ĐẶNG ĐỨC MẠNH

KỸ THUẬT PHẦN MỀM

BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

NGÀNH KỸ THUẬT PHẦN MỀM

NGHIÊN CỨU MỘT SỐ MÔ HÌNH HỌC SÂU ÁP DỤNG XÂY DỰNG TÍNH NĂNG GHI BIÊN BẢN CUỘC HỌP

GVHD: **ThS. Phạm Thị Kim Phượng**

Sinh viên: **Đặng Đức Mạnh**

Mã sinh viên: **2020604935**

Hà Nội – Năm 2024

LỜI CẢM ƠN

Lời đầu tiên, em xin chân thành cảm ơn quý thầy cô trong **Khoa Công Nghệ Thông Tin** đã luôn đồng hành và hỗ trợ em suốt chặng đường học tập và nghiên cứu tại trường. Đặc biệt, em muốn gửi lời tri ân đặc biệt đến cô **ThS. Phạm Thị Kim Phượng**, người đã dành thời gian và tâm huyết để hướng dẫn, giúp đỡ và chia sẻ kiến thức quý báu trong suốt thời gian thực hiện đồ án tốt nghiệp. Em không quên bày tỏ lòng biết ơn sâu sắc đến tất cả các thầy cô ở **Trường Đại học Công nghiệp Hà Nội**, những người đã chia sẻ tình cảm, kinh nghiệm và kỹ thuật giáo dục quý báu. Điều này đã giúp em hiểu rõ hơn về cách xây dựng và phát triển đề tài "**Nghiên cứu một số mô hình học sâu, áp dụng xây dựng tính năng ghi biên bản cuộc họp** "của mình. Mặc dù thời gian hạn chế và những thách thức không tránh khỏi, em đã cố gắng hết sức để áp dụng những kiến thức vào đề tài của mình. Tuy nhiên, em nhận thức rằng còn nhiều điều để hoàn thiện và phát triển trong tương lai. Do đó, em mong nhận được sự đóng góp, góp ý và hỗ trợ chân thành từ thầy cô cùng các bạn. Những đóng góp này sẽ là nguồn động viên quý báu giúp em vượt qua những hạn chế và phát triển hơn trong sự nghiệp nghiên cứu của mình.

Lời cuối cùng em xin kính chúc thầy cô và các bạn sức khỏe và gặt hái thêm nhiều thành công trong sự nghiệp.

Em xin chân thành cảm ơn!

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

GIỚI THIỆU

Trong thời đại công nghệ số ngày nay, việc truy cập và sử dụng các công cụ chuyển văn bản thành giọng nói trực tuyến đã trở thành một xu hướng phổ biến. Với sự phát triển nhanh chóng của Internet và các nền tảng dịch vụ trực tuyến, việc xây dựng một trang web chuyển văn bản thành giọng nói chất lượng cao là một thách thức đáng chú ý. Đồng thời, sự tiện lợi và tương tác của giao diện người dùng (UI) đóng vai trò quan trọng trong trải nghiệm người dùng.

Đồ án tốt nghiệp này nhằm mục đích nghiên cứu và xây dựng một số mô hình web chuyển văn bản thành giọng nói. Các mô hình này sẽ được phát triển dựa trên các công nghệ hiện đại để mang lại chất lượng giọng nói tự nhiên và chính xác. Chủ đề này không chỉ giúp chúng em áp dụng kiến thức đã học một cách thực tế mà còn mang lại cơ hội để tìm hiểu sâu hơn về các công nghệ tiên tiến trong lĩnh vực chuyển văn bản thành giọng nói.

Chúng em hy vọng rằng dự án này sẽ đóng góp vào việc nâng cao hiểu biết và kỹ năng của chúng em trong việc xây dựng các ứng dụng web phức tạp và tương tác. Báo cáo này sẽ trình bày chi tiết về quá trình thiết kế, triển khai và kiểm thử các mô hình web chuyển văn bản thành giọng nói. Nội dung báo cáo sẽ bao gồm giới thiệu về các công nghệ liên quan, thiết kế giao diện người dùng, thiết kế cơ sở dữ liệu, xây dựng phía máy chủ và phía người dùng, kiểm thử và triển khai ứng dụng.

Chúng em hy vọng rằng báo cáo này sẽ cung cấp một cái nhìn tổng quan về quá trình xây dựng một trang web chuyển văn bản thành giọng nói, đồng thời chia sẻ kinh nghiệm và kỹ năng mà chúng em đã tích lũy được trong quá trình thực hiện dự án này.

Xin chân thành cảm ơn!

MỞ ĐẦU

1. Đặt vấn đề

Mục tiêu cốt lõi của việc nhận dạng tiếng nói là giúp máy tính hiểu được ngôn ngữ đa dạng của con người. Thuật toán nhận dạng tiếng nói được thiết kế để phân tích và trích xuất thông tin hữu ích từ bản ghi âm tiếng nói. Thông tin này sau đó được chuyển đổi thành định dạng văn bản để được xử lý bởi máy tính.

Quá trình nhận dạng tiếng nói gồm hai bước chính: mô hình hóa âm thanh và mô hình ngôn ngữ. Mô hình hóa âm thanh liên quan đến việc phân tích sóng âm của tiếng nói và chuyển đổi chúng thành một định dạng máy tính có thể hiểu. Điều này bao gồm việc chia tách sóng âm thành các âm vị nhỏ hơn.

Mô hình ngôn ngữ liên quan đến việc phân tích các từ và cụm từ trong lời nói và tạo ra một cấu trúc ngữ pháp từ đó. Mục tiêu là xác định chuỗi từ có thể nhất quán nhất mà người dùng đã nói, dựa trên mô hình âm.

Một trong những thách thức chính của nhận dạng tiếng nói là xử lý sự khác biệt trong cách phát âm, giọng điệu và mô hình nói. Có nhiều yếu tố có thể ảnh hưởng đến độ chính xác của thuật toán nhận dạng tiếng nói, như tiếng ồn xung quanh, chất lượng microphone và ngôn ngữ được sử dụng.

Mặc dù gặp phải những thách thức đó, nhận dạng tiếng nói mang lại nhiều lợi ích tiềm năng. Nó có thể cải thiện tính sẵn có cho những người khuyết tật như người điếc hoặc khó nghe. Nó cũng có thể được sử dụng để tự động hóa yêu cầu dịch vụ khách hàng, giảm sự phụ thuộc vào nhân viên hỗ trợ. Hơn nữa, các trợ lý cá nhân như Siri và Alexa dựa vào nhận dạng tiếng nói để hiểu yêu cầu của người dùng và đưa ra phản hồi phù hợp. Nhận dạng tiếng nói là một công nghệ đang phát triển nhanh chóng và được sử dụng phổ biến để giúp chúng ta tương tác dễ dàng hơn với máy tính. Khi các thuật toán nhận dạng tiếng nói tiếp tục được cải tiến, chúng ta có thể mong đợi sự ứng dụng của công nghệ này trong cuộc sống hàng ngày của chúng ta.

2. Lý do chọn đề tài

Nghiên cứu và áp dụng nhận dạng giọng nói mang lại nhiều lợi ích cho người dùng và xã hội. Dưới đây là một số lý do quan trọng để nghiên cứu nhận dạng giọng nói:

Tăng cường khả năng tiếp cận cho những người khuyết tật: Công nghệ nhận dạng giọng nói giúp những người mù hoặc không thể sử dụng tay có khả năng tiếp cận thiết bị điện tử và ứng dụng máy tính thông qua giọng nói của họ.

Tăng năng suất và tiện lợi: Sử dụng giọng nói để điều khiển thiết bị và ứng dụng trong cuộc sống hàng ngày giúp tiết kiệm thời gian và mang lại sự tiện lợi. Ví dụ, các trợ lý ảo như Siri và Google Assistant cho phép người dùng thực hiện các tác vụ như gửi tin nhắn hay tìm kiếm thông tin chỉ bằng giọng nói.

Cải thiện trải nghiệm khách hàng: Trong lĩnh vực điện thoại di động, trung tâm cuộc gọi và dịch vụ khách hàng, nhận dạng giọng nói có thể được sử dụng để tự động hóa nhiều quy trình khác nhau, nâng cao trải nghiệm khách hàng và giảm thời gian chờ đợi.

Tăng cường an ninh và giám sát: Công nghệ nhận dạng giọng nói có thể áp dụng trong các lĩnh vực an ninh và giám sát, như sân bay và trung tâm mua sắm, để nhận dạng và xác minh danh tính người dùng.

Đóng góp vào sự phát triển của trí tuệ nhân tạo và học máy: Nghiên cứu và phát triển các thuật toán nhận dạng giọng nói đóng góp vào sự phát triển của trí tuệ nhân tạo và học máy, tạo ra những ứng dụng tiên tiến và đổi mới.

Tổng kết lại, nhận dạng giọng nói là một lĩnh vực đáng nghiên cứu và ứng dụng, mang lại nhiều lợi ích quan trọng cho cá nhân và xã hội.

3. Mục tiêu đề tài

3.1 Về lý thuyết

Tìm hiểu lịch sử hình thành và phát triển của công nghệ nhận dạng giọng nói.

Hiểu được kiến trúc và hoạt động cơ bản của nền tảng Deep learning. Khảo sát và phân tích ưu nhược điểm một số nền tảng điển hình của ASR hiện nay trong thực tiễn để chọn lựa nền tảng xây dựng ứng dụng ghi biên bản họp.

3.2 Về ứng dụng

Xây dựng và triển khai ứng dụng dễ dàng sử dụng và đáp ứng nhu cầu của người dùng, cho phép họ giao tiếp với các thiết bị và hệ thống khác nhau bằng giọng nói một cách dễ dàng và hiệu quả.

4. Phương pháp nghiên cứu

Đề tài này sử dụng phương pháp nghiên cứu chủ yếu là phương pháp phân tích và tổng kết kinh nghiệm.

Phương pháp phân tích và tổng kết kinh nghiệm là phương pháp nghiên cứu xem xét lại những thành quả của hoạt động thực tiễn trong quá khứ để rút ra những kết luận bổ ích cho thực tiễn và cho khoa học. Tổng kết kinh nghiệm thường thường hướng vào nghiên cứu diễn biến và nguyên nhân của các sự kiện và nghiên cứu giải pháp thực tiễn đã áp dụng để tìm ra các giải pháp hoàn hảo nhất

Chính vì vậy mà phương pháp này thường được sử dụng cho các mục đích sau:

- Tìm hiểu bản chất, nguồn gốc, nguyên nhân và cách giải quyết các vấn đề trong cuộc sống.

- Nghiên cứu con đường thực hiện có hiệu quả cách giải quyết trên.

- Tổng kết các sáng kiến của các những người đi trước.

- Tổng kết những nguyên nhân, để loại trừ những sai lầm, thất bại trong hoạt động.

- Tổng kết kinh nghiệm mang tính quần chúng rộng rãi. Cụ thể, từ việc khảo sát và phân tích nhu cầu về ứng dụng cũng như các công cụ hiện có, đề tài đề xuất mô hình hệ thống và xây dựng, phát triển ứng dụng.

5. Cấu trúc báo cáo

Chương I: Cơ sở lý thuyết: Tổng hợp và trình bày các kiến thức lý thuyết liên quan đến đề tài, bao gồm các khái niệm, nguyên lý và công nghệ có liên quan.

Chương II: Phát biểu bài toán: Tiến hành khảo sát về hệ thống tương tự hoặc liên quan đến đề tài, tìm hiểu về các công nghệ, phương pháp và giải pháp đã được áp dụng.

Chương III: Xây dựng hệ thống trang web với Flask: Trình bày quá trình xây dựng phần máy chủ của hệ thống sử dụng Flask, bao gồm cài đặt môi trường, triển khai các chức năng chính và tương tác với cơ sở dữ liệu.

Chương IV: Kết luận: Tập trung vào việc tóm tắt những kết quả đạt được từ quá trình nghiên cứu và xây dựng hệ thống. Đưa ra một tổng quan chung về những thành tựu và đóng góp của đề tài, cũng như đánh giá hiệu quả và khả năng áp dụng của hệ thống.

CHƯƠNG I: CƠ SỞ LÝ THUYẾT

1.1. Học sâu

1.1.1. Giới thiệu

Học sâu (Deep Learning) là một lĩnh vực con của học máy (Machine Learning) tập trung vào việc sử dụng các mạng nơ-ron nhân tạo nhiều lớp để mô phỏng cách mà bộ não con người xử lý dữ liệu và đưa ra quyết định. Các mô hình học sâu thường bao gồm nhiều tầng lớp (layer) nơ-ron, cho phép chúng học và trích xuất các đặc trưng phức tạp từ dữ liệu thô. Điều này làm cho học sâu đặc biệt hữu ích trong việc xử lý các loại dữ liệu phức tạp và đa dạng như hình ảnh, âm thanh, và ngôn ngữ tự nhiên. Các ứng dụng của học sâu rất rộng rãi, từ nhận diện hình ảnh, nhận dạng giọng nói, dịch máy, đến các hệ thống đề xuất và trò chơi.

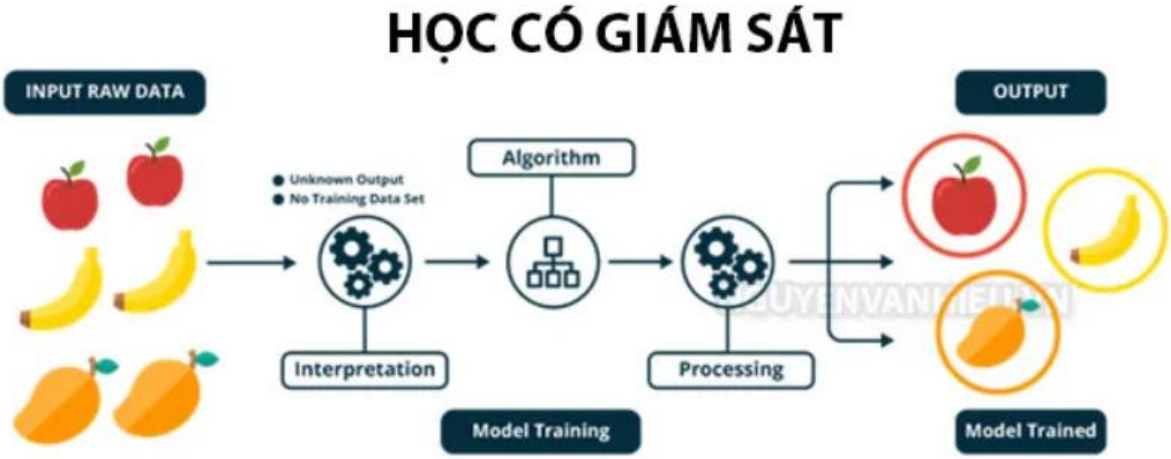
Một trong những ưu điểm nổi bật của học sâu là khả năng tự động học các đặc trưng từ dữ liệu mà không cần phải can thiệp quá nhiều từ con người. Điều này được thực hiện thông qua quá trình huấn luyện mạng nơ-ron trên một lượng lớn dữ liệu, sử dụng các thuật toán tối ưu hóa để điều chỉnh trọng số của các kết nối giữa các nơ-ron. Các mô hình học sâu phổ biến bao gồm Mạng nơ-ron tích chập (Convolutional Neural Networks - CNNs) cho xử lý hình ảnh, Mạng nơ-ron hồi quy (Recurrent Neural Networks - RNNs) cho xử lý chuỗi dữ liệu như âm thanh và văn bản, và các biến thể như Long Short-Term Memory (LSTM) và Gated Recurrent Units (GRUs) để giải quyết vấn đề liên quan đến bộ nhớ ngắn hạn. Học sâu đã và đang tiếp tục cách mạng hóa nhiều lĩnh vực, mang lại những tiến bộ vượt bậc và mở ra nhiều cơ hội mới trong nghiên cứu và ứng dụng thực tiễn.

