Scoring Algorithm for Existing Video Caption Model

Teamprojekt

Group supervisor:
Zohreh Ghaderi
Authors:
Kasra Aghamohammadi
Misheel Ganbold
Nicolai Schibel
Emily List
Nele Wulf

This is the documentation for our group project: "Scoring Algorithm for Existing Video Caption Model", as a part of the "Teamprojekt" of the University of Tübingen in the summer semester of 2022.

25.09.2022

Goals

The goal of the project was building a website where a user could upload a short video clip, which would then be transferred to an AI model to generate and display a caption describing what happens in the video. Furthermore, a rating form with different rating criteria should be displayed and the user would be asked to rate the generated caption. Those ratings were to be saved and taken as inputs for a rating algorithm to help improve the AI-model. We also wanted to add some more features, like being able to choose from some already provided example videos, in case there was no video by hand and being able to view the stored ratings.

How to Use

If you want to use our website, you can either download the AI model and our project from our repository or use this link Video-Caption-Model for a website preview without AI functionality. In either case, you will be greeted by our main page, where you can upload a video file. We ask you to only use short videos with up to ten seconds length, as longer videos will significantly lower the quality of video captioning provided. If you don't have a video at hand, you can select a video from our "list of videos" on the left-hand side of the web-page. You will then see a "choose selected video"-button on the main page. After uploading the video, the AI-model will generate a caption, that may or may not correctly describe the contents of your uploaded video. It is now your turn to rate the generated caption using different criteria. Your given data is then collected in a .csv file on our server if you used the link, or your own device if you downloaded the project. Our project will also calculate a rating from 0 to 100, depending on your given data, which can later be used to help train the AI-model. We also made a third site, called "all ratings", which provides exactly what users would expect from its name. It also displays helpful statistics to each video if needed.

Requirements for downloading the project

If you want to run our application on your own device, you will need to have a few things installed:

- Python 3.9 or higher
- git
- a rust compiler (https://rustup.rs/)

and the following dependencies (you can install them via 'pip install')

- streamlit
- mmcv==1.3.9
- torch
- timm
- scipy
- pytorch_lightning
- einops
- transformers==4.12.3
- decord

At last, you will have to clone our GitHub repository,
cd into Scoring-Algorithm-for-Existing-Video-Caption-Model
and download the VASTA.ckpt file and add it to the folder

You will then be able to start the program with the following command:
streamlit run main.py

How does it work?

Backend

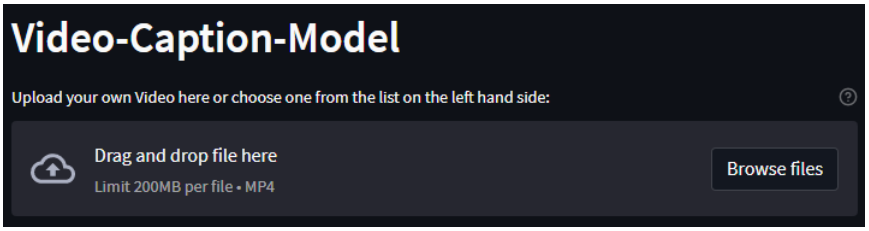
We decided on a very minimal design for our website, as we only need to be able to upload a video and rate it.
Therefore we started with the `dataloader` class, which was mostly given by our group-supervisor, of which the main part of this class is the `__getitem__()` function.
This function takes the video path and transforms it into a "tensor" comprised of 32 uniformly selected frames of the video.
(A "tensor" is a data structure needed in working with neural networks, it resembles a classical multi-dimensional array.)
This function is then used in our `gen_caption()` function of our "main" file, in which we use the "swin-video-transformer" to prepare the aforementioned tensor for the model.
We give the tensor to the model, which then in turn outputs our caption.
Lastly the `gen_caption()` function is packed in the wrapper function `get_caption()`, to get a nice loading-spinner.

Frontend

Our frontend operates solely on the Streamlit-API, so we simply created three pages, the main-page: in which the user can upload a video and get a caption and rate it, the all-ratings-page: in which the user can see all ratings stored on the site, and lastly, the list-of-videos-page: in which the user can watch some prestored videos, select them for caption-generating or examine the statistics for each video.

