CREATING "ABSTRACT" REPRESENTATIONS OF VIDEO USING LATENT SPACE STEERING OF GENERATIVE ADVERSARIAL NEURAL NETWORKS

Kelden Abate, Clive Unger

The University of Texas at Austin

ABSTRACT

Our project uses generative adversarial neural networks to create abstract, interestingly modified versions of a source video. These videos could later be integrated to a social media platform for user-centric sharing. Our project has four main components, predicting an object in each frame, generating an input class vector, generating an input latent vector, feeding the preceding vectors into BigGAN, a state-of-the-art network for image synthesis [1]. Users interact with these components through a mobile application, where they can upload their own original content or view content that has been processed by our network.

1. INTRODUCTION

Videos have become a mainstay of modern internet communication, representing nearly 70% of current internet traffic. A large subset of this traffic is to social media applications where users upload, download, modify, and interact with videos. As the use of neural networks in video processing continues to grow with online video content, generative adversarial neural networks provide an interesting way to let users interact with the visual perception of their content.

Our project presents a method to generate abstract video from latent space steering of generative adversarial neural networks. For application we utilized the BigGAN [1] network presented by DeepMind, but the methods could be applied to any generator network.

Our primary goal is to create new abstract, artistic, or hallucinatory representations of a source video. We wanted to explore the power and limitations of GANs when applied to a sequence of frames. There are some examples of using neural networks to generate "art" using concepts such as style transfer or deep dream [2]. We wanted to contribute to this area with our project.

We believe that a possible useful implementation of these abstract videos is in social media applications. Currently, when users want to modify their own videos in social media applications such as Instagram or Snapchat, they are limited to modification that can be applied to static images, such as filters or stickers. Even the most complicated video modifications in Snapchat only rely on basic object detection algorithms. Abstract videos generated from GANs could be applied by social media users looking to personalize their social media posts. Due to the inherent mobile usage of social media, we found it

best to implement a iOS application that allows users to upload videos from their device and modify it with our network.

2. METHODS

We present a method to produce artificial or abstract video by manipulating the input space of a generative adversarial neural network or GAN. We then encapsulate this method in an end-to-end video processing mobile application that allows a user to upload a video and have a new video returned. Much of the work uses music videos as the source input, as that was our original inspiration for this project, however this method can be applied to any video.
The following sections will first show the video processing technique that was developed, followed by an explanation of the mobile app and API which provide an interface to the algorithm.

The video processing algorithm works in 4 steps:

1. Predict top ImageNet Class in each frame
2. Generate input class vector from predicted class
3. Generate input latent vector using 2 methods
    a. From random noise
    b. From the source frame
4. Feed the latent and class vectors into a Generator (BigGAN) to produce a new image for each frame

2.1. Background

In this project we utilize the state-of-the-art image synthesis network BigGAN developed by DeepMind. The paper sees that GANs benefit greatly from scaling therefore, BigGAN is an extremely large network with roughly 355 million parameters and batch size 8 times prior work. The network produces amazing looking images of relatively high resolution for a GAN. While our team may not have the resources to train or retrain such a large network, we can exploit its inputs to create interesting results. Therefore the majority of our methods revolve around BigGAN specific inputs. However, the methods presented could theoretically be applied to any generator network, but with varying results.

2.2. ImageNet Class Detection

First the source video is read as an array of frames, where each frame is downsized to 224x224. Then for each frame we use the MobileNet image classification algorithm to predict the top ImageNet class [3] in the frame. MobileNet is advantageous due to its lightweight processing, allowing for fast computation on mobile platforms (such as our iOS application). Since there are only 1000 ImageNet classes, not every frame for any given input video will have a corresponding image class, therefore we average the predictions across 10 frames to eliminate outliers [4].

2.3. Create Class Vector Input

Given a set of ImageNet labels, we create a sequence of class vectors that can be input into a generator, in our case BigGAN. Each label is one hot encoded, so if the label is 207 (golden retriever), we create a 1000 length vector full of zeros except at index 207. To increase the variability of the output, we further smooth the

class vectors by taking the average vector over 20 frames and interpolating it to the next 20 frame average. This effect creates vectors that have partial classes. Normally the class vector would just have a binary 1 or 0 in a single specific class. However, we can create class vectors with values between 0 and 1 for more than one class, thus a partial class. This created a blending of classes in the output image. For example, a value of 0.5 at index 207 (golden retriever) and 0.5 at index 270 (white wolf) should create a hybrid golden retriever/white wolf in the output image. By exploiting this property we can get smooth transitions between classes in the final product. Before applying this method, transitions were abrupt and not appropriate for the content we wanted to create.

