CS230

Searching across Video Frames: Video-Text Matching Using Deep Learning Category: Computer Vision

Kunal Sinha
SUNet ID : ksinha2
ksinha2@stanford.edu

Sunny Junyang Sun
SUNet ID : sunnysun
sunnysun@stanford.edu

Laura Wu
SUNet ID : laurawjr
laurawjr@stanford.edu

GitHub link to code

Abstract

This project aims to train a deep learning model that searches for short videos that match a user's textual description. The baseline approach relies upon the pretrained Contrastive Language-Image Pre-training (CLIP) model, while further approaches involve extracting keyframes from videos using hierarchical clustering and finetuning CLIP. The performance of these models are evaluated on an accuracy-based metric, and the results indicate that keyframe extraction improves the accuracy of the baseline model, whereas finetuning actually brings down accuracy as a result of potential underfitting.

1 Introduction

In recent years, the concept of video retrieval has rapidly become more relevant both from the expansion of available video databases and the wide range of potential applications such as video copyright enforcement, video summary, and so on. The goal of this project is to train a deep learning model that allows users to search for short video snippets based on a description of an object or a scene. The model takes in textual descriptions as input and predicts a specific video snippet as output.

Our baseline approach uses the pretrained Contrastive Language-Image Pre-training (CLIP) model, which consists of an image encoder and a text encoder. The baseline model feeds a user's text description along with each video frame into the encoders; the model predicts a specific video based on which set of frames are most similar to the text (measured by cosine similarity between the embeddings). We pursue two modifications to the baseline: the first involves extracting keyframes that summarize the information in each video more accurately; the second involves finetuning CLIP to optimize for this task of video search.

This type of model can greatly improve user experience when searching for a particular GIF or YouTube video on the internet, especially when the content has not been named or tagged correctly. Furthermore, the model can be extended to help users search for relevant scenes or snapshots within a single large video such as a movie (without having to look through each scene manually).

2 Related Work

We found our problem to be in parallel to ad-hoc video search (AVS), which aims to search for unlabeled short videos through a natural-language text. The mainstream approaches to this problem can roughly be divided into two categories, concept-based and concept-free. We will review five specific algorithms, the last of which will become the foundation for this paper.

The first such algorithm W2VV+ adopts a non-concept-based method and concatenates the outputs of three text encoders into a single vector[1]. This vector and the video features are then projected in a common latent space by a multilayer perceptron and a fully connected (FC) layer respectively.

DE uses a multi-level encoding network-integrated from a combination of mean feature pooling, GRU, and 1DCNN. Instead of only taking video-level features as the W2VV++, DE goes a step further to accepting frame-level features[2].

The third algorithm Sentence Encoder Assembly (SEA) supports the matching of text in multiple encoder-specific common spaces instead of just one and produces an output by averaging the computations derived from each individual space. It exploits multi-space multi-loss learning to better capture the complementarities among the individual common spaces[3].

The Multi-modal Transformer (MMT) algorithm capitalizes upon the multimodal and temporal nature of videos and aggregates these features in a compact representation. It relies on Bidirectional Encoder Representations from Transformer (BERT) for text-encoding[4].

Finally, CLIP, the baseline model for this paper, builds upon zero-shot transfer, natural language supervision, and multimodal learning. By generalizing to unseen object categories in image classification, CLIP is able to perform unseen tasks. It learns a multi-modal embedding space by jointly training an image encoder and text encoder with cosine similarity and relies on the symmetric cross entropy loss over these similar scores as a metric. CLIP makes use of the ResNet-50 and Vision Transformer (ViT) as foundational architectures[5].

