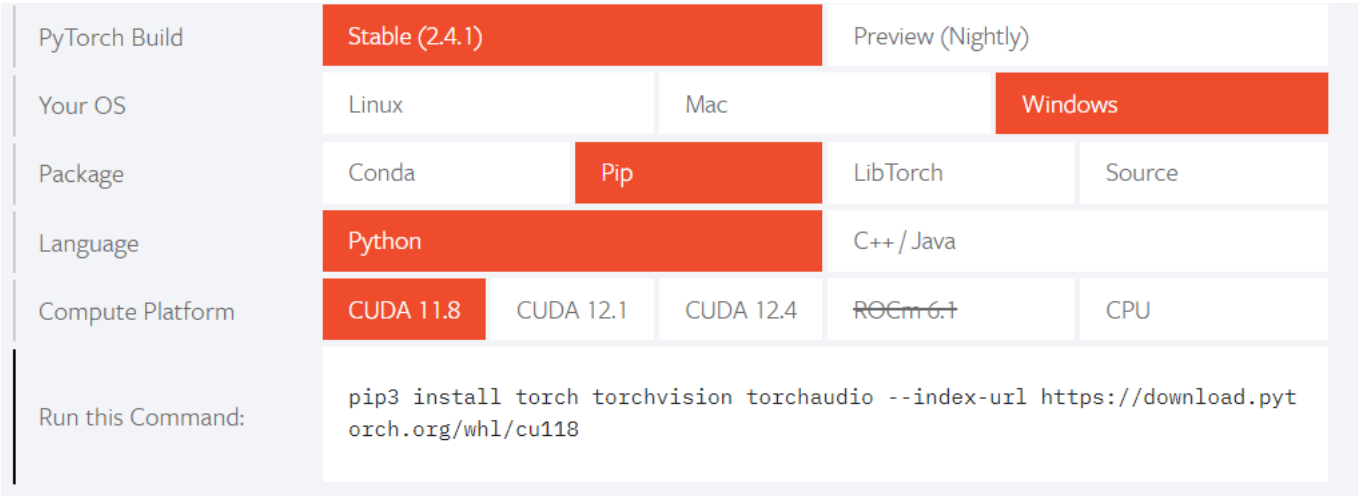
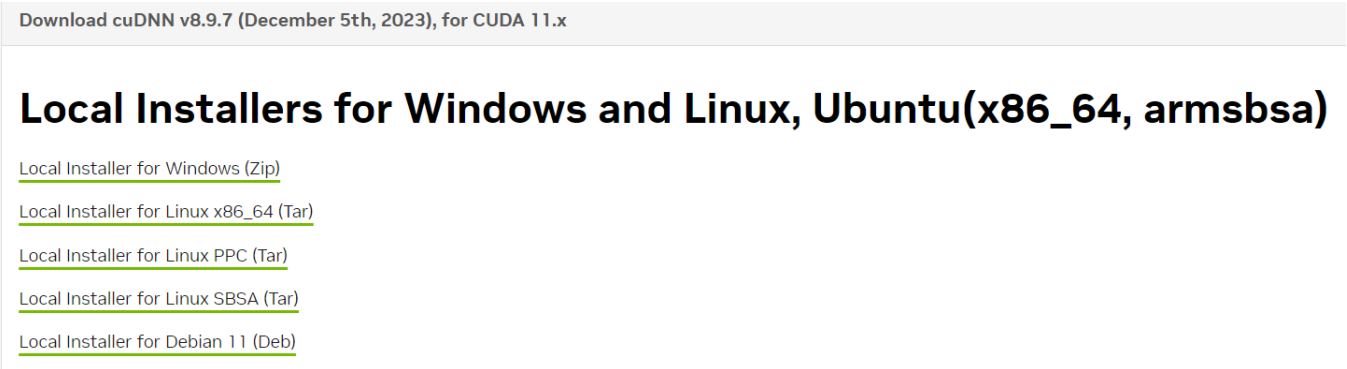
基於自然語言寫詩

- 詩集的收集、資料前處理
- 判斷時詞風格(輸入一篇文章)
- AI 生成詩詞(內容請輸入一句 5~15)
- 圖像生成文章

本系統環境

本系統環境 tensorflow 2.17.0 版本、python 3.12.3、PyTorch 2.4.1、NVIDIA CUDA 11.8、NVIDIA cuDNN NVIDIA 8.9.7 GeForce RTX 3060。





介面是由 django 建置呈現

環境設置

提供了數千個預訓練模型來執行不同模式（例如文字、視覺和音訊）的任務。

Transformers 下載

```
!pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting transformers
  Downloading transformers-4.26.0-py3-none-any.whl (6.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.3/6.3 MB 91.7 MB/s eta 0:00:00
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.8/dist-packages (from transformer
Requirement already satisfied: filelock in /usr/local/lib/python3.8/dist-packages (from transformers) (3.9.0
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.8/dist-packages (from transformers) (4.6
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1
  Downloading tokenizers-0.13.2-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.6/7.6 MB 108.0 MB/s eta 0:00:00
Collecting huggingface-hub<1.0,>=0.11.0
```

```
import transformers
print(transformers.__version__)
print(transformers.__spec__)
```

```
4.26.0
ModuleSpec(name='transformers', loader=<_frozen
```

隨著深度學習技術的不斷發展，大模型在自然語言處理領域的應用越來越廣泛。然而，大模型的訓練和部署需要大量的運算資源和儲存空間，同時也需要對大量的資料進行預處理。其中，分詞是預處理中的重要任務，它能夠將文字轉換為模型可以理解的符號表示。在大模型中，由於詞彙量龐大，傳統的分詞工具往往無法滿足需求。這時，SentencePiece 應運而生。

SentencePiece 是一款開源的分詞工具，由 Google 開發並維護。它採用了基於統計的分詞方法，可以將句子切割成具有相同語義的子字串（即“詞”），並產生一個高效的編碼表示。與傳統的分詞工具不同，SentencePiece 支援增量分詞和變長分詞，可以更精確地處理各種語言資料。

```
!pip install sentencepiece
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting sentencepiece
  Downloading sentencepiece-0.1.97-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 49.5 MB/s eta 0:00:00
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.97
```

bert-extractive-summarizer 是一個使用 Bert 加上 Clustering 進行抽取式摘要的模型，詳細原理、實作可以看作者的 Github 有論文連結。因為範例是英文的，用於中文需要稍作修改，載入中文的模型。

```
!pip install bert-extractive-summarizer
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting bert-extractive-summarizer
  Downloading bert_extractive_summarizer-0.10.1-py3-none-any.whl (25 kB)
Requirement already satisfied: transformers in /usr/local/lib/python3.8/dist-packages (from bert-ex
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-packages (from bert-ex
Requirement already satisfied: spacy in /usr/local/lib/python3.8/dist-packages (from bert-extractiv
```

