INTRODUCTION

- **Lip reading is a powerful skill with challenges at the word level due to visual ambiguities.**
- **Traditional lip-reading models consist of an encoder and a decoder for transcripts.**
- **Recent breakthroughs inspire innovation, leading to an expanded framework.**
- **Novelties include sentiment analysis, 3D ResNet convolution, and external models.**
- **Sentiment analysis enhances emotion detection for content recommendation.**
- **Face extraction adds context to sentiment analysis, useful for video summarization.**
- **3D ResNet convolution improves spatiotemporal feature extraction for diverse applications.**
- **The inference module merges insights for content summarization and recommendation.**
- **External language models enhance fluency and coherence in language generation.**
- **Our work pioneers advancements in cross-modal language modeling with broad applications in multimedia and emotion-aware systems.**

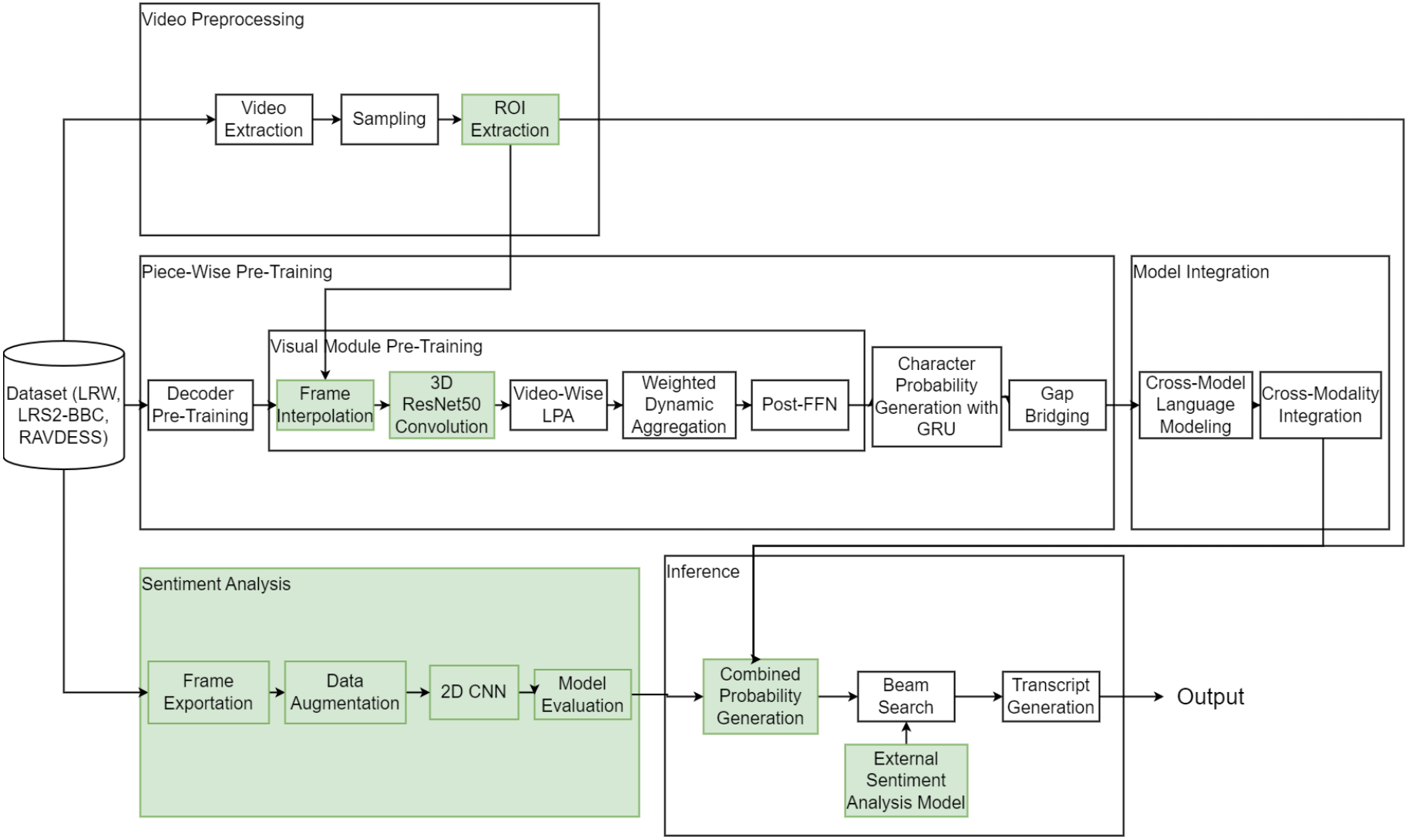
OVERALL OBJECTIVES

- **Enhance lip reading precision by mitigating lip pattern ambiguities.**
- **Achieve deeper emotional and contextual understanding in lip reading.**
- **Enrich lip reading with multimodal data, capturing non-verbal communication.**
- **Advance visual analysis capabilities, recognizing spatiotemporal features.**
- **Achieve holistic content understanding and knowledge extraction.**
- **Elevate language generation quality through external models.**
- **Extend the practical applications of cross-modal language modeling.**

PROPOSED SYSTEM

- **Lip reading, or speech reading, interprets spoken language visually through lip and facial movements.**
- **A system is proposed to advance cross-modal language modeling in lip reading.**
- **The dataset undergoes video processing, extracting speaker's face, and ROI for analysis.**
- **Visual module employs ResNet-50-based 3D convolution and LPA for precise lip reading.**
- **Pre-training prepares decoder and visual module with distinct tasks for lip reading.**
- **"Piece-wise training" is a method used to individually equip the decoder and the visual module with particular pre-training tasks. This readies them to efficiently produce both transcripts and multi-motion-informed contexts for the lip-reading model.**
- **In sentiment analysis, RAVDESS dataset frames are processed, mean faces calculated, and a CNN model implemented using TensorFlow's Keras API, demonstrating effective facial emotion recognition through organized dataset preparation, augmentation, and training/validation accuracy monitoring.**

OVERALL ARCHITECTURE DIAGRAM



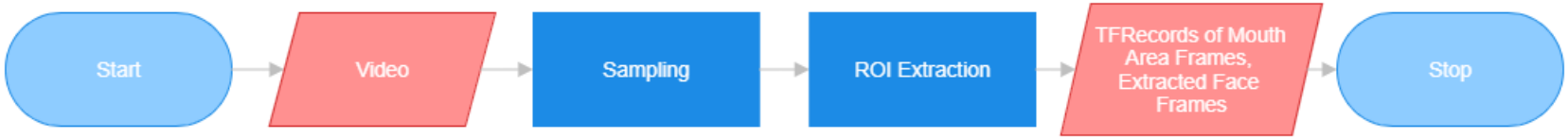
DETAILED MODULE DESIGN

I. Video Preprocessing

- **Input: Video**
- **Output: TFRecords of Mouth Area Frames, Extracted Face Frames**
- **Extracts relevant facial features from videos by identifying and extracting the mouth area, nose, chin, and the speaker's face.**
- **Outputs processed data in TFRecords format for further analysis and model training.**
- **Sets the initial stage for the lip reading process by creating region of interest (ROI) frames and extracting facial data.**
- **Flowchart:**

Pseudocode:

```
1. start
2. video_list = load_videos_from_dataset()
3. frame_width = 224
4. frame_height = 224
5. fps = 25
6. filter_width = 5
7. output_channels = 512
8. video_features = []
9. for video_frames in video_list:
    a. sampled_frames = sample_video_frames(video_frames, fps)
    b. for i in range(0, len(sampled_frames), filter_width):
        i. frame_sequence = sampled_frames[i:i + filter_width]
        ii. landmarks=extract_landmarks(frame_sequence)
        iii. final_sequence=interpolate_landmarks(landmarks)
        iv. transformed=transform(final_sequence)
        v. video_roi=extract(transformed)
```



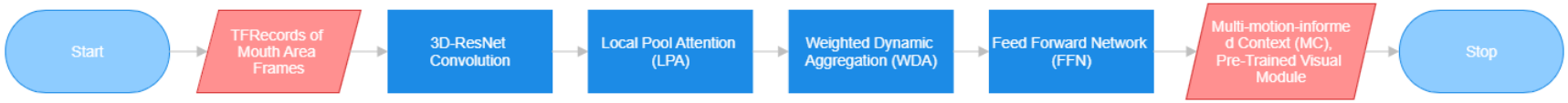
DETAILED MODULE DESIGN

II. Visual Module Pre-Training

- **Input: TFRecords of Mouth Area Frames**
- **Output: Multi-motion-informed Context (MC), Pre-Trained Visual Module**
- **Utilizes 3D convolutional blocks based on ResNet-50 to process video frames frame by frame.**
- **Enhances lip-motion representations using Local-pool Attention (LPA) blocks, which capture fine-grained spatial and temporal details.**
- **Implements a Weighted Dynamic Aggregation (WDA) mechanism to combine multiple lip-motion representations, creating a comprehensive view of visual data.**
- **Flowchart:**

Pseudocode:

1. Start
2. $V=[t,n,m,c]$
3. $scales=\{scale(1),scale(2),scale(3),scale(4),scale(5)\}$ #where $scale_i=[t,n_i,m_i,c_i]$
   a. Start (LPA)
   b. **Input:** Res_i's output $scale^t_i=[t,n_i,m_i,c_i]$
   c. $Res_i=[t,n'_i,m'_i,C_i]$ # 3D-Max-Pooling(Res_i)
   d. $Res_i=[t,n'_i\times m'_i\times C_i]$ # Flatten frame
   e. $Res_i=Res_iW^{input}$ # Input projection
   f. $Res_i=(Res_i+PE)$ # Inject position encoding
   g. $res=LN(Res_i)$ # Layer normalization
   h. $Q=res\times W^Q_i, K=res\times W^K_i, V=res\times W^V_i$
   i. $x=SDA(Q,K,V)$ # Scaled Dot-Product Attention
   j. $LPA_i=Res_i+x$ # Residual connection
   k. **Out:** $LPA_i=[t,n'_i\times m'_i\times c_i]$
   l. **Stop**
4. $WDA=(\sum\lambda_iLPA_i(Res_i))W^{output}$
5. $MC=ReLu(WDA\times W^1+b^1)\times W^2+b^2$
6. Stop



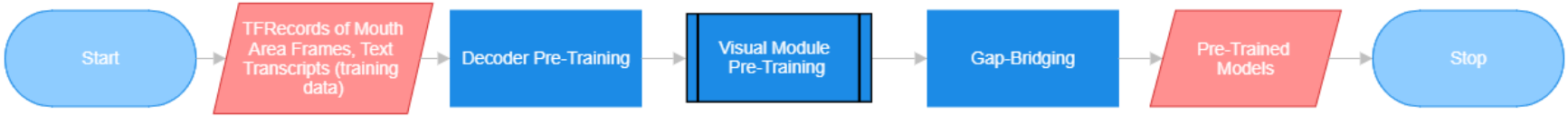
DETAILED MODULE DESIGN

III. Piece-Wise Pre-Training

- **Input: TFRecords of Mouth Area Frames, Text Transcripts**
- **Output: Pre-Trained Models**
- **Pre-trains the decoder as a language model to predict characters one at a time from training transcripts.**
- **Pre-trains the visual module as a context generator for generating multi-motion-informed contexts from video clips.**
- **Connects the two pre-trained components using a shared softmax layer for cross-modality information exchange.**
- **Flowchart:**

Pseudocode:

1. Start
2. $S=\{s_1,\ldots,s_{|S|}\}$ #unsupervised corpus
3. $L_{Dec}=\sum \log P(s|s_{i-k},\ldots,s_{i-1};\theta_{Dec\setminus FC},\theta_{embs})$
4. $L_{Vis}=1/|C|\sum C\in C\log\prod_{i=1}^{n} P(w_i|w_{<i},D^3,C;\theta_{Vis},\theta_{GRU},\theta_{embs})$
5. Stop



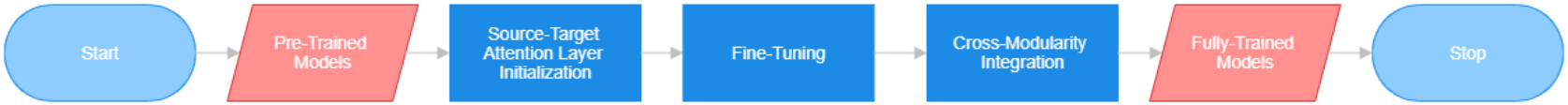
DETAILED MODULE DESIGN

IV. Model Integration

- **Input: Pre-Trained Models**
- **Output: Fully Trained Models**
- **Fine-tunes the pre-trained decoder and visual module for lip-reading tasks.**
- **Employs source-target attention layers for cross-modal language modeling, aiming to maximize transcript likelihood.**
- **Trains the model to align visual information with transcript characters for effective lip reading.**
- **Flowchart:**

Pseudocode:

1. Start
2. $s^0 = \{\}$ # feature vector
3. for l in decoder_layers:
   a. $s^l = F^l_s(s^{l-1}, s^{l-1}, s^{l-1})$
   b. $s^l = F^l_C(s^l, MC, MC)$
   c. $s^l = FFN^l(s^l)$
4. $S = \{s_1, s_2, \ldots, s_n\}$
5. $V = \{v_1, v_2, \ldots, v_m\}$
6. $L_{lip} = 1/|V| \sum_{v \in V} \log \prod_{i=1}^{n} P(s_i | s_{<i}, V; \theta_{Vis}, \theta_{GRU}, \theta_{smax})$
6. Stop



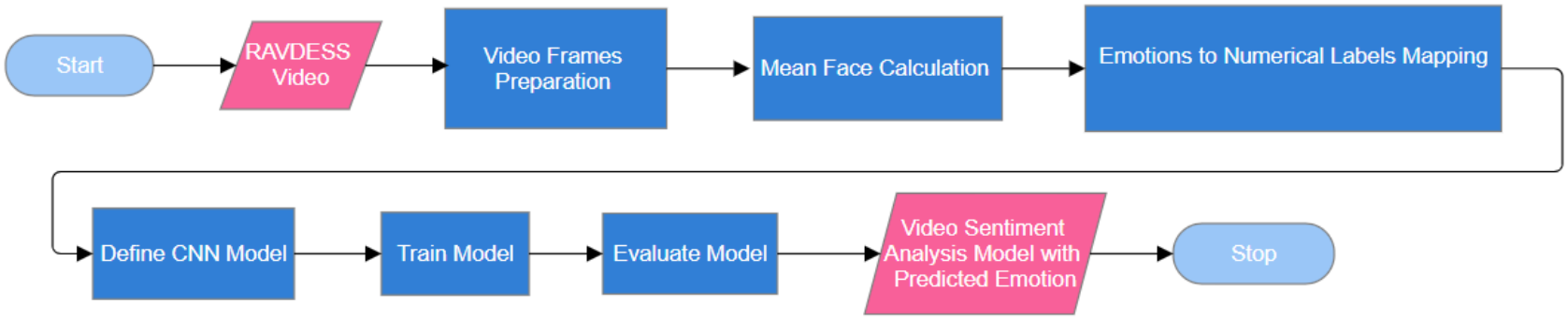
DETAILED MODULE DESIGN

V. Sentiment Analysis

- **Input: RAVDESS dataset**
- **Output: Video Sentiment Analysis Model, Predicted Emotion**
- **Frames are extracted, resized and transformed into images, with mean faces computed for emotion mapping (1-8)**
- **CNN model is defined with convolutional layers, batch normalization, activation, pooling, and dropout regularization.**
- **Trained CNN model achieves robust facial emotion recognition, utilizing data augmentation, mean face subtraction, and monitored training/validation accuracy.**
- **Flowchart:**

Pseudocode/Algorithms:

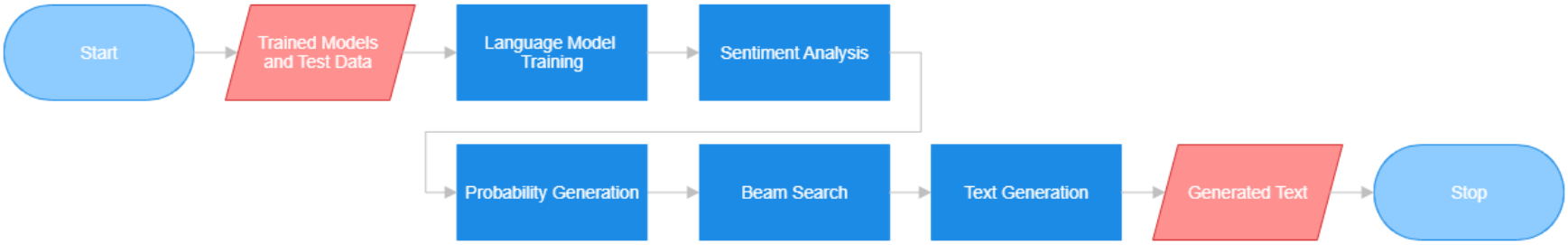
1. Start
2. x=Conv2D(filters=32,kernel_size=(5,5)
3. kernel_initializer='he_normal',name='conv2d_0')(x)
4. x=BatchNormalization(name='batchnorm_0')(x)
5. x=Activation('ReLu')(x)
6. x=MaxPooling2D(poolside=(2,2),name='maxpool2d_0')(x)
7. Stop



DETAILED MODULE DESIGN

- **VI. Inference**

- **Input: Trained models and testing data**
- **Output: Generated Text**
- **Utilizes a character-level language model during the inference phase.**
- **Performs left-to-right beam search for text generation, incorporating language model probabilities.**
- **Integrates sentiment analysis to ensure that generated text aligns with both language model probabilities and the sentiment conveyed by the visual content.**
- **Flowchart:**



IMPLEMENTATION DETAILS – VIDEO PREPROCESSING

Extraction of Datasets:

```
%%capture
!tar -xvf lrw-v1.tar --skip-old-files --remove-files

!unzip -n lrs3_test_v0.4.zip -d /content/drive/MyDrive/LRS/lrs3_v0.4

!unzip -n lrs3_v0.4_txt.zip -d /content/drive/MyDrive/LRS
```

IMPLEMENTATION DETAILS

Landmark Detection for ROI extraction

```python
class LandmarksDetector:
    def __init__(self, device="cuda:0", model_name="resnet50"):
        self.face_detector = RetinaFacePredictor(
            device=device,
            threshold=0.8,
            model=RetinaFacePredictor.get_model(model_name),
        )
        self.landmark_detector = FANPredictor(device=device, model=None)

    def __call__(self, video_frames):
        landmarks = []
        for frame in video_frames:
            detected_faces = self.face_detector(frame, rgb=False)
            face_points, _ = self.landmark_detector(frame, detected_faces, rgb=True)
            if len(detected_faces) == 0:
                landmarks.append(None)
            else:
                max_id, max_size = 0, 0
                for idx, bbox in enumerate(detected_faces):
                    bbox_size = (bbox[2] - bbox[0]) + (bbox[3] - bbox[1])
                    if bbox_size > max_size:
                        max_id, max_size = idx, bbox_size
                landmarks.append(face_points[max_id])
        return landmarks
```

IMPLEMENTATION DETAILS

This class introduces random temporal masks to the input video/tensor to allow the network to focus on different time segments of the input sequence during training. This allows the model to be trained on different temporal sequences of video clips, allowing for better handling of data during testing and real-time translation.

```python
class AdaptiveTimeMask(torch.nn.Module):
    def __init__(self, window, stride):
        super().__init__()
        self.window = window
        self.stride = stride

    def forward(self, x):
        # x: [T, ...]
        cloned = x.clone()
        length = cloned.size(0)
        n_mask = int((length + self.stride - 0.1) // self.stride)
        ts = torch.randint(0, self.window, size=(n_mask, 2))
        for t, t_end in ts:
            if length - t <= 0:
                continue
            t_start = random.randrange(0, length - t)
            if t_start == t_start + t:
                continue
            t_end += t_start
            cloned[t_start:t_end] = 0
        return cloned
```

IMPLEMENTATION DETAILS

- linearly interpolate the landmarks detected between two given indexes to fill in the missing landmarks between frames
- transforms the image based on the landmarks and the output size
- crops a rectangular patch around the mouth to extract mouth ROI (based on the landmarks)

```python
def linear_interpolate(landmarks, start_idx, stop_idx):
    start_landmarks = landmarks[start_idx]
    stop_landmarks = landmarks[stop_idx]
    delta = stop_landmarks - start_landmarks
    for idx in range(1, stop_idx - start_idx):
        landmarks[start_idx + idx] = (
            start_landmarks + idx / float(stop_idx - start_idx) * delta
        )
    return landmarks


def warp_img(src, dst, img, std_size):
    tform = tf.estimate_transform("similarity", src, dst)
    warped = tf.warp(img, inverse_map=tform.inverse, output_shape=std_size)
    warped = (warped * 255).astype("uint8")
    return warped, tform


def apply_transform(transform, img, std_size):
    warped = tf.warp(img, inverse_map=transform.inverse, output_shape=std_size)
    warped = (warped * 255).astype("uint8")
    return warped


def cut_patch(img, landmarks, height, width, threshold=5):
    center_x, center_y = np.mean(landmarks, axis=0)
    if abs(center_y - img.shape[0] / 2) > height + threshold:
        raise OverflowError("too much bias in height")
    if abs(center_x - img.shape[1] / 2) > width + threshold:
        raise OverflowError("too much bias in width")
    y_min = int(round(np.clip(center_y - height, 0, img.shape[0])))
    y_max = int(round(np.clip(center_y + height, 0, img.shape[0])))
    x_min = int(round(np.clip(center_x - width, 0, img.shape[1])))
    x_max = int(round(np.clip(center_x + width, 0, img.shape[1])))
    cutted_img = np.copy(img[y_min:y_max, x_min:x_max])
    return cutted_img
```

IMPLEMENTATION DETAILS

The class VideoProcess utilizes the functions described to preprocess the landmarks by interpolating missing frames and crop patches from the video frames based on the landmarks.

