MVA Delires 2018/2019 Project: CycleGAN

Gabriel ROMON
gabriel.romon@ensae.fr

Skander KARKAR
skander.karkar@gmail.com

1 Introduction

Image-to-image translation is the task of converting an image from a source domain to a target domain, while preserving the main representations of the input image. For example, one could consider the image of some landscape in the summer and wonder how it would look in the winter. One could also ask how van Gogh or Monet would paint such a landscape. Image-to-image translation assumes that the input image contains *domain-independent features* which must be preserved and *domain-specific features* that should be changed when translating. In the previous summer-to-winter example, domain-independent features include grass, trees, moutains and water, while domain-specific features can be the color of grass and leaves, or the presence of snow.

In [1] the authors introduce a new method for image-to-image translation called CycleGAN. It allows for *unpaired* image-to-image translation: in order to translate an image from domain X to domain Y there is no need to train on paired data points $(x_i, y_i)_{1 \leq i \leq N}$ where y_i is the translation of x_i. Instead, CycleGAN makes use of unpaired data in the form of two sets of images: a source set $\{x_i\}_{1 \leq i \leq N}$ and a target set $\{y_i\}_{1 \leq i \leq M}$, where N and M may differ. The unpaired setting makes it easier to apply image-to-image translation to new domains.

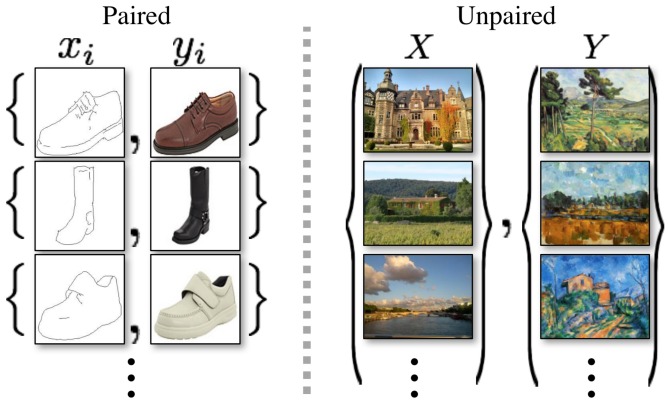

Figure 1: Paired data (left) Unpaired data (right). From [1]

In this report, we start by detailing the inner workings of CycleGAN and its implementation. After that, we reproduce the results of [1] and proceed with our own experiments.

2 CycleGAN

2.1 Formulation

2.1.1 Description of the model

Let $p_{\text{data},x}$ and $p_{\text{data},y}$ denote the respective data-generating distributions for images in domain X and Y. These are known only through samples $\{x_i\}_{1\leq i\leq N}$ and $\{y_i\}_{1\leq i\leq M}$ where $x_i \sim p_{\text{data},x}$ and $y_i \sim p_{\text{data},y}$. CycleGAN is trained to learn 4 functions: two *mapping functions* $G : X \to Y$ and $F : Y \to X$, and two *adversarial discriminators* $D_X : X \to \mathbb{R}$ and $D_Y : Y \to \mathbb{R}$.

F and G act as translators that map images from one domain into the other. D_Y encourages G to produce outputs indistinguishable from the images in Y, and vice-versa with D_X. However, it is noted in [1] that adversarial training alone is not sufficient to get satisfying translation results. Indeed, the discriminator D_Y pushes G to induce an output distribution $\hat{p}_G$ over domain Y such that $\hat{p}_G \approx p_{\text{data},y}$. However, this implies nothing on the relevance of the output $G(x_i)$ with respect to its input x_i. In the introduction we stated that domain-independent features of x_i should be preserved, but adversarial training alone is not enough to guarantee this since any random permutation of images in the target domain Y will induce the same output distribution $\hat{p}_G$.

The authors suggest to impose more structure by adding two cycle consistency constraints: the *forward* constraint $F(G(x)) \approx x$ and the *backward* constraint $G(F(y)) \approx y$.

The interactions between the different parts of CycleGAN are well summed up in Figure 2 (copied from [1]).

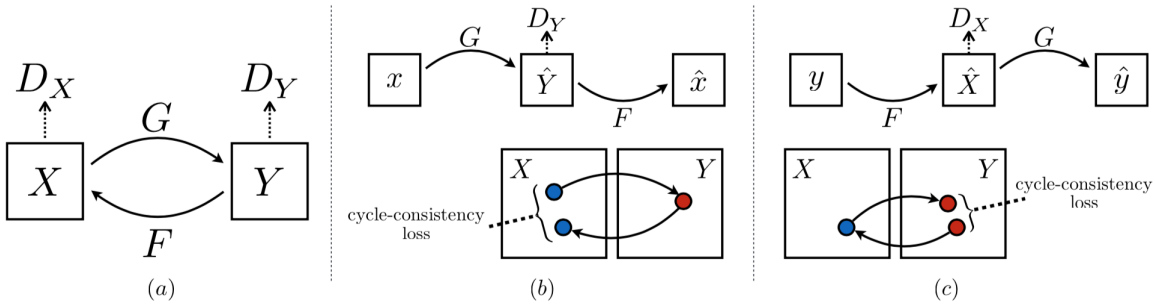

Figure 2: Different parts of CycleGAN and their interactions. From [1]

2.1.2 Loss of the model

The loss follows naturally from the previous description: it is the sum of two *adversarial losses* and a *cycle consistency loss*.

The first adversarial loss relates G and D_Y. It is the standard loss used in GANs, introduced in [2].

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

The second adversarial loss writes similarly as

$$\mathcal{L}_{\text{GAN}}(F, D_X, Y, X) = E_{x\sim p_{\text{data},x}}[\log D_X(x)] + E_{y\sim p_{\text{data},y}}[\log(1 - D_X(F(y)))]$$

The cycle consistency loss is defined by the authors as

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

As a result, the full loss writes as

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

where λ is a parameter that provides balance between the adversarial part and the cycle consistency component of the loss.

The final optimization problem to be solved is

$$\arg\min_{G,F} \max_{D_X,D_Y} \mathcal{L}(G, F, D_X, D_Y)$$

2.2 Implementation

2.2.1 Architectures

Generator

The generator architecture is inspired from the *image transformation network* used in [3] for style transfer and super-resolution. For input images with sides larger than 256 the network is shown in Table 1. It is made up of 3 downsampling layers (stack of convolutional layers) followed by 9 *residual blocks* and 3 upsampling layers (stack of transposed convolutional layers). Note that *fractionally strided convolution* with stride $\frac{1}{2}$ is the same thing as transposed convolution with stride 2. Below (Table 1) is the the architecture as described in the paper. In the `pytorch` code of the authors available on `GitHub` however, no normalization was performed after the last convolution and the last activation was a hyperbolic tangent.

