University of Zurich[UZH]

Interactive Video Retrieval: AGXK-20

Andreas Bucher[1], Gaudenz Halter[2], Xiao'ao Song[3], Kevin Steijn[4]

June 4, 2020

[1]Matriculation Number : 17-717-745, andreas.bucher@.uzh.ch
[2]Matriculation Number : 13-780-970, gaudenz.halter@.uzh.ch
[3]Matriculation Number : 18-747-949, xiaoao.song@uzh.ch
[4]Matriculation Number : 19-770-429, kevin.steijn@uzh.ch

Contents

1 Introduction

With the popularity of digital devices, in today's society, everyone can easily record videos. As such the quantity of video footage is continuously increasing, which poses novel challenges to the management of these large-scale video data sets and especially to the analysis of and retrieval from such video collections. In cases where target videos are to be identified within a large collection, the appropriate information must be obtained to retrieve the correct video within a large number of similar items in the target database.

The purpose of this project is to retrieve target videos in such cases by means of an interactive tool. In this report, we propose a system to retrieve videos by taking queries submitted from the user in the form of text and sketch drawing. Additionally, the implemented system will be evaluated in an event modelled after a state of the art interactive retrieval evaluation campaign, the 'Video Browser Showdown'[5] (short: VBS).

The remainder of this report is structured as follows: In Section 1, we introduce the dataset and the tasks used to evaluate system performance, along with the project management process. In Section 2, we give an overview of our system application architecture. In Section 3, we discuss in detail how to implement our system step by step such as scene detection, image captioning, caption embedding, color classification and other feature acquisition steps. In Section 4, the functionality of the system and how to use them are introduced. Next, we presented our evaluation results in Section 5. Finally, Section 6 discusses the performance of our tool and concludes.

[5]The VBS is an annual live video search competition, where international researchers evaluate and demonstrate the efficiency of their exploratory video retrieval tools on a shared data set in front of the audience. The details about VBS can be found by https://videobrowsershowdown.org/

1.1 Data set

The data set we use is Vimeo Creative Commons Collection, in short V3C, a collection of 28'450 videos (with overall length of about 3'800 hours) published under creative commons license on Vimeo. Apart from the videos themselves, the collection includes meta and shot-segmentation data for each video, together with the resulting keyframes in original as well as reduced resolution. V3C is split into three shards (V3C1, V3C2, V3C3) with 1000h, 1300h and 1500h of content respectively. Table 1 provides an overview of the three partitions (Rossetto et al., 2019). The V3C also includes a master shot reference which segments every video into sequential non-overlapping parts, based on the visual content of the videos. Every video in the collection has been assigned a sequential numerical id. These ids are then used for all aspects of the collection.

Partition	V3C1	V3C2	V3C3	Total
File Size (videos)	1.3TB	1.6TB	1.8TB	4.8TB
File Size (total)	2.4TB	3.0TB	3.3TB	8.7TB
Number of Videos	7'475	9'760	11'215	28'450
Combined Video Duration	1'000 hours, 23 minutes, 50 seconds	1'300 hours, 52 minutes, 48 seconds	1'500 hours, 8 minutes, 57 seconds	3801 hours, 25 minutes, 35 seconds
Mean Video Duration	8 minutes, 2 seconds	7 minutes, 59 seconds	8 minutes, 1 seconds	8 minutes, 1 seconds
Number of Segments	1'082'659	1'425'454	1'635'580	4'143'693

Table 1: Overview of the partitions of the V3C

The V3C1 dataset is composed of 7475 Vimeo videos (1.3 TB, 1000 h) with Creative Commons licenses and mean duration of 8 min. All videos will have some metadata available e.g., title, keywords, and description in json files. The dataset has been segmented into 1,082,659 short video segments according to the provided master shot boundary files. In addition, Keyframes and thumbnails per video segment have been extracted and available. Figure 1 illustrates the directory structure of the V3C1 (Rossetto et al., 2019). The info directory contains one json-file per video which holds metadata obtained from Vimeo. This metadata contains both semantic information – such as video title, description and associated tags – as well as technical information including video duration, resolution, license and upload date. The msb directory contains for each video a file in tab-separated format which lists the temporal start and end-positions for every automatically detected segment in a video. The keyframes and thumbnails directories each contain a subdirectory per video which hold one representative. In our project, we will use a subset of V3C1 of 10%.

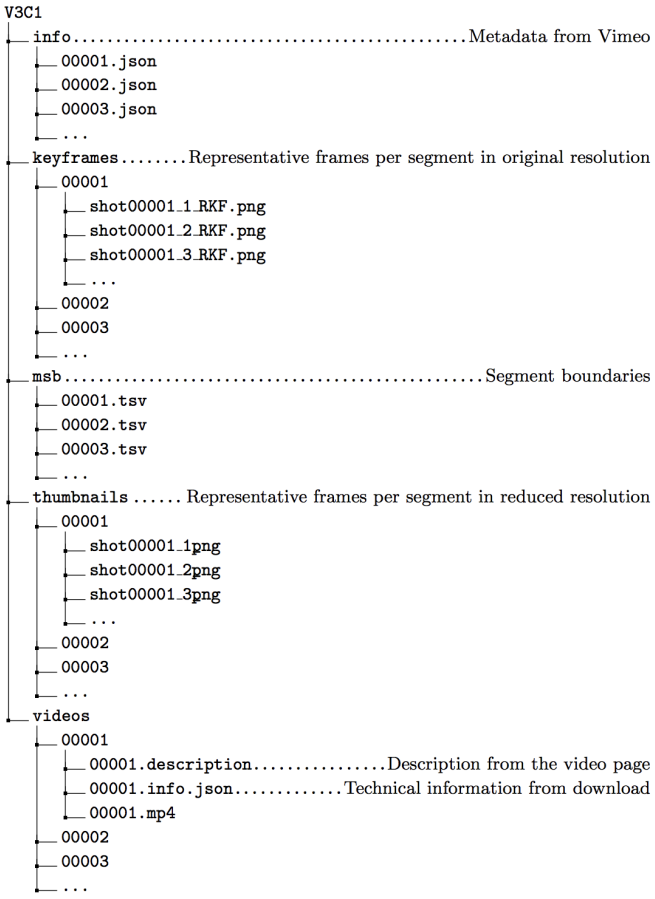


Figure 1: Directory structure of the V3C1

1.2 Tasks

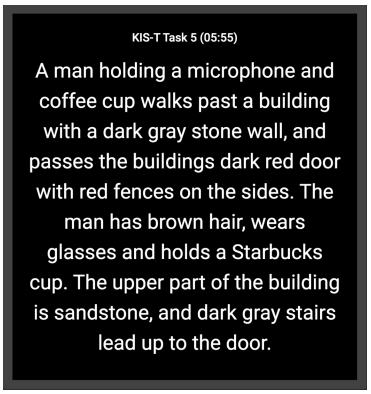
In this project, we implemented an interactive video retrieval system. Our system will be evaluated on three different kind of tasks, which are *visual known-item search*, *textual kown-item search*, and *ad-hoc video search*. The details of each task will be described below.

