SAPIENZA
UNIVERSITÀ DI ROMA

Multimodal Interaction a.y. 23-24

Multimodal Advertisement Sentiment Analysis

Alessio Lucciola - lucciola.1823638@studenti.uniroma1.it
Danilo Corsi - corsi.1742375@studenti.uniroma1.it
Domiziano Scarcelli - scarcelli.1872664@studenti.uniroma1.it

June 19, 2024

Contents

SAPIENZA
UNIVERSITÀ DI ROMA

1 Preface

1.1 Introduction

Emotion recognition is the process of identifying human emotions based on facial expressions, gestures, speech, and other physiological signals. Rapid advancements in artificial intelligence, machine learning, and computer vision have greatly improved the accuracy and effectiveness of emotion recognition systems. In fields such as human-computer interaction and user experience design, it plays a crucial role in creating more intuitive and responsive interfaces. In fact, by understanding users' emotional states, systems can adapt their behavior and responses to meet their needs and preferences better, leading to more satisfying and engaging experiences.
Emotion recognition technology is being increasingly applied across various industries and domains. From customer service and marketing to healthcare and education, organizations recognize the value of understanding and responding to human emotions. This has led to a surge in research and development efforts focused on enhancing emotion recognition capabilities and integrating them into diverse applications and services. This project aims to create an emotion recognition system to understand how people react to video advertisements. Nowadays, many platforms such as YouTube or Netflix offer free content or a discount on the monthly subscription as long as the user watches ads from time to time. Advertisements are usually shown suddenly without any warning, and this may stimulate a reaction. We want to understand if it is possible to capture the user's emotions during this process, and this operation may be useful for understanding when to interrupt the video or what are the advertisements to be shown based on the user's reaction.

1.2 Team members and responsibilities

Alessio Lucciola : Development of the model for predicting emotions from audio signals. Preprocessing of the audio signals before applying the fusion. Implemented the algorithm for fusing the audio and video signals. Creation of the demo web application and tests regarding the fusion process. Report writing.

Danilo Corsi: Development of the model for facial expression recognition from video frames and implemented the pre-processing part of the video component before applying fusion. Carried out tests regarding the fusion process and report writing.

Domiziano Scarcelli: Development of the model for emotion recognition using the PPG signal; extension of the `rPPG-toolbox` to be compatible with custom data; pre-processing part of the video component before applying fusion. Report writing.

2 Background

2.1 A focus on emotion recognition

Before explaining how we solved the problem, we present some works that deal with multimodal emotion recognition:

- Ai et al.[1] (2024) presents a promising approach to multimodal emotion recognition by combining a two-stage classification process with graph contrastive learning. It solves the problem in two stages. Stage 1 focuses on classifying emotions into broad categories like positive, negative, or neutral. Stage 2 drills down into finer-grained emotion recognition based on the output from stage 1. This staged approach allows the model to progressively focus on different levels of emotional detail. Contrastive learning leverages graphs to represent the relationships between data samples from different modalities and employs contrastive learning techniques to continuously improve the model by emphasizing similarities between similar samples and pushing apart dissimilar ones.
- Caridakis et al.[2] (2007) dives into a multimodal approach for recognizing emotions by integrating information from facial expressions, body language, and speech. It explores using a Bayesian classifier for emotion recognition on a multimodal dataset.
- Lian et al.[3] (2023) tries to solve multimodal emotion recognition using deep learning techniques. It focuses specifically on three key modalities for understanding emotions: speech, text, and facial expressions. The paper analyzes how deep learning models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are particularly well-suited for analyzing complex multimodal data. It can be applied to extract emotional cues from each modality:
  - Speech: They analyze vocal characteristics like pitch, intonation, and energy to understand emotions.
  - Text: They examine word choices, sentence structure, and sentiment analysis to infer emotions from written content.
  - Face: They use facial recognition techniques to identify facial expressions associated with emotions.

  They use techniques for combining the information extracted from each modality to improve overall emotion recognition accuracy, and this involves early fusion (combining raw data) or late fusion (combining extracted features).
- Ahmed et al.[4] (2022) goes beyond the typical focus on speech, text, and facial expressions in MER. It explores how other modalities like physiological signals (heart rate, skin conductance) and body language can also contribute to emotion recognition. The paper discusses how various learning algorithms, including traditional machine learning (e.g. Support Vector Machines) and deep learning approaches (e.g. Convolutional Neural Networks), are used for feature extraction and classification in MER tasks. Interestingly, the survey explores how the COVID-19 pandemic

has influenced MER research.

- Shukla et al.[5] (2022) focuses on building a dataset of video advertisements that evoke specific emotions in viewers. It then explores using different techniques, including convolutional neural networks (CNNs) and electroencephalogram (EEG) signals, to recognize the emotions elicited by the ads.
- Bilotti et al. [14] (2023) leverages CNNs for feature extraction from facial expressions (video frames) and speech (Mel-spectrograms). The authors compare different strategies for combining information from the extracted features:
  - Early Fusion: Combines raw data from different modalities before feeding it into the CNN.
  - Late Fusion: Combines features extracted from each modality by the CNNs before the final classification stage.
  - Multi-stream CNNs: Uses separate CNNs for each modality, then combines the outputs at a later stage.

  The paper evaluates the performance of these strategies on two benchmark emotion recognition datasets, BAUM-1 and RAVDESS.
- Lee et al. [23] (2019) used a Convolutional Neural Network to classify high and low valence from a photoplethysmograph signal using the DEAP [17] dataset, focusing on good pre-processing techniques to remove as much noise as possible, and by segmenting it into windows that contained only a *single pulse*.
- Kang et al. [24] (2022) took the idea introduced in the Lee et al. paper and enhanced it by using a 1D convolutional autoencoder architecture, including both PPG and GSR (galvanic skin response) for high/low valence and arousal classification using both the DEAP dataset and the Asian MERTI-Apps dataset.

We took some ideas from previous works and tried to implement a solution to understand how users behave when watching an advertisement. We eventually decided to consider the **tone of the voice**, **facial expressions** and **photoplethysmogram** whose features are used for training deep learning models. Predictions are then fused together to build a robust emotion recognition system. The combination of modalities we used has never been tried before to try to solve the emotion recognizer for advertisements. Therefore, it seemed interesting to see if such a system would work to reach our goal.

2.2 Deep learning summary

Many of the works previously described leverage **Artificial Intelligence (AI)** to identify patterns in brain signals. AI is a broad field within computer science focused on creating intelligent machines. While this field is often generalized, the primary models used in everyday applications rely on specific techniques known as machine learning and deep learning. **Machine learning** refers to the capacity of

computers to learn autonomously, using algorithms to analyze data, detect patterns, and make predictions based on those patterns. **Deep learning**, a subset of machine learning, employs artificial neural networks that are sophisticated algorithms inspired by the structure and functionality of the human brain. Deep learning models excel at finding patterns in data and can be used to perform emotion recognition. In the next sections, we will make a summary of the main features and architectures. This is useful to understand the design choices of the system we implemented.

2.2.1 Classification task

Learning involves the process by which a neural network adjusts its internal parameters (weights and biases) to minimize the discrepancy between its predicted output and the actual output, given a set of input data. There are two primary learning paradigms:

- **Supervised Learning**: In supervised learning, the algorithm learns from a labeled dataset, where each example includes input data paired with corresponding output labels. The goal is to map input variables to output variables based on the labeled data. Suppose we have a dataset with m labeled examples: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$, where x_i represents the input features of the i-th example and y_i represents the corresponding output label. The objective is to learn a function $h : X \to Y$, where X is the input space and Y is the output space, such that $h(x)$ approximates the true output label y as accurately as possible for any input x. The function $h(x)$ is typically represented by a model parameterized by weights (w) and biases (b).
- **Unsupervised Learning**: In unsupervised learning, the algorithm learns from an unlabeled dataset, where no explicit output labels are provided. The aim is to discover patterns, structures, or relationships within the data without explicit guidance. Examples of unsupervised learning algorithms include k-means clustering, hierarchical clustering, principal component analysis (PCA), and autoencoders.

In our project, we used a labeled dataset. Hence, we solved a supervised learning problem. Supervised learning encompasses two main types of tasks:

- **Regression**: The goal is to predict continuous numerical values rather than discrete categories. Examples include predicting house prices based on features like location, size, and number of bedrooms.
- **Classification**: The goal is to predict categorical class labels for new instances based on past observations. This is particularly relevant for emotion recognition, where the task involves assigning an emotion depending on the data used.

There are different types of classification tasks:

- **Binary Classification**: This involves classifying instances into one of two classes. For example, predicting whether an email is spam or not spam.
- **Multiclass Classification**: Here, there are more than two classes, and the task is to classify

instances into one of multiple classes. For example, classifying images of animals into categories like cats, dogs, and birds.

- **Imbalanced Classification**: It occurs when classes are not equally represented in the dataset, with one class having significantly more instances than others. This can lead to biases during the learning process, causing the model to focus more on learning patterns from the majority class and potentially ignoring patterns from the minority classes. Techniques to address this issue include resampling methods (e.g., oversampling the minority class, undersampling the majority class) and data augmentation, which artificially increases the size of a dataset by applying various transformations to existing data samples.

As explained later, we built models to predict an emotion within a fixed subset of emotions. Therefore, it is a multiclass classification. Since classes are imbalance, we also applied techniques such as data augmentation to add some randomicity in the data and increment the number of samples.

2.2.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) [6] are a type of artificial neural network particularly adept at recognizing patterns in grid-like data structures, such as images and videos. They are also used in various tasks, including natural language processing, recommendation systems, and time series forecasting. Unlike traditional neural networks, CNNs utilize the structure of the data through local connectivity and parameter sharing.

- Local Connectivity: In CNNs, neurons in a convolutional layer connect only to a localized region of the previous layer, unlike traditional neural networks where each neuron connects to all neurons in the previous layer.
- Parameter Sharing: CNNs use filters (kernels) that are applied across the entire input, reducing the number of parameters significantly.

Convolutional layers are the core of CNNs, using convolution operations to extract features from the input data. The convolution process involves sliding a filter over the data and computing dot products to create feature maps, which highlight specific patterns.

Pooling layers follow convolutional layers to downsample feature maps, reducing their spatial dimensions while retaining important information. The stride determines the step size of the pooling window, affecting the degree of downsampling. Max pooling, a common technique, retains the maximum value within each region of the feature map.

After several convolutional and pooling layers, the feature maps are flattened and passed through fully connected layers to make the final classification.

Transfer learning is often used instead of training models from scratch. This approach involves reusing a model trained on one task for a related task, which is particularly useful when the target task has limited data. Transfer learning involves two main steps:

- Feature Extraction: Using a pre-trained model as a fixed feature extractor, where only the final layers are trained on the new task.
- Fine-tuning: Further training the pre-trained model on the new task data to improve performance.

Pre-trained CNN models, like ResNet, are frequently used for tasks such as emotion recognition.

2.2.3 LSTM

Another architecture used for emotion recognition that we exploited in our project is the **Long Short-Term Memory (LSTM)** network [7], a specific type of Recurrent Neural Network (RNN). RNNs are designed for processing sequential data, where the order of inputs matters, making them suitable for tasks like time series prediction. Unlike traditional neural networks that treat each input independently, RNNs maintain an internal hidden state that captures information from previous inputs in the sequence. This ability allows them to capture temporal dependencies, which is crucial for analyzing emotions in audio tracks of videos.

A major limitation of traditional RNNs is the vanishing gradient problem, where gradients become very small as they propagate back through time during training. LSTMs address this issue by introducing a memory cell structure that helps preserve long-term information. The key components of an LSTM unit include:

- Forget Gate: Determines how much of the previous cell state to retain or forget.
- Input Gate: Controls how much new information to add to the cell state.
- Output Gate: Regulates how much of the cell state to use as the output.

These gates manage the flow of information in and out of the memory cell, allowing LSTMs to maintain long-term dependencies and make informed decisions based on the input sequence. This architecture enables LSTMs to effectively process and analyze sequential data, such as brain signals, for tasks like personality estimation.

2.2.4 Transformers

Transformers [8] have gained significant popularity, particularly in natural language processing (NLP) tasks, due to their ability to capture long-range dependencies and handle sequential data efficiently. At the heart of transformers is the self-attention mechanism, which calculates attention scores between all pairs of positions in the input sequence to determine the importance of each token relative to others.

Self-attention relies on the scaled dot-product operation. Given a query matrix Q, a key matrix K, and a value matrix V, the attention scores are computed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where d_k is the dimensionality of the queries and keys.

Transformers often use **multi-head attention**, where multiple independent attention operations are performed in parallel, each focusing on different parts of the input sequence. For an input sequence X of length N and dimension d_{model}, multi-head attention with h heads is formulated as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ and W_i^Q, W_i^K, W_i^V are the learnable parameter matrices for each head. The concatenation of the outputs from each head is followed by a linear transformation using a learnable parameter matrix W^O.

The transformer architecture consists of an **encoder** and a **decoder**. The encoder processes the input sequence and generates a sequence of hidden representations, capturing the input's information. The decoder generates the output sequence based on these representations and previously generated tokens. While transformers are often used as end-to-end models for sequence-to-sequence tasks, the encoder alone can also be used to extract representations of input sequences. These representations can then be used in downstream models for various tasks, such as extraction patterns for emotion recognition (as we did in our project).

3 Dataset

3.1 Tone of the voice and facial expressions

The creation of a model to understand the emotion through audio and video started by choosing a dataset. There are many datasets available that offer both with users reacting differently. One of the most famous publicly available datasets is Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) [13] which was created by a team of researchers at Ryerson University in Toronto, Canada. RAVDESS contains recordings of actors portraying various emotions in both speech and song. The emotional states represented include neutral, calm, happy, sad, angry, fearful, surprise, and disgust. Each recording is labeled with the corresponding emotional category, providing ground truth data for emotion recognition research, which is particularly important for creating a deep learning model. The database includes both audio and video recordings, allowing us to analyze emotional expressions through multiple modalities, and that's why we decided to choose it for both approaches (tone of voice and facial expressions).

To train a model, we had to extract features from the video files. RAVDESS video files are provided in *mp4* format, where the information of the single file is coded in its name. The different pieces of information are separated by a "-" so they are easily extractable using a Python script. The information provided is:

- Modality: 01 = full-AV, 02 = video-only, 03 = audio-only;
- Vocal channel: 01 = speech, 02 = song;
- Emotion: 01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised;
- Emotional intensity: 01 = normal, 02 = strong;
- Statement: 01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door";
- Repetition: 01 = 1st repetition, 02 = 2nd repetition;
- Actor: 01 to 24. Odd-numbered actors are male, even-numbered actors are female.

From each of these files we extracted the audio in *wav* format and the frames in *png* format, also for simplicity we decided to map the *Emotion* into the following labels:

- 0 = Positive: [03 = happy, 08 = surprised]
- 1 = Neutral: [01 = neutral, 02 = calm]
- 2 = Negative: [04 = sad, 05 = angry, 06 = fearful, 07 = disgust]

3.2 Photoplethysmogram

In order to train a model that recognized emotions based on the *photoplethysmogram* (PPG) signal, we tried different models with different datasets to compare the performances. The datasets that we tried are: G-Rex [17], CEAP-360VR [16], and DEAP [17].

The final model is trained on the DEAP dataset since it's the most often used in the literature, and it's the one that provided us with the best results.

DEAP (Database for Emotion Analysis using Physiological Signals) consists of a multimodal dataset for the analysis of human affective states and provides physiological signals such as EEG, PPG, and body temperature related to 32 subjects when watching 40 one-minute-long music videos.

The preprocessed version of the dataset provides the signals downsampled to 128hz (from an original sample rate of 512hz) and segmented into 63-second trials.

For each segment two labels are present, one for the *valence* and one for the *arousal*, both real numbers ranging from 1 (low valence/arousal) to 9 (high valence/arousal).

Since we're interested in recognizing three types of emotions (negative, neutral, and positive) using only the PPG signal, we discarded all the other signals but the PPG. Regarding the labels, we only kept the valence and discretized it in three intervals for negative, neutral and positive emotions.

4 Models

To create the emotion recognition system, we decided to build several models that deal with different channels of interactions. We opted for training the models separately, mainly due to the absence of a computer powerful enough to process large datasets. We focused on the recognition of the tone of voice and facial expressions. We trained two different models to predict the emotions of the two inputs, and later merged the results to get a final emotion.

Moreover, we tried to estimate the participant emotion by extracting the photoplethysmogram from the face video using a technique called *remote-photoplethysmography* (rPPG) using deep learning techniques and then by feeding the obtained rPPG signal into a PPG-based emotion recognition deep learning model. This allows us to capture the signal with a contactless and remote procedure, avoiding having to use specialized equipment such as *pulse oximeters*.

