Fusion of Deep Learning Features with Mixture of Brain Emotional Learning for Audio-Visual Emotion Recognition

Report submitted to SASTRA Deemed to be University
As per the requirement for the course

INT300 : MINI PROJECT

Submitted by

Samatha A
(Reg.No:125015103, B. Tech Information Technology)
Shahana S
(Reg.No:125015111, B. Tech Information Technology)
Srividhya S
(Reg.No:125015120, B. Tech Information Technology)

May 2024

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act,1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled **"Fusion of deep learning features with mixture of brain emotional learning for audio-visual emotion recognition"** submitted as a requirement for the course, **INT300 : MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Ms. Samatha A (Reg.No:125015103, B. Tech Information Technology)**,**Ms. Shahana S (Reg.No:125015111, B. Tech Information Technology)** and **Ms. Srividhya S (Reg.No:125015120, B. Tech Information Technology**) during the academic year 2023-24, in the School of Computing, under my supervision.

Signature of Project Supervisor : A. Emily Jenifer

Name with Affiliation : Dr. A Emily Jenifer, AP III, School of Computing

Date : 19/04/2024

Mini Project *Viva voice* 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 my/our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah**, Associate Dean, Research, **Dr. K.Ramkumar,** Associate Dean ,Academics**, Dr. D. Manivannan**, Associate Dean, Infrastructure,
Dr. R. Alageswaran, Associate Dean, Students Welfare.

Our guide **Dr. A Emily Jenifer**, AP III, School of Computing was the driving force behind this whole idea from the start. Her 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 this project.

List of Figures

Abbreviations

3D-CNN	Three Dimensional Convolutional Neural Network
CRNN	Convolutional Recurrent Neural Network
MOBEL	Mixture of Brain Emotion Learning
ReLU	Rectified Linear Unit
SSD	Single Shot Detector
AMG	Amygdala
OFC	Orbito-Frontal Cortex
tansig	Tangent Sigmoid Function

Notations

English Symbols (in alphabetical order)

E	Bel Expert
G	Gating Network

Greek Symbols (in alphabetical order)

α	learning rate of AMG
β	learning rate of OFC
γ	degradation rate
Σ	Summation

ABSTRACT

Multimodal emotion recognition, particularly in the domains of speech and facial expressions, has garnered significant interest in various applications such as e-learning , human-computer interaction, health monitoring , mobile computing and gaming .Multimodal emotion recognition is one of the main challenging tasks due to the multimodality characteristic of human emotional expression. In response to these challenges, this paper introduces an innovative approach—a fusion model combining deep learning features with a Mixture of Brain Emotional Learning (MoBEL) model inspired by the brain's limbic system.

The proposed model addresses the spatial-temporal correlations in video content, recognizing the intricate patterns of emotion expressed through both audio and visual cues.Our methodology involves two primary stages: first, leveraging advanced deep learning techniques, including Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN), to extract highly abstract features from audio and visual data. Second, the introduction of the MoBEL model, which concurrently learns joint audio-visual features by simulating the intricate emotional processing observed in the brain's limbic system. For visual modality representation, a 3D-CNN model captures spatial-temporal features of visual expressions, while Mel-spectrograms of speech signals undergo processing through a CNN-RNN architecture for spatial-temporal feature extraction in the auditory modality. The proposed high-level feature fusion with the MoBEL network aims to exploit the correlation between visual and auditory modalities, thereby enhancing the accuracy of emotion recognition that detects 7 basic emotions with around 90% accuracy.

Our model training was conducted on the Zenodo RAVDESS dataset. Through this work, we contribute to the ongoing discourse on the challenges of multimodal emotion recognition and emphasize the critical importance of effective feature extraction and integration, especially when considering spatial-temporal correlations.

KEY WORDS: MOBEL, Temporal features, Audio and Visual Modality

Table of Contents

CHAPTER 1
SUMMARY OF THE BASE PAPER

Title	**:**	Fusion of deep learning features with mixture of Brain emotional learning for audio-visual emotion recognition
Publisher	**:**	Speech Communication
Year	**:**	2021
Journal name	**:**	Elsevier
DOI	**:**	10.1016/j.specom.2020.12.001
Base paper URL	**:**	https://www.sciencedirect.com/science/article/pii/S0167639320303058?casa_token=qRPehxOqJHIAAAAA:LNUr5qGH5J96MvBh2faGPnBCTFbqnFFAiR6FQZE9k-V7ZU_gub8X6B7-thwkbXI4FG7KbHbhZA

The main contributions of the base paper are:

- Extracting Features from Audio and Visual Modality
- Finding Spatial-Temporal Correlations between Audio and Visual Modality
- Proposing MOBEL architecture.

Our method consists of 3 major steps:

1.1 VISUAL AND AUDIO PREPROCESSING

The video received was split into two streams Audio and Visual stream. The streams were preprocessed separately .

1.1.1 VISUAL STREAM

The Visual stream consists only the video without any audio. 12 key frames were generated from this Video based on a specific frame rate with interval 8ms. After extracting key frames each video had 12 key images . These images were then passed into a SSD generator ,a pre trained model trained on CAFFEE dataset. The output of SSD model is a co-ordinate of face bounding boxes. Using these bounding boxes we crop our RGB key images to knock-off the background and focus on the face and resize each image with fixed size of (96*100*3). Then specific number of keyframes has to be taken from the frames which are generated from the video data. For our data we have taken 12 keyframes .First and last frames has to be removed if the total number of keyframes exceeds 12 . similarly first and last frames has to be repeated if total number of keyframes less than 12 .Thereby ensuring that there are fixed number of keyframes for each sample video in the dataset

1.1.2 AUDIO STREAM

In the audio stream , the audio signal is segmented into small segments of 1000ms and 500ms overlapping. These segments are then converted into Mel-spectrogram images which is passed into the CRNN model. Each Audio was segmented into 8 further segments ,the parameters were fixed based on trial and error method. If number of segments generated exceeded 12 it was deleted. For a given audio we adopted 96 Mel-filter banks with a frequency range of 20 to 8000 Hz and a context window of 32 frames.
The audio signals were converted into Mel-spectrograms based on parameters such as sampling rate of 22058 ,FFT length=2048, hop length=512,no. of mels=96, minimum frequency of 20Hz and maximum frequency of 8000Hz , with a padding of 5 secs.
The spectrogram was also normalized and scaled before saving.

1.2. FEATURE EXTRACTION

Feature extraction is also done separately for the streams.

1.2.1. VISUAL FEATURES

The generated 12 SSD frames for each visual stream data with size (12*96*100*3) are fed into the 3D-CNN network for training which learns the spatial-temporal features from the video directly .In proposed method ,ReLU activation function is used between the hidden layers and oftmax activation function is used in the output layer .The 3D-CNN model learns the spatial-temporal features from the visual data and 256 features are extracted from the last Fully Connected layer with 256 neurons and passed to MOBEL model for Feature level fusion. The results from the softmax layer is passed to the BEL model for decision level fusion. .Drop-Out layer was included to prevent overfitting. 10 fold Cross validation was used to improve test accuracy.

1.2.2. AUDIO FEATURES

The generated Mel-spectrogram is passed into a CRNN model as an image of size (12*96*32*1) .The CRNN consists of a 3D-CNN and LSTM network. The CNN network architecture, including the number of layers, number of filters in the convolutional layer and max-pooling filter sizes is the same as visual stream emotion recognition. In the LSTM part, we have a layer with 256 cells. The last FC layer with a size of 256 is extracted as a speech emotion representation. Afterward, audio and visual learning emotional features of each stream are combined and fed to the MOBEL network model to train jointly spatial-temporal information of different modalities. The output of the softmax layer is passed to decision level fusion model. Batch normalization was implemented with a mini batch size of 12. 10 fold cross validation was implemented to improve test accuracy.

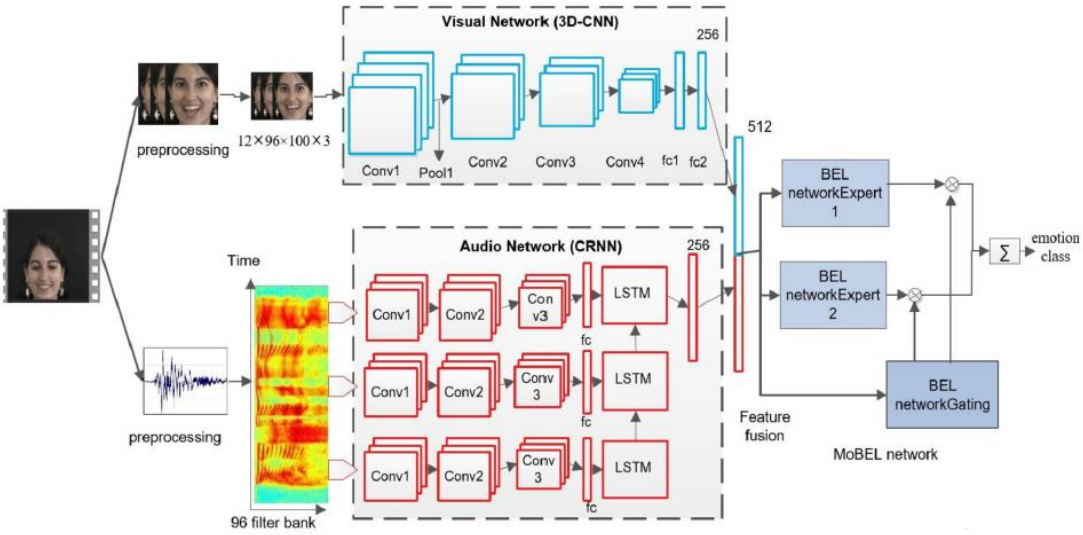


Fig 1.2 Architecture of proposed model for audio-visual emotion recognition using 3D-CNN and CRNN and then high level feature fusion with MoBEL [1]

