A Real-Time, Stable, Style-Invariant Architecture for Artistic Style Transfer of Videos

Ankush Swarnakar
Stanford University
ankushsw@stanford.edu

Michelle Bao
Stanford University
baom@stanford.edu

Abstract

Recent developments in video style transfer have either used a naive approach of applying stylization to individual frames, leading to temporal inconsistencies and flickering, or a computationally-intensive approach of using optical flow estimations to smooth inconsistencies between frames. Huang et al. recently introduced an adaptive instance normalization layer for images that allows for efficient, arbitrary style transfer. In this work, we present an efficient approach that produces stable, stylized videos in real-time for any content and style inputs, while maintaining the speed and straightforwardness of a single-image method. Our network uses a similar adaptive instance normalization approach to shift and scale content features to match style features and a restructured noise-resilient loss function to penalize the effect of differential noise on stylization, thus reducing popping and flickering between consecutive frames. Our method is nearly 25x faster than standard optical flow video stylization approaches and produces pastiches of high style- and content-quality.

1. Introduction

In recent years, algorithms for artistic style transfer to generate an image with the content of one input image and the style of another have progressed significantly. The earliest procedures iteratively update the generated pastiche image to minimize loss from the content image and style reference [4]. More modern architectures learn a network to produce a pastiche for a specific style and any given content image, allowing for real-time, content-invariant style transfer [10]. Neural network architectures have also been proposed to generate the parameters of the style transfer network given any style image, making the transfer process both style- and content-invariant [7].

Most advancements in style transfer architecture focus on style transfer applied to single images; little work has been done on artistic style transfer when applied to videos, which adds a temporal dimension to data. Video style transfer is fundamentally different than single image style transfer, because motion and noise between frames affects the stylization of different frames, which means that pastiche videos may not maintain the same temporal consistency as the original content videos; naively applying style transfer to individual, independent frames of videos generates flickering, false discontinuities, and inconsistencies between video frames [14]. Video style transfer has important applications in camera filters, animation and computer graphics, and data augmentation, all domains where single image style transfer has proved immensely useful [9]. Thus, it is important to develop smooth, stable architectures for video style transfer that operate with comparable speed and efficacy as single-image style transfer algorithms.

Some architectures for video style transfer use post-processing methods or optical flow estimations to help smooth stylized inconsistencies between frames; these operations can be extremely computationally intensive, which prevents the style transfer procedure from working in real-time [14]. We survey various style transfer architectures on single images and video processing techniques to develop an end-to-end, stable pipeline for video style transfer. The input to our pipeline is a sequence of frames corresponding to a video and a style reference; the output is a consistently-stylized series of the frames in the input video.

2. Related Work

2.1. Single-Image Style Transfer

2.1.1 Feed-Forward Methods

The most nascent forms of style transfer use a pre-trained convolutional neural network (CNN) to process an input image to generate a pastiche. The network calculates the output's loss from both the content image and style image, and applies traditional gradient descent on the network's input image to iteratively update it to generate a loss-minimizing pastiche [4]. Such procedures are effective, but require that a network retrain for every new combination of style and content images, which makes the style transfer process extremely slow and computationally expensive.

2.1.2 Image Transformation Network

Newer architectures propose the use of an image transformation network (ITN) that learns the parameters to convert any content image to a given style [10]. The network trains

by passing the content image through a CNN consisting of an encoder and decoder in order to generate a pastiche; the network then computes the pastiche's loss from the original content image and the style image, and uses these values to update the CNN's parameters instead of updating the input image through gradient descent [5].

ITNs allow for real-time style transfer of any content image to a specific style-domain. Unlike the canonical feed-forward method, ITNs do not require forward and backward passes to produce pastiches for multiple content images in the same style, significantly reducing computational complexity [9]. Rather, the ITN learns the parameters for a certain style which enables it to be truly content-invariant.

2.1.3 Adaptive Instance Normalization

ITNs are considerably faster than the canonical feed-forward method for style transfer, but still need to be retrained for every new style image. [12] shows that ITNs trained for different styles mostly differ across the parameters for the instance normalization layer of the CNN, between the encoder and decoder, suggesting that the encoding and decoding layers can be preserved between ITNs for different styles and only the instance normalization layer needs to be retrained.

[7] proposes an algorithm for fast style transfer that uses an adaptive instance normalization layer (AdaIN) between the encoder and decoder to learn the instance normalization parameters for any given style input. AdaIN enables the CNN to produce a pastiche given any content image and any style image, making the single image style transfer process real-time and style- and content-invariant.

