LEARNING FACIAL EXPRESSION AND BODY GESTURE VISUAL INFORMATION FOR VIDEO EMOTION RECOGNITION

Report submitted to the SASTRA Deemed to be University

as the requirement for the course

CSE300: MINI PROJECT

Submitted by

Shreya S

(Reg. No : 125003447, B.Tech Computer Science and Engineering 7)

Pavithra R

(Reg. No: 125003225, B.Tech Computer Science and Engineering)

Divya W B

(Reg. No: 125003074, B.Tech Computer Science and Engineering)

May 2024

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

THANJAVUR | KUMBAKONAM | CHENNAI

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

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR – 613401

Bonafide Certificate

This is to certify that the report titled "**Learning facial expression and body gesture visual information for video emotion recognition**" submitted as a requirement for the course. **CSE300: MINI PROJECT** for B.Tech is a bonafide record of the work done by **Ms. Shreya S (Reg No: 125003447, B.Tech Computer Science and Engineering), Ms. Pavithra R (Reg No: 125003225, B.Tech Computer Science and Engineering) & Ms. Divya W B (Reg. No. 125003074, B.Tech Computer Science and Engineering)** during the academic year 2023-24, in the School of Computing, under my supervision.

Signature of Project Supervisor:

Name with Affiliation : Dr. P L K Priyadarsini, SoC, SASTRA Deemed University

Date : 22-04-24

Project *Vivavoce* 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. 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. Algeswaran**, Associate Dean, Students Welfare

Our guide **Dr. P. L.K. Priyadarsini**, Associate Professor, 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 us an opportunity to showcase our skills through project.

List of Figures

Abbreviations

CNN	Convolutional Neural Networks
LSTM	Long Short Term Memory
RNN	Recurrent Neural Network
SISTCM	Super Image Spatiotemporal Convolutional Model
ACCM	Attention based Channel-wise Convolutional Model
FC	Fully Connected layer
3D	3 Dimensions
2D	2 Dimensions

Notations

English Symbols (in alphabetical order)

G	Body gesture representation
H	Height
nin	Number of input channels
$nout$	Number of output channels
W(t)	Temporal relationship function
T	Time
W	Width

Greek Symbols (in alphabetical order)

Α	Alpha
λ	Lamda
μ	Mu
∑	Summation
Wα	Weight

Abstract

Human interactions are fundamentally based on emotions, and AI’s capacity to perceive and react to emotions creates a plethora of opportunities. Intelligent analysis of emotional states conveyed by video aids in understanding the user’s emotions and improves services to increase marketing competitiveness. Emotion recognition has a significant impact on human-computer interaction, educational practices, intelligent vehicles, marketing and mental health. According to recent studies, body language and facial expressions have a big role in determining emotions. However, the contextual information of neighboring frames is the primary focus of these studies, and the spatiotemporal relationships between distant or global frames are rarely explored.

The authors suggest enhancing the efficiency of video emotion recognition by extracting spatiotemporal features through additional temporal encoding. To capture the local spatiotemporal features of the facial expressions, proposes a super image-based spatiotemporal convolution model (SISTCM) for the modality of facial expressions. This is achieved by stacking the video frames into two super images along the width and height axes, and then applying 2D convolution. IN addition, a two-stream long short-term memory (LSTM) model is presented to acquire additional global temporal cues by considering the progressive relationship of emotion expressions over time. To obtain the final recognition result, it takes as input local spatiotemporal features and clip-level emotion representations.

They propose a body gesture representation method based on body joint movement, in which body gestures are represented by 25 body joints. Using this representation result, an attention-based channel-wise convolutional model (ACCM) is used for learning joint features and recognizing emotions. Data is an essential component of emotion recognition approaches, and obtaining the data required to train machine learning algorithms is often difficult.

KEY WORDS: Video emotion recognition, Facial expression, Spatiotemporal features, Body joints, Gesture representation

Table of Contents

CHAPTER 1

SUMMARY OF THE BASE PAPER

Title: Learning facial expression and body gesture visual information for video emotion recognition

Journal Name: Expert System with Applications

Publisher: MDPI

Year: 2024

Citation Index: SCI

In recent years, study of facial expression and body gesture are two major implications in identifying human emotions. However, existing study primarily concentrates on contextual information within nearby frames and neglects the spatiotemporal relationship. Certain studies acknowledge the importance of both facial and body posture information, primarily focusing on developing fusion techniques to enhance emotion recognition performance. Nevertheless, several constraints persist that these studies did not thoroughly examine. In response, this paper revisits the study of facial expression and body gesture, proposing an improvement in video emotion recognition by extracting the spatiotemporal features.

Facial expression sequence data is distinct from static images which includes both spatial and temporal aspects. For analyzing facial expressions, the paper propose SISTCM .This model aggregates video frames into two super images in the axis of width and height ,enabling 2D convolution to capture the local Spatiotemporal facial expression features.

The shared convolution kernels across the two super images facilitate collaborative learning of local spatiotemporal features from different perspectives. Additionally, they introduce a Two-Stream LSTM model to capture global temporal cues, considering the progressive relationship of emotion expressions over time.

The Two-stream LSTM model integrates clip-level emotion representations and local Spatiotemporal features as input to generate the final recognition outcome, and blend these to enhance facial expression recognition performance. The entire facial expression recognition framework is designed as an end-to-end model and optimized through multi-stage supervised learning to enhance recognition performance.

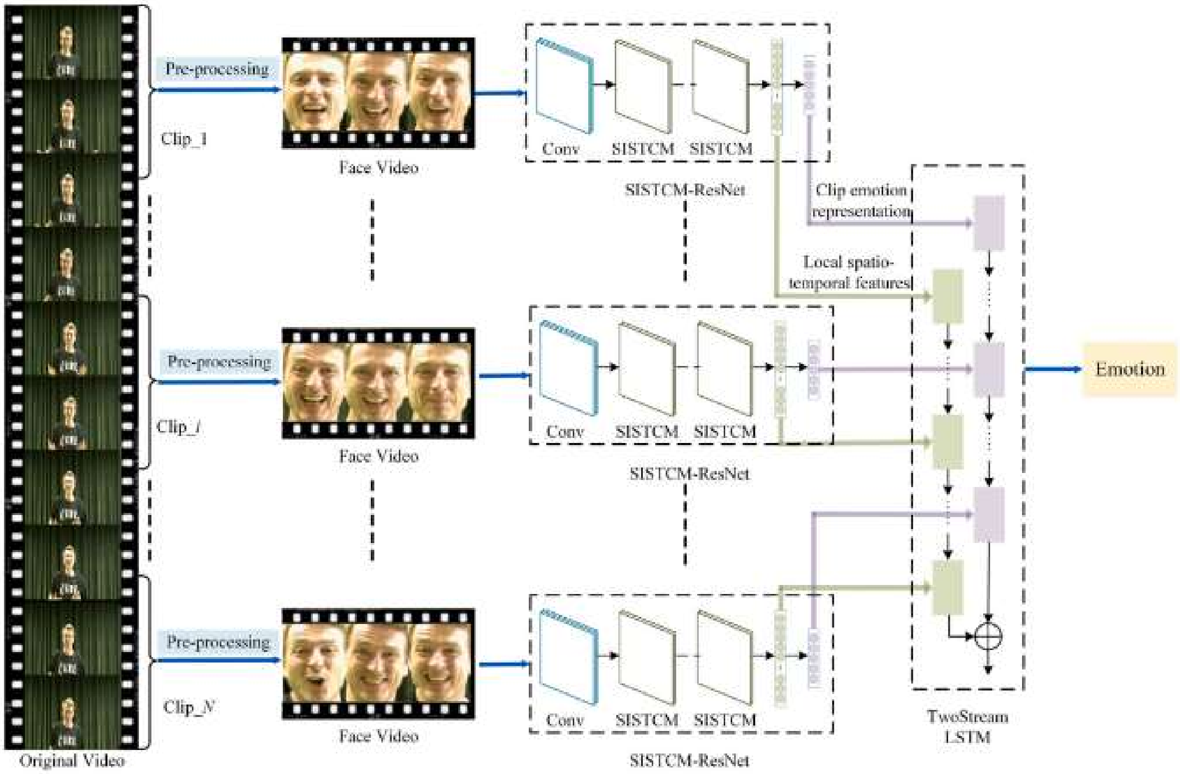


Fig 1.1. The framework of facial expression-based emotion recognition system.

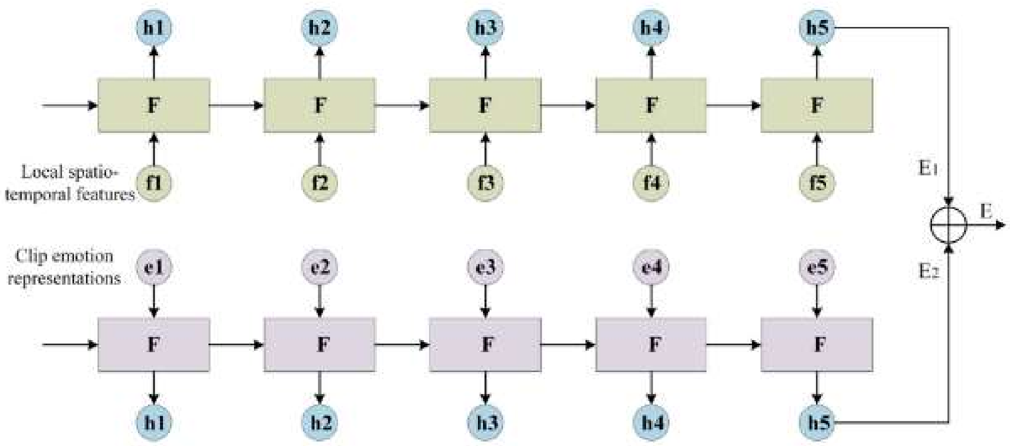


Fig 1.2 The model structure of two stream LSTM

For analyzing body gesture, they introduce body gesture representation method .The method utilizes body join movement, where the representation of body gesture is constructed using data from 25 distinct body joints. Initially this method detects the position of key points and records the changes in their positions over time.