1.3. DECISION LEVEL FUSION

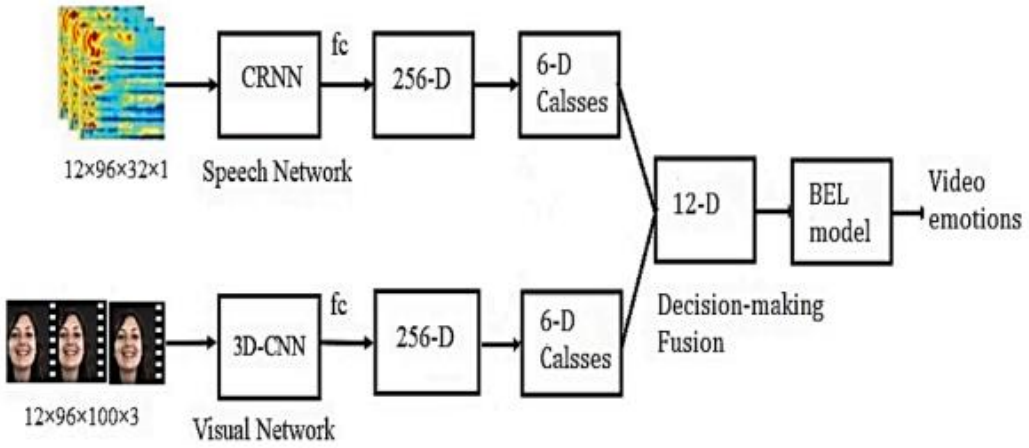


Fig 1.3. The architecture of decision-level fusion with the BEL model [1]

The features extracted from the softmax layer of Audio(6D) and Visual networks(6D) are fused to get 12 features for a single input file. These features are probabilities of the input file belonging to a particular emotion class out of the 6 classes. These are passed into the BEL model. The BEL model consists of two components AMG and OFC. The BEL model's parameters are trained using a Gradient descent algorithm which provides supervised training.

$$e_a = \sum_{j=1}^{n} v_j p_j + v_{n+1} p_{th}$$

$$e_o = \sum_{j=1}^{n} w_j p_j$$

$$E_i = f_e\left(f_{amg}(e_a) - f_{ofc}(e_o)\right)$$

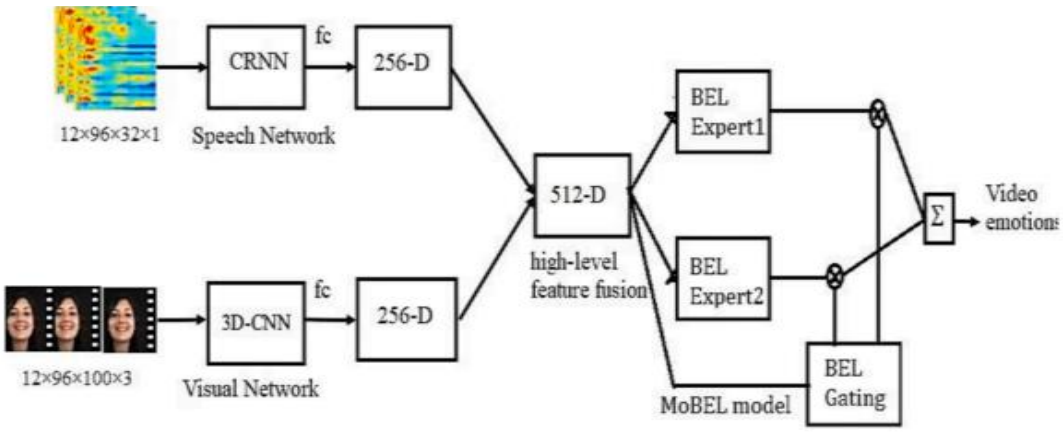
$$\Delta v_{ij} = -\gamma v_{ij} + \alpha_e p_j \max\left(0, t - f_{amg}(e_a)\right)$$

$$\Delta w_{ij} = \beta_e p_j (E_i - t) \qquad \textit{for i=1,2 for j=1,2,3..n}$$

$$e = \sum_{i} ||t - E||^2$$

Where e_a and e_o are the outputs of AMG and OFC. Here $f()$ is the activation function which is tansig. The optimal weights are decided based on the error function e .We take the optimal weights and predict the Emotion. The weights and features are multiplied as per 'product rule' and the emotion class with the maximum multiplied score is given as output. Other ensemble methods along with product rule was experimented and product rule was observed to yield the maximum performance of 74.3%

1.4. FEATURE LEVEL FUSION



. Fig 1.4. The architecture of feature-level fusion with the BEL model [1]

In feature level fusion features are extracted from the fully connected layer of audio network (256D) and visual network (256D). The features are fused together to get 512 high level features which is passed to the MOBEL network. The MOBEL architecture consists of BEL experts and one BEL gating network. The outputs of each BEL expert is calculated using the below equations:

$$e_a = \sum_{j=1}^{n} v_j p_j + v_{n+1} p_{th}$$

$$e_o = \sum_{j=1}^{n} w_j p_j$$

$$E_i = f_e\left(f_{amg}(e_a) - f_{ofc}(e_o)\right)$$

$$\Delta v_{ij} = -\gamma v_{ij} + \alpha_e p_j \max\left(0, t - f_{amg}(e_a)\right)$$

$$\Delta w_{ij} = \beta_e p_j (E_i - t) \qquad \textit{for i=1,2 for j=1,2,3..n}$$

Ei is the output of each BEL expert and $v + \Delta v_{ij}$ and $w + \Delta w_{ij}$ are the updated weights after each epoch. α_e and β_e are the learning rate of AMG and OFC in the BEL expert. The gating expert is the probability that the expert can generate the desired output. The outputs of gating network is estimated using:

$$g_i = f_g\left(f_{amg}(e_a) - f_{ofc}(e_o)\right)$$

g_i is the output of the gating network. g_i is the weight value assigned to each expert .The BEL expert is trained using the Truth value T and the gating expert is trained using H which is the posterior probability that each expert can generate the desired output .

$$h_i = \frac{g_i exp\ (-\frac{1}{2}(t - E_i)^T (t - E_i))}{\sum_j g_i \exp\ (-\frac{1}{2}(t - E_i)^T (t - E_i))}$$

the output of the model is y :

$$y = \sum_i E_i\, g_i$$

The parameters of gating experts are updated based on the error function as follows:

$$e = \sum_i g_i\ ||y - E_i||^2$$

$$\Delta v_{gj} = -\gamma v_{gj} + \alpha_g p_j \max(0, h_i - f(e_a))$$

$$\Delta w_{gj} = \beta_g p_j (g_i - h_i)$$

Where $v_g + \Delta v_{gj}$ and $w_g + \Delta w_{gj}$ are the updated weights of the gating network after each epoch . . α_g and β_g are the learning rate of AMG and OFC in the gating expert. γ is the degradation rate for the momentum of each expert.

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

LITERATURE SURVEY:

There are various methods to recognize human emotional expressions. Some of them are listed below mentioning the merits and demerits of the proposed method over each of the existing methods.

- **"Learning Affective Features with Hybrid Deep Model for Audio-Visual Emotion Recognition"** by Shiqing Zhang[1], Shillang Zhang[1] ,Tiejun Huang[1],Wen Gao[1],Qi Tian[2]. This paper is based on Convolutional Neural Network(CNN) ,3D-Convolutional Neural Network and Deep Belief Networks(DBN). This paper proposes an hybrid model that bridges the emotional gap between the human emotions and audio-visual features using CNN and 3D-CNN and then it fuses audio-visual segment features with DBN . But the disadvantage of this method is that it limits its adaptability to specific human emotion as this method completely relies on pretrained architectures [2]

- **"Audio-Visual Affective Expression Recognition through Multistream Fused HMM"** by Zhihong Zeng [1],Jilin Tu [1],B.Pianfetti [1],Thomas S.Huang [1]. This paper uses Multi stream Fused Hidden Markov Model (MFHMM) algorithm that detects human emotion by using both audio and visual features. It combines both audio and visual streams to detect four cognitive states such as interest , boredom ,frustration and puzzlement and seven prototypical emotions such as neural, happiness, sadness, anger, disgust, fear and surprise. This paper lacks the discussion on noise robustness which is important for real world applications [3]

- **"Human emotion recognition by optimally fusing facial expression and speech feature"** by X Wang, X Chen, C Cao. This paper proposed a bimodal fusion algorithm to detect speech emotion recognition where both facial expressions and speech expressions are fused .CNN-RNN architecture is used to capture visual features and LSTM-CNN to capture audio features. The disadvantage of this paper

over proposed method is that it uses bimodal feature based emotion recognition. As this method is trained on RML bimodal dataset ,this limits the generalizability [4]

- **"Learning Spatiotemporal Features With 3D Convolutional Networks"** by Du Tran[1], Du Tran[2], Lubomir Bourdev[2], Rob Fergus[2], Lorenzo Torresani[1], Manohar Paluri[2] .This paper uses a simple 3D-Covolutional Networks(3D-ConvNets) ,a small homogenous architecture with kernel size as 3x3x3 to learn the spatiotemporal features by using simple linear classifier. The disadvantage is that it is not suitable for unseen data as this model is trained on specific dataset and thus it lacks generalization. [5]

- **"Speech emotion recognition using convolutional and Recurrent Neural Networks"** by Wootaek Lim[1], Dae-Young Jang[1], Taejin Lee[1] . This paper uses Convolutional Neural Network and Recurrent Neural Network .It recognize the human emotion from the speech signals . But the difference between this and the proposed paper is that it focuses on only the speech signals to recognize the emotion whereas the proposed method uses both facial and speech expressions . Thus this paper doesn't give accurate result when compared to the proposed method . [6]

MERITS AND DEMERITS

Merits:

- An audio-visual fusion method ,a promising approach ,proposed for human emotion recognition that addresses the challenges in multimodal emotion recognition
- To extract the spatial-temporal features of audio and visual data ,3D-CNN and CRNN are used which is a sound methodology that has been shown to be effective in previous research
- The proposed method jointly learns the audio-visual features and weights are allocated to each modality. This allows for more effective fusion of two modalities.
- The use of Mixture of Brain Emotion Learning (MOBEL) model which is inspired by the brain's limbic system (that is basically a combination of amgdala and orbito-frontal cortex) is an innovative approach that has the potential to improve the performance of the human emotion recognition.
- The use of MOBEL makes the model to consume less memory with high accuracy which improves the overall performance with high recognition rate .