Figure 1. Class vector generation example

2.4. Create Latent Vectors

A latent vector is produced for each frame in the original video, to produce a video of the same length. Normally a GAN generates a single image output from a latent vector of noise. However, to create a smooth video we need to generate images that appear to follow from the previous, otherwise the result will flash random static images with no context. We accomplish the task of manipulating the GAN output by utilizing the concept of latent space steering presented in [5]. Using simple transformations in the input space, we can achieve transformation in the output space. The mappings are not quite direct and require some experimentation to derive. Initial experiments showed some interesting results from applying the sine function to the latent space shown in Fig. 2. The objects or figures in the output frames would somewhat rotate around the center of the frame.

Figure 2. Sin(z) applied to latent space

We developed two different methods of creating a sequence of latent vectors. The first method starts by generating an initial noise vector sampled from a normal distribution. Most GANs use a Gaussian distribution for their latent space, however BigGAN uses a Gaussian for training but a truncated Gaussian for inference, the authors refer to this as the "truncation trick". Therefore we must resample values to be within a range of -2 and 2. When feeding the network values outside this range, the resulting image is incomprehensible.

For our application, the desire is for the output to have smooth transitions with no quick or jarring transitions from one frame to the next. This is accomplished by ensuring successive input vectors are relatively continuous. We implement this idea by multiplying the initial latent described by an "update" vector which multiples each value by a set value. Additionally, a "wobble" vector of random values from 0 to 0.05 is multiplied by the initial latent vector to add more variability. This process is applied sequentially until there are enough latent vectors to match every frame in the source video.

The result of this random latent method is a video that has objects or figures that morph from

one class to the next and move interestingly throughout the video. However, the movement and transitions in this video output will be completely independent from the source besides the ImageNet class being represented, since the only thing being derived from the source video is the object predictions.

In an attempt to capture the relative pacing of the source video, we instead generate the latent vectors from the frame itself. The reason we want to derive the latent vector from the source frames is because if a portion of the video is stationary or has little movement, the latent input for that portion should also not change much. However if the source video has a lot of action or a scene change, the input vectors will move quickly and reflect this. As a result, the output video should generally follow the pacing of the source. The frame is converted to grayscale and 140 points are subsampled to make up the new latent vector. The intensity values are scaled from (0,255) to (-2, 2) to match the input domain. The resulting latents are smoothed to remove noise. We found the best results with this method, likely due to the variability in the source video being somewhat represented in the output.

2.5. Video Generation

Once a latent and class vector has been created for each corresponding frame in the source video, we feed them into the BigGAN network to generate a new artificial frame. There are multiple variants of the BigGAN model for different resolutions, but we choose the model which creates images of 256x256.

Once all the frames are computed, we sequence them into a single video and match the source audio with the new video. We find the best results on music videos, as they have a variety of classes and movements that create interesting representations from BigGAN. With the matching music, the final product looks especially interesting, both resembling abstract, hallucinogenic art, and creating an obvious abstract version of the input video.

2.6. Mobile Application

We have written a mobile application for iOS devices that allow users to upload their own videos to be modified through our network. We chose iOS devices over Android devices for several reasons. Most importantly, recent Apple A-series processors have a dedicated 8-core processor dedicated to neural network computation. Though we currently process videos on the cloud, we believe that moving to mobile computation in the future could provide time and resource savings when making computations. Additionally, we decided not to make a cross-platform application through react-native (compatible with iOS, Android, and web applications) due to the overly restrictive nature of the platform, poor user experience, and inability to take advantage of the computing resources available in modern smartphones.

Our application is written natively in Swift. When a user opens the application, they are presented with an option to choose a video from their camera roll. The video is then compressed, making it easier to save and push to our API. This video is then saved in a sandboxed filesystem relative to the application, eliminating the possibility of corruption to the user's original video. Relevant file information, such as name,

category, and location, are all stored in a JSON file that allows the application to dynamically populate lists of all content that the user has either uploaded or received on their device. After selecting a video, an API call is made that begins running the video through our network.

2.7. Application and API Communication

We realize the mobile users will not want to wait significant amounts of time for their videos to process. This is why we employed an app that calls a REST API [6] for backend computation. The backend is a Flask app written in Python. Flask was chosen because of its integration with RESTful APIs and its built-in development server. We chose Python because we wanted a Flask app, but the language is also advantageous due to its ability to easily integrate with our neural models. Currently, cloud computing resources process videos faster than our mobile devices are able to. Additionally, we focused on lightweight API calls that improve processing speed. This is a reason why we chose MobileNet to compute our initial object detection on all video frames. Additionally, we tested methods of compressing our videos on-device, which theoretically allows for a quicker API call and less time waiting for videos to be processed before downloading. Finally, once a video has been compressed, a user streams that video on their device instead of downloading it before viewing. We found that this significantly reduces a time that a user has to wait between uploading their video and viewing modified content.

