Contents

Acknowledgements

These are the acknowledgements for reused code.

Facial Emotion Recognition: All the code needed for Facial Emotion Recognition (specifically, all the files in the directory "./feature_extraction/base_feature_extraction_utils/emotion_recognition" were taken from https://github.com/WuJi Expression-Recognition.Pytorch.

Directory Setup

In order to process and analyze videos, you must first move the videos, as well as the pictures of the people you want to analyze, in the appropriate directories.

1. Take a photo or screen shot of each person in the video you want to analyze. Forward facing photos are preferable. The images must be in PNG format.
2. Inside the folder "./data/video", create a folder sharing the same name as the video and move the video into said folder.
3. Inside the just created folder, create another folder called "img" and insert the images of the people you want to analyze into there.

An example of how the directory tree should look like after uploading a video is shown in Figure 1.

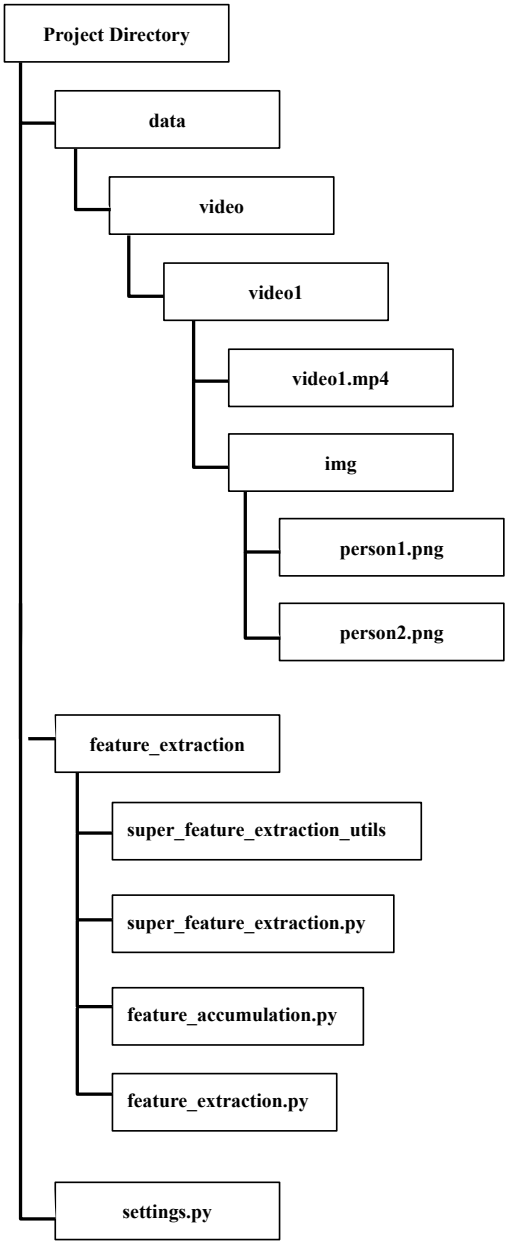

Figure 1: Truncated directory tree after adding the video called "video1".

Feature Extraction Process

The video extraction process is visualized in Figure 2. After initialization (which involves selecting which features you want to process), the base features are extracted from every frame. Then every n frames (i.e., window size of n), the derived features are extracted from the base features and recorded in a CSV file. When they click "Create CSVs for selected videos," it will show a progress message about time is left for completion.

