AI6126 Advanced Computer Vision

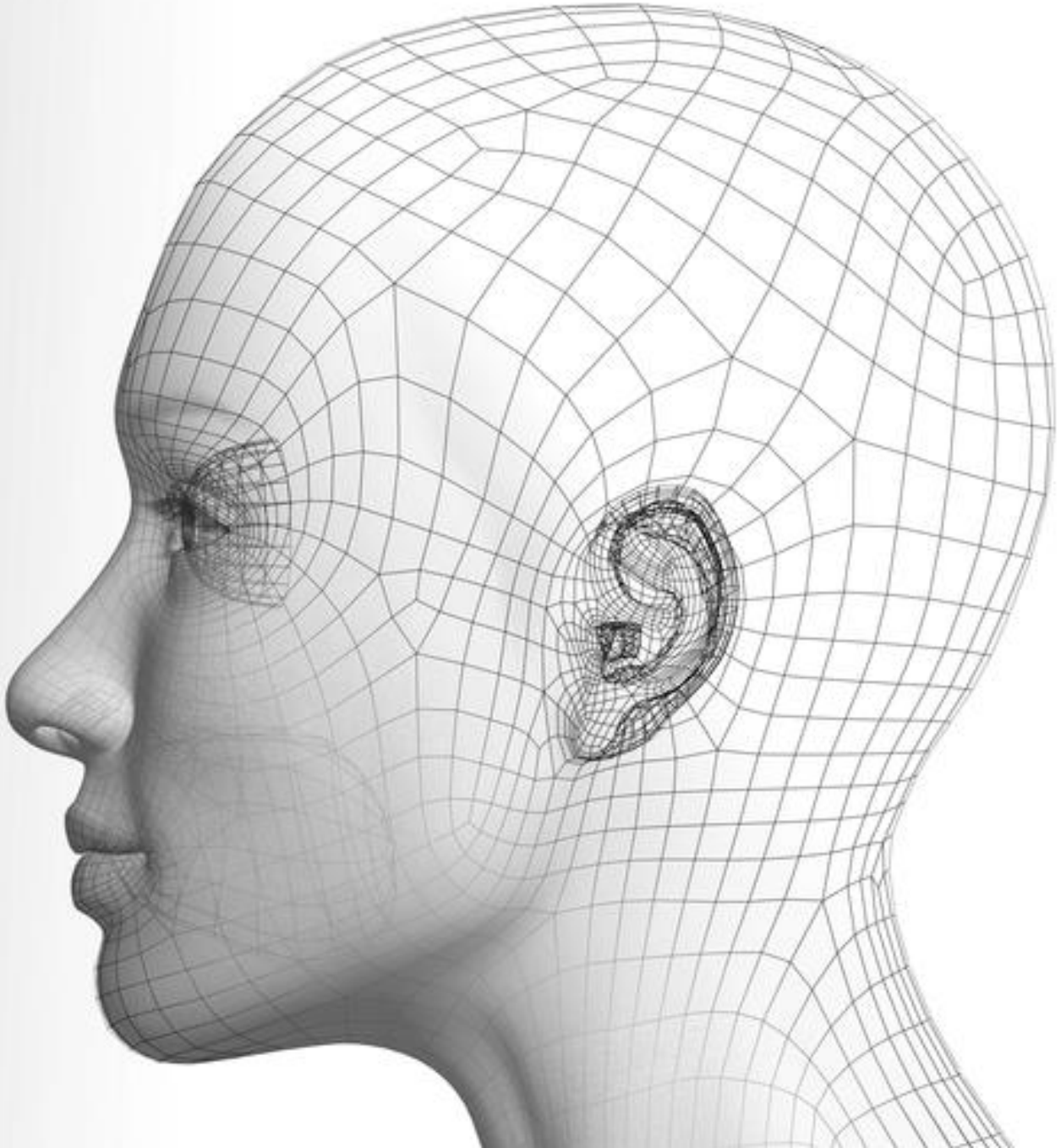
Last update: 23 March 2022

Image Editing

Shuai Yang

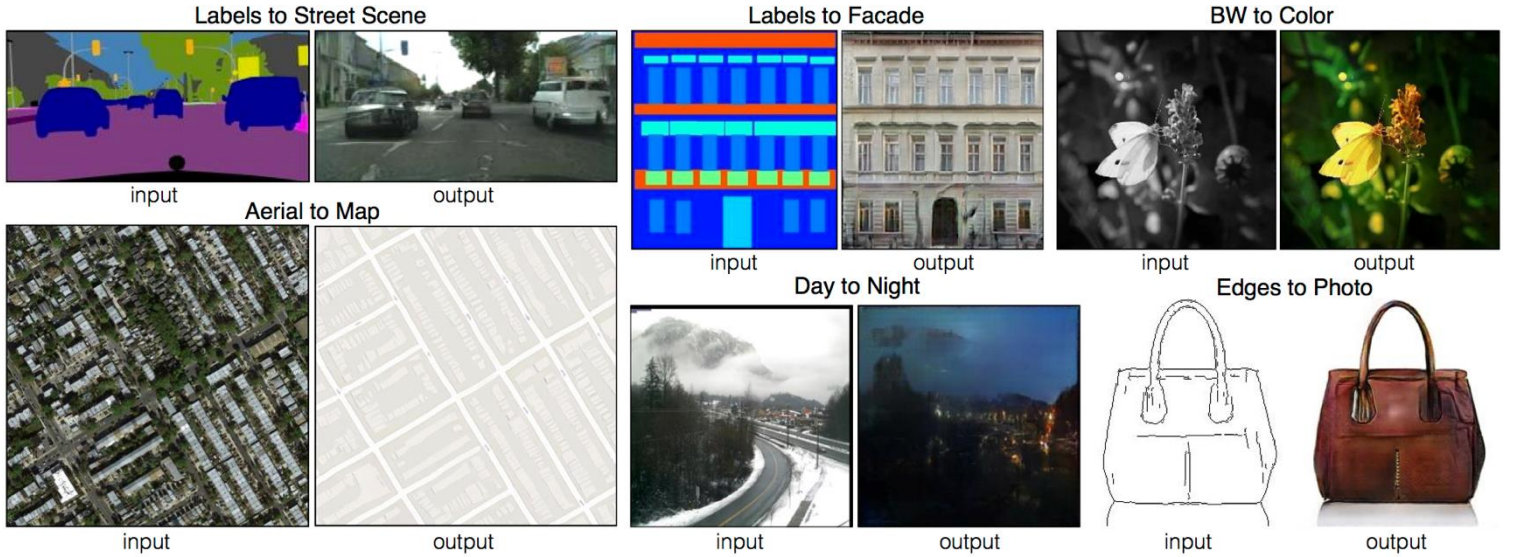
杨 帅

https://williamyang1991.github.io/



Outline

- Image-to-image translation
  - Pix2Pix
  - CycleGAN
- Style transfer
- Exploiting GAN prior



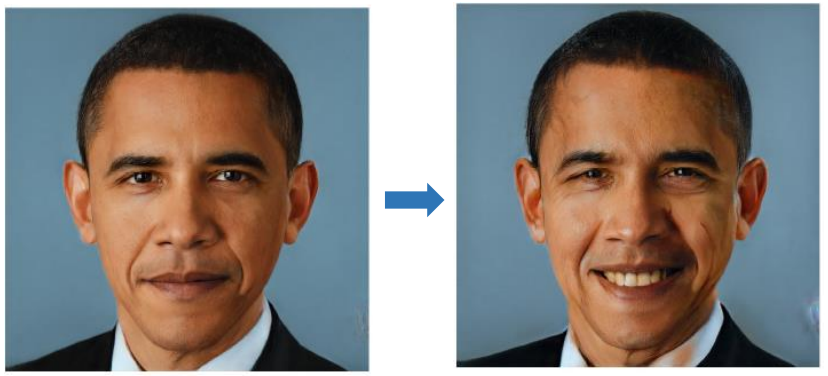
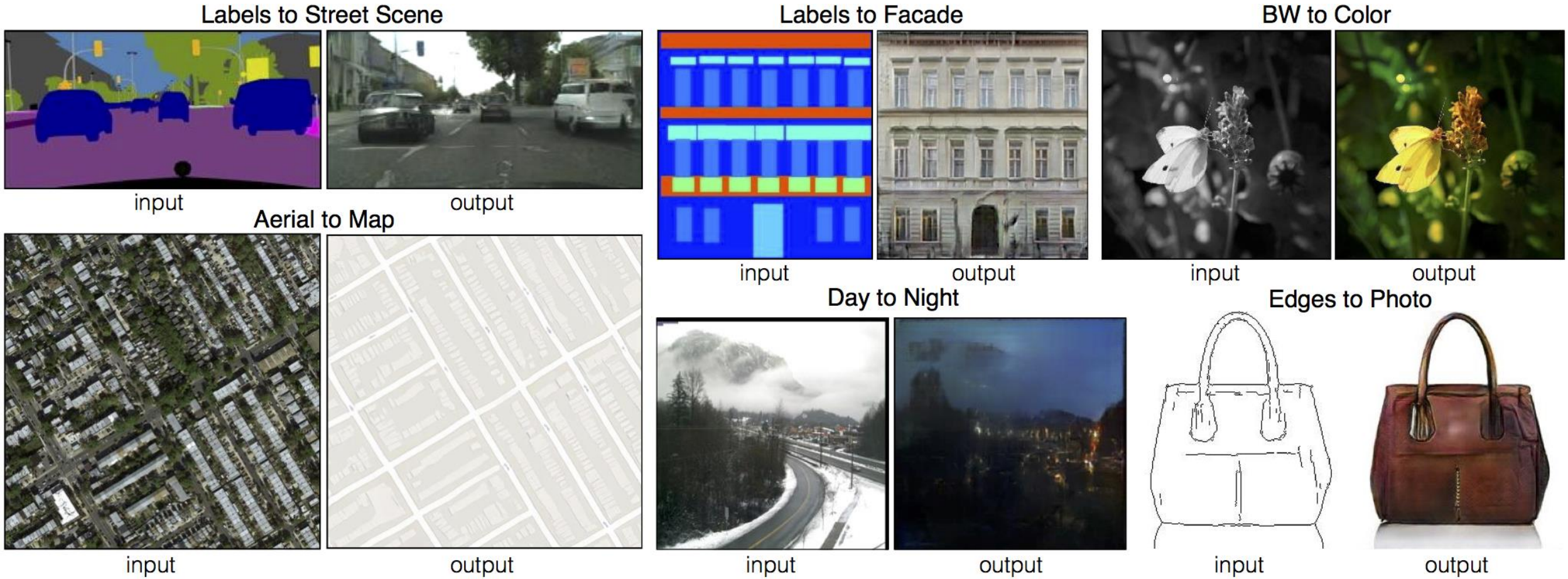


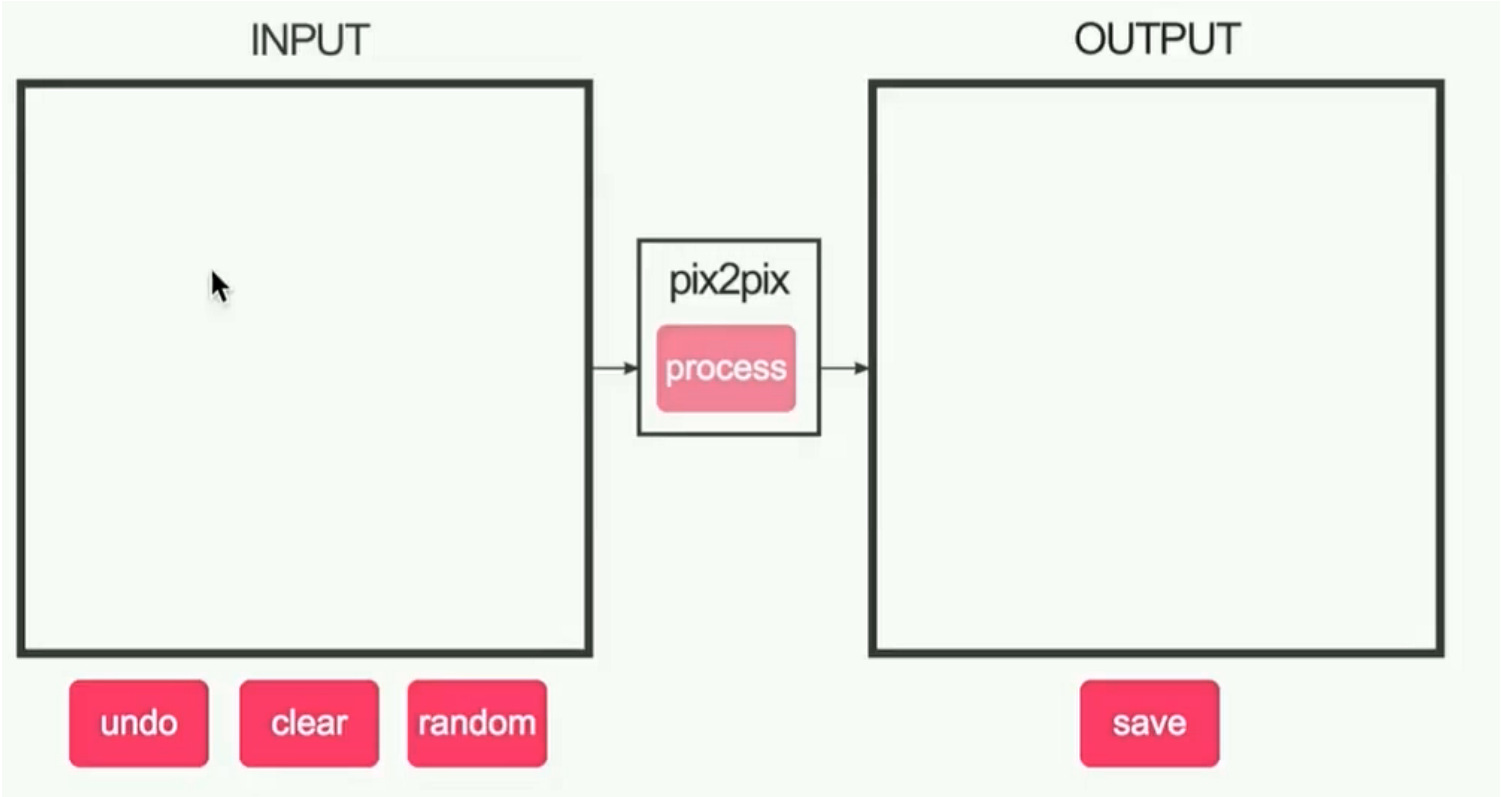
Image-to-Image Translation

Image-to-Image translation with Pix2Pix



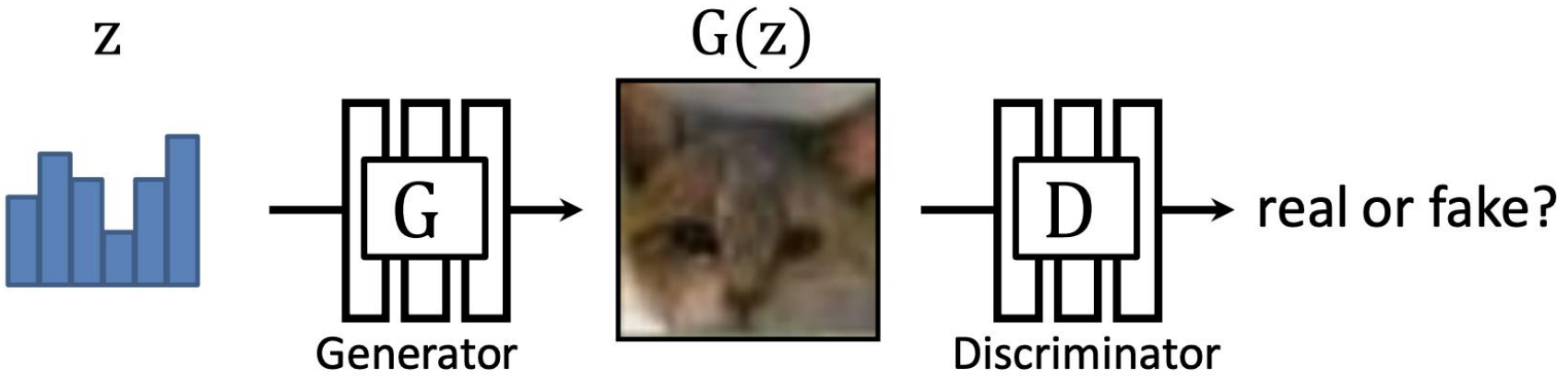
Isola et al., Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

Image-to-Image translation with Pix2Pix



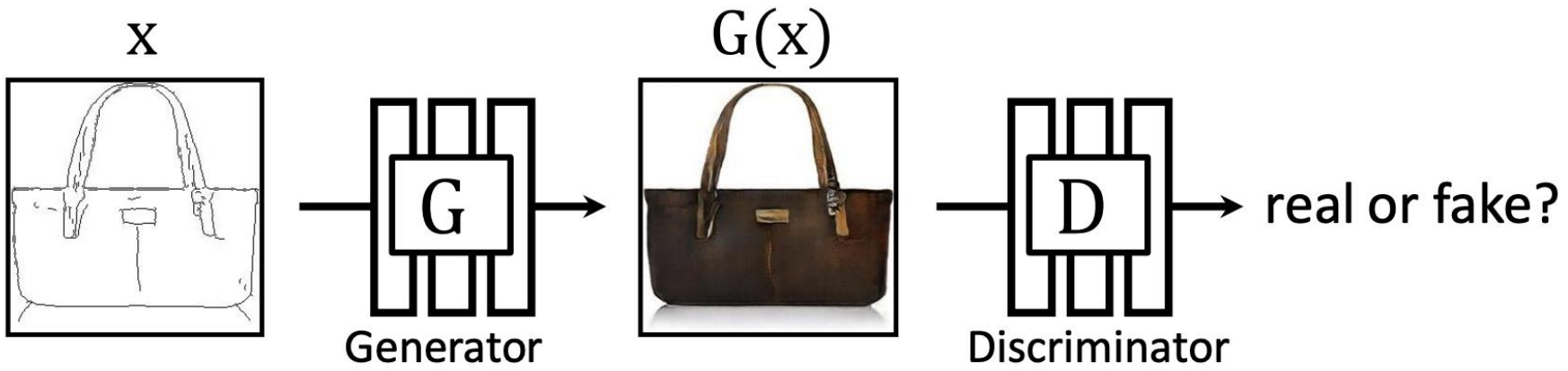
Interactive demo: https://affinelayer.com/pixsrv/

Conventional GAN



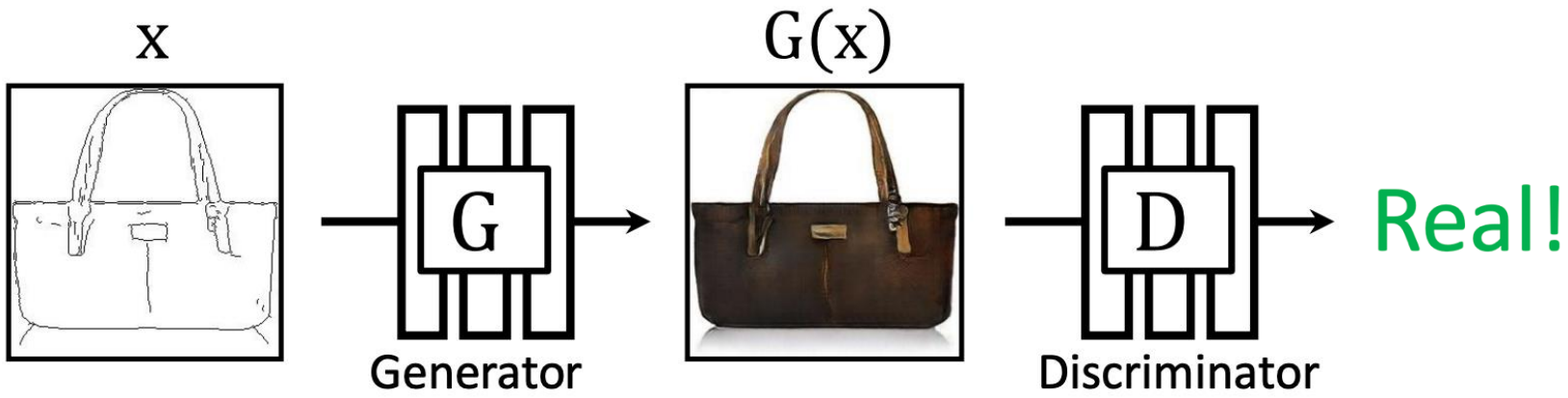
$$\min_{G}\max_{D}\left(E_{y\sim p_{data}}[\log D(y)]+E_{z\sim p(z)}\left[\log\left(1-D(G(z))\right)\right]\right)$$

Image-to-image translation



$$\min_{G} \max_{D} \left(E_{y \sim p_{data}} [\log D(y)] + E_{x \sim p(x)} \left[\log \left(1 - D(G(x)) \right) \right] \right)$$

Image-to-image translation

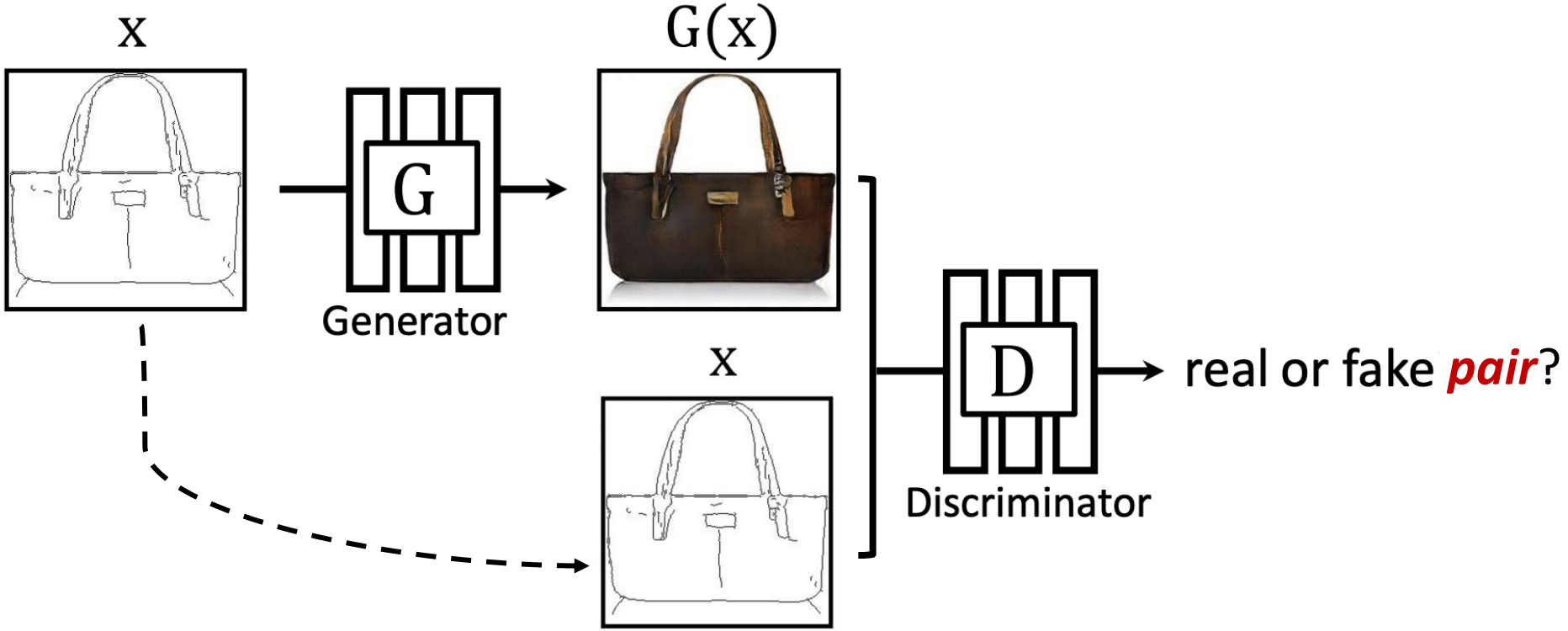


$$\min_{G} \max_{D} \left(E_{y \sim p_{data}} [\log D(y)] + E_{x \sim p(x)} \left[\log \left(1 - D(G(x)) \right) \right] \right)$$

Image-to-image translation

$$\min_{G} \max_{D} \left(E_{y \sim p_{data}} [\log D(y)] + E_{x \sim p(x)} \left[\log \left(1 - D(G(x)) \right) \right] \right)$$

Image-to-image translation with Pix2Pix



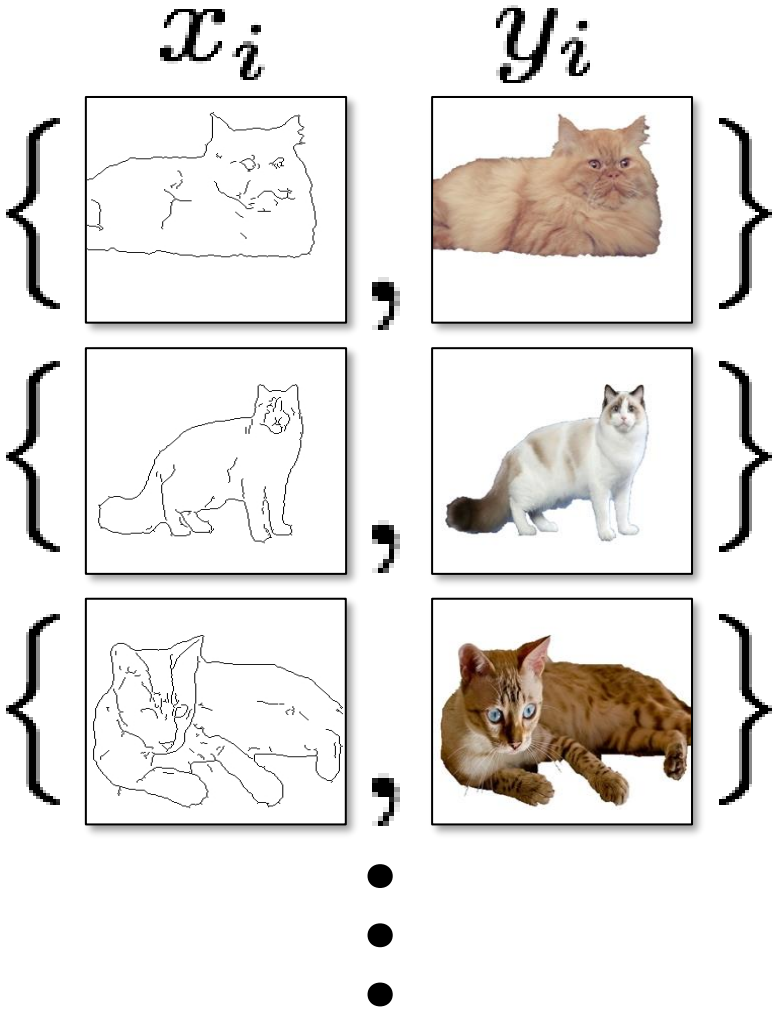
$$\min_{G}\max_{D}\left(E_{x,y}[\log D(x,y)] + E_{x}\left[\log\left(1 - D(x, G(x))\right)\right]\right)$$

Real pair

Fake pair

Image-to-image translation with Pix2Pix

- Data requirement of Pix2Pix
  - Pix2Pix assumes paired setting
  - Great when we have “free” training data, e.g., edges extracted from images serving as x



Loss functions

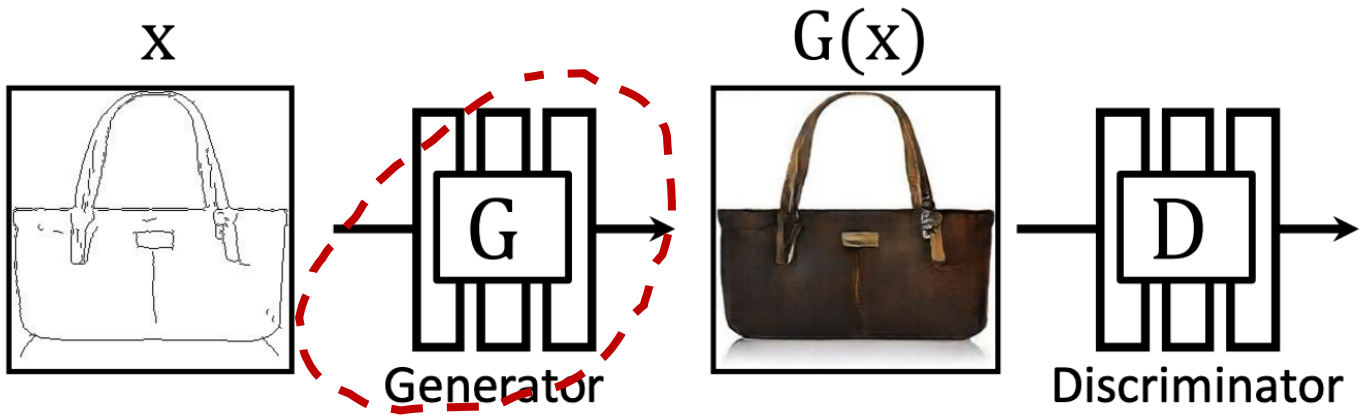
- Need to add L1 loss apart from the GAN loss to make the generation more constrained (to be near to the ground truth output)
- Using L1 is less burring than L2

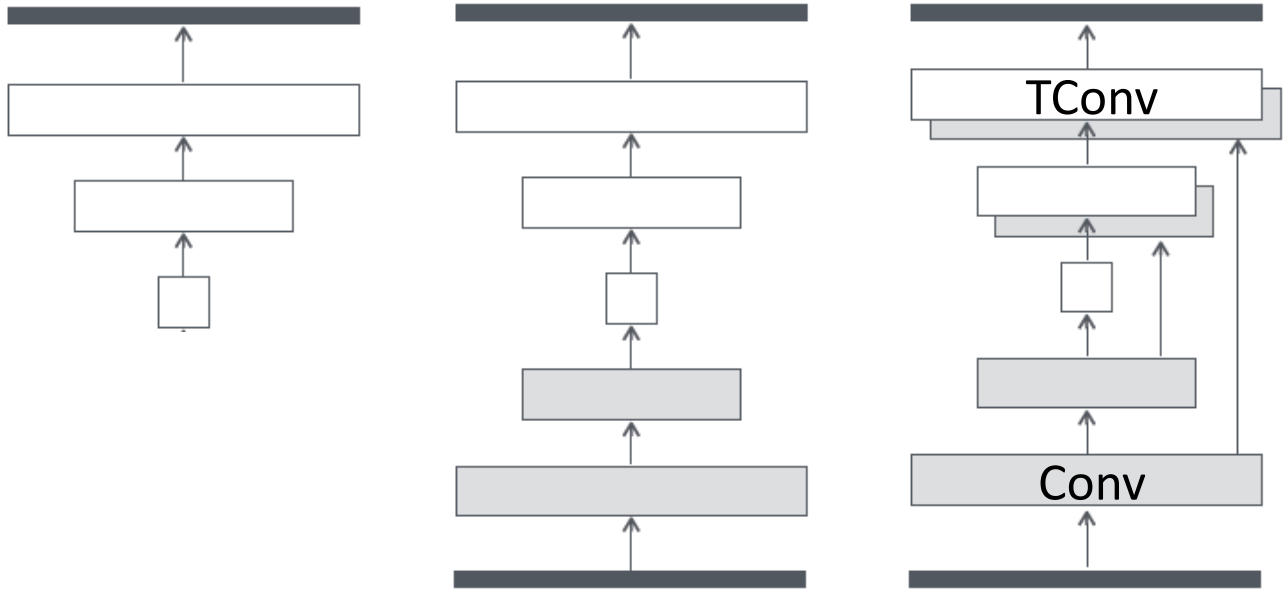
$$\min_{G} \max_{D} \left(E_{x,y}[\log D(x, y)] + E_{x}\left[\log\left(1 - D\left(x, G(x)\right)\right)\right]\right)$$

$$\min_{G} E_{x,y}[\|y - G(x)\|_1]$$

Network architecture

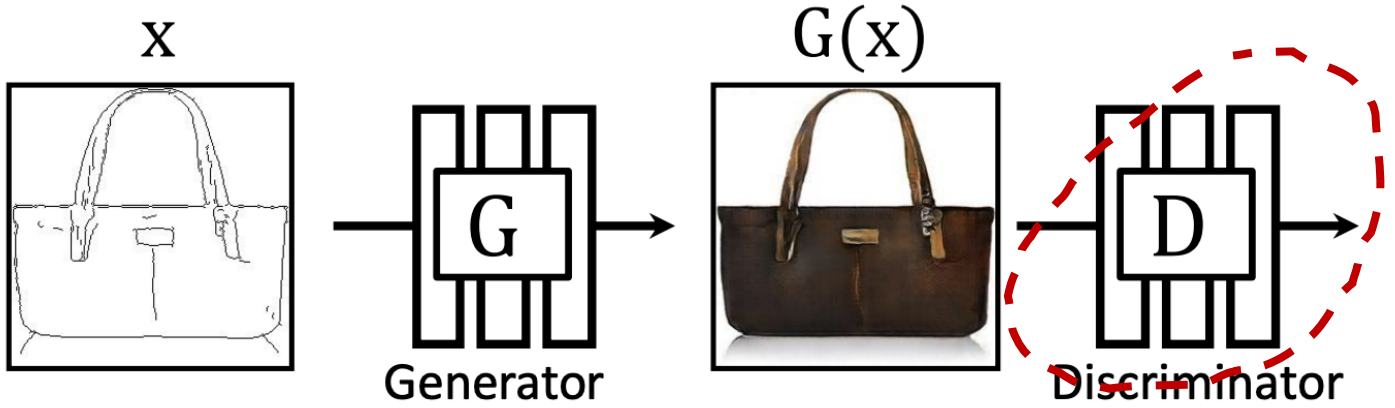
- Network design:
  - An encoder to map X into a latent vector as in conventional GAN
  - U-Net: skip connections to preserve low-level information
  - Encoder:
    - Conv+BN+ReLU
  - Decoder:
    - Tconv+BN+LeackyReLU



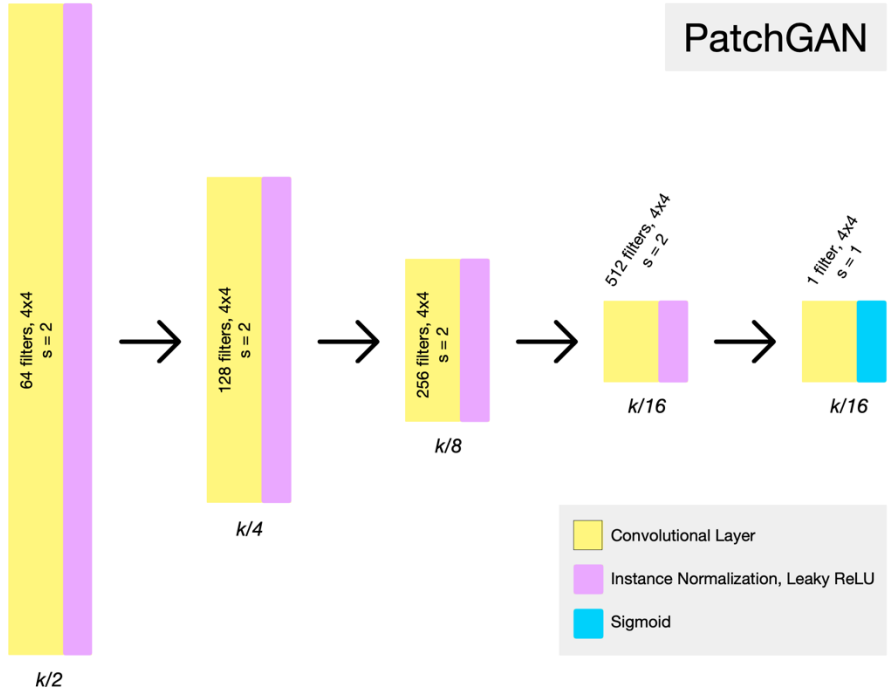


Network architecture

- Discriminator: PatchGAN, a fully convolutional neural network that looks at a “patch” of the input image, and output the probability of the patch being “real”.



