Style transfer models

Victor Kitov

v.v.kitov@yandex.ru

Yandex
School of Data Analysis.

Table of Contents

Neural style transfer

- Input: content image, style image.
- Style transfer - application of artistic style from style image to content image.
- Easy to do with convolutional neural networks.

Example

Applications

- **Enhance social communication**
  - adding personality, emotions
- **Extended drawing tools**
  - 2D drawings for painters, designers, architects.
- **Apply special effects** to movies, computer games (interactive style, depending on situation)
- **Cheap cartoon creation** from filmed scenes with actors.
- **Transfer learning**, e.g:
  - photos->paintings, learn to identify people on paintings
  - day photos->night photos, learn image segmentation at night

Table of Contents

Seminal work[1]

- Consider convolutional network (VGG).
- Content is reconstructed from activations at layer l_c.
- Style is reconstructed from correlations of its activations at layers $l \in L_s$

[1] Gatys et al (2015). A Neural Algorithm of Artistic Style.

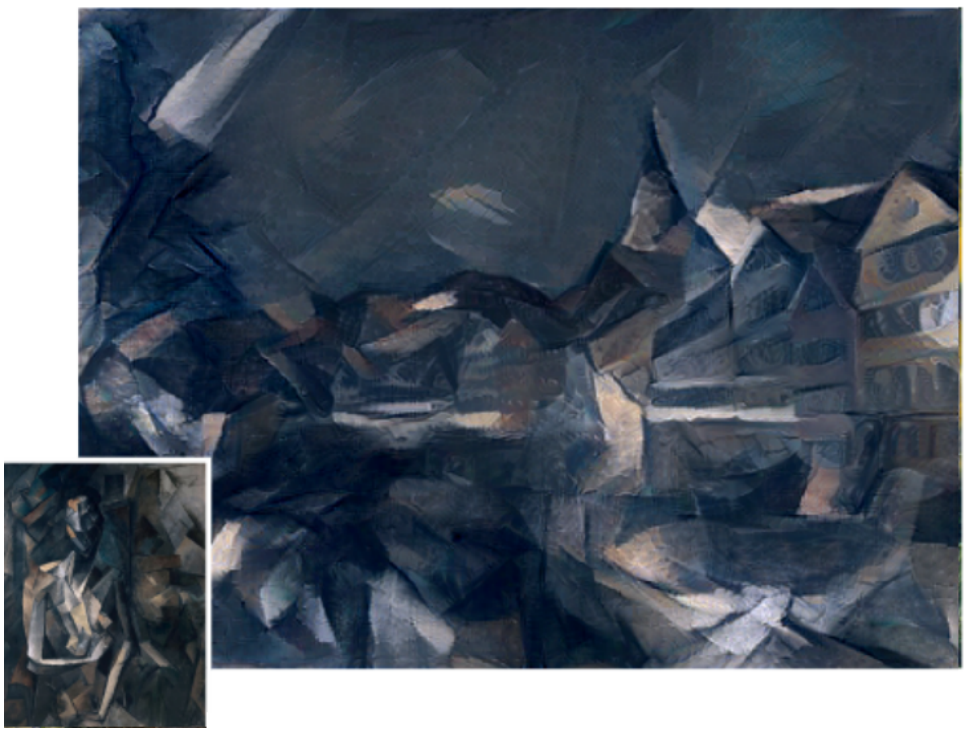
Content image

Style application

Style application

Style application

Style application



Notation

- Images:
  - x_c - content image;
  - x_s - style image
  - x - stylized image (to be found)
- Consider convolutional layer l:
  - feature map has $M_l = H_l \times W_l$ spatial dimensionality
  - N_l filters
- $\Phi_i^l(x) \in \mathbb{R}^{M_l}$ - spatially vectorized output of i-th filter on j-th position on layer l.
  - individual element is $\Phi_{ij}^l(x)$, $i = \overline{1, N_l}$, $j = \overline{1, M_l}$.
  - all feature maps: $\Phi^l(x) \in \mathbb{R}^{N_l \times M_l}$
- $\|A\|_F^2 = \sum_{i,j} a_{i,j}^2$ - squared Frobenius matrix norm.