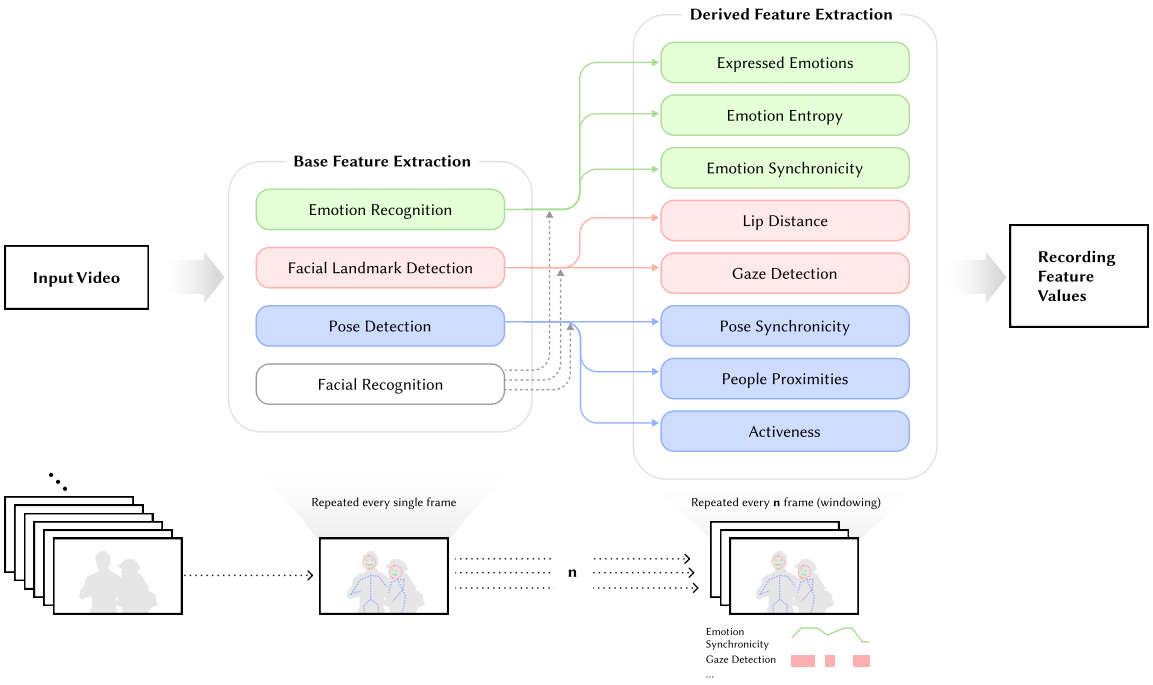

Figure 2: An overview of the feature extraction process.

Base Features

These are the features directly extracted from the video. They serve only to be used in calculating the derived features.

Facial Emotions: To extract facial emotions, we use a pretrained model (found here: https://github.com/WuJie1010/Facial-Expression-Recognition.Pytorch), which claimed to have achieved a 73.11% performance on the FER2013 dataset and a 94.64% on the CK+ dataset. This model classifies the displayed emotion into one of seven unit emotions (i.e., anger, disgust, fear, happiness, sadness, surprise, and neutral) and returns an integer ranging from 0 to 6, signifying the index of the recognized emotion.

In this software, the extracted emotions are stored in an $n \times 7$ array, where n is the number of frames and 7 refers to the number of emotions.

Input: A single video frame
Output: $n \times 7$ array

Facial Landmarks: To detect facial landmarks, the FaceMesh model from Google's MediaPipe library was used (https://developers.google.com/mediapipe/solutions/vision/face_landmarker). This model worked well during our preliminary evaluation and was reasonably lightweight, allowing for reduced processing times. This model detects 468 facial landmarks, each with their own x, y, and z coordinates.

In this software the facial landmarks of each person are stored in an $n \times 468 \times 3$ array, where n is the number of frames, 468 refers to the number of facial landmarks, and 3 refers to x, y, and z coordinates per landmark.

Input: A single video frame
Output: $n \times 468 \times 3$ array

Pose Key Points: To extract pose keypoints, Multipose Movenet from Tensorflow was used (https://www.kaggle.com/models/g
lightning/versions/1?tfhub-redirect=true). This model detects the pose keypoints of up to six people in a given

image. For each person, this model outputs 17 key points each with its own x and y coordinates, the bounding box of the detected person, and scores for each of the keypoints and the person as a whole.

In this software the key points of each person are stored in an $n \times 17 \times 2$ array; where n is the number of frames, 17 refers to the number of keypoints, and 2 refers to the coordinates per keypoint.

Input: A single video frame
Output: $n \times 17 \times 2$ array

Person Matching: In order to match the extracted data to the correct person, we opted to use face recognition; before extracting the features, each person's facial encoding would be extracted. Then, during the extraction, the encoding of the detected face would be compared to the existing catalog of encodings and the relevant information would be matched to the appropriate person. In order to do all of this, we opted to use the face-recognition library. This library claims to have achieved an accuracy of 99.38% on the Labeled Faces in the Wild benchmark.

Adding Custom Base Features

In order to add custom base features, you must do the following:

1. Create your base feature extraction function.
   (a) The extraction function should take the following form: "foo(img, num_people)".

