计算机视觉

张健

数字媒体研究中心
信息工程学院
北京大学深圳研究生院

2021.11.17

课程内容

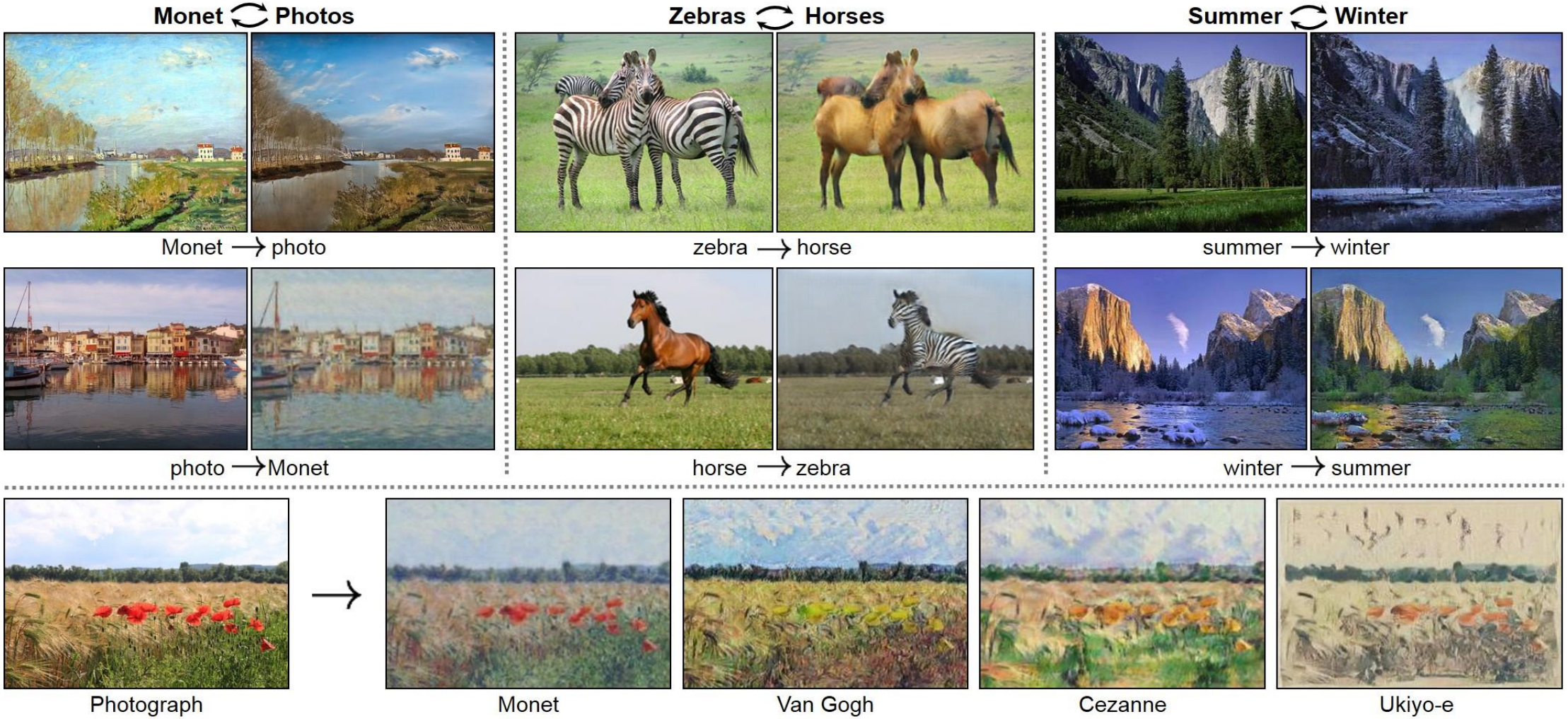
- 作业讨论
- 回顾
- 风格迁移

作业情况

北京大学 PEKING UNIVERSITY

- https://github.com/eriklindernoren/PyTorch-GAN上任选一个感兴趣的GAN的程序，下载运行成功。
- 阅读该程序的论文，写出阅读总结，并对应代码标注出论文中的公式以及网络所对应的代码，阐述清楚。
- 不超过两页。

CycleGAN

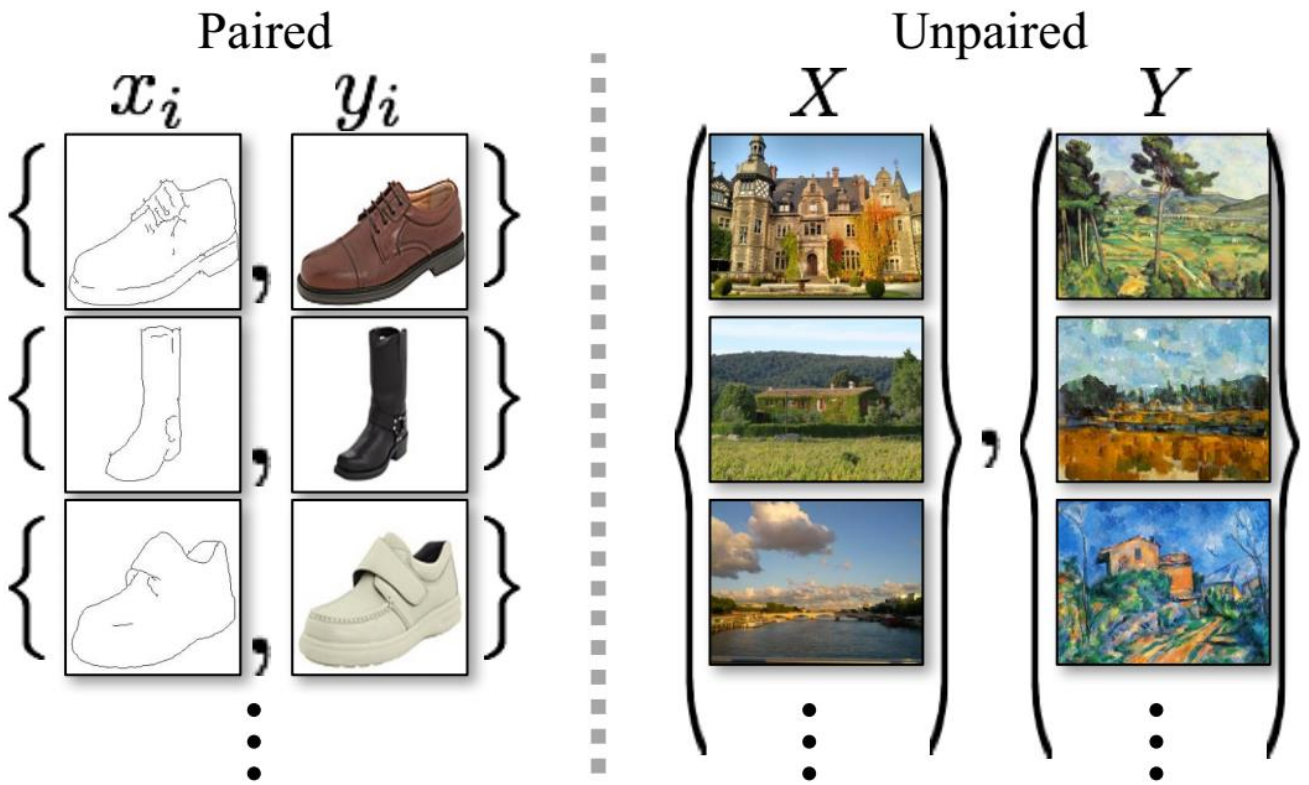


https://github.com/junyanz/CycleGAN

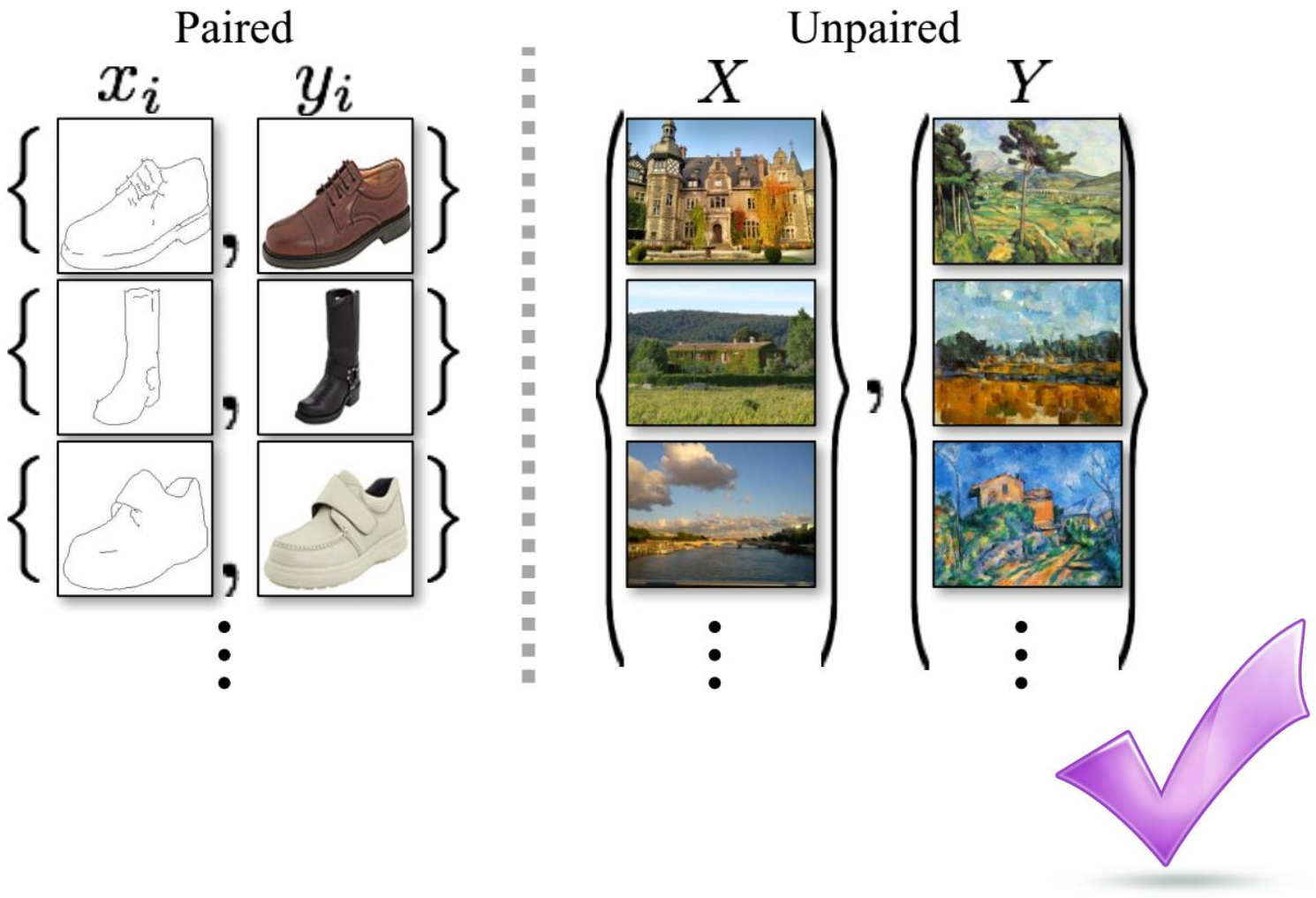
CycleGAN

https://github.com/junyanz/CycleGAN

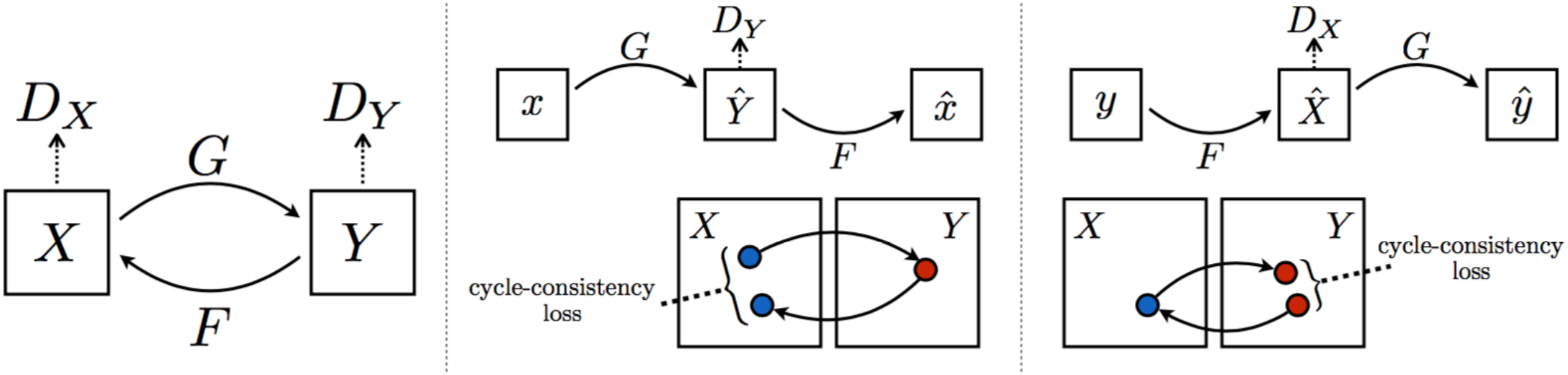
CycleGAN



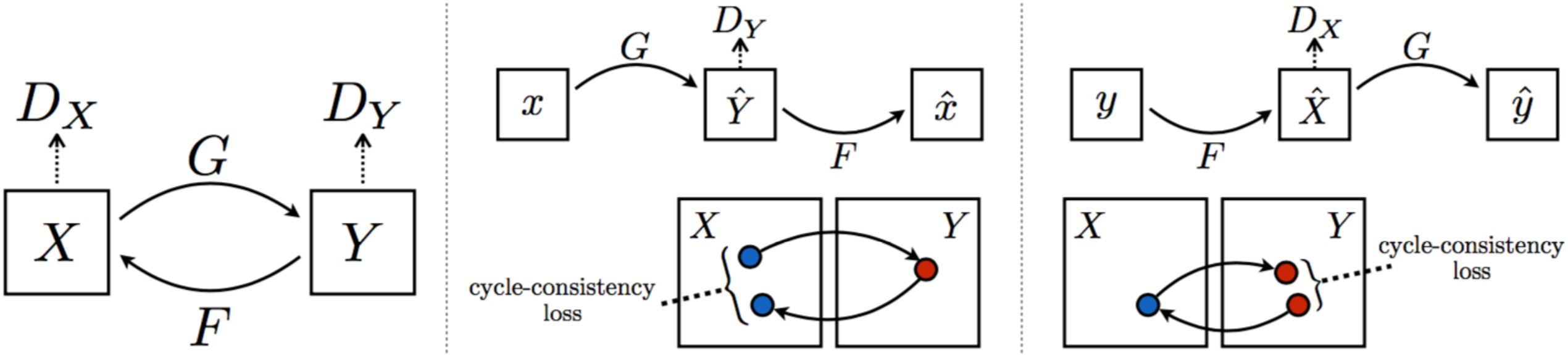
CycleGAN



CycleGAN


D_X
G
D_Y
X
Y
F
x
G
D_Y
$\hat{Y}$
$\hat{x}$
F
cycle-consistency loss
X
Y
y
F
D_X
$\hat{X}$
G
$\hat{y}$
X
Y
cycle-consistency loss