中研院的 CKIP Transformers — 語言模型與 NLP 任務工具，就是一個可以幫助我們處理自然語言工具。

他有三大功能：

中文斷詞

詞性標註

專有名詞辨識

```
!pip install ckip-transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting ckip-transformers
  Downloading ckip_transformers-0.3.2-py3-none-any.whl (26 kB)
Requirement already satisfied: torch>=1.5.0 in /usr/local/lib/python3.8/dist-packages (from ckip-trans
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.8/dist-packages (from ckip-transf
Requirement already satisfied: transformers>=3.5.0 in /usr/local/lib/python3.8/dist-packages (from ck
```

詩集的收集、資料前處理

資料收集、部分資料是重複抽樣與利用生成模型擷取元資料的一部份去 GPT 生成模型去生成資料

```
all_cats = ['創世紀', '原住民', '客家', '新月', '新詩', '現代詩', '笠', '藍星']

for cat in all_cats:
    pd_data = pd_all[pd_all.class_id==cat]
    print('{}: {} (總數)'.format(cat, pd_data.shape[0]))
```

```
創世紀: 300 (總數)
原住民: 300 (總數)
客家: 300 (總數)
新月: 300 (總數)
新詩: 300 (總數)
現代詩: 300 (總數)
笠: 300 (總數)
藍星: 300 (總數)
```

儲存在在 Excel 檔案，再用 pandas 讀取資料

```
df = pd.read_excel("C:\\Users\\user\\Desktop\\django\\literature_project\\AI_PART\\peot_Classification_model\\all_peot_0206.xlsx")
df=df[['athor','Name','class_id','poet']]
print("總數: %d ." % len(df))
df.sample(10)
```

總數: 2400 .

	athor	Name	class_id	poet
774	羅門	詩的歲月——給蓉子.txt	藍星	詩的歲月——給蓉子\n\n要是青鳥不來\n春日照耀的林野\n如何飛入明麗的四月\n踩一路的續...
793	鄧禹平	我的思念.txt	藍星	我的思念\n\n化我的思念為白雲片片；\n飄過平原，\n飄過高山，\n飄到你的頭頂 窗前，...
2188	鄭愁予	未題.txt	現代詩	未題\n無聲地匯流著，在一一二月的雨天\n是我們臂上的靜脈的小青河\n一環環的漩渦，朵朵地跳...
1327	楊煥	花.txt	現代詩	花\n叮呤呤，叮呤呤\n鈴蘭花搖響一串串小鈴子；\n嗚啦啦，嗚啦啦，\n牽牛花吹起一隻隻小喇...
210	新增	出席.txt	客家	出席\n\n該當時\n喉涎頭已像卡著一粒火雁个樣\n該當時\n緊顫个身體無法度企好勢\n該當...
2082	其他	花園：生命之初.txt	新詩	花園：生命之初　\n我們是草本的植物\n體態糾結如藤\n在黑暗裡附著欄杆\n打轉、舒展纖細的...
432	其他	快照亭.txt	新詩	快照亭\n\n金屬疲勞的週五適合反抒情\n一道閃光，二度灼傷\n請順手，帶走那些擠成一團的親...
435	其他	憂鬱販賣機──致活在地獄裡的人們.txt	新詩	憂鬱販賣機──致活在地獄裡的人們　\n可以和我說聲加油嗎？\n（最好是不要）\n太過黏膩的難...
1354	紀弦	海的意志.txt	現代詩	海的意志\n——天哪！天哪！\n在夢的漩渦裡，\n我是時常做著\n苦痛的呻吟的。\n可是颶風...
1906	卞之琳	寂寞.txt	新月	寂寞\n鄉下小孩子怕寂寞，\n枕頭邊養一隻蟈蟈；\n長大了在城裏操勞，\n他買了一個夜明錶。...

內容都是中文,所以要對中文進行一些預處理工作,這包括刪除文本中的標點符號,特殊符號,還要刪除一些無意義的常用詞(stopword),因為這些詞和符號對系統分析預測文本的內容沒有任何幫助,反而會增加計算的複雜度和增加系統開銷,所有在使用這些文本數據之前必須要將它們清理乾淨。

中文停用詞包含了很多日常使用頻率很高的常用詞,如 吧，嗎，呢，啥等一些感嘆詞等,這些高頻常用詞無法反應出文本的主要意思,所以要被過濾掉。

```
# 載入結巴中文
import jieba
# 讀取中文文檔
text = open('speech.txt', 'r', encoding='utf8').read()
# 刪除文檔中的標點符號
cleanText = "".join(c for c in text if c not in (';',',','。','！
','：','「','」','…','、','？','【','】
','.',':','?',';','!','~','‘','+','-
','<','>','/','[',']','{','}',"'",'"'))
# 進行分詞
words = jieba.cut(cleanText, cut_all=False)
# 讀入停用詞
```

stopwords = {}.fromkeys([line.rstrip() for line in open('stopWords.txt', encoding='utf8')])

```
df['cut_review'] = df['clean_review'].apply(lambda x: " ".join([w for w in list(jb.cut(x))]))
df.head(10)
```

	athor	Name	class_id	poet	cat_id	clean_review	cut_review
0	亞弦	一般之歌.txt	創世紀	一般之歌\n鐵蒺藜那廂是國民小學，再遠一些是鋸木廠\n隔壁是蘇阿姨的園子；種著萵苣，玉蜀黍\...	0	一般之歌鐵蒺藜那廂是國民小學再遠一些是鋸木廠隔壁是蘇阿姨的園子種著萵苣玉蜀黍三棵槐樹左邊還有...	一般 之歌 鐵 蒺藜 那廂 是 國民小學 再 遠 一些 是 鋸木廠 隔壁 是 蘇 阿姨 的 ...
1	亞弦	上校.txt	創世紀	上校\n\n那純粹是另一種玫瑰\n自火焰中誕生\n在蕎麥田裡\n他們遇見最大的會戰\n而他的一條...	0	上校那純粹是另一種玫瑰自火焰中誕生在蕎麥田裡他們遇見最大的會戰而他的一條腿訣別於一九四三年他...	上校 那純粹 是 另 一種 玫瑰 自 火焰 中誕生 在 蕎麥田裡 他們 遇見 最大 的 會戰...
2	亞弦	乞丐.txt	創世紀	乞丐\n\n不知道春天來了以後將怎樣\n雪將怎樣\n知更鳥和狗子們，春天來了以後\n以後將怎...	0	乞丐不知道春天來了以後將怎樣雪將怎樣知更鳥和狗子們春天來了以後以後將怎樣依舊是關帝廟依舊是洗...	乞丐 不 知道 春天 來 了 以後 將 怎樣 雪將 怎樣 知 更 鳥 和 狗子 們 春天 來 ...