This class crops patches from video frames based on interpolated landmarks, iterates through the video frames, smoothes the landmarks, and applies affine transformations.

```
class VideoProcess:
    def __init__(
        self,
        mean_face_path="20words_mean_face.npy",
        crop_width=96,
        crop_height=96,
        start_idx=48,
        stop_idx=68,
        window_margin=12,
        convert_gray=True,
    ):
        self.reference = np.load(
            os.path.join(os.path.dirname(__file__), mean_face_path)
        )
        self.crop_width = crop_width
        self.crop_height = crop_height
        self.start_idx = start_idx
        self.stop_idx = stop_idx
        self.window_margin = window_margin
        self.convert_gray = convert_gray

    def __call__(self, video, landmarks):
        # Pre-process landmarks: interpolate frames that are not detected
        preprocessed_landmarks = self.interpolate_landmarks(landmarks)
        # Exclude corner cases: no landmark in all frames or number of fram
        if (
            not preprocessed_landmarks
            or len(preprocessed_landmarks) < self.window_margin
        ):
            return
        # Affine transformation and crop patch
        sequence = self.crop_patch(video, preprocessed_landmarks)
        assert sequence is not None, "crop an empty patch."
        return sequence
```

IMPLEMENTATION DETAILS

All these functions described above are compiled into a python file and run to process the videos and store them for further modules.

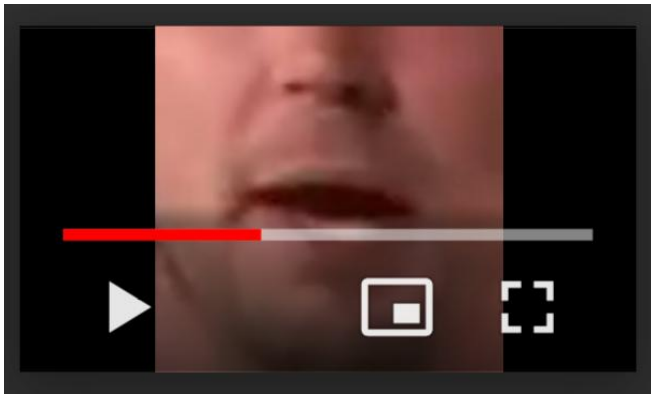
```
save_vid_aud_txt(
    dst_vid_filename,
    dst_aud_filename,
    dst_txt_filename,
    trim_vid_data,
    trim_aud_data,
    content,
    video_fps=25,
    audio_sample_rate=16000,
)
```

```
!python preprocess_lrs2lrs3.py \
    --data-dir /content/drive/MyDrive/LRS/mvlrs_v1 \
    --landmarks-dir /content/drive/MyDrive/LRS/LRS2_landmarks \
    --root-dir /content/drive/MyDrive/LRS/preprocessed \
    --dataset lrs2 \
    --subset train \
    --groups 40 \
    --job-index 3
```

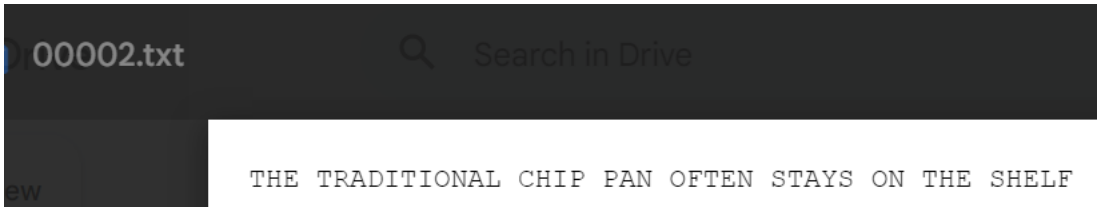
```
Directory /content/drive/MyDrive/LRS/preprocessed/labels created
100% 142160/142160 [1:15:54<00:00,  1.28s/it]
```

IMPLEMENTATION DETAILS

Output:



(cropped video clip)



(transcript of the clip for training)

IMPLEMENTATION DETAILS – PIECEWISE PRE-TRAINING

Pre-training of decoder (language model)

First, all the transcripts are loaded, then appended to create an unsupervised text corpus.

```
import os
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
from torch.optim import Adam

def load_transcripts(directory):
    transcripts = []
    for root, _, files in os.walk(directory):
        for file in files:
            if file.endswith(".txt"):
                with open(os.path.join(root, file), "r") as f:
                    transcript = f.read()
                    transcripts.append(transcript)
    return "\n".join(transcripts)

transcripts_directory = "/content/drive/MyDrive/LRS/preprocessed/lrs2/lrs2_text_seg24s/main"
text_corpus = load_transcripts(transcripts_directory)
```

IMPLEMENTATION DETAILS

Using the unsupervised corpus, a set of all the characters that can be generated by the decoder is instantiated. It reads text data from the saved corpus, tokenizes characters, and creates input-output pairs for training.

```python
class CharDataset(Dataset):
    def __init__(self, filepath, block_size):
        with open(filepath, 'r', encoding='utf-8') as f:
            text = f.read()
        self.vocab = sorted(set(text))
        self.stoi = {ch: i for i, ch in enumerate(self.vocab)}
        self.itos = {i: ch for i, ch in enumerate(self.vocab)}
        self.block_size = block_size
        self.data = [self.stoi[ch] for ch in text]

    def __len__(self):
        return len(self.data) - self.block_size

    def __getitem__(self, idx):
        chunk = self.data[idx:idx+self.block_size+1]
        x = torch.tensor(chunk[:-1], dtype=torch.long)
        y = torch.tensor(chunk[1:], dtype=torch.long)
        return x, y
```

```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
filepath = saved_corpus_path = "/content/drive/MyDrive/LRS/preprocessed/lrs2/lrs2_text_seg24s/saved_corpus.txt"

dataset = CharDataset(filepath, block_size=128)
vocab_size = len(dataset.vocab)
data_loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

IMPLEMENTATION DETAILS

The necessary classes to implement and pretrain the decoder are defined:

```
# Positional Encoding
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + self.pe[:x.size(0), :]
        return self.dropout(x)
```

It provides positional information to the input embeddings. It generates sinusoidal positional embeddings based on the position of tokens in the sequence. Hence, the model can tell where each word is located in a sentence.

IMPLEMENTATION DETAILS

```python
# Decoder Feed-Forward Network
class DecoderFFN(nn.Module):
    def __init__(self, d_model, d_ff, dropout):
        super(DecoderFFN, self).__init__()
        self.linear1 = nn.Linear(d_model, d_ff)
        self.dropout = nn.Dropout(dropout)
        self.linear2 = nn.Linear(d_ff, d_model)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.dropout(self.relu(self.linear1(x)))
        x = self.linear2(x)
        return x
```

This allows the model to capture non-linear patterns in data, as well as prevent overfitting through dropout.

IMPLEMENTATION DETAILS

```python
class TransformerDecoderLayer(nn.Module):
    def __init__(self, d_model, heads, d_ff, dropout):
        super(TransformerDecoderLayer, self).__init__()
        self.self_attn = nn.MultiheadAttention(d_model, heads, dropout=dropout)
        self.norm1 = nn.LayerNorm(d_model)
        self.dropout1 = nn.Dropout(dropout)

        self.cross_attn = nn.MultiheadAttention(d_model, heads, dropout=dropout)
        self.norm2 = nn.LayerNorm(d_model)
        self.dropout2 = nn.Dropout(dropout)

        self.ffn1 = DecoderFFN(d_model, d_ff, dropout)
        self.ffn2 = DecoderFFN(d_model, d_ff, dropout)  # Second FFN as specified
        self.norm3 = nn.LayerNorm(d_model)
        self.dropout3 = nn.Dropout(dropout)

    def forward(self, tgt, memory, tgt_mask=None, memory_mask=None,
                tgt_key_padding_mask=None, memory_key_padding_mask=None):
        tgt2 = self.self_attn(tgt, tgt, tgt, attn_mask=tgt_mask,
                              key_padding_mask=tgt_key_padding_mask)[0]
        tgt = tgt + self.dropout1(tgt2)
        tgt = self.norm1(tgt)

        tgt2 = self.cross_attn(tgt, memory, memory, attn_mask=memory_mask,
                               key_padding_mask=memory_key_padding_mask)[0]
        tgt = tgt + self.dropout2(tgt2)
        tgt = self.norm2(tgt)

        tgt2 = self.ffn1(tgt)
        tgt = tgt + self.dropout3(tgt2)
        tgt = self.norm3(tgt)

        tgt2 = self.ffn2(tgt)  # Second FFN application
        tgt = tgt + self.dropout3(tgt2)
        tgt = self.norm3(tgt)

        return tgt
```

This uses self-attention to understand how words in the target sentence relate to each other, cross-attention to look at words in the source sentence, and feed-forward networks to process the information in a way that understands complex patterns.

Layer normalization and dropout are used to keep things stable and prevent overfitting.

It enables the decoder to generate accurate predictions by effectively leveraging both the target and source sequences.

IMPLEMENTATION DETAILS

```
# Transformer Decoder
class TransformerDecoder(nn.Module):
    def __init__(self, vocab_size, d_model, N, heads, d_ff, dropout):
        super(TransformerDecoder, self).__init__()
        self.embed = nn.Embedding(vocab_size, d_model)
        self.pos_encoder = PositionalEncoding(d_model, dropout)
        self.layers = nn.ModuleList([TransformerDecoderLayer(d_model, heads, d_ff, dropout) for _ in range(N)])
        self.norm = nn.LayerNorm(d_model)
        self.out = nn.Linear(d_model, vocab_size)
        self.d_model = d_model

    def forward(self, src, src_mask=None, src_key_padding_mask=None):
        src = self.embed(src) * math.sqrt(self.d_model)
        src = self.pos_encoder(src)
        for layer in self.layers:
            src = layer(src, src_mask, src_key_padding_mask)
        src = self.norm(src)
        output = self.out(src)
        return F.log_softmax(output, dim=-1)
```

This class implements the classes from before to construct the actual transformer decoder.

IMPLEMENTATION DETAILS

Using classes defined before, as well as the unsupervised corpus generated, we pretrain the decoder for future character probability generation.

```
def train_epoch(model, data_loader, optimizer, criterion, device):
    model.train()
    total_loss = 0
    progress_bar = tqdm(data_loader, desc='Training')
    for x, y in progress_bar:
        x, y = x.to(device), y.to(device)
        optimizer.zero_grad()
        logits = model(x)
        loss = criterion(logits.view(-1, logits.size(-1)), y.view(-1))
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        progress_bar.set_postfix({'loss': loss.item()})
    return total_loss / len(data_loader)
```

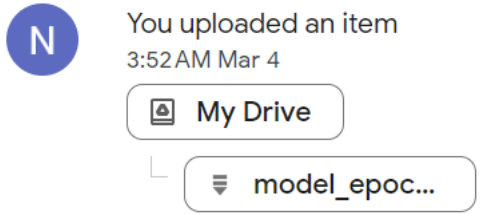
```
model = TransformerDecoder(vocab_size=vocab_size, d_model=1024, N=6, heads=16, d_ff=4096, dropout=0.1).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
```

IMPLEMENTATION DETAILS

```
epochs = 10
for epoch in range(start_epoch, epochs + 1):
    avg_loss = train_epoch(model, data_loader, optimizer, criterion, device)
    print(f'Epoch {epoch}, Loss: {avg_loss}')
    torch.save({
        'epoch': epoch,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'loss': avg_loss,
    }, f'model_epoch_{epoch}.pth')
```

```
Training: 100%|██████████| 63873/63873 [3:04:06<00:00,  5.78it/s, loss=2.95]
Epoch 2, Loss: 2.9174285430177194
Training: 100%|██████████| 63873/63873 [3:03:46<00:00,  5.79it/s, loss=2.91]
Epoch 3, Loss: 2.9173853559181815
```

Final model is saved for for further use:



IMPLEMENTATION DETAILS – VISUAL MODULE PRETRAINING

First, we load and transform the preprocessed videos into tensor format for passing into the model:

```
class VideoDataset(Dataset):
    def __init__(self, root_dir, char_dataset, transform=None, cache_dir="/tmp/processed_videos", max_video_length=30):
        super(VideoDataset, self).__init__()
        self.cache_dir = cache_dir
        os.makedirs(self.cache_dir, exist_ok=True)
        self.max_video_length = max_video_length

        # Use the character-to-integer mapping from the CharDataset
        self.char_dataset=char_dataset
        #self.char_to_int = {char: i+2 for i, char in enumerate(sorted(set("ABCDEFGHIJKLMNOPQRSTUVWXYZ")))}  # Start indices from 2
        #self.char_to_int["UNK"] = 1  # Unknown characters
        #self.char_to_int["<PAD>"] = 0  # Padding token

        self.video_files = []
        self.labels = []

        for dirname, _, filenames in os.walk(root_dir):
            for filename in filenames:

                if filename.endswith('.mp4'):

                    video_file = os.path.join(dirname, filename)
                    frames, _, _ = read_video(video_file, pts_unit='sec')
                    if frames.nelement() == 0 or frames.shape[0] == 0 or frames.shape == torch.Size([0, 1, 1, 3]):
                        continue
                    if frames.nelement() == 1 or frames.shape[0] == 1 or frames.shape == torch.Size([1, 160, 160, 3]):
                        continue
                    label_file = video_file.replace('.mp4', '.txt')

                    if os.path.exists(label_file):
                        with open(label_file, 'r') as f:
                            label = f.read().strip()
                        self.video_files.append(video_file)
                        self.labels.append(label)
                        #print(label)
```

IMPLEMENTATION DETAILS

```
def __len__(self):
    return len(self.video_files)

def __getitem__(self, idx):
    video_file = self.video_files[idx]
    label = self.labels[idx]

    #print(label_tensor)

    frames, _, _ = read_video(video_file, pts_unit='sec')
    label_indices = [self.char_dataset.stoi.get(char) for char in label]
    print("beforeproc:" ,frames.shape)
    print("is this working: ",len(label_indices))
    # Interpolate frames if the number of frames is less than the length of the transcript
    if frames.shape[0] < len(label_indices):
        frames = interpolate_frames(frames, len(label_indices))
        print('after inpll: ',frames.size())

    # Ensure label tensor is padded or trimmed to max_word_length
    label_indices += [0] * (frames.shape[0] - len(label_indices))
    label_tensor = torch.tensor(label_indices, dtype=torch.long)
    print('debugging:',frames.shape[0],"??",len(label_indices))
    if frames.nelement() == 0 or frames.shape[0] == 0:
        frames= torch.zeros(1,self.max_video_length, 3, 224, 224)
        return torch.zeros(self.max_video_length, 3, 224, 224),label_tensor

    if self.transform:
        # Apply transformations
        frames = frames.permute(0, 3, 1, 2)  # Convert frames from T H W C to T C H W format
        frames = self.transform(frames.float())

    print(frames.shape)
    return frames, label_tensor
```

IMPLEMENTATION DETAILS

For videos with less frames than characters, we interpolate the frames instead of utilizing padding.

