TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks

Alexander Geiger*
MIT
Cambridge, USA
geigera@mit.edu

Dongyu Liu*
MIT
Cambridge, USA
dongyu@mit.edu

Sarah Alnegheimish
MIT
Cambridge, USA
smish@mit.edu

Alfredo Cuesta-Infante
Universidad Rey Juan Carlos
Madrid, Spain
alfredo.cuesta@urjc.es

Kalyan Veeramachaneni
MIT
Cambridge, USA
kalyanv@mit.edu

***Abstract*—Time series anomalies can offer information relevant to critical situations facing various fields, from finance and aerospace to the IT, security, and medical domains. However, detecting anomalies in time series data is particularly challenging due to the vague definition of anomalies and said data's frequent lack of labels and highly complex temporal correlations. Current state-of-the-art unsupervised machine learning methods for anomaly detection suffer from scalability and portability issues, and may have high false positive rates. In this paper, we propose TadGAN, an unsupervised anomaly detection approach built on Generative Adversarial Networks (GANs). To capture the temporal correlations of time series distributions, we use LSTM Recurrent Neural Networks as base models for *Generators* and *Critics*. TadGAN is trained with cycle consistency loss to allow for effective time-series data reconstruction. We further propose several novel methods to compute reconstruction errors, as well as different approaches to combine reconstruction errors and *Critic* outputs to compute anomaly scores. To demonstrate the performance and generalizability of our approach, we test several anomaly scoring techniques and report the best-suited one. We compare our approach to 8 baseline anomaly detection methods on 11 datasets from multiple reputable sources such as NASA, Yahoo, Numenta, Amazon, and Twitter. The results show that our approach can effectively detect anomalies and outperform baseline methods in most cases (6 out of 11). Notably, our method has the highest averaged F1 score across all the datasets. Our code is open source and is available as a benchmarking tool.**

***Index Terms*—Anomaly detection, Generative adversarial network, Time series data**

I. Introduction

The recent proliferation of temporal observation data has led to an increasing demand for time series anomaly detection in many domains, from energy and finance to healthcare and cloud computing. A time series anomaly is defined as a time point or period where a system behaves unusually [1]. Broadly speaking, there are two types of anomalies: A *point anomaly* is a single data point that has reached an unusual value, while a *collective anomaly* is a continuous sequence of data points that are considered anomalous as a whole, even if the individual data points may not be unusual [1].

Time series anomaly detection aims to isolate *anomalous* subsequences of **varied lengths** within time series. One of the simplest detection techniques is *thresholding*, which detects data points that exceed a normal range. However, many anomalies do not exceed any boundaries – for example, they may have values that are purportedly "normal," but are unusual at the specific time that they occur (i.e., *contextual anomalies*). These anomalies are harder to identify because the context of a signal is often unclear [1], [2].

Various statistical methods have been proposed to improve upon thresholding, such as Statistical Process Control (SPC) [3], in which data points are identified as anomalies if they fail to pass statistical hypothesis testing. However, a large amount of human knowledge is still required to set prior assumptions on the models.

Researchers have also studied a number of unsupervised machine learning-based approaches to anomaly detection. One popular method consists of segmenting a time series into subsequences (overlapping or otherwise) of a certain length and applying clustering algorithms to find outliers. Another learns

Deep learning based method	Outperforms ARIMA, 1970 [4]
LSTM AutoEncoder, 2016 [5]	5
LSTM, 2018 [6]	5
MAD-GAN, 2019 [7]	0
MS Azure, 2019 [8]	0
DeepAR, 2019 [9]	6
TadGAN	**8**

TABLE I
THE NUMBER OF WINS OF A PARTICULAR METHOD COMPARED WITH ARIMA, THE TRADITIONAL TIME SERIES FORECASTING MODEL, AGAINST AN APPROPRIATE METRIC (F1 SCORE) ON 11 REAL DATASETS.

* The two authors make equal contributions to this work. D. Liu and K. Veeramachaneni are the co-corresponding authors.

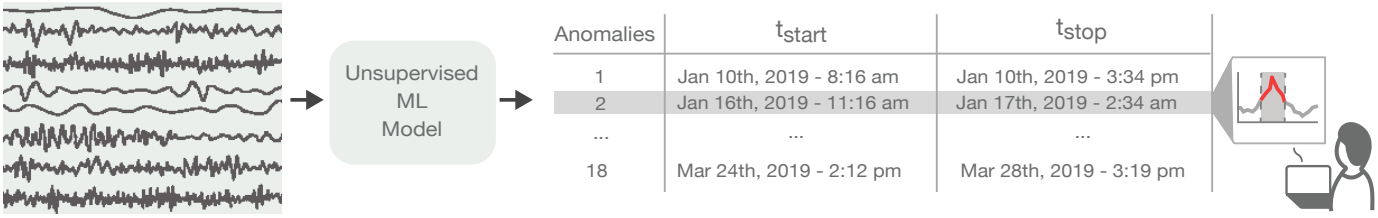

Fig. 1. An illustration of time series anomaly detection using unsupervised learning. Given a multivariate time series, the goal is to find out a set of anomalous time segments that have unusual values and do not follow the expected temporal patterns.

a model that either predicts or reconstructs a time series signal, and makes a comparison between the real and the predicted or reconstructed values. High prediction or reconstruction errors suggest the presence of anomalies.

Deep learning methods [10] are extremely capable of handling non-linearity in complex temporal correlations, and have excellent learning ability. For this reason, they have been used in a number of time series anomaly detection methods [6], [11], [12], including tools created by companies such as Microsoft [8]. Generative Adversarial Networks (GANs) [13] have also been shown to be very successful at generating time series sequences and outperforming state-of-the-art benchmarks [14]. Such a proliferation of methods invites the question: Do these new, complex approaches actually perform better than a simple baseline statistical method? To evaluate the new methods, we used 11 datasets (real and synthetic) that collectively have 492 signals and thousands of known anomalies to set-up a benchmarking system (see the details in Section VI and Table IV). We implemented 5 of the most recent deep learning techniques introduced between 2016 and 2019, and compared their performances with that of a baseline method from the 1970s, ARIMA. While some methods were able to beat ARIMA on 50% of the datasets, two methods failed to outperform it at all (c.f. Table I).

One of the foundational challenges of deep learning-based approaches is that their remarkable ability to fit data carries the risk that they could fit anomalous data as well. For example, autoencoders, using $L2$ objective function, can fit and reconstruct data extremely accurately - thus fitting the anomalies as well. On the other hand, GANs may be ineffective at learning the generator to fully capture the data's hidden distribution, thus causing false alarms. Here, we mix the two methods, creating a more nuanced approach. Additionally, works in this domain frequently emphasize improving the deep learning model itself. However, as we show in this paper, improving post-processing steps could aid significantly in reducing the number of false positives.

In this work, we introduce a novel GAN architecture, TadGAN, for the time series domain. We use TadGAN to reconstruct time series and assess errors in a contextual manner to identify anomalies. We explore different ways to compute anomaly scores based on the outputs from *Generators* and *Critics*. We benchmark our method against several well-known classical- and deep learning-based methods on eleven time series datasets. The detailed results can be found in Table IV.

The key contributions of this paper are as follows:

- We propose a novel unsupervised GAN-reconstruction-based anomaly detection method for time series data. In particular, we introduce a cycle-consistent GAN architecture for time-series-to-time-series mapping.
- We identify two time series similarity measures suitable for evaluating the contextual similarity between original and GAN-reconstructed sequences. Our novel approach leverages GAN's *Generator* and *Critic* to compute robust anomaly scores at every time step.
- We conduct an extensive evaluation using 11 time-series datasets from 3 reputable entities (NASA, Yahoo, and Numenta), demonstrating that our approach outperforms 8 other baselines. We further provide several insights into anomaly detection for time series data using GANs.
- We develop a benchmarking system for time series anomaly detection. The system is open-sourced and can be extended with additional approaches and datasets[1]. At the time of this writing, the benchmark includes 9 anomaly detection pipelines, 13 datasets, and 2 evaluation mechanisms.

The rest of this paper is structured as follows. We formally lay out the problem of time series anomaly detection in Section II. Section III presents an overview of the related literature. Section IV introduces the details of our GAN model. We describe how to use GANs for anomaly detection in Section V, and evaluate our proposed framework in Section VI. Finally, Section VII summarizes the paper and reports our key findings.

II. Unsupervised Time Series Anomaly Detection

Given a time series $\mathbf{X} = (x^1, x^2, \cdots, x^T)$, where $x^i \in \mathbf{R}^{M \times 1}$ indicates M types of measurements at time step i, the goal of *unsupervised* time series anomaly detection is to find a set of anomalous time segments $\boldsymbol{A_{seq}} = \{\boldsymbol{a^1_{seq}}, \boldsymbol{a^2_{seq}}, \cdots, \boldsymbol{a^k_{seq}}\}$, where $\boldsymbol{a^i_{seq}}$ is a continuous sequence of data points in time that show anomalous or unusual behaviors (Figure 1) – values within the segment that appear not to comply with the expected temporal behavior of the signal. A few aspects of this problem make it both distinct from and more difficult than time series classification [15] or supervised time series anomaly detection [16], as well as more pertinent to many industrial applications. We highlight them here:

[1]The software is available at github (https://github.com/signals-dev/Orion)

– No a priori knowledge of anomalies or possible anomalies: Unlike with supervised time series anomaly detection, we do not have any previously identified "known anomalies" with which to train and optimize the model. Rather, we train the model to to learn the time series patterns, ask it to detect anomalies, and then check whether the detector identified anything relevant to end users.
– Non availability of "normal baselines" : For many real-world systems, such as wind turbines and aircraft engines, simulation engines can produce a signal that resembles normal conditions, which can be tweaked for different control regimes or to account for degradation or aging. Such simulation engines are often physics-based and provide "*normal baselines*," which can be used to train models such that any deviations from them are considered anomalous. Unsupervised time series anomaly detection strategies do not rely on the availability of such baselines, instead learning time series patterns from real-world signals – signals that may themselves include anomalies or problematic patterns.
– Not all detected anomalies are problematic: Detected "*anomalies*" may not actually indicate problems, and could instead result from external phenomena (such as sudden shifts in environmental conditions), auxiliary information (such as the fact that a test run is being performed), or other variables that the algorithm did not consider, such as regime or control setting changes. Ultimately, it is up to the end user, the domain expert, to assess whether the anomalies identified by the model are problematic. Figure 1 highlights how a trained unsupervised machine learning model can be used in real time for the incoming data.
– No clear segmentation possible: Many signals, such as those associated with periodic time series, can be segmented – for example, an electrocardiogram signal (ECG) can be separated into similar segments that pertain to periods [16], [17]. The resulting segment clusters may reveal different collective patterns, along with anomalous patterns. We focus on signals that cannot be clearly segmented, making these approaches unfeasible. The length of $\boldsymbol{a^i}$ is also variable and is not known a priori, which further increases the difficulty.
– How do we evaluate these competing approaches? For this, we rely on several datasets that contain "known anomalies", the details of which are introduced in Section VI-A. Presumably, the "*anomalies*" are time segments that have been manually identified as such by some combination of algorithmic approaches and human expert annotation. These "*anomalies*" are used to evaluate the efficacy of our proposed unsupervised models. More details about this can be found in Section VI-B3.

III. Related Work

Over the past several years, the rich variety of anomaly types, data types and application scenarios has spurred a range of anomaly detection approaches [1], [18]–[20]. In this section, we discuss some of the unsupervised approaches. The simplest of these are out-of-limit methods, which flag regions where values exceed a certain threshold [21], [22]. While these methods are intuitive, they are inflexible and incapable of detecting contextual anomalies. To overcome this, more advanced techniques have been proposed, namely proximity-based, prediction-based, and reconstruction-based anomaly detection (Table II).

Methodology	Papers
Proximity	[23]–[25]
Prediction	[2], [6], [26], [27]
Reconstruction	[5], [28]–[30]
Reconstruction (GANs)	[7], [14], [31]

TABLE II
UNSUPERVISED APPROACHES TO TIME SERIES ANOMALY DETECTION.

A. Anomaly Detection for Time Series Data.

Proximity-based methods first use a distance measure to quantify similarities between objects – single data points for point anomalies, or fixed length sequences of data points for collective anomalies. Objects that are distant from others are considered anomalies. This detection type can be further divided into distance-based methods, such as K-Nearest Neighbor (KNN) [24] – which use a given radius to define neighbors of an object, and the number of neighbors to determine an anomaly score – and density-based methods, such as Local Outlier Factor (LOF) [23] and Clustering-Based Local Outlier Factor [25], which further consider the density of an object and that of its neighbors. There are two major drawbacks to applying proximity-based methods in time series data: (1) a priori knowledge about anomaly duration and the number of anomalies is required; (2) these methods are unable to capture temporal correlations.

Prediction-based methods learn a predictive model to fit the given time series data, and then use that model to predict future values. A data point is identified as an anomaly if the difference between its predicted input and the original input exceeds a certain threshold. Statistical models, such as ARIMA [26], Holt-Winters [26], and FDA [27], can serve this purpose, but are sensitive to parameter selection, and often require strong assumptions and extensive domain knowledge about the data. Machine learning-based approaches attempt to overcome these limitations. [2] introduce Hierarchical Temporal Memory (HTM), an unsupervised online sequence memory algorithm, to detect anomalies in streaming data. HTM encodes the current input to a hidden state and predicts the next hidden state. A prediction error is measured by computing the difference between the current hidden state and the predicted hidden state. Hundman et al. [6] propose Long Short Term Recurrent Neural Networks (LSTM RNNs), to predict future time steps and flag large deviations from predictions.

Reconstruction-based methods learn a model to capture the latent structure (low-dimensional representations) of the

given time series data and then create a synthetic reconstruction of the data. Reconstruction-based methods assume that anomalies lose information when they are mapped to a lower dimensional space and thereby cannot be effectively reconstructed; thus, high reconstruction errors suggest a high chance of being anomalous.

Principal Component Analysis (PCA) [28], a dimensionality-reduction technique, can be used to reconstruct data, but this is limited to linear reconstruction and requires data to be highly correlated and to follow a Gaussian distribution [29]. More recently, deep learning based techniques have been investigated, including those that use Auto-Encoder (AE) [30], Variational Auto-Encoder (VAE) [30] and LSTM Encoder-Decoder [5].

However, without proper regularization, these reconstruction-based methods can easily become overfitted, resulting in low performance. In this work, we propose the use of adversarial learning to allow for time series reconstruction. We introduce an intuitive approach for regularizing reconstruction errors. The trained *Generators* can be directly used to reconstruct more concise time series data – thereby providing more accurate reconstruction errors – while the *Critics* can offer scores as a powerful complement to the reconstruction errors when computing an anomaly score.

B. Anomaly Detection Using GANs.

Generative adversarial networks can successfully perform many image-related tasks, including image generation [13], image translation [32], and video prediction [33], and researchers have recently demonstrated the effectiveness of GANs for anomaly detection in images [34], [35].

Adversarial learning for images. Schlegl et al. [36] use the Critic network in a GAN to detect anomalies in medical images. They also attempt to use the reconstruction loss as an additional anomaly detection method, and find the inverse mapping from the data space to the latent space. This mapping is done in a separate step, after the GAN is trained. However, Zenati et al. [37] indicate that this method has proven impractical for large data sets or real-time applications. They propose a bi-directional GAN for anomaly detection in tabular and image data sets, which allows for simultaneous training of the inverse mapping through an encoding network.

The idea of training both encoder and decoder networks was developed by Donahue et al. [38] and Dumoulin et al. [39], who show how to achieve bidirectional GANs by trying to match joint distributions. In an optimal situation, the joint distributions are the same, and the Encoder and Decoder must be inverses of each other. A cycle-consistent GAN was introduced by Zhu et al. [32], who have two networks try to map into opposite dimensions, such that samples can be mapped from one space to the other and vice versa.

Adversarial learning for time series. Prior GAN-related work has rarely involved time series data, because the complex temporal correlations within this type of data pose significant challenges to generative modeling. Three works published in 2019 are of note. First, to use GANs for anomaly detection in time series, Li et al. [7] propose using a vanilla GAN model to capture the distribution of a multivariate time series, and using the Critic to detect anomalies. Another approach in this line is BeatGAN [31], which is a Encoder and Decoder GAN architecture that allows for the use of the reconstruction error for anomaly detection in heartbeat signals. More recently, Yoon et al. [14] propose a time series GAN which adopts the same idea but introduces temporal embeddings to assist network training. However, their work is designed for time series representation learning instead of anomaly detection.

To the best of our knowledge, we are the first to introduce cycle-consistent GAN architectures for time series data, such that *Generators* can be directly used for time series reconstructions. In addition, we systematically investigate how to utilize *Critic* and *Generator* outputs for anomaly score computation. A complete framework of time series anomaly detection is introduced to work with GANs.

IV. Adversarial Learning for Time Series Reconstruction

The core idea behind reconstruction-based anomaly detection methods is to learn a model that can encode a data point (in our case, a segment of a time series) and then decode the encoded one (i.e., reconstructed one). An effective model should not be able to reconstruct anomalies as well as "normal" instances, because anomalies will lose information during encoding. In our model, we learn two mapping functions between two domains X and Z, namely $\mathcal{E} : X \to Z$ and $\mathcal{G} : Z \to X$ (Fig. 2). X denotes the input data domain, describing the given training samples $\{(x_i^{1\dots t})\}_{i=1}^{N}$, $x_i^{1\dots t} \in X$. Z represents the latent domain, where we sample random vectors z to represent white noise. We follow a standard multivariate normal distribution, i.e., $z \sim \mathbb{P}_Z = \mathcal{N}(0, 1)$. For notational convenience we use x_i to denote a time sequence of length t starting at time step i. With the mapping functions, we can reconstruct the input time series: $x_i \to \mathcal{E}(x_i) \to \mathcal{G}(\mathcal{E}(x_i)) \approx \hat{x}_i$.

We propose leveraging adversarial learning approaches to obtain the two mapping functions $\mathcal{E}$ and $\mathcal{G}$. As illustrated in Fig. 2, we view the two mapping functions as *Generators*. Note that $\mathcal{E}$ is serving as an Encoder, which maps the time series sequences into the latent space, while $\mathcal{G}$ is serving as a Decoder, which transforms the latent space into the reconstructed time series. We further introduce two adversarial *Critics* (aka discriminators) $\mathcal{C}_x$ and $\mathcal{C}_z$. The goal of $\mathcal{C}_x$ is to distinguish between the real time series sequences from X and the generated time series sequences from $\mathcal{G}(z)$, while $\mathcal{C}_z$ measures the performance of the mapping into latent space. In other words, $\mathcal{G}$ is trying to fool $\mathcal{C}_x$ by generating real-looking sequences. Thus, our high-level objective consists of two terms: (1) *Wasserstein losses* [40], to match the distribution of generated time series sequences to the data distribution in the target domain; and (2) *cycle consistency losses* [32], to prevent the contradiction between $\mathcal{E}$ and $\mathcal{G}$.

ENCODER

$\mathcal{E} : X \to Z$

DECODER

$\mathcal{G} : Z \to X$

A. Wasserstein Loss

The original formulation of GAN that applies the standard adversarial losses (Eq. 1) suffers from the mode collapse problem.

$$\mathcal{L} = \mathbb{E}_{x\sim\mathbb{P}_X}[\log\ \mathcal{C}_x(x)] + \mathbb{E}_{z\sim\mathbb{P}_Z}[\log(1-\mathcal{C}_x(\mathcal{G}(z)))] \quad (1)$$

where $\mathcal{C}_x$ produces a probability score from 0 to 1 indicating the realness of the input time series sequence. To be specific, the *Generator* tends to learn a small fraction of the variability of the data, such that it cannot perfectly converge to the target distribution. This is mainly because the *Generator* prefers to produce those samples that have already been found to be good at fooling the *Critic*, and is reluctant to produce new ones, even though new ones might be helpful to capture other "modes" in the data.