LSTM 建模

資料預處理完成以後，接下來我們要開始進行 LSTM 的建模工作：

我們要將 cut_review 資料進行向量化處理，我們要將每條 cut_review 轉換成一個整數序列的向量

設定最常使用的 5000 個字

設定每條 cut_review 最大的詞語數為 50 個(超過的將會被截去，不足的將會被補 0)

```
# 設置最頻繁使用50000個詞(在texts_to_matrix是會取前MAX_NB_WORDS,會取前MAX_NB_WORDS列)
MAX_NB_WORDS = 5000
# 每條cut_review最大的長度
MAX_SEQUENCE_LENGTH =50
# 設置Embeddingceng層的維度
EMBEDDING_DIM = 50

tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(df['cut_review'].values)
word_index = tokenizer.word_index
print('共有 %s 個不相同的詞語.' % len(word_index))
```

```
共有 23750 個不相同的詞語.
```

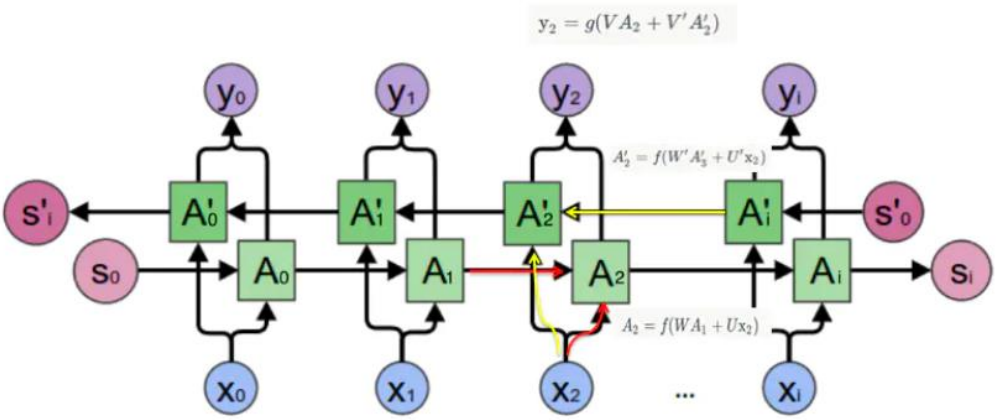
詩集風格的分類

```
[47] from keras.layers import Conv1D, GlobalAveragePooling1D, MaxPooling1D
     #定义模型.
     model = keras.Sequential([
     layers.Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]),
     layers.SpatialDropout1D(0.1),
     layers.Bidirectional(LSTM(8, return_sequences=True), input_shape=(8, 10)),
     layers.Bidirectional(LSTM(30)),
     layers.Dense(8, activation='softmax')
     ])
     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
     print(model.summary())
```

訓練和測試的資料集都準備好以後,接下來我們要定義一個 BiLSTM 雙向 LSTM 序列模型:

模型的第一次是嵌入層(Embedding)，它使用長度為 50 的向量來表示每一個字語
SpatialDropout1D 層在訓練中每次更新時， 隨機將輸入單元的比率設為 0.1， 這有助於防止過度擬合
這裡設置了兩層 BiLSTM 雙向 LSTM
輸出層為包含 8 個分類的全連接層
由於是多分類，所以激活函數設定為'softmax'
由於是多分類, 所以損失函數為分類交叉熵 categorical_crossentropy

LSTM 是 RNN 的一種。因為其可以接受序列資料、多個輸入、有記憶這些特點，非常適合用來處理文本資料。而 BiLSTM 名稱中的 Bi-directional 其實就道盡了它的特點，它是由前向 LSTM 和後向 LSTM 組合而成的「雙向」LSTM。也就是說，在一個序列的輸入中，BiLSTM 能夠同時編碼由前至後的訊息和由後至前的訊息，如下圖所示。這種能力在文本或情緒的分析時是非常有用的。假設有個句子是：「今天晚上的_____很好聽，令人回味無窮。」若是只從前向推測，可能的候選就非常多，有晚餐、聚會、音樂會、月亮.....然而，若將後向的編碼考慮進來，在以上的選項中就只有「音樂會」是最有可能的，顯而易見，範圍縮小了許多。



訓練時跑了 25 個 Epoch

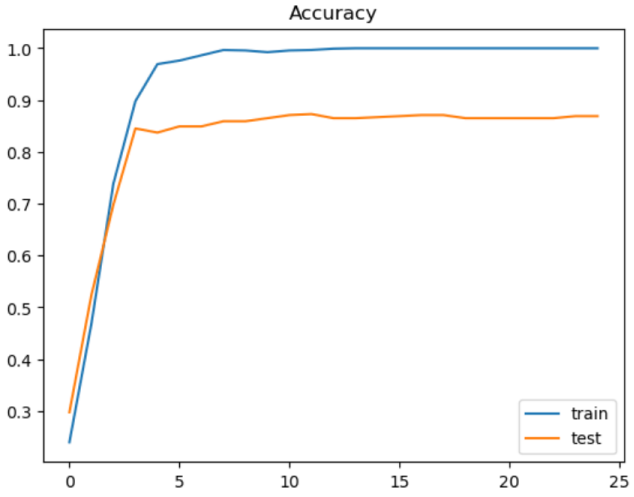
```
epochs = 25
batch_size = 10

#history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,validation_split=0.13)
                    #callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)])
history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,validation_split=0.30)
#                    callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)])
```

```
Epoch 1/25
118/118 ━━━━━━━━━━━━━━━━━━━━ 7s 24ms/step - accuracy: 0.1937 - loss: 2.0114 - val_accuracy: 0.2976 - val_loss: 1.7123
Epoch 2/25
118/118 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.4048 - loss: 1.6098 - val_accuracy: 0.5238 - val_loss: 1.3347
Epoch 3/25
118/118 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.6861 - loss: 0.9747 - val_accuracy: 0.6984 - val_loss: 0.9174
Epoch 4/25
118/118 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8802 - loss: 0.4497 - val_accuracy: 0.8452 - val_loss: 0.6220
Epoch 5/25
118/118 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.9749 - loss: 0.1598 - val_accuracy: 0.8373 - val_loss: 0.6823
Epoch 6/25
118/118 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.9754 - loss: 0.1114 - val_accuracy: 0.8492 - val_loss: 0.6043
Epoch 7/25
118/118 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - accuracy: 0.9892 - loss: 0.0564 - val_accuracy: 0.8492 - val_loss: 0.6672
Epoch 8/25
```

