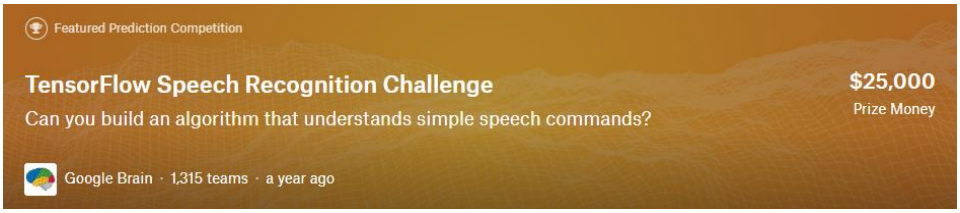
數位語音處理概論 Final Project Report

電機四 B04505025 陳在賢

1. 專題動機及簡介

在修了近一學期的數位語音處理概論後，發現課程的技術核心大都在機器學習的範疇中：從 EM、SVM、PCA 等傳統技巧、到近代蓬勃發展的深度學習皆有提到。因此我希望能在這次 final project 中實作其一演算法。

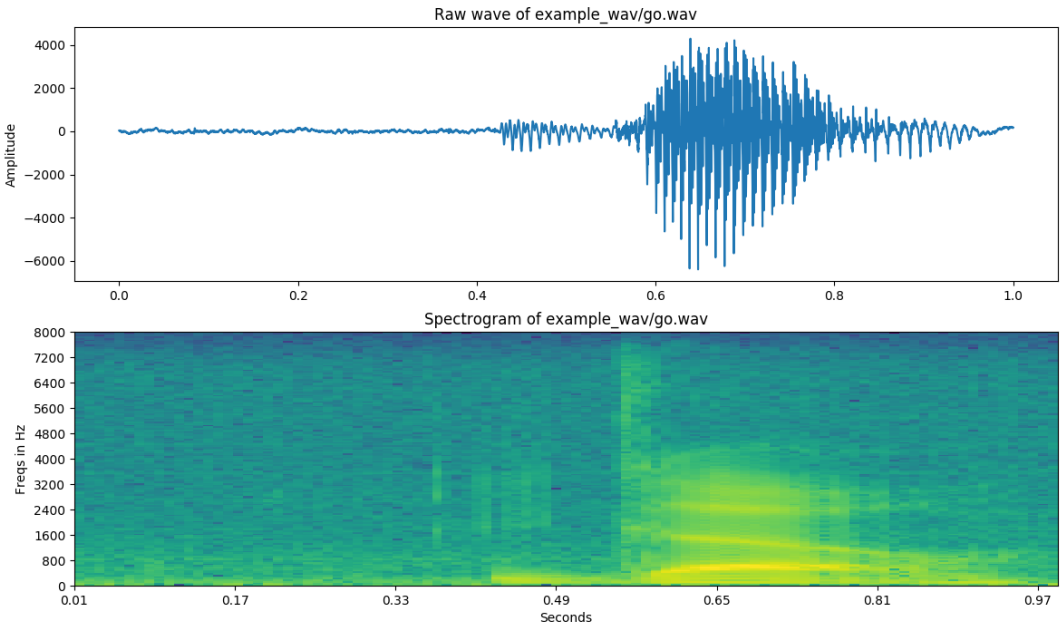
但對我而言，實作機器學習最難的在於獲得 training data。而在網上查詢多日後，發現在 Kaggle（公開的線上機器學習競賽平台）上，一年前有舉辦一個名為「TensorFlow Speech Recognition Challenge」的競賽。



