

# LSTM&Transformer 音樂生成

專題學生:劉冠岑

柯尚維

張守元

指導教授：葉慶隆 教授



大同大學  
資訊工程學系  
專題報告

中華民國 112 年 1 月

LSTM & Transformer 音樂生成

大同大學資工系專題報告 劉冠岑 柯尚維 張守元

大同大學  
資訊工程學系  
專題報告

LSTM&Transformer音樂生成

劉冠岑

柯尚維

張守元

經考試合格特此證明

專題考試委員

指導教授

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

中華民國 112 年 1 月 3 日

## 致謝

耗時將近一年半之久的畢業專題將要拉下帷幕，至此報告完成之際，謹向我們尊敬的指導老師葉慶隆教授致以誠摯的感謝和崇高的敬意。在這一年半的時間裡，您在課務繁忙之餘還抽空擠出時間指導我們一步一步完成我們的專題。

在初期決定要做什麼專題方向時，老師給了我們很大的幫忙，給了我們很多方向的建議和可行性，讓我們得以順利完成畢業專題。

在此我們祝福教授工作與生活都一切順心。

## 摘要

在這個注重智慧財產權的時代，一些影片或是網站的配樂常常會因侵權而導致被下架，本專題便希望能透過當下熱門的深度學習來製作出一個自動生成音樂的系統，除了能有效地規避上述的問題之外，還可以根據使用者的需求來生成不同風格的音樂。

首先我們研究 KernScore 網站上的音樂資料(.krn 檔案)，用 Music21 音樂函式庫轉換成 Long Short-Term Memory (LSTM)模型的輸入資料，經過反覆的訓練後，產生出一段旋律，再將其輸入 Magenta 團隊開發的 Transformer 之 encoder 和 decoder 模型生成配樂並將兩者結合成最後的音樂。

經過多次的測試，目前我們團隊使用了包括德國、中國、英國等國家的民謠和著名音樂家巴赫、舒伯特等人的作品進行生成並產生出了帶有不同風格的音樂，未來我們將會跳出古典音樂的範圍，往生成出更現代的音樂（例如：電音三太子）的方向前進。

# 目錄

致謝.....	I
摘要.....	II
目錄.....	III
圖目錄.....	VI
表目錄.....	VIII
<b>第一章    概述.....</b>	<b>9</b>
1.1    專題動機 .....	9
1.2    專題目標及做法 .....	2
1.3    專案管理 .....	2
<b>第二章    相關文獻與研究.....</b>	<b>5</b>
2.1    資料集 .....	5
2.2    LSTM(LONG SHORT-TERM MEMORY) .....	6
2.3    TRANSFORMER.....	6
<b>第三章    系統簡介 .....</b>	<b>10</b>
3.1    專題相關內容介紹 .....	10
3.2    資料集準備和前處理 .....	10
3.3    訓練、測試和整合階段 .....	11
3.4    使用者介面設計 .....	12
<b>第四章    系統實作與呈現.....</b>	<b>13</b>
4.1    資料準備及前處理 .....	13

4.1.1	讀取資料集.....	13
4.1.2	音調之處理.....	13
4.1.3	資料集中符號之處理.....	15
4.1.4	建立單一資料集.....	17
4.1.5	建立對應之 Map .....	18
4.1.6	One-Hot Encoding .....	19
4.2	神經網路模型建立、編譯，訓練 .....	22
4.2.1	LSTM 模型訓練和生成之原理.....	22
4.2.2	設定參數和建立模型.....	23
4.2.3	訓練模型.....	23
4.3	音樂生成 .....	24
4.3.1	前置作業.....	24
4.3.2	生成 .....	24
4.4	TRANSFORMER 模型在本專題中之應用 .....	26
4.4.1	Transformer 應用方式.....	26
4.4.2	Transformer 應用步驟.....	27
4.5	DJANGO 環境建置 .....	28
4.5.1	開啟空白頁面.....	29
4.5.2	安裝 Django .....	30
4.5.3	建立專案.....	30
4.5.4	移動目錄.....	30
4.5.5	設置端口.....	31
4.5.6	設置可造訪 Django 服務器的網址 .....	31
4.5.7	運行遠端 Django 服務器 .....	31
4.6	DJANGO 架構.....	32

4.6.1	MVC 架構.....	32
4.6.2	MTV 架構.....	33
4.7	DJANGO 前端與後端.....	35
4.7.1	建立應用程式與模板.....	35
4.7.2	訊息應對與傳送.....	35
4.8	網頁呈現.....	39
<b>第五章</b>	<b>結論.....</b>	<b>41</b>
<b>參考文獻</b> .....		<b>42</b>



# 圖目錄

圖 2.1 RNN 和 LSTM 架構，圖片來源: UNDERSTANDING LSTM NETWORKS -- COLAH'S BLOG .....	6
圖 2.2 TRANSFORMER 中 ENCODER 架構，圖片來源: 1706.03762.PDF (ARXIV.ORG).....	8
圖 2.3 TRANSFORMER 中 DECODER 架構，圖片來源: 1706.03762.PDF (ARXIV.ORG).....	9
圖 4.1 科學音高表示法 .....	13
圖 4.2 西洋音樂之音調 .....	14
圖 4.3 轉調範例(轉調前) .....	15
圖 4.4 轉調範例(轉調後) .....	15
圖 4.5 MIDI NOTE 轉化參考表 .....	16
圖 4.6 拍子表示範例 .....	16
圖 4.7 序列形式檔案 .....	17
圖 4.8 單一資料集 .....	18
圖 4.9 統計符號並轉化成數字之 MAP .....	19
圖 4.10 ONE-HOT ENCODING 範例 .....	20
圖 4.11 資料前處理程序 .....	21
圖 4.12 LSTM 模型運作原理 .....	22
圖 4.13 選擇輸入檔案 .....	27
圖 4.14 生成伴奏 .....	27
圖 4.15 音樂生成程序 .....	28
圖 4.16 建立新的記事本 .....	29
圖 4.17 安裝 DJANGO 套件 .....	30
圖 4.18 建立新的 DJANGO 專案 .....	30
圖 4.19 創立好的專案資料夾 .....	30
圖 4.20 移動目錄 .....	31

圖 4.21 設置端口 .....	31
圖 4.22 設置可造訪 DJANGO 服務器的網址 .....	31
圖 4.23 運行遠端 DJANGO 服務器-1 .....	32
圖 4.24 運行遠端 DJANGO 服務器-2 .....	32
圖 4.25 MVC 架構圖 .....	33
圖 4.26 MTV 架構圖 .....	33
圖 4.27 新增應用程式資料夾 .....	35
圖 4.28 新增 TEMPLATE 資料夾 .....	35
圖 4.29 URLS.PY 內部的程式碼 .....	36
圖 4.30 VIEWS.PY 內部部分程式碼 .....	36
圖 4.31 GENERATION.HTML 檔案中的部分程式 .....	37
圖 4.32 VIEWS.PY 內的程式 .....	38
圖 4.33 VIEWS.PY 內的 PREDICT_LSTM 函式 .....	38
圖 4.34 網站首頁 .....	39
圖 4.35 個別資料集介紹 .....	39
圖 4.36 不同資料集之生成結果 .....	40
圖 4.37 使用者選擇資料集之下拉式選單 .....	40
圖 4.38 音樂生成結果呈現之介面圖 .....	40

## 表目錄

表 1.1 時間進度表 .....	2
表 1.2 工作分配表 .....	3

# 第一章 概述

## 1.1 專題動機

在這個資訊發達、多媒體興盛的時代，在網路上充斥著許多的作品，像是音樂、影片...等等的存在著智慧財產的創作。因此，我們上傳東西到網路上時可能會因為版權而導致問題的發生。像是上傳影片到臉書或者 Instagram 時會因為 影片的背景音樂版權問題而被自動下架影片。這對於常在使用臉書、Instagram 的人來說是個非常困擾的問題。因為保護智慧財產權的意識抬頭，所以就算網路上有許多的素材是可以下載但卻沒有辦法上傳使用的！身為常在使用社交平台發文的我們也因為這個問題困擾了許久，因此，我們就思考著有沒有什麼辦法可以解決這個問題。於是我們就想到了有沒有可能由我們自己來創作屬於自己音樂的可能性，這樣就能解決因為背景音樂版權而無法上傳到社交平台的窘境。因緣際會下，在系上的選修課程中剛好有學習到能夠利用電腦去生成音樂的方法，因此我們就想說或許我們可以嘗試看看利用所學去解決我們所遇到的問題！因此就有了我們 現在呈現出來的這個作品。

在研究初期，早期的文獻都是使用 LSTM 來生成音樂[1]，而隨著研究的進行，我們發現使用 Transformer 來處理時序型的資料是現今的主流，本組就想說是否可以將 LSTM 和 Transformer 進行結合來生成音樂。

