長庚大學資訊工程學系

碩士論文

Graduate Institute of Computer Science and Information Engineering

Chang Gung University

Master Thesis

運用基於生成預訓練轉換器架構的 OpenAI Whisper 多語言語音辨識引擎之台語及華語語音辨識之實作

Taiwanese/Mandarin Speech Recognition using OpenAI's Whisper Multilingual Speech Recogtion Engine Based on Generative Pretrained Transformer Architecture

研究生：謝岳哲

Graduate Student: Yueh-Che Hsieh

指導教授：呂仁園 博士

Advisor: Ren-Yuan Lyu, Ph.D.

中華民國 112 年 6 月

June, 2023

摘要

本論文將對 OpenAi 的 Whisper[1]進行研究，使用基於生成預訓練轉換器架構的多語言語音辨識引擎，進行台語之語音辨識。藉由本實驗室收集的台語連續劇，和 Common Voice 的台語語料庫，我們分別對 Whisper 的 Medium 和 Large-v2 模型進行微調，將台語語音轉換成台語(閩南語)漢字與中文。我們使用字元錯誤率(Character Error Rate，CER)作為衡量標準，結果顯示兩者皆能有效辨識台語語音。藉由本次研究的成果，我們能讓使用者能使用 Whisper 進行台語語音辨識，只需要將音檔錄製即可進行辨識，增加使用者使用台語的便利性。

此外，我們也對詞彙辨識實作了即時語音辨識系統，和利用歌聲辨識實作了卡拉 OK 系統。

關鍵字：語音辨識、台語、詞彙辨識、歌聲辨識、即時系統、轉換器架構、模型微調

abstract

This paper investigates OpenAI's Whisper [1], a multilingual speech recognition engine based on the generative pre-trained transformer architecture, for the purpose of Taiwanese speech recognition. Using a collection of Taiwanese dramas from our laboratory and the Taiwanese corpus from Common Voice, we fine-tune Whisper's Medium and Large-v2 models to convert Taiwanese speech into Minnan Chinese characters and Chinese. We evaluate the performance using the Character Error Rate (CER) as the metric, and the results show that both models are capable of accurately recognizing Taiwanese speech. With the outcomes of this study, users can utilize Whisper for Taiwanese speech recognition by simply recording audio files, enhancing the convenience of using Taiwanese.

Additionally, we have implemented a real-time speech recognition system for word recognition and a karaoke system that utilizes voice recognition for singing performance analysis.

Keywords: speech recognition, Taiwanese language, word recognition, voice recognition, real-time system, transformer architecture, model fine-tuning.

目錄

第一章 緒論

1.1 動機

在深度神經網路(Deep Neural Networks, DNN)[2]首次結合隱藏式馬可夫模型(Hidden Markov Model, HMM)[2]應用於語音辨識的研究中，利用 DNN 處理聲學模型，HMM 處理處理語言模型，取得了約 7 成的辨識效果時，即奠定下使用深度學習進行語音辨識模型訓練的基礎。遞迴神經網路(Recurrent Neural Networks, RNN)[3] 在神經網路的隱藏層中引入了時間的概念，使得它可以捕捉序列中的上下文信息，並且能夠根據之前的狀態進行適應性預測，極度適合語音辨識並且已在多個不同的實驗中取得好的結果。卷積神經網絡(Convolutional Neural Networks, CNN)[4]主要用於聲學特徵提取，從語音信號中提取的數值表示。實驗中多用於後續的全連接層進行音素或詞語的分類，或與其他模型如 RNN 結合。注意力(Attention)[5]用於處理輸入和輸出序列之間的對齊和對應關係。注意力機制可以幫助模型更好地對齊輸入聲學特徵和輸出文本之間的對應關係。Transformer[4]使用自注意力機制(Self-Attention)和位置編碼(Positional Encoding)處理序列數據，位置編碼用於區分序列中不同位置的詞元，使模型保持對詞元順序的敏感性。總體來說，只要神經網絡能夠分析文字序列的關係，並能處理聲音特徵與文字的對應，即能進行語音辨識的訓練。

在訓練出語音辨識模型後，我們並不只是一味的追求完美的辨識率，我們希望能夠活用此項技術在不同的生活場景中，例如通過語音指令控制設備的語音命令控制、語音訊號轉換為文字並記錄的

語音轉文字、與機器翻譯技術結合，可以實現即時翻譯的語音翻譯等。語音辨識也不一定只能辨識人的語音，當我們將輸入的訓練資料轉換成不同的的內容，例如不同的語言、情緒，甚至我們給予音樂，進行音樂的旋律辨識、歌詞辨識。這些技術在實作成不同的應用程式後，即能帶給人們更便利的生活。

2021 年 9 月，OpenAI 發表了新的語音辨識模型 Whisper，Whisper 是一個使用 Transformer 網路架構，可以執行多語言語音識別、語音翻譯和語言識別等任務的語音識別模型。它在 680,000 個小時、超過 90 種語言的音訊數據上進行訓練，在英文的字錯誤率(Word Error Rate, WER)[6]為 4.2%，中文亦有 14.7%，官方也提供了 Whisper 各個模型的訓練中斷點(checkpoint)供各個研究者進行模型的微調。

在本論文所稱的臺灣話，通稱**台語**或臺灣台語 (閩南語)，在台灣約有七百萬人使用**台語**作為主要溝通語言，也大約有一千四百萬人會說台語，然而作為一個非書寫(或無正式規範的)語言，目前台語雖然有教育部的推薦用字，但是在日常生活中，許多場合仍然常使用基於華語語音的繁體中文作為書寫文字，並非使用台語用字。

1.2 目的

在本篇論文中，我們主要內容將使用台語資料庫與我們從電視公司 Youtube 網站收集的台語連續劇對現階段的語音辨識技術領導系統之一的 OpenAI-Whisper 進行微調，讓 Whisper 可以對台語進行語音辨識，並嘗試在提供台語語音和中文文字作為訓練輸入的情況下，直接讓 Whisper 做語音辨識並輸出繁體中文。

此外，我們亦對語音辨識、歌聲偵測採用 OpenAI whisper 進行研究，我們也使用各種的網路模型如 CNN、LSTM (Long short-term memory) [7]、Transformer，最後我們會嘗試建立活用這些技術的系統進行展示。本篇論文的第二章會介紹使用 CNN 與語音辨識基底的即時語音辨識遊戲，第三章會介紹使用 CNN、LSTM 合併的 LRCN (Long-term recurrent Convolutional Networks) [8] 和 U-net [9] 進行音樂的歌聲偵測與分離所進行的卡拉 OK 系統，第四章正式進入使用台語語音資料進行 Whisper 微調的台語語音辨識系統支研究。在每個章節的最後我們也提供了些許改進來使上述的系統更加完善，使對這些研究有興趣的讀者提供方向進行探討。

第二章 即時語音辨識遊戲

2.1 動機

在語音辨識技術發展逐漸成熟的現在，只要有語音資料與網路模型，人們即可建構出簡單的語音辨識模型。語音資料像是 LibriSpeech [10] 、 Mozilla Common Voice [11] 、 google fleurs [12] 這些開放的大型語音資料庫皆有上千位不同語者錄製的上千條句子，只要資料庫內包含的句子皆能進行辨識。網路模型從 CNN、LSTM 到現在的 Transformer，只要訓練機器足夠強大任何語音辨識模型皆能成功建立。

然而有時候在對機器下達語音指令時，並不需要大量的詞彙進行控制，像是在觀看智慧電視時，我們只要下達開關、轉台、聲音大小等指令，在搭乘具有聲控系統的電梯時，我們只要說出欲前往的樓層即可。在使用服務時，我們只會用到少量且固定的指令，但是我們卻不希望花太多的時間等待機器進行語音辨識後，才進行動作。

在此章節中，我們會著重在即時的語音辨識模型，並以遊戲作為系統。我們在玩遊戲的過程中，若是有任何延遲的發生，包括畫面延遲遊戲、輸入延遲與載入延遲，都會降低玩家的遊戲體驗。所以選擇遊戲作為我們的成果展現是非常合適的。

2.2 目的

我們的目的為製作一個即時語音辨識的聲控遊戲系統，我們將重點著重在即時的語音辨識系統，遊戲為即時系統的輸出界面。作為即時語音辨識系統的模型，我們需要一個在辨識過程中，不需花太多時間進行前處理和辨識的模型，在這裡我們選用 2D CNN 網路模型架構進行模型訓練，因為輸入的聲音訊號只需要經過數學轉換機能過的頻譜圖，使用 2D CNN 能有效的辨識出輸入的頻譜圖對應到的語音辨識結果，而且訓練出的模型大小不大，遊戲載入模型時不會需要花大量的時間，所以我們選用此模型作為語音辨識模型的網路架構。

2.3 基本知識

2.3.1 語音辨識

語音辨識(Speech Recognition)是將語音訊號轉換成可辨識的文本或指令。通過語音辨識技術，電腦可以理解和解釋人類的語音指令，並作出相應的回應。

語音辨識技術在日常生活中被廣泛應用，例如語音助理、語音命令控制、語音轉錄等。此外，語音辨識技術還可以應用於醫療、客服、安全監控等領域。

語音辨識技術通常涉及訊號處理、模式辨識和自然語言處理等技術。語音辨識技術會通過麥克風或其他語音輸入設備接收語音訊號，然後對語音訊號進行數據預處理、特徵提取和模型訓練等步驟，最終生成文本輸出。

語音辨識技術還存在一些挑戰，例如背景噪聲、不同說話人的語音差異、語言變體和口音等因素都可能對語音辨識的準確性產生影響。因此，語音辨識技術在實際應用中還需要不斷改進和優化，以提供更準確、實用的語音辨識體驗。

語音辨識的主要流程為：語音數據收集、語音數據預處理、特徵提取、語音模型訓練、語音模型驗證和調整參數、語音文本後處理、應用等。

2.3.2 語音數據收集

在本篇論文中，我們將每個錄製的語音檔案固定為一秒、採樣率(Sampling Rate)為 16000hz、 32bit 的浮點數(float)、單聲道的 wav 檔

我們總共收集了 3 種不同的語言，分別為英文、中文和台語 (閩南語)。英文我們使用 Google speech command V2[13]的資料集。此資料集中有 35 個不同的指令，有 105829 個檔案，總時長為 105829 秒，約為 29.4 小時。中文和台語 (閩南語)使用本實驗室錄製的音檔，分別為 4650 個與 18115 個檔案，總時長為 4650 秒與 18115 秒，約為 1.3 小時與 5 小時。

2.3.3 特徵提取

特徵提取是語音訊號預處理的一個關鍵步驟，它將語音訊號轉換為一組可以用於訓練機器學習模型的數值特徵。這些特徵可以捕獲語音訊號的重要訊息，例如語音的頻譜特性、時域特性和能量特性，從而幫助語音辨識模型進行準確的辨識。本篇論文使用對數梅

爾頻譜圖(log mel spectrogram)進行特徵提取。

對數梅爾頻譜圖是聲音信號頻率內容的視覺化表示，常用於語音和音樂處理。它通過對原始音頻信號進行一系列的數學操作來獲取。

對數梅爾頻譜圖的計算流程為，首先將原始音頻信號從時間域轉換為頻率域，使用傅立葉變換等數學工具實現。這將導致將信號表示為頻率成分的集合。接下來，信號被分成小的重疊幀，每幀都分別進行分析。對於每幀，都會計算出一個頻譜圖，該頻譜圖代表時間上每個頻率成分的幅度。然後，通過將頻率成分分組到距離根據梅爾刻度調整的間隔的箱中，將該頻譜圖轉換為梅爾頻譜圖。梅爾刻度是一種更接近於人類對音高的感知的刻度。最後，將梅爾頻譜圖轉換為對數刻度，這樣可以壓縮頻譜圖的動態範圍，並增強信號中低頻率的細節。計算出的對數梅爾頻譜圖可用作機器學習的輸入，用於語音識別、音樂分類或語音識別等任務。計算對數梅爾頻譜圖的公式包含多個步驟，包括對音訊信號應用窗函數以減少頻譜洩漏，通過將短時傅里葉變換(STFT)應用於窗函數的信號來計算幅度頻譜圖，通過將每個頻率 bin 乘以一個梅爾濾波器組成的矩陣將頻譜圖映射到梅爾刻度上， 取得梅爾頻譜圖的對數以壓縮其動態範圍。

以下是公式的數學表示方式：

假設 x[n] 為原始音訊信號，w[n]為窗函數，計算窗函數的信號：

$$\boldsymbol{x_w[n] = x[n] * w[n]} \qquad \textbf{式2.1}$$

因為我們使用陣列儲存音訊資料，所以我們使用 DTFT 計算

STFT：

$$\boldsymbol{X[k]=\sum_{n=0}^{N-1}\left(x_w[n]*e^{-j\frac{2\pi}{N}kn}\right)} \qquad \textbf{式2.2}$$

其中 k 為頻率索引，N 為輸入序列的長度，j 為(sqrt(-1))。計算幅度頻譜圖：

$$\boldsymbol{M[k,t]=|X[k,t]|} \qquad \textbf{式2.3}$$

定義具有 Nf 個頻率 bin 的梅爾濾波器矩陣 H：

$$\boldsymbol{H[i,k]=h[i,f(k)]} \qquad \textbf{式2.4}$$

其中 $\boldsymbol{h[i,f]}$ 是以頻率 f 為中心的第 i 個 梅爾 濾波器的值，f(k) 是對應於 bin k 的頻率。

計算梅爾頻譜圖：

$$\boldsymbol{S[i,t]=\sum_{k}(H[i,k]*M[k,t])} \qquad \textbf{式2.5}$$

計算對數梅爾頻譜圖：

$$\boldsymbol{L[i,t]=log(S[i,t])} \qquad \textbf{式2.6}$$

得到的矩陣 L 就是對數梅爾頻譜圖。

2.3.4 模型訓練

模型訓練是語音辨識系統中的關鍵步驟，它通過使用標記的語音數據集來訓練機器學習模型，使其能夠辨識和理解語音訊號。在考慮語音辨識的即時性跟輕量化的模型後，在本篇論文我們決定使用卷積神經網絡(Convolutional Neural Networks, CNN)的模型架構進行訓練。

CNN 是一種廣泛用於圖像辨識、語音辨識、自然語言處理等領域的深度學習模型。CNN 以其有效處理高維數據和提高辨識準確度

的能力而受到廣泛關注和應用。CNN 的核心思想是利用卷積運算來提取圖像中的特徵。卷積運算是一種數學運算，通過將一個小的矩陣(稱為卷積核)滑動到圖像的每個像素上，計算卷積核與圖像局部區域的乘積，最終得到一個新的矩陣(稱為特徵圖)，其中每個元素都代表了對應的圖像區域的特徵。CNN 通過多層卷積層來進行特徵提取，然後通過全連接層進行分類或回歸。在卷積層中，每一層都可以使用多個卷積核來提取不同的特徵。通常較靠近輸入層的卷積層會提取低層次的特徵，如邊緣和紋理，而較靠近輸出層的卷積層則會提取高層次的特徵，如物體的形狀和顏色等。

CNN 主要有三種類型的層：卷積層(Convolutional layer)、池化層(Pooling layer)、全連接層(Fully-connected layer)。

卷積層：卷積層是卷積神經網絡的第一層。卷積層可以接續其他卷積層或池化層。

卷積層是 CNN 的核心構建塊，也是大部分計算發生的地方。它需要幾個組件，包括輸入數據、濾波器和特徵圖。假設輸入將是一張灰階圖像，由 2D 矩陣組成。這意味著輸入數據有三個維度，分別是高度、寬度和顏色深度。我們還有一個特徵檢測器，也稱為卷積核或濾波器，它會在圖像的感受野上移動，檢查特徵是否存在。這個過程稱為卷積。

特徵檢測器是一個二維的權重數組，表示圖像的一部分。濾波器的大小可以不同，但通常是一個 3x3 的矩陣，這也決定了感受野的大小。然後，將濾波器應用於圖像的一個區域，並計算輸入像素和濾波器之間的點積。這個點積結果被輸入到一個輸出數組中。之後，濾波器按照一個步長進行平移，重複進行這個過程，直到濾波

器掃過整個圖像。從輸入和濾波器的一系列點積的最終輸出被稱為特徵圖或卷積特徵。

假設輸入圖像為一個 2D 矩陣，記為 X，其尺寸為 TxFxD，其中 T 為圖像的橫軸，代表時間，F 為圖像的縱軸，代表頻率，D 為圖像的通道數。捲積層使用 K 個卷積核來對輸入圖像進行特徵提取，每個卷積核的尺寸為 FxFxD，其中 F 為卷積核的大小，D 為卷積核的深度，並必須與輸入圖像的通道數 D 相同。則 2D 捲積層的輸出可以使用以下的數學公式表示：

$$\boldsymbol{Y_{t,f} = \mathbf{activation}_{function}(\sum_{t,f}\sum_{\tau,\varphi} X_{t+\tau,f+\varphi} * W_{\tau,\varphi} + bias)} \qquad \textbf{式2.7}$$

由於類神經網路在訓練時，會以線性的方式進行求解，但實際的問題經常為非線性的，為了減少線性程度，在每次卷積操作後，使用激勵函數(activation_function)對特徵圖 Y 進行非線性映射，此章使用整流線性單位函數(Rectified Linear Unit, ReLU)作為激勵函數。ReLU 的數學公式為：

$$\boldsymbol{f(x) = max(0, x)} \qquad \textbf{式2.8}$$

池化層：池化層通常用於減小圖像的空間尺寸，從而減少後續層的計算負擔。它可以降低圖像的解析度，同時保留重要的特徵。與卷積層類似，池化操作在整個輸入上掃過一個濾波器，但不同的是這個濾波器不包含任何權重。相反，核心應用了一個聚合函數到感受野內的值，並填充到輸出陣列中。

假設輸入特徵圖為一個 2D 張量，記為 X，其尺寸為 HxWxC，其尺寸為 TxFxD，其中 T 為圖像的橫軸，代表時間，F 為圖像的縱軸，代表頻率，D 為圖像的通道數。

假設池化層使用 KxK 的池化核對輸入特徵圖進行聚合，則池化

層的輸出特徵圖可以使用以下的數學公式表示：

$$\boldsymbol{Y_{t,f} = \mathbf{pool_f unction}(X[t: t + k, f: f + k])} \qquad \textbf{式2.9}$$

pool_function 表示池化函數，用於對池化窗口中的值進行聚合操作，例如最大池化(Max Pooling)和平均池化(Average Pooling)等。

最大池化：當濾波器在輸入上移動時，它選擇具有最大值的像素，將其傳送到輸出陣列中。

平均池化：當濾波器在輸入上移動時，它計算感受野內的平均值，並將其傳送到輸出陣列中。

儘管在池化層中會丟失很多資訊，但它對 CNN 也有許多好處。它們有助於降低複雜性，提高效率，並限制過擬合的風險。

全連接層：全連接層是卷積神經網絡的最後一層。隨著每一層的增加，卷積神經網絡的複雜性也會增加，從而辨識圖像的更大部分。較早的層主要關注簡單的特徵，例如顏色和邊緣。隨著圖像數據通過卷積神經網絡的層層處理，它會開始辨識對象的較大元素或形狀，直到最終辨識出目標對象。

假設前一層的輸出特徵向量為 x，大小為 m×1，全連接層有 n 個神經元，則全連接層的輸出特徵向量可以表示為：

$$\boldsymbol{Y = Wx + b} \qquad \textbf{式2.10}$$

其中，W 是一個大小為 n×m 的權重矩陣，b 是一個大小為 n×1 的偏置向量。W 和 b 是全連接層的可訓練參數，需要在訓練過程中進行學習。通過調整 W 和 b 的值，全連接層可以實現任意複雜的非線性映射。

全連接層的名字恰如其分地描述了它自己。正如前面提到的，在部分連接層中，輸入圖像的像素值並不直接連接到輸出層。然

而，在全連接層中，輸出層中的每個節點都直接連接到前一層中的節點。

這一層根據之前層中提取的特徵及其不同的濾波器來執行分類任務。而卷積和池化層通常使用 ReLU 函數，全連接層通常使用 softmax 函數來適當地對輸入進行分類，生成 0 到 1 之間的機率值。

2.4 實驗程式

本節會介紹語音辨識所使用的訓練程式，遊戲前端的主程式畫面和後端的錄音與辨識程式。

2.4.1 訓練語音辨識模型

2.4.1.1 資料集載入

程式碼區段 2.1 主要目的為載入資料集資料。

這段程式碼首先載入了 NumPy 和 time 模組，並定義了 basePath 和 fn 變數，分別表示資料集的基本路徑和資料集的檔案名稱。接著，程式碼使用 np.load()函數載入了一個名為 z 的 NumPy npz 檔案，該檔案包含了訓練、驗證和測試的音訊波形資料和標籤。接下來，程式碼將 z 中的音訊波形資料和標籤分別存入了 x_train、y_train、x_val、y_val、x_test 和 y_test 變數中，用於後續的模型訓練和評估。最後，程式碼定義了 fnModel 變數，表示模型的儲存路徑和檔案名稱。並且輸出了一條顯示訓練資料集檔案載入和模型訓練路徑的訊息。

```
import numpy as np
import time

basePath= '../ryDatasets2/gscV2/'
fn= 'gscV2_data.npz'
t0= time.time()
z= np.load(basePath+fn)

x_train= z['x_trainWithSil']
y_train= z['y_trainWithSil']
x_val= z['x_val']
y_val= z['y_val']
x_test= z['x_test']
y_test= z['y_test']

fnModel= 'ryModel.hdf5'
print(".... z= np.load({}) will train into {}".format(fn, fnModel))
```

程式碼區段 2.1：資料集載入

2.4.1.2 特徵提取

程式碼區段 2.2 主要目的為對音訊片段進行特徵提取。

函數的輸入參數如下：

參數名稱	參數數值
sample_rate	16000
frame_length	1024
frame_step	128
num_mel_bins	128
lower_edge_hertz	20
upper_edge_hertz	8000
mfcc_dim	13

表 2.1：特徵提取函數

x: 輸入的音訊波形，型別為 tf.Tensor。

參數名稱解釋如下：

sample_rate: 音訊的採樣率；frame_length: 每個窗口的長度；frame_step: 窗口之間的間隔；num_mel_bins: 梅爾頻譜的個數；lower_edge_hertz: 梅爾頻譜的下邊緣頻率；upper_edge_hertz: 梅爾頻譜的上邊緣頻率；mfcc_dim: 要保留的 MFCC 維度。