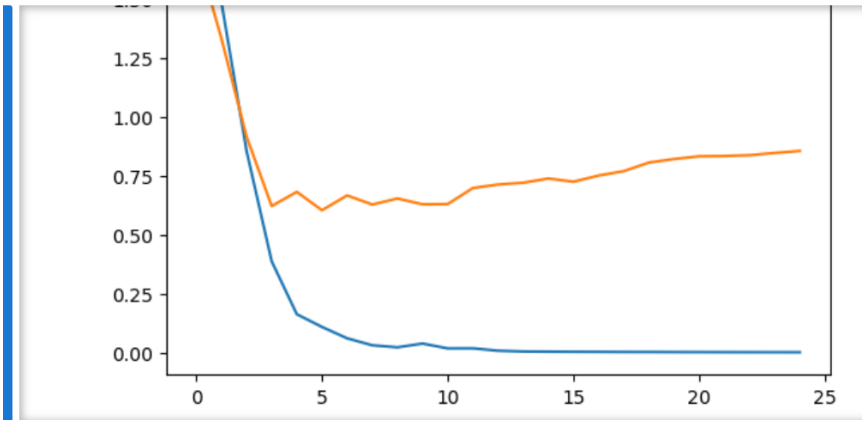
訓練準確率大約 83%

[34]:
```
from  sklearn.metrics import classification_report

print('accuracy %s' % accuracy_score(y_pred, Y_test))
print(classification_report(Y_test, y_pred,target_names=cat_id_df['class_id'].values))
```

```
accuracy 0.8333333333333334
              precision    recall  f1-score   support

          創世紀       0.71      0.78      0.74       106
          原住民       0.78      0.90      0.84        82
           客家       1.00      0.95      0.98        88
           新月       0.90      0.78      0.83        94
           新詩       0.85      0.80      0.82        90
          現代詩       0.83      0.79      0.81        80
           笠       0.85      0.87      0.86        76
           藍星       0.82      0.82      0.82       104

    accuracy                           0.83       720
   macro avg       0.84      0.84      0.84       720
weighted avg       0.84      0.83      0.83       720
```




```
plt.legend()
plt.show();
```




```
predict("夜深，你便夢入願力 見天子的寶冠落地成宮 牆冒光綻蓮放樹撼葉茂實透音氾  波瀾出五百萬億層的莊嚴 於兜率天內院如織的光音中
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
0.949595    創世紀
0.008533    原住民
0.000001     客家
0.004058     新月
0.000063     新詩
0.021079    現代詩
0.001699      笠
0.014972     藍星
dtype: object
```

介面操作:

標題、內容、作者按下送出並預測分析風格

["創世紀詩詞風格", "原住民詩詞風格", "客家詩詞風格", "新月詩詞風格","新詩詩詞風格","現代詩詞風格","笠詩詞風格","藍星詩詞風格"]比例

AI 生成詩詞(內容請輸入一句 5~15)

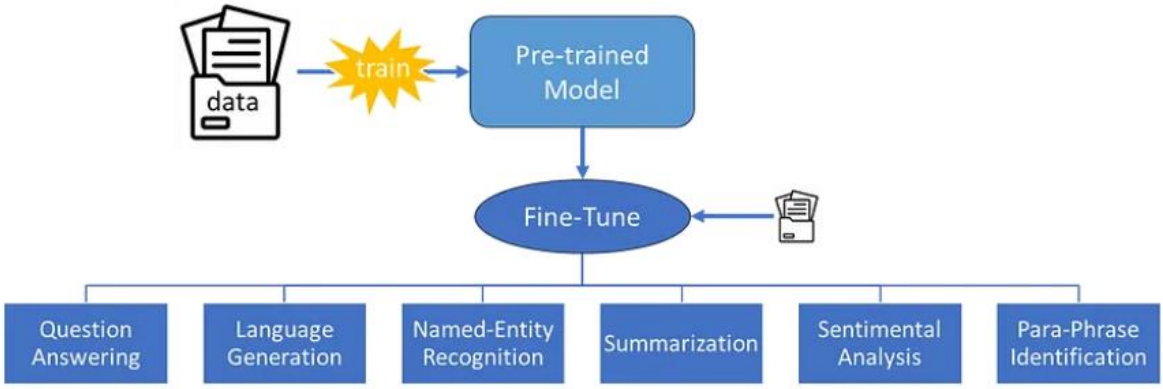
預訓練模型

預訓練模型本質上是一個已保存的網絡，它已經在大量資料集（例如大規模文字生成任務）上進行了預先訓練。預訓練模型的實用性在於它的適應性——您可以按原樣使用它，也可以採用遷移學習來針對特定任務微調模型。

Hugging Face Transformers 提供了數千個預訓練模型來執行文字、視覺和音訊任務。基本原理簡單而強大——我們可以保存經過訓練的模型的參數以供將來使用，然後我們可以與任何人共享它，無論是用於推理還是作為後續微調的基礎。

什麼是微調以及為什麼？

預先訓練的大型語言模型（LLM）在通用、多樣化的語料庫上進行訓練，它們提供許多功能，但並不通用。微調從現有的預訓練模型開始，並繼續在專門的語料庫上進行訓練，以改變參數以在特定任務上。在本文中，我將在蒐集詩集料庫上對已經理解中文的 GPT2 模型進行微調，這裡我使用的是中研院 ckiplab/gpt2-base-chinese 預訓練好的繁體中文模型，使模型能夠根據輸入成分產生文章。遷移學習的這種應用增強了模型在特定領域任務中的能力，而無需從頭開始訓練過程，這可能非常耗時且計算成本高昂。



操作說明、模型準備：

使用 BERT 模型的預訓練版本 bert-base-chinese，透過 ckiplab/gpt2-base-chinese 來建立文本生成模型。

初始化 tokenizer，將文本轉換成模型可接受的輸入格式。

```
from transformers import (
  BertTokenizerFast,
  AutoModel,
)
from transformers import AutoTokenizer,AutoModelForCausalLM

tokenizer = BertTokenizerFast.from_pretrained('bert-base-chinese')
model = AutoModelForCausalLM.from_pretrained('ckiplab/gpt2-base-chinese') # or other models above
```

