**SPEECH EMOTION RECOGNITION USING MULTI-TYPE FEATURES**

*Report submitted to the SASTRA Deemed to be University*

*as the requirement for the course*

**INT 300 - MINI PROJECT**

*Submitted by*

**KORRAPOLU JASWANTH**

**(Reg.No:124015039, B.Tech Information Technology)**

**NALLA SHAMUKHA BALAJI KUMAR**

**(Reg.No:124015064, B.Tech Information Technology)**

**JAVVADI HEMANTH KUMAR**

**(Reg.No:124015040, B.Tech Information Technology)**

**May 2023**

****

**SCHOOL OF COMPUTING**

**THANJAVUR, TAMIL NADU, INDIA – 613 401**



**SCHOOL OF COMPUTING**

**THANJAVUR – 613 401**

**Bonafide Certificate**

This is to certify that the report titled **“Speech Emotion Recognition Using Multi-Type Features ”** submitted as a requirement for the course, **INT300: MINI PROJECT** for B.Tech is a bonafide record of work done by **Mr.Korrapolu Jaswanth(Reg.no:124015039, B.Tech Information Technology),Mr.Javvadi Hemanth Kumar(Reg.no:124015040, B.Tech Information Technology),Mr.Nalla Shanmukha Balaji Kumar(Reg.no:124015064**

**,B.Tech Information Technology)** during the academic year 2022-23, in the School of Computing, under my supervision.

**Signature of Project Supervisor :**

**Name with Affiliation:** Dr. Anila Glory H, Asst. Professor - II, School of Computing

**Date :**

Mini Project *Viva voc*e held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Examiner 1 Examiner 2**

**Acknowledgements**

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing

us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam**

and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. A. Umamakeswari**, Dean, School of Computing ,**Dr. S. Gopalakrishnan**, Associate Dean, Department of Computer Application, **Dr. B.Santhi**, Associate Dean, Research, **Dr. V. S. Shankar Sriram**, Associate Dean, Department of Computer Science and Engineering, **Dr. R. Muthaiah**, Associate Dean, Department of Information Technology and Information & Communication Technology.

Our guide **Dr.Anila Glory H**, AP-II , School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions

helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of

the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me an opportunity to showcase my skills through project.

**Abbreviations**

Adam Adaptive Moment Estimation

CNN Convolution Neural Network

DCT Discrete Cosine Transform

DL Deep Learning

DNN Deep Neural Network

FFT Fast Fourier Transform

HD-MFM Hybrid Deep Learning with Multi type Features

HSFs High Level Static Features

LLDs Low level Descriptors

LSTM Long Short-Term Memory

MFCC Mel-Frequency Cepstral Coefficients

ReLU Rectified Linear Unit

RMS Root Means Square Propagation

RNN Recurrent Neural Network

SER Speech Emotion Recognition

UA Unweighted Accuracy

WA Weighted Accuracy

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No.** |
| 1.1 | HD-MFM Model Architecture | 2 |
| 1.2 | Architecture of CNN model | 3 |
| 1.3 | Architecture of DNN model | 4 |
| 1.4 | Architecture of LSTM model | 5 |
| 1.5 | Three blocks and final emotion classification are trained in one network | 5 |
| 4.1 | Value Count of Emotions and plots of EMODB | 23 |
|  | Waveplots of Waveform, Power Spectrum, Spectrogram and MFCC |  |
| 4.2.1 | Waveform | 23 |
| 4.2.2 | Power Spectrum | 24 |
| 4.2.3 | Spectrogram | 24 |
| 4.2.4 | MFCC | 24 |
|  | Extracted values using Mean, Max and Min |  |
| 4.3.1 | Through mean | 25 |
| 4.3.2 | Through max | 25 |
| 4.3.3 | Through min | 25 |
|  | CNN model |  |
| 4.4.1 | Summary of the CNN architecture | 26-27 |
| 4.4.2 | Accuracy of the CNN Architecture | 27 |
| 4.4.3 | Visualization of Model Accuracy and Model Loss Graph | 28 |
| 4.4.4 | Confusion matrix of the CNN Architecture | 28 |
|  | DNN model |  |
| 4.5.1 | Summary of the DNN Architecture | 29 |
| 4.5.2 | Accuracy of the DNN Architecture | 29 |
| 4.5.3 | Visualization of Model Accuracy and Model Loss | 30 |
| 4.5.4 | Confusion matrix of the DNN Architecture | 30 |
|  | LSTM model |  |
| 4.6.1 | Summary of the LSTM Architecture | 31 |
| 4.6.2 | Accuracy of the LSTM Architecture | 31 |
| 4.6.3 | Visualization of Model Accuracy and Model Loss | 32 |
| 4.6.4 | Confusion Matrix of the LSTM Architecture | 32 |
|  | Combined model of CNN, DNN and LSTM |  |
| 4.7.1 | Summary of the architecture | 33-34 |
| 4.7.2 | Accuracy of the merged architecture | 34 |
| 4.7.3 | Visualization of the Model Accuracy and Model Loss | 35 |
| 4.7.4 | Confusion Matrix of Merged Architecture | 35 |
| 7.1 | Screenshot of Indexing | 38 |

**Abstract**

Speech Emotion Recognition (SER) is a difficult task for improving human-computer interaction. Speech data is represented in various ways, and selecting the appropriate features to express the emotion behind the speech is difficult

The human brain can comprehensively judge the same thing in different dimensional representations to obtain the final result. Inspired by this, we believe that it is reasonable to have complementary advantages between different representations of speech data. To integrate the acoustic, temporal, and image information of speech, a Hybrid Deep Learning with Multi-type features Model (HD-MFM) is proposed. Convolutional Neural Network (CNN) is specifically used to extract picture information from the audio spectrogram. To extract the acoustic information from the statistical aspects of speech, deep neural networks (DNN) are used. Then, the Mel-Frequency Cepstral Coefficients (MFCC) of speech are used to extract the temporal information using Long Short-Term Memory (LSTM). Comprehensive tests have been conducted on the EMO-DB of SER to assess the viability and efficacy of the proposed HD-MFM. On the EMODB tests, the suggested HD-MFM achieves values of 91.25% of accuracy. The obtained results indicate the proposed HD-MFM can make full use of the effective complementary feature representations by separating strategy to further enhance the performance of SER.

**KEY WORDS**: Speech emotion recognition, Multi-type features, Hybrid feature selection

**Table of Contents**

**Title Page No.**

Bonafide Certificate ii

Acknowledgements iii

List of Figures iv

Abbreviations v

Notations vi

Abstract vii

1.Introduction 1

2. Merits and Demerits of the work 6

3. Source Code 10

4. Snapshots 20

5. Conclusion and Future Plans 23

6. References 24

7. Appendix -Base Paper 38

**CHAPTER 1**

**SUMMARY OF THE BASE PAPER**

**Base Paper Details:**

**Base paper URL:** <https://www.sciencedirect.com/science/article/pii/S1568494622006974>

**Title of the Base Paper**: MULTI -TYPE FEATURES SEPERATING FUSION LEARNING FOR SPEECH EMOTION RECOGNITION

**Year of publication**: 2022

**Journal**: Applied soft computing

**Cite score :** 12.4

**Impact factor** : 8.623

**Indexed in:** SCIENCE CITATION INDEX

**1.1 INTRODUCTION**

Emotion is a component of the human mental state, which in general applications like psychology describes the real sentiments that people have about a certain thing. It is possible to think about emotional recognition as multidimensional. For instance, the electroencephalogram, speech, eyes, and facial expressions of people can all be employed as the foundation for emotion identification. Speech emotion recognition (SER) has gained more and more attention in recent years because to the rising demand for human-computer connection. SER is broadly applicable in a variety of industries, including public security, disease diagnostics, fatigue detection, and natural human-machine interaction. The two components of a standard SER system are the exploration of the relevant emotion representative features and the development of an emotion-specific classifier.

With the advancement of deep learning, it has found widespread use nowadays in many industries, including SER. Several perceptrons make up a deep neural network (DNN). DNN is frequently used to learn HSFs in SER because of its high performance in learning plane-independent structural information. A feed forward neural network is a convolutional neural network (CNN). The primary attribute of CNN is its inherent convolution structure, which enhances its capacity for learning image information. In the SER community, recurrent neural networks (RNN) are another well-liked deep model. The benefit of RNN is that it incorporates memory cells into the model, which gives it a competitive advantage when handling temporal data. Long Short-Term Memory (LSTM) is an upgraded RNN model that can memorise information for a longer period of time and decreases the issue of gradient vanishing. The LSTM is frequently employed in SER because many speech features retrieved from frames have substantial temporal information. The ability of the human brain to make decisions using multidimensional information served as inspiration for the paper's exploration of the fusion method of voice data from various representations to enhance SER.

In contrast to earlier articles, this one uses the characteristics of various neural networks to teach high-level representations of speech components in three distinct dimensions: visual, temporal, and statistic. In order to distinguish between four emotional states—angry, happy, neutral, and sad—the Hybrid Deep Learning with Multi-type Features Model (HD-MFM), which integrates three separate neural networks, is developed.

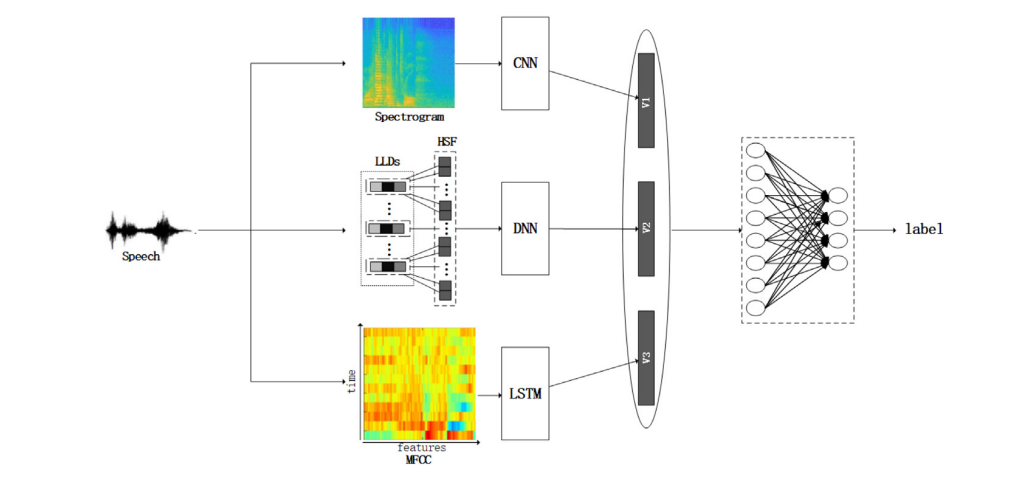


Fig. 1.1HD-MFM Model Architecture

Three blocks make up HD-MFM: the Spectrogram-CNN block, the HSF-DNN block, and the MFCC-LSTM block. Our method's training procedure may be summed up in three simple steps. The spectrogram, HSFs, and MFCC are the first three types of characteristics that we extract from the raw wave file. Then, three distinct deep learning models are used as inputs to train three distinct types of features and extract high-dimensional feature representations. In order to demonstrate the impact of this fusion feature, we finally integrate these three representations and employ two fully-connected layers.

Spectrogram-CNN block:

The time-domain signal that was received is often processed to create the spectrogram. The spectrogram can visualise the majority of speech data, including frequency, energy intensity, peak value, and speech length. Colour is an expression of vitality. The voice energy of the point is stronger the darker the colour. The raw signal S = [s(1), s(2),..., s(T)] ∈ ℜ 1×T , where T is the length of the signal, should first be pre-processed by pre-emphasis, frame division, and windowing in order to obtain the spectrogram. The raw signal S is then transformed into segment frames F = [f (1), f (2),..., f (K)]∈ ℜ K×M, where K is the frame number, and M is the frame length. After that, each frame is subjected to an N-point Fast Fourier Transformation (FFT) in order to determine the frequency spectrum. Short-Time Fourier Transform (STFT), where N is commonly 256 or 512, is another name for this transformation. The following equation is then used to finish the spectrogram:

*Pi = |FFT (fi)| 2/ N*

where fi is the signal S ith frame. As a result, the segment frames F are changed into the Spectrogram P =[p(1), p(2),..., p(K)] ∈ ℜ K×Q , where Q =N/2, after the FFT transformation. After FFT transformation, we will obtain matrices of various sizes in the temporal dimension since the K of each speech signal varies.