## 1.2 專題目標及做法

本專題將製作一個藉由 LSTM 和 Transformer 之 encoder 和 decoder 結合去做到音樂的生成。在現今，無論是大人或小孩幾乎都是人手一機，而社群媒體又以臉書和 Instagram 為大宗。近幾年因為對於智慧財產權的保護意識抬頭，所以臉書和 Instagram 對於貼文的審查也越來越重視，這也導致有許多使用者會因此而受到限制無法上傳貼文，像是影片背景音樂親權之類的問題。為此本專題研究了能夠讓電腦自己生成音樂的方法，其中透過 LSTM 和 Transformer 之 encoder 和 decoder 去做音樂生成，首先是讀取 Humdrum 檔案格式的資料集，轉化成電腦可處理的數字形式、轉換音調、彙整成單一檔案、轉化成模型訓練和生成所需的 One-Hot Encoding 向量格式等前處理步驟，並使用處理好的資料輸入 LSTM 模型進行訓練，完成訓練後我們會將一段數字序列當成生成的依據輸入 LSTM 模型中生成主旋律，將主旋律轉換成 MIDI 檔案後輸入至 Megenta 團隊提供的 Transformer 之 encoder 和 decoder 程式碼[2] 為主旋律生成伴奏並和主旋律結成本專題最終的生成結果，各步驟詳細介紹和虛擬碼請見第四章。

## 1.3 專案管理

本專題執行時間從 2022 年 7 月至 2022 年 12 月，總共 6 個月，進度參見(表 1.1)。專題成員為劉冠岑、柯尚維、張守元。

表 1.1 時間進度表

作業名稱	7 月	8 月	9 月	10 月	11 月	12 月
------	-----	-----	-----	------	------	------

研讀相關資料						
研究 LSTM 模型						
資料前處理						
訓練 LSTM 模型						
架設網站						
優化網站及生成結果						
系展準備						
撰寫專題報告						

表 1.2 工作分配表

組員	分配項目
劉冠岑	製作網頁功能
	資料前處理
	音樂模型訓練
	研究 LSTM 模型
	研究 Transformer 模型
柯尚維	文獻蒐集
	專題海報構思與製作
	研究 Transformer 模型
	論文製作
張守元	研究 LSTM 模型
	資料前處理
	專題簡報製作

## 第二章 相關文獻與研究

在原先做的 LSTM 音樂生成後，因為覺得生成出來的成品略顯單調，有許多地方需要加強，因此本章將參考其他文獻內容找出值得學習的部分，以做我們的參考並做為未來能改進之方法。

### 2.1 資料集

本專題使用 KernScores 作為資料集來源[3]，KernScores 是一個提供 Humdrum 檔案格式的音樂庫網站，Humdrum 檔案格式為一種類似可擴展標記語言(XML)的文字檔案格式，音樂中的音調、音符、休止符號等資訊會以文字的方式儲存在檔案中，但因為其為文字檔案，並不能直接拿來做處理，所以我們需要透過 music21 函式庫[5]將其讀入並轉換成能進行資料集前處理的數字檔案格式。

KernScores 網站中包含了各國民謠和一些知名音樂家的創作，共超過 10 萬首音樂提供瀏覽者免費下載，使用者還可以透過網站上音樂介紹，了解歌曲的背景故事和創作靈感，本專題分別選用了包誇歐洲地區:英國 363 首、德國 564 首、法國 638 首等，美洲地區:美國 213 首、加拿大 285 首，亞洲地區:中國 292 首，等多國的傳統名謠和數位知名作曲家的作品，總計 2000 多首音樂做為訓練資料集。



## 2.2 LSTM(Long Short-Term Memory)

LSTM 是一種時間循環神經網路，相較 RNN，LSTM 更能避免長期記憶流失的問題(圖 2.1)，在 RNN 的架構中，可以看到在每個時間點，所有先前的資料都會經過 activation function 的運算，而重要的資訊經過多次的運算，其值會逐漸變小而失去影響力，而 LSTM 架構中最核心的設計就是設計了一條類似捷徑的通路，並不會經過 activation function 的運算，使重要資訊能很好的傳遞下去，這也使 LSTM 比起 RNN 更適合處理自然語言和音樂方面的生成。

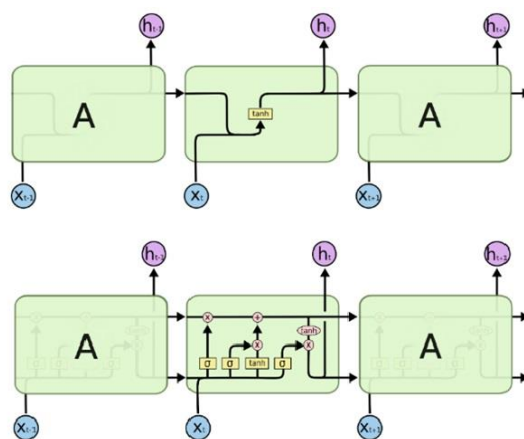


圖 2.1 RNN 和 LSTM 架構，圖片來源: [Understanding LSTM Networks -- colah's blog](https://colah.github.io/posts/Understanding-LSTMs/)

## 2.3 Transformer

本專題加入了 Megenta 團隊提供的 Transformer 之 encoder 和 decoder 模型使生成之音樂更加豐富完整[2]，Magenta 聲稱其在生成音樂上比傳統的 LSTM 模型更加出色[8]。

Transformer 中的 encoder 由多個相同的層組成(圖 2.4)，每個層都具有兩個子層[4]：多頭自注意力機制 (multi-head self-attention) 和前饋神經網路 (feed-

forward neural network)。多頭自注意力機制用於將輸入序列中的詞彙之間建立關聯，並計算每個詞彙對於其他詞彙的重要程度。前饋神經網絡則用於進一步處理這些關聯，提取更高層次的特徵表示。

每一層的輸入是前一層的輸出，這樣逐層堆疊下去，使得模型能夠對輸入序列進行多次的信息提取。在訓練過程中，encoder 會通過反向傳播算法學習適合特定任務的特徵表示。經過多層的處理後，encoder 的輸出特徵向量將被傳遞給 decoder 進行後續的生成或預測任務。

相比起 LSTM 模型，Transformer 的 encoder 具有幾個優勢。首先，自注意力機制能夠在不同位置的詞彙之間捕捉長距離的相互關係，有效解決了長期記憶流失的問題。其次，encoder 具有並行計算的能力，可以加快模型的訓練速度。

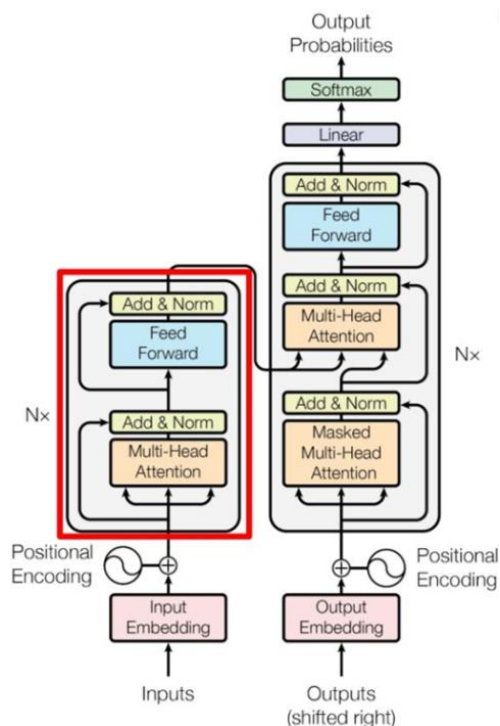


圖 2.2 Transformer 中 encoder 架構，圖片來源: [1706.03762.pdf \(arxiv.org\)](https://arxiv.org/pdf/1706.03762.pdf)

decoder 是 Transformer 中另一個重要架構(圖 2.5)，decoder 接受來自 encoder 的輸出特徵向量作為初始狀態，並透過多層的自注意力機制和前饋神經網絡進行處理，生成一個與目標序列相關的輸出。與 encoder 類似，decoder 由多個相同結構的層堆疊而成，每一層也包含自注意力機制和前饋神經網絡。

在訓練過程中，decoder 的輸入包括目標序列的前一個詞彙以及來自 encoder 的輸出特徵向量，逐步生成目標序列，並將生成的詞彙與目標詞彙進行比對，計算生成的詞彙的損失。這樣，通過反向傳播，模型可以學習如何生成符合目標序列的結果。在生成階段，decoder 則根據生成的概率分佈選擇最有可能的詞彙作為預測結果。

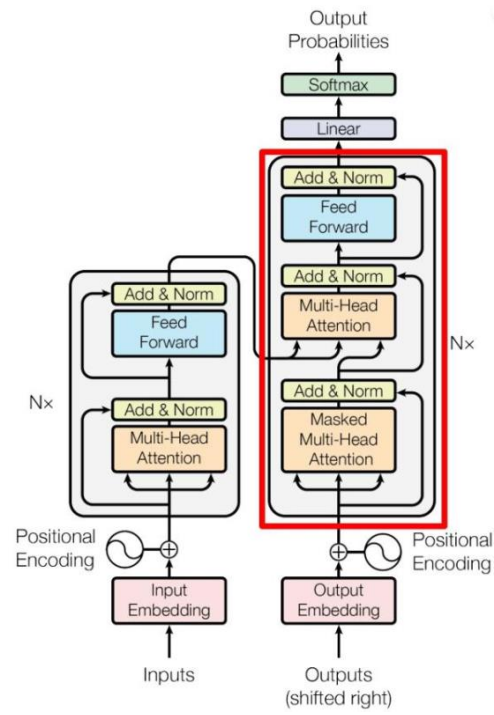


圖 2.3 Transformer 中 decoder 架構，圖片來源: [1706.03762.pdf \(arxiv.org\)](https://arxiv.org/pdf/1706.03762.pdf)

