Lecture 7: Feed-forward generative models

Recap

- Previous lectures: image classification and related tasks (*networks that take images and output high-level semantic information*)

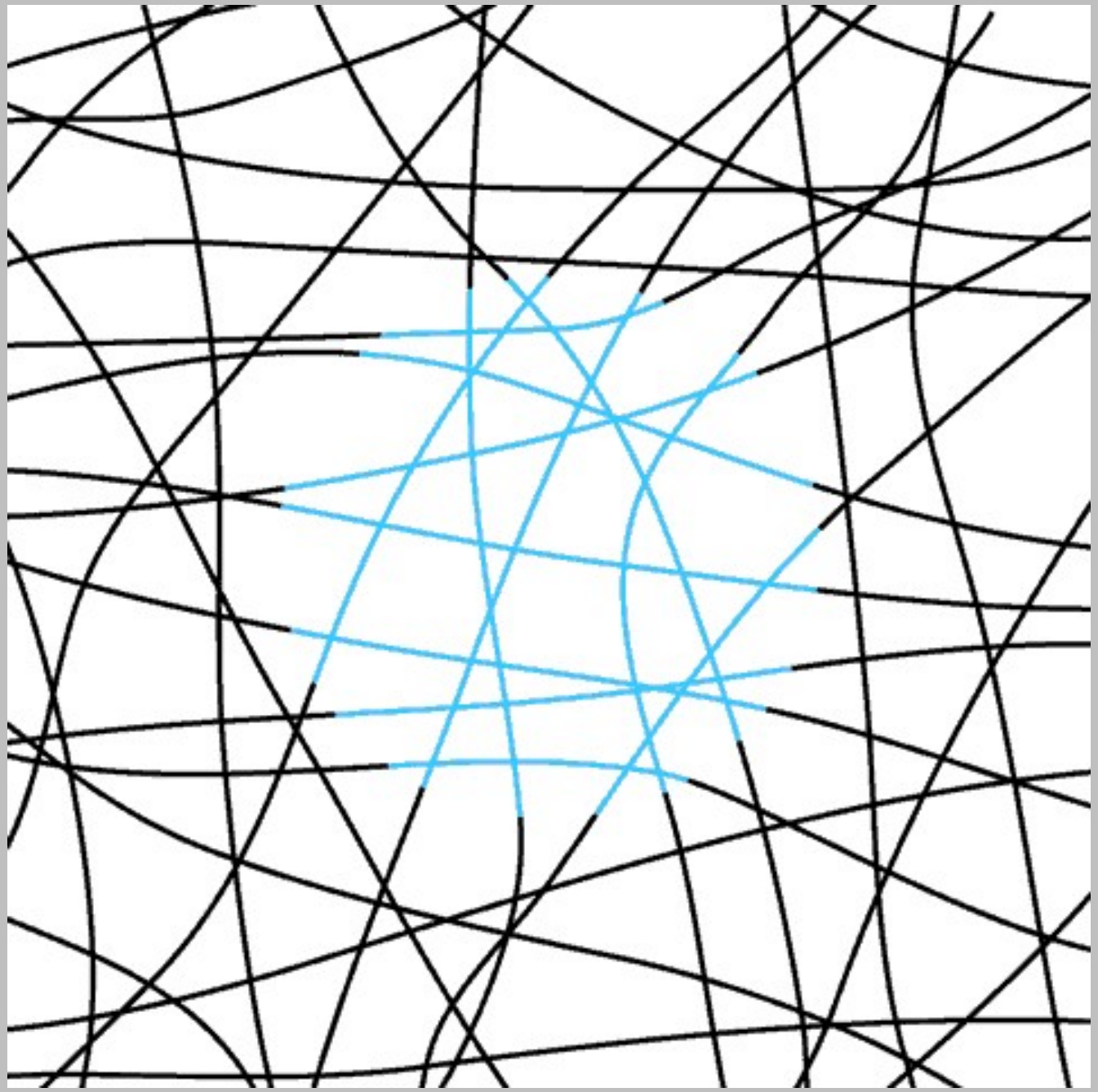
- Next two lectures: networks that *generate* images

- Reminder: we have seen network-generated images before (*pre-image method*). More of that today.

Why do we care about image synthesis

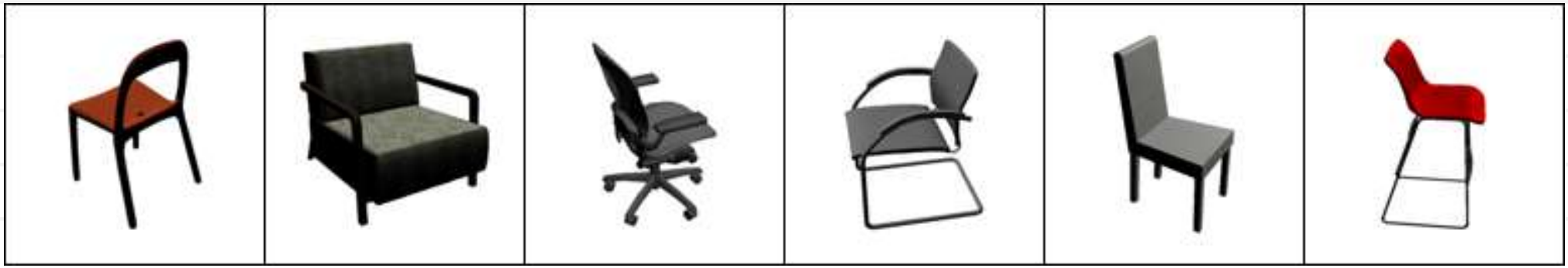
1. Fun
2. A harder problem, a frontier in AI
3. Computer graphics, design, image editing
4. To improve computer vision ("analysis-by-synthesis")

Analysis by synthesis in your brain



Feed-forward conditional generation

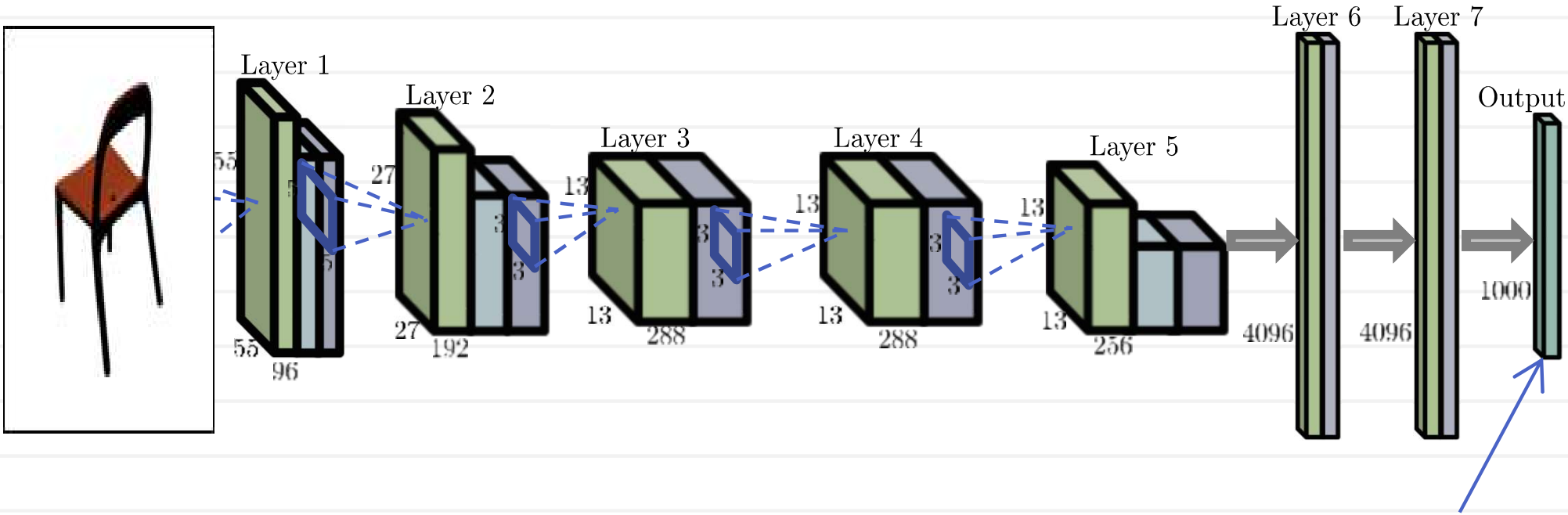
Source images:



Each image has "chair ID", and viewpoint direction

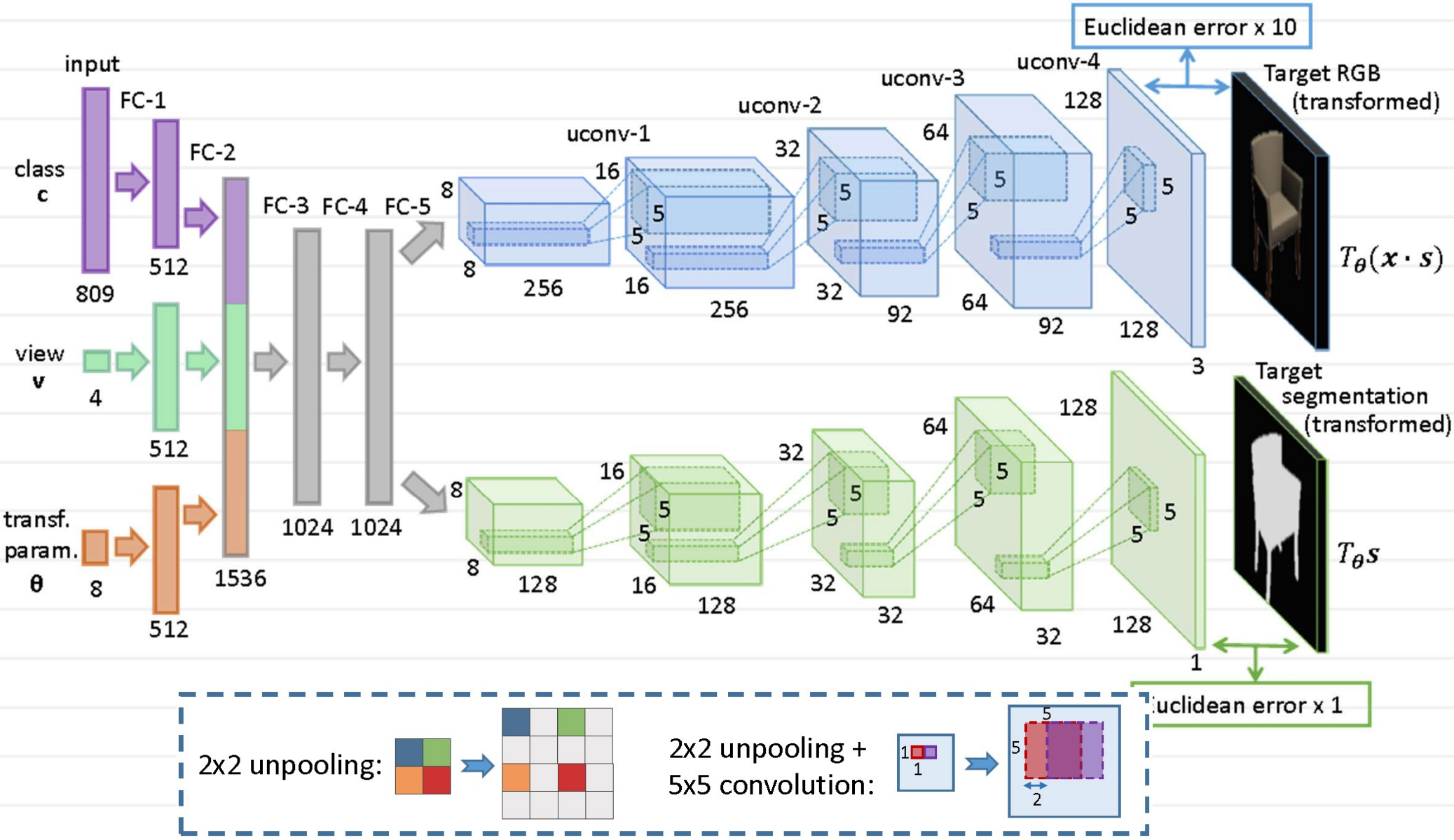
[Dosovitskiy et al. CVPR 2015]

"Natural" dataflow



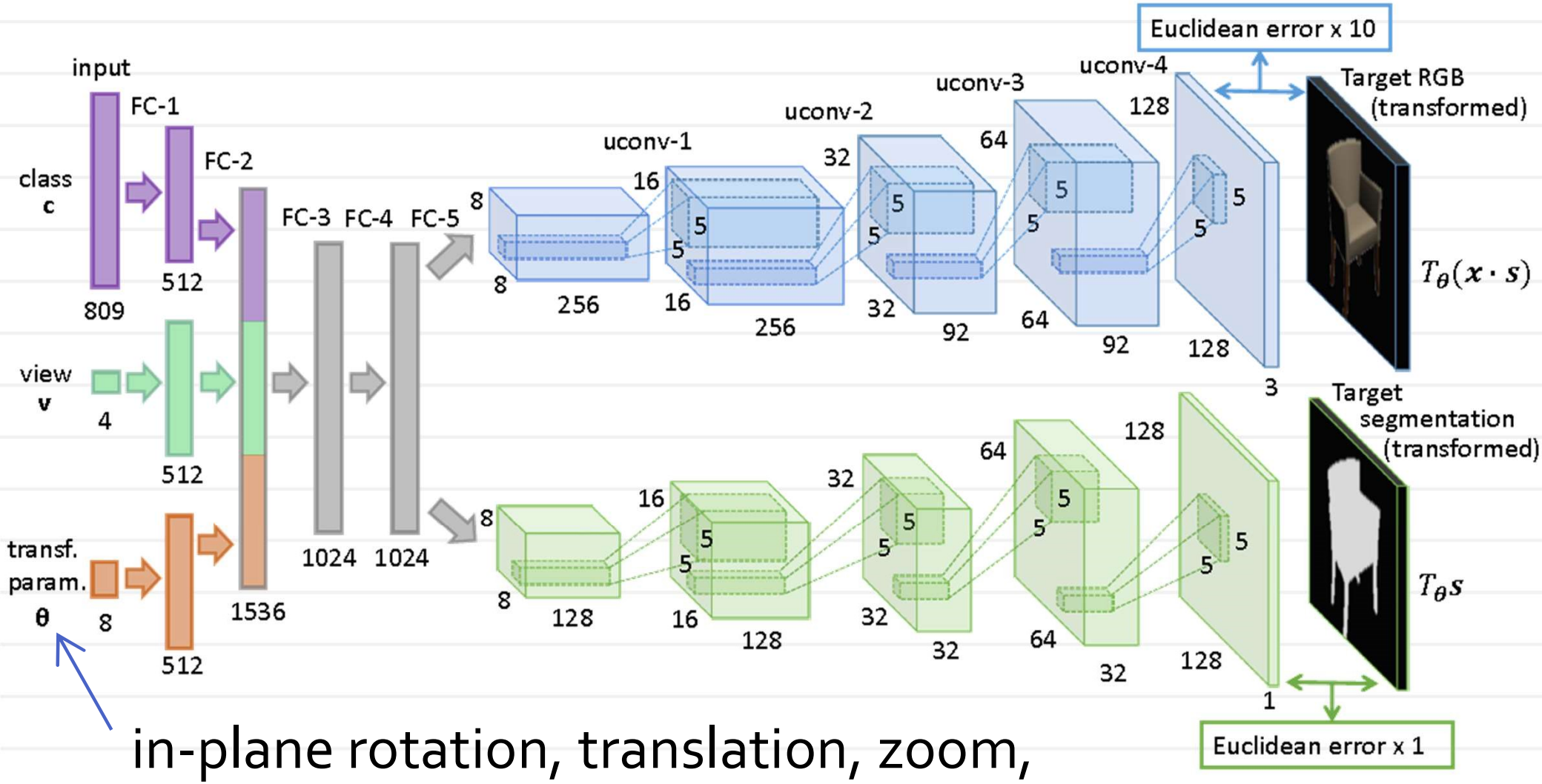
chair type
view angle
etc.

Feed-forward conditional generation



[Dosovitskiy et al. CVPR 2015]

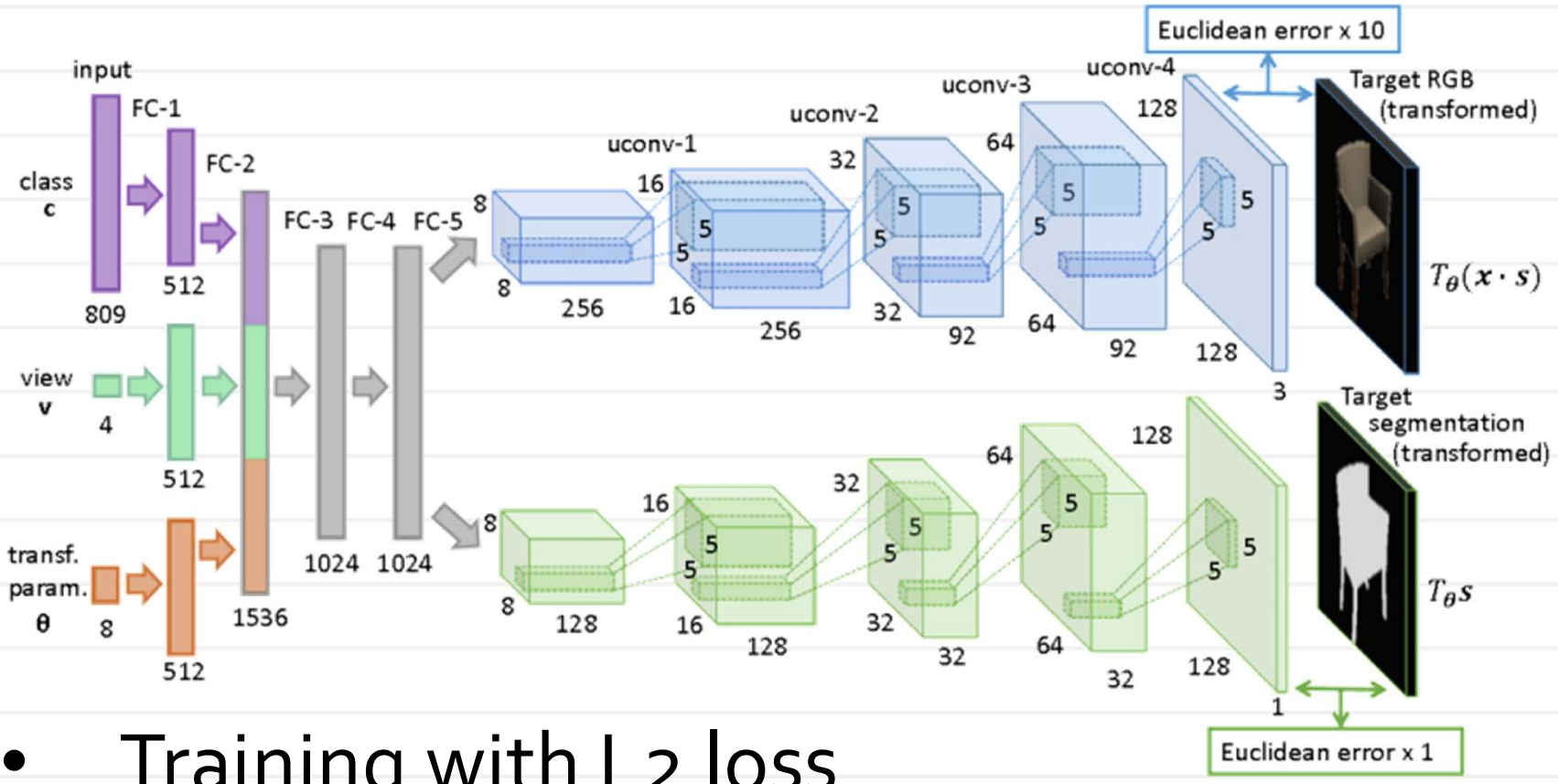
Feed-forward conditional generation



in-plane rotation, translation, zoom, stretching horizontally or vertically, changing hue, changing saturation, changing brightness

[Dosovitskiy et al. CVPR 2015]

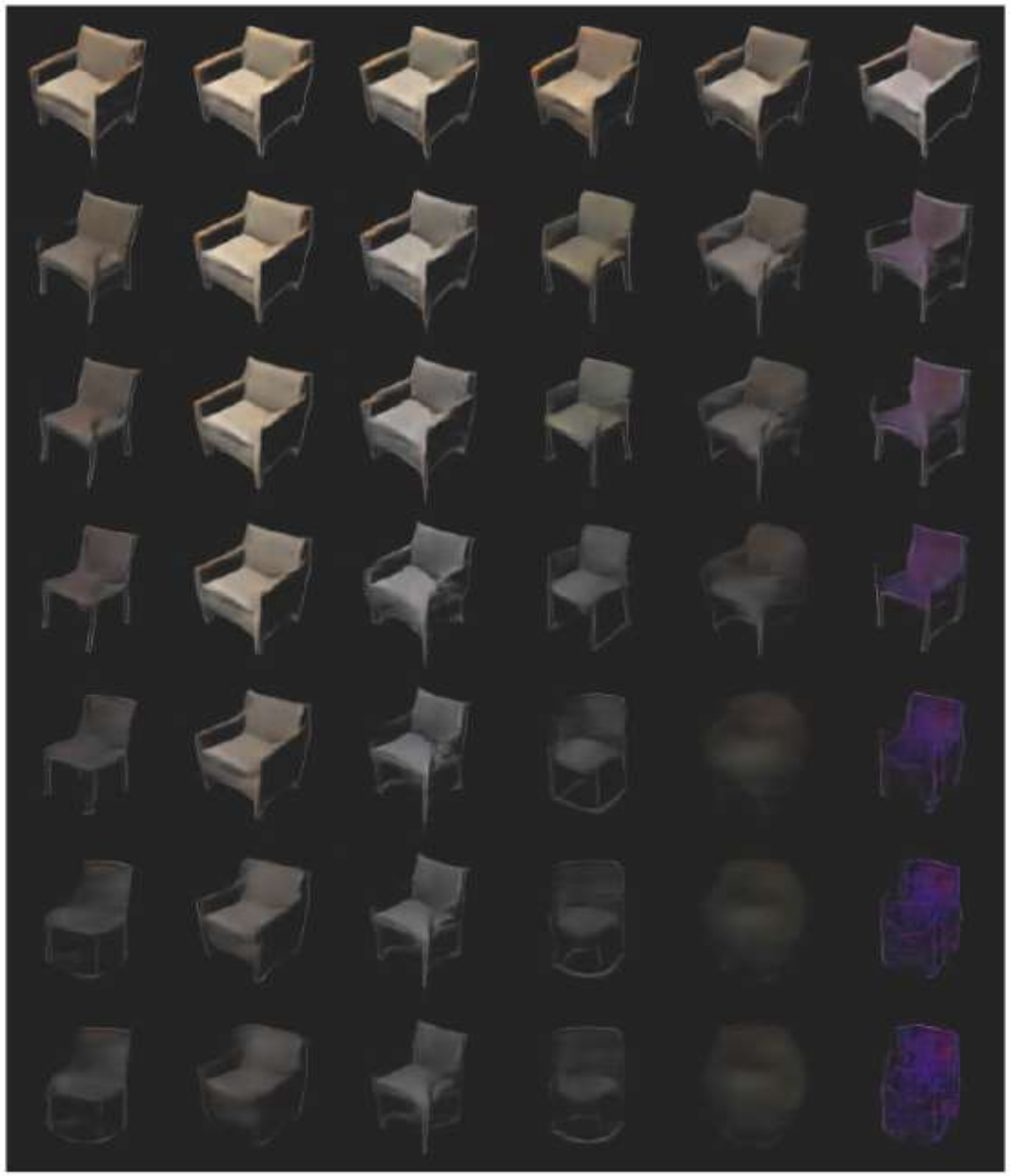
Feed-forward conditional generation

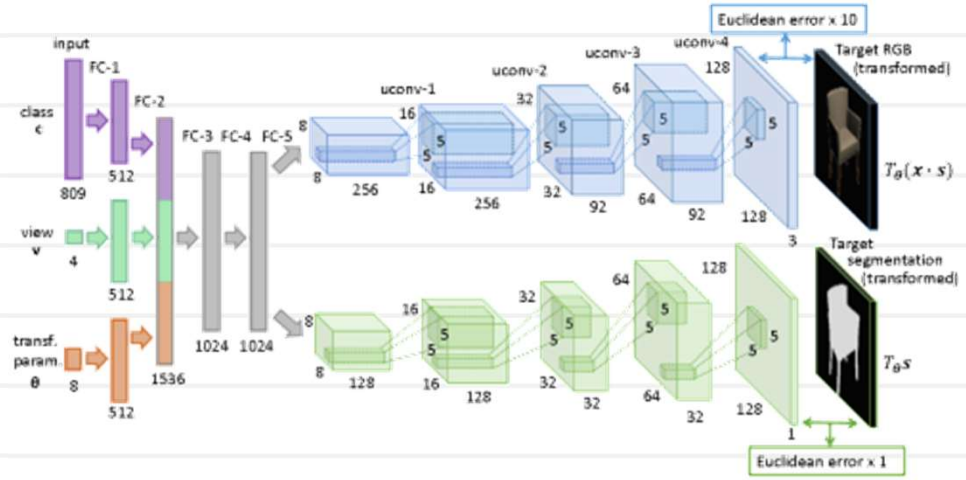


- Training with L2 loss
- Data augmentation
- Mask is used to regularize (multi-task learning) and to clip the results