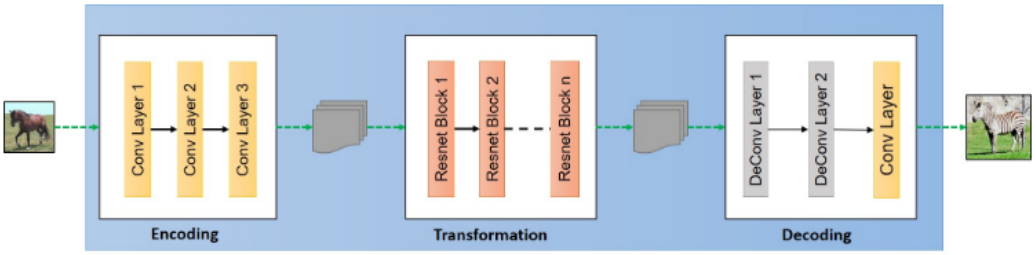

Figure 3: Generator architecture. From [4]

7×7	Conv-InstanceNorm-ReLU	64 filters	stride 1
3×3	Conv-InstanceNorm-ReLU	128 filters	stride 2
3×3	Conv-InstanceNorm-ReLU	256 filters	stride 2
9 residual blocks with 256 3×3 filters in each layer			
3×3	Conv-InstanceNorm-ReLU	128 filters	stride $\frac{1}{2}$
3×3	Conv-InstanceNorm-ReLU	64 filters	stride $\frac{1}{2}$
7×7	Conv-InstanceNorm-ReLU	3 filters	stride 1

Table 1: Generator architecture (first row is first hidden layer)

9 residual blocks are used for 256×256 images and 6 for 128×128 images. All our training images are 256×256 so we use 9 blocks. For the output image to have the same size as the input, as well as to reduce artifacts at the border of the output, the authors use *reflection padding* of size 3 before the first and the last convolutional layers. This pads the image with 3 pixels on each side by using the reflections of its boundary. They also add reflection padding of size 1 before the convolutional layers inside the residual blocks.

Instance normalization is used after each of the downsampling/upsampling layer. Since the batch size is set to 1 in the paper, it is equivalent to batch normalization (without the affine shifting step). In a given convolutional layer and a given feature map i, all the x_{ijk} (where j and k denote spatial dimensions) are normalized as $y_{ijk} = \frac{x_{ijk} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$ with $\mu_i = \frac{1}{WH} \sum_{l=1}^{W} \sum_{m=1}^{H} x_{ilm}$ and $\sigma_i^2 = \frac{1}{WH} \sum_{l=1}^{W} \sum_{m=1}^{H} (x_{ilm} - \mu_i^2)$. The y_{ijk} are then fed to the corresponding activation function. Instance normalization was introduced in [5, 6] and it is known to significantly improve the quality of style transfer. The authors of CycleGAN state that [3] uses instance normalization, but this is not true. [3] uses batch normalization and the batch size is set to 4, so they probably meant to say that they use batch normalization as in [3], but since their batch size is 1, it boils down to instance normalization.

Residual blocks were introduced in [7] as devices that allow to train very deep neural networks reliably. At the time, problems of vanishing/exploding gradients in deep networks (~ 10 layers) had already been addressed with custom initialization methods and batch normalization. However, [7] mentions the *degradation* problem when training deeper networks (> 20 layers): after a number of epochs the training accuracy plateaus and adding more layers results in higher training errors. The

authors consider a shallow network and its deeper counterpart constructed by adding layers. After training, the deeper network is expected to outperform the other since the additional layers could simply learn the identity function. This remark motivates the architecture of the residual block: given a target function $\mathcal{H}(x)$, the authors suggest to learn instead the residual function $\mathcal{F}(x) := \mathcal{H}(x) - x$ with a convolutional network and to retrieve $\mathcal{H}(x)$ by adding x to $\mathcal{F}(x)$ with a skip connection. In the special case where $\mathcal{H}(x) = x$, they make the case that it is much easier to make a network learn $\mathcal{F}(x) = 0$ rather than learning the identity mapping directly. Residual blocks made it possible to successfully train networks with more than 1000 layers. Two residual architectures are shown in Figure 4. The block on the left follows the exposition of [7], while the block on the right was shown in [8] to provide better performance, which is why it was subsequently used in [3]. Since the authors of CycleGAN mimicked the network of [3], the block on the right was used in their implementation of residual blocks.

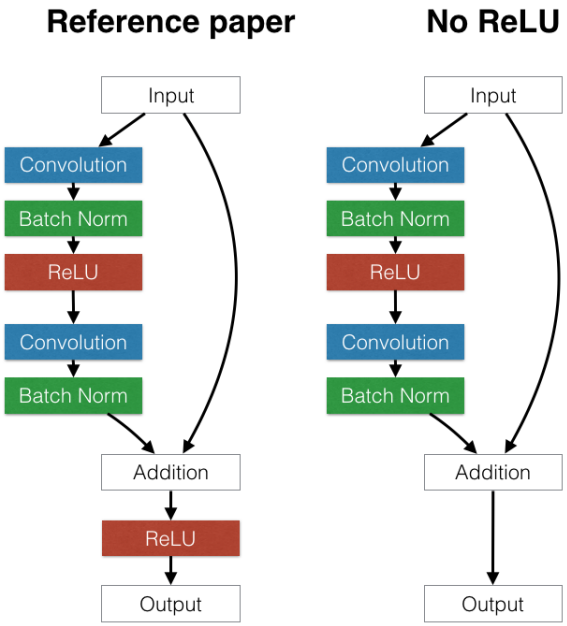

Figure 4: Two architectures of residual blocks

Discriminator

The discriminator referred to by the authors as PatchGAN is actually a simple stack of convolutional layers. The architecture is detailed in Table 2. After stacking the layers, the final layer has a 70×70 view of the original image.

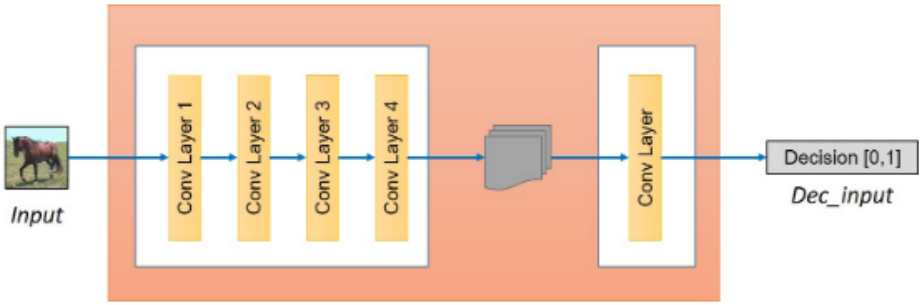

Figure 5: Discriminator architecture. From [4]

Layers				Effective receptive field
4×4	Conv-LeakyReLU	64 filters	stride 2	4×4
4×4	Conv-InstanceNorm-LeakyReLU	128 filters	stride 2	10×10
4×4	Conv-InstanceNorm-LeakyReLU	256 filters	stride 2	22×22
4×4	Conv-InstanceNorm-LeakyReLU	512 filters	stride 1	46×46
4×4	Conv	1 filter	stride 1	70×70

Table 2: Discriminator architecture (first row is first hidden layer)

2.2.2 Training

The authors introduce a number of modifications to the model presented above in order to improve training stability and the quality of output images.