基於 BERT 的中文分詞

深度學習主要是特徵學習，端到端訓練，適合有大量語料的場景。另外各種工具越來越完善，利用 GPU 可大幅提升訓練速度。使用深度學習的方法進行中文分詞

```
inputs = tokenizer("你好嗎", return_tensors="pt")
outputs = model(**inputs)
print(inputs)
```

```
{'input_ids': tensor([[ 101,  872, 1962, 1621,  102]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0]]), 'atten
```

```
raw_inputs = [
    "大家好嗎",
    "我很高興",
    "鳥兒會飛"
]
inputs = tokenizer(raw_inputs, padding=True, truncation=True, return_tensors="pt")
print(inputs)
```

```
{'input_ids': tensor([[ 101, 1920, 2157, 1962, 1621,  102],
        [ 101, 2769, 2523, 7770, 5646,  102],
        [ 101, 7852, 1051, 3298, 7606,  102]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0]
        [0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1]])}
```

- 使用 GPT2 快速測試生成文章

中研院這裡使用了 ckiplab/gpt2-base-chinese 測試

```
from transformers import pipeline
```

bos_token, eop_token, 和eos_token是在文字生成任務中常用的特殊標記。

1. bos_token(Beginning of Sentence Token) 表示句子的開頭。使用生成模型產生文字時，可以將bos_token插入輸入文字的開頭，以指示模型開始產生新的句子。

2. eop_token(End of Paragraph Token) 表示段落的結束。在產生長篇文字時，可以將eop_token插入到段落的結尾，以指示模型產生新的段落。

3. eos_token(End of Sentence Token) 表示句子的結束。當生成模型產生句子時，可以將eos_token插入到句子的末尾，以指示模型停止生成。

```
[ ] # Example of the result of the tokenization process with padding
    base_tokenizer.decode(tokenized_train_dataset['input_ids'][0])
```

```
'[CLS] <|endoftext|> 嫁 耳 環 仔 叮 嗑 搖 在 我 介 耳 公 邊 講 出 嫁 介 心 情 隻 隻 金 指 含 著 傳 統 介 情 愛 首 扼 仔 落
我 三 從 四 德 阿 姑 包 分 我 一 句 話 喚 我 莫 忘 祖 宗 言 雖 然 蒙 等 一 層 濛 濛 介 面 紗 我 也 讀 得 出 這 本 沉 長 介
出 去 介 這 碗 水 紙 扇 輕 輕 跌 落 地 阿 姆 撿 起 搖 清 涼 自 言 自 語 唸 四 句 公 婆 相 惜 早 供 賴 賴 <|EOS|> [SEP] <|p
d|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|>
|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad
> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|> <|pad|>'
```

使用用自己收集的 dataset，來微調模型的 pre-trained model

初始化 tokenizer 和模型：使用 Hugging Face 的 Transformers 庫中的 BertTokenizer，從預訓練的 BERT 模型中初始化 tokenizer 和分類模型。model_name 指定了要使用的預訓練模型，這裡使用了 ckiplab/gpt2-base-chinese

```
# the eos and bos tokens are defined
bos = '<|endoftext|>'
eos = '<|EOS|>'
pad = '<|pad|>'

special_tokens_dict = {'eos_token': eos, 'bos_token': bos, 'pad_token': pad}

base_tokenizer = BertTokenizerFast.from_pretrained('bert-base-chinese',vocab_size=21128, model_max_len=512, is_fast=True)
# the new token is added to the tokenizer
num_added_toks = base_tokenizer.add_special_tokens(special_tokens_dict)

# the model config to which we add the special tokens
config = AutoConfig.from_pretrained('ckiplab/gpt2-base-chinese',vocab_size=21128, model_max_len=512, is_fast=True,
                                    bos_token_id=base_tokenizer.bos_token_id,
                                    eos_token_id=base_tokenizer.eos_token_id,
                                    pad_token_id=base_tokenizer.pad_token_id,
                                    output_hidden_states=False)

# the pre-trained model is loaded with the custom configuration
base_model = AutoModelForCausalLM.from_pretrained('ckiplab/gpt2-base-chinese', config=config)

# the model embedding is resized

#config.vocab_size = base_tokenizer.vocab_size
base_model.resize_token_embeddings(len(base_tokenizer))
```

```
Embedding(21131, 768)
```

資料集劃分為 90% 用於訓練，10% 用於驗證。

```
[ ] df['poet_content'] = bos + ' ' + df['poet_content'] + ' ' + eos

    df_train, df_val = train_test_split(df, train_size = 0.9, random_state = 77)
    print(f'There are {len(df_train)} headlines for training and {len(df_val)} for validation')
```

```
There are 85 headlines for training and 10 for validation
```

```
model_headlines_path = '/content/drive/MyDrive/poet/CChiese_1000_new8_3'

training_args = TrainingArguments(
          output_dir=model_headlines_path,                  # output directory
        num_train_epochs=3,                              # total # of training epochs
        per_device_train_batch_size=3,     # batch size per device during training
        per_device_eval_batch_size=2,      # batch size for evaluation
        warmup_steps=25,                                 # number of warmup steps for learning rate scheduler
        weight_decay=0.01,                               # strength of weight decay
        logging_dir=model_headlines_path,                       # directory for storing logs
        prediction_loss_only=True,
        save_steps=1000
)
```

Transformers 提供了一個 Trainer 類來幫助您在自己的數據集上微調任何預訓練模型。
Trainer 提供了簡單的 API，讓使用者可以輕鬆地訓練和語言模型。同時，他也支援在多個 GPU/TPU 上進行分散式訓練！

TrainingArguments & Trainer
要使用 Trainer，建議先透過 TrainingArguments 設定好客製化的參數。這次有用到參數說明如下：

要使用 Trainer，建議先透過 TrainingArguments 設定好客製化的參數。這次有用到參數說明如下：

1.output_dir：model output 與 checkpoint 寫入的位置
2.evaluation_strategy： training 期間 model 評估的方法，有"no": 不評估、"steps"、"epoch": 每幾個 epochs 或每一次 epochs 就評估一次。如果設定為 "steps" 則需要在 eval_steps 設定次數
3.save_strategy：checkpoint 儲存的方式，設定選項同 evaluation_strategy
4.learning_rate：模型權重更新的效率
5.num_train_epochs：training 回合
6.weight_decay：抑制更新參數的幅度
7.fp16：是否使用 half-precision(fp16) 進行訓練

