emotionandbodylanguageanalysis

October 20, 2025

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

VEATIC Dataset: Video-based Emotion and Affect Tracking in Context

---

Dataset Overview

The VEATIC dataset is designed to analyze emotions and affect in video content with contextual awareness. Key points:

- Source: 124 video clips from Hollywood movies, documentaries, and home videos.
- Annotations: Continuous valence and arousal ratings per frame, focusing on the emotional state of the target character.
- Focus: Captures facial expressions, body language, and scene context influencing emotional states.
- Format: Video files with frame-level annotations available for research purposes.

What Are the Annotations?

The annotations are **numerical labels** that describe the emotional state of the target character in each frame:

- Valence: How pleasant or unpleasant the emotion is.
  - High valence → positive (e.g., happiness)
  - Low valence → negative (e.g., sadness, anger)
- Arousal: How intense or calm the emotion is.
  - High arousal → excitement, anger, fear
  - Low arousal → calm, relaxation, boredom

Why they are important: 1. Provide **ground truth labels** for training machine learning models.
2. Enable **continuous emotion prediction** rather than fixed categories.
3. Help models learn the relationship between **body language, facial expression, and context**.
4. Allow **quantitative evaluation** of models using metrics like MAE, CCC, or correlation.
5. Enable **visualization of emotional dynamics** over time for deeper insights.

Example of emotional trajectory:

Time (s)	Valence	Arousal	Description
0–5	+0.6	0.2	Calm, neutral mood
6–15	+0.3	0.5	Tension rising
16–25	−0.4	0.8	Anger or frustration
26–30	−0.2	0.6	Still negative, calming down

Project Objective

The main goals of this project are:

1. **Analyze emotions and body language** of characters in videos using the VEATIC dataset.
2. **Extract visual features** such as facial expressions, body movements, and contextual cues.
3. **Model emotional states** using machine learning or deep learning techniques.
4. **Predict continuous affect** (valence and arousal) for each frame.
5. **Provide insights** into how context and body language contribute to emotional expression for applications in:
   - Affective computing
   - Human-computer interaction
   - Behavioral analysis

Usage

This dataset is particularly suited for:

- **Emotion recognition** considering **body language** and **contextual cues**.
- **Training and evaluating models** on **continuous emotion prediction** tasks.
- Integrating with computer vision pipelines using frameworks such as **OpenCV**, **PyTorch**, or **TensorFlow**.

Exploring the dataset

[2]:
```
import os
import cv2
import pandas as pd
from collections import Counter

#  Correct base path
base_path = "/content/drive/MyDrive/Colab Notebooks/dataVideo"

print(" Folders in dataVideo:")
for folder in os.listdir(base_path):
    print(" -", folder)

# === 1 Explore video files ===
video_dir = os.path.join(base_path, "videos")
if os.path.exists(video_dir):
```

```
    video_files = [f for f in os.listdir(video_dir) if f.endswith(('.mp4', '.
↪avi', '.mov'))]
    print(f"\n Total videos found: {len(video_files)}")
    print(" Sample video files:", video_files[:5])

    # Display video properties
    print("\n Video details:")
    for vf in video_files[:3]:  # analyze first 3 videos
        path = os.path.join(video_dir, vf)
        cap = cv2.VideoCapture(path)
        if not cap.isOpened():
            print(f" Could not open {vf}")
            continue

        fps = cap.get(cv2.CAP_PROP_FPS)
        frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
        duration = frames / fps if fps > 0 else 0
        width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        print(f" - {vf}: {width}x{height}, {fps:.1f} FPS, {frames} frames,␣
↪{duration:.2f} sec")
        cap.release()

# === 2 Explore rating files ===
rating_dir = os.path.join(base_path, "rating_averaged")
if os.path.exists(rating_dir):
    rating_files = [f for f in os.listdir(rating_dir) if f.endswith(('.csv', '.
↪json', '.txt'))]
    print(f"\n Rating files found: {len(rating_files)}")
    print(" Sample annotation files:", rating_files[:5])

    # Try loading a sample CSV file
    sample_file = None
    for f in rating_files:
        if f.endswith(".csv"):
            sample_file = os.path.join(rating_dir, f)
            break

    if sample_file:
        print(f"\n Previewing annotations from: {os.path.
↪basename(sample_file)}")
        df = pd.read_csv(sample_file)
        print(" Columns:", df.columns.tolist())
        print(df.head())

        # Basic summary
        if 'valence' in df.columns and 'arousal' in df.columns:
```

```
            print("\n Annotation summary:")
            print(f"Valence range: {df['valence'].min():.2f} → {df['valence'].
 ↪max():.2f}")
            print(f"Arousal range: {df['arousal'].min():.2f} → {df['arousal'].
 ↪max():.2f}")
            print(f"Missing values: {df.isna().sum().to_dict()}")
        else:
            print(" Columns 'valence' and 'arousal' not found - check file␣
 ↪structure.")

# === 3 Dataset overview ===
print("\n Dataset summary:")
print(f"Videos folder: {video_dir}")
print(f"Ratings folder: {rating_dir}")
print(f"Number of videos: {len(video_files)}")
print(f"Number of rating files: {len(rating_files)}")
```

```
 Folders in dataVideo:
- rating_averaged
- videos

 Total videos found: 124
 Sample video files: ['52.mp4', '20.mp4', '54.mp4', '19.mp4', '8.mp4']

 Video details:
- 52.mp4: 854x480, 25.0 FPS, 264 frames, 10.56 sec
- 20.mp4: 854x460, 25.0 FPS, 453 frames, 18.12 sec
 Could not open 54.mp4

 Rating files found: 248
 Sample annotation files: ['118_valence.csv', '115_arousal.csv',
'102_valence.csv', '114_arousal.csv', '100_arousal.csv']

 Previewing annotations from: 118_valence.csv
 Columns: ['0', '-0.03124999999999988']
   0  -0.03124999999999988
0  1             -0.063655
1  2             -0.045956
2  3             -0.019395
3  4             -0.015738
4  5             -0.031466
 Columns 'valence' and 'arousal' not found - check file structure.

 Dataset summary:
Videos folder: /content/drive/MyDrive/Colab Notebooks/dataVideo/videos
Ratings folder: /content/drive/MyDrive/Colab Notebooks/dataVideo/rating_averaged
Number of videos: 124
```

```
Number of rating files: 248
```

Align Videos with Emotion Ratings

[3]:
```python
import re

# Match video files to corresponding rating files
matches = []
for vid in video_files:
    vid_id = re.findall(r'\d+', vid)[0]  # extract number from filename
    for rate in rating_files:
        if vid_id in rate:
            matches.append((vid, rate))
            break

print(f"  Matched {len(matches)} video-rating pairs.")
print("  Sample matches:", matches[:5])
```

```
 Matched 124 video-rating pairs.
 Sample matches: [('52.mp4', '52_arousal.csv'), ('20.mp4', '120_arousal.csv'),
('54.mp4', '54_arousal.csv'), ('19.mp4', '119_arousal.csv'), ('8.mp4',
'118_valence.csv')]
```

