# MEMORY-AWARE AND INTERPRETABLE LUNG SOUND CLASSIFICATION BASED ON FEATURE CONCENTRATION LEARNING

## ABSTRACT

Respiratory diseases are a major global cause of death, underscoring the importance of early detection and treatment. Deep learning and machine learning have shown promise in identifying respiratory issues without direct physician involvement, highlighting the potential of automated detection methods. Wheezing, a crucial indicator of airway blockage severity, is typically diagnosed through auscultation, but this method heavily relies on the physician's experience. While previous research has aimed to extract wheezing characteristics from breathing sounds, practical real-time monitoring tools are lacking.

In the proposed way, a digital method utilised Mel-frequency cepstral coefficients (MFCCs) to extract lung sound properties. The K-means algorithm was used to cluster these features, reducing computational complexity, and the K-nearest neighbour technique was employed for lung sound classification. By overlaying the frequency spectrum over respiratory phases, the method facilitated recording navigation and provided insights into the inter-individual ratios of inhalation and exhalation phases. This approach enables data comparison between patients, aids in lung sound analysis, and assists in identifying various respiratory dysfunctions.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| COPD | Chronic Obstructive Pulmonary Disease |
| DCT | Discrete Cosine Transform |
| ICB HI | International Conference on Biomedical and Health Informatics |
| LAN | Local Area Network |
| MFCC | Mel Frequency Cepstral Coefficient |
| STFT | Short-time Fourier Transform |

# CHAPTER 1
# INTRODUCTION

## 1. Introduction

Lung auscultation, performed using stethoscopes, is a standard method to detect lung abnormalities. Despite medical advancements, the traditional stethoscope remains the primary tool. Respiratory diseases, exacerbated by factors like air pollution and lifestyle changes, rank high among leading causes of death globally. These diseases have gained more attention due to the COVID-19 pandemic, where identifying respiratory symptoms accurately becomes crucial. Lung sounds categorize into normal (bronchial, vesicular, tracheal) and abnormal (crackles, rhonchi, wheezes) sounds. Abnormal sounds are vital indicators for diagnosing lung diseases. Adventitious sounds, specifically wheezes and crackles, are significant markers for conditions like asthma and COPD. Traditional lung auscultation heavily relies on physician expertise, making accurate diagnosis challenging. With advancements in computer-based respiratory sound recognition, machine learning-based automatic lung sound analysis has gained clinical significance. Respiratory diseases were previously overshadowed by other health issues but have now become a major global concern. Computer-assisted pulmonary auscultation has grown in importance, especially with the need for remote monitoring during events like the COVID-19 pandemic.

Machine learning and deep learning techniques offer promising avenues for automated lung sound analysis. However, existing studies often face limitations due to small dataset sizes or proprietary datasets, hindering model comparisons. Open-access datasets are essential for advancing these methods. Traditional sound analysis methods use time-frequency characterizations like Fourier transforms. Recent approaches, including cepstral features and multi-time-scale methods, have shown promise but can lack statistical robustness or clinical interpretability. In this project, we propose a multi-time-scale feature set designed to capture crackles and wheezes' characteristics. These features will feed a supervised learning framework for accurate anomaly detection, providing valuable insights into healthy and pathological conditions. This system holds promise for telemedicine applications, enabling real-time remote diagnosis of respiratory conditions.

## 2. Problem Statement

Traditional clinical diagnosis of respiratory diseases relies on chest auscultation, but objective analysis and automatic interpretation of lung sounds are increasingly needed to aid clinical practice. This project introduces a method to differentiate between normal and abnormal lung sounds based on their morphological complexities, including texture information (lacunarity), irregularity index (sample entropy), skewness (third order moment), and kurtosis (fourth order moment).

Various factors can complicate diagnosis, emphasizing the need for accurate and early detection of respiratory diseases. Imbalanced class distributions in sound datasets present another challenge, where some sound classes are overrepresented while others are underrepresented. This imbalance can hinder model performance, especially for minority classes, leading to potential biases and reduced classification accuracy. Addressing this data imbalance and ensuring a balanced dataset is essential for improving the model's robustness and accuracy.

## 3. Aim of the Project

The project aims to develop an efficient predictive model for respiratory data analysis, focusing on segmenting normal and abnormal (wheeze) sounds. This involves handling increasing data volumes, reducing training time, mitigating overfitting, optimizing data quality, and ensuring efficient model training.

## 4. Project Domain

The domain of the project is Machine Learning. Advancements in machine learning techniques have posed challenges in the fields of Medical Signal Processing and memory-aware and interpretable algorithms. Machine learning uses various algorithms based on the requirement of the project.

## 5. Scope of the Project

This type of Interpretable Lung Sound Classification can be implemented in real-time applications such as hospitals, clinics, telemedicine platforms, and medical research institutions to assist healthcare professionals in accurate lung disease diagnosis and monitoring

## 6. Motivation

The methodology comprises four modules. The first module focuses on Data Preprocessing, transforming sound signals into frames of 20~40 ms, enhancing high frequencies, and applying signal transformation to Mel-frequency cepstral coefficients (MFCCs) followed by Discrete Cosine Transform (DCT).

The second module, Data Augmentation, uses Mel filters and logarithmic transformation to enhance the training data, employing triangular bandpass filters to create diverse spectrograms.

The third module introduces the Feature Concentration Learning Network, a CNN architecture with specific layers and regularization techniques like ReLU activations and Dropout layers.

The final module, Experimental Results Evaluation, assesses the model's performance using a confusion matrix, calculating accuracy, precision, recall, and F1 score to evaluate the lung sound classification model comprehensively

# CHAPTER 2

# LITERATURE SURVEY

Tong et al., [1] focuses on the physical modeling of lung bronchi to replicate the sound pressure curves of wheezes and regular breath sounds. The study found that the respiratory sound signals of bronchi are primarily spread below 2,000 Hz, with the peaks of the spectra varying based on the type of respiratory sound. To adaptively weight the spectrum characteristics of respiratory sound signals, the authors created a feature-band attention module. This module was then used as input for an end-to-end respiratory sound classification system. Experimental findings on public databases showed promising results for the suggested system with the feature-band attention module. The authors plan to further investigate the effects of real physiological circumstances, such as elevated mucus output and mucosal edema, in the future. They also intend to optimize the sensitivity value using data augmentation and transfer learning techniques.

Jianqiang Li et al., [2] presents a novel CNN architecture for analyzing respiratory sounds. The architecture incorporates fuzzy decision tree regularization to improve interpretability and address uncertainty in decision-making. The authors demonstrate the effectiveness of their approach using the Respiratory Sound database and show that it can learn from scattered data and reduce model parameters. The ultimate goal is to apply this approach to auscultation systems in hospitals in future research.

Nguyen et al.[3] discusses their techniques for respiratory disease recognition and adventitious lung sound classification using multi-channel lung sound datasets and the ICBHI dataset. They employ various techniques such as vanilla nested tuning, co-tuning, stochastic normalization, and the combination of co-tuning and stochastic normalization to leverage pre-trained models from different ResNet topologies. They also incorporate spectrum correction and mapping data augmentation to improve the system's resilience. The suggested techniques outperform other state-of-the-art systems in classifying respiratory diseases and adventitious lung sounds, achieving average scores of 64.74% and 58.29% for the 2-class adventitious lung sound task and 4-class task, respectively. Similar results are obtained for the 3- and 2-class respiratory disease classification tasks, with average scores of 92.72% and 93.77%, respectively. The authors also evaluate their adventitious lung sound classification method with co-tuning on their multi-channel lung sound dataset, achieving a superior F-score compared to their earlier work.

They also examine classification algorithms for respiratory illnesses and adventitious lung sounds using their datasets.

According to research conducted by Choi et al. [4], focuses on the classification of lung ailments using a lightweight-based model that utilizes depth wise separable convolution. The authors highlight the challenges faced in obtaining a wide variety of respiratory sounds due to the COVID-19 pandemic. They propose the use of lightweight skip connections with effective feature stacking for early disease classification and respiratory sound classification. The authors also compare their co-tuning system for 2-class lung sound classification to their earlier work, showing a superior F-score. They further examine state-of-the-art classification algorithms for respiratory illnesses and adventitious lung sounds using their multi-channel lung sound dataset and the ICBHI dataset. The authors express gratitude to Alexander Fuchs and their colleague for their insightful comments and conversations.

