Digital Speech Processing Final Project

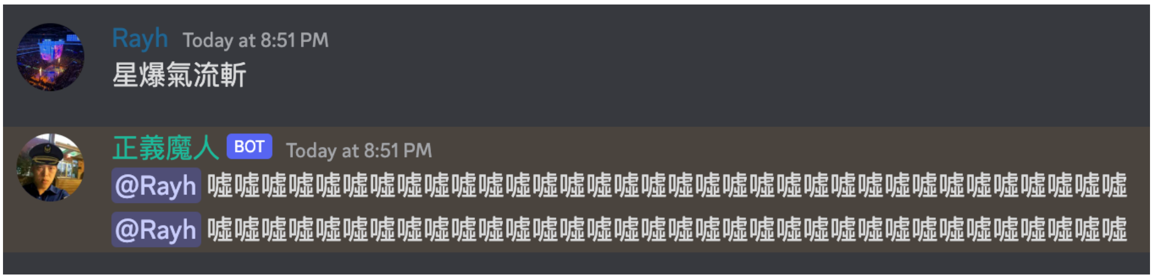
Real Time Lightweight Keyword Spotting

B09901171 黃柏睿

一. 動機

上個學期修了李宏毅教授的機器學習，學習到了許多語音、語言在機器學習上的處理，這學期修了這一堂課，學到了更多以前人在機器學習未問世以前，用少量資料跟較差的計算能力處理語音的技術。兩者相比之下，我較喜歡後者的智慧，可以直接動腦解決問題，學到的東西也更多。

我們高中同學間有一個有趣的文化，如果有人說「星爆」的話，就要噓他，所以我在我們的 Discord 群組寫了一個機器人，有人說「星爆」機器人就會去噓他，在文字頻道中非常容易做到，但在語音頻道就不容易了，所以我一直都想要讓機器人有辨識語音中「星爆」的能力。



（圖一）文字頻道機器人 (ref: me)

由於這個機器人的伺服器只是一個小小的 Raspberry Pi，運算能力有限，這學期我也有修嵌入式系統實驗，學會了一些平行化的程式能力，剛好可以用來處理語音同步的問題，所以我想要時做出一個即時、低運算量的關鍵詞識別系統。

二. 研究過程

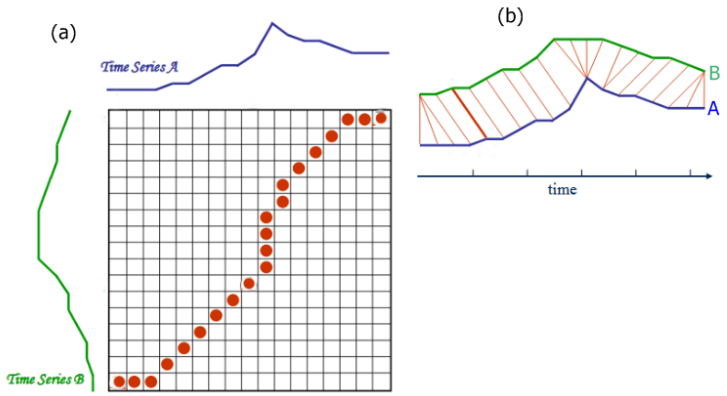
（一） 第一版 - 頭對頭、尾對尾、非即時

要實作關鍵詞識別，我第一個想到的就是 DTW (Dynamic Time Wrapping)，把預先錄製的參考音檔跟要識別的聲音比較，計算出一個最佳配對跟最小 Distance，小於一定的門檻就視為辨識成功。

首先第一步是處理音訊輸入，我使用的是 PyAudio 這個套件，他可以把麥克風時間域的資料用指定的 Sample Rate 一個一個 Chuck 讀出來，我選用網路音訊最常用的 16000Hz 作為 Sample Rate，一次讀取一秒的資料。由於 DTW 有頭對頭、尾對尾的 Constraint，所以需要適當把音訊中前後安靜的片段剪掉，才能完整配對整段音檔，我使用的方法是計算一段的 RMS (方均根)震幅作為平均能量，先錄一段背景音，再錄一段講話的，用兩段的平均能量的平均作為 Threshold，把其他音檔的前後裁切掉，得到精準的關鍵詞音檔。

