Transformation-Based Style Transfer

Victor Kitov

v.v.kitov@yandex.ru

Table of Contents

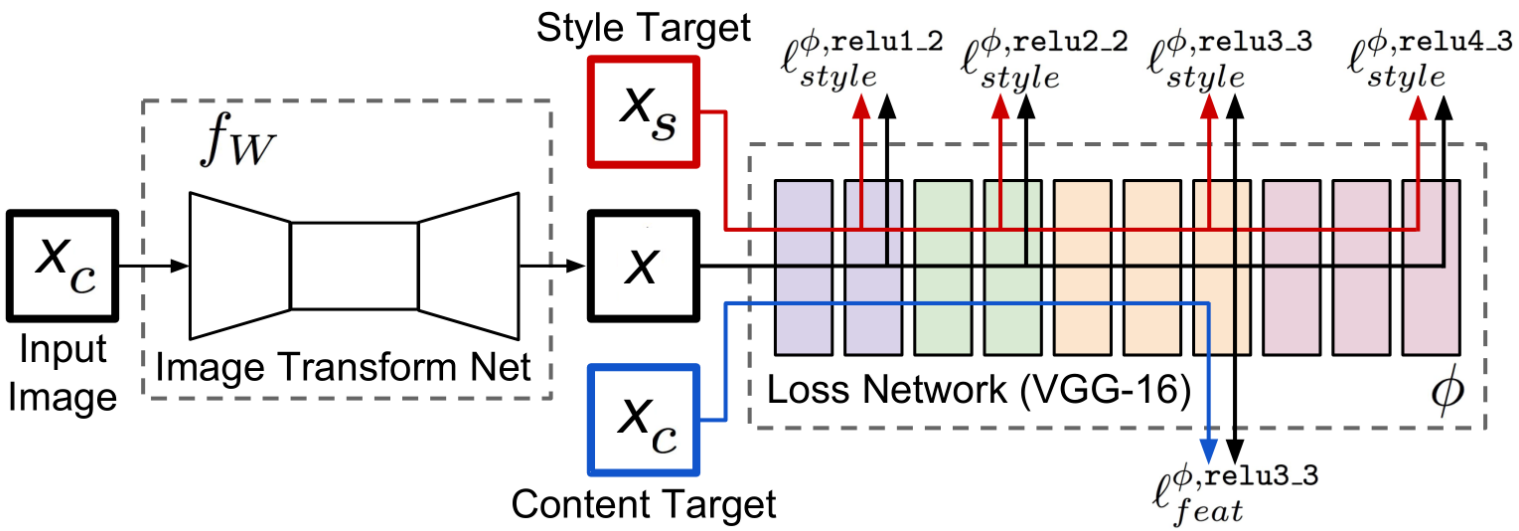
Fast style transfer[1]

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

[1] Link to paper and technical details.

Train

- Image transform net is trained on
  - single style image
  - different content images
  - loss=offline ST loss



Test

- To stylize a new image just pass it through the transformer!
  - real-time, 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

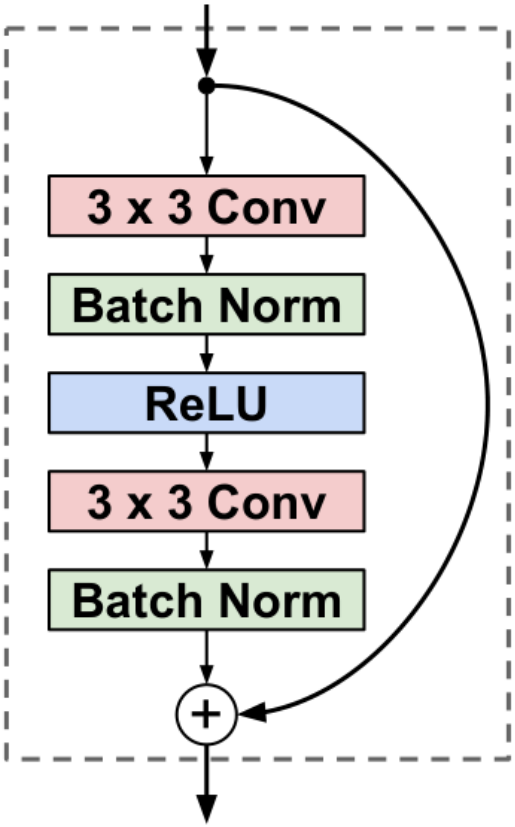
- No pooling layers used
- Downsampling: strided convolutions
- Residual blocks used in the middle
  - good at reproducing identity
  - reasonable: identity gives content image!
- Each convolution-followed by batch normalization
  - later articles: instance normalization better [2].
- Upsampling: fractionally strided convolutions
  - later articles: upscaling & convolution - better, no checkerboard artifacts

