Modular domain-to-domain translation network

Karatsiolis, Savvas; Schizas, Christos N.; Petkov, Nicolai

Published in:
Neural Computing and Applications

DOI:
10.1007/s00521-019-04358-8

Publication date:
2020

Neural Computing and Applications (2020) 32:6779–6791
https://doi.org/10.1007/s00521-019-04358-8

BRAIN INSPIRED COMPUTING &MACHINE LEARNING APPLIED RESEARCH-BISMLARE

Modular domain-to-domain translation network

Savvas Karatsiolis[1] · Christos N. Schizas[1] · Nicolai Petkov[2]

Received: 31 December 2018 / Accepted: 19 July 2019 / Published online: 26 July 2019

Abstract
Domain-to-domain translation methods map images from a source domain to corresponding images from a target domain. The two domains contain images from the same classes, but these images look different. Recent approaches use generative adversarial networks in various configurations and architectures to perform the translation. By using GANs, they inevitably inherit their problems like training instability and mode collapse. We propose a novel approach to the problem that does not use a GAN. Instead, it relies on an hierarchical architecture that encapsulates information of the target domain by using individually trained networks. This hierarchical architecture is then trained as one unified deep network. Using this approach, we show that images from the original domain are translated to the target domain both for the case when there is a one-to-one correspondence between the images of the two domains and for the case that such correspondence information is absent. We visualize and evaluate the translation from one information domain to the other and discuss the proposed model's relation to the conditional generative adversarial networks. We further argue that deep learning can benefit from the proposed hierarchical architecture.

Keywords Unsupervised learning · Auto-encoder · Feature mapping · Neural networks

1 Introduction

Domain translation aims at producing a pattern according to the way data is represented in the target domain, based on an input pattern coming from a source domain. The two domains contain instances of the same classes but are comprised of images that are generated in different ways and look different. Assuming a source domain Z and a target domain X, we need to find a mapping $M: Z \rightarrow \widehat{X}$ such that the output $\widehat{x} = M(z),\ z \in Z$ matches the distribution of X, An optimal mapping M translates Z to a domain $\widehat{X}$ that contains images similar to the images in X. Figure 1 shows a domain translation example where patterns of the Street View Home Numbers dataset [1] are translated to corresponding patterns from the MNIST dataset [2].

Domain translation is important in many application fields: artistic style image translation (converting an input image into an image that exhibits a certain drawing style), rendering images to a similar context and producing counterpart patterns for a given input. Besides such applications, domain translation is an interesting research field: Humans easily identify when the same data are represented in different ways, but the task proves very challenging to achieve in the machine learning framework. For example, humans can effortlessly identify a digit regardless whether it is handwritten on a paper or engraved on sand. On the contrary, machine learning algorithms are inefficient when tested on data that look different from the training data. Accomplishing near human performance on the domain translation task and promoting understanding of its underlying mechanisms may shed some light to human cognitive abilities like visual semantic understanding, imagination and concept transferring.

Several approaches for domain translation were developed in the recent years, and the great majority of them

✉ Savvas Karatsiolis
skarat01@cs.ucy.ac.cy

[1] University of Cyprus, University Avenue 1, Aglantzia, 2109 Nicosia, Cyprus

[2] Department of Intelligent Systems Group, Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, 9712 CP Groningen, The Netherlands

Fig. 1 An example of translating images from one domain to another. Both domains comprise of digits and have the same classes. The first row shows patterns from the SVHN dataset, and the second row shows corresponding patterns from the MNIST dataset. Each column forms a translation pair and consists of images that have a different appearance but belong to the same class

uses generative adversarial networks (GANs) [3]. For example, Kim et al. [4] use two generators, one for each of the domains. These generators learn the mappings from one domain to the other. This means that the model also learns the inverse mapping for reconstructing the input image from an image in the target domain. The input data and their mappings are fed to a discriminator that applies the adversarial loss. This coupled model based on two GANs is called DiscoGAN and is able to discover relations between two unpaired and unlabeled datasets.