接著用上課教的 DTW 比對兩段音訊，我嘗試使用時間域跟 MFCC 兩種訊號做比對，想說時間域只有一維，MFCC 有 39 維，使用時間域訊號若能成功應該能有效提升運算效率，但實驗後發現用時間域訊號的準確率遠不如 MFCC，且 MFCC 的運算速度沒有明顯的變慢，所以我選用 MFCC 作為音訊特徵。這裡我用的套件是 Librosa，他是 Python 用來處理音訊的套件，可以協助我讀檔跟轉換 MFCC，我使用 window size: 25ms、step size: 10ms。

DTW 的演算法跟上課教的一樣，用一個二維矩陣紀錄沿路上最低 Distance，先把第一個 Column 跟 Row 初始化好，把 Transition 的方向限制在上、斜上、右三種，就可以一路用 Dynamic Programming 完成整個表，而對角線的另一頭就是最佳配對的最佳 Distance，但是這個 Distance 要怎麼判別辨識結果上課就沒有教了，我的做法是先 Backtrack 回去最佳序列的長度，再把最佳 Distance 除以長度得到平均，這樣可以降低訊號長短對 Distance 的直接影響。最後再測試幾段關鍵字的訊號跟非關鍵字的訊號，得到一個適當的 Threshold。



（圖二）頭對頭、尾對尾 DTW

(ref: https://medium.datadriveninvestor.com/dynamic-time-warping-dtw-d51d1a1e4afc)

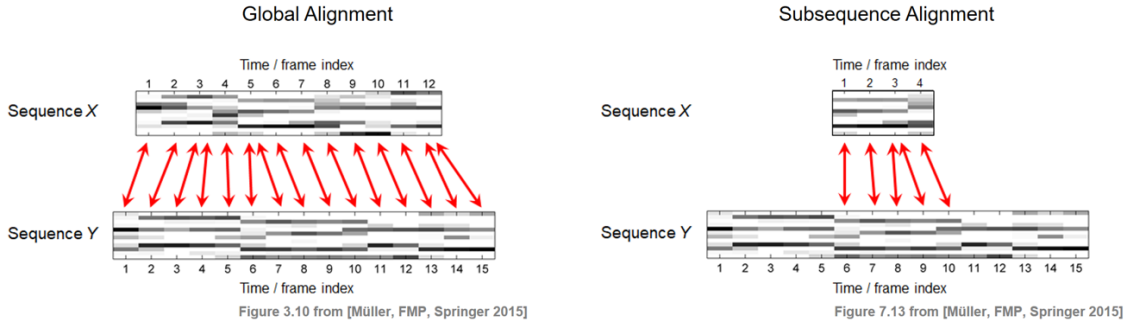
使用這個架構對單詞(沒有前後文)的正確率可以達到 90%以上，並且對較長跟較短的輸入也能正確分辨，展現了 DTW 的優勢。

（二） 第二版 – Subsequence DTW

第一版雖然有很好的正確率，但有一個致命的問題：關鍵字出現在句子中會完全無法分辨識。因為我使用的方法是把前後靜默的片段剪掉，若是連續音序就無法辨識關鍵出現在哪一段，而 DTW 又強制頭對頭、尾對尾對齊，造成 DTW 完全在比較錯誤的音訊。

首先我嘗試能不能把一段句子的音訊切成好幾個單詞，但實際觀察訊號後，發現在時間域會有連音的問題，不是每一組詞之間都有明確的分界，而在頻率域雖然可以看出明確的分界，但分界的單位是 Phoneme，多個 Phoneme 組成詞的可能性過多，並不是一個好的做法。

研究了一陣子後，我發現有另一種形式的 DTW 叫 Subsequence DTW，他的目標是把一段小序列跟一段大序列比對，找出小序列作為大序列的 Subsequence 最好的配對，這就是我想要做到的事情：在一段句子中找出關鍵字出現在哪。



