Image Style Transfer, Neural Doodles & Texture Synthesis

Dmitry Ulyanov

BMMO seminar
Moscow, 2016

Yandex

Skoltech
Skolkovo Institute of Science and Technology

VGG-style networks

- Consist of repeated
  - Convolutions
  - ReLU
  - MaxPool

  \+
  - FC + Softmax at the end

- Activations (feature maps)
  - Tensor of size CxWxH

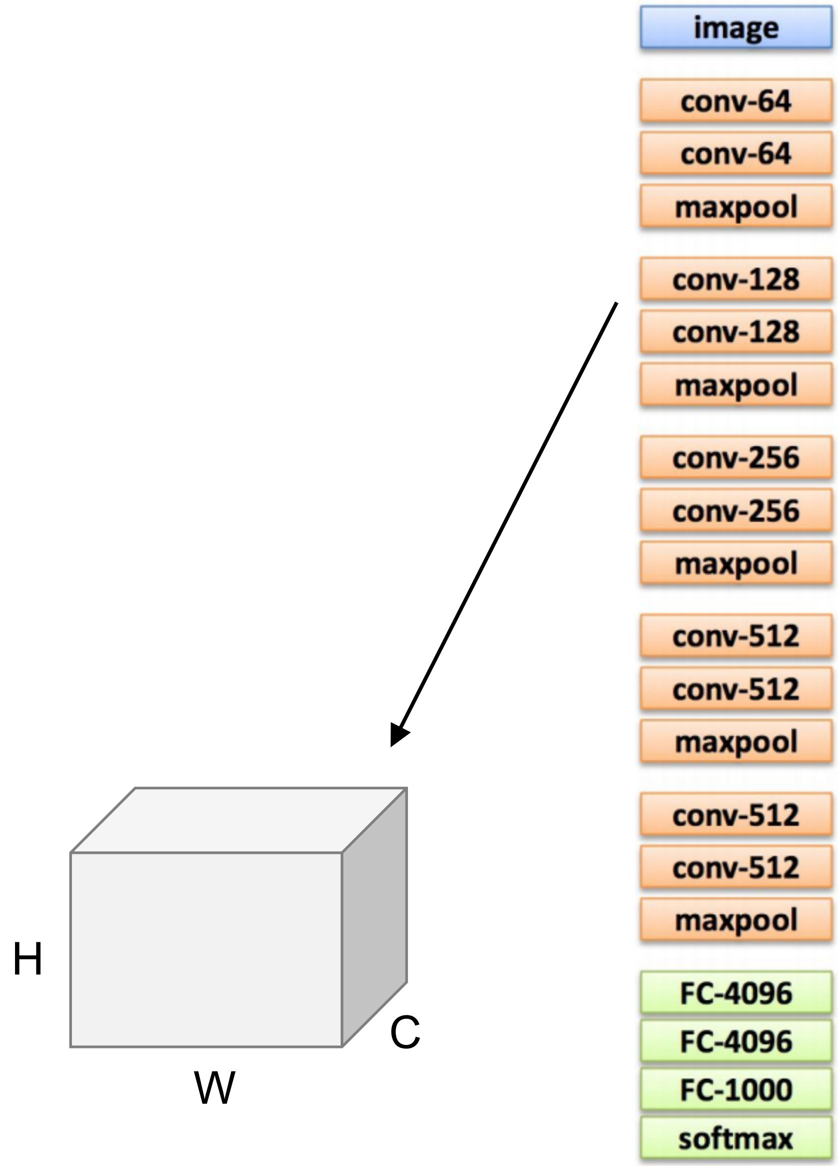


Image credit: [Xavier Giro](), DeepFix slides

Image generation examples

Mordvintsev, 2015

Simonyan et al. 2014

Presentation structure

- General overview:
    1. Texture synthesis
    2. Image style transfer
    3. Neural doodles

- Our work “Texture networks” (ICML 2016):
    - **Fast** texture synthesis
    - **Fast** image style transfer
    - **Fast** neural doodles

