A. Appdendix

A.1. Introduction of state-of-the-art models

CycleGAN uses an adversarial loss to learn the mapping between two different domains. The method regularizes the mapping through cycle-consistency losses, using two down-sampling convolution blocks, nine residual blocks, two up-sampling deconvolution blocks and four discriminator layers. Codes are on https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix.

UNIT consists of two VAE-GAN with shared latent space. The structure of the UNIT is similar to CycleGAN, but different from CycleGAN in that it uses multi-scale discriminators and shares the weight of the high-level layer stage of the encoder and decoder. Codes are on https://github.com/mingyuliutw/UNIT.

MUNIT can generate various outputs for a single input image. MUNIT assumes that the image representation can be decomposed into a content code and a style code. The main difference between MUNIT's network structure and other networks is that it uses AdaIN in the decoder and also a multi-scale discriminator. We generate N = 1 images for each input image in the test set. We use the generated samples and all samples in test set to compute FID and KID. Codes are on https://github.com/NVlabs/MUNIT.

DRIT can also create different outputs for a single input image similar to MUNIT. It decomposes the image into a content code and a style code, using a multi-scale discriminator. The difference between DRIT and MUNIT is that the content code is shared like UNIT. We generate N = 1 images for each input image in the test set. We use the generated samples and all samples in test set to compute FID and KID. Codes are on https://github.com/HsinYingLee/DRIT.

U-GAT-IT is a recent work associated with unsupervised image-to-image translation, which incorporates a CAM (Class Activation Map) module and an AdaLIN (Adaptive Layer-Instance Normalization) function in an end-to-end manner. U-GAT-IT can translate images requiring holistic changes or large shape changes. Light version is applied due to the limited memory of our gpu. Codes are on https://github.com/znxlwm/UGATIT-pytorch.

A.2. Network Architecture

The architectures of the discriminator and generator in NICE-GAN are shown in Table 1 and 2, respectively. For the generator network, we use adaptive layer-instance normalization in decoders except the last output layer. For the discriminator network, Leaky-ReLU is applied with a negative slope of 0.2 and spectral normalization is put in all layers. We apply *softmax* instead of *clip* to limit $\rho \in [0, 1]$ in AdaLIN. Besides, we concat global average & max pooling's feature maps before Classifier0 so that the input channel of MLP-(N1) is 256. More details are presented in our source code. There are some notations: N is the number of output channels; K is the kernel size; S is the side size; P is the padding size; AdaLIN is the adaptive layer-instance normalization; LIN is layer-instance normalization; SN is the spectral normalization; RA is adding residual connection in CAM attention module.

A.3. Additional results

A.3.1 Discussing γ

As for Residual Attention (RA) module, the parameter γ is task-specific (as illustrated in table 3). Regarding tasks like photo $\rightarrow$ vangogh and summer$\rightarrow$ winter, γ is close to 0 indicating more attention is paid to global features, which is reasonable as translating the whole content of the images in these tasks is more necessary than focusing on local details.

A.3.2 More analysis on the multi-scale discriminator.

Table 4 evaluates the impact of (C_x^0, C_x^1, C_x^2) on various datasets. For the cat $\leftrightarrow$ dog task, global characteristics of the semantic of objects is of much importance. For the colorization and stylization task(*e.g.* summer $\leftrightarrow$ winter, photo $\leftrightarrow$ vangogh), preserving middle and local scale still delivers promising performance. Specifically, if removing the local scale, FID increases significantly from 66 to 90 on *horse$\rightarrow$zebra*; and from 76/76 to 88/96 on *summer$\leftrightarrow$winter* if leaving out the medium scale. It implies all three scales are generally necessary.

A.3.3 More visualizations of hidden vectors.

The training process is proceeded in terms of three kinds of losses: adversarial loss, identity reconstruction loss, and cycle-consistency loss. The adversarial loss is to pursue domain transfer, while both reconstruction loss and cycle-consistency loss are for tackling the non-identifiability issue. As shown in Figure 1, our method enables meaningful hidden interpolations since the shared-latent space assumption are enforced by NICE framework and three kinds of losses in our training.