（圖三）Subsequence DTW

(ref: https://www.audiolabs-erlangen.de/resources/MIR/FMP/C7/C7S2_SubsequenceDTW.html)

其演算法跟原本的 DTW 大同小異，只是在初始化長序列時不用累加 Distance，只要算單一個 Frame 的 Distance，接著用一樣的 Dynamic Programming 把表格填滿，但 Backtrack 時不是從最角落的開始，要選最後一個 Row 中 Distance 最小的那個開始，一路走回第一個 Row，這一段序列就是最佳的 Subsequence 配對。

Algorithm: SUBSEQUENCE DTW — Exercise 7.6 from [Müller, FMP, Springer 2015]

Input: Cost matrix $\mathbf{C}$ of size $N \times M$
Output: Accumulated cost matrix $\mathbf{D}$
Indices $a^*, b^* \in [1:M]$ of an optimal subsequence of Y
Optimal warping path P^* between X and $Y(a^*:b^*)$

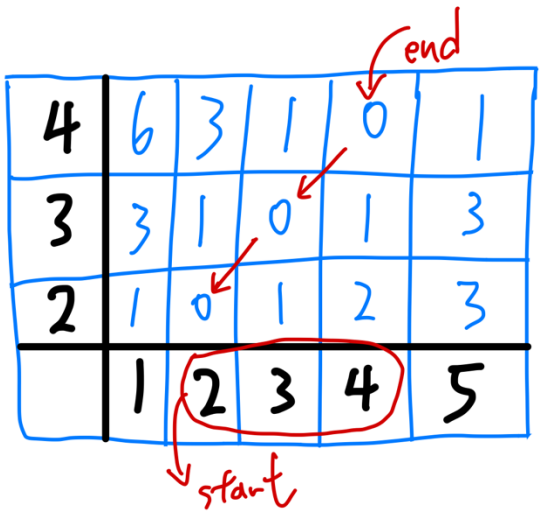
Procedure: Initialize $(N \times M)$ matrix $\mathbf{D}$ by $\mathbf{D}(n,1) = \sum_{k=1}^{n} \mathbf{C}(k,1)$ for $n \in [1:N]$ and $\mathbf{D}(1,m) = \mathbf{C}(1,m)$ for $m \in [1:M]$. Then compute in a nested loop for $n = 2,\ldots,N$ and $m = 2,\ldots,M$:

$$\mathbf{D}(n,m) = \mathbf{C}(n,m) + \min\{\mathbf{D}(n-1,m-1), \mathbf{D}(n-1,m), \mathbf{D}(n,m-1)\}.$$

Set $b^* = \text{argmin}_{b \in [1:M]} \mathbf{D}(N,b)$. (If 'argmin' is not unique, take smallest index.)
Set $\ell = 1$ and $q_\ell = (N, b^*)$.
Then repeat the following steps until $q_\ell = (1,m)$ for some $m \in [1:M]$:

Increase ℓ by one and let $(n,m) = q_{\ell-1}$.
If $m = 1$, then $q_\ell = (n-1, 1)$,
else $q_\ell = \text{argmin}\{\mathbf{D}(n-1,m-1), \mathbf{D}(n-1,m), \mathbf{D}(n,m-1)\}$.
(If 'argmin' is not unique, take lexicographically smallest cell.)

Set $L = \ell$ and $a^* = m$. Return $\mathbf{D}$, a^*, b^*, and $P^* = (q_L, q_{L-1}, \ldots, q_1)$.