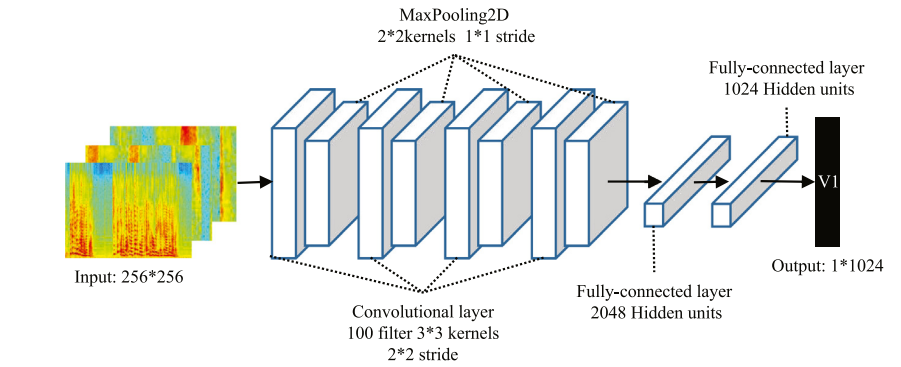


Fig. 1.2 Architecture of CNN model

The local association provided by sliding windows and the method of parameter sharing are the features that set the convolution layer apart from other layers. The convolution network utilised in this study can be described by the equation below:

*hljk = ∑ j ∑ k wljkxl−1jk + bl*

*xljk = σ(hljk)*

Due to its computational simplicity and quicker learning convergence, the activation function employed in this study is the Rectified Linear Unit (ReLU), and it may be represented as follows:

σ(x) = x if x>=0

0 otherwise

The purpose of the pooling layer is to eliminate unnecessary detail from the image, condense the size of the features, and, to a certain extent, prevent against over-fitting the neural network. By producing the maximum of each zone, maxpooling provides feature dimensionality reduction and nonlinear transformation. The following is an expression for the features created by the max-pooling layer:

alij = max(al−1mn)

HSF-DNN block:

For various LLDs, the HSF is a feature set of statistical algorithms. It best reflects the overall information of speech because to its abundant statistical information on each different prosodic, spectral, and voice quality elements. The benefit of choosing HSFs is that they have fewer feature dimensions, require less time to classify, and are more accurate. Two fully-connected layers of DNN are employed to learn the speech information in order to fully acquire and use it. The DNN is able to efficiently extract features that approximate the non-linear correlations between features in the initial collection. As a single-dimensional feature vector, the statistical HSF lacks spatial information. SER frequently uses DNN to train HSFs because of its effectiveness in learning plane-independent structural information. The two following formulas can be used to model DNN iteratively:

hij = ∑k wljkxl−1k + blj

xlj = σ(hlj )

The workflow is as follows in this block:

1. An utterance S = [s(1), s(2),..., s(T)] is given, and it is divided into frames F = [f(1), f(2),..., f (K)].
2. For each frame F , extracted the LLDs, L = [l(1), l(2), . . . , l(ℓ)] ∈ ℜK×ℓ , where ℓ indicates the number of LLDs to be extracted, and computed the HSFs, H(l) = [H1(l), H2(l), . . . , Hη(l)] ∈ ℜ1×ηℓ, where η is the number of statistical functions to be employed.
3. The HSFs were fed through the DNN, and the output value was utilised to create the statistic feature vector V2∈R1xn2, where n2 is the unit number of the final layer of the DNN.

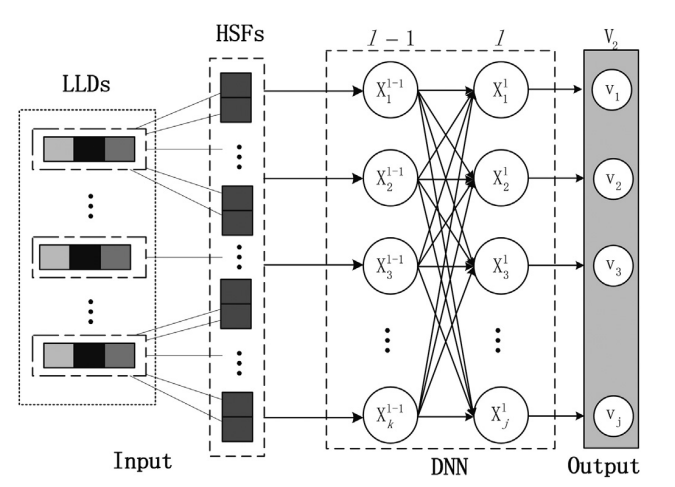


Fig. 1.3 Architecture of DNN model

MFCC-LSTM block:

To achieve the representation of speech in time dimension, the MFCC-LSTM block is used. The cepstrum-based speech qualities are frequently regarded as having a wealth of temporal data. A typical cepstrum feature in SER is MFCC. Because MFCC contains reduced feature dimensions and temporal information, it was chosen as our feature, Melcep = [M(1), M(2),..., M(c)] ∈ RKxc, where c stands for the cepstral coefficients, and frequently the c is 13 is used to generate the MFCC. A recurrent neural network called LSTM is made primarily for deriving long-term relationships from sequences. LSTM can extract the SER features associated with temporal information since MFCC contains abundant temporal information. An LSTM cell can be described using Eqs.

ft = σ(Wf [ht−1,xt] + bf )

it = σ(Wi[ht−1, xt] + bi)

C˜t = tanh(Wc [ht−1, xt] + bc )

Ct = ft ∗ C t−1 + it ∗ C˜t

Ot = σ(Wo[ht−1, xt] + bo)

ht = Ot + tanh(Ct)

Using the LSTM, a feature vector V3 ∈ ℜ 1×n3, where n3 is the unit number of the final LSTM layer, may be created that contains enough temporal data.

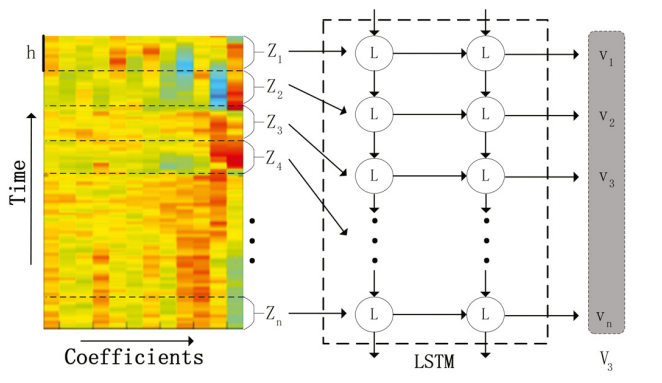


Fig. 1.4 Architecture of LSTM model

The fusion strategy of HD-MFM:

The speech representation in image-dimension is captured using the Spectrogram-CNN Block. The speech representation is provided in statistic-dimension to the HSFDNN Block. The speech representation is captured in time-dimension by the MFCC-LSTM Block. The three feature vectors V1∈ R 1×n1, V2 ∈R 1×n2, and V3 ∈R 1×n3 are concatenated into a vector V = [V1, V2, V3] ∈R1×n, where n = n1 + n2 + n3. This process is repeated three times. The Spectrogram-CNN block's output V1 is1 × 1024. The HSF-DNN block's output V2 is 1 × 1024. The MFCC-LSTM block's output V3 is 1 × 256. The joint feature vector V has a dimension of 1 × 2304. Separating blocks has the benefit of protecting each block's distinct qualities from the effect of other blocks. The benefit of merging is that each block can interact with one another through the global loss during training to update the gradient.

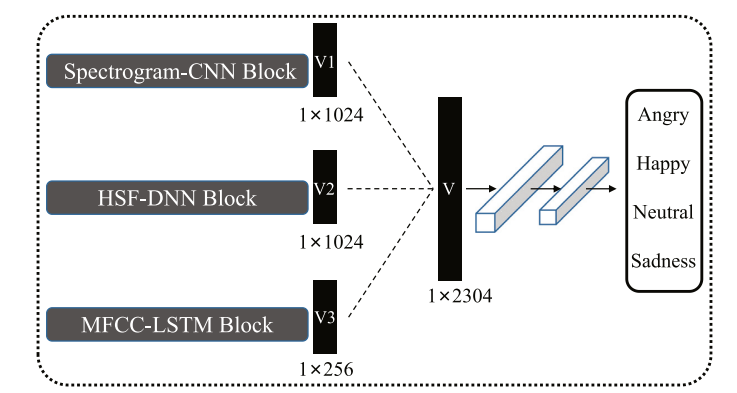


Fig. 1.5 Three blocks and final emotion classification are trained in one network

**CHAPTER-2**

**MERITS AND DEMERITS OF THE BASE PAPER**

**2.1 Literature Review**

Gangamohan P. et a has identified these Fundamental frequency (F0), strength of excitation (SoE) energy of excitation (EoE) speech features from the speech signals .However these features performed very poor on the different speech signals

Z. Peng et al.(2021) has identified these MFCC, X-vector speech features form the speech signals and used the CNN model to predict the emotion of the speaker and the accuracy is 66.8%

W. Fan et al.(2020) has obtained the Speech spectrogram from the speech signals and used the CNN model to train the neural network in order to predict he emotion of the speaker .The accuracy is 73%

H. Li et al. used the Log-mel filterbank energies (Log-MFBs), pitch, energy speech features and trained the CNN , dnn model to predict the emotion ,but it has produced the very less accuracy of 58% than the other methods .

D. Li et al used these MFCC, spectral roll-off point, spectral flux, spectral centroid, spectral entropy, spectral spread, zero-crossing rate, fundamental frequency, energy, energy entropy and their first-order difference speech features to train the CNN and lstm models and it produces an accuracy of 85%

**2.2 Merits**:

Many Deep learning algorithms are used to predict the emotion of speaker by training the models with different features . But every algorithm will not perform well on the data. By specifically taking into account rich speech feature representations, HD-MFM has achieved a significantly higher recognition accuracy in SER. So a multi-type hybrid deep learning model is used. Spectrogram-CNN offers clear advantages over the HSF-DNN and MFCC-LSTM on the EMO-DB database. This demonstrates how voice emotional information can be expressed more effectively using characteristics retrieved by Spectrogram-CNN. Spectrogram-CNN offers clear advantages over the HSF-DNN and MFCC-LSTM on the EMO-DB database.

**2.3 Demerits**:

Despite the good results of HD-MFM. The drawback is the imbalance problem of Speech emotion Recognition. This imbalance ratio makes the prediction of certain emotions very less . So the prediction of happy emotion is less. The size of the dataset is low hence it undergoes underfitting.

**CHAPTER 3**

**SOURCE CODE**

3.1 Waveplots

import numpy as np

import librosa, librosa.display

import matplotlib.pyplot as plt

FIG\_SIZE=(12,4)

file3="C:/Users/kiran/Desktop/MiniProject/archive (6)EmoDB/wav/03a04Ta.wav"

#Time vs Amplitude waveplot

signal3,sr3=librosa.load(file3,sr=22050)

librosa.display.waveshow(signal3,sr=sr3)

plt.xlabel("Time")

plt.ylabel("Amplitude")

plt.show()

#Frequency vs Magnitude

# FFT -> power spectrum

# perform Fourier transform

fft = np.fft.fft(signal3)

# calculate abs values on complex numbers to get magnitude

spectrum = np.abs(fft)

# create frequency variable

f = np.linspace(0, sample\_rate, len(spectrum))

# take half of the spectrum and frequency

left\_spectrum = spectrum[:int(len(spectrum)/2)]

left\_f = f[:int(len(spectrum)/2)]

# plot spectrum

plt.figure(figsize=FIG\_SIZE)

plt.plot(left\_f, left\_spectrum, alpha=0.4)

plt.xlabel("Frequency")

plt.ylabel("Magnitude")

plt.title("Power spectrum")

#Time vs Frequency spectogram

# STFT -> spectrogram

hop\_length =1024 # in num. of samples

n\_fft = 2048 # window in num. of samples

# calculate duration hop length and window in seconds

hop\_length\_duration = float(hop\_length)/sample\_rate

n\_fft\_duration = float(n\_fft)/sample\_rate

print("STFT hop length duration is: {}s".format(hop\_length\_duration))

print("STFT window duration is: {}s".format(n\_fft\_duration))

# perform stft

stft = librosa.stft(signal3, n\_fft=n\_fft, hop\_length=hop\_length)

# calculate abs values on complex numbers to get magnitude

spectrogram = np.abs(stft)

# display spectrogram

plt.figure(figsize=FIG\_SIZE)

librosa.display.specshow(spectrogram, sr=sample\_rate, hop\_length=hop\_length)

plt.xlabel("Time")

plt.ylabel("Frequency")

plt.colorbar()

plt.title("Spectrogram")