Benaim and Wolf [5] proposed a method that does not require any inverse mapping for reconstructing the data of the input domain. Instead, their model learns a mapping that maintains the distance between a pair of samples. Their approach is named one-sided unsupervised domain mapping due to the absence of inverse mapping in their model. Benaim and Wolf [6] also proposed a method for translating an image from a previously unseen domain to a target domain. Interestingly, their approach considers only a single image from the source domain and is called one-shot domain translation. More specifically, they train a variational auto-encoder for the target domain. Another variational auto-encoder for the source domain is trained by only adapting the layers that are representationally closer to the source domain image.

A different approach for discovering the relations between two domains was proposed by Liu and Tuzel [7]. Their strategy is to learn a joint distribution of multi-domain images by enforcing a weight-sharing constraint on a pair of GANs that each generate images from one of the considered domains. The weight-sharing constraint allows the model to learn a joint distribution for the two domains without requiring tuples of corresponding image pairs. The model is called coupled GAN (CoGAN).

A more straightforward approach for image-to-image translation was introduced more recently by Isola et al. [8] and uses conditional GANs. While generating target-domain images conditioned on source domain images is not a new idea for GANs, Isola et al. applied some architectural modifications that enhance the model's performance. More specifically, they use a U-Net architecture [9] applying skip connections between the encoder and the decoder of the generator. They additionally use a Markovian discriminator which they call PatchGAN that applies the loss function at the scale of image patches. This approach mitigates the weakness of L_2 and L_1 losses on neglecting high-frequency crispness. Image-to-image translation with GANs produces good results but deals with an admittedly easier problem since every image pair carries very similar information.

The same problem is addressed by Zhu et al. [10]. Their proposed model (CycleGAN) deals with the problem without requiring aligned image pairs. This is done by ensuring that the output lies in the same low-dimensional embedding space. Such a property is achieved by the introduction of two cycle consistency losses that in addition to the translation from the source domain to the target domain, they also allow for the translation of the target-domain image back to the original input image. Finally, cross-domain image generation has been studied by Taigman et al. [11] by implementing a modified multiclass GAN that they called domain transfer network to produce an image that is relevant to an input image.

Apparently, state-of-the-art approaches implement domain translation through the use of GANs. While GANs are popular generative models due to their expressiveness, they suffer from instability due to unstable gradients and are often difficult to train. Mode collapse is another common problem occurring during training a GAN and results in a generator that outputs a single data point for the different modes of the domain. In the extreme case, the generator collapses to a single output mode which makes the model practically useless since it can only generate extremely similar patterns regardless the input. The problem becomes even greater when more than one GANs are used in a single architecture, as most of the domain translation state-of-the-art models do. Using multiple GANs in a model requires a great deal of effort to synchronize their training and tune them into cooperating in a productive manner.

We propose an alternative approach for domain translation that does not rely on GANs and thus avoids the problems accompanying their training. Our method integrates the operation of distinct network structures in a unified architecture that performs the task efficiently, while dealing with both cases of having a 1:1 and 1:M correspondence of input- and target-domain images. It also

applies the same representational space for both domains in order to efficiently learn the mapping from the input domain to the target domain.

The paper is structured as follows: Sect. 2 describes the proposed model and its architecture, while Sect. 3 presents the experimental results and the implementation details of the architecture. Finally, Sect. 4 discusses the experimental results and aspects of the model and Sect. 5 contains the conclusions.

2 Proposed model

The proposed model aims at the translation of patterns from a source domain to patterns belonging to a different domain such that both domains share the same pattern classes. For our experiments, we use two widely known datasets, MNIST [2] and Street View Home Numbers (SVHN) [1]. We also create three synthetic datasets from the MNIST dataset: the noisy MNIST (MNIST_Noisy), the rotated MNIST (MNIST_Rot) and the rotated plus background image MNIST (MNIST_Rot_Bck). The MNIST dataset consists of gray-scale handwritten digits each centered in a 28×28 pixel bounding box having a black background. The MNIST dataset contains 60,000 training images and 10,000 testing images. The SVHN dataset contains digits and numbers obtained from house numbers in Google Street View images and contains more than 73,257 training images and 26,032 testing images. For our model, the MNIST dataset represents the target domain, while the source domain is either the SVHN dataset or one of the distorted variants (synthetic datasets) of the MNIST dataset. Having a well-performing classifier for the target domain at hand is important for the proposed model because, during training, it provides vital information for performing the translation from one domain to the other. Domain-to-domain translation may deal with two scenarios:

1. Cross-domain pattern correspondence is the case of having the same information expressed by very similar but still different domains (patterns have a 1:1 correspondence). For example, for a target-domain pattern x, the input domain may contain a modified version of this pattern x' which is obtained after x has been altered by a transformation t or distorted by a signal n, that is, $z = x' = t(x) + n$.
2. Cross-domain categorical correspondence is the case of having samples from two quite different domains having the same pattern classes. This kind of relation will be referred to as 1:M correspondence because one sample from the target domain is related to many samples of the input domain because of class resemblance.

