computation

MDPI

Article

Invertible Autoencoder for Domain Adaptation

Yunfei Teng * and Anna Choromanska *

Department of Electrical and Computer Engineering, NYU Tandon School of Engineering, 5 MetroTech Center, Brooklyn, NY 11201, USA

* Correspondence: yt1208@nyu.edu (Y.T.); ac5455@nyu.edu (A.C.)

Received: 17 December 2018; Accepted: 21 March 2019; Published: 27 March 2019

Abstract: The unsupervised image-to-image translation aims at finding a mapping between the source ($\mathcal{A}$) and target ($\mathcal{B}$) image domains, where in many applications aligned image pairs are not available at training. This is an ill-posed learning problem since it requires inferring the joint probability distribution from marginals. Joint learning of coupled mappings $\mathcal{F}_{\mathcal{AB}} : \mathcal{A} \rightarrow \mathcal{B}$ and $\mathcal{F}_{\mathcal{BA}} : \mathcal{B} \rightarrow \mathcal{A}$ is commonly used by the state-of-the-art methods, like CycleGAN to learn this translation by introducing cycle consistency requirement to the learning problem, i.e., $\mathcal{F}_{\mathcal{AB}}(\mathcal{F}_{\mathcal{BA}}(\mathcal{B})) \approx \mathcal{B}$ and $\mathcal{F}_{\mathcal{BA}}(\mathcal{F}_{\mathcal{AB}}(\mathcal{A})) \approx \mathcal{A}$. Cycle consistency enforces the preservation of the mutual information between input and translated images. However, it does not explicitly enforce $\mathcal{F}_{\mathcal{BA}}$ to be an inverse operation to $\mathcal{F}_{\mathcal{AB}}$. We propose a new deep architecture that we call *invertible autoencoder (InvAuto)* to explicitly enforce this relation. This is done by forcing an encoder to be an inverted version of the decoder, where corresponding layers perform opposite mappings and share parameters. The mappings are constrained to be orthonormal. The resulting architecture leads to the reduction of the number of trainable parameters (up to 2 times). We present image translation results on benchmark datasets and demonstrate state-of-the art performance of our approach. Finally, we test the proposed domain adaptation method on the task of road video conversion. We demonstrate that the videos converted with InvAuto have high quality and show that the NVIDIA neural-network-based end-to-end learning system for autonomous driving, known as PilotNet, trained on real road videos performs well when tested on the converted ones.

Keywords: image-to-image translation; autoencoder; invertible autoencoder

1. Introduction

Inter-domain translation problem of converting an instance, e.g., image or video, from one domain to another is applicable to a wide variety of learning tasks, including object detection and recognition, image categorization, sentiment analysis, action recognition, speech recognition, and more. High-quality domain translators ensure that an arbitrary learning model trained on the samples from the source domain, can perform well when tested on the translated samples (Similarly, an arbitrary learning model trained on the translated samples should perform well on the samples from the target domain. Training in this framework is however, much more computationally expensive). The translation problem can be posed in the supervised learning framework, e.g., [1,2], where the learner has access to corresponding pairs of instances from both domains, or unsupervised learning framework, e.g., [3,4], where no such paired instances are available. This paper focuses on the latter case, which is more difficult but at the same time more realistic as acquiring the dataset of paired images is often impossible in practice.

The unsupervised domain adaptation is typically solved using generative adversarial networks (GAN) framework [5]. GANs constitute a family of methods that learn generative models from complicated real-world data. In order to teach the generator to synthesize semantically meaningful

data from standard signal distributions, GANs train a discriminator to distinguish real samples in the training dataset from fake samples synthesized by the generator. The generator aims to deceive the discriminator by producing increasingly more realistic samples. Thus, the generator and discriminator play an adversarial game, during which the generator learns to produce samples from the desired data distribution and the discriminator eventually cannot make a better decision than randomly guessing whether a particular sample is fake or real. GANs have recently been successfully applied to image generation [6–9], image editing [1,3,10,11], video prediction [12–14], and many other tasks [15–17]. In the domain adaptation setting, the generator performs domain translation and is trained to learn the mapping from the source to the target domain and the discriminator is trained to discriminate between original images from the target domain and those provided by the generator. In this setting, the generator usually has the structure of the autoencoder. The two most common state-of-the-art domain adaptation approaches, CycleGAN [3] and UNIT [4], are built on this basic approach. CycleGAN addresses the problem of adaptation from domain $\mathcal{A}$ to domain $\mathcal{B}$ by training two translation networks, where one realizes the mapping $\mathcal{F}_{\mathcal{AB}}$ and the other realizes $\mathcal{F}_{\mathcal{BA}}$. The cycle consistency loss ensures the correlation between input image and the corresponding translation. In particular, to achieve cycle consistency, CycleGAN trains two autoencoders, where each minimizes its own adversarial loss and they both jointly minimize

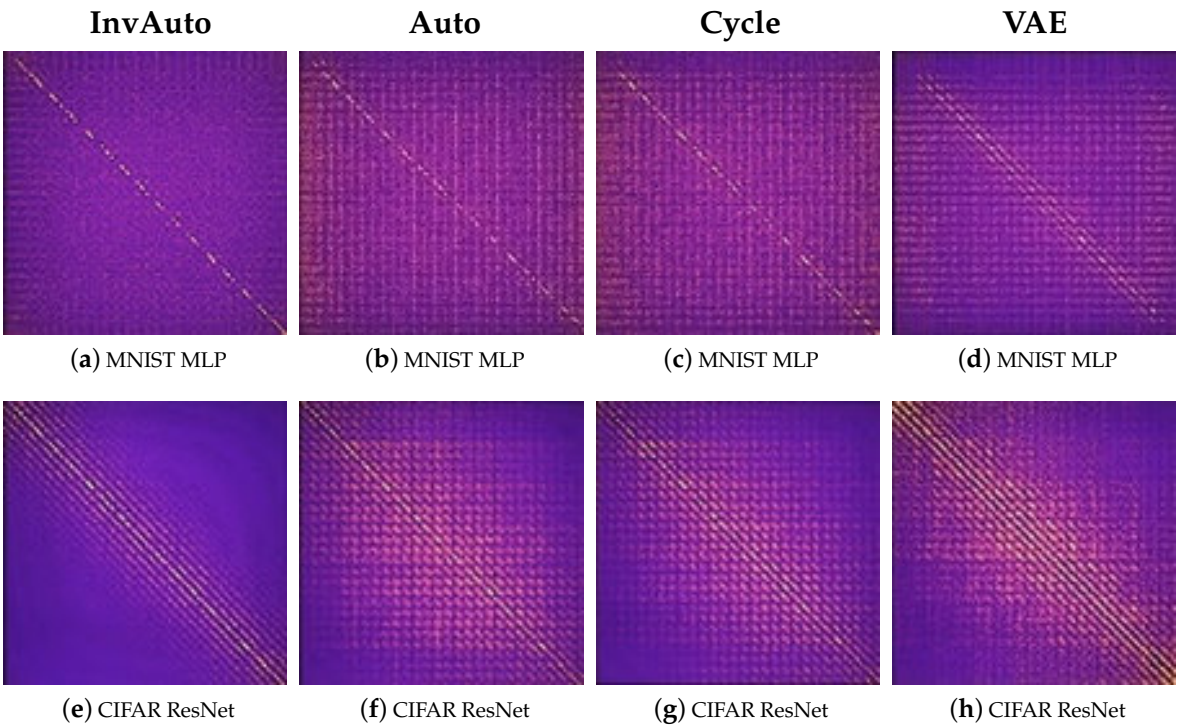
$$\|\mathcal{F}_{\mathcal{AB}}(\mathcal{F}_{\mathcal{BA}}(\mathcal{B})) - \mathcal{B}\|_2^2 \text{ and } \|\mathcal{F}_{\mathcal{BA}}(\mathcal{F}_{\mathcal{AB}}(\mathcal{A})) - \mathcal{A}\|_2^2. \tag{1}$$

Cycle consistency loss is also incorporated into the recent implementations of UNIT. It is implicitly assumed that the model will learn the mappings $\mathcal{F}_{\mathcal{AB}}$ and $\mathcal{F}_{\mathcal{BA}}$ in such a way that $\mathcal{F}_{\mathcal{AB}} = \mathcal{F}_{\mathcal{BA}}^{-1}$, however, it is not explicitly imposed. Consider a simple example. Assume the first autonecoder is a two-layer linear multi-layer perceptron (MLP) where the weight matrix of the first layer (encoder) is denoted as E_1 and the weight matrix of the second layer (decoder) is denoted as D_1. Thus, for an input $x_{\mathcal{A}} \in \mathcal{A}$ it outputs $y_{\mathcal{B}}(x_{\mathcal{A}}) = D_1 E_1 x_{\mathcal{A}}$. The second autoencoder then is a two-layer MLP with encoder weight matrix E_2 and decoder weight matrix D_2 that for an input data point $x_{\mathcal{B}}$ should produce output $y_{\mathcal{A}}(x_{\mathcal{B}}) = D_2 E_2 x_{\mathcal{B}}$. To satisfy cycle consistency requirement, the following should hold: $y_{\mathcal{A}}(y_{\mathcal{B}}(x_{\mathcal{A}})) = (x_{\mathcal{A}})$ and $y_{\mathcal{B}}(y_{\mathcal{A}}(x_{\mathcal{B}})) = (x_{\mathcal{B}})$. These two conditions are equivalent to $D_2 E_2 D_1 E_1 = I$ and $D_1 E_1 D_2 E_2 = I$. This holds for example when $D_1 = E_2^{-1}$ and $D_2 = E_1^{-1}$.