Examples: Texture Synthesis

Source **Synthesized**

L. A. Gatys, A. S. Ecker, M. Bethge; “Texture Synthesis Using Convolutional Neural Networks“; NIPS 2015

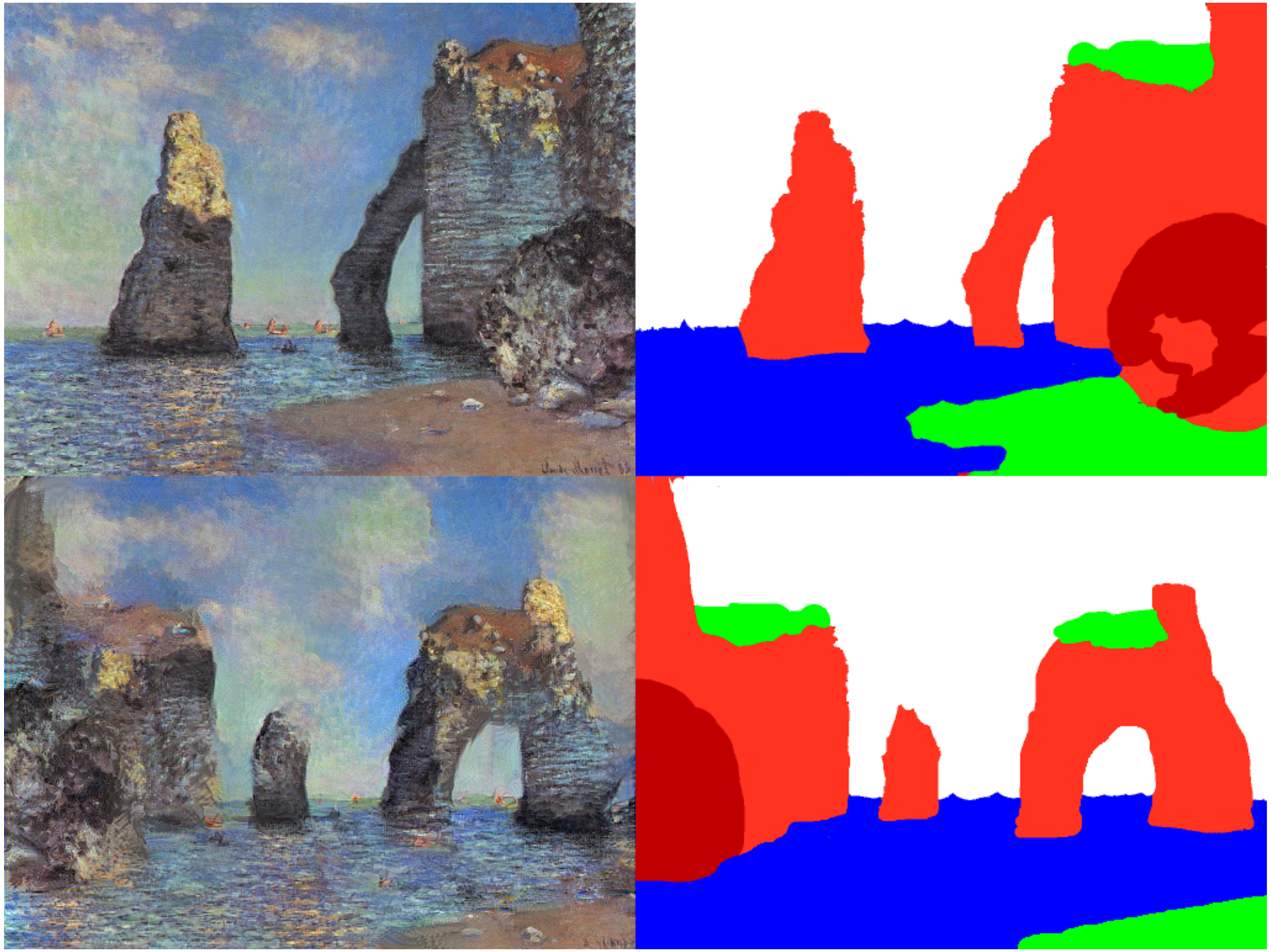
Examples: Image Artistic Style Transfer

Content **Style** **Result**

L. A. Gatys, A. S. Ecker, M. Bethge; "Image Style Transfer Using Convolutional Neural Networks"; CVPR 2016

Examples: Neural Doodles

Template **Mask**



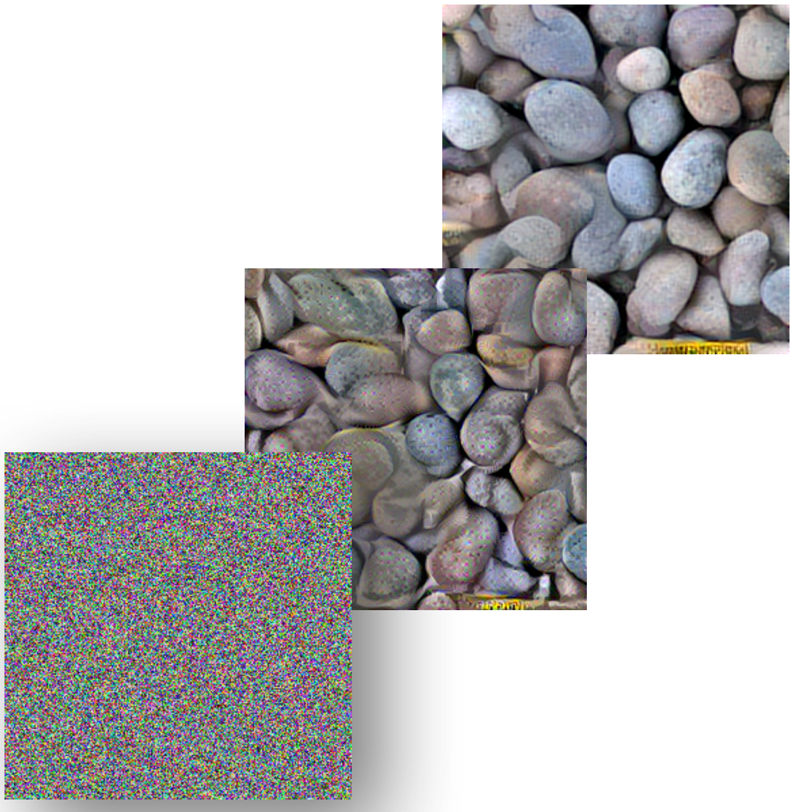
Synthesized image **User mask**

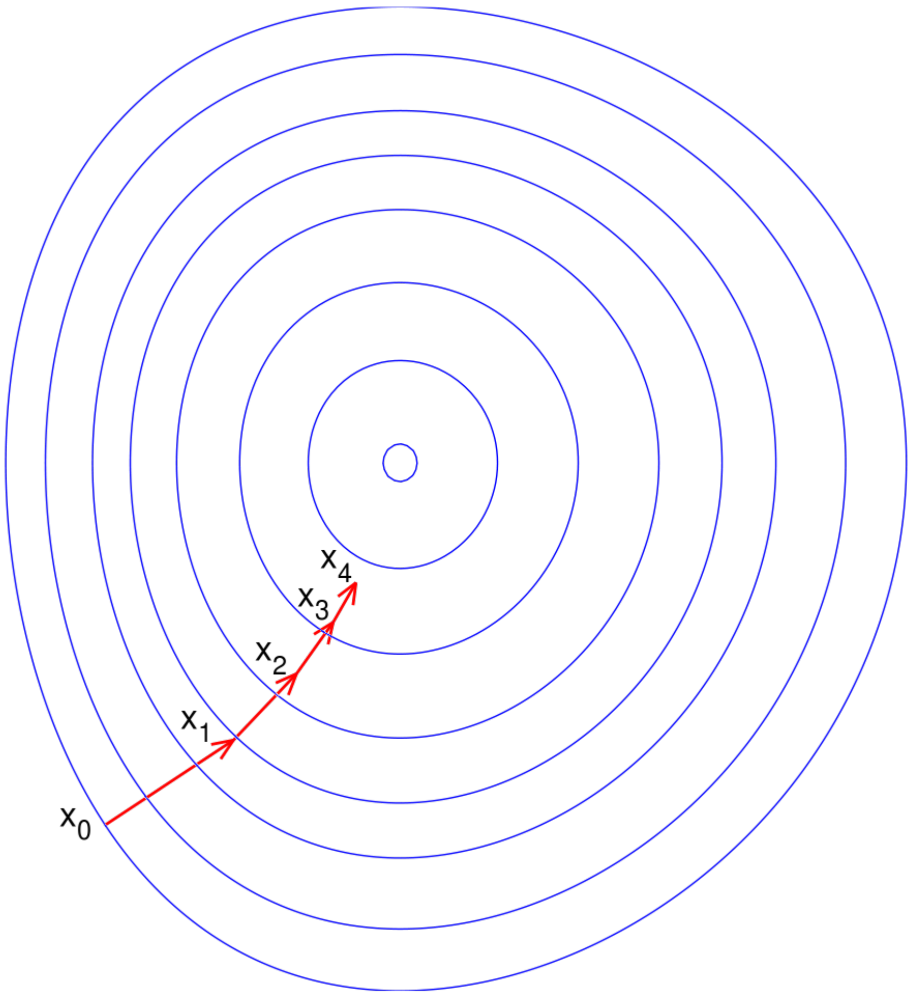
A. J. Champandard. “Semantic Style Transfer and Turning Two-Bit Doodles into Fine Artworks”, 2016

How does it work?

