Computer Systems Lab Final Report
Applying Cartoony Style Transfer to Realistic Videos

Chloe Toda
May 27, 2025
Yilmaz - Period 3

Table of Contents

Abstract

The purpose of this project was to create a tool to make the cartoon animation process quicker and less time consuming. Through the use of generative adversarial neural networks, style transfer is applied to realistic videos of people to convert them into cartoon people in the style of old Disney movies.

I. Introduction

Purpose

Cartoons are a popular form of media. However, in some cartoon shows, such as anime, animators must draw out each individual animation frame, and there are often over twenty frames per second. The animation process can be repetitive and physically taxing, even with the use of modern tools like the ability to copy each frame digitally. I wanted to address this problem by converting realistic videos into cartoony ones using neural networks to mimic the animation process.

Additionally, the concept of style transfer is controversial in itself, especially in the art community. People have raised concerns on copyright issues when it comes to training networks to apply style transfer to pictures/videos. Many are afraid that artificial intelligence will replace fields requiring artistry and creativity (Dericks, 2025). As an artist myself, I wanted to investigate how style transfer works to better understand this issue.

Style Transfer

Neural style transfer, first included in a paper in 2015 (Gatys et al., 2015), is a relatively new process that involves applying the "style" of one image onto the "content" of another to generate a completely new image (Ferlatti, 2021). For example, in the figure below, the content image would be the lion, the subject of the generated image. The style image would be Vincent van Gogh's Starry Night, which is how you want the content to be drawn in the generated image. The same thinking can be applied to this project, with the content being real images of people, and the style being images of cartoon people.

Figure 1. Neural style transfer example. (Style Transfer Example, n.d.)

II. Background

Cartoon style transfer has been implemented before. One of the most popular instances of this is in social media camera filters. A few years ago, the Snapchat anime filter was extremely popular, being able to convert a video of a person into one where they look like an anime character. Now, there are popular style transfer filters across all social media platforms.

There have also been numerous research papers about cartoon style transfer. One that I used/implemented in my research was CartoonGAN. Written in 2018 by Chen et al., CartoonGAN is a generative adversarial network that can convert real photos into ones of various famous anime art styles such as Hayao Miyazaki and Makoto Shinkai. The creators of CartoonGAN focused on content images with buildings and other landscapes, while I used mainly content images with people. Additionally, while CartoonGAN trains on the anime style, I used a non-anime cartoon style. Figure 2 shows example results from CartoonGAN alongside the results from other anime style transfer models. CartoonGAN returned an image with simplified shapes and colors, as well as more apparent edges compared to the input image. These are the results I plan to achieve with my project.

Another paper I took inspiration from was PortraitNET. Unlike CartoonGAN, PortraitNET's style datasets were of non-anime cartoons, such as Rick and Morty and Bitmoji. Similar to this project, PortraitNET uses people's faces as content. However, the creators of ProtraitNET used semantic neural networks to apply style transfer as opposed to generative adversarial networks that will be used in this project. Figure 3 displays PortraitNET outputs for various different cartoon styles. While the colors of the people do match the colors of the input cartoon styles, the style is not recognizable. This leads to the question: is it possible to preserve the original cartoon styles even if they are not anime?

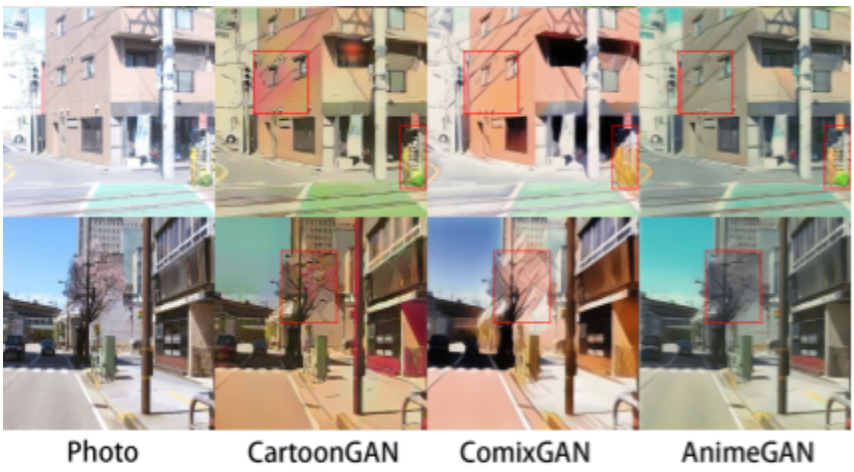

Figure 2. CartoonGAN and other anime-style transfer solutions.
(Tachibana, 2020)

Figure 3. PortraitNET examples. (Cui et al., n.d.)

