A project report on

ACCENT RECOGNTION USING DEEP LEARNING TECHNIQUES

Submitted in partial fulfillment for the award of the degree of

B.Tech (Information Technology)

By

ADITYA KOCHERLAKOTA (19BIT0386)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF INFORMATION TECHNOLOGY & ENGINEERING

January, 2023

DECLARATION

I here by declare that the thesis entitled “ACCENT RECOGNITION USING DEEP LEARNING TECHNIQUES” submitted by me, for the award of the degree of M.Tech. (Software Engineering) is a record of bonafide work carried out by me under the supervision of Prof. Valarmathi B

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 26-01-2023 Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled "ACCENT RECOGNITION USING DEEP LEARNING TECHNIQUES" submitted by ADITYA KOCHERLAKOTA (19BIT0386), School of Information Technology & Engineering, Vellore Institute of Technology, Vellore for the award of the degree B.Tech (Software Engineering) is a record of bonafide work carried out by him/her under my supervision.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The Project report fulfils the requirements and regulations of VELLORE INSTITUTE OF TECHNOLOGY, VELLORE and in my opinion meets the necessary standards for submission.

Signature of the Guide **Signature of the HoD**

Internal Examiner **External Examiner**

ABSTRACT

Accents refer to a distinctive way of pronouncing a language, especially one associated with a particular country, area, or social class. They typically share characteristics with persons from similar racial, cultural, or social backgrounds. Speech classification is the process of identifying the regional accents of one's fellow people. Speech classification makes it simple to determine a person's ethnicity, personal geography, and religious upbringing. Obviously, if it is done manually, mistakes will occur. Some accents or languages that scarcely anyone speaks or has ever heard of are occasionally mentioned. Since most people don't get it and can't decipher its genuine meaning, this presents a dilemma. This technology can also improve research into speech recognition. To fully utilize the power of convolutional neural network systems previous works have used the concept of a spectrogram image, this allows the audio data to be passed as two-dimensional data and allows the convolutional neural network to improve feature extraction. Previous works[1] have achieved an accuracy of 92.9% whilst using spectrograms. This paper works to take the temporal aspect into account by employing a hybrid CNN-LSTM model and also eschewing transfer learning to gauge the strength of each algorithm rather than relying on pre trained models. This paper has compared the performance of certain image classification techniques i.e. Multi Layered Perceptrons with accuracy of 24.5%, Convolutional Neural Networks with an accuracy of 80.1%, Hybrid CNN-LSTM with an accuracy of 90.2% , and Hybrid CNN-GRU with an accuracy of 94.2% . This work also the difference in performance for each algorithm by using multiple datasets, namely OpenSLR and AccentDB

Keywords: Deep learning, MLP, CNN, LSTM, Vision Transformers, Accent classification

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. Valarmathi B, Associate Professor Senior, School of Information Technology & Engineering, Vellore Institute of Technology, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Machine learning.

I would like to express my gratitude to DR.G.VISWANATHAN, Chancellor VELLORE INSTITUTE OF TECHNOLOGY, VELLORE, MR. SANKAR VISWANATHAN, DR. SEKAR VISWANATHAN, MR.G V SELVAM, Vice – Presidents VELLORE INSTITUTE OF TECHNOLOGY, VELLORE, DR. RAMBABU KODALI, Vice – Chancellor, DR. PARTHA SARATHI MALLICK, Pro-Vice Chancellor and Dr. S. Sumathy, Dean, School of Information Technology & Engineering (SITE), for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Usha Devi G., HoD/Professor, all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Vellore

Date: 26-01-2023 Aditya Kocherlaktoa

TABLE OF CONTENTS

CHAPTER 4

IMPLEMENTATION & TESTING

CHAPTER 5

LIST OF ABBREVIATIONS

MLP : Multi layered Perceptrons

CNN: Convolutional Neural Networks

ViT : Vision Transformers

LSTM : Long Short-term Memory

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 BACKGROUND

Accent classification is identifying and classifying a speaker's accent according to their pronunciation and speech patterns. This is often accomplished by examining the phonetic aspects of a speaker's speech, such as stress patterns, rhythm, intonation, and pronunciation of specific sounds. Accent classification is useful for a wide range of tasks, including speech recognition, language instruction, and linguistics research. Accents can be categorised based on geographical, social, or linguistic characteristics, such as the speaker's original language or region, nation, or ethnic origin.

A speaker's accent is influenced by a variety of circumstances, including their linguistic background, life experiences, and exposure to various languages and accents. Classifying an accent is a complex undertaking that can be difficult. In addition to conventional linguistic analysis techniques, researchers frequently combine them with machine learning algorithms that are trained on vast databases of speech samples to reliably categorise accents.

1.2 MOTIVATION

Accent categorization research can be used to improve speech recognition technology for people with non-standard accents or to better comprehend the linguistic diversity of a population. Accent recognition and detection is a major research subject in speech processing that may be utilized to improve automatic speech recognition.

1.3 A REVIEW OF OTHER WORKS

1.3.1 Deep Learning Techniques

Convolutional Neural Networks

In this study [1], regional accents of British English were recognized for both gender-independent and gender-dependent experiments using a convolutional neural network. Many different acoustic features were used in the studies. While there is still no generally accepted feature set, the selection of handcrafted features is a challenging task. Moreover, converting audio signals into images in the most appropriate way is critical for a convolutional neural network, a deep learning model commonly used in image applications. To take advantage of the convolutional neural networks' ability to characterize two-dimensional signals, spectrogram image features that visualize the speech signal frequency distribution were used. The accuracy of 92.92 and 93.38% and the F-score of 92.67 and 93.19% were obtained for gender-independent and gender-dependent experiments, respectively [1]. In this paper [5], they developed a model that utilized machine learning and deep learning algorithms to create a tool for accent classification. The proposed system was validated using the Speech Accent Archive dataset hosted by George Mason University on Kaggle. Results showed that among various machine learning models, Decision Trees performed the best, even when compared to deep learning models like CNN and LSTM, with an accuracy of 97.36%. In this work [9], these five features were used to train a 2-layer Convolution Neural Network on a dataset of five distinct language-accents, namely Arabic, English, French, Mandarin, and Spanish. The accuracy of each feature was evaluated and compared, and MFCC yielded the highest accuracy. The paper [27] presented a novel architecture for accent identification by using a cascade of two deep-learning architecture. They designed and testde their proposed architecture on common voice dataset. The architecture consisted of a cascade of Convolutional Neural Network (CNN) and Convolutional Recurrent Neural Network (CRNN). It was trained on Mel-spectrogram of the audios. They considered five of the most popular English accents groups namely India, Australia, US, England, Canada in this study. The proposed model has an accuracy of 78.48% using CNN and 83.21% using CRNN. In a recent study [29], researchers aimed to improve the accent recognition task for up to nine European accents in English by exploring specific hyperparameter choices for neural network models and searching for the best input speech signal parameters. They used a CNN-based model trained on the Speech Accent Archive dataset, which contains recordings of accented speech collected from crowdsourcing. The researchers discovered that combining time-frequency and energy features, such as spectrogram, chromogram, spectral centroid, spectral rolloff, and fundamental frequency, with the Mel-frequency cepstral coefficients (MFCC) could increase the accuracy of accent classification compared to using MFCC and/or raw spectrograms. They found that amplitude mel-spectrograms on a linear scale produced the most impact and achieved state-of-the-art classification results with recognition accuracy ranging from 0.964 to 0.987 for English with Germanic, Romance, and Slavic accents, surpassing previous

models using the Speech Accent Archive. The study also explored the effect of speech rhythm on recognition accuracy and found that using the original audio recordings with all pauses preserved produced the best results for other accent classification experiments.

Long short-term memory

In this paper [5], they developed a model that utilized machine learning and deep learning algorithms to create a tool for accent classification. The Speech Accent Archive dataset hosted by George Mason University on Kaggle was used to validate a proposed system's performance. The results indicated that Decision Trees outperformed both deep learning models (CNN and LSTM) and achieved the highest accuracy rate of 97.36%. While LSTM did perform slightly better than CNN, the difference was not significant. Overall, the study revealed that Decision Trees are a viable option for speech recognition tasks, even when compared to more advanced deep learning models. In order to improve the robustness of speech recognition systems, this study [28] attempts to classify stressed speech caused by the psychological stress under multitasking workloads. Due to the transient nature and ambiguity of stressed speech, the stress characteristics is not represented in all the segments in stressed speech as labeled. In this paper, we propose a multi-feature fusion model based on the attention mechanism to measure the importance of segments for stress classification. Through the attention mechanism, each speech frame is weighted to reflect the different correlations to the actual stressed state, and the multi-channel fusion of features characterizing the stressed speech to classify the speech under stress. The proposed model further adopts SpecAugment in view of the feature spectrum for data augment to resolve small sample sizes problem among stressed speech. During the experiment, we compared the proposed model with traditional methods on CASIA Chinese emotion corpus and Fujitsu stressed speech corpus, and results show that the proposed model has better performance in speaker-independent stress classification. Transfer learning is also performed for speaker-dependent classification for stressed speech, and the performance is improved. The attention mechanism shows the advantage for continuous speech under stress in authentic context comparing with traditional methods.

Deep Belief Networks

In this paper [4], they presented an effective classifier for foreign accented spoken English to determine the speaker's origin. They created an accented spoken English corpus consisting of 30 speakers from 6 different countries including China, India, France, Germany, Turkey and Spain. They used MFCC as a feature and a Deep Belief Network (DBN) as a classifier. The parameters of the DBN were determined by an iterative approach in which the node weight was updated according to the substitution error. Their method achieved an accuracy of 90.2% for 2 accented

datasets and 71.9% for 6 accented datasets. The result was much better than other state-of-the-art methods such as SVM, k-NN, and random forest which had around 40% accuracy.

Variable Filter Net

In this paper[11], they presented VFNet (Variable Filter Net), a Convolutional Neural Network (CNN) based architecture that captured a hierarchy of features to beat the previous benchmarks of accent classification, through a technique of applying variable filter sizes along the frequency band of the audio utterances. CNN and VFN are both deep learning models used in image and signal processing tasks. While CNN uses a fixed filter size to detect features, VFN employs a variable filter size that adapts to the size of the feature. This results in more efficient use of computational resources and enables VFN to detect both small and large features. Additionally, VFN can learn from smaller datasets compared to CNN, which requires a large dataset to learn features effectively. While both models have their strengths and weaknesses, the choice between them depends on the specific needs and resources of the task at hand.

Multi-layered Perceptrons

In this study[12], 7 classification type ML algorithms were used, including Multilayer Perceptron (MLP), Random Forest (RF), Decision Tree (DT), Radial Basis Function (RBF), k-Nearest Neighbor (k-NN), Naive Bayes (NB), and Logistic Model Tree (LMT) methods used for Speaker Accent Recognition data set on UCI ML Repository. Performance metrics were compared using accuracy, Kappa Statistics, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE), and were compared for each algorithm. In the paper [19], three accents in English language recorded from three main ethnicities in Malaysia (Malay, Chinese and Indian) were classified using an artificial neural network model. All experiments were performed in speaker-independent and three most accent-sensitive words-independent modes. Mel-bands spectral energy was extracted from eighteen bands, taking the statistical values (mean, standard deviation, kurtosis and the ratio of standard deviation to kurtosis) to characterize the spectral energy distribution. The system was evaluated using an independent test dataset, partial-independent test dataset, and training dataset. The best three-class accuracy rate of 99.01% was obtained with the independent test dataset. The overall accuracy rate was averaged to 96.79% with the average learning time at 49 epochs. In [26], the study explored a new focus of research in the field of accent classification from speech, using emotional speech data comprising six emotions in three English accents. Different standard features and classifiers were used for analysis, including PLP, LFBE, MFCC, SSC, CQCC, chroma, MFDWC-A, MFDWC-D, and MFDWC-AD. The study found that Mel-scale based SSC features achieved the highest testing accuracy of up to 94.21% for clean speech, while wavelet-based combined MFDWC-AD features performed well for all cases at low signal-to-noise levels. The study suggested that using

SSC or MFDWC-AD features could be useful in a two-stage approach for speech emotion recognition or automatic speech recognition where accent identification is the first stage.

1.3.2 Machine Learning Techniques

Hidden Markov Model

In this paper [2], an automatic classification system for foreign accents in Australian English (AuE) speech based on accent-dependent parallel phoneme recognition (PPR) was developed. The classifier was designed to process continuous speech and discriminate between native AuE speakers and two migrant speaker groups with foreign accents, whose first languages were Lebanese Arabic (LA) and South Vietnamese (SV). The training of the system was automated and novel in that it did not require manually labeled accented data. The test utterances were processed in parallel by three (AuE, SV and LA) accent-specific recognizers incorporating accent-specific HMMs and phoneme bigram language models to produce accent discrimination likelihood scores. The best average accent classification rates were 85.3% and 76.6% for accent-pair and three-accent class discrimination tasks, respectively. Analyses of the contributions to accent discrimination by phoneme-level processing and the language model were described.

Maximization of mutual information

In this paper [3], they extended language identification (LID) techniques to a large scale accent classification task: 23-way classification of foreign-accented English. They found that a purely acoustic approach based on a combination of heteroscedastic linear discriminant analysis (HLDA) and maximum mutual information (MMI) training was very effective. In contrast to LID tasks, methods based on parallel language models proved much less effective. They focused on the Oregon Graduate Institute Foreign-Accented English dataset and obtained a detection rate of 32%, which was the best reported result for 23-way accent classification, to their knowledge.

Fuzzy Logic

In this paper[6], a new method based on the fuzzy canonical correlation-based Gaussian classifier was proposed. The membership values of the clusters were based on not only minimizing the distance between the cluster centroids but also maximizing the out-of-class variations. The model Achieved an accuracy of 89.03%

Gaussian Mixture Model

This paper[7] presented a new speaker classification scheme based on Australian accents which are broad, general and cultivated. Speakers were classified into speaker groups according to their accents, ages, and genders. Mel-frequency cepstral coefficients extracted after speech processing were used to build Gaussian speaker group mixture models. The fusion of speaker group classifiers was then performed. Experiments showed high performance for the proposed method. In this paper [8], they distinguished between standard American English and Indian Accented English using the second and third formant frequencies of specific accent markers. A GMM classification was used on the feature set for each accent group. The results showed that using just the formant frequencies of these accent markers was sufficient to achieve a suitable classification for these two accent groups. In this work[14], they combined phonetic knowledge, such as vowels, with enhanced acoustic features to build an improved accent classification system. The classifier was based on Gaussian Mixture Model-Universal Background Model (GMM-UBM), with normalized Perceptual Linear Predictive (PLP) features. The features were further optimized by Principle Component Analysis (PCA) and Hetroscedastic Linear Discriminant Analysis (HLDA). Using seven major types of accented speech from the Foreign Accented English (FAE) corpus, the system achieved a classification accuracy of 54% with input test data as short as 20 seconds, which was competitive with the state of the art in this field. In this paper[13], they presented an automated approach for text-independent foreign accent classification. Results of foreign accent classification were used for adapting acoustic models, modifying lexicon, and changing language model with regards to non-native speakers. In their study, they investigated statistical approaches that differed from the a priori knowledge they needed: GMM, which required neither phonetic knowledge nor labeling, phone recognition (without a lexicon), and sentence recognition (with a lexicon and a grammar). This work was done in the framework of the HIWIRE (Human Input that Works In Real Environment) European project. They evaluated the proposed approaches on an English speech corpus pronounced by French, Italian, and Greek speakers. All experiments were performed in speaker-independent and text-independent mode. The best classification rate (83.3%) was achieved using sentence recognition or a phone-based recognition followed by language modeling approach. A method of classification of speakers, based on the regional English accent, is proposed in the paper [30]. A database of English speech, spoken by the native speakers of three closely related Dravidian languages, was collected from a non-overlapping set of speakers, along with the native language speech data. Native speech samples from speakers of the regional languages of India, namely Kannada, Tamil, and Telugu are used for the training set. The testing set contains utterances of non-native English speakers of compatriots of the above three groups. Automatic identification of native language is proposed by using the spectral features of the non-native speech, that are classified using the classifiers such as Gaussian Mixture Models (GMM), GMM-Universal Background Model (GMM-UBM), and i-vector. Identification accuracy of 87.9 % was obtained using the GMM classifier, which was increased to 90.9 % by using the GMM-UBM method. But the i-vector-based approach gave a better accuracy of 93.9 % , along with EER of

6.1 % . The results obtained are encouraging, especially viewing the current state-of-the-art accuracies around 85 % . It is observed that the identification rate of nativity, while speaking English, is relatively higher at 95.2 % for the speakers of Kannada language, as compared to that for the speakers of Tamil or Telugu as their native language.

Random Forest

The paper[10] addressed novel advances in English dialect/accent classification/identification. A word-level based modeling technique was proposed that was shown to outperform an LVCSR-based system with significantly less computational cost. The new algorithm, named WDC (word-based dialect classification), converted the text-independent decision problem into a text-dependent problem and produced multiple combination decisions at the word level rather than making a single decision at the utterance level. There were two sets of classifiers employed for WDC: word classifier, and utterance classifierwas boosted via the real AdaBoost.MH algorithm in the probability space directly instead of the feature space was boosted via the dialect dependency information of the words. Two dialect corpora were used in the evaluation and significant improvement in dialect classification was achieved for both corpora. The Paper [12] had also used Random Forest as on of the algorithms in their experiments, and fond satisfactory results although pailed in comparison to other algorithms

Radial Basis Functions

This study[17] aimed to recognize the American accent among six accents: American, British, French, German, Italian, and Spanish. Two different machine learning-based algorithms were employed for the accent recognition task: Support Vector Machine (SVM) and k-Nearest Neighbor (k-NN). In both methods, the user-defined hyperparameters were optimized to obtain high-accuracy results. Furthermore, the k-fold cross validation technique was applied to ensure the reliability of the results. Experimental results showed that SVM with the Radial Basis Function (RBF) kernel yielded the highest accuracy (89%). As a result, SVM with RBF kernel was a viable option for speaker accent recognition applications.

K Nearest Neighbours

The ultimate goal of this paper[15] was to detect the region of UK citizens, from which region among Ireland, Midland, Northern England, Scotland, Southern England, and Wales he/she belonged, using continuous speech. Firstly, they used Mel Frequency Cepstral Coefficient (MFCC) for extracting features from continuous speech. Then they applied several Machine Learning classifiers to train and test their model. After evaluating performance, they found that

k-Nearest Neighbors (k-NN), Support Vector Machine (SVM), and Random Forest classifier provided comparatively better accuracy than others. They also performed a comparative analysis of these three algorithms. They obtained the best accuracy of 98.48% by applying k-NN classifier. Studies [12],[17] have worked on KNN for this purpose, whilst comparing it with other ML and DL techniques. both studies have comcluded that other approaches have fared better in this scenario.

Support Vector Machines

In a paper[20], an accent classification system using time-based segments consisting of Mel Frequency Cepstral Coefficients as features and employing Support Vector Machines was studied for a small corpus of two accents of English. Accuracy in the binary classification task was up to 75% to 97.5% on one- to four-second audio samples from three topics, with very high recall and precision. Its use with mismatched content was at best 85% with a tendency towards majority-class classification if the accent groups were significantly imbalanced

K-means

A new method based on class inequivalent side information and an evolutionary-based K-means clustering algorithm was proposed in a paper[18]. In a distance metric learning approach, data points were transferred to a new space where the Euclidean distances between similar and dissimilar points were at their minimum and maximum, respectively. The evolutionary-based K-means clustering approach yielded globally optimized Gaussian components for an accent classification system.

1.4 PROJECT STATEMENT

The goal of this project is to create a reliable system for classifying accents that reliably classifies a speaker's accent based on speech patterns and pronunciation. The use of machine learning algorithms and a sizable dataset of speech samples from a wide variety of accents will be used to achieve this. Accents will be categorised based on geographic, social, and linguistic aspects, including the speaker's original language and location, nation, and ethnic origin. As a result of this initiative, accents will be represented in various fields more accurately and in a wider variety, which will benefit speech recognition, language acquisition, and linguistics research.

The task of feature extraction can be labor-intensive and take up a significant amount of time. This project aims to streamline the process by utilizing the spectrogram method and incorporating advanced Deep Learning image classification algorithms. The focus of this system is to recognize regional accents, which is a more challenging task compared to identifying non-native accents. Using spectrogram images as inputs also offers the advantage of using two-dimensional data, which may lead to improved performance.

The methodology will depend on the use of deep learning techniques namely, Multilayered perceptrons, Convolutional Neural Networks and Vision transformers. Prior to this project experiments have been done using CNNs for the purpose of accent classification on spectrogram images. This project is novel in that it will employ the newly formed Vision transformers for the same purpose and then compare the results.

The project will evaluate the performance of the accent classification system on a multi-class classification task, where the system will identify multiple accents. The evaluation will be based on various performance metrics, including accuracy, precision, recall, and F1 score, to compare the results against existing accent classification systems.

1.5 OBJECTIVES

The objectives of this project are two-fold. The first is to use spectrogram images as input for the algorithms. The second is to compare the results of the different algorithms employed. In this project, the algoritms used will be the basic MLP (Multi Layered Perceptrons), the popular CNN (Convolutional Neural Networks), a hybrid CNN-LSTM with attention and the relatively new ViT (Vision Transformers). All the algorithms are popular deep learning methods for Image classification, through this comparison we can find the best perfomer for the usecase of Accent Classification.

1.6 SCOPE OF THE PROJECT

The scope of the accent classification project includes the following:

Data collection: Gathering a large and diverse dataset of speech samples from a range of accents.

Feature extraction: Extracting relevant phonetic features from the speech samples, such as stress patterns, rhythm, intonation, and pronunciation of individual sounds.

Algorithm development: Developing machine learning algorithms to analyze the extracted features and accurately classify the accent of a speaker.

Model evaluation: Evaluating the performance of the accent classification system through testing and analysis.

Documentation: Documenting the process and methodology used in the project, as well as the results and any limitations of the accent classification system.

This project will focus on developing an effective and accurate accent classification systems, but will not include developing or improving speech recognition or language learning applications themselves. The accent classification system developed in this project will be a standalone tool

Chapter 2

LITERATURE SURVEY

2.1 SUMMARY OF THE EXISTING WORKS

	Title	Merits	Demerits
	Deep Learning Techniques		
1	Accent Recognition Using a Spectrogram Image Feature-Based Convolutional Neural Network[1]	takes advantage of the convolutional neural networks' ability to characterize two-dimensional signals	Transfer learning was used
2	Foreign English Accent Classification Using Deep Belief Networks[4]	Much better accuracy than SVM,K-NN and random forest	Requires large amounts of data for good results
3	Accent classification using Machine learning and Deep Learning Models[5]	Good comparison between ML and DL techniques	CNN was not effectively used.
4	Features of Speech Audio for Accent Recognition[9]	Used MFCCs for feature extraction which improved Accuracy	Low number of layers in CNN
5	VFNet: A Convolutional Architecture for Accent Classification[11]	they presented VFNet (Variable Filter Net), a convolutional neural network (CNN) based architecture which captures a hierarchy of features to beat the previous benchmarks of accent classification, hrough a novel and elegant technique of	-

		applying variable filter sizes along the frequency band of the audio utterance	
6	Accent classification from an emotional speech in clean and noisy environments[26]	Novel addition of using different classes of decibel levels	Long training period
7	Foreign accent classification using deep neural nets[27]	CRNN had a moderate improvement over CNN	Expensive to train
8	Attention mechanism based LSTM in classification of stressed speech under workload[28]	Novel approach that yielded high accuracy	Low number of classes
9	Language Accent Detection with CNN Using Sparse Data from a Crowd-Sourced Speech[29]	Moderate accuracy	Imbalanced Data, Low number of classes
10	MARS: A Hybrid Deep CNN-based Multi-Accent Recognition System for English Language[16]	Used and created their own database. Well developed hybrid model with satisfactory accuracy	-
	Machine Learning Techniques		
11	Automatic accent classification of foreign accented Australian English speech[2]	Does not require manually labelled data	Low number of classes
12	An empirical study of automatic accent	Large number of classes	Low detection rate

	classification[3]		
13	Speaker Accent Classification System using Fuzzy Canonical Correlation-Based Gaussian Classifier[6]	Novel method with focus on minimizing the distance between the cluster centroids, but also maximizing the out-of-class variations.	Comparatively a slower algorithm because we have to compute the membership of each data point in each cluster.
14	Australian Accent-Based Speaker Classification,[7]	Novel method that used fusion of classifiers, which showed high performance	
15	Accent classification in speech[8]	Novel approach that tested on the formant frequenceies of the accent markers	Low Number of classes
16	Dialect/accent classification via boosted word modeling,[10]	Uses a novel approach of converting the problem to a word-based dialect classification problem	Low number of classes
17	Speaker Accent Recognition Using Machine Learning Algorithms[12]	Used many types of algorithms and created a good comparison	-
18	Text-Independent Foreign Accent Classification using Statistical Methods[13]	They investigate statistical approaches which differ from the a priori knowledge they need: GMM, which requires neither phonetic knowledge nor labelling, phone recognition (without a lexicon), sentence recognition (with a lexicon and a grammar).	-
19	Improved accent classification combining phonetic vowels with acoustic features[14]	they combined phonetic knowledge, such as vowels, with enhanced acoustic features to build an improved accent classification system	Demanding pre processing steps

20	A Machine Learning Approach to Recognize Speakers Region of the United Kingdom from Continuous Speech Based on Accent Classification[15]	High accuracy due to the MFCC feature extraction, Comparative analysis of three ML algorithms	-
21	US Accent Recognition Using Machine Learning Methods	Comparative analysis on the use of SVM and KNN	-
22	An evolutionary approach for accent classification in IVR systems[17]	Built to handle imbalanced data	Low accuracy
23	Speaker accent recognition through statistical descriptors of Mel-bands spectral energy and neural network model[19]	Novel approach with High accuracy	Low number of classes
24	Accent Classification Using Support Vector Machines[20]	High accuracy	Low number of classes, connot handle imbalanced data well
25	An Automated Classification System Based on Regional Accent[30]	Throughly tested different variations of algorithms to achieve high accuracy	Low number of classes

Table 1 Literature survey of previous works

2.2 CHALLENGES PRESENT IN EXISTING SYSTEMS

From the information present in Table 1, we can derive that neural networks tend to work better for regional accents and when number of classes are higher. But they are also require a demanding feature extraction process to work efficiently Feature extraction is a demanding and time-consuming process. Using the spectrogram method in conjunction with various Deep learning image classification algorithms, the process of feature extraction is simplified. The proposed system focuses on regional accent recognition, a more challenging problem than non-native accent recognition. The process of using spectrogram images also allows inputs to be two-dimensional in nature, which might prove to provide better results.

Chapter 3

ANALYSIS & DESIGN

3.1 PROPOSED METHODOLOGY

This paper proposes the use of deep learning techniques to classify the different accents in a region. The Deep learning techniques will be taking spectrogram images as inputs. This will help simplify the process of feature ectractions, which otherwise considered demanding and time-consuming [1].

In accent recognition systems, distinctive sound features reflecting the information about the accent are extracted, and classification is performed using these data. Although it is critical to determine a classification model and select features, there is no widely acknowledged technique for performing these tasks. Furthermore, the selection of features is difficult and time-consuming [22]. The computational power of image processing methods has improved due to the increased pace of image processing research and developed technologies [24]. The popularity of using features extracted from 2-dimensional data with CNNs has increased in recent years. Therefore, it is important to transform speech feature vectors into 2-dimensional feature maps, best suited for CNN processing [23]. Further, with recent advancements in Vision transformers we can look to increase the performance further, to be more efficient and accurate.