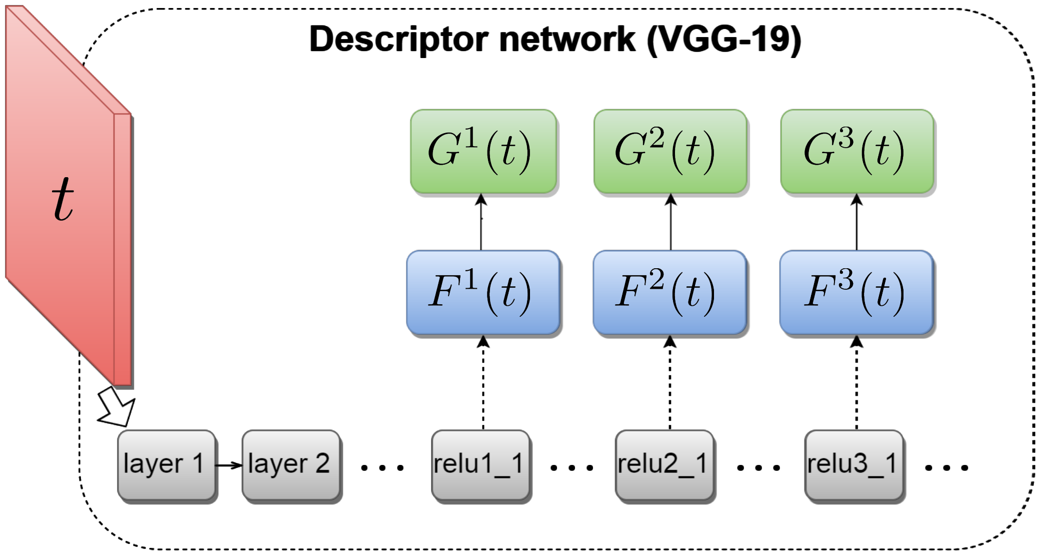
Image generation by optimization

$$x^* = \arg\min_x \mathcal{L}(x)$$





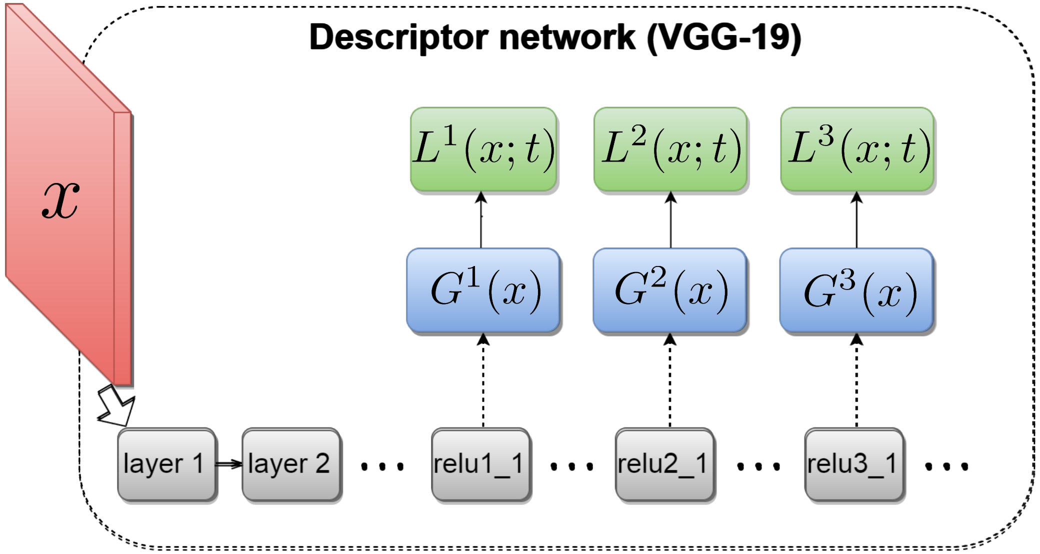
Gatys et. al.: Optimization-based texture synthesis



- Texture: t
- Activations at layer l: $F^l(t)$
- Gram matrix at layer l: $G^l(t)$

$$G^l_{ij}(t) = \sum_{k=1}^{M_l N_l} F^l_{ik}(t) F^l_{jk}(t)$$

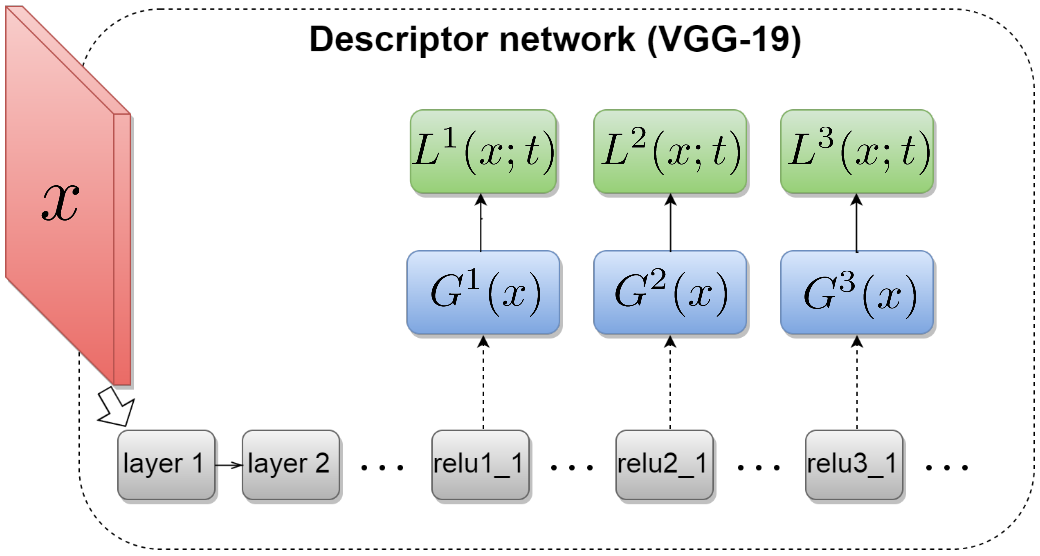
Gatys et. al.: Optimization-based texture synthesis



- Image: x
- Gram matrix at layer l: $G^l(x)$
- Loss at layer l: $L^l(x;t) = ||G^l(t) - G^l(x)||_2^2$

$$\mathcal{L}_{texture}(x;t) = \sum_l L^l(x;t)$$

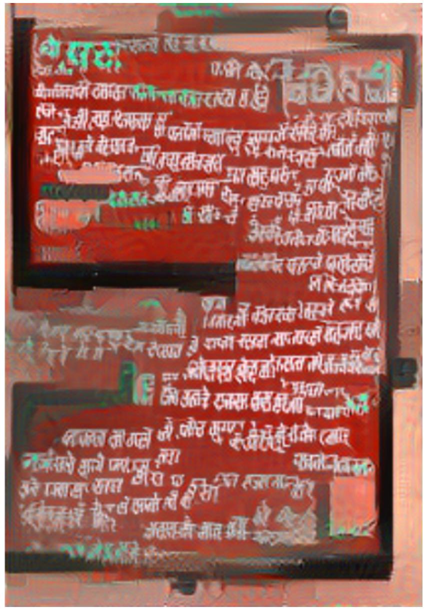
Gatys et. al.: Optimization-based texture synthesis



- Loss: $\mathcal{L}_{texture}(x;t) = \sum_l ||G^l(t) - G^l(x)||_2^2$
- Solve $\min_x \mathcal{L}_{texture}(x;t)$
- By gradient descent $x^{k+1} = x^k - \alpha \frac{\partial \mathcal{L}(x;t)}{\partial x}$

