Neural Style Transfer on Images and Videos

Karthik Venkatesan, Rishu Prakamya Dutt
New York University
{kv730, rpd302}@nyu.edu

1 Introduction

Style transfer is the technique of recomposing images in the style of other images. This is done by taking as input a content image, that we want to repaint in a new style, and a style image, whose style we will use to repaint the content image. In this project, we explored the various ways to implement Neural Style Transfer using a Convolutional Neural Network and tune the parameters to find out which model gives the best results. We also applied Neural Style Transfer to videos. We give a brief description of ideas related to our project in section 2, and explain our method in section 3. We discuss the different parameters we tuned and show the results of the various experiments we performed in section 4.

2 Related work

1. Neural Style Transfer
Gatys et al. perform artistic style transfer, combining the content of one image with the style of another by jointly minimizing the feature reconstruction loss and a style reconstruction loss, also based on features extracted from a pretrained convolutional network. Their method produces high quality results, but is computationally expensive since each step of the optimization problem requires a forward and backward pass through the pretrained network.

2. Fast Neural Style Transfer
This was an extension on Gatys et al. proposed by Johnson et al. Recent methods for style transfer typically train feed-forward convolutional neural networks using a per-pixel loss between the output and ground-truth images. Parallel work has shown that high-quality images can be generated by defining and optimizing perceptual loss functions based on high-level features extracted from pretrained networks.However, Johnson et al. train feedforward transformation networks for image transformation tasks using perceptual loss functions that depend on high-level features from a pretrained loss network. During training, perceptual losses measure image similarities more robustly than per-pixel losses, and at test-time the transformation networks run in real-time. Then loss network used is a pre-trained VGG16 model.

3. Instance Normalization Around the same time in 2016, Ulyanov et. al came up with a similar generative approach to style transfer known as Texture Networks. Though the two approaches produced good results they were still lacking in quality to the slower Gatys. et. al implementation. Ulyanov proposes to replace the batch normalization layer in the networks with Instance Normalization to remove instance-specific contrast information from the content image.

Controlling Perceptual Factors
In this paper Gatys et al. explains how to control factors like scale, color and spatial features. In fast style transfer, the color is controlled by combining the luminance channel of the stylized image with the color channel of the content image. Alternatively we can train the network just using the luminance channels and later add color to it. To control space in fast style transfer, we can use guide masks. The style loss is modified to account for the guided gram matrix. The fast neural transfer algorithm is very robust to spatial control.

Our code can be found at `https://github.com/gamemaker007/Fast_Neural_Style_Transfer`