Packages

The following packages/modules were used in this project:

glob [2]	glob is a Python module used to locate file paths of a specific type. In this project, glob was used to locate all the training image data so that it could be read in using OpenCV.
PyTorch [8]	PyTorch is a deep learning framework that's run on Python, and specializes in computer vision and natural language processing applications. PyTorch is used to create the Generator and Discriminator models in this project, as well as for many of the training processes and the use of GPU.
Tqdm [9]	tqdm is a Python module that creates a progress meter for loops. When training, tqdm is used to visualize how much of the data has been iterated over already.
OpenCV [7]	OpenCV is a library of many different programming functions relating to computer vision (CITE). In this project, OpenCV is used to open and save images, as well as for specific computer vision functions such as Canny edge detection and Gaussian blurring.

III. Applications

Possible applications of this project mostly relate to content creation. For example, videos with different styles applied to them are often popular online. Using the style transfer algorithm in this project, one can also create user-generated content. This project can also apply to educational videos. Many children enjoy watching cartoons, so they might be more inclined to watch an educational video if the people in the video were in a cartoony style rather than simply real.

As both someone interested in art and artificial intelligence, this project was ultimately for my own personal enjoyment and understanding. However, given how computers interact with images and visual content much differently than people do, this project will also contribute to a better understanding of the field of computer vision.

IV. Methods

Data

The training data consists of two different data sets, one for the content images and one for the style data set.

a) Content data

The content dataset is composed of images from the CelebA dataset [5], a dataset of over 200,000 photos of celebrities. While only using images of celebrities, the dataset is still very diverse, with people having different types of hair, accessories, and expressions as well as a variety of races and sexes. As training 200,000 images each epoch would take a large amount of time, I only used 13,514 images from the dataset.

I initially took a sample of 15,000 images from the CelebA dataset for the content data. However, after processing, this number was reduced to 13,514. The glob python package was used to iterate through all images in the sample. After each image was read in using OpenCV, and converted into an array using numpy, a haar cascade classifier was applied. Haar cascade classifiers are a type of machine learning program that specializes in recognizing objects in images and videos (Mittal, 2020). In this case, the haar cascade classifier, created by OpenCV, specialized in recognizing human faces. If a human face was recognized in the iterated image, the image was cropped into a square surrounding the face using OpenCV. Finally, the image was resized to 256x256 pixels for use in the neural networks, and then saved in a directory for processed content images.

b) Style data

For the style dataset, I wanted to use a non-anime style, as that has been implemented in many previous research papers before mine. However, I still wanted the chosen cartoon style characters to have similar proportions to people, to make training quicker. As such, I decided to go with the style of old, animated Disney movies for my style. The dataset was composed of images from an animation screencap website [1], which included screencaps across entire movies for most of the old animated Disney movies. To collect the data, I used an image downloader Google Chrome extension (Imageye) [4] to download all the images on one web page. This took a large amount of time since pages only have around one hundred screen caps per page, and not all pages

included images with cartoon people. Additionally, I took screencaps from multiple movies, which also contributed to the large amount of time it took to create the dataset. I took screencaps from the following movies: Anastasia, Beauty and the Beast, Cinderella, Lilo and Stitch, Mulan, Peter Pan, The Princess and the Frog, Sleeping Beauty, and Snow White.

The original style image sample was 10,970 images. After processing it was 3,458 images. The style images were processed similarly to the content images. Each image was iterated over using Glob. For each image, a haar cascade classifier [3] specializing in classifying anime eyes was used to figure out if the current image had a cartoon face in it. Using this haar cascade classifier was relatively successful, although classified a few images incorrectly. After determining if there was a face in the current image or not, the image was then cropped around the cartoon face, using the eyes as reference. Finally, each image was cropped down to 256x256 pixels, and added to a directory for processed style images.

When training on individual data, the CelebA dataset was always used as the content dataset. However, with video style transfer, the content data varied based on the chosen video input. The style dataset remained the same, regardless of whether videos were being trained or photos were being trained.

Systems Architecture

A generative adversarial network (GAN) is used to generate images in the desired style. A GAN is a type of neural network that specializes in image generation and recognition. It is composed of two convolutional neural networks (CNNs), which take in an image as input and figure out important features in that image that can be used to identify what that image is (Liuberskis, 2019). One of these CNNs is called the Generator, and its goal is to take in random input and convert it into the desired output image. The other CNN is called the Discriminator, and its goal is to take both a real input image and the fake image generated by the Generator and decide whether or not the fake image is “real.” If the Generator fools the Discriminator, it is rewarded. If it does not fool the Discriminator, it is punished. The same logic applies to the Discriminator. The Discriminator and Generator CNNs will “compete” against each other, improving as training continues. Through this process, the CNNs should become so good at generating or judging images that they can fool a human into thinking that the generated images are real (Google Developer, 2025).

