V-CAST : Video Extension of CAST

Alessandro Castillo Ray Forman Stephen Pasch
Columbia University

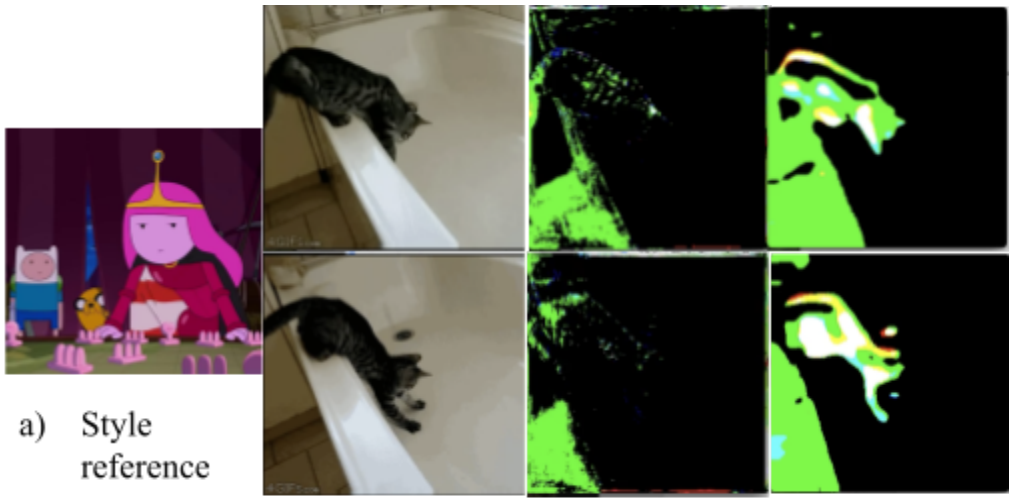


Figure 1: The output of V-CAST with input stylized reference image a), and two raw frames b) of the content video each 1 second apart. In c) we see the same two frames that have been process by V-CAST before we trained with the optical flow loss, and d) shows the same two results after training.

ABSTRACT

Image style transfer has become very popular for fun creative photo editing, branding, and as a tool for the generative artist. We seek with this project to create a framework to accomplish the task of arbitrary artistic style transfer to video. Our result is a video style transfer model that takes a short video and a stylistic reference image as inputs, and outputs a new stylized video according to the art style of the reference image. This manifests in an application interface to support this service in a user friendly way. We extend a previous work of image style transfer, CAST, and bring it into the temporal domain.

1. Introduction

Image style transfer has become very popular for both business and pleasure. Companies like Midjourney[1] have included style transfer networks into their generative AI platform as a way to give users control over their photos like never before. Branding companies like [2] have started leveraging style transfer and other image generation tools to create targeted marketing campaigns for their clients with unprecedented agility. This allows them to be personal, consistent, while reducing production costs. Artists have begun using style transfer as a way to quickly take sketches and augment color, style, or even material properties[3]. This allows artists to iterate ideas and bring their creative vision to life faster than ever before. These tools have also led to the creation of a new generation of "AI artists". These new school methods have caused much controversy in the world of game development especially; some even scorning practitioners of generative artist methods as fake artists whose work will always lack the understanding of composition that "real" artists possess. But just as with the adoption of digital art into the animation

industry in the 1970-80's, these new methods will never fully replace the need for a human touch even if they are here to stay. It will just take a different form.

These use cases encapsulate the importance for further exploration into the world of style transfer. We seek with this project to create a framework to accomplish the task of arbitrary artistic style transfer to video. Our result is a video style transfer model that takes a short video and a stylistic reference image as inputs, and outputs a new stylized video according to the art style of the reference image. This manifests in an application interface to support this service in a user friendly way. We extend a previous work of image style transfer; CAST*(Contrastive Arbitrary Style Transfer*)[4] and bring it into the temporal domain.

Video presents an interesting challenge because if we were to apply the CAST to each frame of the video that would be a) slow due to the most video being shot at 24 fps or more. This quickly becomes a lot of inferencing and does fix the second issue of b) consistency between frames. The contributions that we have made to this work are:

1. A reworking of the network to receive video input in a way that takes advantage of redundancies in inferences between frames to decrease transfer time for the temporal sequence.
2. A new spatiotemporal flow-based loss component that will increase consistency between frames.

These modifications allow us to finetune the model on short videos and show that it is both faster and more visually appealing than to apply the network to individual frames and stitch them into a video.