- PatchGAN evaluates 70x70 sized patches of the input
- This is both more computationally efficient than trying to look at the entire input image, and is also more effective — it allows the discriminator to focus on more surface-level features, like texture, which is often the sort of thing being changed in an image translation task.



Network architecture

- U-Net
  - Preserve details
- PatchGAN
  - Small patch: color greater color diversity
  - Large patch: locally sharp results
  - Full image: artifacts

Encoder-Decoder

U-Net

L1

1x1

16x16

70x70

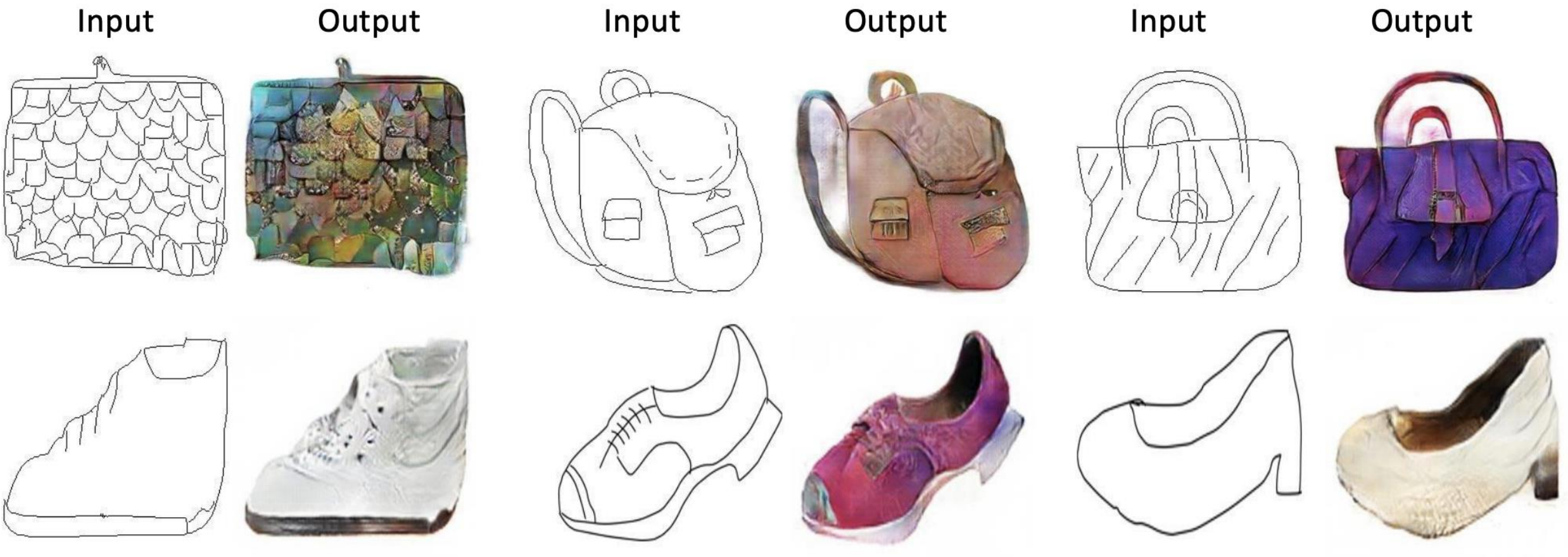
256x256

Isola et al., Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

Edge to image

Input Output Input Output Input Output

Sketch to image



We will solve this problem later (^o^)

Problem with Pix2Pix

painter

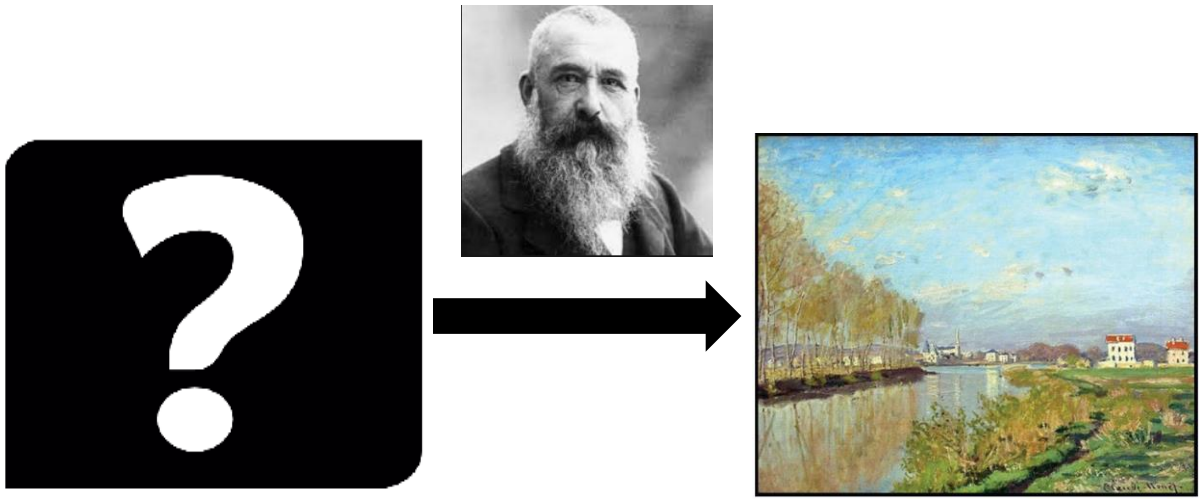
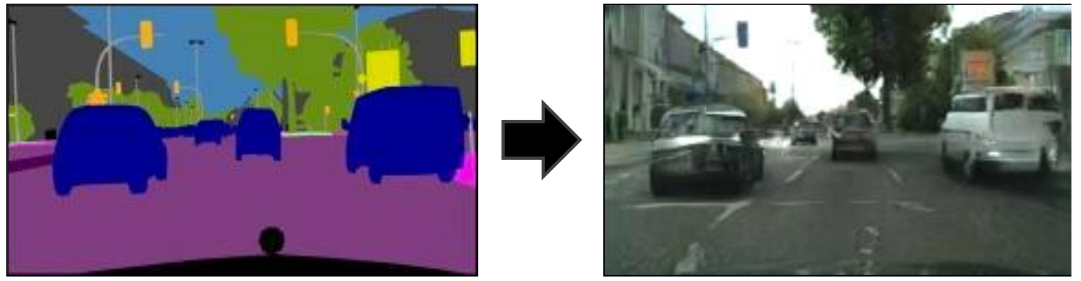
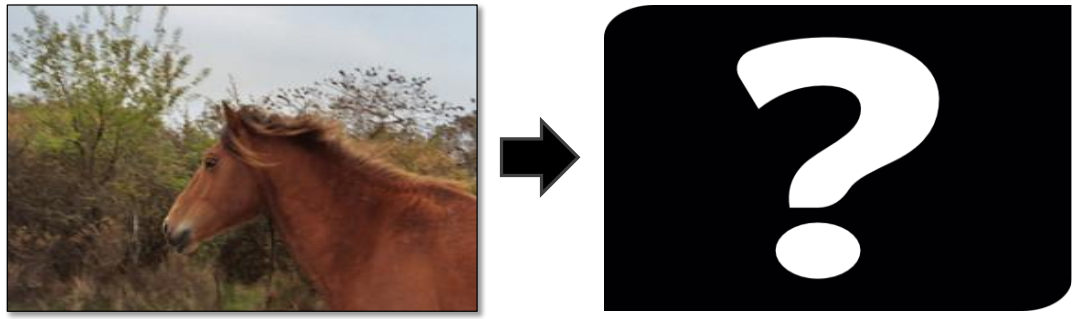


Photo ↔ Painting



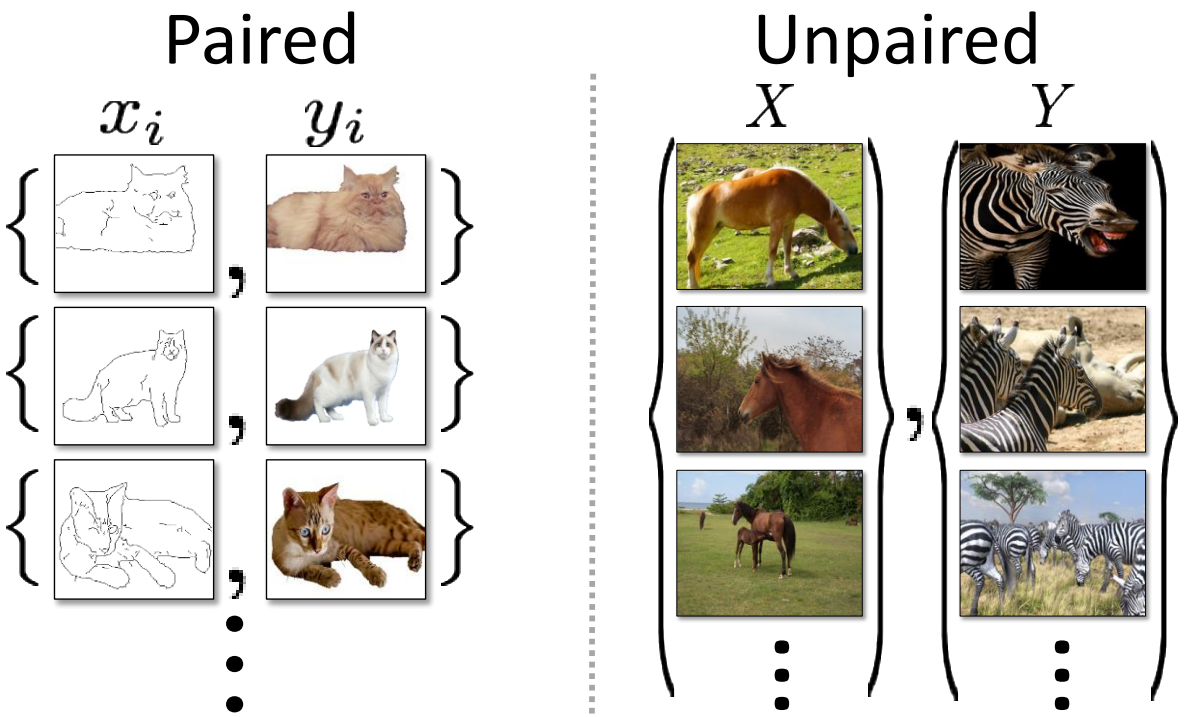
Label ↔ photo: per-pixel labeling



Horse ↔ zebra: how to get zebras?

- Expensive to collect pairs.
- Impossible in many scenarios.

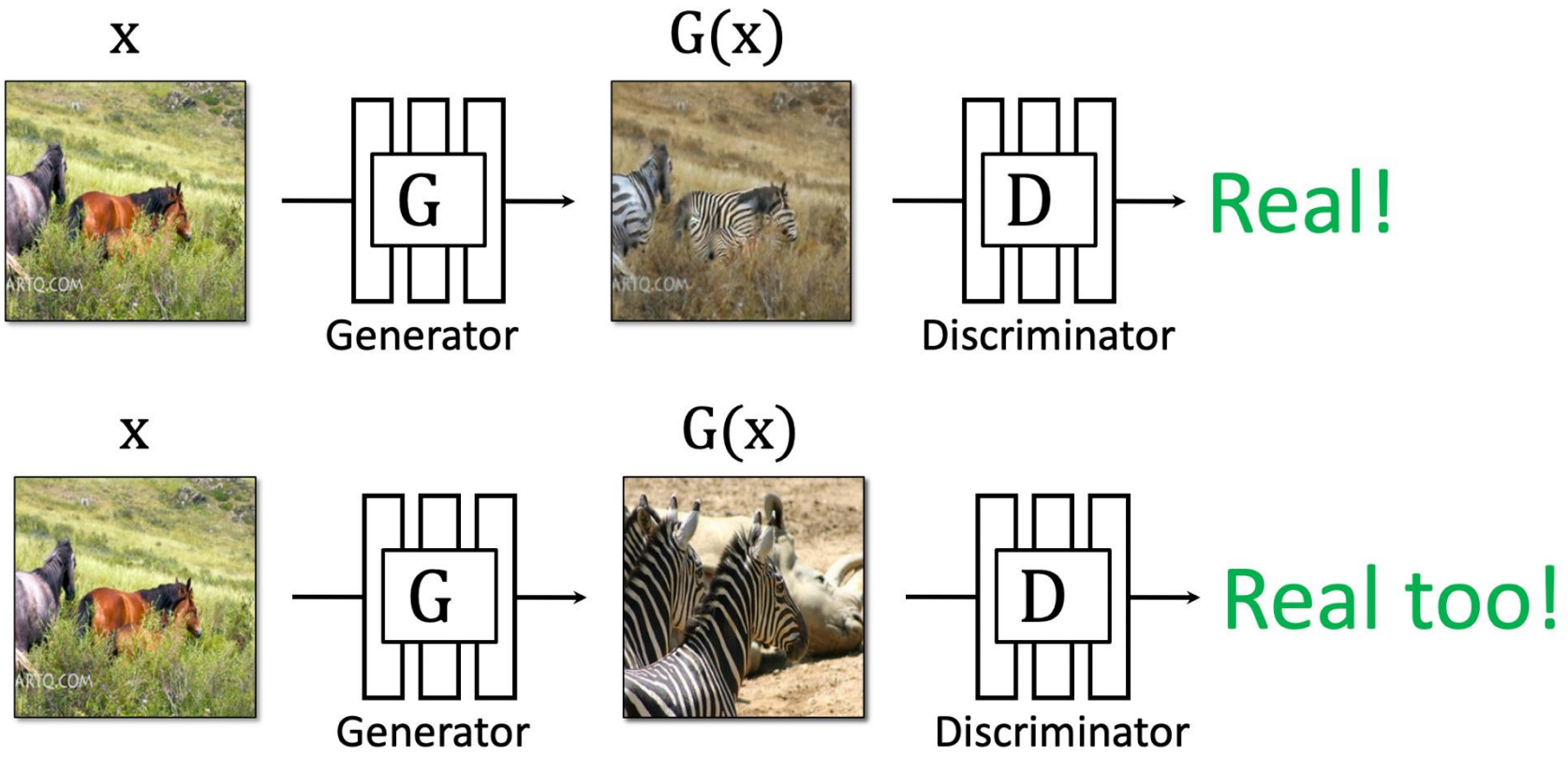
Can we use unpaired data?



Paired training data consists of training examples, where the correspondence between x_i and y_i exists.

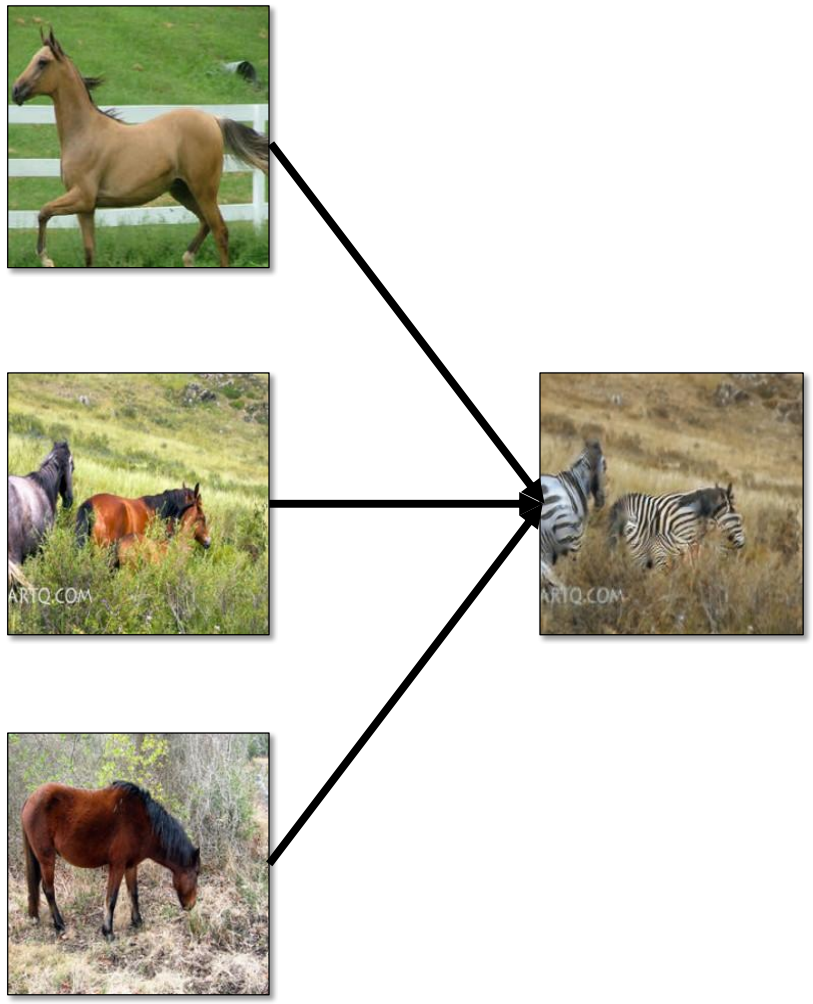
Unpaired data consist of a source set X and a target set Y, with no information provided as to which x_i matches which y_j.

Can we use unpaired data?



GANs do **not** force output to correspond to input

Can we use unpaired data?



Mode collapse

A network can map the same set of input images to any random permutation of images in the target domain, where any of the learned mappings can induce an output distribution that matches the target distribution.

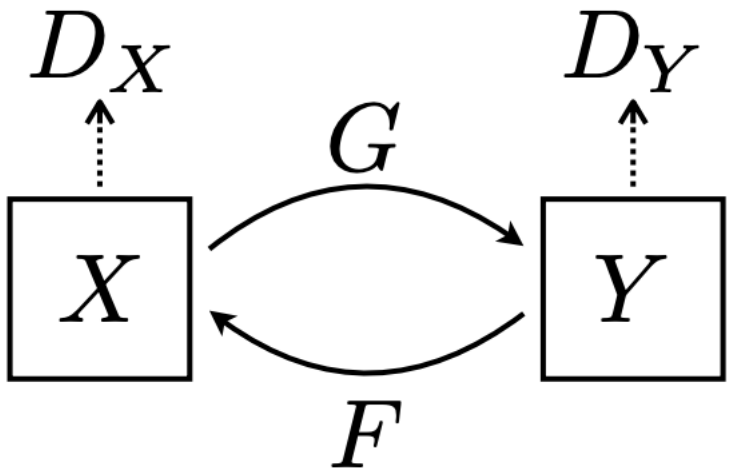
Adversarial losses alone cannot guarantee that the learned function can map an individual input x_i to a desired output y_i

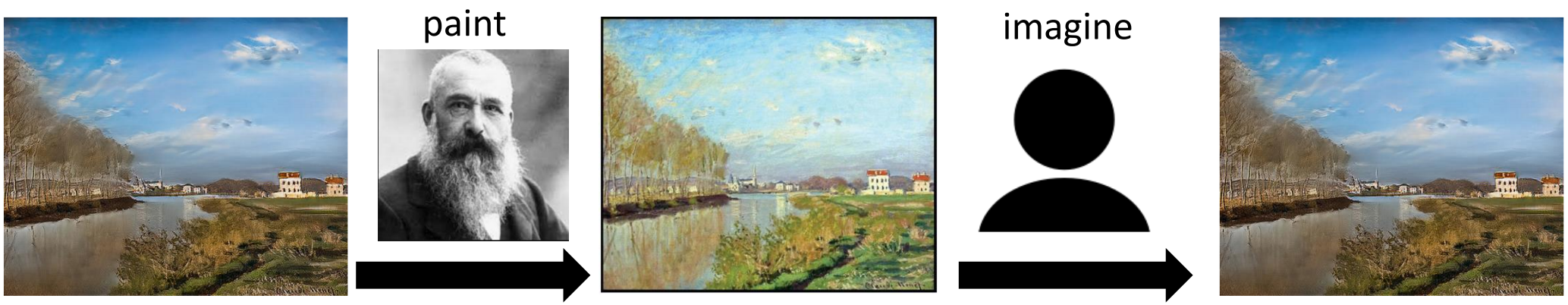
Cycle-Consistent Adversarial Networks

CycleGAN

The model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X.

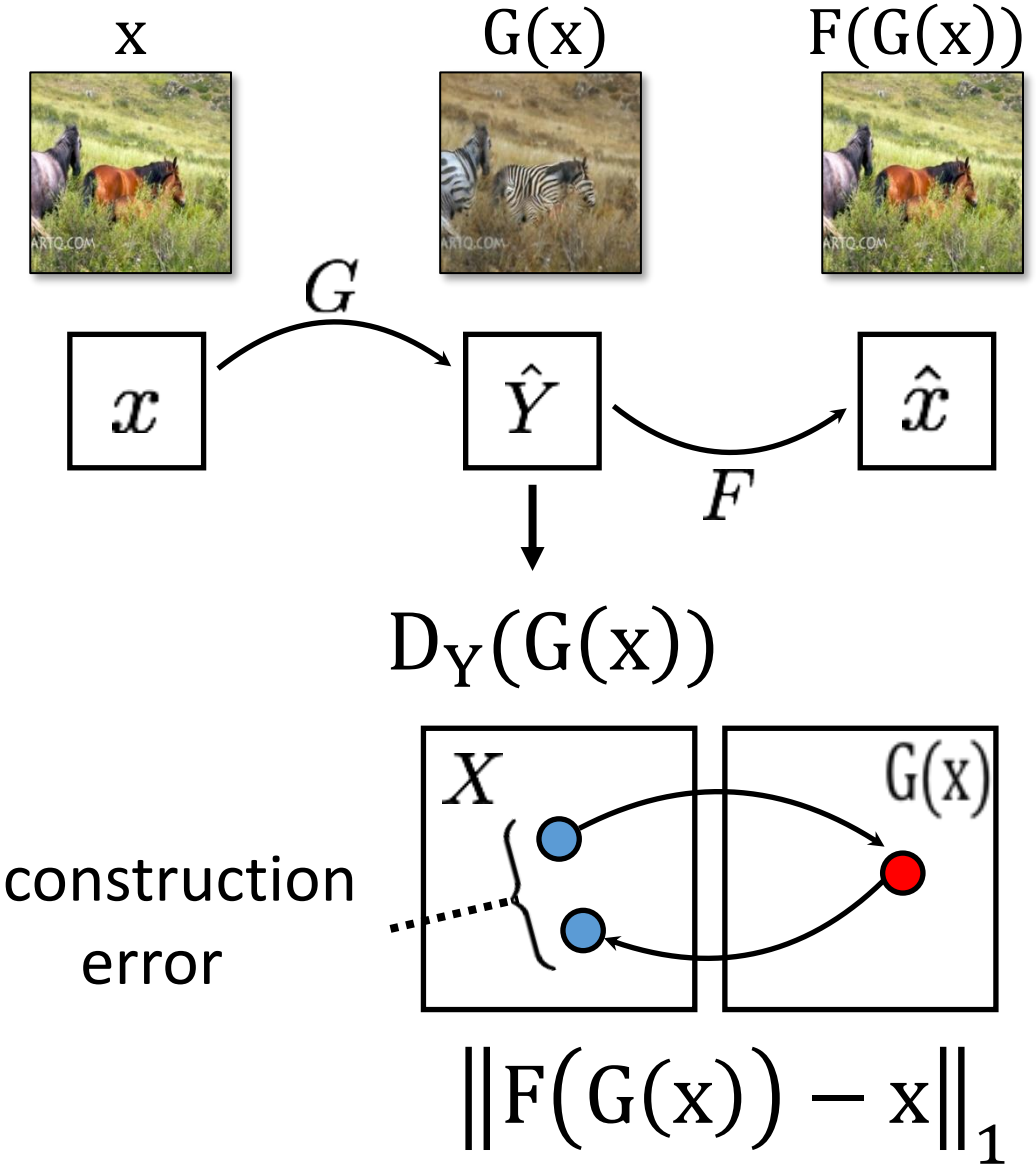
D_Y encourages G to translate X into outputs indistinguishable from domain Y , and vice versa for D_X and F.





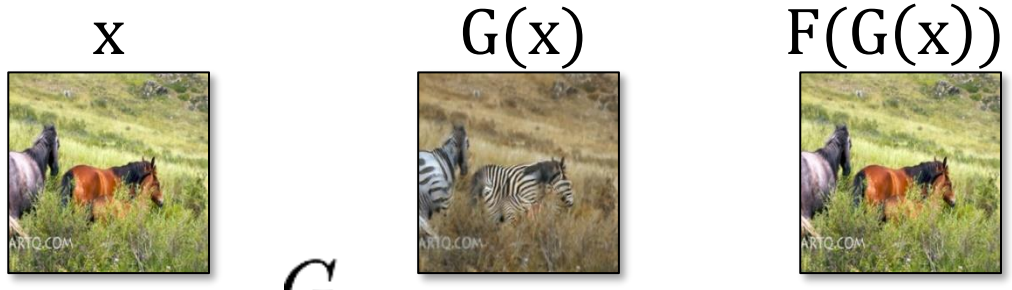
Zhu et al., Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV 2017

