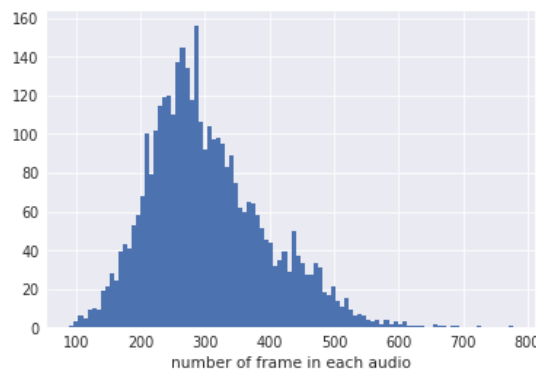


## Model description

### RNN

- **Preprocessing:**

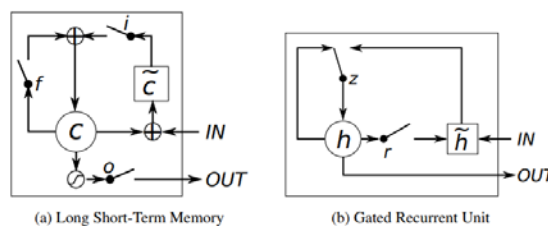
由於 RNN 需要給定三個維度，(batch size, frame size, feature size)，而其中，frame size 也是固定的，所以需要切固定數量的 frame 給單一 sample。這裡我用 200 個 frame 當一個 sample，同一音檔有可能會用到兩至三個 sample (即超過 200 個 frame，相關統計可見圖一)，因此，在同一音檔，sample 跟 sample 之間有用 overlap 的方式讓 RNN 學到連續的特徵。特徵的部分則是將 mfcc 跟 fbank concatenate 起來變成 108 維的 feature。最後 input shape 就是(batch size, 200,108)。Batch size 的部分經實驗後發現 128 表現比較穩定。



圖一、音檔 frame 個數統計

- **Model structure:**

圖三是我初始版的 RNN 架構，用兩層 LSTM 接 2 層 fully connected layer，最後輸出 49 維的 array (48 phone + 1 zero padding class)，optimizer 用 Adam。經過 100 個 epochs 後，kaggle 上的 Levenshtein distance 約為 12.3551。使用 LSTM 的原因是因為該算法可以解決原生 RNN 會遇到 gradient vanishing 的問題。一個 LSTM block 有四個 neuron 組成 (圖二(a))，透過控制這四個 neuron 來選擇需要記憶或者遺忘的資訊。GRU 也是該算法的一種變型，不過構造較簡單，由兩個 neuron 組成 (圖二(b))，所需訓練的參數也因此較少，在許多 task 上都能有不錯的表現。這裡我先嘗試使用 LSTM 建構初步的模型。



圖二、Illustration of (a) LSTM and (b) gated recurrent units<sup>1</sup>

```
model = Sequential()
model.add(LSTM(input_shape = (200,108),units=64,return_sequences= True))
model.add(LSTM(units=128,return_sequences= True))
model.add(Dense(512,activation="relu"))
model.add(Dense(49, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

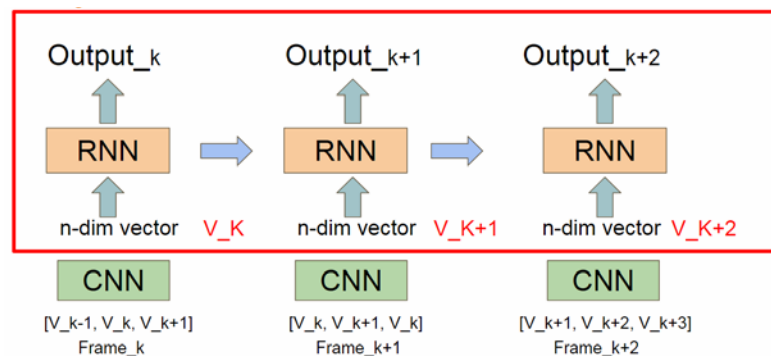
圖三、RNN 程式架構

<sup>1</sup> <https://arxiv.org/pdf/1412.3555.pdf>

## CNN+RNN

- Preprocessing:

CNN+RNN 的架構則是根據助教投影片的說明建構的(如圖四)。因此原本一個 frame 是用  $1 \times 108$  維度的特徵來表示，現在則是  $3 \times 108$  的一個二維陣列來表示。所以 input shape 變成 (batch size, feature size, feature size, number of lookup frame)，在這裡就是 (batch size, 200, 108, 3)。CNN 掃的方向則是與時域垂直，因為時域的特徵在之後的 RNN 中能被學到。



圖四、CNN+RNN 模型 (助教 PPT)

- Model structure:

模型中 CNN 跟 RNN 是 Jointly train，因此我用 Keras 中 TimeDistributed 的 function 讓每一個進到 RNN 中的 sample 的 frame 都個別經過 CNN，以符合圖四中的架構。CNN 層則是用 1D 的 Convolution layer，因為我們只在單一方向掃 feature。之後則是接兩層 RNN 和兩層 fully-connected layers，optimizer 用的一樣是 Adam。