They replace the negative log-likelihood in the adversarial losses $\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y)$ and $\mathcal{L}_{\text{GAN}}(F, D_X, Y, X)$ with the least-squares loss (as in [9]). The problem is then formulated in minimization terms so that the successive optimization problems become

$$G = \arg\min_G \mathbb{E}_{x \sim p_{\text{data},x}} \left[(D_Y(G(x)) - 1)^2 \right] + \lambda \mathcal{L}_{\text{cyc}}(G, F) \tag{1}$$

$$D_Y = \arg\min_D \mathbb{E}_{y \sim p_{\text{data},y}} \left[(D(y) - 1)^2 \right] + \mathbb{E}_{x \sim p_{\text{data},x}} \left[D(G(x))^2 \right] \tag{2}$$

and likewise

$$F = \arg\min_F \mathbb{E}_{y \sim p_{\text{data},y}} \left[(D_X(F(y)) - 1)^2 \right] + \lambda \mathcal{L}_{\text{cyc}}(G, F) \tag{3}$$

$$D_X = \arg\min_D \mathbb{E}_{x \sim p_{\text{data},x}} \left[(D(x) - 1)^2 \right] + \mathbb{E}_{y \sim p_{\text{data},y}} \left[D(F(y))^2 \right] \tag{4}$$

In [9] the authors note that using least-squares loss for GANs mitigates vanishing gradients when training the generator: fake samples that are classified by the discriminator as real are pushed by the squared loss towards the manifold of real data, so fake samples are encouraged to look more like real data, even if they already fool the discriminator (this phenomenon does not occur with the usual cross-entropy loss).

The way the discriminators are trained is also peculiar. Following the work done in [10], they use two *buffers* to store the outputs of the current and previous generators. For the sake of simplicity consider the buffer for the translation $X \to Y$. It is filled with the first 50 images from G at epoch 1. The buffer is updated after each gradient step on G and before each gradient step on D_Y: G generates a fake image but with probability 0.5 the buffer returns an image that it currently stores (i.e. a fake image from the generator of a previous epoch), this older fake image is used in the computation of D_Y's loss, but it is replaced in the buffer by the fake image from the latest generator. And with probability 0.5, the latest fake image is used but discarded (i.e. not added to the buffer). According to [10] this technique endows the discriminators with memory abilities: they should be able to spot fake images generated not only by the latest generator but also by the generators from the previous epochs. This results in improved training stability.

When going from paintings to real-world photos the authors observed that the generator often changes the color composition of images, transforming daytime paintings into sunset photos. To address this problem, they add a loss which they call *identity loss* that encourages each generator to preserve colors: $\mathcal{L}_{\text{identity}}(G, F) = \mathbb{E}_{y \sim p_{\text{data},y}} [\|G(y) - y\|_1] + \mathbb{E}_{x \sim p_{\text{data},x)}} [\|F(x) - x\|_1]$. This loss forces the generator to act as the identity when fed with images from its target domain.

Training is carried out for 200 epochs using Adam optimizer with $\beta_1 = 0.5$ and with learning rate 0.0002 for the first 100 epochs and $0.0004 - 0.000002 \times$ epoch for the next 100 epochs (i.e linearly decreasing to 0). Weights are initialized from a Gaussian distribution $\mathcal{N}(0, 0.02)$. The weight λ of the cycle loss is set to 10 and the weight of the identity loss when used is set to 5. The training of the discriminators is slowed down by dividing discriminator losses in (2) and (4) by 2. The batch size is 1 and if the training sets $\{x_i\}$ and $\{y_i\}$ are of unequal size, some images from the smallest set are reused. The general training loop is the following:

Algorithm 1 Training loop of CycleGAN

Initialize n_epochs $= 200$ and n_batches $= \max(\#\{x_i\}, \#\{y_i\})$
for epoch $= 1, 2, ..,$ n_epochs **do**
  for $i = 1, 2..,$ n_batches **do**
    Get next data sample (x_i, y_i)
    Train generator G on (x_i, y_i) via (1) and get fake image $\tilde{y}_i = G(x_i)$
    Buffer treatment of $\tilde{y}_i$ as explained above
    Train disctiminator D_Y on $(y_i, \tilde{y}_i)$ via (2)
    Train generator F on (x_i, y_i) via (3) and get fake image $\tilde{x}_i = F(y_i)$
    Buffer treatment of $\tilde{x}_i$ as explained above
    Train disctiminator D_X on $x_i, \tilde{x}_i$ via (4)

2.2.3 Preprocessing

A random horizontal flip and a random crop (resizing to 286 then croping back to 256) of the input images are performed.

3 Results

3.1 Code

We implemented CycleGAN in `pytorch` and `tensorflow`. The code in `tensorflow` is more complete and is available on `GitHub` (https://github.com/skander-karkar/MVA-Delires-CycleGAN). Minor differences exist with the authors' code. First, the number of filters in each layer of the generator (table 1) was divided by 2 in most experiments. This almost halves the training time, 200 epochs on around 1000 images from each domain taking around 20 hours instead of 40 on a GTX 1080 TI. Second, both `resnet` architectures (Figure 4) were tested, with no clear difference in results noticed. Finally, a skip connection was added to the generator (between input and final output, before the hyperbolic tangent activation). According to [11], this leads to faster convergence, which we observed on some datasets.

By default, unless specified otherwise, the results are with half as much generator filters as in the article, with skip connection and the original residual block architecture.

Results are arranged in rows that contain from left to right: $x_i, y_i, G(x_i), F(y_i), F(G(x_i)), G(F(y_i))$

3.2 Datasets

We tested the model on 7 classical datasets (monet2photo, vangogh2photo, cezanne2photo, ukiyoe2photo, apple2orange, horse2zebra, summer2winter) downloaded from [12].

3.3 Painting2photo

First wee look at results on painting2photo datasets (Figures 6, 7, 8 and 9). Going from a painting to a photo is obvisouly easier than the other way around. However some artefacts still appear in some cases, especially if a uniform region (a clear blue sky for example) is present in the original image (e.g. third row of Figure 9).

The identity loss mentioned above was proposed to force the generator even more to preserve color composition. Even though we did not observe the problem of mapping say daytime scenes to sunset scenes, we tried adding the identity loss on the vangogh2photo dataset (Figure 10).

