Deep Learning Based Video Compression

Kang Da Ji and Helmut Hlavacs(✉)

University of Vienna, Research Group Entertainment Computing Vienna, Vienna, Austria
helmut.hlavacs@univie.ac.at

Abstract. Our goal is to test the capability of deep learning for compressing the size of video files, e.g., for sending them over digital networks. This is done by extracting keypoint and affine transformation tensors, using a pre-trained face model and then reducing the data by quantization and compression. This minimal information is sent through a network together with full source images used as starting frames for our approach.

The receiver device then reconstructs the video with a generator and a keypoint detector, by transforming and animating the keypoints of the source image according to the video keypoints. We minimized the required data by using LZMA2 compression and a quantization factor of 10 000 for keypoints and 1 000 for transformations.

Lastly, we determined limitations of this approach and found that in regard to file size reduction, our approach was noticeably better, while the quality of the resulting video in comparison to the original one was only half as good.

Keywords: Deep learning · Video compression · Video reconstruction

1 Motivation

Due to the 2020 pandemic, the quantity of video data sent over the Internet has dramatically increased, be it for interactive video conferences, game streaming, video clip and movie streaming, or a multitude of many other applications [1].

Currently, codecs for encoding and decoding videos are commonly used for transmitting videos by compressing the video files for faster transmission to the user. However, by using machine learning, it might be possible to reduce the size of video data even further than with video codecs, while only suffering little quality losses.

The main goal of this work is to implement a tool that takes a few frames and is able to reconstruct the original video with little additional information. The neuronal network should be trained on a certain video or a category of videos and be able to reproduce them so that the result is similar to the original video.

We thus implemented a system that is based on Aliaksandr Siarohin's implementation of his "First Order Motion Model for Image Animation" approach [17,18], which extracts the keypoint information from the source image and

Z. Lv and H. Song (Eds.): INTETAIN 2021, LNICST 429, pp. 127–141, 2022.
https://doi.org/10.1007/978-3-030-99188-3_8

sends these pieces of information and the source image itself to the receiver. We modified Aliaksandr Siarohin's project by including quantization and zip compression methods to reduce the scope of the required keypoint information. By doing this, we were able to generate a video that is similar to the original video, while sending less data than codecs like AV1 and H.264 [2,3,10].

These results were further evaluated by us and compared to the AV1 and H.264 video codecs. Additionally, we analyzed the limitations of the system and formulated what type of videos this project works well on and what should be avoided.

2 Related Work

There is a similar idea to this project coming from Wang et al., a software product called "One-Shot Free-View Neural Talking-Head Synthesis for Video Conferencing" [20]. Wang et al. also extracted the keypoints of a video for use in fast and high quality video conferencing. They took the first frame and discarded the rest of the video. With only using the head position and facial expression, they upsampled the frames and created a video that is similar in quality to the original one. Even though our project sounds similar to this software product, there are several differences.

First, Wang et al. [20] focused on real time video conferencing, while our project aims to send as little data as possible over the network. We are condensing the keypoint and transformation data with the LZMA2 compression algorithm and quantize it, so they take even less bandwidth to transfer.

Secondly, we do not have to limit ourselves to generating face videos. According to the First Order Motion Model [17,18], we can train a model to use taichi movements instead of human faces. Generally, any structured motion with clear movements can be used, as long as it is trained properly. This is an advantage over something that will mainly be utilized as a video conferencing tool or a generator for face videos.

Another technology that is similar, are deep learning video compression algorithms like the one from Djelouah et al. [7]. They try to encode and decode an arbitrary video by decoding the information into latent space, which represents motion and blending coefficients. To mitigate prediction errors, they still require residuals between the original frame and the interpolated one. This rivals the efficiency and quality of state-of-the-art codecs, namely H.264 and H.265, with comparable bits per pixel.

But while they have better quality than codecs, this paper focuses more on quality retention than on data reduction by encoding and decoding it more efficiently. Our main aim is to reduce the file size while retaining as much quality as possible.

Moreover, Kazantsev et al. [12] suggested the much more straight forward idea of using machine learning to tune the encoder configurations for a given codec, treating it as an optimization problem. According to their paper, this would save 9–20% of the bitrate compared to the original H.264 encoded video.