（圖四）Subsequence DTW algorithm

(ref: https://www.audiolabs-erlangen.de/resources/MIR/FMP/C7/C7S2_SubsequenceDTW.html, me)

使用 Subsequence DTW 就可以讓我們在一段句子中找到關鍵字的位置，並且計算相似度，除了這個好處外，Subsequence DTW 也可以提供我們關鍵字在句子中的長度，我們就有長度、Distance 兩個指標可以做判斷，例如「Distance 小於 500 且長度大於 0.3 秒」，我們就可以用較寬鬆的 Distance 門檻去辨識，減少誤判。

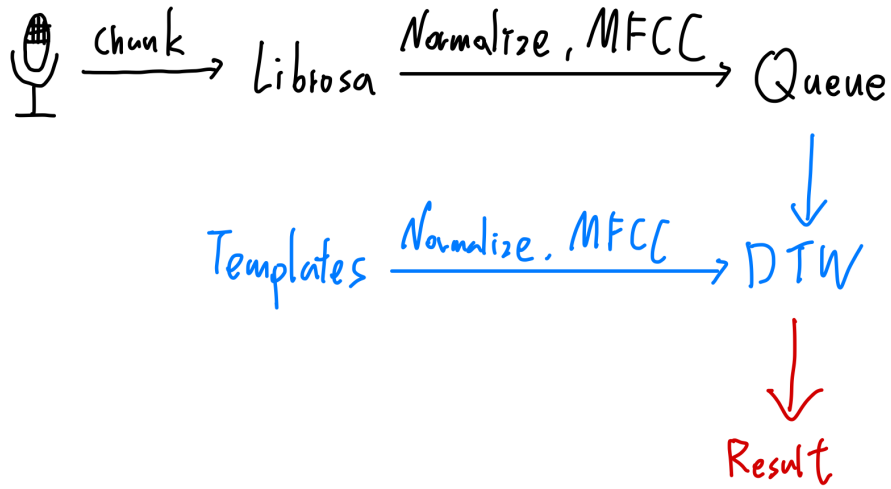
（三） 第三版 – Real Time、平行化、音量標準化

前幾版都還是非同步在處理輸入，錄一段處理一段，並沒有做到 Real Time 辨識，我的做法是使用兩個 Thread，一個負責在背景處理音訊輸入，每收到一個 Chunk 會把資料推到一個 Queue，另一個 Thread 再一一拿出來辨識，我還使用了 Sliding Window 的方式，讓每一個 Chunk 間有 0.25 秒的重疊，避免關鍵字被切斷在兩個 Chunk 之間。

但這樣要處理的資料就會比輸入的資料多 25%，實測下來長期運作下會供過於求，處理資料的速度趕不上輸入的速度，造成 Queue 越來越多，不是真的 Real Time。我的解決方法是用前面提到的 RMS 作為門檻，如果輸入音訊音量過小直接忽略不辨識，這樣一來就算在吵雜時 Queue 有累積資料，在安靜時也會慢慢清空，讓系統能長期運作下去。

除了平行處理音訊輸入跟辨識外，在 DTW 的部分我也有使用平行化，我發現若只使用一個關鍵字模板的辨識效果較差，使用多個有差異性的模板(音高、速度不同)來比較可以有效提升品質，但這樣運算量會跟模板的數量呈正比，要 Real Time 處理就不能直接用一堆模板，所以我先把多個模板兩兩比較，把相近的幾個模板只留一個做為代表，篩選出最後幾個差異性較大的，並且在辨識時用 Multithread 平行處理每個模板，有效兼顧運算量和辨識能力。

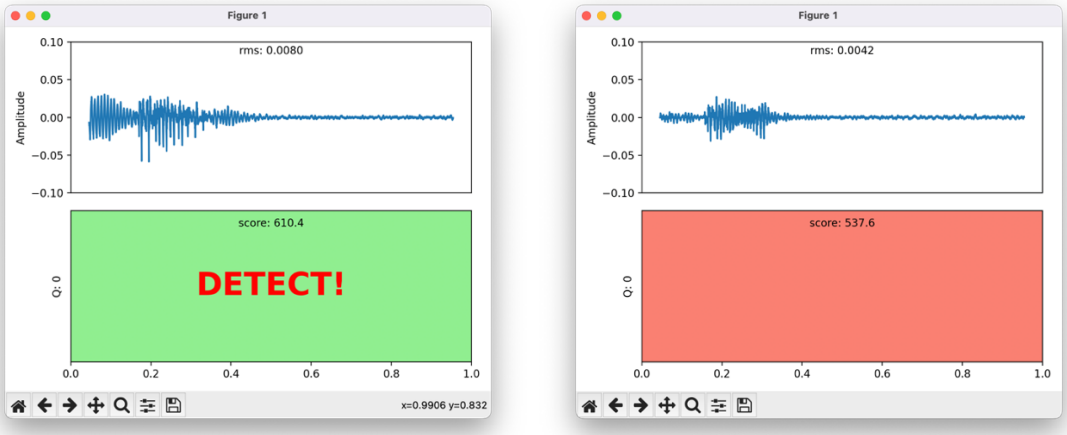
經過幾天的測試跟給同學玩玩看後，我發現有時會突然完全無法辨識，分析了一下發現是麥克風在不同電腦、環境、驅動程式下的表現會有所不同，最明顯的是音量，模板錄製時的狀態跟測試時的狀態有差異，嚴重影響辨識結果。我的解決方式是把模板和輸入的音訊做標準化，讓所有聲音有相同的音量範圍，有效提升此系統隨著時間、環境變化的穩定性。