3.4 Ablation study

All these figures show that the cycle consistency is very well respected. The authors of CycleGAN perform an ablation study in [1] where they remove parts of the loss and note the effect on the results. We ran the model without the cycle loss on the vangogh2photo dataset (Figure 11) and verified that not only were the cycle images $F(G(x))$ and $G(F(y))$ far from x and y, but the quality of the generated images $G(x)$ and $F(y)$ also decreased.

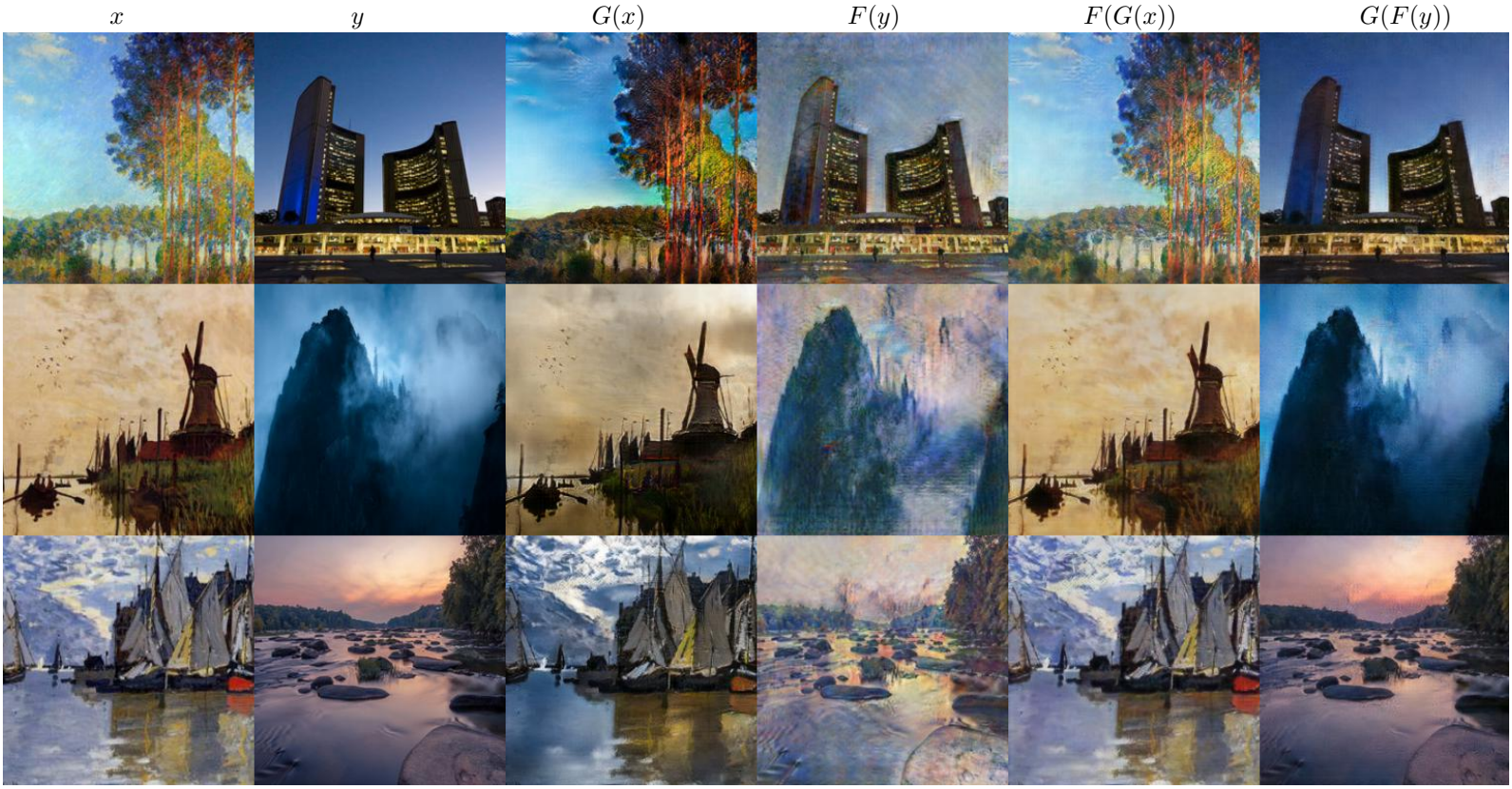

Figure 6: monet2photo results

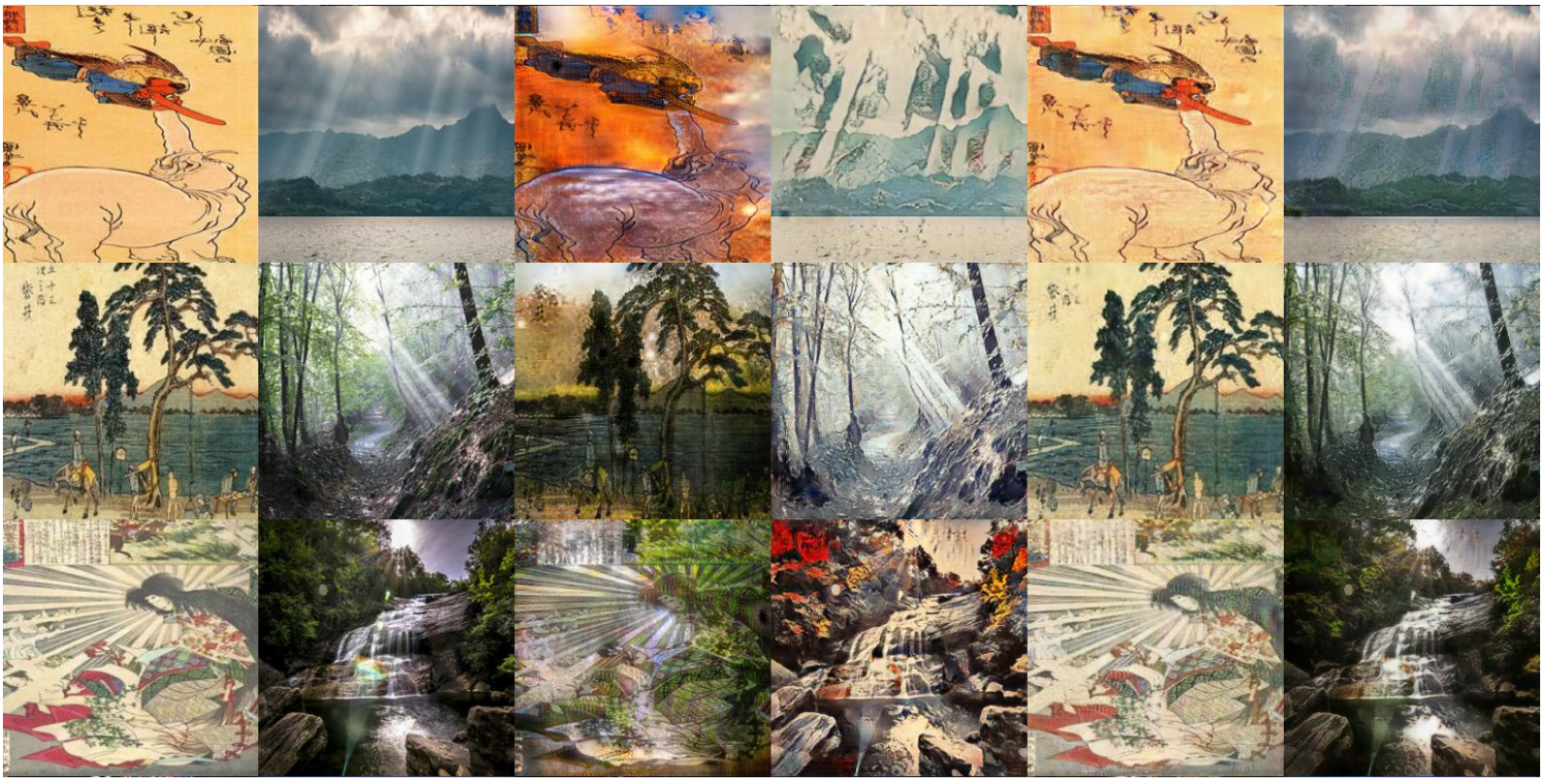

Figure 7: ukiyoe2photo results

Figure 8: cezanne2photo results

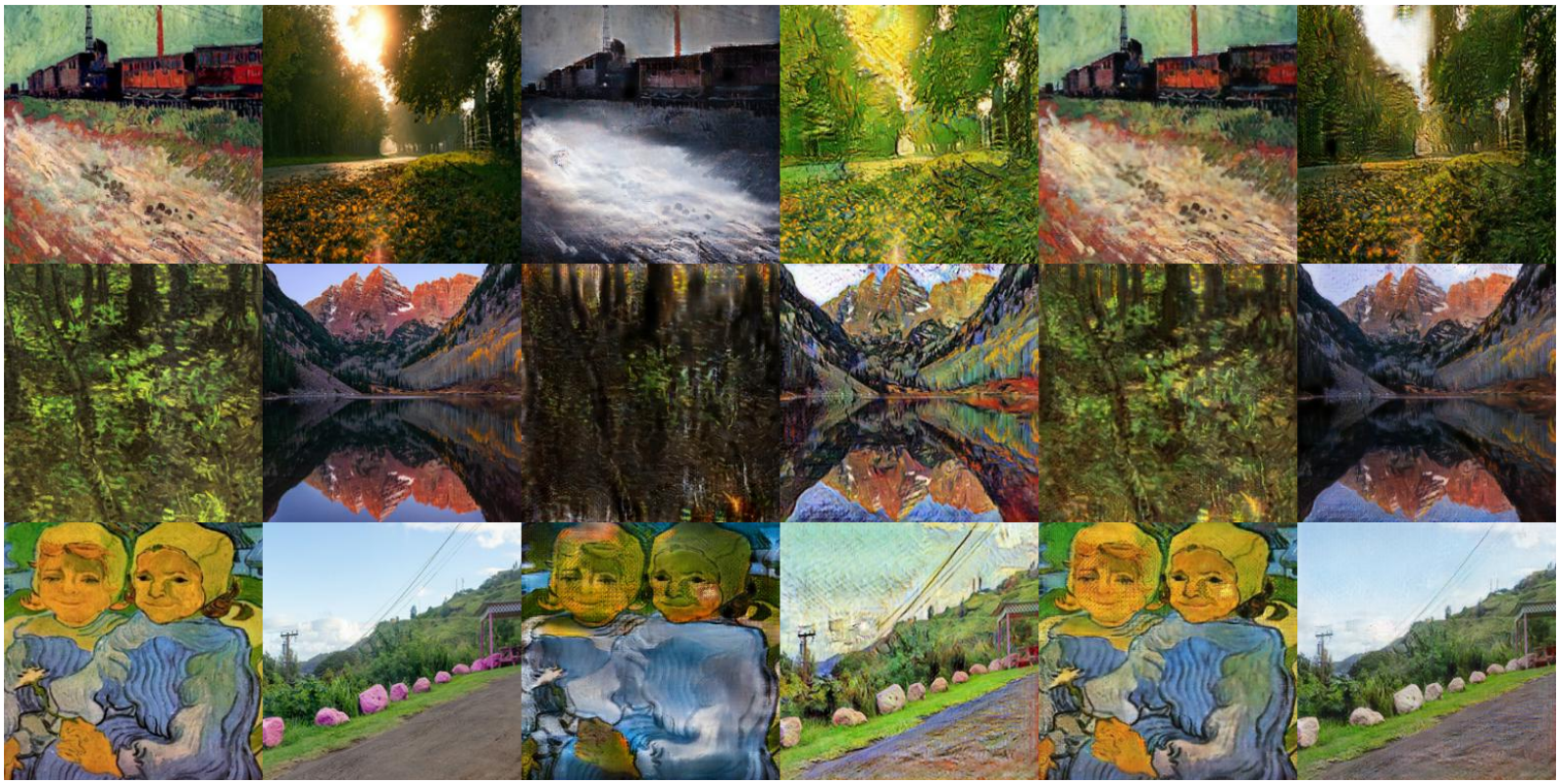

Figure 9: vangogh2photo results

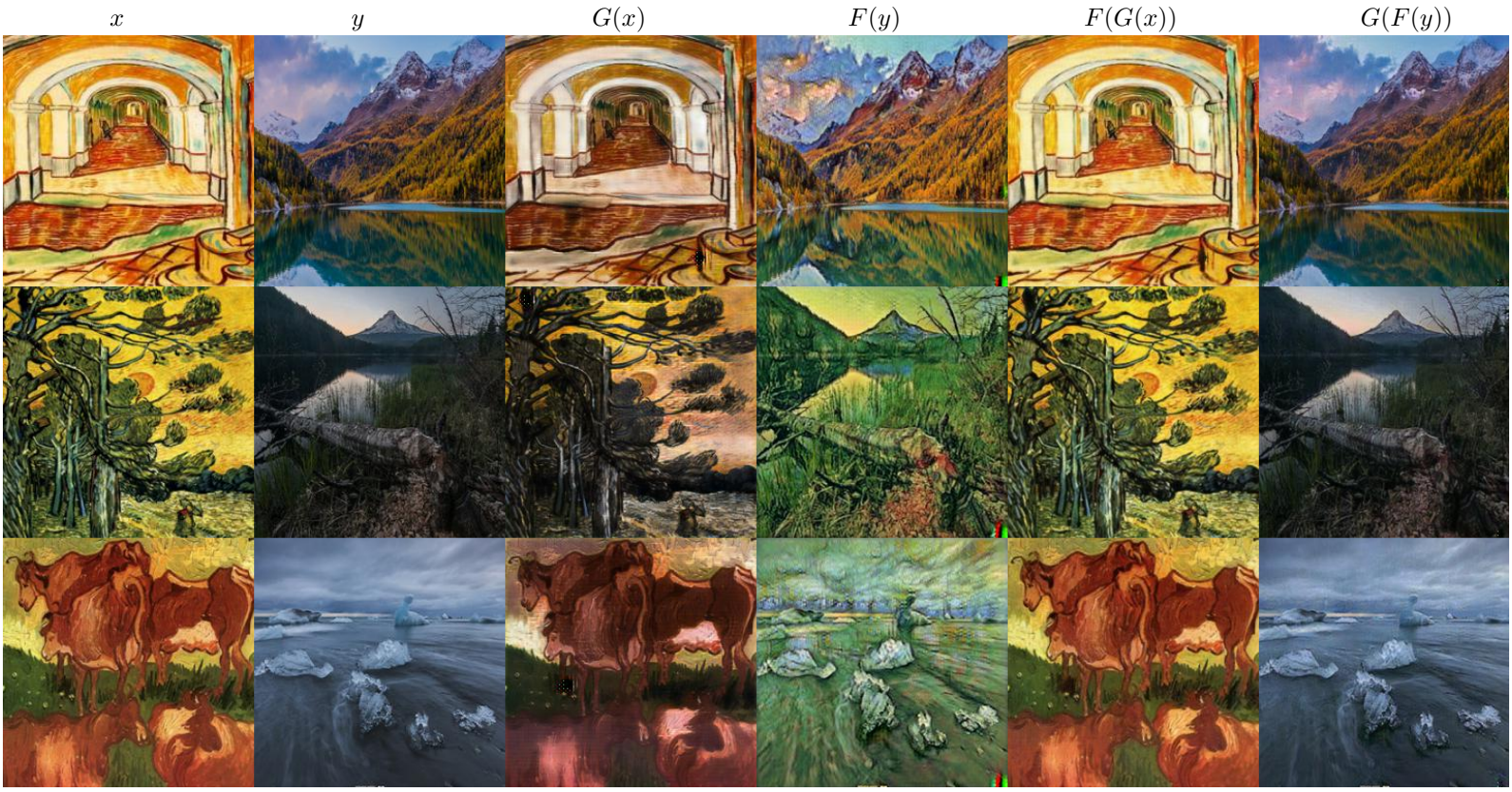

Figure 10: vangogh2photo results, with identity loss

Figure 11: vangogh2photo results, without cycle loss

3.5 Other datasets

Below are results on datasets horse2zebra (Figure 12), apple2orange (Figue 13) and summer2winter (Figure 14). Again, cycle consistency is very well respected.

On horse2zebra, transforming a horse into a zebra works usually very well and seems to come more easily to the model than the opposite, with some stripes often remaining when transforming a zebra into a horse.

On apple2orange, the color transfer is performed well. However, the geometric transformation between an apple and an orange is not successful. Failure to apply geometric transformations is a recognized shortcoming of CycleGAN [1].

On summer2winter, the network learns early to add a sun when transforming a winter scene into a summer one. However, when a lot of snow is present, it often fails to remove all of it (e.g. bottom row of Figure 14).

3.6 Skip connection

The skip connection over the entire generator does seem to produce better early results in some cases, for example on horse2zebra (see Figure 15) and on vangogh2photo (Figure 16).

However, when major changes are needed it mitigates them too much (Figure 17 on summer2winter).

According to the FAQ section of the authors' `GitHub` on CycleGANs, the generators often learn early on to invert the colors of the image and then never learn to undo the inversion (e.g. Figure 15). This skip connection might be a solution to this problem.