It can also be used for other codecs, but since there is no training data available, one has to collect their own. This approach can be used to complement existing and future codecs.

Using codecs is not the only way of reducing data. By using a procedure called "Video Inbetweening" it is also possible to reduce the transmitted data. This type of technique focuses on interpolating images between two adjacent frames, raising the frames per second. A sender could split a video into its frames and send every second image. The receiver can then synthesize the missing frames and reproduce the original video. Such an inbetweening systems was proposed by Wu et al. [21]. Systems only implementing the inbetweening algorithm were for example proposed by Bao et al. [6] or Li et al.[14].

3 Implementation

First of all, the service "Google Colab" was used as the environment of our project to utilize their GPU/TPU and CPU resources. Since extensive computation was expected, we decided to outsource it to Google Colab, to speed up the development process and prevent issues that are linked to hardware requirements. This enabled us to focus less on the setup of the project and more on how to achieve our goals. Furthermore, many machine learning projects already included a demo for Google Colab in their GitHub repository, which reduced the research time to find a suitable basis for the development process.

Secondly, we used a GitHub repository from Siarohin et al. called "First Order Motion Model for Image Animation" [17,18]. We decided to use their work, because, like several other machine learning projects, they already had a demonstration on Colab where we were able to test its capabilities. This also included a few pre-trained neuronal network models that would otherwise require extensive time, experience and resources to train. It turned out his project was working with keypoints, which gave us the idea of trying to send this data over the network, similar to Wang et al. [20].

Another service we utilized is letsenhance.io [13]. It is a neuronal network-based web service that charges a fee or is only usable as a demo, that is able to enhance images without changing the resolution of the picture. Although the exact network architecture is not specified, the website claims that they use a neuronal network which is able to upscale images and remove compression artifacts of pictures. Any other service we found was focused on upscaling the picture or did not work as well as we wanted, for example, they did not sharpen the image like letsenhance.io did without changing the resolution. The utilization of said service improved the output image quality. This was necessary since the 256256 image was often blurry, which resulted in even blurrier results in our project. By enhancing the source image, we were able to counteract this partially.

One of the tools we utilized was the "MSU Video Quality Measurement Tool 13.1" [8], which helped evaluate data. This tool is able to compare two videos and ascertain their differences in quality. The VQMT tool is very helpful in detecting artifacts and inaccuracies that are not visible to the naked eye, which is achieved by using a value called "Peak signal-to-noise ratio" in each frame.

7zip was our preferred zipping software. Unlike the standard zipping method, we were able to use different and more up-to-date compression algorithms. By testing various algorithms, we figured out that LZMA2, which is an improved version of the LZMA compression algorithm, outperformed the other algorithms and produced significantly smaller files than the standard zipping method [4,5].

Lastly, we used the video editing software FFmpeg, more specifically the version 2021-07-04-git-301d275301-full_build. This tool allowed us to manually extract frames from the video, convert videos to another codec, change the resolution, cut videos, and remove audio tracks. It is a powerful and versatile tool for editing videos without using a complicated interface, and is able to perform operations normal video editing tools are not capable of.

4 Design and Approach

The "First Order Motion Model for Image Animation" project by Siarohin et al. was used as the basis for our project [17,18]. This project uses a motion module which can determine which pixels correspond to which object and a generation module that is able to use the information of the motion module and generate an image based on it. The challenge was to find a way to extract the information that contains fewer data than the actual video itself.

Our reason for choosing this project is that contrary to images, binary files are far easier to compress, because images are often already heavily reduced. Since standard zipping is not efficient enough, we often used LZMA2 compression, which performed better than other zip formats [4,5]. The compressed data size was only a fraction of the original video, which was encoded with an AV1 codec [3].

Figure 1 shows the process of our project. We took one frame from a driving video and a source frame. Driving frames are the original video frames processed into readable data. Afterwards, we detected the keypoints and local affine transformations of both images. The local affine transformations were used for object motion in the neighborhood of each predicted keypoint and acted as a guide to describe the motion of each keypoint.

Important to note is that even though the keypoint of a color in the visualization has a consistent placement on the face, it does not mean that it will necessarily have the same color placement on a different face. The keypoints only need to be understandable by the neuronal network, and does not necessarily have to make sense to the user.