While a GAN is used to generate the images in this project, we must go further in order to have the GAN be able to apply the desired cartoon style to the input image. The GAN architecture is modeled after a GAN model created by students at UC Berkeley, called CycleGAN (Efros et al., 2017). CycleGAN specializes in image-to-image translation, which is in essence the same thing as style transfer. CycleGAN focuses on transforming one type of “thing” into another type of similar “thing.” For example, zebras to horses or apples to oranges. What

makes CycleGAN so unique is that it has two Generator-Discriminator pairs instead of one. Using the zebras to horses case as an example, one of these G-D pairs specializes in converting a zebra into a horse, and the other pair specializes in converting a horse into a zebra (hence the "Cycle" in CycleGAN). The use of two G-D pairs allows both pairs of Generators and Discriminators to better understand the similarities between the two types of input. As opposed to the "original" type of style transfer in which one image is the content and the other is the style. In CycleGAN, both images are simultaneously the content and the style, depending on which way the cycle is traveling. I followed Aladdin Persson's CycleGAN tutorial and explanation on YouTube for my own implementation of CycleGAN.

a) Generator

The Generator's structure for both G-D pairs is the same. The current input image is first converted into a numpy array before being fed into the Generator. The Generator consists of two downsampling, with a stride of two. Following this are six residual blocks, whose goal is to prevent the network from never converging, and allows information from earlier layers to bypass one or more layers forward, preserving past information about the image (vanishing gradient problem). Following the residual blocks are two upsampling layers to return the image to its original size. The Generator ends with one convolutional layer to map the different RGB layers of the image.

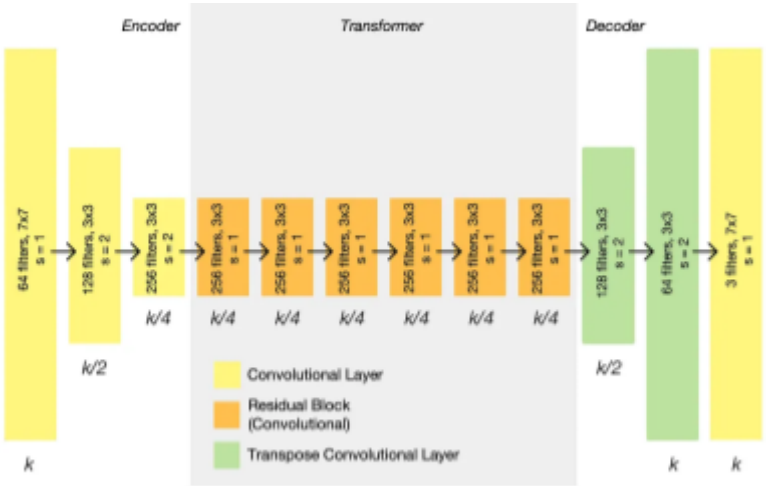

Figure 4. CycleGAN generator architecture. (Wolf, 2018)

b) Discriminator

The Discriminator CNN is modeled after PatchGAN, which compares smaller parts of an image (i.e. patches) to determine if an image is real or not. The Discriminator is simply composed of four convolutional blocks, with the first three having a stride of

two and the fourth one having a stride of four, in order to return a single value at the end of the discriminator, indicating whether or not it believes the input to be real. As with the Generator, the Discriminator's architecture is consistent for both G-D pairs.

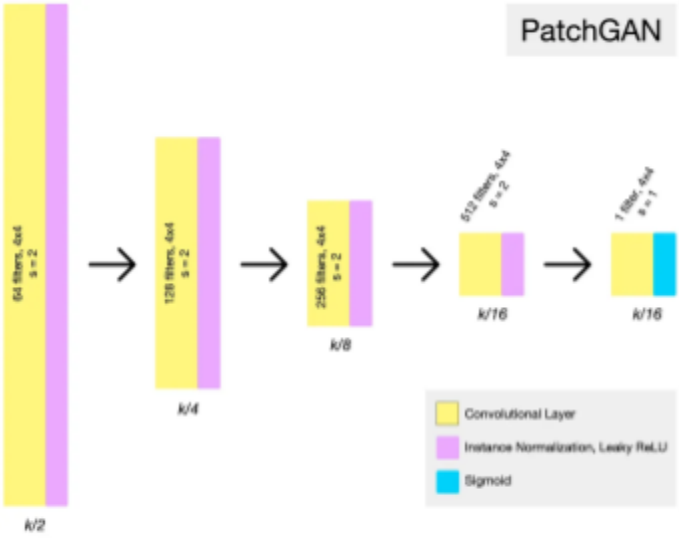

Figure 5. CycleGAN discriminator architecture. (Wolf, 2018)

Cartoon Feature Preservation

CartoonGAN, mentioned in the Background section of this report, used unique processes and loss functions to push its generator towards creating more cartoony images. These processes either focused on cartoon feature or content feature preservation.