2.2. Video Style Transfer

2.2.1 Naive Approaches

Naive approaches for style transfer on videos apply style transfer to individual frames of videos, stylizing each frame independently and incorporating no longitudinal information about the relationship between neighboring frames in videos [10]. These approaches effectively stylize videos, but produce artifacts called popping where similar regions in different frames are stylized drastically differently, resulting in highly inconsistent pastiche videos [6]. Popping generally arises from naive style transfer methods not incorporating motion between frames, noise between frames, and optical and luminance flow, which are important to preserving the temporal perceptual characteristics of the original content video.

2.2.2 Optical Flow Incorporation

For videos, much of the advancements in style transfer rely on optical flow to constrain temporal deviations between successive frames. In addition to the pixel data from individual content and style images, the network computes loss between the optical flow between two content images and two analogous pastiches [14]. This approach helps minimize pastiche variance between similar regions of different frames, resulting in a significantly more stable and smoother stylized video with little popping.

Though effective, incorporating optical flow into the model is incredibly computationally expensive, as it requires the use of 3D convolutional operators, which renders the training and stylizing process extremely slow [2]. Neural networks have been proposed to estimate optical flow from videos, but these procedures are still extremely slow, limiting the test-time run-time of the video style transfer network [17].

2.2.3 Post-Processing Methods

Many traditional video-processing methods exist to enhance color, contrast, exposure, etc. but result in temporally inconsistent videos since enhancements are applied independently to each frame; independent stylization of individual frames produces similar issues with popping. Many algorithms exist for post-processing to smooth out artifacts and temporal inconsistencies between frames; applying these methods to smooth out popping effects has proven effective, but is still quite computationally expensive, preventing real-time style transfer [11].

3. Approach

3.1. Overview

Generally, any method that incorporates data from the entire temporal dimension of a video will be incredibly computationally expensive and exponentially increases the number of operations and learnable parameters for the style transfer network; this inherently makes the network slow and not real-time. In our approach, we focus on improving independent style transfer on individual frames to produce more stable, stylized videos in real-time.

Much of the work in video style transfer methods shows that the primary contributor to popping is differential noise between different frames; regions of different frames may appear similarly to the human eye, but actually have fine-grained differences in noise that result in different stylizations by a computer [2]. We focus on ways to mitigate the effect of such noise on the stylization process, while maintaining the speed and features of modern single-image style transfer methods. Our approach uses an ITN with an adaptive instance normalization layer and a loss function that specifically penalizes the effect of random noise on stylization.

3.2. Encoder and Decoder

We begin with the ITN proposed in [7] that uses an encoder to first extract features from style and content images and then a decoder to generate the pastiche given the style and content features. Our encoder takes the first few layers of a pre-trained VGG-19 network from [15] trained for an image classification task; these layers include convolutional layers and max pooling layers that serve to downsample the

data and extract features that prove useful for image classification, image super-resolution, and style transfer. The encoder itself is agnostic to input image size, and the output size is dependent on the size of the input image [1]

We then use a decoder to map the content features back to the image space; during this mapping process, we use the style features to stylize the re-generated image to create a pastiche that preserves the content of the content image in the style of the style image. The decoder largely mimics the architecture of the encoder in reverse, and consists of several convolutional layers and nearest-neighbor upsampling to generate the pastiche image given style and content features. Inpsired by suggestions from [7], we remove all normalization layers from the decoder, since such layers make the ITN singificantly style-variant and detract from the adaptive instance normalization operator we later introduce.

3.3. AdaIN Layer

Between the encoder and the decoder, we introduce an adaptive instance normalization layer, called AdaIN that shifts and scales the content features to match the style features. Many ITN architectures use batch normalization to effectively scale and shift features for further processing; the scale and shift parameters are learned through the training of the ITN [12]. Batch normalization normalizes the data for every feature channel by first subtracting the mean of the data for a feature from every data point in that feature, and then dividing the difference by the standard deviation of the data. The normalized data is then scaled given the learned shift and scale parameters. At training time, batch normalization normalizes data using the minibatch statistics, but at test-time, it normalizes the data using the statistics encompassing the entire dataset or uses a running average of the statistics computed during the training process.

[16] suggests that batch normalization can be replaced with instance normalization for better results in style transfer; instance normalization normalizes data across features and samples, rather than just normalizing across features. Since instance normalization normalizes across samples too, it recomputes the normalization statistics at both test and runtime, meaning it is not as adversely affected by minibatch size. Vanilla instance normalization normalizes across channels and individual samples, following the formula

$$\text{IN}(x) = \gamma_{N,C}\frac{(x - \mu_{N,C})}{\sigma_{N,C}} + \beta_{N,C}$$

where N and C denote the sample and channel and γ and β denote the scale and shift parameters, respectively.