4.1 Tone of voice

4.1.1 Voice feature extraction

Taking into consideration *waw* files (for example: *"03-01-01-01-01-01-01.wav"* the recording of actor 01 with a neutral voice), we created an algorithm to build a *csv* file with the metadata requested for training the model. We then implemented a data loader to load the data, and this step was useful to extract the necessary features and modify them to be suitable for training a deep learning model.
To load the data, we used the library **Librosa** [10] that provides useful functions to deal with audio files. We exploited the *librosa.load* method to load the files from the disk and turn them into a floating-point time series. RAVDESS audio files last 3.5 seconds with an initial offset of 0.5 seconds, so we trimmed the audio to remove the initial period of silence. Moreover, we used the original sampling rate of 48khz. This means that the resulting waveform is a vector of length 144.000 (value obtained by multiplying the seconds with the sampling rate). If the audio files are shorter than this value for any reason, then they are zero-padded. The extracted time series is not enough to capture the audio patterns so we applied further transformations to extract more comprehensive features. We tried several approaches such as considering the chroma features or the mel spectrograms but **Mel-Frequency Cepstral Coefficients (MFCC)** [11] allowed us to reach a much higher accuracy. Here is an overview of how MFCC work:

- The Mel scale is a perceptual scale of pitches that approximates the human auditory system's response to different frequencies, and it's based on the observation that humans perceive pitch differences on a nonlinear scale. It is defined such that a change in pitch on the Mel scale corresponds roughly to a perceptually similar change in pitch.
- MFCC extraction begins with dividing the audio spectrum into a series of overlapping triangular filters spaced uniformly on the Mel scale. Each filter bank measures the energy within a specific frequency range, and they are designed to mimic the human auditory system's frequency response.
- After computing the energy in each filter bank, the log of the energies is taken to mimic the

logarithmic response of the human ear. The log filter bank energies are then transformed using the Discrete Cosine Transform (DCT) to decorrelate the features and emphasize the most important information.

- The resulting DCT coefficients represent the MFCCs. Typically, only the lower coefficients are retained, as they capture the most relevant information about the audio signal. These coefficients are referred to as "cepstral" because they are derived from the spectrum's logarithm, which is an intermediate step in the process.

We used the *librosa.feature.mfcc* method to compute the coefficients using 40 as the number of MFCC to return and 1024 as the length of the FFT window. The result is the feature vector used for training.

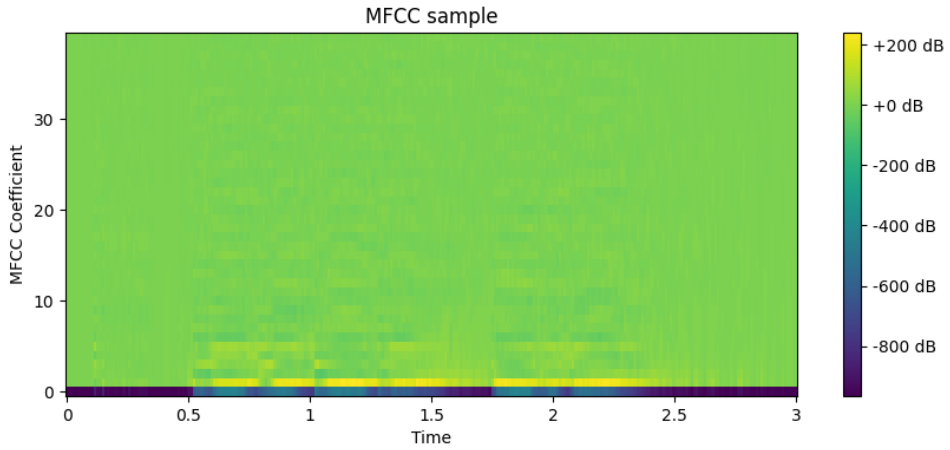

Figure 1: Sample of a MFCC of an audio from RAVDESS.

Training a model with a small dataset is not always good because it could lead to overfitting. To increase the number of samples in the dataset, we applied **oversampling**. What we did was select the class with the highest number of training examples and make copies of samples from the other classes so that all of them could have the same amount of data. For example, assuming to have 3 classes, if the most common class has 200 training examples and the remaining ones have 150 data points, we replicate 50 of them to reach 200 data points in all classes. Of course, we don't want to use the same audio multiple times and for this reason, we applied **data augmentation** only to the replicated data. The transformation that we applied was the **Additive White Gaussian Noise (AWGN)** [12]. The applied function takes parameters such as the input waveform, sampling rate, desired signal-to-noise ratio (SNR) range, and pitch shift range. First, it generates AWGN and normalizes it along with the input waveform. Then, it adjusts the noise covariance based on the desired SNR. Next, it adds the normalized noise to the waveform to apply AWGN. Additionally, it randomly selects a pitch shift amount within the specified range and applies it to the waveform using the *librosa.effects.pitch_shift*

function. This changes the perceived pitch of the audio without altering its duration. To sum up, this function is used to augment audio data by adding noise and introducing variations in pitch to the original waveform.
Another preprocessing step consisted of **scaling the data**. We noticed that we could reach a much higher accuracy by scaling the audio files. We used a standard scaler provided by *Sklearn* that we fitted using the training set, and then used that for scaling the rest of the data. We also save the scaler offline so that it can be used during the demo on new data. Scaling audio files is important for several reasons:

- Consistency in Feature Extraction: Emotion recognition models often rely on various acoustic features such as pitch, intensity, and spectral characteristics and normalizing these features ensures that the model is not biased towards certain files due to differences in amplitude;
- Training benefits: Scaling helps the model converge faster during training and prevents it from being dominated by features with larger scales. Moreover, it can help to reduce overfitting;
- Generalization: By scaling the audio files, we make sure that the model learns patterns that are applicable across the entire dataset, rather than being skewed by the scale of individual files. This helps the model generalize better to unseen data.

The data loader is created by simply splitting the data in training, testing and validation. The training dataset is used for training the model, the validation dataset is used for testing the model at training time, while the testing set is used for testing the best model once training is done. The splitting ratio used is 80/10/10 meaning that 80% of the data is used for training and the remaining is split equally for validation and testing. As already stated before, oversampling was only applied to the training set. Concerning the scaling part, we fitted the scaler using the training set and then scaled all the data with it.

4.1.2 Audio models

We tried several deep learning models, starting from simple CNNs to more complex architectures that considered both spatial and temporal features. In general, CNNs weren't enough to reach a satisfying accuracy, so we decided to create a custom architecture composed of a CNN and an LSTM.
CNNs are a type of deep neural network particularly effective for processing grid-like data, such as images or spectrograms. They consist of layers of convolutional filters followed by non-linear activation functions and pooling layers where convolutional filters scan the input data to detect local patterns and pooling layers downsample the feature maps, reducing the spatial dimensions while retaining important features. **LSTMs** are a type of recurrent neural network (RNN) designed to handle sequential data with long-range dependencies, such as speech and audio files. They contain memory cells that can store information over time and gated mechanisms to control the flow of information. Moreover, it has three gates (the input gate, the forget gate, and the output gate) that regulate the flow of information into and out of the memory cell, allowing LSTMs to selectively remember or forget information over long sequences. **Combining both architectures allows for the integration of local and global**

context information, enabling the model to make more informed predictions by considering both short-term and long-term dependencies. We decided to call this model **AudioNetCL**.
The LSTM input size is given by the number of MFCC used during the data loading part. The hidden size and the number of layers were respectively fixed to 128 and 2. We also tried a higher number of layers, but we didn't manage to train a model since LSTMs take a long time to train parameters since computations are done sequentially. Concerning the CNN, it is a three-layer architecture where a single layer is composed of a convolutional block, batch normalization (for accelerating the training process and stabilizing the gradients), ReLU (for non-linearity), max pooling (for decreasing the spatial resolution at each step), and a dropout layer (to increase the generalization of the model). The kernel size, stride, and dropout probability used for training are respectively 3, 1, and 0.2. At each step, the audio feature vector is embedded by the LSTM and CNN. The result is concatenated, creating a unique feature vector that is used for training. The predictions are done by using a simple fully connected layer.

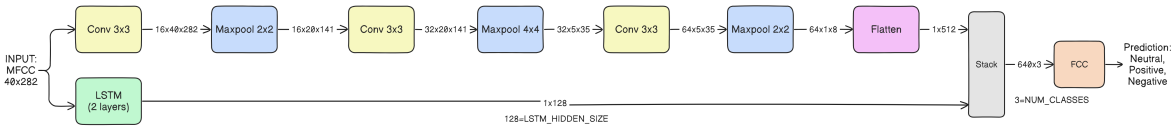

Figure 2: Summary of AudioNetCL model created by stacking the encodings of am LSTM and a CNN

Another test was trying to substitute the LSTM with a **Transformer** that is a state-of-the-art architecture. They were introduced by Vaswani et al. in the paper "Attention Is All You Need" in 2017 and the key innovation of transformers lies in their mechanism for handling sequential data without recurrent neural networks (RNNs) or convolutional neural networks (CNNs). Instead, transformers rely on self-attention mechanisms to weigh the importance of different words in a sequence when processing each data point. Transformers can be used for both encoder-only tasks (such as language modeling) and encoder-decoder tasks (such as machine translation). In encoder-decoder architectures, the input sequence is first processed by an encoder stack of transformer layers, and then the output is decoded into the desired format by a decoder stack of transformer layers. In our case, we used the encoder only since we only needed to extract patterns for the audio files. To enhance the expressiveness and capacity of self-attention, transformers typically use multi-head attention, where the self-attention mechanism is applied multiple times in parallel, each with different learned projection matrices. We used 4 heads in our model. The input of the model is the number of MFCC (like in the LSTM). As in the previous architecture, we also used two CNNs in parallel. The general structure is the same as the one in the AudioNetCL model, but we changed the number of layers. The first CNN has 3 layers, while the second one has only 2 layers. We tried this approach because we wanted to analyze what would have happened if we captured spatial relationships at a different level and put them together for learning patterns in audio files. As in the previous model, we concatenated the embeddings of the audio files creating a single feature vector, and then we trained the model using a simple fully connected layer.

We called this second model **AudioNetCT**.

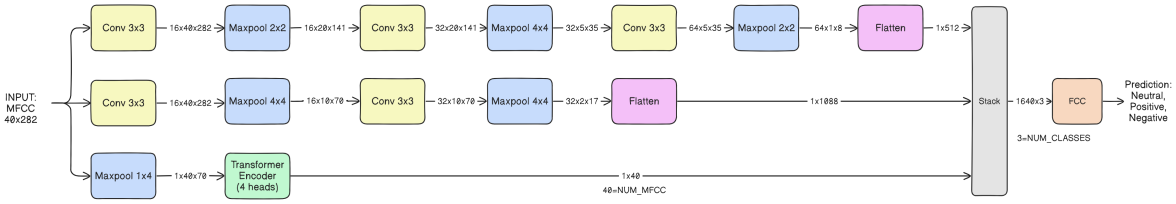

Figure 3: Summary of AudioNetCT model created by stacking the encodings of a Transformer encoder and two CNNs

4.1.3 Experiments and results

We trained the AudioNetCL model for 500 epochs using a learning rate of 1e-3. The hidden size of the LSTM used is 128 while the number of layers is 2. We tried to increase these numbers but since LSTM computes weights sequentially, the time needed to train a model arises dramatically, so we couldn't increase them further for our limited computational power. We noticed that the model tended to overfit because training metrics were higher than the validation ones. We tried to decrease overfitting by using dropout (with a probability of 0.3) and weight decay implemented by the AdamW optimizer (using a regularization parameter of 1e-3). We measured several metrics to see how the model behaved:

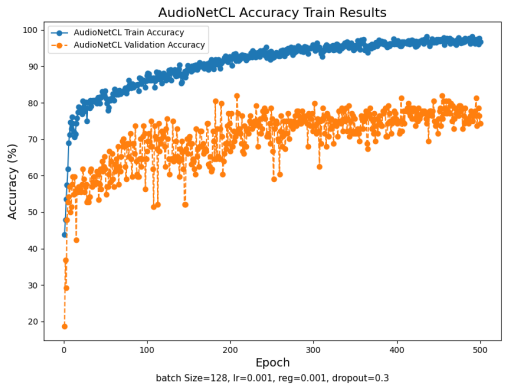

Figure 4: Accuracy in AudioNetCL model

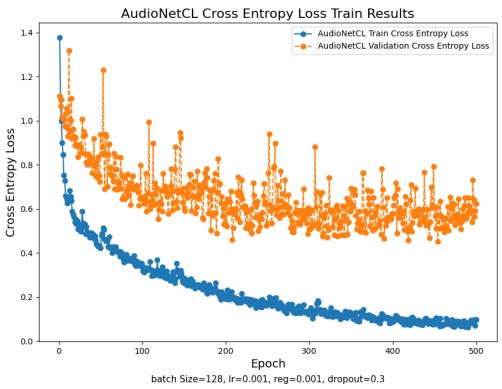

Figure 5: Cross Entropy Loss in AudioNetCL model

Figure 6: Recall in AudioNetCL model

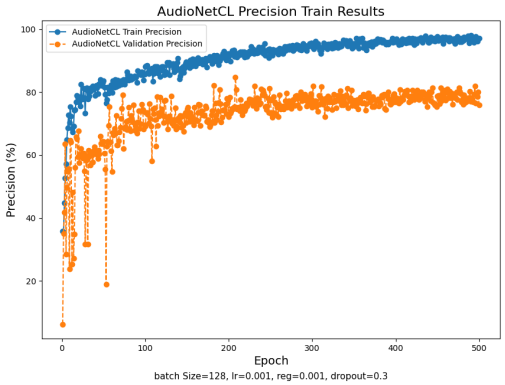

Figure 7: Precision in AudioNetCL model

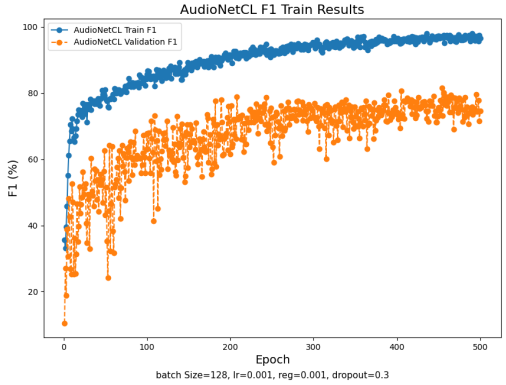

Figure 8: F1 in AudioNetCL model

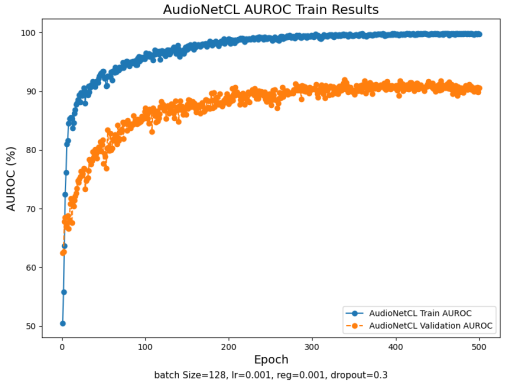

Figure 9: AUROC in AudioNetCL model

From the graphs, it is possible to notice that we managed to reach good results in the validation set. The overfitting is still present, even though we tried to decrease it using several approaches. Moreover, it is possible to notice how the validation metrics go up and down during the training process, but this is probably due to the batch size chosen, and the audio distribution tested in each epoch. We were satisfied with the results, but we wanted to try a state-of-the-art architecture, so we also measured the performances of AudioNetCT:

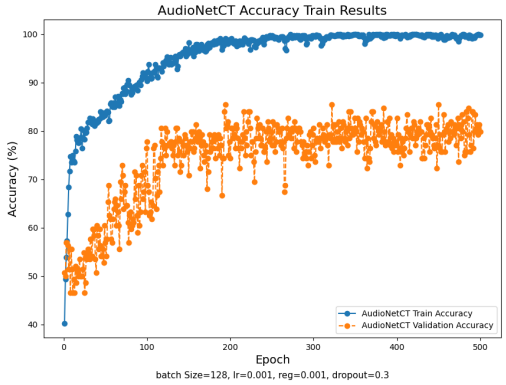

Figure 10: Accuracy in AudioNetCT model

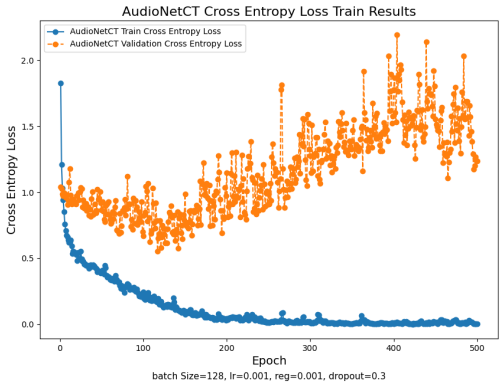

Figure 11: Cross Entropy Loss in AudioNetCT model

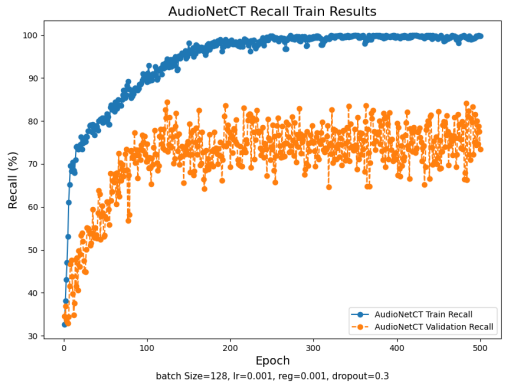

Figure 12: Recall in AudioNetCT model

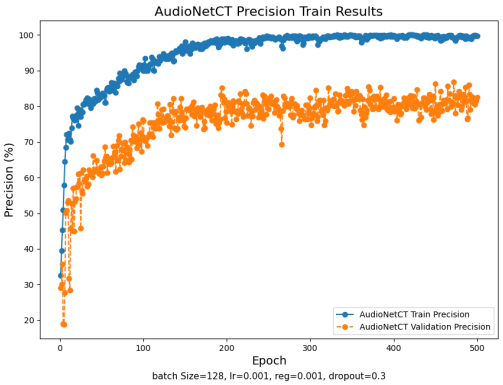

Figure 13: Precision in AudioNetCT model

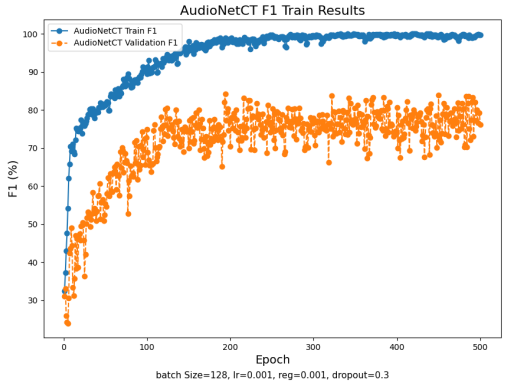

Figure 14: F1 in AudioNetCT model

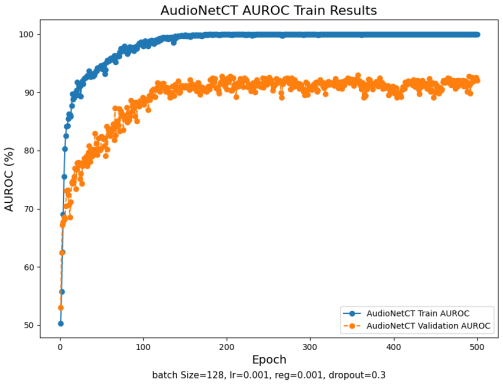

Figure 15: AUROC in AudioNetCT model

AudioNetCT generally performed better than AudioNetCL. We notice though an increase in the loss around the epoch 200. To overcome this problem, we simply applied a model selection in which we discarded the models where the loss tended to arise. Selecting the best models for both architectures (best in terms of accuracy) and testing them using the testing set, we got:

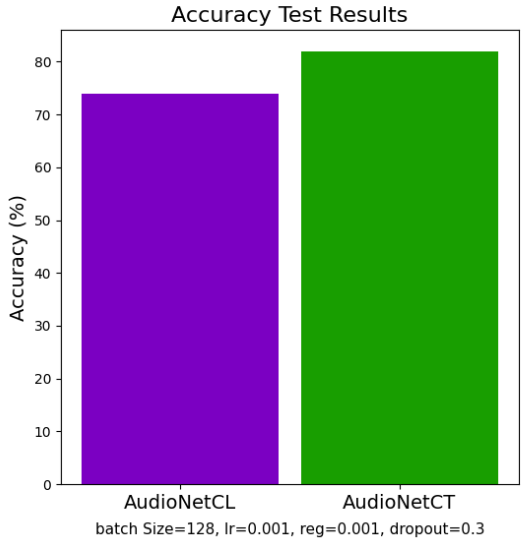

Figure 16: Accuracy comparison in the audio models

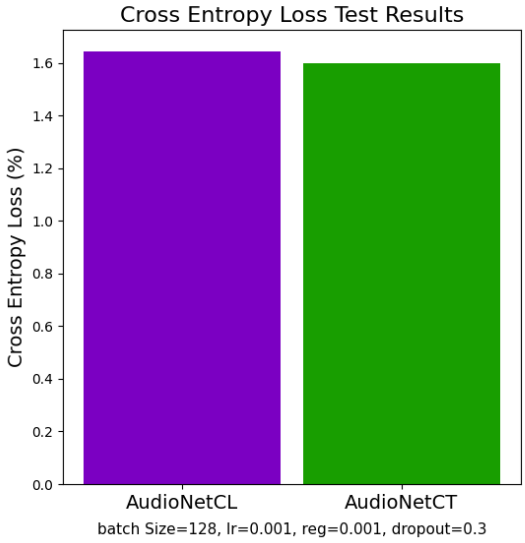

Figure 17: Cross Entropy Loss comparison in the audio models

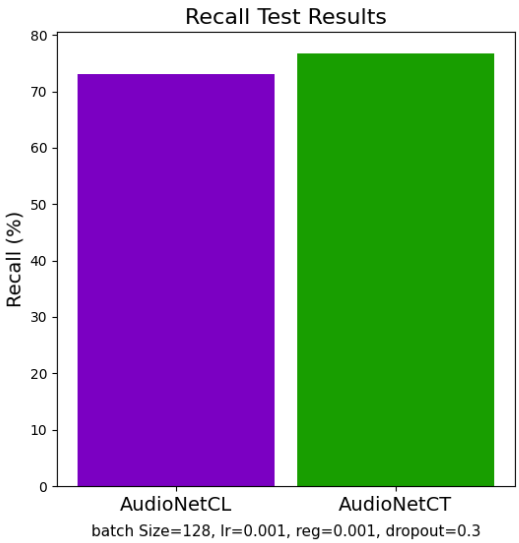

Figure 18: Recall comparison in the audio models

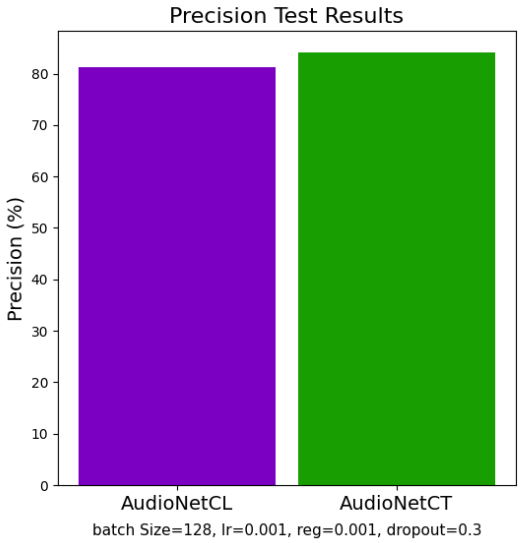

Figure 19: Precision comparison in the audio models

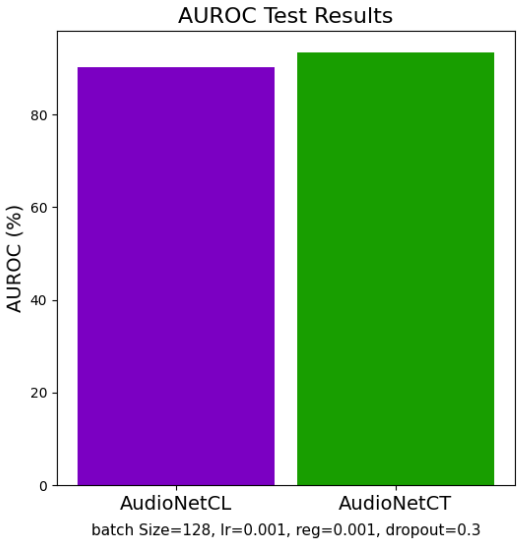

Figure 20: AUROC comparison in the audio models

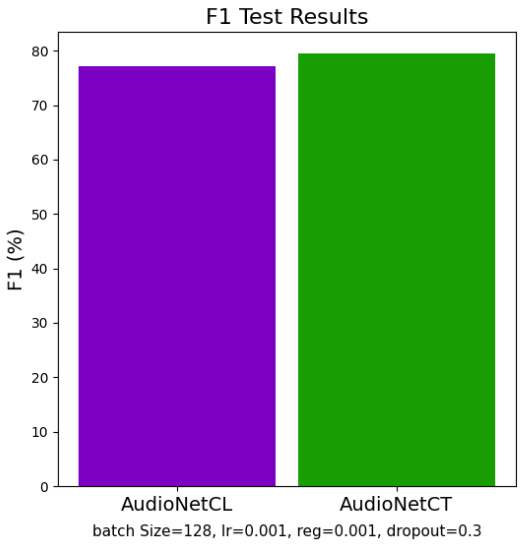

Figure 21: F1 comparison in the audio models

AudioNetCT performances are better also on the testing set, so we eventually decided to use this architecture for the demo.

4.1.4 Limitations

This model has several limitations, mostly due to the nature of the dataset. The audio recordings in RAVDESS are high-quality and standardized for consistency. They are captured using professional recording equipment in a controlled studio environment, minimizing background noise and other distortions. Moreover, recordings are from 24 professional actors (12 male, 12 female), representing a diverse range of ages, ethnicities, and accents, and this helps ensure that the database is applicable across different populations and cultural contexts. Since they are actors, emotions are particularly exaggerated, and this makes it very difficult to perceive micro emotions that are possibly more common in a system like ours. This is to say that in order to achieve a high accuracy in a real scenario, we have to limit the background noise as much as possible and avoid micro emotions that would be otherwise predicted as "neutral". There are other datasets (such as BAUM-1) that also include natural emotions, but they are not publicly available, so we eventually decided to stick with RAVDESS.

4.2 Facial expressions

4.2.1 Video feature extraction

To train video emotion recognition models, we extracted frames of subjects' faces directly from the original RAVDESS videos using a combination of haarcascade classifiers and considering the intermediate portions of video where the subject has the correct expression (since at the beginning/end of video they very often have a neutral expression). From each video, we extract *∼20* frames and for each of these we detect the subject's face and we genenerate an image file *.png* with 224x224 resolution for a total of *∼30k* images. When we generate the images we keep the same video files name encoding, adding only the extracted frame number (for example: if we have the video file named *3-01-01-01-01-01.mp4* we generate a set of frames containing the subject's faces named 3-01-01-01-01-01_x.png, where x is the extracted frame number). Each subject will then have *∼1200* frames of their face with different expressions.

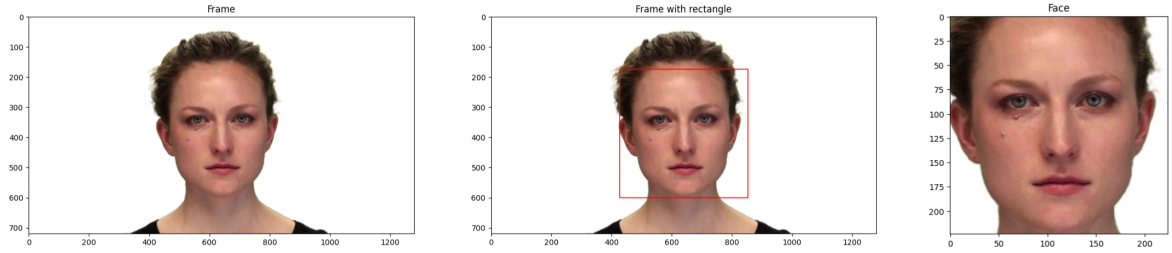

Figure 22: Captured frame, face recognition and face extraction

Then, we created an algorithm to build a *.csv* file from these extracted frames with the required metadata for training the model and implemented a data loader to load the data. The data loader is created by following two criteria of frames splitting:

Overlapping frames of the subjects

This means that all the frames of the subjects are randomly split between train/validation/test sets (using the rule of 80/10/10). In this case we can have frames of the same subject spread across all three sets.

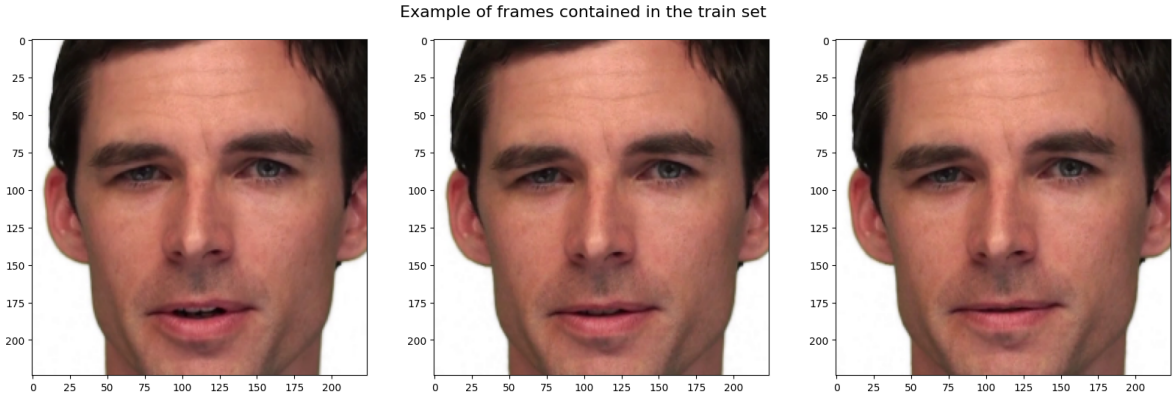

Figure 23: Example of frames contained in the train set

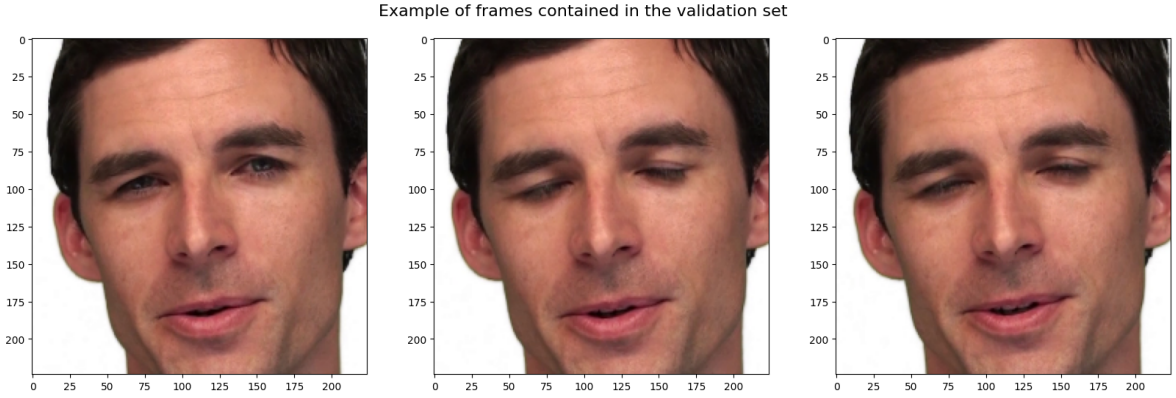

Figure 24: Example of frames contained in the validation set

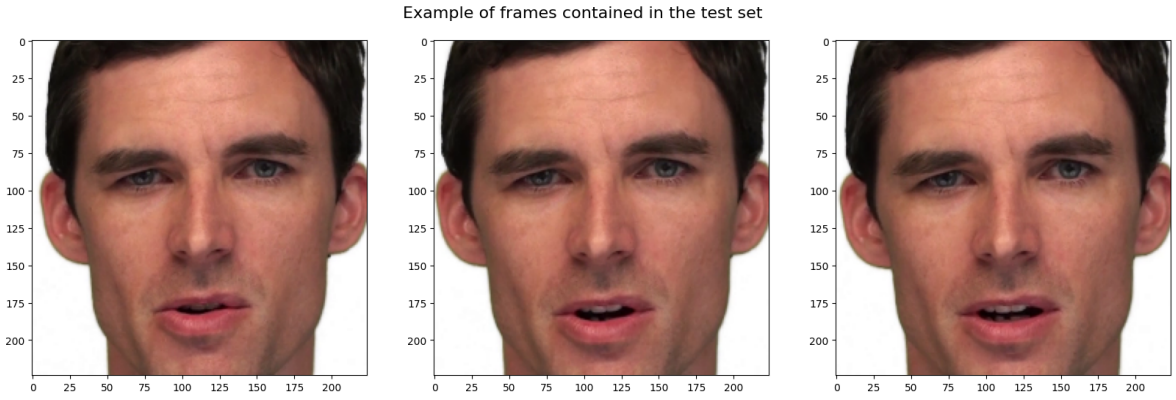

Figure 25: Example of frames contained in the test set

Without overlapping the frames of the subjects

This means that, given **n = total number of subjects in the dataset**, we use the frames of n - 2 subjects for train, 1 for validation and 1 for test.

