University of Hamburg

LT Research Group

Prof. Chris Biemann

Video-Text Retrieval Tool

Author:
Narges Babaahmadi

July 2023

Contents

1 Introduction

The objective of this project is to develop a proof-of-concept (PoC) application for Video-Text Retrieval (Video-IR). The aim is to build a tool that can retrieve relevant **video scenes** based on **user input text queries**. In this project, the MSR-VTT dataset was used for the retrieval operation.

MSR-VTT dataset is a large-scale dataset for open-domain video captioning and video analysis. It consists of 10,000 video clips from 20 diverse categories such as sports, music, and animals. Each video is accompanied by 20 human-annotated sentence descriptions, providing multiple viewpoints and descriptions of the visual content.

The dataset is split into a training set of 6,513 videos, a validation set of 497 videos, and a test set of 2,990 videos. The videos vary in length from 10 seconds to 30 seconds, and have a resolution of 360 x 240 pixels. The sentence descriptions are provided in English and have an average length of 10 words.

One notable aspect of the project is the use of zero-shot learning. Instead of relying on captions, the retrieval model is trained on a combination of visual and textual information, leveraging the Xclip model. **Xclip** is a powerful pre-trained model that can encode both images and text into a joint latent space, allowing for seamless multimodal retrieval. It has two decoders, one for text and one for video, which makes it particularly well-suited for video-text retrieval tasks.

In this project, different variants of Xclip were used to explore the impact of different model architectures, as well as to compare the performance of different variants of the model. The Used variants are "microsoft/xclip-base-patch32," "microsoft/xclip-base-patch16," and "microsoft/xclip-large-patch14."

Xclip model uses a technique called patching to divide an input image into a grid of smaller rectangular regions, or patches. The model then **extracts visual features** from each patch using a **convolutional neural network (CNN)** and combines them into a joint representation with the textual features extracted from the input text. In the context of the Xclip model, **"patches"** refer to the spatial regions of the input image that the **CNN** model uses to extract visual features.

Three variants of Xclip model - "microsoft/xclip-base-patch32", "microsoft/xclip-base-patch16", and "microsoft/xclip-large-patch14" - differ in architecture, number of patches, and training data.

"microsoft/xclip-base-patch32" and "microsoft/xclip-base-patch16" have a base architecture, with 32 and 16 patches respectively, and were trained on a combination of OpenImagesV6 and COCO datasets, containing a total of around 19 million images and 1.2 million captions.

On the other hand, "microsoft/xclip-large-patch14" has a more complex architecture, with 14 patches, and was trained on a larger dataset that includes OpenImagesV6, COCO, and Conceptual Captions datasets, containing a total of around 45 million images and 3.3 million captions.

To perform the retrieval operation, the project is divided into five components:

Component 1: Video Scene Segmentation
This component involves segmenting input videos into smaller scenes, which can then be analyzed and indexed for retrieval.

Component 2: XCLIP Embedding Computation
This component involves computing the Xclip embeddings for each scene. Xclip is a powerful pre-trained model that can encode both images and text into a joint latent space, allowing for seamless multimodal retrieval.

Component 3: FAISS Indexing
This component involves persisting the scene embeddings into a FAISS index. FAISS is an efficient similarity search library that can be used to retrieve similar embeddings based on cosine similarity.

Component 4: Scene Retrieval Program
This component involves implementing the retrieval program, which takes a user's text query as input, retrieves the relevant scene embeddings from the FAISS index, and returns the top-scoring scenes.

Component 5: User Interface
This component involves building the user interface, which provides an easy-to-use interface for users to input their queries and visualize the retrieved scenes.

This report provides a detailed explanation of each of these components, including UI screenshots, explanations of the underlying algorithms and the powerful **CNN** image processing technique used by **Xclip**.