Gram matrix

- Define Gram matrices$\in \mathbb{R}^{N_l \times N_l}$:

$$G_{ij}(x) = \frac{1}{M_l} \sum_{k=1}^{M_l} \Phi_{ik}(x)\Phi_{jk}(x) = \frac{1}{M_l} \Phi^l(x) \left(\Phi^l(x)\right)^T$$

- They capture uncentered correlations[2] between channels (feature maps)
  - spatial information is aggregated

[2] Centering gives same results but is more computationally expensive.

Losses

- Content loss (high level features should be similar):

$$\mathcal{L}_{content}(x, x_c) = \frac{1}{N_{l_c} M_{l_c}} \left\| \Phi^{l_c}(x) - \Phi^{l_c}(x_c) \right\|_F^2$$

- Style loss for one layer (feature correlations should be similar):

$$\mathcal{L}^l_{style} = \frac{1}{N_l^2} \left\| G^l(x) - G^l(x_s) \right\|_F^2$$

- Total style loss:

$$\mathcal{L}_{style}(x, x_s) = \sum_{l \in L_s} w_l \mathcal{L}^l_{style}$$

- l_c-content layer, L_s-layers for style reconstruction.

Losses

- Higher l=>higher content/style abstraction and dimensionality preserved.
- Content: only 1 deep layer l_c is used (versatile reconstruction of content object)
- Style: both shallow (low level pixel info) and deep layers (general style) are used
- x is found from

$$x = \arg\min_x \{\mathcal{L}_{content}(x, x_c) + \alpha \mathcal{L}_{style}(x, x_s)\} \quad (1)$$

- Hyperparameter $\alpha > 0$ measures relative importance of style, compared to content.

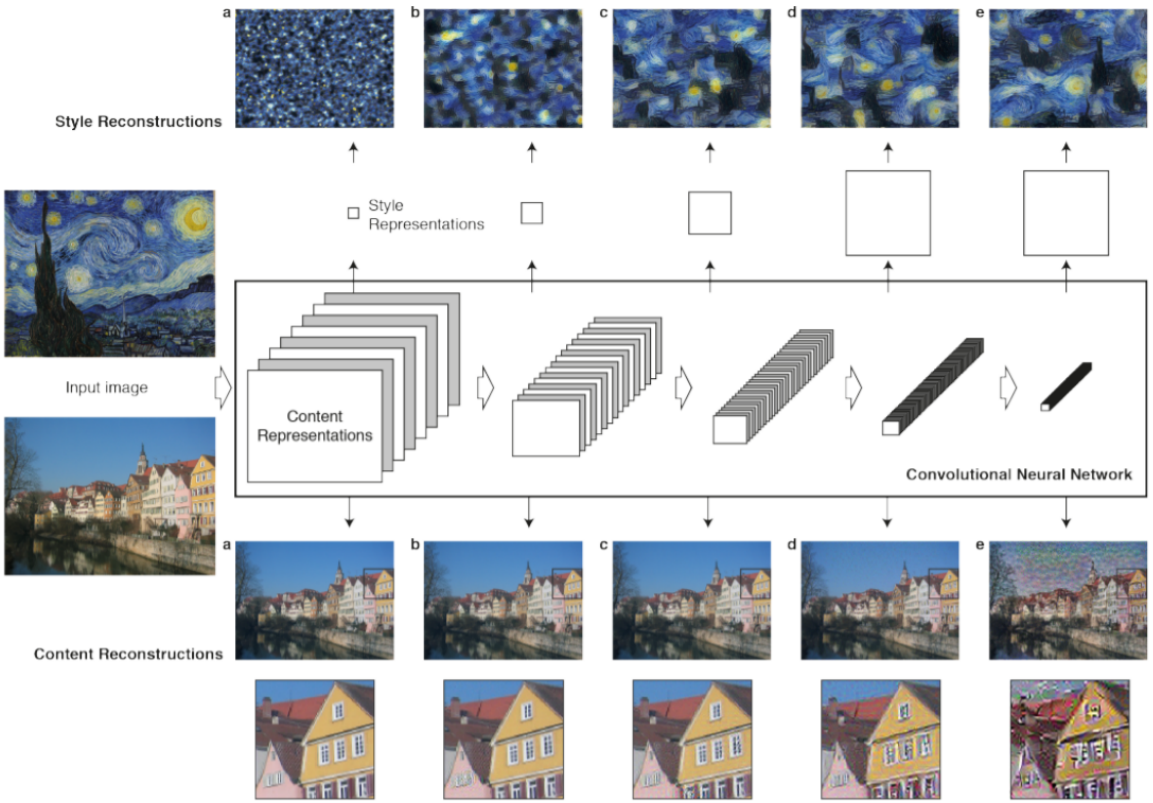
Detailed style transfer algorithm

INPUT: content image x_c, style image x_s.