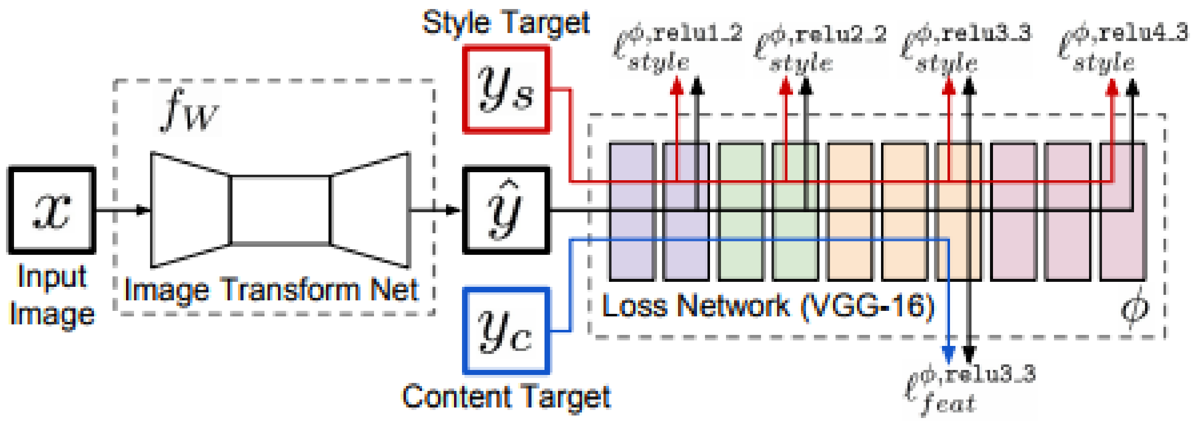

Style Target
y_s
$\ell_{style}^{\phi,relu1\_2}$
$\ell_{style}^{\phi,relu2\_2}$
$\ell_{style}^{\phi,relu3\_3}$
$\ell_{style}^{\phi,relu4\_3}$
f_W
x
Input Image
Image Transform Net
$\hat{y}$
y_c
Loss Network (VGG-16)
ϕ
Content Target
$\ell_{feat}^{\phi,relu3\_3}$

Figure 1: Architecture of our Model

Layer	Activation size
Input	$3 \times 256 \times 256$
$32 \times 9 \times 9$ conv, stride 1	$32 \times 256 \times 256$
$64 \times 3 \times 3$ conv, stride 2	$64 \times 128 \times 128$
$128 \times 3 \times 3$ conv, stride 2	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
$64 \times 3 \times 3$ conv, stride 1/2	$64 \times 128 \times 128$
$32 \times 3 \times 3$ conv, stride 1/2	$32 \times 256 \times 256$
$3 \times 9 \times 9$ conv, stride 1	$3 \times 256 \times 256$

Figure 2: Architecture of our Image Transformation Network

3 Method

Feature Construction Loss
This loss tells the network to match the features of content image rather than the exact pixel values. The level of granularity of the features is determined by the layer of the VGG network from which it is derived, with higher layers being more abstract and lower layer focusing on local features. The loss we are trying to minimize is the mean square loss for the content target and output target.

$$\mathcal{L}_{content} = \frac{1}{N_{l_c} M_{l_c}(x_c)}] \sum_{ij} \left(F_{l_c}(\hat{x}) - F_{l_c}(x_c)\right)^2{}_{ij}$$

Style Reconstruction Loss It is the weighted sum of the loss over the desired layers of the VGG. It is calculated using a Gram Matrix.

$$\mathcal{L}_{style} = \sum_l w_l E_l$$

$$E_l = \frac{1}{4N_l}^2 \sum_{ij} \left(G_l(\hat{x}) - G_l(x_s)\right)^2{}_{ij}$$

where $G_l = \frac{1}{M_l(x)} F_l(x)^T F_l(x)$ is the Gram Matrix of the feature maps in layer l in response to image x.

Total Variation Regularization To encourage spatial smoothness in the output image, total variation regularizer is used which penalizes for difference between output frame pixels.

$$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style} + \lambda \mathcal{L}_{TV}$$

4 Experiments

We implemented a Convolutional Neural Network as described in [1] in PyTorch. We ran a few images through the system and familiarized ourself with the model. We then implemented modified the model with the style and content and total variation losses as described in [2].We trained the model on a small dataset of 100 images from the COCO Dataset to fine tune the hyper parameters. Then we trained the model on the ful COCO image dataset. Training the model on the entire dataset for one style image takes about five hours as told in the paper. We trained our model with the 2014 Validation COCO dataset that contains around 40,000 images. The training time was around 5 hours on the NYU HPC cluster with the GPU enabled. We used a batch size of 4 images, which translates to 10,000 iterations for 40,000 images.

Findings
The weights used by the original paper did not work for us due to the differences in the frameworks. Finally we settled on [1e5,1e5,1e-7] as weights for the three losses. The output layer as suggested by the paper to use scaled tanh function did not work well. We replaced it with the direct output from the CNN. We also feel that downsampling by fractional strides is the main cause of the bordering effect observed in some images. The VGG networks expects input in the range[0,1] after normalization. The input image to the Image transformation network need not be normalized but its' output should be normalized. The final image to be stored in as numpy array must have type 'uint8'. Adding total variation loss helps to produce smoother images.

The following grid shows how the images evolve over epochs.

