ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐỒ ÁN TỐT NGHIỆP

XÂY DỰNG BỘ PHÁT HIỆN GIỌNG NÓI BỊ GIẢ MẠO TẠO BỞI DEEPFAKE BẰNG MÔ HÌNH LSTM

Giáo viên hướng dẫn	**:**	TS. Nguyễn Mạnh Cường
Sinh viên	**:**	Bùi Trường Giang
Lớp - Khóa	**:**	2021DHKHMT01 – K16

Hà Nội – Năm 2025

MỤC LỤC

LỜI CẢM ƠN

Trong quá trình nghiên cứu và hoàn thiện báo cáo này, em đã nhận được sự hỗ trợ và đóng góp quý báu từ nhiều cá nhân và tổ chức. Xin dành những dòng đầu tiên này để bày tỏ lòng tri ân sâu sắc đến tất cả những người đã đồng hành, góp ý và tạo điều kiện thuận lợi để em có thể hoàn thành công trình nghiên cứu của mình. Trước hết, em xin gửi lời cảm ơn chân thành đến thầy Nguyễn Mạnh Cường vì đã dành thời gian đọc và đánh giá các bản thảo ban đầu, đồng thời đưa ra những nhận xét, góp ý quan trọng giúp em hoàn thiện nội dung báo cáo. Sự tận tâm và hướng dẫn của thầy là nguồn động viên lớn, giúp em cải thiện chất lượng nghiên cứu và trình bày một cách khoa học hơn. Đồng thời em cũng xin trân trọng cảm ơn những người bạn cùng lớp 2021DHKHMT01, những người đã không ngừng đóng góp ý kiến, phản biện và chia sẻ quan điểm, giúp em hoàn thiện mô hình nghiên cứu và tiếp cận vấn đề một cách toàn diện hơn. Đặc biệt, em xin gửi lời cảm ơn sâu sắc đến những người anh chị tại Samsung, những người đã tận tình định hướng, cung cấp những gợi ý quý báu và chia sẻ kinh nghiệm thực tiễn, giúp em ứng dụng các phương pháp khoa học một cách hiệu quả vào quá trình nghiên cứu. Lời cuối cùng, em xin bày tỏ lòng biết ơn đến gia đình và những người thân yêu vì đã luôn động viên, khích lệ em trong suốt quá trình thực hiện nghiên cứu này. Báo cáo này là thành quả của sự nỗ lực không ngừng cùng với sự hỗ trợ quý báu từ tất cả những người đã đồng hành cùng em. Một lần nữa, em xin trân trọng cảm ơn!

Sinh Viên Thực Hiện

Bùi Trường Giang

LỜI NÓI ĐẦU

Trong những năm gần đây, trí tuệ nhân tạo đã có những bước phát triển vượt bậc, góp phần thay đổi sâu rộng nhiều lĩnh vực từ y tế, tài chính đến truyền thông và giải trí. Đặc biệt, sự phát triển mạnh mẽ của các mô hình học sâu đã thúc đẩy sự phát triển trong các lĩnh vực như xử lý ngôn ngữ tự nhiên, thị giác máy tính cũng như nhận dạng và tổng hợp giọng nói. Tuy nhiên, bên cạnh những lợi ích to lớn mà trí tuệ nhân tạo mang lại, cũng xuất hiện nhiều mặt tiêu cực liên quan đến các ứng dụng công nghệ này, trong đó Deepfake là một ví dụ điển hình. Deepfake là công nghệ cho phép tổng hợp các đoạn âm thanh, hình ảnh và video giả mạo một người thực nào đó ở mức độ tự nhiên cao. Ở mặt tích cực, công nghệ này hỗ trợ ngành công nghiệp giải trí, giúp các nhà làm phim nhanh chóng thay thế khuôn mặt của các diễn viên đóng thế thành diễn viên chính hoặc tái hiện giọng nói của các diễn viên đã khuất. Tuy nhiên, nếu để lọt vào tay người không có mục đích tốt, công nghệ này lại tạo ra nhiều nguy cơ đáng lo ngại, có thể được sử dụng để lan truyền tin tức sai lệch, được khai thác để giả mạo danh tính hoặc thực hiện các hành vi lừa đảo, gây ảnh hưởng nghiêm trọng đến cá nhân và cộng đồng. Trong thời gian gần đây, âm thanh giả mạo được tạo bởi trí tuệ nhân tạo ngày càng trở nên tinh vi, khiến con người khó lòng phân biệt được đâu là giọng nói hình ảnh thật, đâu là giọng nói hình ảnh do máy tạo ra. Điều này đặt ra một thách thức lớn trong việc phát hiện và kiểm soát thông tin giả mạo.

Nhận thấy tầm quan trọng của bài toán này, em quyết định chọn đề tài **"Xây Dựng Bộ Phát Hiện Giọng Nói Bị Giả Mạo Tạo Bởi Deepfake Bằng Mô Hình LSTM"** nhằm giải quyết vấn đề được nêu trên. Cụ thể, em sử dụng mô hình học sâu với mạng LSTM kết hợp với bộ đặc trưng tần số để phân loại giữa âm thanh giả mạo và âm thanh thật. Đây là một hướng tiếp cận tiềm năng nhằm tăng cường khả năng phát hiện các nội dung giả mạo, đặc biệt là trong bối cảnh công nghệ này ngày càng trở nên phổ biến và khó kiểm soát hơn.

Báo cáo này tóm tắt toàn bộ nội dung em đã thực hiện, báo cáo gồm 5 mục lớn với nội dung cụ thể như sau:

Chương 1: Tổng quan về xử lý tín hiệu giọng nói và phát biểu bài toán

Trình bày kiến thức cơ bản về xử lý tín hiệu giọng nói và đặt vấn đề nghiên cứu.

Chương 2: Một số phương pháp nhận diện âm thanh giả mạo

Giới thiệu các phương pháp phổ biến trong việc phát hiện âm thanh giả mạo, bao gồm các kỹ thuật trích xuất đặc trưng và mô hình học máy, học sâu.

Chương 3: Thực nghiệm và đánh giá kết quả

Mô tả quá trình huấn luyện mô hình, bộ dữ liệu sử dụng và đánh giá hiệu suất của mô hình trên tập kiểm thử.

Chương 4: Phân tích và thiết kế chương trình demo

Trình bày cách xây dựng một chương trình thử nghiệm có khả năng phát hiện âm thanh giả mạo trong video.

Kết luận và hướng phát triển

Tổng kết các kết quả đạt được, đánh giá những hạn chế và đề xuất hướng nghiên cứu trong tương lai.

Đề tài được thực hiện với mong muốn đóng góp vào lĩnh vực nhận diện nội dung giả mạo, nhằm giảm thiểu các nguy cơ lừa đảo trên không gian mạng, rất mong nhận được các ý kiến đóng góp và đề xuất cải tiến từ thầy cô và các bạn.

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

CHƯƠNG 1 TỔNG QUAN VỀ XỬ LÝ TÍN HIỆU GIỌNG NÓI VÀ PHÁT BIỂU BÀI TOÁN

Chương thứ nhất sẽ trình bày các kiến thức cơ sở, nền tảng được dùng để thực hiện đề tài bao gồm các đặc trưng vật lý của âm thanh, nguyên lý cảm thụ âm của tai người, nguyên lý thu âm của loa, nguyên lý hình thành tiếng nói, cách biểu diễn âm thanh trong máy tính, các phương pháp trích xuất đặc trưng từ âm thanh. Đồng thời chương cũng trình bày chi tiết bài toán thực hiện trong đề tài và các ràng buộc liên quan.

1.1 Nguyên lý hình thành và nguyên lý cảm thụ của tiếng nói

Nguyên lý hình thành và nguyên lý tiếp nhận của tiếng nói đóng vai trò quan trọng trong việc hiểu rõ bản chất của âm thanh giọng nói con người, từ đó làm cơ sở cho các phương pháp xử lý và nhận diện giọng nói. Trong nghiên cứu về nhận diện âm thanh giả mạo, việc nắm vững nguyên lý hình thành và nguyên lý tiếp nhận của tiếng nói là rất cần thiết. Quá trình hình thành tiếng nói liên quan đến hoạt động của các cơ quan phát âm như thanh quản, khoang miệng và mũi, trong khi quá trình tiếp nhận lại phụ thuộc vào cách tai và não bộ xử lý tín hiệu âm thanh. Sự khác biệt trong quá trình này giữa giọng nói tự nhiên và giọng nói được tổng hợp bằng máy có thể là cơ sở để phát hiện âm thanh giả mạo. Phần này sẽ trình bày các nguyên lý cơ bản về sự hình thành và tiếp nhận tiếng nói, làm nền tảng cho các phương pháp phân tích và xử lý âm thanh trong đề tài.

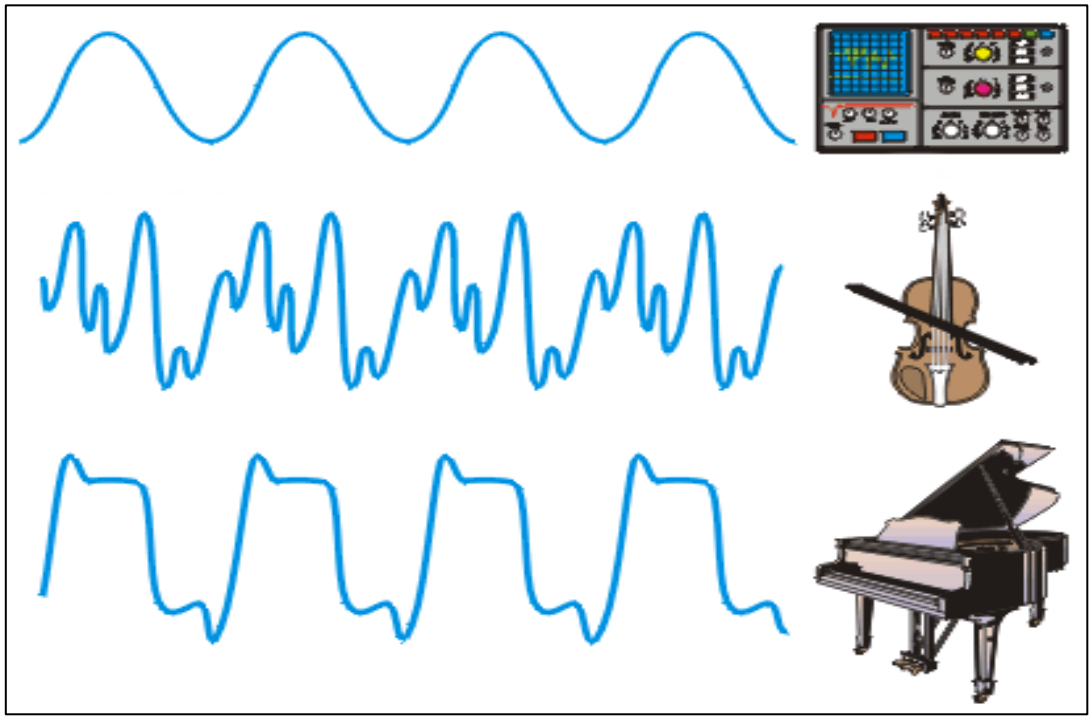
1.1.1 Âm thanh và các đặc trưng của âm thanh

Theo nghĩa hẹp, âm thanh là những sóng được lan truyền trong môi trường rắn, lỏng, khí khi đến tai người sẽ làm cho màng nhĩ dao động, gây ra cảm giác âm. Sóng giao động tạo ra âm thanh được gọi là sóng âm. Về sau khái niệm sóng âm đã được mở rộng cho tất cả những sóng cơ, bất kể chúng có thể gây ra được cảm giác âm hay không. Như vậy sóng âm là những sóng cơ truyền trong môi

trường khí lỏng rắn, tần số của sóng âm cũng là tần số của âm. Âm được sinh ra do sự giao động của một vật, gọi là nguồn âm, tần số của âm phát ra cũng chính bằng với tần số giao động của nguồn âm. Những âm có tác dụng làm cho màng nhĩ trong tai ta dao động, gây ra cảm giác âm được gọi là âm nghe được. Người ta dùng thuật ngữ âm thanh dùng để chỉ âm mà ta nghe được. Âm thanh nghe được có tần số nằm trong khoảng từ 16 Hz đến 20 000 Hz. Âm có tần số nhỏ hơn 16 Hz thì tai người không nghe được thì gọi là hạ âm. Tuy nhiên một số loài vật như voi, chim bồ câu… lại nghe được hạ âm. Âm có tần số lớn hơn 20 000Hz thì tai người cũng không nghe được và gọi là siêu âm, một số loài như dơi, chó, cá heo… có thể nghe được siêu âm. Khi âm truyền qua không khí, mỗi phân tử không khí dao động quanh vị trí cân bằng theo phương trùng với phương truyền sóng, làm cho áp suất không khí tại mỗi điểm cũng dao động quanh giá trị trung bình nào đó.

Những âm có một tần số nhất định thường do các nhạc cụ phát ra, gọi là các nhạc âm. Những âm như tiếng búa đập, tiếng sấm, tiếng ồn ở đường phố, ở chợ… không có một tần số nhất định thì gọi là tạp âm. Báo cáo chỉ tập trung trình bày các đặc trưng vật lí tiêu biểu của nhạc âm. Đặc trưng vật lý đầu tiên là tần số, đây là đặc trưng vật lí quan trọng nhất của âm cho biết tần số giao động của sóng âm tạo ra âm. Tiếp đến là cường độ âm, sóng âm lan đến đâu thì sẽ làm cho các phần tử của môi trường ở đó dao động. Như vậy sóng âm mang theo năng lượng, ta gọi cường độ âm I tại một điểm là đại lượng đo bằng lượng năng lượng mà sóng âm đã truyền tải qua một đơn vị diện tích đặt tại điểm đó, vuông góc với phương truyền của sóng trong một đơn vị thời gian. Đơn vị của cường độ âm là W/m^2. Khi một nhạc cụ phát ra một âm có tần số f_0 thì bao giờ nhạc cụ đó cũng phát ra một loạt các tần số là bội nguyên của f_0 tức $2f_0$, $3f_0$, $4f_0$… có cường độ khác nhau. Âm có tần số f_0 gọi là âm cơ bản hay họa âm thứ nhất, các âm $2f_0$, $3f_0$, $4f_0$… được gọi là họa âm thứ hai, thứ ba, thứ tư… Biên độ của các họa âm lớn, nhỏ không như nhau tùy thuộc và chính nguồn âm. Tập hợp các họa âm tạo thành

phổ của nhạc âm nói trên. Phổ của cùng một âm do các nguồn âm khác nhau phát ra thì hoàn toàn khác nhau. Tổng hợp đồ thị giao động của tất cả các họa âm trong một nhạc âm ta được đồ thị của giao động của nhạc âm đó.



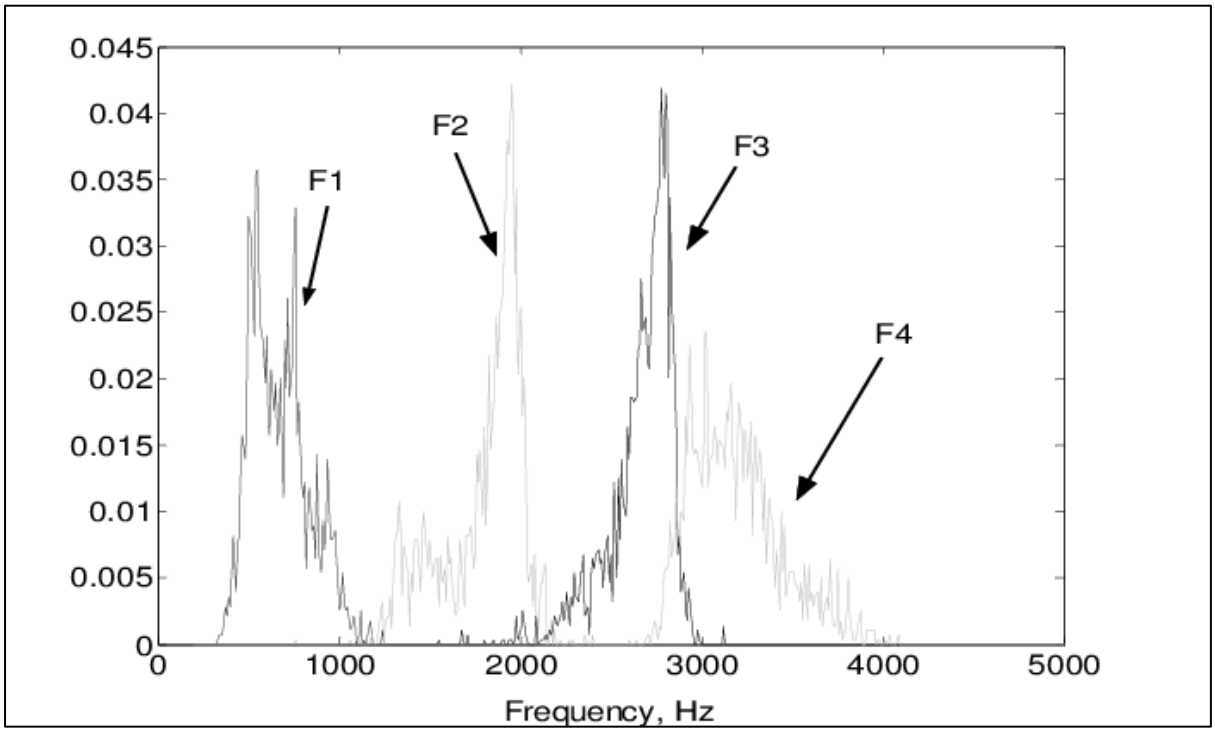
Hình 1.1 Đồ thị giao động của các loại nhạc cụ

Cảm giác mà âm thanh gây ra cho tai người không chỉ phụ thuộc vào các đặc trưng vật lí của âm thanh mà còn phụ thuộc vào sinh lý của tai. Về đặc trưng sinh lí của âm có ba đặc trưng sinh lý bao gồm: Độ cao, độ to, âm sắc. Trong đó độ cao gắn liền với đặc trưng tần số, độ to gắn liền với cường độ âm và âm sắc gắn liền với họa âm. Nhưng điều đó cũng không có nghĩa tần số càng cao thì âm càng cao, cường độ âm càng lớn thì âm càng to.

1.1.2 Nguyên lý hình thành tiếng nói

Đề một tiếng nói có thể phát ra từ cổ họng của một người, đầu tiên một luồng hơi sẽ được đẩy lên từ phổi tạo áp lực lên thanh quản. Dưới áp lực đó, thanh quản mở ra giúp luồng không khí thoát qua, áp lực giảm xuống khiến thanh quản tự động đóng lại. Việc đóng lại như vậy khiến áp lực tăng lên và quá trình tái diễn, các chu kỳ đóng mở thanh quản này lên tục tái diễn, khiến cho thanh quan giao

động, số lần chu kỳ đóng mở thanh quản được thực hiện trên một giây ta gọi là tần số giao động của thanh quản, tần số này cơ bản khoảng 125Hz với nam và 210Hz với nữ. Tần số này được gọi là họa âm bậc một của thanh quản. Như vậy thanh quản đã tạo ra các tần số sóng âm cơ bản. Tuy nhiên để hình thành lên tiếng nói còn cần đến các cơ quan khác như vòm họng, khoang miệng, lưỡi, răng, môi, mũi… Các cơ quan này hoạt động như một bộ cộng hưởng giống như hộp đàn guitar, nhưng khác ở chỗ là có thể thay đổi hình dạng linh hoạt. Bộ cộng hưởng này có tác dụng khuếch đại ột vài tần số và triệt tiêu một số tần số khác, do có sự linh hoạt trong hình dạng nên tần số nó khuếch đại và triệt tiêu cũng có thể được điều chỉnh, từ đó giúp con người có thể tạo ra các âm thanh khác nhau giúp hình thành lên tiếng nói.



