Video Transformation using Retargeting and Artistic Style Transfer

JOEY LITALIEN*, McGill University

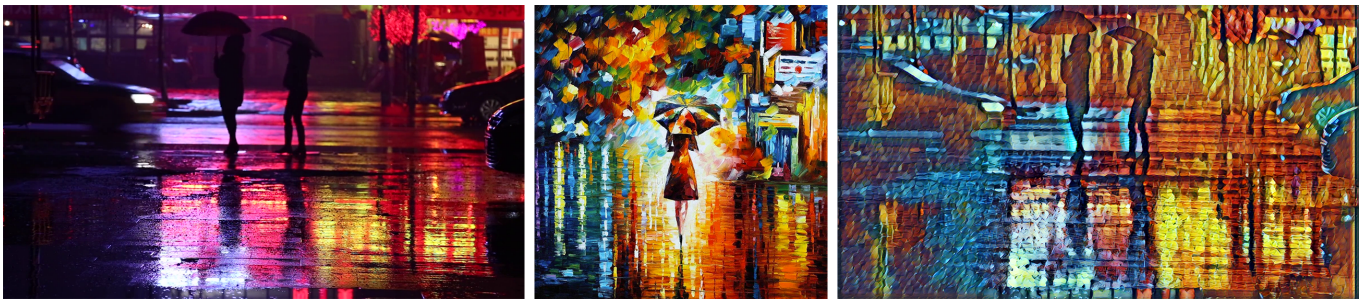


Fig. 1. Transforming a video frame. Left: Original frame with dimensions 1840×1034 pixels. Middle: *Rain Princess* oil painting by Leonid Afremov. Right: Seam insertion is first used to enlarge the frame to 1920×1080 pixels ($16:9$ ratio). The artistic style is then transferred to this image using a deep convolutional neural network with pretrained weights to accelerate the process.

Additional Key Words and Phrases: Video transformation, retargeting, seam insertion, artistic style transfer, convolutional neural networks.

1 INTRODUCTION

In this project, we use image retargeting and artistic style transfer to transform a video. By applying the algorithms sequentially on the individual frames, we are able to first enlarge and then stylize the video. The structure of this report is as follows. First, we introduce seam carving and insertion, and show how can this be used to do content-aware video resizing. We then move to perceptual loss convolutional neural networks (CNNs) for artistic style transfer, and describe how the model is meticulously constructed and trained. Once the mathematical foundations of both techniques are layed down, we explain their implementations in detail, specifying computational constraints encountered in the process. We finally underline some of the limitations in the frame-to-frame technique by showing comparisons between intermediate transformations.

2 SEAM CARVING AND INSERTION

2.1 Overview

Seam carving is a method for resizing an image to a desired target size with the goal of preserving its content. First introduced by [Avidan and Shamir 2007], it is a *content-aware* algorithm that supports both reduction (seam carving) and expansion (seam insertion). A *seam* is a monotonic and connected path of pixels that spans the image along a given axis (horizontal or vertical). Seam carving/insertion uses an energy function defined on the pixels and successively removes/inserts minimum energy paths from the image.

*ID 260532617. Final Project Report for ECSE683: *Computational Photography*, Fall 2017. Due December 14, 2017. Presented to Prof. James J. Clark, Dept. of Electrical & Computer Engineering.

Author's address: Joey Litalien, McGill University, QC, joey.litalien@mail.mcgill.ca.

2.2 Energy Function

The original paper suggests various ways to identify unimportant regions of an image. A simple technique to do so is to determine pixel paths where the gradient is the lowest. Hence, we define the *energy function* on an image I as

$$e(I) = \left|\frac{\partial I}{\partial x}\right| + \left|\frac{\partial I}{\partial y}\right|. \tag{1}$$

Here, the gradient is computed via the Sobel operator, which uses 3×3 kernel convolutions with I to calculate approximations of the partial derivatives. We then take the L^1-norm.