[3] suggests that for every style s, the instance normalization layer should have different scale and shift parameters γ^s and β^s. This conditional instance normalization allows the network to preserve the same parameters for the convolutional layers in feature extraction and image generation, but only change the instance normalization parameters. This increases the degree of style-invariance in the ITN, because only the instance normalization parameters need to be learned for every new style.

[7] proposes an adaptive instance normalization operator, which simply scales the content data to match the scale of the style data. It does this by setting the shift parameter γ to the standard deviation of the style data and the scale parameter β to the mean of the style data. Mathematically, the AdaIN operator is given by

$$\text{AdaIN}(x, y) = \sigma(y)_{N,C}\frac{(x - \mu(x)_{N,C})}{\sigma(x)_{N,C}} + \mu(y)_{N,C}$$

where x is the content data and y is the style data, and $\sigma(z)$ and $\mu(z)$ represent the standard deviation and mean of a dataset z. The AdaIN operator removes the γ and β parameters from the ITN, which means there are no learnable parameters for the instance normalization layers; rather, the effective shift and scale parameters $\sigma(y)$ and $\mu(y)$ are computed in real-time using the style features produced by the encoder. The removal of learnable parameters means that the ITN is truly style-invariant and can produce pastiches of any given style and content combination in real-time.

3.4. Modifications to Loss Functions

Our ITN trains to minimize three separate loss objectives: content, style, and noise.

3.4.1 Content Loss

We compute content loss between the pastiche and the original content by computing the sum of the Euclidean distances between the features of the pastiche and the AdaIN-shifted features of the content image. The AdaIN-shifted features of the content image are generated by passing the content image through the encoder and then the AdaIN operator. The features of the pastiche are generated by passing the pastiche through the encoder. Mathematically, the content loss is

$$\mathcal{L}_C = \sum (f(g(x)) - x)^2$$

where x is the AdaIN-shifted features of the content image, $g(x)$ is the generated pastiche image, and $f(g(x))$ are the features of the generated pastiche image [10].

3.4.2 Style Loss

We compute style loss by summing the Euclidean distances of the Gram matrices of the style image and the pastiche at every activation layer of the VGG-19 encoder. We compute the Gram matrix using

$$G(x)_{c,c'} = \frac{1}{H \times C \times W}\sum_H \sum_W \Phi(x)_{h,w,c}\Phi(x)_{h,w,c'}$$

where $\Phi(z)$ is the activation at a given layer, H is the image height, W is the image width, C is the number of features, and x is the image features [10].

[1] We start with the code presented at https://github.com/naoto0804/pytorch-AdaIN. We rewrite some of the code to better modularize it and also add in the noise-loss operator to make the network noise-resilient.

Given the Gram matrices for every activation layer, we can compute the style loss using

$$\mathcal{L}_S = \sum_j (G(\Phi_j(x)) - G(\Phi_j(g(x)))^2$$

where $\Phi_j(z)$ is the activation function for the jth activation layer, x is the original style image, and $g(x)$ is the pastiche [10].

3.4.3 Noise Loss

We hope to penalize the effect of noise on stylization through the training process. To do so, we introduce a noise loss operator inspired by [8]. We first add uniformly distributed noise on the range $[-R, R]$ to the input content image x, repeated T times, where R and T are hyperparameters. This process serves to create a content image that is visually similar to the original content image, but has non-trivial pixel-wise differences. We denote this function by $n(x)$, where x is the input content image and $n(x)$ is the noise-modified content image. We then pass both the original and the noisy input through the ITN.

Ideally, the noise would have no effect on the pastiche, and we want to penalize different stylizations resulting from introduced noise. Thus, we define a noise loss term $\mathcal{L}_N$ that computes the Euclidean pixel-wise distance between the output of the noisy content image and the output of the original content image. Mathematically

$$\mathcal{L}_N = \frac{1}{N} \sum_{j,k} (g(n(x))_{j,k} - g(x)_{j,k})^2$$

where N is the total number of image pixels, j and k represent the x-coordinate and y-coordinate of a given pixel, x is the original content image, $n(x)$ is the noisy content image, and $g(z)$ is the output of the ITN processing an input z. We add this operator to the repository we referenced in Section 3.2.

3.4.4 Total Loss

We sum the outputs of each loss function to get the total loss. Mathematically

$$\mathcal{L} = \lambda_C \mathcal{L}_C + \lambda_S \mathcal{L}_S + \lambda_N \mathcal{L}_N$$

where λ_C, λ_S, and λ_N represent the weights of the content, style, and noise loss, respectively. These are hyperparameters that we validate in the training process.