Hình 1.2 Các đỉnh cộng hưởng

Hình ảnh trên mô tả chi tiết về cơ chế này, trong đồ thị âm trên ngoài đỉnh cộng hưởng bấc nhất là F_0 ta còn thấy các đỉnh F_1, F_2, F_3… đây được gọi là định sắc(Các dải tần số mà năng lượng của âm thanh được tăng cường do sự cộng hưởng của khoang miệng, họng, mũi). Giá trị theo thời gian của các đỉnh này đặc

trưng cho các âm vị. Trong các phương pháp nhận dạng tiếng nói truyền thống, người ta sẽ cố gắng tách thông tin về các định sắc này ra khỏi F_0 rồi mới sử dụng thông tin này để nhận dạng. Có hai loại âm thanh được tạo ra ở quá trình trên là âm vô thanh và âm hữu thanh. Âm hữu thanh là âm mà khi ta phát âm và đặt tay lên cổ họng sẽ thấy cổ họng rung còn âm vô thanh thì không.

1.1.3 Âm, âm tiết, âm vị

Âm tiết là đơn vị phát âm nhỏ nhất trong lời nói thường được coi là một nhịp nhỏ nhất của tiếng nói. Trong tiếng Anh và nhiều ngôn ngữ khác, một từ được ghép bởi nhiều âm tiết, ví dụ như từ "want" có một âm tiết, "wana" có hai âm tiết, "computer" có ba âm tiết. Trong khi đó trong tiếng Việt, gần như mọi âm tiết đều mang ngữ nghĩa nên ta có thể coi một âm tiết là một từ. Một âm tiết thường là một nguyên âm, có hoặc không có các phụ âm đi kèm.

Nguyên âm là những âm thanh phát ra mà không có sự cản trở trong đường hô hấp, không bị ngắt quãng. Ví dụ như trong tiếng Việt, ta có các nguyên âm a, o, e, i, u, chúng được gọi là nguyên âm vì ta có thể đọc aaaa liên tục mà không không có sự ngắt hơi, chặn họng. Khác với nguyên âm là phụ âm, phụ âm được tạo ra bởi sự đóng hoàn toàn hoặc một phần của thanh quản, làm phá vỡ, ngắt quãng dòng quyên âm, tạo ra các khoảng ngắt rõ ràng.

Âm vị là một đơn vị âm thanh nhỏ nhất trong một ngôn ngữ, có thể phân biệt ý nghĩa với các từ(tức là nếu thay đổi nó từ sẽ mang nghĩa khác hoặc không có nghĩa). Âm là sự hiện thực hóa các âm vị, cùng một âm vị nhưng mỗi người lại có một giọng đọc khác nhau, ví dụ với cùng một từ ba nhưng giọng con trai khác giọng con gái, giọng người Bắc khác giọng người Nam.

1.1.4 Cơ chế hoạt động của tai

Âm thanh tiếng nói mà chúng ta vẫn được nghe hàng ngày là sự pha trộn của rất nhiều sóng âm với tần số khác nhau, như đã đề cập ở mục trước đó các âm này được tạo bởi sóng có tần số nằm trong dải sóng nghe được tức từ 16Hz

đến 20 000 Hz. Tai người và các loài động vật khác cảm nhận âm thanh ở các tần số khác nhau với mức độ nhạy cảm khác nhau. Đối với con người, chúng ta rất nhạy cảm với các âm thanh tần số thấp và lại kém nhạy cảm với các âm thanh tần số cao(chỉ trong dải âm thanh nghe được)

Khi âm thanh truyền tới tai, va đập vào màng nhĩ, màng nhĩ rung lên, truyền rụng động lên ba xương nhỏ malleus, incus, stapes tới ốc tai hay tiếng Việt gọi lần lượt là xương búa, xương đe, xương bàn đạp. Ốc tai là một bộ phận dạng xoắn, rỗng như một con ốc. Ốc tai chứa các dịch nhầy bên trong giúp truyền âm thanh, dọc theo ống tai llaf các tế bào lông cảm nhận âm thanh. Các tế bào này rung lên khi có sóng truyền qua và gửi tế bào tới não bộ. Các tế bào ở phía ngoài cứng hơn nên có tần số rung động cao trong khi đó các tế bào ở sâu bên trong mềm hơn có tần số rung động thấp. Do các tế bào mềm hơn chiếm đa số ở con người nên chúng ta nhạy cảm hơn với các tần số thấp và kém nhạy cảm hơn với tần số cao.

1.1.5 Cơ chế thu thanh của các thiết bị ghi âm

Giống như tai người, cơ chế thu thanh của các thiết bị ghi âm cũng sử dụng một cảm biết có khả năng giao động để đo giao động xung quanh(thường là màng loa) nhưng khác ở chỗ tín hiệu giao động đó không được truyền đến não bộ qua dây thần kinh mà được phiên chuyển thành tín hiệu điện tương tự và được khuếch đại. Đối với các phương pháp ghi âm thanh theo phương pháp tín hiệu tương tự cổ điển ví dụ như đĩa than, sau khi nhận được tín hiệu sau khi đã khuếch đại, một kim khắc bằng kim cương hoặc đá sapphire sẽ rung theo tín hiệu và khắc vào bề mặt đĩa than tạo thành các rãnh xoắc ốc từ mép ngoài vào trung tâm, các giao động khác nhau sẽ tạo ra sự biến đổi khác nhau về độ sâu vào độ rộng trên đĩa. Khi phát lại sẽ có một kim đọc chạm vào rãnh, khi đĩa than quay, những rung động này được truyền đến bộ chuyển đổi trong đầu kim rồi biến đổi lại thành tín hiệu điện, tín hiệu được khuếch đại và đưa ra loa hình thành lên âm thanh ban đầu. Phương thức ghi cơ học này cho phép ghi lại toàn bộ âm thanh liên tục, sát với âm thanh gốc nhưng đổi lại tín hiệu thu được lại ở dạng biểu diễn nguyên sơ

nhất, không phù hợp và thuận tiện cho các công việc như truyền tin, xử lý nhiễu. Ngoài phương pháp lưu trữ theo hình thức tín hiệu tương tự, ngày nay ta còn có phương pháp biểu diễn dưới dạng kỹ thuật số. Để thu thập thông tin về âm thanh và lưu trữ ở dạng kỹ thuật số, một cảm biến đo áp suất không khí sẽ ghi lại các trạng thái môi trường sau mỗi một khoảng thời gian nhất định, số mẫu mà cảm biến ghi được trong một giây được gọi là tần số lấy mẫu, đối với các định dạng âm thanh chất lượng cao tần số lấy mẫu thường giao động từ 96 kHz đến 192kHz. Bất kỳ hàm theo thời gian đều có thể biểu diễn bằng tổng vô hạn các hàm điều hòa với các tần số khác nhau, tức bất kỳ âm thanh nào cũng có thể là sự kết hợp của vô hạn các nhạc âm. Theo định lý Nyquist-Shannon, một tín hiệu liên tục có thể được tái tạo hoàn hảo từ các mẫu rời rạc nếu nó được lấy mẫu với tần số ít nhất gấp đôi tần số cao nhất có trong tín hiệu đó. Vậy nên để thu thanh tốt cho tiếng nói của con người, tần số lấy mẫu nên ở 8kHz cho chất lượng thấp nhất và 48kHz cho chất lượng cao nhất. Sau khi các mẫu được ghi lại, chúng sẽ được phiên dịch một các có cấu trúc thành các giá trị nhị phân biểu diễn tương ứng chỉ toàn 0 và 1 rồi được ghi lại vào trong các thiết bị nhớ thứ cấp như đĩa từ hay ổ cứng thể rắn. Cách biểu diễn này làm mất đi tính liên tục của thông tin âm thanh ban đầu nhưng đánh đổi lại ta có thể áp dụng các cơ chế xử lý nhiễu, truyền tin tin cậy cho thông tin trên. Cụ thể khi cần truyền tin ta chỉ cần sử dụng một hàm sinh ra một thông tin kiểm tra dựa trên thông tin cần truyền đi, hàm này cần đảm bảo rằng khi có một thay đổi nhỏ trong thông tin đầu vào thì cũng sẽ tạo ra một thay đổi lớn trong thông tin đầu ra. Khi truyền tin đi, ta sẽ truyền cả thông tin gốc và thông tin kiểm tra, bên nhận sẽ thực hiện tính toán hàm trên với đầu vào là giá trị nó nhận được và đối sánh với giá trị kiểm tra, nếu giá trị kiểm tra khớp không tin nhận được được coi là toàn vẹn, nếu giá trị kiểm tra không khớp, thông tin có thể đã bị hư hại trong đường truyền và bên nhận có thể lựa chọn yêu cầu bên truyền gửi lại thông tin hoặc đưa ra các phương pháp khôi phục phù hợp với đoạn tin lỗi.

1.2 Xử lý âm thanh

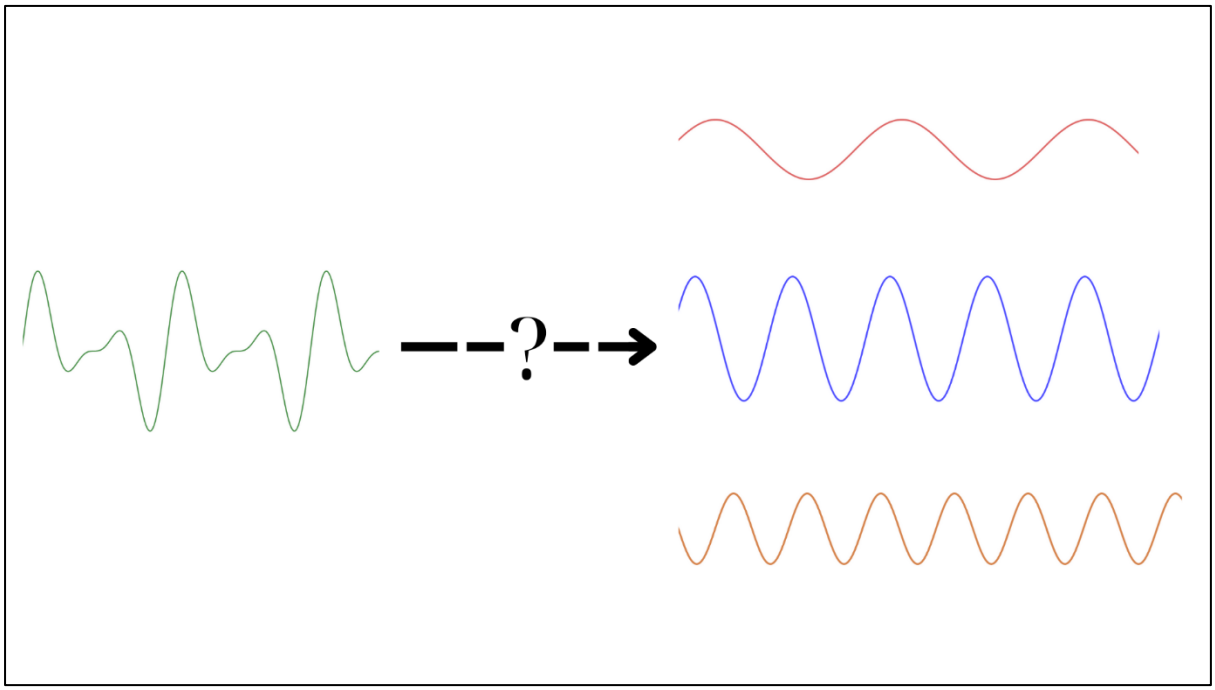
1.2.1 Biểu diễn âm thanh kỹ thuật số trong máy tính

Âm thanh trong máy tính được biểu diễn thông qua nhiều định dạng khác nhau, mỗi định dạng có những đặc điểm riêng nhằm phục vụ các mục đích sử dụng khác nhau. Tùy vào nhu cầu lưu trữ, truyền tải hay xử lý, các định dạng này có thể được phân thành hai nhóm chính: âm thanh không nén và âm thanh nén. Những định dạng âm thanh không nén lưu trữ dữ liệu âm thanh dưới dạng số mà không có bất kỳ sự can thiệp nào nhằm giảm dung lượng tệp. Hai định dạng phổ biến nhất thuộc nhóm này là WAV và AIFF. Đầu tiên là WAV, được phát triển bởi Microsoft và IBM, sử dụng định dạng PCM để lưu trữ âm thanh số hóa mà không làm suy giảm chất lượng. Do đó, WAV thường được sử dụng trong lĩnh vực thu âm chuyên nghiệp, chỉnh sửa âm thanh và lưu trữ dữ liệu âm thanh chất lượng cao. Tiếp đến là AIFF, do Apple phát triển, có chức năng tương tự WAV nhưng phổ biến hơn trên hệ sinh thái macOS. Điểm khác biệt giữa AIFF và WAV chủ yếu nằm ở cách lưu trữ dữ liệu: AIFF sử dụng định dạng big-endian, trong khi WAV sử dụng little-endian, dẫn đến sự khác biệt trong cách đọc dữ liệu trên từng hệ thống. Cả hai định dạng này đều có chất lượng âm thanh cao nhưng chiếm nhiều dung lượng, với mỗi phút âm thanh có thể tốn hơn 10 MB nếu sử dụng chuẩn CD.

1.2.2 Phép biến đổi Fourier

Âm thanh mà ta nghe được hàng ngày là sự kết hợp của nhiều các nhạc âm (âm điều hòa phát ra với tần số cố định). Việc tách các hợp âm phức thành các nhạc âm cơ bản cho phép ta có thể đưa ra các cách xử lý khác nhau với các phổ tần số khác nhau. Ví dụ ta có thể lọc tiếng ù ù trong âm thanh bằng cách tách âm gốc thành thành các nhạc âm cơ bản, sau đó loại bỏ âm có tần số 50Hz(tần số của tiếng ù ù). Nhưng thế nào để ta có thể tách một hàm cho ra giá trị đầu ra theo đầu vào là thời gian thành tổng của các hàm điều hòa? Điều này không hề đơn giản,

nó cũng giống như việc ta tách các thành phần nguyên liệu từ một cốc nước ép đã được trộn đều vậy.



Hình 1.3 Chuyển sóng phức thành tổng các sóng thuần

May thay nhờ vào phép biến đổi Fourier [1] [2], công việc trên trở lên đơn giản hơn bao giờ hết. Vào đầu thế kỷ 19, nhà toán học người Pháp Joseph Fourier đã đặt nền móng cho một trong những công cụ toán học quan trọng nhất trong phân tích tín hiệu và hệ thống – phép biến đổi Fourier. Ban đầu, Fourier nghiên cứu về sự lan truyền nhiệt và nhận thấy rằng bất kỳ hàm tuần hoàn nào cũng có thể được biểu diễn dưới dạng tổng của các hàm sin và cosin có tần số khác nhau. Phát hiện này đã dẫn đến sự ra đời của chuỗi Fourier, một công cụ mạnh mẽ trong việc phân tích tín hiệu và sóng. Mặc dù lúc bấy giờ, công trình của Fourier vấp phải sự hoài nghi từ nhiều nhà toán học đương thời, nhưng sau đó, lý thuyết này đã được chấp nhận rộng rãi và phát triển thành phép biến đổi Fourier tổng quát, áp dụng không chỉ cho các hàm tuần hoàn mà còn cho các hàm bất kỳ. Qua hơn hai thế kỷ, phương pháp của Fourier đã trở thành nền tảng trong nhiều lĩnh vực, từ xử lý tín hiệu, phân tích dữ liệu, đến vật lý lượng tử, giúp con người hiểu rõ hơn về các hiện tượng dao động và biến đổi trong tự nhiên.

Công thức cho phép biến đổi Fourier liên tục được biểu diễn như sau:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\, dt$$

Trong đó

$F(\omega)$ là hàm biểu diễn tín hiệu cho biết sóng thành thành phần có hệ số góc là ôm mê ga chiếm biên độ bao nhiêu

$f(t)$ là hàm tín hiệu gốc ban đầu

$e^{-i\omega t}$ là hàm mũ phức đóng vai trò như bộ lọc giúp trích xuất các tần số riêng lẻ của tín hiệu

Phép biến đổi Fourier liên tục có nhiều ý nghĩa toán học quan trọng, tuy nhiên trên thực tế âm thanh trong máy tính được biểu diễn dưới dạng rời rạc là các giá trị áp suất không khí theo thời gian. Bởi thế ta cần một công cụ khác phù hợp hơn, đó là phép biến đổi Fourier rời rạc một biến thể của phép biến đổi gốc. Đối với một tín hiệu rời rạc $x[n]$ có N mẫu, phép biến đổi Fourier rời rạc được định nghĩa như sau:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\left(\frac{2\pi}{N}\right) k\, n} \quad với\ k\ =\ 0, 1, \dots, N-1$$

Trong đó:

$X[k]$: Là giá trị phổ tại tần số k, biểu diễn mức độ đóng góp thông tin của thành phần tần số trong tín hiệu gốc

$x[n]$: là tín hiệu rời rạc trong miền thời gian, trong bài toán đề thì là áp suất âm thanh mà micro thu được tại mỗi thời điểm

N: Là số mẫu của tín hiệu

j: là đơn vị phức với $j^2 = -1$

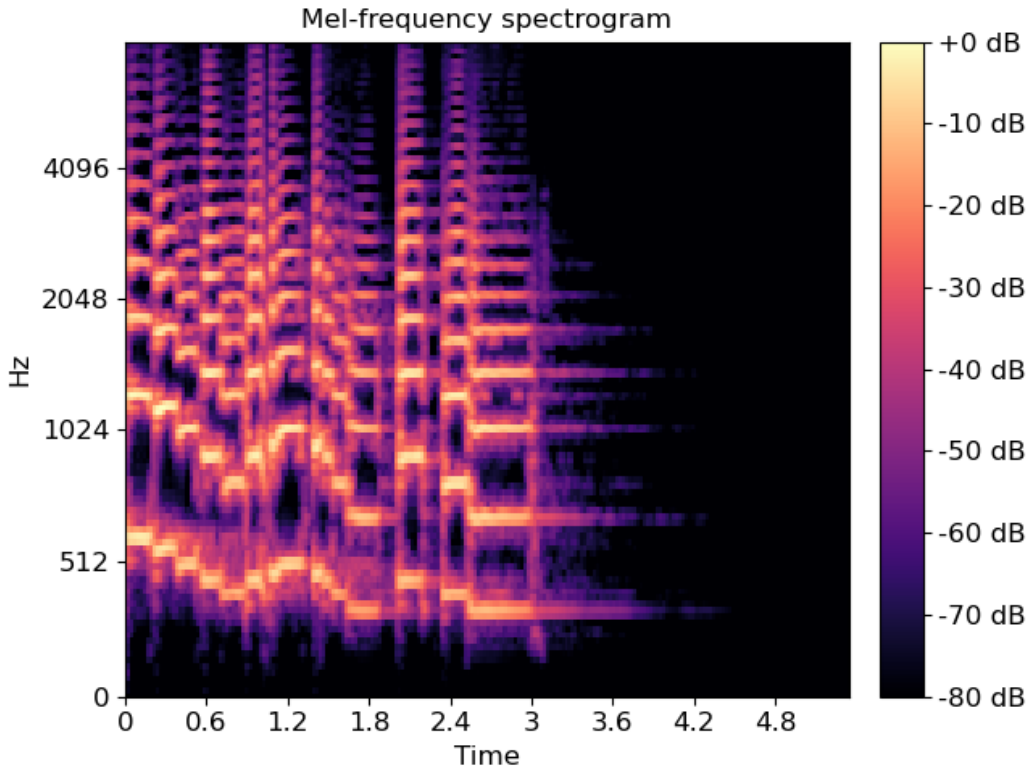
Ngược lại, tín hiệu gốc có thể được khôi phục bằng phép biến đổi Fourier rời rạc nghịch đảo

$$x(n) = \left(\frac{1}{N}\right)\sum_{k=0}^{N-1} X(k)e^{j\left(\frac{2\pi}{N}\right)k\,n} \quad với\ n = 0, 1, \dots, N-1$$

Phép biến đổi Fourier đóng vai trò quan trọng trong xử lý tín hiệu nói chung và xử lý âm thanh nói riêng. Bằng cách chuyển đổi tín hiệu từ miền thời gian sang miền tần số giúp trích xuất và biểu diễn thông tin tần số của âm thanh một cách rõ ràng. Đây là công cụ nền tảng để tiếp cận các phương pháp trích xuất đặc trưng được trình bày trong các mục sau của báo cáo.