函數的運作流程如下：

進行短時傅立葉變換(STFT)，得到音訊的頻譜，計算頻譜的絕對值和對數值。將頻譜轉換為梅爾頻譜，使用 linear_to_mel_weight_matrix 函數進行矩陣乘法，計算梅爾頻譜的對數值，從對數梅爾頻譜(log mel spectrograms)中計算 MFCCs，並保留前 13 個維度，將提取到的特徵以字典形式返回，包括 MFCCs、對數梅爾頻譜、對數頻譜和頻譜，使用對數梅爾頻譜作為音訊片段的特徵提取結果。

```
import tensorflow as tf

def get_Feature(x,
          sample_rate= 16000,
          frame_length= 1024,
          frame_step= 128, # frame_length//2
          num_mel_bins= 128,
          lower_edge_hertz= 20, # 0
          upper_edge_hertz= 16000/2, # sample_rate/2
          mfcc_dim= 13
            ):
    #短時距傅立葉變換
    stfts= tf.signal.stft(x,
                          frame_length,
                          frame_step,
                          #fft_length= 1024
                          pad_end=True
                           )

    spectrograms= tf.abs(stfts)
    log_spectrograms= tf.math.log(spectrograms + 1e-10)

    # Warp the linear scale spectrograms into the mel-scale.
    num_spectrogram_bins= stfts.shape[-1] #.value

    linear_to_mel_weight_matrix= tf.signal.linear_to_mel_weight_matrix(
        num_mel_bins,
        num_spectrogram_bins,
        sample_rate,
        lower_edge_hertz,
        upper_edge_hertz)

    mel_spectrograms= tf.tensordot(
          spectrograms,
          linear_to_mel_weight_matrix, 1)

    mel_spectrograms.set_shape(
       spectrograms.shape[:-1].concatenate(
          linear_to_mel_weight_matrix.shape[-1:]))

    # Compute a stabilized log to get log-magnitude mel-scale
spectrograms.
    log_mel_spectrograms= tf.math.log(mel_spectrograms + 1e-10)

    # Compute MFCCs from log_mel_spectrograms and take the first 13.
    mfccs= tf.signal.mfccs_from_log_mel_spectrograms(
    log_mel_spectrograms)[..., :mfcc_dim]

feature= {'mfcc': mfccs,
              'log_mel_spectrogram':log_mel_spectrograms,
              'log_spectrogram': log_spectrograms,
              'spectrogram': spectrograms}

return feature
```

程式碼區段 2.2：特徵提取

2.4.1.3 提取資料集特徵

程式碼區段 2.3 主要目的為對音訊使用 batch_size 進行分割後使用對數 梅爾頻譜圖提取特徵。

函數接受一個名為 all_x 的輸入參數，表示所有的音訊波形資料。預設的批次大小為 1000，但可以透過 batch_size 參數調整。函數先將 all_x 的資料類型轉換為 np.float32。接著，函數使用 get_Feature 來提取特徵，並將特徵資料存入 X 變數中。這裡的特徵是基於對數梅爾頻譜圖進行提取的，X 的資料類型也轉換為 np.float32。函數使用迴圈來處理輸入資料，每次處理一個批次的資料。在迴圈中，函數會檢查目前處理的資料索引是否超過了 all_x 的總資料量，如果是則取剩餘的資料，否則取一個批次的資料。
將每個批次的特徵資料 X 轉換為 NumPy 陣列後，將其存入名為 XL 的列表中。迴圈處理完所有的資料後，將 XL 列表中的特徵資料堆疊成一個 NumPy 陣列，並將其存入 XL 變數中。函數輸出一條顯示 XL 陣列形狀和執行時間的訊息。函數最終輸出 XL 陣列作為提取的特徵資料。

```
def get_all_fearure(all_x, batch_size= 1000):

    x= all_x.astype(np.float32)
    i=0
    XL=[]
    while i < x.shape[0]:
        if i+batch_size<=x.shape[0]:
        xx= x[i:i+batch_size]
    else:
        xx= x[i:]

    XX= get_Feature(xx)
    X= XX['log_mel_spectrogram']
    X= X.numpy().astype(np.float32)
    i += batch_size
    XL += [X]

    XL= np.concatenate(XL)
    print('XL.shape={}'.format(XL.shape))

    return XL
```

程式碼區段 2.3：提取資料集特徵

2.4.1.4 計算特徵提取執行時間

程式碼區段 2.4 主要目的為對資料集進行特徵提取。使用 time.time()函數記錄了當前時間，並存入 t0 變數中，接著分別呼叫 get_all_fearure()函數來提取測試集(x_test)、驗證集(x_val)和訓練集(x_train)的音訊特徵資料，並將訓練集提取結果分別存入 X_test、X_val 和 X_train 變數中，最後，計算從呼叫 get_all_fearure()函數到目前時間的時間差，並將結果存入 dt 變數中，並印出執行時間。

```
t0= time.time()

X_test= get_all_fearure(x_test)
X_val= get_all_fearure(x_val)
X_train= get_all_fearure(x_train)

dt= time.time()- t0
print('... get_all_fearure() ... dt(sec)= {:.3f}'.format(dt))

nTime, nFreq= X_train[0].shape
```

程式碼區段 2.4：計算特徵提取執行時間

2.4.1.5 正規化

程式碼區段 2.5 主要目的為對輸入的資料進行正規化。

x 為輸入的資料，可以是一個 numpy 陣列或數值。axis 為正規化的軸向，預設為 None，表示對整個資料進行正規化。如果指定了軸向，則會對指定的軸向進行正規化。

函數的操作如下：

如果 axis 為 None，則計算 x 的平均值和標準差，並將 x 減去平均值後再除以標準差，得到正規化後的結果。如果 axis 不為 None，則計算 x 在指定軸向上的平均值和標準差，並將 x 減去平均值後再除以標準差，得到正規化後的結果。最終，函數返回正規化後的結果 x。

```
1.  def normalize(x, axis= None):
2.      if axis== None:
3.          x= (x-x.mean())/x.std()
4.      else:
5.          x= (x-x.mean(axis= axis))/x.std(axis= axis)
6.      return x
```

程式碼區段 2.5：正規化

2.4.1.6 資料集重置向量與正規化

程式碼區段 2.6 主要目的為對資料集重置向量與正規化。

使用 reshape()函數將數據重置向量為 4D 向量，其中第一個維度為-1，表示自動計算該維度的大小，而後面的維度則是 nTime、nFreq 和 1，分別表示時間步數、頻率數和通道數。這是因為神經網絡的輸入通常是 4D 向量，其中第一個維度表示樣本數、第二個維度表示時間步數、第三個維度表示頻率數，最後一個維度表示通道數(例如，灰度圖像的通道數為 1)。使用 astype('float32')將數據類型

轉換為 32 位浮點型，以符合神經網絡的輸入要求。使用程式碼區段五的正規化函數，對數據進行正規化處理。

```
X_train= X_train.reshape(-1, nTime, nFreq, 1).astype('float32')
X_val= X_val.reshape(-1, nTime, nFreq, 1).astype('float32')
X_test= X_test.reshape( -1, nTime, nFreq, 1).astype('float32')

X_train= normalize(X_train)
X_val= normalize(X_val)
X_test= normalize(X_test)
```

程式碼區段 2.6：資料集重置向量與正規化

2.4.1.7 CNN 模型

程式碼區段 2.7 主要目的為使用 Keras 建立的卷積神經網絡(CNN)模型。

引入了需要使用的 Keras 模組，包括 keras、layers 和 Model 等。定義了輸入層，使用 Input 函數創建一個形狀為(nTime, nFreq, 1)的輸入向量 x，其中 nTime 和 nFreq 分別表示時間步數和頻率數，1 表示通道數。通過對 h 的多次卷積、批量標準化、最大池化和 Dropout 操作，來構建卷積層。其中使用了 8 個濾波器、32x32 和 16x16 的卷積核，並使用 ReLU 激活函數。每個卷積層後面都接著批量標準化、最大池化和 Dropout 操作，以增加模型的泛化能力和減少過擬合風險。通過 Reshape 層將數據重置向量為(None, -1, 32)，其中-1 表示自動計算該維度的大小，32 表示維度大小。接著使用 LSTM 層，設置 200 個單元數目，來捕捉時間序列特徵。將 LSTM 層的輸出通過 Flatten 層攤平為一維向量。通過 Dense 層添加全連接層，分別包括 256、128 個神經元，並使用 ReLU 激活函數。其中也包含 Dropout 操作，以減少過擬合風險。最後一層使用 nCategs 個神經

元，並使用 softmax 激活函數，用於多類別分類，其中 nCategs 表示類別的數目。使用 Model 函數定義了模型的輸入和輸出，並將其存儲在 m 變數中。最後使用 m.summary()函數打印出模型的摘要，顯示了模型的結構和參數數量。

```
from tensorflow import keras
from tensorflow.keras import layers, Model
from tensorflow.keras.layers import Input, Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import AveragePooling1D

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

nCategs= len(set(y_train))

x= Input(shape= (nTime, nFreq, 1))
h= x

h= Conv2D(filters=8, kernel_size=(32,32), activation='relu', padding='same')(h)
h= tf.keras.layers.BatchNormalization()(h)
h= MaxPooling2D(pool_size=(4,4), padding='same')(h)
h= Dropout(0.3)(h)

h= Conv2D(16, (16,16), activation='relu', padding='same')(h)
h= tf.keras.layers.BatchNormalization()(h)
h= MaxPooling2D((4,4), padding='same')(h)
h= Dropout(0.3)(h)

h= Conv2D(32, (8,8), activation='relu', padding='same')(h)
h= tf.keras.layers.BatchNormalization()(h)
h= MaxPooling2D((4,4), padding='same')(h)
h= Dropout(0.3)(h)

h= tf.keras.layers.Reshape((-1, 32))(h)
h = tf.keras.layers.LSTM(200)(h)

h= Flatten()(h)
h= Dense(256, activation='relu')(h)
h= Dense(128, activation='relu')(h)
h= Dropout(0.2)(h)
h= Dense(nCategs, activation='softmax')(h)

y= h
m= Model(inputs= x,
                outputs= y)

m.summary()
```

程式碼區段 2.7：CNN 模型

2.4.1.8 訓練模型

程式碼區段 2.8 主要目的為訓練 CNN 模型。

CNN 模型使用 'sparse_categorical_crossentropy' 損失函數，並將準確度設置為評估指標。使用 EarlyStopping 回調函數來監控驗證損失，如果在 200 個訓練輪內沒有改善，則停止訓練。使用 ModelCheckpoint 回調函數來保存基於驗證準確度的最佳模型。

模型使用 fit() 函數進行訓練，使用以下參數：

X_train：訓練數據

y_train：訓練標籤

batch_size：訓練時使用的批次大小(1000)

epochs：訓練輪數(1000)

callbacks：訓練時使用的回調函數列表(包括 EarlyStopping 和 ModelCheckpoint)

validation_data：驗證數據的元組(X_val, y_val)

在訓練期間會印出損失、準確度和驗證損失等訓練進度信息。訓練完成後會印出總共的訓練時間。

```
m.compile(
                loss= 'sparse_categorical_crossentropy',
                metrics= ['accuracy'])

es= EarlyStopping(
                monitor= 'val_loss',
                min_delta= 1e-10,
                patience= 200,
                mode= 'min',
                verbose= 1)

mc= ModelCheckpoint(fnModel,
                monitor= 'val_accuracy',
                verbose= 1,
                save_best_only= True,
                mode= 'max')

t0= time.time()

h= m.fit(X_train, y_train,
                batch_size=1000,
                epochs= 1000,
                callbacks=[es, mc],
                validation_data= (X_val, y_val)
                )

dt= time.time()- t0
```

程式碼區段 2.8：訓練模型

2.4.1.9 繪製訓練結果

程式碼區段 2.9 主要目的為繪製訓練結果。

h.history 是訓練過程中儲存的歷史記錄，包含了訓練期間的損失和準確度等指標的變化。v0 是訓練準確度的歷史記錄，v1 是驗證準確度的歷史記錄。接著使用 matplotlib 繪製兩者的曲線，並使用 label 參數指定標籤名稱，以便顯示在圖例中。最後使用 legend() 函數顯示圖例，grid('on') 函數顯示網格，show() 函數顯示繪製的圖形。

```
import numpy as np
from matplotlib import pyplot as pl
v0= h.history['accuracy']
v1= h.history['val_accuracy']
pl.plot(v0, label='accuracy')
pl.plot(v1, label='val_accuracy')
pl.legend()
pl.grid('on')
pl.show()
```

程式碼區段 2.9：繪製訓練結果

2.4.2 即時聲控遊戲前端

2.4.2.1 初始化與參數設定

程式碼區段 2.10 主要目的為初始化 Pygame 遊戲引擎，載入圖片，並定義參數。

此段共設定了七個部分：初始化 pygame,遊戲視窗圖示和視窗標題，遊戲視窗大小，遊戲使用字體，載入圖片，遊戲物件--蘋果的設定。遊戲使用字體設定為標楷體避免在輸出辨識中文結果時出現字體無法顯示的替代方格。蘋果物件的記錄是分別使用不同的陣列紀錄蘋果的 x 軸、y 軸、x 軸的移動方向、y 軸的移動方向。不將同一個蘋果的座標位置記錄在同一個陣列的原因是因為需要使用座標位置與移動方向的時間點不同，若將所有資訊紀錄在同一個陣列會使程式需要常常使用兩層式陣列，造成程式碼雜亂。分開記錄的話只需要使用陣列的位置當作同一顆蘋果的編號即可進行計算，不需要讀取到其他的值。

```
# initialize pygame
pygame.init()

# caption and icon
pygame.display.set_caption("Catch")
icon = pygame.image.load('ele/ufo.png')
pygame.display.set_icon(icon)

# screen
win = pygame.display.set_mode((800,600))
pygame.display.set_caption("Apple CATCHER")

# font
font = pygame.font.SysFont("dfkaisb",32)
over_font = pygame.font.Font('freesansbold.ttf', 64)

# image
background = pygame.image.load('ele/background.png')
playerImg = pygame.image.load('ele/player_r.png')
logoImg = pygame.image.load('menu/logo.png')
startImg = pygame.image.load('menu/Start.png')
modeImg = pygame.image.load('menu/Mode.png')
engImg = pygame.image.load('menu/eng.png')
zhImg = pygame.image.load('menu/zh.png')
minnanImg = pygame.image.load('menu/mi.png')
bulletImg = pygame.image.load('ele/Shuriken.png')
appleImg = pygame.image.load('ele/apple.png')

# Apples
appleX = []
appleY = []
appleX_change = []
appleY_change = []
num_of_apples = 3

for i in range(num_of_apples):
    appleX.append(random.randint(0, 736))
    appleY.append(random.randint(50, 150))
    appleX_change.append(2)
    appleY_change.append(20)
```

程式碼區段 2.10：初始化與參數設定

2.4.2.2 碰撞判定

程式碼區段 2.11 主要目的為定義子彈與蘋果的碰撞判定。

使用畢氏定理計算距離，若子彈和蘋果距離小於 27 像素，則視為碰撞。

```
def isCollision(appleX, appleY, bulletX, bulletY):
    distance = ((appleX - bulletX) ** 2 + ((appleY - bulletY) ** 2)) ** 0.5
    if distance < 27:
        return True
    else:
        return False
```

程式碼區段 2.11：碰撞判定

2.4.2.3 儲存語音辨識結果

程式碼區段 2.12 主要目的為儲存語音辨識結果。

在遊戲內成功辨識的語音指令會儲存於 history 陣列中，陣列上限為 10 個單字，當超過 10 個單字時會去除掉第一個存入的單子並將最新的單字加在陣列的最末端。迴圈讀取 history_str 中的每個單詞，並將其連接成一個字串，每個單詞之間用空格分隔，儲存於 history_str 供後續輸出至畫面。

```
def save_history(word):
    global history_str
    if len(history) == 10:
        history.pop(0)
        history.append(word)
        history_str = " "
        for i in range(len(history)):
            history_str += history[i]+" "
    else:
        history.append(word)
        history_str += word +" "
```

程式碼區段 2.12：儲存語音辨識結果

2.4.2.4 主選單畫面

程式碼區段 2.13 主要目的為繪製遊戲的主選單函式 main_menu()。

此函式主要分為四個部分：繪製圖片，判斷事件，點擊按鈕與更新

畫面。繪製圖片為繪製背景，繪製對數 o，繪製並設定開始按鈕。判斷事件為函式會持續偵測使用者的操作，包括滑鼠點擊和按鍵盤按鍵。 pygame.mouse.get_pos() 為取得鼠標在視窗內的座標。pygame.QUIT 為視窗右上角的 X 按鍵，點擊時會離開遊戲。pygame.KEYDOWN 為使用者按鍵盤按鍵。pygame.K_ESCAPE 為鍵盤 Esc 按鍵，按 Esc 會離開遊戲 pygame.MOUSEBUTTONDOWN 為使用者滑鼠點擊。 event.button 等於 1 代表點擊左鍵。點擊按鈕為當使用者點擊了開始按鈕，會呼叫 mode_menu() 進入遊戲模式選單。更新畫面為使用 pygame.display.update() 將繪製的畫面顯示於遊戲視窗。

```
def main_menu():
    while True:
        win.fill((0, 0, 0))
        win.blit(background, (0, 0))
        win.blit(logoImg, (275, 60))
        win.blit(startImg, (300,300))
        button_start = pygame.Rect(300,300,200,131)

        mx, my = pygame.mouse.get_pos()

        click=False
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    pygame.quit()
                    sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                    click = True

        if button_start.collidepoint((mx,my)):
            if click:
                mode_menu()

        pygame.display.update()
```

程式碼區段 2.13：主選單畫面

2.4.2.5 語音辨識模型選單

程式碼區段 2.14 主要目的為繪製語音辨識模型的選單。

此函式與主選單畫面的主要差別為按鈕為選擇英文、中文與台語 (閩南語)。遊戲會根據選擇的語言載入不同的語音辨識模型。

```
def mode_menu():
    while True:
        win.fill((0, 0, 0))
        win.blit(background, (0, 0))
        win.blit(engImg, (150, 250))
        win.blit(zhImg, (350,250))
        win.blit(minnanImg, (550, 250))
        win.blit(modeImg, (350, 100))

        button_eng = pygame.Rect(150,250,150,150)
        button_zh = pygame.Rect(350,250,150,150)
        button_minnan = pygame.Rect(550,250,150,150)

        mx, my = pygame.mouse.get_pos()

        click = False
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    pygame.quit()
                    sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                    click = True

            if button_eng.collidepoint((mx,my)):
                if click:
                    speech_game_loop(lan='eng')
                if button_zh.collidepoint((mx,my)):
                    if click:
                        speech_game_loop(lan='zh')
                if button_minnan.collidepoint((mx,my)):
                    if click:
                        speech_game_loop(lan='minnan')

        pygame.display.update()
```

程式碼區段 2.14：語音辨識模型選單

2.4.2.6 遊戲參數設定

程式碼區段 2.15 主要目的為選擇載入的語音辨識模型與定義遊戲使用的變數。

根據模式選單選擇的語言載入相對應的語音辨識模型。設定角色與子彈的初始位置和位移方向歸零。設定子彈的狀態：當子彈在 'ready' 狀態時才能發射子彈，'fire' 狀態為子彈正在飛行。history 為保存語音辨識的歷史紀錄，初始設定為擁有一項'_silence_'的陣列，表示開始為靜音，也避免後續計算錯誤。history_str 為顯示存於 history 內的內容用的字串。score_value 為遊戲分數。Recog_run 為控制是否進行語音辨識迴圈的值。Clock 和 fps 為控制畫面刷新的速率，也用於控制遊戲速度。recProbToConfirm 為語音辨識的機率閾值，用於確認語音辨識結果。sStream 為開啟 sounddevice 的聲音流，用於接收使用者傳入麥克風的聲音。Recog_thread 為進行語音辨識的線程，將 Recog() 置於此線程進行，並將 recogQ 此 Queue 與選擇的語音辨識模型傳入函式，Recog() 會將辨識的結果放入 recogQ。

```
def speech_game_loop(lan = 'eng'):
    # choose the model's language
    if lan == 'eng':
        import ryRecog06_eng as rec
    elif lan == 'zh':
        Import ryRecog06_TW as rec
    elif lan == 'minnan':
        import ryRecog06_minnan as rec

    #inital setting
    playerX = 370
    playerY = 480
    playerX_change = 0
    playerY_change = 0
    bulletX = 0
    bulletY = 480
    bulletX_change = 0
    bulletY_change = 10
    bullet_state = "ready"
    global history
    history = ['_silence_']
    global history_str
    history_str = " "
    score_value = 0
    global Recog_run
    Recog_run = True
    Clock = pgTime.Clock()
    fps = 30 # loop/sec
    recProbToConfirm = 0.8

    sStream = sdStream()
    sStream.start()

    Recog_thread = Thread(target= Recog,
                          args=(recogQ,rec),
                          daemon= True)
    Recog_thread.start()
```

程式碼區段 2.15：遊戲參數設定

2.4.2.7 語音辨識指令控制

程式碼區段 2.16 主要目的為遊戲的主要迴圈語音辨識指令控制部分。

在主要迴圈開始時，設定畫面的影格速率並重新繪製背景畫面，避免將上一幀的畫面重複顯示於這一幀，造成畫面上產生多重的蘋果、子彈和角色。將 Recog 辨識的結果使用 recogQ_Get() 從

recogQ 存入 recogQ_list 。使用 for 迴圈檢視 recogQ_list 中的語音辨識結果，並檢查確認機率是否超過設定的 recProbToConfirm 閾值。如果語音辨識結果確認為有效指令，並確認與最後的正確指令不相同時，則根據不同的語音指令執行相應的角色動作：更動角色 X 軸和 Y 軸位移方向 ，發射子彈或停止移動。並將成功辨識的成果使用 save_history() 儲存後待後續輸出如果關閉遊戲視窗，則退出主要迴圈並離開遊戲畫面。

```
    running = True
    while running:
    pgClock.tick(fps)
    win.fill((0, 0, 0))
    win.blit(background, (0, 0))

    recogQ_list = recogQ_Get(recogQ)
    recResult = ' '
    prob=0.0
    for recResult, prob in recogQ_list:
        if prob > recProbToConfirm:
            if history[-1] == recResult:
                break
        if recResult in ['left','左']:
            playerX_change = -1
            save_history(recResult)
        elif recResult in ['right','右']:
            playerX_change = +1
            save_history(recResult)
        elif recResult in ['forward','up','前進','上']:
            playerY_change = -1
            save_history(recResult)
        elif recResult in ['backward','down','後退','下']:
            playerY_change = +1
            save_history(recResult)
        elif recResult in ['yes', 'on', 'go','可以','開','去']:
            save_history(recResult)
            if bullet_state == "ready":
                bullet_state = "fire"
                bulletSound = mixer.Sound('ele/laser.wav')
                bulletSound.play()
                # Get the current x cordinate of the spaceship
                bulletX = playerX
                bulletY = playerY
            elif recResult in ['no','off','stop','不可','關']:
                save_history(recResult)
                playerX_change = 0
                playerY_change = 0

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
```