2.3 Seams

We now formally define a seam. Let I be an $n \times m$ image and define a *vertical seam* to be

$$s^x = \{s_i^x\}_{i=1}^n = \{(x(i), i)\}_{i=1}^n, \quad \text{s.t.} \quad |x(i) - x(i-1)| \leq 1, \forall i, \tag{2}$$

where $x : [n] \to [m]$ is an index mapping on the pixel space. In other words, s^x is a path from top to bottom with one and only one pixel in each column. We can similarly define a *horizontal seam* s^y with a mapping $y : [m] \to [n]$. The pixels of a seam s_i (*e.g.* vertical) is denoted as $I_s = \{I(s_i)\}_{i=1}^n$. To reduce the width of the original I by one, we simply delete I_s from I and shift the pixels to compensate for the missing path. Thus, removing a seam s_i has only a local effect along the neighboring pixels of s_i.

2.4 Optimal Seam Detection

Given some energy function e, we further define the *cost* of a seam as $E(s) = E(I_s) = \sum_i e(I(s_i))$. We want to minimize this cost:

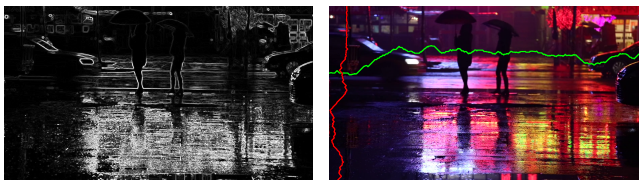
$$s^* = \min_s E(s) = \min_s \sum_{i=1}^n e(I(s_i)). \tag{3}$$

Given $e(I)$, we can solve this optimization problem using dynamic programming by starting from the second row of the image and

move row-by-row by updating a cumulative energy matrix M of same dimensions. Each entry (i, j) of M is then computed as

$$M(i,j) = e(i,j) + \min(M(i-1,j-1), M(i-1,j), M(i-1,j+1)). \quad (4)$$

Once this matrix is constructed, we simply backtrack from the bottom up to obtain the minimal connected vertical seam s^*. This is done by selecting the minimum value of the last row of M as a starting point. The process for horizontal seams is similary defined.



(a) Energy map $e(I)$ (b) Vertical (green) and horizontal (red) seams

Fig. 2. The two main components of the seam carving algorithm, applied to the image in Fig. 1.

The authors also describe a way to pick an optimal seam ordering via a transport map T that specifies, for each desired target image size $n' \times m'$, the cost of the optimal sequence of horizontal and vertical seam removal operations. This strategy is costly however since it requires dynamic programming to solve for T. We thus use a naive implementation where we deal with width first, and then height to significantly reduce the running time.

2.5 Seam Insertion

It is understood that seam carving is an *iterative process*: every step of the algorithm reduces or increases the chosen dimension by one. To enlarge an image (which is the case in this project), we simply inverse the method. If we want to expand an image by k, we find the first k seams for removal and duplicate them and average them with the neighboring pixels. For instance, for a vertical seam s^x, the pixel $p = (i, j) \in I_s$ would be an equal blend of pixels $(i, j-1)$ and $(i, j+1)$. If p lies on the boundary of I (*e.g.* $i = 1$), we give it the value of its immediate neighbor (*e.g.* $p = I(i+1, j)$).

3 PERCEPTUAL LOSS STYLE TRANSFER

3.1 Overview

The approach used by [Johnson et al. 2016] is to train a feedforward transformation network for image transformation tasks, but rather than using *per-pixel* loss functions depending only on low-level pixel information, we train the network using *perceptual* loss functions that depend on high-level features from a pretrained network. The advantages of this technique are that during training, perceptual losses measure image similarities more robustly than per-pixel losses, and at run-time the transformation networks can be evaluated at interactive rates.

More formally, the system has two main components: an *image transformation network* f_W and a *perceptual loss network* ϕ that is used to define multiple loss functions $\ell_1, \ldots, \ell_k$. The image transformation network is a deep residual CNN parameterized by weights W, and maps an input image I into an output image $\hat{y} = f_W(I)$. Every loss function calculates the difference between the output image $\hat{y}$ and a target image y_i. Stochastic gradient descent (SGD) is used to minimize the weighted combination of these loss functions:

$$W^* = \arg\min_W \mathbf{E}_{I,\{y_i\}} \left[\sum_{i=1}^{k} \lambda_i \ell_i(f_W(I), y_i) \right]. \quad (5)$$

The key here is that pretrained CNNs for image classification have already learned to encode perceptual and semantic information, and thus can be used to define the loss functions in (5). To this extent, we can define a perceptual loss network ϕ pretrained for image classification as a fixed loss network. Hence, f_W is trained using ϕ as its loss, which is itself as CNN. In our case, ϕ uses the 19-layer VGG network [Simonyan and Zisserman 2015] pretrained on ImageNet [Russakovsky et al. 2015]. This is shown in Figure 4.