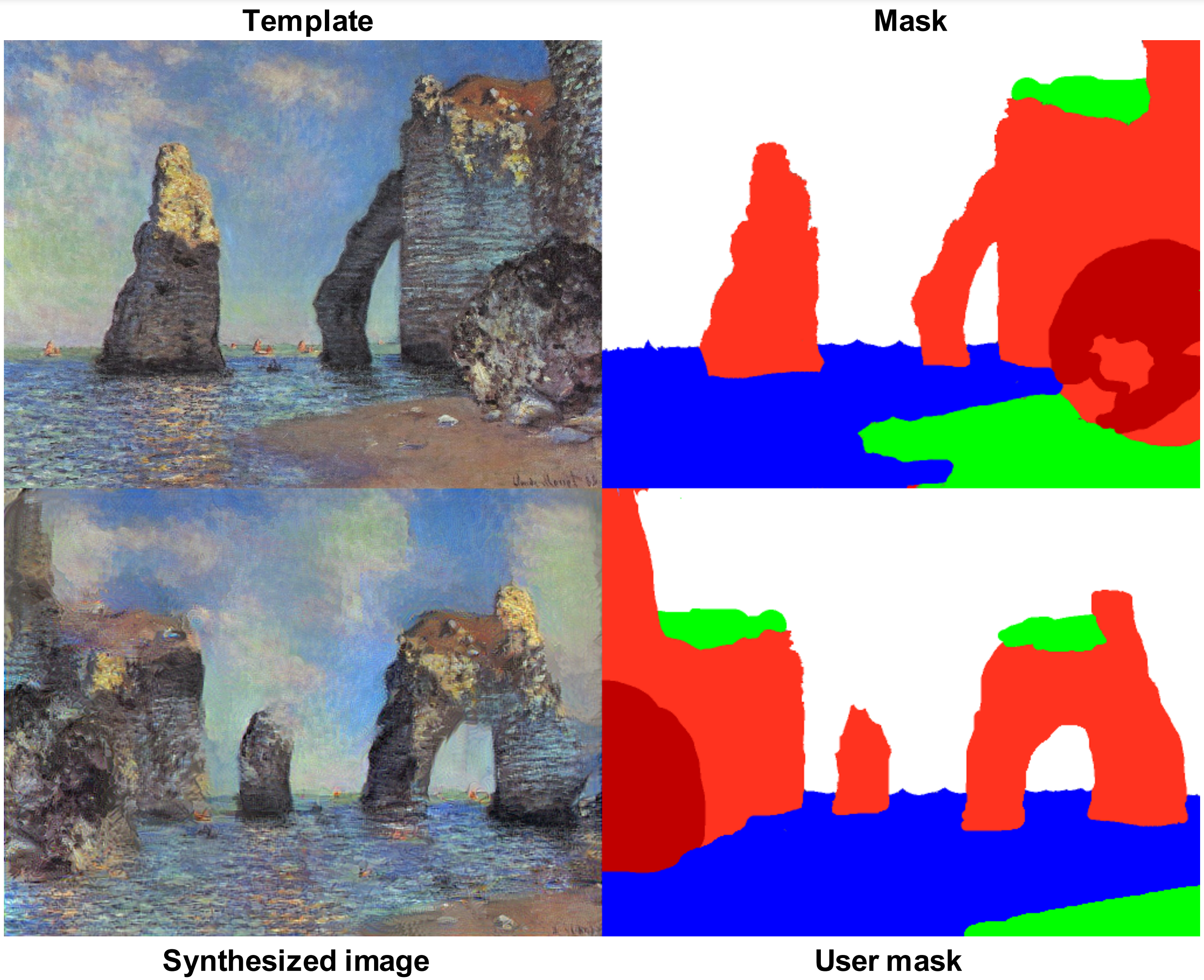
Examples: Texture Synthesis

Source | **Synthesized**



L. A. Gatys, A. S. Ecker, M. Bethge; "Texture Synthesis Using Convolutional Neural Networks"; NIPS 2015

How to: Neural Doodles



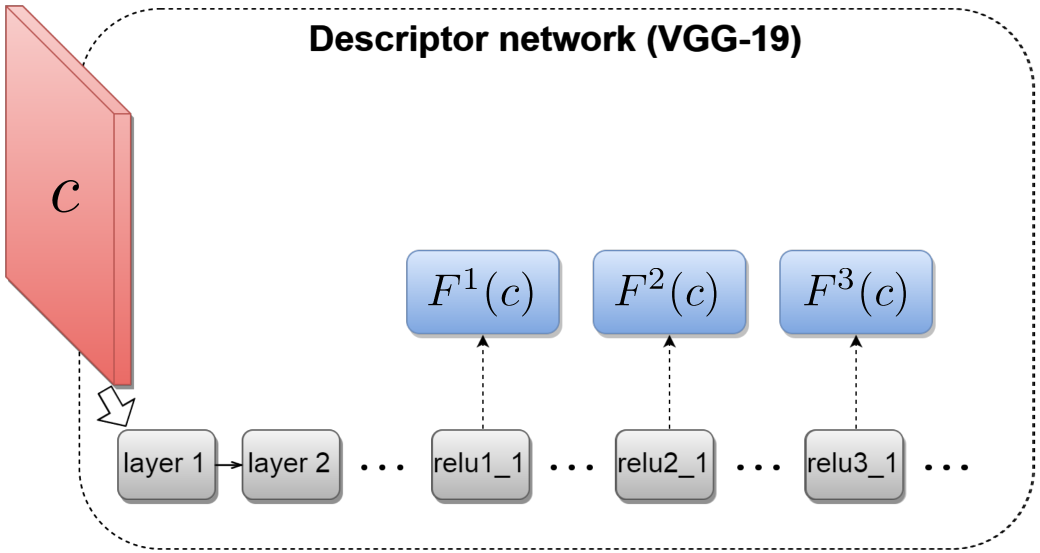
github.com/DmitryUlyanov/fast-neural-doodle

Gatys et. al.: Content loss for style transfer

Content c Style t Result x

- Total loss: $\mathcal{L}(x;t,c) = \mathcal{L}_{texture}(x;t) + \mathcal{L}_{content}(x;c)$
- Texture loss: $\mathcal{L}_{texture}(x;t) = \sum_l ||G^l(t) - G^l(x)||_2^2$
- Content loss: $\mathcal{L}_{content}(x;c) = \quad ?$

Gatys et. al.: Content loss for style transfer



- Content image: c
- Activations at layer l: $F^l(c)$

Gatys et. al.: Content loss for style transfer

Content c **Style** t **Result** x

- Total loss: $\mathcal{L}(x;t,c) = \mathcal{L}_{texture}(x;t) + \mathcal{L}_{content}(x;c)$
- Texture loss: $\mathcal{L}_{texture}(x;t) = \sum_l ||G^l(t) - G^l(x)||_2^2$
- Content loss: $\mathcal{L}_{content}(x;t) = \sum_l ||F^l(t) - F^l(x)||_2^2$

What else?

The results are excellent, but...

It is slow! Several minutes on a high-end GPU.

Texture Networks:

Feed-forward Synthesis of Textures and Stylized Images

Dmitry Ulyanov[1,2], Vadim Lebedev[1,2], Andrea Vedaldi[3], Victor Lempitsky[2]