Cycle-Consistent Adversarial Networks

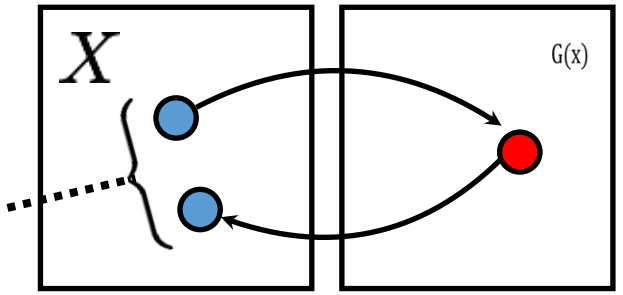


Zhu et al., Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV 2017

Cycle Consistency Loss



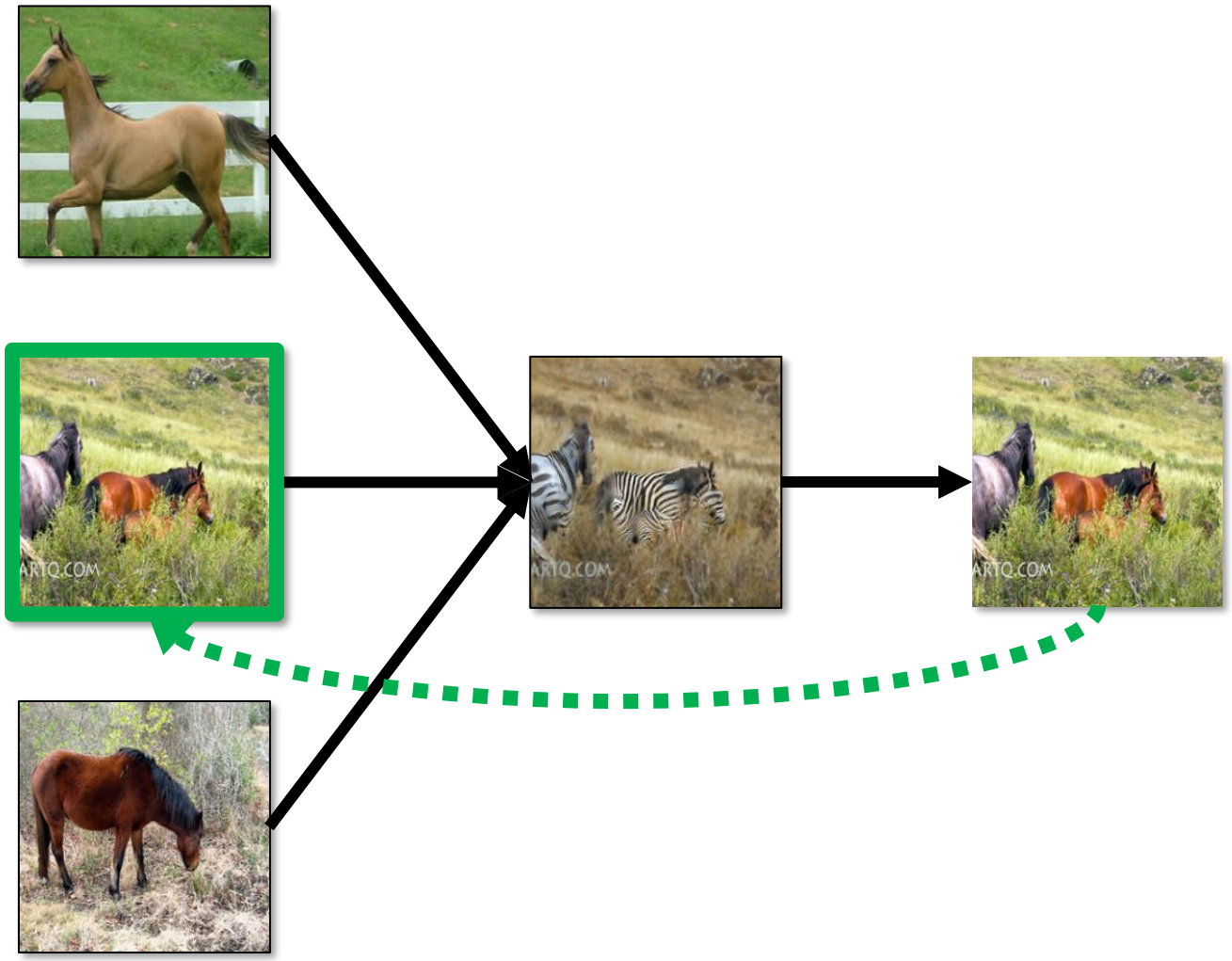
$D_Y(G(x))$



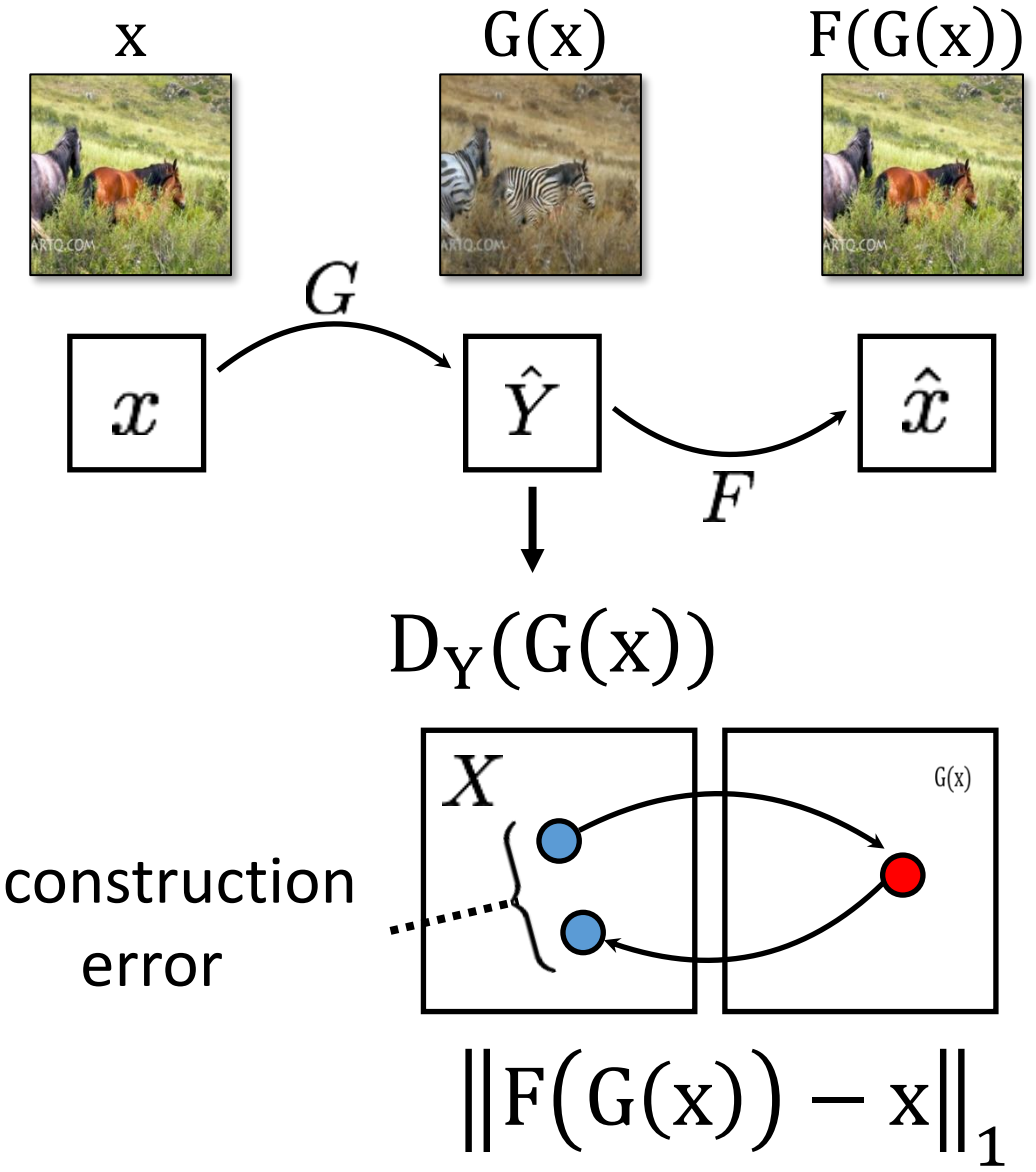
Reconstruction error

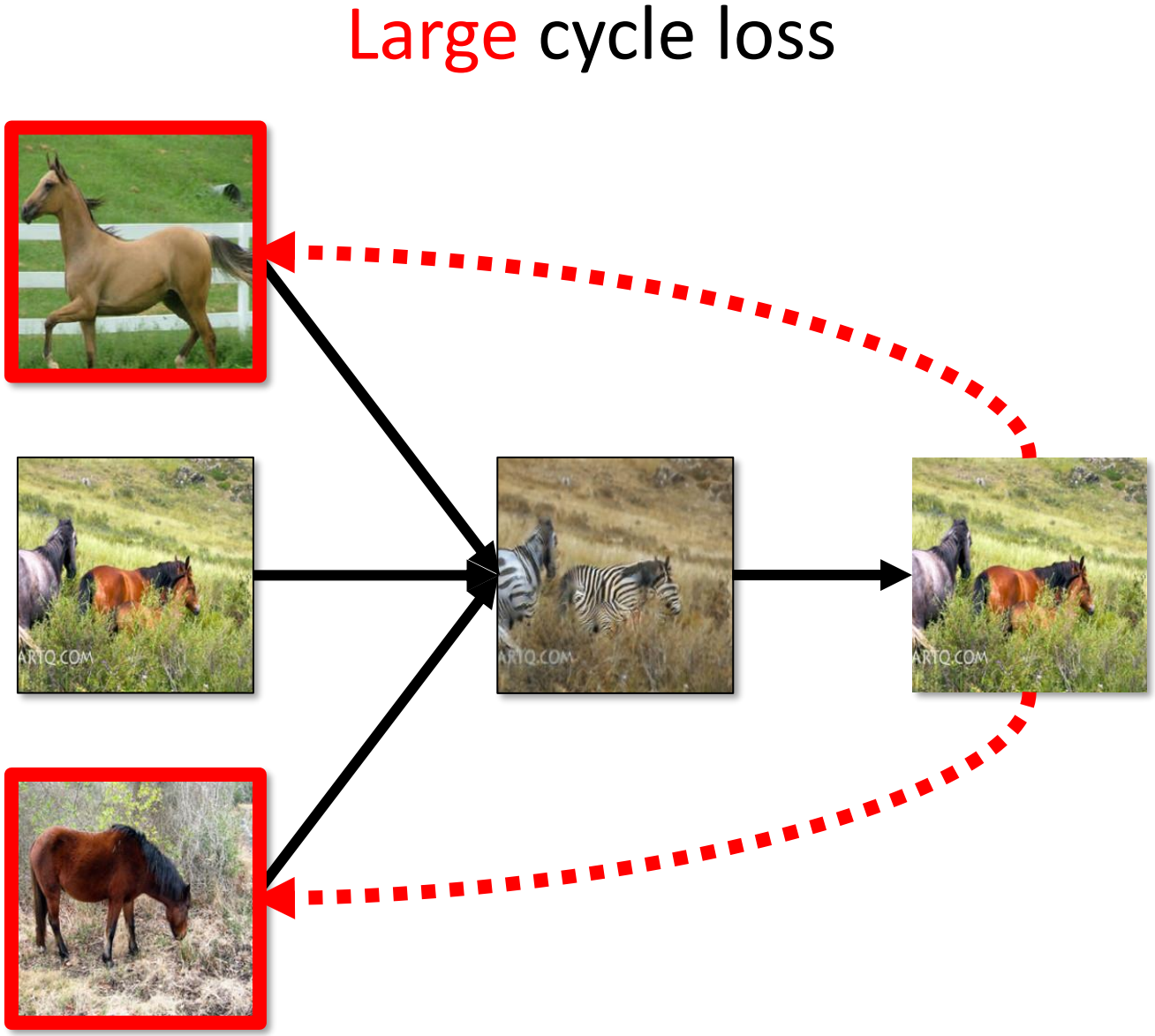
$\|F(G(x)) - x\|_1$

Small cycle loss



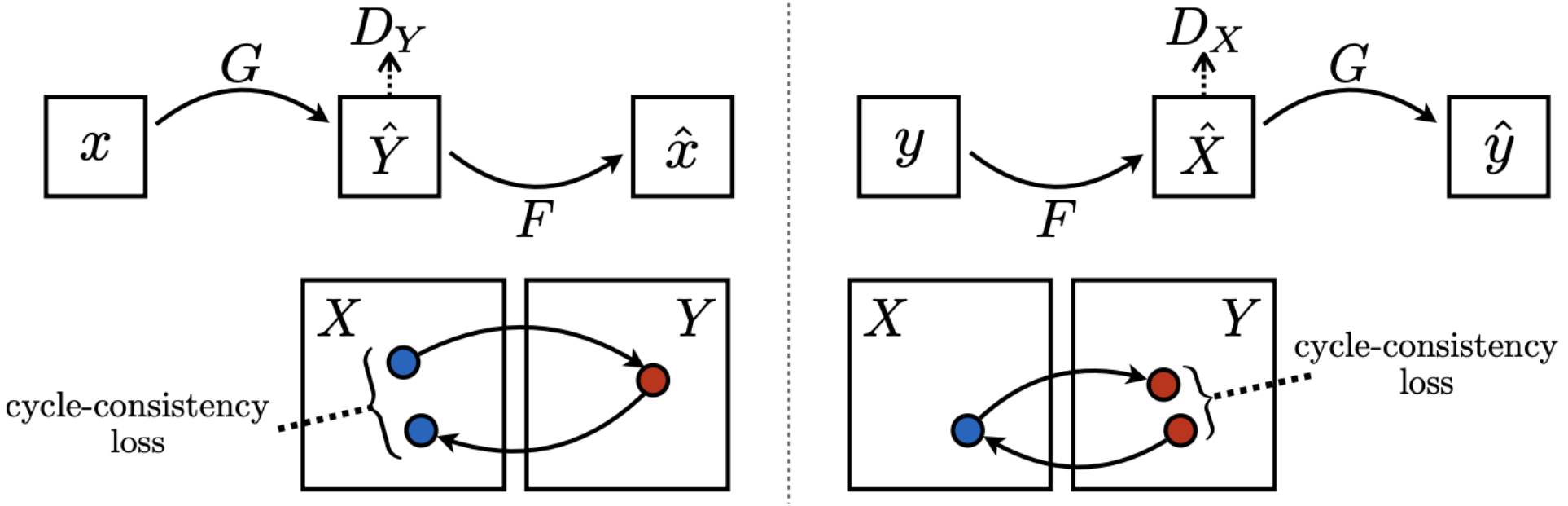
Cycle Consistency Loss



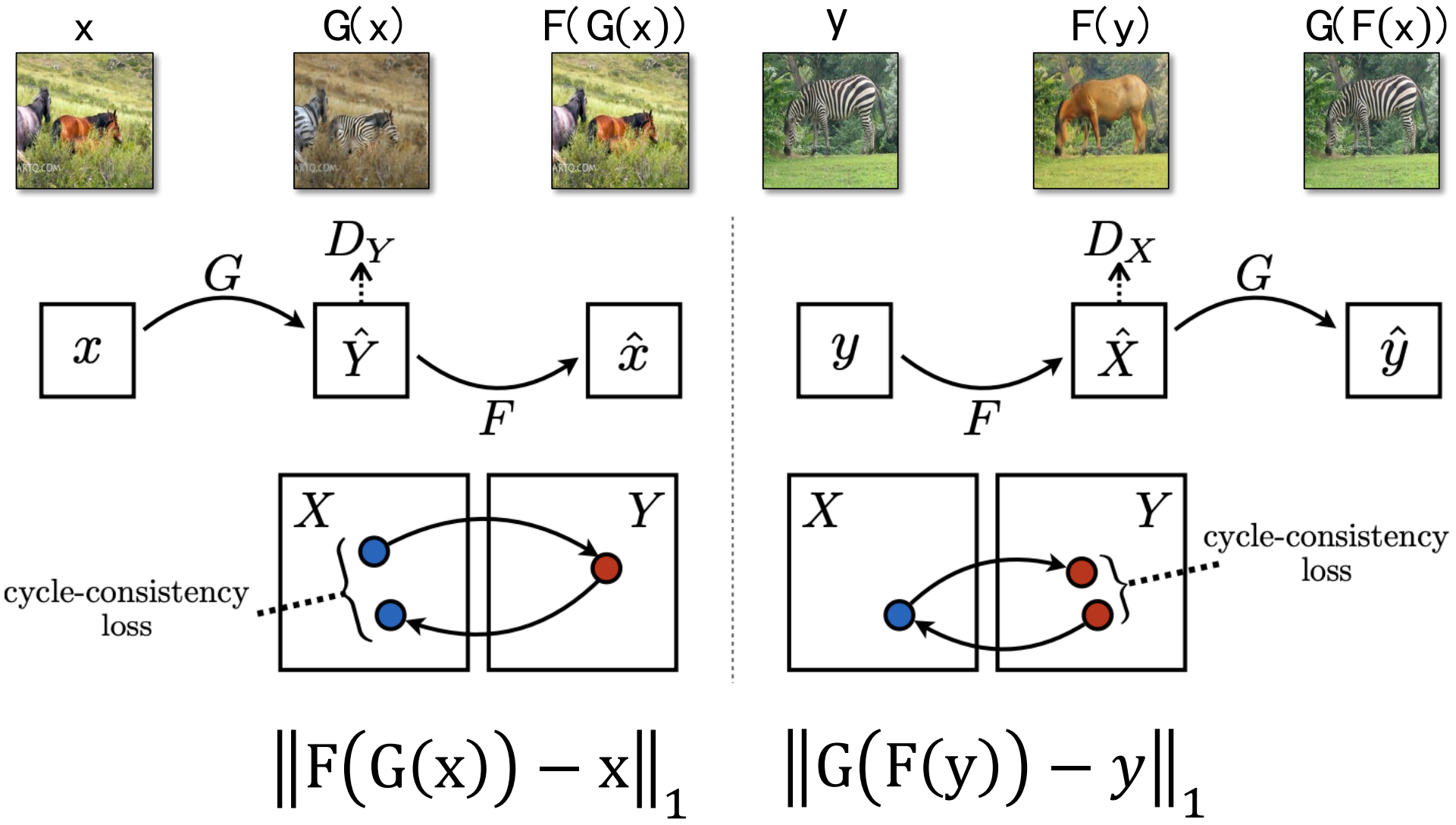


Cycle Consistency Loss

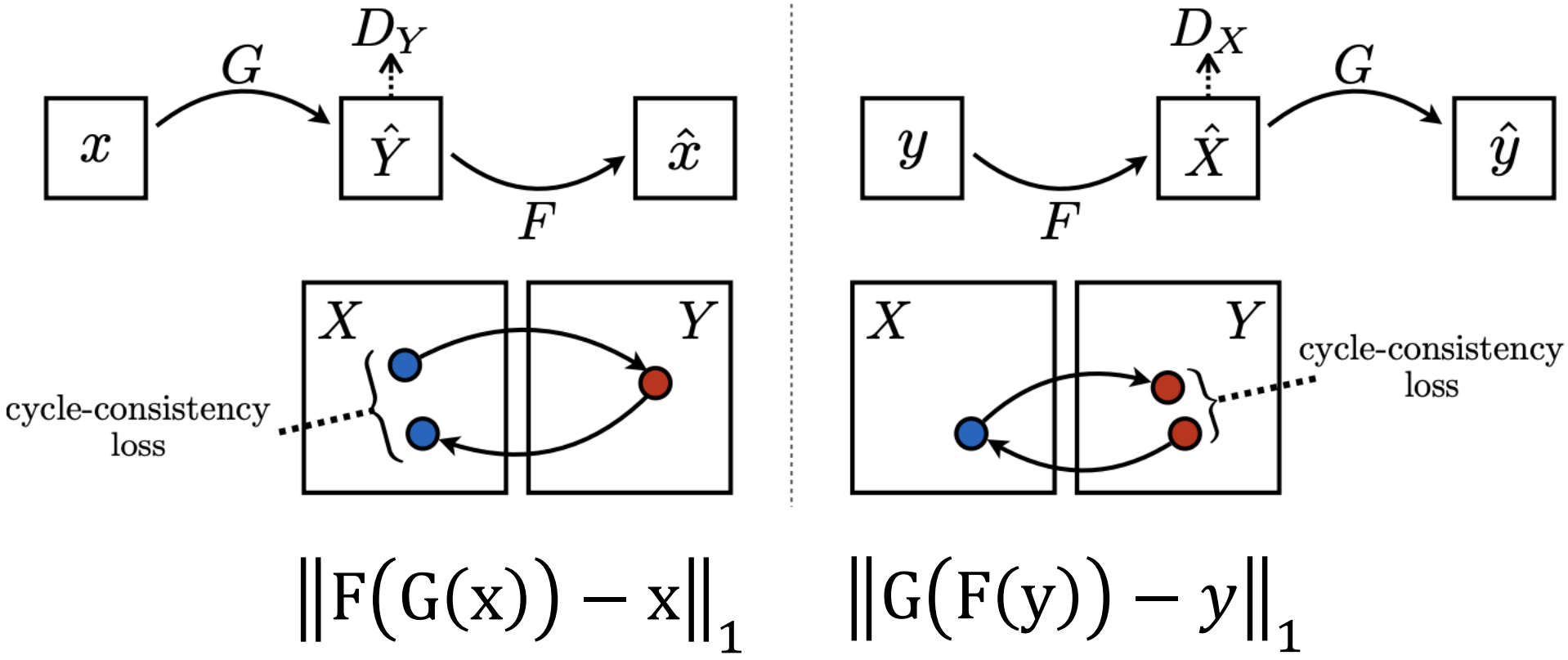
- To further regularize the mappings, we can introduce two cycle consistency losses that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started:
  - Forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and
  - Backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$



Cycle Consistency Loss



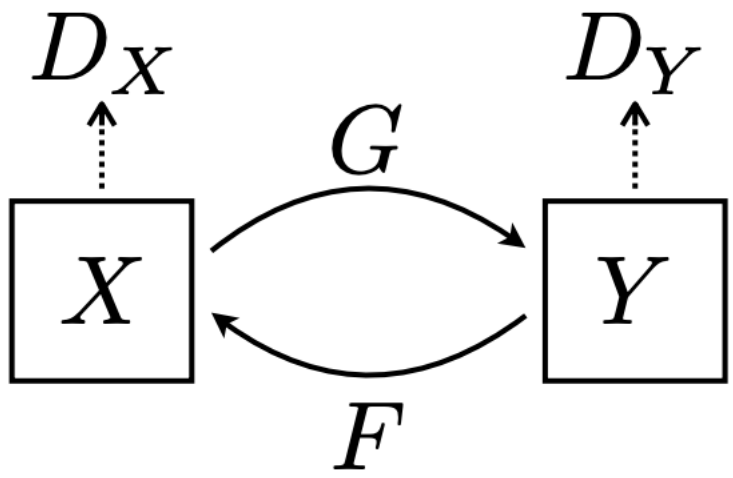
Cycle Consistency Loss



$$\|F(G(x)) - x\|_1 \qquad \|G(F(y)) - y\|_1$$

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

Full objective of CycleGAN



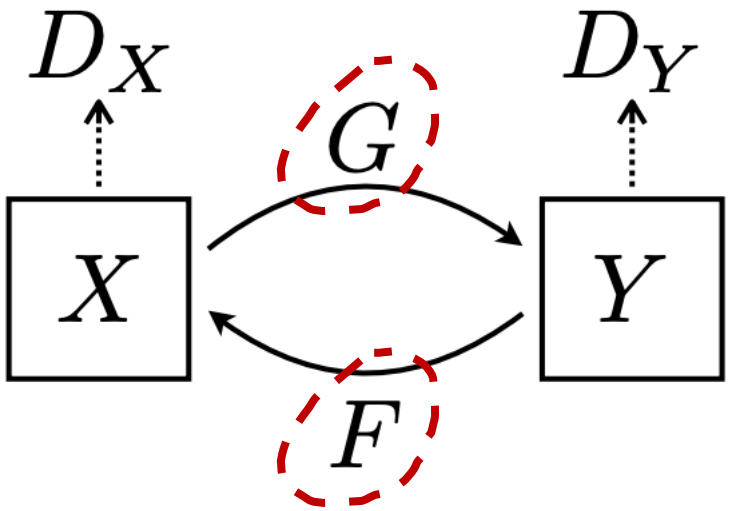
$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F),$$

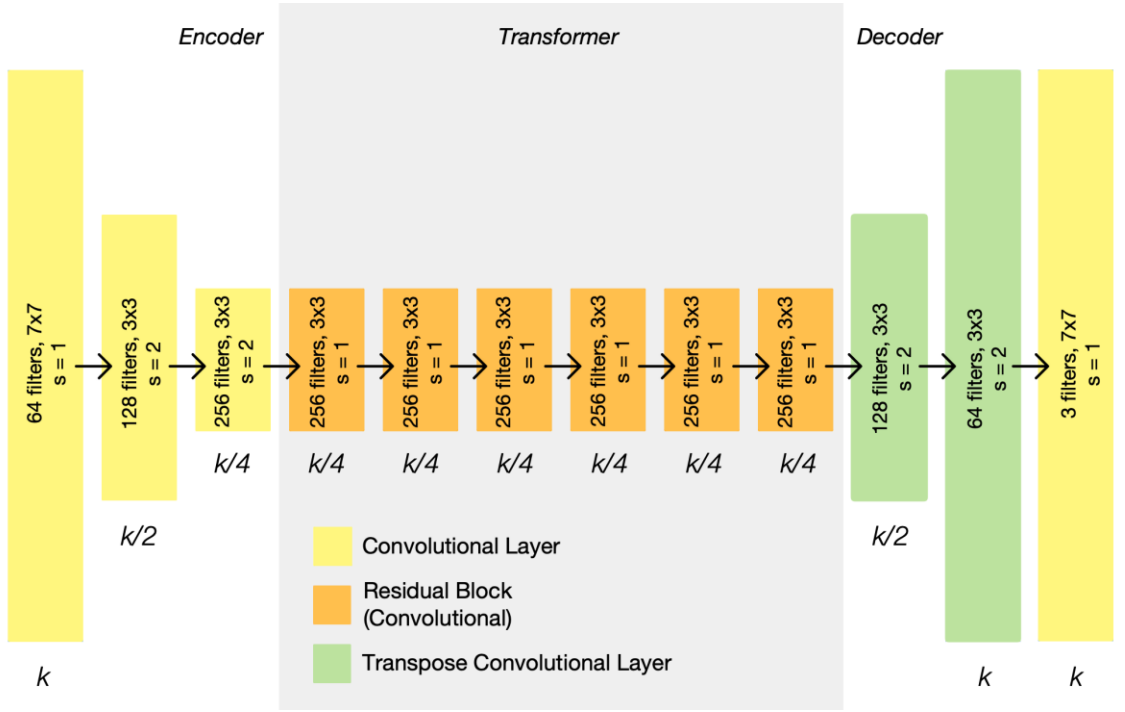
Adversarial Loss: change the style, e.g.,

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))]$$

Cycle Consistency Loss: preserve the content

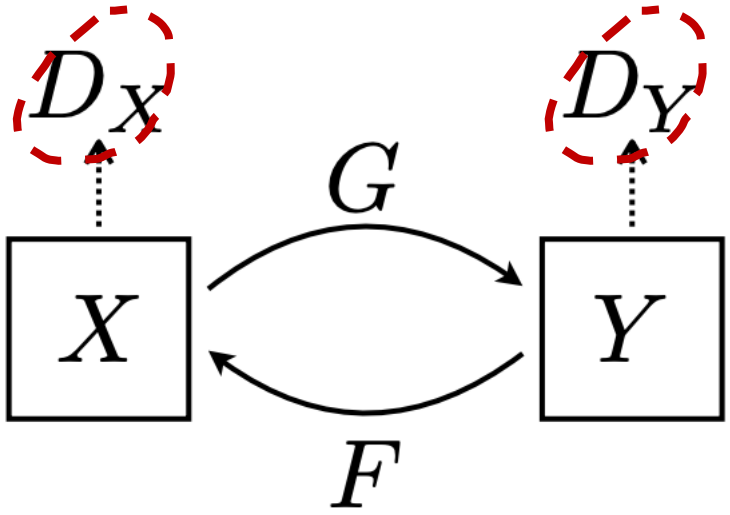
Network architecture

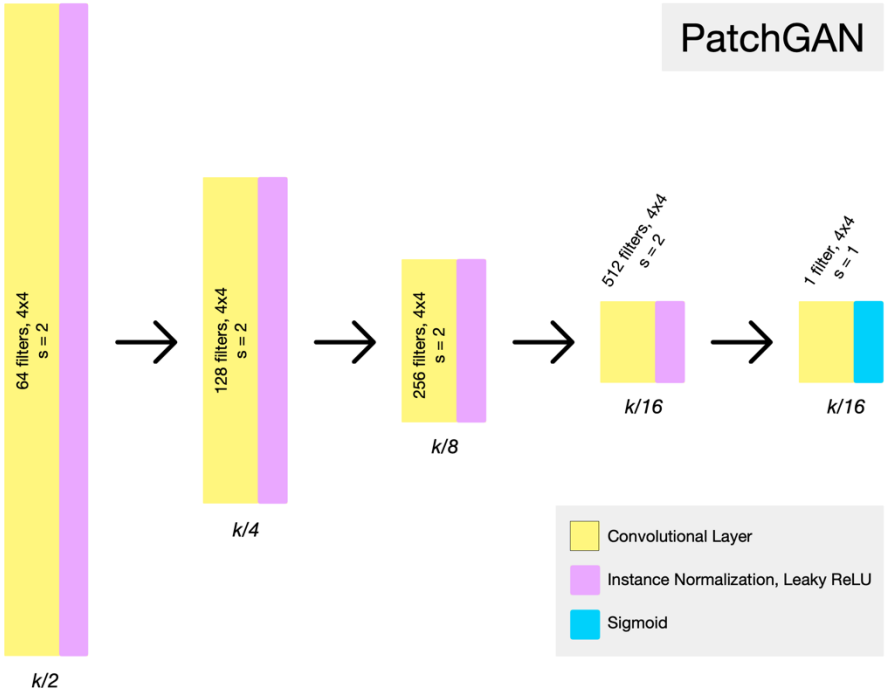




- Each CycleGAN generator has three sections: an *encoder*, a *transformer*, and a *decoder*.
- The input image is fed directly into the encoder, which shrinks the representation size while increasing the number of channels. The encoder is composed of three convolution layers.
- The resulting activation is then passed to the transformer, a series of six residual blocks.
- It is then expanded again by the decoder, which uses two transpose convolutions to enlarge the representation size, and one output layer to produce the final image in RGB.

