Predicting if two frames are part of the same video

Armelle Hours, Florence Osmont, Benoît Müller
Image and Visual Representation Lab, EPFL, Switzerland

***Abstract*—In this project, we tackle the problem of predicting whether two frames come from the same video or not. The motivation is to assess how similar two images are, which could find applications, for instance, for story visualization or video generation. To do so, we use a dataset of videos, we define a class to dynamically extract a pair of frames from the same video or from two different ones. Then, we use the CLIP image encoder (vision transformer) to extract meaningful image features. We then implement two classification methods, a cosine similarity based approach and a neural network with two hidden layers. They take as input our pair of features, and they output the classification prediction, i.e. whether the two frames belong to the same video or not. We find that the cosine similarity method and the neural network trained for 2 epochs have approximately the same accuracy of 88%.**

I. Introduction

The motivation behind this project is to be able to determine if two images are sufficiently similar to be part of a same consistent sequence of images. Videos usually have the good property that successive frames represent a similar scene at different times. They have a notion of similarity in the sense that they do not necessarily represent the same disposition of objects or beings, but they keep the same style during the time: light, colors, drawing style, background etc... Thus, we'll use in this project the similarity criterion "coming from the same video".

To model this motivation, we define our problem as recognizing if two frames come from the same video. This defines a binary classification problem, where a sample is a pair of frames, the positive class indicates that the frames come from the same video, and the negative class indicates that the frames come from two different videos.

In this project, we first do a general pre-processing of the two images independently, and then we compute their similarity by two different methods: the cosine similarity and a neural network.

II. Model and Methods

A. Model

As image encoder to transform our images into features, we use the pretrained ViT-B/32 model from CLIP. Then the label is established either by the use of cosine similarity or a neural network with two hidden layers, each containing 250 neurons. This is resumed in figures 1 and 2.

B. Data

The raw data used is a directory of 4,453 videos, with durations about few minutes. They were extracted from YouTube at [1].

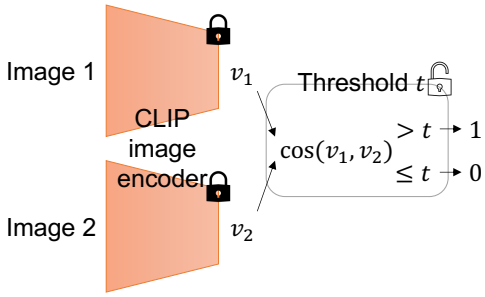

Figure 1. Cosine similarity based method

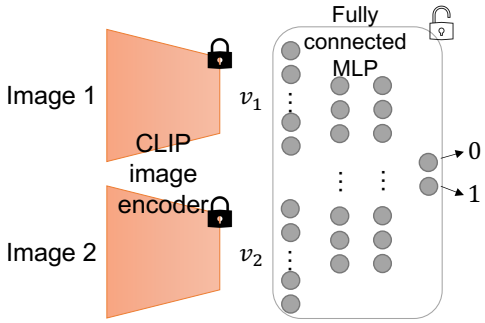

Figure 2. Neural Network based method

We consider pairs of frames from this video, the label indicating if they come from the same video. As a first step, we investigate what is the most adapted data structure to rely on. To implement this, we create a subclass of the inbuilt class "Dataset" of PyTorch. This class takes as data hyperparameter the time interval between two frames (used for positive pairs), the number of samples, the ratio between positive and negative samples and a seed.

Because images can take a lot of memory space, we decided not to store pairs of frames, but to do it in a dynamic extracting way. More precisely, our Dataset class defines how the pairs of frames must be sampled: when the Dataset object is asked to return a sample, it goes to the videos directory, extract the desired pair of frames, and returns it.

To ensure uniformity of sampling according to videos and frames, we use randomization for the choice of sampling. The samples need to have well-defined and deterministic indices, so an external csv file containing the positions of the pairs of frames could have been used, but it would have necessitated to build the whole list of samples at the initialization of the data object, and videos would have been accessed a second time at the return of the samples. To read videos properties such as duration

and rate of frames per seconds, we need to open the videos and it takes an unnecessary long time.