3. RESULTS

We achieved a final product that takes a source video and successfully outputs a generative representation of it. The quality of the output varies with the input. We find that more natural scenes with landscapes or mountains have quite pleasing generated responses, likely due to the presence of ImageNet classes. However, more complex videos, especially ones with many people, gave noisy, uninterpretable videos. Since BigGAN was trained on ImageNet, there is no explicit representation of humans beyond when they are in the background of the image. Therefore, our technique also struggles with humans. Overall, the best generated videos come from scenes with a strong presence of ImageNet classes. Figure X shows a few processed frames from the music video for "Holocene" by Bon Iver [7]. You can see the landscapes are accurately portrayed, but the boy's jacket is classified as a polar bear.

Figure 3. Example Frames from "Holocene" Music Video [7]

The results of this unorthodox video processing method are difficult to quantify. Arguably the initial goal insists that no numeric value could be measured since we are generating "abstract" representations of a video. The objective was really to explore the potential and limitations of state of the art generative networks,

by exploiting their input space to the maximum degree. By doing so we are able to achieve extremely interesting somewhat "psychedelic" outputs from the network, some good, some bad.

The final user-facing application allows users to upload a new, original video from their phone's library, receive the modified version, and view all videos and modifications that they have uploaded or received in the past. After a user has uploaded a video, our app stores the video locally in our application. When storing locally, Apple requires the video to be stored in a sandboxed data structure independent of other system data. To index these sandboxed locations, we are using a single JSON file that keeps track of the relative file paths of a user's videos. By keeping track of these paths, we are able to use Apple's AV Player library to playback both the original content and modified content to the user in our app [8].

In order to save both space and time, we stream the processed video from our web server. We know that users will be impatient when waiting for a video to be processed, so we attempt to minimize the time they spend waiting. Additionally, the aforementioned on-device compression was implemented with HEVC, which Apple requires when storing videos in their sandboxed data structure. This compression of user uploaded videos does indeed reduce the amount of time it takes to upload and receive videos. However, time optimization should certainly be a future focus of this project, as it currently takes about one minute to process a single ten second clip.

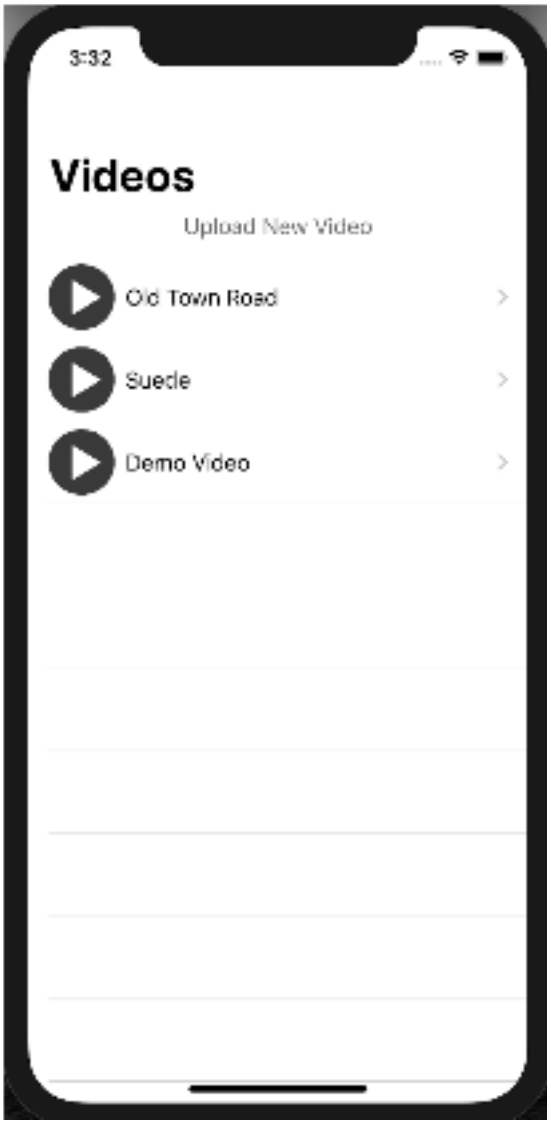

Figure 4. Mobile application homepage displaying ability to upload new video and playback existing videos

4. FUTURE APPLICATIONS

We feel we have just scratched the surface of what is possible with latent space steering. With further experimentation one could discover even more complex transformations of the BigGAN latent space that map to real transformations in the image space. Generating video with GANs is a

young area of research, as most work is focused on static images.