[Dosovitskiy et al. CVPR 2015]

Feed-forward conditional generation

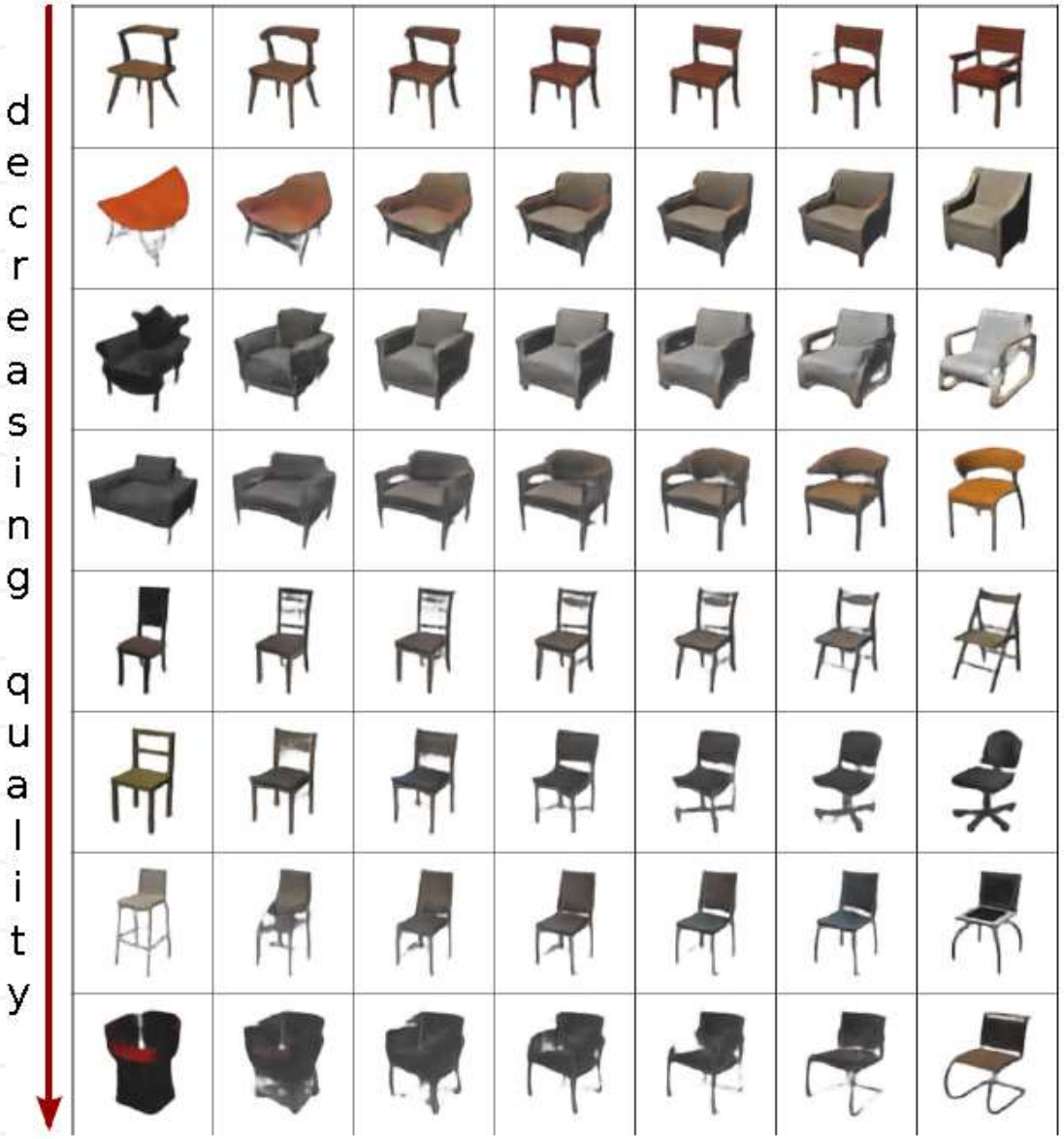


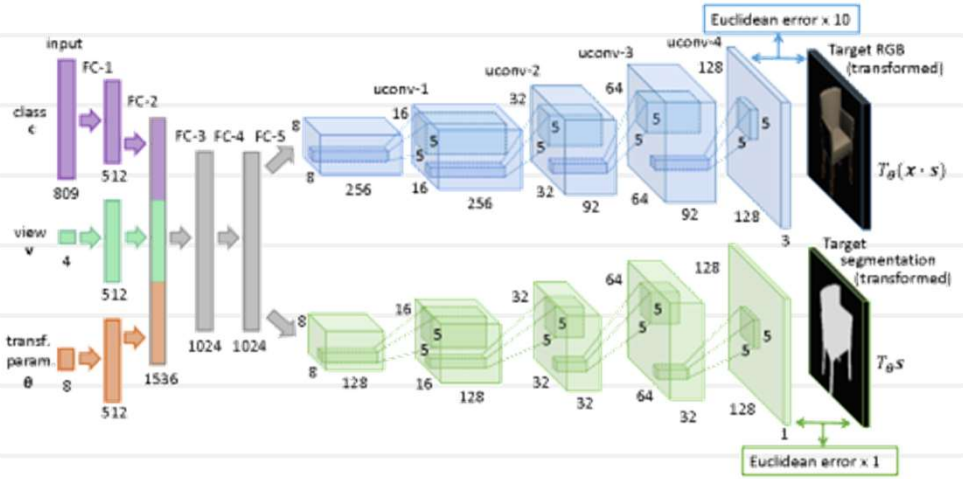


Increasing a single neuron in FC1

[Dosovitskiy et al. CVPR 2015]

Feed-forward conditional generation

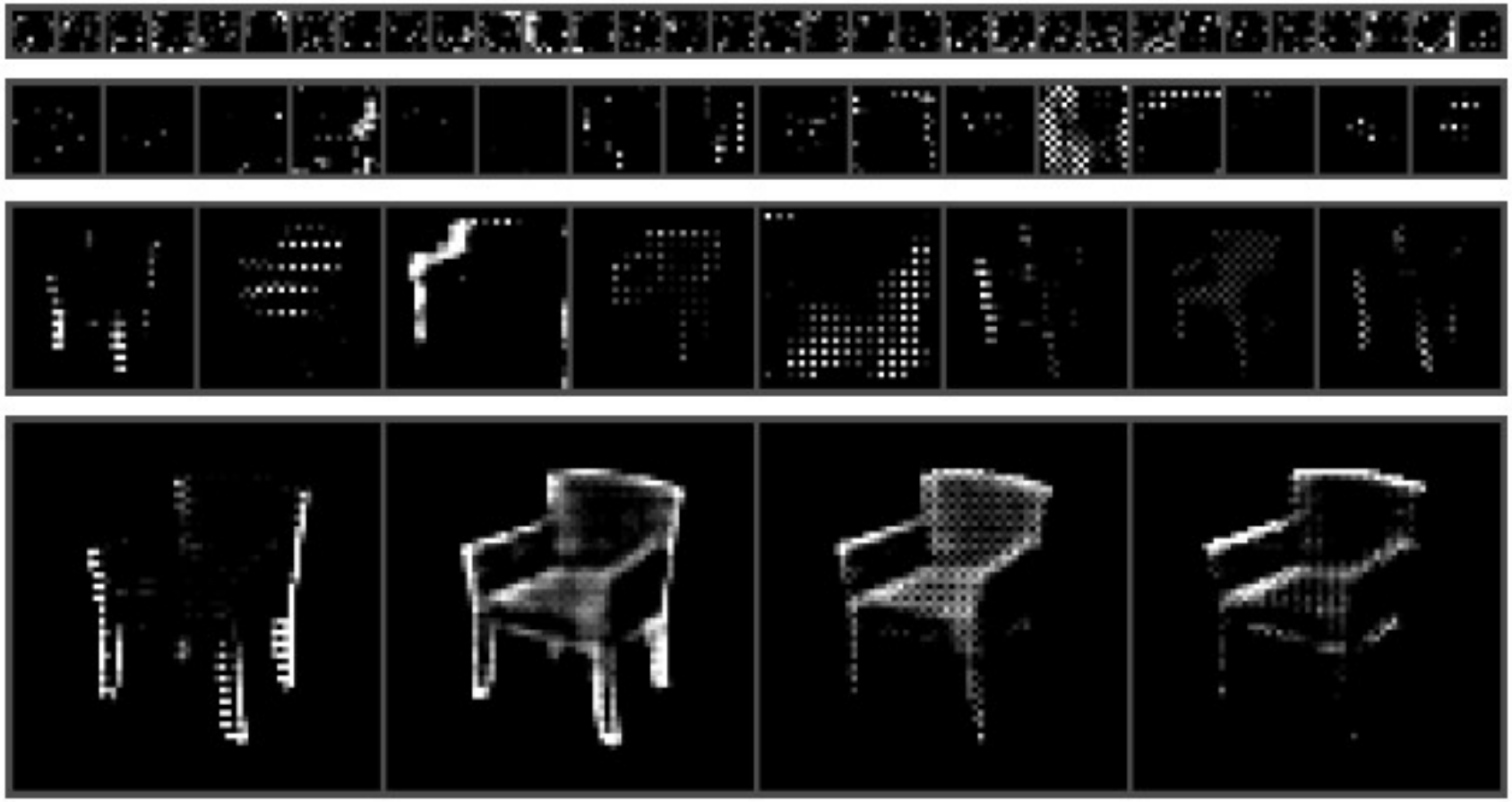


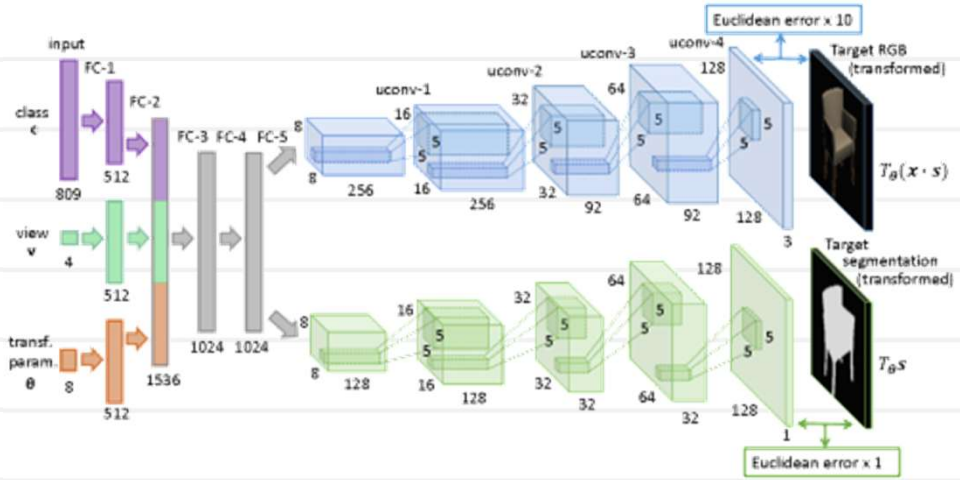


Morphing between chairs

[Dosovitskiy et al. CVPR 2015]

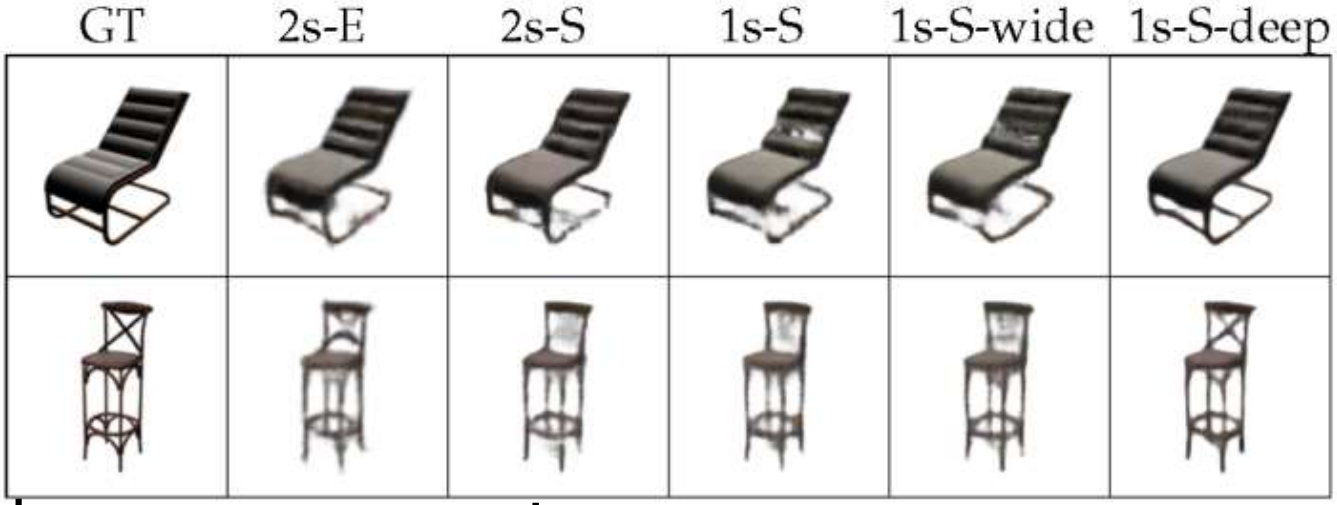
Feed-forward conditional generation



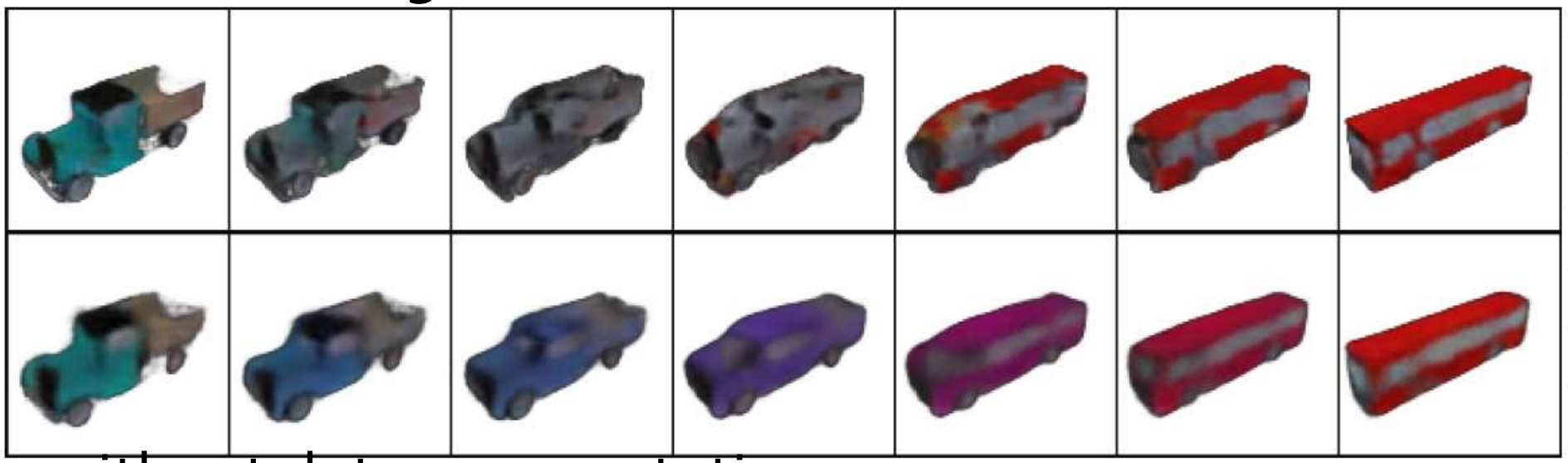


[Dosovitskiy et al. CVPR 2015]

More results



with data augmentation



without data augmentation

Blurriness appears as a result of the L2-loss

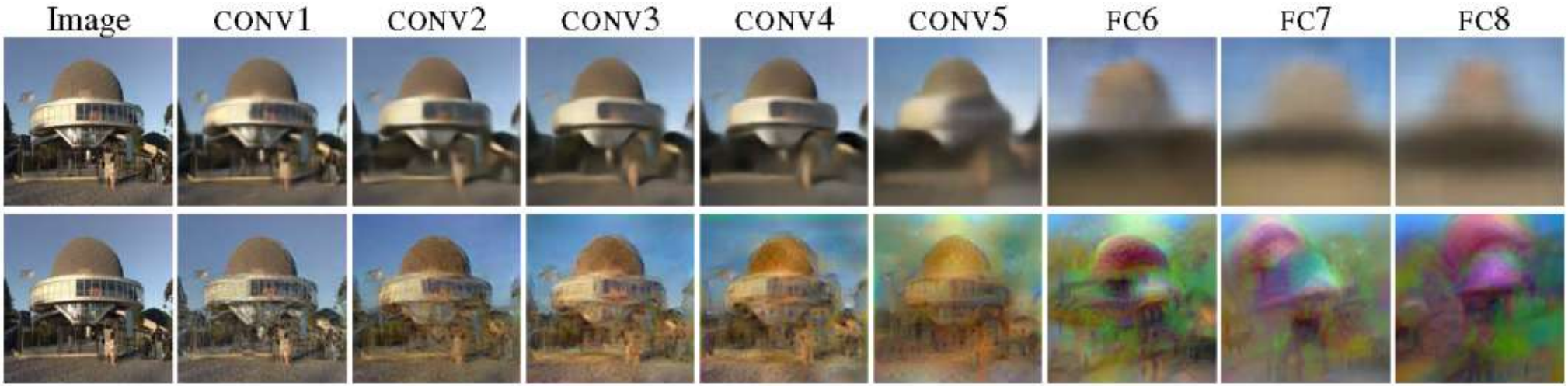
[Dosovitskiy et al. CVPR 2015]

Inverting CNNs with ConvNets

[Dosovitskiy and Brox, Arxiv15]:

- Train a special generating network to reconstruct the original image from AlexNet activations
- Their architecture for FC: 3 FC + 5 up-conv layers
- Their architecture for Conv layers: several conv + several up-convolutional layers

Their result:

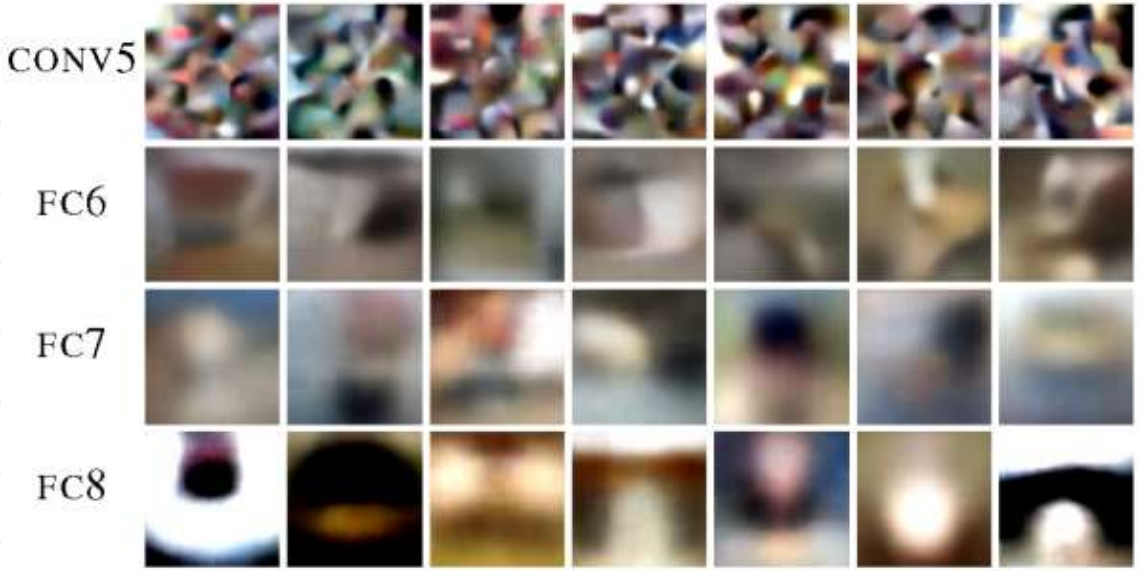


Pre-image method of Mahendran and Vedaldi (Lecture 5)

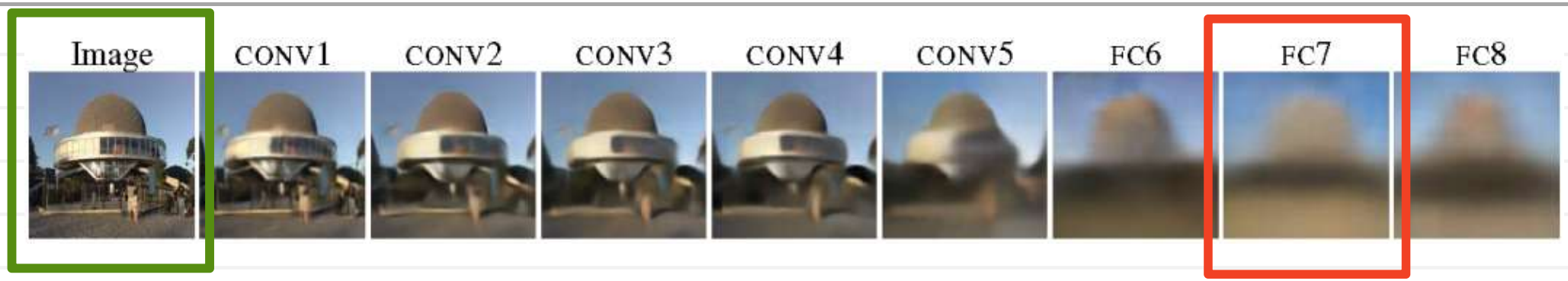
Experiments with the inverting architecture

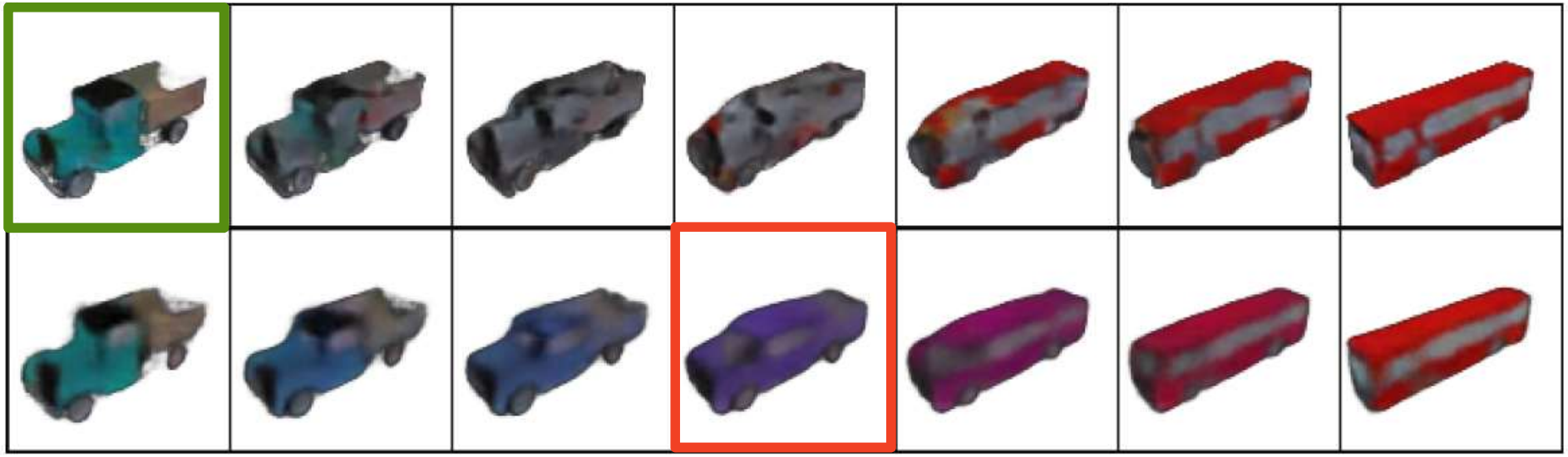
Interpolation between two images:

Sampling from random activations:



Let us look at reconstruction deficiencies





- How can **we** tell between real and synthetic?
- Can a **computer** tell between them?