2 Main Steps

2.1 Cutting Videos into Scenes

The first step is to segment the videos from the MSR-VTT dataset into scenes. The following sub-steps are undertaken to accomplish this:

- **Scene detection using TransNetV2:** TransNetV2, an open-source scene segmentation tool, is utilized to detect scenes within the videos. TransNetV2 employs a deep neural network architecture to identify scene boundaries accurately. (https://github.com/soCzech/TransNetV2)
- **Saving scenes using ffmpeg:** To save each detected scene as an individual video file, the code employs the ffmpeg library. The **trim** function is responsible for cutting the scenes from the original video and saving them as separate files. It takes as input the input file path, output file path, start and end frames of the scene, and utilizes ffmpeg to perform the trimming operation.
- **Storing scene metadata:** Instead of using a pandas DataFrame or a dedicated data structure, the code employs a straightforward and efficient approach to store scene metadata. It generates filenames that encode relevant information about each scene, such as the frame range and the video identifier. For example, the filename "4-10-vid10" represents frames 4 to 10 of video 10. This filename-based approach simplifies the storage of scene metadata and facilitates easy retrieval of scenes based on the filename information.
- **Iterating Over Videos and Scenes:** The code iterates over the video files in the specified video repository directory. For each video file, it applies the TransNetV2 model to predict the video frames, single frame predictions, and all frame predictions. These predictions are then processed to identify individual scenes using the **predictions-to-scenes** function.
- **Cutting and Saving Scenes:** Once the scenes are identified, the code proceeds to cut each scene from the original video file. It calls the **trim** function to trim the video frames within the scene's start and end frames, and saves the resulting scene as a separate video file. The save path for each scene is generated based on the scene's frame range and the video's identifier.
- **Error Handling:** The code includes error handling to catch any exceptions that may occur during the scene cutting and saving process. If an error occurs, the code writes the path of the video file to an error log file for later reference.

By implementing this code, the videos from the MSR-VTT dataset are efficiently processed and segmented into individual scenes. The TransNetV2 model, in conjunction with the ffmpeg library, allows accurate detection and extraction of scenes, ensuring that each scene is saved as an independent video file.

In conclusion, this code effectively automates the process of cutting videos into scenes. The integration of the TransNetV2 scene detection model and the ffmpeg library provides a reliable and efficient solution for scene segmentation. The generated scene metadata filenames allow for easy organization and retrieval of the scenes based on the identified frame ranges and video identifiers.

2.2 Computing XCLIP Embeddings for Scenes

In this step, the focus is on computing XCLIP embeddings for each scene, enabling efficient similarity matching. The code implementation incorporates PyAV for video decoding and the Transformers library for loading and utilizing the XCLIP model. The resulting embeddings are then stored for subsequent retrieval and similarity analysis.

The process of computing XCLIP embeddings for scenes involves several key steps:

- **Reading and decoding the video frames:** The code employs the **read-video-pyav** function, which utilizes the PyAV library, to read and decode video frames from the scene. This function takes a PyAV container object and a list of frame indices to decode. It then returns the decoded frames as a numpy array. By decoding only the selected frames specified by the indices, the code reduces computational requirements and focuses on relevant frames for embedding calculation.
- **Sampling frame indices:** To create shorter clips for embedding calculation, the **sample-frame-indices** function randomly selects frame indices from the video. This function considers parameters such as clip length, frame sample rate, and segment length to determine the indices to sample. By randomly selecting frames within the specified parameters, the code ensures diversity in the sampled frames while keeping the computation manageable.
- **Loading the XCLIP model:** XCLIP model, specified by the **model-name** parameter, is loaded using the AutoModel class from the Transformers library. Preprocessing the video frames: The frames are processed using the AutoProcessor from the Transformers library. This step prepares the frames for input to the XCLIP model.

- **Preprocessing Video Frames:** Before passing the frames to the XCLIP model, they undergo preprocessing using the AutoProcessor from the Transformers library. This preprocessing step prepares the frames to meet the input requirements of the XCLIP model. By ensuring that the frames are properly processed, the code maximizes the accuracy and reliability of the resulting scene embeddings.
- **Calculating video features:** Preprocessed frames are then fed into the loaded XCLIP model, and the video features are computed using the get-video-features method. This step leverages the power of the XCLIP model to extract informative and discriminative features from the video frames. The resulting video features capture crucial visual and semantic information, enabling effective scene representation.
- **Saving the scene embeddings:** The computed scene embeddings, represented as numpy arrays, are saved to disk for future retrieval and similarity matching. Each scene's embedding is stored as a separate file, utilizing a consistent naming convention to associate the embeddings with their corresponding scenes. This storage scheme ensures easy access and retrieval of embeddings during the similarity analysis phase.