1.2.3 Biểu đồ phổ tần số Mel

Khi phân tích tín hiệu âm thanh, việc chỉ quan sát phổ tần số tại một thời điểm là chưa đủ để hiểu rõ sự thay đổi của tín hiệu theo thời gian. Mel Spectrogram [3] là một phương pháp biểu diễn âm thanh trực quan giúp giải quyết vấn đề nêu trên bằng việc kết hợp thông tin tần số và thời gian. Bằng cách áp dụng phép biến đổi Fourier trên từng đoạn nhỏ của tín hiệu và sử dụng thang tần số Mel, ta thu được một dải số liệu gần như liên tục cho biết độ mạnh tín hiệu theo tần số và thời điểm. Mel Spectrogram cung cấp một biểu đồ hai chiều giúp theo dõi sự thay đổi của các thành phần tần số trong suốt quá trình phát âm, trong đó trục hoành thể hiện thời gian, trục tung thể hiện tần số trên thang Mel, còn cường độ của từng thành phần tần số được mã hóa bằng mức độ màu sắc. Điều này làm cho Mel Spectrogram trở thành công cụ quan trọng trong các bài toán liên quan đến xử lý âm thanh.



Hình 1.4 Lược đồ Mel Spectrogram

Ví dụ nhìn vào biểu đồ này ta có thể thấy

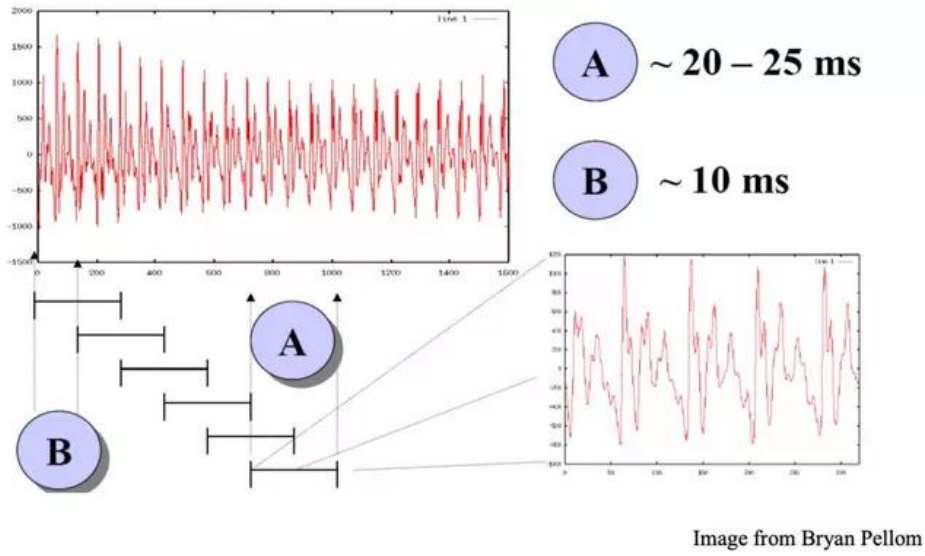
- Trục hoành: Thể hiện thời gian của tín hiệu âm thanh, kéo dài từ 0 đến khoảng 5 giây.
- Trục tung: Hiển thị tần số của tín hiệu theo thang Mel, trải dài từ 0 Hz đến hơn 4000 Hz.
- Thanh màu: Thể hiện mức năng lượng của tín hiệu, từ -80 dB (màu đen, tín hiệu yếu) đến 0 dB (màu vàng sáng, tín hiệu mạnh).
- Dải tần số thấp (0 - 512 Hz) có mức năng lượng khá cao, thể hiện qua các dải màu sáng rõ rệt, điều này thường gặp trong giọng nói con người hoặc âm thanh có âm trầm.
- Các đường sọc ngang xuất hiện theo chu kỳ, cho thấy tín hiệu có đặc tính dao động đều đặn, có thể là một giọng nói hoặc âm thanh nhạc cụ.
- Phần tín hiệu mạnh xuất hiện chủ yếu từ 0 - 2.5 giây, sau đó giảm dần về cuối (sau 3 giây), cho thấy âm thanh đang tắt dần hoặc kết thúc.

1.2.4 Hệ số MFCC

Hệ số cepstrum tần số Mel là một trong những đặc trưng quan trọng nhất trong xử lý giọng nói và âm thanh. MFCC là một phương pháp trích chọn đặc trưng cho âm thanh được sử dụng rộng rãi trong các ứng dụng như nhận diện giọng nói, nhận dạng cảm xúc, phân loại âm thanh và cả trong bài toán phát hiện giả mạo âm thanh giả mảo bởi Deepfake.

Quá trình tính toán MFCC bao gồm các bước sau:

- Khuếch đại các âm có tần số cao: Do đặc điểm cấu tạo thanh quản và các bộ phận phát âm nên tiếng nói của chúng ta có đặc điểm: các âm ở tần số thấp có mức năng lượng cao, các âm ở tần số cao lại có mức năng lượng khá thấp. Trong khi đó, các tần số cao này vẫn chứa nhiều thông tin về âm vị. Vì vậy chúng ta cần 1 bước pre-emphasis để kích các tín hiệu ở tần số cao này lên.
- Phân thành các đoạn: Thay vì biến đổi Fourier trên cả đoạn âm thanh dài, ta trượt 1 cửa sổ dọc theo tín hiệu để lấy ra các đoạn nhỏ rồi mới áp dụng DFT trên từng đoạn này. Tốc độ nói của con người trung bình khoảng 3, 4 từ mỗi giây, mỗi từ khoảng 3-4 âm, mỗi âm chia thành 3-4 phần, như vậy 1 giây âm thanh được chia thành 36 - 40 phần, ta chọn độ rộng mỗi đoạn khoảng 20 - 25ms là vừa đủ rộng để bao 1 phần âm thanh. Các đoạn được lồng lên nhau khoảng 10ms để có thể lưu lại lại sự thay đổi ngữ cảnh.



Hình 1.5 Quá trình phân đoạn âm thanh gốc

- Áp dụng phép biến đổi Fourier rời rạc trên để chuyển đổi tín hiệu từ miền thời gian sang miền tần số, kết quả ta thu được một Mel spectrogram
- Mel filterbank: Như đã mô tả ở phần trước, cách cảm nhận của tai người là phi tuyến tính, không giống các thiết bị đo. Tai người cảm nhận tốt ở các tần số thấp, kém nhạy cảm với các tần số cao. Do đó ta cần 1 cơ chế ánh xạ tương tự như vậy. Trước hết ta sẽ bình phương các giá trị ban đầu để chuyển biểu đồ trên sang phổ công suất, sau đó lọc đi các tần số có công suất không nằm trong khoảng giá trị mà giải tần số đó bao phủ.
- Sau đó ta lấy logarit các giá trị trên để đưa thông tin về dạng chuẩn hóa

1.3 Bài toán nhận diện âm thanh bị làm giả bởi Deepfake

Trong mục này, báo cáo sẽ trình bày cụ thể hơn về bài toán nhận diện âm thanh bị làm giả bởi Deepfake thông qua việc xác định tính cấp thiết của bài toán, lợi ích và tính ứng dụng của lời giải, phạm vi nghiên cứu, những khó khăn thách thức cũng như đầu vào và đầu ra của bài toán.

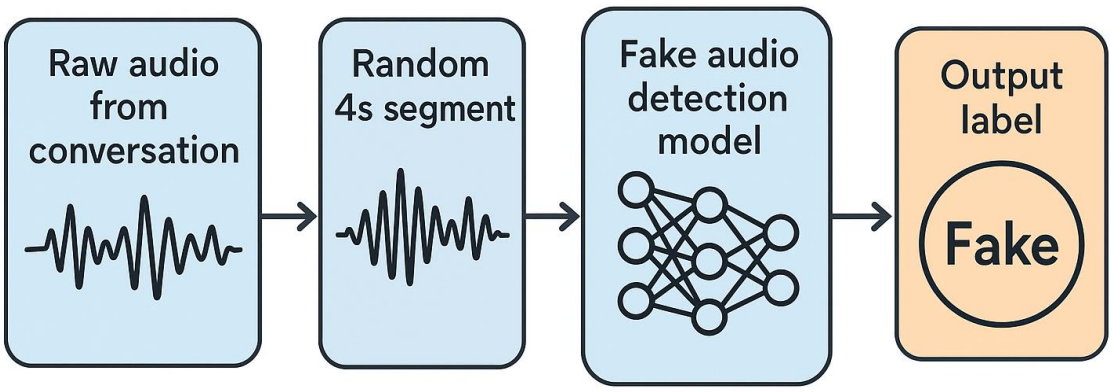
1.3.1 Tính cấp thiết của đề tài

Như đã đề cập phía trên, với sự phát triển của công nghệ trí tuệ nhân tạo, đặc biệt là các mô hình tổng hợp giọng nói và video dựa trên Deepfake, giọng nói và video giả mạo ngày càng trở nên tinh vi và khó phân biệt với giọng nói và hình ảnh thật. Điều này đặt ra nhiều thách thức và nguy cơ, từ gian lận danh tính, lừa đảo trực tuyến, đến việc thao túng thông tin và gây ảnh hưởng tiêu cực đến xã hội. Do đó, việc xây dựng một hệ thống phát hiện giọng nói bị giả mạo là vô cùng cần thiết nhằm bảo vệ an ninh thông tin, nâng cao độ tin cậy của các hệ thống nhận diện giọng nói, và hạn chế các hành vi lạm dụng công nghệ tổng hợp giọng nói vào mục đích xấu.

1.3.2 Đầu vào và đầu ra của bài toán

Bài toán có đầu vào là một đoạn âm thanh giọng nói bất kỳ. Đoạn âm thanh này có thể được thu từ một cuộc trò chuyện, một đoạn ghi âm, hoặc một tập tin âm thanh bất kỳ chứa giọng nói con người, với yêu cầu định dạng chuẩn là .wav. Để đảm bảo chất lượng và tính đồng nhất của dữ liệu, tần số lấy mẫu yêu cầu cho tất cả các đoạn âm thanh là 16 000 mẫu/giây. Trong trường hợp âm thanh đầu vào không đáp ứng được tần số lấy mẫu này, hệ thống sẽ tự động thực hiện các thao tác tiền xử lý bao gồm sao chép hoặc lược bỏ mẫu phù hợp, nhằm khôi phục hoặc tinh chỉnh tín hiệu âm thanh mà không làm sai lệch quá nhiều đặc trưng của giọng nói. Sau đó sẽ lấy ngẫu nhiên 4 giây liên tiếp từ đoạn âm thanh ban đầu tức 64 000 điểm. Sau quá trình tiền xử lý trên, dữ liệu sẽ được đưa qua một mô hình nhận diện âm thanh giả mạo. Mô hình sẽ tính toán và cho ra đầu ra ở dạng xác

suất cho biết âm thanh đầu vào có xác suất bao nhiêu phần trăm giả mạo và bao nhiêu phần trăm là thật.



Hình 1.6 Quá trình phát hiện âm thanh giả mạo

1.3.3 Lợi ích và tính ứng dụng của lời giải

Việc xây dựng thành công một mô hình phát hiện giọng nói giả mạo giải quyết nhiều rủi ro an ninh mạng. Mô hình có thể được tích hợp vào các ứng dụng mạng xã hội giúp người dùng phân biệt được đâu là âm thanh và video được con người tạo ra, đâu là âm thanh và video được làm giả với mục đích xấu. Điều này làm giảm nguy cơ lừa đảo trên không gian mạng.

1.3.4 Mục tiêu nghiên cứu

Nghiên cứu tập trung xây dựng một phương pháp tiếp cận toàn diện cho bài toán nhận diện âm thanh bị làm giả bởi Deepfake. Nghiên cứu sẽ bao gồm việc tìm hiểu các lý thuyết cơ sở về âm thanh và cảm thụ giọng nói, phân tích các phương pháp đã được thực hiện trong quá khứ, xác định các ưu nhược điểm, từ đó tổng hợp lại và xây dựng một phương pháp mới khắc phục các nhược điểm nếu khả thi.

1.3.5 Phạm vi nghiên cứu

Đề tài tập trung vào phát hiện giọng nói bị giả mạo được tổng hợp bởi Deepfake, không xem xét các phương pháp giả mạo khác. Đề tài chỉ sử dụng âm thanh để đánh giá giả mạo, không tập trung vào dữ liệu hình ảnh cùng được tạo

ra đồng thời, không tập trung vào ý nghĩa câu từ lời nói. Bộ dữ liệu được sử dụng trong đề tài là bộ SceneFake [4], đây là một bộ dữ liệu lớn và cân bằng bao gồm hơn 100 000 mẫu được sử dụng rộng rãi cho việc giải quyết bài toán này.

1.3.6 Phương pháp nghiên cứu

Nghiên cứu sẽ được thực hiện dựa trên việc khám phá bộ dữ liệu SceneFake, cố gắng trực quan hóa bộ dữ liệu thông qua một số kỹ thuật, tìm điểm khác biệt giữa giọng nói bình thường và giọng nói giả mạo thông qua biểu đồ trực quan, liệt kê và lượng hóa điểm khác biệt, tìm cách trích chọn đặc trưng mang đủ các thông tin về điểm khác biệt trên, đưa vào một mô hình học sâu huấn luyện và đánh giá. Phát triển một ứng dụng trên hệ điều hành android để minh họa hoạt động của mô hình.

1.3.7 Những thuận lợi trong quá trình giải quyết bài toán

Nhận diện âm thanh giả mạo được tạo bởi Deepfake không phải là một bài toán mới, đã có nhiều nghiên cứu và lời giải trước đây xoay xung quanh vấn đề này. Đề tài này có thể nghiên cứu và tận dụng các ý tưởng trước đó.

1.3.8. Những khó khăn trong quá trình giải quyết bài toán

Âm thanh thu thập được không ở dạng thuần khiết mà luôn tồn tại những nhiễu loạn nhỏ, có thể do môi trường hoặc các thiết bị ghi âm bởi nên quá trình trích chọn đặc trưng cần đảm bảo đưa ra các thông tin đã được tách bỏ các yếu tố không mang ý nghĩa. Thêm nữa các phiên bản đời mới tân tiến nhất của Deepfake cho ra các âm thanh giọng nói có sự khác biệt rất nhỏ so với giọng nói gốc.

1.4 Tóm tắt chương

- Tổng quan về xử lý âm thanh
    - Tiếng nói có bản chất là sóng âm và truyền qua môi trường dựa trên giao động của các phần tử vật chất, tiếng nói có thể được ghi lại bằng việc đo áp xuất không khí theo thời gian

- Giống như mọi thông tin khác, tiếng nói có thể được biểu diễn, lưu trữ và xử lý trên máy tính
- Hai công cụ khoa học cơ sở để làm việc với âm thanh là phép biến đổi Fourier và lược đồ mel spectro, phép biến đổi Fourier cho phép chuyển một sóng phức thành tổng các sóng thuần theo từng miền tần số còn lược đồ mel cho phép trực quan hóa phổ công suất của từng tần số theo thang Mel theo thời gian

- Bài toán nhận diện âm thanh giả mạo được tạo bởi Deepfake
  - Deepfake là công nghệ cho phép tổng hợp âm thanh giọng nói và hình ảnh khuôn mặt giống một người khác dựa trên một số ít mẫu ví dụ về họ
  - Khi bị những cá nhân không có mục đích tốt lợi dụng, Deepfake có thể gây ra những rủi ro an ninh mạng đối với con người hoặc các hệ thống sử dụng giọng nói như là một bước xác thực danh tính
  - Đề tài hướng đến xây dựng một mô hình nhận diện âm thanh giọng nói giả mạo được tổng hợp bởi Deepfake sử dụng học sâu

CHƯƠNG 2 MỘT SỐ PHƯƠNG PHÁP GIẢI QUYẾT BÀI TOÁN

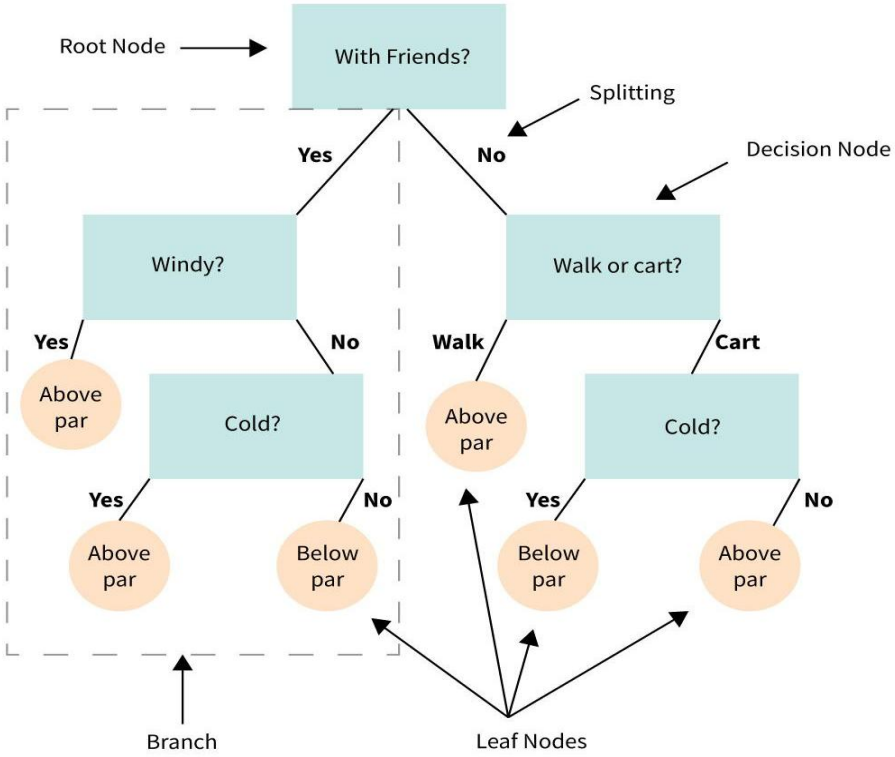
Trong chương này, báo cáo sẽ trình bày các phương pháp đã được được nghiên cứu trong quá khứ đồng thời phân tích các ưu nhược điểm sau đó trình bày phương pháp được sử dụng trong đề tài.

2.1 Hướng tiếp cận học máy

Dễ thấy khi sử dụng phương pháp trích chọn đặc trưng với bộ đặc trưng MFCC, ta nhận được một vector đặc trưng cho mỗi khoảng thời gian trong đoạn âm thanh gốc, lấy trung bình theo phần tử của các vector trên ta được một vector đặc trưng duy nhất. Nếu vector đặc trưng này mang đầy đủ thông tin liên hệ tới đầu ra, ta hoàn toàn có thể sử dụng một mô hình phân lớp [5] để biểu diễn mối liên hệ. Mục này sẽ trình bày một số thuật toán phân lớp có thể được sử dụng để giải quyết vấn đề trên.