1. Train classification CNN or take pretrained (e.g. VGG)
2. Using CNN, compute
   1. content image features $\Phi^{l_c}(x_c)$
   2. compute Gram matrices $\left\{G^l(x_s)\right\}_{l \in L_s}$
3. Initialize new image I (using content image or randomly)
4. Repeat until convergence:
   1. Pass I through CNN
   2. Compute loss (1)
   3. Backprop w.r.t I, holding CNN fixed
   4. Update I
5. Output I as stylization result

Image reconstruction from content/style information

Content (lower row) and style reconstruction (upper row) from layers of increasing depth:



Deeper layers contain more abstract information.

Increasing content importance

$\downarrow \alpha$ ($\Leftrightarrow \uparrow$importance of content preservation):

1

2

3

4

Total variation loss

- In ST we revers max-pooling=>solution underdetermined.
- Adding total variation loss makes it determined:

$$\mathcal{L}_{tv}(x) = \sum_{i,j} (x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2$$

- Total variation loss act as smoothness prior.
- Alternative less common definitions:

$$\sum_{i,j} |x_{i+1,j} - x_{i,j}| + |x_{i,j+1} - x_{i,j}|$$

$$\sum_{i,j} \left((x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2 \right)^{\gamma}, \quad \gamma > 0$$

Table of Contents

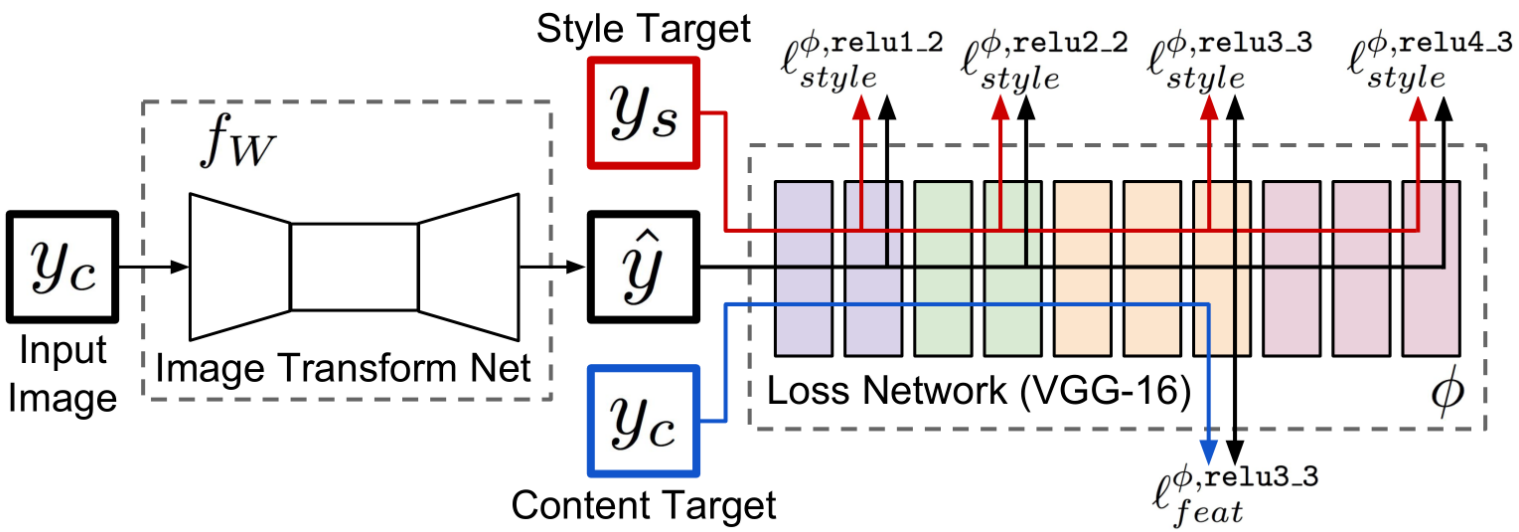
Fast style transfer[3]

- 2 nets:
  - image transform net (generator)
  - loss net (VGG-16)
- Loss net is held constant
  - it uses pretrained weights from image classification

[3] Link to paper and technical details.