程式碼區段 2.16：語音辨識指令控制

2.4.2.8 物件移動

程式碼區段 2.17 主要目的為物件移動部分與子彈碰撞判定。

角色移動為根據語音辨識的結果進行移動。子彈移動的方式為在接收到發射指令時每幀向上 10 像素，若是子彈超過視窗範圍時重置子彈狀態。蘋果的移動方式為個別判斷蘋果是否到達遊戲結束位置，進行蘋果的移動，判斷蘋果是否與子彈碰撞，若碰撞會在非遊戲結束位置隨機產生新的一顆蘋果。

```
    # player Movement
    playerX += playerX_change
    if playerX <= 0:
        playerX = 0
    elif playerX >= 736:
        playerX = 736

    playerY += playerY_change
    if playerY <= 0:
        playerY = 0
    elif playerY >= 480:
        playerY = 480

    win.blit(playerImg, (playerX, playerY))

    # Bullet Movement
    if bulletY <= 0:
        bulletY = 480
        bullet_state = "ready"

    if bullet_state == "fire":
        bulletY -= 10
        win.blit(bulletImg, (bulletX , bulletY))

    # Apple Movement
    for i in range(num_of_apples):
        # Game Over
        if appleY[i] > 440:
            for j in range(num_of_apples):
                appleY[j] = 2000

            over_text = over_font.render("GAME OVER", True, (255, 255, 255))
            win.blit(over_text, (200, 250))
            break

        appleX[i] += appleX_change[i]
        if appleX[i] <= 0:
            appleX_change[i] = 4
            appleY[i] += appleY_change[i]
        elif appleX[i] >= 736:
            appleX_change[i] = -4
            appleY[i] += appleY_change[i]

        # Collision
        collision = isCollision(appleX[i], appleY[i], bulletX, bulletY)
        if collision:
            explosionSound = mixer.Sound("coin.wav")
            explosionSound.play()
            score_value += 1
            appleX[i] = random.randint(0, 736)
            appleY[i] = random.randint(50, 150)

        win.blit(appleImg, (appleX[i], appleY[i]))
```

程式碼區段 2.17：物件移動

2.4.2.9 結果繪製與結束迴圈

程式碼區段 2.18 主要目的為繪製歷史辨識結果、即時辨識結果、分數，和結束迴圈後的處理。

history_label 為將 save_history() 儲存的成功辨識的指令繪製於畫面的下方。分數繪製於畫面的左上方。recResult 為當前辨識到的結果，並不一定為成功的指令。在結束遊戲主迴圈時會關閉 sounddevice 的聲音流和語音辨識的線程，使切換語音辨識模型時可重新開啟使用。

```
            history_label = font.render(history_str ,True , (255,255,0))
            win.blit(history_label, (10, 550))

            score = font.render("Score : " + str(score_value), True, (255, 255, 255))
            win.blit(score, (10, 10))

            recR = font.render(recResult, True, (0, 255, 255))
            win.blit(recR, (playerX-50, playerY+50))

            pygame.display.update()
            sStream.stop()
            sStream.close()

        Recog_run = False
        Recog_thread.join()
```

程式碼區段 2.18：結果繪製與結束迴圈

2.4.3 語音辨識後端

2.4.3.1 麥克風輸入參數設定

程式碼區段 2.19 主要目的為設定語音辨識麥克風輸入的參數。

聲道(Channel)設定為單聲道，採樣率(sample rate)設定為 16khz 供語音辨識模型吻合輸入。每幀採樣數(SamplePerFrame)設定為 1000 代表每幀有 1000 個採樣點，供儲存麥克風輸入的資料用。BufferSize 設定為 160 幀代表可以儲存 10 秒鐘的長度，因為採樣率為每秒採樣數，每秒採樣數除以每幀採樣數為每秒幀數，所以 16000 / 1000 = 16，16 * 10 = 160 得出。Buffer 以浮點數儲存麥克風輸入的資料陣列，建立時隨機產生 0 到 (1e-10) 的數字代表微小雜訊，Buffer 以環狀陣列使用。幀索引數(Frame_index)為用於記錄此幀在 Buffer 內的位置，因為 Buffer 為環狀陣列，所以 Buffer 的陣列位置並不代表幀的位置，需要此參數儲存位置。recogQ 為儲存語音辨識結果的 queue，長度為 100。

```
1. CHANNEL = 1
2. sampleRate = 16000 # SamplePerSec
3. SamplePerFrame = 1000
4. BufferSize = 160 # 16000 / 1000 * 10 s
5. Buffer = (1e-10) * np.random.random((BufferSize, SamplePerFrame,
   CHANNEL))
6. Frame_index = 0
7. recogQ = Queue(100)
```

程式碼區段 2.19：麥克風輸入參數設定

2.4.3.2 建立錄音音訊流

程式碼區段 2.20 主要目的為使用 sounddevice 模組初始化音訊錄製和播放串流。

callback 為串流重複呼叫的函數，設定為 input_to_buffer() 處理錄製的音訊幀。channels 為音訊串流中的聲道數，設定為單聲道。samplerate 為音訊串流的採樣率，設定為 16000。blocksize 為每個音訊幀中的音訊樣本數，設定為 1000。

```
def sdStream():
    aStream= sd.Stream(callback = input_to_buffer,
                       channels = CHANNEL,
                       samplerate = sampleRate,
                       blocksize = SamplePerFrame
                      )
    return aStream
```

程式碼區段 2.20：建立錄音音訊流

2.4.3.3 處理輸入語音

程式碼區段 2.21 主要目的為將麥克風輸入的資料存放於 Buffer 中。

sounddevice 的播放串流會將麥克風接收到的資料透過 input_data 傳入此函數，因為 Buffer 是環狀陣列，Buffer 會根據 Frame_index 除以 Buffer 大小的餘數放入 Buffer 陣列中，再使 Frame_index 加一。

```
def input_to_buffer(input_data, output_data, frames, time, status):
    global Frame_index, Buffer

    if status:
        print(status)

    Buffer[Frame_index % BufferSize] = input_data
    Frame_index += 1
```

程式碼區段 2.21：處理輸入語音

2.4.3.4 取得每秒鐘語音

程式碼區段 2.22 主要目的為取得 Buffer 中從目前時間點開始的一秒音訊資料。

此函數會從 Buffer 中先找出當下時間點的資料位置，將此點當作陣列的頭，將 Buffer 頭尾相接成這一秒結束的陣列。最後回傳這一秒的資料。

```
def Get1secSpeech():
    global Buffer, BufferSize, Frame_index

    x = Buffer
    t1 = (Frame_index % BufferSize)
    x = np.vstack((x[t1:], x[0:t1]))
    x = x.flatten()
    x = x.astype(np.float32)
    x = x[-16000:]
    print('.', end='', flush=True)

    return x
```

程式碼區段 2.22：取得每秒鐘語音

2.4.3.5 語音辨識

程式碼區段 2.23 主要目的為對語音資料進行辨識並回傳結果與機率。

將輸入的語音資料轉換成一維陣列後，計算音訊的對數梅爾頻譜圖，對頻譜圖進行正規化，轉換模型的輸入格式後進行預測。預測後會回傳機率最高的結果，也可以選擇增加回傳最高的結果的機率

```
def recWav(x, probOut= False):

    x= x.flatten()
    X= get_Feature(x)['log_mel_spectrogram']
    X= normalize(X) # normalized for only one utterence x
    X= tf.reshape(X, (1, X.shape[0], X.shape[1], 1))

    prob= model.predict(X)[0]
    index= prob.argsort()[-1::-1]
    maxProb= prob[index]

    y= labels[index]

    if probOut==True:
        y= np.vstack((y, maxProb))
    return y
```

程式碼區段 2.23：語音辨識

2.4.3.6 語音辨識主迴圈

程式碼區段 2.24 主要目的為語音辨識的主要迴圈。

在取得每秒的語音資料後，進行語音辨識並將結果放入 queue 中。使用者能從 queue 當中取出當前秒數的語音辨識結果。

```
def Recog(q,rec):
    global Recog_run
    print('Recog start ....')
    while Recog_run:
        x= Get1secSpeech()
        yp= rec.recWav(x, probOut= True)
        y= yp[0,0]
        prob= yp[1,0].astype('float32')
        q.put((y,prob))

    print('Recog ended ....')
```

程式碼區段 2.24：語音辨識主迴圈

2.4.3.7 取得辨識結果

程式碼區段 2.25 主要目的為取得儲存於 queue 的辨識結果。

使用迴圈取出 queue 當中的資料，並以陣列方式回傳。

```
def recogQ_Get(q):
    return [q.get() for i in range(q.qsize())]
```

程式碼區段 2.25：取得辨識結果

2.5 實驗結果

2.5.1 模型訓練結果

2.5.1.1 語音數據收集

在本篇論文中，我們將每個錄製的語音檔案固定為一秒、採樣率(Sampling Rate)為 16000hz、 32bit 的浮點數(float)、單聲道的 wav 檔

我們總共收集了 3 種不同的語言，分別為英文、中文和台語 (閩南語)。英文我們使用 Google speech command V2 的資料集。此資料集中有 35 個不同的指令，有 105829 個檔案，總時長為 105829 秒，約為 29.4 小時。中文和台語 (閩南語)使用本實驗室錄製的音檔 [14]，分別為 4650 個與 18115 個檔案，總時長為 4650 秒與 18115 秒，約為 1.3 小時與 5 小時。

語言	小時	人數
英文	29.4	2618
中文	1.3	3
台語 (閩南語)	5	15

表 2.2：不同語言的音檔長度與錄音人數

2.5.1.2 訓練成果

在 tensorflow 2.5.0 RTX 4090 的環境下，訓練參數為表 2.3：

參數名稱	參數值
compile	
loss	sparse_categorical_crossentropy
metrics	accuracy
EarlyStopping	
monitor	val_loss
min_delta	1e-10
patience	200
mode	min
verbose	1
ModelCheckpoint	
monitor	val_accuracy
verbose	1
save_best_only	True
mode	max
fit	
epochs	1000

表 2.3：語音辨識訓練參數

我們參考了 tensorflow 官方[16]使用的 CNN 網路模型，在修改後進行訓練，訓練的模型參數如表 2.4：

Model: "CNN"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 125, 128, 1)]	0
conv2d (Conv2D)	(None, 125, 128, 8)	8200
batch_normalization (BatchNo	(None, 125, 128, 8)	32
max_pooling2d (MaxPooling2D)	(None, 32, 32, 8)	0
dropout (Dropout)	(None, 32, 32, 8)	0
conv2d_1 (Conv2D)	(None, 32, 32, 16)	32784
batch_normalization_1 (Batch	(None, 32, 32, 16)	64
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 16)	0
dropout_1 (Dropout)	(None, 8, 8, 16)	0
conv2d_2 (Conv2D)	(None, 8, 8, 32)	32800
batch_normalization_2 (Batch	(None, 8, 8, 32)	128
max_pooling2d_2 (MaxPooling2	(None, 2, 2, 32)	0
dropout_2 (Dropout)	(None, 2, 2, 32)	0
reshape (Reshape)	(None, 4, 32)	0
lstm (LSTM)	(None, 200)	186400
flatten (Flatten)	(None, 200)	0
dense (Dense)	(None, 256)	51456
dense_1 (Dense)	(None, 128)	32896
dropout_3 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 32)	4128

Total params: 348,888
Trainable params: 348,776
Non-trainable params: 112

表 2.4：CNN 模型架構與參數

訓練結果如下：

訓練語言	辨識率	loss	訓練集人數
英文	0.8998	0.4086	2618
中文	0.9506	0.3109	2
台語 (閩南語)	0.8969	0.5240	15

表 2.5：語音辨識結果

三個語言的辨識率皆達到 90%，顯示大多情況下三個語言模型皆能成功辨識使用者的輸入。在中文的模型訓練，因為資料集所包含的人數較少，所以我們在分訓練集、驗證集和測試集時所使用的方式為訓練集二人，驗證集和測試集是另一人的方式，避免模型因為過度擬和而失去準確性。

2.5.2 介面

2.5.2.1 主畫面

圖 2.1 為即時聲控遊戲的主畫面，此畫面為開啟遊戲後的首個畫面。在按下畫面中的 start 後會進入遊戲模式選擇畫面。

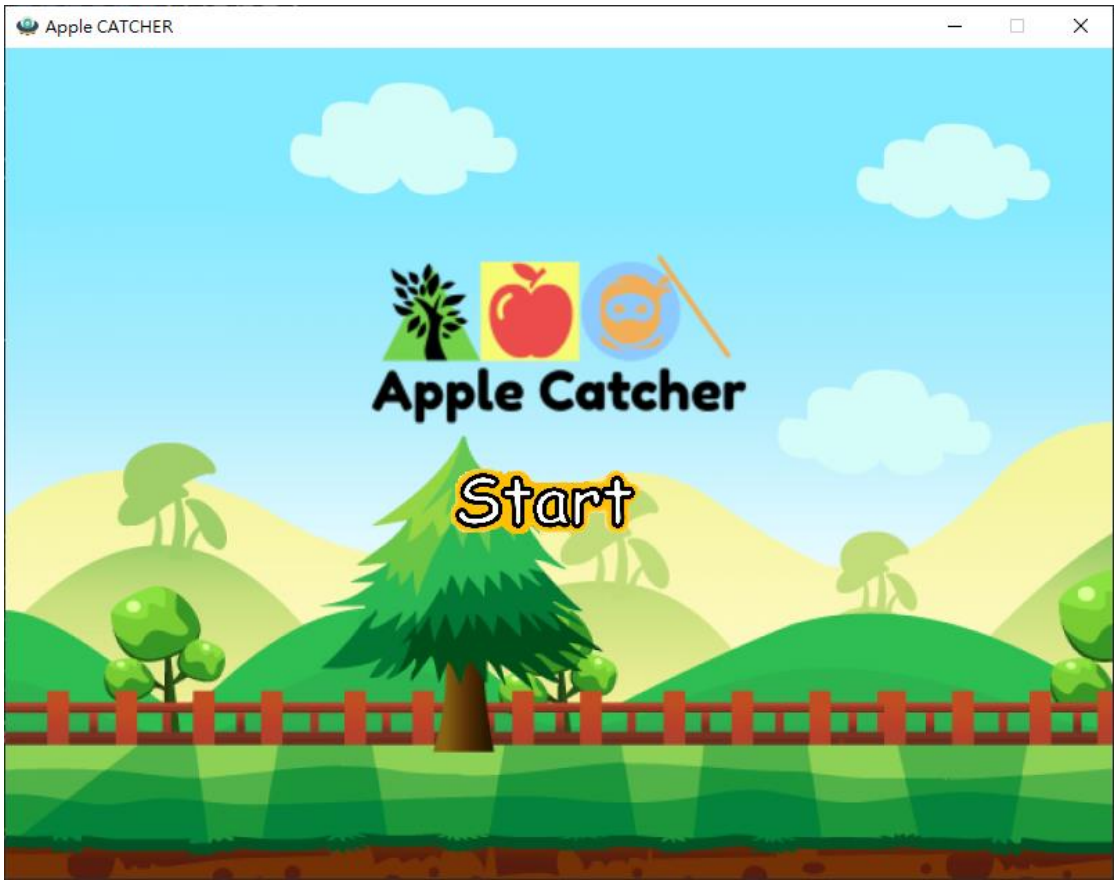


圖 2.1：即時聲控遊戲的主畫面

2.5.2.2 語言選擇畫面

圖 2.2 為遊戲語言選擇畫面，此畫面有三個遊戲語言進行選擇，由左至右分別是英文、中文、台語 (閩南語)。點選圖示即可根據選擇的語言進入遊戲。

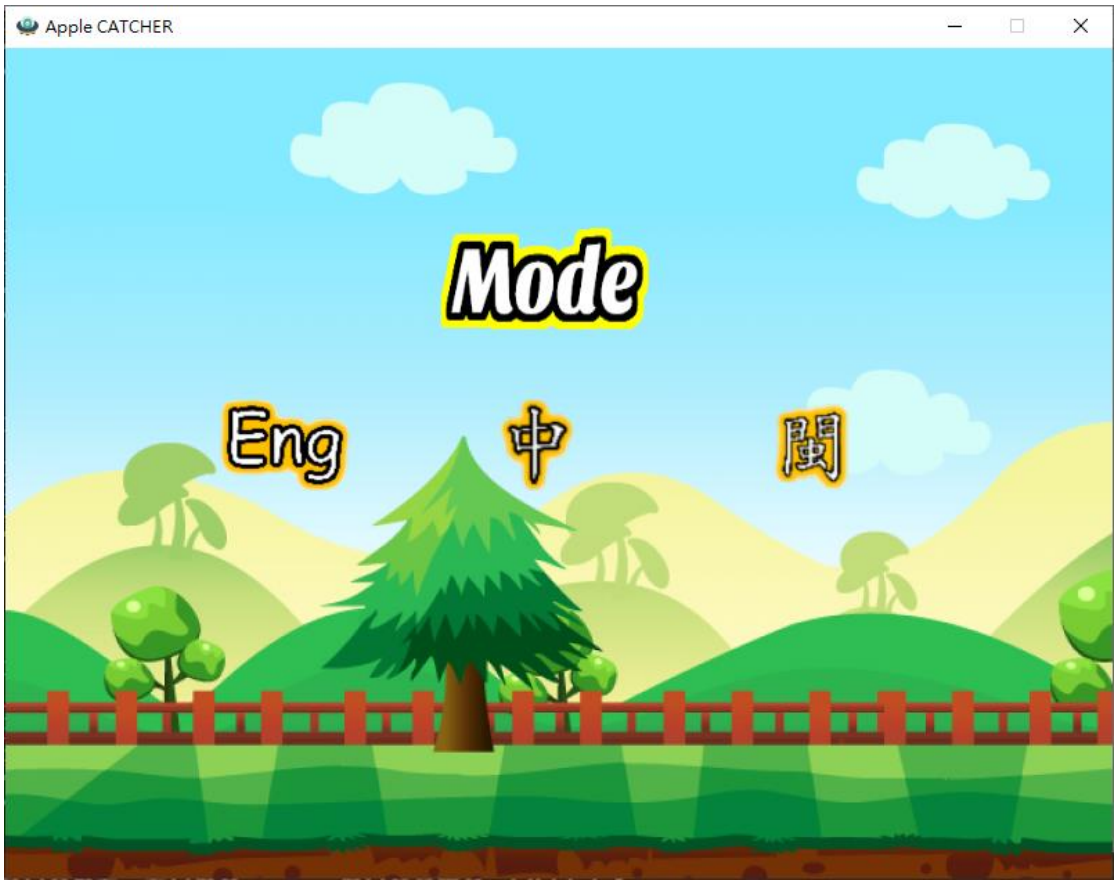


圖 2.2：語言選擇畫面

2.5.2.3 遊戲畫面

圖 2.3 為遊戲畫面。玩家根據選擇的語言說出'上'、'下'、'左'、'右'來控制角色移動、'否'來停止角色移動並裝填子彈，'是'來發射子彈，子彈擊中畫面上蘋果時即能獲得分數。蘋果在被擊中後會消失並在畫面上隨機產生一顆新的蘋果。

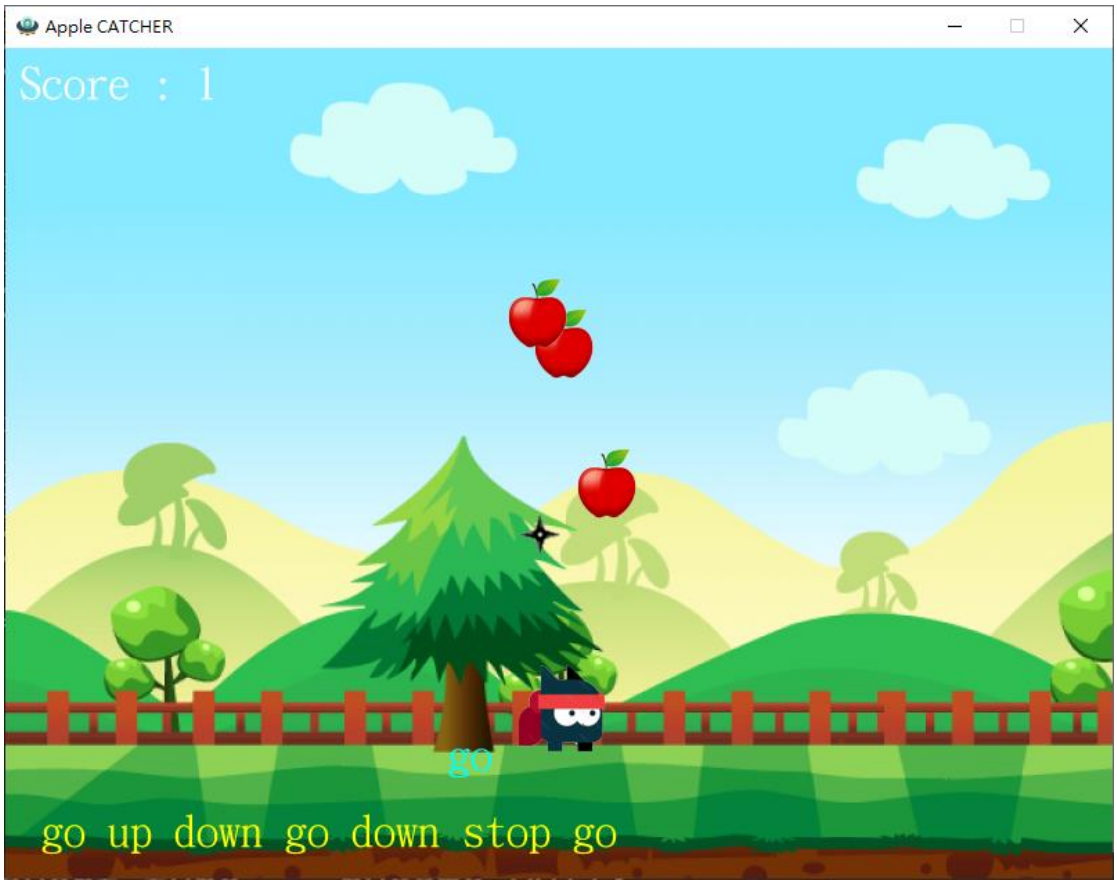


圖 2.3：遊戲畫面

2.6 改進

2.6.1 個人化

個人化語音辨識是一種使用機器學習技術來訓練語音辨識系統以識別特定個體的語音。傳統的語音辨識系統通常使用大規模的語音資料集進行訓練，以識別一般性的語音模型。但是，這樣的模型可能無法有效地辨識不同個體之間的語音差異，因為每個人的語音特徵都是獨特的。

個人化語音辨識旨在解決這個問題，通過使用特定個體的語音資料集來訓練語音辨識系統，使其能夠更好地識別特定個人的語音。個人化語音辨識可以應用於多個領域，包括語音助理、語音指令、語音識別系統等。

個人化語音辨識具有以下優勢：

提高辨識準確性並適應個體差異：不同個體的語音特徵存在差異，例如音調、音量、發音方式等，傳統的語音辨識模型可能難以充分適應這些差異。個人化語音辨識模型可以根據特定個體的語音資料集進行訓練，使得模型更加熟悉特定使用者的語音特徵，從而提高辨識準確性。

提升使用者體驗與安全性：個人化語音辨識模型可以識別特定個體的語音，從而實現更加個性化和定制化的語音應答或操作，亦可用於語音驗證和語音身份識別等應用場景，提升使用者的體驗和滿意度，並提高語音安全性和防篡改能力。

2.6.2 增加詞彙與啟動詞

目前能辨識的詞彙共有 35 個，已經可以為遊戲角色進行移動、選擇、數字等。但是我們不希望這項技術只用於遊戲上，若是需要將詞彙辨識應用與其他的場景上，則需要收集其他的場景常使用到的詞彙，例如對智慧冷氣使用「調高」、「調低」、「溫度」，對智慧餐車使用「前往」、「過來」、「回去」等指令。