4. Dataset

For content images, we use MPI-Sintel, a publicly-available data set of animated videos for the evaluation of optical flow and apparent motion through time. The data set is commonly used to evaluate video style transfer models since it contains videos with very long sequences, large motions, motion blur, atmospheric effects, etc. The training scenes vary between 20 to 50 frames of resolution 1024 × 436 pixels per scene. The data set provides 23 training sequences and 12 testing sequences [1].

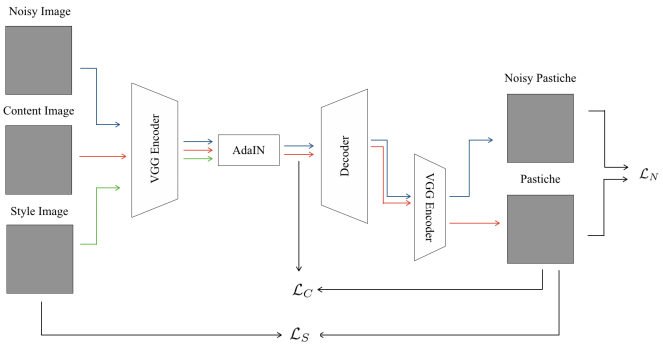

Figure 1: Proposed ITN architecture with encoder, AdaIN layer, decoder, and noise loss objective.

For style images, we use a selection of 20 paintings from the Kaggle Painter by Numbers challenge which cover a broad range of artistic styles and motifs. We use 14 style images to train the ITN, and 6 to test style-invariance in the model [13].

5. Evaluations

To evaluate our model, we train an two ITNs with adaptive instance normalization: one with noise loss, and another without noise loss. We preserve the parameters of the encoder from the pre-trained VGG-19 network at SIMONYAN, and learn the decoder parameters after introducing the new loss objective. We then compare the outputs of the two models to understand the contributions of the noise loss term. For both ITNs, we perform 160,000 iterations of stochastic gradient descent with an Adam optimizer with learning rate 1×10^{-2}, a minibatch size of 8, a content weight of $\lambda_C = 1$, and a style weight of $\lambda_S = 10$ as inspired by [7].

5.1. Noise Hyperparameters

Introducing a noise loss objective introduces three new hyperparameters: R or the scale of added noise, T or the number of times we generate and add random noise, and λ_N or the weight given to the noise term in computing loss. We find that $R = 40$ and $T = 1000$ produces the most significant reduction in popping without compromising the convergence of the model; additionally, we find that λ_N is the strongest determinant of the smoothing in the model. All 3 hyperparameters ultimately control the extent to which noise affects training, so only fine-tuning λ_N provides sufficient control over the extent to which noise affects training.

We tested the model with λ_N values of 0.5, 1, 10, 100, and 10000; noise weights over 10 prevented the model from converging during training. We evaluated the model with $\lambda_N = 0.5$ and 1.0. We found that $\lambda_N = 0.5$ yielded sufficient smoothing of popping without compromising stylization and also optimized our quantitative metrics most. Thus, our final noise hyperparameters were $\lambda_N = 0.5$, $R = 40$,

and $T = 1000$.

5.2. Qualitative Evaluations

5.2.1 Style Invariance

We first evaluate the effectiveness of the AdaIN operator to make our ITN style-invariant. We use the same ITN to generate pastiches of a given ground truth image in 5 different styles. For each style, the ITN uses the exact same encoder and decoder parameters and does not learn any new parameters; it only uses the AdaIN operator to shift the content features between the encoder and decoder layers.

Qualitative evaluation of the 5 pastiches for a given ground truth image show that the AdaIN operator does make the ITN style-invariant. Pastiches are quite similar to their style reference image in color composition, artistic motifs, and texture. Fig. 2A shows one video frame stylized in 5 different styles; all of the pastiches maintain the structure of the content image but effectively introduce the motifs of the style image. The ITN with AdaIN produces pastiches of high perceptual quality, content preservation, and stylization, indicating that AdaIN can effectively replace models that must relearn encoding and decoding parameters for different styles.