Figure 2 visualizes more heat-maps of the hidden vectors. Generally, the heat-maps by the model with NICE show more concise and distinguishable semantics encoding than that without NICE (namely an independent encoder is used). It shows using NICE captures the texture and local parts of the object more clearly, exhibiting the superiority of NICE-GAN.

A.3.4 Additional comparisons with state of the arts

Due to the lack of standard protocol so far, our experiments use released codes to train all baselines over the

Table 1: Discriminator network architecture

Component	Input → Output Shape	Layer Information
Encoder	$(h, w, 3) \rightarrow (\frac{h}{2}, \frac{w}{2}, 64)$	CONV-(N64, K4, S2, P1), SN, Leaky-ReLU
Down-sampling0	$(\frac{h}{2}, \frac{w}{2}, 64) \rightarrow (\frac{h}{4}, \frac{w}{4}, 128)$	CONV-(N128, K4, S2, P1), SN, Leaky-ReLU
RA of Encoder& Classifier0	$(\frac{h}{4}, \frac{w}{4}, 128) \rightarrow (\frac{h}{4}, \frac{w}{4}, 256)$	Global Average & Max Pooling, MLP-(N1), Multiply the weights of MLP
	$(\frac{h}{4}, \frac{w}{4}, 256) \rightarrow (\frac{h}{4}, \frac{w}{4}, 128)$	CONV-(N128, K1, S1), RA, Leaky-ReLU
Down-sampling1	$(\frac{h}{4}, \frac{w}{4}, 128) \rightarrow (\frac{h}{8}, \frac{w}{8}, 256)$	CONV-(N256, K4, S2, P1), SN, Leaky-ReLU
Classifier1	$(\frac{h}{8}, \frac{w}{8}, 256) \rightarrow (\frac{h}{8}-1, \frac{w}{8}-1, 512)$	CONV-(N512, K4, S1, P1), SN, Leaky-ReLU
	$(\frac{h}{8}-1, \frac{w}{8}-1, 512) \rightarrow (\frac{h}{8}-2, \frac{w}{8}-2, 1)$	CONV-(N1, K4, S1, P1), SN
Down-sampling2	$(\frac{h}{8}, \frac{w}{8}, 256) \rightarrow (\frac{h}{16}, \frac{w}{16}, 512)$	CONV-(N512, K4, S2, P1), SN, Leaky-ReLU
	$(\frac{h}{16}, \frac{w}{16}, 512) \rightarrow (\frac{h}{32}, \frac{w}{32}, 1024)$	CONV-(N1024, K4, S2, P1), SN, Leaky-ReLU
Classifier2	$(\frac{h}{32}, \frac{w}{32}, 1024) \rightarrow (\frac{h}{32}-1, \frac{w}{32}-1, 2048)$	CONV-(N2048, K4, S1, P1), SN, Leaky-ReLU
	$(\frac{h}{32}-1, \frac{w}{32}-1, 2048) \rightarrow (\frac{h}{32}-2, \frac{w}{32}-2, 1)$	CONV-(N1, K4, S1, P1), SN