Figure 12: horse2zebra results

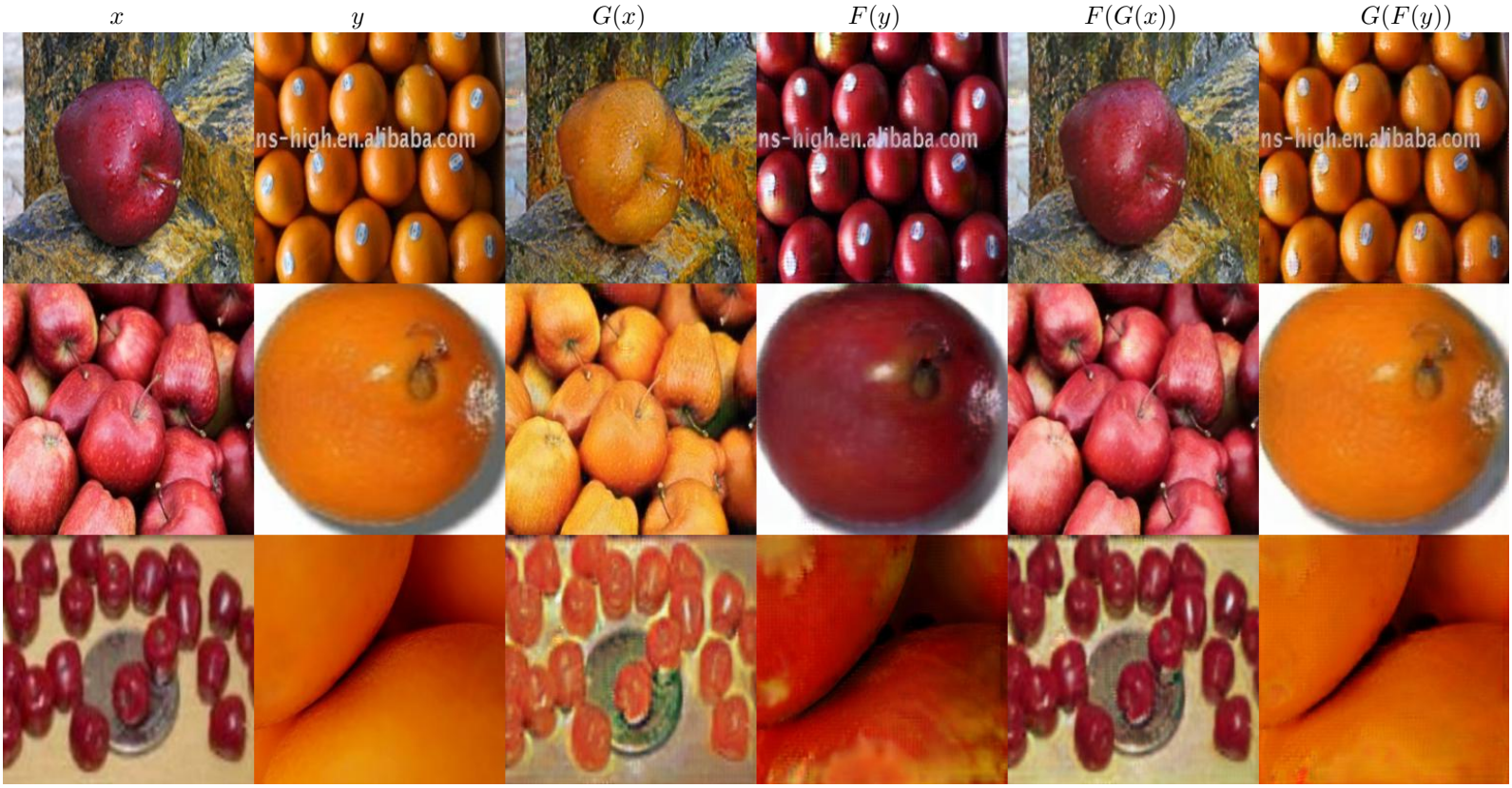

Figure 13: apple2orange results

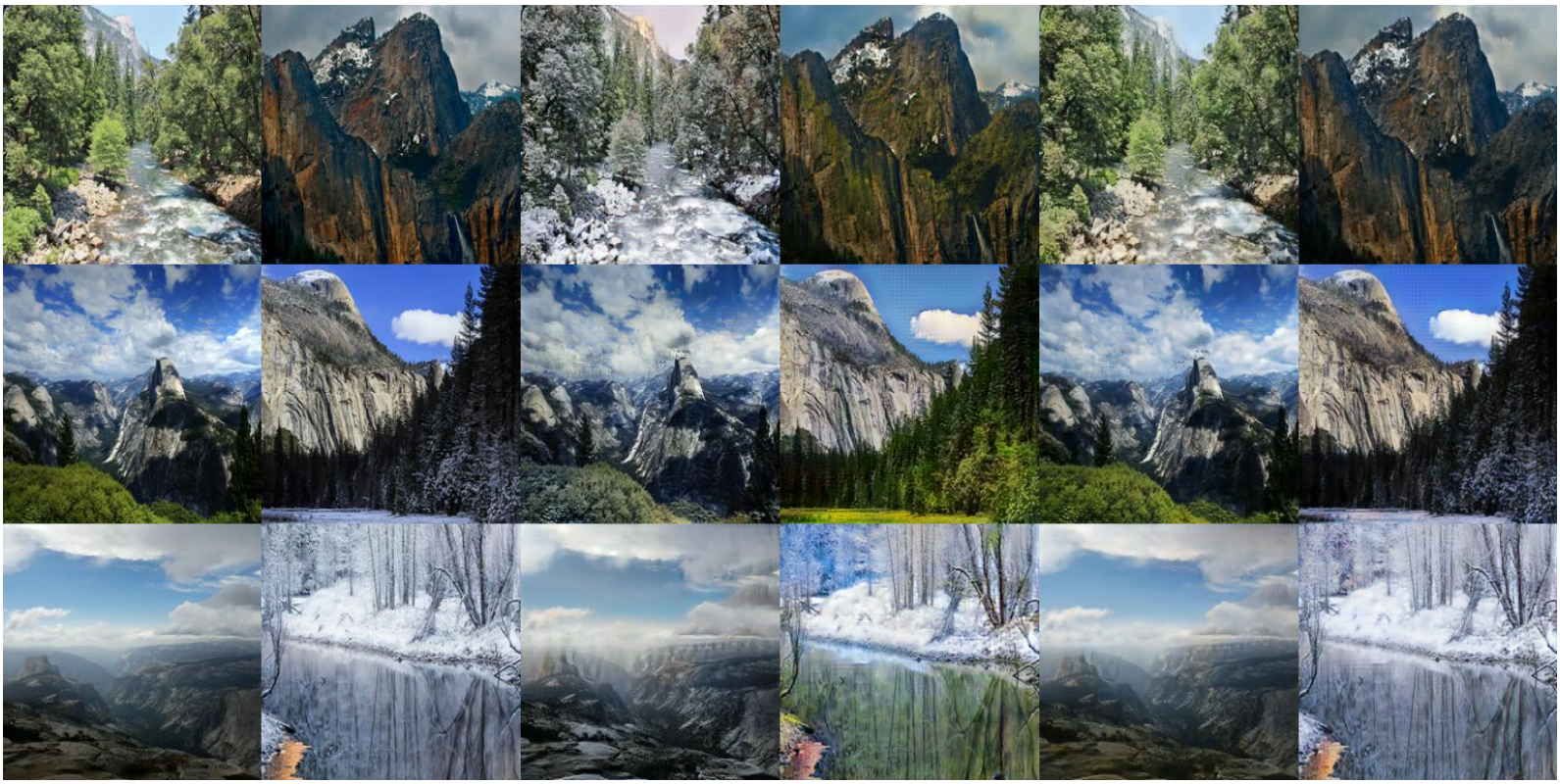

Figure 14: summer2winter results

Figure 15: Left to right: two fake zebras after 19 epochs without skip