Visual Known-Item Search
In visual known-item search, a unique $\sim$ 20 second video clip will be given. The system needs to find out the exact match in the video collection. The task has to be completed within 5 minutes and the retrieved result will be submitted onto the server for evaluation. Figure 2a shows an example of this task.

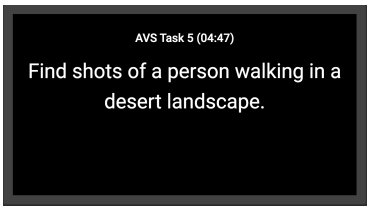
Textual Known-Item Search
In textual known-item search, the textual description will be given. The system needs to match exactly one $\sim$ 20 second video clip to this textual description from the video collection. This task is permitted 8 minutes to complete and the retrieved result will be submitted onto the server for evaluation. Figure 2b shows an example of this task.

Ad-hoc Video Search
In ad-hoc video search, a textual description is given again, but this time the system has to find as many matching clips as possible from the video collection. The task needs to be completed within 5 minutes and the retrieved result will be submitted onto the server for evaluation. Figure 2c shows an example of this task.

(a) Visual Known-Item Search



(b) Textual Known-Item Search



(c) Ad-hoc Video Search

Figure 2: Search Task Samples

1.3 Planning

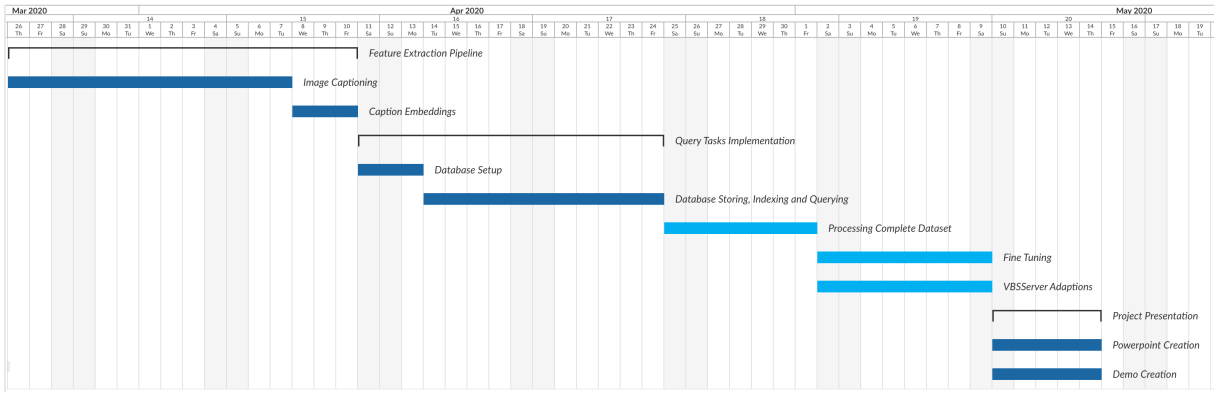


Figure 3: Project planning and procedures

We divided the project into various tasks and subtasks. Figure 3 shows a Gantt chart of our project planing. In the first step, we extracted features from images. This involved training a model for image captioning and converting it into sentence embeddings. The second step is query task implementation, in which we set up the database and handled storing, indexing and querying issues. In the third step, we processed the complete dataset and stored the information of each image in our database. Next, we built the user interface and adapted it to the VBSServer. Meanwhile, we also fine-tuned on our model. Finally, we made the system instruction and operation demo. Our presentation and operation demo can be found on our github at https://github.com/ghalter/interactive_video_retrieval.

2 Application Architecture

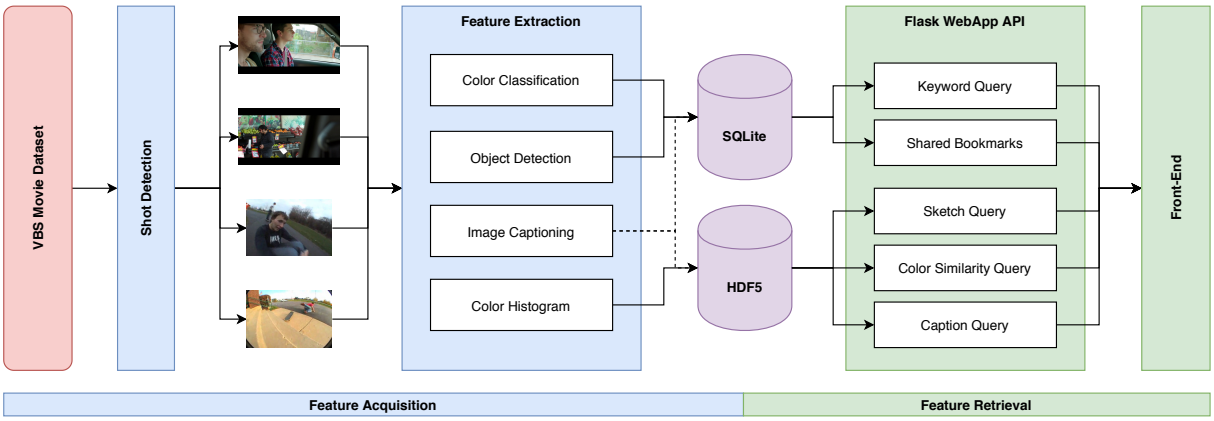


Figure 4: An overview over the architecture of our system. The data is segmented into the shots, features are computed and stored in corresponding databases. The webapp exposes selected API end-points to perform queries on the dataset.

Our final system uses a webapp architecture as its foundation, where the user interaction is directed to a server end-point, which performs an action and retrieves the request's result back to the front-end in the user's browser.

In an essence, the architecture can be split into two stages, the *feature acquisition* and the *feature retrieval* stage. Both will be elaborated in the subsequent sections.

