Generating Video thumbnails based on Web Scraped Data Applied Deep Learning - Final Report

Maximilian Michel
TU Vienna
e01552754@student.tuwien.ac.at

1. Introduction

The main goal for this project was to create and train a network that can generate passable YouTube video thumbnails based on the title. The network in question was to be structured similarly to what is described in Reed et al. (2016), Generative Adversarial Text to Image Synthesis [1].

Their method proposes training of a Generative Adversarial Network (GAN) to synthesize pictures based on vector embeddings of a sentence describing the picture contents. This approach was followed regarding the network structure, but using a different method to generate the word embeddings for its input. In the past I achieved good results[2] using pre-trained word vectors, provided for example as part of the fastText[1] library, as input for neural networks. This seems to me to be a logical decision as it reduces the scale of the project in an aspect that is not its primary scope.

To set a reasonable goal and to quantify the concept of a "passable" image two thresholds were defined. The first was the percentage of percentage of images correctly classified as real by the dicriminator. The second was the percentage of pictures created by the generator that could fool the discriminator. Each of these values alone does not identify a model as 'good', but in conjunction they create a quantifyable standard to work towards. The set goal for this project was an accuracy of 80% for the discriminator and an accuracy of 65% of the generator.

2. Method

2.1. Data Collection

The data set created contains real video thumbnails in combination with their title as found on YouTube.com. Additionally, a vector embedding of the cleaned title is included in the final data set. Data was collected using the API provided by YouTube itself. Due to the large diversity of Video genres on YouTube a sub selection of search terms was chosen to serve as a basis for data collection. This was done in hope of enabling the network to find patterns more easily and also keeping the size of the training set at a reasonable level. Search Topics included Vlogs, cats, dogs, music, cars, fishing, restoration, tutorial,tech, travel, ASMR, lifestyle, Beauty, DIY, and learning. The topics were chosen based on a list of popular YouTube genres.

Data collection was done in five steps, each step is performed in a separate python notebook.

2.1.1. Scraping This notebook handles interaction with the YT API in order to gather the raw data. It includes multiple ways to scrape data. Due to Youtubes API quota not all ways to search are equally feasible. Searching for keywords is very costly and could only generate around 5000 videos per day. Searching for playlists by keyword and then gathering videos from the playlists is very inexpensive but introduces some duplicates. Efficiency of this method is further improved by letting the search results be ordered by the amount of videos in the playlists. This does not result in additional costs but increases the number of videos scraped per keyword search.

2.1.2. Removing Duplicates Searching playlists instead of directly searching for videos resulted in some duplicate entries in the scraped data. Entries with duplicate video IDs were removed. Entries where the titles match but the video ID is differnt remain in the dataset.

2.1.3. Getting Images Youtube does not provide the thumbnail picture itself but rather URLs to the picture in different resolutions. Therefore, the thumbnail of each video in the dataset had to be downloaded separately. The lowest resolution was chosen to aide in training the network. This can easily be changed changed for future experiments.

2.1.4. Clean Up Following the removal of duplicates are several cleaning steps. First, the instances that did not include a valid thumbnail link are removed. Second, the title text is cleaned and standardized to make it ready for vectorization. This includes casting all titles to lower case, replacing emojis with their test counter parts and removing unnecessary white-space.

[1] https://fasttext.cc/

2.1.5. Vectorization The Python library FastText was used in combination with a pre-trained model available from their website to translate the title strings in the dataset to word vector representations. To make the process of translation quicker, a vocabulary dictionary of all words in the dataset was created, these were then translated to vector representations. Vocabulary and translation form a lookup table which was then used to add a word vector column to the data set. Each word corresponds to a 25 dimensional vector and a title contains at most 64 words.

The final dataset CSV files and a folder with containing the scraped pictures in low quality was mada publicly available by uploading it to archive.org[2]. As it stands the data encompasses 448789 unique videos, their thumbnail in a resolution of 120x90 and 25x64 word vector each.

2.2. Image Generation

A Generative Adversarial Network was used to generate Video Thumbnails based on a vector representation of the corresponding title. This Models architecture follows the structure of a GAN for created for Image Synthesis[1]. The Model described by Reed et al. consists of a generator with four deconvolution layers and a discriminator with four convolution layer. Additionally both include a linear compression layer to reduce the text input vector to a smaller scale. This structure in explained well but when it comes to window dimension, stride values and channels the researchers do not go into much detail. For the model in this project (ThumbGan) the same underlying structure was used and several parameters regarding the convolution and deconvolution layers were tried.