Subsequently, the changes in joint positions are aggregated over time to capture the time-dependent relationships inherent in body gestures. Using this representation, we introduce an attention-based channel-wise convolutional model (ACCM) to learn features from the joints and recognize emotions. The ACCM effectively preserves the unique characteristics of each joint through channel-wise convolutional layers, while capitalizing on key features using an attention mechanism.

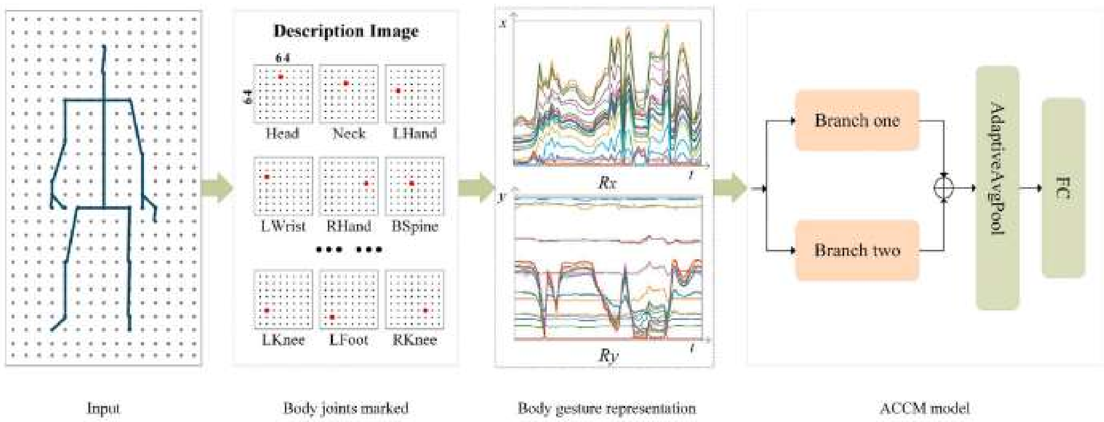


Fig 1.3 The framework of body gesture-based emotion recognition

To verify the effectiveness of the methods used for facial expression recognition and body posture recognition and the performance, we explore different fusion mechanisms. Utilizing the distinct advantages and complementary aspects of both visual modalities, our approach aims to optimize emotion recognition performance.

2. Method for facial expression:

2.1 video pre-processing:

We have got three dataset for recognizing facial expressions and they are eNTERFACE05, CK+ and Aff-Wild2. In the first stage of preprocessing we first extract all the frames from the video .The next stage is to use the DBFace model to detect and crop the frames that only contain facial parts. Further we divide each video into C clips and each clips contain certain number of frames for processing .It enables a more systematic analysis of emotional expressions while expanding the dataset and preventing over fitting to some extent.

2.2 Spatiotemporal features extraction:

To extract spatiotemporal features we use SISTCM , we sample T frames for each clip to learn about spatiotemporal relationships of complete frames which reduce the computational complexity .the concept is to conceptualize the video sequence as a stack of frames along different axes capturing spatial and temporal features.

By stacking frames along H and W dimensions, two super image are generated $H\times WT$ and $HT\times W$.This arrangement preserves spatial information from individual frames while also encoding temporal dependencies between consecutive frames.SISTCM uses 3×3 2D convolution to learn Spatiotemporal features.

In SISTCM each clip of data is in the shape of $H\times W\times T$ that servers as the input. Initially the sistcm convert the input into two super image $H\times WT$ and $HT\times W$.then 3×3 2D convolution are applied to these super image, with convolutional kernels ,to extract local spatiotemporal features efficiently.Atlast the obtained two spatiotemporal feature maps, denoted as XH and XW are first reverted to their original dimension $H\times W\times T$ Then, they are combined using a weighted fusion technique to produce the final result.

The two feature maps are connected and then processed through a fully-connected layer followed by a softmax layer to compute weights:

$$\alpha = \mathrm{Softmax}[W\alpha(XH , XW)]$$

We use the ResNet18 and SISTCM model to extract local spatiotemporal feature and clip-level emotion representation .We use FC for sentiment classification to obtain the clip-level emotion representation.

2.3: Two-stream LSTM model:

They propose a 2-stream LSTM model to learn global temporal cues for facial expression recognition. The approach involves treating the local spatiotemporal feature sequence as the feature stream and the clip-level emotion representations as the emotion stream. The feature stream is responsible for conducting emotion recognition, resulting in the emotion vector E1 derived from local spatiotemporal features.

Meanwhile, the emotion stream is trained to derive emotion vector E2 from clip-level emotion representations. Subsequently, the final video emotional state E is attained by fusing both emotion vectors.

2.4 Multi-stage supervision:

To ensure that the recognized emotion at each stage aligns closely with the label throughout the entire recognition process, we introduce a multi-stage supervised learning approach. By doing so, the model receives guidance and feedback at every stage of Processing, resulting in comprehensive training and alignment between the predicted emotions and the ground truth labels.

After obtaining the clip-level emotion representations post SISTCM-ResNet18, we compute the L1 Cross-Entropy-Loss between these representations and the corresponding labels. The L1 loss is given by:

$$L1 = -\sum \log(pc),$$

Where pc is the estimated probability for the c-th example.

Similarly, within the two-stream LSTM module, Cross-Entropy-Loss is employed to supervise the emotion vectors of both the feature stream and the emotion stream, denoted as L2 loss and L3 loss, respectively.
Thus, the final loss L is formulated as:

$$L = L1 + \lambda L2 + \mu L3$$

Where λ and μ are equilibrium coefficients, allowing for balanced weighing between the various loss components.

3. Method for body gesture

3.1Body joints marked:

The position data of key joints in each frame of the video is obtained using methods like Open Pose. The position data consist of (x,y) coordinates of each joint. Identify and select key joints relevant for gesture representation. In this paper 25 body joints are selected as key joints.

3.2 Body gesture representation:

Choose a temporal relationship function W(t) to assign weights to the descriptive images base on their timestamps.

Linear relationship : $W\ (t) = (T/\ T{-}1)\ (t-1)$

For each descriptive image It, corresponding weighted representation Gt = Tt * W(t) ,Sum up the weighted representations to obtain final body gesture representation $G = \sum Gt$. The final body gesture representation G is obtained, consisting of 25 channels corresponding to the key joints. By following these steps, a body gesture representation without a timeline is constructed.

3.3 ACCM model:

Attention based convolutional models consist of two branches. The first branch contains two blocks, each composed of a convolutional layer, and a ReLU layer .The second layer includes a channel-wise convolutional layer, attention layer, and the same blocks as the first branch. These branches operate independently and then are aggregated. Input to this ACCM is the body gesture representations obtained from the previous step. The outputs from the two branches are aggregated, possibly by concatenation, to combine the extracted features from both branches. AN adaptive AvgPooling layer is applied to adaptively reduce the spatial dimensions of the features. A softmax layer is employed for classification, producing the final emotion label. This model handles body gesture representations which preserves simplified information as compared to the original input. This model does not require pre-training, making it efficient. No of parameters is also smaller than ResNet18.

3.3.1 Channel-wise Convolutional Layer:

Each channel of the input tensor undergoes independent convolution operations. For each channel, a separate convolution is applied between the channel and its corresponding kernel. Element-wise multiplication occurs between the input region and the kernel, followed by summation to produce a single value in the output feature map. This process repeats for each spatial location dimension as the input.

3.3.2 Attention Layer:

In emotion recognition the importance of each body's joints may vary. To address this attention layer is introduced. The weight and biases of the fully-connected layers are initialized using random sampling from a uniformly distributed range $U[-a, a]$.

$$a = \sqrt{6}\ nin{+}nout$$

nin and $nout$ represent the number of input and output channels, respectively. Once the weights are obtained, they are applied to the original input. Each channel of the original input is multiplied element-wise by its corresponding attention weight to obtain the result.

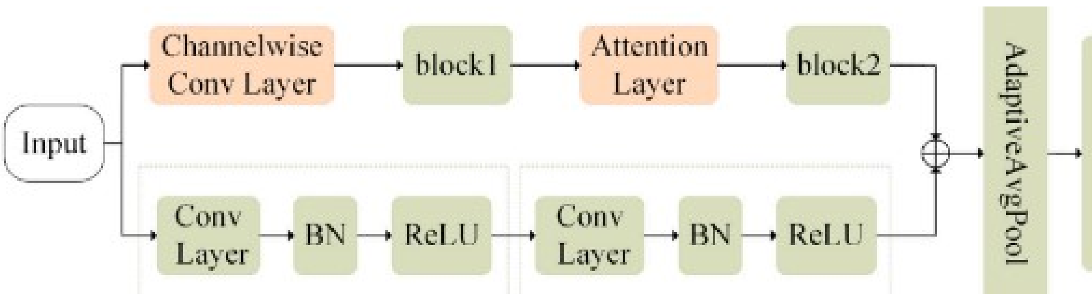


Fig 1.4 The structure of ACCM.