Table 2: Generator network architecture

Component	Input → Output Shape	Layer Information
Sampling	$(\frac{h}{4}, \frac{w}{4}, 128) \rightarrow (\frac{h}{4}, \frac{w}{4}, 256)$	CONV-(N256, K3, S1, P1), LIN, ReLU
$\gamma_{AdaLIN}, \beta_{AdaLIN}$	$(\frac{h}{4}, \frac{w}{4}, 256) \rightarrow (1, 1, 256)$	Global Average Pooling
	$(1, 1, 256) \rightarrow (1, 1, 256)$	MLP-(N256), ReLU
	$(1, 1, 256) \rightarrow (1, 1, 256)$	MLP-(N256), ReLU
	$(1, 1, 256) \rightarrow (1, 1, 256)$	MLP-(N256), ReLU
Bottleneck	$(\frac{h}{4}, \frac{w}{4}, 256) \rightarrow (\frac{h}{4}, \frac{w}{4}, 256)$	AdaResBlock-(N256, K3, S1, P1), AdaLIN, ReLU
	$(\frac{h}{4}, \frac{w}{4}, 256) \rightarrow (\frac{h}{4}, \frac{w}{4}, 256)$	AdaResBlock-(N256, K3, S1, P1), AdaLIN, ReLU
	$(\frac{h}{4}, \frac{w}{4}, 256) \rightarrow (\frac{h}{4}, \frac{w}{4}, 256)$	AdaResBlock-(N256, K3, S1, P1), AdaLIN, ReLU
	$(\frac{h}{4}, \frac{w}{4}, 256) \rightarrow (\frac{h}{4}, \frac{w}{4}, 256)$	AdaResBlock-(N256, K3, S1, P1), AdaLIN, ReLU
	$(\frac{h}{4}, \frac{w}{4}, 256) \rightarrow (\frac{h}{4}, \frac{w}{4}, 256)$	AdaResBlock-(N256, K3, S1, P1), AdaLIN, ReLU
	$(\frac{h}{4}, \frac{w}{4}, 256) \rightarrow (\frac{h}{4}, \frac{w}{4}, 256)$	AdaResBlock-(N256, K3, S1, P1), AdaLIN, ReLU
Up-sampling	$(\frac{h}{4}, \frac{w}{4}, 256) \rightarrow (\frac{h}{2}, \frac{w}{2}, 128)$	Sub-pixel-CONV-(N128, K3, S1, P1), LIN, ReLU
	$(\frac{h}{2}, \frac{w}{2}, 128) \rightarrow (h, w, 64)$	Sub-pixel-CONV-(N64, K3, S1, P1), LIN, ReLU
	$(h, w, 64) \rightarrow (h, w, 3)$	CONV-(N3, K7, S1, P3), Tanh

same iterations for fair comparison. Table 5 shows additional comparisons with state of the arts in 200K-th iterations. Still, NICE-GAN (trained for more iterations) generally performs superiorly.

A.3.5 More visualizations of translated images.

In addition to the results presented in the paper, we show more generated images for the four datasets in Figure 3, 4, 5, 6, 7, 8, 9 and 10.

Table 3: **RA Analysis.** $a(x) = \gamma w E_x(x) + E_x(x)$, where the trainable parameter γ determines the trade-off between the attended features and the original ones. When $\gamma = 0$, it returns to $E_x(x)$ indicating no attention is used, and otherwise, the attention is activated.

Object	dog	winter	photo	zebra
γ	-0.2492	0.2588	-0.0006	-0.2699
Object	cat	summer	vangogh	horse
γ	0.3023	0.0006	0.3301	0.2723

Table 4: **Multi-Scale Analysis.** For both FID and KID, lower is better. Results of methods are all in 100K iterations of discriminator.

Method \ Dataset	dog2cat		winter2summer		photo2vangogh		zebra2horse	
	FID	KID × 100	FID	KID × 100	FID	KID × 100	FID	KID × 100
C_x^0	216.03	18.57	81.12	1.50	135.17	3.92	215.79	12.79
C_x^0, C_x^1	203.56	15.27	77.52	1.14	121.47	2.86	193.11	10.37
C_x^0, C_x^1, C_x^2	48.79	1.58	76.44	1.22	122.27	3.71	149.48	4.29
C_x^1, C_x^1	45.46	0.85	77.50	1.17	131.38	5.38	147.24	3.92
C_x^0, C_x^2	54.31	2.20	88.02	2.45	130.73	4.87	154.13	5.43
Method \ Dataset	cat2dog		summer2winter		vangogh2photo		horse2zebra	
	FID	KID × 100	FID	KID × 100	FID	KID × 100	FID	KID × 100
C_x^0	231.24	22.12	76.88	0.63	155.50	7.40	168.57	10.74
C_x^0, C_x^1	238.62	21.41	77.10	0.67	132.08	4.67	104.46	4.60
C_x^0, C_x^1, C_x^2	44.67	1.20	76.03	0.67	112.00	2.79	65.93	2.09
C_x^1, C_x^1	53.94	1.95	79.91	1.11	128.47	4.87	90.00	3.77
C_x^0, C_x^2	65.99	2.62	96.26	2.08	123.05	4.32	80.50	2.85