2.1.1 Mô hình Random Forest

Random Forest [6] [7] là một thuật toán học máy có giám sát được xây dựng bằng việc xây dựng hàng loạt các cây quyết định [8] với độ sâu cho trước, kết quả dự báo cuối cùng sẽ được quyết định bởi số đông cây. Trong đó cây quyết định được cấu thành từ nhiều nút có liên kết với nhau, mỗi nút trên cây quyết định là một phép toán kiểm tra điều kiện, với mỗi kết quả của phép kiểm tra điều kiện sẽ là một liên kết tới một nút con khác. Các nút lá sẽ là các nút quyết định, đưa ra kết quả dự báo cuối cùng.



Hình 2.1 Ảnh minh họa cây quyết định

Giống như mọi thuật toán học máy khác, cây quyết định cũng có hai pha là huấn luyện và dự báo.

a, Pha huấn luyện

Tại pha huấn luyện của cây quyết định, ta cần tìm cách xây dựng cây quyết định. Cây sẽ được định hình bằng việc ta lựa chọn thuộc tính nào làm thuộc tính quyết định ở mỗi nút. Để công bằng nhất cho các thuộc tính, ta sẽ xem xét từng thuộc tính và đánh giá xem thuộc tính nào mang lại nhiều "thuận lợi" nhất. Từ "thuận lợi" ở đây được hiểu là tính phân chia khi ta phân hoạch tập dữ liệu ban đầu. Cụ thể hơn, nếu tập phân hoạch có phân bố nhãn không đều, một nhãn xuất hiện nhiều hơn hoàn toàn các nhãn còn lại khi đó phân hoạch của ta theo một thuộc tính được đánh giá là "thuận lợi". Ngược lại nếu các nhãn phân bố đều tập phân hoạch của ta được đánh giá là không thuận lợi. Việc chọn được một thuộc tính phân tập dữ liệu ra thành các tập con chỉ có 1 nhãn chiếm đa số cũng đồng nghĩa với việc ta đã tìm ra được một yếu tố quyết định nhãn. Vậy nên khi xây dựng cây, tại mỗi nút ta cần quyết định thuộc tính nào là thuận "lợi nhất".

Về mặt toán học để đánh giá độ thuận lợi cho phân hoạch ta có thể sử dụng tới công thức tính entropy.

$$H(S) = \sum_{i}^{n} - log\ p_i log\ p_i$$

Với:

n là tập các nhãn trong phân hoạch S

p_i là xác suất nhãn i xuất hiện trong phân hoạch s

Với công thức trên ta có thể thấy nếu các nhãn phân bố đều, H(S) sẽ đạt giá trị lớn nhất là log(n), ngược lại nếu chỉ có một nhãn duy nhất trong phân hoạch S, giá trị của H(S) sẽ đạt nhỏ nhất là 0. Từ đó ta có thể kết luận giá trị entropy trong một phân hoạch càng nhỏ thì độ "thuận lợi" càng lớn.

Do một thuộc tính có thể cho ra nhiều phân hoạch, do đó ta cần tìm cách đánh giá độ "thuận lợi" cho thuộc tính đó mà vẫn giữ được tính công bằng cho từng phân hoạch. Công thức đó được xây dựng về mặt toán học như sau.

$$H(x, S) = \sum_{i}^{k} \frac{m_i}{n} H(S_i)$$

Với:

x là thuộc tính đang xét

S là phân hoạch hiện tại

k là tập các giá trị của thuộc tính x

m_i là số phần tử trong phân hoạch hiện tại có thuộc tính x mang giá trị i

n là số phần tử trong phân hoạch hiện tại

S_i là phân hoạch con của phân hoạch hiện tại chứa các phần tử có thuộc tính x mang giá trị i

Với công thức trên ta tổng hợp được mã giả như sau cho quá trình xây dựng quyết định:

```
Algorithm xây_dựng_cây_quyết_định(tập_dữ_liệu)
  If có_phải_chỉ_có_một_nhãn(tập_dữ_liệu) == 0 then
    Return Nút_quyết_định(nhãn_duy_nhất)
  nút_hiện_tại ← NULL
  thuộc_tính_thuận_lợi ← NULL
  min_h ← VÔ_CÙNG

  For each thuộc_tính in tập_thuộc_tính do
    entropy ← h(tập_dữ_liệu, thuộc_tính)
    If entropy < min_h then
      thuộc_tính_thuận_lợi ← thuộc_tính
      min_h ← entropy

  nút_hiện_tại.thuộc_tính_quyết_định ← thuộc_tính_thuận_lợi

  For each giá_trị in các_giá_trị_của(thuộc_tính_thuận_lợi) do
    tập_con ← phân_hoạch(tập_dữ_liệu, thuộc_tính_thuận_lợi, giá_trị)
    cây_con ← xây_dựng_cây_quyết_định(tập_con)
    nút_hiện_tại.bổ_sung_cây_con(giá_trị, cây_con)

  Return nút_hiện_tại
```

b, Pha dự báo

Sau khi quá trình huấn luyện diễn ra, mô hình sẽ được làm việc với dữ liệu thực tế chưa được thấy trong tập huấn luyện. Để dự báo nhãn cho một mẫu, ta sẽ thực hiện duyệt cây từ gốc xuống lá. Tại mỗi nút, ta sẽ quan sát thuộc tính mà nút đó yêu cầu rồi di chuyển đến nút con thỏa mãn điều kiện tương ứng với giá trị thuộc tính. Quá trình này kết thúc khi ta di chuyển được đến nút lá(nút quyết định) và có được nhãn.

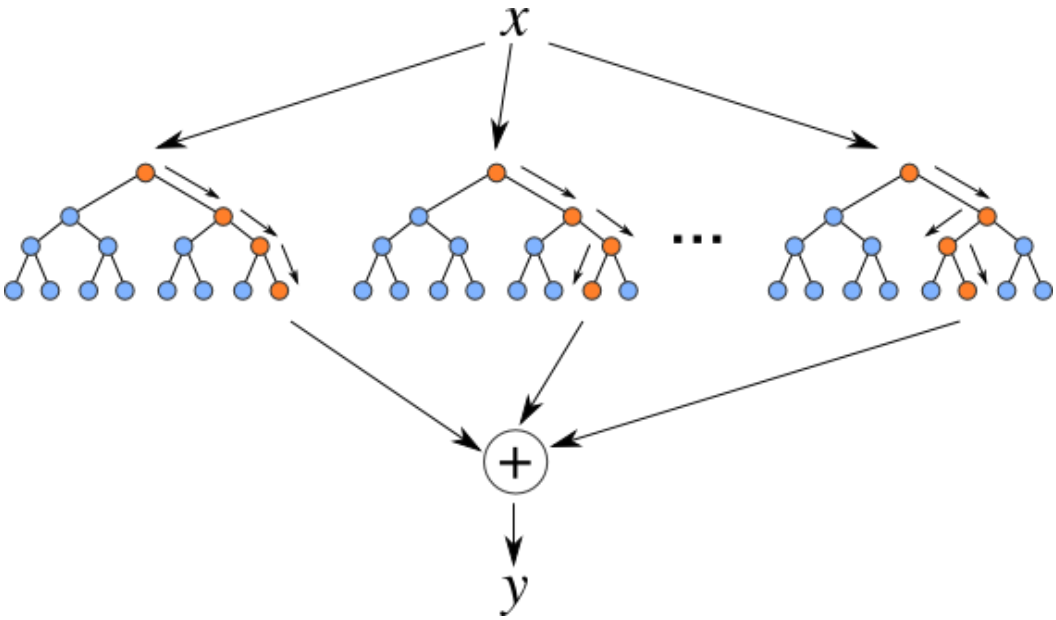
Ta tổng hợp được mã giả cho quá trình dự báo:

```
Algorithm dự_báo(mẫu_quan_sát, nút_cây_quyết_định)
  If là_lá(nút_cây_quyết_định) then
    Return nút_cây_quyết_định.giá_trị_dự_báo
  thuộc_tính_cần_quan_sát ← nút_cây_quyết_định.thuộc_tính_cần_quan_sát
  nút_con ← nút_con_theo_giá_trị_thuộc_tính(mẫu_quan_sát,
thuộc_tính_cần_quan_sát)
  Return dự_báo(mẫu_quan_sát, nút_con)
```

c, Khắc phục Overfitting bằng việc sử dụng nhiều cây quyết định

Việc sử dụng duy nhất một cây quyết định trên toàn tập dữ liệu nhiều khả năng sẽ khiến cho mô hình của chúng ta gặp vấn đề quá khớp, bởi chỉ cần gia tăng chiều sâu của cây ta có thể biểu diễn thông tin về gần như toàn bộ các mẫu trong tập huấn luyện.

Để giải quyết vấn đề này, một phương pháp tiếp cận tốt hơn đã được đề xuất. Thay vì chỉ dùng duy nhất một cây quyết định với tập huấn luyện, ta có thể xây dựng nhiều cây quyết định khác nhau từ các tập con bất kỳ của tập huấn luyện, quyết định về nhãn cuối cùng trong pha dự báo sẽ là quyết định đa số. Phương pháp này được gọi là rừng ngẫu nhiên. Rừng ngẫu nhiên tỏ ra vô cùng hiệu quả trong bài toán dự báo, quyết định cuối cùng không được đưa ra bởi một cây quyết định duy nhất mà là từ nhãn chiếm đa số từ tất cả các cây, điều này giảm thông số variance đi đáng kể khiến cho mô hình đạt được trạng thái lý tưởng.



Hình 2.2 Ảnh minh họa quá trình dự đoán với rừng ngẫu nhiên

2.1.2 Mô hình Support Vector Machine

a, Ý tưởng về phân chia lớp theo siêu phẳng

Giả sử tập dữ liệu của chúng ta là tập tách biệt tuyến tính, bằng trực giác ta có thể nảy ngay ra một ý tưởng xấp xỉ hàm dự đoán nhãn cho các mẫu quan

sát. Đúng vậy, đó là tìm một siêu phẳng phân tách chúng, các điểm dữ liệu nằm trên cùng một bờ của siêu phẳng của dữ liệu mang nhãn giống nhau và ngược lại.

Trong không gian n chiều, siêu phẳng được mô tả là một phương trình với n tham số.

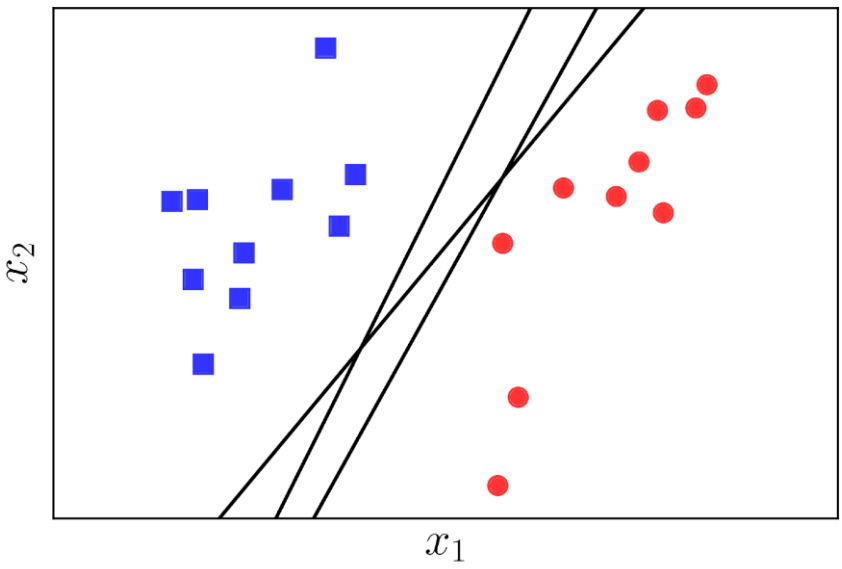
$$f(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 + \ldots + w_{n-1}x_{n-1} = 0$$

Để xác định một điểm dữ liệu thuộc bờ nào của siêu phẳng, ta chỉ cần thay điểm dữ liệu đó vào f(**x**) và quan sát dấu.

Nếu f(**x**) = 0 điểm dữ liệu nằm trên siêu phẳng

Nếu f(**x**) < 0 điểm dữ liệu thuộc bờ âm

Nếu f (**x**) > 0 điểm dữ liệu thuộc bờ dương



Hình 2.3 Tập dữ liệu với hai nhãn phân tách tuyến tính

b, Cách tìm siêu phẳng truyền thống và hạn chế

Vấn đề đặt ra là nếu siêu phẳng tồn tại, làm thế nào để ta tìm ra được chúng và nếu có nhiều siêu phẳng cùng tồn tại cái nào là cái tốt nhất ta nên chọn. Giải quyết câu hỏi đầu tiên dẫn ta đến với giải thuật học Perceptron [9] [10]. Ý tưởng của thuật toán vô cùng đơn giản, ta xuất phát với một siêu phẳng bất kỳ, tại mỗi

bước lặp ta lần lượt duyệt qua từng mẫu quan sát trong tập huấn luyện và dự đoán nhãn của chúng với siêu phẳng hiện tại. Nếu nhãn đúng ta không làm gì, nếu nhãn sai ta thực hiện hiệu chỉnh tham số của siêu phẳng. Thuật toán này có tính đúng và đã được chứng minh hội tụ, nếu tồn tại sau một số lần hữu hạn ta sẽ tìm được siêu phẳng phân tách hai lớp dữ liệu.

Công thức hiệu chỉnh cho các mẫu sai:

$$\mathbf{w} = \mathbf{w} + y_i * \mathbf{x}_i * \eta$$

Trong đó:

w là tham số của siêu phẳng

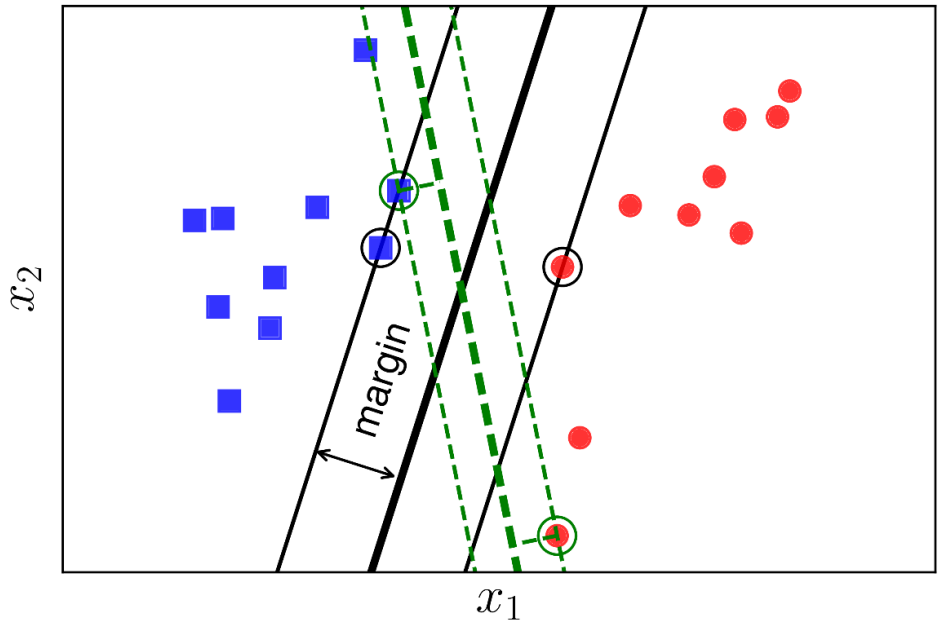
y_i là nhãn thực tế của mẫu quan sát

$\mathbf{x}_i$ là vector mô tả mẫu quan sát thứ i

η là hệ số học

c, Tìm siêu phẳng với bằng việc giải bài toán tối ưu

Vấn đề thứ nhất đã được giải quyết giờ đến với vấn đề thứ hai, làm thế nào để ta biết siêu phẳng nào là tốt nhất. Siêu phẳng tốt nhất có lẽ là siêu phẳng thỏa mãn hai điều kiện. Một là "công bằng" nhất, không thiên vị một nhãn nào hơn nhãn nào. Nếu định nghĩa khoảng cách một nhãn tới một siêu phẳng là khoảng cách từ điểm dữ liệu gần nhất mang nhãn đó tới siêu phẳng thì để đạt được "công bằng" khoảng cách từ hai nhãn tới siêu phẳng phải bằng nhau. Hai là "rộng rãi" nhất, khoảng cách từ hai nhãn tới siêu phẳng phải lớn nhất. Đó cũng chính là ý tưởng chủ chốt cho mô hình Support Vector Machine [11] [12].



Hình 2.4 So sánh về tính công bằng và rộng lượng của hai siêu phẳng

Câu trả lời trên cho câu hỏi thứ 2 lại yêu cầu ta tìm ra một hướng tiếp cận khác cho câu hỏi thứ nhất. Vậy để tìm được siêu phẳng đảm bảo cả hai yếu tố công bằng và rộng lượng ta cần làm gì.

Bằng chứng minh toán học, người ta đã xây dựng được bài toán tối ưu cho vấn đề này. Bài toán được xây dựng là bài toán quy hoạch toàn phương và hoàn toàn có thể giải quyết được bằng công cụ tìm nghiệm

Đối với SVM khâu trích chọn đặc trưng sẽ khác với rừng ngẫu nhiên. Do SVM chỉ làm việc tốt trên các tập dữ liệu được phân tách tuyến tính do đó trong trường hợp bộ MNIST không thỏa mãn đặc điểm trên có thể là một rào cản lớn. Do đó cần có một ánh xạ đưa tập MNIST huấn luyện ban đầu sang một không gian mới mà tại đó dữ liệu từ các lớp khác nhau có thể được phân tách bởi các siêu phẳng.

Kernel là một phương pháp phổ biến được sử dụng, ý tưởng của kernel là thêm một chiều mới mang giá trị được tính toán từ các chiều đã có cho các điểm dữ liệu. Việc lựa chọn phương pháp tính toán cho chiều mới không được cố định.

2.2 Hướng tiếp cận học sâu

Thay vì lấy trung bình theo từng phần tử như trong các hướng tiếp cận học máy ta có thể dữ nguyên các vector đặc trưng cho mỗi khung thời gian trong đoạn âm thanh gốc, dễ thấy các vector này xác định thứ tự và có cấu trúc giống nhau, đều cùng số chiều và tương tự về loại thông tin biểu diễn, cấu trúc dữ liệu như vậy được gọi là chuỗi thời gian [13]. Trong học sâu, có nhiều mô hình mạng nơ-ron cho phép tìm ra mối liên hệ giữa đầu vào là chuỗi thời gian và nhãn đầu ra. Mục này sẽ trình bày một số phương pháp tiếp cận sử dụng mạng nơ-ron cho bài toán nhận diện âm thanh giả mạo được tạo bởi Deepfake.