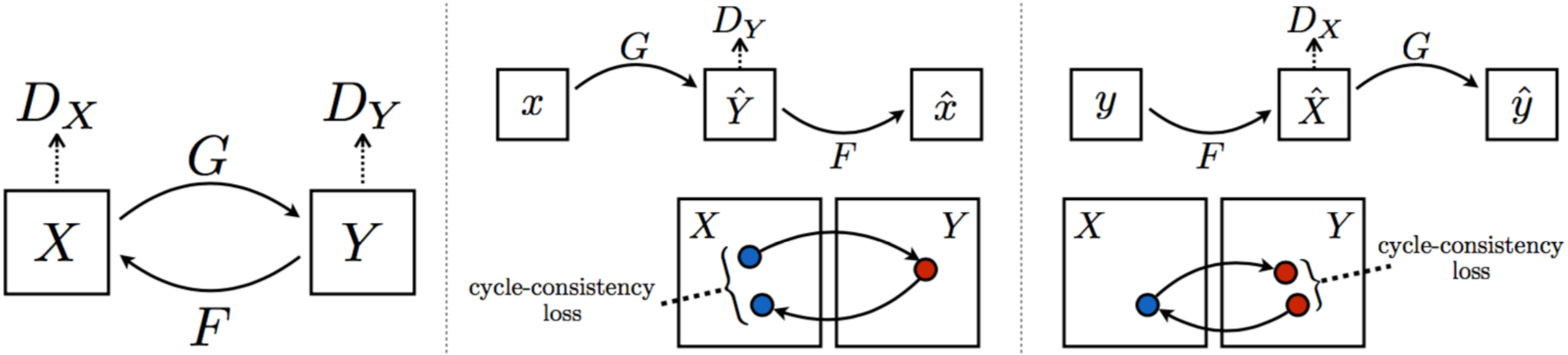
CycleGAN



$$L_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_y}[\log D_Y(y)] + \mathbb{E}_{x \sim p_x}[\log(1 - D_Y(G(x)))]$$

$$L_{GAN}(F, D_X, Y, X) = \mathbb{E}_{x \sim p_x}[\log D_X(x)] + \mathbb{E}_{y \sim p_y}[\log(1 - D_X(F(y)))]$$

CycleGAN

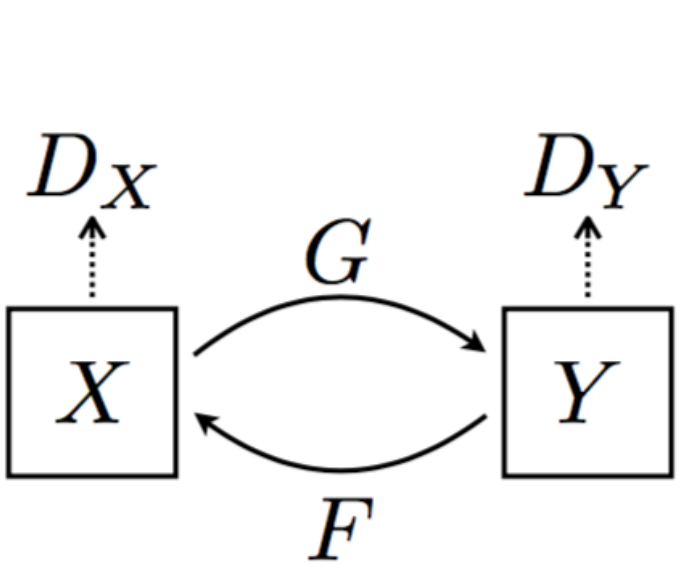


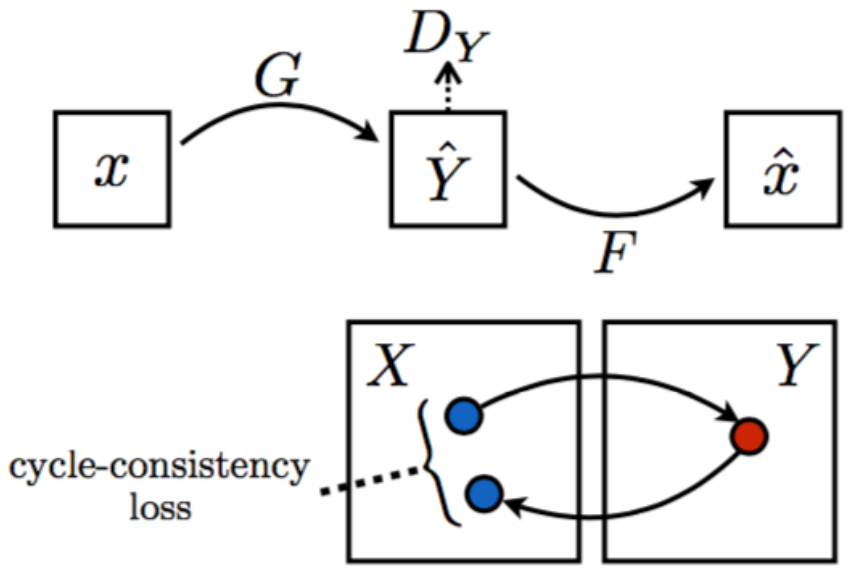
$$L_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_y}[\log D_Y(y)] + \mathbb{E}_{x \sim p_x}[\log(1 - D_Y(G(x)))]$$

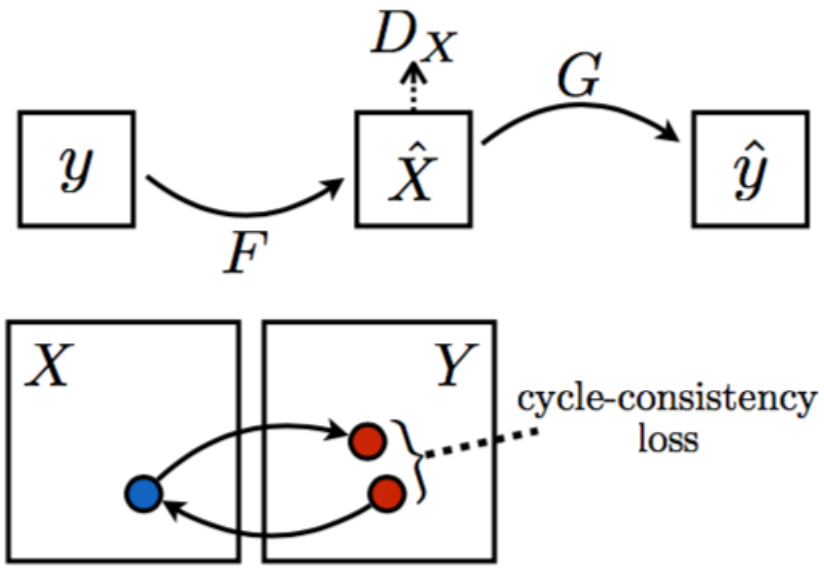
$$L_{GAN}(F, D_X, Y, X) = \mathbb{E}_{x \sim p_x}[\log D_X(x)] + \mathbb{E}_{y \sim p_y}[\log(1 - D_X(F(y)))]$$

$$L_{cyc}(G, F) = \mathbb{E}_{x \sim p_x}[\| F(G(x)) - x \|_1] + \mathbb{E}_{y \sim p_y}[\| G(F(y)) - y \|_1]$$

CycleGAN







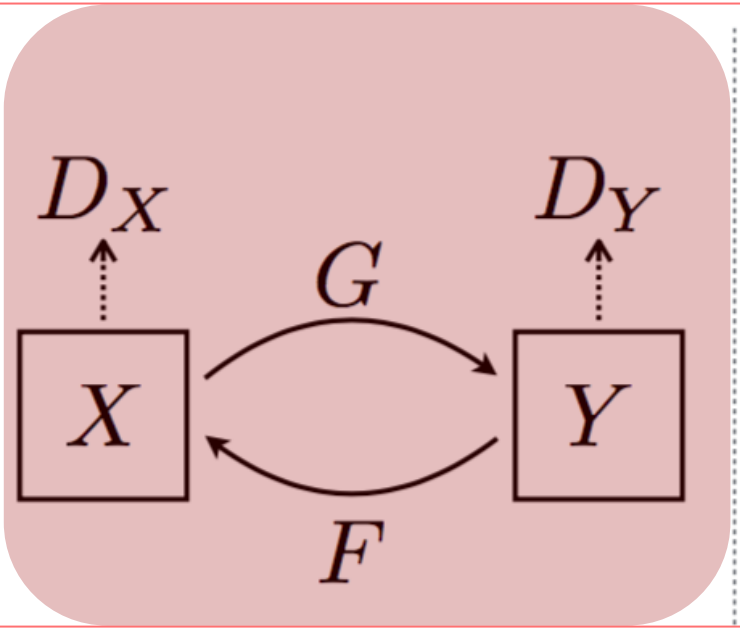
$$L_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_y}[\log D_Y(y)] + \mathbb{E}_{x \sim p_x}[\log(1 - D_Y(G(x)))]$$

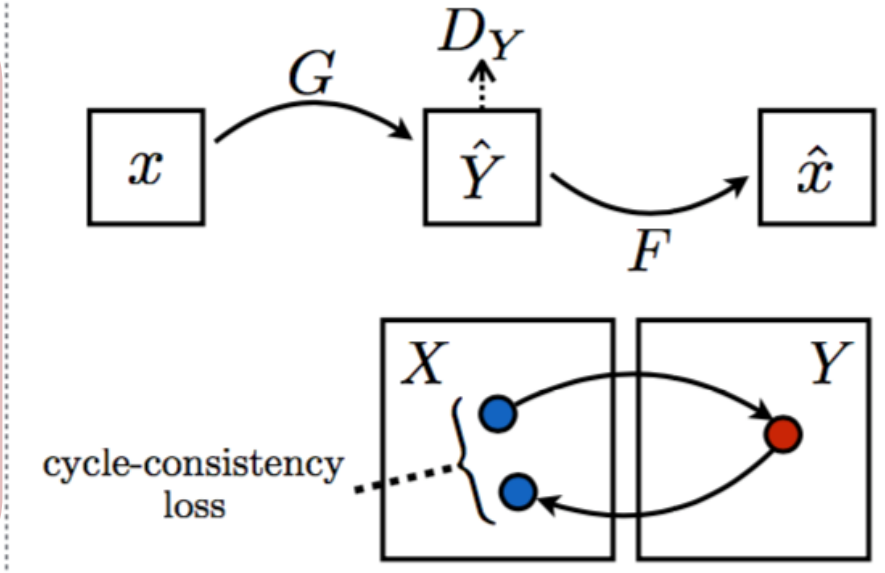
$$L_{GAN}(F, D_X, Y, X) = \mathbb{E}_{x \sim p_x}[\log D_X(x)] + \mathbb{E}_{y \sim p_y}[\log(1 - D_X(F(y)))]$$

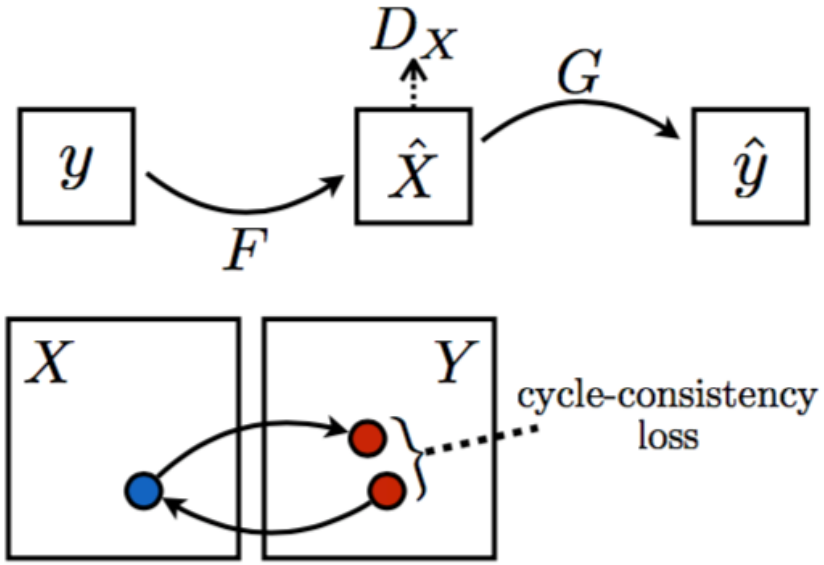
$$L_{cyc}(G, F) = \mathbb{E}_{x \sim p_x}[\| F(G(x)) - x \|_1] + \mathbb{E}_{y \sim p_y}[\| G(F(y)) - y \|_1]$$

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

CycleGAN







$$L_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_y}[\log D_Y(y)] + \mathbb{E}_{x \sim p_x}[\log(1 - D_Y(G(x)))]$$

$$L_{GAN}(F, D_X, Y, X) = \mathbb{E}_{x \sim p_x}[\log D_X(x)] + \mathbb{E}_{y \sim p_y}[\log(1 - D_X(F(y)))]$$

$$L_{cyc}(G, F) = \mathbb{E}_{x \sim p_x}[\| F(G(x)) - x \|_1] + \mathbb{E}_{y \sim p_y}[\| G(F(y)) - y \|_1]$$

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

补充

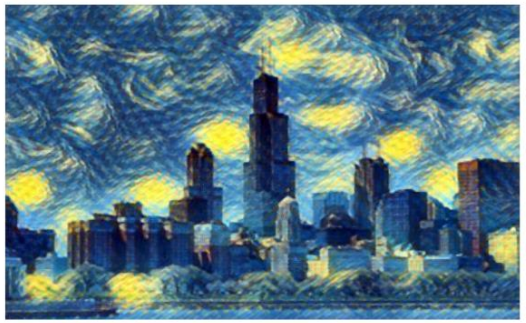
- Contrastive Learning for Unpaired Image-to-Image Translation ECCV 2020
- https://github.com/taesungp/contrastive-unpaired-translation

- SinGAN: Learning a Generative Model from a Single Natural Image ICCV 2019
- Best paper award
- https://github.com/tamarott/SinGAN

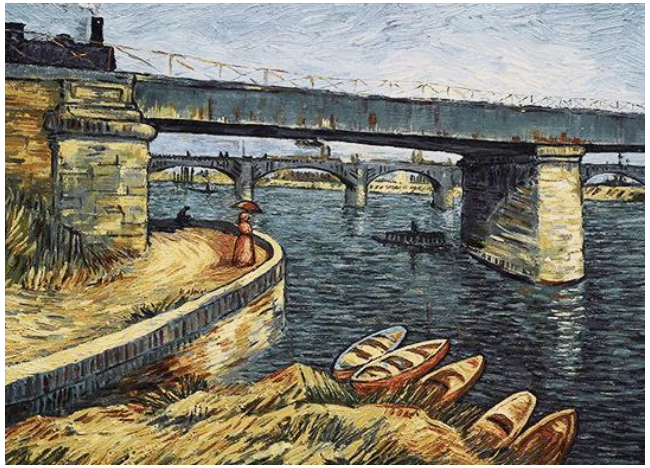
风格迁移

content

style



result



内容

- 导引
  - Adversarial Example
  - Adversarial Noise for MNIST
- 风格迁移
  - 普通风格迁移（固定风格固定内容）
  - 快速风格迁移（固定风格任意内容）
  - 任意风格迁移（任意风格任意内容）

Adversarial Example

Original Image:
bow tie (97.22%)

Image + Noise:
bow tie (0.00%)
bookcase (99.72%)

Amplified Noise

Adversarial Example

Original Image:
macaw (97.38%)