The deep learning techniques used will be namely MLPs, CNNs and ViT, their performance will compared to find the best performing algorithm for this use case.

3.2 SYSTEM ARCHITECTURE

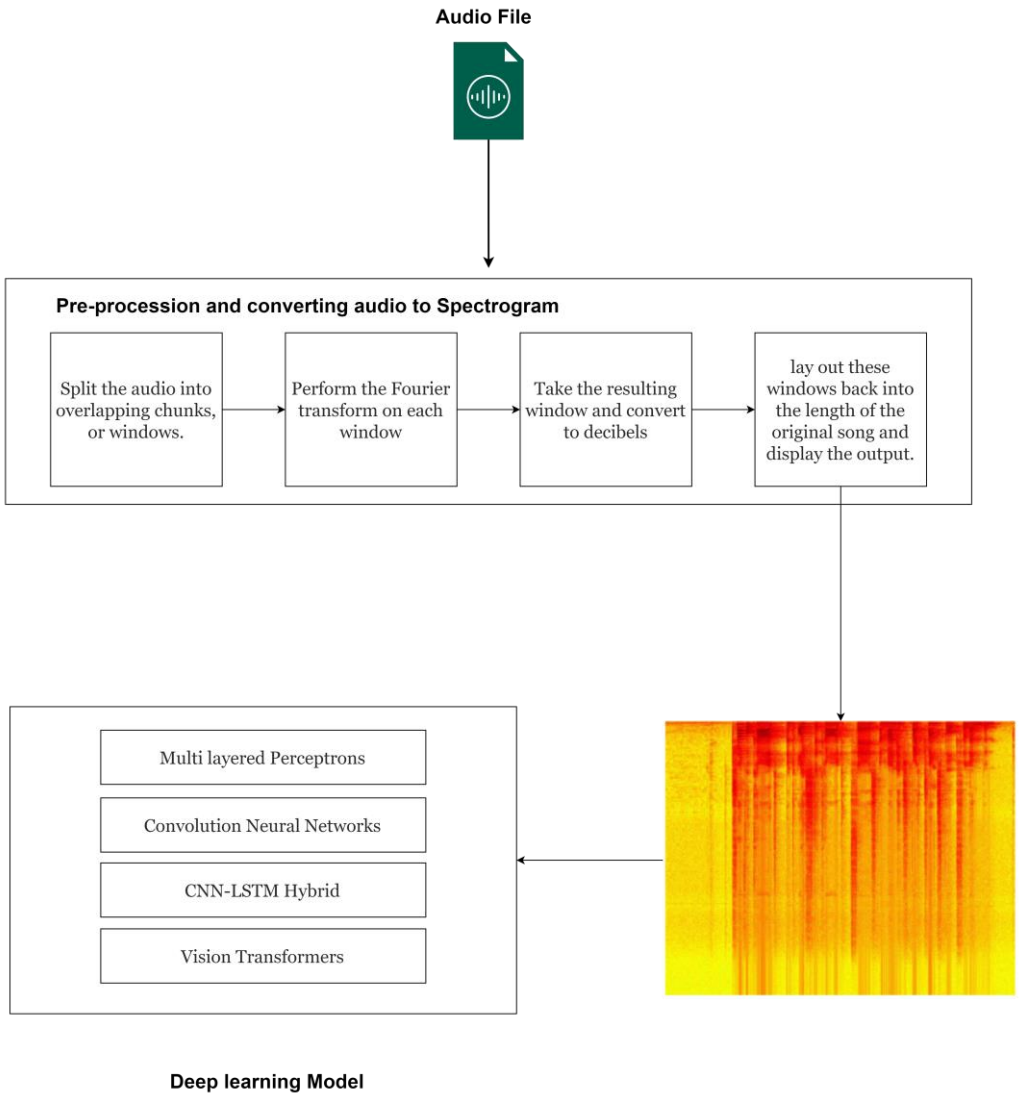


Fig 3.2.1 High-Level Architecture of proposed system

The process of analyzing speech using deep learning models typically involves several stages. The first step is to record the speech signal, which is then segmented into frames of fixed duration. Each frame is then windowed using a suitable window function, such as a Hamming or Hanning window, to reduce spectral leakage.

The resulting signal is then transformed into the frequency domain using a fast Fourier transform (FFT), which produces the spectrum of the signal. The spectrum can be further processed by applying various techniques, such as filtering, normalization, or feature extraction, to extract relevant information.

One of the most common ways to visualize the spectrum is to plot a spectrogram, which is a 2D representation of the frequency content of the signal over time. The spectrogram can reveal important features of the speech signal, such as formants, pitch, and phonemes.

Once the spectrogram is obtained, it can be fed into a deep learning model for further analysis. The choice of the deep learning model depends on the task at hand and the amount of data available. For example, a simple multi-layer perceptron (MLP) may be sufficient for basic classification tasks, while more complex tasks such as speech recognition or natural language processing may require more sophisticated models such as convolutional neural networks (CNNs), CNN-LSTM with attention, or vision transformers.

In summary, the process of analyzing speech using deep learning models involves several stages, including speech recording, framing, windowing, spectral analysis, spectrogram plotting, and deep learning model selection.

3.3 MODULE DESCRIPTIONS

For the purposes of the following experiments, two databases will be used

3.3.1 OpenSLR Dataset description

Accent	Participants	Gender	Lines	Audio Duration		Word Tokens	
				Total (h:m:s)	Average(s)	Total	Unique
Irish	3	M	450	00:42:56	5.72	6042	1888
Northern	5	F	750	01:22:11	6.58	10215	2707
	14	M	2097	03:37:42	6.23	28595	5438
Southern	28	F	4161	07:11:17	6.22	57508	6781
	29	M	4331	07:24:49	6.16	59697	6804
Midlands	2	F	246	00:28:12	6.88	3468	1395
	3	M	450	00:43:55	5.86	6310	1978
Scotish	6	F	894	01:35:05	6.38	12187	3069
	11	M	1649	02:44:42	5.99	22194	4539
Welsh	8	F	1199	02:28:12	7.42	16139	3425
	11	M	1650	02:58:13	6.48	22204	4355
Total	**120**		**17877**	**31:17:19**			

Table 2 Dataset description

The dataset described in Table 2 includes high-quality audio clips of volunteers speaking with various English accents (dataset access via "https://openslr.org/83/") [25]. Northern England, Southern England, the Midlands, Irish English, Scottish English, and Welsh English are the six accent classes that have been identified. As shown in Table 2, The collection comprises of 17,877 records from 120 participants (49 women and 71 men) for six accents, along with the

corresponding transcriptions. Natural language words and uncommon word token expressions, like numerals, are included in transcriptions. The bulk of transcriptions fall between 39 and 119 characters, however they can be as long as 169 characters. As in table 2 The shortest speech is 1.62 seconds long, while the longest is 20.1 seconds long. 244,558 tokens, including 7,646 unique ones, totaling over 31 hours of audio recordings were gathered [25].

Although this paper uses the work of [25], [16] will help improve models in the future. This paper[16] proposed a novel, well-structured database of non-native Indian English speaker accents, referred to as IndicAccentDB. IndicAccentDB contained speech samples from six different states to address the unbalanced dataset (gender-bias) and speaker mismatch problems observed in the past. The proposed work also discussed the requirements for creating the IndicAccentDB database and pre-processing tasks performed on the dataset.

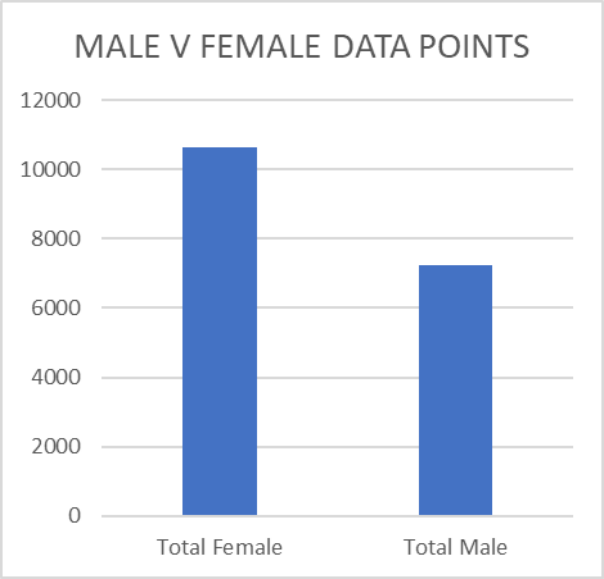


Fig 3.3.2.1 A bar graph representing Male and Female data points

Figure 4.3.1 reveals that the dataset consists of a higher number of female data points, totaling 10,627 in comparison to male data points, which number at 7,250. This gender distribution may potentially cause an imbalance in the dataset, leading to bias towards the female population. To address this issue, appropriate measures need to be taken during data preparation, such as oversampling the minority class (in this case, male data points), undersampling the majority class (female data points), or using more advanced techniques such as data augmentation or generating synthetic data. This will ensure that the final model is trained on a balanced dataset that represents the population well and can make accurate predictions for both genders. It is crucial to

consider dataset imbalance during the machine learning pipeline to prevent biased predictions and promote fairness in the model's performance.

In machine learning, having imbalanced data can cause issues with bias and skew results towards certain classes. This is the case with the data shown in Figure 4.3.2, which heavily leans towards the "Southern" accent class, potentially causing skewed results. Additionally, the "Irish" accent class only includes male participants, further favoring a particular subset during probability value prediction.

To address this issue, data augmentation and cleaning methods can be implemented. Data augmentation techniques involve creating additional training data by applying transformations to existing data, such as rotating or flipping images, or adding noise to audio data. These techniques can help balance the dataset and increase the variety of examples in underrepresented classes. Additionally, data cleaning methods such as removing duplicates or outliers can improve the quality of the data and reduce the impact of noisy data on the model's performance.

Another approach to address this issue is to collect more data from underrepresented classes. This could involve recruiting more participants from the "Irish" accent class or collecting data from a broader range of geographic regions to increase the representation of other accent classes.

Ultimately, it is essential to recognize and address issues with imbalanced data to avoid biased results and ensure that machine learning models perform effectively across all classes. By implementing data augmentation and cleaning methods and collecting additional data, we can improve the quality of the data and the accuracy of the model's predictions.

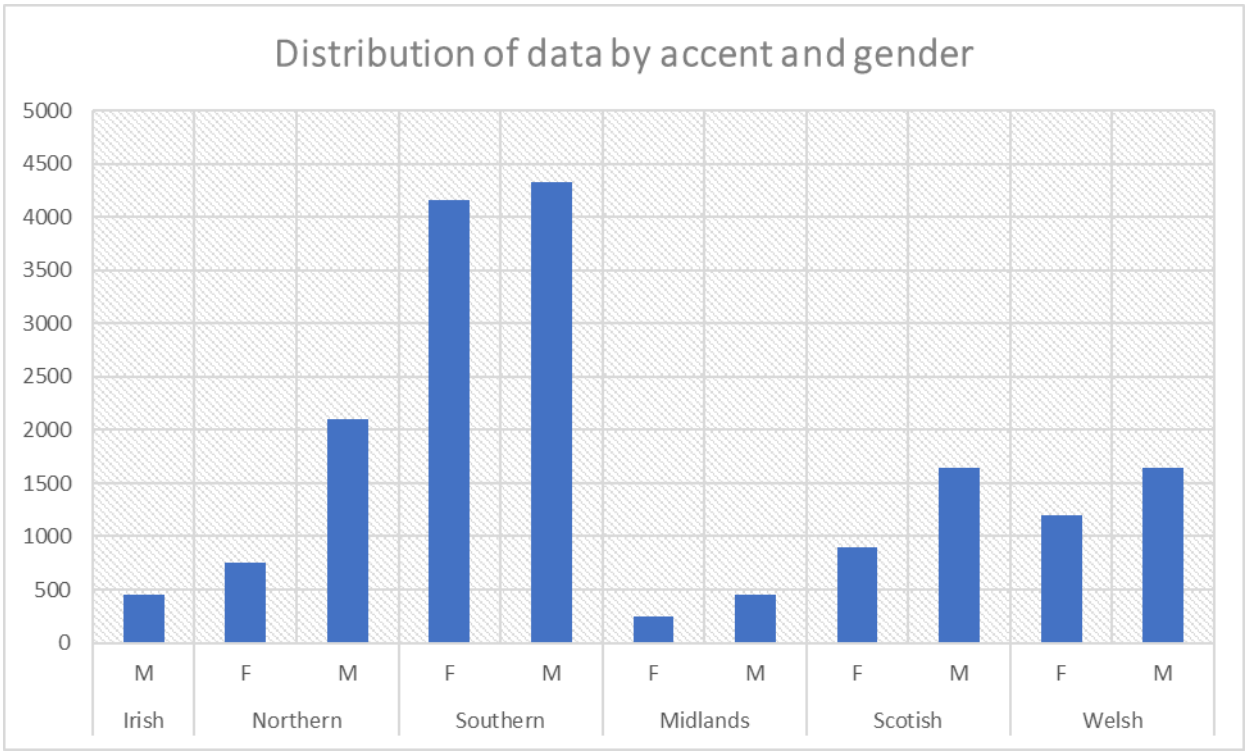


Fig 3.3.2.2 Distribution of data on the basis of accent and gender

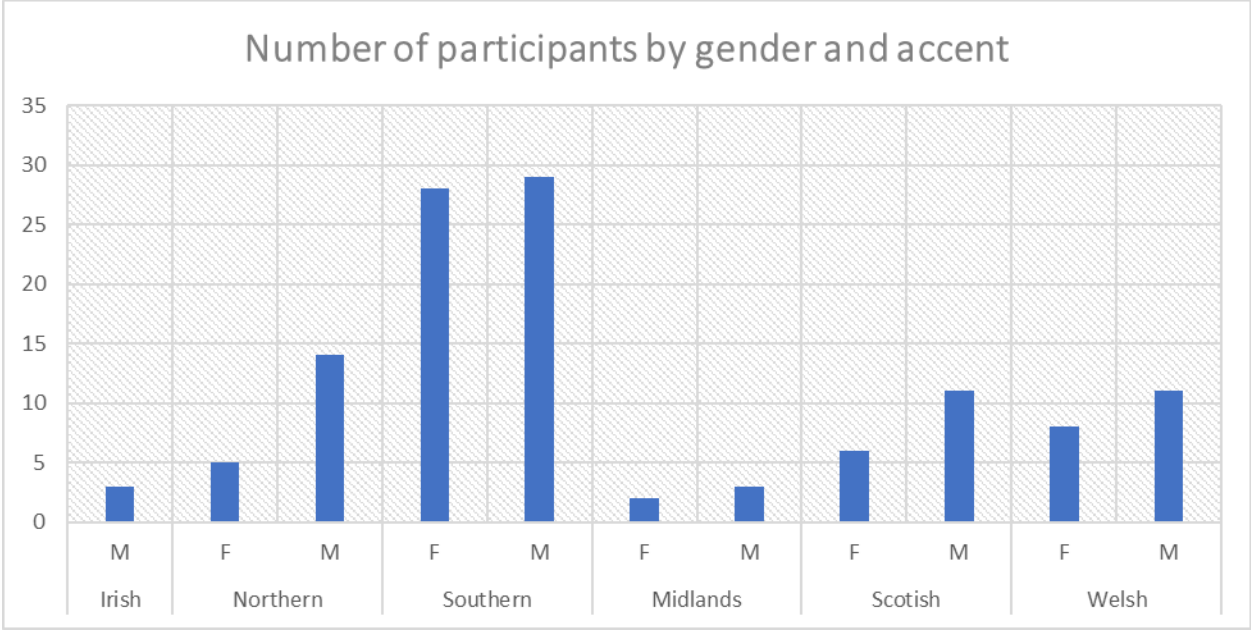


Fig 3.3.2.3 Number of participants by gender and accent

The number of participants is crucial for a database of different accents because it directly impacts the quality and representativeness of the data. The more participants there are in the database, the more diverse and representative the accents in the database will be. For instance, if a database only has a small number of participants, it may not capture the full range of variation within a particular accent, leading to biased or incomplete representation of that accent in the dataset. This can impact the accuracy of the model trained on the data, as it may struggle to generalize to new data with accents that were not well-represented in the training set.Moreover, having a larger number of participants can also increase the variability of accents within each class, making it easier for the model to distinguish between different accents. This increased variability also makes it more likely that the model can generalize to new accents that were not included in the training set, which is essential for real-world applications.Therefore, having a sufficient number of participants from various regions and backgrounds is crucial for building a comprehensive and representative database of different accents. This ensures that the machine learning model can be trained effectively and generalize well to new accents, resulting in more accurate and reliable predictions in real-world scenarios.

3.3.2 AccentDB dataset description

Accent	Number of sample	Time Duration	Number of Speakers
Bangla	1528	02:13:00	2
Malayalam	2393	03:32:00	3
Odiya	748	01:11:00	1
Telugu	1515	02:10:00	2

Table 3 AccentDB Dataset description

The database, AccentDB, across 4 non-native accents of Bangla, Malayalam, Odiya and Telugu; The number of samples, duration of all samples and the number of speakers per accent are listed in Table 3. AccentDB is collected by employing the Harvard Sentences (IEEE, 1969) which are phonetically balanced sentences that use specific phonemes at the same frequency as they appear in English language. The sentences in this dataset are neither too short nor too long, making them suitable for proper manifestation of accents in sentence-level speech. Harvard Sentences dataset contains 72 sets, each consisting of 10 sentences.

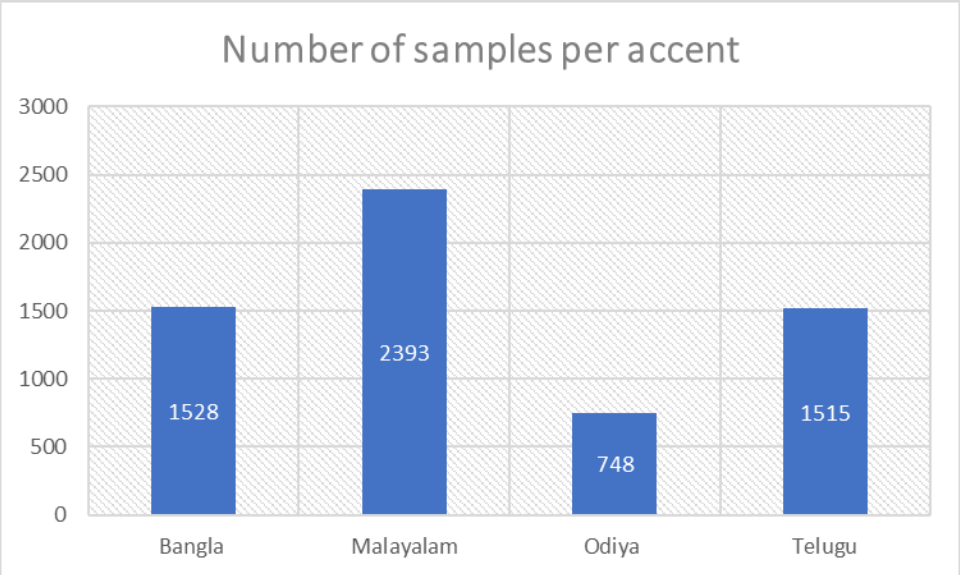


Fig 3.3.2.4 Number of samples per accent

As evident in figure 3.3.4 Malyalam has the highest number of sample at 2393 followed by Bangla, Telugu and Odiya in that order. The same order is also observed in Fig 3.3.2.5

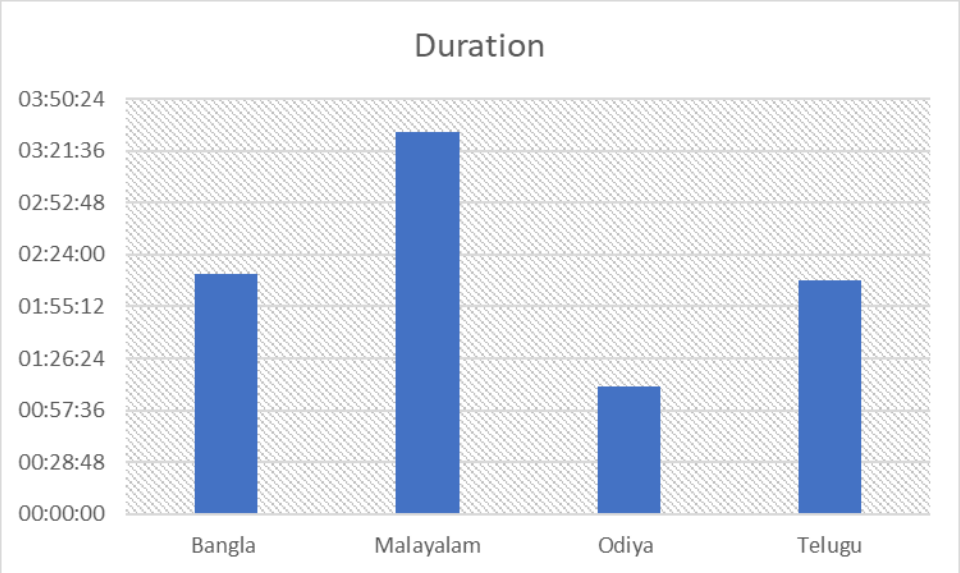


Fig 3.3.2.5 Total Duration of the samples

3.3.3 Pre-processing

Step 1 : Split the audio into overlapping chunks or windows

Since the signal may contain a lot of information, as depicted in Fig 3.3.2.1, it is divided into small, overlapping segments of equal length. Each segment is typically 10-30 milliseconds long and is selected to be long enough to contain a few cycles of the lowest frequency components of interest, but short enough to capture any changes in the signal over time.

When the signal is divided into segments, it is necessary to apply a window function to each segment. The windowing process aims to reduce spectral leakage, which occurs when a signal's frequency content "leaks" into adjacent frequency bins, leading to distortion in the spectrogram. Common window functions include the Hamming, Hanning, and Blackman-Harris windows

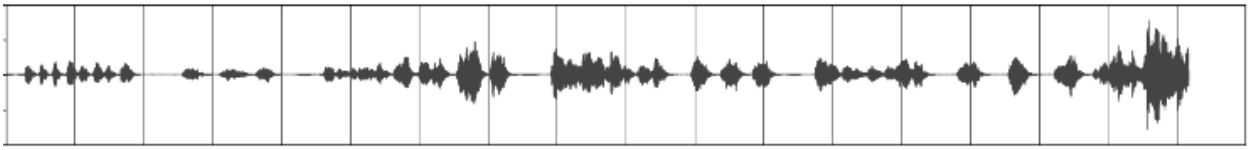


Fig 3.3.3.1 Splitting the audio into chunks

Step 2 : Perform the Fourier transform on each window

The Fourier transform is used to convert each segment from the time domain to the frequency domain. The Fast Fourier Transform (FFT) is typically used to perform this transformation, which calculates the frequency spectrum of each segment.

The discrete Fourier transform (DFT) of a time-domain signal x(n) is defined as:

$$X(k) = \Sigma(n=0 \text{ to } N-1)\ x(n) * \exp(-2\pi ikn/N) \qquad (1)$$

where k is the frequency index and N is the length of the signal.

The FFT is a fast algorithm for computing the DFT, which reduces the computational complexity from O(N^2) to O(N log N) for signals of length N.

Step 3: Compute the power spectrum

The magnitude of the complex FFT output is squared to obtain the power spectrum. The power spectrum represents the distribution of energy across different frequencies in the signal.

The power spectrum of the Fourier transform X(k) is defined as the squared magnitude of the complex Fourier coefficients:

$$P(k) = |X(k)|^2 \quad (2)$$

formula 3.2 Obtaining the power spectrum using Fourier transform

where |X(k)| is the magnitude of the complex Fourier coefficient at frequency index k.

In practice, the power spectrum is usually calculated in decibels (dB) to better represent the dynamic range of the signal's power spectrum, using the following formula:

$$P\_dB(k) = 10 * \log_{10}(P(k) / P\_ref) \quad (3)$$

where P_ref is the reference power level, typically set to the maximum value of the power spectrum.

The resulting power spectrum represents the distribution of energy across different frequencies in the signal, which is used to create the spectrogram.

Step 4: Combine the spectra

The power spectra from all the segments are combined to obtain a 2D matrix, with time on one axis and frequency on the other. The segments are typically overlapped to ensure continuity between segments.

Step 5: Plot the spectrogram:

Finally, the spectrogram is plotted as a 2D image, with time on the x-axis, frequency on the y-axis, and color indicating the power of each frequency component. The color scale is chosen to represent the dynamic range of the signal's power spectrum, typically using a logarithmic scale for the frequency axis to better represent the human perception of sound. The resulting image allows for easy visualization and analysis of the frequency content of the signal as it varies over time.

The aforementioned steps are easily achieved by using various python libraries like numpy, matplotlib and librosa

```
import librosa

import librosa.display

import numpy as np

import matplotlib.pyplot as plt

# Load in sounds

guitar, sr = librosa.load('guitar.wav')

kick, _ = librosa.load('kick.wav')

snare, _ = librosa.load('snare.wav')

def to_decibles(signal):

    # Perform short time Fourier Transformation of signal and take absolute value of results

    stft = np.abs(librosa.stft(signal))

    # Convert to dB

    D = librosa.amplitude_to_db(stft, ref = np.max) # Set reference value to the maximum value
of stft.

    return D # Return converted audio signal

# Function to plot the converted audio signal

def plot_spec(D, sr, instrument):

    fig, ax = plt.subplots(figsize = (30,10))

    spec = librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='linear', ax=ax)

    ax.set(title = 'Spectrogram of ' + instrument)

fig.colorbar(spec)
plot_spec(to_decibles(guitar), sr, 'Guitar')
```

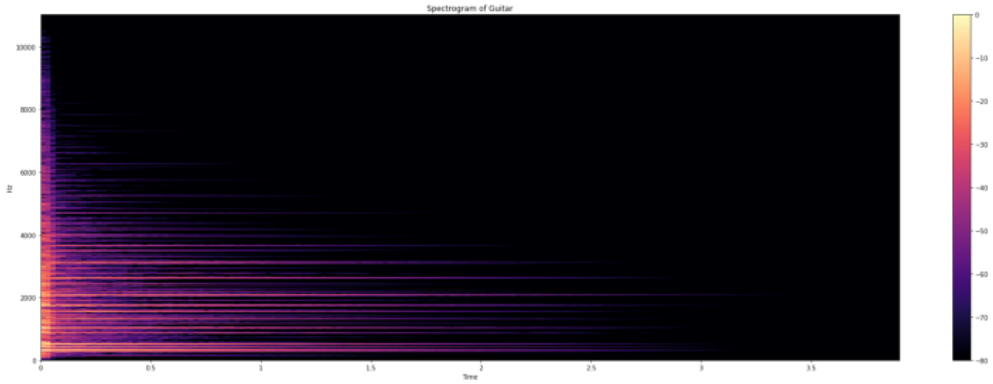


fig 3.3.3.2 Spectrogram image of sample audio created using librosa

3.3.4 Spectrogram Image

A spectrogram is a visual map of a signal's time-varying frequency spectrum, in which the horizontal axis represents time and the vertical axis represents frequency [21], making it a useful tool for analyzing and understanding various signals and signals processing applications.