Demerits:

- Since this model is trained and tested on only one dataset ,the eNTERFACE'05 ,this model limits the generalizability to the results of other dataset .
- Limited metrics are used to compare the performance with other methods which

makes it difficult in understanding the strength and weaknesses of the proposed method.

- Lack of discussion on hyper parameter tuning which is more important to achieve optimal solution .
- Doesn’t provide any information on computaional complexity of the proposed method which is crucial for understanding the scalability of the model to larger datasets .
- This paper did not provide any information related to the weights initialization in MoBEL architecture.

CHAPTER 3
SOURCE CODE

3.1. SPLITTING AUDIO-VIDEO FILE INTO VIDEO ONLY AND AUDIO ONLY FILE

```
import moviepy.editor as mp
import os
from moviepy.editor import VideoFileClip
import cv2
import pandas as pd
import numpy as np
import shutil
import tensorflow as tf
from tensorflow.keras.models import Model
from pydub import AudioSegment
import numpy as np
import librosa
import librosa.display

def split_audio_video(av_file,av_file_path, audio_path, video_path):
    #av_file_path = os.path.join("input_folder", av_file)  # Path to input audio-video file
    if not os.path.isfile(av_file_path):
        print(f"File {av_file} not found at {av_file_path}. Skipping...")
        return

    # Load the video clip
    video_clip = VideoFileClip(av_file_path)

    # Extract and save audio
    audio_clip = video_clip.audio

    audio_clip.write_audiofile(os.path.join(audio_path,
f"{av_file.split('.')[0]}_audio_only.wav"))

    # Write the video clip without audio
    video_clip_without_audio = video_clip.set_audio(None)
    video_clip_without_audio.write_videofile(os.path.join(video_path,
f"{av_file.split('.')[0]}_video_only.mp4"), codec='libx264')

    print(f"Audio and video splitted successfully: {av_file}")
    video_path=os.path.join(video_path, f"{av_file.split('.')[0]}_video_only.mp4")
    audio_path=os.path.join(audio_path, f"{av_file.split('.')[0]}_audio_only.wav")
    return video_path,audio_path
```

```
def flush_folder_contents(folder_path):
    # Iterate over all the contents of the folder
    for filename in os.listdir(folder_path):
        # Construct the full path to the file or folder
        file_path = os.path.join(folder_path, filename)

        # Check if it's a file or a folder
        if os.path.isfile(file_path):
            # If it's a file, simply delete it
            os.unlink(file_path)
        elif os.path.isdir(file_path):
            # If it's a folder, delete it recursively using shutil
            shutil.rmtree(file_path)
```

3.2. VIDEO PREPROCESSING

3.2.1. KEY FRAMES GENERATION

```
def generate_keyframes(video_path, output_folder,cnt, interval=1000):
    # Open the video file
    cap = cv2.VideoCapture(video_path)
    # Check if the video file is opened successfully
    if not cap.isOpened():
        print("Error: Couldn't open the video file")
        return
    frame_count = 0
    success, frame = cap.read()
    while success:
        if frame_count % interval == 0:
            #emotion_folder = os.path.join(destination_folder, emotion+"_key")
            if not os.path.exists(output_folder):
                os.makedirs(output_folder)
            keyframe_path = f"{output_folder}/frame_{cnt}_{frame_count}.jpg"
            cv2.imwrite(keyframe_path, frame)
            print(f"Keyframe saved: {keyframe_path}")
            success, frame = cap.read()
            frame_count += 1

    # Release the video capture object
    cap.release()
    return output_folder
```

3.2.2. EXTRACTING 12 FRAMES

```
def extract_12_frames_video(folder_path):
    videos = {}
    # Iterate through files in the folder
    for filename in os.listdir(folder_path):
        if filename.endswith('.jpg'):  # Assuming frames are in JPG format
            # Parse the filename to extract video identifier and frame number
            video_id, frame_num = map(str, filename.split('_')[1:])
            f_num=(frame_num.split(".")[0])
            #print(video_id,frame_num)
            # Add the frame to the corresponding video
            if video_id not in videos:
                videos[video_id] = []
            videos[video_id].append((int(f_num), filename))

    # Process each video to ensure it has only 12 frames
    for video_id, frames in videos.items():
        frames.sort()  # Sort frames by frame number
        # If video has more than 12 frames, adjust the frames
        while len(frames) > 12:
            if len(frames) % 2 == 0:
                # Delete the first frame
                os.remove(os.path.join(folder_path, frames[0][1]))
                del frames[0]
            else:
                # Delete the last frame
                os.remove(os.path.join(folder_path, frames[-1][1]))
                del frames[-1]

    # Output a message indicating processing is complete
    print("Processing complete. Each video now has 12 frames.")
```

3.2.3. SSD GENERATION

```
def ssd_generation_new(folder_path,output_folder):
    for filename in os.listdir(folder_path):
        if filename.endswith(".jpg") or filename.endswith(".png"): # Ensure only image files
are processed
            # Read the image
            image_path = os.path.join(folder_path, filename)
```

```
        image = cv2.imread(image_path)
        if image is None:
            print(f"Error: Unable to read image at {image_path}")
            continue

        base_img = image.copy()
        original_size = image.shape
        target_size = (300, 300)
        aspect_ratio_x = (original_size[1] / target_size[1])
        aspect_ratio_y = (original_size[0] / target_size[0])

        # Resize the image
        image = cv2.resize(image, target_size)

        # Generate blob from the resized image
        imageBlob = cv2.dnn.blobFromImage(image=image)
        detector.setInput(imageBlob)

        # Perform object detection
        detections = detector.forward()
        detections_df = pd.DataFrame(detections[0][0], columns=["img_id", "is_face",
"confidence", "left", "top", "right", "bottom"])
        detections_df = detections_df[detections_df['is_face'] == 1.0]  # Filter for faces
        detections_df = detections_df[detections_df['confidence'] >= 0.90]  # Confidence
threshold

        # Process each detected face
        for i, instance in detections_df.iterrows():
            confidence_score = str(round(100*instance["confidence"], 2)) + " %"
            left = int(instance["left"] * 300)
            bottom = int(instance["bottom"] * 300)
            right = int(instance["right"] * 300)
            top = int(instance["top"] * 300)

            # Extract detected face region from the original image
            detected_face = base_img[int(top*aspect_ratio_y):int(bottom*aspect_ratio_y),
int(left*aspect_ratio_x):int(right*aspect_ratio_x)]

            if detected_face.shape[0] > 0 and detected_face.shape[1] > 0:
                # Write the detected face image to disk
                if not os.path.exists(output_folder):
                    os.makedirs(output_folder)
```

```
                output_path = os.path.join(output_folder,
f"detected_face_{filename.split('.')[0]}.jpg")
                detected_face=cv2.resize(detected_face,(96,100))
                cv2.imwrite(output_path, detected_face)
                print(f"Detected face saved: {output_path}")
            else:
                print(f"Empty detected face for {filename}")
        else:
            print(f"Ignore non-image file: {filename}")
    return output_folder
```

3.2.4. VIDEO SEQUENCE GENERATION

```
def sequence_generation_video(folder_path):
    label_dic={"angry":0,"calm":1,"fearful":2,"happy":3,"neutral":4,"sad":5}
    # Define parameters
    num_frames = 12
    height = 100
    width = 96
    channels = 3

    # Path to the folder containing images
    #folder_path = "Video_Output/neutral_ssd"
    label=str(folder_path.split('/')[1]).split('_')[0]
    image_files = [os.path.join(folder_path, file) for file in os.listdir(folder_path) if
file.endswith(('.jpg', '.jpeg', '.png'))]
    # Create an empty NumPy array with the specified shape
    sequence_of_frames = np.zeros((num_frames, height, width, channels))
    main_arr = []
    main_labels = []
    cnt=0
    for img_path in image_files:
        frame_data = cv2.imread(img_path)
        frame_data = cv2.cvtColor(frame_data, cv2.COLOR_BGR2RGB)

        # Store frame data in sequence_of_frames array
        sequence_of_frames[cnt] = frame_data
        cnt += 1

        # If sequence_of_frames array is filled, append it to main_arr and reset
sequence_of_frames
        if cnt >= num_frames:
            main_arr.append(sequence_of_frames)
            main_labels.append(label_dic[label])
```

```
            cnt = 0
            sequence_of_frames = np.zeros((num_frames, height, width, channels))
    #Print the number of sequences stored in main_arr
    print("Number of sequences:", len(main_arr))
    print("Number of label:", len(main_labels))
    return main_arr,main_labels
```