Image + Noise:
macaw (0.00%)
bookcase (99.12%)

Amplified Noise

Adversarial Example

Original Image:
sweatshirt (19.73%)

Image + Noise:
sweatshirt (0.00%)
bookcase (99.09%)

Amplified Noise

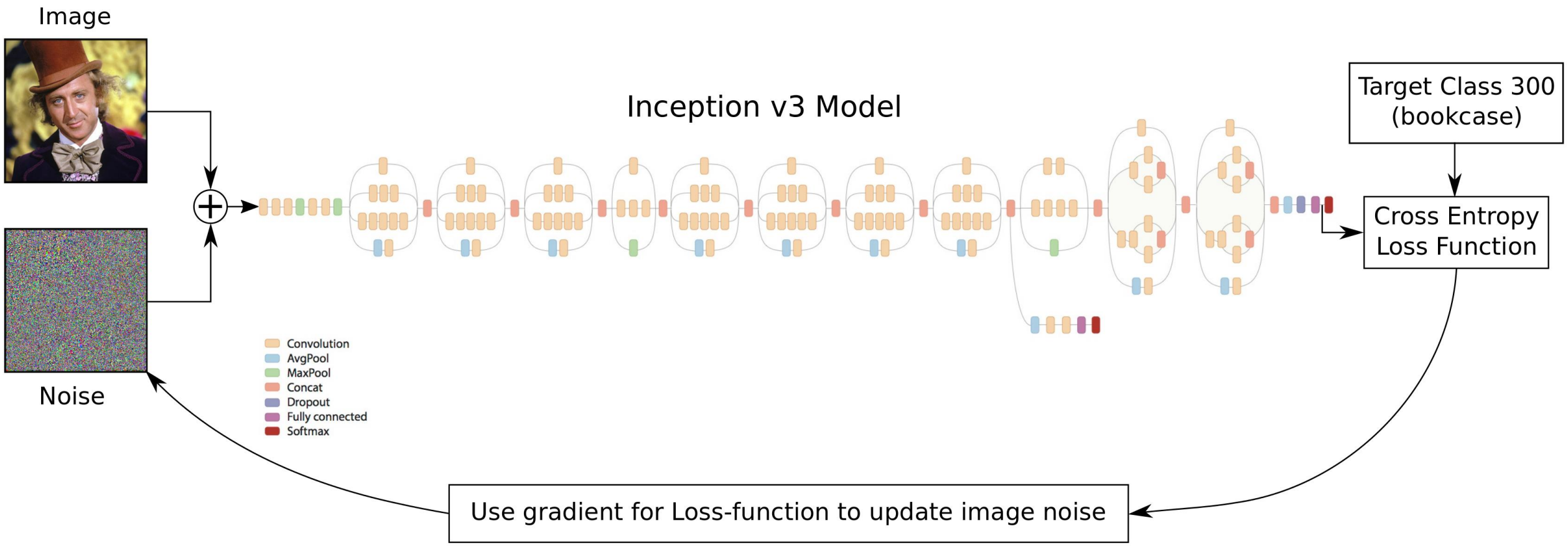
Adversarial Example

Original Image:
sunglasses (31.48%)

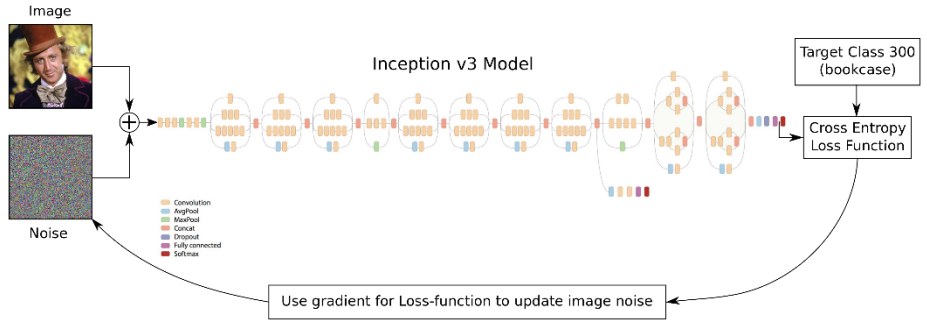
Image + Noise:
sunglasses (0.03%)
bookcase (99.03%)

Amplified Noise

Adversarial Example

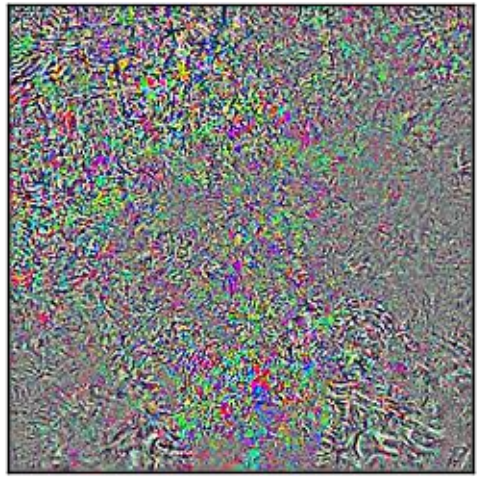


Adversarial Example



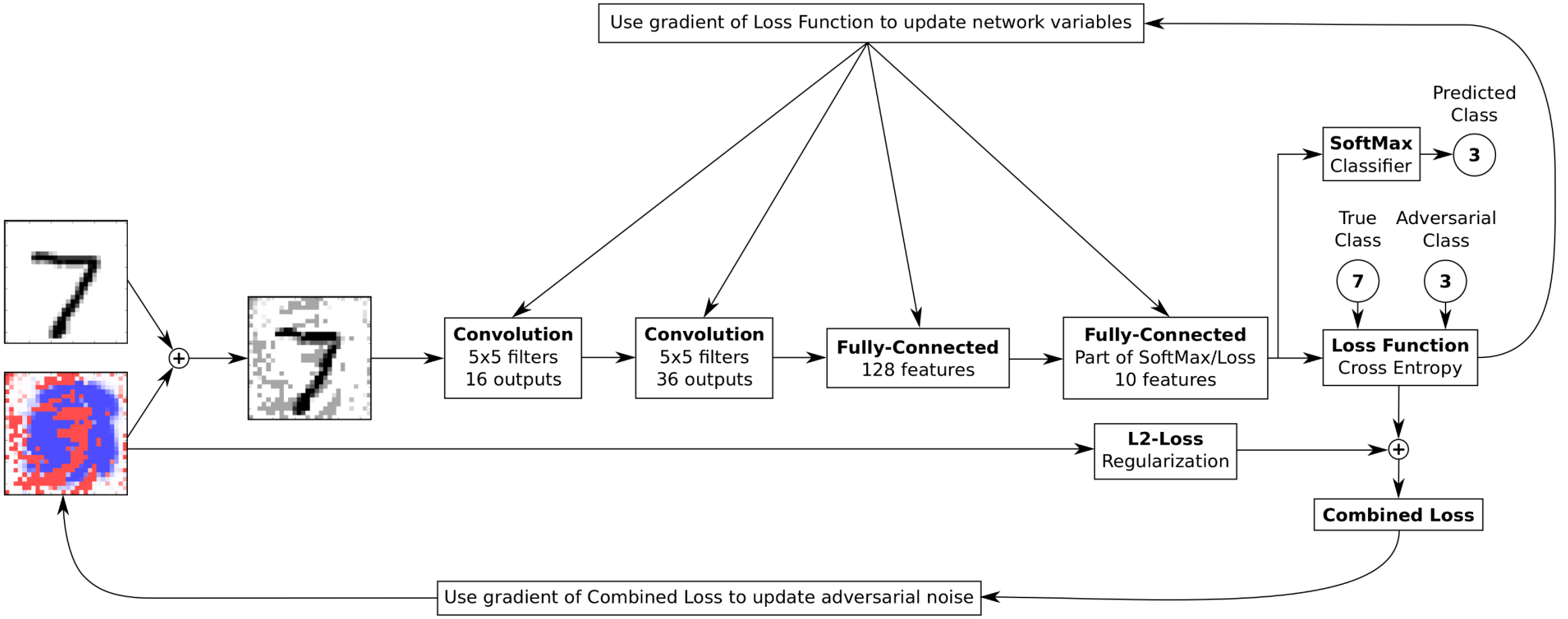
Original Image:
bow tie (97.22%)

Image + Noise:
bow tie (0.00%)
bookcase (99.72%)



Amplified Noise

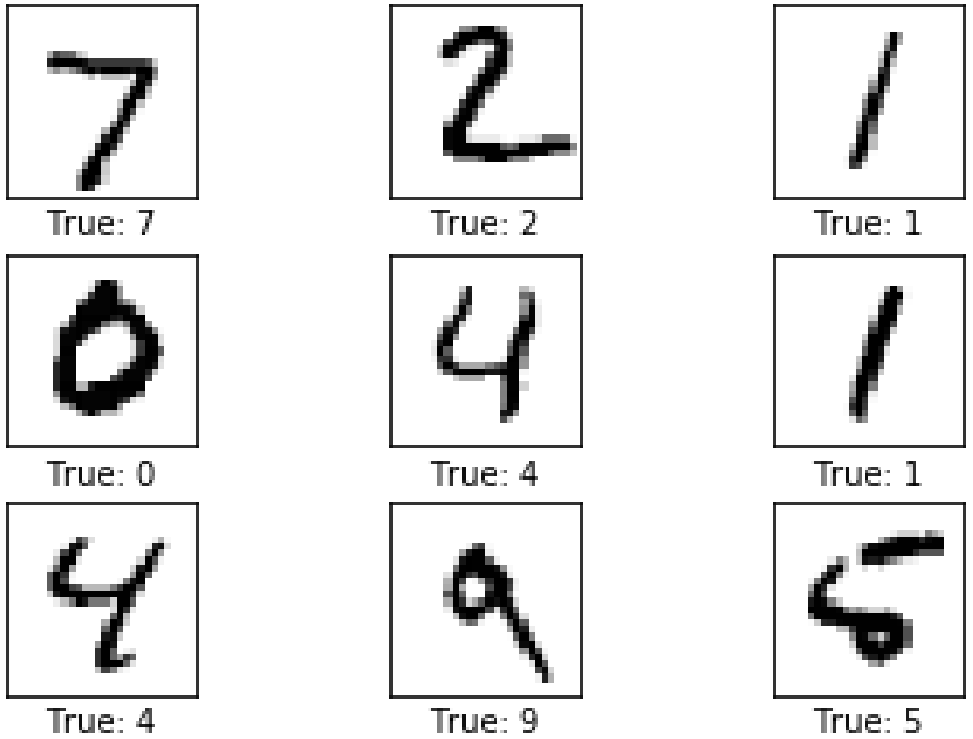
Adversarial Noise for MNIST



https://github.com/Hvass-Labs/TensorFlow-Tutorial

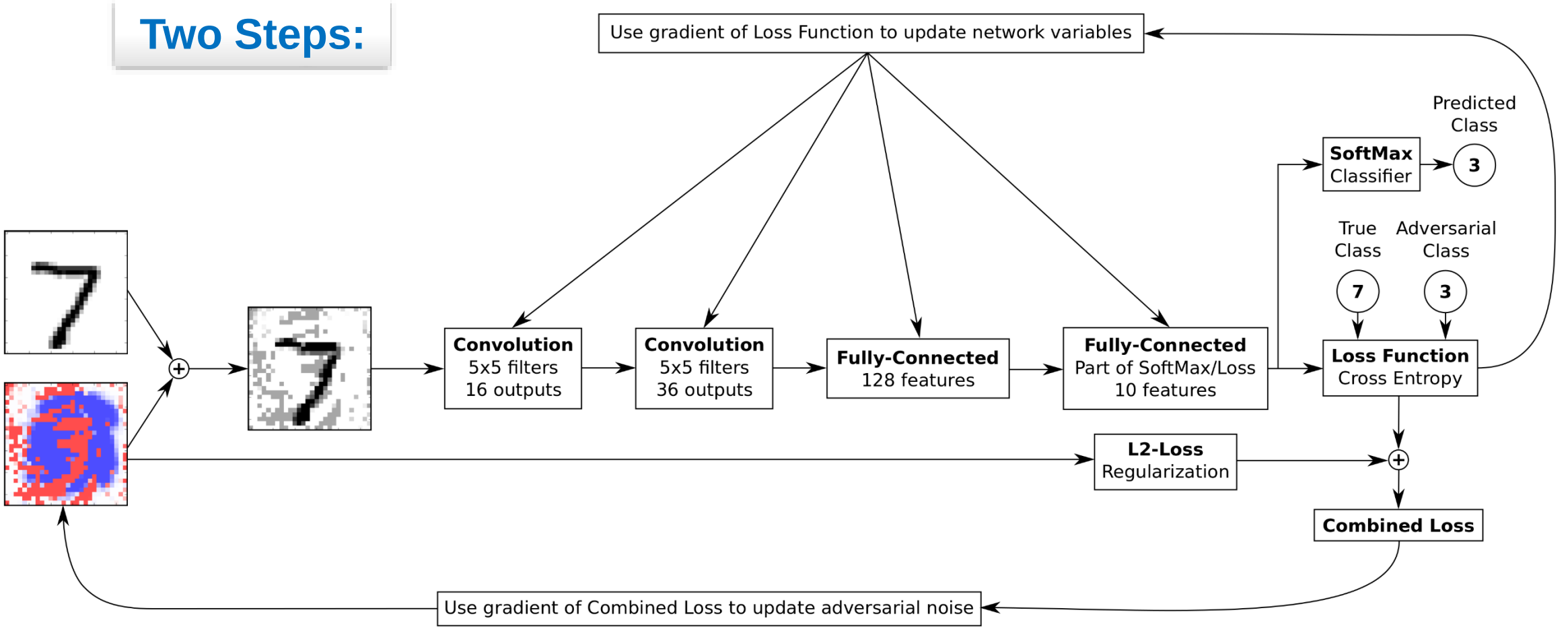
Adversarial Noise for MNIST

Plot MNIST data:



Adversarial Noise for MNIST

Two Steps:

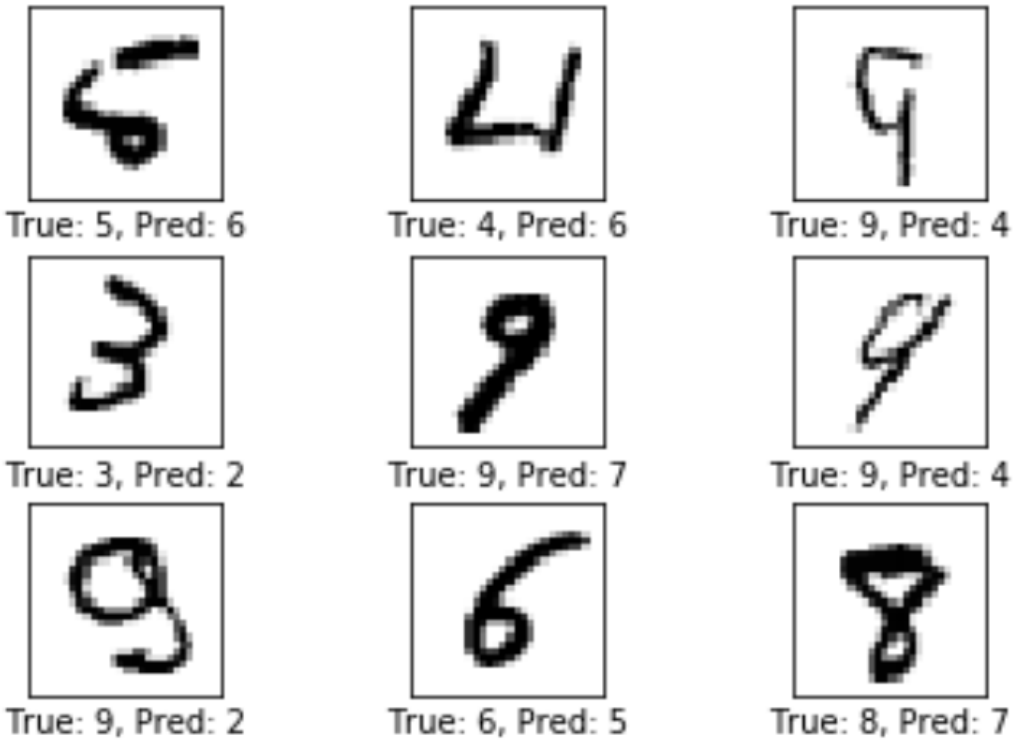


https://github.com/Hvass-Labs/TensorFlow-Tutorial

Adversarial Noise for MNIST

First Step:

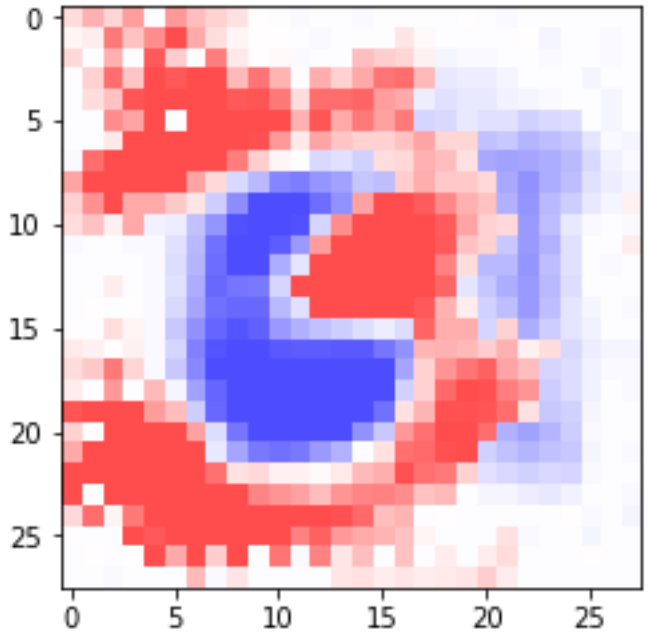
```
Accuracy on Test-Set: 94.2% (9417 / 10000)
Example errors:
```



Adversarial Noise for MNIST

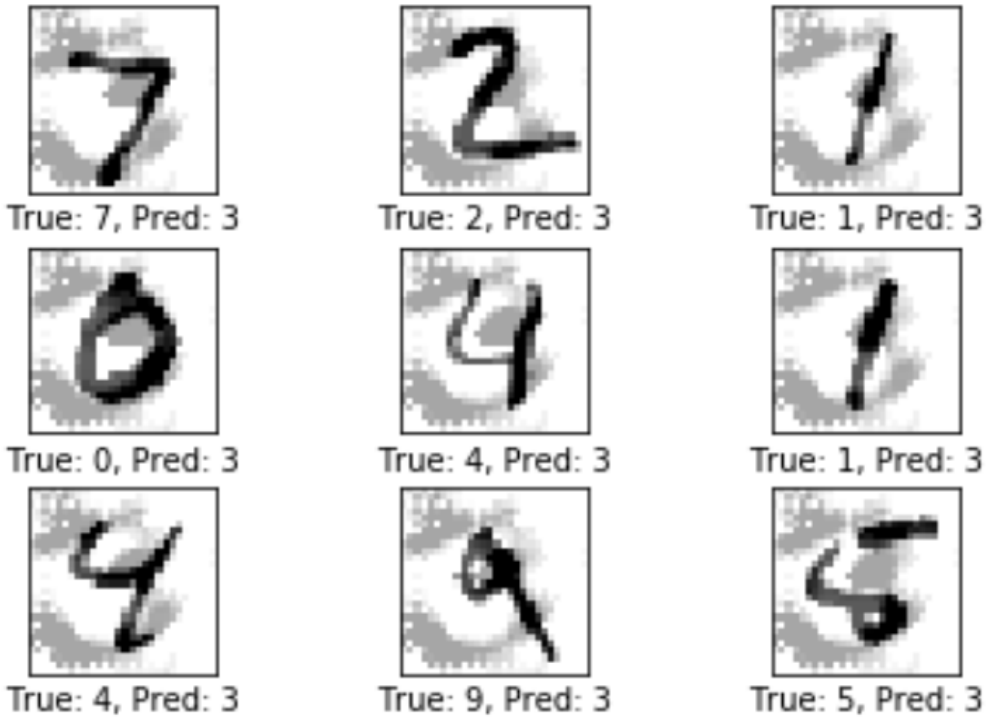
Second Step:

plot_noise()



Adversarial Noise for MNIST

```
Accuracy on Test-Set: 13.8% (1378 / 10000)
Example errors:
```

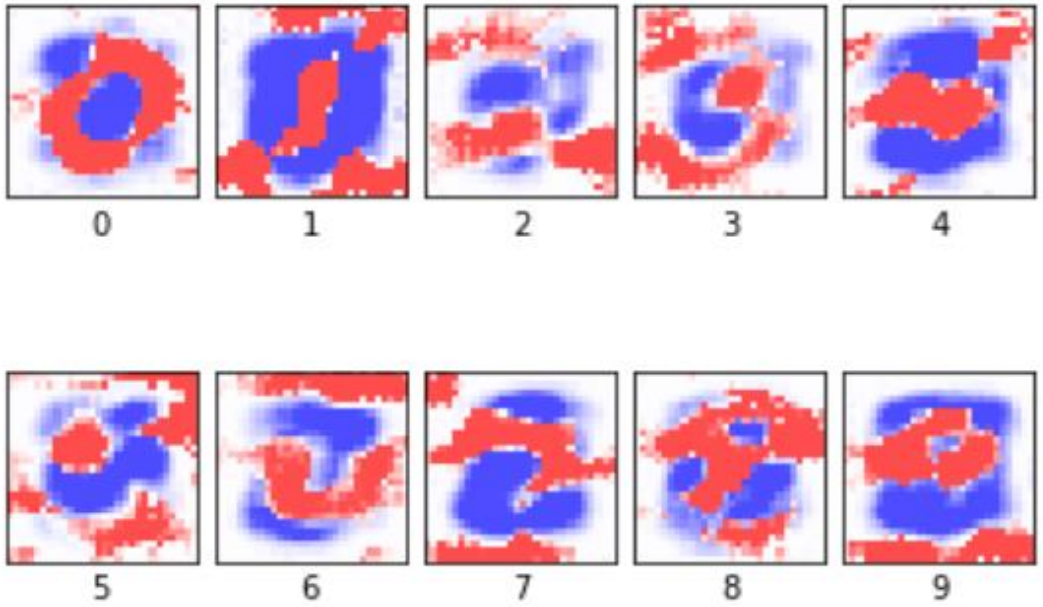


Adversarial Noise for MNIST

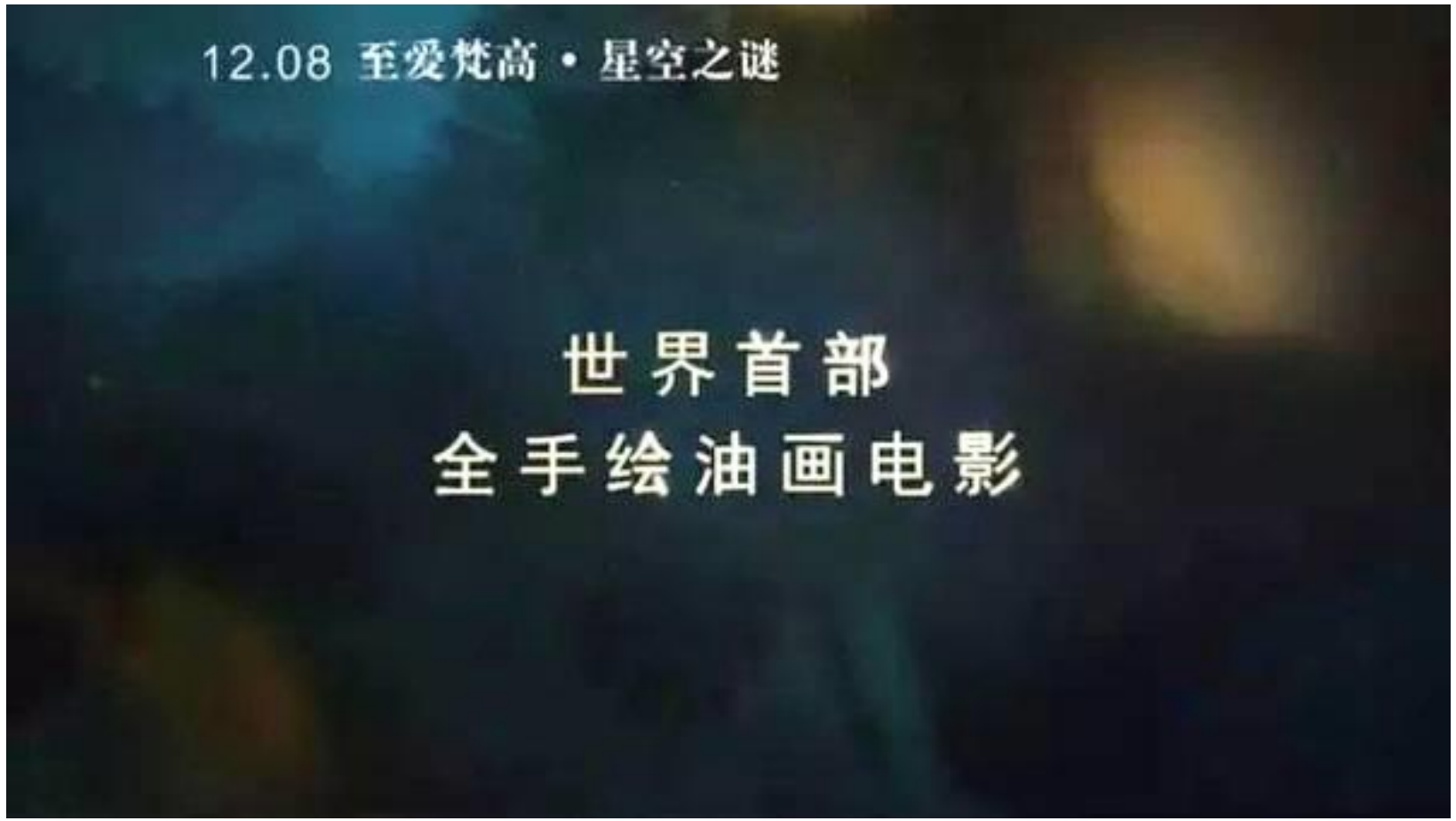
```
Confusion Matrix:
[[ 115    0    0  863    0    0    0    2    0    0]
 [   0    0    0 1135    0    0    0    0    0    0]
 [   0    0  146  886    0    0    0    0    0    0]
 [   0    0    0 1010    0    0    0    0    0    0]
 [   0    0    0  966   16    0    0    0    0    0]
 [   0    0    0  865    0   27    0    0    0    0]
 [   0    0    0  946    0    1   11    0    0    0]
 [   0    0    0  981    0    0    0   47    0    0]
 [   0    0    0  968    0    0    0    0    6    0]
 [   0    0    1 1008    0    0    0    0    0    0]]
```

Adversarial Noise for MNIST

```
plot_all_noise(all_noise)
```

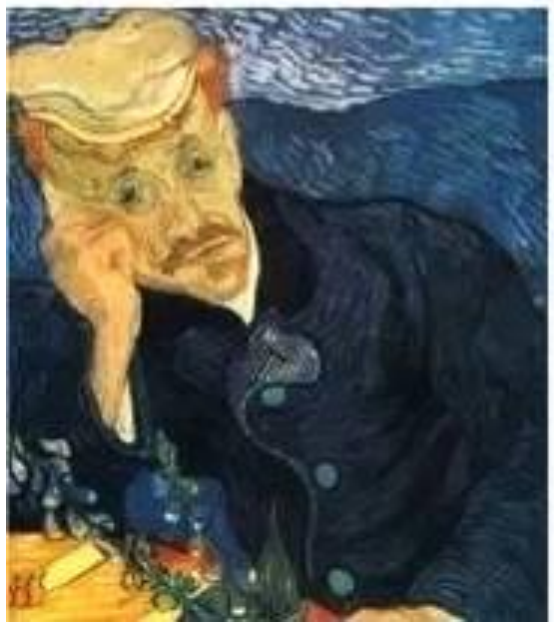


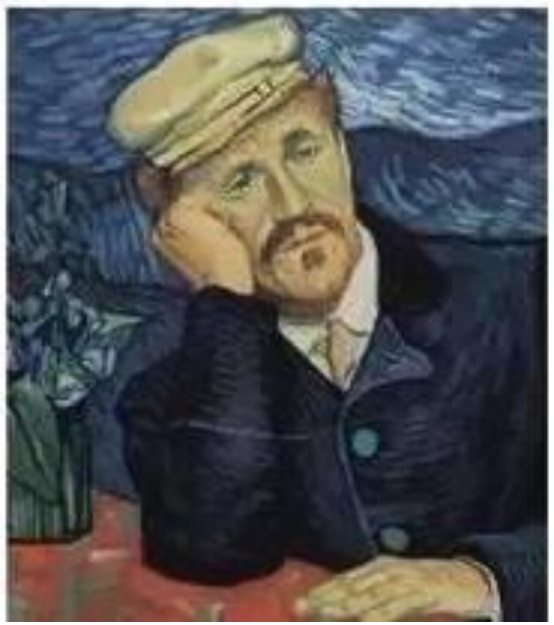
风格迁移


12.08 至爱梵高 • 星空之谜
世界首部
全手绘油画电影

THE ECCENTRIC DOCTOR

"Two weeks later I am sitting by his beside, he is dying. And all he says to me is 'maybe it will be better for everyone.' "