To overcome this limitation, we apply Wasserstein loss [40] as the adversarial loss to train the GAN. We make use of the Wasserstein-1 distance when training the *Critic* network. Formally, let $\mathbb{P}_X$ be the distribution over X. For the mapping function $\mathcal{G} : Z \rightarrow X$ and its *Critic* C_x, we have the following objective:

$$\min_{\mathcal{G}} \max_{\mathcal{C}_x\in\mathbf{C_x}} V_X(\mathcal{C}_x, \mathcal{G}) \quad (2)$$

with

$$V_X(\mathcal{C}_x, \mathcal{G}) = \mathbb{E}_{x\sim\mathbb{P}_X}[\mathcal{C}_x(x)] - \mathbb{E}_{z\sim\mathbb{P}_Z}[\mathcal{C}_x(\mathcal{G}(z)))] \quad (3)$$

where $\mathcal{C}_x \in \mathbf{C_x}$ which denotes the set of 1-Lipschitz continuous functions. K-Lipschitz continuous functions are defined as follows: $\|f(x_1) - f(x_2)\| \leq K\|x_1 - x_2\|, \forall x_1, x_2 \in dom\, f$. The Lipschitz continuous functions constrain the upper bound of the function, further smoothing the function. Therefore, the weights will not change dramatically when updated with gradient descent methods. This reduces the risk of gradient explosion, and makes the model training more stable and reliable. In addition, to enforce the 1-Lipschitz constraint during training, we apply a gradient penalty regularization term as introduced by Gulrajani et al. [41], which penalizes gradients not equal to 1 (cf. line 5).

Following a similar approach, we introduce a Wasserstein loss for the mapping function $\mathcal{E} : X \rightarrow Z$ and its *Critic* C_z. The objective is expressed as:

$$\min_{\mathcal{E}} \max_{\mathcal{C}_z\in\mathbf{C_z}} V_Z(\mathcal{C}_z, \mathcal{E}) \quad (4)$$

The purpose of the second *Critic* C_z is to distinguish between random latent samples $z \sim \mathbb{P}_Z$ and encoded samples $\mathcal{E}(x)$ with $x \sim \mathbb{P}_X$. We present the model type and architecture for $\mathcal{E}$, $\mathcal{G}$, $\mathcal{C}_x$, $\mathcal{C}_z$ in section VI-B.

B. Cycle Consistency Loss

The purpose of our GAN is to reconstruct the input time series: $x_i \rightarrow \mathcal{E}(x_i) \rightarrow \mathcal{G}(\mathcal{E}(x_i)) \approx \hat{x}_i$. However, training the GAN with adversarial losses (i.e., Wasserstein losses) alone cannot guarantee mapping individual input x_i to a desired output z_i which will be further mapped back to $\hat{x}_i$. To reduce the possible mapping function search space, we adapt cycle consistency loss to time series reconstruction, which was first introduced by Zhu et al. [32] for image translation tasks. We train the generative network $\mathcal{E}$ and $\mathcal{G}$ with the adapted cycle consistency loss by minimizing the L2 norm of the difference between the original and the reconstructed samples:

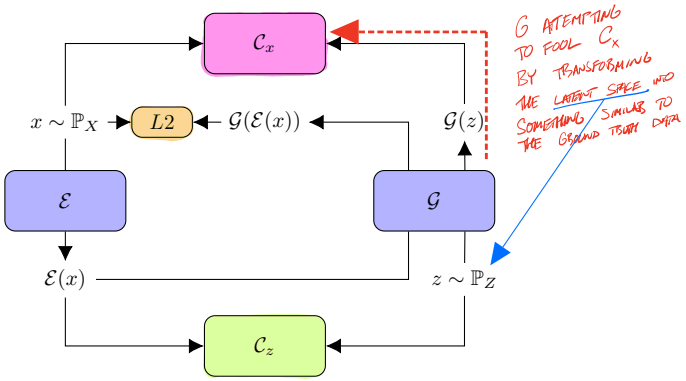

Fig. 2. Model architecture: *Generator* $\mathcal{E}$ is serving as an Encoder which maps the time series sequences into the latent space, while *Generator* $\mathcal{G}$ is serving as a Decoder that transforms the latent space into the reconstructed time series. *Critic* $\mathcal{C}_x$ is to distinguish between real time series sequences from X and the generated time series sequences from $\mathcal{G}(z)$, whereas *Critic* $\mathcal{C}_z$ measures the goodness of the mapping into the latent space.

$$V_{L2}(\mathcal{E}, \mathcal{G}) = \mathbb{E}_{x\sim\mathbb{P}_X}[\|x - \mathcal{G}(\mathcal{E}(x))\|_2] \quad (5)$$

Considering that our target is anomaly detection, we use the L2 norm instead of L1 norm (the one used by Zhu et al. [32] for image translation) to emphasize the impacts of anomalous values. In our preliminary experiments, we observed that adding the backward consistency loss (i.e., $\mathbb{E}_{z\sim\mathbb{P}_z}[\|z - \mathcal{E}(\mathcal{G}(z))\|_2]$) did not improve performance.

C. Full Objective

Combining all of the objectives given in (3),(4) and (5) leads to the final MinMax problem:

$$\min_{\{\mathcal{E},\mathcal{G}\}} \max_{\{\mathcal{C}_x\in\mathbf{C_x}, \mathcal{C}_z\in\mathbf{C_z}\}} V_X(\mathcal{C}_x, \mathcal{G}) + V_Z(\mathcal{C}_z, \mathcal{E}) + V_{L2}(\mathcal{E}, \mathcal{G}) \quad (6)$$

The full architecture of our model can be seen in Figure 2. The benefits of this architecture with respect to anomaly detection are twofold. First, we have a *Critic* $\mathcal{C}_x$ that is trained to distinguish between real and fake time series sequences, hence the score of the *Critic* can directly serve as an anomaly measure. Second, the two *Generators* trained with cycle consistency loss allow us to encode and decode a time series sequence. The difference between the original sequence and the decoded sequence can be used as a second anomaly detection measure. For detailed training steps, please refer to the pseudo code (cf. line 1–14). The following section will introduce the details of using TadGAN for anomaly detection.

V. Time-series GAN for Anomaly Detection (TadGAN)

Let us assume that the given time series is $\mathbf{X} = (x^1, x^2, \cdots, x^T)$, where $x^i \in \mathbf{R}^{M\times 1}$ indicates M types of measurements at time step i. For simplicity, we use $M = 1$

Algorithm 1: TadGAN

Require: m, batch size.
$epoch$, number of iterations over the data.
n_{critic}, number of iterations of the critic per epoch.
η, step size.

1 **for** *each epoch* **do**
2 **for** $\kappa = 0, \ldots, n_{critic}$ **do**
3 Sample $\{(x_i^{1\cdots t})\}_{i=1}^m$ from real data.
4 Sample $\{(z_i^{1\cdots k})\}_{i=1}^m$ from random.
5 $g_{w_{\mathcal{C}_x}} = \nabla_{w_{\mathcal{C}_x}}[\frac{1}{m}\sum_{i=1}^m \mathcal{C}_x(x_i) - \frac{1}{m}\sum_{i=1}^m \mathcal{C}_x(\mathcal{G}(z_i)) + gp(x_i, \mathcal{G}(z_i))]$
6 $w_{\mathcal{C}_x} = w_{\mathcal{C}_x} + \eta \cdot \texttt{adam}(w_{\mathcal{C}_x}, g_{w_{\mathcal{C}_x}})$
7 $g_{w_{\mathcal{C}_z}} = \nabla_{w_{\mathcal{C}_z}}[\frac{1}{m}\sum_{i=1}^m \mathcal{C}_z(z_i) - \frac{1}{m}\sum_{i=1}^m \mathcal{C}_z(\mathcal{E}(x_i)) + gp(z_i, \mathcal{E}(x_i))]$
8 $w_{\mathcal{C}_z} = w_{\mathcal{C}_z} + \eta \cdot \texttt{adam}(w_{\mathcal{C}_z}, g_{w_{\mathcal{C}_z}})$
9 **end**
10 Sample $\{(x_i^{1\cdots t})\}_{i=1}^m$ from real data.
11 Sample $\{(z_i^{1\cdots k})\}_{i=1}^m$ from random.
12 $g_{w_{\mathcal{G},\mathcal{E}}} = \nabla_{w_{\mathcal{G}},w_{\mathcal{E}}}[\frac{1}{m}\sum_{i=1}^m \mathcal{C}_x(x_i) - \frac{1}{m}\sum_{i=1}^m \mathcal{C}_x(\mathcal{G}(z_i)) + \frac{1}{m}\sum_{i=1}^m \mathcal{C}_z(z_i) - \frac{1}{m}\sum_{i=1}^m \mathcal{C}_z(\mathcal{E}(x_i)) + \frac{1}{m}\sum_{i=1}^m \|x_i - \mathcal{G}(\mathcal{E}(x_i))\|_2]$
13 $w_{\mathcal{G},\mathcal{E}}, = w_{\mathcal{G},\mathcal{E}} + \eta \cdot \texttt{adam}(w_{\mathcal{G},\mathcal{E}}, g_{w_{\mathcal{G},\mathcal{E}}})$
14 **end**
15 $X = \{(x_i^{1\cdots t})\}_{i=1}^n$
16 **for** $i = 1, \ldots, n$ **do**
17 $\hat{x}_i = \mathcal{G}(\mathcal{E}(x_i))$;
18 $RE(x_i) = f(x_i, \hat{x}_i)$;
19 $score = \alpha Z_{RE}(x_i) + (1-\alpha) Z_{\mathcal{C}_x}(\hat{x}_i)$
20 **end**

in the later description. Therefore, $\mathbf{X}$ is now a univariate time series and x^i is a scalar. The same steps can be applied for multivariate time series (i.e., when $M > 1$).

To obtain the training samples, we introduce a sliding window with window size t and step size s to divide the original time series into N sub-sequences $X = \{(x_i^{1\cdots t})\}_{i=1}^N$, where $N = \frac{T-t}{s}$. In practice, it is difficult to know the ground truth, and anomalous data points are rare. Hence, we assume all the training sample points are normal. In addition, we generate $Z = \{(z_i^{1\cdots k})\}_{i=1}^N$ from a random space following normal distribution, where k denotes the dimension of the latent space. Then, we feed X and Z to our GAN model and train it with the objective defined in (6). With the trained model, we are able to compute anomaly scores (or likelihoods) at every time step by leveraging the reconstruction error and *Critic* output (cf. line 15–20).

A. Estimating Anomaly Scores using Reconstruction Errors

Given a sequence $x_i^{1\cdots t}$ of length t (denoted as x_i later), TadGAN generates a reconstructed sequence of the same length: $x_i \rightarrow \mathcal{E}(x_i) \rightarrow \mathcal{G}(\mathcal{E}(x_i)) \approx \hat{x}_i$. Therefore, for each time point j, we have a collection of reconstructed values $\{\hat{x}_i^q, i+q=j\}$ We take the median from the collection as the final reconstructed value $\hat{x}^j$. Note that in the preliminary experiments, we found that using the median achieved a better performance than using the mean. Now, the reconstructed time series is $(\hat{x}^1, \hat{x}^2, \cdots, \hat{x}^T)$. Here we propose three different types of functions (cf. line 18) for computing the reconstruction errors at each time step (assume the interval between neighboring time steps is the same).

Point-wise difference. This is the most intuitive way to define the reconstruction error, which computes the difference between the true value and the reconstructed value at every time step:

$$s_t = \left| x^t - \hat{x}^t \right| \tag{7}$$

Area difference. This is applied over windows of a certain length to measure the similarity between local regions. It is defined as the average difference between the areas beneath two curves of length l:

$$s_t = \frac{1}{2*l}\left| \int_{t-l}^{t+l} x^t - \hat{x}^t \, dx \right| \tag{8}$$

Although this seems intuitive, it is not often used in this context – however, we will show in our experiments that this approach works well in many cases. Compared with the point-wise difference, the area difference is good at identifying the regions where small differences exist over a long period of time. Since we are only given fixed samples of the functions, we use the trapezoidal rule to calculate the definite integral in the implementation.

Dynamic time warping (DTW). DTW aims to calculate the optimal match between two given time sequences [42] and is used to measure the similarity between local regions. We have two time series $X = (x_{t-1}, x_{t-l+1}, \ldots, x_{t+l})$ and $\hat{X} = (\hat{x}_{t-1}, \hat{x}_{t-l+1}, \ldots, \hat{x}_{t+l})$ and let $W \in \mathbf{R}^{2*l \times 2*l}$ be a matrix such that the (i,j)-th element is a distance measure between x_i and $\hat{x}_j$, denoted as w_k. We want to find the warp path $W^* = (w_1, w_2, \ldots, w_K)$ that defines the minimum distance between the two curves, subject to boundary conditions at the start and end, as well as constraints on continuity and monotonicity. The DTW distance between time series X and $\hat{X}$ is defined as follows:

$$s_t = W^* = \text{DTW}(X, \hat{X}) = \min_W \left[\frac{1}{K} \sqrt{\sum_{k=1}^{K} w_k} \right] \tag{9}$$

Similar to area difference, DTW is able to identify the regions of small difference over a long period of time, but DTW can handle time shift issues as well.

B. Estimating Anomaly Scores with Critic Outputs

During the training process, the *Critic* C_x has to distinguish between real input sequences and synthetic ones. Because we use the Wasserstein-1 distance when training C_x, the outputs can be seen as an indicator of how real (larger value) or fake (smaller value) a sequence is. Therefore, once the *Critic* is

	NASA		Yahoo S5				NAB				
Property	SMAP	MSL	A1	A2	A3	A4	Art	AdEx	AWS	Traf	Tweets
# SIGNALS	53	27	67	100	100	100	6	5	17	7	10
# ANOMALIES	67	36	178	200	939	835	6	11	30	14	33
point ($len = 1$)	0	0	68	33	935	833	0	0	0	0	0
collective ($len > 1$)	67	36	110	167	4	2	6	11	30	14	33
# ANOMALY POINTS	54696	7766	1669	466	943	837	2418	795	6312	1560	15651
# out-of-dist	18126	642	861	153	21	49	123	15	210	86	520
(% tot.)	33.1%	8.3%	51.6%	32.8%	2.2%	5.9%	5.1%	1.9%	3.3%	5.5%	3.3%
# DATA POINTS	562800	132046	94866	142100	168000	168000	24192	7965	67644	15662	158511
IS SYNTHETIC?				✓	✓	✓	✓				

TABLE III
DATASET SUMMARY: OVERALL THE BENCHMARK DATASET CONTAINS A TOTAL OF 492 SIGNALS AND 2349 ANOMALIES.

trained, it can directly serve as an anomaly measure for time series sequences.

Similar to the reconstruction errors, at time step j, we have a collection of *Critic* scores $(c_i^q, i + q = j)$. We apply kernel density estimation (KDE) on the collection and then take the maximum value as the smoothed value c^j Now the *Critic* score sequence is $(c^1, c^2, \ldots, c^T)$. We show in our experiments that it is indeed the case that the *Critic* assigns different scores to anomalous regions compared to normal regions. This allows for the use of thresholding techniques to identify the anomalous regions.

C. Combining Both Scores

The reconstruction errors $RE(x)$ and *Critic* outputs $\mathcal{C}_x(x)$ cannot be directly used together as anomaly scores. Intuitively, the larger $RE(x)$ and the smaller $\mathcal{C}_x(x)$ indicate higher anomaly scores. Therefore, we first compute the mean and standard deviation of $RE(x)$ and $\mathcal{C}_x(x)$, and then calculate their respective z-scores $Z_{RE}(x)$ and $Z_{\mathcal{C}_x}(x)$ to normalize both. Larger z-scores indicate high anomaly scores.

We have explored different ways to leverage $Z_{RE}(x)$ and $Z_{\mathcal{C}_x}(x)$. As shown in Table V (row 1–4), we first tested three types of $Z_{RE}(x)$ and $Z_{\mathcal{C}_x}(x)$ individually. We then explored two different ways to combine them (row 5 to the last row). First, we attempt to merge them into a single value $\boldsymbol{a}(x)$ with a convex combination (cf. line 19) [7], [36]:

$$\boldsymbol{a}(x) = \alpha Z_{RE}(x) + (1 - \alpha) Z_{\mathcal{C}_x}(x) \tag{10}$$

where α controls the relative importance of the two terms (by default $alpha = 0.5$). Second, we try to multiply both scores to emphasize the high values:

$$\boldsymbol{a}(x) = \alpha Z_{RE}(x) \odot Z_{\mathcal{C}_x}(x) \tag{11}$$

where $\alpha = 1$ by default. Both methods result in robust anomaly scores. The results are reported in Section VI-C.

D. Identifying Anomalous Sequences

Finding anomalous sequences with locally adaptive thresholding: Once we obtain anomaly scores at every time step, thresholding techniques can be applied to identify anomalous sequences. We use sliding windows to compute thresholds, and empirically set the window size as $\frac{T}{3}$ and the step size as $\frac{T}{3*10}$. This is helpful to identify contextual anomalies whose contextual information is usually unknown. The sliding window size determines the number of historical anomaly scores to evaluate the current threshold. For each sliding window, we use a simple static threshold defined as 4 standard deviations from the mean of the window. We can then identify those points whose anomaly score is larger than the threshold as anomalous. Thus, continuous time points compose into anomalous sequences (or windows): $\{\boldsymbol{a}_{seq}^i, i = 1, 2, \ldots, K\}$, where $\boldsymbol{a}_{seq}^i = (\boldsymbol{a}_{start(i)}, \ldots, \boldsymbol{a}_{end(i)})$.

Mitigating false positives: The use of sliding windows can increase recall of anomalies but may also produce many false positives. We employ an anomaly pruning approach inspired by Hundman et al. [6] to mitigate false positives. At first, for each anomalous sequence, we use the maximum anomaly score to represent it, obtaining a set of maximum values $\{\boldsymbol{a}_{max}^i, i = 1, 2, \ldots, K\}$. Once these values are sorted in descending order, we can compute the decrease percent $p^i = (\boldsymbol{a}_{max}^{i-1} - \boldsymbol{a}_{max}^i)/\boldsymbol{a}_{max}^{i-1}$. When the first p^i does not exceed a certain threshold θ (by default $\theta = 0.1$), we reclassify all subsequent sequences (i.e., $\{\boldsymbol{a}_{seq}^j, i \leq j \leq K\}$) as normal.

VI. EXPERIMENTAL RESULTS

A. Datasets

To measure the performance of TadGAN, we evaluate it on multiple time series datasets. In total, we have collected 11 datasets (a total of 492 signals) across a variety of application domains. We use **spacecraft telemetry** signals provided by NASA[2], consisting of two datasets: Mars Science Laboratory (MSL) and Soil Moisture Active Passive (SMAP). In addition, we use **Yahoo S5** which contains four different sub-datasets [3] The A1 dataset is based on real production traffic to Yahoo computing systems, while A2, A3 and A4 are all synthetic datasets. Lastly, we use **Numenta Anomaly Benchmark**

[2]Spacecraft telemetry data: https://s3-us-west-2.amazonaws.com/telemanom/data.zip

[3]Yahoo S5 data can be requested here: https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70

(NAB). NAB [43] includes multiple types of time series data from various application domains[4]. We have picked five datasets: Art, AdEx, AWS, Traf, and Tweets.