3.3. AUDIO PREPROCESSING

3.3.1. AUDIO SEGMENTATION

```
from pydub import AudioSegment
import os
def split_audio_with_overlap(audio_path, segment_length, overlap_length, output_folder,
video_number, segment_number):
    audio = AudioSegment.from_file(audio_path)
    total_duration = len(audio)
    start = 0
    end = segment_length
    segment_index = segment_number
   c=0
    while start < total_duration:
        if end > total_duration:
            end = total_duration
            c=c+1
            if c>=1:
                break
        segment = audio[start:end]
        segment.export(os.path.join(output_folder,
f"audio_{video_number}_segment_{segment_index}.wav"), format="wav")
        print(output_folder+f"audio_{video_number}_segment_{segment_index}.wav")
       start = end - overlap_length
        end = start + segment_length
        segment_index += 1

    return
```

3.3.2. EXTRACTING 12 SEGMENTS FORM AUDIO

```
def extract_12_frames_audio(folder_path):
    videos = {}
    # Iterate through files in the folder
```

```
for filename in os.listdir(folder_path):
    if filename.endswith('.wav'):
        video_id=filename.split('_')[1]
        frame_num=filename.split('_')[3]
        f_num=(frame_num.split(".")[0])
        #print(video_id,frame_num)
        # Add the frame to the corresponding video
        if video_id not in videos:
            videos[video_id] = []
        videos[video_id].append((int(f_num), filename))

# Process each video to ensure it has only 12 frames
for video_id, frames in videos.items():
    frames.sort()  # Sort frames by frame number
    #print(frames)
    # If video has more than 12 frames, adjust the frames
    while len(frames) > 12:
        if len(frames) % 2 == 0:
            # Delete the first frame
            os.remove(os.path.join(folder_path, frames[0][1]))
            del frames[0]
        else:
            # Delete the last frame
            os.remove(os.path.join(folder_path, frames[-1][1]))
            del frames[-1]

    if len(frames) < 12:
        last_frame_num = frames[-1][1]
        segment_num = (last_frame_num.split("_")[-1]) # Extract the segment number
        #segment_num=last_frame_num
        segment_num=int(segment_num.split('.')[0])
        last_segment_filename = os.path.join(folder_path, last_frame_num)
        for i in range(len(frames), 12):
            segment_num += 1
            new_filename = f"audio_{video_id}_segment_{segment_num}.wav"
            new_segment_filename = os.path.join(folder_path, new_filename)
            frames.append((new_filename,))
            shutil.copyfile(last_segment_filename, new_segment_filename)

# Output a message indicating processing is complete
print("Processing complete. Each audio now has 12 frames.")
return folder_path
```

3.3.3. CONVERTING AUDIO SEGMENTS TO MEL SPECTROGRAM IMAGES

```
def spec_to_image(spec,output_folder,file_new ,eps=1e-6):
        mean = spec.mean()
        std = spec.std()
        spec_norm = (spec - mean) / (std + eps)
        spec_min, spec_max = spec_norm.min(), spec_norm.max()
        spec_scaled = 255 * (spec_norm - spec_min) / (spec_max - spec_min)
        spec_scaled = spec_scaled.astype(np.uint8)
        #mel_spec_db = librosa.power_to_db(spec_scaled)
        # Print the shape of the Mel spectrogram image
        #print("Shape of Mel Spectrogram Image:", spec_scaled.shape)
        # Plot the Mel spectrogram
        #plt.figure(figsize=(10, 4))
        librosa.display.specshow(spec_scaled, sr=22050)
        #output_folder="emotion_filesA/"
        output_path = os.path.join(output_folder,file_new)
        #plt.savefig(output_path,bbox_inches='tight', pad_inches=0, transparent=True)
        cv2.imwrite(output_path,spec_scaled)
        #return mel_spec_db
        return spec_scaled

def get_melspectrogram_db(file_path, sr=22050, n_fft=2048, hop_length=512,
n_mels=96, fmin=20, fmax=8000, top_db=80):
    wav,sr = librosa.load(file_path,sr=sr,duration=5) # i added
    if wav.shape[0]<5*sr:#5
        wav=np.pad(wav,int(np.ceil((5*sr-wav.shape[0])/2)),mode='reflect')#5
    else:
        wav=wav[:5*sr]#5
    spec=librosa.feature.melspectrogram(y=wav, sr=sr,
n_fft=n_fft,hop_length=hop_length,n_mels=n_mels,fmin=fmin,fmax=fmax)
    spec_db=librosa.power_to_db(spec,top_db=top_db)
    return spec_db

def get_audio(spec, sr=22050, n_fft=2048, hop_length=512, n_mels=96, fmin=20,
fmax=8000, top_db=80):
                                res = librosa.feature.inverse.mel_to_audio(spec,
                                sr=sr,
                                n_fft=2048,
                                hop_length=512,
                                win_length=None,
                                window='hann',
                                center=True,
                                pad_mode='reflect',
```

```
                                power=2.0,
                                n_iter=32)

def mel_generation(output_folder,dirpath):
    #output_folder = "Audio_Output/neutral_segment_mel/"
    #dirpath="Audio_Output/neutral_segment/"
    #os.makedirs(output_folder, exist_ok=True)
    for filename in os.listdir(dirpath):
        file_new="mel_"+str(filename.split('.')[0])+".jpg"
        spec=get_melspectrogram_db(dirpath+"/"+filename)

        spec_to_image(spec,output_folder,file_new)
        print(f'Mel_Spectrogram_img saved in {output_folder}{file_new}')
```

3.3.4. AUDIO SEQUENCE GENERATION

```
def sequence_generation_audio(folder_path):
    label_dic={"angry":0,"calm":1,"fearful":2,"happy":3,"neutral":4,"sad":5}
    # Define parameters
    num_frames = 12
    height = 96
    width = 216
    channels = 3

    # Path to the folder containing images
    #folder_path = "Video_Output/neutral_ssd"
    label=str(folder_path.split('/')[1]).split('_')[0]
    image_files = [os.path.join(folder_path, file) for file in os.listdir(folder_path) if
file.endswith(('.jpg', '.jpeg', '.png'))]
    # Create an empty NumPy array with the specified shape
    sequence_of_frames = np.zeros((num_frames, height, width, channels))
    main_arr = []
    main_labels = []
    cnt=0
    for img_path in image_files:
        frame_data = cv2.imread(img_path)
        frame_data = cv2.cvtColor(frame_data, cv2.COLOR_BGR2RGB)

        # Store frame data in sequence_of_frames array
        sequence_of_frames[cnt] = frame_data
        cnt += 1

        # If sequence_of_frames array is filled, append it to main_arr and reset
sequence_of_frames
```

```
        if cnt >= num_frames:
            main_arr.append(sequence_of_frames)
            main_labels.append(label_dic[label])
            cnt = 0
            sequence_of_frames = np.zeros((num_frames, height, width, channels))
    #Print the number of sequences stored in main_arr
    print("Number of sequences:", len(main_arr))
    print("Number of label:", len(main_labels))
    return main_arr,main_labels
```

3.4. MODEL TRAINING

3.4.1. 3D-CNN TRAINING

3.4.1.1. Training Sequence Generation

```
import numpy as np
import cv2
import pandas as pd

# Load CSV file
csv_path = "G:/mini-project/ssd_images_1_to_17_new.csv"
df = pd.read_csv(csv_path)

# Define parameters
num_frames = 12
height = 100
width = 96
channels = 3

# Create an empty NumPy array with the specified shape
sequence_of_frames = np.zeros((num_frames, height, width, channels))
main_arr = []
main_labels = []

# Iterate through each row in the DataFrame
cnt = 0
for index, row in df.iterrows():
    # Load the image file
    img_path = row['filepath']
    frame_data = cv2.imread(img_path)
    frame_data = cv2.cvtColor(frame_data, cv2.COLOR_BGR2RGB)
```

```
    # Store frame data in sequence_of_frames array
    sequence_of_frames[cnt] = frame_data
    cnt += 1

    # If sequence_of_frames array is filled, append it to main_arr and reset
sequence_of_frames
    if cnt >= num_frames:
        main_arr.append(sequence_of_frames)
        main_labels.append(row['label'])
        cnt = 0
        sequence_of_frames = np.zeros((num_frames, height, width, channels))

# Print the number of sequences stored in main_arr
print("Number of sequences:", len(main_arr))
print("Number of label:", len(main_labels))
```

3.4.1.2. Test Sequence Generation

```
import numpy as np
import cv2
import pandas as pd

# Load CSV file
csv_path = "G:/mini-project/ssd_images_19_to_24(test_data).csv"
dff = pd.read_csv(csv_path)

# Define parameters
num_frames = 12
height = 100
width = 96
channels = 3

# Create an empty NumPy array with the specified shape
sequence_of_frames = np.zeros((num_frames, height, width, channels))
main_arr_test = []
main_labels_test = []

# Iterate through each row in the DataFrame
cnt = 0
for index, row in dff.iterrows():
    # Load the image file
    img_path = row['filepath']
    frame_data = cv2.imread(img_path)
    frame_data = cv2.cvtColor(frame_data, cv2.COLOR_BGR2RGB)
```

```
    # Store frame data in sequence_of_frames array
    sequence_of_frames[cnt] = frame_data
    cnt += 1

    # If sequence_of_frames array is filled, append it to main_arr and reset
sequence_of_frames
    if cnt >= num_frames:
        main_arr_test.append(sequence_of_frames)
        main_labels_test.append(row['label'])
        cnt = 0
        sequence_of_frames = np.zeros((num_frames, height, width, channels))

# Print the number of sequences stored in main_arr
print("Number of sequences:", len(main_arr_test))
print("Number of label:", len(main_labels_test))
```