2. Previous Work

The previous work we have built upon is CAST, contrastive arbitrary style transfer by Zhang et. al[4]. This work learns style representation directly from image features instead of their second-order statistics like previous methods showed effective. This provides a more intuitive understanding of the style distribution that the model learns. It also employs the notion of contrastive learning by analyzing the similarities and differences between multiple styles. The framework "consists of three key components, i.e., a **multi-layer style projector** for style code encoding, a **domain enhancement module** for effective learning of style distribution, and a **generative network** for image style transfer."[4]

2.1 Multi-layer Style Projector

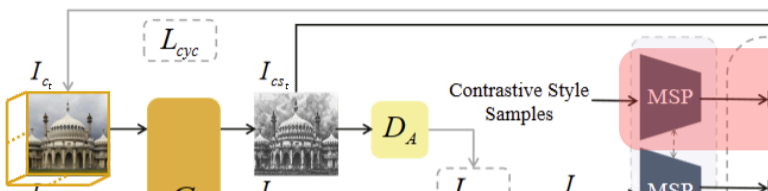
To find a suitable style representation which can discriminate between different styles guide the transfer to a new image the authors of CAST design a multi-layer style projector module(MSP). It consists of two parts, the style feature extractor(VGG-19 based) and multi-layer projector that projects the style features into a set of latent style codes $\{z\}$. These can be plugged into an existing style transfer network as the guidance for stylization.

2.2 Contrastive Style Learning

Typical deep learning uses labeled data to minimize a loss of a ground truth and the network output. The authors of CAST use contrastive learning to maximize the similarity between positive examples and minimize the dot product similarity of style representations that are "negative examples." A negative example is anything of a different style label.

2.3 Domain Enhancement

The authors also include a domain enhancement step. This is an adversarial loss during training that uses an artistic discriminator and a realistic discriminator. These are used in training to ensure that the network output of the real image with artistic style will be sufficiently



artistic looking. In the same way, if the artistic image is tasked to transfer the style of the real image onto it, the output should be discriminated as real. These showed to improve the results of the generator.

3. Architecture

To generalize the CAST network to video we change the original architecture. We show below that there are redundancies in the network

Figure 2: The CAST framework with our augmen
to a stack of images from a video sequence and no
the red regions, that denote calculations that we wi
other frames of the video. This greatly improves o

that are constant for applying the same style to many frames of a video. Taking advantage of these redundancies for subsequent frames affords us a speed up at inference and training time. Additionally, we introduce a new loss term for spatiotemporal consistency to account for strange results with using a frame to frame technique.

3.1 Abstraction to Video

The naive approach to video style transfer is to apply an image style transfer network on each frame of a video with the same style reference. There are two main concerns with this approach. First, that is wasteful computationally since many calculations are needlessly repeated. Secondly, that due to the generative nature of the style transfer, each frame can be styled very differently than the last frame, leading to a disruptive and inconsistent result when viewed as a video. To correct for the first concern we cache any intermediate results that pertain to the style input because that will remain constant over the course of the video transfer. In Fig.2 we show in the red highlighted regions of the model training process where this caching can take place.

3.2 Flow Based Loss

To account for the previously mentioned frame-to-frame style transfer discrepancy we propose a new loss term to the previous work. It is based on the optical flow between the frames. This concept is borrowed from computer vision and was first popularized by Lucas & Kanade [5] in 1981. The theory goes that the function of motion is described by a taylor series(an infinite sum of its derivatives) but we

a) b) c)

Figure 3: Visual results of optical flow calculation c) using Lucas-Kanade method between two successive frames, a) & b), of a hand holding a mouse moving from left to right. Direction of motion is shown with red vectors.

can approximate that to a first order taylor series assuming the motion of a pixel is small from one frame to the next(higher order terms approach zero). This holds true for most videos since we are sampling at 24 fps. The second assumption is that the brightness of each pixel varies smoothly over the image. This means that to properly track motion the lighting must stay relatively consistent over the course of the video.

Thus we will compute the flow between two of the raw frames of the video. This will return a H*W*2 array holding the flow information, where H & W are height and width, and each position holds a two dimensional vector(x,y) of the direction that pixel moved. We define the flow between two frames as $of(I_i, I_{i+1})$, where I_i and I_{i+1} are two adjacent frames.

$$L_{flow} = \| of(I_i, I_{i+1}) - of(G(I_i), G(I_{i+1})) \|_2 \quad (1)$$

We construct our loss term, L_{flow}, as the mean squared error of the flow between two raw

frames and the corresponding two generated frames.

3.2.1 Training Objective

We utilize the l2 loss of the proposed difference between flow fields to preserve the original intended motion and ensure that the generated frames have a similar deformations. Since we are only comparing motion and not appearance we do not suppress the style transfer, but only ensure that the style transfer is consistent to the next frame. This is because it penalizes regions that show differences where there was no motion in the original video.

4. Implementation

The CAST network was implemented in pytorch and made publicly available. We took their implementation. Changed how the input was processed (from image to video) by creating a video dataset class. For the first frame we process it the same as originally. Then for every subsequent frame following we hold onto all the results in Fig.2 marked in red to be reused. We additionally hold onto the previous generated frame to be used in flow calculations.

4.1 Training & Data

We train our network starting with the pretrained weights from CAST. We update these weights with our new flow based loss. But to employ this we require videos, and could not use the same image data as in CAST. Instead we use TGIF[6], a dataset of ~2000 GIFs extracted from the social network Tumblr. Using these GIFs we were able to build a dataset of short (2–8 second) videos by converting each to an MP4. The result was a dataset of short-form videos suitable for training our model.