3 Datasets

We are currently using the Microsoft Research Video Description Corpus (MSVD). First collected in 2010, this dataset contains 85K English descriptions for 2,080 video clips. These descriptions were written by human volunteers, meaning they likely mimic the type of queries a user would provide when searching for particular video footage. The short videos are about 10 seconds long and contain a wide variety of objects and environments, making them an ideal dataset for training and testing [6].

```
-4wsuPCjDBc_5_15 the squirrel ate the peanut out of the shell
-4wsuPCjDBc_5_15 the squirrel is eating
-7KMZQEsJW4_205_208 a man demonstrating how to clean a flower
-7KMZQEsJW4_205_208 a man brushes off the roots of a sunflower
```

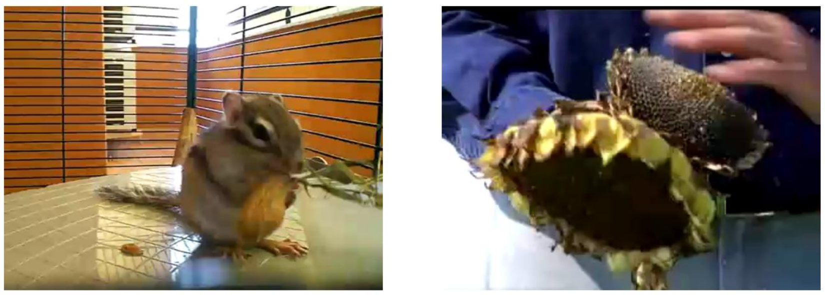

Figure 1: Example frames from the two videos in the textual descriptions

Above is an example of the textual annotations for the videos accordingly. Each video will have multiple textual descriptions that describe the main content of the image.

During the finetuning process, we deployed an $80\%, 10\%, 10\%$ split for training, development, and testing dataset.

4 Model Description

4.1 Baseline Model

To predict which short video matches the text most closely, our baseline model matches the frames of each video to the text using CLIP's pretrained encoders. The model first breaks down a video into a series of frames. Because saving each frame in each video consumes too much time and memory, we include a parameter called "save_fps"that specifies how many frames to save per second of video footage. (For example, if save_fps=1, and the frame rate of the videos is 30 frames per second, then we only save frame 0, frame 30, frame 60, etc. We settled on a save_fps value of 1 after some brief hyperparameter tuning).

The model then uses CLIP's image encoder to compute an image embedding for every frame that we have saved. When a user submits a text description, the model feeds the description into CLIP's text encoder to produce a text embedding for it. Finally, the model computes the cosine similarity of the text embedding to each image embedding, selecting the top k images with the highest similarity scores. The video that a majority of these frames belong to becomes the predicted video.

CLIP's encoders were trained to learn a multi-modal embedding space that images and text could be mapped onto. To do so, the authors jointly trained the image and text encoder on a set of 400 million image-text pairs taken from the internet. The objective was to produce embeddings that maximized the cosine similarity of positive examples (where the text accurately described the image) and minimized that of negative examples (where the text did not).

4.2 Keyframe Extraction

In the baseline model, predicting the top k frames requires iterating over almost every frame saved, which can be a time-consuming process. Furthermore, most of the frames in a video were highly similar to one another. We thus use a keyframe extraction algorithm that can locate a few "keyframes" that summarize the information (i.e the relevant objects and actions) in each video. By deleting the rest of the frames, we aim to save time without sacrificing substantial accuracy.

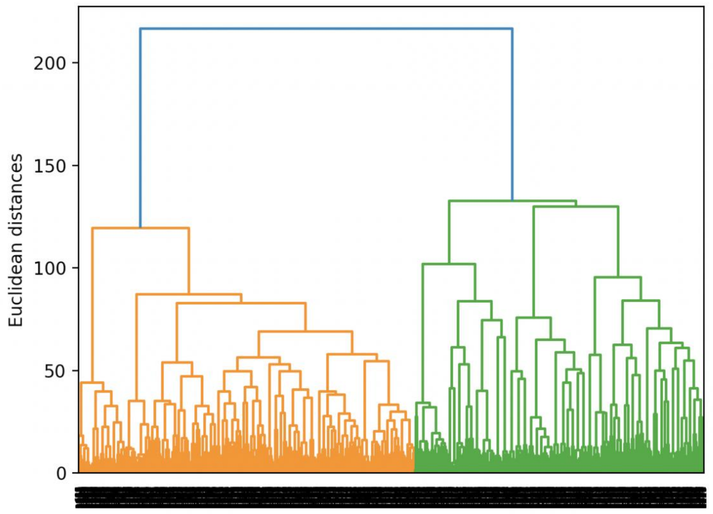

Figure 2: A dentrogram that visualizes five example data points by calculating the euclidean distance.