Datasets from different sources contain different numbers of signals and anomalies, and locations of anomalies are known for each signal. Basic information for each dataset is summarized in Table III. For each dataset, we present the total number of signals and the number of anomalies pertaining to them. We also observe whether the anomalies in the dataset are single "point" anomalies, or one or more collections. In order to suss out the ease of anomaly identification, we measure how out-of-the-ordinary each anomaly point is by categorizing it as "*out-of-dist*" if it falls 4 standard deviations away from the mean of all the data for a signal. As each dataset has some quality that make detecting its anomalies more challenging, this diverse selection will help us identify the effectiveness and limitations of each baseline.

B. Experimental setup

1) Data preparation: For each dataset, we first normalize the data betweeen $[-1, 1]$. Then we find a proper interval over which to aggregate the data, such that we have several thousands of equally spaced points in time for each signal. We then set a window size $t = 100$ and step size $s = 1$ to obtain training samples for TadGAN. Because many signals in the Yahoo datasets contain linear trends, we apply a simple detrending function (which subtracts the result of a linear least-squares fit to the signal) before training and testing.

2) Architecture: In our experiments, inputs to TadGAN are time series sequences of length 100 (domain X), and the latent space (domain Z) is 20-dimensional. We use a 1-layer bidirectional Long Short-Term Memory (LSTM) with 100 hidden units as *Generator* $\mathcal{E}$, and a 2-layer bidirectional LSTM with 64 hidden units each as *Generator* $\mathcal{G}$, where dropout is applied. We add a 1-D convolutional layer for both *Critics*, with the intention of capturing local temporal features that can determine how anomalous a sequence is. The model is trained on a specific signal from one dataset for 2000 iterations, with a batch size of 64.

3) Evaluation metrics: We measure the performance of different methods using the commonly used metrics Precision, Recall and F1-Score. In many real-world application scenarios, anomalies are rare and usually window-based (i.e. a continuous sequence of points—see Sec. V-D). From the perspective of end-users, the best outcome is to receive timely true alarms without too many false positives (FPs), as these may waste time and resources. To penalize high FPs and reward the timely true alarms, we present the following window-based rules: (1) If a known anomalous window overlaps any predicted windows, a **TP** is recorded. (2) If a known anomalous window does not overlap any predicted windows, a **FN** is recorded. (3) If a predicted window does not overlap any labeled anomalous region, a **FP** is recorded. This method is also used in Hundman et al's work [6].

[4]NAB data: https://github.com/numenta/NAB/tree/master/data

4) Baselines: The baseline methods can be divided into three categories: prediction-based methods, reconstruction-based methods, and online commercial tools.

ARIMA (Prediction-based). An autoregressive integrated moving average (ARIMA) model is a popular statistical analysis model that learns autocorrelations in the time series for future value prediction. We use point-wise prediction errors as the anomaly scores to detect anomalies.

HTM (Prediction-based). Hierarchial Temporal Memory (HTM) [2] has shown better performance over many statistical analysis models in the Numenta Anomaly Benchmark. It encodes the current input to a hidden state and predicts the next hidden state. Prediction errors are computed as the differences between the predicted state and the true state, which are then used as the anomaly scores for anomaly detection.

LSTM (Prediction-based). The neural network used in our experiments consists of two LSTM layers with 80 units each, and a subsequent dense layer with one unit which predicts the value at the next time step (similar to the one used by Hundman et al. [6]). Point-wise prediction errors are used for anomaly detection.

AutoEncoder (Reconstruction-based). Our approach can be viewed as a special instance of "adversarial autoencoders" [44], $\mathcal{E} \circ \mathcal{G} : X \to X$. Thus, we compare our method with standard autoencoders with dense layers or LSTM layers [5]. The dense autoencoder consists of three dense layers with 60, 20 and 60 units respectively. The LSTM autoencoder contains two LSTM layers, each with 60 units. Again, a point-wise reconstruction error is used to detect anomalies.

MAD-GAN (Reconstruction-based). This method [7] uses a vanilla GAN along with an optimal instance searching strategy in latent space to support multivariate time series reconstruction. We use MAD-GAN to compute the anomaly scores at every time step and then apply the same anomaly detection method introduced in Sec. V-D to find anomalies.

Microsoft Azure Anomaly Detector (Commercial tool). Microsoft uses Spectral Residual Convolutional Neural Networks (SR-CNN) in which the models are applied serially [8]. The SR model is responsible for saliency detection, and the CNN is responsible for learning a discriminating threshold. The output of the model is a sequence of binary labels that is attributed to each timestamp.

Amazon DeepAR (Commercial tool). DeepAR is a probabilistic forecasting model with autoregressive recurrent networks [9]. We use this model in a similar manner to LSTM in that it is a prediction-based approach. Anomaly scores are presented as the regression errors which are computed as the distance between the median of the predicted value and true value.

C. Benchmarking Results

TadGAN outperformed all the baseline methods by having the highest averaged F1 score (0.7) across all the datasets. Table IV ranks all the methods based on their averaged F1 scores (the last column) across the eleven datasets. The second (LSTM, 0.623) and the third (Arima, 0.599) best

	NASA		Yahoo S5				NAB					
Baseline	MSL	SMAP	A1	A2	A3	A4	Art	AdEx	AWS	Traf	Tweets	Mean±SD
TadGAN	**0.623**	**0.704**	**0.8**	0.867	0.685	0.6	**0.8**	**0.8**	0.644	0.486	**0.609**	**0.700±0.123**
(P) LSTM	0.46	0.69	0.744	**0.98**	0.772	0.645	0.375	0.538	0.474	**0.634**	0.543	0.623±0.163
(P) Arima	0.492	0.42	0.726	0.836	**0.815**	**0.703**	0.353	0.583	0.518	0.571	0.567	0.599±0.148
(C) DeepAR	0.583	0.453	0.532	0.929	0.467	0.454	0.545	0.615	0.39	0.6	0.542	0.555±0.130
(R) LSTM AE	0.507	0.672	0.608	0.871	0.248	0.163	0.545	0.571	**0.764**	0.552	0.542	0.549±0.193
(P) HTM	0.412	0.557	0.588	0.662	0.325	0.287	0.455	0.519	0.571	0.474	0.526	0.489±0.108
(R) Dense AE	0.507	0.7	0.472	0.294	0.074	0.09	0.444	0.267	0.64	0.333	0.057	0.353±0.212
(R) MAD-GAN	0.111	0.128	0.37	0.439	0.589	0.464	0.324	0.297	0.273	0.412	0.444	0.35±0.137
(C) MS Azure	0.218	0.118	0.352	0.612	0.257	0.204	0.125	0.066	0.173	0.166	0.118	0.219±0.145

TABLE IV
F1-SCORES OF BASELINE MODELS USING WINDOW-BASED RULES. COLOR ENCODES THE PERFORMANCE OF THE F1 SCORE. ONE IS EVENLY DIVIDED INTO 10 BINS, WITH EACH BIN ASSOCIATED WITH ONE COLOR. FROM DARK RED TO DARK BLUE, F1 SCORE INCREASES FROM 0 TO 1.

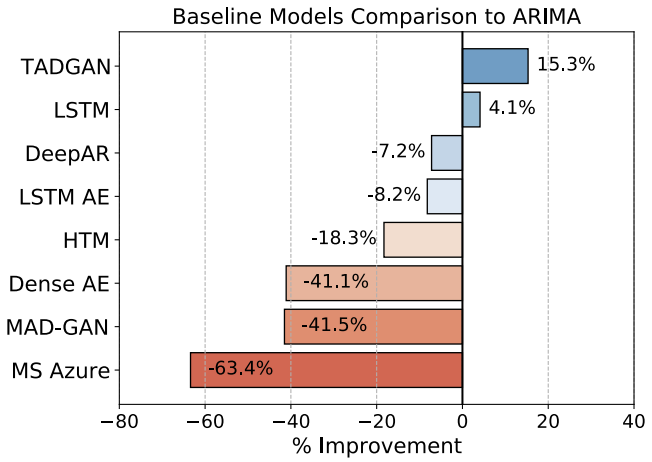

Fig. 3. Comparing average F1-Scores of baseline models across all datasets to ARIMA. The x-axis represents the percentage of improvement over the ARIMA score by each one of the baseline models.

are both prediction-based methods and TadGAN outperformed them by 12.46% and 16.86%, respectively, compared to the averaged F1 score.

Baseline models in comparison to Arima. Figure 3 depicts the performance of all baseline models with respect to Arima. It shows how much improvement in F1-Score is gained by each model. The F1-Score presented is the average across the eleven datasets. TadGAN achieves the best overall improvement with an over 15% improvement in score, followed by LSTM with a little over 4% improvement. It's worth noting that all the remaining models struggle to beat Arima.

Synthetic data v.s. real-world datasets. Although TadGAN outperforms all baselines on average, we note that it ranks below Arima when detecting anomalies within synthetic dataset with point anomalies. Specifically, TadGAN achieved an average of 0.717 while Arima scored an average of 0.784. However, TadGAN still produces competitive results in both scenarios.

How well do AutoEncoders perform? To view the superiority of GAN, we compare it to other reconstruction-based method such as LSTM AE, and Dense AE. One striking result is that the autoencoder alone does not perform well on point anomalies. We observe this as LSTM, AE, and Dense AE obtained an average F1 Score on A3 and A4 of 0.205 and 0.082 respectively, while TadGAN and MAD-GAN achieved a higher score of 0.643 and 0.527 respectively. One potential reason could be that AutoEncoders are optimizing L2 function and strictly attempt to fit the data, resulting in that anomalies get fitted as well. However, adversarial learning does not have this type of issue.

TadGAN v.s. MadGAN. Overall, TadGAN (0.7) outperformed Mad-GAN (0.219) significantly. This fully demonstrates the usage of forward cycle-consistency loss (Eq. 5) which prevents the contradiction between two *Generators* $\mathcal{E}$ and $\mathcal{G}$ and paves the most direct way to the optimal z_i that corresponds to the testing sample x_i. Mad-GAN uses only vanilla GAN and does not include any regularization mechanisms to guarantee the mapping route $x_i \rightarrow z_i \rightarrow \hat{x}_i$. Their approach to finding the optimal z_i is that they first sample a random z from the latent space and then optimize it with the gradient descent algorithm by optimizing the anomaly detection loss.

D. Ablation Study

We evaluated multiple variations of TadGAN, using different anomaly score computation methods for each (Sec. V-C). The results are summarized in Table V. Here we report some noteworthy insights.

Using *Critic* alone is unstable, because it has the lowest average F1 score (0.29) and the highest standard deviation (0.237). While only using *Critic* can achieve a good performance in some datasets, such as SMAP and Art, its performance may also be unexpectedly bad, such as in A2, A3, A4, AdEx, and Traf. No clear shared characteristics are identified among these five datasets (see Table III). For example, some datasets contain only collective anomalies (Traf, AdEx), while other datasets, like A3 and A4, have point anomalies as the majority types. One explanation could be that *Critic*'s behavior

	NASA		Yahoo S5				NAB					
Variation	MSL	SMAP	A1	A2	A3	A4	Art	AdEx	AWS	Traf	Tweets	Mean+SD
Critic	0.393	0.672	0.285	0.118	0.008	0.024	0.625	0	0.35	0.167	0.548	0.290±0.237
Point	0.585	0.588	0.674	0.758	0.628	**0.6**	0.588	0.611	0.551	0.383	0.571	0.594±0.086
Area	0.525	0.655	0.681	0.82	0.567	0.523	0.625	0.645	0.59	0.435	0.559	0.602±0.096
DTW	0.514	0.581	0.697	0.794	0.613	0.547	0.714	0.69	0.633	0.455	0.559	0.618±0.095
Critic×Point	0.619	0.675	0.703	0.75	**0.685**	0.536	0.588	0.579	0.576	0.4	0.59	0.609±0.091
Critic+Point	0.529	0.653	**0.8**	0.78	0.571	0.44	0.625	0.595	**0.644**	0.439	0.592	0.606±0.111
Critic×Area	0.578	**0.704**	0.719	**0.867**	0.587	0.46	**0.8**	0.6	0.6	0.4	0.571	0.625±0.131
Critic+Area	0.493	0.692	0.789	0.847	0.483	0.367	0.75	0.75	0.607	0.474	0.6	0.623±0.148
Critic×DTW	**0.623**	0.68	0.667	0.82	0.631	0.497	0.667	0.667	0.61	0.455	0.605	**0.629±0.091**
Critic+DTW	0.462	0.658	0.735	0.857	0.523	0.388	0.667	**0.8**	0.632	**0.486**	**0.609**	0.620±0.139
Mean	0.532	0.655	0.675	0.741	0.529	0.438	0.664	0.593	0.579	0.409	0.580	
SD	0.068	0.039	0.137	0.211	0.182	0.154	0.067	0.209	0.081	0.087	0.02	

TABLE V
F1-SCORES OF ALL THE VARIATIONS OF OUR MODEL.

is unpredictable when confronted with anomalies ($x \nsim \mathbb{P}_X$), because it is only taught to distinguish real time segments ($x \sim \mathbb{P}_X$) from generated ones.

DTW outperforms the other two reconstruction error types slightly. Among all variations, Critic×DTW has the best score (0.629). Further, its standard deviation is smaller than most of the other variations except for Point, indicating that this combination is more stable than others. Therefore, this combination should be the safe choice when encountering new datasets without labels.

Combining *Critic* outputs and reconstruction errors does improve performance in most cases. In all datasets except A4, combinations achieve the best performance. Let us take the MSL dataset as an example. We observe that when using DTW alone, the F1 score is 0.514. Combining this with the *Critic score*, we obtain a score of 0.623, despite the fact that the F1 score when using *Critic* alone is 0.393. In addition, we find that after combining the Critic scores, the averaged F1 score improves for each of the individual reconstruction error computation methods. However, one interesting pattern is that for dataset A4, which consists mostly of point anomalies, using only point-wise errors achieve the best performance.

Multiplication is a better option than convex combination. Multiplication consistently leads to a higher averaged F1 score than convex combination does when using the same reconstruction error type (e.g., *Critic*×Point v.s. *Critic*+Point). Multiplication also has consistently smaller standard deviations. Thus, multiplication is the recommended way to combine reconstruction scores and *Critic* scores. This can be explained by the fact that multiplication can better amplify high anomaly scores.

E. Limitations and Discussion

Here we compare our approach to one well-known GAN-based anomaly detection method [7]. However, there are many other GAN architectures tailored for time series reconstruction, such as Time-Series GAN [14]. Due to our modular design, any reconstruction-based algorithm of time series can employ our anomaly scoring method for time series anomaly detection. In the future, we plan to investigate various strategies for time series reconstruction and compare their performances to the current state-of-the-art. Moreover, it is worth understanding how better signal reconstruction affects the performance of anomaly detection. In fact, it is expected that better reconstruction might overfit to anomalies. Therefore, further experiments are required to understand the relationship between reconstruction and detecting anomalies.

VII. CONCLUSION

In this paper, we presented a novel framework, TadGAN , that allows for time series reconstruction and effective anomaly detection, showing how GANs can be effectively used for anomaly detection in time series data. We explored point-wise and window-based methods to compute reconstruction errors. We further proposed two different ways to combine reconstruction errors and *Critic* outputs to obtain anomaly scores at every time step. We have also tested several anomaly-scoring techniques and reported the best-suited one in this work. Our experimental results showed that (1) TadGAN outperformed all the baseline methods by having the highest averaged F1 score across all the datasets, and showed superior performance over baseline methods in 6 out of 11 datasets; (2) window-based reconstruction errors outperformed the point-wise method; and (3) the combination of both reconstruction errors and critic outputs offers more robust anomaly scores, which help to reduce the number of false positives as well as increase the number of true positives. Finally, our code is open source and is available as a tool for benchmarking time series datasets for anomaly detection.

VIII. ACKNOWLEDGEMENT

The authors are grateful to SES S.A. of Betzdorf, Luxembourg, for their financial and non financial support in this

work. Dr. Cuesta-Infante is funded by the Spanish Government research fundings RTI2018-098743-B-I00 (MICINN/FEDER) and Y2018/EMT-5062 (Comunidad de Madrid). Alnegheimish is supported by King Abdulaziz City for Science and Technology (KACST).

REFERENCES

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[2] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, 2017.

[3] D. Zheng, F. Li, and T. Zhao, "Self-adaptive statistical process control for anomaly detection in time series," *Expert Systems with Applications*, vol. 57, pp. 324–336, 2016.

[4] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[5] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based encoder-decoder for multi-sensor anomaly detection," in *Anomaly Detection Workshop at 33rd ICML*, 2016.

[6] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding," in *Proc. of the 24th ACM SIGKDD*, 2018.

[7] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng, "Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks," in *International Conference on Artificial Neural Networks*. Springer, 2019, pp. 703–716.

[8] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-series anomaly detection service at microsoft," in *Proc. of the 25th ACM SIGKDD*, 2019, pp. 3009–3017.

[9] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "Deepar: Probabilistic forecasting with autoregressive recurrent networks," *International Journal of Forecasting*, 2019.

[10] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Computing*, vol. 22, pp. 949–961, 2017.

[11] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long Short Term Memory Networks for Anomaly Detection in Time Series," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2015.

[12] J. Goh, S. Adepu, M. Tan, and Z. S. Lee, "Anomaly detection in cyber physical systems using recurrent neural networks," in *IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, 2017, pp. 140–145.

[13] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Proc. of Advances in neural information processing systems*, 2014, pp. 2672–2680.

[14] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," in *Proc. of Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019, pp. 5509–5519.

[15] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[16] J. Qiu, Q. Du, and C. Qian, "Kpi-tsad: A time-series anomaly detector for kpi monitoring in cloud applications," *Symmetry*, vol. 11, no. 11, p. 1350, 2019.

[17] P. De Chazal, M. O'Dwyer, and R. B. Reilly, "Automatic classification of heartbeats using ecg morphology and heartbeat interval features," *IEEE transactions on biomedical engineering*, vol. 51, no. 7, pp. 1196–1206, 2004.

[18] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004.

[19] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PLOS ONE*, vol. 11, no. 4, pp. 1–31, 04 2016.

[20] R. A. A. Habeeb, F. Nasaruddin, A. Gani, I. A. T. Hashem, E. Ahmed, and M. Imran, "Real-time big data processing for anomaly detection: A survey," *International Journal of Information Management*, vol. 45, pp. 289–307, 2019.

[21] J.-A. Martínez-Heras and A. Donati, "Enhanced Telemetry Monitoring with Novelty Detection," *AI Magazine*, vol. 35, no. 4, p. 37, 2014.

[22] D. Decoste, "Automated Learning and Monitoring of Limit Functions," in *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 1997.

[23] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proc. of the ACM SIGMOD*, 2000, pp. 93–104.

[24] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces," in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2002, pp. 15–27.

[25] Z. He, X. Xu, and S. Deng, "Discovering cluster-based local outliers," *Pattern Recognition Letters*, vol. 24, no. 9-10, pp. 1641–1650, 2003.

[26] E. H. Pena, M. V. de Assis, and M. L. Proença, "Anomaly detection using forecasting methods arima and hwds," in *International Conference of the Chilean Computer Science Society (SCCC)*, 2013, pp. 63–66.

[27] J. M. Torres, P. G. Nieto, L. Alejano, and A. Reyes, "Detection of outliers in gas emissions from urban areas using functional data analysis," *Journal of hazardous materials*, vol. 186, no. 1, pp. 144–149, 2011.

