DETECTION OF COVID-19 USING DEEP LEARNING

Guide Details:

Mrs. Y. Bhanusree

Assistant Professor

Department of CSE

Team Details:

17071A05I8 - B. Sree Deeksha

17071A05I9 - D. Ajay

17071A05L1 - M. Eshwar

17071A05N4 - V. Sohan

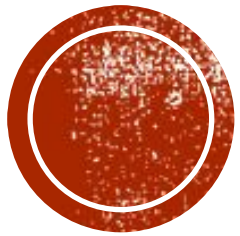
AGENDA

- Abstract
- Introduction
- Literature Survey
- System Requirements
- Methodology
- System Design (UML Diagrams)
- Implementation
- Conclusion
- Future Scope
- References

ABSTRACT

ABSTRACT

- COVID-19 has been proved to be deadly contagious and with new variants forming everyday, people are at greater risk.
- One should visit the test centre to get tested for COVID-19 and the test is not only uncomfortable but also very risky.
- Scientists have been using machine learning algorithms to analyse different types of coughs an distinguish them.
- This project proposes a deep learning model which analyses negative and positive tested cough samples to detect COVID-19.
- The proposed model uses a CNN with Bidirectional LSTM to build a deep neural network model which is used to analyse the cough samples.



INTRODUCTION

INTRODUCTION

- Increase in the number of cases and new variants.
- Limited number of test centres for the disease.
- Medical workers are getting out-numbered and are in constant danger of contracting the virus.
- To create a simpler way to diagnose COVID-19 for everyone, without having to visit a test centre.
- A deep learning algorithm which uses cough samples as input to check whether a person has COVID-19 or not.
- A CNN-Bidirectional LSTM model to train and test on the voice data.
- Requested and acquired dataset from Cambridge University for this project.

LITERATURE SURVEY

LITERATURE SURVEY

Sno	Author	Paper title and year of publication	Publication	Methodology/ drawbacks
1	Gowtham Ramesh, Dr Sundeep Teki	Covid or just a cough? AI for detecting covid from cough sounds.	KDNuggets, december, 2020	Used CNN on the same dataset and got 80% accuracy.
2	Ali Imran, Iryna Posokhova, Haneya N. Qureshi, Usama Masood, Muhammad Sajid Riaz, 04/05/2020	AI enabled preliminary diagnosis for COVID-19 from cough samples via an app	Elsevier, May 2020	This model detects the cough and splits the output into three possible outcomes using SVM and deep transfer learning. This algorithm needs feature extraction, which is time consuming.

LITERATURE SURVEY

Sno	Author	Paper title and year of publication	Publication	Methodology/ drawbacks
3	Chloë Brown, Jagmohan Chauhan, Andreas Grammenos, 19/07/2020	Exploring Automatic Diagnosis of COVID-19 from Crowdsourced Respiratory Sound Data	arXiv: 2006.05919v2 [cs.SD], July 2020	This model uses logistic regression, and SVM to detect covid. It becomes inaccurate as the data increases.
4	Tim morris, Simon Herper	Classification of covid19 coughs using Mel frequency cepstral and CNN, 26/08/2020	researchsquare, August 2020.	This paper uses MFCC CNN for the classification model, sensitive to noise due to its dependence on the spectral form.

LITERATURE SURVEY (RELATED WORK)

Sno	Author	Paper title and year of publication	Publication	Methodology/ drawbacks
5	Thomas F. Quatiery, Tanya Talkar, Jeffery S. palmer.	A framework for biomarkers of COVID-19 based on coordination of speech production subsystems.	29 May, 2020	The approach is based on the complexity of neuromotor coordination across speech subsystems
6	Jia-ming liu, Myingyu you, Zheng wang, Guo Zheng Li.	Cough detection using deep neural networks.	December, 2014	This is a two step cough detection system which uses deep neural networks on recordings of 20 patients which lasts upto 24 hours to detect and assess cough.

LITERATURE SURVEY (RELATED WORK)

Sno	Author	Paper title and year of publication	Publication	Methodology/ drawbacks
7	Myingyu You, Zheng Wang	Cough event classification by pre trained deep neural network.	November, 2015	This deep neural network contains pre trained and fine tuning, followed by hidden Markov model to assess the cough.
8	N. L. Loo, Y. S. Chew, G. Arunachalam.	A machine learning model for real time asynchronous breathing monitoring.	12 December, 2018	This proposes a method for asynchronous breathing detection using CNN algorithm.