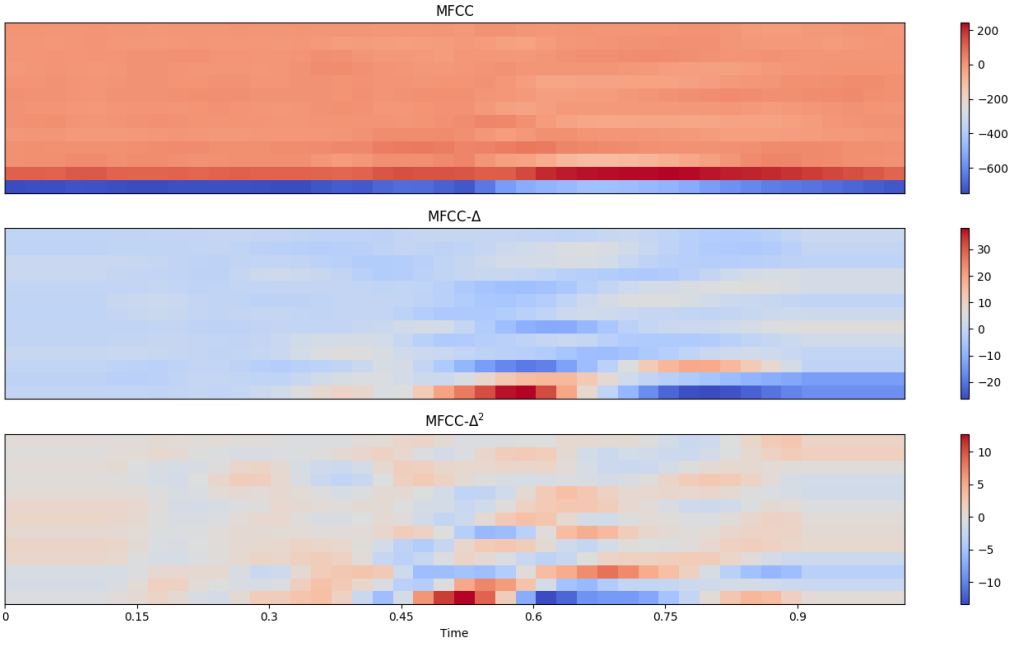
競賽中提供大量長度為 1 秒的音訊作為 training data，內容可能為一個單字（共 30 種字）或 silence，也提供背景雜訊以增強對 silence 模型的學習。因此我就借用大會提供的 data 來完成這次專題，但考量到電腦硬體資源不足（ex: GPU, RAM…），本次專題僅用其中 6 個指令單字 (yes, no, go, stop, right, left)，而每字約有 2350 筆 training data。

在模型的選擇上，由於我一直對圖像辨識很感有興趣，而且教授也常常在上課時提到結合深度學習的語音辨識是目前主流的做法。因此就決定用 Ch9 所提及的：使用 CNN 作圖像辨識以達成語音辨識。順便藉此機會了解 CNN 能辨識物件的基本原理，以及如何實現 CNN 及增進其學習效能。

至於要根據哪種圖像做辨別呢？講義中是說用 spectrogram，如下圖所示：



但在看了網路上其他以 CNN 做語音辨識的相關文章，發現多數人是以 39dims-MFCC vs frame(time) 為輸入圖像，如下圖所示：



P.S.上幾張圖皆為我用 code 模擬出來的，可於 demo 時展示

後來想想也合理，畢竟目前語音辨識最成功的 HMM 也是將語音轉換為 MFCC 後做學習及預測，代表 MFCC 在某方面應該是個更好的 feature。

2. 模型說明

根據上述簡介可知，本次專題大致分為語音前處理、CNN 模型訓練及預測等兩個步驟。底下將對此二部份做進一步的說明：

(1) 語音前處理

有關語音處理的部分我是使用 python 的 librosa lib.。在此步須將輸入的 wav 檔（長度為 1 秒），轉換為共 #frame 個 39dims-MFCC。為此我寫了個 wav2mfcc 函式，相關 code 如下圖所示：

```
19  def wav2mfcc(file_path, max_len=11, n_mfcc=20):
20      wave, sr = librosa.load(file_path, mono=True, sr=None)
21      #mfcc = librosa.feature.mfcc(wave, sr=sr, n_mfcc=n_mfcc)
22
23      # pad on the first and second deltas while we're at it
24      mfcc = librosa.feature.mfcc(wave, sr=sr, n_mfcc=n_mfcc)
25      delta1_mfcc = librosa.feature.delta(mfcc, order=1)
26      delta2_mfcc = librosa.feature.delta(mfcc, order=2)
27      mfcc = np.vstack((mfcc,delta1_mfcc,delta2_mfcc))
28
29      # If maximum length exceeds mfcc lengths then pad the remaining ones
30      if (max_len > mfcc.shape[1]):
31          pad_width = max_len - mfcc.shape[1]
32          mfcc = np.pad(mfcc, pad_width=((0, 0), (0, pad_width)), mode='constant')
33
34      # Else cutoff the remaining parts
35      else:
36          mfcc = mfcc[:, mfcc.shape[1]-max_len:]
37
38      return mfcc
```