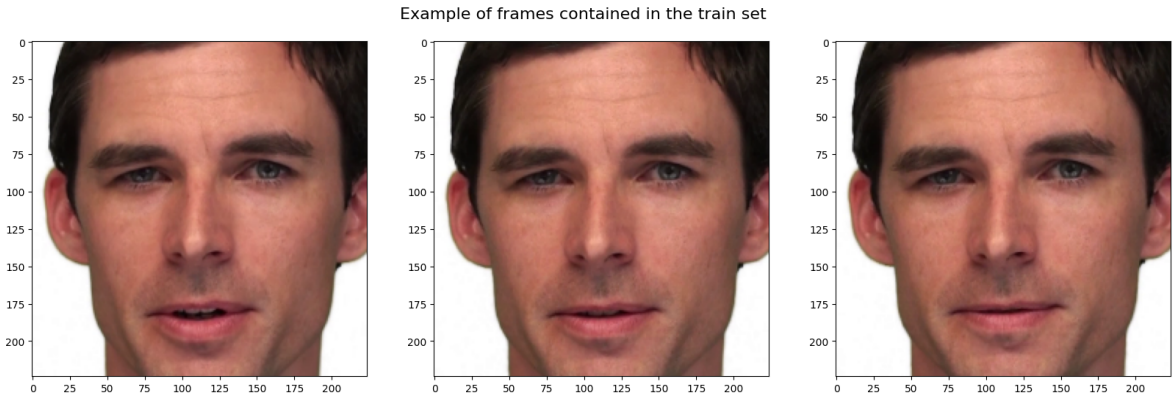

Figure 26: Example of frames contained in the train set

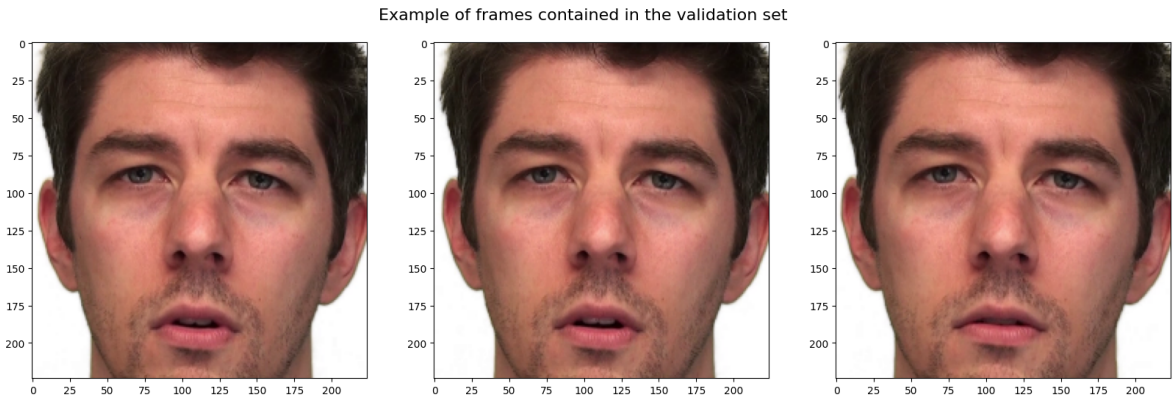

Figure 27: Example of frames contained in the validation set

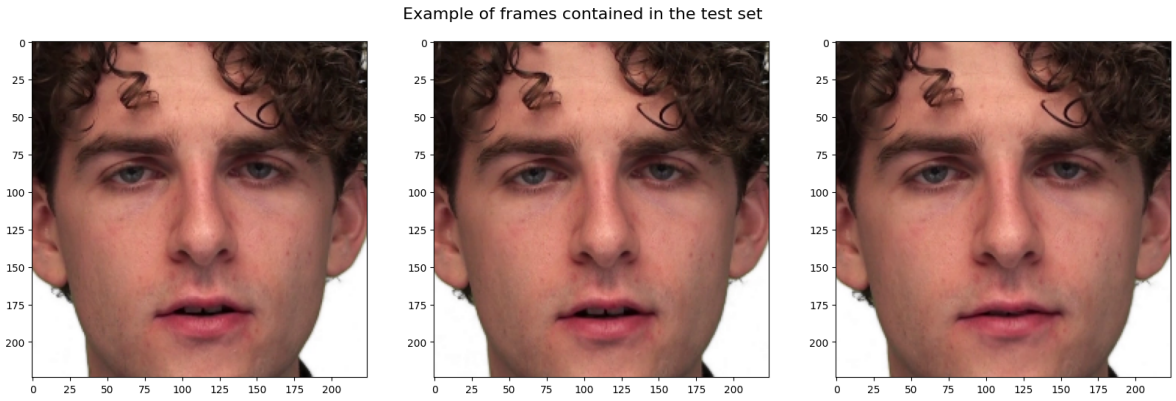

Figure 28: Example of frames contained in the test set

The results of these two splitting methods will be discussed in more detail in the next section. In general, the train images were normalized using ImageNet normalization and balanced by applying some transformations to the images, such as random rotations and horizontal/vertical flipping.

4.2.2 Video models

We started with **CNNs** and then moved to **Vision transformers** to understand if we could improve the results. Train a CNN from scratch led to unsatisfactory results so we eventually decided to test some pretrained models (testing various configurations and not using them as black boxes) such as **ResNet101** [20] and **DenseNet121** [21] and eventually picked the last one which gave us the best results.

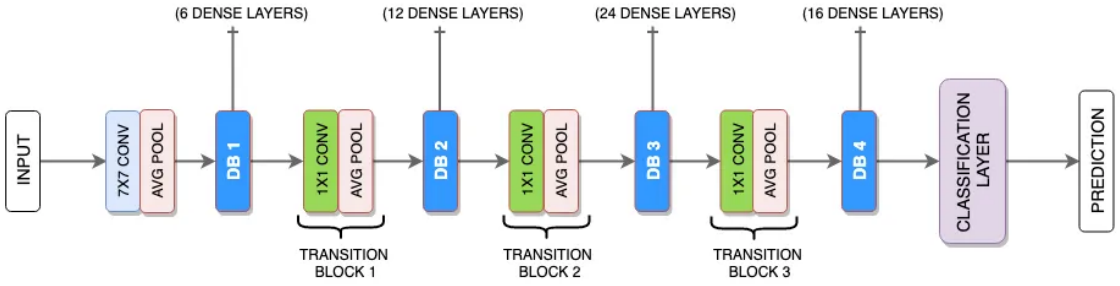

Figure 29: DenseNet121 architecture (Image source)

Concerning the **Vision Transformers**, we opted for **ViT-b/16** [22] pre-trained on ImageNet.

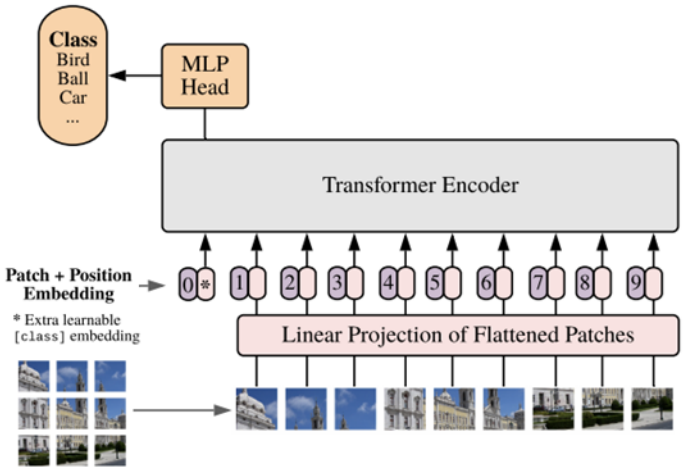

Figure 30: Vision Transformers (ViTs) architecture (Image source)

For each model, we applied hyperparameter tuning on a small set of data to find the best hyperparameters for training and in all pretrained models we applied fine-tuning by training again the head of the original models.

4.2.3 Experiments and results

We trained both **densenet121** and **vit-pretrained** for 30 epochs using a learning rate of 1e-3. We noticed that the model tended to overfit because training metrics were higher than the validation ones. We tried to decrease overfitting by using dropout (with a probability of 0.2) and weight decay implemented by the AdamW optimizer (using a regularization parameter of 1e-3). We measured several metrics to see how the models behaved:

Without overlapping frames of the subjects

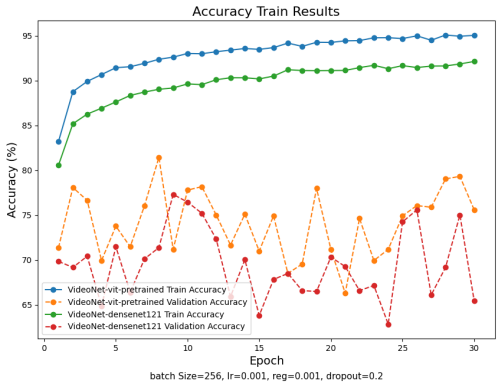

Figure 31: Accuracy in VideoNet models

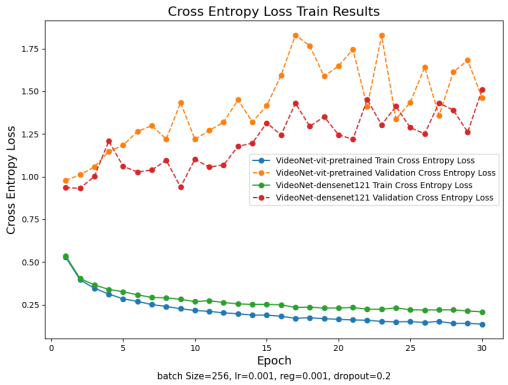

Figure 32: Cross Entropy Loss in VideoNet models

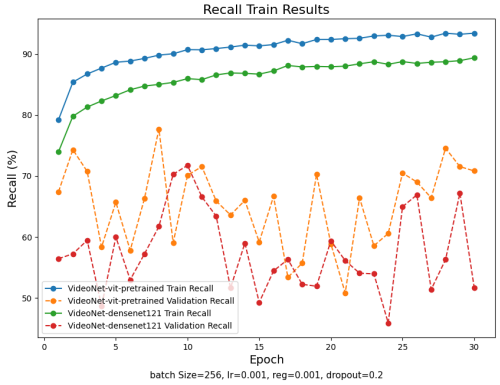

Figure 33: Recall in VideoNet models

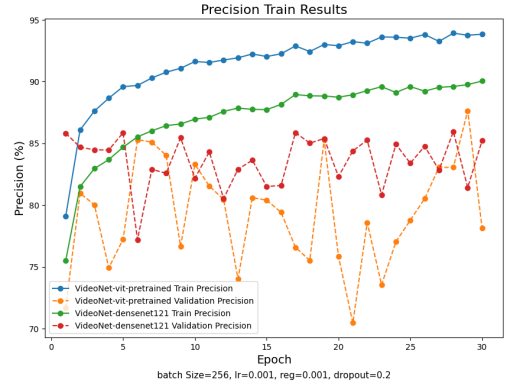

Figure 34: Precision in VideoNet models

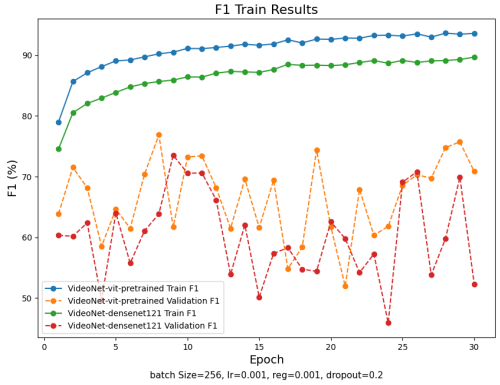

Figure 35: F1 in VideoNet models

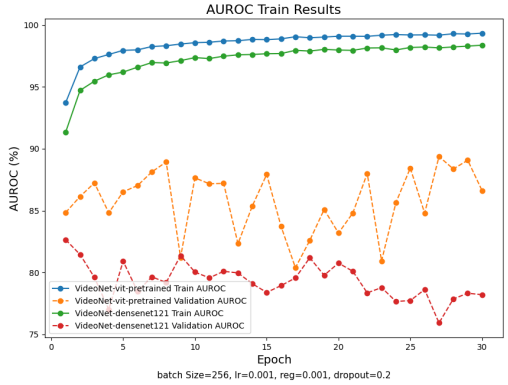

Figure 36: AUROC in VideoNet models

From these plots, it is possible to notice that this type of splitting causes tremendous overfitting in both models and despite various parameter changes we have not been able to reduce it. We think the cause is due to the fact that having n - 2 subjects for train, 1 for validation and 1 for test the models have no reference regarding the subject used for the validation set, so they tend to rely only on all the others and this can cause the following results. Moreover, it is possible to notice how the validation metrics go up and down during the training process, but this is probably due to the batch size chosen and the frames' distribution tested in each epoch.

By overlapping the frames of the subjects

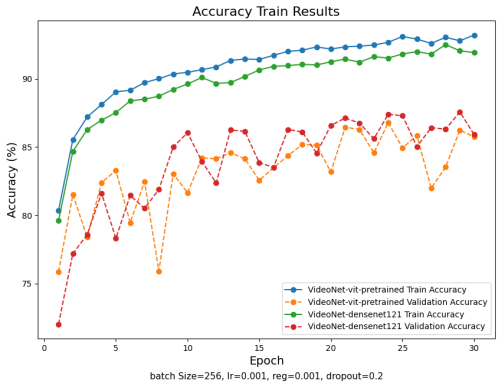

Figure 37: Accuracy in VideoNet models

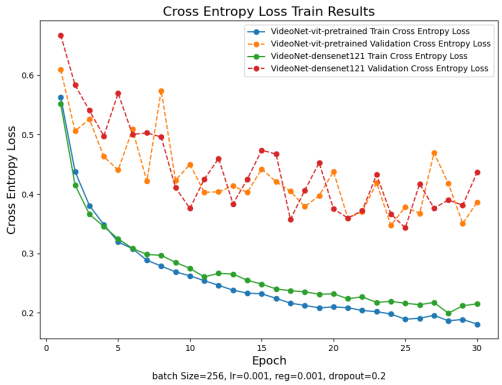

Figure 38: Cross Entropy Loss in VideoNet models

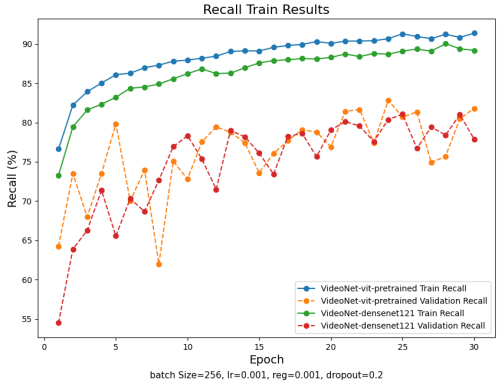

Figure 39: Recall in VideoNet models

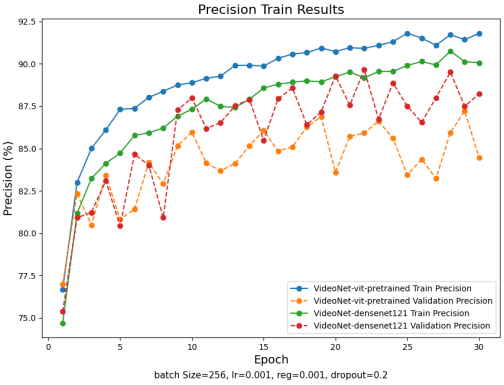

Figure 40: Precision in VideoNet models

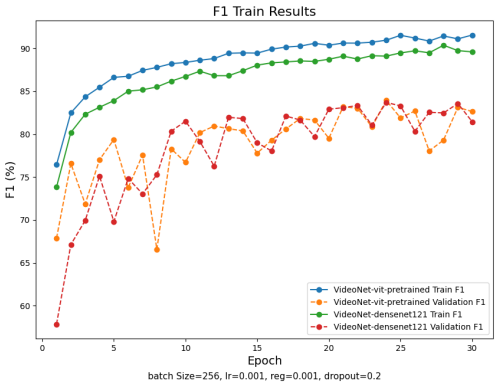

Figure 41: F1 in VideoNet models

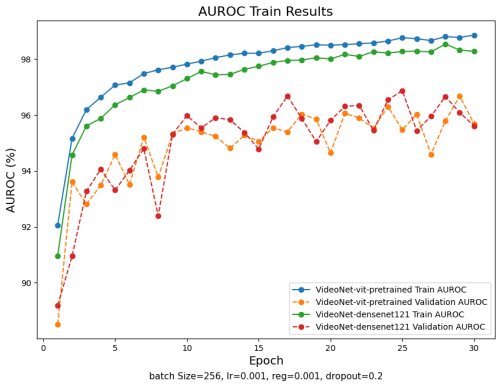

Figure 42: AUROC in VideoNet models

Using this other splitting method instead, we get better results. In fact, having mixed frames in the train/validation set from all the subjects helped the models achieve better performances. More in details, we can see how the performance of vit-pretrained and densenet121 are similar, although some metrics favor the transformer-based model, stopping for both models at around 30 epochs because looking at these results we see a tendency toward overfitting.