In contrast to this approach, we implicitly require $\mathcal{F}_{\mathcal{AB}} = \mathcal{F}_{\mathcal{BA}}^{-1}$. Thus, in the context of the given simple example, we correlate encoders and decoders to satisfy inversion conditions $D_1 = E_2^{-1}$ and $D_2 = E_1^{-1}$. We avoid performing prohibitive inversions of large matrices and instead guarantee these conditions to hold through two steps: (i) introducing shared parametrization of encoder E_2 and decoder D_1 such that $D_1 = E_2^\top$ (E_1 and D_2 is treated similarly) and (ii) appropriate training to achieve orthonormality $E_2^\top = E_2^{-1}$ and $E_1^\top = E_1^{-1}$, i.e., we train autoencoder (E_2, D_1) to satisfy $D_1 E_2 x_{\mathcal{B}} = x_{\mathcal{B}}$ for arbitrary input $x_{\mathcal{B}}$ and autoencoder (E_1, D_2) to satisfy $D_2 E_1 x_{\mathcal{A}} = x_{\mathcal{A}}$ for arbitrary input $x_{\mathcal{A}}$. Since the encoder and decoder are coupled as given in (i), such training leads to satisfying inversion conditions. Practical networks contain linear and non-linear transformations. We therefore propose specific architectures, which are invertible.

Figure 1 (see also its extended version, Figure A4, in the Appendix A) and Figure 2 illustrate the basic idea behind InvAuto. The plots were obtained by training a single autoencoder (E, D) to reconstruct its input. InvAuto has shared weights satisfying $D = E^\top$ and inverted non-linearities and clearly obtains matrix DE that is the closest to identity compared to other methods, i.e., vanilla autoencoder (Auto), autoencoder with cycle consistency (Cycle), and variational autoencoder (VAE) [18]. Note also that, at the same time, InvAuto requires half of the number of trainable parameters. This is because the encoder and decoder use the same parameters.

This paper is organized as follows: Section 2 reviews the literature, Section 3 explains InvAuto in details, Section 4 explains how to apply InvAuto to domain adaptation, Section 5 demonstrates experimental verification of the proposed approach, and Section 6 provides conclusions.



(**a**) MNIST MLP (**b**) MNIST MLP (**c**) MNIST MLP (**d**) MNIST MLP

(**e**) CIFAR ResNet (**f**) CIFAR ResNet (**g**) CIFAR ResNet (**h**) CIFAR ResNet

Figure 1. Heatmap of the values of matrix DE for InvAuto (**a**,**e**), Auto (**b**,**f**), Cycle (**c**,**g**), and VAE (**d**,**h**) on MLP and ResNet architectures and MNIST and CIFAR datasets. Matrices E and D are constructed by multiplying the weight matrices of consecutive layers of multi-layer encoder and decoder, respectively, e.g., $E = E_L \dots E_2 E_1$ and $D = D_L \dots D_2 D_1$ for a $2L$-layer autoencoder. In case of InvAuto, DE is the closest to the identity matrix.

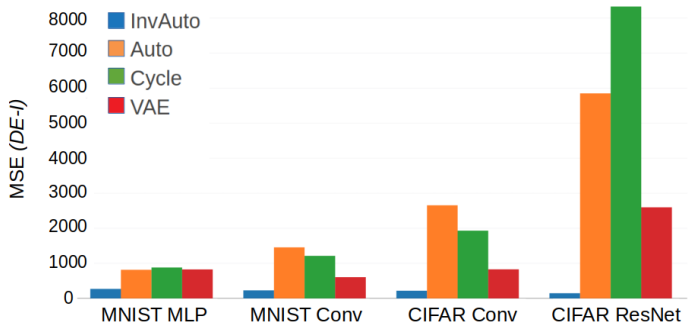


Figure 2. Comparison of the mean squared error (MSE) MSE$(DE - I)$ for InvAuto, Auto, Cycle, and VAE on MLP, convolutional, and ResNet architectures and MNIST and CIFAR datasets. Matrices E and D are constructed by multiplying the weight matrices of consecutive layers of encoder and decoder, respectively.

2. Related Work

Unsupervised image-to-image translation models were developed to tackle domain adaptation problem with unpaired datasets. A plethora of existing approaches utilize autoencoders trained in the GAN framework, where autoencoder serves as a generator, for this learning problem. This includes approaches based on conditional GAN [2,19] and methods introducing additional components to the loss function forcing partial cycle consistency [20]. Another approach [21] introduces two coupled GANs, where each generator is an autoencoder and the coupling is obtained by sharing a subset of weights between autoencoders as well as between discriminators. This technique was later extended to utilize variational autoencoders as generators [4]. The resulting approach is commonly known as UNIT. CycleGAN presents yet another way of addressing the image-to-image translation by specific

training scheme that preserves the mutual information between input and translated images [22]. Both UNIT and CycleGAN constitute the most popular choices for performing image-to-image translation.

There also exist other learning tasks that can be viewed as instances of image-to-image translation problem. Among them, notable approaches focus on style transfer [23–26]. They aim at preserving the content of the input image while altering its style to mimic the style of the images from the target domain. This goal is achieved by introducing content and style loss functions that are jointly optimized. Finally, inverse problems, such as super-resolution, also fall into the category of image-to-image translation problems [27].

3. Invertible Autoencoder

Here we explain the details of the architecture of InvAuto. The architecture needs to be symmetric to allow invertibility, e.g., the layers should be arranged as $(\underbrace{T_1, T_2, \ldots, T_M}_{\text{encoder } E}, \underbrace{T_M^{-1}, T_{M-1}^{-1}, \ldots, T_1^{-1}}_{\text{decoder } D})$, where $T_1, T_2, \ldots, T_M$ denote subsequent transformations of the signal that is being propagated through the network (M is the total number of those) and $T_1^{-1}, T_2^{-1}, \ldots, T_M^{-1}$ denote their inversions. Thus, the architecture is inverted layer by layer, where any layer of the encoder has its mirror inverted counterpart in the decoder. The autoencoder is trained to reconstruct its input. Below we explain how to invert different types of layers of the deep model.

3.1. Fully Connected Layer

Consider transformation T^E of an input signal performed by an arbitrary fully connected layer of an encoder E parametrized with weight matrix W. Let x denote layer's input and y denote its output. Thus,

$$T^E : y = Wx. \tag{2}$$

An inverse operation is then defined as

$$(T^E)^{-1} : x = W^{-1}y, \tag{3}$$

We parametrize the counterpart layer of the decoder with a transpose of W, Thus, the considered encoder and decoder layers will share parametrization. Therefore, we enforce the counterpart decoder's layer to perform transformation:

$$T^D : x = W^\top y. \tag{4}$$

By training the autoencoder to reconstruct its input on its output we will enforce orthonormality $W^{-1} = W^\top$ and Thus, equivalence of transformations $(T^E)^{-1}$ and T^D, i.e., $(T^E)^{-1} \equiv T^D$.

3.2. Convolutional Layer

Consider transformation T^E of an input image performed by an arbitrary convolutional layer of an encoder E. Let x denote this layer's vectorized input image and y denote corresponding output. 2D convolution can be implemented using matrix multiplication involving a Toeplitz matrix [28]. Toeplitz matrix is obtained from the set of kernels of the 2D convolutional filters . Thus, transformation T^E and its inverse $(T^E)^{-1}$ can be explained with the same equations as the ones used before, Equations (2) and (3), however, now W is a Toeplitz matrix. We will again parametrize the counterpart layer of the decoder with a transpose of a Toeplitz matrix W. The transpose of the Toeplitz matrix is in practice obtained by copying weights from the considered convolutional layer to the counterpart decoder's layer that is implemented as a transposed convolutional layer (also known as a deconvolutional layer). Therefore, as before, we enforce the counterpart decoder's layer to perform transformation $T^D : x = W^\top y$ and by appropriate training ensure $(T^E)^{-1} \equiv T^D$.

3.3. Activation Function

Invertible activation function should be a bijection. In this paper, we consider a modified LeakyReLU activation function σ and use only this non-linearity in the model. Consider transformation T^E of an input signal performed by this non-linearity applied in the encoder E. This non-linearity is defined as

$$T^E : y = \sigma(x) = \begin{cases} \frac{1}{\alpha}x, & if\ x \geq 0 \\ \alpha x, & otherwise. \end{cases} \tag{5}$$

An inverse operation is then defined as

$$(T^E)^{-1} : x = \sigma^{-1}(y) = \begin{cases} \alpha y, & if\ x \geq 0 \\ \frac{1}{\alpha}y, & otherwise. \end{cases} \tag{6}$$

The corresponding non-linearity in the decoder will therefore realize the operation of an inverted modified LeakyReLU given in Equation (6). In the experiments we set $\alpha = 2$.

3.4. Residual Block

Consider transformation T^E of an input signal performed by a residual block [29] of an encoder E. We modify the residual block to remove the internal non-linearity as given in Figure 3a. The residual block is parametrized with weight matrices W_1 and W_2. Those are Toeplitz matrices corresponding to the convolutional and transposed convolutional layers of the residual block. Let x denote this block's vectorized input and y denote its corresponding output. Thus, transformation T^E is defined as

$$T^E : y = \sigma((W_2 \cdot W_1 + I) \cdot x) \tag{7}$$

An inverse operation is then defined as

$$(T^E)^{-1} : x = (W_2 \cdot W_1 + I)^{-1}\sigma^{-1}(y). \tag{8}$$

We will parametrize the counterpart residual block of the decoder with a transpose of matrix $W_2 \cdot W_1 + I$ as given in Figure 3b. Therefore we enforce the counterpart decoder's residual block to perform transformation:

$$T^D : x = (W_1^\top W_2^\top + I)y. \tag{9}$$

As before, at training will enforce orthonormality $(W_2 \cdot W_1 + I)^{-1} = (W_2 \cdot W_1 + I)^\top$ and Thus, $(T^E)^{-1} \equiv T^D$.