```python
import cv2
import torch
import os

def interpolate_frames(frames, target_frame_count):
    # Open the video file
    fr=frames.shape[0]
    interpolation_factor = target_frame_count / frames.shape[0]
    interpolation_factor+=1
    interpolator = Interpolator()

    # Initialize list to store interpolated frames
    interpolated_frames = []


    frames=list(frames)
    prev_frame=frames[0]
    # Read and interpolate subsequent frames
    for i in range(1,fr):
        next_frame=frames[i]
        print('test')
        print(frames[i].shape)
        print(prev_frame.unsqueeze(0).numpy().astype(np.float32).shape)
        f=list(interpolate_recursively([prev_frame.numpy().astype(np.float32),next_frame.numpy().astype(np.float32)], interpolation_factor-1,interpolator))
            #interpolated_frame = cv2.addWeighted(prev_frame, (interpolation_factor - i) / interpolation_factor, next_frame, i / interpolation_factor, 0)
            #interpolated_frames.append(interpolated_frame)
        print(':)')

        # Save the original frame
        interpolated_frames+=f
        interpolated_frames=interpolated_frames[:-1]
        # Update previous frame
        prev_frame = next_frame
    interpolated_frames.append(frames[-1])

    # Convert list of frames to tensor
    interpolated_frames = torch.tensor(interpolated_frames)
```

IMPLEMENTATION DETAILS

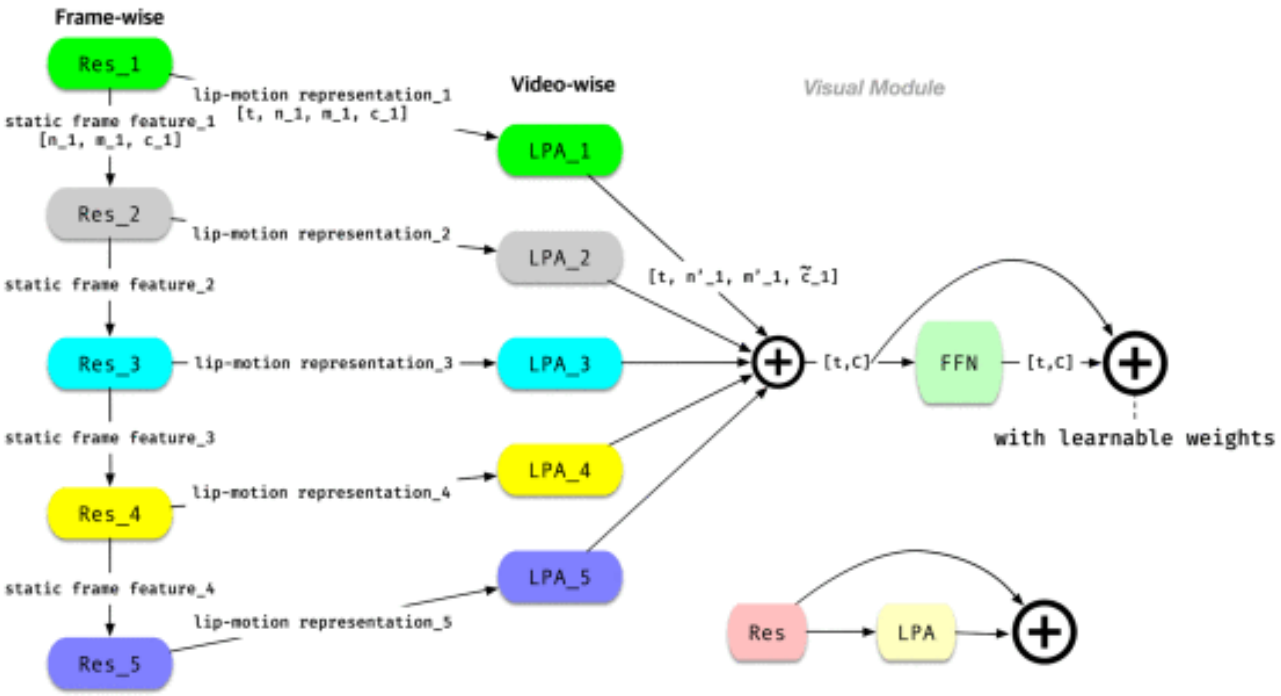
```
# Transform for preprocessing the video frames
transform = Compose([
    Resize((224, 224)),
    #ToTensor(),
    VideoNormalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Instantiate Dataset
root_dir = '/content/drive/MyDrive/LRS/word_clips/pretrain'  # Update this path
video_dataset = VideoDataset(root_dir, char_dataset=d, transform=transform)
```

```
import torch
from torch.nn.utils.rnn import pad_sequence

def custom_collate_fn(batch):
    videos, labels_list = zip(*batch)  # Transpose the batch (list of tuples to tuple of lists)

    # Stack videos if they are already padded to the same shape
    videos = torch.stack(videos, dim=0)

    # Convert labels into tensors and pad them
    labels = torch.stack([torch.tensor(label, dtype=torch.long) for label in labels_list], dim=0)

    return videos, labels
```

```
def custom_collates_fn(batch):
    videos, labels_list = zip(*batch)  # Transpose the batch (list of tuples to tuple of lists)

    # Stack videos if they are already padded to the same shape
    videos = torch.stack(videos, dim=0)

    # Pad labels to match the length of videos
    max_video_length = videos.size(1)  # Assuming videos have the same length
    padded_labels = pad_sequence([torch.tensor(label, dtype=torch.long) for label in labels_list], batch_first=True, padding_value=0)

    # Trim or pad videos to ensure they have the same length
    #if padded_labels.size(1) > max_video_length:
    #    padded_labels = padded_labels[:, :max_video_length]
    if padded_labels.size(1) < max_video_length:
        padding = torch.full((padded_labels.size(0), max_video_length - padded_labels.size(1)), 0, dtype=torch.long)  # Using 0 as the
        padded_labels = torch.cat([padded_labels, padding], dim=1)

# When creating your DataLoader, specify the custom collate function
data_loader = DataLoader(video_dataset, batch_size=1, shuffle=True, collate_fn=custom_collate_fn)
```

IMPLEMENTATION DETAILS

The four parts needed for implementation of visual module include ResNet, Local Pooling Attention (LPA), Weighted Dynamic Aggregation (WDA) and Post Feed Forward Network (Post-FFN).



IMPLEMENTATION DETAILS

This is the ResNet Block which will be used for the final 3D-ResNet50 Architecture in the Visual Module.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class block(nn.Module):
    def __init__(self, in_channels, intermediate_channels, identity_downsample=None, stride=1):
        super(block, self).__init__()
        self.expansion = 4
        self.conv1 = nn.Conv2d(in_channels, intermediate_channels, kernel_size=1, stride=1, padding=0, bias=False)
        self.bn1 = nn.BatchNorm2d(intermediate_channels)
        self.conv2 = nn.Conv2d(intermediate_channels, intermediate_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(intermediate_channels)
        self.conv3 = nn.Conv2d(intermediate_channels, intermediate_channels * self.expansion, kernel_size=1, stride=1, padding=0, bias=False)
        self.bn3 = nn.BatchNorm2d(intermediate_channels * self.expansion)
        self.relu = nn.ReLU(inplace=True)
        self.identity_downsample = identity_downsample
        self.stride = stride

    def forward(self, x):
        identity = x

        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.relu(x)
        x = self.conv3(x)
        x = self.bn3(x)

        if self.identity_downsample is not None:
            identity = self.identity_downsample(identity)

        x += identity
        x = self.relu(x)
        return x
```

IMPLEMENTATION DETAILS

```python
class ResNet(nn.Module):
    def __init__(self, block, layers, image_channels, num_classes):
        super(ResNet, self).__init__()
        self.in_channels = 64
        self.conv1 = nn.Conv2d(image_channels, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        # ResNet layers
        self.layer1 = self._make_layer(block, layers[0], intermediate_channels=64, stride=1)
        self.layer2 = self._make_layer(block, layers[1], intermediate_channels=128, stride=2)
        self.layer3 = self._make_layer(block, layers[2], intermediate_channels=256, stride=2)
        self.layer4 = self._make_layer(block, layers[3], intermediate_channels=512, stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * 4, num_classes)

    def _make_layer(self, block, num_residual_blocks, intermediate_channels, stride):
        identity_downsample = None
        layers = []

        if stride != 1 or self.in_channels != intermediate_channels * 4:
            identity_downsample = nn.Sequential(
                nn.Conv2d(self.in_channels, intermediate_channels * 4, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(intermediate_channels * 4),
            )

        layers.append(block(self.in_channels, intermediate_channels, identity_downsample, stride))
        self.in_channels = intermediate_channels * 4

        for i in range(1, num_residual_blocks):
            layers.append(block(self.in_channels, intermediate_channels))  # Identity downsample is only used in t

        return nn.Sequential(*layers)
```

IMPLEMENTATION DETAILS

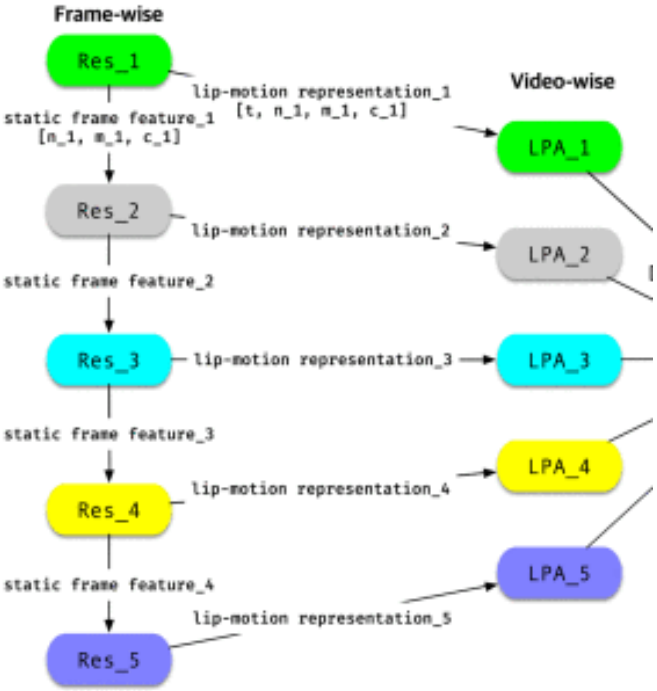
```python
def _forward_impl(self, x):
    # Helper method to process a single frame through all layers and collect outputs
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    outputs = [x]
    x = self.layer1(x)
    outputs.append(x)
    x = self.layer2(x)
    outputs.append(x)
    x = self.layer3(x)
    outputs.append(x)
    x = self.layer4(x)
    outputs.append(x)

    return outputs
```

```python
def forward(self, x):
    batch_size, time_steps, C, H, W = x.size()
    all_outputs = []

    for t in range(time_steps):
        frame = x[:, t, :, :, :]
        frame_outputs = self._forward_impl(frame)
        if not all_outputs:
            all_outputs = [[o.unsqueeze(1)] for o in frame_outputs]
        else:
            for i, o in enumerate(frame_outputs):
                all_outputs[i].append(o.unsqueeze(1))  # Append alo

    # Now, stack along the time dimension for each block's output
    for i in range(len(all_outputs)):
        all_outputs[i] = torch.cat(all_outputs[i], dim=1)  # Concat
        # Permute the tensor to have dimensions [batch, time_steps,
        #all_outputs[i] = all_outputs[i].permute(0, 1, 4, 2, 3)
        print(all_outputs[i].shape)

    return all_outputs  # List of tensors with shapes [batch, time_
```

IMPLEMENTATION DETAILS

As shown from the screenshots, our architecture utilizes five ResNet blocks, the first four extracting information from the input at different scales, the final block pooling all previous output together.

Each ResNet Block has a corresponding LPA block to integrate the spatial information with the temporal information and improve video representation.



IMPLEMENTATION DETAILS

```python
class LPABlock(nn.Module):
    def __init__(self, num_channels, input_height, input_width, max_pool_kernel, d_k=16, d_v=1024, d_model=1024):
        super(LPABlock, self).__init__()
        self.max_pool = nn.MaxPool2d(kernel_size=max_pool_kernel, stride=max_pool_kernel)
        self.flatten = nn.Flatten(start_dim=1)
        # Adjust the in_features based on the expected flattened shape after pooling
        pool_out_height = input_height // max_pool_kernel[0]
        pool_out_width = input_width // max_pool_kernel[1]
        self.input_proj = nn.Linear(num_channels * pool_out_height * pool_out_width, d_model)
        self.pos_enc = PositionalEncoding(d_model=d_model)
        self.layer_norm = nn.LayerNorm(d_model)
        self.attention = ScaledDotProductAttention(d_model, d_k, d_v)

    def forward(self, x):
        batch_size, time_steps, C, H, W = x.size()
        print("1: Input received")
        print(f"2: Input shape {x.size()}")
        x = x.view(-1, x.size(2), x.size(3), x.size(4))
        print(f"3: Reshaped for pooling {x.size()}")
        x = self.max_pool(x)
        print(f"4: After pooling {x.size()}")
        x = self.flatten(x)
        print(f"5: After flattening {x.size()}")

        # Ensure the input projection layer matches the flattened tensor's dimensions
        x = self.input_proj(x)
        print(f'6: After input projection {x.shape}')

        # Split batch and time dimensions again and proceed
        x = x.view(batch_size, time_steps, -1)
        print(f'7: After reshaping back {x.shape}')
        x = self.pos_enc(x)

        x = self.layer_norm(x)
        q, k, v = x, x, x  # Queries, Keys, and Values for self-attention
        x = self.attention(q, k, v)
        return x
```

The LPABlock processes input video data by downsampling spatial dimensions, projecting to a higher-dimensional space, incorporating positional information, and capturing temporal dependencies using attention mechanisms.

As we will be integrating with decoder later, decoder dimensions are utilized for instantiation.

IMPLEMENTATION DETAILS

```python
class CombinedModelWithWDA(nn.Module):
    def __init__(self, resnet, decoder_dim):
        super(CombinedModelWithWDA, self).__init__()
        self.resnet = resnet  # Use the passed-in ResNet instance

        # Assuming the output channels of the ResNet blocks are 256, 512, 1024, and 2048 respecti
        self.lpa_blocks = nn.ModuleList([
            LPABlock(num_channels=64, input_height=56, input_width=56, max_pool_kernel=(1, 1)),
            LPABlock(num_channels=256, input_height=56, input_width=56, max_pool_kernel=(4, 4)),
            LPABlock(num_channels=512, input_height=28, input_width=28, max_pool_kernel=(2, 4)),
            LPABlock(num_channels=1024, input_height=14, input_width=14, max_pool_kernel=(2, 2)),
            LPABlock(num_channels=2048, input_height=7, input_width=7, max_pool_kernel=(1, 1))
        ])

        # Weighted Dynamic Aggregation
        self.lambda_weights = nn.Parameter(torch.ones(len(self.lpa_blocks)))  # Learnable weights
        self.output_projection = nn.Linear(decoder_dim, decoder_dim)  # Woutput projection

        # Post-FFN
        self.ffn1 = nn.Linear(decoder_dim, decoder_dim * 4)
        self.ffn2 = nn.Linear(decoder_dim * 4, decoder_dim)
        self.relu = nn.ReLU()
```

Here, we instantiate the ResNet Layer, instantiating a corresponding LPA block for each block in the layer.

Through Weighted Dynamic Aggregation (WDA), we dynamically aggregate the outputs of different LPA blocks.

Aggregated output is further processed through a post-FFN consisting of linear layers (self.ffn1 and self.ffn2) and ReLU activation functions (self.relu).

IMPLEMENTATION DETAILS

```python
def forward(self, x):
    # The ResNet instance `self.resnet` is now used directly
    resnet_outputs = self.resnet(x)
    lpa_outputs = []

    for i, output in enumerate(resnet_outputs):
        lpa_output = self.lpa_blocks[i](output)
        lpa_outputs.append(lpa_output)

    # Weighted Dynamic Aggregation
    print(lpa_outputs)
    wda = sum(self.lambda_weights[i] * lpa_outputs[i] for i in range(len(lpa_outputs)))
    wda = self.output_projection(wda)

    # Post-FFN
    print('reached ffn')
    mc = self.ffn1(wda)
    mc = self.relu(mc)
    print('Shape of tensor before self.ffn2:', mc.shape)
    mc = self.ffn2(mc)
    print('Shape of tensor after self.ffn2:', mc.shape)
    print(mc)
    return mc
```

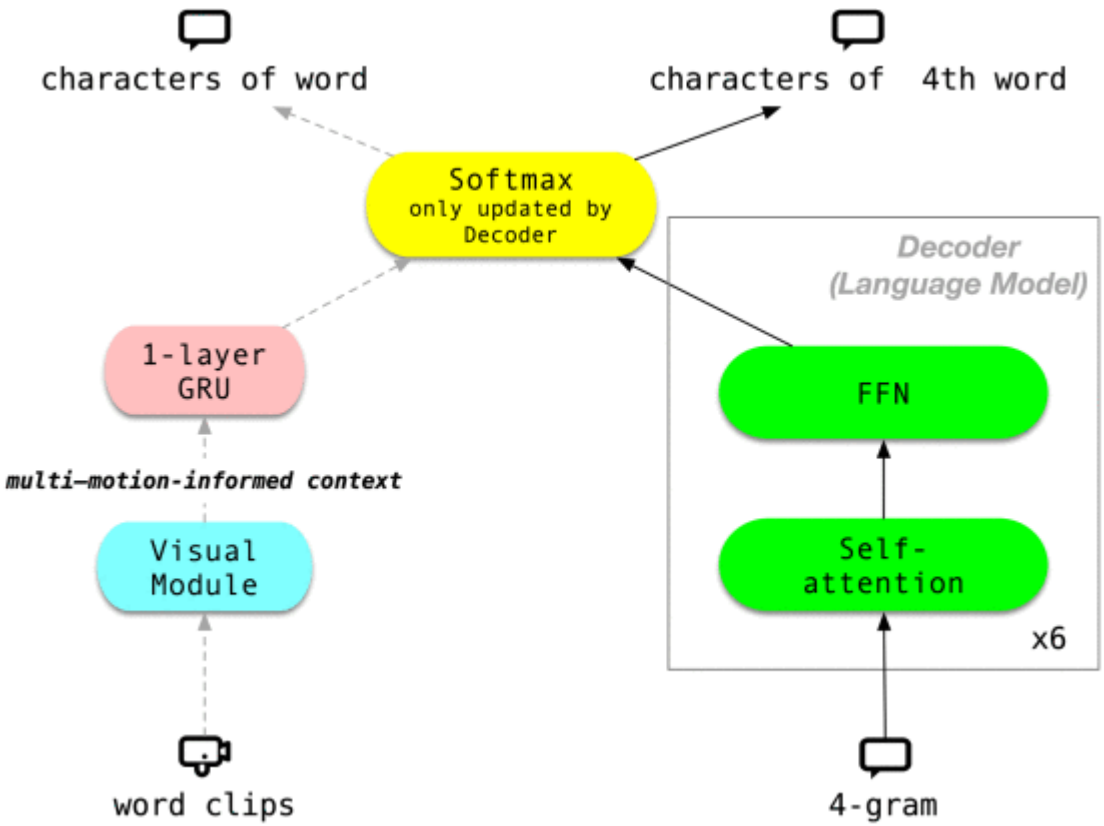
Output is the <u>multi-motion-informed context.</u>

It incorporates information from various temporal scales and motions in the input video data.

```
Shape of tensor after self.ffn2: torch.Size([1, 10, 1024])
tensor([[[ 0.7179, -0.5788,  1.7845,  ..., -4.8309, -0.7968, -1.1135],
         [ 0.7179, -0.5788,  1.7845,  ..., -4.8309, -0.7968, -1.1135],
         [ 0.7179, -0.5788,  1.7845,  ..., -4.8309, -0.7968, -1.1135],
         ...,
         [ 0.7178, -0.5787,  1.7845,  ..., -4.8309, -0.7968, -1.1134],
         [ 0.7178, -0.5785,  1.7845,  ..., -4.8309, -0.7968, -1.1134],
         [ 0.7178, -0.5787,  1.7845,  ..., -4.8309, -0.7968, -1.1134]]],
       device='cuda:0', grad_fn=<ViewBackward0>)
```

<- Example

IMPLEMENTATION DETAILS



After instantiation of visual module, we bridge the gap between the decoder and visual module through the pre-trained softmax layer from the decoder.

We deploy a 2-layer generator on top of the visual module to generate character probabilities over the character-level vocabulary. The probability generator consists : a 1-layer GRU [35] and the softmax layer of the decoder.