a) Initialization

Before beginning training on the network, CartoonGAN must first become familiar with the content dataset's features through a process called initialization. For the first couple epochs, both G-D pairs will become familiar with the content images, but not the style. In the case of the G-D pair specializing in creating cartoon images, the pair will first "learn" what real people look like. The process is displayed in Figure 6. First, the real image is passed into the cartoon generator. The cartoon generator takes this image and generates a new image, which is then compared to the original real image using the cartoon discriminator. In the actual training phase, the cartoon image would be compared to the generated image. However, since this is the initialization phase, the goal of both G-D pairs is to first become familiar with the content before applying the style. The idea behind this is for the G-D pairs to learn what important features must be present in the images with style transfer applied. In this project, important features would be facial attributes like the eyes, nose, and mouth.

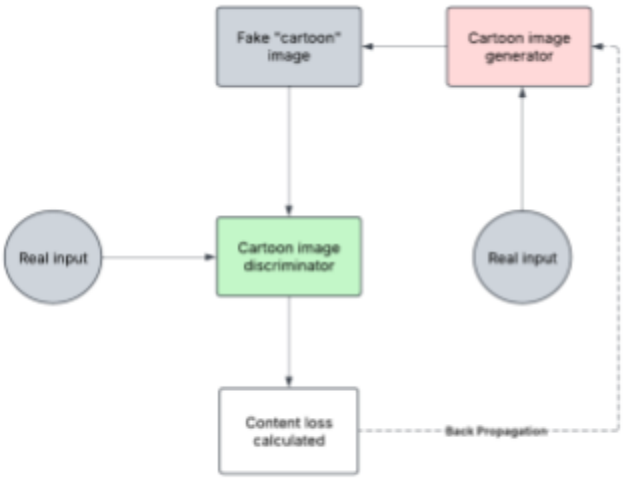

Figure 6. Initialization phase flowchart.

b) Edge Loss

To preserve cartoon features in the generated images, CartoonGAN incorporates an additional loss function to increase the likelihood of clear, defined edges being present in the generated images. When the generated cartoon image is passed into the cartoon discriminator, it is compared with the original cartoon image. However, with edge loss involved, the generated cartoon image is also compared to a cartoon image with the edges blurred. This is done by first locating all the edges of the cartoon image using OpenCV's Canny edges detection algorithm, then dilating the regions where edges were detected to include some of the space outside of the edges. A gaussian blur is then applied to these regions, also using OpenCV. This creates a similar cartoon image, just with blurry edges. The goal of this is to teach the cartoon discriminator what the generated cartoon images should *not* look like, further pushing the cartoon generator to create images with clear edges if it does not want to be punished.

Training

To reiterate, the architecture of this project is modeled after CycleGAN, with the added loss and initialization phases of CartoonGAN. The discriminators and generators are trained in two separate groups. To train the real discriminator (DR), the fake real image must first be generated by the real generator (GR), which takes a cartoon image as input. The fake real image is then inputted into DR, which generates a "patch" of numbers. The original real image is also inputted into DR. DR's total loss is the sum of the mean squared error of the original real image and the real image DR output and the MSE of the fake real image and the fake real image DR output.

The training process for the cartoon discriminator (DC) is almost identical to the DR training process, although it takes the opposite images as input: a fake cartoon image is first

generated by the cartoon generator (GC), and then inputted into the DC, and so on. The DC takes into account the edge loss value introduced by CartoonGAN. The blurry cartoon image is inputted into the DC, and a loss is calculated by taking the MSE of the DC output and a matrix of zeros of the same size.

The losses calculated from the DC and DR are averaged and then back propagated through both discriminator networks.

The training process for the GC and GR involve a few more losses than the training process for the discriminators. The first loss is the adversarial loss. For the GR's adversarial loss, the MSE is calculated between the DR's output from the fake real image, and a matrix of ones of the same size. As opposed to the discriminators' training process, in which its output is compared to a matrix of zeros, thus punishing the discriminators, the generators are rewarded the more the discriminators return the wrong decision. The GC adversarial loss is calculated the same way, except with the fake cartoon image.

The second loss is the content loss, which compares how much the content input and generated image have changed from each other. The GR's content loss is found by calculating the L1 loss between the original cartoon image and the generated real image (which used the cartoon image as "content"). The GC's content loss is found by calculating the L1 loss between the original real image and the generated cartoon image.

The third loss is the cycle loss, a key feature of CycleGAN. Two additional images are first generated: a cartoon image generated by the GC with the fake real image as input, and a real image generated by the GR with the fake cartoon image as input. In other words, the real image is first transformed into a cartoon image, and then back into a real image; a cycle. This cycle is to ensure that the generators are learning important features in both datasets. The GR's cycle loss is found by calculating the L1 loss between the original real image and the second generated real image, and vice versa for the GC.