Extract Body Language Features

[4]:
```python
import os
import cv2
import pandas as pd
import numpy as np

# Pick one sample pair (video + rating)
sample_video, sample_rating = matches[0]
print(f"  Selected video: {sample_video}")
print(f"  Matched rating file: {sample_rating}")

video_path = os.path.join(video_dir, sample_video)
rating_path = os.path.join(rating_dir, sample_rating)

# Load the emotion annotation file
ratings = pd.read_csv(rating_path)
print("\n  Annotation columns:", ratings.columns.tolist())
print(ratings.head())

# Inspect video properties
cap = cv2.VideoCapture(video_path)
fps = cap.get(cv2.CAP_PROP_FPS)
total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
duration = total_frames / fps if fps > 0 else 0
```

```
print(f"\n  Video details - {sample_video}")
print(f"Frames: {total_frames}, FPS: {fps:.1f}, Duration: {duration:.2f} sec")

# Select 5 evenly spaced frames
frame_indices = np.linspace(0, total_frames - 1, 5, dtype=int)
print("  Frame indices to extract:", frame_indices)
cap.release()
```

```
  Selected video: 52.mp4
  Matched rating file: 52_arousal.csv

  Annotation columns: ['0', '0.012568215500490885']
    0  0.012568215500490885
0   1             -0.014127
1   2             -0.001268
2   3             -0.010029
3   4             -0.006445
4   5             -0.010460

  Video details - 52.mp4
Frames: 264, FPS: 25.0, Duration: 10.56 sec
  Frame indices to extract: [  0  65 131 197 263]
```

[7]:
```
!pip install mediapipe
```

```
Collecting mediapipe
  Downloading mediapipe-0.10.21-cp312-cp312-manylinux_2_28_x86_64.whl.metadata
(9.7 kB)
Requirement already satisfied: absl-py in /usr/local/lib/python3.12/dist-
packages (from mediapipe) (1.4.0)
Requirement already satisfied: attrs>=19.1.0 in /usr/local/lib/python3.12/dist-
packages (from mediapipe) (25.4.0)
Requirement already satisfied: flatbuffers>=2.0 in
/usr/local/lib/python3.12/dist-packages (from mediapipe) (25.9.23)
Requirement already satisfied: jax in /usr/local/lib/python3.12/dist-packages
(from mediapipe) (0.5.3)
Requirement already satisfied: jaxlib in /usr/local/lib/python3.12/dist-packages
(from mediapipe) (0.5.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-
packages (from mediapipe) (3.10.0)
Collecting numpy<2 (from mediapipe)
  Downloading
numpy-1.26.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(61 kB)
                           61.0/61.0 kB
3.1 MB/s eta 0:00:00
Requirement already satisfied: opencv-contrib-python in
/usr/local/lib/python3.12/dist-packages (from mediapipe) (4.12.0.88)
```

```
Collecting protobuf<5,>=4.25.3 (from mediapipe)
  Downloading protobuf-4.25.8-cp37-abi3-manylinux2014_x86_64.whl.metadata (541
bytes)
Collecting sounddevice>=0.4.4 (from mediapipe)
  Downloading sounddevice-0.5.3-py3-none-any.whl.metadata (1.6 kB)
Requirement already satisfied: sentencepiece in /usr/local/lib/python3.12/dist-
packages (from mediapipe) (0.2.1)
Requirement already satisfied: CFFI>=1.0 in /usr/local/lib/python3.12/dist-
packages (from sounddevice>=0.4.4->mediapipe) (2.0.0)
Requirement already satisfied: ml_dtypes>=0.4.0 in
/usr/local/lib/python3.12/dist-packages (from jax->mediapipe) (0.5.3)
Requirement already satisfied: opt_einsum in /usr/local/lib/python3.12/dist-
packages (from jax->mediapipe) (3.4.0)
Requirement already satisfied: scipy>=1.11.1 in /usr/local/lib/python3.12/dist-
packages (from jax->mediapipe) (1.16.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->mediapipe) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-
packages (from matplotlib->mediapipe) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->mediapipe) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->mediapipe) (1.4.9)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->mediapipe) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-
packages (from matplotlib->mediapipe) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->mediapipe) (3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->mediapipe)
(2.9.0.post0)
INFO: pip is looking at multiple versions of opencv-contrib-python to determine
which version is compatible with other requirements. This could take a while.
Collecting opencv-contrib-python (from mediapipe)
  Downloading opencv_contrib_python-4.11.0.86-cp37-abi3-manylinux_2_17_x86_64.ma
nylinux2014_x86_64.whl.metadata (20 kB)
Requirement already satisfied: pycparser in /usr/local/lib/python3.12/dist-
packages (from CFFI>=1.0->sounddevice>=0.4.4->mediapipe) (2.23)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-
packages (from python-dateutil>=2.7->matplotlib->mediapipe) (1.17.0)
Downloading mediapipe-0.10.21-cp312-cp312-manylinux_2_28_x86_64.whl (35.6 MB)
                         35.6/35.6 MB
31.5 MB/s eta 0:00:00
Downloading
numpy-1.26.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.0
MB)
                         18.0/18.0 MB
```

```
101.4 MB/s eta 0:00:00
Downloading protobuf-4.25.8-cp37-abi3-manylinux2014_x86_64.whl (294 kB)
                           294.9/294.9 kB
26.4 MB/s eta 0:00:00
Downloading sounddevice-0.5.3-py3-none-any.whl (32 kB)
Downloading opencv_contrib_python-4.11.0.86-cp37-abi3-manylinux_2_17_x86_64.many
linux2014_x86_64.whl (69.1 MB)
                           69.1/69.1 MB
11.6 MB/s eta 0:00:00
Installing collected packages: protobuf, numpy, sounddevice, opencv-
contrib-python, mediapipe
  Attempting uninstall: protobuf
    Found existing installation: protobuf 5.29.5
    Uninstalling protobuf-5.29.5:
      Successfully uninstalled protobuf-5.29.5
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
  Attempting uninstall: opencv-contrib-python
    Found existing installation: opencv-contrib-python 4.12.0.88
    Uninstalling opencv-contrib-python-4.12.0.88:
      Successfully uninstalled opencv-contrib-python-4.12.0.88
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
opencv-python-headless 4.12.0.88 requires numpy<2.3.0,>=2; python_version >=
"3.9", but you have numpy 1.26.4 which is incompatible.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.26.4 which is
incompatible.
ydf 0.13.0 requires protobuf<7.0.0,>=5.29.1, but you have protobuf 4.25.8 which
is incompatible.
opencv-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but
you have numpy 1.26.4 which is incompatible.
opentelemetry-proto 1.37.0 requires protobuf<7.0,>=5.0, but you have protobuf
4.25.8 which is incompatible.
grpcio-status 1.71.2 requires protobuf<6.0dev,>=5.26.1, but you have protobuf
4.25.8 which is incompatible.
Successfully installed mediapipe-0.10.21 numpy-1.26.4 opencv-contrib-
python-4.11.0.86 protobuf-4.25.8 sounddevice-0.5.3
```