McFarlane et al. [5] focuses on the classification of respite sounds using pre-existing deep neural network-based image classifiers. The researchers assessed three tasks related to classification and used different visualisation techniques. The results showed that the Mel-spectrogram and aggregate image approaches performed best in separating wet cough from dry cough, with testing accuracies of 0.94 and 0.88, respectively. The aggregate image-based technique also performed well in categorising across four classes and differentiating between cough and respiratory distress, with testing accuracies of 0.88 and 0.91, respectively. The study also included additional examples from open sources to evaluate the final aggregate image classifier, and all classifiers struggled with differentiating between wet and dry cough sounds in the new data.

Fattahi et al. [6] discusses the challenges faced in auscultation of chest recordings, particularly in neonates. The recordings often contain a combination of heart and lung sounds, as well as other noises, which can reduce the accuracy of medical diagnosis. The authors emphasize the importance of filtering the unprocessed signals before clinical auscultation. They also mention that the characteristics of an ideal frequency filter can vary among different cases, despite the common use of frequency filtering.

Tsai et al. [7] In a single-channel recording scenario, the suggested PC-DAE is generated based on the periodicity features of the signal to accomplish blind source separation. PC-DAE

does not require supervised training data, in contrast to the traditional supervised source separation approach.

To the best of our knowledge, the suggested PCDAE is the first study to perform heart-lung sound separations by combining the benefits of deep learning-based feature representations with the periodicity property.

The authors of the paper, Zheng et al.[8] assembled and labeled the largest data collection of gastric sounds called the GI Sound collection. They developed a CRNN system based on this dataset that can recognize sound events at the segment and recording levels using weak labeling. The system achieved 81.06% F1 score on Task A, 63.40% F1 score on Task B, and 70.13% error rate. The authors suggest that their experimental techniques can be applied to larger datasets with a greater variety of sound occurrences and can provide frame-level results that can help determine a patient's prognosis and treatment.

The literature review discusses a research study conducted by Meng et al.[9] The study focuses on the classification of lung sounds into crackles, rhonchi, and healthy individuals using a classification model. The sample included 705 signals from 130 patients at a hospital in China. The feature vector used in the model included wavelet entropy, relative wavelet energy in seven wavelet layers, and Gaussian kernel functions to calculate wavelet similarity. The artificial neural network technique achieved the highest classification accuracy of 85.43%. The study found that the time-frequency properties of the signals are reflected in the similarity between wavelet sub-signals and the original signal. However, there are some limitations in the techniques used that need refinement, particularly in the normalization step and the measurement of signal similarity.

The study by Chen et al.[10] proposes using the ResNet deep learning technique with OST to identify wheezes, crackles, and other common respiratory noises. The researchers use various rescaled spectrum maps, including STFT, ST, and OST, to construct the suggested ResNet-based technique. The experimental results show that the proposed method outperforms the hard dataset of loud respiratory sounds, achieving a classification accuracy of 98.79%. The method is also reliable for differentiating respiratory sounds, with a sensitivity of 96.27.

# CHAPTER 3

# PROJECT DESCRIPTION

## 1. Existing System

The existing system employs empirical mode decomposition and harmonic-percussive source separation to analyze respiratory sounds, achieving an F1 score of 41.9% on the ICBHI 2017 Respiratory Sound Database. Despite this success, challenges arise from environmental noise and dataset limitations, underscoring the need for robust benchmarks. Adapting the system to patient demographics improves its clinical viability, but efficiency and interpretability require further attention. High model complexity demands significant computational power, with risks of overfitting and sensitivity to outliers. There's a risk of overfitting if the training set doesn't capture the domain's underlying information, necessitating retraining for new input types. Moreover, the system is sensitive to outliers and is limited to numerical values.

## 2. Proposed System

The proposed system standardizes sound characteristics like sampling frequency and channel number to ensure uniform processing. Sounds are converted to monophonic and split into train (60%), validation (20%), and test sets (20%). Mel spectrograms are computed using STFT and a Mel filter bank to mimic human ear perception. Our CNN architecture comprises convolutional, activation, pooling, and fully-connected layers. These layers extract features, enhance learning, and reduce spatial size, aiming to prevent overfitting and computational complexity. SoftMax activation is used for the final classification. Our contributions involve employing transfer learning with CNNs, adapting image CNNs for sound classification, and evaluating classification results and method fusion for improved consistency.

## 3. Advantages

- Easier to use with less need for tweaking parameters.
- Combines several statistical approaches to improve the final predictive performance.
- Performs better and achieves good outcome in terms of accuracy and F-measure object detection model.
- High accuracy and Modeling complex relationships.
- Easier to use with less need for tweaking parameters

## 3. Feasibility Study

A Feasibility study is carried out to check the viability of the project and to analyze the strengths and weaknesses of the proposed system. The application usage of the project must be evaluated. The feasibility study is carried out in three forms.

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

### 3.1. Economic Feasibility

- **Computational Resources**: Will require the adequate and qualitative number of libraries for implementation.
- **Data Acquisition**: Training the model likely requires a large dataset of medical sounds with accurate lung sounds. Acquiring this data may involve licensing costs or collaborations with medical institutions.
- **Reduced Treatment Costs**: Prediction of respiratory diseases requires accurate estimation of wheeze patterns. Early diagnosis may prevent the severity of the disease and potentially reducing the cost of the treatment.

### 3.2. Technical Feasibility

- **Technical Expertise**: Requires expertise in machine learning, Jupyter Notebook, Sklearn and Librosa.
- **Data Availability**: The project's success hinges on the availability of a high-quality lung sounds and noise free breathing patterns.
- **Computational Resources**: Training the model necessitates access to computational resources with high processing power and sufficient memory, potentially including GPUs.

### 3.3. Social/Organizational Feasibility

- **Ethical Considerations:** The project must adhere to strict data privacy regulations concerning medical sounds.
- **Acceptance by Medical Community:** For real-world application, the medical community needs to be convinced of the model's accuracy, reliability, and generalizability

across different patient populations.

- **Social Impact:** A successful project has the potential to significantly improve patient outcomes for respiratory diseases such as COPD, wheezing, etc. diagnosis and treatment leading to better quality of life for patients.

# 4. System Specification

## 4.1. Hardware Specification

- Processor: Minimum i5 Dual Core
- Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)
- Hard Drive: Minimum 200 GB; Recommended 200 GB or more
- Memory (RAM): Minimum 16 GB; Recommended 32 GB or above

## 4.2. Software Specification

- Python For AI/ML/DL Programming
- Jupyter Notebook IDE (Integrated Development Environment) for Development.
- PyTorch or TensorFlow for Deep Learning Coding.
- Sklearn for Machine Learning/Feature Extraction/Evaluation Metrics Coding.
- Numpy for implementing Linear Algebra.
- Plotly for Data Visualization (For Graphs).
- Matplotlib for Data Visualization (For Graphs).
- Seaborn for Data Visualization (For Graphs).
- Librosa for Audio Analysis.

# CHAPTER 4 PROPOSED WORK

The system standardizes sound characteristics like sampling frequency and channel numbers for uniform processing. It divides the dataset into training, validation, and test sets, comprising 60%, 20%, and 20% of the files, respectively. Mel spectrograms are generated using Short-Time Fourier Transform (STFT) and a Mel filter bank to mimic human perception of sound. The system employs Convolutional Neural Networks (CNNs) and uses Feature Concentration Learning Model

## 1. General Architecture

Figure 4.1 represents the architecture diagram of the project. Firstly, the input is taken as sound and algorithm is trained using data sets, which is used as training data.

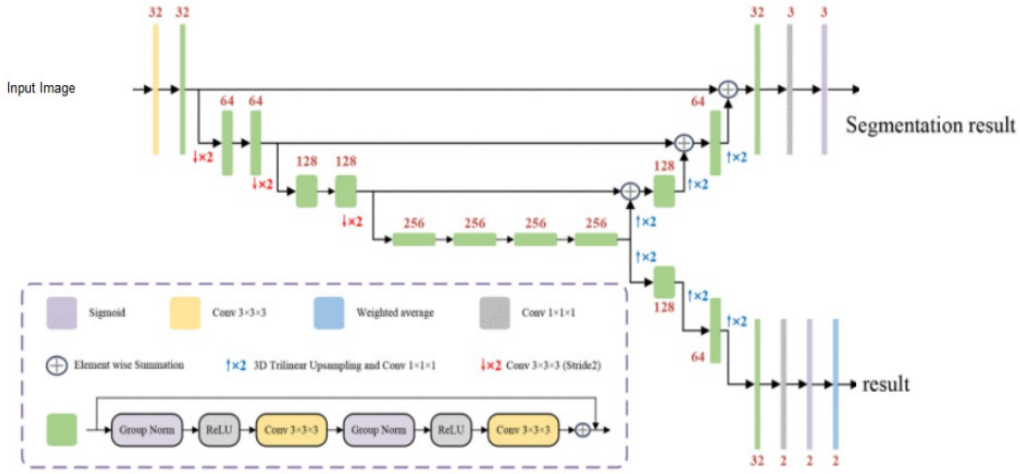


**Fig 4.1 General Architecture Diagram**

Then the images are forwarded to Feature Concentration model and detection model. Then confusion matrix is created which delivers performance and used to evaluate the classification model.