Our ITN primarily fails to stylize images effectively when the content and/or style inputs are fairly sparse and homogenous. Fig. 2B shows the stylization of a video frame depicting a mountain, which contains a fairly vast homogenous region and little contrast. The style image is fairly monochromatic, lacks texture, and is sparsely populated too. The pastiche only seems to preserve the color composition of the style image, but not textural or artistic motifs are incredibly apparent. Additionally, the content of the pastiche is largely unclear; it is incredibly hard to discern that the image depicts a mountain and the pastiche does not look very similar to the original content image. Still, this style transfer task would prove difficult for any style transfer architecture because there are fundamentally not that many features to extract and utilize to produce a pastiche. AdaIN is therefore still a comparably-effective ITN layer for truly style-invariant style transfer that preserves the quality of generated pastiches.

5.2.2 Reduced Popping

To qualitatively evaluate whether our model effectively reduced popping effects and motion stylization artifacts in videos, we processed sequential frames of videos with an ITN with and without the style loss operator. Generally, the ITN with the style loss operator significantly reduces popping between frames and enhances the perceived consistency across the temporal dimension.

Fig. 3 displays a series of three frames stylized by the baseline ITN without noise-resilience and our ITN with noise-resilience. The sky in the ground truth image has fairly consistent color composition and homogeneous texture across all three frames; this would ideally produce consistent stylizations in the pastiche images that are also consistent in color composition and texture. The baseline ITN without noise resilience produces a highly-stylized sky that does not maintain the homogeneity perceived in the ground truth image. This results in popping between frames, as is highlighted by the red ovals. Our ITN with noise resilience still effectively stylizes the sky, but maintains the consistency of the texture and the color of the sky that is observed in the original content image. This corroborates that the introduced noise loss operator effectively reduces popping and enhances temporal consistency.

Additionally, our ITN produces smoother output pastiches which are more consistent with the smooth input frames in Fig. 2. The baseline ITN also introduces a discolored halo surrounding the female avatar that is not present in the original input frame; our ITN with noise-resilience does not present as significant of a halo. Ultimately, the ITN with noise-resilience effectively maintains the temporal consistency of the input frames without significantly compromising the degree of stylization.

5.3. Quantitative Evaluations

5.3.1 Temporal Consistency

We sought to quantify the temporal consistency of our model in relation to the ground truth by using calculating the pixel-wise Euclidean distance between consecutive frames. We defined the temporal error as the following:

$$E_{temporal}(\hat{x}, \hat{y}) = \sum_{t=1}^{T-1} \frac{((\hat{x}^t - \hat{x}^{t+1}) - (\hat{y}^t - \hat{y}^{t+1}))^2}{\hat{x}_k^t - \hat{x}_k^{t+1} + \epsilon}$$

$$E_{tempAvg}(\hat{x}, \hat{y}) = \frac{E_{temporal}(\hat{x}, \hat{y})}{T - 1}$$

where T is the total number of frames, $\hat{x}$ is the ground truth frame, $\hat{y}$ is the stylized model frame, and $\epsilon = 0.01$, inspired by [11]. By dividing the difference between the ground truth and the model's differences in consecutive frames by the ground truth's difference in consecutive frames, the error is scaled against the ground truth , resulting in a high temporal error if the model's temporal consistency is very different from that of the ground truth, or a temporal error approaching zero if the temporal consistency is the same. We calculated the average temporal error across all frames for each video-style combination.

Table 1 illustrates that for select videos, the ITN with noise-resilience decreased the temporal error resulting from stylization by over 60%. Averaging over possible all video-style combinations, the temporal error decreased by 6.69%, indicating that the ITN both perceptually and quantitatively reduces popping artifacts.

5.3.2 SSIM

While the temporal error captures the objective similarity maintained across frames, we also wanted to quantify how well the perceptual similarity was being maintained after

