

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG  
KHOA CÔNG NGHỆ THÔNG TIN I



**ĐỒ ÁN TỐT NGHIỆP**  
**ĐỀ TÀI**  
**NGHIÊN CỨU MẠNG CNN VÀ THUẬT**  
**TOÁN SVM ÁP DỤNG TRONG NHẬN**  
**DẠNG TIẾNG HÓ**

Giảng viên hướng dẫn: TS. NGUYỄN TRỌNG KHÁNH  
Sinh viên thực hiện: CHU QUẾ PHƯƠNG  
Mã sinh viên: B15DCCN420  
Lớp: D15HTTT01  
Khoá: 2015 - 2020  
Hệ: ĐẠI HỌC CHÍNH QUY

Hà Nội – 12/2019

### LỜI CẢM ƠN

Lời cảm ơn đầu tiên em xin gửi tới người thầy và cũng là người hướng dẫn của em, TS. Nguyễn Trọng Khánh vì sự định hướng và giúp đỡ tận tình của thầy trong quá trình thực hiện đồ án. Lĩnh vực học máy là một lĩnh vực hoàn toàn mới đối với em nhưng nhờ có sự hướng dẫn và góp ý của thầy, em đã giải quyết được những vấn đề và khó khăn để hoàn thiện được đồ án này. Toàn bộ quá trình làm việc với thầy đã giúp em có thể những kinh nghiệm và kiến thức quý báu trong lĩnh vực mới đối với bản thân em.

Em cũng xin được gửi lời cảm ơn tới các thầy cô trong Học viện và trong Khoa Công nghệ thông tin I đã dạy bảo và truyền đạt những kiến thức, kinh nghiệm quý giá trong suốt 4 năm qua.

Đồng thời em cũng muốn gửi lời cảm ơn tới bố mẹ và chị đã luôn ủng hộ em với những quyết định của mình. Sự tin tưởng và tình yêu từ gia đình chính là động lực lớn lao nhất đối với em để vượt qua những khó khăn và trở ngại.

Em cũng xin gửi lời cảm ơn tới những người bạn đã quan tâm và giúp đỡ em trong suốt quá trình học tập và làm đồ án. Nhờ có các bạn mà quãng đời sinh viên của em đã trở thành quãng thời gian đáng nhớ và học hỏi được nhiều điều.

Hà Nội, tháng 12 năm 2019

**Sinh viên**

**Chu Quế Phương**

# **NHẬN XÉT, ĐÁNH GIÁ, CHO ĐIỂM (Của giảng viên hướng dẫn)**

**Điểm:** .....(bằng chữ: .....)  
**Đồng ý/Không đồng ý** cho sinh viên bảo vệ trước hội đồng chấm tốt nghiệp.

Hà Nội, tháng 12 năm 2019

# CÁN BỘ - GIẢNG VIÊN HƯỚNG DẪN

# NGUYỄN TRỌNG KHÁNH

## **NHẬN XÉT, ĐÁNH GIÁ, CHO ĐIỂM (Của giảng viên phản biện)**

Điểm: .....(bằng chữ: .....)

**Đồng ý/Không đồng ý** cho sinh viên bảo vệ trước hội đồng chấm tốt nghiệp.

Hà Nội, tháng 12 năm 2019

## CÁN BỘ - GIẢNG VIÊN PHẢN BIỆN

## MỤC LỤC

LỜI CẢM ƠN.....	i
MỤC LỤC .....	i
DANH MỤC HÌNH ẢNH.....	iii
DANH MỤC BẢNG BIÊU.....	v
DANH MỤC TỪ VIẾT TẮT .....	vi
LỜI MỞ ĐẦU .....	1
1. CHƯƠNG I: BÀI TOÁN NHẬN DIỆN TỰ ĐỘNG TIẾNG HO .....	3
1.1. Bài toán nhận diện tự động tiếng ho .....	3
1.2. Một số giải pháp hiện có cho bài toán nhận diện tiếng ho.....	4
1.2.1. Hull Automatic Cough Counter (HACC).....	4
1.2.2. DeepCough.....	5
1.3. Giải pháp đề xuất cho bài toán nhận diện tiếng ho .....	8
1.3.1. Đặc trưng logarit phổ tần số mel.....	8
1.3.2. Mạng nơ ron tích chập VGGish .....	9
1.3.3. Thuật toán SVM .....	10
1.4. Kết luận .....	11
2. CHƯƠNG II: NHẬN DIỆN TIẾNG HO SỬ DỤNG MẠNG NƠ RON TÍCH CHẬP VÀ THUẬT TOÁN SVM .....	12
2.1. Biểu diễn dữ liệu bằng đặc trưng logarit của phổ tần số mel .....	12
2.2. Giới thiệu về mạng CNN VGGish .....	16
2.2.1. Mạng nơ ron tích chập .....	16
2.2.2. Mạng nơ ron tích chập VGGish .....	24
2.3. Thuật toán phân loại SVM .....	28
2.4. Kết luận .....	38
3. CHƯƠNG III: THỬ NGHIỆM VÀ ĐÁNH GIÁ .....	39
3.1. Kịch bản thử nghiệm.....	39
3.1.1. Tiền xử lý dữ liệu .....	41
3.1.2. Trích chọn, lưu trữ đặc trưng .....	42
3.1.3. Huấn luyện và đánh giá hiệu quả thuật toán phân loại .....	46

3.2. Cài đặt thử nghiệm .....	50
3.2.2. Cài đặt trích chọn đặc trưng .....	52
3.2.3. Cài đặt huấn luyện và sử dụng thuật toán phân loại .....	57
3.3. Kết quả thu được .....	58
3.3.1. Kết quả thử nghiệm .....	58
3.3.2. Đánh giá kết quả thu được .....	61
3.4. Chương trình ứng dụng mô hình phân loại thu được .....	62
3.5. Kết luận .....	64
KẾT LUẬN .....	65
TÀI LIỆU THAM KHẢO .....	66

**DANH MỤC HÌNH ẢNH**

Hình 1. 1: Quy trình thường thấy trong việc phân loại tiếng ho .....	4
Hình 1. 2: Kết quả số tiếng ho đếm được .....	5
Hình 1. 3: Cấu trúc của mạng CNN sử dụng cho DeepCough.....	6
Hình 1. 4: Kết quả của thí nghiệm 1 .....	7
Hình 1. 5: Kết quả của thí nghiệm 2 .....	7
Hình 1. 6: Đồ thị của hàm giá trị cao độ theo mel theo biến là tần số .....	8
Hình 1. 7: Minh họa mạng VGG16 .....	10
Hình 1. 8: Minh họa đường biên ra quyết định mà một SVM cài đặt bằng thư viện scikit-learn, trên bộ dữ liệu hoa diên vĩ (iris) .....	11
Hình 2. 1: Sơ đồ quá trình tính logarit phổ tần số .....	12
Hình 2. 2: Sơ đồ quá trình tính logarit phổ tần số mel .....	12
Hình 2. 3: Bảng lọc mel theo công thức (2.7) .....	16
Hình 2. 4: Ví dụ một ANN có hai lớp .....	17
Hình 2. 5: Một số hàm kích hoạt thường được sử dụng .....	18
Hình 2. 6: Ví dụ mảng hai chiều các nơ ron ở lớp đầu vào của CNN .....	20
Hình 2. 7: Ví dụ tương ứng từ một vùng trên đầu vào hai chiều với một giá trị trên bản đồ kích hoạt .....	21
Hình 2. 8: Ví dụ tương ứng một vùng 2x2 trên bản đồ đặc trưng với một đơn vị ở lớp gộp .....	22
Hình 2. 9: Ví dụ hoạt động ở lớp gộp với bộ lọc 2x2 và hàm cực đại .....	23
Hình 2. 10: Ví dụ đầu ra của lớp gộp với đầu vào là bản đồ đặc trưng 24x24x3.....	23
Hình 2. 11: Cấu trúc các mạng VGG.....	25
Hình 2. 12: Minh họa cấu trúc mạng VGG16 .....	26
Hình 2. 13: Minh họa cấu trúc VGGish .....	26
Hình 2. 14: Ví dụ kết quả của nhúng một số từ trong tiếng Anh (word embedding)....	27
Hình 2. 15: Mảng các giá trị của đặc trưng ở đầu ra mạng VGGish cho một file âm thanh trong bộ dữ liệu của đồ án .....	28
Hình 2. 16: Các siêu phẳng có thể phân chia dữ liệu có hai lớp .....	28
Hình 2. 17: Hai đường biên quyết định với lề khác nhau .....	29
Hình 2. 18: Ví dụ về đường biên quyết định và lề có độ rộng d .....	30
Hình 2. 19: Dữ liệu có một điểm nhiễu nằm quá gần lớp còn lại .....	33
Hình 2. 20: Dữ liệu có các điểm thuộc hai lớp xen kẽ nhau .....	33
Hình 2. 21: Một bộ dữ liệu có đường biên quyết định không tuyến tính .....	36
Hình 2. 22: Bộ dữ liệu trên sau khi chuyển về miền không gian khác bằng phép biến đổi $\Phi$ : $(x_1, x^2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$ .....	36
Hình 3. 1: Tổ chức các lớp phân loại của AudioSet ở hai mức đầu tiên .....	40
Hình 3. 2: Ví dụ về nội dung một tf.Example .....	44

Hình 3. 3: Ví dụ nội dung một SequenceExample .....	45
Hình 3. 4: Ví dụ nội dung của một bản ghi (tương ứng một video) trong AudioSet theo định dạng của tf.SequenceExample.....	46
Hình 3. 5: Ví dụ khi các phần dữ liệu được chia làm 5 phần, màu xanh lá là phần dữ liệu huấn luyện, phần xanh dương là phần dữ liệu kiểm chứng.....	48
Hình 3. 6: Ví dụ lưới tìm kiếm với hai hệ số, giá trị các hệ số gồm (10, 20, 30, 40, 50, 60, 70, 80, 90, 100) và (3, 5, 7, 9, 11, 13, 15) .....	49
Hình 3. 7: Quy trình huấn luyện và đánh giá hoạt động của mô hình SVM trong đồ án .....	50
Hình 3. 8: Ví dụ một đoạn code sử dụng TensorFlow để xây dựng và thực thi một sơ đồ tính hàm ReLu viết bằng Python .....	51
Hình 3. 9: Minh họa sơ đồ được tạo bởi đoạn code ở Hình 3.7 .....	52
Hình 3. 10: Phần code cho tiền xử lý cung cấp bởi nhóm phát triển AudioSet .....	53
Hình 3. 11: Định nghĩa mô hình VGGish theo kiến trúc đã mô tả ở phần 2.2 .....	54
Hình 3. 12: Mô tả việc sử dụng mạng VGGish trích thuỷ tinh từ example_batch là mảng thu được sau khi tiền xử lý .....	55
Hình 3. 13: Phần code cho quá trình hậu xử lý đặc trưng cung cấp bởi nhóm phát triển AudioSet .....	55
Hình 3. 14: Phần code cho việc đọc file TFRecord.....	56
Hình 3. 15: Phần code ghi các đặc trưng trích bởi VGGish ra file TFRecord với tên được chỉ định sẵn, writer là một object thuộc tf.python_io.TFRecordWrite dùng để ghi với tên file đó.....	57
Hình 3. 16: Phần code cho quá trình huấn luyện mô hình SVM với kiểm chứng chéo (cross validation - cv) và tìm kiếm theo lưới.....	58
Hình 3. 17: Giao diện chính của chương trình để chọn nguồn âm thanh sẽ phân loại..	63
Hình 3. 18: Giao diện chương trình khi chọn file có sẵn .....	63
Hình 3. 19: Giao diện hiện kết quả phân loại file.....	63
Hình 3. 20: Giao diện chương trình khi chọn ghi âm.....	64

**DANH MỤC BẢNG BIỂU**

Bảng 3. 1: Môi trường cài đặt thử nghiệm của đồ án .....	50
Bảng 3. 2: Confusion matrix cho mô hình huấn luyện với dữ liệu tự trích khi nhận diện tiếng ho trên bộ dữ liệu test .....	59
Bảng 3. 3: Giá trị các tham số đánh giá hoạt động của mô hình huấn luyện với dữ liệu tự trích khi nhận diện tiếng ho trên bộ dữ liệu test.....	59
Bảng 3. 4: Confusion matrix cho mô hình huấn luyện với dữ liệu tổng hợp khi nhận diện tiếng ho trên bộ dữ liệu test .....	60
Bảng 3. 5: Giá trị các tham số đánh giá hoạt động của mô hình huấn luyện với dữ liệu tổng hợp khi nhận diện tiếng ho trên bộ dữ liệu test .....	60
Bảng 3. 6: Confusion matrix cho mô hình huấn luyện với dữ liệu tự trích khi nhận diện tiếng ho trên một phần dữ liệu từ bộ dữ liệu FSDKaggle2018 .....	60
Bảng 3. 7: Giá trị các tham số đánh giá hoạt động của mô hình huấn luyện với dữ liệu tự trích khi nhận diện tiếng ho trên một phần dữ liệu từ bộ dữ liệu FSDKaggle2018..	60
Bảng 3. 8: Confusion matrix cho mô hình huấn luyện với dữ liệu tổng hợp khi nhận diện tiếng ho trên một phần dữ liệu từ bộ dữ liệu FSDKaggle2018 .....	61
Bảng 3. 9: Giá trị các tham số đánh giá hoạt động của mô hình huấn luyện với dữ liệu tổng hợp khi nhận diện tiếng ho trên một phần dữ liệu từ bộ dữ liệu FSDKaggle2018 .....	61
Bảng 3. 10: Môi trường cài đặt chương trình ứng dụng mô hình SVM tổng hợp .....	62

**DANH MỤC TỪ VIẾT TẮT**

CNN: Convolutional Neural Network	Mạng nơ ron tích chập
MFCC: Mel-frequency Cepstral Coefficients	Bộ tham số cepstrum tần số mel
kNN: k-Nearest Neighbors	K láng giềng gần nhất
LPC: Linear Predictive Coding	Mã hóa dự đoán tuyến tính
HACC: Hull Automatic Cough Counter	Máy đếm tiếng ho tự động
PNN: Probabilistic Neural Network	Mạng nơ ron dựa trên xác suất
PCA: Principal Component Analysis	Phân tích thành phần chính
STFT: Short Time Fourier Transform	Biến đổi Fourier thời gian ngắn
SVM: Support Vector Machine	Máy vectơ hỗ trợ
HMM: Hidden Markov Model	Mô hình Markov ẩn
ROC: Receiver Operating Characteristic	Đường đồ thị thể hiện khả năng nhận diện của thuật toán phân loại bằng giá trị true positive rate và false positive rate
DFT: Discrete Fourier Transform	Biến đổi Fourier rời rạc
FFT: Fast Fourier Transform	Biến đổi Fourier nhanh
ANN: Artificial Neural Network	Mạng nơ ron nhân tạo
KKT: Karush-Kuhn-Tucker	Hệ điều kiện của bài toán tối ưu
HACC: Hull Automatic Cough Counter	Máy đếm tiếng ho Hull tự động
FSDKaggle2018: Freesound Dataset Kaggle 2018	Bộ dữ liệu Freesound sử dụng trên Kaggle vào 2018
DCASE: Detection and Classification of Acoustic Scenes and Events	Nhận diện và phân loại tự động âm thanh bối cảnh và âm thanh chủ đạo
ReLU: Rectified linear unit	Hàm nắn tuyến tính
LRN: Local Response Normalisation	Phương pháp chuẩn hóa phản ứng cục bộ
SMO: Sequential minimal optimization	Thuật toán tuần tự hóa tối ưu cực tiểu

## LỜI MỞ ĐẦU

Vấn đề sức khỏe đã luôn là một vấn đề được quan tâm vì nó liên hệ trực tiếp đến đời sống của mỗi người. Sức khoẻ hô hấp cũng không phải là một ngoại lệ vì đây là cơ quan rất dễ bị ảnh hưởng và mọi vấn đề liên quan tới hệ hô hấp đều rất dễ dàng nhận thấy. Một trong những triệu chứng dễ nhận thấy của các bệnh hô hấp là ho. Các bệnh về hô hấp có thể gây ra đờm, viêm hoặc kích thích trong đường hô hấp, dẫn đến ho như là phản ứng cho các vấn đề đó và ho cũng có thể là biểu hiện cho một số bệnh như cảm, cúm, viêm họng hay thậm chí là hen, lao, viêm phổi. Do đó việc nhận diện được tiếng ho và phân tích các đặc điểm, thông tin về nó sẽ có ích trong việc chẩn đoán các bệnh hô hấp. Tuy nhiên, thường thì các thông tin về tiếng ho là do bệnh nhân cung cấp và nó có thể bị sai lệch. Với trường hợp có bác sĩ theo dõi được tiếng ho thì việc chẩn đoán cũng vẫn phụ thuộc nhiều vào chủ quan của bác sĩ. Xuất phát từ những khó khăn trên, yêu cầu về các giải pháp nhận dạng tự động và phân tích tiếng ho đã hình thành. Các nhà nghiên cứu đã và đang phát triển các giải pháp nhận dạng tự động và phân tích các âm thanh hô hấp, bao gồm cả tiếng ho, với sự hỗ trợ của các công nghệ xử lý, lưu trữ âm thanh và các hệ thống dựa trên tri thức. Các giải pháp này sẽ hỗ trợ cho việc chẩn đoán ở điểm là giảm sự phụ thuộc vào chuyên môn y tế, mang tính khách quan và có thể đưa ra các số liệu rõ ràng làm cơ sở chẩn đoán. Những giải pháp nhận dạng và phân tích âm thanh hô hấp tự động này cũng có thể ứng dụng vào việc giám sát sức khỏe ở bệnh viện hoặc tại gia cho các bệnh nhân.

Bên cạnh vấn đề về thu, giám sát âm thanh hay khử nhiễu, một trong những vấn đề của hệ thống nhận dạng tiếng ho là nó cần phải chính xác, cần bắt được tiếng ho mỗi khi nó xuất hiện và đồng thời vẫn không nhầm tiếng ho với các âm thanh khác. Để xử lý vấn đề này, các nhà nghiên cứu có cách tiếp cận là xác định các thuộc tính có ý nghĩa trong việc phân loại tiếng ho với những âm thanh không phải ho. Những đặc trưng như vậy ngoài được lựa chọn thủ công còn có thể được chọn ra thông qua mạng học sâu (deep neural network). Sau đó việc nhận dạng có thể được thực hiện bằng cách sử dụng các thuật toán phân loại để biết âm thanh là tiếng ho hay không.

Đồ án này sẽ tập trung vào việc nhận dạng tự động tiếng ho dựa trên dữ liệu âm thanh ho có sẵn đã khử nhiễu. Để phục vụ cho mục đích này, các file âm thanh sẽ được trích đặc trưng logarit phổ tần số mel và đưa vào mạng nơ ron tích chập (convolutional neural network – CNN) có tên VGGish đã được huấn luyện sẵn. Đặc trưng thu được từ file âm thanh sẽ được đưa vào thuật toán phân loại để học và sau đó phân loại âm thanh ho và không ho. Trong phạm vi đồ án, thuật toán phân loại được sử dụng là máy vectơ hỗ trợ (SVM - Support Vector Machine). Kết quả nhận dạng thu được sẽ được ghi lại để đánh giá hiệu quả của mô hình phân loại SVM.

Cấu trúc của đồ án sẽ gồm 3 phần: Chương 1 sẽ trình bày về bài toán nhận dạng tiếng ho, các giải pháp cho bài toán này hiện có trên thế giới và để xuất giải pháp của đồ án. Chương 2 sẽ trình bày về đặc trưng trích được từ tiếng ho được sử dụng bao

gồm đặc trưng logarit phổ tần số mel và đặc trưng được trích bởi mạng CNN, cụ thể là bởi mạng VGGish, và cuối cùng là thuật toán phân loại SVM. Chương 3 sẽ trình bày về kịch bản thử nghiệm, chi tiết việc cài đặt, kết quả thu được, đánh giá kết quả đó và chương trình ứng dụng mô hình phân loại thu được.

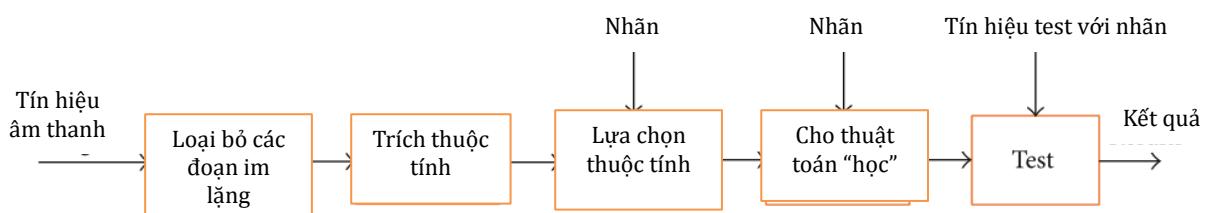
## CHƯƠNG I: BÀI TOÁN NHẬN DIỆN TỰ ĐỘNG TIẾNG HO

Trong chương 1, đồ án sẽ trình bày sự hình thành bài toán nhận dạng tự động tiếng ho và một số giải pháp áp dụng cho bài toán này hiện đang có trên thế giới

### 1.1. Bài toán nhận diện tự động tiếng ho

Ho là một trong những triệu chứng dễ nhận biết của nhiều bệnh về đường hô hấp như cảm, cúm, viêm họng hay thậm chí là hen, lao, viêm phổi [1]. Do đó việc nhận diện tiếng ho và phân tích các đặc điểm, thông tin về nó cũng được dùng trong việc chẩn đoán các bệnh hô hấp. Tuy nhiên, thông thường thì các thông tin về tiếng ho là do bệnh nhân cung cấp và nó có thể bị sai lệch do bệnh nhân tự nghĩ ra thông tin hoặc do ghi nhớ các thông tin không chính xác [1]. Với trường hợp được bác sĩ theo dõi thì việc chẩn đoán dựa trên tiếng ho cũng vẫn phụ thuộc nhiều vào chủ quan của bác sĩ, cụ thể là chuyên môn và kinh nghiệm để nhận diện được tiếng ho là kiểu gì và có liên quan tới bệnh nào. Những khó khăn trên đã hình thành yêu cầu cần có giải pháp nhận dạng tự động và phân tích tiếng ho. Các nhà nghiên cứu đã và đang phát triển các giải pháp nhận dạng tự động và phân tích các âm thanh hô hấp, bao gồm cả tiếng ho, với sự hỗ trợ của các công nghệ xử lý, lưu trữ âm thanh và các hệ thống dựa trên tri thức. Các giải pháp này sẽ hỗ trợ cho việc chẩn đoán ở điểm là giảm sự phụ thuộc vào chuyên môn y tế, mang tính khách quan và có thể đưa ra các số liệu rõ ràng làm cơ sở chẩn đoán. Những giải pháp nhận dạng và phân tích âm thanh hô hấp tự động này cũng có thể ứng dụng vào việc giám sát sức khỏe ở bệnh viện hoặc tại gia cho các bệnh nhân.

Quá trình phân loại/nhận dạng nói chung bao gồm các bước: tiền xử lý dữ liệu, trích thuộc tính dữ liệu huấn luyện, chọn thuộc tính phù hợp, cho thuật toán “học” bằng dữ liệu đó và đưa dữ liệu thử nghiệm vào để kiểm tra việc phân loại. Việc nhận dạng tiếng ho, tức là phân loại tiếng ho thành hai lớp, một là tiếng ho, hai là không phải ho, cũng có thể được thực hiện theo quy trình này. Trong quá trình này, ta có thể nhận thấy việc lựa chọn thuộc tính để thể hiện dữ liệu nắm vai trò quan trọng. Riêng trong bài toán nhận dạng tiếng ho, hệ thống nhận dạng cần phải chính xác, cần bắt được tiếng ho mỗi khi nó xuất hiện và đồng thời vẫn không nhầm tiếng ho với các âm thanh khác. Việc lựa chọn thuộc tính sao cho có ý nghĩa trong việc phân loại ho với các âm thanh khác có thể là giải pháp cho vấn đề này[1]. Những đặc trưng như vậy ngoài việc lựa chọn thủ công còn có thể được chọn tự động thông qua mạng học sâu (deep neural network). Nhiều nghiên cứu đã sử dụng đặc trưng Mel-Frequency Cepstral Coefficients (MFCC), Linear Predictive Coding (LPC) [4, 3], hoặc tự tìm ra những đặc trưng khác [13]. Bên cạnh đó, một số nghiên cứu chọn sử dụng mạng học sâu, ví dụ như mạng CNN để nghiên cứu [1, 5].



Hình 1. 1: Quy trình thường thấy trong việc phân loại tiếng ho [27]

## 1.2. Một số giải pháp hiện có cho bài toán nhận diện tiếng ho

### 1.2.1. Hull Automatic Cough Counter (HACC)

**HACC** là một chương trình được phát triển vào năm 2006 bởi nhóm các nhà nghiên cứu bao gồm Samantha J. Barry, Adrie D. Dane, Alyn H. Morice và Anthony D. Walmsley làm việc tại Đại học Hull, Vương quốc Anh (University of Hull) phục vụ cho mục đích phân tích các bản ghi số lưu lại tiếng ho [4]. Theo Barry et al. [4], HACC được phát triển nhằm tạo nên một phương thức nhận dạng và tính toán tần suất ho tiết kiệm thời gian và dễ sử dụng, không cần chuyên môn. HACC sử dụng phương pháp xử lý tín hiệu số để tính toán các hệ số miền tần số đặc trưng của các đoạn có âm thanh. Các đoạn âm thanh này sau đó được phân loại thành là tiếng ho hay không bằng mạng nơ ron dựa trên xác suất (probabilistic neural network – PNN).

Số lần ho hay tần suất xuất hiện tiếng ho có thể được tính toán sau bước tiền xử lý âm thanh thu được. Trong dự án, việc tính toán những số liệu này được thực hiện bởi con người [4].

Cụ thể theo Barry et al. [4], phương pháp của HACC gồm ba bước:

1. Trích các đoạn có âm thanh trong các bản ghi, bỏ qua các đoạn không có âm thanh sẽ bỏ qua.
2. Áp dụng Linear Predictive Coding (LPC) và một loạt các bộ lọc (filter banks) để xử lý sau đó, rồi dùng phân tích thành phần chính (PCA - Principal Component Analysis) để giảm số lượng các hệ số thu được. Việc này được thực hiện nhằm lấy ra những phần thông tin có sự khác biệt nhiều nhất để phân tích tiếp.
3. Đoạn âm thanh sẽ được phân loại bằng mạng PNN sử dụng thuật toán phân loại Bayes. Mạng PNN sẽ được huấn luyện trước với vectơ thuộc tính của một số âm thanh ho và không phải ho được chọn.

Thí nghiệm của HACC đã thực hiện việc huấn luyện cho mô hình PNN bằng dữ liệu thu thập được từ 33 người có hút thuốc, 20 nam và 13 nữ, tuổi từ 20 đến 54 gấp ván đề ho kinh niên. Sau đó mô hình sẽ được sử dụng để phân loại dữ liệu kiểm tra lấy từ các bản thu âm dài 1 tiếng từ 10 người khác, không phải những người cung cấp dữ liệu ho và không ho cho việc huấn luyện. Để đánh giá hiệu quả của mô hình, âm thanh thu được cũng sẽ được phân tích thủ công bởi một người có kinh nghiệm làm việc tại phòng khám và một người thì không có kinh nghiệm. Đồng thời một người nữa sẽ đếm

số tiếng ho trên các đoạn âm thanh đã được mạng PNN phân loại. Số tiếng ho ba người này đếm được đã được ghi lại.