## 第三章 系統簡介

### 3.1 專題相關內容介紹

本專題之研究流程可分為資料前處理(3.2 小節)、模型訓練和音樂生成(3.3 小節)和使用 Django 網頁框架建設使用者介面。其中，讀者可將音樂生成步驟想成是，輸入一段音樂至本專題的自動生成音樂裝置中，此裝置會生成出完整且可以使用電腦將其播放出來的音樂。

我們進一步介紹上述的自動生成音樂裝置，它是由 LSTM 模型和 Megenta 團隊提供的 Transformer 之 encoder 和 decoder 模型所組成，而輸入的一段音樂在本專題中是用一串數字序列 input 所表示，例如: "30\_60\_55\_41\_\_54\_"，數字代表音符，"\_"代表拍子，輸入至 LSTM 模型中，LSTM 模型會根據數字序列生成一個機率最高的輸出，列如:"61"，並更新 input 成"30\_60\_55\_41\_\_54\_61"，並重複此動作直到達成終止條件，並將 input 轉換成 MIDI 檔案，形成本專題的單音主旋律，再將其輸入至 Transformer 之 encoder 和 decoder 模型中生成出伴奏並和單音主旋律結合成最終生成結果並輸出，因為輸出的檔案格式為 MIDI 檔案格式，所以可以直接用電腦播放出來，詳細介紹和虛擬碼請見 3.3 小節和 4.3.2 小節。

### 3.2 資料集準備和前處理

將 Humdrum 檔案從 Kernscores 網站下載後，首先使用 music21 函式庫[5]將其讀入並轉換成 MIDI 檔案格式，再處理音樂的音調，將所有資料集中的音

樂轉換成 C 大調或者是 a 小調，以減少訓練所需的資料量，接著處理音樂中的符號，除了音符之外的符號，像是休止符號、拍子等等，我們需要使用自訂的符號來代表它們，完成後我們需要建立單一資料集，將所有的音樂彙整成一個完整的資料集，接著是建立對應之 map，我們需要將單一資料集中所有出現的不同符號進行編號，最後使用上一個步驟建立的 map 將資料轉換成 One-Hot Encoding 的格式。

### 3.3 訓練、測試和整合階段

經過資料前處理後，我們得到兩筆資料，分別是經過 One-Hot Encoding 的 inputs 和真正的值，將這兩筆資料輸入 LSTM 模型中進行訓練，訓練的模式為 LSTM 模型會根據 inputs 中的一段資料生成出下一個輸出為何，計算出兩者差距並對模型中的參數進行調整，使預設的精準度得到提升，LSTM 模型的訓練完成後，我們會將一段經過 One-Hot Encoding 的資料輸入 LSTM，就如同訓練時的步驟一樣，LSTM 會生成出下一個輸出為何，但此時已經不需要透過真正的值進行比對，將預測的輸出加入到原先的資料中進行下一個輸出的預測，直到達成結束條件為止。

最後將 One-Hot Encoding 的資料轉換成 MIDI 格式，形成本專題的單音主旋律，再將主旋律輸入至 Megenta 團隊提供的 Transformer 之 encoder 和 decoder 程式碼[2]為主旋律生成伴奏並和主旋律結成本專題最終的生成結果，最後經由本校大約 20 幾位同學的試聽來判斷音樂的品質是否符合預期。

### 3.4 使用者介面設計

在本專題架設的網站上，除了首頁介紹的本專題之概述、動機和研究方法之外。使用者還可以透過本組的資料集介紹頁面來了解本組所使用的音樂分別來自什麼國家和作曲家，而從其個別的解說中使用者也能對該國家或作曲家獨有的風格有更進一步的認識。

在網站的第三個網頁上，呈現了本組的生成結果，本組分別使用了三個國家和三個作曲家作為代表，分別為：

國家：(1) 德國 (2) 加拿大 (3) 中國

作曲家：(1) 巴赫 (2) 舒伯特 (3) 史蒂芬·福斯特

總共有六組，每組分別有兩段音樂，分別是只使用了 LSTM 模型生成的主旋律和經過 LSTM 模型和 Transformer 之 encoder 和 decoder 模型生成出的主旋律加上伴奏，使用者可以從中感受出加上伴奏後音樂的變化和不同國家和作曲家之間的風格差異。

在使用者對網站上的內容有所了解後，可以使用網站上提供的音樂生成功能來生成出屬於自己的音樂。首先第一步先選擇自己喜歡的音樂資料集，接著按下 PREDICT MUSIC 按鍵，而使用者不需要輸入任何資料，等待約 30 秒到 2 分鐘後便可完成音樂的生成，如果使用者有需要的話，還可以按下方的 DOWNLOAD 鍵將生成出來的結果下載下來。

## 第四章 系統實作與呈現

本章將介紹本專題資料前處理、LSTM 模型之訓練、音樂生成、Transformer 之 encoder 和 decoder 模型之使用和用運 Django 架設專題網站之過程和虛擬碼。

### 4.1 資料準備及前處理

#### 4.1.1 讀取資料集

開始訓練和生成之前，需要對本專題中所使用的資料集進行一連串的處理，在 KernScores 將所需之資料下載後，首先要先將其內容使用 music21 中的 converter 函式轉換成「科學音高表示法」(圖 4.1)。



圖 4.1 科學音高表示法

#### 4.1.2 音調之處理

本專題使用之資料集為西洋音樂，而在西洋音樂裡，所有的大調加上小調總共有 24 種不同的音調(圖 4.2)，如果要將所有音調都納入本專題的資料集中會使得蒐集資料集成為一項非常困難的工作，而如果資料集中某些特殊音調的音樂出現的次數太少，也會造成後續訓練和生成上的問題，本專題採取了另一種方式來解決資料及蒐集上的困難——轉調。所謂的轉調就是將資料集中出現的所有的大調和小調分別轉換成單一的大調和小調，本專題選擇將所有的大調



轉換成最常出現的 C 大調，而小調則是轉換成 a 小調。

```
def transpose(song):  
  parts <-- 從 song 中使用 getElementsByClass 所取得所有的 part  
  measures_part0 <--從 parts 的第一個 part 中使用 getElementsByClass 取得  
  所有 measure  
  if measures_part0[0][4]存在  
    Key <--measures_part0[0][4]  
  else  
    Key <--用 song.analyze 分析音調  
  if key 的 mode 為"major"  
    將音樂轉換成 C 大調  
  else  
    將音樂轉換成 a 小調  
  Return
```

所謂的轉調只是將音調做調整，並不會影響到整首音樂的旋律，範例可參考將 B 大調(參見圖十四)轉化成 C 大調(圖 4.3)，可以很清楚的看到轉調後的樂譜，開頭的降 Si 和降 Mi 記號消失了，這就表示成功將音調轉換成所需要的 C 大調了。

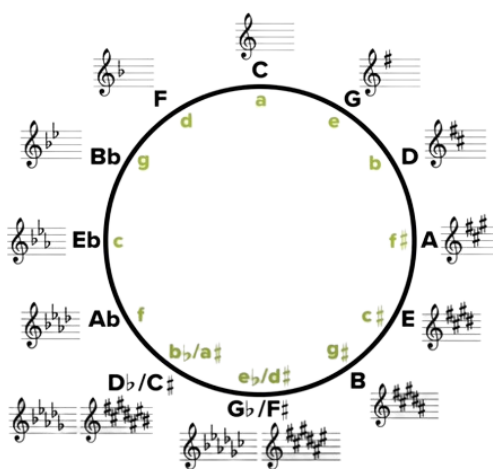


圖 4.2 西洋音樂之音調



圖 4.3 轉調範例(轉調前)



圖 4.4 轉調範例(轉調後)

### 4.1.3 資料集中符號之處理

本小節要處理樂譜上會出現的一些符號，首先，在 4.1.1 小節中所介紹的「科學音高表示法」並不能當作後續進行訓練和生成時 LSTM 模型的輸入資料，需要將其轉化成 MIDI note 表示法(圖 4.5)。

而在所有出現的符號當中，最為重要的就是如何使用符號來表示拍子，我們都知道不同樣式的音符有著不同的拍子，像是四分音符，八分音符等等，而我們該怎麼在電腦裡表示拍子呢？本專題使用“\_”符號來表示拍子，範例(圖 4.6)，可以看到高音譜記號後表示為 6/8 拍，意思是指每一個小節有六拍，八分音符為一拍，範例中第一個小節中的第一個音符為四分音符，在本範例中為兩拍來使用，所以表示方法為「69 \_ \_ \_」，下一個為標準的八分音符，本範例中為一拍來使用，表示方法為「71 \_」。

接著將資料集中每一首音樂依照上述的對應方式轉化成 MIDI note 和拍

子符號、休止符號轉換成一個一個獨立的序列形式檔案，每一個檔案中以

MIDI note 對應音符、“r” 對應休止符號、“\_” 對應拍子(圖 4.7)。

```

Def encode_song(song, time_step=0.25):
    初始化一個空的編碼列表 (encoded_song)
    for 樂譜中的每一個音符和休止符 do:
        if 當前元素是音符
            將其轉換為 MIDI 數字，並將其添加到編碼列表中
        if 當前元素是休止符
            將其轉換為字串 "r"，並將其添加到編碼列表中
    steps <-- 當前樂譜的時間長度
    for step in steps:
        if step 為 0
            將音符加入 encoded_song 中
        else
            添加一個 "_" (表示當前時間步長沒有音符) 到編碼列表
    中
    return encoded_song