Setup MediaPipe for Pose Detection

[4]:
```
import mediapipe as mp

mp_drawing = mp.solutions.drawing_utils
mp_pose = mp.solutions.pose
pose = mp_pose.Pose(static_image_mode=True)

def extract_pose(frame):
    """Detect pose landmarks in a given frame."""
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = pose.process(rgb)
    return results
```

[37]:
```
import pandas as pd
import os
import numpy as np

# === Debug: Let's see ALL files in rating_averaged ===
base_path = "/content/drive/MyDrive/Colab Notebooks/dataVideo"
ratings_path = f"{base_path}/rating_averaged"

print(" ALL files in rating_averaged folder:")
all_files = os.listdir(ratings_path)
print(f"Total files: {len(all_files)}")

# Separate valence and arousal files
valence_files = [f for f in all_files if 'valence' in f.lower()]
arousal_files = [f for f in all_files if 'arousal' in f.lower()]
other_files = [f for f in all_files if 'valence' not in f.lower() and 'arousal'␣
 ↪not in f.lower()]

print(f"\n Valence files: {len(valence_files)}")
print(f" Arousal files: {len(arousal_files)}")
print(f" Other files: {len(other_files)}")

# Show sample of each
print(f"\nSample valence files: {valence_files[:5]}")
print(f"Sample arousal files: {arousal_files[:5]}")
print(f"Sample other files: {other_files[:5]}")

# === Check if we have matching valence and arousal files ===
print(f"\n Checking for matching valence/arousal pairs:")

# Extract video IDs from arousal files we've been using
arousal_ids = set()
for file in arousal_files:
```

```
    vid_id = file.split('_')[0]
    arousal_ids.add(vid_id)

valence_ids = set()
for file in valence_files:
    vid_id = file.split('_')[0]
    valence_ids.add(vid_id)

print(f"Unique video IDs in arousal files: {len(arousal_ids)}")
print(f"Unique video IDs in valence files: {len(valence_ids)}")
print(f"Videos with both valence and arousal: {len(arousal_ids.
 ↪intersection(valence_ids))}")

# === Analyze BOTH valence and arousal files for the same video ===
if valence_files:
    # Check a video that should have both
    test_video_id = "106"  # Using the same video from your example

    valence_file = None
    arousal_file = None

    for file in valence_files:
        if file.startswith(test_video_id + '_') or file.
 ↪startswith(test_video_id + '.'):
            valence_file = file
            break

    for file in arousal_files:
        if file.startswith(test_video_id + '_') or file.
 ↪startswith(test_video_id + '.'):
            arousal_file = file
            break

    print(f"\n Analyzing video {test_video_id}:")
    print(f"Valence file: {valence_file}")
    print(f"Arousal file: {arousal_file}")

    if valence_file and arousal_file:
        # Load both files
        valence_path = os.path.join(ratings_path, valence_file)
        arousal_path = os.path.join(ratings_path, arousal_file)

        valence_df = pd.read_csv(valence_path)
        arousal_df = pd.read_csv(arousal_path)

        print(f"\n VALENCE data ({valence_file}):")
        print(f"Columns: {valence_df.columns.tolist()}")
```

```
        print(f"Shape: {valence_df.shape}")
        print("First 5 rows:")
        print(valence_df.head())

        print(f"\n AROUSAL data ({arousal_file}):")
        print(f"Columns: {arousal_df.columns.tolist()}")
        print(f"Shape: {arousal_df.shape}")
        print("First 5 rows:")
        print(arousal_df.head())

        # Check if they have same number of frames
        print(f"\n Data alignment:")
        print(f"Valence frames: {len(valence_df)}")
        print(f"Arousal frames: {len(arousal_df)}")
        print(f"Match: {len(valence_df) == len(arousal_df)}")

# === If no valence files found, show what we actually have ===
if not valence_files:
    print(f"\n NO VALENCE FILES FOUND! Only arousal data available.")
    print(f"Let me check the actual content structure of arousal files:")

    for file in arousal_files[:3]:  # Check first 3 arousal files
        file_path = os.path.join(ratings_path, file)
        df = pd.read_csv(file_path)
        print(f"\n File: {file}")
        print(f"Columns: {df.columns.tolist()}")
        print(f"First 3 rows:")
        for i in range(min(3, len(df))):
            print(f"  Row {i}: {df.iloc[i].values}")
        print(f"Data range: {df.iloc[:, 1].min():.3f} to {df.iloc[:, 1].max():.
 ↪3f}")
```

```
 ALL files in rating_averaged folder:
Total files: 248

 Valence files: 124
 Arousal files: 124
 Other files: 0

Sample valence files: ['118_valence.csv', '102_valence.csv', '121_valence.csv',
'110_valence.csv', '114_valence.csv']
Sample arousal files: ['115_arousal.csv', '114_arousal.csv', '100_arousal.csv',
'116_arousal.csv', '104_arousal.csv']
Sample other files: []

 Checking for matching valence/arousal pairs:
Unique video IDs in arousal files: 124
```

```
Unique video IDs in valence files: 124
Videos with both valence and arousal: 124

 Analyzing video 106:
Valence file: 106_valence.csv
Arousal file: 106_arousal.csv

 VALENCE data (106_valence.csv):
Columns: ['0', '-0.03125000000000004']
Shape: (1815, 2)
First 5 rows:
   0  -0.03125000000000004
0  1             -0.073266
1  2             -0.044258
2  3             -0.011404
3  4             -0.016019
4  5             -0.039558

 AROUSAL data (106_arousal.csv):
Columns: ['0', '-0.009374999999999984']
Shape: (1815, 2)
First 5 rows:
   0  -0.009374999999999984
0  1              -0.020624
1  2              -0.012869
2  3              -0.004028
3  4              -0.005262
4  5              -0.011622

 Data alignment:
Valence frames: 1815
Arousal frames: 1815
Match: True
```

Visualize 5 Frames with Pose + Emotion Overlays

[51]:
```python
import os
import re
import cv2
import pandas as pd
import mediapipe as mp
from google.colab.patches import cv2_imshow

# === Base paths ===
base_path = "/content/drive/MyDrive/Colab Notebooks/dataVideo"
videos_path = f"{base_path}/videos"
ratings_path = f"{base_path}/rating_averaged"
```