IMPLEMENTATION DETAILS

```
# Load pre-trained decoder model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

decoder_checkpoint = torch.load('/content/drive/MyDrive/model_epoch_3.pth', map_location=device)
decoder_model = TransformerDecoder(vocab_size, d_model=1024, N=6, heads=16, d_ff=4096, dropout=0.1).to(device)


# Load state_dict into the model
decoder_model.load_state_dict(decoder_checkpoint['model_state_dict'])

# Extract only the softmax layer
softmax_layer = decoder_model.out
```

```
class VisualModuleWithGRU(nn.Module):
    def __init__(self, visual_module, hidden_size, output_size, d_model, num_layers=1, softmax_layer=None):
        super(VisualModuleWithGRU, self).__init__()
        self.visual_module = visual_module
        # Ensure the input size to GRU matches the d_model of the visual module
        self.gru = nn.GRU(input_size=d_model, hidden_size=hidden_size, num_layers=num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

        # Assign the pre-trained softmax layer
        self.softmax_layer = softmax_layer

        # Freeze the softmax layer if it's provided
        if self.softmax_layer is not None:
            for param in self.softmax_layer.parameters():
                param.requires_grad = False

    def forward(self, x):
        # Ensure x is correctly reshaped or preprocessed to match what visual_module expects
        print('Input shape:', x.shape)
        visual_output = self.visual_module(x)  # Ensure this output matches GRU's input size expectation
        print('Visual output shape:', visual_output.shape)
        gru_output, _ = self.gru(visual_output)
        print('GRU output shape:', gru_output.shape)
        fc_output = self.fc(gru_output)
        print('FC output shape:', fc_output.shape)
        print("Shape of softmax layer weight matrix:", self.softmax_layer.weight.shape)
        if self.softmax_layer is not None:
            softmax_output = self.softmax_layer(fc_output)
        else:
            softmax_output = F.log_softmax(fc_output, dim=-1)  # Fallback if no softmax layer is provided

        return softmax_output
```

IMPLEMENTATION DETAILS

```python
# Instantiate the ResNet model with a specific configuration, for example, ResNet50
resnet_instance = ResNet(block, [3, 4, 6, 3], image_channels=3, num_classes=1000).to(device)

# Create the combined model with the ResNet instance
decoder_dim = 1024  # Example decoder dimension
visual_module = CombinedModelWithWDA(resnet_instance, decoder_dim).to(device)


# When initializing VisualModuleWithGRU, pass the d_model of ResNetWithLPAAndWDA
visual_model_with_gru = VisualModuleWithGRU(visual_module, hidden_size=1024, output_size=1024, d_model=1024, softmax_layer=softmax_layer).to(device)
```

```python
criterion = torch.nn.CrossEntropyLoss()  # Assuming you're using CrossEntropyLoss, a
optimizer = torch.optim.Adam(visual_model_with_gru.parameters(), lr=1e-4)

# Number of training epochs
num_epochs = 10

for epoch in range(num_epochs):
    visual_model_with_gru.train()  # Set model to training mode
    total_loss = 0.0

    # Wrap your data_loader with tqdm for a progress bar
    progress_bar = tqdm(data_loader, desc=f'Epoch {epoch+1}/{num_epochs}')

    for videos, targets in progress_bar:
        videos, targets = videos.to(device), targets.to(device)
        #print(videos[0])
        optimizer.zero_grad()
        print(targets)
        # Forward pass through your model
        outputs = visual_model_with_gru(videos)
        print("Output batch size:", outputs.size(0))
        print("Target batch size:", targets.size(0))

        # Calculate loss; you might need to adjust targets' shape to fit your needs
        loss = criterion(outputs.view(-1, outputs.size(-1)), targets.view(-1))

        # Backward pass and optimize
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        progress_bar.set_postfix(loss=total_loss / len(progress_bar))

    # Optionally save your model after each epoch
    torch.save({
        'epoch': epoch,
        'model_state_dict': visual_model_with_gru.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'loss': total_loss / len(data_loader),
    }, f'visual_module_with_gru_epoch_{epoch+1}.pth')
```

Through this integration of the decoder and the visual module, character probabilities are generated, and the pre-trained models are given as output. These bridged models will be fully integrated through cold fusion in the Integration Module.

IMPLEMENTATION DETAILS

```
torch.Size([12, 3, 224, 224])
tensor([[26, 17, 35, 15, 13, 31, 32, 24, 17,  0,  0,  0]], device='cuda:0')
Input shape: torch.Size([1, 12, 3, 224, 224])
torch.Size([1, 12, 64, 56, 56])
torch.Size([1, 12, 256, 56, 56])
torch.Size([1, 12, 512, 28, 28])
torch.Size([1, 12, 1024, 14, 14])
torch.Size([1, 12, 2048, 7, 7])
1: Input received
2: Input shape torch.Size([1, 12, 64, 56, 56])
3: Reshaped for pooling torch.Size([12, 64, 56, 56])
4: After pooling torch.Size([12, 64, 56, 56])
5: After flattening torch.Size([12, 200704])
6: After input projection torch.Size([12, 1024])
7: After reshaping back torch.Size([1, 12, 1024])
```

```
Shape of tensor before self.ffn2: torch.Size([1, 12, 4096])
Shape of tensor after self.ffn2: torch.Size([1, 12, 1024])
tensor([[[ 0.6451, -0.5699,  1.5868,  ..., -4.4236, -0.6532, -1.0554],
         [ 0.6445, -0.5704,  1.5870,  ..., -4.4219, -0.6527, -1.0548],
         [ 0.6440, -0.5710,  1.5871,  ..., -4.4205, -0.6523, -1.0544],
         ...,
         [ 0.6441, -0.5706,  1.5870,  ..., -4.4210, -0.6524, -1.0545],
         [ 0.6440, -0.5709,  1.5871,  ..., -4.4204, -0.6523, -1.0545],
         [ 0.6440, -0.5709,  1.5871,  ..., -4.4205, -0.6523, -1.0545]]],
       device='cuda:0', grad_fn=<ViewBackward0>)
Visual output shape: torch.Size([1, 12, 1024])
GRU output shape: torch.Size([1, 12, 1024])
FC output shape: torch.Size([1, 12, 1024])
Shape of softmax layer weight matrix: torch.Size([39, 1024])
```

As shown, the input shape is of [batch, t, c, n, m]. When passed through ResNet, it's converted to [batch,t,c,n,m], then follows the LPA algorithm for further extraction of information.

After aggregation, the dimension becomes [batch,t,C], which is then fed into post-ffn for final multi-motion-informed context. The GRU layer allows for probabilities to be generated through motion-informed context.

```
    torch.save({
        'epoch': epoch,
        'model_state_dict': visual_model_with_gru.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'loss': total_loss / len(data_loader),
    }, f'visual_module_with_gru_epoch_{epoch+1}.pth')

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {total_loss / len(data_loader):.4f}')
```

```
Epoch 1/10: 100%|██████████| 15021/15021 [2:39:18<00:00,  2.37s/it, loss=2.54]
Epoch 2/10: 100%|██████████| 15021/15021 [2:39:29<00:00,  2.30s/it, loss=2.48]
Epoch 3/10: 100%|██████████| 15021/15021 [2:39:51<00:00,  2.34s/it, loss=2.44]
Epoch 4/10: 100%|██████████| 15021/15021 [2:38:51<00:00,  2.29s/it, loss=2.41]
Epoch 5/10: 100%|██████████| 15021/15021 [2:39:20<00:00,  2.28s/it, loss=2.38]
Epoch 6/10: 100%|██████████| 15021/15021 [2:39:25<00:00,  2.35s/it, loss=2.37]
Epoch 7/10: 100%|██████████| 15021/15021 [2:38:55<00:00,  2.33s/it, loss=2.36]
Epoch 8/10: 100%|██████████| 15021/15021 [2:39:12<00:00,  2.31s/it, loss=2.36]
Epoch 9/10: 100%|██████████| 15021/15021 [2:39:40<00:00,  2.34s/it, loss=2.36]
Epoch 10/10: 100%|██████████| 15021/15021 [2:39:02<00:00,  2.26s/it, loss=2.36]
```

IMPLEMENTATION DETAILS

First, we load the pre-trained visual module and the transformer decoder model.

We now randomly instantiate the source-target attention layer for training. The input for the source-target layers is the multi-motion context from pre-training the visual module. In this way, cold fusion occurs.

```
if __name__ == "__main__":
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    vocab_size = len(d.vocab)  # This needs to be defined correctly before use
    decoder_instance = TransformerDecoder(vocab_size=vocab_size, d_model=1024, N=6, heads=16, d_ff=4

    decoder_checkpoint = torch.load('/content/drive/MyDrive/LRS/dec_model_epoch_1.pth', map_location
    decoder_instance.load_state_dict(decoder_checkpoint['model_state_dict'])

    softmax_layer=decoder_instance.out
```

```
[ ] resnet_instance = ResNet(block, [3, 4, 6, 3], image_channels=3, num_classes=1000).to(device)
    decoder_dim=1024
    visual_module_instance = CombinedModelWithWDA(resnet_instance, decoder_dim).to(device)  # Assuming i


    combined_model = VisualModuleWithGRU(visual_module_instance, hidden_size=1024, output_size=1024, d_m

    load_visual_module_from_combined('/content/drive/MyDrive/LRS/visual_module/visual_module_with_gru_ep

    visual_module=combined_model.visual_module

    # Training setup
    optimizer = optim.Adam(list(visual_module.parameters()) + list(decoder_instance.parameters()), lr=0.
    criterion = nn.CTCLoss(blank=0, zero_infinity=True).to(device)  # Or any other suitable loss functic

    # Dataset and DataLoader setup
    #dataset = PreprocessedDataset('path_to_dataset', 'lrs2', 'train')
    #dataloader = DataLoader(dataset, batch_size=4, shuffle=True)
    sos_token_id = 39
    # Training loop
```

IMPLEMENTAITON DETAILS

The outputs are tensors of character probabilities that are used to produce sequences of characters for transcripts. These probabilities will be further utilized during inference (beam search) for the final transcript.

```
[tensor([[[-1.3828, -0.2547,  0.3228,  ..., -1.3744, -0.0838, -0.3444],
          [-1.3835, -0.2556,  0.3234,  ..., -1.3742, -0.0842, -0.3446],
          [-1.3838, -0.2558,  0.3236,  ..., -1.3740, -0.0847, -0.3446],
          ...,
          [-1.3839, -0.2562,  0.3241,  ..., -1.3753, -0.0823, -0.3446],
          [-1.3810, -0.2533,  0.3218,  ..., -1.3763, -0.0803, -0.3440],
          [-1.3801, -0.2512,  0.3205,  ..., -1.3762, -0.0802, -0.3435]]],
        device='cuda:0', grad_fn=<UnsafeViewBackward0>), tensor([[[ 0.4376, -0.5051, -0.7441,  ...,  1.1032,  0.2342, -0.3877],
          [ 0.4376, -0.5051, -0.7441,  ...,  1.1032,  0.2342, -0.3877],
          [ 0.4376, -0.5051, -0.7441,  ...,  1.1032,  0.2342, -0.3877],
          ...,
          [ 0.4377, -0.5051, -0.7440,  ...,  1.1031,  0.2345, -0.3877],
          [ 0.4377, -0.5050, -0.7441,  ...,  1.1032,  0.2343, -0.3877],
          [ 0.4378, -0.5050, -0.7439,  ...,  1.1031,  0.2349, -0.3876]]],
        device='cuda:0', grad_fn=<UnsafeViewBackward0>), tensor([[[ 0.3778, -0.1190, -1.1567,  ...,  1.8425, -0.6215, -0.0472],
          [ 0.3778, -0.1190, -1.1567,  ...,  1.8426, -0.6216, -0.0472],
          [ 0.3778, -0.1190, -1.1567,  ...,  1.8426, -0.6216, -0.0472],
          ...,
          [ 0.3778, -0.1191, -1.1566,  ...,  1.8426, -0.6217, -0.0472],
          [ 0.3778, -0.1192, -1.1566,  ...,  1.8426, -0.6219, -0.0471],
          [ 0.3778, -0.1191, -1.1566,  ...,  1.8426, -0.6219, -0.0471]]],
        device='cuda:0', grad_fn=<UnsafeViewBackward0>), tensor([[[ 0.9173, -0.0406,  0.2962,  ..., -0.9092,  0.1906,  0.3222],
          [ 0.9173, -0.0406,  0.2962,  ..., -0.9092,  0.1906,  0.3222],
```

IMPLEMENTATION DETAILS

Dataset(RAVDESS) Preparation:

- **Necessary Python libraries for data manipulation, image processing, and machine learning are imported.**
- **A directory path to a dataset is set, and files in that directory are iterated through, extracting filenames, features, labels, and paths.**
- **This information is stored in separate lists for subsequent use.**

```
DATA PREPARATION
```

In [6]:
```
emotions = {1:'neutral', 2:'calm', 3:'happy', 4:'sad', 5:'angry', 6:'fear', 7:'disgust', 8:'surprise'}
emotional_intensity = {1:'normal', 2:'strong'}
```

In [7]:
```
import re
import os
import pandas as pd
import cv2
import random
import numpy as np
from tensorflow import keras
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA
from tqdm import tqdm
```

In [15]:
```
path = "C:/Users/Sheeba/Downloads/Datasets/RAVDESS"
```

In [16]:
```
filenames = []
feats = []
labels = []
paths = []

for (dirpath, dirnames, fn) in os.walk(path):
    for name in fn:
        filename = name.split('.')[0]
        feat = filename.split('-')[2:]
        label = feat[0]
        filenames.append(filename)
        feats.append(feat)
        labels.append(label)
        paths.append(dirpath + '/' + filename)

filenames[:5]
```

Out[16]:
```
['02-01-01-01-01-01-01',
 '02-01-01-01-01-02-01',
 '02-01-01-01-02-01-01',
 '02-01-01-01-02-02-01',
 '02-01-02-01-01-01-01']
```

IMPLEMENTATION DETAILS

DATASET EXPLORATION:

- **A pandas DataFrame df is created using the feats list, where each sublist represents a row.**
- **The columns are named 'emotion', 'emotional intensity', 'statement', 'repetition', and 'actor', and all values are converted to integers.**
- **Numeric values in the 'emotion' and 'emotional intensity' columns are mapped to their labels using dictionaries.**
- **The original filenames are added as an index to the DataFrame, which is then modified in place.**

```
DATA EXPLORATION
```

In [17]:

```
df = pd.DataFrame(feats, columns = ['emotion', 'emotional intensity', 'statement', 'repetition', 'actor']).astype(int)

df['emotion'] = df['emotion'].map(emotions)
df['emotional intensity'] = df['emotional intensity'].map(emotional_intensity)

df['index'] = filenames
df.set_index('index', inplace=True)

df
```

Out[17]:

index	emotion	emotional intensity	statement	repetition	actor
02-01-01-01-01-01-01	neutral	normal	1	1	1
02-01-01-01-01-02-01	neutral	normal	1	2	1
02-01-01-01-02-01-01	neutral	normal	2	1	1
02-01-01-01-02-02-01	neutral	normal	2	2	1
02-01-02-01-01-01-01	calm	normal	1	1	1
...	...	...	...	...	...
02-01-08-01-02-02-24	surprise	normal	2	2	24
02-01-08-02-01-01-24	surprise	strong	1	1	24
02-01-08-02-01-02-24	surprise	strong	1	2	24
02-01-08-02-02-01-24	surprise	strong	2	1	24
02-01-08-02-02-02-24	surprise	strong	2	2	24

1440 rows × 5 columns

IMPLEMENTATION DETAILS

PREPARE VIDEO FRAMES FROM RAVDESS DATASET:

- **For each video, frames are saved using the save_frames function. The progress of video processing is printed.**
- **In save_frames, a video capture object is initialized using OpenCV, and an empty list frames and frame counter count are initialized.**
- **If the current frame number meets specified conditions, the frame is resized and saved as a PNG image.**
- **Overall, videos are processed, and frames are extracted, resized, and saved as PNG images in specific directories, facilitating subsequent analysis or model training with video data.**

```
EXPORT FRAMES
```

```
398x224 NORMAL
```

In [18]:

```
def prepare_all_videos(filenames, paths, skip=1):
    nframes_tot = 0

    for count, video in enumerate(zip(filenames, paths)):
        # Gather all its frames
        save_frames(video[0], video[1], video[1].replace('RAVDESS', 'RAVDESS_frames'), skip)
        print(f"Processed videos {count+1}/{len(paths)}")
    return


def save_frames(filename, input_path, output_path, skip):
    # Initialize video reader
    cap = cv2.VideoCapture(input_path + '.mp4')
    frames = []
    count = 0

    if not os.path.exists(output_path):
        os.makedirs(output_path)
    try:
        # Loop through all frames
        while True:
            # Capture frame
            ret, frame = cap.read()
            if (count % skip == 0 and count > 20):
                #print(frame.shape)
                if not ret:
                    break
                frame = cv2.resize(frame, (398, 224))
                cv2.imwrite(output_path + '/' + f'{filename}_{count}' + '.png', frame)
            count += 1
    finally:
        cap.release()
    return
```

In [20]:

```
prepare_all_videos(filenames, paths, skip=3)
```

```
Processed videos 1421/1440
Processed videos 1422/1440
Processed videos 1423/1440
Processed videos 1424/1440
Processed videos 1425/1440
Processed videos 1426/1440
Processed videos 1427/1440
Processed videos 1428/1440
Processed videos 1429/1440
Processed videos 1430/1440
Processed videos 1431/1440
Processed videos 1432/1440
Processed videos 1433/1440
Processed videos 1434/1440
Processed videos 1435/1440
Processed videos 1436/1440
Processed videos 1437/1440
Processed videos 1438/1440
Processed videos 1439/1440
```

IMPLEMENTATION DETAILS

Similarly, Preparing RAVDESS Frames Black:

```
224X224 BLACK BACKGROUND
```

In [22]:

```python
def prepare_all_videos(filenames, paths, skip=1):
    nframes_tot = 0

    for count, video in enumerate(zip(filenames, paths)):
        # Gather all its frames
        save_frames(video[0], video[1], video[1].replace('RAVDESS', 'RAVDESS_frames_black'), skip)
        print(f"Processed videos {count+1}/{len(paths)}")
    return


def save_frames(filename, input_path, output_path, skip):
    # Initialize video reader
    cap = cv2.VideoCapture(input_path + '.mp4')
    frames = []
    count = 0

    if not os.path.exists(output_path):
        os.makedirs(output_path)
    try:
        # Loop through all frames
        while True:
            # Capture frame
            ret, frame = cap.read()
            if (count % skip == 0 and count > 20):
                #print(frame.shape)
                if not ret:
                    break
                #####
                gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)                  # background from white to black
                ret, thresh = cv2.threshold(gray, 220, 255, cv2.THRESH_BINARY)
                frame[thresh == 255] = 0
                #####
                frame = cv2.resize(frame, (398, 224))
                frame = frame[0:224, 87:311]
                cv2.imwrite(output_path + '/' + f'{filename}_{count}' + '.png', frame)
            count += 1
    finally:
        cap.release()
    return
```