The x-axis of a spectrogram represents time, the y-axis represents frequency, and the amplitude of the signal at each time-frequency point is represented by the color or grayscale intensity. The color or grayscale intensity typically indicates the strength or power of the signal at that particular time-frequency point, with brighter or higher intensity values representing stronger signals and darker or lower intensity values representing weaker signals.

One of the main uses of spectrograms is in the analysis of speech signals. By examining the spectrogram of a speech signal, linguists and speech scientists can identify the frequency components that make up speech sounds, such as vowels and consonants, and gain insights into the production and perception of speech.

To create a spectrogram, a signal is typically first transformed into the frequency domain using techniques such as the Fourier transform or the Short-Time Fourier Transform (STFT). The frequency domain representation of the signal is then divided into overlapping segments, and the magnitude of the signal in each segment is computed and displayed as a function of time and frequency.

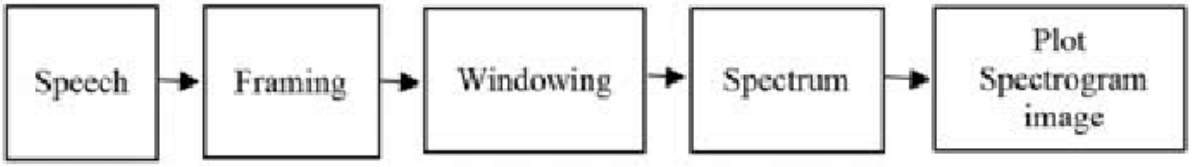


Fig 3.3.4.1 Process of creating spectrogram image

The steps mentioned in Fig 4.3.3.1 have be further explained below

Speech: The speech signal can be recorded using a microphone or obtained from a pre-recorded audio file.

Framing: The speech signal is divided into smaller, overlapping frames to ensure that the signal remains stationary within each frame. This is necessary because speech signals can change rapidly over time, and analyzing stationary segments makes it easier to identify frequency content.

Windowing: Each frame is multiplied by a window function to taper the edges of the frame and reduce spectral leakage, which occurs when spectral components of the signal "leak" into neighboring frequency bins. The Hamming window is a common choice because it offers a good balance between frequency resolution and amplitude accuracy.

Spectrum: The Fourier transform is applied to each windowed frame to calculate the power spectrum, which represents the distribution of power across different frequencies in the signal. The power spectrum can be calculated using different types of Fourier transforms, such as the Discrete Fourier Transform (DFT) or Fast Fourier Transform (FFT).

Plot spectrogram image: The power spectrum for each frame is plotted over time to create the spectrogram image. The resulting image provides a visual representation of the frequency content of the speech signal over time, with brighter areas indicating higher power or intensity at a particular frequency and time. Spectrograms are commonly used in speech analysis to identify phonetic units, analyze prosody, and detect various speech disorders.

Overall, the process of creating a spectrogram image involves dividing the speech signal into smaller frames, applying a window function to each frame, calculating the power spectrum using the Fourier transform, and finally plotting the power spectrum over time to create the spectrogram image. The resulting image can be used to analyze various aspects of the speech signal, such as frequency content, intensity, and duration.

3.3.5 Employ Deep learning Techniques

In this project we use multiple Deep learning techniques to classify the audio into different accents, the end of it the goal is to compare the performance of each of these algorithms.

3.3.5.1 Multi-Layered Perceptrons (MLPs)

Multi-Layer Perceptrons (MLPs) are a type of artificial neural network that is commonly used for supervised learning tasks such as classification and regression. They are called Multi-Layer Perceptrons because they consist of multiple layers of artificial neurons, or "perceptrons".As shown inf fig 4.3.4.2 Each layer of the MLP consists of a set of artificial neurons, and each neuron receives input from the neurons in the previous layer, processes the input using an activation function, and outputs a signal to the neurons in the next layer.

The final layer of the MLP outputs the final prediction or classification of the input data. MLPs are trained using a process called backpropagation, where the error between the predicted output and the actual output is calculated, and the weights of the neurons are adjusted to minimize the error.

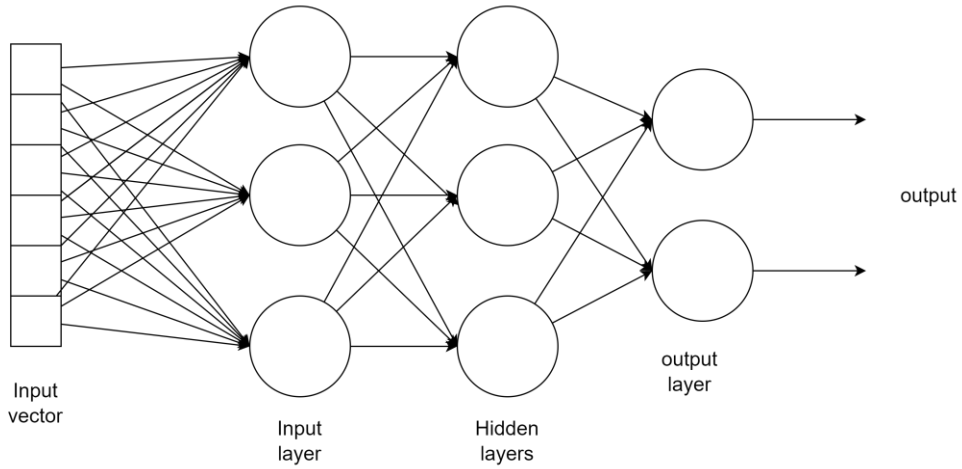


Fig 3.3.5.1 Architecture of an MLP

This process is repeated many times until the MLP is able to make accurate predictions on the training data. One of the key advantages of MLPs is their ability to learn non-linear relationships between inputs and outputs, as opposed to linear models such as linear regression. This makes them well-suited for complex data and tasks, such as image classification and speech recognition. However, MLPs can be prone to overfitting, where the network becomes too complex and performs well on the training data but poorly on unseen data. To combat this, techniques such as regularization and early stopping can be used.

To perform image classification using MLP, the input image is typically pre-processed to extract relevant features, such as edges and textures. These features are then fed into the MLP as input, and the network uses its multiple layers of neurons to process the features and make a prediction. The MLP is trained on a labeled dataset of images, where the goal is to learn the relationships between the features of the images and their corresponding class labels. The training process involves using an optimization algorithm, such as gradient descent, to minimize the error between the predicted class labels and the actual class labels.

3.3.4.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a type of neural network designed specifically for image classification tasks. They are called Convolutional Neural Networks because they use a technique called convolution to extract meaningful features from the input image. In image classification, the goal is to predict the class label of an input image. With CNNs, as shown in fig 3.3.3.3 the input image is processed through multiple layers of neurons, each of which performs a convolution operation. The convolution operation involves applying a small filter to each portion of the image, producing a feature map that captures the important patterns and structures in the image. The output of the convolution operation is then passed through activation functions, which introduce non-linearity into the network. The resulting feature maps are then processed

through multiple fully connected layers, which learn the relationships between the features and the class labels.

One of the key advantages of CNNs is their ability to learn hierarchical representations of the input data. This allows the network to automatically learn and extract increasingly complex features from the input image, such as edges, textures, and shapes, and use them to make accurate predictions.

Another advantage of CNNs is their ability to handle large amounts of data, as they can be trained on large datasets of images using GPUs, which allows for faster processing.

In conclusion, Convolutional Neural Networks are a powerful and widely used approach for image classification tasks. By using convolution to extract meaningful features from the input image, and learning the relationships between the features and the class labels, CNNs can make accurate predictions and handle large amounts of data.

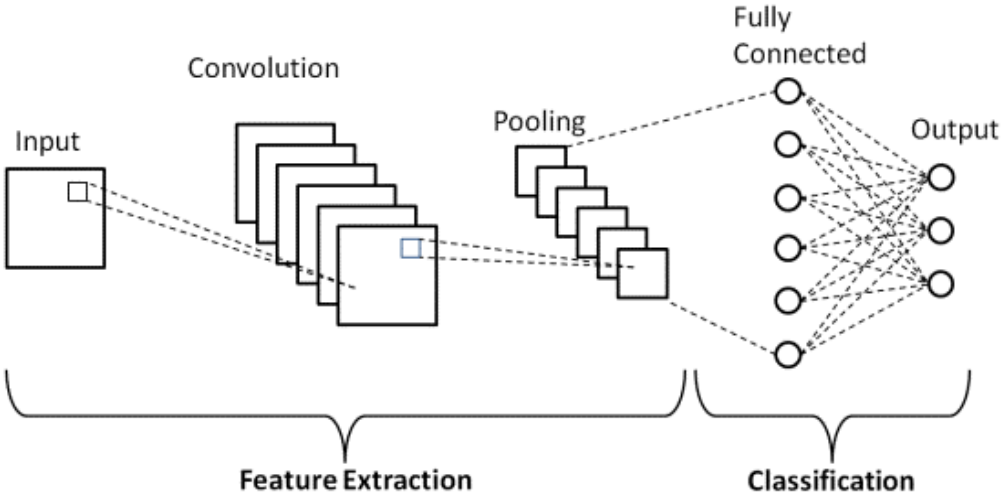


Fig 3.3.5.2 Architecture of a CNN

3.3.4.3 Long Short-term memory (LSTM)

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) that was designed to overcome the vanishing gradient problem. LSTMs are capable of capturing long-term dependencies in time-series data by selectively remembering and forgetting information.

As depicted in Fig 4.3.3.4The architecture of an LSTM includes a memory cell, input gate, output gate, and forget gate. The memory cell is responsible for storing information over time, and the gates control the flow of information into and out of the cell. The input gate

determines which information should be stored in the cell, and the forget gate decides which information should be discarded. The output gate decides which information should be outputted from the cell.

LSTMs have been successfully applied to various tasks, including language modeling, speech recognition, machine translation, and image captioning. They are particularly useful for applications where long-term dependencies are crucial, such as in speech recognition where the context of a word depends on the words that precede it.

Overall, LSTMs are a powerful tool for modeling sequential data, and their ability to selectively remember and forget information has led to significant improvements in various applications.

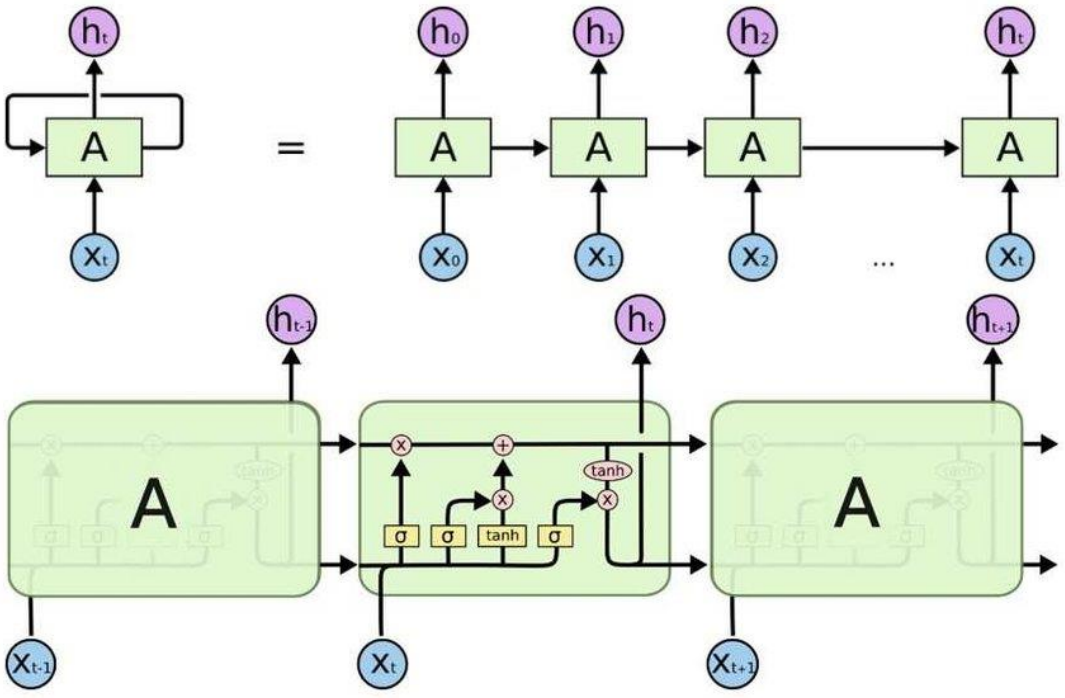


Fig 3.3.5.3 Architecture of an LSTM

3.3.4.4 Gated recurrent units (GRU)

A GRU algorithm is a type of recurrent neural network (RNN) that uses gating mechanisms to selectively update the hidden state of the network at each time step. The GRU algorithm was introduced by Cho et al. in 2014 as a simpler alternative to the Long Short-Term Memory (LSTM) algorithm, which also uses gating mechanisms but has more parameters and an output gate. The GRU algorithm can process sequential data such as text, speech, and time-series data, and has been shown to be effective in various natural language processing tasks, such as language modeling, machine translation, and speech recognition.

The GRU algorithm consists of two gates: the reset gate and the update gate. The reset gate determines how much of the previous hidden state should be forgotten, while the update gate determines how much of the new input should be used to update the hidden state. The output of the GRU algorithm is calculated based on the updated hidden state. The equations used to calculate the gates and the hidden state of a GRU algorithm are as follows:

$$Reset\ gate:\ r_t = \sum (W_r * [h_{t-1}, x_t])$$

(4)

$$Update\ gate:\ z_t = \sum (W_z * [h_{t-1}, x_t])$$

(5)

$$Candidate\ hidden\ state:\ h_{t'} = tanh\ (W_h * [r_t * h_{t-1}, x_t])$$

(6)

$$Hidden\ state:\ h_t = (1 - z_t) * h_{t-1} + z_t * h_{t'}$$

(7)

where W_r, W_z, and W_h are learnable weight matrices, x_t is the input at time step t, h_ {t-1} is the previous hidden state, and h_t is the current hidden state.

In summary, a GRU algorithm is a type of RNN that uses gating mechanisms to selectively update the hidden state at each time step, allowing it to effectively model sequential data. It has fewer parameters than LSTM, but has similar performance on certain tasks.

3.3.4.5 CNN-LSTM/GRU Hybrid with attention

The CNN-LSTM/GRU hybrid with attention is a deep learning model designed for classifying spectrograms. Spectrograms are two-dimensional representations of sound signals that are widely used in speech recognition and music classification tasks. By combining the strengths of convolutional neural networks (CNNs), long short-term memory (LSTM) networks, Gated recurrent units (GRU) and attention mechanisms, this hybrid model can improve the accuracy of spectrogram classification.

As depicted in Fig 4.3.3.6 the CNN component of the hybrid model plays a critical role in feature extraction from the spectrogram. This involves identifying relevant frequency patterns and time-dependent variations. In contrast, the LSTM/GRU component takes these extracted features as input and uses them to learn the temporal dependencies that exist between them. This enables the model to recognize long-term patterns within the sound signal, a crucial aspect of accurate spectrogram classification. Meanwhile, the attention mechanism determines the importance of various parts of the spectrogram, allowing the model to focus on the most relevant features.

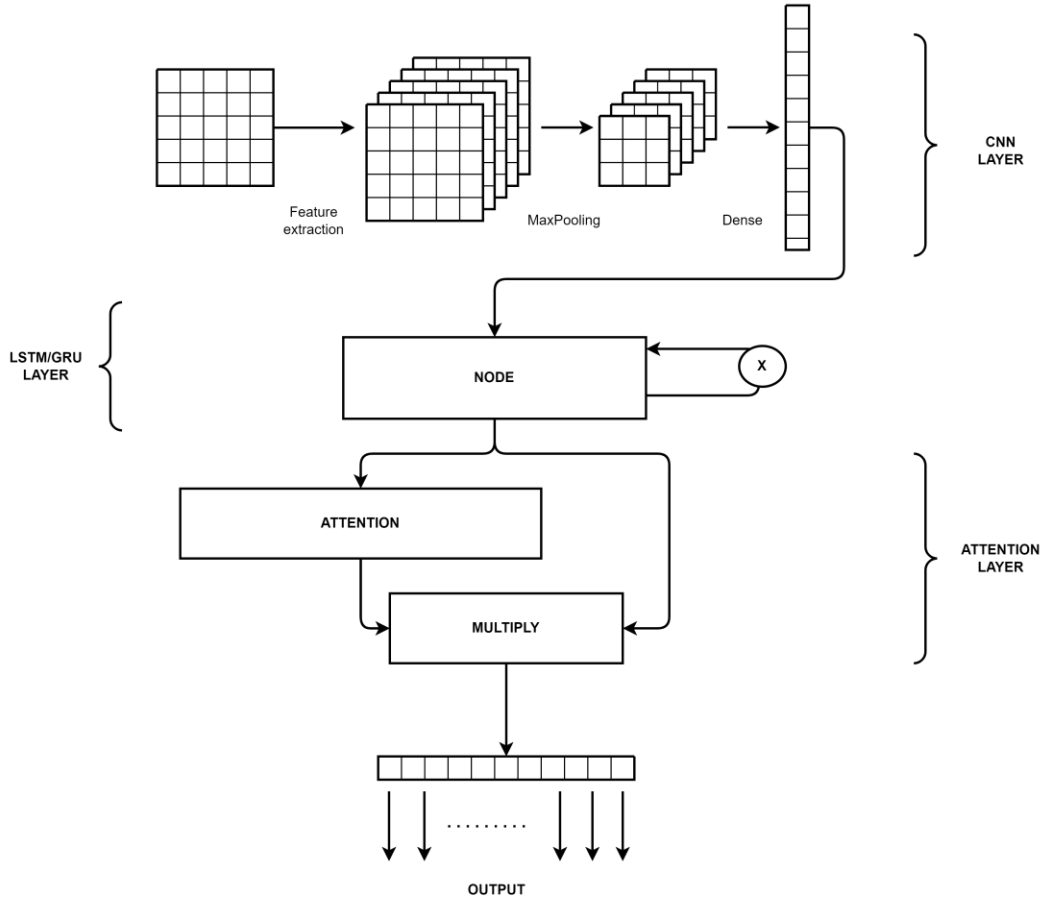


Fig 3.3.5.4 Architecture of CNN-LSTM with attention

One of the most significant benefits of the CNN-LSTM hybrid with attention is its ability to handle time-varying inputs. This is particularly relevant in tasks like speech recognition and music classification, where sounds evolve over time. By leveraging the LSTM component, the model can account for the temporal evolution of sound signals, which is critical for achieving high accuracy. Furthermore, the attention mechanism enables the model to highlight the most critical parts of the spectrogram, resulting in improved classification performance.

Researchers have used the CNN-LSTM hybrid with attention in a wide range of spectrogram classification tasks, such as speech recognition, music genre classification, and environmental sound classification. The model has shown promising results in these domains, demonstrating its versatility and ability to perform well in different contexts.

Chapter 4

IMPLEMENTATION AND TESTING

In this work, three different models have been implemented and tested namely CNN, Hybrid CNN-LSTM with attention, Hybrid CNN-GRU with attention and MLP. Each model uses the same training and validation data to maintain uniformity and obtain accurate results that relies on the model itself. Since the dataset consists of audio files, it is quite large and hence while implementing the models we will be using a data generator.

4.1 Data pre-processing

When working with a large dataset that consists of audio files, it is important to optimize the way that the data is loaded to ensure that it is being used efficiently. One approach that can be used is to create a dataset of the file paths, rather than directly loading the audio files. This approach has several benefits, including reducing the memory overhead of loading the files and allowing for more efficient data processing.

By creating a dataset of the file paths, we can use a data generator to feed the files in batches to the respective model. This approach ensures that only the necessary data is being loaded into memory at any given time, which can significantly improve the efficiency of the training process. Additionally, this approach allows for more flexible data processing, as we can easily apply various transformations to the data before it is fed to the model.

Overall, by creating a dataset of the file paths and using a data generator, we can effectively manage large amounts of audio data and optimize the training process for our model. This approach allows us to efficiently process the data in batches, while minimizing memory overhead and maximizing flexibility in our data processing.

To achieve this affect we implement the below code.

```
#For openSLR dataset

main_path="C:\\Users\\user\\Desktop\\Capstone\\data"


data_files = list(os.listdir(main_path))


metadata = {"file_path":[],"category":[]}


count=-1

cats = ['irish_english_male', 'midlands_english_female',
'midlands_english_male',

        'northern_english_female', 'northern_english_male',
'scottish_english_female', 'scottish_english_male',

        'southern_english_female', 'southern_english_male',
'welsh_english_female', 'welsh_english_male']

for dirname, dirnames, filenames in os.walk(main_path):

    # print path to all filenames.

    file_paths=[]

    for filename in filenames:

        path= os.path.join(dirname, filename)

        path = os.path.normpath(path)

        file_paths.append(path.replace("\\","/"))

    metadata['file_path']=metadata['file_path']+file_paths


metadata['category']=metadata['category']+([cats[count]]*len(file_path
s))

    count+=1
```

```
df = pd.DataFrame(metadata)

df.columns=['file_path','class_label']
df.to_csv("data_metadata.csv",index=False)
def split_data(data):
    data = data.sample(frac=1,axis=0,ignore_index=True)
    train = data[0:int(len(data)*0.8)]
    test = data[int(len(data)*0.8):]
    return train,test

train_data,val_data = split_data(data)
train_data.to_csv('train_data.csv',index=False)
val_data.to_csv('val_data.csv',index=False)

#For AccentDB dataset
metadata = {"file_path":[],"class_label":[]}

count=-1
cats = ['bangla', 'malayalam', 'odiya', 'telugu']
for dirname, dirnames, filenames in os.walk(main_path):
    # print path to all filenames.
    file_paths=[]
    for filename in filenames:
        path= os.path.join(dirname, filename)
        path = os.path.normpath(path)
```

```
        file_paths.append(path.replace("\\","/"))
    metadata['file_path']=metadata['file_path']+file_paths

metadata['class_label']=metadata['class_label']+([cats[count]]*len(fil
e_paths))
    count+=1
```

4.2 Implementing CNN

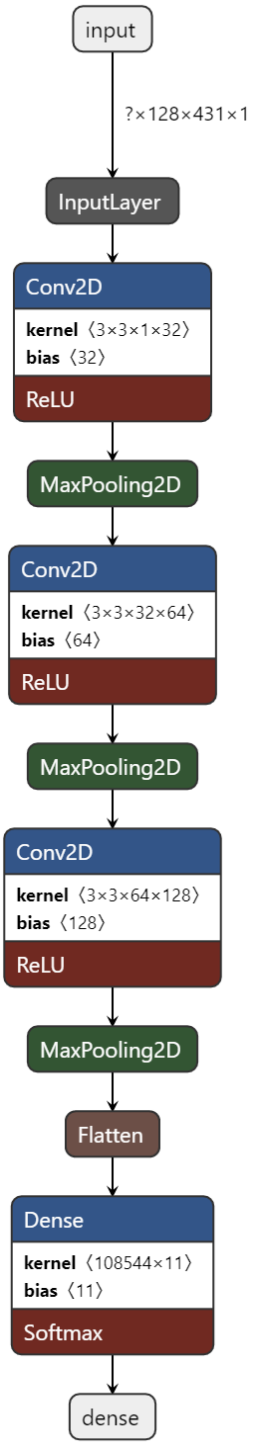


Fig 4.2.1 Architecture of CNN model implementation

For OpenSLR database

```
import librosa

import matplotlib.pyplot

import os

import pandas as pd

import numpy as np

import tensorflow as tf

from keras.utils import to_categorical
```

Defining the generator

```
class AudioDataGenerator(tf.keras.utils.Sequence):

    def __init__(self, csv_file, batch_size=32, num_classes=11,
sample_rate=22050, duration=10.0, shuffle=True, n_mels=128):

        self.csv_file = csv_file

        self.batch_size = batch_size

        self.num_classes = num_classes

        self.sample_rate = sample_rate

        self.duration = duration

        self.shuffle = shuffle

        self.n_mels  = n_mels


        # Read the CSV file

        self.data = pd.read_csv(csv_file)


        # Get the unique class labels
```

```
        self.classes = sorted(self.data['class_label'].unique())

        # Create a dictionary to map class labels to integers
        self.class_to_int = dict(zip(self.classes,
range(len(self.classes))))

        # Shuffle the data if requested
        if self.shuffle:
            self.data =
self.data.sample(frac=1).reset_index(drop=True)

    def __len__(self):
        # Return the number of batches
        return int(np.ceil(len(self.data) / float(self.batch_size)))

    def __getitem__(self, idx):
        # Get the batch of file paths and labels
        batch_data = self.data[idx * self.batch_size:(idx + 1) *
self.batch_size]
        batch_data = batch_data.reset_index()

        # Initialize the arrays for the audio data and labels
        batch_x = np.zeros((len(batch_data), self.n_mels, 431,1))
        batch_y = np.zeros((len(batch_data), self.num_classes))

        # Load the audio files and their corresponding labels
```

```
        for i, row in batch_data.iterrows():
            file_path = row['file_path']
            class_label = row['class_label']

            # Load the audio file
            signal, sr = librosa.load(file_path, sr=self.sample_rate,
mono=True)

            # Pad or truncate the signal to the desired length
            signal = librosa.util.fix_length(signal,
size=self.sample_rate * self.duration)

            # Convert the audio file to spectrogram
            S = librosa.feature.melspectrogram(y=signal, sr=sr)
            S_dB = np.array(librosa.power_to_db(S, ref=np.max))
            S_dB = S_dB.reshape(S_dB.shape[0],S_dB.shape[1],1)

            # Save the audio data and label to the batch arrays
            batch_x[i, :] = S_dB
            batch_y[i, :] =
to_categorical(self.class_to_int[class_label],
num_classes=self.num_classes)

        return batch_x, batch_y

train_data = pd.read_csv("train_data.csv")
```

```
val_data = pd.read_csv("val_data.csv")
```

Instantiating the generator

```
train_generator = AudioDataGenerator('train_data.csv', batch_size=32,
num_classes=11, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Defining the metrics

```
from keras import backend as K


def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall


def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision


def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
```

```
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Defining the model

```
import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Activation,
Flatten, Conv2D, MaxPooling2D

model = Sequential()


model.add(Conv2D(32,(3,3),input_shape=(128,431,1),activation='relu',
padding='same'))

model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Conv2D(64,(3,3), activation='relu', padding='same'))

model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Conv2D(128,(3,3), activation='relu', padding='same'))

model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Flatten())


model.add(Dense(11,activation='softmax'))


model.compile(loss='categorical_crossentropy',optimizer='adam',metrics
=['accuracy',f1_m,precision_m, recall_m])
```

Training the model

```
history = model.fit(train_generator,batch_size=32,epochs=10,verbose=1)
```

Instantiating validation data generator

```
val_generator = AudioDataGenerator('val_data.csv', batch_size=32,
num_classes=11, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Evaluating the model

```
loss, accuracy, f1_score, precision, recall =
model.evaluate(val_generator)
```

Plotting training history