# apply logarithm to cast amplitude to Decibels

log\_spectrogram = librosa.amplitude\_to\_db(spectrogram)

plt.figure(figsize=FIG\_SIZE)

librosa.display.specshow(log\_spectrogram, sr=sample\_rate, hop\_length=hop\_length)

plt.xlabel("Time")

plt.ylabel("Frequency")

plt.colorbar(format="%+2.0f dB")

plt.title("Spectrogram (dB)")

#Time VS Mfcc Coefficients

# MFCCs

MFCCs = librosa.feature.mfcc(signal3, sample\_rate, n\_fft=n\_fft, hop\_length=hop\_length, n\_mfcc=39)

# display MFCCs

plt.figure(figsize=FIG\_SIZE)

librosa.display.specshow(MFCCs, sr=sample\_rate, hop\_length=hop\_length)

plt.xlabel("Time")

plt.ylabel("MFCC coefficients")

plt.colorbar()

plt.title("MFCCs")

# show plots

plt.show()

3.2 Loading the audio files, playing the audio file and feature extraction through mean, min and max

# Loading the audio files

import librosa

import librosa.display

import IPython.display as ipd

import os

for file in os.listdir("C:/Users/kiran/Desktop/MiniProject/archive (6)EmoDB/wav/"):

    print(file)

# Playing the sample audio file

ipd.Audio("C:/Users/kiran/Desktop/MiniProject/archive (6)EmoDB/wav/03a04Lc.wav")

# Through mean

def feature\_extraction(file\_path):

    #load the audio file

    x,sample\_rate=librosa.load(file\_path,res\_type='kaiser\_fast')

    #extract features from the audio

    mfcc=np.mean(librosa.feature.mfcc(y=x,sr=sample\_rate,n\_mfcc=50).T,axis=0)

    return mfcc

features={}

directory="C:/Users/kiran/Desktop/MiniProject/archive (6)EmoDB/wav/"

for audio in os.listdir(directory):

    audio\_path=directory+audio

    features[audio\_path]=feature\_extraction(audio\_path)

print(audio\_path)

print(features[audio\_path])

print(len(features[audio\_path]))

# Through max

def feature\_extraction(file\_path):

    #load the audio file

    x,sample\_rate=librosa.load(file\_path,res\_type='kaiser\_fast')

    #extract features from the audio

    mfcc=np.max(librosa.feature.mfcc(y=x,sr=sample\_rate,n\_mfcc=50).T,axis=0)

    return mfcc

features={}

directory="C:/Users/kiran/Desktop/MiniProject/archive (6)EmoDB/wav/"

for audio in os.listdir(directory):

    audio\_path=directory+audio

    features[audio\_path]=feature\_extraction(audio\_path)

print(audio\_path)

print(features[audio\_path])

len(features[audio\_path])

# Through min

def feature\_extraction(file\_path):

    #load the audio file

    x,sample\_rate=librosa.load(file\_path,res\_type='kaiser\_fast')

    #extract features from the audio

    mfcc=np.min(librosa.feature.mfcc(y=x,sr=sample\_rate,n\_mfcc=50).T,axis=0)

    return mfcc

features={}

directory="C:/Users/kiran/Desktop/MiniProject/archive (6)EmoDB/wav/"

for audio in os.listdir(directory):

    audio\_path=directory+audio

    features[audio\_path]=feature\_extraction(audio\_path)

print(audio\_path)

print(features[audio\_path])

len(features[audio\_path])

3.3 Feature\_Extraction.py

import numpy as np

import librosa

from tensorflow import keras

from sklearn.model\_selection import train\_test\_split

import numpy as np

import pandas as pd

emodb\_data = pd.read\_csv('Downloads/Emodb\_dataset.csv')

emodb\_data.head()

def get\_data(flatten=False, feature=1, mfcc\_len=39, mslen = 35000, n\_fft=512, hop\_length=128):

    """

    Read the files get the data perform the test-train split and return them to the caller

    :param mfcc\_len: Number of mfcc features to take for each frame

    :param flatten: Boolean specifying whether to flatten the data or not

    :return: 4 arrays, x\_train x\_test y\_train y\_test

    """

    from sklearn.preprocessing import LabelEncoder

    label=LabelEncoder()

    z=label.fit\_transform(emodb\_data['labels'])

    emodb\_data['labels']=z

    emodb\_data.drop(emodb\_data.index[emodb\_data['labels'] == 4], inplace=True)

    emodb\_data.drop(emodb\_data.index[emodb\_data['labels'] == 5], inplace=True)

    emodb\_data.drop(emodb\_data.index[emodb\_data['labels'] == 6], inplace=True)

    # store features in here

    data = []

    # obtain labels directly from data frame

    labels = np.array(emodb\_data.labels)

    fea=[]

    max\_fs = 0

    min\_sample = int('9' \* 10)

    cnt = 0

    # for paths in dataframe's column paths

    for path in emodb\_data.paths:

        # load the audio

        signal,fs = librosa.core.load(path,sr=16000)

        max\_fs = max(max\_fs, fs)

        s\_len = len(signal)

        # pad the signals to have same size if lesser than required

        if s\_len < mslen:

            pad\_len = int(mslen - s\_len)

            pad\_rem = int(pad\_len % 2)

            pad\_len /= 2

            signal = np.pad(signal, (int(pad\_len), int(pad\_len + pad\_rem)), 'constant', constant\_values=0)

        # else slice them

        else:

            pad\_len = int(s\_len - mslen)

            pad\_rem = pad\_len % 2

            pad\_len /= 2

            signal = signal[int(pad\_len):int(pad\_len + mslen)]

            min\_sample = min(len(signal), min\_sample)

        # Extract Mel Spectrogram features

        if feature==1:

            melspecfea = librosa.feature.melspectrogram(y=signal, sr=fs,n\_fft=n\_fft, hop\_length=hop\_length)

            melspecfea = librosa.power\_to\_db(melspecfea,ref=np.max)

            fea = (melspecfea- np.min(melspecfea))/np.ptp(melspecfea)

        # Extract Mfcc, delta, delta-delta features

        if feature==2:

            melspecfea = librosa.feature.melspectrogram(y=signal, sr=fs,n\_fft=n\_fft, hop\_length=hop\_length)

            mfcc = librosa.feature.mfcc(S=melspecfea, n\_mfcc=20)

            mfcc\_delta = librosa.feature.delta(mfcc)

            mfcc\_delta2 = librosa.feature.delta(mfcc, order=2)

            mfcc\_all= np.vstack((mfcc,mfcc\_delta,mfcc\_delta2))

            fea = (mfcc\_all- np.min(mfcc\_all))/np.ptp(mfcc\_all)

        # Extract Spectrogram features

        if feature==3:

            spectrogram1 = librosa.core.stft(signal, n\_fft=512)

            spectrogram1 = np.abs(spectrogram1)

            fea = (spectrogram1- np.min(spectrogram1))/np.ptp(spectrogram1)

        if flatten:

          # Flatten the data

          fea = fea.flatten()

        data.append(fea)

        cnt += 1

    # train test split on data

    x\_train, x\_test, y\_train, y\_test = train\_test\_split(data, labels, test\_size=0.1, random\_state=42)

    return np.array(x\_train),np.array(y\_train),np.array(x\_test),np.array(y\_test)

3.4 plotutils.py

import itertools

import numpy as np

from sklearn.metrics import classification\_report,accuracy\_score,confusion\_matrix

import numpy as np

import matplotlib.pyplot as plt

def model\_history(model\_history):

    #ploting 2 plots on horizontal axis

    fig,(ax1,ax2) =  plt.subplots(1,2,figsize=(16,8))

    # summarize history for accuracy

    ax1.plot(model\_history.history['accuracy'],c ="darkblue")

    ax1.plot(model\_history.history['val\_accuracy'],c ="crimson")

    ax1.set\_title('model accuracy')

    ax1.set\_ylabel('accuracy')

    ax1.set\_xlabel('epoch')

    ax1.legend(['train', 'test'], loc='upper right')

    # summarize history for loss

    ax2.plot(model\_history.history['loss'],c ="darkblue")

    ax2.plot(model\_history.history['val\_loss'],c ="crimson")

    ax2.set\_title('model loss')

    ax2.set\_ylabel('loss')

    ax2.set\_xlabel('epoch')

    ax2.legend(['train', 'test'], loc='upper right')

    fig.suptitle("Model History")

#Prints a classifcation report with accuracy below it

def c\_report(y\_true,y\_pred,target\_names=[]):

    print("Classifictaion Report")

    print(classification\_report(y\_true, y\_pred, target\_names=target\_names))

    acc\_scr = accuracy\_score(y\_true, y\_pred)

    print("Accuracy : "+ str(acc\_scr))

#plots confusion matrix

def plot\_confusion\_matrix(cm,

                          target\_names,

                          title='Confusion matrix',

                          cmap=None,

                          normalize=True):

    accuracy = np.trace(cm) / float(np.sum(cm))

    misclass = 1 - accuracy

    #give blueish color mapping

    if cmap is None:

        cmap = plt.get\_cmap('Blues')

    plt.figure(figsize=(10, 7))

    plt.imshow(cm, interpolation='nearest', cmap=cmap)

    plt.title(title)

    plt.colorbar()

    if target\_names is not None:

        tick\_marks = np.arange(len(target\_names))

        plt.xticks(tick\_marks, target\_names, rotation=0)

        plt.yticks(tick\_marks, target\_names)

    if normalize:

        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 1.5 if normalize else cm.max() / 2

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

        if normalize:

            plt.text(j, i, "{:0.4f}".format(cm[i, j]),

                    horizontalalignment="center",

                    color="white" if cm[i, j] > thresh else "black")

        else:

            plt.text(j, i, "{:,}".format(cm[i, j]),

                    horizontalalignment="center",

                    color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')

    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, misclass))

    plt.show()

3.5 read\_data.py

import os

import pandas as pd

#path where data is stored

path\_to\_files = 'C:/Users/kiran/Desktop/MiniProject/archive (6)EmoDB/wav'

#function to read data

def read\_data():

    #list  of emotion data

    emotion = []

    #list of paths of files in wav format

    path = []

    names = []

    for root, dirs, files in os.walk(path\_to\_files):

        for name in files:

            #Arger (Wut) -> Angry

            if name[5] == 'W':

                emotion.append('angry')

            #Langeweile -> Boredom

            elif name[5] == 'L':

                emotion.append('bored')

            # Ekel -> Disgusted

            elif name[5] == 'E':

                emotion.append('disgust')

            #Angst -> Angry

            elif name[5] == 'A':

                emotion.append('fear')

            #Freude -> Happiness

            elif name[5] == 'F':

                emotion.append('happy')

            #Trauer -> Sadness

            elif name[5] == 'T':

                emotion.append('sad')

            #Neutral

            elif name[5] == 'N':

                emotion.append('neutral')

            else:

                emotion.append('unknown')

            names.append(name)

            path.append(os.path.join(path\_to\_files,name))

        emodb\_df = pd.DataFrame(emotion, columns=['labels'])

        emodb\_df['source'] = 'EMODB'

        emodb\_df = pd.concat([emodb\_df, pd.DataFrame(names, columns=['names'])], axis=1)

        emodb\_df = pd.concat([emodb\_df, pd.DataFrame(path, columns=['paths'])], axis=1)

        return emodb\_df

Data = read\_data()

Data.to\_csv('Emodb\_data.csv')

3.6 Implementation of CNN

from FeatureExtractor import get\_data

from plotutils import model\_history,c\_report,plot\_confusion\_matrix

import numpy as np

from tensorflow import keras

import sklearn

from keras.models import  Model

from keras.layers import \*

from keras.regularizers import \*

from keras.layers import BatchNormalization

from sklearn.metrics import classification\_report,accuracy\_score,confusion\_matrix

from keras.callbacks import \*

import numpy as np

import pandas as pd

# read data and call get\_data function to get splits of data

emodb\_data = pd.read\_csv('Downloads/Emodb\_dataset.csv')

emodb\_data

from sklearn.preprocessing import LabelEncoder

label=LabelEncoder()

z=label.fit\_transform(emodb\_data['labels'])

z

emodb\_data['labels']=z

emodb\_data

# for 4 classes do some changes in feature extarctor file

X\_train, Y\_train, X\_test, Y\_test = get\_data()

Y\_train = keras.utils.to\_categorical(Y\_train,7)

