是真的剛好讀到上述的[鐵人賽系列文章](#)，幫助很大，站在巨人的肩膀上，果然能看得更遠。當然也趁機多弄懂很多python 的語法，把本來偷懶不學、似懂非懂的 lambda 精熟。

心得更新：

在寫這份報告的同時我用下午得到智輝帥哥助教的開示，將 shuffle_buffer size 設定 ABAP as big as

possible，得到了這個

```
✓ [26] # 讀入測試資料並評估模型
2 test_ds = make_dataset(test_dir)
3 test_ds = test_ds.batch(batch_size)
4 score = model.evaluate(test_ds)
5 print("Test loss:", score[0])
6 print("Test accuracy:", score[1])

27/27 [=====] - 3s 53ms/step - loss: 1.3856 - accuracy: 0.7078
Test loss: 1.3855880498886108
Test accuracy: 0.7077844142913818
```

我好頂，我好開心 XD