

A MINOR PROJECT REPORT
ON
EARLY AND ACCURATE DISEASE DETECTION
AND DIAGNOSIS SYSTEM

SUBMITTED IN PARTIAL FULFILLMENT FOR THE AWARD OF DEGREE OF

BACHELOR OF TECHNOLOGY
IN ELECTRONICS AND COMMUNICATION ENGINEERING



Submitted by:

HARSH KUMAR MAHOUR – 9919102051

PRANAV PANDEY – 9919102049

RAHUL PAMNANI – 9919102059

Under the Guidance of

DR. BHARTENDU CHATURVEDI

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

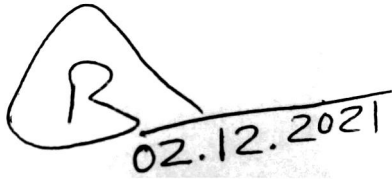
JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA (U.P.)

December, 2021

CERTIFICATE

This is to certify that the minor project report entitled, “**EARLY AND ACCURATE DISEASE DETECTION AND DIAGNOSIS SYSTEM**” submitted by **Pranav Pandey, Harsh Kumar Mahour, Rahul Pamnani** in partial fulfillment of the requirements for the award of Bachelor of Technology Degree in **Electronics and Communication Engineering** of the Jaypee Institute of Information Technology, Noida is authentic work carried out by them under my supervision and guidance. The matter embodied in this report is original and has not been submitted for the award of any other degree.

Signature of Supervisor:



02.12.2021

Name of the Supervisor: Dr. Bhartendu Chaturvedi

ECE Department,

JIIT, Sec-128,

Noida-201304

Dated: December 2, 2021

DECLARATION

We hereby declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, have been adequately cited, and referenced the sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission.

Place: Noida

Date: December 2, 2021



Name: Pranav Pandey

Enrollment: 9919102049



Name: Harsh Kumar Mahour

Enrollment: 9919102051



Name: Rahul Pamnani

Enrollment: 9919102059

Table of Contents

Certificate	ii
Declaration	iii
Abstract	vi
1. Introduction	01
1.1. Physical Effects	01
1.2. COVID-19 Pneumonia	02
1.3. Acute Respiratory Distress Syndrome (ARDS)	02
1.4. Sepsis	02
1.5. Superinfection	02
1.6. Concluding Remarks	02
2. Literature Survey	03
2.1. Research paper summary	03
2.2. Concluding Remarks	05
3. Implementation	09
3.1. Libraries and Algorithms	09
3.1.1. Csv	09
3.1.2. Cv2 (OpenCV)	09
3.1.3. Librosa	10
3.1.4. Pandas	10
3.1.5. Numpy	11
3.1.6. Matplotlib	11
3.1.7. IPython	11
3.1.8. TensorFlow	11
3.1.9. Keras	12
3.1.10. Layers	12
3.1.11. Utils	13
3.2. Pre-Emphasis	14
3.2.1. Windowing	14
3.2.2. DFT	14
3.2.3. Mel Filterbank	15
3.2.4. DCT	16
3.2.5. Convolutional Neural Network	17
3.3. Steps and Procedure	17
3.4. Concluding Remarks	27

4. Results and Verification	28
4.1 Model Loss	28
4.2 Predictions based on Original Data	28
4.3 Predictions based on Augmented Data	29
4.4 Web Portal	31
5. Conclusion and Future Scopes	34
5.1 Conclusion	34
5.2 Future Scopes	34
References	

Abstract

The COVID-19 epidemic, in which millions of people suffer, has affected the whole world in a short time. This virus, which has a high rate of transmission, directly affects the respiratory system of people. While symptoms such as difficulty in breathing, cough, and fever are common, hospitalization and fatal consequences can be seen in progressive situations. For this reason, the most important issue in combating the epidemic is to detect COVID19(+) early and isolate those with COVID-19(+) from other people. In addition to the RT-PCR test, those with COVID-19(+) can be detected with imaging/voice detection methods. In this study, it was aimed to detect COVID-19(+) patients with cough acoustic data, which is one of the important symptoms. Based on these data, features were obtained from traditional feature extraction methods using empirical mode decomposition (EMD) and discrete wavelet transform (DWT). Deep features were also obtained using pre-trained ResNet50 and pre-trained MobileNet models. Feature selection was applied to all obtained features with the ReliefF algorithm. In this case, the highest 98.4% accuracy and 98.6% F1-score values were obtained by selecting the EMD + DWT features using ReliefF. In another study in which deep features were used, features obtained from ResNet50 and MobileNet using scalogram images were used. For the features selected using the ReliefF algorithm, the highest performance was found with support vector machines-cubic as 97.8% accuracy and 98.0% F1-score. It has been determined that the features obtained by traditional feature approaches show higher performance than deep features. Among the chaotic measurements, the approximate entropy measurement was determined to be the highest distinguishing feature. According to the results, a highly successful study is presented with cough acoustic data that can easily be obtained from mobile and computer-based applications. We anticipate that this study will be useful as a decision support system in this epidemic period, when it is important to correctly identify even one person.

The early detection of COVID-19 is a challenging task due to its deadly spreading nature and existing fear in the minds of people. Speech-based detection can be one of the safest tools for this purpose as the voice of the suspect can be easily recorded. The Mel Frequency Cepstral Coefficient (MFCC) analysis of speech signals is one of the oldest but potential analysis tools. The performance of this analysis mainly depends on the use of conversion between normal frequency scale to perceptual frequency scale and the frequency range of the filters used. Traditionally, in speech recognition, these values are fixed. But the characteristics of speech signals vary from disease to disease. In the case of detection of COVID19, mainly the coughing sounds are used whose bandwidth and properties are quite different from the complete speech signal. By exploiting these properties, the efficiency of COVID-19 detection can be improved. To achieve this objective the frequency range and the conversion scale of frequencies have been suitably optimized. Further to enhance the accuracy of detection performance, speech enhancement has been carried out before extraction of features. Finally, the performance of these features has been compared with the pretested data.

Chapter 1

Introduction

Coronavirus disease 19 (COVID-19) which exhibits acute respiratory syndrome is a deadly viral infection. As reported, it started in Wuhan, China in 2019 and has affected the whole world. As per the report of the World Health Organization, more than a hundred million people have suffered till 7th March 2021 out of which more than 2.5 million deaths have been reported. The social distancing of 1.6 m to 3 m is recommended to control the rapid spreading of COVID-19 cases. It is observed from the experiences of the medical practitioners that rather than the deadly nature of the virus, its fear of stigma is stopping people from going to medical laboratories for testing purposes.

COVID-19 can cause lung complications such as pneumonia and, in the most severe cases, acute respiratory distress syndrome, or ARDS. Sepsis, another possible complication of COVID-19, can also cause lasting harm to the lungs and other organs.

1.1 Physical Effects

Once in the chest, the virus begins to impact a person's airways — causing inflammation. As inflammation increases, a barking, dry cough that sounds and feels like asthma develops. In addition, this can cause chest tightness or deep pain while breathing. Even though it's generally mild for some people, the swelling and tightness that results from airway inflammation are essentially like having a sprained windpipe. Think of it like having a sprained ankle, but the effects and discomfort that come with having a sprain are felt inside of your chest. For some people, the infection becomes more serious and the lung tissue itself becomes swollen and filled with fluid and debris from dead cells — which is clinically referred to as **pneumonia**.

This fluid build-up can affect a person's oxygen levels, and pneumonia can be mild, moderate, severe or even life-threatening, depending on how impaired gas transfer becomes and how difficult it is to breathe. If the transfer of oxygen into the bloodstream is reduced, a person will often need supplemental oxygen and very close monitoring in a hospital setting. In very serious cases, a person may need to be placed on ventilator support in the ICU.

1.2 COVID-19 Pneumonia

In pneumonia, the lungs become filled with fluid and inflamed, leading to breathing difficulties. For some people, breathing problems can become severe enough to require treatment at the hospital with oxygen or even a ventilator. Pneumonia that COVID-19 causes tend to take hold in both lungs. Air sacs in the lungs fill with fluid, limiting their ability to take in oxygen and causing shortness of breath, cough and other symptoms.

While most people recover from pneumonia without any lasting lung damage, pneumonia associated with COVID-19 can be severe. Even after the disease has passed, lung injury may result in breathing difficulties that might take months to improve.

1.3 Acute Respiratory Distress Syndrome (ARDS)

As COVID-19 pneumonia progresses, more of the air sacs become filled with fluid leaking from the tiny blood vessels in the lungs. Eventually, shortness of breath sets in and can lead to acute respiratory distress syndrome (ARDS), a form of lung failure. Patients with ARDS are often unable to breathe on their own and may require ventilator support to help circulate oxygen in the body. Whether it occurs at home or in the hospital, ARDS can be fatal. People who survive ARDS and recover from COVID-19 may have lasting pulmonary scarring.

1.4 Sepsis

Another possible complication of a severe case of COVID-19 is sepsis. Sepsis occurs when an infection reaches and spreads through the bloodstream, causing tissue damage everywhere it goes. Lungs, hearts, and other body systems work together like instruments in an orchestra. In sepsis, the cooperation between the organs falls apart. Entire organ systems can start to shut down, one after another, including the lungs and heart. Sepsis, even when survived, can leave a patient with lasting damage to the lungs and other organs.

1.5 Superinfection

When a person has COVID-19, the immune system is working hard to fight the invader. This can leave the body more vulnerable to infection with another bacterium or virus on top of the COVID-19 — a superinfection. More infection can result in additional lung damage.

1.6 Concluding Remarks

A human being suffers from many different diseases. Diseases can have a physical, but also a psychological impact on people. Mainly for four reasons, diseases are formed: (i) infection, (ii) deficiency, (iii) heredity and (iv) body organs dysfunction. In our society, doctors or medical professionals have the responsibility to detect and diagnose appropriate disease and provide medical therapies or treatments to cure or restrain the disease. Some diseases are cured after treatment, but chronic diseases are never cured despite the treatment; and some chronic diseases lead up to global pandemics like the COVID-19 pandemic that humankind is currently facing. This chapter is all based on about helping the common people get an early diagnosis and detection of COVID-19 through the “Early and Accurate Disease Diagnosis and Detection System”. This system aims to provides a user-friendly and hassle-free experience while maintaining the accuracy and precision of the diagnosis of COVID-19. Different databases have been used from Kaggle machine learning database to implement the CNN model.