By employing the PyAV library for video decoding, the Transformers library for model loading and inference, and efficient storage mechanisms, the code successfully computes XCLIP embeddings for scenes. These embeddings provide compact representations of scenes, enabling efficient comparison and similarity matching between scenes.

In conclusion, the code implementation efficiently calculates XCLIP embeddings for scenes by leveraging video decoding capabilities, the powerful XCLIP model, and the numpy array representation. This step sets the foundation for the subsequent similarity analysis phase, facilitating the efficient retrieval and matching of scenes based on their computed embeddings.

2.3 Persisting scene embeddings with FAISS

After computing the scene embeddings, the next step is to persist them using the **FAISS** library. FAISS is a powerful library for efficient similarity search and indexing. It allows for fast retrieval of similar embeddings based on distance metrics.

The specific implementation in our project utilizes the **autofaiss** library, which is a wrapper around FAISS providing convenient functions for index building and management.

The **build-index** function from autofaiss is used to create the FAISS index. It takes the embeddings path, index folder path, and other optional parameters as input.

The **build-index** function performs the following tasks:

- It builds the FAISS index by specifying the embeddings path and the desired location for storing the index files.
- The index is created with the specified parameters, such as maximum memory usage and available memory.
- Scene embeddings are added to the index, creating an efficient data structure for similarity matching.

By utilizing autofaiss and FAISS, we ensure that the scene embeddings are indexed in a way that allows for efficient retrieval and similarity search operations. This significantly speeds up the scene retrieval process in the Video-IR tool, enabling quick and accurate matching of user queries with relevant video scenes.

2.4 Implementing Scene Retrieval Program

Scene retrieval program is responsible for retrieving the most relevant scene(s) based on user input text queries. The following sub-steps are carried out:

- Query processing function: A function is implemented to receive a query string and an integer parameter, k. This function outputs the filenames of the top-k similar scenes and their respective similarity scores.
- Loading models and data: The XCLIP model, FAISS index with scene embeddings, and the required metadata are loaded into memory for efficient retrieval and processing of scenes. In this implementation, both the videos and queries are encoded using the same XCLIP model, ensuring consistency in the encoding process.

- Similarity search: Using the encoded query string, a similarity search is performed using the FAISS index. The search function utilizes the index to find the top-k scenes that are most similar to the query embedding. The FAISS index efficiently calculates the distances between the query embedding and the scene embeddings, enabling fast retrieval of similar scenes.
- Retrieving scene metadata: The filenames obtained from the similarity search are used to retrieve the respective scenes based on the previously defined naming convention. By parsing the filenames, the scene metadata, including the original video, start and end times, can be retrieved. This metadata provides valuable information about the scenes and aids in understanding the context of the retrieved results.
- Displaying results: The metadata of the scenes, such as the original video and start and end times are shown to provide an overview of the retrieved scenes. This allows users to quickly assess the relevance and similarity of the retrieved scenes to their query.
- Returning scene filenames: The filenames of the scenes that match the query are returned as the output of the function. This enables further processing or presentation of the retrieved scenes based on the specific requirements of the Video-IR tool.

By implementing these steps, the scene retrieval program effectively matches user text queries with the most relevant video scenes, providing users with an efficient and user-friendly way to explore and retrieve video content based on their interests and needs.

2.5 Building the User Interface

The final step involves creating a user interface (UI) for the program to enable easy interaction with the Video-IR tool. This section outlines the details of the UI development using the Gradio framework and highlights its components and functionality.