Test results

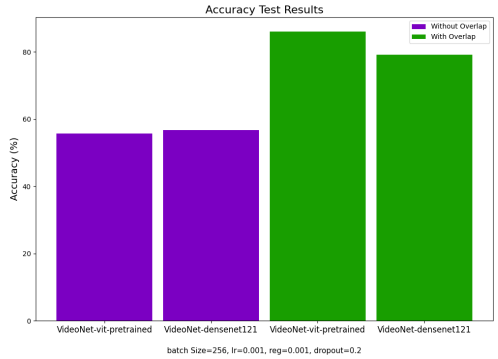

Figure 43: Accuracy in VideoNet models

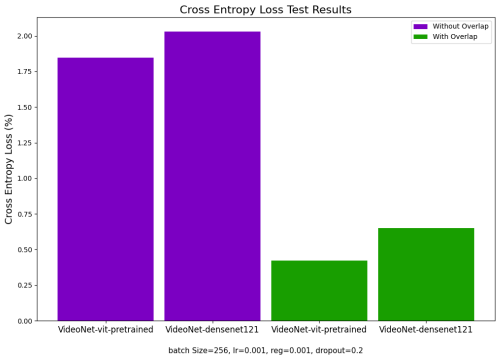

Figure 44: Cross Entropy Loss in VideoNet models

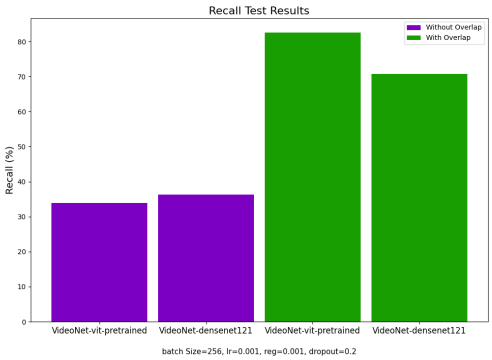

Figure 45: Recall in VideoNet models

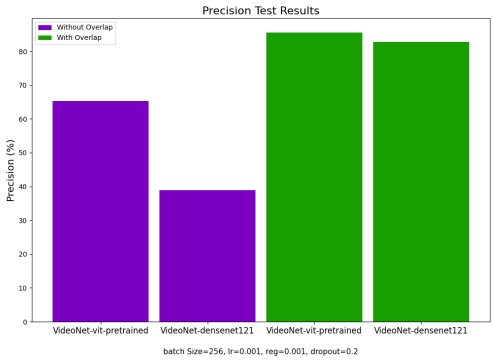

Figure 46: Precision in VideoNet models

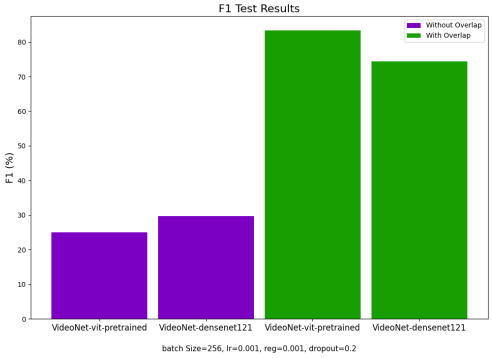

Figure 47: F1 in VideoNet models

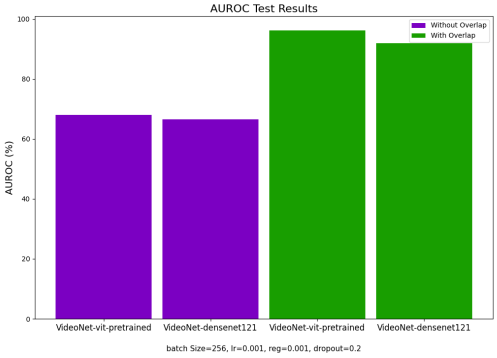

Figure 48: AUROC in VideoNet models

Through these graphs we see even better how the performance degrades when using the splitting method that does not overlap subject frames in the datasets instead of the others. Given the results obtained in the train, validation and test phase we decided to use the models trained using the overlap of subject frames, more specifically we decided to use **vit-pretrained** model as architecture for the demo since it is the one that even during the testing phase reported better results, calling it **VideoNet**.

4.2.4 Limitations

Linking back with the limitations encountered in the audio model, there are also some concerning about the video model. The main limitations are due to the fact that the RAVDESS videos used to train the model were recorded with professional equipment in a controlled environment, so to test the model and achieve good performance, the same instruments and recording conditions should be used. In addition, the extracted emotion frames lack generalization due to the small number of subjects

considered during the training phase. This could be improved by using a dataset with more subjects or adding more subjects to the current dataset to increase generalization.

4.3 Photoplethysmogram

The emotion recognition based on the photoplethysmogram physiological signal is divided into two main parts: obtaining the PPG signal with the rPPG procedure; and classifying the emotion using the obtained signal.

4.3.1 Remote photoplethysmography

Remote photoplethysmography is a technique used to measure the Photoplethysmogram signal, also called Blood Volume Pulse (BVP), without direct contact with the human skin. The technique involves using a high-resolution camera to capture a video of a person's face and analyze the changes in light reflection caused by blood volume variations.

To achieve that, we used an open-source toolbox called *rPPG-toolbox* [18] which contains various supervised and unsupervised deep learning methods for rPPG. The toolbox is written to be used with a set of compatible datasets, and because of this, we extended it to work with our custom data.

Since it already contains pre-trained state-of-the-art models for rPPG, we don't need a training process.

For the model, we used the DeepPhys [19] model, which is based on *convolutional attention networks*.

When feeding a video to this model, we need to extract the region of interest where the face is located for each frame and resize the image to be compatible with the model architecture. Once obtained the output, we post-process it in the same way the DEAP dataset was preprocessed (see 4.3.3), in order for the extracted data to be consistent with the data the emotion recognition model was trained on.

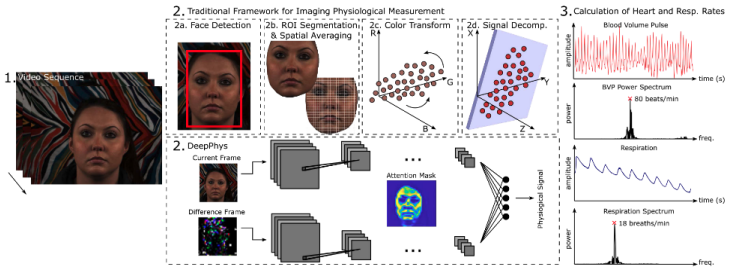

Figure 49: DeepPhys rPPG extraction pipeline

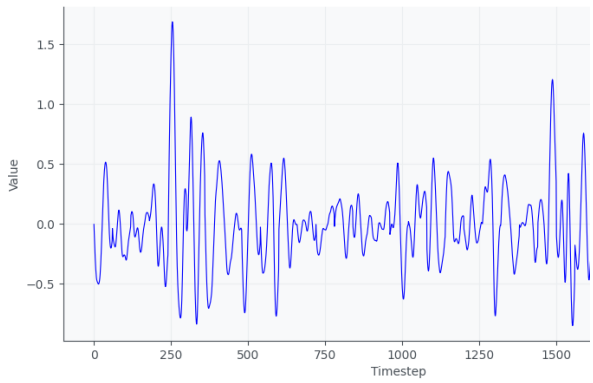

Figure 50: Example of the extracted PPG by using rPPG

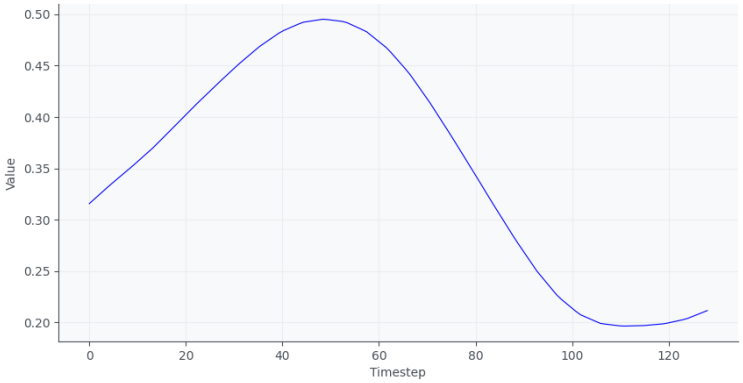

Figure 51: Example of a single pulse taken from the extracted PPG

4.3.2 Photoplethysmogram emotion recognition

Once we've obtained the PPG signal using the rPPG technique, we need a way to classify the emotion of the individual, which we do by using a 1D CNN-based deep learning model.

4.3.3 Signal preprocessing

Apart from being downsampled to 128hz and divided into segments of 63 seconds, the PPG signal present in the DEAP dataset is not further pre-processed. Our preprocessing pipeline is the following:

1. Discretize the labels into three groups
2. Remove the signal trend
3. Remove the noise using a *bandpass filter*
4. Smooth the signal by using a *moving average filter*
5. Normalization
6. Slice the signal into single-pulse windows
7. Perform Fast Fourier Transform to extract frequency features of the signal
8. Undersampling training data

Discretize the labels As stated before, the dataset provides real-valued labels for valence and arousal in the range of [1-9].

The combination of *valence* (which measures the positivity or negativity of the emotions) and the *arousal* (which measures the intensity of the emotion) can represent numerous different emotions.

Designing a model to regress valence and arousal real values to recognize this large amount of emotions is a complex task, and because of this we decided to to a 3-class multiclassification on *negative*, *neutral* and *positive* emotions, which can be done by only considering the *valence* label.

We discretize it into three intervals:

- Negative: valence $\in [1, 2]$
- Neutral: valence $\in (2, 7]$
- Positive: valence $\in (7, 9]$

Signal detrending Being captured from a *pulse oximeter*, which relies on light to detect blood volume changes, even small movements of the finger during wear can introduce some bias in the readings, making the signal appear to trend upwards or downwards.

This trend is not beneficial for the model and has to be removed. A common *detrending* technique is to fit a polynomial to the signal and then subtract it from the signal itself, for this reason, we decided to use a 50-order polynomial.

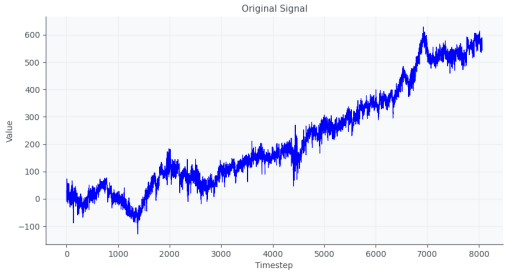

Figure 52: Original Signal

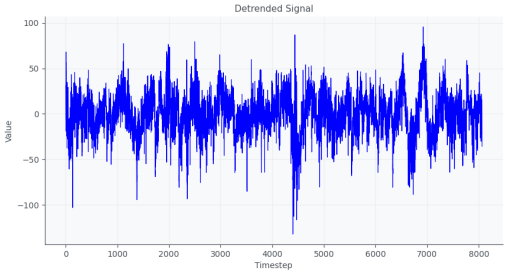

Figure 53: Detrended Signal

Signal denoising To remove eventual noise present in the signal, we applied a *band-pass filter* with an interval of [0.5hz, 40hz]. The band-pass filter removes all the frequencies that do not fall into that particular interval, effectively denoising the signal.

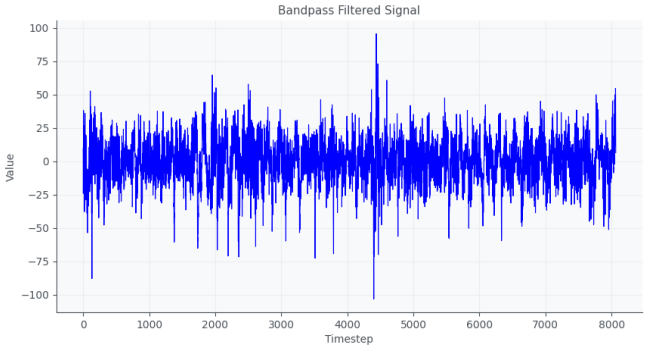

Figure 54: Signal after Band-pass Filter

Signal smoothing To further remove some noise, we applied a *moving average filter* with a window size of 10. This particular filter applies a moving window on the signal and substitutes the values inside with their mean, smoothing it. It has to be used carefully since too large of a window can smooth the signal too much, causing a loss of useful information.

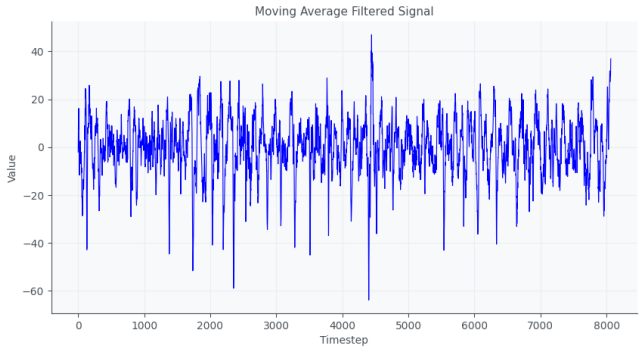

Figure 55: Signal after Moving Average Filter

Normalization The original mean μ and standard deviation σ of the data are respectively $\mu = 0.55$ and $\sigma = 4206.83$. It's known that standardized data ($\mu = 0$, $\sigma = 1$) is beneficial for the model, and because of this, we standardized the dataset by subtracting the overall dataset mean from each signal and then dividing it by the standard deviation.

Single-pulse window splitting As said before, each signal in the DEAP dataset is 63 seconds long with a sample rate of 128hz, resulting in an 8064-dimensional vector.

Apart from being more computationally expensive, we noticed that using the whole 63-second window to predict the emotion would actually degrade the recognition accuracy. We experimented with multiple window sizes, and, by taking inspiration from similar works in the literature (see [24, 23]) we

finally chose a 1s window size, which translates to a vector of 128 elements, that represents a single PPG pulse.

When performing the slicing, we were also able to remove the bad-quality signals by analyzing the number of peaks.

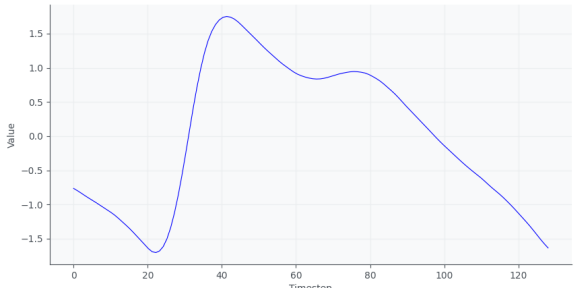

Figure 56: Good quality signal (single pulse)

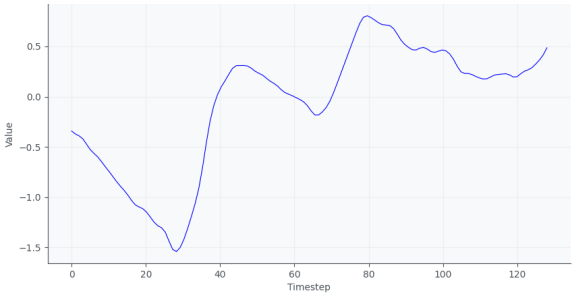

Figure 57: Bad quality signal (single pulse)

Fast Fourier Transform feature extraction For the signal to be used by the 1D-CNN, we decided to extract frequency features using Fast Fourier Transform, and produce a vector made out of stacking these elements:

- The original signal x
- The **magnitude** of the signal, obtained by taking the absolute value of the Fourier transform result.
- The **phase** of the signal, obtained by taking the arctangent of the imaginary and real parts of the Fourier transform result.

Each element represents a different channel for the 1D-CNN.

We chose Fast Fourier Transform over Short-Time Fourier Transform or Wavelet Transform, since we don't need temporal information if we're using only a single pulse window.

It has to be noted that many experiments have been done with different window sizes (see 4.3.6), by maintaining temporal information with larger windows and sequence models, but this final preprocessing and architecture is the one that gave us the best results.

