一、Colab 連結：

https://colab.research.google.com/drive/11tN27tUltETq-MmTNYkdcFKEKvW1PT9k?usp=sharing

二、Test accuracy:

本模型最終 test accuracy 為 51.86%

```
# 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])
```

```
27/27 [==============================] - 2s 52ms/step - loss: 2.0323 - accuracy: 0.5186
Test loss: 2.032339334487915
Test accuracy: 0.5185628533363342
```

圖(一)、模型正確率

三、撰寫過程與截圖：

我在這次作業的準確率從 20%左右的正確率一路上升到 51%。其中，正確率在 35%、40%卡住了許久，中間試過不同激活函數、不同優化器，最後費時 40 小時左右的訓練，正確率成功突破 50%。

以下我會講述我如何寫這份作業並且是如何思考的：

作業的一開始便要進行資料前處理，我撰寫 make_author_dict 圖(二)和 get_label 圖(三)來建立藝術家名稱與標籤之間的映射。我先從每一個檔案提取出畫家的名稱，並創建一個字典將每個畫家的名字對應到一個唯一的數字。另外，我也創建 get_path 圖(四)和 make_paths_label 圖(五)以生成圖像路徑和對應的標籤，並且轉成 one_hot label 圖(六)輸出。

```
# 請建立將英文映射成數字的 dict，EX: Van_Gogh --> 0
def make_author_dict():
    authors = ['_'.join(name.split('_')[:-1]) for name in os.listdir(train_dir) if name.endswith('.jpg')]
    unique_authors = list(set(authors))
    author_to_id = {author: id for id, author in enumerate(unique_authors)}
    return author_to_id

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

```
{'Camille_Pissarro': 0, 'Amedeo_Modigliani': 1, 'Vasiliy_Kandinskiy': 2, 'Henri_Rousseau': 3, 'Paul_Gauguin'
{0: 'Camille_Pissarro', 1: 'Amedeo_Modigliani', 2: 'Vasiliy_Kandinskiy', 3: 'Henri_Rousseau', 4: 'Paul_Gaugu
```

圖(二)、make_author_dict 函數

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

圖(三)、get_label 函數

```python
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
```

圖(四)、get_path 函數

```python
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 img_name in img_list:
        path = get_path(dir, img_name)
        label = get_label(img_name)
        paths.append(path)
        labels.append(label)

    # 將 labels 轉成 onehot
    onehot_labels = tf.keras.utils.to_categorical(labels, num_classes=len(class_name))
    return paths, onehot_labels
```

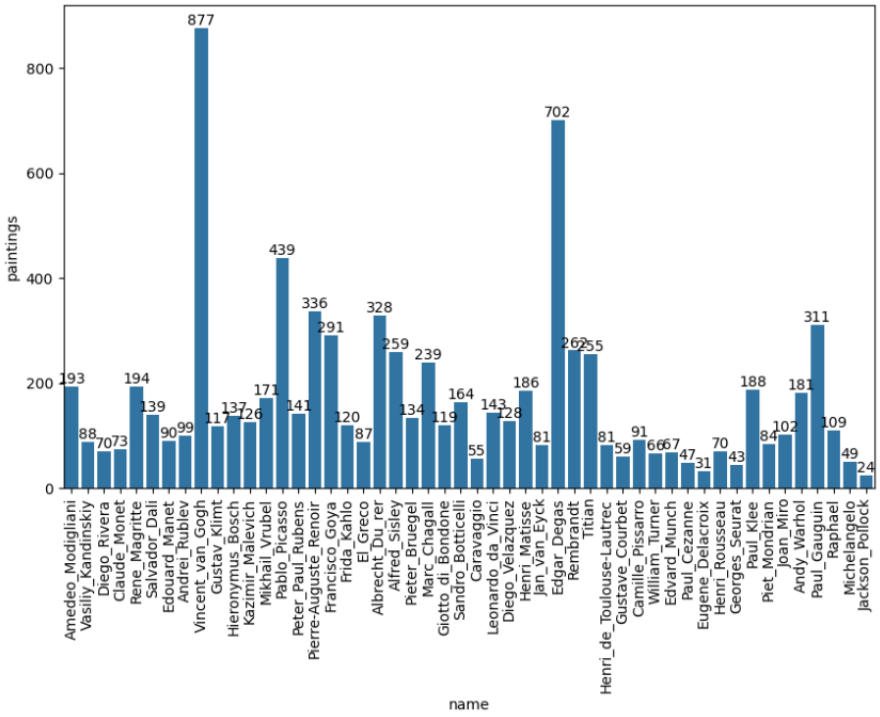
圖(五)、make_path_label 函數

