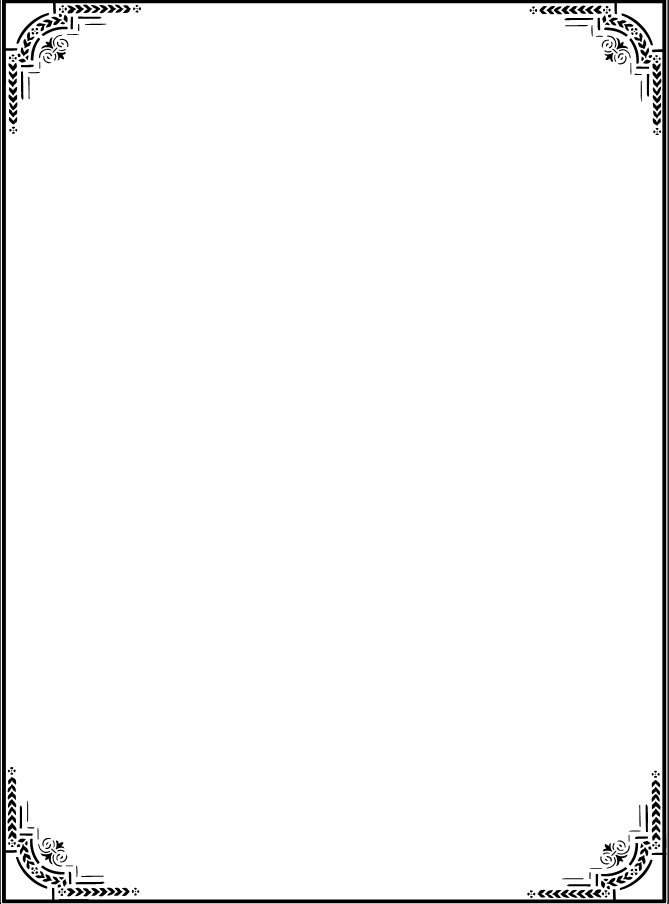
## TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI



**Đồ Án Lập Trình**

***Đề tài:*   
Tìm hiểu RNN và LSTM để gán nhãn file âm thanh**

|  |
| --- |
| Giảng viên hướng dẫn: Nguyễn Tuấn Dũng |
| Sinh viên: Hồ Trung Công-20194237 |
| Lớp : KSCLC HTTT&TT K64 |

Mục Lục

[TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI 1](#_Toc110988185)

[Giảng viên hướng dẫn: Nguyễn Tuấn Dũng 1](#_Toc110988186)

[Sinh viên: Hồ Trung Công-20194237 1](#_Toc110988187)

[Lớp : KSCLC HTTT&TT K64 1](#_Toc110988188)

[**1.** **Tổng quan lý thuyết** 3](#_Toc110988189)

[**1.1.** **Vấn đề gán nhãn âm thanh** 3](#_Toc110988190)

[**1.2.** **Giải quyết vấn đề** 3](#_Toc110988191)

[**1.3.** **Long Short-term Memory** 4](#_Toc110988192)

[*a)* *RNN- Mạng Neural Hồi quy* 4](#_Toc110988193)

[*b)* *Long Short-term Memory: kiến trúc và ý tưởng* 5](#_Toc110988194)

[**2.** **Xây dựng cơ sở dữ liệu cho việc huấn luyện** 6](#_Toc110988195)

[**3.** **Xây dựng mô hình LSTM ( module *models.py*)** 9](#_Toc110988196)

[**3.1.** **Qua các tài liệu tham khảo, em chọn mô hình LSTM như sau:** 9](#_Toc110988197)

[**3.2.** **Huấn luyện mạng ( module *train.py*)** 9](#_Toc110988198)

[*a)* *Data Generator* 9](#_Toc110988199)

[*b)* *Traning* 10](#_Toc110988200)

[**4.** **Test mô hình sau khi đã huấn luyện (module *predict.py*)** 11](#_Toc110988201)

[**5.** **Tài liệu tham khảo** 11](#_Toc110988202)