	A	B	C
Nguời 1	8	6	8
Nguời 2	21	22	25
Nguời 3	5	6	9
Nguời 4	26	25	31
Nguời 5	14	30	28
Nguời 6	9	13	9
Nguời 7	8	8	15
Nguời 8	20	29	27
Nguời 9	28	53	50
Nguời 10	98	150	140
Trung bình	23.7	34.2	34.2

*Hình 1. 2: Kết quả số tiếng ho đếm được [4]*

Nghiên cứu đã ghi nhận việc đếm số tiếng ho nhanh hơn khi đã xử lý âm thanh bằng HACC trước, cụ thể thời gian đếm giảm hơn 97% so với đếm thủ công. Độ chính xác đạt được cũng cao, thể hiện ở độ nhạy (sensitivity) và độ đặc hiệu (specificity) cao.

Tuy nhiên nhược điểm của HACC là việc đếm tiếng ho còn phải thực hiện thủ công dù đã nhận diện tiếng ho và sẽ hướng tới cải tiến HACC để chương trình có thể tự đếm. Một nhược điểm nữa là HACC nhận diện tất cả âm thanh ho, bất kể là từ người đang được thu âm chính trong đoạn âm thanh hay do người khác xung quang họ tạo ra. Điều này có thể được giải quyết bằng cách loại bỏ các tiếng ho không phải từ đối tượng chính được ghi âm và sử dụng micro có độ nhạy thấp hơn. Việc sử dụng micro có độ nhạy thấp cũng sẽ giải quyết vấn đề về các âm thanh từ môi trường xung quanh.

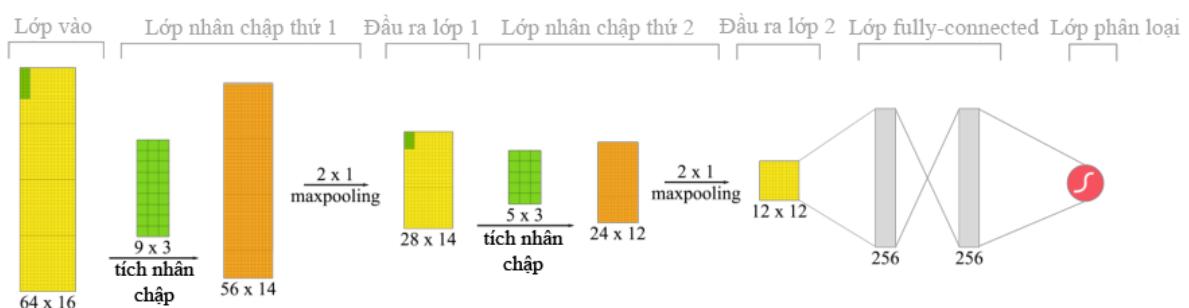
### 1.2.2. DeepCough

**DeepCough** là một hệ thống được phát triển vào năm 2015 bởi nhóm hai nhà nghiên cứu là Justin Amoh và Kofi Odame làm việc tại Thayer School of Engineering, Đại học Dartmouth thuộc Hannover, New Hampshire phục vụ cho việc nhận dạng tiếng ho [2]. Theo Amoh et al. [2], động lực để phát triển hệ thống được dựa trên nhu cầu cần có giải pháp nhận diện tiếng ho tự động theo thời gian thực hỗ trợ cho việc chẩn đoán và điều trị các bệnh đường hô hấp. Bên cạnh đó là yêu cầu cần các hệ thống này có độ chính xác cao và đủ nhạy để nhận biết những tiếng ho xuất hiện không thường xuyên, liên tục. Các nhà nghiên cứu đã cố gắng giải quyết các vấn đề này bằng các đặc trưng phức tạp được lựa chọn hoặc tinh chỉnh thủ công. Tuy nhiên, để xây dựng các đặc trưng như vậy tốn rất nhiều thời gian và chúng có thể không phù hợp

nhất với việc nhận diện tiếng ho. Từ đó hệ thống DeepCough được phát triển, áp dụng cảm biến âm thanh và mạng CNN học sâu để nhận dạng tiếng ho.

Theo Amoh et al. [2], hệ thống DeepCough bao gồm một cảm biến có thể mang theo trên người để thu nhận âm thanh hô hấp hoặc âm thanh phát ra từ miệng theo thời gian thực của người mang. Cảm biến sẽ truyền âm thanh về máy tính hoặc điện thoại để xử lý và phân loại các đoạn âm thanh. Bộ cảm biến bao gồm một bộ chuyển đổi áp lực thành dòng điện (bộ chuyển đổi áp điện) và cảm biến gia công tín hiệu (signal conditioning sensor) có nhiệm vụ tăng cường các đoạn âm thanh cần được quan tâm, tức là âm thanh hô hấp. Bộ chuyển đổi áp điện còn thu nhận năng lượng rung gây ra khi ho. Sau đó hệ thống sử dụng một mạng CNN để tìm được những thuộc tính đại diện phù hợp và đưa đặc trưng đó vào huấn luyện thuật toán phân loại.

Trong thí nghiệm của nghiên cứu, dữ liệu được cho việc xây dựng và đánh giá hệ thống được thu bằng cảm biến áp điện khi những người tình nguyện ho và đọc các câu văn có sẵn. Các âm thanh này cũng được một máy ghi âm chuyên nghiệp ghi lại để làm cơ sở so sánh với cảm biến áp điện. Dữ liệu sẽ được được tiền xử lý để làm giảm những thông tin không liên quan trước khi đưa vào mạng CNN. Dữ liệu cũng được tăng bằng cách dùng lại dữ liệu đang có nhưng sẽ lệch đi một chút, đồng thời cũng để kiểm tra khả năng nhận diện dữ liệu cho dù có sai khác nhất định với dữ liệu gốc (translation invariance). Mạng CNN sẽ được huấn luyện và sau đó chạy nhiều lần trên dữ liệu kiểm nghiệm để thu được các hệ số phải có trước (hyperparameters) như tốc độ học, số bộ lọc, v.v... Khi đã tìm được ra các hệ số đó, mô hình sẽ được huấn luyện lại và chạy với dữ liệu test để đánh giá.



Hình 1. 3: Cấu trúc của mạng CNN sử dụng cho DeepCough [2]

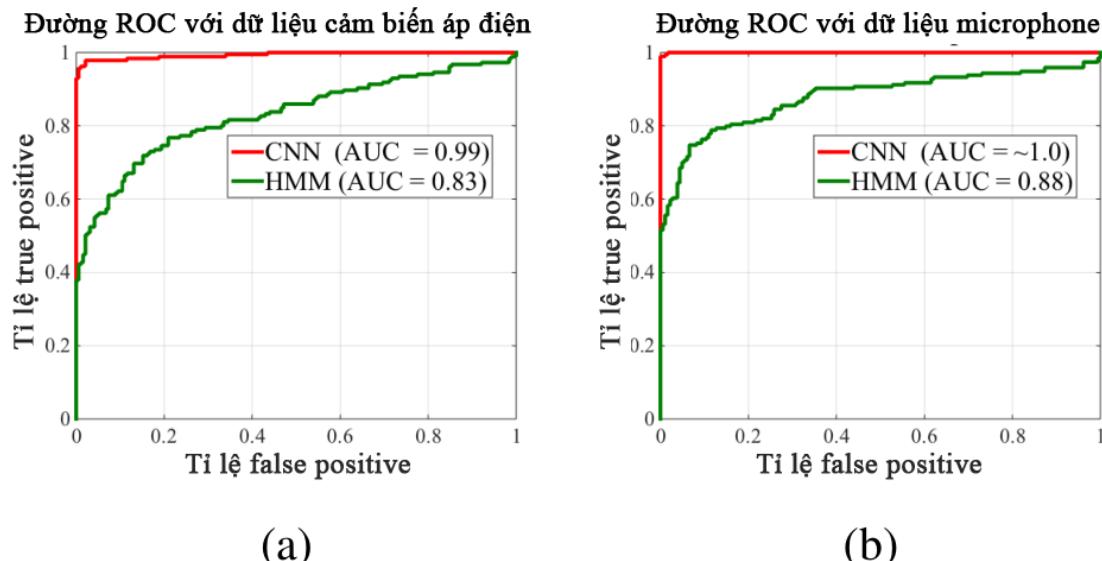
Nhóm nghiên cứu đã kiểm nghiệm hiệu quả của đặc trưng mà mạng CNN trích được so với đặc trưng MFCC và hiệu quả phân loại của mạng so với SVM. Kết quả thu được của DeepCough trong thí nghiệm này được trình bày ở Hình 1.4. STFT là dữ liệu âm thanh sau khi được tiền xử lý nhưng chưa được đưa vào mạng. SM tức là hàm softmax, một hàm mà mạng CNN của DeepCough cũng sử dụng làm hàm kích hoạt:

Mô hình	Độ nhạy	Độ đặc hiệu
MFCC+SM	87.5	86.2
MFCC+SVM	86.3	90.7
STFT+SVM	84.2	80.3
<b>STFT+CNN</b>	<b>94.0</b>	<b>91.7</b>

*Hình 1. 4: Kết quả của thí nghiệm 1 [2]*

Mô hình CNN của DeepCough hoạt động tốt hơn (khoảng 10%) so với mô hình SVM với dữ liệu STFT gốc. Hơn nữa, mô hình CNN đã hoạt động tốt hơn các phương pháp sử dụng MFCC. Như vậy nhóm nghiên cứu kết luận mô hình CNN đã trích được các đặc trưng hiệu quả cho việc phân loại tiếng ho và phân loại cũng hiệu quả hơn SVM.

Nhóm nghiên cứu cũng kiểm nghiệm hiệu quả của mạng CNN của DeepCough so với phương pháp sử dụng mô hình Markov ẩn (HMM - Hidden Markov Model) với đặc trưng MFCC. Kết quả thu được của DeepCough trong thí nghiệm này như sau

*Hình 1. 5: Kết quả của thí nghiệm 2 [2]*

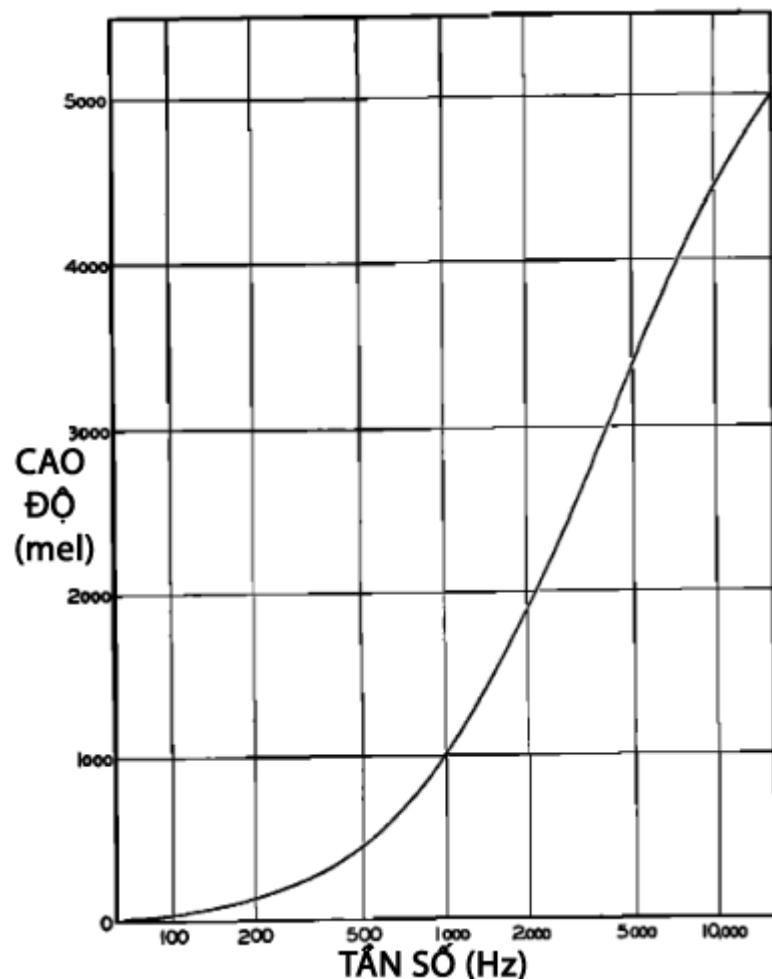
Đường Receiver Operating Characteristic (ROC) thể hiện ở hai biểu đồ trên so sánh mô hình sử dụng trong DeepCough với mô hình HMM. Giải pháp dựa trên CNN hoạt động tốt hơn hẳn HMM khi có AUC gần với 1 dù sử dụng cảm biến áp điện hay microphone. Tuy nhiên khi sử dụng microphone thì hiệu quả cao hơn một chút so với cảm biến áp điện.

### 1.3. Giải pháp đề xuất cho bài toán nhận diện tiếng ho

Dựa trên những yêu cầu đối với bài toán nhận diện tiếng ho và hiệu quả của các công nghệ đã được thể hiện trong các lĩnh vực tương tự, đồ án đề xuất giải pháp cho bài toán nhận diện tiếng ho sử dụng các thành phần sau đây.

#### 1.3.1. Đặc trưng logarit phổ tần số mel

Mel là một đơn vị đo về cao độ của âm thanh mà người nghe cảm nhận được, đề xuất bởi Stevens, Volkmann và Newmann vào năm 1937. Đơn vị đo được đặt tên là mel dựa trên từ melody trong tiếng Anh. Thang mel được xây dựng sao cho các cao độ nghe được sẽ cách đều nhau. Cao độ cảm nhận được của 1000 mel sẽ bằng với cao độ của âm thanh 1000 Hz, cường độ 40dB trên ngưỡng của người nghe. Thang mel được xây dựng nhằm mô phỏng lại cách con người cảm nhận âm thanh, đó là phân biệt được các âm thanh ở tần số thấp tốt hơn các âm thanh ở tần số cao [42, 43]. Do đó khi áp dụng thang này trong xử lý âm thanh, các đặc trưng có thể tương tự với những gì con người cảm nhận được hơn. Thang mel được sử dụng phổ biến trong các hệ thống về âm thanh như nhận diện tiếng nói tự động.



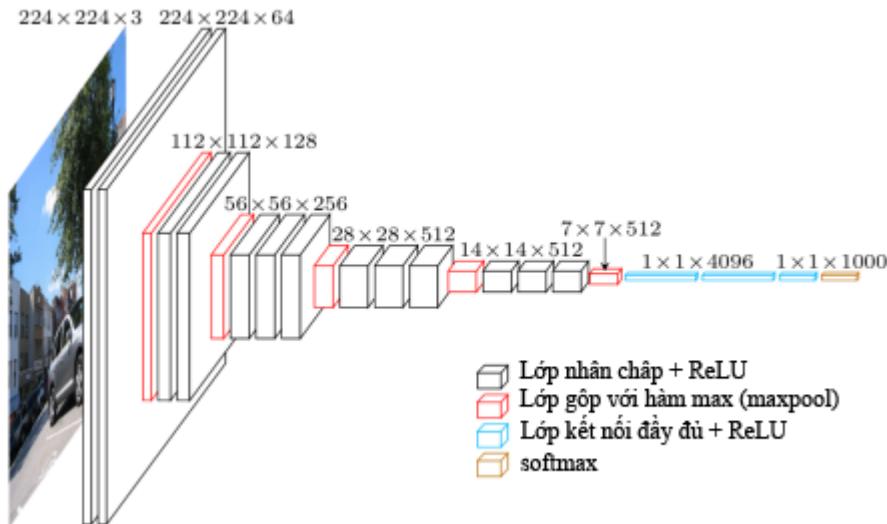
Hình 1. 6: Đồ thị của hàm giá trị cao độ theo mel theo biến là tần số [42]

Phổ tần số, thường là được xét trong một thời điểm, là một đặc trưng của âm thanh thể hiện những tần số có trong âm thanh (hoặc tín hiệu âm thanh). Phổ tần số có thể thu được thông qua biến đổi Fourier để chuyển tín hiệu âm thanh từ miền thời gian về miền tần số. Sau đó việc chuyển đổi các giá trị tần số về đơn vị mel có thể thực hiện qua công thức chuyển đổi. Giá trị logarit của phổ tần số thường được sử dụng vì cách con người cảm nhận mức độ của tín hiệu âm thanh giống như hàm logarit. Thông thường logarit sẽ tính trên độ lớn hoặc bình phương độ lớn của phổ tần số để loại bỏ pha của tín hiệu. [25]

### 1.3.2. Mạng nơ ron tích chập VGGish

**Mạng nơ ron tích chập** (CNN) thường được sử dụng trong các bài toán phân loại, đặc biệt là trong các bài toán liên quan tới hình ảnh. Một số mạng CNN phổ biến có thể kể đến như là LeNet, AlexNet và VGG. Bên cạnh việc trực tiếp học và phân loại dữ liệu, mạng CNN, cũng như một số mạng học sâu, có khả năng thể hiện dữ liệu bằng đặc trưng có hiệu quả trong bài toán phân loại. Tuy nhiên việc xây dựng và huấn luyện các mạng để thể hiện được dữ liệu về các đặc trưng như vậy rất tốn kém, cả về mặt tính toán lẫn thu thập dữ liệu do không phải lúc nào cũng có bộ dữ liệu lớn phục vụ cho bài toán cần giải quyết. Do đó các nhà nghiên cứu đã nghĩ đến việc “tái sử dụng” cách mà một mô hình đã được huấn luyện cho một bài toán khác giống như cách con người dùng những kiến thức đã học được áp dụng cho lĩnh vực khác, đây được gọi là chuyển giao việc học (transfer learning). Hầu hết các nghiên cứu về đặc trưng trích xuất mạng CNN tập trung vào đặc trưng thu được từ các lớp nhân chập hoặc lớp kết nối đầy đủ. [8]

Trước khi giới thiệu về mạng **VGGish**, cần phải nói về nguyên mẫu của nó là mạng **VGG**. Mạng VGG là tên gọi chung cho một nhóm các mạng nơ ron được đề xuất bởi Karen Simonyan và Andrew Zisserman, làm việc tại Visual Geometry Group thuộc khoa Khoa học kỹ thuật của trường Đại học Oxford (Department of Engineering Science, University of Oxford) làm nền tảng cho giải pháp của họ trong cuộc thi ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC 2014). Nhóm dự thi của Simonyan và Zisserman, dưới tên là VGG, đã giành được giải nhất và nhì lần lượt cho bài toán định vị và bài toán phân loại của cuộc thi. Nghiên cứu của nhóm đã thử nghiệm và đánh giá hoạt động của các mạng CNN sử dụng vùng tiếp nhận nhỏ với số lượng lớp tăng dần và chứng minh được độ sâu (lên tới 19) giúp cải thiện hiệu quả của các mạng trong việc phân loại ảnh. Trong nhóm mạng VGG, mạng VGG16 và VGG19 đã đạt được kết quả tốt và được cung cấp miễn phí thông qua trang web của bài báo khoa học cho nghiên cứu của Simonyan và Zisserman [21].



Hình 1. 7: Minh họa mạng VGG16 [34]

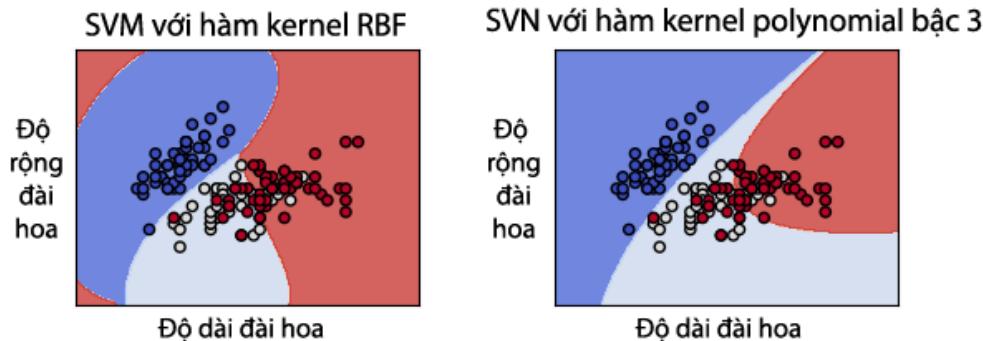
**VGGish** là một biến thể dựa trên VGG. VGGish được sử dụng cho dự án về dữ liệu âm thanh AudioSet với mục đích là để trích đặc trưng của âm thanh trong các video của bộ dữ liệu [35]. VGGish được huấn luyện với bộ dữ liệu YouTube-100M là tiền thân của bộ dữ liệu YouTube-8M. Đây là một bộ dữ liệu lớn, chứa lên tới 100 000 video. Đầu ra của mạng VGGish đã được xử lý để tương tự với dữ liệu đặc trưng âm thanh của YouTube-8M. Nhóm xây dựng AudioSet đã cung cấp miễn phí mạng VGGish, cùng checkpoint sau khi huấn luyện, và code tiền xử lý, hậu xử lý dữ liệu vào ra của mạng trên repository *models* của TensorFlow trên GitHub. Tất cả các code nằm trong repository này đều có thể được chỉnh sửa theo mục đích cá nhân hoặc thương mại theo giấy phép Apache 2.0 (Apache License 2.0) [29]. Mạng VGGish được sử dụng trong phạm vi đồ án cũng với mục đích giống như chuyển giao việc học đã nói ở trên, tức là tận dụng các “kiến thức” của mạng VGGish đã được huấn luyện với bộ dữ liệu lớn YouTube-100M. Hiệu quả của mạng cũng đã được kiểm chứng thông qua việc được sử dụng để trích dữ liệu đặc trưng cho AudioSet, với độ chính xác trung bình (average precision) cao nhất đạt được cho một lớp phân loại là 0,896..

### 1.3.3. Thuật toán SVM

**SVM (Support Vector Machine – máy vectơ hỗ trợ)** là một thuật toán được sử dụng rộng rãi trong bài toán phân loại dữ liệu. SVM đã có nhiều ứng dụng trong thực tế, ví dụ như nhận dạng số viết tay hay phân loại văn bản và cũng hoạt động hiệu quả với dữ liệu có nhiều thuộc tính. [24]

Thuật toán khởi nguồn cho SVM là Generalized Portrait được phát triển bởi Vladimir Vapnik, Alexander Lerner và Alexey Chervonenkis vào những năm 60 của thế kỷ 20 tại Nga. Thuật toán SVM hiện đại được sử dụng ngày nay phần lớn được phát triển tại phòng thí nghiệm AT&T Bell bởi Vapnik và các đồng nghiệp của mình. Do được phát triển trong bối cảnh công nghiệp như vậy, các nghiên cứu về vectơ hỗ trợ có định hướng là cho những ứng dụng trong thực tế. Từ đó, thuật toán phân loại

dựa trên vectơ hỗ trợ đã nhanh chóng cạnh tranh được với các thuật toán tốt nhất có lúc bấy giờ trong bài toán nhận diện ký tự bằng hình ảnh và nhận diện vật thể. Hiện nay, việc học dựa trên các vectơ hỗ trợ đang là một trong những lĩnh vực được tích cực nghiên cứu và nó đang dần trở thành một trong những phương pháp tiêu chuẩn trong các bộ công cụ học máy. [23]



*Hình 1. 8: Minh họa đường biên ra quyết định mà một SVM cài đặt bằng thư viện scikit-learn, trên bộ dữ liệu hoa diên vĩ (iris) [41]*

#### 1.4. Kết luận

Chương 1 đã trình bày về bài toán nhận dạng âm thanh, một số nghiên cứu về bài toán này hiện có trên thế giới, những kết quả mà họ đã đạt được và giải pháp mà đồ án đề xuất cho bài toán.

Trong chương tiếp theo, đồ án sẽ trình bày chi tiết hơn về các thành phần trong giải pháp của đồ án cho việc phân loại tiếng ho, bao gồm đặc trưng logarit phổ tần số Mel, mạng CNN và thuật toán phân loại SVM..

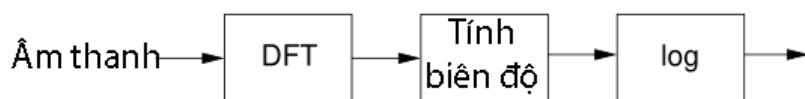
## CHƯƠNG II: NHẬN DIỆN TIẾNG HO SỬ DỤNG MẠNG NÓ RƠN TÍCH

### CHẬP VÀ THUẬT TOÁN SVM

Trong chương 2, đồ án sẽ trình bày về các thành phần sẽ sử dụng trong giải pháp của đồ án cho bài toán nhận diện tiếng ho, bao gồm các đặc trưng từ âm thanh, phương pháp trích đặc trưng và thuật toán phân loại sử dụng để nhận diện.

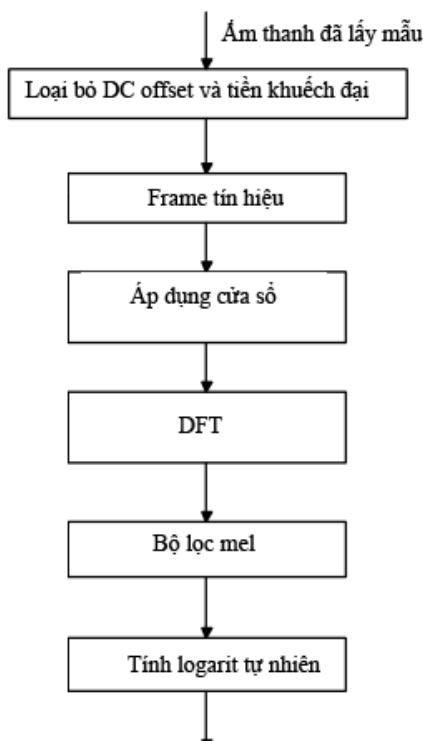
#### 2.1. Biểu diễn dữ liệu bằng đặc trưng logarit của phổ tần số mel

Từ tín hiệu âm thanh theo miền thời gian, phổ tần số sẽ thu được bằng biến đổi Fourier chuyển sang miền tần số, trong đó biến đổi Fourier rời rạc (Discrete Fourier Transform - DFT) thường được sử dụng. Quá trình tính toán logarit phổ tần số có thể mô tả bằng sơ đồ như sau:



Hình 2. 1: Sơ đồ quá trình tính logarit phổ tần số [25]

Có thể thấy được là phổ tần số mel cũng được tính như phổ tần số thông thường, nhưng cần thêm bước chuyển đổi phổ tần số về thang mel. Như vậy các bước tính đặc trưng logarit phổ tần số mel là như sau [25]: Thực hiện DFT trên các khung của âm thanh, sau đó tiến hành phân tích với băng lọc mel (mel-filter bank), kết quả thu được chính là phổ tần số mel, từ đây có thể tính logarit dựa trên độ lớn hoặc bình phương độ lớn. Bên cạnh các bước này còn có thể có thêm một số bước xử lý khác. Từ đó quá trình tính toán có thể có trình tự như ở sơ đồ trong Hình 2.2:



Hình 2. 2: Sơ đồ quá trình tính logarit phổ tần số mel [26]

Sau đây là cụ thể về các bước trong quá trình tính toán theo Xiong [26]:

## a) Tiền xử lý: Tiền khuếch đại và loại bỏ DC offset

DC offset là hiện tượng xảy ra khi cường độ trung bình của tín hiệu âm thanh là một giá trị khác 0, có thể gây nên những hiệu ứng không mong muốn khi tiếp tục xử lý trên tín hiệu nên cần loại bỏ [28]. Việc tiền khuếch đại nhằm khuếch đại các tần số cao, làm tăng năng lượng của tín hiệu tại các tần số đó để tăng hiệu quả của việc phân tích trên miền tần số [15]. Tùy vào mục đích sử dụng và dữ liệu, có thể tiến hành việc tiền xử lý này hoặc không. Các file âm thanh trong phạm vi đồ án không áp dụng bước xử lý này.