CHAPTER 2

MERITS AND DEMERITS

Merits:

- Consideration of both spatial and temporal relationship
- Use of two-stream LSTM improves the performance
- This ACCM model handles body gesture representations which preserves simplified information as compared to the original input.
- This ACCM model does not require pre-training, making it efficient.
- Number of parameters is also lesser as compared to ResNet18.
- Fusion of facial expression-based and body gesture-based methods, which effectively improves the accuracy of emotion recognition. This highlights the importance of leveraging multiple modalities for enhancing performance

Demerits:

- Lack of External Validation and limited use of datasets.
- Limited Discussion on Generalizability.

CHAPTER 3

IMPLEMENTATION

1 .Video Pre Processing:

```
#EXTRACT FRAMES FROM ALL VIDEOS IN eINTERFACE DATASET
import os
import cv2

# Path to the root folder of your video database
root_folder = "D:/125003447  SHREYA/III YEAR/SEMESTER
6/MiniProject/Datasets/eINTERFACE05/enterface database"

# Define the output folder where frames will be saved
output_folder = "D:/125003447  SHREYA/III YEAR/SEMESTER
6/MiniProject/Datasets/eINTERFACE05/preprocess_output"

# Define the frame skip interval (e.g., extract every 10th frame)
frame_skip = 3

# Function to extract frames from a video
def extract_frames(video_path, output_path):
    vidcap = cv2.VideoCapture(video_path)
    success, image = vidcap.read()
    count = 0
    while success:
        if count % frame_skip == 0:
            frame_output_path = os.path.join(output_path, "frame_%d.jpg" % count)
            cv2.imwrite(frame_output_path, image)
        success, image = vidcap.read()
        count += 1
    vidcap.release()

# Iterate over each subject folder
for subject_folder in os.listdir(root_folder):
    subject_path = os.path.join(root_folder, subject_folder)
    if os.path.isdir(subject_path):
        # Iterate over each emotion folder
        for emotion_folder in os.listdir(subject_path):
            emotion_path = os.path.join(subject_path, emotion_folder)
            if os.path.isdir(emotion_path):
                # Iterate over each sentence folder
                for sentence_folder in os.listdir(emotion_path):
                    sentence_path = os.path.join(emotion_path, sentence_folder)
                    if os.path.isdir(sentence_path):
                        # Iterate over each .avi video file in the sentence folder
                        for video_file in os.listdir(sentence_path):
                            if video_file.endswith(".avi"):
                                video_path = os.path.join(sentence_path, video_file)
                                output_path = os.path.join(output_folder,
subject_folder, emotion_folder, sentence_folder, os.path.splitext(video_file)[0])
                                os.makedirs(output_path, exist_ok=True)
                                extract_frames(video_path, output_path)
```

2 . Facial expression recognition:

SISTCM-LSTM Model:

```
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
from sklearn.metrics import confusion_matrix
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from PIL import Image
import os

# Define transforms for the input images
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize images to 224x224
    transforms.ToTensor(),           # Convert images to tensors
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  #
Normalize images
])
class CustomDataset(Dataset):
    def __init__(self, root_dir):
        self.root_dir = root_dir
        self.subjects = os.listdir(root_dir)
        self.data = self._load_data()
        self.classes = self._get_classes()  # Get unique classes
 def _load_data(self):
        data = []
        for subject in self.subjects:
emotions = os.listdir(os.path.join(self.root_dir, subject))
            for emotion in emotions:
                sentences = os.listdir(os.path.join(self.root_dir, subject, emotion))
                for sentence in sentences:
                    images = os.listdir(os.path.join(self.root_dir, subject, emotion,
sentence))
                    for i, image_file in enumerate(images):
                        image_path = os.path.join(self.root_dir, subject, emotion,
sentence, image_file)
                        data.append((image_path, i, emotion))  # (image path, sentence
index, emotion label)
        return data

    def _get_classes(self):
        classes = []
        for _, _, emotion in self.data:
            if emotion not in classes:
                classes.append(emotion)
        return classes
     def __len__(self):
        return len(self.data)
     def __getitem__(self, idx):
        image_path, sentence_idx, emotion = self.data[idx]
        try:
            image = Image.open(image_path).convert('RGB')
```

```
        except (FileNotFoundError, PermissionError) as e:
            print(f"Error loading image: {e}")
            # Return None or a placeholder image here, or skip this sample
            return None, None, None

        # Perform any necessary preprocessing on the image here
        return image, sentence_idx, self.classes.index(emotion)  # Return class index
instead of emotion string

# Define custom collate function
def custom_collate(batch):
    images = []
    labels = []
    for item in batch:
        image, sentence_idx, label = item
        if image is not None:
            images.append(image)
            labels.append(label)
    # Apply transforms
    images = [transform(image) for image in images]
    # Stack images and labels
    images = torch.stack(images)
    labels = torch.tensor(labels)
    return images, labels
# Example usage
dataset = CustomDataset(root_dir='preprocess_output')
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size,
test_size])
train_dataloader = DataLoader(train_dataset, batch_size=32, shuffle=True,
collate_fn=custom_collate)
test_dataloader = DataLoader(test_dataset, batch_size=32, shuffle=False,
collate_fn=custom_collate)

# Define SISTCM model
class SISTCM(nn.Module):
    def __init__(self, num_classes):
        super(SISTCM, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=7, stride=2,
padding=3)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.sistcm1 = self._make_layer(64, 64, 2, 1)
        self.sistcm2 = self._make_layer(64, 128, 2, 2)
        self.sistcm3 = self._make_layer(128, 256, 2, 2)
        self.sistcm4 = self._make_layer(256, 512, 2, 2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)
        self.spatio_temporal_fc = nn.Linear(512, num_classes)  # Adjust output size if
needed

    def _make_layer(self, in_channels, out_channels, num_blocks, stride):
        layers = []
        for _ in range(num_blocks):
            layers.append(nn.Conv2d(in_channels, out_channels, kernel_size=3,
stride=stride, padding=1))
```

```
 layers.append(nn.BatchNorm2d(out_channels))
            layers.append(nn.ReLU(inplace=True))
            in_channels = out_channels
            stride = 1  # Reset stride for subsequent blocks
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.maxpool(x)
        x = self.sistcm1(x)
        x = self.sistcm2(x)
        x = self.sistcm3(x)
        spatio_temporal_feature_sequence = self.sistcm4(x)  # Output for spatio-temporal
feature sequence
        x = self.avgpool(spatio_temporal_feature_sequence)
        x = torch.flatten(x, 1)

        # Output for clip-level emotion representation sequence
        clip_level_emotion_sequence = self.fc(x)

        # Additional output for spatio-temporal feature sequence
        spatio_temporal_output = self.spatio_temporal_fc(x)

        return spatio_temporal_feature_sequence, clip_level_emotion_sequence,
spatio_temporal_output
class TwoStreamLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(TwoStreamLSTM, self).__init__()
self.lstm_feature = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
batch_first=True)
        self.lstm_emotion = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
batch_first=True)
        self.fc = nn.Linear(hidden_size * 2, num_classes)

    def forward(self, feature_sequence, emotion_sequence):
        # Assuming feature_sequence and emotion_sequence are both 3D tensors
        _, (h_feature, _) = self.lstm_feature(feature_sequence)
        _, (h_emotion, _) = self.lstm_emotion(emotion_sequence)

        # Get the last hidden state (output) of each LSTM
        h_feature_last = h_feature[-1]
        h_emotion_last = h_emotion[-1]

        # Concatenate the last hidden states
        fused_features = torch.cat((h_feature_last, h_emotion_last), dim=1)

        output = self.fc(fused_features)
        return output
# Define the number of classes
num_classes = len(dataset.classes)  # Number of classes in the dataset
# Instantiate SISTCM model with the specified number of classes
sistcm_model = SISTCM(num_classes)
# Instantiate Two-stream LSTM model
input_size = 512  # Size of spatio-temporal feature sequence
hidden_size = 128
num_classes = len(dataset.classes)  # Number of classes in the dataset
two_stream_lstm_model = TwoStreamLSTM(input_size, hidden_size, num_classes)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
                                  sistcm_model.to(device)
```

```
two_stream_lstm_model.to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(list(sistcm_model.parameters()) +
list(two_stream_lstm_model.parameters()), lr=0.001)


# Evaluation function
def evaluate_with_confusion_matrix(model, dataloader):
    model.eval()
    all_preds = []
    all_labels = []
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in dataloader:
            images = images.to(device)
            labels = labels.to(device)
            spatio_temporal_feature_sequence, clip_level_emotion_sequence,
spatio_temporal_output = sistcm_model(images)
            batch_size, channels, height, width = spatio_temporal_feature_sequence.size()
spatio_temporal_feature_sequence = spatio_temporal_feature_sequence.view(batch_size,
channels, -1)
            clip_level_emotion_sequence =
clip_level_emotion_sequence.unsqueeze(1).expand(-1,
spatio_temporal_feature_sequence.size(2), -1)
            lstm_output = two_stream_lstm_model(spatio_temporal_feature_sequence,
clip_level_emotion_sequence)
            _, predicted = torch.max(lstm_output.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    accuracy = correct / total
    cm = confusion_matrix(all_labels, all_preds)
    return accuracy, cm
# Initialize lists to store true labels and predicted labels
true_labels = []
predicted_labels = []
# Training loop with validation
num_epochs = 4
for epoch in range(num_epochs):
    sistcm_model.train()
    running_loss = 0.0
    correct_train = 0
    total_train = 0
    for i, (images, labels) in enumerate(train_dataloader):
        images = images.to(device)
        labels = labels.to(device)
        spatio_temporal_feature_sequence, clip_level_emotion_sequence,
spatio_temporal_output = sistcm_model(images)
           batch_size, channels, height, width = spatio_temporal_feature_sequence.size()
spatio_temporal_feature_sequence = spatio_temporal_feature_sequence.view(batch_size,
channels, -1)
        clip_level_emotion_sequence = clip_level_emotion_sequence.unsqueeze(1).expand(-1,
spatio_temporal_feature_sequence.size(2), -1)
```

```
        lstm_output = two_stream_lstm_model(spatio_temporal_feature_sequence,
clip_level_emotion_sequence)
        loss = criterion(lstm_output, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
       # Accumulate statistics
        running_loss += loss.item()
        _, predicted = torch.max(lstm_output.data, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()
       print_freq=100
       if (i+1) % print_freq== 0:
             print(f'Epoch [{epoch+1}/{num_epochs}], Step
[{i+1}/{len(train_dataloader)}], Loss: {running_loss/print_freq:.4f}, Accuracy:
{(correct_train/total_train) * 100:.2f}%')
             running_loss = 0.0
             correct_train = 0
             total_train = 0
        true_labels.extend(labels.cpu().numpy())
        predicted_labels.extend(predicted.cpu().numpy())
val_accuracy, confusion = evaluate_with_confusion_matrix(sistcm_model, test_dataloader)
print(f'Validation Accuracy: {val_accuracy:.4f}')


print('Confusion Matrix:')
print(confusion)
conf_matrix = confusion_matrix(true_labels, predicted_labels)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=dataset.classes,
yticklabels=dataset.classes)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
print('Training finished!')
```