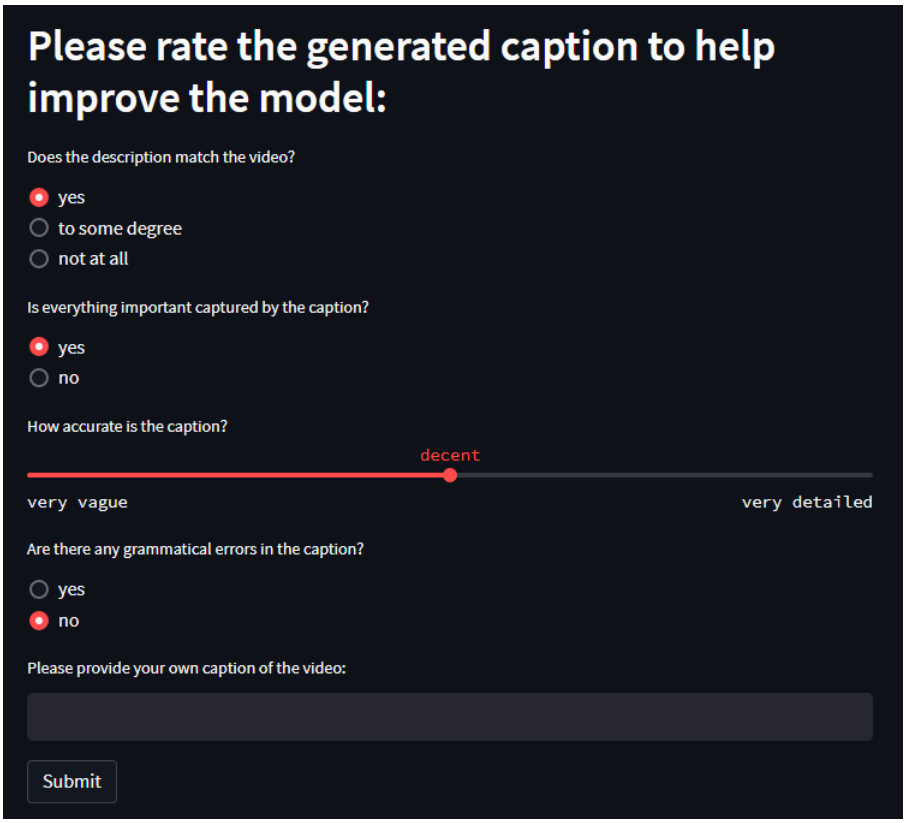
`main-page`

At first the only thing the user can do is upload a video, if the user decides to do so, the caption will be generated, this may take some time.
For uploading the video we simply used the `streamlit.file_uploader()` command, which looks like this on the site:

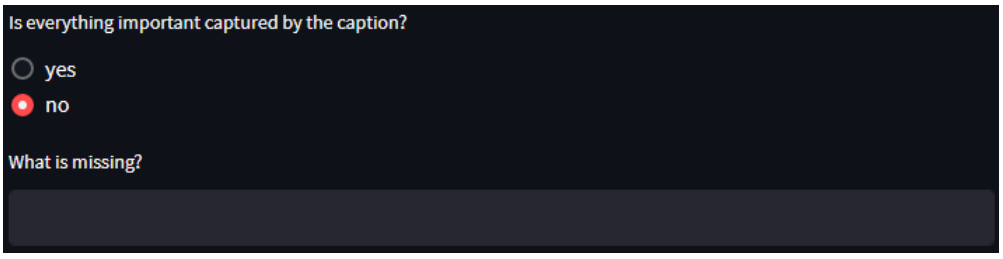


The uploaded video is then given to the `get_caption()` function which was mentioned in the backend section.

The caption is then generated and revealed to the user, which then unlocks the ratings section below:



We decided to implement the feature to store user-written captions, if the user could provide a better caption than the generated one. Also a 'what is missing' input will pop up, if not everything important is captured by the caption.



Lastly if the user then submits their rating, that rating is stored on the ratings.csv file in the hosting-server, and added to the all-ratings-page.