## 2. Design Phase

### 2.1. Flow Diagram

The Figure 4.2 represents the flow diagram of our project. The Dataset is collected and the preprocessed. Then the Data is split and sent to learning model

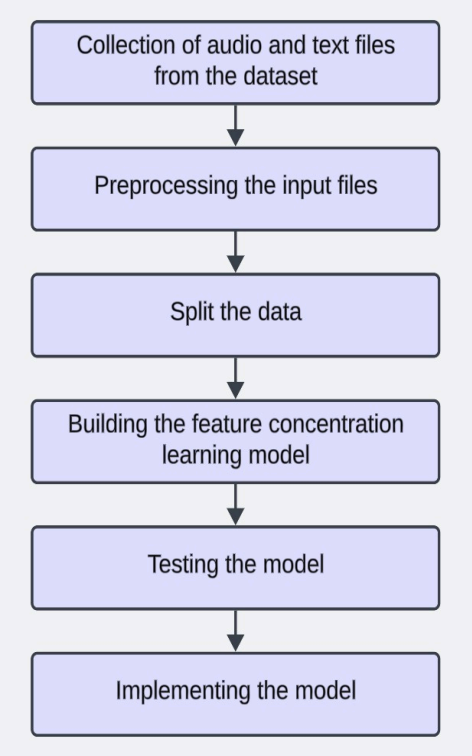


**Fig 4.2 Flow Diagram**

Then the model is tested and implemented. The results of the patient's Wheeze classification is showed. The flow diagram is a visualization of a sequence of actions, movements within a system and/or decision points. They're a detailed explanation of each step in a process, no matter the level of complexity of that process

## 3. UML Diagram

### 3.1. Activity Diagram

Figure 4.3 represent the UML diagram of our model. There are two phases for our project. In Phase1 the model is trained with the augmented data using Feature concentration.

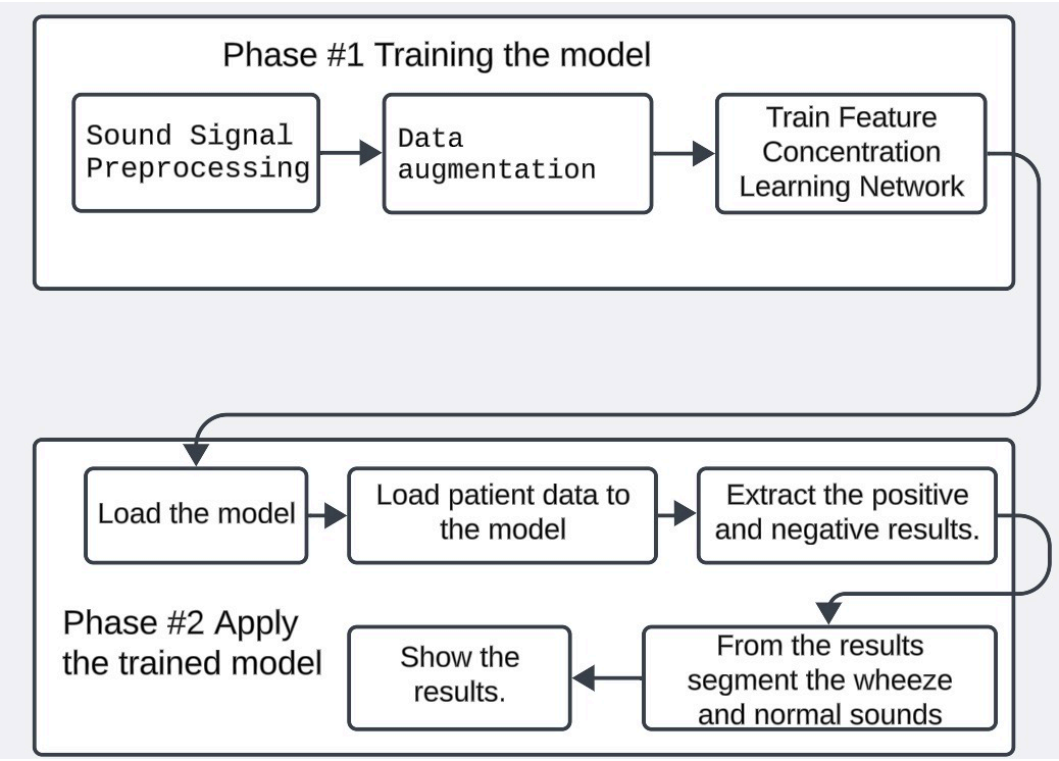


**Fig 4.3 Activity Diagram**

Activity diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity. This fig show the activity diagram for the project phases and continuum.

## 3.2. Use Case Diagram

The Figure 4.4 represents the Use Case diagram of our model. In the interface the patient uploads the lung sound. Then the model is implemented.

The uploaded sound is captured and sound processing is carried out by the model. The model then classifies the sound into Normal and Wheeze sound patterns. The results are shown to the physician and the necessary actions are taken.

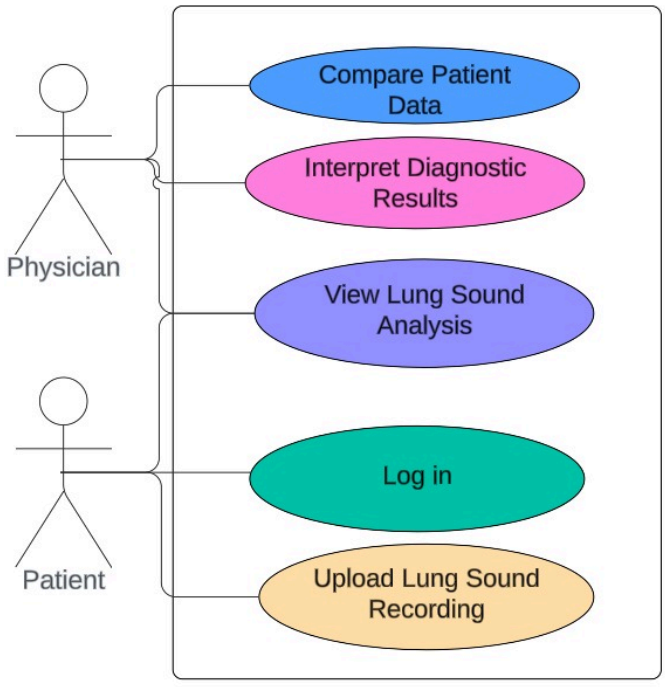


**Fig 4.4 Use Case Diagram**

## 3.3. Sequence Diagram

The sequence diagram illustrates a brain tumour segmentation workflow. The user first inputs an image, which is then pre-processed. Following segmentation of the image into different regions, UNet neural network analyses the image. The system then selects informative features and uses them to classify whether a tumour is present in the image. Finally, the classification result is displayed to the user as shown in Fig 4.5.