2.2.1 Mạng Dense

Mạng Dense là mô hình cơ bản nhất trong các lớp mạng nơ-ron [14]. Mạng gồm nhiều lớp, mỗi lớp bao gồm một tập các đơn vị hay nút xử lý được gọi là nơ-ron. Một nút bất kỳ ở một lớp trước có đầy đủ các liên kết tới các nút ở lớp sau và ngược lại.

a, Quá trình tính toán tại nơ-ron

Nơ-ron là đơn vị xử lý nhỏ nhất trong mạng Dense, một nơ-ron nhận đầu vào là một vector và cho ra đầu ra là một giá trị thực. Vector trên có thể là biểu diễn cho đầu vào mạng nếu nơ-ron trên thuộc lớp đầu vào hoặc là đầu ra của lớp trước nếu nơ-ron thuộc lớp ẩn. Quá trình tính toán tại các nơ ron của một lớp được xác định bởi công thức

$$y = f(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b})$$

Trong đó:

$\boldsymbol{W} \in \mathbb{R}^{\{m \times n\}}$— ma trận trọng số (m nơ − ron, n đầu vào)

$\boldsymbol{b} \in \mathbb{R}^{m}$— vector bias

$\boldsymbol{y} \in \mathbb{R}^{m}$— vector đầu ra

f là hàm kích hoạt áp dụng từng phần tử

Nhiệm vụ quan trọng nhất tại một nút là tìm mối liên hệ giữa các giữa các vô hướng trong vector đầu vào, để làm được điều đó ta thực hiện nhân chúng với trọng số của nơ-ron và lấy tổng của chúng, ma trận $\boldsymbol{W}$ và b trong công thức trên là biểu diễn cho trọng số của các nơ-ron, là giá trị mà ta sẽ đi tìm trong quá trình huấn luyện, còn $\boldsymbol{x}$ biểu diễn cho các giá trị đầu vào.

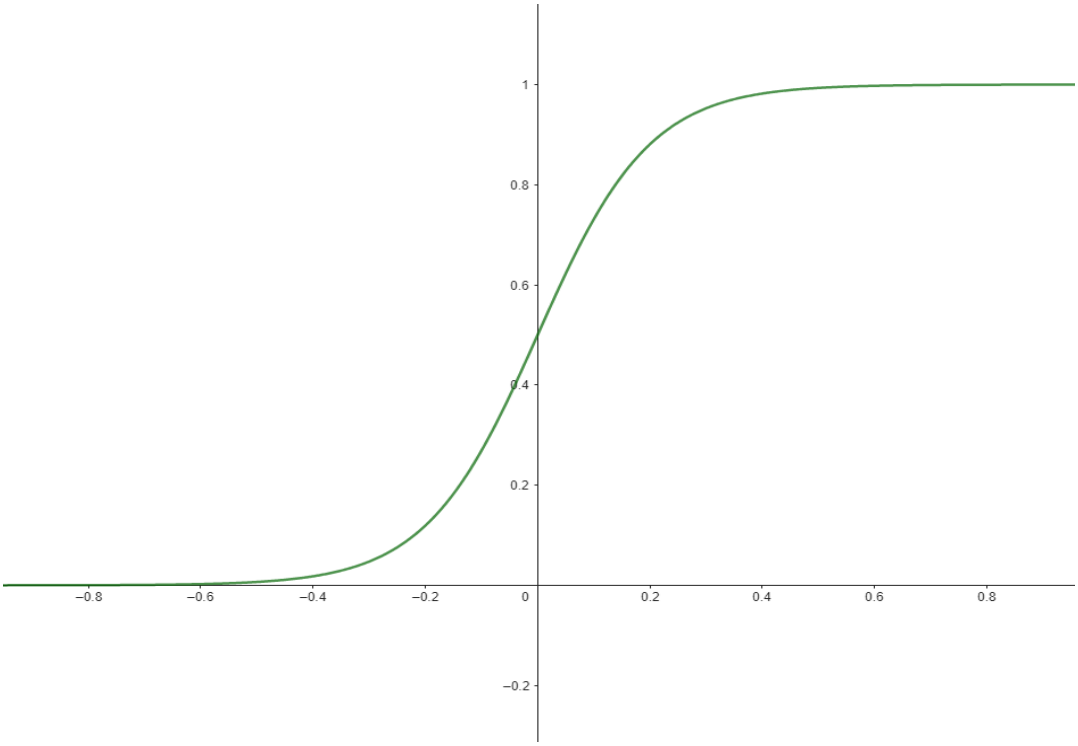
b, Hàm kích hoạt

Dễ thấy khi chỉ có phép nhân với phép cộng, mối liên hệ giữa đầu vào và đầu ra của mạng là một mối quan hệ tuyến tính, do đó nếu mối quan hệ giữa đầu vào và đầu ra trên dữ liệu thực tế là phi tuyến mạng sẽ không thể biểu diễn được. Vậy nên sau quá trình nhân và cộng, giá trị đầu ra sẽ được đưa qua một hàm kích hoạt [15] để làm mềm giá trị đầu ra đồng thời tạo tính phi tuyến cho mô hình. Điều này giúp mạng mở rộng khả năng biểu diễn và có thể xấp xỉ được các mối quan hệ phi tuyến. Dưới đây là một số hàm kích hoạt phổ biến thường được sử dụng.

Hàm sigmoid

Công thức:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

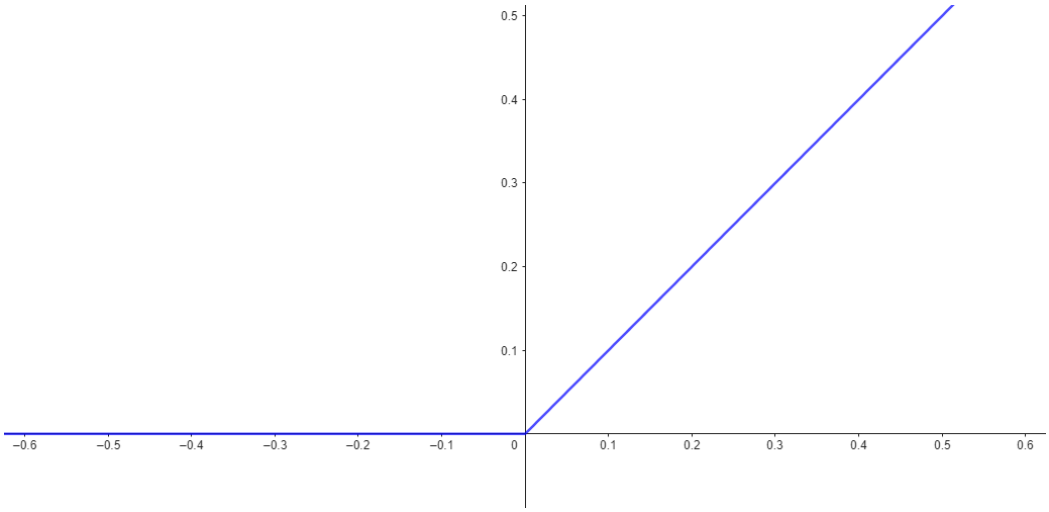


Hình 2.5 Đồ thị hàm sigmoid

Hàm Relu

Công thức

$$ReLU(x) = max(0, x)$$

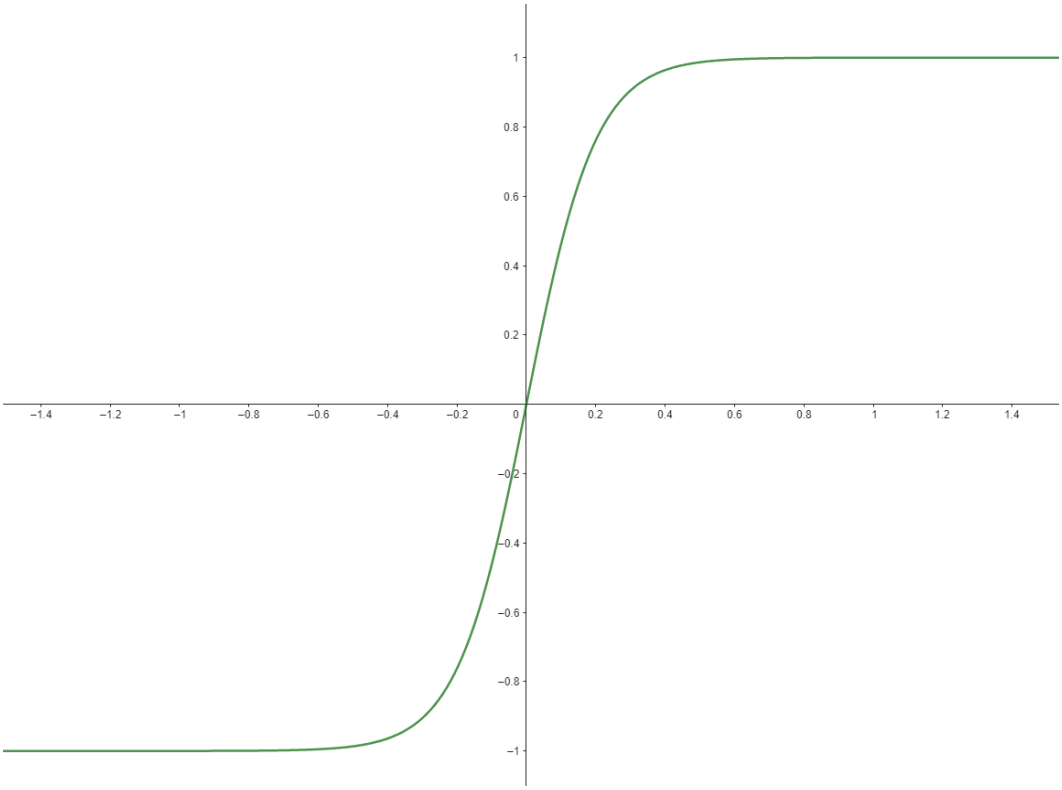


Hình 2.6 Đồ thị hàm ReLU

Hàm tanh

Công thức

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Hình 2.7 Đồ thị hàm tanh

c, Giải thuật học

Bên cạnh việc đảm bảo một kiến trúc nơ-ron có đủ khả năng xấp xỉ mối liên hệ giữa đầu vào và đầu ra ta cũng phải cần tìm ra được bộ tham số cho phép mạng làm được điều đó. Công đoạn này được gọi là quá trình tối ưu tham số, chi tiết cụ thể các bước trong công đoạn được gọi là giải thuật học.

Một trong những cách phổ biến nhất để xây dựng giải thuật học cho mạng nơ-ron là xây dựng một hàm đo đạc độ khác biệt giữa giá trị mạng cho ra và giá trị thực tế sau đó tìm bộ $\boldsymbol{W}$ và b sao cho giá trị hàm trên nhỏ nhất. Hàm trên được gọi là hàm mất mát, quá trình tìm bộ $\boldsymbol{W}$ và b nhằm tối thiểu hóa hàm mất mát được gọi là quá trình huấn luyện.

Trong bài toán nhận diện âm thanh giả mạo đầu ra của của mạng là một giá trị thực lớn hơn 0 và nhỏ hơn 1 biểu diễn xác suất cho biết âm thanh đó có phải giả mạo hay không. Vậy nên hiển nhiên giá trị nhãn thực tế của các mẫu âm thanh giả mạo sẽ là 1 còn các mẫu âm thanh thường sẽ là 0. Để đo độ chênh lệnh giữa giá trị xác suất đầu vào và giá trị xác suất thực tế ta sử dụng hàm Log Loss được xác định bởi công thức

$$L = -\left(\frac{1}{N}\right) \cdot \sum[\, y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)]$$

Trong đó:

L: *tổng hàm mất mát trung bình*

N: *số mẫu*

y_i: *nhãn thực tế* (0 *hoặc* 1)

$\hat{y}_i$: *xác suất mô hình dự đoán mẫu* i *thuộc lớp* 1

log: *logarit tự nhiên* (*cơ số* e) *tổng từ* $i = 1$ *đến* N

Sau khi định nghĩa được hàm mất mát ta cần tìm ra bộ $\boldsymbol{W}$ và b sao cho hàm mất mát nhỏ nhất. Thông thường ta sẽ sử dụng giải thuật học gradient descent, đây là một ý tưởng được lấy cảm hứng từ giải tích, xuất phát từ một điểm, muốn đi tới cực tiểu của một hàm số ta chỉ cần đi ngược chiều đạo hàm tại điểm đó. Hay nói một cách cụ thể bằng toán học, ta sẽ lấy một giá trị bắt đầu cụ thể của $\boldsymbol{W}$ và b sau đó lặp lại việc tính đạo hàm của hàm mất mát theo $\boldsymbol{W}$ và b tại điểm hiện tại và hiệu chỉnh bằng việc trừ $\boldsymbol{W}$ và b đi bằng một lượng là đạo hàm nhân với hệ số học. Quá trình trên sẽ kết thúc khi giá trị hiệu chỉnh quá nhỏ và không còn đáng kể để hiệu chỉnh.

d, Hạn chế của mạng Dense với bài toán chuỗi thời gian

Do mạng Dense chỉ quan sát các giá trị đặc trưng đầu vào một cách độc lập và đưa ra dự đoán do đó mạng sẽ khó có thể học được các thông tin về mối liên hệ giữa các giá trị đó trong chuỗi thời gian.

2.2.2 Mạng RNN

a, Cơ chế tính toán

Mạng RNN ra đời nhằm giải quyết vấn đề trên của mạng Dense. Thay vì nối các vector đặc trưng trong chuỗi thời gian thành một vector duy nhất rồi tính toán qua các lớp ẩn và cho ra đầu ra, mạng RNN sẽ tính toán đầu một giá trị **h** cho mọi bước trong chuỗi thời gian sử dụng vector đặc trưng tại bước đó và giá trị **h** tính ở bước trước đó. Đầu ra y sẽ được tính bằng giá trị vector đặc trưng tại bước cuối cùng và giá trị h tại bước cuối cùng. Công thức cụ thể cho từng giai đoạn được trình bày ở phía dưới:

Bước tính $\mathbf{h_t}$

$$\boldsymbol{h_t} = tanh(\boldsymbol{W_{xh}x_t} + \boldsymbol{W_{hh}h_{t-1}} + \boldsymbol{b_h})$$

Bước tính đầu ra cuối:

$$\boldsymbol{y} = \boldsymbol{W_{hy}h_t} + \boldsymbol{b_y}$$

Dễ dàng thấy rằng ở RNN không chỉ có duy nhất một ma trận tham số $\boldsymbol{W}$ mà có tới ba ma trận tham số là $\boldsymbol{W_{xh}}$, $\boldsymbol{W_{hh}}$ *và* $\boldsymbol{W_{hy}}$. Trong đó $\boldsymbol{W_{xh}}$ là ma trận sẽ học các đặc trưng từ vector đầu vào, $\boldsymbol{W_{hh}}$ là ma trận sẽ học các đặc trưng từ trạng thái trước đó dùng để kết hợp với đầu vào còn $\boldsymbol{W_{hy}}$ là ma trận sinh ra đầu ra từ trạng thái kết hợp. Nhờ có ma trận $\boldsymbol{W_{xh}}$ và $\boldsymbol{W_{hh}}$ mô hình đã có thể kết hợp được thông tin của các vector đặc trưng trong chuỗi thời gian một cách có cấu trúc từ đó học được các mối liên hệ phức tạp hơn.

Đối với bài toán nhận diện âm thanh giả mạo được tạo bởi Deepfake, đầu ra y cuối cùng sẽ được đi qua hàm sigmoid nhằm chuẩn hóa giá trị về xác suất trong khoảng từ 0 đến 1.

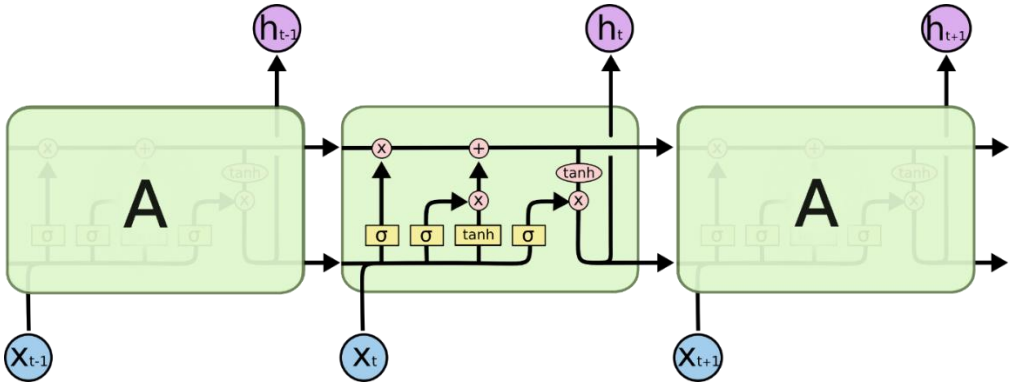
b, Hạn chế của mạng RNN với bài toán chuỗi thời gian

Mặc dù cho phép học được mối liên hệ giữa các thành phần trong chuỗi thời gian nhưng RNN lại mang một hạn chế đáng kể trong việc ghi nhớ các thông

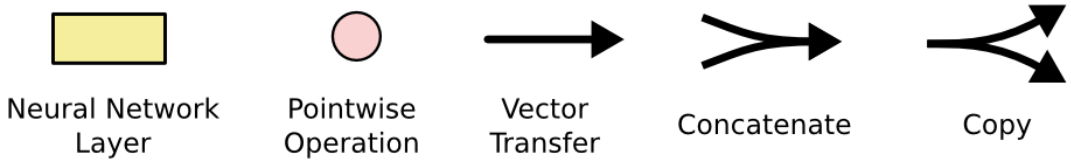
tin phụ thuộc xa, cụ thể hơn một vector đặc trưng càng nằm gần ở phần đầu của chuỗi thời gian thì lại càng ít ảnh hưởng tới giá trị đầu ra. Thêm nữa trong quá trình huấn luyện, đạo hàm của các tham số trong mạng qua các lớp RNN giảm về 0 một cách nhanh chóng khiến cho việc cập nhật trọng số trở lên không đáng kể, hiện tượng này được gọi là gradient vanishing.

2.2.3 Mạng LSTM

Để giải quyết các nhược điểm của mạng RNN cho bài toán dự đoán chuỗi thời gian, mạng LSTM [16] [17] được ra đời. Đây cũng chính là phương pháp được sử dụng trong đề tài này nhằm giúp nhận dạng âm thanh giả mạo. LSTM được thiết kế để tránh khỏi hiện tượng gradient vanishing, nhờ đó việc nhớ thông tin lâu dài là đặc tính nội tại của mạng, ta không cần phải huấn luyện để mạng có thể nhớ dài giống như trong kiến trúc RNN. LSTM được thiết kế theo kiến trúc mô-đun, cũng giống như RNN để xử lý thông tin chuỗi thời gian, LSTM là tập các khối xử lý nối tiếp chịu trách nhiệm tính toán từng bước trong chuỗi thời gian bằng việc sử dụng thông tin được tính toán trước đó và thông tin tại bước thời gian đang xét. LSTM khác RNN ở chỗ thay vì chỉ có một khối **tanh** duy nhất được sử dụng để tính toán trạng thái ẩn, mạng có tận 4 khối tương tác với nhau.



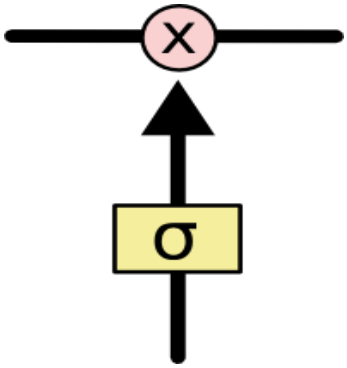
Hình 2.8 Sơ đồ tính toán tại một bước thời gian trong LSTM



Hình 2.9 Mô tả ký hiệu trong sơ đồ

a, Ý tưởng về cell state và cổng lọc thông tin

Ý tưởng chính đằng sau kiến trúc LSTM là cell state, được tượng chưng bởi đường chạy ngang qua tất cả các bước tính toán trên hình ảnh trên. Cell state là đối tượng mang thông tin chính trong mạng LSTM, thông tin trên cell state có thể được bổ sung hoặc lược bớt tại mỗi bước tính toán trong chuỗi thời gian dựa trên một cấu trúc bao gồm các cổng lọc thông tin.