For our data we use two different types storage: the shots, captions and keywords are stored in an SQLite database directly. It would also be possible to store the extracted color histograms as blobs in this database. However, retrieving these blobs is generally not very fast. This is a problem, which is especially pronounced in SQlite. Therefore, we have chosen a different approach for our numeric feature vectors, which boils down to the following: every feature vector is put into a HDF5 file structure, the location of the feature vector in the respective array is then stored in the relational database. This allows us to store more or less arbitrary large feature vectors in our system, without slowing down the database (see Figure 4). Additionally, our HDF5 based feature vector data storage allows us to retrieve ranked lists based on user defined comparators and to return the associated shot entries in the SQLite database.

3 Feature Acquisition

In this section we describe how we obtained the features we used for our project.

3.1 Scene Detection

Given the dataset, we had multiple options to consider for feature extraction. For each video there were a collection of keyframes and thumbnails, images extracted from the video itself. These keyframes are the same resolution of the video, where the thumbnail images were the images from keyframes down sampled to 200x133 pixels. We found that the higher resolution images performed better for the image captioning step, which follows this one. Hence with the next step in mind we elected to focus on the keyframes images.

Since the method of selecting these keyframes involved shot detection that we had not implemented, we felt it prudent to evaluate whether these images are a good source of the shots in a video, shots being interchangeable with scenes in our case. To address this concern we set out to create our own collection of frames. Due to computational and time constraints we couldn't build two models with two collections of images solely for the purpose of discovering which image collection performed best. Therefore we chose to manually compare the two collections.

For our own implementation we relied on a library called *PyScenedetect*. This library goes through the video and computes values for each frame with the light intensity among others. This we store in a csv file for each video to speed up the process for the next stage. The following stage is the detection of the scenes itself, which involve detectors. The library has a couple detectors, we chose to use "Contentdetector" as it is the most advanced, along with "Thresholddetector" for edge cases. These edge cases involved the detector only detecting one scene, due to lack of significant changes in light and color intensity.

These detectors go over the values generated in the stage before and compare the values of nearby frames to find the transitions of scenes. Once it has finished this process for a video we generate a csv file which is formatted as two columns. Those being the start and end time of the detected scenes in seconds. With this csv file we extract the frame from the middle of each scene to generate our collection of frames.

When we compared our collection of frames found using scenedetect and the frames in the keyframes folder we found that scenedetect produced less frames. Which can significantly improve the training of our model, with the question being whether the lower quantity of images were also of similar quality. During the evaluation for the quality of the images we found no noticeable decrease for the scenedetect images. We found that the keyframes often had duplicates of the same scene and as such had redundant images present. Therefore we decided to continue using the images we found using our implementation using the scenedetect library.

3.2 Image Captioning

In order to make the single scenes searchable, we planned to generate captions for the most important scenes of each video and then embed them in a multi-dimensional vector space. This allows us to compare search queries with the embedded captions using similarity measures like Cosine Similarity or Euclidean Distance. To generate captions for the created thumbnails, we initially experimented with two approaches: a) use a pre-trained image captioning model, and b) train our own image captioning model. When testing several pre-trained image captioning models on different images of the thumbnail dataset, we could not achieve satisfying results, i.e. we were not able to retrieve meaningful captions. Moreover, the pre-trained models that we tested did not allow for fine-tuning on our specific dataset to improve the caption generation. Hence, we decided to train our own image captioning model, and if necessary, fine-tune this model to our needs. In the following section, we will first introduce the dataset that we used for training the image captioning model, before covering the model and the training set-up in more detail. Lastly, we will briefly discuss the results of generating the thumbnail captions.

Dataset and Data Processing

The most prominent datasets for training an image captioning model include the Microsoft Common Objects in Context (short: MS COCO) dataset, as well as the Flickr30k (Young et al., 2014) or Flickr8k datasets. All images in the three datasets are annotated with five captions. As the MS COCO dataset is much larger in size (MS COCO 2014 includes in total about 164.000 images), we expected better results from using this dataset and decided to train our model on the COCO 2014 dataset. Overall the MS COCO 2014 dataset has the following characteristics (Lin et al., 2014):

MS COCO 2014	
Size (images)/ split	Train: 40.775 images Validation: 82.783 images Test: 40.504 images
Size (GB)	37.57 GB
Annotations	captions (5x/image), bounding boxes, labels
Classes	COCO defines 91 classes
Other variants	MS COCO 2017

Table 2: Characteristics of the MS COCO 2014 dataset

For the training of the image captioning model, we only used the training and the validation set because the test set does not include any annotations. Before loading the images into the model during training, we pre-processed all captions by a) constructing a vocabulary of all words that appear in the training dataset more often than a certain cutoff value (e.g. 15), and b) by tokenizing all captions in the training and validation set using the word-indices in the vocabulary. Furthermore, we added a start ([BOS]) and an end ([EOS]) token to each caption before tokenization to fulfill the requirements of the chosen recurrent neural network (short: RNN). To save time during hyperparameter tuning and step-wise training, the vocabulary was extracted at the end of the data processing stage and could be loaded at any time during the training.

Model Architecture

Many approaches in automatic image captioning involve recurrent neural networks as the task is similar to other sequence generation problems such as sentence translation. Analogously to sentence translation, RNN-based approaches, especially using an encoder-decoder structure, "translate" images into captions word by word. In such approach, as depicted by Figure 5, the encoder creates an encoded vector representation of an image, denoted as a, whereas the decoder generates the t^{th} word of a caption. At any time step t^i the decoder processes the previous hidden state h_{t-1}, the output of the previous decoder step y_{t-1} and a context vector $\hat{z}_t$. Following the paper "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention" by Xu et al. (2015), the context vector is the dynamically weighted representation of the encoded image, which focuses on relevant parts of the image at each time step t. This so called attention mechanism allows the model to capture certain visual aspects of an image by adjusting the weights of the generated feature vector a. To implement such attention mechanism for $\hat{z}_t$, the hidden state h_{t-1} as well as the encoded image a are fed into a feed forward neural network (short: FFNN) at each time step t_i.