[28] H. Ringberg, A. Soule, J. Rexford, and C. Diot, "Sensitivity of pca for traffic anomaly detection," in *Proc. of the 2007 ACM SIGMETRICS*, 2007, pp. 109–120.

[29] X. Dai and Z. Gao, "From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2226–2238, 2013.

[30] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, 2015.

[31] B. Zhou, S. Liu, B. Hooi, X. Cheng, and J. Ye, "BeatGAN: Anomalous Rhythm Detection using Adversarially Generated Time Series," in *Proc. of the 28th Int. Joint Conf. on Artificial Intelligence, (IJCAI)*, 2019, pp. 4433–4439.

[32] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," in *IEEE Int. Conf. on Computer Vision (ICCV)*, oct 2017, pp. 2242–2251.

[33] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," in *Proc. of Advances in neural information processing systems*, 2016, pp. 613–621.

[34] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, "f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks," *Medical Image Analysis*, vol. 54, pp. 30 – 44, 2019.

[35] L. Deecke, R. Vandermeulen, L. Ruff, S. Mandt, and M. Kloft, "Anomaly detection with generative adversarial networks," 2018. [Online]. Available: https://openreview.net/forum?id=S1EfylZ0Z

[36] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," in *International Conference on Information Processing in Medical Imaging*. Springer, 2017, pp. 146–157.

[37] H. Zenati, M. Romain, C.-S. Foo, B. Lecouat, and V. Chandrasekhar, "Adversarially Learned Anomaly Detection," in *IEEE ICDM*, nov 2018, pp. 727–736.

[38] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial Feature Learning," in *IEEE ICLR*, 2017.

[39] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, "Adversarially Learned Inference," in *IEEE ICLR*, 2017.

[40] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. of the 34th ICML*, 2017, pp. 214–223.

[41] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," in *Proc. of the 31st Int. Conf. on Neural Information Processing Systems*, 2017, pp. 5769–5779.

[42] D. J. Bemdt and J. Clifford, "Using Dynamic Time Warping to Find Patterns in Time Series," in *AAAI Workshop on Knowledge Discovery in Databases*, Seattle, Washington, 1994.

[43] A. Lavin and S. Ahmad, "Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark," in *Proc. of IEEE ICMLA*, 2015, pp. 38–44.

[44] A. Makhzani, J. Shlens, N. Jaitly, and I. Goodfellow, "Adversarial autoencoders," in *Proc. of ICLR, Workshop Track*, 2016.

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

Jun-Yan Zhu* Taesung Park* Phillip Isola Alexei A. Efros
Berkeley AI Research (BAIR) laboratory, UC Berkeley

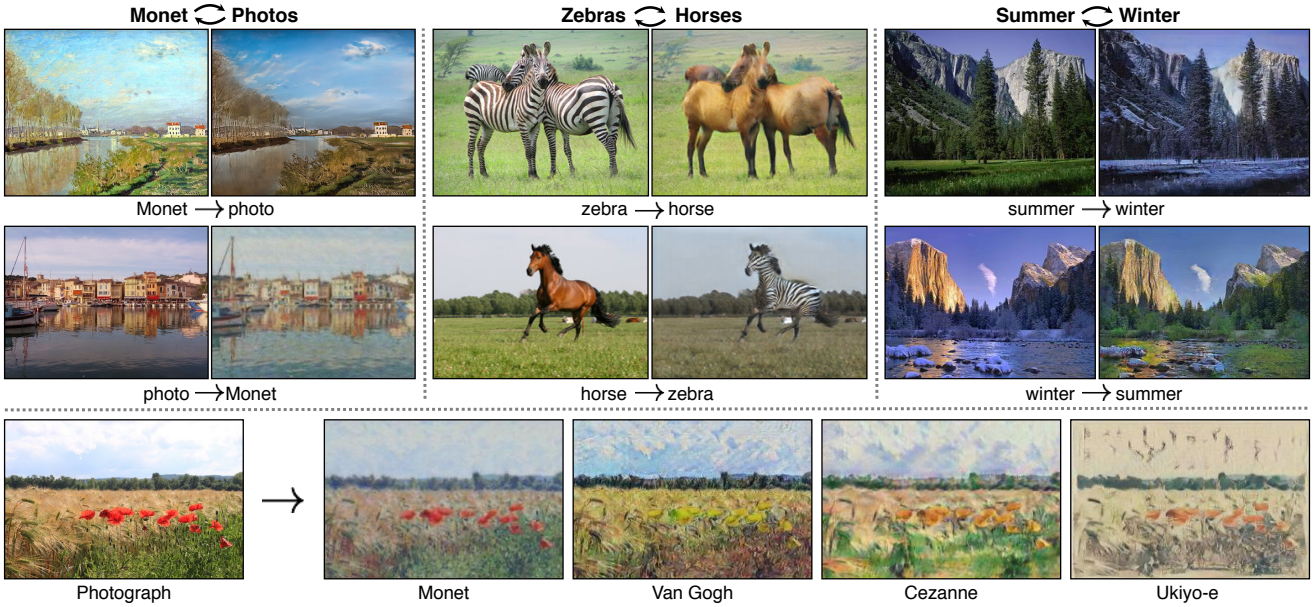

Figure 1: Given any two unordered image collections X and Y, our algorithm learns to automatically "translate" an image from one into the other and vice versa: *(left)* Monet paintings and landscape photos from Flickr; *(center)* zebras and horses from ImageNet; *(right)* summer and winter Yosemite photos from Flickr. Example application *(bottom)*: using a collection of paintings of famous artists, our method learns to render natural photographs into the respective styles.

Abstract

Image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. However, for many tasks, paired training data will not be available. We present an approach for learning to translate an image from a source domain X to a target domain Y in the absence of paired examples. Our goal is to learn a mapping $G : X \rightarrow Y$ such that the distribution of images from $G(X)$ is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained, we couple it with an inverse mapping $F : Y \rightarrow X$ and introduce a cycle consistency loss *to enforce $F(G(X)) \approx X$ (and vice versa). Qualitative results are presented on several tasks where paired training data does not exist, including collection style transfer, object transfiguration, season transfer, photo enhancement, etc. Quantitative comparisons against several prior methods demonstrate the superiority of our approach.*

1. Introduction

What did Claude Monet see as he placed his easel by the bank of the Seine near Argenteuil on a lovely spring day in 1873 (Figure 1, top-left)? A color photograph, had it been invented, may have documented a crisp blue sky and a glassy river reflecting it. Monet conveyed his *impression* of this same scene through wispy brush strokes and a bright palette.

What if Monet had happened upon the little harbor in Cassis on a cool summer evening (Figure 1, bottom-left)? A brief stroll through a gallery of Monet paintings makes it possible to imagine how he would have rendered the scene: perhaps in pastel shades, with abrupt dabs of paint, and a somewhat flattened dynamic range.

We can imagine all this despite never having seen a side by side example of a Monet painting next to a photo of the scene he painted. Instead, we have knowledge of the set of Monet paintings and of the set of landscape photographs. We can reason about the stylistic differences between these

* indicates equal contribution

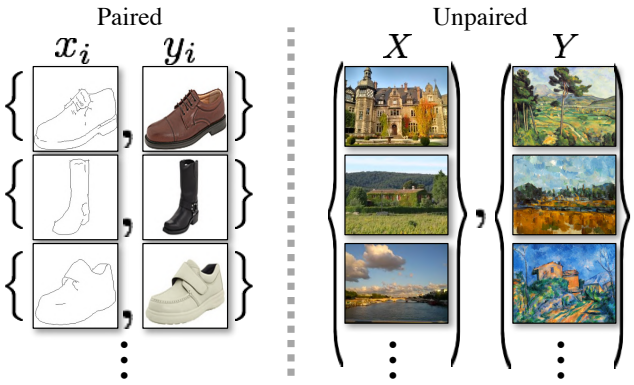

Figure 2: *Paired* training data (left) consists of training examples $\{x_i, y_i\}_{i=1}^N$, where the correspondence between x_i and y_i exists [22]. We instead consider *unpaired* training data (right), consisting of a source set $\{x_i\}_{i=1}^N$ ($x_i \in X$) and a target set $\{y_j\}_{j=1}^M$ ($y_j \in Y$), with no information provided as to which x_i matches which y_j.

two sets, and thereby imagine what a scene might look like if we were to "translate" it from one set into the other.

In this paper, we present a method that can learn to do the same: capturing special characteristics of one image collection and figuring out how these characteristics could be translated into the other image collection, all in the absence of any paired training examples.

This problem can be more broadly described as image-to-image translation [22], converting an image from one representation of a given scene, x, to another, y, e.g., grayscale to color, image to semantic labels, edge-map to photograph. Years of research in computer vision, image processing, computational photography, and graphics have produced powerful translation systems in the supervised setting, where example image pairs $\{x_i, y_i\}_{i=1}^N$ are available (Figure 2, left), e.g., [11, 19, 22, 23, 28, 33, 45, 56, 58, 62]. However, obtaining paired training data can be difficult and expensive. For example, only a couple of datasets exist for tasks like semantic segmentation (e.g., [4]), and they are relatively small. Obtaining input-output pairs for graphics tasks like artistic stylization can be even more difficult since the desired output is highly complex, typically requiring artistic authoring. For many tasks, like object transfiguration (e.g., zebra↔horse, Figure 1 top-middle), the desired output is not even well-defined.

We therefore seek an algorithm that can learn to translate between domains without paired input-output examples (Figure 2, right). We assume there is some underlying relationship between the domains – for example, that they are two different renderings of the same underlying scene – and seek to learn that relationship. Although we lack supervision in the form of paired examples, we can exploit supervision at the level of sets: we are given one set of images in domain X and a different set in domain Y. We may train a mapping $G : X \rightarrow Y$ such that the output $\hat{y} = G(x)$, $x \in X$, is indistinguishable from images $y \in Y$ by an adversary trained to classify $\hat{y}$ apart from y. In theory, this objective can induce an output distribution over $\hat{y}$ that matches the empirical distribution $p_{data}(y)$ (in general, this requires G to be stochastic) [16]. The optimal G thereby translates the domain X to a domain $\hat{Y}$ distributed identically to Y. However, such a translation does not guarantee that an individual input x and output y are paired up in a meaningful way – there are infinitely many mappings G that will induce the same distribution over $\hat{y}$. Moreover, in practice, we have found it difficult to optimize the adversarial objective in isolation: standard procedures often lead to the well-known problem of mode collapse, where all input images map to the same output image and the optimization fails to make progress [15].

These issues call for adding more structure to our objective. Therefore, we exploit the property that translation should be "cycle consistent", in the sense that if we translate, e.g., a sentence from English to French, and then translate it back from French to English, we should arrive back at the original sentence [3]. Mathematically, if we have a translator $G : X \rightarrow Y$ and another translator $F : Y \rightarrow X$, then G and F should be inverses of each other, and both mappings should be bijections. We apply this structural assumption by training both the mapping G and F simultaneously, and adding a *cycle consistency loss* [64] that encourages $F(G(x)) \approx x$ and $G(F(y)) \approx y$. Combining this loss with adversarial losses on domains X and Y yields our full objective for unpaired image-to-image translation.

We apply our method to a wide range of applications, including collection style transfer, object transfiguration, season transfer and photo enhancement. We also compare against previous approaches that rely either on hand-defined factorizations of style and content, or on shared embedding functions, and show that our method outperforms these baselines. We provide both PyTorch and Torch implementations. Check out more results at our website.

2. Related work

Generative Adversarial Networks (GANs) [16, 63] have achieved impressive results in image generation [6, 39], image editing [66], and representation learning [39, 43, 37]. Recent methods adopt the same idea for conditional image generation applications, such as text2image [41], image inpainting [38], and future prediction [36], as well as to other domains like videos [54] and 3D data [57]. The key to GANs' success is the idea of an *adversarial loss* that forces the generated images to be, in principle, indistinguishable from real photos. This loss is particularly powerful for image generation tasks, as this is exactly the objective that much of computer graphics aims to optimize. We adopt an adversarial loss to learn the mapping such that the translated

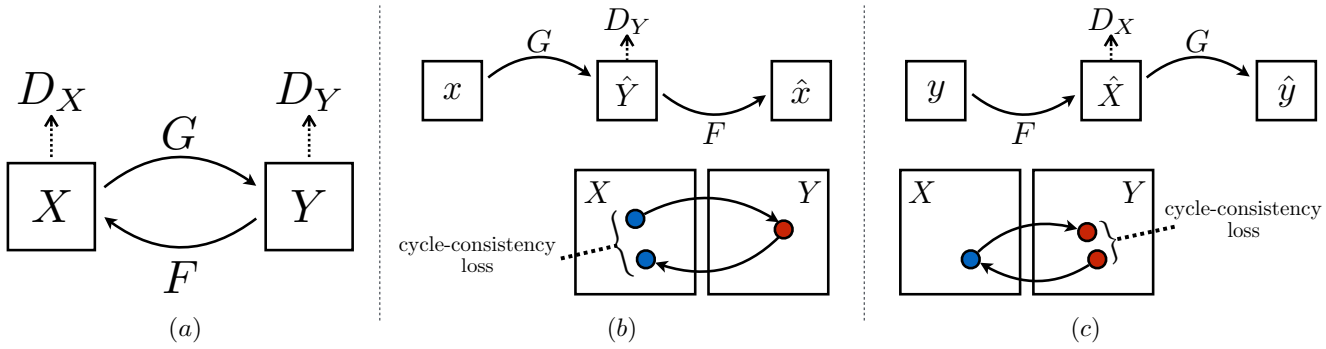

Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

images cannot be distinguished from images in the target domain.

Image-to-Image Translation The idea of image-to-image translation goes back at least to Hertzmann et al.'s Image Analogies [19], who employ a non-parametric texture model [10] on a single input-output training image pair. More recent approaches use a *dataset* of input-output examples to learn a parametric translation function using CNNs (e.g., [33]). Our approach builds on the "pix2pix" framework of Isola et al. [22], which uses a conditional generative adversarial network [16] to learn a mapping from input to output images. Similar ideas have been applied to various tasks such as generating photographs from sketches [44] or from attribute and semantic layouts [25]. However, unlike the above prior work, we learn the mapping without paired training examples.

Unpaired Image-to-Image Translation Several other methods also tackle the unpaired setting, where the goal is to relate two data domains: X and Y. Rosales et al. [42] propose a Bayesian framework that includes a prior based on a patch-based Markov random field computed from a source image and a likelihood term obtained from multiple style images. More recently, CoGAN [32] and cross-modal scene networks [1] use a weight-sharing strategy to learn a common representation across domains. Concurrent to our method, Liu et al. [31] extends the above framework with a combination of variational autoencoders [27] and generative adversarial networks [16]. Another line of concurrent work [46, 49, 2] encourages the input and output to share specific "content" features even though they may differ in "style". These methods also use adversarial networks, with additional terms to enforce the output to be close to the input in a predefined metric space, such as class label space [2], image pixel space [46], and image feature space [49].

Unlike the above approaches, our formulation does not rely on any task-specific, predefined similarity function between the input and output, nor do we assume that the input and output have to lie in the same low-dimensional embedding space. This makes our method a general-purpose solution for many vision and graphics tasks. We directly compare against several prior and contemporary approaches in Section 5.1.

Cycle Consistency The idea of using transitivity as a way to regularize structured data has a long history. In visual tracking, enforcing simple forward-backward consistency has been a standard trick for decades [24, 48]. In the language domain, verifying and improving translations via "back translation and reconciliation" is a technique used by human translators [3] (including, humorously, by Mark Twain [51]), as well as by machines [17]. More recently, higher-order cycle consistency has been used in structure from motion [61], 3D shape matching [21], cosegmentation [55], dense semantic alignment [65, 64], and depth estimation [14]. Of these, Zhou et al. [64] and Godard et al. [14] are most similar to our work, as they use a *cycle consistency loss* as a way of using transitivity to supervise CNN training. In this work, we are introducing a similar loss to push G and F to be consistent with each other. Concurrent with our work, in these same proceedings, Yi et al. [59] independently use a similar objective for unpaired image-to-image translation, inspired by dual learning in machine translation [17].

Neural Style Transfer [13, 23, 52, 12] is another way to perform image-to-image translation, which synthesizes a novel image by combining the content of one image with the style of another image (typically a painting) based on matching the Gram matrix statistics of pre-trained deep features. Our primary focus, on the other hand, is learning the mapping between two image collections, rather than between two specific images, by trying to capture correspondences between higher-level appearance structures. Therefore, our method can be applied to other tasks, such as

painting→ photo, object transfiguration, etc. where single sample transfer methods do not perform well. We compare these two methods in Section 5.2.

3. Formulation

Our goal is to learn mapping functions between two domains X and Y given training samples $\{x_i\}_{i=1}^N$ where $x_i \in X$ and $\{y_j\}_{j=1}^M$ where $y_j \in Y$[1]. We denote the data distribution as $x \sim p_{data}(x)$ and $y \sim p_{data}(y)$. As illustrated in Figure 3 (a), our model includes two mappings $G : X \rightarrow Y$ and $F : Y \rightarrow X$. In addition, we introduce two adversarial discriminators D_X and D_Y, where D_X aims to distinguish between images $\{x\}$ and translated images $\{F(y)\}$; in the same way, D_Y aims to discriminate between $\{y\}$ and $\{G(x)\}$. Our objective contains two types of terms: *adversarial losses* [16] for matching the distribution of generated images to the data distribution in the target domain; and *cycle consistency losses* to prevent the learned mappings G and F from contradicting each other.

3.1. Adversarial Loss

We apply adversarial losses [16] to both mapping functions. For the mapping function $G : X \rightarrow Y$ and its discriminator D_Y, we express the objective as:

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

where G tries to generate images $G(x)$ that look similar to images from domain Y, while D_Y aims to distinguish between translated samples $G(x)$ and real samples y. G aims to minimize this objective against an adversary D that tries to maximize it, i.e., $\min_G \max_{D_Y} \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y)$. We introduce a similar adversarial loss for the mapping function $F : Y \rightarrow X$ and its discriminator D_X as well: i.e., $\min_F \max_{D_X} \mathcal{L}_{\text{GAN}}(F, D_X, Y, X)$.

3.2. Cycle Consistency Loss

Adversarial training can, in theory, learn mappings G and F that produce outputs identically distributed as target domains Y and X respectively (strictly speaking, this requires G and F to be stochastic functions) [15]. However, with large enough capacity, a network can map the same set of input images to any random permutation of images in the target domain, where any of the learned mappings can induce an output distribution that matches the target distribution. Thus, adversarial losses alone cannot guarantee that the learned function can map an individual input x_i to a desired output y_i. To further reduce the space of possible mapping functions, we argue that the learned mapping functions should be cycle-consistent: as shown in Figure 3 (b), for each image x from domain X, the image translation cycle should be able to bring x back to the original image, i.e., $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$. We call this *forward cycle consistency*. Similarly, as illustrated in Figure 3 (c), for each image y from domain Y, G and F should also satisfy *backward cycle consistency*: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$. We incentivize this behavior using a *cycle consistency loss*:

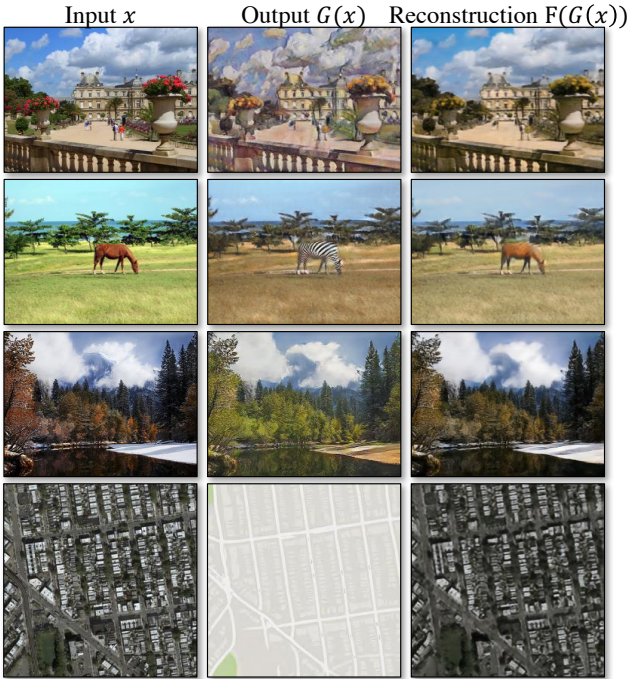

Figure 4: The input images x, output images $G(x)$ and the reconstructed images $F(G(x))$ from various experiments. From top to bottom: photo↔Cezanne, horses↔zebras, winter→summer Yosemite, aerial photos↔Google maps.