The 1:1 correspondence is identified in the case of having the MNIST domain as the target domain and a distorted variant of MNIST (e.g., rotated MNIST) as the input domain. An example of the 1:M correspondence is the case of using the SVHN as the input domain because the images of this specific dataset are quite different from the images of the MNIST domain in the sense that the digits to be recognized are accompanied with other digits placed on their sides, there is color information and there is a lot of distortion, blurring, spatial shifting etc. Still, the pattern classes are the same for both datasets.

2.1 Modular domain-to-domain translation architecture

We propose a model that is comprised of three distinct components: the input-domain to target-domain mapping network, the decoding of this representation to the target domain and a well-performing target-domain classifier. All these stages (representation, decoding and classifier) are embedded into a deep architecture that is trained with backpropagation to produce a translation network from the input domain to the target domain. However, the three distinct stages must be trained before being embedded in the final deep architecture. This pre-training strategy places the unified architecture in the vicinity of a good initial training state that maintains the objective function and prevents overfitting due to the increased number of model parameters. It also prevents underfitting caused by the vanishing gradients phenomenon caused by the deep architecture. The detailed steps for the construction of the model are shown in Tables 1 and 2. Figure 2 shows the model architecture and the distinguishable stages that we train before their unification into a deep network architecture.

Every distinct stage of the model is not restricted to a specific architecture and can be shallow or deep according to the application's complexity. However, the width and depth of every stage affects the final model size accordingly. As soon as we construct the individual networks, we embed them into a deep architecture and further train them as a unified network with tiny learning rates for the final two stages in order to preserve the information transferred from the pre-training procedure. If we train the target-domain-dependent stages with anything but tiny learning rates, then the network will not necessarily maintain its prior knowledge. During the experiments, we assigned these learning rates a value of $1e^{-8}$. Since we initialize the last stage to be a well-performing target-domain classifier allowed to undergo only slight changes due to a very small learning

Table 1 Steps for constructing the deep domain-to-domain translation network

Assuming an input domain dataset of the form $\{(z_1, y_1), (z_2, y_2), \ldots, (z_M, y_M)\}$ and a target domain dataset of the form $\{(x_1, y_1), (x_2, y_2), \ldots, (x_C, y_C)\}$, with z_i, x_i being the individual patterns of the source and target domains respectively and y_i being their label. For 1:1 pattern correspondence between Z and X, it should also hold that $M = C$.

1. Construct an auto-encoder for the target domain (shown as (a) in Figure 2) such as $f: X \to L$, $g: L \to X$ with f being the encoder, g the decoder and $L \in \Re^K$, for some K, is the latent representation. The auto-encoder should use sigmoid functions to maintain a direct probabilistic interpretation for the latent representations.
2. Train a representation network (shown as (b) in Figure 2) $T: Z \to R$ with the M input domain patterns with cross-entropy loss function. The training targets are binary vectors $\vec{r}_m$ sampled from the latent representations calculated in step (1) such as $r_m^k \in \{0,1\}$. The sampling is performed for every mini-batch of the training. Table 2 describes this step in detail.
3. Train a well performing classifier D for the target domain (shown as (c) in Figure 2) such as $D: X \to Y$
4. Create a deep architecture model by embedding the model from step (2), the decoder g from step (1) and the classifier D from step (3) in this exact order as shown in Figure 2.
5. Train the unified model $F = \{T, g, D\}$ with tiny learning rates for stages g and D such as $F: Z \to Y$
6. The final translation model S is formed by discarding the classifier D stage and keeping $S = \{T, g\}$