3.4.1.3. 3D-CNN

```
from sklearn.model_selection import KFold
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers,models
from tensorflow.keras.layers import Conv3D, MaxPooling3D, Flatten, LSTM, Dense,
Dropout, TimeDistributed, Reshape
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras import models,layers

frames=12
height=100
width=96
channels=3
input_shape = (frames, height, width, channels)
num_classes=6

def create_3dcnn(input_shape, num_classes):
    model = Sequential()

    # Convolutional layers
    model.add(Conv3D(64, (3, 3, 3), activation='relu', padding="same",
input_shape=input_shape))
    print(model.output_shape)
```

```
    model.add(MaxPooling3D((2, 2, 2), padding='same'))

    model.add(Conv3D(128, (3, 3, 3), activation='relu', padding="same"))
    model.add(MaxPooling3D((2, 2, 2), padding='same'))

    model.add(Conv3D(256, (3, 3, 3), activation='relu', padding="same"))
    model.add(MaxPooling3D((2, 2, 2), padding='same'))

    model.add(Conv3D(512, (3, 3, 3), activation='relu', padding="same"))
    model.add(MaxPooling3D((2, 2, 2), padding='same'))

    model.add(Dropout(0.5))

    # Flatten the output for fully connected layers
    model.add(Flatten())

    # Fully connected layers
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    return model

model = create_3dcnn(input_shape, num_classes)
model.summary()
main_arr = np.array(main_arr)
main_labels = np.array(main_labels)
main_arr_test = np.array(main_arr_test)
main_labels_test = np.array(main_labels_test)

frames=12
height=100
width=96
channels=3
input_shape = (frames, height, width, channels)
num_classes=6

#main_arr=(main_arr/255.0)
l=LabelEncoder()
main_labels_encoded=l.fit_transform(main_labels)
```

```
#x_train,x_val,y_train,y_val=train_test_split(main_arr,main_labels_encoded,test_size=0.
2,random_state=42)

kf = KFold(n_splits=5, shuffle=True, random_state=42)

cv_scores = []
# Lists to store training and validation accuracies
train_accuracies = []
val_accuracies = []
for train_index, val_index in kf.split(main_arr):
    x_train, x_val = main_arr[train_index], main_arr[val_index]
    y_train, y_val = main_labels[train_index], main_labels[val_index]

    # Compile the model
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

    # Train the model
    history=model.fit(x_train, y_train, epochs=20, batch_size=12,
verbose=1,validation_data=(x_val, y_val))

    # Evaluate the model
    test_loss, test_accuracy = model.evaluate(x_val, y_val)
    cv_scores.append(test_accuracy)
    # Record training and validation accuracies
    train_accuracy = history.history['accuracy']
    val_accuracy = history.history['val_accuracy']
    train_accuracies.append(train_accuracy)
    val_accuracies.append(val_accuracy)

mean_train_accuracy = np.mean(train_accuracies, axis=0)
mean_val_accuracy = np.mean(val_accuracies, axis=0)

# Print the cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean Accuracy:", np.mean(cv_scores))


model.save('3DCNN_model_complex.h5')
print("the model saved successfully")
```

3.4.1.4. Feature Model Extraction

```
from tensorflow.keras.models import Model
```

```
model = tf.keras.models.load_model('3DCNN_model_complex.h5')
layer_names = [‘dense_22’]
feature_extractor = Model(inputs=model.input, outputs=[model.get_layer(name).output
for name in layer_names])
feature_extractor.save('3DCNN_feature.h5')
```

3.4.1.5. Analysis of Model testing

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(main_arr_test, main_labels_test)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
print(f'Test Loss: {test_loss * 100:.2f}%')

# Make predictions
predictions = model.predict(main_arr_test)
print("The predictions are:", predictions)

# Get predictions for the test set
predicted_classes = np.argmax(predictions, axis=1)

from sklearn.metrics import
accuracy_score,precision_score,recall_score,f1_score,classification_report

# Calculate accuracy
accuracy = accuracy_score(main_labels_test, predicted_classes)
print(f'Accuracy: {accuracy * 100:.2f}%')
precision = precision_score(main_labels_test, predicted_classes,average="weighted")
print(f'Precision: {precision * 100:.2f}%')
recall = recall_score(main_labels_test, predicted_classes,average="weighted")
print(f'Recall: {recall * 100:.2f}%')
f1 = f1_score(main_labels_test, predicted_classes,average="weighted")
print(f'F1-Score: {f1 * 100:.2f}%')
cl_report=classification_report(main_labels_test,predicted_classes)
print("Classification Report : ",cl_report)
from sklearn.metrics import confusion_matrix

# Get predictions for the test set
predicted_classes = np.argmax(predictions, axis=1)

# Calculate confusion matrix
conf_matrix = confusion_matrix(main_labels_test, predicted_classes)
```

```
# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=["angry","calm","fearful","happy","neutral","sad"],
            yticklabels=["angry","calm","fearful","happy","neutral","sad"])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
# Plot the graph
epochs = range(1, len(mean_train_accuracy) + 1)
plt.plot(epochs, mean_train_accuracy, 'bo', label='Mean Training Accuracy')
plt.plot(epochs, mean_val_accuracy, 'r', label='Mean Validation Accuracy')
plt.title('Mean Training and Validation Accuracy Across Folds')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

3.4.2. CRNN TRAINING

3.4.2.1. Training Data Sequence Generation

```
# Load CSV file
csv_path = "mel_images_1_17.csv"
dff = pd.read_csv(csv_path)

# Define parameters
num_frames = 12
height = 96
width = 216
channels = 3

# Create an empty NumPy array with the specified shape
sequence_of_frames = np.zeros((num_frames, height, width, channels))
main_arr = []
main_labels = []

# Iterate through each row in the DataFrame
cnt = 0
for index, row in dff.iterrows():
    # Load the image file
    img_path = row['filepath']
    frame_data = cv2.imread(img_path)
```

```
    frame_data = cv2.cvtColor(frame_data, cv2.COLOR_BGR2RGB)

    # Store frame data in sequence_of_frames array
    sequence_of_frames[cnt] = frame_data
    cnt += 1

    # If sequence_of_frames array is filled, append it to main_arr and reset
sequence_of_frames
    if cnt >= num_frames:
        main_arr.append(sequence_of_frames)
        main_labels.append(row['label'])
        cnt = 0
        sequence_of_frames = np.zeros((num_frames, height, width, channels))

# Print the number of sequences stored in main_arr
print("Number of sequences:", len(main_arr))
print("Number of label:", len(main_labels))
```

3.4.2.2.Test Data Sequence Generation

```
# Load CSV file
csv_path = "mel_images_18_24.csv"
dff = pd.read_csv(csv_path)

# Define parameters
num_frames = 12
height = 96
width = 216
channels = 3

# Create an empty NumPy array with the specified shape
sequence_of_frames = np.zeros((num_frames, height, width, channels))
main_arr_test = []
main_labels_test = []

# Iterate through each row in the DataFrame
cnt = 0
for index, row in dff.iterrows():
    # Load the image file
    img_path = row['filepath']
    frame_data = cv2.imread(img_path)
    frame_data = cv2.cvtColor(frame_data, cv2.COLOR_BGR2RGB)

    # Store frame data in sequence_of_frames array
```

```
    sequence_of_frames[cnt] = frame_data
    cnt += 1

    # If sequence_of_frames array is filled, append it to main_arr and reset
sequence_of_frames
    if cnt >= num_frames:
        main_arr_test.append(sequence_of_frames)
        main_labels_test.append(row['label'])
        cnt = 0
        sequence_of_frames = np.zeros((num_frames, height, width, channels))

# Print the number of sequences stored in main_arr
print("Number of sequences:", len(main_arr_test))
print("Number of label:", len(main_labels_test))
```

3.4.2.3. CRNN

```
main_arr_test=np.array(main_arr_test)
main_labels_test=np.array(main_labels_test)

from sklearn.model_selection import KFold
import numpy as np
from tensorflow.keras.layers import TimeDistributed
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, LSTM, Dense,
Dropout, TimeDistributed, Reshape
from tensorflow.keras.layers import TimeDistributed, BatchNormalization

# Define input shape and number of classes
input_frames = 12
height = 96
width = 216
channels = 3
num_classes = 6
input_shape = (input_frames, height, width, channels)

def create_crnn_model(input_shape, num_classes):
    model = Sequential()

    # Convolutional layers
    model.add(TimeDistributed(Conv2D(64, kernel_size=(3, 3), activation='relu'),
input_shape=input_shape))
    model.add(TimeDistributed(BatchNormalization()))
    model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2))))
```

```
    model.add(TimeDistributed(Conv2D(64, kernel_size=(3, 3), activation='relu')))
    model.add(TimeDistributed(BatchNormalization()))
    model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2))))

    model.add(TimeDistributed(Conv2D(64, kernel_size=(3, 3), activation='relu')))
    model.add(TimeDistributed(BatchNormalization()))
    model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2))))

    model.add(TimeDistributed(Flatten()))

    # Fully connected layers
    model.add(TimeDistributed(Dense(256, activation='relu')))
    model.add(TimeDistributed(BatchNormalization()))
    model.add(TimeDistributed(Dense(256, activation='relu')))
    model.add(TimeDistributed(BatchNormalization()))

    # LSTM layer
    model.add(LSTM(256, return_sequences=False))

    # Fully connected layer after LSTM
    model.add(Dense(256, activation='relu'))
    model.add(BatchNormalization())

    # Output layer
    model.add(Dense(num_classes, activation='softmax'))

    return model

# Create the CRNN model
model = create_crnn_model(input_shape, num_classes)
# Convert your data to numpy arrays if needed
x_train_np = np.array(main_arr)
y_train_np = np.array(main_labels)

main_arr = np.array(main_arr)
main_labels = np.array(main_labels)


# Split the data using KFold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

cv_scores = []
```

```
for train_index, val_index in kf.split(main_arr):
    x_train, x_val = main_arr[train_index], main_arr[val_index]
    y_train, y_val = main_labels[train_index], main_labels[val_index]

    # Compile the model
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

    # Train the model
    model.fit(x_train, y_train, epochs=20, batch_size=12, verbose=1)

    # Evaluate the model
    test_loss, test_accuracy = model.evaluate(x_val, y_val)
    cv_scores.append(test_accuracy)

# Print the cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean Accuracy:", np.mean(cv_scores))


model.save('CRNN_model.h5')
print("the model saved successfully")
```