1.1.2. Học có giám sát

Học có giám sát (Supervised Learning) là một phương pháp trong lĩnh vực học máy (Machine Learning), nơi mà mô hình được huấn luyện trên một

bộ dữ liệu đã được gán nhãn. Mỗi mẫu dữ liệu trong bộ huấn luyện đi kèm với một nhãn đầu ra chính xác, cho phép mô hình học cách ánh xạ từ đầu vào đến đầu ra dựa trên các ví dụ này. Mục tiêu của học có giám sát là xây dựng một hàm ánh xạ hoặc mô hình có thể dự đoán chính xác nhãn cho các dữ liệu mới chưa từng thấy. Các thuật toán học có giám sát phổ biến bao gồm hồi quy tuyến tính (Linear Regression), hồi quy logistic (Logistic Regression), cây quyết định (Decision Trees), và các mạng nơ-ron nhân tạo (Artificial Neural Networks).[10]

Trong quá trình huấn luyện, mô hình học có giám sát sẽ điều chỉnh các tham số của mình dựa trên các mẫu dữ liệu đã gán nhãn để tối ưu hóa hiệu suất dự đoán. Điều này thường được thực hiện bằng cách giảm thiểu một hàm mất mát (loss function) đo lường sự khác biệt giữa dự đoán của mô hình và nhãn thực tế. Sau khi được huấn luyện, mô hình có thể được đánh giá bằng cách sử dụng một bộ dữ liệu kiểm tra riêng biệt để kiểm tra khả năng tổng quát hóa của nó đối với các dữ liệu mới. Học có giám sát được ứng dụng rộng rãi trong nhiều lĩnh vực như nhận dạng hình ảnh, phân loại văn bản, dự đoán bệnh, và phân tích tài chính. Với khả năng học từ các ví dụ cụ thể, các mô hình học có giám sát đã trở thành công cụ quan trọng và hiệu quả trong việc giải quyết các bài toán phức tạp và đa dạng trong thế giới thực.



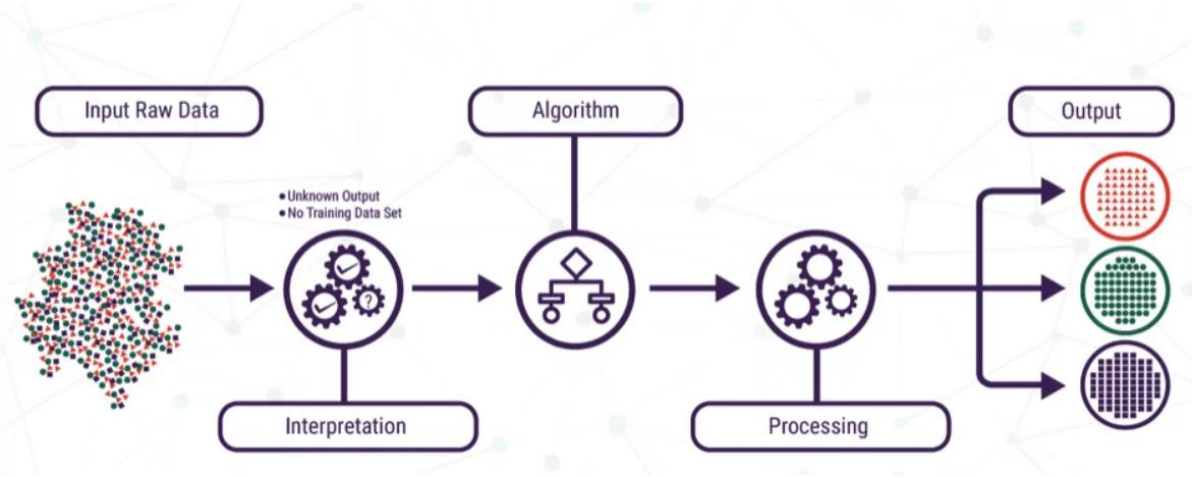
Hình 1.1. Học có giám sát

1.1.3. Học không giám sát

Học không giám sát (Unsupervised Learning) là một nhánh của học máy (Machine Learning) tập trung vào việc tìm kiếm các cấu trúc ẩn trong dữ liệu mà không cần đến các nhãn hoặc đầu ra đã được xác định trước. Mục tiêu chính của học không giám sát là khám phá các mẫu, nhóm, hoặc mối quan hệ trong dữ liệu mà không có sự giám sát từ bên ngoài. Điều này thường được thực hiện thông qua các thuật toán như phân cụm (clustering), giảm chiều dữ liệu (dimensionality reduction), và các phương pháp học không giám sát khác. Phân cụm K-means (K-means Clustering) và Phân tích thành phần chính (Principal Component Analysis - PCA) là hai ví dụ điển hình của các kỹ thuật học không giám sát phổ biến. [2]

Trong học không giám sát, các thuật toán phân cụm như K-means tìm kiếm các nhóm dữ liệu có đặc điểm tương đồng, giúp nhận diện các cụm hoặc phân đoạn trong dữ liệu mà chưa được gán nhãn trước. Các kỹ thuật giảm chiều dữ liệu

như PCA hoặc T-SNE giúp giảm số lượng biến số trong dữ liệu, làm cho việc phân tích và trực quan hóa dữ liệu trở nên dễ dàng hơn mà vẫn giữ lại được các thông tin quan trọng. Học không giám sát được ứng dụng rộng rãi trong các lĩnh vực như phân tích khách hàng, phát hiện gian lận, và khám phá tri thức từ dữ liệu lớn. Bằng cách cho phép khám phá dữ liệu một cách tự động mà không cần các nhãn trước, học không giám sát cung cấp những công cụ mạnh mẽ để khai thác thông tin tiềm ẩn và hiểu rõ hơn về cấu trúc dữ liệu.

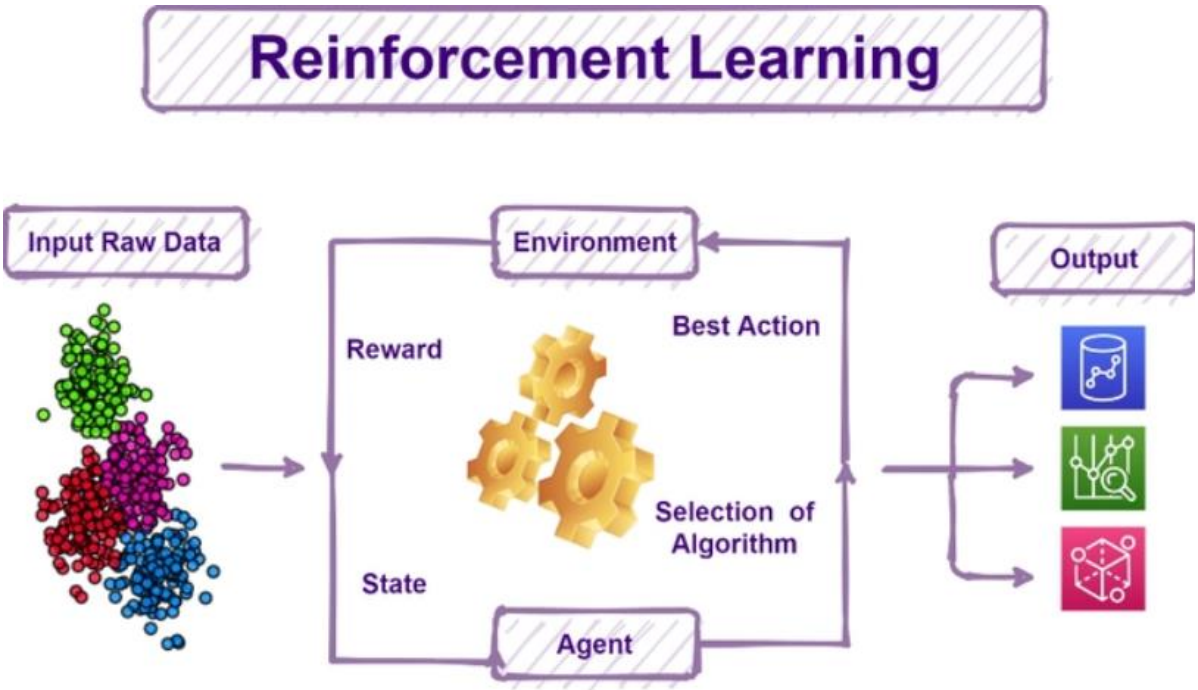


Hình 1.2. Học không giám sát

1.1.3. Học tăng cường

Học tăng cường (Reinforcement Learning - RL) là một lĩnh vực của học máy (Machine Learning) tập trung vào việc đào tạo các tác nhân (agents) để đưa ra quyết định thông qua tương tác với môi trường. Trong học tăng cường, tác nhân học cách tối ưu hóa một hàm phần thưởng bằng cách thực hiện các hành động và nhận phản hồi dưới dạng phần thưởng hoặc hình phạt từ môi trường. Mục tiêu là tìm ra chiến lược (policy) tốt nhất để tối đa hóa phần thưởng tích lũy theo thời gian. Các phương pháp học tăng cường phổ biến bao gồm Q-learning, Deep Q-Networks (DQN), và các thuật toán chính sách (policy-based) như Proximal Policy Optimization (PPO). Học tăng cường đã được áp dụng thành

công trong nhiều lĩnh vực như chơi game (ví dụ: AlphaGo của Google DeepMind), robot tự hành, quản lý tài nguyên và hệ thống đề xuất, mang lại những kết quả ấn tượng.



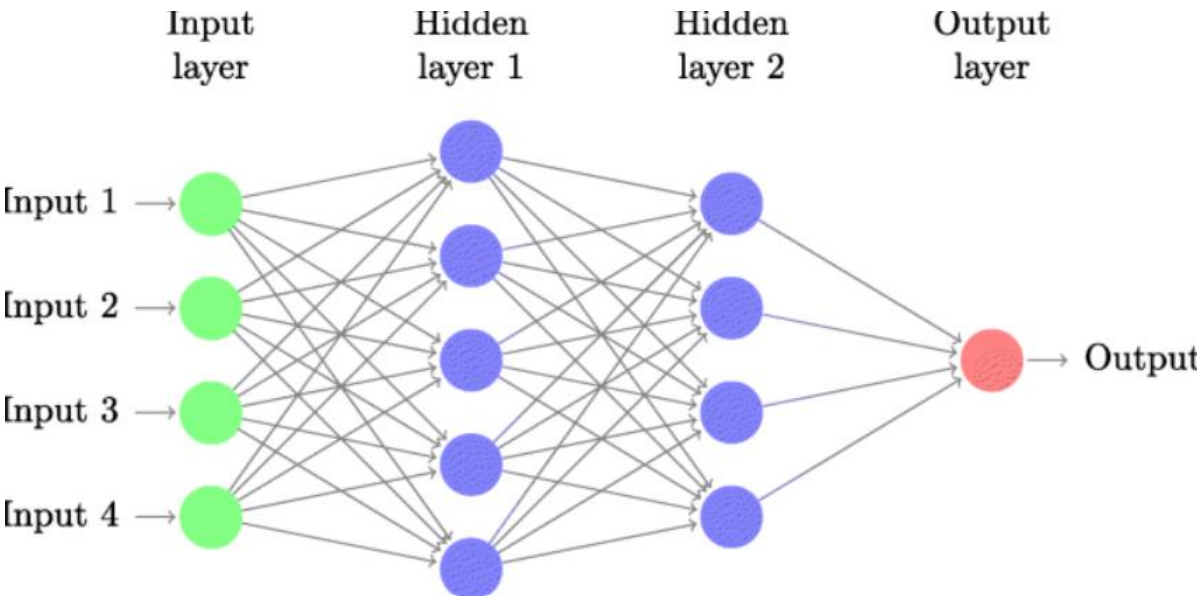
Hình 1.3. Học tăng cường

1.2. Mạng neuron nhân tạo

1.2.1. Giới thiệu

Mạng nơ-ron nhân tạo (Artificial Neural Networks - ANNs) là một mô hình toán học lấy cảm hứng từ cấu trúc và chức năng của bộ não con người, được sử dụng để nhận dạng mẫu và học từ dữ liệu. Một mạng nơ-ron bao gồm nhiều đơn vị tính toán nhỏ gọi là nơ-ron (neurons), được tổ chức thành các lớp (layers): lớp đầu vào (input layer), lớp ẩn (hidden layer), và lớp đầu ra (output layer). Mỗi nơ-ron kết nối với nơ-ron ở các lớp khác thông qua các liên kết có trọng số (weights) mà được điều chỉnh trong quá trình huấn luyện để giảm thiểu sai số dự đoán. Mạng nơ-ron nhân tạo có khả năng học hỏi và tổng quát hóa từ

dữ liệu lớn, làm cho chúng đặc biệt hiệu quả trong các nhiệm vụ như phân loại hình ảnh, nhận dạng giọng nói, và dự đoán chuỗi thời gian. Với sự phát triển của học sâu (deep learning), các mô hình mạng nơ-ron sâu (deep neural networks) đã trở thành công cụ mạnh mẽ trong nhiều ứng dụng trí tuệ nhân tạo hiện đại. [3]



Hình 1.4. Mạng neuron nhân tạo

1.2.2. Mạng CNN

Mạng nơ-ron tích chập (Convolutional Neural Networks - CNNs) là một loại mạng nơ-ron nhân tạo đặc biệt, được thiết kế để xử lý và phân tích dữ liệu dạng lưới như hình ảnh và video. CNNs sử dụng các lớp tích chập (convolutional layers) để tự động trích xuất các đặc trưng từ dữ liệu đầu vào. Mỗi lớp tích chập bao gồm một tập hợp các bộ lọc (filters) hoặc nhân tích chập (kernels), trượt qua dữ liệu đầu vào để tạo ra các bản đồ đặc trưng (feature maps). Quá trình này giúp CNNs nhận diện các đặc trưng cục bộ trong dữ liệu như cạnh, góc, và kết cấu, từ đó xây dựng các đặc trưng phức tạp hơn qua nhiều lớp.

Một điểm mạnh của CNNs là khả năng giảm thiểu số lượng tham số cần thiết, so với các mạng nơ-ron truyền thống. Thay vì kết nối tất cả các nơ-ron của một lớp với tất cả các nơ-ron của lớp tiếp theo, CNNs chỉ kết nối các nơ-ron trong một vùng cục bộ của lớp trước với các nơ-ron của lớp sau. Điều này không chỉ giúp giảm số lượng tham số mà còn giúp CNNs học các đặc trưng không gian cục bộ, làm cho chúng đặc biệt hiệu quả trong các nhiệm vụ liên quan đến xử lý hình ảnh. Sau các lớp tích chập, thường có các lớp gộp (pooling layers) để giảm kích thước không gian của các bản đồ đặc trưng và giảm thiểu overfitting.