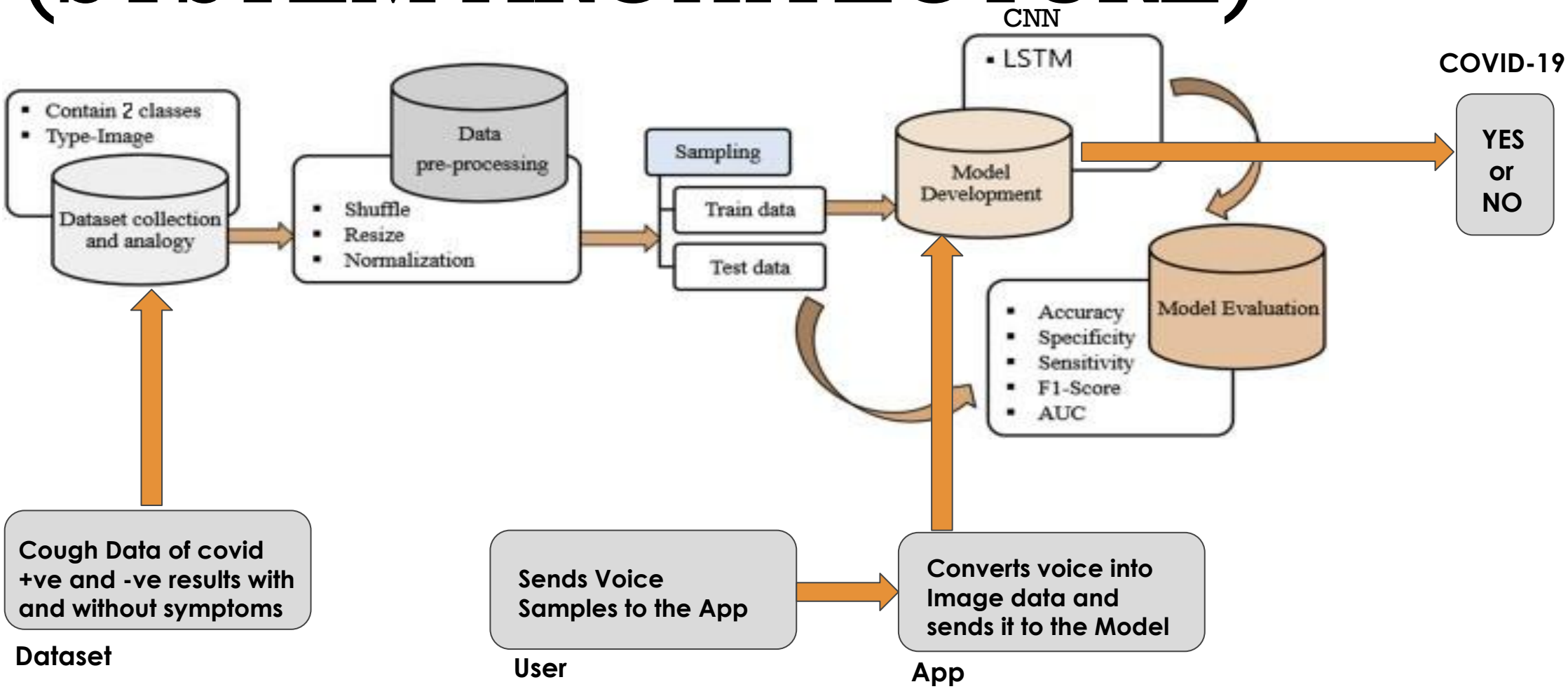
SYSTEM REQUIREMENTS

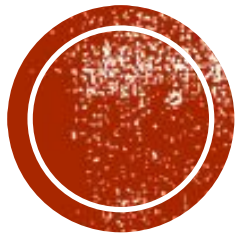
- Software Requirements:
  - Google Colab
  - Jupyter Notebook
  - Streamlit framework
- Hardware Requirements:
  - RAM – 128 GB DDR4 2133 MHz.
  - 2 TB Hard Disk (7200 RPM) + 512 GB SSD
  - GPU – NVidia TitanX Pascal (12 GB VRAM)
  - Jupyter notebook to run the model.

METHODOLOGY

System Architecture

METHODOLOGY (SYSTEM ARCHITECTURE)