```
import matplotlib.pyplot as plt


plt.plot(history['loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.show()

plt.close()
```

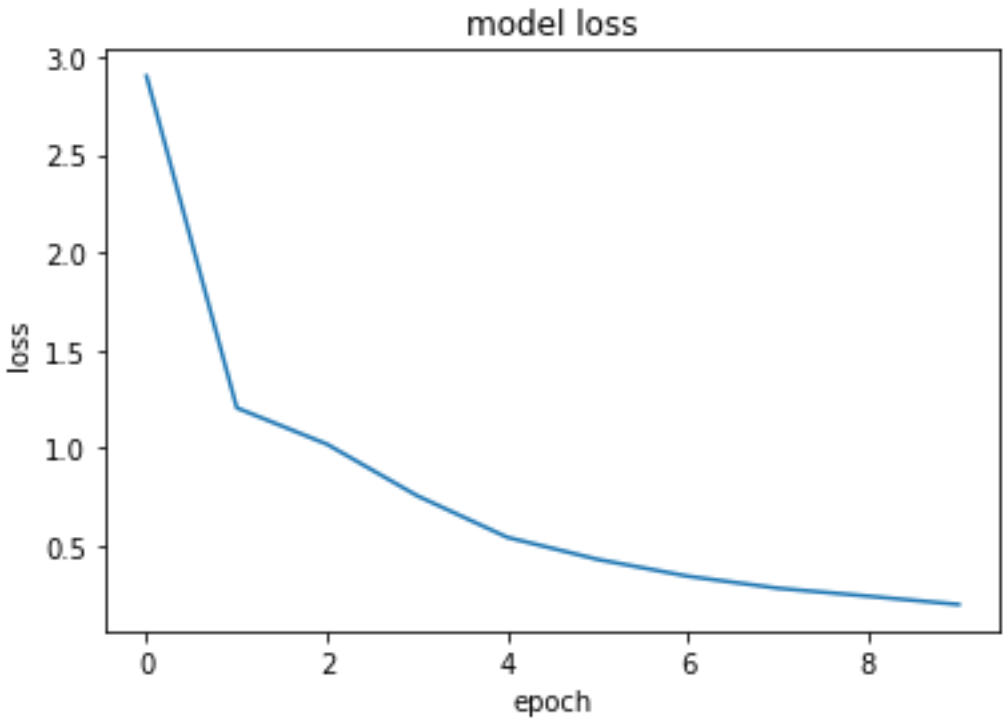


Fig 5.2.2 CNN Model training loss evolution

```
plt.plot(history['accuracy'])
plt.title('model accuracy')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
plt.close()
```

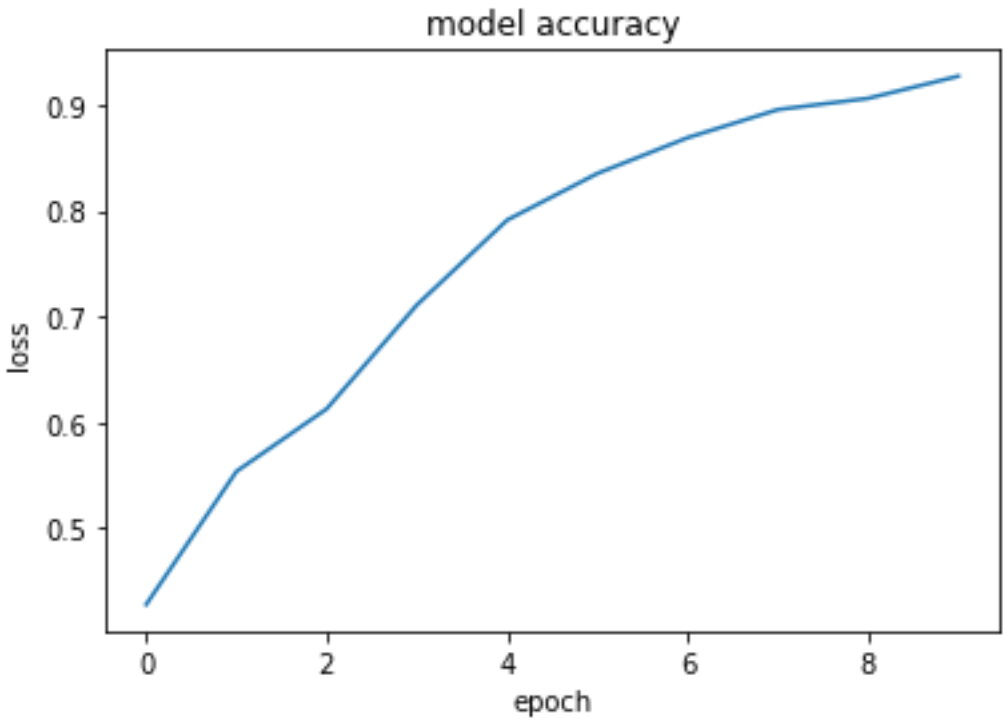


Fig 4.2.3 CNN Model training accuracy evolution

```
plt.plot(history['f1_m'])
plt.title('model f1 score')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
plt.close()
```

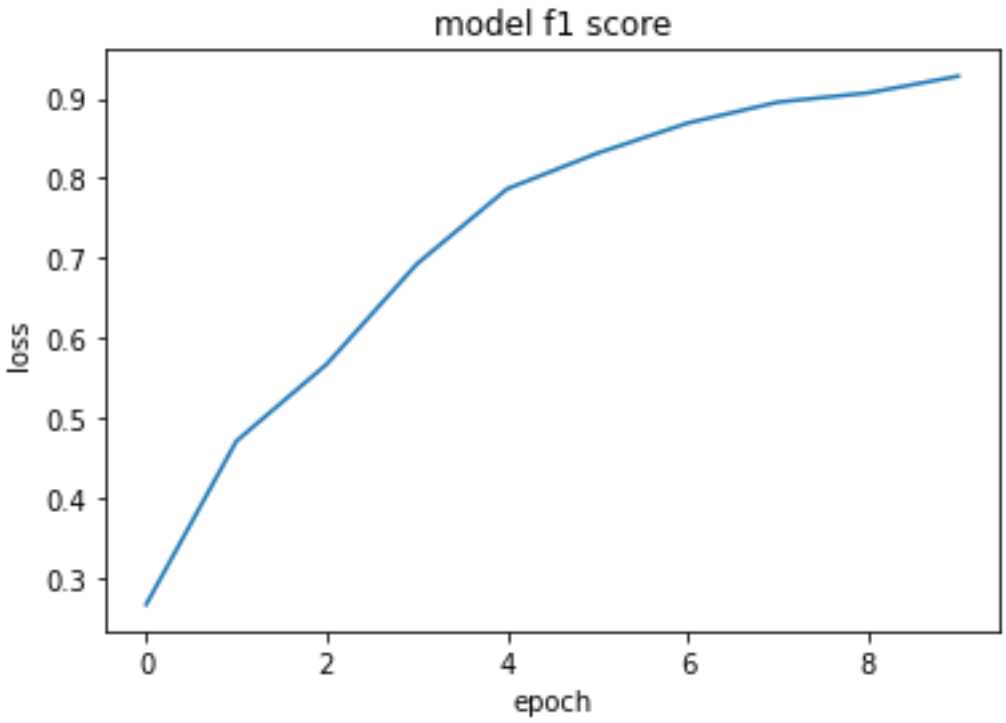


Fig 4.2.4 CNN Model training F1 score evolution

```
plt.plot(history['precision_m'])
plt.plot(history['recall_m'])
plt.title('model precision and recall')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['precision','recall'])
plt.show()
plt.close()
```

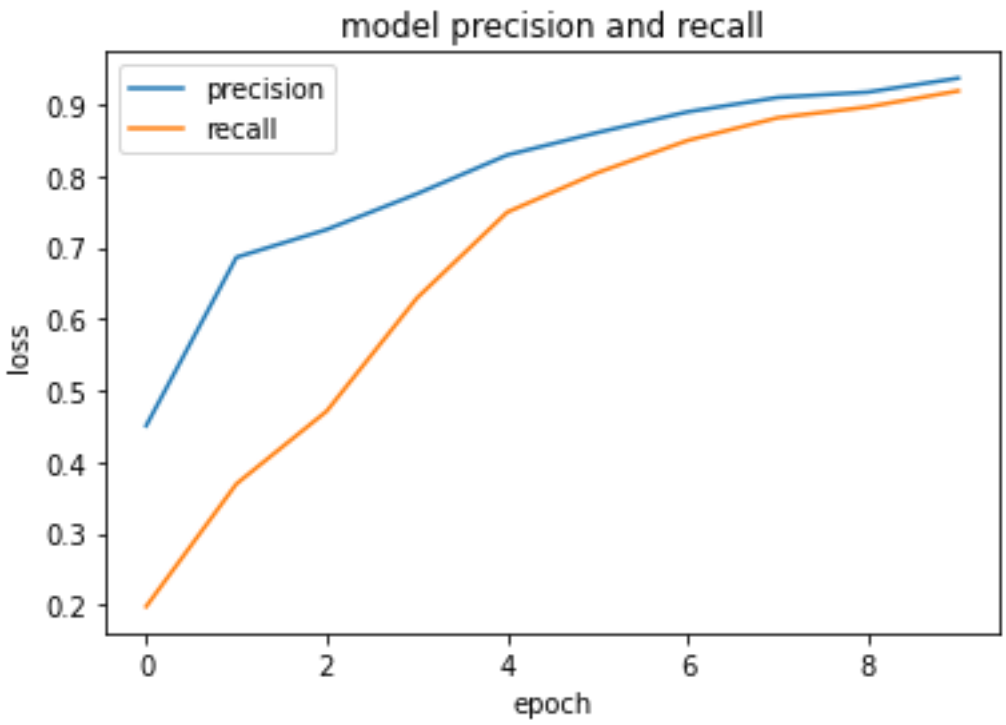


Fig 4.2.5 CNN Model training precision and recall evolution

For AccentDB database

```
import librosa

import matplotlib.pyplot

import os

import pandas as pd

import numpy as np

import tensorflow as tf

from keras.utils import to_categorical
```

Defining the generator

```
class AudioDataGenerator(tf.keras.utils.Sequence):
```

```
    def __init__(self, csv_file, batch_size=32, num_classes=4,
sample_rate=22050, duration=10.0, shuffle=True, n_mels=128):
        self.csv_file = csv_file
        self.batch_size = batch_size
        self.num_classes = num_classes
        self.sample_rate = sample_rate
        self.duration = duration
        self.shuffle = shuffle
        self.n_mels  = n_mels

        # Read the CSV file
        self.data = pd.read_csv(csv_file)

        # Get the unique class labels
        self.classes = sorted(self.data['class_label'].unique())

        # Create a dictionary to map class labels to integers
        self.class_to_int = dict(zip(self.classes,
range(len(self.classes))))

        # Shuffle the data if requested
        if self.shuffle:
            self.data =
self.data.sample(frac=1).reset_index(drop=True)

    def __len__(self):
```

```
        # Return the number of batches
        return int(np.ceil(len(self.data) / float(self.batch_size)))

    def __getitem__(self, idx):
        # Get the batch of file paths and labels
        batch_data = self.data[idx * self.batch_size:(idx + 1) *
self.batch_size]
        batch_data = batch_data.reset_index()

        # Initialize the arrays for the audio data and labels
        batch_x = np.zeros((len(batch_data), self.n_mels, 431,1))
        batch_y = np.zeros((len(batch_data), self.num_classes))

        # Load the audio files and their corresponding labels
        for i, row in batch_data.iterrows():
            file_path = row['file_path']
            class_label = row['class_label']

            # Load the audio file
            signal, sr = librosa.load(file_path, sr=self.sample_rate,
mono=True)

            # Pad or truncate the signal to the desired length
            signal = librosa.util.fix_length(signal,
size=self.sample_rate * self.duration)
```

```
            # Convert the audio file to spectrogram
            S = librosa.feature.melspectrogram(y=signal, sr=sr)
            S_dB = np.array(librosa.power_to_db(S, ref=np.max))
            S_dB = S_dB.reshape(S_dB.shape[0],S_dB.shape[1],1)

            # Save the audio data and label to the batch arrays
            batch_x[i, :] = S_dB
            batch_y[i, :] =
to_categorical(self.class_to_int[class_label],
num_classes=self.num_classes)

        return batch_x, batch_y
train_data = pd.read_csv("train_data.csv")
val_data = pd.read_csv("val_data.csv")
```

Instantiating the generator

```
train_generator = AudioDataGenerator('train_data.csv', batch_size=32,
num_classes=4, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Defining the metrics

```
from keras import backend as K

def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
```

```
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Defining the model

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation,
Flatten, Conv2D, MaxPooling2D
model = Sequential()

model.add(Conv2D(8,(8,8),input_shape=(128,431,1),activation='relu',
padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
model.add(Conv2D(16,(3,3), activation='relu', padding='same'))

model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Conv2D(32,(3,3), activation='relu', padding='same'))

model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Flatten())


model.add(Dense(4,activation='softmax'))


model.compile(loss='categorical_crossentropy',optimizer='adam',metrics
=['accuracy',f1_m,precision_m, recall_m])

model.compile(loss='categorical_crossentropy',optimizer='adam',metrics
=['accuracy',f1_m,precision_m, recall_m])
```

Training the model

```
history = model.fit(train_generator,batch_size=32,epochs=10,verbose=1)
```

Instantiating validation data generator

```
val_generator = AudioDataGenerator('val_data.csv', batch_size=32,
num_classes=4, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Evaluating the model

```
loss, accuracy, f1_score, precision, recall =
model.evaluate(val_generator)
```

Plotting training history

```
import matplotlib.pyplot as plt


plt.plot(history['loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.show()

plt.close()
```

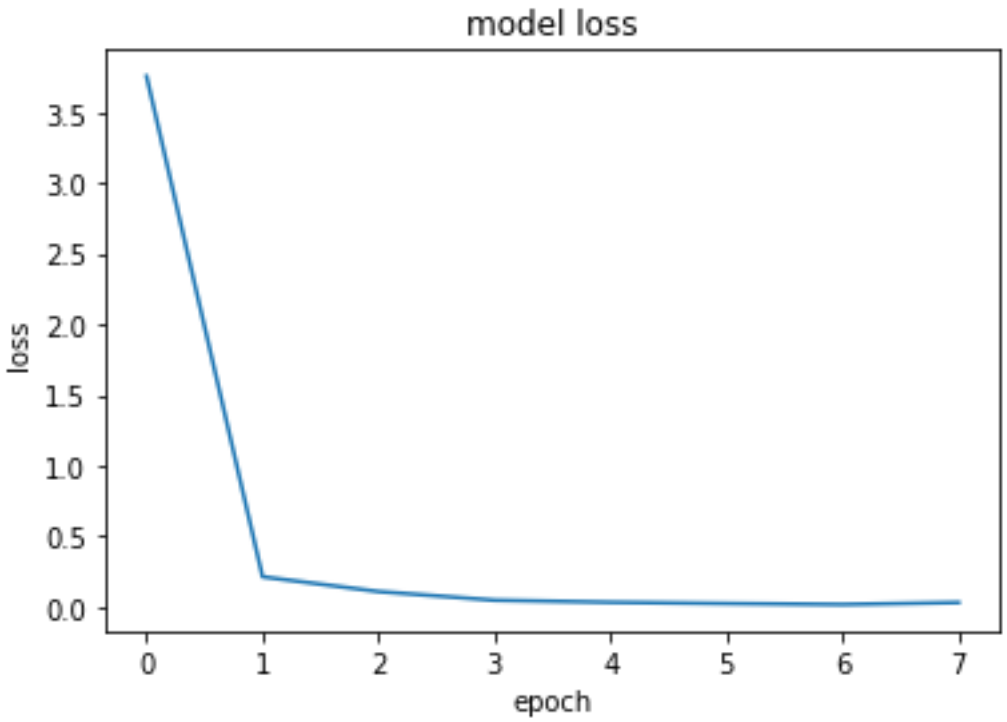


Fig 4.2.6 CNN Model training loss evolution

```
plt.plot(history['accuracy'])
plt.title('model accuracy')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
plt.close()
```

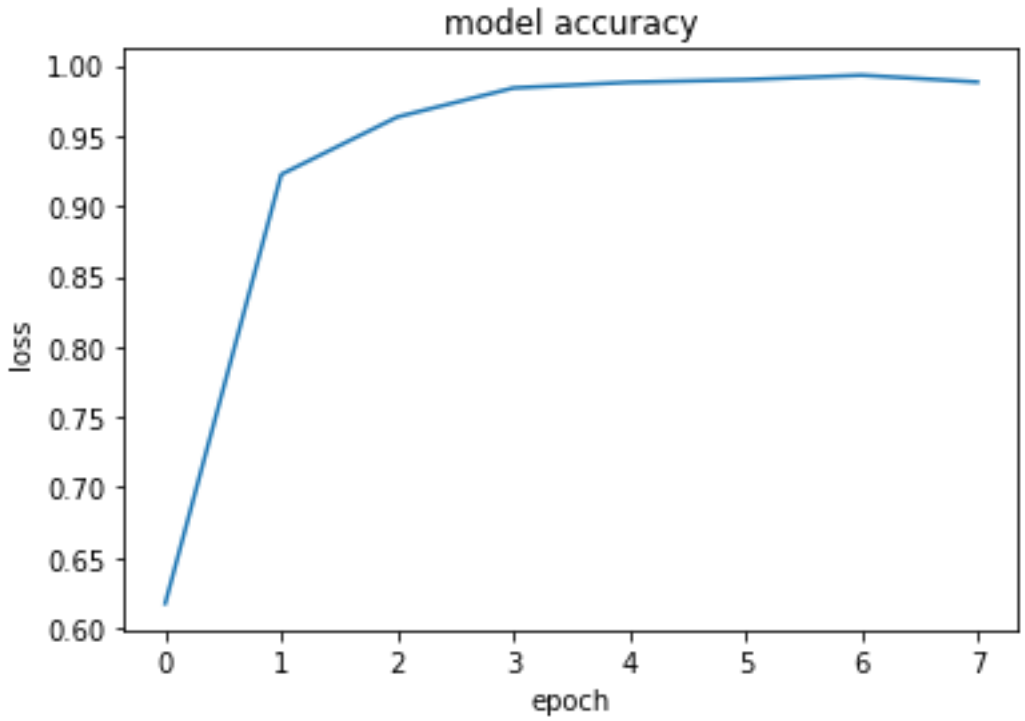


Fig 4.2.7 CNN Model training accuracy evolution

```
plt.plot(history['f1_m'])
plt.title('model f1 score')
plt.ylabel('loss')
plt.xlabel('epoch')
```

```
plt.show()

plt.close()
```

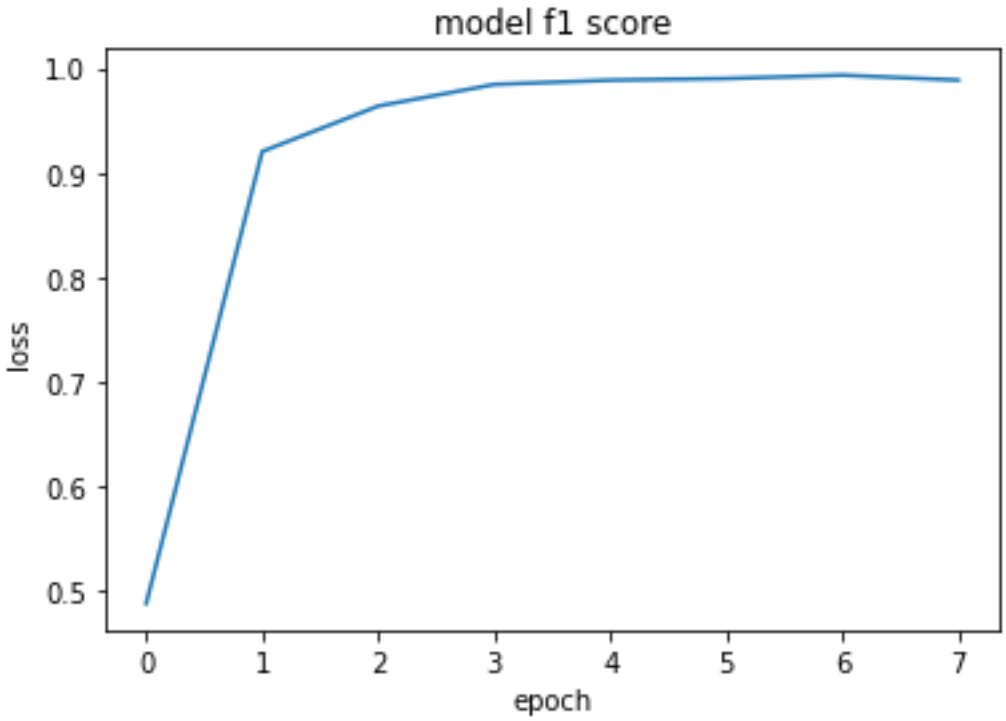


Fig 4.2.8 CNN Model training F1 score evolution

```
plt.plot(history['precision_m'])

plt.plot(history['recall_m'])

plt.title('model precision and recall')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['precision','recall'])

plt.show()

plt.close()
```

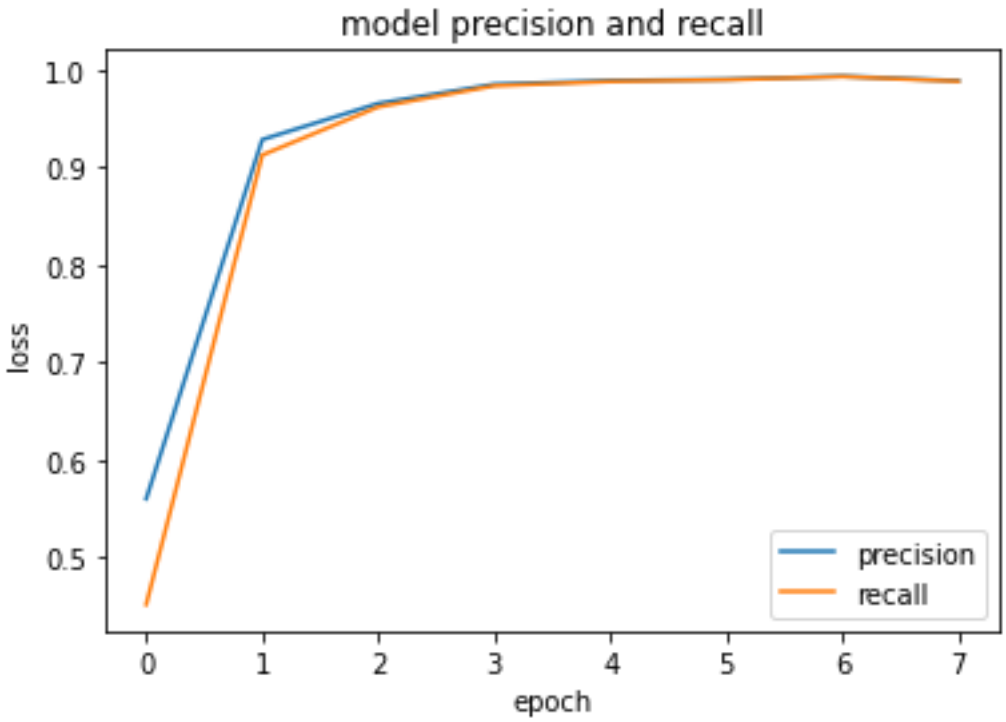
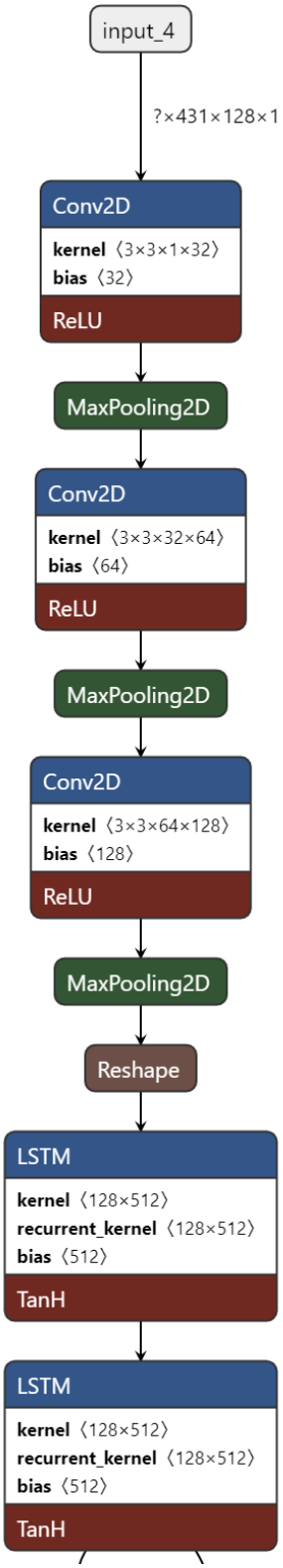


Fig 4.2.9 CNN Model training precision and recall evolution

4.3 Implementing CNN-LSTM with attention for OpenSLR database



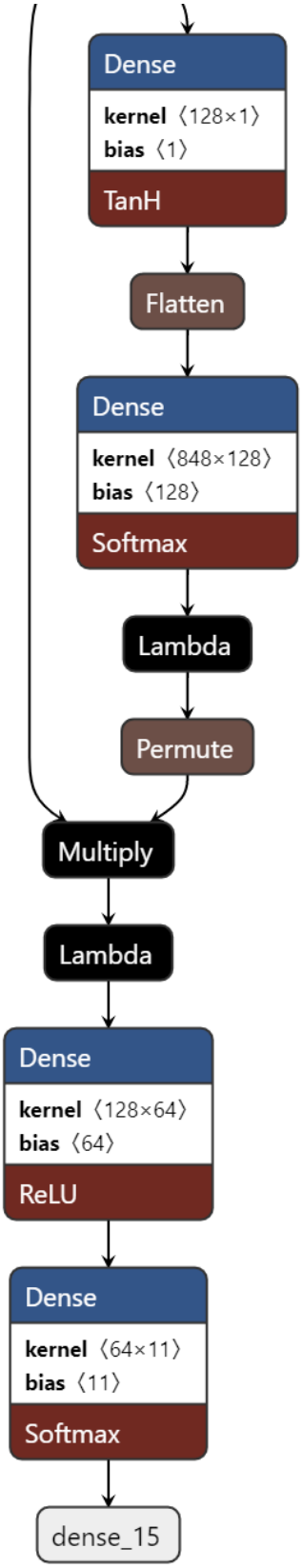


Fig 4.3.1 Architecture of CNN-LSTM model

For OpenSLR dataset

```
import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras.layers import Dense, Flatten, Reshape, Conv2D,
MaxPooling2D, Conv1D, MaxPooling1D, LSTM

from keras.utils import to_categorical

from tensorflow.keras.models import Sequential

import librosa
```

Defining the generator

```
class AudioDataGenerator(tf.keras.utils.Sequence):

    def __init__(self, csv_file, batch_size=32, num_classes=4,
sample_rate=22050, duration=10.0, shuffle=True, n_mels=128):

        self.csv_file = csv_file

        self.batch_size = batch_size

        self.num_classes = num_classes

        self.sample_rate = sample_rate

        self.duration = duration

        self.shuffle = shuffle

        self.n_mels  = n_mels


        # Read the CSV file

        self.data = pd.read_csv(csv_file)


        # Get the unique class labels
```

```
        self.classes = sorted(self.data['class_label'].unique())

        # Create a dictionary to map class labels to integers
        self.class_to_int = dict(zip(self.classes,
range(len(self.classes))))