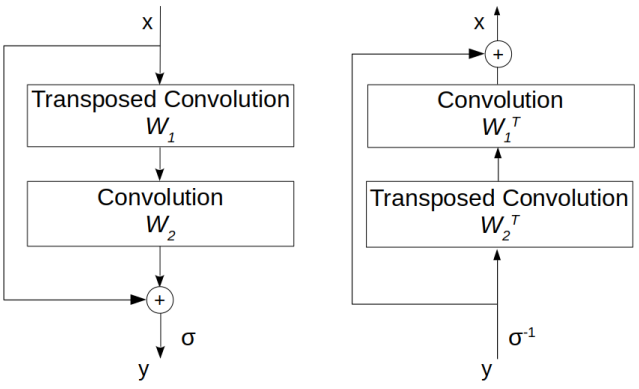


Figure 3. (**a**) Residual block. (**b**) Inverted residual block.

3.5. Bias

We consider bias as a separate layer in the network. Then, handling biases is straightforward. In particular, the layer in the encoder that performs bias addition has its counterpart layer in the decoder, where the same bias is subtracted.

3.6. Experimental Validation of Orthonormality

In this section, we validate the concept of InvAuto. The goal of this section is to show that proposed shared parametrization and training enforce orthonormality and that at the same time the orthonormality property is not organically achieved by standard architectures. We compare InvAuto with previously mentioned vanilla autoencoder, autoencoder with cycle consistency, and variational autoencoder. We experimented with various datasets (MNIST and CIFAR-10) and architectures (MLP, convolutional (Conv), and ResNet). All the networks were designed to have two down-sampling layers and two up-sampling layers. Encoder's matrix E and decoder's matrix D are constructed by multiplying the weight matrices of consecutive layers of encoder and decoder, respectively.

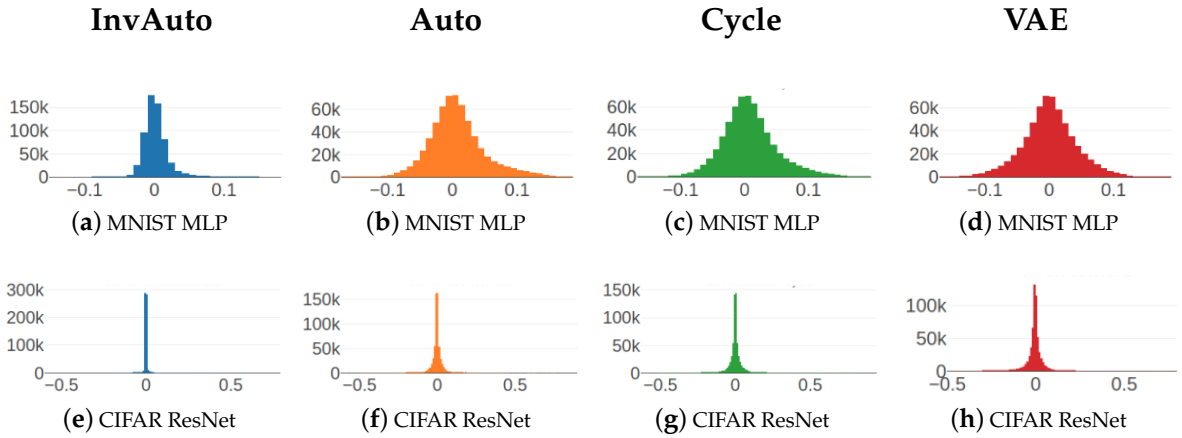
We test orthonormality by reporting the histograms of the cosine similarity of each pair of rows of matrix E for all methods (Figure 4) along with their mean and standard deviation (Table 1) as we expect the cosine similarity to be close to 0 for InvAuto. We then show the ℓ_2-norm of the rows of E as we expect the rows of InvAuto to have close-to-unit norm (Table 2). InvAuto enforces the encoder, and consequently the decoder, to be orthonormal. Other methods do not explicitly demand that and Thus, the orthonormality of their encoders is weaker. This observation is further confirmed by Figures 1 and 2 shown before in the Introduction. In the Appendix A, we provide three more figures that complement Figure 2 (recall that the latter reports the MSE of $DE - I$). They show the MSE of the diagonal (Figure A1) and off-diagonal of $DE - I$ (Figure A2) as well as the ratio of the MSE of the off-diagonal and diagonal of DE (Figure A3) for various methods. The reconstruction loss obtained for all methods is also shown in Appendix A (Table A1).

Table 1. Mean and standard deviation of cosine similarity of rows of E. InvAuto achieves cosine similarity that is the closest to 0. Best performer is in bold.

Dataset and Model	InvAuto	Auto	Cycle	VAE
MNIST MLP	**0.001 ±0.118**	0.008 ±0.210	0.007 ±0.207	0.001 ±0.219
MNIST Conv	**0.001 ±0.148**	0.001 ±0.179	0.001 ±0.176	−0.001 ±0.190
CIFAR Conv	**0.001 ±0.145**	0.002 ±0.176	0.004 ±0.195	0.003 ±0.268
CIFAR ResNet	**0.000 ±0.134**	0.000 ±0.203	0.000 ±0.232	0.001 ±0.298

Table 2. Mean and standard deviation of the ℓ_2-norm of the rows of E. InvAuto achieves the ℓ_2-norm of the rows that is the closest to the unit norm. Best performer is in bold.

Dataset and Model	InvAuto	Auto	Cycle	VAE
MNIST MLP	**0.976 ±0.190**	1.326 ±0.095	1.268 ±0.095	1.832 ±0.501
MNIST Conv	**0.905 ±0.321**	1.699 ±0.732	1.780 ±0.779	1.971 ±0.794
CIFAR Conv	**0.908 ±0.219**	3.027 ±0.816	2.463 ±0.688	1.176 ±0.356
CIFAR ResNet	**0.868 ±0.078**	2.890 ±0.895	2.650 ±0.937	1.728 ±0.311



(**a**) MNIST MLP (**b**) MNIST MLP (**c**) MNIST MLP (**d**) MNIST MLP

(**e**) CIFAR ResNet (**f**) CIFAR ResNet (**g**) CIFAR ResNet (**h**) CIFAR ResNet

Figure 4. The histograms of cosine similarity of the rows of E for InvAuto (**a**,**e**), Auto (**b**,**f**), Cycle (**c**,**g**), and VAE (**d**,**h**) on MLP and ResNet architectures and MNIST and CIFAR datasets.

Next we describe how InvAuto is applied to the problem of domain adaptation.

4. Invertible Autoencoder for Domain Adaptation

For the purpose of performing domain adaptation, we construct the dedicated architecture that is similar to CycleGAN, but we use InvAuto at the feature level of the generators. This InvAuto contains encoder E and decoder D that themselves have the form of autoencoders. Each of these internal autoencoders is used to do the conversion between the features corresponding to two different domains, and thus, the encoder E performs the conversion from the features corresponding to domain $\mathcal{A}$ into the features corresponding to domain $\mathcal{B}$. The decoder D, on the other hand, performs the conversion from the features corresponding to domain $\mathcal{B}$ into the features corresponding to domain $\mathcal{A}$. Since E and D form InvAuto, E realizes an inversion of D (and vice versa) and shares parameters with D. This introduces strong correlations between two generators and reduces the number of trainable parameters, which distinguishes our approach from CycleGAN. The proposed architecture is illustrated in Figure 5. The details of the architecture and training are provided in Appendix A.

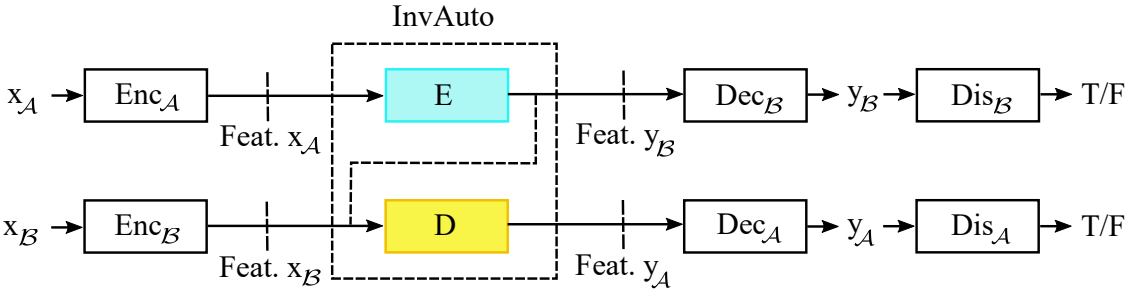


Figure 5. The architecture of the domain translator with InvAuto (E, D). $x_{\mathcal{A}} \in \mathcal{A}$ and $x_{\mathcal{B}} \in \mathcal{B}$ are the inputs of the translator. $y_{\mathcal{B}}$ is a converted image $x_{\mathcal{A}}$ into the $\mathcal{B}$ domain and $y_{\mathcal{A}}$ is a converted image $x_{\mathcal{B}}$ into the $\mathcal{A}$ domain. Invertible autoencoder (E, D) is built of encoder E and decoder D, where each of those itself is an autoencoder. $\text{Enc}_{\mathcal{A}}, \text{Enc}_{\mathcal{B}}$ are feature extractors, and $\text{Dec}_{\mathcal{A}}, \text{Dec}_{\mathcal{B}}$ are the final layers of the generators $\text{Gen}_{\mathcal{B}}$, i.e., $(\text{Enc}_{\mathcal{A}}, E, \text{Dec}_{\mathcal{B}})$, and $\text{Gen}_{\mathcal{A}}$, i.e., $(\text{Enc}_{\mathcal{B}}, D, \text{Dec}_{\mathcal{A}})$, respectively. Discriminators $\text{Dis}_{\mathcal{A}}$ and $\text{Dis}_{\mathcal{B}}$ discriminate whether their input comes from the generator (True) or original dataset (False).