Y\_test = keras.utils.to\_categorical(Y\_test,7)

Y=np.array(emodb\_data.labels)

from sklearn.utils.class\_weight import compute\_class\_weight

weight = compute\_class\_weight(class\_weight='balanced', classes=np.unique(Y), y=Y)

weight = {i : weight[i] for i in range(4)}

epochs = 15

input1 = Input(shape=(X\_train.shape[1], X\_train.shape[2], 1))

# First Conv2D block

conv1 = Conv2D(100, kernel\_size=(3, 3),strides=(2, 2),activation=None,padding='same',

               kernel\_initializer='he\_normal', kernel\_regularizer=l2(0.0001), bias\_regularizer=l2(0.0001))(input1)

batch1 = BatchNormalization()(conv1)

elu1 = ELU(alpha=1.0)(batch1)

pool1 = MaxPooling2D(pool\_size=(2, 2),strides=(1,1))(elu1)

dropout1 = Dropout(0.5)(pool1)

# Second Conv2D block

conv2 = Conv2D(100, kernel\_size=(3, 3),strides=(2, 2),padding='same',

               activation=None,kernel\_regularizer=l2(0.0001), bias\_regularizer=l2(0.0001) )(dropout1)

batch2 = BatchNormalization()(conv2)

elu2 = ELU(alpha=1.0)(batch2)

pool2 = MaxPooling2D(pool\_size=(2,2),strides=(1,1))(elu2)

dropout2 = Dropout(0.5)(pool2)

conv3 = Conv2D(100, kernel\_size=(3, 3),strides=(2, 2),activation=None,padding='same',

               kernel\_regularizer=l2(0.0001), bias\_regularizer=l2(0.0001))(dropout2)

batch3 = BatchNormalization()(conv3)

elu3 = ELU(alpha=1.0)(batch3)

pool3 = MaxPooling2D(pool\_size=(2, 2),strides=(1,1))(elu3)

dropout3 = Dropout(0.5)(pool3)

conv4 = Conv2D(100, kernel\_size=(3, 3),strides=(2, 2),activation=None,padding='same',

                kernel\_regularizer=l2(0.0001), bias\_regularizer=l2(0.0001))(dropout3)

batch4 = BatchNormalization()(conv4)

elu4 = ELU(alpha=1.0)(batch4)

pool4 = MaxPooling2D(pool\_size=(2, 2),strides=(1,1))(elu4)

dropout4 = Dropout(0.5)(pool4)

flat1=Flatten()(dropout4)

den1 = Dense(64, activation='relu',kernel\_regularizer=l2(0.0001))(flat1)

den2 = Dense(7, activation='softmax')(den1)

model= Model(inputs=input1,outputs=den2)

# Early stopping callback tracking val\_loss

stop\_early = EarlyStopping(monitor='val\_loss', mode='min',

                           verbose=1, patience=15)

# Model Checkpoint callback tracking val\_accuracy

checkpoint = ModelCheckpoint(

    'model1.h5',

    monitor = 'val\_accuracy',

    verbose = 1,

    save\_best\_only = True

)

model.compile(optimizer='adam',loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=False),metrics=['accuracy'])

model

history = model.fit(X\_train, Y\_train, validation\_split=0.15, class\_weight = weight, batch\_size=32,epochs = epochs,callbacks=[checkpoint,stop\_early])

predict = model.predict(X\_test)

loss, accu = model.evaluate(X\_test,Y\_test,verbose=1)

labels\_pred = np.argmax(predict, axis = -1)

labels\_true = np.argmax(Y\_test, axis = -1)

model\_history(history)

plot\_confusion\_matrix(cm = confusion\_matrix(labels\_true, labels\_pred),normalize = True,

                    target\_names = ['Angry','Happy', 'Neutral', 'Sad'],title = "Confusion Matrix")

3.7 Implementation of DNN

from FeatureExtractor import get\_data

from plotutils import model\_history,c\_report,plot\_confusion\_matrix

import numpy as np

from tensorflow import keras

import sklearn

from keras.models import  Model

from keras.layers import \*

from keras.regularizers import \*

from keras.layers import BatchNormalization

from sklearn.metrics import classification\_report,accuracy\_score,confusion\_matrix

from keras.callbacks import \*

import numpy as np

import pandas as pd

# read data and call get\_data function to get splits of data

emodb\_data = pd.read\_csv('Downloads/Emodb\_dataset.csv')

from sklearn.preprocessing import LabelEncoder

label=LabelEncoder()

z=label.fit\_transform(emodb\_data['labels'])

z

emodb\_data['labels']=z

emodb\_data

# for 4 classes do some changes in feature extarctor file

X\_train, Y\_train, X\_test, Y\_test = get\_data()

Y\_train = keras.utils.to\_categorical(Y\_train,7)

Y\_test = keras.utils.to\_categorical(Y\_test,7)

Y=np.array(emodb\_data.labels)

from sklearn.utils.class\_weight import compute\_class\_weight

weight = compute\_class\_weight(class\_weight='balanced', classes=np.unique(Y), y=Y)

weight = {i : weight[i] for i in range(7)}

input1 = Input(shape=(X\_train.shape[1], X\_train.shape[2], 1))

flat1=Flatten()(input1)

dense1=Dense(1024,activation="relu",kernel\_regularizer=l1(0.0001))(flat1)

batch1=BatchNormalization()(dense1)

dropout1=Dropout(0.5)(batch1)

dense2=Dense(1024,activation="relu",kernel\_regularizer=l1(0.0001))(dropout1)

batch2=BatchNormalization()(dense2)

dropout2=Dropout(0.5)(batch2)

den1 = Dense(64, activation='relu',kernel\_regularizer=l2(0.002))(dropout2)

den2 = Dense(7, activation='softmax')(den1)

model= Model(inputs=input1,outputs=den2)

# Early stopping callback tracking val\_loss

stop\_early = EarlyStopping(monitor='val\_loss', mode='min',

                           verbose=1, patience=15)

# Model Checkpoint callback tracking val\_accuracy

checkpoint = ModelCheckpoint(

    'model1.h5',

    monitor = 'val\_accuracy',

    verbose = 1,

    save\_best\_only = True

)

model.compile(optimizer='adam',loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=False),metrics=['accuracy'])

model

history = model.fit(X\_train, Y\_train, validation\_split=0.15, class\_weight = weight, batch\_size=32,epochs =15,callbacks=[checkpoint,stop\_early])

predict = model.predict(X\_test)

loss, accu = model.evaluate(X\_test,Y\_test,verbose=1)

labels\_pred = np.argmax(predict, axis = -1)

labels\_true = np.argmax(Y\_test, axis = -1)

model\_history(history)

3.8 Implementation of LSTM

from FeatureExtractor import get\_data

from plotutils import model\_history,c\_report,plot\_confusion\_matrix

import numpy as np

from tensorflow import keras

import sklearn

from keras.models import  Model

from keras.layers import \*

from keras.regularizers import \*

from keras.layers import BatchNormalization

from sklearn.metrics import classification\_report,accuracy\_score,confusion\_matrix

from keras.callbacks import \*

import numpy as np

import pandas as pd

# read data and call get\_data function to get splits of data

emodb\_data = pd.read\_csv('Downloads/Emodb\_dataset.csv')

from sklearn.preprocessing import LabelEncoder

label=LabelEncoder()

z=label.fit\_transform(emodb\_data['labels'])

z

emodb\_data['labels']=z

emodb\_data

# for 4 classes do some changes in feature extarctor file

X\_train, Y\_train, X\_test, Y\_test = get\_data()

Y\_train = keras.utils.to\_categorical(Y\_train,7)

Y\_test = keras.utils.to\_categorical(Y\_test,7)

Y=np.array(emodb\_data.labels)

from sklearn.utils.class\_weight import compute\_class\_weight

weight = compute\_class\_weight(class\_weight='balanced', classes=np.unique(Y), y=Y)

weight = {i : weight[i] for i in range(7)}

X\_train.shape

input1 = Input(shape=(X\_train.shape[1], X\_train.shape[2]))

lstm1 = Bidirectional(LSTM(512, return\_sequences = True,))(input1)

lstm2 = Bidirectional(LSTM(256, return\_sequences = True,))(lstm1)

flat1 = Flatten()(lstm2)

den1 = Dense(64, activation='relu',kernel\_regularizer=l2(0.002))(flat1)

den2 = Dense(7, activation='softmax')(den1)

model= Model(inputs=input1,outputs=den2)

# Early stopping callback tracking val\_loss

stop\_early = EarlyStopping(monitor='val\_loss', mode='min',

                           verbose=1, patience=15)

# Model Checkpoint callback tracking val\_accuracy

checkpoint = ModelCheckpoint(

    'model1.h5',

    monitor = 'val\_accuracy',

    verbose = 1,

    save\_best\_only = True

)

model.compile(optimizer='adam',loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=False),metrics=['accuracy'])

model

history = model.fit(X\_train, Y\_train, validation\_split=0.15, class\_weight = weight, batch\_size=128,epochs =15,callbacks=[checkpoint,stop\_early])

predict = model.predict(X\_test)

loss, accu = model.evaluate(X\_test,Y\_test,verbose=1)

labels\_pred = np.argmax(predict, axis = -1)

labels\_true = np.argmax(Y\_test, axis = -1)

model\_history(history)

plot\_confusion\_matrix(cm = confusion\_matrix(labels\_true, labels\_pred),normalize = True,

                    target\_names = ['Angry','Happy', 'Neutral', 'Sad'],title = "Confusion Matrix")

3.9 Implementation of merging of CNN,DNN,LSTM

from FeatureExtractor import get\_data

from plotutils import model\_history,c\_report,plot\_confusion\_matrix

import numpy as np

from tensorflow import keras

import sklearn

from keras.models import  Model

from keras.layers import \*

from keras.regularizers import \*

from keras.layers import BatchNormalization

from sklearn.metrics import classification\_report,accuracy\_score,confusion\_matrix

from keras.callbacks import \*

import numpy as np

import pandas as pd

# read data and call get\_data function to get splits of data

emodb\_data = pd.read\_csv('Downloads/Emodb\_dataset.csv')

from sklearn.preprocessing import LabelEncoder

label=LabelEncoder()

z=label.fit\_transform(emodb\_data['labels'])

emodb\_data['labels']=z

X\_train, Y\_train, X\_test, Y\_test = get\_data()

Y\_train = keras.utils.to\_categorical(Y\_train,7)

Y\_test = keras.utils.to\_categorical(Y\_test,7)

Y=np.array(emodb\_data.labels)

from sklearn.utils.class\_weight import compute\_class\_weight

weight = compute\_class\_weight(class\_weight='balanced', classes=np.unique(Y), y=Y)

weight = {i : weight[i] for i in range(4)}

#CNNModel

input1 = Input(shape=(X\_train.shape[1], X\_train.shape[2], 1))

# First Conv2D block

conv1 = Conv2D(100, kernel\_size=(3, 3),strides=(2, 2),activation=None,padding='same',

               kernel\_initializer='he\_normal', kernel\_regularizer=l2(0.002), bias\_regularizer=l2(0.001))(input1)

batch1 = BatchNormalization()(conv1)

elu1 = ELU(alpha=1.0)(batch1)

pool1 = MaxPooling2D(pool\_size=(2, 2),strides=(1,1))(elu1)

dropout1 = Dropout(0.5)(pool1)

# Second Conv2D block

conv2 = Conv2D(100, kernel\_size=(3, 3),strides=(2, 2),padding='same',

               activation=None,kernel\_regularizer=l2(0.001), bias\_regularizer=l2(0.001) )(dropout1)

batch2 = BatchNormalization()(conv2)

elu2 = ELU(alpha=1.0)(batch2)

pool2 = MaxPooling2D(pool\_size=(2,2),strides=(1,1))(elu2)

dropout2 = Dropout(0.5)(pool2)

conv3 = Conv2D(100, kernel\_size=(3, 3),strides=(2, 2),activation=None,padding='same',

               kernel\_regularizer=l2(0.002), bias\_regularizer=l2(0.001))(dropout2)

batch3 = BatchNormalization()(conv3)

elu3 = ELU(alpha=1.0)(batch3)

pool3 = MaxPooling2D(pool\_size=(2, 2),strides=(1,1))(elu3)

dropout3 = Dropout(0.5)(pool3)

conv4 = Conv2D(100, kernel\_size=(3, 3),strides=(2, 2),activation=None,padding='same',

                kernel\_regularizer=l2(0.002), bias\_regularizer=l2(0.001))(dropout3)

batch4 = BatchNormalization()(conv4)

elu4 = ELU(alpha=1.0)(batch4)

pool4 = MaxPooling2D(pool\_size=(2, 2),strides=(1,1))(elu4)

dropout4 = Dropout(0.5)(pool4)