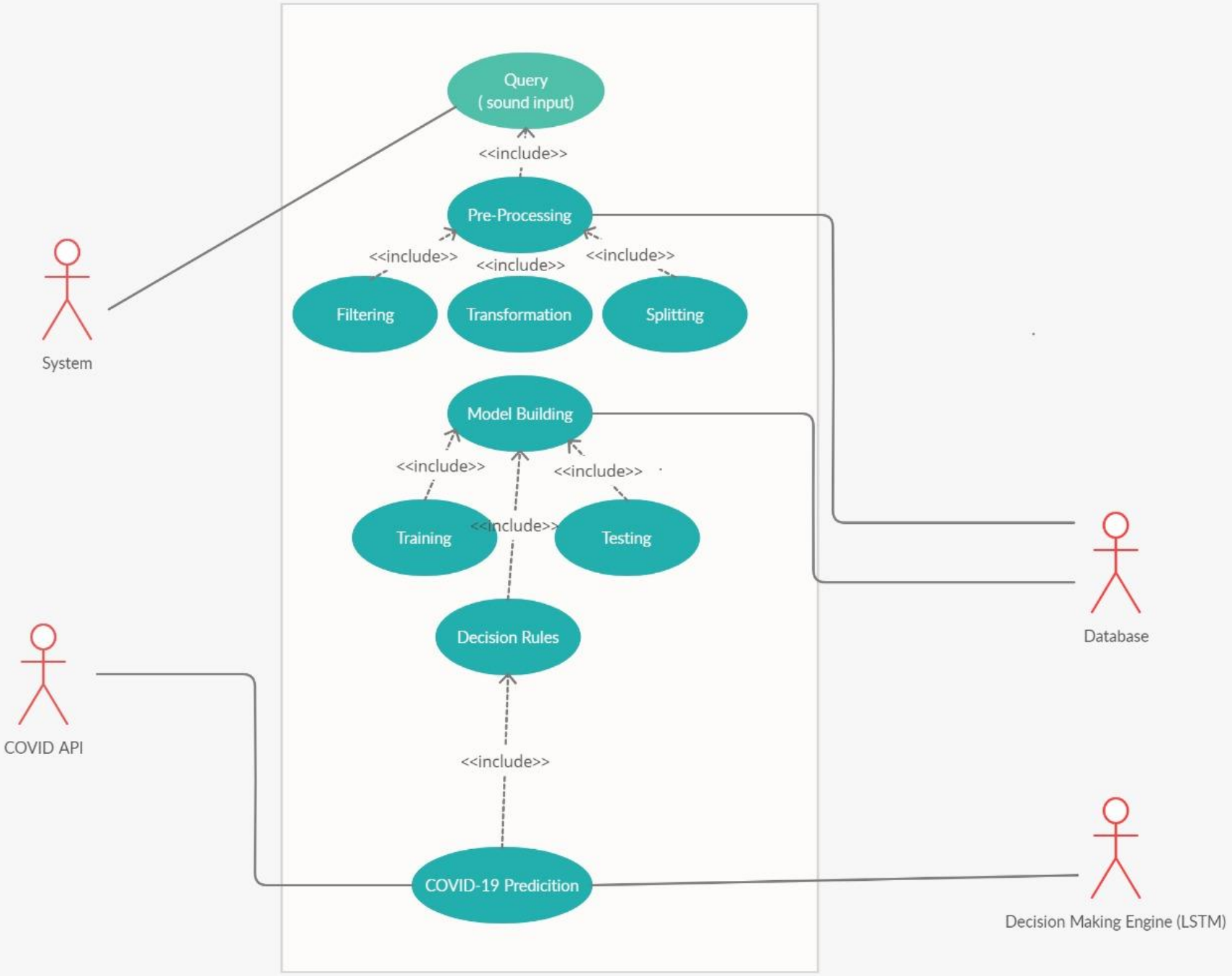




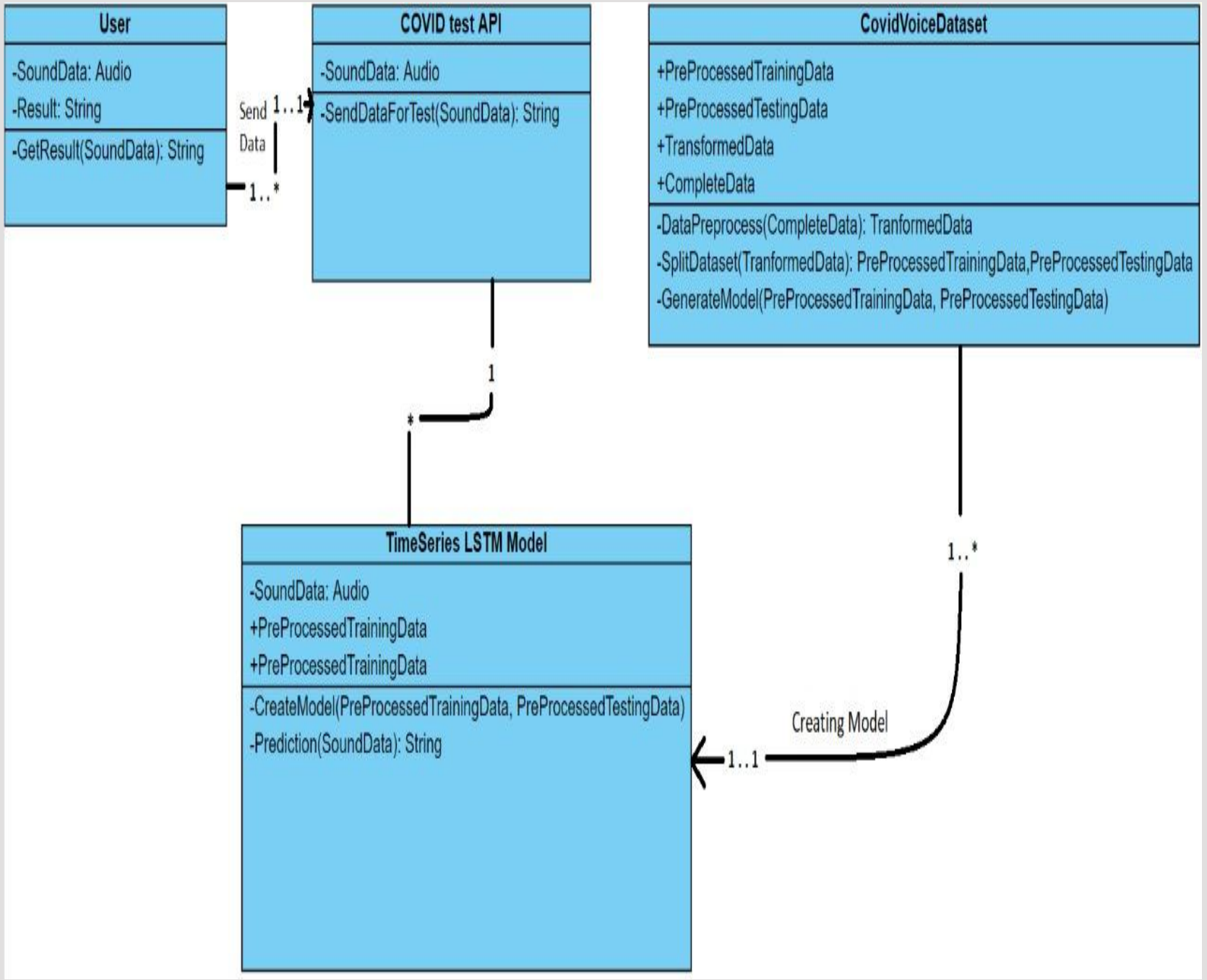
SYSTEM DESIGN

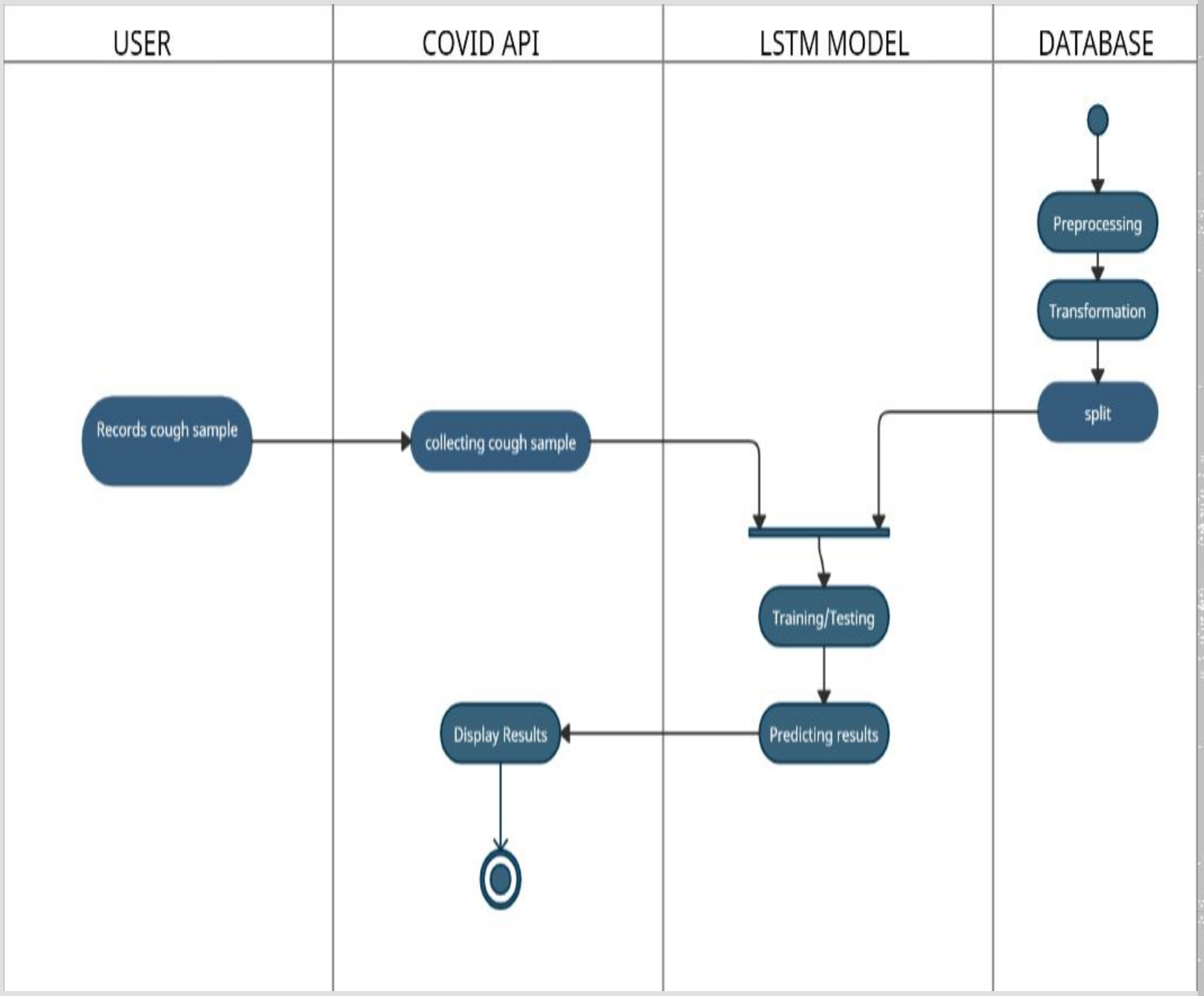
Use case, Class, Activity, and Sequence diagrams.