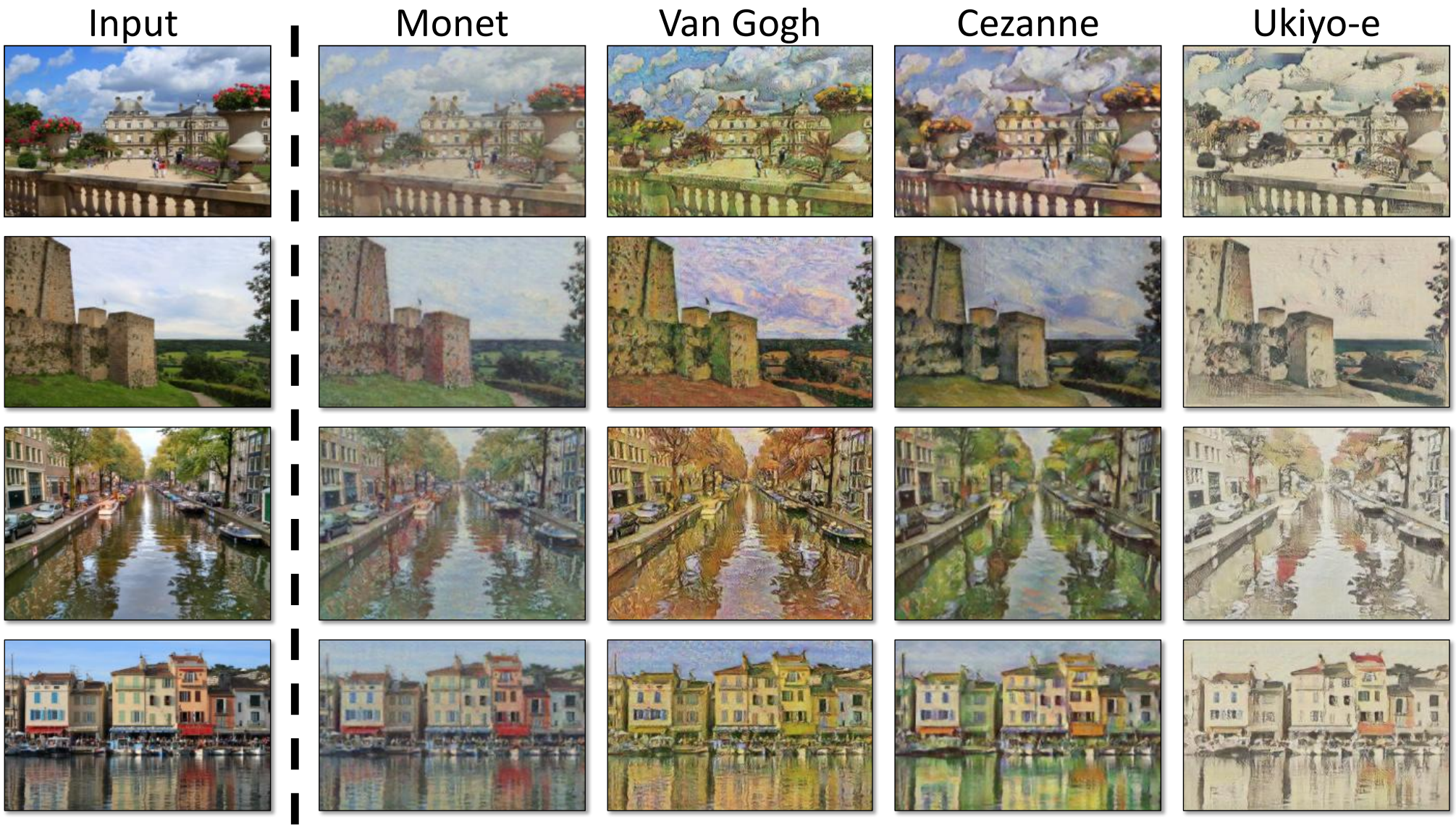
Network architecture





- The discriminators are PatchGANs, the same as Pix2Pix

Style transfer

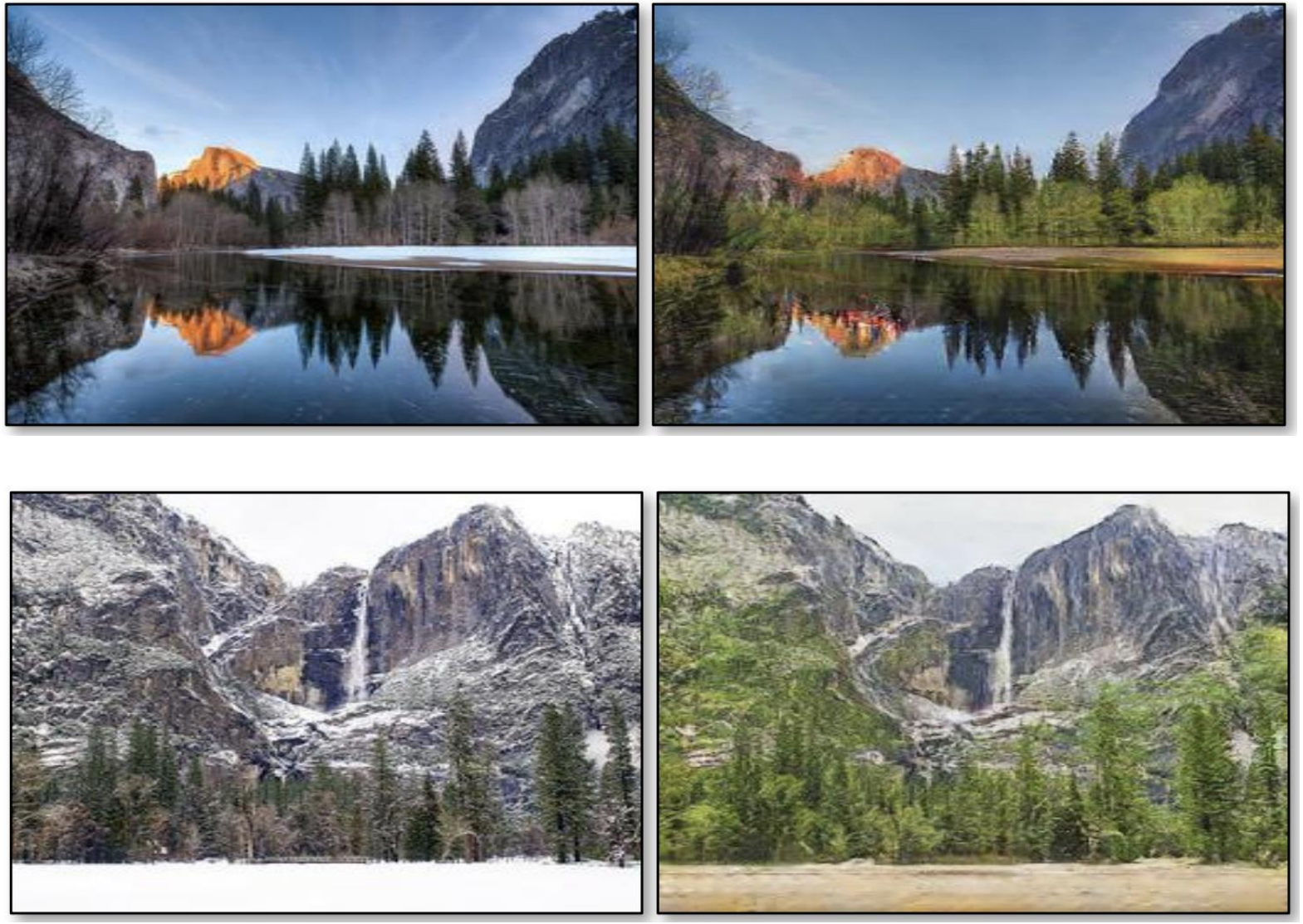


Monet’s paintings → photos

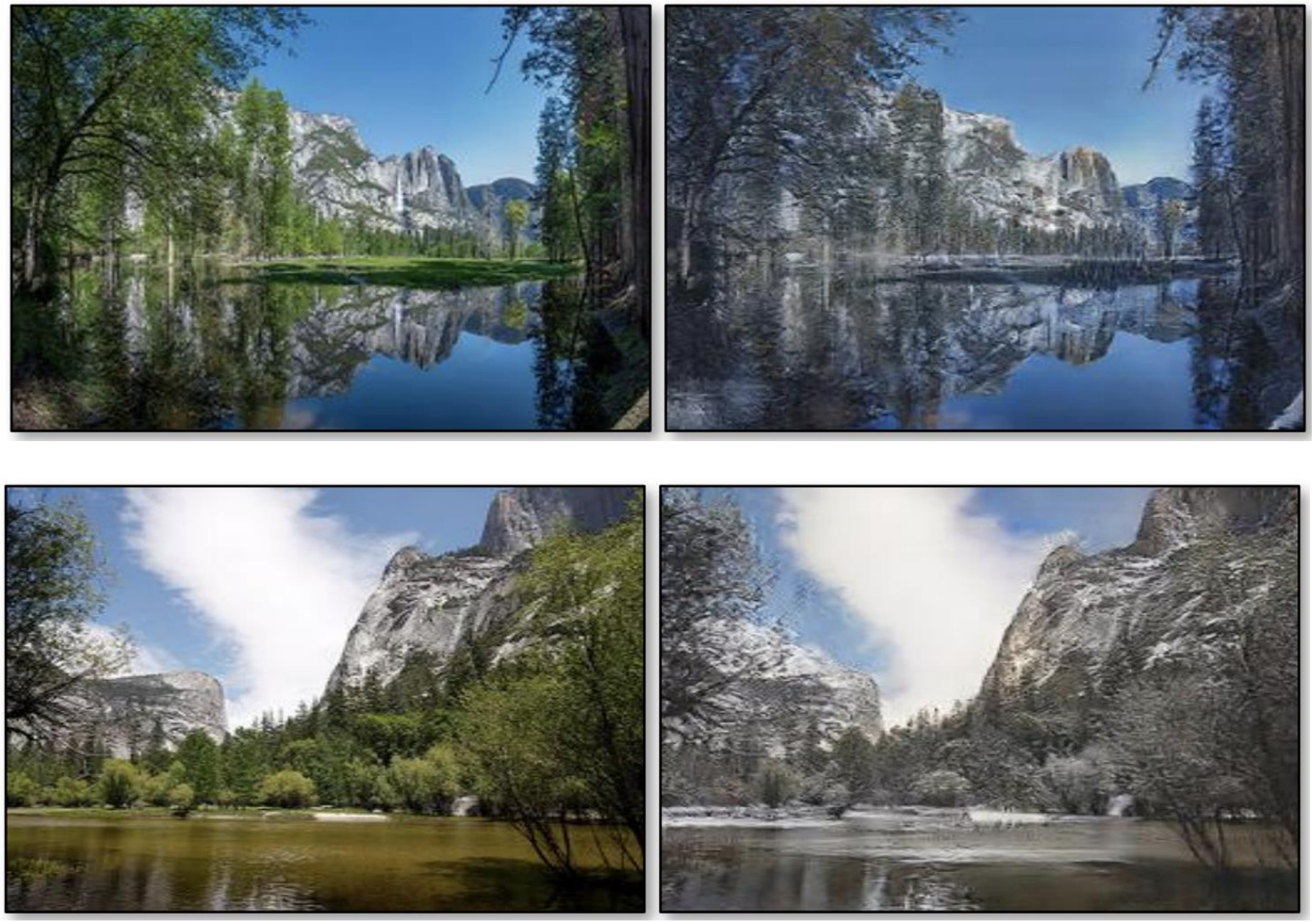
Monet’s paintings → photos

Monet’s paintings → photos

winter Yosemite → summer Yosemite



summer Yosemite → winter Yosemite



apple → orange

orange → apple

CG2Real: GTA5 → real streetview

GTA5 CG Input

Output

Real2CG: real streetview → GTA




Cityscape Input

Output

Failure cases

Cat to dog

Horse to zebra

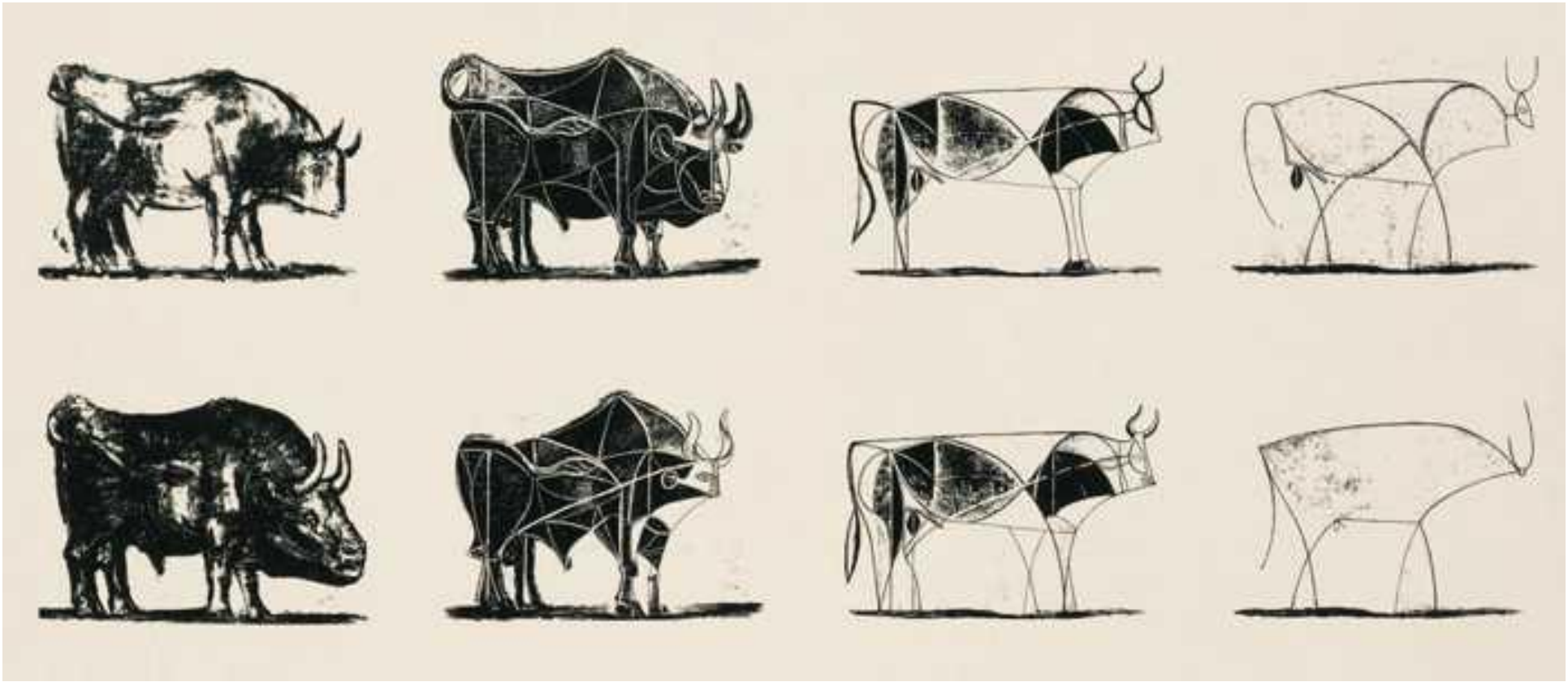
On translation tasks that involve color and texture changes, the method often succeeds

Fail on those that require geometric changes, e.g., cat to dog transfiguration, the learned translation degenerates into making minimal changes to the input

Confused when there are other objects not seen in the training, because the model was trained on the wild horse and zebra synsets of ImageNet, which does not contain images of a person riding a horse or zebra

Failure cases

In Style Transfer, the Cycle Consistency Loss is too strict, sometimes.



PABLO PICASSO (1881-1973)
'Bull', 1945 (a series of eleven lithographs)

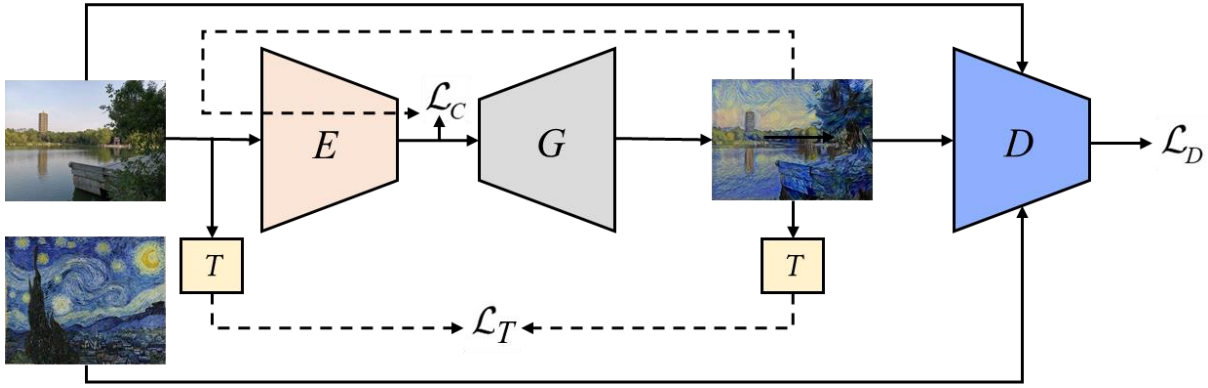
Too abstract to recover its original form
The translation is imbalanced.

Failure cases

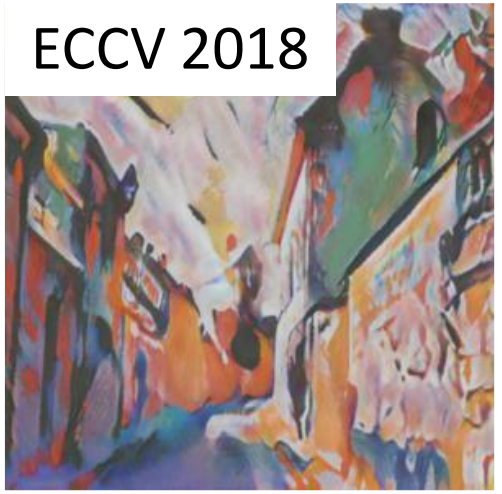
Instead pixel-level reconstruction, we use feature-level consistency

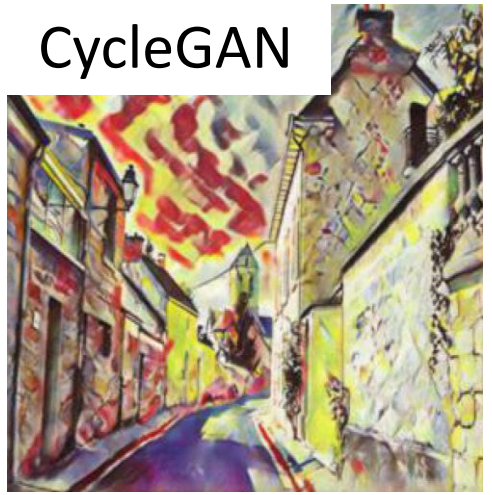
$$\|F(G(x)) - x\|_1 \rightarrow$$

$$\|E(G(E(x))) - E(x)\|_1$$



→

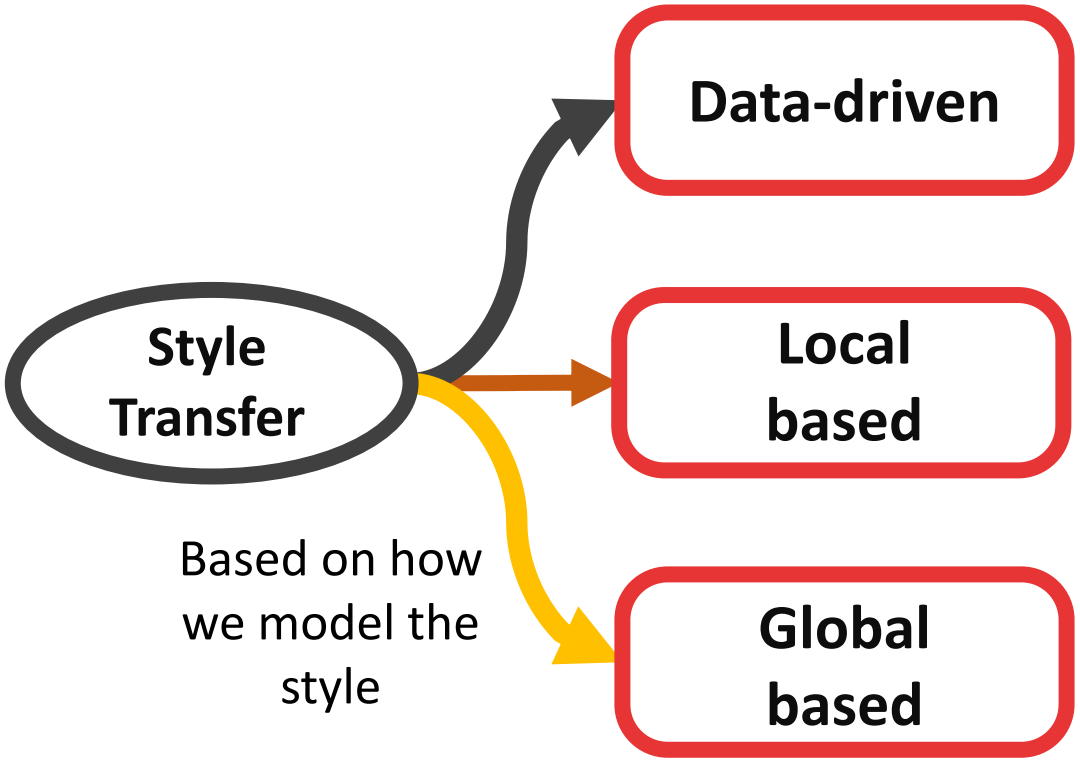


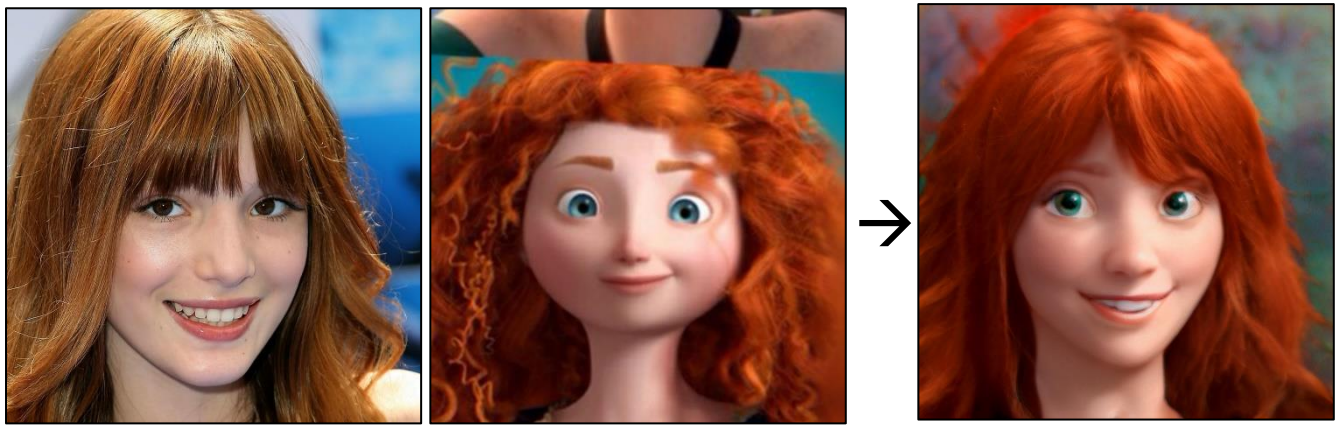


Sanakoyeu, Artsiom, et al. "A style-aware content loss for real-time hd style transfer." ECCV 2018.

Style transfer

Taxonomy





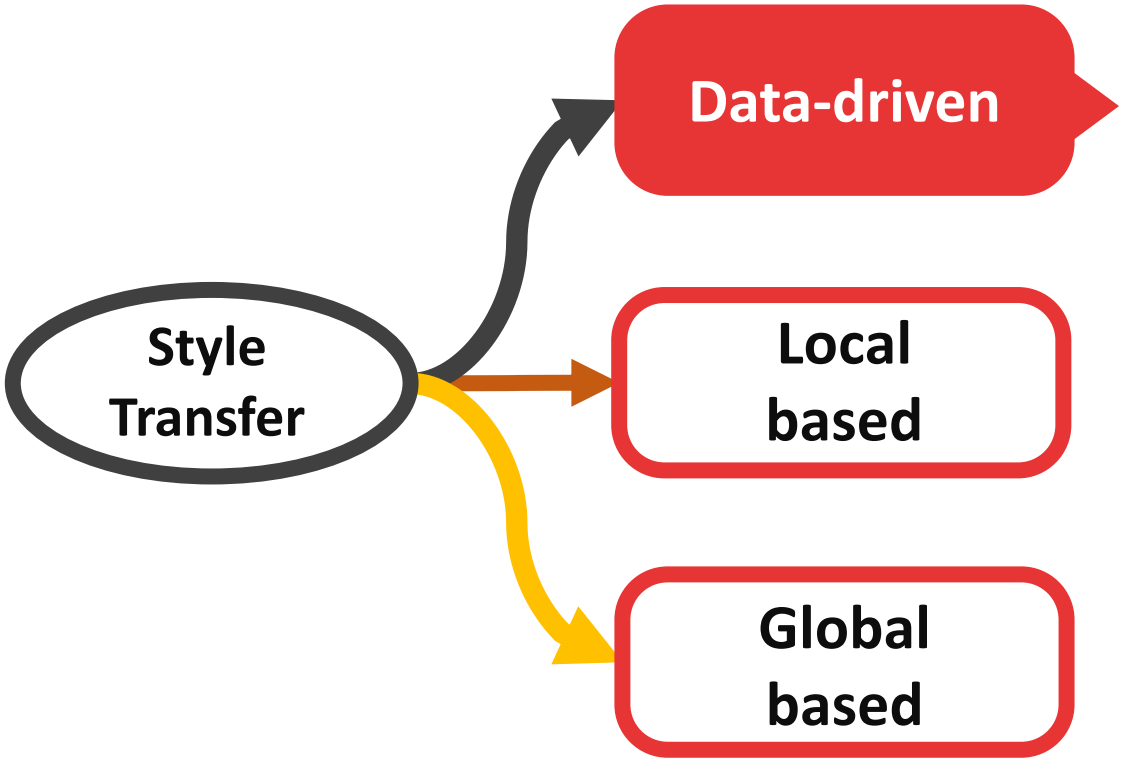
Content image | Style image | Style transfer result

Style collections

S. Yang, L. Jiang, Z. Liu, C. C. Loy. Pastiche Master: Exemplar-Based High-Resolution Portrait Style Transfer. CVPR 2022

Taxonomy

Implicit modelling



Pix2Pix, CycleGAN

Main idea: GAN

Collection-wise styles

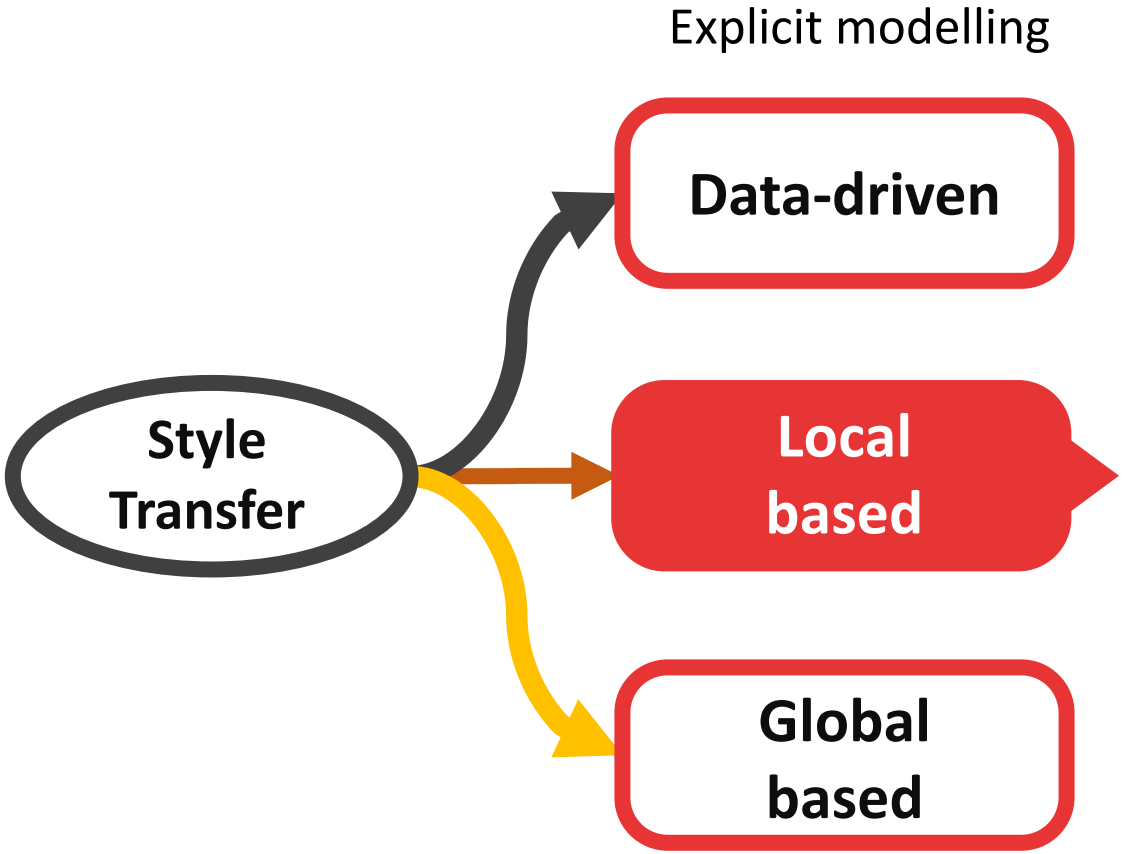
Learn mapping between two domains

Instead of defining what style is,

Learn to classify the real or fake style

- Paired: Pix2Pix
- Unpaired: CycleGAN

Taxonomy

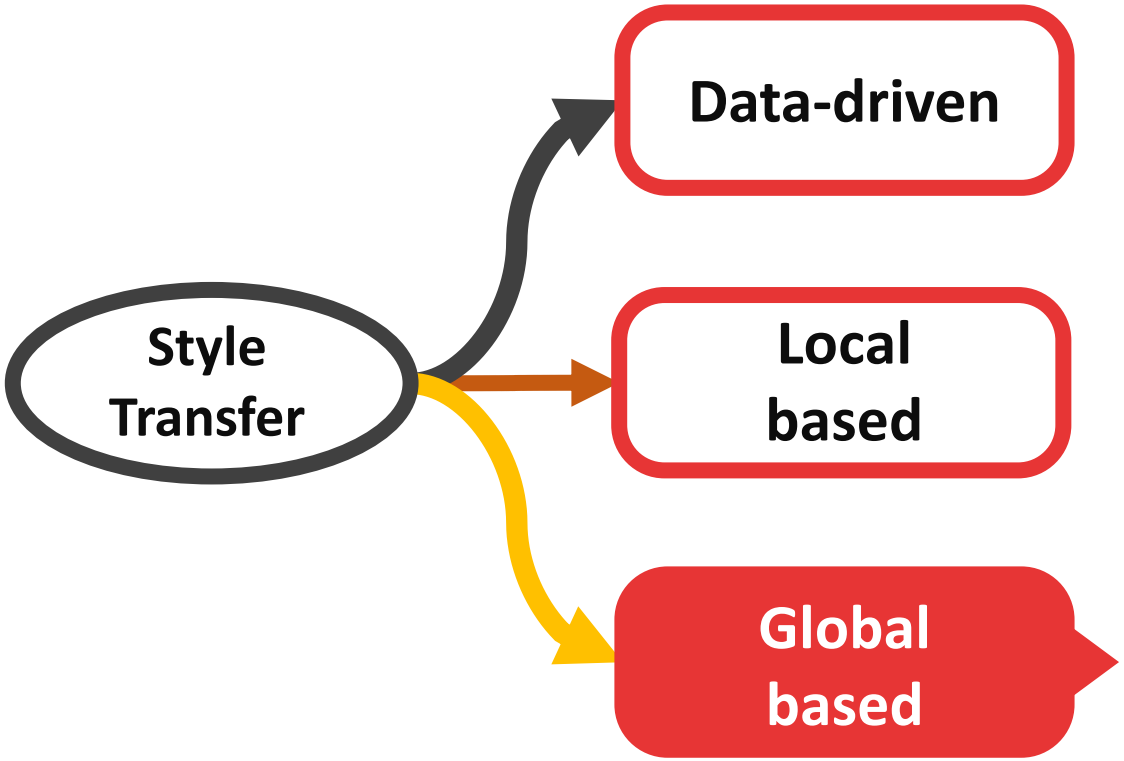


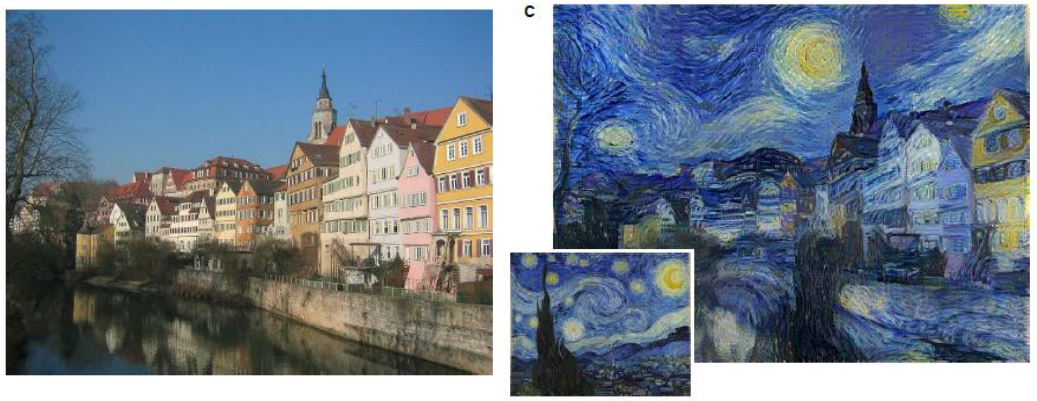
Main idea:
Image-wise styles
Based on the idea of strokes
Style as patches
match local patches/pixels
Fuse patches/pixels into the content image

C. Li and M. Wand. Combining Markov random fields and convolutional neural networks for image synthesis. CVPR. 2016.

Taxonomy

Explicit modelling





Main idea:

Match the global feature distributions between the style image and the target image

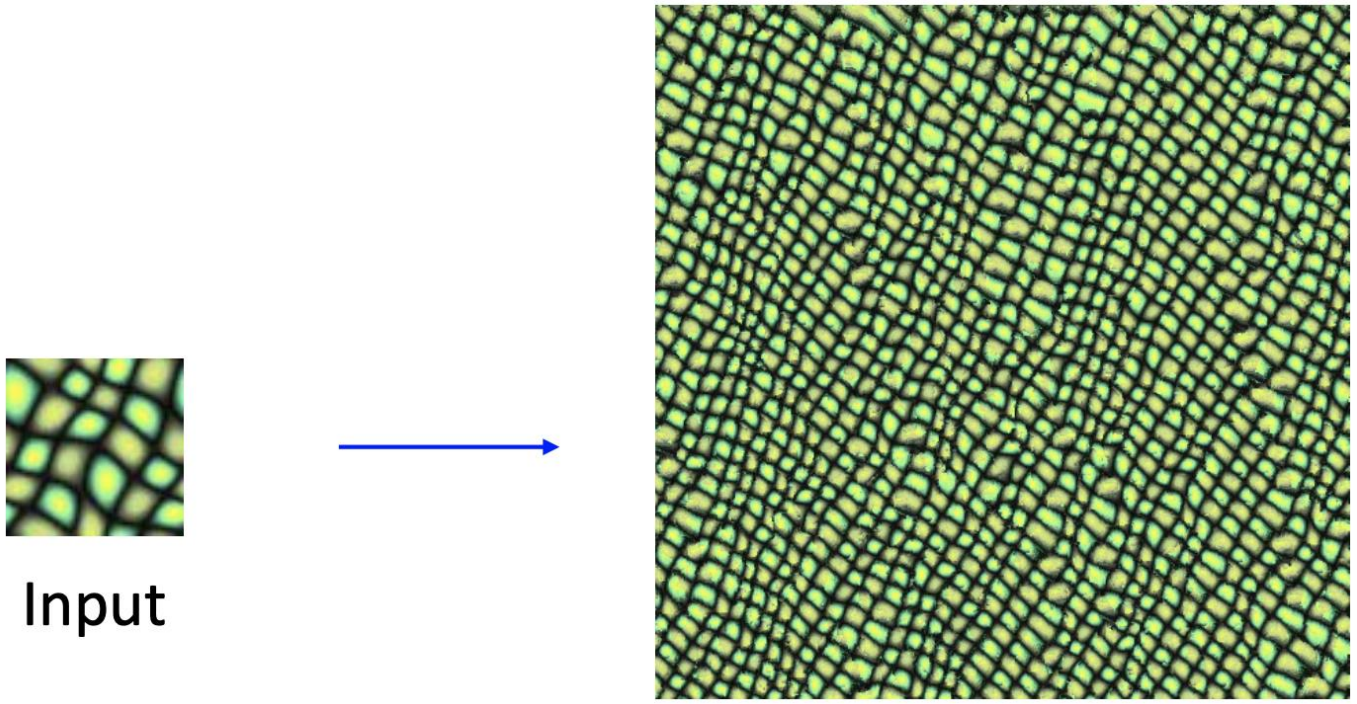
- Gram Matrix (Covariance)
- Mean & Variance
- Whitening & Coloring

L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using nonvolutional neural networks. CVPR. 2016.

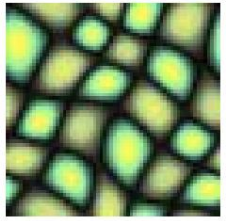
Texture Synthesis

A special case of Style Transfer – Texture synthesis

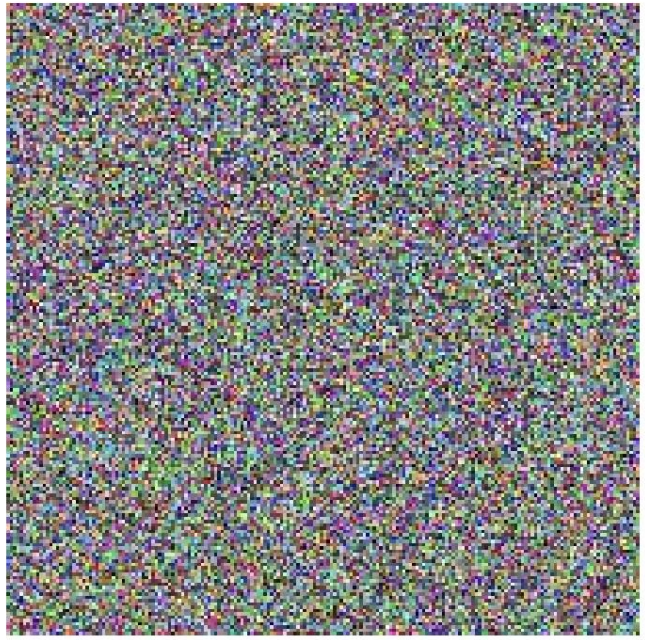
Given a sample patch of some texture, can we generate a bigger image of the same texture?