Next we describe the cost function that we use to train our deep model. The first component of the cost function is the adversarial loss [5], i.e.,

$$\begin{aligned} L_{\text{adv}}(\text{Gen}_{\mathcal{A}}, \text{Dis}_{\mathcal{A}}) =& \mathbb{E}_{x_{\mathcal{A}} \sim p_d(\mathcal{A})}[\log \text{Dis}_{\mathcal{A}}(x_{\mathcal{A}})] + \\ & \mathbb{E}_{x_{\mathcal{B}} \sim p_d(\mathcal{B})}[\log(1 - \text{Dis}_{\mathcal{A}}(\text{Gen}_{\mathcal{A}}(x_{\mathcal{B}})))] \\ L_{\text{adv}}(\text{Gen}_{\mathcal{B}}, \text{Dis}_{\mathcal{B}}) =& \mathbb{E}_{x_{\mathcal{B}} \sim p_d(\mathcal{B})}[\log \text{Dis}_{\mathcal{B}}(x_{\mathcal{B}})] + \\ & \mathbb{E}_{x_{\mathcal{A}} \sim p_d(\mathcal{A})}[\log(1 - \text{Dis}_{\mathcal{B}}(\text{Gen}_{\mathcal{B}}(x_{\mathcal{A}})))], \end{aligned} \tag{10}$$

where $p_d(\mathcal{A})$ and $p_d(\mathcal{B})$ denote the distribution of data from $\mathcal{A}$ and $\mathcal{B}$, respectively.

The second component of the loss function is the cycle consistency loss defined as

$$\begin{aligned} L_{cc}(\text{Gen}_{\mathcal{A}}, \text{Gen}_{\mathcal{B}}) = & \mathbb{E}_{x_{\mathcal{A}} \sim p_d(\mathcal{A})}[\|x_A - \text{Gen}_{\mathcal{A}}(\text{Gen}_{\mathcal{B}}(x_A))\|_1] \\ & + \mathbb{E}_{x_{\mathcal{B}} \sim p_d(\mathcal{B})}[\|x_B - \text{Gen}_{\mathcal{B}}(\text{Gen}_{\mathcal{A}}(x_B))\|_1]. \end{aligned} \tag{11}$$

The objective function that we minimize therefore becomes

$$\begin{aligned} L(\text{Gen}_{\mathcal{A}}, \text{Gen}_{\mathcal{B}}, \text{Dis}_{\mathcal{A}}, \text{Dis}_{\mathcal{B}}) = & \lambda L_{cc}(\text{Gen}_{\mathcal{A}}, \text{Gen}_{\mathcal{B}}) \\ & + L_{\text{adv}}(\text{Gen}_{\mathcal{A}}, \text{Dis}_{\mathcal{A}}) \\ & + L_{\text{adv}}(\text{Gen}_{\mathcal{B}}, \text{Dis}_{\mathcal{B}}), \end{aligned} \tag{12}$$

where λ controls the balance between the adversarial loss and cycle consistency loss. The cycle consistency loss enforces the orthonormality property of InvAuto.

5. Experiments

We next demonstrate experiments on domain adaptation problems. We compare our model against UNIT [4] and CycleGAN [3]. We used publicly available implementations of both methods available from https://github.com/mingyuliutw/UNIT/ and https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/. The details of our architecture and the training process are summarized in Appendix A.

5.1. Experiments with Benchmark Datasets

We considered the following domain adaptation tasks:

(i) Day-to-night and night-to-day image conversion: we used unpaired road pictures recorded during the day and at night obtained from KAIST dataset [30].
(ii) Day-to-thermal and thermal-to-day image conversion: we used road pictures recorded during the day with a regular camera and a thermal camera obtained from KAIST dataset [30].
(iii) Maps-to-satellite and satellite-to-maps: we used satellite images and maps obtained from Google Maps [1].

The datasets for the last two tasks, i.e., (ii) and (iii), are originally paired, however, we randomly permuted them and train the model in an unsupervised fashion. The training and testing images were furthermore resized to 128×128 resolution.

The visual results of image conversion are presented in Figures 6–8 (Appendix A contains the same figures in higher resolution). We see that InvAuto visually performs comparably to other state-of-the-art methods.

To evaluate the performance of the methods numerically we use the following approach:

- For the tasks (ii) and (iii), we directly calculated the ℓ_1 loss between the converted images and the ground truth.
- For the task (i), we trained two autoencoders $\Omega_{\mathcal{A}}$ and $\Omega_{\mathcal{B}}$ on both domains, i.e., we trained each of them to perform high-quality reconstruction of the images from its own domain and low-quality

reconstruction of the images from the other domain. Then we use these two autoencoders to evaluate the quality of the converted images, where high ℓ_1 reconstruction loss of the autoencoder for the images converted to resemble those from its corresponding domain implies low-quality image translation.

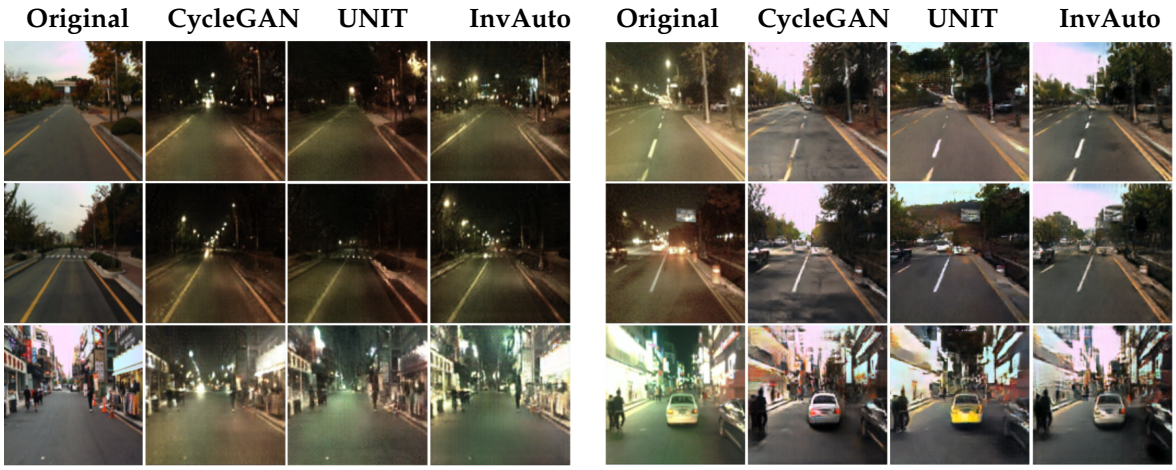


Figure 6. (**Left**) Day-to-night image conversion. Zoomed image is shown in Figure A5 in Appendix A. (**Right**) Night-to-day image conversion. Zoomed image is shown in Figure A6 in Appendix A.

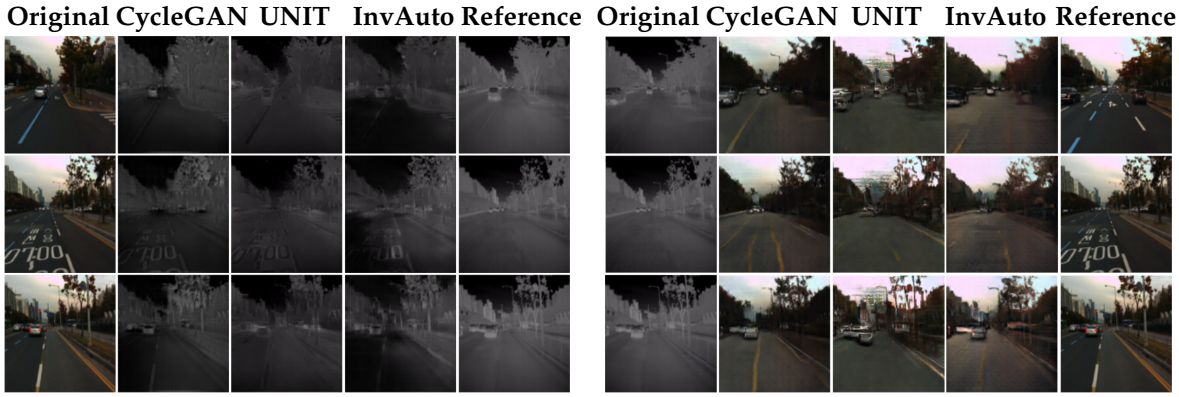


Figure 7. (**Left**) Day-to-thermal image conversion. Zoomed image is shown in Figure A7 in Appendix A. (**Right**) Thermal-to-day image conversion. Zoomed image is shown in Figure A8 in Appendix A.

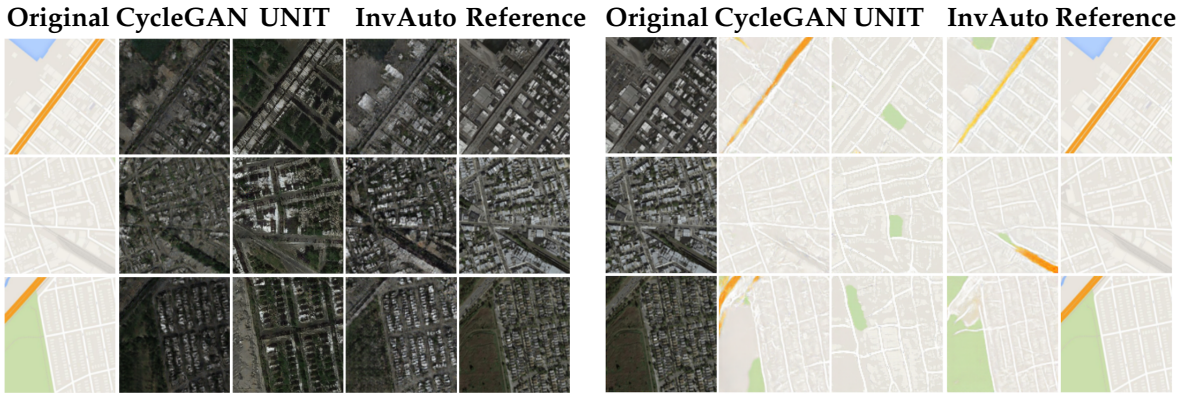


Figure 8. (**Left**) Maps-to-satellite image conversion. Zoomed image is shown in Figure A9 in Appendix A. (**Right**) Satellite-to-maps image conversion. Zoomed image is shown in Figure A10 in Appendix A.

Table 3 contains the results of the numerical evaluation and shows that the performance of InvAuto is similar to the state-of-the-art techniques that we compare InvAuto with and is furthermore contained within the performance range established by the CycleGAN (best performer) and UNIT (consistently slightly worst from CycleGAN).