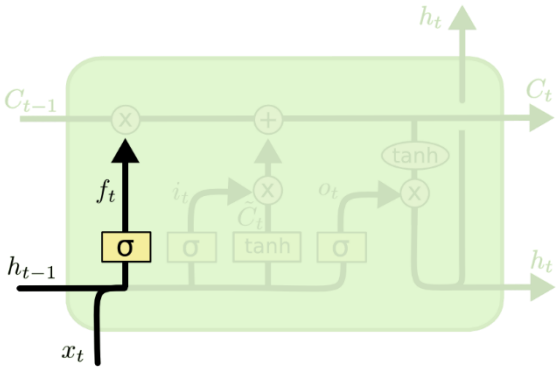


Hình 2.10 Cổng sàng lọc thông tin

Một cổng lọc thông tin bao gồm một lớp nơ-ron mang hàm kích hoạt sigmoid và một toán tử theo phần tử tương ứng. Nhờ vào đặc tính cho ra giá trị đầu ra nằm giữa khoảng 0 và 1, cổng lọc thông tin sẽ quyết định có bao nhiêu phần trăm nên được đi qua cổng. Với giá trị 1, toàn bộ thông tin sẽ được đi qua cổng còn với giá trị 0 sẽ không có thông tin nào được đi qua cổng. Trong một mạng LSTM điển hình sẽ có 4 loại cổng như vậy là cổng quên, cổng vào, cổng ra, tầng tanh(tầng tanh sử dụng hàm kích hoạt tanh).

b, Cách thức hoạt động của mạng

Khi thông tin đi vào mạng, đầu tiên nó sẽ đi qua cổng quên, cổng quên có nhiệm vụ quyết định xem giữ lại bao nhiêu phần thông tin từ cell state trước đó.



Hình 2.11 Cổng quên

Giá trị đầu ra của cổng quên được xác định bởi công thức

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Trong đó

f_t là giá trị đầu ra của cổng quên

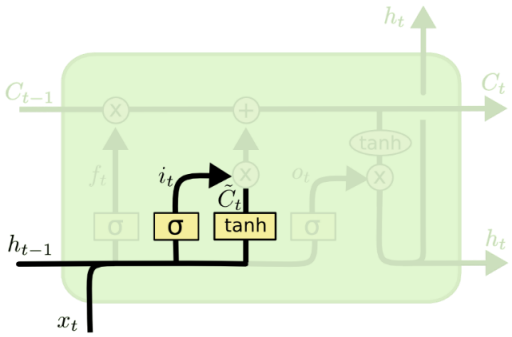
W_f là ma trận trọng số của lớp nơ-ron trong cổng

h_{t-1} là vector trạng thái ẩn ở bước trước đó

x_t là vector đặc trưng tại thời điểm t trong chuỗi thời gian

b_f là vector trọng số bias của lớp nơ-ron trong cổng

Sau khi quá trình tính toán tại cổng quên kết thúc, giá Cell state mới sẽ được cập nhật bằng cách kết hợp giá trị của cổng vào, tầng tanh và giá trị của cổng quên.



Hình 2.12 Quá trình tính toán và cập nhật cell state

Giá trị của cổng vào và tầng tanh được xác định bởi công thức:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\widetilde{C_t} = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Trong đó

W_i, W_C, b_C, b_i là tham số tương ứng với từng cổng

i_t là đầu ra của cổng vào

$\widetilde{C_t}$ là đầu ra của tầng tanh

Kết thúc quá trình tính toán là quá trình kết hợp các giá trị $i_t, \widetilde{C_t}, f_t$ thành C_t. Công thức kết hợp được xác định như sau:

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t}$$

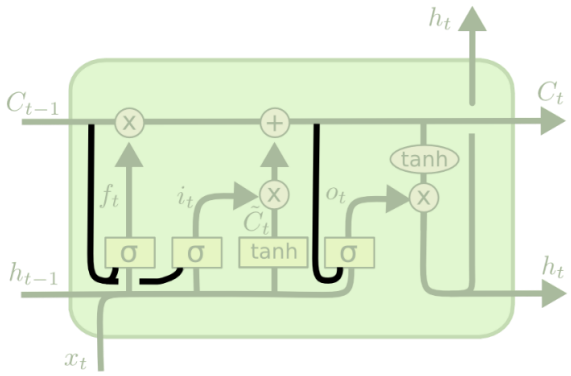
Sau cùng trạng thái ẩn h_t ở bước thời gian *t* được tính toán bằng công thức

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh\,(C_t)$$

c, Một số biến thể của mạng LSTM

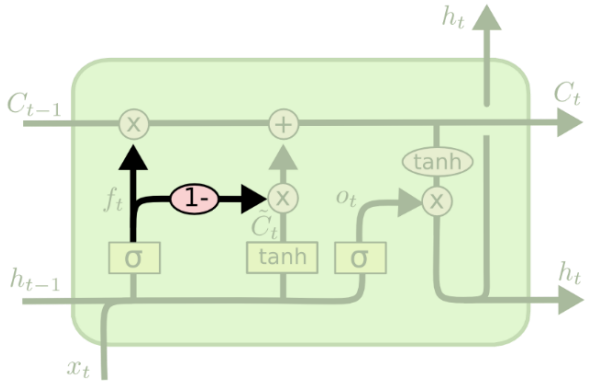
Ý tưởng được trình bày phía trên là thiết kế cơ bản nhất của một mạng LSTM, ngoài thiết kế trên LSTM còn có một số biến thể với các thay đổi nhỏ. Điển hình như biến thể được đề xuất bởi Gers & Schmidhuber.



Hình 2.13 Biến thể được đề xuất bởi Gers & Schmidhuber

Trong biến thể này, ngoài sử dụng h_{t-1} và x_t cho quá trình tính toán f_t, i_t, o_t ta còn sử dụng thêm cả C_{t-1}và C_t, điều này giúp các cổng có đủ thông tin nhất có thể để đưa ra quyết định.

Một biến thể khác cũng được sử dụng rộng rãi, trong biến thể này thay vì sử dụng hai cổng riêng biệt là cổng quên và cổng vào để quyết định xem có bao nhiêu phần nên quên từ thông tin cũ và có bao nhiêu phần nên nhớ từ thông tin mới, lượng quên và nhớ sẽ được xác định đồng thời bởi một cổng duy nhất.



Hình 2.14 Biến thể sử dụng chung một cổng cho việc nhớ và quên

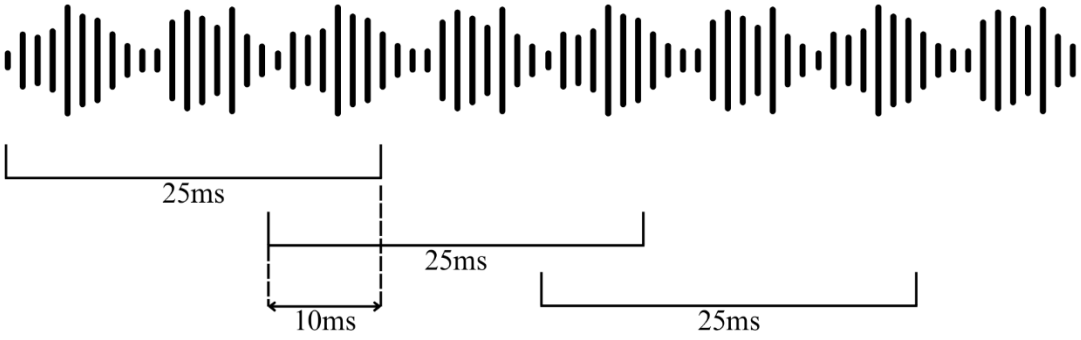
Phương pháp này giúp giảm lượng tham số học và giúp mô hình tránh khỏi hiện tượng quá khớp.

d, Mạng LSTM cho bài toán phát hiện âm thanh giả mạo

Trong bài toán phát hiện âm thanh giả mạo được tạo bởi Deepfake, hệ thống phát hiện âm thanh giả mạo sử dụng kiến trúc LSTM sẽ được triển khai như sau.

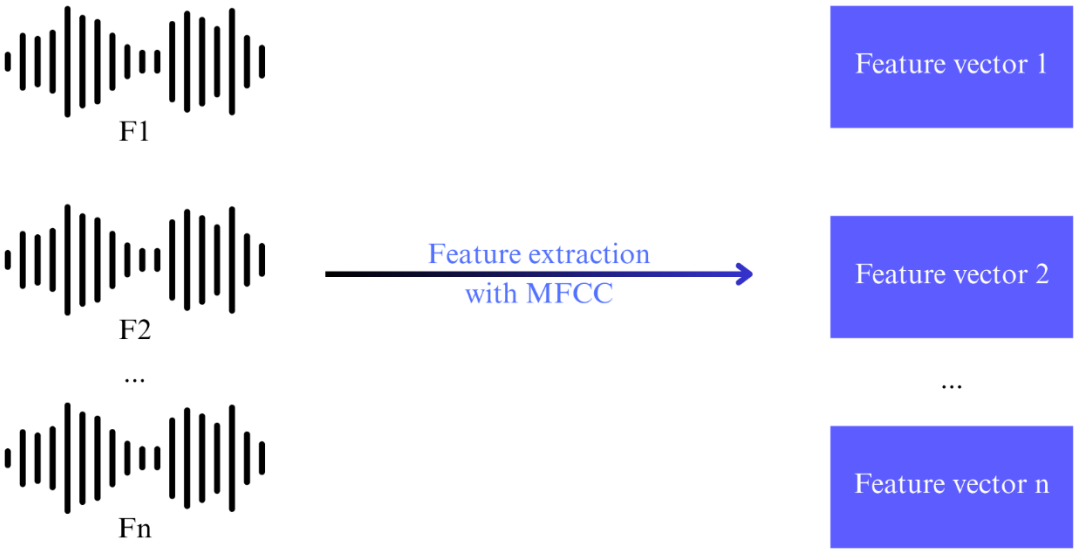
Khâu chuyển âm thanh thô trên một mẫu thành tập các vector đặc trưng

Như đã trình bày ở chương 1, đầu vào của hệ thống là 4 giây âm thanh thô, các đoạn âm thanh này sẽ được cắt thành các đoạn dài 25ms, mỗi đoạn trong chuỗi âm thanh sẽ giao với đoạn trước và sau đó một khoảng là 10ms.



Hình 2.15 Quá trình phân đoạn âm thanh đầu vào

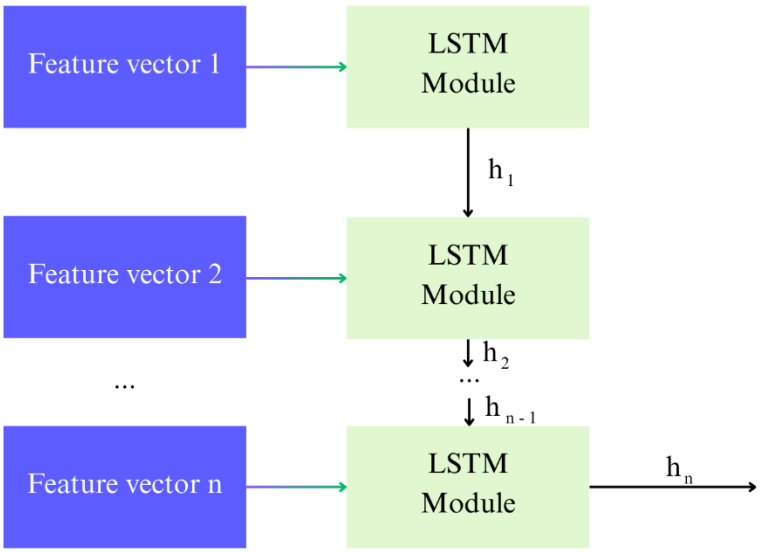
Sau đó thực hiện phương pháp trích xuất đặc trưng MFCC trên mỗi đoạn để tạo ra tập vector đặc trưng.



Hình 2.16 Quá trình trích chọn đặc trưng cho mỗi khung

Khâu đưa âm thanh thô qua lớp LSTM

Sau bước trên, lần lượt các vector đặc trưng theo thứ tự được đưa vào trong mô-đun LSTM, đầu vào của mỗi thời điểm là đầu ra của thời điểm trước đó và vector đặc trưng tại thời điểm hiện tại. Lưu ý trong hình vẽ dưới đây, các LSTM Module là một và dùng chung một bộ tham số.



Hình 2.17 Quá trình tính toán tại lớp LSTM

Kết thúc quá trình này chúng ta thu được một vector đầu ra duy nhất là $\mathbf{h_1}$ trong $\mathbf{h_1}$ chứa đầy đủ các thông tin về ngữ cảnh, chúng ta có thể đưa $\mathbf{h_1}$ vào các lớp Dense với 2 nơ-ron ở lớp cuối cùng hàm kích hoạt softmax để dự đoán xác suất đầu ra cho mỗi lớp.

2.3 Tóm tắt chương

- Sau quá trình trích chọn đặc trưng thô, dữ liệu sẽ tiếp tục được đưa qua các mô hình để huấn luyện và đánh giá. Các mô hình được sử dụng trong đề tài được chia ra làm hai nhóm chính bao gồm
    - Các mô hình học máy: Random Forest, SVM
    - Các mô hình học sâu: RNN, LSTM

- Nhóm các mô hình học máy làm việc với giá trị trung bình của các vector đặc trưng trong chuỗi thời gian
- Nhóm các mô hình học sâu làm việc trực tiếp với các vector trong chuỗi thời gian

CHƯƠNG 3 THỰC NGHIỆM

3.1 Môi trường thực nghiệm

3.1.1 Phần cứng

Cấu hình thiết bị được sử dụng trong quá trình thực nghiệm như sau:

- CPU: Intel Core i5-12500H (12th Gen, 12 nhân, 16 luồng, xung nhịp cơ bản 2.50GHz)
- RAM: 16.0 GB DDR4
- GPU: NVIDIA GeForce RTX 3060 Laptop GPU (6GB VRAM, kiến trúc Ampere)
- Ổ cứng: SSD NVMe 512GB
- Việc huấn luyện mô hình học sâu (RNN và LSTM) được thực hiện trên GPU RTX 3060 nhằm tăng tốc xử lý và rút ngắn thời gian huấn luyện. Việc tận dụng phần cứng hỗ trợ CUDA nhằm cải thiện hiệu suất so với huấn luyện trên CPU đơn thuần.

3.1.2 Phần mềm

Các phần mềm và thư viện được sử dụng trong quá trình xây dựng, huấn luyện và thử nghiệm mô hình bao gồm:

Hệ điều hành: Windows 11 Home, 64-bit

Ngôn ngữ lập trình: Python 3.10

Thư viện và framework:

- numpy, pandas: xử lý số liệu và dữ liệu đặc trưng
- scikit-learn: xây dựng và đánh giá các mô hình Random Forest, SVM
- soundfile, torchaudio, librosa: đọc, xử lý và trích xuất đặc trưng âm thanh
- matplotlib, seaborn: trực quan hóa kết quả

PyTorch: xây dựng và huấn luyện mô hình học sâu (RNN, LSTM)

Môi trường phát triển: PyCharm

Công cụ hỗ trợ: GPU hỗ trợ CUDA (NVIDIA RTX 3060 Laptop GPU)

3.1.3 Bộ dữ liệu

Bộ dữ liệu được sử dụng trong đề tài là bộ SceneFake, một tập dữ liệu tổng hợp gồm các đoạn âm thanh gồm 2 nhãn là real và fake. Bộ dữ liệu này được sử dụng với mục tiêu phát hiện các tín hiệu giọng nói bị giả mạo bằng các kỹ thuật tổng hợp giọng nói như Deepfake. Bộ được chia làm 3 tập con là train, dev, eval. Tập train được dùng cho mục đích huấn luyện, tập dev được dùng cho mục đích hiệu chỉnh siêu tham số và tập eval dùng cho mục đích đánh giá. Số lượng mẫu và phân bố nhãn trên mỗi tập được trình bày trong bảng phía dưới.

Bảng 3.1 Số lượng mẫu và thống kê nhãn trên mỗi tập con của bộ SceneFake

Tập dữ liệu	**Số lượng mẫu nhãn real**	**Số lượng mẫu nhãn fake**	**Tổng cộng**
train	2.525	10.660	13.185
dev	2.548	10.295	12.843
eval	6.334	26.412	32.746
Tổng cộng	**11.407**	**47.367**	**58.774**

Mỗi mẫu trong bộ dữ liệu SceneFake là một tệp âm thanh định dạng wav, với tần số lấy mẫu 16kHz có thời lượng bất kỳ. Có thể thấy bộ dữ liệu có sự mất cân bằng đáng kể giữa số lượng mẫu âm thanh thật và âm thanh giả, trong đó số lượng âm thanh giả nhiều gấp hơn 4 lần. Điều này cần được lưu ý trong quá trình huấn luyện mô hình để tránh mô hình bị lệch nhãn.

3.2 Chiến lược huấn luyện

Trong đề tài, cả bốn mô hình được đề cập ở chương 2 gồm Random Forest, Support Vector Machine, RNN, LSTM đều sẽ được cài đặt để huấn luyện, so sánh nhận xét.

3.2.1 Đặc trưng đầu vào

Các mô hình học máy truyền thống Random Forest và SVM sẽ sử dụng duy nhất một vector là trung bình của chuỗi vector theo thời gian.

Các mô hình học sâu RNN và LSTM sẽ sử dụng toàn bộ chuỗi vector MFCC theo thời gian.

3.2.2 Tham số huấn luyện

Đối với hai mô hình học máy SVM và RandomForest, do có thời gian huấn luyện ngắn, các siêu tham số sẽ không được quyết định tĩnh, thay vào đó sẽ sử dụng Grid Sreach để tìm kiếm tham số tối ưu nhất trên tập dev. Không gian tìm kiếm tham số của từng thuật toán được đề cập trong bảng dưới.

Bảng 3.2 Không gian tìm kiếm siêu tham số cho thuật toán Random Forest

Tên siêu tham số	**Ý nghĩa**	**Không gian tìm kiếm**
n_estimators	Số cây trong rừng	[100, 200, 500, 1000]
max_depth	Độ sâu tối đa của mỗi cây	[None, 10, 20, 30, 50]
min_samples split	Số lượng mẫu tối thiểu để chia một node	[2, 5, 10]
min_samples_leaf	Số lượng mẫu tối thiểu tại mỗi lá	[1, 2, 4]
max_features	Số đặc trưng được xét tại mỗi split	['auto', 'sqrt', 'log2']
bootstrap	Có dùng bootstrap mẫu dữ liệu không	[True, False]
class weight	Giải quyết vấn đề mất cân bằng nhãn	[None, 'balanced', 'balanced_subsample']

Bảng 3.2 Không gian tìm kiếm siêu tham số cho thuật toán SVM

Tên siêu tham số	**Ý nghĩa**	**Không gian tìm kiếm**
C	Hệ số phạt lỗi (trade-off giữa margin và lỗi)	[0.1, 1, 10, 100, 1000]
kernel	Hàm kernel	['linear', 'rbf', 'poly', 'sigmoid']
gamma	Độ ảnh hưởng của điểm mẫu (chỉ với RBF/poly)	['scale', 'auto', 0.01, 0.001, 0.0001]
degree	Bậc của kernel poly	[2, 3, 4, 5]
coef0	Hệ số độc lập trong kernel poly, sigmoid	[0.0, 0.1, 0.5, 1.0]
class_weight	Cân bằng nhãn	[None, 'balanced']

Đối với hai mô hình học sâu RNN và LSTM, do thời gian huấn luyện lớn nên trong đề tài sẽ không duyệt siêu tham số để tìm ra cấu hình tốt nhất, thay vào đó siêu tham số sẽ được khởi tạo tĩnh ngay từ đầu. Chi tiết siêu tham số được liệt kê cụ thể trong bảng phía dưới.