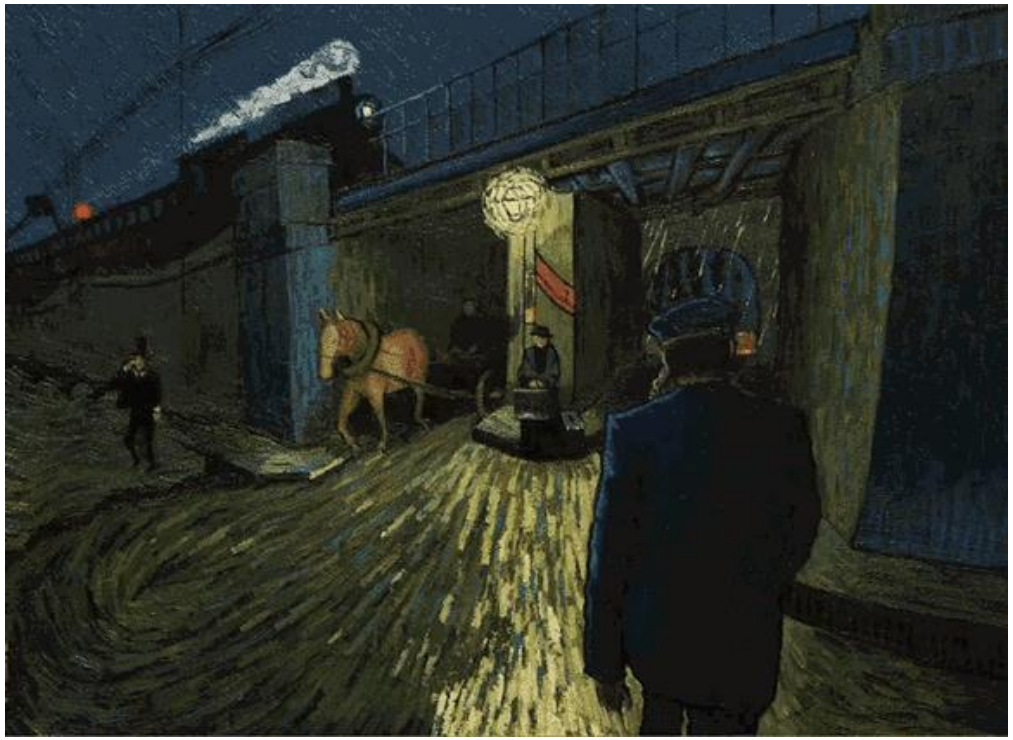
THE RELUCTANT DETECTIVE

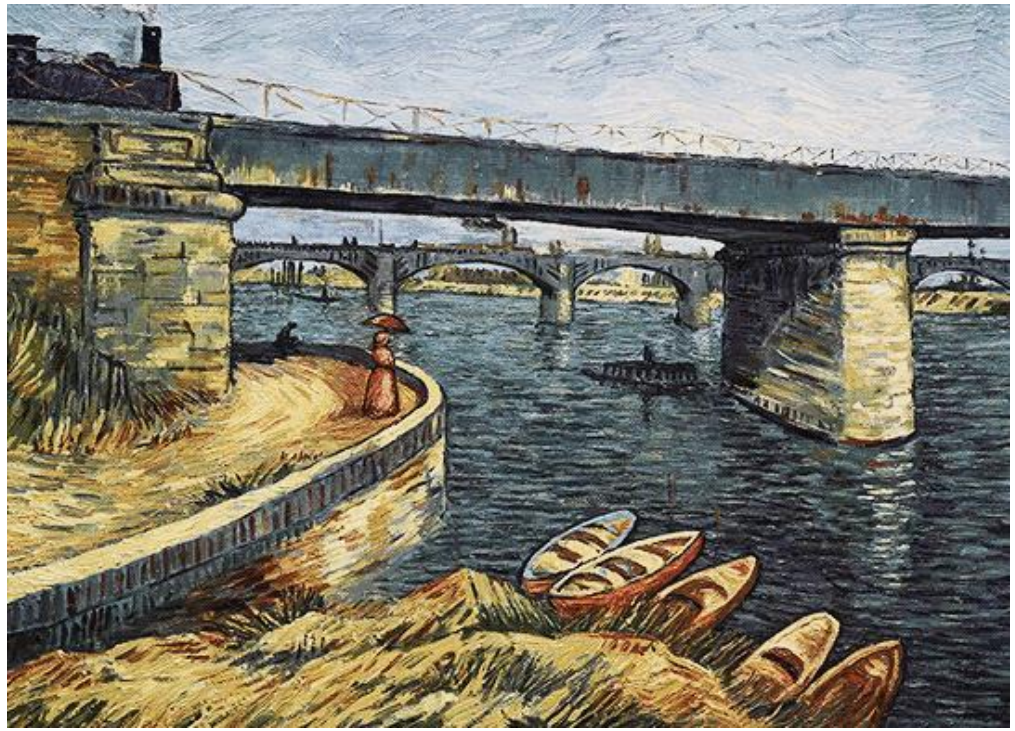
"I don't see the point in delivering a dead man's letter"

ORIGINAL PAINTING

KEYFRAME

PHOTO





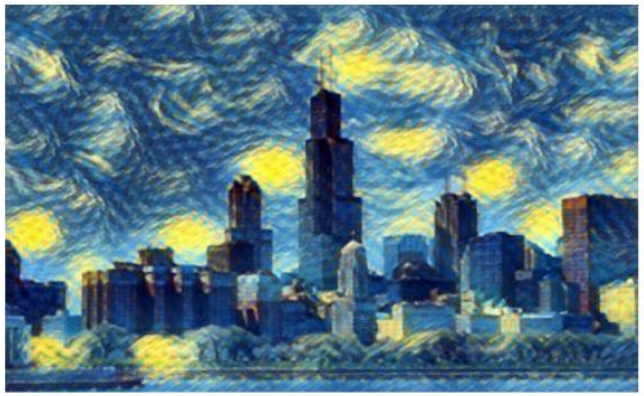
风格迁移开山之作

Gatys et al, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

风格迁移开山之作

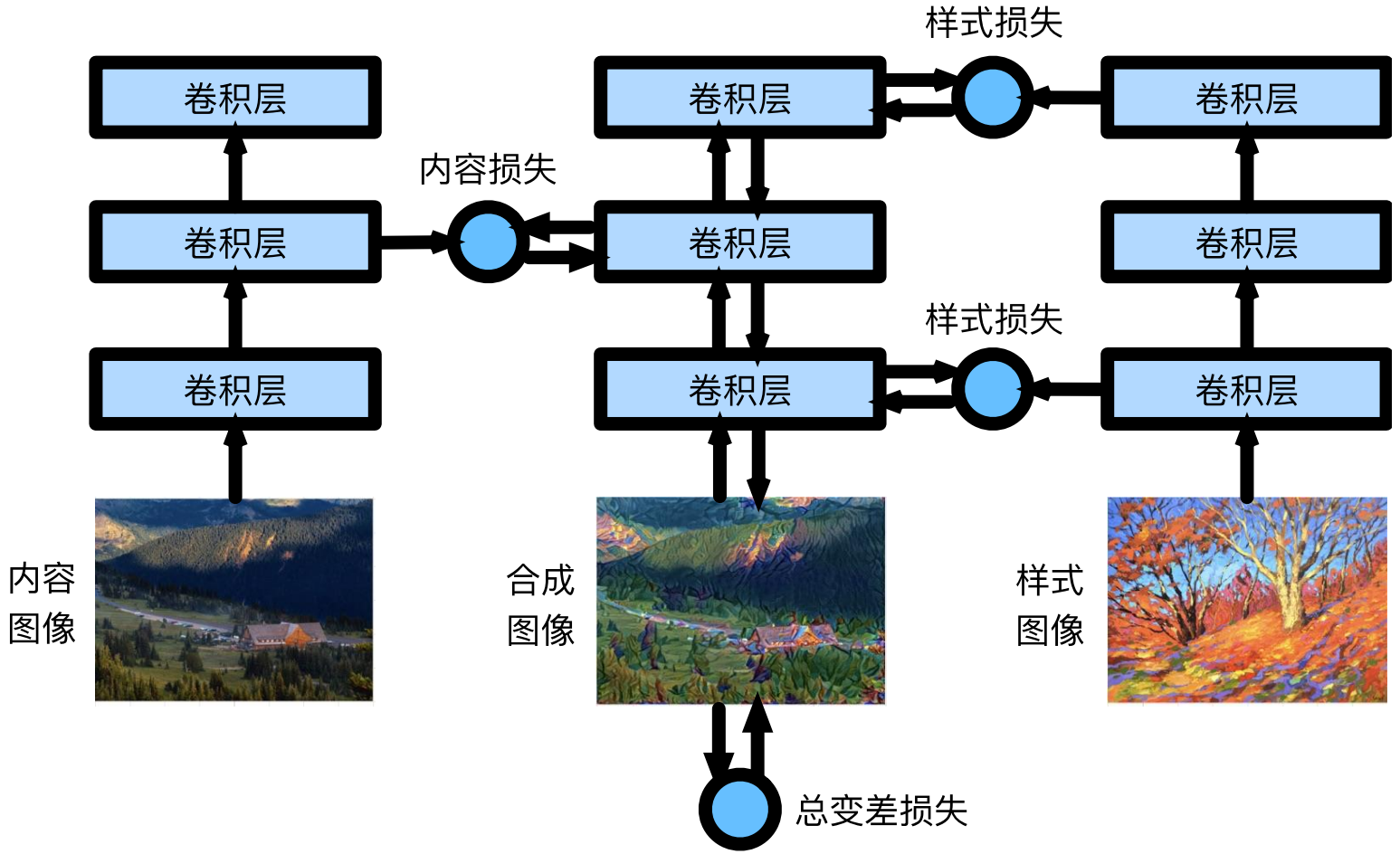
content

style

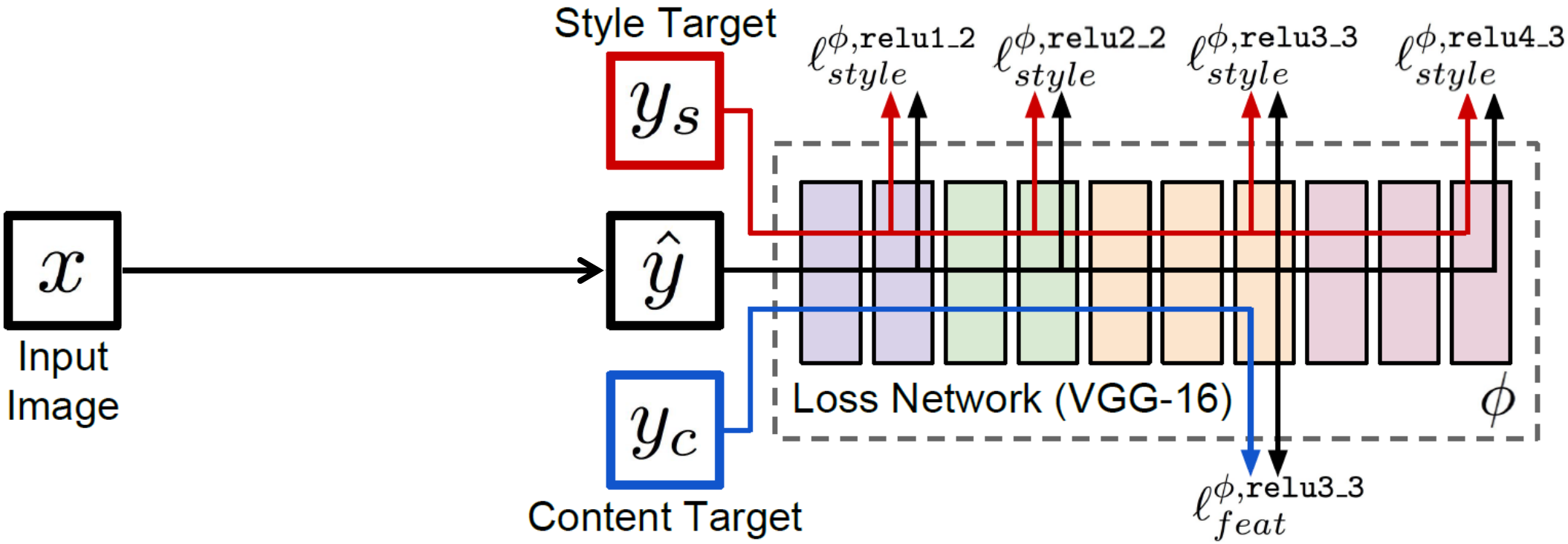


result

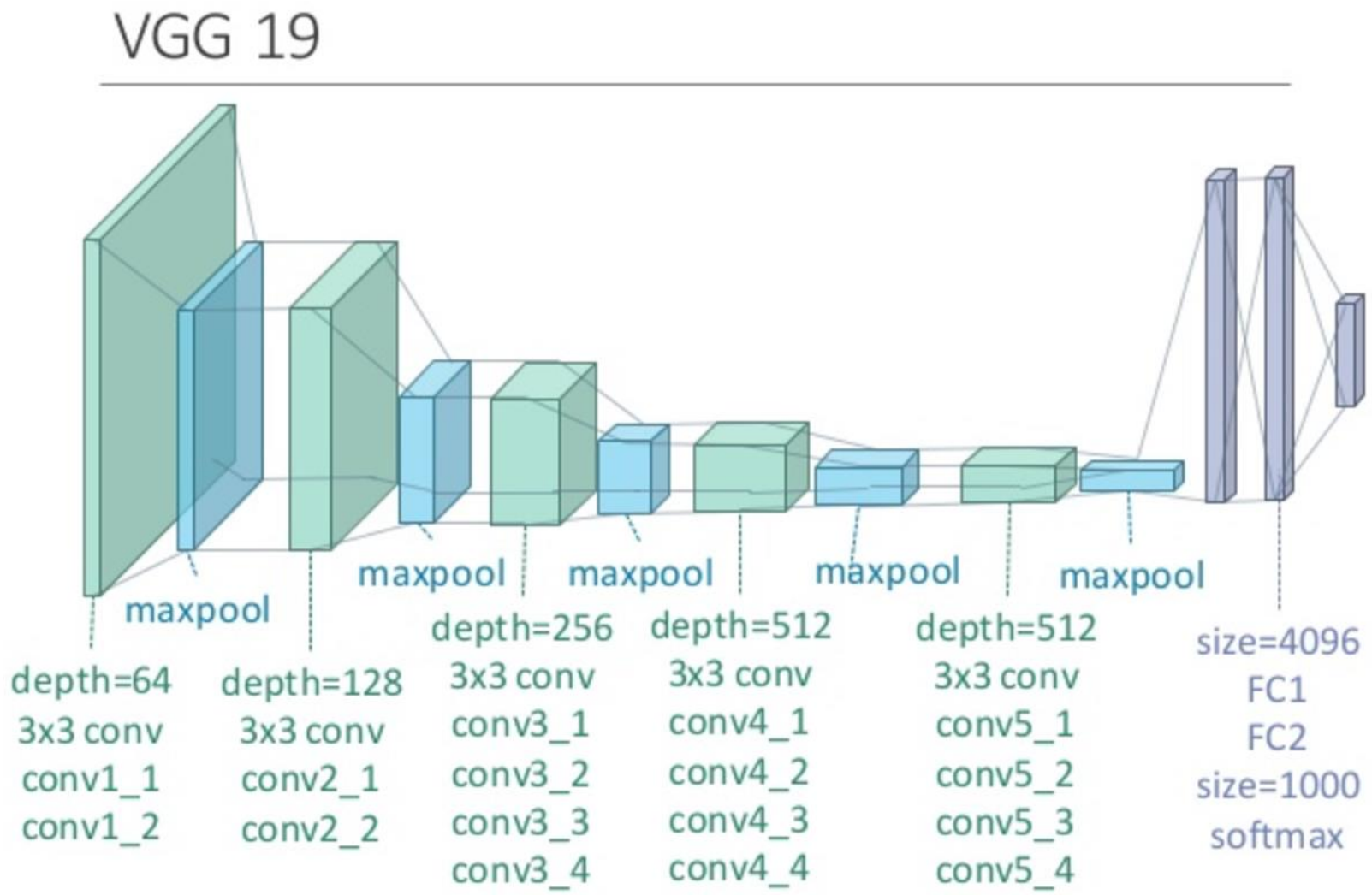
风格迁移开山之作


样式损失
卷积层
卷积层
卷积层
内容损失
卷积层
卷积层
卷积层
卷积层
卷积层
样式损失
卷积层
卷积层
内容
图像
合成
图像
样式
图像
总变差损失

风格迁移开山之作


Style Target
y_s
$\ell^{\phi,\texttt{relu1\_2}}_{style}$
$\ell^{\phi,\texttt{relu2\_2}}_{style}$
$\ell^{\phi,\texttt{relu3\_3}}_{style}$
$\ell^{\phi,\texttt{relu4\_3}}_{style}$
x
Input Image
$\hat{y}$
y_c
Content Target
Loss Network (VGG-16)
ϕ
$\ell^{\phi,\texttt{relu3\_3}}_{feat}$