1. **Tổng quan lý thuyết**
   1. **Vấn đề gán nhãn âm thanh**

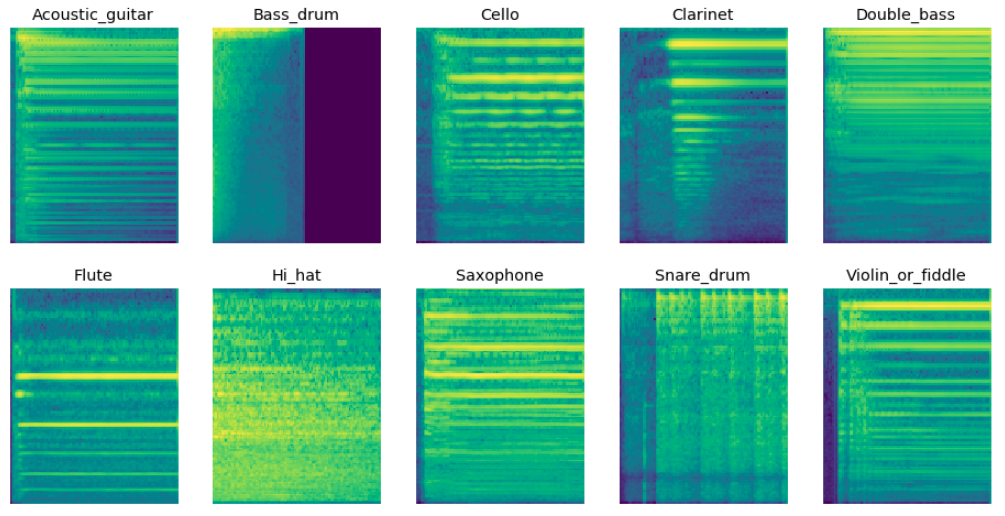
Với sự đa dạng của âm thanh thì việc gán nhãn mất rất nhiều công sức cũng như thời gian. Âm thanh được chia thành nhiều thể loại ví dụ như âm nhạc, tiếng động vật, ….

A picture containing shape

Description automatically generatedTrong project này em lựa chọn thể loại âm thanh là nhạc cụ : guitar, saxophone, … để xây dựng mô hình dự đoán tiếng nhạc cụ sử dụng mô hình học sâu (Deep Learning) là RNN và LTSM.

Hình 1 : Dữ liệu dạng sóng của các nhạc cụ

* 1. **Giải quyết vấn đề**

Để xây dựng mô hình máy học như trên, ta sẽ chuyển dữ liệu dạng sóng của nhạc cụ về dữ liệu dạng quang phổ để huấn luyện

Hình 2 : Dữ liệu dạng quang phổ của các loại nhạc cụ

* 1. **Long Short-term Memory**

1. *RNN- Mạng Neural Hồi quy*

Trong mô hình Recurrent (Hồi quy) - các dự đoán theo theo thời gian hồi quy (recurrent tỉme.

preliction) đều phụ thuộc vào tất cả kết quả từ trước. Thiết kế của mô hình hồi quy nhằm

tận dụng lượng thông tin được học từ trước để làm nền tảng cho các dự đoán hiện tại, không cần

phải được học lại từ số không

Graphical user interface, application, Teams

Description automatically generated

Hình 3: Minh họa mô hình Reeurrent Neural Network

Mô hình toán học của mô hình RNN : Cho chuỗi input ), mô hình RNN chuẩn sẽ tính hidden vector ) bằng các phương trình sau :

)

Với W kí hiệu cho các ma trận weight, b kí hiệu cho vector bias, là hàm activation của hidden layer

RNN có khả năng học và dự đoán dựa trên Long-term Dependencies nhưng thực tế chứng minh ngược lại. Việc tăng số lượng Dependencies dẫn đến việc mất kết nối giữa input và outcome.