3.2 Image Transformation Network

The body of f_W comprises five residual blocks [He et al. 2016] using the architecture of [Gross and Wilber 2016]. All nonresidual convolutional layers are followed by batch normalization [Ioffe and Szegedy 2015] and ReLU nonlinearities with the exception of the output layer, which uses a scaled tanh. This ensures that the output has pixels in the range $[0, 255]$. The interested reader is referred to the orignal paper for more information about the choice of hyperparameters, such as convolutional filter sizes and residual connections.

3.3 Perceptual Loss Functions

To measure high-level perceptual and semantic differences between images we define two perceptual loss functions, the *feature reconstruction loss* $\ell^{\phi}_{\text{feat}}$ and the *style reconstruction loss* $\ell^{\phi}_{\text{style}}$ that measure differences in *content* and *style* between images. Given an input image I, we have an associated content target y_c and a style target y_s. The content target y_c is the input image I and the output image $\hat{y}$ should combine the content of $I = y_c$ with the style of y_s. We thus train one network per style target.

Feature Reconstruction Loss. The main idea behind feature reconstruction is that we want to encourage the pixels of the output image $\hat{y} = f_W(I)$ to have similar feature representations than the pixels of the target image y, as computed by the loss network ϕ. To further understand how we do this, let $\phi_j(I)$ be the activations of the j-th layer of the network ϕ when processing the image I. If j is a convolutional layer then $\phi_j(I)$ is a feature map of size $C_j \times H_j \times W_j$, where C_j is the number of filters and $H_j \times W_j$ are their dimensions. In this case, the feature reconstruction loss is the Euclidean distance between feature representations:

$$\ell^{\phi,j}_{\text{feat}}(y, \hat{y}) = \frac{1}{C_j H_j W_j} \left\| \phi_j(y) - \phi_j(\hat{y}) \right\|_2^2. \quad (6)$$

Put simply, this loss penalizes the output image $\hat{y}$ when it deviates in content from the target y.

Style Reconstruction Loss. In a similar vein, we also want to penalize differences in style such as different colors, textures and common patterns. We can achieve this desired effect by borrowing ideas from

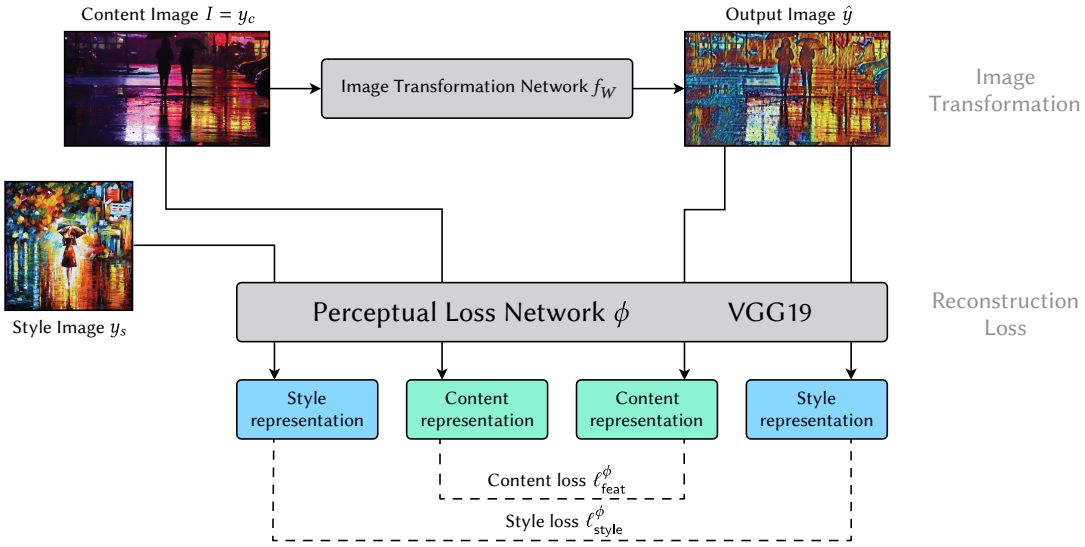


Fig. 3. Model architecture. We first train an image transformation network f_W to transform input images into output images. We use a loss network ϕ pretrained for image classification to define perceptual loss functions that measure perceptual differences in content and style between images. The loss network remains fixed during the training process.