USE CASE DIAGRAM

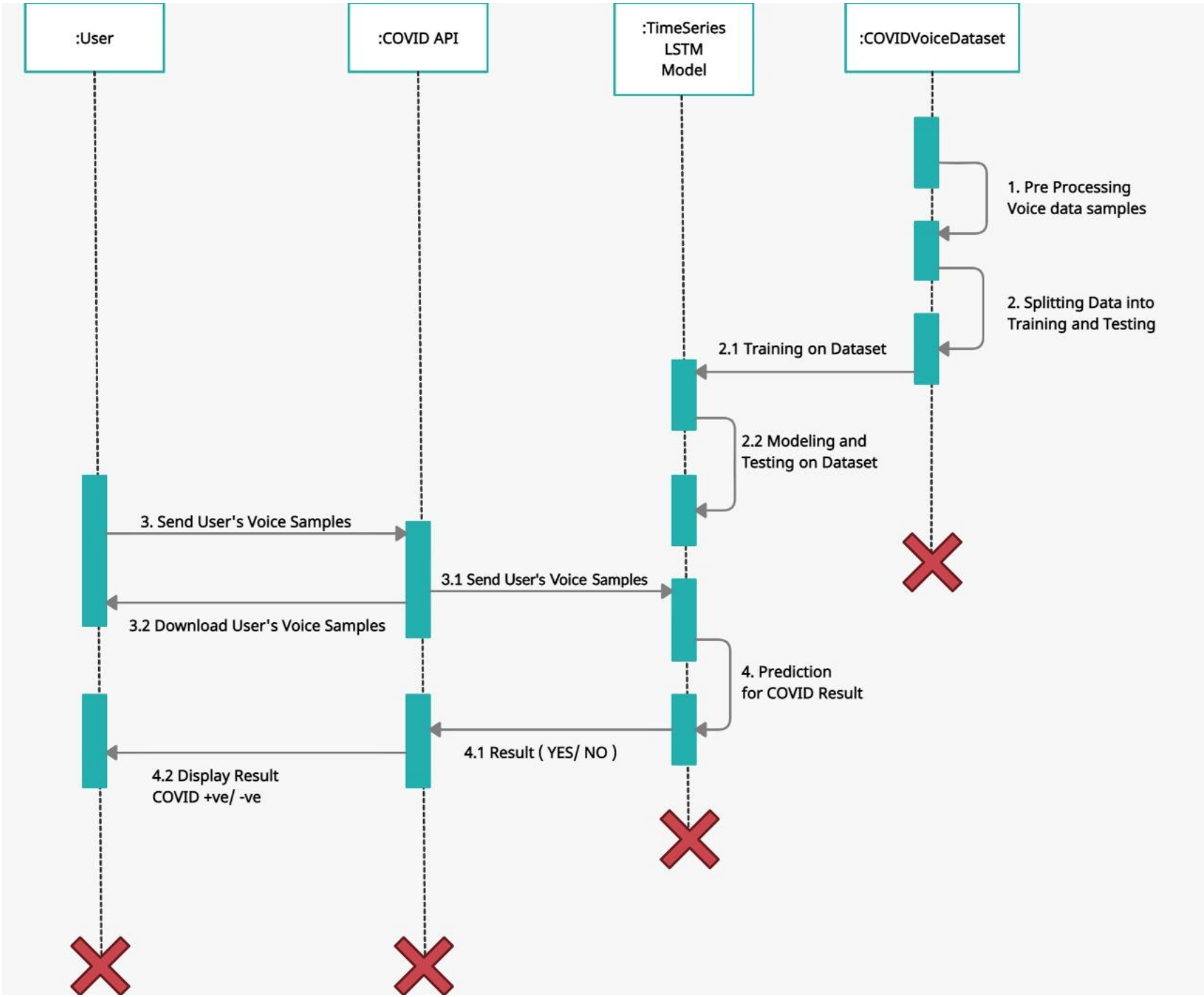


CLASS DIAGRAM

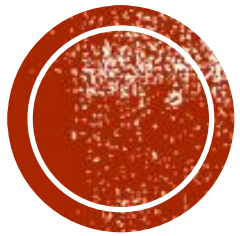




ACTIVITY DIAGRAM



SEQUENCE DIAGRAM



IMPLEMENTATION

IMPLEMENTATION

- Data augmentation
- Utility Functions for Data Augmentation
- Comprehensive MFCC Features
- Feature Extraction
- Building the Model
- Model Architecture
- Fitting Data into the Model

```
import os
import glob
import matplotlib.pyplot as plt
import numpy as np
import scipy
import math
from scipy.io.wavfile import write
from random import gauss
from audiomentations import Compose, AddGaussianNoise, Shift
from scipy.stats import zscore

from sklearn.model_selection import train_test_split

#for loading, visualizing and playing audio files
import librosa
import librosa.display
import IPython
from IPython.display import Audio
```

```
yes_data = glob.glob("C:/Users/Alpha/Desktop/Major Project/Data1/yes/*.wav")
no_data = glob.glob("C:/Users/Alpha/Desktop/Major Project/Data1/no/*.wav")
print(len(yes_data),len(no_data))
```

```
141 408
```

LIBRARIES USED AND INITIAL DATA

This was initially performed to increase the size of the dataset.

Librosa is the package which plays the major role in augmenting audio data.

Initial data: 141 yes and 408 no samples.

```
# Adding noise to data
i=0
# len(yes_data)
for i in range(len(yes_data)):
    # Loading one file at a time from yes_data
    signal = np.array(scipy.io.wavfile.read(yes_data[i])[1]).astype(np.float32)
    if type(signal[0])!=np.float32:
        continue
    # Augmentnoise to the audio data
    noisy_1 = gauss(0,100)
    noisy_2 = gauss(100,200)
    noisy_3 = gauss(200,300)
    noisy_4 = gauss(300,400)
    noisy_5 = gauss(400,500)

    noise_augmented_signal_1 = signal + noisy_1
    scaled = np.int16(noise_augmented_signal_1/np.max(np.abs(noise_augmented_signal_1)) * 32767)
    file_name = 'C:/Users/Alpha/Desktop/Major Project/Data1/yes_augmented/'+str(i+1)+'_11.wav'
    write(file_name, 44100, scaled)
    file_name = 'C:/Users/Alpha/Desktop/Major Project/Data1/yes_augmented/'+str(i+1)+'_12.wav'
    x,y=np.split(scaled,2)
    write(file_name, 44100, y+x)

    noise_augmented_signal_2 = signal + noisy_2
    scaled = np.int16(noise_augmented_signal_2/np.max(np.abs(noise_augmented_signal_2)) * 32767)
    file_name = 'C:/Users/Alpha/Desktop/Major Project/Data1/yes_augmented/'+str(i+1)+'_21.wav'
    write(file_name, 44100, scaled)
    file_name = 'C:/Users/Alpha/Desktop/Major Project/Data1/yes_augmented/'+str(i+1)+'_22.wav'
    x,y=np.split(scaled,2)
    write(file_name, 44100, y+x)

    noise_augmented_signal_3 = signal + noisy_3
    scaled = np.int16(noise_augmented_signal_3/np.max(np.abs(noise_augmented_signal_3)) * 32767)
    file_name = 'C:/Users/Alpha/Desktop/Major Project/Data1/yes_augmented/'+str(i+1)+'_31.wav'
    write(file_name, 44100, scaled)
    file_name = 'C:/Users/Alpha/Desktop/Major Project/Data1/yes_augmented/'+str(i+1)+'_32.wav'
    x,y=np.split(scaled,2)
    write(file_name, 44100, y+x)
```