The next step was to use relative motion transfer, so we were able to extract the image movements from the driving frame to the source frame. Relative motion was done by comparing the motion of the first frame of the driving video and the currently processed frame and transferring this difference to the source image. By doing this, each frame could then be processed independently of the other video frames.

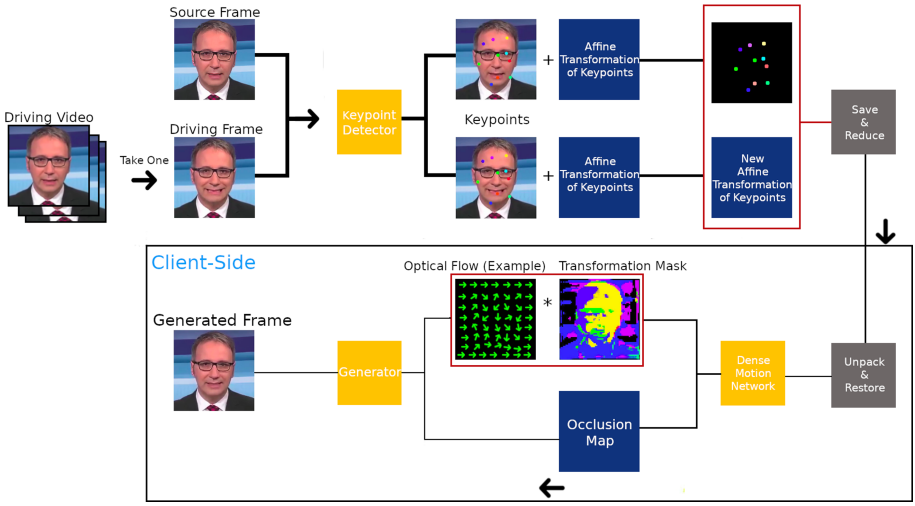

Fig. 1. Overview of the frame generation process

After generating the output from the relative motion transfer, we saved and quantized it, turned them into binary files, and compressed the output using LZMA2 compression. These files were then sent to the receiver to be decompressed, loaded and reversed to approximately its original float values.

The dense motion network continued to generate an optical flow and a transformation mask, which together told us which areas had to move where and how far. While the optical flow described the motion of a frame, the transformation mask indicated which areas were affected. The occlusion map dictated which areas needed to be inpainted by the generator and which areas just needed to be warped. This information was then used to generate the final output, a frame that was close to being identical to the frame it tried to duplicate, while only sending a fraction of the size to the receiver.

This process only shows how one frame will be generated. This needs to be done for each frame of the driving video, which requires more time and resources the longer the video gets. If it reads too many frames at once, it will overstrain the RAM, which is applicable to long videos. This is why we divide the video into multiple smaller segments and discard the frames that are already generated to clear out RAM space.

5 Theoretical Background

The "First Order Motion Model for Image Animation" that we use utilizes a U-Net for keypoint detection and dense motion prediction, which are a part of the motion module [17]. U-Nets are a subtype of convolution neural network that is used for image segmentation. More specifically, semantic segmentation, which means that the neural network not only knows what is inside the picture, but

where the corresponding pixels related to the object are on [15]. This is a stark difference to CNNs, which can only determine what type of object the picture contains.

In the project, this U-Net architecture is used for grouping pixels in the image into heatmaps, where each heatmap correspond to one keypoint. These heatmaps indicate which areas are affected by the transformation, when the keypoint is moved [17].

Additionally, the project also uses the Johnson architecture for training the model, which is a deep residual convolutional neural network. A residual CNN is helpful to recognize more complex images that require many layers. Johnson et al. uses multiple loss functions that outputs a scalar, which evaluates the difference between input image and target image [11].

Contrary to other image comparison approaches, it uses perceptual comparison instead of pixel by pixel comparison. A perceptual comparison determines how similar two pictures are by comparing the style and content, while a pixel by pixel comparison only checks if the pixels in the same position are similar. This would mean that even though an image that would only have an offset of one pixel, pixel by pixel would detect it as a completely different image, while a perceptual comparison will detect it as the same [11].

For image generation, a specialized generation module is used, which is able to warp the features of an object in an image. Similar warping techniques can be found in [9,16,19]. This architecture allows us, if properly trained, to generate an image that warps the object of the foreground according to an optical flow map, a transformation mask and an occlusion map.