3.4.2.4. Feature Model Extraction

```
import tensorflow as tf
from tensorflow.keras.models import Model

# Load your trained model
model = tf.keras.models.load_model('CRNN_model.h5')

# Choose layers from which to extract features
layer_names = [‘dense_14’]

# Create a new model that outputs the features from chosen layers
feature_extractor = Model(inputs=model.input, outputs=[model.get_layer(name).output
for name in layer_names])
feature_extractor.save('CRNN_feature.h5')
```

3.4.2.5. Analysis of Model testing

```
import tensorflow as tf
model = tf.keras.models.load_model('CRNN_model.h5')
# Evaluate the model on the test set
```

```
test_loss, test_accuracy = model.evaluate(main_arr_test, main_labels_test)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
print(f'Test Loss: {test_loss * 100:.2f}%')
predictions = model.predict(main_arr_test)
print(predictions)
predicted_classes = np.argmax(predictions, axis=1)
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score

# Calculate accuracy
accuracy = accuracy_score(main_labels_test, predicted_classes)
print(f'Accuracy: {accuracy * 100:.2f}%')
precision = precision_score(main_labels_test, predicted_classes,average="weighted")
print(f'Precision: {precision * 100:.2f}%')
recall = recall_score(main_labels_test, predicted_classes, average="weighted"))
print(f'Recall: {recall * 100:.2f}%')
f1 = f1_score(main_labels_test, predicted_classes, average="weighted"))
print(f'F1-Score: {f1 * 100:.2f}%')

from sklearn.metrics import confusion_matrix

import seaborn as sns

# Get predictions for the test set
predicted_classes = np.argmax(predictions, axis=1)

# Calculate confusion matrix
conf_matrix = confusion_matrix(main_labels_test, predicted_classes)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
        xticklabels=["angry","calm","fearful","happy","neutral","sad"],
        yticklabels=["angry","calm","fearful","happy","neutral","sad"])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

3.5 FEATURE FUSION

3.5.1. 512 FEATURES GENERATION

```
directory = 'F:/mini-project/Emotion_Folders_AV_1_to_17/'

detector = cv2.dnn.readNetFromCaffe("deploy.prototxt",
"res10_300x300_ssd_iter_140000.caffemodel")
```

```
# Create an empty DataFrame
df = pd.DataFrame(columns=['features', 'labels'])
for dirpath, dirnames, filenames in os.walk(directory):
    print(dirpath)
    #print(dirnames)
    for filename in filenames:
        print(filename)
        label=str(dirpath.split("/")[3])
        output_audio_path="Audio_Output/"
        output_video_path="Video_Output/"
        flush_folder_contents("Audio_Output")
        flush_folder_contents("Video_Output")
        print("Existing contents in the output folders are flushed")
        av_file_path=dirpath+"/"+filename

video_path,audio_path=split_audio_video(filename,av_file_path,output_audio_path,outp
ut_video_path)

        # for video
        cnt=0
        output_folder =  "Video_Output/"+label+"_key"
        print(output_folder)
        os.makedirs(output_folder, exist_ok=True)
        folder_path=generate_keyframes(dirpath+"/"+filename,
"Video_Output/"+label+"_key",cnt, interval=8)
        extract_12_frames_video(folder_path)
        f_p=ssd_generation_new(folder_path,"Video_Output/"+label+"_ssd/")
        main_arr_v,main_labels_v=sequence_generation_video(f_p)
        main_arr_v=np.array(main_arr_v)
        main_labels_v=np.array(main_labels_v)

        model_3dcnn = tf.keras.models.load_model('3DCNN_feature.h5')
        video_features_256=model_3dcnn.predict(main_arr_v)

        #for Audio
        audio_number = 0
        output_folder = "Audio_Output/" + label + "_segment/"
        os.makedirs(output_folder, exist_ok=True)
        split_audio_with_overlap(audio_path, segment_length=1000, overlap_length=700,
output_folder=output_folder, video_number=audio_number, segment_number=0)
        seg_folder_path=extract_12_frames_audio(output_folder)
        o_f_p="Audio_Output/"+str(seg_folder_path).split("/")[1]+"_mel/"
        os.makedirs(o_f_p, exist_ok=True)
```

```
        mel_generation(o_f_p,seg_folder_path)
        main_arr_a,main_labels_a=sequence_generation_audio(o_f_p)
        main_arr_a=np.array(main_arr_a)
        main_labels_a=np.array(main_labels_a)

        model_crnn = tf.keras.models.load_model('CRNN_feature.h5')
        audio_features_256=model_crnn.predict(main_arr_a)

        concatenated_features = np.concatenate((video_features_256, audio_features_256),
axis=1)

        features = list(concatenated_features)    #512 features of single sample
        labels=list(main_labels_v)   # label of particular video in number which is index of
emotion

        # Add records row by row
        for feature, label in zip(features, labels):
            df = df.append({'features': feature, 'labels': label}, ignore_index=True)
        print("Feature appended in dataframe")

df.to_csv('AV_512_features_labels_new_all.csv', index=False)

import csv
import numpy as np
# Path to your CSV file
csv_file = "AV_512_features_labels_new_all.csv"

# Initialize a list to store the features
features_list = []

# Read the CSV file
with open(csv_file, "r") as file:
    reader = csv.reader(file)
    next(reader)  # Skip the header row
    # Iterate through each row in the CSV file
    for row in reader:
        # Strip the square brackets and split the row string by space
        values = row[0].strip("[]").split()
        # Convert each value to float and store in a list
        feature_row = [float(val) for val in values]
        # Append the feature row to the features list
        features_list.append(feature_row)

# Convert the list of lists into a NumPy array
```

```
features_array = np.array(features_list)

# Check the shape of the array
print("Features shape:", features_array.shape)

import pandas as pd
import numpy as np

# Path to your CSV file
csv_file = "AV_512_features_labels_new_all.csv"

# Read the CSV file into a DataFrame
df = pd.read_csv(csv_file)

labels_array = df['labels'].values.reshape(-1, 1)

# Convert the extracted features and labels into NumPy arrays

labels_array = np.array(labels_array)

# Check the shapes of the arrays

print("Labels shape:", labels_array.shape)

P=features_array
T=labels_array

import matplotlib.pyplot as plt
unique_labels = np.unique(labels_array)
symbols = ['o', 's', '^', 'D', 'v', 'P']
labels=['angry','calm','fearful','happy','neutral','sad']
plt.figure(figsize=(10, 6))
for idx, feature_list in enumerate(features_array):
    average_feature = np.mean(feature_list)  # Calculate the average of the feature list
    label_value = labels_array[idx]  # Get the corresponding label value
    label_idx = np.where(unique_labels == label_value)[0][0]  # Find the index of the label
in unique_labels
    plt.scatter(average_feature, label_value, label=f'Feature {idx+1}',
marker=symbols[label_idx])
plt.xlabel('Average Feature Value')
plt.ylabel('Emotion Label')
plt.yticks(range(len(labels)), labels)
plt.title('Scatter Plot of Average Feature Value vs. Emotion Label for Each Feature')
plt.legend()
```

```
plt.show()
import matplotlib.pyplot as plt
import numpy as np
unique_labels = np.unique(labels_array)

symbols = ['o', 's', '^', 'D', 'v', 'P']
colors = ['b', 'g', 'r', 'c', 'm', 'y']

plt.figure(figsize=(10, 6))
for idx, feature_list in enumerate(features_array):
    average_feature = np.std(feature_list)
    label_value = labels_array[idx][0]
    label_idx = np.where(unique_labels == label_value)[0][0]  in unique_labels
    plt.scatter(average_feature, label_value, label=f'Emotion {label_value}',
marker=symbols[label_idx], color=colors[label_idx])

plt.xlabel('Standard Deviation of Feature Value')
plt.ylabel('Emotion Label')
plt.title('Scatter Plot of Standard Deviation of Feature Value vs. Emotion Label for Each
Feature')
plt.legend()
plt.show()
```

3.5. MOBEL TRAINING

```
def bel_expert(V, P, Vn, W):
    # Initialize an empty list to store the results for each row
    result = []
    # Loop through each row of P
    for row in P:
        # Perform the dot product of the row of P with the weight vector V
        #print(row)
        #print(V)
        e_A_row = np.dot(row, V) + Vn * np.max(row)
        # Compute other operations based on the dot product result
        e_O_row = np.dot(row, W)
        f_e_A_row = tansig(e_A_row)
        f_e_O_row = tansig(e_O_row)
        f_E_row = tansig(f_e_A_row - f_e_O_row)
        # Append the result for this row to the list
        result.append(f_E_row)
    # Convert the list of results to a NumPy array
    f_E = np.array(result)
```