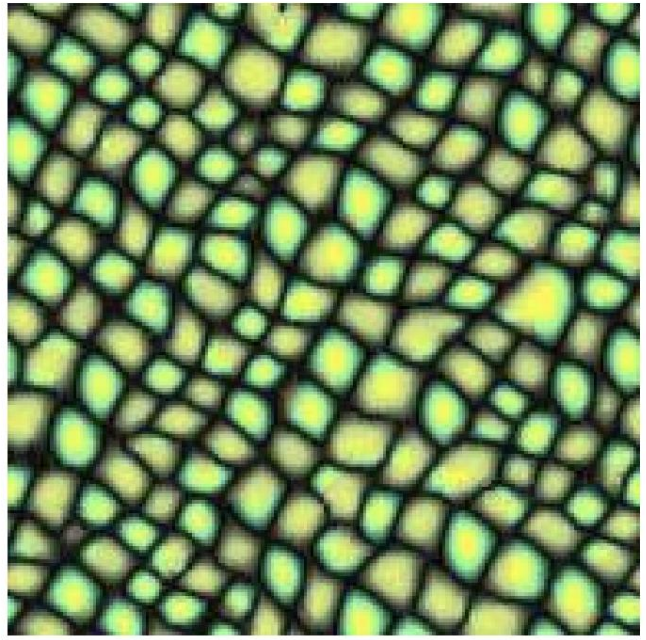
Texture Synthesis: Local-Based Method



Style image



Content image



Style transfer result

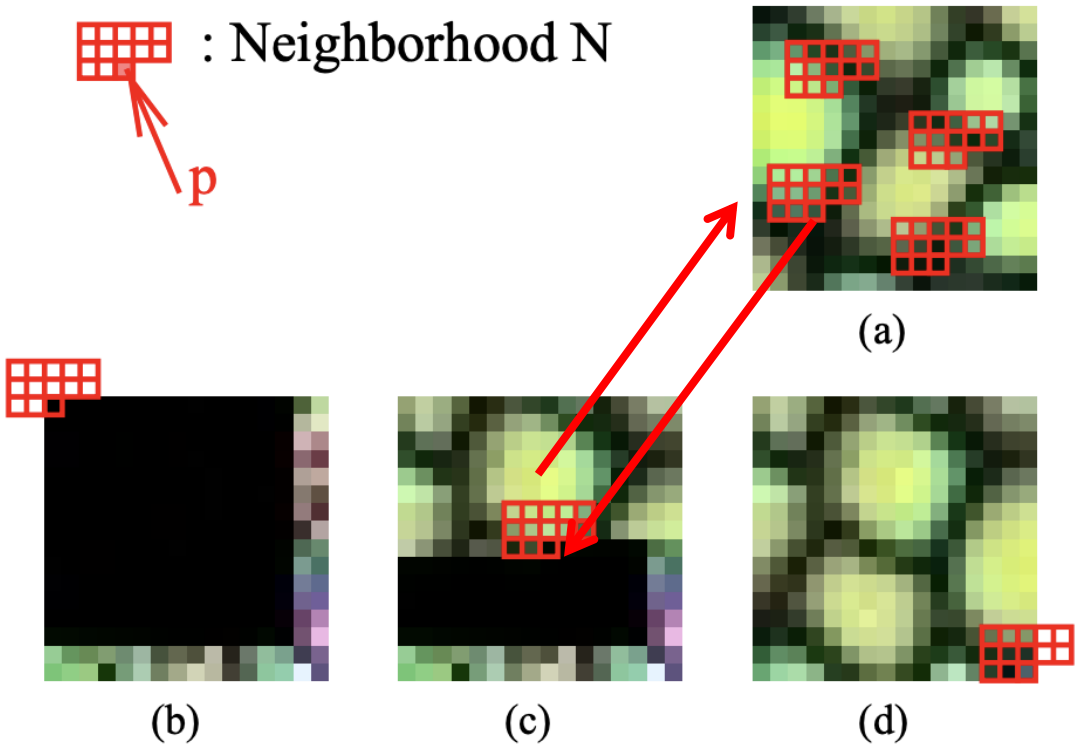
The generation process takes an example texture patch (left) and a random noise (middle) as input, and modifies this random noise to make it look like the given example texture.

The synthesized texture (right) can be of arbitrary size, and is perceived as very similar to the given example.

Texture is a kind of Style

Wei and Levoy, "Fast Texture Synthesis using Tree-structured Vector Quantization", SIGGRAPH 2000
Efros and Leung, "Texture Synthesis by Non-parametric Sampling", ICCV 1999

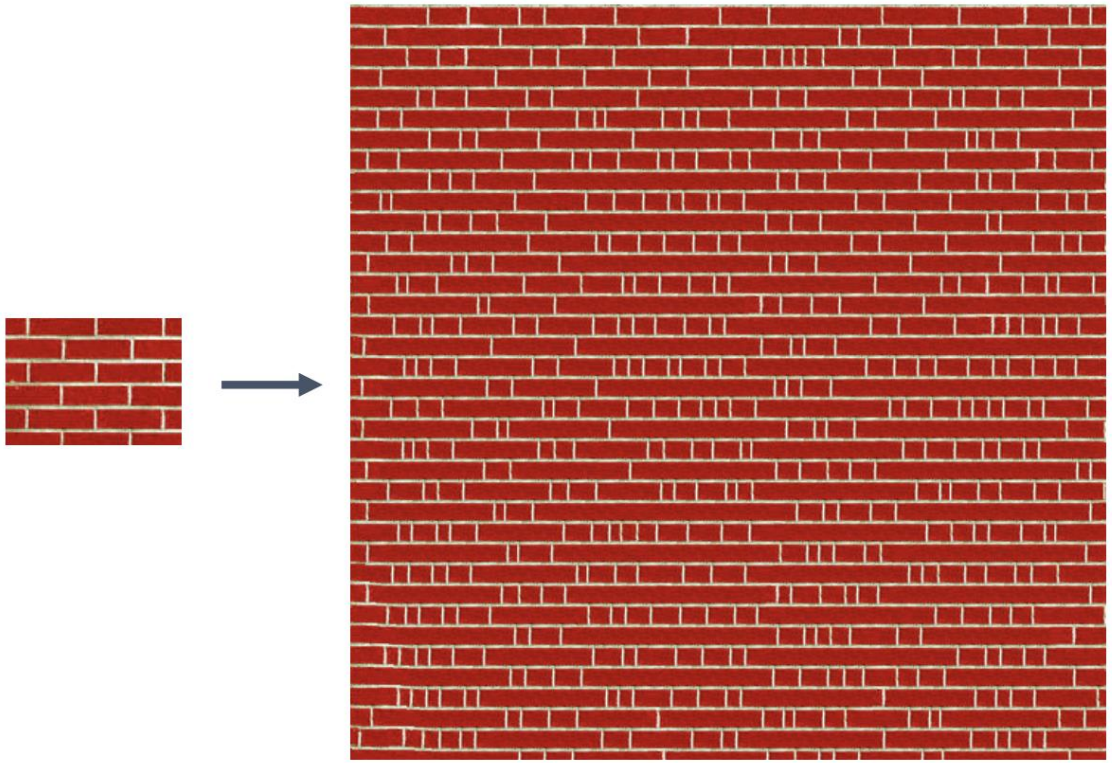
Texture Synthesis: Local-Based Method

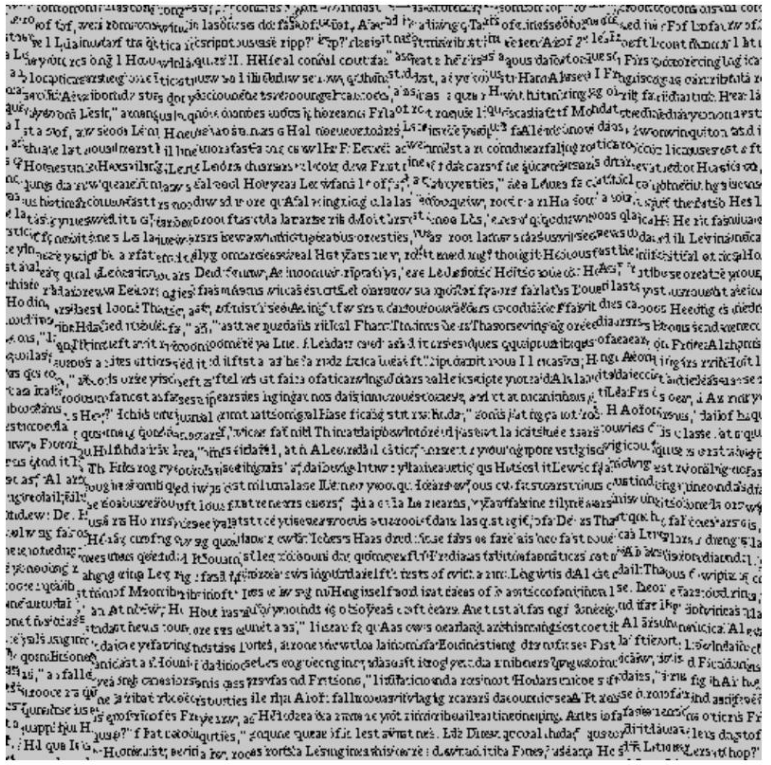


1. Given an example texture patch I_a and a white random noise I_s
2. Generate pixels one at a time in scanline order
3. To determine the pixel value p at I_s, its spatial neighborhood $N(p)$ (the L-shaped patch) is compared against all possible neighborhoods $N(p_i)$ from I_a. The input pixel p_i with the most similar $N(p_i)$ is assigned to p. Use L2 distance to measure similarity
4. The goal is to ensure that the newly assigned pixel p will maintain as much local similarity between I_s and I_a as possible. The same process is repeated for each output pixel until all the pixels are determined.

Wei and Levoy, "Fast Texture Synthesis using Tree-structured Vector Quantization", SIGGRAPH 2000
Efros and Leung, "Texture Synthesis by Non-parametric Sampling", ICCV 1999

Texture Synthesis: Local-Based Method





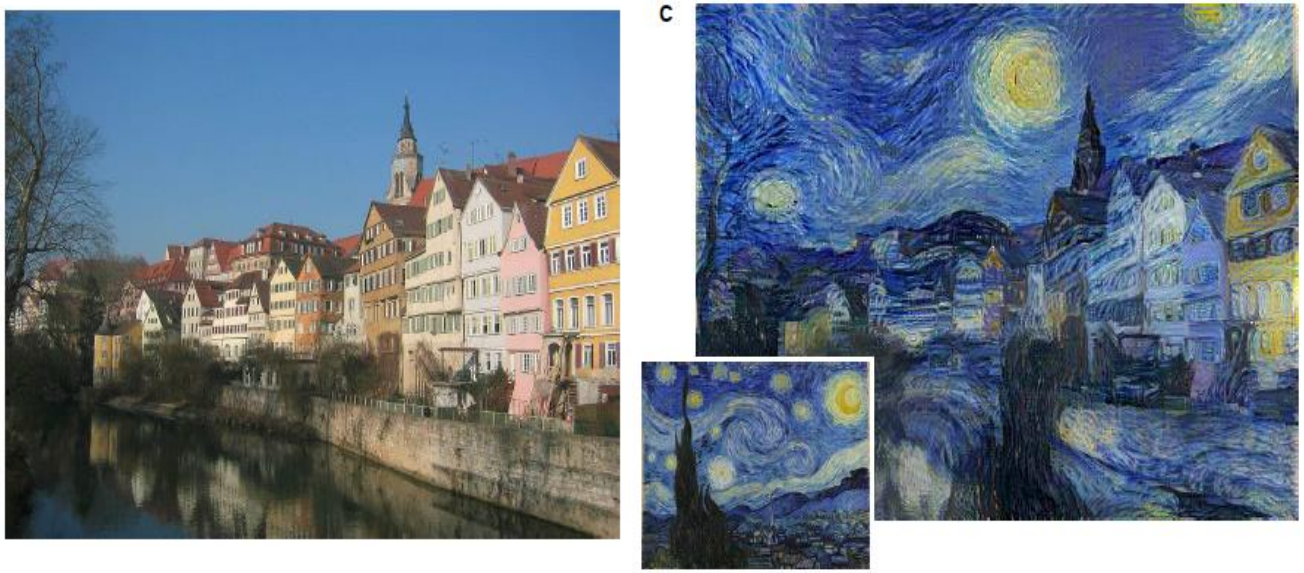
Style Transfer: Local-Based Method

1. Besides texture similarity, additionally consider the content similarity
2. Build correspondence maps to find content correspondences

Summary: style as local patches, making the result locally similar to the style image

Efros, Alexei A., and William T. Freeman. "Image quilting for texture synthesis and transfer." SIGGRAPH 2001.

Texture Synthesis: Global-based Method

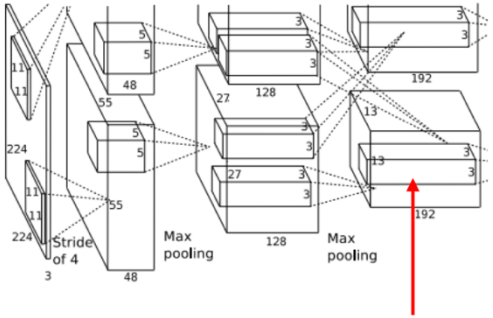


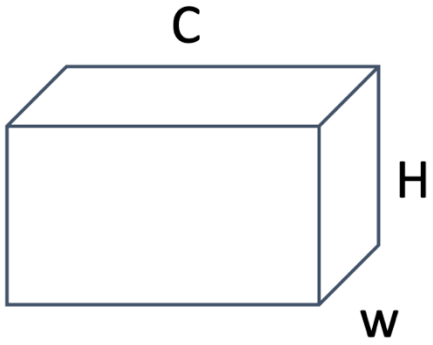
Able to create new patterns not seen in the style images

L. A. Gatys, A.S. Ecker, M. Bethge Texture synthesis using convolutional neural networks. NeurIPS 2015.
L. A. Gatys, A. S. Ecker, M. Bethge. Image style transfer using convolutional neural networks. CVPR 2016.

Texture Synthesis with Neural Networks: Gram Matrix

This image is in the public domain.

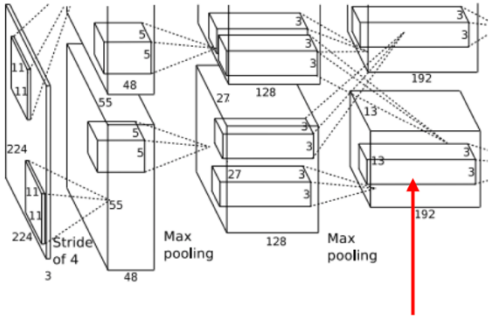


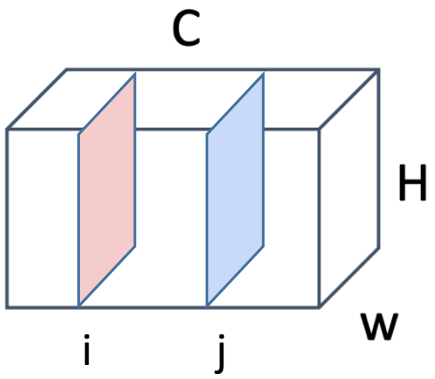


Each layer of CNN gives C x H x W tensor of features; Each channel is a H x W tensor

Texture Synthesis with Neural Networks: Gram Matrix

This image is in the public domain.





Channel i: HW-dimensional vector

$$G_{ij} = \mathbf{c}_i \times \mathbf{c}_j$$

Channel j: HW-dimensional vector

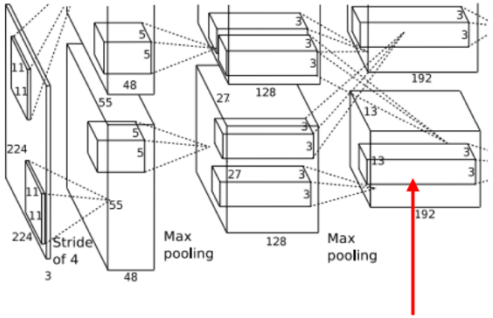
Each layer of CNN gives C x H x W tensor of features; Each channel is a H x W tensor,

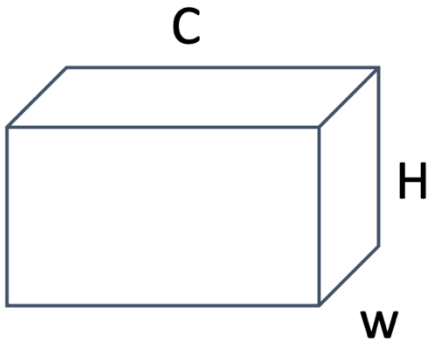
Which can be transform into a HW-dimensional vector

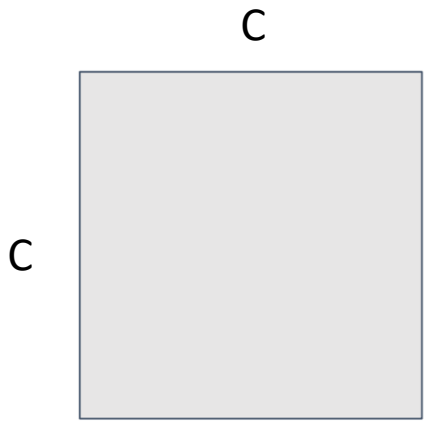
Inner product of two HW-dimensional vectors gives a value of a single element of the Gram Matrix

Texture Synthesis with Neural Networks: Gram Matrix

This image is in the public domain.







Gram Matrix

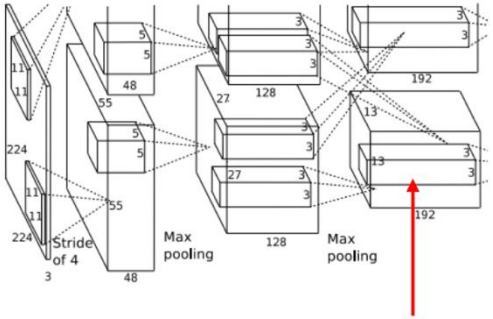
Each layer of CNN gives C x H x W tensor of features; Each channel is a H x W tensor,
Which can be transform into a HW-dimensional vector

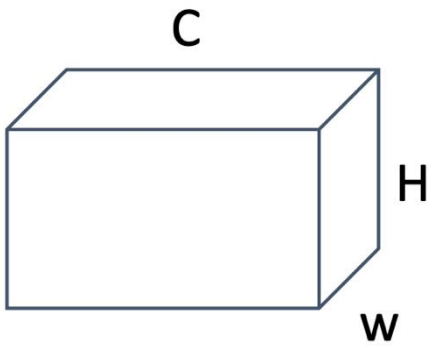
Inner product of two HW-dimensional vectors gives a value of a single element of the Gram Matrix

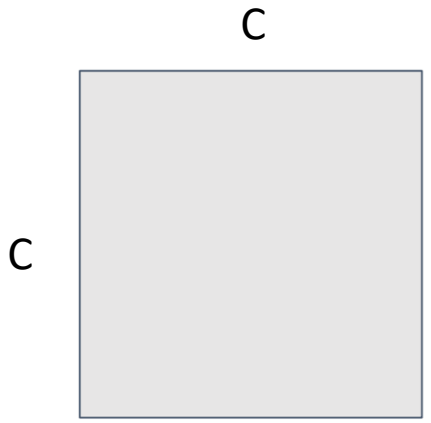
C vectors interact with each other, gives Gram Matrix of shape C x C giving unnormalized covariance

Texture Synthesis with Neural Networks: Gram Matrix

This image is in the public domain.





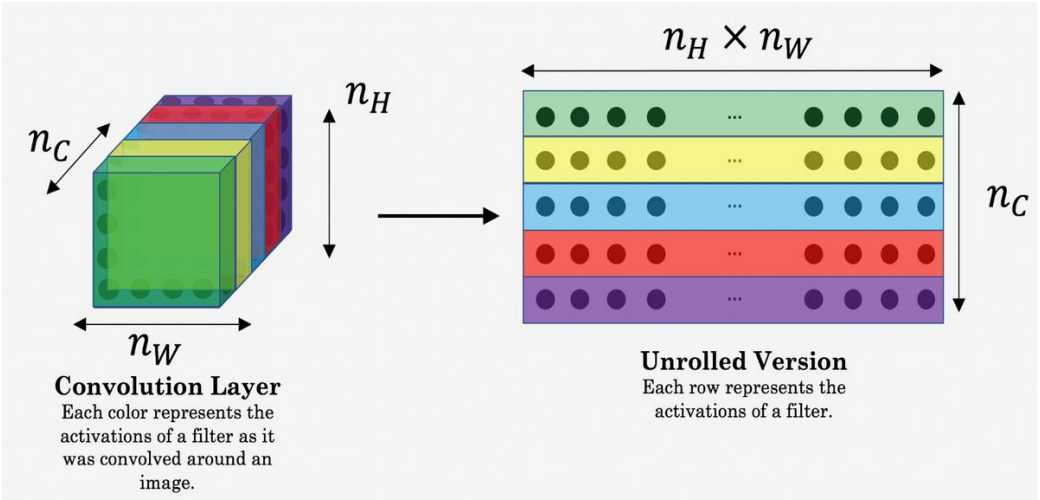


Gram Matrix

Efficient to compute; reshape features from

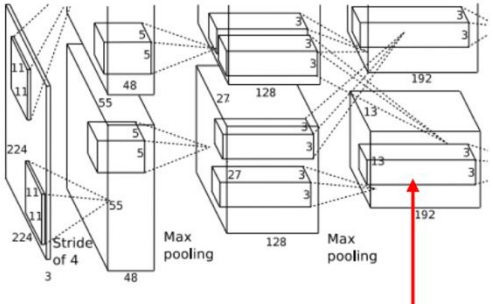
C x H x W to F = C x HW

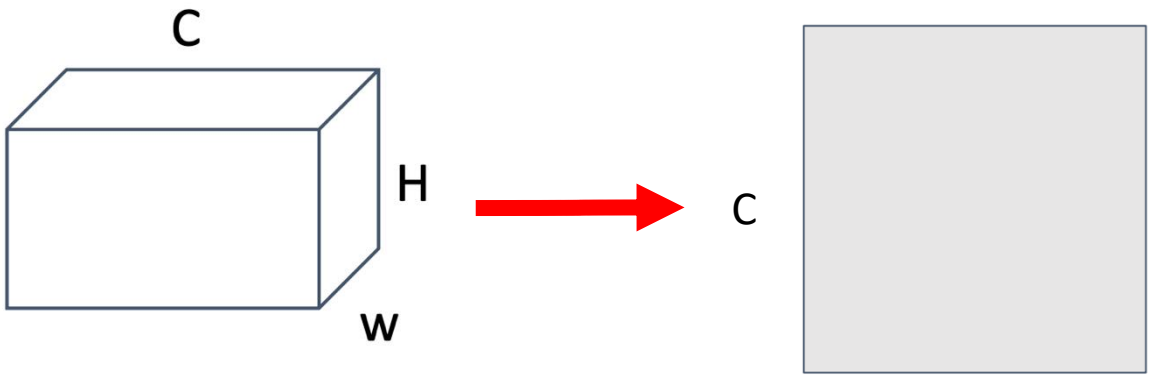
then compute G = FF^T



Texture Synthesis with Neural Networks: Gram Matrix

This image is in the public domain.



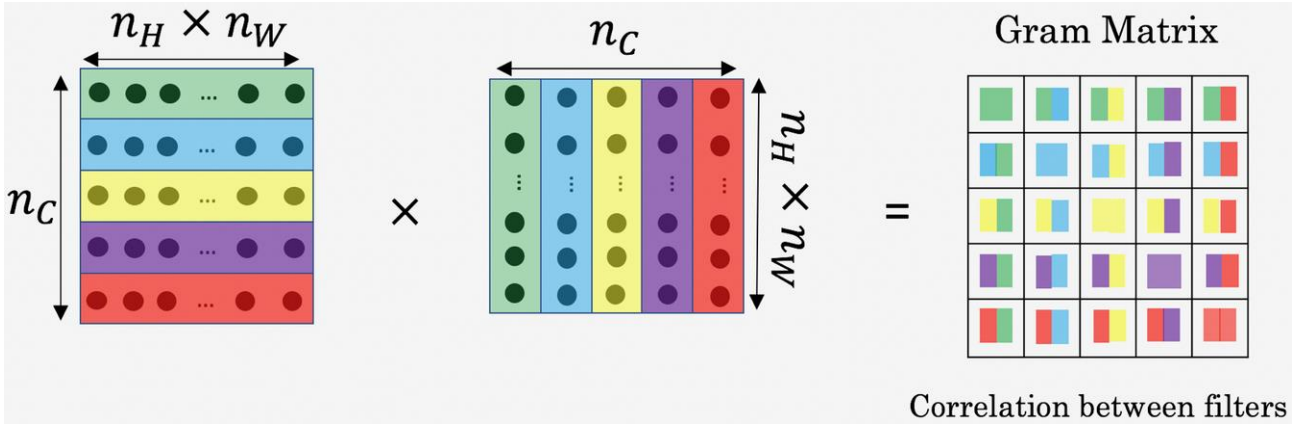


Gram Matrix

Efficient to compute; reshape features from

C x H x W to F = C x HW

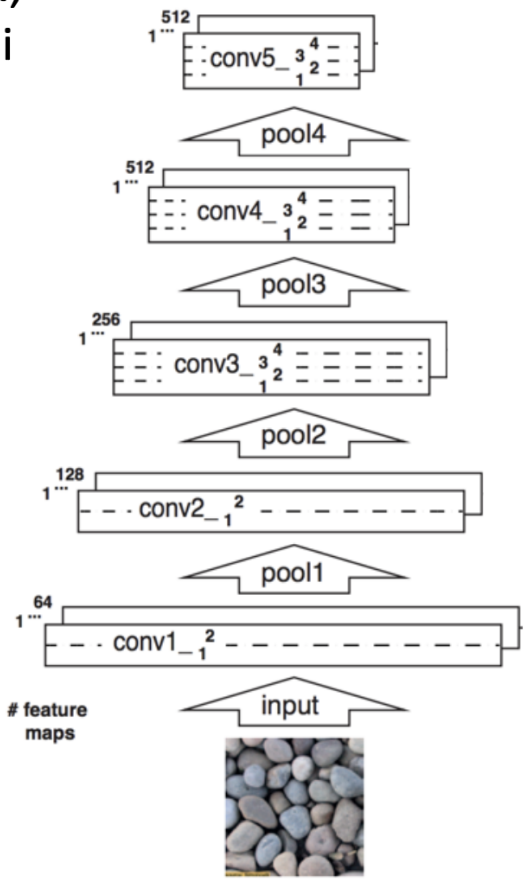
then compute G = FF^T



It captures the similarity between each vector. If two vectors are similar to one another, then their dot product will be large, and thus the Gram matrix will be large.

Neural Texture Synthesis

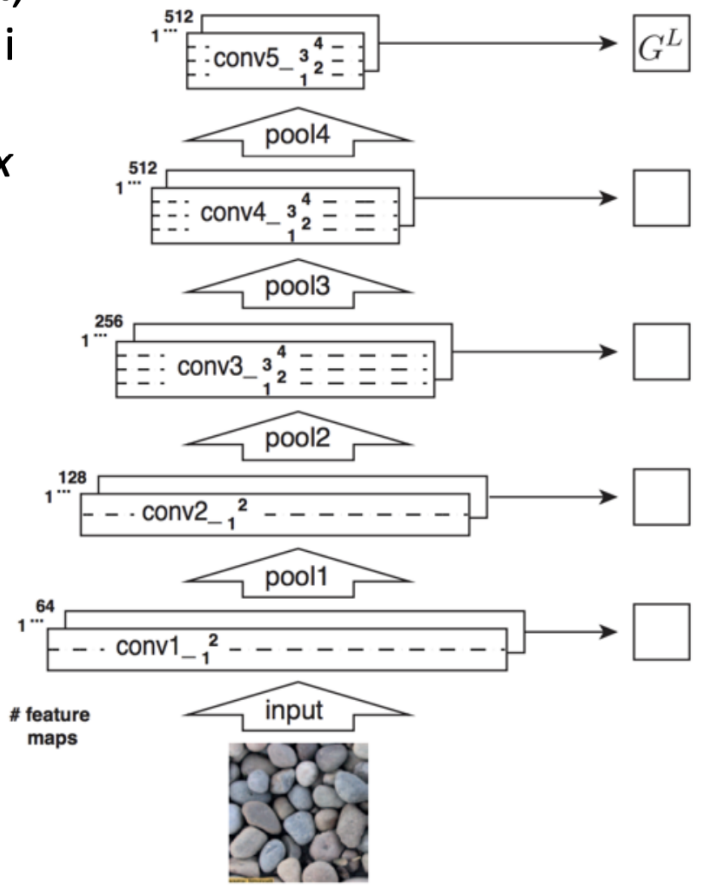
1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$



Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$ (shape $C_i \times C_i$)

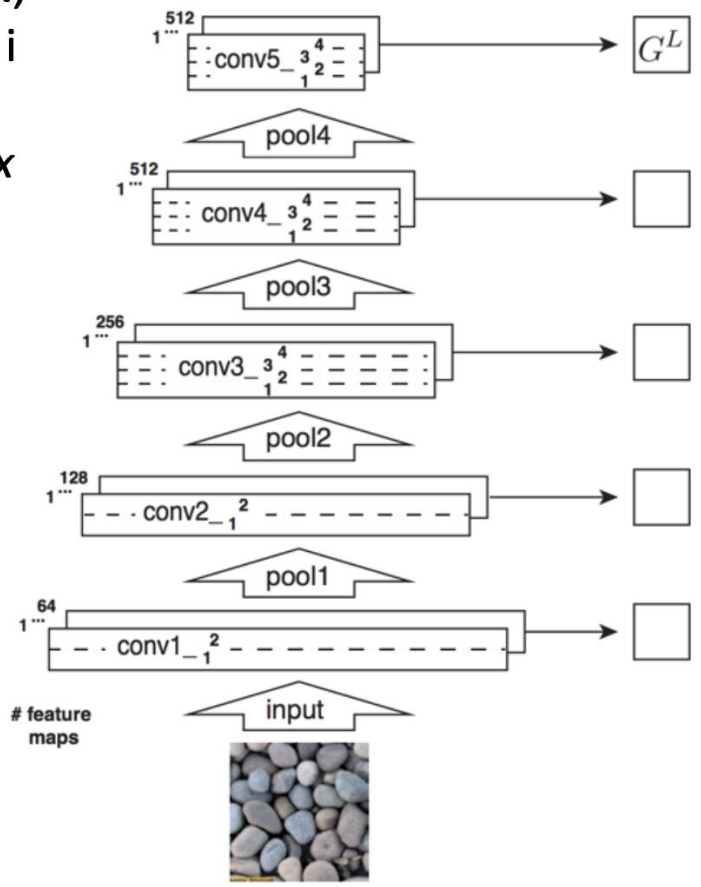


Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$ (shape $C_i \times C_i$)

4. Initialize generated image from random noise

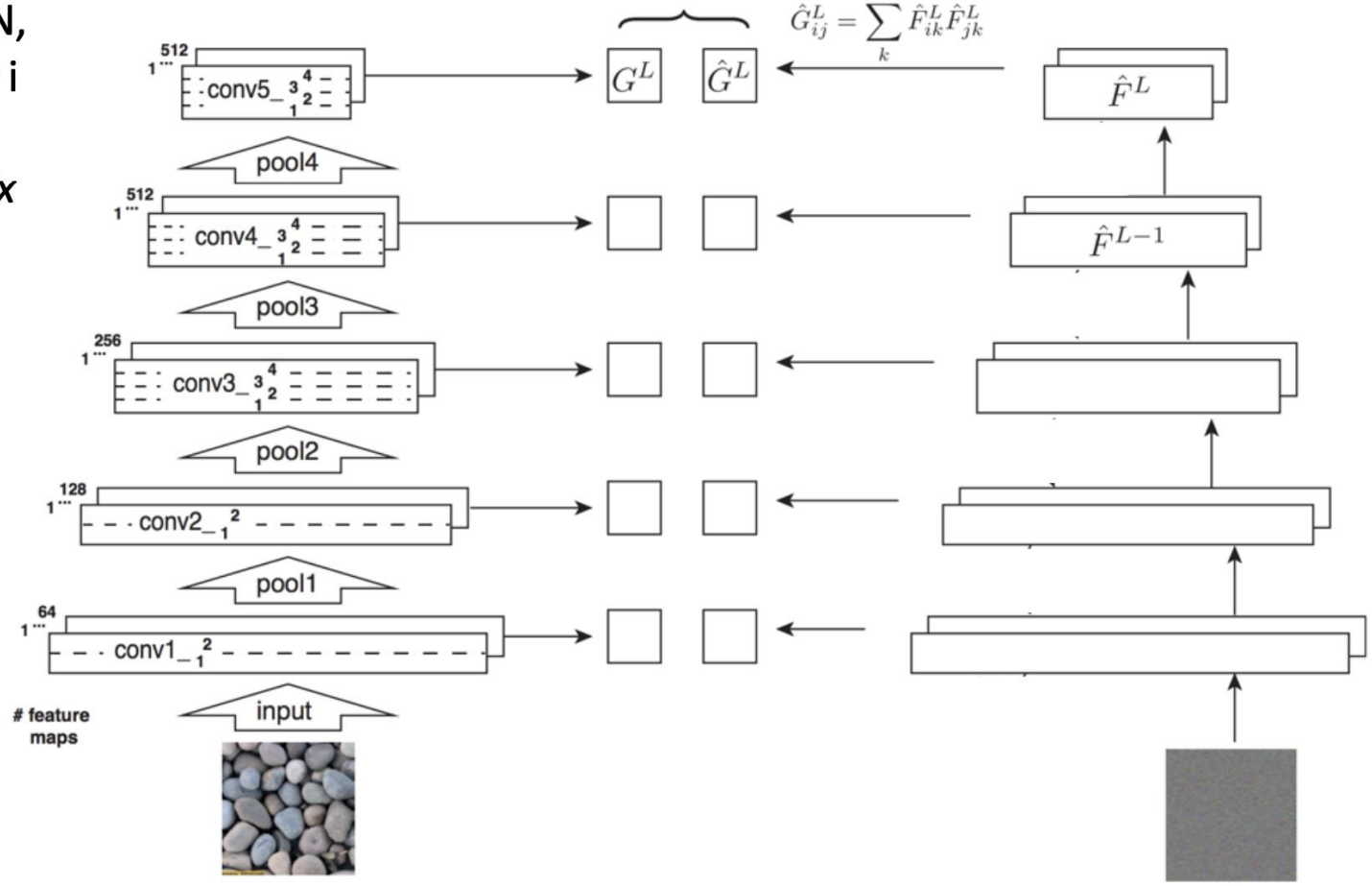


Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$ (shape $C_i \times C_i$)