[2] May omitting normalization at all be even 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	$3 \times 256 \times 256$

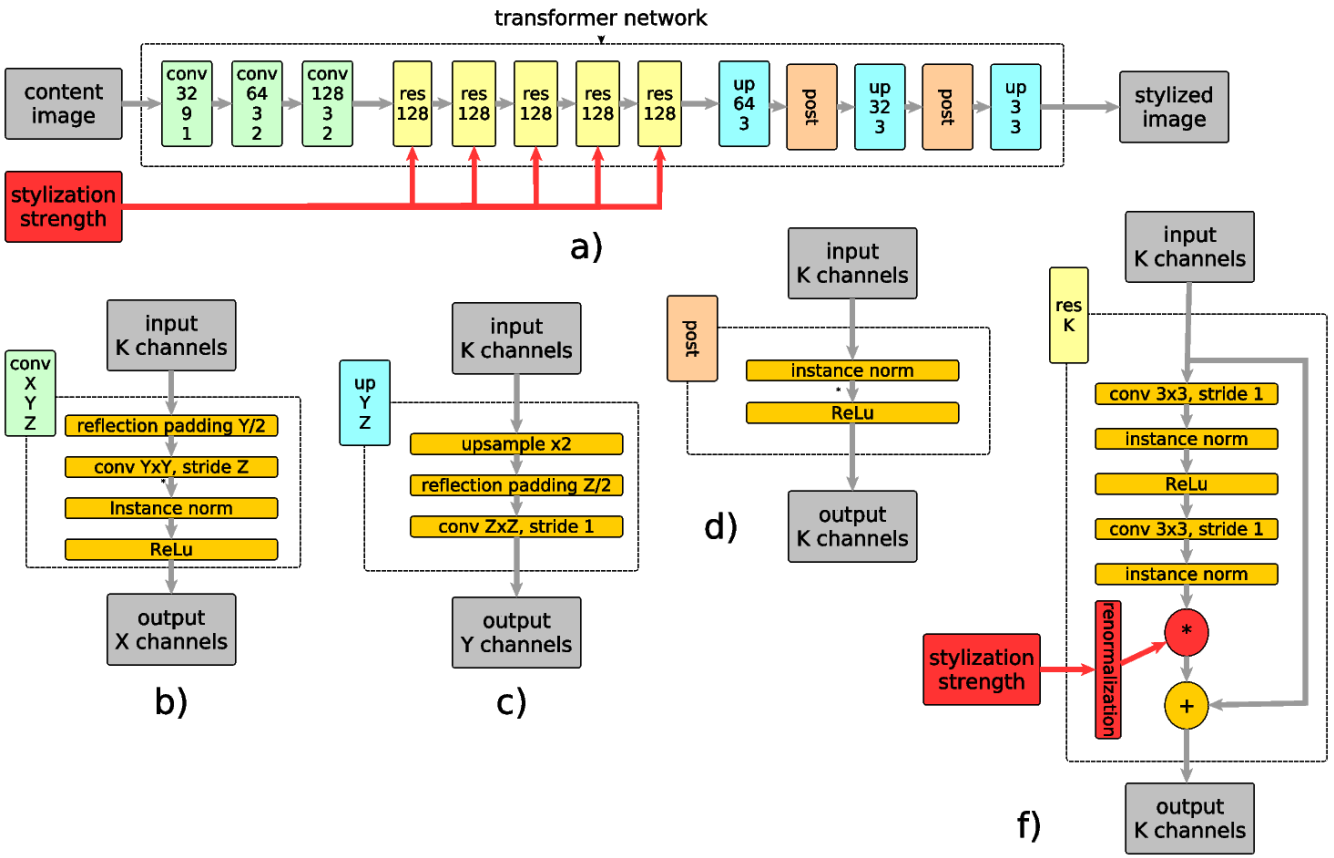
Residual block



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

Table of Contents

Real-time ST with strength control[3]



- Train transformer on (x_c, α) pairs, α - stylization strength, randomly sampled from grid.

[3] Kitov (2019) Real-Time Style Transfer With Strength Control.

