Immersivaudio: audio generation based on video features.

Michele Vitale
ist1111558

Daniele Avolio
ist1111559

Teodor Chakarov
ist1111601

I. Introduction

II. Problem description

III. Proposed solution

IV. Architecture

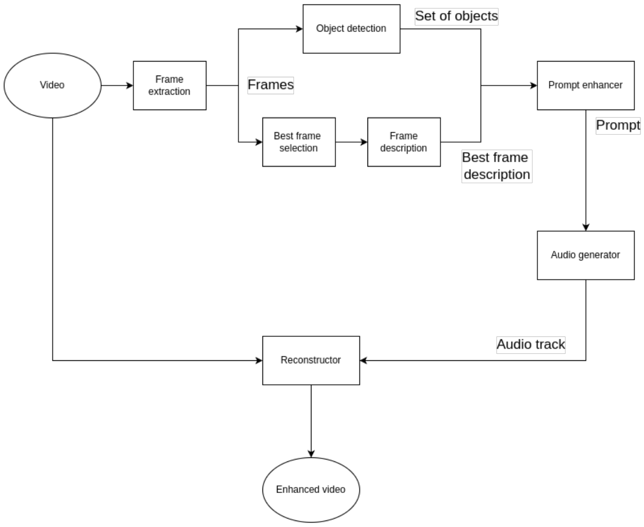

Fig. 1. Architecture of the system.

The complete architecture of the system is shown in Figure 1. The figure shows the architecture of the whole pipeline applied to a video, but other pipelines are provided to adapt to different use cases and input types. The different components of the system are communicating using the JSON format and each module is appending its output rather than reshaping the output to be only the required one from the following step, enabling the process to be modular and adaptable to different scenarios.
Each video is internally identified with an unique token, which is computed at input as the MD5 hash of the video itself. The Input/Output process is made using the Gradio [6] library, which allows the fast development of GUIs, especially for machine learning models. The GUIs are served on a simple web interface.

A. Frame extraction

The frame extraction module takes in input a video and extracts a number of frames. Not every frame in the video is used, because it would be computationally expensive and not necessary since close frames usually have similar features and objects.
The initial idea to sample frames was to fix a specific number for each second of the video, but this approach was discarded due to the increasing amount of frames for longer videos. The final approach used in the project is to sample a number of frames based on a function of the input length video, specifically:

$$n = k \cdot log_{10}(s)$$

with n being the number of frames, s the length of the video in seconds and k a constant factor given in input. This approach was chosen to have a logarithmic growth of the number of frames, which is more computationally efficient and still provides a good representation of the video.

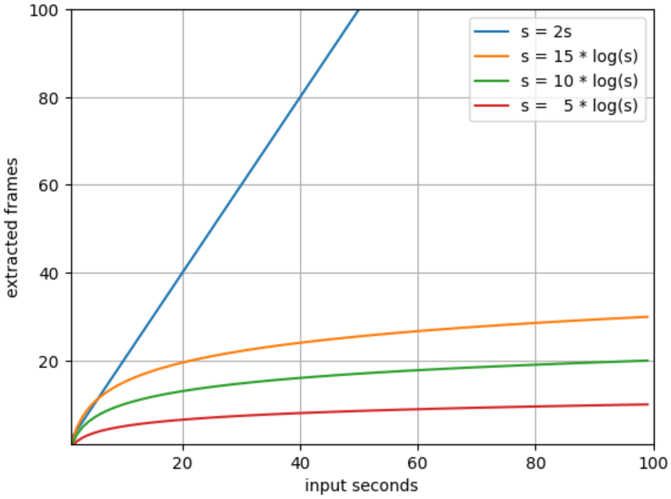

Fig. 2. Comparison of different frame extraction functions.

Figure 2 shows the growth of the number of frames extracted based on the length of the video in seconds, with different values of k, compared to the function $n = 2 * s$ that takes one frame each half second.
In case of image input, this module is not used.
The output of this modue consists in a path to the folder containing the extracted frames and the number of frames extracted.

B. Object detection

The object detection module detects objects in the frames extracted by the previous module. In case of image input, this module performs the object recognition only on the single image. The module uses the YOLOv9 [1] model, which is a state-of-the-art and widely used object detection model that can run in short time and provide good results. The output of this module consists of a list of objects detected in each frame, as well as a set of all objects detected in the batch.

C. Best frame selection

This module is designed to select the best frame from the ones extracted by the first module. The main reason behind this choise is the computational intensity coming from the frame description module, that takes $\sim 5-7$ seconds to process a single frame. Thus, passing all the frames would generate a sensitive delay in the process of each video.

The extraction of frames is done using FFMPEG [3], a versatile and easy CLI tool that is capable of processing audio and video files.

The criterion used to select the best frame has been object of many tests, due to the fact that it is a difficult to generalize task that is highly dependent on the input video. Our final proposed solution picks the best frame using a KMedoid algorithm over a latent representation of the images. The latent representation is obtained by passing the image through ResNet18 [4], which is a widely used and efficient image classification model. Since the task is not classification, the ResNet module is used to only get the latent representation of the input, of size $512 \times 16 \times 29$. The said representation is then reshaped and used to cluster the video and calculate the medoid. In particular, the KMedoid algorithm is used with $k = 1$ to get the medoid of the complete.

The output of this module is the path and index of the selected frame.

D. Frame description

For this crucial task different options of image to text models were evaluated and tested. The main source used for models is the Hugginface text-to-image section [7]. In the end, the selected model for this task is MoonDream2 [5], a image-text to text model that is able to produce the answer to a question over the image. In our case, the used question is *"Describe shortly this image considering it will be used as input to a sound generation model"* that, over different tests, provides good results combined with the following modules. The choise switched to an image-text to text model since it provides a wider inference than image to text models, that can be used to condition the produced description to better fit our goals.

The output of this module is the description of the input image.

E. Prompt enhancer

The prompt enhancer module is only composed of a single string interpolation that combines the outputs of both the frame description and the object detection modules. An initial option was to combine them using a lightweight Large Language Model, but we decided not to include another model to avoid the increase in computational cost. The output of this module is the composed prompt, that will be fed in input at the audio generation module.

F. Audio generation

G. Reconstructor

V. Deployment

The deployment of the system is done using Python 3.10.12 on Google Colab [2]. The first intention was to whole process run on a local machine, but it quickly became clear that this is not possible due to the high hardware requirements that audio generation models have. The next idea was to serve the audio generation module over an API, loading it on the machine provided at IST. This hypotesis got aswell discarded for lack of sufficient resources.

The Gradio interface is accessed through a public link, which is generated by the library itself. The link is accessible by anyone and can be used to interact with the system.

The informations about the used libraries and technologies can be found in the **Architecture** section of this report.

VI. Implemented features

VII. Evaluation

VIII. Conclusions and future work

References

[1] Yolov9 documentation and Yolov9 paper reference.

[2] Google Colab main page.

[3] FFMPEG website.

[4] ResNet18 implementation on PyTorch.

[5] MoonDream2 model card on HugginFace.

[6] Gradio library main page.

[7] HuggingFace image-to-text section.