3 . Body Gesture recognition :

```
import os
import torch
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Dataset, random_split
from PIL import Image
import torch.nn as nn
import numpy as np

class CNNWithAttention(nn.Module):
    def __init__(self, input_channels, num_classes):
        super(CNNWithAttention, self).__init__()

        # Branch 1
        self.branch1_conv1 = nn.Conv2d(input_channels, 64, kernel_size=3, padding=1)
        self.branch1_bn1 = nn.BatchNorm2d(64)
        self.branch1_relu1 = nn.ReLU()
        self.branch1_conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
        self.branch1_bn2 = nn.BatchNorm2d(64)
        self.branch1_relu2 = nn.ReLU()

        # Branch 2
        self.branch2_conv = nn.Conv2d(input_channels, 64, kernel_size=3, padding=1)
        self.branch2_att = nn.Sequential(
            nn.Conv2d(64, 1, kernel_size=1),
            nn.Sigmoid()
```

```
        self.branch2_conv1 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
        self.branch2_bn1 = nn.BatchNorm2d(64)
        self.branch2_relu1 = nn.ReLU()
        self.branch2_conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
        self.branch2_bn2 = nn.BatchNorm2d(64)
        self.branch2_relu2 = nn.ReLU()

        # Aggregation and Classification
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(128, num_classes)  # Adjusted input size for fully connected
layer
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        # Branch 1
        out1 = self.branch1_relu1(self.branch1_bn1(self.branch1_conv1(x)))
        out1 = self.branch1_relu2(self.branch1_bn2(self.branch1_conv2(out1)))

        # Branch 2
        out2 = self.branch2_conv(x)
        att_weights = self.branch2_att(out2)
        out2 = torch.mul(out2, att_weights)
        out2 = self.branch2_relu1(self.branch2_bn1(self.branch2_conv1(out2)))
        out2 = self.branch2_relu2(self.branch2_bn2(self.branch2_conv2(out2)))

        # Aggregation
        out = torch.cat((out1, out2), dim=1)
        out = self.avgpool(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        out = self.softmax(out)
        return out

class EmotionDataset(Dataset):
    def __init__(self, root_dir, transform=None, stride=1):
        self.root_dir = root_dir
        self.transform = transform
        self.classes = os.listdir(root_dir)
        self.class_to_idx = {cls_name: idx for idx, cls_name in enumerate(self.classes)}
        self.images = self._load_images(stride)
def _load_images(self, stride):
        images = []
        for class_name in self.classes:
            class_dir = os.path.join(self.root_dir, class_name)
            image_list = sorted(os.listdir(class_dir))
            for i in range(0, len(image_list), stride):
                img_path = os.path.join(class_dir, img_name)
                images.append((img_path, self.class_to_idx[class_name]))
        return images

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img_path, label = self.images[idx]
        img = Image.open(img_path).convert('RGB')
        if self.transform:
            img = self.transform(img)
        return img, label


# Define data transformation
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
# Define dataset path
dataset_path = "Datasets/RAF_DB/train"
dataset = EmotionDataset(dataset_path, transform=transform)
```

```
# Split dataset into training and testing sets
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

# Define DataLoader for training and testing sets
batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size)

# Instantiate the model
input_channels = 3
num_classes = len(dataset.classes)
model = ACCM(input_channels, num_classes)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for i, (inputs, labels) in enumerate(train_loader):
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
       # Calculate training accuracy
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        running_loss += loss.item()
        if i % 5== 0:  # Print every 10 mini-batches
            print('[%d, %5d] loss: %.3f | accuracy: %.2f %%' % (epoch + 1, i + 1,
running_loss / 10, 100 * correct / total))
            running_loss = 0.0
# Evaluation on the testing set
model.eval()
total = 0
with torch.no_grad():
    for inputs, labels in test_loader:
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print('Finished Epoch %d. Test Accuracy: %.2f %%' % (epoch + 1, accuracy))

print('Finished Training')
```

4.Bimodal Fusion:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset, random_split
from torchvision import transforms
from PIL import Image
import os
class SISTCM(nn.Module):
    def __init__(self, num_classes):
        super(SISTCM, self).__init__()
```

```
            self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=7,
                                   stride=2, padding=3)


        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.sistcm1 = self._make_layer(64, 64, 2, 1)
        self.sistcm2 = self._make_layer(64, 128, 2, 2)
        self.sistcm3 = self._make_layer(128, 256, 2, 2)
        self.sistcm4 = self._make_layer(256, 512, 2, 2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)

        # Additional layers for spatio-temporal feature sequence output
        self.spatio_temporal_fc = nn.Linear(512, num_classes)  # Adjust output size
if needed

    def _make_layer(self, in_channels, out_channels, num_blocks, stride):
        layers = []
        for _ in range(num_blocks):
            layers.append(nn.Conv2d(in_channels, out_channels, kernel_size=3,
stride=stride, padding=1))
            layers.append(nn.BatchNorm2d(out_channels))
            layers.append(nn.ReLU(inplace=True))
            in_channels = out_channels
            stride = 1  # Reset stride for subsequent blocks
        return nn.Sequential(*layers)

# Define a simple LSTM model
class TwostreamLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, num_classes=3):
        super(SimpleLSTM, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        batch_size, _, _, _ = x.size()
        x = x.view(batch_size, -1, x.size(3))

        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.lstm(x, (h0, c0))
        out = self.fc(out[:, -1, :])
        return out
class ACCM(nn.Module):
    def __init__(self, num_classes=3):
        super(AnotherConvolutionalClassifier, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
```

```
            self.classifier = nn.Sequential(
            nn.Linear(64 * 56 * 56, 128),
            nn.ReLU(inplace=True),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

# Define FusionModel
class FusionModel(nn.Module):
    def __init__(self, lstm_model, cnn_model):
        super(FusionModel, self).__init__()
        self.lstm_model = lstm_model
        self.cnn_model = cnn_model

    def forward(self, x):
        lstm_output = self.lstm_model(x)
        cnn_output = self.cnn_model(x)
        fused_output = torch.max(lstm_output, cnn_output)
        return fused_output

# Define custom dataset class
# Define custom dataset class
class EmotionDataset(Dataset):
    def __init__(self, root_dir, transform=None, stride=1,
max_images_per_folder=100):
        self.root_dir = root_dir
        self.transform = transform
        self.classes = ['1', '2', '3']  # Labels from folders
        self.class_to_idx = {cls_name: idx for idx, cls_name in
enumerate(self.classes)}
        self.images = self._load_images(stride, max_images_per_folder)
def _load_images(self, stride, max_images_per_folder):
        images = []
        for class_name in self.classes:
            class_dir = os.path.join(self.root_dir, class_name)
            if not os.path.isdir(class_dir):
                continue  # Skip if the directory doesn't exist
            image_list = sorted(os.listdir(class_dir))[:max_images_per_folder]
            for i in range(0, len(image_list), stride):
                img_name = image_list[i]
                img_path = os.path.join(class_dir, img_name)
                images.append((img_path, self.class_to_idx[class_name]))
        return images

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img_path, label = self.images[idx]
        img = Image.open(img_path).convert('RGB')
        if self.transform:
            img = self.transform(img)
        return img, label
```

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Define dataset path
dataset_path = "DATASET/train"
custom_dataset = EmotionDataset(root_dir=dataset_path, transform=transform,
max_images_per_folder=100)
# Split dataset into training and testing sets
train_size = int(0.8 * len(custom_dataset))
test_size = len(custom_dataset) - train_size
train_dataset, test_dataset = random_split(custom_dataset, [train_size, test_size])
# Define DataLoader for training and testing sets
batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
# Define the models
lstm_model = TwostreamLSTM(input_size=224, hidden_size=64, num_layers=1,
num_classes=3)
cnn_model = ACCM(num_classes=3)
# Define your fusion model
fusion_model = FusionModel(lstm_model, cnn_model)
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(fusion_model.parameters(), lr=0.001)
num_epochs = 10
for epoch in range(num_epochs):
    fusion_model.train()
    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0
for images, labels in train_loader:
        optimizer.zero_grad()
       outputs = fusion_model(images)
       loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs, 1)
        correct_predictions += (predicted == labels).sum().item()
        total_samples += labels.size(0)
    epoch_loss = running_loss / total_samples
    epoch_accuracy = correct_predictions / total_samples
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}, Accuracy:
{epoch_accuracy:.4f}")
fusion_model.eval()
correct_predictions = 0
total_samples = 0
with torch.no_grad():
    for images, labels in test_loader:
        outputs = fusion_model(images)
        _, predicted = torch.max(outputs, 1)
        correct_predictions += (predicted == labels).sum().item()
        total_samples += labels.size(0)
test_accuracy = correct_predictions / total_samples
print(f"Test Accuracy: {test_accuracy:.4f}")
```