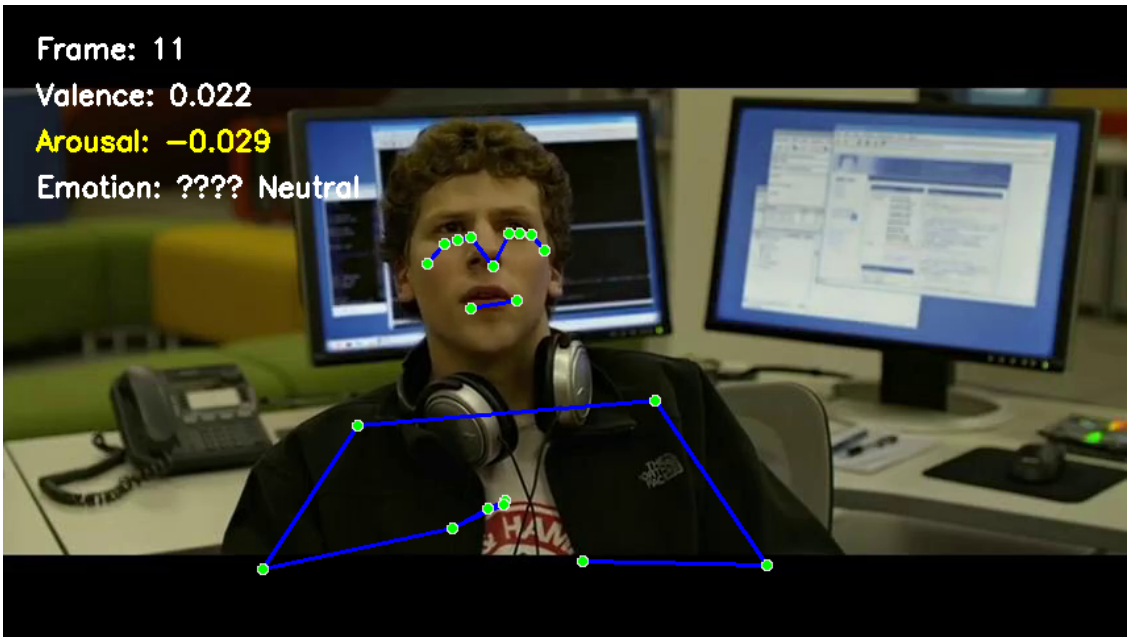
```
# === List all video and rating files ===
video_files = sorted([f for f in os.listdir(videos_path) if f.endswith(".mp4")])
valence_files = sorted([f for f in os.listdir(ratings_path) if 'valence' in f])
arousal_files = sorted([f for f in os.listdir(ratings_path) if 'arousal' in f])

print(f"  Found {len(video_files)} videos")
print(f"  Found {len(valence_files)} valence files")
print(f"  Found {len(arousal_files)} arousal files")

# === Match video files to corresponding valence and arousal files ===
matches = []
for vid in video_files:
    vid_id = re.findall(r'\d+', vid)[0]
    valence_file = next((v for v in valence_files if vid_id == v.
 ↪split('_')[0]), None)
    arousal_file = next((a for a in arousal_files if vid_id == a.
 ↪split('_')[0]), None)
    if valence_file and arousal_file:
        matches.append((vid, valence_file, arousal_file))

print(f"  Matched {len(matches)} video-valence-arousal triplets.")

# === Initialize Mediapipe Pose ===
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
pose = mp_pose.Pose(static_image_mode=False, min_detection_confidence=0.5)

# === Emotion mapping function (ranges for VEATIC dataset) ===
def get_emotion(valence, arousal):
    if valence > 0.1 and arousal > 0.1:
        return "  Happy/Excited", (0, 255, 0)
    elif valence > 0.1 and arousal <= 0.1:
        return "  Relaxed/Calm", (100, 255, 100)
    elif valence < -0.1 and arousal > 0.1:
        return "  Angry/Stressed", (0, 0, 255)
    elif valence < -0.1 and arousal <= 0.1:
        return "  Sad/Depressed", (255, 0, 0)
    elif -0.1 <= valence <= 0.1 and -0.1 <= arousal <= 0.1:
        return "  Neutral", (255, 255, 255)
    else:
        return "  Mixed", (255, 255, 0)

# === Select video 0 for testing ===
video_file, valence_file, arousal_file = "25.mp4", "25_valence.csv",␣
 ↪"25_arousal.csv"
video_path = os.path.join(videos_path, video_file)
valence_path = os.path.join(ratings_path, valence_file)
```

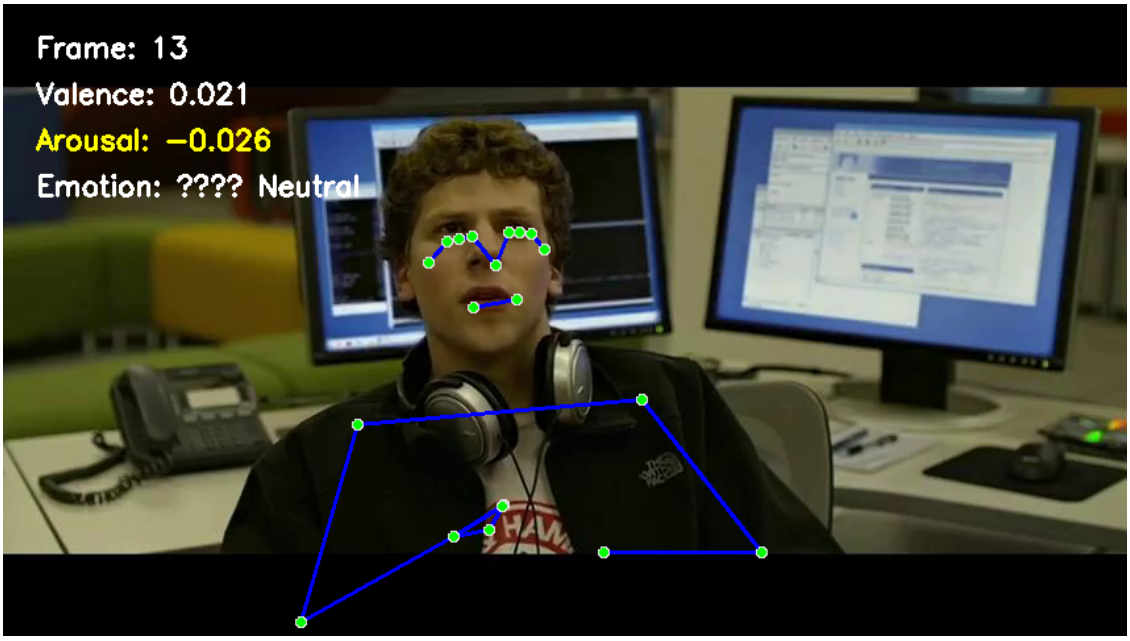
```
arousal_path = os.path.join(ratings_path, arousal_file)

print(f"\n Using video: {video_file}")
print(f" Using valence file: {valence_file}")
print(f" Using arousal file: {arousal_file}")

# === Load emotion data ===
valence_df = pd.read_csv(valence_path, header=None)
arousal_df = pd.read_csv(arousal_path, header=None)
valence_data = valence_df.iloc[:, 1]
arousal_data = arousal_df.iloc[:, 1]

ratings_clean = pd.DataFrame({'valence': valence_data, 'arousal': arousal_data})
print(f"\n Cleaned data shape: {ratings_clean.shape}")
print("First 10 rows of emotion data:")
print(ratings_clean.head(10))

# === Open video ===
cap = cv2.VideoCapture(video_path)
total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
print(f"\n Total frames in video: {total_frames}")
print(f" Total emotion ratings available: {len(ratings_clean)}")

# === Select frames with emotional variation ===
high_arousal_frames = ratings_clean.nlargest(3, 'arousal').index.tolist()
low_valence_frames = ratings_clean.nsmallest(3, 'valence').index.tolist()
high_valence_frames = ratings_clean.nlargest(2, 'valence').index.tolist()
frame_indices = sorted(set(high_arousal_frames + low_valence_frames +␣
 ↪high_valence_frames))[:5]
print(f" Selected frames with emotional variation: {frame_indices}")

# === Function to extract pose landmarks ===
def extract_pose(frame):
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    return pose.process(rgb)

# === Process video and display frames ===
frame_no = 0
frame_data = []

while True:
    ret, frame = cap.read()
    if not ret:
        break

    if frame_no in frame_indices:
        results = extract_pose(frame)
```