Scoring Algorithm

The scoring algorithm we decided on is very simple.
We have our 4 rating criteria with weights, which then get added up to a score of 0-100%:

$$\texttt{rating\_match} = \begin{cases} \text{yes} \\ \text{to some degree} \\ \text{no} \end{cases}$$

weight:35%

$$\texttt{rating\_capture} = \begin{cases} \text{yes} \\ \text{no} \end{cases}$$

weight:25%

$$\texttt{rating\_accuracy} = \begin{cases} \text{very detailed} \\ \text{detailed} \\ \text{decent} \\ \text{vague} \\ \text{very vague} \end{cases}$$

weight:25%

$$\texttt{rating\_grammar} = \begin{cases} \text{yes} \\ \text{no} \end{cases}$$

weight:15%

Based on the selected ratings, we uniformly map a fraction of the selected weight to the ratings, for example if we have a rating of this form:

$$\begin{aligned} \texttt{rating\_match} &= \text{yes} \\ \texttt{rating\_capture} &= \text{no} \\ \texttt{rating\_accuracy} &= \text{decent} \\ \texttt{rating\_grammar} &= \text{yes} \end{aligned}$$

Since `rating_match = yes` and the question is: "Does the description match the video?", we give the full 35%, in case its "to some degree" we give half, and 0 if its "no".

The other ratings are handled the same way. Therefore the transformation of the ratings looks something like this:

$$
\begin{aligned}
&\texttt{rating\_match} = \text{yes} \\
&\texttt{rating\_capture} = \text{no} \\
&\texttt{rating\_accuracy} = \text{decent} \\
&\texttt{rating\_grammar} = \text{yes} \\
&\Downarrow \\
&\texttt{rating\_match} = 35\% \\
&\texttt{rating\_capture} = 0\% \\
&\texttt{rating\_accuracy} = 12.5\% \\
&\texttt{rating\_grammar} = 0\% \\
&\Downarrow \\
&\texttt{percentage} = \texttt{rating\_match + rating\_capture + rating\_accuracy + rating\_grammar} \\
&\qquad\qquad = 35\% + 0\% + 12.5\% + 0\% = 47.5\%
\end{aligned}
$$

In this case the model would achieve a score of 47.5%.

`all-ratings-page`

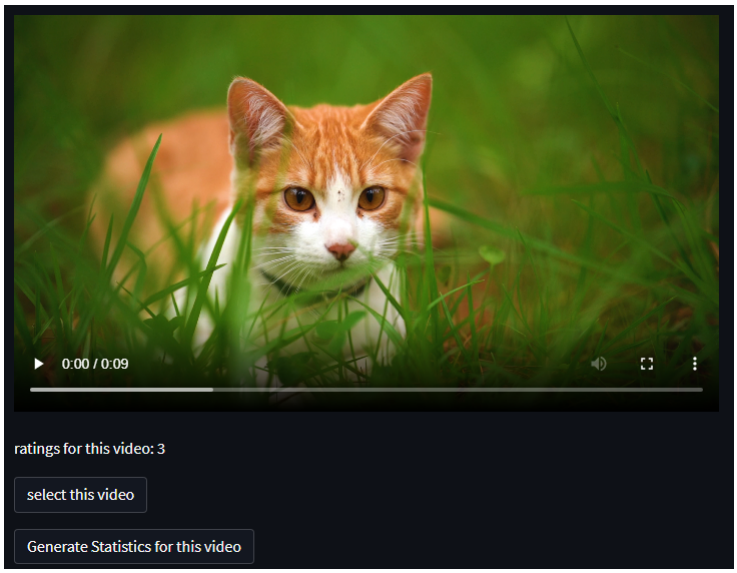
The all-ratings-page simply outputs the contents of the ratings.csv file in a table with the `streamlit.table()` function, with a data frame created by the `pandas.DataFrame()` function of the pandas-library.
The table looks like this:

	video name	generated caption	does it match	rating capture	what is missing	how accurate	grammar errors	user caption	percentage
1	video8798.mp4	TEST	to some degree	no	sdf	decent	yes	dsafg	30.0
2	video8798.mp4	TEST	yes	yes	empty	decent	yes	asd	72.5
3	video8798.mp4	TEST	yes	yes	empty	decent	no	ad	87.5
4	video8798.mp4	TEST	yes	yes	empty	decent	no	ad	87.5
5	video8798.mp4	TEST	yes	yes	empty	decent	no	ad	87.5
6	video8798.mp4	TEST	yes	yes	empty	decent	no	ad	87.5
7	video8798.mp4	TEST	yes	yes	empty	decent	no	ad	87.5
8	video8798.mp4	TEST	yes	yes	empty	decent	no	ad	87.5