CHAPTER 4

SANPSHOT

1.Sample Output of preprocessing:

Fig 4.1 sample output of video preprocessing

2. Facial recognition output:

Output for CK+ Dataset:

```
Epoch [1/15], Step [10/25], Loss: 1.8117, Accuracy: 37.50%
Epoch [1/15], Step [20/25], Loss: 1.4158, Accuracy: 50.62%
Epoch [2/15], Step [10/25], Loss: 0.9776, Accuracy: 63.75%
Epoch [2/15], Step [20/25], Loss: 0.8937, Accuracy: 65.31%
Epoch [3/15], Step [10/25], Loss: 0.7203, Accuracy: 74.69%
Epoch [3/15], Step [20/25], Loss: 0.6846, Accuracy: 76.25%
Epoch [4/15], Step [10/25], Loss: 0.5782, Accuracy: 81.56%
Epoch [4/15], Step [20/25], Loss: 0.4955, Accuracy: 84.06%
Epoch [5/15], Step [10/25], Loss: 0.3967, Accuracy: 87.81%
Epoch [5/15], Step [20/25], Loss: 0.3250, Accuracy: 88.75%
Epoch [6/15], Step [10/25], Loss: 0.3929, Accuracy: 87.19%
Epoch [6/15], Step [20/25], Loss: 0.3492, Accuracy: 90.94%
Epoch [7/15], Step [10/25], Loss: 0.2469, Accuracy: 93.44%
Epoch [7/15], Step [20/25], Loss: 0.1632, Accuracy: 94.38%
Epoch [8/15], Step [10/25], Loss: 0.0870, Accuracy: 97.81%
Epoch [8/15], Step [20/25], Loss: 0.1118, Accuracy: 96.56%
Epoch [9/15], Step [10/25], Loss: 0.1590, Accuracy: 94.38%
Epoch [9/15], Step [20/25], Loss: 0.2239, Accuracy: 92.19%
Epoch [10/15], Step [10/25], Loss: 0.1285, Accuracy: 95.31%
Epoch [10/15], Step [20/25], Loss: 0.1367, Accuracy: 95.00%
Epoch [11/15], Step [10/25], Loss: 0.0474, Accuracy: 98.75%
Epoch [11/15], Step [20/25], Loss: 0.0610, Accuracy: 98.12%
Epoch [12/15], Step [10/25], Loss: 0.0639, Accuracy: 97.50%
Epoch [12/15], Step [20/25], Loss: 0.0398, Accuracy: 99.38%
Epoch [13/15], Step [10/25], Loss: 0.1438, Accuracy: 95.94%
Epoch [13/15], Step [20/25], Loss: 0.0707, Accuracy: 97.81%
Epoch [14/15], Step [10/25], Loss: 0.0275, Accuracy: 99.38%
Epoch [14/15], Step [20/25], Loss: 0.0720, Accuracy: 97.50%
Epoch [15/15], Step [10/25], Loss: 0.0285, Accuracy: 99.69%
Epoch [15/15], Step [20/25], Loss: 0.0464, Accuracy: 98.12%
Test Accuracy: 0.9695
```

Fig 4.2 output of ck+ dataset

Confusion matrix:

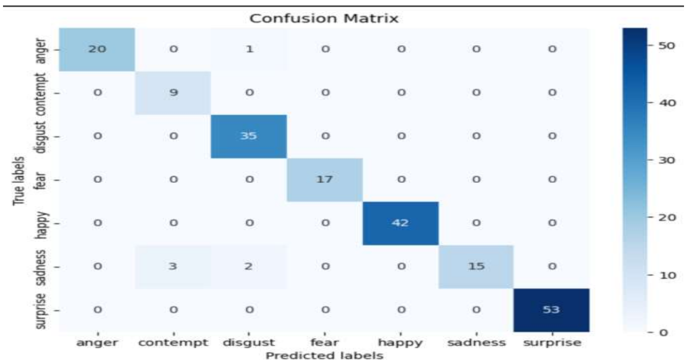


Fig 4.3 confusion matrix of ck+ dataset

Output for eINTERFACE:

```
Epoch [1/4], Step [100/765], Loss: 1.7888, Accuracy: 19.31%
Epoch [1/4], Step [200/765], Loss: 1.7798, Accuracy: 22.50%
Epoch [1/4], Step [300/765], Loss: 1.7205, Accuracy: 26.75%
Epoch [1/4], Step [400/765], Loss: 1.6174, Accuracy: 31.72%
Epoch [1/4], Step [500/765], Loss: 1.5118, Accuracy: 37.03%
Epoch [1/4], Step [600/765], Loss: 1.3531, Accuracy: 45.75%
Epoch [1/4], Step [700/765], Loss: 1.2460, Accuracy: 50.53%
Epoch [2/4], Step [100/765], Loss: 1.0097, Accuracy: 60.66%
Epoch [2/4], Step [200/765], Loss: 0.9326, Accuracy: 64.75%
Epoch [2/4], Step [300/765], Loss: 0.7862, Accuracy: 70.97%
Epoch [2/4], Step [400/765], Loss: 0.7765, Accuracy: 70.88%
Epoch [2/4], Step [500/765], Loss: 0.6539, Accuracy: 75.41%
Epoch [2/4], Step [600/765], Loss: 0.6350, Accuracy: 77.22%
Epoch [2/4], Step [700/765], Loss: 0.5940, Accuracy: 78.47%
Epoch [3/4], Step [100/765], Loss: 0.4907, Accuracy: 82.38%
Epoch [3/4], Step [200/765], Loss: 0.4501, Accuracy: 83.62%
Epoch [3/4], Step [300/765], Loss: 0.4236, Accuracy: 85.56%
Epoch [3/4], Step [400/765], Loss: 0.4205, Accuracy: 84.38%
Epoch [3/4], Step [500/765], Loss: 0.3661, Accuracy: 86.38%
Epoch [3/4], Step [600/765], Loss: 0.3481, Accuracy: 87.12%
Epoch [3/4], Step [700/765], Loss: 0.3773, Accuracy: 86.75%
Epoch [4/4], Step [100/765], Loss: 0.3226, Accuracy: 89.06%
Epoch [4/4], Step [200/765], Loss: 0.2797, Accuracy: 90.53%
Epoch [4/4], Step [300/765], Loss: 0.3000, Accuracy: 89.03%
Epoch [4/4], Step [400/765], Loss: 0.2555, Accuracy: 91.47%
Epoch [4/4], Step [500/765], Loss: 0.2634, Accuracy: 90.88%
Epoch [4/4], Step [600/765], Loss: 0.2525, Accuracy: 91.00%
Epoch [4/4], Step [700/765], Loss: 0.2305, Accuracy: 92.47%
Validation Accuracy: 0.9198
```

Fig 4.4 Output for eINTERFACE:

Confusion matrix :

Fig 4.5 Confusion matrix of eINTERFACE

Output for FER2013:

```
Epoch [5/10], Step [200/718], Loss: 1.2152, Accura
Epoch [5/10], Step [400/718], Loss: 1.2174, Accura
Epoch [5/10], Step [600/718], Loss: 1.2068, Accura
Epoch [6/10], Step [200/718], Loss: 1.1491, Accura
Epoch [6/10], Step [400/718], Loss: 1.1218, Accura
Epoch [6/10], Step [600/718], Loss: 1.1380, Accura
Epoch [7/10], Step [200/718], Loss: 1.0733, Accura
Epoch [7/10], Step [400/718], Loss: 1.0674, Accura
Epoch [7/10], Step [600/718], Loss: 1.0847, Accura
Epoch [8/10], Step [200/718], Loss: 0.9985, Accura
Epoch [8/10], Step [400/718], Loss: 0.9998, Accura
Epoch [8/10], Step [600/718], Loss: 1.0014, Accura
Epoch [9/10], Step [200/718], Loss: 0.8800, Accura
Epoch [9/10], Step [400/718], Loss: 0.9152, Accura
Epoch [9/10], Step [600/718], Loss: 0.9440, Accura
```

Fig 4.6 Output for FER2013

Confusion matrix :