Let us look at reconstruction deficiencies

- How can we tell between real and synthetic?
- Can a **computer** tell between them?

content

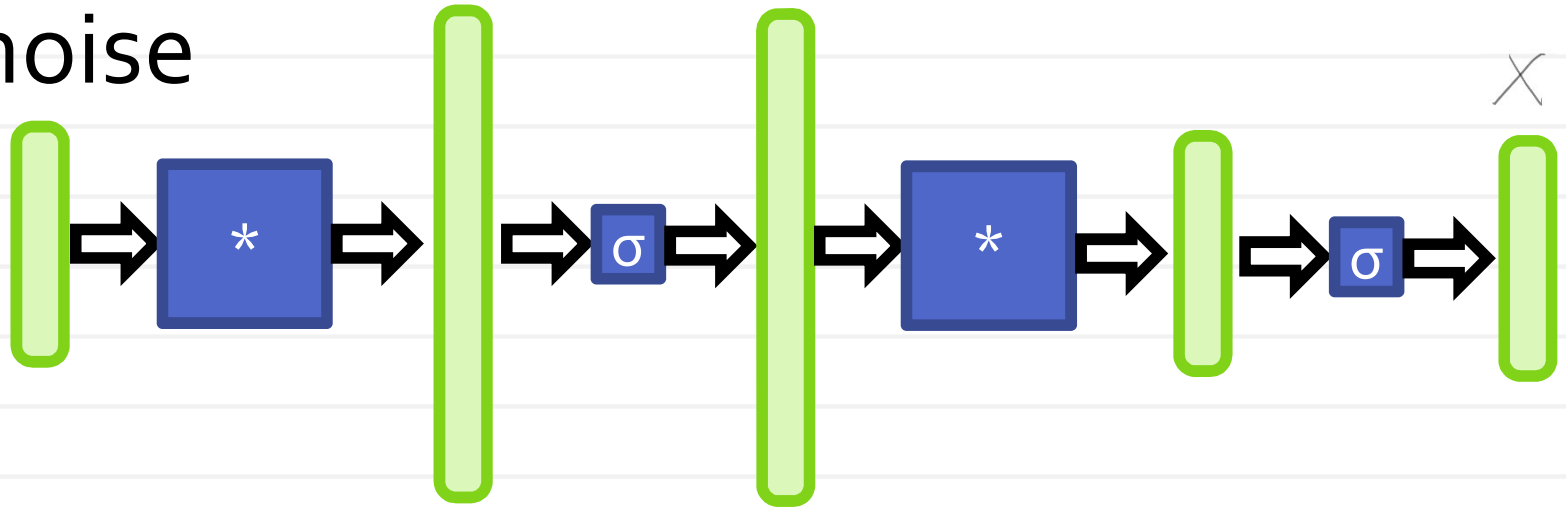
feedback

Generator Network: generates content (e.g. images)

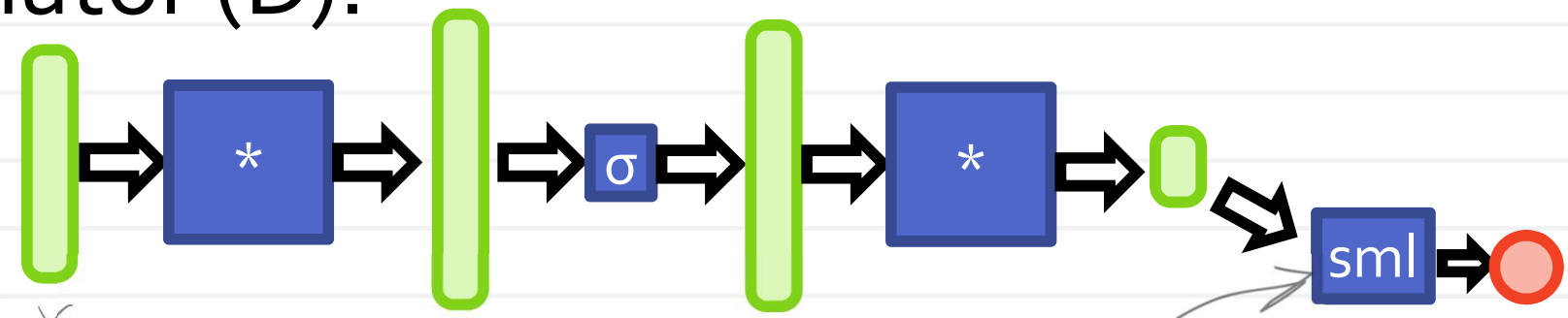
Discriminator Network: assesses the result (e.g. images)

Adversarial generative networks

Generator (G):



Discriminator (D):



[Goodfellow et al. NIPS 14]

Adversarial learning

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

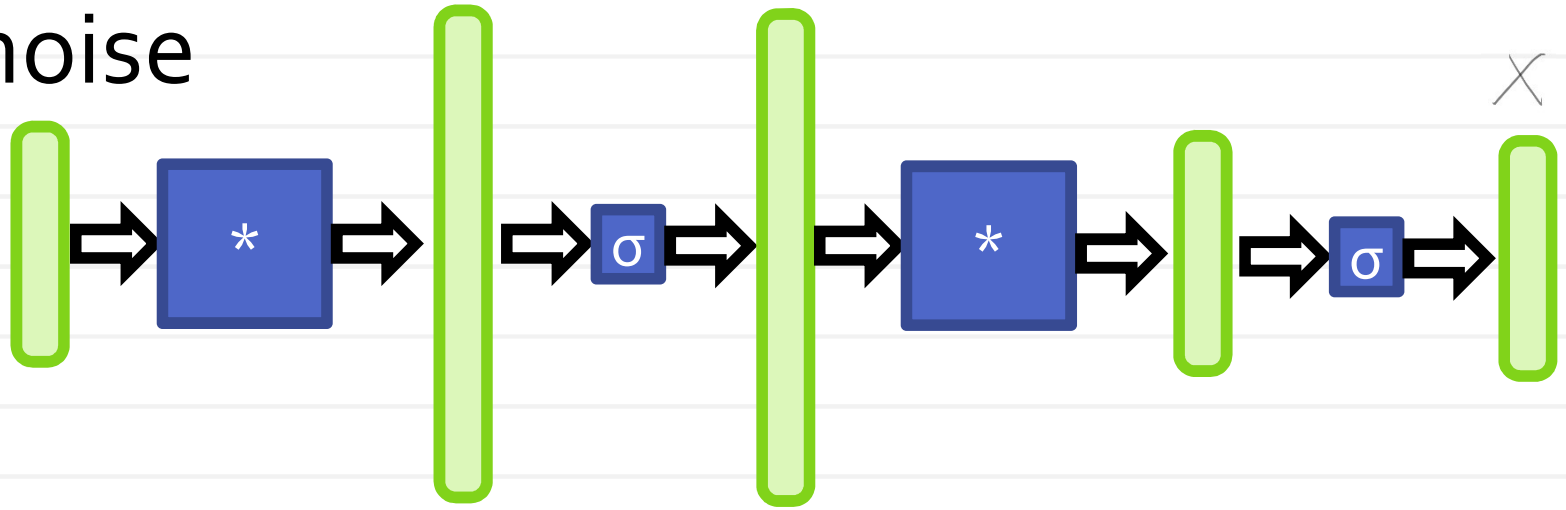
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

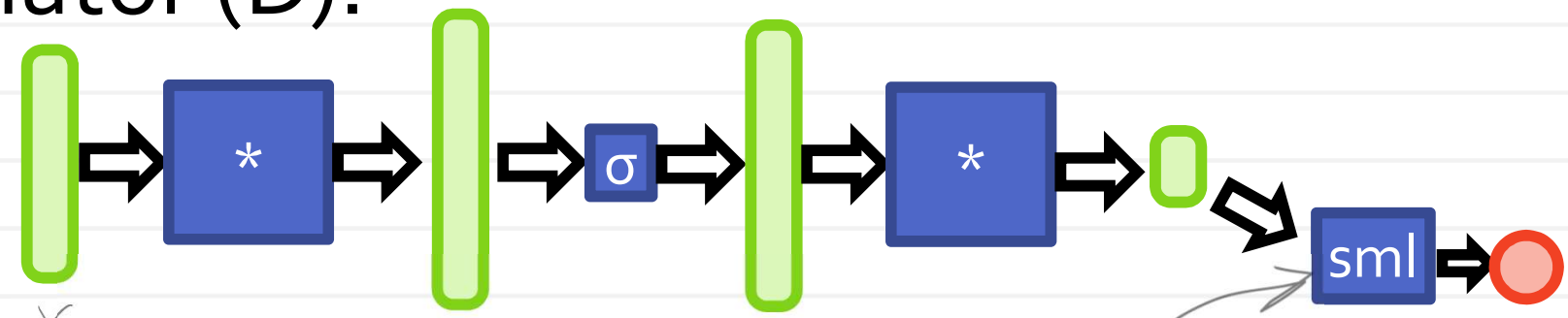
[Goodfellow et al. NIPS 14]

Adversarial generative networks

Generator (G):



Discriminator (D):

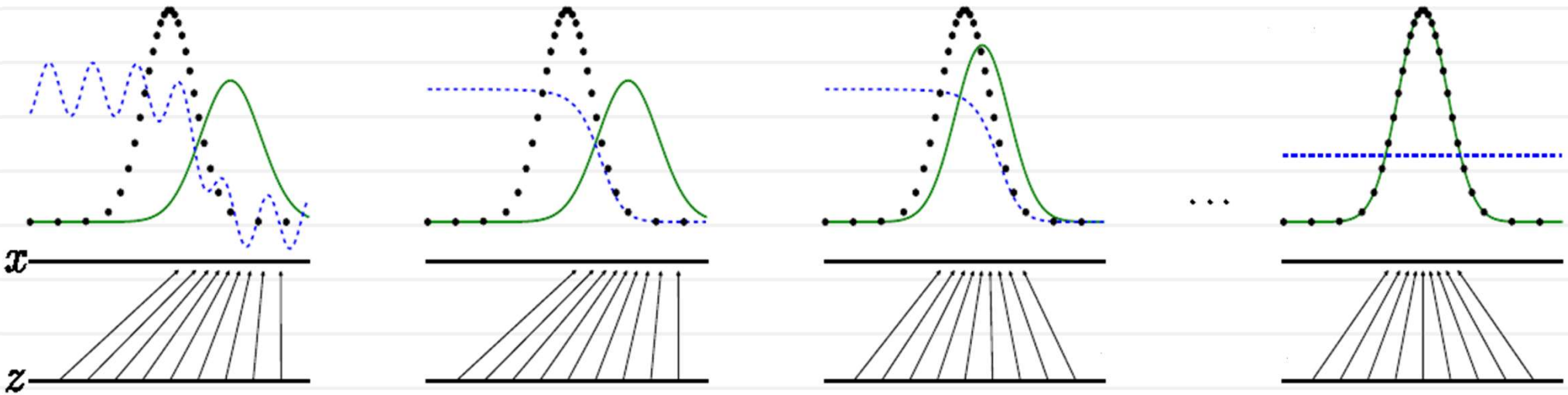


[Goodfellow et al. NIPS 14]

Adversarial learning

Looking for a *saddle point*:

$$\min_G \max_D V(D,G) = E_{x \sim p_D(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$



[Goodfellow et al. NIPS 14]

Modified GANs

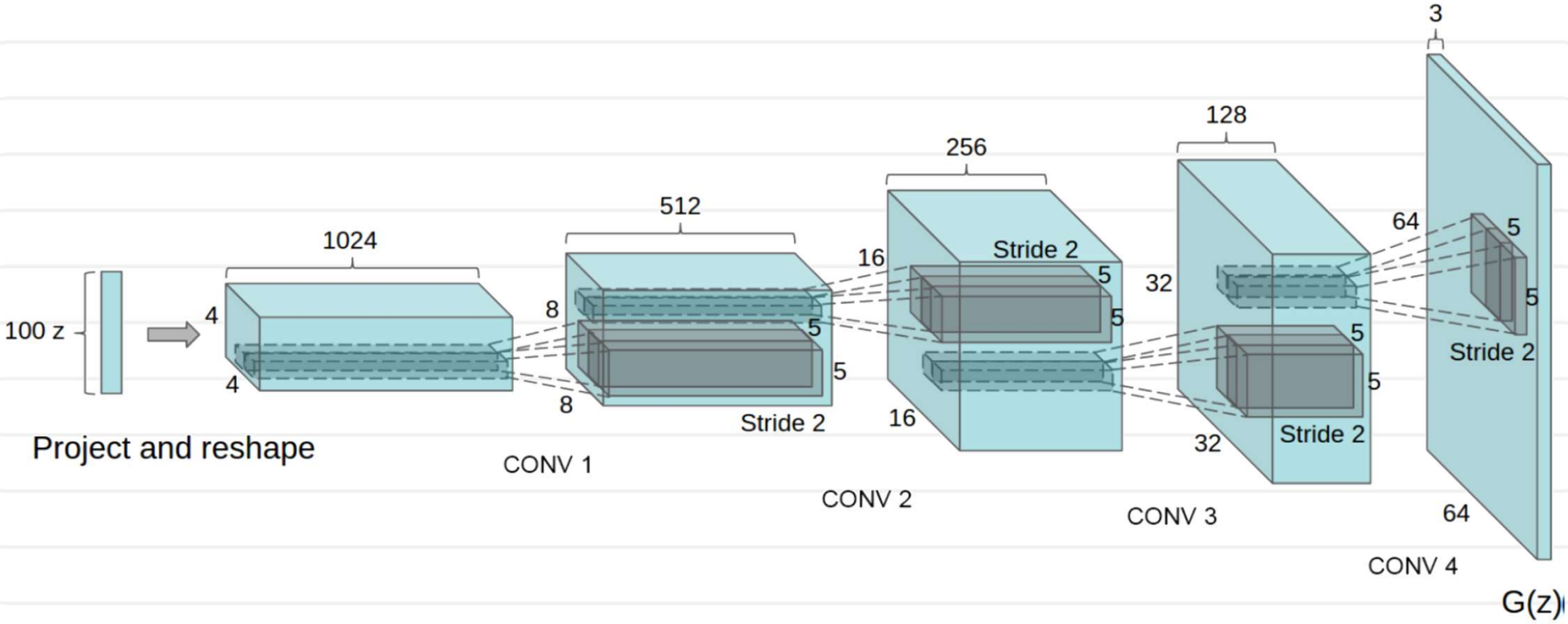
$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_D(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log (1 - D(G(z)))]$$

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log (1 - D(G(\boldsymbol{z})))$$

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D(G(\boldsymbol{z}))$$

- Not a zero-sum game anymore
- Works better in the initial stage, when discriminator is much better than generator

DCGAN



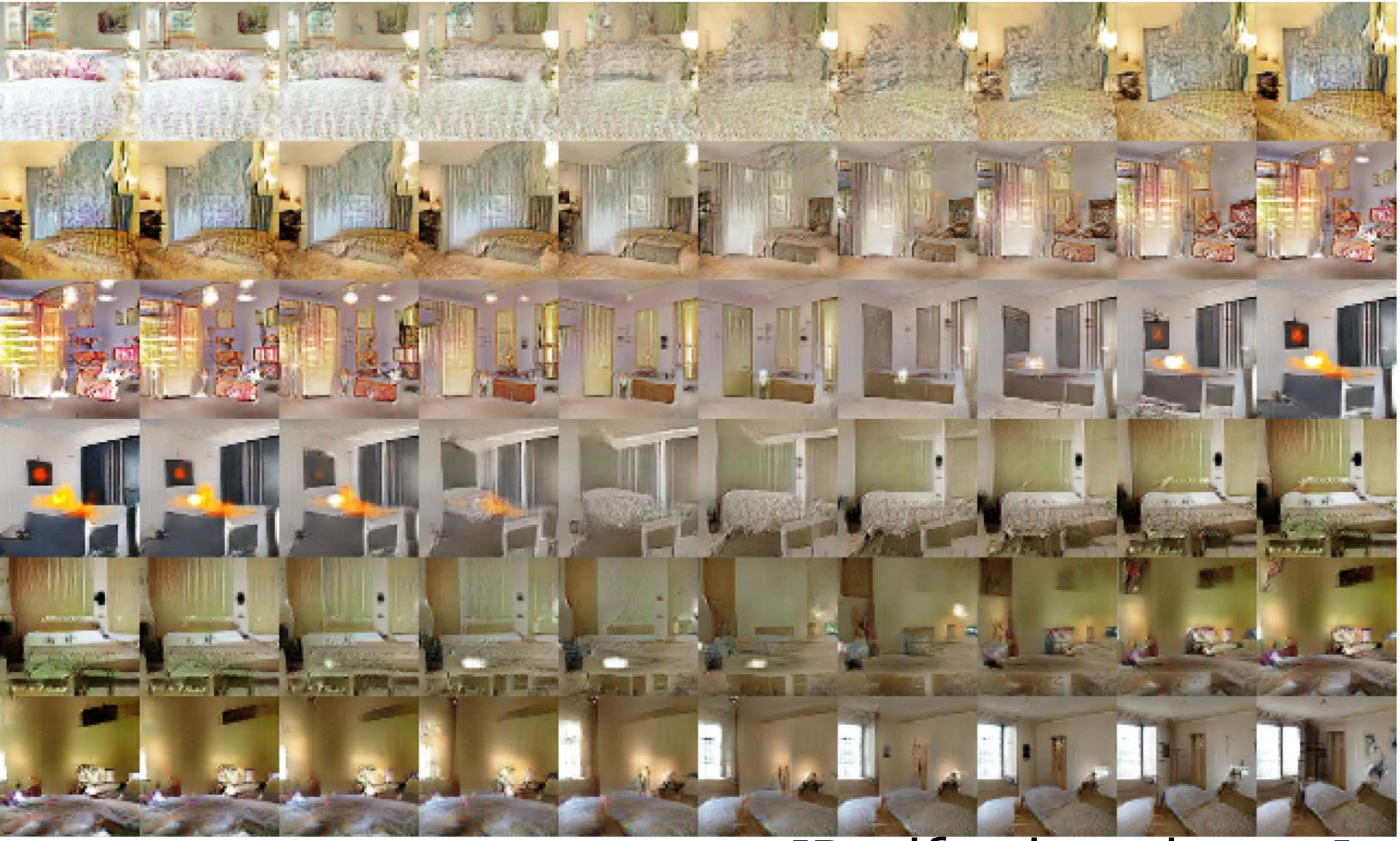
- Discriminator uses Leaky ReLU + tanH (last layer)

[Radford et al.2015]

DCGAN

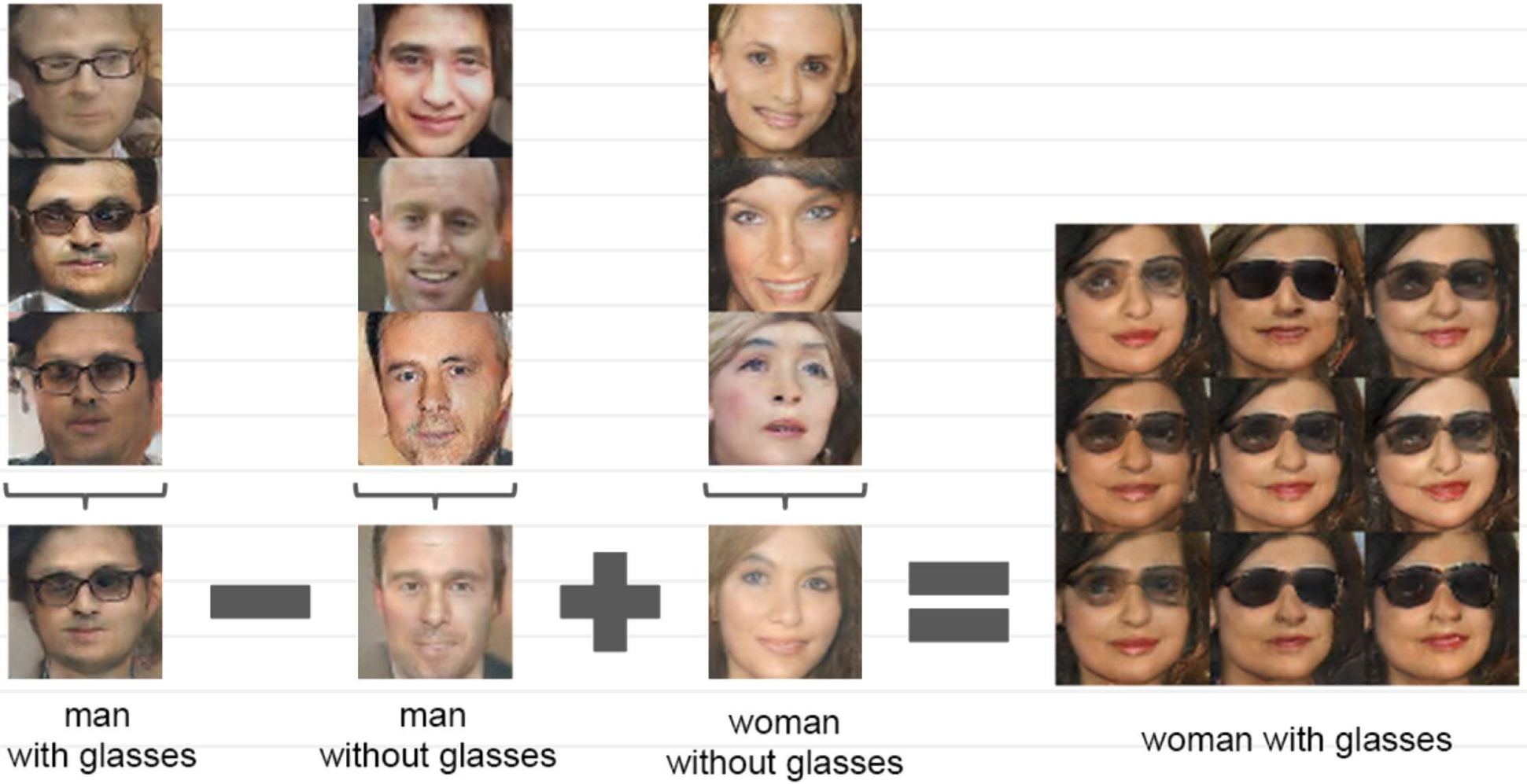
[Radford et al.2015]

DCGAN



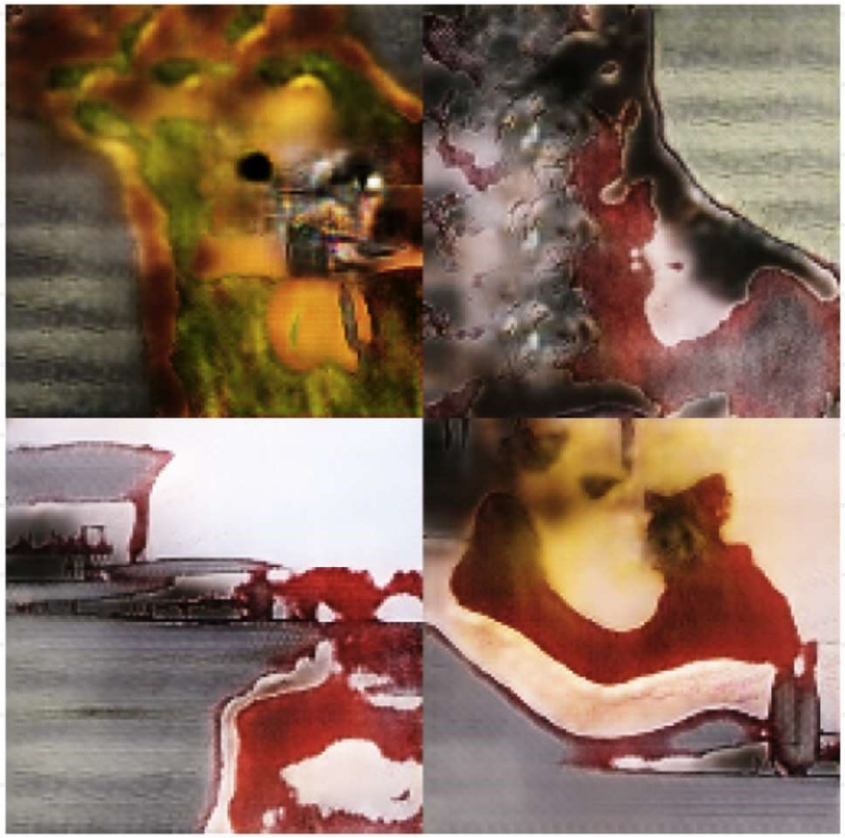
[Radford et al.2015]

DCGAN



[Radford et al.2015]

SOTA with GANs



DCGAN

[Salimans 16]

Optimal discriminator

$$\min_G \max_D V(D,G) = E_{x \sim p_D(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

$p_{model}(x)$ vs $p_{data}(x)$

Optimizing discriminator gives:

$$D(x) = \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)}$$

Optimal discriminator works as *ratio estimator*!