        # Shuffle the data if requested
        if self.shuffle:
            self.data =
self.data.sample(frac=1).reset_index(drop=True)

    def __len__(self):
        # Return the number of batches
        return int(np.ceil(len(self.data) / float(self.batch_size)))

    def __getitem__(self, idx):
        # Get the batch of file paths and labels
        batch_data = self.data[idx * self.batch_size:(idx + 1) *
self.batch_size]
        batch_data = batch_data.reset_index()
        # Initialize the arrays for the audio data and labels
        batch_x = np.zeros((len(batch_data), 431, self.n_mels,1))
        batch_y = np.zeros((len(batch_data), self.num_classes))

        # Load the audio files and their corresponding labels
        for i, row in batch_data.iterrows():
```

```
            file_path = row['file_path']

            class_label = row['class_label']


            # Load the audio file

            signal, sr = librosa.load(file_path, sr=self.sample_rate,
mono=True)


            # Pad or truncate the signal to the desired length

            signal = librosa.util.fix_length(signal,
size=self.sample_rate * self.duration)


            # Convert the audio file to spectrogram

            S = librosa.feature.melspectrogram(y=signal, sr=sr,
n_mels=self.n_mels)

            S_dB = np.array(librosa.power_to_db(S, ref=np.max))

            S_dB = np.rot90(S_dB, k=3)

            S_dB = S_dB.reshape(S_dB.shape[0],S_dB.shape[1],1)


            # Save the audio data and label to the batch arrays

            batch_x[i, :] = S_dB

            batch_y[i, :] =
to_categorical(self.class_to_int[class_label],
num_classes=self.num_classes)



        return batch_x, batch_yy
```

```
train_data = pd.read_csv("train_data.csv")

val_data = pd.read_csv("val_data.csv")
```

Instantiating the generator

```
train_generator = AudioDataGenerator('train_data.csv', batch_size=32,
num_classes=4, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Defining the metrics

```
from keras import backend as K


def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall


def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision


def f1_m(y_true, y_pred):
```

```
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Defining the model

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
Flatten, LSTM, Dense, Multiply, Reshape, Permute, Lambda


# Define the input shape of the spectrogram (time_steps,
frequency_bins, channels)
input_shape = (431, 128, 1)


# Define the input tensor
inputs = Input(shape=input_shape)


# Define the CNN layers
x = Conv2D(8, kernel_size=(3, 3), activation='relu',
padding='same')(inputs)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(16, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(32, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
```

```
# Reshape the output of the CNN layers to be a sequence of feature
vectors
x = Reshape((-1, 128))(x)

# Define the LSTM layers
x = LSTM(128, return_sequences=True)(x)
x = LSTM(128, return_sequences=True)(x)

# Define the attention mechanism
att = Dense(1, activation='tanh')(x)
att = Flatten()(att)
att = Dense(128, activation='softmax')(att)
att = Lambda(lambda x: K.expand_dims(x))(att)
att = Permute((2, 1))(att)
x = Multiply()([x, att])
x = Lambda(lambda x: K.sum(x, axis=1))(x)

# Define the classification layers
x = Dense(64, activation='relu')(x)
predictions = Dense(11, activation='softmax')(x)

# Define the model
model = Model(inputs=inputs, outputs=predictions)

# Compile the model
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy',f1_m,precision_m, recall_m])
```

Training the model

```
history = model.fit(train_generator,batch_size=32,epochs=20,verbose=1)
```

Instantiating validation data generator

```
val_generator = AudioDataGenerator('val_data.csv', batch_size=32,
num_classes=4, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Evaluating the model

```
loss, accuracy, f1_score, precision, recall =
model.evaluate(val_generator)
```

Plotting training history

```
import matplotlib.pyplot as plt


plt.plot(history['loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.show()

plt.close()
```

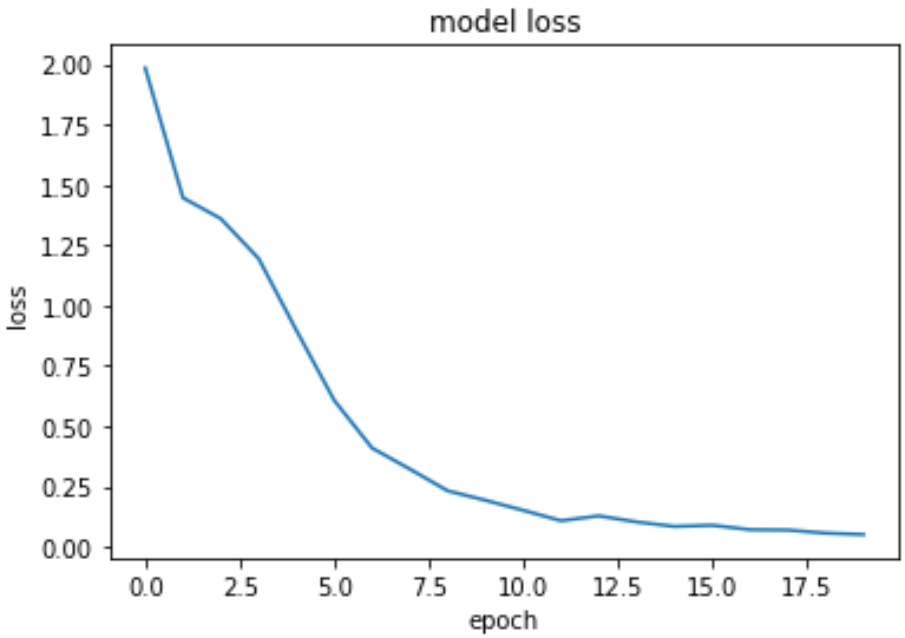


Fig 4.3.2 CNN-LSTM Model training loss evolution

```
plt.plot(history['accuracy'])
plt.title('model accuracy')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
plt.close()
```

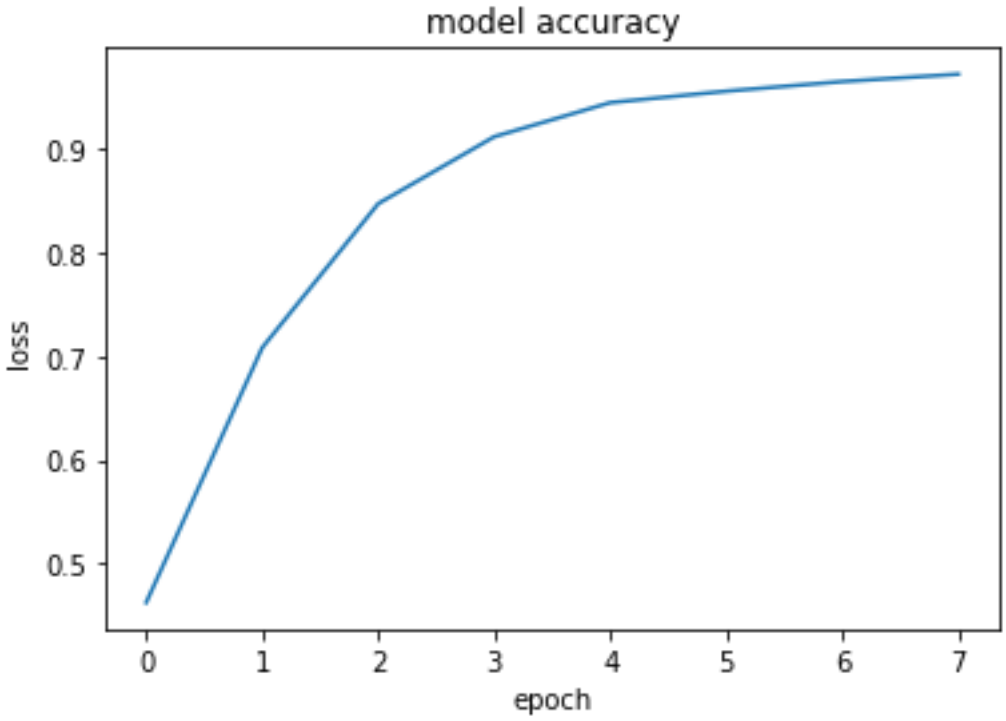


Fig 4.3.3 CNN-LSTM Model training accuracy evolution

```
plt.plot(history['f1_m'])
plt.title('model f1 score')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
plt.close()
```

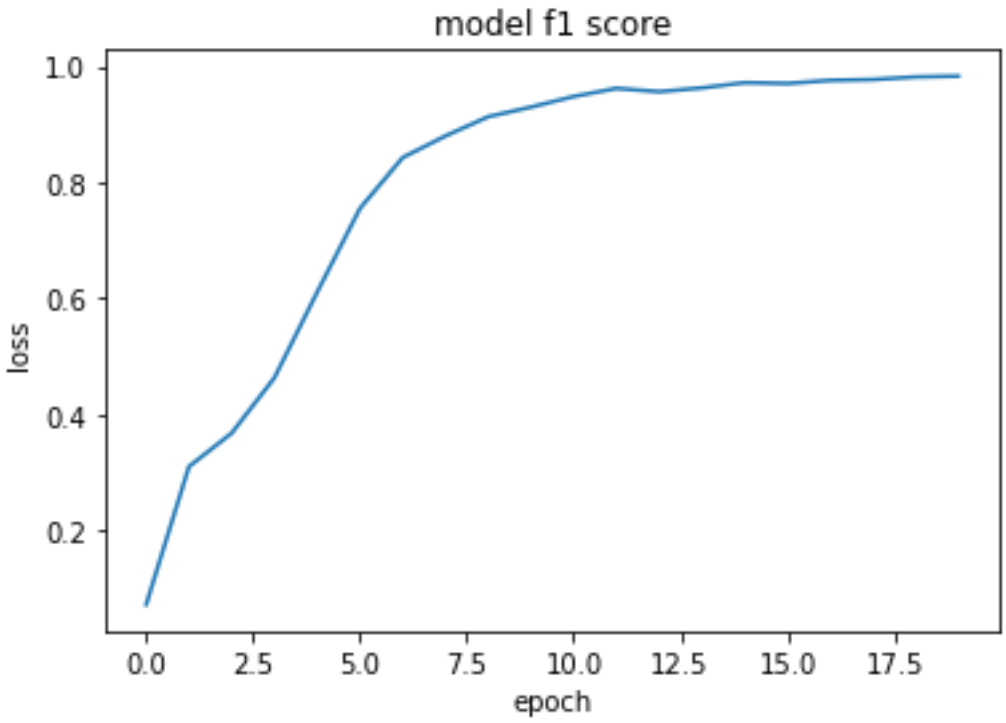


Fig 4.3.4 CNN-LSTM Model training F1 score evolution

```
plt.plot(history['precision_m'])
plt.plot(history['recall_m'])
plt.title('model precision and recall')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['precision','recall'])
plt.show()
plt.close()
```

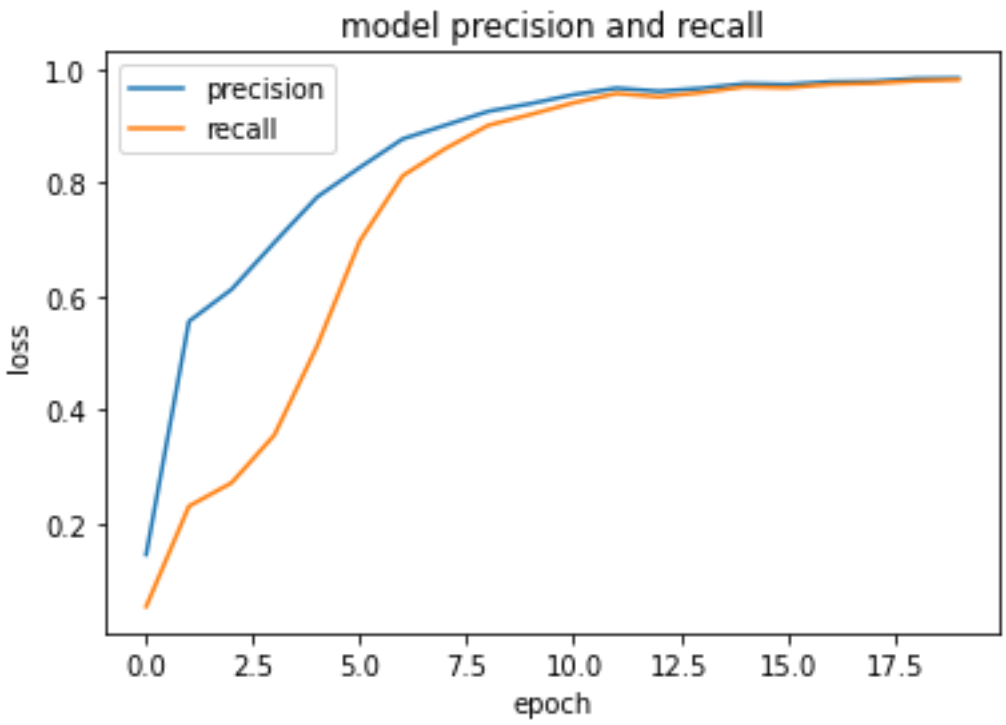


Fig 4.3.5 CNN-LSTM Model training precision and recall evolution

For AccentDB dataset

```
import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras.layers import Dense, Flatten, Reshape, Conv2D,
MaxPooling2D, Conv1D, MaxPooling1D, LSTM

from keras.utils import to_categorical

from tensorflow.keras.models import Sequential

import librosa
```

Defining the generator

```
class AudioDataGenerator(tf.keras.utils.Sequence):
```

```
    def __init__(self, csv_file, batch_size=32, num_classes=11,
sample_rate=22050, duration=10.0, shuffle=True, n_mels=128):
        self.csv_file = csv_file
        self.batch_size = batch_size
        self.num_classes = num_classes
        self.sample_rate = sample_rate
        self.duration = duration
        self.shuffle = shuffle
        self.n_mels  = n_mels

        # Read the CSV file
        self.data = pd.read_csv(csv_file)

        # Get the unique class labels
        self.classes = sorted(self.data['class_label'].unique())

        # Create a dictionary to map class labels to integers
        self.class_to_int = dict(zip(self.classes,
range(len(self.classes))))

        # Shuffle the data if requested
        if self.shuffle:
            self.data =
self.data.sample(frac=1).reset_index(drop=True)

    def __len__(self):
```

```
        # Return the number of batches
        return int(np.ceil(len(self.data) / float(self.batch_size)))

    def __getitem__(self, idx):
        # Get the batch of file paths and labels
        batch_data = self.data[idx * self.batch_size:(idx + 1) *
self.batch_size]
        batch_data = batch_data.reset_index()
        # Initialize the arrays for the audio data and labels
        batch_x = np.zeros((len(batch_data), 431, self.n_mels,1))
        batch_y = np.zeros((len(batch_data), self.num_classes))

        # Load the audio files and their corresponding labels
        for i, row in batch_data.iterrows():
            file_path = row['file_path']
            class_label = row['class_label']

            # Load the audio file
            signal, sr = librosa.load(file_path, sr=self.sample_rate,
mono=True)

            # Pad or truncate the signal to the desired length
            signal = librosa.util.fix_length(signal,
size=self.sample_rate * self.duration)

            # Convert the audio file to spectrogram
```

```
            S = librosa.feature.melspectrogram(y=signal, sr=sr,
n_mels=self.n_mels)

            S_dB = np.array(librosa.power_to_db(S, ref=np.max))

            S_dB = np.rot90(S_dB, k=3)

            S_dB = S_dB.reshape(S_dB.shape[0],S_dB.shape[1],1)


            # Save the audio data and label to the batch arrays

            batch_x[i, :] = S_dB

            batch_y[i, :] =
to_categorical(self.class_to_int[class_label],
num_classes=self.num_classes)



        return batch_x, batch_yy


train_data = pd.read_csv("train_data.csv")

val_data = pd.read_csv("val_data.csv")
```

Instantiating the generator

```
train_generator = AudioDataGenerator('train_data.csv', batch_size=32,
num_classes=11, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Defining the metrics

```
from keras import backend as K
```

```
def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Defining the model

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
Flatten, LSTM, Dense, Multiply, Reshape, Permute, Lambda


# Define the input shape of the spectrogram (time_steps,
frequency_bins, channels)
```

```
input_shape = (431, 128, 1)


# Define the input tensor

inputs = Input(shape=input_shape)


# Define the CNN layers

x = Conv2D(32, kernel_size=(3, 3), activation='relu',
padding='same')(inputs)

x = MaxPooling2D(pool_size=(2, 2))(x)

x = Conv2D(64, kernel_size=(3, 3), activation='relu',
padding='same')(x)

x = MaxPooling2D(pool_size=(2, 2))(x)

x = Conv2D(128, kernel_size=(3, 3), activation='relu',
padding='same')(x)

x = MaxPooling2D(pool_size=(2, 2))(x)


# Reshape the output of the CNN layers to be a sequence of feature
vectors

x = Reshape((-1, 128))(x)


# Define the LSTM layers

x = LSTM(128, return_sequences=True)(x)

x = LSTM(128, return_sequences=True)(x)


# Define the attention mechanism

att = Dense(1, activation='tanh')(x)
```

```
att = Flatten()(att)

att = Dense(128, activation='softmax')(att)

att = Lambda(lambda x: K.expand_dims(x))(att)

att = Permute((2, 1))(att)

x = Multiply()([x, att])

x = Lambda(lambda x: K.sum(x, axis=1))(x)


# Define the classification layers

x = Dense(64, activation='relu')(x)

predictions = Dense(11, activation='softmax')(x)


# Define the model

model = Model(inputs=inputs, outputs=predictions)


# Compile the model

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy',f1_m,precision_m, recall_m])
```

Training the model

```
history = model.fit(train_generator,batch_size=32,epochs=20,verbose=1)
```

Instantiating validation data generator

```
val_generator = AudioDataGenerator('val_data.csv', batch_size=32,
num_classes=11, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Evaluating the model

```
loss, accuracy, f1_score, precision, recall =
model.evaluate(val_generator)
```

Plotting training history

```
import matplotlib.pyplot as plt


plt.plot(history['loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.show()

plt.close()
```

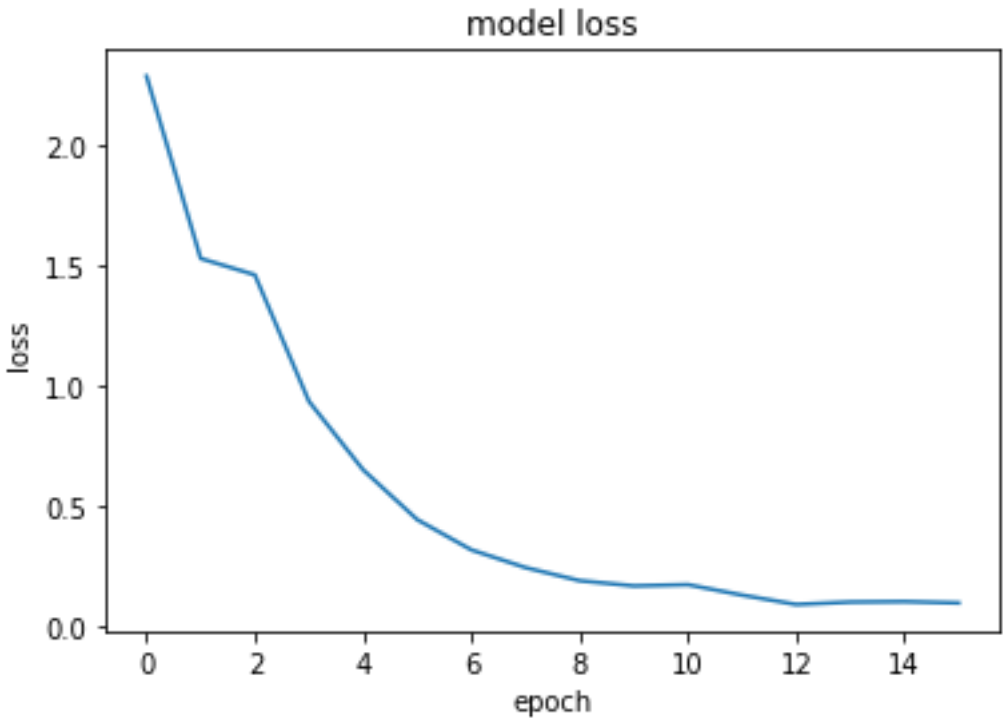


Fig 4.3.6 CNN-LSTM Model training loss evolution

```
plt.plot(history['accuracy'])
plt.title('model accuracy')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
plt.close()
```

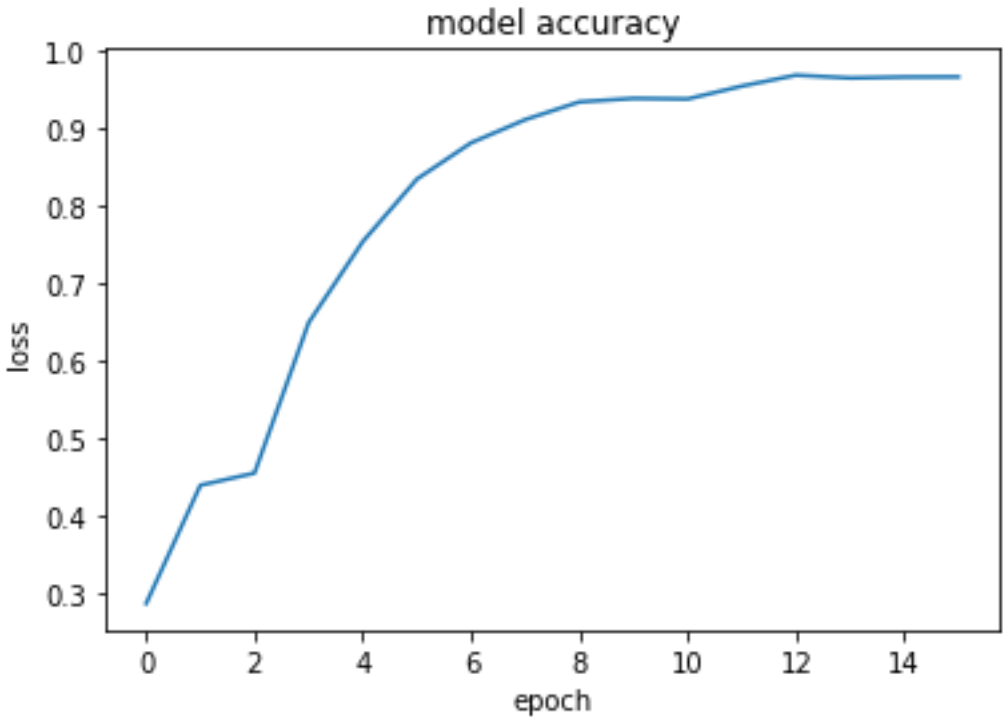


Fig 4.3.7 CNN-LSTM Model training accuracy evolution

```
plt.plot(history['f1_m'])
plt.title('model f1 score')
plt.ylabel('loss')
plt.xlabel('epoch')
```

```
plt.show()

plt.close()
```

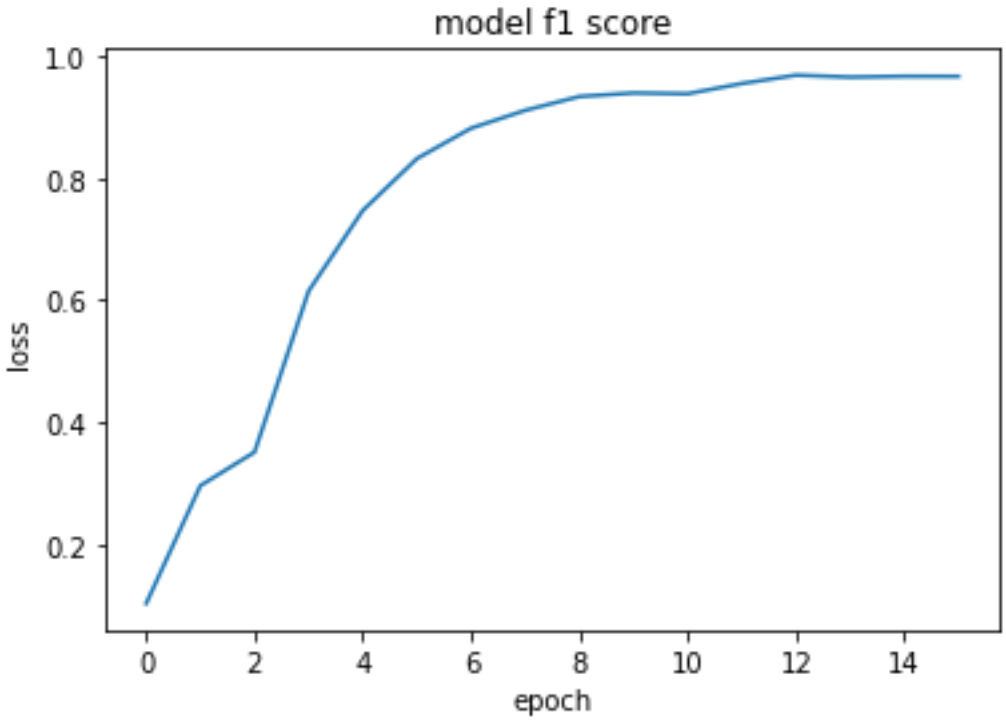


Fig 4.3.8 CNN-LSTM Model training F1 score evolution