$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G, F) = &\mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] \\ &+ \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].\end{aligned} \tag{2}$$

In preliminary experiments, we also tried replacing the L1 norm in this loss with an adversarial loss between $F(G(x))$ and x, and between $G(F(y))$ and y, but did not observe improved performance.

The behavior induced by the cycle consistency loss can be observed in Figure 4: the reconstructed images $F(G(x))$ end up matching closely to the input images x.

3.3. Full Objective

Our full objective is:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = &\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ &+ \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ &+ \lambda \mathcal{L}_{\text{cyc}}(G, F),\end{aligned} \tag{3}$$

[1] We often omit the subscript i and j for simplicity.

where λ controls the relative importance of the two objectives. We aim to solve:

$$G^*, F^* = \arg\min_{G,F} \max_{D_x, D_Y} \mathcal{L}(G, F, D_X, D_Y). \quad (4)$$

Notice that our model can be viewed as training two "autoencoders" [20]: we learn one autoencoder $F \circ G : X \rightarrow X$ jointly with another $G \circ F : Y \rightarrow Y$. However, these autoencoders each have special internal structures: they map an image to itself via an intermediate representation that is a translation of the image into another domain. Such a setup can also be seen as a special case of "adversarial autoencoders" [34], which use an adversarial loss to train the bottleneck layer of an autoencoder to match an arbitrary target distribution. In our case, the target distribution for the $X \rightarrow X$ autoencoder is that of the domain Y.

In Section 5.1.4, we compare our method against ablations of the full objective, including the adversarial loss $\mathcal{L}_{\text{GAN}}$ alone and the cycle consistency loss $\mathcal{L}_{\text{cyc}}$ alone, and empirically show that both objectives play critical roles in arriving at high-quality results. We also evaluate our method with only cycle loss in one direction and show that a single cycle is not sufficient to regularize the training for this under-constrained problem.

4. Implementation

Network Architecture We adopt the architecture for our generative networks from Johnson et al. [23] who have shown impressive results for neural style transfer and super-resolution. This network contains three convolutions, several residual blocks [18], two fractionally-strided convolutions with stride $\frac{1}{2}$, and one convolution that maps features to RGB. We use 6 blocks for 128×128 images and 9 blocks for 256×256 and higher-resolution training images. Similar to Johnson et al. [23], we use instance normalization [53]. For the discriminator networks we use 70×70 PatchGANs [22, 30, 29], which aim to classify whether 70×70 overlapping image patches are real or fake. Such a patch-level discriminator architecture has fewer parameters than a full-image discriminator and can work on arbitrarily-sized images in a fully convolutional fashion [22].

Training details We apply two techniques from recent works to stabilize our model training procedure. First, for $\mathcal{L}_{\text{GAN}}$ (Equation 1), we replace the negative log likelihood objective by a least-squares loss [35]. This loss is more stable during training and generates higher quality results. In particular, for a GAN loss $\mathcal{L}_{\text{GAN}}(G, D, X, Y)$, we train the G to minimize $\mathbb{E}_{x \sim p_{\text{data}}(x)}[(D(G(x)) - 1)^2]$ and train the D to minimize $\mathbb{E}_{y \sim p_{\text{data}}(y)}[(D(y) - 1)^2] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[D(G(x))^2]$.

Second, to reduce model oscillation [15], we follow Shrivastava et al.'s strategy [46] and update the discriminators using a history of generated images rather than the ones produced by the latest generators. We keep an image buffer that stores the 50 previously created images.

For all the experiments, we set $\lambda = 10$ in Equation 3. We use the Adam solver [26] with a batch size of 1. All networks were trained from scratch with a learning rate of 0.0002. We keep the same learning rate for the first 100 epochs and linearly decay the rate to zero over the next 100 epochs. Please see the appendix (Section 7) for more details about the datasets, architectures, and training procedures.

5. Results

We first compare our approach against recent methods for unpaired image-to-image translation on paired datasets where ground truth input-output pairs are available for evaluation. We then study the importance of both the adversarial loss and the cycle consistency loss and compare our full method against several variants. Finally, we demonstrate the generality of our algorithm on a wide range of applications where paired data does not exist. For brevity, we refer to our method as `CycleGAN`. The PyTorch and Torch code, models, and full results can be found at our website.

5.1. Evaluation

Using the same evaluation datasets and metrics as "pix2pix" [22], we compare our method against several baselines both qualitatively and quantitatively. The tasks include semantic labels↔photo on the Cityscapes dataset [4], and map↔aerial photo on data scraped from Google Maps. We also perform ablation study on the full loss function.

5.1.1 Evaluation Metrics

AMT perceptual studies On the map↔aerial photo task, we run "real vs fake" perceptual studies on Amazon Mechanical Turk (AMT) to assess the realism of our outputs. We follow the same perceptual study protocol from Isola et al. [22], except we only gather data from 25 participants per algorithm we tested. Participants were shown a sequence of pairs of images, one a real photo or map and one fake (generated by our algorithm or a baseline), and asked to click on the image they thought was real. The first 10 trials of each session were practice and feedback was given as to whether the participant's response was correct or incorrect. The remaining 40 trials were used to assess the rate at which each algorithm fooled participants. Each session only tested a single algorithm, and participants were only allowed to complete a single session. The numbers we report here are not directly comparable to those in [22] as our ground truth images were processed slightly differently [2] and the participant pool we tested may be differently dis-

[2] We train all the models on 256×256 images while in pix2pix [22], the model was trained on 256×256 patches of 512×512 images, and

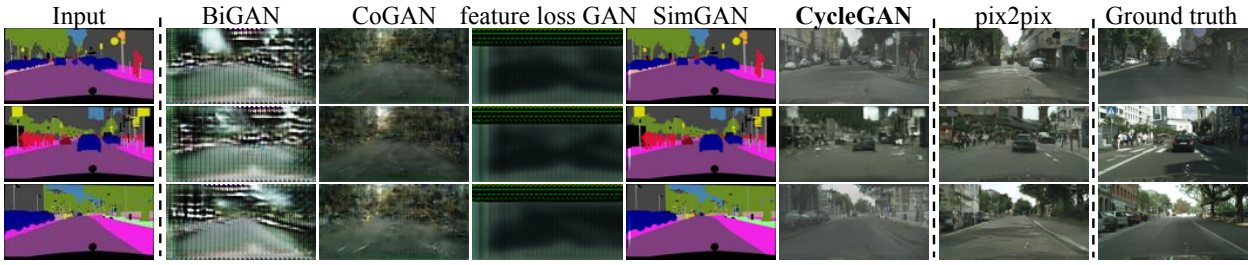

Figure 5: Different methods for mapping labels↔photos trained on Cityscapes images. From left to right: input, BiGAN/ALI [7, 9], CoGAN [32], feature loss + GAN, SimGAN [46], CycleGAN (ours), pix2pix [22] trained on paired data, and ground truth.

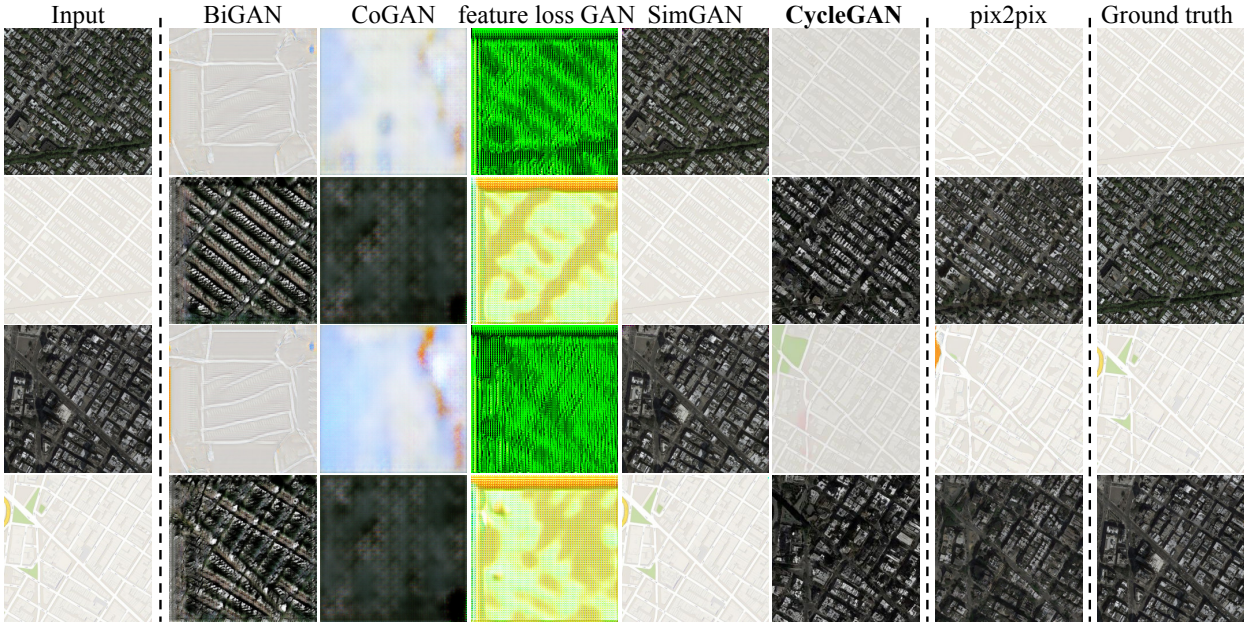

Figure 6: Different methods for mapping aerial photos↔maps on Google Maps. From left to right: input, BiGAN/ALI [7, 9], CoGAN [32], feature loss + GAN, SimGAN [46], CycleGAN (ours), pix2pix [22] trained on paired data, and ground truth.

tributed from those tested in [22] (due to running the experiment at a different date and time). Therefore, our numbers should only be used to compare our current method against the baselines (which were run under identical conditions), rather than against [22].

FCN score Although perceptual studies may be the gold standard for assessing graphical realism, we also seek an automatic quantitative measure that does not require human experiments. For this, we adopt the "FCN score" from [22], and use it to evaluate the Cityscapes labels→photo task. The FCN metric evaluates how interpretable the generated photos are according to an off-the-shelf semantic segmentation algorithm (the fully-convolutional network, FCN, from [33]). The FCN predicts a label map for a generated photo. This label map can then be compared against the input ground truth labels using standard semantic segmentation metrics described below. The intuition is that if we generate a photo from a label map of "car on the road", then we have succeeded if the FCN applied to the generated photo detects "car on the road".

Semantic segmentation metrics To evaluate the performance of photo→labels, we use the standard metrics from the Cityscapes benchmark [4], including per-pixel accuracy, per-class accuracy, and mean class Intersection-Over-Union (Class IOU) [4].

5.1.2 Baselines

CoGAN [32] This method learns one GAN generator for domain X and one for domain Y, with tied weights on the first few layers for shared latent representations. Translation from X to Y can be achieved by finding a latent representation that generates image X and then rendering this latent representation into style Y.

SimGAN [46] Like our method, Shrivastava et al.[46] uses an adversarial loss to train a translation from X to Y.

run convolutionally on the 512×512 images at test time. We choose 256×256 in our experiments as many baselines cannot scale up to high-resolution images, and CoGAN cannot be tested fully convolutionally.

Loss	Map → Photo % Turkers labeled *real*	Photo → Map % Turkers labeled *real*
CoGAN [32]	0.6% ± 0.5%	0.9% ± 0.5%
BiGAN/ALI [9, 7]	2.1% ± 1.0%	1.9% ± 0.9%
SimGAN [46]	0.7% ± 0.5%	2.6% ± 1.1%
Feature loss + GAN	1.2% ± 0.6%	0.3% ± 0.2%
CycleGAN (ours)	**26.8% ± 2.8%**	**23.2% ± 3.4%**

Table 1: AMT "real vs fake" test on maps↔aerial photos at 256 × 256 resolution.

Loss	Per-pixel acc.	Per-class acc.	Class IOU
CoGAN [32]	0.40	0.10	0.06
BiGAN/ALI [9, 7]	0.19	0.06	0.02
SimGAN [46]	0.20	0.10	0.04
Feature loss + GAN	0.06	0.04	0.01
CycleGAN (ours)	**0.52**	**0.17**	**0.11**
pix2pix [22]	0.71	0.25	0.18

Table 2: FCN-scores for different methods, evaluated on Cityscapes labels→photo.

Loss	Per-pixel acc.	Per-class acc.	Class IOU
CoGAN [32]	0.45	0.11	0.08
BiGAN/ALI [9, 7]	0.41	0.13	0.07
SimGAN [46]	0.47	0.11	0.07
Feature loss + GAN	0.50	0.10	0.06
CycleGAN (ours)	**0.58**	**0.22**	**0.16**
pix2pix [22]	0.85	0.40	0.32

Table 3: Classification performance of photo→labels for different methods on cityscapes.

The regularization term $\|x - G(x)\|_1$ i s used to penalize making large changes at pixel level.

Feature loss + GAN We also test a variant of SimGAN [46] where the L1 loss is computed over deep image features using a pretrained network (VGG-16 `relu4_2` [47]), rather than over RGB pixel values. Computing distances in deep feature space, like this, is also sometimes referred to as using a "perceptual loss" [8, 23].

BiGAN/ALI [9, 7] Unconditional GANs [16] learn a generator $G : Z \rightarrow X$, that maps a random noise z to an image x. The BiGAN [9] and ALI [7] propose to also learn the inverse mapping function $F : X \rightarrow Z$. Though they were originally designed for mapping a latent vector z to an image x, we implemented the same objective for mapping a source image x to a target image y.

pix2pix [22] We also compare against pix2pix [22], which is trained on paired data, to see how close we can get to this "upper bound" without using any paired data.

For a fair comparison, we implement all the baselines using the same architecture and details as our method, except for CoGAN [32]. CoGAN builds on generators that produce images from a shared latent representation, which is incompatible with our image-to-image network. We use the public implementation of CoGAN instead.

5.1.3 Comparison against baselines

As can be seen in Figure 5 and Figure 6, we were unable to achieve compelling results with any of the baselines. Our method, on the other hand, can produce translations that are often of similar quality to the fully supervised pix2pix.

Loss	Per-pixel acc.	Per-class acc.	Class IOU
Cycle alone	0.22	0.07	0.02
GAN alone	0.51	0.11	0.08
GAN + forward cycle	**0.55**	**0.18**	**0.12**
GAN + backward cycle	0.39	0.14	0.06
CycleGAN (ours)	0.52	0.17	0.11

Table 4: Ablation study: FCN-scores for different variants of our method, evaluated on Cityscapes labels→photo.

Loss	Per-pixel acc.	Per-class acc.	Class IOU
Cycle alone	0.10	0.05	0.02
GAN alone	0.53	0.11	0.07
GAN + forward cycle	0.49	0.11	0.07
GAN + backward cycle	0.01	0.06	0.01
CycleGAN (ours)	**0.58**	**0.22**	**0.16**

Table 5: Ablation study: classification performance of photo→labels for different losses, evaluated on Cityscapes.

Table 1 reports performance regarding the AMT perceptual realism task. Here, we see that our method can fool participants on around a quarter of trials, in both the maps→aerial photos direction and the aerial photos→maps direction at 256 × 256 resolution[3]. All the baselines almost never fooled participants.

Table 2 assesses the performance of the labels→photo task on the Cityscapes and Table 3 evaluates the opposite mapping (photos→labels). In both cases, our method again outperforms the baselines.

5.1.4 Analysis of the loss function

In Table 4 and Table 5, we compare against ablations of our full loss. Removing the GAN loss substantially degrades results, as does removing the cycle-consistency loss. We therefore conclude that both terms are critical to our results. We also evaluate our method with the cycle loss in only one direction: GAN + forward cycle loss $\mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1]$, or GAN + backward cycle loss $\mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]$ (Equation 2) and find that it often incurs training instability and causes mode collapse, especially for the direction of the mapping that was removed. Figure 7 shows several qualitative examples.

5.1.5 Image reconstruction quality

In Figure 4, we show a few random samples of the reconstructed images $F(G(x))$. We observed that the reconstructed images were often close to the original inputs x, at both training and testing time, even in cases where one domain represents significantly more diverse information, such as map↔aerial photos.

[3] We also train CycleGAN and pix2pix at 512 × 512 resolution, and observe the comparable performance: maps→aerial photos: CycleGAN: 37.5% ± 3.6% and pix2pix: 33.9% ± 3.1%; aerial photos→maps: CycleGAN: 16.5% ± 4.1% and pix2pix: 8.5% ± 2.6%

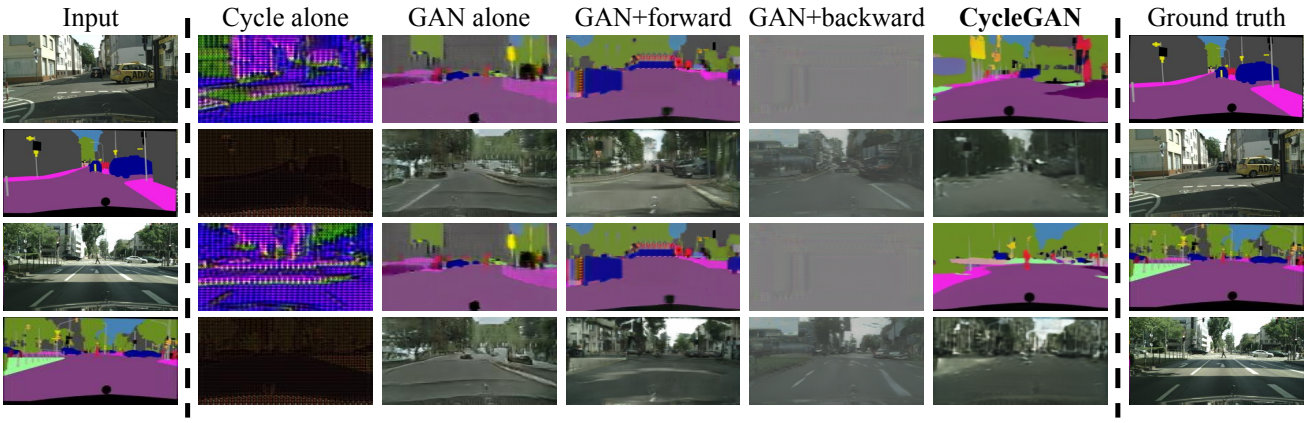

Figure 7: Different variants of our method for mapping labels↔photos trained on cityscapes. From left to right: input, cycle-consistency loss alone, adversarial loss alone, GAN + forward cycle-consistency loss ($F(G(x)) \approx x$), GAN + backward cycle-consistency loss ($G(F(y)) \approx y$), CycleGAN (our full method), and ground truth. Both *Cycle alone* and *GAN + backward* fail to produce images similar to the target domain. *GAN alone* and *GAN + forward* suffer from mode collapse, producing identical label maps regardless of the input photo.