設定好了之後，將 args 、model 與 前面建立好的 dataset（前處理過的！）、tokenizer 放到 Trainer 進行訓練：

```
trainer = Trainer(
        model=base_model,                                    # the instantiated T
        args=training_args,                                  # training arguments, defined abo
        data_collator=data_collator,
        train_dataset=tokenized_train_dataset,               # training dataset
        eval_dataset=tokenized_val_dataset                   # evaluation dataset
)
trainer.train()
trainer.save_model()
base_tokenizer.save_pretrained(model_headlines_path)
```

```
The following columns in the training set don't have a corresponding argument in `GPT2LMHea
/usr/local/lib/python3.8/dist-packages/transformers/optimization.py:306: FutureWarning: Thi
  warnings.warn(
***** Running training *****
  Num examples = 88
  Num Epochs = 3
  Instantaneous batch size per device = 3
  Total train batch size (w. parallel, distributed & accumulation) = 3
  Gradient Accumulation steps = 1
  Total optimization steps = 90
  Number of trainable parameters = 102071040
[90/90 00:35, Epoch 3/3]
```

Save Model

訓練好模型之後，可以將模型存起來，之後需要時可以 load 回來使用。

```
[ ] # save whole model
    FILE = '/content/drive/MyDrive/poet/Cmodel_all.pt'
    torch.save(model, FILE)
```

```
[ ] trainer.evaluate()
```

```
The following columns in the evaluation set don't have a corresponding ar
***** Running Evaluation *****
  Num examples = 10
  Batch size = 2
[5/5 00:00]
{'eval_loss': 4.56458044052124,
 'eval_runtime': 0.2846,
 'eval_samples_per_second': 35.133,
 'eval_steps_per_second': 17.566,
 'epoch': 3.0}
```

Text generation strategies

```
def Top_p_nucleus_sampling():
    # text generation example
    generated_text_samples = headlines_model.generate(
        text_ids,
        max_length= 50,
        do_sample=True,
        top_k=100,
        top_p=0.92,
        temperature=0.8,
        repetition_penalty= 1.5,
        num_return_sequences= 3
    )

    for i, beam in enumerate(generated_text_samples):
      print(f"{i}: {headlines_tokenizer.decode(beam, skip_special_tokens=True)}")
      print()
```

模型分為兩部分：編碼器和解碼器。編碼器讀取輸入文字並傳回表示該輸入的向量。然後，解碼器會取得該向量並透過一次產生一個標記來產生對應的文字。模型生成文字有多種解碼策略。

Greedy

貪婪採樣是最直接的文本生成方法。在每一步中，語言模型都會選擇機率最高的標記作為序列中的下一個單字。雖然這種方法高效且易於實施，但它有一個明顯的缺點：它經常會導致重複的文字。該模型傾向於偏向最可能的標記，導致生成的輸出缺乏多樣性。

Beam Search

透過維護一組稱為束的候選序列來擴展貪婪搜尋。與貪婪搜尋相比，束搜尋通常會產生更多樣化和連貫的文本，但由於它選擇最佳可能的回應，因此它仍然可能會受到重複或通用輸出的影響。

Random Sampling with Temperature

隨機採樣為文字生成引入了令人興奮的驚喜元素。該模型不是總是選擇最可能的標記，而是根據其機率分佈隨機選擇下一個單字。此方法將隨機性注入生成過程，使模型能夠探索不同的路徑並產生更多樣化的輸出。然而，純隨機採樣可能過於不可預測，並且會產生不太連貫的文字。

溫標是一種巧妙的技術，可以微調隨機取樣的隨機性。透過調整“溫度”參數，可以控制機率分佈的形狀。

Top-K Sampling

Top-k 取樣解決了純隨機取樣中的不可預測性問題。該模型僅考慮前 k 個最可

能的標記，其中 k 可以是預先定義的數字或詞彙大小的百分比。透過丟棄不太可能的標記，該方法促進了生成序列之間的多樣性。 Top-k 取樣可確保模型探索受控的可能性子集，從而在隨機性和品質之間取得平衡。

Top-P Sampling（Nucleus Sampling）

Top-P（即核採樣）提供了比 Top-K 更動態的方法。它選擇累積機率超過閾值 P 的最小單字集。

Top-P 取樣也稱為核心取樣，它引入了一種動態方式來選擇下一個單詞，即從可能單字的「核心」（下一個單字的取樣子集）中進行選擇。這裡，P 表示累積機率閾值。

較高的 P 值：包含更多可能的單字，增加文字多樣性。
較低的 P 值：專注於範圍較窄、可能性較大的單字集，從而增強連貫性。
該方法透過根據單字的機率分佈動態調整所考慮單字的範圍來平衡創造力和連貫性。它在必須保持自然流暢但又需要一些驚喜或創新元素的場景中特別有用，例如創造性的說故事

- **Top-P Sampling (Nucleus Sampling)(此系統選擇此演算法生成文章、散文、詩詞)**

```python
# text generation example
generated_text_samples = headlines_model.generate(
    text_ids,
    max_length= 100,
    do_sample=True,
    top_k=0,
    temperature=0.90,
    num_return_sequences= 5
)

for i, beam in enumerate(generated_text_samples):
  print(f"{i}: {headlines_tokenizer.decode(beam, skip_special_tokens=True)}")
  print()
```

Python

```
0: 一粒星仔樣有爭議一粒星仔係星仔樣樣有爭議一粒星仔係一粒星仔樣有爭議一粒星仔係一粒星仔樣有爭議一粒星仔係一粒

1: 一粒星仔盒到手就籠到腳下。伯公伯公伯公伯公伯公伯公伯公伯公伯公伯公伯公伯公伯伯公伯伯伯伯伯伯伯伯伯伯公伯伯伯伯伯伯

2: 一粒星仔啼紅土香便來到土牛肚埂拋下土牛肚步步到哪頂想快樂到一隻星仔對煞車一番仔腳步。也有人喝了共在等地過目

3: 一粒星仔連枕頭就好像一隻大人仔蹺濕一圈就像一隻瓶在左上腳上等等等等一隻血流蜷取下等一匹白頭顆就像一隻雨水一

4: 一粒星仔栗打赤个蔥農會桌桌桌桌桌桌桌桌桌桌桌桌桌桌介白个菜籃，桐花香味糯米香米香勢濃肚抵消供過於求時間、習慣
```