The final loss is the identity loss, which helps the generators avoid creating images with inverted colors. An additional two images are generated: a fake cartoon image created by the GC with the original cartoon image as input, and a fake real image created by the GR with the original real image as input. Since the "style" images are being inputted into the generators that apply their style, the output image should essentially be the same. The GR's identity loss is the L1 loss between the original real image and the fake real image, and vice versa for the GC.

The sum of all the GC and GR losses is taken and back propagated through both generators as the total loss. The training process for one direction of the cycle is outlined in Figure 7 below.

Figure 7. System flowchart for one side of the cycle, converting a real image into a cartoon image.

Video Training

Style transfer with videos is relatively similar to style transfer with individual images. The videos used in this project were either found and screen recorded on YouTube or recorded by myself. As opposed to plugging in random individual content images, the video is split up into different frames and each frame is plugged into the Generator consecutively. If each real frame was run through the Generator individual, it would result in large amounts of flickering and differences between frames. Additionally, the G-D pairs would not be picking up any new information about the content without the training aspect. As such, more features must be put in place to minimize the flickering between video frames (Huang et al., 2017).

Instead of taking each individual frame one at a time, the current frame and the previous frame were inputted into the network at the same time. After being taken in by the cartoon generator, thus generating the fake cartoon images, the two are compared to each other and their differences are calculated using an L1 loss function. This loss, temporal loss, is then back propagated to the cartoon generator to push it towards creating similar consecutive frames. In addition to temporal loss, some papers I looked at also implemented optical flow, which is an algorithm run before training that allows the computer to “predict” what the next frame should look like even before it is inputted into the network. I did not have enough time to implement optical flow in this project. However, there are many already-trained optical flow algorithms online that I could have incorporated into this project should I have had the opportunity to. The video style transfer system is shown in Figure 8.

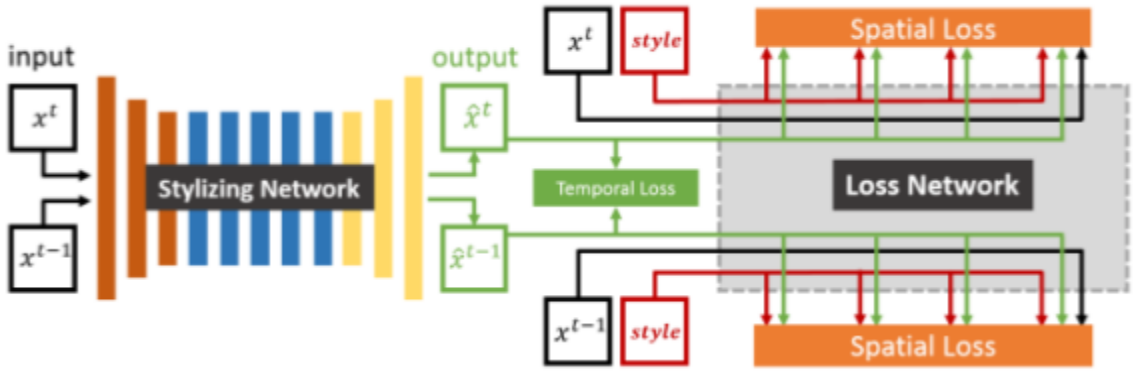

Figure 8. Video style transfer flowchart. (Huang et al., 2017)

Trials and Errors

When first researching neural style transfer, I came across tutorials involving just a single CNN instead of using GANs (Renotte, 2021). This method of style transfer requires pairs of content and style images to be inputted into the network in order to generate an image. However, the goal of this project was to generate images only given a content image (i.e. a real person), without the need for any paired cartoon image. If I were to have taken the paired image style transfer approach, the cartoon style image also inputted to the network would have to be similar to the real content image of the person. This leads to additional problems of figuring out a way to automatically pair content and style images together using a computer. Additionally, after following the Renotte style transfer tutorial on YouTube and testing it with paired images, it was clear that this approach did not include the lengthy training required to "teach" the neural network important features congruent between both image datasets. CycleGAN offered a new approach to style transfer, using unpaired images instead.

V. Results

Individual

Below are example results for training with individual images.While the style is not completely apparent, there are noticeable cartoony features like larger eyes and less detail in the hair and face. As training progresses, the generated images appear to have more of their content preserved, with cartoon people having more important features like the eyes, nose, and mouth. In many generated images, the background ended up being more simplified or one solid color, most likely due to the lack of consistent background content images being fed into the neural networks.