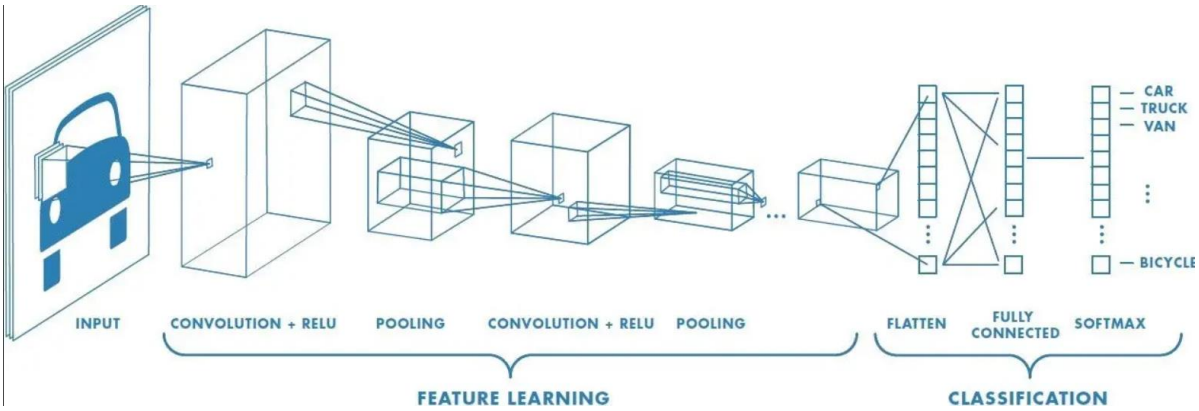
Một công thức tiêu biểu trong CNNs là phép tính tích chập. Giả sử (f) là một hàm đầu vào và (h) là một nhân tích chập, phép tính tích chập ((f * h)) tại vị trí ((i,j)) được định nghĩa như sau:

$$(f * h)(i, j) = \sum_{m} \sum_{n} f(i + m, j + n) \cdot h(m, n)$$

Trong đó, (f(i+m, j+n)) là giá trị của hàm đầu vào tại vị trí ((i+m, j+n)) và (h(m, n)) là giá trị của nhân tích chập tại vị trí ((m, n)). Kết quả của phép tích chập là một bản đồ đặc trưng mới, biểu diễn các đặc trưng đã được trích xuất từ dữ liệu đầu vào. CNNs tiếp tục sử dụng các bản đồ đặc trưng này qua nhiều lớp để xây dựng các biểu diễn trừu tượng và phân loại dữ liệu một cách hiệu quả.[8]

Nhờ vào cấu trúc đặc biệt và khả năng tự động trích xuất đặc trưng, CNNs đã trở thành công cụ mạnh mẽ trong nhiều ứng dụng thị giác máy tính như nhận diện đối tượng, phân loại hình ảnh, và phát hiện biên trong ảnh. Các mô hình CNN nổi tiếng như LeNet, AlexNet, VGG, và ResNet đã đạt được nhiều thành tựu vượt trội và được áp dụng rộng rãi trong các lĩnh vực khác nhau, từ xe tự hành đến y học và giải trí. Sự phát triển của CNNs tiếp tục mở ra nhiều tiềm

năng.



Hình 1.5. Mạng CNNs

1.2.3. Mạng RNN

Mạng nơ-ron hồi quy (Recurrent Neural Networks - RNNs) là một loại mạng nơ-ron nhân tạo được thiết kế để xử lý và mô hình hóa các chuỗi dữ liệu có tính tuần tự, như văn bản, âm thanh, hoặc dữ liệu thời gian. RNNs có khả năng lưu trữ thông tin từ quá khứ và sử dụng thông tin này để đưa ra dự đoán hoặc quyết định cho các dữ liệu hiện tại. Điều này làm cho chúng rất phù hợp để xử lý các nhiệm vụ yêu cầu hiểu và dự đoán các phụ thuộc thời gian hoặc chuỗi.[4]

Cấu trúc chính của một RNN bao gồm các đơn vị nơ-ron (neurons) kết nối với nhau theo một chuỗi thời gian. Mỗi đơn vị nơ-ron không chỉ nhận đầu vào từ lớp hiện tại mà còn nhận thông tin từ lớp trước đó trong chuỗi, tạo thành một vòng lặp cho phép truyền ngược các thông tin qua các thời điểm. Điều này giúp RNNs xử lý các chuỗi dữ liệu có độ dài khác nhau và học các mẫu phức tạp trong dữ liệu tuần tự.

Một điểm yếu của RNNs là vấn đề biến mất/xoá (vanishing/exploding gradient), khi thông tin quá lâu trong quá trình lan truyền ngược có thể dẫn đến mất mát thông tin quan trọng. Để giải quyết vấn đề này, các biến thể như Long Short-Term Memory (LSTM) và Gated Recurrent Unit (GRU) đã được phát

triển, cho phép RNNs lưu trữ thông tin quan trọng trong thời gian dài mà vẫn tránh được vấn đề biến mất/xoá gradient.

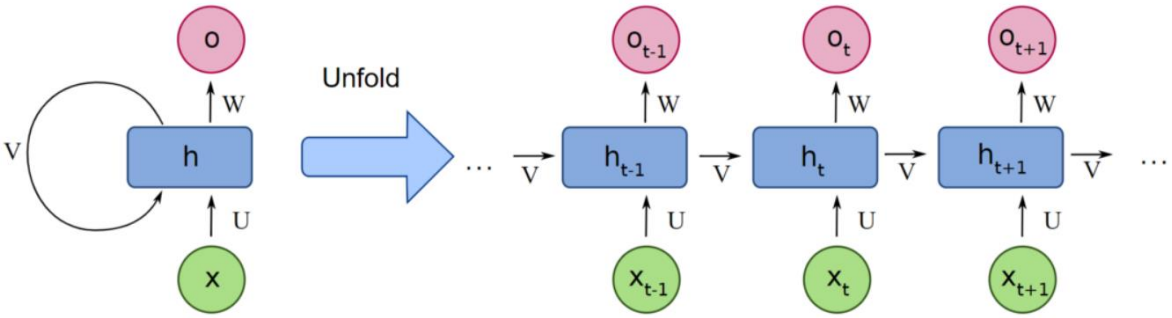
Một công thức tiêu biểu trong RNN là công thức cập nhật của mạng LSTM (Long Short-Term Memory). Công thức này giúp LSTM quyết định những thông tin nào sẽ được giữ lại và những thông tin nào sẽ bị bỏ qua qua các thời điểm:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Trong đó:

- x_t là đầu vào tại thời điểm t,
- h_{t-1} là trạng thái ẩn từ lớp trước,
- c_{t-1} là trạng thái tế bào của LSTM từ lớp trước,
- i_t, f_t, o_t là các cổng (gates) của LSTM để điều khiển thông tin,
- σ là hàm sigmoid và $\odot$ là phép nhân Hadamard (element-wise multiplication).

Công thức trên biểu diễn quá trình cập nhật trạng thái tế bào và trạng thái ẩn trong mạng LSTM, cho phép nó lưu trữ và truyền thông tin qua nhiều thời điểm một cách hiệu quả. Nhờ vào tính linh hoạt và khả năng học các phụ thuộc thời gian phức tạp, RNNs và các biến thể của chúng đã có những đóng góp đáng kể trong các lĩnh vực như dịch máy, nhận diện giọng nói, và dự đoán chuỗi thời gian.



Hình 1.6. Mạng RNN

1.3. Xử lý tín hiệu số

1.1.1. Giới thiệu

Kỹ thuật xử lý tín hiệu số (Digital Signal Processing - DSP) là một lĩnh vực quan trọng trong khoa học máy tính và kỹ thuật, tập trung vào phân tích và xử lý các tín hiệu được số hóa. Tín hiệu số có thể là dữ liệu từ âm thanh, hình ảnh, video hoặc các dạng dữ liệu khác, được chuyển đổi từ dạng tín hiệu liên tục (analog) sang dạng tín hiệu rời rạc (digital). Một trong những mục tiêu chính của DSP là biến đổi và cải thiện tín hiệu để dễ dàng phân tích hoặc sử dụng trong các ứng dụng khác. Các kỹ thuật cơ bản trong DSP bao gồm lọc tín hiệu, biến đổi Fourier, và phân tích tần số, giúp loại bỏ nhiễu, nén dữ liệu, và trích xuất các đặc trưng quan trọng từ tín hiệu.

Trong lĩnh vực chuyển đổi giọng nói thành văn bản, DSP đóng vai trò quan trọng trong việc chuẩn bị và tối ưu hóa dữ liệu âm thanh trước khi đưa vào các mô hình học sâu. Các phương pháp như Mel-Frequency Cepstral Coefficients (MFCC) và Fourier Transform được sử dụng để chuyển đổi tín hiệu âm thanh từ miền thời gian sang miền tần số, giúp trích xuất các đặc trưng quan trọng như cao độ, âm sắc và năng lượng. Những đặc trưng này sau đó được sử dụng làm đầu vào cho các mô hình học máy để nhận dạng giọng nói. Bằng cách áp dụng DSP, các hệ thống chuyển đổi giọng nói thành văn bản có thể hoạt động

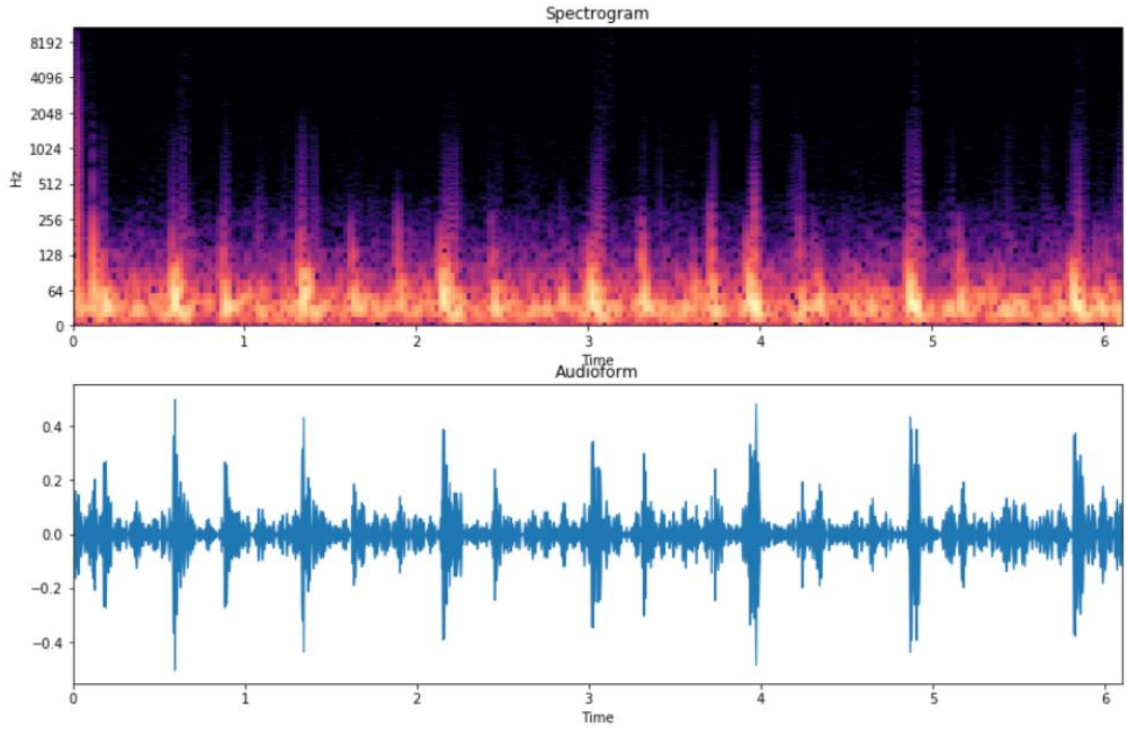
hiệu quả hơn, đạt độ chính xác cao hơn, và xử lý được các tín hiệu âm thanh phức tạp và đa dạng.[1]

1.1.2. Quang phổ

Quang phổ âm thanh là một biểu diễn đồ họa của các tín hiệu âm thanh trong không gian tần số. Nó được sử dụng để hiển thị mức độ phân bố của các tần số khác nhau trong một tín hiệu âm thanh. Quang phổ âm thanh là một công cụ quan trọng trong nhiều lĩnh vực, từ âm nhạc đến kỹ thuật âm thanh và xử lý tín hiệu.

Các phổ âm thanh được đo bằng cách sử dụng các thiết bị phân tích tín hiệu như máy phân tích quang phổ và phần mềm xử lý tín hiệu. Các tín hiệu âm thanh được đưa vào máy phân tích và sau đó phân tích để xác định mức độ phân bố của các tần số khác nhau trong tín hiệu. Kết quả của quá trình này được hiển thị dưới dạng một biểu đồ đường hoặc đường cong hình thang với các tần số được hiển thị trên trục x và mức độ phân bố được hiển thị trên trục y.

Quang phổ âm thanh được sử dụng rộng rãi trong việc phân tích và xử lý tín hiệu âm thanh. Nó cung cấp cho người dùng một cái nhìn tổng quan về mức độ phân bố các tần số trong tín hiệu, giúp họ xác định các vấn đề về âm thanh và tối ưu hóa chất lượng âm thanh. Nó cũng được sử dụng trong việc phát hiện các vấn đề kỹ thuật của thiết bị âm thanh, giúp người dùng xác định các lỗi và sửa chữa chúng.



Hình 1.7. Quang phổ

1.1.3. Phương pháp MFCC

Kỹ thuật Mel-Frequency Cepstral Coefficients (MFCC) là một phương pháp phổ biến trong xử lý tín hiệu âm thanh để trích xuất đặc trưng chủ yếu cho nhận dạng giọng nói và các ứng dụng liên quan đến âm thanh. MFCCs giúp biểu diễn âm thanh dưới dạng một vector đặc trưng có số chiều thấp, giúp giảm chiều dữ liệu mà vẫn bảo toàn được các đặc tính quan trọng của tín hiệu âm thanh.

Quá trình trích xuất MFCC bao gồm các bước chính sau:

- Pre-emphasis: Đây là bước đầu tiên để tăng độ nhạy của tín hiệu âm thanh bằng cách áp dụng một bộ lọc thông cao, nhấn mạnh các thành phần cao tần trong tín hiệu.