风格迁移开山之作


VGG 19
maxpool
maxpool
maxpool
maxpool
maxpool
depth=64
3x3 conv
conv1_1
conv1_2
depth=128
3x3 conv
conv2_1
conv2_2
depth=256
3x3 conv
conv3_1
conv3_2
conv3_3
conv3_4
depth=512
3x3 conv
conv4_1
conv4_2
conv4_3
conv4_4
depth=512
3x3 conv
conv5_1
conv5_2
conv5_3
conv5_4
size=4096
FC1
FC2
size=1000
softmax

风格迁移开山之作

内容损失

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l} \right)^2 .$$

Gatys et al, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

风格迁移开山之作

样式损失

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}.$$

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^{l} - A_{ij}^{l}\right)^2$$

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

Gatys et al, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

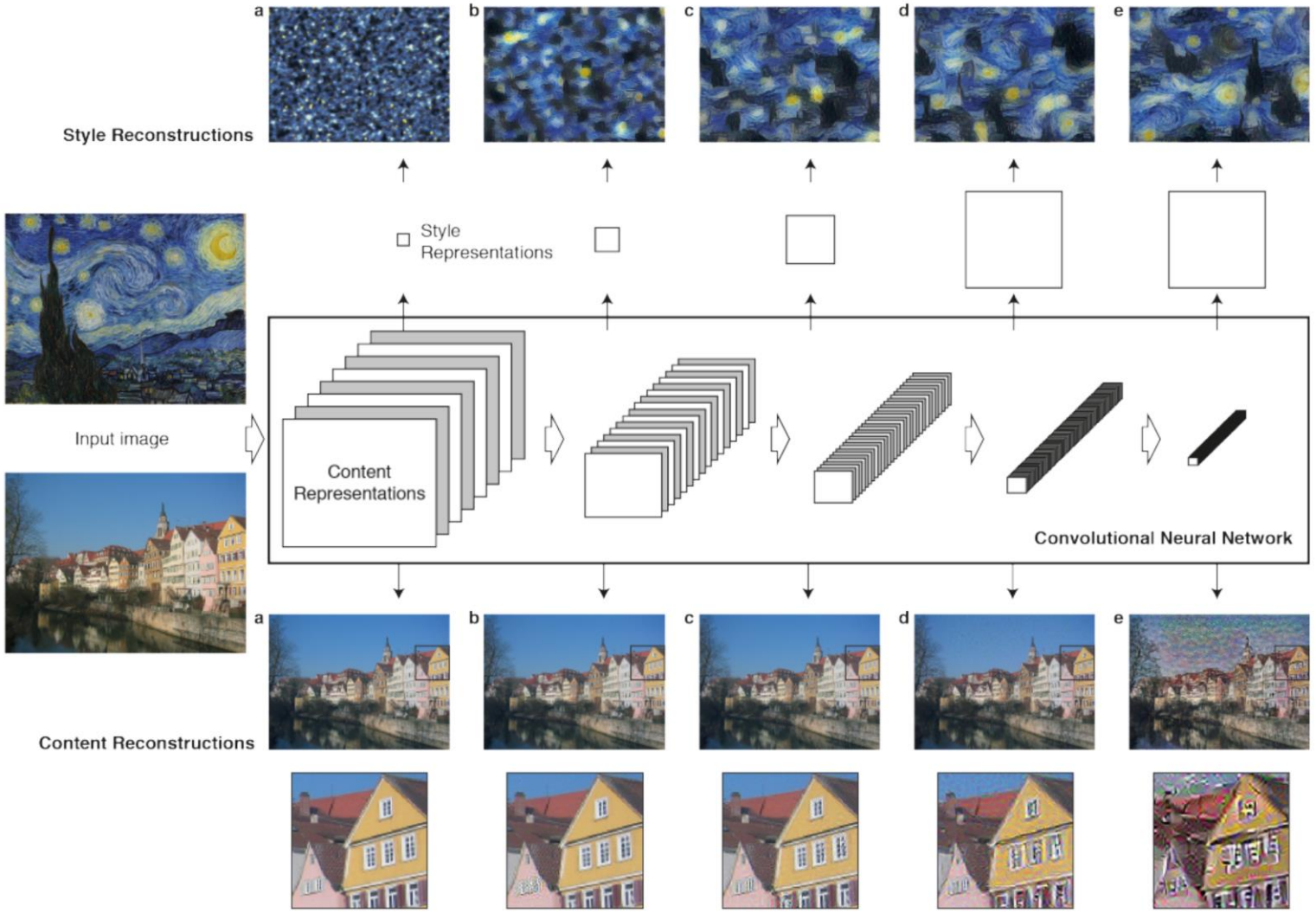
风格迁移开山之作

总损失

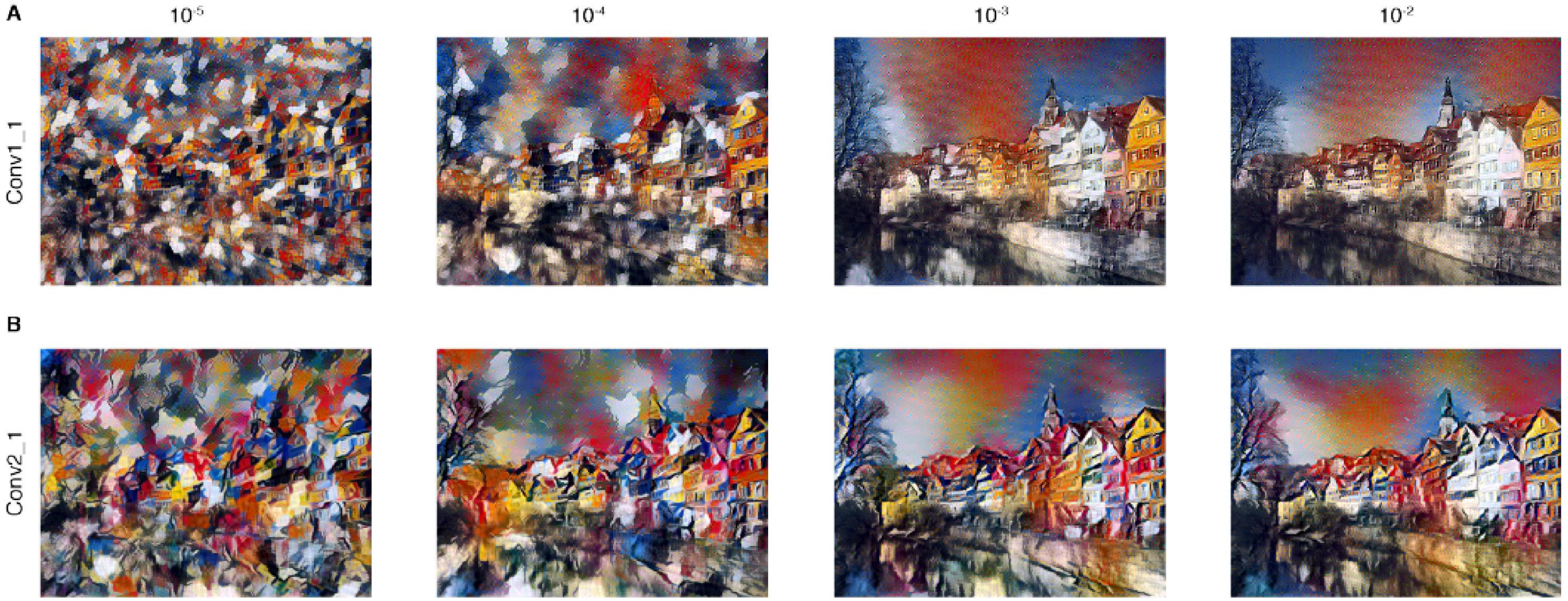
$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

Gatys et al, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

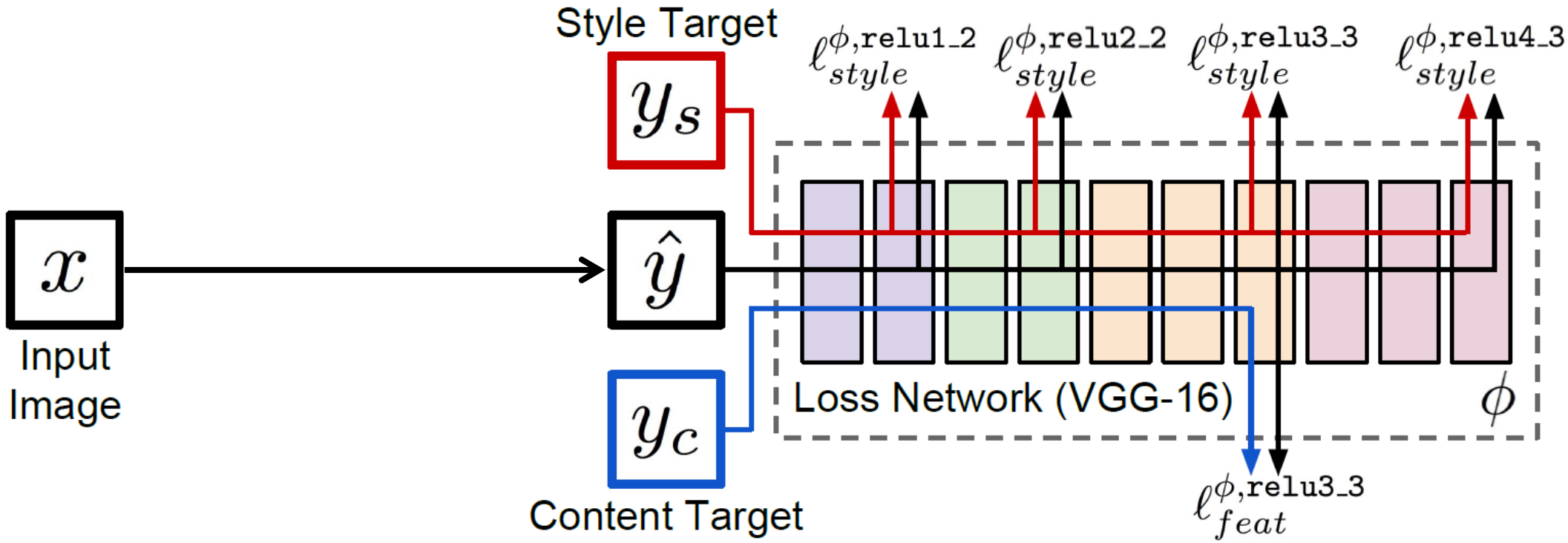
风格迁移开山之作


a
b
c
d
e
Style Reconstructions
Style Representations
Input image
Content Representations
Convolutional Neural Network
Content Reconstructions

风格迁移开山之作

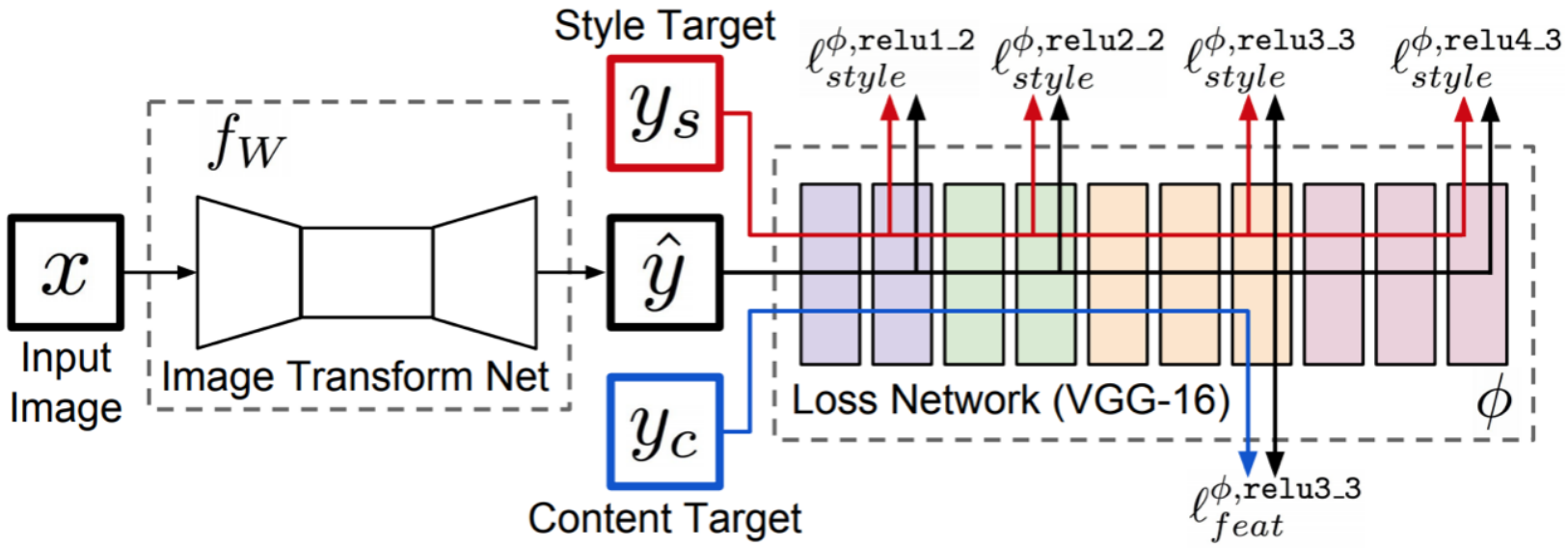


快速风格迁移



Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super resolution." *European Conference on Computer Vision*. Springer, Cham, 2016.