   It must take the parameters *img* (which refers to the video frame) and *num_people* (which refers to the maximum number of people in the video.
   (b) The extraction function should return three values: *face_bounding_boxes*, *facial_encodings*, and *extracted_values*. *face_bounding_boxes* refers to the bounding boxes of the recognized faces. *facial_encodings* refers to the facial encodings of said recognized faces. *extracted_values* refers to the values extracted from the extraction function.

   Utilities for the bounding boxes and facial encodings already exist; you simply have to look at the other functions to see how to obtain them.
2. Add the extraction function to the file "./feature_extraction/base_feature_extraction_utils.py".
3. Add the name of the function to line 42 of the file "./settings.py" (this should be a list called "base_feature_functions")
4. Add any helper functions, models, etc. to the folder "./feature_extraction/base_feature_extraction_utils".
5. Add a numpy array to the *Person* class found in the file "./feature_extraction/feature_extraction_utils.py" corresponding to your feature.
6. Add the numpy array created in **step 5** to the list "self.extracted_features" on line 20 of the file "./feature_extraction/feature_extraction_utils.py" in the Person class.

Derived Features

These are the features which provide insight into the video and are calculated from the base features.

Expressed Emotions: This is simply a collection of all the emotions expressed by a person over a given amount of frames.

The expressed emotions for person i, denoted as TEE_i, is the vector containing all the emotions (7 basic emotions) displayed by person i over n frames. This is calculated by simply adding all the values for one emotion, denoted as e_{ij} where j is between 0 and 6 (representing the 7 basic emotions), over all the frames in the base extracted emotions for person i, denoted as BEE_i, for each basic expressed emotion:

$$TEE_i = [e_{i0}, e_{i1}, ..., e_{i6}]$$

$$e_{ij} = \sum_{k=0}^{n-1} BEE_i[k][j]$$

Emotion Entropy: Entropy is the measure of disorder or randomness. Hence emotion entropy is defined as the randomness of unit emotions.

The emotion entropy of person i over n frames, denoted by $EEnt_i$, is calculated by applying Shannon's entropy to a person's Total Expressed Emotions vector, TEE_i.

$$EEnt_i = -\sum_{j=0}^{6} p(TEE_i[j]) \cdot log_7(p(TEE_i[j]) + \varepsilon)$$

$$p(TEE_i[j]) = \frac{TEE_i[j]}{\sum_{k=0}^{6} TEE_i[k]}$$

where ε is an arbitrarily small value to avoid errors when encountering log(0).

Emotion Synchronicity: This measures the degree of similarity between two peoples' emotions during a given time frame.

The emotion synchronicity between person i and person j over n frames, denoted by $ESync_{ij}$, is calculated by taking the euclidean distance between the Total Expressed Emotions vectors of person i and person j, which are TEE_i and TEE_j respectively.

$$ESync_{ij} = \sqrt{\sum_{k=0}^{6} (TEE_i[k] - TEE_j[k])^2}$$

Lip Distance: This measures the average change in lip distances of a person over a period of time, which in turn serves to detect if a person is talking or not over the duration.

The average change in lip distance of person i over n frames, denoted by LD_i, is calculated by first taking the average distance between the facial landmarks in the upper and lower lips, denoted by UL_i and LL_i respectively, for each of the n frames. For clarity, let the number of landmarks in the set LL_i be denoted as m_U and the number of landmarks in the set UL_i be denoted as m_L. Let this average distance be denoted as $FrameLD_{im}$, where m is the frame number, ranging from 0 to $n-1$. Lastly, the average of the absolute value of the differences of the lip distances are taken.

$$LD_i = \frac{\sum_{j=0}^{n-2} |FrameLD_{ij} - FrameLD_{i(j+1)}|}{n-1}$$

$$FrameLD_{ij} = \frac{\sum_{k=0}^{m_U-1} LL_i[j][k]}{m_U} - \frac{\sum_{k=0}^{m_L-1} UL_i[j][k]}{m_L}$$

Gaze Detection: This is used to measure whether two people look at each other and also serves to determine whether people are interacting with each other.