Table 3. Numerical evaluation of CycleGAN, UNIT, and InvAuto with ℓ_1 reconstruction loss.

	Methods		
Tasks	**CycleGAN**	**UNIT**	**InvAuto**
Night-to-day	0.033	0.227	0.062
Day-to-nigth	0.041	0.114	0.067
Thermal-to-day	0.287	0.339	0.299
Day-to-thermal	0.179	0.194	0.205
Maps-to-satellite	0.261	0.331	0.272
Satellite-to-maps	0.069	0.104	0.080

5.2. Experiments with Autonomous Driving System

To test the quality of the image-to-image translations obtained by InvAuto, we use the NVIDIA evaluation system for autonomous driving described in detail in [31]. The system evaluates the performance of an already trained NVIDIA neural-network-based end-to-end learning platform for autonomous driving (PilotNet) on a test video using a simulator for autonomous driving. The system uses the following performance metrics for evaluation: autonomy, position precision, and comfort. We do not describe these metrics as they are described well in the mentioned paper. We only emphasize that these metrics are expressed as a percentage, where 100% corresponds to the best performance. We collected the high-resolution videos of the same road during the day and night from the camera inside the car. Each video had ∼45 K frames. The pictures were resized to 512 × 512 resolution for the conversion and then resized back to the original size of 1920 × 1208. We used our domain translator as well as CycleGAN to convert the collected day video to a night video and also the collected night video to a day video (Figure 9). To evaluate our model, we used the aforementioned NVIDIA evaluation system, where the converted videos where used as testing sets for this system. We report results in Table 4.

Table 4. Experimental results with autonomous driving system: autonomy, position precision, and comfort.

Video Type	**Autonomy**	**Position Precision**	**Comfort**
Original day	99.6%	73.3%	89.7%
Original night	98.6%	63.1%	86.3%
Day-to-night **InvAuto**	99.0%	69.6%	83.2%
Night-to-day **InvAuto**	99.3%	68.0%	84.7%
Day-to-night **CycleGAN**	99.0%	68.4%	84.7%
Night-to-day	98.8%	64.0%	87.3%

The PilotNet model used for testing was trained mostly on day videos. Thus, it is expected to perform worse on night videos. Therefore, the performance for original night video is worse than for the same video converted to a day video in terms of autonomy and position precision. The comfort deteriorates due to the inconsistency of consecutive frames in the converted video, i.e., the videos

are converted frame-by-frame and we do not apply any post-processing to ensure smooth transition between frames. The results for InvAuto and CycleGAN are comparable.

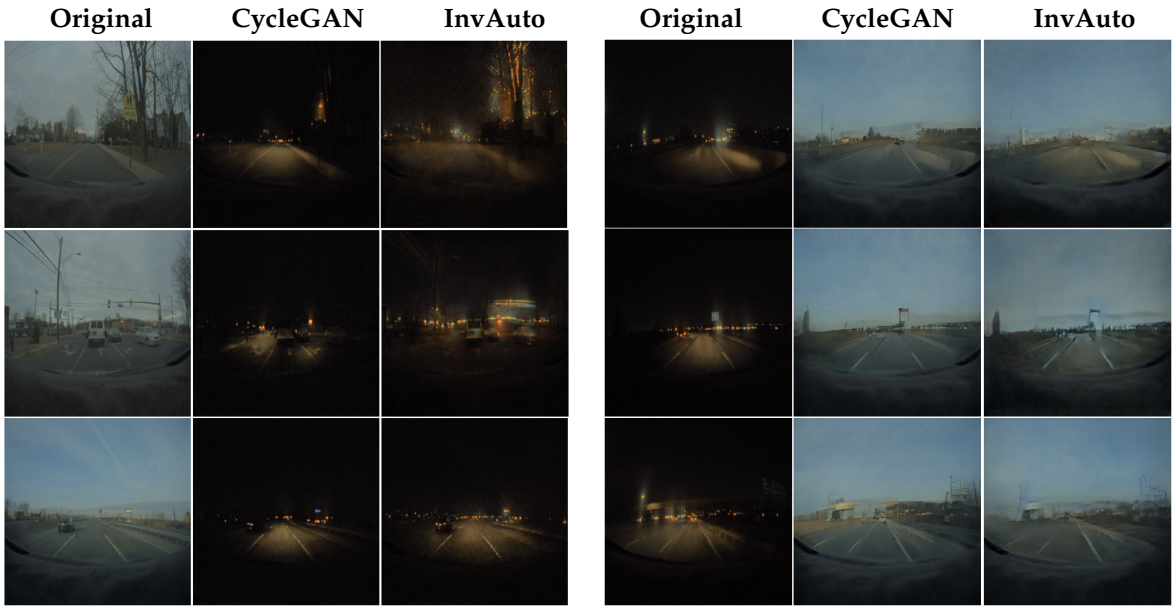


Figure 9. (**Left**) Experimental results with autonomous driving system: day-to-night conversion. Zoomed image is shown in Figure A11 in Appendix A. (**Right**) Experimental results with autonomous driving system: night-to-day conversion. Zoomed image is shown in Figure A12 in Appendix A.

6. Conclusions

We proposed a novel architecture that we call invertible autoencoder, which, as opposed to the common deep learning architectures, allows the layers of the model performing opposite operations (like encoder and decoder) to share weights. This is achieved by enforcing orthonormal mappings in the layers of the model. We demonstrate the applicability of the proposed architecture to the problem of domain adaptation and evaluate it on benchmark datasets and an autonomous driving task. The performance of the proposed approach matches state-of-the-art methods and requires less trainable parameters.

Author Contributions: Y.T. is the lead author of this work. A.C. provided project supervision.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Invertible Autoencoder for Domain Adaptation

- Additional Plots and Tables for Section 3.6

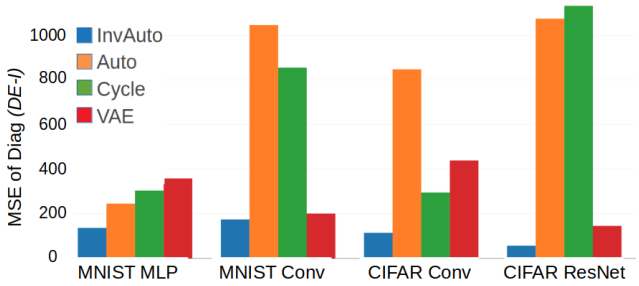


Figure A1. Comparison of the MSE of the diagonal of $DE - I$ for InvAuto, Auto, Cycle, and VAE on MLP, convolutional (Conv), and ResNet architectures and MNIST and CIFAR datasets.

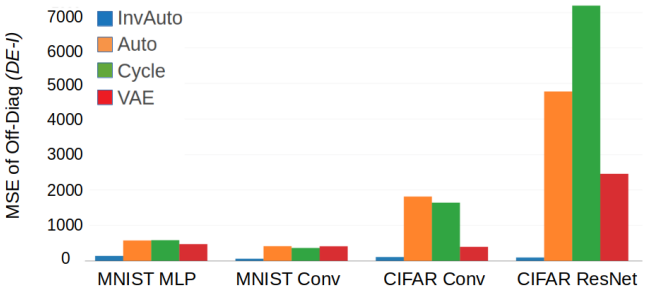


Figure A2. Comparison of the MSE of the off-diagonal of $DE - I$ for InvAuto, Auto, Cycle, and VAE on MLP, convolutional (Conv), and ResNet architectures and MNIST and CIFAR dataset.

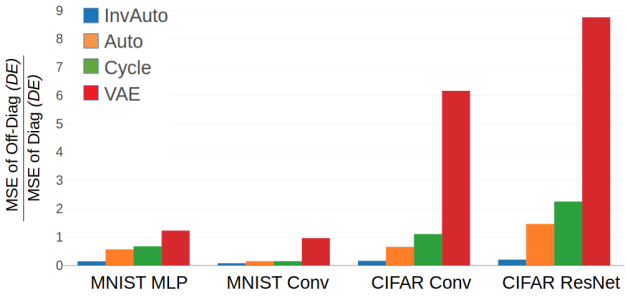


Figure A3. Comparison of the ratio of MSE of the off-diagonal and diagonal of DE for InvAuto, Auto, Cycle, and VAE on MLP, convolutional (Conv), and ResNet architectures and MNIST and CIFAR datasets.

Table A1. Test reconstruction loss (MSE) for InvAuto, Auto, Cycle, and VAE on MLP, convolutional (Conv), and ResNet architectures and MNIST and CIFAR datasets. VAE has significantly higher reconstruction loss by construction.