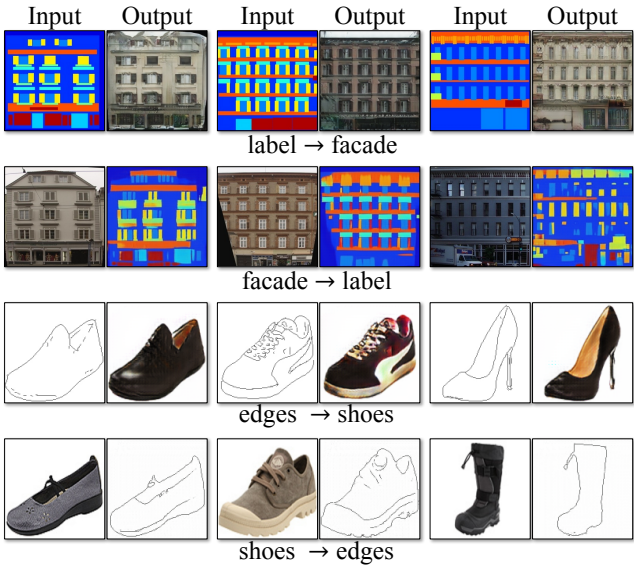

Figure 8: Example results of CycleGAN on paired datasets used in "pix2pix" [22] such as architectural labels↔photos and edges↔shoes.

5.1.6 Additional results on paired datasets

Figure 8 shows some example results on other paired datasets used in "pix2pix" [22], such as architectural labels↔photos from the CMP Facade Database [40], and edges↔shoes from the UT Zappos50K dataset [60]. The image quality of our results is close to those produced by the fully supervised pix2pix while our method learns the mapping without paired supervision.

5.2. Applications

We demonstrate our method on several applications where paired training data does not exist. Please refer to the appendix (Section 7) for more details about the datasets. We observe that translations on training data are often more appealing than those on test data, and full results of all applications on both training and test data can be viewed on our project website.

Collection style transfer (Figure 10 and Figure 11) We train the model on landscape photographs downloaded from Flickr and WikiArt. Unlike recent work on "neural style transfer" [13], our method learns to mimic the style of an entire *collection* of artworks, rather than transferring the style of a single selected piece of art. Therefore, we can learn to generate photos in the style of, e.g., Van Gogh, rather than just in the style of Starry Night. The size of the dataset for each artist/style was 526, 1073, 400, and 563 for Cezanne, Monet, Van Gogh, and Ukiyo-e.

Object transfiguration (Figure 13) The model is trained to translate one object class from ImageNet [5] to another (each class contains around 1000 training images). Turmukhambetov et al. [50] propose a subspace model to translate one object into another object of the same category, while our method focuses on object transfiguration between two visually similar categories.

Season transfer (Figure 13) The model is trained on 854 winter photos and 1273 summer photos of Yosemite downloaded from Flickr.

Photo generation from paintings (Figure 12) For painting→photo, we find that it is helpful to introduce an additional loss to encourage the mapping to preserve color composition between the input and output. In particular, we adopt the technique of Taigman et al. [49] and regularize the generator to be near an identity mapping when real samples of the target domain are provided as the input to the generator: i.e., $\mathcal{L}_{\text{identity}}(G, F) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(y) - y\|_1] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(x) - x\|_1]$.

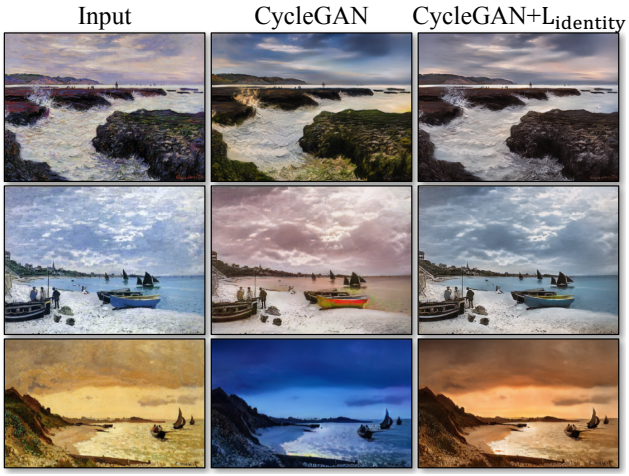

Figure 9: The effect of the *identity mapping loss* on Monet's painting→ photos. From left to right: input paintings, CycleGAN without identity mapping loss, CycleGAN with identity mapping loss. The identity mapping loss helps preserve the color of the input paintings.

Without $\mathcal{L}_{\text{identity}}$, the generator G and F are free to change the tint of input images when there is no need to. For example, when learning the mapping between Monet's paintings and Flickr photographs, the generator often maps paintings of daytime to photographs taken during sunset, because such a mapping may be equally valid under the adversarial loss and cycle consistency loss. The effect of this *identity mapping loss* are shown in Figure 9.

In Figure 12, we show additional results translating Monet's paintings to photographs. This figure and Figure 9 show results on paintings that were included in the *training set*, whereas for all other experiments in the paper, we only evaluate and show test set results. Because the training set does not include paired data, coming up with a plausible translation for a training set painting is a nontrivial task. Indeed, since Monet is no longer able to create new paintings, generalization to unseen, "test set", paintings is not a pressing problem.

Photo enhancement (Figure 14) We show that our method can be used to generate photos with shallower depth of field. We train the model on flower photos downloaded from Flickr. The source domain consists of flower photos taken by smartphones, which usually have deep DoF due to a small aperture. The target contains photos captured by DSLRs with a larger aperture. Our model successfully generates photos with shallower depth of field from the photos taken by smartphones.

Comparison with Gatys et al. [13] In Figure 15, we compare our results with neural style transfer [13] on photo stylization. For each row, we first use two representative artworks as the style images for [13]. Our method, on the other hand, can produce photos in the style of entire *collection*. To compare against neural style transfer of an entire collection, we compute the average Gram Matrix across the target domain and use this matrix to transfer the "average style" with Gatys et al [13].

Figure 16 demonstrates similar comparisons for other translation tasks. We observe that Gatys et al. [13] requires finding target style images that closely match the desired output, but still often fails to produce photorealistic results, while our method succeeds to generate natural-looking results, similar to the target domain.

6. Limitations and Discussion

Although our method can achieve compelling results in many cases, the results are far from uniformly positive. Figure 17 shows several typical failure cases. On translation tasks that involve color and texture changes, as many of those reported above, the method often succeeds. We have also explored tasks that require geometric changes, with little success. For example, on the task of dog→cat transfiguration, the learned translation degenerates into making minimal changes to the input (Figure 17). This failure might be caused by our generator architectures which are tailored for good performance on the appearance changes. Handling more varied and extreme transformations, especially geometric changes, is an important problem for future work.

Some failure cases are caused by the distribution characteristics of the training datasets. For example, our method has got confused in the horse → zebra example (Figure 17, right), because our model was trained on the *wild horse* and *zebra* synsets of ImageNet, which does not contain images of a person riding a horse or zebra.

We also observe a lingering gap between the results achievable with paired training data and those achieved by our unpaired method. In some cases, this gap may be very hard – or even impossible – to close: for example, our method sometimes permutes the labels for tree and building in the output of the photos→labels task. Resolving this ambiguity may require some form of weak semantic supervision. Integrating weak or semi-supervised data may lead to substantially more powerful translators, still at a fraction of the annotation cost of the fully-supervised systems.

Nonetheless, in many cases completely unpaired data is plentifully available and should be made use of. This paper pushes the boundaries of what is possible in this "unsupervised" setting.

Acknowledgments: We thank Aaron Hertzmann, Shiry Ginosar, Deepak Pathak, Bryan Russell, Eli Shechtman, Richard Zhang, and Tinghui Zhou for many helpful comments. This work was supported in part by NSF SMA-1514512, NSF IIS-1633310, a Google Research Award, Intel Corp, and hardware donations from NVIDIA. JYZ is supported by the Facebook Graduate Fellowship and TP is supported by the Samsung Scholarship. The photographs used for style transfer were taken by AE, mostly in France.

Figure 10: Collection style transfer I: we transfer input images into the artistic styles of Monet, Van Gogh, Cezanne, and Ukiyo-e. Please see our website for additional examples.

Figure 11: Collection style transfer II: we transfer input images into the artistic styles of Monet, Van Gogh, Cezanne, Ukiyo-e. Please see our website for additional examples.

Figure 12: Relatively successful results on mapping Monet's paintings to a photographic style. Please see our website for additional examples.

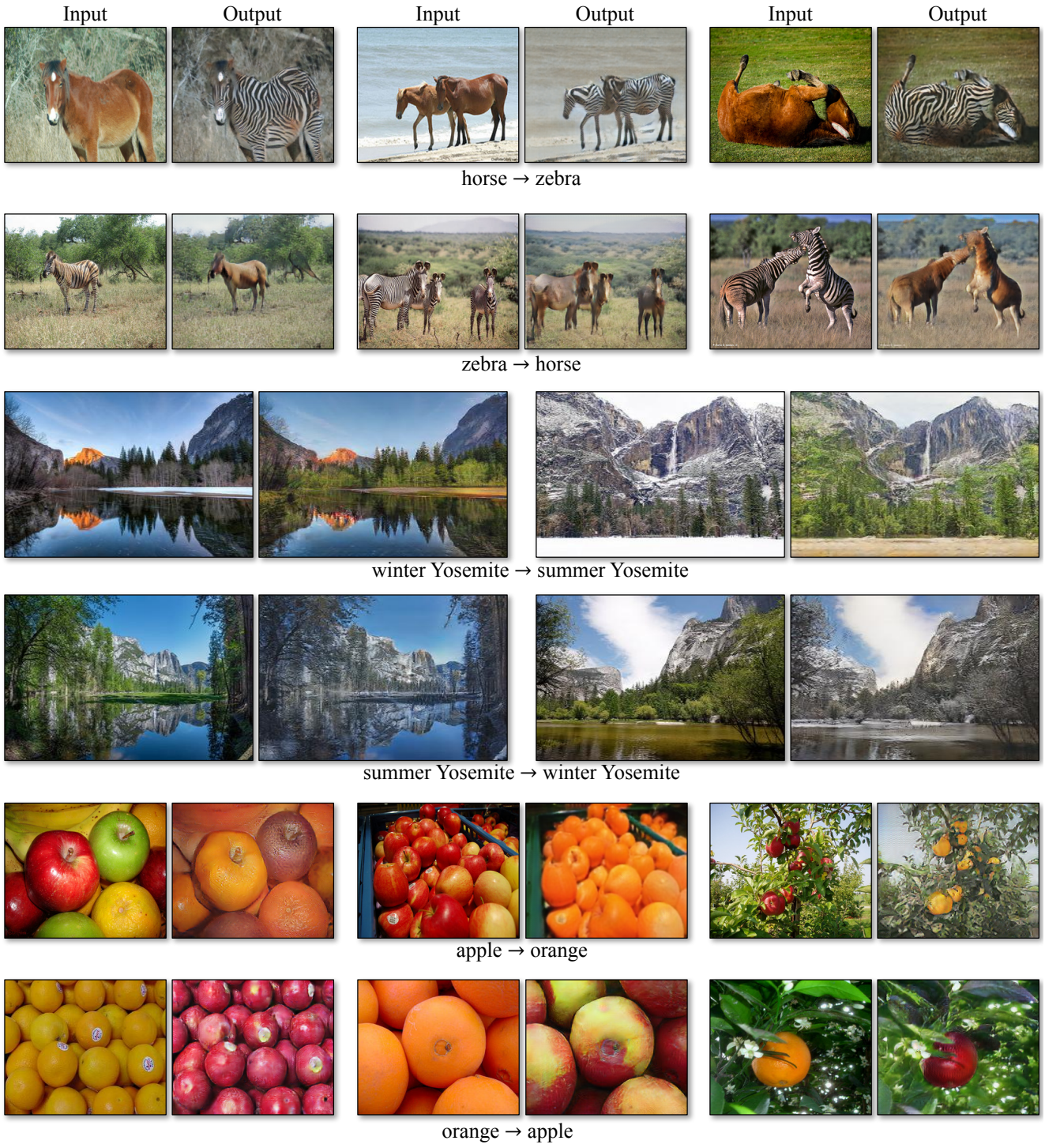

Figure 13: Our method applied to several translation problems. These images are selected as relatively successful results – please see our website for more comprehensive and random results. In the top two rows, we show results on object transfiguration between horses and zebras, trained on 939 images from the *wild horse* class and 1177 images from the *zebra* class in Imagenet [5]. Also check out the horse→zebra demo video. The middle two rows show results on season transfer, trained on winter and summer photos of Yosemite from Flickr. In the bottom two rows, we train our method on 996 *apple* images and 1020 *navel orange* images from ImageNet.

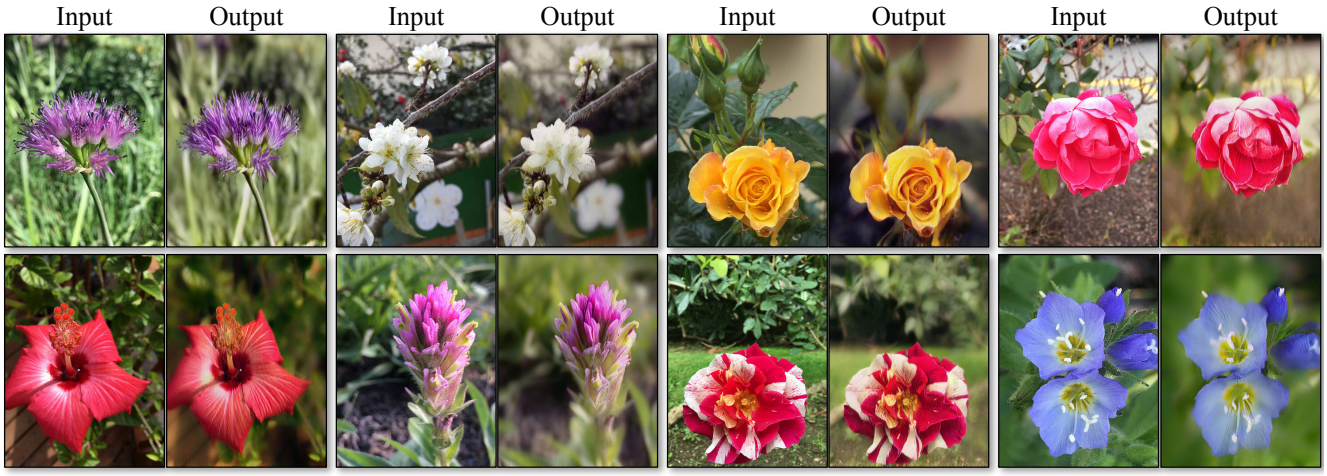

Figure 14: Photo enhancement: mapping from a set of smartphone snaps to professional DSLR photographs, the system often learns to produce shallow focus. Here we show some of the most successful results in our test set – average performance is considerably worse. Please see our website for more comprehensive and random examples.

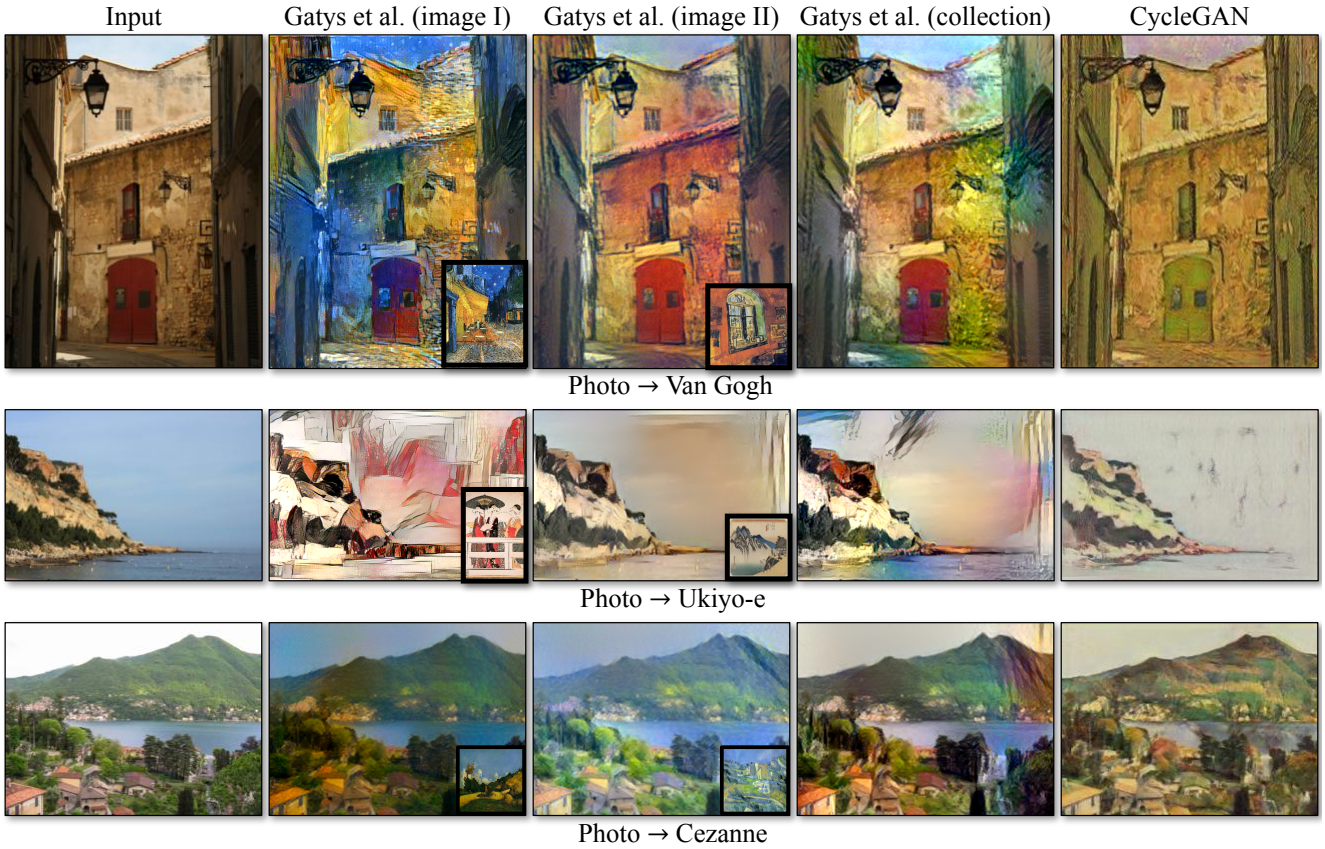

Figure 15: We compare our method with neural style transfer [13] on photo stylization. Left to right: input image, results from Gatys et al. [13] using two different representative artworks as style images, results from Gatys et al. [13] using the entire collection of the artist, and CycleGAN (ours).