```

Octave	Note numbers											
	Do	Do#	Re	Re#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	0	1	2	3	4	5	6	7	8	9	10	11
1	12	13	14	15	16	17	18	19	20	21	22	23
2	24	25	26	27	28	29	30	31	32	33	34	35
3	36	37	38	39	40	41	42	43	44	45	46	47
4	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79	80	81	82	83
7	84	85	86	87	88	89	90	91	92	93	94	95
8	96	97	98	99	100	101	102	103	104	105	106	107
9	108	109	110	111	112	113	114	115	116	117	118	119
10	120	121	122	123	124	125	126	127				

圖 4.5 MIDI note 轉化參考表



圖 4.6 拍子表示範例

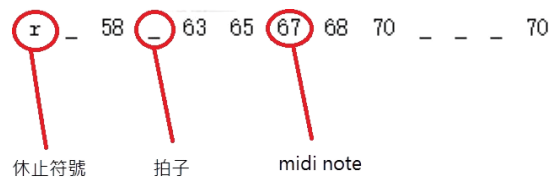


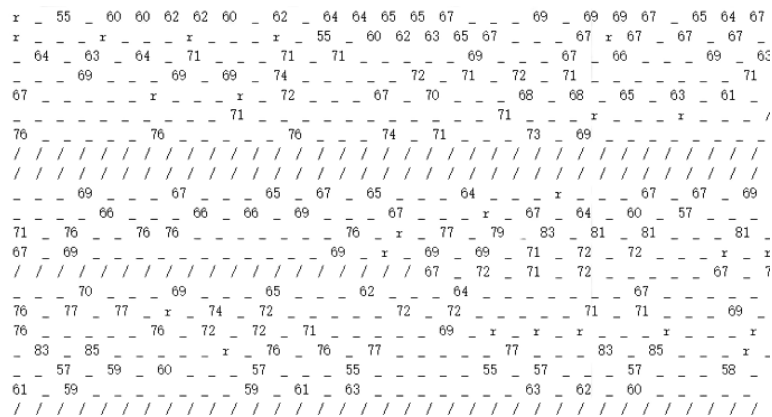
圖 4.7 序列形式檔案

#### 4.1.4 建立單一資料集

經過上一小節處理後，資料集中的每一首音樂都已經轉換成序列形式的檔案，本小節將介紹如何將所有轉化好的檔案彙整成一個單一資料集。

要將所有轉化好的檔案彙整成一個單一資料集，需要在序列和序列間加上適當的符號來將其分開，本專題使用 / 做為間隔符號，將一段序列加入至下列程式碼中所定義之 songs 中後，加入間隔符號，再將下一段序列加入，重複操作後便可得到所需要之單一資料集(圖 4.8)。

```
def create_single_file_dataset(dataset_path, file_dataset_path, sequence_length):
    new_song_delimiter <-- "/" * sequence_length
    初始化一個空的字串 songs
    for file in 資料路徑:
        song <-- 載入的資料
        songs <-- songs + song + new_song_delimiter
    將單一資料集寫回輸入文件路徑
    return songs
```



#### 4.1.5 建立對應之 Map

```
def create_mapping(songs, mapping_path):
    初始化一個空的字典 mappings
    vocabulary <-- 所有在 songs 中出現過的符號
    for i in vocabulary:
        將 i 加入 vocabulary 中
    將 mappings 寫回

def convert_songs_to_int(songs):
    初始化一個空的列表 int_songs
    載入 mappings
    for symbol in songs:
        將用 mappings 對應後的數字加入 int_songs
    return int_songs
```

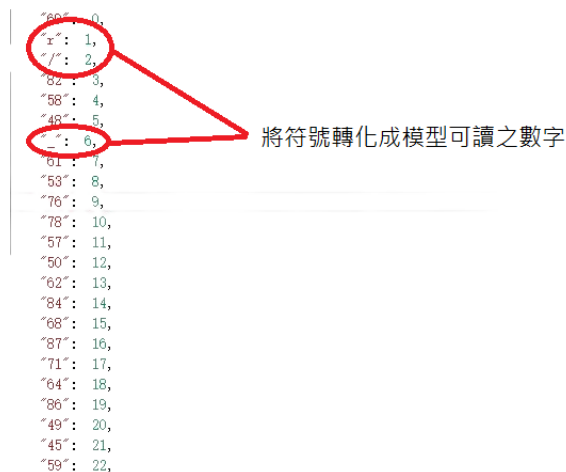


圖 4.9 統計符號並轉化成數字之 Map

#### 4.1.6 One-Hot Encoding

經過一連串的处理，最後一步是要以矩陣的形式呈現訓練資料，也就是所謂的 One-Hot Encoding，簡單來說，需要兩個不同的輸入，第一個為需要以向量形式表示的 inputs，可以將它視為考卷，其形式為一個三維陣列(圖 4.10)，第一列從右到左有三個參數，分別是「34」代表總共出現多少個不同的符號、「64」代表要根據前面多少個符號進行預測，這部分在下一章中會有更詳細的講解、「1」代表 input 中的資料數量，舉例來說，如果今天資料序列中總共有 100 個符號，那 input 中的資料數量會是「100 - 64」共 36 筆資料，第一筆為[0 : 63]、第二筆為 [1 : 64]，以此類推，而第二個是不需要以向量形式表示的 targets，可以將它視為答案，LSTM 型是依據先前的資料來推斷下一個資料為何，所以它會需要一個對照來審視自己的預測結果是否正確，這就是 targets 的功用。

```
def generate_training_sequences(sequence_length):
    songs <-- 載入單一資料集
```



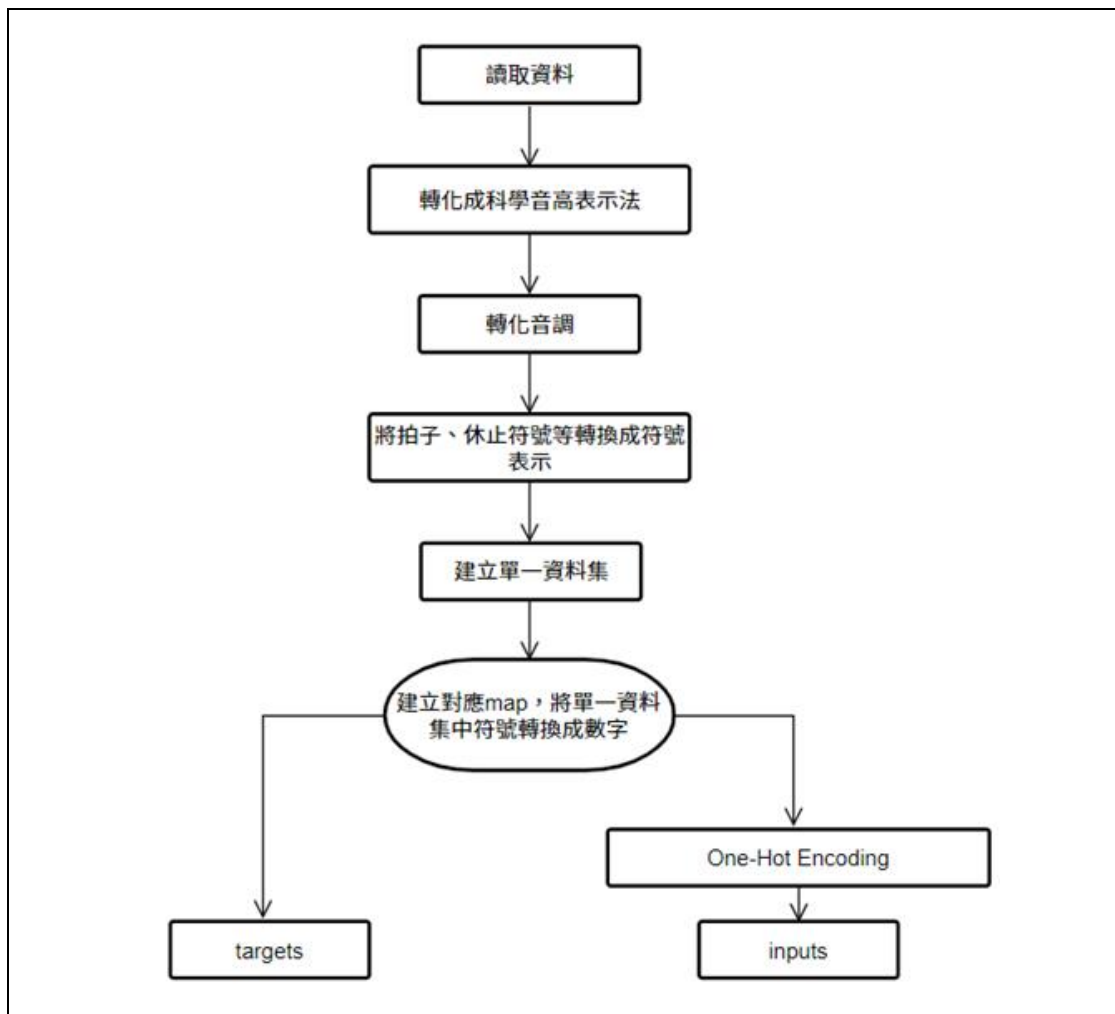


圖 4.11 資料前處理程序



## 4.2 神經網路模型建立、編譯，訓練

### 4.2.1 LSTM 模型訓練和生成之原理

本小節將會介紹 LSTM 模型訓練和生成之原理，首先先參考下列語句：

「他是個美國人，所以他會說 \_\_\_\_」，在不失一般性的前提下，我們可以依據語句中的「美國人」來判斷出空格處應該為「英文」，而音樂跟上述的語句形式類似，都為所謂的「時間序列」，LSTM 模型正是運用此原理，根據先前資料來預測下一個出現的音符為何(圖 4.12)，那究竟是要根據多遠的資料來進行預測呢？經過長時間的研究後，本專題選擇根據前面 64 個符號來預測下一個符號，這也就是 4.1.6 小節中第二個參數「64」所代表之涵義。