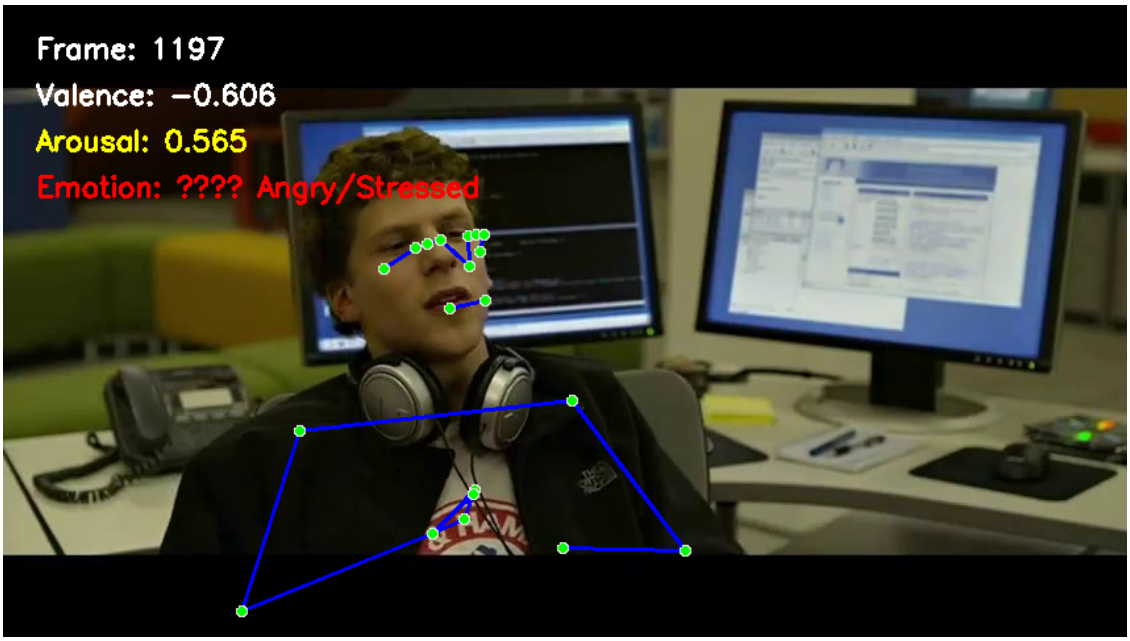
```
        if results.pose_landmarks:
            mp_drawing.draw_landmarks(
                frame, results.pose_landmarks, mp_pose.POSE_CONNECTIONS,
                mp_drawing.DrawingSpec(color=(0,255,0), thickness=2,␣
 ↪circle_radius=2),
                mp_drawing.DrawingSpec(color=(255,0,0), thickness=2,␣
 ↪circle_radius=2)
            )

        val = ratings_clean.loc[frame_no, 'valence']
        aro = ratings_clean.loc[frame_no, 'arousal']
        emotion, color = get_emotion(val, aro)

        # Overlay text
        y_offset = 40
        line_height = 35
        cv2.putText(frame, f"Frame: {frame_no}", (25, y_offset),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,255), 2)
        cv2.putText(frame, f"Valence: {val:.3f}", (25, y_offset + line_height),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,255), 2)
        cv2.putText(frame, f"Arousal: {aro:.3f}", (25, y_offset +␣
 ↪line_height*2),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,255), 2)
        cv2.putText(frame, f"Emotion: {emotion}", (25, y_offset +␣
 ↪line_height*3),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)

        # Display frame in Colab
        cv2_imshow(frame)
        cv2.waitKey(1000)  # Wait 1 second per frame

        frame_data.append({
            'frame_no': frame_no,
            'valence': val,
            'arousal': aro,
            'emotion': emotion
        })

    frame_no += 1

cap.release()

# === Print frame-level emotion analysis ===
print(f"\n REAL Emotion Analysis for {video_file}:")
print("=" * 70)
print("Based on ACTUAL dataset values from rating_averaged files")
print("-" * 70)
```

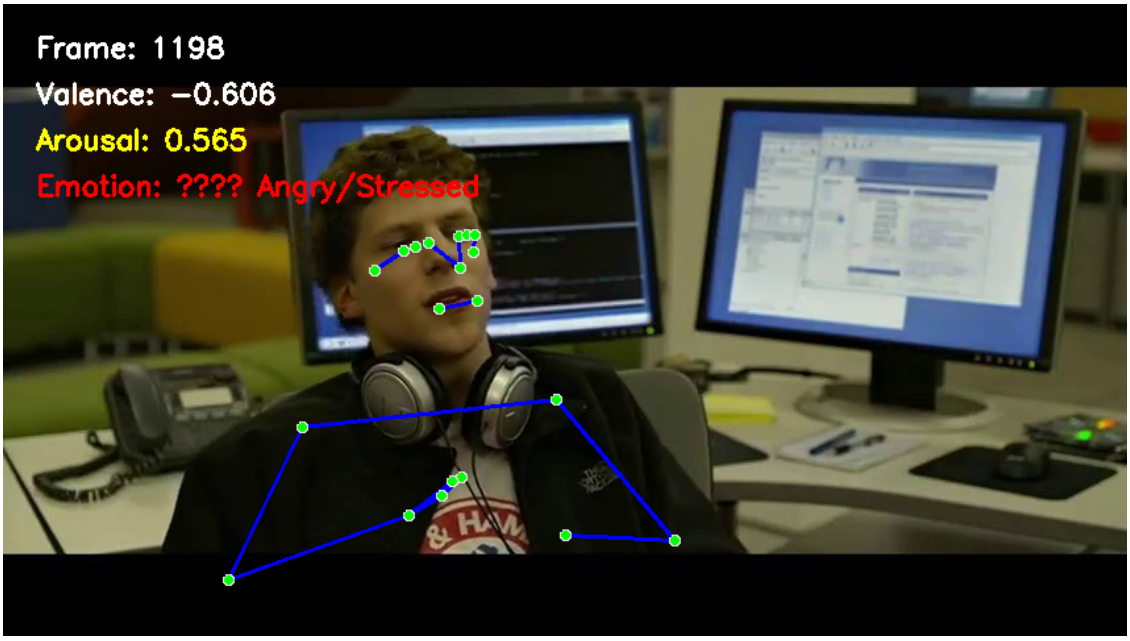
```
for data in frame_data:
    print(f"Frame {data['frame_no']:4d}: {data['emotion']:20} (V:␣
 ↪{data['valence']:7.3f}, A: {data['arousal']:7.3f})")
```