[Gatys et al. 2015]. Define the *Gram matrix* $G^{\phi}_j(I)$ to be the $C_j \times C_j$ matrix whose elements are given by

$$G^{\phi}_j(I)_{p,q} = \frac{1}{C_j H_j W_j} \sum_{h=1}^{H_j} \sum_{w=1}^{W_j} \phi_j(I)_{h,w,p} \phi_j(I)_{h,w,q}. \tag{7}$$

The reasons why this captures information about which features tend to activate together is described in the paper. We can then define the style reconstruction loss as the squared Frobenius norm of the difference between the Gram matrices of the output and target images:

$$\ell^{\phi}_{\text{style}}(y, \hat{y}) = \left\| G^{\phi}_j(y) - G^{\phi}_j(\hat{y}) \right\|^2_F. \tag{8}$$

Combining Losses. Given style and content targets y_s and y_c and layers j and j' at which to perform feature and style reconstruction, an image $\hat{y}$ is finally generated by solving

$$\hat{y} = \arg\min_y \ \lambda_c \ell^{\phi,j}_{\text{feat}}(y, y_c) + \lambda_s \ell^{\phi,j'}_{\text{style}}(y, y_s) + \lambda_{\text{TV}} \ell_{\text{TV}}(y), \tag{9}$$

where $\lambda_c, \lambda_s, \lambda_{\text{TV}}$ are scalars, y is initialized with white noise, and optimization is performed using L-BFGS. These scalars can be used to tweak the model by setting how much we value style and content separately. ℓ_{TV} is a total variation regularizer to enforce spatial smoothness in the output.

4 IMPLEMENTATION AND RESULTS

4.1 Choice of Video

The video we chose is a clip of two women walking on a rainy night trying to catch a cab. The MPEG-4 Part 14 (4210 kb/s, .mp4) video was obtained from Pexels.com. The video was originally 1:01-long at 25 frame per second, but was later clipped to 20 seconds. The original resolution is 1840×1034 pixels ($920:517$ ratio), making it a convenient candidate for image enlargement to 1080p. The style used for artistic style transfer is the oil painting *Rain Princess* by Belarusian artist Leonid Afremov, shown in Figure 1. Note that the style need not be similar to the image content.

4.2 Video-Frames Conversion

Video to Frames. We first extract the individual frames from the video using FFmpeg. We retrieve the framerate using FFprobe and pipe it to FFmpeg to decompose the video into loseless Bitmap images in native resolution. The chosen video is 20-second long at 25 frame per second (exact rate), so 500 images are sequentially created, totalling 2.9 GB of raw data.

Frames to Video. To generate a video from a new sequence of augmented frames, we use the same command-line tool with encoding parameters. A constant rate factor of 23 is used to achieve loseless quality, and color encoding is done via YUV420p. The H.264 format is used for video compression, resulting in a full HD video.

4.3 Content-Aware Frame Resizing

Seam insertion is done using ImageMagick as other, more popular frameworks (*e.g.* OpenCV, scikit-image) do not support upsampling. The implementation is straightforward: for each images in the raw data directory, the algorithm reads the frame, computes its Sobel gradient from which it computes the new seams, and then scales the image. This optimized version makes use of multi-threading via OpenMP, and takes 2.5 seconds per frame on average. To visualize the energy map and the seams, we use a simple implementation in Python 3.6 that uses SciPy to compute the image gradient.

Fig. 4. Resizing a frame. Using seam insertion, we can enlarge the image by "filling up" the green border.

4.4 Neural Style Transfer

Training. [Johnson et al. 2016] train the image transformation network on the MS-COCO 2014 dataset [Lin et al. 2014]. Since this requires a lot of computing power, we opt for a pretrained model on the chosen painting. The resulting network was trained using the hyperparameters described in the original paper, except that the input image is 512×512 for a better resolution. We use a TensorFlow port [Engstrom 2016] from the original implementation in Torch [Johnson 2016].