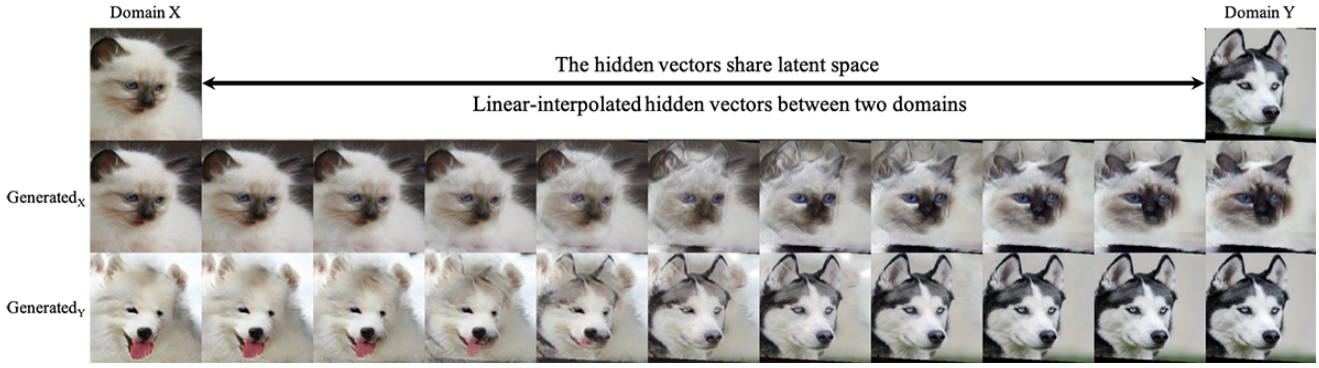

Figure 1: **Translation results with linear-interpolated hidden vectors between two domains.** $\mathrm{Generated_X}$: images of Domain X generated from the hidden vectors; $\mathrm{Generated_Y}$: images of Domain Y generated from the hidden vectors. Results show that the hidden vectors share latent space since it successfully generates reasonable image from linear-interpolated hidden vectors between two domains.