Table 2 Training algorithm for the representation network

Assuming an input domain dataset of the form $\{z_1, z_2, z_3, z_4, \ldots, z_M\}$, the corresponding target domain latent representations $\{l_1, l_2, l_3, l_4, \ldots, l_M\}$ as calculated from step (1) of Table 1 and the binary form of these representations as the formal problem targets $\{t_1, t_2, \ldots, t_M\}$ such as

$$t_i = \begin{cases} 1 & l_i \geq 0.5 \\ 0 & l_i < 0.5 \end{cases}$$

Repeat until convergence

Repeat until the whole dataset is examined

1. Sample a k-size mini batch such as $\{z_1, z_2, \ldots, z_k\}$ associated with binary targets $\{r_1, r_2, \ldots, r_k\}$ sampled from the probabilities $\{l_1, l_2, \ldots, l_k\}$ such as

$$r_i = \begin{cases} 1 & p \leq l_i \\ 0 & p > l_i \end{cases}, \quad p \sim U(0,1)$$

2. Perform batch-normalization training with cross-entropy loss on the current mini batch and $\{r_1, r_2, \ldots, r_k\}$ as targets.

End

3. Evaluate network for dataset $\{z_1, z_2, z_3, \ldots, z_M\}$ with targets $\{t_1, t_2, \ldots, t_M\}$.

End

rate, we expect it to maintain a great portion of this ability after the training of the model is over. Furthermore, we expect it to guide (through the backpropagation of its gradient information) the early stages of the unified architecture in adapting their feature mapping in such a way that the constructed features are a match for the last stage feature space. Consequently, this drives the early stages of the architecture to figure out a way to construct features that are related to the target domain. We did not apply batch normalization in the final model because this could destroy the pre-training and provoke the loss of the target-domain information encapsulated in the network. Training the stages provided by the target domain with learning rates that are not tiny can have the same effect. In turn, we trained the first three hidden layers with a small learning rate and the next layers with a tiny learning rate. The final domain