快速风格迁移



内容损失

$$\ell^{\phi,j}_{feat}(\hat{y},y)=\frac{1}{C_jH_jW_j}\|\phi_j(\hat{y})-\phi_j(y)\|_2^2$$

风格损失

$$G^{\phi}_{j}(x)_{c,c'}=\frac{1}{C_jH_jW_j}\sum_{h=1}^{H_j}\sum_{w=1}^{W_j}\phi_j(x)_{h,w,c}\phi_j(x)_{h,w,c'}.$$

$$\ell^{\phi,j}_{style}(\hat{y},y)=\|G^{\phi}_{j}(\hat{y})-G^{\phi}_{j}(y)\|_F^2.$$

快速风格迁移

内容损失

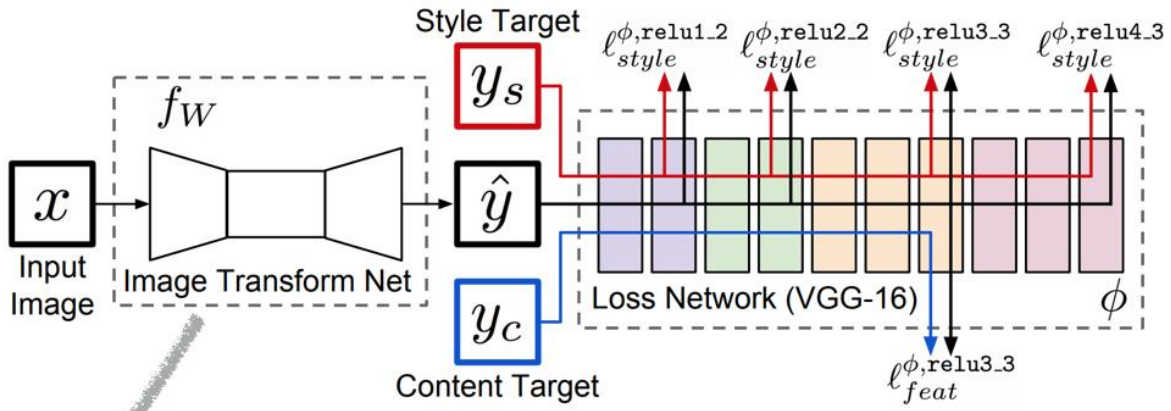
$$\ell_{feat}^{\phi,j}(\hat{y},y)=\frac{1}{C_jH_jW_j}\|\phi_j(\hat{y})-\phi_j(y)\|_2^2$$

风格损失

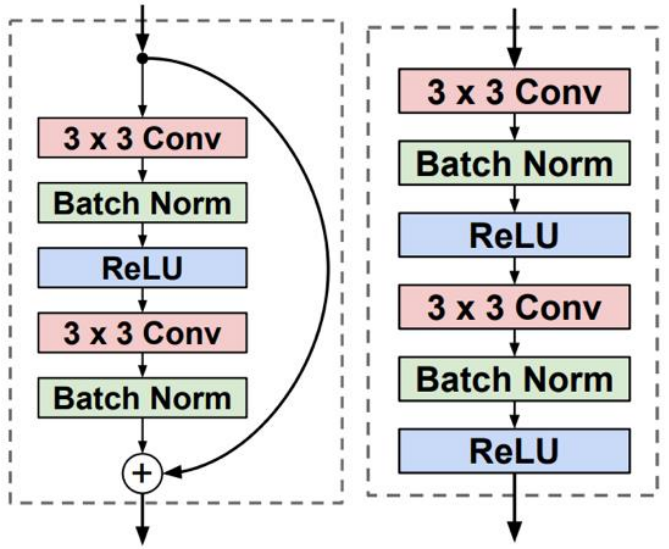
$$G_j^{\phi}(x)_{c,c'}=\frac{1}{C_jH_jW_j}\sum_{h=1}^{H_j}\sum_{w=1}^{W_j}\phi_j(x)_{h,w,c}\phi_j(x)_{h,w,c'}.$$

$$\ell_{style}^{\phi,j}(\hat{y},y)=\|G_j^{\phi}(\hat{y})-G_j^{\phi}(y)\|_F^2.$$

快速风格迁移



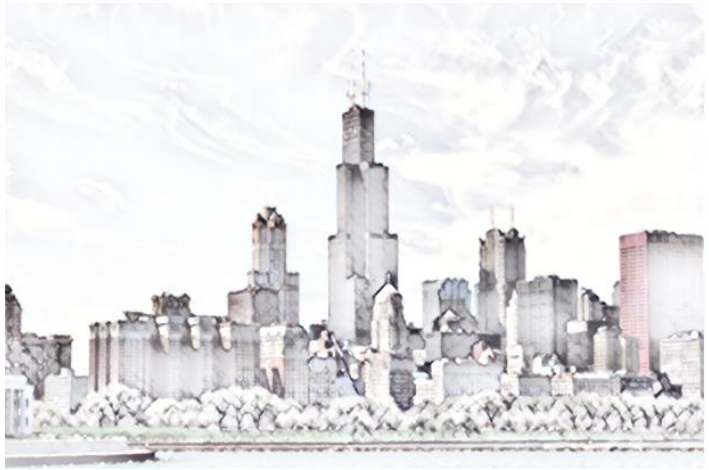
Layer	Activation size
Input	3 × 256 × 256
32 × 9 × 9 conv, stride 1	32 × 256 × 256
64 × 3 × 3 conv, stride 2	64 × 128 × 128
128 × 3 × 3 conv, stride 2	128 × 64 × 64
Residual block, 128 filters	128 × 64 × 64
Residual block, 128 filters	128 × 64 × 64
Residual block, 128 filters	128 × 64 × 64
Residual block, 128 filters	128 × 64 × 64
Residual block, 128 filters	128 × 64 × 64
64 × 3 × 3 conv, stride 1/2	64 × 128 × 128
32 × 3 × 3 conv, stride 1/2	32 × 256 × 256
3 × 9 × 9 conv, stride 1	3 × 256 × 256



变换网络-结构

残差块

快速风格迁移



快速风格迁移

Image Size	Gatys *et al.* [11] 100	300	500	Ours	Speedup 100	300	500
256 × 256	3.17	9.52s	15.86s	**0.015s**	212x	636x	**1060x**
512 × 512	10.97	32.91s	54.85s	**0.05s**	205x	615x	**1026x**
1024 × 1024	42.89	128.66s	214.44s	**0.21s**	208x	625x	**1042x**

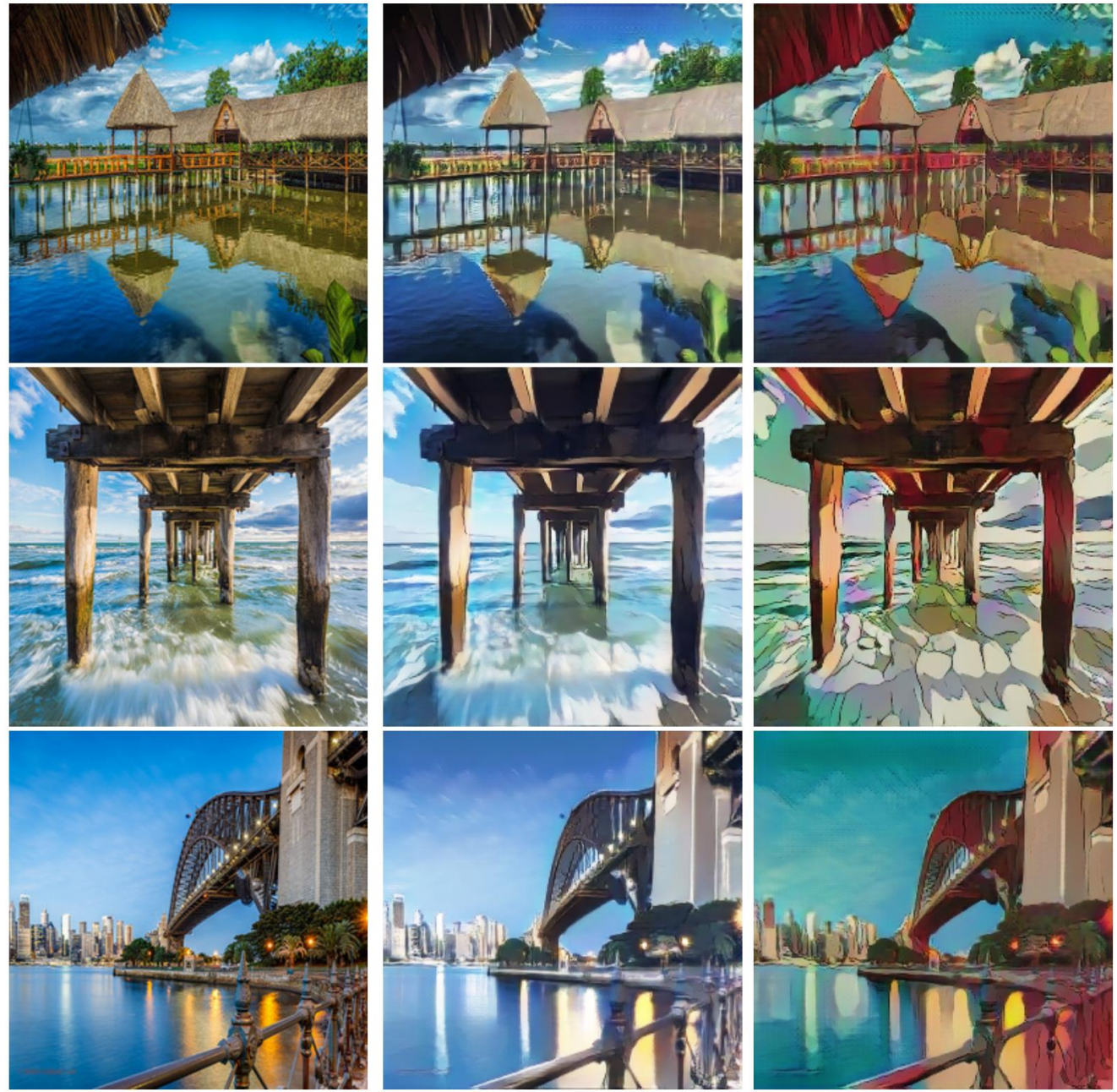
CartoonGAN

https://github.com/Yijunmaverick/CartoonGAN-Test-Pytorch-

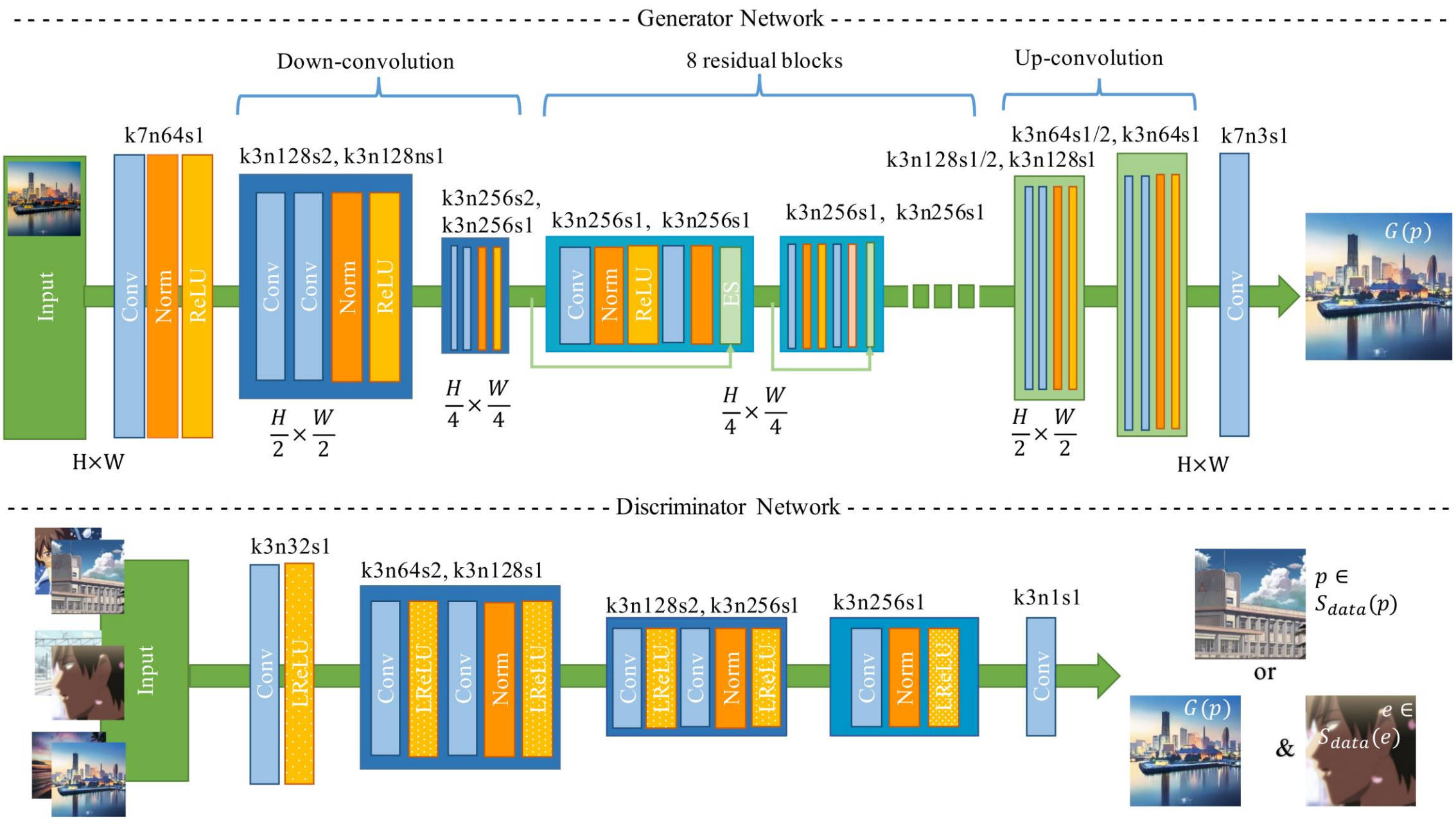
CartoonGAN

(a) Original scene

(b) Our result



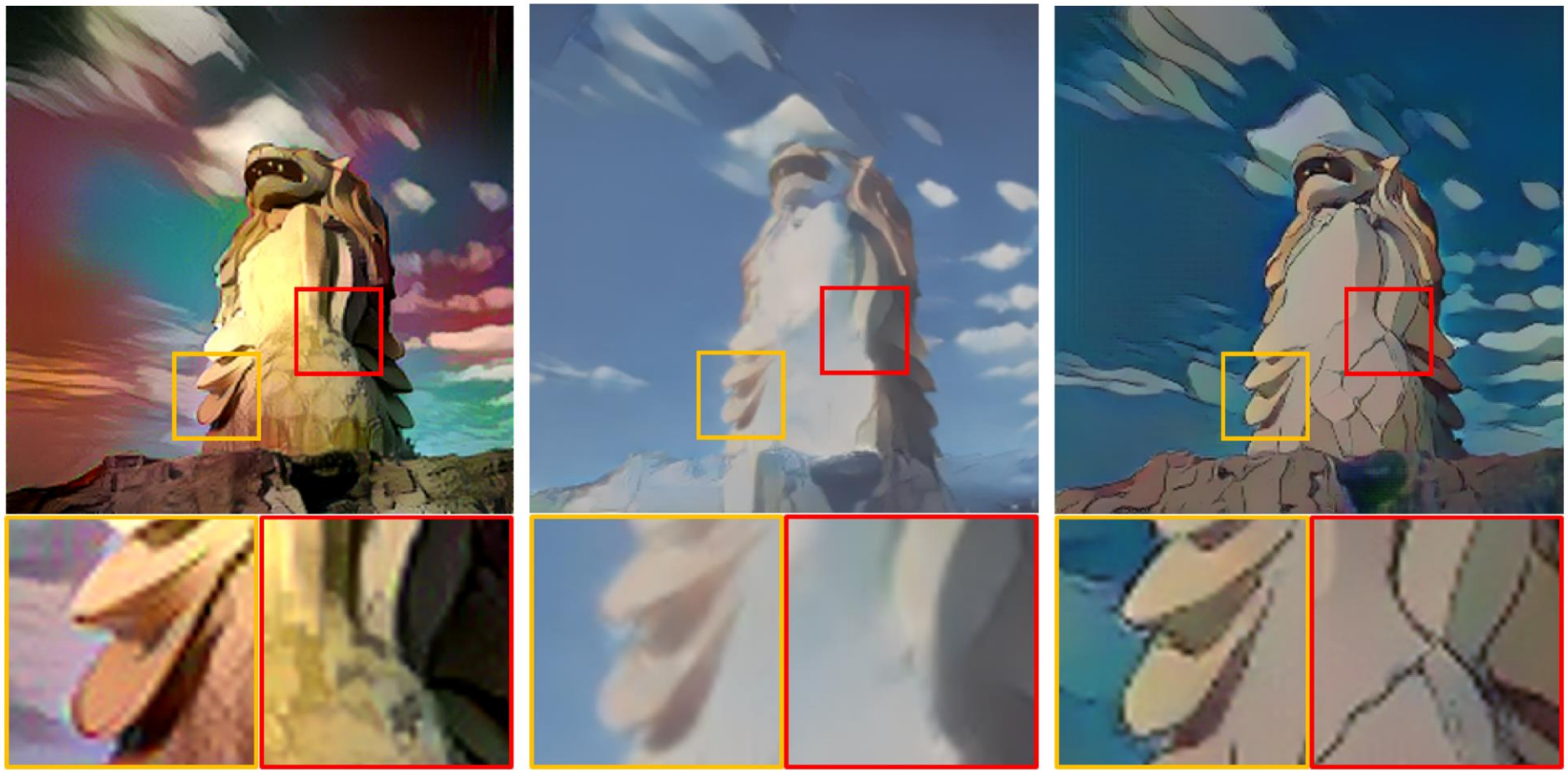
(a) input photo (b) Shinkai style (c) Hayao style