Optimal generator

$$\min_G \max_D V(D,G) = E_{x \sim p_D(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

What is the optimal generator for the fixed discriminator?

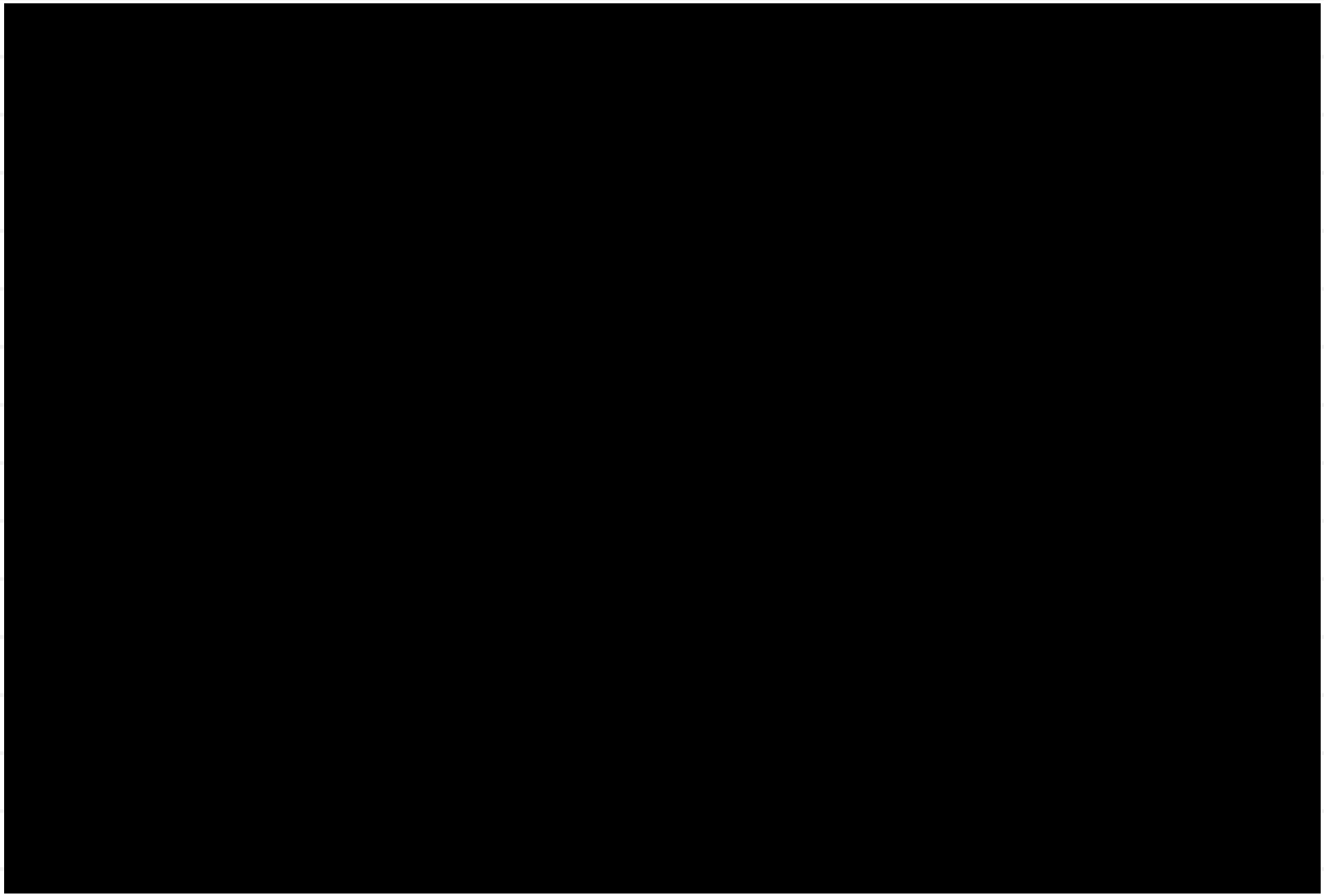
Answer: $G(z) = \arg\max_x D(x)$

Main source of instability ("mode collapse").

Problem with GAN convergence

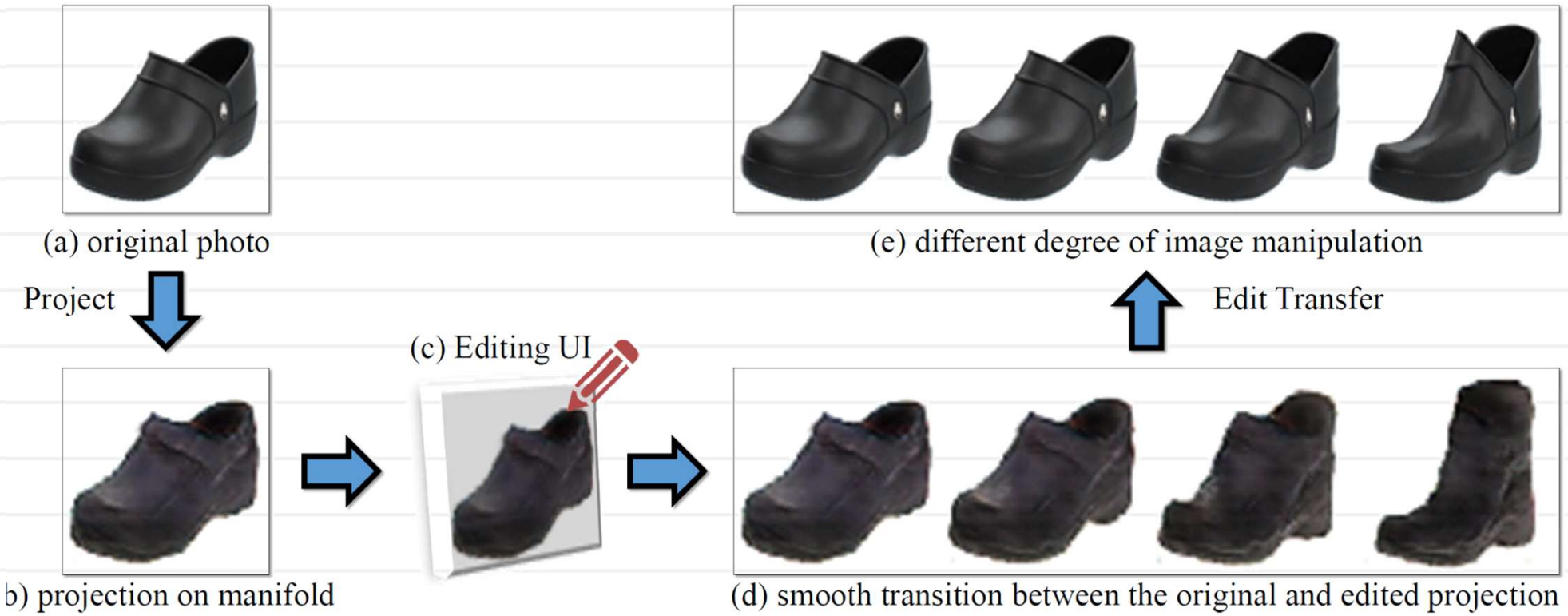
- Gradient updates are not guaranteed to converge to a saddle point
- Local generator collapse (parts of distribution collapse to a single point)
- Low diversity solution still look good for a naked eye
- A lot of recent research is about preventing GAN collapse (e.g. Unrolled GAN [Metz et al. 2016])

Image editing on image manifold



[Zhu et al. 2016]

Image editing on image manifold



Projecting x^R on manifold: $$x^* = \arg\min_{x \in \tilde{\mathbb{M}}} \mathcal{L}(x, x^R)$$

Combination of L2-distance and VGG-activation-based distance

[Zhu et al. 2016]

Image editing on image manifold

Projecting x^R on manifold:

$$x^* = \arg\min_{x \in \tilde{\mathbb{M}}} \mathcal{L}(x, x^R)$$

Method of projection:

- Optimizing in the latent space z (hard, slow)
- Predicting z by a feedforward network:

$$\theta_P^* = \arg\min_{\theta_P} \sum_n \mathcal{L}(G(P(x_n^R; \theta_P)), x_n^R)$$

- Both: predict initialization and then optimize

[Zhu et al. 2016]

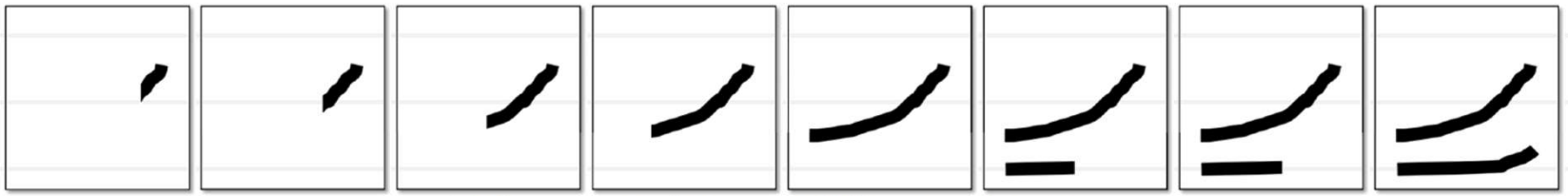
Image editing on image manifold

[Zhu et al. 2016]

Image editing on image manifold

Editing process:

$$z^* = \arg\min_{z \in \mathbb{Z}} \Big\{ \underbrace{\sum_g \|f_g(G(z)) - v_g\|^2}_{\text{data term}} + \underbrace{\lambda_s \cdot \|z - z_0\|^2}_{\text{manifold smoothness}} + E_D \Big\}$$



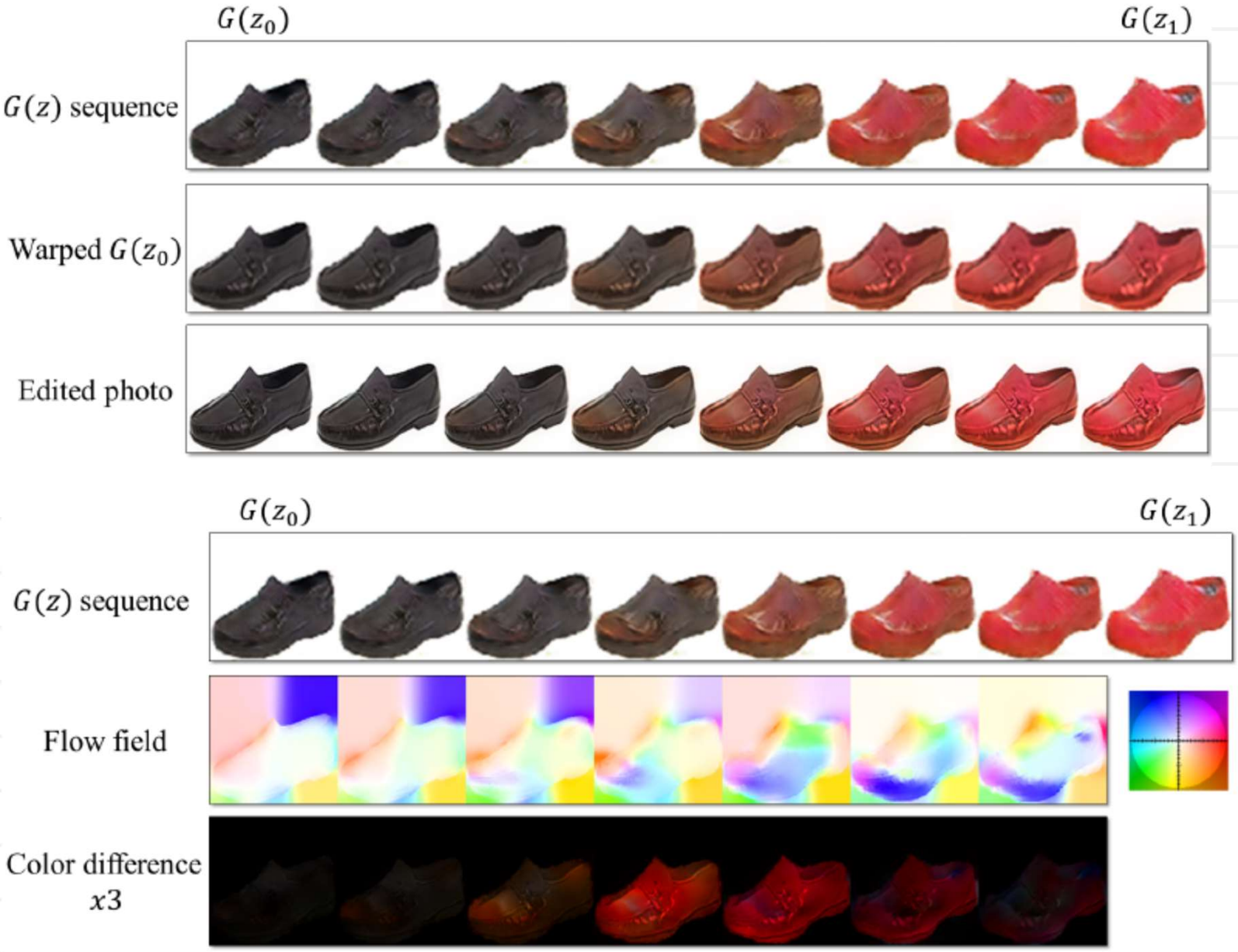
(a) User constraints v_g at different update steps

$G(z_0)$ (b) Updated images according to user edits $G(z_1)$

(c) Linear interpolation between $G(z_0)$ and $G(z_1)$

[Zhu et al. 2016]

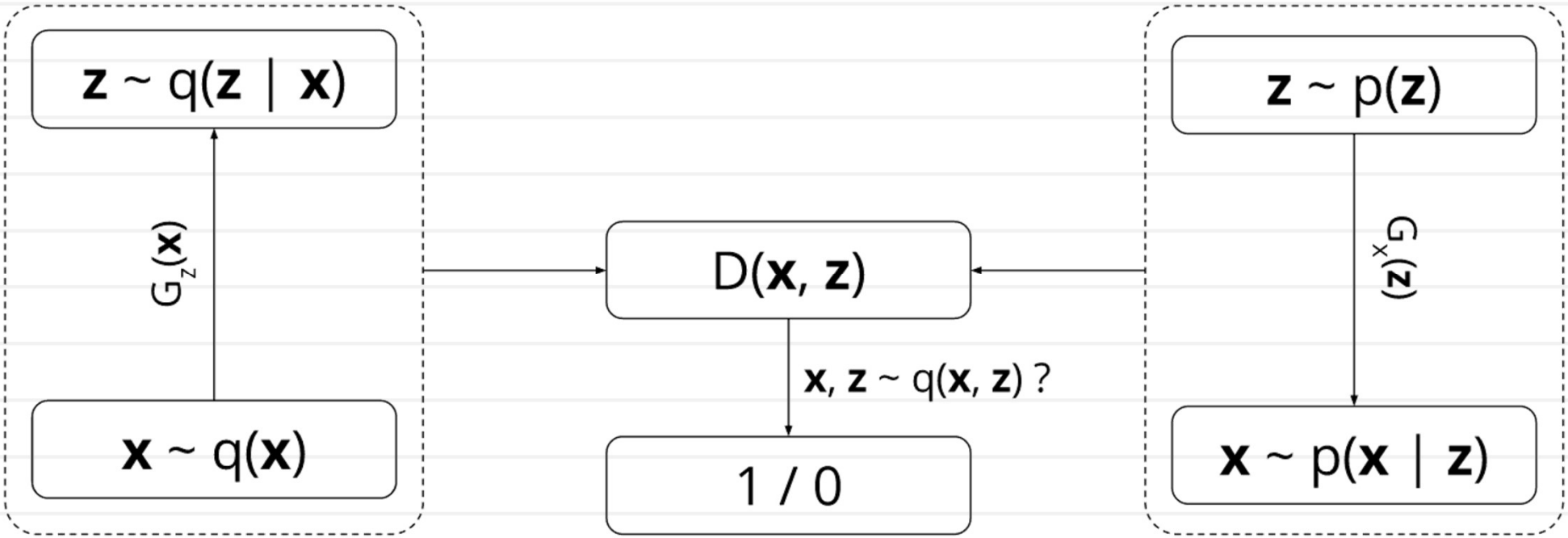
Transfering edits



[Zhu et al. 2016]

Adversarially-learned inference

Idea: train z->x and x->z mappings simulataneously



[Dumolin et al .2016]

Adversarially-learned inference

Samples z->x:

[Dumolin et al .2016]

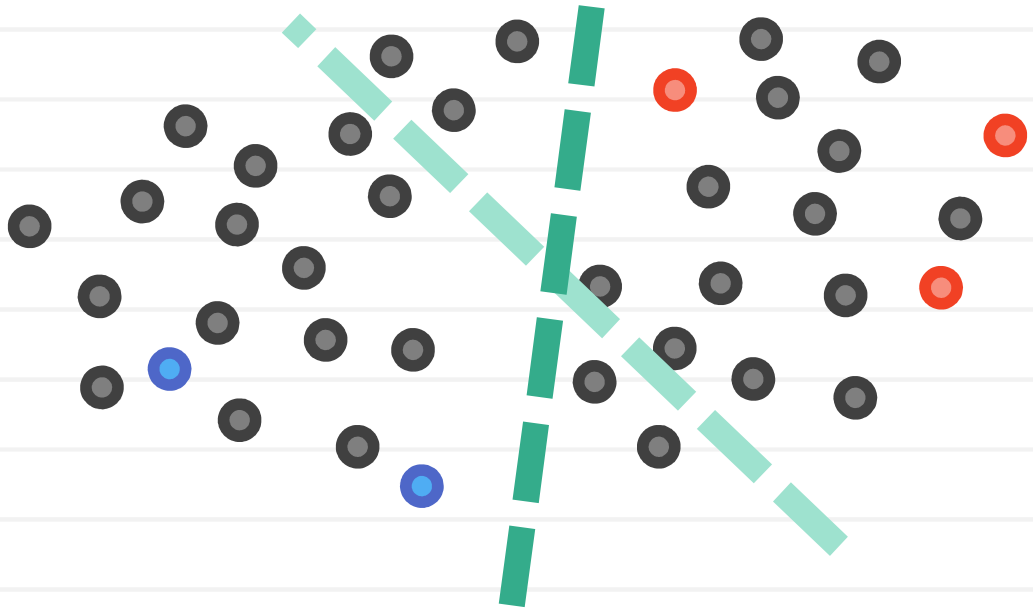
Adversarially-learned inference

Reconstructions

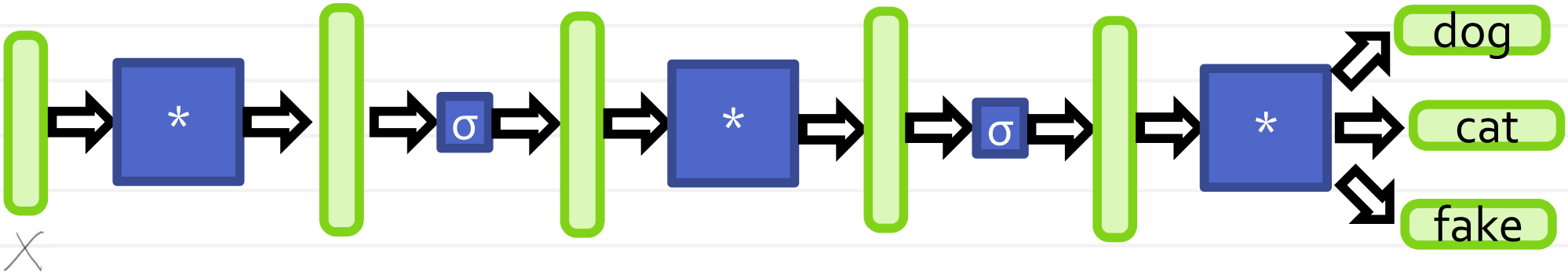
x->z->x:

[Dumolin et al .2016]

Semi-supervised learning



New discriminator:



[Salimans 2016]

Semi-supervised learning

$$
\begin{aligned}
L &= -\mathbb{E}_{\boldsymbol{x},y\sim p_{\text{data}}(\boldsymbol{x},y)}[\log p_{\text{model}}(y|\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim G}[\log p_{\text{model}}(y = K+1|\boldsymbol{x})] \\
&= L_{\text{supervised}} + L_{\text{unsupervised}}, \text{ where} \\
L_{\text{supervised}} &= -\mathbb{E}_{\boldsymbol{x},y\sim p_{\text{data}}(\boldsymbol{x},y)} \log p_{\text{model}}(y|\boldsymbol{x}, y < K+1) \\
L_{\text{unsupervised}} &= -\{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \log[1 - p_{\text{model}}(y = K+1|\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim G} \log[p_{\text{model}}(y = K+1|\boldsymbol{x})]\}
\end{aligned}
$$

Perturbed MNIST results:

Model	Number of incorrectly predicted test examples for a given number of labeled samples			
	20	50	100	200
DGN [21]			333 ± 14	
Virtual Adversarial [22]			212	
CatGAN [14]			191 ± 10	
Skip Deep Generative Model [23]			132 ± 7	
Ladder network [24]			106 ± 37	
Auxiliary Deep Generative Model [23]			96 ± 2	
Our model	1677 ± 452	221 ± 136	93 ± 6.5	90 ± 4.2
Ensemble of 10 of our models	1134 ± 445	142 ± 96	86 ± 5.6	81 ± 4.3

[Salimans 2016]