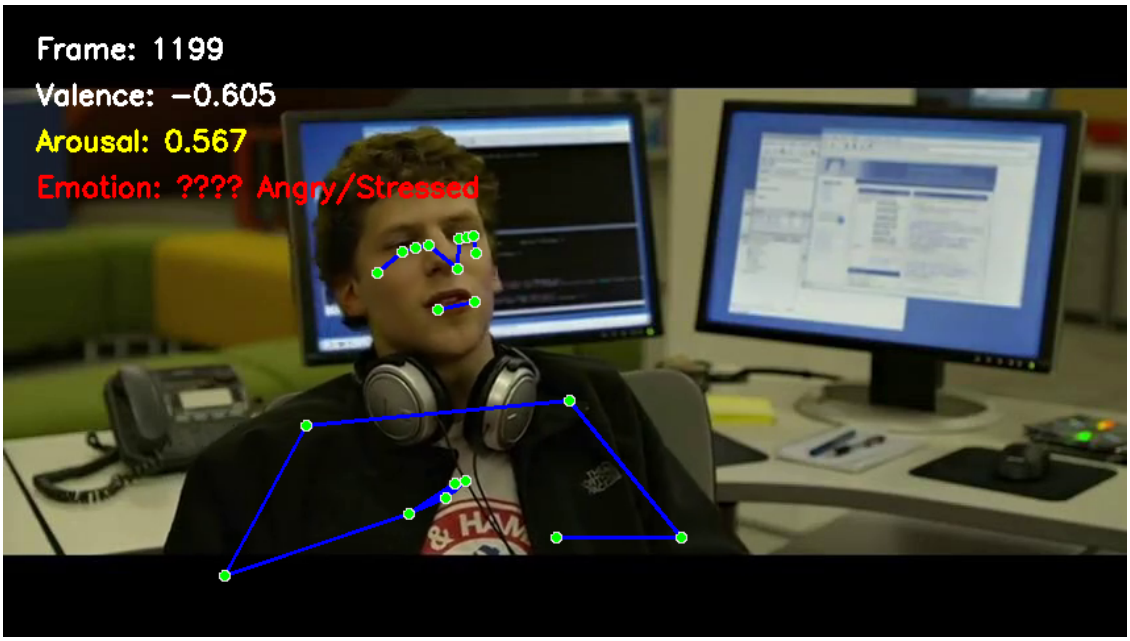
```
 Found 124 videos
 Found 124 valence files
 Found 124 arousal files
 Matched 124 video-valence-arousal triplets.

 Using video: 25.mp4
 Using valence file: 25_valence.csv
 Using arousal file: 25_arousal.csv

 Cleaned data shape: (2850, 2)
First 10 rows of emotion data:
    valence   arousal
0 -0.183397  0.057701
1 -0.037024 -0.051884
2 -0.046951 -0.013447
3 -0.007390 -0.024112
4 -0.028800 -0.007691
5 -0.013141 -0.018609
6 -0.024474 -0.013921
7 -0.000608 -0.031470
8  0.004643 -0.023587
9  0.016282 -0.031866

 Total frames in video: 2850
 Total emotion ratings available: 2850
 Selected frames with emotional variation: [11, 13, 1197, 1198, 1199]
```