（圖五）Framework (ref: me)

三. 研究成果

把程式運行起來後會開啟一個每 0.75 秒更新一次視窗，上半部是音訊時間域的波形，並且會標示每個 Frame RMS 的數值，下半部是辨識結果，辨識到關鍵詞時會顯示綠底「DETECT!」，平時為紅底空白，並且會標示每個 Frame 的 Score (也就是 DTW 的平均 Distance)，也會標示 Queue 目前累積了幾筆資料。



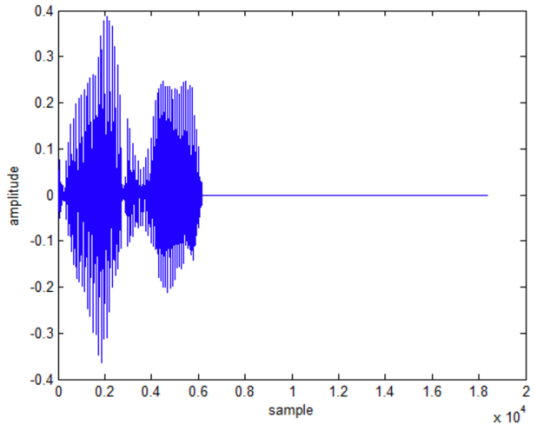
（圖六）系統運作圖 (ref: me)

四. Paper Review

在完成這個專題後，我研究了一些目前較新的 Keyword Spotting 研究，也研究了一些使用 DTW 的相關做法。

（一） Speech recognition using MFCC and DTW, 2014

這篇論文的做法和我的第一版相似，直接用 MFCC 跟 DTW 做 Isolated word recognition，跟我不同的地方在他做 Preprocessing 時是先過一個 Noise-gate，把震幅低於一個值訊號直接歸零，接著做 Zero Alignment，把音訊開始的地方對齊在時間零的地方，再對這段訊號取 MFCC。我認為這個做法的優勢在取 MFCC 時不會被雜訊影響，因為雜訊被歸零了，而且 Noise-gate 連關鍵字中的靜默都能處理到，是我的方法無法處理的，這種處理方法值得學習。