```python
# 來查看一下
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/Rene_Magritte_3.jpg
./train_resized/Rene_Magritte_75.jpg
./train_resized/Kazimir_Malevich_73.jpg
./train_resized/Albrecht_Du_rer_118.jpg
./train_resized/Pierre-Auguste_Renoir_242.jpg
--------------------
labels :
[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. 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. 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. 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. 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.]
```

圖(六)、one_hot label 輸出

在進行前處理時也遇到了 class 資料不平衡的問題。一開始，我依照作業中給的提示使用 keras 的 class_weight 來解決這個問題，然而，後來我發現自己為所有的每個 class 進行權重分配的效果比較好，因此最後我選擇使用圖(八)的方法，計算每個類別的權重，將每個類別轉成相同的值來計算。



圖(七)、將人數打印出來顯示在圖上

```
[ ]  def create_class_weights(artist_dataframe, artist_to_id):
         total_paintings = artist_dataframe['paintings'].sum()
         class_weights = {}

         for artist_name, artist_index in artist_to_id.items():
             num_paintings = artist_dataframe[artist_dataframe['name'] == artist_name]['paintings'].values[0]
             class_weight = total_paintings / (num_paintings * len(artist_to_id))
             class_weights[artist_index] = class_weight

         return class_weights

     # 使用你的 artists DataFrame 和 make_author_dict 函數
     author_to_id = make_author_dict()
     class_weights = create_class_weights(artists, author_to_id)

     print("Class Weights:", class_weights)
```

圖(八)、Class weight 程式碼

優化權重的同時，我嘗試了各式各樣的參數優化模型，見圖(九、十、十一)，搭配我的訓練模型(圖十三)與訓練計畫(圖十四)，最後我達到最佳值 51%。至於模型訓練的方式，我交替使用了 5 次 Convolution(filter 數分別為 64、128、256、512、1024)與 5 次 MaxPooling 還有 5 次 BatchNormalization，我也使用了 3 次 Dropout(2 次 0.2、1 次 0.4)並採用 GlobalAveragePooling()取代 Flatten()以及第一次 Dense 層採用神經數量為 1024。

```
# 決定你輸入模型的圖片長寬
IMG_WIDTH = 256
IMG_HEIGHT = 256
IMG_SIZE = (IMG_WIDTH, IMG_HEIGHT)
# shuffle buffer size
SHUFFLE_BUFFER = 7000
```

圖(九)、input_size 及 shuffle_buffer

```python
# 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)
```

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

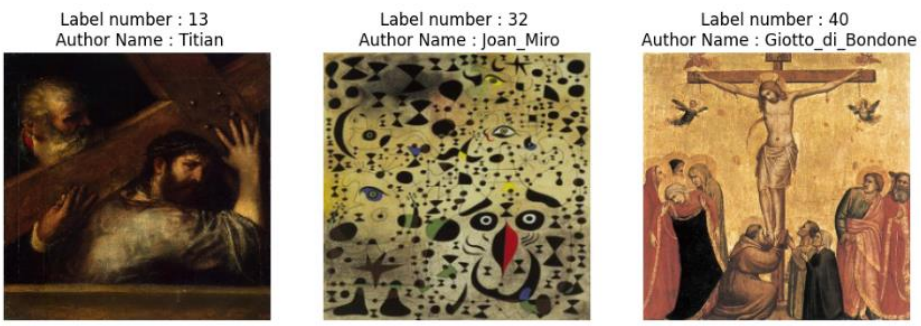
圖(十)、分割 training 與 validation data

```python
[ ] # 查看添加batch後的維度
    trainiter = iter(train_ds)
    x, y = trainiter.next()
    print("training image batch shape : ", x.shape)
    print("training label batch shape : ", y.shape)
```

```
training image batch shape :  (32, 256, 256, 3)
training label batch shape :  (32, 50)
```

圖(十一)、添加 batch 後的維度