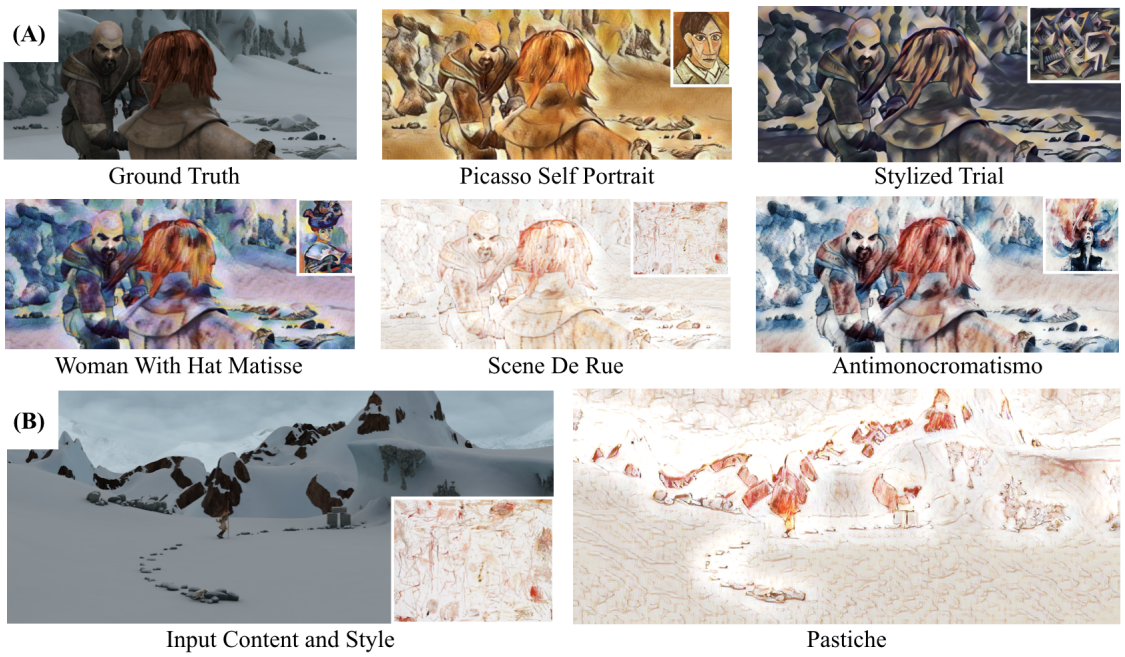

Figure 2: Qualitative evaluation of stylization and style-invariance. (A) Frame 23 of ambush_1 in 5 different test styles (style-invariance). (B) Frame 11 of mountain_2 stylized as scene de rue (failed stylization).

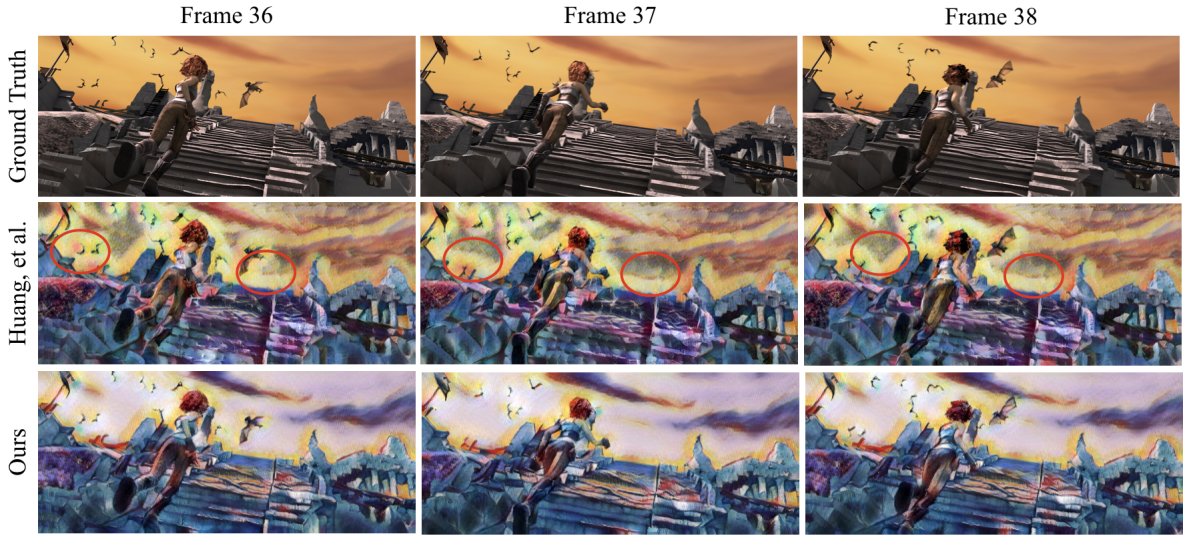

Figure 3: Stylization of three sequential frames from temple_1 in the style of woman with hat matisse by Huang AdaIN model and our model. Cases of of popping in [7] model but not ours are circled in red.

stylization. Hence, we calculated the Structural Similarity Index (SSIM) to compare the similarities within pixels between the ground truth and the stylized images. SSIM uses a value of 1.00 to indicate perfect similarity and a value of -1.00 to indicate perfect dissimilarity. Just as was done for temporal error, we calculated the average SSIM value across all frames for each video-style combination.

As seen in the examples highlighted in Table 1, we had results that increased the SSIM values by over 70%. Averaging over all video-style combinations, the SSIM increased by 3.20%.