在聆聽使用者的指令時，目前程式會每 0.1 秒辨識一次使用者的聲音並執行辨識的結果，但是容易會因為外部雜訊或是辨識錯誤造成執行的指令錯誤。可以使用啟動詞進行解決。

啟動詞是一種用於啟動語音助理的特定詞語或短句。當使用者說出啟動詞時，語音助理會開始監聽和分析使用者的語音指令，並作出相應的回應或執行相關的任務。

2.6.3 端到端模型訓練

目前在進入模型訓練之前，需要先對資料集的語音資料進行對數梅爾頻譜圖的前處理，才使用 CNN 進行語音辨識之訓練。前處理的缺點為需要花時間轉換資料的型態，和使用不同的數學參數也會影響到最後辨識的結果。目前有研究指出，在訓練語音辨識時，直接輸入原始的音訊波型並使用 M5 網路架構：四層的卷積層、relu、池化層和一層全連接層，進行訓練即能在兩次 epochs、三分鐘內獲得正確率約 70%的辨識結果，21 次 epochs 能獲得正確率約 85%的辨識結果 [15]。未來在詞彙辨識等研究時，可以嘗試對直接輸入音訊波型與特徵汲取後訓練的結果差異進行研究。

第三章 卡拉 OK 系統

3.1 動機

從音樂中直接進行語音處理如歌詞辨識、歌聲偵測、音高計算等，準確率容易受到背景音樂、噪聲等因素的影響。目前歌聲分離大多是使用 U-Net 網路模型並以歌唱聲音的頻譜作為輸入。然而如果有弦樂伴奏，則人聲仍然無法被清晰地分離，因為分離的人聲部分還是有可能會包含噪聲。

在處理歌聲分離中人聲部分的噪聲，我們能使用歌聲偵測。歌聲偵測為判斷輸入的音訊片段是否包含歌聲。歌聲偵測從早期的特徵工程，到網路模型如 CNN、LSTM、GRU (Gated Recurrent Unit) [18] 等對於歌聲偵測皆能有不錯的辨識結果，而 LRCN 模型結合了 CNN 的捲積特徵提取和 LSTM 的時間關係。

在此章節中，我們將結合歌聲分離與歌聲偵測，建立一個卡拉 OK 系統。卡拉 OK 系統在獲取純伴奏的音樂時需要使用到歌聲分離，使用歌聲偵測模型，分析出分離的人聲歌唱與非歌唱部分，降低噪聲對後續音高計算的影響。

3.2 目的

我們的目的為將歌聲分離、歌聲偵測、即時計算集成一個卡拉 OK 系統。

我們將利用 spleeter [19] 進行背景音樂與歌聲的分離，訓練歌聲偵測模型對分離的歌聲部分辨識歌唱部分與非歌唱部分以降低歌聲分離所產生的雜音影響。讀取使用者演唱的音高，經過自我相關函數(autocorrelation)計算後同時使用 pygame 進行卡拉 OK 畫面和音樂的輸出。

3.3 基本知識

3.3.1 歌聲分離

歌聲分離是指將混合音訊如歌曲中的主唱聲部和伴奏分開的過程。本章將使用 Spleeter 函式庫歌聲分離。

Spleeter 為使用 Tensorflow，並帶有預訓練模型的人聲分離函式庫。 當使用者使用自身提供的分離音源的資料集時，它能訓練人聲分離模型，並提供已經訓練過的模型來執行不同程度的分離：人聲與伴奏二部分；人聲、鼓、貝斯、其他共四個部分；人聲、鼓、貝斯、鋼琴、其他共五部分。

spleeter 使用的預訓練模型是 U-net 。 U-net 是一種帶有殘差連接(skip connect)的編碼器/解碼器 CNN 架構。使用了 12 層 U-net，編碼器與解碼器各使用 6 層。 每個編碼器層由步幅為 2 且 kernel size 為 5x5 的 2D 卷積、batch normalization 和 α 為 0.2 的 Leaky-ReLU 。在解碼器中，使用步幅為 2 且 kernel size 為 5x5 的

反捲積、batch normalization、ReLU，並在前三層使用 50% 的 dropout，在最後一層使用 sigmoid 激活函數。 此模型使用 Adam Optimizer 進行訓練。

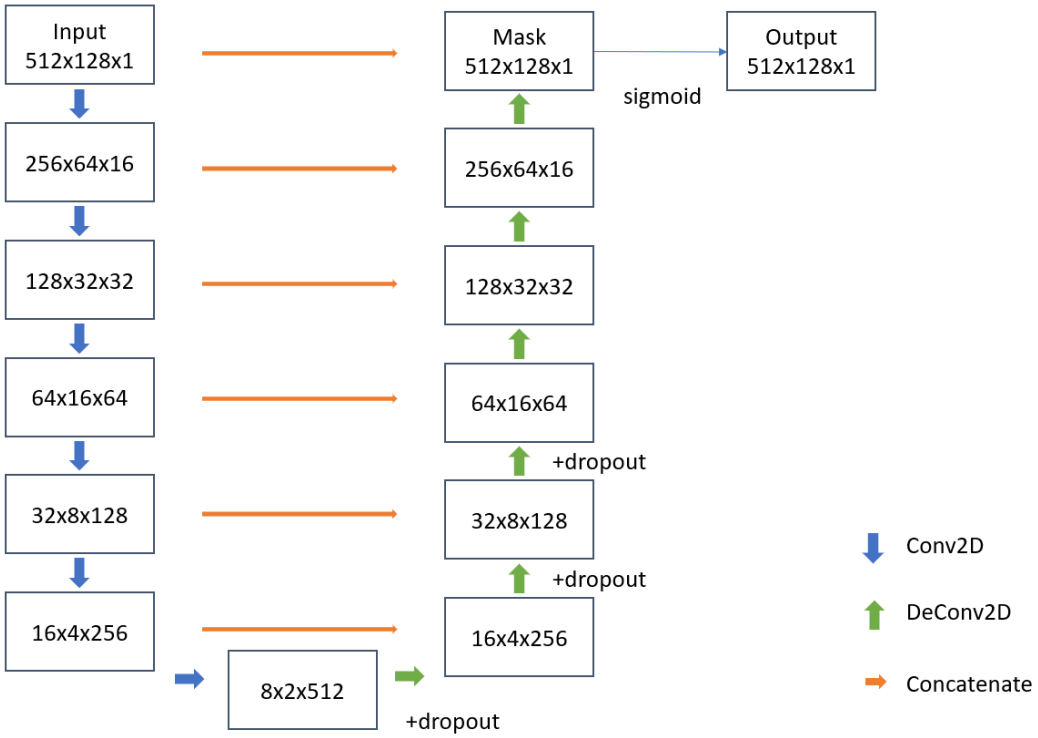


圖 3.1：U-net 網路架構圖

使用 Spleeter 將人聲與音樂部分分離後，人聲部分會在後續進行歌聲偵測去除雜音後進行音高檢測獲取歌手音高。

3.3.2 歌聲偵測

唱歌聲音檢測或稱為聲音檢測，是一個分類任務，目的是確定一個給定的音訊片段中是否有唱歌聲音。這個過程是一個關鍵的預處理步驟，可以用來提高其他任務的性能，例如自動歌詞對齊、唱歌旋律轉譜、唱歌聲音分離、聲音旋律提取等等。

在 spleeter 分離出背景音樂和人聲後，人聲的部分仍然會有些許雜音，雜音會被自我相關函數辨識為一個音高，我們不希望在程式

畫面上顯示非歌手歌唱的部分。

本章使用 LRCN 網路模型對分離出的人聲進行歌聲偵測，確認出歌聲與非歌唱的部分。

LRCN 為一個將 CNN 和長短期記憶網絡(LSTM)串聯在一起。CNN 層將音頻特徵融合成一個向量，LSTM 層則是用來學習它們的時序關係，以輸出唱或不唱的標籤。

在 LSTM 中，每個單元包括了一個記憶單元和三個門控制單元，分別是遺忘門、輸入門和輸出門。其中，遺忘門用於控制前一個時間步的記憶是否要被遺忘，輸入門用於控制當前時間步的新信息是否要被添加到記憶中，輸出門用於控制當前時間步的記憶是否要被輸出，LSTM 可以在長時間序列數據中有效地保留和遺忘重要的信息，以及控制信息的輸入和輸出。

LRCN 與 LSTM 層類似，但輸入和循環都是以卷積形式去計算。 LRCN 模型具有一個輸入層，大小為結合的特徵向量，一個 sigmoid 單元的輸出層，以及三個隱藏層，並為每個幀區塊輸出介於 0(沒有唱歌聲音)和 1(唱歌聲音)之間的值。通過結合特徵提取的 CNN 和 LSTM，模型將連續音頻幀的融合特徵傳遞到新的向量中，然後按順序綜合其時間動態。

以下為各個控制單元的數學公式：

輸入門 i(x)：

$$\boldsymbol{i_t = \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \odot C_{t-1} + b_i)} \qquad \textbf{式3.1}$$

輸出門 o(x)：

$$\boldsymbol{o_t = \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \odot C_t + b_o)} \qquad \textbf{式3.2}$$

遺忘門 f(x)：

$$\boldsymbol{f_t = \sigma\left(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \odot C_{t-1} + b_f\right)} \qquad \textbf{式3.3}$$

輸入 LRCN cell C(x)：

$$\boldsymbol{C_t = f_t \odot C_{t-1} + i_t \odot tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c)} \qquad \textbf{式3.4}$$

輸出 LRCN cell H(x)：

$$\boldsymbol{H_t = o_t \odot tanh(C_t)} \qquad \textbf{式3.5}$$

其中：

W＝ 權重矩陣；b＝ 偏差向量；σ ＝Sigmoid 函數；*＝ 逐元素相乘；conv.＝ 卷積運算；⊙ denotes the Hadamard product

3.3.3 音高檢測

音高為聲音頻率的高低程度。由發音體在一定時間內所發生的顫動次數或音波次數而決定。音波振動次數多，聲音就高，反之則低。聲音的音高通常用基頻來描述，基頻是聲波中最低的頻率分量，對應聲音的感知音高。

音高檢測，也被稱為基頻估計，為計算周期性或準周期性訊號的音高或基本頻率，主要用於語音或樂音的訊號處理中。音高檢測可以單獨從時域或頻域的角度實現，也可以同時利用時域和頻域進行檢測。本章使用自我相關函數(Autocorrelation)進行音高檢測。

自我相關函數是一種聲調檢測演算法，其透過計算輸入信號的自相關來運作。自我相關函數測量了訊號與其時間延遲版本之間的相似程度，並隨著時間延遲的變化而變化。

在自我相關函數中，會針對一系列可能的聲調週期計算自我相關函數，然後選擇自相關值最高的週期作為估計的聲調週期。然後，基頻(F0)會被計算為聲調週期的倒數。

自我相關函數的一個優點是它對噪聲相對穩健，對低信噪比的訊號工作良好。它還可以處理具有複雜頻譜內容的訊號並檢測多個泛音。然而，它對計算負擔較重，特別是對於較長的信號持續時間，並且對某些類型的噪聲(例如與訊號呈調和關係的周期性噪聲)較為敏感。

自我相關函數的計算公式為：

$$\boldsymbol{R(k) = 1/N * sum\big(x(n) * x(n-k)\big)} \qquad \textbf{式3.6}$$

其中：

k 是時間延遲；x(n)是輸入信號；N 是輸入信號 x(n)的長度。

自我相關函數 R(k)測量信號 x(n)和其時間延遲版本 x(n-k)之間的相似度。自我相關函數通常在一系列落度範圍內進行計算，從 0 到最大落值，對應於信號的最小預期週期。

為了估計調期，自我相關函數通常通過除以落 0 處的自相關值進行正規化，然後選擇在正規化自我相關函數中最高峰所對應的落作為估計的調期。然後，基頻(F0)被計算為調期的倒數。

3.3.4 多線程

執行緒是作業系統能夠進行運算排程的最小單位，一個執行緒包含了執行緒 ID、程式計數器(program counter)、暫存器集(register set)和堆疊(stack)。一個執行緒與其他在同個程序下的執行緒共享程式分段、資料分段與其他作業系統的資源。單執行緒為一個程序只有一個執行緒控制，多執行緒為一個程序有多個執行緒控制，也就能同時執行多個任務。

多執行緒的程式設計主要有四種好處：回應能力，資料分享，

節省資源，可擴展性。回應能力指在互動式的應用程式中，即使程式的一部分被其他任務阻擋或是在進行冗長的計算，仍舊能依靠其他執行緒使程式繼續執行或是處理回應使用者的部分，單執行緒任務執行結束前無法回應使用者。資料分享指同個程序下的執行緒共用記憶體與資料，而不同的程序需要使用共享記憶體(shared memory)和訊息傳遞(message passing)等方式才能共用資料。節省資源指通常為執行緒創建所需的分配記憶體與資源所消耗的時間與記憶體比起創建程序還要少，且在切換不同執行緒也比較快能完成。可擴展性指在多處理器的架構中，執行緒可以在不同的核心中平行執行，而單執行緒的程序不論有多少個處理器都只能在單個處理器上執行。

在某些類型的應用程序中，多執行緒可以提供顯著的性能優勢，特別是涉及大量輸入/輸出操作或並行處理的應用程序。通過將任務劃分為可以同時執行的較小子任務，多執行緒可以實現更快的執行和更好的資源利用。

但是，多執行緒也帶來了一些挑戰，例如需要同步訪問共享資源以防止衝突並確保數據完整性。此外，與單線程編程相比，為多執行緒編程可能更加複雜，因為開發人員必須小心避免多個線程同時運行時可能出現的競態條件、死鎖和其他潛在問題。

在 Python 中，可以使用內建的 threading 模組實現多線程。此模組提供了一個簡單易用的介面，用於創建和管理線程。

在 python 中因為存在著全局解釋器鎖定(Global Interpreter Lock, GIL) [37]，同一時間只有一個執行緒可以執行 Python 程式碼，所以在 python 使用多執行緒並不會使得程式執行的比較快，必須使用多

處理程序(multiprocess)才能真正執行平行運算。但是如果需要同時運行多個 I/O 密集型任務時，多執行緒在共用記憶體與資料下仍然是一個合適的方式。

3.4 實驗程式

本節會介紹卡拉 OK 的主程式和歌聲辨識所使用的訓練程式。

3.4.1 卡拉 OK 系統

3.4.1.1 下載音樂

程式碼區段 3.1 主要目的為為從 YouTube 下載音樂的函式，在使用者輸入網址後，將影片或音樂 mp4 檔下載，並將 mp4 檔轉換成 wav 檔供後續處理使用。

函式的輸入為 YouTube 網址與選擇是否需要背景影片的布林值(bool)，回傳音樂名稱，產生的檔案為 tmp.wav 與儲存於 localfile/mp4 資料夾的音樂名稱 mp4 檔。

函式主要分為 3 個部分，第一部分為確認 tmp.wav 是否存在，因為歌曲的名稱不一定為英文，若是為其他的語言有可能會因為編碼的問題而無法下載音樂，所以我們會先移除上次的下載暫存檔以利下個部分的執行。第二部分為下載音樂檔，首先先從函式的輸入取得 YouTube 連結和是否在背景播放影片的選項，選擇下載純音樂或有影片的 mp4 檔後，下載並取名為 tmp.mp4，並將原本的歌曲名去掉不符合取名規則的冒號、斜線與反斜線後儲存於變數中。第三部分為將 mp4 檔轉為 wav 檔，因為我們需要純音樂檔才能進行後續的人聲分離和頻率分析，於是我們在此使用 ffmpeg 將下載的

tmp.mp4 轉為 wav 檔並將檔名改回原本歌曲的名稱，最後再將歌曲的名稱輸出給下個函式使用。

```
def Download_music(link,video):
    #1
    song_there = os.path.isfile("tmp.wav")
    if song_there:
    try:
        os.remove("tmp.mp4")
        os.remove("tmp.wav")
        os.remove('output/tmp/accompaniment.wav')
        os.remove("output/tmp/vocals.wav")
     except:
         pass
    #2
    yt=YouTube(link)
    if video == True:
        t=yt.streams.get_highest_resolution()
        v = t
    elif video == False:
        t=yt.streams.filter(only_audio=True, file_extension='mp4')
        v = t[0]
    v.download(filename='tmp.mp4')

    name= v.title
    symbol = [':','/','\\']
    for s in symbol:
        if name.find(s) >= 0:
            name = name.replace(s,' ')

    #3
    video = ffmpeg.input('./tmp.mp4')
    audio = video.audio
    stream = ffmpeg.output(audio, "tmp.wav")
    ffmpeg.run(stream)
    shutil.move('tmp.mp4', 'localfile/mp4/' + name + '.mp4')
    return name
```

程式碼區段 3.1：從 YouTube 下載音樂

3.4.1.2 spleeter 進行人聲分離

程式碼區段 3.2 主要目的為使用 spleeter 進行人聲分離的函式，使用程式段一輸出的 tmp.wav 進行人聲分離，分離後會產生純人聲的 vocals.wav 與純背景音樂的 accompaniment.wav，再根據選擇的 onvocal 或是 offvocal 合併為三音軌的 wav 檔，onvocal 的合成是將

tmp.wav 與 vocals.wav 的左聲道合併，offvocal 的合成是將 accompaniment.wav 與 vocals.wav 的左聲道合併，合併後設定取樣率(sample rate)為 16000，最後輸出為 KaraOKE.wav 並複製一份更名為音樂名稱的 wav 檔存儲存於 localfile/onvocal/或 localfile/offvocal/下供以後重複使用。

函式的輸入為音樂名稱與選擇的 onvocal 或 offvocal 模式，需要使用程式段一輸出的 tmp.wav，不回傳任何參數，產生在 output/tmp/資料夾的 KaraOKE.wav 與 localfile/onvocal/或 localfile/offvocal/的音樂名稱 wav 檔。

函式主要分為 3 個部分，第一個部分為使用 spleeter 分離人聲與背景音樂的部分，因為 spleeter 需要使用 cmd 進行呼叫，所以使用 subprocess 模組的子程序(subprocess)功能執行並等待正常結束後再繼續執行後續部分。第二部分為處理音軌，我們將第一部分分離出的人聲取出左聲道，並與先前選擇的是否去人聲的結果，將人聲的左聲道與原本 wav 檔或是 spleeter 分離出的純背景音樂檔合併為三音軌的 wav 檔，以供後續歌手頻率分析和卡拉 OK 主程式進行使用。第三部分為更名和移動檔案，將合併好的 wav 檔依照選擇的是否去人聲，將檔案分別放入 onvocal 與 offvocal 資料夾內，下次若要再使用同一首歌進行卡拉 OK 演唱的話就不需要再次下載音樂。

```
def download_spleeter(song_name = 'KaraOKE',mode = 'offvocal'):
    #1
    command_line = "spleeter separate -o output/ tmp.wav"
    args = shlex.split(command_line)
    p = subprocess.Popen(args)
    p.wait()

    #2
    if mode == 'offvocal':
    accompaniment_channel_1,accompaniment_channel_2 = Splitting_stereo_audio('output/tmp/accompaniment.wav')
    elif mode == 'onvocal':
    accompaniment_channel_1,accompaniment_channel_2 = Splitting_stereo_audio('./tmp.wav')

    vocal_channel_1,vocal_channel_2 = Splitting_stereo_audio(r"output/tmp/vocals.wav")
    mutli_channel = AudioSegment.from_mono_audiosegments(accompaniment_channel_1,accompaniment_channel_2,vocal_channel_1)
    mutli_channel.export("output/tmp/KaraOKE.wav",format="wav")

    sound = AudioSegment.from_file("output/tmp/KaraOKE.wav")
    sound = sound.set_frame_rate(16000)
    sound.export("output/tmp/KaraOKE.wav",format="wav")
    shutil.copyfile('./output/tmp/KaraOKE.wav', './localfile/KaraOKE.wav')

    #3
    if mode == 'offvocal':
        shutil.move('./localfile/KaraOKE.wav', './localfile/offvocal/' + song_name + '.wav')
    elif mode == 'onvocal':
        shutil.move('./localfile/KaraOKE.wav', './localfile/onvocal/' + song_name + '.wav')
    os.remove('./tmp.wav')
```

程式碼區段 3.2：spleeter 進行人聲分離

3.4.1.3 歌手音高分析

程式碼區段 3.3 主要目的為歌手音高(pitch)分析，讀取程式段二產生的音樂名稱 wav 檔，將第三音軌 vocals.wav 的左聲道取出，使用預訓練的模型分析歌手雜音的部分，並對此音軌進行音高分析後

互相比對，將音高分析的結果去掉被判定成雜音的部分後，儲存於陣列並輸出。

函式的輸入為音樂名稱的 wav 檔名，需要讀取音樂名稱的 wav 檔與預訓練的模型，回傳音高分析的結果陣列。

函式主要分為 2 個部分，第一部分為去除雜音，因為我們不能保證 spleeter 分離出的人聲沒有任何雜音，所以我們使用機器學習嘗試地去找出雜音的部分並去除。首先讀取三音軌的 wav 檔並選擇人聲的左聲道的音軌，再經過訓練好的歌聲辨識模型針對整首歌進行分析，尋找出雜音與非雜音的部分。第二部分為音高分析，使用 freq_from_autocorr()將人聲的左聲道的音軌以每 0.1 秒的長度轉換成頻率，並與先前的雜音分析進行比對，若是被訓練模型評斷為是雜音或是頻率超過 2000 赫茲，則會被認定是非歌手的聲音而將此 0.1 秒的頻率設定為 0 赫茲，得出歌手所唱的音高後存入陣列後回傳供主程式使用。

```
def wav_pitch(wav):
    '''
    read wav's 3rd channel and get the singer's pitch
    '''
    sr, data = wavfile.read(wav)
    pre_out_data = pred.preprocess(wav.split('.wav')[0])
    pre_out_pred = pred.predict(pre_out_data)
    target = data[:,2]
    time = len(target)/sr
    hz = []

    for i in range(int(time)*10):
        pitch = freq_from_autocorr(target[i*1600:(i+1)*1600], sr)
        if pitch > 2000:
            pitch = 0
        if pre_out_pred[i//10] == 0:
            hz.append(0)
        else:
            hz.append(pitch)
    return hz
```

程式碼區段 3.3：歌手音高分析

3.4.1.4 麥克風輸入執行緒

程式碼區段 3.4 主要目的為麥克風輸入執行緒，將麥克風的輸入資料先填滿陣列後，再使用迴圈無限接收麥克風的輸入並將超過陣列上限的值丟棄，直到 self.record_thread_is_alive 為 False 為止。