```python
# 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")
```



圖(十二)、label 與 author 的對應

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

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

    layers.Conv2D(128, kernel_size=(3, 3), padding = "same", activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.BatchNormalization(),
    layers.Dropout(0.2),

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

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

    layers.Conv2D(1024, kernel_size=(3, 3), padding = "same", activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.BatchNormalization(),
    layers.Dropout(0.4),

    layers.GlobalAveragePooling2D(),
    #layers.Flatten(),
    layers.Dense(1024, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(num_classes, activation='softmax')
])

model.summary()
```

圖(十三)、模型訓練方式

```python
# todo
from tensorflow.keras.optimizers import Adam
optimizer = tf.keras.optimizers.Adam(learning_rate=0.00008)
EPOCHS = 35
model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
history = model.fit(train_ds, epochs=EPOCHS, validation_data=val_ds, class_weight = class_weights)
```

圖(十四)、訓練計畫

另外，我使用 predict_author 函數(圖十五)，幫助我預測畫作是由哪位藝術家創作的。在這個函數中我先擴展圖像維度，將圖像從(高度，寬度，顏色通道)轉變為(1，高度，寬度，顏色通道)，接著，我利用模型進行預測找到最大機率對應的索引，最後再轉成畫家名字(圖十六)。最後，我也成功上傳圖片並預測畫家(圖十七、圖十八)。

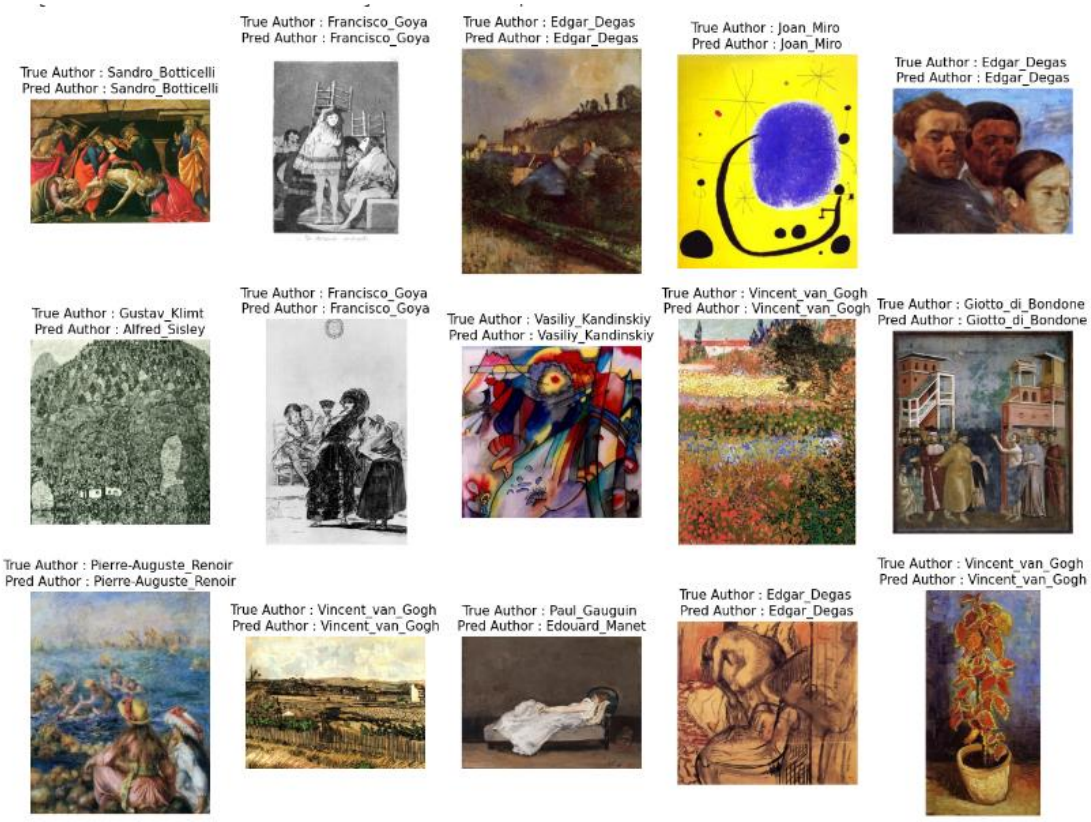
```python
def predict_author(img):
    # 擴展圖像維度
    img_expanded = np.expand_dims(img, axis=0)

    # 使用模型進行預測
    predictions = model.predict(img_expanded)

    # 找到最大概率對應的索引
    predicted_index = np.argmax(predictions)

    # 將索引轉換為畫家名字
    author_name = rev_class_name[predicted_index]
    return author_name
```