輸入參數包含：file_path（欲轉換音訊檔檔名）、max_len（#frame，本次專題使用 32）、n_mfcc（MFCC 維度，本次專題使用 13）。

大致可分為三步：

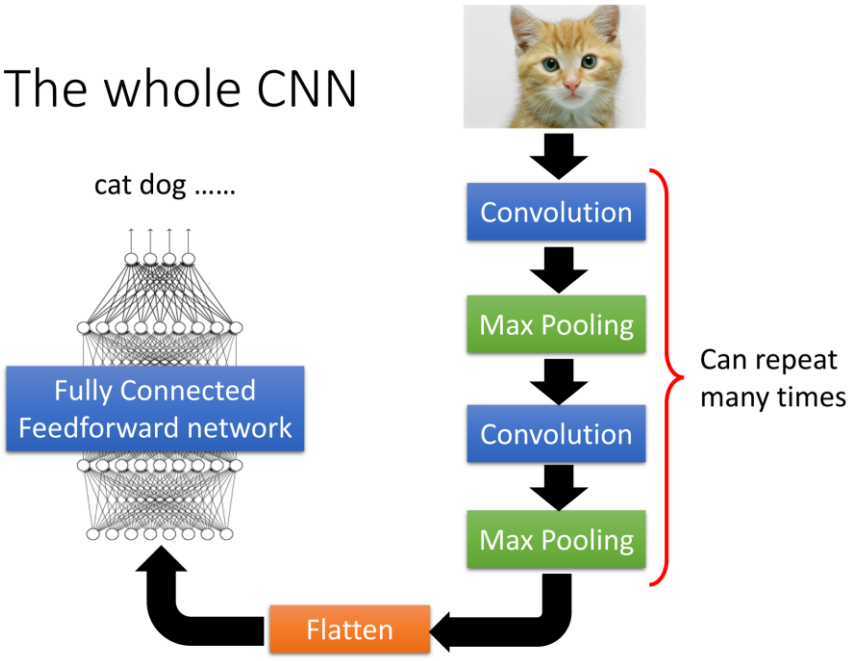
1) (Line20) 將音訊檔讀入為一維 array（為不同時間的音訊振福）。
2) (Line25-27) 再來對音訊做 MFCC 的特徵抽取。首先對 librosa.feature.mfcc 函數輸入前述一維 array，會輸出大小為 (n_mfcc = 13, #frame) 的二維 array。再計算 MFCC 的一次及二次 delta 值，並將三者合併為 39dims 的 MFCC。
3) (Line30-36) 對 #frame 不足/超過 max_len = 32 的音訊，以 padding/去除後面 frame 的方式，強迫將 #frame 調整至 32。

經過前處理後，語音訊號可被轉換為 (39, 32) 的二維 MFCC array。

(2) CNN 模型訓練及預測

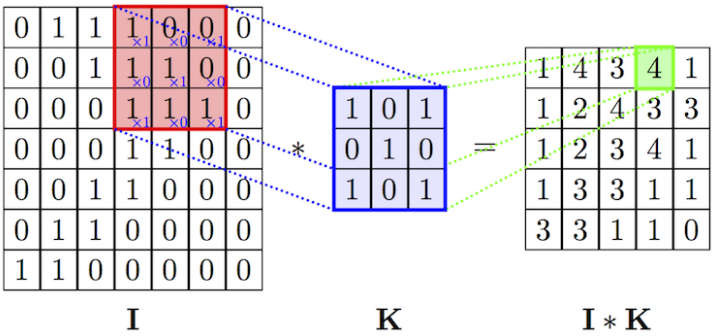
有關 Neural Network 建置的部分我是使用 python 的 Keras lib.。在此步，會將上述資料視為長寬為 (39, 32) 的 1 channel（灰階）影像，送入 CNN 做影像訓練及辨識，輸出即為該音訊檔對應的單字。