#DNNModel

flat1=Flatten()(dropout4)

dense1=Dense(1024,activation="relu",kernel\_regularizer=l1(0.0001))(flat1)

batch5=BatchNormalization()(dense1)

dropout5=Dropout(0.5)(batch5)

dense2=Dense(1024,activation="relu",kernel\_regularizer=l1(0.0001))(dropout5)

batch6=BatchNormalization()(dense2)

dropout6=Dropout(0.5)(batch6)

#LSTMModel

reshape1 = Reshape((4\*8, 32))(dropout6)

lstm1 = Bidirectional(LSTM(512, return\_sequences = True,))(reshape1)

flat1 = Flatten()(lstm1)

lstm2 = Bidirectional(LSTM(256, return\_sequences = True,))(reshape1)

flat2 = Flatten()(lstm2)

den1 = Dense(64, activation='relu',kernel\_regularizer=l2(0.002))(flat1)

den2 = Dense(7, activation='softmax')(den1)

model= Model(inputs=input1,outputs=den2)

# Early stopping callback tracking val\_loss

stop\_early = EarlyStopping(monitor='val\_loss', mode='min',

                           verbose=1, patience=150)

# Model Checkpoint callback tracking val\_accuracy

checkpoint = ModelCheckpoint(

    'model1.h5',

    monitor = 'val\_accuracy',

    verbose = 1,

    save\_best\_only = True

)

model.compile(optimizer='adam',loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=False),metrics=['accuracy'])

model

import matplotlib.pyplot as plt

plt.plot(history.history['val\_accuracy'])

plt.plot(history.history['accuracy'])

plt.title("Model accuracy")

plt.xlabel("Accuracy")

plt.ylabel("Epochs")

plt.show()

import matplotlib.pyplot as plt

plt.plot(history.history['val\_loss'])

plt.plot(history.history['loss'])

plt.title("Model loss")

plt.xlabel("loss")

plt.ylabel("Epochs")

plt.show()

predict = model.predict(X\_test)

loss, accu = model.evaluate(X\_test,Y\_test,verbose=1)

labels\_pred = np.argmax(predict, axis = -1)

labels\_true = np.argmax(Y\_test, axis = -1)

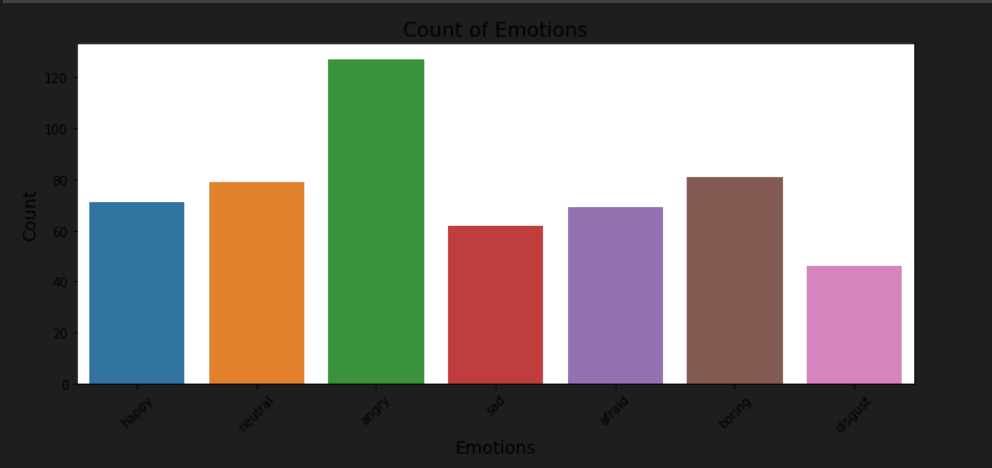
model\_history(history)

plot\_confusion\_matrix(cm = confusion\_matrix(labels\_true, labels\_pred),normalize = True,

                    target\_names =['Angry','Happy', 'Neutral', 'Sad'],title = "Confusion Matrix")

**CHAPTER - 4 SNAPSHOTS**

Fig.4.1 Value Count of Emotions and plots of EMODB



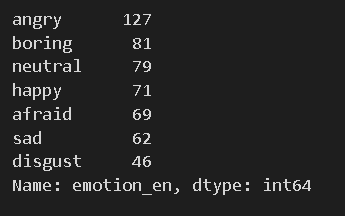


Fig. 4.2 Waveplots of Waveform, Power Spectrum, Spectrogram and MFCC

Fig 4.2.1 Waveform

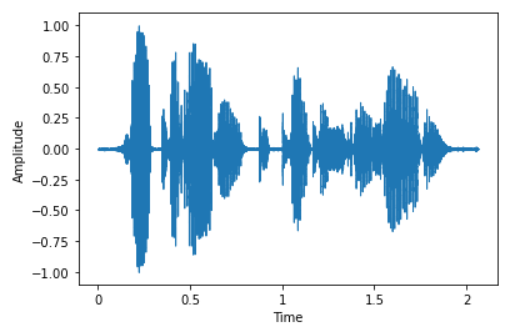


Fig.4.2.2 Power Spectrum

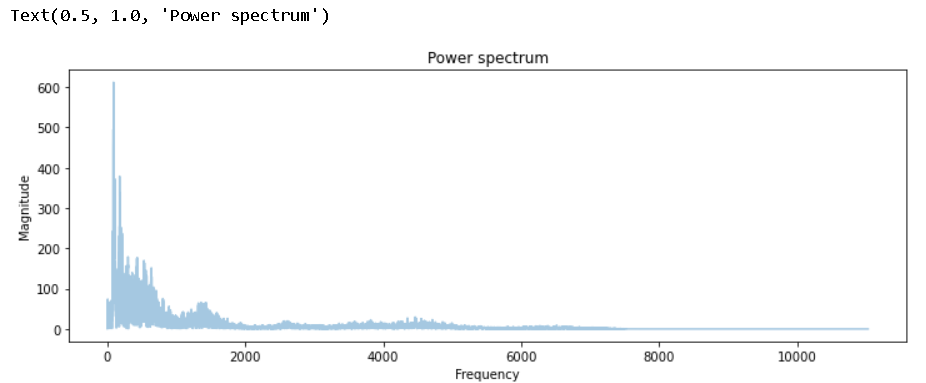
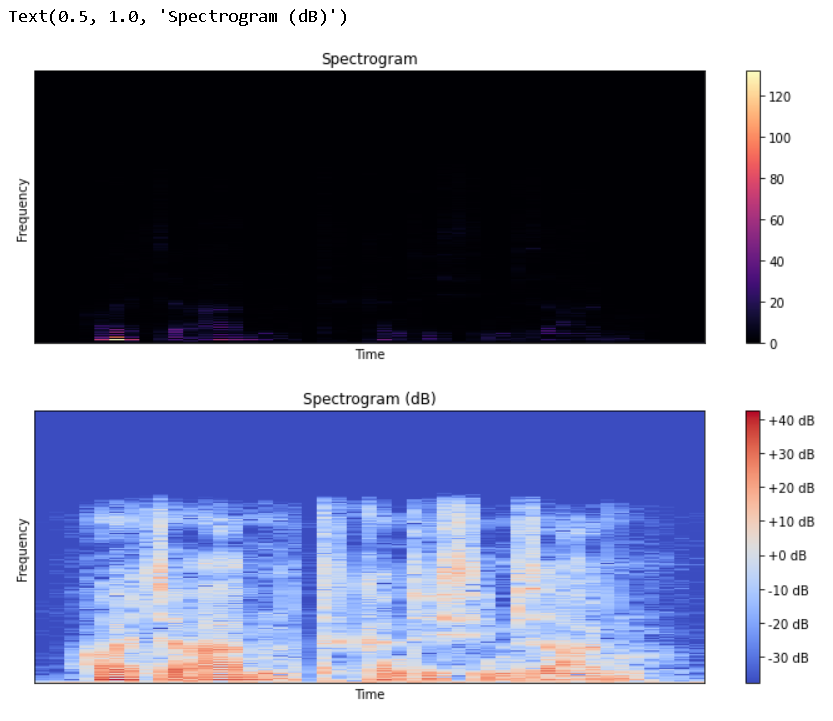


Fig. 4.2.3 Spectrogram



4.2.4 MFCC

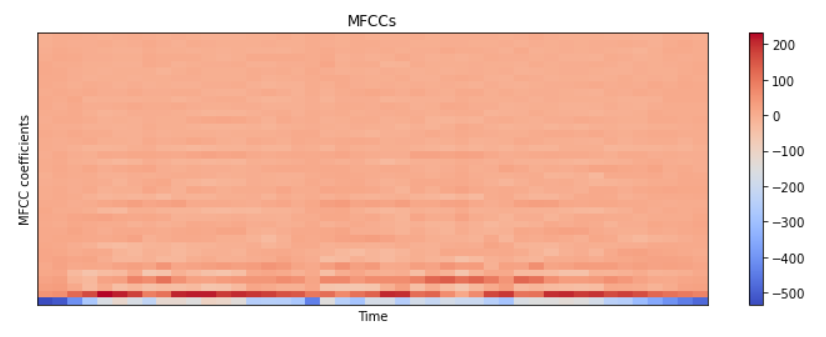


Fig.4.3 Extracted values using Mean, Max and Min

Fig.4.3.1 Through mean

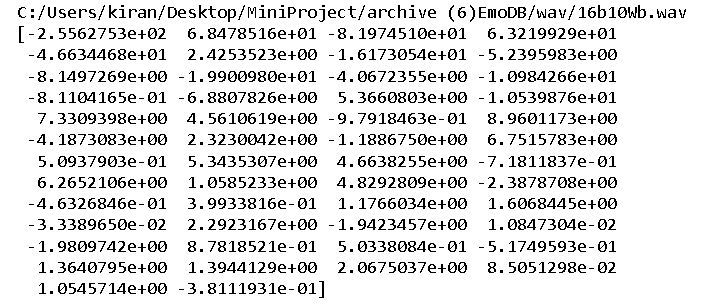


Fig.4.3.2 Through max

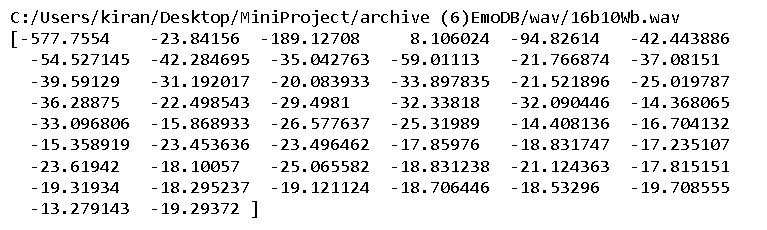
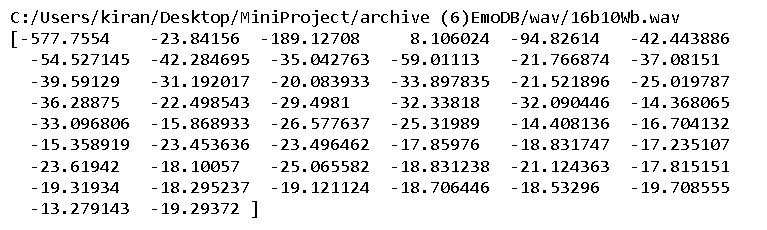
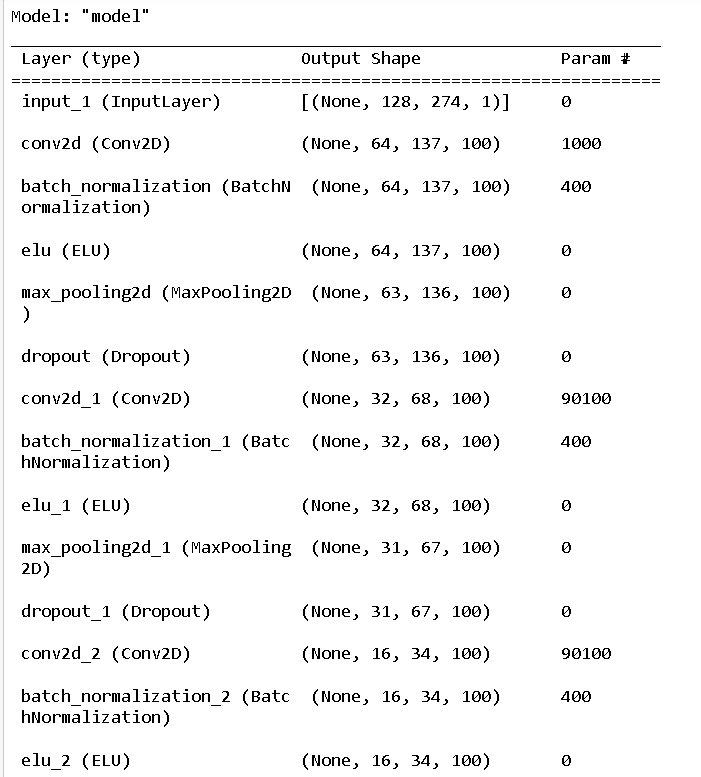