Dataset and Model	InvAuto	Auto	Cycle	VAE
MNIST MLP	0.189	0.100	0.112	1.245
MNIST Conv	0.168	0.051	0.057	1.412
CIFAR Conv	0.236	0.126	0.195	1.457
CIFAR ResNet	0.032	0.127	0.217	0.964

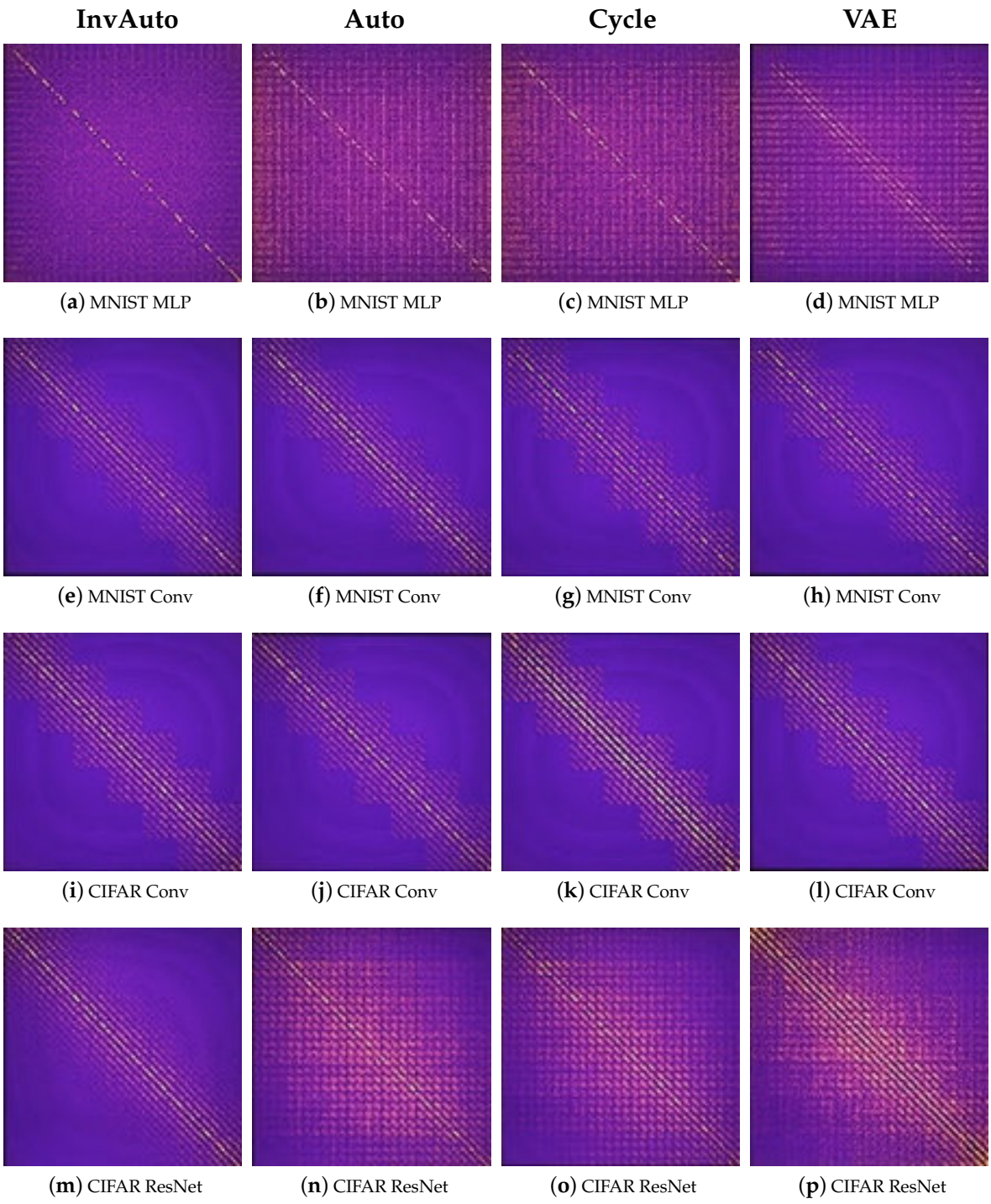


Figure A4. Heatmap of the values of matrix DE for InvAuto, (**a**,**e**,**i**,**m**) Auto (**b**,**f**,**j**,**n**), Cycle (**c**,**g**,**k**,**o**), and VAE (**d**,**h**,**l**,**p**) on MLP, convolutional (Conv), and ResNet architectures and MNIST and CIFAR datasets. Matrices E and D are constructed by multiplying the weight matrices of consecutive layers of encoder and decoder, respectively. In case of InvAuto, DE is the closest to the identity matrix.

- Additional Experimental Results for Section 5

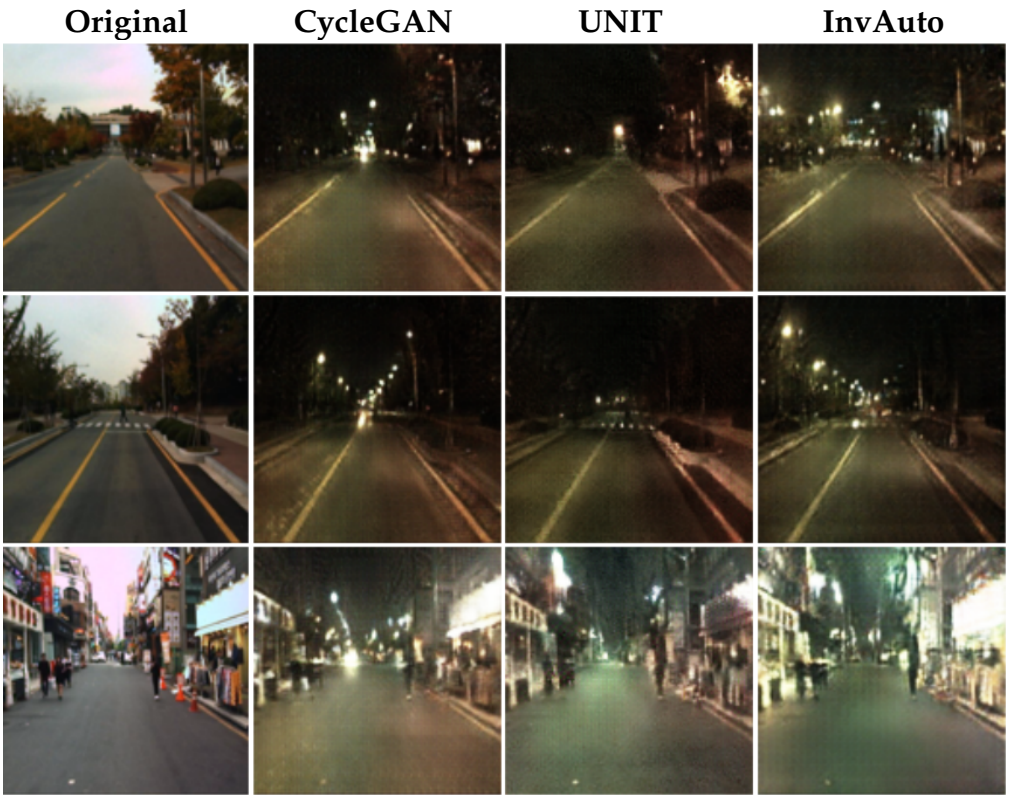


Figure A5. Day-to-night image conversion.

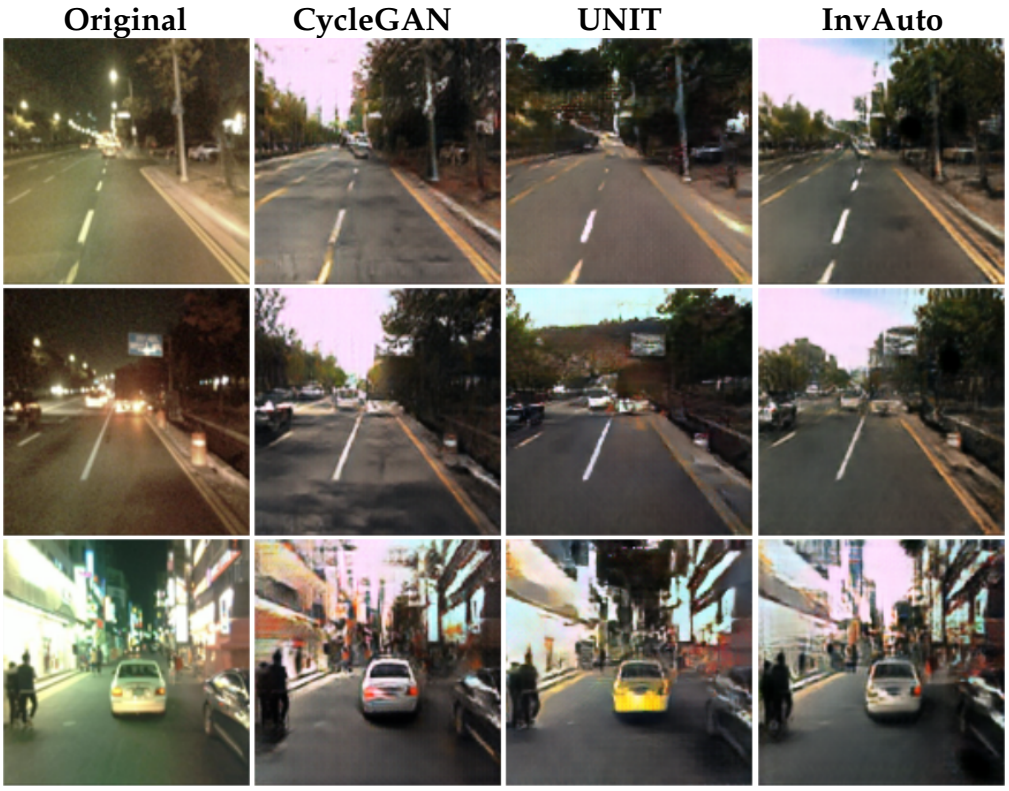


Figure A6. Night-to-day image conversion.