## b) Khung tín hiệu

Để tiến hành phân tích trên miền tần số, tín hiệu âm thanh thường được chia thành các khung ngắn. Đó là vì tín hiệu âm thanh thay đổi theo thời gian nhưng có thể coi là không thay đổi về những đặc trưng thống kê trên những đoạn ngắn. Các khung sau có thể chồng lên khung trước một đoạn từ  $\frac{1}{2}$  tới  $\frac{2}{3}$  độ dài khung.

Trong phạm vi đồ án này, tín hiệu âm thanh sẽ được chia thành các khung với số khung tùy thuộc vào số mẫu đã lấy trên tín hiệu, độ dài cửa sổ và độ dài khoảng cách giữa mỗi lần đặt cửa sổ theo công thức (2.1). Giá trị của độ dài cửa sổ và khoảng cách giữa mỗi lần đặt bằng giá trị sẽ sử dụng trong biến đổi Fourier thời gian ngắn (Short-time Fourier Transform - STFT) sẽ áp dụng ở bước **d) DFT**. Công thức (2.1) như sau [29]:

$$num_{frames} = 1 + floor\left(\frac{num_{samples} - length_{window}}{length_{hop}}\right) \quad (2.1)$$

Với:  $num_{frames}$  là số khung,

$num_{samples}$  là số mẫu của tín hiệu,

$length_{window}$  là độ dài của cửa sổ, trong phạm vi đồ án sử dụng độ dài là 25ms,

$length_{hop}$  là độ dài khoảng cách giữa mỗi lần đặt cửa sổ, trong phạm vi đồ án sử dụng độ dài là 10 ms.

## c) Cửa sổ

Để làm giảm sự không liên tục khi chia tín hiệu âm thanh thành các khung, mỗi khung sẽ được nhân với một hàm cửa sổ. Hai hàm cửa sổ thường được sử dụng là Hamming và Hanning. Cả hai đều làm giảm giá trị của mẫu tại đầu và cuối khung. Nếu không sử dụng bộ lọc cửa sổ nào thì có thể coi là sử dụng bộ lọc cửa sổ chữ nhật được định nghĩa như sau [25]:

$$w[n] = f(x) = \begin{cases} 1 & \text{với } 0 \leq n \leq L - 1 \\ 0 & \text{với } n \text{ còn lại} \end{cases} \quad (2.2)$$

Trong phạm vi đồ án này, một cửa sổ Hanning sẽ được trượt trên tín hiệu âm thanh sau khi nó được chia thành các khung. Cửa sổ Hanning có độ dài bằng với độ dài cửa sổ của STFT sẽ sử dụng, tức là 25ms và các cửa sổ sẽ cách nhau 10ms [29]. Hàm cửa sổ Hanning có công thức như sau [25]:

$$w[n] = f(x) = \begin{cases} 0,5 - 0,5 \cos \frac{2\pi n}{L} & \text{với } 0 \leq n \leq L - 1 \\ 0 & \text{với } n \text{ còn lại} \end{cases} \quad (2.3)$$

#### d) DFT

Bước này được thực hiện để tính toán phổ tần số của tín hiệu âm thanh. DFT sẽ được tiến hành trên từng khung để chuyển tín hiệu từ miền thời gian rời rạc về miền tần số [15]. Công thức của biến đổi DFT là như sau [10]:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j \frac{2\pi}{N} kn} \quad (2.4)$$

Với  $x[n]$  là tín hiệu âm thanh trên miền thời gian rời rạc.

Thông thường để phục vụ cho việc phân loại tiếng nói thì độ lớn của phổ sẽ được quan tâm nên tín hiệu âm thanh sau khi chuyển sang miền tần số bằng DFT sẽ được tính độ lớn hoặc bình phương độ lớn làm đầu vào cho bước tiếp theo.

Theo Plakal et al. [29], phần code cài đặt biến đổi DFT tiến hành trên các file âm thanh khi tiền xử lý trong phạm vi đồ án là theo phương pháp biến đổi Fourier nhanh (Fast Fourier Transform – FFT). Biến đổi Fourier ở sử dụng độ dài cửa sổ là 25ms, tương đương 400 mẫu, độ dài khoảng cách giữa mỗi lần đặt cửa sổ là 10ms, tương đương 160 mẫu (với tốc độ lấy mẫu là 16kHz). Do đó số điểm của FFT là  $2^9$ , được tính theo công thức (2.5) [29]:

$$FFT_{length} = 2^{\lceil \frac{\ln(\text{lengthInSamples}_{\text{window}})}{\ln(2)} \rceil} \quad (2.5)$$

Với:  $\text{lengthInSamples}_{\text{window}}$  là độ dài của cửa sổ tính theo số mẫu.

Sau khi áp dụng DFT như trên, độ lớn của tín hiệu âm thanh sẽ được tính và chuyển cho bước tiếp theo.

#### e) Áp dụng bộ lọc mel và tính logarit tự nhiên

Phổ tần số (hoặc độ lớn, bình phương độ lớn phổ) của tín hiệu âm thanh sẽ được đưa qua một bộ lọc thông dải gọi là bộ lọc mel để chuyển đổi phổ tần số về thang mel.

Trình tự tính toán ở bước này được thực hiện như sau [10]:

- Chọn giá trị tần số lớn nhất  $f_h$  và nhỏ nhất  $f_l$  rồi chuyển hai giá trị này từ Hz sang thang mel theo công thức sau [20]:

$$B(f) = 1127 \log_e(1 + \frac{f}{700}) \quad (2.6)$$

Việc chuyển đổi cũng có thể sử dụng công thức sau [26]:

$$B(f) = 2595 \log(1 + \frac{f}{700}) \quad (2.7)$$

Trong phạm vi đồ án, khi tính toán logarit của phô mel trước khi trích đặc trưng bằng VGGish, tần số lớn nhất  $f_h$  được chọn là 7500Hz và tần số nhỏ nhất  $f_l$  được chọn là 125Hz. Công thức chuyển từ Hz sang mel được sử dụng là công thức (2.6) [29].

- Tính toán các điểm biên, tức là các điểm đầu và điểm cuối mà các bộ lọc mel sẽ được đặt vào bằng công thức:

$$f[m] = \frac{N}{F_s} B^{-1} \left( B(f_l) + m \frac{B(f_h - f_l)}{M + 1} \right) \quad (2.8)$$

Với:  $f[m]$  là điểm biên

$N$  là số điểm của biến đổi FFT.

$F_s$  là tần số lấy mẫu tính bằng Hz, trong phạm vi đồ án được chọn là 16kHz.

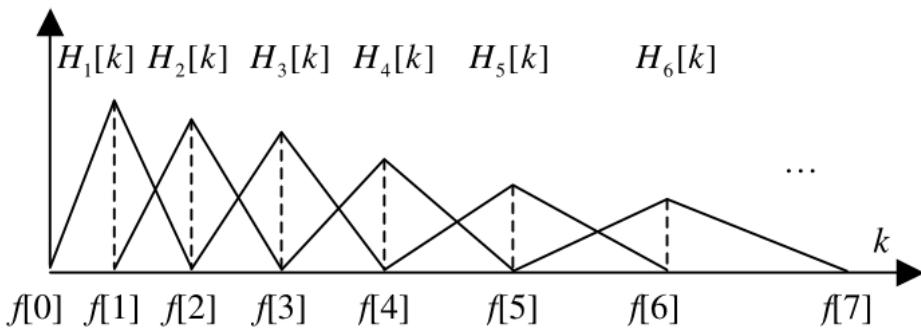
$B^{-1}$  là biến đổi tần số từ mel sang Hz, tức là  $B^{-1}(f) = 700(e^{\frac{f}{1125}} - 1)$ .

$M$  là số bộ lọc mel, số bộ lọc thông thường trong khoảng từ 24 tới 40.

- Áp dụng băng lọc mel có  $M$  bộ lọc được định nghĩa như sau:

$$H_m[k] = \begin{cases} 0 & \text{với } k < f[m] \\ \frac{2(k - f[m - 1])}{(f[m + 1] - f[m - 1])(f[m] - f[m - 1])} & \text{với } f[m - 1] \leq k \leq f[m] \\ \frac{2(f[m + 1] - k)}{(f[m + 1] - f[m - 1])(f[m + 1] - f[m])} & \text{với } f[m] \leq k \leq f[m + 1] \\ 0 & \text{với } k > f[m + 1] \end{cases} \quad (2.9)$$

Với  $0 \leq m < M$ .



Hình 2. 3: Băng lọc mel theo công thức (2.7) [10]

Với mỗi bộ lọc mel, áp dụng nó lên tín hiệu âm thanh và tính logarit tự nhiên kết quả thu được:  $S[m] = \log_e \sum_{k=0}^{N-1} |X[k]|^2 H_m[k]$ , nếu ở bước trước sử dụng bình phương độ lớn phô ( $|X[k]|^2$ ) làm đầu ra.

Việc sử dụng logarit tự nhiên là để xấp xỉ cách tiếp nhận không tuyến tính của tai người về độ to và mật độ âm thanh và để chuyển phép nhân về phép cộng, từ đó có thể loại bỏ một số hiệu ứng làm biến đổi âm thanh đi kèm tín hiệu dưới dạng tích chập như hiệu ứng lọc của micro [26].

Theo Theo Plakal et al. [29], số bộ lọc mel sử dụng để tính phô mel là 64 bộ lọc. Sau khi logarit phô tần số mel của các file âm thanh đã được tính, kết quả thu được được khung thành các mẫu với tốc độ lấy mẫu 100 Hz, độ dài cửa sổ và bước nhảy là 960 ms.

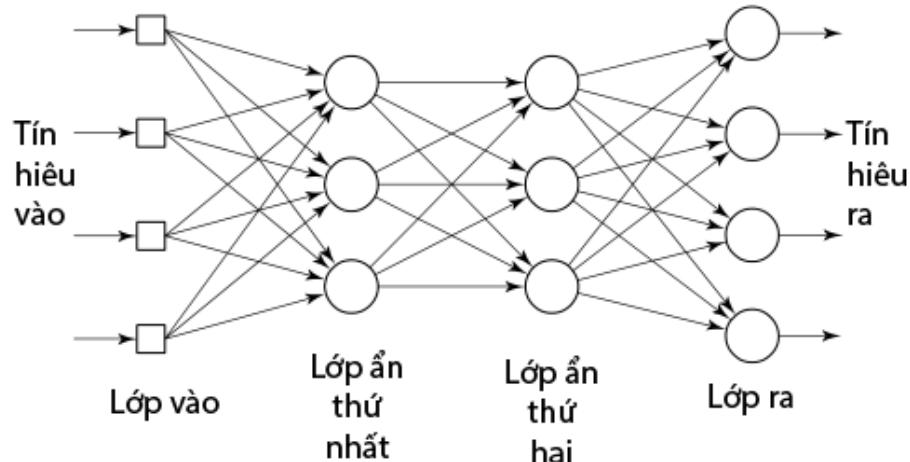
## 2.2. Giới thiệu về mạng CNN VGGish

### 2.2.1. Mạng nơ ron tích chập

#### a) Mạng nơ ron

Theo Negnevitsky [16], **mạng nơ ron**, hay mạng nơ ron nhân tạo (neural network, artificial neural network - ANN) là một mô hình suy luận mô phỏng theo não người. Bộ não con người bao gồm dày đặc các tế bào thần kinh kết nối với nhau, được gọi là nơ ron và kết nối giữa chúng được gọi là xinap. Tín hiệu được truyền giữa các nơ ron thông qua các phản ứng hóa-sinh phức tạp. Các chất hóa học giải phóng từ xinap sẽ gây nên sự thay đổi điện thế trong thân của tế bào và khi nó đạt đến một mức nào đó sẽ tạo nên xung điện truyền ra sợi trực của nơ ron, phát tán tới xinap, làm thay đổi hiệu điện thế tại đó. Khi phản ứng lại với các kích thích như vậy xuất hiện thường xuyên, nơ ron sẽ cung cấp kết nối của nó tới các nơ ron khác hoặc hình thành các kết nối mới. Cơ chế này chính là nền tảng cho việc học ở não bộ. Mô phỏng lại hoạt động này của não, các mạng nơ ron nhân tạo sẽ gồm các đơn vị xử lý kết nối với nhau, gọi là nơ ron. Mỗi kết nối sẽ có trọng số riêng và đóng vai trò truyền “tín hiệu” giữa các nơ ron. Mỗi nơ ron sẽ nhận một số tín hiệu đầu vào thông qua các kết nối và sinh ra chỉ một tín hiệu ra. Kết nối đầu ra của nơ ron sẽ ra làm nhiều nhánh tương ứng với các kết nối tới

các nơ ron khác, tất cả đều truyền cùng một tín hiệu đầu ra của nơ ron. Một mạng có thể chia làm một hoặc nhiều lớp chứa các nơ ron hoạt động tương tự nhau.



Hình 2. 4: Ví dụ một ANN có hai lớp [16]

Cụ thể, mỗi nơ ron sinh tín hiệu ra bằng cách tính toán như sau: Nơ ron tính tổng của tất cả tín hiệu vào, trừ đi một giá trị ngưỡng để thu được [16]:

$$X = \sum_{i=1}^N x_i w_i - \theta \quad (2.10)$$

Với:  $N$  là số đầu vào của nơ ron, do đó  $x_i$  là giá trị tín hiệu vào thứ  $i$ ,  $w_i$  là trọng số tương ứng

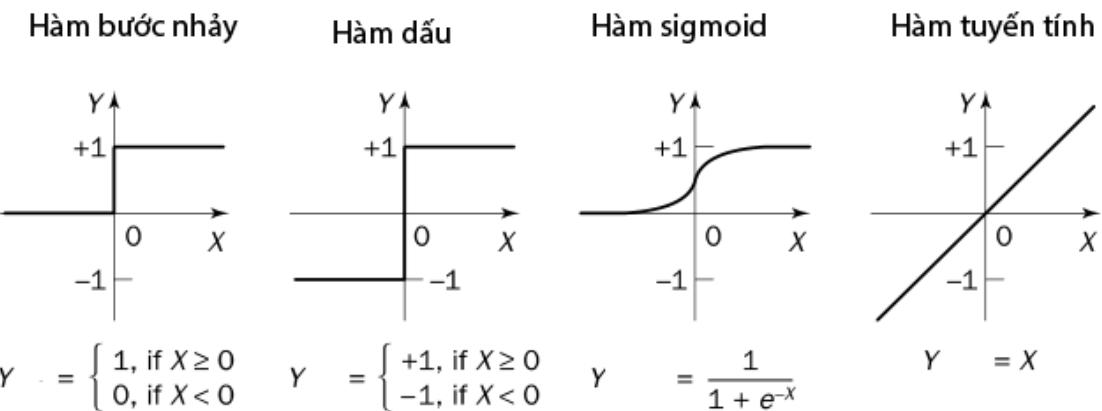
$\theta$  là giá trị ngưỡng.

Đặt  $b = -\theta$  thì công thức (2.13) sẽ là:

$$X = \sum_{i=1}^N x_i w_i + b \quad (2.11)$$

$b$  được gọi là độ lệch (bias).

Sau đó áp dụng hàm kích hoạt lên  $X$ . Giá trị thu được được gọi là mức độ kích hoạt. Một số hàm kích hoạt thường được sử dụng là các hàm chặn như hàm bước nhảy, hàm dấu, hàm sigmoid, hàm tuyến tính, v.v...



Hình 2.5: Một số hàm kích hoạt thường được sử dụng [16]

Theo Negnevitsky [16], các trọng số chính là cách mà mạng nơ ron đánh giá độ quan trọng của một tín hiệu vào. Mạng nơ ron tiến hành học bằng cách chỉnh sửa các trọng số sao cho mô phỏng độ quan trọng của các tín hiệu vào sát với thực tế nhất. Từ đó, các thuật toán học của mạng nơ ron thường bao gồm các bước: Khởi tạo trọng số, tính toán mức độ kích hoạt, cập nhật trọng số (bao gồm cả độ lệch bias) và lặp lại các bước này cho đến khi có thể đánh giá được là mạng đã hoạt động được theo mong muốn. Chủ yếu điểm khác biệt là hàm kích hoạt, luật học, hay chính là cách cập nhật các trọng số và hàm mất mát (cost function/loss function). Ví dụ thuật toán perceptron sử dụng hàm kích hoạt là hàm dấu và luật học như sau: [16]

$$w_i(p+1) = w_i(p) + \Delta w_i(p) \quad (2.12)$$

Với  $\Delta w_i(p)$  là độ chỉnh sửa trọng số, được tính theo luật delta như sau [16]:

$$\Delta w_i(p) = \alpha \times x_i(p) \times e(p) \quad (2.13)$$

Ở đây,  $i$  tương ứng là số thứ tự của trọng số,  $p$  là số vòng lặp đang thực hiện,  $\alpha$  là tốc độ học,  $e(p)$  là sai lệch so với đầu ra kỳ vọng, còn gọi là lỗi [16]:

$$e(p) = Y_d(p) - Y(p) \quad (2.14)$$

Với  $Y_d(p)$  là đầu ra kỳ vọng,  $Y(p)$  là đầu ra thực tế.

Với thuật toán lan truyền ngược sử dụng hàm kích hoạt sigmoid, thì độ chỉnh sửa trọng số  $\Delta w_{jk}(p)$  lại được tính như sau [16]:

$$\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p) \quad (2.15)$$

Với  $j$  là số thứ tự của nơ ron ở lớp trước,  $k$  là số thứ tự nơ ron của lớp sau,  $\delta_k(p)$  là gradien lỗi tại nơ ron  $k$ , được tính như sau [16]:

$$\delta_k(p) = \frac{\partial y_k(p)}{\partial X_k(p)} \times e_k(p) = y_k(p) \times (1 - y_k(p)) \times e_k(p) \quad (2.16)$$

Ở đây,  $e_k(p)$  là lỗi tại nơ ron  $k$ , công thức tính vẫn như với perceptron,  $y_k(p)$  là đầu ra tại nơ ron  $k$  nên được tính bằng hàm sigmoid. Phương pháp tính  $\delta_k(p)$  như công thức (2.19) được gọi là phương pháp tính gradien lan truyền ngược (back-

propagation) [9]. Thuật toán này là công cụ chính sử dụng trong việc học của các mạng nơ ron [17].

Hàm mất mát là một cách để đánh giá việc huấn luyện mạng nơ ron [17], được thực hiện sau khi mạng sinh ra giá trị đầu ra với một dữ liệu vào. Mạng được gọi là hội tụ khi hàm này đạt được đến giá trị đủ nhỏ, tức là mạng đã tiệm cận với thực tế cần được mô phỏng lại bằng dữ liệu huấn luyện, và thường việc huấn luyện sẽ dừng lại khi mạng hội tụ [16]. Một số hàm mất mát thường được sử dụng là tổng bình phương lỗi, trung bình bình phương lỗi [17, 28].

## b) Mạng nơ ron tích chập

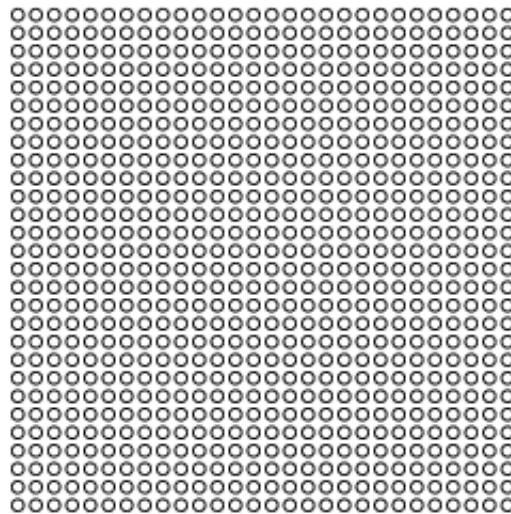
Về khái niệm mạng nơ ron tích chập, theo Goodfellow et al. [9], **mạng nơ ron tích chập** (Convolutional neural network, convolutional network – CNN) được định nghĩa là một loại mạng nơ ron sử dụng tích chập trong ít nhất là một lớp của mạng, thường được sử dụng trong phân tích hình ảnh.

Một trong những nền tảng đầu tiên của mạng CNN hiện đại ngày nay là mô hình neocognitron được đề xuất bởi Kunihiko Fukushima vào năm 1980. Theo Fukushima [6], mô hình này được dựa trên nghiên cứu của David Hubel và Torsten Wiesel về hệ thống thị giác của động vật có vú, cụ thể là hai khái niệm tế bào đơn giản và tế bào phức tạp trong vỏ não thị giác. Theo Hubel et al. [11], tế bào đơn giản ở mèo sẽ phản hồi lại những đường viền hoặc đường thẳng có hướng và tế bào phức tạp cũng tương tự, nhưng tế bào phức tạp sẽ phản hồi với những đường viền dù chúng có bị thay đổi vị trí trong tầm nhìn. Hubel và Wiesel cũng đề xuất rằng [12]: tế bào phức tạp ở mèo có thể làm được như vậy là nhờ tổng hợp đầu ra của nhiều tế bào đơn giản phản hồi với cùng một hướng nhưng ở trong các tầm nhìn khác nhau. Khái niệm này cũng được phát hiện trong hệ thống thị giác của con người và chính là nền tảng cơ bản của các mô hình CNN [32]. Dựa trên điều này, Fukushima với ý tưởng chung là đi từ cơ bản tới phức tạp đã xây dựng mô hình neocognitron có hai thành phần được gọi là tế bào S và tế bào C là những toán tử, tương ứng với các tế bào đơn giản và phức tạp. Tế bào S nằm ở lớp đầu tiên và được kết nối tới tế bào C ở lớp thứ hai. Hai khái niệm này có sự tương đương với lớp nhân chập và lớp gộp thường được sử dụng trong mạng CNN ngày nay. Dựa trên nghiên cứu của Fukushima, Yann LeCun et al. [14] vào năm 1998 đã trình bày về mô hình LeNet-5, một trong những ứng dụng mạng CNN thành công đầu tiên, dùng để chuyển đổi những đặc trưng đơn giản thành những đặc trưng phức tạp hơn có hiệu quả trong việc nhận diện ký tự viết tay. LeNet-5 được huấn luyện trên bộ dữ liệu MNIST, là bộ dữ liệu mở gồm các chữ số được viết tay từ 0 tới 9 được đề xuất vào năm 1998 cùng với LeNet-5. Việc huấn luyện được thực hiện bằng cách đưa một hình ảnh mẫu để mạng nhận diện chữ số và mạng sẽ được cập nhật lại dựa trên việc nó dự đoán đúng hay sai.

Mạng CNN về cơ bản sử dụng ba loại lớp là **lớp nhân chập** (convolutional layer), **lớp gộp** (pooling layer) và **lớp kết nối đầy đủ** (fully-connected layer). [33]

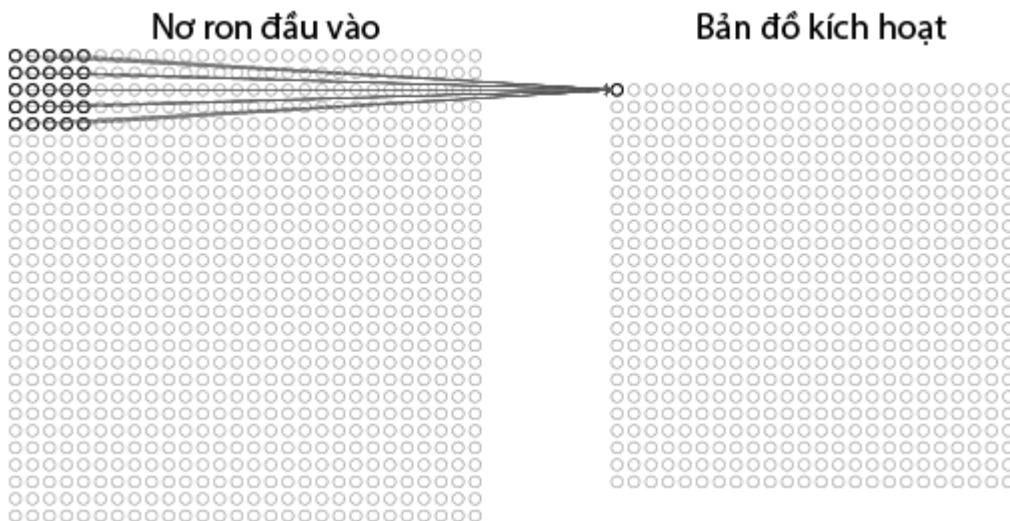
Trước tiên cần lưu ý là mạng CNN nhận đầu vào là dữ liệu dưới dạng lưới, ví dụ như dữ liệu trên miền thời gian một chiều gồm các mẫu được lấy tại các thời điểm hay dữ liệu hình ảnh có thể được coi là hai chiều hoặc ba chiều gồm giá trị của các pixel trên ảnh [9]. Từ đó có thể coi đầu vào của CNN là mảng gồm các nơ ron lấy giá trị từ mảng dữ liệu đầu vào tại vị trí tương ứng trong mảng [17].

### Nơ ron đầu vào



Hình 2. 6: Ví dụ mảng hai chiều các nơ ron ở lớp đầu vào của CNN [17]

Theo Johnson et al. [33], **lớp nhân chập** bao gồm một nhóm các bộ lọc có thể học được. Kích thước của các bộ lọc này là nhỏ nhưng sẽ di chuyển khắp các chiều của mảng đầu vào để thực hiện nhân vô hướng bộ lọc với một vùng của đầu vào. Với mỗi một vị trí của bộ lọc sẽ thu được một giá trị phản hồi sau khi nhân chập, và khi chạy hết các chiều của mảng đầu vào sẽ thu được một bản đồ kích hoạt (activation map), hay còn gọi là bản đồ đặc trưng (feature map) [17], được minh họa bằng Hình 2.5 dưới đây, mỗi vị trí trên bản đồ tương ứng một nơ ron. Mạng CNN có thể học được các bộ lọc được kích hoạt khi thấy đặc trưng trên dữ liệu, ví dụ như với dữ liệu ảnh thì là đường viền theo một hướng nào đó. Mỗi bộ lọc để nhận diện các đặc trưng khác nhau sẽ sinh ra một bản đồ kích hoạt.