Image transform net

- Image transform net is trained on
  - single style image
  - different content images
  - loss: offline ST loss (1)+total variation loss



Application

- To stylize a new image just pass it through the transformer net!
  - no optimization required
  - but need separate transformer net for each style
- Speedup of online ST compared to offline ST:

	Gatys *et al*				Speedup		
Image Size	100	300	500	Ours	100	300	500
256×256	3.17	9.52s	15.86s	**0.015s**	212x	636x	**1060x**
512×512	10.97	32.91s	54.85s	**0.05s**	205x	615x	**1026x**
1024×1024	42.89	128.66s	214.44s	**0.21s**	208x	625x	**1042x**

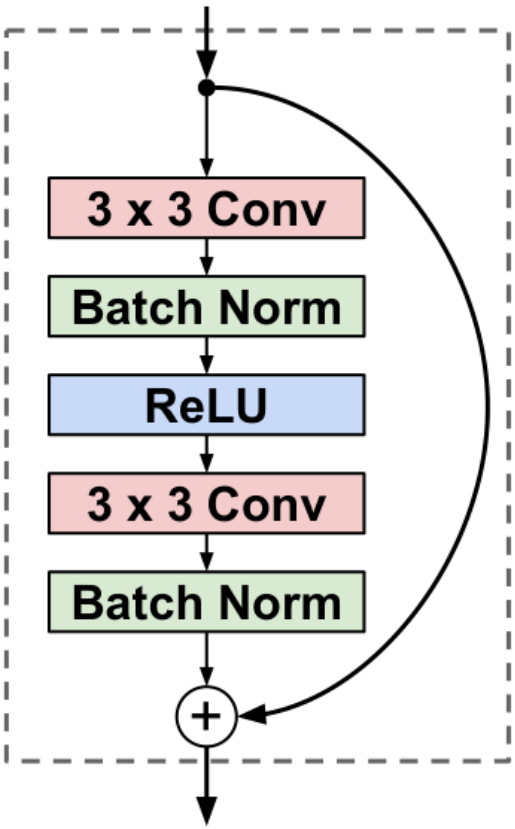
Architecture of generator

- No pooling layers used
- Downsampling: strided convolutions
- Residual blocks used in the middle
  - good at reproducing identity
  - reasonable: identity gives content image!
- Each convolution-followed by batch normalization
  - later articles: instance normalization better and probably better without normalization at all.
- Upsampling: fractionally strided convolutions
  - in later articles replaced by upsampling with NN & convolution - better

Architecture of transformet net

Layer	Activation size
Input	$3 \times 256 \times 256$
Reflection Padding (40×40)	$3 \times 336 \times 336$
$32 \times 9 \times 9$ conv, stride 1	$32 \times 336 \times 336$
$64 \times 3 \times 3$ conv, stride 2	$64 \times 168 \times 168$
$128 \times 3 \times 3$ conv, stride 2	$128 \times 84 \times 84$
Residual block, 128 filters	$128 \times 80 \times 80$
Residual block, 128 filters	$128 \times 76 \times 76$
Residual block, 128 filters	$128 \times 72 \times 72$
Residual block, 128 filters	$128 \times 68 \times 68$
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	$3 \times 256 \times 256$

Residual block



- No padding is used (causes artifacts at image border)
- To match size, identity is cropped at the end