但在生成過程 GPT2 模型還是會生成重複的詞語

```
0: 一 粒 星 仔 樣 有 爭 議 一 粒 星 仔 係 星 仔 樣 樣 有 爭 議 一 粒 星 仔 係 一 粒 星 仔 樣 有 爭 議 一 粒 星 仔 係 一 粒 星 仔 樣 有 爭 議 一 粒 星 仔 係 一 粒
1: 一 粒 星 仔 盒 到 手 就 籠 到 腳 下 。 伯 公 伯 公 伯 公 伯 公 伯 公 伯 公 伯 公 伯 公 伯 公 伯 伯 公 伯 伯 伯 伯 伯 伯 伯 伯 伯 伯 公 伯 伯 伯 伯 伯 伯 伯
2: 一 粒 星 仔 啼 紅 土 香 便 來 到 土 牛 肚 埂 拋 下 土 牛 肚 步 步 到 哪 頂 想 快 樂 到 一 隻 星 仔 對 煞 車 一 番 仔 腳 步 。 也 有 人 喝 了 共 在 等 地 過 目
3: 一 粒 星 仔 連 枕 頭 就 好 像 一 隻 大 人 仔 蹺 濕 一 圈 就 像 一 隻 瓶 在 左 上 腳 上 等 等 等 等 一 隻 血 流 蜷 取 下 等 一 匹 白 頭 顆 就 像 一 隻 雨 水 一
4: 一 粒 星 仔 栗 打 赤 个 蔥 農 會 桌 桌 桌 桌 桌 桌 桌 桌 桌 桌 桌 桌 桌 桌 介 白 个 菜 籃, 桐 花 香 味 糯 米 香 米 香 勢 濃 肚 抵 消 供 過 於 求 時 間 、 習 慣
```

這裡我使用 jieba 結巴中文斷詞，再把斷詞出來的詞語，例如尋找 5 條文章中重複文字最少的，和把出現連續的字詞，只刪除連續的部分

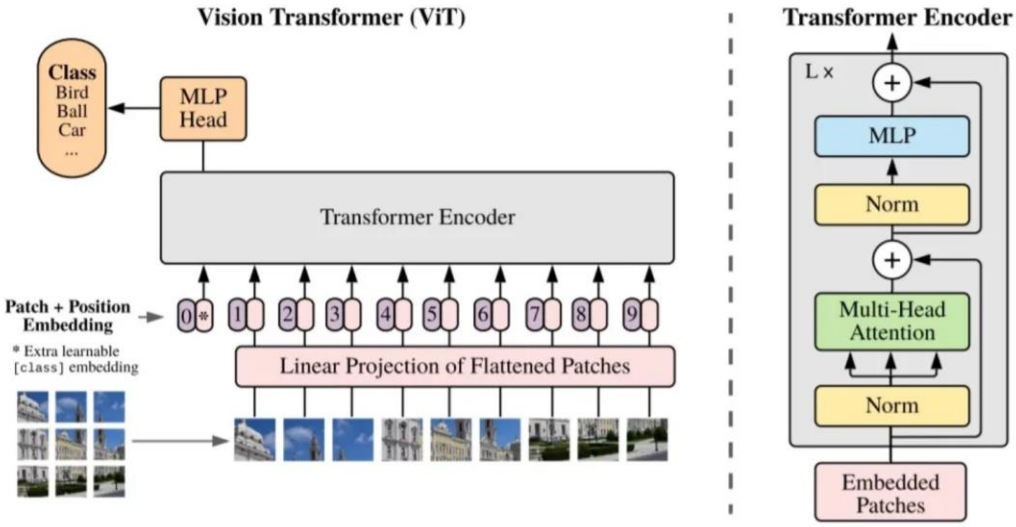
```python
def peot_generation(model_headlines_path,text):
    model_headlines_path = 'C:\\Users\\user\\Desktop\\literature_project\\AI_MODEL\\GPT_WRITE_POET\\'+ model_headlines_path
    headlines_model = AutoModelForCausalLM.from_pretrained(model_headlines_path)
    headlines_tokenizer = BertTokenizerFast.from_pretrained(model_headlines_path)
    text = text
    text_ids = headlines_tokenizer.encode(text, return_tensors = 'pt')

    generated_text_samples = headlines_model.generate(
    text_ids
    )
    #generated_text_samples
    # text generation example
    generated_text_samples = headlines_model.generate(
        text_ids,
        max_length= 150,
        do_sample=True,
        top_k=100,
        top_p=0.92,
        temperature=0.8,
        repetition_penalty= 1.5,
        num_return_sequences= 5
    )
    peot = []
    count_list = []
    for i, beam in enumerate(generated_text_samples):
        peot.append(f"{headlines_tokenizer.decode(beam, skip_special_tokens=True)}".replace(" ",""))
    for j in peot:
        jieba.case_sensitive = True # 可控制對於詞彙中的英文部分是否為case sensitive, 預設False
        seg_list = jieba.cut(j.replace(" ",""))
        list_jieba = list(seg_list)
        set_jieba = set(list_jieba)
        count_jieba = len(list_jieba) - len(set_jieba)
        count_list.append(count_jieba)
    p = count_list.index(min(count_list))
    return peot[p]
```

- 介面操作

此介面我 Find_tune、["創世紀詩詞風格", "原住民詩詞風格", "客家詩詞風格", "新月詩詞風格","新詩詩詞風格","現代詩詞風格","笠詩詞風格","藍星詩詞風格"]8 個模型風格選取，標題、作者按下送出

圖像生成文章

Vision Transformer 原理




```python
# Get Model
model = VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
# Get Image Feature Extractor
feature_extractor = ViTImageProcessor.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
# Get Tokenizer Model
tokenizer = AutoTokenizer.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

# Apply model on GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# The maximum length the generated tokens can have
max_length = 16
#  Number of beams for beam search
num_beams = 4
# Generation Config
gen_kwargs = {"max_length": max_length, "num_beams": num_beams}
```

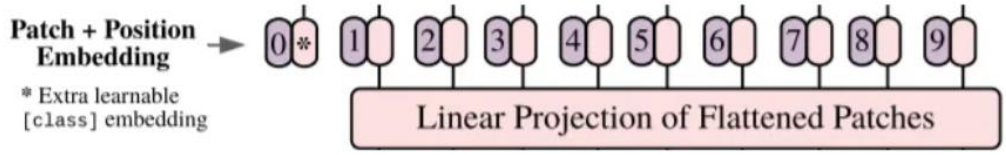
1. 將圖片轉成序列化資訊 (Split image)

為了將一張影像變成一串序列編碼，我們需要把 HxWxC 的影像變成 Nx(P^2x C)。以下圖為例，假設我們有一張寬(W)和高(H) 16 X 16 的彩色影像(C=3)。Patch size 表示為 (P, P) 範例中使用 4 X 4 大小的 patch。N 表示 pacth 的總數量，其計算方式為 $N=HW/P^2$