Generator Network
Down-convolution
8 residual blocks
Up-convolution
k7n64s1
k3n128s2, k3n128ns1
k3n256s2, k3n256s1
k3n256s1, k3n256s1
k3n256s1, k3n256s1
k3n128s1/2, k3n128s1
k3n64s1/2, k3n64s1
k7n3s1
Input
Conv
Norm
ReLU
ES
H×W
$\frac{H}{2}\times\frac{W}{2}$
$\frac{H}{4}\times\frac{W}{4}$
$G(p)$
Discriminator Network
k3n32s1
k3n64s2, k3n128s1
k3n128s2, k3n256s1
k3n256s1
k3n1s1
LReLU
$p \in S_{data}(p)$
or
$e \in S_{data}(e)$
&

$$\mathcal{L}(G, D) = \mathcal{L}_{adv}(G, D) + \omega \mathcal{L}_{con}(G, D)$$

$$\begin{aligned}\mathcal{L}_{adv}(G, D) = & \mathbb{E}_{c_i \sim S_{data}(c)}[\log D(c_i)] \\ & + \mathbb{E}_{e_j \sim S_{data}(e)}[\log(1 - D(e_j))] \\ & + \mathbb{E}_{p_k \sim S_{data}(p)}[\log(1 - D(G(p_k)))].\end{aligned}$$

$$\begin{aligned}\mathcal{L}_{con}(G, D) = & \\ & \mathbb{E}_{p_i \sim S_{data}(p)}[||VGG_l(G(p_i)) - VGG_l(p_i)||_1]\end{aligned}$$



(a) NST (b) CycleGAN (c) CartoonGAN

任意风格迁移

Shen, Falong, Shuicheng Yan, and Gang Zeng. "Neural style transfer via meta networks." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

任意风格迁移

情况1

$$\min_{I}\left(\lambda_c||\mathbf{CP}(I;w_f)-\mathbf{CP}(I_c;w_f)||_2^2+\lambda_s||\mathbf{SP}(I;w_f)-\mathbf{SP}(I_s;w_f)||_2^2\right)$$

其中:

- $\mathbf{CP}$ 是内容损失函数
- $\mathbf{SP}$ 是风格损失函数
- λ_c 是内容权重
- λ_s 是风格权重
- w_f 是VGG16的固定权值
- I_s 是风格图像
- I_c 是内容图像
- I 是输入图像

任意风格迁移

情况2

$$\min_{w} \sum_{I_c} \left(\lambda_c ||\mathbf{CP}(I_w; w_f) - \mathbf{CP}(I_c; w_f)||_2^2 + \lambda_s ||\mathbf{SP}(I_w; w_f) - \mathbf{SP}(I_s; w_f)||_2^2 \right)$$

其中：

- I_w 是生成图像，$I_w = \mathcal{N}(I_c; w)$，$\mathcal{N}$ 是图像转换网络

通过对权值的优化，我们可以得到一个快速风格迁移模型，它能够对任何内容图像进行风格转换，输出同一种风格的风格迁移图像。

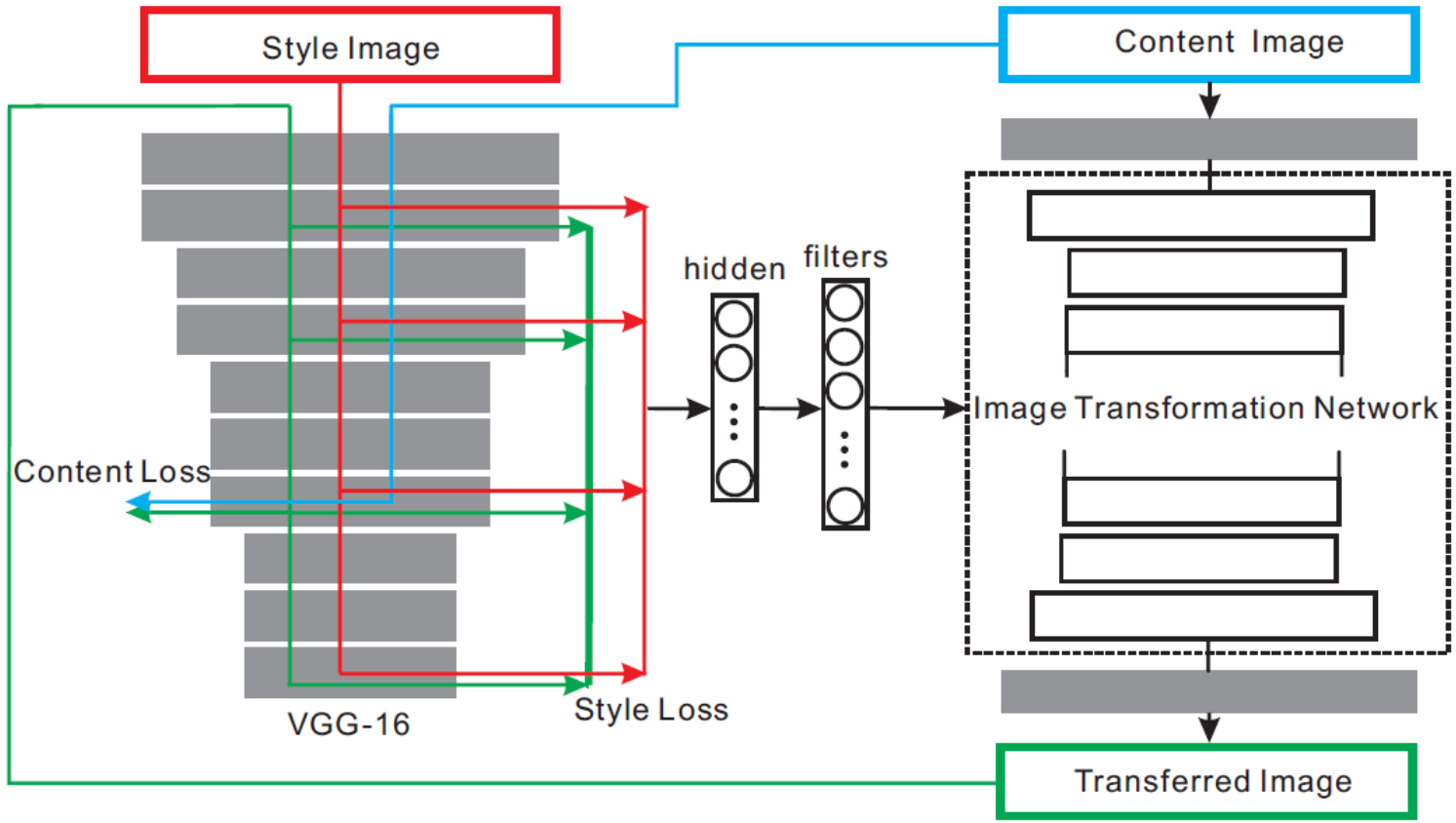
任意风格迁移

情况3

$$\min_{\theta} \sum_{I_c, I_s} \left(\lambda_c ||\mathbf{CP}(I_{w_\theta}; w_f) - \mathbf{CP}(I_c; w_f)||_2^2 + \lambda_s ||\mathbf{SP}(I_{w_\theta}; w_f) - \mathbf{SP}(I_s; w_f)||_2^2 \right)$$

- θ 是 $MetaN$ 的权值
- w_θ 是转换网络的权值，$w_\theta = MetaN(I_s; \theta)$，所以我们可以说转换网络的权值是 MetaNet 通过风格图像生成的。
- I_{w_θ} 是转换网络生成的图像，$I_{w_\theta} = N(I_c; w_\theta)$

任意风格迁移


Style Image
Content Image
hidden
filters
Image Transformation Network
Content Loss
VGG-16
Style Loss
Transferred Image

任意风格迁移

layer	activation size
input	$3 \times 256 \times 256$
Reflection Padding (40×40)	$3 \times 336 \times 336$
$8 \times 9 \times 9$ conv, stride 1	$8 \times 336 \times 336$
$16 \times 3 \times 3$ conv, stride 2	$16 \times 168 \times 168$
$32 \times 3 \times 3$ conv, stride 2	$32 \times 84 \times 84$
Residual block, 32 filters	$32 \times 80 \times 80$
Residual block, 32 filters	$32 \times 76 \times 76$
Residual block, 32 filters	$32 \times 72 \times 72$
Residual block, 32 filters	$32 \times 68 \times 68$
Residual block, 32 filters	$32 \times 64 \times 64$
$16 \times 3 \times 3$ deconv, stride 2	$16 \times 128 \times 128$
$8 \times 3 \times 3$ deconv, stride 2	$8 \times 256 \times 256$
$3 \times 9 \times 9$ conv, stride 1	$3 \times 256 \times 256$

任意风格迁移

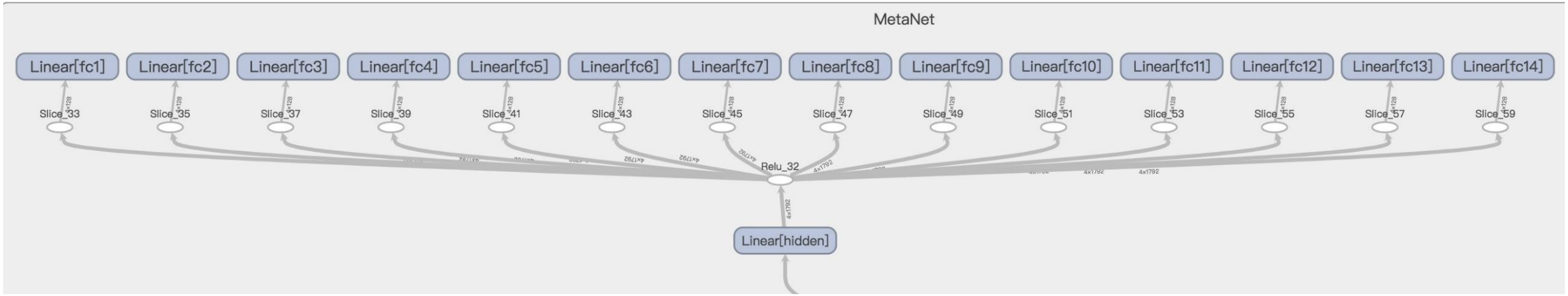
```
class MetaNet(nn.Module):
    def __init__(self, param_dict):
        super(MetaNet, self).__init__()
        self.param_num = len(param_dict)
        self.hidden = nn.Linear(1920, 128*self.param_num)
        self.fc_dict = {}
        for i, (name, params) in enumerate(param_dict.items()):
            self.fc_dict[name] = i
            setattr(self, 'fc{}'.format(i+1), nn.Linear(128, params))

    def forward(self, mean_std_features):
        hidden = F.relu(self.hidden(mean_std_features))
        filters = {}
        for name, i in self.fc_dict.items():
            fc = getattr(self, 'fc{}'.format(i+1))
            filters[name] = fc(hidden[:,i*128:(i+1)*128])
        return filters
```

任意风格迁移

```
metanet.eval()
transform_net.eval()

rands = torch.rand(4, 3, 256, 256).to(device)
features = vgg16(rands);
weights = metanet(mean_std(features));
transform_net.set_weights(weights)
transformed_images = transform_net(torch.rand(4, 3, 256, 256).to(device));

if not is_hvd or hvd.rank() == 0:
    print('features:')
    display([x.shape for x in features])

    print('weights:')
    display([x.shape for x in weights.values()])

    print('transformed_images:')
    display(transformed_images.shape)
```

```
features:
[torch.Size([4, 64, 256, 256]),
 torch.Size([4, 128, 128, 128]),
 torch.Size([4, 256, 64, 64]),
 torch.Size([4, 512, 32, 32])]
weights:
[torch.Size([4, 18496]),
 torch.Size([4, 73856]),
 torch.Size([4, 147584]),
 torch.Size([4, 147584]),
 torch.Size([4, 147584]),
 torch.Size([4, 147584]),
 torch.Size([4, 147584]),
 torch.Size([4, 147584]),
 torch.Size([4, 147584]),
 torch.Size([4, 147584]),
 torch.Size([4, 147584]),
 torch.Size([4, 147584]),
 torch.Size([4, 147584]),
 torch.Size([4, 147584]),
 torch.Size([4, 147584]),
 torch.Size([4, 73792]),
 torch.Size([4, 18464])]
transformed_images:
torch.Size([4, 3, 256, 256])
```

任意风格迁移


MetaNet
Linear[fc1]
Linear[fc2]
Linear[fc3]
Linear[fc4]
Linear[fc5]
Linear[fc6]
Linear[fc7]
Linear[fc8]
Linear[fc9]
Linear[fc10]
Linear[fc11]
Linear[fc12]
Linear[fc13]
Linear[fc14]
Slice_33
Slice_35
Slice_37
Slice_39
Slice_41
Slice_43
Slice_45
Slice_47
Slice_49
Slice_51
Slice_53
Slice_55
Slice_57
Slice_59
Relu_32
Linear[hidden]

任意风格迁移

$$512 * 512 * 147584 = 38688260096$$

$$(64 + 128 + 256 + 512) * 2 = 1920$$

$$1920 * (18496 + 73856 + 147584 * 10 + 73792 + 18464) = 3188060160$$

$$14 * 128 = 1792$$

任意风格迁移

Algorithm 1 Minibatch stochastic gradient descent training of meta networks for neural style transfer. We use $k = 20$ and $m = 8$ in our experiments.

for number of training iterations **do**

- Sample a style image I_s.

for k steps **do**

- Feed-forward propagation of the meta network to get the transformation network

$$w \leftarrow meta\mathcal{N}(I_s; \theta).$$

- Sample minibatch of m input images $\{I_c^{(1)}, ..., I_c^{(m)}\}$.
- Feed-forward propagation of the transformation network to get transferred images.
- Computing the content loss and style loss and update θ

$$\nabla_\theta \sum_{I_c} \left(\lambda_c ||(\mathbf{CP}(I) - \mathbf{CP}(I_c))||_2^2 + \lambda_s ||(\mathbf{SP}(I) - \mathbf{SP}(I_s))||_2^2 \right)$$

end for

end for

任意风格迁移

(a) Input (b) Style (c) Gatys *et al.* (d) John *et al.* (e) Ours (e) Ours-Fast

本次作业

北京大学 PEKING UNIVERSITY

- 寻找一篇2020/2021年风格迁移的文章
- 翻译其摘要和贡献；对代码主体部分进行注释，截图
- 配置环境，测试自己的图片进行风格迁移的结果，截图