4.3.4 Model architecture

As the backbone model that is responsible for the actual emotion classification task, we built and trained a 1-dimensional Convolutional Neural Network from scratch. The architecture is the following:

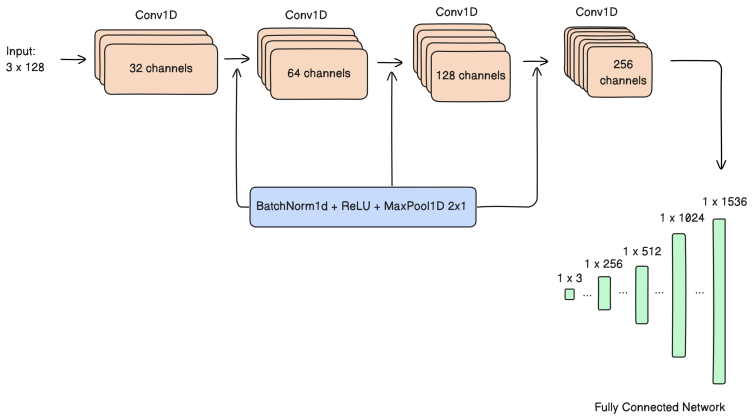

Figure 58: Model architecture of the proposed PPG emotion recognition model

We trained the model for 160 epochs, with a batch size of 512, and a learning rate of 0.0001. We used the ADAM optimizer with a weight decay of 0.03 and Cross Entropy Loss as the criterion. These parameters were chosen after an extensive hyperparameter tuning phase.

We split the dataset into 80% training, 10% validation, and 10% testing. For the evaluation to be reliable, we made sure to split the dataset just before the window slicing, otherwise two windows taken from the same signal could be present one in the training set and one in the validation set, making the final evaluation not valid. In this way, the model is evaluated on the windows that are taken from signals that the model has never seen during the training.

To prevent overfitting, we used early stopping by monitoring the evaluation metrics on the validation set.

4.3.5 Experiments and results

After experimenting with different architectures, preprocessing pipelines and extensive hyperparameter tuning, the best model gave us an **accuracy** on the testing set of **51.3%** and a **recall** of **40.4%.**

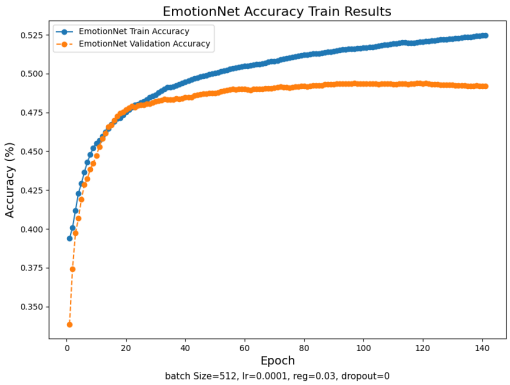

Figure 59: Training and Validation Accuracy

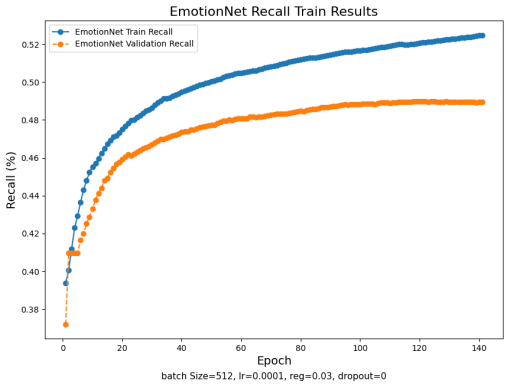

Figure 60: Training and Validation Recall

4.3.6 Discarded models

As said before, during the development of the model, different datasets, architectures and ideas have been developed, and the one with the best result has been chosen.

The ideas we experimented with are:

- **LSTM** sequence-to-sequence modeling using the CEAP-360VR dataset, which provided continuous labeling (one label for each sample in the PPG signal) instead of one label for the entire signal;
- **1D CNN + LSTM** on the DEAP and G-REx PPG signals of different lengths;
- **2D CNN-LSTM** hybrid on DEAP PPG signal of different lengths in order to avoid Short-Time Fourier Transform or Wavelet Transform and learn the features using the CNN.
- **2D CNN** on the Wavelet Transformed DEAP PPG signal;

All these experimented methods didn't provide us with satisfactory results, so we decided to use the **1D CNN** on the **single-pulse PPG** signal as our final method.

4.3.7 Limitations

As we can see from the final results, the proposed model has a lot of limitations, which may be caused by a high number of factors. One of these can be the fact that the DEAP dataset contains a single label for the entire 63-second span, where the user's emotion may oscillate in the meantime. Perhaps this problem can solved by considering the full 63-second signal with temporal relationships, maybe by using a more sophisticated combination of preprocessing techniques and sequence model. Furthermore, the user's self-evaluation of their own emotion may not be always correct, which increases the noise inside the dataset.

5 Fusion

5.1 Introduction

Once obtained the different models, we had to fuse their results to make the emotion classification more robust. We faced various challenges because we had to consider many possible cases regarding data availability in an experiment. For example, if we have the face input, but the user doesn't speak, do we need to discard the experiment or consider only the data we have? We are going to focus on the main design choices in the next sections. For now, we aim to analyze the pre-processing operations we performed before fusing the data.

5.2 Audio Pre-processing

Let's assume the user provides an audio track of their voice. Since we consider both audio and video, the length of the audio file should be exactly the same as the video file from which the audio is extracted. The audio file is preprocessed in a way that it can be later applied to the AudioNetCT model previously explained. This means that if the audio track is longer than 3 seconds, it is split in audio windows of 3 seconds with a stride of 0.5 seconds. For example, assuming to have an audio file of 5 seconds, we divide the audio file into audio windows of 3 seconds to have 5 of them (the first window would go from 0 to 3 seconds, the second one from 0.5 to 3.5, and so on...). We decided to follow the approach of overlapping windows to avoid situations in which the user speaks between 2 windows: since the slide windows, it is likely that one window contains all the user speech with no interruption and this translates to a more confident prediction.

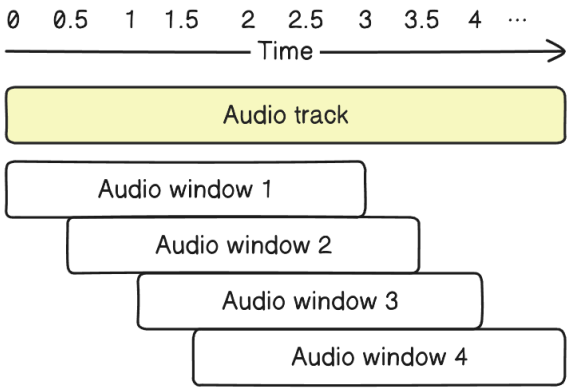

Figure 61: Example of splitting an audio track into overlapping audio windows of 3 seconds

Now that we had all the windows, we were ready for making a prediction and using the results for fusion. Unfortunately, we faced a problem regarding the windows in which the user didn't speak. These windows were useless because they gave no information, and they were always predicted as negative (probably because they were full of low-pitched values typical of negative emotions and coming from

environmental noises). Hence, we decided to discard all the windows in which the user didn't speak. We loaded each window and calculated the energy using *librosa.feature.rms*. We set a threshold for the energy and then iterated over the audio window to understand if the user was talking in the associated time window. We made several tests and found out that 0.2 is a good compromise between recognizing the user's voice and discarding the windows in which they don't talk. This threshold also depends on the microphone used for capturing the voice. if the microphone is of poor quality, it could capture environmental noises that are recognized as user dialogue. Therefore, it must be adapted to the user's settings while making real-life tests. If the energy in a certain location is higher than the threshold, then we assume the user is talking, otherwise we assume he is not. We measured the consecutive energy frames within a window and created sub-windows within the audio windows of 3 seconds. We do it because the user might talk for the first second in a window and then start talking again before the end of the audio window. We then pre-processed all the sub-windows by unifying the ones where the period of silence among them is lower than 0.3 seconds. After that, we simply decided to remove the sub-windows shorter than 1 second. To sum up, we aggregated all the audio frames in which the user was talking, eventually selecting the longest consecutive audio sub-window within the main 3 seconds window. In the end, we collect 2 other parameters which are *longest_segment_start* and *longest_segment_end* which delimit the speaking audio fraction within the window and are later used in the fusion step.

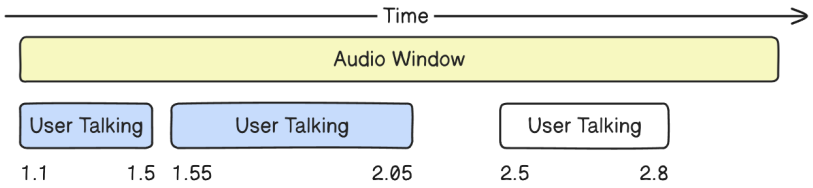

Figure 62: Example of speech detection in an audio window. There might be several sub-windows in which speech is detected. These windows are the relevant parts in which to apply audio emotion recognition.

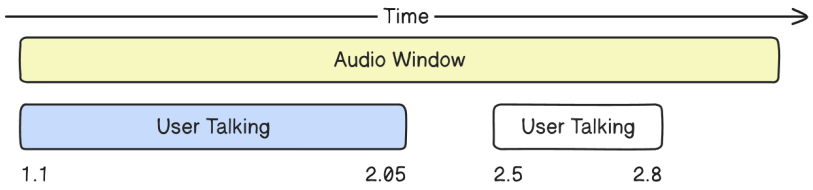

Figure 63: If the time difference between two sequential sub-windows is less than 0.3 seconds, they are combined. This is to avoid having windows that are too short and not relevant.

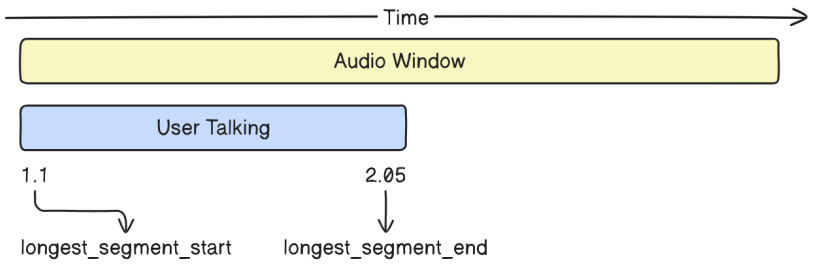

Figure 64: We discard the sub-windows whose duration is less than 1 second. Generally, after this step, we remain with a single sub-window or none. If there is more than one sub-window (even though it's a remote case), we simply select the longest one and discard the others.

At this point, we are only left with the audio windows where the user is speaking and that can give a greater contribution to the emotion classification. The next step was making a prediction by passing the audio windows to the model. So, we loaded the best model and simply applied each window to extract the logits. Logits are used for fusion and indicate the probability of belonging to a certain class. Even though we removed some windows in the previous step, we still may have overlapping windows in the cases in which the user talks between of sequential frames. This feature was useful before prediction, but later we preferred to join them to make the fusion step easier. We iterated over each sequential window and if there was an overlapping we simply joined them only if they had the same predicted label (for example, if two sequential windows were positive/negative we merged them). We modified the window starting time and ending time, based on the values of the merged windows and averaged their logits (this ensured that positive windows are still classified as positive when merged in a larger window). We further made modifications to the remaining windows in order to have non-overlapping audio windows.

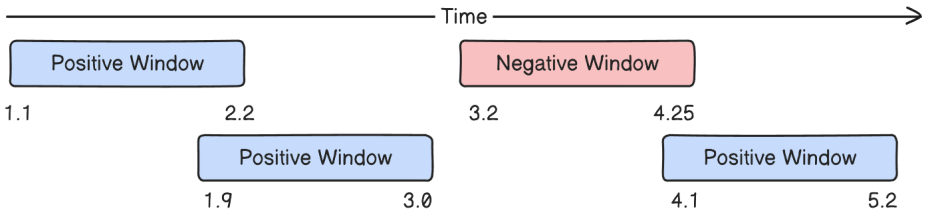

Figure 65: This image shows an example of the overlapping audio windows that we may have after the speech detection process. The resulting windows are applied to the model, getting a prediction for each window.

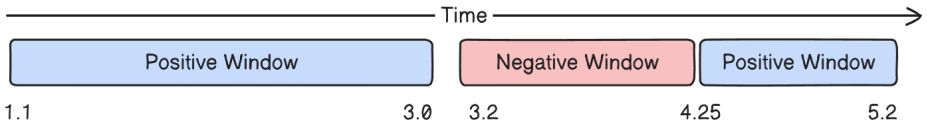

Figure 66: To remove the overlapping and reduce the number of audio windows, we merge the sequential windows with the same prediction and modify the start time and end time of the windows if they have a different prediction. This pre-processing step makes the fusion process easier.

The resulting windows are ready for the fusion process.

5.3 Video Pre-processing

During this process we receive as input the user's video, this must be as long as the audio track that is sent to the audio pre-processing stage (to avoid any problems what we do is to separate audio and video from the same file and send the respective outputs to audio and video pre-processing). From the video file, using a combination of *haarcascades classifiers*, the frames where the user's face is detected are extracted and these are sent directly to the **VideoNet** model, which makes the predictions (*logits*) and adds them to a feature vector. Along with the predictions, for each frame we also add the instant at which it was extracted from the video (*frame_duration*) so that we later match this information with the audio pre-processing feature vector.

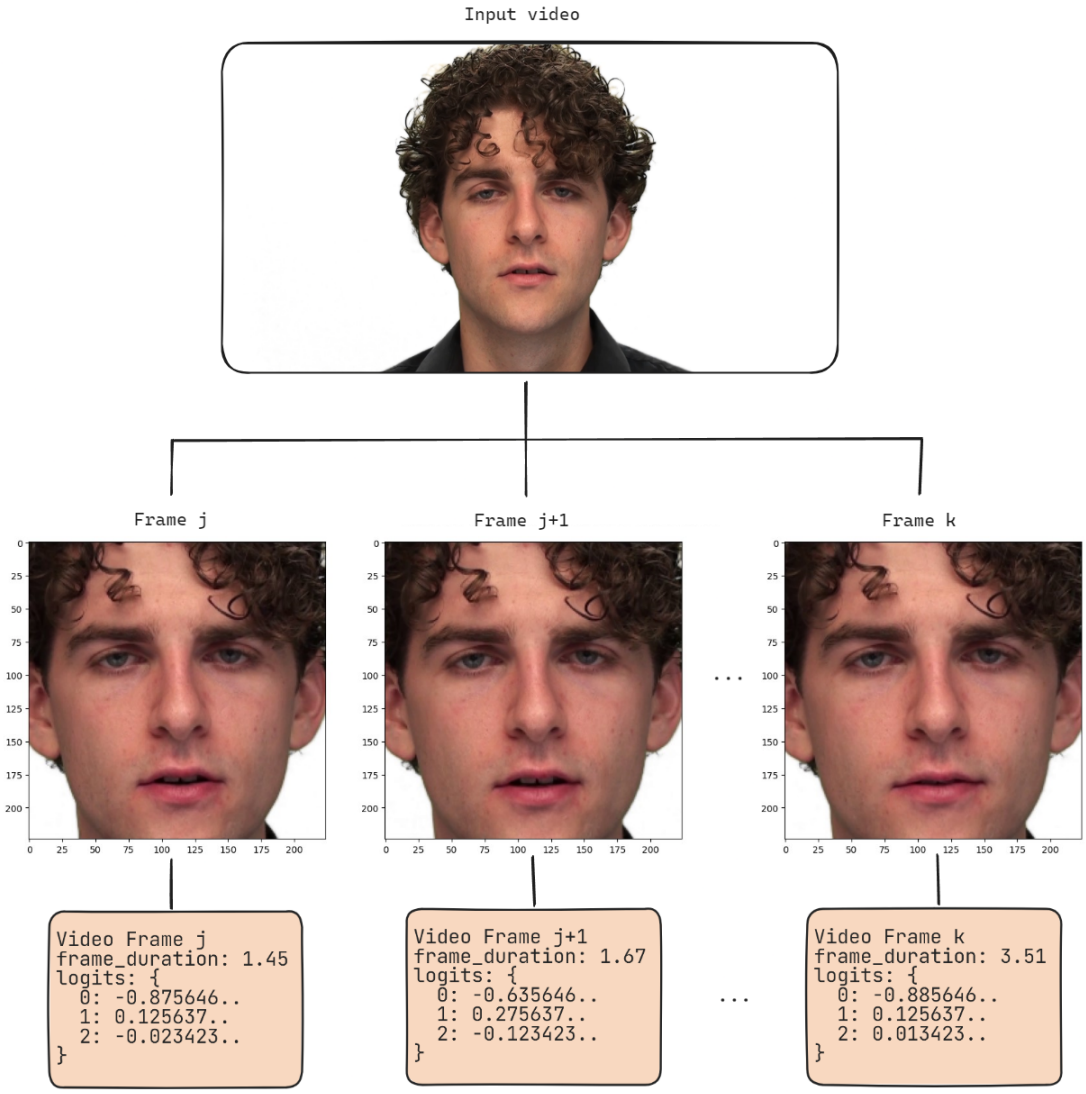

Figure 67: An example of the video preprocessing

Once this is done, we can perform the fusion process.

5.4 PPG Pre-processing

The video that goes in input to the PPG module is first split into one-second-long clips. Each clip is then normalized and the face ROI is extracted. Let's say the video frame rate is 60fps, then the clip will have 60 frames, and so the extracted PPG signal will be a vector of length 60. Since the PPG emotion recognition model is trained with signals whose sample rate is 128hz, we use linear interpolation on the extracted PPG to match the 128hz sample rate, effectively producing a 128-long vector.