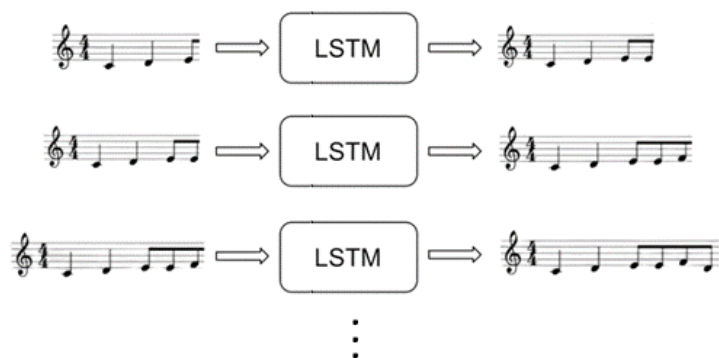


圖 4.12 LSTM 單音生成原理

#### 4.2.2 設定參數和建立模型

在開始訓練之前，需要先建立訓練所需之模型並且設定相關參數，包括有幾種模型輸出的可能、模型的層數、一層要有多少 units、訓練的次數、模型的儲存路徑等等，並將設定好的參數輸入建立模型的函示中，並設定優化器和 dropout 的比例，即可得到建置好的模型。

```
OUTPUT_UNITS <-- 36
NUM_UNITS <-- 256
LOSS <-- "sparse_categorical_crossentropy"
LEARNING_RATE <-- 0.001
EPOCHS <-- 40
BATCH_SIZE <-- 64
SAVE_MODEL_PATH <--
"/content/gdrive/MyDrive/LSTM_genMusic/model/model.h5"

def build_model(output_units, num_units, loss, learning_rate):
    建立輸入層
    加入一層 LSTM 層
    加入一層 20% 的 dropout 層
    加入一層輸出層，將激活函數設為 softmax
    使用指定的損失函數並將優化函數設為 Adam
    列印出 model 數據
    return model
```

#### 4.2.3 訓練模型

建立好模型之後，首先呼叫 4.1.6 小節中所介紹的 `generate_training_sequences` 函式產生出 `input` 和 `target` 兩個序列，並將其輸入模型進行訓練，經過大約 20 分鐘後，即可得到訓練完成的模型，本專題中所有的模型的訓練流程皆和上述步驟相同

```
def train(output_units=OUTPUT_UNITS, num_units=NUM_UNITS, loss=LOSS,
learning_rate=LEARNING_RATE):
inputs, targets <-- generate_training_sequences(SEQUENCE_LENGTH)
model <-- build_model(output_units, num_units, loss, learning_rate)
訓練模型
將訓練好的模型儲存
```

## 4.3 音樂生成

### 4.3.1 前置作業

在開始生成音樂之前，我們需要先將已經訓練完成的模型和用於對應的

map 載入。

```
def __init__(self,
model_path="/content/gdrive/MyDrive/LSTM_genMusic/model/model.h5") :
self.model_path <-- model_path
self.model <-- keras.models.load_model(model_path)
self._mappings <-- json.load(fp)
self._start_symbols <-- [" "] * SEQUENCE_LENGTH
```

### 4.3.2 生成

我們在 4.2.1 小節中解釋過 LSTM 模型訓練的原理，而生成音樂的原理跟

模型訓練的原理相同，都是用現有的資訊去預測下一個音符為何，而這邊會

產生一個問題，那就是第一個被預測出來的音符要用什麼作為根據呢？答案

非常簡單，我們需要先輸入一小段資料作為其依據，在本專題中稱為「種

子」。

生成的第一個步驟為處理種子，種子之格式範例為「67 \_ 72 \_ 67 \_ 65 \_

67 \_ \_ \_ 」，我們可以從範例中看出，種子的格式和 4.1.3 小節中處理完的資料相同，這也代表著，再輸入種子後，我們必須要將種子依照 4.1.4 小節到 4.1.6 小節中的內容進行相同的處理。

在處理完種子後即可將其輸入模型進行生成，但在這之前，值得一提的是，generate\_melody 函式中的第四個參數 temperature 為本專題設計上的一個小巧思，我們都知 LSTM 模型的運作原理為根據先前的資料預測出下一個「最有可能」的輸出，這樣預測固然沒有問題，但本專題希望在預測的過程中能加入一點「隨機」的概念，可以讓最後生成出的音樂更加有趣，temperature 這個參數之功用即為調整機率，將其設定成 0 到 1 之間的數字，0 為完全按照預測結果，1 則為完全隨機。

```
def _sample_with_temperature(self, probabilitites, temperature):
    predictions <-- np.log(probabilitites) / temperature
    probabilitites <-- np.exp(predictions) / np.sum(np.exp(predictions))
    choices <-- range(len(probabilitites))
    index <-- np.random.choice(choices, p=probabilitites)
    return index
```

接下來就是生成的步驟，LSTM 模型會根據輸入的資料預測出下一個符號為何，並且將種子更新，接著判斷預測的符號是否為結束符號 / 或者預測次數已超過我們設定的 num\_steps 如果滿足兩個條件之中的一個便結束生成，如果沒有的話就重複這個步驟，直到結束，我們便可得到一段由 LSTM 模型生成出的音樂，但此時的音樂還只有旋律，我們還需要用下一章將會介紹的

Transformer 之 encoder 和 decoder 模型幫其生成伴奏。

```
def generate_melody(self, seed, num_steps, max_sequence_length, temperature):
    seed <-- seed.split()
    melody <-- seed
    seed <-- self._start_symbols + seed
    seed <-- [self._mappings[symbol] for symbol in seed]#將種子轉化成數字
    for i in num_steps:
        seed <-- seed[-max_sequence_length:]
        onehot_seed <-- keras.utils.to_categorical(seed,
num_classes=len(self._mappings))
        probabilities <-- self.model.predict(onehot_seed)[0]
        output_int <-- self._sample_with_temperature(probabilities, temperature) # 更新種子
        seed.append(output_int)
        output_symbol <-- [k for k, v in self._mappings.items() if v == output_int][0]
        if output_symbol 為 "/"
            break
    melody.append(output_symbol)
    return melody
```

到這邊 LSTM 模型生成的步驟已經完成，但為了讓生成的音樂可以更好的呈現出來，我們決定將其轉化成可以撥放的 MIDI 檔案格式，而這個步驟需要大量的使用 music21 函式庫的內容，詳細程式碼請見 <https://github.com/jerry762>

## 4.4 Transformer 模型在本專題中之應用

### 4.4.1 Transformer 應用方式

經過上一小節，一段音樂已經被生成出來了，但此時的音樂還只是一段旋律，比起其他音樂還說還是過於單調，本組決定使用 Magenta 團隊開發出的 Transformer 之 encoder 和 decoder 模型來生成出配樂，其有著獨特的 self

attention 機制，在生成方面會比 LSTM 模型來的更加出色。

#### 4.4.2 Transformer 應用步驟

本專題從 Magenta 開發之程式碼中擷取 Transformer 之 encoder 和 decoder 模型[2]並加至本專題中，使用方式非常簡單，將上一小節中轉化成功之 MIDI 檔案輸入，Transformer 之 encoder 和 decoder 模型會將輸入的音樂旋律進行優化並加上伴奏(圖 4.13、圖 4.14)，到這裡，本專題的生成步驟已經全部結束，詳細流程圖可以參考下方「音樂生成程序」(圖 4.15)。