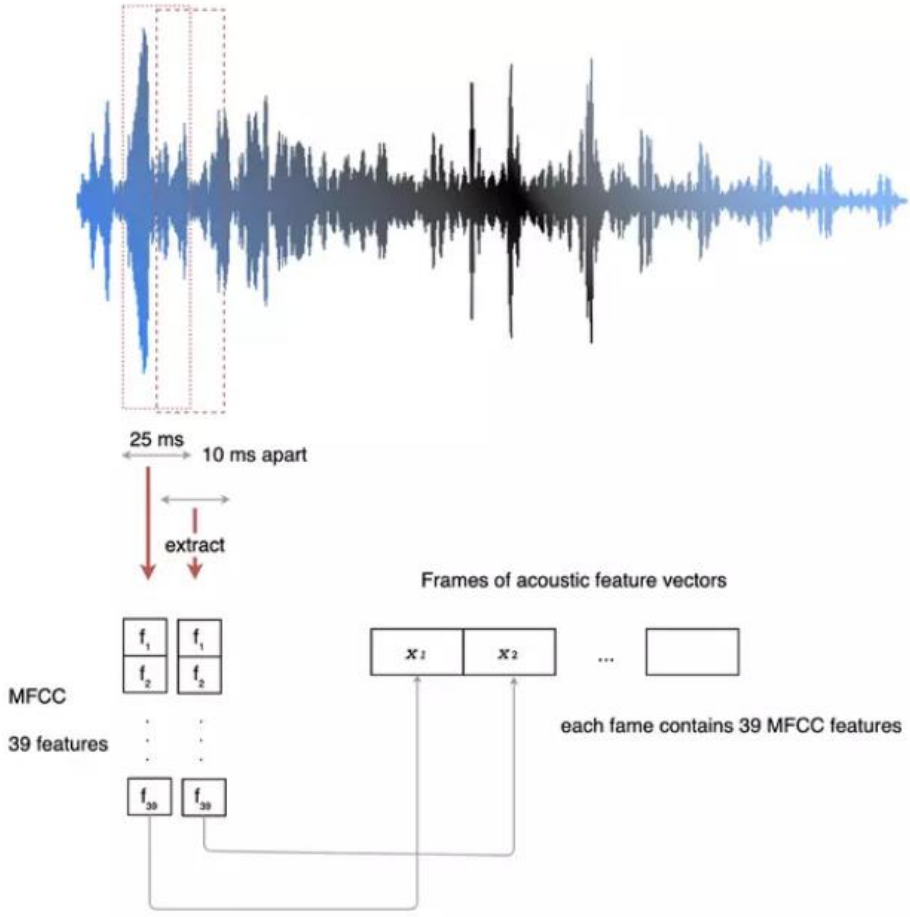
- Chia khung (Framing): Tín hiệu âm thanh được chia thành các khung nhỏ và chồng lên nhau. Mỗi khung thường có độ dài khoảng 20-40 ms và có thể chồng lên nhau khoảng 10-20 ms.
- Biến đổi Fourier (Fourier Transform): Áp dụng biến đổi Fourier (thường là biến đổi Fourier rời rạc - DFT) để chuyển đổi tín hiệu từ miền thời gian sang miền tần số. Điều này cho phép tính toán phổ tần số của mỗi khung.
- Mel Filtering: Tính toán phổ Mel bằng cách áp dụng bộ lọc Mel, mô phỏng cách người nghe nhận biết âm thanh. Bộ lọc Mel là một tập hợp các bộ lọc tam giác có trọng số, phù hợp với thang Mel.
- Logarithm: Áp dụng logarithm cơ số 10 cho phổ Mel để thu được phổ log-Mel.
- Tính toán Cepstral Coefficients: Sử dụng biến đổi cosin của phổ log-Mel để thu được các hệ số cepstral.

Công thức tiêu biểu cho quá trình tính toán MFCCs có thể được biểu diễn như sau:

$$MFCC_k = \sum_{m=1}^{M} \log \left(\sum_{n=1}^{N} |X[n]|^2 H_m[n] \cos \left[\frac{\pi k}{M} \left(n + \frac{1}{2} \right) \right] \right)$$

Trong đó:

- $X[n]$ là phổ của tín hiệu âm thanh ở khung thời gian n,
- $H_m[n]$ là bộ lọc Mel thứ m ở tần số n,
- M là số lượng hệ số cepstral,
- N là chiều dài của phổ tần số,
- k là chỉ số của hệ số cepstral.



Hình 1.8. Phương pháp MFCC

1.1.4. Phương pháp Fourier Transform

Biến đổi Fourier (Fourier Transform) là một kỹ thuật toán học cơ bản trong xử lý tín hiệu và tính toán, cho phép chuyển đổi một tín hiệu từ miền thời gian sang miền tần số. Được đặt theo tên nhà toán học Joseph Fourier, kỹ thuật này phân

tích một tín hiệu phức tạp thành các thành phần tần số cơ bản, giúp hiểu rõ hơn về cấu trúc và tính chất của tín hiệu đó.[5]

Biến đổi Fourier có nhiều ứng dụng quan trọng trong nhiều lĩnh vực, bao gồm:

- Xử lý tín hiệu âm thanh và hình ảnh: Biến đổi Fourier cho phép phân tích và xử lý tín hiệu âm thanh và hình ảnh, giúp nén dữ liệu, lọc tín hiệu, và nhận diện mẫu.
- Truyền thông và viễn thông: Trong các hệ thống truyền thông số, Biến đổi Fourier giúp phân tích và thiết kế các tín hiệu để tối ưu hóa hiệu suất truyền dẫn.
- Khoa học và kỹ thuật: Sử dụng để nghiên cứu và phân tích các hiện tượng sóng, từ trường, và các tín hiệu điện từ khác.

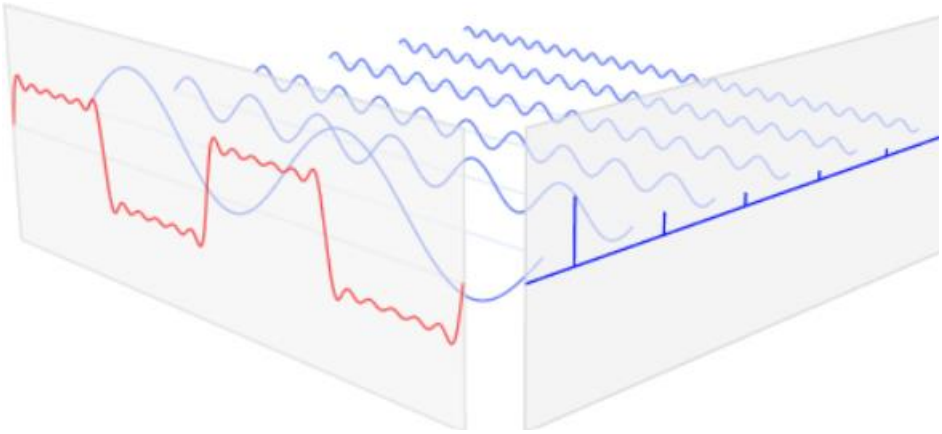
Biến đổi Fourier là một công cụ cơ bản nhưng mạnh mẽ trong xử lý và hiểu các tín hiệu phức tạp, với ứng dụng rộng rãi từ công nghệ thông tin đến khoa học tự nhiên và y sinh học.

Biến đổi Fourier của một hàm x(t) liên tục được định nghĩa bởi công thức sau:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft}\, dt$$

Trong đó:

- $X(f)$ là biến đổi Fourier của $x(t)$ tại tần số f,
- $x(t)$ là hàm thời gian,
- $e^{-j2\pi ft}$ là phần mũ phức tạp (phụ thuộc vào tần số f).



Hình 1.9. Phương pháp Fourier Transform

1.4. Python và các thư viện học sâu

1.4.1. Python

Python là một ngôn ngữ lập trình đa năng, được Guido van Rossum phát triển và phát hành lần đầu vào năm 1991. Với cú pháp đơn giản và dễ hiểu, Python giúp người dùng viết mã rõ ràng và ngắn gọn, đồng thời tăng cường khả năng đọc hiểu mã. Điều này làm cho Python trở thành lựa chọn phổ biến cho cả người mới bắt đầu học lập trình và các lập trình viên chuyên nghiệp. Python hỗ trợ nhiều phong cách lập trình khác nhau, bao gồm lập trình hướng đối tượng, lập trình chức năng và lập trình thủ tục. Một trong những điểm mạnh lớn của Python là thư viện chuẩn phong phú và cộng đồng hỗ trợ mạnh mẽ, cung cấp nhiều công cụ và tài nguyên cho các lập trình viên trong mọi lĩnh vực.

Python không chỉ giới hạn trong việc phát triển phần mềm mà còn được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau. Trong khoa học dữ liệu, Python là ngôn ngữ phổ biến nhất nhờ vào các thư viện như NumPy, pandas, và Matplotlib. Trong trí tuệ nhân tạo và học máy, các thư viện như TensorFlow và PyTorch đã đưa Python trở thành ngôn ngữ ưu tiên. Python cũng đóng vai trò quan trọng trong phát triển web với các framework như Django và Flask. Ngoài

ra, Python còn được sử dụng trong tự động hóa, phân tích dữ liệu, phát triển trò chơi, và nhiều lĩnh vực khác. Với sự linh hoạt và sức mạnh của mình, Python tiếp tục là một trong những ngôn ngữ lập trình quan trọng nhất và được ưa chuộng trên toàn thế giới.

Hình 1.10. Ngôn ngữ Python

1.4.2 Một số thư viện học sâu

Hai thư viện học sâu phổ biến trong Python là TensorFlow và PyTorch. Cả hai đều được sử dụng rộng rãi trong cộng đồng học máy và trí tuệ nhân tạo, hỗ trợ phát triển các mô hình học sâu phức tạp và hiệu quả.

TensorFlow là một thư viện mã nguồn mở được phát triển bởi Google Brain. Được ra mắt lần đầu vào năm 2015, TensorFlow đã trở thành một trong những công cụ mạnh mẽ nhất cho các nhà nghiên cứu và kỹ sư để xây dựng và triển khai các mô hình học sâu. TensorFlow hỗ trợ các hoạt động toán học đa chiều và tối ưu hóa mô hình, cho phép xây dựng các mạng nơ-ron phức tạp và đào tạo chúng trên các tập dữ liệu lớn. Một trong những điểm mạnh của TensorFlow là khả năng triển khai trên nhiều nền tảng khác nhau, từ các thiết bị di động đến các hệ thống máy chủ lớn. TensorFlow cũng cung cấp một giao diện lập trình

dễ sử dụng và một hệ sinh thái phong phú với các công cụ và thư viện bổ sung như TensorBoard để trực quan hóa và Keras để xây dựng mô hình dễ dàng hơn.

PyTorch là một thư viện mã nguồn mở được phát triển bởi Facebook AI Research. Ra mắt vào năm 2016, PyTorch nhanh chóng trở thành sự lựa chọn ưa thích của nhiều nhà nghiên cứu nhờ vào tính năng linh hoạt và dễ sử dụng. PyTorch hỗ trợ tính toán động, cho phép thay đổi và kiểm tra mô hình trong quá trình xây dựng một cách trực quan. Điều này giúp việc phát triển mô hình trở nên nhanh chóng và thuận tiện, đặc biệt là trong giai đoạn nghiên cứu và thử nghiệm. PyTorch cũng tích hợp chặt chẽ với các công cụ Python khác, tạo điều kiện thuận lợi cho việc tích hợp vào các dự án hiện có. Khả năng hỗ trợ GPU mạnh mẽ và các công cụ như TorchServe giúp triển khai mô hình PyTorch trên quy mô lớn, từ các thiết bị đơn lẻ đến các hệ thống phân tán.

Hình 1.11. Thư viện Pytorch

CHƯƠNG II: PHÁT BIỂU BÀI TOÁN VÀ ĐỀ XUẤT MỘT SỐ MÔ HÌNH

2.1. Phát biểu bài toán

2.1.1. Mục tiêu

Bài toán xây dựng ứng dụng chuyển đổi giọng nói thành văn bản (speech-to-text) nhằm giải quyết vấn đề chuyển đổi tín hiệu âm thanh chứa giọng nói con người thành văn bản chính xác. Vấn đề này bao gồm việc nhận diện và phân biệt âm thanh của giọng nói trong môi trường có nhiễu, hiểu và phân tích ngôn ngữ tự nhiên, đồng thời chuyển đổi chúng thành các ký tự và câu có nghĩa. Việc này đặc biệt quan trọng trong nhiều lĩnh vực như trợ lý ảo, dịch vụ khách hàng, giáo dục và chăm sóc sức khỏe. Ngoài ra, công nghệ này cũng cần xử lý sự đa dạng của các giọng nói, ngữ điệu, tốc độ nói và các yếu tố ngữ âm khác để đảm bảo tính chính xác và khả dụng cao.

Dữ liệu đầu vào của bài toán này là các tín hiệu âm thanh chứa giọng nói. Thông thường, dữ liệu này có thể được thu thập thông qua micro của các thiết bị điện tử như điện thoại di động, máy tính bảng, máy tính xách tay hoặc các thiết bị ghi âm chuyên dụng. Tín hiệu âm thanh đầu vào cần được xử lý để loại bỏ nhiễu và tăng cường chất lượng âm thanh trước khi chuyển đổi thành văn bản. Để đạt hiệu quả cao, dữ liệu đầu vào cũng cần được ghi nhận trong các điều kiện khác nhau về âm lượng, khoảng cách và môi trường xung quanh để hệ thống có thể học hỏi và thích ứng với các tình huống thực tế đa dạng.

Để xây dựng ứng dụng chuyển đổi giọng nói thành văn bản, cần sử dụng các công cụ và công nghệ tiên tiến trong lĩnh vực xử lý tín hiệu âm thanh và trí tuệ nhân tạo. Một trong những công cụ quan trọng là các mô hình học sâu (deep learning), đặc biệt là mạng nơ-ron hồi tiếp (Recurrent Neural Networks - RNN) và mạng nơ-ron tích chập (Convolutional Neural Networks - CNN). Ngoài ra, các công cụ như Google Cloud Speech-to-Text, Microsoft Azure Speech

Services, và Amazon Transcribe cũng cung cấp các API mạnh mẽ để thực hiện chuyển đổi giọng nói thành văn bản. Để xử lý và phân tích dữ liệu âm thanh, các thư viện như Librosa, NLTK, và PyDub có thể được sử dụng. Hơn nữa, việc sử dụng các công cụ tối ưu hóa và điều chỉnh mô hình như TensorFlow, PyTorch và Keras sẽ giúp cải thiện hiệu suất và độ chính xác của hệ thống.[9]

Kết quả mong muốn của bài toán này là một ứng dụng chuyển đổi giọng nói thành văn bản hoạt động hiệu quả và chính xác, có thể xử lý được nhiều loại giọng nói và ngữ cảnh khác nhau. Ứng dụng cần có khả năng nhận diện giọng nói trong môi trường có nhiều tiếng ồn, hiểu được các từ ngữ và câu phức tạp, đồng thời chuyển đổi chúng thành văn bản một cách nhanh chóng và chính xác. Mục tiêu là tạo ra một hệ thống có thể áp dụng rộng rãi trong các lĩnh vực khác nhau, từ trợ lý ảo, dịch vụ khách hàng, đến hỗ trợ người khuyết tật và cải thiện năng suất làm việc. Ứng dụng cũng cần thân thiện với người dùng, dễ sử dụng và có khả năng học hỏi để cải thiện chất lượng dịch vụ theo thời gian.

1.1.2. Dữ liệu đầu vào

Một trong những yêu cầu quan trọng nhất đối với dữ liệu âm thanh đầu vào là chất lượng âm thanh phải rõ ràng và không bị méo. Để đạt được điều này, tín hiệu âm thanh cần được thu thập bằng các thiết bị ghi âm chất lượng cao và trong môi trường ít nhiễu. Tín hiệu âm thanh cần có tần số lấy mẫu (sampling rate) cao, thường là 16kHz hoặc 44.1kHz, để đảm bảo rằng tất cả các chi tiết của giọng nói được ghi nhận chính xác. Ngoài ra, độ phân giải bit (bit depth) của âm thanh cũng cần đủ cao, thường là 16 bit hoặc 24 bit, để giảm thiểu tiếng ồn và nhiễu kỹ thuật số. Một tín hiệu âm thanh rõ ràng và chất lượng cao giúp các mô hình xử lý giọng nói có thể phân tích và nhận dạng các đặc điểm ngữ âm một cách chính xác hơn.