4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer

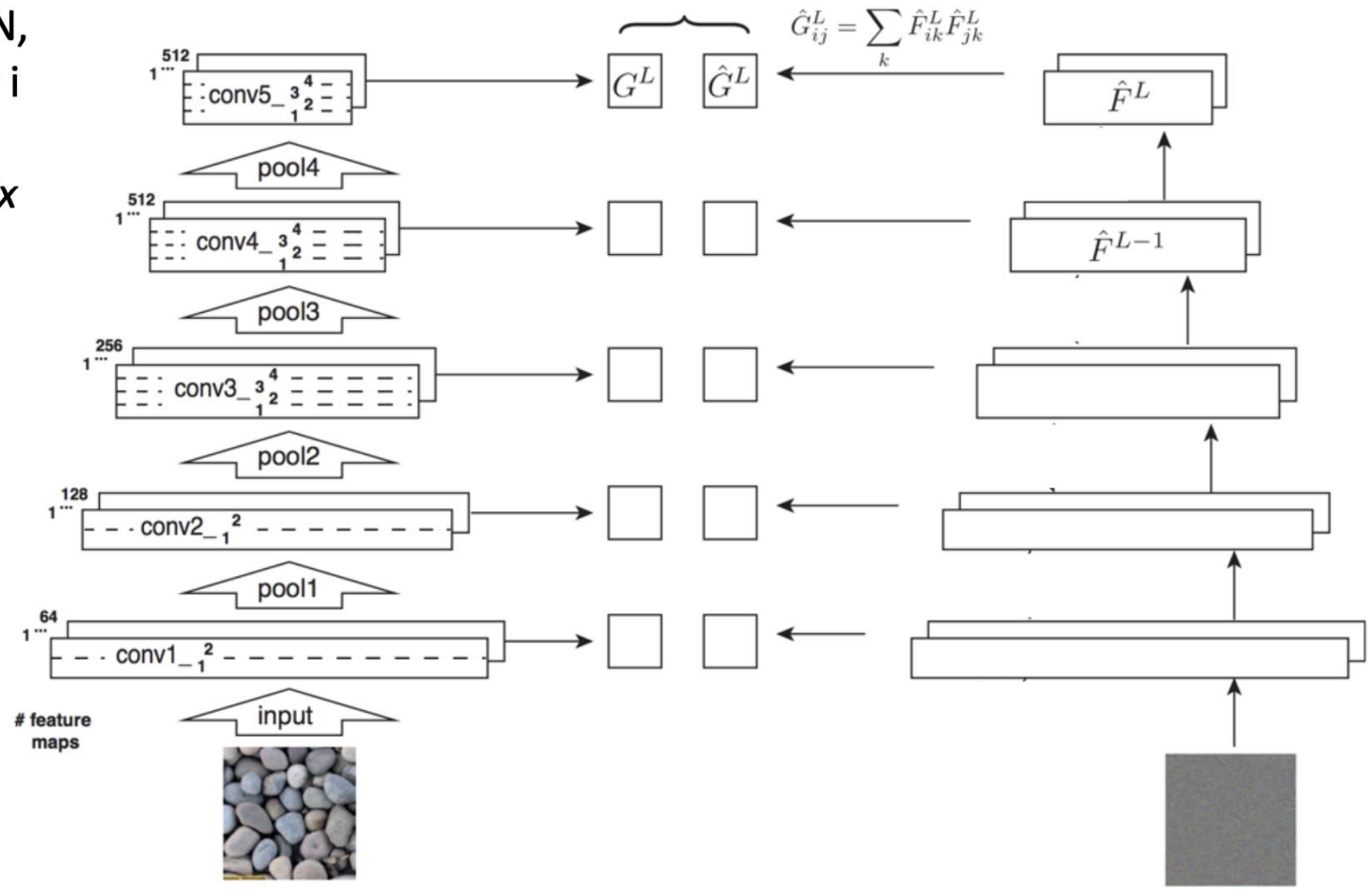


Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$ (shape $C_i \times C_i$)

4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices



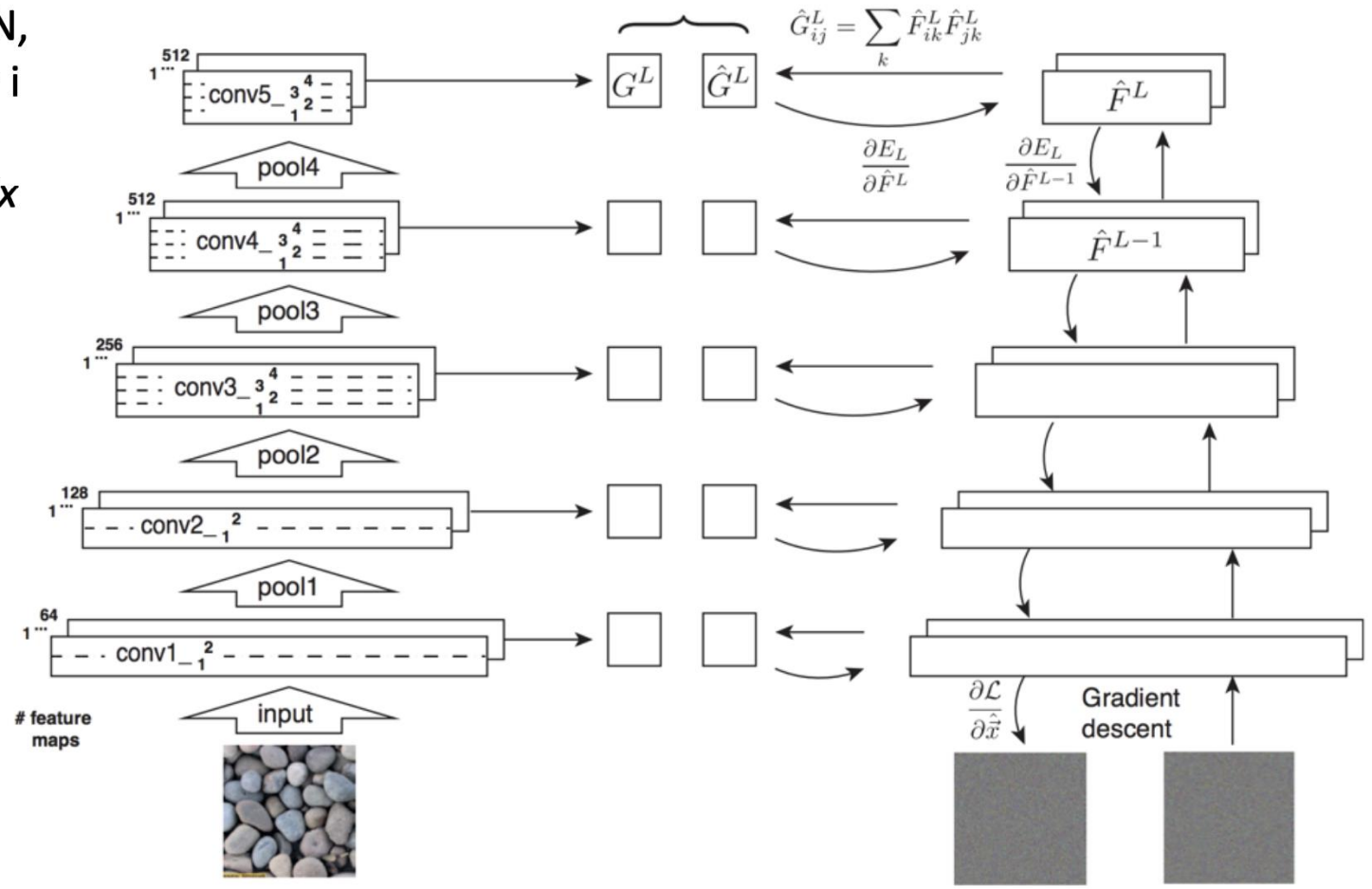
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G^l_{ij} - \hat{G}^l_{ij} \right)^2 \qquad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$$

Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$ (shape $C_i \times C_i$)

4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image



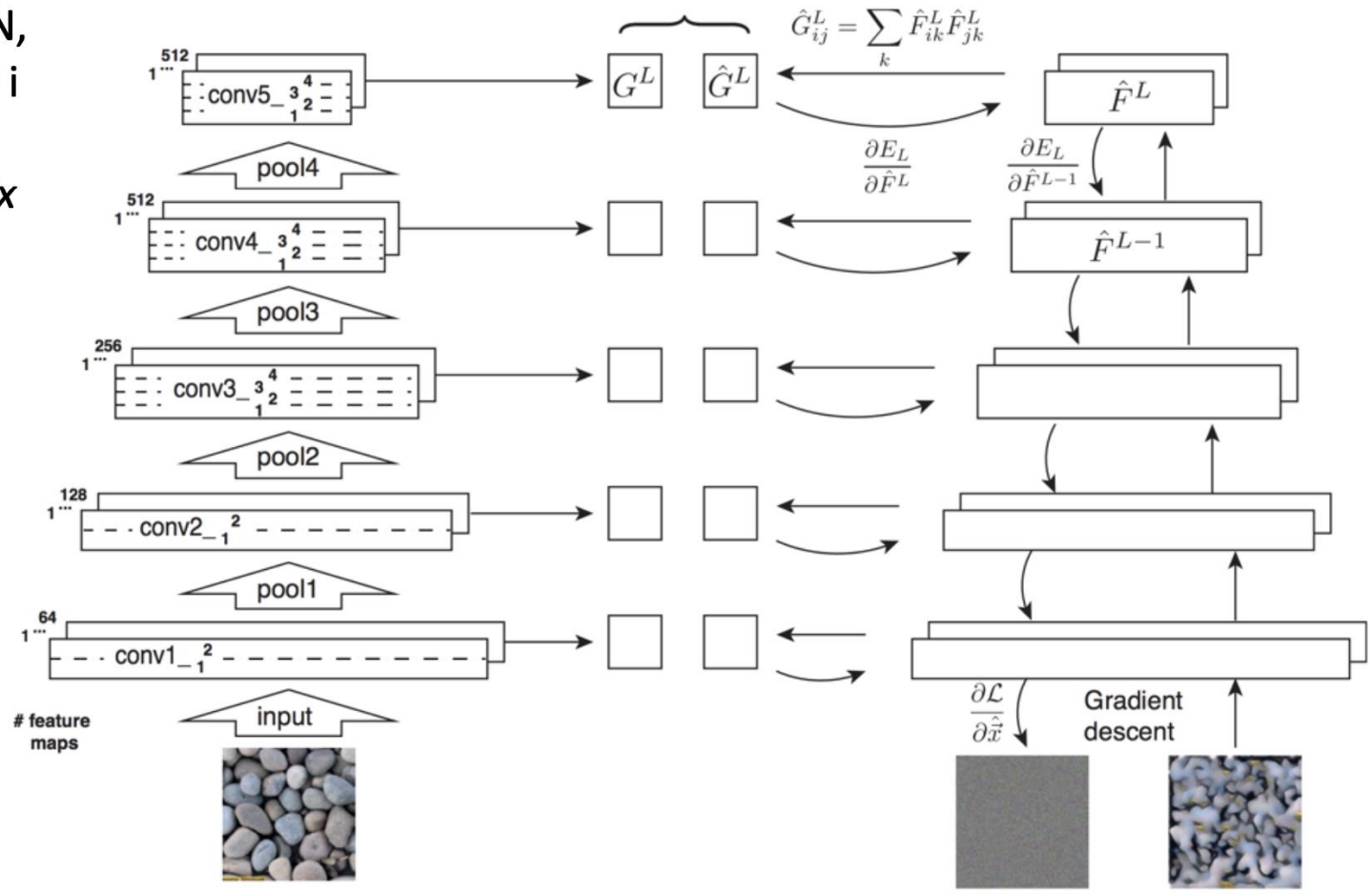
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G^l_{ij} - \hat{G}^l_{ij}\right)^2 \qquad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$$

Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$ (shape $C_i \times C_i$)

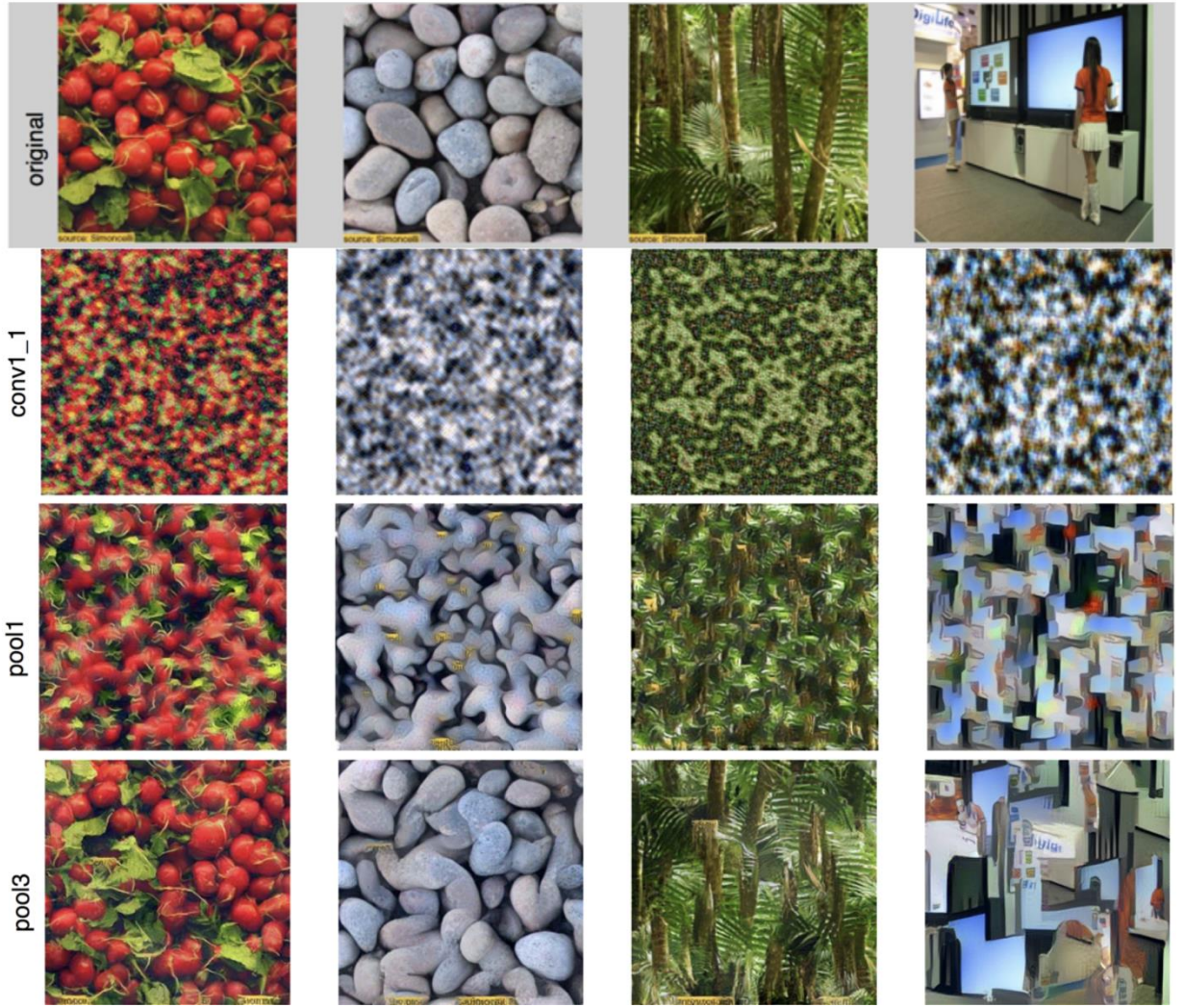
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5



$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G^l_{ij} - \hat{G}^l_{ij}\right)^2 \qquad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$$

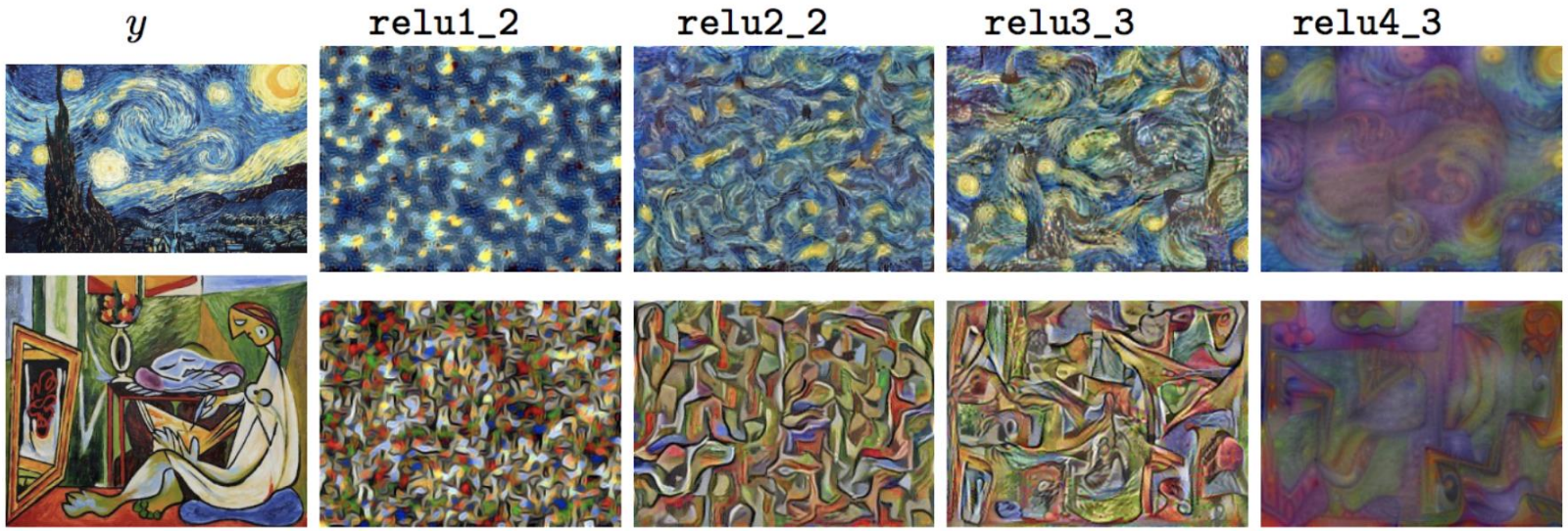
Neural Texture Synthesis

Reconstructing texture from higher layers recovers larger features from the input texture



Neural Texture Synthesis: Texture = Artwork

Texture synthesis (Gram reconstruction)



Neural Style Transfer: Feature + Gram Reconstruction

Matching G

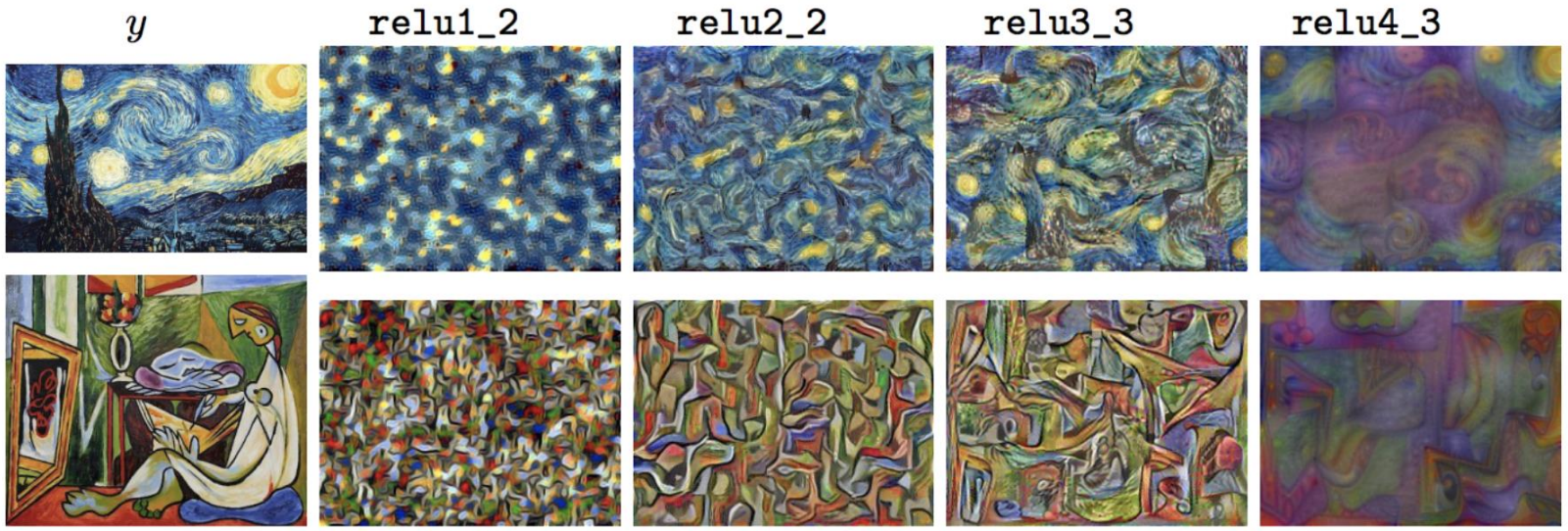
Texture synthesis (Gram reconstruction)

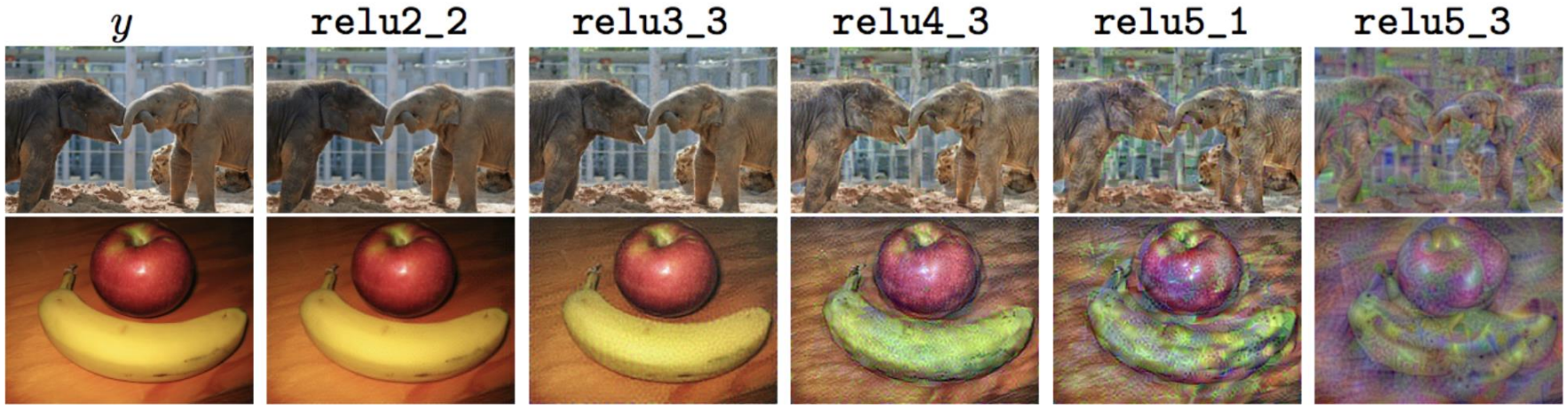
C x H x W to F = C x HW

then compute G = FF^T

Feature reconstruction

Matching F





Neural Style Transfer

Content Image

This image is licensed under CC-BY 3.0

+

Style Image

Starry Night by Van Gogh is in the public domain

=

Output Image

Match features from content image and Gram matrices from style image

Neural Style Transfer

Content Image

This image is licensed under CC-BY 3.0

+

Style Image

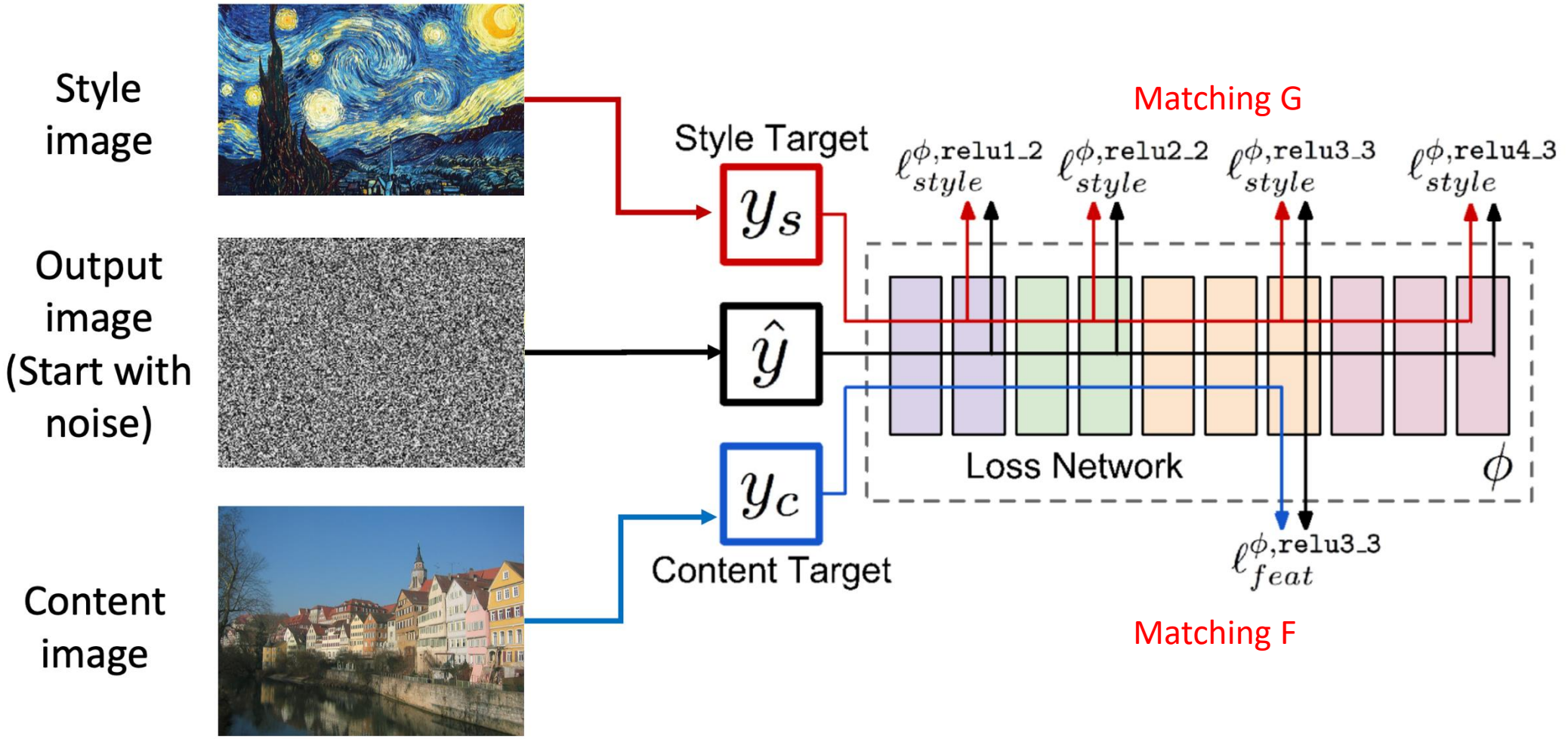
Starry Night by Van Gogh is in the public domain

=

Output Image

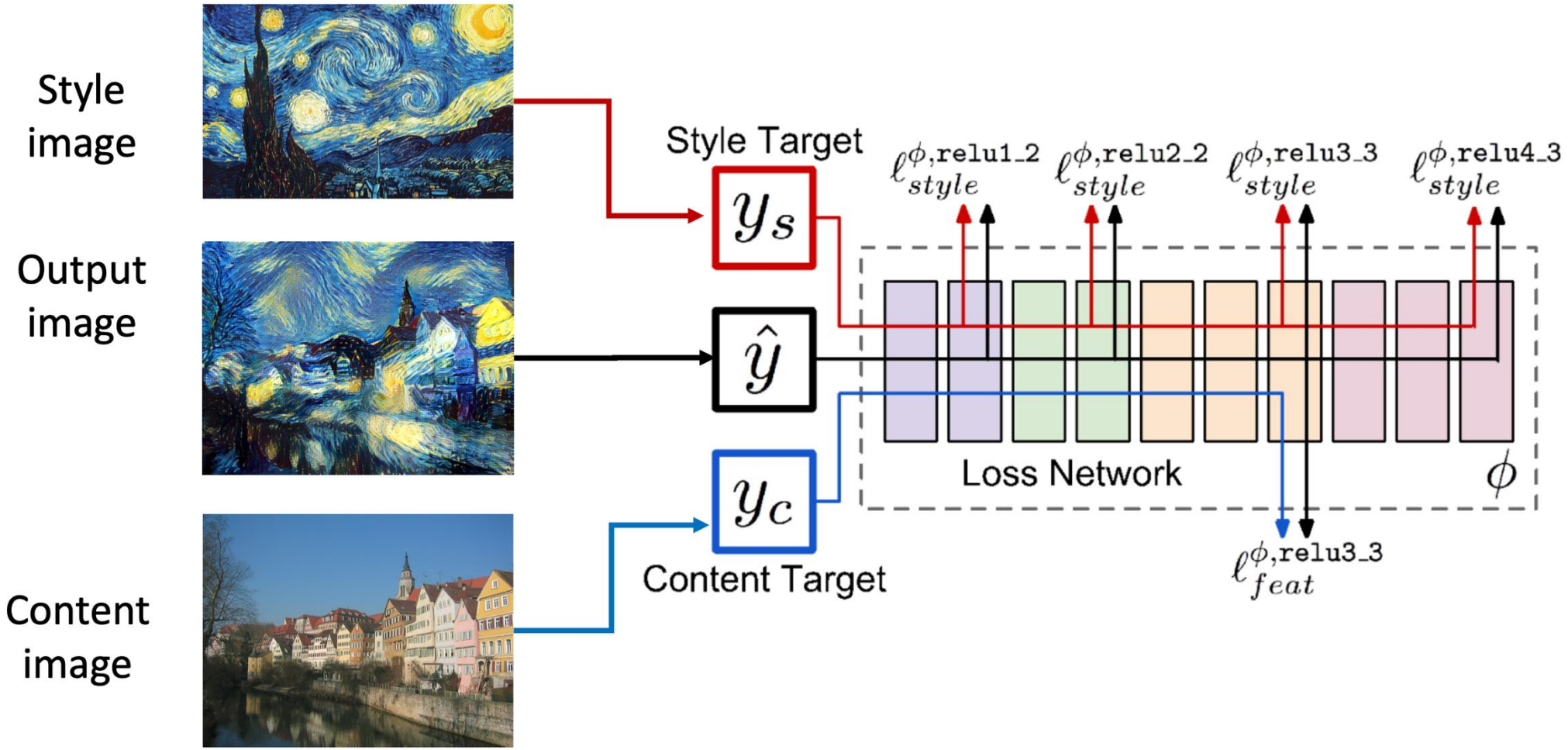
This image copyright Justin Johnson, 2015. Reproduced with permission.

Neural Style Transfer



Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016
Figure adapted from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016.

Neural Style Transfer



Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016
Figure adapted from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016.

Neural Style Transfer

- Usually we start from the content image.
- The results better preserve the structures, and have less artifacts

Neural Style Transfer

Gatys, Ecker, and Bethge, “Image style transfer using convolutional neural networks”, CVPR 2016 Figure copyright Justin Johnson, 2015.