```
plt.plot(history['precision_m'])

plt.plot(history['recall_m'])

plt.title('model precision and recall')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['precision','recall'])

plt.show()

plt.close()
```

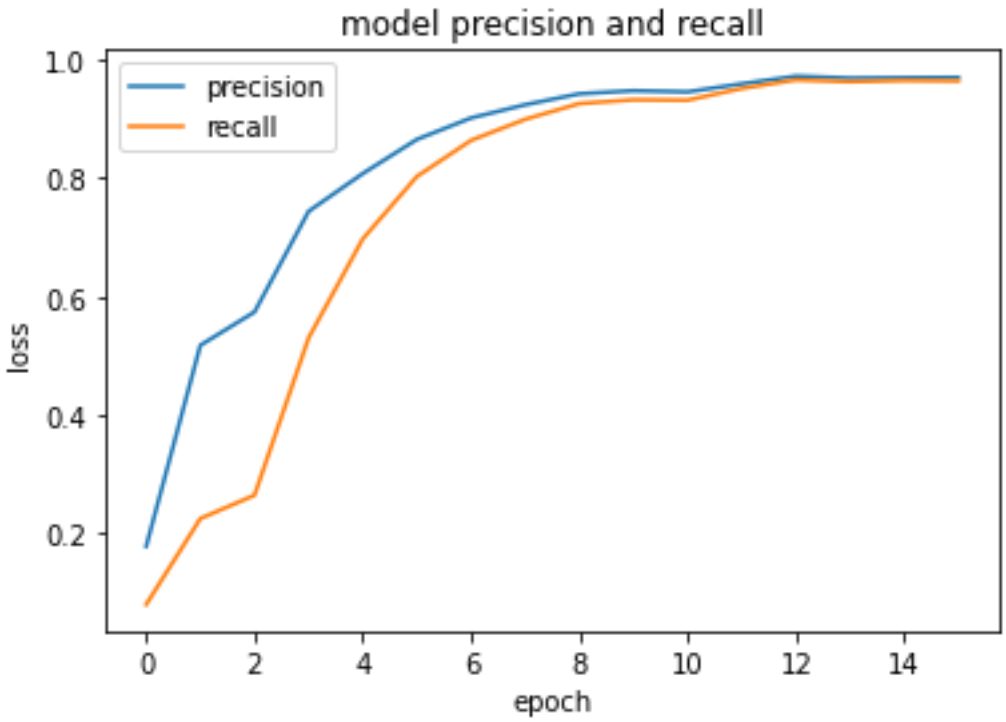
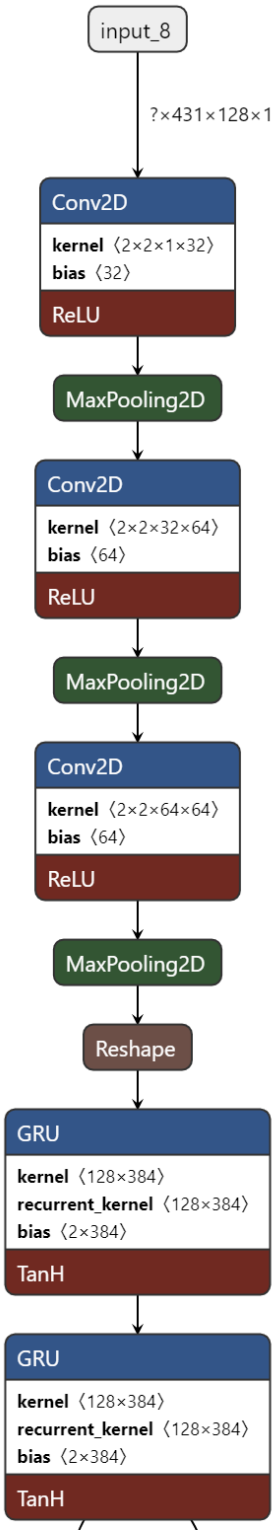


Fig 4.3.9 CNN-LSTM Model training precision and recall evolution

4.4 Implementing CNN-GRU with attention



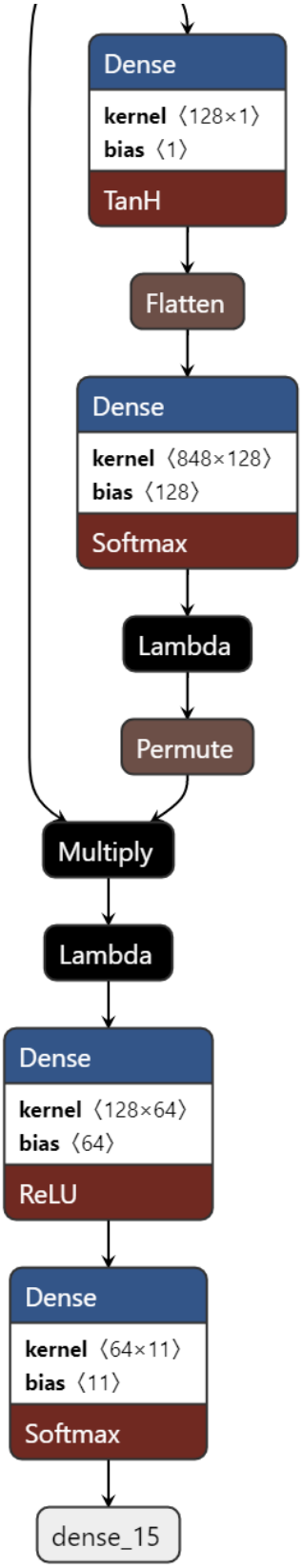


Fig 4.4.1 Architecture of CNN-GRU model

For OpenSLR Dataset

```
import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras.layers import Dense, Flatten, Reshape, Conv2D,
MaxPooling2D, Conv1D, MaxPooling1D, GRU

from keras.utils import to_categorical

from tensorflow.keras.models import Sequential

import librosa
```

Defining the generator

```
class AudioDataGenerator(tf.keras.utils.Sequence):

    def __init__(self, csv_file, batch_size=32, num_classes=11,
sample_rate=22050, duration=10.0, shuffle=True, n_mels=128):

        self.csv_file = csv_file

        self.batch_size = batch_size

        self.num_classes = num_classes

        self.sample_rate = sample_rate

        self.duration = duration

        self.shuffle = shuffle

        self.n_mels  = n_mels


        # Read the CSV file

        self.data = pd.read_csv(csv_file)


        # Get the unique class labels
```

```
        self.classes = sorted(self.data['class_label'].unique())

        # Create a dictionary to map class labels to integers
        self.class_to_int = dict(zip(self.classes,
range(len(self.classes))))

        # Shuffle the data if requested
        if self.shuffle:
            self.data =
self.data.sample(frac=1).reset_index(drop=True)

    def __len__(self):
        # Return the number of batches
        return int(np.ceil(len(self.data) / float(self.batch_size)))

    def __getitem__(self, idx):
        # Get the batch of file paths and labels
        batch_data = self.data[idx * self.batch_size:(idx + 1) *
self.batch_size]
        batch_data = batch_data.reset_index()
        # Initialize the arrays for the audio data and labels
        batch_x = np.zeros((len(batch_data), 431, self.n_mels,1))
        batch_y = np.zeros((len(batch_data), self.num_classes))

        # Load the audio files and their corresponding labels
        for i, row in batch_data.iterrows():
```

```
            file_path = row['file_path']

            class_label = row['class_label']


            # Load the audio file

            signal, sr = librosa.load(file_path, sr=self.sample_rate,
mono=True)


            # Pad or truncate the signal to the desired length

            signal = librosa.util.fix_length(signal,
size=self.sample_rate * self.duration)


            # Convert the audio file to spectrogram

            S = librosa.feature.melspectrogram(y=signal, sr=sr,
n_mels=self.n_mels)

            S_dB = np.array(librosa.power_to_db(S, ref=np.max))

            S_dB = np.rot90(S_dB, k=3)

            S_dB = S_dB.reshape(S_dB.shape[0],S_dB.shape[1],1)


            # Save the audio data and label to the batch arrays

            batch_x[i, :] = S_dB

            batch_y[i, :] =
to_categorical(self.class_to_int[class_label],
num_classes=self.num_classes)



        return batch_x, batch_yy
```

```
train_data = pd.read_csv("train_data.csv")

val_data = pd.read_csv("val_data.csv")
```

Instantiating the generator

```
train_generator = AudioDataGenerator('train_data.csv', batch_size=32,
num_classes=11, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Defining the metrics

```
from keras import backend as K


def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall


def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision


def f1_m(y_true, y_pred):
```

```
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Defining the model

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
Flatten, LSTM, Dense, Multiply, Reshape, Permute, Lambda


# Define the input shape of the spectrogram (time_steps,
frequency_bins, channels)
input_shape = (431, 128, 1)


# Define the input tensor
inputs = Input(shape=input_shape)


# Define the CNN layers
x = Conv2D(32, kernel_size=(3, 3), activation='relu',
padding='same')(inputs)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(64, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(128, kernel_size=(3, 3), activation='relu',
padding='same')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
```

```
# Reshape the output of the CNN layers to be a sequence of feature
vectors
x = Reshape((-1, 128))(x)

# Define the LSTM layers
x = GRU(128, return_sequences=True)(x)
x = GRU(128, return_sequences=True)(x)

# Define the attention mechanism
att = Dense(1, activation='tanh')(x)
att = Flatten()(att)
att = Dense(128, activation='softmax')(att)
att = Lambda(lambda x: K.expand_dims(x))(att)
att = Permute((2, 1))(att)
x = Multiply()([x, att])
x = Lambda(lambda x: K.sum(x, axis=1))(x)

# Define the classification layers
x = Dense(64, activation='relu')(x)
predictions = Dense(11, activation='softmax')(x)

# Define the model
model = Model(inputs=inputs, outputs=predictions)

# Compile the model
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy',f1_m,precision_m, recall_m])
```

Training the model

```
history = model.fit(train_generator,batch_size=32,epochs=20,verbose=1)
```

Instantiating validation data generator

```
val_generator = AudioDataGenerator('val_data.csv', batch_size=32,
num_classes=11, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Evaluating the model

```
loss, accuracy, f1_score, precision, recall =
model.evaluate(val_generator)
```

Plotting training history

```
import matplotlib.pyplot as plt


plt.plot(history['loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.show()

plt.close()
```

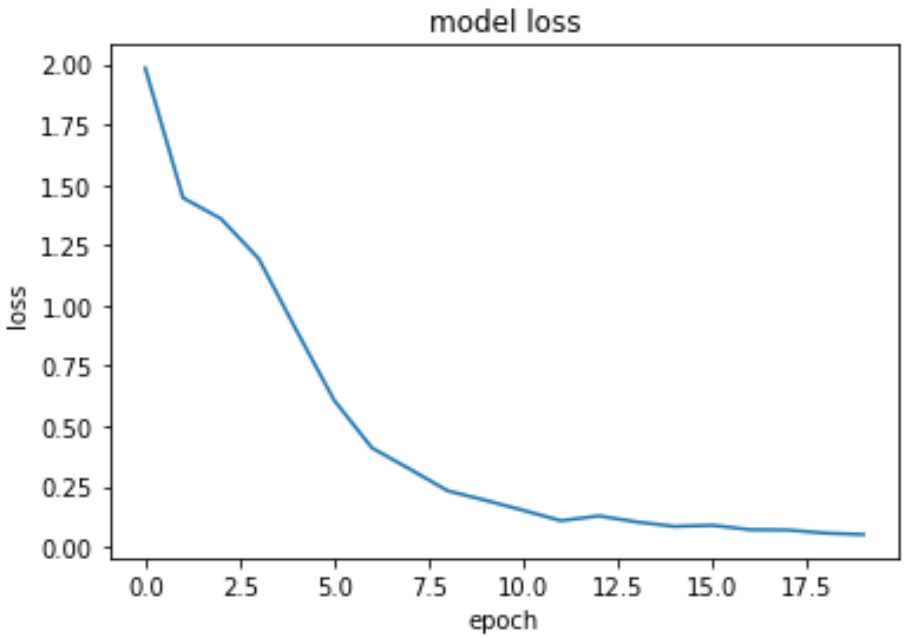


Fig 4.4.2 CNN-GRU Model training loss evolution

```
plt.plot(history['accuracy'])
plt.title('model accuracy')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
plt.close()
```

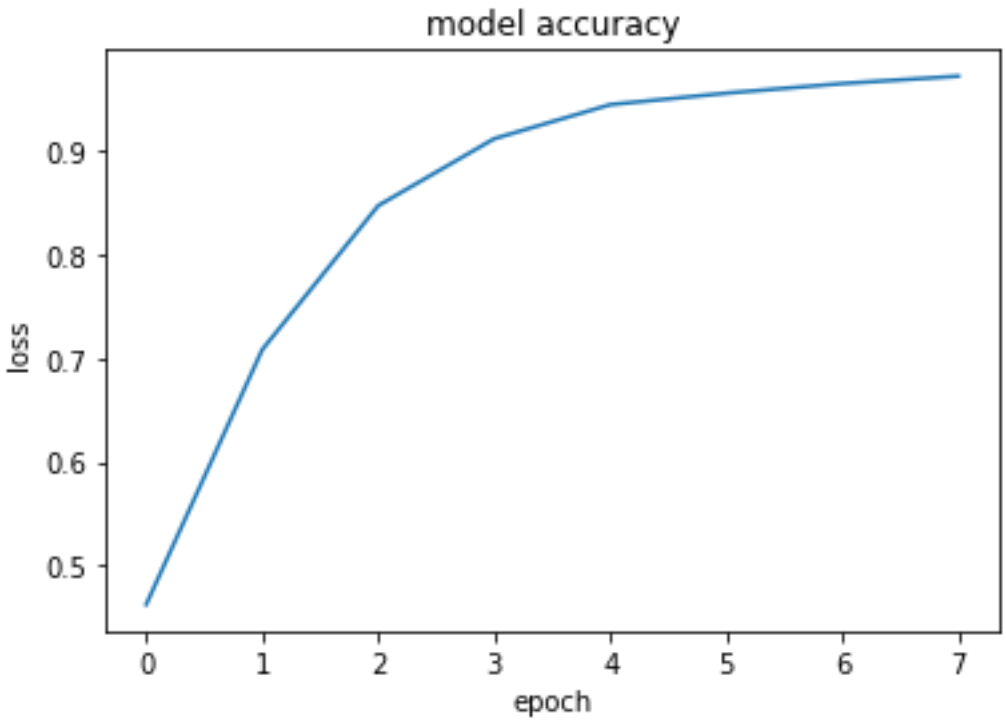


Fig 4.4.3 CNN-GRU Model training accuracy evolution

```
plt.plot(history['f1_m'])
plt.title('model f1 score')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
plt.close()
```

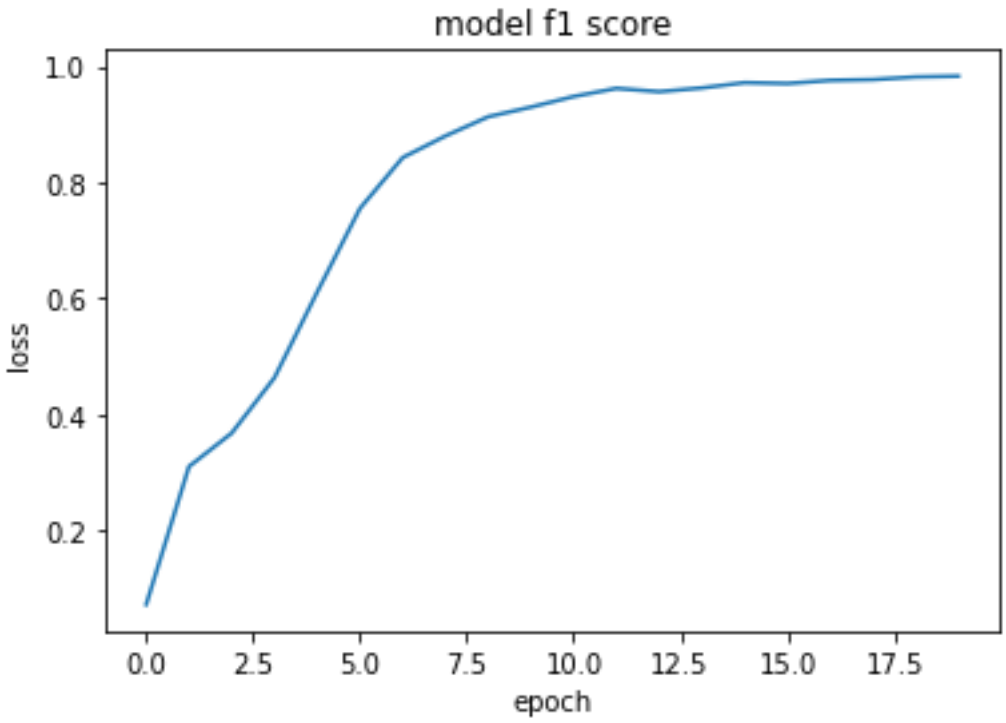


Fig 4.4.4 CNN-GRU Model training F1 score evolution

```
plt.plot(history['precision_m'])
plt.plot(history['recall_m'])
plt.title('model precision and recall')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['precision','recall'])
plt.show()
plt.close()
```

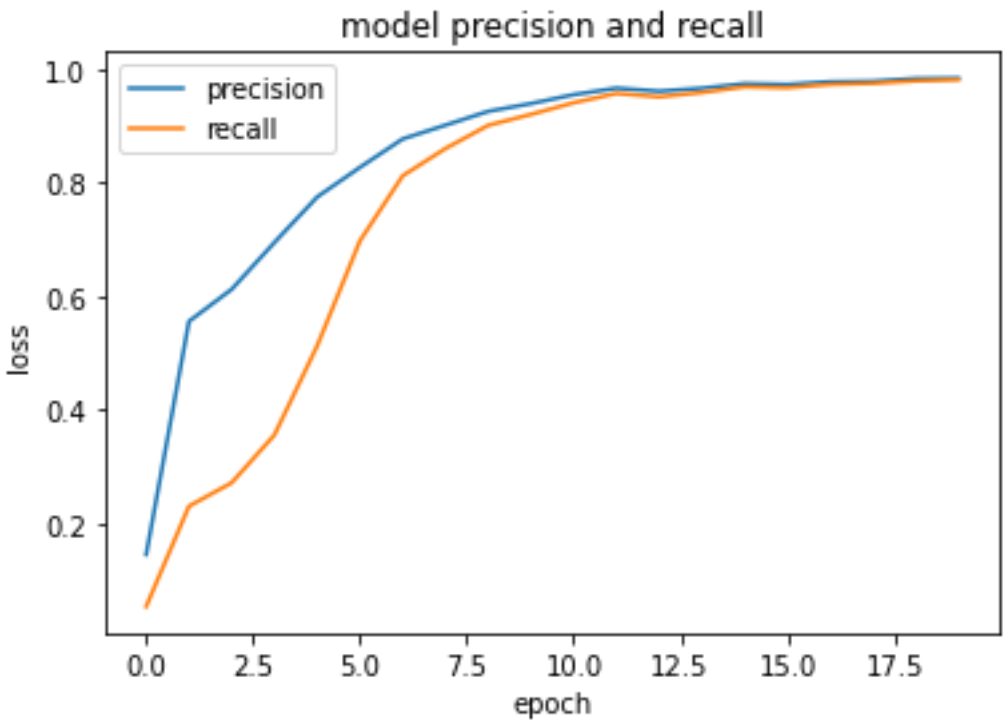


Fig 4.4.5 CNN-GRU Model training precision and recall evolution

For AccentDB Dataset

```
import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras.layers import Dense, Flatten, Reshape, Conv2D,
MaxPooling2D, Conv1D, MaxPooling1D, GRU

from keras.utils import to_categorical

from tensorflow.keras.models import Sequential

import librosa
```

Defining the generator

```
class AudioDataGenerator(tf.keras.utils.Sequence):
```

```
    def __init__(self, csv_file, batch_size=32, num_classes=4,
sample_rate=22050, duration=10.0, shuffle=True, n_mels=128):
        self.csv_file = csv_file
        self.batch_size = batch_size
        self.num_classes = num_classes
        self.sample_rate = sample_rate
        self.duration = duration
        self.shuffle = shuffle
        self.n_mels  = n_mels

        # Read the CSV file
        self.data = pd.read_csv(csv_file)

        # Get the unique class labels
        self.classes = sorted(self.data['class_label'].unique())

        # Create a dictionary to map class labels to integers
        self.class_to_int = dict(zip(self.classes,
range(len(self.classes))))

        # Shuffle the data if requested
        if self.shuffle:
            self.data =
self.data.sample(frac=1).reset_index(drop=True)

    def __len__(self):
```

```
        # Return the number of batches
        return int(np.ceil(len(self.data) / float(self.batch_size)))

    def __getitem__(self, idx):
        # Get the batch of file paths and labels
        batch_data = self.data[idx * self.batch_size:(idx + 1) *
self.batch_size]
        batch_data = batch_data.reset_index()
        # Initialize the arrays for the audio data and labels
        batch_x = np.zeros((len(batch_data), 259, self.n_mels,1))
        batch_y = np.zeros((len(batch_data), self.num_classes))

        # Load the audio files and their corresponding labels
        for i, row in batch_data.iterrows():
            file_path = row['file_path']
            class_label = row['class_label']

            # Load the audio file
            signal, sr = librosa.load(file_path, sr=self.sample_rate,
mono=True)

            # Pad or truncate the signal to the desired length
            signal = librosa.util.fix_length(signal,
size=self.sample_rate * self.duration)

            # Convert the audio file to spectrogram
```

```
            S = librosa.feature.melspectrogram(y=signal, sr=sr,
n_mels=self.n_mels)

            S_dB = np.array(librosa.power_to_db(S, ref=np.max))

            S_dB = np.rot90(S_dB, k=3)

            S_dB = S_dB.reshape(S_dB.shape[0],S_dB.shape[1],1)


            # Save the audio data and label to the batch arrays

            batch_x[i, :] = S_dB

            batch_y[i, :] =
to_categorical(self.class_to_int[class_label],
num_classes=self.num_classes)



        return batch_x, batch_yy


train_data = pd.read_csv("train_data.csv")

val_data = pd.read_csv("val_data.csv")
```

Instantiating the generator

```
train_generator = AudioDataGenerator('train_data.csv', batch_size=32,
num_classes=4, sample_rate=22050, duration=6, shuffle=True,
n_mels=128)
```

Defining the metrics

```
from keras import backend as K
```

```
def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Defining the model

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
Flatten, LSTM, Dense, Multiply, Reshape, Permute, Lambda


# Define the input shape of the spectrogram (time_steps,
frequency_bins, channels)
```

```
input_shape = (259, 128, 1)


# Define the input tensor

inputs = Input(shape=input_shape)


# Define the CNN layers

x = Conv2D(32, kernel_size=(3, 3), activation='relu',
padding='same')(inputs)

x = MaxPooling2D(pool_size=(2, 2))(x)

x = Conv2D(64, kernel_size=(3, 3), activation='relu',
padding='same')(x)

x = MaxPooling2D(pool_size=(2, 2))(x)

x = Conv2D(128, kernel_size=(3, 3), activation='relu',
padding='same')(x)

x = MaxPooling2D(pool_size=(2, 2))(x)


# Reshape the output of the CNN layers to be a sequence of feature
vectors

x = Reshape((-1, 128))(x)


# Define the LSTM layers

x = GRU(128, return_sequences=True)(x)

x = GRU(128, return_sequences=True)(x)


# Define the attention mechanism

att = Dense(1, activation='tanh')(x)
```

```
att = Flatten()(att)

att = Dense(128, activation='softmax')(att)

att = Lambda(lambda x: K.expand_dims(x))(att)

att = Permute((2, 1))(att)

x = Multiply()([x, att])

x = Lambda(lambda x: K.sum(x, axis=1))(x)


# Define the classification layers

x = Dense(64, activation='relu')(x)

predictions = Dense(11, activation='softmax')(x)


# Define the model

model = Model(inputs=inputs, outputs=predictions)


# Compile the model

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy',f1_m,precision_m, recall_m])
```

Training the model

```
history = model.fit(train_generator,batch_size=32,epochs=20,verbose=1)
```

Instantiating validation data generator

```
val_generator = AudioDataGenerator('val_data.csv', batch_size=32,
num_classes=4, sample_rate=22050, duration=6, shuffle=True,
n_mels=128)
```

Evaluating the model

```
loss, accuracy, f1_score, precision, recall =
model.evaluate(val_generator)
```

Plotting training history

```
import matplotlib.pyplot as plt


plt.plot(history['loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.show()

plt.close()
```

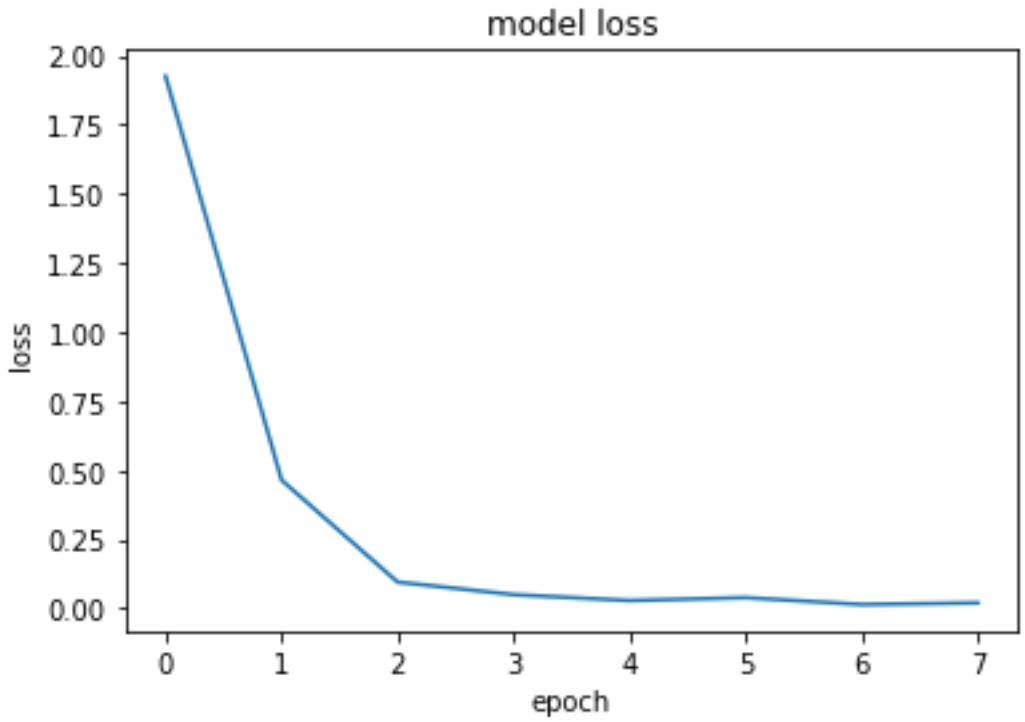


Fig 4.4.6 CNN-GRU Model training loss evolution

```
plt.plot(history['accuracy'])
plt.title('model accuracy')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
plt.close()
```

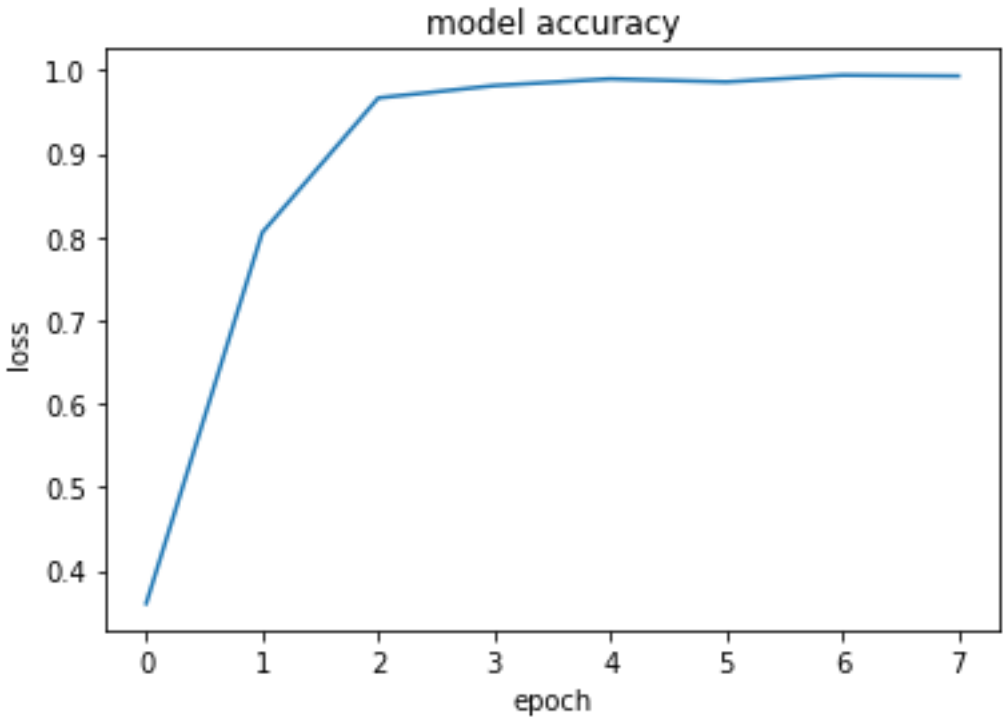


Fig 4.4.7 CNN-GRU Model training accuracy evolution

```
plt.plot(history['f1_m'])
plt.title('model f1 score')
plt.ylabel('loss')
plt.xlabel('epoch')
```