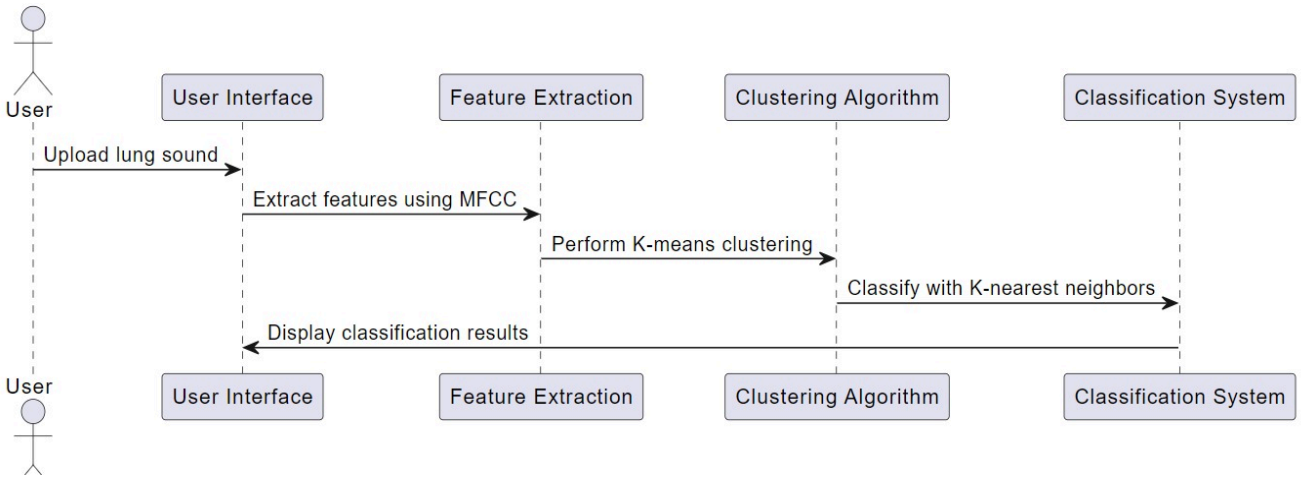


**Fig 4.5 Sequence Diagram**

## 4. Module Description

This project is divided into four modules, Data Preprocessing, Data Augmentation, Feature Concentration Learning Network and Training and Validation.

### 4.1. Data Preprocessing

- In order to enable the sound signal to become the input of the CNN classification model after being properly processed, a preprocessing procedure was performed.
- The preprocessing procedures include frame division, pre-emphasis, Hamming window, signal transformation, Mel filter, and discrete cosine transform (DCT), and each procedure is described as follows:
- Frame blocking: The sound signal is continuously changing. To simplify the continuously changing signal, it is assumed that the sound signal does not change in a short time scale. Therefore, the sound signal is aggregated into a unit with multiple sampling points (N), which is called an audio frame.
- An audio frame is 20~40 ms. If the length of the audio frame is shorter, there will not be enough sampling points in each audio frame to perform reliable spectrum calculation, but

if the length is too long, the signal of each audio frame will change too much.

- In addition, to avoid excessive changes between two adjacent audio frames, we will allow an overlapping area between two adjacent audio frames, and this overlapping area contains about half or one-third of the sampling points (M) in the audio frame.

## 4.2. Data Augmentation

- The classification efficiency is limited by the amount of training data required for the solid classification model. Additionally, the collection and processing of training data are not very easy.
- Therefore, the use of data augmentation methods will help to improve the performance of sound classification. Consider the Mel filter, which is designed based on the characteristics of the frequency response of the human auditory system to sound.
- Since the perception of the human ear is on a logarithmic scale, the logarithmic transformation can better simulate the human ears perception of sound. Using a set of Mel filters, the speech signal can be divided into several different frequency bands, and the strength of each frequency band can be represented by a logarithmic value.
- These logarithmic values are often used as acoustic features for classification and modeling in tasks such as speech recognition. Using different numbers of triangular bandpass filters for the same sound signal will produce a similar but not identical logarithmic energy.
- Therefore, if K sets of triangular bandpass filters with different numbers are set, the same sound can produce K times similar sound characteristics, in order to achieve the purpose of data augmentation.

## 4.3.Feature Concentration Learning Network

- The input layer uses an image 40 173 pixels in size as the input layer, and a total of 3 hidden layers (including convolutional layer and the pooling layer) are used, and the number of convolution kernels for each convolutional layer is 64, 128, and 256.
- The pooling layer uses the maximum pooling operation, and the activation function used is the Rectified Linear Unit (ReLU), in order to prevent overfitting problems during model training convergence. We added Dropout to each hidden layer to reduce overfitting, and finally classified the data through the output layer.
- The operation steps of the proposed sound classification model are as follows:

  Step (1): The image after the preprocessing of the data is set to a 40173-pixel image as the input data of the input layer.

  Step (2): This is the first convolution layer. The convolution kernel of this layer is set

to 2 * 2, the feature map is set to 64, and the activation function is ReLU. The convolutional layer-processing method is similar to the image-processing method. Using the sliding-window calculation, by giving different weight combinations to the convolution kernel, it is possible to detect the edges and corners of the shape, and it also has the effect of removing noise and sharpening, as well as extracting these features as the basis for identification.

Step (3): This step is a pooling layer using maximum pooling. The size of the pooling layer is set to 2 * 2. The pooling layer is a method of compressing images and retaining important information. The sampling method is the same as for sliding windows, but maximum pooling is generally used. If the sliding-window size is set to 2 and the stride is also set to 2, the amount of data will be reduced to a quarter of the original, but because the maximum value is taken, it still retains the greatest possibility of local range comparison. That is, the pooled information is more focused on whether there are matching features in the picture, rather than where these features exist in the picture. Therefore, if the image is shifted, it can still be recognized.

Step (4): This is the second convolutional layer, which performs convolution operations on the data again to find various features in the image that are more detailed. The convolution kernel of this layer is set to 2 * 2, the feature map is set to 128, and the activation function is ReLU.

Step (5): This step is a pooling layer using maximum pooling. The size of the pooling layer is set to 2 * 2.

Step (6): This is the third convolutional layer, which performs convolution operations on the data again to find more detailed features in the image. The convolution kernel of this layer is set to 2 * 2, the feature map is set to 256, and the activation function is ReLU.

Step (7): This step is a pooling layer using maximum pooling. The size of the pooling layer is set to 2 * 2.

Step (8): This step consists of two flattening layers, and each node is formed into a fully connected layer to form a classifier. This means that the feature data obtained through the convolution operation will be converted into the corresponding output classification results.

Step (9): The sound classification category is output.

## 4.4. Experimental Results

- The confusion matrix is a method used to verify the classification effect , which is used to evaluate the performance of the classification model.
- True positive (TP) indicates that the result of the forecast data is the same as the actual data, and true negative (TN) indicates that the result of the forecast data is not the same as the actual data.
- False positive (FP) indicates that while the forecast data result is the same as the actual data, the true result is not the same as the actual data, and false negative (FN) means that while the result of forecast data is not the same as the actual data, the true result is the same as the actual data.
- Based on the results of the confusion matrix, the accuracy, precision, recall, and F1 score will be discussed separately to evaluate the proposed models classification performance.

1. Accuracy: The accuracy of sound classification is defined as shown in Equation, representing the ratio of correct classification cases to all classification cases in the classification model:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad 4.1$$

2. Precision: The precision of the sound classification is defined as Equation; among all the predicted results, the proportion of correct results is predicted:

$$\text{PRECISION} = \frac{TP}{TP + FP} \quad 4.2$$

3. Recall: The recall rate for sound classification is defined as Equation, representing the proportion of predictions among all actual results:

$$\text{RECALL} = \frac{TP}{TP + FN} \quad 4.3$$

**4.** f1-score: The F1 score of sound classification is defined as Equation, where P stands for precision and R stands for recall, which is a comprehensive index of the two evaluation methods, where the value range is between 0 and 1, and the closer to 1 the value is, the better the classification result is:

$$\text{F1} - \text{SCORE} = \frac{2*\text{PRECISION}*\text{RECALL}}{\text{PRECISION} + \text{RECALL}} \quad 4.4$$

## 4.5. Training & Validation

- Train the model using the training dataset.
- Set training parameters such as the number of iterations, batch size, optimizer and initial learning rate.
- Set the model training count to 200 and the batch size to 6. The network optimizer to Adams algorithm and the initial learning rate to 0.0001.
- The training set comprises of 80% of the total dataset and 20% for testing. 20% of the 80% of training dataset is used for validation.
- Implement a dropout strategy with a ratio of 0.5 to prevent overfitting during training.

## 4.6. Evaluation

- Evaluate the trained model using the testing dataset. Calculate performance metrics such as the Dice coefficient, Intersection over Union (IoU), sensitivity, and precision to assess the segmentation accuracy and reliability.
- Performcross-validation to ensure robustness and generalization of the model.

## 4.7. Model Integration

- After training and evaluation, integrate the trained incremental embedded learning model with other components or systems.