5.3.3 Less Successful Examples

While our model improved both temporal error and SSIM aggregately, the average temporal error and SSIM values are imperfect because it fails to recognize the patterns between successful and less successful examples. To better understand why our model was less successful for some styles or some videos, we performed a qualitative analysis of the results that were the most temporally inconsistent.

While the styles that reduced temporal error the most were impronte_d_artista and the_resevoir_at_poitiers, averaging -

Table 1: Examples of Temporal Error and SSIM Percent Change

Video	Style	Temp. Error % Change	SSIM % Change
mountain_2	the_resevoir_at_poitiers	-64.93	17.11
market_4	impronte_d_artista	-56.69	0.77
cave_3	en_campo_gris	1.08	74.03
bamboo_3	en_campo_gris	-16.07	60.31

43.4% and -22.9% respectively, the styles that reduced temporal error the least were asheville and flower_of_life, at 6.2% and 4.4% respectively. We chose to analyze the asheville style, which had lowest values for temporal error decrease and SSIM increase, as well as the trial style, with third-to-last and second-to-last, respectively.

5.3.4 Consecutive Frame Comparison of Temporally Inconsistent Examples

As shown in the white rectangles of Figure 4, there are some areas of temporal inconsistency, in which consecutive frames with little movement are stylized significantly differently, resulting in the higher temporal error. This error is magnified in the case of two very similar ground truth frames from the construction E_{temp}, which divides by the pixel-wise ground truth differences — this is most likely what happened in the cave_3 video stylized by en_campo_gris, with the slight increase in temporal error shown in the third row of Table 1. Furthermore, if these areas in the ground truth are essentially constant, such as the snow in Figure 4, the perceptual similarity measured by the SSIM metric is low as well. The styles that were less successful stood out as well - both asheville and trial are visually chaotic. Therefore, our model is more likely to fail in stylizing videos with low contrast using styles with high tonal contrast.

5.3.5 Run-Time Efficiency

Our model aims to preserve adaptive instance normalization and reduced popping for video style transfer, while maintaining comparable run-time speeds to single-image style transfer algorithms. Table 2 shows the test-time run time in frames per second of three style transfer architectures; runtimes were computed while testing the network on a single NVIDIA V100 GPU. The video style transfer model proposed by [14] which incorporates optical flow to minimize popping is an order of magnitude slower than single-image style transfer with AdaIN proposed by [7]. Our network runs slightly slower than Huang, but is still considerably faster than the optical flow architecture without compromising temporal consistency enhancement. Thus, our architecture offers a style-invariant, popping-reducing solution that operates in real-time.

Table 2: Run-Time Comparison of Style Transfer Architectures

Model	Average Frames Per Second
Ruder et al. (Optical Flow) [14]	0.8
Huang et al. (AdaIN) [7]	20.8
Ours (AdaIN + Noise-Resilience)	19.9

6. Conclusion

Here, we investigate an image-transformation network for smooth, real-time, style- and content-invariant style transfer of videos. Our model uses an adaptive instance normalization layer inside an image transformation network to guarantee style-invariance. We also introduce a new noise loss objective for training the model that minimizes the effect of noise on image stylization to reduce popping effects between frames.

Our results demonstrate that adaptive instance normalization is highly effective at generating high-quality outputs with great similarities to both content and style inputs in a style-invariant network. We preserve the encoder and decoder parameters of the image transformation network, and only use the AdaIN layer to generate instance normalization parameters for each new style; thus, our model learns no new parameters for different styles and can still produce pastiches across a wide range of style and content domains. The effectiveness of AdaIN further corroborates that instance normalization parameters are key determinants of style transfer architectures and outputs.

Our results also demonstrate that training our ITN to be noise-resilient with the addition of a noise-loss objective is effective at qualitatively and quantitatively improving temporal consistency between frames. Similar regions of different frames are stylized similarly in our ITN, while being stylized differently in an ITN trained without noise resilience. It enhances temporal consistency without compromising the run-time efficiency of single-image style transfer architectures, unlike other video style transfer networks that incorporate optical flow information and operate slowly. The considerably better performance of our noise-resilient model also confirms that noise between frames is one of the largest contributors to video-processing artifacts across the temporal dimension.