Fig.4.3.3 Through min



4.4 CNN model

4.4.1 Summary of the CNN architecture



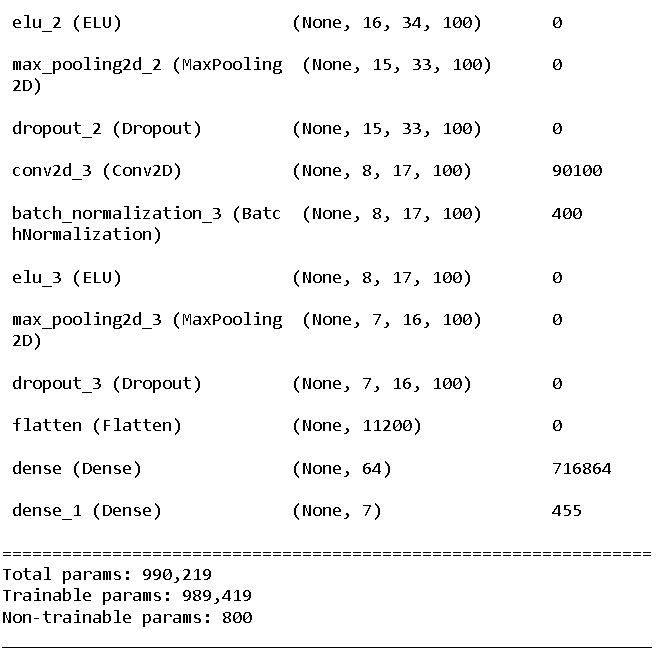


Fig. 4.4.2 Accuracy of the CNN Architecture

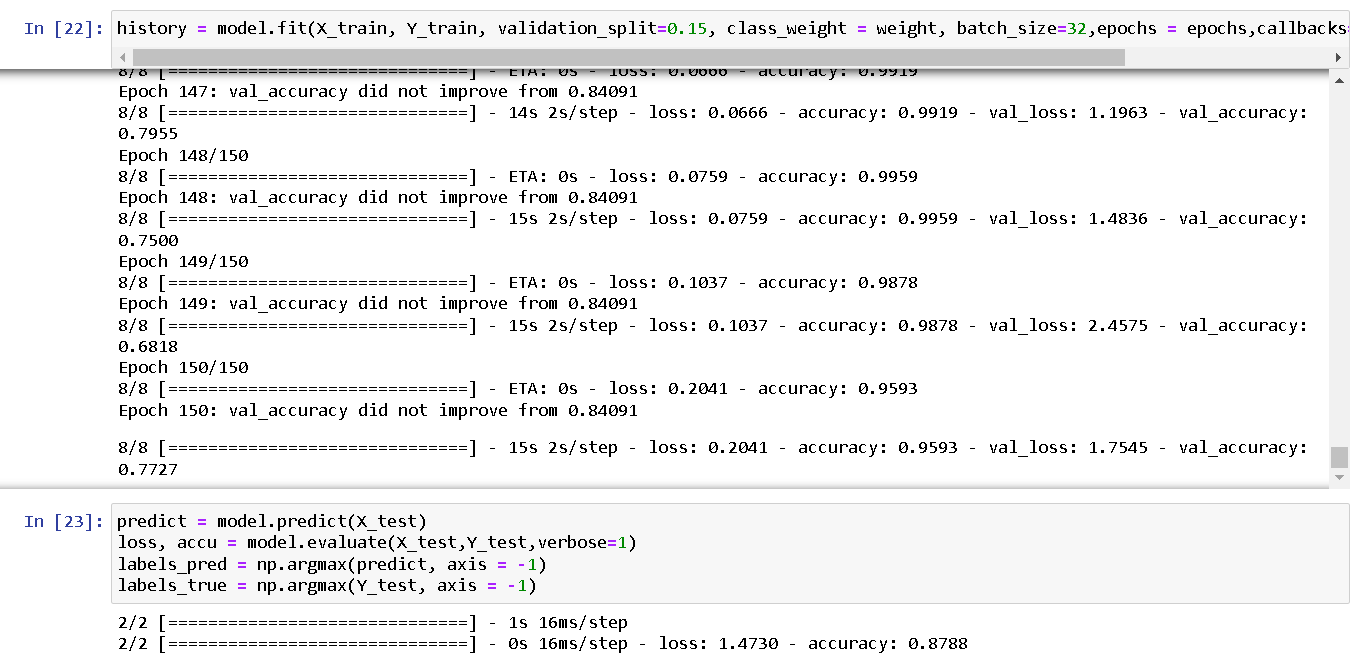
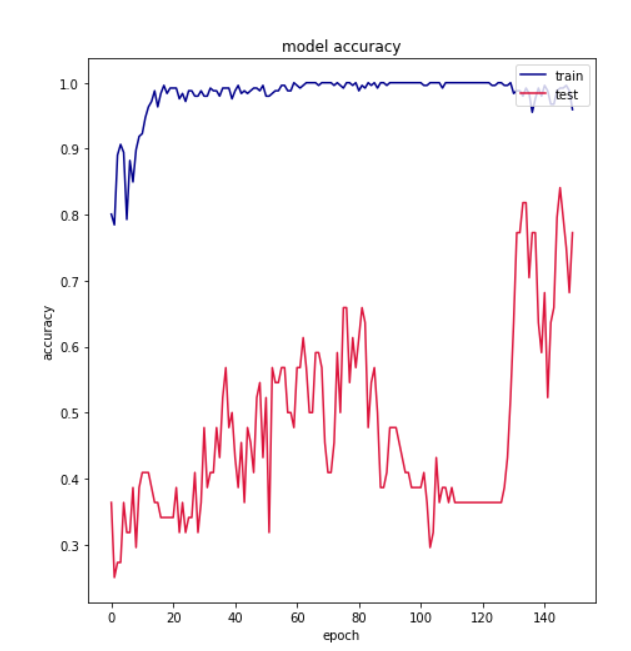