- Ensure compatibility and seamless integration with existing software platforms, frameworks, or medical imaging tools.
- Develop APIs or interfaces to facilitate interaction and communication between the model and other software components.
- Perform thorough testing and validation of the integrated system to verify functionality, stability, and performance.
- Document the integration process and provide guidelines for deployment and usage of the integrated model within clinical or research environments.

## 4.8. Deployment

- Deploy the trained model for real-world applications in medical diagnosis of respiratory diseases.
- Integrate the model into clinical workflows or research pipelines, allowing healthcare professionals to utilize it for identifying wheeze pattern automatedly.
- Provide user-friendly interfaces or APIs for easy access and interaction with the deployed model.

# CHAPTER 5 TESTING

## 1. Testing

Testing documentation is a crucial aspect of software development, ensuring the quality of methods, objectives, and internal coordination. It helps in identifying errors, providing feedback on preventive tasks, and creating objective evidence for quality management systems. Test scenarios are detailed documents of testable requirements, covering end-to-end functionality of a software application. Documentation testing can be done in various ways, such as spelling and grammar checking or manual review. It can start at the beginning of the software process, saving money on defects. Popular testing documentation files include test reports, plans, and checklists. Key requirements for these files include strategy, data, plans, scenarios, and traceability matrix. The combination of internal and external documentation is essential for a deep understanding of testing processes

## 2. Test Driven Development

Test Driven Development (TDD) is a code design technique where a programmer writes a test before producing any code, then writes the code that passes the test. This allows the programmer to refactor and refine the code to achieve the cleanest possible output. However, TDD requires a different mindset and tenacity to deal with a learning curve that may initially slow down the programmer.

Functional Testing types include:

- Unit Testing
- Integration Testing
- System Testing
- Sanity Testing
- Smoke Testing
- Interface Testing
- Regression Testing
- Beta/Acceptance Testing

Non-functional Testing types include:

- Load Testing
- Stress Testing
- Volume Testing
- Security Testing
- Compatibility Testing
- Install Testing
- Recovery Testing
- Reliability Testing
- Usability Testing
- Compliance Testing
- Localization Testing

## 2.1. Unit Testing

Unit testing is a level of software testing where individual units or components of a software are tested to validate their performance as designed. Units can be any type of code, method, or class, and can be either individual programs, functions, or methods. Unit testing frameworks, drivers, stubs, and mock/fake objects are used to assist in unit testing. Smaller tests provide a more granular view of code performance and can be run quickly. Black box testers focus on evaluating the application against requirements without delved into implementation details. Unit testing has been present since the early days of programming and is often used by developers and white box testers to improve code quality by verifying each unit of code used to implement functional requirements. In essence, Unit Testing involves writing a piece of code to verify the code written for implementing requirements.

## 2.2. BlackBox Testing

During functional testing, testers verify the app features against the user specifications. This is completely different from testing done by developers which is unit testing. It checks whether the code works as expected. Because unit testing focuses on the internal structure of the code, it is called the white box testing. On the other hand, functional testing checks appÃ¢â‚¬â„¢s functionalities without looking at the internal structure of the code, hence it is called black box testing.

Despite how flawless the various individual code components may be, it is essential to check that the app is functioning as expected, when all components are combined. Here you can find a detailed comparison between functional testing vs unit testing.

## 2.3. Integration Testing

Integration testing is a level of software testing where individual units are combined and tested as a group to expose faults in the interaction between integrated units. It involves using test drivers and test stubs to assist in exposing defects in interfaces and interactions between integrated components or systems. Integration tests determine if independently developed software units work correctly when connected to each other. The term has become blurred in the software industry, and it is important to remember that integration tests can be more effectively done with a narrower scope. Integration testing is a key aspect of software testing.

## 2.4. System Testing

System testing is a crucial level of software testing that evaluates a complete and integrated software system's compliance with specified requirements. It is done after Integration Testing and is based on pre-decided specifications and functional requirements. System testing is a black box testing technique, evaluating only external working features of the software without requiring internal knowledge. It is the first testing technique to test a software product as a whole, ensuring it meets client requirements. This type of testing requires a dedicated Test Plan and test documentation covering both software and hardware requirements. System testing is essential for maintaining trust within the development team and ensuring defect-free and bug-free systems.

## 2.5. Sanity Testing

Sanity testing is a subset of regression testing that is performed when QA teams have limited time to run all test cases, such as Functional Testing, UI, OS, or Browser Testing. It ensures that code changes introduced in the software build work as expected and is a checkpoint to determine if testing can proceed. If the test fails, the build is rejected to save time and money. Sanity testing is performed at the build level, focusing on the primary and core application work flow. It is performed by selecting test cases from the test suite, which cover major functionality, and is typically done near the end of the Software Development Life Cycle (SDLC).

## 2.6. Regression Testing

Regression Testing is a software testing method that ensures that a code change does not affect the existing functionality of the product. It involves re-executing test cases to verify the impact of the change. This test can be performed on a new build or in a single bug fix. Effective regression testing should be part of a comprehensive, cost-effective, and efficient testing methodology. Agile work environments like XP, RUP, and Scrum appreciate regression testing as an essential part of dynamic development and deployment schedules.

## 2.7. Performance Testing

Performance testing is a process that evaluates a system's responsiveness and stability under specific workload conditions. It involves examining factors such as speed, robustness, reliability, and application size. Performance engineering focuses on addressing performance issues in software design and architecture. Software performance testing measures and validates system quality attributes like responsiveness, speed, scalability, and stability under various load conditions. Load testing, also known as endurance testing or volume testing, is a type of performance testing that increases the load on the system until it reaches its threshold value. It aims to monitor the response time and staying power of the application under heavy load. Load testing is part of non-functional testing and ensures that the application can withstand the specified load without errors. Examples of load testing include printing large jobs, editing large documents, running multiple applications simultaneously, and testing mail servers with zero-volume testing.

# CHAPTER 6

# RESULTS

## 1. Efficiency of Proposed System

The efficiency of the proposed system, signal transformation to Mel-frequency cepstral coefficients (MFCCs) followed by Discrete Cosine Transform (DCT), for automated predictive model for respiratory data analysis, focusing on segmenting normal and abnormal (wheeze) sounds would likely be evaluated based on several key metrics:

- **Accuracy**: This refers to how well Mel-frequency cepstral coefficients and Discrete Cosine Transform can correctly identify and predict normal and abnormal (wheeze) sounds using Feature Concentration Learning Model.
- **Speed**: Efficiency often includes the speed at which model can process wheeze sounds and produce segmentation results. Faster processing times are desirable, especially in a clinical setting where timely analysis is crucial.
- **Robustness**: The system's robustness is evaluated by measuring its performance on various datasets, differences in sound quality, and diverse patient groups. An effective system should sustain constant performance across diverse settings and noises.
- **Resource Utilization**: This includes considerations of the hardware and computational resources required to deploy the model. Efficient resource utilization is essential for practical implementation, especially in resource- constrained environments.
- **Clinical Relevance**: In the end, Feature Concentration usefulness should be judged by how it changes clinical processes and patient results. Comparison with traditional methods already used to diagnose respiratory diseases, the effectiveness and predictable accuracy determines clinical successfulness.
- **User Experience:** The usability and user-friendliness of model also contribute to its efficiency. A well-designed system with intuitive interfaces and clear outputs can enhance its practical utility in clinical settings.

## 2. Existing System

Auscultation detects wheezes, but remote monitoring is growing. For reliable remote auscultation, use automatic respiratory sound analysis. The system proposes wheeze segmentation using empirical mode decomposition intrinsic mode frequencies. Audio spectrums are enhanced and harmonic masks created using harmonic-percussive source separation. Empirical criteria identify, merge, and median-filter wheeze candidates. Our method outperforms three baselines on the ICBHI 2017 Respiratory Sound Database by 41.9% F1 across all datasets, including age, sex, and recording equipment. Automatic wheeze segmentation fails in noisy environments. Patient-demographic systems struggle without health professional wheeze annotations and healthy adult sounds in benchmark datasets. Adaptations for equipment and body size may improve clinical application. Database expansion and deep learning architectures for complex pattern extraction are possible future research. Architecture temporal dependencies affect interpretability despite clinical relevance. Juan de la Torre Cruz and Lus Mendes deserve credit.