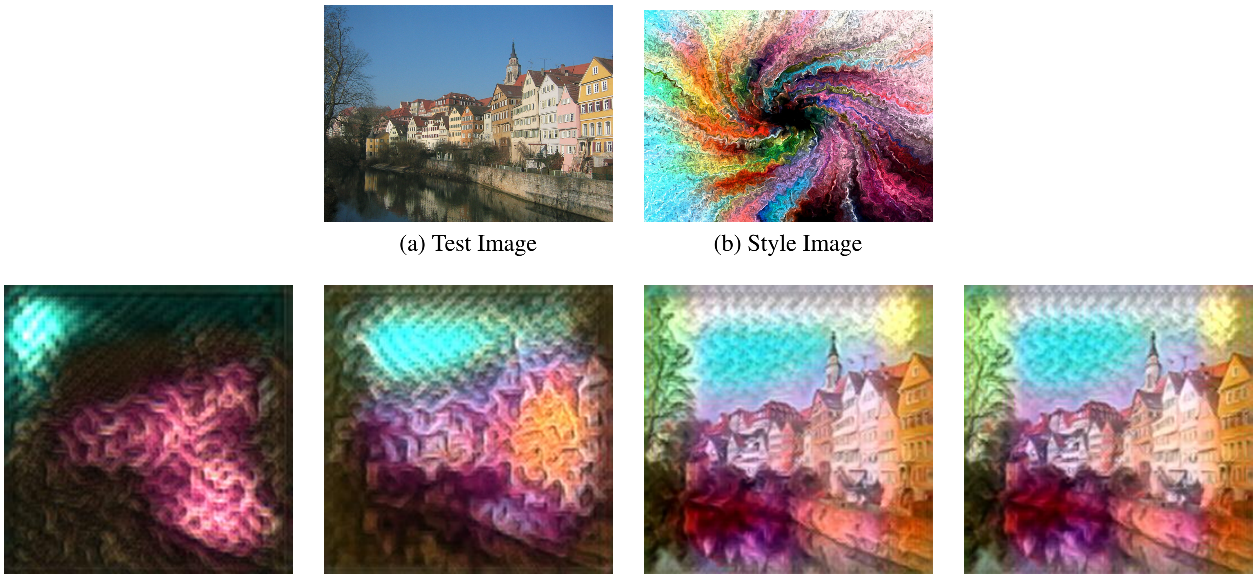
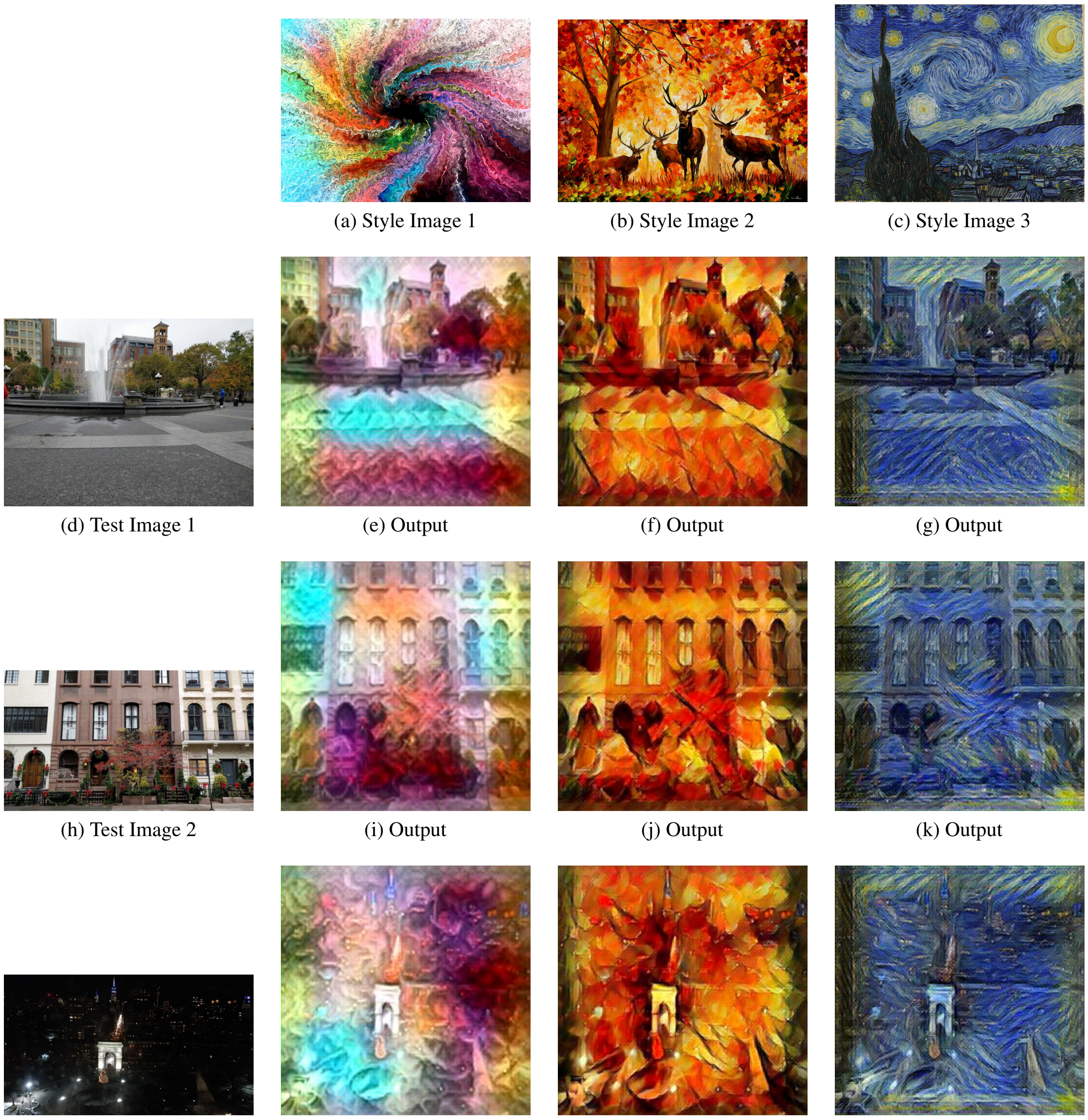