Fig. 4.4.3 Visualization of Model Accuracy and Model Loss Graph



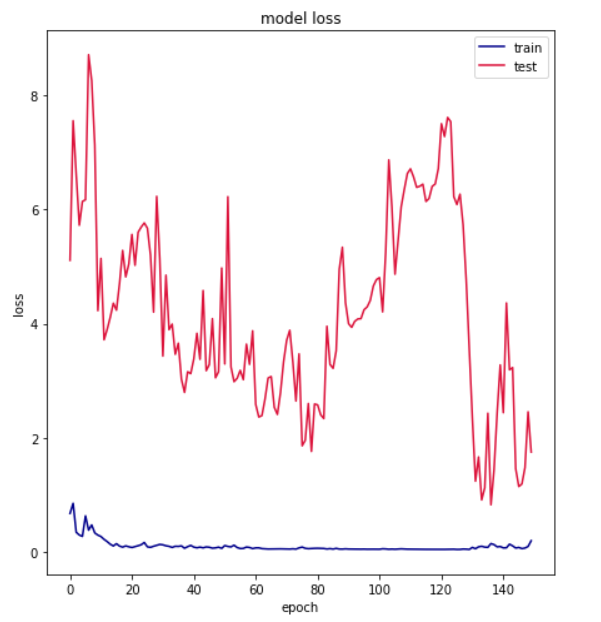


Fig.4.4.4 Confusion matrix of the CNN Architecture

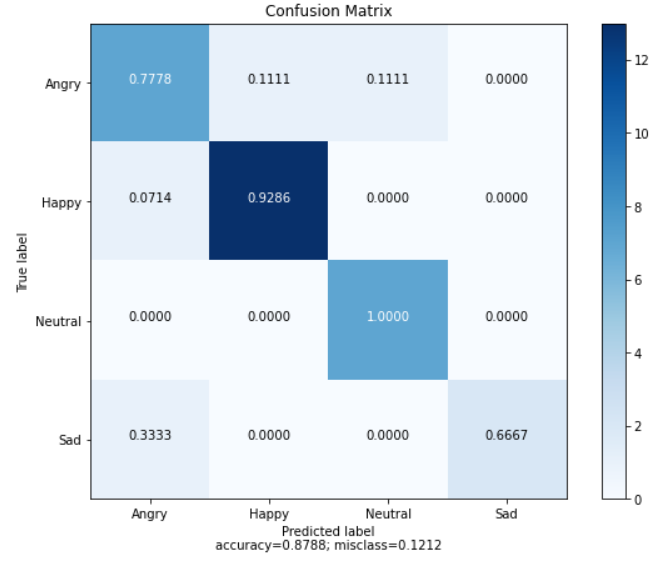


Fig. 4.5 DNN Model

Fig. 4.5.1 Summary of the DNN Architecture

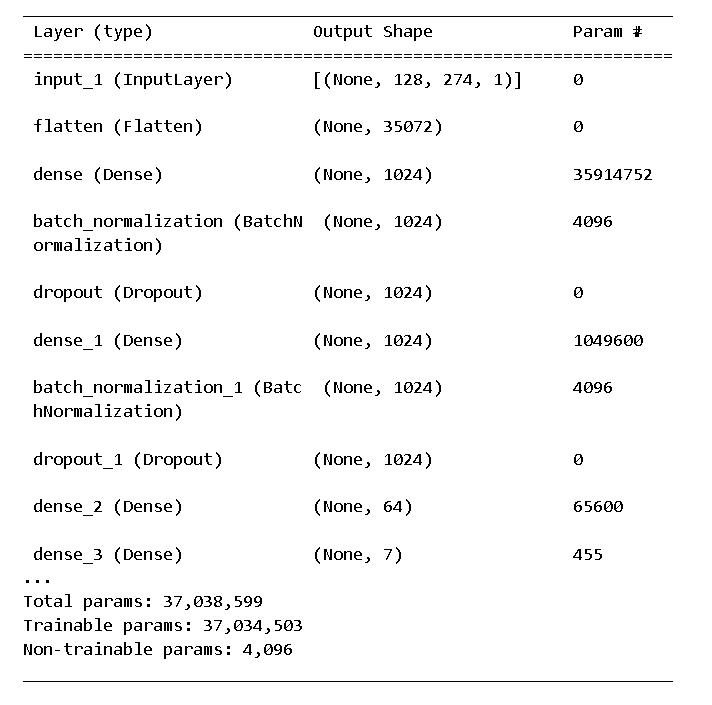


Fig.4.4.2 Accuracy of the DNN Architecture

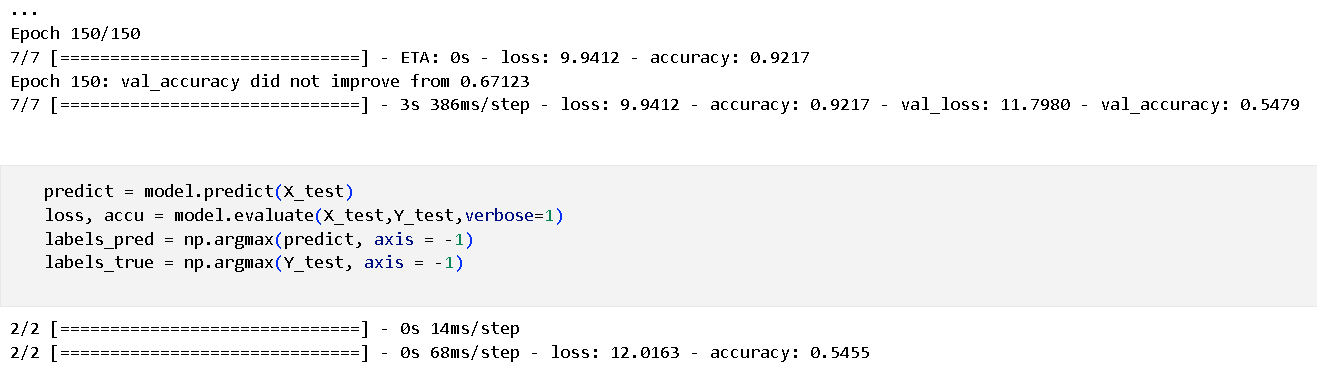


Fig. 4.4.3 Visualization of Model Accuracy and Model Loss

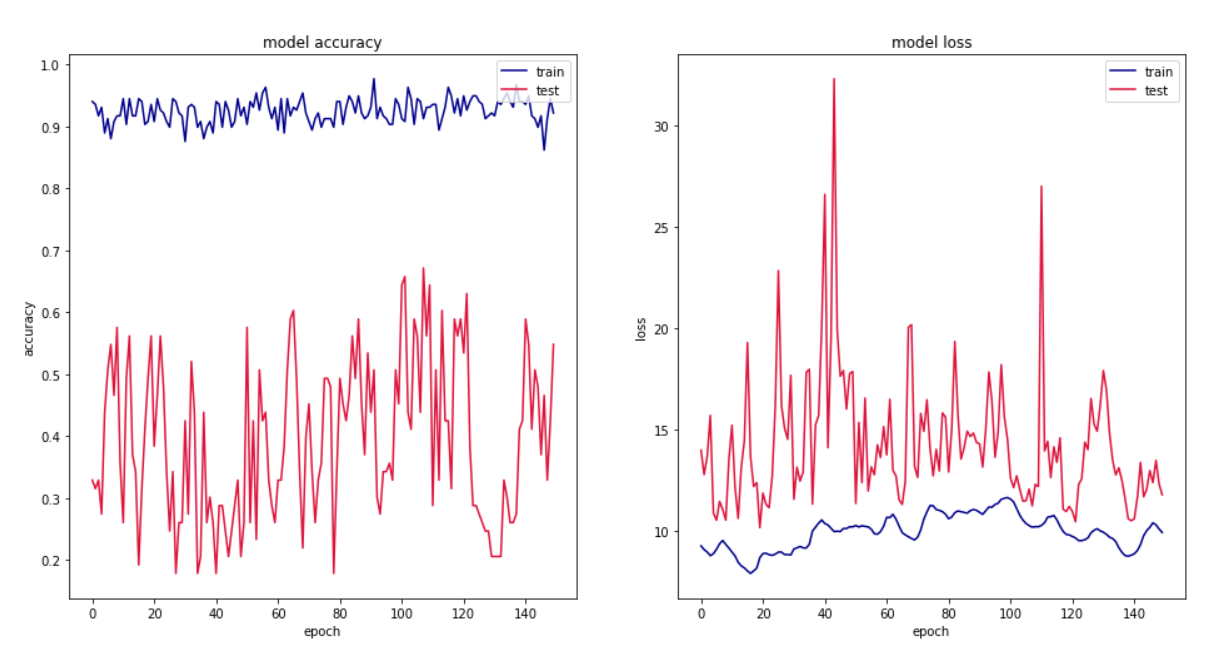


Fig.4.4.4 Confusion matrix of the DNN Architecture

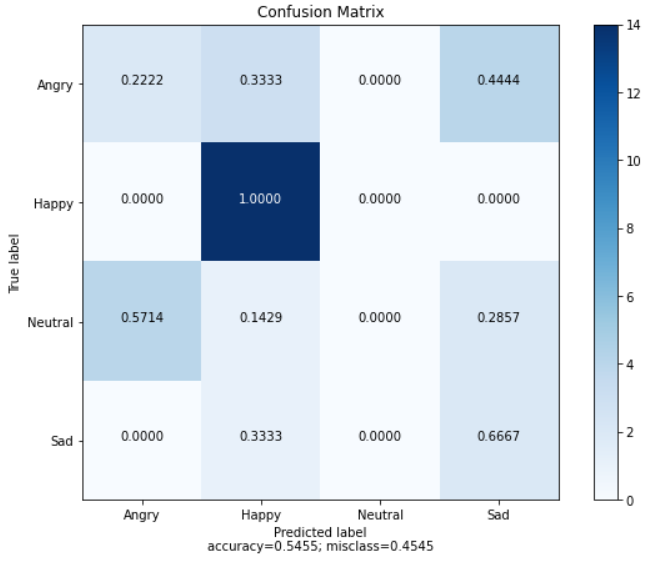


Fig.4.5 LSTM Architecture

Fig. 4.5.1 Summary of the LSTM Architecture

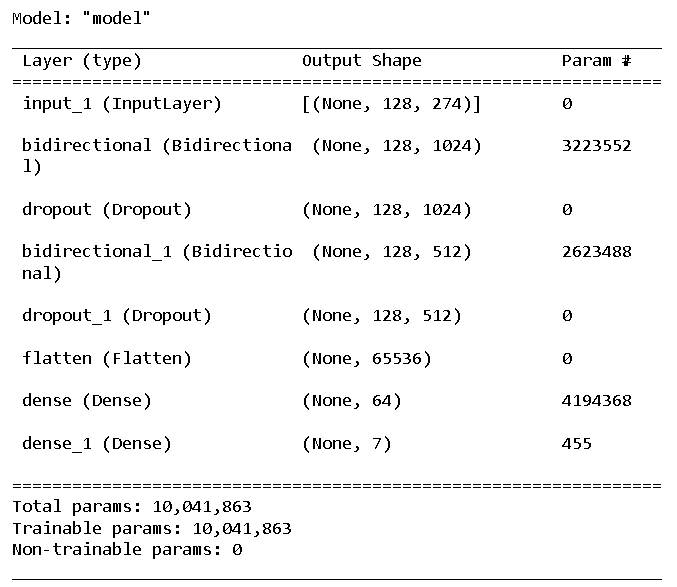


Fig.4.5.2 Accuracy of the LSTM Architecture

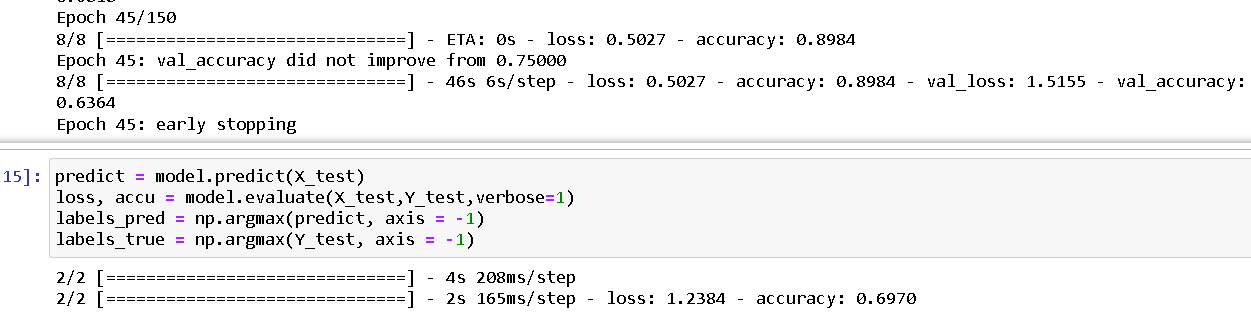


Fig. 4.5.3 Visualization of Model Accuracy and Model Loss

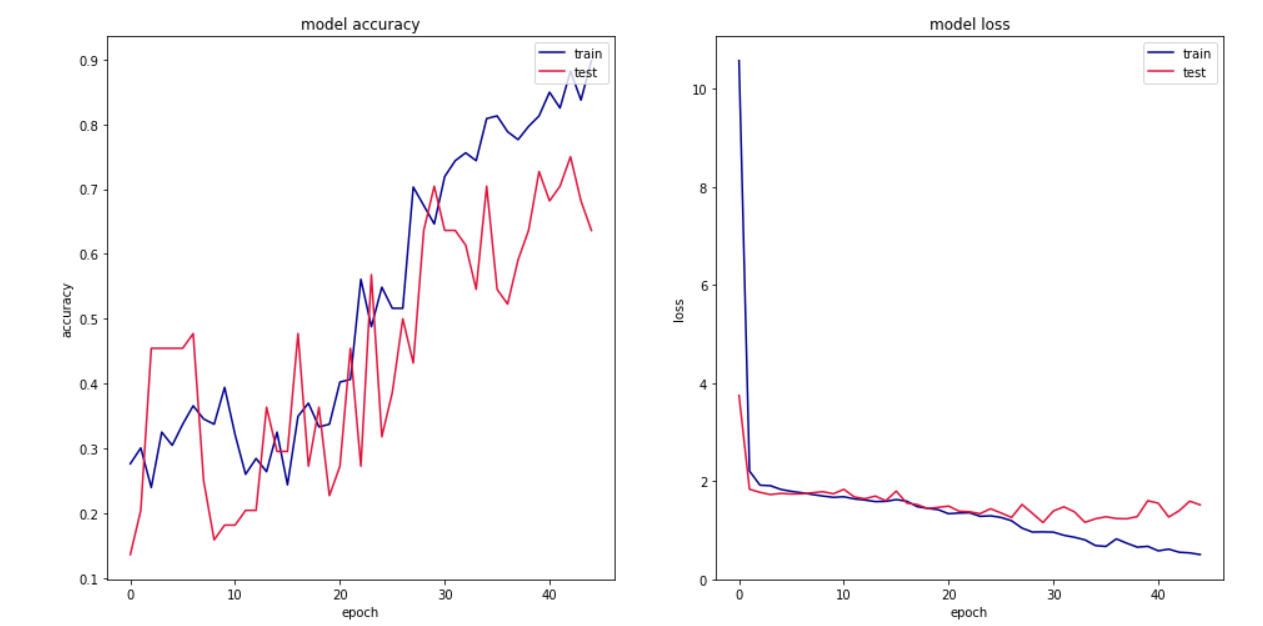


Fig. 4.5.4 Confusion Matrix of the LSTM Architecture

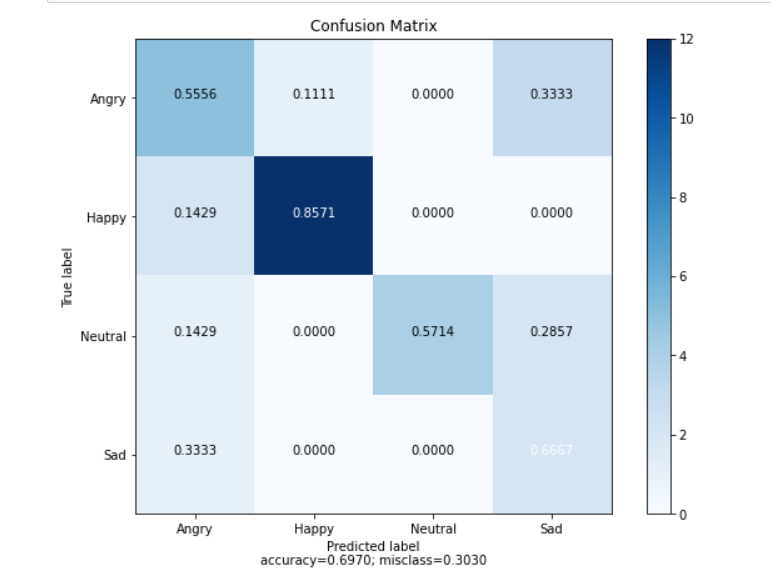
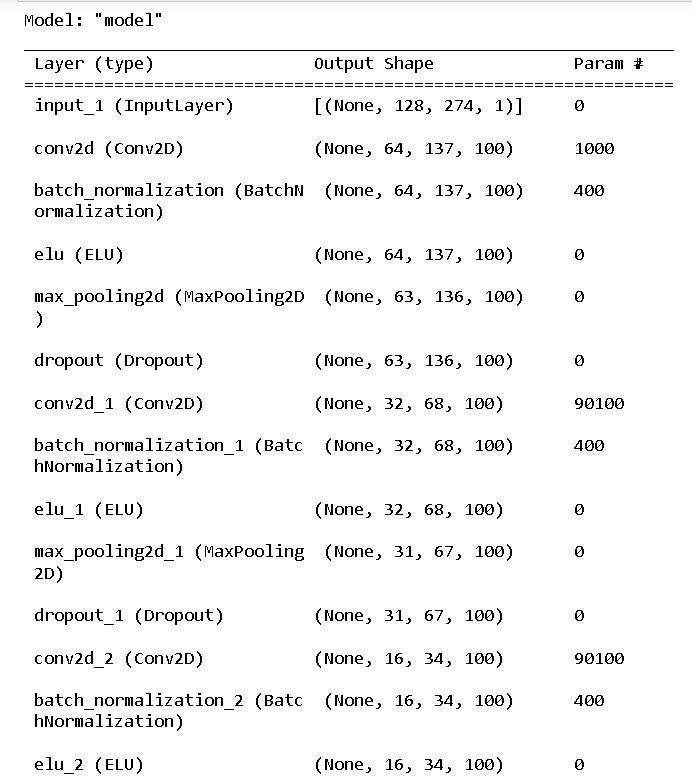
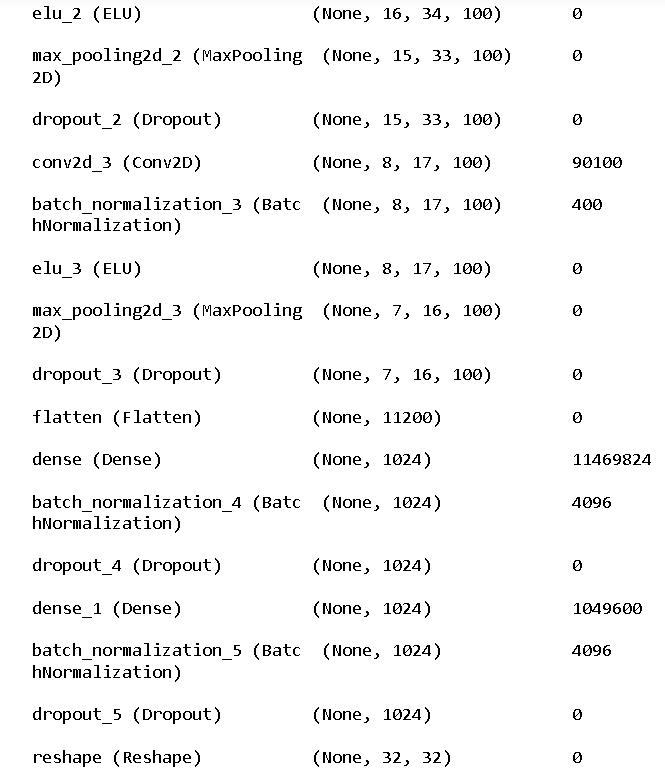