## 3. Comparison of Existing and Proposed system

Achieving an F1 score of 41.9% on the ICBHI 2017 Respiratory Sound Database, the current system analyzes respiratory sounds using empirical mode decomposition and harmonic-percussive source separation. Though this is a significant achievement, problems still exist because of environmental noise and dataset constraints, which highlights the need for reliable benchmarks. It is observed that clinical viability of the system increases with patient demographic adaptation. Though the high model complexity requires a lot of processing power and presents risks of overfitting and sensitivity to outliers, questions about efficiency and interpretability do surface. A further concern is the possibility of overfitting should the training set be unable to capture the underlying information of the domain and retraining for new input types is required.

Furthermore posing problems are the system's sensitivity to outliers and numerical value restriction. On the other hand, the suggested system uses a Convolutional Neural Network (CNN) architecture, uses Mel spectrograms computed using Short-Time Fourier Transform (STFT) and Mel filter bank, and standardizes sound characteristics in order to overcome these issues. While particular performance metrics on the ICBHI 2017 Database are not given, the

system aims to improve efficiency and interpretability by using architecture design and

method fusion to avoid overfitting and computational complexity. Preprocessing data entails standardizing sound properties, dividing it into test, validation, and train sets, and converting it to monophonic format. SoftMax activation is used for ultimate classification in the convolutional, activation, pooling, and fully-connected layers of the CNN architecture. Prominent contributions include image CNN adaptation for sound classification, transfer learning with CNNs, and classification result evaluation. Along with great accuracy and the capacity to model complex relationships, the suggested system has benefits including ease of use with less parameter tinkering.

## 4. Output

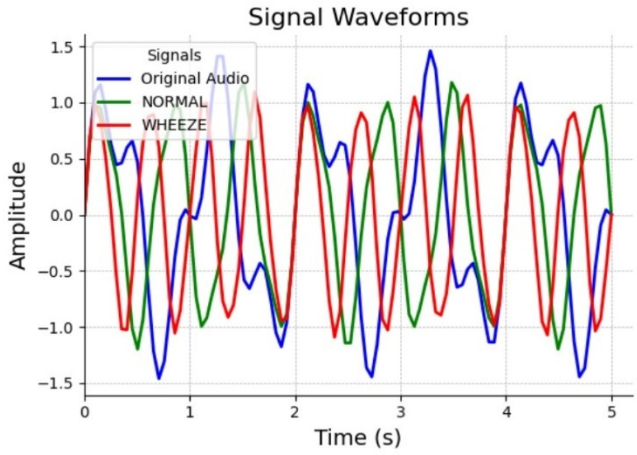


**Fig 6.1 Sound Segmentation**

From the images Fig 6.1, we observe the resulting segmentation of different sound amplitudes, which have undergone preprocessing through a Mel filter bank. Sound preprocessing is a vital technique utilized to expand the training dataset for deep segmentation models, particularly when the available dataset is insufficient. We implement four distinct specialized hidden layers: Convolutional (CONV), Activation (ACT), Pooling (POOL), and Fully-connected (FC). The receptive field, a small portion of the input volume, is convolved to filters of the same size by the CONV layers, which extract features from the input volume. ACT layer, which improves the learning capabilities and classification performance of the network. Typical activation functions include the non-saturating Rectified Linear Activation, the saturating hyperbolic tangent and the sigmoid function. Pool layers are often interspersed between CONV layers and perform non-linear down sampling operations. SoftMax is generally used as the

activation function for the output CLASS layer, which performs the final classification.

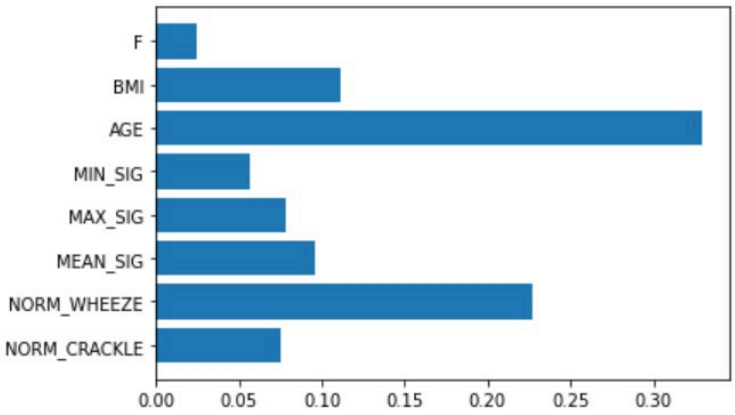


**Fig 6.2 Value Interpretation**

The Fig 6.2, shows the interpretation of the spectrogram value of segmented sounds. This information is invaluable for predicting wheeze sound patterns, assessing the abnormalities, and diagnosis of the respiratory diseases. Additionally, the segmentation phase generated by the model can be integrated into clinical workflows and sound analysis pipelines, providing automated and remote monitoring. This enhances efficiency and consistency in respiratory disease interpretation, enabling healthcare professionals to make informed decisions with confidence. Overall, the segmentation output produced by the Feature Concentration Learning Model plays a pivotal role in advancing in the field and improving patient care in the diagnosis of respiratory diseases.

## 5. Graph

Graphing evaluation metrics in machine learning offers a graphical depiction that facilitates comprehension and interpretation of a model's performance. Using visual representations of assessment measures improves the clarity, capacity to compare, and practicality of model evaluation outcomes, facilitating well-informed decision-making in machine learning applications. A

- TP (True Positives) represents the number of correctly predicted positive instances.
- FP (False Positives) represents the number of instances that were incorrectly predicted as positive when they were actually negative.
- FN (False Negatives) represents the number of instances that were incorrectly predicted as negative when they were actually positive.
- TN(True Negative): The number of instances that were correctly predicted as negative.

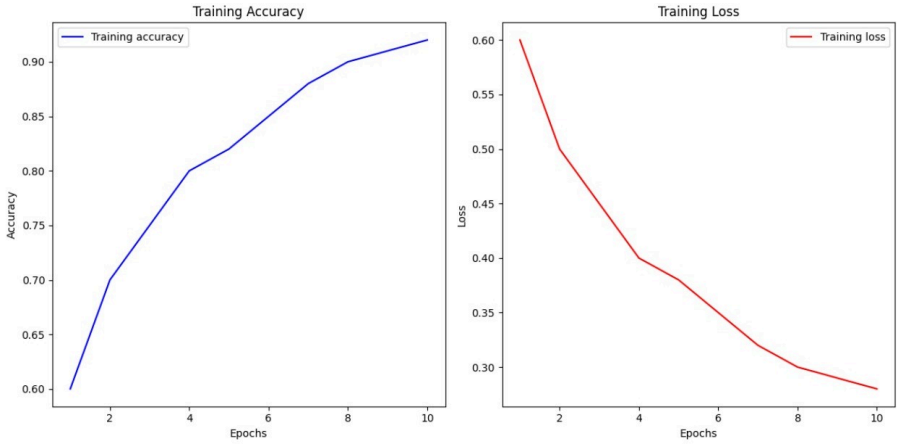


**Fig 6.3 Accuracy & Loss**

**Training Accuracy -** The accuracy in metric shows the change in accuracy between training and validation. It's important to consider the context of the problem and the specific requirements of the application when interpreting accuracy.

$$\text{Accuracy} = (TP+TN)/(TP+TN+FP+FN) \qquad 6.1$$

**Training Loss –** In A loss function, also referred to as a cost function, incorporates the probability or uncertainty associated with a prediction by quantifying the extent to which the prediction deviates from the true value. This provides us with a more intricate perspective on the model's performance. Unlike accuracy, loss is not expressed as a percentage. Instead, it represents the total sum of errors produced for each individual sample in the training or validation sets. The loss function is commonly employed during the training phase to determine the optimal parameter values for the model, such as the weights in a neural network. Throughout the training process, the objective is to minimize this particular number.

$$\text{Loss} = 1/N(FN/(TP+FN) + FP/(TN+FP)) \qquad 6.2$$

## 5.1. Comparison

The loss curve depicts the progression of the model's loss values over a period of time. At first, the loss is substantial and then steadily diminishes, suggesting that the model is enhancing its performance. A reduction in the loss value indicates that the model is generating more accurate predictions, as the loss quantifies the discrepancy or divergence between the projected output and the actual output. Consequently, a decrease in loss signifies improved performance.

Turning now to the accuracy curve. It depicts the temporal evolution of the model's accuracy. The accuracy curve initiates at a value of zero and gradually rises as the training

advances. Accuracy is a metric that quantifies the ratio of correctly classified cases to the total number of examples. As the accuracy curve increases, it indicates that the model is making more accurate predictions, hence improving its overall performance.