6 Result Evaluation and Discussion

6.1 File Size Comparison

The aim of our project was to compress the video file in such a way that we only needed to send little data to a receiver. We compared our LZMA2 files that contain kp_norm.pickle files to the standard H.264 encoding and AV1 encoding to determine how much data is needed to be transmitted [3,10].

Figure 3 shows the difference in file size. The nicknames of the used videos are written on the x-axis, while the file size is on the y-axis. All videos here are only a few seconds long and contain less than 2000 frames. The computation time of each of them lasted a few minutes and generated a single kp_norm.pickle file, which is a binary representation of the keypoints and their affine transformations. The kp_norm.pickle file was then zipped by the LZMA2 algorithm to reduce the file size further [4].

As shown in the bar chart, our approach approximately stayed around 20 and 40 KB for each video, while the AV1 encodings were noticeably bigger. This showed, that short videos were much more effectively than other by state-of-the-art encodings compressed. In these experiments we used a factor of 10 000 for keypoints and a factor of 1 000 for the affine transformations for quantization,

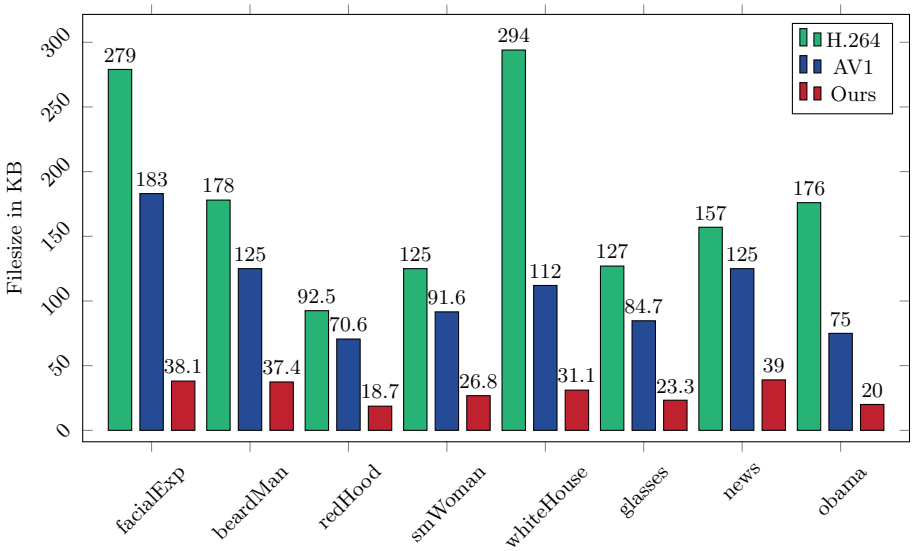

Fig. 2. Comparison between different transferred data sizes. Quantization Ours: 10 000 keypoints, 1 000 transformations. The source image file is not considered here.

but the source images that we used for generation were not considered in the bar charts, since the file size depends on the image type.

We further tested the file size reduction of our approach to see if the video size reduction also works for longer videos. By looping the same video, we extended the video length, since it was difficult to find longer videos that fit our requirements. For example, a person moving out of the picture or the camera changing positions would drastically reduce the quality of the generated videos. As we can see in Fig. 3 our approach performs even better, when the videos are longer. Here we managed to reduce the video size to 1/10 compared to the AV1 encoding on the longNews and even more on the longObama video.

These experiments helped answer the question, if we are able to find an artificial intelligence that can generate files that are much smaller than the original video. In this area, our approach was very efficient and not only beat the older H.264 encoding, but even beat the cutting edge AV1 encoding. The downside is that the process of creating and sending these files is not fully automated, which can lead to errors and consume more time.

The user should also consider that the source image can also take a few kilobytes depending on the image format, so the image format should be picked carefully, according to the user's aim. While .png are compressed losslessly and give better quality, these formats are a few times bigger than .jpg files, which are considerably smaller but use a lossy compression.