Our solution has been to use a pseudo random selection of the pair of frames, by using the wanted index as a seed of the random process. This way, the uniformity of the random process ensures uniform sampling, and the use of the seed make it a well-defined and deterministic indexation, that does not need to be computed in advance.

The positive examples are created with two frames coming from the same video with a 2-second difference. We use this arbitrary time difference for our positive samples because the motivation of the project is to say that two frames in the same video are similar and some video may change a lot between the beginning and the end. The negative examples are selected as two random frames from two random different videos. We used the library OpenCV to extract these frames.

Since we generate the pairs of frames ourselves, we are able to choose the positive case rate to be equal to 0.5. This way, we minimize the risk of having a label bias on our model by having balanced classes.

All the randomness used in the creation of our data is governed by the seed entered in the parameters of the class.

C. Preprocessing

Our pre-processing step is done with the CLIP image encoder [7] and in particular its model 'ViT-B/32'. CLIP is trained to extract features from an image jointly with a text encoder to align the text-encoding and image-encoding of text-image pairs. Although that is not our goal, it suits our problem well, as we also want to study the similarity between images. The CLIP image encoder generates high quality general features for each frame.

When we call our Dataset class for a specific sample, it calls the image encoder of CLIP to extract image features. Since those features are one dimensional, have a reasonable size but are a bit long to compute, we decided to save them in a separate folder. Hence, the data set loads them if they have been already pre-processed.

Clip also normalizes and rescales the images to match its training configuration.

D. Similarity measures : methods

We use two different methods to measure the similarity of two images and predict their labels.

1) Cosine Similarity: The first similarity measure that we use is the cosine similarity on the two features vectors.

We take the cosine similarity (normalized scalar product) of the two features obtained with CLIP, and return the predicted class based on a threshold. We optimize this threshold to maximize the accuracy on train data. The fact that the classes are perfectly balanced gives the accuracy a low risk of being a bad metric . We then measure the accuracy on the validation set and we also compute the F1 score.

2) Neural Network: In a second approach, we also decide to code a multilayer perceptron. Since our features are 1x512 dimensions tensors, we chose to define our network with an input layer of 1,024 neurons (to match the size of 2 stacked feature-vectors), 2 hidden layers with 250 neurons in each, and an output layer consisting of 2 neurons as we are studying a classification problem. Since we already extracted the features using CLIP, there was no need for convolution layers. After each layer, the ReLU activation function is used. The Adam algorithm is used as the optimizer and the cross-entropy is used as our loss function. We chose the ReLU because it is computationally efficient. We chose the Adam optimizer because it is an improved version of gradient descent handling well large data (requires less memory and is efficient). As hyperparameters, we use a batch size of 128; 2 epochs and a learning rate $= 5 \cdot 10^{-3}$. We chose the cross-entropy because it suits well classification problems. If our model outputs the pair (a, b) for a datapoint, its predicted label is 0 if $a > b$ and 1 otherwise. We also save our model parameters, the optimizer parameters, the number of epochs we ran it on and the loss function used in a .pt file. This would allow us to use this trained model on future data.

E. Partition of the Dataset

To ensure independence in our data between the train-validation-test data, we divide our videos in three corresponding lists with the ratio 70:15:15. This division of videos is done only once when the class is created according to the seed given. All pairs of frames created within a list are independent from the rest, and the three data sets induced have independence. Notice that the size of each three data sets could be chosen independently, but for consistency with the video splitting and the density of sampling per video, we use the same ratio 70:15:15 to define the number of pairs of frames in each video set. Eventually, we considered 10,000 samples, 7,000 for training, 1,500 for validation and 1,500 for testing.

For cosine similarity, the train set is used to optimize the threshold. The validation set is used to compute the metrics associated.