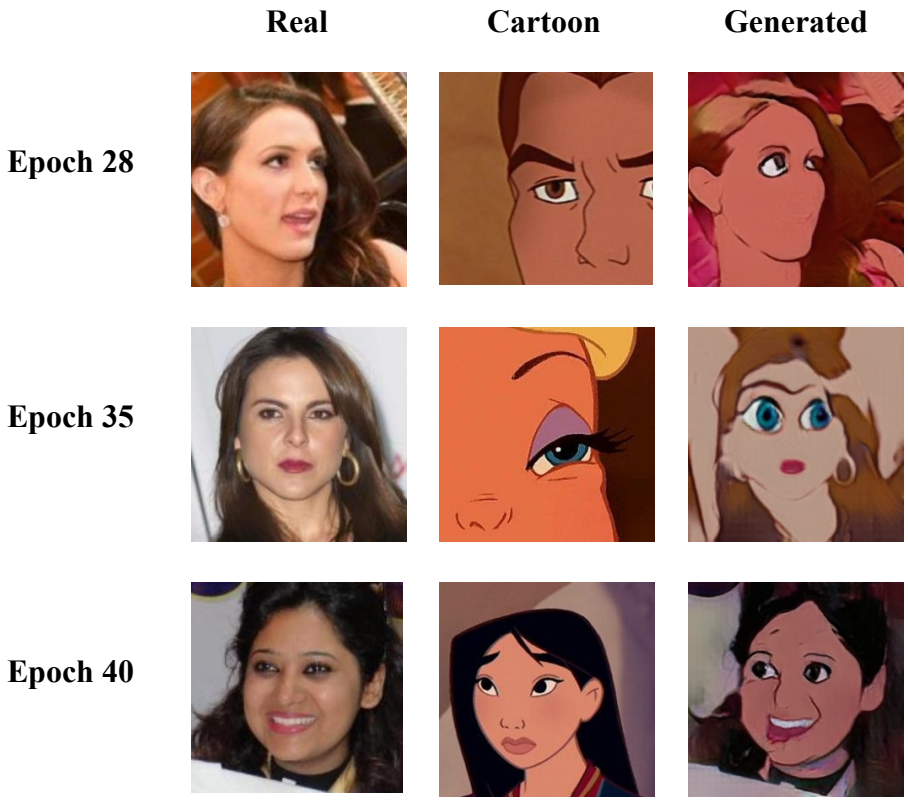

Figure 9. Individual image training results.

Frame by Frame

Below is a diagram containing example results for training with individual images. The video used for this example was found on YouTube (CITE). Similar to the individual image results, the generated video frames had noticeable cartoony features, despite the style not being completely apparent. The top row in the figure are seven of the many input frames taken from the video. The middle row shows the results of simply plugging each individual frame into the GC one by one, without any training. The cartoony style is less pronounced in this row, and there is more obvious discoloration in the face. The bottom row shows the results of training consecutive frames through the neural network, as well as taking into account temporal loss. There is more even coloration in these frames compared to the ones in the middle row, and the resulting video from the temporal loss frames is also much smoother.

The network was also trained on other videos found on YouTube. If the person in the video was further from the camera, the cartoony style was less apparent since the face was taking up less pixels in the frame. A compilation of style transfer being applied to different video clips can be found here.

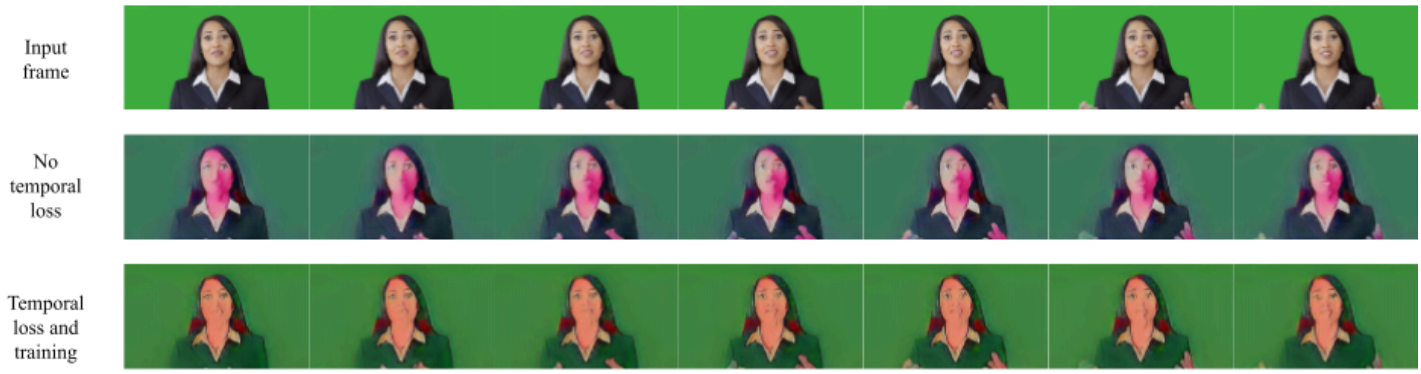

Figure 9. Video frame training results.