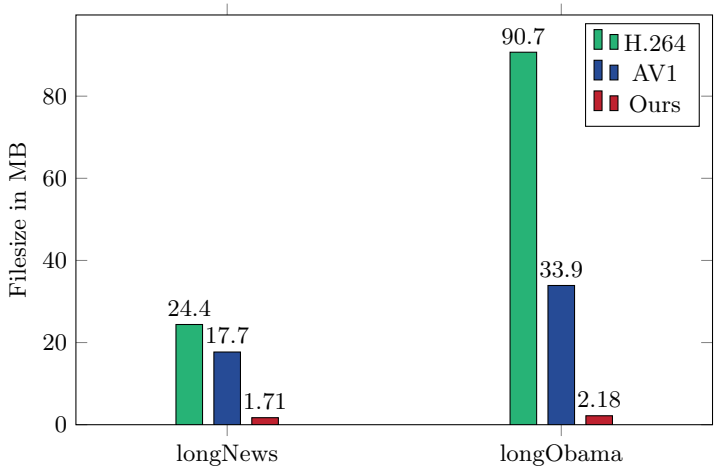

Fig. 3. Comparison between different transferred data sizes for longer videos. longNews: 46 min 05 sec, longObama: 1h 17 min 44 sec. Quantization is 10 000 for keypoints and transformations. The source image file is not considered here.

6.2 Quality Measurement

Besides comparing the file size, we also need to consider the image quality of the generated video. Here we used the MSU Video Quality Measurement tool, VQMT for short, to compare the difference between the original video in H.264 and our generated video [8].

Figure 4, an error bar chart, shows us how much AV1 reduces the quality of a H.264 video and how much our approach does.

As previously mentioned, we used a tool called letsenhance.io to improve our image quality by sharpening edges of the source image and making it less blurry without changing the 256×256 resolution. Although image enhancement is not required to get a result, the data that is collected in this work used enhanced images from said website. While it enhances the output, the difference in quality might be marginal. If there is a method to sharpen an image drastically without upscaling or changing the resolution of 256×256, the quality of the output should be equal to the ones that were used in this project.

VQMT is able to determine the "Peak signal-to-noise ratio" (PSNR) which is used to quantify reconstruction quality of a video in each frame. The higher the PSNR value is, the more similar the frames are to each other. The highest PSNR value is 100 which indicates that they are identical.

On the y-axis we can see the mean of each video comparison, and on the x-axis we see the nicknames of the videos. The lines above each bar represent the standard deviation of the mean PSNR value. The error bar chart shows us that our approach is on average only approximately half as good as the AV1 encoding.

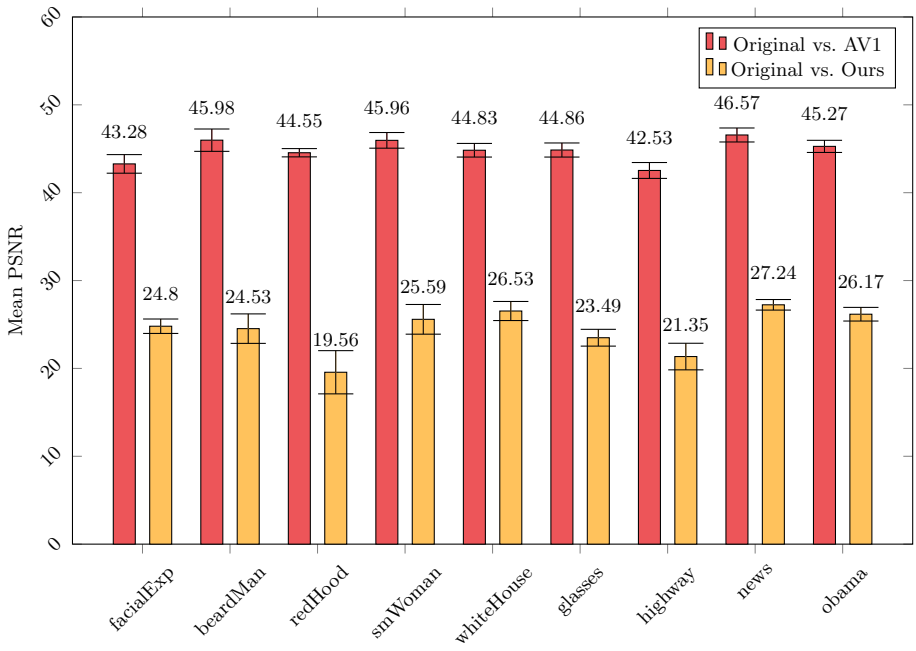

Fig. 4. Quality comparison with the MSU tool. Difference between AV1 encoding quality and Our encoding quality. Max PSNR is 100. Quantization Ours: 10 000 keypoints, 1 000 transformations.