A screenshot of a computer

Description automatically generated with low confidence

A picture containing text, clock, gauge

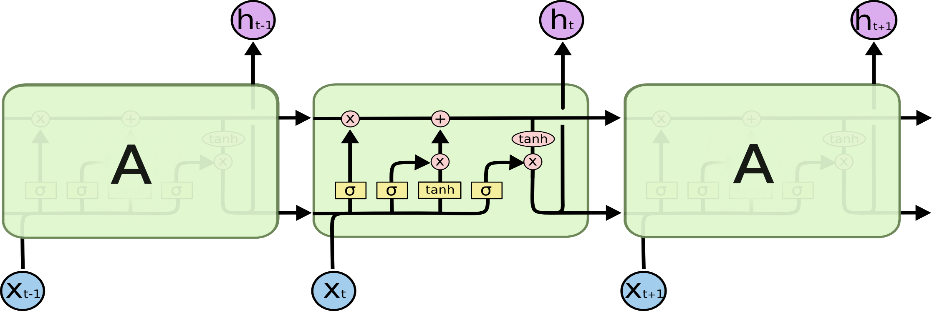
Description automatically generatedHình 5 : Với khoảng cách từ input đến outcome nhỏ, RNN vẫn có khả năng kết nối

Hình 5 : Khi tăng khoảng cách, RNN mất khả năng kết nối thông tin

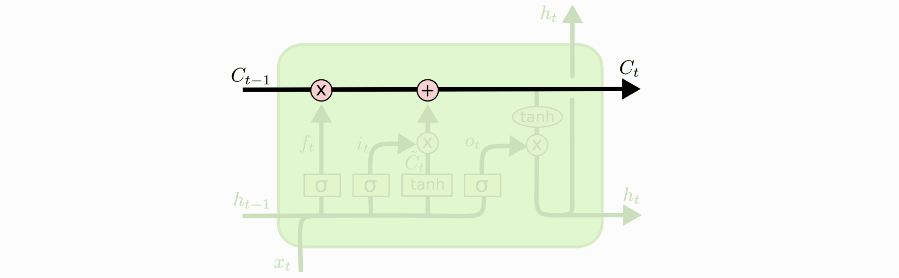
1. *Long Short-term Memory: kiến trúc và ý tưởng*

Mạng bộ nhớ dài-ngắn (Long Short Term Memory networks), thường được gọi là **LSTM** - là một dạng đặc biệt của RNN, nó có khả năng học được các phụ thuộc xa. LSTM được giới thiệu bởi Hochreiter & Schmidhuber (1997), và sau đó đã được cải tiến và phổ biến bởi rất nhiều người trong ngành

LSTM được thiết kế để tránh được vấn đề phụ thuộc xa (long-term dependency). Việc nhớ thông tin trong suốt thời gian dài là đặc tính mặc định của chúng, chứ ta không cần phải huấn luyện nó để có thể nhớ được

LSTM cũng có kiến trúc dạng chuỗi như vậy, nhưng các mô-đun trong nó có cấu trúc khác với mạng RNN chuẩn. Thay vì chỉ có một tầng mạng nơ-ron, chúng có tới 4 tầng tương tác với nhau một cách rất đặc biệt

Hình 6 : Mô hình LSTM

Chìa khóa của LSTM là trạng thái tế bào (cell state) - chính đường chạy thông ngang phía trên của sơ đồ hình vẽ. Trạng thái tế bào là một dạng giống như băng truyền. Nó chạy xuyên suốt tất cả các mắt xích (các nút mạng) và chỉ tương tác tuyến tính đôi chút. Vì vậy mà các thông tin có thể dễ dàng truyền đi thông suốt mà không sợ bị thay đổi

Hình 7: trạng thái tế bào (cell state)

* Forget gate layer(Tầng cổng quên):

Diagram

Description automatically generated

* Input gate layer( Tầng cổng vào) và *tanh*:

Diagram

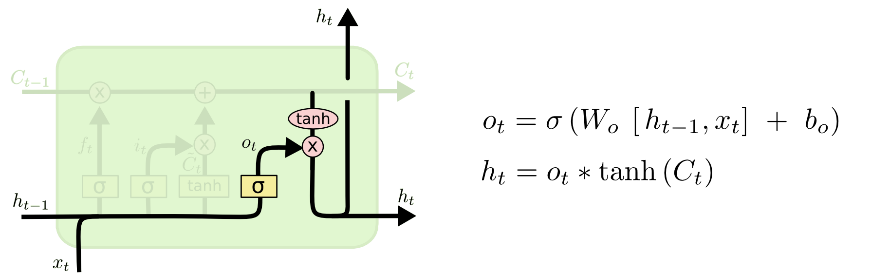
Description automatically generated

* Cập nhập trạng thái tế bào:

Diagram

Description automatically generated with medium confidence

* Quyết định đầu ra:



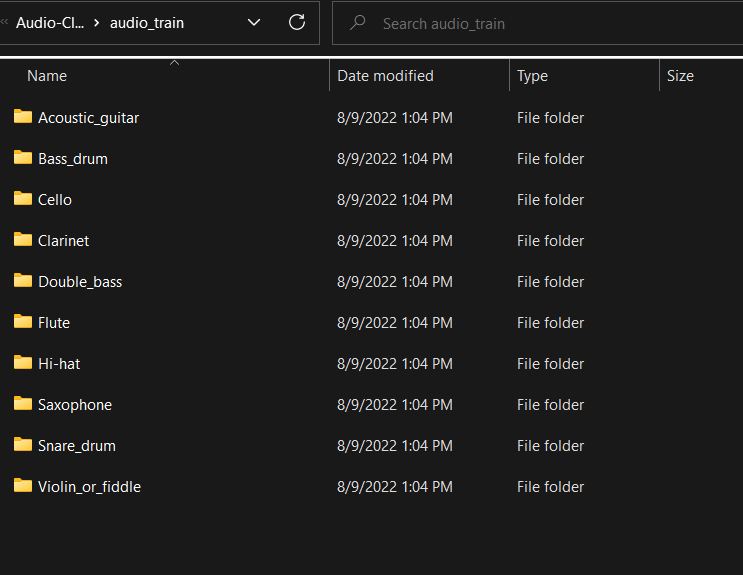
1. **Xây dựng cơ sở dữ liệu cho việc huấn luyện**

Dữ liệu các nhạc cụ được lấy từ trang [kaggle](https://www.kaggle.com/c/freesound-audio-tagging) gồm audio\_train, audio\_test, train.csv. Sau đó, quá trình huấn luyện mạng được thực hiện. Kết th́úc quá trình này, mạng nơ-ron có thể phân loại các loại nhạc cụ khác nhau, từ đó có thể nhận dạng được các âm thanh đã học mà không cần đến không gian dữ liệu mẫu nữa.

**Tiền xử lý dữ liệu(module : *clean.py*)**

Sau khi đã tải tệp audio\_train và audio\_test, vì tệp có dung lượng lớn nên em chỉ giữ lại 436 file của 10 loại nhạc cụ là :

['Acoustic\_guitar', 'Bass\_drum', 'Cello', 'Clarinet', 'Double\_bass', 'Flute', 'Hi-hat', 'Saxophone', 'Snare\_drum', 'Violin\_or\_fiddle']

Và đưa từng file vào từng thư mục riêng từ label của file train.csv

Sau khi đã có file âm thanh, chúng ta cần được xử lý để hạn chế nhiễu.

Mỗi file âm thanh này có :

* Độ dài : 16000 mẫu/1 giây
* Tần số mẫu : 16kHz

src, sr = librosa.load('clean/Cello/0a98104d\_0.wav', sr=None, mono=True)  
print('Audio length: %d samples, %04.2f seconds. \n' % (len(src), len(src) / sr) +  
 'Audio sample rate: %d Hz' % sr)  
dt = 1.0  
\_src = src[:int(sr\*dt)]  
src = np.expand\_dims(\_src, axis=1)  
input\_shape = src.shape  
print(input\_shape)

Audio length: 16000 samples, 1.00 seconds.