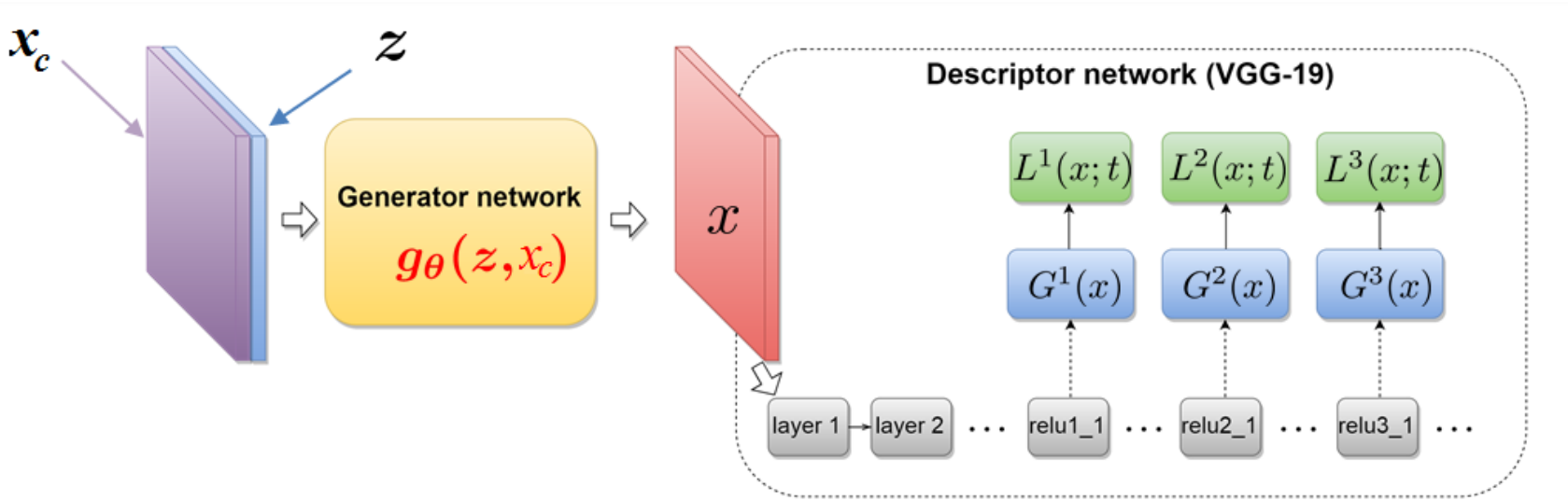
Table of Contents

Idea[4]

- Train generative network $g_\theta(x_c, z)$
  - x_c: content image
  - z: uniform random noise
  - θ: trained parameters (weights)
- By passing different random noise z we hope to generate many versions of stylized x_c.
- Use standard offline ST loss (1).

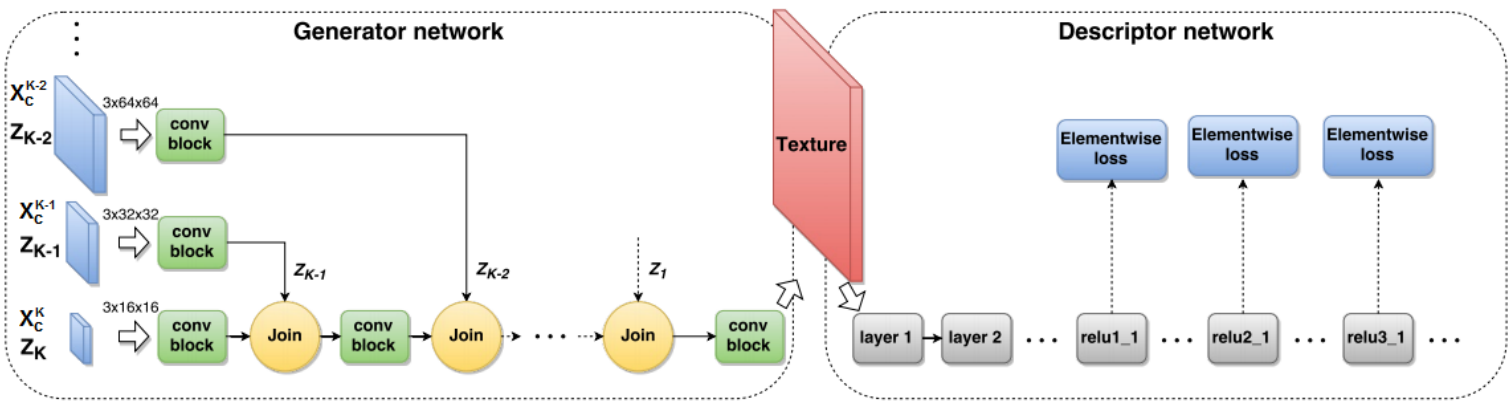
[4] Ulyanov et. al. (2016). Texture Networks: Feed-forward Synthesis of Textures and Stylized Images.

Proposed architecture



Architecture

- x_c^k - downsampled content image x_c by 2^{k-1}, $k = \overline{1, K}$.
- $z_1, z_2, z_3, \ldots$ - random noise at different resolutions (abstract levels).
- Only generator network is trained, descriptor network held fixed.
- In descriptor network (first layers of VGG) style transfer loss (1) is calculated.



Architecture in detail