Unfortunately, when we compare the videos side by side, they perceptually look very similar, but sometimes there are major artifacts like weird face and background warping and inaccurate facial movements as seen in Fig. 7. This happens more often on longer videos, since we are using segmentation, which does not always start in the same position as the original frame. This causes the project to misinterpret the movement and translate the motion incorrectly. This can be mitigated by choosing a video with little movement and similar positions throughout the video.

The best generations that were perceptually similar were the news anchor, obama and the facialExpression videos. The news anchor had almost no artifacts, besides the fact that the blinking was not working properly. The obama video had lighting issues, since the generator was not able to learn that the lighting would change when the head moves and the background also was warped by the movement of the head. And lastly, the facialExpression video had issues in filling in hair information that was outside the image and could not reproduce all facial expressions accurately, but it was still very well generated.

Other videos were also performing well, since they perceptually looked similar to the original video, but often had big issues such as weird distortions and inaccuracies that were visible even without knowing the driving video. However,

Fig. 5. Generated and original video frames. Green (top row) are the original frames, blue (bottom row) are the generated. (Color figure online)

most of the problems can be resolved by carefully choosing the driving video and understanding what the underlying limitations are.

Most of the time, the generated video resembles the original, as seen in Fig. 5, but it still has drawbacks in quality compared to AV1 or other codecs. Even though our goal to have minimal quality loss is not met entirely, the generated video is still perceptually close to the source.

6.3 Quantization

Since we used the method of quantization, by dividing the keypoint values and cut off the last few numbers, we reduced the accuracy of the keypoints and affine transformations. This has an impact on the quality of the generated video.

We tried different quantization factors depending on the maximal signed integer value that the int format is able to hold. For example, the int16 format can only assume values from –32,768 to 32,767.

We concluded that any power of ten under 1 000, no matter if keypoint or affine transformation tensors, resulted in much more visible artifacts. These artifacts were often noticeable on edges of the person, like the shoulders or the face. In each frame, the edges of the person seemed to shift rapidly and then shift back to their correct position, which looks like the person "glitched". This was not very noticeable if one stopped in a single frame, but it was visible when looking at it as a video. The smaller the power of ten it was, the more artifacts appeared. This is why we had to find a factor that made the file as small as possible, while not creating more artifacts.

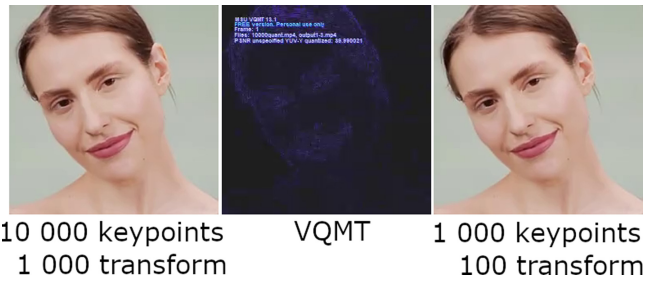

Fig. 6. The difference in quantization shown with the MSU Video Quality Management Tool.

At the end, we chose to use a factor of 10 000 for keypoints and a factor of 1 000 for the transformations, since it seemed to retain most of the quality, while reducing the file size drastically. Using 1 000 for keypoints was also possible, although there was very little difference in file size and quality, so we decided to choose a bit more quality over file size.

Figure 6 shows us how much of an effect quantization has on images. By comparing our normal 10 000 keypoint and 1 000 transformation approach to a generated video with one power less, we can make the inaccuracies visible. Even though the changes are not visible to the naked eye, the VQMT tool was able to show us the difference in quality.

6.4 Limitations

It is very important to understand what this project is capable of before using it. Besides using a square video format, optimally 256×256 and not expecting high-resolution outputs, we also have to consider perceptual information and filter out videos that do not fit these requirements.