Audio sample rate: 16000 Hz

(16000, 1)

Vì vậy chúng ta có tần số lấy mẫu là 16kHz và thời gian lấy mẫu là 1(s)

Trong module clean.py :

Hàm *envelope* dùng để loại bỏ những dữ liệu ở cường độ thấp

def envelope(y, rate, threshold):  
 mask = []  
 y = pd.Series(y).apply(np.abs)  
 y\_mean = y.rolling(window=int(rate / 20),  
 min\_periods=1,  
 center=True).max()  
 for mean in y\_mean:  
 if mean > threshold:  
 mask.append(True)  
 else:  
 mask.append(False)  
 return mask

Hàm *split\_wavs* dùng để đưa dữ liệu về dạng mảng với thư viện librosa

signal, rate = librosa.load(src\_fn,sr)

#scr\_fn là đường dẫn file và sr là tần số lấy mẫu  
mask = envelope(signal, rate, threshold=threshold)  
signal = signal[mask]

Hàm *save\_sample* dùng để ghi dữ liệu sau khi đã xử lí

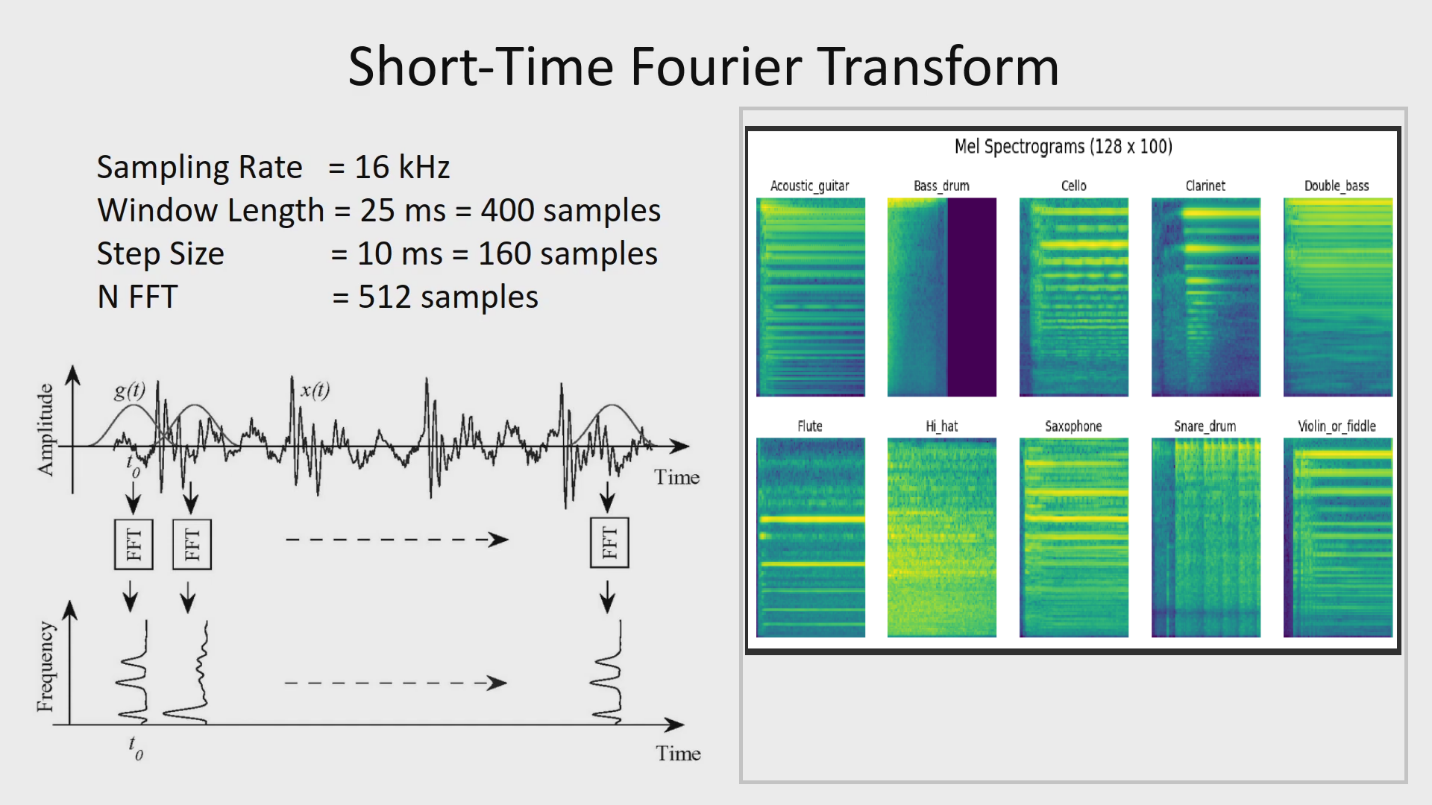
def save\_sample(sample, rate, target\_dir, fn, ix):  
 fn = fn.split('.wav')[0]  
 dst\_path = os.path.join(target\_dir.split('.')[0], fn + '\_{}.wav'.format(str(ix)))  
 if os.path.exists(dst\_path):  
 return  
 wavfile.write(dst\_path, rate, sample)