Dữ liệu âm thanh đầu vào cần đa dạng về giọng nói và ngữ cảnh để hệ thống có thể học và xử lý hiệu quả trong các tình huống thực tế. Điều này bao gồm việc thu thập dữ liệu từ nhiều người nói khác nhau với các giọng điệu, tốc độ nói, và độ tuổi khác nhau. Đặc biệt, hệ thống cần được đào tạo với cả giọng nam và giọng nữ, giọng trẻ và giọng già, cũng như các giọng địa phương khác nhau để đảm bảo tính toàn diện. Ngoài ra, dữ liệu âm thanh cũng nên được ghi nhận trong các môi trường khác nhau như phòng yên tĩnh, ngoài trời, trong ô tô, và các nơi công cộng để hệ thống có thể xử lý tốt. Sự đa dạng này giúp hệ thống học hỏi và thích ứng với các biến đổi trong giọng nói và môi trường, từ đó cải thiện độ chính xác và khả năng ứng dụng của công nghệ chuyển đổi giọng nói thành văn bản.[6]

1.1.3. Kết quả mong muốn

Kết quả mong muốn từ quá trình huấn luyện mô hình chuyển đổi giọng nói thành văn bản là một hệ thống có khả năng nhận diện và chuyển đổi giọng nói thành văn bản với độ chính xác cao. Sau khi hoàn thành quá trình huấn luyện, mô hình cần có khả năng xử lý các tệp âm thanh đầu vào và tạo ra văn bản phản ánh đúng nội dung giọng nói với mức độ lỗi tối thiểu, bao gồm việc nhận diện đúng các từ, cụm từ, và câu trong các ngữ cảnh khác nhau, cũng như khả năng phân biệt và bỏ qua các tiếng ồn nền không liên quan. Mô hình cũng cần phải hoạt động hiệu quả với nhiều loại giọng nói, từ giọng nói rõ ràng trong môi trường yên tĩnh đến giọng nói trong điều kiện nhiều nhiễu âm. Độ chính xác cao của dữ liệu âm thanh sau khi huấn luyện sẽ đảm bảo rằng hệ thống có thể áp dụng được trong các tình huống thực tế và cung cấp kết quả đáng tin cậy.

Kết quả mong muốn của bài toán này cũng bao gồm việc phát triển một ứng dụng web chuyển giọng nói thành văn bản, cung cấp một giao diện người dùng thân thiện và dễ sử dụng. Ứng dụng cần có khả năng thu âm trực tiếp hoặc tải lên các tệp âm thanh, sau đó xử lý và hiển thị văn bản tương ứng một cách

nhanh chóng và chính xác. Tốc độ xử lý nhanh và khả năng tương tác trực quan là những yếu tố quan trọng để đảm bảo trải nghiệm người dùng tốt. Ứng dụng cũng nên cung cấp các tùy chọn chỉnh sửa văn bản sau khi chuyển đổi để người dùng có thể sửa chữa các lỗi nhận dạng nếu có. Ngoài ra, khả năng lưu trữ và chia sẻ văn bản trực tiếp sẽ tăng thêm tiện ích cho người dùng. Ứng dụng này sẽ giúp các cá nhân và doanh nghiệp tiết kiệm thời gian và công sức trong việc chuyển đổi giọng nói thành văn bản, từ đó nâng cao hiệu quả công việc và giao tiếp hàng ngày.

2.2 Một số mô hình đề xuất

2.2.1 Mô hình DeepSpeech

DeepSpeech là một hệ thống chuyển đổi giọng nói thành văn bản mã nguồn mở, phát triển bởi Mozilla, dựa trên các mô hình học sâu (deep learning). Mô hình này được lấy cảm hứng từ bài báo "Deep Speech: Scaling up end-to-end speech recognition" của Baidu, nhắm đến việc tạo ra một hệ thống nhận dạng giọng nói hiệu quả và dễ sử dụng. DeepSpeech sử dụng kiến trúc mạng nơ-ron hồi tiếp (Recurrent Neural Network - RNN) với các lớp LSTM (Long Short-Term Memory) để xử lý chuỗi tín hiệu âm thanh và chuyển đổi chúng thành văn bản. Một trong những điểm mạnh của DeepSpeech là nó sử dụng kỹ thuật Connectionist Temporal Classification (CTC), giúp mô hình học cách liên kết giữa chuỗi âm thanh và văn bản mà không cần nhãn thời gian chính xác.

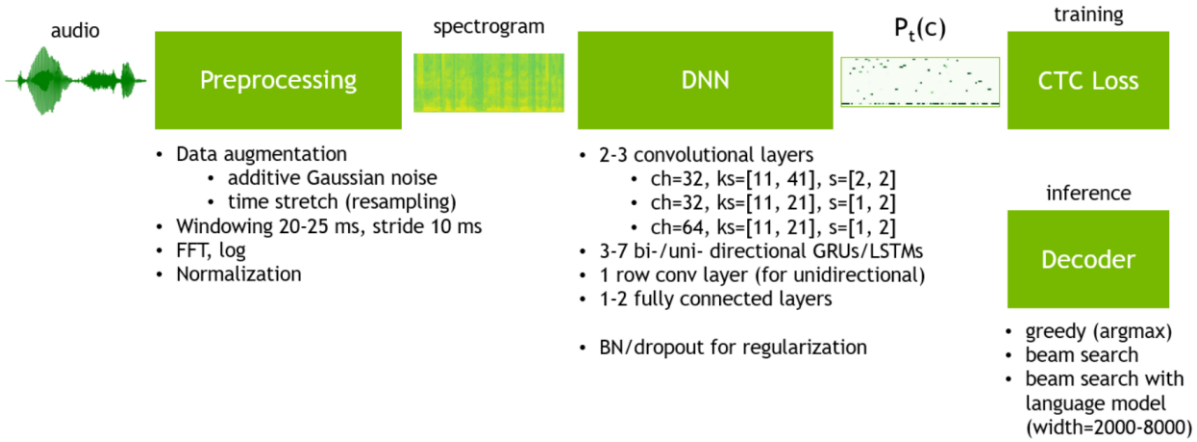
Ưu điểm của DeepSpeech

Một trong những ưu điểm lớn của DeepSpeech là tính đơn giản và hiệu quả của kiến trúc mô hình. Việc sử dụng LSTM giúp mô hình có khả năng ghi nhớ và xử lý thông tin trong các chuỗi dài, làm cho DeepSpeech hoạt động tốt với các đoạn âm thanh dài và phức tạp. Kỹ thuật CTC giúp giảm bớt yêu cầu về nhãn thời gian chi tiết, làm cho quá trình huấn luyện trở nên dễ dàng và ít tốn kém hơn. DeepSpeech là mã nguồn mở, nên cộng đồng có thể dễ dàng đóng góp và

cải tiến mô hình, cũng như sử dụng mô hình này cho các dự án cá nhân và thương mại mà không gặp rào cản về bản quyền. Thêm vào đó, DeepSpeech có khả năng hoạt động tốt trên nhiều nền tảng khác nhau, từ máy tính cá nhân đến thiết bị di động.

Hạn chế của DeepSpeech

DeepSpeech cũng có một số nhược điểm. Đầu tiên, do sử dụng kiến trúc RNN và LSTM, DeepSpeech có thể gặp khó khăn với các đoạn âm thanh có độ nhiễu cao hoặc giọng nói không rõ ràng. Điều này có thể làm giảm độ chính xác của hệ thống trong các điều kiện thực tế. Ngoài ra, mặc dù DeepSpeech là mã nguồn mở và có tính linh hoạt cao, nhưng việc cài đặt và triển khai mô hình có thể phức tạp đối với người dùng không có nhiều kinh nghiệm trong lĩnh vực học sâu và xử lý giọng nói. Cuối cùng, mặc dù cộng đồng đóng góp tích cực, DeepSpeech vẫn còn phụ thuộc nhiều vào tài nguyên và dữ liệu huấn luyện chất lượng cao để đạt được hiệu suất tốt nhất. Điều này có thể là một rào cản đối với những ai không có đủ dữ liệu hoặc tài nguyên tính toán.



Hình 2.1. Kiến trúc mô hình DeepSpeech

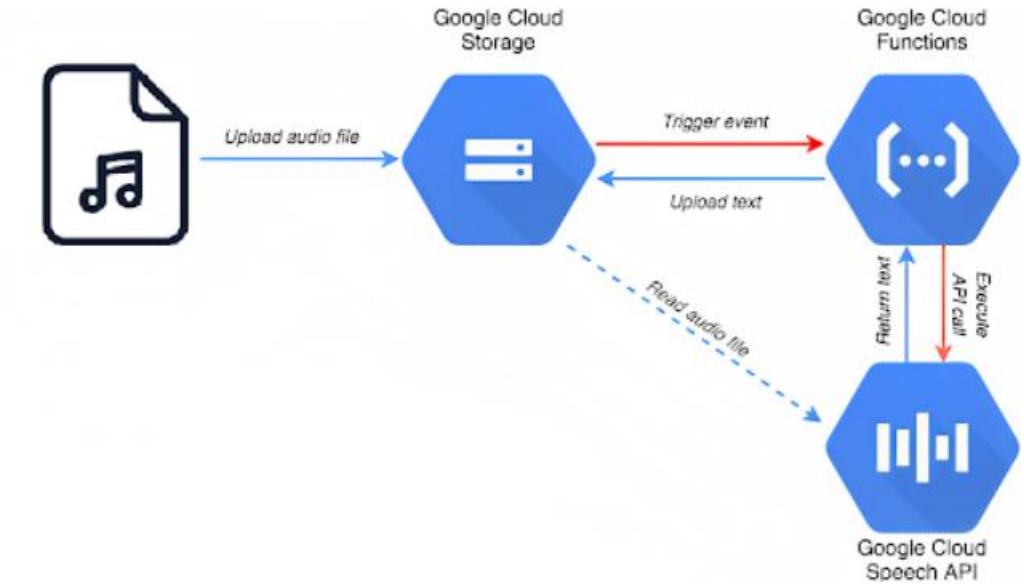
2.2.2 Mô hình Google Speech-to-text

Google Speech-to-Text là một dịch vụ chuyển đổi giọng nói thành văn bản do Google phát triển, sử dụng các công nghệ học sâu tiên tiến. Dịch vụ này cung

cấp khả năng nhận diện giọng nói mạnh mẽ và hỗ trợ nhiều ngôn ngữ, giúp người dùng chuyển đổi âm thanh thành văn bản một cách nhanh chóng và chính xác. Google Speech-to-Text sử dụng các mô hình mạng nơ-ron tích chập (Convolutional Neural Networks - CNN) và Attention-based Encoder-Decoder để cải thiện độ chính xác và hiệu quả xử lý.

Google Speech-to-Text có nhiều ưu điểm nổi bật. Đầu tiên, nó hỗ trợ nhận diện giọng nói đa ngôn ngữ và trong nhiều ngữ cảnh khác nhau, bao gồm cả các giọng nói có tiếng địa phương. Thứ hai, dịch vụ này cung cấp API dễ sử dụng, cho phép tích hợp dễ dàng vào các ứng dụng web và di động. Thứ ba, Google Speech-to-Text có khả năng xử lý dữ liệu âm thanh trong thời gian thực, cung cấp kết quả nhanh chóng và chính xác, giúp cải thiện trải nghiệm người dùng.

Tuy nhiên, Google Speech-to-Text cũng có một số nhược điểm. Đầu tiên, dịch vụ này phụ thuộc vào kết nối Internet, do đó hiệu suất có thể bị ảnh hưởng trong điều kiện mạng yếu. Thứ hai, chi phí sử dụng dịch vụ có thể cao đối với các doanh nghiệp nhỏ hoặc các dự án cá nhân khi xử lý một lượng lớn dữ liệu âm thanh. Cuối cùng, mặc dù dịch vụ này rất mạnh mẽ, nhưng trong một số trường hợp, độ chính xác của nhận diện giọng nói có thể bị giảm sút khi gặp phải tiếng ồn lớn hoặc giọng nói không rõ ràng.



Hình 2.2. Kiến trúc mô hình Google Speech-to-text

2.2.3 Mô hình Jasper

Jasper là một mô hình chuyển đổi giọng nói thành văn bản được phát triển bởi Google, nổi bật với khả năng xử lý âm thanh nhanh chóng và chính xác. Jasper sử dụng kiến trúc mạng nơ-ron học sâu tiên tiến, bao gồm các mạng nơ-ron tích chập (CNN) và mạng nơ-ron hồi tiếp (RNN), để phân tích và chuyển đổi tín hiệu âm thanh thành văn bản. Mô hình này được tối ưu hóa cho hiệu suất cao, giúp giảm thiểu độ trễ và tăng cường độ chính xác trong nhận diện giọng nói. Jasper được thiết kế để hoạt động tốt trong nhiều ngữ cảnh khác nhau, từ các ứng dụng di động đến các hệ thống nhận diện giọng nói trong các thiết bị thông minh.

So với DeepSpeech của Mozilla, Jasper có những cải tiến đáng kể về tốc độ và độ chính xác. Trong khi DeepSpeech sử dụng kiến trúc RNN với LSTM và kỹ thuật CTC, Jasper kết hợp các CNN và RNN để tận dụng ưu điểm của cả hai loại mạng. Điều này giúp Jasper xử lý âm thanh nhanh hơn và với độ chính xác

cao hơn trong nhiều trường hợp. Ngoài ra, Jasper còn được tối ưu hóa để hoạt động hiệu quả trên nhiều nền tảng phần cứng khác nhau, từ máy tính đến thiết bị di động, giúp mở rộng phạm vi ứng dụng.

So với Google Speech-to-Text, Jasper được phát triển với trọng tâm vào hiệu suất và khả năng tùy chỉnh cao hơn. Mặc dù cả hai đều sử dụng các mô hình học sâu tiên tiến và hỗ trợ đa ngôn ngữ, Jasper cho phép người dùng dễ dàng điều chỉnh và tối ưu hóa mô hình cho các ứng dụng cụ thể. Điều này mang lại lợi thế trong các tình huống yêu cầu độ chính xác cao và tốc độ xử lý nhanh. Tuy nhiên, Google Speech-to-Text có lợi thế về tích hợp liền mạch với các dịch vụ và nền tảng của Google, tạo ra một hệ sinh thái phong phú và dễ sử dụng.

Ưu điểm của Jasper

Jasper có nhiều ưu điểm nổi bật. Đầu tiên, nhờ vào kiến trúc kết hợp CNN và RNN, Jasper có khả năng xử lý nhanh chóng và chính xác các tín hiệu âm thanh, ngay cả trong môi trường nhiễu loạn. Thứ hai, Jasper được thiết kế với tính năng tùy chỉnh cao, cho phép các nhà phát triển điều chỉnh mô hình theo nhu cầu cụ thể của từng ứng dụng. Điều này làm cho Jasper trở thành lựa chọn linh hoạt và mạnh mẽ cho nhiều loại dự án khác nhau. Cuối cùng, Jasper hỗ trợ nhiều ngôn ngữ và có khả năng hoạt động tốt trên nhiều nền tảng phần cứng, từ máy tính cá nhân đến các thiết bị di động và nhúng.