Bảng 3.3 Siêu tham số cho mạng LSTM và RNN

Thành phần	LSTM	RNN
Tầng tuần tự	nn.LSTM(input_size=13, hidden_size=64, batch_first=True)	nn.RNN(input_size=13, hidden_size=64, batch_first=True)
Kích thước trạng thái ẩn	64	64
Tầng ẩn phi tuyến	Linear(64 → 32) kết hợp ReLU()	Linear(64 → 32) kết hợp ReLU()
Tầng đầu ra	Linear(32 → 2)	Linear(32 → 2)
Hàm mất mát	CrossEntropyLoss(weight=torch.tensor([4.15, 1.0]))	CrossEntropyLoss(weight=torch.tensor([4.15, 1.0]))
Trọng số lớp	[4.15, 1.0]	[4.15, 1.0]
Số lớp đầu ra	2 (phân loại: âm thanh thật hoặc giả)	2 (phân loại: âm thanh thật hoặc giả)

3.2.3 Hàm mất mát cho bộ dữ liệu mất cân bằng

Đối với hai mô hình học sâu, hàm mất mát được sử dụng là CrossEntropyLoss, do bộ dữ liệu SceneFake có sự mất cân bằng nhãn rõ rệt giữa số lượng âm thanh thật và giả nên trong đề tài sẽ sử dụng biến thế có trọng số của hàm mất mát:

$$L(x, y) = -w_y \cdot \log\left(\frac{e^{x_y}}{\sum_j e^{x_j}}\right)$$

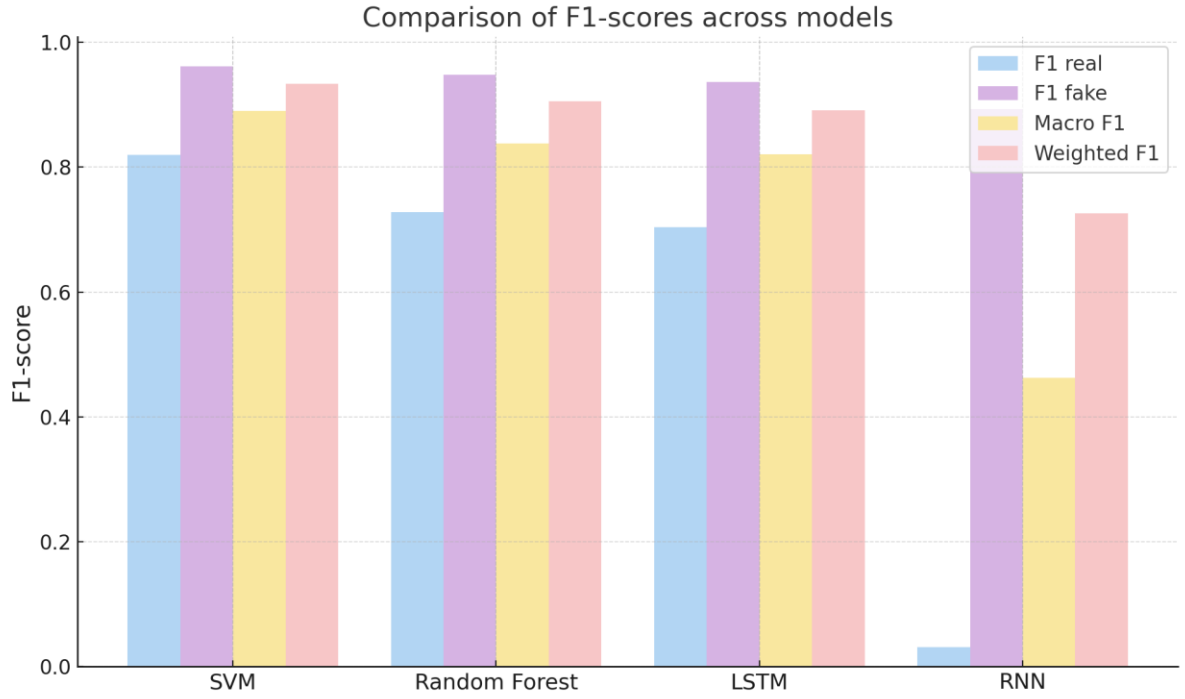
Trong đó w_y là trọng số tương ứng với lớp y, được đặt theo tỉ lệ ngược với số lượng mẫu mỗi lớp. Cụ thể, các trọng số được sử dụng là:

- Lớp real: 4.15
- Lớp fake: 1.0

3.3 Kết quả thực nghiệm

Bảng 3.4 Kết quả thực nghiệm

Mô hình	Accuracy	F1-score (real)	F1-score (fake)	Macro F1	Weighted F1
SVM	93.60%	0.8200	0.9611	0.8905	0.9338
Random Forest	91.35%	0.7279	0.9485	0.8382	0.9059
LSTM	89.55%	0.7038	0.9365	0.8202	0.8915
RNN	80.76%	0.0311	0.8932	0.4621	0.7265



Hình 3.1 Biểu đồ so sánh điểm F1-score giữa các mô hình

Bốn mô hình được huấn luyện và đánh giá trong nghiên cứu này gồm hai nhóm chính: mô hình học máy truyền thống (SVM, Random Forest) và mô hình học sâu (LSTM, RNN). Kết quả thực nghiệm cho thấy sự khác biệt rõ rệt về hiệu suất và đặc tính hoạt động giữa hai nhóm phương pháp.

Đầu tiên với SVM, mô hình SVM sử dụng kernel RBF và các siêu tham số được tối ưu thông qua GridSearchCV đã cho kết quả tốt nhất trong tất cả các mô hình. Cụ thể, mô hình đạt:

- Accuracy: 93.60%
- F1-score lớp thật: 0.8200
- F1-score lớp giả: 0.9611

Điều đáng chú ý là SVM duy trì được sự cân bằng giữa precision và recall ở cả hai lớp, giúp tổng thể macro F1 và weighted F1 đều đạt mức cao. Điều này cho thấy mô hình có khả năng tổng quát hóa tốt, đặc biệt khi đã chuẩn hóa dữ liệu đầu vào và xử lý tốt sự mất cân bằng nhãn thông qua trọng số trong hàm mất mát. Ngoài ra, do số chiều đặc trưng không quá lớn, SVM hoạt động rất hiệu quả và nhanh.

Tiếp đến là Random Forest, Mặc dù không đạt kết quả cao bằng SVM, Random Forest vẫn là một mô hình mạnh với độ chính xác 91.35% và F1-score lớp giả lên đến 0.9485. Tuy nhiên:

- F1-score lớp thật chỉ đạt 0.7279
- Recall lớp thật rất thấp (chỉ 59.84%)

Điều này phản ánh rằng mô hình Random Forest có xu hướng thiên lệch về lớp chiếm đa số, do bản chất thuật toán bootstrap sampling không cân đối số lượng mẫu giữa các lớp. Mặc dù sử dụng số lượng cây lớn và tối ưu siêu tham số, hiệu suất lớp thiểu số vẫn không được cải thiện đáng kể. Điều này gợi ý rằng cần kết hợp thêm các kỹ thuật như class weighting hoặc oversampling để cân bằng.

Xếp tiếp ngay sau đó ta có LSTM, LSTM được kỳ vọng sẽ thể hiện tốt trong bài toán nhận diện tín hiệu tuần tự như âm thanh, nhờ vào khả năng lưu giữ thông tin dài hạn. Kết quả mô hình đạt:

- Accuracy: 89.55%
- F1-score lớp giả: 0.9365 (rất cao)
- F1-score lớp thật: 0.7038

Mặc dù có khả năng nhận diện âm thanh giả hiệu quả, LSTM vẫn chưa khai thác hết tiềm năng nhận diện âm thanh thật. Lý do có thể đến từ kích thước tập huấn luyện tương đối nhỏ so với nhu cầu của mạng nơ-ron sâu.

Xếp chót là RNN, RNN là mô hình hoạt động kém nhất trong bốn phương pháp được đánh giá. Kết quả rất mất cân bằng:

- F1-score lớp thật: chỉ 0.0311
- F1-score lớp giả: 0.8932
- Accuracy tổng thể: 80.76%

Mặc dù RNN vẫn có khả năng phát hiện âm thanh giả tương đối tốt, nhưng hầu như bỏ qua lớp âm thanh thật. Điều này bắt nguồn từ việc RNN truyền thống không thể duy trì gradient ổn định cho chuỗi dài như LSTM, dẫn đến hiện tượng "quên" ngữ cảnh quan trọng.

3.5 Tóm tắt chương

- Quá trình thực nghiệm được thực hiện trên thiết bị có cấu hình:
    - CPU: Intel Core i5-12500H (12th Gen, 12 nhân, 16 luồng, xung nhịp cơ bản 2.50GHz)
    - RAM: 16.0 GB DDR4
    - GPU: NVIDIA GeForce RTX 3060 Laptop GPU (6GB VRAM, kiến trúc Ampere)
- Có tổng cộng 4 mô hình được cài đặt bao gồm
    - Mô hình học máy truyền thống
        - Random Forest
        - Support Vector Machine
    - Mô hình học sâu
        - Long Short-Term Memory
        - Recurrent Neural Network

- Sau quá trình thực nghiệm, SVM là mô hình cho ra kết quả tốt nhất, sau đó đến Random Forest, Long Short-Term Memory và Recurrent Neural Network
- SVM hoạt động tốt nhất do tập huấn luyện sau khi tiền xử lý và sử dụng kernel để bổ sung chiều thì gần như đã phân tách tuyến tính trong không gian mới, điều này khiến mô hình dễ dàng đạt được trạng thái tổng quát hóa
- Random Forest hoạt động tốt nhưng không phải lựa chọn tối ưu bởi tính không có khả năng tự cân bằng khi có hiện tượng chênh lệch nhãn lớn
- LSTM hoạt động ở mức trung bình do số lượng mẫu dữ liệu vẫn còn quá nhỏ so với lượng tham số dùng để học
- RNN cho ra kết quả kém nhất do ảnh hưởng bởi hiện tượng gradient vanishing, làm mất mát thông tin ngữ cảnh

CHƯƠNG 4 DEMO SẢN PHẨM

Trong chương này báo cáo sẽ trình bày sơ lược về chương trình demo được tích hợp module đã được xây dựng trong phần trước. Nội dung cụ thể không đi sâu vào phân tích thiết kế và cài đặt mà chỉ dừng lại ở việc mô tả chức năng cũng như đánh giá các hoạt động. Chi tiết hơn về mã nguồn sẽ được trình bày trong phần phụ lục của tài liệu.

4.1. Mô tả chung

4.1.1 Công nghệ sử dụng

- Ngôn ngữ lập trình Python: Python được sử dụng làm ngôn ngữ chính để xây dựng toàn bộ hệ thống. Với cú pháp rõ ràng, thư viện phong phú và khả năng tích hợp tốt với các công cụ học máy, Python là một lựa chọn phổ biến cho các ứng dụng liên quan đến xử lý tín hiệu và trí tuệ nhân tạo
- Thư viện đồ họa Tkinter: Đây là thư viện đồ họa được tích hợp sẵn trong Python để tạo giao diện đồ họa
- Thư viện học sâu Pytorch: Pytorch được dùng trong bước này để phục vụ mục đích nạp các tham số đã huấn luyện của mô hình LSTM

4.1.2 Phân tích kịch bản sử dụng

Một người dùng nghi ngờ rằng một đoạn ghi âm trong cuộc trò chuyện trực tuyến có thể đã bị làm giả bằng công nghệ Deepfake. Người dùng quyết định sử dụng công cụ demo để kiểm tra độ xác thực của đoạn âm thanh này. Quy trình xác thực âm thanh giả mạo như sau:

- Người dùng nghe được một giọng nói tiếng Anh và nghi ngờ là âm thanh giả mạo được tổng hợp bởi Deepfake
- Người dùng ghi âm lại bằng phần mềm ghi âm trên thiết bị khác hoặc chính thiết bị phát ra âm
- Người dùng mở phần mềm và tải lên âm thanh nghi ngờ là giả mạo
- Hệ thống trả về kết quả kiểm tra: giả mạo hoặc không giả mạo

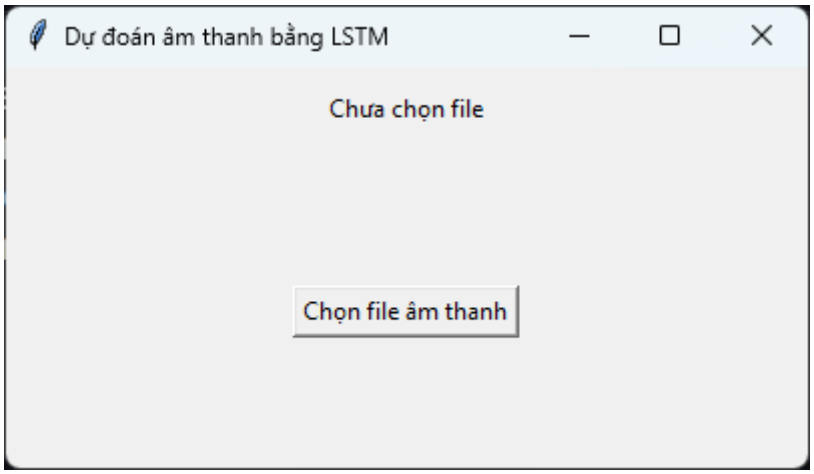
4.1.3 Chức năng chung

Căn cứ vào kịch bản sử dụng trên, ứng dụng cần có những chức năng như sau:

- Tải lên âm thanh từ định dạng âm thanh bất kỳ
- Dự đoán nhãn cho âm thanh đã tải lên và trả về kết quả

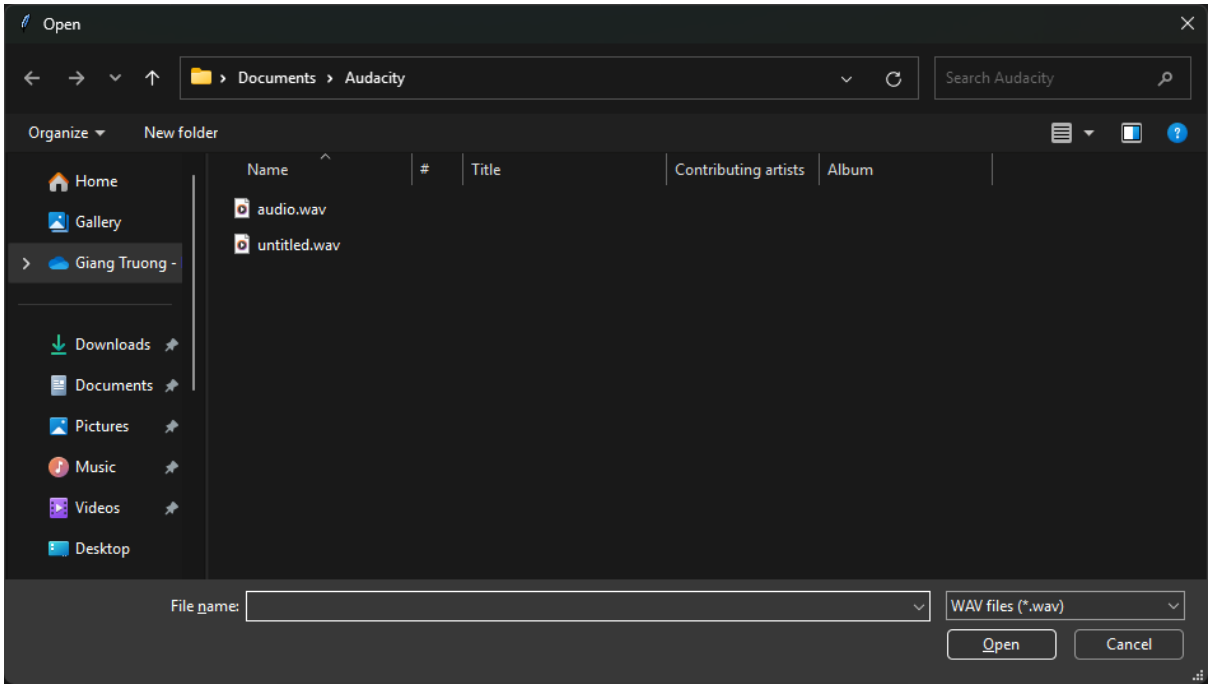
4.2 Trải nghiệm sơ bộ

Người dùng mở phần mềm, màn hình hiển thị như hình *4.1*.



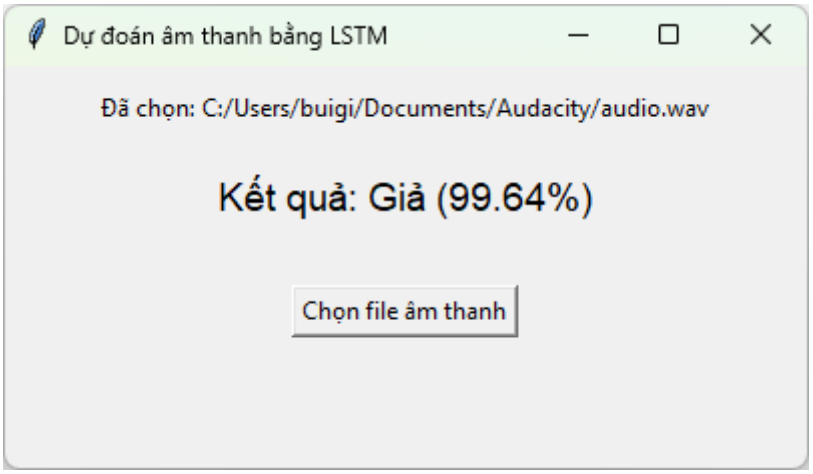
Hình 4.1 Màn hình khi khởi chạy ứng dụng

Người dùng nhấn nút “Chọn file âm thanh”, ứng dụng mở cửa sổ duyệt file cho phép người dùng chọn file âm thanh cần kiểm tra như hình *4.2*.



Hình 4.2 Cửa sổ yêu cầu người dùng chọn file âm thanh muốn kiểm tra

Sau khi tệp được tải lên, mô hình hiển thị nhãn dự báo mức độ chính xác kèm xác suất như hình *4.3*.



Hình 4.3 Kết quả trả về cho biết âm thanh là giả mạo hay là thật kèm xác suất ước lượng

4.3 Tóm tắt chương

- Chương trình Demo được thiết kế để sử dụng trong kịch bản một người nghi ngờ một âm thanh là giả mạo bởi Deepfake và muốn kiểm tra
- Chương trình bao gồm các chức năng, tải lên âm thanh giả mạo, kiểm tra, trả về kết quả gồm nhãn và xác suất tương ứng
- Chương trình hoạt động đúng như mong đợi

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Trong đồ án này, em đã xây dựng thành công một hệ thống có khả năng phát hiện giọng nói bị giả mạo tạo ra bởi công nghệ Deepfake. Hệ thống bao gồm các bước: tiền xử lý âm thanh đầu vào, trích xuất đặc trưng MFCC theo từng đoạn ngắn trong chuỗi thời gian, sau đó sử dụng mô hình học sâu LSTM để phân loại xem âm thanh đó là thật hay giả. Để đánh giá hiệu quả của mô hình LSTM, em cũng đã huấn luyện và so sánh kết quả với hai mô hình học máy phổ biến khác là Random Forest và SVM. Các kết quả thực nghiệm cho thấy mô hình LSTM có độ chính xác và độ ổn định cao hơn, đặc biệt là trong việc xử lý các mẫu âm thanh có nội dung dài và biến thiên phức tạp.