Orthogonally, the methods presented could potentially be used to learn more about interpretability of generator networks.

Further, our mobile application and API is not limited to this single model. With our modular design software architecture we could easily add new video processing models to do things such as style transfer or motion detection.

As mentioned previously, we make an API call in order to perform all video computation with cloud resources. However, in the future, we believe that performing at least partial computation on the mobile device may require less time from initially uploading a video to playing back a new generated video. Implementing this would begin with implementing our model with Apple's Core ML 3 framework, which would allow us to use the Neural Engine present in modern iPhones.

5. REFLECTIONS

This project offered an immense learning opportunity for both of us. The team had limited experience with GAN's beyond knowing how to train them, and that they are difficult to train. We found it interesting to explore the latent space of BigGAN and find all the interesting things it could create, many of which did not make it to the final product. Also, when exploring other concepts to manipulate the style of a video we learned about the basics of Style Transfer and Deep Dream.

We also gained experience using industry standard machine learning and video processing tools such as TensorFlow, Keras, and OpenCV. Google Colab greatly enhanced our productivity while working remotely.

Upon beginning this project, neither of us had ever written an iPhone application in Swift. In order to write the frontend of this project, we had to research basic concepts such as data types, file storage, and user interface components. Additionally, we had to find methods to improve the performance of our application, such as on-device compression and fast API calls. This led us to use several frameworks already implemented in Swift, such as Apple's provided HEVC compression for user selected videos or the AlamoFire library for fast and lightweight API calls [9].

While we worked hard to implement an effective API that allowed end-to-end video processing, we understand that it would be handled by several people in several teams at a large company. It was not difficult to get a basic API and application that communicated with each other. However, we recognize that our application and model have limitations in terms of speed, efficiency, and robustness that real-world users require. We faced several challenges to this extent, including formatting data, transmitting data, and compressing data while maintaining integrity. However, a large company would likely have developers already experienced in languages that were new to us, with the goal of scaling to millions of users.

The work for the project was evenly distributed between team members. Initially, Clive researched GAN modifications such as BigGAN, CycleGAN, and DeepDream while Kelden worked on using MobileNet to generate labels to feed to BigGAN. Due to the restrictions on teamwork originating from COVID-19, the

two of us had to find a way to work relatively independently. After narrowing the scope of the project, Clive worked on generating the correct input vectors and refining our modifications of BigGAN due to his experience in machine learning, while Kelden developed the mobile application, due to his previous experience in mobile development. Both of us worked on the final paper and presentation.

Overall, we had fun with this project because of the creativity we could apply. We frequently passed some of our favorite music videos to each other and discovered how they could be manipulated. Thank you for the wonderful class Professor Bovik!

6. REFERENCES

[1] A. Brock, J. Donahue, K. Simonyan, "Large Scale GAN Training for High Fidelity Natural Image Synthesis," *2019 International Conference on Learning Representations (ICLR),*Cambrdge, MA, USA, 2019.

[2] Deep Dream Generator, *Human AI Collaboration,* https://deepdreamgenerator.com/, 2020.

[3] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Google Inc, 2017.

[4] J. Deng, W. Dong, R. Socher, L. Li, K. Li, L. Fei, "ImageNet: A Large-Scale Hierarchical Image Database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, 2009, pp. 248-255.

[5] A. Jahanian, L. Chai, P. Isola, "On the Steerability of Generative Adversarial Networks," *2020 International Conference on Learning Representations (ICLR),* Cambridge, MA, USA, 2020.

[6] Arteko, "The Best Way to Use REST APIs in Swift," *Medium.com*, 2019.

[7] Iver, B., "Bon Iver - Holocene (Official Music Video)" *Aug 17, 2011 Youtube*, 2011 https://www.youtube.com/watch?v=8T0cHQb39GY

[8] C. Mash, "AVPlayer & SwiftUI," *Flawless iOS,* https://medium.com/flawless-app-stories/avplayer-swiftui-b87af6d0553, 2019.

[9] J. Shier et. al, "Alamofire," *GitHub Repository,* 2020, https://github.com/Alamofire/Alamofire

[10] J. Brownlee, "A Gentle Introduction to BigGAN the Big Generative Adversarial Network," Machine Learning Mastery, 2019, https://machinelearningmastery.com/a-gentle-introduction-to-the-biggan/

[11] Z. Alyafei, "BigGanEx: A Dive into the Latent Space of BigGan," The Gradient, 2018, https://thegradient.pub/bigganex-a-dive-into-the-latent-space-of-biggan/