Hình 2.7: Ví dụ tương ứng từ một vùng trên đầu vào hai chiều với một giá trị trên bản đồ kích hoạt [17]

Có thể hình dung mỗi nơ ron trong lớp nhân chập sẽ kết nối với một vùng nơ ron đầu vào, thay vì chỉ một nơ ron. Vùng này được gọi là vùng tiếp nhận cục bộ, chiều ngang và dọc của nó sẽ được xác định tùy ý nhưng chiều sâu, nếu có, sẽ bằng với chiều sâu của mảng đầu vào. Bộ lọc đã nhắc ở trên sẽ có kích thước bằng với kích thước của vùng này. Số lượng nơ ron trong lớp nhân chập sẽ phụ thuộc vào các thông số cài đặt sẵn là: độ sâu của đầu ra, bước nhảy của vùng tiếp nhận và kích cỡ viền được thêm vào. Độ sâu tương ứng với số lượng bộ lọc, tức là có bao nhiêu bộ lọc thì có bấy nhiêu độ sâu. Mỗi nhóm các nơ ron cùng quan sát một đặc trưng của dữ liệu, tức là xuất phát từ cùng một bản đồ đặc trưng của một bộ lọc, được gọi là một cột cùng độ sâu. Bước nhảy của vùng tiếp nhận là khoảng cách giữa mỗi lần di chuyển vùng này. Ví dụ như với dữ liệu ảnh thì có thể là 1 hoặc 2 pixel. Viền được thêm vào là các số 0 được chèn thêm vào bao quanh đầu ra để điều chỉnh kích thước đầu ra tùy ý, có thể là để kích thước đầu ra bằng kích thước đầu vào. Từ đó kích thước đầu ra của lớp, hay chính là bản đồ kích hoạt, có thể được tính như sau [33]

$$\frac{(W - F + 2P)}{S} + 1 \quad (2.17)$$

Với: W là kích thước đầu vào (giả sử mảng đầu vào vuông, nếu không có thể sử dụng công thức trên với riêng chiều dài và chiều rộng)

F là kích thước vùng tiếp nhận cục bộ

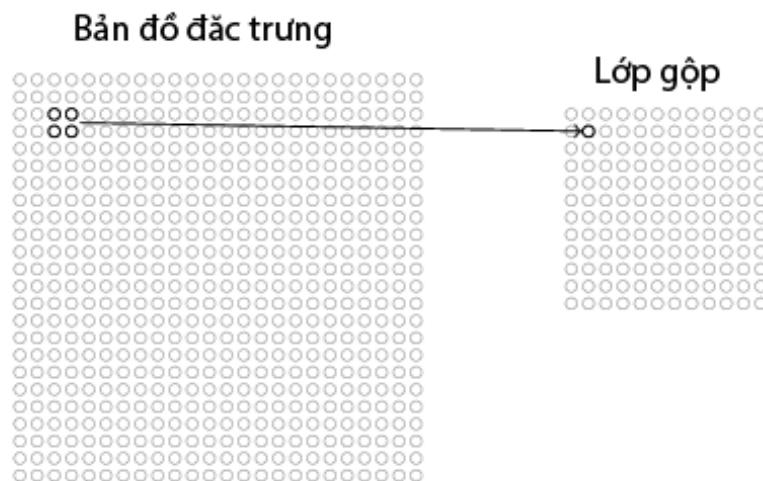
P là kích thước viền thêm vào

S là độ dài bước nhảy

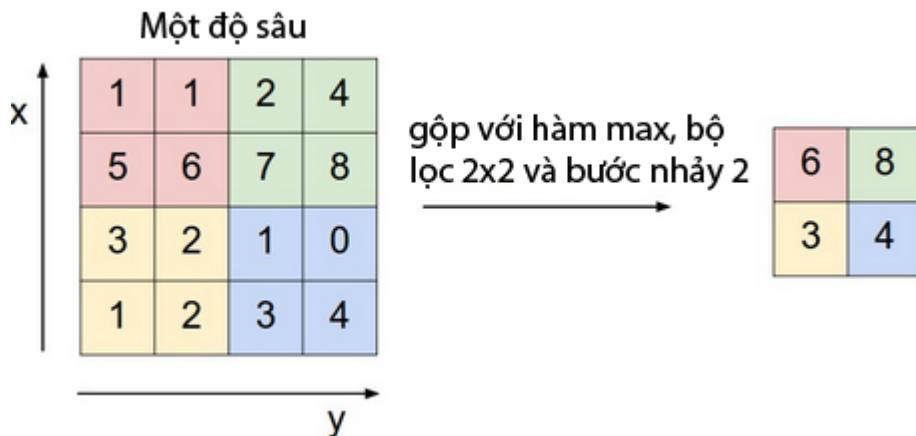
Việc chia sẻ hệ số giữa các nơ ron cũng có thể được thực hiện để kiểm soát số lượng hệ số. Ví dụ nếu lớp nhân chập có  $55*55*96 = 290\ 400$  nơ ron (với 96 là số bộ lọc), mà mỗi nơ ron lại có  $11*11*3 = 363$  hệ số (11 là kích thước vùng tiếp nhận, 3 là độ sâu của ảnh) sẽ dẫn tới số lượng hệ số lên tới  $105\ 705\ 600$ . Việc chia sẻ hệ số tức là

trên mỗi cột cùng độ sâu thì các nơ ron sẽ cùng sử dụng một bộ trọng số và bias. Nếu thực hiện chia sẻ trọng số với ví dụ trên, số trọng số sẽ giảm xuống còn 96 bộ trọng số khác nhau, tức là  $96 \times 11 \times 11 \times 3 = 34\,848$ , cộng thêm 96 bias là 34 944 hệ số. Trong ứng dụng thực tế, khi thực hiện lan truyền ngược, mỗi nơ ron sẽ tính gradien cho trọng số của mình, rồi gradien của các nơ ron trên cùng độ sâu sẽ được cộng tổng lại và chỉ cập nhật một bộ trọng số duy nhất thống nhất trên toàn cột cùng độ sâu. Khi các nơ ron cùng độ sâu cùng sử dụng một bộ trọng số thì trong quá trình truyền xuôi qua lớp nhân chập, kết quả có thể tính bằng tích chập của trọng số với đầu vào. Vì thế nên lớp này có tên là lớp nhân chập và bộ trọng số thường được gọi là bộ lọc.

**Lớp gộp** là lớp thường được chèn vào giữa các lớp nhân chập, được sử dụng với mục đích làm giảm kích thước của đầu vào, tức là bản đồ đặc trưng ở đầu ra của lớp nhân chập đứng trước nó, qua đó giúp làm giảm số lượng các hệ số và việc tính toán trong mạng để kiểm soát hiện tượng quá khớp (overfitting) [33, 27]. Theo Johnson et al. [33], lớp gộp sẽ hoạt động độc lập giữa các độ sâu khác nhau của đầu vào, tức là các bản đồ đặc trưng, để làm giảm kích thước của chúng. Việc gộp thường được thực hiện bởi hàm cực đại, tuy nhiên một số hàm khác như hàm trung bình hay hàm L2-norm (hàm lấy căn bậc hai tổng của bình phương các giá trị) cũng được sử dụng. Cách thực hiện gộp thường được sử dụng là dùng bộ lọc kích thước  $2 \times 2$  với bước nhảy là 2 và hàm cực đại chạy dọc cả chiều ngang và dọc của từng bản đồ đặc trưng, khiến cho kích thước của bản đồ sau khi qua lớp gộp giảm một nửa. Tại mỗi vị trí đặt bộ lọc sẽ thu được một giá trị cực đại của vùng  $2 \times 2$  thuộc bản đồ đặc trưng nằm dưới bộ lọc, tương ứng với một đơn vị trong đầu ra của lớp gộp.



Hình 2. 8: Ví dụ tương ứng một vùng  $2 \times 2$  trên bản đồ đặc trưng với một đơn vị ở lớp gộp [17]



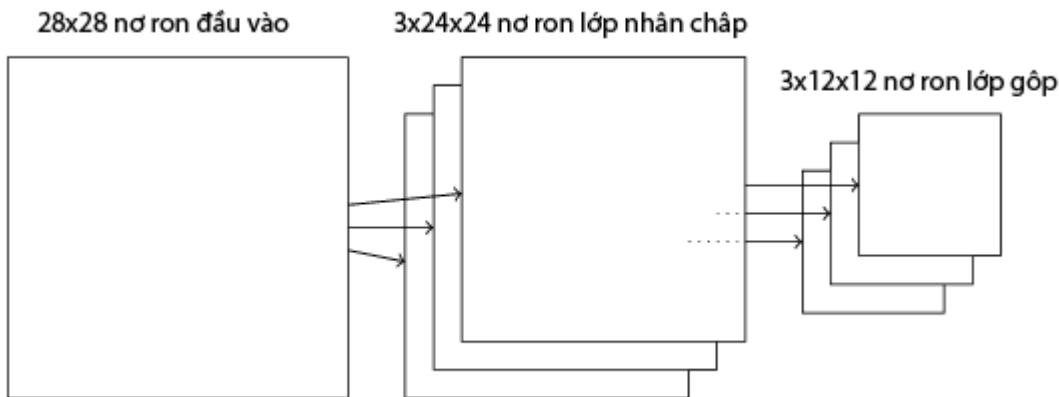
Hình 2. 9: Ví dụ hoạt động ở lớp gộp với bộ lọc  $2 \times 2$  và hàm cực đại [33]

Lớp gộp không sinh ra hệ số mới và thường không thực hiện chèn thêm với đầu vào. Kích thước đầu ra của lớp gộp phụ thuộc vào kích thước đầu vào, kích thước bộ lọc và bước nhảy của bộ lọc. Với đầu vào có kích thước  $W_1 \times H_1 \times D_1$ , bộ lọc kích thước  $F$  (bộ lọc vuông) và độ dài bước nhảy  $S$  thì đầu ra của lớp gộp sẽ có kích thước  $W_2 \times H_2 \times D_2$  là [33]:

$$W_2 = \frac{(W_1 - F)}{S} + 1 \quad (2.18)$$

$$H_2 = \frac{(H_1 - F)}{S} + 1$$

$$D_2 = D_1$$



Hình 2. 10: Ví dụ đầu ra của lớp gộp với đầu vào là bản đồ đặc trưng  $24 \times 24 \times 3$  [17]

Khi thực hiện lan truyền ngược, gradien sẽ được tính với đầu vào có giá trị lớn nhất khi truyền xuôi vào, do đó thường thì chỉ số của giá trị đi qua hàm max sẽ được lưu lại để việc tính gradien hiệu quả hơn khi lan truyền ngược [33].

Việc thực hiện gộp có thể được coi như là một cách để mạng kiểm tra xem một đặc trưng có xuất hiện ở vùng nào trong dữ liệu không và bỏ qua vị trí xuất hiện cụ thể của nó, chỉ lưu lại thông tin việc nó đã xuất hiện. Việc này thực hiện theo quan điểm phát hiện đặc trưng cho dù nó ở vị trí nào [9, 27]. Tuy nhiên, không phải lúc nào quan

điểm này cũng đúng với mọi trường hợp dữ liệu, ví dụ như khi muốn tìm góc tạo bởi hai đường thẳng giao nhau trong một bức ảnh thì cần phải biết vị trí cụ thể của chúng để xem chúng sẽ giao ở đâu [9]. Theo Johnson et al. [33], một số nghiên cứu đã đề xuất việc loại bỏ lớp gộp để thu được cấu trúc mạng chỉ gồm các lớp nhân chập. Để vẫn làm giảm được kích thước các đặc trưng thu được, họ đề xuất sử dụng bước nhảy lớn hơn ở các lớp nhân chập.

Theo Johnson et al. [33], **Lớp kết nối đầy đủ** cũng có thể sử dụng trong mạng CNN phục vụ mục đích như phân loại dữ liệu đầu vào của mạng. Vì lớp kết nối đầy đủ có đầu vào chỉ có một chiều nên nếu lớp trước nó đưa ra đầu ra hai chiều hoặc ba chiều thì cần tiến hành làm phẳng (flatten) trước khi đưa kết quả đó vào lớp kết nối đầy đủ. Đầu ra của lớp kết nối đầy đủ sẽ có một chiều. Với lớp kết nối đầy đủ dùng để phân loại, kích thước đầu ra sẽ bằng số lượng các lớp mà dữ liệu sẽ được phân vào. Lớp kết nối đầy đủ sử dụng trong mạng CNN không có gì khác so với ở mạng nơ ron thông thường, có thể coi như là một mạng nơ ron khác kết nối với mạng CNN đang sử dụng. Do đó lớp kết nối đầy đủ có thể bao gồm nhiều lớp ẩn sử dụng các hàm kích hoạt khác nhau. Một trong những phương pháp phân loại thường được sử dụng ở lớp này với mạng CNN là sử dụng hàm softmax, hàm mất mát sử dụng có thể là hàm entropy chéo [9].

Theo Johnson et al. [33], cấu trúc của một mạng CNN có thể bao gồm nhiều lần lặp lại nhóm gồm lớp gộp theo sau bởi lớp nhân chập, cho tới một điểm sẽ chuyển sang lớp kết nối đầy đủ. Lớp kết nối đầy đủ ở cuối đóng vai trò đưa ra kết quả, ví dụ như là phân loại dữ liệu đầu vào của mạng. Kiến trúc phổ biến của mạng CNN có thể được mô tả như sau:

$$\text{INPUT} \rightarrow [\text{CONV} \times N \rightarrow \text{POOL?}] \times M \rightarrow \text{FC} \times K \rightarrow \text{FC}$$

Với  $\times$  thể hiện việc lặp,  $N, M, K$  là số lần lặp, lớn hơn hoặc bằng 0, thường thì  $N \leq 3, K < 3$ ,  $\text{POOL?}$  là có thể có lớp gộp hoặc không.

### 2.2.2. Mạng nơ ron tích chập VGGish

#### a) Mạng VGG

##### Cấu trúc mạng

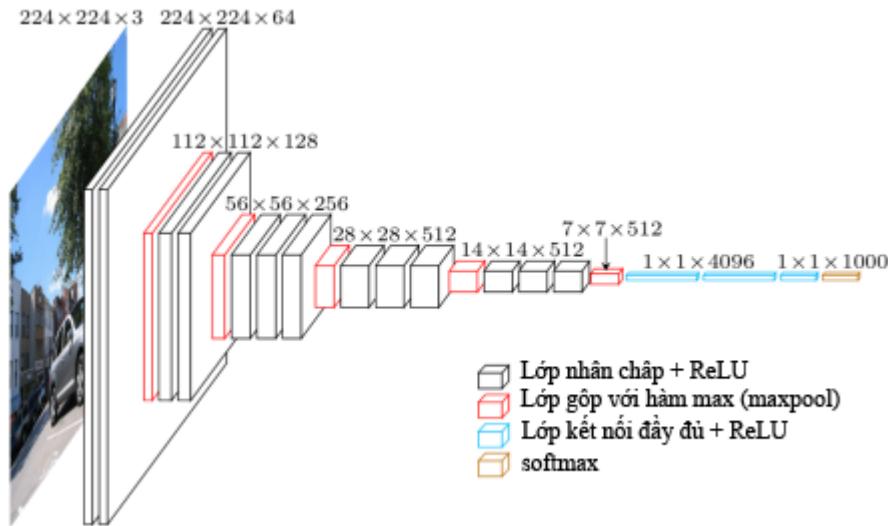
Theo Simonyan et al. [21], kiến trúc mà họ sử dụng cho VGG bao gồm các lớp nhân chập với vùng tiếp nhận kích cỡ 3x3 (có một mạng dùng cỡ 1x1), bước nhảy là 1 pixel và thêm viền vào mỗi vùng tiếp nhận để đảm bảo kích thước sau khi nhân chập giữ nguyên. Các lớp nhân chập đều sử dụng hàm kích hoạt là hàm nắn tuyến tính (Rectified linear unit - ReLU). Kiến trúc này có năm lớp gộp sử dụng hàm max, được thêm vào sau một số lớp nhân chập và dùng cửa sổ 2x2, bước nhảy là 2. Các lớp cuối cùng trong kiến trúc là ba lớp kết nối đầy đủ, hai lớp đầu có 4096 nơ ron và hàm kích hoạt ReLU, lớp cuối có 1000 nơ ron tương ứng với 1000 lớp phân loại và sử dụng hàm softmax. Cấu trúc ba lớp cuối này là giống nhau giữa các mạng, khác biệt là ở số lớp nhân chập. Các mạng được sử dụng, đặt tên từ A tới E, được mô tả theo bảng ở Hình

2.11. Đầu vào của mạng là ảnh có kích thước 224x224 và đầu ra là xác suất của mỗi nhãn phân loại cho ảnh.

Cấu trúc					
A	A-LRN	B	C	D	E
11 lớp trọng số	11 lớp trọng số	13 lớp trọng số	16 lớp trọng số	16 lớp trọng số	19 lớp trọng số
Đầu vào (ảnh 224 x 224)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
Gộp bằng hàm max (maxpool)					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
Gộp bằng hàm max (maxpool)					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
Gộp bằng hàm max (maxpool)					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
Gộp bằng hàm max (maxpool)					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
Gộp bằng hàm max (maxpool)					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Hình 2. 11: Cấu trúc các mạng VGG [21]

Trong Hình 2.11, cách viết conv3-64 thể hiện 3 lớp nhân chập với số nơ ron là 64. Cách viết FC-4096 tức là lớp kết nối đầy đủ với 4096 nơ ron. LRN là Local Response Normalisation, là phương pháp chuẩn hóa phản ứng cục bộ, thể hiện là đầu ra lớp đó có dùng chuẩn hóa LRN.



Hình 2. 12: Minh họa cấu trúc mạng VGG16 [34]

Trong số các mạng thuộc nhóm VGG, mạng D và E, hay còn được gọi là VGG16 và VGG19 cho kết quả tốt nhất trong bài toán phân loại và nhóm nghiên cứu đã cung cấp miễn phí hai mô hình đã huấn luyện của họ thông qua trang web của bài báo khoa học để mọi người có thể sử dụng [21].

### b) Mạng VGGish

#### Cấu trúc mạng

Cấu trúc VGGish có sự thay đổi so với cấu trúc A của VGG ở điểm VGGish đã bỏ đi một nhóm lớp nhân chập – lớp gộp ở cuối nên chỉ còn bốn nhóm như vậy. Đồng thời, lớp kết nối đầy đủ ở cuối thay vì có 1000 nơ ron thì là 128, vì lớp này ở VGGish đóng vai trò chuyển các đặc trưng mà các lớp trước đã trích được về một vectơ có số chiều giới hạn chứ không còn là lớp phân loại. Ngoài ra đầu vào của mạng cũng chuyển thành  $96 \times 64$  để có thể tiếp nhận đặc trưng logarit phổ tần số mel của âm thanh đã được trích trước đó [29]. Như vậy cấu trúc của VGGish sẽ như sau:



Hình 2. 13: Minh họa cấu trúc VGGish

Lớp nhân chập của VGGish vẫn sử dụng hàm kích hoạt ReLU, vùng tiếp nhận giữ nguyên là  $3 \times 3$ , bước nhảy là 1. Lớp gộp vẫn dùng hàm max, cửa sổ  $2 \times 2$  với bước nhảy là 2 và như ở cấu trúc A của VGG.

#### Nhúng (Embedding) là đầu ra của mạng

**Nhúng (Embedding)** trong bối cảnh học máy, hay cụ thể hơn là về học sâu, là một khái niệm để chỉ việc ánh xạ các đối tượng rời rạc, ví dụ như các từ trong một câu, về các vectơ số thực. Từng chiều trong vectơ vốn không có ý nghĩa gì cả, mà toàn bộ

vectơ là để mô tả các mẫu khái quát của vị trí và khoảng cách giữa các vectơ [30]. Tức là việc nhúng sẽ cố gắng đặt những dữ liệu đầu vào giống nhau về mặt ngữ nghĩa trong không gian của dữ liệu ki vào để đảm bảo nắm bắt được các thông tin ngữ nghĩa của dữ liệu [31].

Việc nhúng được thực hiện thông qua mạng nơ ron. Kết quả của việc nhúng sẽ thu được từ một lớp được gọi là lớp nhúng trong mạng. Cụ thể, chúng chính là bộ trọng số của lớp nhúng [30].

```
blue: (0.01359, 0.00075997, 0.24608, ..., -0.2524, 1.0048, 0.06259)
blues: (0.01396, 0.11887, -0.48963, ..., 0.033483, -0.10007, 0.1158)
orange: (-0.24776, -0.12359, 0.20986, ..., 0.079717, 0.23865, -0.014213)
oranges: (-0.35609, 0.21854, 0.080944, ..., -0.35413, 0.38511, -0.070976)
```

Hình 2. 14: Ví dụ kết quả của nhúng một số từ trong tiếng Anh (word embedding) [30]

Các hàm nhúng là một phương pháp hiệu quả và thường được sử dụng để chuyển đổi các đối tượng rời rạc như các từ trong văn bản về các vectơ liên tục hữu ích. Việc nhúng cũng ánh xạ các đối tượng về vectơ, giúp các ứng dụng có thể sử dụng các phương thức tính khoảng cách không gian vectơ (ví dụ như khoảng cách Euclid) để tính sự tương đồng giữa các đối tượng [30, 19]. Bên cạnh tạo nên đại diện cho các đối tượng rời rạc như các từ, nhúng còn có thể được sử dụng với các đối tượng có nhiều chiều để chuyển đổi chúng về đại diện có số chiều ít hơn [31], điều này sẽ giúp ích cho việc tiếp tục tính toán, xử lý. Đây chính là một trong những lý do mạng VGGish được sử dụng trong phạm vi đồ án này. Việc sử dụng VGGish trong đồ án tức là nhúng dữ liệu âm thanh đã được trích logarit phổ tần số mel về không gian khác thành các vectơ có số chiều nhỏ hơn.

Ví dụ trong phạm vi đồ án với một tín hiệu âm thanh dài 10s, độ dài cửa sổ là 10ms và khoảng cách giữa mỗi lần đặt cửa sổ là 25ms (tương đương 400 mẫu và 160 mẫu với tốc độ lấy mẫu là 16kHz), thì số khung của tín hiệu là 998. Do đó logarit của phổ tần số mel tính toán được sẽ là một mảng hai chiều 998x64, với 64 là số bộ lọc mel. Sau khi lấy mẫu đặc trưng này sẽ thu được một mảng ba chiều 10x96x64, với độ dài cửa sổ và khoảng cách giữa các lần đặt cửa sổ là 960ms (tức là các cửa sổ không đặt chồng lên nhau), tốc độ lấy mẫu là 100Hz, 64 là số bộ lọc mel. Mảng này có tổng cộng 61 440 phần tử nên thực hiện tính toán trực tiếp trên đặc trưng logarit phổ mel thu được trong các thuật toán phân loại sẽ tương đương mất thời gian. Đồng thời việc mảng có ba chiều thay vì chỉ một sẽ khiến cho việc áp dụng các thuật toán phân loại khó hơn vì nhiều thuật toán chỉ chấp nhận vectơ. Mạng VGGish được sử dụng trong đồ án là để ánh xạ đặc trưng logarit phổ mel về đặc trưng 128 phần tử tương ứng với mỗi giây của âm thanh đầu vào. Từ đó đặc trưng này có thể được làm phẳng dễ hơn và đưa vào SVM để tiến hành phân loại âm thanh là tiếng ho hay không phải ho.

```

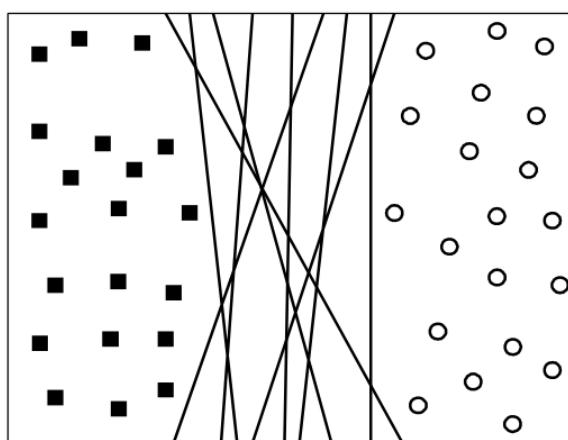
[[0.10866483 0.04020527 0.25316703 ... 0.          0.06762506 0.04453431]
 [0.10416898 0.05015743 0.2527121   ... 0.          0.08081449 0.0631513 ]
 [0.10484773 0.0564881   0.24223402 ... 0.          0.07472788 0.04696748]
 ...
 [0.11414219 0.06137158 0.25479448 ... 0.          0.0713653  0.07017002]
 [0.11238334 0.03793398 0.2664804   ... 0.          0.07233952 0.04293886]
 [0.10532385 0.05245313 0.26535988 ... 0.          0.0793746  0.04525241]]
[[170  9 162 ... 59  25 255]
 [171 10 161 ... 65  13 255]
 [170 10 162 ... 63  16 255]
 ...
 [172 10 162 ... 71  1 255]
 [169 10 162 ... 51  19 255]
 [170 10 162 ... 55  15 255]]

```

Hình 2. 15: Mảng các giá trị của đặc trưng ở đầu ra mạng VGGish cho một file âm thanh trong bộ dữ liệu của đồ án

### 2.3. Thuật toán phân loại SVM

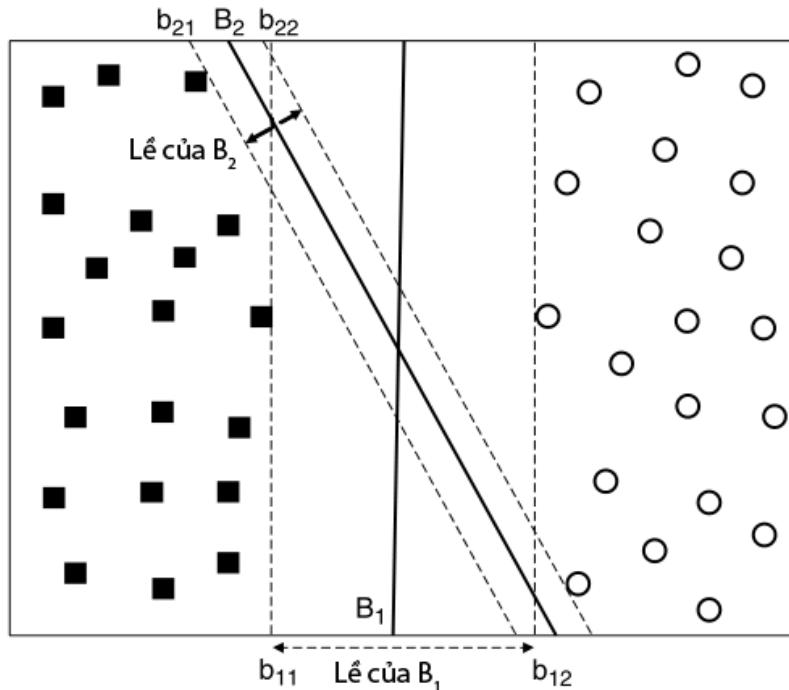
Theo Tan et al. [24], SVM thể hiện đường biên ra quyết định (decision boundary) bằng một nhóm trong những mẫu dữ liệu huấn luyện, được gọi là vectơ hỗ trợ (support vectors), vì thế nên thuật toán có tên là Support Vector Machine. Ý tưởng cơ bản của SVM được dựa trên việc tìm ra siêu mặt phẳng nằm xa các điểm dữ liệu nhất trong bài toán phân loại dữ liệu thuộc hai lớp khác nhau một cách tuyến tính. Dữ liệu được gọi là có thể phân tách một cách tuyến tính (linearly separable) nếu có một siêu phẳng sao cho một lớp dữ liệu nằm hẳn về một phía và lớp còn lại nằm ở phía khác của nó. Tuy nhiên thì có vô số các mặt phẳng như vậy và không phải mặt phẳng nào cũng sẽ thực hiện phân loại tốt với các dữ liệu ngoài dữ liệu huấn luyện. Do đó cần phải tìm ra một siêu phẳng có thể sẽ phân chia dữ liệu tốt nhất.