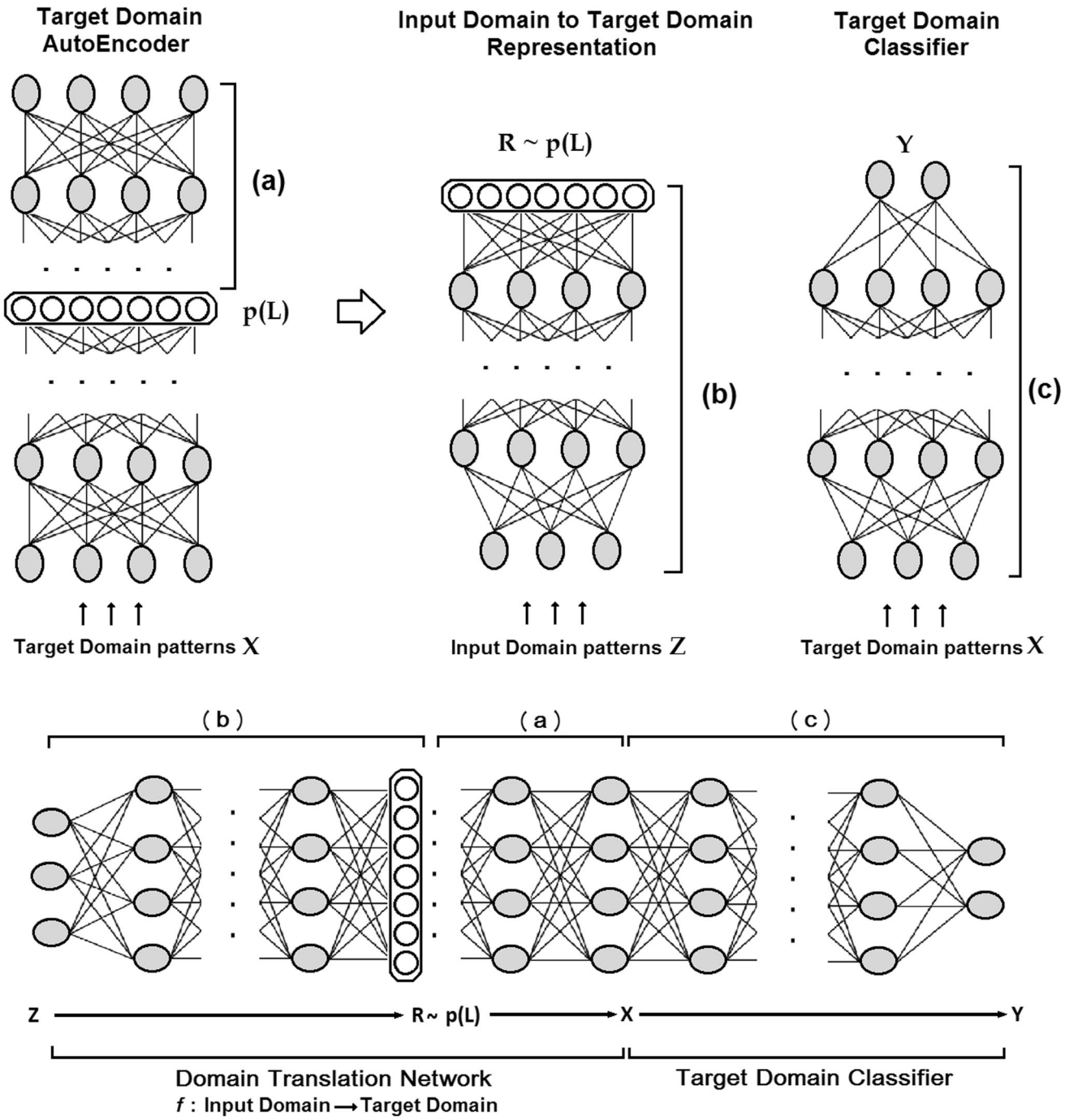


Fig. 2 The proposed model architecture. Three distinct networks are pre-trained and then placed into the final deep architecture: **a** a target-domain auto-encoder, **b** an input-domain to target-domain representation network trained on cross-entropy loss of sampled binary values from the latent variables of the target-domain auto-encoder and **c** a well-performing pre-trained classifier of the target domain. After these stages are placed as shown in the final deep architecture model, they are trained with very small learning rates in stages (a), (c) in order to avoid destruction of the knowledge transferred from the target domain. The final model effectively consists of the domain translation network formed by stages (a), (b). At the output of the domain translation network, just before the final classifier, it is expected to observe patterns belonging to the target domain

translation model's performance could be enhanced by adding the available target-domain auto-encoder as an extra stage at the output of the model. This approach produces sharper and more detailed images, but we did not use it in the experimental results because the main purpose is to evaluate the proposed model in its basic form. Performance enhancements and output image improvements may be explored and implemented in future work.

2.2 The input-domain representation network

For the construction of the representation network (Fig. 2b) which is the first stage of the deep unified model, a target-domain auto-encoder must be trained first (Fig. 2a). This auto-encoder has an encoder and a decoder stage. The output of the encoder provides the latent variables of the target domain's patterns. We used these representations as probabilistic targets to train the input-domain representation network. More specifically, we used the latent representations produced by the target-domain auto-encoder to sample binary targets in order to train a model to map the input domain to the desired target-domain features. Sampling the targets of the mapping model from the latent representations of the target domain enables the representation network to learn the most essential elements required to reconstruct the target domain. In essence, the input representation network translates the source domain to the fundamental structures that make up the target domain. In other words, this approach trains the representation network into a state in which it processes knowledge of the target-domain data manifold. After we trained the representation network, we placed it in the final architecture. Starting the training of the deep model from the point that the initial stage of the architecture lies around the target-domain manifold highly promotes learning of the domain translation task. However, for the latent representations to provide quality information for the input-domain representation network they should be free of constant values. Some of the target-domain latent codes are always 1 or 0 for all the target-domain patterns. In practice, we considered two threshold values, 0.99 and 0.01. Any neuron having an output greater than 0.99 or smaller than 0.01 for all input patterns is considered to have a constant output. Maintaining these constant values is of course useful for decoding the digits through the decoder stage of the auto-encoder but conveys no special information in terms of training a model that maps the input-domain patterns to the latent representations of the target domain. On the contrary, these always switched on or off neurons will constantly provide the same output target values after latent code sampling (either 1s or 0s). We discovered that this reduces the quality of the representation network by compromising the impact of other meaningful latent representations that provide different target values through sampling. It is important for the input representation network to stay in the vicinity of the target-domain manifold. This means that the latent constant-valued variables must be exempted from these representation codes but the decoder stage should keep their effect in order to sustain its data reconstruction