```
    return f_E

import numpy as np

def initialize_parameters(n_x, n_y):
    np.random.seed(1)

    V = np.random.randn(n_x, 1) * 0.01
    W = np.random.randn(n_x, 1) * 0.01
    Vn = 0.01

    parameters = {"V": V, "W": W, "Vn": Vn}

    return parameters

def relu(x):
    return np.maximum(0, x)

def tansig(x):
    return 2 / (1 + np.exp(-2*x)) - 1


def bel_expert(V, P, Vn, W):
    # Initialize an empty list to store the results for each row
    result = []
    # Loop through each row of P
    for row in P:
        # Perform the dot product of the row of P with the weight vector V
        #print(row)
        #print(V)
        e_A_row = np.dot(row, V) + Vn * np.max(row)
        # Compute other operations based on the dot product result
        e_O_row = np.dot(row, W)
        f_e_A_row = tansig(e_A_row)
        f_e_O_row = tansig(e_O_row)
        f_E_row = tansig(f_e_A_row - f_e_O_row)
        # Append the result for this row to the list
        result.append(f_E_row)
    # Convert the list of results to a NumPy array
    f_E = np.array(result)

    return f_E
```

```
'''def error_fun(E, Y, G):
    e = np.sum(G * (np.square(Y - E)))
    return e'''

def error_fun(Y, T, G):
    e = np.sum(G * (np.square(T - Y)))
    return e

def posterior_prob(G, E, T):

    #numerator = np.exp(-0.5 * np.sum((T - E)**2))
    numerator=(-0.5*np.sum((T-E)**2))
    denominator = np.sum(G * (-0.5 * np.sum((T - E)**2)))
    print("numerator:",numerator)
    result = numerator / denominator
    print("res",result)
    return result

def update_params_BEL(W, V, P, Vn, gamma, alpha, beta, T):
    e_A = np.dot(P, V) + Vn * np.max(P)
    e_O = np.dot(P, W)
    f_e_A = tansig(e_A)
    f_e_O = tansig(e_O)
    f_E = tansig(f_e_A - f_e_O)

    dV = -gamma * V + alpha * np.dot(np.maximum(0, T - relu(e_A)).reshape(1, -1),
P).reshape(-1, 1)
    V = V + dV

    dW = beta * np.dot((f_E - T).reshape(1, -1), P).reshape(-1, 1)
    W = W + dW

    updt = {"V": V, "W": W}
    return updt

def update_params_GE(Vg, Wg, Vng, H, P, G, gamma, alpha_g, beta_g):
    e_A = np.dot(P, Vg) + Vng * np.max(P)
    dVg = -gamma * Vg + alpha_g * np.dot(np.maximum(0, H - tansig(e_A)).reshape(1, -
1), P).reshape(-1, 1)
    Vg = dVg + Vg

    dWg = beta_g * np.dot((G - H).reshape(1, -1), P).reshape(-1, 1)
    Wg = dWg + Wg
```

```
    updt = {"Vg": Vg, "Wg": Wg}
    return updt

def MODEL(num_epochs, P, T, gamma, alpha, beta, alpha_g, beta_g):
    params1 = initialize_parameters(512, 1)
    params2 = initialize_parameters(512, 1)
    params3 = initialize_parameters(512, 1)
    params4 = initialize_parameters(512, 1)

    hist = []
    V1 = params1["V"]
    W1 = params1["W"]
    Vn1 = params1["Vn"]

    V2 = params2["V"]
    W2 = params2["W"]
    Vn2 = params2["Vn"]

    Vg1 = params3["V"]
    Wg1 = params3["W"]
    Vng1 = params3["Vn"]

    Vg2 = params4["V"]
    Wg2 = params4["W"]
    Vng2 = params4["Vn"]

    for epoch in range(100):
        E1 = bel_expert(V1, P, Vn1, W1)
        E2 = bel_expert(V2, P, Vn2, W2)

        G1 = bel_expert(Vg1, P, Vng1, Wg1)
        G2 = bel_expert(Vg2, P, Vng2, Wg2)

        H1 = posterior_prob(G1, E1, T)
        H2 = posterior_prob(G2, E2, T)

        E = np.array([[E1], [E2]])
        G = np.array([[G1], [G2]])

        Y = (E1 * G2)+(E2 * G1)
        # Y = (E1 * G2) + (E2 * G1)  # Original line
        #Y = E1 if G1 > G2 else E2  # Modified line
        #sy=softmax(Y)
        #print("sy:",sy)
```

```
error = error_fun(Y, T, G)
#print("Error:",error)
#error = error_fun(E, Y, G)
print("H1:",H1)
hist.append(error)

U1 = update_params_BEL(W1, V1, P, Vn1, gamma, alpha, beta, T)
U2 = update_params_BEL(W2, V2, P, Vn2, gamma, alpha, beta, T)
V1 = U1["V"]
W1 = U1["W"]

V2 = U2["V"]
W2 = U2["W"]
#print("V1:",V1)
U3 = update_params_GE(Vg1, Wg1, Vng1, H2, P, G1, gamma, alpha_g, beta_g)
U4 = update_params_GE(Vg2, Wg2, Vng2, H1, P, G2, gamma, alpha_g, beta_g)
Vg1 = U3["Vg"]
Wg1 = U3["Wg"]
#print("vg1:",Wg1)
Vg2 = U4["Vg"]
Wg2 = U4["Wg"]