Hình 2. 16: Các siêu phẳng có thể phân chia dữ liệu có hai lớp [24]

Theo Tan et al. [24], coi mỗi đường biên ra quyết định có hai siêu phẳng gắn với nó là siêu phẳng nằm song song với nó và đi qua điểm dữ liệu thuộc mỗi lớp gần nhất tính từ đường biên. Khi đó khoảng cách giữa hai siêu phẳng này được gọi là lề (margin). Có thể thấy ở hình dưới là biên ra quyết định B1 có lề lớn hơn. Những biên

có lề lớn thường có lỗi ít hơn so với những biên có lề nhỏ bởi vì với lề nhỏ, dữ liệu nằm ngoài dữ liệu huấn luyện chỉ cần lệch một chút về phía biên đã có thể gây ra lỗi sai khi phân loại, đây là hiện tượng quá khớp (over-fitting). Do đó việc lựa chọn siêu mặt phẳng nằm xa các điểm dữ liệu nhất là cần thiết và SVM tuyến tính là một thuật toán có khả năng chọn ra một siêu phẳng như vậy.



Hình 2. 17: Hai đường biên quyết định với lề khác nhau [24]

Xét trong bài toán phân loại dữ liệu thành hai lớp với  $N$  mẫu huấn luyện. Mỗi mẫu sẽ được mô tả gồm  $(x_i, y_i)$  với  $i = 1, 2, \dots, N$  và  $x_i$  là vectơ các thuộc tính của mẫu thứ  $i$ , có  $d$  phần tử là số chiều của vectơ.  $y_i$  sẽ có giá trị  $-1$  hoặc  $+1$ . Thể hiện nhãn phân loại. Đường biên quyết định đưa ra bởi mô hình phân loại sẽ có dạng như sau [24]:

$$w \cdot x + b = 0 \quad (2.19)$$

Với  $w$  và  $b$  là hệ số của mô hình, trong đó  $w$  phải là một vectơ có hướng vuông góc với đường biên quyết định.

Với mỗi điểm dữ liệu  $x_+$  nằm về phía dương của đường biên quyết định, ta có [24]:

$$w \cdot x_+ + b = k, \text{ với } k > 0 \quad (2.20)$$

Tương tự với điểm dữ liệu  $x_-$  nằm về phía âm của đường biên quyết định [24]:

$$w \cdot x_- + b = k', \text{ với } k' < 0 \quad (2.21)$$

Nếu đặt các điểm nằm ở phía dương có nhãn  $+1$  còn ở phía âm có nhãn  $-1$  thì ta có thể dự đoán nhãn của các điểm dữ liệu kiểm tra  $z$  như sau [24]:

$$y = \begin{cases} +1, & \text{nếu } w \cdot z + b > 0 \\ -1, & \text{nếu } w \cdot z + b < 0 \end{cases} \quad (2.22)$$

Nếu thay  $w$  và  $b$  bằng chính chúng nhân với một hệ số nhất định, hai siêu phẳng  $b_1$  và  $b_2$  đi qua điểm dữ liệu gần nhất thuộc hai lớp gắn với đường biên ra quyết định có thể được mô tả như sau mà vẫn giữ nguyên đường biên ban đầu [24, 33]:

$$b_1: w \cdot x + b = 1, \quad (2.23)$$

$$b_2: w \cdot x + b = -1 \quad (2.24)$$

Giả sử  $x_1$  là điểm nằm trên  $b_1$ ,  $x_2$  là điểm nằm trên  $b_2$ , thay vào (2.23) và (2.24) ta được:

$$w \cdot x_1 + b = 1,$$

$$w \cdot x_2 + b = -1$$

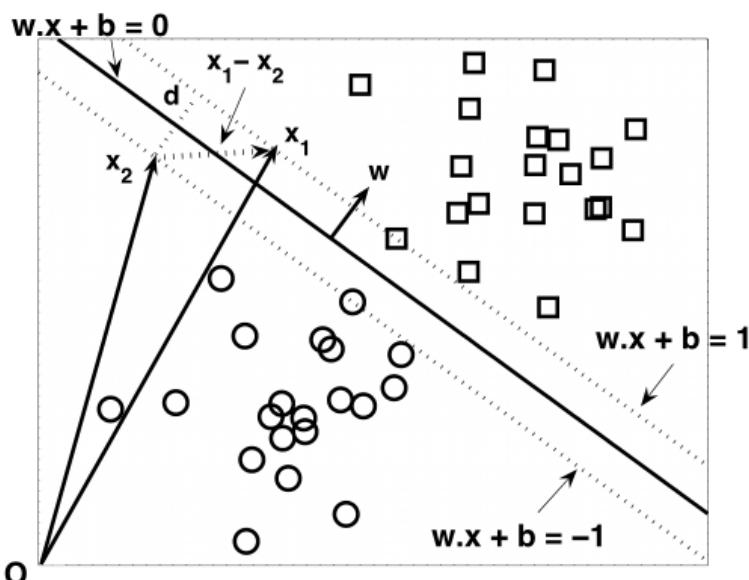
Trừ hai vế cho nhau ta được [24]:

$$w \cdot (x_1 - x_2) = 2$$

$$\Rightarrow \|w\| \times d = 2$$

Với  $d = \sqrt{(x_1 - x_2)^2}$ , tức  $d$  là độ rộng của lề và ta có [24]:

$$d = \frac{2}{\|w\|} \quad (2.25)$$



Hình 2. 18: Ví dụ về đường biên quyết định và lề có độ rộng  $d$  [24]

Bài toán trong SVM chính là bài toán tối ưu  $d$ , hay cũng chính là tìm cực tiểu của hàm sau [24]:

$$f(w) = \frac{\|w\|^2}{2} \quad (2.26)$$

Quá trình học của SVM là để tìm ra hệ số  $w$  và  $b$  sao cho thỏa mãn điều kiện sau [24]:

$$\begin{aligned} w \cdot x_i + b &\geq 1, \text{ nếu } y_i = +1, \\ w \cdot x_i + b &\leq -1, \text{ nếu } y_i = -1, \end{aligned} \quad (2.27)$$

Điều kiện (2.27) có thể được gộp thành [24]:

$$y_i(w \cdot x_i + b) \geq 1, \text{ với } i = 1, 2, \dots, N \quad (2.28)$$

Như vậy bài toán của SVM có thể tổng quát thành bài toán tối ưu như sau [24]:

$$\min_w \frac{\|w\|^2}{2}$$

với điều kiện:  $y_i(w \cdot x_i + b) \geq 1, \text{ với } i = 1, 2, \dots, N$

Đây là một bài toán tối ưu lồi vì hàm mục tiêu và điều kiện ràng buộc đều là hàm lồi, thậm chí hàm mục tiêu là lồi chặt (strictly convex) nên có thể suy ra SVM có nghiệm duy nhất [24, 33]. Theo Tan et al. [24], bài toán tối ưu ở đây có thể giải được bằng phương pháp nhân tử Lagrange. Đầu tiên cần viết lại hàm mục tiêu sao cho bao gồm cả điều kiện ràng buộc. Hàm mục tiêu mới này được gọi là Lagrangian của bài toán tối ưu, công thức như sau [24]:

$$L_P = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \lambda_i (y_i(w \cdot x_i + b) - 1) \quad (2.29)$$

Với  $\lambda_i$  được gọi là nhân tử Lagrange.

Với điều kiện  $\lambda_i \geq 0$ , có thể thấy là với mỗi nghiệm  $w$  và  $b$  mà không thỏa mãn điều kiện ràng buộc, hay  $y_i(w \cdot x_i + b) - 1 < 0$  sẽ làm tăng giá trị của Lagrangian. Để tìm cực tiểu của Lagrangian, cần lấy đạo hàm của nó theo  $w$  và  $b$  và đặt nó bằng 0, tức là như sau [24]:

$$\frac{\partial L_P}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^N \lambda_i y_i x_i, \quad (2.30)$$

$$\frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{i=1}^N \lambda_i y_i = 0 \quad (2.31)$$

Giải bài toán tìm cực tiểu cho Lagrangian ở trên không phải là đơn giản vì nó liên quan tới nhiều biến số  $w$ ,  $b$  và  $\lambda$ . Do đó bài toán đó sẽ được chuyển thành bài toán tìm cực đại của hàm đối ngẫu Lagrangian với các điều kiện ràng buộc  $\lambda$ , hay chính là bài toán đối ngẫu Lagrange [24, 33]. Hàm đối ngẫu Lagrangian là hàm cực tiểu của Lagrangian, thu được khi thay (2.30) và (2.31) vào (2.29), được kết quả như sau [24]:

$$L_D = \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j x_i \cdot x_j \quad (2.32)$$

Như vậy, bài toán đối ngẫu Lagrange là như sau [36]:

$$\max_{\lambda} L_D$$

với điều kiện:  $\lambda_i \geq 0$  với  $i = 1, 2, \dots, N$

$$\sum_{i=1}^N \lambda_i y_i = 0$$

Bài toán trên thỏa mãn hệ điều kiện Karush-Kuhn-Tucker (KKT) như sau [24, 33]:

$$\lambda_i \geq 0 \text{ với } i = 1, 2, \dots, N \quad (2.33)$$

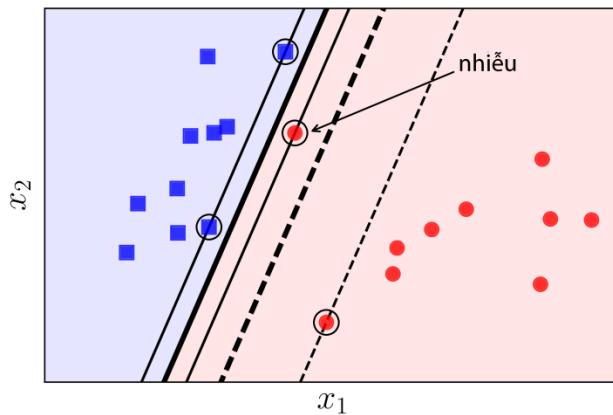
$$\lambda_i(y_i(w \cdot x_i + b) - 1) = 0 \text{ với } i = 1, 2, \dots, N \quad (2.34)$$

Từ điều kiện (2.33) có thể suy ra hoặc  $\lambda_i = 0$  hoặc  $y_i(w \cdot x_i + b) = 1$ . Có thể thấy những điểm thỏa mãn điều kiện thứ hai sẽ nằm trên siêu phẳng song song với đường biên ra quyết định và đi qua điểm dữ liệu gần biên nhất, đồng thời khi đó  $\lambda_i > 0$ . Những điểm như vậy tạo nên vectơ hỗ trợ. Từ công thức (2.30) và (2.34) cũng có thể thấy là  $w$  và  $b$  chỉ phụ thuộc vào tọa độ của các vectơ hỗ trợ và nhân tử Lagrange. Bài toán đối ngẫu Lagrange có thể được giải bằng một số phương pháp như quy hoạch động. Sau khi giải bài toán và tìm được các  $\lambda > 0$ , ta có thể sử dụng công thức (2.30) và (2.34) tìm ra  $w$  và  $b$ . Thông thường các  $\lambda$  được tính toán có thể có lầm tròn, dẫn đến các giá trị  $b$  tìm ra sử dụng công thức (2.34) khác nhau thay vì có một giá trị duy nhất. Do đó trong thực tế, người ta thường sử dụng giá trị  $b$  trung bình. Sau khi đã tính được các giá trị  $w, b$  của đường biên quyết định, một điểm dữ liệu mới có thể được phân loại dựa vào hàm sau [24]:

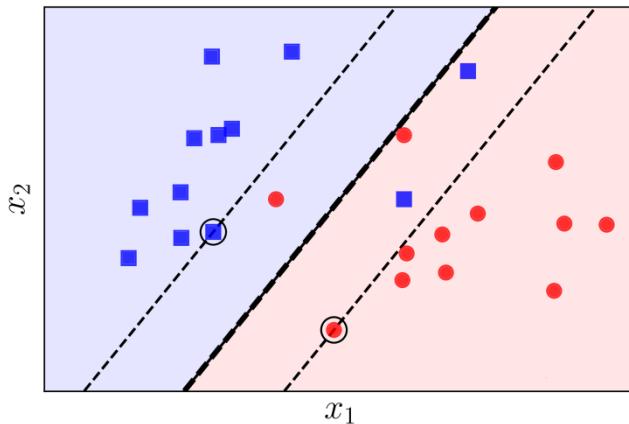
$$f(z) = sign(w \cdot z + b) = sign\left(\sum_{i=1}^N \lambda_i y_i x_i \cdot z + b\right) \quad (2.35)$$

Nếu  $f(z) = 1$ , điểm dữ liệu thuộc lớp dương, ngược lại, điểm dữ liệu thuộc lớp âm.

Tuy nhiên, trường hợp ở trên là với dữ liệu tách biệt hoàn toàn, không có nhiễu. Trong một số trường hợp thực tế, dữ liệu có thể có các điểm thuộc các lớp nằm xen kẽ nhau hoặc có nhiễu nằm sát về phía lớp khác khiến cho đường biên ra quyết định tìm được có lè quá nhỏ hoặc không tìm được.



Hình 2. 19: Dữ liệu có một điểm nhiễu nằm quá gần lớp còn lại [37]



Hình 2. 20: Dữ liệu có các điểm thuộc hai lớp xen kẽ nhau [37]

Trong những trường hợp như thế này, các biên phù hợp được tìm ra (thể hiện bằng đường nét đứt ở các hình minh họa trên) được gọi là biên mềm (soft-margin), khác với biên tìm được theo mô tả ở trên được gọi là biên cứng (hard-margin) ở chỗ nó chấp nhận cho một số điểm dữ liệu huấn luyện nằm trong lề của mình. Để tìm một biên như vậy, thuật toán của SVM cần phải vừa thực hiện được việc tối ưu lề mà vẫn tối thiểu được số lượng các điểm nằm trong lề vì nếu chỉ quan tâm tối ưu lề có thể sẽ khiến thuật toán đưa ra một biên có lề bao trùm lên tất cả các điểm dữ liệu huấn luyện.

Việc tối đa lề vẫn có thể đưa về tối thiểu hàm mục tiêu (2.23) như trong bài toán tìm biên cứng. Tuy nhiên các điều kiện trong bài toán tối thiểu hàm (2.23) không còn có thể thỏa mãn được trong bài toán tìm biên mềm ở đây nên các điều kiện đó cần thay đổi cho phù hợp. Việc này có thể được giải quyết bằng cách sử dụng biến  $\xi$  có giá trị không âm (còn được gọi là slack variable). Với những điểm nằm ngoài lề,  $\xi = 0$ , còn những điểm nằm trong lề,  $\xi > 0$ . Điều kiện ràng buộc sẽ trở thành như sau [24]:

$$\begin{aligned} w \cdot x_i + b &\geq 1 - \xi_i, \text{ nếu } y_i = +1, \\ w \cdot x_i + b &\leq -1 + \xi_i, \text{ nếu } y_i = -1, \end{aligned} \quad (2.36)$$

Điều kiện (2.32) có thể được gộp thành:

$$y_i(w \cdot x_i + b - 1 + \xi_i) \geq 0 \quad (2.37)$$

Với  $\xi_i \geq 0 \forall i$ . Có thể thấy rằng những điểm có  $0 < \xi \leq 1$  là những điểm nằm trong lề nhưng vẫn được phân loại đúng. Còn những điểm có  $\xi > 1$  là những điểm bị phân loại sai. [37]

Hàm mục tiêu cũng cần được thay đổi để đảm bảo tối thiểu số điểm rơi vào trong lề như sau [24]:

$$f(w) = \frac{\|w\|^2}{2} + C \left( \sum_{i=1}^N \xi_i \right)^k \quad (2.38)$$

Với C và k là các giá trị dương được chọn trước, thể hiện những điểm rơi vào trong lề. Để đơn giản, thường k=1 và C có thể được xác định giá trị tốt nhất nhờ kiểm chứng chéo (cross validation). C được dùng để điều chỉnh ưu tiên cho việc tối thiểu số điểm rơi vào trong lề. Nếu C nhỏ, giá trị của các  $\xi$  sẽ không ảnh hưởng nhiều tới giá trị của hàm mục tiêu và thuật toán sẽ tập trung điều chỉnh w sao cho nhỏ nhất, tức là sao cho lề lớn nhất. Tuy nhiên việc này có thể dẫn tới  $\xi$  lớn. Ngược lại nếu C rất lớn, thuật toán sẽ tập trung làm cho tổng các  $\xi$  nhỏ nhất. Nếu dữ liệu có thể được phân lớp một cách tuyến tính, tổng này có thể bằng 0, tức không có điểm nào nằm trong lề và biên tìm được chính là biên cứng.

Bài toán tối ưu biên mềm sẽ là như sau [37]:

$$\min_w \frac{\|w\|^2}{2} + C \sum_{i=1}^N \xi_i$$

với điều kiện:  $y_i(w \cdot x_i + b - 1 + \xi_i) \geq 0$   
 $\xi_i \geq 0 \forall i = 1, 2, \dots, N$

Đây cũng là một bài toán tối ưu lồi [37], Lagrangian của nó là [24]:

$$L_P = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \lambda_i (y_i(w \cdot x_i + b) - 1 + \xi_i) - \sum_{i=1}^N \mu_i \xi_i \quad (2.39)$$

Với  $\mu_i, \lambda_i$  là các nhân tử Lagrange

Lấy đạo hàm bậc nhất của L theo w, b và  $\xi_i$ , cho chúng bằng 0 ta được [24]:

$$\frac{\partial L_P}{\partial w_j} = 0 \Rightarrow w_j = \sum_{i=1}^N \lambda_i y_i x_{ij}, \quad (2.40)$$

$$\frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{i=1}^N \lambda_i y_i = 0 \quad (2.41)$$

$$\frac{\partial L_P}{\partial \xi_i} = 0 \Rightarrow \lambda_i + \mu_i = C \quad (2.42)$$

Thay thế (2.40), (2.41), (2.42) vào Lagrangian ở (2.39) ta được hàm đôi ngẫu [24]:

$$L_D = \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j x_i \cdot x_j \quad (2.43)$$

Các điều kiện KKT thu được khi chuyển các điều kiện ràng buộc về đẳng thức là [24]:

$$\xi_i \geq 0, \lambda_i \geq 0, \mu_i \geq 0 \quad (2.44)$$

$$\lambda_i (y_i (w \cdot x_i + b) - 1 + \xi_i) = 0 \quad (2.45)$$

$$\xi_i \mu_i = 0 \quad (2.46)$$

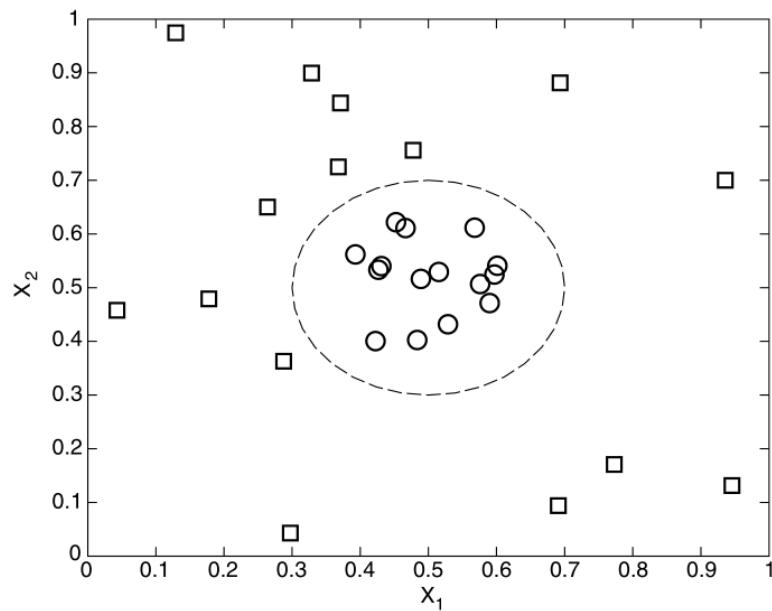
Bài toán đôi ngẫu lúc này là [37]:

$$\begin{aligned} & \max_{\lambda} L_D \\ & \text{với điều kiện: } 0 \leq \lambda_i \leq C \\ & \sum_{i=1}^N \lambda_i y_i = 0 \end{aligned}$$

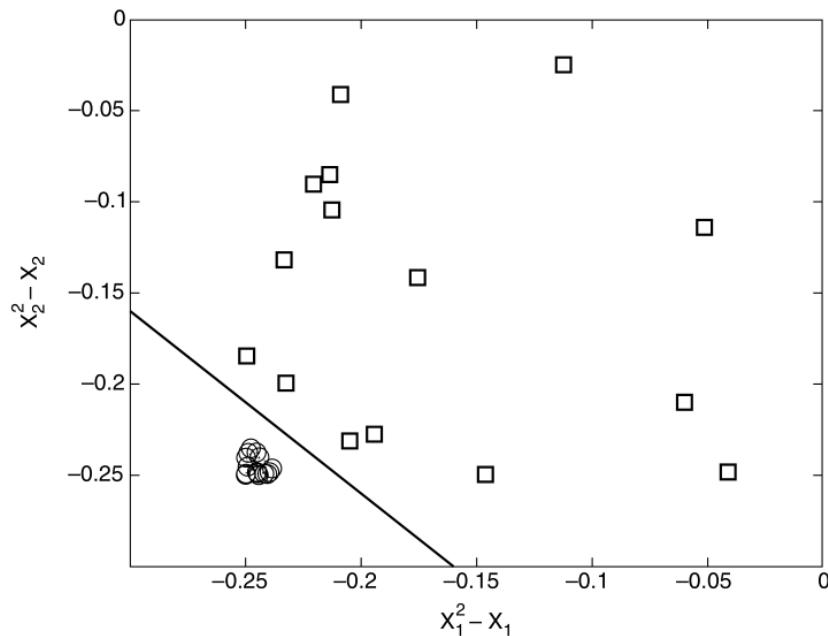
Điều kiện  $0 \leq \lambda_i \leq C$  có được là do (2.44) và (2.46).

Bài toán đôi ngẫu có thể được giải bằng quy hoạch động, cụ thể trong phạm vi đồ án khi cài đặt thuật toán bằng thư viện, thư viện sẽ sử dụng thuật toán tuần tự tối ưu cực tiểu (Sequential minimal optimization – SMO). Kết quả sẽ thu được nhân tử  $\lambda_i$ , sau đó dựa vào (2.40) và (2.45) để tìm  $w$  và  $b$ , tương tự như ở bài toán biên cứng.

Đối với các dữ liệu hoàn toàn không thể phân tách một cách tuyến tính, thuật toán SVM sẽ chuyển dữ liệu sang một chiều không gian khác mà tại đó có thể sử dụng một đường biên tuyến tính để phân lớp dữ liệu. Sau khi chuyển dữ liệu sang không gian như vậy, đường biên quyết định sẽ được tìm ra theo phương pháp đã mô tả ở trên. Tuy nhiên cách tiếp cận này có một vấn đề là việc tính toán sẽ rất khó khăn với các dữ liệu có nhiều chiều và các phép biến đổi như trên thường đưa dữ liệu về có nhiều chiều, thậm chí vô hạn chiều khiến cho việc tính toán trong bài toán tối ưu khó khăn và tốn kém. SVM sẽ giải quyết vấn đề này bằng cách sử dụng các hàm kernel mô tả quan hệ giữa hai điểm dữ liệu trong không gian mà nó sẽ được chuyển sang (phương pháp này còn gọi là kernel trick). [24, 35]



Hình 2. 21: Một bộ dữ liệu có đường biên quyết định không tuyến tính [24]



Hình 2. 22: Bộ dữ liệu trên khi chuyển về miền không gian khác bằng phép biến đổi  $\Phi: (x_1, x^2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$  [24]

Giả sử có một hàm ánh xạ  $\Phi(x)$  có thể chuyển dữ liệu về không gian mà chúng có thể phân tách tuyến tính. Đường biên quyết định trong không gian mới sẽ là [24]:

$$w \cdot \Phi(x) + b = 0 \quad (2.47)$$

Bài toán tối ưu sẽ là [24]:

$$\min_w \frac{\|w\|^2}{2}$$

với điều kiện:  $y_i(w \cdot \Phi(x_i) + b) \geq 1$ , với  $i = 1, 2, \dots, N$

Có thể thấy là so với bài toán gốc thì điểm khác biệt là thuật toán được thực hiện với thuộc tính đã biến đổi  $\Phi(x)$  chứ không phải  $x$  nữa.

Hàm đối ngẫu Lagrangian sẽ là [24]:

$$L_D = \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(x_i) \cdot \Phi(x_j) \quad (2.48)$$

Sau khi tìm ra  $\lambda$ ,  $w$  và  $b$  sẽ được tính bằng các công thức, suy ra từ công thức (2.30) và (2.34) [24]:

$$w = \sum_{i=1}^N \lambda_i y_i \Phi(x_i), \quad (2.49)$$

$$\begin{aligned} \lambda_i (y_i (w \cdot \Phi(x_i) + b) - 1) &= 0 \\ \Rightarrow \lambda_i \left( y_i \left( \sum_j \lambda_j y_j \Phi(x_j) \cdot \Phi(x_i) + b \right) - 1 \right) &= 0 \end{aligned} \quad (2.50)$$

Nhân của một điểm dữ liệu sẽ được xác định bằng công thức sau (suy ra từ công thức (2.35) [24]):

$$f(z) = sign(w \cdot \Phi(z) + b) = sign \left( \sum_{i=1}^N \lambda_i y_i \Phi(x_i) \cdot \Phi(z) + b \right) \quad (2.51)$$

Việc tính toán trực tiếp tích  $\Phi(x_i) \cdot \Phi(x_j)$  trong (2.47), (2.48) và bài toán đối ngẫu sẽ khó khăn và tốn kém cho nên SVM sẽ sử dụng kernel trick. Cụ thể, đặt hàm  $K(u, v) = \Phi(u) \cdot \Phi(v)$ , ta có hàm đối ngẫu trở thành [38]:

$$L_D = \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j K(x_i, x_j) \quad (2.52)$$

Và nhãn của một điểm dữ liệu có thể được xác định bằng dấu của công thức [24]:

$$f(z) = sign \left( \sum_{i=1}^N \lambda_i y_i K(x_i, z) + b \right) \quad (2.53)$$

Theo Pan et al. [24], hàm  $K$  được gọi là hàm kernel. Việc sử dụng hàm kernel giúp cho việc tính toán dễ hơn vì việc tính toán với hàm này rõ ràng là ít khó khăn và tốn kém hơn là tính tích  $\Phi(x_i) \cdot \Phi(x_j)$ . Đồng thời sử dụng hàm kernel cũng giảm đi công việc phải tìm một hàm ánh xạ  $\Phi(x)$  cụ thể. Hàm  $K$  cần phải thỏa mãn điều kiện Mercer để đảm bảo  $K$  luôn có thể mô tả dưới dạng tích vô hướng của hai vectơ. Một số hàm kernel thường được sử dụng có thể kể đến là [38]:

**Hàm tuyến tính:**  $K(x, z) = x \cdot z$

**Hàm Polynomial:**  $K(x, z) = (r + \gamma x \cdot z)^d$ , với d là một số dương để chỉ bậc đa thức

**Hàm Radial Basic Function:**  $K(x, z) = \exp(-\gamma \|x - z\|^2), \gamma > 0$

**Hàm Sigmoid:**  $K(x, z) = \tanh(\gamma x \cdot z + r)$

Theo Pang et al.[24], bài toán của SVM là bài toán tối ưu lồi, nên có thể sử dụng một số thuật toán để tìm ra cực tiểu toàn cục của hàm mục tiêu thay vì chỉ cực tiểu cục bộ ở các thuật toán phân loại dựa trên luật hay mạng nơ ron sử dụng cách tiếp cận tham lam. Trong phạm vi đồ án, thuật toán được sử dụng là SMO, được cài đặt thông qua thư viện có sẵn. SVM ít gặp tình trạng quá khớp, đặc biệt với các dữ liệu nhiều chiều. Tuy nhiên cũng có thể thấy là việc tinh chỉnh để tối ưu SVM có phần khó hơn do cần phải lựa chọn hàm kernel phù hợp.