ICML 2016

1 Yandex

2 Skoltech
Skolkovo Institute of Science and Technology

3 UNIVERSITY OF OXFORD

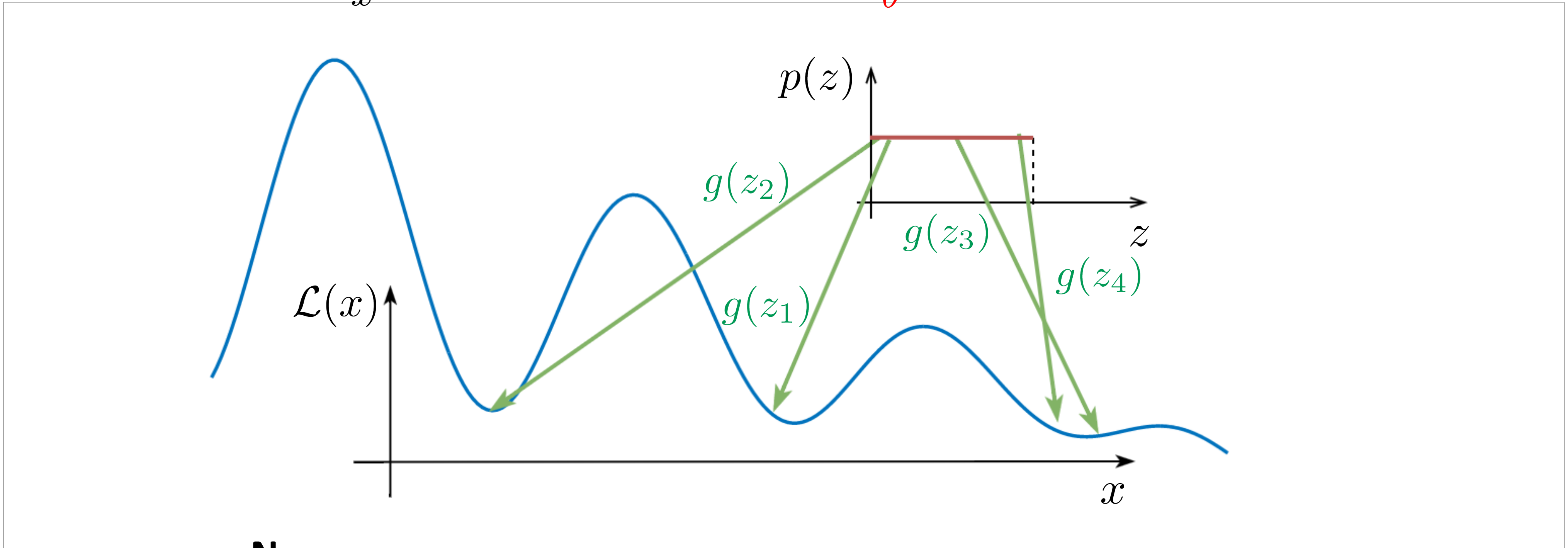
Our method: learn a neural net to generate

Instead of solving

$$\min_{x} \mathcal{L}(x)$$

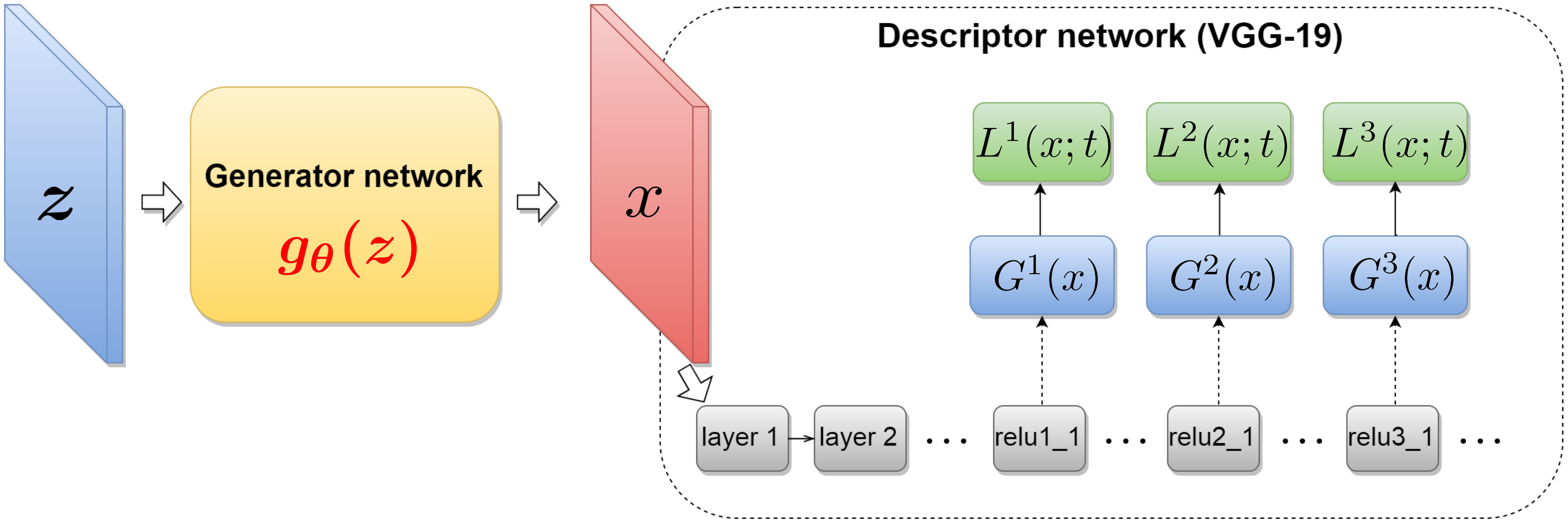
Solve

$$\min_{\theta} \mathbb{E}\mathcal{L}(g_\theta(\boldsymbol{z})) \quad \boldsymbol{z} \sim \mathrm{U}(0,1)$$



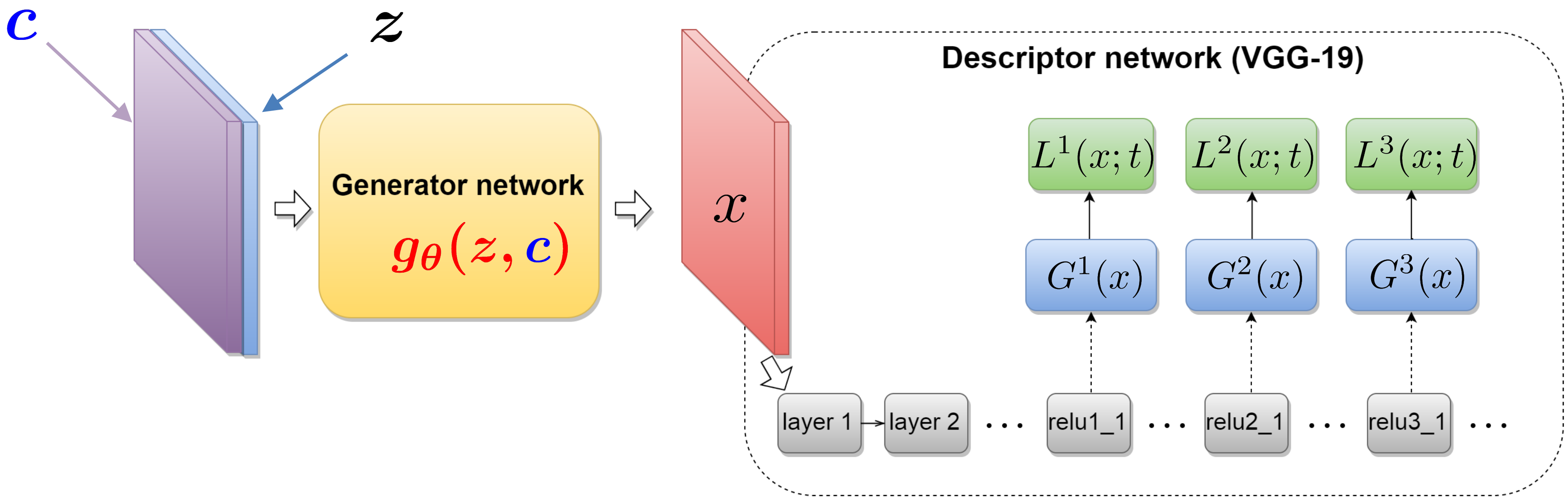
- **Now**
  - Generation requires *a single* $g_\theta(\boldsymbol{z})$ evaluation
- **But**
  - Need to make sure $g_\theta(\boldsymbol{z})$ does not collapse everything into one point

We propose: texture network



- Solve $$\min_{\theta} \mathbb{E}\mathcal{L}_{texture}(g_\theta(z); t), \quad z \sim U(0,1)$$
- By gradient descent $$\theta^{k+1} = \theta^k - \alpha \frac{\partial \mathcal{L}(g_\theta(z); t)}{\partial \theta}$$
- Generate x: $$x = g_\theta(z), \quad z \sim U(0,1)$$

We propose: stylization network



- Solve $\min_\theta \mathbb{E}\mathcal{L}(g_\theta(\boldsymbol{z}, \boldsymbol{c}); c, t), \quad \boldsymbol{z} \sim U(0,1)$
- By gradient descent $\theta^{k+1} = \theta^k - \alpha \frac{\partial \mathcal{L}(g_\theta(z))}{\partial \theta}$
- Generate x: $x = g_\theta(\boldsymbol{z}, \boldsymbol{c}), \quad \boldsymbol{z} \sim U(0,1)$

Qualitative evaluation: textures

Texture

Gatys et. al.
(90 sec.)