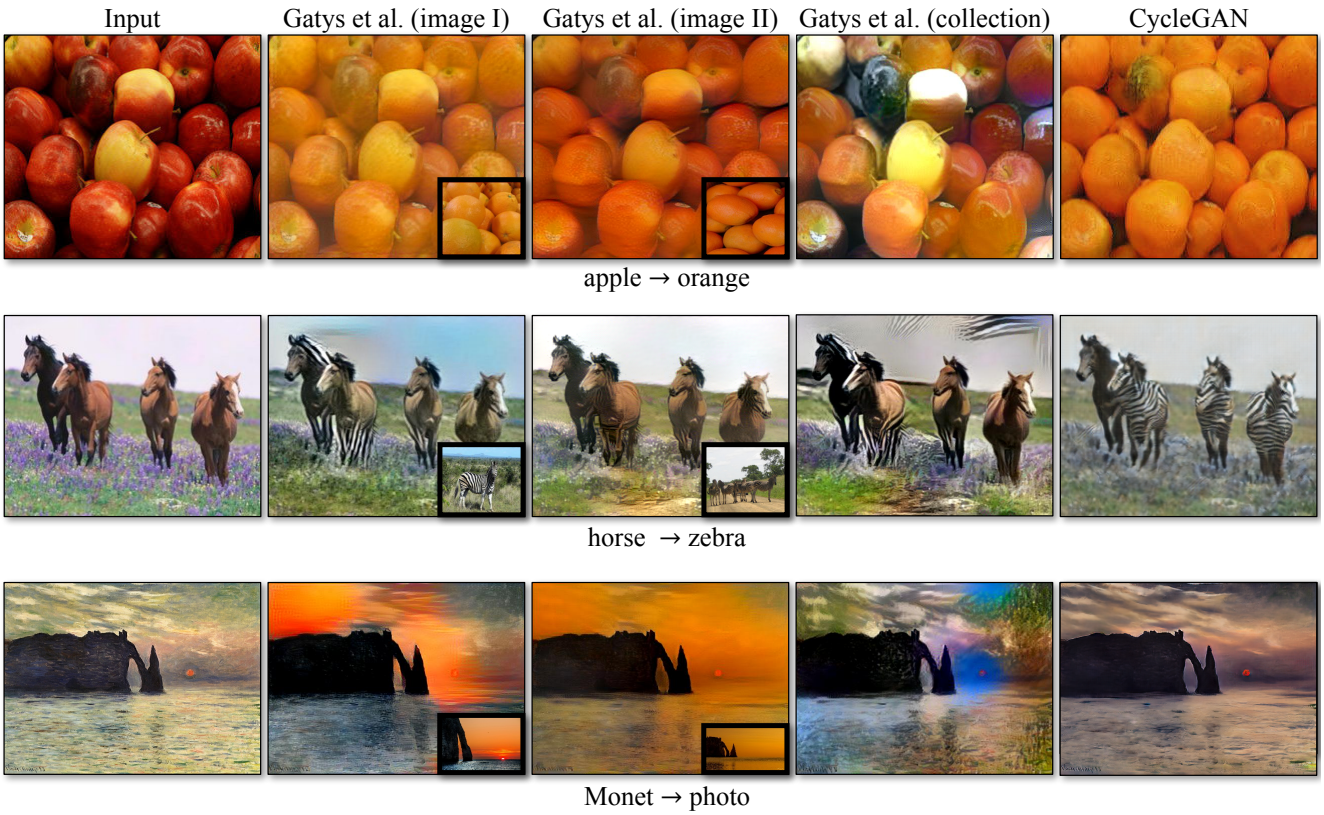

Figure 16: We compare our method with neural style transfer [13] on various applications. From top to bottom: apple→orange, horse→zebra, and Monet→photo. Left to right: input image, results from Gatys et al. [13] using two different images as style images, results from Gatys et al. [13] using all the images from the target domain, and CycleGAN (ours).

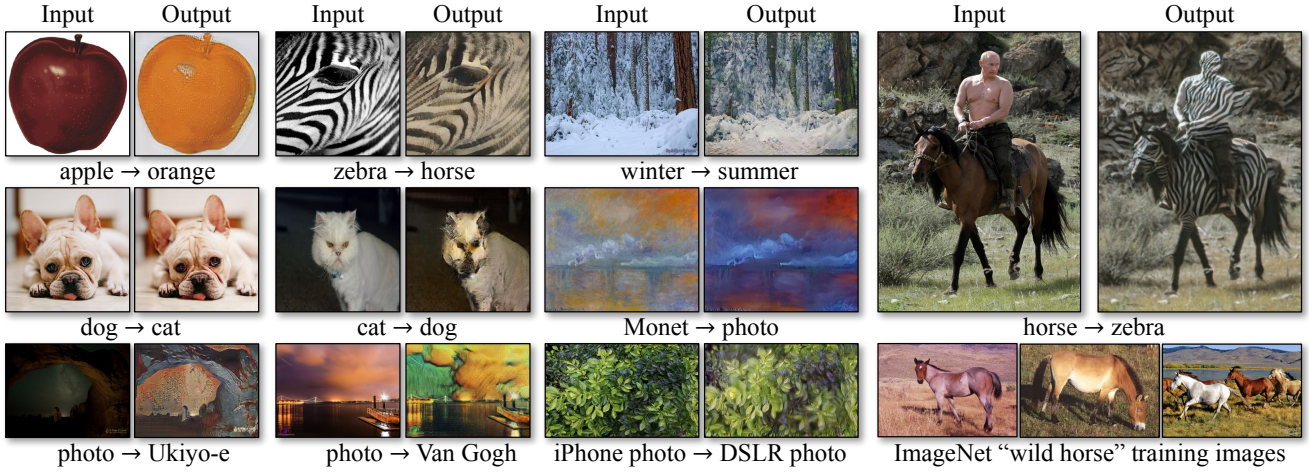

Figure 17: Typical failure cases of our method. Left: in the task of dog→cat transfiguration, CycleGAN can only make minimal changes to the input. Right: CycleGAN also fails in this horse → zebra example as our model has not seen images of horseback riding during training. Please see our website for more comprehensive results.

References

[1] Y. Aytar, L. Castrejon, C. Vondrick, H. Pirsiavash, and A. Torralba. Cross-modal scene networks. *PAMI*, 2016. 3

[2] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *CVPR*, 2017. 3

[3] R. W. Brislin. Back-translation for cross-cultural research. *Journal of cross-cultural psychology*, 1(3):185–216, 1970. 2, 3

[4] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. 2, 5, 6, 18

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 8, 13, 18

[6] E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, 2015. 2

[7] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. In *ICLR*, 2017. 6, 7

[8] A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. In *NIPS*, 2016. 7

[9] V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. Courville. Adversarially learned inference. In *ICLR*, 2017. 6, 7

[10] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *ICCV*, 1999. 3

[11] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *ICCV*, 2015. 2

[12] L. A. Gatys, M. Bethge, A. Hertzmann, and E. Shechtman. Preserving color in neural artistic style transfer. *arXiv preprint arXiv:1606.05897*, 2016. 3

[13] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. *CVPR*, 2016. 3, 8, 9, 14, 15

[14] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *CVPR*, 2017. 3

[15] I. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016. 2, 4, 5

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014. 2, 3, 4, 7

[17] D. He, Y. Xia, T. Qin, L. Wang, N. Yu, T. Liu, and W.-Y. Ma. Dual learning for machine translation. In *NIPS*, 2016. 3

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5

[19] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *SIGGRAPH*, 2001. 2, 3

[20] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. 5

[21] Q.-X. Huang and L. Guibas. Consistent shape maps via semidefinite programming. In *Symposium on Geometry Processing*, 2013. 3

[22] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. 2, 3, 5, 6, 7, 8, 18

[23] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 2, 3, 5, 7, 18

[24] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-backward error: Automatic detection of tracking failures. In *ICPR*, 2010. 3

[25] L. Karacan, Z. Akata, A. Erdem, and E. Erdem. Learning to generate images of outdoor scenes from attributes and semantic layouts. *arXiv preprint arXiv:1612.00215*, 2016. 3

[26] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 5

[27] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *ICLR*, 2014. 3

[28] P.-Y. Laffont, Z. Ren, X. Tao, C. Qian, and J. Hays. Transient attributes for high-level understanding and editing of outdoor scenes. *ACM TOG*, 33(4):149, 2014. 2

[29] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, 2017. 5

[30] C. Li and M. Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. *ECCV*, 2016. 5

[31] M.-Y. Liu, T. Breuel, and J. Kautz. Unsupervised image-to-image translation networks. In *NIPS*, 2017. 3

[32] M.-Y. Liu and O. Tuzel. Coupled generative adversarial networks. In *NIPS*, 2016. 3, 6, 7

[33] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 2, 3, 6

[34] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders. In *ICLR*, 2016. 5

[35] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks. In *CVPR*. IEEE, 2017. 5

[36] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. In *ICLR*, 2016. 2

[37] M. F. Mathieu, J. Zhao, A. Ramesh, P. Sprechmann, and Y. LeCun. Disentangling factors of variation in deep representation using adversarial training. In *NIPS*, 2016. 2

[38] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. *CVPR*, 2016. 2

[39] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016. 2

[40] R. Š. Radim Tyleček. Spatial pattern templates for recognition of objects with regular structure. In *Proc. GCPR*, Saarbrucken, Germany, 2013. 8, 18

[41] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In *ICML*, 2016. 2

[42] R. Rosales, K. Achan, and B. J. Frey. Unsupervised image translation. In *ICCV*, 2003. 3

[43] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training GANs. In *NIPS*, 2016. 2

[44] P. Sangkloy, J. Lu, C. Fang, F. Yu, and J. Hays. Scribbler: Controlling deep image synthesis with sketch and color. In *CVPR*, 2017. 3

[45] Y. Shih, S. Paris, F. Durand, and W. T. Freeman. Data-driven hallucination of different times of day from a single outdoor photo. *ACM TOG*, 32(6):200, 2013. 2

[46] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. In *CVPR*, 2017. 3, 5, 6, 7

[47] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 7

[48] N. Sundaram, T. Brox, and K. Keutzer. Dense point trajectories by gpu-accelerated large displacement optical flow. In *ECCV*, 2010. 3

[49] Y. Taigman, A. Polyak, and L. Wolf. Unsupervised cross-domain image generation. In *ICLR*, 2017. 3, 8

[50] D. Turmukhambetov, N. D. Campbell, S. J. Prince, and J. Kautz. Modeling object appearance using context-conditioned component analysis. In *CVPR*, 2015. 8

[51] M. Twain. The jumping frog: in english, then in french, and then clawed back into a civilized language once more by patient. *Unremunerated Toil*, 3, 1903. 3

[52] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, 2016. 3

[53] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 5

[54] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *NIPS*, 2016. 2

[55] F. Wang, Q. Huang, and L. J. Guibas. Image co-segmentation via consistent functional maps. In *ICCV*, 2013. 3

[56] X. Wang and A. Gupta. Generative image modeling using style and structure adversarial networks. In *ECCV*, 2016. 2

[57] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NIPS*, 2016. 2

[58] S. Xie and Z. Tu. Holistically-nested edge detection. In *ICCV*, 2015. 2

[59] Z. Yi, H. Zhang, T. Gong, Tan, and M. Gong. Dualgan: Unsupervised dual learning for image-to-image translation. In *ICCV*, 2017. 3

[60] A. Yu and K. Grauman. Fine-grained visual comparisons with local learning. In *CVPR*, 2014. 8, 18

[61] C. Zach, M. Klopschitz, and M. Pollefeys. Disambiguating visual relations using loop constraints. In *CVPR*, 2010. 3

[62] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *ECCV*, 2016. 2

[63] J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. In *ICLR*, 2017. 2

[64] T. Zhou, P. Krahenbuhl, M. Aubry, Q. Huang, and A. A. Efros. Learning dense correspondence via 3d-guided cycle consistency. In *CVPR*, 2016. 2, 3

[65] T. Zhou, Y. J. Lee, S. Yu, and A. A. Efros. Flowweb: Joint image set alignment by weaving consistent, pixel-wise correspondences. In *CVPR*, 2015. 3

[66] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros. Generative visual manipulation on the natural image manifold. In *ECCV*, 2016. 2

7. Appendix

7.1. Training details

We train our networks from scratch, with a learning rate of 0.0002. In practice, we divide the objective by 2 while optimizing D, which slows down the rate at which D learns, relative to the rate of G. We keep the same learning rate for the first 100 epochs and linearly decay the rate to zero over the next 100 epochs. Weights are initialized from a Gaussian distribution $\mathcal{N}(0, 0.02)$.

Cityscapes label↔Photo 2975 training images from the Cityscapes training set [4] with image size 128×128. We used the Cityscapes val set for testing.

Maps↔aerial photograph 1096 training images were scraped from Google Maps [22] with image size 256×256. Images were sampled from in and around New York City. Data was then split into train and test about the median latitude of the sampling region (with a buffer region added to ensure that no training pixel appeared in the test set).

Architectural facades labels↔photo 400 training images from the CMP Facade Database [40].

Edges→shoes around $50,000$ training images from UT Zappos50K dataset [60]. The model was trained for 5 epochs.

Horse↔Zebra and Apple↔Orange We downloaded the images from ImageNet [5] using keywords *wild horse*, *zebra*, *apple*, and *navel orange*. The images were scaled to 256×256 pixels. The training set size of each class: 939 (horse), 1177 (zebra), 996 (apple), and 1020 (orange).

Summer↔Winter Yosemite The images were downloaded using Flickr API with the tag *yosemite* and the *date-taken* field. Black-and-white photos were pruned. The images were scaled to 256×256 pixels. The training size of each class: 1273 (summer) and 854 (winter).

Photo↔Art for style transfer The art images were downloaded from Wikiart.org. Some artworks that were sketches or too obscene were pruned by hand. The photos were downloaded from Flickr using the combination of tags *landscape* and *landscapephotography*. Black-and-white photos were pruned. The images were scaled to 256×256 pixels. The training set size of each class was 1074 (Monet), 584 (Cezanne), 401 (Van Gogh), 1433 (Ukiyo-e), and 6853 (Photographs). The Monet dataset was particularly pruned to include only landscape paintings, and the Van Gogh dataset included only his later works that represent his most recognizable artistic style.

Monet's paintings→photos To achieve high resolution while conserving memory, we used random square crops of the original images for training. To generate results, we passed images of width 512 pixels with correct aspect ratio to the generator network as input. The weight for the identity mapping loss was 0.5λ where λ was the weight for cycle consistency loss. We set $\lambda = 10$.

Flower photo enhancement Flower images taken on smartphones were downloaded from Flickr by searching for the photos taken by *Apple iPhone 5, 5s, or 6*, with search text *flower*. DSLR images with shallow DoF were also downloaded from Flickr by search tag *flower, dof*. The images were scaled to 360 pixels by width. The identity mapping loss of weight 0.5λ was used. The training set size of the smartphone and DSLR dataset were 1813 and 3326, respectively. We set $\lambda = 10$.

7.2. Network architectures

We provide both PyTorch and Torch implementations.

Generator architectures We adopt our architectures from Johnson et al. [23]. We use 6 residual blocks for 128×128 training images, and 9 residual blocks for 256×256 or higher-resolution training images. Below, we follow the naming convention used in the Johnson et al.'s Github repository.

Let `c7s1-k` denote a 7×7 Convolution-InstanceNorm-ReLU layer with k filters and stride 1. `dk` denotes a 3×3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2. Reflection padding was used to reduce artifacts. `Rk` denotes a residual block that contains two 3×3 convolutional layers with the same number of filters on both layer. `uk` denotes a 3×3 fractional-strided-Convolution-InstanceNorm-ReLU layer with k filters and stride $\frac{1}{2}$.

The network with 6 residual blocks consists of:
`c7s1-64,d128,d256,R256,R256,R256,`
`R256,R256,R256,u128,u64,c7s1-3`

The network with 9 residual blocks consists of:
`c7s1-64,d128,d256,R256,R256,R256,`
`R256,R256,R256,R256,R256,R256,u128`
`u64,c7s1-3`

Discriminator architectures For discriminator networks, we use 70×70 PatchGAN [22]. Let `Ck` denote a 4×4 Convolution-InstanceNorm-LeakyReLU layer with k filters and stride 2. After the last layer, we apply a convolution to produce a 1-dimensional output. We do not use InstanceNorm for the first `C64` layer. We use leaky ReLUs with a slope of 0.2. The discriminator architecture is:
`C64-C128-C256-C512`

Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie[*], Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair[†], Aaron Courville, Yoshua Bengio[‡]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

1 Introduction

The promise of deep learning is to discover rich, hierarchical models [2] that represent probability distributions over the kinds of data encountered in artificial intelligence applications, such as natural images, audio waveforms containing speech, and symbols in natural language corpora. So far, the most striking successes in deep learning have involved discriminative models, usually those that map a high-dimensional, rich sensory input to a class label [14, 22]. These striking successes have primarily been based on the backpropagation and dropout algorithms, using piecewise linear units [19, 9, 10] which have a particularly well-behaved gradient . Deep *generative* models have had less of an impact, due to the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies, and due to difficulty of leveraging the benefits of piecewise linear units in the generative context. We propose a new generative model estimation procedure that sidesteps these difficulties. [1]

In the proposed *adversarial nets* framework, the generative model is pitted against an adversary: a discriminative model that learns to determine whether a sample is from the model distribution or the data distribution. The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistiguishable from the genuine articles.

[*] Jean Pouget-Abadie is visiting Université de Montréal from Ecole Polytechnique.
[†] Sherjil Ozair is visiting Université de Montréal from Indian Institute of Technology Delhi
[‡] Yoshua Bengio is a CIFAR Senior Fellow.
[1] All code and hyperparameters available at `http://www.github.com/goodfeli/adversarial`

This framework can yield specific training algorithms for many kinds of model and optimization algorithm. In this article, we explore the special case when the generative model generates samples by passing random noise through a multilayer perceptron, and the discriminative model is also a multilayer perceptron. We refer to this special case as *adversarial nets*. In this case, we can train both models using only the highly successful backpropagation and dropout algorithms [17] and sample from the generative model using only forward propagation. No approximate inference or Markov chains are necessary.

2 Related work

An alternative to directed graphical models with latent variables are undirected graphical models with latent variables, such as restricted Boltzmann machines (RBMs) [27, 16], deep Boltzmann machines (DBMs) [26] and their numerous variants. The interactions within such models are represented as the product of unnormalized potential functions, normalized by a global summation/integration over all states of the random variables. This quantity (the *partition function*) and its gradient are intractable for all but the most trivial instances, although they can be estimated by Markov chain Monte Carlo (MCMC) methods. Mixing poses a significant problem for learning algorithms that rely on MCMC [3, 5].

Deep belief networks (DBNs) [16] are hybrid models containing a single undirected layer and several directed layers. While a fast approximate layer-wise training criterion exists, DBNs incur the computational difficulties associated with both undirected and directed models.

Alternative criteria that do not approximate or bound the log-likelihood have also been proposed, such as score matching [18] and noise-contrastive estimation (NCE) [13]. Both of these require the learned probability density to be analytically specified up to a normalization constant. Note that in many interesting generative models with several layers of latent variables (such as DBNs and DBMs), it is not even possible to derive a tractable unnormalized probability density. Some models such as denoising auto-encoders [30] and contractive autoencoders have learning rules very similar to score matching applied to RBMs. In NCE, as in this work, a discriminative training criterion is employed to fit a generative model. However, rather than fitting a separate discriminative model, the generative model itself is used to discriminate generated data from samples a fixed noise distribution. Because NCE uses a fixed noise distribution, learning slows dramatically after the model has learned even an approximately correct distribution over a small subset of the observed variables.

Finally, some techniques do not involve defining a probability distribution explicitly, but rather train a generative machine to draw samples from the desired distribution. This approach has the advantage that such machines can be designed to be trained by back-propagation. Prominent recent work in this area includes the generative stochastic network (GSN) framework [5], which extends generalized denoising auto-encoders [4]: both can be seen as defining a parameterized Markov chain, i.e., one learns the parameters of a machine that performs one step of a generative Markov chain. Compared to GSNs, the adversarial nets framework does not require a Markov chain for sampling. Because adversarial nets do not require feedback loops during generation, they are better able to leverage piecewise linear units [19, 9, 10], which improve the performance of backpropagation but have problems with unbounded activation when used ina feedback loop. More recent examples of training a generative machine by back-propagating into it include recent work on auto-encoding variational Bayes [20] and stochastic backpropagation [24].