AUGMENTING DATA

gauss function is used to return a random floating point value with gaussian distribution.

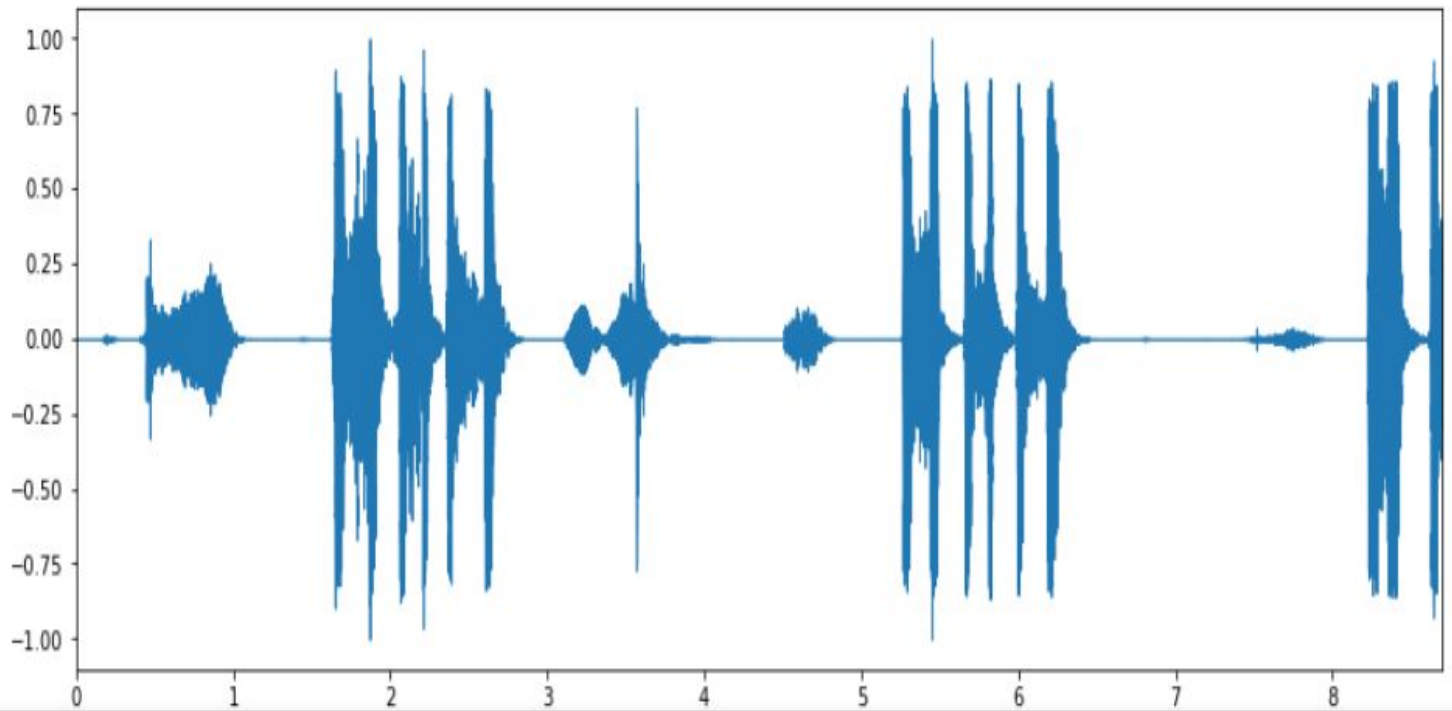
```
noise_augmented_signal_4 = signal + noisy_4
scaled = np.int16(noise_augmented_signal_4/np.max(np.abs(noise_augmented_signal_4)) * 32767)
file_name = 'C:/Users/Alpha/Desktop/Major Project/Data1/yes_augmented/'+str(i+1)+'_41.wav'
write(file_name, 44100, scaled)
file_name = 'C:/Users/Alpha/Desktop/Major Project/Data1/yes_augmented/'+str(i+1)+'_42.wav'
x,y=np.split(scaled,2)
write(file_name, 44100, y+x)

noise_augmented_signal_5 = signal + noisy_5
scaled = np.int16(noise_augmented_signal_5/np.max(np.abs(noise_augmented_signal_5)) * 32767)
file_name = 'C:/Users/Alpha/Desktop/Major Project/Data1/yes_augmented/'+str(i+1)+'_51.wav'
write(file_name, 44100, scaled)
file_name = 'C:/Users/Alpha/Desktop/Major Project/Data1/yes_augmented/'+str(i+1)+'_52.wav'
x,y=np.split(scaled,2)
write(file_name, 44100, y+x)

# print(str(i+1)+' "'+yes_data[i]+'"',"augmented with noise")
```

AUGMENTING DATA

Adding noise to the data and storing them in files.

```
signal = scipy.io.wavfile.read(yes_data[0])[1].astype(np.float32)
x, sr = librosa.load(yes_data[0], sr=44100)
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)

file_name = 'C:/Users/Alpha/Desktop/Major Project/Data1/yes_augmented/1_41.wav'
x, sr = librosa.load(file_name, sr=44100)
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)

file_name = 'C:/Users/Alpha/Desktop/Major Project/Data1/yes_augmented/1_42.wav'
x, sr = librosa.load(file_name, sr=44100)
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)
```

```
<matplotlib.collections.PolyCollection at 0x238334a10f0>
```



AUGMENTING DATA

matplotlib is used to plot the file.