(a) Test Image
(b) Style Image
(c) 400 Iterations
(d) 1200 Iterations
(e) 8000 Iterations
(f) Final Output

Figure 3: Outputs after different epochs.

Figure 3 shows our results with the implementation of [2] .



(a) Style Image 1 (b) Style Image 2 (c) Style Image 3

(d) Test Image 1 (e) Output (f) Output (g) Output

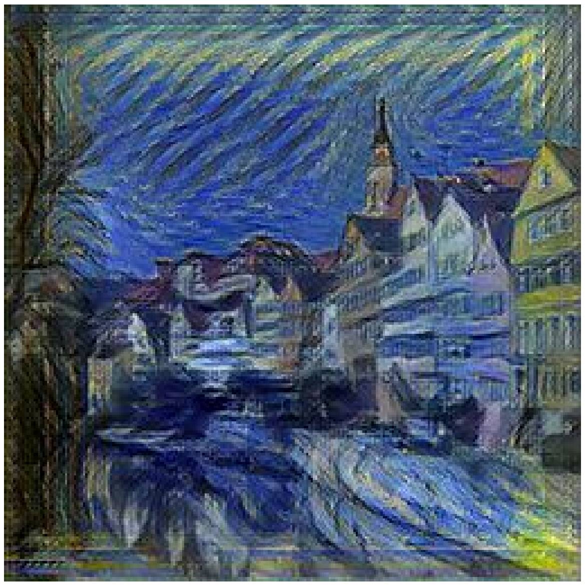
(h) Test Image 2 (i) Output (j) Output (k) Output