Do tín hiệu file nhạc là tín hiệu không dừng, nên không thể áp dụng phép phân tích Fourier thông thường. Nhưng nếu chúng ta chia tín hiệu tiếng nói ra thành từng đoạn đủ nhỏ theo thời gian, thì tín hiệu tiếng nói trong mỗi đoạn có thể xem là tín hiệu dừng, và do đó có thể lấy biến đổi Fourier trên từng đoạn tín hiệu này. Ðây là nguyên lý của phép biến đổi Fourier thời gian ngắn, còn gọi là biến đổi Fourier cửa sổ hóa (STFT)

Trong STFT, tín hiệu cần phân tích f(t) đầu tiên được nhân với một hàm cửa sổ w(t-τ) để lấy được tín hiệu trong một khoảng thời gian ngắn xung quanh thời điểm τ. Sau đó phép biến đổi Fourier bình thường được tính trên đoạn tín hiệu này. Kết quả ta được một hàm theo tần số và thời gian STFT f(ω,τ) xác định bởi (dấu (\*) ký hiệu cho thành phần liên hợp phức:

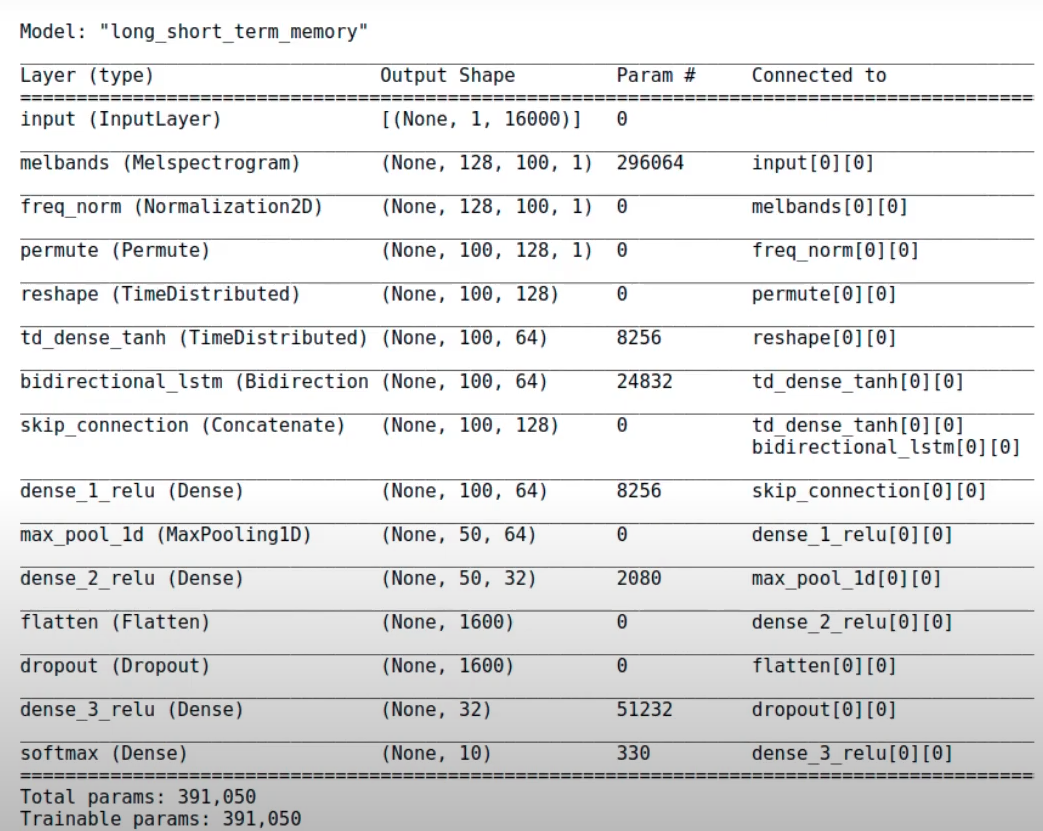
STFT tại thời điểm τ được xem là phổ cục bộ của f(t) xung quanh thời điểm τ, do cửa sổ tương đối ngắn làm triệt tiêu tín hiệu ngoài vùng lân cận. Vì vậy STFT có tính định vị theo thời gian. Cửa sổ phân tích càng hẹp thì sự định vị này càng tốt. Ðể thấy rõ STFT cũng định vị trong miền tần số, ta có thể áp dụng định lý Parserval:

Kết quả phân trích đặc trưng của từ file âm thanh được minh họa trên hình 8 . Sau khi rời rạc hóa các formant tại 100 thời điểm trên trục thời gian và 128 vị trí trên trục tần số



Hình 8 : Kết quả phân tích

1. **Xây dựng mô hình LSTM ( module *models.py*)**
   1. **Qua các tài liệu tham khảo, em chọn mô hình LSTM như sau:**

****

Hình 9: Mô hình mạng LSTM

* 1. **Huấn luyện mạng ( module *train.py*)**

Để huấn luyện mạng thì chúng tá sẽ sử dụng thư viện tensorflow, keras và sklearn trong python

Đầu tiên chúng ta sẽ chia nhỏ dữ liệu để load từng phần trong quá trình train với lớp Data Generator

1. *Data Generator*

* Class :

class DataGenerator(tf.keras.utils.Sequence):  
 def \_\_init\_\_(self, wav\_paths, labels, sr, dt, n\_classes,  
 batch\_size=32, shuffle=True):  
 self.wav\_paths = wav\_paths #thư mục file  
 self.labels = labels # nhãn của âm thanh  
 self.sr = sr # tần số lấy mẫu  
 self.dt = dt # thời gian lấy mẫu  
 self.n\_classes = n\_classes # số lượng nhãn  
 self.batch\_size = batch\_size # số lượng mẫu/1 lần huấn luyện  
 self.shuffle = True  
 self.on\_epoch\_end()

* Hàm \_\_len\_\_: Trả về số lượng batch trên 1 epoch

def \_\_len\_\_(self):  
 return int(np.floor(len(self.wav\_paths) / self.batch\_size))

* Hàm \_\_getitem\_\_(): Hàm này sẽ tạo từng batch cho data theo thứ tự được truyền vào, xử lý dữ liệu và trả về dữ liệu để training

def \_\_getitem\_\_(self, index):  
 indexes = self.indexes[index\*self.batch\_size:(index+1)\*self.batch\_size]  
  
 wav\_paths = [self.wav\_paths[k] for k in indexes]  
 labels = [self.labels[k] for k in indexes]  
  