`list-of-videos-page`

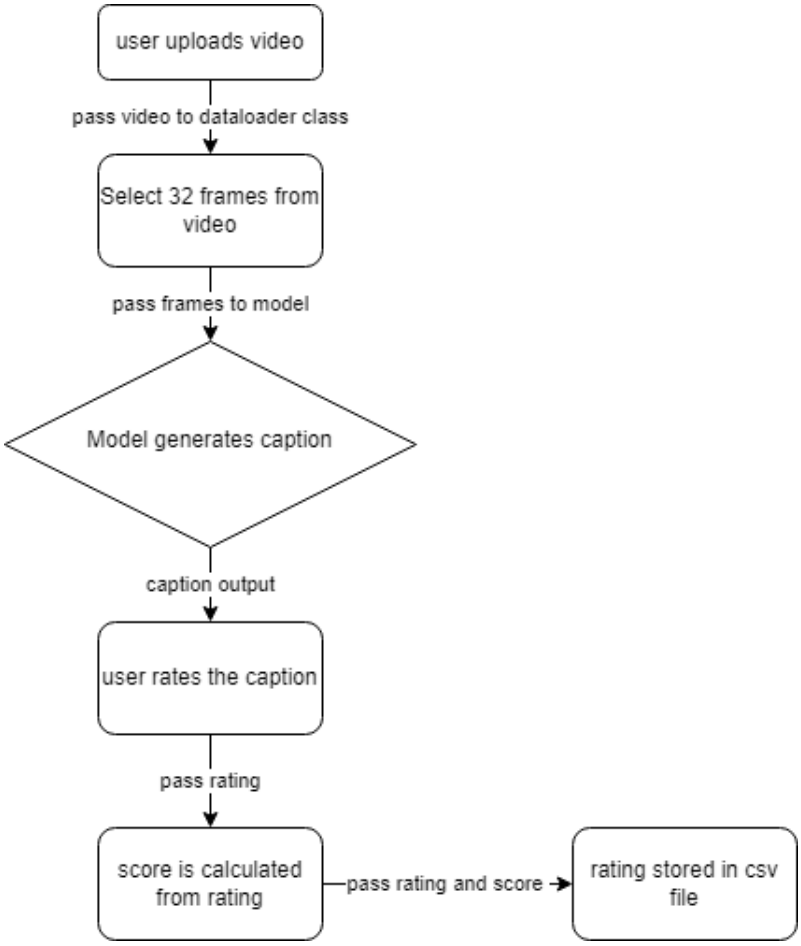
This page contains some example videos, if the user does not want to provide one of his own.
Directly below the video is a counter for the number of ratings this video has. Below that the user can click a button that says `select this video`, this video is then selected, and

can be chosen below the file-uploader of the main-page.



Below that button is another one that says `Generate Statistics for this video`, this one generates the statistics for this video, based on all ratings for this particular video, found in the ratings.csv file.

Frontend Flowchart



Why Streamlit

To realise our project, we had to select a web framework. Since the model was written in Python, it would only make sense to find one which also used this language. Our supervisor suggested to use Flask or Streamlit. Flask is widely used to create simple applications. It is easy to learn and also supports many extensions. It certainly could have worked well for our project. Still, it seemed that Streamlit was the way to go. Flask doesn't offer direct layout and design features, while Streamlit is a data dash boarding tool, with many data visualization components and an overall very pretty website design by default. To achieve the same visualization components with Flask we would have had to search for suitable libraries and spend more time on including those. Streamlit seemed to offer just the right tools we needed: different input widgets like buttons, sliders and select boxes, but also media inputs for videos and files. Session state variables, a multi-page app layout, an option for caching, data charts, tables and much more. In addition, Streamlit is relatively new, while Flask has existed since 2004, so we were tempted to try something new and trendy that we had not heard of yet. Streamlit is an open-source-python library that was built in 2018. Over the last few years, a large community has gathered around Streamlit, and its developers are constantly implementing new features that the community is asking for. Streamlit especially addresses data science and machine learning projects. It can be used to easily create a website to demonstrate project results and data without knowing anything about frontend programming and designing. Also, Streamlit offers a free sharing server, where you can publish your projects. We soon realized that we all had no problems learning how to use Streamlit for our purposes, as it has great documentation and helpful tutorials, while being very basic and understandable.