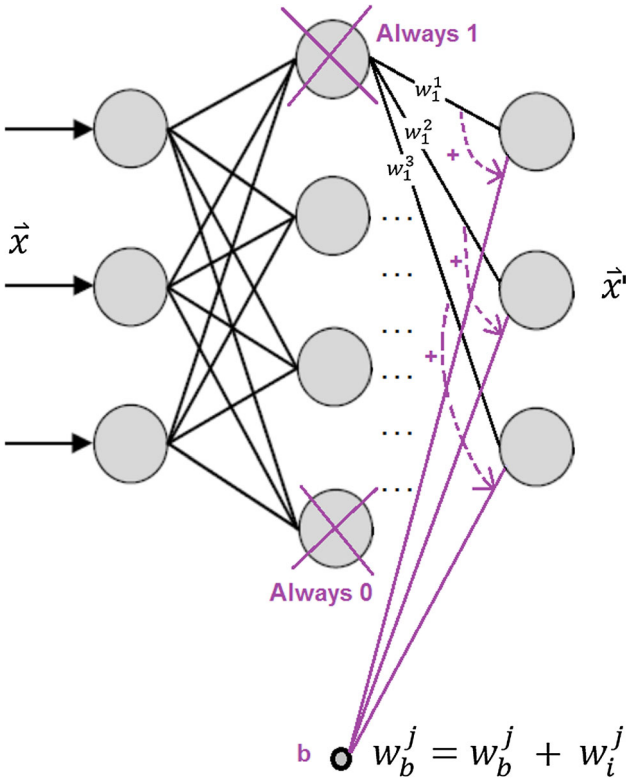


Fig. 3 Latent variables of the target-domain auto-encoder that are always 1 or 0 are omitted. The case of constant 0 neuron output is trivially omitted since the neuron has no effect on the model's functionality. The effect caused by the neurons that have constant output 1 over all target-domain patterns may be embedded in the bias terms of the next layer neurons and thus the decoder's functionality is kept intact. This is accomplished by adding the weights of the corresponding neurons to the bias weight of the affected output neurons which results in an equivalent model function

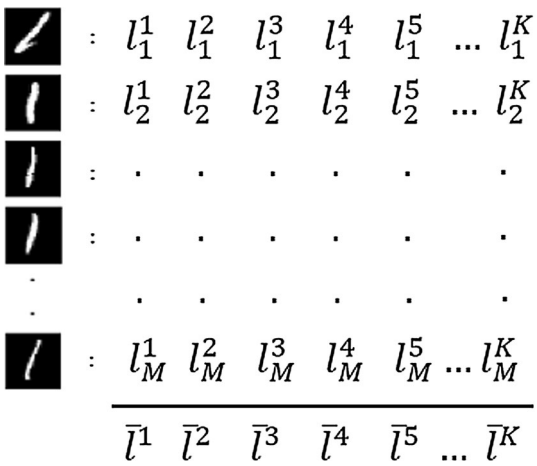


Fig. 4 Aggregating the auto-encoder's latent variables l_i^j for various instances of digit "1" in the dataset. The index of each digit instance is represented by *i*, while each element of the latent representation is indexed by *j*. Different writing styles produce different latent vectors whose elements are averaged in order to calculate a mean representation for the specific class. This representation is interpreted as a vector of probabilities and each corresponding element is sampled to form binary targets for the input-domain representation network

ability. It turns out that these constant-valued variables can be safely removed from the latent representations, while their effect in the calculation of the output activations is maintained and thus the decoder function is preserved intact. To achieve this, the latent variables that are always 1 have their output weights added to the bias input of the affected neuron in the next layer, while constant 0 latent variables are just ignored since they have no impact to the calculation of the output activation. This concept is shown in Fig. 3. The specific modification yielded faster and better training for the input-domain representation network.