Trong phạm vi đồ án, hàm kernel, các tham số C, γ sẽ được lựa chọn giá trị phù hợp từ một số giá trị chọn trước cho dữ liệu của đồ án bằng phương pháp kiểm chứng chéo sẽ được mô tả ở chương 3.

#### 2.4. Kết luận

Chương 2 đã trình bày cơ sở lý thuyết cho những thành phần sẽ sử dụng trong giải pháp cho bài toán phân loại tiếng ho của đồ án, bao gồm có đặc trưng sẽ trích từ dữ liệu, phương pháp trích đặc trưng và thuật toán phân loại dữ liệu âm thanh về tiếng ho hay không ho dựa trên đặc trưng trích được.

Trong chương 3, đồ án sẽ trình bày về quá trình tiền xử lý dữ liệu, kịch bản thử nghiệm, cài đặt, đánh giá kết quả thử nghiệm thu được và chương trình ứng dụng mô hình phân loại thu được.

### CHƯƠNG III: THỬ NGHIỆM VÀ ĐÁNH GIÁ

Trong chương 3, đồ án sẽ trình bày về kịch bản thử nghiệm giải pháp đã đề xuất cho bài toán nhận diện tiếng ho, chi tiết việc cài đặt, kết quả thu được, đánh giá kết quả đó và một chương trình nhỏ ứng dụng mô hình phân loại thu được.

#### 3.1. Kịch bản thử nghiệm

Bài toán nhận diện tiếng ho là một trong những bài toán ứng dụng trong lĩnh vực y tế nhằm hỗ trợ trong việc chẩn đoán và giám sát, làm giảm những thiếu sót của con người. Những hệ thống nhận diện tiếng ho có yêu cầu là cần phát hiện chính xác được tiếng ho mỗi khi nó xuất hiện và đồng thời vẫn không nhầm tiếng ho với các âm thanh khác. Việc xác định và trích xuất các đặc trưng có ý nghĩa trong việc phân biệt tiếng ho với các âm thanh khác là một trong những giải pháp để giải quyết yêu cầu này. Đồ án đi theo phương án này và đề xuất sử dụng mạng CNN để trích được những thuộc tính có nghĩa như vậy cùng với thuật toán phân loại SVM để phân biệt các âm thanh ho hay không dựa trên đặc trưng đó.

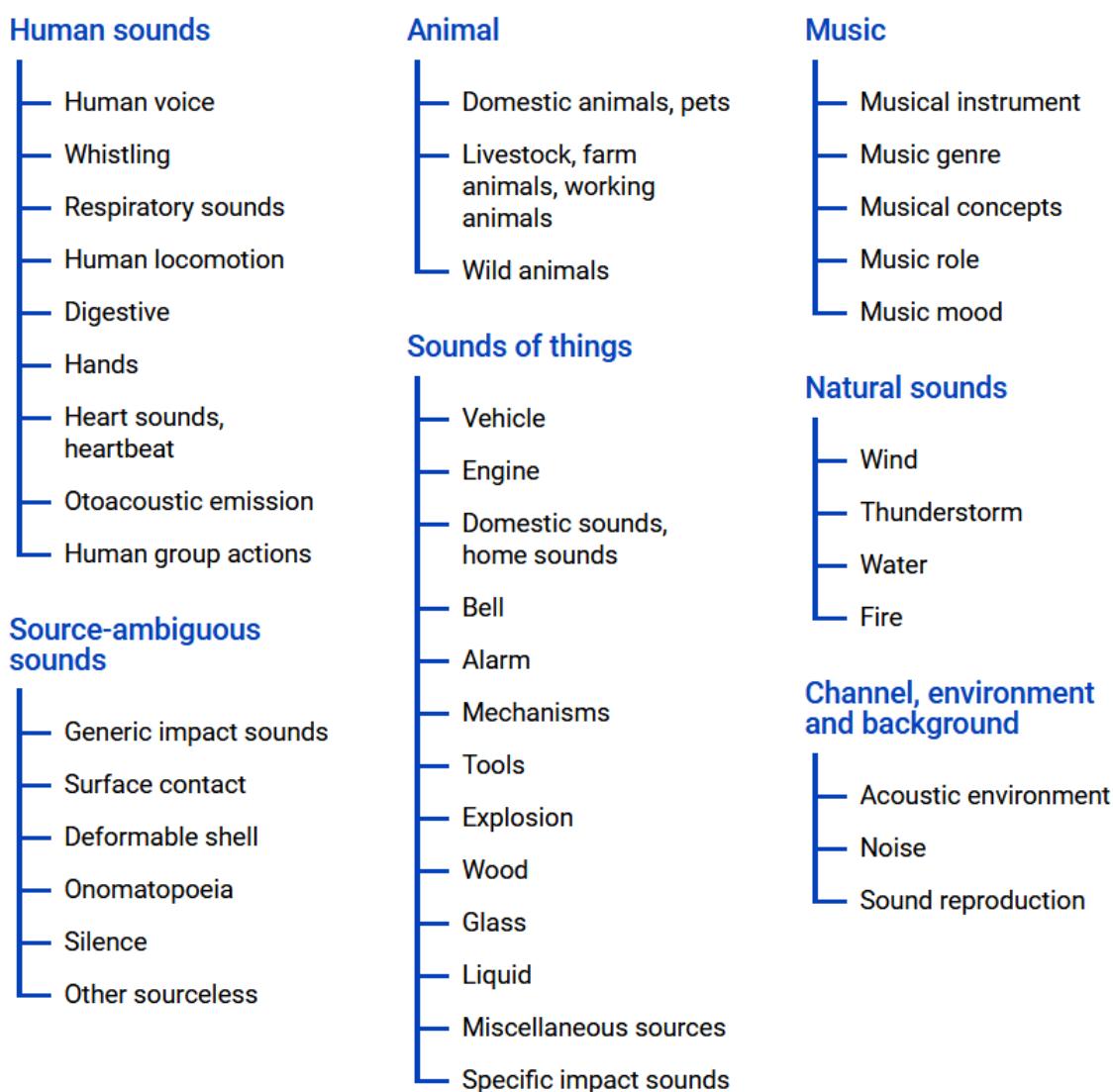
Để thử nghiệm giải pháp đồ án đã đề xuất trong bài toán nhận diện tiếng ho, đồ án thực hiện như sau:

- Tiền xử lý dữ liệu:** Tách các đoạn có tiếng ho và có các âm thanh khác ngoài ho từ các file ghi âm có tiếng ho.
- Trích chọn và lưu trữ đặc trưng:** Trích đặc trưng của các âm thanh này (bao gồm đặc trưng logarit phổ tần số mel và đặc trưng trích bởi VGGish) và âm thanh lấy từ một bộ dữ liệu bên ngoài có tên FSDKaggle2018. Đồng thời, dữ liệu đặc trưng tương tự đã trích sẵn từ bộ dữ liệu AudioSet của các âm thanh ho và không ho cũng được thu thập.
- Huấn luyện thuật toán phân loại:** Tiến hành huấn luyện thuật toán SVM bằng một phần của dữ liệu đã thu được.
- Đánh giá mô hình phân loại đã huấn luyện:** Sử dụng thuật toán SVM đã huấn luyện để thử phân loại phần dữ liệu còn lại và dữ liệu từ bộ dữ liệu bên ngoài FSDKaggle2018, từ đó tính toán các tham số đánh giá hiệu quả của SVM đã huấn luyện.

Dữ liệu sử dụng trong kịch bản bao gồm có ba bộ dữ liệu: Dữ liệu được trích từ các file chứa âm thanh tiếng ho đã loại bỏ nhiễu, có độ dài khác nhau cả của nam và nữ, thuộc nhiều độ tuổi, dữ liệu từ bộ AudioSet và dữ liệu từ bộ FSDKaggle2018. Tất cả bản ghi trong bộ dữ liệu tự trích và FSDKaggle2018 đều có file wav tương ứng để lưu âm thanh được thu âm. Toàn bộ bản ghi trong cả ba bộ dữ liệu đều có file tfrecord để lưu trữ đặc trưng của các đoạn âm thanh tương ứng. Tổng cộng có 155 file wav sử dụng cho bộ dữ liệu tự trích và 60 file đã được lấy từ bộ dữ liệu FSDKaggle2018. Bộ dữ liệu tự trích bao gồm 449 bản ghi tương ứng các đoạn âm thanh ho và 449 bản ghi tương ứng đoạn không chứa tiếng ho. Số lượng bản ghi ho và không ho lấy từ AudioSet đều là 412 bản ghi, với FSDKaggle2018 là 30 bản ghi cho cả hai lớp. Bộ dữ

liệu tự trích và AudioSet sẽ được sử dụng để huấn luyện, đánh giá mô hình phân loại còn bộ dữ liệu FSDKaggle chỉ sử dụng trong đánh giá mô hình.

AudioSet là một bộ dữ liệu về âm thanh của Google, được xây dựng bởi các nhóm Sound and Video Understanding thực hiện các nghiên cứu về giác quan máy tính tại Google. AudioSet sử dụng 527 lớp phân loại các loại âm thanh, bao gồm 2 084 320 các đoạn video dài 10s dán nhãn sẵn bởi con người và được cắt ra từ các video trên YouTube. AudioSet chia các lớp phân loại âm thanh thành các nhóm và các nhóm được tổ chức theo dạng cây thư mục cùng với mô tả về chúng. Các loại âm thanh trong AudioSet bao gồm cả âm thanh tạo bởi con người, động vật cho đến âm thanh tự nhiên hay âm nhạc. Dữ liệu trong AudioSet bao gồm đặc trưng trích từ âm thanh của các video YouTube như đã nói ở trên và các đặc trưng này được lưu trữ dưới dạng các file tfrecord, có thể tải về miễn phí và sử dụng theo giấy phép Creative Commons Attribution 4.0 International. [35, 15]



Hình 3. 1: Tổ chức các lớp phân loại của AudioSet ở hai mức đầu tiên [35]

Freesound Dataset Kaggle 2018 (FSDKaggle2018) là một bộ dữ liệu được thu thập từ trang web freesound.org cho cuộc thi Detection and Classification of Acoustic

Scenes and Events (DCASE) Challenge 2018 nhằm sử dụng trong phần công việc thứ 2 của cuộc thi là phân loại tự động âm thanh. Phần này của cuộc thi được tổ chức trên Kaggle, một nền tảng để tổ chức các cuộc thi học máy, bởi các nhà nghiên cứu từ Music Technology Group thuộc Universitat Pompeu Fabra và từ nhóm nghiên cứu nhận thức máy tính của Google (hay còn được gọi là nhóm Sound and Video Understanding). Bộ dữ liệu gồm 11 073 file âm thanh được phân loại bằng 41 lớp phân loại lấy từ các lớp của AudioSet (mỗi file chỉ được phân vào một lớp). Tất cả mẫu âm thanh trong bộ dữ liệu đều có định dạng wav PCM 16-bit, tần số lấy mẫu 44.1 kHz và là đơn kênh. Lớp phân loại của mỗi file âm thanh có ý nghĩa đánh dấu sự xuất hiện của loại âm thanh đó trong âm thanh lưu trong file. [7]

### 3.1.1. Tiền xử lý dữ liệu

Các bước trong tiền xử lý dữ liệu âm thanh bao gồm: chuyển file âm thanh ban đầu về đơn kênh và loại bỏ DC offset nếu có, cắt các đoạn âm thanh cần thiết từ file âm thanh, lấy mẫu lại với tần số lấy mẫu 16kHz, chèn thêm vào sau các đoạn âm thanh (nếu nó chưa đủ độ dài 10s) và cuối cùng là lưu đoạn âm thanh dưới định dạng wav.

Đầu tiên các file âm thanh sử dụng cho bộ dữ liệu tự trích sẽ được tiền xử lý. Các file âm thanh đó sẽ được tiến hành chuyển đổi về đơn kênh và loại bỏ DC offset rồi từ đó cắt các đoạn âm thanh có tiếng ho. Các đoạn này đều phải có ít nhất là một tiếng ho đầy đủ, không bị cắt ngang. Các đoạn âm thanh khác ngoài tiếng ho cũng được cắt ra từ các file âm thanh ban đầu. Âm thanh trong các đoạn này có thể là âm thanh từ môi trường, tiếng thở, v.v..., tuy nhiên không được chứa bất cứ tiếng ho nào, dù đầy đủ hay không. Các đoạn âm thanh đều được cắt sao cho dài đúng 10s, nếu không thể cắt được với độ dài như thế, đoạn âm thanh sẽ được chèn thêm phần im lặng sau nó để đủ độ dài 10s. Tất cả các đoạn âm thanh này đều sẽ được lấy mẫu lại với tần số lấy mẫu 16 kHz và đều được chuyển về là âm thanh mono (đơn kênh) và được lưu trữ ở định dạng wav. Các file âm thanh lấy từ bộ dữ liệu FSDKaggle cũng được xử lý tương tự. Việc tiền xử lý được thực hiện bởi phần mềm Audacity để tiến hành được việc chuyển các file âm thanh về đơn kênh, loại bỏ DC offset, lấy mẫu lại các đoạn âm thanh và đảm bảo độ dài các file khi cắt.

Tần số lấy mẫu 16 kHz là một trong những tần số được sử dụng phổ biến cho âm thanh số, cụ thể là các tín hiệu giọng nói. Theo định lý lấy mẫu Nyquist–Shannon, tần số lấy mẫu sẽ giới hạn tần số thể hiện được của tín hiệu âm thanh, mà âm thanh con người nghe được nằm trong khoảng từ 20 tới 20 000 Hz. Do đó nếu muốn thể hiện đầy đủ âm thanh nhất có thể, cần sử dụng tần số lấy mẫu lớn hơn 40 000 Hz. Tuy nhiên, âm thanh nói của con người có rất ít âm thanh ở tần số cao nên nếu chỉ cần thể hiện tiếng nói thì không cần tần số lấy mẫu quá cao. Giá trị 16 kHz thường được sử dụng trong trường hợp này và thường là đã đủ đảm bảo thể hiện đầy đủ tiếng nói [25]. Trong bài toán nhận diện tiếng ho, âm thanh ho cũng ít có âm thanh ở tần số cao nên việc sử

dụng tần số lấy mẫu 16 kHz cho các đoạn âm thanh cắt là hợp lý và cũng giúp cho dung lượng file lưu trữ không quá cao.

Âm thanh mono, hay còn gọi là monaural, tức là âm thanh đơn kênh. Âm thanh số mono có thể được phát chỉ bằng một loa và thường được sử dụng trong các trường hợp chỉ cần tập trung vào một âm thanh duy nhất. Trong phạm vi đồ án chỉ tập trung vào tiếng ho trong các đoạn âm thanh ho và coi các âm thanh ngoài ho trong các đoạn âm thanh không phải ho là không đáng kể do nhiều đã được loại bỏ thì việc sử dụng định dạng đơn kênh là phù hợp và đã đủ.

Định dạng wav là một định dạng cho các file âm thanh được phát triển bởi Microsoft và IBM. Đây là định dạng thường được sử dụng trong các hệ thống Microsoft Windows cho các âm thanh gốc, không bị nén. Các file ở định dạng này có thể được chỉnh sửa tương đối dễ dàng bằng các phần mềm. Đây cũng là định dạng được hỗ trợ bởi nhiều phần mềm và thư viện cho nên việc sử dụng định dạng file này để lưu trữ các đoạn âm thanh cũng thuận tiện cho việc tiền xử lý.

Như đã trình bày ở phần 2.1, DC offset là một hiện tượng xảy ra với tín hiệu âm thanh số khi mà cường độ trung bình của tín hiệu là một giá trị khác 0. Hiện tượng này có thể gây nên những hiệu ứng không mong muốn khi tiếp tục xử lý trên tín hiệu nên cần loại bỏ. Do đó việc loại bỏ DC offset cũng được thực hiện trong phạm vi đồ án để đảm bảo chất lượng âm thanh trước khi trích thuộc tính. [28]

Kết quả thu được sau khi tiền xử lý các file âm thanh có sẵn, thu được 449 đoạn tiếng ho và 449 đoạn không chứa tiếng ho.

### 3.1.2. Trích chọn, lưu trữ đặc trưng

Các đoạn âm thanh sau khi được cắt ra sẽ được tính toán logarit phô tần số mel, sau đó đặc trưng này sẽ được đưa vào mạng VGGish để trích đặc trưng và cuối cùng tiến hành phân tích thành phân chính (Principal Components Analysis – PCA). Kết quả thu được là đặc trưng có 128 phần tử trên mỗi giây độ dài đoạn âm thanh (tức là mảng  $128 \times 10$  vì độ dài mỗi đoạn âm thanh đều là 10s). Đặc trưng này được lưu trữ trong file nhị phân dưới định dạng TFRecord để sử dụng trong các bước tiếp theo. Kết quả thu được sau quá trình trích chọn đặc trưng là 449 file TFRecord cho các đoạn âm thanh ho và 449 file cho các đoạn không chứa tiếng ho từ các file âm thanh đã có ban đầu.

Bên cạnh các đặc trưng được trích từ các đoạn âm thanh đã cắt, đồ án còn sử dụng dữ liệu đặc trưng từ bộ dữ liệu AudioSet. Các đặc trưng này cũng được xử lý theo quy trình ở trên, tuy nhiên việc cài đặt có sự khác biệt. Cụ thể, AudioSet là sản phẩm của Google và việc trích đặc trưng cho bộ dữ liệu này sử dụng code sản phẩm nội bộ của họ. Còn code cho việc xử lý các đoạn âm thanh tự thu thập của đồ án, được cung cấp bởi nhóm xây dựng AudioSet thông qua GitHub, thì dùng các thư viện như NumPy và Tensorflow để cài đặt. Các phần code này được đảm bảo là hoạt động tương tự như với code sử dụng cho AudioSet, tuy nhiên kết quả không thể giống hoàn toàn từng con số được. [29]

Đồ án sử dụng 412 bản ghi về âm thanh ho và 412 bản ghi về âm thanh không phải ho (tức các bản ghi có lớp phân loại không bao gồm ho) từ AudioSet, tương ứng với 412 file TFRecord cho âm thanh ho và 412 file cho âm thanh không phải ho.

TFRecord là một định dạng đơn giản sử dụng bởi thư viện TensorFlow để lưu dữ liệu dưới dạng nhị phân. Để lưu dữ liệu dưới định dạng này, dữ liệu thường được chuyển về một trong hai cấu trúc hỗ trợ bởi thư viện TensorFlow là tf.train.Example và tf.train.SequenceExample, sau đó tuần tự hóa nó (serialize) và ghi chuỗi ra file. tf.Example dùng được cho dữ liệu nói chung còn tf.SequenceExample có thể được dùng cho dữ liệu có thuộc tính gồm nhiều giá trị có kiểu giống nhau. Dữ liệu đặc trưng thu được là mảng gồm 128 phần tử tương ứng với mỗi giây, mỗi phần tử đều là số nguyên nên sử dụng tf.SequenceExample sẽ phù hợp hơn. Đây cũng là cấu trúc được sử dụng bởi dữ liệu từ AudioSet nên đồ án sử dụng cấu trúc này để đảm bảo sự tương đồng. tf.Example gồm có danh sách các thuộc tính là features, bao gồm các feature tương ứng với một thuộc tính. Mỗi feature có một cặp tên và giá trị của thuộc tính đó. tf.SequenceExample bao gồm có context cũng có cấu trúc giống như một features và danh sách các thuộc tính là feature\_lists, bao gồm các feature\_list tương ứng với một thuộc tính. Mỗi feature\_list có một cặp tên và giá trị (là một mảng các giá trị) của thuộc tính đó.

```
features {
    feature {
        key: "Age"      (thuộc tính tuổi)
        value {
            int64_list { (kiểu int64)
                value: 29
            }
        }
    }
    feature {
        key: "Movie" (thuộc tính tên phim)
        value {
            bytes_list { (kiểu chuỗi byte)
                value: "The Shawshank Redemption"
                value: "Fight Club"
            }
        }
    }
    feature {           (thuộc tính
        key: "Movie Ratings" điểm đánh giá)
        value {
            float_list { (kiểu float)
                value: 9.0
                value: 9.699999809265137
            }
        }
    }
    feature {
        key: "Purchase Price"(thuộc tính giá)
        value {
            float_list { (kiểu float)
                value: 9.989999771118164
            }
        }
    }
}
```

Hình 3. 2: Ví dụ về nội dung một tf.Example [39]

```
context {
    feature {          (thuộc tính tuổi
        key: "Age"      người xem)
        value {
            int64_list { (kiểu int64)
                value: 19
            }
        }
    }
    feature {          (thuộc tính các
        key: "Favorites" phim yêu thích)
        value {
            bytes_list { (kiểu chuỗi byte)
                value: "Majesty Rose"
                value: "Savannah Outen"
                value: "One Direction"
            }
        }
    }
    feature {
        key: "Locale"(thuộc tính ngôn ngữ)
        value {
            bytes_list { (kiểu chuỗi byte)
                value: "pt_BR"
            }
        }
    }
}
feature_lists {
    feature_list {      (thuộc tính các
        key: "Movie Actors" diễn viên)
        value {
            feature {
                bytes_list { (kiểu chuỗi byte)
                    value: "Tim Robbins"
                    value: "Morgan Freeman"
                }
            }
            feature {
                bytes_list {
                    value: "Brad Pitt"
                    value: "Edward Norton"
                    value: "Helena Bonham Carter"
                }
            }
        }
    }
}
```

Hình 3. 3: Ví dụ nội dung một SequenceExample [39]

```

context: {
  feature: {
    key : "video_id"      (chuỗi id video trên YouTube tương ứng)
    value: {
      bytes_list: {
        value: [YouTube video id string]
      }
    }
  }
  feature: {
    key : "start_time_seconds"   (thời gian bắt đầu của đoạn âm thanh
    value: {                      trong video)
      float_list: {
        value: 6.0
      }
    }
  }
  feature: {
    key : "end_time_seconds"   (thời gian kết thúc của đoạn âm thanh
    value: {                      trong video)
      float_list: {
        value: 16.0
      }
    }
  }
  feature: {                    (phân loại đoạn âm thanh)
    key : "labels"
    value: {
      int64_list: {
        value: [1, 522, 11, 172]
      }
    }
  }
}
feature_lists: {
  feature_list: {                (đặc trưng thu được sau khi trích)
    key : "audio_embedding"
    value: {
      feature: {
        bytes_list: {
          value:
        }
      }
      feature: {
        bytes_list: {
          value:
        }
      }
    }
  ....(độ dài bằng số giây của đoạn âm thanh)
}
}

```

*Hình 3. 4: Ví dụ nội dung của một bản ghi (tương ứng một video) trong AudioSet theo định dạng của tf.SequenceExample [35]*

### 3.1.3. Huấn luyện và đánh giá hiệu quả thuật toán phân loại

Sau khi đã tập hợp được dữ liệu về các đoạn âm thanh ho và không phải ho, bộ dữ liệu được sử dụng cho huấn luyện và đánh sẽ bao gồm ba bộ: Một bộ chỉ bao gồm dữ liệu đặc trưng tự trích và một bộ bao gồm cả dữ liệu đặc trưng tự trích và dữ liệu đặc trưng từ AudioSet và bộ dữ liệu đặc trưng trích từ các file thuộc FSDKaggle2018. Hai bộ dữ liệu đầu tiên được chia thành hai phần, dữ liệu huấn luyện và dữ liệu kiểm tra

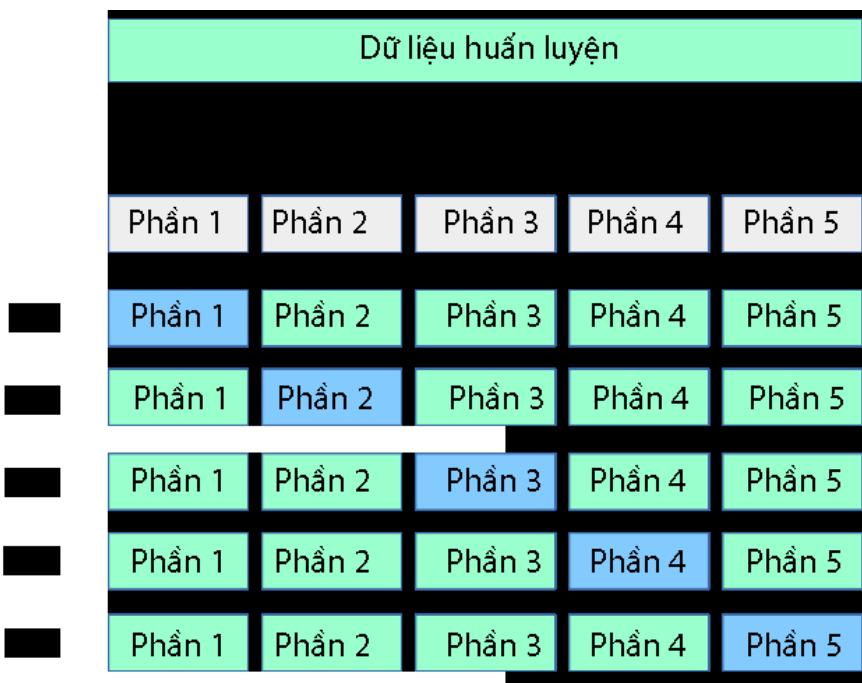
(test) theo tỉ lệ 80:20. Dữ liệu huấn luyện được sử dụng để huấn luyện mô hình phân loại SVM và dữ liệu kiểm tra được sử dụng để kiểm tra và đánh giá hoạt động của mô hình thu được. Cụ thể, bộ dữ liệu tự trích sẽ có 718 bản ghi dành cho dữ liệu huấn luyện và 180 bản ghi dành cho dữ liệu kiểm tra. Đối với bộ dữ liệu tổng hợp dữ liệu tự trích và AudioSet, số lượng này là 1380 và 342. Trong mỗi phần huấn luyện – kiểm tra của mỗi bộ dữ liệu, số lượng bản ghi ho và không ho là bằng nhau. Ví dụ với bộ dữ liệu tự trích, trong phần dữ liệu huấn luyện sẽ có 359 bản ghi là ho và 359 bản ghi không phải ho. Riêng bộ dữ liệu đặc trưng từ FSDKaggle2018 sẽ được sử dụng toàn bộ cho việc kiểm tra.