VI. Limitations

This project faced many limitations related to resources. When working with images and neural networks, a Graphics Processing Unit (GPU) is needed in order to reduce the amount of time required to train on the images. My computer did not have a GPU that was compatible with PyTorch, and so I trained my networks on the JupyterHub servers run by the school, which gave me access to supercomputers like Unicron and Snowy. A more in depth tutorial can be found in the Appendix. However, with the JupyterHub servers overheating multiple times over the third quarter, thus preventing me from using them, I was not able to train my GANs as much as I wanted. Additionally, while coding on VSCode I am able to leave my computer on to train my neural networks overnight, JupyterHub servers shut down after extended use. With one training epoch taking around an hour to complete, I was only able to get through two epochs of training before my server shut down. Because of this resource limitation, I was also limited with the length of the videos I could train, as longer videos would take more generation time.

VII. Conclusion

The goal of this project was to use a neural network to apply a cartoony style to realistic videos, while still preserving the original content of the video. Training a GAN takes a large amount of time and GPU. As such, with more time to train and stronger GPUs, the Disney style might be more pronounced in the videos. Given the large amount of resources required to produce such outputs, perhaps it is better to leave animation to the actual animators, despite how interesting and novel this approach is.

VIII. Future Work

One extension I would like to pursue following TJ Star is working with longer video clips. The video clips used in this project were 5-15 seconds long and took around 30 minutes to train. I am interested in finding a way to minimize the amount of time required to train on videos, even if they are longer than a few seconds. Additionally, I want to implement optical flow, as I ran out of time to incorporate it into my final project. Cartoon people in the Disney animation style have features that are relatively proportional to a real human face. However, I am interested in seeing how computers interpret other styles of cartoon animation that are less anatomically correct. For example, characters in The Powerpuff Girls have extremely exaggerated facial features, with large eyes and sometimes no noses.

IX. References

Github Link:
https://github.com/kloweee/Applying-Cartoony-Style-Transfer-to-Realistic-Videos/blob/main/README.md

Tools and Packages

[1] Animation Screencaps [website used to collect Disney images for style dataset]:

https://animationscreencaps.com/

[2] glob: https://docs.python.org/3/library/glob.html

[3] Haar cascade for recognizing cartoon/anime eyes:

https://github.com/recette-lemon/Haar-Cascade-Anime-Eye-Detector

[4] Imageye Chrome Extension:

https://chromewebstore.google.com/detail/image-downloader-imageye/agionbommeaifngbhincahgmoflcikhm?hl=en-US

[5] Large-scale CelebFaces Attributes (CelebA) Dataset [collection of images used for content dataset]: https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[6] OpenCV Haar cascade for recognizing real human faces:

https://github.com/opencv/opencv/blob/4.x/data/haarcascades/haarcascade_frontalface_alt2.xml

[7] OpenCV: https://opencv.org/

[8] PyTorch: https://pytorch.org/

[9] tqdm: https://tqdm.github.io/

References

Chen, Y., Lai, Y., & Liu, Y. (2018). CartoonGAN: Generative adversarial networks for photo

cartoonization. https://openaccess.thecvf.com/content_cvpr_2018/papers/Chen_CartoonGAN_Generative_Adversarial_CVPR_2018_paper.pdf

Dericks, C. (2025, April 7). *ChatGPT, Princess Mononoke, And The Exploitation Of Studio Ghibli*. Next Best Picture. https://nextbestpicture.com/chatgpt-princess-mononoke-and-the-exploitation-of-studio-ghibli/#google_vignette

Efros, A., Isola, P., Park, T., & Zhu, J. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. https://arxiv.org/pdf/1703.10593

Ferlatti, A. (2021, November 25). Neural style transfer (NST) - theory and implementation. Medium. https://medium.com/@ferlatti.aldo/neural-style-transfer-nst-theory-and-implementation-c26728cf969d

Gatys, L., Ecker, A., & Bethge, M. (2015). A neural algorithm of artistic style. https://doi.org/10.48550/arXiv.1508.06576

Google Developer. (2025, February 26). Overview of gan structure | machine learning | google for developers. Google. https://developers.google.com/machine-learning/gan/gan_structure

Huang, H., Jiang, W., Li, Z., Liu, W., Luo, W., Ma, L., Wang, H., & Zhu, X. (2017). Real-time neural style transfer for videos. https://openaccess.thecvf.com/content_cvpr_2017/papers/Huang_Real-Time_Neural_Style_CVPR_2017_paper.pdf

J. Cui, Y.Q. Liu, H.J. Lu, Q.Q. Cai, M.X. Tang, M. Qi, Z.Y. Gu, PortraitNET: Photo-realistic portrait cartoon style transfer with self-supervised semantic supervision, Neurocomputing, Volume 465, 2021, Pages 114-127, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2021.08.088.