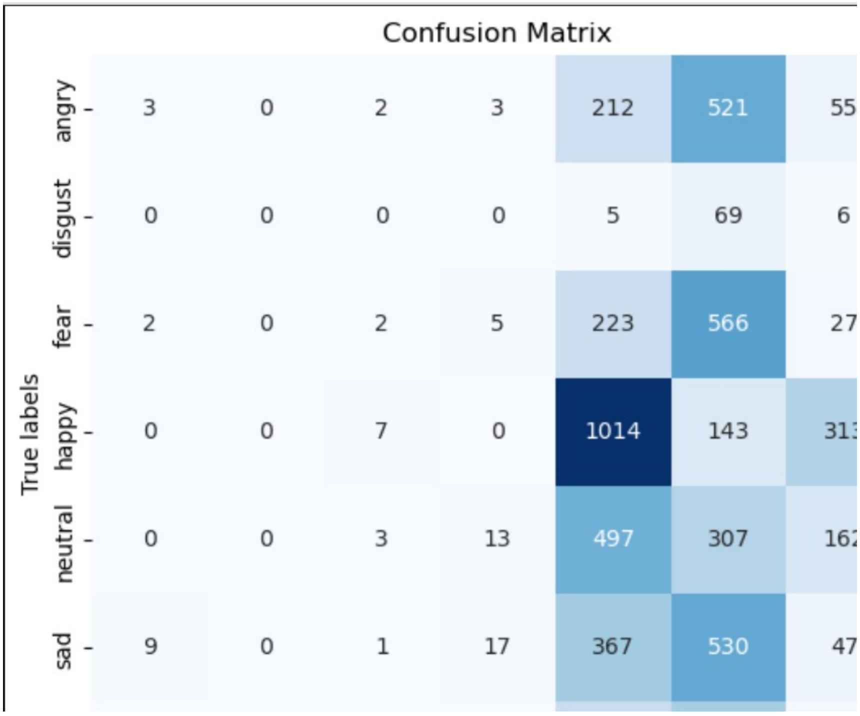


Fig 4.7 Confusion matrix of FER2013

BRED dataset output (CSV) :

```
Epoch [1/10], Training Loss: 1.7564, Training Accuracy: 37.84%
Epoch [2/10], Training Loss: 1.7738, Training Accuracy: 40.36%
Epoch [3/10], Training Loss: 1.7602, Training Accuracy: 45.84%
Epoch [4/10], Training Loss: 1.8026, Training Accuracy: 50.99%
Epoch [5/10], Training Loss: 1.5455, Training Accuracy: 56.14%
Epoch [6/10], Training Loss: 1.5614, Training Accuracy: 56.85%
Epoch [7/10], Training Loss: 1.5170, Training Accuracy: 62.21%
Epoch [8/10], Training Loss: 1.7099, Training Accuracy: 64.44%
Epoch [9/10], Training Loss: 1.6369, Training Accuracy: 66.35%
Epoch [10/10], Training Loss: 1.6241, Training Accuracy: 68.23%
Testing Accuracy: 67.55%
```

Fig 3.7 BRED dataset output (CSV)

Confusion matrix:

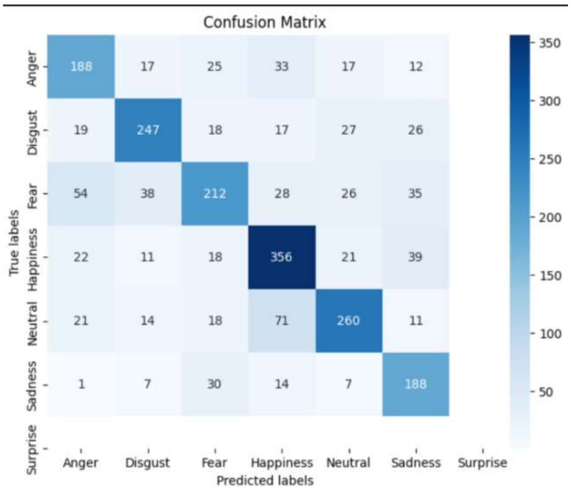


Fig 3.8 Confusion matrix of BRED dataset

3. Body Gesture recognition output:

Output for RAF_DB:

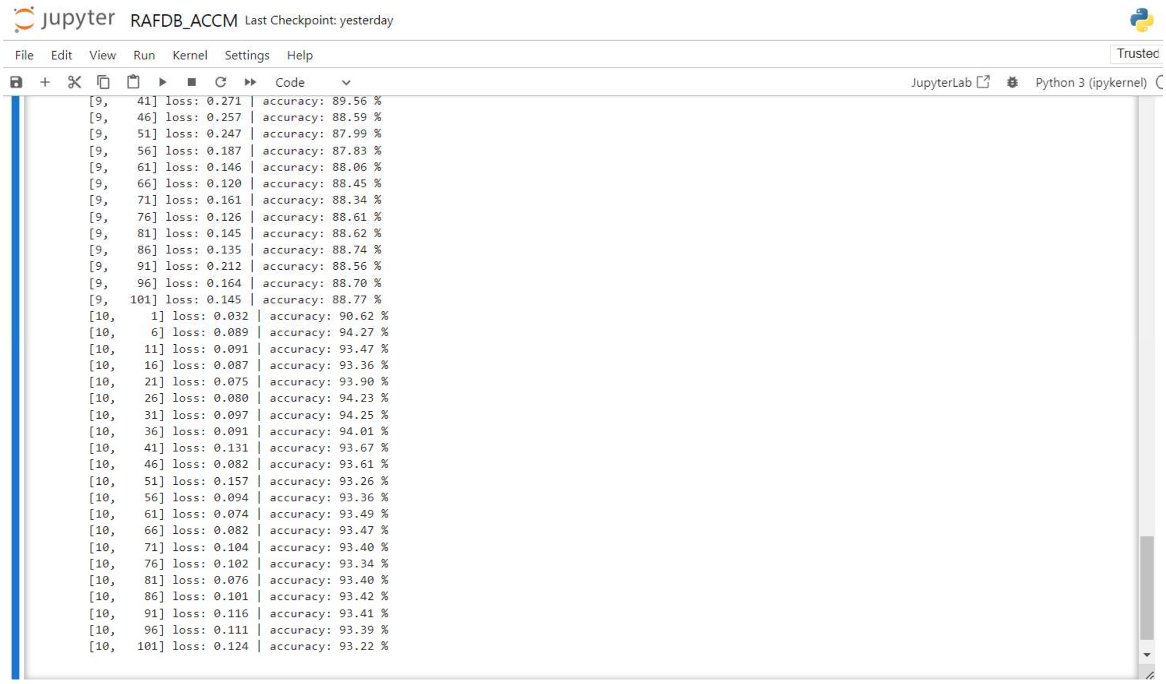


Fig 4.9 Output for RAF_DB

Confusion matrix:

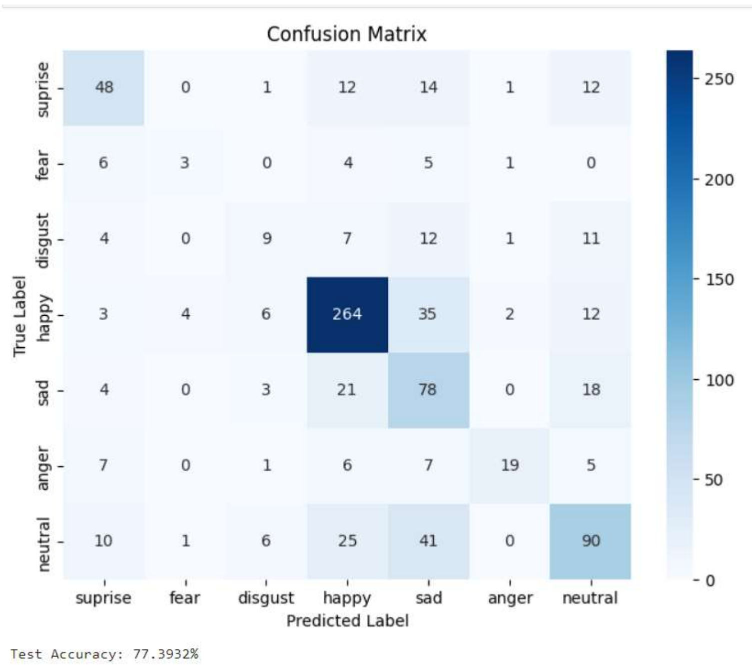


Fig 4.10 Confusion matrix of RAF_DB

4. Bimodal fusion output:

Bimodal fusion maximum output:

```
Epoch [1/10], Loss: 2.0924, Accuracy: 0.
Epoch [2/10], Loss: 1.1024, Accuracy: 0.
Epoch [3/10], Loss: 1.0140, Accuracy: 0.
Epoch [4/10], Loss: 0.9354, Accuracy: 0.
Epoch [5/10], Loss: 0.8562, Accuracy: 0.
Epoch [6/10], Loss: 0.7420, Accuracy: 0.
Epoch [7/10], Loss: 0.5960, Accuracy: 0.
Epoch [8/10], Loss: 0.4461, Accuracy: 0.
Epoch [9/10], Loss: 0.3068, Accuracy: 0.
```

Fig 4.11 Bimodal fusion maximum output

Bimodal fusion average output:

```
Epoch [1/10], Loss: 1.9392, Accuracy: 0.
Epoch [2/10], Loss: 1.0743, Accuracy: 0.
Epoch [3/10], Loss: 1.0579, Accuracy: 0.
Epoch [4/10], Loss: 0.9866, Accuracy: 0.
Epoch [5/10], Loss: 0.9335, Accuracy: 0.
Epoch [6/10], Loss: 0.8715, Accuracy: 0.
Epoch [7/10], Loss: 0.7362, Accuracy: 0.
Epoch [8/10], Loss: 0.6231, Accuracy: 0.
Epoch [9/10], Loss: 0.5143, Accuracy: 0.
```

Fig 4.12 Bimodal fusion average output

CHAPTER 5

Conclusion:

In this study, we leverage the distinct visual characteristics of facial expressions and body gestures and propose suitable methods for video emotion recognition. For facial expression sequences, we introduced the SISTCM model, which extracts local spatiotemporal features and learns clip-level emotional states. We then utilize a two-stream LSTM to further capture global temporal cues and refine emotion recognition. By fusing features and emotion pathways, the two-stream LSTM enhances accuracy significantly. For body gesture sequences, we present a body gesture representation method to represent gesture changes using joint information and develop the ACCM model for emotion recognition. This representation simplifies gesture information, reducing training complexity, while ACCM maximizes the advantages of key channel features and preserves their independence. Extensive experiments demonstrate the superiority of our proposed unimodal emotion recognition methods over alternatives. Furthermore, integrating facial expression and body gesture methods effectively enhances emotion recognition accuracy.