SVM sẽ được huấn luyện lần lượt với bộ dữ liệu tự trích và bộ dữ liệu tổng hợp của dữ liệu tự trích và AudioSet. Dữ liệu trước khi huấn luyện sẽ được normalize để có giá trị nằm trong khoảng [0,1]. Hai mô hình SVM huấn luyện được sẽ được lưu lại để kiểm tra hiệu quả của chúng bằng dữ liệu kiểm tra.

Trong quá trình huấn luyện SVM, đồ án đồng thời thực hiện việc kiểm chứng bằng phương pháp kiểm chứng chéo kết hợp với tìm kiếm theo lưới (grid search) để thu được mô hình với hệ số hoạt động hiệu quả nhất. Mô hình SVM có một số hệ số cần đặt trước như  $C$ ,  $\gamma$  và loại hàm kernel nên trong quá trình huấn luyện cần tìm những giá trị tốt nhất có thể. Thuật toán tìm kiếm theo lưới được sử dụng là để phục vụ cho việc này. Đồng thời khi thực hiện tinh chỉnh hệ số cần có một tập dữ liệu ngoài những dữ liệu đã dùng để huấn luyện để đánh giá các hệ số đã lựa chọn. Việc sử dụng tập dữ liệu kiểm tra cho việc này có thể dẫn đến hiện tượng quá khớp nên thường sẽ sử dụng một bộ dữ liệu nữa gọi là dữ liệu kiểm chứng (validation) được tách từ bộ dữ liệu huấn luyện. Trong phạm vi đồ án, phương pháp kiểm chứng chéo sẽ được sử dụng thực hiện việc này mà không làm giảm kích thước bộ dữ liệu huấn luyện quá mức.

## a) Kiểm chứng chéo (cross validation)

Kiểm chứng chéo là một trong những phương pháp thường được sử dụng trong việc đánh giá hoạt động của một mô hình phân loại. Phương pháp này sẽ chia dữ liệu huấn luyện thành  $k$  phần bằng nhau. Sau đó với mỗi lần chạy, một phần dữ liệu sẽ được để lại để tiến hành kiểm chứng hoạt động của mô hình sau khi huấn luyện bằng các phần dữ liệu còn lại. Quá trình này sẽ lặp lại  $k$  lần để mỗi phần đều được sử dụng cho việc kiểm chứng đúng một lần. Đồng thời, giá trị tham số để kiểm chứng hoạt động của mô hình có thể được tính toán ở mỗi lần chạy như vậy và từ đó tính được giá trị trung bình của nó để đánh giá hoạt động chung cho mô hình. [24, 39]

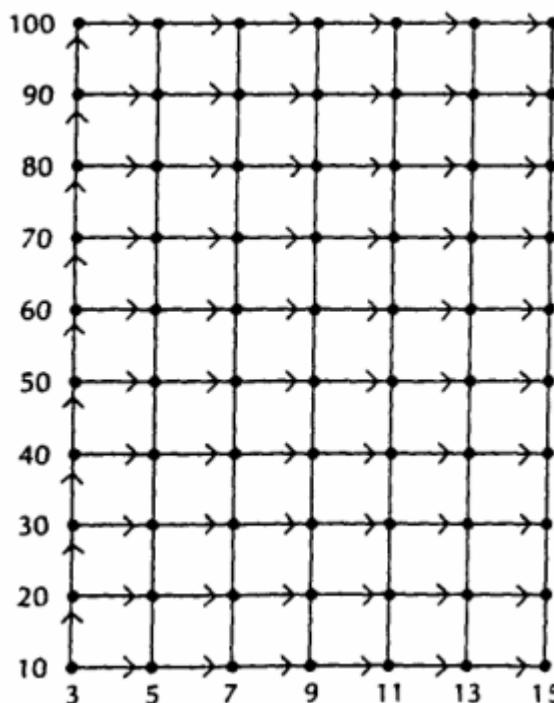


*Hình 3. 5: Ví dụ khi các phần dữ liệu được chia làm 5 phần, màu xanh lá là phần dữ liệu huấn luyện, phần xanh dương là phần dữ liệu kiểm chứng [40]*

Rõ ràng có thể thấy phương pháp này sẽ mất thời gian tính toán, tuy nhiên sẽ đảm bảo tận dụng được tối đa dữ liệu đang có, không làm giảm kích thước dữ liệu huấn luyện quá nhiều. Trong trường hợp của đồ án, lượng dữ liệu dành cho việc huấn luyện là nhỏ (1380 bản ghi) nên việc sử dụng phương pháp này là cần thiết. Đồng thời việc cài đặt phương pháp này trong quá trình huấn luyện cũng sẽ được thực hiện trên môi trường đủ khả năng thực thi với thời gian không quá lâu. Môi trường này sẽ được trình bày cụ thể ở phần 3.2

#### b) Tìm kiếm theo lưới (grid search)

Tìm kiếm theo lưới là một thuật toán tìm kiếm vét cạn và là một phương pháp đơn giản trong việc tìm kiếm các bộ hệ số sẽ hoạt động hiệu quả dựa trên một tham số nào đó. Cụ thể, tìm kiếm theo lưới sẽ dựa trên một số giá trị hay khoảng giá trị của các hệ số cho trước để tạo thành một bộ giá trị cho các hệ số. Tham số đánh giá hoạt động của mô hình mang bộ hệ số sẽ được tính toán với mỗi lần sử dụng bộ hệ số đó. Thuật toán sẽ chạy với tất cả bộ giá trị của các hệ số có thể tìm được. [18]



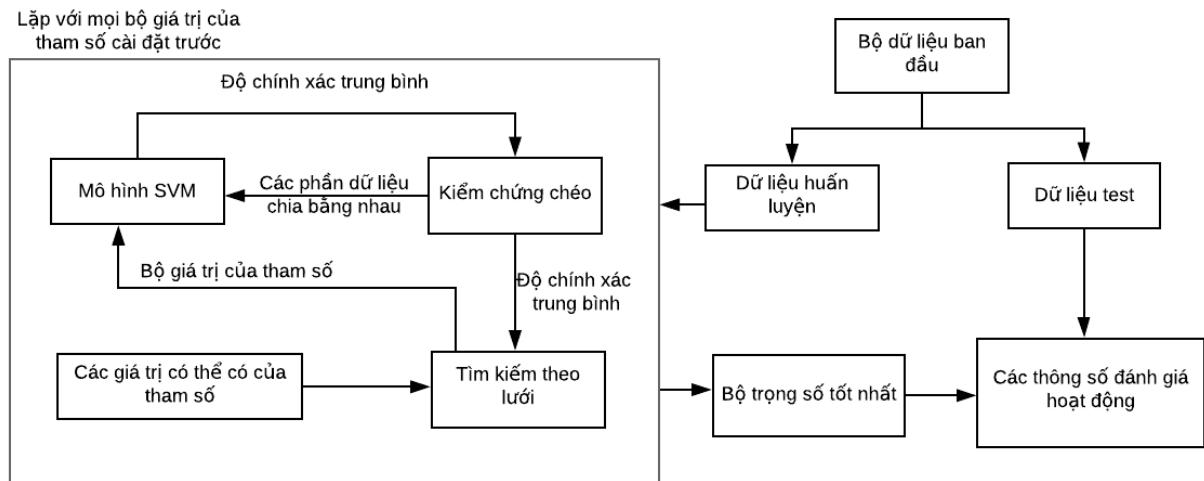
Hình 3. 6: Ví dụ lưới tìm kiếm với hai hệ số, giá trị các hệ số gồm (10, 20, 30, 40, 50, 60, 70, 80, 90, 100) và (3, 5, 7, 9, 11, 13, 15) [18]

Thuật toán này có thể có thời gian tính toán lâu, đặc biệt nếu số lượng tham số nhiều. Tuy nhiên, đây là một thuật toán đơn giản và có thể có độ tin cậy hơn so với các thuật toán sử dụng hàm heuristic hay sử dụng phương pháp xấp xỉ, đặc biệt trong trường hợp số lượng tham số nhỏ (ví dụ như 1-2 tham số). Đồng thời việc tìm kiếm có thể tiến hành thực hiện song song vì các tham số C và  $\gamma$  độc lập với nhau [5]. Trong phạm vi đồ án, việc tìm kiếm sẽ thực hiện với hai lưới, một bao gồm hai tham số là C và loại hàm kernel và lưới còn lại gồm ba tham số là C,  $\gamma$  và loại hàm kernel. Các giá trị của hệ số C là [1, 10, 100, 1000, 1e5], của hàm kernel là polynomial, radial basic function (rbf), sigmoid, tuyến tính và của  $\gamma$  (sử dụng cho các hàm polynomial, radial basic function, sigmoid) nằm trong khoảng [1e-3, 1e-4].

Việc thực hiện sẽ tốn thời gian do có một lưới gồm ba tham số, tuy nhiên số lượng này chưa quá lớn nên sẽ không quá lâu. Đồng thời việc cài đặt thuật toán này cho quá trình huấn luyện cũng là trên môi trường sẽ cài đặt kiểm chứng chéo, đủ khả năng thực thi với thời gian không quá lâu.

Trong phạm vi đồ án, kiểm chứng chéo sẽ chia dữ liệu huấn luyện thành 10 phần sao cho số bản ghi ho và không ho bằng nhau trong mỗi phần. Các phần dữ liệu này sẽ được sử dụng trong tìm kiếm theo lưới để đánh giá hiệu quả hoạt động của các bộ giá trị của hệ số C,  $\gamma$  và loại hàm kernel theo tham số là độ chính xác (accuracy). Cụ thể với mỗi bộ giá trị, mô hình sẽ được huấn luyện và tính accuracy dựa trên bộ dữ liệu huấn luyện – đánh giá đã chia bởi kiểm chứng chéo. Từ đó accuracy trung bình của mô hình với bộ giá trị hệ số được tính trên 10 lần chạy. Từ đó bộ hệ số có accuracy cao nhất sẽ được chọn.

Sau khi hoàn tất quá trình huấn luyện, mô hình thu được sẽ được sử dụng để phân loại dữ liệu kiểm tra và dữ liệu từ bộ dữ liệu FSDKaggle2018. Dữ liệu được đưa vào mô hình SVM để phân loại cũng được normalize tương tự dữ liệu huấn luyện. Trong khi phân loại, các tham số đánh giá hiệu quả của mô hình sẽ được tính toán. Cụ thể các tham số được dùng và giá trị của chúng với ba mô hình sẽ thu được sẽ được trình bày ở phần 3.3



Hình 3.7: Quy trình huấn luyện và đánh giá hoạt động của mô hình SVM trong đồ án

### 3.2. Cài đặt thử nghiệm

Việc cài đặt thử nghiệm được thực hiện trên môi trường máy ảo cung cấp bởi Google Colaboratory. Thông số cụ thể của môi trường được mô tả cụ thể trong bảng 3.1:

Môi trường máy ảo của Google Colaboratory	
Vị xử lý	Intel Xeon 2.30GHz 2 luồng
Dung lượng RAM	13GB
Dung lượng bộ nhớ	49 GB
Hệ điều hành	Ubuntu 18.04

Bảng 3.1: Môi trường cài đặt thử nghiệm của đồ án

Google Colaboratory là một môi trường Jupyter notebook miễn phí không yêu cầu cài đặt và hoạt động toàn bộ trên cloud. Jupyter Notebook là một ứng dụng chạy trên web cho phép người dùng tạo ra các văn bản, thường được gọi là notebook, bao gồm các đoạn code và đầu ra khi chạy chúng (có thể là chữ, hình ảnh, video, biểu đồ, v.v...). Jupyter Notebook hỗ trợ nhiều ngôn ngữ, bao gồm Python, R, Julia,... Google Colaboratory được xây dựng dựa trên Jupyter Notebook, tuy nhiên có điểm khác là cung cấp tài nguyên tính toán mạnh bao gồm: dung lượng RAM và bộ nhớ lớn, vi xử lý có hiệu năng tốt và GPU hỗ trợ tính toán. Tuy nhiên Google Colab hiện đang chỉ hỗ trợ ngôn ngữ Python. Các tài liệu chia sẻ trên Google Colaboratory đều là một notebook, nhưng được chạy trên cloud của Google Colaboratory và lưu trữ trong Google Drive tương ứng với tài khoản Google được sử dụng để tạo notebook. Mọi

môi trường của Colabotary có thể liên kết với Google Drive của người dùng để đọc và ghi các file từ đó. Môi trường có sẵn một số thư viện Python, bao gồm cả các thư viện dùng cho học máy phổ biến như TensorFlow hay scikit-learn.

Để phục vụ cho việc cài đặt và chạy các thành phần chính trong thử nghiệm (mạng VGGish và thuật toán SVM), đồ án sử dụng hai thư viện là TensorFlow và scikit-learn trên môi trường Colabotary.

## a) TensorFlow

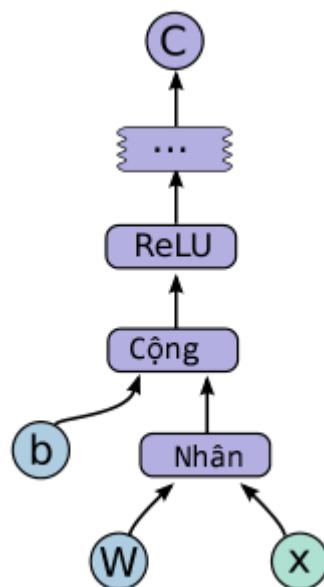
TensorFlow là một thư viện mã nguồn mở sử dụng cho lĩnh vực học máy được phát triển bởi nhóm Google Brain. API của TensorFlow được cung cấp dưới dạng mã nguồn mở từ tháng 12/2015 dưới giấy phép Apache 2.0. TensorFlow có thể sử dụng được trên nhiều nền tảng. TensorFlow có thể tham gia thực hiện suy luận trên các nền tảng di động như Android, iOS cho đến chạy các hệ thống huấn luyện và suy luận cỡ vừa với một thiết bị có một hoặc nhiều CPU hoặc hệ thống cỡ lớn với hàng trăm thiết bị chuyên dụng và hàng nghìn GPU. Thư viện TensorFlow được mô tả là linh động và có thể được sử dụng để mô tả nhiều thuật toán, bao gồm các thuật toán huấn luyện và suy luận cho các mạng học sâu. TensorFlow mô tả việc tính toán bằng một sơ đồ có hướng gồm nhiều nút để thể hiện việc tính toán với một luồng dữ liệu, sau đó chuyển chúng vào chạy trên các nền tảng. Sơ đồ này có thể được người dùng xây dựng bằng các ngôn ngữ front-end như Python hay C++. Thư viện có phần lõi được viết bằng C++ để giúp cho việc cài đặt các mô hình huấn luyện dễ hơn khi phải thực hiện trên nhiều môi trường khác nhau. [3]

```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))                      # Vector 100 phần tử 0
W = tf.Variable(tf.random_uniform([784,100],-1,1))    # Ma trận 784x100 giá trị random
x = tf.placeholder(name="x")                          # Biến chứa đầu vào
relu = tf.nn.relu(tf.matmul(W, x) + b)               # Relu(Wx+b)
C = [...]                                              # Hàm mất mát của Relu
# 

s = tf.Session()
for step in xrange(0, 10):
    input = ...construct 100-D input array ...          # Tạo vector 100 phần tử
    result = s.run(C, feed_dict={x: input})            # Lưu lại giá trị hàm mất mát,
    print step, result                                 # x là đầu vào
```

Hình 3. 8: Ví dụ một đoạn code sử dụng TensorFlow để xây dựng và thực thi một sơ đồ tính hàm ReLu viết bằng Python [3]



Hình 3. 9: Minh họa sơ đồ được tạo bởi đoạn code ở Hình 3.7 [3]

Đồ án sử dụng phiên bản r1 của thư viện Tensorflow để cài đặt thử nghiệm.

### b) scikit-learn

scikit-learn là một thư viện học máy miễn phí cho ngôn ngữ Python, được phát hành chính thức lần đầu vào 1/2/2010 bởi Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort và Vincent Michel làm việc tại INRIA (Viện nghiên cứu Khoa học Máy tính và Tự động hóa quốc gia của Pháp). Thư viện hướng tới việc giúp những người không chuyên tiếp cận với học máy và tập trung vào tính dễ sử dụng, hiệu năng, tài liệu hướng dẫn và tính thống nhất của API. scikit-learn được viết chủ yếu bằng Python nhưng cũng sử dụng các thư viện C++, bao gồm LibSVM và LibLinear đóng vai trò cung cấp cách cài đặt tham khảo cho SVM và các mô hình tuyến tính tổng quát. Scikit-learn có thể chạy trên nhiều nền tảng, bao gồm nền tảng Windows và POSIX. scikit-learn bao gồm các thuật toán học máy mới nhất cho các bài toán học không có giám sát và học có giám sát cỡ vừa. [19]

Đồ án sử dụng phiên bản 0.21.3 của thư viện scikit-learn để cài đặt thử nghiệm.

Việc cài đặt thử nghiệm sẽ được chia ra làm ba mô đun nhỏ là: trích chọn đặc trưng, huấn luyện và sử dụng thuật toán phân loại. Tiếp theo đồ án sẽ mô tả cụ thể việc sử dụng các lý thuyết trình bày ở chương II và các thư viện đã nêu trong hai mô đun này.

#### 3.2.2. Cài đặt trích chọn đặc trưng

Như đã nói ở phần 3.1.2, việc trích chọn đặc trưng gồm có tính toán logarit phổ tần số mel từ file âm thanh ban đầu (coi như là tiền xử lý), trích chọn đặc trưng từ đó và hậu xử lý bằng biến đổi PCA. Để thực hiện việc này, đồ án sử dụng code tiền xử lý và hậu xử lý, mạng VGGish được cung cấp miễn phí bởi nhóm phát triển AudioSet thông qua repository *models* của TensorFlow trên GitHub.

Việc tiền xử lý bao gồm có: đảm bảo yêu cầu với file âm thanh (âm thanh đơn kênh, tốc độ lấy mẫu 16kHz), tính toán logarit phổ mel theo các bước đã trình bày ở phần

2.1 và lấy mẫu chuỗi giá trị thu được. Việc cài đặt có sử dụng thư viện numpy để tính toán và các thư viện cho việc đọc, xử lý âm thanh bao gồm resampy, soundfile.

```
# Chuyển về mono
if len(data.shape) > 1:
    data = np.mean(data, axis=1)
# Resample to the rate assumed by VGGish.
if sample_rate != vggish_params.SAMPLE_RATE:
    data = resampy.resample(data, sample_rate, vggish_params.SAMPLE_RATE)

# Tính đặc trưng logarit phổ tần số mel
log_mel = mel_features.log_mel_spectrogram(
    data,
    audio_sample_rate=vggish_params.SAMPLE_RATE,
    log_offset=vggish_params.LOG_OFFSET,
    window_length_secs=vggish_params.STFT_WINDOW_LENGTH_SECONDS,
    hop_length_secs=vggish_params.STFT_HOP_LENGTH_SECONDS,
    num_mel_bins=vggish_params.NUM_MEL_BINS,
    lower_edge_hertz=vggish_params.MEL_MIN_HZ,
    upper_edge_hertz=vggish_params.MEL_MAX_HZ)

# Khung đặc trưng
features_sample_rate = 1.0 / vggish_params.STFT_HOP_LENGTH_SECONDS
example_window_length = int(round(
    vggish_params.EXAMPLE_WINDOW_SECONDS * features_sample_rate))
example_hop_length = int(round(
    vggish_params.EXAMPLE_HOP_SECONDS * features_sample_rate))
log_mel_examples = mel_features.frame(
    log_mel,
    window_length=example_window_length,
    hop_length=example_hop_length)
return log_mel_examples
```

Hình 3. 10: Phần code cho tiền xử lý cung cấp bởi nhóm phát triển AudioSet [29]

Việc trích đặc trưng được thực hiện bởi mạng VGGish đã được trình bày ở phần 2.2.2. Đồ án sử dụng mô hình VGGish cài đặt bằng TensorFlow cung cấp bởi nhóm phát triển AudioSet. Định nghĩa cho mô hình theo định dạng TensorFlow Slim được mô tả ở Hình 3.11. Việc sử dụng VGGish để trích thuộc tính được mô tả ở Hình 3.12.

```

# Tất cả bias ban đầu bằng 0
# Hàm ReLU sử dụng làm hàm kích hoạt cho tất cả
# Nhân chập với cửa sổ 3x3 và bước nhảy 1
# Gộp với hàm max bằng cửa sổ 2x2, bước nhảy 2x2
with slim.arg_scope([slim.conv2d, slim.fully_connected],
                     weights_initializer=tf.truncated_normal_initializer(
                         stddev=params.INIT_STDDEV),
                     biases_initializer=tf.zeros_initializer(),
                     activation_fn=tf.nn.relu,
                     trainable=training), \
    slim.arg_scope([slim.conv2d],
                  kernel_size=[3, 3], stride=1, padding='SAME'), \
    slim.arg_scope([slim.max_pool2d],
                  kernel_size=[2, 2], stride=2, padding='SAME'), \
    tf.variable_scope('vggish'):
    # Đầu vào là mảng 2 chiều logarit phổ tần ssô Mel
    features = tf.placeholder(
        tf.float32, shape=(None, params.NUM_FRAMES, params.NUM_BANDS),
        name='input_features')
    # Chuyển thành 4 chiều để nhân chập với conv2d()
    net = tf.reshape(features, [-1, params.NUM_FRAMES, params.NUM_BANDS, 1])

    # Các lớp của mạng
    net = slim.conv2d(net, 64, scope='conv1')
    net = slim.max_pool2d(net, scope='pool1')
    net = slim.conv2d(net, 128, scope='conv2')
    net = slim.max_pool2d(net, scope='pool2')
    net = slim.repeat(net, 2, slim.conv2d, 256, scope='conv3')
    net = slim.max_pool2d(net, scope='pool3')
    net = slim.repeat(net, 2, slim.conv2d, 512, scope='conv4')
    net = slim.max_pool2d(net, scope='pool4')

    # Flatten trước khi đi vào các lớp kết nối đầy đủ
    net = slim.flatten(net)
    net = slim.repeat(net, 2, slim.fully_connected, 4096, scope='fc1')
    # Lớp nhúng
    net = slim.fully_connected(net, params.EMBEDDING_SIZE, scope='fc2')
return tf.identity(net, name='embedding')

```

Hình 3. 11: Định nghĩa mô hình VGGish theo kiến trúc đã mô tả ở phần 2.2 [29]

```
with tf.Graph().as_default(), tf.Session() as sess:  
    # Định nghĩa mô hình để suy diễn, tải checkpoint và xác định tensor vào, ra  
  
    vggish_slim.define_vggish_slim(training=False)  
    vggish_slim.load_vggish_slim_checkpoint(sess, FLAGS.checkpoint)  
    features_tensor = sess.graph.get_tensor_by_name(  
        vggish_params.INPUT_TENSOR_NAME)  
    embedding_tensor = sess.graph.get_tensor_by_name(  
        vggish_params.OUTPUT_TENSOR_NAME)  
  
    # Để mạng suy diễn  
    [embedding_batch] = sess.run([embedding_tensor],  
                               feed_dict={features_tensor: examples_batch})
```

Hình 3. 12: Mô tả việc sử dụng mạng VGGish trích thuộc tính từ example\_batch là mảng thu được sau khi tiền xử lý [29]

Việc hậu xử lý được thực hiện bằng biến đổi PCA. Các hệ số sử dụng trong PCA cũng được nhóm phát triển AudioSet cung cấp dưới dạng một file npz. Việc cài đặt biến đổi này sử dụng thư viện numpy trong việc thực hiện các tính toán.

```
# Áp dụng PCA  
pca_applied = np.dot(self._pca_matrix,  
                      (embeddings_batch.T - self._pca_means)).T  
  
# Lượng tử hóa:  
# Chuyển đổi giá trị về nằm trong khoảng [min, max]  
clipped_embeddings = np.clip(  
    pca_applied, vggish_params.QUANTIZE_MIN_VAL,  
    vggish_params.QUANTIZE_MAX_VAL)  
# Chuyển đổi về 8-bit trong khoảng [0.0 - 255.0]  
quantized_embeddings = (  
    (clipped_embeddings - vggish_params.QUANTIZE_MIN_VAL) *  
    (255.0 /  
     (vggish_params.QUANTIZE_MAX_VAL - vggish_params.QUANTIZE_MIN_VAL)))  
# Chuyển số 8-bit dưới từ float về uint8  
quantized_embeddings = quantized_embeddings.astype(np.uint8)  
  
return quantized_embeddings
```

Hình 3. 13: Phần code cho quá trình hậu xử lý đặc trưng cung cấp bởi nhóm phát triển AudioSet [29]

Bên cạnh việc tiền xử lý, hậu xử lý và cài đặt mạng VGGish, thư viện TensorFlow cũng được sử dụng để đọc và ghi các file TFRecord do đây là định dạng chỉ sử dụng bởi thư viện này. Việc đọc file TFRecord là để nhằm tìm các bản ghi được phân loại là ho trong bộ dữ liệu của AudioSet, được thực hiện như ở hình 3.14. Việc ghi đặc trưng

ra file TFRecord được thực hiện sau khi trích đặc trưng bởi mạng VGGish để lưu trữ chúng cho việc sử dụng nhiều lần trong quá trình huấn luyện và kiểm tra.

```
# Sử dụng tf.data.TFRecordDataset để đọc
raw_data = tf.data.TFRecordDataset(dir+file_name)

# Các thuộc tính trong context
context_features = {
    # 'video_id': tf.io.FixedLenFeature([1], dtype=tf.string),
    # 'start_time_seconds': tf.io.FixedLenFeature([1], dtype=tf.float32),
    # 'end_time_seconds': tf.io.FixedLenFeature([1], dtype=tf.float32),
    'labels': tf.io.VarLenFeature(dtype=tf.int64)
}

# Các thuộc tính trong đặc trưng
sequence_features = {
    'audio_embedding': tf.io.VarLenFeature(dtype=tf.string)
}

def _parse_function(example_proto):
    # Parse the input tf.Example proto using the dictionary above.
    parsed_example = tf.io.parse_single_sequence_example(
        example_proto,
        context_features,
        sequence_features)
    return parsed_example

parsed_data = raw_data.map(_parse_function)
return parsed_data
```

Hình 3. 14: Phần code cho việc đọc file TFRecord

```
    seq_example = tf.train.SequenceExample(
        feature_lists=tf.train.FeatureLists(
            feature_list={
                vggish_params.AUDIO_EMBEDDING_FEATURE_NAME:
                    tf.train.FeatureList(
                        feature=[
                            tf.train.Feature(
                                bytes_list=tf.train.BytesList(
                                    value=[embedding.tobytes()])
                                for embedding in postprocessed_batch
                            )
                        ]
                    )
                }
            )
        )
    print(seq_example)
    if writer:
        writer.write(seq_example.SerializeToString())
```

Hình 3. 15: Phần code ghi các đặc trưng trích bởi VGGish ra file TFRecord với tên được chỉ định sẵn, writer là một object thuộc `tf.python_io.TFRecordWriter` dùng để ghi với tên file đó [29]

### 3.2.3. Cài đặt huấn luyện và sử dụng thuật toán phân loại

Thuật toán huấn luyện sử dụng cho bài toán nhận diện tiếng ho trong phạm vi đồ án là thuật toán SVM, chi tiết đã được trình bày ở phần 2.3. Để cài đặt thuật toán, cụ thể là quá trình huấn luyện, sử dụng nó để phân loại và việc chuẩn hóa dữ liệu trước khi đưa vào thuật toán, đồ án sử dụng thư viện scikit-learn. Quá trình huấn luyện, bao gồm cả kiểm chứng chéo và tìm kiếm theo lưới cho các hệ số cài đặt trước của SVM, được cài đặt như ở Hình 3.16. Mô hình huấn luyện được sẽ được lưu lại dưới dạng file pkl sử dụng thư viện pickle và do đó khi tiến hành sử dụng mô hình, pickle cũng được sử dụng để đọc mô hình từ file. Mô hình đã được huấn luyện thông qua thư viện scikit-learn cũng sẽ được sử dụng để phân loại bằng hàm predict có trong thư viện này.

```
# Huấn luyện đồng thời tìm bộ tham số tốt nhất
print("Training model...")
hyperparams = [{"kernel": ["poly"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000, 1e5]},
               {"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000, 1e5]},
               {"kernel": ["sigmoid"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000, 1e5]},
               {"kernel": ["linear"], "C": [1, 10, 100, 1000, 1e5]}]
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=0)
grid_search = GridSearchCV(SVC(), hyperparams, cv=skf, scoring='accuracy')
grid_search.fit(data, label)
# In kết quả hoạt động của mỗi tham số, trả về tham số có accuracy cao nhất
print("Best parameters found: ")
print(grid_search.best_params_)
print("Detailed scores on each set: ")
means = grid_search.cv_results_['mean_test_score']
stds = grid_search.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, grid_search.cv_results_['params']):
    print("{0:.3f} (+-{1:.03f}) for {2}".format(mean, std * 2, params))
return grid_search.best_estimator_
```

*Hình 3. 16: Phần code cho quá trình huấn luyện mô hình SVM với kiểm chứng chéo (cross validation - cv) và tìm kiếm theo lưới*

### 3.3. Kết quả thu được

#### 3.3.1. Kết quả thử nghiệm

Các tham số được sử dụng để đánh giá hiệu quả của mô hình bao gồm các giá trị trong confusion matrix: TP - true positive (số các bản ghi là ho được phân loại là ho), FN - false negative (số các bản ghi không phải ho được phân loại là ho), FP - false positive (số các bản ghi không phải ho được phân loại là ho) và TN - true negative (số các bản ghi không phải ho được phân loại không phải ho) và các tham số tính toán từ các tham số trên bao gồm: độ chuẩn xác (precision), độ nhạy (recall, hay còn gọi là sensitivity), F<sub>1</sub> score và độ chính xác (accuracy).