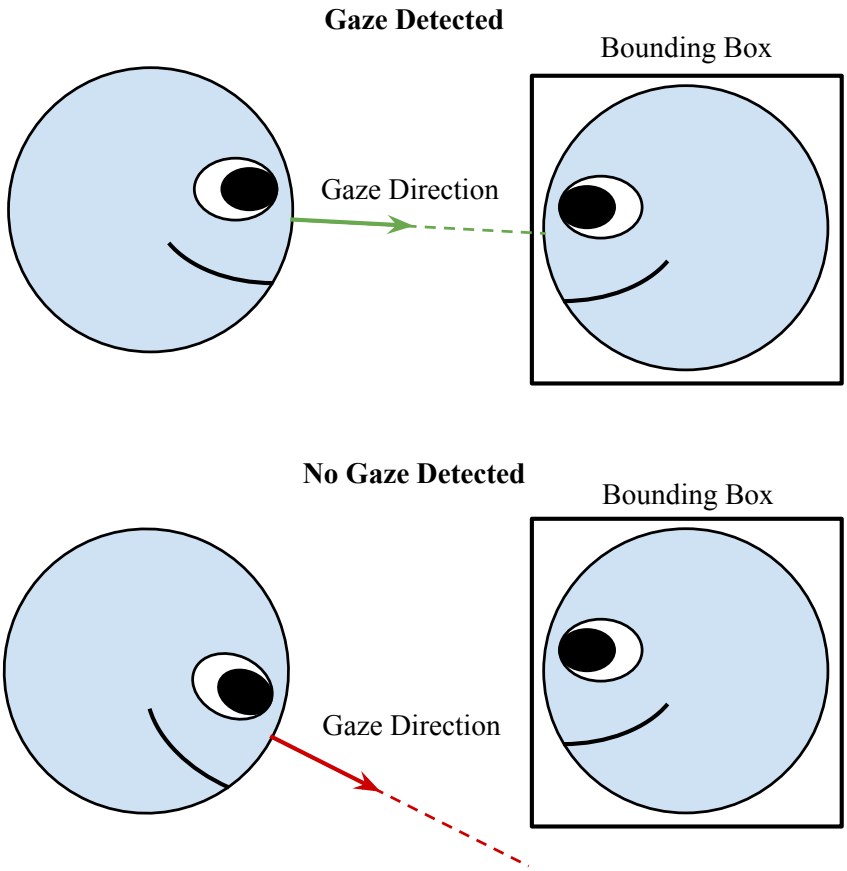

Figure 3: This figure displays what is counted as a detected gaze.

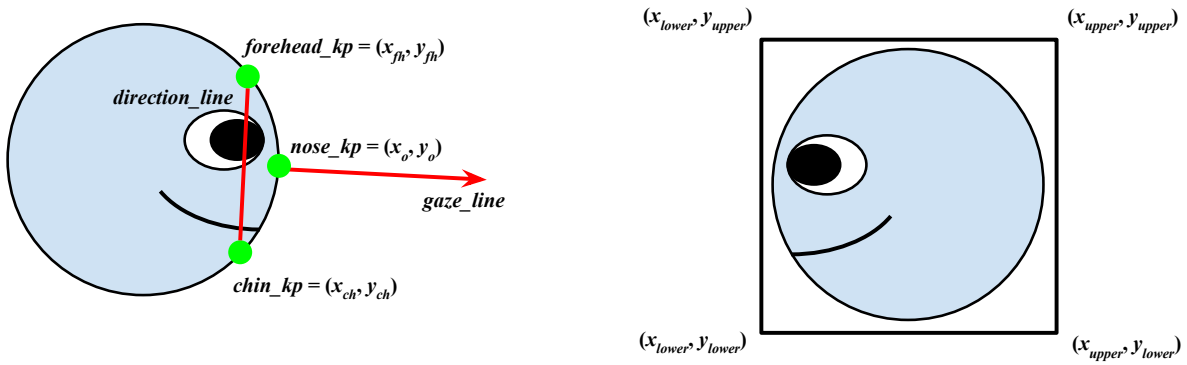

Figure 4: This figure displays the values used to calculate whether a social gaze is detected or not.