Chapter 2

Literature Survey

2.1 Research paper summary

Under such circumstances, it has become a huge challenge to develop an appropriate method for the early detection of this disease. It is a fact that the speech-based detection of COVID-19 is a simpler and safer approach for this purpose. In this section, a review of related literature is carried out in two parts: Speech-based COVID-19 detection and speech recognition using MFCC features. In this section, the literature review has been carried out in two parts: speech-based COVID-19 detection and use of MFCC features based on speech recognition and various other features that are frequently used in speech recognition.

Machine learning ML extracts features from raw data and creates a dense representation of the content. This forces us to learn the core information without the noise to make inferences (if it is done correctly). ML approaches have long been employed for the development of diagnosis and treatment systems. Now this pandemic has created a new challenge for this field of science. Developing intelligent systems that can help practitioners in terms of diagnosis, monitoring, prediction of patient's conditions, and offering treatment measures can be very helpful to help the already under pressure health systems. ML applications can potentially aid to solve the problem of the "iron triangle" in the healthcare sector. It involves three interlocking factors which are, namely, access, affordability, and effectiveness. ML is usually used for diagnosis and treatment recommendations, patient engagement, and adherence or administration activities. This chapter aims to perform a comprehensive survey on the applications of ML in battling against the difficulties the outbreak has caused and to explain the most common techniques and the biggest challenges in disease detection and summarize the various results from the newest papers. In this sense, we tried to cover every way that ML approaches have been employed and to cover all the research until the writing of this paper. Surely this would result in covering a large number of researches that are hard to put on the same canvas. Such a picture, although full of details, is very helpful in understanding where ML sits in the current pandemonium. Since the pandemic is a new and developing problem, much of the research has not yet been peer-reviewed. Therefore, this paper also covers pre-print works. This report tried to conclude the paper with ideas on how the problems can be tackled in a better way and provide some suggestions for the future. Therefore, it is very important to compare the results of a newly created model produced with the newest information and not only with state-of-the-art methods. This chapter is a source of such information for researchers for them to be precisely correct in results comparison before publishing new achievements in this field.

The libraries that were used for executing the research are:

1. ScienceDirect
2. IEEE Xplore
3. ResearchGate

The surveyed papers were grouped into two categories: MFCCs for audio feature extraction and Convolutional Neural Networking for audio post-processing. MFCCs play a vital role in learning about sounds because they are coefficients based on Mel-spectrograms which is one of the best ways to visually represent sound. The Mel frequency cepstral coefficients (MFCC) method is a reliable method with high accuracy for high-quality audio recordings. The accuracy rate is more than 90%. The high accuracy of the MFCC method is due to the Mel scale which has characteristics similar to human hearing. The word sample is extracted by the MFCC method, the value of the frequency component will match the characteristics of the Mel scale.

The resulting features represent human hearing. Mel scale sensitivity is one of the factors that increase the accuracy value. This method is superior for the identification of high-quality audio recordings. In daily life, there are a variety of complex sound sources. It is important to effectively detect certain sounds in some situations. It is necessary to distinguish the sound of coughing. To estimate suspected patients in the population, a method for cough recognition based on a Mel-spectrogram can be useful. (A spectrogram is a visual way of representing the signal strength, or “loudness”, of a signal over time at various frequencies present in a particular waveform. Not only can one see whether there is more or less energy at, for example, 2 Hz vs 10 Hz, but one can also see how energy levels vary over time. In other sciences, spectrograms are commonly used to display frequencies of sound waves produced by humans, machinery, animals, whales, jets, etc., as recorded by microphones. In the seismic world, spectrograms are increasingly being used to look at the frequency content of continuous signals recorded by individuals or groups of seismometers to help distinguish and characterize different types of earthquakes or other vibrations in the earth).

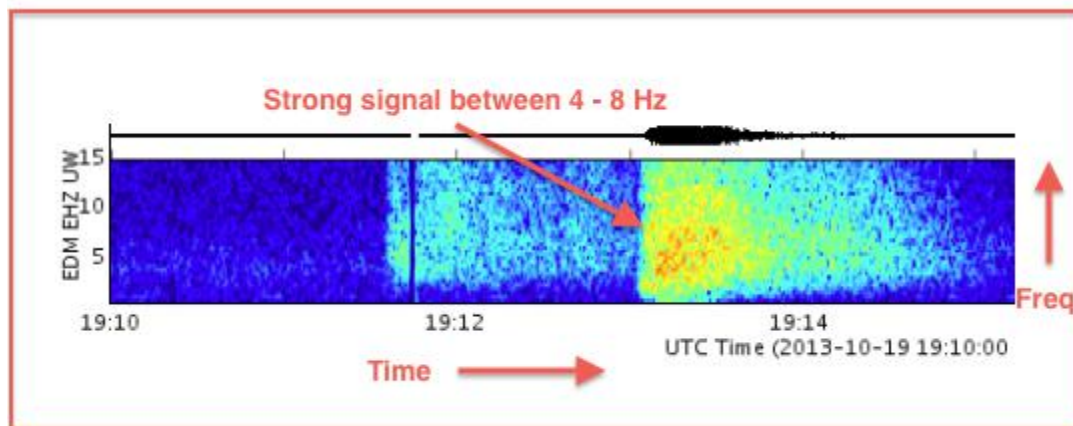


Fig. 2.1 Example of a spectrogram

The development of deep learning has contributed significantly to improving the power and capabilities of recent AI applications. Several deep learning-based convolutional neural network (CNN) architectures, e.g. AlexNet, VGGNet, and ResNet, have been presented and applied successfully in different areas.

In the literature, different types of audio categorization have been well investigated and among the various machine learning-based techniques, Convolutional Neural Network (CNN) based approaches are a popular choice for this task. The Convolutional Recurrent Neural Networks are the combination of two of the most prominent neural networks. The CRNN (convolutional recurrent neural network) involves CNN (convolutional neural network) followed by RNN (Recurrent neural networks). The presented network is similar to the CRNN but generates better or optimal results, especially towards audio signal processing. The relative success of using a CRNN for cough detection from recorded audio via image recognition serves as the basis for the machine learning model used for this system.

The diagnosis database of the research consists of 268 total cough sound items, distributed among bronchiolitis, pertussis, and bronchitis at counts of 35, 131, and 102, respectively. The database is again split into 70 % training, 15 % validation, and 15 % testing sets. The model is trained using early stopping criteria on the validation dataset so that the model training stops when it performs best on the validation dataset. The system came with an accuracy of 93.87% and 100% for pertussis and bronchitis respectively.

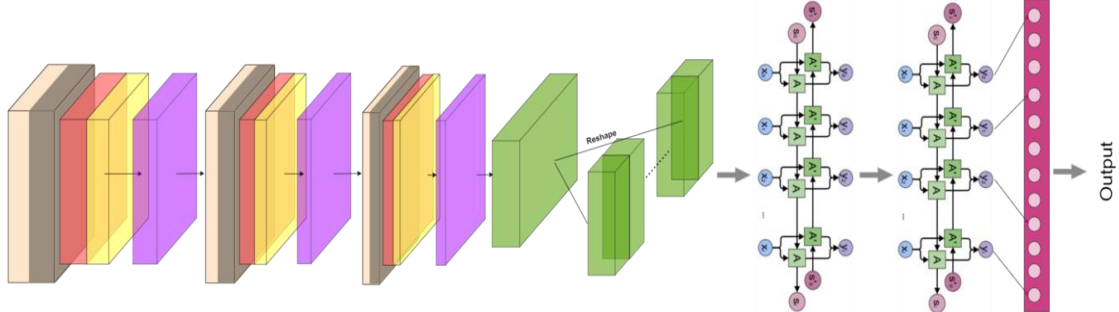


Fig. 2.2 Convolutional Neural Recurrent Networking (CRNN)

2.2 Concluding Remarks

Vishnu Vandana Kolisetty et. al. [1] have explained that the big data revolution is changing how it works and thinks, by facilitating improvements in vision and decision-making. However, the technical dilemma of big science is that no knowledge can manage and analyze large amounts of actively growing data and obtain valuable information. However, conventional machine learning approaches have been extended to meet the needs of other applications, but with increasing amounts of information or large databases, there are significant challenges for ML algorithms for big data analysis. It

discusses the ability to extract value from large data for decision-making and predictive analytics through data transformation and knowledge extraction.

Extracted conclusion

This research paper provides strong pillars for the idea of analyzing the data and the significance of machine learning for data analysis in the particularly big data domain, which is just a reference to a vast collection of data and how reliable it is.

Ian McLoughlin et. al. [2] have explained that while convolutional neural networks (CNN) produce state-of-the-art results in many applications including biomedical audio analysis, they are not robust to variability in the data that is not. It is represented by the training set. An important source of variability in biomedical images is the appearance of objects such as contrast and texture due to different audio settings. The presented NSL transforms its input feature map at a given pixel by computing its similarity to the surrounding neighbourhood. This transformation is spatially varying, hence not a convolution. It is differentiable; therefore, networks including the presented layer can be trained in an end-to-end manner.

Extracted conclusion

This research paper motivated us to CNN model for audio processing and tells us about the robust nature of the CNN model in audio processing - motivated by the inherent image-like nature of the spectrogram representation – and encouraged by recently reported good CNN performance.

Charles Bales et. al. [3] have explained that one of the notable modern medical concerns that impose an immense worldwide health burden is respiratory infections. Since cough is an essential symptom of many respiratory infections, an automated system to screen for respiratory diseases based on raw cough data would have a multitude of beneficial research and medical applications. In this it has been presented that a low complexity, automated recognition, and diagnostic tool for screening respiratory infections that utilize Convolutional Neural Networks to detect cough within the environment audio and diagnose three potential illnesses based on their unique cough audio features.

Extracted conclusion

From this research paper, it is concluded that the approach uses cough audio and converts it to Mel-spectrograms for both detection and diagnosis. With only a small number of modifications, the ML model can be trained sufficiently.

Emre C et. al. [4] have expelld that sound events often occur in unstructured environments where they exhibit wide variations in their frequency content and temporal structure. Convolutional neural networks (CNN) can extract higher-level features that are invariant to local spectral and temporal variations. Recurrent neural networks (RNNs) are useful in learning the long-term temporal context in the audio signals. CNN's and RNNs as classifiers have recently shown improved performances over established methods in various sound recognition tasks. In this, it combines these two approaches in a Convolutional Recurrent Neural Network (CRNN) and applies them to a polyphonic sound event detection task. In this, it compares the performance of the presented CRNN method with CNN, RNN, and other established methods, and observes a considerable improvement for four different datasets consisting of everyday sound events.

Extracted conclusion

This paper provides the sole evidence of using the CRNN model for our and to be specific the recurrent structure for implementing in our project and to provide evidence that CRNN can efficiently be used for sound classifications.