而 CNN 的模型架構及實現主要是參考林弘毅教授的開放式課程，架構圖如下所示：

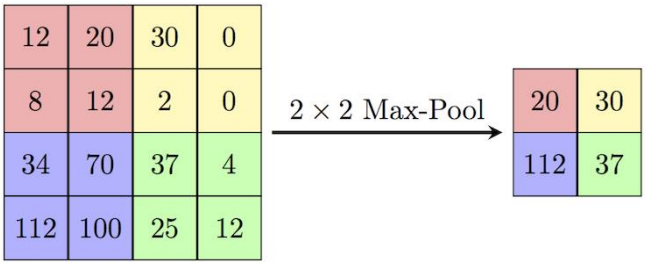


對模型輸入影像後，會先交錯進入多層 Convolution layer 及 Max Pooling layer，兩種 layer 的功用為：

1) Convolution: 為用多 channels 的小尺寸 kernel 對輸入影像作 convolution 運算，如下圖所示。由於當 sub-image（kernel 在輸入影像所涵蓋的範圍）的 pixel value 分布類似 kernel 值分布時，在旋積後可得較大的值（或說較大的分數），反之亦然，所以此步可取出輸入影像的特殊幾何特徵。



2) Max Pooling: 同樣為用小尺寸 kernel 走完整張輸入影像，只是對每個 sub-image，會取出最大的 pixel value 作為代表，其他 pixel 則去除，如下圖所示。此步會去除 convolution 後分數較低的 pixel，而專注於在 convolution 時所抓取到特殊幾何特徵的 pixel，因此可大幅降低之後的 layer 的計算量。



通過此兩種 layer 後，會將影像 flatten 為 1 維 array，之後送入多層 Dense layer 運算，此部分跟 DNN 相同就不多贅述。

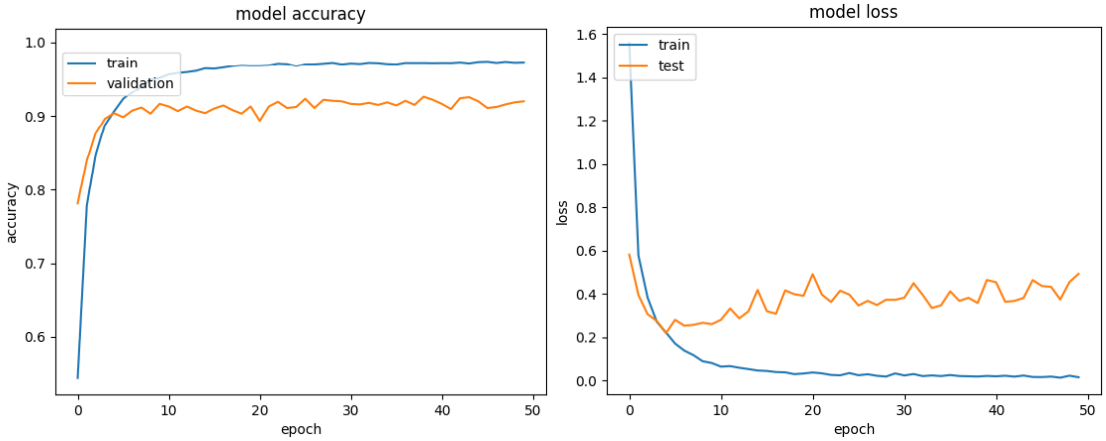
了解 CNN 的架構後，接著就可以用 python 實現，相關 code 如下：