In future work, we hope to investigate how masking video frames to selectively add noise to temporally-consistent regions during training might improve model performance. This might allow us to preserve higher-degrees of stylization while still ensuring temporal consistency between frames. Additionally, we hope to incorporate adaptive instance layers and noise-resilience into other proposed video style transfer architectures, like those that use traditional smoothing or

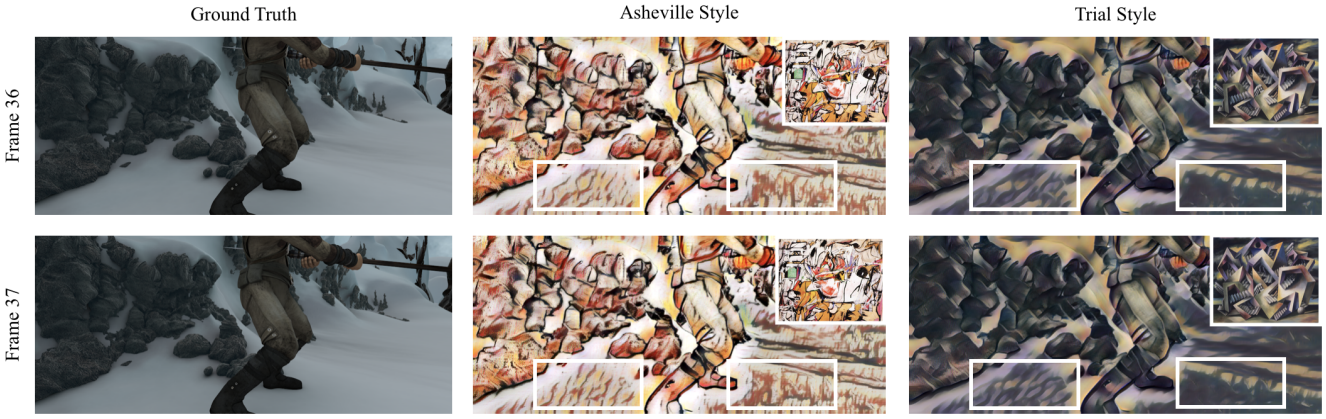

Figure 4: Stylization of two consecutive frames from ambush_3 in the style of asheville and trial. Cases of temporal inconsistency and dissimilarities to the ground truth highlighted in white rectangles.

optical flow metrics, to understand how these architectures interact. Lastly, we hope to explore how our model can be used for camera filters, data augmentation, computer graphics, and other applications within the domain of style transfer.

7. Contributions & Acknowledgements

Ankush and Michelle developed the idea for the noise-resilient, style-invariant ITN together. We began with a PyTorch implementation of [7] on GitHub at https://github.com/naoto0804/pytorch-AdaIN. Ankush then modified some of the adaptive instance normalization layer to take in non-square inputs and added the noise-addition procedure and noise loss objective to train the model. Michelle developed the metrics for quantitative evaluation of the model (temporal consistency & SSIM) and processed ∼9 GB of test data for quantitative analysis. We would like to acknowledge the CS231N course staff for introducing us to this field and their guidance throughout the quarter and this project.

References

[1] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *European Conf. on Computer Vision (ECCV)*, Part IV, LNCS 7577, pages 611–625. Springer-Verlag, Oct. 2012.

[2] D. Chen, J. Liao, L. Yuan, N. Yu, and G. Hua. Coherent online video style transfer. *CoRR*, abs/1703.09211, 2017.

[3] V. Dumoulin, J. Shlens, and M. Kudlur. A learned representation for artistic style. *CoRR*, abs/1610.07629, 2016.

[4] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[5] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, June 2016.

[6] M. Honke, R. Iyer, and D. Mittal. Photorealistic style transfer for videos. *CoRR*, abs/1807.00273, 2018.

[7] X. Huang and S. J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *CoRR*, abs/1703.06868, 2017.

[8] A. d. B. Jeffrey Rainy. Stabilizing neural-style transfer for video, February 2018.

[9] Y. Jing, Y. Yang, Z. Feng, J. Ye, and M. Song. Neural style transfer: A review. *CoRR*, abs/1705.04058, 2017.

[10] J. Johnson, A. Alahi, and F. Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[11] W. Lai, J. Huang, O. Wang, E. Shechtman, E. Yumer, and M. Yang. Learning blind video temporal consistency. *CoRR*, abs/1808.00449, 2018.

[12] Y. Li, N. Wang, J. Liu, and X. Hou. Demystifying neural style transfer. *CoRR*, abs/1701.01036, 2017.

[13] K. Nichol. Painter by numbers, 2016.

[14] M. Ruder, A. Dosovitskiy, and T. Brox. Artistic style transfer for videos and spherical images. *CoRR*, abs/1708.04538, 2017.

[15] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *2015 International Conference on Learning Representations (ICLR)*, 2015.

[16] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. *CoRR*, abs/1603.03417, 2016.

[17] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. Deepflow: Large displacement optical flow with deep matching. In *2013 IEEE International Conference on Computer Vision*, pages 1385–1392, Dec 2013.