Ngoài phần nghiên cứu lý thuyết và thực nghiệm, em cũng đã xây dựng một ứng dụng demo hoạt động trên máy tính, cho phép người dùng chọn một file âm thanh bất kỳ và nhận được kết quả phân tích giọng nói: thật hay giả, kèm theo xác suất ước lượng. Đây là bước minh chứng cho tính ứng dụng thực tiễn của mô hình, đồng thời giúp người dùng dễ dàng kiểm tra nhanh các đoạn ghi âm khả nghi.

Tuy nhiên, mô hình hiện tại vẫn còn một số hạn chế. Đầu tiên, hệ thống khá nhạy với các nhiễu nền trong tệp âm thanh đầu vào. Nếu chất lượng ghi âm kém hoặc có nhiều tạp âm, mô hình có thể đưa ra kết quả sai lệch. Ngoài ra, hệ thống chưa được tối ưu cho thời gian phản hồi nhanh, nên chưa thể ứng dụng trong các tình huống yêu cầu xử lý thời gian thực như cuộc gọi điện thoại, chatbot thoại hoặc các hệ thống xác thực trực tiếp bằng giọng nói. Một điểm hạn chế khác là em mới chỉ sử dụng mô hình LSTM đơn tầng, chưa tích hợp các kiến trúc hiện đại hơn như Attention hay BiLSTM để cải thiện khả năng học ngữ cảnh dài.

Trong thời gian tới, em định hướng mở rộng nghiên cứu theo các hướng sau:

- Thứ nhất, em sẽ cải tiến kiến trúc mô hình bằng cách kết hợp LSTM với Attention hoặc thử nghiệm với các mô hình hiện đại như Transformer, nhằm tăng khả năng ghi nhớ và phân tích giọng nói dài, phức tạp, cũng như cải thiện độ chính xác trong môi trường nhiễu
- Thứ hai, em sẽ thu thập và huấn luyện mô hình với tập dữ liệu lớn hơn và phong phú hơn về giọng nói, bao gồm nhiều độ tuổi, vùng miền và cả các kỹ thuật giả mạo mới, để nâng cao khả năng tổng quát của mô hình
- Thứ ba, em sẽ triển khai lại hệ thống nhận diện giọng nói giả mạo dưới dạng ứng dụng di động trên nền tảng Android. Việc này giúp người dùng có thể kiểm tra giọng nói ngay trên điện thoại cá nhân mà không cần phần mềm chuyên dụng, từ đó tăng khả năng ứng dụng thực tế và phục vụ cộng đồng hiệu quả hơn

Em hy vọng kết quả nghiên cứu trong đồ án này sẽ là tiền đề vững chắc để phát triển các công cụ nhận diện nội dung giả mạo ngày càng tinh vi trong tương lai, góp phần bảo vệ người dùng khỏi những rủi ro lừa đảo trên không gian mạng.

TÀI LIỆU THAM KHẢO

[1] R. S. A.V. Oppenheim, "Discrete-Time Signal Processing," Pearson, 2010.

[2] Contributors, "Fourier transform," *Wikipedia, The Free Encyclopedia.*

[3] T. Y. K. Tokuda, "Speech Feature Extraction Toolkit," 2021.

[4] Jiangyan Yi, Chenglong Wang, Jianhua Tao, Chu Yuan Zhang, Cunhang Fan, Zhengkun Tian, Haoxin Ma, Ruibo Fu, "SceneFake: An Initial Dataset and Benchmarks for Scene Fake Audio Detection," arXiv, 2024.

[5] Bishop, "C. M. Pattern Recognition and Machine Learning," Springer, 2006.

[6] V. H. Tiep, "Random forest," 2021. [Online]. Available: https://machinelearningcoban.com/tabml_book/ch_model/random_forest.html. [Accessed 15 5 2025].

[7] Breiman, "Random Forests," in *Machine Learning*, Springer, 2001, p. 5–32.

[8] T. Mitchell, "Decision Tree Learning," in *Machine Learning*, 1997.

[9] F. Rosenblatt, "The Perceptron: A probabilistic model for information storage and organization in the brain," Psychological Review, 1958.

[10] MARIUS-CONSTANTIN POPESCU, VALENTINA E. BALAS, LILIANA PERESCU-POPESCU, NIKOS MASTORAKIS, "Multilayer Perceptron and Neural Networks," 2009.

[11] C. &. V. V. Cortes, "Support-vector networks," in *Machine Learning*, 1995, p. 273–297.

[12] V. H. Tiep, "Support Vector Machine," 2017. [Online]. Available: https://machinelearningcoban.com/2017/04/09/smv/. [Accessed 14 5 2025].

[13] J. Elman, "Finding structure in time," Cognitive Science, 1990.

[14] E. S. J. J. T. Kandel, "Principles of Neural Science 5th Edition," McGraw-Hill, 2013.

[15] Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall, "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning," arXiv, 2018.

[16] Hochreiter, S., & Schmidhuber, J., "Long short-term memory. Neural Computation," 1780.

[17] C. Olah, "Understanding LSTM Networks," 2015.

PHỤ LỤC

PHỤ LỤC A – Mã nguồn chương trình

A.1 Tệp data_utils.py

```
from torch.utils.data import Dataset
from torch import nn
import os
import numpy as np
import soundfile as sf
import torchaudio
import torch


class SceneFakeDataset(Dataset):
    def __init__(self, root_dir, target_length=64000):
        self.real_path = os.path.join(root_dir, 'real')
        self.fake_path = os.path.join(root_dir, 'fake')
        self.real_files = [os.path.join(self.real_path, f) for f in
os.listdir(self.real_path)]
        self.fake_files = [os.path.join(self.fake_path, f) for f in
os.listdir(self.fake_path)]
        self.all_files = self.real_files + self.fake_files
        self.labels = [0] * len(self.real_files) + [1] * len(self.fake_files)
        self.target_length = target_length
    def __len__(self):
        return len(self.all_files)
    def __getitem__(self, idx):
        filepath = self.all_files[idx]
        waveform, sr = torchaudio.load(filepath)
        waveform = waveform[0]
        if waveform.size(0) < self.target_length:
            pad = self.target_length - waveform.size(0)
            waveform = nn.functional.pad(waveform, (0, pad))
        else:
            waveform = waveform[:self.target_length]
        return waveform, self.labels[idx]


def extract_mfcc_mean(file_path, duration=4.0, sample_rate=16000, n_mfcc=40):
    waveform, sr = sf.read(file_path)
    waveform = waveform[:int(sample_rate * duration)]
    if sr != sample_rate:
        waveform = torchaudio.functional.resample(torch.tensor(waveform), sr,
sample_rate).numpy()
    waveform = torch.tensor(waveform).float().unsqueeze(0)
    mfcc_transform = torchaudio.transforms.MFCC(
```

```
        sample_rate=sample_rate,
        n_mfcc=n_mfcc,
        melkwargs={"n_fft": 512, "hop_length": 160, "n_mels": 64}
    )
    mfcc = mfcc_transform(waveform)
    mfcc_mean = mfcc.squeeze(0).mean(dim=1).numpy()
    return mfcc_mean


def load_or_extract_features(root_dir, cache_path):
    if os.path.exists(cache_path):
        print(f" Đang tải đặc trưng từ cache: {cache_path}")
        data = np.load(cache_path)
        return data['X'], data['y']
    print(f" Trích xuất đặc trưng MFCC từ {root_dir}")
    real_path = os.path.join(root_dir, 'real')
    fake_path = os.path.join(root_dir, 'fake')
    X, y = [], []
    for folder, label in [(real_path, 0), (fake_path, 1)]:
        for fname in os.listdir(folder):
            file_path = os.path.join(folder, fname)
            try:
                feat = extract_mfcc_mean(file_path)
                X.append(feat)
                y.append(label)
            except Exception as e:
                print(f" Bỏ qua file: {file_path} ({str(e)})")
    X, y = np.array(X), np.array(y)
    np.savez_compressed(cache_path, X=X, y=y)
    print(f"  Đã lưu đặc trưng tại: {cache_path}")
    return X, y
```

A.2 Tệp model_rnn.py

```
import torch
import torch.nn as nn
import torchaudio
from torch.nn.utils.rnn import pad_sequence


class RNNFakeDetectionModel(nn.Module):
    def __init__(self, n_mfcc=13, hidden_size=64, dense_size=32,
num_classes=2):
        super(RNNFakeDetectionModel, self).__init__()
        self.mfcc_transform = torchaudio.transforms.MFCC(
            sample_rate=16000,
            n_mfcc=n_mfcc,
            melkwargs={
                'n_fft': 400,
```

```
                'hop_length': 160,
                'n_mels': 40
            }
        )
        self.rnn = nn.RNN(input_size=n_mfcc, hidden_size=hidden_size,
batch_first=True)
        self.fc1 = nn.Linear(hidden_size, dense_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(dense_size, num_classes)
        self.softmax = nn.Softmax(dim=1)


    def forward(self, waveforms):
        mfcc_list = []
        for i in range(waveforms.size(0)):
            waveform = waveforms[i].unsqueeze(0)
            mfcc = self.mfcc_transform(waveform)
            mfcc = mfcc.squeeze(0).transpose(0, 1)
            mfcc_list.append(mfcc)
        mfcc_padded = pad_sequence(mfcc_list, batch_first=True)
        rnn_out, _ = self.rnn(mfcc_padded)
        last_hidden = rnn_out[:, -1, :]
        x = self.fc1(last_hidden)
        x = self.relu(x)
        x = self.fc2(x)
        return self.softmax(x)
```

A.3 Tệp model_lstm.py

```
import torch
import torch.nn as nn
import torchaudio
from torch.nn.utils.rnn import pad_sequence


class LSTMFakeDetectionModel(nn.Module):
    def __init__(self, n_mfcc=13, hidden_size=64, dense_size=32,
num_classes=2):
        super(LSTMFakeDetectionModel, self).__init__()
        self.mfcc_transform = torchaudio.transforms.MFCC(
            sample_rate=16000,
            n_mfcc=n_mfcc,
            melkwargs={
                'n_fft': 400,
                'hop_length': 160,
                'n_mels': 40
            }
        )
```

```
        self.lstm = nn.LSTM(input_size=n_mfcc, hidden_size=hidden_size,
batch_first=True)
        self.fc1 = nn.Linear(hidden_size, dense_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(dense_size, num_classes)
        self.softmax = nn.Softmax(dim=1)


    def forward(self, waveforms):
        mfcc_list = []
        for i in range(waveforms.size(0)):
            waveform = waveforms[i].unsqueeze(0)
            mfcc = self.mfcc_transform(waveform)
            mfcc = mfcc.squeeze(0).transpose(0, 1)
            mfcc_list.append(mfcc)
        mfcc_padded = pad_sequence(mfcc_list, batch_first=True)
        lstm_out, _ = self.lstm(mfcc_padded)
        last_hidden = lstm_out[:, -1, :]
        x = self.fc1(last_hidden)
        x = self.relu(x)
        x = self.fc2(x)
        return self.softmax(x)
```

A.3 Tệp model_driver.py

```
import os
import torch
import torch.nn as nn
import torchaudio
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
from torch.utils.data import DataLoader
from sklearn.metrics import classification_report
import joblib
import numpy as np
from model_lstm import LSTMFakeDetectionModel
from model_rnn import RNNFakeDetectionModel
from data_utils import SceneFakeDataset
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from data_utils import load_or_extract_features


class ModelDriver:
    def __init__(self, model_path):
        self.model_path = model_path
        self.device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
```

```
    def train(self):
        raise NotImplementedError
    def evaluate(self):
        raise NotImplementedError


class LSTMDriver(ModelDriver):
    def __init__(self, model_path="models/model_lstm.pth"):
        super().__init__(model_path)
        self.model = LSTMFakeDetectionModel().to(self.device)
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=1e-3)
        self.criterion = nn.CrossEntropyLoss()
        self.train_loader = DataLoader(SceneFakeDataset("ScenceFake/train"),
batch_size=1024, shuffle=True)
        self.eval_loader = DataLoader(SceneFakeDataset("ScenceFake/eval"),
batch_size=1024)
    def _load_checkpoint(self):
        if os.path.exists(self.model_path):
            checkpoint = torch.load(self.model_path, map_location=self.device)
            self.model.load_state_dict(checkpoint["model_state"])
            self.optimizer.load_state_dict(checkpoint["optimizer_state"])
            return checkpoint["epoch"] + 1
        return 0
    def train(self, num_epochs=10):
        start_epoch = self._load_checkpoint()
        for epoch in range(start_epoch, start_epoch + num_epochs):
            self.model.train()
            total_loss = 0
            progress = tqdm(self.train_loader, desc=f"Epoch {epoch + 1}")
            for waveforms, labels in progress:
                waveforms, labels = waveforms.to(self.device),
labels.to(self.device)
                outputs = self.model(waveforms)
                loss = self.criterion(outputs, labels)
                self.optimizer.zero_grad()
                loss.backward()
                self.optimizer.step()
                total_loss += loss.item()
                avg_loss = total_loss / (progress.n + 1)
                progress.set_postfix(loss=f"{avg_loss:.4f}")
            torch.save({
                "epoch": epoch,
                "model_state": self.model.state_dict(),
                "optimizer_state": self.optimizer.state_dict()
            }, self.model_path)


    def evaluate(self):
        self._load_checkpoint()
```

```
        self.model.eval()
        all_preds, all_labels = [], []
        with torch.no_grad():
            for waveforms, labels in tqdm(self.eval_loader,
desc="Evaluating"):
                waveforms = waveforms.to(self.device)
                outputs = self.model(waveforms)
                preds = torch.argmax(outputs, dim=1)
                all_preds.extend(preds.cpu().numpy())
                all_labels.extend(labels.numpy())
        print(classification_report(all_labels, all_preds, digits=4))


class RNNDriver(LSTMDriver):
    def __init__(self, model_path="models/model_rnn.pth"):
        ModelDriver.__init__(self, model_path)
        self.model = RNNFakeDetectionModel().to(self.device)
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=1e-3)
        self.criterion = nn.CrossEntropyLoss()
        self.train_loader = DataLoader(SceneFakeDataset("ScenceFake/train"),
batch_size=1024, shuffle=True)
        self.eval_loader = DataLoader(SceneFakeDataset("ScenceFake/eval"),
batch_size=1024)


    def _load_checkpoint(self):
        if os.path.exists(self.model_path):
            checkpoint = torch.load(self.model_path, map_location=self.device)
            self.model.load_state_dict(checkpoint["model_state"])
            self.optimizer.load_state_dict(checkpoint["optimizer_state"])
            return checkpoint["epoch"] + 1
        return 0


class SVMDriver(ModelDriver):
    def __init__(self, model_path="models/model_svm.pkl"):
        super().__init__(model_path)
        self.X_train, self.y_train =
load_or_extract_features("ScenceFake/train", "features/features_train.npz")
        self.X_eval, self.y_eval = load_or_extract_features("ScenceFake/eval",
"features/features_eval.npz")
        scaler = StandardScaler()
        self.X_train = scaler.fit_transform(self.X_train)
        self.X_eval = scaler.transform(self.X_eval)
        self.model = None


    def _load_model(self):
        if os.path.exists(self.model_path):
```

```
            self.model = joblib.load(self.model_path)


    def train(self):
        self._load_model()
        print("Tuning SVM vi GridSearchCV...")
        param_grid = {
            "C": [0.1, 1, 10],
            "kernel": ["rbf", "linear"],
            "gamma": ["scale", "auto"]
        }
        grid = GridSearchCV(SVC(), param_grid, cv=5, n_jobs=-1, verbose=1)
        grid.fit(self.X_train, self.y_train)
        print("Best params:", grid.best_params_)
        self.model = grid.best_estimator_
        joblib.dump(self.model, self.model_path)
    def evaluate(self):
        self._load_model()
        y_pred = self.model.predict(self.X_eval)
        print(classification_report(self.y_eval, y_pred, digits=4))


class RandomForestDriver(ModelDriver):
    def __init__(self, model_path="models/model_rf.pkl"):
        super().__init__(model_path)
        self.X_train, self.y_train =
load_or_extract_features("ScenceFake/train", "features/features_train.npz")
        self.X_eval, self.y_eval = load_or_extract_features("ScenceFake/eval",
"features/features_eval.npz")
        scaler = StandardScaler()
        self.X_train = scaler.fit_transform(self.X_train)
        self.X_eval = scaler.transform(self.X_eval)
        self.model = None


    def _load_model(self):
        if os.path.exists(self.model_path):
            self.model = joblib.load(self.model_path)


    def train(self):
        self._load_model()
        print("Tuning Random Forest vi GridSearchCV...")
        param_grid = {
            "n_estimators": [100, 200, 500],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1, 2]
        }
```

```
        grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=5,
n_jobs=-1, verbose=1)
        grid.fit(self.X_train, self.y_train)
        print("Best params:", grid.best_params_)
        self.model = grid.best_estimator_
        joblib.dump(self.model, self.model_path)


    def evaluate(self):
        self._load_model()
        y_pred = self.model.predict(self.X_eval)
        print(classification_report(self.y_eval, y_pred, digits=4))
```

A.4 Tệp main.py

```
from model_driver import LSTMDriver, RNNDriver, SVMDriver, RandomForestDriver


def main():
    model = "lstm"
    action = "eval"
    drivers = {
        "lstm": LSTMDriver,
        "rnn": RNNDriver,
        "svm": SVMDriver,
        "rf": RandomForestDriver,
    }
    if model not in drivers:
        raise ValueError(f" Model không hợp lệ: {model}")
    if action not in ["train", "eval"]:
        raise ValueError(f" Action không hợp lệ: {action}")
    driver = drivers[model]()
    if action == "train":
        driver.train()
        driver.evaluate()
    else:
        driver.evaluate()


if __name__ == "__main__":
    main()
```

A.5 Tệp demo.py

```
import torch
import torchaudio
import tkinter as tk
from tkinter import filedialog, messagebox
from model_driver import LSTMDriver
driver = LSTMDriver()
```

```
driver._load_checkpoint()


def predict_label_with_confidence(file_path):
    waveform, sr = torchaudio.load(file_path)
    waveform = waveform[0]
    target_length = 64000
    if waveform.size(0) < target_length:
        pad = target_length - waveform.size(0)
        waveform = torch.nn.functional.pad(waveform, (0, pad))
    else:
        waveform = waveform[:target_length]
    waveform = waveform.unsqueeze(0).to(driver.device)
    driver.model.eval()
    with torch.no_grad():
        output = driver.model(waveform)
        probabilities = output.squeeze().cpu().numpy()
        label = "Thật" if probabilities[0] > probabilities[1] else "Giả"
        confidence = max(probabilities)
        return label, confidence


def run_gui():
    window = tk.Tk()
    window.title("Deepfake Detector")
    window.geometry("400x200")
    file_label = tk.Label(window, text="Chưa chọn file", wraplength=380)
    file_label.pack(pady=10)
    result_label = tk.Label(window, text="", font=("Helvetica", 14))
    result_label.pack(pady=10)


    def choose_file():
        file_path = filedialog.askopenfilename(
            filetypes=[("WAV files", "*.wav"), ("All files", "*.*")]
        )
        if file_path:
            file_label.config(text=f"Đã chọn: {file_path}")
            try:
                label, confidence = predict_label_with_confidence(file_path)
                result_label.config(
                    text=f"Kết quả: {label} ({confidence*100:.2f}%)"
                )
            except Exception as e:
                messagebox.showerror("Lỗi", f"Không thể dự đoán: {str(e)}")
    choose_button = tk.Button(window, text="Chọn file âm thanh",
command=choose_file)
    choose_button.pack(pady=20)
    window.mainloop()
```

```
if __name__ == "__main__":
    run_gui()
```