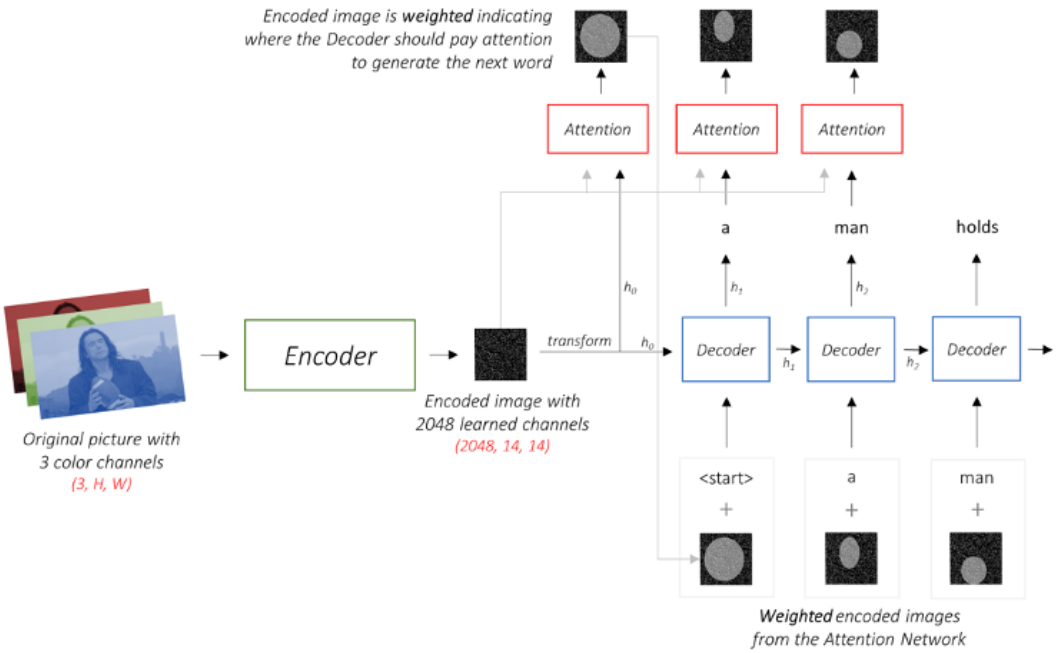


Figure 5: Encoder-decoder network with attention (Vinodababu, 2018)

Adapting the approach of Vinodababu (2018), we implemented the encoder using the pre-trained ResNet101 model, which is further fine-tuned during the training. For the decoder, we implemented a RNN with LSTM-cells with an additional linear layer on top of the RNN as attention network. Before feeding the caption word-by-word into the LSTM-cells, we tokenized each word in the caption and generated pre-trained GloVe word-embeddings. Similar to Word2vec embeddings, GloVe embeddings capture the local context of words and model the meaning of the words in a vector space. Using pre-trained GloVe embeddings provides the advantage that we do not need to train an additional embedding layer, which would increase the parameter space for the gradient updates. Unlike the image-encoder, the GloVe embeddings are not further fine-tuned during training. As described above, these word-embeddings are then fed into the RNN together with the previous decoder step and the weighted context vector.

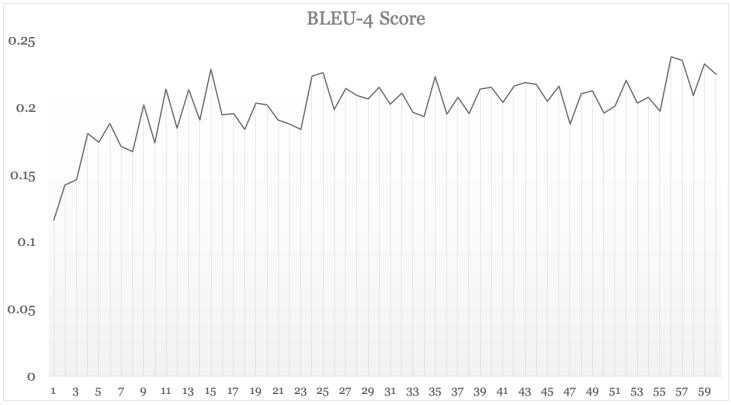


Figure 6: Training results measured in BLEU-4 score on the development set

Generated caption
a group of birds sitting on a tree branch

Generated caption
a motorcycle parked on a street next to a motorcycle

Figure 7: Generated captions for two example images of the MS COCO2014 dataset

Model Training and Results

After more than 23.0 hours of parameter-tuning, we derived at the final set-up for the model and trained it for more than 37.0 hours on a cloud-instance with a 16GB Pascal GPU, 8 vCPUs, and 30GB RAM. The most common metric to measure the performance of image captioning models is the BLEU-4 score. This metric describes how well the generated caption matches the true caption by averaging the percentage of matching 4-grams. The BLEU-4 score can take values from [0,1], whereas a "1" describes a perfect match between the captions. Compared to a 0.25 BLEU-4 score reported in the 2016 “Show, Attend and Tell" paper (Xu et al., 2015) and to a 0.217 BLEU-4 score of human-performance on this task (Vinyals et al., 2017), our final model achieved reasonable results with a maximum BLEU-4 score of 0.238 on the development set. The improvement in BLEU-4 scores during training is depicted in Figure 6.

When testing the model on some images of the MS COCO2014 dataset, the results were also quite satisfying. Figure 7 shows the generated captions by our final model of two exemplary images. These examples demonstrate that the model is not only capable of detecting multiple objects within an image (e.g. multiple birds), but it is also able to spot objects in the background of an image (e.g. a motorcycle). Nevertheless, the model faces some limitations, especially when objects within an image are only partly visible, obstructed or cut-off, and when the color distribution of an image is small.

Thumbnail Captions
Based on the trained model, we retrieved the captions for all thumbnails in our video retrieval dataset. Compared to the previous training, this process was quite fast and took only about 6-7 hours on a local machine without GPU. As expected, the performance of the model in captioning the thumbnails were weaker than captioning the images of the MS COCO2014 dataset. Even though the model derives reasonable captions for a large portion of the dataset, it is sometimes not able to correctly identify the objects in the thumbnails. We believe that these limitations are caused by inferior quality, higher pose variation and sensor blur of the frames in our video dataset compared to the images in the MS COCO2014 dataset. However, as multiple thumbnails belong to a single video and as querying the thumbnail captions is not the only search option, the impact of incorrect captions should be manageable.

3.3 Caption Embedding

To compare the generated captions against search queries, we transformed the captions to sentence embeddings using pre-trained Siamese BERT-networks. In contrast to word embeddings like Word2vec or GloVe, which we used for the decoder in the image captioning model, transformer-based sentence embeddings provide contextualized vector representations of the captions and search queries. These contextualized representations differ from traditional word embedding approaches as they are not restricted to learning a single context-independent vector representation for each word. Instead, contextualized representations can capture non-static semantic meaning in such way that the vector representation of a single word, e.g. 'star', can vary in different contexts, e.g. a 'musician/ movie star' or a 'star in the universe'. We believe that using contextualized representations of the captions allow for more fine-grained search queries than with simple word embeddings.