Once the latent variables that maintain a constant value for all input patterns are removed, the remaining variables provide the probabilistic target-domain representations. More specifically, we sample these variables for obtaining binary targets for the input-domain representation network. For the cases when there is 1:1 correspondence between target-domain examples and input-domain examples, the training of the representation network is straight forward: just sample the latent variables' vector $\vec{l}_m$ of each target-domain example $\vec{x}_m$ and create binary targets $\vec{r}_m \sim p\left(\vec{l}_m\right)$, $r_m \in \{0, 1\}$ for the corresponding input-domain example in each mini-batch. In the case that the input domain and the target domain are not related with a 1:1 pattern correspondence, the probabilistic target-domain representations require a further processing step to be finalized. This further step is necessary because under the 1:1 pattern correspondence regime, the same information is just expressed in different but similar domains, also viewed as 'information channels' that are linked by simple transformations or domain-transition operations. This means that same-source information mappings are also linked in a pairwise (input to target domain) fashion. That is why sampling the latent representations of the target domain is sufficient to construct the feature vectors of the representation network. On the contrary, when a 1:1 pattern correspondence does not exist between the two domains of the ongoing translation, using latent representation statistics in a pairwise fashion may be misleading and difficult. First of all, there is no easy way for deciding an appropriate cross-domain pairing between the patterns, since cross-domain similarity metric construction is itself a problem of its own merit.

Under these circumstances, we use a more general association: the mean of the target-domain latent variable vectors belonging to the same class. In other words, the latent representations of the target-domain examples are aggregated based on their class and a mean representation is calculated for each class. These mean representations define the probabilistic targets for training the input-domain representation network. For example, assuming the MNIST to be the target domain and SVHN to be the input domain, the 1:1 correspondence between the patterns is absent. In order to create a more general correspondence, namely 1:M, we aggregate the latent representations of the target-domain patterns per problem class and calculate their mean value. The mean value represents the latent probabilities for each problem class, and we use these probabilities to sample the binary target values for the input-domain patterns according to their class during training of the representation network. The aggregation of the latent representations for the specific class "1" in MNIST is demonstrated in Fig. 4.

Assuming there are M target-domain examples belonging to a certain problem class C and the auto-encoder calculates a K number of latent variables l, then the binary

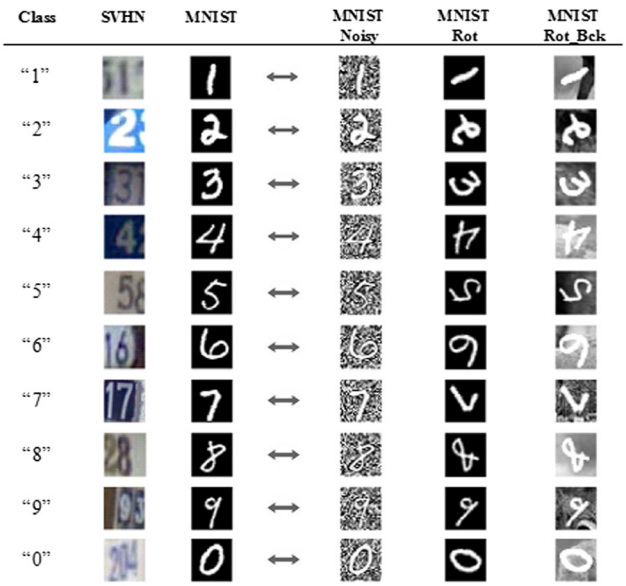


Fig. 6 Samples from the SVHN, MNIST, MNIST_Rot, MNIST_-Noisy and MNIST_Rot_Back datasets. The arrows represent a 1:1 correspondence between the patterns of MNIST and its variants. This correspondence is not applicable when a domain translation from SVHN-to-MNIST domain is applied or vice versa

Fig. 5 Samples of images from which some 28 × 28 patches are extracted to use as background texture for the MNIST_Rot_Bck dataset

target's sampling probability y for a specific latent variable k for the input-domain representation network is

$$p(y^k|C) = \frac{1}{M}\sum_{m=1}^{M} l_m^k = \overline{l^k}$$

After we calculate these mean representations, we sample them for each mini-batch of the training procedure of the input-domain representation network. This general mapping surprisingly works better than expected and performs similarly to the more informative 1:1 correspondence. Matching a target-domain example to an input-domain example provides information that should promote learning. However, for the specific translation model it suffices to initialize the input-domain representation network with a mapping function which lies at the vicinity of the target-domain manifold. Consequently, the more general 1:M mapping between input-domain and target-domain patterns performs equally well.