Frame: 11
Valence: 0.022
Arousal: −0.029
Emotion: ???? Neutral


Frame: 13
Valence: 0.021
Arousal: −0.026
Emotion: ???? Neutral


Frame: 1197
Valence: −0.606
Arousal: 0.565
Emotion: ???? Angry/Stressed


Frame: 1198
Valence: −0.606
Arousal: 0.565
Emotion: ???? Angry/Stressed

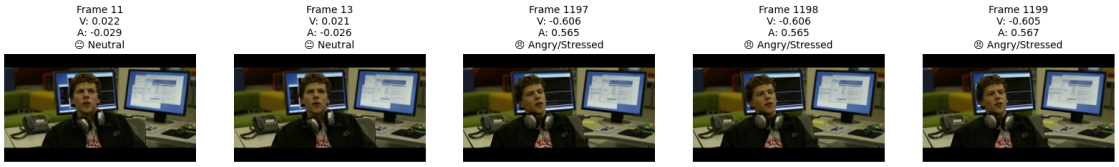


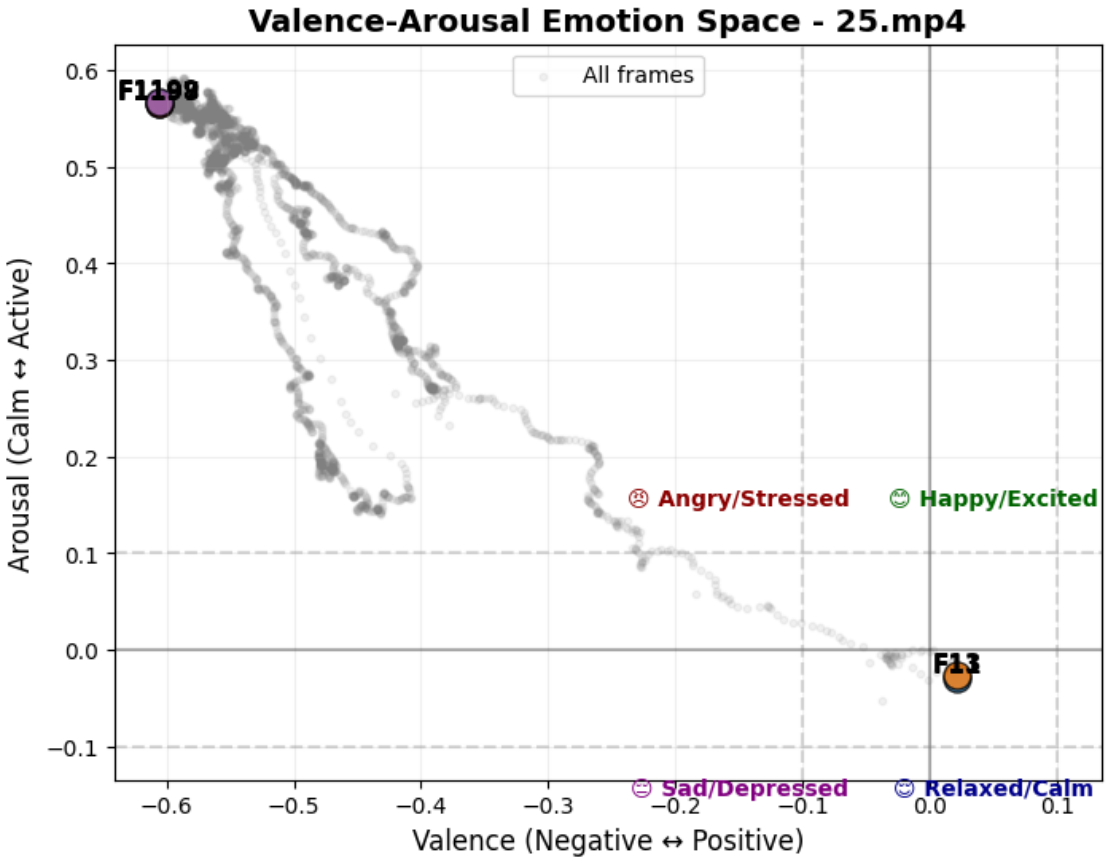

```
 REAL Emotion Analysis for 25.mp4:
======================================================================
Based on ACTUAL dataset values from rating_averaged files
----------------------------------------------------------------------
Frame   11:   Neutral            (V:   0.022, A:  -0.029)
Frame   13:   Neutral            (V:   0.021, A:  -0.026)
Frame 1197:   Angry/Stressed     (V:  -0.606, A:   0.565)
Frame 1198:   Angry/Stressed     (V:  -0.606, A:   0.565)
Frame 1199:   Angry/Stressed     (V:  -0.605, A:   0.567)
```

plots

[52]:
```
import matplotlib.pyplot as plt

# === Display selected frames with pose landmarks and emotion labels ===
plt.figure(figsize=(20, 6))

for i, data in enumerate(frame_data):
    frame_no = data['frame_no']
    # Reload the frame for plotting (to avoid cv2_imshow limitations in␣
 ↪matplotlib)
    cap = cv2.VideoCapture(video_path)
    cap.set(cv2.CAP_PROP_POS_FRAMES, frame_no)
    ret, frame = cap.read()
    if not ret:
        continue
```

```
    cap.release()
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    plt.subplot(1, len(frame_data), i + 1)
    plt.imshow(rgb_frame)
    plt.axis('off')
    plt.title(f"Frame {frame_no}\nV: {data['valence']:.3f}\nA: {data['arousal']:
 ↪.3f}\n{data['emotion']}", fontsize=10)

# === Valence-Arousal scatter plot ===
plt.figure(figsize=(8, 6))
plt.scatter(ratings_clean['valence'], ratings_clean['arousal'], alpha=0.1,␣
 ↪color='gray', s=10, label='All frames')

for data in frame_data:
    plt.scatter(data['valence'], data['arousal'], s=150, alpha=0.8,␣
 ↪edgecolors='black')
    plt.text(data['valence'], data['arousal'] + 0.005, f"F{data['frame_no']}",␣
 ↪fontsize=11, ha='center', fontweight='bold')

# Emotion quadrants
plt.text(0.05, 0.15, "  Happy/Excited", fontsize=10, ha='center',␣
 ↪color='darkgreen', fontweight='bold')
plt.text(-0.15, 0.15, "  Angry/Stressed", fontsize=10, ha='center',␣
 ↪color='darkred', fontweight='bold')
plt.text(0.05, -0.15, "  Relaxed/Calm", fontsize=10, ha='center',␣
 ↪color='darkblue', fontweight='bold')
plt.text(-0.15, -0.15, "  Sad/Depressed", fontsize=10, ha='center',␣
 ↪color='purple', fontweight='bold')

plt.axhline(y=0, color='black', linestyle='-', alpha=0.3)
plt.axvline(x=0, color='black', linestyle='-', alpha=0.3)
plt.axhline(y=0.1, color='gray', linestyle='--', alpha=0.3)
plt.axhline(y=-0.1, color='gray', linestyle='--', alpha=0.3)
plt.axvline(x=0.1, color='gray', linestyle='--', alpha=0.3)
plt.axvline(x=-0.1, color='gray', linestyle='--', alpha=0.3)

plt.xlabel('Valence (Negative  Positive)', fontsize=12)
plt.ylabel('Arousal (Calm  Active)', fontsize=12)
plt.title(f'Valence-Arousal Emotion Space - {video_file}', fontsize=14,␣
 ↪fontweight='bold')
plt.grid(True, alpha=0.2)
plt.legend()
plt.show()
```





1 Notes: Video-Emotion Analysis

This notebook uses the **matches list** you built (containing all 124 video-rating pairs) to randomly pick and analyze a single pair.

1.1 Steps:

1. **Select a Random Pair**
   - Picks one full pair (`video.mp4, rating.csv`) for analysis.
2. **Extract Random Frames**

- Extracts **5 random frames** from the selected video.

3. **Overlay Emotion Data**
   - Displays the **real valence/arousal data** on each extracted frame.

1.2 Example Visualization

.

Feature Extraction Function for All Videos

We'll extract pose landmarks (x, y, z, visibility) for each frame, then merge them with emotion annotations (valence/arousal).

[ ]:
```
import tqdm

def extract_features_from_video(video_path, rating_path, sample_rate=10):
    """
    Extract pose landmarks + emotion values from a single video.
    - sample_rate: process every Nth frame to speed up extraction
    """
    cap = cv2.VideoCapture(video_path)
    ratings = pd.read_csv(rating_path)
    fps = cap.get(cv2.CAP_PROP_FPS)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    features = []
    frame_idx = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        if frame_idx % sample_rate == 0:
            results = extract_pose(frame)
            if results.pose_landmarks:
                row = []
                for lm in results.pose_landmarks.landmark:
                    row.extend([lm.x, lm.y, lm.z, lm.visibility])

                # Add valence/arousal values
                if frame_idx < len(ratings):
                    val = ratings.loc[frame_idx, 'valence'] if 'valence' in␣
 ↪ratings else np.nan
                    aro = ratings.loc[frame_idx, 'arousal'] if 'arousal' in␣
 ↪ratings else np.nan
```

```
                else:
                    val, aro = np.nan, np.nan

                row.extend([val, aro])
                features.append(row)

        frame_idx += 1

    cap.release()

    # Create dataframe
    landmark_cols = [f"{axis}_{i}" for i in range(33) for axis in ['x', 'y',␣
 ↪'z', 'vis']]
    df = pd.DataFrame(features, columns=landmark_cols + ['valence', 'arousal'])
    return df
```