The neural network is trained on the train set and we use the validation set to compute the accuracy and display it during optimization.

The test set is used only once, to estimate the metrics associated to the neural network final model.

III. Results

Cosine Similarity

On the $10,000$ sample train set, we first plot the ROC curve for various thresholds to get insight of the over quality of the cosine similarity score, see Figure 3.

We can see that the curve has an overall good area, and shows almost symmetry with respect to the axe $(-1, 1)$. The threshold having the best accuracy is plotted on the curve of Figure 3. We see it induces a higher priority on true positives than true negatives, which is of interest with

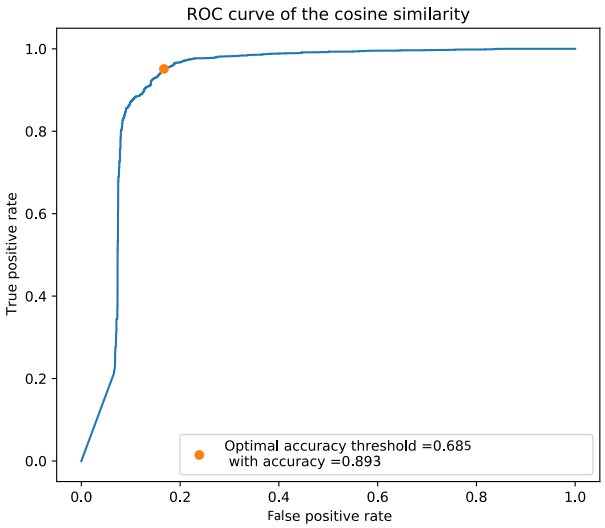

Figure 3. ROC curve of the cosine similarity

the convention that the class need to be chosen such that type I errors are less desirable than type II errors. Its value is around 0.685, and gives an accuracy of 89.3%.

Using this threshold on a $1,500$ validation set, we get an accuracy of 87.9%. We see that we loose a small amount of accuracy in the validation set, namely 1.3%, showing that the model don't over-fit. As a matter of confirmation of the fact that our accuracy is meaningful, we compute the F1-score. It is not very far from the accuracy, at 87.0%.

Neural network

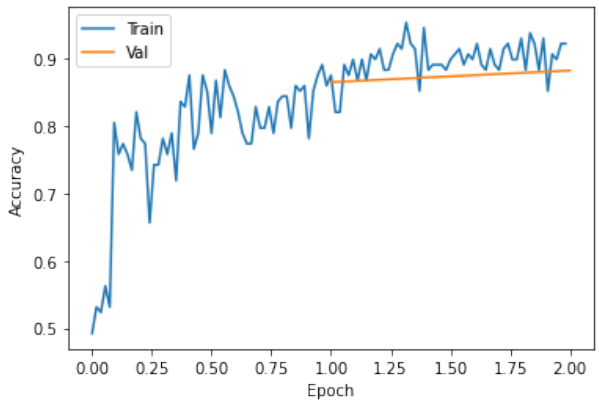

Figure 4. Accuracy of our model depending on the epoch, where we consider the accuracy at 0.5 epoch as the accuracy of half of our Dataset during the first epoch (and similarly the accuracy at 1.5 as the accuracy of half of our dataset during the second epoch)

After the first epoch, the validation set accuracy is of 86% and average loss of 0.3180. After the second epoch, accuracy of 88% and average loss of 0.3256. As seen in figure 4, the accuracy of the train set increases as the epoch increases and in figure 5 the loss decreases as the epoch increases. This is consistent with the fact our model learns the more epoch we run it through. We also notice that both