Nhược điểm của Jasper

Tuy nhiên, Jasper cũng có một số nhược điểm. Một trong những thách thức chính là yêu cầu tài nguyên tính toán lớn để huấn luyện và triển khai mô hình. Điều này có thể tạo ra rào cản đối với các cá nhân hoặc tổ chức có hạn chế về tài nguyên phần cứng. Thêm vào đó, mặc dù Jasper cung cấp tính năng tùy chỉnh cao, nhưng việc điều chỉnh và tối ưu hóa mô hình đòi hỏi kiến thức sâu về học sâu và xử lý ngôn ngữ tự nhiên, có thể là thách thức đối với người dùng không chuyên. Cuối cùng, như với bất kỳ mô hình học sâu nào, Jasper cũng

phụ thuộc mạnh mẽ vào chất lượng dữ liệu huấn luyện để đạt được hiệu suất tốt nhất, do đó yêu cầu một lượng lớn dữ liệu âm thanh chất lượng cao.

Kiến trúc mô hình Jasper

Jasper là một nhóm các mô hình ASR đầu cuối thay thế các mô hình âm thanh và phát âm bằng mạng lưới thần kinh tích chập. Jasper sử dụng tính năng mel-filterbank tính từ 20ms các cửa sổ có độ chồng chéo 10ms và đưa ra phân phối xác suất cho các ký tự trên mỗi khung

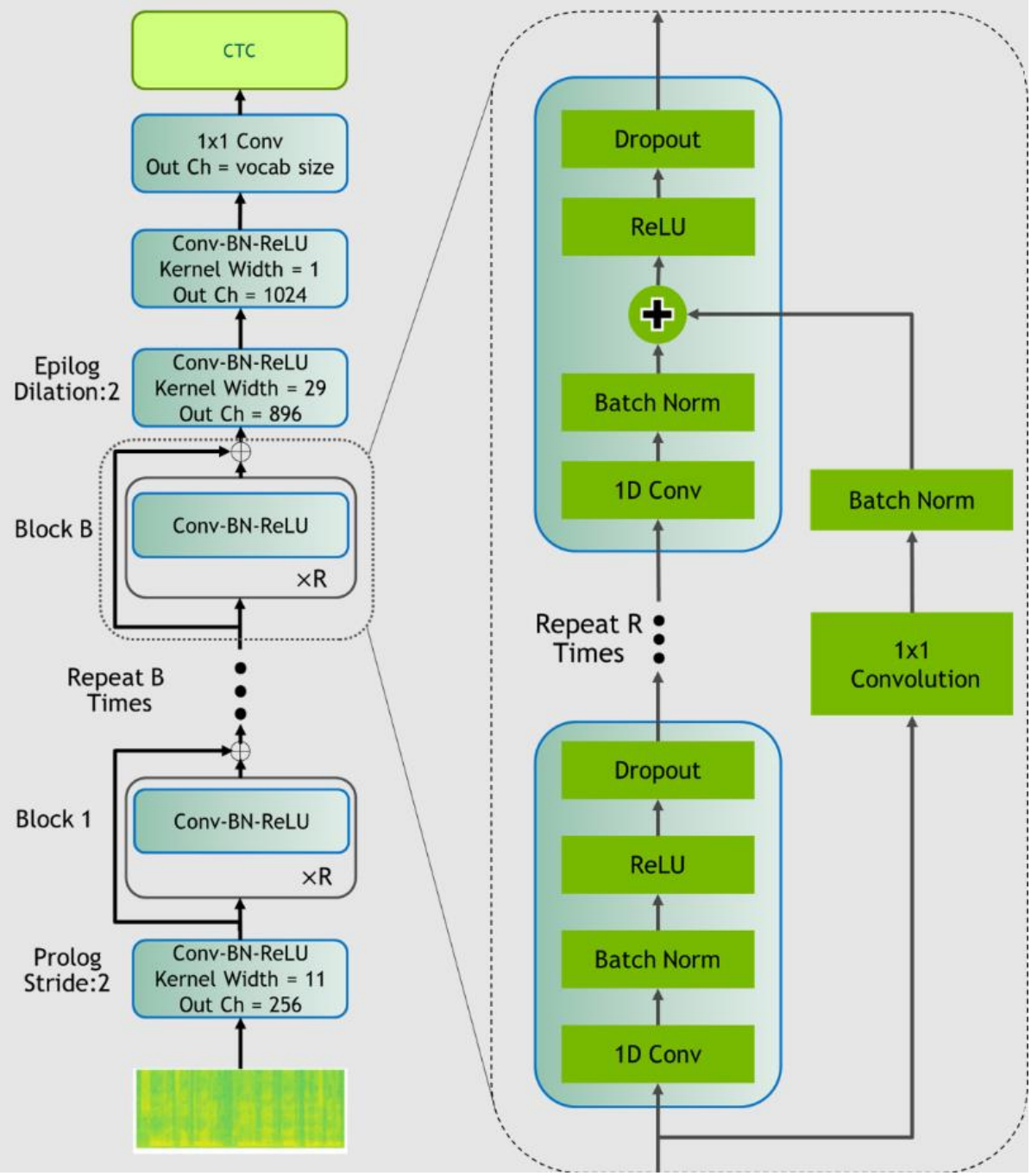
Jasper có cấu trúc khối: mô hình Jasper BxR có khối B, mỗi khối có R khối phụ. Mỗi khối con áp dụng các thao tác sau: tích chập 1D, định mức lô, ReLU và bỏ học. Tất cả các khối con trong một khối có cùng số kênh đầu ra. Mỗi đầu vào khối được kết nối trực tiếp vào khối con cuối cùng thông qua kết nối dư. Kết nối còn lại là đầu tiên được chiếu thông qua tích chập 1x1 để tính số lượng kênh đầu vào và đầu ra khác nhau, sau đó thông qua một đợt lớp định mức. Đầu ra của lớp định mức lô này được thêm vào đầu ra của lớp định mức lô trong khối con cuối cùng. Kết quả của tổng này được chuyển qua hàm kích hoạt và dropout để tạo ra đầu ra của khối hiện tại.

Kiến trúc khối con của Jasper được thiết kế để tạo điều kiện cho việc suy luận GPU nhanh chóng. Mỗi khối con có thể được hợp nhất thành một nhân GPU đơn: dropout không được sử dụng tại thời điểm suy luận và được bị loại bỏ, định mức lô có thể được hợp nhất với tích chập trước đó, ReLU kẹp kết quả và tổng dư có thể được được coi như một thuật ngữ sai lệch được sửa đổi trong hoạt động hợp nhất này.

Tất cả các mô hình Jasper đều có thêm bốn phép chập khối: một khối tiền xử lý và ba khối xử lý hậu kỳ. Xem Hình 1 và Bảng 1 để biết chi tiết.

# Blocks	Block	Kernel	# Output Channels	Dropout	# Sub Blocks
1	Conv1	11 *stride=2*	256	0.2	1
2	B1	11	256	0.2	5
2	B2	13	384	0.2	5
2	B3	17	512	0.2	5
2	B4	21	640	0.3	5
2	B5	25	768	0.3	5
1	Conv2	29 *dilation=2*	896	0.4	1
1	Conv3	1	1024	0.4	1
1	Conv4	1	# graphemes	0	1

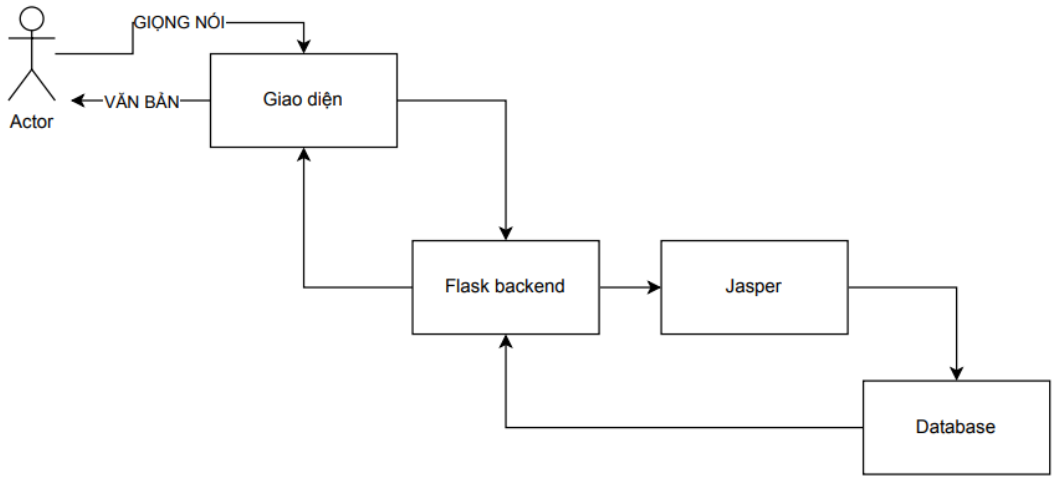
Bảng 2.1. Jasper 10x5: 10 khối, mỗi khối gồm 5 khối con chập 1D, cộng thêm 4 khối bổ sung.



Hình 2.3. Kiến trúc mô hình Jasper

CHƯƠNG III: XÂY DỰNG MÔ HÌNH HỌC SÂU TÍCH HỢP TRANG WEB

3.1 Tổng quan mô hình



Hình 3.1. Tổng quan cấu trúc ứng dụng

Để chuyển đổi giọng nói thành văn bản bằng Jasper và nhận giọng nói từ một ứng dụng web Flask, cần thực hiện các bước sau:[11]

Thiết lập ứng dụng Flask: Tạo một ứng dụng Flask để tạo một giao diện web và xử lý yêu cầu từ người dùng.

Ghi âm giọng nói: Trong giao diện web, cung cấp một phần để người dùng có thể ghi âm giọng nói. Sử dụng các công nghệ như HTML5 Recorder hoặc MediaRecorder API để ghi âm trực tiếp từ trình duyệt.

Lưu trữ và xử lý âm thanh: Khi người dùng ghi âm, lưu trữ tệp âm thanh được ghi âm trên máy chủ Flask. Ta có thể lưu trữ tệp âm thanh trong thư mục tạm thời hoặc cơ sở dữ liệu, tuỳ thuộc vào yêu cầu

Gửi âm thanh cho Jasper: Sử dụng Jasper để xử lý tệp âm thanh đã được ghi âm. Ta cần truyền tệp âm thanh tới Jasper và sử dụng các công cụ nhận dạng

giọng nói của nó để chuyển đổi giọng nói thành văn bản. Để làm điều này, ta có thể sử dụng các thư viện hoặc API liên quan đến Jasper.

Xử lý kết quả và hiển thị văn bản: Sau khi Jasper đã chuyển đổi giọng nói thành văn bản, nhận kết quả và xử lý nó trong ứng dụng Flask . Ta có thể hiển thị văn bản được chuyển đổi lên giao diện web để người dùng xem.

3.2 Huấn luyện mô hình học sâu

3.2.1 Dữ liệu huấn luyện

Giới thiệu:

VIVOS là một kho ngữ liệu tiếng Việt miễn phí gồm 15 giờ ghi âm giọng nói được chuẩn bị cho nhiệm vụ Nhận dạng giọng nói tự động. Kho ngữ liệu được AILAB, một phòng thí nghiệm khoa học máy tính của Đại học Khoa học Tự nhiên, ĐHQG-HCM, biên soạn, với GS. Vũ Hải Quân là người đứng đầu. Dữ liệu xuất bản với hy vọng thu hút thêm nhiều nhà khoa học tham gia giải quyết các vấn đề nhận dạng giọng nói tiếng Việt. Kho ngữ liệu này chỉ nên được sử dụng cho mục đích học thuật.

Giấy phép bản quyền:

Creative Commons Attribution NonCommercial ShareAlike v4.0 (CC BY-NC-SA 4.0)

Đặc điểm bộ dữ liệu:

Bài phát biểu được ghi âm trong môi trường yên tĩnh với micro chất lượng cao, người nói được yêu cầu đọc từng câu một.

	Training	Testing
Speakers	46	19
Male	22	12
Female	24	7
Utterances	11660	760
Duration	14:55	00:45
Unique Syllables	4617	1692

Bảng 3.1. Một số đặc trưng bộ dữ liệu VIVOS

Đánh giá bộ dữ liệu tiền huấn luyện:

Ngữ liệu cũng được đánh giá bằng cách sử dụng công thức không chuyên của chúng tôi dành cho hệ thống Nhận dạng giọng nói tiếng Việt, được mô tả chi tiết trong bài báo liên quan.

	baseline	+pitch	+tone
mGMM	19.66	15.14	14.91
mGMM+MMI	18.08	14.96	13.91
mGMM+SAT	15.79	12.07	12.13
mDNN+SAT	13.34	9.54	9.48

Bảng 3.2. Đánh giá bộ dữ liệu qua một số công thức

3.2.2 Tiền xử lý dữ liệu

Một số thư viện cần cài đặt

```
import pandas as pd

import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from IPython import display
from jiwer import wer
```

Chuẩn bị vocabulary để sử dụng

```
# The set of characters accepted in the transcription.
characters = [x for x in "abcdefghijklmnopqrstuvwxyz'?! "]
# Mapping characters to integers
char_to_num = keras.layers.StringLookup(vocabulary=characters, oov_token="")
# Mapping integers back to original characters
num_to_char = keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)

print(
    f"The vocabulary is: {char_to_num.get_vocabulary()} "
    f"(size ={char_to_num.vocabulary_size()})"
)
```

Tạo hàm mô tả phép biến đổi áp dụng cho từng phần tử của tập dữ liệu

```
# An integer scalar Tensor. The window length in samples.
frame_length = 256
# An integer scalar Tensor. The number of samples to step.
frame_step = 160
# An integer scalar Tensor. The size of the FFT to apply.
# If not provided, uses the smallest power of 2 enclosing frame_length.
fft_length = 384
```

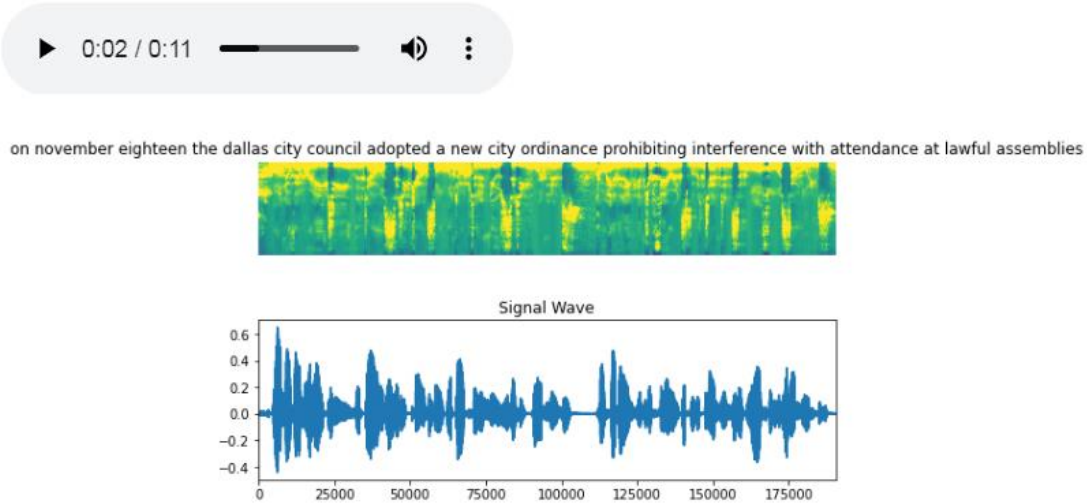
```
def encode_single_sample(wav_file, label):
    ###########################################
    ##  Process the Audio
    ###########################################
    # 1. Read wav file
    file = tf.io.read_file(wavs_path + wav_file + ".wav")
    # 2. Decode the wav file
    audio, _ = tf.audio.decode_wav(file)
    audio = tf.squeeze(audio, axis=-1)
    # 3. Change type to float
    audio = tf.cast(audio, tf.float32)
    # 4. Get the spectrogram
    spectrogram = tf.signal.stft(
        audio,         frame_length=frame_length,         frame_step=frame_step,
fft_length=fft_length
    )
    # 5. We only need the magnitude, which can be derived by applying tf.abs
    spectrogram = tf.abs(spectrogram)
    spectrogram = tf.math.pow(spectrogram, 0.5)
    # 6. normalisation
    means = tf.math.reduce_mean(spectrogram, 1, keepdims=True)
    stddevs = tf.math.reduce_std(spectrogram, 1, keepdims=True)
    spectrogram = (spectrogram - means) / (stddevs + 1e-10)
    ###########################################
    ##  Process the label
    ###########################################
    # 7. Convert label to Lower case
    label = tf.strings.lower(label)
    # 8. Split the label
    label = tf.strings.unicode_split(label, input_encoding="UTF-8")
    # 9. Map the characters in label to numbers
    label = char_to_num(label)
    # 10. Return a dict as our model is expecting two inputs
    return spectrogram, label
```