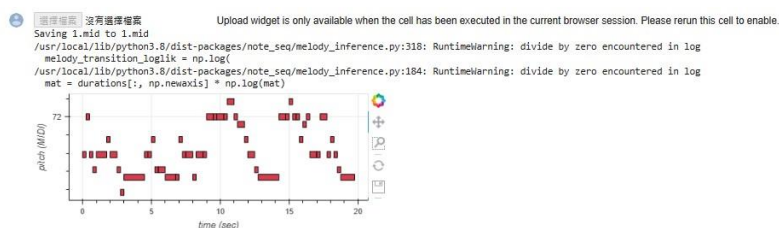


圖 4.13 選擇輸入檔案

##### ▼ Generate Accompaniment for Melody (生成伴奏)

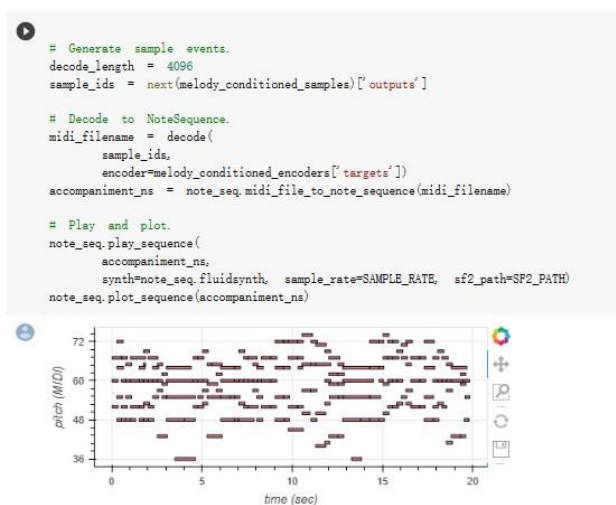


圖 4.14 生成伴奏

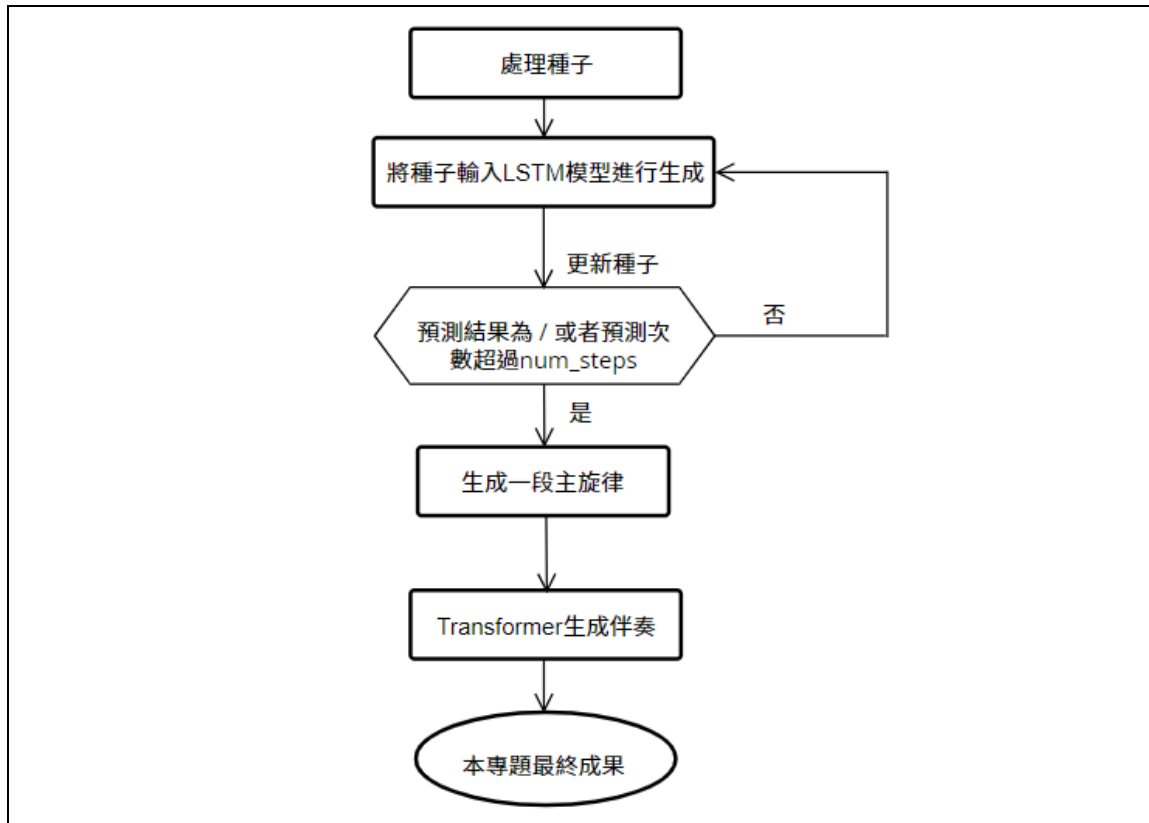


圖 4.15 音樂生成程序

## 4.5 Django 環境建置

為了使本專題能更好的呈現，我們使用了 Django 這個網頁框架，這個網頁框架的特別之處在於一般網頁的後端是使用 PHP 撰寫，而 Django 的後端是使用 Python 撰寫，由於這個特點，我們可以把我們之前訓練好的模型放入後端執行，已達成使用者只要在前端網頁上操作，就可以間接執行我們訓練好的模型，然後生成出音樂。

想像一下，如果專題做完想要呈現專題給別人看時有兩種方式，第一種方式是直接打開 IDE(整合開發環境)跑給別人看，第二種方式是把整個專題放在網站上供其他人瀏覽使用。想必使用第二種方式呈現專題的效果絕對比第一種方式要來的優，一來我們可以在網頁上介紹我們專題與資料集等等的資

訊，二來也可在網頁上做一些互動性的功能。為了達成這個目的，我們想到最快的方式就是直接去學網頁框架。但網頁框架有幾十種到底要怎麼選呢？因為我們的專題與 AI 有關，寫的程式碼都是 Python，所以我們找的網頁框架勢必要能與 Python 程式碼相容，我們找到兩個網頁框架 Flask、Django 都能支援 Python，但 Django 社群比較大且網路資源也比較多，所以最終我們選擇了 Django 網頁框架[6]。

Django 這個網頁框架的特別之處在於一般網頁的後端是使用 PHP 撰寫，而 Django 的後端是使用 Python 撰寫，由於這個特點，我們可以把我們之前訓練好的模型放入後端執行，已達成使用者只要在前端網頁上操作，就可以間接執行我們訓練好的模型，然後生成出音樂。

因本專題整個開發環境都是使用 Google Colab，所以我們 Django 框架也選擇安裝在 Colab 上[7]，以下將介紹如何在 Google Colab 上安裝 Django。

#### 4.5.1 開啟空白頁面

你可以上 Google 搜尋 Colab(你可能還需登入你的 Google 帳號)，進入之後建立新的記事本。

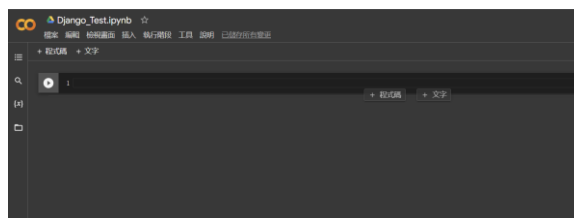


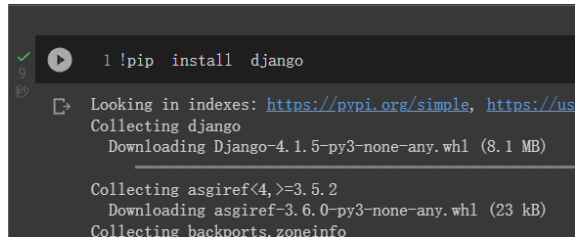
圖 4.16 建立新的記事本



## 4.5.2 安裝 Django

在程式碼編輯區輸入以下程式安裝 Django 套件。

`!pip install Django`



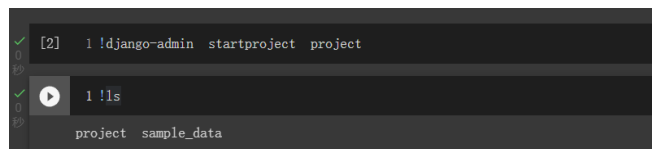
```
1 !pip install django
Looking in indexes: https://pypi.org/simple, https://us
Collecting django
  Downloading Django-4.1.5-py3-none-any.whl (8.1 MB)
Collecting asgiref<4,>=3.5.2
  Downloading asgiref-3.6.0-py3-none-any.whl (23 kB)
Collecting backports.zoneinfo
```

圖 4.17 安裝 Django 套件

## 4.5.3 建立專案

輸入以下指令建立新的 Django 專案，建立好之後可以看到剛創立好的專案資料夾。

輸入指令：`!django-admin startproject project`



```
[2] 1 !django-admin startproject project
1 !ls
project sample_data
```

圖 4.18 建立新的 Django 專案

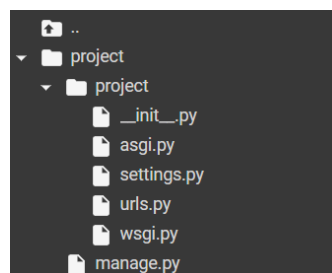


圖 4.19 創立好的專案資料夾

## 4.5.4 移動目錄

我們需要移動到 `project` 資料夾裡面，因為裡面有一個檔案叫 `manage.py`，這個檔案包含了許多事先寫好的腳本，我們只要使用這些腳本，

就可以跟遠端的伺服器做溝通。



圖 4.20 移動目錄

#### 4.5.5 設置端口

我在下面的代碼中提供了一個端口，我們需要使用相同的端口號，每當運行 Django 服務器時，我們需要從本地機器訪問服務器的鏈結。

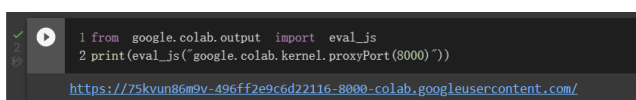


圖 4.21 設置端口

#### 4.5.6 設置可造訪 Django 服務器的網址

因為我們是靠 Colab 跑 Django 的服務器，所以我們需要讓遠端的 Django 服務器知道來源才能做拜訪，在 Project 資料夾裡有一個檔案叫 Setting，裡面有一行程式 ALLOWED\_HOSTS = []，把 Colab 的網址加進去。輸入指令:ALLOWED\_HOSTS = ['colab.research.google.com']