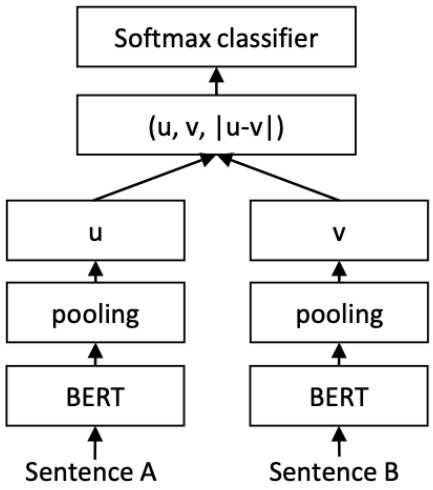


Figure 8: Architecture of Siamese-BERT networks (Reimers & Gurevych, 2019)

Currently, the Bi-directional Encoder Representation from Transformers (short: BERT) is one of the most popular transformer-based architectures for deriving contextualized word representations in Natural Language Processing. Because of its success on a wide array of tasks, including General Language Understanding (short: GLUE), Named Entity Recognition (short: NER), or Natural Language Understanding (short: NLU), we decided to build our approach for generating contextualized caption embeddings on the BERT architecture. However, when testing a pre-trained BERT model provided by the Python library Hugging Face on its ability to generate comparable sentence embeddings by using the class token of the BERT output layer (= 'CLS' token), we achieved only very poor results. Apparently, using the plain, pre-trained BERT model

without further fine-tuning is not feasible for this task. Another, more promising approach is to use pre-trained Siamese BERT-networks as described in the paper "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks" by Reimers and Gurevych (2019). As shown in Figure 8, Siamese BERT-networks are trained on a classification objective function on top of a pre-trained BERT model, which optimizes the element-wise differences between to sentences.

Implementing the pre-trained Siamese BERT-model of Reimers and Gurevych (2019) we were able to derive meaningful and comparable vector representations of our captions. To retrieve captions and videos that fit a specific search term, we transform the query using Siamese BERT-networks and compare the embedded search term with all caption embeddings using cosine similarity. The below table shows the top-five results, i.e. the most similar captions, for a specific search query:

Search Query: 'a man on a bike in the streets'		
Rank	Caption	Cosine Similarity
1	'a man riding a bike on a street'	0.9754
2	'a man is riding a bike in the street'	0.9737
3	'a man riding a bike down a street next to a person'	0.9676
4	'a man riding a bike on a street with a person on a bike'	0.9636
5	'a man riding a bike down a street next to a bike'	0.9633

Table 3: Ranking of search results using cosine similarity

3.4 Color Classification

Color is one of the predominant feature when it comes to human visual perception. It is therefore undoubtedly crucial to exploit this trait, thus we support search by color and color distribution, especially useful for the visual known-item search. There are several ways to think about color in images, one way is to describe an image by its color palette, which is a set of the most abundant colors within the image. Many ways have been purposed for color palette extraction from images, most of them embody some clustering algorithm to aggregate pixel values of color to a small number of clusters, yielding a color palette. In its most naive form, a color palette is computed by simply extracting k centroids using a KMeans based algorithm. However, color in image material consist by definition of a spatial domain, which is lost in the described process, this leads to problematic results in the clustering process, namely, clusters can evolve, although they are not perceived as such.

To circumvent this problem, we implemented a two-step extraction for our color palettes, followed by classifying the resulting colors by a set of given colors names and their corresponding value:

1. SuperPixel extraction using the SEEDS Superpixel(Van den Bergh et al., 2012) algorithm.
2. Clustering the mean values of the superpixels to form a palette.
3. Classify the resulting colors by a given set of color names

(a) Input Image (b) Extracted superpixels (c) KMeans clustering

Figure 9: Processing pipeline for the color classification

Using a superpixel extraction prior to the clustering, our method not only takes the distance of pixels in the color space, but also their spatial distance into account.

To allow a text-based query for colors in an image, we classified each color in the palette by a set of given color names. In our case, we used a dataset from the wikipedia color thesaurus, containing 200 different colors and converted the associated color values into the CIE-Lab colorspace.
This color space yields several advantages over others and associated color models, most importantly, it is designed to be *perceptually uniform*, meaning that colors close together are also perceived to be close by a human observer, distances between colors become comparable from a perceptual point of view this way. Because of this, we can simply use the euclidean distance classify a given color by a set of color values and pick the lowest one.
A more elaborate method would have been to use a data-driven approach to define the centers of these colors, for example by pulling images for each color name from a database such as Flickr and compute a distribution for each name in the CIE-Lab colorspace (Lindner et al., 2012).

3.5 Color Histogram

Color histograms are simple, yet expressive feature vectors to describe a color distribution for a given image, or sets thereof. Color histogram are relatively small in comparison to the actual