connection and two fake zebras after 13 epochs with skip connection

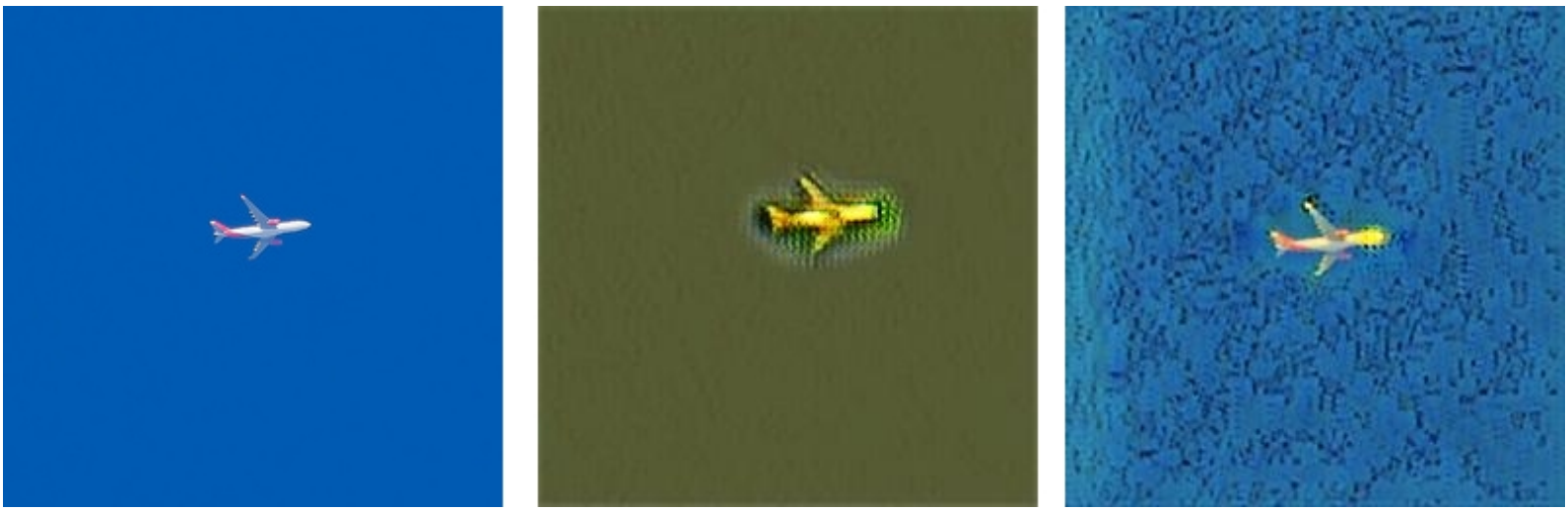

Figure 16: Left to right: real photo, fake Vangogh after 32 epochs without skip connection, fake Vangogh after 29 epochs with skip connection