P. von Platen et. al. [6] demonstrated that frames of traditional acoustic features, such as MFCC and FBANK, are usually derived using the short-time Fourier transform (STFT) based on a 25ms window, within which the speech signal is assumed to be stationary, and a window shift of 10ms. Conventional cross-entropy (CE) trained feed-forward DNN AMs have been found to yield in this when 11 concatenated frames (or 9 concatenated frames if first-order differentials are included) are used as the AM input 2 22, which results in an input span of 125ms of the raw waveform signal. It has been found that more power is useful for ANN AMs, such as recurrent or time-delayed neural networks. can effectively use a much longer span than DNNs 23.02. This shows the importance of input span for acoustic modeling Csv.

Extracted conclusion

From this research paper, it is concluded that the result of our model can be highly improved by the implementation of various data augmentation methods by adding padding and changing the gain of our spectrogram.

Roy Rudolf Huizen et. al. [7] mainly focuses on improving the accuracy of noise audio recordings. High-quality audio recording, extraction using the Mel frequency cepstral coefficients (MFCC) method produces high accuracy. While the low-quality is because of noise, the accuracy is low. Improved accuracy by investigating the effect of bandwidth on the Mel scale. The presented improvement uses the Mel scale separation methods into two frequency channels (MFCC dual channel). For the comparison method using the Mel scale bandwidth without separation (MFCC single-channel).

Extracted conclusion

This paper simply explained how MFCC features both for single-channel and for double-band channel separate different audios and how to extract them. For instance, in our case, we used the double-channel method for extracting the features as its results are much closer to real-life scenarios and provide more accuracy up to 97.5% when there is no noise and with noise up to 76.25%.

Quan Zhou et. al. [8] explained that with the outbreak of COVID-19, it is necessary to distinguish the sound of coughing, to estimate suspected patients in the population. The Mel-spectrogram is an effective tool to extract hidden features from audio and visualize them as an image. A CNN model can effectively extract features from images, and then complete tasks such as classification and recognition. Therefore, In this, it uses the CNN model to effectively classify the audio and to realize the accurate recognition and detection of coughing.

Extracted conclusion

This paper provides the soul working building model and evidence for us for using the CNN model for classifying cough into different domains based on Mel-spectrogram from the experiment conducted. In this, it can be said that recurrent layer activation may be informative on the degree of relevance of the temporal context information for various sound events.

Chapter 3

Implementation

This project is developed using machine learning to detect COVID-19 in earlier stages by utilizing convolutional neural networking as well as the very basic audio extraction features in the industry such as MFCCs to provide an asap diagnosis of the disease combining great accuracy. This project is implemented completely using python. The system will initially be fed with data from different sources i.e. available datasets on the internet. The data is then preprocessed before the further process is carried out.

3.1 Libraries and Algorithms

3.1.1 Csv

Csv stands for **comma-separated values**. It is not a file type, like you may be used to with a spreadsheet like Excel or Numbers. Csv, in fact, is a format. It refers to the way the data is structured (or formatted), not the type of file or program used to open it. This type of data structure (Csv) enables programs - like Excel, Numbers, cloud apps and databases and more - to read your data in an organized way. The single biggest use for Csv is to move data between two (or more!) places. This means importing and exporting. Csv format is the simplest and most universally accepted data format. As you recall from the previous post - Csv organizes your dataset with a simple structure of separating each value with a comma. No fancy formulas, no complex formatting, no proprietary programming language - just simple plain text and commas.

3.1.2 Cv2 (OpenCV)

OpenCV is a library of Python bindings designed to solve computer vision problems. In this, everything is returned as NumPy objects like *ndarray* and *native* Python objects like *lists*, *tuples*, *dictionary*, etc. So due to this NumPy support, you can do any *numpy* operation here. NumPy is a highly stable and fast array processing library. It is used to speed up the use of real-time machine recognition of images, objects, and video processing applications. There are major domains — image processing, video capture and analysis, face detection, and object detection — associated with computer vision, but it needs a cross-platform library to develop real-time applications. This is where OpenCV came in, which was originally developed in C++ and later followed by Java and Python. It runs on various platforms such as Windows, macOS, Android, iOS, and Linux. OpenCV is a perfect tool for computer vision, but system development without thinking of its broadest audience is still a huge problem among entrepreneurs.

3.1.3 Librosa

Librosa is a Python package **for music and audio analysis**. Librosa is used when we work with audio data like in music generation (using LSTMs), Automatic Speech Recognition. It provides the building blocks necessary to create the music information retrieval systems. The emerging research field of MUSIC INFORMATION RETRIEVAL(MIR) broadly covers topics at the intersection of musicology, digital signal processing, machine learning, information retrieval, and library science. Although the field is relatively young—the first INTERNATIONAL SYMPOSIUM ON MUSIC INFORMATION RETRIEVAL(ISMIR) 1 was held in October of 2000—it is rapidly developing, thanks in part to the proliferation and practical scientific needs of digital music services, such as iTunes, Pandora, and Spotify. While the preponderance of MIR research has been conducted with custom tools and scripts developed by researchers in a variety of languages such as MATLAB or C++, the stability, scalability, and ease of use of these tools has often left much to be desired.

3.1.4 Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labelled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real-world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open-source data analysis/manipulation tool available in any language**. It is already well on its way toward this goal.

Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in SQL table or Excel spreadsheet.
- Ordered and unordered (not necessarily fixed-frequency) time-series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels.
- Any other form of observational/statistical data sets. The data need not be labelled at all to be placed into a pandas data structure.

Pandas allows importing data from various file formats such as comma-separated-values, JSON, SQL, and Microsoft Excel. Pandas allow various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features.

3.1.5 Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the *ndarray* object. This encapsulates *n*-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

3.1.6 Matplotlib

Matplotlib is a cross-platform, **data visualization and graphical plotting library** for Python and its numerical extension NumPy. As such, it offers a viable open-source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

3.1.7 IPython

The goal of IPython is to create a comprehensive environment for interactive and exploratory computing. To support this goal, IPython has three main components:

- An enhanced interactive Python shell.
- A decoupled two-process communication model, which allows for multiple clients to connect to a computation kernel, most notably the web-based notebook
- An architecture for interactive parallel computing.

3.1.8 TensorFlow

TensorFlow is an open-source end-to-end platform for creating Machine Learning applications. It is a symbolic math library that uses dataflow and differentiable programming to perform various tasks focused on the training and inference of deep

neural networks. It allows developers to create machine learning applications using various tools, libraries, and community resources. Currently, the most famous deep learning library in the world is Google's TensorFlow. Google uses machine learning in all of its products to improve the search engine, translation, image captioning, or recommendations.

TensorFlow makes use of a graph framework. The graph gathers and describes all the series computations done during the training. The graph has lots of advantages:

- It was done to run on multiple CPUs or GPUs and even mobile operating system
- The portability of the graph allows it to preserve computations for immediate or later use. The graph can be saved to be executed in the future.
- All the computations in the graph are done by connecting tensors together
- A tensor has a node and an edge. The node carries the mathematical operation and produces an endpoint output. The edges explain the input/output relationships between nodes.

3.1.9 Keras

Keras runs on top of open-source machine libraries like TensorFlow, Theano, or Cognitive Toolkit (CNTK). Theano is a python library used for fast numerical computation tasks. TensorFlow is the most famous symbolic math library used for creating neural networks and deep learning models. TensorFlow is very flexible and the primary benefit is distributed computing. CNTK is a deep learning framework developed by Microsoft. It uses libraries such as Python, C#, C++, or standalone machine learning toolkits. Theano and TensorFlow are very powerful libraries but difficult to understand for creating neural networks. Keras is based on a minimal structure that provides a clean and easy way to create deep learning models based on TensorFlow or Theano. Keras is designed to quickly define deep learning models. Well, Keras is an optimal choice for deep learning applications.

Keras leverages various optimization techniques to make high-level neural network API easier and more performant. It supports the following features –

- Consistent, simple, and extensible API.
- Minimal structure - easy to achieve the result without any frills.
- It supports multiple platforms and backends.
- It is a user-friendly framework that runs on both CPU and GPU.
- Highly scalability of computation.

3.1.10 Layers

A layer is a callable object that takes as input one or more tensors and that outputs one or more tensors. It involves *computation*, defined in the `call()` method and

a *state* (weight variables), defined either in the constructor `__init__()` or in the `build()` method.

- Training (boolean, whether the call is in inference mode or training mode). See more details in the layer/model subclassing guide.
- Besides trainable weights, updated via back propagation during training, layers can also have non-trainable weights. These weights are meant to be updated manually during the call().
- Some losses (for instance, activity regularization losses) may be dependent on the inputs passed when calling a layer. Hence, when reusing the same layer on different inputs a and b, some entries in the layer. Losses may be dependent on a and some on b. This method automatically keeps track of dependencies.

3.1.11 Utils

Python Utils is a collection of small Python functions and classes that make common patterns shorter and easier. It is by no means a complete collection but it has served me quite a bit in the past and I will keep extending it. This module makes it easy to execute common tasks in Python scripts such as converting text to numbers and making sure a string is in Unicode or bytes format.

The project starts with analyzing the audio signal datasets using MFCCs and provides the related spectrogram as required. A block diagram of the same is given in **Fig. 3.1**

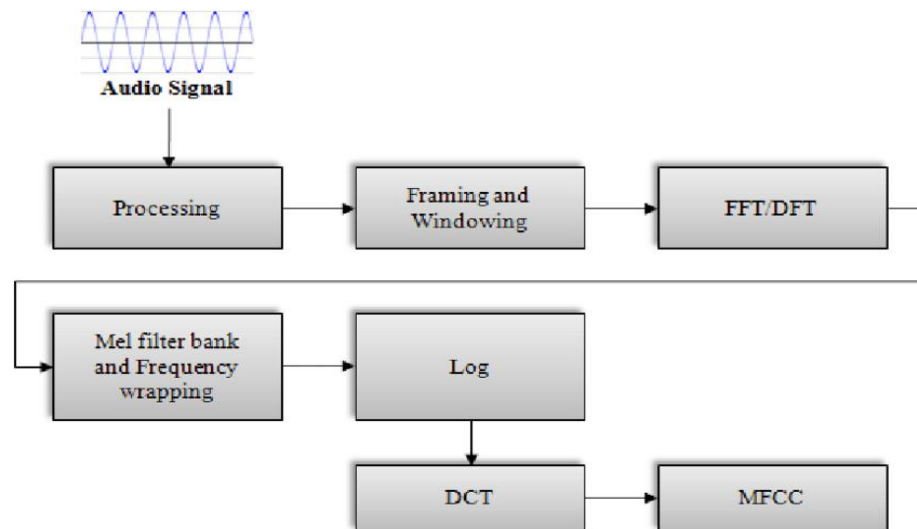


Fig. 3.1 Block Diagram of Audio Processing

To create an acoustic model, observation X is represented by a sequence of acoustic feature vectors (x_1, x_2, x_3, \dots) . Pitch varies with people. However, this has little role in recognizing what he/she said. F0 is related to the pitch. It provides no value in speech recognition and should be removed. What is more important is the formants F1, F2, F3, ..., FN. The extracted features will be robust to who the speaker is, and the noise in the environment. Also, like any ML problem, we want to extract features to be independent of others. It is easier to develop models and to train these models with independent features.