Social gaze is detected when the gaze of one person intersects the facial bounding box of another person, as detailed in Figure 3. The gaze direction of a person is determined as the line perpendicular to the line connecting the chin and forehead landmarks, as shown in Figure 4. For clarity, the chin and forehead landmarks are denoted as $chin\_kp$ and $forehead\_kp$, respectively, in the figure. This figure also defines several variables used to determine if a gaze is detected or not. Social gaze detection is done in the following steps. First we define $gaze\_line = m_{gaze}x + b$ for some arbitrary b, where $m_{gaze} = \frac{x_{ch} - x_{fh}}{y_{fh} - y_{ch} + \varepsilon}$. ε is 0 unless $y_{fh} - y_{ch}$ is 0, in which case it is an arbitrarily small value. ε must be variable in order to avoid division by 0; if ε is static, then division by 0 is possible since $y_{fh} - y_{ch}$ can be any real number. Next we project the gaze line of one person onto the closest vertical line originating from the other person's facial bounding box and calculate the y-coordinate of the projection; $y_{projected} = y_o + \Delta x \cdot m_{gaze}$. $\Delta x = x_{lower} - x_o$ if the subject is facing the right; else $\Delta x = x_{upper} - x_o$. If $x_o > \frac{x_{fh} + x_{ch}}{2}$, then the subject is

considered to be facing left; otherwise they are considered to be facing right. Finally, if $y_{lower} < y_{projected} < y_{upper}$, then a gaze is said to be detected; otherwise a gaze is not detected.

People Proximities: This is used to measure the distance between two people.

The proximity between person i and person j over n frames, denoted PP_ij, is calculated by taking the average distance between the 17 pose keypoints of person i and person j, denoted by PKP_i and PKP_j respectively, over the n frames.

$$PP_{ij} = \frac{\sum_{k_1=0}^{n-1} \sum_{k_2=0}^{16} \sqrt{(dif(k_1, k_2, 0)^2 + dif(k_1, k_2, 1)^2}}{n}$$

$$dif(k_1, k_2, l) = PKP_i[k_1][k_2][l] - PKP_j[k_1][k_2][l]$$

Pose Synchronicity: This feature measures the degree of similarity between the pose, or posture, of two people during social interactions.

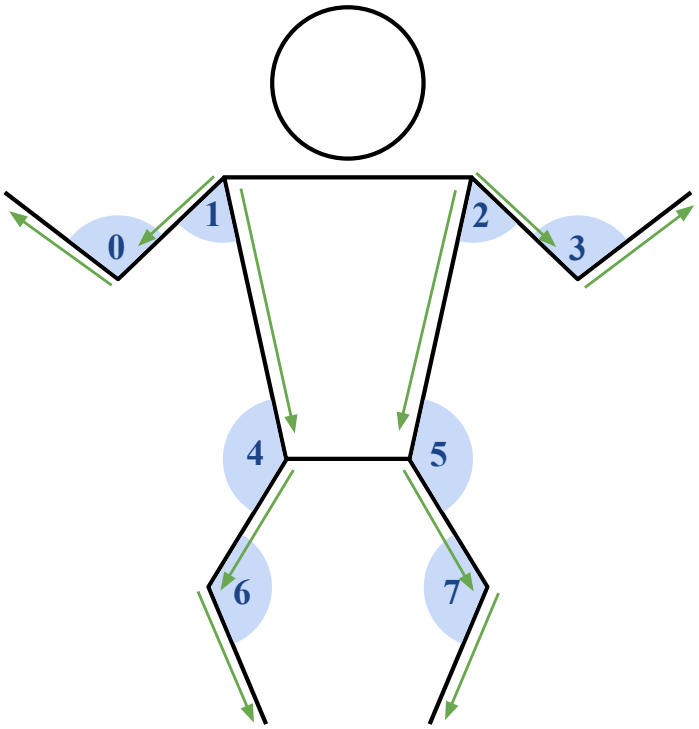

Figure 5: This figure displays the pose angles, as well as the connecting vectors used to form them, that are used to calculate the pose synchronicity.

The pose synchronicity between person i and person j over n frames, denoted by PS_ij, is calculated by taking the average euclidean distance between the 8 pose angles of person i and person j, denoted by PA_i and PA_j respectively. The pose angles are calculated by using the dot product of the 10 vectors connecting select keypoints of each person's pose skeleton, denoted by v_i and v_j for person i and person j respectively; the resultant angles are the only extractable angles using the given pose detection model. The pose angles and connecting vectors are detailed in Figure 5; the pose angles are in blue and the connecting vectors are in green.

$$PS_{ij} = \sqrt{\sum_{k=0}^{7} (PA_i[k] - PA_j[k])^2}$$

$$PA[j] = arccos(\frac{v_{j1} \cdot v_{j2}}{|v_{j1}||v_{j2}| + \varepsilon})$$

Here $j1$ and $j2$ are integers ranging from 0 to 9 indicating the specific vector used for a given angle calculation. ε is an arbitrarily small value to avoid division by 0.