Figure 17: Left to right: real winter-fake summer after 30 epochs without skip connection and real winter-fake summer after 30 epochs with skip connection

3.7 More generator filters

A problem with CycleGAN and GANs in general, is that not only do the losses usually not converge and not convey information about the quality of the results, but the results are also highly dependent of the random initialization. Experiments therefore need to be run multiple times and since they take a long time, it is difficult to objectively compare results. So it is difficult to say whether doubling the number of generator filters to use the same number as the authors has indeed improved results.

Below are results with exactly the same architecture as the authors' (so without skip connection).

Figure 18: ukiyoe2photo results with bigger generators

Figure 19: vangogh2photo results with bigger generators

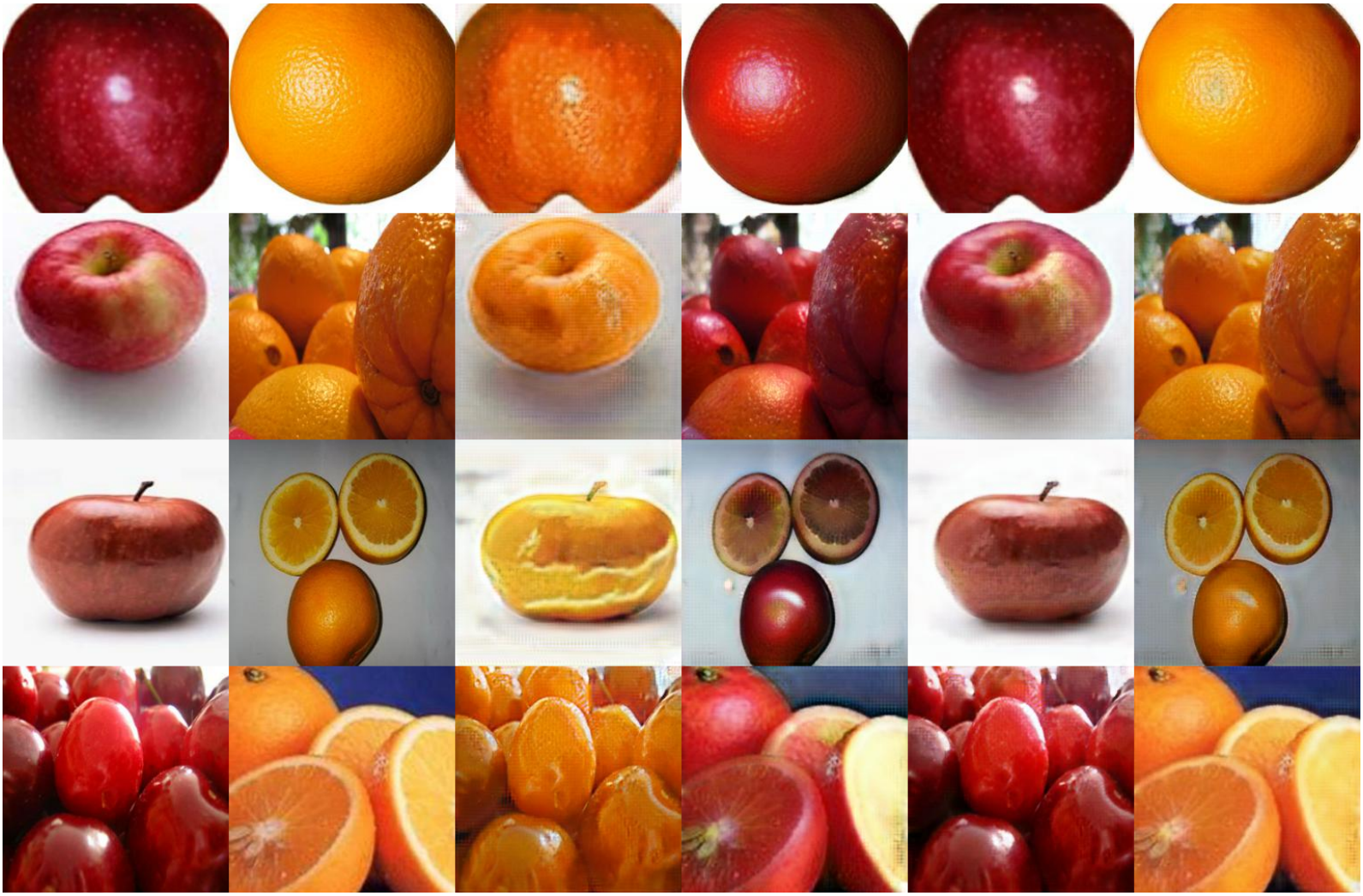

Figure 20: apple2orange results with bigger generators

4 Conclusion

CycleGAN provides a working framework for unpaired image-to-image translation. Although numerous tricks are required to stabilize training, the method was shown to work well on several datasets, without any extensive preprocessing or model tuning. It is especially powerful when only color and texture changes are needed.

However, it performs poorly between datasets that require geometric transformations (e.g. from cats to dogs). Compared to the paired setting, the authors of [1] note that in some cases, CycleGAN performs significantly worse. As a possible improvement they consider adding weak or semi-supervision to the model. This could drastically improve the results on some dataset without requiring large amounts of paired data.

There is certainly some room for improvement in the architectures used for the generators, since the authors simply mimicked the network used in [3] for style transfer and super-resolution. The architecture of their discriminators is also quite primitive, and a more complex one could certainly yield better results.

References

[1] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2223–2232, 2017.

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[3] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[4] Hardik Bansal and Archit Rathore. `https://hardikbansal.github.io/CycleGANBlog/`.

[5] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[6] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6924–6932, 2017.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Training and investigating residual nets. `http://torch.ch/blog/2016/02/04/resnets.html`.

[9] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[10] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2107–2116, 2017.

[11] Harry Yang and Nathan Silberman. `https://github.com/leehomyc/cyclegan-1`.

[12] Taesung Park. `http://people.eecs.berkeley.edu/~taesung_park/CycleGAN/`.