```
plt.show()

plt.close()
```

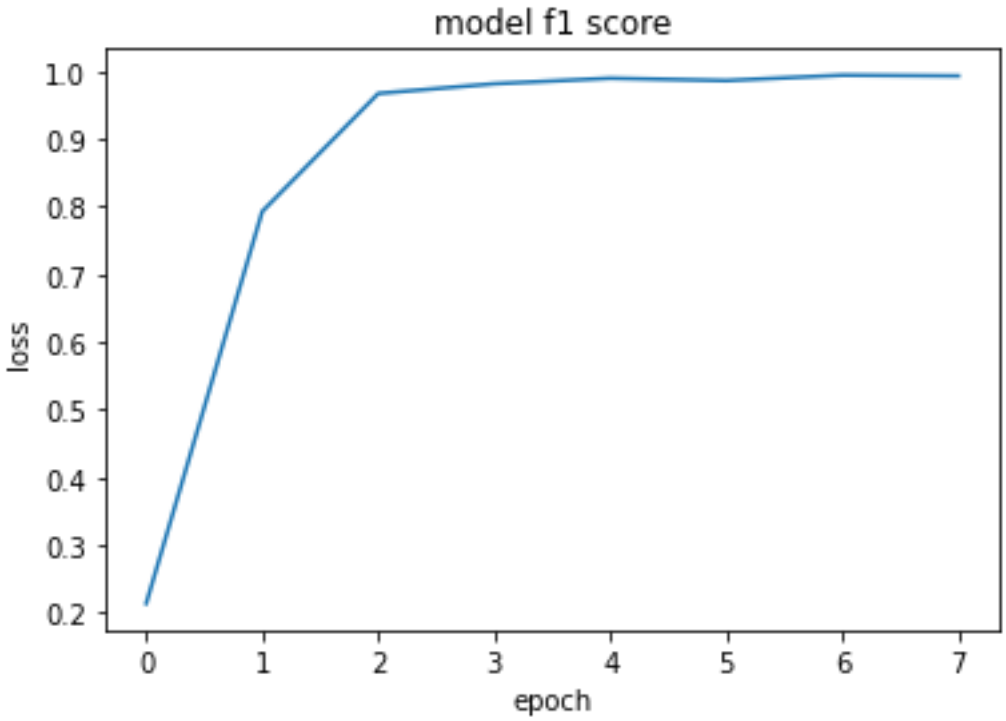


Fig 4.4.8 CNN-GRU Model training F1 score evolution

```
plt.plot(history['precision_m'])

plt.plot(history['recall_m'])

plt.title('model precision and recall')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['precision','recall'])

plt.show()

plt.close()
```

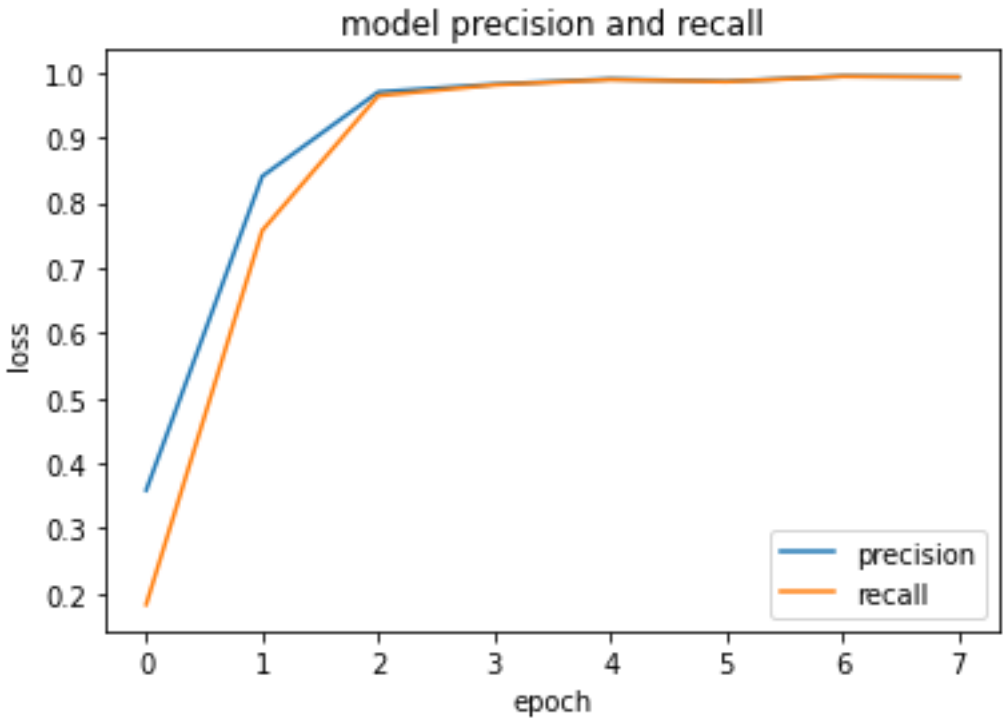


Fig 4.4.9 CNN-GRU Model training precision and recall evolution

4.5 Implementing MLP

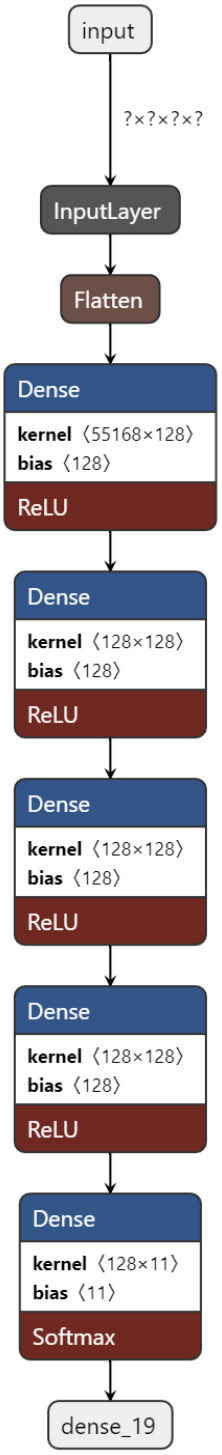


Fig 4.5.1 Architecture of MLP model implementation

For OpenSLR dataset

```
import librosa

import matplotlib.pyplot

import os

import pandas as pd

import numpy as np

import tensorflow as tf

from keras.utils import to_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense,Flatten
```

Defining the generator

```
class AudioDataGenerator(tf.keras.utils.Sequence):

    def __init__(self, csv_file, batch_size=32, num_classes=11,
sample_rate=22050, duration=10.0, shuffle=True, n_mels=128):

        self.csv_file = csv_file

        self.batch_size = batch_size

        self.num_classes = num_classes

        self.sample_rate = sample_rate

        self.duration = duration

        self.shuffle = shuffle

        self.n_mels  = n_mels


        # Read the CSV file

        self.data = pd.read_csv(csv_file)
```

```
        # Get the unique class labels
        self.classes = sorted(self.data['class_label'].unique())

        # Create a dictionary to map class labels to integers
        self.class_to_int = dict(zip(self.classes,
range(len(self.classes))))

        # Shuffle the data if requested
        if self.shuffle:
            self.data =
self.data.sample(frac=1).reset_index(drop=True)

    def __len__(self):
        # Return the number of batches
        return int(np.ceil(len(self.data) / float(self.batch_size)))

    def __getitem__(self, idx):
        # Get the batch of file paths and labels
        batch_data = self.data[idx * self.batch_size:(idx + 1) *
self.batch_size]
        batch_data = batch_data.reset_index()

        # Initialize the arrays for the audio data and labels
        batch_x = np.zeros((len(batch_data), self.n_mels, 431,1))
        batch_y = np.zeros((len(batch_data), self.num_classes))
```

```
        # Load the audio files and their corresponding labels
        for i, row in batch_data.iterrows():
            file_path = row['file_path']
            class_label = row['class_label']

            # Load the audio file
            signal, sr = librosa.load(file_path, sr=self.sample_rate,
mono=True)

            # Pad or truncate the signal to the desired length
            signal = librosa.util.fix_length(signal,
size=self.sample_rate * self.duration)

            # Convert the audio file to spectrogram
            S = librosa.feature.melspectrogram(y=signal, sr=sr)
            S_dB = np.array(librosa.power_to_db(S, ref=np.max))
            S_dB = S_dB.reshape(S_dB.shape[0],S_dB.shape[1],1)

            # Save the audio data and label to the batch arrays
            batch_x[i, :] = S_dB
            batch_y[i, :] =
to_categorical(self.class_to_int[class_label],
num_classes=self.num_classes)

        return batch_x, batch_y
```

```
train_data = pd.read_csv("train_data.csv")

val_data = pd.read_csv("val_data.csv")
```

Instantiating the generator

```
train_generator = AudioDataGenerator('train_data.csv', batch_size=32,
num_classes=11, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Defining the metrics

```
from keras import backend as K


def recall_m(y_true, y_pred):

    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))

    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))

    recall = true_positives / (possible_positives + K.epsilon())

    return recall


def precision_m(y_true, y_pred):

    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))

    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))

    precision = true_positives / (predicted_positives + K.epsilon())

    return precision


def f1_m(y_true, y_pred):
```

```
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Defining the model

```
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(11,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics
=['accuracy',f1_m,precision_m, recall_m])
```

Training the model

```
history = model.fit(train_generator,batch_size=32,epochs=8,verbose=1)
```

Instantiating validation data generator

```
val_generator = AudioDataGenerator('val_data.csv', batch_size=32,
num_classes=11, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Evaluating the model

```
loss, accuracy, f1_score, precision, recall =
model.evaluate(val_generator)
```

Plotting training history

```
import matplotlib.pyplot as plt


plt.plot(history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
plt.close()
```

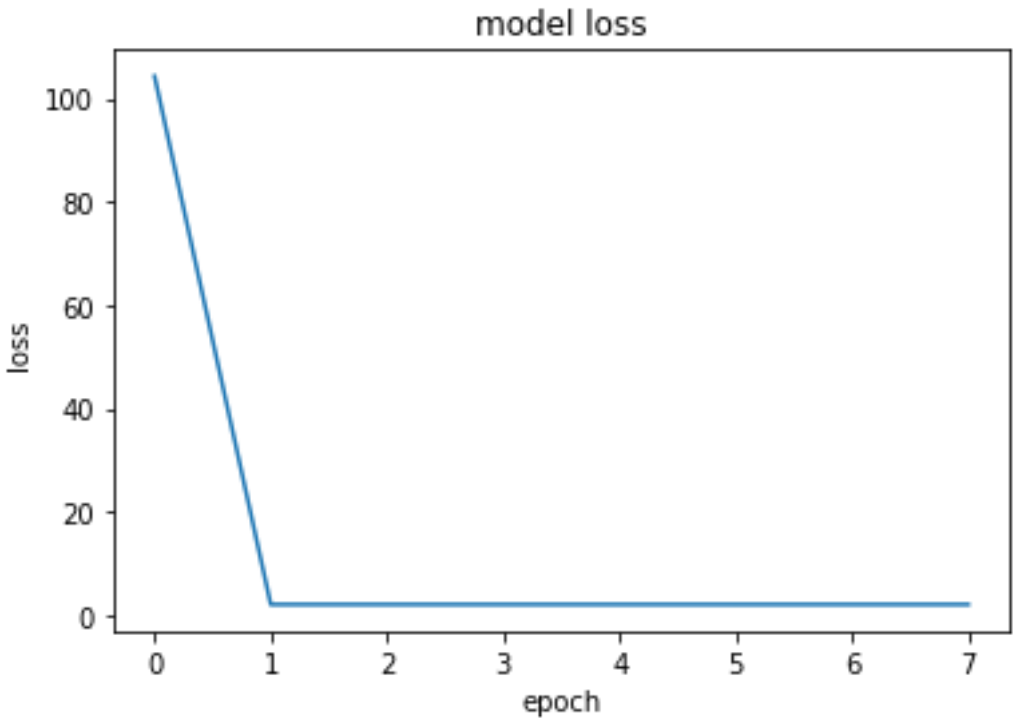


Fig 4.5.2 MLP Model training loss evolution

```
plt.plot(history['accuracy'])
plt.title('model accuracy')
plt.ylabel('loss')
```

```
plt.xlabel('epoch')

plt.show()

plt.close()
```

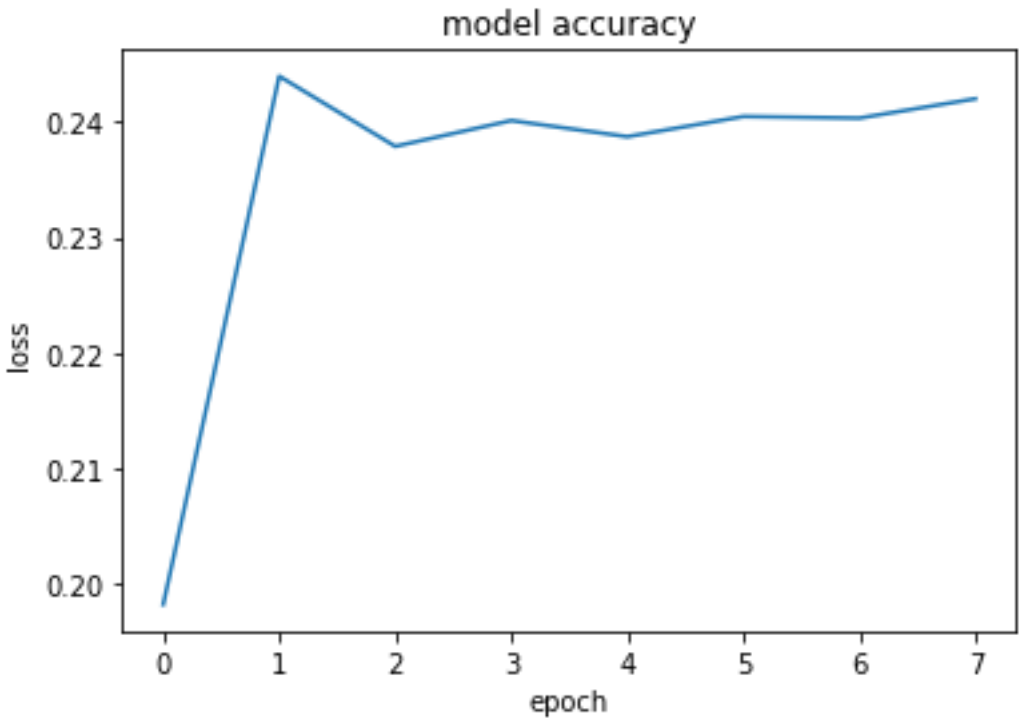


Fig 4.5.3 MLP Model training accuracy evolution

For AccentDB dataset

```
import librosa

import matplotlib.pyplot

import os

import pandas as pd

import numpy as np

import tensorflow as tf

from keras.utils import to_categorical
```

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense,Flatten
```

Defining the generator

```
class AudioDataGenerator(tf.keras.utils.Sequence):

    def __init__(self, csv_file, batch_size=32, num_classes=11,
sample_rate=22050, duration=10.0, shuffle=True, n_mels=128):

        self.csv_file = csv_file

        self.batch_size = batch_size

        self.num_classes = num_classes

        self.sample_rate = sample_rate

        self.duration = duration

        self.shuffle = shuffle

        self.n_mels  = n_mels


        # Read the CSV file

        self.data = pd.read_csv(csv_file)


        # Get the unique class labels

        self.classes = sorted(self.data['class_label'].unique())


        # Create a dictionary to map class labels to integers

        self.class_to_int = dict(zip(self.classes,
range(len(self.classes))))
```

```
        # Shuffle the data if requested
        if self.shuffle:
            self.data =
self.data.sample(frac=1).reset_index(drop=True)

    def __len__(self):
        # Return the number of batches
        return int(np.ceil(len(self.data) / float(self.batch_size)))

    def __getitem__(self, idx):
        # Get the batch of file paths and labels
        batch_data = self.data[idx * self.batch_size:(idx + 1) *
self.batch_size]
        batch_data = batch_data.reset_index()

        # Initialize the arrays for the audio data and labels
        batch_x = np.zeros((len(batch_data), self.n_mels, 431,1))
        batch_y = np.zeros((len(batch_data), self.num_classes))

        # Load the audio files and their corresponding labels
        for i, row in batch_data.iterrows():
            file_path = row['file_path']
            class_label = row['class_label']

            # Load the audio file
```

```
            signal, sr = librosa.load(file_path, sr=self.sample_rate,
mono=True)


            # Pad or truncate the signal to the desired length

            signal = librosa.util.fix_length(signal,
size=self.sample_rate * self.duration)


            # Convert the audio file to spectrogram

            S = librosa.feature.melspectrogram(y=signal, sr=sr)

            S_dB = np.array(librosa.power_to_db(S, ref=np.max))

            S_dB = S_dB.reshape(S_dB.shape[0],S_dB.shape[1],1)


            # Save the audio data and label to the batch arrays

            batch_x[i, :] = S_dB

            batch_y[i, :] =
to_categorical(self.class_to_int[class_label],
num_classes=self.num_classes)


        return batch_x, batch_y


train_data = pd.read_csv("train_data.csv")

val_data = pd.read_csv("val_data.csv")
```

Instantiating the generator

```
train_generator = AudioDataGenerator('train_data.csv', batch_size=32,
num_classes=11, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Defining the metrics

```
from keras import backend as K

def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Defining the model

```
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(128,activation='relu'))
```

```
model.add(Dense(128,activation='relu'))

model.add(Dense(128,activation='relu'))

model.add(Dense(11,activation='softmax'))

model.compile(loss='categorical_crossentropy',optimizer='adam',metrics
=['accuracy',f1_m,precision_m, recall_m])
```

Training the model

```
history = model.fit(train_generator,batch_size=32,epochs=8,verbose=1)
```

Instantiating validation data generator

```
val_generator = AudioDataGenerator('val_data.csv', batch_size=32,
num_classes=11, sample_rate=22050, duration=10, shuffle=True,
n_mels=128)
```

Evaluating the model

```
loss, accuracy, f1_score, precision, recall =
model.evaluate(val_generator)
```

Plotting training history

```
import matplotlib.pyplot as plt


plt.plot(history['loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.show()
```

```
plt.close()
```

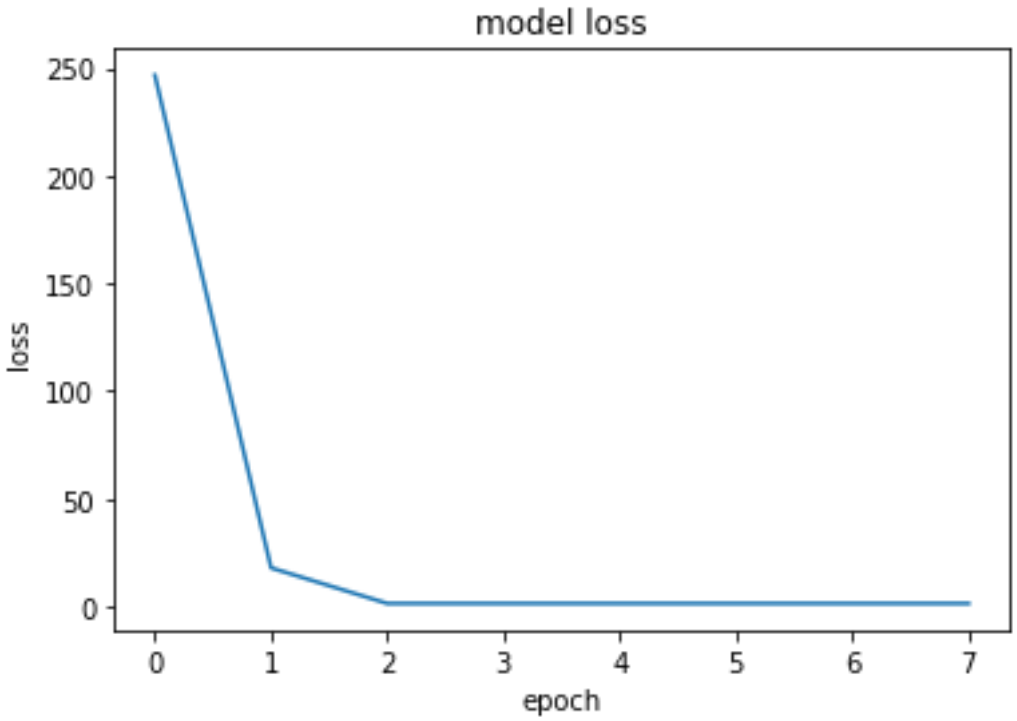


Fig 4.5.4 MLP Model training loss evolution