Example: Internet images -> Webcam sensor

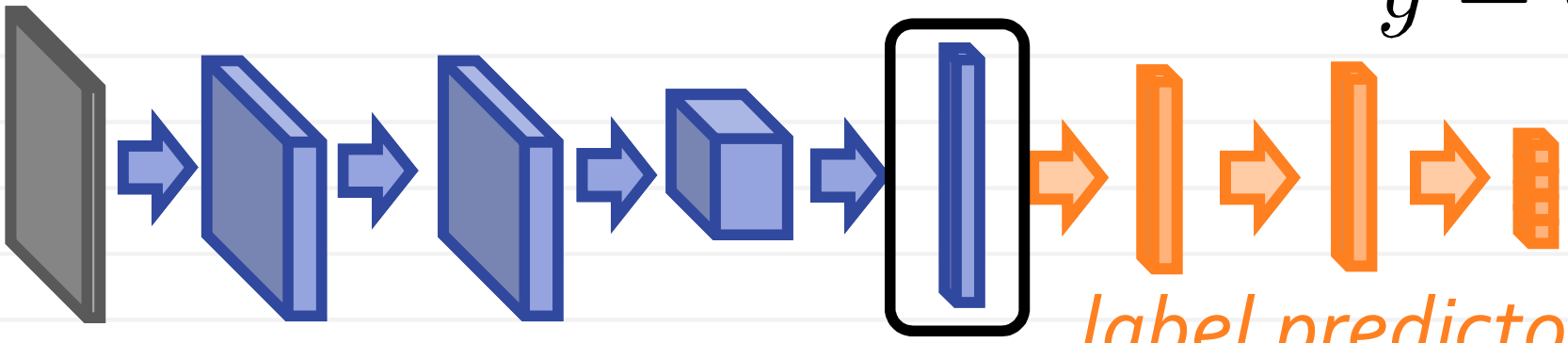
[Saenko et al. ECCV2010]

Domain shift in a deep architecture

$$\mathbf{f} = G_f(\mathbf{x}; \theta_f)$$

$$y = G_y(\mathbf{f}; \theta_y)$$

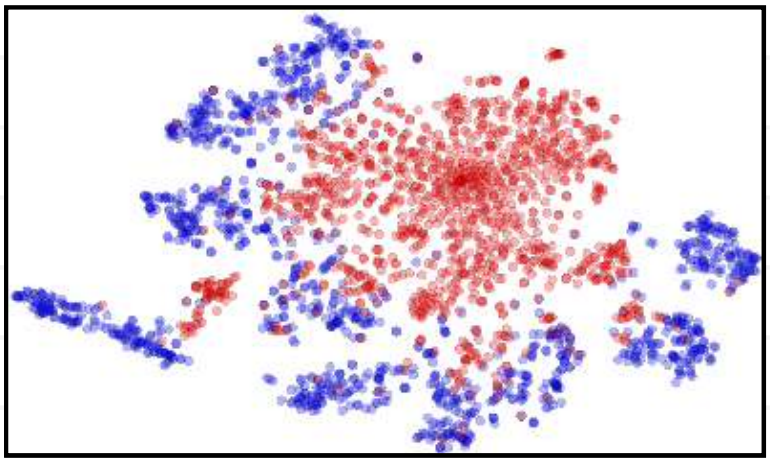
$\mathbf{x}$



feature extractor

label predictor

When trained on source only, feature distributions do not match:

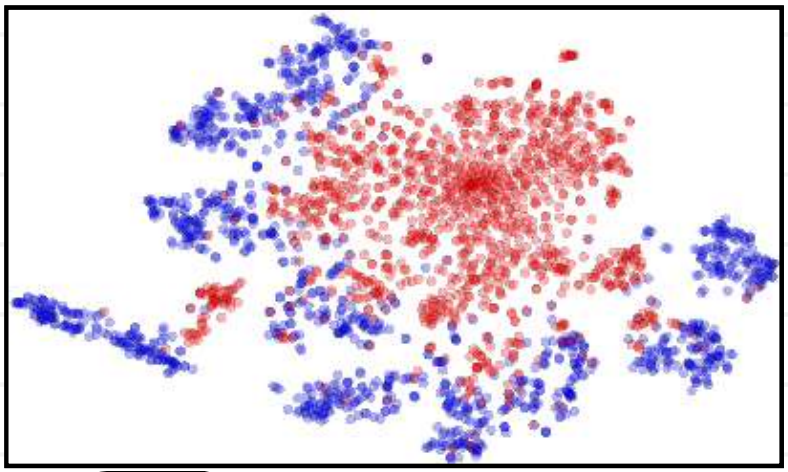


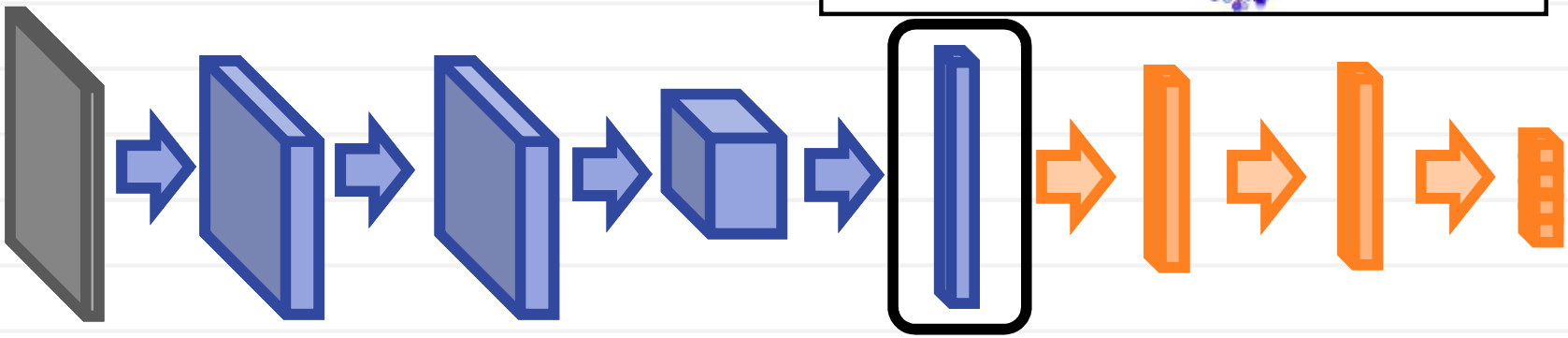
$$S(\mathbf{f}) = \{G_f(\mathbf{x}; \theta_f) \,|\, \mathbf{x} \sim S(\mathbf{x})\}$$

$$T(\mathbf{f}) = \{G_f(\mathbf{x}; \theta_f) \,|\, \mathbf{x} \sim T(\mathbf{x})\}$$

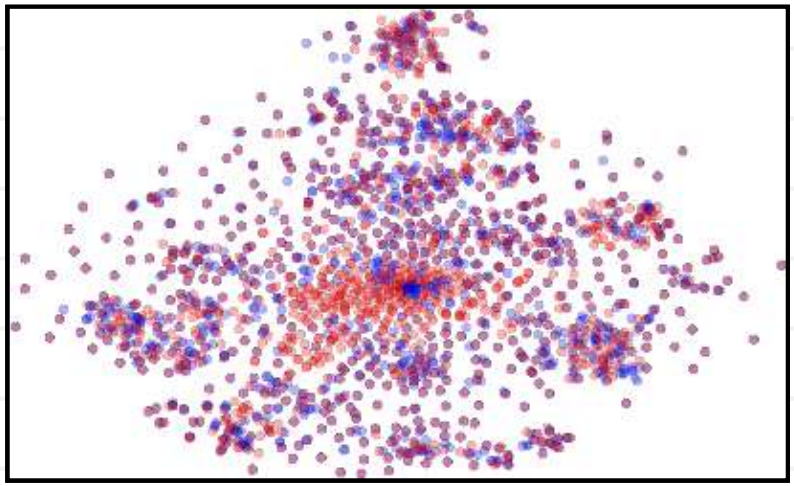
Idea 1: domain-invariant features wanted

Feature distribution without adaptation:

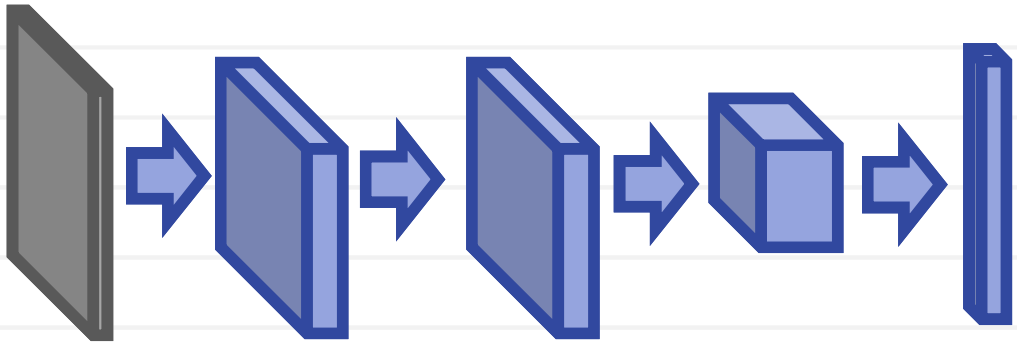


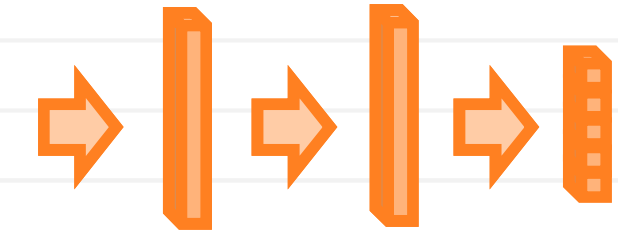


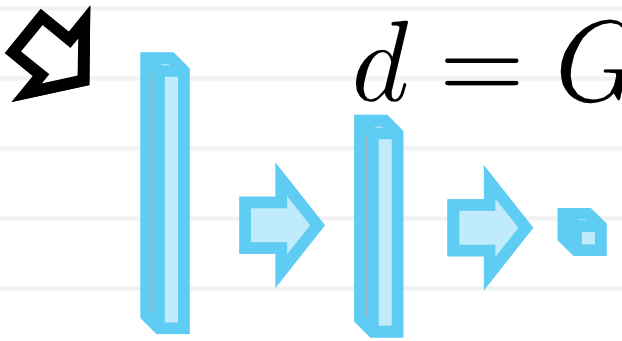
Our goal (after adaptation):



Idea 2: measuring domain shift

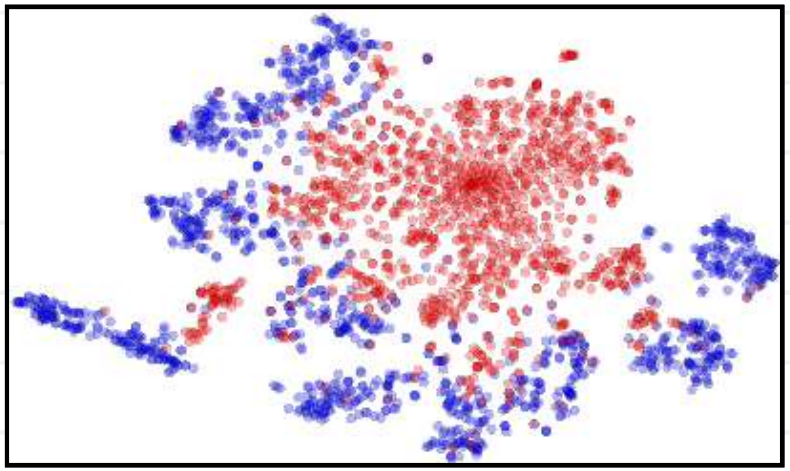




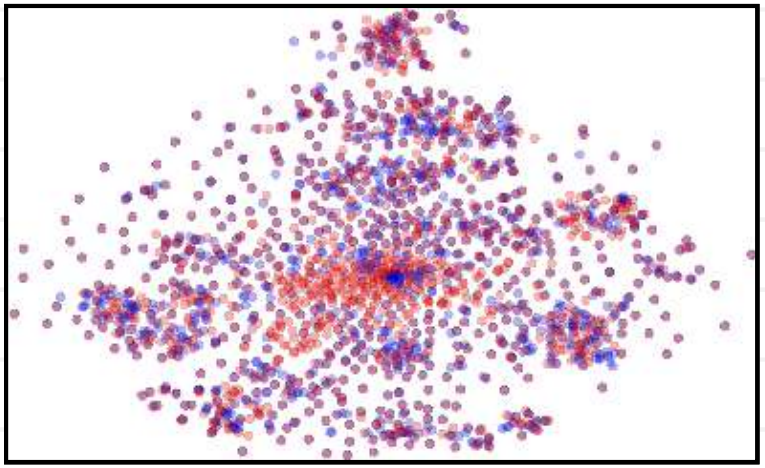


$$d = G_d(\mathbf{f}; \theta_d)$$

Domain classifier:

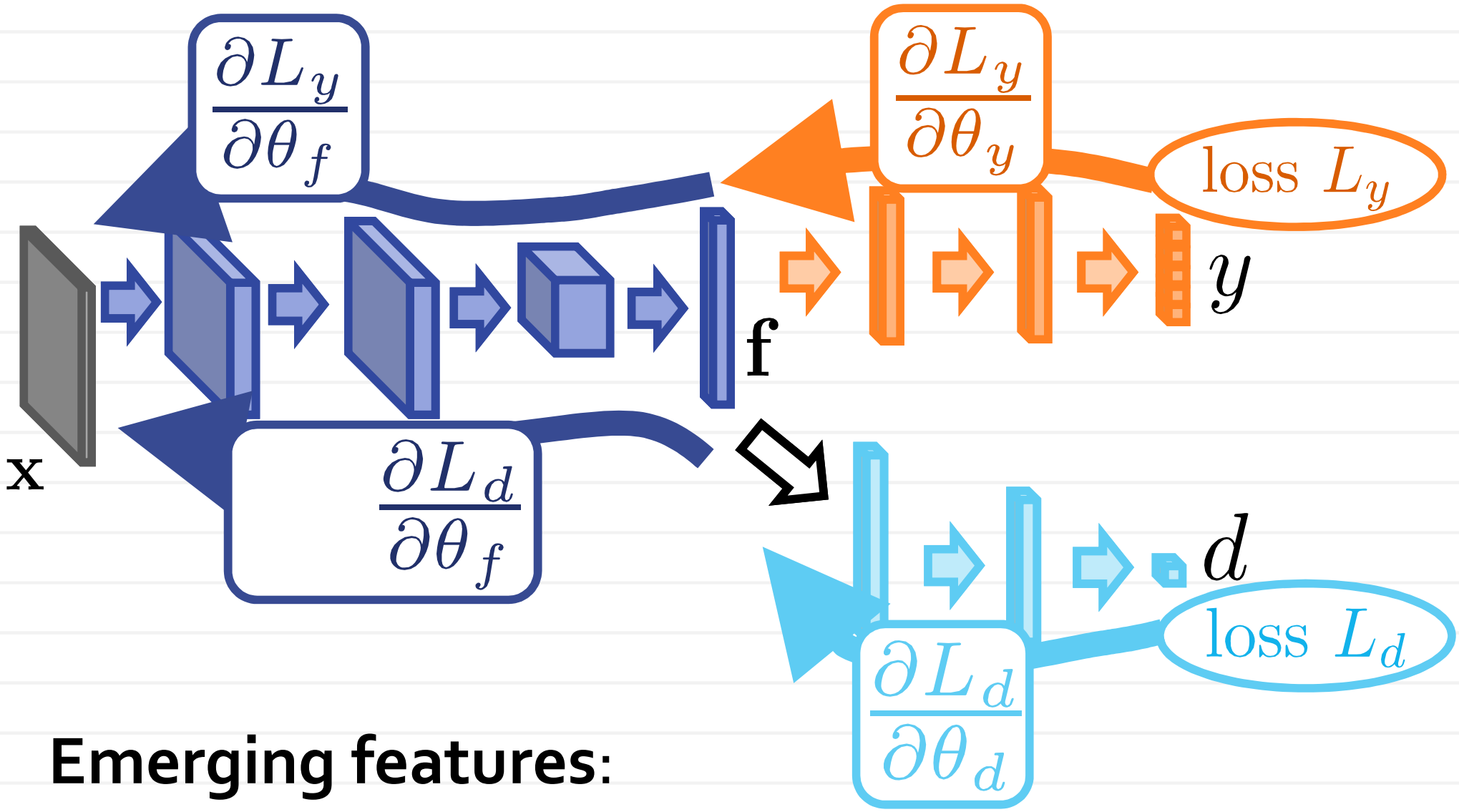


Domain loss low



Domain loss high

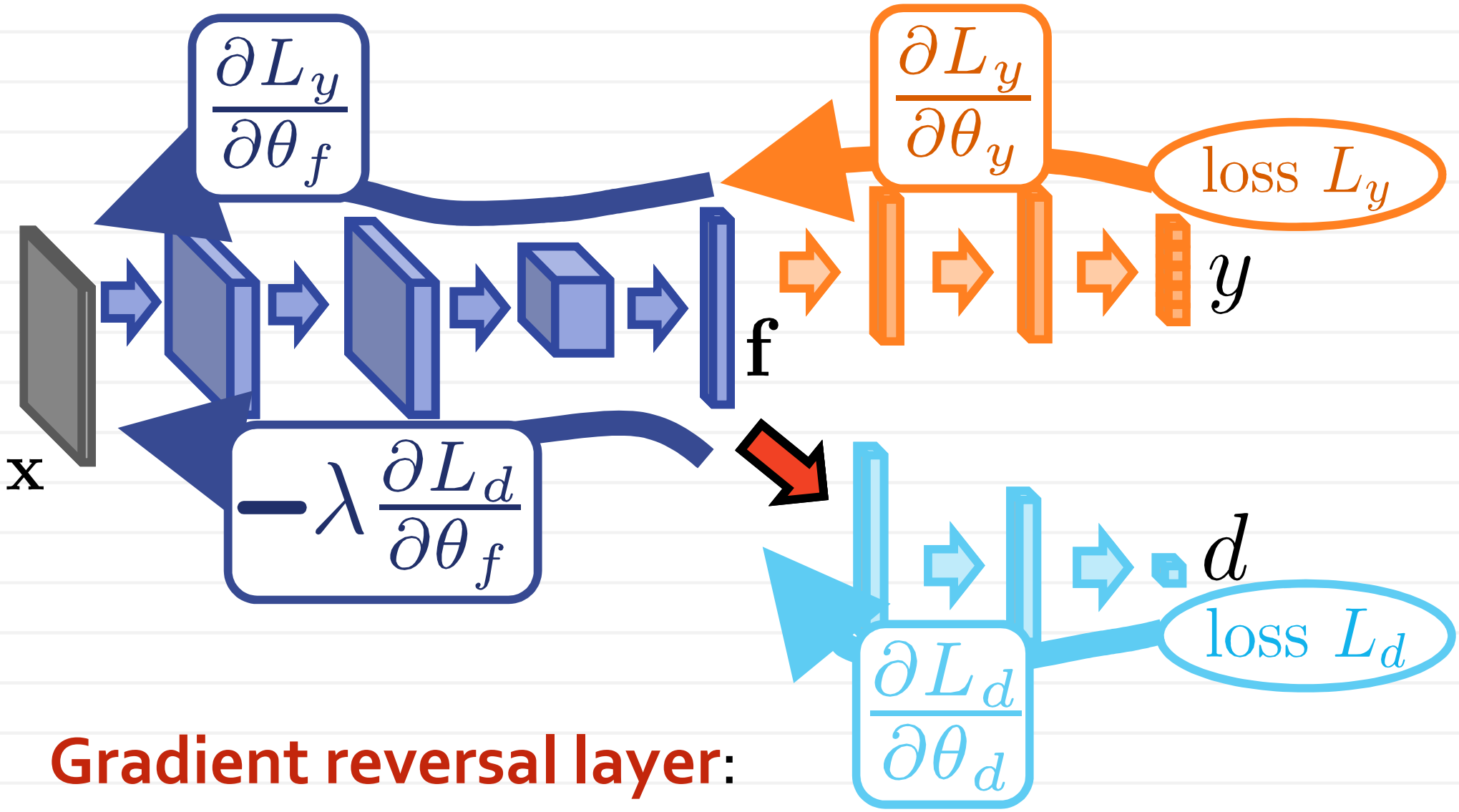
Idea 3: minimizing domain shift



Emerging features:

- Discriminative (good for predicting *y*)
- Domain-discriminative (good for predicting *d*)

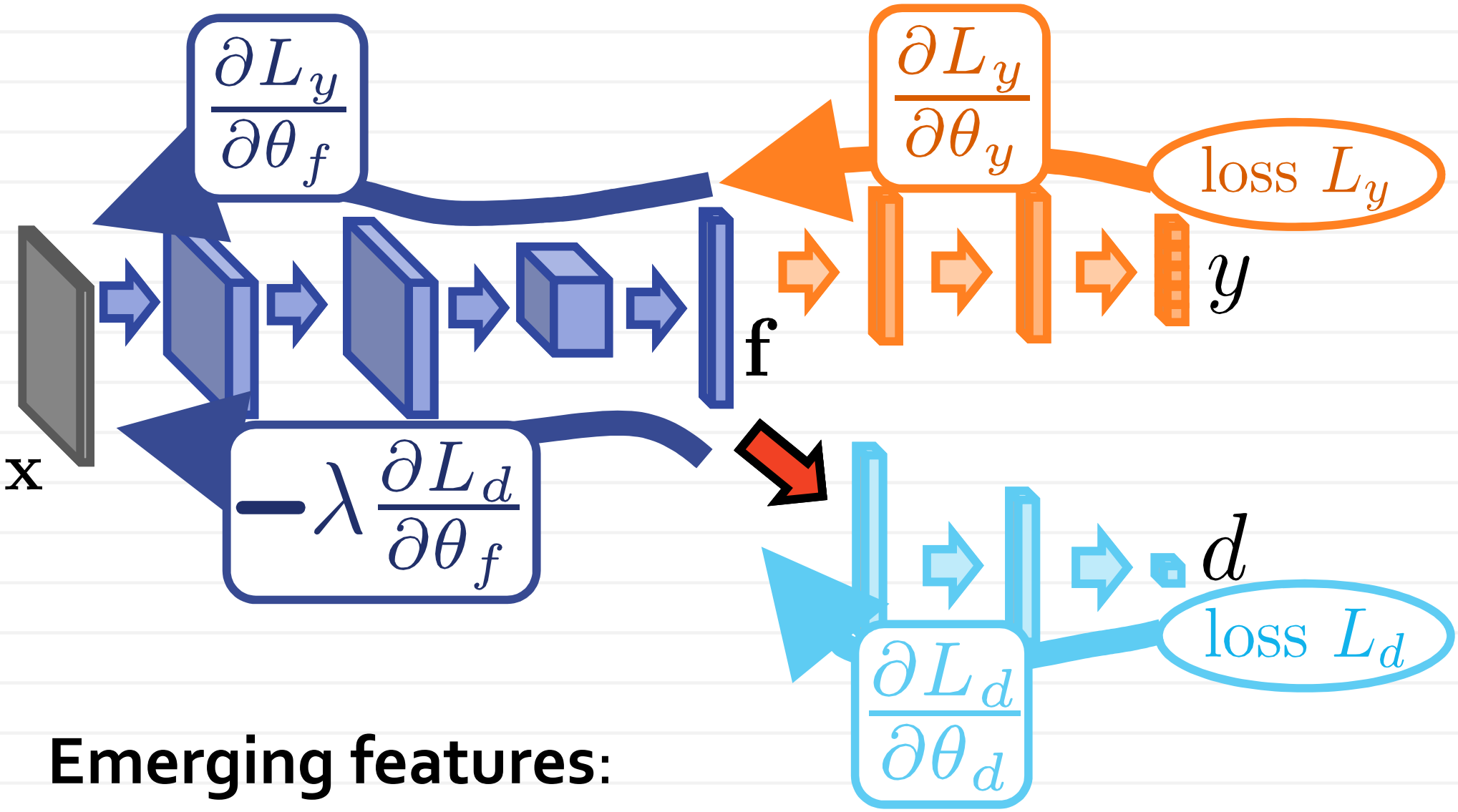
Idea 3: minimizing domain shift



Gradient reversal layer:

- Copies data without change at forwardprop
- Multiplies the gradient by -λ at backprop