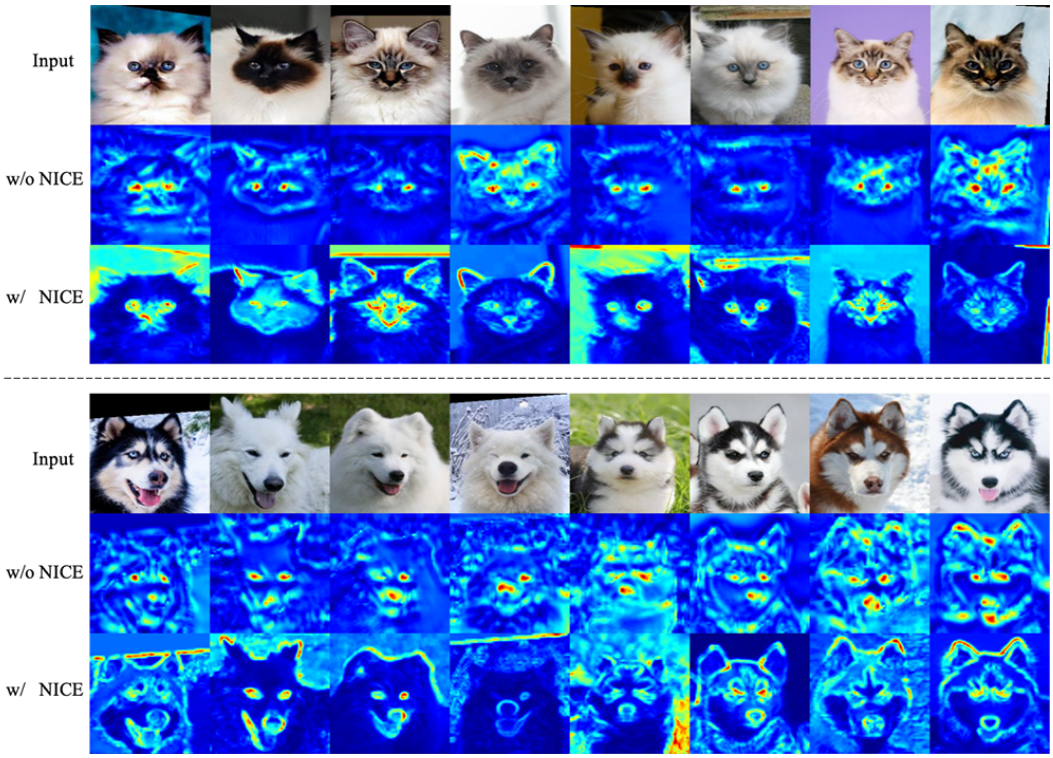

Figure 2: The heat-map visualizations of the hidden vectors.

Table 5: The FID and the KID $\times 100$ for different algorithms. Lower is better. All of the methods are trained to the 200K-th iterations.

Dataset / Method	dog → cat		winter → summer		photo → vangogh		zebra → horse	
	FID	KID × 100	FID	KID × 100	FID	KID × 100	FID	KID × 100
NICE-GAN	**42.22**	**0.73**	77.51	1.37	126.29	4.35	**138.77**	**3.26**
U-GAT-IT-light	63.85	2.08	**72.58**	1.99	120.92	3.68	150.34	3.64
CycleGAN	93.72	3.46	77.01	**1.07**	**115.74**	**2.90**	140.65	3.64
UNIT	53.18	1.36	95.76	4.59	135.37	5.03	174.65	6.36
MUNIT	48.52	1.21	99.14	4.36	132.22	4.75	190.06	6.32
DRIT	63.13	2.75	83.30	2.03	126.11	4.28	164.92	6.78
Dataset / Method	cat → dog		summer → winter		vangogh → photo		horse → zebra	
	FID	KID × 100	FID	KID × 100	FID	KID × 100	FID	KID × 100
NICE-GAN	**34.71**	**0.61**	78.87	**0.78**	**107.53**	**2.99**	75.64	1.77
U-GAT-IT-light	69.43	2.48	84.16	1.16	110.03	3.54	85.66	2.78
CycleGAN	103.95	5.41	**78.39**	0.82	117.88	3.08	**68.11**	**1.52**
UNIT	42.32	0.90	111.14	5.34	125.85	5.97	118.98	6.34
MUNIT	45.17	1.14	110.91	4.90	131.25	6.01	104.72	5.26
DRIT	53.19	1.73	81.64	1.10	111.46	3.76	92.26	4.58

Figure 3: **Examples of cat → dog translation images.** As is shown in these examples, images generated by NICE-GAN, UNIT and MUNIT have better quality.

Figure 4: **Examples of dog → cat translation images.** Most images are optimistic except those generated by CycleGAN and DRIT.