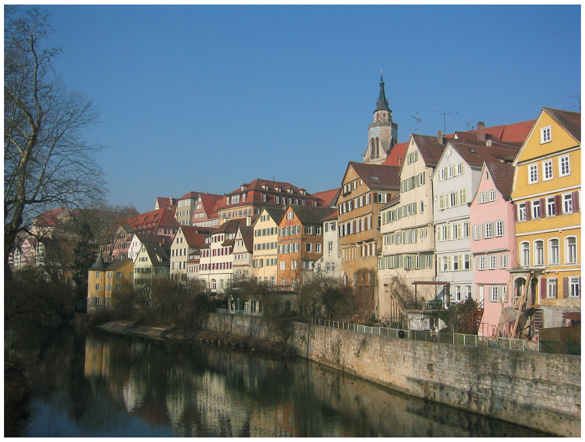
(l) Test Image 3 (m) Output (n) Output (o) Output

Figure 4: Style Transfer Examples

We also tried to implement color preservation as described in [3]. Figure 4 shows the results we got.

(a) Style Image


(b) Regular NST


(c) Test Image

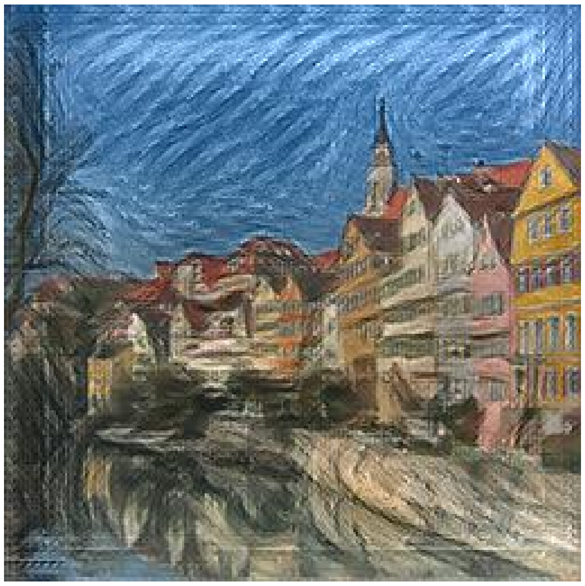
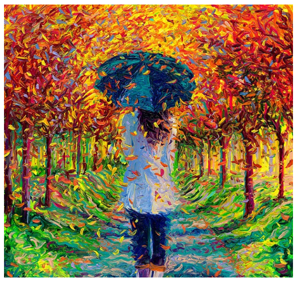
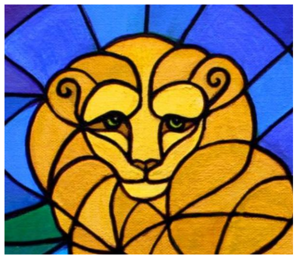

(d) Color Preserved

Figure 5: Color Preservation.

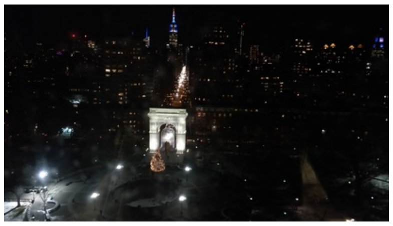
We thought it would also be interesting to use multiple style images at the same time to create a hybrid stylized image. Here are a few results:


(a) Style Image 1


(b) Style Image 2

(c) Original Image

(d) Original Image


(e) Original Image

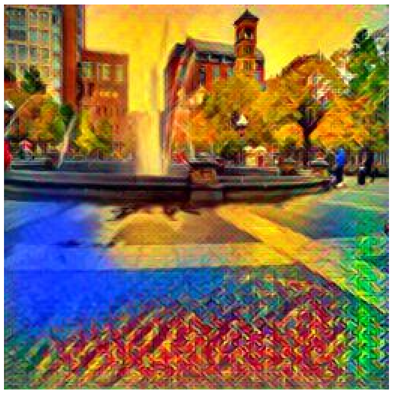
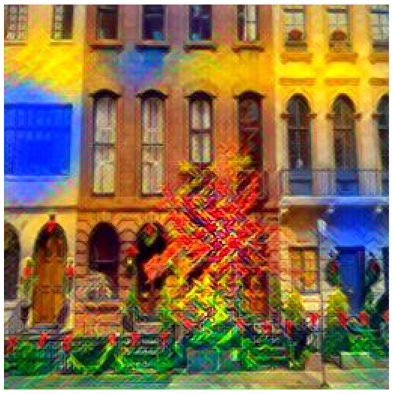
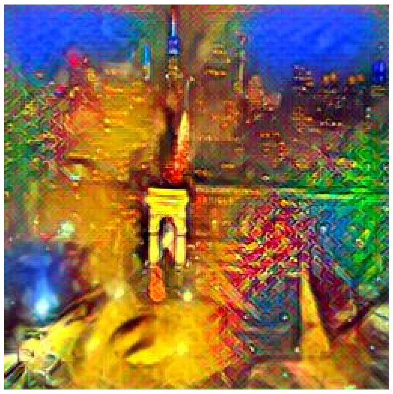