FUTURE WORK:

Human interactions rely heavily on emotions, and AI's ability to recognize and respond to emotions opens up numerous possibilities. By intelligently analyzing the emotional cues in videos, it becomes possible to better understand user emotions and enhance services, thereby boosting marketing competitiveness. Here are a few potential directions for the further scope of learning facial expression and body posture.

1. In future, we aim to disentangle identity attributes and prioritize the analysis of general emotional features.

2. Given that certain datasets may exclusively include upper-body data, we also conduct experiments using upper-body joint information. Despite a slight decrease in performance due to the reduced amount of information, the approach remains effective for upper-body analysis. We can add the reduced amount of information and improve our analysis for body posture.

3. We are using more dataset and it take times to run 50 epochs for each model and their corresponding dataset .In future we can improve processing of dataset.

CHAPTER 6

REFERENCES

1. Abdullah, S. M. S., & Abdulazeez, A. M. (2021). Facial expression recognition based on deep learning convolution neural network: A review. Journal of Soft Computing and Data Mining, 2(1), 53–65. http://dx.doi.org/10.30880/jscdm.2021.02.01.006.

2. Ambady, N., & Rosenthal, R. (1992). Thin slices of expressive behavior as predictors of interpersonal consequences: A meta-analysis. Psychological Bulletin, 111(2), 256–274, https://psycnet.apa.org/doi/10.1037/0033-2909.111.2.256.

3. Aouayeb, M., Hamidouche, W., Soladie, C., Kpalma, K., & Seguier, R. (2021). Learning vision transformers with squeeze and excitation for facial expression recognition. https://arxiv.org/abs/2107.03107, arXiv preprint arXiv:2107.03107.

4. Atanassov, A. V., Pilev, D. I., Tomova, F. N., & Kuzmanova, V. D. (2021). Hybrid system for emotion recognition based on facial expressions and body gesture recognition. In 2021 International conference automatics and informatics (pp. 135–140). IEEE, https://ieeexplore.ieee.org/document/9639829/.

5. Avola, D., Cinque, L., Fagioli, A., Foresti, G. L., & Massaroni, C. (2020). Deep temporal analysis for non-acted body affect recognition. IEEE Transactions on Affective Computing, 13(3), 1366–1377. http://dx.doi.org/10.1109/TAFFC.2020.3003816.

6. Avots, E., Sapiński, T., Bachmann, M., & Kamińska, D. (2019). Audiovisual emotion recognition in wild. Machine Vision and Applications, 30(5), 975–985. http://dx.doi.org/10.1007/s00138-018-0960-9.

7. Camurri, A., Lagerlöf, I., & Volpe, G. (2003). Recognizing emotion from dance movement: comparison of spectator recognition and automated techniques. International Journal of Human-Computer Studies, 59(1–2), 213–225.5. http://dx.doi.org/10.1016/S1071-5819(03)00050-8.

8. Cao, Z., Simon, T., Wei, S.-E., & Sheikh, Y. (2017). Realtime multi-person 2d pose estimation using part affinity fields. In IEEE conference on computer vision and pattern recognition (pp. 7291–7299). IEEE, http://dx.doi.org/10.48550/arXiv.1611.08050

9. Chen, J., Chen, Z., Chi, Z., & Fu, H. (2018). Facial expression recognition in video with multiple feature fusion. IEEE Transactions on Affective Computing, 9(1), 38–50. http://dx.doi.org/10.1109/TAFFC.2016.2593719

CHAPTER 7

Appendix

Base paper:

Jie Wei , Guanyu Hu , Xinyu Yang , Anh Tuan Luu and Yizhou Dong, "Learning facial expression and body gesture visual information for video emotion recognition" ,Expert Systems with Applications,Volume 237, Part A, 1 March 2024, 121419.

Link:

https://www.sciencedirect.com/science/article/pii/S0957417423019218

The Report is Generated by DrillBit Plagiarism Detection Software

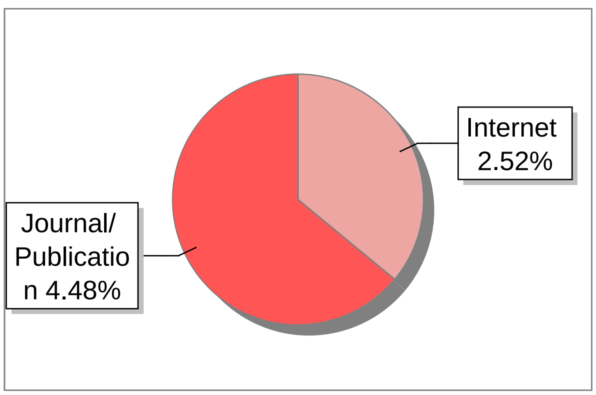
Submission Information

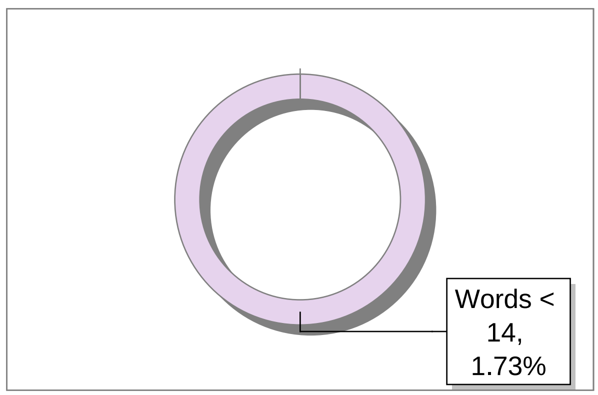
Author Name	Divya
Title	Mini1
Paper/Submission ID	1682105
Submitted by	priya.ayyagari@it.sastra.edu
Submission Date	2024-04-22 16:52:50
Total Pages	8
Document type	Project Work

Result Information

Similarity **7 %**

1 10 20 30 40 50 60 70 80 90





Exclude Information

Quotes	Excluded
References/Bibliography	Excluded
Sources: Less than 14 Words %	Not Excluded
Excluded Source	**0 %**
Excluded Phrases	Not Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	Yes

A Unique QR Code use to View/Download/Share Pdf File

DrillBit

DrillBit Similarity Report

7	8	A	A-Satisfactory (0-10%) B-Upgrade (11-40%) C-Poor (41-60%) D-Unacceptable (61-100%)
SIMILARITY %	MATCHED SOURCES	GRADE	

LOCATION	MATCHED DOMAIN	%	SOURCE TYPE
1	dochero.tips	1	Internet Data
2	Is the 2D Unlabelled Data Adequate for Facial Expression Recognition by Tao-216	1	Publication
3	Multiclass Multimodal Detection and Tracking in Urban Environments by Spinello-2010	1	Publication
4	infonomics-society.org	1	Publication
5	Multispectral Fusion Approach for Traffic Target Detection in Bad Weather by Han-2020	1	Publication
6	slideshare.net	1	Internet Data
7	ijrte.org	<1	Publication
8	academic.oup.com	<1	Internet Data

Abstract

Human interactions are fundamentally based on emotions, and AI's capacity to perceive and react to emotions creates a plethora of opportunities. Intelligent analysis of emotional states conveyed by video aids in understanding the user's emotions and improves services to increase marketing competitiveness. Emotion recognition has a significant impact on human-computer interaction, educational practices, intelligent vehicles, marketing and mental health. According to recent studies, body language and facial expressions have a big role in determining emotions. However, the contextual information of neighboring frames is the primary focus of these studies, and the spatiotemporal relationships between distant or global frames are rarely explored.

The authors suggest enhancing the efficiency of video emotion recognition by extracting spatiotemporal features through additional temporal encoding. To capture the local Spatiotemporal features of the facial expressions, proposes a super image-based Spatiotemporal convolution model (SISTCM) for the modality of facial expressions. This is achieved by stacking the video frames into two super images along the width and height axes, and then applying 2D convolution. IN addition, a two-stream long short-term memory (LSTM) model is presented to acquire additional global temporal cues by considering the progressive relationship of emotion expressions over time. To obtain the final recognition result, it takes as input local spatiotemporal features and clip-level emotion representations.

They propose a body gesture representation method based on body joint movement, in which body gestures are represented by 25 body joints. Using this representation result, an attention-based channel-wise convolutional model (ACCM) is used for learning joint features and recognizing emotions. Data is an essential component of emotion recognition approaches, and obtaining the data required to train machine learning algorithms is often difficult.

KEY WORDS: Video emotion recognition, Facial expression, Spatiotemporal features, Body joints, Gesture representation

CHAPTER 1

SUMMARY OF THE BASE PAPER

Title: Learning facial expression and body gesture visual information for video emotion recognition

Journal Name: Expert System with Applications

Publisher: MDPI

Year: 2024

Citation Index: SCI

In recent years, study of facial expression and body gesture are two major implications in identifying human emotions. However, existing study primarily concentrates on contextual information within nearby frames and neglects the Spatiotemporal relationship. Certain studies acknowledge the importance of both facial and body posture information, primarily focusing on developing fusion techniques to enhance emotion recognition performance. Nevertheless, several constraints persist that these studies did not thoroughly examine. In response, this paper revisits the study of facial expression and body gesture, proposing an improvement in video emotion recognition by extracting the Spatiotemporal features.