#print(f"Epoch {epoch}: Error = {error}")
print("Y:",Y)
```

CHAPTER 4

OUTPUT SNAPSHOTS

Fig 4.1. Sample image before SSD resnet

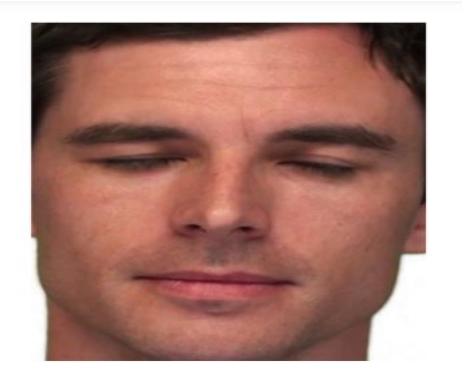
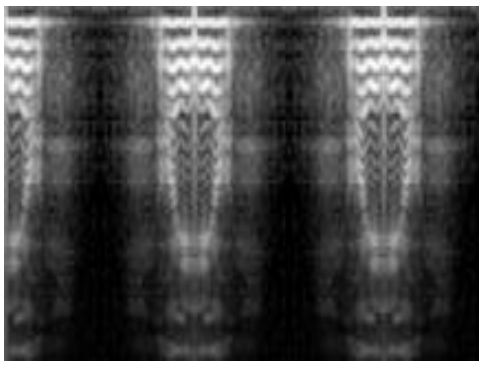


Fig 4.2. Sample image after SSD resnet



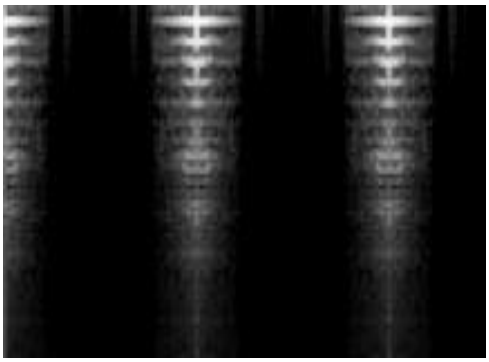


Fig 4.3. Sample Mel-spectrogram images

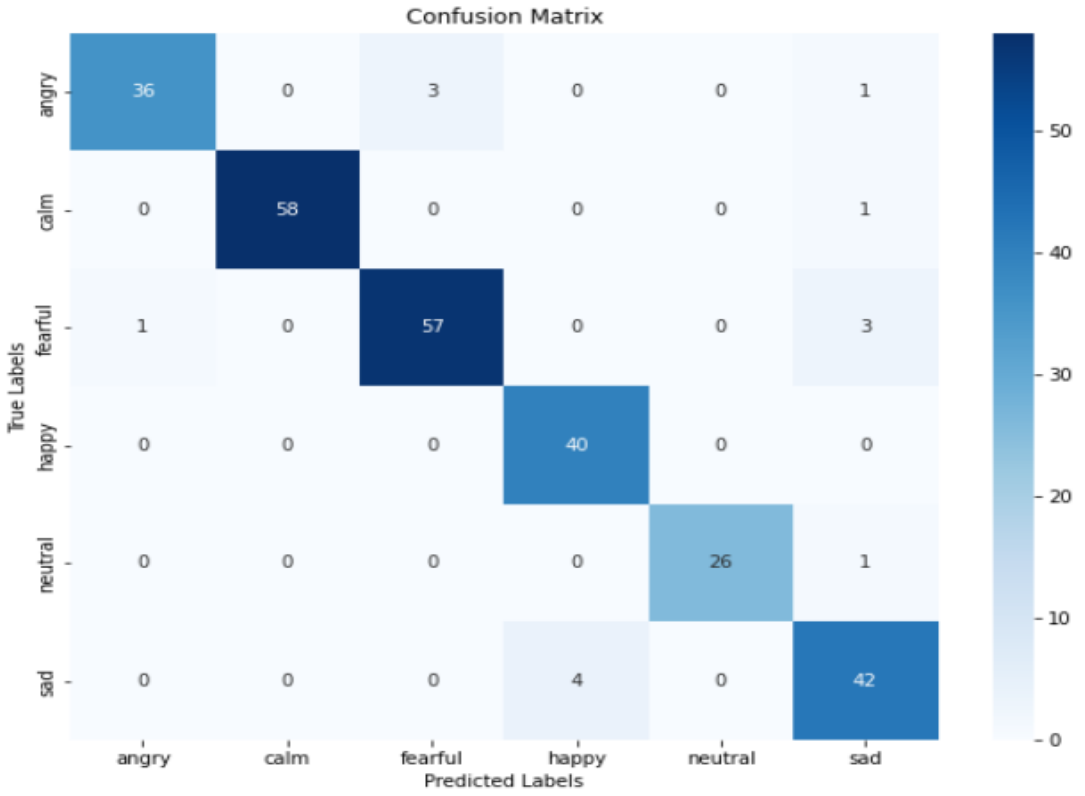


Fig 4.4. Confusion matrix for 3D-CNN

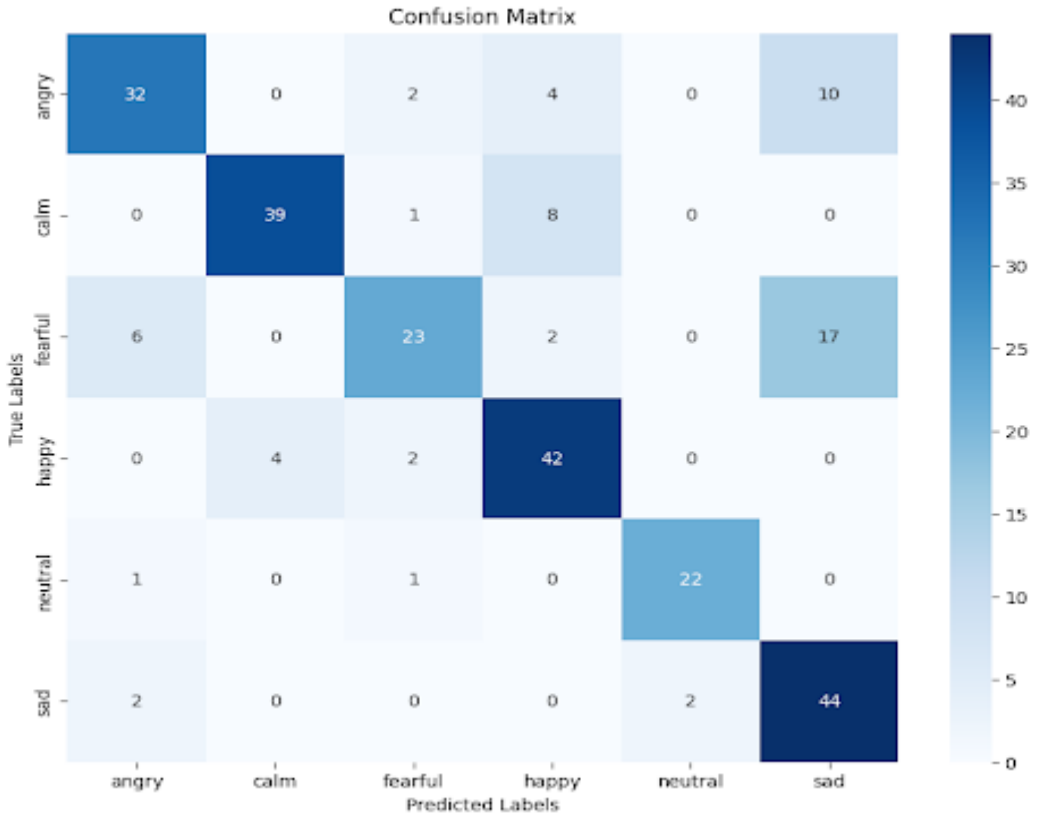


Fig 4.5.Confusion matrix for CRNN

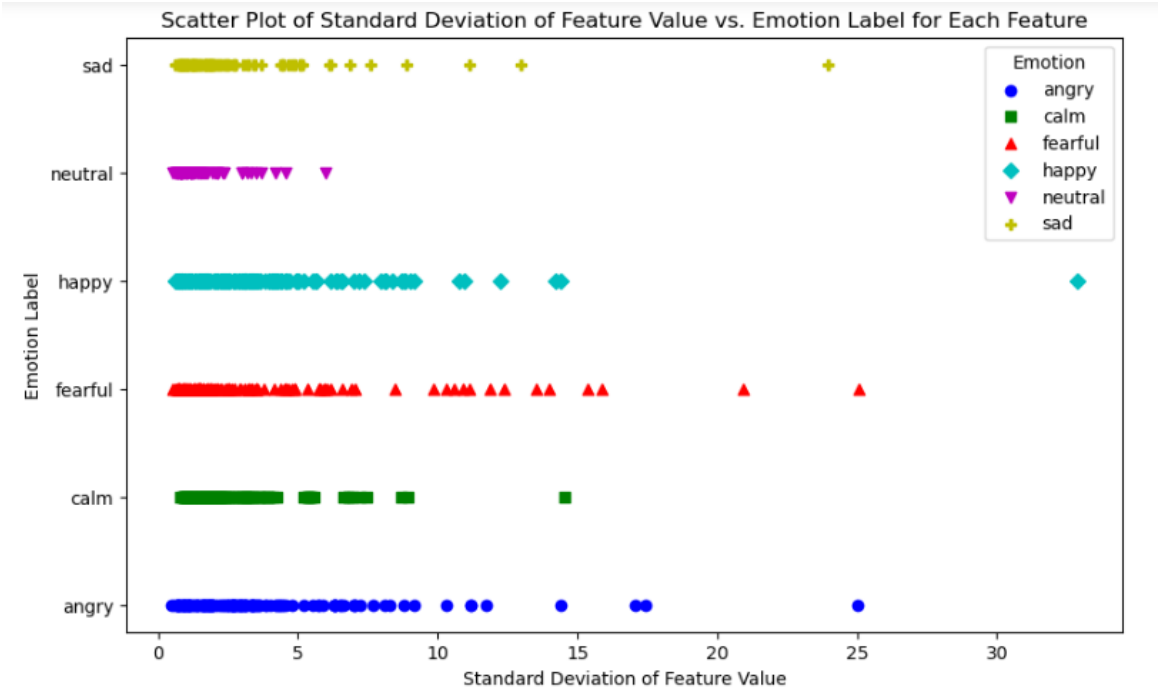


Fig 4.6. Scatter plot of Standard Deviation of Feature Vs Emotion Label

CHAPTER 5
CONCLUSION AND FUTURE PLANS

In this project, an innovative approach to recognize emotion from video files has been proposed. This method aims at learning the nonlinear spatial-temporal correlations between Visual and Audio modality. It first splits the video files into audio and visual streams and then features are extracted using 3DCNN and CRNN model. The extracted features are then passed to the Decision Level Fusion and Feature Level Fusion model. When feature-level fusion and decision-level fusion methods were compared ,we found that the recognition accuracy of multimodal emotion recognition is significantly increased from 74% at decision-level fusion using product rule and BEL model to 81.7% at the feature-level fusion using the MOBEL method. Also our multimodal
Emotion recognition model predicts emotion better than either visual based emotion recognition or audio based emotion recognition models.

Our model currently recognizes Sad, Happy, Fear, Disgust , Surprise with High accuracy. We can improvise the model to recognize emotions like Sarcasm, frustation etc.

CHAPTER 6

REFERENCES

1.Farhoudi, Z. and Setayeshi, S., 2021. Fusion of deep learning features with mixture of brain emotional learning for audio-visual emotion recognition. *Speech Communication*, *127*, pp.92-103.

2. Zhang, S., Zhang, S., Huang, T., Gao, W. and Tian, Q., 2017. Learning affective features with a hybrid deep model for audio–visual emotion recognition. *IEEE transactions on circuits and systems for video technology*, *28*(10), pp.3030-3043.

3. Zeng, Z., Tu, J., Pianfetti, B.M. and Huang, T.S., 2008. Audio–visual affective expression recognition through multistream fused HMM. *IEEE Transactions on multimedia*, *10*(4), pp.570-577.

4. Wang, X., Chen, X. and Cao, C., 2020. Human emotion recognition by optimally fusing facial expression and speech feature. *Signal Processing: Image Communication*, *84*, p.115831.

5. Tran, D., Bourdev, L., Fergus, R., Torresani, L. and Paluri, M., 2015. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 4489-4497).

6. Lim, W., Jang, D. and Lee, T., 2016, December. Speech emotion recognition using convolutional and recurrent neural networks. In *2016 Asia-Pacific signal and information processing association annual summit and conference (APSIPA)* (pp. 1-4). IEEE.

CHAPTER 7

APPENDIX

BASE PAPER

Farhoudi, Z. and Setayeshi, S., 2021. Fusion of deep learning features with mixture of brain emotional learning for audio-visual emotion recognition. *Speech Communication*, *127*, pp.92-103.

doi: 10.1016/j.specom.2020.12.001

keywords: {Spatial temporal features, 3D-CNN, CRNN, SSD ,MEL spectrogram, Feature Fusion, MOBEL }

URL:
https://www.sciencedirect.com/science/article/pii/S0167639320303058?casa_token=qRPehxOqJHIAAAAA:LNUr5qGH5J96MvBh2faGPnBCTFbqnFFAiR6FQZE9k-V7ZU_gub8X6B7-thwkbXI4FG7KbHbhZA