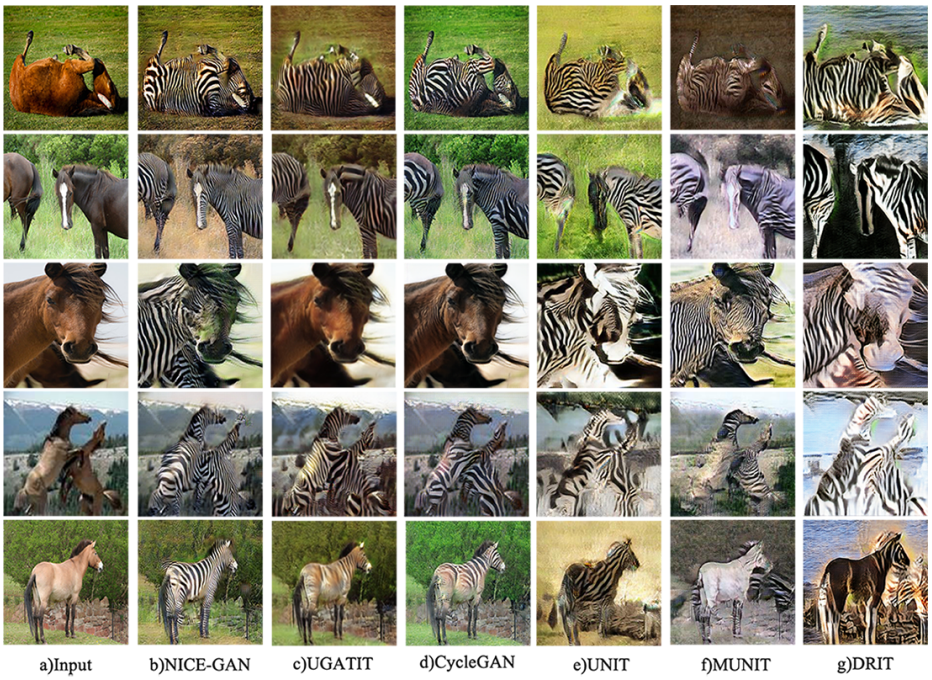

Figure 5: **Examples of horse → zebra translation images.** The translation images shows that NICE-GAN has better ability in adding textures except for subtle color differences during the translation process.

Figure 6: **Examples of zebra → horse translation images.** As is shown in the examples, images generated by U-GAT-IT gain the best results. The disadvantage of NICE-GAN still lies in subtle color differences.

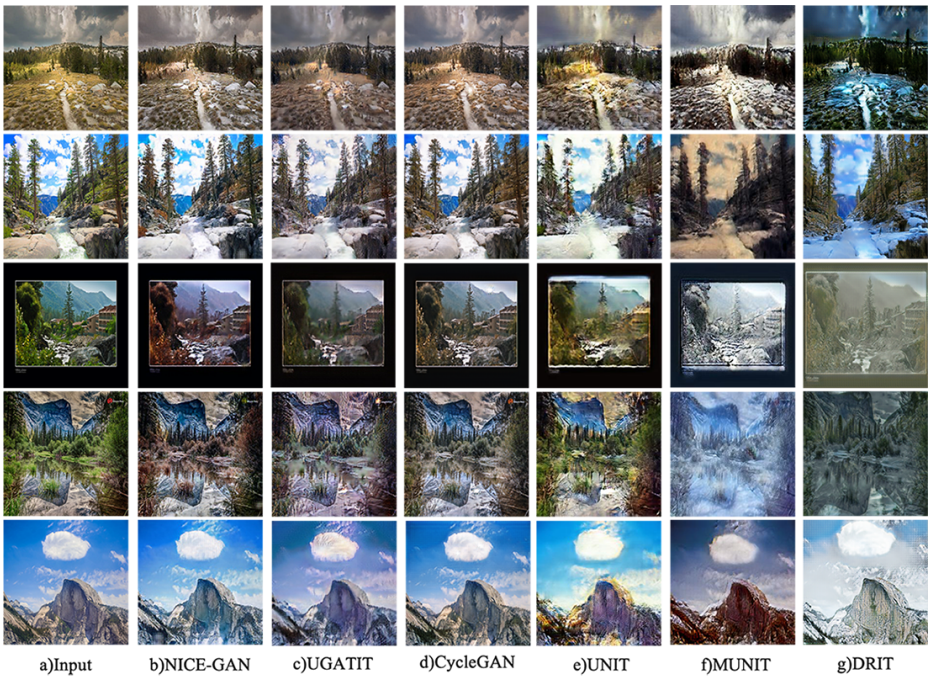

Figure 7: **Examples of summer → winter translation images.** Images generated by different methods gain relatively ideal and realistic results.