3 Adversarial nets

The adversarial modeling framework is most straightforward to apply when the models are both multilayer perceptrons. To learn the generator's distribution p_g over data $\boldsymbol{x}$, we define a prior on input noise variables $p_{\boldsymbol{z}}(\boldsymbol{z})$, then represent a mapping to data space as $G(\boldsymbol{z}; \theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g. We also define a second multilayer perceptron $D(\boldsymbol{x}; \theta_d)$ that outputs a single scalar. $D(\boldsymbol{x})$ represents the probability that $\boldsymbol{x}$ came from the data rather than p_g. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize $\log(1 - D(G(\boldsymbol{z})))$:

In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \tag{1}$$

In the next section, we present a theoretical analysis of adversarial nets, essentially showing that the training criterion allows one to recover the data generating distribution as G and D are given enough capacity, i.e., in the non-parametric limit. See Figure 1 for a less formal, more pedagogical explanation of the approach. In practice, we must implement the game using an iterative, numerical approach. Optimizing D to completion in the inner loop of training is computationally prohibitive, and on finite datasets would result in overfitting. Instead, we alternate between k steps of optimizing D and one step of optimizing G. This results in D being maintained near its optimal solution, so long as G changes slowly enough. This strategy is analogous to the way that SML/PCD [31, 29] training maintains samples from a Markov chain from one learning step to the next in order to avoid burning in a Markov chain as part of the inner loop of learning. The procedure is formally presented in Algorithm 1.

In practice, equation 1 may not provide sufficient gradient for G to learn well. Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $\log(1 - D(G(\boldsymbol{z})))$ saturates. Rather than training G to minimize $\log(1 - D(G(\boldsymbol{z})))$ we can train G to maximize $\log D(G(\boldsymbol{z}))$. This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning.

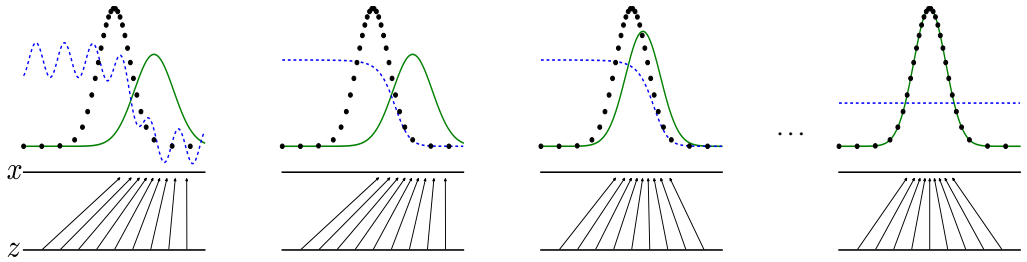

Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

4 Theoretical Results

The generator G implicitly defines a probability distribution p_g as the distribution of the samples $G(\boldsymbol{z})$ obtained when $\boldsymbol{z} \sim p_{\boldsymbol{z}}$. Therefore, we would like Algorithm 1 to converge to a good estimator of p_{data}, if given enough capacity and training time. The results of this section are done in a non-parametric setting, e.g. we represent a model with infinite capacity by studying convergence in the space of probability density functions.

We will show in section 4.1 that this minimax game has a global optimum for $p_g = p_{\text{data}}$. We will then show in section 4.2 that Algorithm 1 optimizes Eq 1, thus obtaining the desired result.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.

  • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

4.1 Global Optimality of $p_g = p_{\text{data}}$

We first consider the optimal discriminator D for any given generator G.

Proposition 1. *For G fixed, the optimal discriminator D is*

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{2}$$

Proof. The training criterion for the discriminator D, given any generator G, is to maximize the quantity $V(G, D)$

$$\begin{aligned} V(G,D) &= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))dx + \int_{z} p_{\boldsymbol{z}}(\boldsymbol{z}) \log(1 - D(g(\boldsymbol{z})))dz \\ &= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))dx \end{aligned} \tag{3}$$

For any $(a, b) \in \mathbb{R}^2 \setminus \{0, 0\}$, the function $y \to a\log(y) + b\log(1 - y)$ achieves its maximum in $[0, 1]$ at $\frac{a}{a+b}$. The discriminator does not need to be defined outside of $Supp(p_{\text{data}}) \cup Supp(p_g)$, concluding the proof. □

Note that the training objective for D can be interpreted as maximizing the log-likelihood for estimating the conditional probability $P(Y = y|\boldsymbol{x})$, where Y indicates whether $\boldsymbol{x}$ comes from p_{data} (with $y = 1$) or from p_g (with $y = 0$). The minimax game in Eq. 1 can now be reformulated as:

$$\begin{aligned} C(G) =& \max_D V(G,D) \\ =& \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1 - D_G^*(G(\boldsymbol{z})))] \\ =& \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))] \\ =& \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] \end{aligned} \tag{4}$$

Theorem 1. *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $-\log 4$.*

Proof. For $p_g = p_{\text{data}}$, $D^*_G(\boldsymbol{x}) = \frac{1}{2}$, (consider Eq. 2). Hence, by inspecting Eq. 4 at $D^*_G(\boldsymbol{x}) = \frac{1}{2}$, we find $C(G) = \log \frac{1}{2} + \log \frac{1}{2} = -\log 4$. To see that this is the best possible value of $C(G)$, reached only for $p_g = p_{\text{data}}$, observe that

$$\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} [-\log 2] + \mathbb{E}_{\boldsymbol{x} \sim p_g} [-\log 2] = -\log 4$$

and that by subtracting this expression from $C(G) = V(D^*_G, G)$, we obtain:

$$C(G) = -\log(4) + KL\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \middle\| \frac{p_{\text{data}} + p_g}{2}\right) \tag{5}$$

where KL is the Kullback–Leibler divergence. We recognize in the previous expression the Jensen–Shannon divergence between the model's distribution and the data generating process:

$$C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right) \tag{6}$$

Since the Jensen–Shannon divergence between two distributions is always non-negative and zero only when they are equal, we have shown that $C^* = -\log(4)$ is the global minimum of $C(G)$ and that the only solution is $p_g = p_{\text{data}}$, i.e., the generative model perfectly replicating the data generating process. □

4.2 Convergence of Algorithm 1

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{\boldsymbol{x} \sim p_{data}}[\log D^*_G(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_g}[\log(1 - D^*_G(\boldsymbol{x}))]$$

then p_g converges to p_{data}

Proof. Consider $V(G, D) = U(p_g, D)$ as a function of p_g as done in the above criterion. Note that $U(p_g, D)$ is convex in p_g. The subderivatives of a supremum of convex functions include the derivative of the function at the point where the maximum is attained. In other words, if $f(x) = \sup_{\alpha \in \mathcal{A}} f_\alpha(x)$ and $f_\alpha(x)$ is convex in x for every α, then $\partial f_\beta(x) \in \partial f$ if $\beta = \arg\sup_{\alpha \in \mathcal{A}} f_\alpha(x)$. This is equivalent to computing a gradient descent update for p_g at the optimal D given the corresponding G. $\sup_D U(p_g, D)$ is convex in p_g with a unique global optima as proven in Thm 1, therefore with sufficiently small updates of p_g, p_g converges to p_x, concluding the proof. □

In practice, adversarial nets represent a limited family of p_g distributions via the function $G(\boldsymbol{z}; \theta_g)$, and we optimize θ_g rather than p_g itself. Using a multilayer perceptron to define G introduces multiple critical points in parameter space. However, the excellent performance of multilayer perceptrons in practice suggests that they are a reasonable model to use despite their lack of theoretical guarantees.

5 Experiments

We trained adversarial nets an a range of datasets including MNIST[23], the Toronto Face Database (TFD) [28], and CIFAR-10 [21]. The generator nets used a mixture of rectifier linear activations [19, 9] and sigmoid activations, while the discriminator net used maxout [10] activations. Dropout [17] was applied in training the discriminator net. While our theoretical framework permits the use of dropout and other noise at intermediate layers of the generator, we used noise as the input to only the bottommost layer of the generator network.

We estimate probability of the test set data under p_g by fitting a Gaussian Parzen window to the samples generated with G and reporting the log-likelihood under this distribution. The σ parameter

Model	MNIST	TFD
DBN [3]	138 ± 2	1909 ± 66
Stacked CAE [3]	121 ± 1.6	$\mathbf{2110 \pm 50}$
Deep GSN [6]	214 ± 1.1	1890 ± 29
Adversarial nets	$\mathbf{225 \pm 2}$	$\mathbf{2057 \pm 26}$

Table 1: Parzen window-based log-likelihood estimates. The reported numbers on MNIST are the mean log-likelihood of samples on test set, with the standard error of the mean computed across examples. On TFD, we computed the standard error across folds of the dataset, with a different σ chosen using the validation set of each fold. On TFD, σ was cross validated on each fold and mean log-likelihood on each fold were computed. For MNIST we compare against other models of the real-valued (rather than binary) version of dataset.

of the Gaussians was obtained by cross validation on the validation set. This procedure was introduced in Breuleux *et al.* [8] and used for various generative models for which the exact likelihood is not tractable [25, 3, 5]. Results are reported in Table 1. This method of estimating the likelihood has somewhat high variance and does not perform well in high dimensional spaces but it is the best method available to our knowledge. Advances in generative models that can sample but not estimate likelihood directly motivate further research into how to evaluate such models.

In Figures 2 and 3 we show samples drawn from the generator net after training. While we make no claim that these samples are better than samples generated by existing methods, we believe that these samples are at least competitive with the better generative models in the literature and highlight the potential of the adversarial framework.

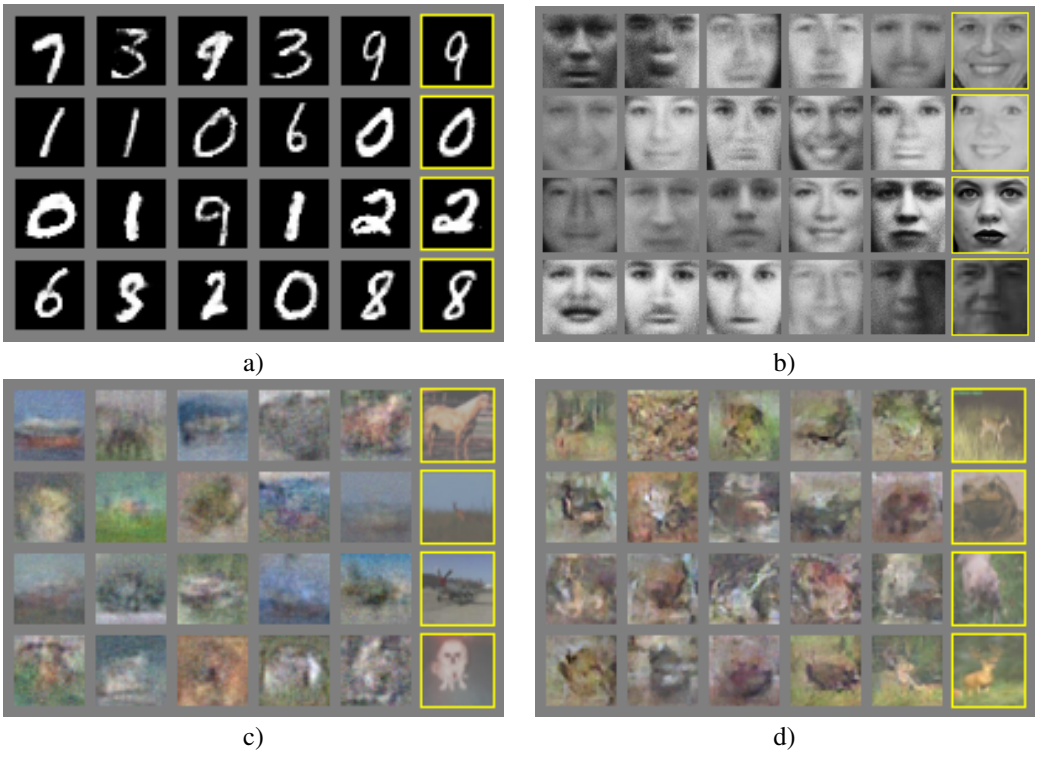

Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and "deconvolutional" generator)

Figure 3: Digits obtained by linearly interpolating between coordinates in $\boldsymbol{z}$ space of the full model.

	Deep directed graphical models	Deep undirected graphical models	Generative autoencoders	Adversarial models
Training	Inference needed during training.	Inference needed during training. MCMC needed to approximate partition function gradient.	Enforced tradeoff between mixing and power of reconstruction generation	Synchronizing the discriminator with the generator. Helvetica.
Inference	Learned approximate inference	Variational inference	MCMC-based inference	Learned approximate inference
Sampling	No difficulties	Requires Markov chain	Requires Markov chain	No difficulties
Evaluating $p(x)$	Intractable, may be approximated with AIS	Intractable, may be approximated with AIS	Not explicitly represented, may be approximated with Parzen density estimation	Not explicitly represented, may be approximated with Parzen density estimation
Model design	Nearly all models incur extreme difficulty	Careful design needed to ensure multiple properties	Any differentiable function is theoretically permitted	Any differentiable function is theoretically permitted

Table 2: Challenges in generative modeling: a summary of the difficulties encountered by different approaches to deep generative modeling for each of the major operations involving a model.

6 Advantages and disadvantages

This new framework comes with advantages and disadvantages relative to previous modeling frameworks. The disadvantages are primarily that there is no explicit representation of $p_g(\boldsymbol{x})$, and that D must be synchronized well with G during training (in particular, G must not be trained too much without updating D, in order to avoid "the Helvetica scenario" in which G collapses too many values of $\mathbf{z}$ to the same value of $\mathbf{x}$ to have enough diversity to model p_{data}), much as the negative chains of a Boltzmann machine must be kept up to date between learning steps. The advantages are that Markov chains are never needed, only backprop is used to obtain gradients, no inference is needed during learning, and a wide variety of functions can be incorporated into the model. Table 2 summarizes the comparison of generative adversarial nets with other generative modeling approaches.

The aforementioned advantages are primarily computational. Adversarial models may also gain some statistical advantage from the generator network not being updated directly with data examples, but only with gradients flowing through the discriminator. This means that components of the input are not copied directly into the generator's parameters. Another advantage of adversarial networks is that they can represent very sharp, even degenerate distributions, while methods based on Markov chains require that the distribution be somewhat blurry in order for the chains to be able to mix between modes.

7 Conclusions and future work

This framework admits many straightforward extensions:

1. A *conditional generative* model $p(\boldsymbol{x} \mid \boldsymbol{c})$ can be obtained by adding $\boldsymbol{c}$ as input to both G and D.
2. *Learned approximate inference* can be performed by training an auxiliary network to predict $\boldsymbol{z}$ given $\boldsymbol{x}$. This is similar to the inference net trained by the wake-sleep algorithm [15] but with the advantage that the inference net may be trained for a fixed generator net after the generator net has finished training.

3. One can approximately model all conditionals $p(\boldsymbol{x}_S \mid \boldsymbol{x}_{\not S})$ where S is a subset of the indices of $\boldsymbol{x}$ by training a family of conditional models that share parameters. Essentially, one can use adversarial nets to implement a stochastic extension of the deterministic MP-DBM [11].
4. *Semi-supervised learning*: features from the discriminator or inference net could improve performance of classifiers when limited labeled data is available.
5. *Efficiency improvements:* training could be accelerated greatly by divising better methods for coordinating G and D or determining better distributions to sample $\mathbf{z}$ from during training.

This paper has demonstrated the viability of the adversarial modeling framework, suggesting that these research directions could prove useful.

Acknowledgments

We would like to acknowledge Patrice Marcotte, Olivier Delalleau, Kyunghyun Cho, Guillaume Alain and Jason Yosinski for helpful discussions. Yann Dauphin shared his Parzen window evaluation code with us. We would like to thank the developers of Pylearn2 [12] and Theano [7, 1], particularly Frédéric Bastien who rushed a Theano feature specifically to benefit this project. Arnaud Bergeron provided much-needed support with LaTeX typesetting. We would also like to thank CIFAR, and Canada Research Chairs for funding, and Compute Canada, and Calcul Québec for providing computational resources. Ian Goodfellow is supported by the 2013 Google Fellowship in Deep Learning. Finally, we would like to thank Les Trois Brasseurs for stimulating our creativity.

References

[1] Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

[2] Bengio, Y. (2009). *Learning deep architectures for AI*. Now Publishers.

[3] Bengio, Y., Mesnil, G., Dauphin, Y., and Rifai, S. (2013a). Better mixing via deep representations. In *ICML'13*.

[4] Bengio, Y., Yao, L., Alain, G., and Vincent, P. (2013b). Generalized denoising auto-encoders as generative models. In *NIPS26*. Nips Foundation.

[5] Bengio, Y., Thibodeau-Laufer, E., and Yosinski, J. (2014a). Deep generative stochastic networks trainable by backprop. In *ICML'14*.

[6] Bengio, Y., Thibodeau-Laufer, E., Alain, G., and Yosinski, J. (2014b). Deep generative stochastic networks trainable by backprop. In *Proceedings of the 30th International Conference on Machine Learning (ICML'14)*.

[7] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

[8] Breuleux, O., Bengio, Y., and Vincent, P. (2011). Quickly generating representative samples from an RBM-derived process. *Neural Computation*, **23**(8), 2053–2073.

[9] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *AISTATS'2011*.

[10] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013a). Maxout networks. In *ICML'2013*.

[11] Goodfellow, I. J., Mirza, M., Courville, A., and Bengio, Y. (2013b). Multi-prediction deep Boltzmann machines. In *NIPS'2013*.

[12] Goodfellow, I. J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., Bergstra, J., Bastien, F., and Bengio, Y. (2013c). Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*.

[13] Gutmann, M. and Hyvarinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS'2010*.

[14] Hinton, G., Deng, L., Dahl, G. E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012a). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, **29**(6), 82–97.

[15] Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The wake-sleep algorithm for unsupervised neural networks. *Science*, **268**, 1558–1161.

[16] Hinton, G. E., Osindero, S., and Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, **18**, 1527–1554.

[17] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors. Technical report, arXiv:1207.0580.

[18] Hyvärinen, A. (2005). Estimation of non-normalized statistical models using score matching. *J. Machine Learning Res.*, **6**.

[19] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*, pages 2146–2153. IEEE.

[20] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[21] Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

[22] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *NIPS'2012*.

[23] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

[24] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. Technical report, arXiv:1401.4082.

[25] Rifai, S., Bengio, Y., Dauphin, Y., and Vincent, P. (2012). A generative process for sampling contractive auto-encoders. In *ICML'12*.

[26] Salakhutdinov, R. and Hinton, G. E. (2009). Deep Boltzmann machines. In *AISTATS'2009*, pages 448–455.

[27] Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge.

[28] Susskind, J., Anderson, A., and Hinton, G. E. (2010). The Toronto face dataset. Technical Report UTML TR 2010-001, U. Toronto.

[29] Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. In W. W. Cohen, A. McCallum, and S. T. Roweis, editors, *ICML 2008*, pages 1064–1071. ACM.

[30] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *ICML 2008*.

[31] Younes, L. (1999). On the convergence of Markovian stochastic algorithms with rapidly decreasing ergodicity rates. *Stochastics and Stochastic Reports*, **65**(3), 177–228.