Idea 3: minimizing domain shift

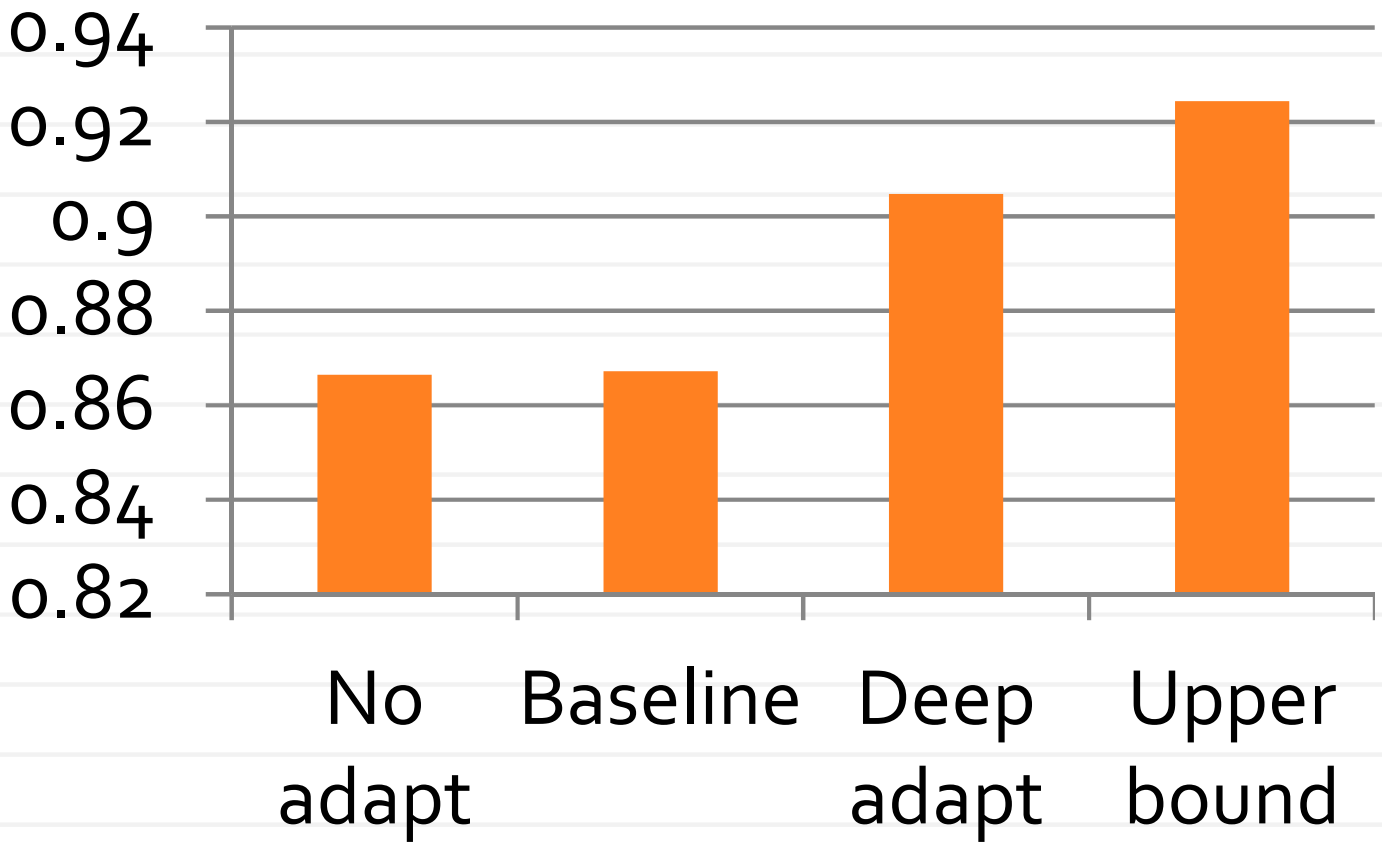


Emerging features:

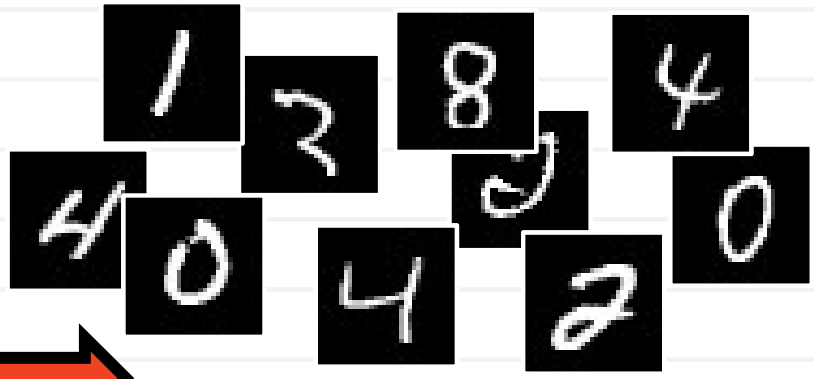
- Discriminative (good for predicting *y*)
- Domain-invariant (not good for predicting *d*)

Example: from synthetic to real

"Windows digits" "House numbers"

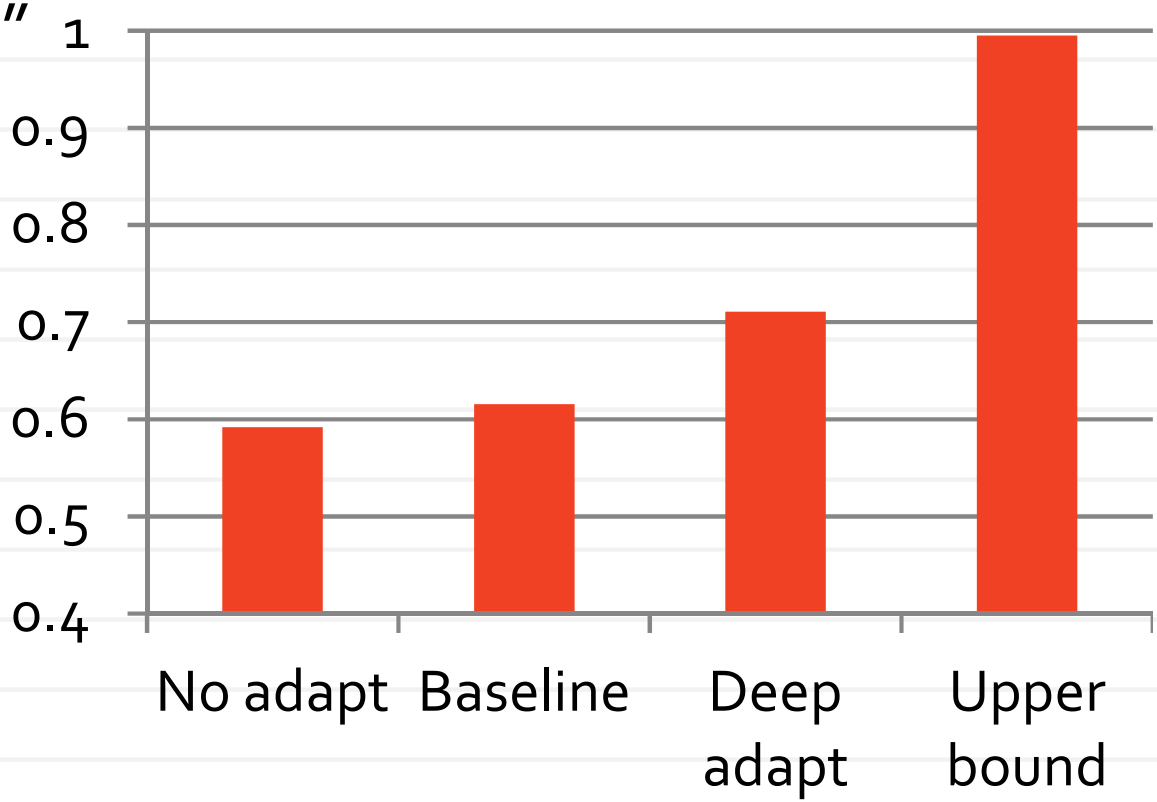


Example: large gap



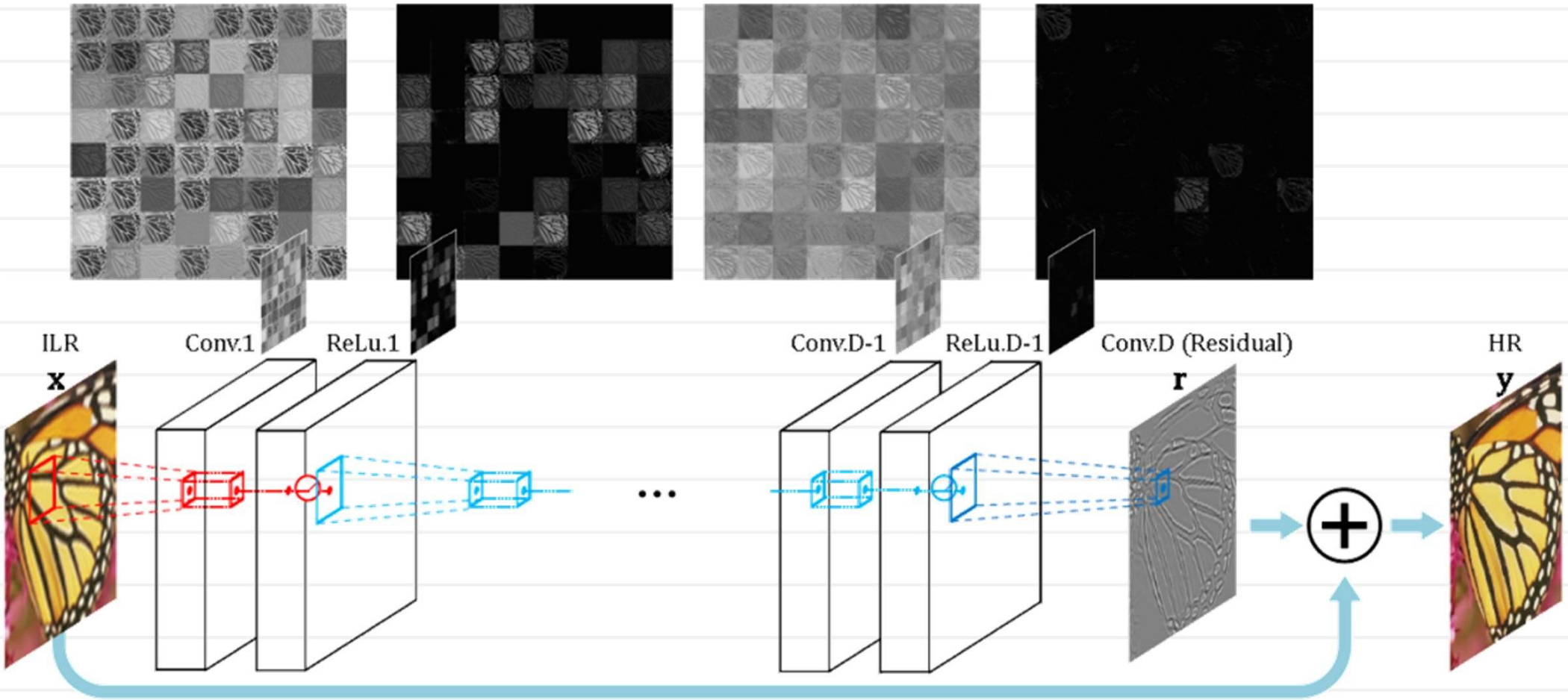
MNIST

"House numbers"



Reverse direction does not work ☹

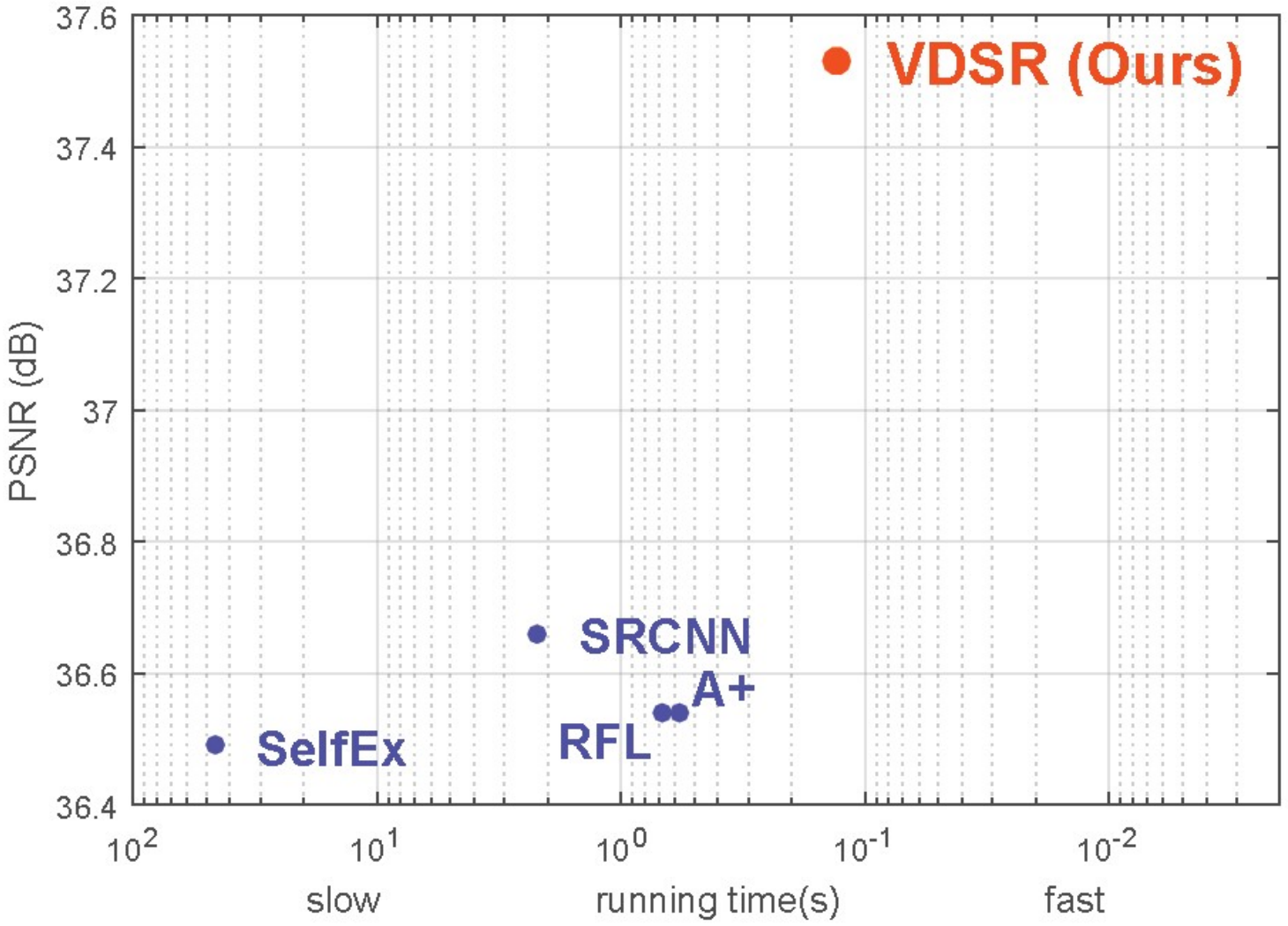
Image superresolution with ConvNets



- 19 layers with 3x3 filters (VGG inspired)

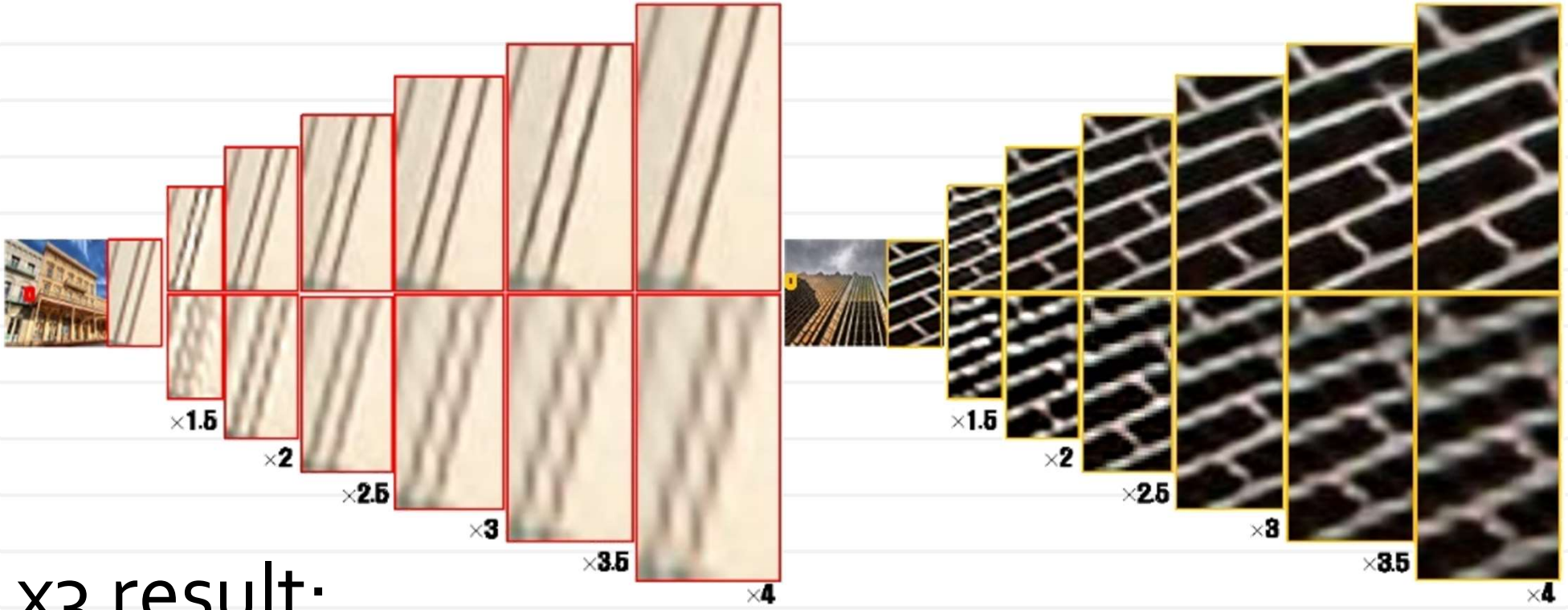
[Kim et al. CVPR16]

Image superresolution with ConvNets

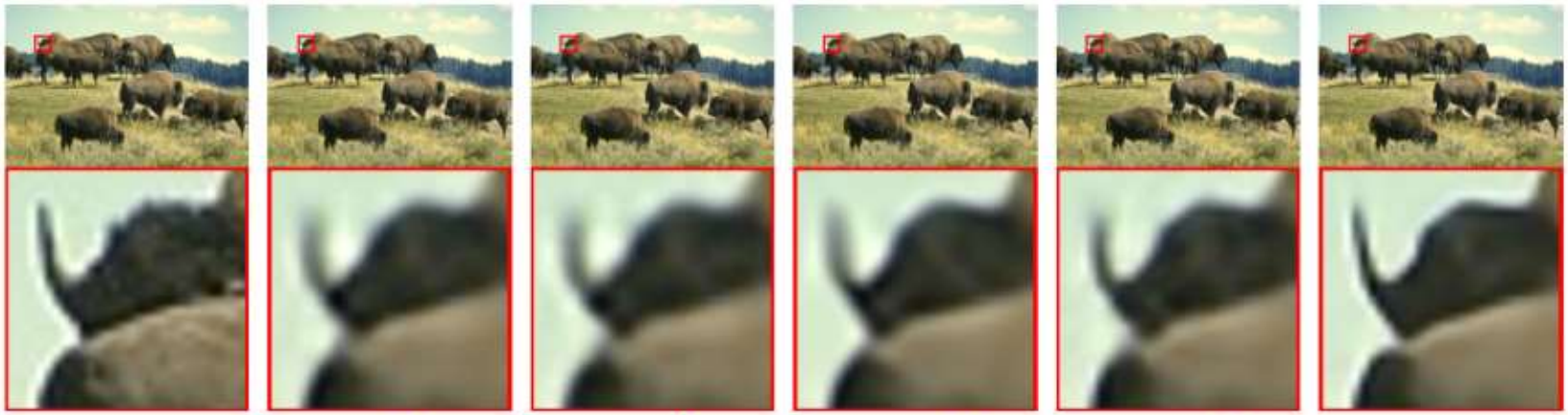


[Kim et al. CVPR16]

Image superresolution with ConvNets



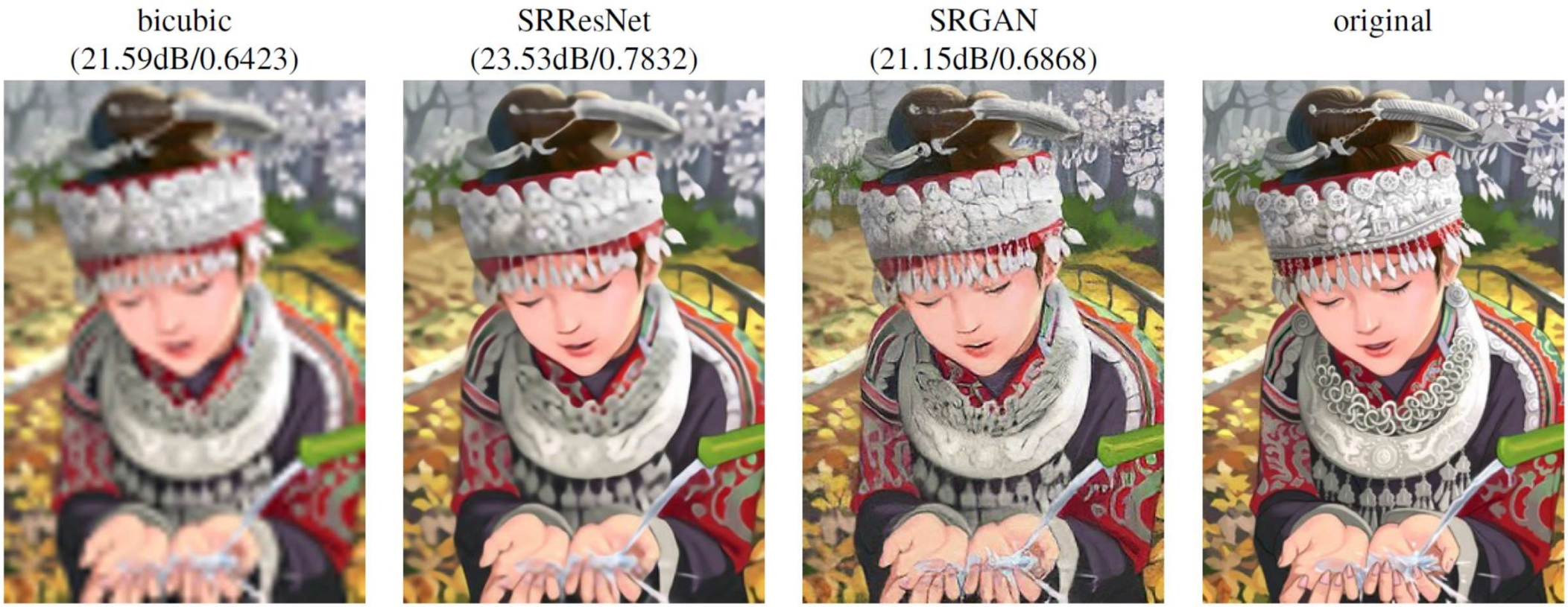
x3 result:



n Ground Truth (PSNR, SSIM) | A+ [22] (27.08, 0.7514) | RFL [18] (27.08, 0.7508) | SelfEx [11] (27.02, 0.7513) | SRCNN [5] (27.16, 0.7545) | VDSR (Ours) (27.32, 0.7606)

[Kim et al. CVPR16]