Mô hình hóa dữ liệu huấn luyện

```
fig = plt.figure(figsize=(8, 5))
```

```
for batch in train_dataset.take(1):
    spectrogram = batch[0][0].numpy()
    spectrogram = np.array([np.trim_zeros(x) for x in np.transpose(spectrogram)])
    label = batch[1][0]
    # Spectrogram
    label = tf.strings.reduce_join(num_to_char(label)).numpy().decode("utf-8")
    ax = plt.subplot(2, 1, 1)
    ax.imshow(spectrogram, vmax=1)
    ax.set_title(label)
    ax.axis("off")
    # Wav
    file = tf.io.read_file(wavs_path + list(df_train["file_name"])[0] + ".wav")
    audio, _ = tf.audio.decode_wav(file)
    audio = audio.numpy()
    ax = plt.subplot(2, 1, 2)
    plt.plot(audio)
    ax.set_title("Signal Wave")
    ax.set_xlim(0, len(audio))
    display.display(display.Audio(np.transpose(audio), rate=16000))
plt.show()
```

Dữ liệu sau khi được đưa về quang phổ:



Hình 3.2. Dữ liệu âm thanh được mô tả dưới dạng quang phổ

3.2.3 Huấn luyện dữ liệu

Để đào tạo mô hình bằng độ chính xác hỗn hợp hoặc TF32 với Tensor Cores hoặc bằng FP32, hãy thực hiện các bước sau bằng cách sử dụng các tham số mặc định của mô hình Jasper trên tập dữ liệu VIVOS.

3.2.3.1 Tải về code

```
git clone https://github.com/NVIDIA/DeepLearningExamples

cd DeepLearningExamples/PyTorch/SpeechRecognition/Jasper
```

3.2.3.2 Xây dựng Jasper PyTorch container

Chạy các tập lệnh sau sẽ xây dựng và khởi chạy vùng chứa chứa tất cả các phụ thuộc cần thiết để tải xuống và xử lý dữ liệu cũng như đào tạo và suy luận mô hình.

```
bash scripts/docker/build.sh
```

3.2.3.3 Bắt đầu phiên tương tác trong vùng chứa NGC để chạy tải xuống/đào tạo/suy luận dữ liệu

```
bash scripts/docker/launch.sh <DATA_DIR> <CHECKPOINT_DIR> <OUTPUT_DIR>
```

Trong vùng chứa, nội dung của kho lưu trữ này sẽ được sao chép vào thư mục /workspace/jasper. Các thư mục /datasets, /checkpoints, /results được gắn kết dưới dạng ổ đĩa và được ánh xạ tới các thư mục tương ứng <DATA_DIR>, <CHECKPOINT_DIR>, <OUTPUT_DIR> trên máy chủ.

3.2.3.4 Tải và tiền xử lý dữ liệu (Xem mục 3.2.2)

3.2.3.5 Huấn luyện dữ liệu

Bên trong container, sử dụng tập lệnh sau để bắt đầu đào tạo. Đảm bảo tập dữ liệu đã tải xuống và được xử lý trước nằm tại <DATA_DIR>/LibriSpeech trên máy chủ (xem Bước 3), tương ứng với /datasets/LibriSpeech bên trong container.

```
[OPTION1=value1 OPTION2=value2 ...] bash scripts/train.sh
```

Các parameter có thể để dưới dạng biến môi trường. Ta có thể tìm thêm thông tin chi tiết về các tùy chọn có sẵn trong danh sách tham số dưới đây.

```
DATA_DIR:directory of dataset. (default: '/datasets/LibriSpeech')

MODEL_CONFIG: relative path to model configuration .(default: 'configs/jasper10x5dr_speedp-online_speca.yaml')

OUTPUT_DIR: directory for results, logs, and created checkpoints. (default: '/results')

CHECKPOINT: a specific model checkpoint to continue training from. To resume training from the last checkpoint, see the RESUME option.

RESUME: resume training from the last checkpoint found in OUTPUT_DIR, or from scratch if there are no checkpoints (default: true)

CUDNN_BENCHMARK: boolean that indicates whether to enable cudnn benchmark mode for using more optimized kernels. (default: true)

NUM_GPUS: number of GPUs to use. (default: 8)

AMP:if set to `true`, enables automatic mixed precision (default: false)

BATCH_SIZE: effective data batch size. The real batch size per GPU might be lower, if gradient accumulation is enabled (default: 64)

GRAD_ACCUMULATION_STEPS: number of gradient accumulation steps until optimizer updates weights. (default: 2)

LEARNING_RATE: initial learning rate. (default: 0.01)

MIN_LEARNING_RATE: minimum learning rate, despite LR scheduling (default: 1e-5)

LR_POLICY: how to decay LR (default: exponential)

LR_EXP_GAMMA: decay factor for the exponential LR schedule (default: 0.981)

EMA: decay factor for exponential averages of checkpoints (default: 0.999)

SEED: seed for random number generator and used for ensuring reproducibility. (default: 0)

EPOCHS: number of training epochs. (default: 440)

WARMUP_EPOCHS: number of initial epoch of linearly increasing LR. (default: 2)
```

```
HOLD_EPOCHS: number of epochs to hold maximum LR after warmup. (default: 140)

SAVE_FREQUENCY: number of epochs between saving the model to disk. (default: 10)

EPOCHS_THIS_JOB: run training for this number of epochs. Does not affect LR schedule like the EPOCHS parameter. (default: 0)

DALI_DEVICE:device to run the DALI pipeline on for calculation of filterbanks. Valid choices: cpu, gpu, none. (default: gpu)

PAD_TO_MAX_DURATION: pad all sequences with zeros to maximum length. (default: false)

EVAL_FREQUENCY: number of steps between evaluations on the validation set. (default: 544)

PREDICTION_FREQUENCY: the number of steps between writing a sample prediction to stdout. (default: 544)

TRAIN_MANIFESTS:lists of .json training set files

VAL_MANIFESTS: lists of .json validation set files
```

Theo mặc định, độ chính xác tự động bị vô hiệu hóa, kích thước lô là 64 qua hai bước tích lũy gradient và công thức được chạy trên tổng cộng 8 GPU. Các siêu tham số được điều chỉnh cho GPU có ít nhất 32GB bộ nhớ và sẽ yêu cầu điều chỉnh cho GPU 16GB (ví dụ: bằng cách giảm kích thước lô và sử dụng nhiều bước tích lũy gradient hơn).

Các tùy chọn đang được truyền dưới dạng biến môi trường. Có thể tìm thấy thêm thông tin chi tiết về các tùy chọn khả dụng trong Quy trình tham số và đào tạo.

- Quá trình huấn luyện:

Quá trình đào tạo được thực hiện bằng tập lệnh train.py cùng với các tham số được xác định trong scripts/train.sh Tập lệnh scripts/train.sh chạy một tác vụ trên một nút duy nhất để đào tạo mô hình Jasper từ đầu bằng cách sử dụng VIVOS làm dữ liệu đào tạo. Để đào tạo hiệu quả hơn, chúng tôi loại bỏ các mẫu âm thanh dài hơn 16,7 giây khỏi tập dữ liệu đào tạo, tổng số các mẫu này ít hơn

1%. Việc lọc như vậy không làm giảm độ chính xác, nhưng nó cho phép chúng tôi giảm số bước thời gian trong một đợt, điều này yêu cầu ít bộ nhớ GPU hơn và tăng tốc độ đào tạo. Ngoài các đối số mặc định được liệt kê trong phần Tham số, theo mặc định, tập lệnh đào tạo:[7]

- Chạy trên 8 GPU với ít nhất 32GB bộ nhớ và kích thước lô đào tạo/đánh giá là 64, chia thành hai bước tích lũy gradient
- Sử dụng độ chính xác TF32 (GPU A100) hoặc FP32 (các GPU khác)
- Đào tạo trên sự nối kết của tất cả 3 tập dữ liệu đào tạo VIVOS và đánh giá trên tập dữ liệu dev-clean VIVOS
- Duy trì trung bình động theo cấp số nhân của các tham số để đánh giá
- Đã bật chuẩn cudnn
- Chạy trong 440 epochs
- Sử dụng tốc độ học ban đầu là 0,01 và tốc độ học giảm theo cấp số nhân
- Lưu một điểm kiểm tra sau mỗi 10 epochs
- Tự động xóa các điểm kiểm tra cũ và bảo toàn các điểm kiểm tra mốc
- Chạy đánh giá trên tập dữ liệu phát triển sau mỗi 544 lần lặp và khi kết thúc quá trình đào tạo
- Duy trì một điểm kiểm tra riêng biệt với WER thấp nhất trên tập phát triển
- In ra tiến trình đào tạo sau mỗi lần lặp đến stdout
- Tạo tệp nhật ký DLLogger và nhật ký Tensorboard
- Tính toán nhiễu loạn tốc độ trực tuyến trong quá trình đào tạo
- Sử dụng SpecAugment trong quá trình xử lý trước dữ liệu
- Lọc các mẫu âm thanh dài hơn 16,7 giây
- Bổ sung từng lô để độ dài của nó chia hết cho 16
- Sử dụng phép tích chập che và phần dư dày đặc như mô tả trong bài báo
- Sử dụng suy giảm trọng số 0,001
- Sử dụng Novograd làm trình tối ưu hóa với beta = (0,95, 0)

3.2.4 Đánh giá mô hình

Bên trong container, sử dụng tập lệnh sau để chạy đánh giá. Đảm bảo tập dữ liệu đã tải xuống và được xử lý trước nằm tại <DATA_DIR>/LibriSpeech trên máy chủ, tương ứng với /datasets/LibriSpeech bên trong container.

```
[OPTION1=value1 OPTION2=value2 ...] bash scripts/evaluation.sh [OPTIONS]
```

- Quá trình đánh giá

- Đánh giá thử nghiệm VIVOS-bộ dữ liệu khác
- Đầu ra mô hình

3.3 Tích hợp mô hình lên web

3.3.1 Giới thiệu Flask

Flask là một micro web framework cho Python, được phát triển bởi Armin Ronacher của Pocoo, một nhóm phát triển phần mềm quốc tế. Được ra mắt lần đầu vào năm 2010, Flask nhanh chóng trở thành một trong những framework phổ biến nhất nhờ vào tính linh hoạt và dễ sử dụng. Flask được thiết kế với một lõi đơn giản, nhưng có thể mở rộng thông qua các module và tiện ích bổ sung, cho phép các nhà phát triển xây dựng ứng dụng từ các ứng dụng nhỏ gọn đến các hệ thống phức tạp. Một trong những ưu điểm lớn của Flask là khả năng tùy biến cao, cho phép nhà phát triển tự do trong việc cấu trúc và thiết kế ứng dụng theo nhu cầu cụ thể của họ. Flask không đi kèm với các công cụ hoặc thư viện mặc định cho các chức năng như quản lý cơ sở dữ liệu hoặc xác thực người dùng, điều này giúp giảm thiểu sự ràng buộc và cho phép lựa chọn những công cụ phù hợp nhất cho dự án.

Một điểm mạnh khác của Flask là cộng đồng người dùng và tài liệu phong phú, cung cấp sự hỗ trợ mạnh mẽ cho những người mới bắt đầu cũng như các nhà phát triển giàu kinh nghiệm. Flask sử dụng WSGI (Web Server Gateway Interface) cho việc xử lý các yêu cầu web, và Jinja2 làm hệ thống template mặc

định. Flask cũng tích hợp tốt với các công cụ và thư viện khác của Python, tạo điều kiện thuận lợi cho việc phát triển các ứng dụng web mạnh mẽ và hiệu quả. Với tính năng mở rộng thông qua các blueprint, Flask cho phép phân chia ứng dụng thành các module nhỏ, giúp quản lý và bảo trì dễ dàng hơn. Ngoài ra, Flask cũng hỗ trợ tốt việc kiểm thử, giúp các nhà phát triển đảm bảo chất lượng và tính ổn định của ứng dụng trước khi triển khai. Flask thực sự là một lựa chọn lý tưởng cho những ai muốn xây dựng ứng dụng web từ các dự án nhỏ cho đến các hệ thống phức tạp với sự linh hoạt và hiệu quả.

Hình 3.3. Thư viện Flask

3.3.2 Cài đặt

Tải source code:

Cập nhật và cài đặt Linux libs:

```
apt-get update && apt-get install -y libsndfile1 ffmpeg
```

Cài đặt các thư viện cần thiết sau:

```
Cython

wheel

numpy==1.23

flask

requests

ruamel.yaml

frozendict
```

```
torch_stft

python-datautil

wrapt

wget

kaldi_io

pandas

inflect

unidecode

pyctcdecode

Flask-SocketIO==4.3.1

python-engineio==3.13.2

python-socketio==4.6.0

Flask==2.0.3

Werkzeug==2.0.3
```

Cài đặt Torch 1.8.1:

```
# cpu only, you can install CUDA version if you have NVidia GPU

pip install torch==1.8.1+cpu torchvision==0.9.1+cpu torchaudio==0.8.1 -f https://download.pytorch.org/whl/torch_stable.html
```

Chạy ứng dụng:

```
python app.py # app will run on address: https://localhost:5000
```

CHƯƠNG IV: KẾT LUẬN

4.1 Đánh giá mô hình học sâu

4.1.1 Chỉ số Word error rate (WER)

Tỷ lệ lỗi từ (WER) là một phép đo phổ biến về hiệu suất của hệ thống nhận dạng giọng nói hoặc dịch máy. Khó khăn chung khi đo lường hiệu suất nằm ở thực tế là chuỗi từ được nhận dạng có thể có độ dài khác với chuỗi từ tham chiếu (được cho là chuỗi đúng). WER bắt nguồn từ khoảng cách Levenshtein, hoạt động ở cấp độ từ thay vì cấp độ âm vị. WER là một công cụ có giá trị để so sánh các hệ thống khác nhau cũng như để đánh giá các cải tiến trong một hệ thống. Tuy nhiên, loại phép đo này không cung cấp thông tin chi tiết về bản chất của lỗi dịch và do đó cần phải làm việc thêm để xác định nguồn lỗi chính và tập trung mọi nỗ lực nghiên cứu. Vấn đề này được giải quyết bằng cách đầu tiên căn chỉnh chuỗi từ được nhận dạng với chuỗi từ tham chiếu (nói) bằng cách căn chỉnh chuỗi động. Việc kiểm tra vấn đề này được xem xét thông qua một lý thuyết gọi là luật lũy thừa nêu mối tương quan giữa sự bối rối và tỷ lệ lỗi từ.