Ours
(0.06 sec.)

Almost similar but ours 500 times faster.

Qualitative evaluation: textures

Texture

Gatys et. al.
(90 sec.)

Ours
(0.06 sec.)

Qualitative evaluation: textures

Texture | Gatys et. al. (90 sec.) | Ours (0.06 sec.) | Texture | Gatys et. al. (90 sec.) | Ours (0.06 sec.)

Qualitative results: stylization

Content | Ours (0.06 sec.) | Gatys et. al. (90 sec.) | Style

Qualitative results: stylization

Content

Style

Ours

Gatys et. al.

Generator network

- Works good with any fully convolutional architectures.
- Use *Instance normalization* instead of Batch Normalization.

Cornell University Library

We gratefully acknowledge support from the Simons Foundation and member institutions

arXiv.org > cs > arXiv:1607.08022

Search or Article-id

(Help | Advanced search)

All papers Go!

Computer Science > Computer Vision and Pattern Recognition

Instance Normalization: The Missing Ingredient for Fast Stylization

Dmitry Ulyanov, Andrea Vedaldi, Victor Lempitsky

(Submitted on 27 Jul 2016)

It this paper we revisit the fast stylization method introduced in Ulyanov et. al. (2016). We show how a small change in the stylization architecture results in a significant qualitative improvement in the generated images. The change is limited to swapping batch normalization with instance normalization, and to apply the latter both at training and testing times. The resulting method can be used to train high-performance architectures for real-time image generation. The code will be made available at this https URL

Subjects: **Computer Vision and Pattern Recognition (cs.CV)**
Cite as: **arXiv:1607.08022 [cs.CV]**
(or **arXiv:1607.08022v1 [cs.CV]** for this version)

Submission history
From: Dmitry Ulyanov [view email]
[v1] Wed, 27 Jul 2016 10:23:00 GMT (4209kb,D)

Download:
- PDF
- Other formats

(license)

Current browse context:
cs.CV
< prev | next >
new | recent | 1607

Change to browse by:
cs

References & Citations
- NASA ADS

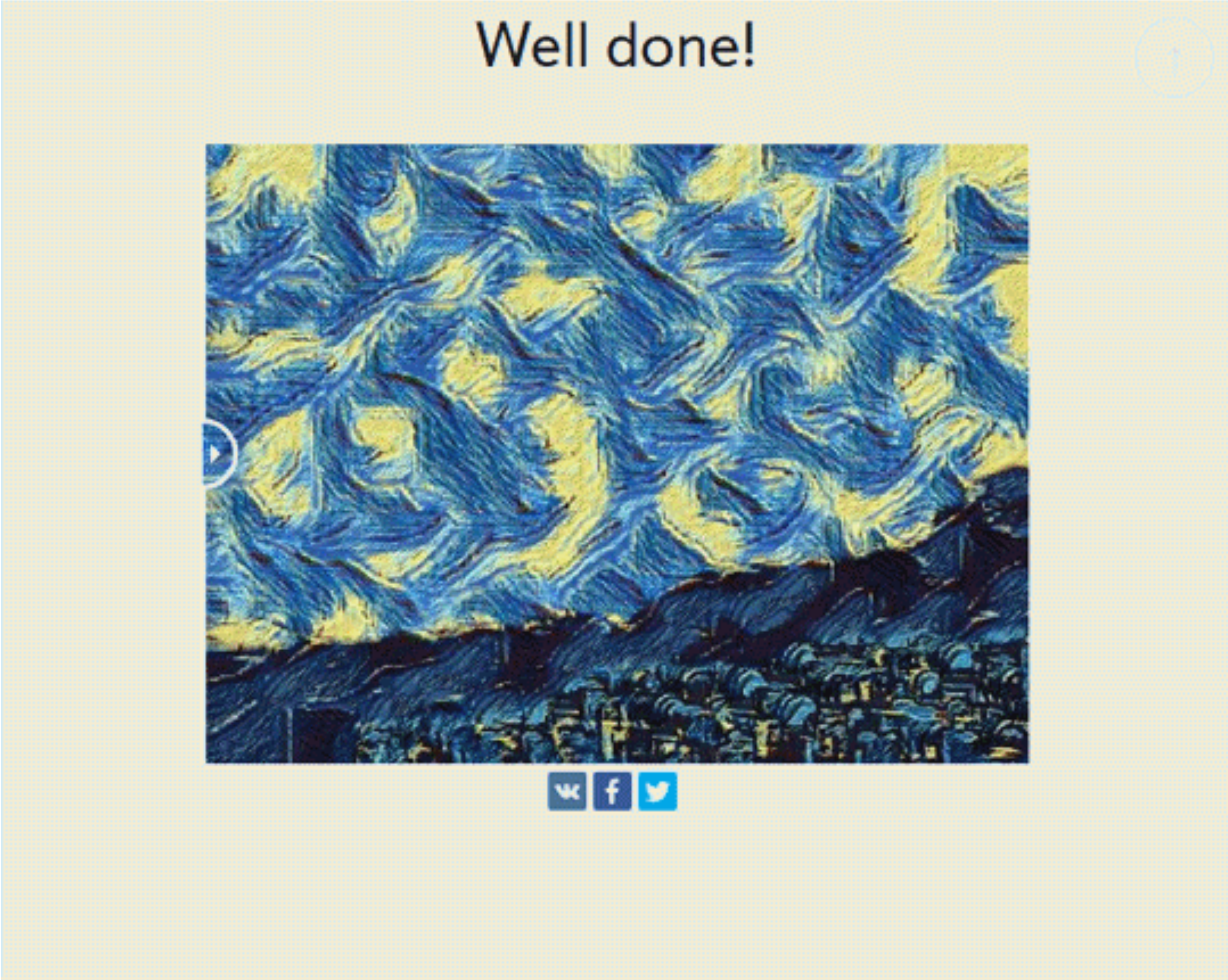
DBLP - CS Bibliography
listing | bibtex
Dmitry Ulyanov
Andrea Vedaldi
Victor S. Lempitsky

Bookmark (what is this?)

Was the technology used somewhere?

Yes!