We employ a hierarchical clustering algorithm to identify groups of similar frames. After using CLIP's image encoder to produce feature vectors from each image, we feed the list of feature vectors into an agglomerative clustering function from sklearn. This algorithm recursively groups pairs of clusters by computing the euclidean distance between the image feature vectors. Figure 2 visualizes the hierarchical clustering process.

The algorithm requires either $n\_clusters$ or $distance\_threshold$ to group the image feature vectors together. $n\_clusters$ represents the number of clusters that are expected to form whereas $distance\_threshold$ represents the euclidean distance threshold above which clusters will not be merged together. Because each short video has a different length and contains varying levels of diversity between frames (e.g, some videos have more complex movement than others), we found it difficult to predict the correct number of keyframes in advance; as such, we set a distance threshold. We tuned this threshold value as a hyperparamter, settling on a value of 0.05.

After assigning all frames to a cluster, we randomly sample one frame uniformly from each cluster to serve as a keyframe. The algorithm returns a list of keyframe indices; all non-keyframes are deleted before the model begins computing its predictions. Although keyframe extraction is initially slower than the baseline approach due to the overhead of running hierarchical clustering and editing the dataset, this method is faster in the long-run. Once the frames have been saved, the model needs to search fewer frames to make a prediction.

4.3 Finetuning

We then decided to finetune CLIP's encoders for two reasons. First, video search datasets consisting of short video clips with labeled frames are relatively small, meaning they are insufficient to train a high quality model. As such, we sought to use transfer learning to take advantage of the large datasets that CLIP was trained on (consisting of millions of image-text pairs). Second, because CLIP's pretrained encoders were trained on a wide variety of images taken from the internet, they are not specifically optimized for the domain of video frames. Frames likely differ from typical images on the internet: for example, they are often blurrier and depict people or objects in the middle of some sort of motion.

The CLIP text encoder has the architecture of a Transformer, while the image encoder has that of the Vision Transformer described by Dosovitskiy et al [7]. We froze the architecture of both networks except the last 10% of layers, which corresponded to the final few Residual Attention Blocks of the encoders. (The percentage of layers to unfreeze was a hyperparameter that we tuned to maximize performance and minimize time taken during training). We kept the architecture of these blocks the same, because our outputs were the same shape (i.e embedding vectors of length 512).

Because OpenAI did not provide model training code, we wrote our own script to jointly train both encoders. In each forward pass, we fed a batch of 32 image-text pairs into the model and predicted the cosine similarity of each image to each text. The groundtruth was 1 for image-text pairs that matched and 0 for those that did not. We then computed computed Cross Entropy Loss to compare the cosine similarities to the groundtruth.

$$Cross - Entropy = L\left(\mathbf{y}, \hat{\mathbf{y}}\right) = -\sum_i \mathbf{y}_i \ \ln \hat{\mathbf{y}}_i$$

We performed Adam Optimization with a learning rate of 5×10^{-5}, and with $\beta_1 = 0.9$ and $\beta_2 = 0.98$. Finally we set a weight decay regularization parameter of 0.2. This training strategy (along with these choices of hyperparameters), were adapted from the original CLIP paper [5]. The model was finetuned for 40 epochs on frames taken from 800 video-text pairs in the MVSD dataset.

5 Results/Discussions

5.1 Evaluation Metrics

Accuracy is our primary metric for measuring the performance of the algorithm. We define accuracy as "how frequently the model identifies the correct video that matches the text."