Training with different α acts as regularization

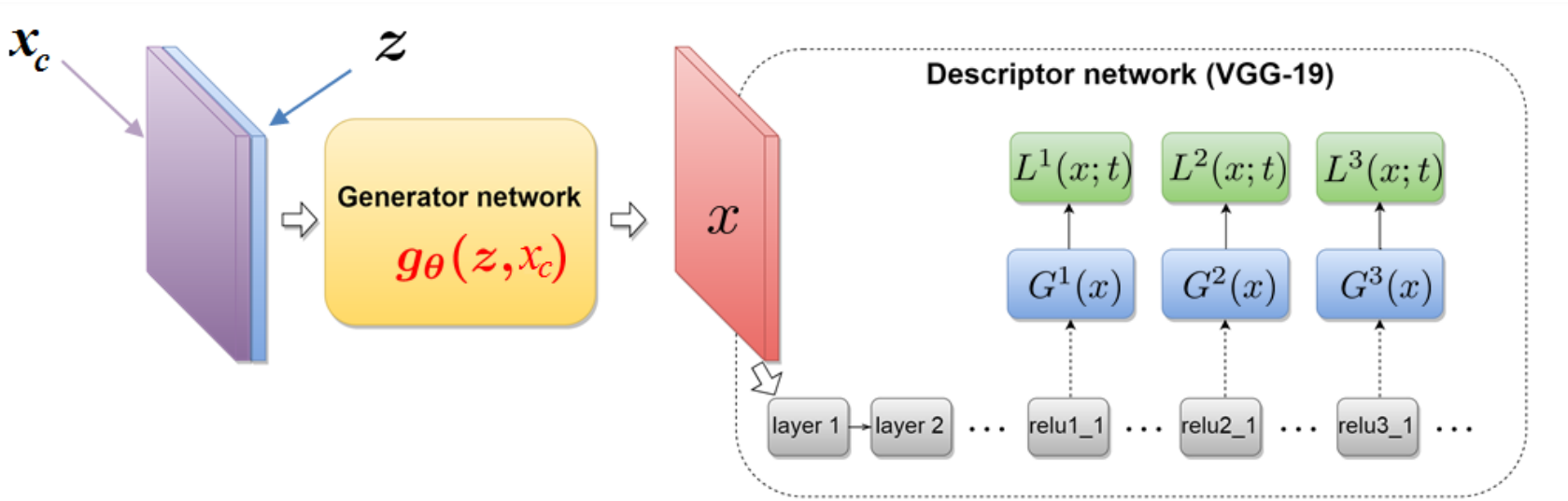
Table of Contents

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 ST loss (Gatys et al.)

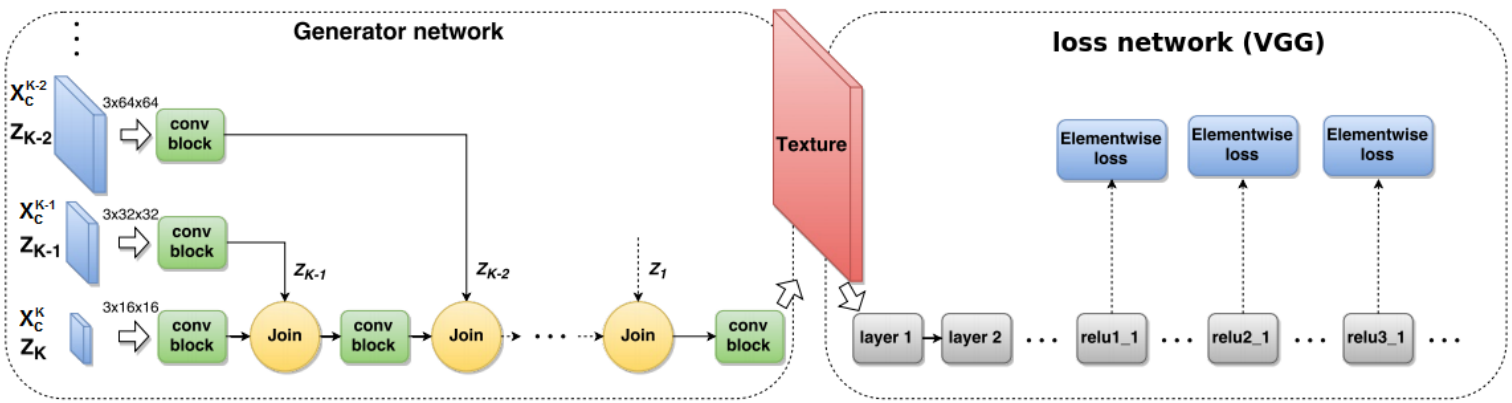
[4] Ulyanov et. al. (2016). Texture Networks: Feed-forward Synthesis of Textures and Stylized Images.

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).
- loss=standard offline ST loss
- Only generator network is trained, loss network (VGG) is fixed.