Trong bài toán phân loại, precision cho từng lớp được tính bằng tỉ lệ những lần phân loại đúng trên tổng số lần phân loại thuộc lớp đó. Như vậy trong trường hợp phân thành hai lớp ho – không ho của bài toán nhận dạng tiếng ho thì precision cho hai lớp lần lượt là [24]:

$$precision_{ho} = \frac{TP}{TP + FP} \quad (3.1)$$

$$precision_{không\ ho} = \frac{TN}{TN + FN} \quad (3.2)$$

Recall là tham số được tính bằng tỉ lệ số lần phân loại đúng trên tổng số bản ghi thuộc một lớp. Tương tự, recall cho mỗi lớp ho – không ho được tính như sau [24]:

$$recall_{ho} = \frac{TP}{TP + FN}, \text{ giá trị này còn được gọi là sensitivity} \quad (3.3)$$

$$precision_{không\ ho} = \frac{TN}{TN + FP} \quad (3.4)$$

$F_1$  score cho mỗi lớp là tham số được tính dựa trên precision và recall của lớp đó. Công thức tính  $F_1$  score như sau [24]:

$$F_1\ score = \frac{2 \cdot precision \cdot recall}{precision + recall} \quad (3.5)$$

Accuracy là tham số được tính bằng tỉ lệ những lần phân loại đúng trên tổng số bản ghi được phân loại, công thức như sau [24]:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.6)$$

Kết quả thu được khi sử dụng mô hình đã huấn luyện để phân loại dữ liệu kiểm tra trong từng bộ dữ liệu sẽ được trình bày ở các phần a và b.

**a) Kết quả thử nghiệm mô hình huấn luyện bằng dữ liệu tự trích trên bộ dữ liệu test**

Nhân	Ho	Không ho
Thực		
Ho	89	1
Không ho	0	90

Bảng 3. 2: Confusion matrix cho mô hình huấn luyện với dữ liệu tự trích khi nhận diện tiếng ho trên bộ dữ liệu test

Lớp	Precision	Recall	F1-score	Số bản ghi
Ho	1,00	0,99	0,99	90
Không ho	0,99	1,00	0,99	90
Trung bình	0,99	0,99	0,99	
Độ chính xác (Accuracy)			0,99	

Bảng 3. 3: Giá trị các tham số đánh giá hoạt động của mô hình huấn luyện với dữ liệu tự trích khi nhận diện tiếng ho trên bộ dữ liệu test

**b) Kết quả thử nghiệm mô hình huấn luyện bằng dữ liệu tổng hợp trên bộ dữ liệu test**

	Nhân	Ho	Không ho
Thực			
Ho	164	9	
Không ho	20	153	

Bảng 3. 4: Confusion matrix cho mô hình huấn luyện với dữ liệu tổng hợp khi nhận diện tiếng ho trên bộ dữ liệu test

Lớp	Precision	Recall	F1-score	Số bản ghi
Ho	0,89	0,95	0,92	173
Không ho	0,94	0,88	0,91	173
Trung bình	0,92	0,92	0,92	
Độ chính xác (Accuracy)		0,92		

Bảng 3. 5: Giá trị các tham số đánh giá hoạt động của mô hình huấn luyện với dữ liệu tổng hợp khi nhận diện tiếng ho trên bộ dữ liệu test

Kết quả thu được khi sử dụng mô hình đã huấn luyện để phân loại dữ liệu từ FSDKaggle2018 sẽ được trình bày ở các phần c và d. Các tham số được sử dụng để đánh giá hiệu quả của mô hình vẫn bao gồm các tham số như trên.

**c) Kết quả thử nghiệm mô hình huấn luyện bằng dữ liệu tự trích trên dữ liệu từ FSDKaggle2018**

	Nhân	Ho	Không ho
Thực			
Ho	29	1	
Không ho	29	1	

Bảng 3. 6: Confusion matrix cho mô hình huấn luyện với dữ liệu tự trích khi nhận diện tiếng ho trên dữ liệu từ FSDKaggle2018

Lớp	Precision	Recall	F1-score	Số bản ghi
Ho	0,50	0,97	0,66	30
Không ho	0,50	0,03	0,06	30
Trung bình	0,50	0,50	0,36	
Độ chính xác (Accuracy)		0,50		

Bảng 3. 7: Giá trị các tham số đánh giá hoạt động của mô hình huấn luyện với dữ liệu tự trích khi nhận diện tiếng ho trên dữ liệu từ FSDKaggle2018

**d) Kết quả thử nghiệm mô hình huấn luyện bằng dữ liệu tổng hợp trên dữ liệu từ FSDKaggle2018**

	Nhân	Ho	Không ho
Thực			
Ho	27	3	
Không ho	12	18	

Bảng 3. 8: Confusion matrix cho mô hình huấn luyện với dữ liệu tổng hợp khi nhận diện tiếng ho trên dữ liệu từ FSDKaggle2018

Lớp	Precision	Recall	F1-score	Số bản ghi
Ho	0,69	0,90	0,78	30
Không ho	0,86	0,60	0,71	30
Trung bình	0,77	0,75	0,74	
Độ chính xác (Accuracy)		0,75		

Bảng 3. 9: Giá trị các tham số đánh giá hoạt động của mô hình huấn luyện với dữ liệu tổng hợp khi nhận diện tiếng dữ liệu từ FSDKaggle2018

### 3.3.2. Đánh giá kết quả thu được

Từ kết quả thu được, có thể thấy cả hai mô hình huấn luyện bằng dữ liệu tự trích và dữ liệu tổng hợp đều nhận ra tiếng ho rất hiệu quả, với cả dữ liệu kiểm tra từ bộ dữ liệu ban đầu và dữ liệu từ FSDKaggle2018, thể hiện ở giá trị recall cho lớp ho đều từ 0,9 trở lên. Điều này thể hiện đặc trưng trung trích bởi mạng VGGish đã có hiệu quả trong việc nhận diện được đúng tiếng ho khi chúng xuất hiện.

Khi phân loại dữ liệu không phải ho từ dữ liệu kiểm tra cùng bộ dữ liệu với dữ liệu huấn luyện, các mô hình cũng hoạt động tốt. Đặc biệt mô hình huấn luyện bằng dữ liệu tự trích đã phân loại đúng tất cả bản ghi không phải ho, khiến cho recall của lớp này và precision của lớp ho có giá trị 1. Mô hình huấn luyện bằng dữ liệu tổng hợp cũng phân loại đúng tới 153/173 bản ghi không phải ho, giúp cho recall đạt 0,88.

Nhờ việc phân loại chính xác ở cả hai lớp khi phân loại trên dữ liệu kiểm tra cùng bộ dữ liệu với dữ liệu huấn luyện, các tham số của cả hai mô hình đều đạt giá trị tốt. Đối với mô hình huấn luyện bằng dữ liệu tổng hợp, precision của lớp ho và không ho là 0,89 và 0,94, F<sub>1</sub> score lần lượt là 0,92 và 0,91. Đối với mô hình huấn luyện bằng dữ liệu tự trích, precision của lớp ho và không ho là 1 và 0,99, F<sub>1</sub> score của cả hai lớp là 0,99.

Tuy nhiên, kết quả phân loại của mô hình huấn luyện bằng dữ liệu tự trích đã quá chính xác khi chạy với dữ liệu kiểm tra nhưng lại rất kém với dữ liệu không phải ho từ FSDKaggle2018. Mô hình này đã phân loại đúng gần như tuyệt đối tất cả bản ghi

trong bộ dữ liệu kiểm tra, khiến cho các tham số cho cả hai lớp đầu có giá trị gần 1. Nhưng với dữ liệu từ FSDKaggle2018, mô hình đã phân loại sai gần hết các bản ghi không phải ho, khiến các tham số của lớp này đều ở mức thấp và rất thấp, đặc biệt recall chỉ có 0,03. Đồng thời việc phân loại sai các bản ghi không ho làm ảnh hưởng tới precision của lớp ho rất nhiều, chỉ còn 0,50. Có thể lí giải điều này như là hiện tượng quá khớp của mô hình với dữ liệu không phải ho của bộ dữ liệu tự trích. Dữ liệu không phải ho trong bộ dữ liệu này chỉ bao gồm các âm thanh từ môi trường trong các file ghi âm tiếng ho ban đầu nên có thể không đủ đa dạng để tổng quát cho âm thanh không ho nói chung. Các âm thanh này còn chủ yếu là các đoạn ngắn, được đệm thêm các đoạn im lặng để đủ 10s nên không thể đạt được hiệu quả mô phỏng các âm thanh không ho thực tế khi huấn luyện.

Điều này có thể được cải thiện bằng cách bổ sung dữ liệu để tăng độ đa dạng cho dữ liệu không phải ho. Hiệu quả của việc này đã được thể hiện một phần ở kết quả của mô hình dữ liệu tổng hợp khi phân loại dữ liệu không phải ho từ FSDKaggle2018. Mô hình này nhận diện được đúng nhiều hơn các bản ghi thuộc lớp này so với mô hình dữ liệu tự trích. Điều này thể hiện ở số bản ghi không phải ho phân loại đúng tăng từ 1 lên 18 trên số 30 bản ghi và từ đó recall tăng từ 0,03 lên 0,60, thể hiện sự cải thiện đáng kể. Precision của cả hai lớp qua đó cũng tăng, lớp ho và không ho lần lượt tăng từ 0,50 lên 0,69 và 0,50 lên 0,86. F<sub>1</sub> score của lớp ho cũng được cải thiện từ 0,66 lên 0,78 và lớp không ho tăng nhiều từ 0,06 lên 0,71. Accuracy chung cũng tăng từ 0,50 lên 0,75. Tuy nhiên đây vẫn là độ chính xác khá thấp cho một mô hình phân loại.

### 3.4. Chương trình ứng dụng mô hình phân loại thu được

Trong phạm vi đồ án, một chương trình nhỏ để nhận diện tiếng ho từ file âm thanh đã được xây dựng bằng cách sử dụng mô hình SVM thu được sau thử nghiệm, cụ thể là mô hình huấn luyện với dữ liệu tổng hợp từ dữ liệu tự trích và dữ liệu từ AudioSet. Chương trình được viết sử dụng thư viện tkinter cho giao diện và python-sounddevice, mô đun wavfile trong scipy.io để ghi âm..

Chương trình có hai chức năng là nhận diện tiếng ho thông qua file wav hoặc tfrecord và nhận diện tiếng ho thông qua file ghi âm trực tiếp bằng micro.

Chương trình được viết và chạy trên môi trường có thông số như sau:

Vị xử lý	Intel Core i7-4500U 1,8 Ghz 4 luồng
Dung lượng RAM	4GB
Dung lượng bộ nhớ	30 GB
Hệ điều hành	Ubuntu 18.04

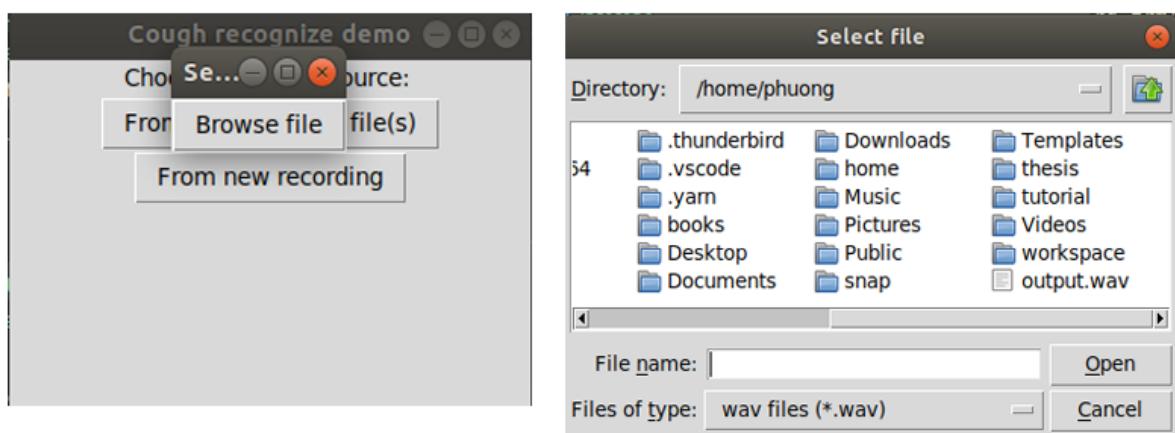
Bảng 3. 10: Môi trường cài đặt chương trình ứng dụng mô hình SVM tổng hợp

Khi chạy chương trình, cửa sổ chính sẽ hiện lên để người dùng chọn nguồn âm thanh muốn phân loại: file sẵn hay ghi âm mới. Khi người dùng chọn file có sẵn sẽ hiện lên cửa sổ để chọn file từ máy tính. Người dùng có thể nháy đúp vào file hoặc

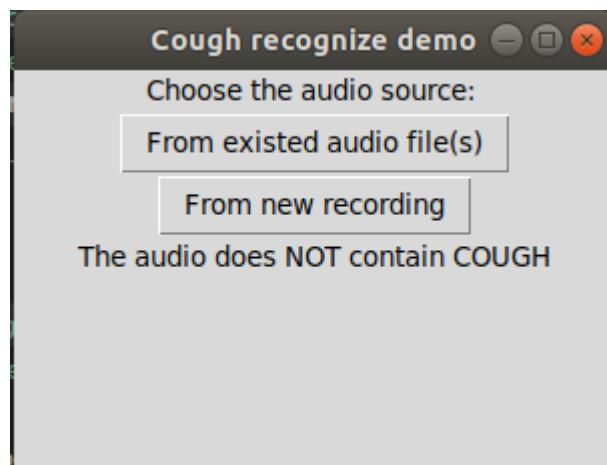
nhấn Open để chọn file. Sau đó file sẽ được xử lý, nếu là file wav, hoặc đọc thông tin đặc trưng từ nó nếu là file tfrecord. Dữ liệu thu được sau đó sẽ chuyển cho SVM phân loại và kết quả sẽ hiện lên màn hình chính



Hình 3. 17: Giao diện chính của chương trình để chọn nguồn âm thanh sẽ phân loại



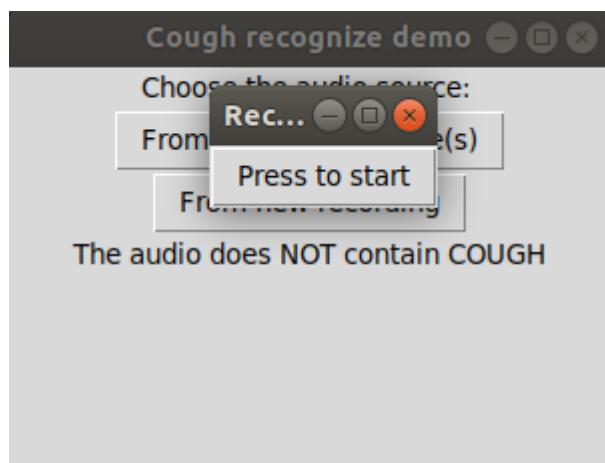
Hình 3. 18: Giao diện chương trình khi chọn file có sẵn



Hình 3. 19: Giao diện hiện kết quả phân loại file

Nếu người dùng chọn ghi âm, một cửa sổ để bắt đầu ghi âm sẽ hiện lên. Chương trình sẽ chỉ thu đúng 10s, sau đó lưu lại âm thanh dưới file wav để xử lý. Sau đó dữ

liệu được phân loại bằng SVM. Kết quả phân loại sau đó cũng hiện trên màn hình chính như Hình 3.19.



Hình 3. 20: Giao diện chương trình khi chọn ghi âm

### 3.5. Kết luận

Chương 3 đã trình bày về kịch bản thử nghiệm cho giải pháp của đồ án, việc cài đặt cũng như môi trường thực hiện cài đặt và kết quả thu được, đánh giá về kết quả đó và chương trình ứng dụng mô hình phân loại thu được.

## KẾT LUẬN

Đồ án đã đề xuất và thử nghiệm giải pháp cho bài toán nhận diện tự động tiếng ho với mạng CNN kết hợp thuật toán SVM và đã có những đóng góp chính như sau:

Thông qua kết quả của thử nghiệm, đặc trưng trích xuất bởi mạng CNN VGGish được huấn luyện sẵn với bộ dữ liệu lớn, đã được chứng minh là có ý nghĩa trong bài toán nhận diện tiếng ho. Đặc trưng thu được đã thể hiện hiệu quả cao trong việc nhận ra tiếng ho khi nó xuất hiện, với độ chính xác lên tới 164 trên 173 bản ghi.

Đồng thời cũng có thể thấy được vai trò quan trọng của bộ dữ liệu huấn luyện, thể hiện qua việc hiệu quả nhận diện tiếng ho được nâng cao khi bổ sung dữ liệu từ AudioSet vào bộ dữ liệu tự trích. Khi bổ sung dữ liệu từ AudioSet, độ đa dạng của dữ liệu được tăng lên giúp cho việc phân loại các âm thanh không phải ho chính xác hơn, từ đó giúp cho các kết luận có xuất hiện tiếng ho cũng chuẩn xác, đáng tin hơn.

Đồ án cũng đã xây dựng được chương trình đơn giản thể hiện được ứng dụng có thể có của một mô hình nhận diện tiếng ho trong thực tế.

Tuy nhiên đồ án cũng không tránh khỏi những thiếu sót như sau:

Mô hình thu được bị quá khớp do dữ liệu chưa được đa dạng. Do đó mới chỉ chứng minh được hiệu quả của đặc trưng cho âm thanh ho.

Giải pháp mới chỉ được thử nghiệm ở quy mô nhỏ, chưa đảm bảo được hiệu quả với lượng dữ liệu lớn và phức tạp hơn. Điều này khiến cho các mô hình thu được chỉ mang tính chất tham khảo, không đảm bảo ứng dụng được trong thực tế.

Từ đó, một số hướng phát triển tương lai từ đồ án có thể được đề xuất như sau:

Sử dụng bộ dữ liệu đa dạng hơn để có thể kiểm chứng được hiệu quả của đặc trưng trong việc nhận dạng các âm thanh khác. Từ đó có thể phát triển thành giải pháp nhận dạng nhiều âm thanh hơn bên cạnh ho. Đồng thời có thể gia tăng số lượng các bản ghi cho bộ dữ liệu để đảm bảo hiệu quả nhận dạng, từ đó phát triển thành giải pháp ứng dụng trong thực tế.

Kết hợp với các giải pháp ghi âm, lọc nhiễu với ứng dụng nhận dạng tiếng ho để hiệu quả phân loại với các bản ghi mới tốt hơn. Qua đó cũng có thể thu thập thêm dữ liệu có thể sử dụng để huấn luyện mô hình phân loại.

## TÀI LIỆU THAM KHẢO

### Tài liệu tiếng Anh:

1. J. Amoh and K. Odame, "Deep Neural Networks for Identifying Cough Sounds," in IEEE Transactions on Biomedical Circuits and Systems, vol. 10, no. 5, pp. 1003-1011, (2016)
2. J. Amoh and K. Odame, "DeepCough: A deep convolutional neural network in a wearable cough detection system", in IEEE Biomedical Circuits and Systems Conference (BioCAS), 1-4, (2015).
3. Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, "TensorFlow: a system for large-scale machine learning". In Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation (OSDI'16). USENIX Association, Berkeley, CA, USA, 265-283, (2016).
4. Samantha J. Barry, Adrie D. Dane, Alyn H. Morice, and Anthony D. Walmsley, "The automatic recognition and counting of cough," in Cough, vol. 2, pp. 8 London, England, (2006)
5. Hsu Chih-wei & Chang Chih-chung & Lin Chih-Jen, "A Practical Guide to Support Vector Classification, (2003).
6. Kunihiko Fukushima, "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift", in Position. Biological Cybernetics, vol. 36, pp. 193—202, (1980).
7. Eduardo Fonseca, Manoj Plakal, Frederic Font, Daniel P. W. Ellis, Xavier Favory, Jordi Pons and Xavier Serra. "General-purpose Tagging of Freesound Audio with AudioSet Labels: Task Description, Dataset, and Baseline". Proceedings of the DCASE 2018 Workshop (2018)
8. Dario Garcia-Gasulla and Ferran Parés and Armand Vilalta and Jonatan Moreno and Eduard Ayguadé and Jesús Labarta and Ulises Cortés and Toyotaro Suzumura, "On the Behavior of Convolutional Nets for Feature Extraction" ArXiv, abs/1703.01127, (2017).
9. Ian J. Goodfellow, Yoshua Bengio and Aaron Courville, "Deep Learning", MIT Press, (2016).
10. Xuedong Huang, Alex Acero and Hsiao-Wuen Hon, Spoken Language Processing: A Guide to Theory, Algorithm, and System Development (1st ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA, (2001).
11. D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex", in The Journal of Physiology. 124, (1959).

12. D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex", in *The Journal of Physiology*. 160, (1962).
13. Eric C. Larson, TienJui Lee, Sean Liu, Margaret Rosenfeld, and Shwetak N. Patel, "Accurate and privacy preserving cough sensing using a low-cost microphone", in *Proceedings of the 13th international conference on Ubiquitous computing (UbiComp '11)*. ACM, New York, NY, USA, 375-384, (2011)
14. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, (Nov. 1998).
15. Lindasalwa Muda, Mumtaj Begam and I. Elamvazuthi, "Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Technique", in *Journal of Computing*, Volume 2, Issue 3, (2010).
16. Michael Negnevitsky, *Artificial Intelligence: A Guide to Intelligent Systems*. Second ed. Addison-Wesley, (2005).
17. Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, (2015).
18. Robert Pardo. *Design, Testing, and Optimization of Trading Systems*. Perry J. Kaufman (Ed.). John Wiley & Sons, Inc., New York, NY, USA, (1992).
19. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay, "Scikit-learn: Machine Learning in Python", *J. Mach. Learn. Res.*, vol. 12, pp. 2825-2830, (November 2011).
20. Lawrence R. Rabiner and Ronald W. Schafer, *Introduction to digital speech processing*, *Found. Trends Signal Process*, (January 2007)
21. Karen Simonyan and Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", in *International Conference on Learning Representations*, San Diego, CA, USA, (May 7-9, 2015).
22. Stanley Smith Stevens, John Volkmann and Edwin B. Newman, "A scale for the measurement of the psychological magnitude pitch", in *Journal of the Acoustical Society of America*. vol. 8, no. 3, pp. 185–190, (1937).
23. J. Smola and Bernhard Schölkopf, "A tutorial on support vector regression", in *Statistics and Computing*, vol. 14, no. 3, pp. 199-222, (August 2004).
24. Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar, *Introduction to Data Mining* (2nd ed.). Pearson, (2018).
25. Paul Taylor, *Text-to-Speech Synthesis*. Cambridge: Cambridge University Press, (2009).

26. X. Xiong, "Robust speech features and acoustic models for speech recognition," PhD. Thesis, 194 p., Nanyang Technological University, Singapore, (2009).

27. Shi Yan, Liu He, Wang Yixuan, Cai Maolin and Xu Weiqing, "Theory and Application of Audio-Based Assessment of Cough", in Journal of Sensors. vol. 2018, pp. 1-10, (2018)

**Danh mục các website tham khảo:**

28. <[https://manual.audacityteam.org/man/dc\\_offset.html](https://manual.audacityteam.org/man/dc_offset.html)>, xem 19/09/2019

29. <<https://github.com/tensorflow/models/tree/master/research/audioset/vggish>>, xem 19/09/2019

30. <<https://www.tensorflow.org/guide/embedding>>, xem 20/09/2019

31. <<https://cloud.google.com/solutions/machine-learning/overview-extracting-and-serving-feature-embeddings-for-machine-learning>>, xem 20/09/2019

32. <<https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/>>, xem 28/09/2019

33. <<http://cs231n.github.io/convolutional-networks>>, xem 28/09/2019

34. <<https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/>>, xem 29/12/2017

35. <<https://research.google.com/audioset/index.html>>, xem 17/10/2019

36. <<https://machinelearningcoban.com/2017/04/09/sm/>>, xem 22/10/2019

37. <<https://machinelearningcoban.com/2017/04/13/softmarginsmv/>>, xem 25/10/2019

38. <<https://machinelearningcoban.com/2017/04/22/kernelsmv/>>, xem 01/11/2019

39. <<https://medium.com/mostly-ai/tensorflow-records-what-they-are-and-how-to-use-them-c46bc4bbb564>>, xem 22/11/2019

40. <[https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html)>, xem 23/11/2019

41. <<https://scikit-learn.org/stable/modules/svm.html>>, xem 25/10/2019

42. <<https://www.sfu.ca/sonic-studio-webdav/handbook/Mel.html>>, xem 19/09/2019

43. <<http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>>, xem 19/09/2019