In [23]:

```python
prepare_all_videos(filenames, paths, skip=3)
```

```
Processed videos 1421/1440
Processed videos 1422/1440
Processed videos 1423/1440
Processed videos 1424/1440
Processed videos 1425/1440
Processed videos 1426/1440
Processed videos 1427/1440
Processed videos 1428/1440
Processed videos 1429/1440
Processed videos 1430/1440
Processed videos 1431/1440
Processed videos 1432/1440
Processed videos 1433/1440
Processed videos 1434/1440
Processed videos 1435/1440
Processed videos 1436/1440
Processed videos 1437/1440
Processed videos 1438/1440
Processed videos 1439/1440
Processed videos 1440/1440
```

IMPLEMENTATION DETAILS

Similarly, Preparing RAVDESS Frames BW:

```
22X224 ONLY FACES BW
```

In [28]:

```python
def prepare_all_videos(filenames, paths, skip=1):
    nframes_tot = 0

    for count, video in enumerate(zip(filenames, paths)):
        # Gather all its frames
        save_frames(video[0], video[1], video[1].replace('RAVDESS', 'RAVDESS_frames_face_BW'), skip)
        print(f"Processed videos {count+1}/{len(paths)}")
    return


def save_frames(filename, input_path, output_path, skip):
    # Initialize video reader
    cap = cv2.VideoCapture(input_path + '.mp4')
    haar_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    frames = []
    count = 0

    if not os.path.exists(output_path):
        os.makedirs(output_path)
    try:
        # Loop through all frames
        while True:
            # Capture frame
            ret, frame = cap.read()
            if (count % skip == 0 and count > 20):
                #print(frame.shape)
                if not ret:
                    break
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                faces = haar_cascade.detectMultiScale(frame, scaleFactor=1.12, minNeighbors=9)
                # if len(faces) != 1:

                if len(faces) == 0:
                    faces = haar_cascade.detectMultiScale(frame, scaleFactor=1.02, minNeighbors=9)
                    if len(faces) == 0:
                        raise Exception(f"Still no faces {len(faces)} {filename}")
                if len(faces) > 1:
                    ex = []
                    print(type(faces))
                    for elem in faces:
                        for (x, y, w, h) in [elem]:
                            ex.append(frame[y:y + h, x:x + w])

                    print(filename)
                    plt.figure()
                    f, axarr = plt.subplots(4,1)
                    axarr[0].imshow(ex[0])
                    axarr[1].imshow(ex[1])
                    plt.show()

                    inp = int(input())
                    faces = [faces[inp]]
                #     raise Exception(f"More than 1 faces detected in {filename}")

                for (x, y, w, h) in faces:
                    face = frame[y:y + h, x:x + w]

                face = cv2.resize(face, (234, 234))
                face = face[5:-5, 5:-5]
                cv2.imwrite(output_path + '/' + f'{filename}_{count}' + '.png', face)
            count += 1
    finally:
        cap.release()
    return
```

In [29]:

```python
prepare_all_videos(filenames, paths, skip=3)
```

```
Processed videos 1421/1440
Processed videos 1422/1440
Processed videos 1423/1440
Processed videos 1424/1440
Processed videos 1425/1440
Processed videos 1426/1440
Processed videos 1427/1440
Processed videos 1428/1440
Processed videos 1429/1440
Processed videos 1430/1440
Processed videos 1431/1440
Processed videos 1432/1440
Processed videos 1433/1440
Processed videos 1434/1440
Processed videos 1435/1440
Processed videos 1436/1440
Processed videos 1437/1440
Processed videos 1438/1440
Processed videos 1439/1440
Processed videos 1440/1440
```

IMPLEMENTATION DETAILS

CALCULATE MEAN FACE FROM SAMPLED FRAMES FOR TRAINING:

- **The provided code defines dictionaries for emotion label transposition and mapping, and specifies paths and dimensions for face image processing.**
- **It initializes lists for training filenames and defines functions for sampling frames and computing the mean face from sampled face images.**
- **The code iterates through directories to extract filenames and labels for training, excluding neutral emotions and selecting only training actors.**
- **It samples frames to create a balanced dataset and computes the mean face from the sampled face images, which is then saved as a numpy array file.**

```
MEAN FACE
```

In [45]:
```
emotions_tras = {1:1, 2:4, 3:5, 4:0, 5:3, 6:2, 7:6}
emotions = {0:'angry', 1:'calm', 2:'disgust', 3:'fear', 4:'happy', 5:'sad', 6:'surprise'}

dataset_path = "C:/Users/Sheeba/Downloads/Datasets/RAVDESS_frames_face_BW/"

height_orig = 224
width_orig = 224
height_targ = 112
width_targ = 112

val_actors = ['19', '20']
test_actors = ['21', '22', '23', '24']
```

In [49]:
```
filenames_train = [] # train

for (dirpath, dirnames, fn) in os.walk(dataset_path):
    if fn != []:
        class_temp = int(fn[0].split('-')[2]) - 1
        if class_temp != 0:                                                    # exclude 'neutral' label
            if any(act in dirpath for act in (test_actors+val_actors))==False:  # select only train actors
                path = [os.path.join(dirpath, elem) for elem in fn]
                label = [emotions_tras[class_temp]] * len(fn)                  # emotion transposition
                filenames_train.append(list(zip(path, label)))
```

In [50]:
```
def sampling(list, num_frames_desired):
    tot = []
    for elem in list:
        sampled_list = random.sample(elem, num_frames_desired)
        tot += sampled_list
    return(tot)


def compute_mean_face(filenames):
    frames_per_vid = min([len(elem) for elem in filenames])     # number of frames per clip in order to have balanced classe
    print("frames per video:", frames_per_vid)

    filenames_sampled = sampling(filenames, frames_per_vid)
    random.shuffle(filenames_sampled)

    faces = []

    for path, label in tqdm(filenames_sampled):
        face = cv2.imread(path)
        face = cv2.resize(face, (112, 112))
        face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
        faces.append(face)

    faces = np.array(faces)
    mean_face = np.mean(faces, axis=0)
    mean_face = mean_face/255
    mean_face = np.expand_dims(mean_face, axis=2)
    np.save('C:/Users/Sheeba/Downloads/Other/mean_face.npy', mean_face)
```

In [51]:
```
compute_mean_face(filenames_train)
```

```
frames per video: 23
100%|██████████| 23184/23184 [00:46<00:00, 494.85it/s]
```

IMPLEMENTATION DETAILS

ORGANIZE FILENAMES FOR TRAINING AND VALIDATION DATASETS:

- **Filenames are organized for training and validation datasets.**
- **Neutral emotions and specific actors are excluded.**

```
VIDEO STREAM TRAIN
```

In [57]:

```
import numpy as np
# import pandas as pd
import matplotlib.pyplot as plt
import os
import random
from datetime import datetime

import tensorflow as tf

# Keras module and tools
from keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
from keras.layers import Input, Flatten, Conv2D, BatchNormalization, Activation, MaxPooling2D, Dense, Dropout, RandomFlip
from keras.metrics import sparse_categorical_accuracy, sparse_top_k_categorical_accuracy
from keras.losses import sparse_categorical_crossentropy
from keras.models import Model, Sequential
from keras.optimizers import Adam
from keras_cv.layers import RandomCutout
import keras_tuner
```

In [63]:

```
emotions_tras = {1:1, 2:4, 3:5, 4:0, 5:3, 6:2, 7:6} # to match the audio stream labels
emotions = {0:'angry', 1:'calm', 2:'disgust', 3:'fear', 4:'happy', 5:'sad', 6:'surprise'}

path_frames_face_BW = "C:/Users/Sheeba/Downloads/Datasets/RAVDESS_frames_face_BW/"
path_frames = "C:/Users/Sheeba/Downloads/Datasets/RAVDESS_frames/"

height_orig = 224
width_orig = 224
height_targ = 112
width_targ = 112

batch_size = 64
num_classes = len(emotions)

val_actors = ['19', '20']
test_actors = ['21', '22', '23', '24']
```

IMPLEMENTATION DETAILS

PREPARE DATASET BY SAMPLING FRAMES FOR BALANCED CLASSES:

- **Dictionaries are initialized for transposing emotion labels and mapping them to textual representations.**

- **Paths for accessing directories containing face images are defined, and dimensions for image processing are set. Actors are excluded and selected for training and validation sets based on predefined identifiers.**

- **File paths and emotion labels are prepared for training and validation datasets. Overall, a structured framework is established for organizing and preprocessing data essential for subsequent model training in emotion recognition tasks.**

```
MODEL: 112X112 BW FACES
```

In [ ]:
```
DATASET CREATION
```

In [64]:
```
filenames_train = [] # train
filenames_val = [] # validation

for (dirpath, dirnames, fn) in os.walk(path_frames_face_BW):
    if fn != []:
        class_temp = int(fn[0].split('-')[2]) - 1
        if class_temp != 0:                                                  # exclude 'neutral' label
            if any(act in dirpath for act in (test_actors+val_actors))==False:  # select only train actors
                path = [os.path.join(dirpath, elem) for elem in fn]
                label = [emotions_tras[class_temp]] * len(fn)                # emotion transposition
                filenames_train.append(list(zip(path, label)))

            if any(act in dirpath for act in val_actors):                    # select only validation actors
                path = [os.path.join(dirpath, elem) for elem in fn]
                label = [emotions_tras[class_temp]] * len(fn)
                filenames_val.append(list(zip(path, label)))
```

In [65]:
```
def sampling(list, num_frames_desired):
    tot = []
    for elem in list:
        sampled_list = random.sample(elem, num_frames_desired)
        tot += sampled_list
    return(tot)

def parse_image(filename):
    image = tf.io.read_file(filename)
    image = tf.image.decode_jpeg(image, channels=1)
    image = tf.image.convert_image_dtype(image, dtype=tf.float16)
    image = tf.image.resize_with_crop_or_pad(image, height_orig, width_orig)
    image = tf.image.resize(image, [height_targ, width_targ])
    print('shape frames:', image.shape)
    return image

def configure_for_performance(ds):
    ds = ds.shuffle(buffer_size=1000) # serve?
    ds = ds.batch(batch_size)
    ds = ds.repeat()
    ds = ds.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
    return ds
def load_dataset(filenames, batch_size):
    frames_per_vid = min([len(elem) for elem in filenames])     # number of frames per clip in order to have balanced classe
    print("frames per video:", frames_per_vid)

    filenames_sampled = sampling(filenames, frames_per_vid)
    random.shuffle(filenames_sampled)

    zipped = [list(t) for t in zip(*filenames_sampled)]

    names = zipped[0]
    labels = zipped[1]

    names = tf.data.Dataset.from_tensor_slices(names)
    images = names.map(parse_image, num_parallel_calls=tf.data.experimental.AUTOTUNE)

    labels = [elem for elem in labels]
    labels = tf.data.Dataset.from_tensor_slices(labels)

    ds = tf.data.Dataset.zip((images, labels))
    ds = configure_for_performance(ds)

    frame_number = len(filenames_sampled)
    step_per_epoch = frame_number // batch_size
    print('frames number:', frame_number, '\nbatch size:', batch_size, '\nbatch number:', step_per_epoch)
    return ds, step_per_epoch
```

IMPLEMENTATION DETAILS

- **The dataset for training is loaded using the provided filenames and batch size, resulting in a training dataset (train_ds) and the number of steps per epoch (step_per_epoch_train).**

- **Each video contains an average of 23 frames, with a total of 23,184 frames across all videos. The batch size is set to 64, resulting in 362 batches for training.**

- **Similarly, the validation dataset (val_ds) and the number of steps per epoch for validation (step_per_epoch_val) are loaded, with each video containing an average of 24 frames and a total of 2,688 frames across all validation videos. The batch size for validation is also set to 64.**

```
In [66]:
1 train_ds, step_per_epoch_train = load_dataset(filenames_train, batch_size)
```

```
frames per video: 23
shape frames: (112, 112, 1)
frames number: 23184
batch size: 64
batch number: 362
```

```
In [77]:
1 val_ds, step_per_epoch_val = load_dataset(filenames_val, batch_size)
```

```
frames per video: 24
shape frames: (112, 112, 1)
frames number: 2688
batch size: 64
batch number: 42
```

```
1 DATA AUGMENTATION
```

```
In [78]:
1 data_augmentation = tf.keras.Sequential([
2     RandomFlip("horizontal"),
3     RandomCutout(0.4, 0.4, fill_mode="constant", fill_value=0.0, seed=None)
4 ])
```

IMPLEMENTATION DETAILS

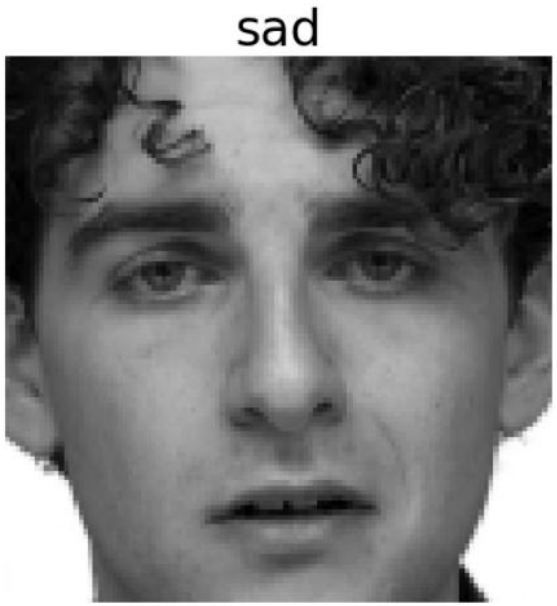
- **A TensorFlow Sequential model is defined to represent the data augmentation pipeline, consisting of two augmentation layers: RandomFlip for horizontal flipping and RandomCutout for randomly cutting out portions of the image.**

- **The fill mode is set to constant with a fill value of 0.0.**

```
1  EXAMPLES
```

```
In [79]:
1  examples = train_ds.unbatch().take(1)
2
3  for elem in examples:
4      print(elem[0].shape)
5  plt.imshow(elem[0], cmap='gray')
6  plt.title(emotions[int(elem[1])], fontdict={'fontsize': 24})
7  plt.axis('off')
8  # plt.savefig('Plots/model4_input.png')
```

```
(112, 112, 1)
Out[79]: (-0.5, 111.5, 111.5, -0.5)
```



IMPLEMENTATION DETAILS

- **Next, a batch of images and labels is retrieved from the training dataset (train_ds) using the take(1) method. A subplot grid with 3 rows and 3 columns is created using matplotlib, and the title of the figure is set to the corresponding emotion label extracted from the labels.**

- **The first image from the batch is selected, and the data augmentation pipeline is applied to it nine times. Each augmented image is displayed in a subplot, and the axis is turned off for each subplot to display only the image.**

- **Finally, the figure is saved as an image file named "model4_augmentation.png" at the specified path.**

In [80]:
```
for images, labels in train_ds.take(1):
    fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(10, 10))
    fig.suptitle(emotions[int(labels[0])], fontsize=40)
    first_image = images[0]
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        augmented_image = data_augmentation(
            tf.expand_dims(first_image, 0), training=True
        )
        plt.imshow(augmented_image[0], cmap='gray')
        plt.axis("off")
plt.savefig('C:/Users/Sheeba/Downloads/Plots/model4_augmentation.png')
```

IMPLEMENTATION DETAILS

MODEL ARCHITECTURE AND TRAINING:

- **Training parameters and callbacks are defined, and a data augmentation model is created to prepare the training images.**

- **A convolutional neural network architecture specifically designed for emotion recognition is constructed.**

- **The model is compiled with appropriate settings and a summary of its architecture is generated.**

In [83]:

```
input = Input(shape=(width_targ, height_targ, 1))
x = input
x = data_augmentation(x)

x = Conv2D(filters=32, kernel_size=(5,5), padding='same', kernel_initializer='he_normal', name='conv2d_0')(x)
x = BatchNormalization(name='batchnorm_0')(x)
x = Activation('elu')(x)
x = MaxPooling2D(pool_size=(2,2), name='maxpool2d_0')(x)

x = Conv2D(filters=64, kernel_size=(5,5), padding='same', kernel_initializer='he_normal', name='conv2d_1')(x)
x = BatchNormalization(name='batchnorm_1')(x)
x = Activation('elu')(x)
x = MaxPooling2D(pool_size=(2,2), name='maxpool2d_1')(x)

x = Dropout(0.5, name='dropout_1')(x)

x = Conv2D(filters=128, kernel_size=(3,3), padding='same', kernel_initializer='he_normal', name='conv2d_2')(x)
x = BatchNormalization(name='batchnorm_2')(x)
x = Activation('elu')(x)
x = MaxPooling2D(pool_size=(2,2), name='maxpool2d_2')(x)

x = Dropout(0.5, name='dropout_2')(x)

x = Conv2D(filters=256, kernel_size=(3,3), padding='same', kernel_initializer='he_normal', name='conv2d_3')(x)
x = BatchNormalization(name='batchnorm_3')(x)
x = Activation('elu')(x)
x = MaxPooling2D(pool_size=(2,2), name='maxpool2d_3')(x)

x = Dropout(0.5, name='dropout_3')(x)

# x = Conv2D(filters=256, kernel_size=(3,3), padding='same', kernel_initializer='he_normal', name='conv2d_4')(x)
# x = BatchNormalization(name='batchnorm_x')(x)
# x = Activation('elu')(x)
# x = MaxPooling2D(pool_size=(2,2), name='maxpool2d_4')(x)

# x = Dropout(0.5, name='dropout_4')(x)

x = Flatten(name='flatten')(x)
x = Dense(128, kernel_initializer='he_normal', name='dense_1')(x)
x = BatchNormalization(name='batchnorm_4')(x)
x = Activation('elu')(x)

x = Dropout(0.6, name='dropout_4')(x)

x = Dense(num_classes, activation='softmax', name='out_layer')(x)

output = x

net_4 = Model(inputs=input, outputs=output)
net_4.summary()
```

IMPLEMENTATION DETAILS

- The model is prepared for training by defining hyperparameters and callbacks, and by creating a data augmentation model to enhance the training data.
- A convolutional neural network architecture specifically designed for recognizing emotions is built. This architecture utilizes convolutional layers with various functionalities to extract features from the images.
- To optimize the training process, the model is compiled with a suitable loss function and optimizer.
- To understand the model's structure and complexity, a summary of its architecture is generated.