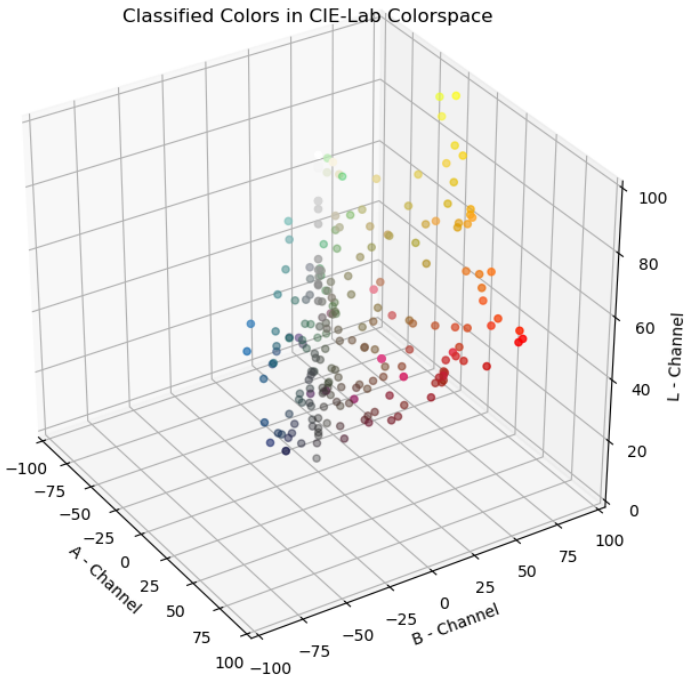


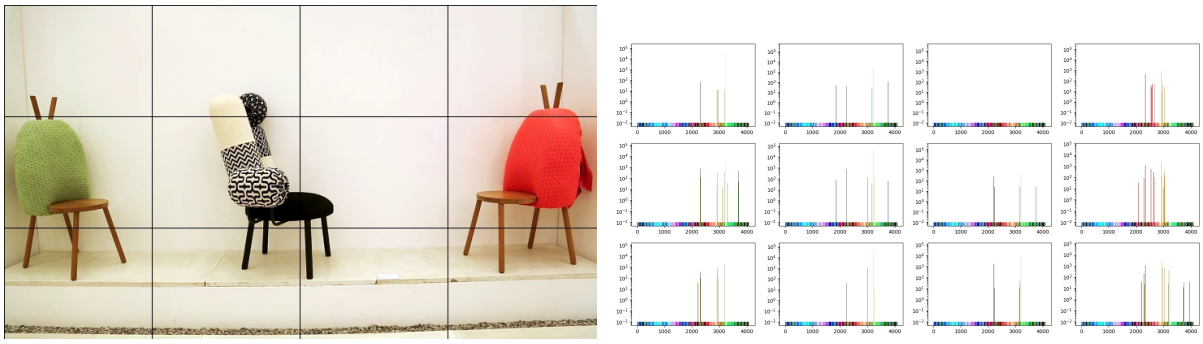
Figure 10: Our classified colors in the CIE-Lab color space,

image, preserve a lot of the original color information and have been shown to perform well in color-based image retrieval(Aibing Rao et al., 1999). The downside of histograms is of course the loss of spatial information, although one can be sure, there was *red* in the image, it is not possible to retrain the position of this color. However, we wanted the user to be able to search for a certain color in a certain region of the image, and thus used a 2D-grid of color histograms to partition the image into 12 equal cells, where a color histogram is computed for each. This approach has already been successfully used in other video retrieval systems such as (Aibing Rao et al., 1999; Ho Young Lee et al., 2003).

Since we used the CIE-Lab color space for all color related computation, we also used it for the color histograms, however, several difficulties arise with this color space when computing a histogram. Notably, the gamut of visible colors is much smaller than the actual CIE-Labs coordinate system. In fact, when converting all possible sRGB (uint8)[6] colors into CIE-Lab colors, and computing a colors histogram with 16 bins in each channel, a dominant number of the bins in the color histogram remain zero, while only 855 bins of 4096 are actually non-zero. This is due to the relatively small gamut of the sRGB color space and is problematic in two ways, first, such histograms will use a lot more space than they actually require when stored and secondly, these empty bins create a computational overhead not needed during histogram comparison.

Our approach to this problem would be, to clamp the boundaries of the histogram to the extent

[6]We tested this with OpenCV, which to our knowledge only supports uint8 bit BGR images to be retrieved from a movie directly, more accuracy can be retieved with ffmpeg or similar libraries specialized in movie reading.



(a) Input image with grid

(b) Extracted histograms

Figure 11: Grid color histogram extraction, colors are aligned using a Hilbert curve through the color space

of the sRGB Gamut, since our application will not support a large color space anyway, this will already remove a significant amount of the bins which always remain zero, in our case 2390 of 4096 bins can be filled. A more efficient implementation would only store the bins which are actually covering the gamut of sRGB.

3.6 Object Recognition

Additionally to the extracted labels describing color as explained in Sec. 3.4, we used a Xception (Chollet, 2017) model implemented in *Keras* which was pretrained on the imagenet (Deng et al., 2009) dataset, to extract objects contained in a specific image as well as scene location descriptions. This allows us to filter images by their object content which may not have been extracted by the image captioning system. As for the color labels, these get stored in the SQLite database and can be queried using straight-forward boolean search techniques.

4 Feature Retrieval

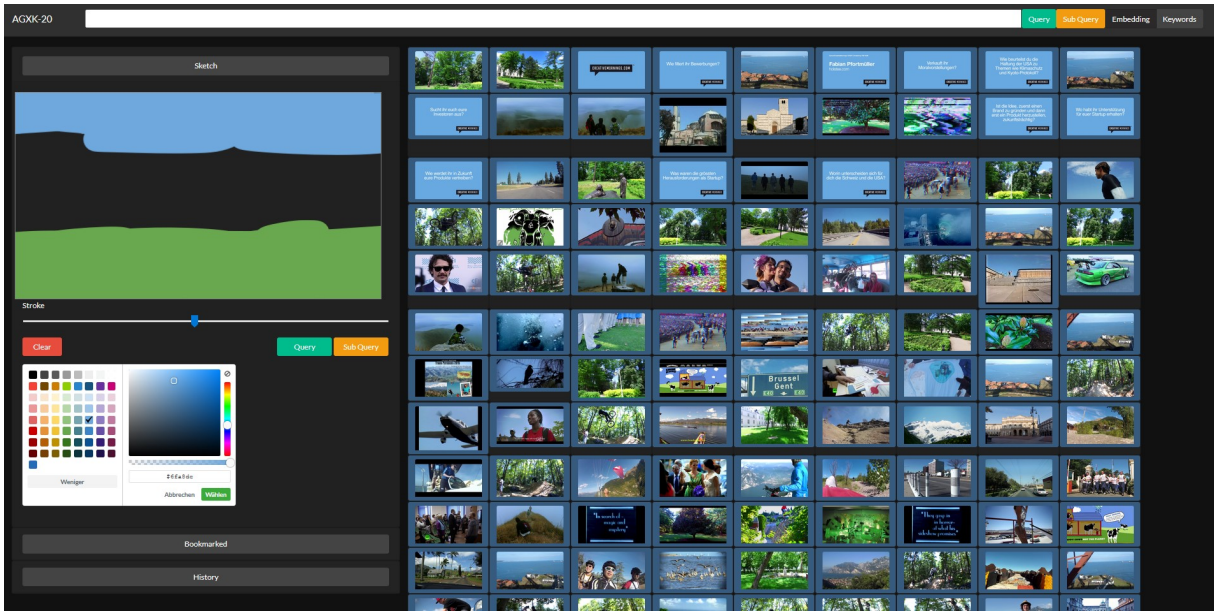


Figure 12: Our retrieval system's user interface.

We compiled our previously described feature vectors into a cohesive interactive retrieval application. Our tool focuses on the advantages of interactivity, namely, it it exploits the user's ability to *sub-query* and filter already retrieved results, this way, the user can narrow down the queried results until the final result is found. It also allows the user to go back and forth in his/her query history as well as using a shared bookmarks section collaborative search within a server session.