Architecture in detail

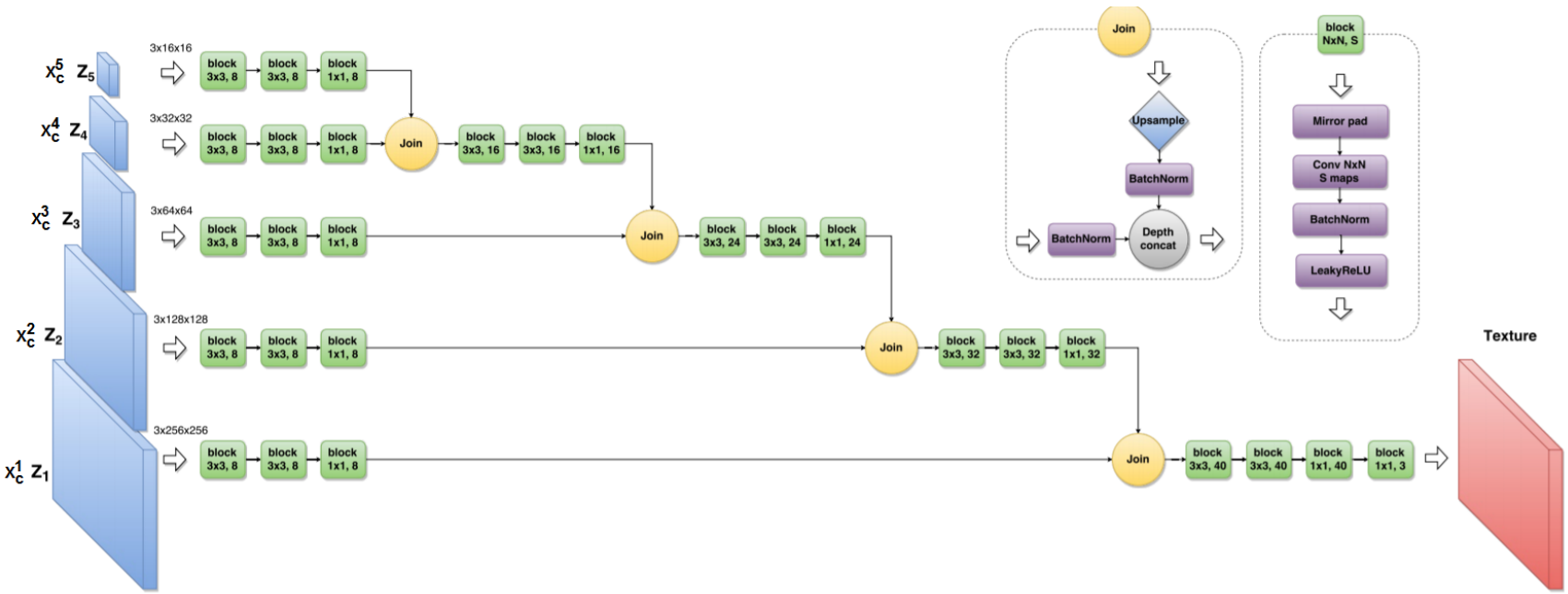


Table of Contents

Instance normalization[5]

- Instance normalization works better than batch normalization in online ST generators.
- Batch normalization: normalize spatial feature within minibatch of objects
- Instance normalization: normalize spatial feature within single object
- Normalizes distribution of features of the content image.
  - easier to map this distribution to given style

[5] 2017 - Instance normalization - The Missing Ingredient for Fast Stylization - Ulyanov.

Formula

- Let a_{icxy} be neuron activation for object n in the minibatch, channel c and coordinates x, y.
- Batch normalization (B-size of minibatch):

$$\tilde{a}_{ncxy} = \gamma \frac{a_{ncxy} - \mu_c}{\sqrt{\sigma_c^2 + \varepsilon}} + \beta$$

$$\mu_c = \frac{1}{BWH} \sum_{n=1}^{B} \sum_{x=1}^{W} \sum_{y=1}^{H} a_{ncxy}, \quad \sigma_c^2 = \frac{1}{BWH} \sum_{n=1}^{B} \sum_{x=1}^{W} \sum_{y=1}^{H} (a_{ncxy} - \mu_c)^2$$

- Instance normalization (proposed approach)

$$\tilde{a}_{ncxy} = \gamma \frac{a_{ncxy} - \mu_{nc}}{\sqrt{\sigma_{nc}^2 + \varepsilon}} + \beta$$

$$\mu_{nc} = \frac{1}{WH} \sum_{x=1}^{W} \sum_{y=1}^{H} a_{ncxy}, \quad \sigma_{nc}^2 = \frac{1}{WH} \sum_{x=1}^{W} \sum_{y=1}^{H} (a_{ncxy} - \mu_c)^2$$

Comments

- γ, β - learned parameters.

At test time:

- batch normalization: substitute train sample mean & std.deviation instead of μ_c, σ_c^2.
- instance normalization: recalculate μ_{nc}, σ_{nc}^2 for test object.

Results with instance normalization are better

Results for generator of Ulyanov (2016) (left) and Johnson (2016) (right). Batch normalization (upper row) and instance normalization (lower row).

Results are robust to contrast/brightness of content

(a) Content image.

(b) Stylized image.