Any sound generated by humans is determined by the shape of their vocal tract (including the tongue, teeth, etc.). If this shape can be determined correctly, any sound produced can be accurately represented. The envelope of the temporal power spectrum of the speech signal is representative of the vocal tract and MFCC (which is nothing but the coefficients that make up the Mel-frequency cepstrum) accurately represents this envelope.

3.2 Pre-emphasis

Pre-emphasis boosts the amount of energy in the high frequencies. For voiced segments like vowels, there is more energy at the lower frequencies than at the higher frequencies. This is called spectral tilt which is related to the glottal source (how vocal folds produce sound). Boosting the high-frequency energy makes the information in higher formants more available to the acoustic model. This improves phone detection accuracy. Humans start having hearing problems when they cannot hear these high-frequency sounds. Also, noise has a high frequency. In the engineering field, pre-emphasis is used to make the system less susceptible to noise introduced in the process later. For some applications just undoing the boosting at the end is required.

3.2.1 Windowing

The MFCC technique aims to develop features from the audio signal which can be used for detecting coughs in speech. It's necessary to break the audio signal into different segments with each segment having 25ms width and with the signal at 10ms apart. Moreover, while breaking the signal, if we directly chop it off at the edges of the signal, the sudden fall in amplitude at the edges will produce noise in the high-frequency domain. So instead of a rectangular window, we will use Hamming/Hanning windows to chop the signal which won't produce noise in the high-frequency region.

3.2.2 DFT (Discrete Fourier Transform)

The program will convert the signal from the time domain to the frequency domain by applying the DFT transform. For audio signals, analyzing in the frequency domain is easier than in the time domain.

3.2.3 Mel filterbank

The way human ears will perceive the sound is different from how machines will perceive the sound. Our ears have higher resolution at a lower frequency than at a higher frequency. So, if humans hear sound at 200 Hz and 300 Hz, they can differentiate it easily when compared to the sounds at 1500 Hz and 1600 Hz even though both had a difference of 100 Hz between them. While for the machine the resolution is the same at all frequencies. It is noticed that modelling the human hearing property at the feature extraction stage will improve the performance of the model. Thus, the system will use the Mel scale to map the actual frequency to the frequency that human beings will perceive.

The formula for the mapping is given below:

$$M_F(x) = 2595 \times \log_{10} (1 + (x/100)) \quad [\text{eq. 3.1}]$$

Humans are less sensitive to changes in audio signal energy at higher energy compared to lower energy. Log function also has a similar property. At a low value of input x gradient of the log, the function will be higher but at high values of the input, the gradient value is less. So, the log is applied to the output of Mel-filter to mimic the human hearing system.

3.2.4 DCT (Discrete Cosine Transform)

A **discrete cosine transform (DCT)** expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. The DCT, first presented by Nasir Ahmed in 1972, is a widely used transformation technique in signal processing and data compression. It is used in most digital media, including digital images. Uncompressed digital media, as well as lossless compression, had impractically high memory and bandwidth requirements, which were significantly reduced by the highly efficient DCT lossy compression technique. In the final step, the DCT (Discrete Cosine Transformation) for the energy of the log filter bank is measured. Since the energies of the filter banks are connected with each other, and the presented system filter banks all overlap with one another. The result is known as IMFCC (Improved Mel-frequency Cepstral Coefficient) after DCT.

The formula for DCT is given below:

$$C_n = \sqrt{\frac{2}{S}} \sum_{k=0}^{S-1} \left((\log_{10}[c \times (k + 1)]) U \sin\left(\frac{\pi}{2}\right) - \left[n \times \left(\frac{2k-1}{2}\right) \dot{U} \frac{\pi}{S} \right] \right) \quad [\text{eq. 3.2}]$$

3.2.5 Convolutional Neural Network (ConvNet or CNN):

It is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets can learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields over-lap to cover the entire visual area. The particular type of ConvNet that was used is a combination of convolutional neural networking (CNN) and recurrent neural networking (RNN) and is known as CRNN or Convolutional Recurrent Neural Networking.

3.3 Steps and Procedure

Step1: Loading all the libraries necessary for preprocessing, audio processing, data augmentation.

```
# feature extracting and preprocessing data
%matplotlib inline
import csv
import cv2
import librosa
import librosa.display
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
from matplotlib import pyplot
import IPython.display as ipd
#Keras and Tensorflow
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.utils import np_utils
# Preprocessing sklearn
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
import warnings
warnings.filterwarnings('ignore')
```

Python

Step 2: Load features 2 of our audio datasets for preprocessing into variables.

DATASET

```

metadata_train_challenge = pd.read_csv('../final_minor_cov19/archive (1)/aicv115m_public_train/aicv
print('len metadata_train_challenge',len(metadata_train_challenge.iloc[:,-1]))
metadata_train_challenge.head(2)

```

[4] ✓ 0.1s Python

... len metadata_train_challenge 1199

	uuid	subject_gender	subject_age	assessment_result	file_path
0	3284bcf1-2446-4f3a-ac66-14c76b294177	male	23.0	0	3284bcf1-2446-4f3a-ac66-14c76b294177.wav
1	431334e1-5946-4576-bb51-8e342ccc22b4	NaN	NaN	0	431334e1-5946-4576-bb51-8e342ccc22b4.wav

```

cough_trial_extended = pd.read_csv('../final_minor_cov19/archive (2)/cough_trial_extended.csv')
print('len cough_trial_extended',len(cough_trial_extended.iloc[:,-1]))
cough_trial_extended.head(4)

```

[5] ✓ 0.7s Python

... len cough_trial_extended 170

	file_properties	class
0	0v8MGxNetjg_10.000_20.000.wav	not_covid
1	1j1duoxdxBg_70.000_80.000.wav	not_covid
2	1MSYO4wgiag_120.000_130.000.wav	not_covid
3	1PajbAKd8Kg_0.000_10.000.wav	not_covid

Step 3: For further processing of the data sets we would like to combine the two datasets into one. And label the non-covid patients as 0 and patients affected as 1.

(a) LABELING

```

header = 'filePath label'
header = header.split()

file = open('data_file_Path.csv', 'w')
with file:
    writer = csv.writer(file)
    writer.writerow(header)
data = []
for i,label in enumerate(cough_trial_extended['class']):
    if label == 'not_covid':
        label = '0'
    else:
        label = '1'
    filename = cough_trial_extended.iloc[i,0]
    filePath = '../final_minor_cov19/archive (2)/trial_covid/' + str(filename)
    data = [filePath, label]
    file = open('data_file_Path.csv', 'a')
    with file:
        writer = csv.writer(file)
        writer.writerow(data)

```

Python

(b) COMBINING

```
data = []
for i,label in enumerate(metadata_train_challenge['assessment_result']):
    filename = metadata_train_challenge.iloc[i, -1]
    filePath = '../final_minor_cov19/archive (1)/aicv115m_public_train/aicv115m_public_train/train_
    data = [filePath, label]
    file = open('data_file_Path.csv', 'a')
    with file:
        writer = csv.writer(file)
        writer.writerow(data)
data = pd.read_csv('./data_file_Path.csv')
print ('len data', len(data.iloc[:,1]))
data.head(5)
```

✓ 1.5s Python

Step 4:

Preprocessing

We will extract 2D features like:

- * Mel-frequency Spectrogram
- * Chroma

And then combine them into images to feed into the model we also did add 1D features and mean them through time:

- * MFCCs
- * Spectral Centroid
- * Spectral Bandwidth
- * Spectral Roll-off
- * ZCR + energy

```
Features = []
Number = len(data.iloc[:,1]) #Number of files we want to try on Number = len(data.iloc[:,1]) for all data
for file in data['filePath'][:Number]:
    y,sr=librosa.load(file)
    if librosa.get_duration(y=y, sr=sr) > 30:
        y,sr=librosa.load(file, duration = 30)
    mel_spec = librosa.feature.melspectrogram(y=y, sr=sr, S=None, n_fft=2048, hop_length=512, win_length=None, window='hann',
        center=True, pad_mode='reflect', power=2.0)
    chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr, S=None, norm=None, n_fft=2048,
        hop_length=512, win_length=None, window='hann',
        center=True, pad_mode='reflect', tuning=None, n_chroma=12)
    MFCC = librosa.feature.mfcc(y=y, sr=sr, S=None, n_mfcc=20, dct_type=2, norm='ortho', lifter=0)
    ZCR = librosa.feature.zero_crossing_rate(y, frame_length=2048, hop_length=512, center=True)
    spectral_centroid = librosa.feature.spectral_centroid(y=y, sr=sr, S=None, n_fft=2048,
        hop_length=512, freq=None, win_length=None, window='hann',
        center=True, pad_mode='reflect')
    spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr, S=None, n_fft=2048,
        hop_length=512, win_length=None, window='hann',
        center=True, pad_mode='reflect', freq=None, centroid=None, norm=True, p=2)
    spectral_rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr, S=None, n_fft=2048,
        hop_length=512, win_length=None, window='hann',
        center=True, pad_mode='reflect', freq=None, roll_percent=0.85)
    feature = np.concatenate((mel_spec,chroma_stft,MFCC,ZCR,spectral_centroid,spectral_bandwidth,spectral_rolloff), axis=0)
    feature = librosa.power_to_db(feature, ref=np.max)
    Features.append(feature)
print ('Features',len(Features) )
```

Step 5: Visualizing the data to feed into our CRNN model (this part involves concatenating our features and conversion of the images to dB value for better results).