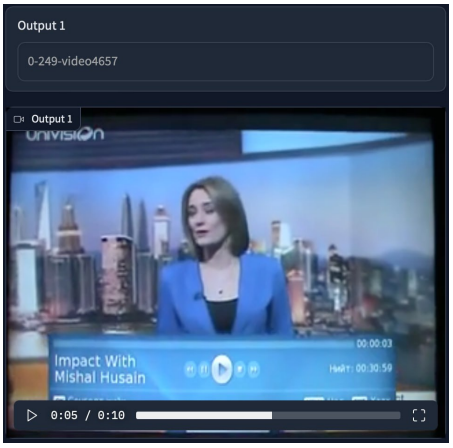
- **UI framework selection:** To develop the UI, we selected the Gradio framework, a powerful tool for creating interactive interfaces for machine learning models and applications. Gradio simplifies the process of integrating user-friendly input and output components into the Video-IR tool, enhancing usability and accessibility.
- **UI components:** They were carefully designed to provide an intuitive and user-friendly experience. The following key elements were incorporated:

– **Search Bar:** The main input field of the UI, where users can enter their text queries. Once the query is entered, users can press the ENTER key to initiate the scene retrieval process.

– **Model Selection Dropdown:** A dropdown menu that allows users to select the desired video retrieval model. The available options are "xclip-base-patch32," "xclip-base-patch16," and "xclip-large-patch14."

– **Output Display:** The UI provides multiple output components to present the retrieved scenes in an organized manner. For each retrieved scene, the following information is displayed:

  * **Text Output:** A textbox that shows relevant information about the scene, such as the video title and start and end times.
  * **Video Output:** A video player that plays the retrieved scene, allowing users to visualize and analyze the content.



- **Background processing:** To ensure a responsive user experience, the Video-IR tool runs continuously in the background, ready to handle user queries. Gradio handles the background processing efficiently, ensuring that the tool remains responsive and performs scene retrieval quickly and accurately.
- **Displaying results:** The UI presents the results of the scene retrieval process to users in a clear and organized manner. For each scene retrieved, the relevant metadata, such as the original video, start and end times, and similarity scores, are displayed. This presentation format enables users to easily evaluate and explore the retrieved scenes based on their search queries.

2.6 Integrating the UI with the Video Retrieval Code

The UI is seamlessly integrated with the video retrieval code, which includes components such as the text embedding, similarity search, and video path extraction. When a user enters a text query and selects a model, the UI triggers the underlying video retrieval functions, enabling the search for relevant scenes. The retrieved scenes are then displayed to the user through the UI's output components.

By developing this intuitive and user-friendly UI with Gradio, we have successfully created a Video-IR tool that enables users to interactively search and explore relevant scenes within video content. The combination of a powerful backend and an intuitive frontend ensures a seamless user experience, making the tool accessible to both technical and non-technical users.

In conclusion, the UI development phase enhances the usability and accessibility of the Video-IR tool, enabling users to easily interact with the system and obtain meaningful results from their text queries. The incorporation of Gradio framework and its interactive components provides a visually appealing and user-friendly interface, contributing to an enhanced overall user experience.

3 Conclusion:

In conclusion, this report has provided a detailed explanation of the step-by-step process involved in developing a Video-Text Retrieval tool using the MSR-VTT dataset. The tool effectively segments videos into scenes, computes XCLIP embeddings, implements a scene retrieval program based on user text queries, and builds a user-friendly interface using the Gradio framework.

This comprehensive solution enables efficient video retrieval based on textual information and offers diverse applications across various domains. For instance, in the entertainment industry, the tool can be used to quickly search for specific video scenes or clips based on user text queries. In the education sector, the tool can be used to create more engaging and interactive learning materials by adding relevant video clips to text-based content. In the field of video surveillance, the tool can be used to quickly search for relevant video footage based on textual information, which can be particularly useful for law enforcement agencies.

The developed tool can be further improved by incorporating advanced techniques such as deep learning-based feature extraction and incorporating more sophisticated text processing methods. Additionally, the tool can be extended to support larger datasets and more complex retrieval scenarios, such as video summarization and event detection.

Overall, this project demonstrates the feasibility of Video-IR and highlights its potential for real-world applications. The developed tool offers a customizable and scalable solution for video retrieval based on textual information and provides a foundation for future research and development in this exciting field.