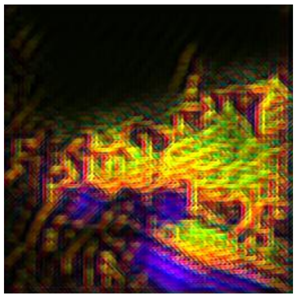
Figure 6: (a) and (b) are the style images used to train the model. The second row shows the stylized images produced by the model.

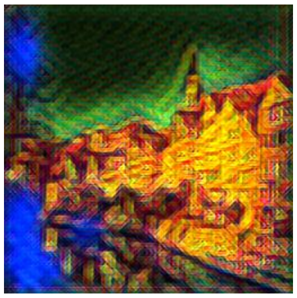
The following grid shows how the images evolve over epochs when simultaneously using multiple style images.

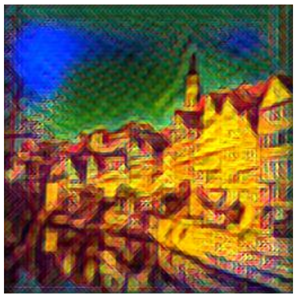
(a) Test Image

(b) Style Image 1

(c) Style Image 2


(d) 400 Iterations


(e) 800 Iterations


(f) 1200 Iterations

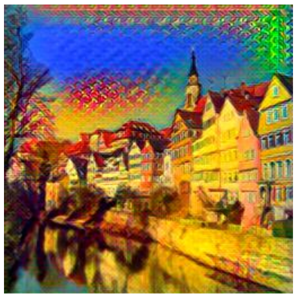

(g) Final Output

Figure 7: Outputs after different epochs when using multiple style images.

We also tried to apply **neural style transfer to videos**. The zip file of the submission contains some of the videos that we generated.

5 Conclusion

In this project, we looked at the different variants of Neural Style Transfer that have been published and implemented a few of those. We experimented with optimizers, learning rates, batch sizes, style and content weights, and added regularization. We found the best learning rate to be 0.001, and the best optimizer to be Adam. We kept a batch size of 4 images. The best values of α and β we obtained were each 10^5. The models performed did a satisfactory job with ordinary images, and we went on to extend the implementation to Videos. The models fared well when processing videos as well. However, we do see a little discoloration near the borders. We would like to continue our work further on ways to improve how our model handles borders.

References

[1] Leon A. Gatys Alexander S. Ecker Matthias Bethge: A Neural Algorithm of Artistic Style

[2] Justin Johnson, Alexandre Alahi, and Li Fei-Fei: Perceptual Losses for Real-Time Style Transfer and Super-Resolution.

[3] Leon A. Gatys Alexander S. Ecker Matthias Bethge Aaron Hertzmann Eli Shechtman:Controlling Perceptual Factors in Neural Style Transfer.

[4] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, Victor Lempitsky: Texture Networks: Feed-forward Synthesis of Textures and Stylized Images

[5] Manuel Ruder, Alexey Dosovitskiy, Thomas Brox: Artistic style transfer for videos

[6] Yongcheng Jing Yezhou Yang Zunlei Feng Jingwen Ye Mingli Song: Neural Style Transfer: A Review

[7] `http://cocodataset.org`

[8] Dmitry Ulyanov, Andrea Vedaldi: Instance Normalization: The Missing Ingredient for Fast Stylization