2. Position embedding

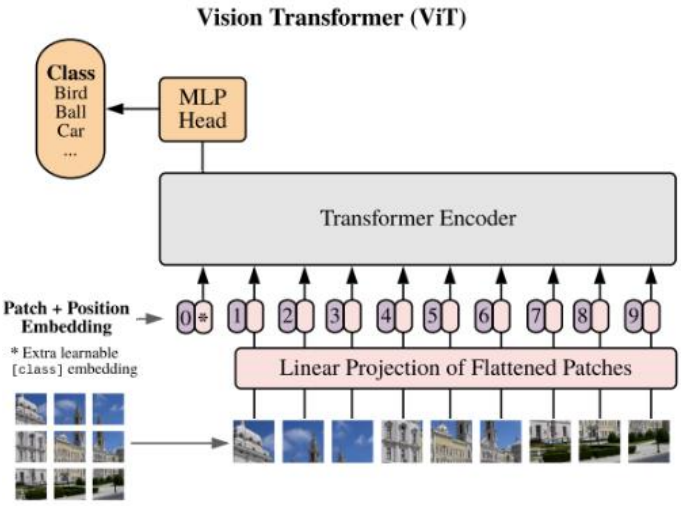
由於每個 patch 在整張影像中是有順序性的，因此我們需要為這些 patch embedding 向量添加一些位置的資訊。如圖所示，將編號 0~9 的紫色框表示各個位置的 position embedding(編碼方式是透過神經網路學習)，而紫色框旁邊的粉色框則是上一部所提到的經過 linear projection 後的 patch embedding 向量。最後將每個 patch 的紫框和粉框相加後正式得到 Embadded Patches 的

輸出。

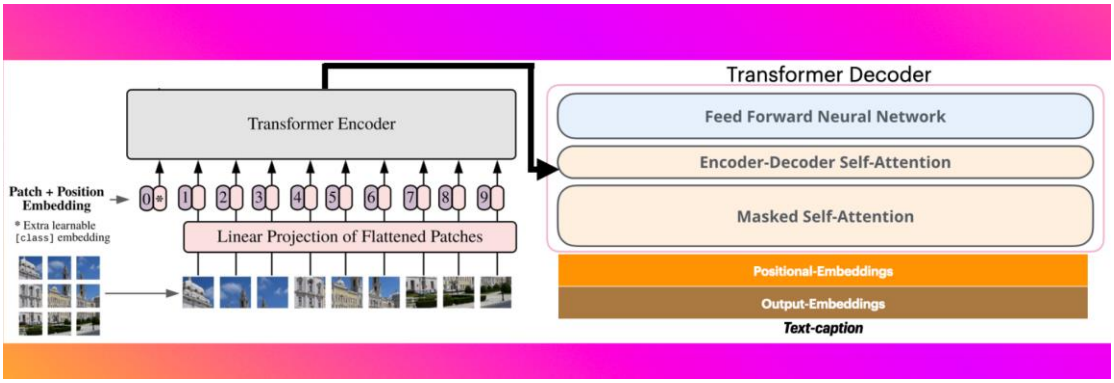


輸出交由下游任務

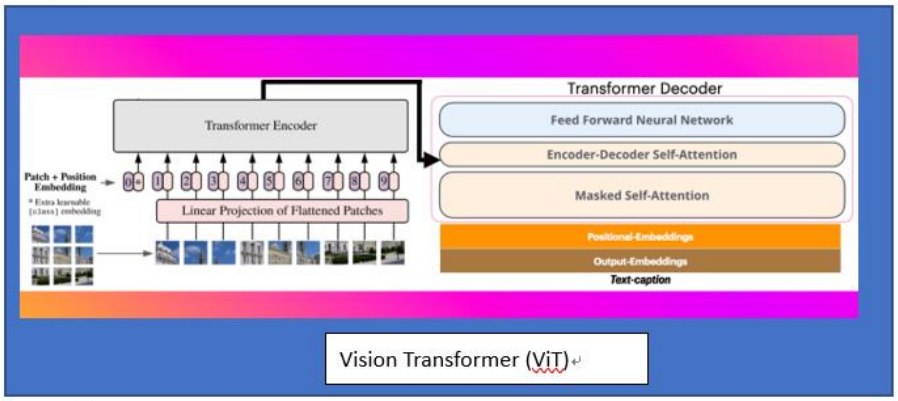
例如最後要進行影像的分類，將經過 N 個 block 後得到的輸出僅拿取其中的 [CLS] token Encode 後的結果，也就是 z^0_L。將它丟入 MLP 最後再接 softmax 產生出每個 class 的機率輸出預測結果。



如果要生成文本而非分類任務可以在接 Transformer Decoder(vit-gpt2)




```
# Get Model
model = VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
# Get Image Feature Extractor
feature_extractor = ViTImageProcessor.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
# Get Tokenizer Model
tokenizer = AutoTokenizer.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
```



把圖像生成文字

```
import googletrans

translator = googletrans.Translator()
translation = translator.translate(predict_step([image])[0], dest='zh-tw')
translation.text
```

```
'樹上一簇紅白相間的花'
```

介面選取生成文章風格後並把圖像生成的句子送進 Find-tune 好的模型生成文章

```
def image_to_text_gpt(image,category):
    poet_style = ["創世紀詩詞風格", "原住民詩詞風格", "客家詩詞風格", "新月詩詞風格","藍星詩詞風格"]
    translator = googletrans.Translator()
    translation = translator.translate(predict_step([image])[0], dest='zh-tw')
    context = geration_poet.peot_generation(str(category),translation.text)

    return context
```

GPT2

原住民詩詞風格 花

2024年10月9日 14:36 105_104546102702.jpg

樹上一簇紅白相間的花,它們都用手指緊握著他,開始抖動。如果我看到牠們搖晃時我聽見這群小鳥猛然傾聽而來像是煙草般迅速飛馳在人煙稀少下只剩下一朵蜜蜂發出陣聲,呼叫喚醒說:「你好!」他把焦點放給嘴巴味道真正對話的語言能夠成為未知數理學者?還有一定程度地掌控中心,忽略了自己親身體力及思念力;另外也可以充

介面操作:

此介面我 Find_tune、["創世紀詩詞風格", "原住民詩詞風格", "客家詩詞風格", "新月詩詞風格","新詩詩詞風格","現代詩詞風格","笠詩詞風格","藍星詩詞風格"]8 個模型風格選取，填寫路徑、標題、作者按下送出