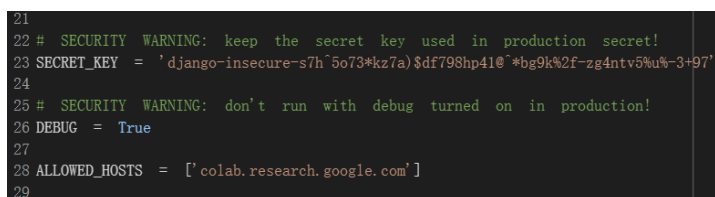


圖 4.22 設置可造訪 Django 服務器的網址

#### 4.5.7 運行遠端 Django 服務器

以上步驟都做完之後再新增以下程式!python manage.py runserver 8000，執行此程式之後點選 1.1.5 的連結就可訪問預設好的網站。

```
project sample_data

[5] 1 %cd project/

/content/project

[6] 1 from google.colab.output import eval_js
2 print(eval_js('google.colab.kernel.proxyPort(8000)'))

https://75kvun86m9v-496ff2e9c6d22116-8000-colab.googleusercontent.com/

1 !python manage.py runserver 8000
```

圖 4.23 運行遠端 Django 服務器-1

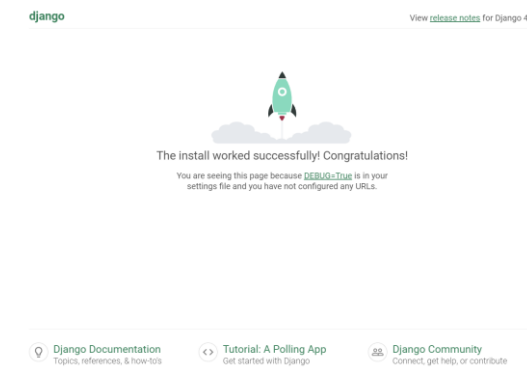


圖 4.24 運行遠端 Django 服務器-2

## 4.6 Django 架構

如何設計一個程式的結構，這是一門專門的學問，叫做架構模式（architectural pattern），屬於程式設計的方法論。

### 4.6.1 MVC 架構

MVC 模式就是架構模式的一種，MVC 模式的目的是實作一種動態的程式設計，使後續對程式的修改和擴充功能簡化，並且使程式某一部分的重複利用成為可能。除此之外，此模式通過對複雜度的簡化，使程式結構更加直覺。

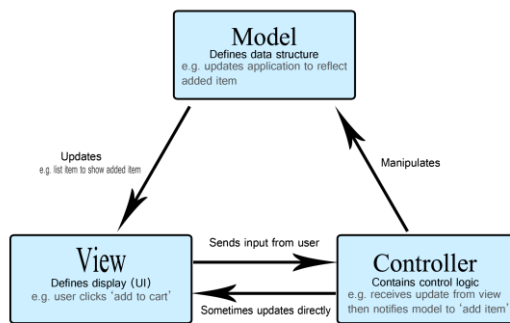


圖 4.25 MVC 架構圖，圖片來源: <https://developer.mozilla.org/en-US/docs/Glossary/MVC>

- 模型 Model - 程式設計師編寫程式應有的功能、資料庫專家進行資料管理和資料庫設計。
- 視圖 View - 介面設計人員進行圖形介面設計。
- 控制器 Controller - 負責轉發請求，對請求進行處理。

#### 4.6.2 MTV 架構

MTV 主要目標是使得開發複雜、資料庫驅動的網站變得簡單。它注重組件的重用性和「可插拔性」，敏捷開發和 DRY 法則（Don't Repeat Yourself）。

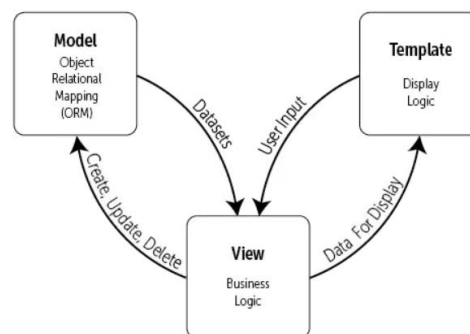


圖 4.26 MTV 架構圖，圖片來源: <https://djangobook.com/mdj2-django-structure/>

- 模型 Model - 程式設計師編寫程式應有的功能、資料庫專家進行資料管理和資料庫設計。
- 模板 Template - 生成頁面展現給使用者的部分，也就是我們看見的前端 HTML、CSS、JavaScript 的部分。
- 視圖 View - 處理業務邏輯、封裝結果的部分，負責處理 URL 與 callback 函式之間的關係。

## 4.7 Django 前端與後端

本節將會介紹在 Django 框架上撰寫內部程式，包括前、後端如何做整合、設置網頁的 URL、如何做到訊息應對、如何使用 Django 內部 API 等設定。

### 4.7.1 建立應用程式與模板

原本專案資料夾只有提供一些系統檔案，需要輸入 `!python manage.py startapp MusicAPP` 在專案資料夾內新增應用程式資料夾，裡面存放管理應用程式的檔案。

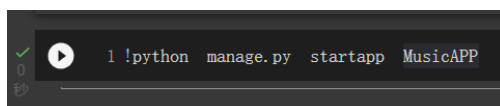


圖 4.27 新增應用程式資料夾

另外也在專案資料夾新增 `template` 資料夾，裡面存放的檔案基本上都是 `html` 檔，為網頁前端顯示部分。

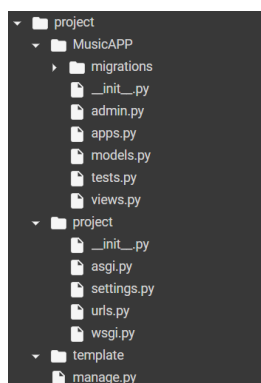


圖 4.28 新增 `template` 資料夾

### 4.7.2 訊息應對與傳送

此小節將介紹 `project` 資料夾中 `urls.py`、`views.py`、`template` 內的 `html` 檔的內部運作原理。

- **Urls.py**

```
16 from django.contrib import admin
17 from django.urls import path
18 from TestApp import views
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('', views.index, name='index'),
23     path('music_generation/predict_lstm', views.predict_lstm, name='predict_lstm'),
24     path('music_dataset/', views.music_dataset, name='music_dataset'),
25     path('music_generation/', views.music_generation, name='music_generation'),
26     path('music_home/', views.music_home, name='music_home'),
27 ]
```

圖 4. 29 urls.py 內部的程式碼

在 urls.py 內部程式碼中(圖 4.29)，path 函數的第一個參數是網頁的相對路徑，Django 服務器會根據你給的相對路徑去尋找該檔案在網路上的位置，第二個參數是去呼叫寫在 views.py 裡的函式，只要 path 函式被觸發(點擊連結)，他就會跑到 views.py 內部呼叫該對應的函式執行，第三個參數給的是這個網頁的名稱。

- **Views.py**

```
184 @csrf_exempt
185 def predict_lstm(request):
186     if request.method == 'POST':
187         key = int(request.POST.get('models'))
188     else:
189         print("Not catch the POST value")
190
191     # print(SAVE_LSTM_MODELS_PATH[key])
192     # print(MAPPINGS_PATH[key])
193
194     mg = MelodyGenerator(SAVE_LSTM_MODELS_PATH[key], MAPPINGS_PATH[key])
195
196     seeds = [
197         "67 - 67 - 65 64 - 64 - 64 - _",
198         "60 - 60 - 67 - 67 - 69 - 69 - 67 - _",
199         "67 - 72 - 67 - 65 - 67 - _ - _",
200         "64 - 62 - 60 - 62 - 64 - 64 - 64 - _",
201         "60 - 60 - 67 - 67 - 69 - 69 - 67 - _"]
202
203     rand_int = random.randint(0, 4)
204
205     melody = mg.generate_melody(seeds[rand_int], 500, SEQUENCE_LENGTH * 2, 0.3)
206     mg.save_melody(melody)
207
208     return render(request, 'generation.html')
209
210 @csrf_exempt
211 def music_dataset(request):
212     return render(request, 'dataset.html')
```

圖 4. 30 views.py 內部部分程式碼

在 views.py 內部部分程式碼中(圖 4.30)，可以看到有定義兩個函式分別為 predict\_lstm 與 music\_dataset，只要有人點擊網頁中某個連結，django 服務器就會去呼叫 urls.py 裡對應路徑的 path 函式，然後再根據

path 函式的第二個參數去呼叫在 views.py 裡對應的函式，例如：假設有有人在 index.html 網頁上點擊 dataset 的按鈕，它就會去呼叫 path('music\_dataset/', views.music\_dataset, name="music\_dataset") 函式，再去執行這個函數的第二個參數跑到 views.py 的 music\_dataset 函式執行，最後在使用渲染器(render)把 template 裡的 dataset.html 回傳，使用者便能看到更新後的網頁了。

- Template

Django 中的模板之所以強大在於模板變數與其他內部的 API，使其可以在網頁與網頁之間互相傳遞訊息與處理訊息。

```
<div class="m-0">
  <form method="post" enctype="application/x-www-form-urlencoded" action="predict_lstm">
    {% csrf_token %}
    <select name="models" class="form-select form-select-lg mb-5" required>
      <option selected disabled value="">Select the models</option>
      <option value="0">Africa</option>
      <option value="1">America</option>
      <option value="2">British</option>
      <option value="3">Canada</option>
      <option value="4">China</option>
      <option value="5">France</option>
      <option value="6">Germany</option>
      <option value="7">Ireland</option>
      <option value="8">Luxembourg</option>
      <option value="9">Pentatonic Scale</option>
      <option value="10">Poland</option>
    </select>
    <div class="text-center">
      <button class="btn btn-primary text-uppercase fs-5" id="submitButton"
        type="submit">Predict Music</button>
    </div>
    <!-- Submit Button -->
  </form>
</div>
```