Superresolution with GAN



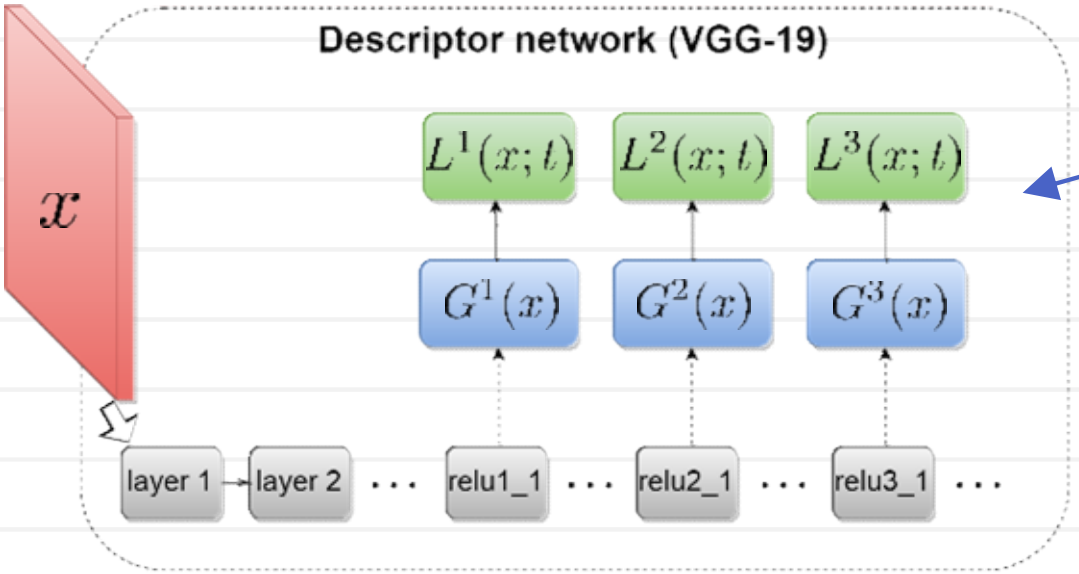
Adding GAN term into the loss

[Ledig et al. 2016]

Superresolution with GAN

[Ledig et al. 2016]

Gatys et al. texture loss



Large pretrained network (VGG-19 network)

Activations at layer *l*: $F^l(t)$

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

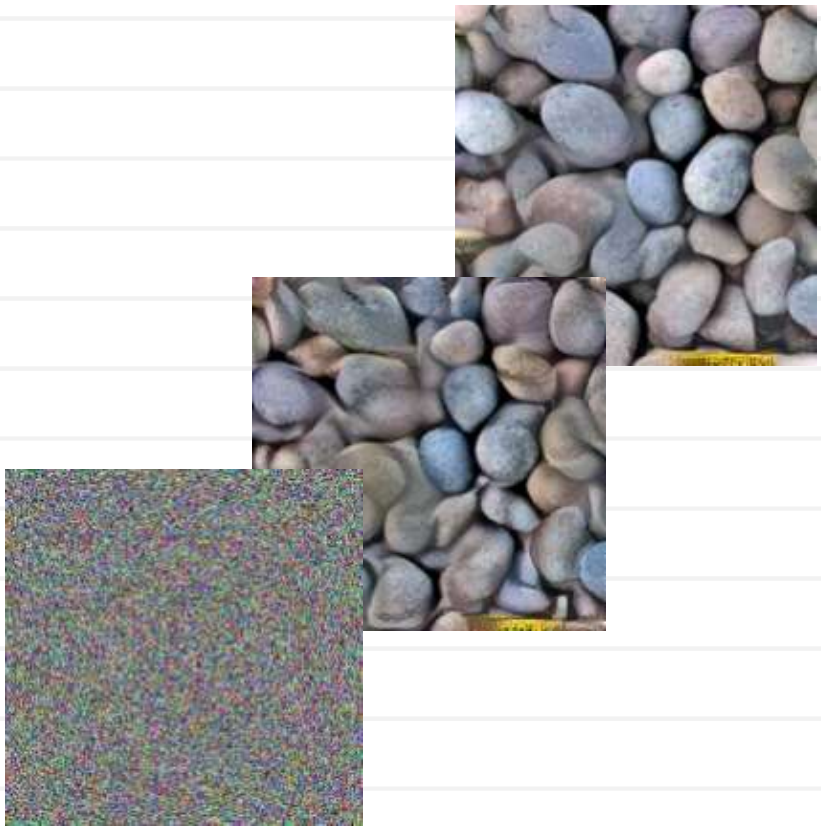
Texture loss (dissimilarity between *t* and *x*):

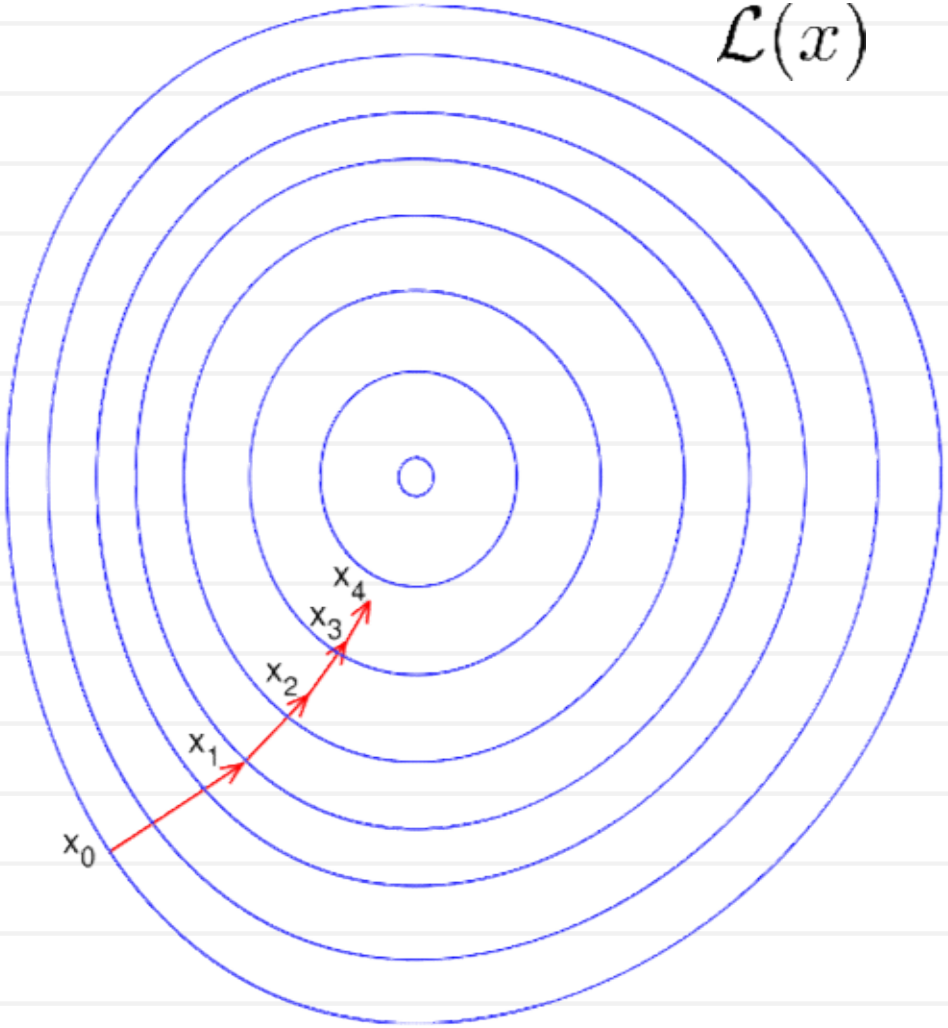
$$\mathcal{L}_{texture}(x;t) = \sum_l ||G^l(t) - G^l(x)||_2^2$$

[Gatys, Ecker, Bethge 2015]

Texture generation by optimization

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





[Gatys, Ecker, Bethge 2015]

Pre-image texture synthesis

[Gatys, Ecker, Bethge 2015]

Texture network

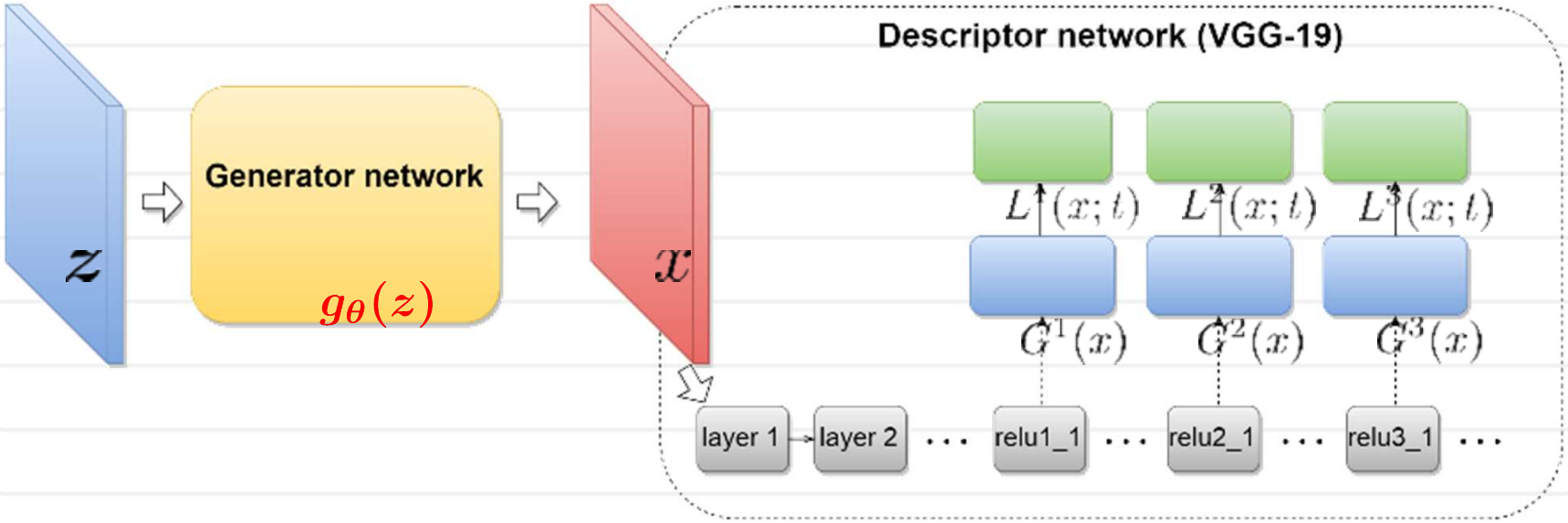


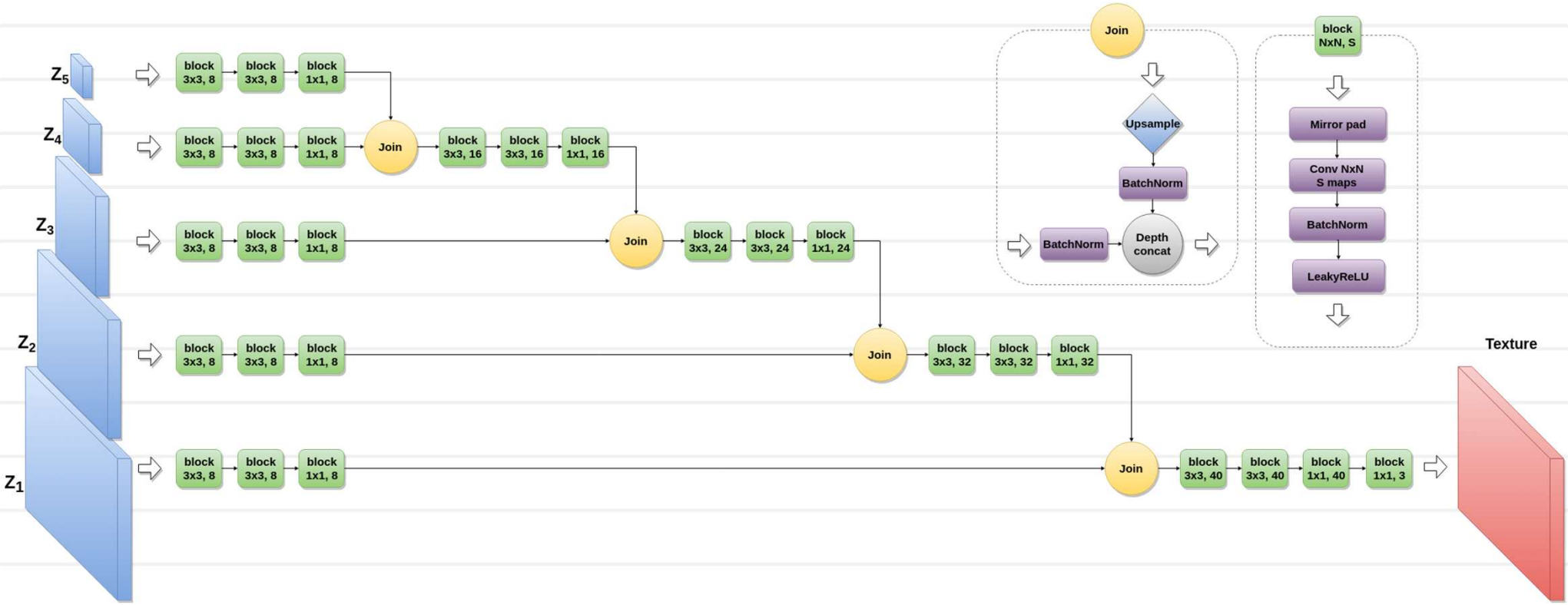
Image generation: $x = g_\theta(\boldsymbol{z}), \quad \boldsymbol{z} \sim U(0,1)$

Optimization task becomes:

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

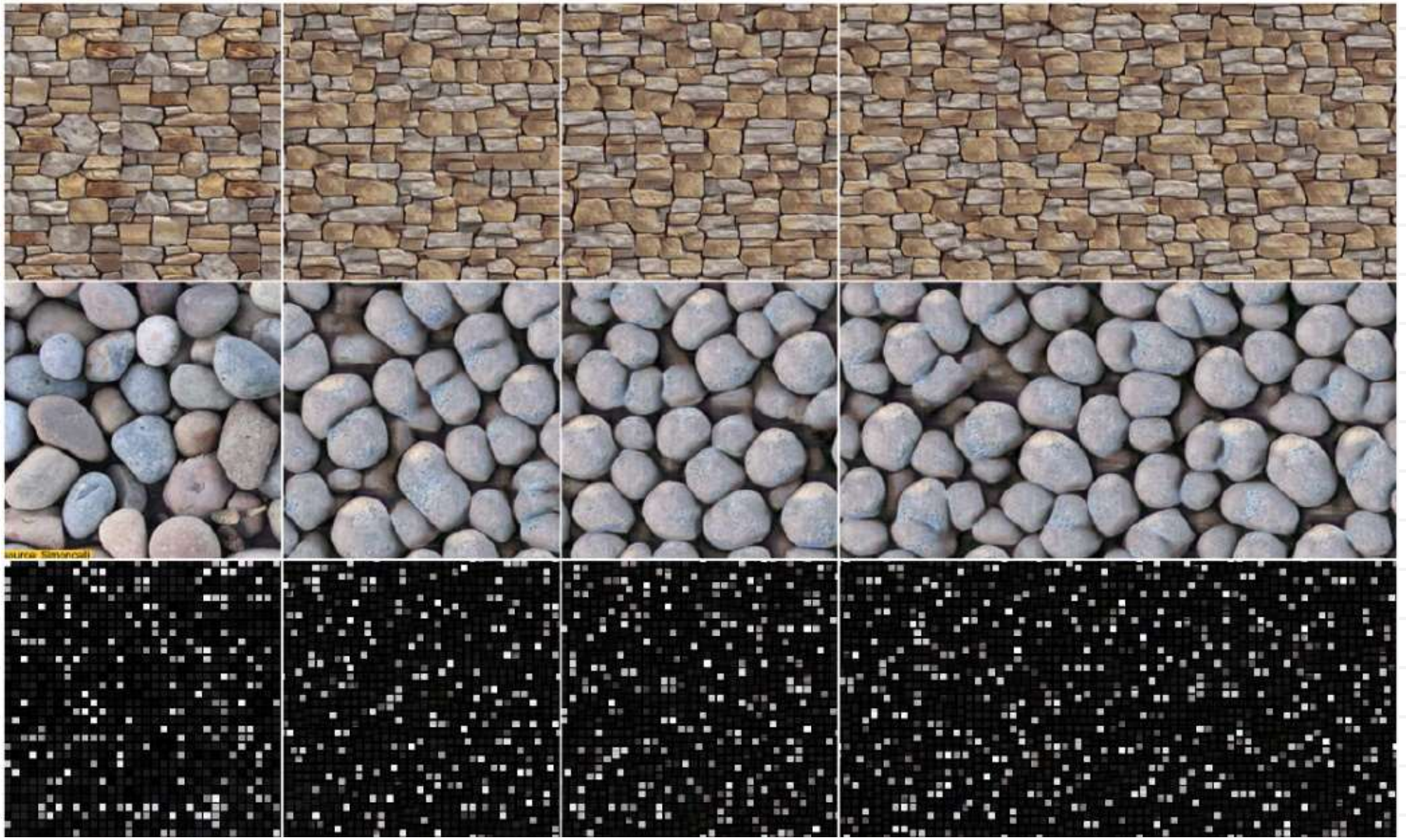
$$\theta^{k+1} = \theta^k - \alpha \frac{\partial \mathcal{L}(g_\theta(z);t)}{\partial \theta}$$

Generator networks details



- Multi-scale approach simplifies learning
- Looks scary, but still feedforward and fully convolutional
- Bias terms removed to avoid overfitting