 # generate a batch of time data  
 X = np.empty((self.batch\_size, int(self.sr\*self.dt), 1),dtype=np.float32)  
 Y = np.empty((self.batch\_size, self.n\_classes), dtype=np.float32)  
  
 for i, (path, label) in enumerate(zip(wav\_paths, labels)):  
 rate, wav = wavfile.read(path)  
 X[i,] = wav.reshape(-1, 1)  
 Y[i,] = keras.utils.np\_utils.to\_categorical(label, num\_classes=self.n\_classes)  
  
 return X, Y

1. *Traning*

Hàm train() : huấn luyện và lưu mô hình

* Tần số lấy mẫu 16kHz
* thời gian lấy mẫu 1(s)
* số lượng mẫu/ lần huấn luyện 16

Chúng ta sẽ chuyển 10 nhãn âm thanh thành 1 mảng và dùng module train\_test\_split để chia tập dữ liệu thành tập huấn luyện và tập thử nghiệm

classes = sorted(os.listdir(direct))  
le = LabelEncoder()  
le.fit(classes)  
labels = [os.path.split(x)[0].split('/')[-1] for x in wav\_paths]  
labels = le.transform(labels)  
wav\_train, wav\_val, label\_train, label\_val = train\_test\_split(wav\_paths,labels,test\_size=0.1,random\_state=0)

Sau đó chúng ta sẽ dùng module Modelcheckpoint để lưu file weight và fit model

tg = DataGenerator(wav\_train, label\_train, sr, dt,  
 params['N\_CLASSES'], batch\_size=batch\_size)   
vg = DataGenerator(wav\_val, label\_val, sr, dt,  
 params['N\_CLASSES'], batch\_size=batch\_size)  
model = models[model\_type]  
cp = ModelCheckpoint('models/{}.h5'.format(model\_type), monitor='val\_loss',  
 save\_best\_only=True, save\_weights\_only=False,  
 mode='auto', save\_freq='epoch', verbose=1)  
csv\_logger = CSVLogger(csv\_path, append=False)  
model.fit(tg, validation\_data=vg,  
 epochs=30, verbose=1,  
 callbacks=[csv\_logger, cp])

1. **Test mô hình sau khi đã huấn luyện (module *predict.py*)**

Sau khi đã có file weight chúng ta sẽ load model để Test dữ liệu

model = load\_model(modelFile, #địa chỉ file wieght  
 custom\_objects={'STFT':STFT,   
 'Magnitude':Magnitude,  
 'ApplyFilterbank':ApplyFilterbank,  
 'MagnitudeToDecibel':MagnitudeToDecibel})

Sau khi thử nghiệm với 188 file kết quả được ghi vào file Pre.csv

Hàm accuracy\_score cho thấy mô hình đã dự đoán đúng xấp xỉ 75%

Vì số lượng file huấn luyện ít nên mô hình có độ chính xác không cao.

Nếu có thêm file để huấn luyện thì tỉ lệ sẽ cao hơn

Và nếu test số lượng file lớn thì tỉ lệ dự đoán đúng sẽ giảm đi

Text

Description automatically generated

**Link github :** [**https://github.com/Htcong10/LSTM-Audio-Classification**](https://github.com/Htcong10/LSTM-Audio-Classification)

1. **Tài liệu tham khảo**

# Keras Callbacks: <https://miai.vn/2020/09/05/keras-callbacks-tro-thu-dac-luc-khi-train-models/>

LSTM : <https://dominhhai.github.io/vi/2017/10/what-is-lstm/>

Spectrograms : <https://towardsdatascience.com/audio-deep-learning-made-simple-part-2-why-mel-spectrograms-perform-better-aad889a93505>

STFT: <https://by.com.vn/vz9eg2>

https://towardsdatascience.com/audio-deep-learning-made-simple-part-3-data-preparation-and-augmentation-24c6e1f6b52

Data: <https://www.kaggle.com/c/freesound-audio-tagging>

RNN for Audio Classification :

<https://towardsdatascience.com/recurrent-neural-nets-for-audio-classification-81cb62327990>

ML : <https://www.phamduytung.com/blog/2018-10-02-understanding-epoch-batchsize-iterations/>

Data Generator : <https://viblo.asia/p/data-generator-voi-keras-3P0lPqQg5ox>

…