Neural Style Transfer

More weight to content loss

More weight to style loss

Neural Style Transfer: Multiple Style Images

Mix style from multiple images by taking a weighted average of Gram matrices








Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016

Fast Neural Style Transfer

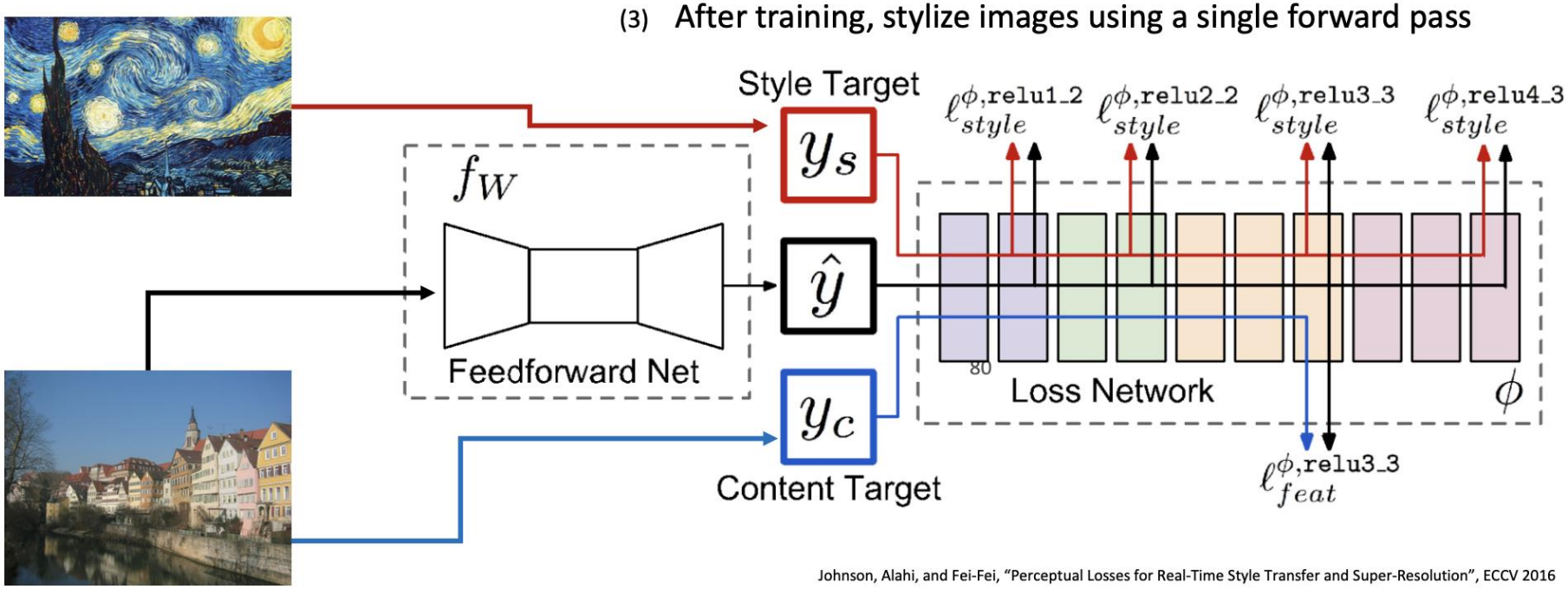
Problem: Style transfer requires many forward / backward passes through VGG; very slow!

Style transfer process is actually a training process

Solution: Train another neural network to perform style transfer for us!

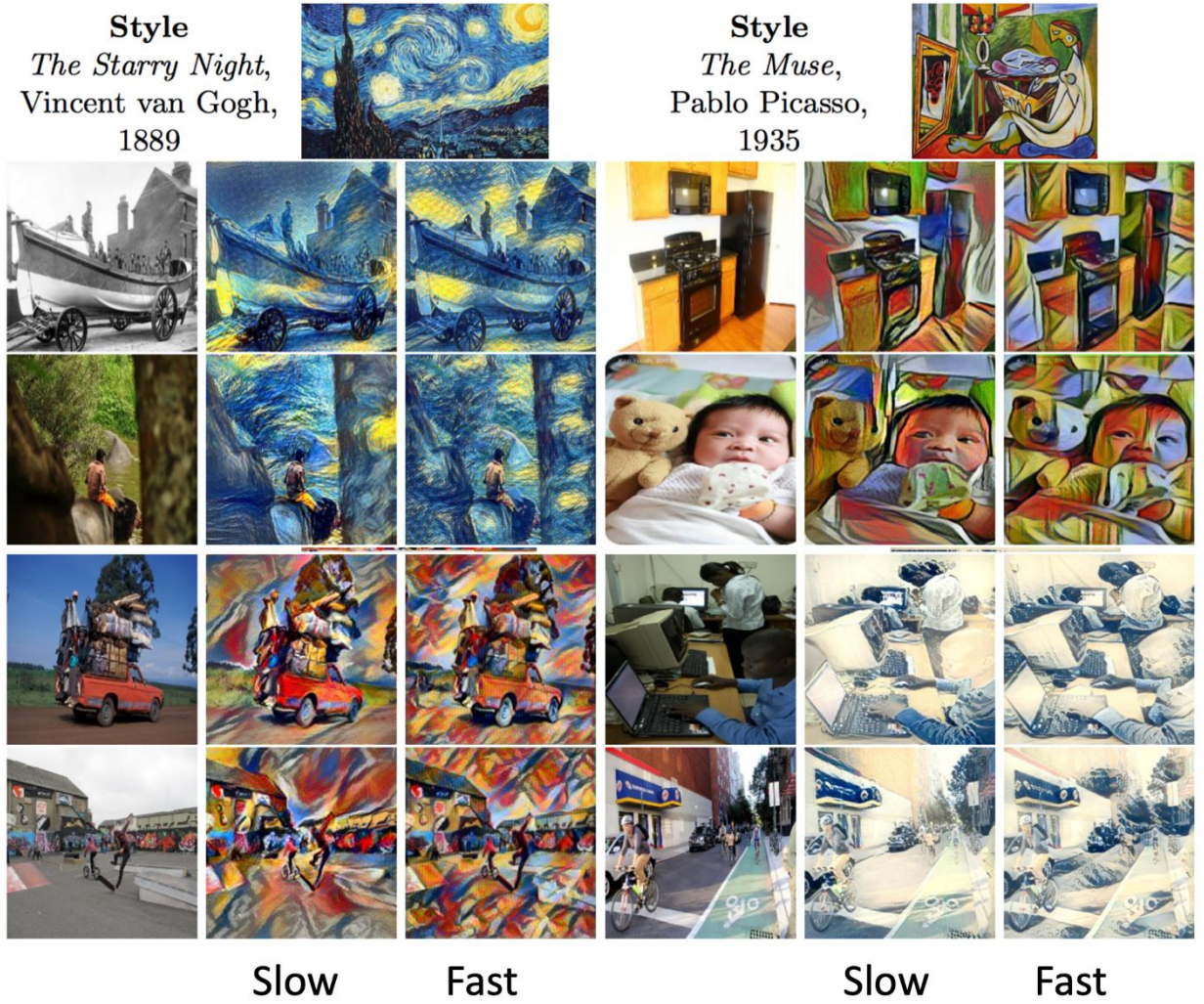
f_W: the same as CycleGAN

(1) Train a feedforward network for each style
(2) Use pretrained CNN to compute same losses as before
(3) After training, stylize images using a single forward pass



Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016

Fast Neural Style Transfer

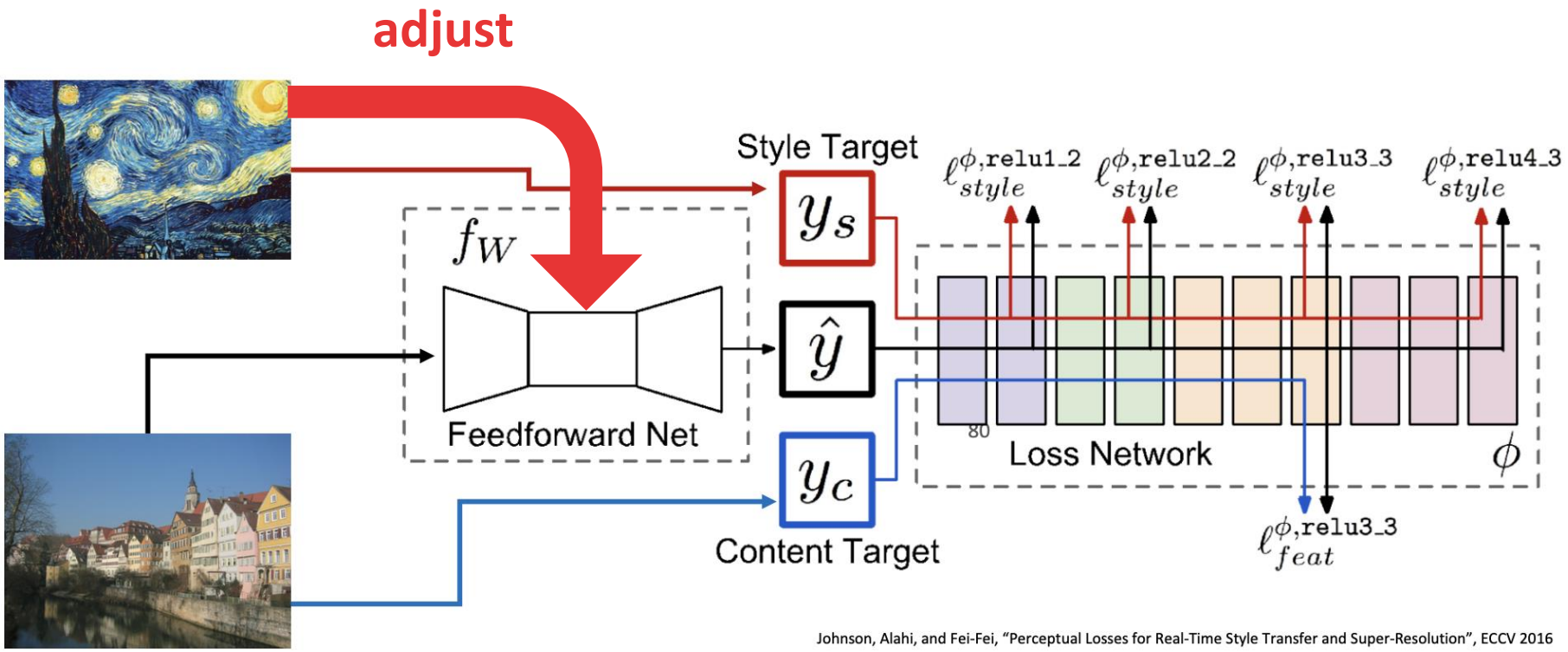


Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016

AdaIN Style Transfer

Problem: Per-Style-Per-Model: Each network only learns a single style.

Solution: AdaIN → use the statistics of the style image to adjust the feature

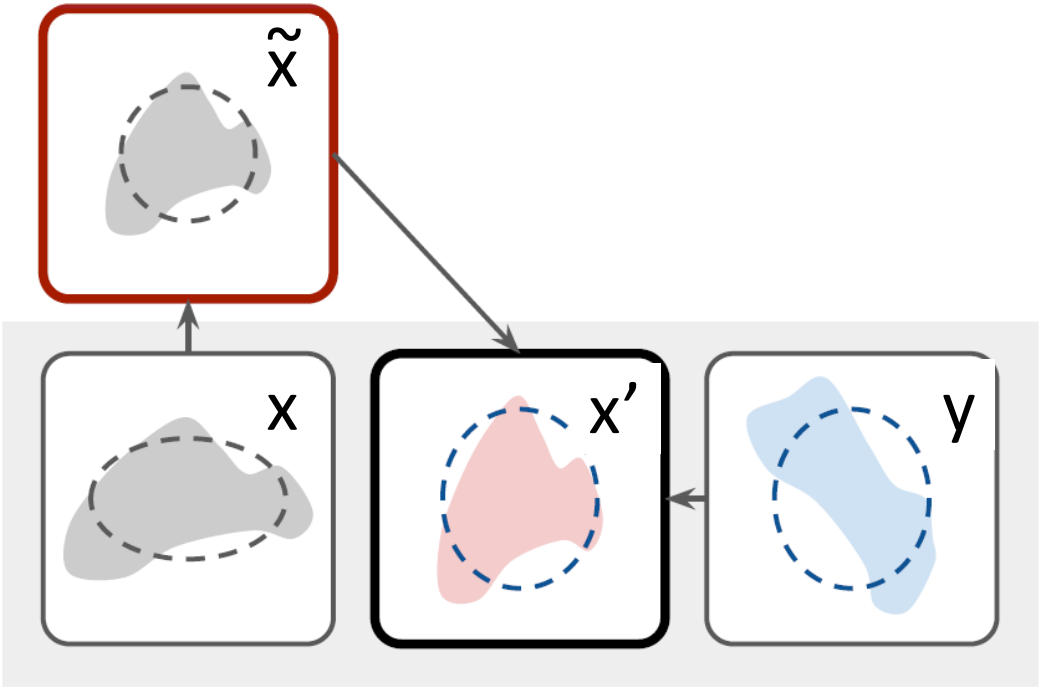


Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016

AdaIN Style Transfer

- Recall Gram-matrix
  - Matching (unnormalized) covariance between channels
  - Global distribution of features
- Can we use simpler distributions?
  - Mean and variance
  - Channel-wise mean μ & standard deviation σ

$$x' = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$



Huang, Xun, and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. ICCV. 2017.

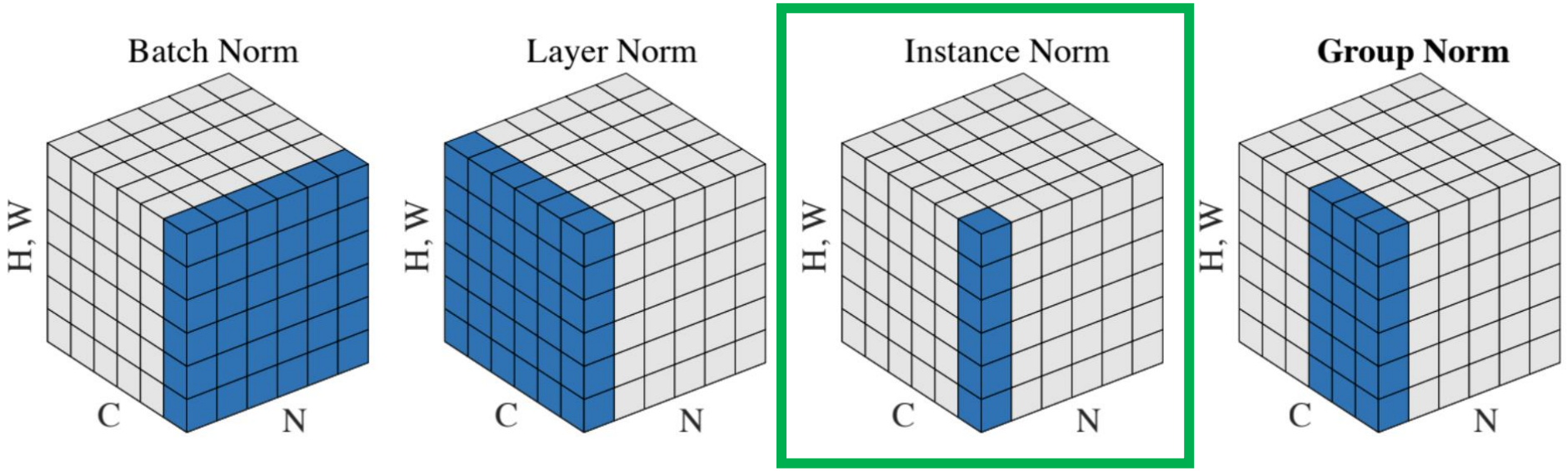
Sheng, Lu, et al. Avatar-net: Multi-scale zero-shot style transfer by feature decoration. CVPR. 2018.

AdaIN Style Transfer

- Can we use more simple distributions?

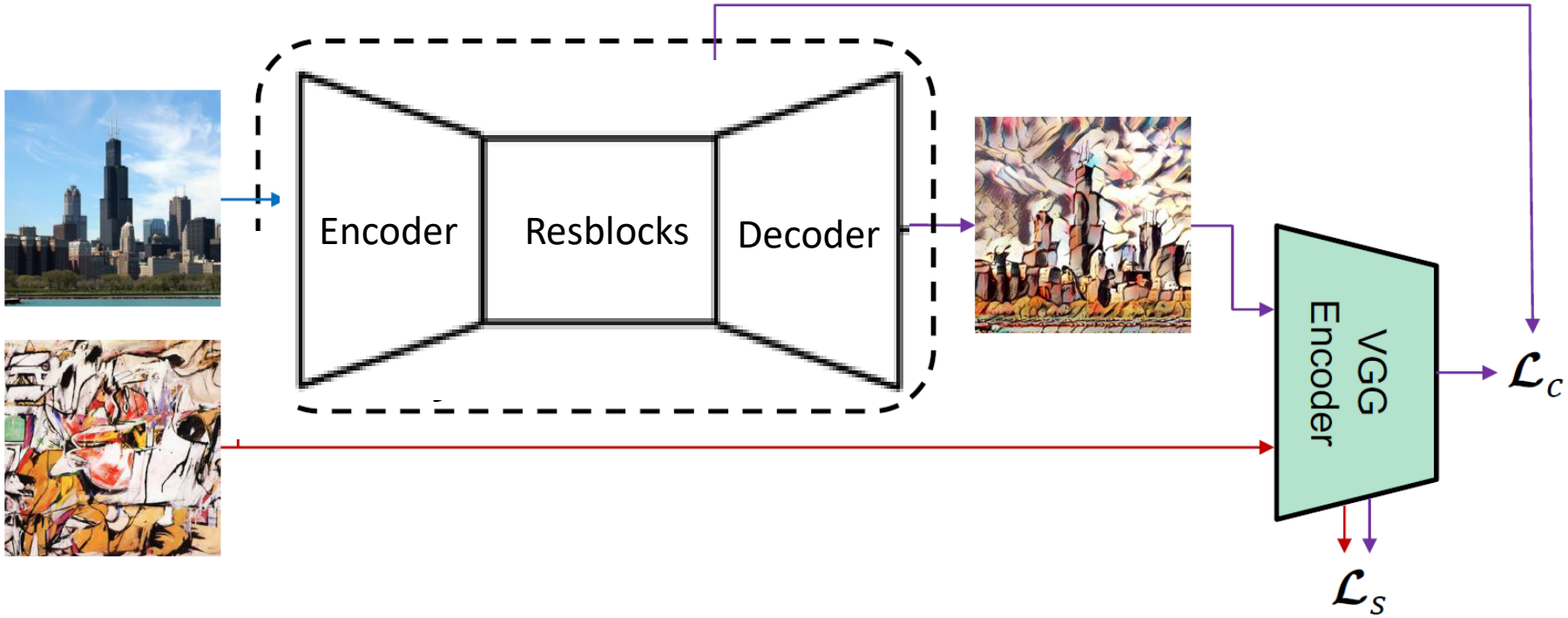
$$x' = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y) \quad \rightarrow \quad \mathrm{AdaIN}(x, y) = \sigma(y)\mathrm{IN}(x) + \mu(y)$$

Instance Normalization was developed for style transfer!
Replacing BN with IN improves results



AdaIN Style Transfer

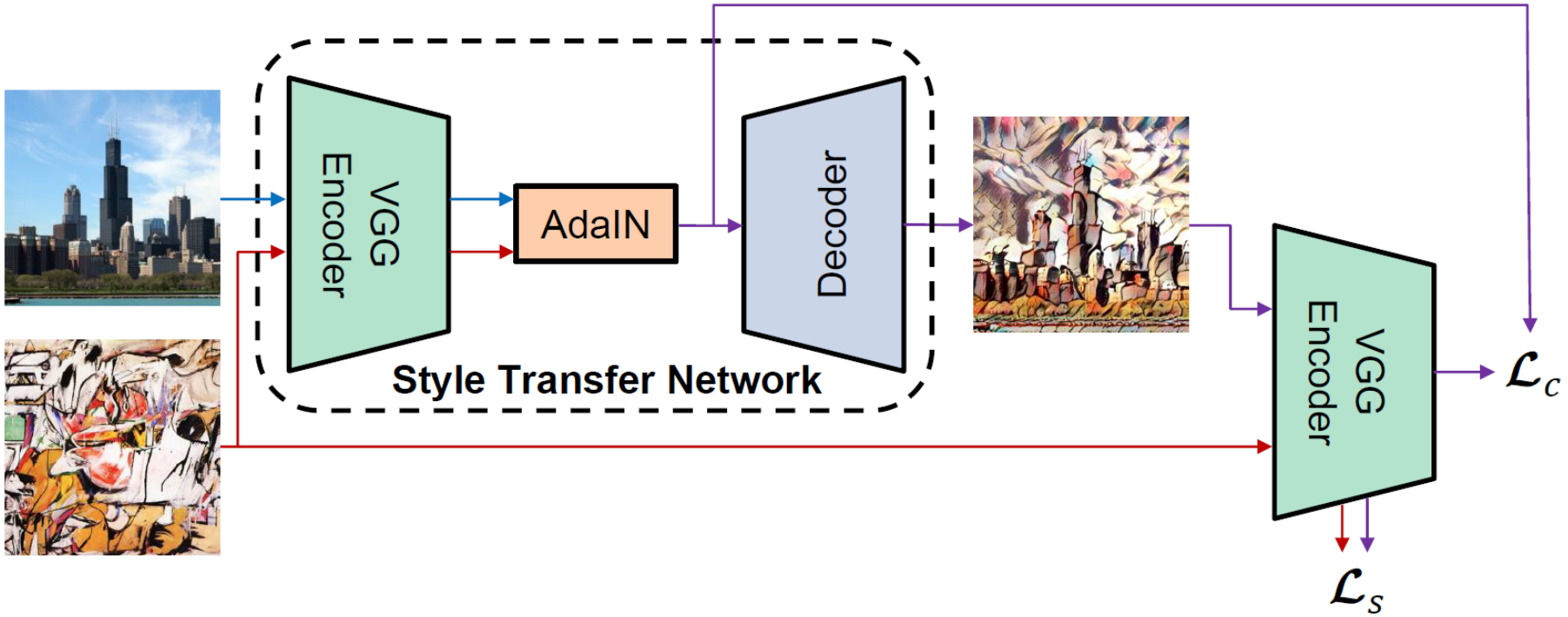
- Recall Fast Neural Style Transfer
  - Encoder-Transformer (Resblocks)-Decoder
  - Also used in CycleGAN



Huang, Xun, and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. ICCV. 2017.

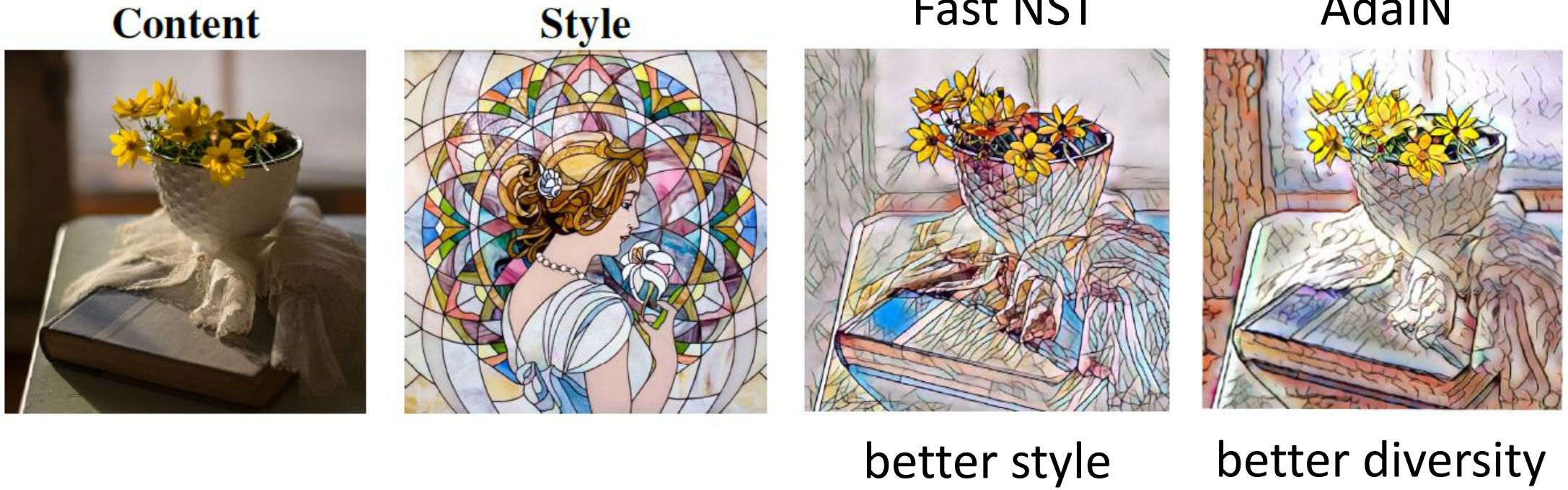
AdaIN Style Transfer

- AdaIN Style Transfer
- Replace Resblocks with AdaIN
- Style loss: matching Gram → matching mean μ & standard deviation σ



Huang, Xun, and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. ICCV. 2017.

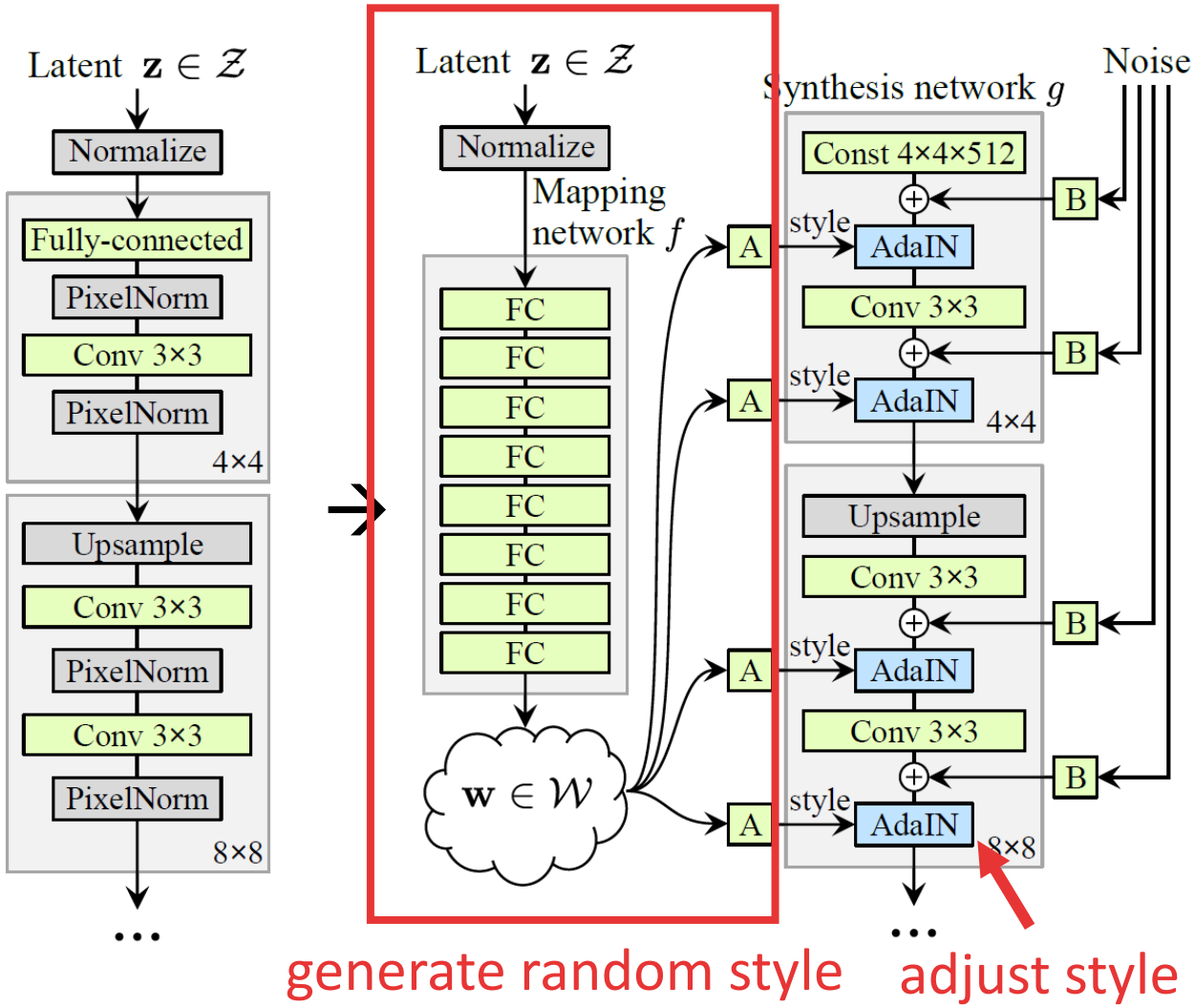
AdaIN Style Transfer



Jing, Yongcheng, et al. Stroke controllable fast style transfer with adaptive receptive fields. ECCV2018.

StyleGAN

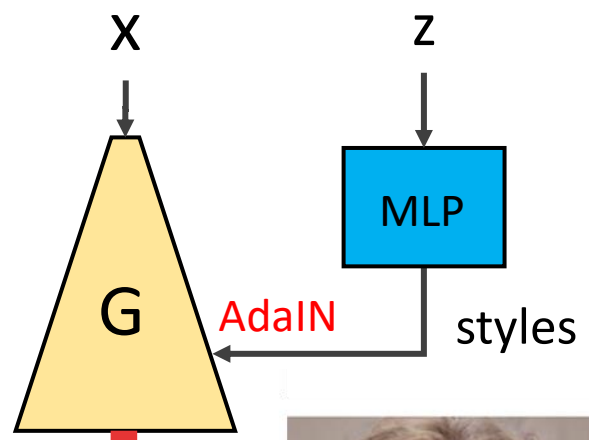
- Neural Style Transfer
  - Texture / color as style
  - Can we model more semantical and structured styles like hair color, face shapes, face expressions?
- Revisit StyleGAN in Lecture 8
  - Add the idea of style transfer in GAN
  - Random input → Fixed input
    - Fixed content features
  - Add random style inputs
    - Randomly stylize the content feature

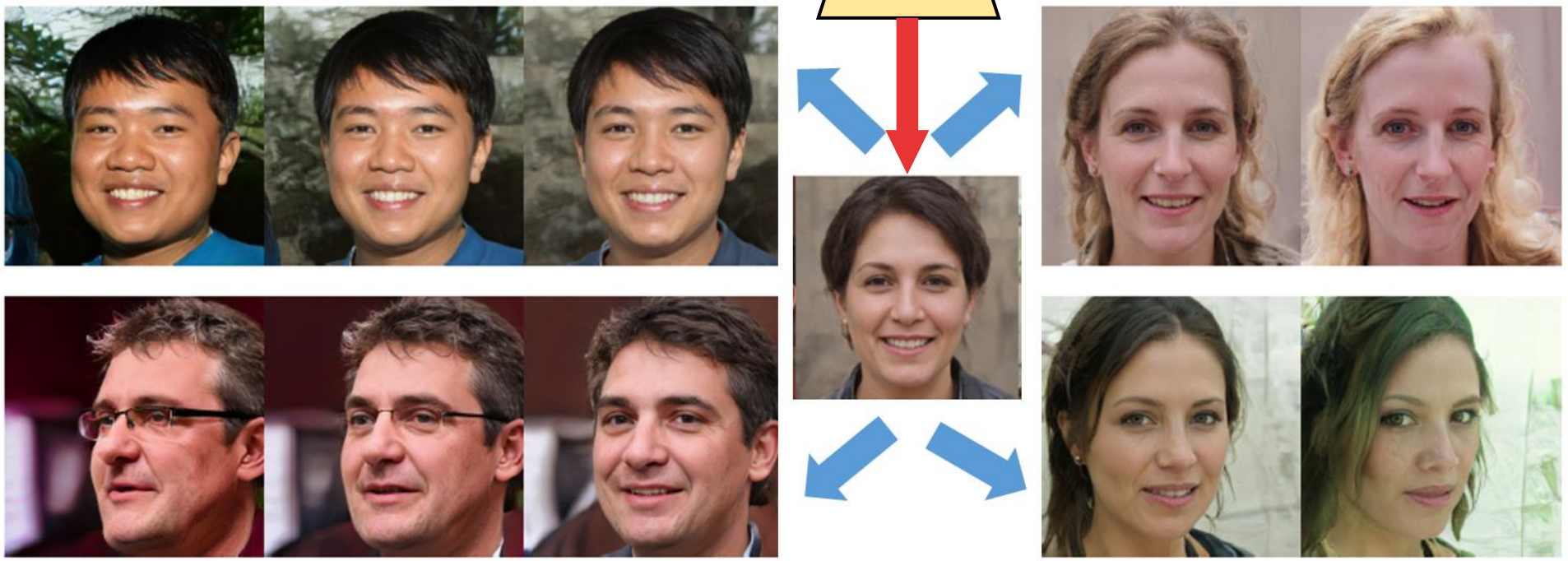


Karras, Tero, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. CVPR 2019.