Tỷ lệ lỗi từ sau đó có thể được tính như sau:

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}$$

trong đó

S là số từ thay thế,

D là số từ xóa,

I là số từ chèn,

C là số từ đúng,

N là số từ trong tham chiếu (N=S+D+C)

Thực tế đằng sau 'xóa' và 'chèn' là cách đi từ tham chiếu đến giả thuyết. Vì vậy, nếu chúng ta có tham chiếu "Đây là wikipedia" và giả thuyết "Đây _ wikipedia", chúng ta gọi đó là xóa.

4.1.2 Kết quả đánh giá mô hình trên một số cấu hình máy

Đánh giá độ chính xác của mô hình trên những cấu hình máy khác nhau thông qua chỉ số WER

Number of GPUs	**Batch size per GPU**	**Precision**	**dev-clean WER**	**dev-other WER**	**test-clean WER**	**test-other WER**	**Time to train**
8	64	mixed	3.20	9.78	3.41	9.71	70 h

Bảng 4.1. Mô hình huấn luyện trên GPU ***NVIDIA DGX A100 (8x A100 80GB)***

Number of GPUs	**Batch size per GPU**	**Precision**	**dev-clean WER**	**dev-other WER**	**test-clean WER**	**test-other WER**	**Time to train**
8	64	mixed	3.26	10.00	3.54	9.80	130 h

Bảng 4.2. Mô hình huấn luyện trên GPU ***NVIDIA DGX-1 (8x V100 32GB)***

Bảng dưới đây so sánh tỷ lệ lỗi qua các seed dữ liệu training khác nhau nhằm cho thấy tính ổn định của mô hình

DGX A100 80GB, FP16, 8x GPU	**Seed #1**	**Seed #2**	**Seed #3**	**Seed #4**	**Seed #5**	**Seed #6**	**Seed #7**	**Seed #8**	**Mean**	**Std**
dev-clean	3.46	3.55	3.45	3.44	3.25	3.34	3.20	3.40	3.39	0.11
dev-other	10.30	10.77	10.36	10.26	9.99	10.18	9.78	10.32	10.25	0.27
test-clean	3.84	3.81	3.66	3.64	3.58	3.55	3.41	3.73	3.65	0.13
test-other	10.61	10.52	10.49	10.47	9.89	10.09	9.71	10.26	10.26	0.31

*Bảng 4.3. Mô hình huấn luyện trên GPU **DGX A100 80GB, FP16, 8x GPU***

DGX-1 32GB, FP16, 8x GPU	**Seed #1**	**Seed #2**	**Seed #3**	**Seed #4**	**Seed #5**	**Seed #6**	**Seed #7**	**Seed #8**	**Mean**	**Std**
dev-clean	3.31	3.31	3.26	3.44	3.40	3.35	3.36	3.28	3.34	0.06
dev-other	10.02	10.01	10.00	10.06	10.05	10.03	10.10	10.04	10.04	0.03
test-clean	3.49	3.50	3.54	3.61	3.57	3.58	3.48	3.51	3.54	0.04
test-other	10.11	10.14	9.80	10.09	10.17	9.99	9.86	10.00	10.02	0.13

*Bảng 4.4. Mô hình huấn luyện trên GPU **DGX-1 32GB, FP16, 8x GPU***

4.2 Kết quả đạt được

4.2.1 Kết quả huấn luyện mô hình trên cấu hình máy cá nhân

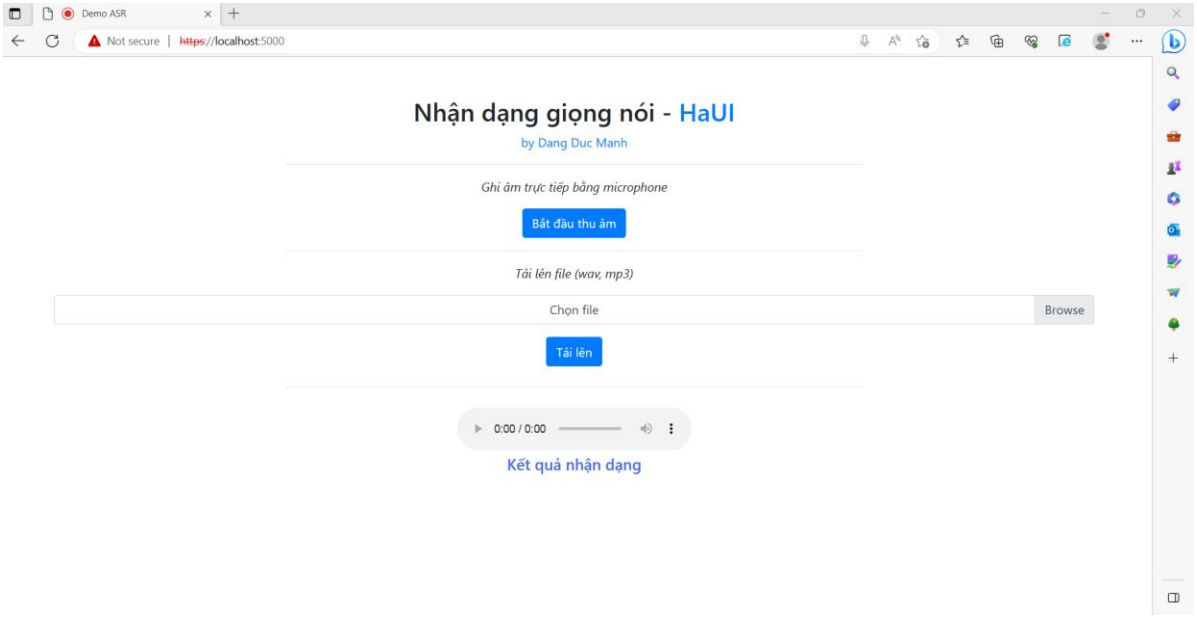
Dưới đây là môi trường thử nghiệm mô hình Jasper. Mô hình được cài đặt và huấn luyện trên máy tính cá nhân có thông số như sau

Device name	MCComputer
Processor	Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz 2.30 GHz
Installed RAM	16.0 GB (15.9 GB usable)
Device ID	B65032DB-F705-414B-A2D8-1BC3B257FA30
Product ID	00331-10000-00001-AA133
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

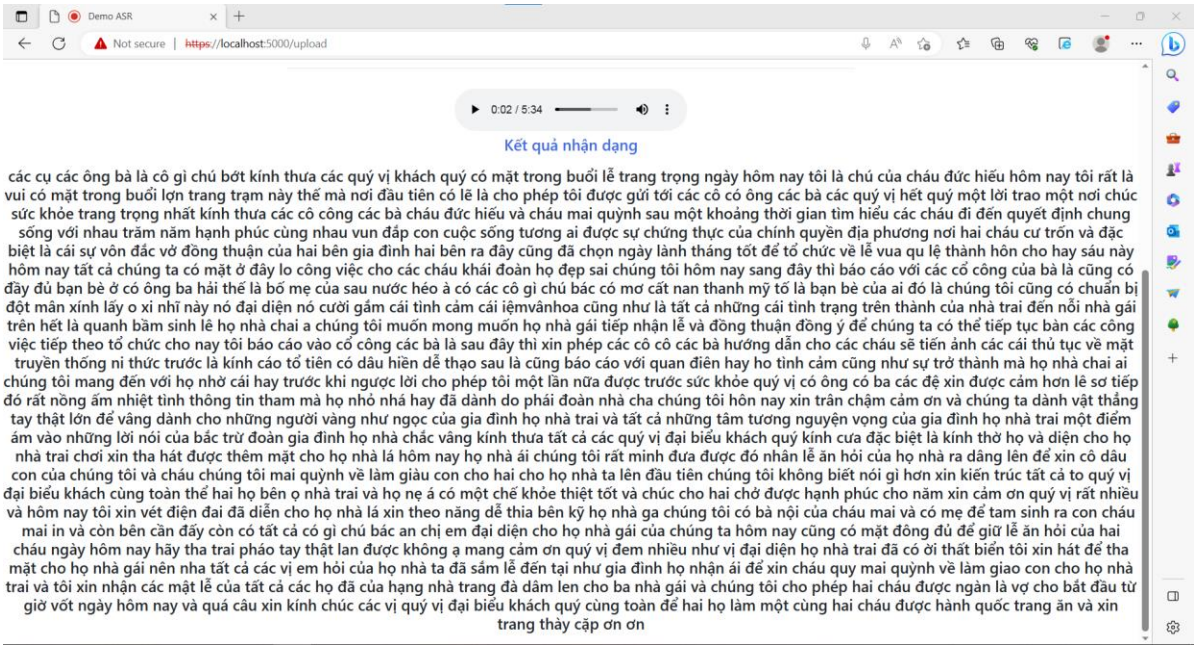
Số GPU	Số batch /GPU	Kiểu đánh giá	dev-clean WER	dev-other WER	test-clean WER	test-other WER	Thời gian huấn luyện
1 CPU	16	mixed	12.20	20.08	6.20	18.72	180h

Bảng 4.5. Mô hình huấn luyện trên GPU ***NVIDIA GTX-1050 (4GB RAM)***

4.2.2 Một số hình ảnh của ứng dụng



***Hình 4.1**. Giao diện của ứng dụng*



***Hình 4.2.** Ứng dụng trả về kết quả nhận dạng giọng nói*

4.3 Hướng phát triển

Để cải thiện ứng dụng chuyển giọng nói thành văn bản, có thể tập trung vào ba hướng phát triển chính: cải thiện độ chính xác và nhận diện ngữ cảnh, tăng

cường khả năng xử lý ngôn ngữ địa phương và ngôn ngữ khác nhau, cùng cải thiện giao diện người dùng và tích hợp công nghệ.

Cải thiện độ chính xác và nhận diện ngữ cảnh: Một trong những thách thức lớn nhất của công nghệ chuyển giọng nói thành văn bản là đảm bảo độ chính xác cao, đặc biệt khi đối diện với các giọng nói và ngữ cảnh khác nhau. Sử dụng các mô hình học sâu như Transformer và BERT có thể giúp nâng cao khả năng nhận diện và xử lý ngôn ngữ tự nhiên. Các mô hình này có khả năng học và hiểu ngữ cảnh tốt hơn, từ đó tăng cường độ chính xác trong việc nhận diện từ ngữ và ngữ pháp. Bên cạnh đó, việc huấn luyện các mô hình này trên một tập dữ liệu đa dạng, bao gồm nhiều loại giọng nói, ngữ điệu, và tình huống khác nhau, cũng là một yếu tố quan trọng. Điều này giúp mô hình học được các đặc điểm khác nhau của giọng nói và ngữ cảnh, từ đó cải thiện khả năng nhận diện. Cuối cùng, việc phát triển khả năng hiểu ngữ cảnh không chỉ giúp chuyển giọng nói thành văn bản chính xác hơn mà còn giúp mô hình có thể dịch nghĩa tốt hơn trong các lĩnh vực chuyên ngành, từ y tế, luật pháp đến kỹ thuật.

Tăng cường khả năng xử lý ngôn ngữ địa phương và ngôn ngữ khác nhau: Việc hỗ trợ nhiều ngôn ngữ và giọng địa phương là một hướng phát triển quan trọng để mở rộng phạm vi ứng dụng của công nghệ chuyển giọng nói thành văn bản. Điều này không chỉ giúp tăng cường trải nghiệm người dùng mà còn đáp ứng nhu cầu của người dùng từ các vùng miền và quốc gia khác nhau. Bằng cách thu thập dữ liệu giọng nói từ nhiều nguồn và nhiều ngôn ngữ, các nhà phát triển có thể huấn luyện mô hình để nhận diện và chuyển đổi giọng nói của nhiều ngôn ngữ và giọng địa phương khác nhau. Ngoài ra, việc tùy chỉnh mô hình cho từng người dùng cụ thể cũng là một yếu tố quan trọng. Sử dụng các kỹ thuật học máy, mô hình có thể được điều chỉnh và cải thiện theo giọng nói của từng người dùng, giúp tăng cường độ chính xác và trải nghiệm cá nhân hóa.

Cải thiện giao diện người dùng và tích hợp công nghệ: Một ứng dụng thành công không chỉ dựa trên công nghệ mạnh mẽ mà còn cần có giao diện thân thiện và dễ sử dụng. Thiết kế giao diện người dùng trực quan, dễ thao tác và tương tác là yếu tố quan trọng để người dùng có thể dễ dàng sử dụng ứng dụng. Điều này bao gồm việc cho phép người dùng dễ dàng hiệu chỉnh văn bản sau khi chuyển đổi, giúp đảm bảo văn bản cuối cùng là chính xác và phù hợp. Bên cạnh đó, khả năng tích hợp với các ứng dụng và dịch vụ khác cũng là một yếu tố quan trọng. Việc tích hợp dễ dàng với các ứng dụng như email, trình soạn thảo văn bản, và các dịch vụ đám mây giúp người dùng có thể sử dụng công nghệ chuyển giọng nói thành văn bản một cách linh hoạt và hiệu quả trong nhiều tình huống và môi trường khác nhau. Điều này không chỉ giúp tăng cường tính ứng dụng mà còn mở rộng phạm vi sử dụng của công nghệ này.

TÀI LIỆU THAM KHẢO

1. Nguyễn Đăng Hải, Nguyễn Gia Huy, Phạm Minh Hoàng, Đỗ Đức Hào, Phát hiện giọng nói với mạng kết hợp CNN-BiLSTM
2. Aurélien Géron (2019), "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition"
3. Afshine Amidi and Shervine Amidi, "Deep Learning cheatsheet"
4. Afshine Amidi and Shervine Amidi, "Recurrent Neural Networks cheatsheet"
5. Jason Li1 , Vitaly Lavrukhin1 , Boris Ginsburg1 , Ryan Leary1 , Oleksii Kuchaiev1 , Jonathan M. Cohen1 , Huyen Nguyen1 , Ravi Teja Gadde2 ," Jasper: An End-to-End Convolutional Neural Acoustic Model"
6. C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch normalized recurrent neural networks," in 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), March 2016, pp. 2657–2661.
7. G. Hinton et al., "Deep neural networks for acoustic modeling in speech recognition," IEEE Signal Processing Magazine, 2012.
8. R. Collobert, C. Puhrsch, and G. Synnaeve, "Wav2letter: an endto-end convnet-based speech recognition system," arXiv preprint arXiv:1609.03193, 2016.
9. Y. Bengio, R. De Mori, G. Flammia, and R. Kompe, "Global optimization of a neural network-hidden markov model hybrid," IEEE Transactions on Neural Networks, 3(2), 252259, 1992.
10. E. Battenberg et al., "Exploring neural transducers for end-to-end speech recognition," in 2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Dec 2017, pp. 206–213.
11. Mohamed Reda Bouadjenek and Ngoc Dung Huynh (2021)," Automatic Speech Recognition using CT