函式的輸入為麥克風的輸入，設定 self.frame 類別(Class)內的屬性(Attribute)陣列。

函式主要分為兩個部分，第一部分為 frame 此陣列在還沒滿的時候的情況，使用 for 迴圈讀取從麥克風輸入的聲音並寫入至 frame 陣列中。第二部分為 frame 此陣列在全滿的時候的情況，與第一部分不同的地方在於這裡要處理兩個問題：執行緒結束與 frame 陣列溢出。首先執行緒結束的解決方法是使用物件導向的方式，利用

self.record_thread_is_alive 此項屬性的布林值去判斷，在第一部分 frame 被填滿的時候會被設定為 True，等到主程式結束或是不需要錄音的時候則會被設定為 False；frame 陣列溢出的問題使用雙向陣列 deque 解決，deque 的優點為可從陣列的兩端 pop 和 push 資料，也可以使用迭代器(iterator)直接讀取陣列內的資料。這裡在 frame 陣列溢出的時候使用 popleft()將最舊的資料移出陣列，以防後續主程式讀取陣列與顯示於螢幕出現問題。

```
def record_thread(self):
    #1
    self.frame_counter = 0
    self.frame = deque()
    for i in range(self.n_frame):
        z= self.stream.read(self.frame_size)
        self.frame += [z]
        self.frame_counter += 1

    #2
    self.record_thread_is_alive = True
    while self.record_thread_is_alive:
        z = self.stream.read(self.frame_size)
        self.frame += [z]
        self.frame_counter += 1
        self.frame.popleft()
```

程式碼區段 3.4：麥克風輸入執行緒

3.4.1.5 設定播放音樂

程式碼區段 3.5 主要目的為設定播放音樂，讀取三音軌的 wav 檔後，將前二音軌取出並儲存成 tmp.wav 後，使用 pygame.mixer.music 讀取 tmp.wav。若為不播放背景影片模式，則開始播放音樂且記錄開始時間。

函式需要讀取三音軌的 wav 檔，在不播放背景影片模式下設定音樂開始時間至 self.audio_start_time 屬性，在播放背景影片模式下

無輸出。

函式主要分為兩個部分，第一部分為讀取三聲道的音樂檔，先使用 wavfile 讀取音樂後，numpy 的 vstack 獲得左右聲道的資料後進行轉置矩陣，最後寫入 tmp.wav 檔並讀取此檔案來播放音樂。因為使用 vstack 函式時會使得原本音樂資料的陣列從(n,3)讀取成(2,n)，我們需要(n,2)才能寫入 wav 檔，於是將讀取的資料進行轉置後即可將(2,n)之資料轉成(n,2)的陣列。第二部分為當不需要背景影片的時候會開始播放音樂並使用 pygame 的 get_ticks 函式紀錄音樂開始的時間，紀錄音樂開始的目的為讓後續歌手音高的顯示能與音樂對齊，而不會造成顯示延遲，使得解決使用者唱對但卻顯示錯誤的問題。若是需要播放背景影片的話，音樂會在影片處理的時候開始播放，進而減少音樂開始到背景影片開始之間的時差。

```
1.  def audio_setting():
2.      #1
3.      fs, data = wavfile.read(self.audio)
4.      audio = np.vstack([data[:, 0],data[:, 1]])
5.      audio = audio.transpose()
6.      wavfile.write('output/tmp.wav', fs, audio)
7.      pg.mixer.music.load('output/tmp.wav')
8.
9.      #2
10.     if self.video == None:
11.         pg.mixer.music.play()
12.         self.audio_start_time = pg.time.get_ticks()
```

程式碼區段 3.5：設定播放音樂

3.4.1.6 顯示背景影片

程式碼區段 3.6 主要目的為顯示背景影片，使用 opencv 讀取影片畫面寬高、每秒顯示影格數(frame per second,FPS)，將資訊儲存於

類別內的屬性供畫面設定使用。開始播放音樂並儲存開始時間。而後進入迴圈中，設定影片播放速度，讀取影片內容，將影片內容轉成位元組(bytes)後儲存於緩衝(buffer)並使用 pg.image.frombuffer()讀取，最後將讀取的影片內容繪製在視窗上，等待 show_spectrum()函式完成後，將繪製的視窗顯示於螢幕。

函式需要讀取背景影片，不回傳任何結果，顯示繪製的視窗於螢幕。

函式主要分為五個部分，第一部分為使用 opencv 進行影片讀取，在成功讀取 mp4 影片後將每秒的幀數儲存於參數中供後續 pygame 設定。第二部分為 pygame 的設定，將 pygame 主程式的視窗寬高依照讀取影片的寬高進行調整，並使用 pygame 的 clock 函式進行時間的設定，使得後續的背景影片幀數、背景影片與音樂的對齊和背景影片與歌手音高的對齊能夠執行。第三部分為音樂播放，將設定播放音樂函式設定好的音樂開始播放並紀錄音樂開始的時間。第四部分為影片播放，在 while 迴圈底下每當影片的資料讀取尚未完畢時，即會將讀取的資料轉換成 bytes 後放入 pygame 的影像緩衝區使得 pygame 去播放影片，若是影片的資料讀取完畢時，則會跳出 while 迴圈並結束主迴圈。第五部分為畫面的刷新與歌手音高的顯示，需要使用 screen.blit 將影片繪製在視窗上，再進入歌手音高的函式，將歌手音高繪製在影片上，最後將整個繪製好的視窗顯示於螢幕。

```
def background_video(mp4):
    #1
    video = cv2.VideoCapture(mp4)
    success, video_image = video.read()
    fps = video.get(cv2.CAP_PROP_FPS)

    #2
    self.screen = pg.display.set_mode(video_image.shape[1::-1])
    self.width = pg.Surface.get_width(self.screen)
    self.height = pg.Surface.get_height(self.screen)
    clock = pg.time.Clock()

    #3
    pg.mixer.music.play()
    self.audio_start_time = pg.time.get_ticks()
    run = success

    #4
    while run:
        clock.tick(fps)
        success, video_image = video.read()
        if success:
            video_surf = pg.image.frombuffer(
                video_image.tobytes(), video_image.shape[1::-1], "BGR")
        else:
            run = False
            self.main_loop = False

        #5
        self.screen.blit(video_surf, (0, 0))
        show_spectrum(self, keyboard)
        pg.display.flip()
```

程式碼區段 3.6：顯示背景影片

3.4.1.7 獲取歌聲資料

程式碼區段 3.7 主要目的為獲取歌聲資料，將 Mic 此 class 內的 frame 陣列所存的麥克風輸入訊號複製至暫時的緩衝陣列中，再使用 numpy 的 frombuffer 將緩衝陣列中的訊號以 int16 的格式存於 self.mic_array 供後續轉換頻率使用。因為從麥克風輸入的訊號為

byte，但是後續訊號處理需要用 int 的格式才需要額外進行轉換。

輸入為 Mic.frame 陣列的麥克風輸入訊號，設定 self.mic_array 此 ndarray。

```
def get_wav_mic_array():
    b0= b''
    for k in range(-frame_for_spectrum,0):
        b0 += self.Mic.frame[k]
    self.mic_array= np.frombuffer(b0,'int16')
```

程式碼區段 3.7：獲取歌聲資料

3.4.1.8 計算音高之頻率

程式碼區段 3.8 主要目的為使用自我相關函數計算音高之頻率，首先先對輸入訊號進行快速傅立葉變換(Fast Fourier Transform, FFT)，找出第一個最低點，再找出此點後的最高峰的位置後，使用二次插值估計訊號最大值的位置，最後再除以取樣率後輸出頻率結果。輸入為訊號陣列與取樣率，回傳音高的頻率。

```
def freq_from_autocorr(sig, fs):
    corr = fftconvolve(sig, sig[::-1], mode='full')
    corr = corr[len(corr)//2:]
    d = diff(corr)

    try:
        start = find(d > 0)[0]
    except:
        fs_px= 0
        return fs_px

    peak = argmax(corr[start:]) + start
    px, py = parabolic(corr, peak)
    return fs / px
```

程式碼區段 3.8：計算音高之頻率

3.4.1.9 頻率處理

程式碼區段 3.9 主要目的為頻率處理。計算當前音樂的時間後，從歌手的音高資訊陣列中取出對應的時間點，並與使用者的麥克風輸入，計算過的音高做走音比較，最後再將音高轉換成音名。

輸入為歌手的音高資訊陣列 self.wav_pitch_array 與使用者麥克風輸入的訊號陣列 self.mic_array，使用者的音高 mic_freq、歌手的音高 wav_freq、音高正確次數 self.pitchMatching、使用者的音高音名 mic_note 與歌手的音高音名 wav_note。

函式主要分為五個部分，第一部分為確認當前音樂播放時間，因為我們已經將歌手的音高以間隔 0.1 秒的長度放入陣列，而我們需要知道當下距離開始已過了幾秒才能確定需要顯示的音高與背景音樂同步，這裡以 pygame 的 get_ticks 函式減去 audio_start_time 參數所存取的音樂開始時間獲得當下的時間點，若當下的時間點等於歌手音高的陣列時，將認定為是歌曲結束，會關閉主程式與停止麥克風錄音。第二部分為計算使用者與歌手音高，使用者的音高在經過程式段八的 freq_from_autocorr()函式後獲得，而歌手的音高會從程式段三輸出的陣列獲得。第三部分為去除麥克風輸入的雜音，使用者的聲音會在絕對值後取平均，並與程式剛開始的錄音的聲音振幅比較，因為程式剛開始的時候通常是音樂的前奏，麥克風只會接受到環境因與自身的電子雜訊，所以若是輸入的聲音超過振幅平均加上三個標準差時就判定為不是雜音。如果算出的頻率超過 2000 赫茲的話，因為 2093 赫茲是高三個八度的 Do，一般流行音樂不會存在這麼高的音高，所以也會去掉。第四部分為判斷走音，若使用者的音高與歌手的音高相差超過 5 赫茲的話，將會被判定為是走音。

第五部分為量化音高，會將算出來的赫茲轉換成音名，即 C、C#、D、D#、E、F、F#、G、G#、A、A#、B。

```
def freq_process():
    #1
    frame = pg.time.get_ticks() - self.audio_start_time
    if frame//100 == len(self.wav_pitch_array):
        pg.quit()
        self.Mic.stop()
        sys.exit()

    #2
    mic_freq= freq_from_autocorr(self.mic_array, self.Mic.sample_rate)
    wav_freq = self.wav_pitch_array[frame//100]

    #3
    mic_not_noise= False
    wav_not_noise= False
    en0= abs(self.mic_array).mean()
    if en0 > self.Mic.first_frame['mean'] + self.Mic.first_frame['std']*3:
        mic_not_noise= True
    if mic_not_noise==False or mic_freq > 2000:
        mic_freq=-20

    #4
    pitchHit= False
    if (abs(wav_freq - mic_freq) <= 5 or abs(wav_freq/2 - mic_freq) <= 5):
        self.pitchMatching += 1
        pitchHit= True

    #5
    _, mic_note= pitchQuantization(mic_freq)
    _, wav_note= pitchQuantization(wav_freq)
```

程式碼區段 3.9：頻率處理

3.4.1.10 卡拉 OK 主畫面顯示

程式碼區段 3.10 主要目的為卡拉 OK 主畫面顯示，繪製左下與右下的藍色黃色長條背景格，顯示使用者與歌手所唱的音名與音高相同的次數，顯示歌曲的當前時間與總時長，計算歌手音高與使用者音高顯示於視窗的位置並放入陣列。

輸入為程式段九的輸出：使用者的音高 mic_freq、歌手的音高 wav_freq、音高正確次數 self.pitchMatching、使用者的音高音名 mic_note 與歌手的音高音名 wav_note，輸出為歌手與使用者音高顯示於視窗的位置陣列 self.fQ。

函式主要分為五個部分，第一部分為繪製左下與右下的藍色黃色長條背景格，為了讓後續顯示的資料字體比較容易被看見。第二部分為顯示使用者與歌手所唱的音名與音高相同的次數，aFont 是在主函式下宣告的字體格式，而且 pygame 不能直接在畫面上繪製文字，需要先使用 Font.render()將文字轉為圖像，再使用 blit()繪製到畫面上。第三部分為顯示歌曲的當前時間與總時長，當前時間是從程式段九的第一部分獲得毫秒再轉成分：秒的格式，總時長則是從程式段三輸出的陣列總長度除以 10 而獲得秒後再轉成分：秒的格式。第四部分為將第二與第三部分的訊息繪製在畫面上，繪製到第一部分的長條背景格的上層。第五部分為確認使用者與歌手所唱的音要顯示於畫面的位置並存於 List 中，再把超過 List 上限的資料刪除，因為需要知道灰色直條移動的位置，所以並未一起在這個地方輸出。

```
        #1
        pg.draw.rect(self.screen, pg.Color('blue'),[(2, self.height-20),(80,20)])
        pg.draw.rect(self.screen, pg.Color('blue'),[(2, self.height-40),(80,20)])
        pg.draw.rect(self.screen, pg.Color('yellow'),[(2, self.height-60),(80,20)])
        pg.draw.rect(self.screen, pg.Color('blue'),[(self.width-140,self.height-
    20),(140,20)])

        #2
        aMsg= '{}, {:.0f}'.format(mic_note, mic_freq)
        aText= aFont.render(aMsg, True, pg.Color('white'))
        bMsg= '{}, {:.0f}'.format(wav_note, wav_freq)
        bText= aFont.render(bMsg, True, pg.Color('yellow'))
        cMsg= '{}'.format(self.pitchMatching)
        cText= aFont.render(cMsg, True, pg.Color('blue'))

        #3
        current_time = time.strftime('%M:%S', time.gmtime(frame//1000))
        dMsg= '{} / {}'.format(current_time,self.audio_length)
        dText= aFont.render(dMsg, True, pg.Color('white'))

        #4
        self.screen.blit(aText, (2, self.height-20))
        self.screen.blit(bText, (2, self.height-40))
        self.screen.blit(cText, (2, self.height-60))
        self.screen.blit(dText, (self.width-140,self.height-20))

        #5
        while mic_freq > 440: mic_freq = mic_freq/2
        while wav_freq > 440: wav_freq = wav_freq/2
        mic_freq_h= self.height - mic_freq/440 * self.height
        wav_freq_h= self.height - wav_freq/440 * self.height
        self.fQ += [(mic_freq_h, wav_freq_h)]
        while len(self.fQ)>128:
            self.fQ.popleft()
```

程式碼區段 3.10：卡拉 OK 主畫面顯示

3.4.1.11 音高顯示

程式碼區段 3.11 主要目的為音高顯示，繪製灰色直條並定位 x 軸位置，將歌手與使用者音高根據定位的 x 軸位置以白色圓圈與黃色圓圈繪製在視窗上。

輸入為歌手與使用者音高顯示於視窗的位置陣列 self.fQ，將歌

手與使用者音高顯示於視窗。

函式主要分為二個部分，第一部分為繪製灰色直條，這裡使用的方式是利用 self.frame_counter 此計數器除以 self.n_frame 的餘數，再乘上畫面寬度除以 self.n_frame 當作灰色直條 x 軸的位置。第二部分為繪製使用者與歌手的音高於畫面上，在得到灰色直條的 x 軸位置後，將使用者與歌手的音高資訊從 self.fQ 取出，並將資訊以使用者為白色圓圈歌手為黃色圓圈繪製於畫面，這裡使用迴圈是因為我們不希望唱的音高只顯示一次就消失，所以我們將資料存在 self.fQ 中，self.fQ 在程式段十設定總長度為 128，也就是我們會顯示最後 128 筆資料與螢幕上，再者因為 x 軸的設定，在灰色直條向右移動的同時，過去的資料會看起來是在向左移動。

```
def draw_pitch():
    #1
    self.frame_counter += 1
    x= (self.frame_counter % self.n_frame) * self.width / self.n_frame
    pg.draw.line(self.screen, pg.Color('gray'),(x,self.height),(x,0) , 5)

    #2
    for n in range(len(self.fQ)):
        mic_x_axis= x+ (n-len(self.fQ)) *8
        mic_y_axis= self.fQ[n][0]

        wav_x_axis= x+ (n-len(self.fQ)) *8
        wav_y_axis= self.fQ[n][1]

        pg.draw.ellipse(self.screen, pg.Color('white'), [(mic_x_axis, mic_y_axis), (16,16)],4)
        pg.draw.ellipse(self.screen, pg.Color('yellow'), [(wav_x_axis, wav_y_axis), (12,12)],4)
```

程式碼區段 3.11：音高顯示

3.4.2 歌聲辨識模型

3.4.2.1 訓練主函數

程式碼區段 3.12 主要目的為輸出訓練結果和設定模型訓練參數。輸出訓練結果為先開啟一個 csv 檔並等待模型訓練輸出結果之後再將其資料寫入 csv 檔。

設定模型訓練參數的部分，首先我們先選擇 6 項特徵，分別是 'feat_spec_all'、 'feat_mfcc'、 'feat_MFCC_delta'、 'feat_MFCC_delta_delta'、 'feat_lpc'、 'feat_plp'，並與資料集的檔案位置一起初始化 LRCN 模型。

呼叫訓練模型函數設定 load = False 為重新訓練模型, step_size = 25 指讀取資料集時陣列每步長 25 個單位，對應 0.1 秒。訓練模型後會得到準確率、精確率、召回率、F1 值，四項指標會寫入 csv 檔。

```
def LRCN_Standard(dataset_h5_path):
    # Open a CSV file
    res_csv = open('.\plots\LRCN_standard.csv', 'a')

    # Create a file writer
    writer = csv.writer(res_csv)

    # List of features to add
    feat_list = ['feat_spec_all', 'feat_mfcc', 'feat_MFCC_delta',
'feat_MFCC_delta_delta', 'feat_lpc', 'feat_plp']

    # Train the LRCN with the current dataset
    LRCN_Build = LRCN_Model('standard', dataset_h5_path,
feature_list=feat_list)

    # Extract the model metrics
    accuracy, precision, recall, f1measure = LRCN_Build.train_load('lrcn',
'standard', load=False, step_size=25)

    # Print the model metrics
    print('%s\t %.3f\t%.3f\t%.3f\t%.3f\n' % (dataset_h5_path, accuracy,
precision, recall, f1measure))

    # Write the metrics in the CSV file
    writer.writerow([dataset_h5_path, accuracy, precision, recall,
f1measure])

    return 0
```

程式碼區段 3.12：訓練主函數

3.4.2.2 模型訓練

程式碼區段 3.13 主要目的為訓練模型和評估模型。

設定模型的輸入形狀和時間步長，StandardScaler() 和 scaler.transform() 為對訓練資料和驗證資料進行平均數與變異數標準化。get_step_data()從資料集根據時間步長設定訓練資料與驗證資料的形狀，train_X 和 valid_X 為訓練和驗證的聲音資料，train_Y 和 valid_Y 訓練和驗證的標記資料。若是先前選擇載入模型則讀取模型

和模型的權重，選擇不載入模型時會最小的 loss 設定 ModelCheckpoint 和 EarlyStopping，訓練後將結果畫出並儲存於 svg 檔。對驗證資料進行預測並將預測閥值設為 0.5，大於 0.5 的值改為 1，意思為有歌唱；小於 0.5 的值改為 0，意思為無歌唱。將驗證資料使用評估函式得出準確率、精確率、召回率、F1 值四項指標後回傳指標。

```
def train_load(self, case, use_model='lrcn', load=True, scale_normal=True, step_size=20):

    # Call LRCN Model
    model = LRCN(input_size=self.trainX.shape[1], time_steps=step_size)

    # Scale the values through standarization
    if scale_normal:
        scaler = StandardScaler()
        scaler.fit(self.validX)
        self.trainX = scaler.transform(self.trainX)
        self.validX = scaler.transform(self.validX)

    # Get step data in train and valid sets
    train_X, train_Y = get_step_data(self.trainX, self.trainY, step=step_size)
    valid_X, valid_Y = get_step_data(self.validX, self.validY, step=step_size)

    # If model is loaded
    if load:
        model = load_model('models/%s.model' % use_model)
        model.load_weights('models/%s.weights' % use_model)
    else:
    # Load checkpoints of model during training
        checkpoint = ModelCheckpoint('models/weights_best.{epoch:02d}-{loss:.2f}.hdf5', monitor='loss', verbose=1, save_best_only=True, mode='min')

        # Stop the training when the model stops improving
        early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=30, verbose=0, mode='auto', ) # baseline=None, restore_best_weights=True)

        # Fit training data into model history
        history = model.fit(train_X, train_Y,
        epochs=10000, verbose=VERBOSE, # 10000
        shuffle=True,
        validation_data=(valid_X, valid_Y),
        callbacks=[checkpoint, early_stop, TensorBoard(log_dir='logs'), ])

        # Plot Model Accuracy
        plt.figure()
        plt.plot()
        plt.plot(history.history['val_accuracy'])
        plt.title('model accuracy')
        plt.ylabel('accuracy')
```

```
41.             plt.xlabel('epoch')
42.             plt.legend(['train', 'test'], loc='upper left')
43.             plt.savefig('plots/%s_accuracy_%s.svg' % (use_model, case),
    format='svg')
44.
45.             # Plot Model Loss
46.             plt.figure()
47.             plt.plot(history.history['loss'])
48.             plt.plot(history.history['val_loss'])
49.             plt.title('model loss')
50.             plt.ylabel('loss')
51.             plt.xlabel('epoch')
52.             plt.legend(['train', 'test'], loc='upper left')
53.             plt.savefig('plots/%s_loss_%s.svg' % (use_model, case),
    format='svg')
54.
55.             # Save model and weights
56.             model.save('models/%s.model' % use_model)
57.             model.save_weights('models/%s.weights' % use_model)
58.
59.         # Predict the class from Valid set
60.         predict_Y = model.predict(valid_X)
61.         predict_Y = numpy.where(predict_Y > 0.5, 1, 0)
62.
63.         # Call Evaluator instance
64.         evaluator = Evaluator(predict_Y, valid_Y)
65.
66.         # Get metrics from predicting valid set
67.         accuracy, precision, recall, f1measure = evaluator.evaluate()
68.
69.         # Return metrics
70.         return accuracy, precision, recall, f1measure
```

程式碼區段 3.13：模型訓練

3.5 實驗結果

3.5.1 模型訓練結果

3.5.1.1 語料庫

我們使用二個有標記歌唱和非歌唱部分的語料庫進行模型訓練，分別是 Jamendo Corpus dataset [20] 和 Electrobyte dataset [21]。

Jamendo Corpus dataset 為公開資料集，包含了 93 首無版權的流行歌曲。每首歌曲以 44.1 kHZ 立體聲.ogg 或.mp3 格式進行編碼，並且提供歌唱和非歌唱部分的聲音標記檔。該資料集分為 61 首歌曲的訓練集、16 首歌曲的驗證集和 16 首歌曲的測試集。語料庫總長度為 443 分鐘。

Electrobyte dataset 為針對歌聲偵測所建立的無版權電子音樂數據集，包括 90 首電子音樂，所有歌曲均以.mp3 格式存儲對於每首歌曲，都進行了人工標記歌唱和非歌唱部分，並存儲在.lab 文件中，數據集分為 60 首訓練歌曲，15 首驗證歌曲和 15 首測試歌曲，每個集合均為隨機選擇。

3.5.1.2 網路架構

訓練參數為表 3.1：

參數名稱	參數值
Batch size	32
Drop-out rate	0.2
Learning rate	1e-4
Number of epochs	10,000

Early stopping	true
Sampling rate	16,000 Hz
time_steps	25
feat_list	['feat_spec_all', 'feat_mfcc', 'feat_MFCC_delta', 'feat_MFCC_delta_delta', 'feat_lpc', 'feat_plp']

表 3.1：歌聲偵測訓練參數

我們使用 LRCN 網路模型進行訓練，訓練的模型參數如表 3.2：

Model: "sequential"