After the augmentation:
682 YES and
524 NO samples.

```
## Some utility functions for augmenting dataset

def noise(data):
    noise_amp = 0.04*np.random.uniform()*np.amax(data)
    data = data + noise_amp*np.random.normal(size=data.shape[0])
    return data

def stretch(data, rate=0.70):
    return librosa.effects.time_stretch(data, rate)

def shift(data):
    shift_range = int(np.random.uniform(low=-5, high = 5)*1000)
    return np.roll(data, shift_range)

def pitch(data, sampling_rate, pitch_factor=0.8):
    return librosa.effects.pitch_shift(data, sampling_rate, pitch_factor)

def higher_speed(data, speed_factor = 1.25):
    return librosa.effects.time_stretch(data, speed_factor)

def lower_speed(data, speed_factor = 0.75):
    return librosa.effects.time_stretch(data, speed_factor)
```

UTILITY FUNCTIONS FOR AUGMENTING DATA

- noise : Adding random values.
- stretch: Resampling data using different sampling rate
- shift: Shifts data to left and right
- pitch: Changes pitch randomly
- speed: Changes speed of the file.

```python
## For extracting MFCC features

def extract_features(data):

    result = np.array([])

    mfccs = librosa.feature.mfcc(y=data, sr=22050, n_mfcc=58)
    # Calculate delta and delta2 MFCCs
    delta_mfccs = librosa.feature.delta(mfccs)
    delta2_mfccs = librosa.feature.delta(mfccs, order=2)
    # Comprehensive MFCCs for better feature extraction
    comprehensive_mfccs = np.concatenate((mfccs, delta_mfccs, delta2_mfccs))
    mfccs_processed = np.mean(comprehensive_mfccs.T,axis=0)
    result = np.array(mfccs_processed)

    return result
```

COMPREHENSIVE MFCCS

coefficients: signal, sample_rate, winlen, winstep, n_mfccs

```python
## Get all the mfcc features as well as augmented data

def get_features(data):
    # duration and offset are used to take care of the no audio in start and the ending of each audio files as seen above.
    #data, sample_rate = librosa.load(path, duration=3, offset=0.5, res_type='kaiser_fast')

    #without augmentation
    res1 = extract_features(data)
    result = np.array(res1)

    #noised
    noise_data = noise(data)
    res2 = extract_features(noise_data)
    result = np.vstack((result, res2)) # stacking vertically

    #stretched
    stretch_data = stretch(data)
    res3 = extract_features(stretch_data)
    result = np.vstack((result, res3))

    #shifted
    shift_data = shift(data)
    res4 = extract_features(shift_data)
    result = np.vstack((result, res4))

    #pitched
    pitch_data = pitch(data, sample_rate)
    res5 = extract_features(pitch_data)
    result = np.vstack((result, res5))

    #speed up
    higher_speed_data = higher_speed(data)
    res6 = extract_features(higher_speed_data)
    result = np.vstack((result, res6))

    #speed down
    lower_speed_data = higher_speed(data)
    res7 = extract_features(lower_speed_data)
    result = np.vstack((result, res7))

    return result
```

FEATURE EXTRACTION

noise, stretch, shift, pitch and speed functions are used to extract the features that are stored in vertical stack

```
## Building our model

from tensorflow.keras import backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (BatchNormalization, Conv1D, Dense, Input, LeakyReLU,
    TimeDistributed, Activation, Bidirectional, SimpleRNN, GRU, LSTM, MaxPooling1D, Dropout)
def final_model(input_dim, filters, kernel_size, conv_stride,
    conv_border_mode, units, output_dim=2):
    """ Build a deep network for speech
    """
    # Main acoustic input
    input_data = Input(name='the_input', shape=(None, input_dim))

    # LeakyRelu
    leaky_relu = LeakyReLU(alpha=0.3)(input_data)

    # CNN + maxpool
    conv_1d = Conv1D(filters, kernel_size,
                     strides=conv_stride,
                     padding=conv_border_mode,
                     activation='relu',
                     name='conv1d') (leaky_relu)
    # maxpool = MaxPooling1D(pool_size=4, strides=2, padding='valid')(conv_1d)

    # Batch Normalization
    conv1_normalized = BatchNormalization(name="maxpool_normalized")(conv_1d)

    # Bidirectionnal RNN
    bidir_rnn = Bidirectional(LSTM(units, activation="relu",return_sequences=True,
                implementation=2, name="bidir_rnn"))(conv1_normalized)
```

BUILDING THE MODEL

Input ---> Layer 1
Layer 1: Leaky Relu
Layer 2: Convolution
Layer 3: Batch Normalization
Layer 4: Bi-directional LSTM
Layer 5: Batch Normalization
Layer 6: Time Distributed Dense Layer
Layer 7: Dropout Layer
Layer 8: Time Distributed Dense Layer
Layer 9: Sigmoid Activation Function
Layer 9 ---> Output

```
# Batch Normalization
bidir_rnn_normalized = BatchNormalization(name="bidir_rnn_normalized")(bidir_rnn)

# Time distributed
time_dense1 = TimeDistributed(Dense(2))(bidir_rnn_normalized)

# Dropout
dropout = Dropout(0.4) (time_dense1)

# Time distributed
time_dense2 = TimeDistributed(Dense(output_dim))(dropout)

# Add sigmoid activation layer
y_pred = Activation('sigmoid', name='sigmoid')(time_dense2)

# Specify the model
model = Model(inputs=input_data, outputs=y_pred)

# DONE: Specify model.output_length
model.output_length = lambda x: cnn_output_length(
    x, kernel_size, conv_border_mode, conv_stride)

print(model.summary())
return model
```

BUILDING THE MODEL

Input ---> Layer 1
Layer 1: Leaky Relu
Layer 2: Convolution
Layer 3: Batch Normalization
Layer 4: Bi-directional LSTM
Layer 5: Batch Normalization
Layer 6: Time Distributed Dense Layer
Layer 7: Dropout Layer
Layer 8: Time Distributed Dense Layer
Layer 9: Sigmoid Activation Function
Layer 9 ---> Output

```
Model: "model_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       [(None, None, 174)]       0
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, None, 174)         0
_________________________________________________________________
conv1d (Conv1D)              (None, None, 1)           175
_________________________________________________________________
maxpool_normalized (BatchNor (None, None, 1)           4
_________________________________________________________________
bidirectional_13 (Bidirectio (None, None, 400)         323200
_________________________________________________________________
bidir_rnn_normalized (BatchN (None, None, 400)         1600
_________________________________________________________________
time_distributed_22 (TimeDis (None, None, 2)           802
_________________________________________________________________
dropout_11 (Dropout)         (None, None, 2)           0
_________________________________________________________________
time_distributed_23 (TimeDis (None, None, 2)           6
_________________________________________________________________
sigmoid (Activation)         (None, None, 2)           0
=================================================================
Total params: 325,787
Trainable params: 324,985
Non-trainable params: 802
_________________________________________________________________
None
```