圖(十五)、predict_author 函數



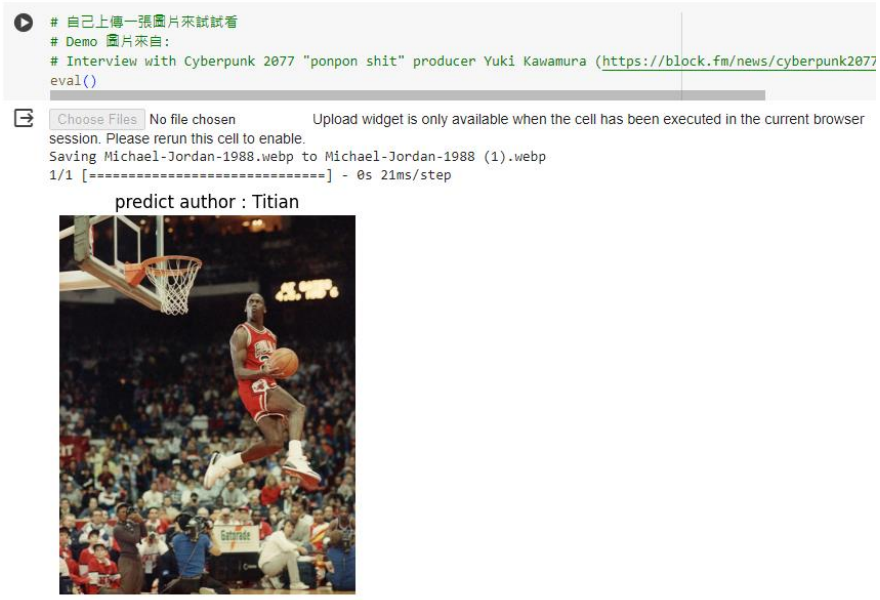
圖(十六)、predict_author 執行結果

```
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(predict_author(img)))
    plt.axis("off")
    plt.show()
```

圖(十七)、上傳圖片



圖(十八)、上傳結果

四、心得：

這次作業是我對於 AI 的第一次探索，在寫這次作業前，我對於 AI 的模型製作懵懵懂懂，總認為 AI 是一個非常高深的技術，但是，經過這次作業後，我才發現原來 AI 模型的訓練非常有趣，並且我也有能力做出一個有模有樣的模型。

在完成這次作業的過程中我遇到了許多問題，例如，keras 相關相關程式碼的不熟悉、Colab 的 RAM 不夠用等等。然而，我並沒有因此而氣餒，反而是更勤勞的在網路上查資料，學習許多解法與技巧。在學習的過程中，我印象最深刻的便是制訂各式各樣的模型訓練計劃，其中不乏自己設定 learning rate 的 decay rate，嘗試各式各樣的激活函數例如 tanh、sigmoid，使用不同 Optimizer 例如 Adagrad 等等。有些訓練計劃導致訓練時間非常長，有些則是訓練時間快速但準確率很普通，最後，經過無數次的嘗試與訓練時煎熬的等待，我訓練出了一個正確率超過 50%的模型。

總而言之，這次的經歷不只讓我學到了 CNN 技術，更讓我學會了如何面對挑戰、如何不斷探索和學習，希望未來我可以持續精進人工智慧相關的能力，帶著這次的經驗繼續在 AI 這條路上前行。

五、參考資料：

計算 class_weight
https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html
如何選擇適當 epoch
https://blog.51cto.com/u_12207/8051965
GlobalAveragePooling 與 Flatten 的差異與比較
https://ithelp.ithome.com.tw/articles/10276291?sc=iThomeR
Predicting the Artist from his work using CNNs
https://www.kaggle.com/code/yadhua/predicting-the-artist-from-his-work-using-cnns
SGD, Momentum, Adagrad, Adam Optimizer
https://medium.com/%E9%9B%9E%E9%9B%9E%E8%88%87%E5%85%94%E5%85%94%E7%9A%84%E5%B7%A5%E7%A8%8B%E4%B8%96%E7%95%8C/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92ml-note-sgd-momentum-adagrad-adam-optimizer-f20568c968db