Layer (type)	Output Shape	Param #
reshape (Reshape)	(None, 1, 1, 25, 276)	0
conv_lstm2d (ConvLSTM2D)	(None, 1, 7, 13)	15080
dropout (Dropout)	(None, 1, 7, 13)	0
max_pooling2d (MaxPooling2D)	(None, 1, 3, 13)	0
dropout_1 (Dropout)	(None, 1, 3, 13)	0
flatten (Flatten)	(None, 39)	0
dense (Dense)	(None, 200)	8000
dropout_2 (Dropout)	(None, 200)	0
dense_1 (Dense)	(None, 50)	10050
dropout_3 (Dropout)	(None, 50)	0
dense_2 (Dense)	(None, 1)	51

Total params: 33,181
Trainable params: 33,181
Non-trainable params: 0

表 3.2：LRCN 網路模型

3.5.1.3 結果

訓練的結果如下：

Model	accuracy	precision	recall	f1measure
Jamendo vocal	0.9350	0.9251	0.9499	0.9373
Jamendo normal	0.7314	0.7182	0.7821	0.7488
Electrobyte vocal	0.8402	0.8007	0.8685	0.8333
Electrobyte normal	0.8331	0.7957	0.8573	0.8253

表 3.3：歌聲偵測訓練結果

我們將二個語料集和使用原始音樂或分離的人聲四項分開統計，結果發現語料集有事先經過人聲分離的辨識結果皆優於未進行人聲分離，原因是因為雖然人聲分離不能完全的將人聲與背景音樂分離，仍然會留下些許雜音干擾，但還是可以去除掉大部分的非人聲資訊，使得在特徵提取更能提取出人聲的資訊。而在二個語料集互相比較辨識結果的情況下，發現 Jamendo 語料集較優於 Electrobyte 語料集，我們猜測原因可能是因為電子音樂的背景音樂較容易影響人聲分離，造成 Electrobyte 語料集的辨識結果在有人聲分離和沒有人聲分離的差異不像 Jamendo 語料集一樣明顯。

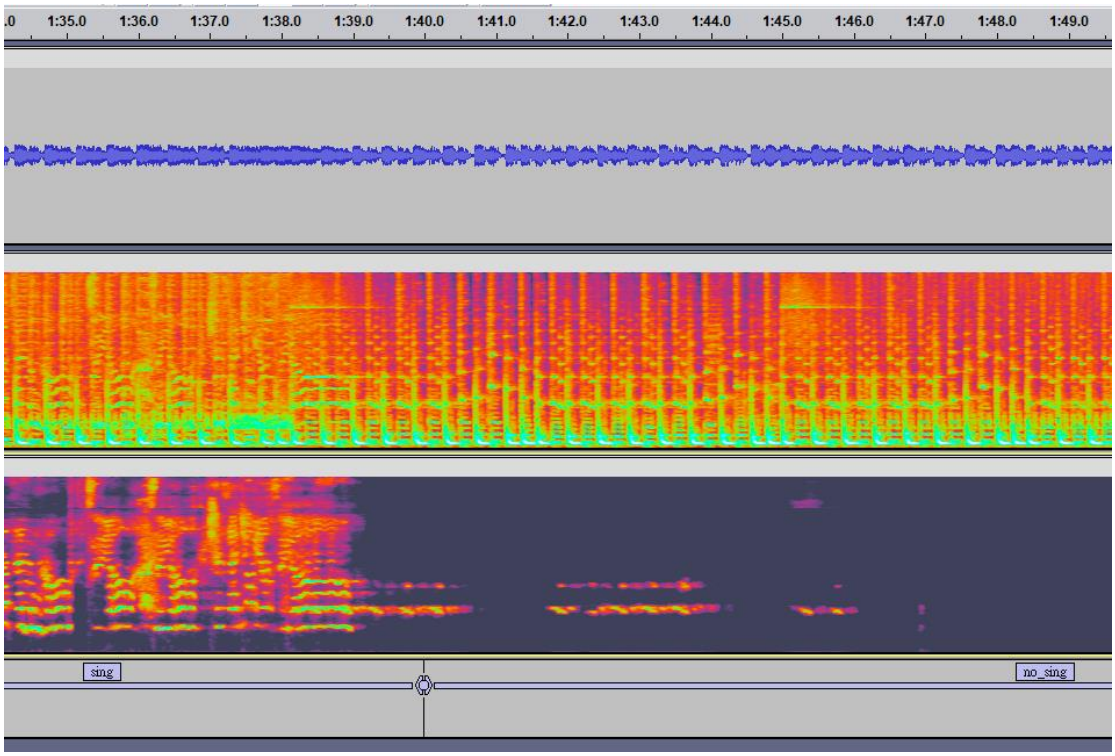


圖 3.2 鄭秀文-眉飛色舞歌聲分離與歌聲偵測結果

圖 3.2 為對鄭秀文的「眉飛色舞」進行歌聲分離與歌聲偵測結果的片段。圖片最上層的數字為歌曲的時間點，第一音軌的藍色線為原本歌曲的波形，第二音軌的頻譜圖為原本歌曲的頻譜圖，第三音軌的頻譜圖為歌聲分離後的頻譜圖，最下面的標記部分為歌聲偵測的結果。我們可以看到在進行歌聲分離後，在 1 分 39 秒後的分離音軌還是會有殘響，而歌聲偵測能夠分辨出歌手歌聲與非歌聲的部分。

3.5.2 卡拉 OK 系統

3.5.2.1 程式流程

複製欲進行演唱的音樂之 youtube 連結，貼上至 textbox 中後，選擇是否要有人聲，或選擇本地檔案中的 mp4,wav 檔。選擇後等待

程式進行下載音樂，人聲分離，分析主旋律後，進入卡拉 OK 畫面，在音樂結束或關閉卡拉 OK 視窗後結束主程式。

如果是要從 youtube 選取音樂，首先將 youtube 的網址輸入至欄內，選擇是否播放背景影片後，按下 onvocal 或 offvocal 按鈕。Onvocal 會在音樂播放時保留人聲的部分，而 offvocal 則是會去除人聲。若是從電腦選取音樂，則是直接點選本地檔案按鈕，選擇 mp4 或是 wav 檔案。使用者在選擇從 YouTube 選取音樂的時候，程式會先從 YouTube 上下載 MP4 檔，使用 ffmpeg 轉換成 wav 檔後，利用 spleeter 分離出背景音樂和人聲的部分，人聲的部分會先經過歌聲偵測尋找出歌手唱歌的部分，再使用自我相關函數計算出歌手唱歌的音高，此計算的音高會使用在卡拉 OK 的畫面上。卡拉 OK 的背景會播放 YouTube 上下載的影片，而螢幕上的黃色圓圈則是剛剛計算出歌手唱歌的音高，白色圓圈是麥克風接收到的聲音即時計算出來的音高，左邊的藍色方塊內則會顯示歌手唱的音名和頻率與麥克風接收到的音名和頻率。

圖 3.3：程式流程圖

3.5.1.2 選歌畫面

使用 PyQt 繪製選歌畫面視窗，在此視窗上的物件有：標題標籤、錯誤訊息標籤、onvocal 按鈕、offvocal 按鈕、本地檔案按鈕、背景播放影片核取方塊。標題標籤顯示文字為"請輸入想下載的音樂

連結"，錯誤訊息標籤顯示時機為 python 執行發生錯誤時會顯示錯誤資訊，onvocal 按鈕與 offvocal 按鈕為從 YouTube 下載音樂後，在卡拉 OK 主程式是否要去掉歌手歌聲的選項按鈕，本地檔案按鈕為選擇本地檔案中的 mp4 影片檔或是使用此程式產生的三音軌的 wav 檔，背景播放影片核取方塊為是否要在按下 onvocal 按鈕與 offvocal 按鈕後，卡拉 OK 主程式是否要播放影片。

圖 3.4：選歌畫面

3.5.1.3 錄音

在主程式外開啟新的執行緒(thread)並進行錄音，使用 pyaudio 進行錄音，目的是為了讀取麥克風的輸入並儲存於陣列 array 供卡拉 OK 畫面顯示與歌手所唱的主旋律的差異。使用 muti-thread 的主要原因是因為需要隨時的讀取輸入且不能影響到主畫面的執行。

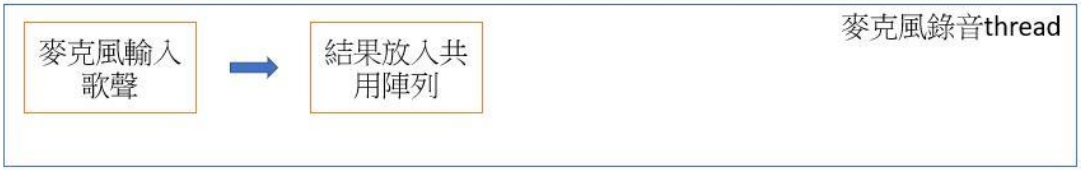


圖 3.5：麥克風錄音執行緒

3.5.1.4 卡拉 OK 畫面

卡拉 ok 畫面是由 pygame 所繪製而成，畫面上會隨著歌手和使用者唱出的音高而有高底起伏的圓圈，左下角會以頻率和音名顯示歌手所唱的主旋律的音高和使用者唱出的音高與兩者相同的次數，右下角會顯示目前歌唱時長和總時長，程式背景會顯示選擇的 MP4 影片。

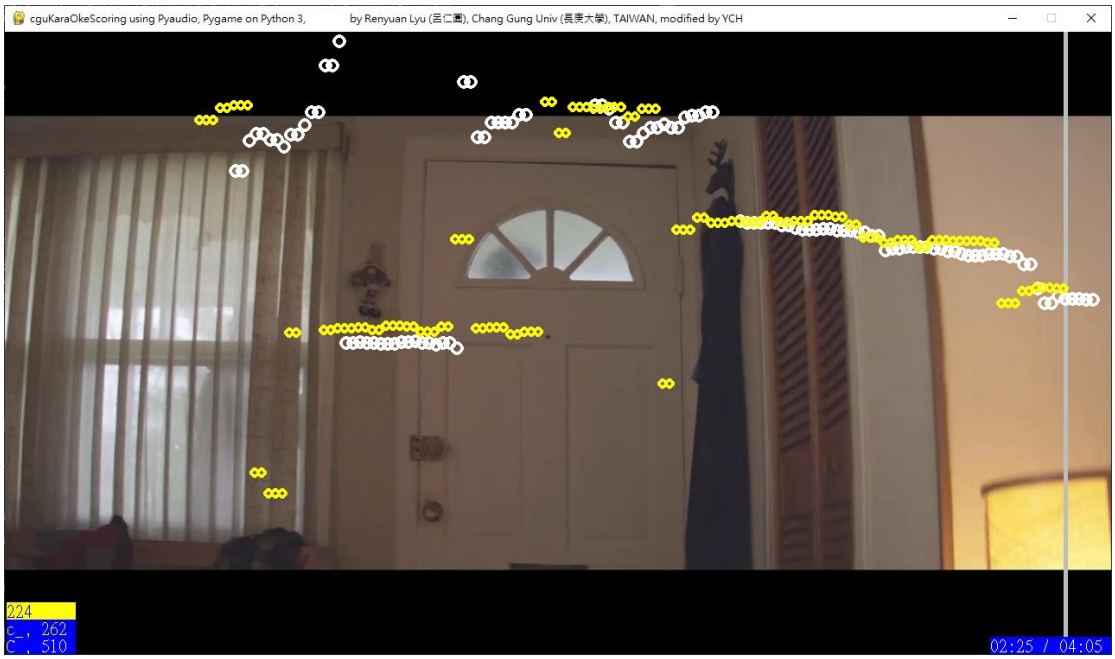


圖 3.6：卡拉 OK 畫面

3.5.1.5 背景顯示影片

背景影片是由 opencv 讀取後，再由 pygame 的 image 模組處理。這裡有二個問題：一是因為讀取影片需要花費較多的時間，主程式不會等待讀取影片才繼續執行，而且音樂和影片的播放是由不同的函數去控制的，會造成兩者不同步的情況，這裡的處理方式是先讀取好音樂和影片後再同時播放。二是因為主旋律分析的結果是放在 array 裡，隨著卡拉 OK 畫面開啟的時間去顯示，所以這裡的解決方法是使用 pygame 的 time 模組中的 get_ticks()抓取音樂和影片開始的時間，等到要顯示主旋律分析的結果時，再用 get_ticks()抓取當前時間，與影片開始的時間相減後獲得應該要顯示的 array 位置。

3.5.1.6 主旋律分析

在選擇音樂之後，首先會經過 spleeter 分離出背景音樂和人聲的部分，人聲的部分會再使用自己訓練的歌聲偵測分離出人聲和雜音的部分，最後再經過自我相關(autocorrelation)函數去分析人聲的頻率。分析完人聲的頻率後，會將整個結果存於 array 內等稍後主程式使用。

3.6 改進

3.6.1 歌詞顯示

目前系統並沒有顯示歌詞於畫面上，主要原因是因為在 YouTube 上不是每首歌都有歌詞內嵌於字幕中，所以沒有實作抓取字幕並顯示於畫面上的功能。

有兩個方向可以實作歌詞顯示的功能：抓取網路資料和對歌詞

進行語音辨識。抓取網路資料可以是從多個歌詞網站上使用網路爬蟲或是 API 的方式抓取歌詞，優點是這些個詞通常已經過人工處理，只需要進行畫面輸出即可，缺點是可能找不到歌詞造成程式執行上的問題。對歌詞進行語音辨識可以嘗試使用如 Whisper 等語音辨識系統，在處理音樂時同步進行語音辨識，好處是所有音樂都能有歌詞輸出，但有可能遇到辨識錯誤的問題。

若是要像一般卡拉 OK 一樣每句顯示歌詞還需要面對歌詞和音樂時間點對齊的部分，有可能還需要處理歌詞斷句的問題，這些都是一大挑戰。

3.6.2 歌唱表現力評分

在唱歌的時候，除了音高要準確以外，我們希望唱歌的人的咬字清晰，歌唱表現如顫音、轉音等，表現出更多的歌唱技巧，而不單單只是發出聲音。

咬字的部分我們可以嘗試即時對使用者和原歌手的聲音輸入進行語音辨識，若是能辨識出相同的字，即代表咬字與原歌手的表現程度相同。而歌唱表現則是能嘗試使用逐音框進行辨識，找出有使用表現技巧的部分。使得此卡拉 ok 系統能進行使用者歌唱表現的評分。

第四章 Whisper 台語語音辨識

4.1 動機

根據台灣 109 年人口及住宅普查，台灣人有 6,897,535 人使用台語為主要使用語言，約占總人口的 31.7%；甚至有 18,728,839 人會說台語，約占總人口的 86.0%。然而台語為非書寫語言，沒有正式的書寫方式，也鮮少有語音資料庫，在目前的語音辨識中難以進行建置。

台語的書寫方式目前多以漢字表示，少部分未收入漢語字典以台羅拼音表示。且部分台語音調與中文相近，亦使用相同的中文字表示。

在 2022 年 9 月 21 日，OpenAI 先前發表了 whisper ：一個使用了 680,000 小時的標記音訊，可對超過 90 種語言進行語音辨識的模型，我們實驗室在發現了 whisper 後，立即對此模型對生活中的語音資料進行辨識正確率的統計。

在 2022 年 10 月 20 日，在 Meta 發表了使用台語連續劇語料建立的閩南語對英文的 AI 翻譯系統後，我們也開始嘗試對 whisper 輸入我們收集的台語連續劇，並看 whisper 對台語的辨識效果。

而在 2022 年 12 月 huggingface 舉辦了 whisper Fine-Tuning Event [22] 並提供 Whisper 各個模型的訓練 checkpoints 讓所有人使用不同的語言微調模型，我們在此活動中嘗試對 whisper 使用台語進行微調，本章我們將展示我們微調的結果。

4.2 目的

本篇論文將對 whisper 的微調進行研究，我們使用 CommonVoice 的台語資料集和我們收集的台語連續劇對 whisper 進行微調，嘗試使用 whisper 對台語進行語音辨識，並輸出台語漢字或是華語文字。

比較結果的方式，因為中文是以每個字為輸出，所以我們將使用字元錯誤率 (Character error rate, CER)作為指標。

4.3 基本知識

4.3.1 whisper

whisper 為 openAI 所建立的弱監督預訓練語音辨識模型。使用了 680,000 小時的標記音訊數據，其中有 117,000 小時是英文以外的其他 96 種語言，還包括 125,000 小時的其他語言翻譯至英文的數據。在訓練 Whisper 模型時並無進行文本標準化，而是直接預測轉錄的原始文本，這依賴序列到序列模型的表達能力來學習在語句和其轉錄形式之間的映射關係，但是也簡化了語音識別流程，因為它不需要單獨進行文本反標準化步驟以生成自然的轉錄。whisper 使用音訊語言檢測器確認轉錄的語言，將音訊文件分割為 30 秒的片段後與轉錄子集配對並進行訓練，能檢測出時間片段的語言或沒有語音的片段。

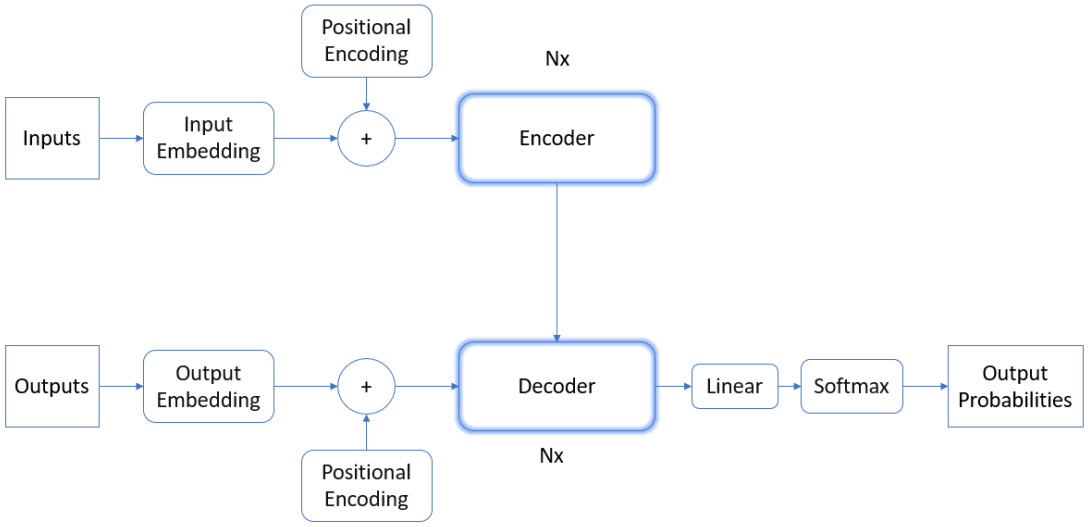
whisper 使用編碼器-解碼器 Transformer(Vaswani et al., 2017)架構，所有音訊都被重新取樣為 16,000 Hz，並且在 25 毫秒窗口上以 10 毫秒的步長計算出 80 通道的對數幅度 Mel 頻譜表示。其他模型

細節將在 4.3.2 節 transformer 部分說明 whisper 在純英文的模型使用了與 GPT-2 相同的字節級 BPE(byte-level BPE text tokenizer)文本分詞器，在多語言模型使用了相同大小的詞彙表避免在其他語言上出現過多的碎片。

whisper 使用 <|startoftranscript|> 作為預測的開始 token ，預測出正在說的語言如<|zh|> 或是沒有說話<|nospeech|> token，指定了任務 token <|transcribe|> 或 <|translate|>， 決定不產生時間標記的<|notimestamps|>，開始輸出辨識結果，最後使用<|endoftranscript|>結束辨識任務。

4.3.2 transformer

Transformer 是一種用於自然語言處理和機器翻譯等任務的神經網絡架構，由 Vaswani 等人於 2017 年提出。它在處理序列數據時不需要使用循環神經網絡(RNN)或卷積神經網絡(CNN)，而是通過自注意力機制實現了長距離依賴性的建模，Transformer 的訓練過程使用自監督學習的方法，使用遮罩語言模型(Masked Language Model)預測下一個詞彙的任務進行訓練，以下將根據 Transformer 的流程說明各個結構。



The Transformer - model architecture.