MODEL ARCHITECTURE

```
## Adding new axis for all the features (x_train, y_train) and labels (x_test,y_test)

x_train = x_train.reshape(x_train.shape[0],1,x_train.shape[1])
y_train = y_train.reshape(y_train.shape[0],1,y_train.shape[1])
x_test = x_test.reshape(x_test.shape[0],1,x_test.shape[1])
y_test = y_test.reshape(y_test.shape[0],1,y_test.shape[1])
import os
# Compiling the model with binary loss and accuracy
model.compile(loss='binary_crossentropy', optimizer='adam', metrics= 'accuracy' )

# make results/ directory, if necessary
if not os.path.exists('results'):
    os.makedirs('results')

history = model.fit(x_train,y_train, epochs=100, validation_data=(x_test,y_test))
```

FITTING DATA INTO THE MODEL

Adding dummy dimension to get 3D data.
Compiling Model
Fitting Model

```
Epoch 1/100
212/212 [==============================] - 2s 8ms/step - loss: 0.6468 - accuracy: 0.6196 - val_loss: 0.6775 - val_accuracy:
0.5761
Epoch 2/100
212/212 [==============================] - 1s 5ms/step - loss: 0.6030 - accuracy: 0.6927 - val_loss: 0.6665 - val_accuracy:
0.6057
Epoch 3/100
212/212 [==============================] - 1s 5ms/step - loss: 0.5511 - accuracy: 0.7373 - val_loss: 0.5975 - val_accuracy:
0.6649
```

```
Epoch 97/100
212/212 [==============================] - 1s 5ms/step - loss: 0.3979 - accuracy: 0.8230 - val_loss: 0.3480 - val_accuracy:
0.8709
Epoch 98/100
212/212 [==============================] - 1s 5ms/step - loss: 0.3933 - accuracy: 0.8256 - val_loss: 0.3444 - val_accuracy:
0.8662
Epoch 99/100
212/212 [==============================] - 1s 5ms/step - loss: 0.4050 - accuracy: 0.8186 - val_loss: 0.4018 - val_accuracy:
0.8396
Epoch 100/100
212/212 [==============================] - 1s 5ms/step - loss: 0.4005 - accuracy: 0.8220 - val_loss: 0.3733 - val_accuracy:
0.8419
```

```
model.evaluate(x_test, y_test, batch_size=64, verbose=2)
```

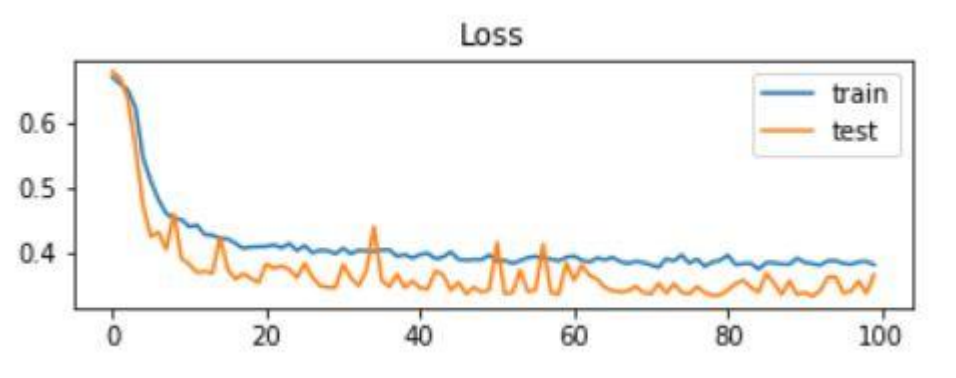
```
27/27 - 0s - loss: 0.3424 - accuracy: 0.8567

[0.34244832396507263, 0.856719970703125]
```

FITTING DATA INTO THE MODEL (100 Epochs)

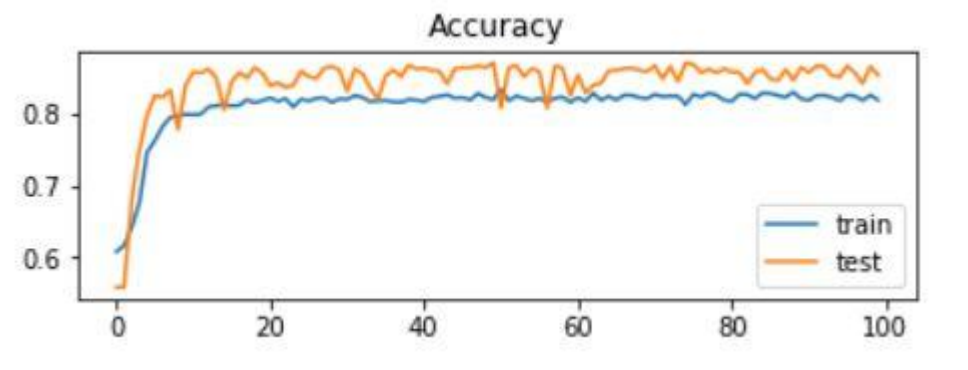
Accuracy: 0.8567
Loss: 0.3424

LOSS AND ACCURACY GRAPHS

```python
from matplotlib import pyplot

# plot loss during training
pyplot.subplot(211)
pyplot.title('Loss')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
```




```python
# plot accuracy during training
pyplot.subplot(212)
pyplot.title('Accuracy')
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='test')
pyplot.legend()
pyplot.show()
```



```
## classification report
print(classification_report(real,preds))
```

	precision	recall	f1-score	support
no	0.89	0.77	0.82	745
yes	0.83	0.93	0.88	944
accuracy			0.86	1689
macro avg	0.86	0.85	0.85	1689
weighted avg	0.86	0.86	0.85	1689