In [84]:
```
net_4.compile(
        optimizer = Adam(learning_rate=lr),
        # optimizer=keras.optimizers.SGD(learning_rate=learningrate, momentum=momentum)
        loss = sparse_categorical_crossentropy,
        metrics = [sparse_categorical_accuracy],
    )
```

```
TRAINING
```

In [85]:
```
history_4 = net_4.fit(train_ds,
                    epochs=epochs,
                    validation_data=val_ds,
                    batch_size=batch_size,
                    steps_per_epoch=step_per_epoch_train,
                    validation_steps=step_per_epoch_val,
                    # callbacks=[reduce_lr, early_stop, save_best],
                    # callbacks=[save_best],
                    verbose=1)
```

```
Epoch 1/10
362/362 [==============================] - 592s 2s/step - loss: 1.8849 - sparse_categorical_accuracy: 0.3566 - val_loss: 2.9731
- val_sparse_categorical_accuracy: 0.2478
Epoch 2/10
362/362 [==============================] - 594s 2s/step - loss: 1.2112 - sparse_categorical_accuracy: 0.5529 - val_loss: 2.2446
- val_sparse_categorical_accuracy: 0.3367
Epoch 3/10
362/362 [==============================] - 589s 2s/step - loss: 0.9618 - sparse_categorical_accuracy: 0.6479 - val_loss: 1.8370
- val_sparse_categorical_accuracy: 0.4371
Epoch 4/10
362/362 [==============================] - 577s 2s/step - loss: 0.8356 - sparse_categorical_accuracy: 0.6993 - val_loss: 1.8647
- val_sparse_categorical_accuracy: 0.4431
Epoch 5/10
362/362 [==============================] - 554s 2s/step - loss: 0.7435 - sparse_categorical_accuracy: 0.7325 - val_loss: 2.0286
- val_sparse_categorical_accuracy: 0.4178
Epoch 6/10
362/362 [==============================] - 551s 2s/step - loss: 0.6722 - sparse_categorical_accuracy: 0.7594 - val_loss: 1.9905
- val_sparse_categorical_accuracy: 0.4390
Epoch 7/10
362/362 [==============================] - 548s 2s/step - loss: 0.6091 - sparse_categorical_accuracy: 0.7858 - val_loss: 1.8621
- val_sparse_categorical_accuracy: 0.4542
Epoch 8/10
362/362 [==============================] - 547s 2s/step - loss: 0.5602 - sparse_categorical_accuracy: 0.8013 - val_loss: 2.1706
- val_sparse_categorical_accuracy: 0.4126
Epoch 9/10
362/362 [==============================] - 533s 1s/step - loss: 0.5222 - sparse_categorical_accuracy: 0.8151 - val_loss: 1.8086
- val_sparse_categorical_accuracy: 0.4903
Epoch 10/10
362/362 [==============================] - 547s 2s/step - loss: 0.4848 - sparse_categorical_accuracy: 0.8294 - val_loss: 2.0236
- val_sparse_categorical_accuracy: 0.5093
```

IMPLEMENTATION DETAILS

MODEL EVALUATION:

- **The CNN model is set up for training with an Adam optimizer, a specific learning rate, and a loss function for multi-class classification. Accuracy is used to evaluate its performance.**
- **The model is trained on the provided data for a set number of epochs, using validation data to monitor progress.**
- **Training details like batch size and number of steps are defined to control how the data is processed.**

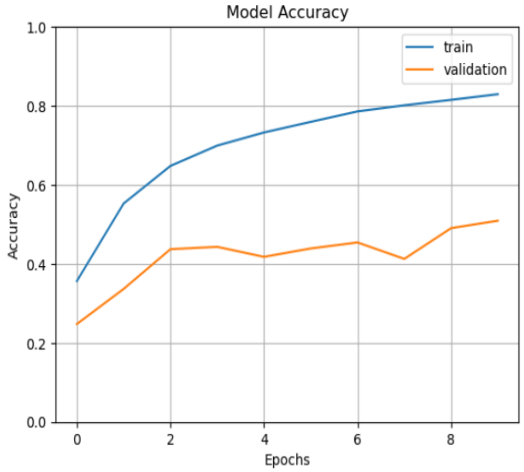
```
1  EVALUATION
```

```
In [86]:
1  net_4.evaluate(val_ds,
2             batch_size=batch_size,
3             steps=step_per_epoch_val)
```

```
42/42 [==============================] - 13s 302ms/step - loss: 2.0353 - sparse_categorical_accuracy: 0.5052
```

```
Out[86]: [2.035303831100464, 0.5052083134651184]
```

```
In [87]:
1  fig1 = plt.gcf()
2  plt.plot(history_4.history['sparse_categorical_accuracy'])
3  plt.plot(history_4.history['val_sparse_categorical_accuracy'])
4  plt.axis(ymin=0, ymax=1)
5  plt.grid()
6  plt.title('Model Accuracy')
7  plt.ylabel('Accuracy')
8  plt.xlabel('Epochs')
9  plt.legend(['train', 'validation'])
10 # plt.savefig('Plots/model4_accuracy.png')
11 plt.show()
```



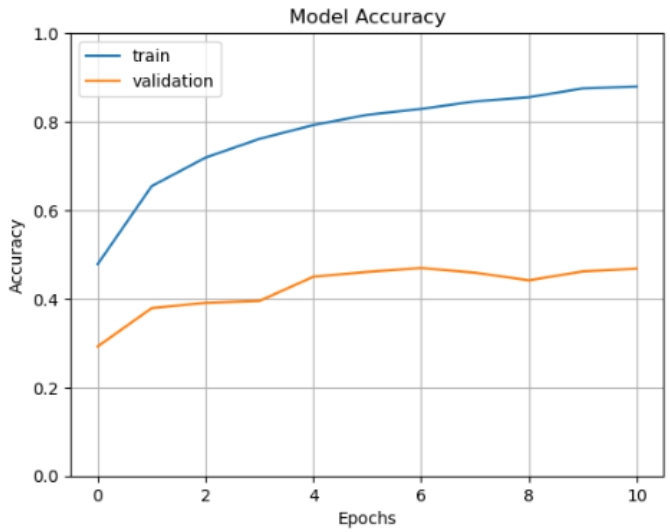
IMPLEMENTATION DETAILS

- **With early stopping, accuracy was improved**

In [41]:
```
net_4.compile(
    optimizer=optimizers.Adam(learning_rate=lr),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)
```

In [42]:
```
history_4 = net_4.fit(train_ds,
                      epochs=epochs,
                      validation_data=val_ds,
                      batch_size=batch_size,
                      steps_per_epoch=400,
                      validation_steps=400,
                      callbacks=[reduce_lr, early_stop, save_best],
                      verbose=1)
```

```
Epoch 1/15
400/400 [==============================] - 863s 2s/step - loss: 1.5318 - sparse_categorical_accuracy: 0.4784 - val_loss: 2.4616
- val_sparse_categorical_accuracy: 0.2925 - lr: 0.0010
Epoch 2/15
400/400 [==============================] - 817s 2s/step - loss: 0.9502 - sparse_categorical_accuracy: 0.6544 - val_loss: 2.1602
- val_sparse_categorical_accuracy: 0.3789 - lr: 0.0010
Epoch 3/15
400/400 [==============================] - 837s 2s/step - loss: 0.7823 - sparse_categorical_accuracy: 0.7189 - val_loss: 2.1774
- val_sparse_categorical_accuracy: 0.3909 - lr: 0.0010
Epoch 4/15
400/400 [==============================] - 855s 2s/step - loss: 0.6759 - sparse_categorical_accuracy: 0.7612 - val_loss: 2.2042
- val_sparse_categorical_accuracy: 0.3952 - lr: 0.0010
Epoch 5/15
400/400 [==============================] - 848s 2s/step - loss: 0.5834 - sparse_categorical_accuracy: 0.7923 - val_loss: 2.1555
- val_sparse_categorical_accuracy: 0.4498 - lr: 0.0010
Epoch 6/15
400/400 [==============================] - 844s 2s/step - loss: 0.5211 - sparse_categorical_accuracy: 0.8155 - val_loss: 2.1245
- val_sparse_categorical_accuracy: 0.4607 - lr: 0.0010
Epoch 7/15
400/400 [==============================] - 947s 2s/step - loss: 0.4879 - sparse_categorical_accuracy: 0.8288 - val_loss: 2.1336
- val_sparse_categorical_accuracy: 0.4694 - lr: 0.0010
Epoch 8/15
400/400 [==============================] - 1176s 3s/step - loss: 0.4413 - sparse_categorical_accuracy: 0.8457 - val_loss: 2.139
2 - val_sparse_categorical_accuracy: 0.4589 - lr: 0.0010
Epoch 9/15
400/400 [==============================] - ETA: 0s - loss: 0.4156 - sparse_categorical_accuracy: 0.8552
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
400/400 [==============================] - 4116s 10s/step - loss: 0.4156 - sparse_categorical_accuracy: 0.8552 - val_loss: 2.32
25 - val_sparse_categorical_accuracy: 0.4417 - lr: 0.0010
Epoch 10/15
400/400 [==============================] - 939s 2s/step - loss: 0.3656 - sparse_categorical_accuracy: 0.8751 - val_loss: 2.2850
- val_sparse_categorical_accuracy: 0.4618 - lr: 5.0000e-04
Epoch 11/15
400/400 [==============================] - ETA: 0s - loss: 0.3430 - sparse_categorical_accuracy: 0.8793
Epoch 11: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
Restoring model weights from the end of the best epoch: 7.
400/400 [==============================] - 881s 2s/step - loss: 0.3430 - sparse_categorical_accuracy: 0.8793 - val_loss: 2.2468
- val_sparse_categorical_accuracy: 0.4680 - lr: 5.0000e-04
Epoch 11: early stopping
```



In [51]:
```
from sklearn.metrics import classification_report
y_val = []
y_pred = []
i = 1
for image, target in train_ds:
  if i > step_per_epoch_val:
    break
  y_val.extend(target.numpy())
  y_pred.extend(net_4(image, training=False).numpy().argmax(axis=1))
  i+=1

print(classification_report(y_val, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.67	0.80	405
1	0.90	0.96	0.93	377
2	0.89	0.95	0.92	354
3	0.81	0.96	0.88	373
4	0.97	0.95	0.96	384
5	0.93	0.86	0.89	394
6	0.80	0.90	0.85	401
accuracy			0.89	2688
macro avg	0.90	0.89	0.89	2688
weighted avg	0.90	0.89	0.89	2688

IMPLEMENTATION DETAILS

- The trained model is tested on the validation data to see how well it performs. This includes calculating a loss value and accuracy score.
- The results of the evaluation, like a loss of around 2.0353 and accuracy of around 0.5052, are displayed.
- A graph is created to visualize how the model's accuracy improves over time for both the training and validation data.

In [88]:

```
from sklearn.metrics import classification_report
y_val = []
y_pred = []
i = 1
for image, target in val_ds:
  if i > step_per_epoch_val:
    break
  y_val.extend(target.numpy())
  y_pred.extend(net_4(image, training=False).numpy().argmax(axis=1))
  i+=1

print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.57      0.49      0.53       384
           1       0.37      0.97      0.53       384
           2       0.57      0.67      0.62       384
           3       0.42      0.13      0.20       384
           4       0.68      0.53      0.60       384
           5       0.64      0.39      0.49       384
           6       0.52      0.32      0.39       384

    accuracy                           0.50      2688
   macro avg       0.54      0.50      0.48      2688
weighted avg       0.54      0.50      0.48      2688
```

IMPLEMENTATION DETAILS

TESTING THE MODELS

The trained models are tested using a part of the dataset first as a single frame and then the full video.

```
In [ ]:  1  Prediction
```

```
In [ ]:  1  Evaluate Single Frame
```

```
In [124]:  1  loss_single_frame, acc_single_frame = reconstructed_model.evaluate(test_ds, steps=step_per_epoch_test)
           2  print('accuratezza single frame:', round(acc_single_frame, 4))
```

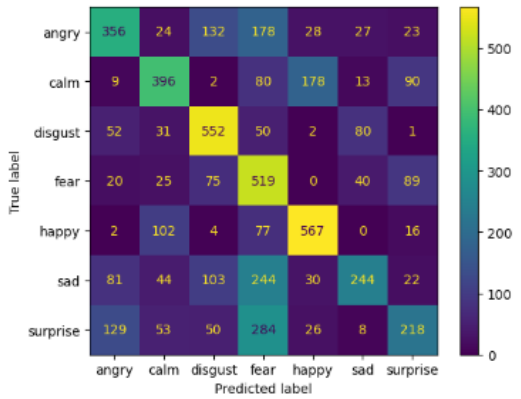
```
224/224 [==============================] - 27s 120ms/step - loss: 1.5862 - sparse_categorical_accuracy: 0.5305
accuratezza single frame: 0.5305
```

```
In [125]:  1  predic = reconstructed_model.predict(test_ds, steps=step_per_epoch_test).argmax(axis=1)
           2  ground = [elem[0][1]  for elem in filenames_test]
           3  ground = np.repeat(ground, frames_per_vid)
```

```
224/224 [==============================] - 27s 121ms/step
```

```
In [126]:  1  cm = confusion_matrix(ground, predic)
           2  disp = ConfusionMatrixDisplay(cm, display_labels=list(emotions.values()))
           3  disp.plot()
           4  # plt.savefig('PLots/VideoCM_Frame.png')
```

```
Out[126]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21fd1a7d7e0>
```




```
In [127]:  1  recall = np.diag(cm) / np.sum(cm, axis = 1)
           2  precision = np.diag(cm) / np.sum(cm, axis = 0)
           3
           4  stats = pd.DataFrame({'Recall': recall, 'Precision': precision}, index=emotions.values())
           5  round(stats, 2)
```

```
Out[127]:
```

	Recall	Precision
angry	0.46	0.55
calm	0.52	0.59
disgust	0.72	0.60
fear	0.68	0.36
happy	0.74	0.68
sad	0.32	0.59
surprise	0.28	0.47

IMPLEMENTATION DETAILS

TESTING THE MODELS

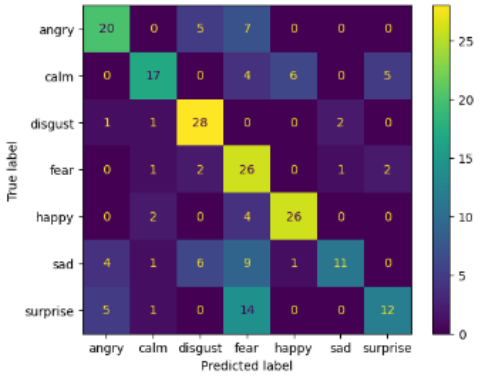
The trained models are tested using a part of the dataset first as a single frame and then the full video.

In [130]:
```
pred_list = np.array_split(pred, num_videos)

ground = []
predic = []

i = 0
for count, video in enumerate(pred_list):
    predic.append(np.mean(video, axis=0).argmax())
    # predic.append(np.average(video, axis=0, weights=np.max(video, axis=1)).argmax()) # weighted mean on highest predictior
    ground.append(filenames_test[count][0][1])

    if predic[count] == ground[count]:
        i += 1

print('accuracy full video:\t', round(i/len(pred_list), 4))
print('Num videos:\t\t', count+1)
```

```
accuracy full video:    0.625
Num videos:             224
```

In [131]:
```
cm = confusion_matrix(ground, predic)
disp = ConfusionMatrixDisplay(cm, display_labels=list(emotions.values()))
disp.plot()
# plt.savefig('Plots/VideoCM_FullVideo.png')
```

Out[131]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21fd1b300d0>



In [132]:
```
recall = np.diag(cm) / np.sum(cm, axis = 1)
precision = np.diag(cm) / np.sum(cm, axis = 0)

stats = pd.DataFrame({'Recall': recall, 'Precision': precision}, index=emotions.values())
round(stats, 2)
```

Out[132]:

	Recall	Precision
angry	0.62	0.67
calm	0.53	0.74
disgust	0.88	0.68
fear	0.81	0.41
happy	0.81	0.79
sad	0.34	0.79
surprise	0.38	0.63

IMPLEMENTATION DETAILS

We load the trained models, aka the lip reading model and the sentiment analysis model for final probability generation. During this phase, we utilize beam search for generating transcripts.

```
class CombinedModel(nn.Module):
    def __init__(self, visual_module, decoder):
        super(CombinedModel, self).__init__()
        self.visual_module = visual_module
        self.decoder = decoder
        self.sos_token_id=39
        self.s_analysis = keras.models.load_model('/content/drive/MyDrive/sent_analysis.h5')
        self.s_analysis.trainable = False
    def forward(self, x, state=None, step=False):
            video_frames=x
            # Full sequence processing for training
            memory = self.visual_module(x)
            #print(memory.size())
            #memory = memory.transpose(0,1)  # Ensure memory is in shape [batch_size, seq_len, feat
            batch_size = 1
            initial_input = torch.full((1, batch_size), self.sos_token_id, dtype=torch.long, device

            predictions=[]
            for i in range(video_frames.size(1)+20):  # Assuming transcripts have been properly pad
                if i == 0:
                    # Start the sequence with [SOS] token
                    output = decoder_instance(initial_input, memory, use_cross_attention=True)
                else:
                    last_token_indices = output[:, -1, :].max(dim=-1)[1].unsqueeze(1).type(torch.lo
                    output = decoder_instance(last_token_indices, memory, use_cross_attention=True)

                predictions.append(output)
            return predictions
```

IMPLEMENTATION DETAILS

```
Using TensorFlow backend
CombinedModel(
  (visual_module): CombinedModelWithWDA(
    (resnet): ResNet(
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
      (layer1): Sequential(
        (0): block(
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
          (identity_downsample): Sequential(
            (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          )
        )
        (1): block(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (2): block(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

IMPLEMENTATION DETAILS

```python
import torch

def beam_search_decoder(probabilities, beam_width=12):
    """
    Perform beam search decoding given a sequence of softmax probabilities from a model.

    Arguments:
    probabilities -- Tensor of shape (sequence_length, vocab_size) containing softmax probabilit
    beam_width -- int, the number of sequences to consider at each step (beam width).

    Returns:
    top_sequences -- List of tuples (sequence, probability) for the top beam_width sequences.
    """
    # Initialize the beam with the start of the sequence
    # Each element in the beam is a tuple (probability, [sequence of token indices])
    start_token_idx = 39  # Assuming 0 is the index of the start token [SOS]
    beam = [(1.0, [start_token_idx])]
    print(probabilities)
    # Iterate over the sequence of probabilities
    for step in range(1, probabilities.size(0)):
        new_beam = []
        for prob, seq in beam:
            # Consider all possible next tokens
            for token_idx in range(probabilities.size(1)):
                new_prob = prob * probabilities[step, token_idx].item()
                new_seq = seq + [token_idx]
                new_beam.append((new_prob, new_seq))

        # Sort all sequences in the new_beam by probability in descending order
        new_beam.sort(reverse=True, key=lambda x: x[0])

        # Select the top beam_width sequences to keep in the beam
        beam = new_beam[:beam_width]

    return beam
```

IMPLEMENTATION DETAILS

	Video Frames	Ground Truth	Transcript
0	[[[tensor([[1.6324, 1.6275, 1.6128, ..., ...	(I'M STILL HAPPY,)	WHAT TIME STILL
1	[[[tensor([[0.7248, 0.7248, 0.7248, ..., ...	(BUT BEFORE I DO,)	REMEMBER BEFORE I DO
2	[[[tensor([[-1.0390, -1.0293, -0.9999, ..., ...	(THERE'S NO OBVIOUS REASON,)	
3	[[[tensor([[1.7694, 1.7719, 1.7792, ..., ...	(YOU NEVER KNOW BY THE END OF THE BUILD,)	YOU NEVER KNOW BY THE END OF THE BUILD
4	[[[tensor([[0.6221, 0.6147, 0.5927, ..., ...	(FROM SMALL SHIPS,)	FROM SMALL SHIPS
5	[[[tensor([[-1.1559e-02, -1.4006e-02, -2.1345e...	(FOR THE FIRST TIME,)	SEARCH FOR THE FIRST
6	[[[tensor([[-0.4054, -0.3981, -0.3761, ..., -...	(IT WILL BE A GREAT INVESTMENT,)	IT WILL BE A GREAT INVESTMENT
7	[[[tensor([[0.1597, 0.1597, 0.1597, ..., 1.63...	(PROBABLY AROUND 85,)	PROBABLY AROUND 8
8	[[[tensor([[-2.1179, -2.1179, -2.1179, ..., ...	(WE'VE GOT A GREAT RELATIONSHIP,)	WE'VE GOT A GREAT RELATIONSHIP
9	[[[tensor([[0.2796, 0.2869, 0.3089, ..., ...	(OVER THE COURSE OF HIS LIFE,)	WHO WERE THE CAUSE OF HIS
10	[[[tensor([[-0.4397, -0.4323, -0.4103, ..., ...	(WHO'S VERY MUCH INTO FAMILY HISTORY,)	WHO'S VERY MUCH INTO FAMILY HISTORY
11	[[[tensor([[-0.5082, -0.5277, -0.5865, ..., -...	(PRETTY ON THE OUTSIDE,)	PLAY PRETTY ON THE OUTSIDE

METRICS FOR EVALUATION

1.Word Error Rate (WER):

- WER is used to measure the accuracy of recognized words compared to the ground truth. This metric accounts for substitutions, insertions, and deletions in the recognized words.
- WER = (S + D + I) / N

where:

- S: Number of substitutions (words in the reference that are replaced in the transcription).
- D: Number of deletions (words in the reference that are missing in the transcription).
- I: Number of insertions (extra words in the transcription that are not in the reference).
- N: Total number of words in the reference.