Some information needs to be used on the time axis to work well like Mel-spectrogram, MFCC, or ZCR/energy. In fact, they usually combine ZCR/Energy to distinguish when it

is sound and when it is not. When there is sound ZCR will suddenly decrease and energy suddenly rise at the same time.

```
plt.figure(num='Features',figsize=(9,9))
for i, img in enumerate(Features[:3]):
    plt.subplot(3,1,i+1)
    plt.title('Feature {}'.format(i))
    S_db = librosa.power_to_db(img, ref=np.max)
    img = librosa.display.specshow(S_db, x_axis='time', y_axis='mel', sr=sr, fmax=8000)
    plt.imshow(img)
    plt.colorbar(format='%+2.0f dB')
    plt.tight_layout(pad=3.0)
    #plt.axis('off')
plt.show()
```

Step 6: Saving our features against the suitably labelled row.

```
header = 'features label'
header = header.split()
file = open('raw_features.csv', 'w')
with file:
    writer = csv.writer(file)
    writer.writerow(header)
value = []
for i, feature in enumerate(Features):
    feature = feature.flatten()
    feature = ( " ".join(str(e) for e in feature))
    label = data['label'][i]
    value = [feature, label]
    file = open('raw_features.csv', 'a')
    with file:
        writer = csv.writer(file)
        writer.writerow(value)
raw_feature = pd.read_csv('./raw_features.csv')
raw_feature.head(2)
```

Step 7: This step involves preparing our data to feed in our model as the spectrograms are of variable lengths i.e. of 20 sec, 30 sec, or 1 min long so we'll cut short the longer audios to not make much of a difference.

```
Time = np.array([x.shape[1] for x in Features])
unique, counts = np.unique(Time, return_counts=True)
#print(dict(zip(unique, counts)))
max_length = np.max(Time)
print('max_length', max_length)

width = 3 # the width of the bars
plt.figure(num='Time',figsize=(9,9))
plt.xlabel('Duration')
plt.ylabel('Number of files')

plt.title('time duration')
plt.bar(unique, counts, width, ec='blue')
plt.show()
```

We'll pad or specific to say we'll apply sorted padding with a batch of each file concerning the frame of the longest audio file.

Function definition:

```
def preprocess(feature, featureSize):  
    widthTarget, heightTarget = featureSize  
    height, width = feature.shape  
  
    # scale according to factor  
    newSize = (int(width / 4),41)  
    #print ('newSize = {}, old size = {}'.format(newSize, feature.shape ))  
    feature = cv2.resize(feature, newSize)  
    # Normalization  
    feature = scaler.fit_transform(feature)  
    feature = np.pad(feature, ((0, 0), (0, widthTarget - feature.shape[1])), 'constant')  
    #transpose  
    feature = np.transpose(feature)  
  
    return feature
```

Python

Function calling:

```
scaler = StandardScaler()  
#print ('raw Features', Features[1])  
scale_features = []  
for feature in Features[:Number]:  
    feature = preprocess(feature,featureSize = (int(max_length/4),41)) #41 = 164/4  
    scale_features.append(feature)  
  
#print ('scale feature',scale_features[1])  
plt.figure(num='Features transpose',figsize=(9,9))  
  
for i, img in enumerate(scale_features[:6]):  
    plt.subplot(3,3,i+1)  
    plt.tight_layout(pad=3.0)  
    plt.title('Feature {}'.format(i))  
    plt.imshow(img)  
    #plt.axis('off')  
plt.show()
```

Python

Step 8: Now we'll divide our data into negatives and positives (currently we are limited to an imbalanced dataset with positives of 481 i.e. 35.14% of total and negatives of 888).

```
genre_list = data.iloc[:Number, -1]  
#print ('genre_list\n',genre_list)  
encoder = LabelEncoder()  
y = encoder.fit_transform(genre_list)  
neg, pos = np.bincount(y)  
total = neg + pos  
print ('positive: {} ({:.2f}% of total) \nnegative cases: {}'.format(pos, 100 * pos/total ,neg))
```

Python

Step 9: Dividing data into TRAINING, VALIDATION and TEST set.

```

scale_features = np.array(scale_features).reshape(-1, int(max_length/4), 41, 1)
indices = range(len(scale_features))

x_train, x_test, y_train, y_test, indices_train, indices_test = train_test_split(scale_features, y,
                                                                                shuffle = True,
                                                                                random_state = None)
X_train, X_valid, Y_train, Y_valid, Indices_train, Indices_valid = train_test_split(x_train, y_train,
                                                                                test_size=0.2,
                                                                                random_state =

Y_train = np_utils.to_categorical(Y_train, 2)
Y_valid = np_utils.to_categorical(Y_valid, 2)
print (len(Y_valid))

233

print ('\nlen(X_train)', len(X_train))
print ('len(X_valid)', len(X_valid))
print ('\n X_train.shape', X_train.shape)
print ('\n X_valid.shape', X_valid.shape)

```

Step 10: This step is the heart of our model i.e the CRNN model itself which includes recurrently applying convolutional blocks to our data, pooling and classifying by flattening the output.

```

def build_model(img_width = int(max_length/4), img_height = 41):
    # Inputs to the model

    input_img = layers.Input(shape=(img_width, img_height, 1), name="image", dtype="float32")

    # First conv block
    x = layers.Conv2D(64, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same", name="Conv1")(input_img)
    x = layers.MaxPooling2D((2, 2), strides = 2, name="pool1")(x)

    # Second conv block
    x = layers.Conv2D(128, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same", name="Conv2")(x)
    x = layers.MaxPooling2D((2, 2), strides = 2, name="pool2")(x)

    # Third conv block
    x = layers.Conv2D(256, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same", name="Conv3")(x)

    # Fourth conv block
    x = layers.Conv2D(256, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same", name="Conv4")(x)
    x = layers.MaxPooling2D((1, 2), name="pool4")(x)

    # Fifth conv block
    x = layers.Conv2D(512, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same", name="Conv5")(x)
    x = layers.BatchNormalization(momentum = 0.8, name="BatchNormalization_1")(x)

    # Sixth conv block
    x = layers.Conv2D(512, (3, 3), activation="relu", kernel_initializer="he_normal", padding="same", name="Conv6")(x)
    x = layers.BatchNormalization(momentum = 0.8, name="BatchNormalization_2")(x)
    x = layers.MaxPooling2D((1, 2), name="pool6")(x)

    # Seventh conv block
    x = layers.Conv2D(512, (2, 2), activation="relu", kernel_initializer="he_normal", padding="valid", name="Conv7")(x)

```

```

# The number of filters in the last layer is 512. Reshape accordingly before
# passing the output to the RNN part of the model
new_shape = (int(max_length/16)-1,512)

x = layers.Reshape(target_shape=new_shape, name="reshape")(x)

def attention_rnn(inputs):
    input_dim = int(inputs.shape[2])
    timestep = int(inputs.shape[1])
    a = layers.Permute((2, 1))(inputs) #Permutates the dimensions of the input according to a given pattern.
    a = layers.Dense(timestep, activation='softmax')(a)
    a = layers.Lambda(Lambda x: keras.backend.mean(x, axis=1), name='dim_reduction')(a)
    a = layers.RepeatVector(input_dim)(a)
    a_probs = layers.Permute((2, 1), name='attention_vec')(a)
    output_attention_mul = layers.multiply([inputs, a_probs], name='attention_mul')
    return output_attention_mul

x = attention_rnn(x)

# RNNs
x = layers.Bidirectional(layers.LSTM(128, return_sequences=True, dropout=0.25))(x)
x = layers.Bidirectional(layers.LSTM(128, return_sequences=True, dropout=0.25))(x)
x = layers.Flatten()(x)

# Output layer
x = layers.BatchNormalization(momentum = 0.8)(x)
x = layers.Dense(512, activation="relu")(x)
x = layers.Dense(256, activation="relu")(x)

y_pred = layers.Dense(2, activation="softmax", name="last_dense")(x) # y pred
model = keras.models.Model(inputs=input_img, outputs=y_pred, name="model")

return model
model = build_model()
model.summary()

```

Step 11: Training i.e. calling back for each of our invalid data (where the noise is more or muted).

```

epochs = 50
batch_size = 32
early_stopping_patience = 10

def scheduler(epoch):
    if epoch <= 10:
        return 1e-3
    elif 10 < epoch <= 15:
        return 1e-4
    else:
        return 1e-5

# Add early stopping
my_callbacks = [
    tf.keras.callbacks.LearningRateScheduler(scheduler),
    tf.keras.callbacks.ModelCheckpoint(filepath='./covid_model/covid_model_{epoch:02d}.h5',
                                      save_freq='epoch',
                                      monitor='val_loss',
                                      mode='min',
                                      save_best_only=True,
                                      period = 5),
    tf.keras.callbacks.EarlyStopping(
        monitor="val_loss", patience=early_stopping_patience, restore_best_weights=True
    )
]

model.compile(optimizer=keras.optimizers.Adam(), loss='categorical_crossentropy')

history = model.fit(x=X_train, y=Y_train,
                    validation_data=(X_valid, Y_valid),
                    epochs = epochs,
                    batch_size = batch_size,
                    callbacks = my_callbacks,
                    )

# list all data in history
print(history.history.keys())
# summarize history for loss
fig, ax = plt.subplots()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.savefig('./covid_model/covid_model_loss.png')
plt.show()

```


Step 12: Now comes the testing and loading of the model with the best weights (since this is based on training history) as the previous model showed poor output so we changed all the model weights manually.

```
load_model = 0
load_model = build_model()
load_model.load_weights('./covid_model/covid_model_10.h5')
```

```
predictions = []
predictions = model.predict(x_test)
y_predict = []
for i in range(len(predictions)):
    predict = np.argmax(predictions[i])
    y_predict.append(predict)
```

```
import seaborn as sns
def evaluate_matrix(y_test, y_predict, name):
    cm = confusion_matrix(y_test, y_predict)
    cm_df = pd.DataFrame(cm, index=["Negative", "Positive"], columns=["Negative", "Positive"])

    plt.figure(figsize=(10, 10))

    sns.set(font_scale=1)

    ax = sns.heatmap(cm_df, annot=True, square=True, fmt='d', linewidths=.2, cbar=0, cmap=plt.cm.Bl)
    ax.set_xticklabels(ax.get_xticklabels(), rotation=0)

    plt.ylabel("True labels", font_size = 'x-large')
    plt.xlabel("Predicted labels", font_size = 'x-large')
    plt.tight_layout()
    plt.title(name, font_size = 'xx-large', pad = 20)

    plt.show()

    print(classification_report(y_test, y_predict, target_names=["Negative", "Positive"]))
#evaluate_matrix(y_test, y_predict, 'Evaluate_matrix on original data')
```