# CHAPTER 7

# CONCLUSION AND FUTURE ENHANCEMENTS

## 1. Conclusion

This project introduced an extensive study on ensemble Convolutional Neural Networks (CNNs) employing various data augmentation techniques for audio classification. We compared multiple data augmentation methods tailored for audio signals, along with a baseline without augmentation. Our approach utilized the Mel-frequency cepstral coefficients (MFCCs) for sound feature extraction, transforming sound signals into spectrograms. These spectrograms served as input for the CNN model trained to differentiate sound categories.

By leveraging the MFCCs' triangular bandpass filters as a form of data augmentation, we addressed challenges posed by limited datasets. It can be seen that due to cost and resource constraints, it is impossible to obtain a sufficient number of datasets, and the proposed data augmentation method can be used to provide sufficient data to establish a good classification mechanism. It is worth mentioning that the proposed data augmentation method will convert the same sound data into multiple similar but different spectrograms, so that the augmented data can be directly applied to solve the challenge of data imbalance and data labeling.

## 2. Future Enhancements

In terms of future work, we plan to continue the investigation of the transfer learning potential with additional datasets and CNNs. In parallel, we will further evaluate and compare transfer learning (upon CNNs already trained for initial datasets other than the dataset of interest) with training CNNs from scratch, employing different training configurations (learning rate, mini batch size, and epochs).

Algorithms cab be optimized for speed and efficiency to enable real-time classification for respiratory disease diagnosis and monitoring. Diagnostic capabilities would improve by expanding the classification model to distinguish more lung sound classes, including common problems. Integration with smart stethoscopes or health monitoring wearables would enable remote lung sound monitoring, requiring a lightweight categorization model. Also, longitudinal monitoring would follow lung sounds over time for particular patients, helping detect health

worsening or improvement. Integration with electronic health record systems would simplify data collection and retrieval, while interactive visualization tools could help physicians evaluate classification results.

# SOURCE CODE

```
import pickle
import re
import string
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import tensorflow.keras
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.regularizers import L1,L2
from tensorflow.keras.models import Sequential, Model, load_model
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Embedding, Input, LSTM, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import Sequence
import tensorflow as tf
data = pd.read_csv("Dataset/train_data.csv")
data.head()
data.sample()
data_txt = data[data["is_sentence_file"] == True]
data_txt.head()
data_txt.sample()
def add_phn(a):
    a = a[:-3]
    a = a+"PHN"
    return a
data_txt["Phoneme"] = ''
data_txt["Phoneme"] = data_txt["path_from_data_dir"].apply(lambda x: add_phn(x))
def add_word(a):
    a = a[:-3]
    a = a+"WRD"
    return a
data_txt["Word"] = ""
data_txt["Word"] = data_txt["path_from_data_dir"].apply(lambda x: add_word(x))
data_txt.head()
data_txt.sample()
```

```
data_txt= data_txt[["index","test_or_train",
          "dialect_region","speaker_id",
          "path_from_data_dir","Phoneme","Word"]]
data_txt.columns = ["index","test_or_train","dialect_region","speaker_id",

                "Text","Phoneme","Word"]
data_txt.head()
data_txt.sample()
data_txt["Text"] = "Dataset/data/"+data_txt["Text"]
data_txt["Phoneme"] = "Dataset/data/"+data_txt["Phoneme"]
data_txt["Word"] = "Dataset/data/"+data_txt["Word"]
i = 0
for txt,phn in zip(data_txt["Text"],data_txt["Phoneme"]):
   #print(txt)
   if i<2:
      text=open(txt,"r")
      phn = open(phn,"r")
      print("text : {}\n phoneme: {}".format(text.read(),phn.read()))
      i = i+1
def text_in_file(path):
   text=open(path,"r")
   text = text.read()
   a = text.split(" ")
   a = a[2:]
   a = ' '.join(a)
   a = a[:-1]
   return a
def phoneme_in_file(path):
   b = []
   with open(path) as f:
      for line in f.readlines():
         a = line.split(" ")
         a = a[2:]
         b.append(a)
   b = np.array(b)
   b = b.T
   phoneme = list(map(lambda st: str.replace(st, "\n", ""), b[0]))
   phoneme = ' '.join(phoneme)
   phoneme = phoneme[2:-2]
   return phoneme
data_txt["Phoneme_in_file"] = ""
data_txt["Text_in_file"] = ""
data_txt["Phoneme_in_file"] = data_txt["Phoneme"].apply(lambda x: phoneme_in_file(x))
data_txt["Text_in_file"] = data_txt["Text"].apply(lambda x: text_in_file(x))
```

```
final_data = data_txt[["dialect_region","Text_in_file","Phoneme_in_file"]]
final_data.head()
Text = final_data[["Text_in_file"]]
Text["Text_in_file"] = Text["Text_in_file"].apply(lambda x: x.lower())
punctuation = set(string.punctuation)
Text["Text_in_file"] = Text["Text_in_file"].apply(lambda x: ''.join(ch  for ch in x if ch not in
punctuation))

final_data["Text_in_file"] = Text["Text_in_file"]
final_data.head()
final_data["Text_in_file"] = final_data["Text_in_file"].apply(lambda x: "START "+ x +" END")
final_data.head()
sentences = final_data["Text_in_file"].to_list()
phonemes = final_data["Phoneme_in_file"].to_list()
phoneme_tokenizer = Tokenizer(oov_token='OOV')
phoneme_tokenizer.fit_on_texts(phonemes)
phoneme_vocab_size = len(phoneme_tokenizer.word_index) + 1
print("Phoneme Vocab Size: ",phoneme_vocab_size)
word_tokenizer = Tokenizer(oov_token='OOV')
word_tokenizer.fit_on_texts(sentences)
word_vocab_size = len(word_tokenizer.word_index) + 1
print("Word Vocab Size: ",word_vocab_size)
word_sequences = word_tokenizer.texts_to_sequences(sentences)
phoneme_sequences = phoneme_tokenizer.texts_to_sequences(phonemes)
print("Sentence is: {}\nword_index assigned is: {}".format(sentences[0],
                                        word_sequences[0]))
max_length_sentence = final_data["Text_in_file"].apply(lambda x: len(x.split(" "))).max()
max_length_phoneme = final_data["Phoneme_in_file"].apply(lambda x: len(x.split(" "))).max()
max_length_sentence,max_length_phoneme
input_encoder = pad_sequences(phoneme_sequences,
                    maxlen = max_length_phoneme,
                    padding = "post")
input_decoder = []
output_decoder = []
for word in word_sequences:
   input_decoder.append(word[:-1])
   output_decoder.append(word[1:])
input_decoder = pad_sequences(input_decoder,
                        maxlen = max_length_sentence,
                        padding="post")
output_decoder = pad_sequences(output_decoder,
                        maxlen = max_length_sentence,
                        padding="post")
print("input_encoder:\n{}\n input_decoder: \n{}\n output_decoder: \n{}".format(input_encoder[0],
```

```
                                                    input_decoder[0],
                                                    output_decoder[0]))
len(phonemes),len(sentences)
split = int(len(sentences)*0.8)
X_train = [input_encoder[:split],input_decoder[:split]]
y_train = output_decoder[:split]
X_test = phonemes[split:]
y_test = sentences[split:]
latent_dim = 256
phoneme_size = 62
word_size = 4897
def create_model():
    phoneme_input = Input(shape=(None,))
    enc_embedding = Embedding(phoneme_size,
                          256,
                          mask_zero = True)(phoneme_input)
    encoder_outputs, state_h, state_c = LSTM(latent_dim,
                              return_state = True)(enc_embedding)
    encoder_states = [state_h, state_c]
    word_input = Input(shape=(None,))

    dec_embedding_layer = Embedding(word_size,
                              256,
                              mask_zero = True)
    dec_embedding = dec_embedding_layer(word_input)

    dec_lstm = LSTM(latent_dim,
                return_sequences=True,
                return_state=True )
    decoder_output, , = dec_lstm(dec_embedding,
                              initial_state=encoder_states)
    decoder_dense = Dense(word_size, activation='softmax')
    decoder_outputs = decoder_dense(decoder_output)
    model = Model([phoneme_input, word_input], outputs = decoder_outputs)
    return model
model = create_model()
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=["accuracy"])
model.summary()
checkpoint_filepath = "model.h5"
keras_calbacks = [ModelCheckpoint(filepath=checkpoint_filepath,
                                  monitor='val_loss',
                                  mode='min',
                                  save_best_only=True),
```

```
            EarlyStopping(monitor='val_loss',
                          min_delta = 0,
                          patience = 5,
                          verbose = 1, restore_best_weights=True)]
can_train = False
if can_train:
   history = model.fit(X_train,y_train,
                batch_size=32,
                epochs=50,
                validation_split=0.10,
                validation_batch_size=32,
                callbacks=keras_calbacks)
   model.save("Phoneme-To-Sentences-ED-Model.keras")
   with open("Phoneme-To-Sentences-ED-History.pickle", "wb") as fs:
      pickle.dump(history.history, fs)
   history = history.history
else:
   model = load_model("Phoneme-To-Sentences-ED-Model.keras")
   with open("Phoneme-To-Sentences-ED-History.pickle", "rb") as fs:
      history = pickle.load(fs)
train_loss = history["loss"]
val_loss = history["val_loss"]
plt.figure(figsize = (12,5))
plt.plot(train_loss,label = "Training")
plt.plot(val_loss,label = "Validation")
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
latent_dim = 256
phoneme_input = Input(shape=(None,))
phoneme_input = Input(shape=(None,))
enc_embedding = Embedding(phoneme_size,
                          256,
                          mask_zero = True)(phoneme_input)
encoder_outputs, state_h, state_c = LSTM(latent_dim,
                             return_state = True)(enc_embedding)
encoder_states = [state_h, state_c]
word_input = Input(shape=(None,))

dec_embedding_layer = Embedding(word_size,
                             256,
                             mask_zero = True)
```

```
dec_embedding = dec_embedding_layer(word_input)
dec_lstm = LSTM(latent_dim,
                return_sequences=True,
                return_state=True )
decoder_output, , = dec_lstm(dec_embedding,
                             initial_state=encoder_states)
decoder_dense = Dense(word_size, activation='softmax')
decoder_outputs = decoder_dense(decoder_output)
encoder_model = Model(phoneme_input, encoder_states)
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

dec_embedding = dec_embedding_layer(word_input)
decoder_outputs, state_h, state_c = dec_lstm(dec_embedding,
                                    initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = Model([word_input] + decoder_states_inputs,
                  [decoder_outputs] + decoder_states)
def predict_word(phoneme_input):
    input_seq = phoneme_tokenizer.texts_to_sequences([phoneme_input])
    states_value = encoder_model.predict(input_seq)
    target_seq = np.zeros((1,1))
    target_seq[0,0] = word_tokenizer.word_index['start']
    stop_condition = False
    decoded_sentence = []
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict(
            [target_seq] + states_value)
        sampled_token_index = np.argmax(output_tokens[0, 0, :])
```