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(2, 2), activation='relu',
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(2, 2), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(2, 2), activation='relu')
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(lr=0.35),
              metrics=['accuracy'])
log = model.fit(X_train, y_train_hot, batch_size=batch_size,
```

從上圖可知用 Keras 的好處在於容易實現模型的建置，就如同堆積木一樣，一層層將 layer 堆疊。我的模型架構是使用三層 Convolution + MaxPooling layer，並在 flatten 後用三層 Dense layer。

堆完模型後，要對 model 做 compile，此時可決定優化時要根據哪個 loss function 及 optimizer。最後再將 model fit 於 training data，在做 training 時，模型會訓練多次 epoch (iteration)，我們可根據每個 epoch 的 training/validation loss/accuracy 畫出 training curve，觀察模型進步狀況。

我的初始模型的 training curve 如下圖所示：



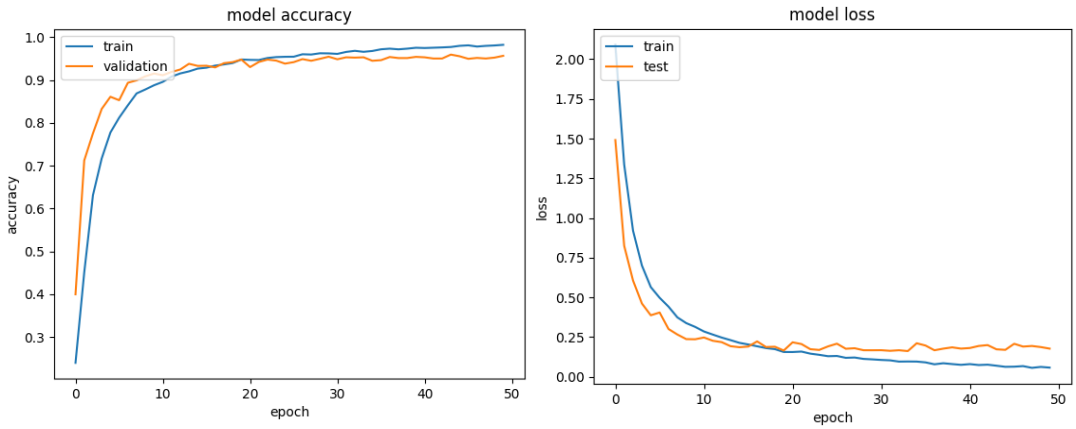
從上兩張圖可看出初始模型有三個主要的問題：

1) Validation acc./loss 起伏很大，模型相當不穩定
2) Validation acc. 很快就到達極大值，loss 甚至到後來有上升的趨勢，為明顯的 overfitting 現象
3) Training accuracy 約在 97% 就到達極大值，表示模型還有能力在 training data 中訓練得更好，卻沒有達到

關於上述三個問題，我各自提出三個應對的解法：

1) 模型不穩定表示每次修正模型的修正量太大，因此在 compile 選擇 optimizer 的同時，應指定較小的 learning rate，如此可限制每個 epoch 訓練後，不會過度更新模型。
2) overfitting 是機器學習中普遍需要克服的大問題，在查詢多筆資料後，得知目前在深度學習中處理 overfitting 的主流方法為加入 dropout layer。原理為在每個 epoch 訓練時，都用限定部分架構訓練，以確保不讓整個架構過度學習於特定 training data。
3) training accuracy 的過早飽和通常表示架構的複雜度不夠（即參數量不夠），而無法完全學會 training data。因此可透過增加 convolution layer 中 kernel 的 channel 數，以及 dense layer 中的 node 數，提升模型的學習能力。

根據上三個解法調整完模型後（前頁 code 為已調整完後的架構），新的 training curve 如下所示：



可看出上述三個問題都被解決了，validtion acc./loss 不再過早飽和及不穩定跳動，training acc. 在後幾個 epoch 也可上到 99%。

另外在新模型中，validation (testing) accuracy 可達到 94% 以上，也表示至此我的模型已經幾乎可對此 6 個語音指令正確辨別了！

3. 結論及未來展望

我在本次專題所設計出的模型，可先將語音資料轉為 (n_mfcc = 39, #frame = 32) 的二維資料，再送入 CNN 訓練語音內容所對應的指令單字，而最終模型可在 testing data 中達到 94% 以上的準確率。

然而這只是開始，往後還可結合其他技巧，已達到更廣泛應用。例如：搭配背景雜訊的訓練及聲音採樣的功能，就可能可實現語音喚醒的功能。

而我在本次專題也真的收益良多，除了學習到如何以 code 達到上課所教的：對語音抽取 MFCC 特徵的量化方式，還了解到目前相當火熱的 CNN 的基本原理、架構設計、實現方法及改善策略。也算是邁出了自己在語音辨識的第一步，往後還可繼續鑽研，做出更具應用性的產品。