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

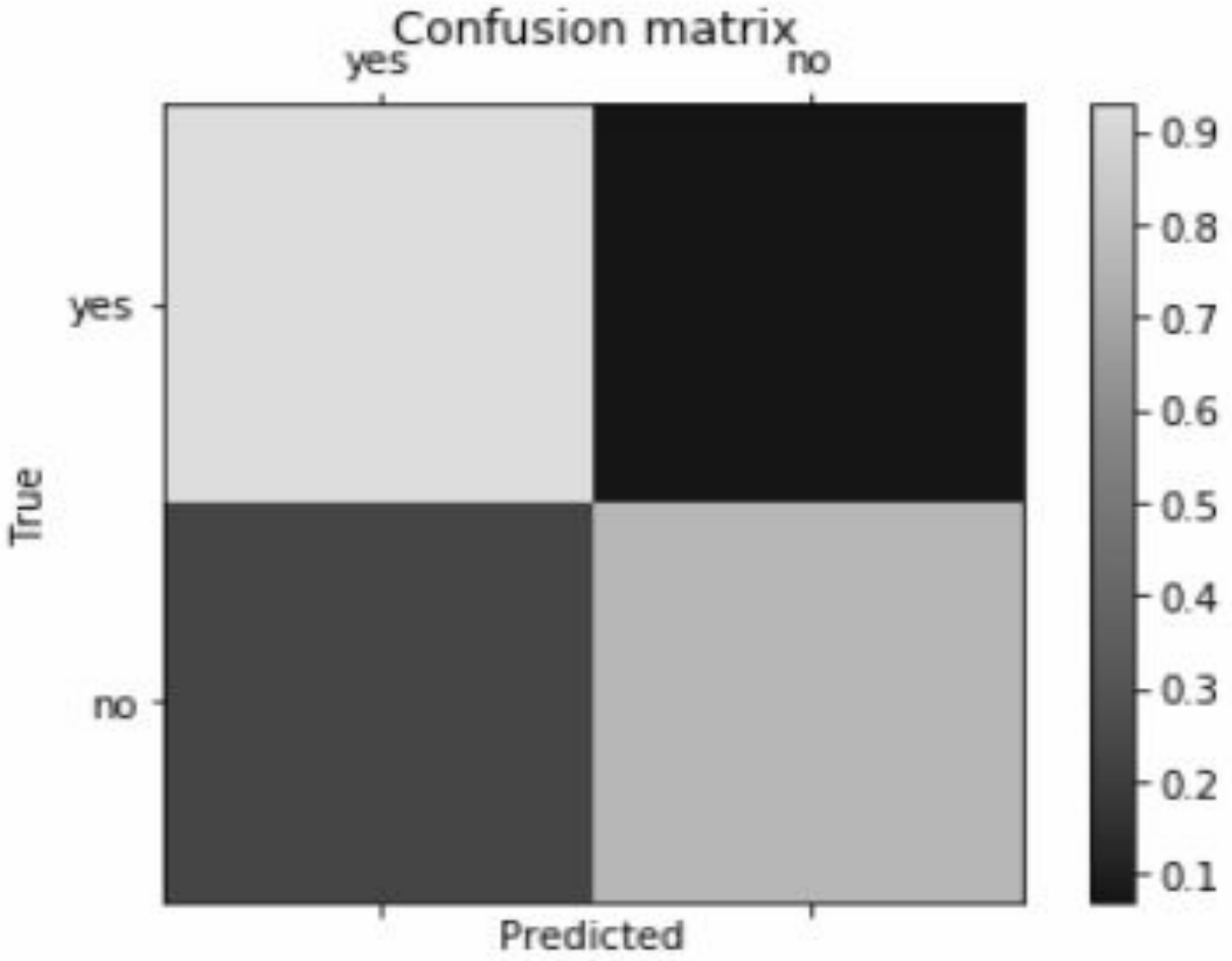
```
## Confusion Matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

labels = ['yes', 'no']
cm = confusion_matrix(real, preds, labels, normalize='true')
print(cm)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cm)
plt.title('Confusion matrix')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

F1 SCORE & CONFUSION MATRIX

F1 Score:

yes: 0.88
no: 0.82

```
[[0.92902542 0.07097458]
 [0.23489933 0.76510067]]
```



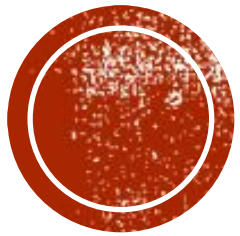
CONFUSION MATRIX

0.92 True Positive
0.07 False Negative
0.23 False Positive
0.76 True Negative

CONCLUSION

CONCLUSION

In the proposed model, instead of the conventional single model, hybrid model was used. This is because CNN does not provide contextual support of sequential data. LSTM was used along with CNN to provide contextual support. Contextual support plays a vital role because sound is nothing but sequential data of frames. Cough is indicated by fluctuation in these MFCC parameters of sequential data frames. For accurate cough analysis LSTM is used which takes into consideration the principle of sequentiality. Instead of traditional LSTM, bidirectional LSTM is used. Bidirectional LSTM uses a two way approach forward and reverse, which helps in getting full context of data which is not possible in traditional LSTM. This generates more data in the network which is not possible in other models. Thereby, high accurate results can be generated with minimal number of False Negative cases. By providing home based solutions, people can avoid getting exposed to the virus by getting tested at home and following home quarantine.



FUTURE SCOPE

FUTURE SCOPE

With the increase in automation of modern life and reliance on technology, AI induced models are useful not only to clinicians but also to patients alike, allowing the patients to an increased facility to monitor and manage their own condition. For healthcare services, such models/devices can be used to keep a track of patient data and screen patients based on their diseases and severity.

The proposed model helps in preliminary diagnosis of Covid-19 so that people are not needed to go to the test centres to get tested all the time. The model can be used as the preliminary test and one can decide to get the RT-PCR test after the results tested on cough samples by the model. It also opens the door to future learning in analysing different kinds of diseases through audio data.

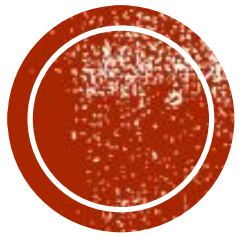
REFERENCES

REFERENCES

- Aykanat, M., Kılıç, Ö., Kurt, B. et al. Classification of lung sounds using convolutional neural networks. J Image Video Proc. 2017, 65 (2017). https://doi.org/10.1186/s13640-017-0213-2
- Brett Dalhberg IEEE spectrum, Detection of covid19 through sound. Wbur (2020). https://www.wbur.org/hereandnow/2020/11/27/coronavirus-sound-test
- T. F. Quatieri, T. Talkar and J. S. Palmer, "A Framework for Biomarkers of COVID-19 Based on Coordination of Speech-Production Subsystems," in IEEE Open Journal of Engineering in Medicine and Biology, vol. 1, pp. 203-206, 2020, doi: 10.1109/OJEMB.2020.2998051
- World Health Organization. Coronavirus disease (COVID-19) outbreak situation [Online].Available,https://www.who.int/emergencies/diseases/novel-coronavirus-2019. [Accessed 28 April 2020].

REFERENCES

- BBC. Coronavirus in South Korea: how ‘trace, test and treat’ may be saving lives. Accessed on: Mar. 31, 2020. [Online]. Available, https://www.bbc.com/news/world-asia-51836898; 2020.
- Science. Not wearing masks to protect against coronavirus is a ‘big mistake,’ top Chinese scientist says. Accessed on: April. 1, 2020. [Online]. Available, https://www.sciencemag.org/news/2020/03/not-wearing-masks-protect-against-coronavirus-big-mistake-top-chinese-scientist-says; 2020.
- Jort F. Gemmeke, Daniel P.W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio Set: An ontology and human labeled dataset for audio events. ICASSP, IEEE 9NIH audio collection, http://archive.is/2an9c International Conference on Acoustics, Speech and Signal Processing - Proceedings, pages 776–780, 2017.



THANK YOU