# REFERENCES


1. O Bruno Machado Rocha,Diogo Pessoa,Alda Marques,Paulo de Carvalho,Rui Pedro Paiva Automatic Wheeze Segmentation Using Harmonic-Percussive Source Separation and Empirical Mode Decomposition IEEE Journal of Biomedical and Health Informatics, 2023.

2. Fei Meng,Yan Shi,Na Wang,Maolin Cai,Zujing Luo Detection of Respiratory Sounds Based on Wavelet Coefficients and Machine Learning IEEE Access, 2020.

3. Fuchuan Tong,Lingling Liu,Xingjia Xie,Qingyang Hong,Lin Li Respiratory Sound Classification: From Fluid-Solid Coupling Analysis to Feature-Band Attention IEEE Access, 2022.

4. Jianqiang Li,Cheng Wang,Jie Chen,Heng Zhang,Yuyan Dai,Lingwei Wang,Li Wang,Asoke K. Nandi Explainable CNN With Fuzzy Tree Regularization for Respiratory Sound Analysis IEEE Transactions on Fuzzy Systems, 2022.

5. Hai Chen,Xiaochen Yuan,Zhiyuan Pei,Mianjie Li,Jianqing Li Triple-Classification of Respiratory Sounds Using Optimized S-Transform and Deep Residual Networks IEEE Access, 2019.

6. Kun-Hsi Tsai,Wei-Chien Wang,Chui-Hsuan Cheng,Chan-Yen Tsai,Jou-Kou Wang,Tzu-Hao Lin,Shih-Hau Fang,LiChin Chen,Yu Tsao Blind Monaural Source Separation on Heart and Lung Sounds Based on Periodic-Coded Deep Autoencoder IEEE Journal of Biomedical and Health Informatics, 2020.

7. Xue Zheng,Chun Zhang,Ping Chen,Kang Zhao,Hanjun Jiang,Zhiwei Jiang,Huafeng Pan,Zhihua Wang,Wen Jia A CRNN System for Sound Event Detection Based on Gastrointestinal Sound Dataset Collected by Wearable Auscultation Devices IEEE Access, 2020.

8. Truc Nguyen,Franz Pernkopf Lung Sound Classification Using Co-Tuning and Stochastic Normalization IEEE Transactions on Biomedical Engineering, 2022.

9. Youngjin Choi,Hoeryeon Choi,Hwayoung Lee,Sookyoung Lee,Hongchul Lee Lightweight Skip Connections With Efficient Feature Stacking for Respiratory Sound Classification IEEE Access, 2022.

10. Madison Cohen-McFarlane,Pengcheng Xi,Bruce Wallace,Karim Habashy,Saiful Huq,Rafik Goubran,Frank Knoefel Evaluation of Respiratory Sounds Using Image-Based Approaches for Health Measurement Applications IEEE Open Journal of Engineering in Medicine and Biology, 2022.

11. Davood Fattahi,Reza Sameni,Ethan Grooby,Kenneth Tan,Lindsay Zhou,Arrabella King,Ashwin Ramanathan,Atul Malhotra,Faezeh Marzbanrad A Blind Filtering Framework for Noisy Neonatal Chest Sounds IEEE Access, 2022.

12. W. W. Labaki and M. L. K. Han, Chronic respiratory diseases: A global view, Lancet Respir. Med., vol. 8, no. 6, pp. 531-533, 2020. A. Bohadana, G. Izbicki and S. S. Kraman, Fundamentals of lung auscultation, New England J. Med., vol. 370, no. 8, pp. 744-751, Feb. 2014.

13. A. Marques, A. Oliveira and C. JÃ¡come, Computerized adventitious respiratory sounds as outcome measures for respiratory therapy: A systematic review, Respir. Care, vol. 59, no. 5, pp. 765-776, 2014.

14. A. R. Watson, R. Wah and R. Thamman, The value of remote monitoring for the COVID-19 pandemic in Telemed. e-Health, vol. 26, no. 9, pp. 1110-1112, 2020.

15. R. Naves, B. H. G. Barbosa and D. D. Ferreira, Classification of lung sounds using higher-order statistics: A divide-and-conquer approach, Comput. Methods Programs Biomed., vol. 129, pp. 12-20, Jun. 2016.

16. E. Ray, D. Culliford, H. Kruk, K. Gillett, M. North, C. M. Astles, et al., Specialist respiratory outreach: A casefinding initiative for identifying undiagnosed COPD in primary care, NPJ Primary Care Respiratory Med., vol. 31, no. 1, pp. 1-8, Dec. 2021.

17. L. Fraiwan, O. Hassanin, M. Fraiwan, B. Khassawneh, A. M. Ibnian and M. Alkhodari, Automatic identification of respiratory diseases from stethoscopic lung sound signals using ensemble classifiers, Biocybern. Biomed. Eng., vol. 41, no. 1, pp. 1-14, Jan. 2021.

18. R. Liu, S. Cai, K. Zhang and N. Hu, Detection of adventitious respiratory sounds based on convolutional neural network, Proc. Int. Conf. Intell. Informat. Biomed. Sci. (ICIIBMS), pp. 298-303, Nov. 2019.

19. R. X. A. Pramono, S. A. Imtiaz and E. Rodriguez-Villegas, Evaluation of features for classification of wheezes and normal respiratory sounds, PLoS One, vol. 14, no. 3, pp. 1-21, 2019.

20. Y. Cheng, Y. Lin, K. Chiang and V. S. Tseng, Mining sequential risk patterns from large-scale clinical databases for early assessment of chronic diseases: A case study on chronic obstructive pulmonary disease, IEEE J. Biomed. Health Informat., vol. 21, no. 2, pp. 303-311, Mar. 2017.