Python

```
def ROC_curve(y_test, predictions, name):
    # calculate roc curves
    lr_fpr, lr_tpr, _ = roc_curve(y_test, predictions[:,1])
    print ('AUC = {:.3f}'.format( auc(lr_fpr, lr_tpr)))
    # plot the roc curve for the model
    lw = 2
    plt.plot(lr_fpr, lr_tpr, color="darkorange",
             lw=lw, label="ROC curve (area = %.2f)" % auc(lr_fpr, lr_tpr))
    plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
    plt.xlim([-0.02, 1.0])
    plt.ylim([0.0, 1.05])
    # axis labels
    pyplot.xlabel('False Positive Rate', font_size = 'x-large')
    pyplot.ylabel('True Positive Rate', font_size = 'x-large')
    plt.title(name, font_size = 'xx-large', pad = 20)
    # show the legend
    pyplot.legend()
    # show the plot
    pyplot.show()

#ROC_curve(y_test, predictions, 'AUC on original data')
```

Python

Step 13: So, the result above is quite bad and it can't be used in real life. we guess that Data Augmentation would help improve the outcome -

Methods:

- Time Shift
- Adding background noise
- Stretching the sound (just a little bit)
- Changing Gain

we try not to generate fake sounds that rarely happen in real life, and distort the sound so much. So, we don't recommend :

- Time stretch (too much)

Note: You have to use data augmentation on just the train and valid set

```
pos_indices = []
Aug_feature = []
for i,value in enumerate(y_train):
    if value == 1:
        pos_indices.append(indices_train[i])
y, sr = librosa.load(data['filePath'][pos_indices[1]])
def white_noise(y):
    wn = np.random.randn(len(y))
    y_wn = y + random.uniform(0, 0.005)*wn
    return y_wn
def time_shift(y):
    y = np.roll(y, random.randint(-10000,10000))
    return y
def Gain(y):
    y = y + random.uniform(-0.2,0.2)*y
    return y
def stretch(y, rate=random.uniform(0.8,1.2)):
    y = librosa.effects.time_stretch(y, rate)
    return y
```

Step 14:

Data balancing:

Due to the dataset imbalanced problems as the result above

- positive: 481 (35.14% of total)
- negative cases: 888 we need to generate around 200 to 250 positive cases more

```
num_aug = (neg - pos)/1.5
if num_aug > len(pos_indices):
    iteration = int(num_aug/len(pos_indices))
else:
    iteration = 1
print ('Number of file generated based on one positive case',iteration)
```

```

for file in data['filePath'][pos_indices]:
    if librosa.get_duration(y=y, sr=sr) > 30:
        y,sr=librosa.load(file, duration = 30)
        for i in range(iteration):
            y,sr=librosa.load(file)

            chance = random.randint(0,100)
            if chance <=20:
                stretch(y)
            if chance <=40:
                time_shift(y)
            if chance <=60:
                Gain(y)
            if chance <=80:
                white_noise(y)

            ZCR = librosa.feature.zero_crossing_rate(y, frame_length=2048, hop_length=512, center=True)
            if ZCR.shape[1] >= max_length:
                continue
            mel_spec = librosa.feature.melspectrogram(y=y, sr=sr, S=None, n_fft=2048,
                                                    hop_length=512, win_length=None, window='hann',
                                                    center=True, pad_mode='reflect', power=2.0)
            chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr, S=None, nrm=None, n_fft=2048,
                                                    hop_length=512, win_length=None, window='hann',
                                                    center=True, pad_mode='reflect', tuning=None, n_chroma=12)
            MFCC = librosa.feature.mfcc(y=y, sr=sr, S=None, n_mfcc=20, dct_type=2, norm='ortho', lifter=0)

            spectral_centroid = librosa.feature.spectral_centroid(y=y, sr=sr, S=None, n_fft=2048,
                                                                hop_length=512, freq=None, win_length=None, window='hann',
                                                                center=True, pad_mode='reflect')
            spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr, S=None, n_fft=2048,
                                                                hop_length=512, win_length=None, window='hann',
                                                                center=True, pad_mode='reflect', freq=None, centroid=None, norm=True, p=2)
            spectral_rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr, S=None, n_fft=2048,
                                                                hop_length=512, win_length=None, window='hann',
                                                                center=True, pad_mode='reflect', freq=None, roll_percent=0.85)
            aug_feature = np.concatenate((mel_spec, chroma_stft, MFCC, ZCR, spectral_centroid, spectral_bandwidth, spectral_rolloff),
                                         axis=0)
            aug_feature = librosa.power_to_db(aug_feature, ref=np.max)
            Aug_feature.append(aug_feature)
            #print ('Aug_feature', Aug_feature.shape)
            print ('Aug_feature', len(Aug_feature) )

```

Step 15: Now combining and splitting the data again into training and testing set for again putting through our librosa model and testing again.

```

scale_aug_features = np.array(scale_aug_features).reshape(-1, int(max_length/4), 41, 1)
x_train = np.concatenate((x_train, scale_aug_features), axis=0)
print (len(x_train))

```

Python

1572

```

y_train = np.concatenate((y_train, y_aug), axis=0)

```

Python

```

X_train, X_valid, Y_train, Y_valid = train_test_split(x_train, y_train, test_size=0.2, shuffle = True,
                                                    random_state = None, stratify = y_train )

Y_train = np_utils.to_categorical(Y_train, 2)
Y_valid = np_utils.to_categorical(Y_valid, 2)
#print (Y_valid)
print (len(Y_valid))
print('X_train.shape', X_train.shape)
print('X_valid.shape', X_valid.shape)

```

Python

315

```

X_train.shape (1257, 323, 41, 1)
X_valid.shape (315, 323, 41, 1)

```

```

my_callbacks = [
    tf.keras.callbacks.LearningRateScheduler(scheduler),
    tf.keras.callbacks.ModelCheckpoint(filepath='./covid_model/new_covid_model_{epoch:02d}.h5',
                                       save_freq='epoch',
                                       monitor='val_loss',
                                       mode='min',
                                       save_best_only=True,
                                       period = 5),
    tf.keras.callbacks.EarlyStopping(
        monitor="val_loss", patience=early_stopping_patience, restore_best_weights=True
    )
]

new_model.compile(optimizer=keras.optimizers.Adam(), loss='categorical_crossentropy')

new_history = new_model.fit(x= X_train, y= Y_train,
                           validation_data=(X_valid, Y_valid),
                           epochs = epochs,
                           batch_size = batch_size,
                           callbacks = my_callbacks,
                           )

```

Step 16: Now Finally we are somewhat satisfied with our result and have obtained our final model so we will save our result and model.

```

new_load_model = 0
new_load_model = build_model()
new_load_model.load_weights('./covid_model/new_covid_model_10.h5')

new_predictions = []
new_predictions = new_load_model.predict(x_test)
y_new_predict = []
for i in range(len(new_predictions)):
    predict = np.argmax(new_predictions[i])
    y_new_predict.append(predict)

```

Result based on our original data:

```

evaluate_matrix(y_test, y_predict, 'Evaluate_matrix on Original data')
ROC_curve(y_test, predictions, 'AUC on Original data')

```

Result based on our augmented data:

```

evaluate_matrix(y_test, y_new_predict, 'Evaluate_matrix on Augmented data')
ROC_curve(y_test, new_predictions, 'AUC on Augmented data')

```


3.4. Concluding Remarks

In this chapter, various libraries and algorithms have been discussed. Algorithms and mathematical terms like MFCCs, Convolutional Neural Networking, Image processing, Audio Processing, etc. have been discussed. The main advantage of using machine learning is that, once an algorithm learns what to do with data, it can do its work automatically. Machine learning is used to teach machines how to handle the data more efficiently. Sometimes after viewing the data, we cannot interpret the pattern or extract information from the data. In that case, we apply machine learning. With the abundance of datasets available, the demand for machine learning is in rise. And with machine learning, there's no need to worry about the precision or accuracy of the whole system as the larger the database, the more precise the model will become. The model develops and evolves itself with each step. Today each and every person is using machine learning knowingly or unknowingly. From getting a recommended product in online shopping to updating photos in social networking sites, machine learning is everywhere around us, making our lives much smarter and easier than ever.

Chapter 4

Results and Verification

4.1 Model Loss

Our model has some losses in its test data due to the quality of recorded audio as it assumes to receive cough sounds for a time window of 5 seconds and any disturbance and quiet time hinders its data between the valid dataset one and the trainable dataset as shown in Fig. 4.1.

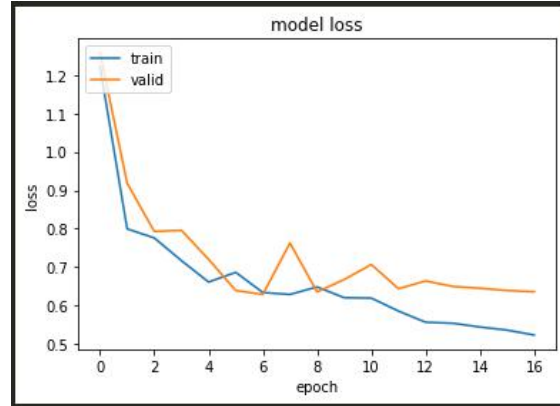


Fig. 4.1 Model Loss Graph

4.2 Predictions based on original data

Predictions based on our original data are coming out to be 64% in case of predicting negative cases and 22% in case of predicting positive test cases as shown in Fig. 4.2 and Fig 4.3 the overall accuracy with F1-score is calculated to be 60%.

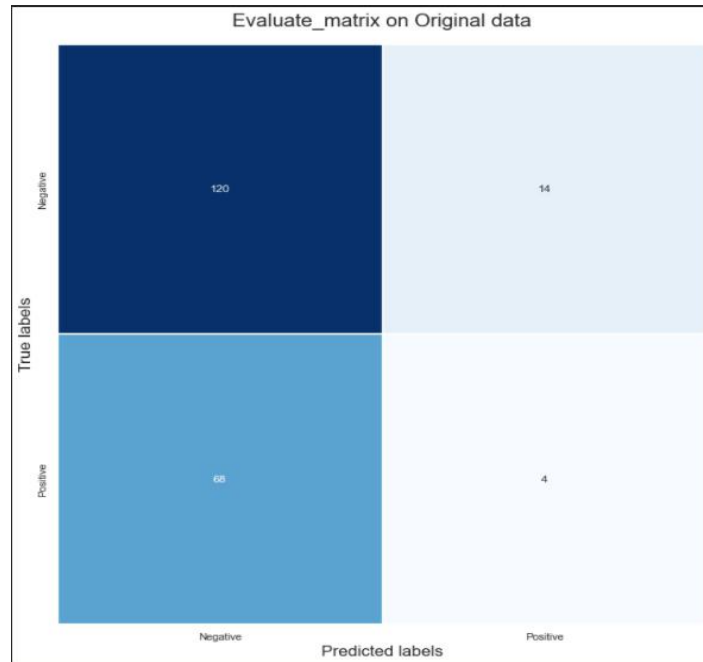


Fig. 4.2 Matrix representing Original Data

	precision	recall	f1-score	support
Negative	0.64	0.90	0.75	134
Positive	0.22	0.06	0.09	72
accuracy			0.60	206
macro avg	0.43	0.48	0.42	206
weighted avg	0.49	0.60	0.52	206
AUC = 0.544				

Fig. 4.3 System results on Original Data

The AUC(area under the curve)/ROC(region of convergence) between FPR(false positive rate) and TPR(true positive rate) to be 54.4% as shown in **Fig. 4.4**. Also shown in **Table 4.1** .