Inference. To accelerate the transfer of style, we use a NVIDIA GeForce GTX 780 GPU (3 GB of VRAM) with CUDA 8 and cuDNN 6, a GPU-accelerated library of primitives for deep neural networks. This decreases the evaluation time to roughly 4.2 seconds per frame, for a total of 35 minutes for the entire video.

4.5 Overall Performance

Below is a summary table with performance statistics regarding the algorithms used for transforming the video.

Table 1. Algorithms Relative Performance

1st	SEAM INSERTION	
	Per frame (Avg)	Total
Size	5.9 MB	2.95 GB
Time	2.5 s	20 mins 50 s
2nd	NEURAL STYLE TRANSFER	
	Per frame (Avg)	Total
Size	6.1 MB	3.05 GB
Time	4.2 s	35 mins

4.6 Limitations

Our method is not without its limitations. In particular, temporal stability is an issue when using seam insertion on static frames. Since we assume temporal independence between the extracted frames, visual artifacts such as rigid shapes deformation and camera wiggling are present in the final video. This is mainly because when enlarging specific frames, the energy maps can change drastically. This is the case when a background car crosses the street for instance. In this example, the horizontal path of minimal energy happens to go across the parked car's bumper on the right-hand side of the video. This is shown in Figure 5.

(a) Original frame. The car bumper retains its shape, as it should.

(b) Enlarged frame (before style transfer). The car bump changes shape when a car crosses in the background despite being rigid.

Fig. 5. Visual artifacts, such as rigid body deformation and camera wiggling, are present in the final video.

One way to alleviate this problem would be to use improved seam carving for videos as described by [Rubinstein et al. 2008]. This approach essentially replaces the dynamic programming part with graph cuts to remove or insert 2D seam manifolds from 3D space-time volumes. This effectively removes any kind of flickering in the output, since temporal coherence is taken into account.

5 CONCLUSION AND FUTURE WORK

In this project, we successfully used image retargeting and artistic style transfer to transform a 20-second video. By applying two algorithms one after the other on the 500 extracted frames, we were able to enlarge and stylize the video. Seam insertion was first used to resize the original dimension to 1080p, and perceptual loss convolutional neural networks were employed to transfer the style of an oil painting to the content of the video. We observed that temporal stability was an issue when retargeting; however this visual artifacts is attenuated by the painting style applied afterward. Running these algorithms sequentially took less than an hour using a GPU to accelerate the CNN inference.

In the future, it would be interesting to implement better algorithms to achieve our initial goal. Using [Rubinstein et al. 2008] would definitely improve the flickering issues when retargeting, although cloud computing would be necessary to do so for HD images. Investigating other artistic style transfer techniques such as recent work in universal style transfer [Li et al. 2017] would also be a promising path to explore.

REFERENCES

Shai Avidan and Ariel Shamir. 2007. Seam Carving for Content-aware Image Resizing. *ACM Trans. Graphs* 26, 3, Article 10 (July 2007).

Logan Engstrom. 2016. fast-style-transfer. https://github.com/lengstrom/fast-style-transfer. (2016).

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2015. A Neural Algorithm of Artistic Style. (2015). arXiv:arXiv:1508.06576

S. Gross and M. Wilber. 2016. Training and investigating residual nets. http://torch.ch/blog/2016/02/04/resnets.html. (2016).

K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Francis Bach and David Blei (Eds.), Vol. 37. PMLR, Lille, France, 448–456.

Justin Johnson. 2016. fast-neural-style. https://github.com/jcjohnson/fast-neural-style. (2016).

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual Losses for Real-Time Style Transfer and Super-resolution. In *European Conference on Computer Vision*.

Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. 2017. Universal Style Transfer via Feature Transforms. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 385–395.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision (ECCV)* (2014-01-01).

Michael Rubinstein, Ariel Shamir, and Shai Avidan. 2008. Improved Seam Carving for Video Retargeting. *ACM Trans. Graphs* 27, 3 (2008), 1–9.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252.

K. Simonyan and A. Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.