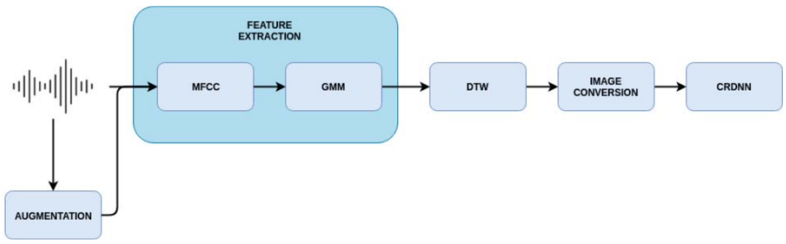


（圖七）Preprocessed signal after noise-gate and zero alignment

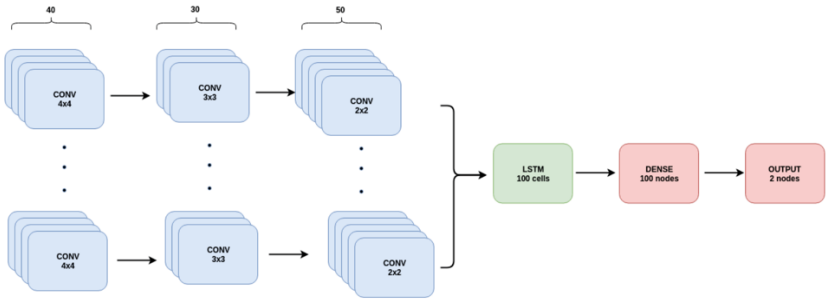
在 DTW 的部分，他用的是頭對頭、尾對尾的版本，並且是在多個關鍵字模板中找最小的那個作為辨識結果，沒有處理 OOV 的問題，跟我的目標較不相同。

（二） Keyword Spotting using Dynamic Time Warping and Convolutional Recurrent Networks

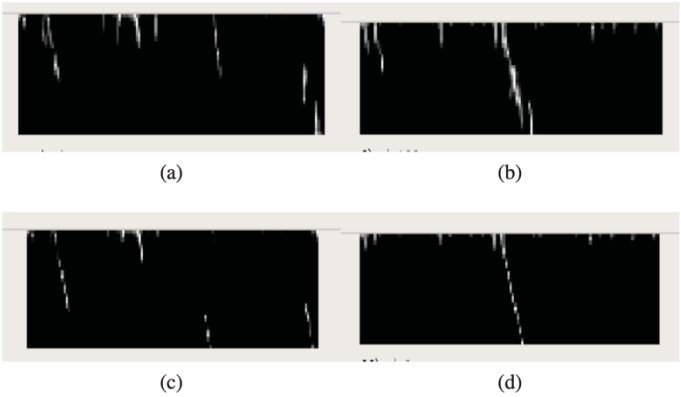
這篇也是用 MFCC 作為 Feature，但不是直接用 MFCC 當作輸入，他有先經過一個 GMM，再用這些 Gaussian 當作輸入，在資料量夠大時這種做法可以讓 MFCC 有一個更 Smooth 表達方式，讓模型可以更適應在各種環境。接著通過 DTW，因為前面經過了 GMM，這裡用的 Distance Function 是 KL Divergence 不是 Euclidean Distance，得到兩組序列配對的 Distance Matrix 後轉成一個灰階圖片，再通過 CRDNN (Convolutional Recurrent Neural Network) 做 Classification，這裡作者提到一個有趣的地方，一般在訓練 CNN 時，為了避免 Overfitting 會做 Data Augmentation，像是旋轉、縮放、調整亮度等，但是在這個任務下直接改變 Distance Matrix 並不合理，所以作者是在音訊檔做 Augmentation，在每一個句子做亂數的平移。最後可以在十個字中得到 94.5%的正確率。



（圖八）Pipeline



（圖九）CRDNN architecture



（圖十）Grayscale DTW matrix image

一開始看到標題時以為作者是想用 DTW 切出可能序列後，再用 CRDNN 直接看 MFCC，沒想到是用 CRDNN 看 DTW 的 Distance Matrix，我覺得非常有趣，比起直接用 CNN 去看 Spectrum 可以減少很多運算量跟訓練難度，也有不錯的結果，用 GMM 處理 MFCC 也是一個值得學習的技巧。

（三） Learning Efficient Representations for Keyword Spotting with Triplet Loss, 2021

這篇是 Google Speech Command Dataset 上的 SOTA，他直接把音訊轉成 80 維的 MFCC，後丟進一個 Resnet 當作圖片去辨識，並且使用 Triplet Loss，最後可以在 35 個關鍵字中達到 97%的正確率。

基本上我查到比較新在做 keyword spotting 的都是用 CNN，也都做得不錯，可見未來 ML 在語音處理上仍很有潛力。

五. 未來展望與問題討論

（一） 相似音問題

我的方法使用 DTW 的相似度作為辨識指標，在遇到相似詞時容易造成誤判，尤其是我有一個同學叫做「星號」，跟我要辨識的「星爆」極為相似。同學如果只是在叫他的名字卻被機器人噓不太合理，而且這是一個經常發生的情境。