2.2.1. The Generator of the ThumbGan Model starts with a compression layer. This fully connected layer reduces the size of the input from 30x64 (25 Words vectors + 5 noise vectors) to 192 units. The subsequent four deconvolution layers have kernel size 3,4,4 and 7 and a stride size of 2,2,2 and 1. Some padding was used to arrive at an output dimension of 90x120 which fits the size of the input images. The number of channels used in each layer was increased over time but the final model had channels sizes of 128, 64, 32 and 3. The last one was not changes since it represents the three color values of the output. The 'ReLU' was used as activation function for all layers except the last one which used a 'Sigmoid' activation. The choice of activation functions was based on recommendations in Unsupervised representation learning with deep convolutional generative adversarial networks (2015)[3].

2.2.2. The Discriminator also encompasses 5 Layers. It starts with two convolution layers that take as input either generated or the real images. After that a third convolution layer is fed a combination of the output of the second layer and a compressed version of the input text vector the current image corresponds to. This compression is done in a separate fully connected layer the takes as input the word vectors of a title but without noise as in the generator. Lastly a final fully connected layer converts the previous output to a single neuron, that determines if a picture is real or fake. All convolution layers have a kernel size of 4 and a padding of 2. The stride sizes are 2,3 and 2. Channel sizes are smaller in the discriminator ranging between 25 and 50. Again based on recommendation[3] for the discriminators activations the 'leakyReLU' was used.

2.2.3. For Normalization a Batch-Normalization layer was added between each of the layers and their activation function to keep the weights from accumulating large values which slows down training and could lead to memory issues.

2.2.4. The Optimizer used was 'Adam'. Different parameters were tried, mainly based on descriptions from related literature [3, 1] but in the end the default parameters of a learning rate of 0.001 and betas 0.9 and 0.999 worked best.

2.2.5. Loss was calculated in the traditional[4] way by letting Generator and Discriminator compete in a Minimax Game. The discriminators loss was calculated as d_loss = E[D(x)] + E[1-D(z)], where D() is the discriminator network and x and z are the real images and fake images plus the relevant word vector. The generators loss was calculated as g_loss = -D(z) trying to maximize the probability of generating a real looking image.

2.3. Training

The model was first trained using a batch size of 128. This became a problem later as it lead to memory issues with the graphics card used in training. The batch size was subsequently reduced to 64 for all further runs. The Models were trained for around 25000 batches, before deciding to change parameters. Several subsets created from the large amount of data collected were used in training in hopes of getting better results due to a smaller size or less variation (subsamples with specific keywords in the title). Combinations of model parameters and data sets that worked well were trained longer, between 50,000 and 90,000 batch iterations. With a data set comprising 10,000 images this results in between 450 and 800 training epochs.

2.4. Results

With the large data sets, 100% 50% and 25% of all available training images, the model tended to oscillate

[2]https://archive.org/details/YT-Thumbnail-DataSet

between generating lighter shapes and noise (see Figure 1). The model did however produce dark bars on the top and bottom of most pictures. This is caused by YouTube resizing video frames to a ratio of 4:3 when using them for thumbnails substituting the empty space with black. Though not really a necessary part of a thumbnail it showed that the model pipeline was working and that the noise was not the result of a mistake in the setup but of a poorly trained network.

As longer training times did not improve the issue significantly, the next step was to reduce the size of the input data in hopes of limiting the inputs variance. This was done by filtering the data set for certain keywords in the title. An 'Animal' data set was created that contained around 15,000 pictures. As this set contained only videos with titles including words like 'cat', 'dog' or 'animals' it was assumed that the variance in pictures was reduced. When inspecting the set it turned out that this was only true to a certain degree. The data still included many thumbnails that did not necessarily to their animal themed title additionally it still included many titles that had animal related words in them but did not necessarily correspond to the topic of animals. Nonetheless, using this data set reduced oscillation. The generator did still not create anything resembling the input pictures however.

It was further noticed that the discriminator had far higher loss values than the generator (see Figure 2. Therefore, the channel size was increased almost doubling their values for all convolution layers. Despite increasing training time significantly, it did lead to more clearly defined shapes in the generators output as well as a more balanced use of colors, resembling the color values of the input pictures far better.

It also caused the imbalance of loss values to shift to the generator. In hopes of decreasing this imbalance the channel size was increased in the deconvolution layers as well. This again did help the quality of images produced but the imbalance could not be evened out before running into memory allocation issues on the GPU (4GB) used for training. The final imbalance of the two competing networks is not to drastic as can be seen from Figure 3.