Online neural doodles: *likemo.net*

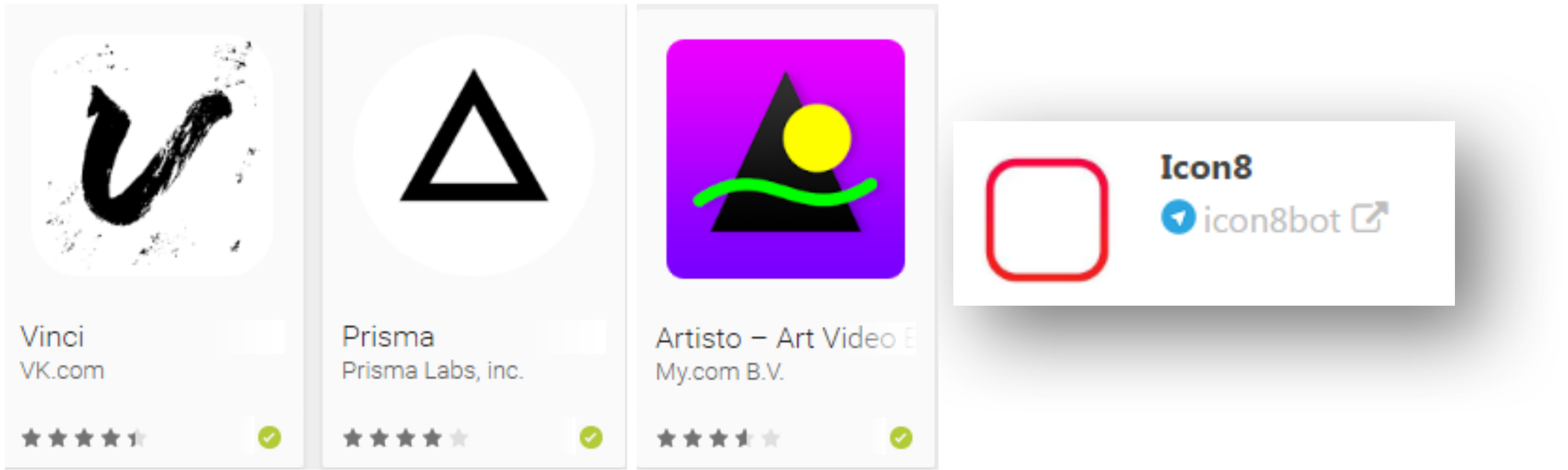


GIF: prostheticknowledge-online-neural-doodle

Code: github.com/DmitryUlyanov/online-neural-doodle

Fast stylization

- Made possible many stylization apps for mobile devices



Source code

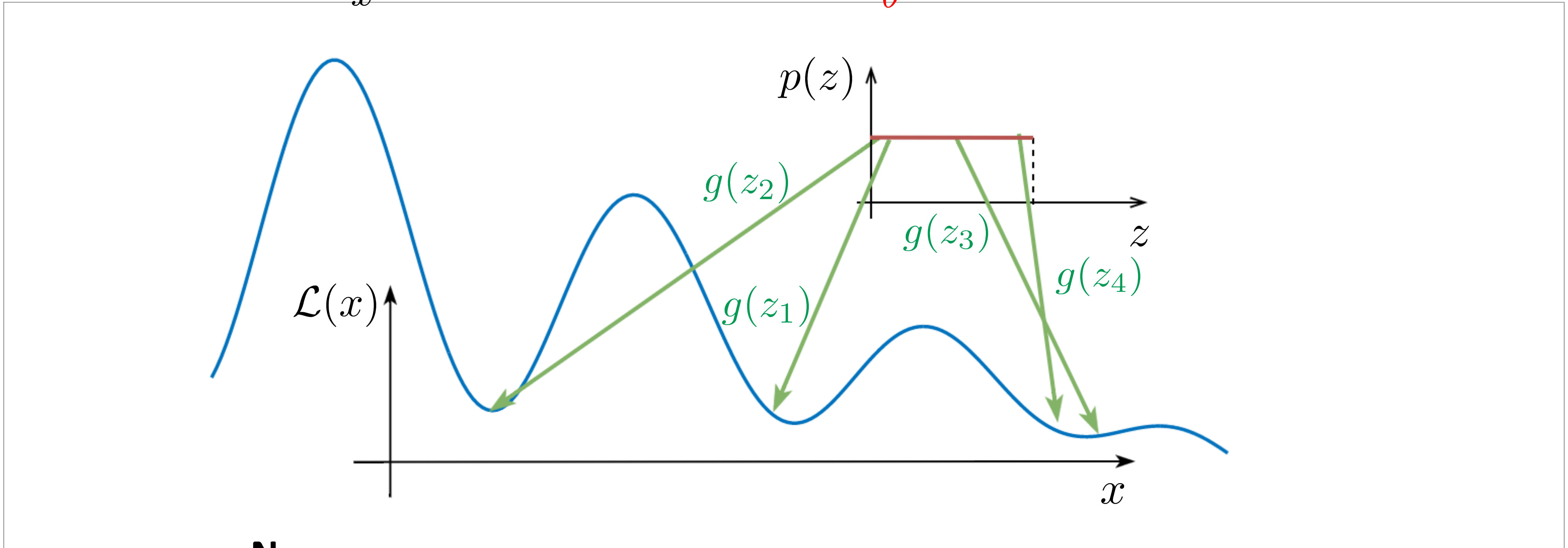
Source code is open at

https://github.com/DmitryUlyanov/

Our method: learn a neural net to generate

Instead of solving $\min_x \mathcal{L}(x)$

Solve $\min_\theta \mathbb{E}\mathcal{L}(g_\theta(\boldsymbol{z})) \quad \boldsymbol{z} \sim \mathrm{U}(0,1)$



- **Now**
  - Generation requires *a single* $g_\theta(\boldsymbol{z})$ evaluation
- **But**
  - Need to make sure $g_\theta(\boldsymbol{z})$ does not collapse everything into one point

Learning to sample

- Say our distribution $p(x)$ is known up to a normalizing constant:

$$\hat{p}(x) = e^{-L(x)}$$

$$Z = \int \hat{p}(x)dx$$

$$p(x) = \frac{\hat{p}(x)}{Z}$$

- **We want to learn to sample from distribution** $p(x)$.
- First, we approximate $p(x)$ with a distribution $q(x)$ from which we have a convenient way to sample.
- Note, that we will not define $q(x)$ explicitly, instead we say we have a sampler $z \sim q(x)$, $z = g_\theta(\epsilon)$, $\epsilon \sim N(0, 1)$.

Learning to sample

- Minimize $KL(q||p)$:

$$min_q KL(q||p)$$

- Decompose it first:

$$KL(q||p) = \int_x q(x) \ln \frac{q(x)}{p(x)} dx = \int_x q(x) \ln \frac{q(x)Z}{\hat{p}(x)} dx =$$

$$= \int_x q(x) \ln q(x) dx + \int_x q(x) \ln Z dx - \int_x q(x) \ln \hat{p}(x) dx$$

$$= \mathbb{E}_{x \sim q} \ln q(x) + \mathbb{E}_{x \sim q} L(x) + \ln(Z)$$

Learning to sample

- Decompose it first:

$$KL(q||p) = \mathbb{E}_{x\sim q} \ln q(x) + \mathbb{E}_{x\sim q} L(x) + \ln(Z)$$

- Second term estimator (texure nets!)

$$\mathbb{E}_{x\sim q} L(x) \approx \sum_{i=1}^{N} L(g_\theta(\epsilon_i)), \quad \epsilon \sim N(0,1)$$

- A Monte Carlo estimation for entropy is based on nearest neighbours.

$$\mathbb{E}_{x\sim q} \ln q(x) \approx \frac{D}{N} \sum_{i=1}^{N} \ln \rho_i + const(X)$$

where D is samples dimensionality, $\rho_i = min_{j\neq i}||X_i - X_j||$, $X_i = g_\theta(\epsilon_i)$

- Minimize $KL(q||p)$:

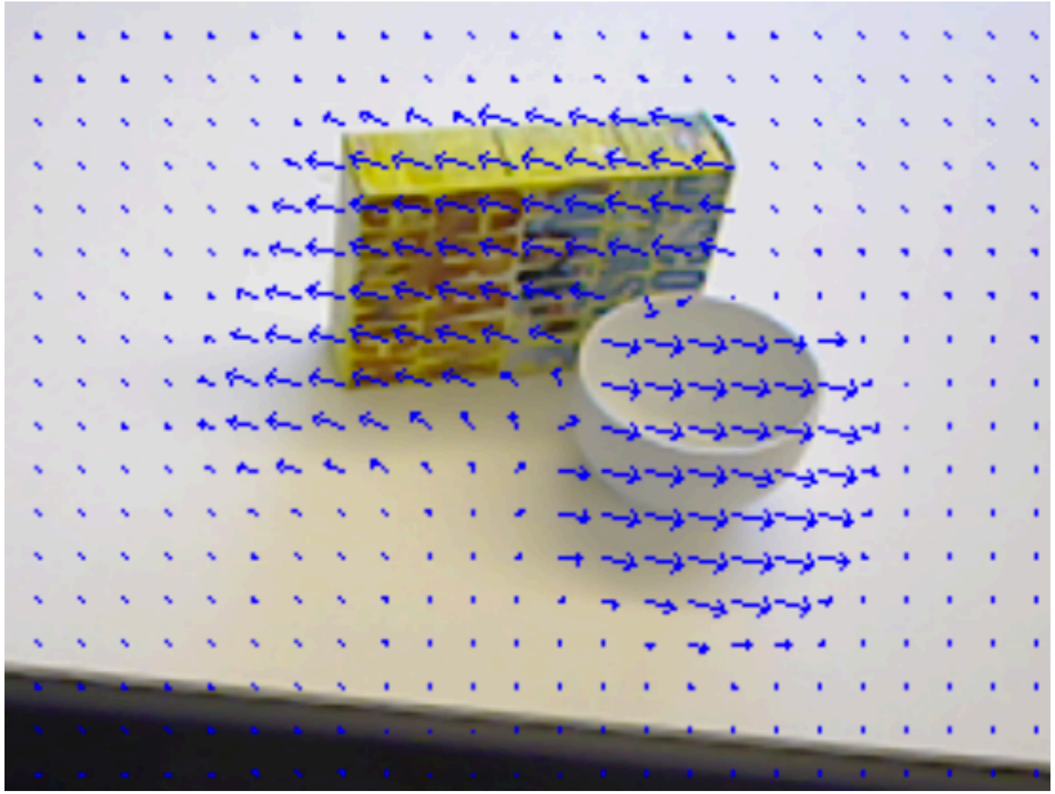
$$min_\theta \left[\frac{D}{N} \sum_{i=1}^{N} \ln \rho_i + \sum_{i=1}^{N} L(g_\theta(\epsilon_i)) \right]$$

Video stylization

- Process each frame independently

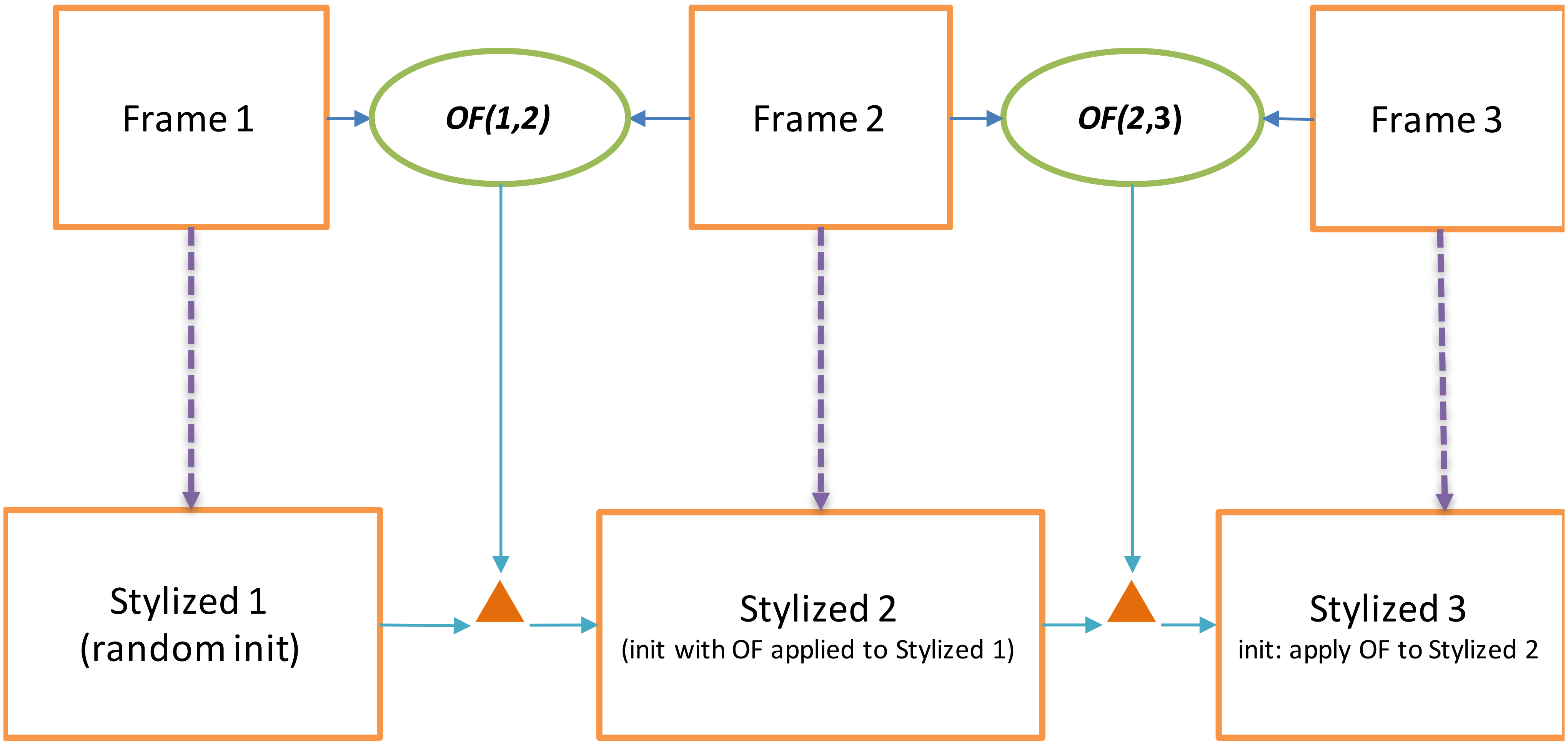
Video stylization

- Use optical flow (OF)



Video stylization

- Use optical flow (OF)



Video stylization

- Use optical flow (OF)

Video stylization

- Use optical flow (OF)

DeepWarp

- Ganin, Kononenko, Sungatullina, Lempitsky, ECCV 2016

DeepWarp

- Ganin, Kononenko, Sungatullina, Lempitsky, ECCV 2016

The last slide

Thank you!

Related work

Feed-forward generator

- **Generative Adversarial Networks** *(Goodfellow et. al., NIPS 2014)*: a neural network aims to produce samples that are indistinguishable from real examples

Similar concurrent work

- **Perceptual Losses for Real-Time Style Transfer and Super-Resolution**, *(Johnson et. al., ECCV 2016)*: very similar approach fast stylization approach.

- **Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks** *(Li & Wand, ECCV 2016)*: similar patch-based style transfer acceleration approach.