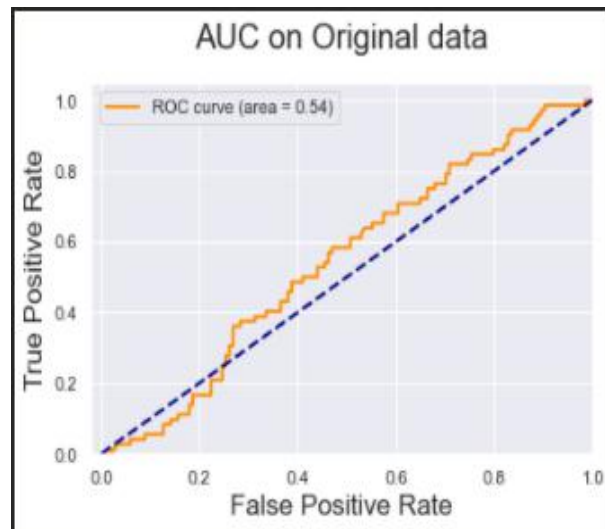


Fig. 4.4 TPR vs FPR (Original Data)

Table 4.1 Below table shows overall accuracy obtained so far based on original dataset

Header	Precision	Recall	F1-score	Support
Negative	64%	90%	75%	134
Positive	22%	6%	9%	72
Accuracy			60%	206
Macro Avg	43%	48%	42%	206
Weighted Avg	49%	60%	52%	206
Area under the curve: 54.4%				

4.3 Predictions based on augmented data

After using Data Augmentation, Our model's predictions obtained were undeniably better than previous predictions.

The accuracy lowered down by 3% at the cost of precision but it's not a matter of concern because topics like this usually get evaluated or many times based on other physically visible symptoms too. The positive metrics have increased so that is a good sign as shown in **Fig. 4.5** and **Fig. 4.6**.

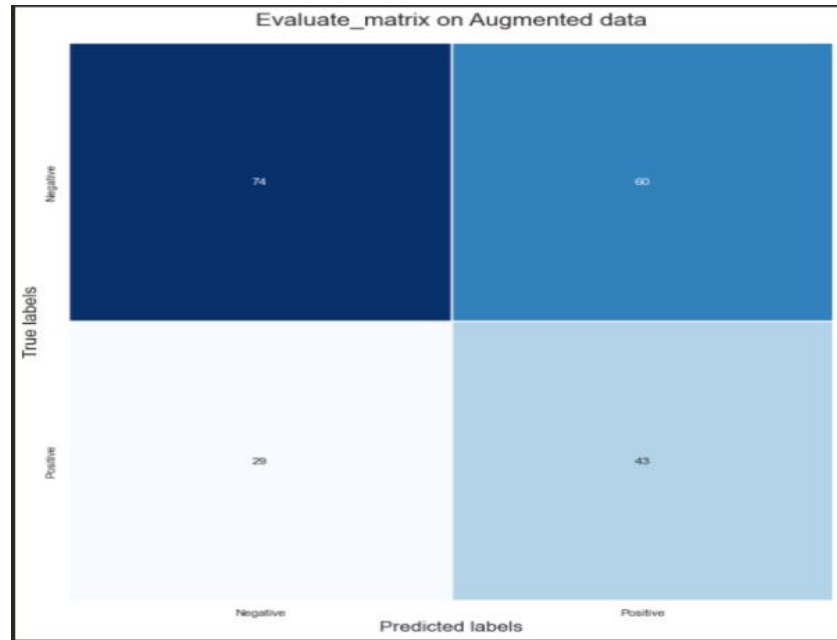


Fig. 4.5 Matrix representing Augmented Data

	precision	recall	f1-score	support
Negative	0.72	0.55	0.62	134
Positive	0.42	0.60	0.49	72
accuracy			0.57	206
macro avg	0.57	0.57	0.56	206
weighted avg	0.61	0.57	0.58	206
AUC =	0.623			

Fig. 4.6 System Results on Augmented Data

Predictions based on our augmented data are coming out to be 72% in case of predicting negative cases and 42% in case of predicting positive test cases as shown in **Fig 4.5** and **Fig 4.6** an overall accuracy with F1-score is calculated as 57%. The AUC (area under the curve)/ROC (region of convergence) between FPR (false positive rate) and TPR (true positive rate) to be 62.3% as shown in **Fig. 4.7**.

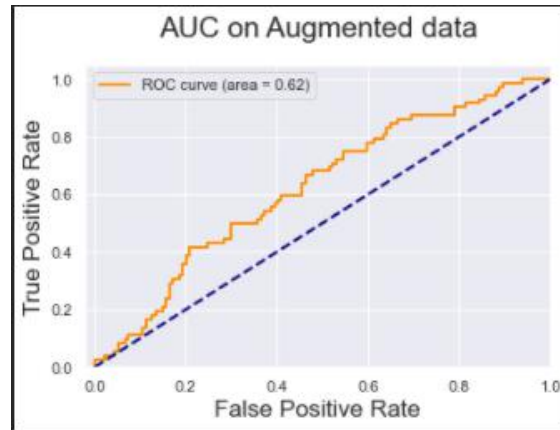


Fig. 4.7 TPR vs FPR (Augmented Data)

Table 4.2 Below table shows the overall achieved accuracy when the data is augmented

Header	Precision	Recall	F1-score	Support
Negative	72%	55%	62%	134
Positive	42%	60%	49%	72
Accuracy			57%	206
Macro Avg	57%	57%	56%	206
Weighted Avg	61%	57%	58%	206
Area under the curve: 62.3%				

Results obtained in **Table 4.2** clearly indicates that there is future scope for this model for diagnosing as with a much better dataset or more data better results can be achieved.

4.4 Web portal

Fig. 4.8 , **Fig 4.9** and **Fig 4.10**, Shows the front end part of the web portal, including our home page, our diagnostic form, and some gist about our methods.

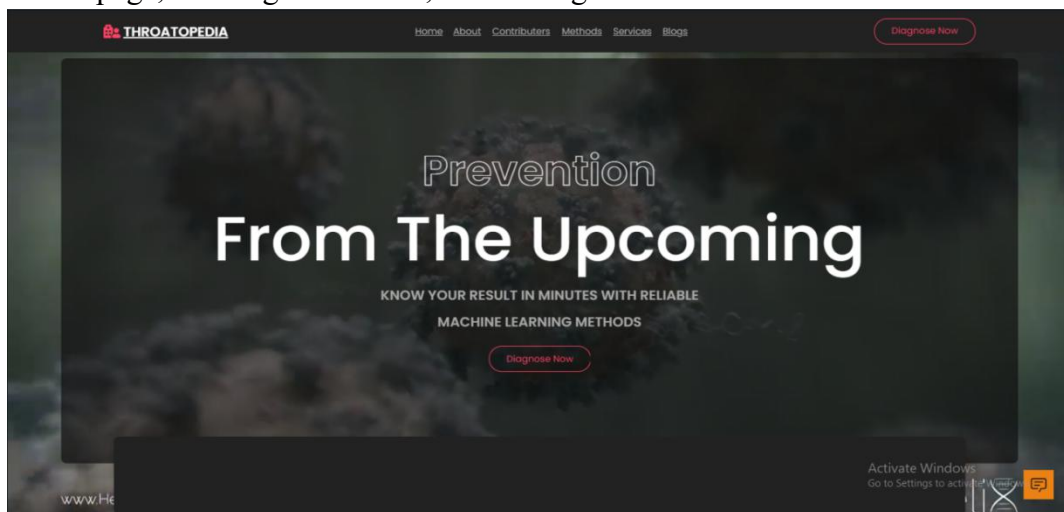


Fig.4.8 A representation of homepage of our Web Portal

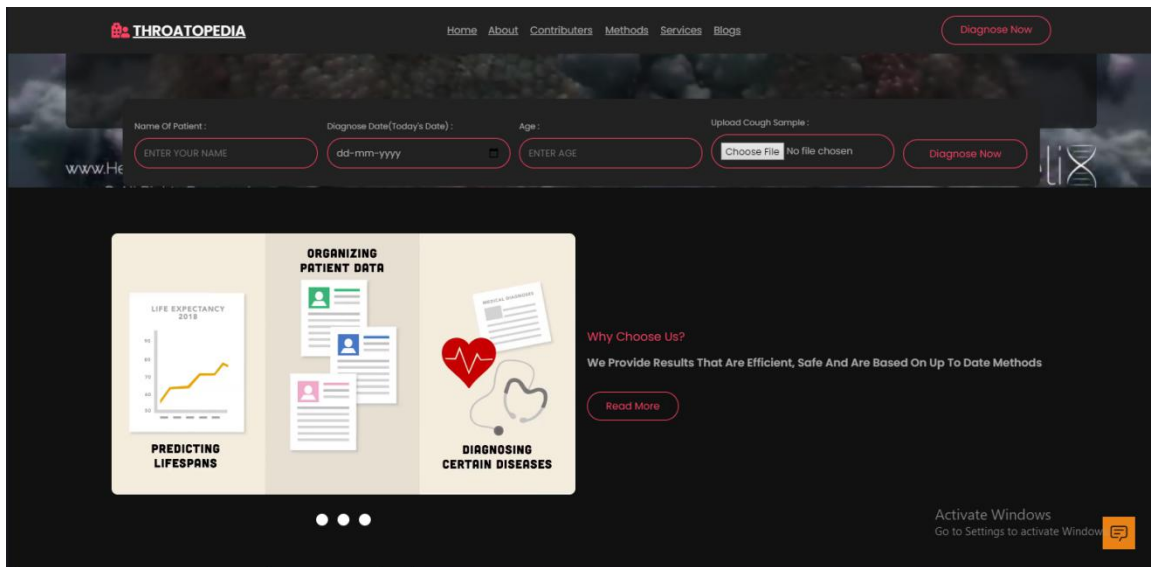


Fig. 4.9 Show the diagnosing form consisting sample uploading and general information input and and about section