The final results of training the Model on the scaled down, cat themed version of the data set can be seen in Figure 4. In the bottom row and example picture can be seen that corresponds to the word vector used in training. The rows above it show results for the same ten input vectors through the training at 150, 2000, 8000, 32,150 and finally at 50,000 batch iterations. This corresponds to around 500 epochs over data. The model improves seems to be learning better than before creating shapes and color values that more closely resemble thumbnails, but are in still very for away from generating actual images. Some oscillation in the hue is still present but the shapes stay rather consistently. Additionally the network does not degrade to only producing one type of output image as it did before.

2.5. Discussion

In training it became clear that the data set probably included to much variation to for the network to handle. In collecting the data I wanted to cover a representative cross section of the content available on YouTube and not just cat pictures, but it seems that a closer range in topics can help significantly. Further I underestimated how little YouTube titles have in common with the thumbnail pictures used. Eben a human would have a terribly hard time deciding which thumbnail corresponds with which title without seeing the actual video.

Additionally to troubles with the training performance I had significant issues with memory leakage. Until I finally found the cause, not applying .detach() on the loss values I recorded, this hindered the amount of iterations I could run before the system crashed. This also meant that I only started to take larger more complex models and longer training times into consideration later in the project.

I still feel that a GAN architecture is the right approach to solve this problem but I would switch to using tags instead of titles as a basis for generation. I wanted to also scrape tags with the titles and video URLs but my trick to get around the Quota limits also meant that I did not have access to the as they required an extra call to the API which would have made the work-around redundant.

The stopping Criteria defined in the beginning were set too high as they were not reachable by the model. As said before this is not an issue with the model however but with the choice of data. I would like to try running the same experiment again with data including video tags and that follow a general theme. Not as narrow as just cats but maybe concentrating on only one genre.

My time management was mainly hindered by unexpectedly large stepping stones like the Quota limit while aggregating the data and the memory leak issues in training. This threw off my time table quite considerably and meant that I had to stop more because of time restraints than based on the aforementioned mentioned stopping criteria.

References

[1] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," in *International Conference on Machine Learning*, pp. 1060–1069, PMLR, 2016.

[2] M. Michel, D. Djurica, and J. Mendling, "Identification of decision rules from legislative documents using machine learning and natural language processing,"