First of all, this software is not very good at filling in data that is not visible in the source image. For example, if the person moves their head, areas that were not in the original source image might appear. This information needs to be filled in by coloring or warping the missing information in the area, which is done, but sometimes it does not work well. Figure 7 in the upper left corner, shows how such a result could look like. Even though this can produce some valid information, it does not do its job perfectly, thus the image quality suffers from this issue.

The fact that the generator can not generate new information is a big issue. Use cases like moving backgrounds, occluded areas, turning heads or, objects, and excessive movement in general often lead to unsatisfying output videos. This needs to be considered when picking an appropriate driving video.

One has to be careful that the background does not move. If it does, it warps not only the moving parts, but it also might warp the face, which will result in minor distortions. These distortions can make the face more shaky and distracting. Even if the background is still, the generator might not have a clear

Fig. 7. Image examples of badly generated frames.

line indicating where the head ends and the background begins. This results in the background sometimes being warped by head movements.

Another important point is that the person in the image needs to have clear facial expressions. Things like barely moving lips or subtle eyebrow movement will not be picked up by the keypoint detector and generator. This will result in almost no visible motion in the output frames. Sometimes, it also does not translate blinking correctly. Often you can see a shift in the pixels that should represent the closing eyelid, but by closer inspection one can see that it does not look like a full blink. This can be seen in the lower left comparison in Fig. 7.

Additionally, turning the head too far to the side, for example by almost 90 °C, will result in a very faulty face position. Here it is not able to generate more information about the side view of the face and since the network only understands the front facing views, it will display a distorted front faced frame. This can be seen in the upper right comparison in Fig. 7.

Furthermore, the whole head of the person must be covering most of the frame area, optimally being centered and looking into the camera. It appears that if the head is not in focus, face features, like eyes, cannot be correctly detected. Figure 7 in the lower right corner is a good example. Here it almost works, but still does not seem close enough to detect all face features correctly.

The camera itself also should not move too much. The generator does not handle things like zooming very well. Since the background moves unproportionally to the face, the face will get distorted.

Important to note is that, even though the video does look very similar to the original, the video could have minor facial differences depending on the source image. For example, in Fig. 7 in the upper left corner, we can see that even though the comparison looks similar, there are still facial differences because the expression changed noticeably since the first frame. Since the project is not

able to generate new information, it assumes that the face will stay the same throughout the video, thus creating frames that are not completely accurate.

A related issue is that if the source image is not in the same position as the first frame, we get very different facial expressions or movements that are faulty. This occurs when the video is too long and sthe project uses segmentation, where the head position in the first frame of the video differs immensely from the position in the source image. Because the starting position is wrong, the head will still use the same motion as in the driving video, which will cause inaccurate frame generation. This happened in our longNews video once and multiple times on the longObama video, because the news anchor did not move a lot, while Obama switched from side to side in his interview.

To sum it up, the best results can be produced when the video is short, contains a person, looking forward, with clear facial expressions, a still background, few head rotations and the person is covering most of the frame with a non-moving camera. These properties can be found, for example in news reporter videos.

6.5 Discussion and Future Work

Considering all problems and limitations of the project, the requirements are very restrictive. Many videos can not be used and even if the video fits, the user still needs to crop the video correctly and perform multiple operations by hand to achieve the goal, for example LZMA2 zipping. This could affect how quickly and well outputs can be generated.

In relation to this, there is a possibility to train a model or reuse the already trained face model to detect faces in videos and crop them accordingly with the help of video editing tools like FFmpeg.

Despite the shortcomings of this project, the results highly resembled the driving videos. Even with impressive breakthroughs in the area of artificial intelligence, it was surprising to see that the generated videos managed to replicate the original video so well.

What we did not expect was how little data was needed to create videos, and even more unexpected was the difference of data size compared to H.264 and AV1 codecs. It showed us that utilizing artificial intelligence as video codecs instead of years of development of a better algorithm is possible. This can lead to new codecs that utilize neural networks to encode videos and are better than normal codecs. There are still potential improvements on the shortcomings we did not succeed to fix in this project. Additionally, to the suggestion of reusing the face model for cropping, it is also possible to implement a neuronal network that improves the image quality of faces.

Another issue of the project is that it poorly handles longer videos with a lot of motion. Because the first frame of the segmentation likely starts from a different position than the source image, the motions will either move the face differently than in the original video or it can not handle the difference and generate frames that do not look pleasant.