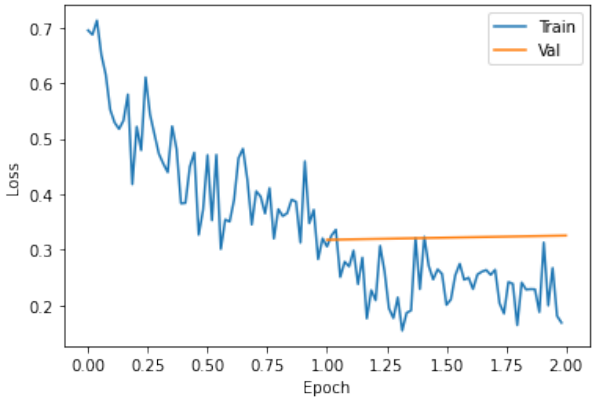

Figure 5. Loss of our model associated to the epoch, where we consider the loss at 0.5 epoch as the loss of half of our Dataset during the first epoch (and similarly the loss at 1.5 as the loss of half of our dataset during the second epoch)

slopes stabilize after the 1st epoch. This means that our model doesn't learn significantly more when we run it by 2 epochs than when we run it by one. This phenomenon is also observed when looking at the validation set curve. One hypothesis could be that we get our tensor features using the machine learning model CLIP which has been trained on a very large scale Dataset, so a large part of image processing might've already been done by the image encoder. Moreover, after applying our model on the test set, we had again an accuracy of 88% and an average loss of 0.3186.

IV. Discussion

One original feature of our code is that it takes as data the videos themselves, and not an already sampled data consisting of fixed pairs of frames. As a result, the data can easily be changed, refined with new videos in mp4 format, and the size of our Dataset can easily be increased.

As mentioned in the introduction, the motivation of the project is to assess how similar two images are, among others by recognizing the style of a created image: light, colors, drawing versus picture, etc...

However, this puts into light the issue of how precise do we want to be with our predictions. If we consider two frames to be similar but the lighting is different, what should our methods output?

The time interval chosen between two frames of the same video could be studied according to this purpose. If it is too short, the frames could be too similar and the generated images could never be categorized as similar. If it is too long, the frames will not have enough in common and would confuse our prediction task, or the generated frames could always be categorized as similar. For example, two frames taken from the same video but from different shots can be totally different. A study of the set of video could also be done, to select the ones that have a suitable frequency of cuts or a suitable frame style consistency.

V. Summary

To conclude, we created our Dataset dynamically by extracting pairs of frames from the same video with constant time interval or from different videos. We also used seeds for consistency. We then used the CLIP image encoder in the pre-processing part to extract general features for each frame, and we stored them in files to save on computation time. The cosine similarity with an optimal threshold gives us the same accuracy as running our data through a Neural Network for 2 epochs. That accuracy is approximately 88% which is high. That might be because CLIP does a large part of the image processing. Further research questions could be if choosing a smaller time interval increases the accuracy, to what extent does our methods generalize for frames that do not belong to videos and to what extent is the similarity criterion "belonging to the same video" strong.

Acknowledgements

The authors thank Martin Nicolas Everaert for his helpful suggestions and weekly discussions.

References

[1] Various authors. *Trending YouTube Video Statistics*. 2018. URL: https://www.kaggle.com/datasets/datasnaek/youtube-new?select=USvideos.csv.

[2] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[3] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[4] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[5] TorchVision maintainers and contributors. *TorchVision: PyTorch's Computer Vision library*. Nov. 2016. URL: https://github.com/pytorch/vision.

[6] Nicki Skafte Detlefsen et al. *TorchMetrics - Measuring Reproducibility in PyTorch*. Feb. 2022. DOI: 10.21105/joss.04101. URL: https://github.com/Lightning-AI/metrics.

[7] OpenAI. *CLIP*. 2021. URL: https://github.com/openai/CLIP/tree/d50d76daa670286dd6cacf3bcd80b5e4823fc8e1.

[8] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[9] Robyn Speer. *ftfy*. Zenodo. Version 5.5. 2019. DOI: 10.5281/zenodo.2591652. URL: https://doi.org/10.5281/zenodo.2591652.

[10] P Umesh. "Image Processing in Python". In: *CSI Communications* 23 (2012).