Fig. 4.6 Combined model of CNN, DNN and LSTM

Fig.4.6.1 Summary of the architecture  




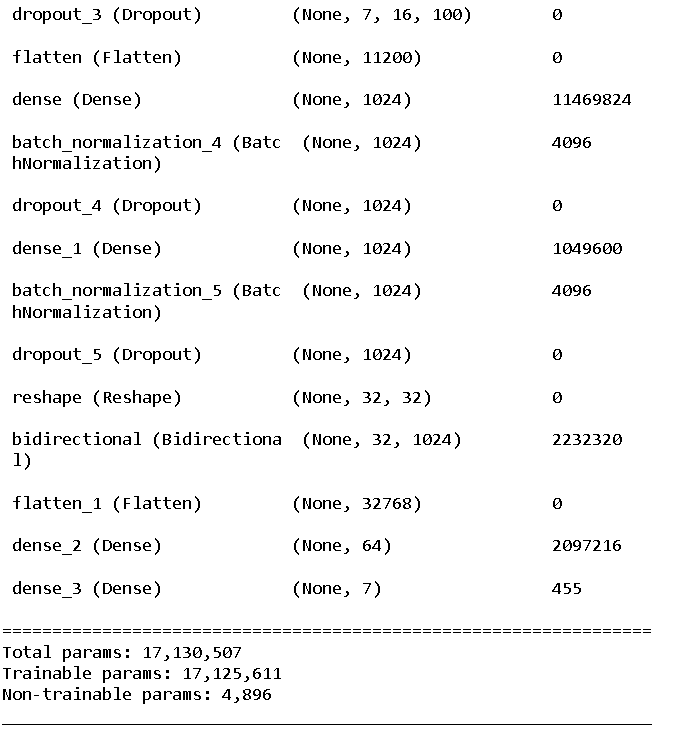


Fig. 4.6.2 Accuracy of the merged architecture

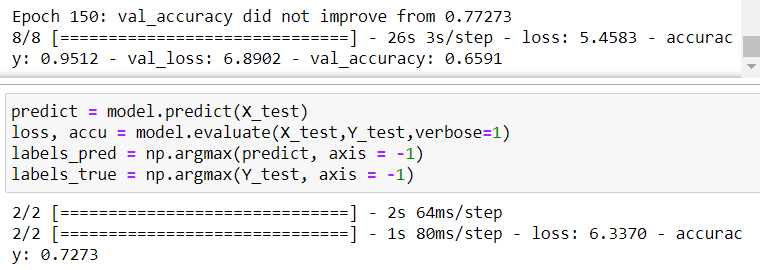


Fig.4.6.3 Visualization of the Model Accuracy and Model Loss

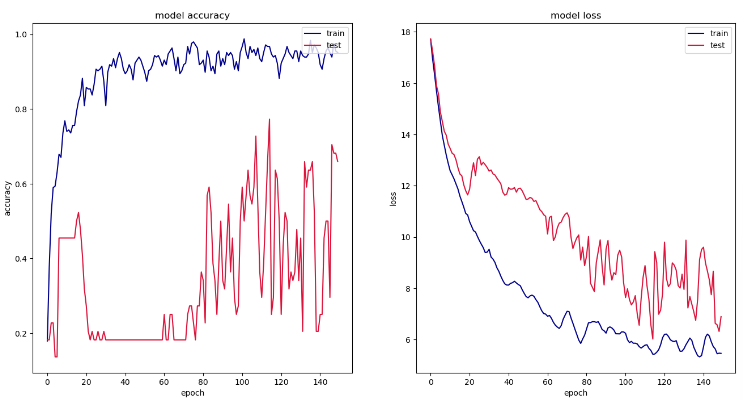


Fig.4.6.4 Confusion Matrix of Merged Architecture



**CHAPTER - 5**

**CONCLUSION AND FUTURE PLANS**

**Conclusions:**

The emotional information contained in different forms of speech features is different and complementary. Three dimensions features of speech are combined by separating strategy to obtain the higher performance. HD-MFM utilizes the complementarity of speech features from three different dimensions, image, temporal and statistical speech feature dimension, and combines the respective characteristics of three different neural networks, CNN, DNN and LSTM to obtain a high-dimensional speech feature. The performance of the HD-MFM is tested on EMODB database. CNN model obtains accuracy of 87.88%, DNN model obtains accuracy of 54.55%, LSTM model obtains accuracy of 69.70%. The experiments show that the HD-MFM can learn distinguishing features and provide more accurate predictions compared with those three original blocks. After merging all the three blocks the accuracy obtained is 72.73%.

**Future Plans:**

In future, we plan to explore the more effective ways to combine the information that was extracted from the speech. Besides that we also consider the usage of the different types speech features and also use the different speech algorithms in order to increase the accuracy of the speech emotion recognition.

**CHAPTER - 6**

**REFERENCES**

[1] B. Huebner, S. Dwyer, M. Hauser, The role of emotion in moral psychology, Trends Cogn. Sci. 13 (1) (2009) 1–6.

[2] S.L. Koole, The psychology of emotion regulation: An integrative review, Cogn. Emot. 23 (1) (2009) 4–41.

[3] P.M. Niedenthal, F. Ric, Psychology of Emotion, Psychology Press, 2017.

[4] S. Ramakrishnan, I.M.M.E. Emary, Speech emotion recognition approaches in human computer interaction, Telecommun. Syst. 52 (3) (2013) 1467–1478.

[5] M.S. Fahad, A. Ranjan, J. Yadav, A. Deepak, A survey of speech emotion recognition in natural environment, Digit. Signal Process. (2020) 102951.

[6] S. Li, X. Xing, W. Fan, B. Cai, P. Fordson, X. Xu, Spatiotemporal and frequential cascaded attention networks for speech emotion recognition, Neurocomputing 448 (2021) 238–248

[7] R. Khokale, A.R. Panat, Y.H. Gulhane, Analysis of affective speech for fatigue detection, in: Proc. ICWET ’10 Int. Conf. & Work. Emerg. Trends Technol. Mumbai, Maharashtra, India, 2010, pp. 237–240.

[8] J. Zhao, X. Mao, L. Chen, Speech emotion recognition using deep 1D & 2D CNN LSTM networks, Biomed. Signal Proc. Control 47 (2019) 312–323.

[9] K. Huang, C. Wu, T. Yang, M. Su, J. Chou, Speech emotion recognition using autoencoder bottleneck features and LSTM, in: Int. Conf. Orange Technol., 2016, pp. 1–4.

[10] Y. Sun, G. Wen, J. Wang, Weighted spectral features based on local Hu moments for speech emotion recognition, Biomed. Signal Proc. Control 18 (2015) 80–90.

**CHAPTER -7**

**Appendix -Base Paper**

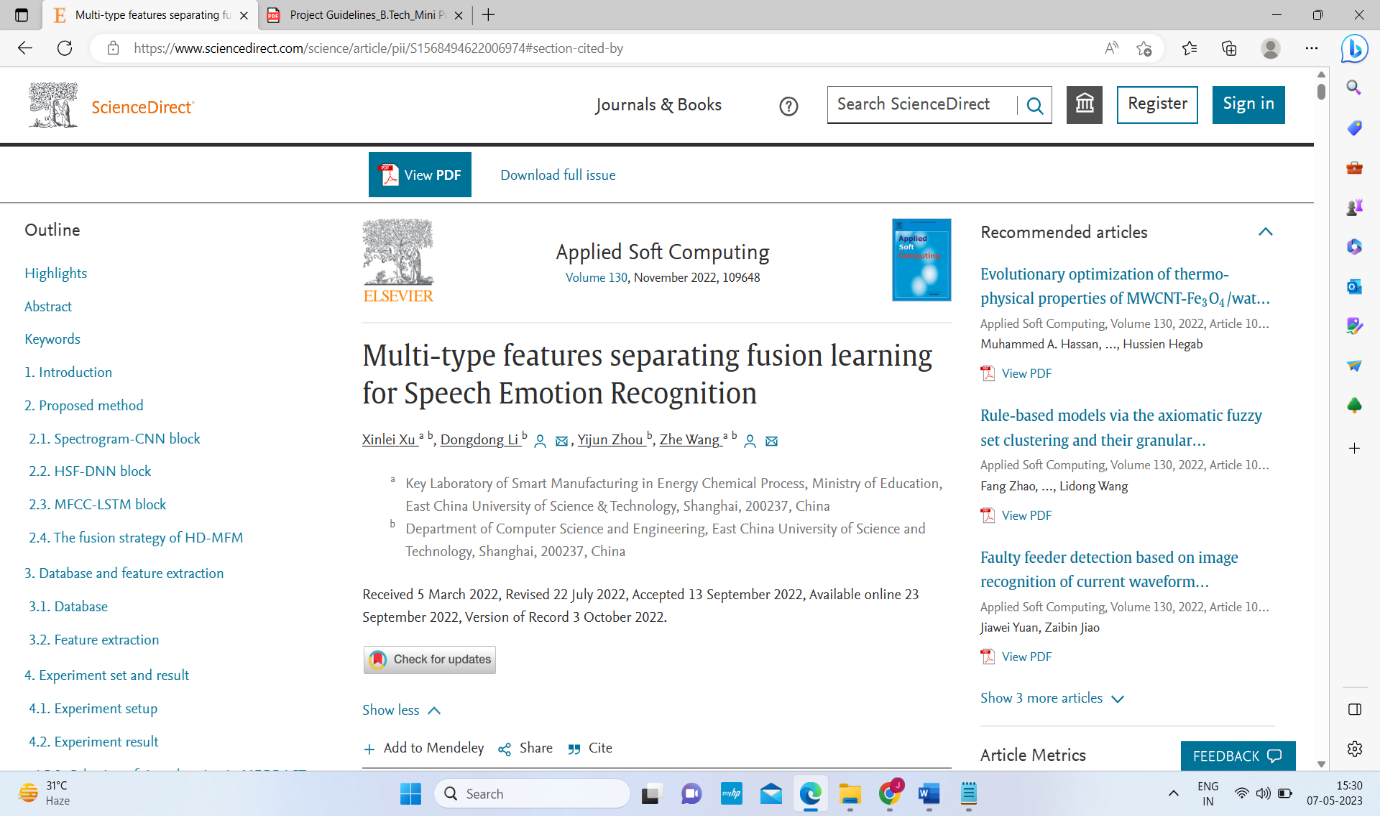
****

Fig. 7.1. Screenshot of Indexing