In the remainder of this section we explain the different widgets of our application. For each widget, we will discuss in detail what interactions are possible and how these can be used based on a real world example.

4.1 Text search

Figure 13: The searchbar with text input, query/subquery button and toggle for the type of query.

The primary widget to perform a query is a text input field on the top of our page (Figure 13). The user can type any kind of search string to query the database. By pressing *Query* a new search is performed on the server. Alternatively, the user can press *Sub-Query* to filter the already retrieved results with the new search string. The toggle between *Embedding* and *Keywords* provides the option to either query the image captions using the caption embeddings, or to pass a list of keywords, which are compared against the color names (see Sec. 3.4) and the detected objects as explained in Sec. 3.6. Using comma separation, multiple keywords can be added to perform an inner join in the database.

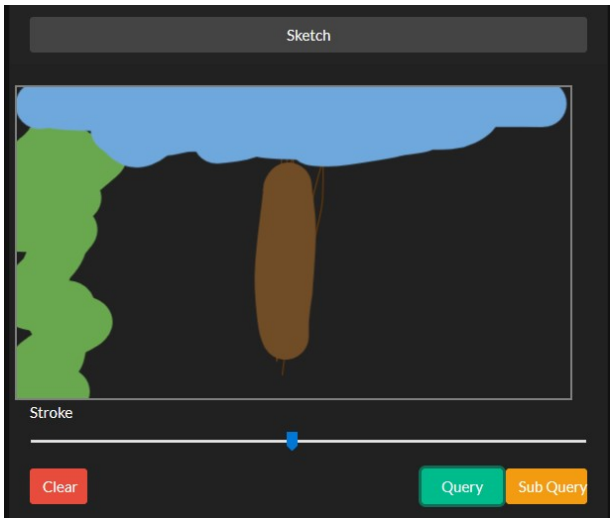


Figure 14: The sketching widget.

4.1.1 Sketch search

Figure 14 displays our sketching widget, which allows the user to draw a colored image. Our system then searches for this image in the database using the color histogram grid system. Again, the widget allows for new *Queries* and *Sub-Queries*. In case of a sub-query, the already retrieved results get sorted by their histogram distance to the sketch drawn in the widget. Grid cells, which are not painted in canvas, are treated as wildcard, i.e. they are not taken into account in the comparison.

4.1.2 Similarity search

Every item retrieved in the results page can also be used to retrieve similar images based on its color content. To do so, a user may hover over a certain item and click on *find similar*. This triggers a histogram comparison on the server, retrieving a list of images with similar grid color histograms. Additionally, the user can click on a second button to retrieve the complete list of shots belonging to the movie.

4.1.3 Search History

The search history allows the user to go back and forth in the already performed queries, this goes in line with our iterative search approach: If a query or sub-query led to a undesired result, one can simply move back to a previous state and continue the search from there.

4.1.4 Collaborative Bookmarks

Finally our tool supports collaborative search by means of a *shared bookmarks* space. When a user clicks on a result on the results region, the image gets added to the bookmarks, from there any user connected to the current session can submit it as a result or use it to retrieve similar images and proceed with sub-querying from there on.

4.2 Results

The results of each the latest query are presented in an image grid, where the depicted image is the middle frame of the shot. When the user hovers over a certain item, he has the possibility to either put the image into the collaborative bookmarks, find images which have a similar color distribution or show all shots of the corresponding movie. When a query returns a distance

metric, the shots are sorted by this in an increasing fashion, if the query returns a boolean list, the shots are ordered depending on the database index of the specific item.

4.3 Implementation

Our back-end has been implemented in *Python 3.6* using *NumPy* for numeric the numerics, *OpenCV* for the image processing and color transformations, *Keras* and *PyTorch* for the deep learning tasks. For the detection of scenes we used *PySceneDetect*. The front-end is implemented in JavaScript using *Bootstrap*, *JQuery* and *Fabric.JS* for the sketch canvas.

5 Results

5.1 End-to-end test

Besides constantly and iteratively testing the front- and backend during the development of the system, we conducted an end-to-end test similar to the procedure of the Video Browser Showdown. In this end-to-end test, we assessed the system on the three tasks - visual known-item search, textual known-item search and ad-hoc video search - and measured the performance of the system in terms of correctly identified videos and scenes as well as the time needed to retrieve the correct items. Before conducting the test, we manually created a test set for all three tasks and set the time limit for completing each search to three minutes. The following table shows the results for all three tasks after three hours of testing.

Results end-to-end test		
Task	Performance	Average Retrieval Time
Visual Known-Item Search	66.6%	71.5s
Textual Known-Item Search	53.6%	58.8s
Ad-Hoc Video Search	ϕ 10.5 videos	-

Table 4: Results of the end-to-end test on all three tasks

Regarding the visual- and textual known-item search tasks, the test showed that our final system performs well on tasks that involve clearly distinguishable search parameters, e.g. 'a landscape of mountains covered in snow with a guy on skis in the background' or 'a guy and a woman standing on top of a mountain looking into the distant'. Typically, we did not only find the corresponding scenes for such tasks with high accuracy, but also very quickly. The average retrieval time for correctly identified scenes, as shown in Table 3, supports this claim as it is far below the three minutes time limit. However, the more ambiguous and less distinguishable the search parameters are, the lower the chances to find the correct video or scene. Regarding the ad-hoc video search task, the system was able to retrieve scenes for 90 % of the tasks, although the individual performance, i.e. how many scenes were detected, was highly dependent on the granularity of the search query. For instance, we did not find scenes with 'children eating ice cream', but retrieved numerous videos of 'people having dinner in a restaurant'.

5.2 Showdown Results

On the 28th of May 2020 we participated in a showdown competition against our fellow students in the course. The competition spanned three hours and involved us completing a selection of the three tasks. It gave both our tool and the designated user a fair challenge and are happy with the results we obtained. Among the other student teams participating we placed second, as can be seen in figure 15. Aside from our placement we are content with our overall performance, so we'll briefly discuss our performance for each task.

Visual Known-Item Search

This task was quite well suited for the search by drawing functionality we had built into the tool. As the competition progressed we did notice that although it functioned correctly, the quickest way to find the prompt was to search for distinct subjects in the scene. If these distinct subjects were also present in the MS COCO data set the chance of finding them increased greatly.