One approach could be to send more source images to the receiver. For each segment, we would require to send a new source image, which would be the first frame of each segment, but at the cost of increasing the file transmission size. The source image also needs to be carefully selected, as seen in Fig. 7 what can go wrong. Even without careful selection of the source image, using multiple source images will probably increase the quality of longer generated videos, since major distortions are worse than blurry images.

At first, finding a neuronal network to outperform state-of-the-art codecs was hard, but even though many things are not fully satisfactory, the project gave us new views on what artificial intelligence is capable of and how much potential it still has. This project should encourage further research into that area and find even better ways to encode videos with neuronal networks.

7 Conclusion

We have posed the question if it is possible to generate videos with artificial intelligence that only requires little information extracted from a video and a source image, which can be sent over the network. By adapting the project from Siarohin et al. [17], we managed to extract the keypoint and local affine transformation from a neuronal network face model and compressed it to send the data to a receiver, which was able to reconstruct the original video approximately.

Although the results were not perfect, the partial success of this project showed us the potential of artificial intelligence for video codecs and should encourage further research into this area.

References

1. In-home media consumption due to the coronavirus outbreak among internet users worldwide as of March 2020, by country. www.statista.com/statistics/1106498/home-media-consumption-coronavirus-worldwide-by-country/. Accessed 14 Nov 2021
2. H.264 : Advanced video coding for generic audiovisual services. www.itu.int/rec/T-REC-H.264. Accessed 14 Nov 2021
3. AV1 Video Codec Homepage. aomedia.org/av1/. Accessed 14 Nov 2021
4. LZMA and LZMA2 7zip. www.7-zip.org/7z.html. Accessed 12 Nov 2021
5. LZMA2 7zip Documentation Page. sevenzip.osdn.jp/chm/cmdline/switches/method.htm#LZMA2. Accessed 12 Nov 2021
6. Bao, W., Lai, W., Ma, C., Zhang, X., Gao, Z., Yang, M.: Depth-aware video frame interpolation. CoRR abs/1904.00830 (2019)
7. Djelouah, A., Campos, J., Schaub-Meyer, S., Schroers, C.: Neural inter-frame compression for video coding. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 6420–6428 (2019)
8. Dmitriy, V., et al.: MSU video quality measurement tool (MSU VQMT). www.compression.ru/video/quality/measure/videomeasurement/tool.html (2009)
9. Grigorev, A., Sevastopolsky, A., Vakhitov, A., Lempitsky, V.: Coordinate-based texture inpainting for pose-guided image generation (2019)

10. Han, J., et al.: A technical overview of av1 (2021)
11. Johnson, J., Alahi, A., Li, F.: Perceptual losses for real-time style transfer and super-resolution. CoRR abs/1603.08155 (2016)
12. Kazantsev, R., Zvezdakov, S., Vatolin, D.: Machine-learning-based method for finding optimal video-codec configurations using physical input-video features, pp. 374–374
13. Let's Enhance, Inc., Let's enhance.io. letsenhance.io/
14. Li, Y., Roblek, D., Tagliasacchi, M.: From here to there: video inbetweening using direct 3d convolutions (2019)
15. Ronneberger, O., Fischer, P., Brox, T. U-net: Convolutional networks for biomedical image segmentation. CoRR abs/1505.04597 (2015)
16. Siarohin, A., Lathuilière, S., Tulyakov, S., Ricci, E., Sebe, N.: Animating arbitrary objects via deep motion transfer (2019)
17. Siarohin, A., Lathuilière, S., Tulyakov, S., Ricci, E., Sebe, N.: First order motion model for image animation. In: Conference on Neural Information Processing Systems (NeurIPS) (2019)
18. Siarohin, A., Lathuilière, S., Tulyakov, S., Ricci, E., Sebe, N.: First order motion model for image animation repository (2019)
19. Siarohin, A., Sangineto, E., Lathuiliere, S., Sebe, N.: Deformable gans for pose-based human image generation (2018)
20. Wang, T.-C., Mallya, A., Liu, M.-Y.: One-shot free-view neural talking-head synthesis for video conferencing. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2021)
21. Wu, C.-Y., Singhal, N., Krähenbühl, P.: Video compression through image interpolation (2018)