Physical Activeness: This feature measures how physically active a person is, or how much their body moves around, in a given time period.

The activeness of person i over n frames, denoted by AC_i, is calculated by taking the average of the differences between the euclidean distances of the pose keypoints of person i, denoted by PKP_i between each frame.

$$AC_i = \frac{\sum_{j=0}^{n-2} \sum_{k=0}^{16} \sqrt{dif(j,k,0)^2 + dif(j,k,1)^2}}{n-1}$$

$$dif(j,k,l) = PKP_i[j][k][l] - PKP_i[j][k+1][l]$$

Adding Custom Derived Features

In order to add custom derived features, you must do the following:

1. Create your derived feature extraction function.
   (a) The extraction function should take the following form: "foo(collected_features, extra_features)".

   It must take the parameters *collected_features* ("self.extracted_features" from the Person class in "./feature_extraction/feature_extraction_utils.py". You must index the feature you want to use; for example since the expressed emotions are stored in the first element in "self.extracted_features", to access the emotion expressed you would use "collected_features[0]".) and *extra_features* (this simply refers to additional information collected in the Person class; currently only the bounding box information can be accessed so you can simply access this information with "extra_features[0]").

   (b) The extraction function should return one value: *extracted_values*. *extracted_values* refers to an array containing the extracted information from that frame for a given person.

   For functions that extract information involving two people, an array containing the interaction between the person you're extracting for and every other person is returned.

2. Add your extraction function to the file "./feature_extraction/feature_extraction.py".
3. Add a dictionary to the list called "features" on line 43 of the file "./settings.py".
   (a) This dictionary should include the feature name, the function, and the index of the associated base feature in the list "base_feature_functions" on line 42 of the same file.
4. Add an entry to the dictionary called "feature_processing_info" right below "features" in the file "./settings.py".
   (a) The key of this entry is the name of the feature given in the previous list; the value is a dictionary consisting of a boolean indicating if this is a pair feature or not (if the calculation of the feature requires data from 2 people), and a list of sub categories associated with the feature. If there are no subcategories, simply write None.

Video Overlays

These are overlays that can be applied to a video to analyze certain features in real time.

Person Identification: This draws bounding boxes around the faces of recognized people and displays their names as well.

Emotion Recognition: This displays the emotion displayed by a person under them.

Pose Detection: This draws the pose keypoints and pose skeleton of each person.

Adding Custom Video Overlays

In order to add custom video overlays, you must do the following:

1. Create your video overlay function.
   (a) The overlay function should take the following form: "foo(img)".

   It must take the parameter *img*, which refers to the video frame.
   (b) This function should return one output: *img*. *img* refers to the video frame after the particular overlay has been added on it.
2. Add the function to the file "./feature_visualization/video_overlays.py".
3. Add the name of the overlay to the list "overlay_names" on line 8 of "./feature_visualization/video_overlays.py".
4. Add the reference to the function to the list "overlay_functions" on the last line of "./feature_visualization/video_overlays.py"

Significant Moments

These are frames which are deemed significant according to some criteria.

Interaction Detected: This highlights moments in which an interaction occurs.

Adding Custom Significant Moments

In order to add custom significant moments, you must do the following:

1. Create your significant moment extraction function.
   (a) The extraction function should take the following form: "foo(collected_features, extra_features)".

   It must take the parameters *collected_features* ("self.extracted_features" from the Person class in "./feature_extraction/feature_extraction_utils.py". You must index the feature you want to use; for example since the expressed emotions are stored in the first element in "self.extracted_features", to access the emotion expressed you would use "collected_features[0]".) and *extra_features* (this simply refers to additional information collected in the Person class; currently only the bounding box information can be accessed so you can simply access this information with "extra_features[0]").
   (b) This function should return one output: *is_significant*. *is_significant* is either a 1 (if that set of frames is considered significant) or 0.
2. Add the function to the file "./feature_extraction/feature_extraction.py"
3. Add the name of the significant moment to the list "significant_moment_names" on line 82 of the file "./settings.py".
4. Add the reference to the function to the list "significant_moment_funcs" on line 83 of the file "./settings.py"