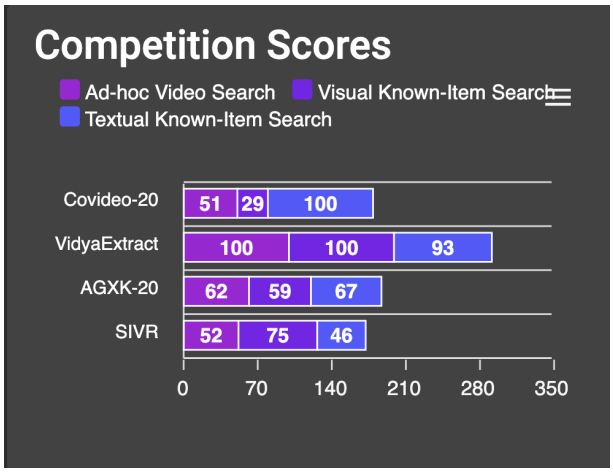


Figure 15: Results when competition ended

Textual Known-Item Search

Similar to the Visual search, we found that when the prompts included subjects that were also found in the MS COCO data set our search time decreased by quite a margin. This was especially clear during the task of finding a scene which included a blue coffee cup on a table. Initially we searched for the surrounding information which included a house with snow, but after switching to searching for the coffee cup we found it quite quickly.

Ad-Hoc Video Search

For this task we experienced little problem finding frames which included subjects mentioned in the prompt. What was noticeable was that for the more subtle parts of the prompts we had some difficulties. As the judges would receive a video segment instead of one frame we would ideally have evaluated whether the prompt held for the entire segment instead of just one frame. Since our tool does not have video scrubbing functionality this issue could not be sufficiently addressed, which resulted in a couple of submitted frames resulting in failed entries. Another functionality that may have been useful is submitting more than one entry at a time, as we found suitable segments we had to submit them one by one. This didn't have significant impact but would be a quality of life improvement for the tool. Overall we were still quite happy with the results for this task, finding the subjects in the prompt worked well and our functionality of finding similar frames came in very useful to our success in this task.

Noteworthy functionality of competitors

The competition was the perfect place to discover what the other students had developed and gain additional perspective on where our tool could improve. The tool fulfilled all the requirements we set out for it, but since no tool is perfect we will describe some areas where we see clear potential for future development.
During the competition it was clear that the functionality of video scrubbing greatly improved the retrieval of relevant video segments. The biggest improvement by implementing this would be in the ad-hoc video search task, as there is an increased chance of finding similar video segments that fit the query in a video which already has one segment that suffices. Another feature that can potentially improve performance is grouping the found frames of the same video together. Since a video which has a significant proportion of relevant frames has a higher chance of including the shot we are searching for. These improvements are areas for future development.

6 Discussion and Conclusion

It has become apparent during the showdown, that our caption model performed adequately in many situations often yielding close results high up the ranking. However, relying heavily on the captioning system also yielded some pitfalls, namely, what to do, when the item searched for is not retrieved by the system. In such cases our tool only provides few options to proceed with the search, either the user can try to search for objects recognized in the image, or he may use some color hints to find the correct item. The latter can seldom be applied to the task where the actual items is not known, since the color content of the image is usually not given, or only sparsely described. In a future improvement of the tool a better mechanism could be provided to find such items which are not retrieved by the captioning or the object recognition model.

A second problem we encountered has been, that our tool is not able to show the actual shot the image depicts in the result view. This often made it hard to determine the correct location within the movie. This problem became especially important in movies, where pretty much all shots looked very similar. Often the detail which identifies one shot as the correct one has not even been within the frame which we analysed but may be some seconds before or after, thus finding the correct result, especially for the *textual-known-item search* often pure resulted trial and error. This could have been improved by providing a way to actually scrub through the movie using a player widget. The reason we didn't implement this feature has however been a practical one: We assumed everybody needs to participate during the showdown, but not everybody of our team had enough storage on his computer to keep all movie files locally, thus we opted for a architecture, where the movies were not needed.

We have to admit that we didn't use the color-based search functionality too often during the showdown. The reasons are multi-fold: first, the color histogram may not be a good descriptor for the sketch-based search, mainly because it can be sensitive to the actually picked colors. To counter this problem we tried to tune its bins during the development of the system. Even though the sketch-based search can retrieve the correct image for the visual known-item search, the problem is that we do not actually know, which frame of the shown item has been analysed for our database. Nevertheless, the search based on color similarity has often been useful to find similar shots which do not belong to the same movie. This has also been one of our coping strategies with the problem of not retrieving the correct images using our caption system, as explained earlier.

A minor problem, which played an important role when the result was found in a short amount of time by all groups, was the fact that our tool did not provide a way to *directly* submit a result. Instead, a frame always had to be added to the bookmarks first, and then had to be clicked again to post a result to the VBS server. This has sometimes been limiting, especially in the ad-hoc search task.

Concluding, we note that we are happy with our final product. While there are still open questions to solve and functionalities to tweak, we were surprised, how efficient our captioning system performed during the showdown. The implementation of a search history has been very valuable during the showdown, as it allowed to quickly check another query for a better result and, if unsuccessful, navigate back to the previous one. We also think that our collaborative search would have been very helpful and our performance would have been a notch better with it.

References

Aibing Rao, Srihari, R. K., & Zhongfei Zhang. (1999). Spatial color histograms for content-based image retrieval, In *Proceedings 11th international conference on tools with artificial intelligence*.

Ho Young Lee, Ho Keun Lee, & Yeong Ho Ha. (2003). Spatial color descriptor for image retrieval and video segmentation. *IEEE Transactions on Multimedia*, *5*(3), 358–367.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database, In *2009 IEEE conference on computer vision and pattern recognition*, Ieee.

Lindner, A., Li, B. Z., Bonnier, N., & Süsstrunk, S. (2012). A large-scale multi-lingual color thesaurus [Issue: 1], In *Color and Imaging Conference*, Society for Imaging Science; Technology. Issue: 1.

Van den Bergh, M., Boix, X., Roig, G., de Capitani, B., & Van Gool, L. (2012). Seeds: Superpixels extracted via energy-driven sampling, In *European conference on computer vision*. Springer.

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2014). Microsoft coco: Common objects in context.

Young, P., Lai, A., Hodosh, M., & Hockenmaier, J. (2014). From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, *2*, 67–78. https://doi.org/10.1162/tacl_a_00166

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions, In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2017). Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(4), 652–663.

Vinodababu, S. (2018). *Show, attend, and tell | a pytorch tutorial to image captioning*. https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning (accessed: 04.06.2020)

Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks.

Rossetto, L., Schuldt, H., Awad, G., & Butt, A. A. (2019). V3c–a research video collection, In *International conference on multimedia modeling*. Springer.