METRICS FOR EVALUATION

2. Character Error Rate (CER):

- Character Error Rate (CER) is a metric used to measure the quality of the transcription by calculating the rate of character-level errors in the recognized text compared to the ground truth (i.e., the correct reference text). It is expressed as a percentage.

- CER = (S + D + I) / N

where:

- S: Number of substitutions (characters in the reference that are replaced in the transcription).
- D: Number of deletions (characters in the reference that are missing in the transcription).
- I: Number of insertions (extra characters in the transcription that are not in the reference).
- N: Total number of characters in the reference.

METRICS FOR EVALUATION

3.Accuracy,Precision,Recall,F1 Score:

- These are common metrics used in information retrieval tasks. In lip reading, precision measures the accuracy of correctly recognized words or phonemes, recall measures how many true words or phonemes were found, and the F1-score combines both to provide a balanced measure of performance.
- Accuracy:

  Accuracy = (Number of Correct Predictions) / (Total Number of Predictions)

- Precision:

  Precision = (True Positives) / (True Positives + False Positives)

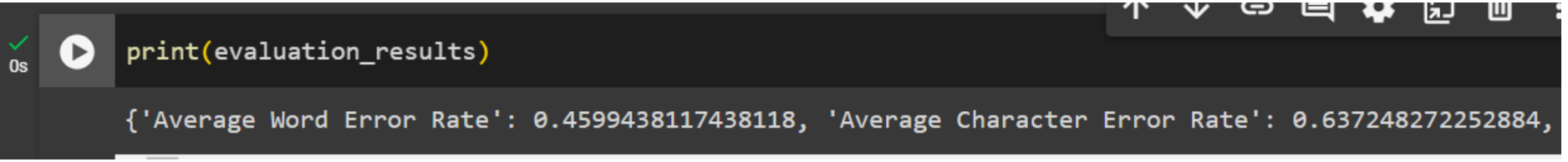
- Recall:

  Recall = (True Positives) / (True Positives + False Negatives)

- F1 Score:

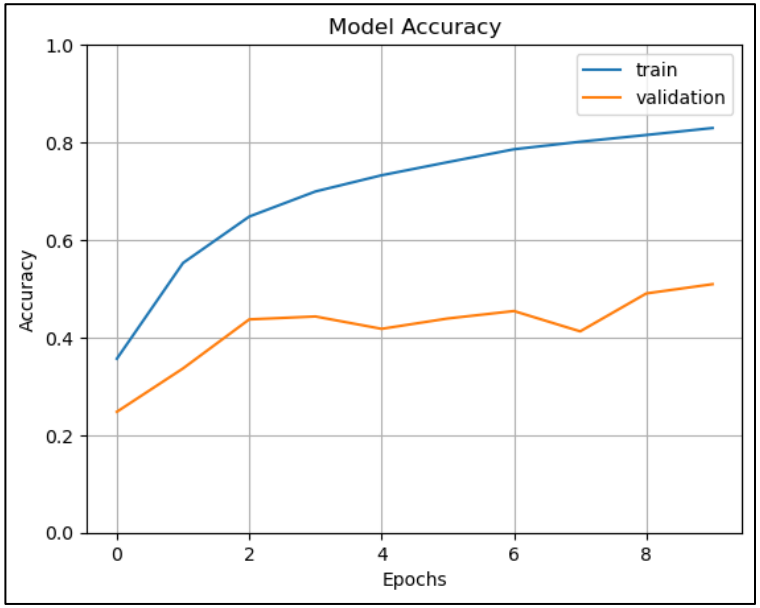
  F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

METRICS FOR EVALUATION



METRICS FOR EVALUATION

SENTIMENT ANALYSIS WITHOUT MEAN FACE

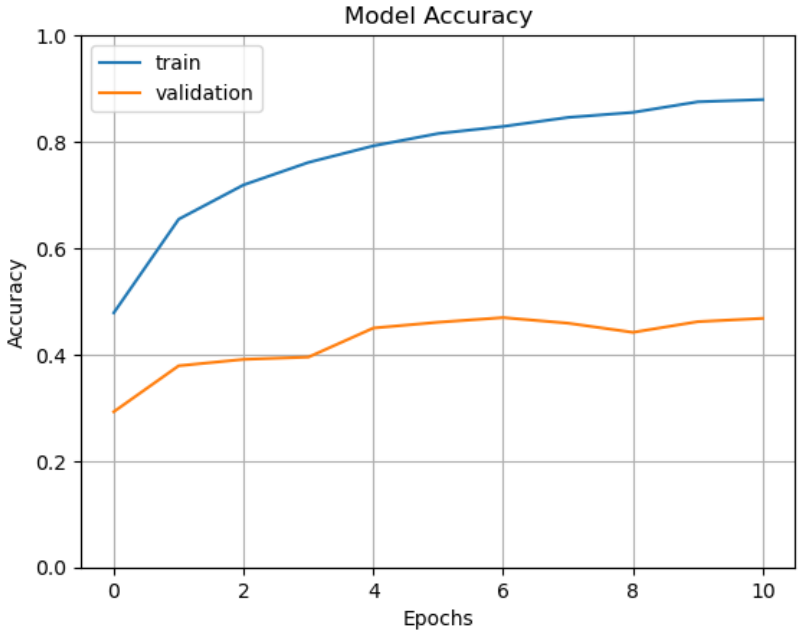


Graph of Accuracy vs Number of Epochs for sentiment analysis model without mean face

```
Epoch 10/10
362/362 [==============================] - 547s 2s/step - loss: 0.4848 - sparse_categorical_accuracy: 0.8294
```

METRICS FOR EVALUATION

SENTIMENT ANALYSIS WITH MEAN FACE



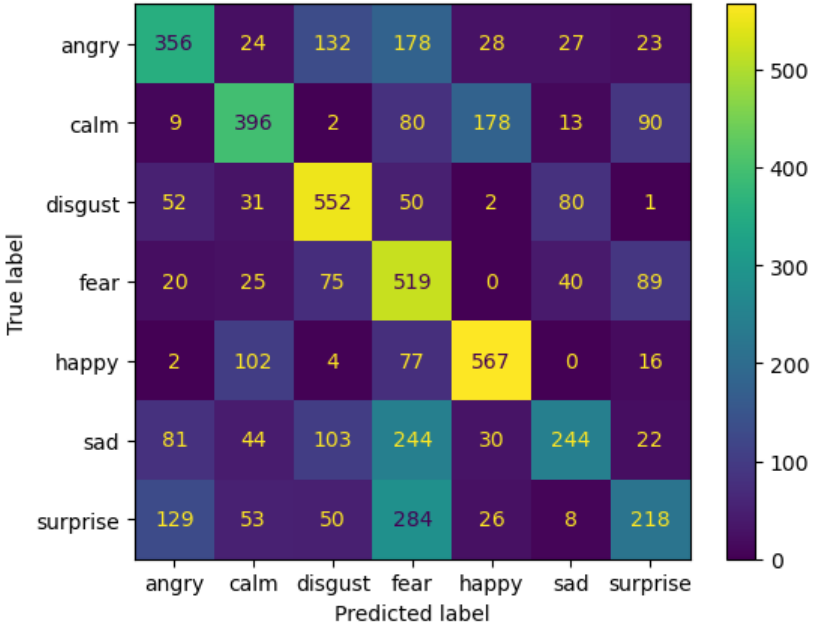
Graph of Accuracy vs Number of Epochs for sentiment analysis model with mean face

	precision	recall	f1-score	support
0	1.00	0.67	0.80	405
1	0.90	0.96	0.93	377
2	0.89	0.95	0.92	354
3	0.81	0.96	0.88	373
4	0.97	0.95	0.96	384
5	0.93	0.86	0.89	394
6	0.80	0.90	0.85	401
accuracy			0.89	2688
macro avg	0.90	0.89	0.89	2688
weighted avg	0.90	0.89	0.89	2688

Performance metrics for sentiment analysis model with mean face

METRICS FOR EVALUATION

SENTIMENT ANALYSIS MODEL TESTING WITHOUT MEAN FACE:



Confusion matrix for full video

```
In [115]:
pred_list = np.array_split(pred, num_videos)

ground = []
predic = []

i = 0
for count, video in enumerate(pred_list):
    predic.append(np.mean(video, axis=0).argmax())
    # predic.append(np.average(video, axis=0, weights=np.max(vide
    ground.append(filenames_test[count][0][1])

    if predic[count] == ground[count]:
        i += 1

print('accuracy full video:\t', round(i/len(pred_list), 4))
print('Num videos:\t\t', count+1)
```

```
accuracy full video:    0.1786
Num videos:             224
```

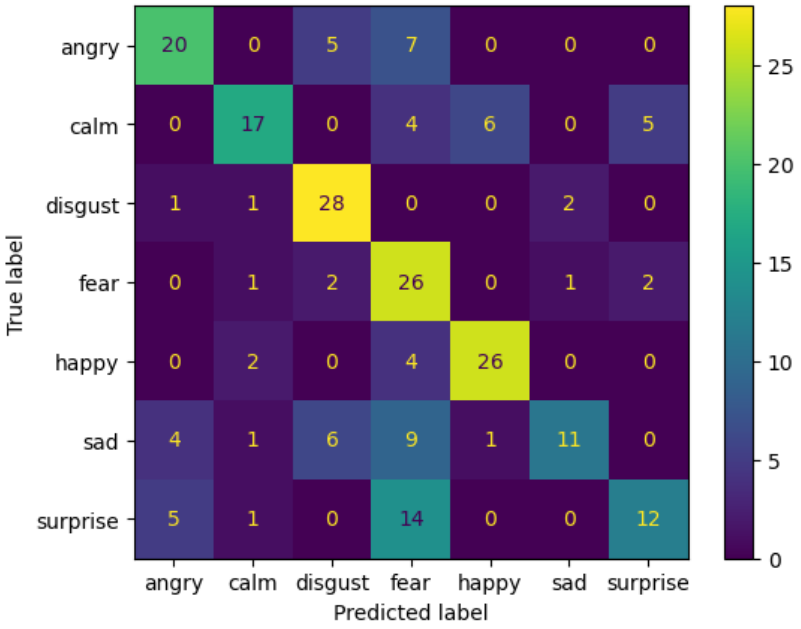
Accuracy for full video

	Recall	Precision
angry	0.46	0.55
calm	0.52	0.59
disgust	0.72	0.60
fear	0.68	0.36
happy	0.74	0.68
sad	0.32	0.59
surprise	0.28	0.47

Performance metrics for full video

METRICS FOR EVALUATION

SENTIMENT ANALYSIS MODEL TESTING WITH MEAN FACE



Confusion matrix for full video

```
In [130]:
pred_list = np.array_split(pred, num_videos)

ground = []
predic = []

i = 0
for count, video in enumerate(pred_list):
    predic.append(np.mean(video, axis=0).argmax())
    # predic.append(np.average(video, axis=0, weights=np.max(video, a
    ground.append(filenames_test[count][0][1])

    if predic[count] == ground[count]:
        i += 1

print('accuracy full video:\t', round(i/len(pred_list), 4))
print('Num videos:\t\t', count+1)
```

```
accuracy full video:    0.625
Num videos:             224
```

Accuracy for full video

	Recall	Precision
angry	0.62	0.67
calm	0.53	0.74
disgust	0.88	0.68
fear	0.81	0.41
happy	0.81	0.79
sad	0.34	0.79
surprise	0.38	0.63

Performance metrics for full video

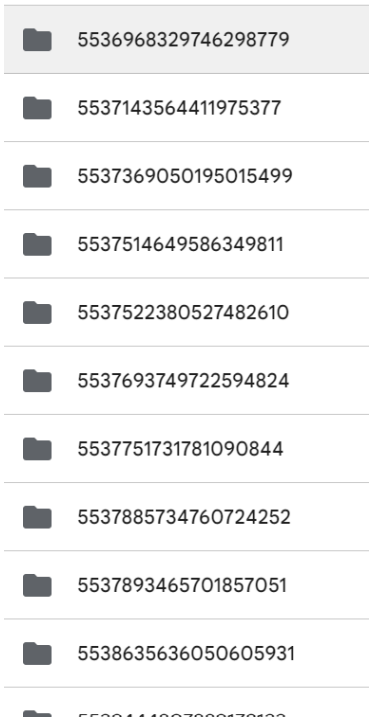
COMPARATIVE ANALYSIS

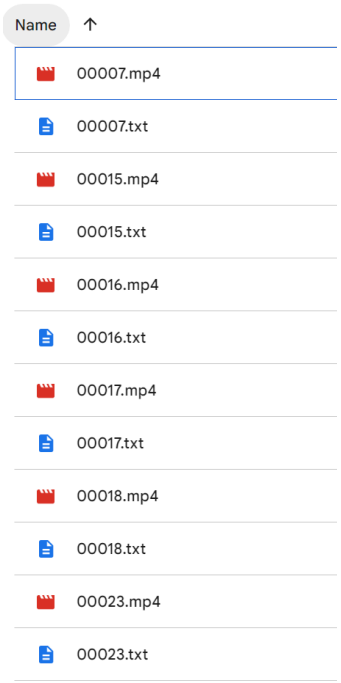
S.NO	MODEL	WER SCORE
1	WAS [9] (Pretrained)	70.4%
2	TM-seq2seq (Pretrained)	49.8%
3	TM-seq2seq+LM (Pretrained)	56.1%
4	TM+ResNet	47.6%
5	TCN	51.1%
6	TM+ResNet+Subword Vocabulary	46.4%
7	Proposed Model	45.9%

Comparative Analysis of existing models with the proposed model

TEST CASES

The original datasets are split into pretrain, train, validation, and test sets. The test sets contain videos unseen by the model during the training process, which will be tested to see if the transcripts are accurately generated.





TEST CASES

Sentiment Analysis Module:

Samples from RAVDESS Dataset:

index	emotion	emotional intensity	statement	repetition	actor
02-01-01-01-01-01-01	neutral	normal	1	1	1
02-01-01-01-01-02-01	neutral	normal	1	2	1
02-01-01-01-02-01-01	neutral	normal	2	1	1
02-01-01-01-02-02-01	neutral	normal	2	2	1
02-01-02-01-01-01-01	calm	normal	1	1	1
...	...	...	...	...	...
02-01-08-01-02-02-24	surprise	normal	2	2	24
02-01-08-02-01-01-24	surprise	strong	1	1	24
02-01-08-02-01-02-24	surprise	strong	1	2	24
02-01-08-02-02-01-24	surprise	strong	2	1	24
02-01-08-02-02-02-24	surprise	strong	2	2	24

1440 rows × 5 columns

TEST CASES

Sentiment Analysis Module:

Test Case ID	Test Description	Input	Actual Output	Expected Output	Passed/ Failed
TC_01	Video Frame	Video frame with a sad talking face	sad	sad	Passed
TC_02	Video Frame	Video frame with a talking angry face	angry	angry	Passed
TC_03	Video Frame	Video frame with a talking calm face	calm	calm	Passed
TC_04	Sequence of Video Frames	Videos frames capturing different angles of a surprised talking face	surprise	surprise	Passed
TC_05	Sequence of Video Frames	Videos frames capturing different angles of a fearful talking face	fear	fear	Passed
TC_06	Sequence of Video Frames	Videos frames capturing different angles of a happy talking face	happy	happy	Passed

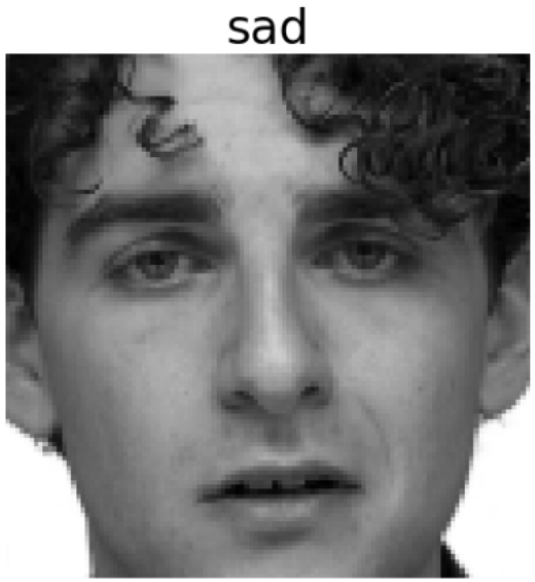
TEST CASES

Test Case 1:

In [79]:
```
examples = train_ds.unbatch().take(1)

for elem in examples:
    print(elem[0].shape)
plt.imshow(elem[0], cmap='gray')
plt.title(emotions[int(elem[1])], fontdict={'fontsize': 24})
plt.axis('off')
# plt.savefig('Plots/model4_input.png')
```

```
(112, 112, 1)
```

Out[79]: (-0.5, 111.5, 111.5, -0.5)