圖 4. 31 generation.html 檔案中的部分程式

以上程式碼便是 template 資料夾中的 generation.html 檔案中的部分程式，這段程式是處理選擇模型的邏輯，首先需要將 html <form> tag 中設定一些參數，傳表單的方式使用 post、加密的部分是預設加密、把表單的傳送目的地設為 views.py 內的函式 predict\_lstm，然後在 html <select> tag 給一個名稱，將來要使用這個名稱才能把資料(模型編號)



讀出來，如此一來就可以把表單中的資料(模型編號)傳送給另一支程式執行。

```
16 MAPPINGS_PATH = [Path.joinpath(APP_BASE_DIR, 'mappings/mapping_Africa.json'),
17                  Path.joinpath(APP_BASE_DIR, 'mappings/mapping_America.json'),
18                  Path.joinpath(APP_BASE_DIR, 'mappings/mapping_British.json'),
19                  Path.joinpath(APP_BASE_DIR, 'mappings/mapping_Canada.json'),
20                  Path.joinpath(APP_BASE_DIR, 'mappings/mapping_China.json'),
21                  Path.joinpath(APP_BASE_DIR, 'mappings/mapping_France.json'),
22                  Path.joinpath(APP_BASE_DIR, 'mappings/mapping_Germany.json'),
23                  Path.joinpath(APP_BASE_DIR, 'mappings/mapping_Ireland.json'),
24                  Path.joinpath(APP_BASE_DIR, 'mappings/mapping_Luxembourg.json'),
25                  Path.joinpath(APP_BASE_DIR, 'mappings/mapping_Pentatonic.json'),
26                  Path.joinpath(APP_BASE_DIR, 'mappings/mapping_Poland.json')]
27
28
29 SAVE_LSTM_MODELS_PATH = [Path.joinpath(APP_BASE_DIR, 'models/model_Africa.h5'),
30                           Path.joinpath(APP_BASE_DIR, 'models/model_America.h5'),
31                           Path.joinpath(APP_BASE_DIR, 'models/model_British.h5'),
32                           Path.joinpath(APP_BASE_DIR, 'models/model_Canada.h5'),
33                           Path.joinpath(APP_BASE_DIR, 'models/model_China.h5'),
34                           Path.joinpath(APP_BASE_DIR, 'models/model_France.h5'),
35                           Path.joinpath(APP_BASE_DIR, 'models/model_Germany.h5'),
36                           Path.joinpath(APP_BASE_DIR, 'models/model_Ireland.h5'),
37                           Path.joinpath(APP_BASE_DIR, 'models/model_Luxembourg.h5'),
38                           Path.joinpath(APP_BASE_DIR, 'models/model_Pentatonic.h5'),
39                           Path.joinpath(APP_BASE_DIR, 'models/model_Poland.h5')]
40
```

圖 4.32 views.py 內的程式

在 views.py 內的程式(圖 4.32)，分別有兩個 list，一個是存模型路徑，一個是存模型的 map，每個模型需要用自己的 map 才能正常生成音樂，只要有模型編號(index)就可以根據 list 存取到正確的模型與 map 路徑。

```
184 @csrf_exempt
185 def predict_lstm(request):
186     if request.method == 'POST':
187         key = int(request.POST.get('models'))
188     else:
189         print("Not catch the POST value")
190
191     # print(SAVE_LSTM_MODELS_PATH[key])
192     # print(MAPPINGS_PATH[key])
193
194     mg = MelodyGenerator(SAVE_LSTM_MODELS_PATH[key], MAPPINGS_PATH[key])
195
196     seeds = [
197         "67 - 67 - 67 - 65 64 - 64 - 64 -",
198         "60 - 60 - 67 - 67 - 69 - 69 - 67 -",
199         "67 - 72 - 67 - 65 - 67 - - -",
200         "64 - 62 - 60 - 62 - 64 - 64 - 64 -",
201         "60 - 60 - 67 - 67 - 69 - 69 - 67"]
202
203     rand_int = random.randint(0, 4)
204
205     melody = mg.generate_melody(seeds[rand_int], 500, SEQUENCE_LENGTH * 2, 0.3)
206     mg.save_melody(melody)
207
208     return render(request, 'generation.html')
```

圖 4.33 views.py 內的 predict\_lstm 函式

在 views.py 內的 predict\_lstm 函式(圖 4.33)，剛剛傳送的表單內容會到此，首先使用 request.method 檢查是否有資料傳送過來，如果有的話就使用 request.POST.get() 函式，括弧內需要給名稱，這個函式會回傳模

型編號，再把編號當作 index 就可以正確的選擇模型然後生成音樂，

最後在使用渲染器(render)把 template 裡的 generation.html 回傳，使用

者便能聽到剛生成的音樂與看到更新過後的網頁了。

## 4.8 網頁呈現

經過上述的介紹，在本章節中將架設的網頁頁面呈現給讀者參考，其中包括網站首頁(圖 4.34)、個別的音樂資料集介紹(圖 4.35)、不同資料集之生成出來的音樂結果(圖 4.36)、使用者選擇資料集之下拉式選單(圖 4.37)，以及音樂生成結果呈現之介面圖(圖 4.38)。



圖 4.34 網站首頁



圖 4.35 個別資料集介紹

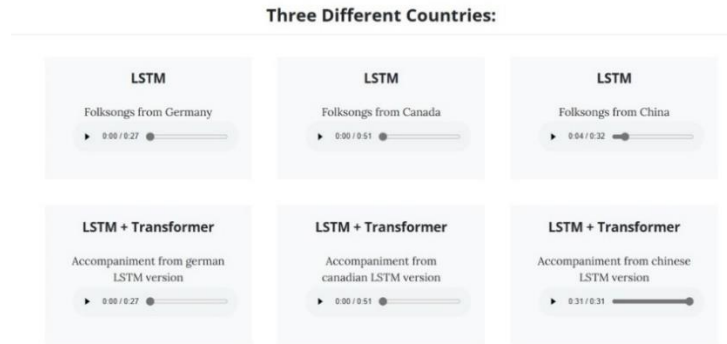


圖 4.36 不同資料集之生成結果

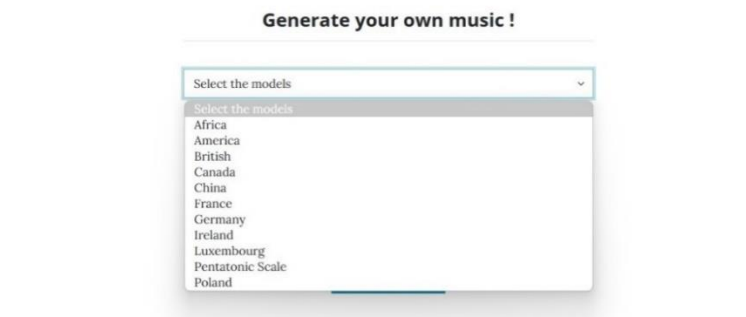


圖 4.37 使用者選擇資料集之下拉式選單



圖 4.38 音樂生成結果呈現之介面圖

## 第五章 結論

本專題利用 LSTM 和 Transformer 的 encoder 和 decoder 模型製作出一個自動生成音樂裝置，並選用了簡單的鋼琴民謠做為資料集，在學習、自行操作的過程經歷了許多各種不同的問題，最終在大家一起討論、修改下順利的完成了本次的專題，雖然只是一個小型的研究，但本組還是從中看到了其中發展的潛能，未來若增加下列功能，生成的成品將會更加多樣、豐富。

- 增加更多不同風格的音樂。
- 開發出多樣化的樂器，像是吉他、鼓.....等其他不同種類的音樂。
- 朝著生成更現代的音樂，像是電音三太子之類的方向前進。
- 結合各種不同樂器去生成音樂，讓生成的音樂更具多樣性和豐富性。
- 將本專題中的 One-Hot Encoding 技術替換成更被廣泛使用的 word embedding 技術。
- 專題網站上增加能讓使用者自行輸入生成種子的功能。

## 參考文獻

- [1] Nikhil Kotecha, Paul Young , “Generating Music using an LSTM Network”.  
Apr,2018. [Online]. Available:  
<https://arxiv.org/ftp/arxiv/papers/1804/1804.07300.pdf>
- [2] Ian Simon, Anna Huang, Jesse Engel, Curtis "Fjord" Hawthorne, “Generating Piano Music with Transformer”.2019. [Online]. Available: [Generating Piano Music with Transformer.ipynb - Colaboratory \(google.com\)](#)
- [3] “kernScores”. [Online]. Available: <http://kern.humdrum.org/help/tour/>
- [4] Ashish Vaswani,Noam Shazeer,Niki Parmar,Jakob Uszkoreit,Llion Jones,Aidan N. Gomez,Łukasz Kaiser,Illia Polosukhin, “Attention Is All You Need”. Dec,2017. [Online]. Available: <https://arxiv.org/pdf/1706.03762.pdf>
- [5] “music21:a toolkit for computer-aided musicology” . [Online]. Available:  
<http://web.mit.edu/music21/>
- [6] “Django documentation” . [Online]. Available:  
<https://docs.djangoproject.com/en/4.1/>
- [7] Amit Sindoliya, “Running Django on Google Colab” . Dec,2020. [Online].  
Available: <https://medium.com/@arsindoliya/running-django-on-google-colab-ea9392cdee86>
- [8] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer,

Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman,

Monica Dinculescu, Douglas Eck, “Music Transformer”. Dec,2018. [Online].

Available: <https://arxiv.org/pdf/1809.04281.pdf>