StyleGAN

- Revisit StyleGAN in Lecture 8
  - Fixed x; zeros styles → mean face
  - Increase style → different pose/color/expression
  - StyleGAN learns a mean face and stylizes this face with random styles for diverse face generation

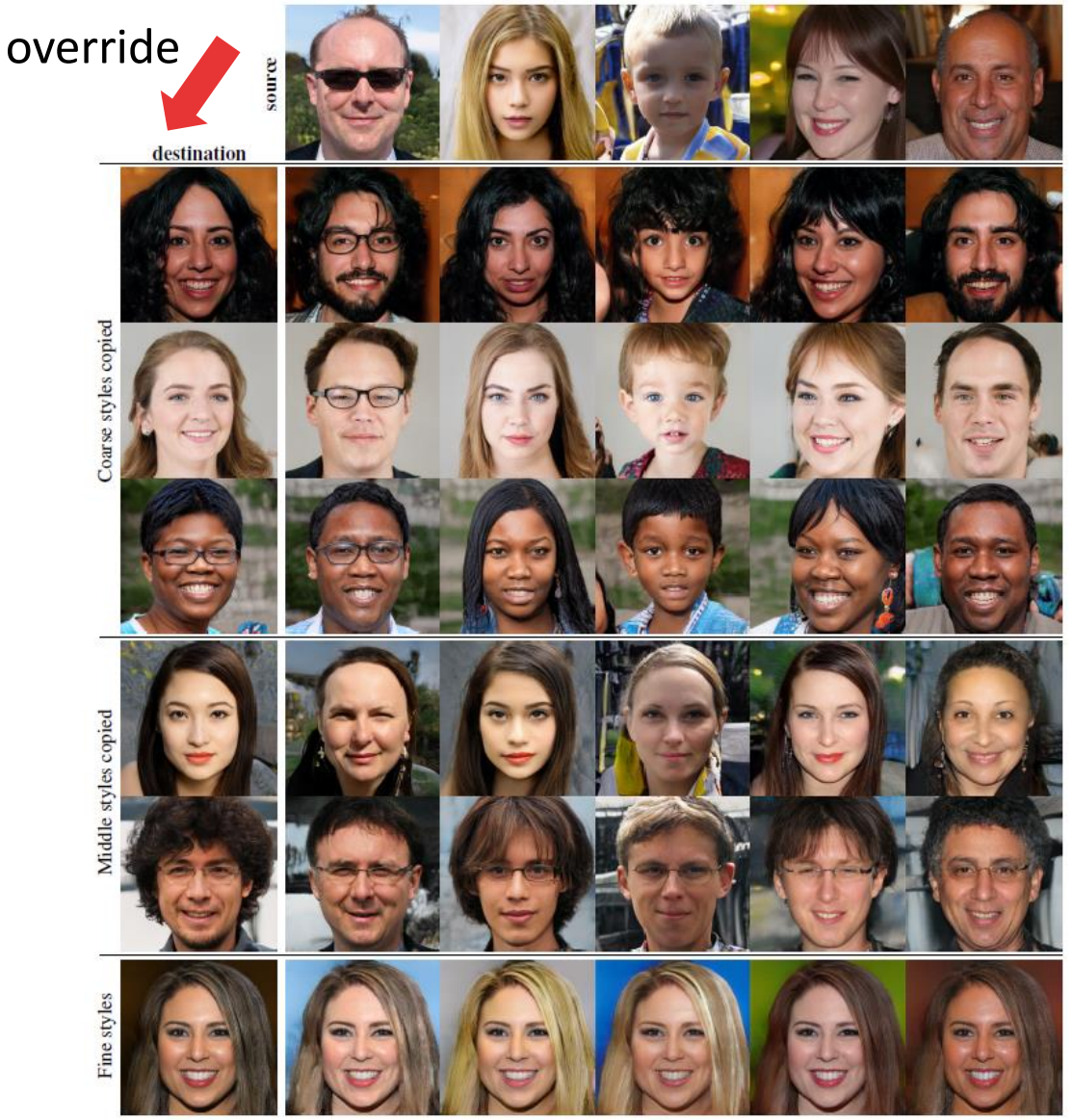




Karras, Tero, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. CVPR 2019.

StyleGAN

- Hierarchical Style
  - Shallow layers
    - Gender
    - Pose
    - Expression
  - Middle layers
    - Hair style
    - Eyes open/closed
    - Appearance
  - Deep layers
    - Color
- Style fusion



Karras, Tero, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. CVPR 2019.

StyleGAN

Problem: All we do is to play with the randomly generated face images.

Can we edit real-world face images?

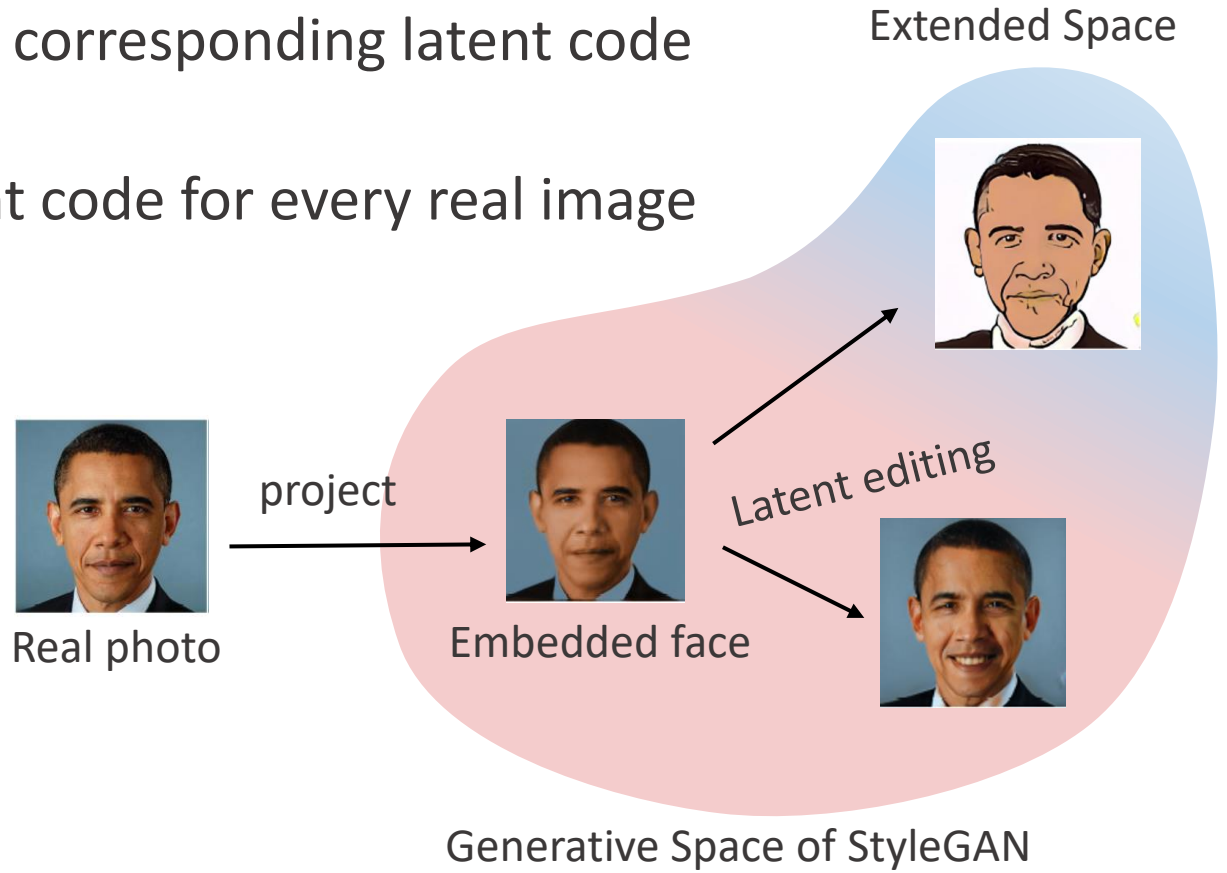
Solution: Image embedding: find the latent code of the real image

Exploiting GAN Prior

Image2StyleGAN: GAN Prior

- **Good GAN prior:**
  - **Compactness**: every latent code (z) can be mapped to realistic images
    - Ensure image quality of the edited image
  - **Sufficiency**: every real image can found its corresponding latent code
    - Edit real-world image
  - **Invertibility**: have a method to find a latent code for every real image within a certain error range
    - Edit real-world image

Given a pretrained GAN,
It provides a good generative space for use to explore.
Images in this generative space can be edited via latent codes.



Gabbay, Aviv, and Yedid Hoshen. Style generator inversion for image enhancement and animation. arXiv 2019.

Image2StyleGAN: Background

- How to find a good latent code of an image?
- Applications for image editing

Image2StyleGAN: Background

- How to find a good latent code of an image?
  - Recall Neural Style Transfer
    - Input image; Fixed network; Output Image
    - Optimize the input image to make the output image minimize given losses
    - Loss: style matching; content matching
  - Image Embedding
    - Input latent code; Fixed network; Output Image
    - Optimize the latent code to make the output image minimize given losses
    - Loss: approach the target photo

Find latent code

- Update latent code via back-propagation

Algorithm 1: Latent Space Embedding for GANs

Input: An image $I \in \mathbb{R}^{n \times m \times 3}$ to embed; a pre-trained generator $G(\cdot)$.

Output: The embedded latent code w^* and the embedded image $G(w^*)$.

1 Initialize latent code $w^* = w$;

2 **while** *not converged* **do**

3 $\quad L \leftarrow L_{perceptual}(G(w^*), I) + \|G(w^*) - I\|_2$;

4 $\quad w^* \leftarrow w^* - \eta F'(\nabla_{w^*} L)$;

5 **end**

Content loss in Neural Style Transfer

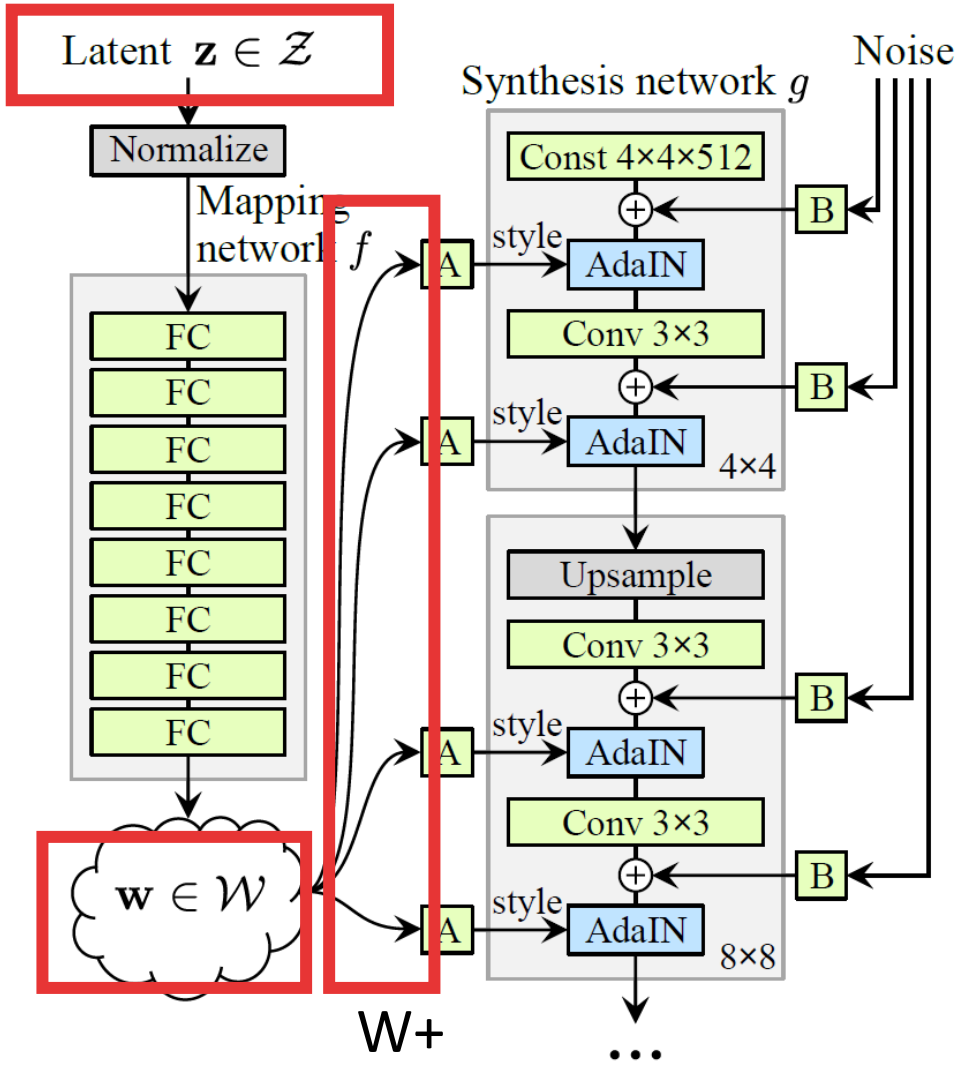
Abdal, Rameen, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? ICCV. 2019.

Which latent code

- Z vs. W vs. W+
- W+: each layer uses independent W
- W+: best reconstruction

input Z

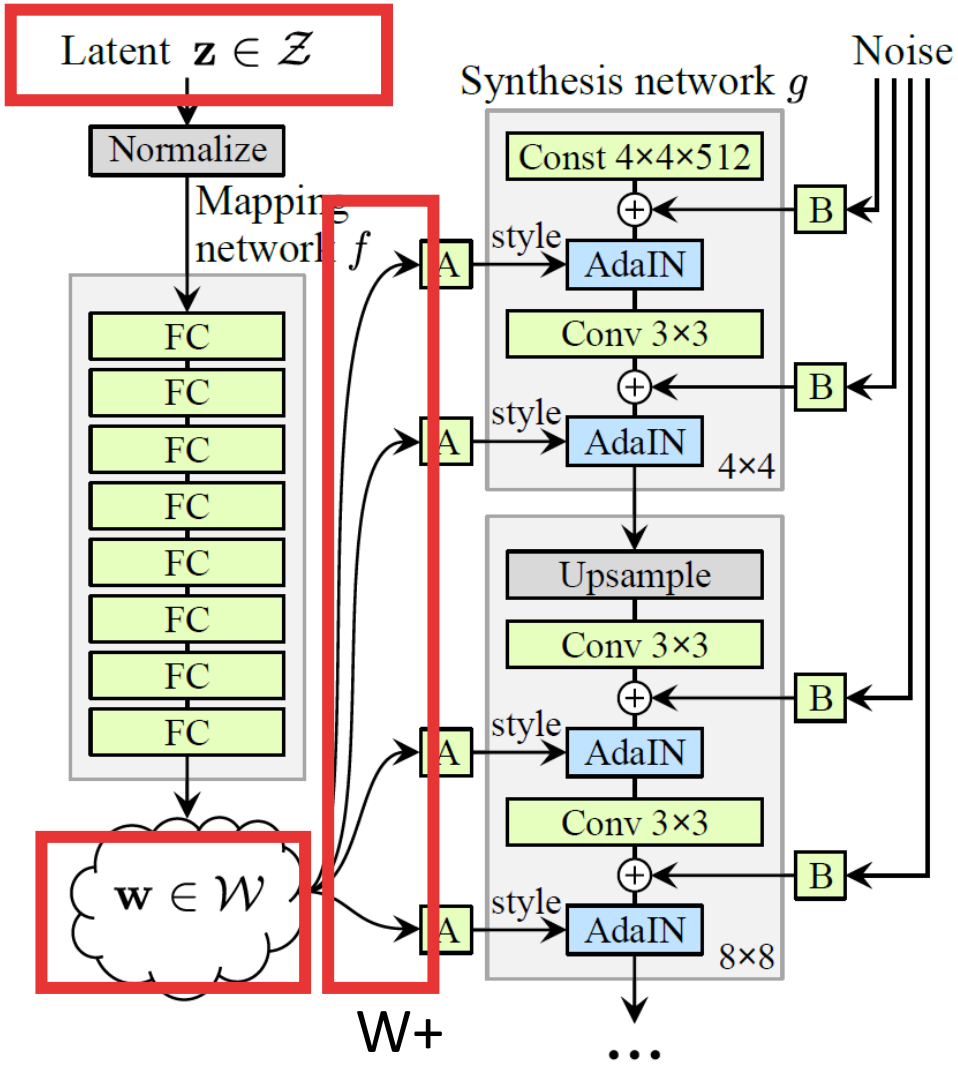
W W+



Which latent code

- Z vs. W vs. W+
- W+: each layer uses independent W
- W+ can even reconstruct non-face images

input W W+



Abdal, Rameen, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? ICCV. 2019.

Which latent code

- Z vs. W vs. W+
- W+: each layer uses independent W
- W+ can even reconstruct non-face images

Abdal, Rameen, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? ICCV. 2019.

Find latent code

- Loss

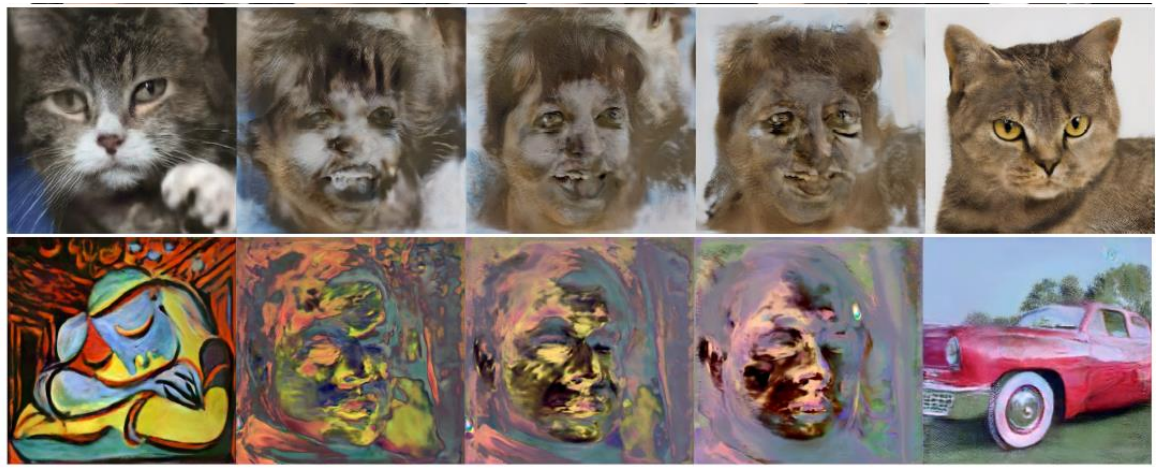
input L2 perceptual L2+perceptual

Abdal, Rameen, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? ICCV. 2019.

Image Editing with GAN prior

- Applications for image editing
  - Image Morphing
  - $w = \lambda * w1 + (1-\lambda) * w2$

only the embedding of faces
is semantically meaningful



Abdal, Rameen, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? ICCV. 2019.

Image Editing with GAN prior

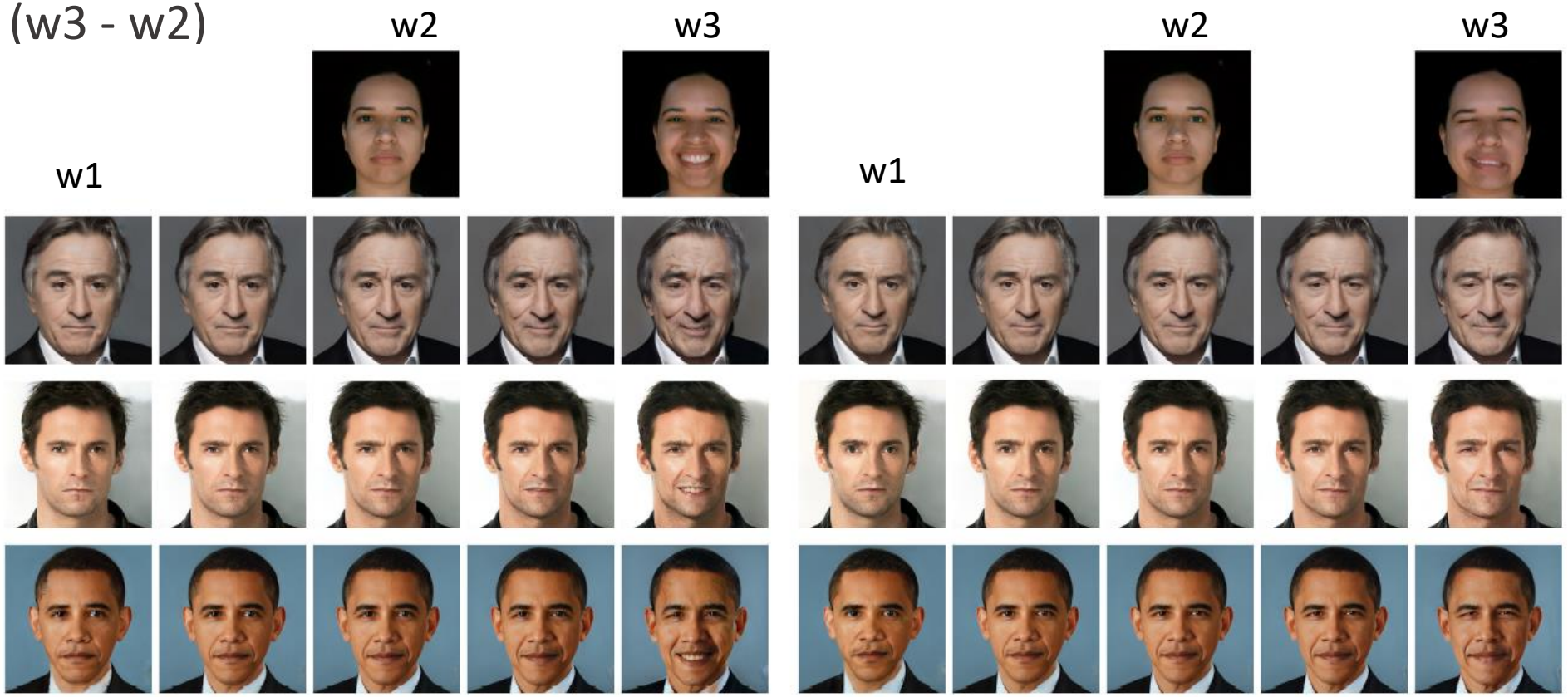
- Applications for image editing
  - Style Transfer
  - w = [shallow layers of w1, deep layers of w2]

- Shallow layers
  - Gender
  - Pose
  - Expression
- Middle layers
  - Hair style
  - Eyes open/closed
  - Appearance
- Deep layers
  - Color

Abdal, Rameen, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? ICCV. 2019.

Image Editing with GAN prior

- Applications for image editing
  - Expression Transfer and Face Reenactment
  - w = w1 +λ* (w3 - w2)



Abdal, Rameen, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? ICCV. 2019.

Image Editing with GAN prior

- Applications for image editing
  - SR & Image Inpainting
  - Find a latent code to generate a image that
    - its LR version is the same as the input image
    - is the same as the input image in the unmasked part

Gabbay, Aviv, and Yedid Hoshen. Style generator inversion for image enhancement and animation. arXiv 2019.

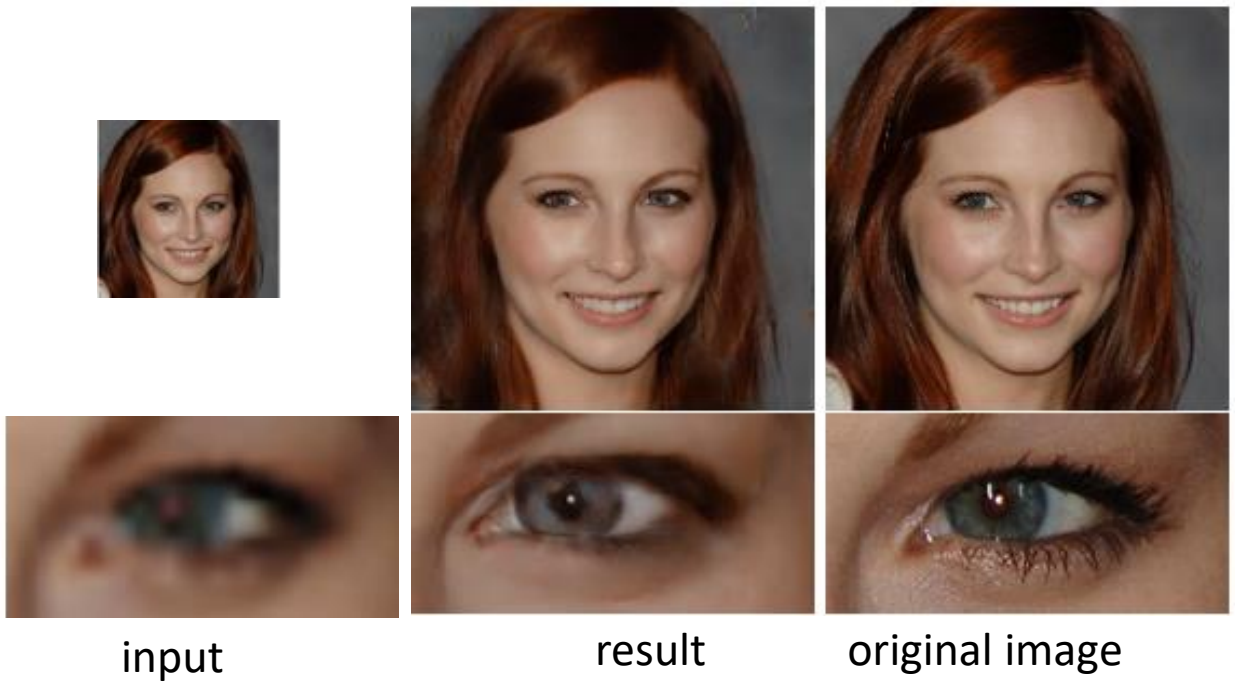
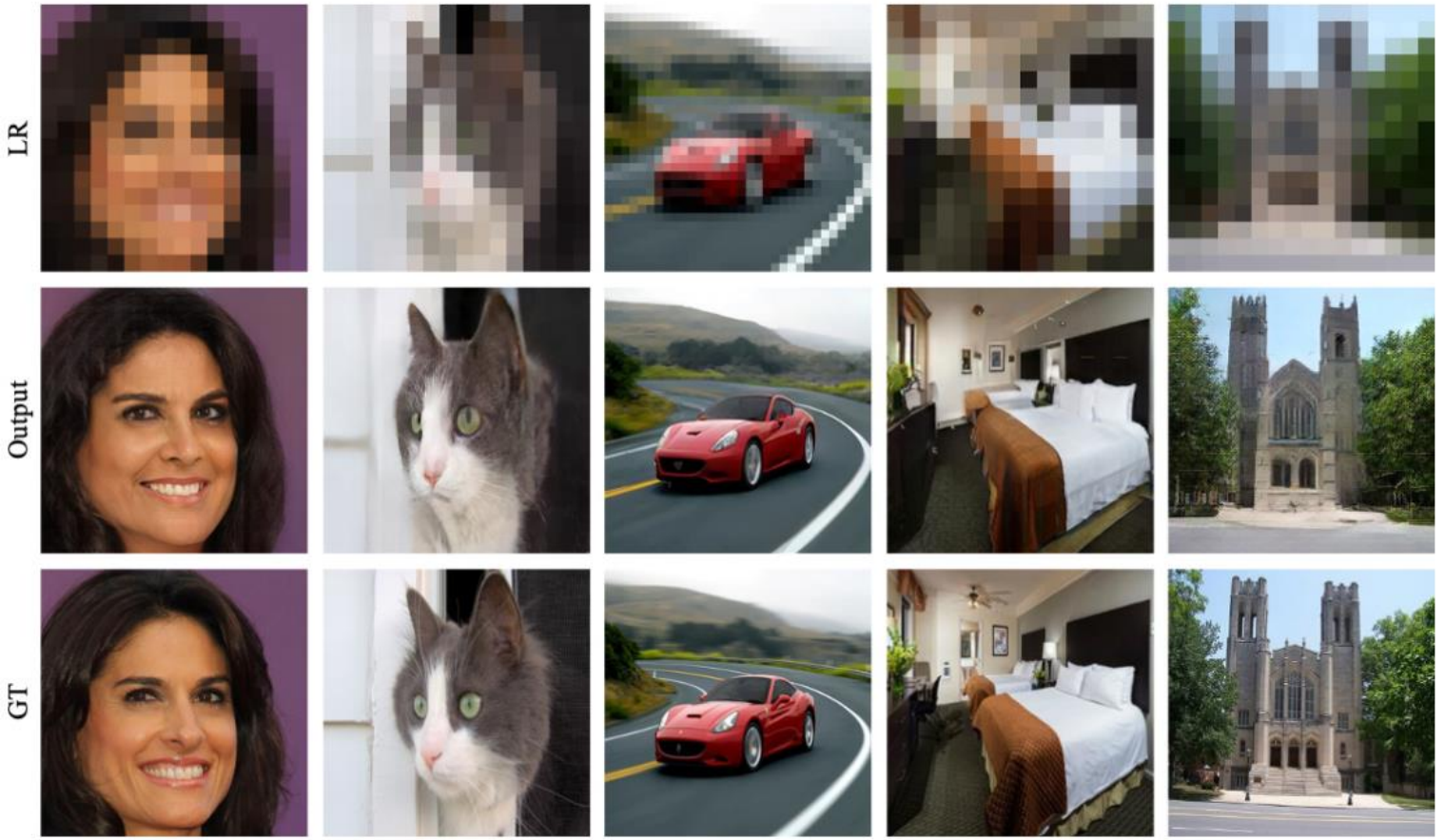


Image Editing with GAN prior

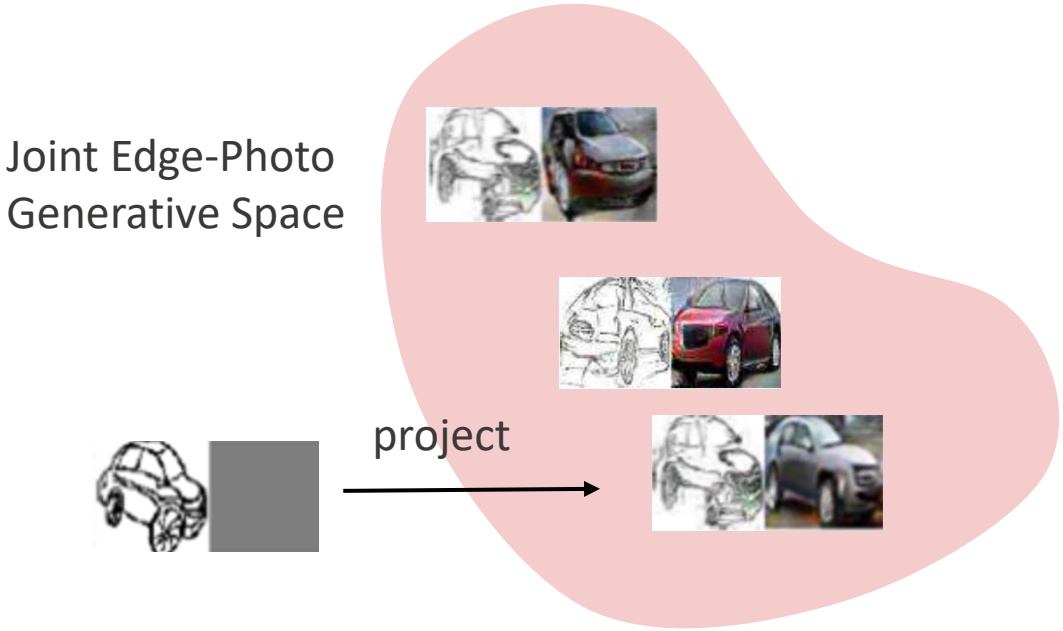
- Applications for image editing
  - SR
  - Combine the features of StyleGAN and the SR Network

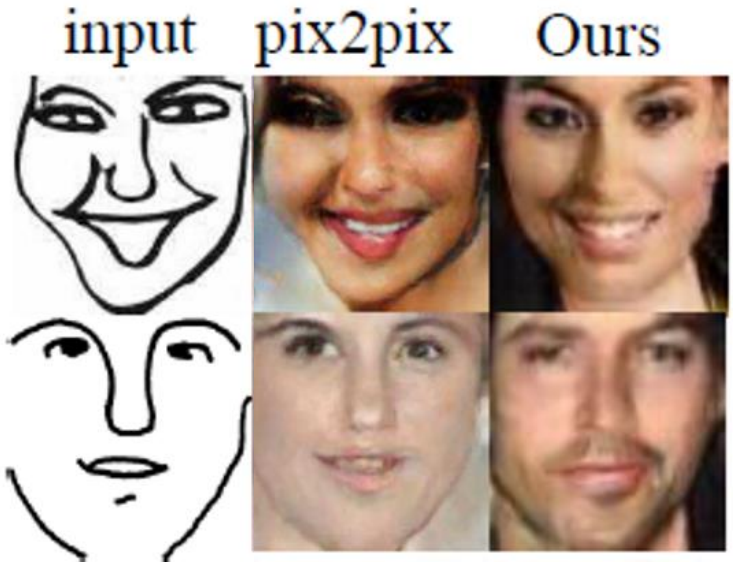


Chan, Kelvin CK, et al. Glean: Generative latent bank for large-factor image super-resolution. CVPR 2021.

Image Editing with GAN prior

- Applications for image editing
  - Robust Sketch-to-Image Translation
  - Find a latent code to generate a image that its edge part approaches the input sketch





Lu, Yongyi, et al. Image generation from sketch constraint using contextual gan. ECCV 2018

Summary

Classic methods :

- Pix2Pix
- CycleGAN
- Neural Style Transfer
- AdaIN, StyleGAN
- Image Embedding

Key ideas:

- conditional GAN, U-Net, PatchGAN, Encoder-Transformer-Decoder
- cycle consistency loss
- style representation: data-driven, local-based, global-based; perceptual loss
- StyleGAN: hierarchical style representation
- GAN prior: find latent code of an image with specialized losses; ensure output quality