After this phase, we have a $(n, 128)$ shaped tensor of extracted PPGs, where n is the number of extracted one-second-long clips. We feed each extracted PPG to the 1D convolutional network that gives us the corresponding emotion label, which will be the user emotion in that one-second interval.

As it already happens with video and audio, timestamps are added to each frame to show in the UI when specific emotions occur.

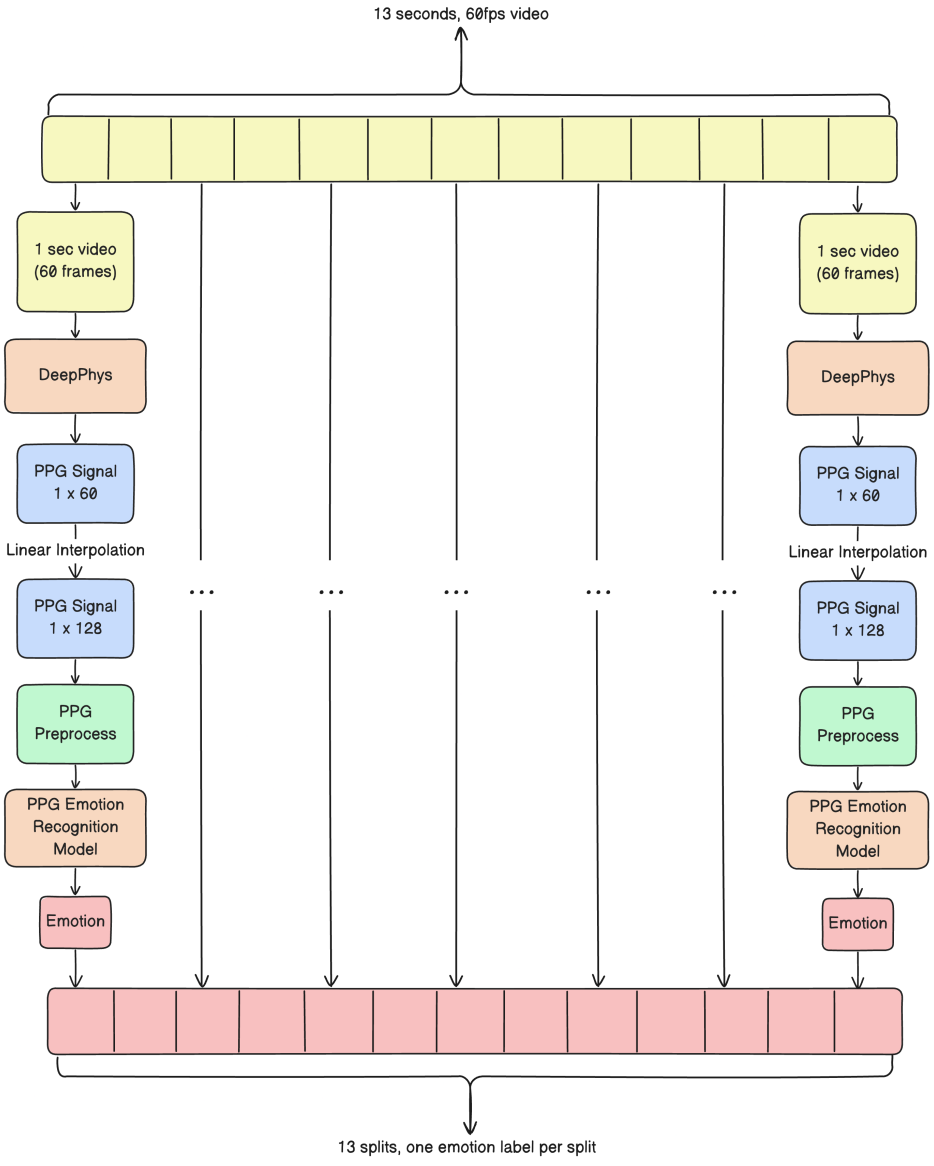

Figure 68: An example of the PPG process in the fusion pipeline

5.5 Models decision fusion

As explained in the previous sections, understanding how to fuse the predictions given by the models wasn't a trivial operation. We may have cases in which the user doesn't speak, fractions of time in which we only have the audio or the face, and cases in which we have both. In general:

- If we don't have audio windows and valid video frames, we simply return an empty list of predictions, since we were unable to detect the user's emotions from the provided data. This is a remote case that typically only happens if the video is corrupted or the user is not in the video.
- If we only have audio windows, we simply consider the emotions that can be extracted from audio. This is the case in which we can't detect the face of the user, but they talk in the background.
- If we only have valid video frames, we only consider the emotions that can be extracted from the user's face. This is the case in which the user doesn't speak during the test.
- If we have both audio and video, we consider them both by making an average of the models' predictions.

The most interesting case where fusion applies is the latest one. There are cases in which we capture some video frames with the face of the user, and they are also talking. This means that we can use the predictions given by the video and audio models and redundantly combine them. Hence, we aim to obtain the user's emotions more accurately. The fusion process works as follows:

- For each audio window (remember that an audio window has the information on when it starts and when it ends):
  - Select all video frames that were captured within the time range of the audio window
  - Average the logits of the video frames
  - Compute the softmax so that each class has a probability in the range [0, 1] and the sum of all probabilities is 1.
- Compute the softmax of the predictions in the audio window
- Compute the average of the audio and video softmax results
- Apply the *argmax* to retrieve the class with the highest probability (this is the final emotion)

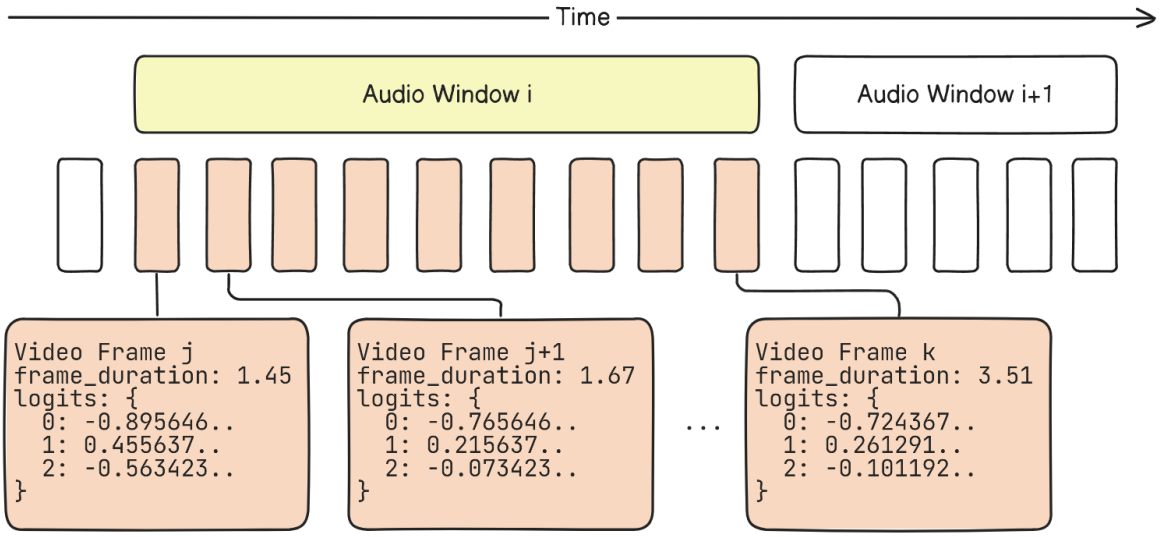

Figure 69: An example of the fusion process within an audio window. We select all the video frames at the same time range as the audio window.

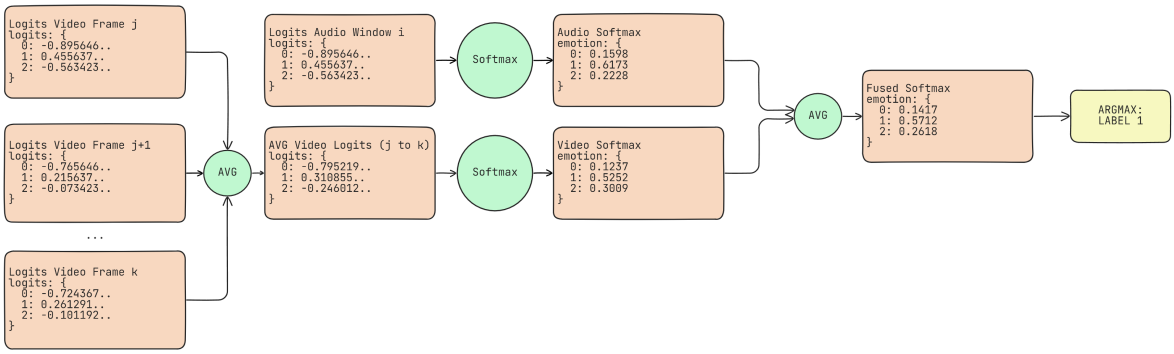

Figure 70: We compute the average of the selected video frame logits, and then we compute the softmax of the result. We then average the result and the softmax of the audio window. Finally, we compute the argmax and get the final prediction for that time window of the test.

This approach allows us to obtain a final predition with a lower degree of uncertainty, given that both models are considered. There might also be a case in which the models return different predictions (e.g. the audio model predicts “negative” while the video model predicts “positive”). Since we compute the average, the model with the least uncertainty will predominate and the final prediction will be the emotion given by that model. If we had only one model, we would not have been able to resolve the uncertainty given by the input and therefore fix the incorrect prediction. This is to say that multimodality allowed us to build a more robust emotion detection system.

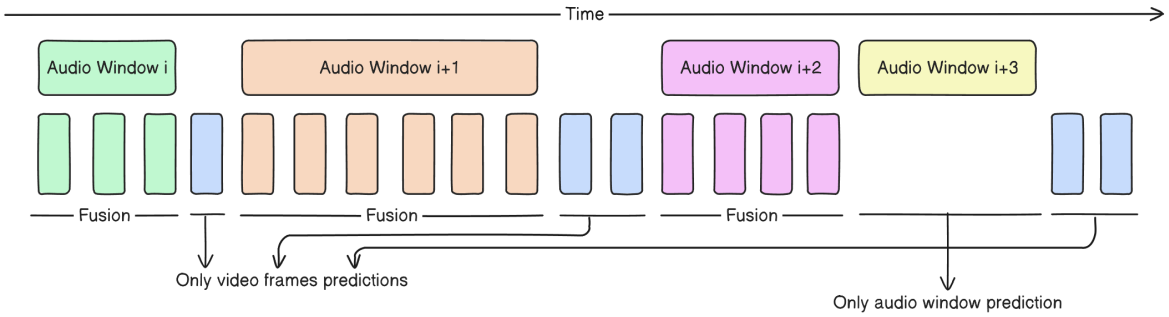

Figure 71: This example shows what a real test could look like. We have cases in which we have both audio windows and video frames captured in it (fusion of results is applied here) and cases in which we have only video frames (with no audio) or audio windows (without no face capturing).

Once the fusion process was implemented, we also added the PPG. We decided to leave predictions of the PPG model separated from the others in order to use it as a confirmation method. PPG can potentially capture emotional changes even when someone is not speaking, while speech analysis relies on vocal cues. Moreover, it might help to detect if the user is faking the facial expression. Think of the case in which the user looks happy because they smile but in reality, they are feeling anxious. This situation cannot be detected using facial expressions but could be detected using PPG. For this reason, the PPG predictions are not fused with the tone of the voice and the facial expressions but are simply presented concurrently.

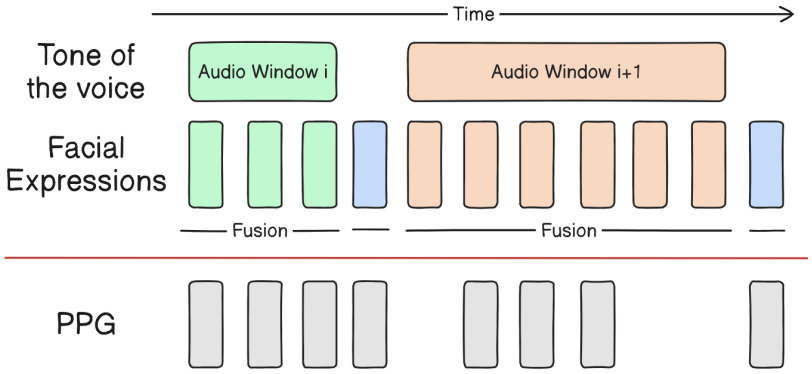

Figure 72: This example is to show that PPG predictions are handled separately and are not involved in the fusion process. Empty spaces mean that PPG predictions are not available in that time window.

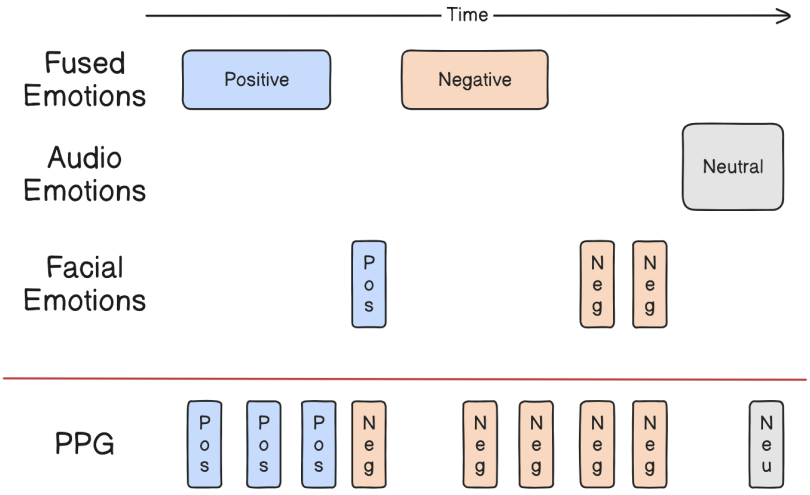

Figure 73: This is a possible real scenario example. There are windows in which we only have audio predictions, cases in which we have only video frames for making facial expression predictions, and cases in which we have both (in the latter case, fusion applies). PPG is handled separately. The fourth window might be a case in which the user is faking the emotion. The facial expression is predicted as positive, but the PPG shows a negative emotion.

6 Demo

Our initial goal was to implement a system to analyze the users' emotions when watching advertisements. We build a small web app to test the models in real life. We used a famous library called **Streamlit** [12] that allowed us to turn data scripts into shareable web apps, in pure Python and with no front-end experience required. We implemented two versions of the web app:

- Online demo: It is a real-life simulation of what an advertisement emotion detection system could look like. The user can click on "start recording" to record a video where they react to an advertisement, and click on "stop recording" to end it. Video frames and the audio tracks captured during the test are processed using the approach described so far and a graph with the results is returned
- Offline demo: Same logic as before with the difference that the user can upload a video that is later processed by the system (so the video is not captured real-life). This approach is useful to make faster tests.