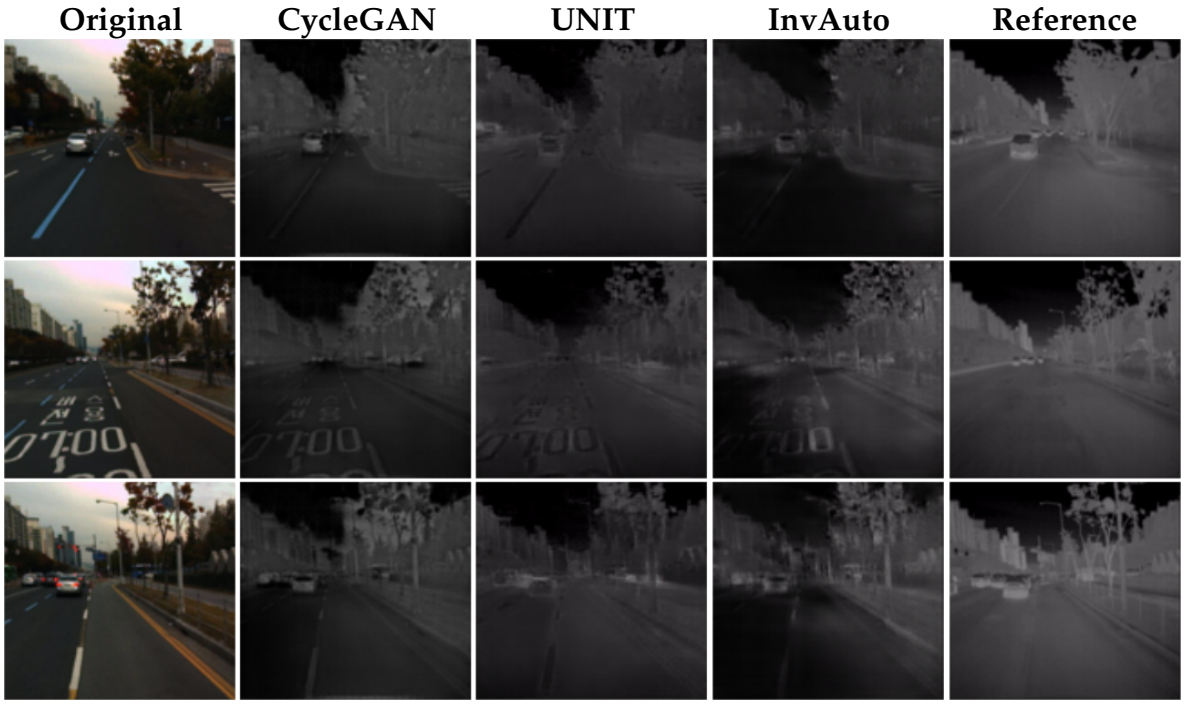


Figure A7. Day-to-thermal image conversion.

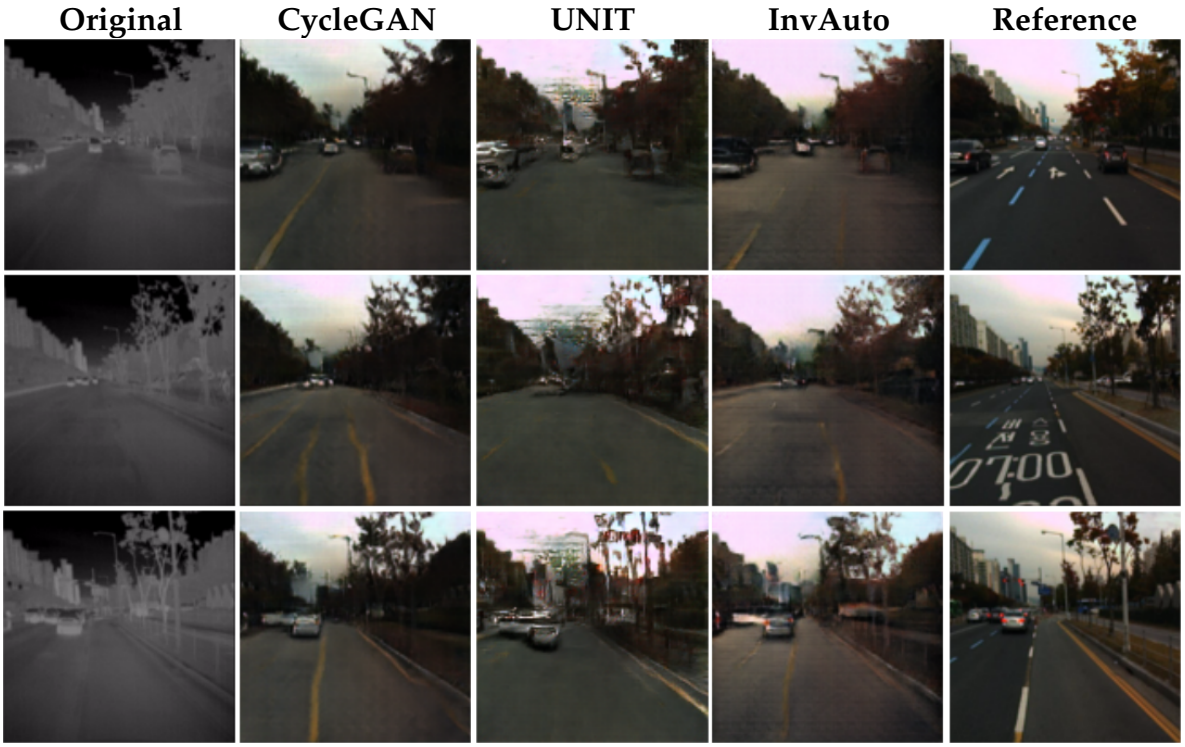


Figure A8. Thermal-to-day image conversion.

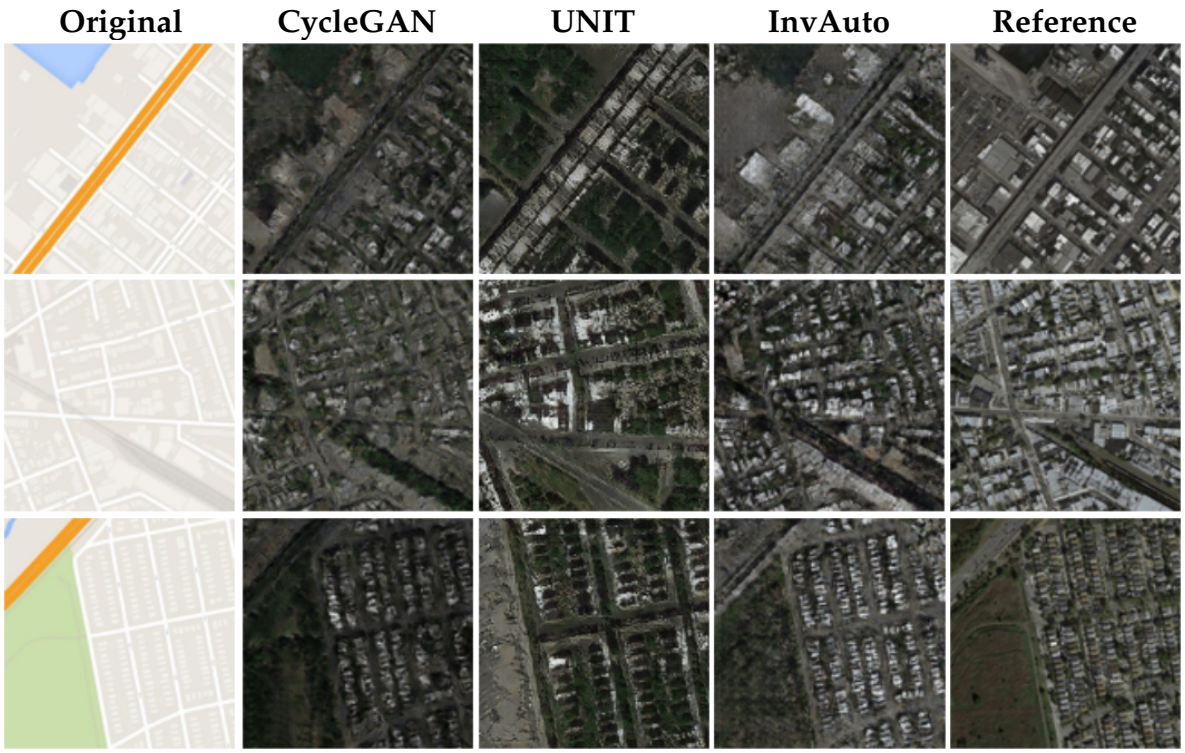


Figure A9. Maps-to-satellite image conversion.

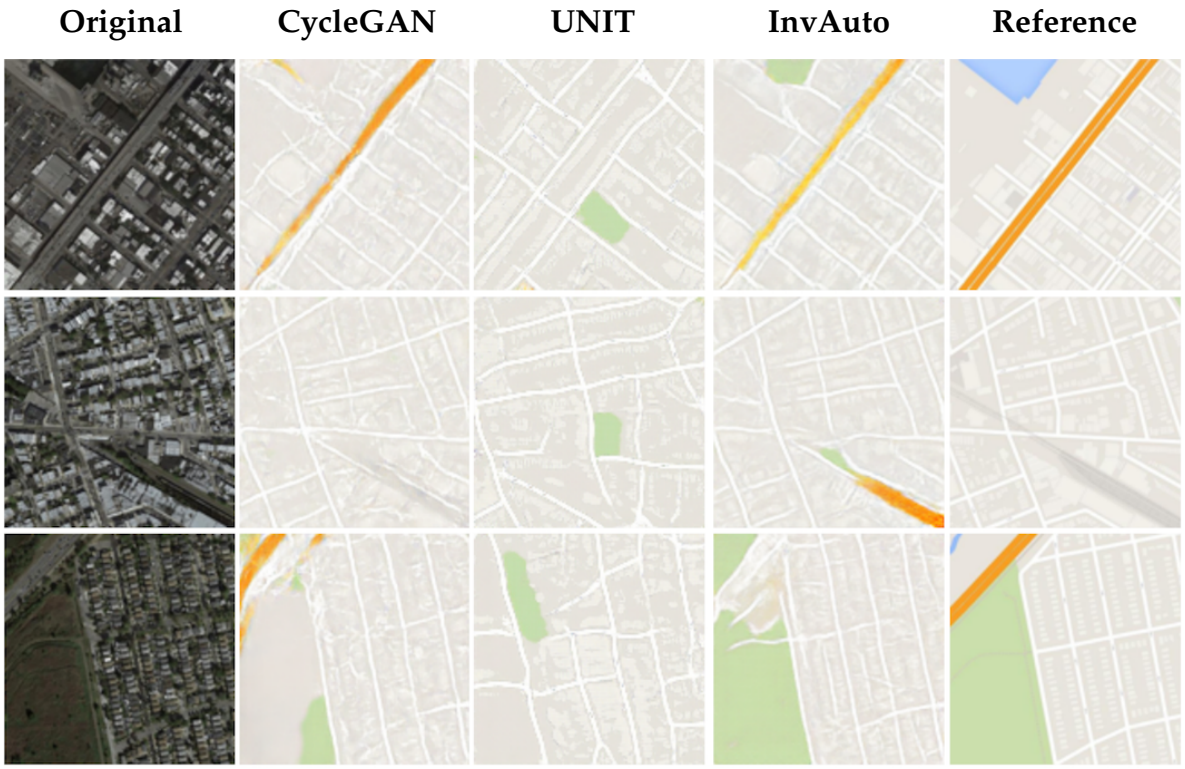


Figure A10. Satellite-to-maps image conversion.