```
plt.plot(history['accuracy'])
plt.title('model accuracy')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
plt.close()
```

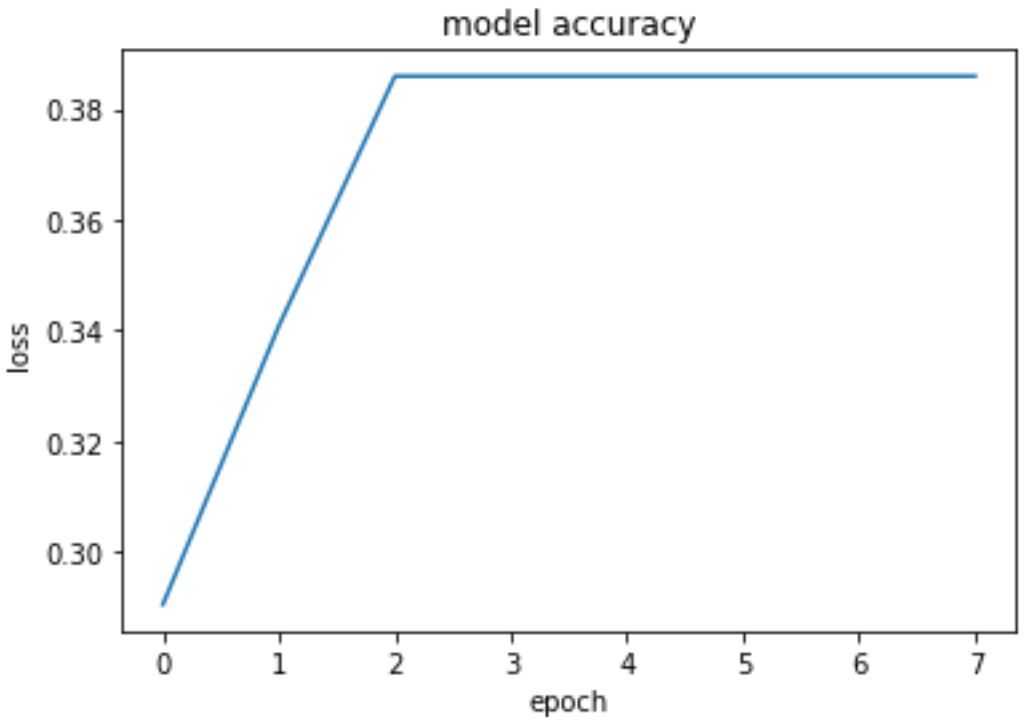


Fig 4.5.5 MLP Model training accuracy evolution

Chapter 5

RESULTS AND ANALYSIS

The classification of regional accents in speech signals has been a challenging task for researchers in the field of speech recognition. In recent years, deep learning techniques have shown great potential in improving the accuracy of accent classification by leveraging the discriminative features of spectrograms. In this study, we evaluate the performance of three deep learning models, namely MLP, CNN, and CNN-LSTM with attention, for classifying regional accents in speech signals using spectrograms.

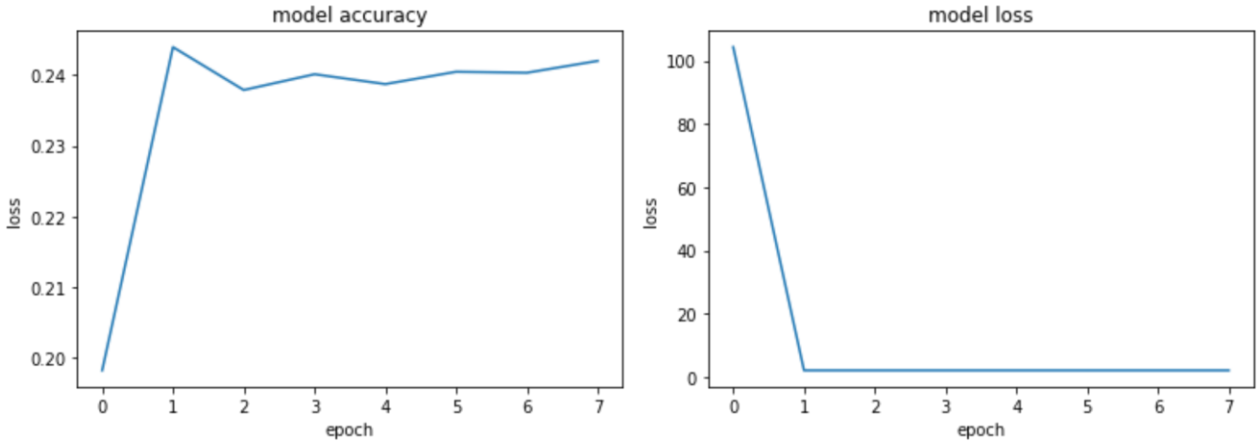
The OpenSLR dataset comprises of speech samples from 6 regional accents split between genders to create a total of 11 different classes, We preprocess the data by converting the speech signals to spectrograms using the Short Time Fourier Transform (STFT). The resulting spectrograms are used as input to our three deep learning models.

The results as described in Table 3 , show that for the OpenSLR dataset, with 11 different classes, the MLP model performed poorly, with an accuracy of only 24.5%. The CNN model achieved a much higher accuracy of 80.1%, the CNN-LSTM with attention model outperformed them with an accuracy of 90.2%, but the best performer was the CNN-GRU with attention model which recorded an accuracy of 94.1%. This not only outperforms the other algorithms that were implemented in his work but also the accuracy recorded in other works [1].

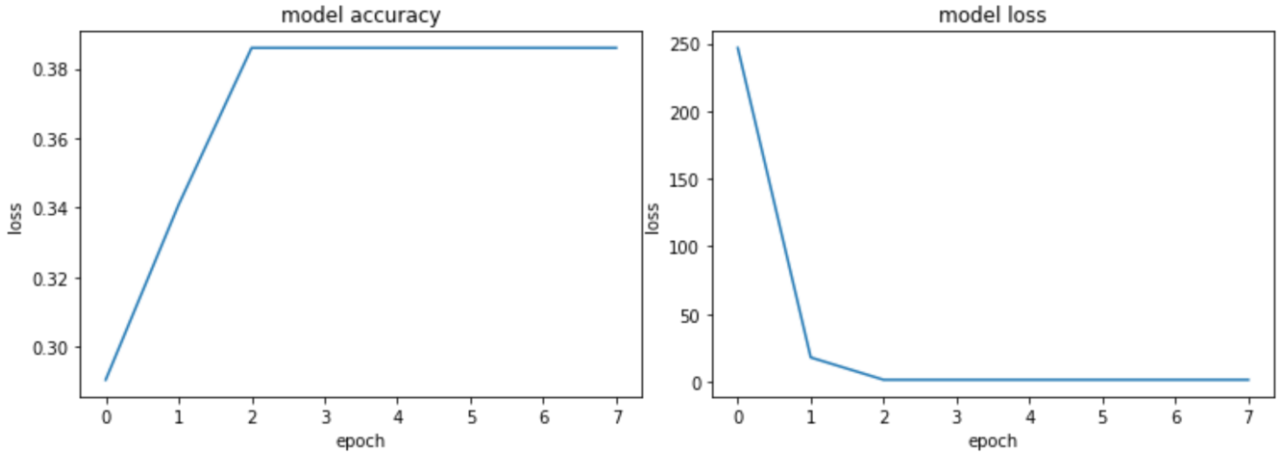
MODEL	*Accuracy*	*F1 Score*	*Precision*	*Recall*
MLP	24.5	0.22	0.21	0.24
CNN	80.1	0.80	0.81	0.79
CNN-LSTM	90.2	0.90	0.91	0.89
CNN-GRU	94.1	94.1	95.1	93.2

Table 3. Results of implemented models for OpenSLR dataset

The poor performance of the MLP model can be attributed to its inability to capture the temporal dependencies in the speech signals. MLP is a feedforward neural network that lacks the ability to handle time-varying inputs. This makes it unsuitable for tasks such as speech recognition, where the temporal evolution of sound signals is crucial for accurate classification.



(a)



(b)

Fig 5.1 Performance graphs of MLP: (a) OpenSLR (b) AccentDB

On the other hand, the CNN and CNN-LSTM with attention models have shown promising results, with the CNN-LSTM with attention model outperforming the CNN model. The CNN component of the hybrid model is responsible for feature extraction from the spectrograms, while the LSTM component captures the temporal dependencies between these features. The attention

mechanism enables the model to weigh the importance of different parts of the spectrogram, allowing the model to focus on the most relevant features.

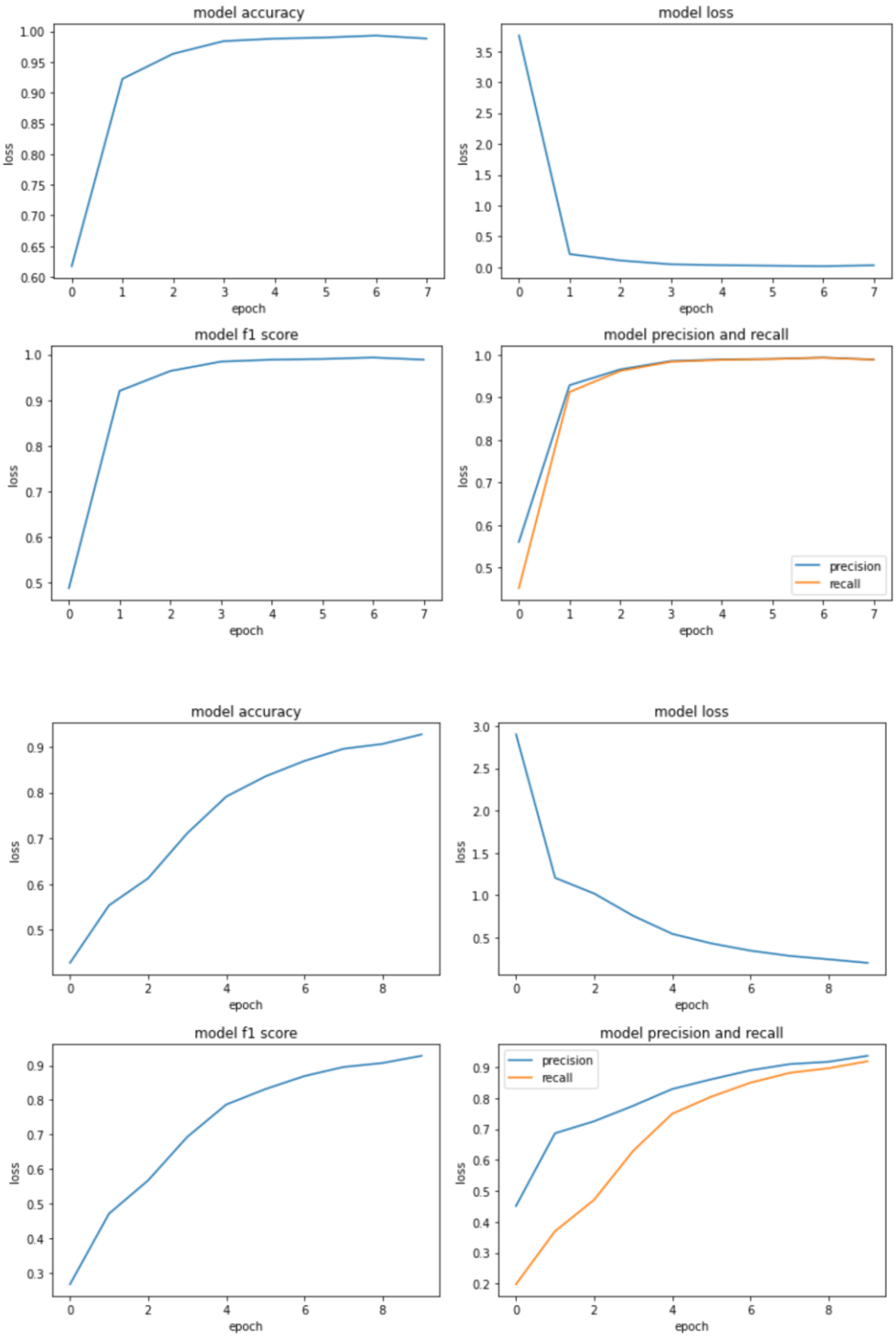
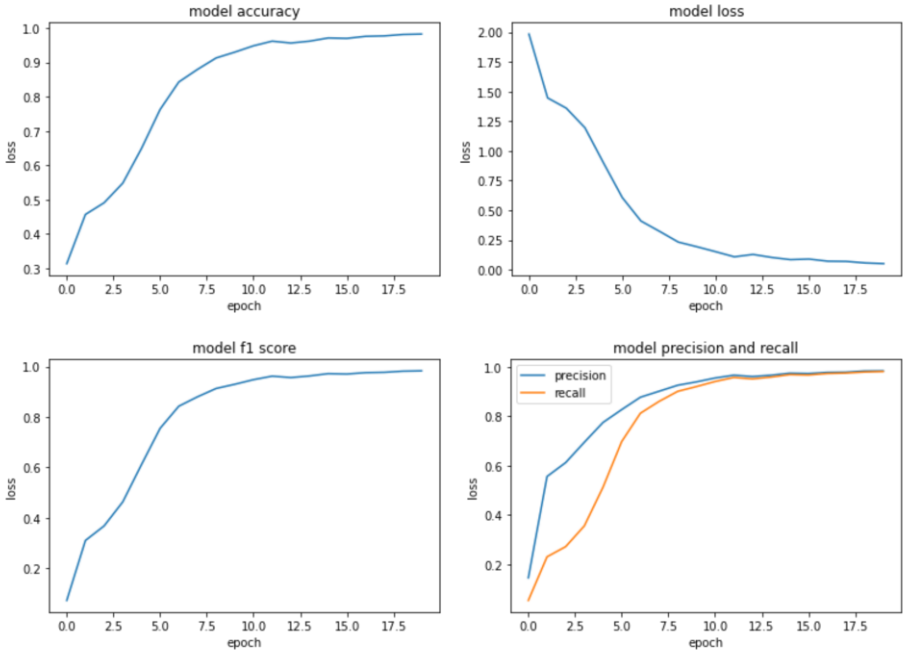
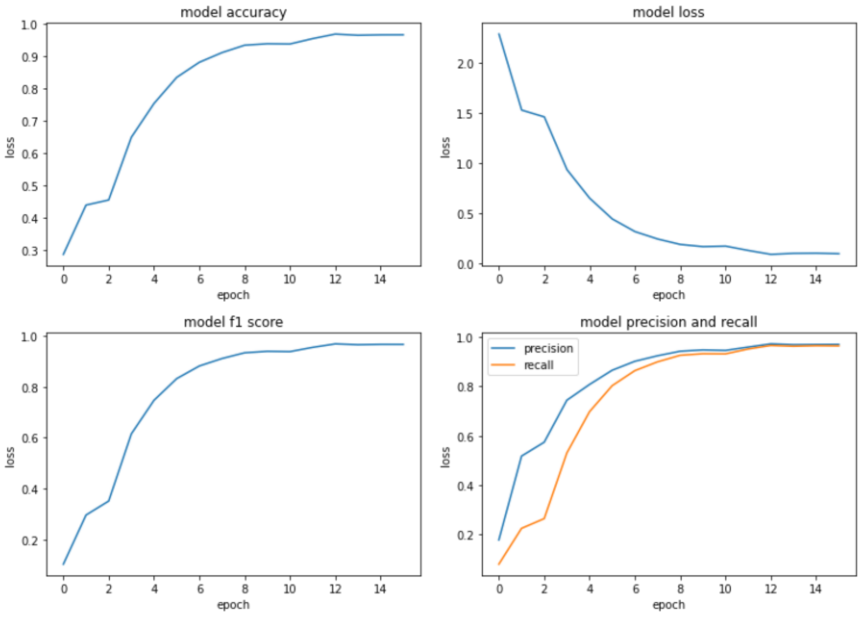


Fig 5.2 Performance graphs for CNN implementation (a) OpenSLR (b) AccentDB

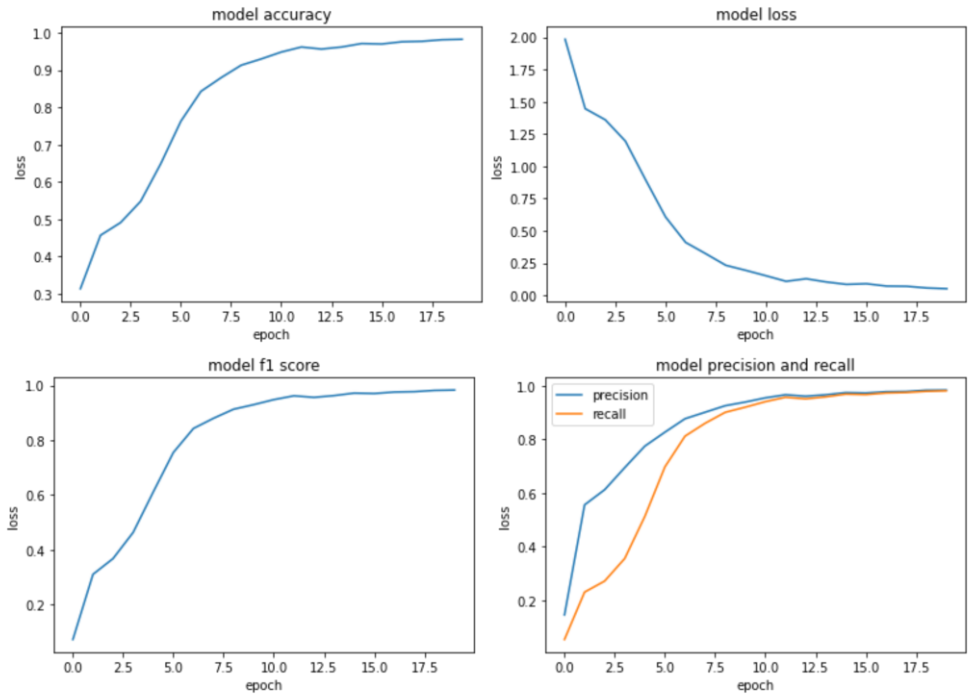


(a)

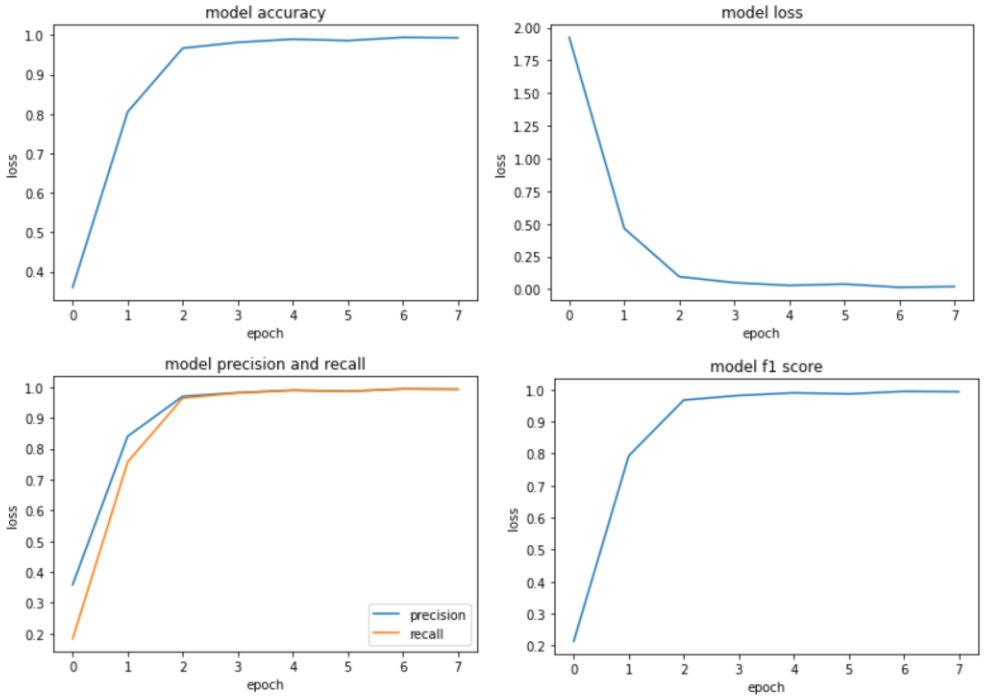


(b)

Fig 5.3 Performance graphs for CNN-LSTM implementation (a) OpenSLR (b) AccentDB



(a)



(b)

Fig 5.4 Performance graphs for CNN-GRU implementation (a) OpenSLR (b) AccentDB

This combination of feature extraction, temporal modeling, and attention mechanism has led to significant improvements in accuracy, making the CNN-LSTM with attention model a powerful tool for regional accent classification.

The performance of two models, a CNN and a CNN-LSTM with attention model, were compared in terms of their ability to accurately classify a dataset. The CNN model, as depicted in Fig 5.2, quickly reached a plateau in terms of accuracy and F1 score, providing a reasonable accuracy metric of 80.1% and an F1 score of 0.801. In contrast, as depicted in Fig 5.3, the CNN-LSTM with attention model achieved a much higher accuracy and F1 score, reaching a plateau at around 10-12 epochs. The model was able to provide a validation accuracy of 90.2% and an F1 score of 0.902. Further more, the CNN-GRU model has proved extremely suitable for the purpose and achieved an accuracy of 94.2%

MODEL	*Accuracy*	*F1 Score*	*Precision*	*Recall*
MLP	39.1	0.39	0.39	0.38
CNN	97.8	0.97	0.97	0.97
CNN-LSTM	98.9	0.98	0.99	0.98
CNN-GRU	99.1	0.98	0.99	0.98

Table 4. Results of implemented models for AccentDB dataset

The second dataset, by AccentDB, which had 4 classes, had models performing significantly better across the board as evident in Table 4. The improved performance may be attributed to the low number of classes

Once again, GRU has outperformed all other models tested by achieving an accuracy of an incredible 99.1% . And closely trailed by LSTM with an accuracy of 98.9%. The CNN model has also performed much better by scoring an accuracy of 97.8% and MLP was last with a very low accuracy of 39.1%

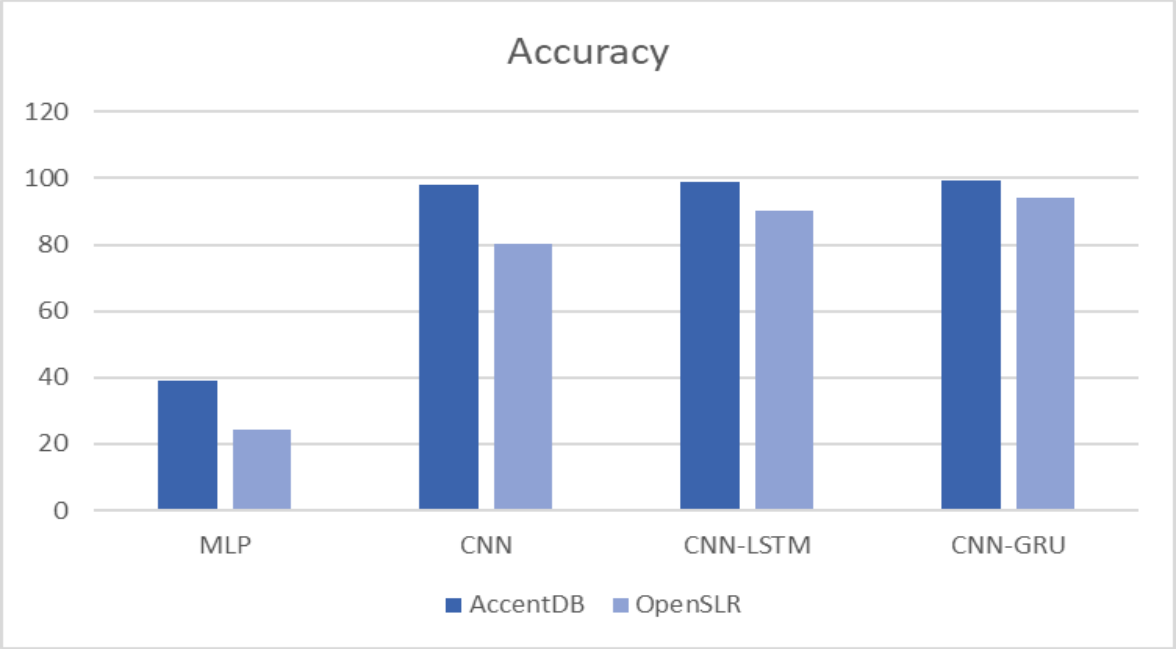


Fig 5.5 Bar graph of Accuracies of all models for both datasets

Overall, as evident int Fig 5.5, the results suggest that the CNN-GRU with attention model is the most effective approach for accurately classifying this dataset, as it was able to achieve a much higher accuracy and F1 score compared to the CNN model and also the CNN-LSTM with attention model. The use of CNNs and GRUs in combination with attention mechanisms may be a useful strategy for other classification tasks that require the identification of both spatial and temporal features.

In conclusion, our results demonstrate that deep learning models, particularly the CNN-GRU with attention model, can be highly effective for regional accent classification using spectrograms. This study highlights the importance of considering the temporal evolution of sound signals and the ability to capture temporal dependencies for accurate classification. Our findings have important implications for the development of speech recognition systems that can handle regional accents, which can have a significant impact on various applications, including language learning, speech therapy, and human-computer interaction.

REFERENCES

1. Cetin, O. Accent Recognition Using a Spectrogram Image Feature-Based Convolutional Neural Network. Arab J Sci Eng (2022). https://doi.org/10.1007/s13369-022-07086-9 I nternational Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075 (Online), Volume-9 Issue-6, April 2020

2. K. Kumpf and R. W. King, "Automatic accent classification of foreign accented Australian English speech," Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP '96, Philadelphia, PA, USA, 1996, pp. 1740-1743 vol.3, doi: 10.1109/ICSLP.1996.607964., 1 (March 2018), 93–120. https://doi.org/10.1007/s10772-018-9491-z

3. G. Choueiter, G. Zweig and P. Nguyen, "An empirical study of automatic accent classification," 2008 IEEE International Conference on Acoustics, Speech and Signal Processing, Las Vegas, NV, USA, 2008, pp. 4265-4268, doi: 10.1109/ICASSP.2008.4518597.

4. R. Upadhyay and S. Lui, "Foreign English Accent Classification Using Deep Belief Networks," 2018 IEEE 12th International Conference on Semantic Computing (ICSC), Laguna Hills, CA, USA, 2018, pp. 290-293, doi: 10.1109/ICSC.2018.00053..

5. A. Purwar, H. Sharma, Y. Sharma, H. Gupta and A. Kaur, "Accent classification using Machine learning and Deep Learning Models," 2022 1st International Conference on Informatics (ICI), Noida, India, 2022, pp. 13-18, doi: 10.1109/ICI53355.2022.9786885.NY, USA, 801–804. https://doi.org/10.1145/2647868.2654984

6. S. Ullah and F. Karray, "Speaker Accent Classification System using Fuzzy Canonical Correlation-Based Gaussian Classifier," 2007 IEEE International Conference on Signal Processing and Communications, Dubai, United Arab Emirates, 2007, pp. 792-795, doi: 10.1109/ICSPC.2007.4728438.

7. P. Nguyen, D. Tran, X. Huang and D. Sharma, "Australian Accent-Based Speaker Classification," 2010 Third International Conference on Knowledge Discovery and Data Mining, Phuket, Thailand, 2010, pp. 416-419, doi: 10.1109/WKDD.2010.80.

8. S. Deshpande, S. Chikkerur and V. Govindaraju, "Accent classification in speech," Fourth IEEE Workshop on Automatic Identification Advanced Technologies (AutoID'05), Buffalo, NY, USA, 2005, pp. 139-143, doi: 10.1109/AUTOID.2005.10

9. Y. Singh, A. Pillay and E. Jembere, "Features of Speech Audio for Accent Recognition," 2020 International Conference on Artificial Intelligence, Big Data, Computing and Data

Communication Systems (icABCD), Durban, South Africa, 2020, pp. 1-6, doi: 10.1109/icABCD49160.2020.9183893.

10. Rongqing Huang and J. H. L. Hansen, "Dialect/accent classification via boosted word modeling," Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005., Philadelphia, PA, USA, 2005, pp. I/585-I/588 Vol. 1, doi: 10.1109/ICASSP.2005.1415181.

11. A. Ahmed, P. Tangri, A. Panda, D. Ramani and S. Karmakar, "VFNet: A Convolutional Architecture for Accent Classification," 2019 IEEE 16th India Council International Conference (INDICON), Rajkot, India, 2019, pp. 1-4, doi: 10.1109/INDICON47234.2019.9030363.

12. A. A. Ayrancı, S. Atay and T. Yıldırım, "Speaker Accent Recognition Using Machine Learning Algorithms," 2020 Innovations in Intelligent Systems and Applications Conference (ASYU), Istanbul, Turkey, 2020, pp. 1-6, doi: 10.1109/ASYU50717.2020.9259902.

13. D. Fohr and I. Illina, "Text-Independent Foreign Accent Classification using Statistical Methods," 2007 IEEE International Conference on Signal Processing and Communications, Dubai, United Arab Emirates, 2007, pp. 812-815, doi: 10.1109/ICSPC.2007.4728443.

14. Z. Ge, "Improved accent classification combining phonetic vowels with acoustic features," 2015 8th International Congress on Image and Signal Processing (CISP), Shenyang, China, 2015, pp. 1204-1209, doi: 10.1109/CISP.2015.7408064.

15. M. F. Hossain, M. M. Hasan, H. Ali, M. R. K. R. Sarker and M. T. Hassan, "A Machine Learning Approach to Recognize Speakers Region of the United Kingdom from Continuous Speech Based on Accent Classification," 2020 11th International Conference on Electrical and Computer Engineering (ICECE), Dhaka, Bangladesh, 2020, pp. 210-213, doi: 10.1109/ICECE51571.2020.9393038.

16. S. Darshana, H. Theivaprakasham, G. Jyothish Lal, B. Premjith, V. Sowmya and K. Soman, "MARS: A Hybrid Deep CNN-based Multi-Accent Recognition System for English Language," 2022 First International Conference on Artificial Intelligence Trends and Pattern Recognition (ICAITPR), Hyderabad, India, 2022, pp. 1-6, doi: 10.1109/ICAITPR51569.2022.9844177.

17. M. Muttaqi, A. Degirmenci and O. Karal, "US Accent Recognition Using Machine Learning Methods," 2022 Innovations in Intelligent Systems and Applications Conference (ASYU), Antalya, Turkey, 2022, pp. 1-6, doi: 10.1109/ASYU56188.2022.9925265.

18. S. Ullah and F. Karray, "An evolutionary approach for accent classification in IVR systems," 2008 IEEE International Conference on Systems, Man and Cybernetics, Singapore, 2008, pp. 418-423, doi: 10.1109/ICSMC.2008.4811311.

19. Y. Ma, M. Paulraj, S. Yaacob, A. Shahriman and S. K. Nataraj, "Speaker accent recognition through statistical descriptors of Mel-bands spectral energy and neural network model," 2012 IEEE Conference on Sustainable Utilization and Development in Engineering and Technology (STUDENT), Kuala Lumpur, Malaysia, 2012, pp. 262-267, doi: 10.1109/STUDENT.2012.6408416.

20. C. Pedersen and J. Diederich, "Accent Classification Using Support Vector Machines," 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007), Melbourne, VIC, Australia, 2007, pp. 444-449, doi: 10.1109/ICIS.2007.47

21. Ozer, I., Ozer, Z., & Findik, O. (2018). Noise robust sound event classification with convolutional neural network. Neurocomputing, 272, 505–512. https://doi.org/10.1016/j.neucom.2017.07.021

22. OZER, I. (2021). Pseudo-colored rate map representation for speech emotion recognition. Biomedical Signal Processing and Control, 66. https://doi.org/10.1016/j.bspc.2021.102502

23. Abdel-Hamid, O., Mohamed, A.-R., Hui Jiang, Li Deng, Penn, G., & Dong Yu. (2014). Convolutional Neural Networks for Speech Recognition. IEEE/ACM Transactions on Audio, Speech, and Language Processing, Audio, Speech, and Language Processing, IEEE/ACM Transactions on, IEEE/ACM Trans. Audio Speech Lang. Process, 22(10), 1533–1545. https://doi.org/10.1109/TASLP.2014.2339736

24. Özseven, T. (2018). Investigation of the effect of spectrogram images and different texture analysis methods on speech emotion recognition. Applied Acoustics, 142, 70–77. https://doi.org/10.1016/j.apacoust.2018.08.003

25. Demirsahin, I., Kjartansson, O., Gutkin, A., & Rivera, C. (2020). Open-source Multi-speaker Corpora of the English Accents in the British Isles. International Conference on Language Resources and Evaluation.

26. G, P. D. (1), & Rao, K. S. (2). (2023). Accent classification from an emotional speech in clean and noisy environments. Multimedia Tools and Applications, 82(3), 3485-3508–3508. https://doi.org/10.1007/s11042-022-13236-w

27. Singh, U., Gupta, A., Bisharad, D., Arif, W., Thampi, S. M., El-Alfy, E.-S. M., & Trajkovic, L. (2020). Foreign accent classification using deep neural nets. Journal of Intelligent & Fuzzy Systems, 38(5), 6347–6352. https://doi.org/10.3233/JIFS-179715

28. Yao, X., Sheng, Z., Gu, M., Wang, H., Xu, N., & Liu, X. (2021). Attention mechanism based LSTM in classification of stressed speech under workload. Intelligent Data Analysis, 25(6), 1603–1627. https://doi.org/10.3233/IDA-205429

29. Mikhailava, V., Lesnichaia, M., Bogach, N., Lezhenin, I., Blake, J., & Pyshkin, E. (2022). Language Accent Detection with CNN Using Sparse Data from a Crowd-Sourced Speech Archive. Mathematics (2227-7390), 10(16), 2913. https://doi.org/10.3390/math10162913

30. Guntur, R. K., Ramakrishnan, K., & Vinay Kumar, M. (2022). An Automated Classification System Based on Regional Accent. Circuits, Systems & Signal Processing, 41(6), 3487–3507. https://doi.org/10.1007/s00034-021-01948-7

31. Ahamad, A., Anand, A., and Bhargava, P. 2020. AccentDB: A Database of Non-Native English Accents to Assist Neural Speech Recognition. In Proceedings of the Twelfth Language Resources and Evaluation Conference (pp. 5351–5358). European Language Resources Association.