圖 4.1：Transformer 模型架構圖

4.3.3 輸入嵌入(Input Embedding)

輸入嵌入(Input Embedding)：將輸入序列中的每個詞彙轉換為其對應的低維嵌入向量的過程。這些嵌入向量將單詞的語義信息和特徵表示編碼為連續向量空間中的點。

假設有一個輸入序列為 $x = [x_1, x_2, ..., x_n]$，其中 x_i 表示第 i 個詞彙的索引(整數)。

輸入嵌入使用一個稱為嵌入矩陣(Embedding Matrix)的可學習參數矩陣 E，它的大小為 V × d，其中 V 是詞彙表的大小(詞彙的總數)，d 是嵌入維度。嵌入矩陣 E 的每一行表示一個詞彙的嵌入向量。輸入嵌入的計算公式如下：

$$\boldsymbol{E} = [\boldsymbol{e^1}, \boldsymbol{e^2}, ..., \boldsymbol{e_V}] \qquad \textbf{式4.1}$$

其中，e_i 表示第 i 個詞彙的嵌入向量。

則輸入序列 x 的嵌入表示為：

$$\hat{\boldsymbol{x}} = [\boldsymbol{e^1}, \boldsymbol{e^2}, \ldots, \boldsymbol{e}_{\mathrm{n}}] \qquad \textbf{式4.2}$$

其中，x̂ 表示輸入序列 x 的嵌入表示。

以上是輸入嵌入的數學公式表示，它將輸入序列中的每個詞彙轉換為對應的嵌入向量。這些嵌入向量可以作為神經網絡的輸入，進一步進行特徵提取和序列處理。

Transformer 將輸入的音訊轉換成 $\boldsymbol{d_{model}}$= 512 維度的向量。

4.3.4 位置編碼(Positional Encoding)

位置編碼(Positional Encoding)用於將序列中的詞彙位置信息嵌入到嵌入向量中，以便 Transformer 網絡能夠捕捉序列中的順序和位置信息。位置編碼通常使用正弦和餘弦函數來編碼詞彙的位置信息。假設輸入序列的嵌入表示為 $\hat{x} = [e_1, e_2, ..., e_n]$，其中 e_i 是第 i 個詞彙的嵌入向量。對於每個嵌入向量 e_i，我們計算它的位置編碼 p_i，並將其添加到對應的嵌入向量中。位置編碼的計算公式如下：

$$\mathbf{Pe}_{(\mathbf{pos},\,\mathbf{2i})} = \sin\left(\mathbf{pos} / \mathbf{10000}^{(\mathbf{2i/d})}\right) \qquad \textbf{式4.3}$$

$$\mathbf{Pe}_{(\boldsymbol{pos},\boldsymbol{2i+1})} = \cos\left(\boldsymbol{pos}/\mathbf{10000}^{(\boldsymbol{2i/d})}\right) \qquad \textbf{式4.4}$$

其中，$\mathbf{Pe}_{(\boldsymbol{pos},\boldsymbol{2i})}$ 表示位置編碼的第 i 維(i = 0, 1, ..., d-1)，POS 是詞彙的位置索引，d 是嵌入維度。

位置編碼的最終表示 $\mathbf{Pe}_{(\boldsymbol{pos},\boldsymbol{i})}$ 與對應的嵌入向量 e_i 相加，得到最終的輸入表示：

$$\hat{\boldsymbol{x}}_{\boldsymbol{i}} = \boldsymbol{e}_{\mathrm{i}} + \boldsymbol{Pe}_{\mathrm{i}} \qquad \textbf{式4.5}$$

這樣每個詞彙的嵌入向量就被位置編碼信息所擴展。

4.3.5 自注意力機制(Self-Attention)

自注意力機制(Self-Attention)：自注意力機制通過將輸入序列中的每個詞彙關聯到其他詞彙，從而獲得每個詞彙的上下文信息。通過計算詞彙間的相似度得分，獲得一組權重，這些權重指示著每個詞彙應該關注哪些詞彙。這使得 Transformer 能夠捕捉長距離依賴性，並且在並行計算方面效率更高。

假設輸入序列 $X = [x_1, x_2, ..., xn]$。

首先計算注意力權重(Attention Weights)

，我們通過將序列 X 中的每個元素映射到三個不同的線性投影空間中，分別為 Query(Q)、Key(K)和 Value(V)

$$\mathbf{Q} = \mathbf{X}@\boldsymbol{W_Q} \qquad \textbf{式4.6}$$

$$\mathbf{K} = \mathbf{X}@\boldsymbol{W_K} \qquad \textbf{式4.7}$$

$$\mathbf{V} = \mathbf{X}@\boldsymbol{W_V} \qquad \textbf{式4.8}$$

得到對應的特徵表示 $Q = [q_1, q_2, ..., qn]$，$K = [k_1, k_2, ..., kn]$，$V = [v_1, v_2, ..., vn]$。然後，計算自注意力機制的輸出特徵 Z，計算公式如下：

$$\mathbf{Z} = \mathbf{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathbf{softmax}\left(\boldsymbol{QK^T/sqrt(d_k)}\right)@\mathbf{V} \qquad \textbf{式4.9}$$

其中@ 為矩陣相乘，d_k表示特徵的維度總數：

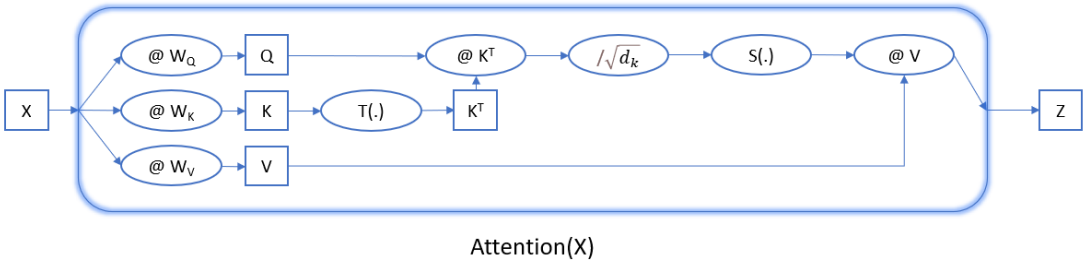
$$\boldsymbol{d_k} = \boldsymbol{d_{model}}/\mathbf{h} = \mathbf{64} \qquad \textbf{式4.10}$$

$\boldsymbol{d_{model}}$為輸入和輸出的維度總數，h 為平行的注意力層數，或稱為頭(head)。

多頭注意力機制(Multi-Head Attention) 將所有注意力層數的輸出特徵表示進行連接，得到最終輸出特徵表示。計算公式如下：

$$\boldsymbol{MultiHead(Q,K,V) = Concat(Z_1, Z_2, \dots, Z_h)@W^o = \sum_{i=1}^{h} Z_i @W^o} \quad \text{式 4.11}$$

其中 $\boldsymbol{W^o}$ 為多頭權重。



T(.) = transpose
S(.) = softmax
@ = matrix multiplication

圖 4.2：自注意力機制計算流程

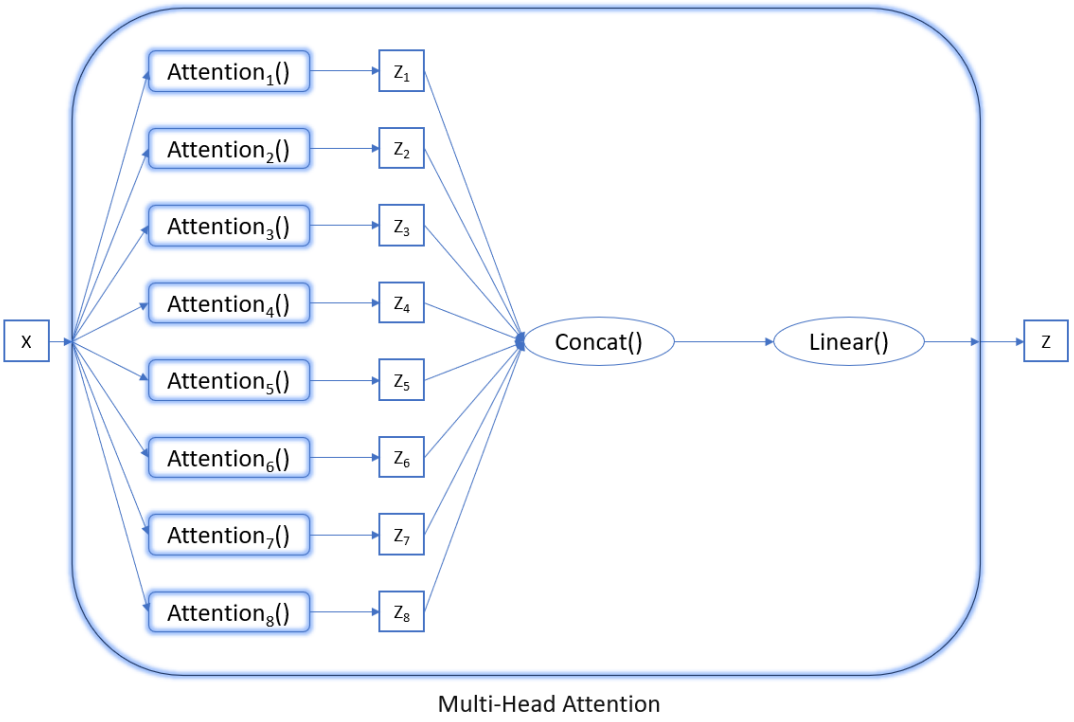


圖 4.3：多頭注意力機制流程

4.3.6 前饋神經網絡(Feed-Forward Neural Network, FFN)

在自注意力機制之後，編碼器還包含一個前饋神經網絡(FFN)，用於對自注意力機制的輸出特徵表示進行非線性轉換和特徵提取。FFN 由兩個線性層和一個非線性激活函數 ReLU 組成，，FFN 的計算公式如下：

$$\mathbf{FFN}(\boldsymbol{X})\mathbf{max} = \mathbf{0}(\boldsymbol{X}, \boldsymbol{W}\}_{\mathbf{1}} + \boldsymbol{b}_{\mathbf{1}}@\boldsymbol{W}_{\mathbf{2}} + \boldsymbol{b}_{\mathbf{2}} \qquad \text{式}\mathbf{4.12}$$

其中 W_1、W_2表示權重矩陣，b_1、b_2表示偏差向量。

4.3.7 編碼器與解碼器

編碼器由多個相同結構的編碼器堆疊而成，每個編碼器包含兩個子層：多頭自注意力機制和前饋神經網絡。編碼器的目的是將輸入序列編碼成一系列上下文關係的特徵表示。

解碼器與編碼器類似，解碼器由多個相同結構的解碼器堆疊而成。解碼器比編碼器多添加了一個遮罩多頭注意力機制，防止模型看見要預測的資料。這樣解碼器可以關注輸入序列中的不同部分並生成輸出序列。

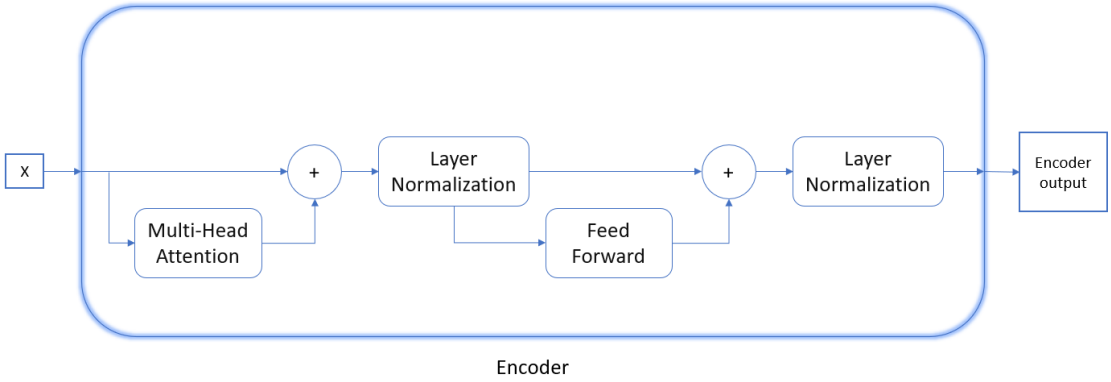


圖 4.4：編碼器架構

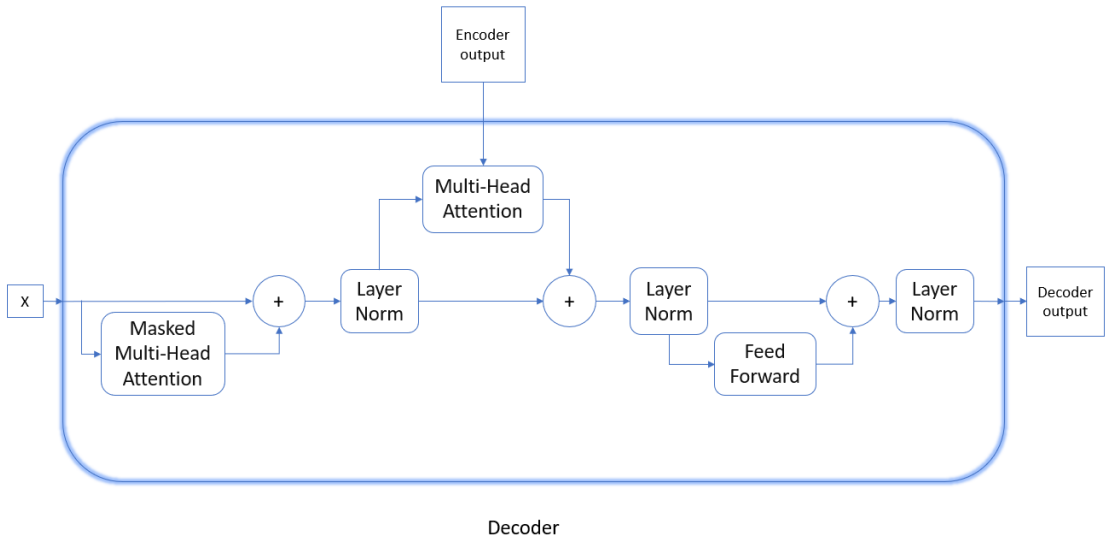


圖 4.5：解碼器架構

4.3.8 字元錯誤率

字元錯誤率(Character Error Rate，CER)是用於評估自動語音辨識(automatic speech recognition, ASR)系統準確性的指標。它衡量系統輸出與參考文本或標準答案之間的字元錯誤百分比。

CER 的計算公式為：

$$\mathbf{CER} = (\boldsymbol{S} + \boldsymbol{D} + \boldsymbol{I})/\mathbf{N} = (\boldsymbol{S} + \boldsymbol{D} + \boldsymbol{I})/(\boldsymbol{S} + \boldsymbol{D} + \boldsymbol{C}) \qquad \textbf{式4.13}$$

其中，S 代表替換的字元數量、D 代表刪除的字元數量、I 代表插入的字元數量、C 代表正確的字元數量，而 N 代表參考文本中的字元總數。

CER 的結果並不一定介於 0 和 1 之間，特別是在插入的字元數量較高時。此值通常與被錯誤預測的字元百分比相關聯。數值越低，ASR 系統的性能越好，而 CER 為 0 表示完美的分數。

4.3.9 台語漢字

本篇論文所使用的台語漢字為根據教育部於 2009 年公布「臺灣

閩南語推薦用字」[39] 所訂定的用字，此推薦字共有 700 筆資料，每筆資料皆有建議用字、音讀、對應華語、用例等。本章的台語輸出漢字輸出部分皆按照此推薦用字進行辨識輸出。

建議用字	音讀	對應華語	用例
會當	ē-tàng	可以、能夠	會當去
風颱	hong-thai	颱風	做風颱、風颱天
代誌	tāi-tsì	事、事情	無代誌、好代誌

表 4.1 臺灣閩南語推薦用字 700 字表簡例

4.4 實驗程式

本章將說明 whisper 從載入音訊到輸出結果的程式碼，在參數部分我們將使用 large-v2 的模型參數進行帶入，large-v2 與 large 在架構參數上大部分相同：32 層(Layers)的編碼器與解碼器，每層的輸出維度(Width)為 1280，注意力頭數(Heads)為 20 層。

4.4.1 載入音訊

程式碼區段 4.1 主要目的為將資料集的音訊波形轉換成對數梅爾頻譜圖。

首先檢查波形輸入是否已經是一個 PyTorch 張量如果它不是張量，使用 load_audio 函數載入音訊數據。然後將音頻波形轉換為 PyTorch 張量，如果它尚未是一個張量的話，使用 torch.from_numpy 進行轉換。

頻譜圖計算部分使用 torch.hann_window 創建一個大小為 N_FFT = 400 的漢明窗。

使用 torch.stft 在音頻波形上計算 STFT，使用參數 N_FFT=400 、HOP_LENGTH = 160 和漢明窗。STFT 返回形狀為(201, celi(音訊資料長度/160))。通過取絕對值然後平方來計算 STFT 系數的幅度。梅爾頻譜圖計算部分使用 mel_filters 函數計算梅爾濾波器，傳遞音頻裝置和所需的梅爾頻帶數量 n_mels = 80。通過將梅爾濾波器乘以 STFT 系數的幅度獲得梅爾頻譜圖。

對數縮放部分將梅爾頻譜圖的值夾縮到最小值 1e-10，然後使用 log10 進行對數縮放。進一步調整對數梅爾頻譜圖的值，使其最大值為 8.0，通過將值剪切到最大值減去 8.0。然後通過添加 4.0 並除以 4.0 來對調整後的頻譜圖值進行縮放，結果的值範圍為 0 到 1。最後以 NumPy 陣列輸出得到的對數梅爾頻譜圖。

```
def log_mel_spectrogram(self, waveform: np.array) -> np.ndarray:
    if not torch.is_tensor(audio):
        if isinstance(audio, str):
            audio = load_audio(audio)
        audio = torch.from_numpy(audio)

    window = torch.hann_window(N_FFT).to(audio.device)
    stft = torch.stft(audio, N_FFT, HOP_LENGTH, window=window, return_complex=True)
    magnitudes = stft[:, :-1].abs() ** 2

    filters = mel_filters(audio.device, n_mels)
    mel_spec = filters @ magnitudes

    log_spec = torch.clamp(mel_spec, min=1e-10).log10()
    log_spec = torch.maximum(log_spec, log_spec.max() - 8.0)
    log_spec = (log_spec + 4.0) / 4.0
    return log_spec
```

程式碼區段 4.1：音訊波形轉換成對數梅爾頻譜圖

4.4.2 qkv 注意力計算

程式碼區段 4.2 主要目的為計算 qkv 注意力。

輸入形狀為(n_batch, n_ctx, n_state) = (1, 1500, 1280) 的查詢 (query) 張量 q，鍵 (key) 張量 k，值 (value) 張量 v。遮罩(mask)在解碼器的訓練中用於遮罩句子中欲預測和後續部分，形狀為(n_ctx, n_ctx) = (1500, 1500)

重塑和縮放部分，將查詢 q、鍵 k 和值 v 張量重塑為形狀為 (n_batch, n_head, n_ctx, n_state//n_head) = (1, 20, 1500, 1280//20) = (1, 20, 1500, 64)的張量，通過將最後一維分為 n_head = 20 個較小的維度。計算縮放因子 scale = (n_state // n_head) ** -0.25 = (1280 // 20) ** -0.25 ~= 0.3536 後將查詢 q 和鍵 k 張量乘以縮放因子 scale。

注意力計算部分，計算 q 和 k 張量之間的點積，得到 qk = q @ k。如果提供了遮罩，則將其添加到 qk 張量中。將 qk 張量通過 softmax 函數沿著最後一個維度計算注意權重 w，使用注意權重 w 對值 v 張量進行加權求和，為每個注意力頭計算加權值，最後，將輸出重塑並展平為形狀為 (1, 1500, 1280) 的張量。

```
1.  def qkv_attention(self, q: Tensor, k: Tensor, v: Tensor, mask:
    Optional[Tensor] = None):
2.      n_batch, n_ctx, n_state = q.shape
3.      scale = (n_state // self.n_head) ** -0.25
4.      q = q.view(*q.shape[:2], self.n_head, -1).permute(0, 2, 1, 3) * scale
5.      k = k.view(*k.shape[:2], self.n_head, -1).permute(0, 2, 3, 1) * scale
6.      v = v.view(*v.shape[:2], self.n_head, -1).permute(0, 2, 1, 3)
7.
8.      qk = q @ k
9.      if mask is not None:
10.         qk = qk + mask[:n_ctx, :n_ctx]
11.
12.     w = F.softmax(qk.float(), dim=-1).to(q.dtype)
13.     return (w @ v).permute(0, 2, 1, 3).flatten(start_dim=2)
```

程式碼區段 4.2： qkv 注意力計算

4.4.3 多頭注意力

程式碼區段 4.3 主要目的為呈現多頭注意力層的部分，用於捕捉輸入元素之間基於注意力的交互作用。

類別(Class)初始化部分，多頭注意力接收輸入維度 n_state = width = 1280 和注意力頭數 n_head = 20，並宣告四個線性層 (nn.Linear) 對稍後的查詢 q，鍵 k，值 v 和多頭注意力輸出 out 進行線性轉換。

函式計算部分輸入參數為輸入張量 x、用於交叉注意力的編碼器輸入張量 xa、用於遮罩的遮罩張量 mask 和用於沿用交叉注意力 (cross attention)的鍵 k 和值 v 的暫存字典 kv_cache。將輸入 x 通過查詢線性層，獲得查詢張量 q。如果 kv_cache 為 None 或 xa 為 None，代表從同為編碼器或解碼器的自身注意力(self attention)計算，通過將 x 分別通過鍵和值線性層，計算鍵 k 和值 v 張量；如果 kv_cache 不為 None 且 xa 不為 None，代表解碼器從編碼器取得鍵 k 和值 v 的交叉注意力計算，則從 kv_cache 中檢索已暫存的鍵 k 和值 v 張量。將 q、k 和 v 張量傳遞給 qkv_attention ，以計算注意權重並執行注意力操作。最後輸出通過 out 線性層獲得輸出張量。

```
class MultiHeadAttention(nn.Module):
    def __init__(self, n_state: int, n_head: int):
        super().__init__()
        self.n_head = n_head
        self.query = Linear(n_state, n_state)
        self.key = Linear(n_state, n_state, bias=False)
        self.value = Linear(n_state, n_state)
        self.out = Linear(n_state, n_state)

    def forward(
        self,
        x: Tensor,
        xa: Optional[Tensor] = None,
        mask: Optional[Tensor] = None,
        kv_cache: Optional[dict] = None,
        ):
        q = self.query(x)

        if kv_cache is None or xa is None or self.key not in kv_cache:
        k = self.key(x if xa is None else xa)
        v = self.value(x if xa is None else xa)
        else:
            k = kv_cache[self.key]
            v = kv_cache[self.value]

        wv = self.qkv_attention(q, k, v, mask)
        return self.out(wv)
```

程式碼區段 4.3：多頭注意力

4.4.4 注意力區塊

程式碼區段 4.4 主要目的為建立編碼器與解碼器內每層注意力區塊結構，對輸入數據進行分層處理。

類別初始化部分，注意力區塊接收維度 n_state = width = 1280，注意力頭數 n_head = 20 和是否使用交叉注意力的布林值(bool) cross_attention。每層注意力區塊包含一個多頭注意力層 attn、 多頭

注意力層輸出正規化 attn_ln、 多層感知器(Multilayer Perceptron) MLP、多層感知器層輸出正規化 mlp_ln。若是此注意力區塊為解碼器所用，則會多加交叉注意力層 cross_attn 和交叉注意力層輸出正規化 cross_attn_ln。

函式計算部分輸入參數為輸入張量 x、用於交叉注意力的編碼器輸入張量 xa、用於遮罩的遮罩張量 mask 和用於沿用交叉注意力(cross attention)的鍵 k 和值 v 的暫存字典 kv_cache。將輸入 x 通過 attn 與 attn_ln 後，並將結果與輸入 x 進行殘差連接(skip connect)。如果 cross_attention 為 True，則將前者 x 的輸出通過 cross_attn 與 cross_attn_ln 後，並將結果與輸入 x 進行殘差連接。最後將 attn_ln 或 cross_attn_ln 的輸出通過 MLP ，並將結果與 MLP 的輸入進行殘差連接後，回傳最終的輸出。

```
class ResidualAttentionBlock(nn.Module):
    def __init__(self, n_state: int, n_head: int, cross_attention: bool = False):
        super().__init__()

        self.attn = MultiHeadAttention(n_state, n_head)
        self.attn_ln = LayerNorm(n_state)

        self.cross_attn = MultiHeadAttention(n_state, n_head) if
cross_attention else None
        self.cross_attn_ln = LayerNorm(n_state) if cross_attention else
None

        n_mlp = n_state * 4
        self.mlp = nn.Sequential(Linear(n_state, n_mlp), nn.GELU(),
Linear(n_mlp, n_state))
        self.mlp_ln = LayerNorm(n_state)

    def forward(
        self,
        x: Tensor,
        xa: Optional[Tensor] = None,
        mask: Optional[Tensor] = None,
        kv_cache: Optional[dict] = None,
    ):
        x = x + self.attn(self.attn_ln(x), mask=mask, kv_cache=kv_cache)
        if self.cross_attn:
            x = x + self.cross_attn(self.cross_attn_ln(x), xa,
kv_cache=kv_cache)
        x = x + self.mlp(self.mlp_ln(x))
        return x
```

程式碼區段 4.4：注意力區塊

4.4.5 音訊編碼器

程式碼區段 4.5 主要目的為對輸入的音訊進行編碼和建立 whisper 編碼器部分，使模型能夠捕捉音訊數據中的相關特徵和依賴關係，以供解碼器使用。

類別初始化部分，音訊編碼器接收梅爾濾波器數量 n_mels =

80 、輸入音訊序列的長度 n_ctx = 1500、編碼器輸出維度大小 n_state = 1280、 注意力頭的數量 n_head = 20 和編碼器的層數 n_layer = 32。音訊編碼器使用二層的卷積層 conv1 和 conv2 處理輸入的對數梅爾頻譜圖，位置編碼緩存區 positional_embedding 用於存儲位置嵌入的值 ，多個注意力區塊堆疊的編碼器 blocks 和對編碼器輸出正規化 ln_post。