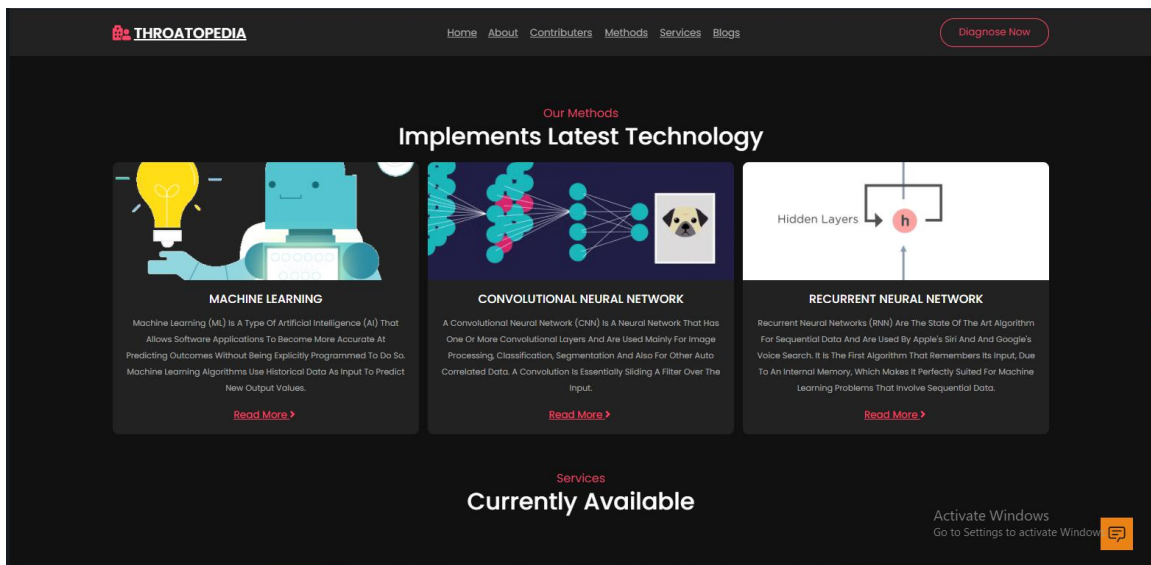


Fig. 4.10 Represents about us section

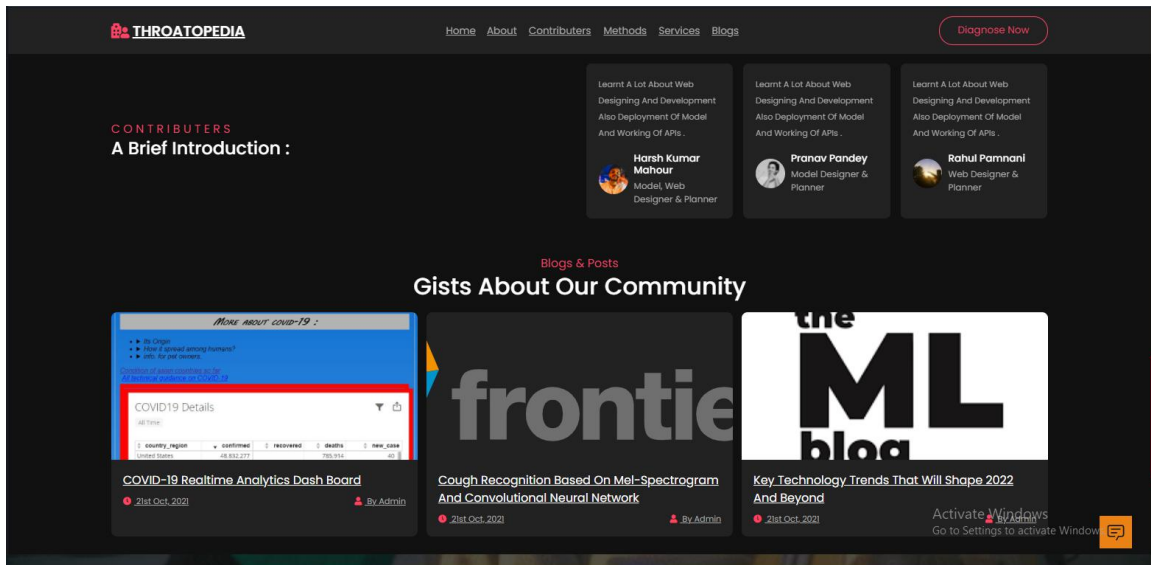


Fig. 4.11 Shows brief introduction about members and some blogs about COVID-19

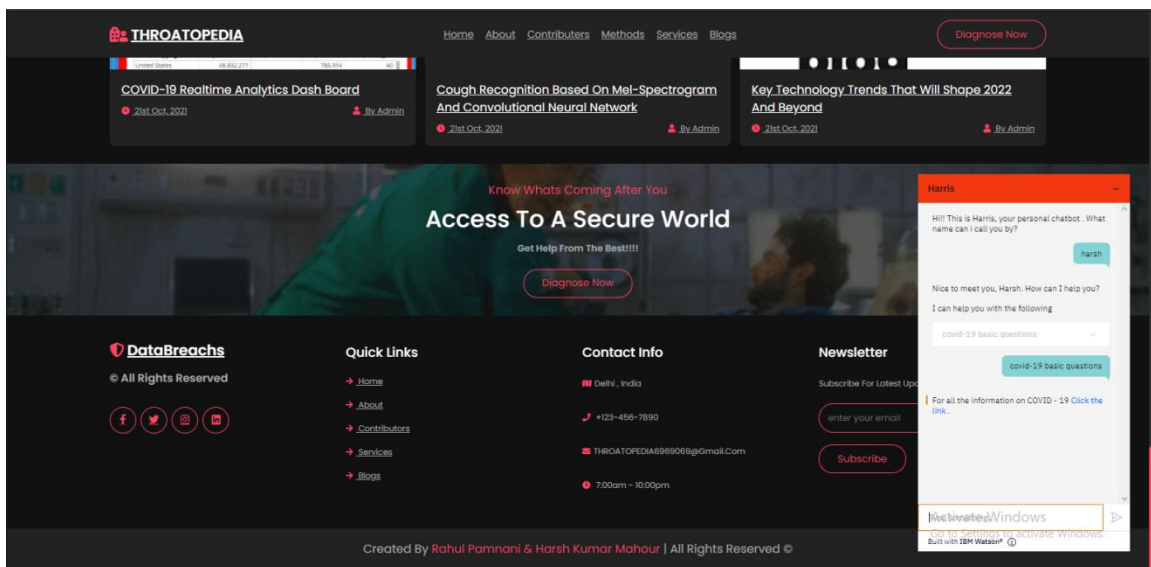


Fig. 4.12 Representing the footer section and a chat bot providing basic information about COVID-19 and other interactive talks

Fig. 4.11 and **Fig. 4.12** represents the contributors section and footer section along with that a personalised chat bot is provided which gives basic information about spreading awareness about covid-19 and other information.

Chapter 5

Conclusion and Future Scopes

Based on the model presented in this project report, the following conclusions have been drawn about working with neural network models:

5.1 Conclusion

The detection of COVID-19 using speech signals can serve as an important cost-effective tool as it does not involve any complicated medical test. This approach can easily diagnose the preliminary condition of a patient even without visiting a hospital and without the help of any medical staff as it serves as an automatic detection tool. A reliable audio feature called MFCC is presented and used for the detection of COVID-19 and the performance of the method is tested using two standard speech databases. The presented model has been demonstrated to be superior to other existing speech-based COVID-19 detection models reported in the literature. However, it is suggested that the detection accuracy needs to be ascertained by appropriate medical experts. The performance can be further increased by combining the MFCC with other temporal and statistical features. The presented MFCC features are based on the selection of the best possible conversion scale and frequency range of the Cepstral filter bank by using the bio-inspired technique. This is achieved by the identification of the appropriate sound patterns to efficiently detect COVID-19 and the application of speech enhancement schemes for the improvement of the classification performance.

5.2 Future Scopes

However, the classification accuracy can further be improved by using deep learning-based techniques. The attributes given in the dataset are breath, cough, and voiced vowel sounds. Moreover, the analysis can be extended to study the phonetic relevance and identification of phonemic grouping of speech-based COVID-19 detection. For this study, there is a requirement for the preparation of the phonetically balanced dataset of COVID19. The optimization method of the filter bank parameters can also be extended to different mechanical applications of cepstral analysis, where the properties of the input signal are quite different from that of standard human speech signals.

It has been determined that the features obtained with MFCC from traditional machine learning methods can distinguish COVID-19(+) ones with high performance. We think that alternative automatic detection systems with such high performance will be useful in the evaluation of COVID-19(+) and many other various cases from several pulmonary diseases as we get more datasets of patients from databases across the globe. It has also been observed that existing CRNN algorithms work with better efficiency as we provide more and more data to our model as the machine trains itself for diverse situations and samples and improves its precision so adding/combining more data to our existing data will also improve its precision and accuracy.

References

- [1] Vishnu Vandana Kolisetty and Dharmendra Singh Rajput, “A REVIEW ON THE SIGNIFICANCE OF MACHINE LEARNING FOR DATA ANALYSIS IN BIG DATA”, (Received: 2-Aug.-2019, Revised: 26-Oct.-2019, Accepted: 16-Nov.-2019).
- [2] Haomin Zhang, Ian McLoughlin, Yan Song, “ROBUST SOUND EVENT RECOGNITION USING CONVOLUTIONAL NEURAL NETWORKS”, 19-24 April, 2015.
- [3] Charles Bales, Muhammad Nabeel, Charles N. John, Usama Masood, Haneya N. Qureshi, Hasan Farooq, Iryna Posokhov and Ali Imran, “Can Machine Learning Be Used to Recognize and Diagnose Coughs?”, 2020.
- [4] Emre C. Akır, Giambattista Parascandolo, Toni Heittola, Heikki Huttunen, and Tuomas Virtanen, “Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection”, 21 Feb, 2017.
- [5] P. von Platen, C. Zhang, P. C. Woodland, “Multi-Span Acoustic Modelling using Raw Waveform Signals”, 3 Oct, 2019.
- [6] Roy Rudolf Huizen, Florentina Tatrin Kurniati, “Feature extraction with Mel scale separation method on noise audio recordings”, Sep 15, 2021.
- [7] Quan Zhou, Jianhua Shan, Wenlong Ding, Chengyin Wang, Shi Yuan, Fuchun Sun, Haiyuan Li and Bin Fang, “Cough Recognition Based on Mel-Spectrogram and Convolutional Neural Network”, Shanghai University, China, 07 May, 2021.
- [8] World Health Organization. Coronavirus Disease 2019 (COVID-19) Situation Report; Technical Report March; World Health Organization: Geneva, Switzerland, 2020.