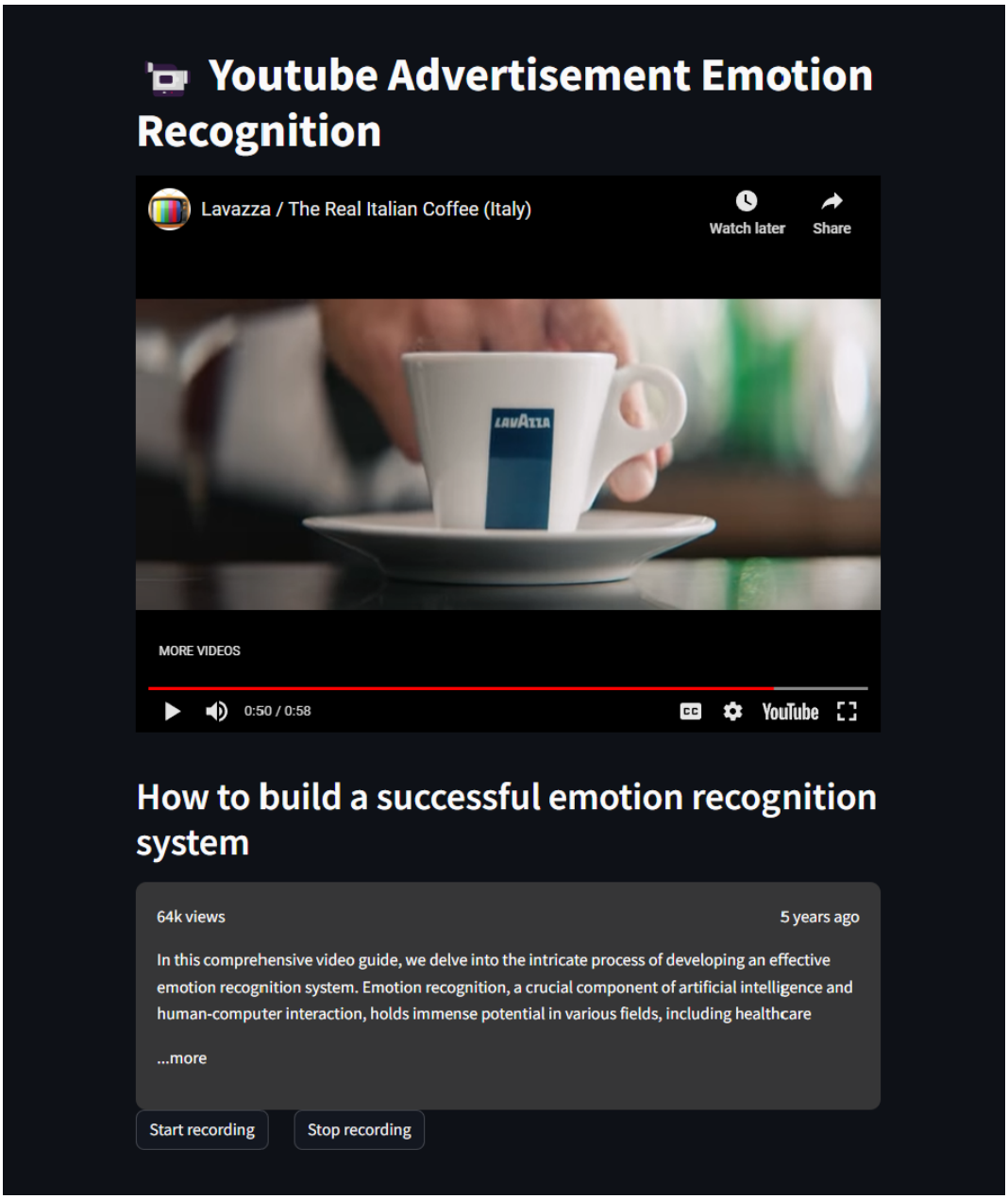

Figure 74: Example of the online demo. This allows to simulate a real-test scenario in which an advertisement pops up while the user is watching another video on Youtube. The user can click on "start recording", do the test, and then click on "end recording" to visualize the results. Of course, this is just a test in a controlled environment. In production, the recording of the user's data should begin automatically as the advertisement starts (or it is about to start) upon their consent.

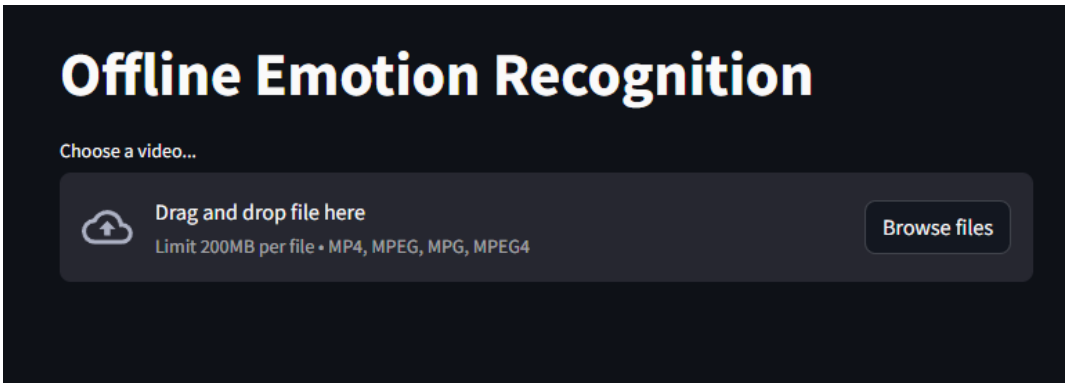

Figure 75: Example of the offline demo. The user can upload a video that is processed and the results are immediately shown. The logic is the same as the online demo but this approach allows us to test every possible video without recording it live.

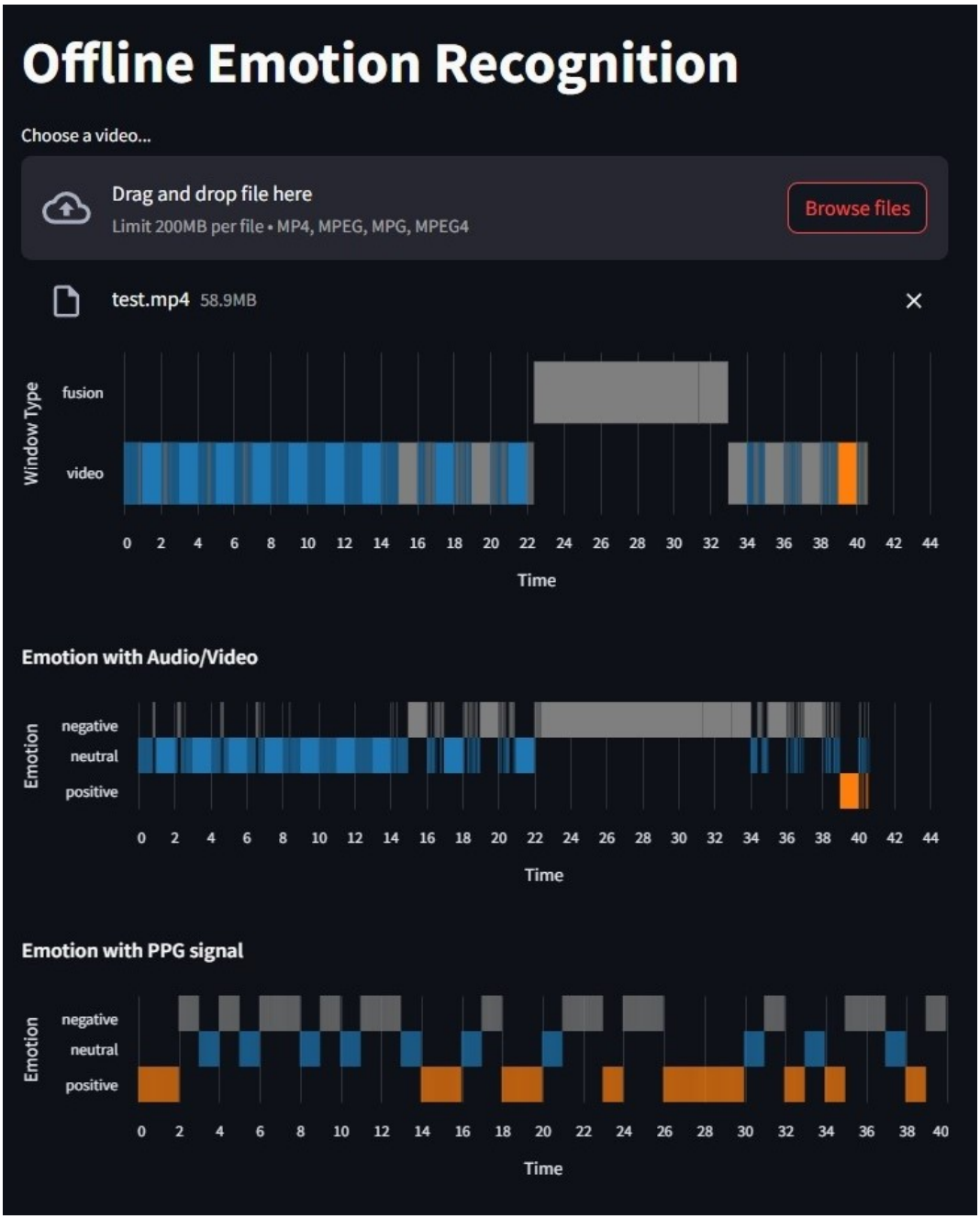

Figure 76: Example of offline emotion recognition

This is an example of offline emotion recognition in which the user can upload a video that is processed and the results of which are shown. The video presents a neutral emotion from the beginning of the video until about second 22, after which the user is disgusted and verbally expresses it until second 34 and remains with a disgusted expression until second 38 where he hints at a micro-smile until the end

of the video. Audio-video processing was able to correctly capture facial and verbal expressions quite well, while regarding the PGG signal the recognition is not consistent, probably due to the combination of low accuracy on the trained model and the noise introduced in the extracted PPG because of a not enough high-quality webcam or bad lighting conditions.

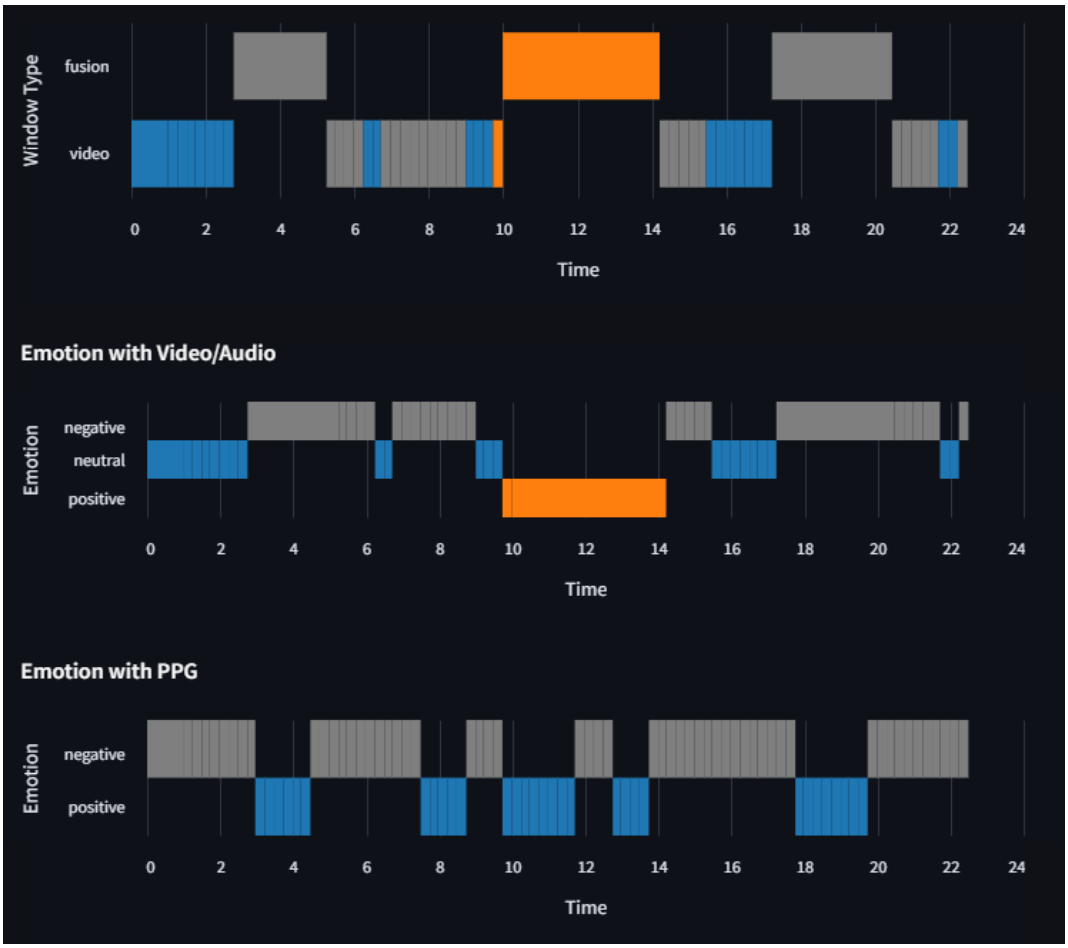

Figure 77: Example of online emotion recognition

This is an example of online emotion recognition. The result has the same structure as before with the online difference that the user must react to the video live as in a real scenario. The user clicks on "start recording", reacts to the video, and then clicks on "stop recording". The captured data is processed and results are shown. In this case, the user has a fused negative reaction, followed by a positive one and a negative again. Some neutral emotions are captured by the video frames in the middle (where the user doesn't talk). The PPG predictions show how the model created has a decent performance. In some cases the results agree with those of the audio and video models while in other cases they are completely different.

A video showing the demo is available at this LINK.

7 Conclusions

Emotion recognition has become increasingly accurate and effective thanks to advancements in artificial intelligence, machine learning, and computer vision. It plays a crucial role in various fields, including human-computer interaction and user experience design, by allowing systems to adapt to users' emotional states for more intuitive and responsive interfaces. This technology is being applied across industries such as customer service, marketing, healthcare, and education, leading to a surge in research and development efforts. Our project aimed to create an emotion recognition system to understand how people react to video advertisements, which could inform decisions on when to interrupt videos or which ads to show based on users' reactions. We created 3 models to extract emotions from the tone of the voice (in case the user speaks during the test), facial expressions, and photoplethysmography using state-of-the-art deep learning architectures. Both audio and video models were trained and tested on the RAVDESS dataset, while we used the DEAP dataset for PPG. To make the system more robust we exploited multimodality: Since the user's data is temporally synchronized (happens in the same time range), we combined the predictions of the audio and video models, getting a more accurate prediction. We decided to not fuse the PPG predictions and left them as a confirmation of the other models' predictions. Finally, we built a demo that allowed us to test the models in a real scenario, reaching our initial goal.

References

[1] A Two-Stage Multimodal Emotion Recognition Model Based on Graph Contrastive Learning (2024) by Wei Ai and FuChen Zhang and Tao Meng and YunTao Shou and HongEn Shao and Keqin Li - https://arxiv.org/abs/2401.01495

[2] Multimodal emotion recognition from expressive faces, body gestures and speech (2024) by George Caridakis, Ginevra Castellano, Loic Kessous, Amaryllis Raouzaiou, Lori Malatesta, Stelios Asteriadis and Kostas Karpouzis - https://doi.org/10.1007/978-0-387-74161-1_41

[3] A Survey of Deep Learning-Based Multimodal Emotion Recognition: Speech, Text, and Face (2023) by Lian, Hailun, Cheng Lu, Sunan Li, Yan Zhao, Chuangao Tang, and Yuan Zong - https://doi.org/10.3390/e25101440

[4] A systematic survey on multimodal emotion recognition using learning algorithms (2022) by Naveed Ahmed, Zaher Al Aghbari, Shini Girija - https://doi.org/10.1016/j.iswa.2022.200171

[5] Recognition of Advertisement Emotions With Application to Computational Advertising (2022) by Abhinav Shukla, Shruti Shriya Gullapuram, Harish Katti, Mohan Kankanhalli and Stefan Winkler - https://doi.org/10.1109/TAFFC.2020.2964549

[6] Convolutional Neural Networks, Explained (2020) by Mayank Mishra, published in Towards Data Science - https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

[7] LSTM Networks — A Detailed Explanation (2020) by Ryan Dolphin, published in Towards Data Science - https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9

[8] How Transformers Work (2019) by Giuliano Giacaglia, published in Towards Data Science - https://towardsdatascience.com/transformers-141e32e69591

[9] The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) (2018) by Steven R. Livingstone and Frank A. Russo - https://doi.org/10.1371%2Fjournal.pone.0196391

[10] Librosa Library -https://librosa.org/doc/latest/index.html

[11] Intuitive understanding of MFCCs (2022) by Emmanual Deruty, published in Medium https://medium.com/@derutycsl/intuitive-understanding-of-mfccs-836d36a1f779

[12] How to Create Additive White Gaussian Noise (AWGN) (2022) by Wave Walker DSP https://www.wavewalkerdsp.com/2022/06/01/how-to-create-additive-white-gaussian-noise-awgn/

[13] Streamlit website https://streamlit.io/

[14] Multimodal Emotion Recognition via Convolutional Neural Networks: Comparison of different strategies on two multimodal datasets (2023) by Bilotti, Bisogni, De Marsico, and Tramonte https://doi.org/10.1016/j.engappai.2023.107708

[15] G-Rex dataset https://www.nature.com/articles/s41597-023-02905-6

[16] CEAP-360VR dataset https://www.dis.cwi.nl/ceap-360vr-dataset/

[17] DEAP dataset https://www.eecs.qmul.ac.uk/mmv/datasets/deap/

[18] rPPG-toolbox GitHub repository https://github.com/ubicomplab/rPPG-Toolbox

[19] DeepPhys: Video-Based Physiological Measurement Using Convolutional Attention Networks https://openaccess.thecvf.com/content_ECCV_2018/papers/Weixuan_Chen_DeepPhys_Video-Based_Physiological_ECCV_2018_paper.pdf

[20] Deep Residual Learning for Image Recognition (Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, 2016) https://arxiv.org/abs/1608.06993

[21] Rethinking the Inception Architecture for Computer Vision (Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, 2015) https://arxiv.org/abs/1512.00567

[22] ViT Base model (Google, 2020) https://huggingface.co/google/vit-base-patch16-224

[23] Fast Emotion Recognition Based on Single Pulse PPG Signal with Convolutional Neural Network https://www.mdpi.com/2076-3417/9/16/3355

[24] 1D Convolutional Autoencoder-Based PPG and GSR Signals for Real-Time Emotion Classification https://ieeexplore.ieee.org/document/9866042