Facial expression sequence data is distinct from static images which includes both spatial and temporal aspects. For analyzing facial expressions, the paper propose SISTCM. This model aggregates video frames into two super images in the axis of width and height-enabled 2D convolution to capture the local Spatiotemporal facial expression features.

The shared convolution kernels across the two super images facilitate collaborative learning of local Spatiotemporal features from different perspectives. Additionally, they introduce a Two-Stream LSTM model to capture global temporal cues, considering the progressive relationship of emotion expressions over time.

The Two-stream LSTM model integrates clip-level emotion representations and local Spatiotemporal features as input to generate the final recognition outcome, and blend these to enhance facial expression recognition performance. The entire facial expression recognition framework is designed as an end-to-end model and optimized through multi-stage supervised learning to enhance recognition performance.

For analyzing body gesture, they introduce body gesture representation method .The method utilizes body join movement, where the representation of body gesture is constructed using data from 25 distinct body joints. Initially this method detects the position of key points and records the changes in their positions over time.

Subsequently, the changes in joint positions are aggregated over time to capture the time-dependent relationships inherent in body gestures. Using this representation, we introduce an attention-based channel-wise convolutional model (ACCM) to learn features from the joints and recognize emotions. The ACCM effectively preserves the unique characteristics of each joint through channel-wise convolutional layers, while capitalizing on key features using an attention mechanism.

To verify the effectiveness of the methods used for facial expression recognition and body posture recognition and the performance, we explore different fusion mechanisms. Utilizing the distinct advantages and complementary aspects of both visual modalities, our approach aims to optimize emotion recognition performance.

2. Method for facial expression:

2.1 video pre-processing:

We have got three dataset for recognizing facial expressions and they are eNTERFACE05, CK+ and Aff-Wild2. In the first stage of preprocessing we first extract all the frames from the video .The next stage is to use the DBFace model to detect and crop the frames that only contain facial parts.further we divide each video into C clips and each clips contain certain number of frames for processing .It enables a more systematic analysis of emotional expressions while expanding the dataset and preventing overfitting to some extent.

2.2 Spatiotemporal features extraction:

To extract Spatiotemporal features we use SISTCM , we sample T frames for each clip to learn about Spatiotemporal relationships of complete frames which reduce the computational complexity .the concept is to conceptualize the video sequence as a stack of frames along different axes capturing spatial and temporal features.

By stacking frames along H and W dimensions, two super image are generated $H\times WT$ and $HT\times W$.This arrangement preserves spatial information from individual frames while also encoding temporal dependencies between consecutive frames.SISTCM uses 3×3 2D convolution to learn Spatiotemporal features.

In SISTCM each clip of data is in the shape of H×W×T that servers as the input. Initially the sistcm convert the input into two super image H× WT and HT×W.then 3×3 2D convolution are applied to these super image, with convolutional kernels ,to extract local Spatiotemporal features efficiently.Atlast the obtained two Spatiotemporal feature maps, denoted as XH and XW are first reverted to their original dimension H×W×T Then, they are combined using a weighted fusion technique to produce the final result.

The two feature maps are connected and then processed through a fully-connected layer followed by a softmax layer to compute weights:

$$\alpha = \mathrm{Softmax}[W\alpha(XH , XW)]$$

We use the ResNet18 and SISTCM model to extract local Spatiotemporal feature and clip-level emotion representation .We use FC for sentiment classification to obtain the clip-level emotion representation.

2.3: Two-stream LSTM model:

They propose a 2-stream LSTM model to learn global temporal cues for facial expression recognition. The approach involves treating the local Spatiotemporal feature sequence as the feature stream and the clip-level emotion representations as the emotion stream. The feature stream is responsible for conducting emotion recognition, resulting in the emotion vector E1 derived from local Spatiotemporal features.

Meanwhile, the emotion stream is trained to derive emotion vector E2 from clip-level emotion representations. Subsequently, the final video emotional state E is attained by fusing both emotion vectors.

2.4 Multi-stage supervision:

To ensure that the recognized emotion at each stage aligns closely with the label throughout the entire recognition process, we introduce a multi-stage supervised learning approach. By doing so, the model receives guidance and feedback at every stage of processing, resulting in comprehensive training and alignment between the predicted emotions and the ground truth labels.

After obtaining the clip-level emotion representations post SISTCM-ResNet18, we compute the L1 Cross-Entropy-Loss between these representations and the corresponding labels. The L1 loss is given by:

$$L1 = -\sum \log(pc),$$

Where pc is the estimated probability for the c-th example.

Similarly, within the two-stream LSTM module, Cross-Entropy-Loss is employed to supervise the emotion vectors of both the feature stream and the emotion stream, denoted as L2 loss and L3 loss, respectively.
Thus, the final loss L is formulated as:

$$L = L1 + \lambda L2 + \mu L3$$

Where λ and μ are equilibrium coefficients, allowing for balanced weighing between the various loss components.

3. Method for body gesture

3.1Body joints marked:

The position data of key joints in each frame of the video is obtained using methods like Open Pose. The position data consist of (x,y) coordinates of each joint. Identify and select key joints relevant for gesture representation. In this paper 25 body joints are selected as key joints.

3.2 Body gesture representation:

Choose a temporal relationship function W(t) to assign weights to the descriptive images base on their timestamps.

Different relationship: $W(t) = (1\ T{-}1)\ (t\hat{}2 - t)$
Linear relationship : $W(t) = (T/\ T{-}1)\ (t - 1)$

For each descriptive image It, corresponding weighted representation Gt = Tt * W(t) ,Sum up the weighted representations to obtain final body gesture representation $G = \sum G$t. The final body gesture representation G is obtained, consisting of 25 channels corresponding to the key joints. By following these steps, a body gesture representation without a timeline is constructed.

3.3 ACCM model:

Attention based convolutional models consist of two branches. The first branch contains two blocks, each composed of a convolutional layer, and a ReLU layer .The second layer includes a channel-wise convolutional layer, attention layer, and the same blocks as the first branch. These branches operate independently and then are aggregated. Input to this ACCM is the body gesture representations obtained from the previous step. The outputs from the two branches are aggregated, possibly by concatenation, to combine the extracted features from both branches. AN adaptive AvgPooling layer is applied to adaptively reduce the spatial dimensions of the features. A softmax layer is employed for classification, producing the final emotion label. This model handles body gesture representations which preserves simplified information as compared to the original input. This model does not require pre-training, making it efficient. No of parameters is also smaller than ResNet18.

3.3.1 Channel-wise Convolutional Layer:

Each channel of the input tensor undergoes independent convolution operations. For each channel, a separate convolution is applied between the channel and its corresponding kernel. Element-wise multiplication occurs between the input region and the kernel, followed by summation to produce a single value in the output feature map. This process repeats for each spatial location dimension as the input.

3.3.2 Attention Layer:

In emotion recognition the importance of each body's joints may vary. To address this attention layer is introduced. The weight and biases of the fully-connected layers are initialized using random sampling from a uniformly distributed range $U[-a, a]$.

$$a = \surd\ 6\ nin+nout$$

nin and $nout$ represent the number of input and output channels, respectively. Once the weights are obtained, they are applied to the original input. Each channel of the original input is multiplied element-wise by its corresponding attention weight to obtain the result.

CHAPTER 2

MERITS AND DEMERITS

Merits:

- Consideration of both spatial and temporal relationship
- Use of two-stream LSTM improves the performance
- This ACCM model handles body gesture representations which preserves simplified information as compared to the original input.
- This ACCM model does not require pre-training, making it efficient.
- Number of parameters is also lesser as compared to ResNet18.
- Fusion of facial expression-based and body gesture-based methods, which effectively improves the accuracy of emotion recognition. This highlights the importance of leveraging multiple modalities for enhancing performance

Demerits:

- Lack of External Validation and limited use of datasets.
- Limited Discussion on Generalizability.

CHAPTER 5

Conclusion:

In this study, we leverage the distinct visual characteristics of facial expressions and body gestures and propose suitable methods for video emotion recognition. For facial expression sequences, we introduced the SISTCM model, which extracts local Spatiotemporal features and learns clip-level emotional states. We then utilize a two-stream LSTM to further capture global temporal cues and refine emotion recognition. By fusing features and emotion pathways, the two-stream LSTM enhances accuracy significantly. For body gesture sequences, we present a body gesture representation method to represent gesture changes using joint information and develop the ACCM model for emotion recognition. This representation simplifies gesture information, reducing training complexity, while ACCM maximizes the advantages of key channel features and preserves their independence. Extensive experiments demonstrate the superiority of our proposed unimodal emotion recognition methods over alternatives. Furthermore, integrating facial expression and body gesture methods effectively enhances emotion recognition accuracy.

FUTURE WORK:

Human interactions rely heavily on emotions, and AI's ability to recognize and respond to emotions opens up numerous possibilities. By intelligently analyzing the emotional cues in videos, it becomes possible to better understand user emotions and enhance services, thereby boosting marketing competitiveness. Here are a few potential directions for the further scope of learning facial expression and body posture.

1. In future, we aim to disentangle identity attributes and prioritize the analysis of general emotional features.

2. Given that certain datasets may exclusively include upper-body data, we also conduct experiments using upper-body joint information. Despite a slight decrease in performance due to the reduced amount of information, the approach remains effective for upper-body analysis. We can add the reduced amount of information and improve our analysis for body posture.

3. We are using more dataset and it take times to run 50 epochs for each model and their corresponding dataset .In future we can improve processing of dataset.