[3] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

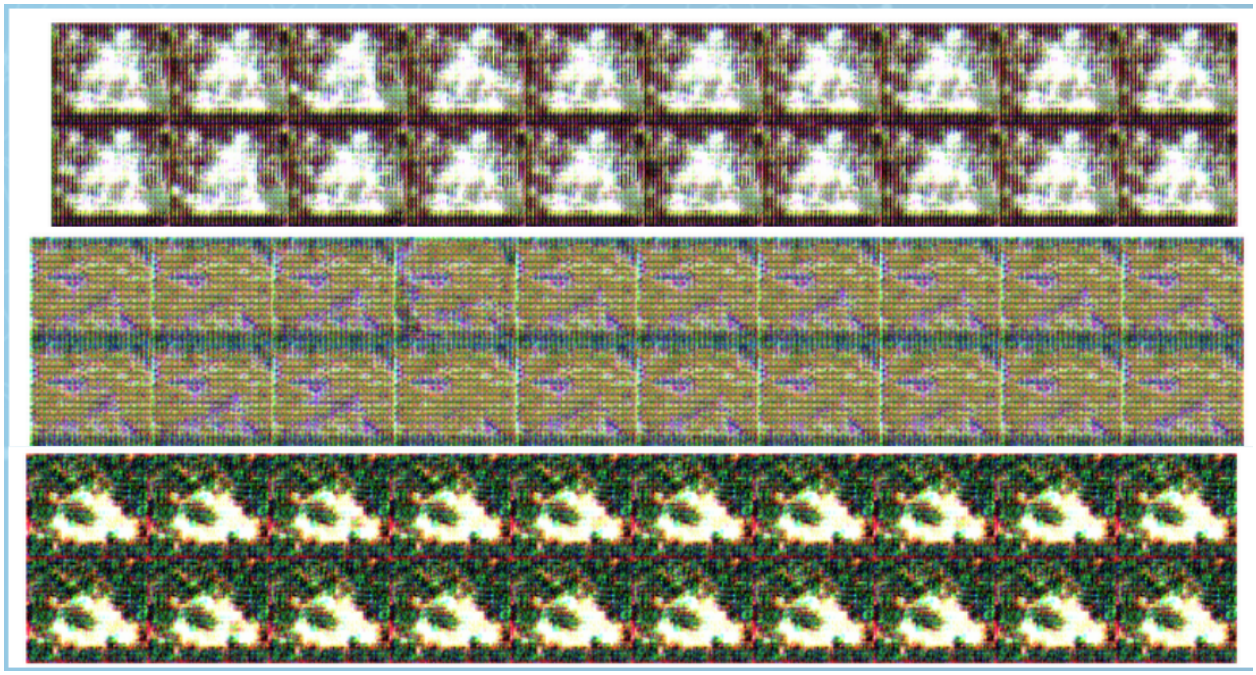
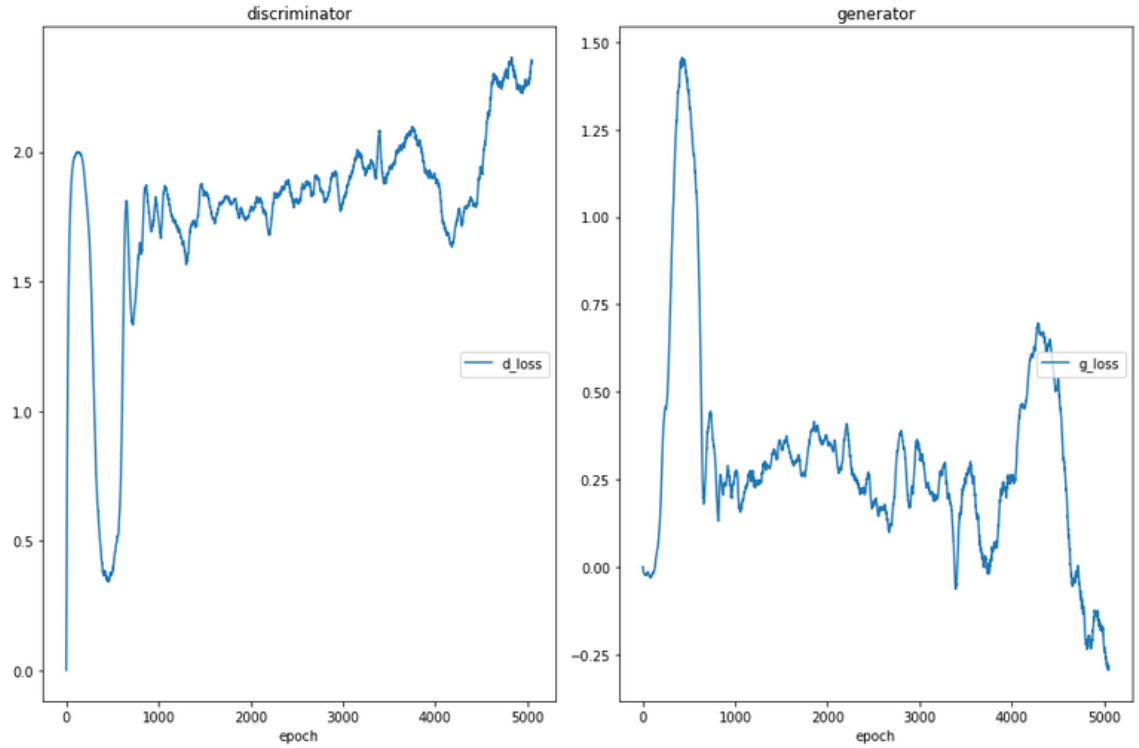


Figure 1. Images Generated based on large full Data Set




```
discriminator
        d_loss                  (min:    0.000, max:    2.365, cur:    2.343)
generator
        g_loss                  (min:   -0.296, max:    1.457, cur:   -0.285)
```

```
(d_loss= 2.503696, g_loss= 0.128631):  10%| | 5075/50000 [42:33<6:18:02,  1.98it
```