For our style image references, we chose to diverge from the CAST paper's use of WikiArt as our goal was to emulate digital art styles used in animation motion pictures. However, we encountered considerable difficulty trying to find a dataset of frames from animated movies due to copyright issues. After an extensive search, we opted to pivot in the direction of TV cartoons, which were easier to find. Ultimately we chose to use a cartoon dataset[7] containing ~2000 images from the animated series *Adventure Time (2010)*.

In hindsight, we have identified a few key issues pertaining to the datasets we used that led to suboptimal results. First, while the GIFs we used were of the desired length, some of the GIFs by nature were choppy, with significant jumps between frames, often compromising our optical-flow assumption of marginal changes between frames. Additionally, these GIFs sometimes varied in frame rate, data that was lost in their conversion to MP4 and was not accounted for in the training process. Secondly, our choice to train on cartoon-style digital art proved challenging as compared to artwork in WikiArt. This is due to its discrete and precise nature, void of the gradients and brush strokes that are present in analog artwork and are best learned using the CAST architecture. Thirdly, we were largely limited by our lack of other digital art styles, which we were unable to obtain or create legally and within the time constraints of the semester. Ideally, we would have had several similar datasets from other animated movies or shows to improve the contrastive learning of the model and therefore improve results. Finally, with limited GPU space we were forced to only process one video at a time. A "batch" size of one significantly increased the training time.

Figure 4: A collection of samples from the cartoon classification dataset. Of which we chose samples from the show Adventure Time.

5. Results

The results in Fig.1 shows the improvement in overall smoothness of frame-to-frame motion. This supports that the optical flow loss has a positive impact on style transfer. Qualitatively these results show that there is less high frequency noise in our outputs. In the untrained video output there is a lot of high frequency flickering noise from the generation. In the results folder attached to the code you can view the video and confirm that the post-training result does not suffer from this flickering artifact.

	No Training	2 GPU hours	10 GPU hours
MSE	8.19E12	9.94E12	4.94E12
% change in MSE err. from notraining		+21.94%	-39.79%
Abs Err.	2.31E6	2.60E6	1.84E6
% change in abs err. from no training.		+14.19%	-10.74%

Table 1: We define motion as the total intensity change over the image from one frame to the next. The "motion" of the raw video serves as our ground truth. We then compare both MSE and mean absolute difference across three stages: no training, little training, and more extensive training.

Quantitatively, we see the model initially(no training) struggle to adapt to the Adventure Time style because of the previously mentioned issue with the discrete artstyle of cartoons. A few hours of training on the adventure time dataset performed worse as it still figured out how to leverage its transfer methods to flat shading. Our final model was able to produce a much smoother video of the cat, although not yet fully stylized. While the stylized video is far from an impressive result, we see it prescribes brighter, poppy, solid colors to the video as it learns which is reminiscent of the style of Adventure Time.

6. V-CAST Application

Fig. 4 shows the interface of the application made to directly work with the V-CAST model. It is a simple python tkinter application that allows the user to upload a short video and style reference image. When the user clicks stylize, the output video downloads after inference.

6. Distribution of Labor

In alphabetical order

Alessandro Castillo: In charge of implementing the proposed changes to the CAST network. Created a new input system, cached redundant calculations for better training runtime, and new training loop.

Ray Forman: Worked with/against the datasets to be used in training. Also created the V-CAST application.

Stephen Pasch: Designed the video training framework modifications to CAST. Architected the flow based loss and implemented the function to find the optical flow between two images.

7. Conclusion

We found that V-CAST was able to produce video that was less temporally disruptive than the naive method. With further time we would have liked to compare against other methods of video stabilization and more extensive training on video, to see how a fully trained version would operate. Especially video with rapid motions, which our model performed poorly.

[7] Link to Cartoon Dataset: https://www.kaggle.com/datasets/volkandl/cartoon-classification

References

[1] Midjourney
https://www.midjourney.com/home
[2] AdCreative.AI
https://www.adcreative.ai/post/generative-ai-and-style-transfer
[3] Adobe Generative Style Transfer & Recolor
https://www.adobe.com/products/firefly/features/generative-match.html
[4]Yuxin Zhang, Fan Tang, Weiming Dong, Haibin Huang, Chongyang Ma, Tong-Yee Lee, and Changsheng Xu. 2022. Domain Enhanced Arbitrary Image Style Transfer via Contrastive Learning. In Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings (SIGGRAPH '22 Conference Proceedings), August 7–11, 2022, Vancouver, BC, Canada. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3528233.3530736
[5] LUCAS, B. D., AND KANADE, T. 1981. An iterative image registration technique with an application to stereo vision. In Proceedings of IJCAI, 674–679.
[6]Link to TGIF; Gif Dataset:
https://huggingface.co/datasets/HuggingFaceM4/