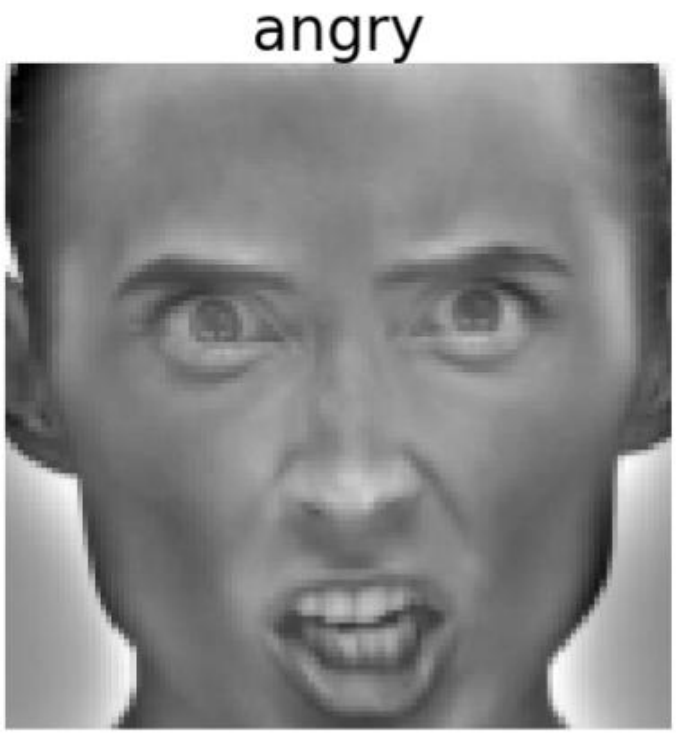
TEST CASES

Test Case 2:

In [23]:
```
examples = train_ds.unbatch().take(1)

for elem in examples:
    print(elem[0].shape)
plt.imshow(elem[0], cmap='gray')
plt.title(emotions[int(elem[1])], fontdict={'fontsize': 24})
plt.axis('off')
#plt.savefig('C:/Users/Sheeba/Downloads/Plots/model5_input.png')
```

```
(112, 112, 1)
```

Out[23]: (-0.5, 111.5, 111.5, -0.5)



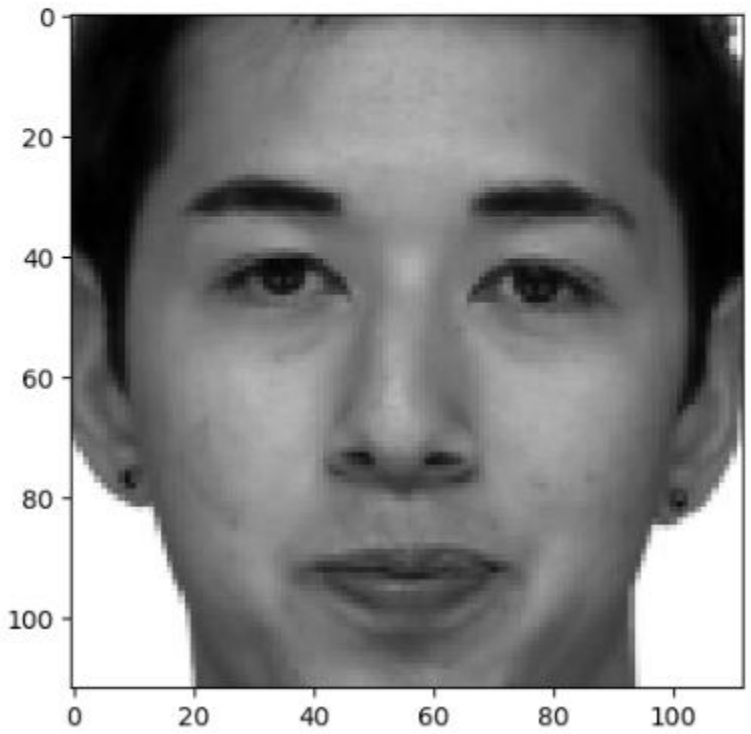
TEST CASES

Test Case 3:

In [122]:
```
b = test_ds.unbatch().take(1)

for elem in b:
    print(elem[0].shape)
    print(emotions[int(elem[1])])
plt.imshow(elem[0], cmap='gray')
```

```
(112, 112, 1)
calm
```

Out[122]: <matplotlib.image.AxesImage at 0x21fd1d4e050>

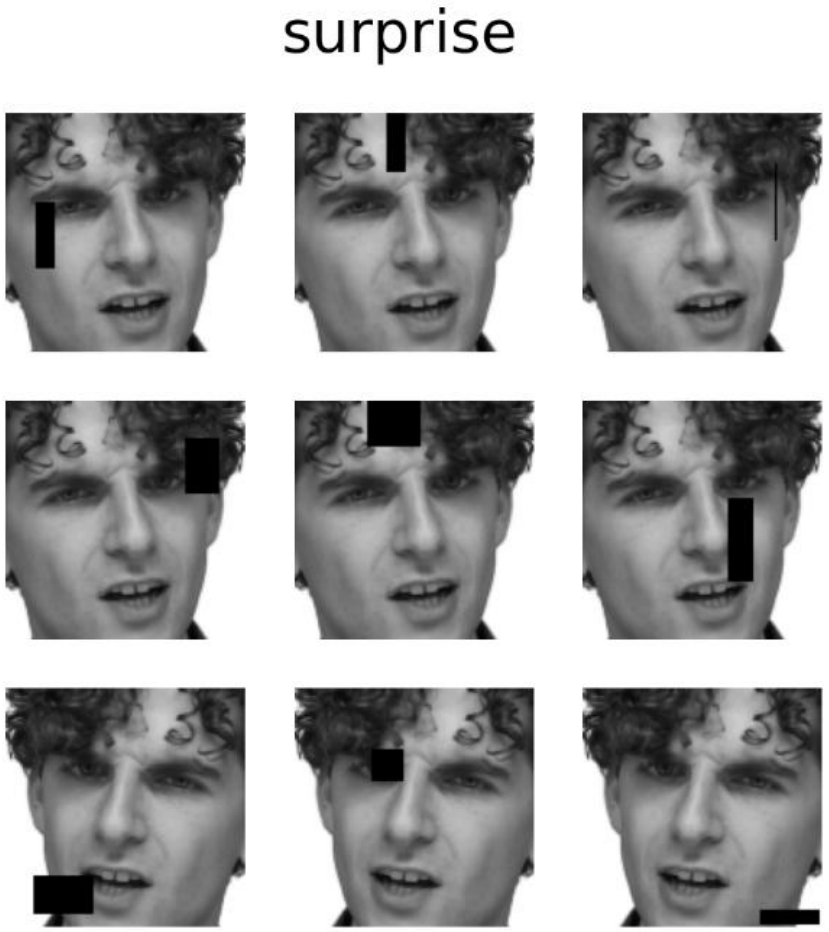


TEST CASES

Test Case 4:

In [80]:

```
for images, labels in train_ds.take(1):
    fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(10, 10))
    fig.suptitle(emotions[int(labels[0])], fontsize=40)
    first_image = images[0]
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        augmented_image = data_augmentation(
            tf.expand_dims(first_image, 0), training=True
        )
        plt.imshow(augmented_image[0], cmap='gray')
        plt.axis("off")
plt.savefig('C:/Users/Sheeba/Downloads/Plots/model4_augmentation.png')
```

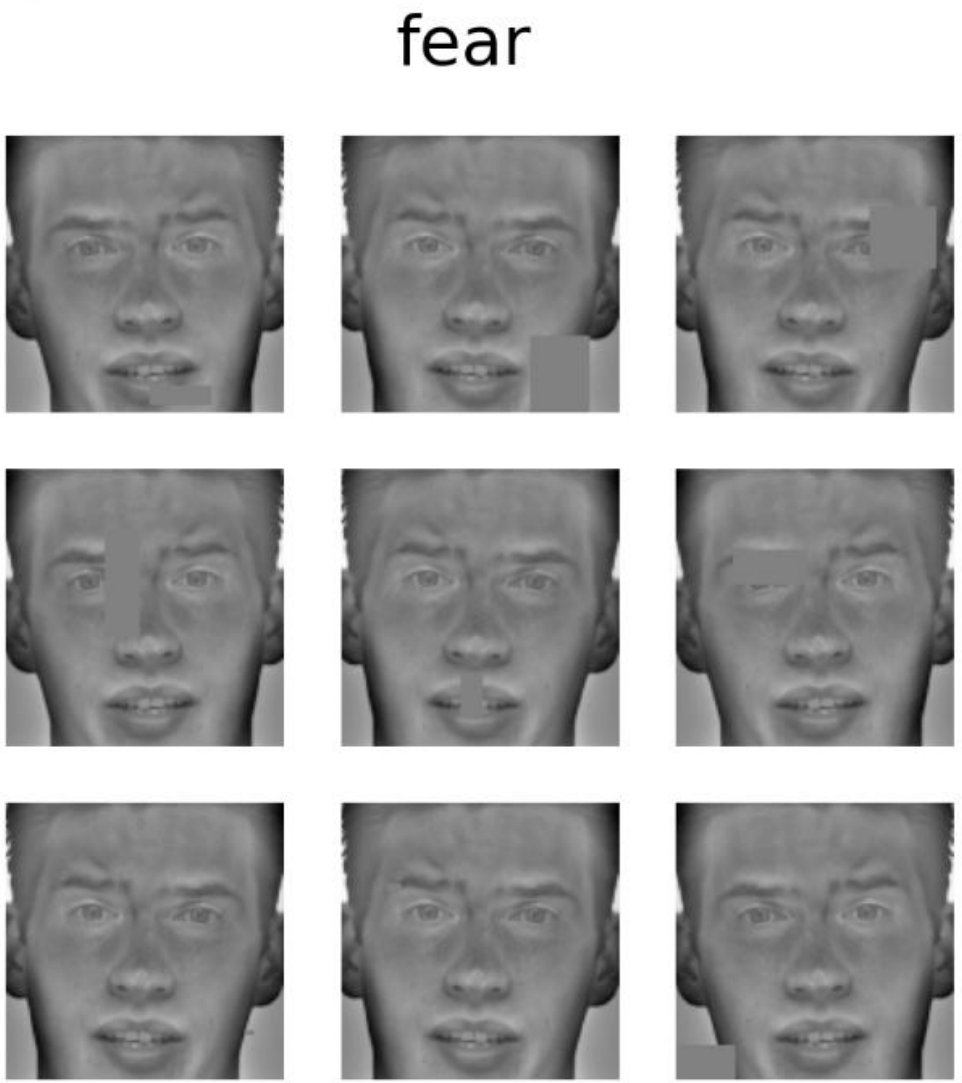


TEST CASES

Test Case 5:

In [27]:

```
for images, labels in train_ds.take(1):
    fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(10, 10))
    fig.suptitle(emotions[int(labels[0])], fontsize=40)
    first_image = images[0]
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        augmented_image = data_augmentation(
            tf.expand_dims(first_image, 0), training=True
        )
        plt.imshow(augmented_image[0], cmap='gray')
        plt.axis("off")

```

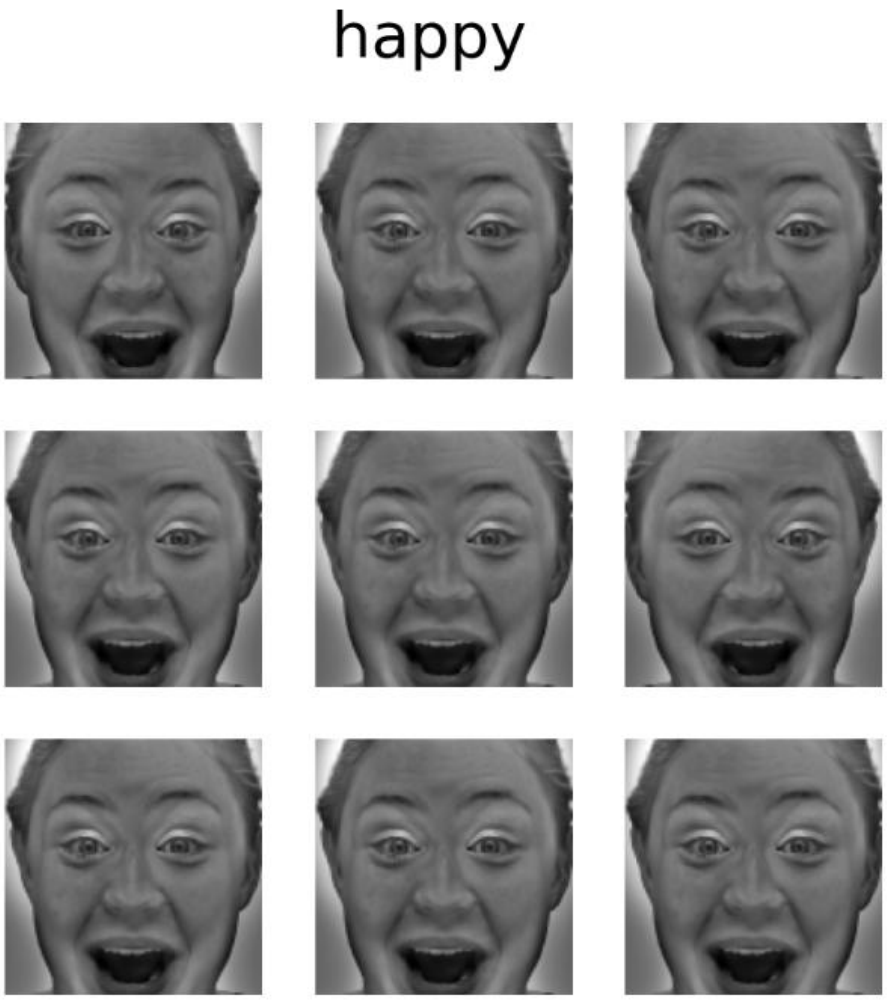


TEST CASES

Test Case 6:

In [97]:

```
for images, labels in train_ds.take(1):
    fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(10, 10))
    fig.suptitle(emotions[int(labels[0])], fontsize=40)
    first_image = images[0]
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        augmented_image = data_augmentation(
            tf.expand_dims(first_image, 0), training=True
        )
        plt.imshow(augmented_image[0], cmap='gray')
        plt.axis("off")
# plt.savefig('Plots/model5_augmentation.png')
```



TEST CASES

	Video Frames	Ground Truth	Transcript
0	[[[tensor([[1.6324, 1.6275, 1.6128, ..., ...	(I'M STILL HAPPY,)	WHAT TIME STILL
1	[[[tensor([[0.7248, 0.7248, 0.7248, ..., ...	(BUT BEFORE I DO,)	REMEMBER BEFORE I DO
2	[[[tensor([[-1.0390, -1.0293, -0.9999, ..., ...	(THERE'S NO OBVIOUS REASON,)	
3	[[[tensor([[1.7694, 1.7719, 1.7792, ..., ...	(YOU NEVER KNOW BY THE END OF THE BUILD,)	YOU NEVER KNOW BY THE END OF THE BUILD
4	[[[tensor([[0.6221, 0.6147, 0.5927, ..., ...	(FROM SMALL SHIPS,)	FROM SMALL SHIPS
5	[[[tensor([[-1.1559e-02, -1.4006e-02, -2.1345e...	(FOR THE FIRST TIME,)	SEARCH FOR THE FIRST
6	[[[tensor([[-0.4054, -0.3981, -0.3761, ..., -...	(IT WILL BE A GREAT INVESTMENT,)	IT WILL BE A GREAT INVESTMENT
7	[[[tensor([[[0.1597, 0.1597, 0.1597, ..., 1.63...	(PROBABLY AROUND 85,)	PROBABLY AROUND 8
8	[[[tensor([[-2.1179, -2.1179, -2.1179, ..., ...	(WE'VE GOT A GREAT RELATIONSHIP,)	WE'VE GOT A GREAT RELATIONSHIP
9	[[[tensor([[0.2796, 0.2869, 0.3089, ..., ...	(OVER THE COURSE OF HIS LIFE,)	WHO WERE THE CAUSE OF HIS
10	[[[tensor([[-0.4397, -0.4323, -0.4103, ..., ...	(WHO'S VERY MUCH INTO FAMILY HISTORY,)	WHO'S VERY MUCH INTO FAMILY HISTORY
11	[[[tensor([[-0.5082, -0.5277, -0.5865, ..., -...	(PRETTY ON THE OUTSIDE,)	PLAY PRETTY ON THE OUTSIDE

DATASET DESCRIPTION

- **LRW (Lip Reading in the Wild):** Contains 450,000 word-level utterances from BBC television recordings, each with a 1.16-second duration.
- **LRS2-BBC:** Real-world lip-reading dataset with 143,000 utterances from BBC programs, featuring diverse topics and a vocabulary of 41,000 words.
- **RAVDESS:** A comprehensive dataset containing 7356 recordings in English, featuring 24 professional actors and encompassing both speech and song variants. These recordings, categorized into 8 emotions (Neutral, Calm, Happy, Sad, Angry, Fearful, Disgust, and Surprised), have a duration of three seconds each and are available in three formats: audio only, video only (without sound), and video with sound. The filenames follow a 7-part numerical identifier structure.

DELIVERABLES TO BE DEMONSTRATED

I. Video Preprocessing:

- a. TFRecords of Mouth Area Frames: Encoded video feature vectors capturing speaker's mouth movements, essential for lip-reading analysis.
- b. Extracted Face Frames: Key snapshots of human expressions within videos, crucial for accurate sentiment analysis and emotion recognition.

II. Visual Module Pre-Training:

- a. Multi-motion-informed Context (MC): Dynamic fusion of lip-motion and linguistic context, bridging visual data with accurate transcript generation.
- b. Pre-Trained Visual Module: Feature extractor for lip-motion frames, converting raw data into meaningful visual representations for lip-reading.

III. Piece-Wise Pre-Training:

- a. Pre-Trained Decoder: A language model pre-trained to predict text sequences character by character, excelling in generating coherent and contextually appropriate linguistic content.
- b. Pre-Trained Visual Module: A computational model pre-trained to create meaningful visual representations from lip-motion frames, enhancing lip-reading accuracy.

DELIVERABLES TO BE DEMONSTRATED

IV. Model Integration:

- a. Fine-Tuned Lip-Reading Decoder: Transforms visual input into textual output after undergoing fine-tuning on lip-reading datasets.
- b. Optimized Lip-Reading Model: A fully trained model combining pre-trained visual module and fine-tuned decoder for improved accuracy in generating transcripts.

V. Sentiment Analysis:

- a. Trained Sentiment Analysis Model: A fully trained model for analyzing sentiments in short speech videos, leveraging concatenated visual features for understanding long-term spatio-temporal dynamics.

VI. Inference:

- a. Generated Texts: Model-generated transcripts of visual content, character by character, crucial for video captioning and sentiment analysis.
- b. Language Model Probability Scores: Scores indicating character likelihood for assessing fluency and context appropriateness.
- c. Sentiment Analysis Scores: Reflect the emotional tone of generated text in relation to visual content for sentiment analysis integration.

CONCLUSION AND FUTURE WORK

- **Decoder pretraining refines linguistic representations, while visual module pretraining captures essential visual cues from lip-reading videos.**

- **Integrating these components, along with gap bridging mechanisms, ensures accurate lip-reading. The model excels in transcribing speech and analyzing sentiment from visual cues.**

- **For the sentiment analysis part an accuracy of 89% was obtained and for the overall model with integration of results of piecewise pre-training and sentiment analysis a Word Error Rate (WER) of 45.9% which shows an improvement from previous models.**

- **Utilization of a larger character vocabulary to facilitate more accurate generation of words that would handle a larger set of unknown vocabulary. The use of 3D visual modules as the visual backend can be done.**

- **This adaptation may encourage the model to understand each frame in a window context and is beneficial to frames-to-character (multiple-to-one) mappings. The LPA model could benefit from it.**