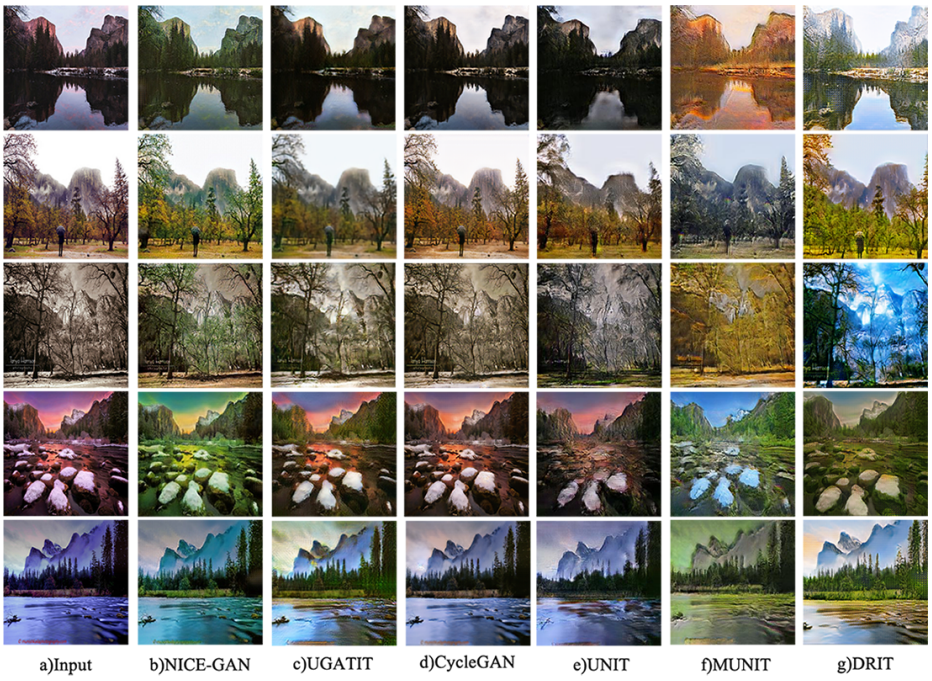

Figure 8: **Examples of winter→ summer translation images.** Images generated by different methods look optimistic except for images generated by CycleGAN and UNIT.

Figure 9: **Examples of vangogh → photo translation images.** The translation of vangogh → photo is a difficult task, most methods could barely finish the task.

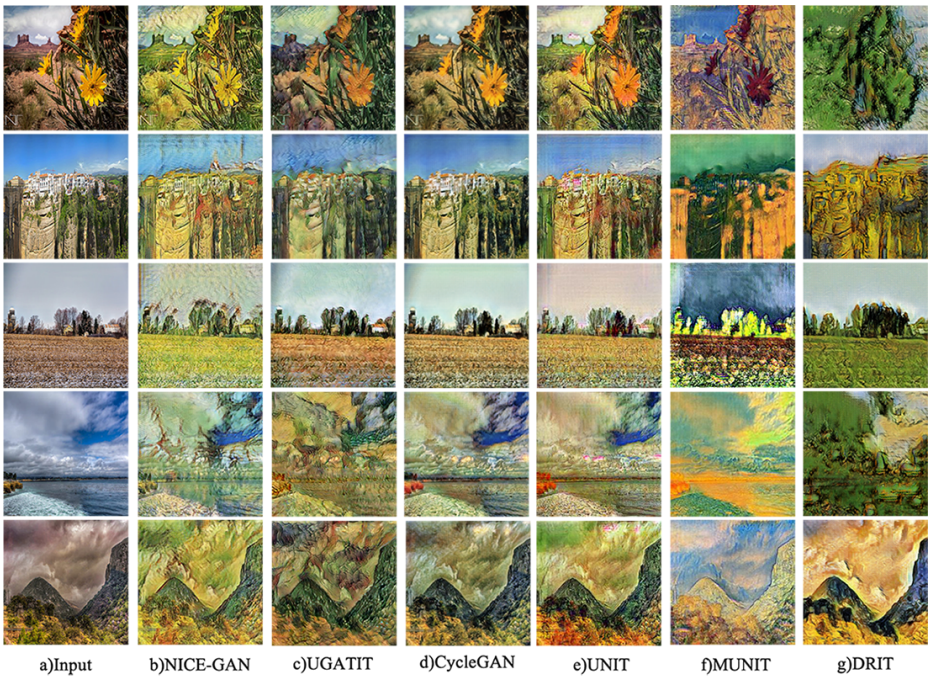

Figure 10: **Examples of photo → vangogh translation images.** Images generated by different methods gain relatively ideal results except for DRIT.