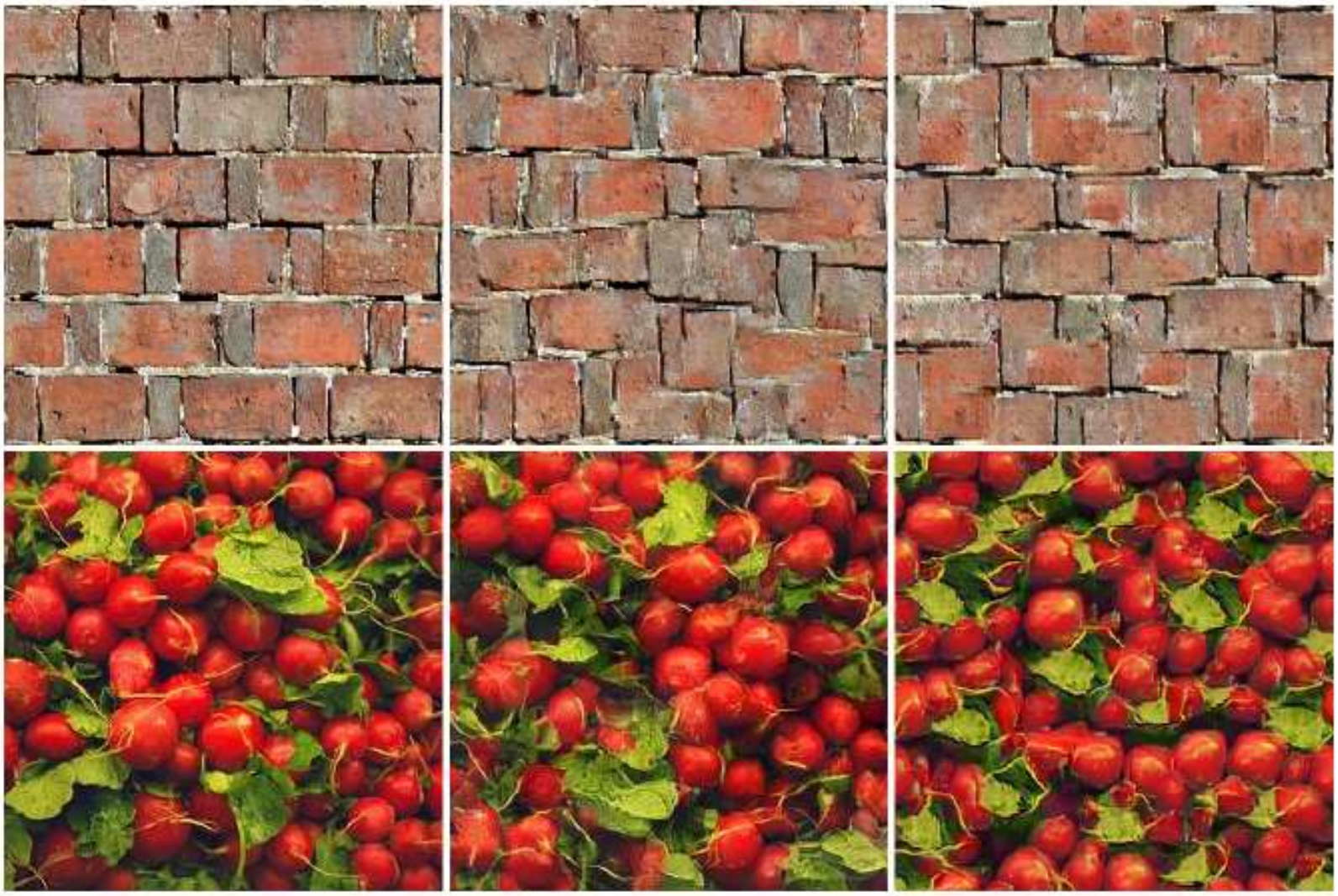
Feed-forward texture synthesis



sample

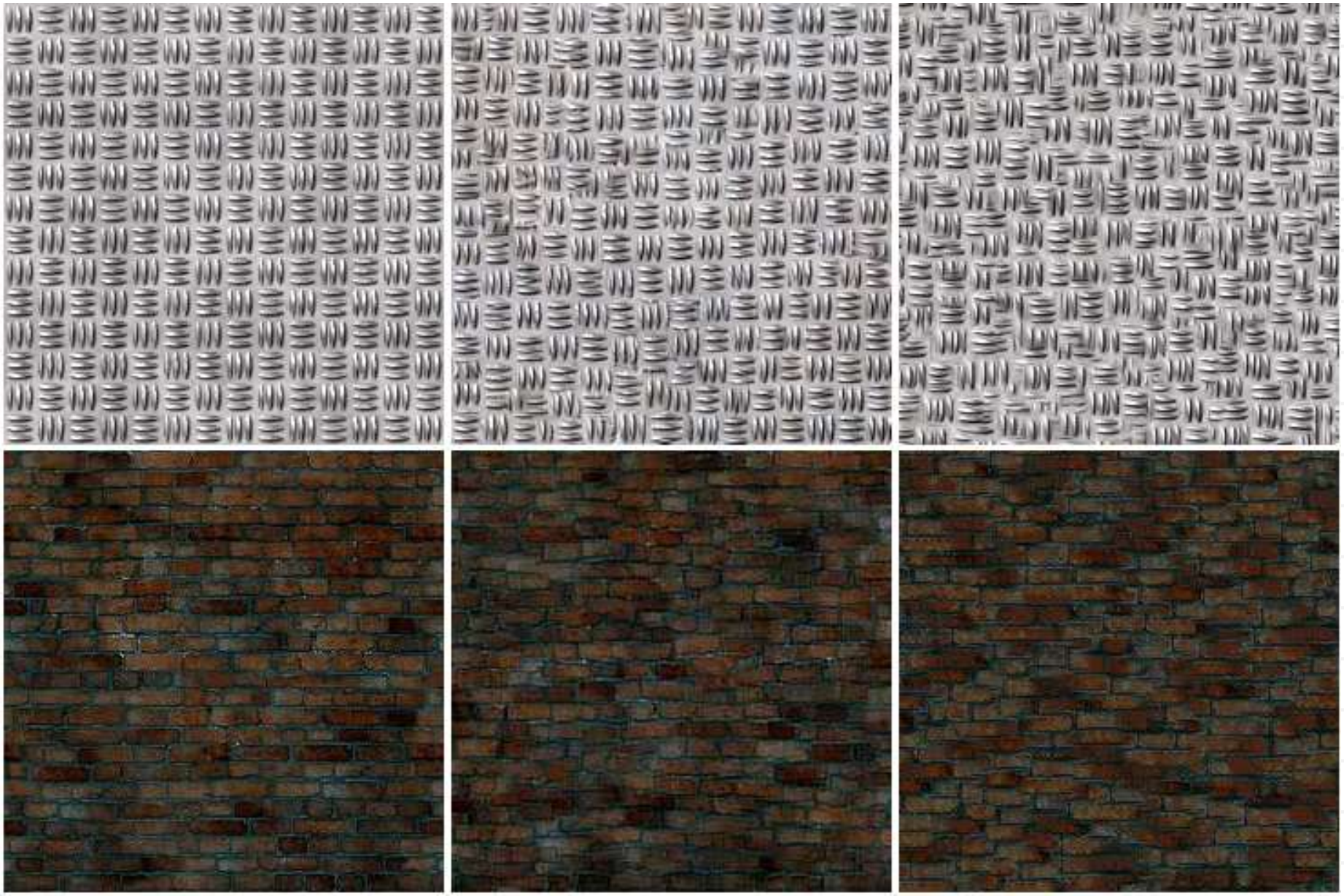
Results

Pre-image vs. feed-forward



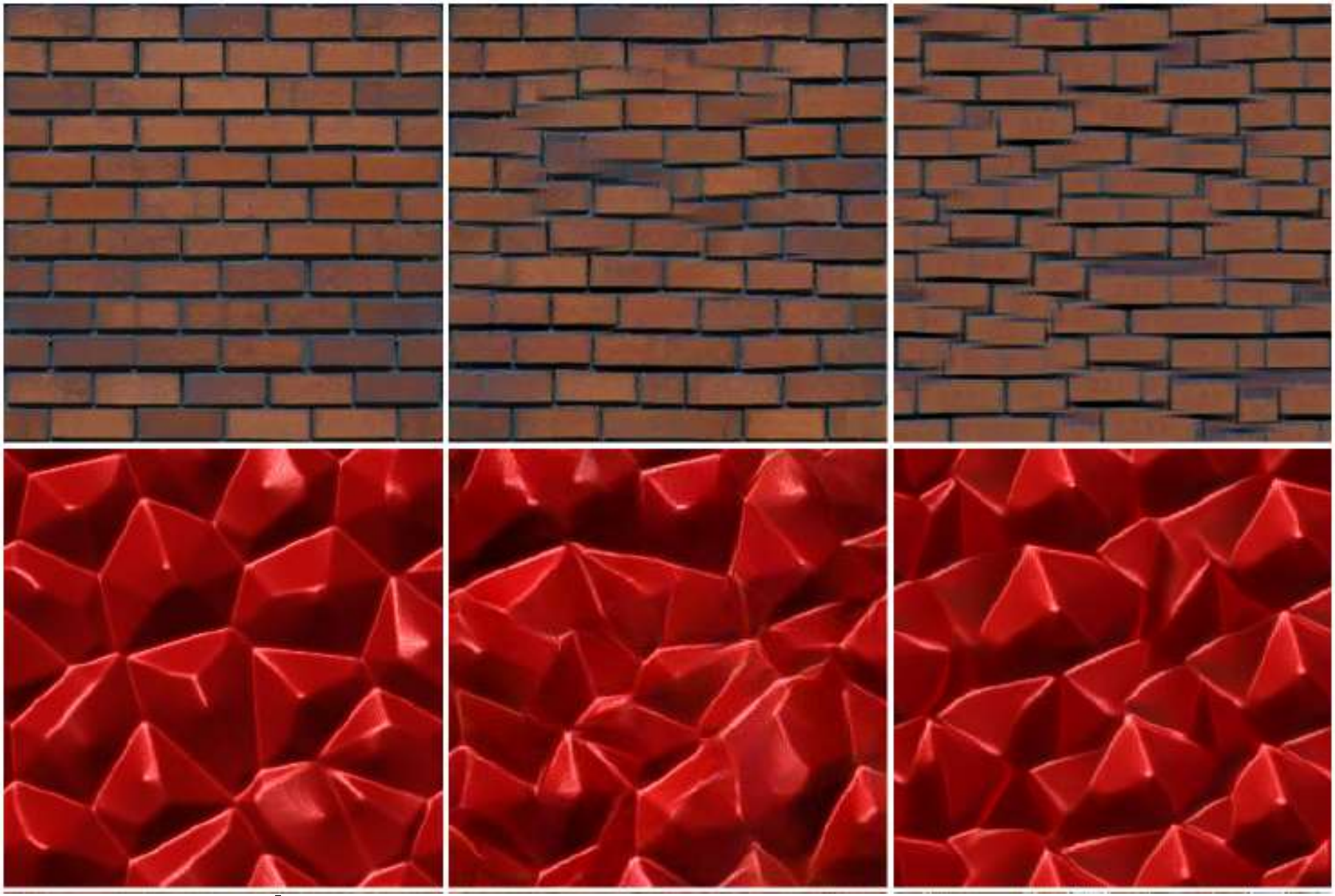
sample Optimized Feed-forward

Pre-image vs. feed-forward

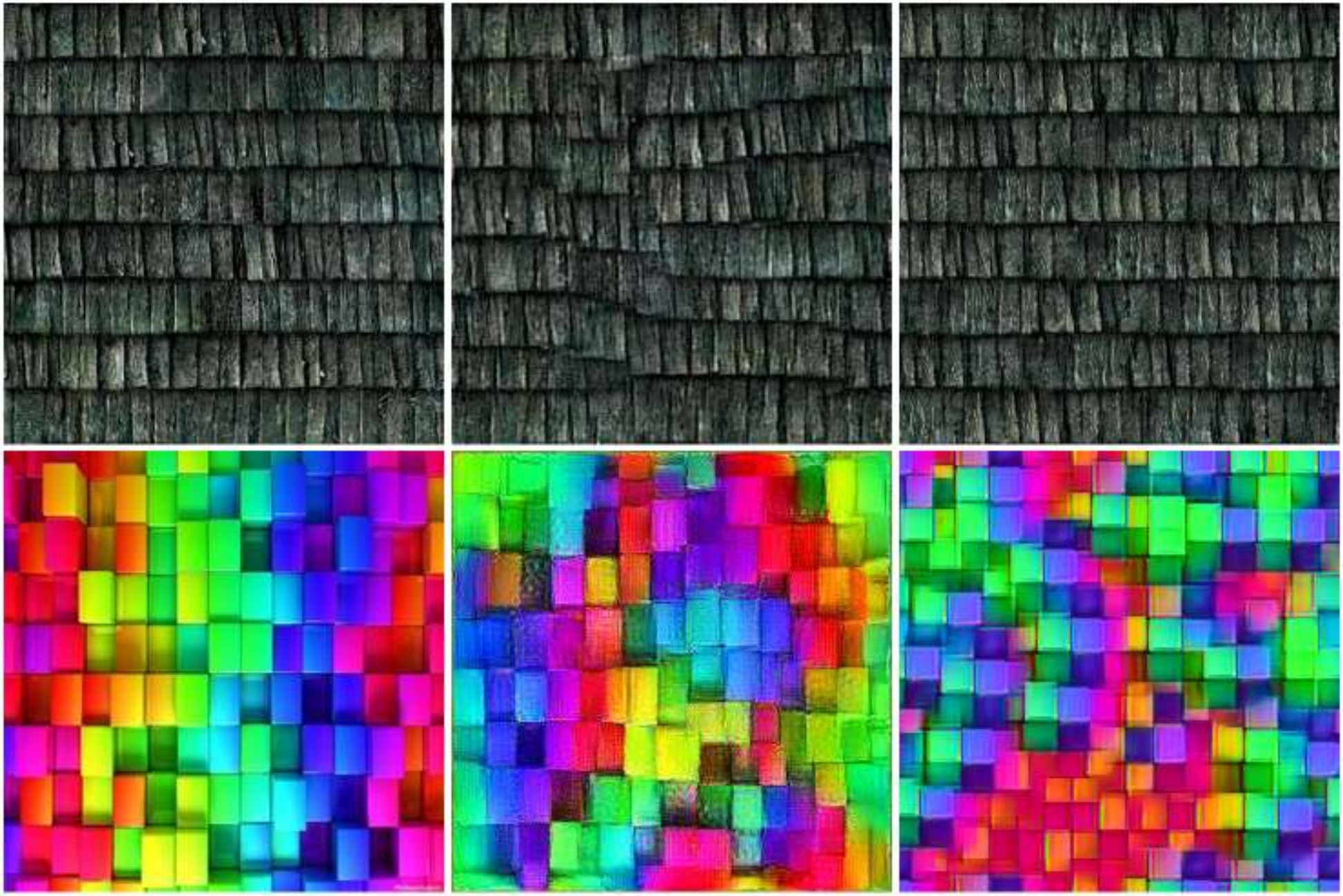


sample Optimized Feed-forward

Pre-image vs. feed-forward

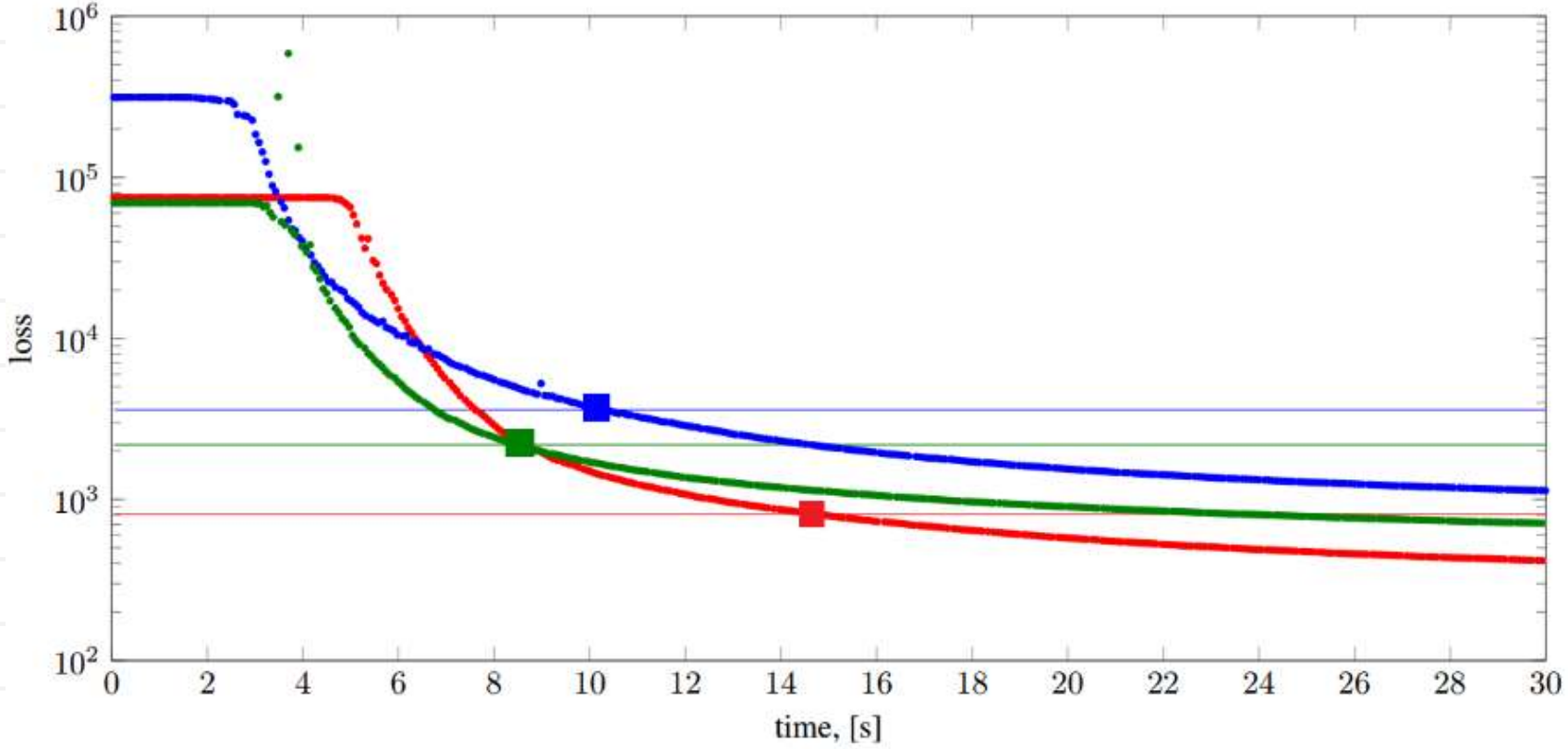


Pre-image vs. feed-forward



sample Optimized Feed-forward

Quantitative evaluation



- Horizontal lines: average sample loss
  - **0.06 second** to generate sample
- Dotted lines: *optimization-based* loss as a function of time
  - **10 seconds** to achieve the same loss values

"Style" network

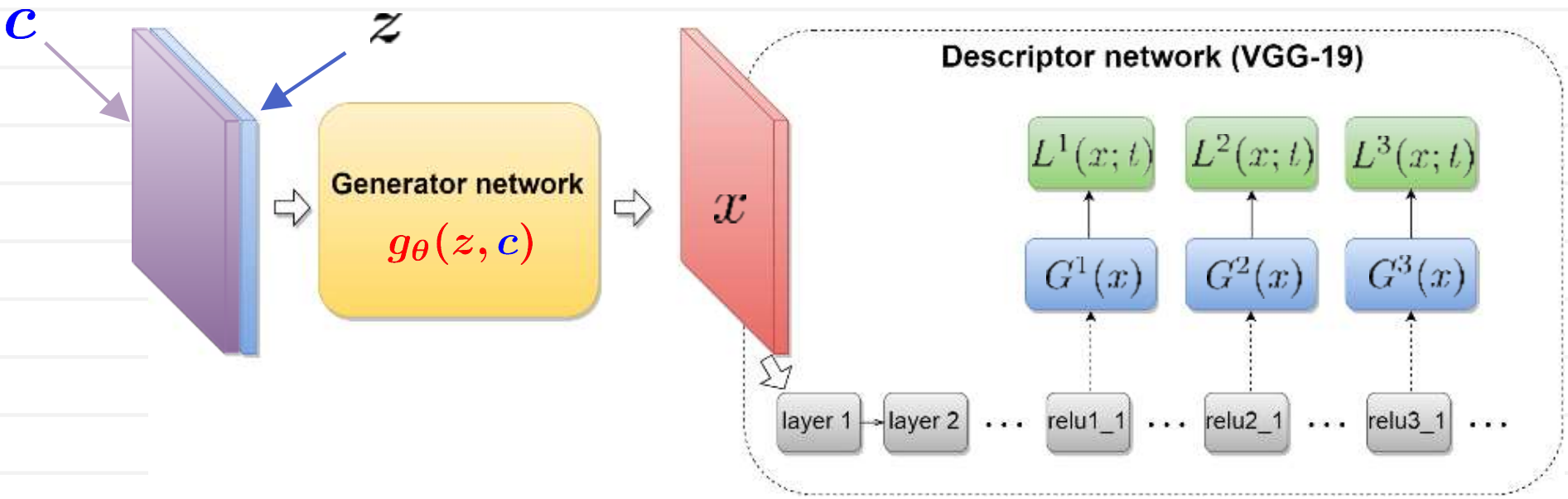


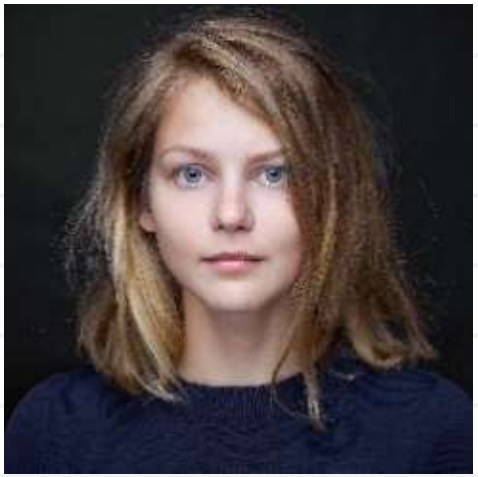
Image generation:

$$x = g_\theta(\boldsymbol{z}, \boldsymbol{c}), \quad \boldsymbol{z} \sim U(0,1)$$

Optimization task:

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

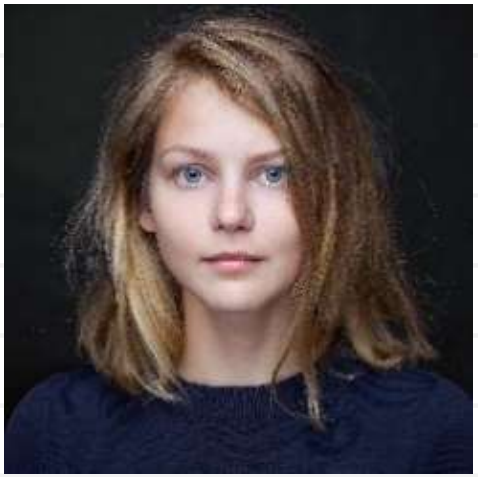
Feed-forward stylization

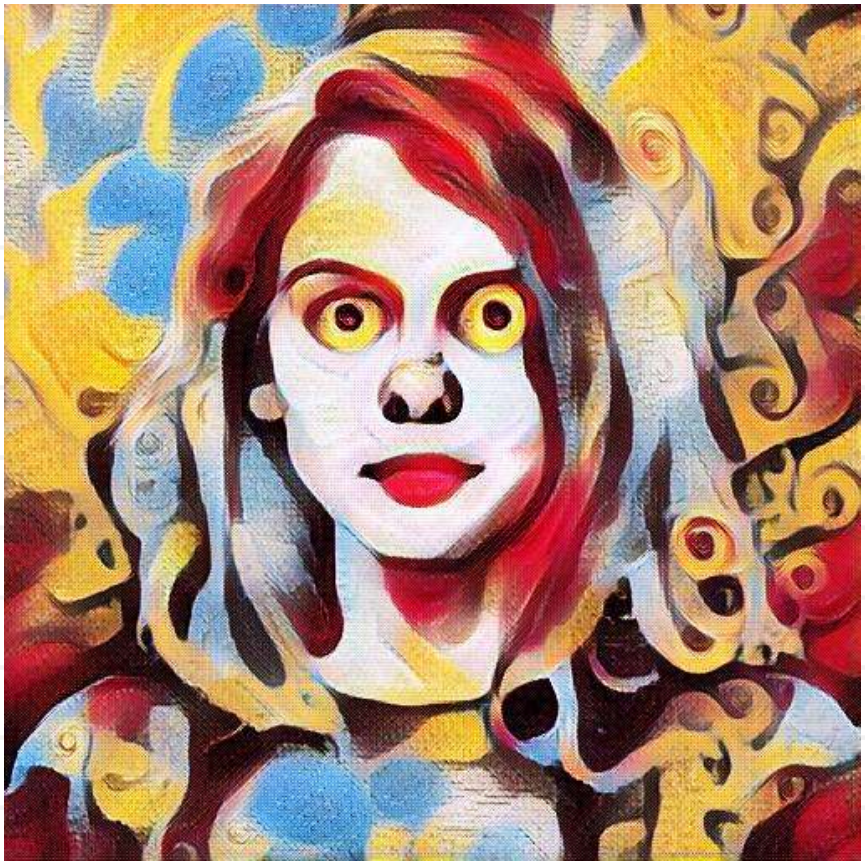


feedforward

optimization

Feed-forward stylization

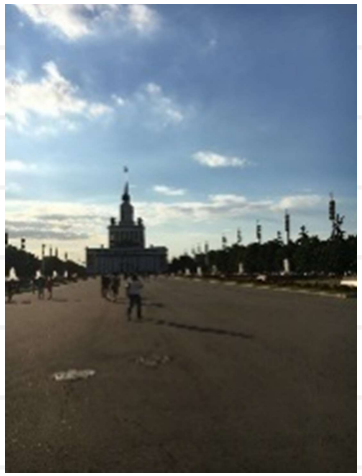




feedforward

optimization

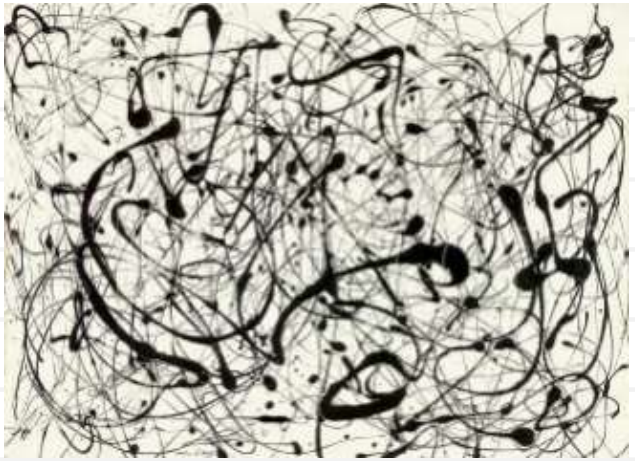
Feed-forward stylization

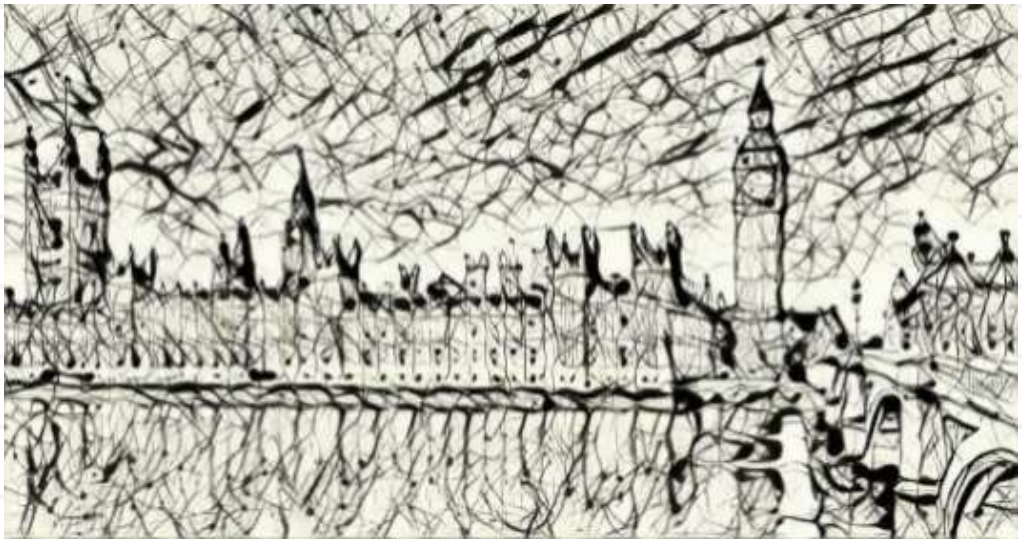


feedforward

optimization

Feed-forward stylization





feedforward

optimization

Recap

- *Synthesizing* realistic images is hard
- Better loss functions are key
- With good loss functions, we can train feedforward generators
- Big promise/hype about GANs, some fundamental challenges remain

Bibliography

Alexey Dosovitskiy, Jost Tobias Springenberg, Thomas Brox:
Learning to generate chairs with convolutional neural networks. CVPR 2015: 1538-1546

Alexey Dosovitskiy, Thomas Brox:
Inverting Convolutional Networks with Convolutional Networks. CoRR abs/1506.02753 (2015)

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, Yoshua Bengio:
Generative Adversarial Nets. NIPS 2014

Peter J. Burt, Edward H. Adelson:
A Multiresolution Spline with Application to Image Mosaics. ACM Trans. Graph. 2(4): 217-236 (1983)

Emily L. Denton, Soumith Chintala, Arthur Szlam, Robert Fergus:
Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. ICLR 2016

Bibliography

Burt, Peter J., and Edward H. Adelson. "The Laplacian pyramid as a compact image code." Communications, IEEE Transactions on 31.4 (1983): 532-540.

J Kim, JK Lee, KM Lee, Accurate Image Super-Resolution Using Very Deep Convolutional Networks, CVPR 2016

Leon A. Gatys, Alexander S. Ecker, Matthias Bethge:
Texture Synthesis Using Convolutional Neural Networks. NIPS 2015: 262-270

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

Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, Victor S. Lempitsky:
Texture Networks: Feed-forward Synthesis of Textures and Stylized Images.
CoRR abs/1603.03417 (2016)