Specifically, during testing, we feed in a series of text descriptions to the model. For each description, the model iterates over every frame of every video that is saved, then selects the top k closest matching frames. If a majority of these k frames belong to the correct video, we label this model prediction as correct.

5.2 Results

To examine the performance of the different modifications to the baseline model, we tested 4 different models and measured each of their accuracy: the baseline model, the baseline model with keyframe extraction, the finetuned model, and the finetuned model with keyframe extraction. Each model was tested on 100 pairs of text descriptions and videos. The results are as follows:

Keyframe \ CLIP	Baseline	Finetuned
Without Keyframe	0.36	0.32
With Keyframe	0.4	0.35

5.3 Analysis

From the table above, we see that the accuracies (out of 1) of the four different combinations of models are respectively 0.36 for baseline and no keyframe, 0.4 for baseline and keyframe, 0.32 for finetuning and no keyframe, and 0.35 for both finetuning and keyframe.

For each text description fed in during testing, the model had to predict which 1 video out of a set of 100 was the closest match. Therefore, a model that made predictions based on random chance would achieve an accuracy of 0.01. As such, the table suggests that our model performs above chance, but accuracy is still significantly lower than would be required to deploy the model commercially.

Fortunately, keyframe extraction appears to improve accuracy. This result was somewhat unexpected given that extraction was originally only intended to reduce time costs. We present one explanation for the result. As discussed in the Methods section, the baseline model uniformly samples 1 frame per second to save (due to logistical costs); only these frames are used to compute predictions. This naive sampling method likely often fails to capture frames containing important information (e.g objects, people, or other entities) that the model could use to identify particular videos. Employing keyframe extraction seems to slightly ameliorate this risk.

Unfortunately, finetuning appeared to decrease accuracy. To understand why, note that the training accuracy was 0.33, which is roughly comparable to the testing accuracy of 0.32. The lack of a huge gap between the two metrics suggests that the model did not overfit; conversely, because they are both low, the model likely underfit. This is for a couple of potential reasons: the learning rate, the number of epochs, or the number of unfrozen residual attention blocks were too low (making the trained network architecture insufficiently deep).

6 Conclusion / Future Work

With greater time and computational power, we would have focused on finetuning the CLIP encoders for longer and unfreezing more layers, with the goal of boosting training accuracy. We also would have experimented with alternative hyperparameter choices or head architectures.

7 Contributions

All members generally contributed to all aspects of the project, but each member was more involved in one or more areas.
Kunal worked on writing the code for testing the model, breaking the videos into frames, and defining the Dataset class in data.py. He also worked on finetuning the model.
Sunny worked on searching for proper datasets to use at initial stages, preprocessing the dataset for training and testing, writing the code for the keyframe extraction algorithm, and handled some hyperparameter tuning.
Laura wrote up the Related Work section and worked on adapting code from the CLIP model to this

project as well as implementing evaluation metrics. In addition, she helped with identifying potential keyframe extraction methods and ran tests on the performance of the different models.

References

[1] Li, X., Xu, C., Yang, G., Chen, Z., & Dong, J. (2019). W2VV++:: Fully deep learning for ad-hoc video search. *ACMMM*.

[2] Dong, J., Li, X., Xu, C., Ji, S., He, Y., Yang, G., & Wang, X. (2019). Dual encoding for zero-example video retrieval. *CVPR*.

[3] Li, X., Zhou, F., Xu, C., Ji, J., & Yang, G. (2020). SEA: Sentence encoder assembly for video retrieval by textual queries. *TMM*.

[4] Gabeur, V., Sun, C., Alahari, K., & Schmid, C. (2020). Multi-modal transformer for video retrieval. *ECCV*.

[5] Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., & Sutskever, I. (2021). Learning Transferable Visual Models From Natural Language Supervision. *ICML*.

[6] Chen, D., Dolan, W. B. (2011, June). Collecting highly parallel data for paraphrase evaluation. In Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies (pp. 190-200).

[7] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.