我有嘗試過在模版同時存放「星號」跟「星爆」兩種，在辨識時就算「星爆」通過辨識門檻，但更像「星號」的話仍當作 False，這個做法可以有效解決「星號」跟「星爆」的問題。但中文的相似音非常多，聲調也常常很相似，如果每個字都要建模版並不是一個好方法，DTW 在這個問題上可能就較不適合。

（二） 適應各種語者的聲音

我完成後有給同學測試，發現換一個語者會嚴重影響辨識正確率，但如果有存放同學的模板，辨識率就可以大幅上升。推測是 MFCC 是一個頻率域上的特徵，不同人的聲音

理所當然會有不同的 MFCC，跟同學的聲音跟我的模板比較自然會有較高的 Distance。

我研究了一下別人是如何處理這個問題的，最多人的做法是直接搜集足夠多人的聲音作為模板，用運算量跟資料量彌補。這個方法對我來說可行，因為我的高中同學只有數十個，搜集所有人的聲音作為模板並不困難。另外也有做法是讓 MFCC 過一個 GMM，跟上述那篇論文做法相似，讓原本的 MFCC 變成一個較 Smooth 的 Vector Space，降低辨識難度，此方法對我來說較困難，因為 GMM 需要足夠多的資料來訓練，要同學們一直幫我搜集資料並不容易。

（三） 網路音訊 (VoIP) 的處理

我的系統最終要接收的是網路音訊的輸入，而網路音訊在解析度、延遲、音量、完整性都較不穩定，且 Discord 目前對語音的 API 較不完整，要實際變成一個 Discord 機器人上線做辨識和能還需要努力。

六. 結論與心得

我成功時做出了一個可以即時分辨關鍵字的系統，並且只需要非常少的資料跟運算效能，不管是獨立的單詞或是出現在句子中都能成功辨識。

在這個過程中我學到了許多東西，尤其是在音訊前處理的部分，以前做 ML 時都是用別人的 Dataset，不知道原來真的要處理自己的聲音跟資料會有這麼多問題，尤其是在音量標準化的部分。

DTW 的部分也學到了很多，從上課教的頭對頭、尾對尾版本到後來的 Subsequence DTW，讓我的模型可以辨識一整段話中的一個詞。後來讀別人的論文時也能看懂作者遇到的問題，跟他為什麼嘗試這些方法。把上課教的知識經過思考後做出自己的作品，再到能看懂別人的研究，讓我體驗到了做研究大概的感覺，我覺得相當踏實。

七. 參考資料

[1] Albert, E.-T., Lemnaru, C., Dinsoreanu, M., & Potolea, R. (2019). Keyword spotting using dynamic time warping and convolutional recurrent networks. *2019 IEEE 15th International Conference on Intelligent Computer Communication and Processing (ICCP)*. https://doi.org/10.1109/iccp48234.2019.8959645

[2] Bhadragiri Jagan Mohan, & Ramesh Babu N. (2014). Speech recognition using MFCC and DTW. *2014 International Conference on Advances in Electrical Engineering (ICAEE)*. https://doi.org/10.1109/icaee.2014.6838564

[3] Librosa. (n.d.). Retrieved December 28, 2022, from https://librosa.org/

[4] *PyAudio*. PyAudio: Cross-platform audio I/O for Python, with PortAudio. (n.d.). Retrieved December 28, 2022, from https://people.csail.mit.edu/hubert/pyaudio/

[5] *Subsequence DTW*. AudioLabs. (n.d.). Retrieved December 28, 2022, from https://www.audiolabs-erlangen.de/resources/MIR/FMP/C7/C7S2_SubsequenceDTW.html

[6] Vygon, R., & Mikhaylovskiy, N. (2021). Learning efficient representations for keyword spotting with triplet loss. *Speech and Computer*, 773–785. https://doi.org/10.1007/978-3-030-87802-3_69