Figure A11. Experimental results with autonomous driving system: day-to-night conversion.

Figure A12. Experimental results with autonomous driving system: night-to-day conversion.

- Invertible Autoencoder for Domain Adaptation: Architecture and Training

Generator architecture Our implementation of InvAuto contains 18 invertible residual blocks for both 128×128 and 512×512 images, where 9 blocks are used in the encoder and the remaining in the decoder. All layers in the decoder are the inverted versions of encoder's layers. We furthermore add two down-sampling layers and two up-sampling layers for the model trained on 128×128 images, and three down-sampling layers and three up-sampling layers for the model trained on 512×512 images. The details of the generator's architecture are listed in Tables A3 and A4. For convenience, we use Conv to denote convolutional layer, ConvNormReLU to denote Convolutional-InstanceNorm-LeakyReLU layer, InvRes to denote invertible residual block, and Tanh to denote hyperbolic tangent activation function. The negative slope of LeakyReLU function is set to 0.2. All filters are square and we have the following notations: K represents filter size and F represents the number of output feature maps. The paddings are added correspondingly.

Discriminator architecture We use similar discriminator architecture as PatchGAN [1]. It is described in Table A2. We use this architecture for training both on 128×128 and 512×512 images.

Criterion and Optimization At training, we set $\lambda = 10$ and use l_1 loss for the cycle consistency in Equation (12). We use Adam optimizer [32] with learning rate $l_r = 0.0002$, $\beta_1 = 0.5$ and $\beta_2 = 0.999$. We also add l_2 penalty with weight 10^{-6}.

Table A2. Discriminator for both 128×128 and 512×512 images.

Name	Stride	Filter
ConvNormReLU	2×2	K4-F64
ConvNormReLU	2×2	K4-F128
ConvNormReLU	2×2	K4-F256
ConvNormReLU	1×1	K4-F512
Conv	1×1	K4-F1

Table A3. Generator for 128×128 images.

Name	Stride	Filter
ConvNormReLU	1×1	K7-F64
ConvNormReLU	2×2	K3-F128
ConvNormReLU	2×2	K3-F256
InvRes	1×1	K3-F256
InvRes	1×1	K3-F256
InvRes	1×1	K3-F256
InvRes	1×1	K3-F256
InvRes	1×1	K3-F256
InvRes	1×1	K3-F256
InvRes	1×1	K3-F256
InvRes	1×1	K3-F256
InvRes	1×1	K3-F256
ConvNormReLU	$1/2 \times 1/2$	K3-F128
ConvNormReLU	$1/2 \times 1/2$	K3-F64
Conv	1×1	K7-F3
Tanh		

Table A4. Generator for 512×512 images.

Name	Stride	Filter
ConvNormReLU	1×1	K7-F64
ConvNormReLU	2×2	K3-F128
ConvNormReLU	2×2	K3-F256
ConvNormReLU	2×2	K3-F512
InvRes	1×1	K3-F512
InvRes	1×1	K3-F512
InvRes	1×1	K3-F512
InvRes	1×1	K3-F512
InvRes	1×1	K3-F512
InvRes	1×1	K3-F512
InvRes	1×1	K3-F512
InvRes	1×1	K3-F512
InvRes	1×1	K3-F512
InvRes	1×1	K3-F512
ConvNormReLU	$1/2 \times 1/2$	K3-F256
ConvNormReLU	$1/2 \times 1/2$	K3-F128
ConvNormReLU	$1/2 \times 1/2$	K3-F64
Conv	1×1	K7-F3
Tanh		

References

1. Isola, P.; Zhu, J.Y.; Zhou, T.; Efros, A.A. Image-to-Image Translation with Conditional Adversarial Networks. In Proceedings of the 2017 Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017.
2. Wang, T.; Liu, M.; Zhu, J.; Tao, A.; Kautz, J.; Catanzaro, B. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. InProceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2017.
3. Zhu, J.Y.; Park, T.; Isola, P.; Efros, A.A. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In Proceedings of the IEEE International Conference on Computer Vision (ICCV 2017), Venice, Italy, 22–29 October 2017.
4. Liu, M.; Breuel, T.; Kautz, J. Unsupervised Image-to-Image Translation Networks. In Proceedings of the Annual Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017.
5. Goodfellow, I.J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.C.; Bengio, Y. Generative Adversarial Nets. In Proceedings of the Annual Conference on Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014.
6. Chen, X.; Duan, Y.; Houthooft, R.; Schulman, J.; Sutskever, I.; Abbeel, P. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In Proceedings of the Annual Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
7. Nguyen, A.; Yosinski, J.; Bengio, Y.; Dosovitskiy, A.; Clune, J. Plug & play generative networks: Conditional iterative generation of images in latent space. In Proceedings of the 2017 Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017.
8. Gan, Z.; Chen, L.; Wang, W.; Pu, Y.; Zhang, Y.; Liu, H.; Li, C.; Carin, L. Triangle Generative Adversarial Networks. In Proceedings of the Annual Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017.
9. Zhang, H.; Xu, T.; Li, H. StackGAN: Text to Photo-Realistic Image Synthesis with Stacked Generative Adversarial Networks. In Proceedings of the IEEE International Conference on Computer Vision (ICCV 2017), Venice, Italy, 22–29 October 2017.
10. Wang, C.; Wang, C.; Xu, C.; Tao, D. Tag Disentangled Generative Adversarial Network for Object Image Re-rendering. In Proceedings of the 2017 International Joint Conference on Artificial Intelligence, Melbourne, Australia, 19–25 August 2017.
11. Wang, W.; Huang, Q.; You, S.; Yang, C.; Neumann, U. Shape Inpainting Using 3D Generative Adversarial Network and Recurrent Convolutional Networks. In Proceedings of the IEEE International Conference on Computer Vision (ICCV 2017), Venice, Italy, 22–29 October 2017.
12. Vondrick, C.; Pirsiavash, H.; Torralba, A. Generating Videos with Scene Dynamics. In Proceedings of the Annual Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
13. Finn, C.; Goodfellow, I.; Levine, S. Unsupervised Learning for Physical Interaction Through Video Prediction. In Proceedings of the Annual Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
14. Vondrick, C.; Torralba, A. Generating the Future with Adversarial Transformers. In Proceedings of the 2017 Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017.
15. Reed, S.; Akata, Z.; Yan, X.; Logeswaran, L.; Schiele, B.; Lee, H. Generative Adversarial Text to Image Synthesis. In Proceedings of the 33rd International Conference on Machine Learning (ICML 2016), New York, NY, USA, 19–24 June 2016.
16. Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; Kavukcuoglu, K. WaveNet: A Generative Model for Raw Audio. *arXiv* **2016**, arXiv:1609.03499.
17. Lu, J.; Kannan, A.; Yang, J.; Parikh, D.; Batra, D. Best of Both Worlds: Transferring Knowledge from Discriminative Learning to a Generative Visual Dialog Model. In Proceedings of the Annual Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017.
18. Kingma, D.P.; Welling, M. Auto-Encoding Variational Bayes. In Proceedings of the 2nd International Conference on Learning Representations (ICLR2014), Banff, AB, Canada, 14–16 April 2014.
19. Dong, H.; Neekhara, P.; Wu, C.; Guo, Y. Unsupervised Image-to-Image Translation with Generative Adversarial Networks. *arXiv* **2017**, arXiv:1701.02676.

20. Taigman, Y.; Polyak, A.; Wolf, L. Unsupervised Cross-Domain Image Generation. *arXiv* **2016**, arXiv:1611.02200.
21. Liu, M.Y.; Tuzel, O. Coupled Generative Adversarial Networks. In Proceedings of the Annual Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016.
22. Vincent, P.; Larochelle, H.; Bengio, Y.; Manzagol, P.A. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th International Conference on Machine Learning (ICML 2008), Helsinki, Finland, 5–9 July 2008.
23. Gatys, L.A.; Ecker, A.S.; Bethge, M. Image Style Transfer Using Convolutional Neural Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016.
24. Johnson, J.; Alahi, A.; Fei-Fei, L. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In Proceedings of the 14th European Conference on Computer Vision, Amsterdam, The Netherlands, 8–16 October 2016.
25. Ulyanov, D.; Lebedev, V.; Vedaldi, A.; Lempitsky, V.S. Texture Networks: Feed-forward Synthesis of Textures and Stylized Images. In Proceedings of the 33rd International Conference on Machine Learning (ICML 2016), New York, NY, USA, 19–24 June 2016.
26. Gatys, L.A.; Bethge, M.; Hertzmann, A.; Shechtman, E. Preserving Color in Neural Artistic Style Transfer. *arXiv* **2016**, arXiv:1606.05897.
27. McCann, M.T.; Jin, K.H.; Unser, M. Convolutional Neural Networks for Inverse Problems in Imaging: A Review. *IEEE Signal Process. Mag.* **2017**, *34*, 85–95.
28. Vasudevan, A.; Anderson, A.; Gregg, D. Parallel Multi Channel convolution using General Matrix Multiplication. In Proceedings of the 28th Annual IEEE International Conference on Application-specific Systems, Architectures and Processors, Seattle, WA, USA, 10–12 July 2017.
29. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016.
30. Hwang, S.; Park, J.; Kim, N.; Choi, Y.; Kweon, I.S. Multispectral pedestrian detection: Benchmark dataset and baseline. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–10 June 2015.
31. Bojarski, M.; Del Testa, D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L.D.; Monfort, M.; Muller, U.; Zhang, J.; et al. End to End Learning for Self-Driving Cars. *arXiv* **2016**, arXiv:1604.07316.
32. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980.