函式計算部分輸入參數為輸入對數梅爾頻譜圖張量 x、將輸入 x 通過二層卷積層 conv1 和 conv2 並。在每個卷積之後使用 GELU 激勵函數(Activation Function)。GELU 的公式為

$$\mathbf{GELU}(\boldsymbol{x}) = \mathbf{0.5x}\left(\mathbf{1} + \boldsymbol{tanh}\left(\sqrt{\mathbf{2/\pi}}(\boldsymbol{x} + \mathbf{0.044715}\boldsymbol{x}^{\mathbf{3}})\right)\right) \quad \text{式}\mathbf{4.14}$$

將 x 重新排列為(batch_size，n_ctx，n_state)的尺寸，以匹配位置嵌入的形狀。使用矩陣相加方式將位置嵌入添加到 x 中。然後將 x 輸入至編碼器的多個注意力區塊中進行多頭注意力計算，最後將最後一個區塊的輸出通過 ln_post 並回傳作為編碼的音訊的結果張量。

```python
class AudioEncoder(nn.Module):
    def __init__(self, n_mels: int, n_ctx: int, n_state: int, n_head: int, n_layer: int):
        super().__init__()
        self.conv1 = Conv1d(n_mels, n_state, kernel_size=3, padding=1)
        self.conv2 = Conv1d(n_state, n_state, kernel_size=3, stride=2, padding=1)
        self.register_buffer("positional_embedding", sinusoids(n_ctx, n_state))

        self.blocks: Iterable[ResidualAttentionBlock] = nn.ModuleList(
            [ResidualAttentionBlock(n_state, n_head) for _ in range(n_layer)]
        )
        self.ln_post = LayerNorm(n_state)
    def forward(self, x: Tensor):
    """
    x : torch.Tensor, shape = (batch_size, n_mels, n_ctx)
    the mel spectrogram of the audio
    """
        x = F.gelu(self.conv1(x))
        x = F.gelu(self.conv2(x))
        x = x.permute(0, 2, 1)

        assert x.shape[1:] == self.positional_embedding.shape, "incorrect audio shape"
        x = (x + self.positional_embedding).to(x.dtype)

        for block in self.blocks:
            x = block(x)

        x = self.ln_post(x)
        return x
```

程式碼區段 4.5：音訊編碼器

4.4.6 文本解碼器

程式碼區段 4.6 主要目的為對輸入的文本進行編碼和建立 whisper 解碼器部分。輸入的文本將與音訊特徵進行交叉注意力，為文本序列中的下一個標記生成預測。

類別初始化部分，文本解碼器接收詞彙表大小 n_vocab = 51865、文本序列的最大長度 n_ctx = 448、解碼器輸出維度大小 n_state = 1280、注意力頭的數量 n_head = 20 和編碼器的層數 n_layer = 32。文本解碼器使用一個嵌入層(embedding)將文本標記映射為大小為 n_state = 1280 的連續向量 token_embedding，文本序列的位置嵌入學習參數 positional_embedding，多個注意力區塊堆疊的解碼器 blocks，對解碼器輸出正規化 ln 和用於防止關注文本序列中的未來位置的注意力遮罩 mask。

函式計算部分輸入參數為形狀是(1, 1, 1280)的文本標記張量 x ，形狀為(1, 1500, 1280)的編碼音訊特徵張量 xa，儲存編碼器鍵 k 和值 v 的暫存字典 kv_cache。首先計算偏移，根據 kv_cache 計算偏移變數。如果緩存不為 None，則根據值的形狀並通過取列數來確定偏移量。否則，將偏移設置為 0。將文本標記 x 通過嵌入層 token_embedding，將每個標記映射為連續向量表示。從 positional_embedding 參數中提取位置編碼，使用偏移量選擇適當的位置編碼以供應於標記並將標記嵌入和位置編碼相加。將偏移量的計算結果使用 x.to(xa.dtype)將獲得的嵌入轉換為與音頻特徵 xa 相同的數據類型後，x、 xa、 mask 和 kv_cache 輸入到解碼器的注意力區塊進行多頭注意力計算。回傳的值在正規化 ln 後與原本的輸入 x 與標記嵌入權重的轉置相乘，以獲得下一個標記預測的 logits 並將 logits 作為輸出回傳。

```python
class TextDecoder(nn.Module):
    def __init__(self, n_vocab: int, n_ctx: int, n_state: int, n_head: int, n_layer: int):
        super().__init__()

        self.token_embedding = nn.Embedding(n_vocab, n_state)
        self.positional_embedding = nn.Parameter(torch.empty(n_ctx, n_state))

        self.blocks: Iterable[ResidualAttentionBlock] = nn.ModuleList(
            [ResidualAttentionBlock(n_state, n_head, cross_attention=True) for _ in range(n_layer)]
            )
        self.ln = LayerNorm(n_state)

        mask = torch.empty(n_ctx, n_ctx).fill_(-np.inf).triu_(1)
        self.register_buffer("mask", mask, persistent=False)

    def forward(self, x: Tensor, xa: Tensor, kv_cache: Optional[dict] = None):
        """
        x : torch.LongTensor, shape = (batch_size, <= n_ctx)
        the text tokens
        xa : torch.Tensor, shape = (batch_size, n_mels, n_audio_ctx)
        the encoded audio features to be attended on
        """
        offset = next(iter(kv_cache.values())).shape[1] if kv_cache else 0
        x = self.token_embedding(x) + self.positional_embedding[offset : offset + x.shape[-1]]
        x = x.to(xa.dtype)

        for block in self.blocks:
        x = block(x, xa, mask=self.mask, kv_cache=kv_cache)

        x = self.ln(x)
        logits = (x @ torch.transpose(self.token_embedding.weight.to(x.dtype), 0, 1)).float()

        return logits
```

程式碼區段 4.6：文本解碼器

4.5 實驗結果

4.5.1 使用模型與資料集

在原本 whisper 論文不同大小的模型對中文的辨識結果中，medium 和 large 的結果較優於其他三項。而台語的語法結構和中文較相近，所以在本章實驗中，我們主要對 Hugging face 所提供的 whisper 的 medium 和 large-v2 模型進行研究，對這兩個模型進行台語語音的微調，嘗試使 whisper 能對台語進行語音辨識。

在進行台語語音的微調前，我們先對 Whisper 論文中提出的中文辨識結果進行比較，我們會先對 Hugging face 的 medium 和 large-v2 模型使用 Common Voice 的繁體中文的語料庫進行微調前後的比較，一方面確認 Hugging face 提供的模型與 Whisper 論文的數據是否相近，一方面測試我們進行微調繁體中文是否真的能提高辨識結果。

台語語音的微調部分，我們使用 Common Voice 的台語資料集和我們收集的台語連續劇進行模型微調與測試結果。Common Voice 資料集是用於語音技術研究和開發的大量多語言轉錄語音集合。資料集中包含 27,142 小時錄製完成的片段，其中包含 17,690 小時 108 種語言的已驗證資料。台語的部分有 120 人錄音，包含 11 小時錄製完成的片段，其中包含 3 小時的已驗證資料。在這 3 小時中，每個語音資料是以 MP3 格式儲存，標記上提供台語漢字和台語羅馬字。我們將在去除台語羅馬字後，使用此資料集嘗試在輸入台語語音時產生台語漢字輸出。

台語連續劇部分我們從民視戲劇館 Youtube 收集了約 920 小時的影片，其中市井豪門 74 小時，阿不拉的三個女人 46 小時，風水世

家 800 小時，並使用官方提供的字幕檔作為訓練輸入文字，我們也將各個連續劇的 80%做為訓練資料集，剩下各 10%為驗證和測試資料集。在影片前處理我們將所有影片根據每句字幕時間點以不超過 10 秒和不超過 30 秒分割。並依照 huggingface 的 load_dataset [23] 方式建立資料集，我們將使用此資料集嘗試在輸入台語語音時直接產生相對應的中文文字輸出。

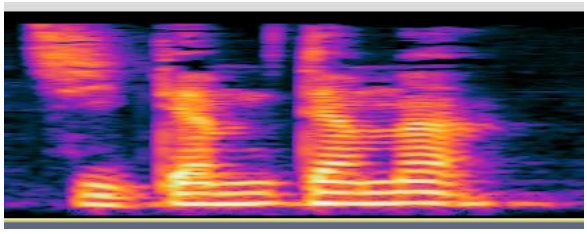
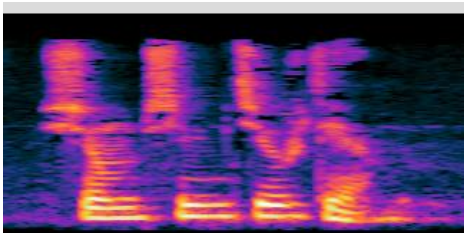
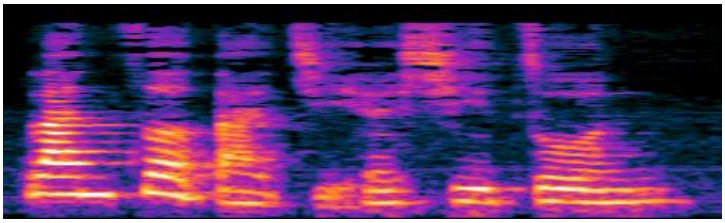
台語漢字(台語羅馬字)	音檔頻譜圖
一點點仔(tsi̍t-tiám-tiám-á)	
傷心酒店(siong-sim tsiú-tiàm)	
咱做代誌的時陣(Lán tsò tāi-tsì ê sî-tsūn)	

表 4.2 Common Voice nan-tw 資料集內容

檔名	文本	長度
市井_001_0094.mp3	世明 春梅 欠錢要還我還有孩子的補習費要繳	6 秒

阿不_001_0188.mp3	你會說台灣話啊 我是台灣人當然會說台灣話	4 秒
風水_001_0299.mp3	媽，先喝杯熱開水祛寒	3 秒

表 4.3 台語連續劇資料集內容

4.5.2 微調 whisper

我們參考 Whisper Fine-Tuning Event 的微調方式，使用 Hugging face 上提供之 openai/whisper-large-v2 和 openai/whisper-medium 模型訓練中繼點，並使用 Github 上的 whisper-fine-tuning-event/run_speech_recognition_seq2seq_streaming.py 進行微調，微調過程主要使用 python 的 Transformer 模組中的 Seq2SeqTrainer 進行微調。使用 CER 作為結果的指標。微調的主要參數如表 4.4：

參數名稱	參數值
max_steps	5000
per_device_train_batch_size	2
per_device_eval_batch_size	2
logging_steps	25
learning_rate	1e-5
warmup_steps	500
evaluation_strategy	steps
eval_steps	1000
save_strategy	steps

save_steps	1000
generation_max_length	225
optim	adamw_bnb_8bit

表 4.4 微調的主要參數

我們將訓練好的模型放在 HuggingFace 上，並提供 gradio 空間做測試使用。

4.5.3 辨識結果

辨識結果如下：

辨識模型	CER(%)
whisper 論文 medium	23.2
whisper 論文 large-v2	26.8
Hugging face Medium	13.4
Hugging face Large-v2	12.7
Medium Fine-tune	8.9
Large-v2 Fine-tune	8.9

表 4.5：對 Common Voice zh-tw 語料集進行 Fine-tune 之結果

辨識模型	CER(%)
Hugging face Medium	96.6
Hugging face Large-v2	96.7
Medium Fine-tune	50.9
Large-v2 Fine-tune	52.8

表 4.6：對 Common Voice nan-tw 語料集進行 Fine-tune 之結果

辨識模型	每句台詞長度(秒)	CER(%)
Medium Fine-tune	10	82.6
Medium Fine-tune	30	71.5
Large-v2 Fine-tune	10	53.8
Large-v2 Fine-tune	30	50.7

表 4.7：對台語連續劇進行 Fine-tune 之結果

台語文字	未微調辨識結果	微調後辨識結果
我攏有看著	我都看到了	我攏有看著
大海毋驚大水	大害不怕大罪	大海毋驚大水
大鑼大鼓	豆河老豆河公	大路大股

表 4.8：台語輸出微調辨識結果

字幕	未微調辨識結果	微調後辨識結果
李有志 不義之財不可得 不倫之愛不可行	余悠季 不羈季哉不叩叮 不倫季艾不叩行	李有志 不濟自財不可定 不倫不可行
你只會嗯 快點想辦法啊	你就 eκa 個這樣 農角園是有辦法的	你只要想 肯定有辦法
土地公 我真的不可能再有孩子了嗎	到底說我感情是不可能過敏的	究竟我會不會不可能有孩子

表 4.9：華語輸出微調辨識結果

從表 4.5 中我們可以發現，在 Whisper 的原始論文中，中文的辨識結果分別為 23.2%和 26.8%，我們猜測這可能是對簡體中文進行辨識的結果。而我們使用 huggingface 的 Whisper 和 Transformer 模組進行對繁體中文的語音辨識，在未進行微調的 medium 和 large-v2 模型的結果皆約為 13%，在進行微調後兩者的辨識結果皆可降低至 8.9%，顯示此微調的程式在對繁體中文的語料進行微調是有效果的。

從表 4.6 中我們可以發現，在使用 Common Voice 微調過後的 CER 有明顯的減少，代表著對 whisper 的模型進行台語的微調是可以讓 whisper 進行台語的語音辨識的。在從表 4.8 中觀察未微調和微調後的辨識結果時，我們也發現在未微調的 whisper 模型進行台語語音辨識時，可以辨識出為繁體中文，而且第一句「我攏有看著」也能翻譯出是「我都看到了」。但是其他二句皆是用聲音直接轉譯的結果，無法產生有意義的句子。在微調後的辨識結果，我們能看出前二句的辨識結果是完全正確的，雖然第三句「大鑼大鼓」未完全正確，但是仍比未微調的辨識正確率還要好，代表 whisper 微調是有效果的。

從表 4.7 中我們可以得出，在辨識結果去除跳針的輸出後，使用 Large-v2 模型進行 Fine-tune 後對台語連續劇進行語音辨識，是有能力直接對台語的語音輸出與字幕相似的結果的。我們在分析辨識結果的時候發現，表中的四種模型皆容易在輸出辨識結果時，發生某些詞語重複跳針的情況，我們認為有可能是因為在收集連續劇的聲音資料時，我們並沒有進行聲音的後處理，而是直接將抓取的資料直接進行訓練和辨識。 從表 4.9 中我們也可以發現，微調後的辨識

結果比起未進行微調的 whisper 更能產生正確的結果。

4.6 未來方向

4.6.1 語料集

本次實驗所使用的 common voice 語料集中，台語語料集只有 80 分鐘的資料量。雖然已能有不錯的微調結果，但是資料量還是稍嫌不足。在未來對 whisper 與台語的研究中，可以嘗試收集更多的台語語料，讓微調模型時能使用更多好的語料進行訓練。

而本次實驗所使用的連續劇，雖然資料量有約 920 個小時，但所有的聲音與台詞皆是原始資料，僅進行了時間點對齊和長度切割。在後續的實驗可以對此語料集進行聲音的後處理，去除連續劇的背景音樂，和對台詞的整理，嘗試用不同的方式切割音檔以符合斷句等。

4.6.2 微調小型模型

根據 whisper 對中文的微調，我們發現在 medium 與 large 的辨識結果皆優於其他三個模型，所以這次的研究主要是對這兩個模型進行。但是在使用或微調這兩個模型時都需要大量的 VRAM 才能進行，這會造成其他人在沒有強大的機器下會無法使用此模型。在未來的研究中，可以嘗試微調 base 或 small 等模型，再加上輸出結果的後續處理，看看是否能與 medium 或 large 有相近的辨識結果，使得大多數的人能夠使用此台語微調的 whisper 模型。

4.6.3 驗證微調結果的辨識率

目前我們用來驗證微調結果的方式是使用 CER 逐字比較辨識結果與答案的文本的差異，但是我們發現在使用台語辨識產生華語的時候，常常會有多個答案都是正確的情況，像是「毋過(m̄-koh)」可被翻譯成「不過」或「但是」;「媠(suí)」可被翻譯成「美麗」或「漂亮」。但是目前沒有人做出台語對華語的文字對應關係，也就造成了辨識率難以證實模型的辨識效果。我們認為我們需要有個台語對華語的文字對應關係標準，這樣對於以後對台語進行研究的人皆能有一套標準，在同樣的標準上才能進行辨識效果的比對。

第五章 結論

在本篇論文中，我們對語音辨識、歌聲偵測、 whisper 進行研究，並實作了即時聲控遊戲、卡拉 OK、台語語音辨識系統。

在即時聲控遊戲中，我們使用 CNN 訓練了辨識率約 90%的指令辨識模型，並實用於聲控遊戲中；在卡拉 OK 系統，我們使用 LRCN 訓練了歌聲偵測模型，在歌聲分離後能有效降低雜音的干擾；在台語語音辨識系統中，我們微調了 whisper，使 whisper 能對台語進行語音辨識。

參考資料

[1]. Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2022). Robust speech recognition via large-scale weak supervision. arXiv preprint arXiv:2212.04356.

[2]. Dahl, G. E., Yu, D., Deng, L., & Acero, A. (2011). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Transactions on audio, speech, and language processing, 20(1), 30-42.

[3]. Graves, A., Mohamed, A. R., & Hinton, G. (2013, May). Speech recognition with deep recurrent neural networks. In 2013 IEEE international conference on acoustics, speech and signal processing (pp. 6645-6649). Ieee.

[4]. Abdel-Hamid, O., Mohamed, A. R., Jiang, H., Deng, L., Penn, G., & Yu, D. (2014). Convolutional neural networks for speech recognition. IEEE/ACM Transactions on audio, speech, and language processing, 22(10), 1533-1545.

[5]. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[6]. Morris, A. C., Maier, V., & Green, P. (2004). From WER and RIL to MER and WIL: improved evaluation measures for connected speech recognition. In Eighth International Conference on Spoken Language Processing.

[7]. Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., &

Schmidhuber, J. (2016). LSTM: A search space odyssey. IEEE transactions on neural networks and learning systems, 28(10), 2222-2232.

[8]. Shi, X., Chen, Z., Wang, H., Yeung, D. Y., Wong, W. K., & Woo, W. C. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. Advances in neural information processing systems, 28.

[9]. Jansson, A., Humphrey, E., Montecchio, N., Bittner, R., Kumar, A., & Weyde, T. (2017). Singing voice separation with deep u-net convolutional networks.

[10]. Panayotov, V., Chen, G., Povey, D., & Khudanpur, S. (2015, April). Librispeech: an asr corpus based on public domain audio books. In 2015 IEEE international conference on acoustics, speech and signal processing (ICASSP) (pp. 5206-5210). IEEE.

[11]. Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., ... & Weber, G. (2019). Common voice: A massively-multilingual speech corpus. arXiv preprint arXiv:1912.06670.

[12]. Conneau, A., Ma, M., Khanuja, S., Zhang, Y., Axelrod, V., Dalmia, S., ... & Bapna, A. (2023, January). Fleurs: Few-shot learning evaluation of universal representations of speech. In 2022 IEEE Spoken Language Technology Workshop (SLT) (pp. 798-805). IEEE.

[13]. Warden, P. (2018). Speech commands: A dataset for limited-vocabulary speech recognition. arXiv preprint arXiv:1804.03209.

[14]. Liu, C. H., Lyu, R. Y., Zhan, W. Z., Wu, J. S., Zhu, D. D., & Shi, J. L. (2019, October). 基於卷積神經網路之台語關鍵詞辨識 (Taiwanese keyword recognition using Convolutional Neural Networks). In Proceedings of the 31st Conference on Computational Linguistics and Speech Processing (ROCLING 2019) (pp. 182-191).

[15]. SPEECH COMMAND CLASSIFICATION WITH TORCHAUDIO https://pytorch.org/tutorials/intermediate/speech_command_classification_with_torchaudio_tutorial.html

[16]. Simple audio recognition: Recognizing keywords https://www.tensorflow.org/tutorials/audio/simple_audio

[17]. 即時語音辨識遊戲：https://github.com/thomas880104/speech-command-game

[18]. Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

[19]. Hennequin, R., Khlif, A., Voituret, F., & Moussallam, M. (2020). Spleeter: a fast and efficient music source separation tool with pre-trained models. Journal of Open Source Software, 5(50), 2154.

[20]. Ramona, M., Richard, G., & David, B. (2008, March). Vocal detection in music with support vector machines. In 2008 IEEE international conference on acoustics, speech and signal processing (pp. 1885-1888). IEEE.

[21]. Raymundo Romero Arenas, Alfonso Gómez Espinosa, &

Benjamín Valdés Aguirre. (2022). Electrobyte for Singing Voice Detection (1.0) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.6757945

[22]. Whisper Fine-Tuning Event https://github.com/huggingface/community-events/tree/main/whisper-fine-tuning-event

[23]. Hugging Face Load https://huggingface.co/docs/datasets/loading

[24]. Sainath, T., & Parada, C. (2015). Convolutional neural networks for small-footprint keyword spotting.

[25]. 卡拉 OK 系統：https://github.com/thomas880104/py-karaoke

[26]. Silberschatz, Galvin, and Gagne, “Operating System Concepts,” 10th Edition, John Wiley & Sons, 2018.

[27]. pyQt https://doc.qt.io/qt-6/index.html

[28]. PyAudio http://www.portaudio.com/

[29]. Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., & Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2625-2634).

[30]. Zhang, X., Yu, Y., Gao, Y., Chen, X., & Li, W. (2020). Research on singing voice detection based on a long-term recurrent convolutional network with vocal separation and temporal smoothing. Electronics, 9(9), 1458.

[31]. Monir, R., Kostrzewa, D., & Mrozek, D. (2022). Singing voice

detection: a survey. Entropy, 24(1), 114.

[32]. Romero-Arenas, R., Gómez-Espinosa, A., & Valdés-Aguirre, B. (2022). Singing Voice Detection in Electronic Music with a Long-Term Recurrent Convolutional Network. Applied Sciences, 12(15), 7405.

[33]. Dai, W., Dai, C., Qu, S., Li, J., & Das, S. (2017, March). Very deep convolutional neural networks for raw waveforms. In 2017 IEEE international conference on acoustics, speech and signal processing (ICASSP) (pp. 421-425). IEEE.

[34]. Luo, Y., & Mesgarani, N. (2018, April). Tasnet: time-domain audio separation network for real-time, single-channel speech separation. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 696-700). IEEE.

[35]. Luo, Y., & Mesgarani, N. (2019). Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation. IEEE/ACM transactions on audio, speech, and language processing, 27(8), 1256-1266.

[36]. Yuan, W., Wang, S., Li, X., Unoki, M., & Wang, W. (2019). A skip attention mechanism for monaural singing voice separation. IEEE Signal Processing Letters, 26(10), 1481-1485.

[37]. global interpreter lock https://wiki.python.org/moin/GlobalInterpreterLock

[38]. Chen, P. J., Tran, K., Yang, Y., Du, J., Kao, J., Chung, Y. A., ... & Lee, A. (2022). Speech-to-Speech Translation For A Real-world

Unwritten Language. arXiv preprint arXiv:2211.06474.

[39]. https://ws.moe.edu.tw/001/Upload/userfiles/file/iongji/700iongji_1031222.pdf