Our Workflow

It was clear from the beginning, that we would use Git and GitHub to code our application, primarily because the AI-model was given us on GitHub, secondly because we learned a lot about GitHub in the presupposed lecture. To communicate we used discord for our team and slack for contact with our supervisor. All that worked really well for us.

For our workflow, we did not decide in advance for a specific method like Waterfall, Kanban, Scrum or Design Thinking. We just had a meeting every two or three weeks, either in person or online. Our supervisor viewed our results and gave us our next task. As the project was not that big and there weren't so many people involved that worked really great for us.

One thing that didn't work so well was that we did not decide for a branching strategy. This was a huge mistake and led to some confusion. We started to build our website on the shared GitHub repository, created by our supervisor. We created a new branch named 'develop'. There we implemented the first steps. Pretty fast we moved on to continue wor-

king on another branch called 'develop load model'. We implemented most of the project on this branch. As we wanted to share the project on the Streamlit Cloud, we had to fork the whole project to our own GitHub account. We again created a new branch called 'final changes'. From our own GitHub account we were then able to upload the project to the World Wide Web. Unfortunately, the free access is limited in storage, therefore we had to upload a version without the actual model, only the frontend design. That was done on another branch called 'final-changes-without-ai'. Now all changes that we made had to be committed to the 'final changes' and the 'final-changes-without-ai' branch. At last we transferred the final version back to our supervisors GitHub. Not knowing which branch we were working on at any given time led to us having to communicate way more than needed, before pushing any changes.

What did we learn?

Implementing the application was a huge success for us, as we were able to learn a lot from the project.

First of all most of us had never used Git and GitHub before and we were a bit scared of all its functionality. Of course we had learned everything in theory, but then using it in the real world, making commits and resolving merging conflicts was a completely different thing. However, we quickly learned that it looks much more complicated than it actually is and we are now able to use Git and GithHub for other projects without any worries.

The same applies for Streamlit. Having to understand a whole new framework seems difficult at first, not to mention that some have never used Python before. Again it turns out that Streamlit was very easy to learn. The developers provide a very good documentation website with lots of examples. Thanks to the big community of Streamlit, problems and additional features that are not discussed in the documentation can be found on other websites like the Streamlit forum or Stack Overflow. All in all we can only recommend Streamlit and will probably use it for our own future projects, as it only takes a few minutes to understand the main concepts and is very easy to use.

Apart from getting to know new programs, we were also able to learn a lot about teamwork and communication. For most of us it was the first time creating a program as a team. We had to implement the program together, understand the code of our team members and ask and communicate if there was any ambiguity. We also had to admit to problems that we couldn't solve ourselves and then ask our supervisor for help. The project was an important experience in our 'career' path and we are very happy that we had such a nice and helpful supervisor.

Some other small things that we want to mention are that thanks to this project we learned

how to understand a given AI-model, at least along general lines and what the inputs and output properties are. We learned how to present our results and finally how to write a documentation.

Difficulties

Some of the difficulties we experienced were in installing all the necessary packages needed by our model, so our code could run without problems. Especially finding the correct version of said packages, which would be compatible with the model. In testing our project, we experienced significant delays because the model does some heavy calculations, resulting in long loading times, if your device doesn't have much computing power. When trying to work on it in person, some of us could not install the model on our laptops, as it simply needed too much space. Lastly, we had some issues regarding the hosting of our Streamlit app. As previously explained, the model was very big, so we couldn't properly host our site on the free server. We had decided to use the free Streamlit hosting service, and cut down on needed storage, by decoupling our model from the frontend. That way, we had fixed both issues, so we could test only our frontend by using the free server.