```
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=16, kernel_size=6), input_shape=(200, 108, 3)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(units=64, return_sequences=True))
model.add(LSTM(units=128, return_sequences=True))
model.add(Dense(512, activation="relu"))
model.add(Dense(49, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

圖五、CNN+RNN 程式架構

經過多次實驗，包含調整 filter 數量、kernel\_size 等等，發現這樣的作法對 kaggle 上的表現沒有明顯的提升，Levenshtein distance 約為 12.4871。



## Experimental results and settings

Model	Basic RNN	Basic CNN	RNN+Dropout	CNN+Bi-Lstm+Dropout	Bi-Lstm + Dropout
Levenshtein distance	12.3551	12.3551	11.16098	8.5539	7.904895

表一、各模型表現

- Compare and analyze the results between RNN and CNN  
實作兩種模型，並且多次修改模型層數後發現兩個模型得到的效果類似，(兩種的 RNN 均用 LSTM 的情況下)。有可能是 LSTM 在本次 dataset 上已經能有很強的表現，將 CNN 的效果稀釋了。若將 LSTM 換成 simpleRNN 的話可以發現兩個模型中，CNN+RNN 會有較好的效果，Levenshtein distance 大約差 1。

若將 CNN+RNN 的模型中加上 aggressive 的 Dropout 和使用 Bi-directional LSTM 後，Levenshtein distance 可以降低到 8.5539 (平均)，這也是我在 kaggle 上最後成績。

模型程式碼及架構如下圖：

```
model.add(TimeDistributed(Conv1D(filters=16,
                                   kernel_size=5,
                                   activation="relu"),
                                   input_shape=(frame_count,108,3)))
model.add(TimeDistributed(Flatten()))
model.add(Bidirectional(LSTM(units=128, return_sequences= True)))
model.add(Dropout(0.3))
model.add(Bidirectional(LSTM(units=64, return_sequences= True)))
model.add(Dropout(0.5))
model.add(Bidirectional(GRU(units=32, return_sequences= True)))
model.add(Dropout(0.5))
model.add(Dense(512))
model.add(Dropout(0.5))
model.add(Dense(49, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

圖八、CNN+Bi-Lstm+Dropout

- Compare and analyze the results with other models  
除了上述模型之外，寫 report 的過程中意外發現不加任何 CNN 層也可以取得不錯的效果。用 4 層 LSTM 各帶 0.5 的 Dropout rate 可以拿到約 7.904895 (平均) 的 Levenshtein distance。整體而言，加 Dropout 跟使用 Bi-directional LSTM 在這個 task 上可以有較明顯的進步。而根據個人的實驗中，BatchNormalization 在這個模型中沒有特別的效果，但使用了也不會讓模型弱掉。

```
model = Sequential()
model.add(Bidirectional(LSTM(units=256, return_sequences= True), input_shape = (200,108)))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(units=128, return_sequences= True), input_shape = (200,108)))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(units=64, return_sequences= True)))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(units=32, return_sequences= True)))
model.add(Dropout(0.5))
model.add(Dense(512))
model.add(Dense(49, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

圖九、Bi-Lstm + Dropout