Koushik. (2023). Optimization algorithms in machine learning: A comprehensive guide to understand the concept and implementation. Medium. https://medium.com/@koushikkushal95/optimization-algorithms-in-machine-learning-a-comprehensive-guide-to-understand-the-concept-and-3db1df7a2f59

Liuberskis, R. [Python Lessons]. (2019). Convolutional Neural Networks (CNN) explained step by step [Video]. YouTube. https://www.youtube.com/watch?v=sgL7RrqhGKI

Mittal, A. (2020, December 20). Haar Cascades, Explained. Medium. https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d

[Style Transfer Example]. (n.d.). https://github.com/cysmith/neural-style-tf?tab=readme-ov-file

Persson, A. [Aladdin Persson]. (2021). CycleGAN implementation from scratch [Video]. https://www.youtube.com/watch?v=4LktBHGCNfw&t=2405s

Renotte, N. [Nicholas Renotte]. (2021). Neural style transfer tutorial with tensorflow and python in 10 minutes [Video]. YouTube. https://www.youtube.com/watch?v=bFeltWvzZpQ&t=662s

[Style Transfer Example]. (n.d.). https://github.com/cysmith/neural-style-tf?tab=readme-ov-file

Tachibana, Y. (2020, May 26). AnimeGAN: A Novel Lightweight GAN for Photo Animation. GitHub. https://github.com/TachibanaYoshino/AnimeGAN

Wolf, S. (2018, November 19). *CycleGAN: Learning to Translate Images (Without Paired Training Data)*. Medium; TDS Archive. https://medium.com/data-science/cyclegan-learning-to-translate-images-without-paired-training-data-5b4e93862c8d

Yosinski, J. [Jason Yosinski]. (2016). Deep visualization toolbox [Video]. YouTube. https://www.youtube.com/watch?v=AgkfIQ4IGaM&t=222s

APPENDIX

I. **CODE:**
https://drive.google.com/drive/u/1/folders/1ErelOplYIpKaHadoH8X0IC-jjiKP7ZsF

II. **Using JupyterHub:**

In the case that your computer can't use GPU to run code, the TJ supercomputers can be used instead through JupyterHub. To do this, follow the instructions below:

1. Log into Ion and access JupyterHub through the Apps drop down menu on Ion's home page.
2. A list of server options should show up on the Jupyterlab home page. To work with GPU, select one of the servers with GPU. I usually went with the server that had the most CPU, RAM, and GPU.
3. Different TJ supercomputers have different types of GPU, and I found that some of these GPUs can be utilized by PyTorch while others can't. PyTorch relies on CUDA, which is used on NVIDIA devices. The supercomputers that support PyTorch are Unicron, Snowy, and Borg 28. The supercomputer you get for your server seems to be assigned randomly, or based on how many other users are also on that supercomputer. As such, if I got assigned a supercomputer that wasn't compatible with PyTorch, I had to reset my server multiple times until I was assigned a supercomputer that did work.

III. **Running Project Code:**

Below is an instructions list on how to run my project code.

1. The training datasets were too large to upload to GitHub, but they can be found in the Google Drive folder instead. Download all the datasets to be used in training later. If you would like to use and process your own training data, follow the process below:
    a. Open folder 00_DATA_PROCESSING. Open file make_inds.py and change variable globname to the file path to your unprocessed content data. Run the file to get a list of random training and testing indices.
    b. Open file content_process.py and change variable globname to the file path to your unprocessed content data. Create folders for your train and test data, and change the variable newfname to the file path of these folders. Run the file to get your processed content data.
    c. Open file style_process.py and change variable globname to the file path to your unprocessed style data. Change variable newfname to the file path and desired file name of your processed style data. Run the file to get your processed style data.
    d. Open file blurry_style_process.py and change variable globname to the file path of your processed style data. Change variable fname to the file

path of the desired folder and file name of your processed blurry data. Run the file to get your processed blurry style data.

2. Training with individual images (content dataset):
    a. Open folder 01_TRAIN_INDIVIDUAL. Change the variables in file config.py to match your intended file and directory locations. Weights can be altered in config.py too, as well as whether or not you are in the initialization phase of training.
    b. Open and run file train.py to begin training. The current epoch's progress will be displayed in the terminal, and the network weights will be saved after each epoch if you have indicated saving in config.py.
3. Training with videos:
    a. Download the video you want to apply style transfer to and save it somewhere you can access.
    b. Open folder 02_TRAIN_VIDEO. Open and run file combined_vid_transfer.py and follow the instructions displayed in the terminal.
    c. Run file join_frames.py to create the video from the frames in the directory specified by join_frames.py's variable path.