Process All Video–Rating Pairs

[ ]:
```
output_csv_dir = "/content/features_csv"
os.makedirs(output_csv_dir, exist_ok=True)

all_feature_paths = []

for vid, rate in tqdm.tqdm(matches, desc="Extracting all features"):
    video_path = os.path.join(video_dir, vid)
    rating_path = os.path.join(rating_dir, rate)

    try:
        df_features = extract_features_from_video(video_path, rating_path,␣
 ↪sample_rate=10)
        save_path = os.path.join(output_csv_dir, f"{os.path.
 ↪splitext(vid)[0]}_features.csv")
        df_features.to_csv(save_path, index=False)
        all_feature_paths.append(save_path)
    except Exception as e:
        print(f"  Error processing {vid}: {e}")

print(f"\n  Saved features for {len(all_feature_paths)} videos in␣
 ↪{output_csv_dir}")
print("  Sample saved file:", all_feature_paths[:3])
```

```
Extracting all features: 100%|      | 124/124 [45:41<00:00, 22.10s/it]


  Saved features for 124 videos in /content/features_csv
  Sample saved file: ['/content/features_csv/52_features.csv',
'/content/features_csv/20_features.csv',
'/content/features_csv/54_features.csv']
```

Combine All Features into One Dataset

Let's merge all the extracted CSVs into one global DataFrame to use later for model training or LLM/NLP reasoning

[ ]:
```
import glob

csv_files = glob.glob(os.path.join(output_csv_dir, "*.csv"))
print(f" Found {len(csv_files)} feature CSV files.")

all_data = []
for file in csv_files:
    df = pd.read_csv(file)
    df['video_id'] = os.path.basename(file).replace("_features.csv", "")
    all_data.append(df)

df_all = pd.concat(all_data, ignore_index=True)
print(" Combined dataset shape:", df_all.shape)
df_all.head()
```

```
 Found 124 feature CSV files.
 Combined dataset shape: (20838, 135)
```

[ ]:

	x_0	y_0	z_0	vis_0	x_1	y_1	z_1	\
0	0.408414	0.539313	0.007355	0.998324	0.409541	0.543411	0.003608	
1	0.645497	0.464137	-0.599259	0.997906	0.648463	0.450400	-0.614624	
2	0.645310	0.435042	-0.537506	0.999903	0.645936	0.427878	-0.541782	
3	0.631560	0.520894	-0.584044	0.999330	0.638934	0.518813	-0.595929	
4	0.626765	0.550526	-0.677637	0.954170	0.635234	0.533640	-0.700971	

	vis_1	x_2	y_2	…	y_31	z_31	vis_31	x_32	\
0	0.998358	0.409445	0.544313	…	0.342018	0.037960	0.913903	0.356942	
1	0.997515	0.651311	0.450205	…	0.509669	0.583108	0.098459	0.594292	
2	0.999936	0.647359	0.428312	…	0.631293	-0.230493	0.246536	0.636998	
3	0.999390	0.643239	0.520742	…	0.520506	0.545497	0.215943	0.579173	
4	0.975016	0.639835	0.531045	…	0.482068	0.855918	0.052336	0.604148	

	y_32	z_32	vis_32	valence	arousal	video_id
0	0.333808	0.059818	0.571860	NaN	NaN	55
1	0.498924	0.435351	0.174437	NaN	NaN	55
2	0.657710	-0.291722	0.477723	NaN	NaN	55
3	0.465584	0.516097	0.187936	NaN	NaN	55
4	0.564834	0.824694	0.068473	NaN	NaN	55

[5 rows x 135 columns]

Normalize and Label Emotions

We'll normalize valence/arousal and create categorical emotion labels (e.g. happy, calm, angry, sad).

[ ]:
```
from sklearn.preprocessing import MinMaxScaler

# Normalize valence and arousal between 0 and 1
scaler = MinMaxScaler()
df_all[['valence', 'arousal']] = scaler.fit_transform(df_all[['valence',␣
 ↪'arousal']])

# Map to emotion categories (based on valence-arousal model)
def get_emotion(val, aro):
    if val >= 0.5 and aro >= 0.5:
        return "happy/excited"
    elif val >= 0.5 and aro < 0.5:
        return "calm/content"
    elif val < 0.5 and aro >= 0.5:
        return "angry/fearful"
    else:
        return "sad/tired"

df_all['emotion_label'] = df_all.apply(lambda r: get_emotion(r['valence'],␣
 ↪r['arousal']), axis=1)

print(df_all['emotion_label'].value_counts())
df_all.head()
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/_array_api.py:776:
RuntimeWarning: All-NaN slice encountered
  return xp.asarray(numpy.nanmin(X, axis=axis))
/usr/local/lib/python3.12/dist-packages/sklearn/utils/_array_api.py:793:
RuntimeWarning: All-NaN slice encountered
  return xp.asarray(numpy.nanmax(X, axis=axis))
```

```
emotion_label
sad/tired    20838
Name: count, dtype: int64
```

[ ]:

	x_0	y_0	z_0	vis_0	x_1	y_1	z_1	\
0	0.408414	0.539313	0.007355	0.998324	0.409541	0.543411	0.003608	
1	0.645497	0.464137	-0.599259	0.997906	0.648463	0.450400	-0.614624	
2	0.645310	0.435042	-0.537506	0.999903	0.645936	0.427878	-0.541782	
3	0.631560	0.520894	-0.584044	0.999330	0.638934	0.518813	-0.595929	
4	0.626765	0.550526	-0.677637	0.954170	0.635234	0.533640	-0.700971	

	vis_1	x_2	y_2	…	z_31	vis_31	x_32	y_32	\
0	0.998358	0.409445	0.544313	…	0.037960	0.913903	0.356942	0.333808	
1	0.997515	0.651311	0.450205	…	0.583108	0.098459	0.594292	0.498924	

```
2  0.999936  0.647359  0.428312  … -0.230493  0.246536  0.636998  0.657710
3  0.999390  0.643239  0.520742  …  0.545497  0.215943  0.579173  0.465584
4  0.975016  0.639835  0.531045  …  0.855918  0.052336  0.604148  0.564834

       z_32    vis_32  valence  arousal  video_id  emotion_label
0  0.059818  0.571860      NaN      NaN        55      sad/tired
1  0.435351  0.174437      NaN      NaN        55      sad/tired
2 -0.291722  0.477723      NaN      NaN        55      sad/tired
3  0.516097  0.187936      NaN      NaN        55      sad/tired
4  0.824694  0.068473      NaN      NaN        55      sad/tired

[5 rows x 136 columns]
```

Cell 12 – Save Final Feature Dataset

```
[ ]: final_path = "/content/veatic_emotion_pose_dataset.csv"
     df_all.to_csv(final_path, index=False)
     print(f" Final dataset saved at: {final_path}")
```

```
Final dataset saved at: /content/veatic_emotion_pose_dataset.csv
```