Figure 2. Loss of Discriminator and Generator through training

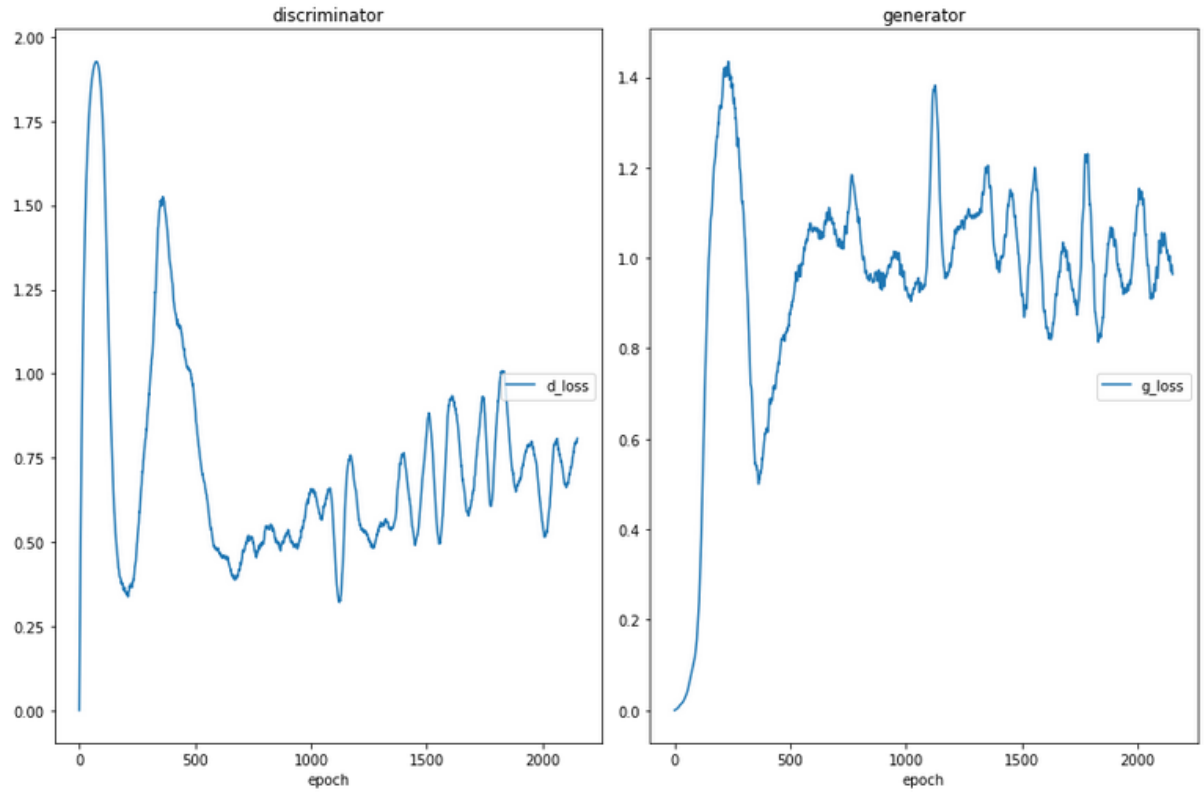


Figure 3. Unequal Loss of Discriminator and Generator while training

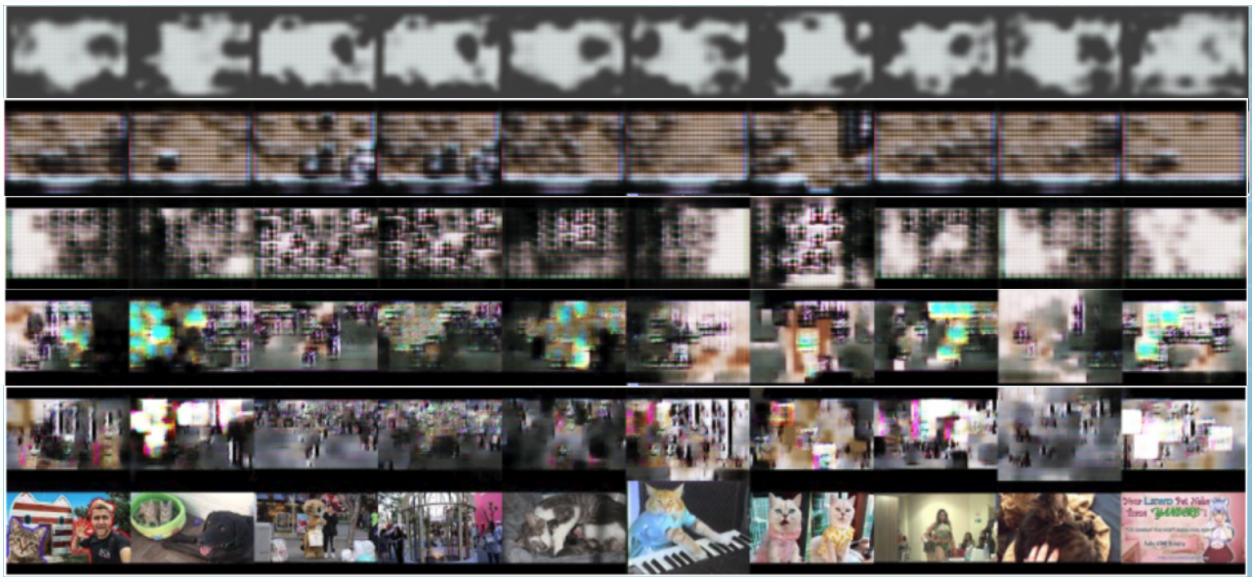


Figure 4. Results trough training: Last row is the original for comparison.