3 Results

3.1 Datasets

Having two forms of information describing the same or similar observations is not a rare situation. For example, a digit recognition problem is described by two datasets, the MNIST and the SVHN (Street View Home Numbers). In the case of the SVHN-MNIST pair, every example in one of the datasets can be associated with many examples in the other dataset linked by their common class label. Thus, their correspondence is of a general nature (simply categorical) since it is difficult or meaningless to link the examples with a more detailed relation like style, orientation, displacement, etc. In other words, there is not a one-to-one correspondence between the examples of the MNIST dataset and the examples of the SVHN dataset. In order to explore such a one-to-one correspondence between the examples of the two domains, we introduce three datasets which are corrupted versions of the original MNIST dataset. These MNIST variants are listed below:

- Rotated MNIST (MNIST_Rot): The original digits are rotated by an angle generated randomly in the range 0 to 2π.
- Noisy MNIST (MNIST_Noisy): Each pixel in the original image is replaced with 20% probability by a random number uniformly sampled from the range 0 to 1.
- Rotated + Background image (MNIST_Rot_Bck): The digits are randomly rotated as in rotated MNIST and the background is replaced with a black and white image patch. These 28×28 patches are extracted from random nature pictures downloaded from the Internet and screened for having a minimum level of pixel variation. More specifically, we discard patches that have a variance less than 0.01. Samples from the random images used to provide these patches are shown in Fig. 5.

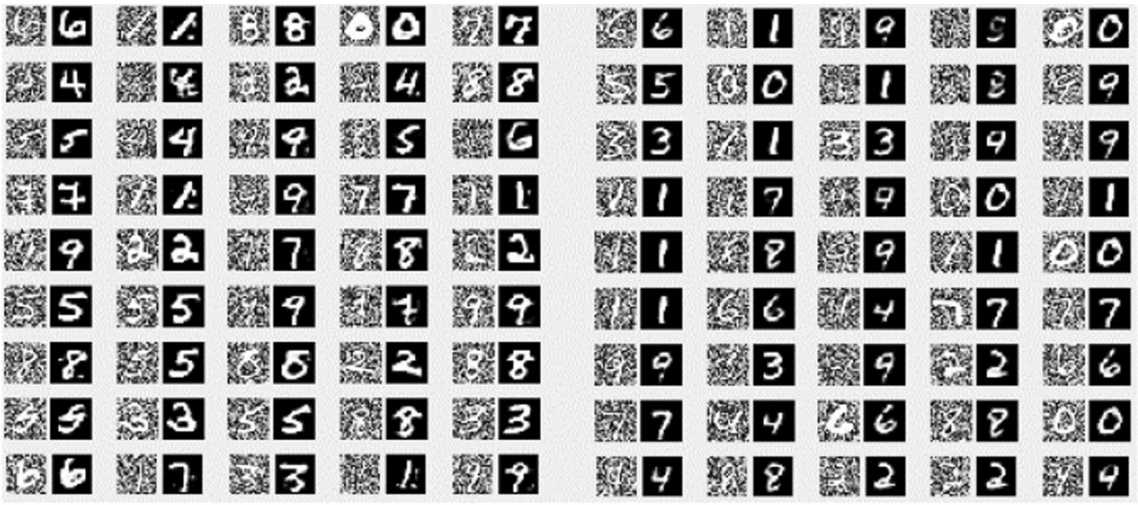
Fig. 7 Random samples of the **a** MNIST_Noisy, **b** MNIST_Rot, **c** MNIST_RotBck and **d** SVHN domain-to- MNIST domain translation. Each pair of images is formed by the input-domain pattern on the left and the model's output on the right. The left side of the figures **a**–**c** shows the 1:1 cross-domain pattern correspondence case while the right side shows the translation results for the 1:M case. For the case of the SVHN-to-MNIST translation (**d**), the 1:1 cross-domain pattern correspondence is not applicable

While creating the MNIST variant datasets, we noted the image correspondence between the original image and its variant and thus is available during the training of the translation model. Figure 6 shows some samples from the available datasets and indicates the direct 1:1 correspondence between MNIST and its distorted versions. The goal is to train models that can translate a pattern belonging to either the SVHN, the MNIST_Noisy, the MNIST_Rot or the MNIST_Rot_Bck domain to an appropriate pattern belonging to the MNIST domain. Additionally, for each MNIST variant, we study the two correspondence relations (1:1 and 1:M).

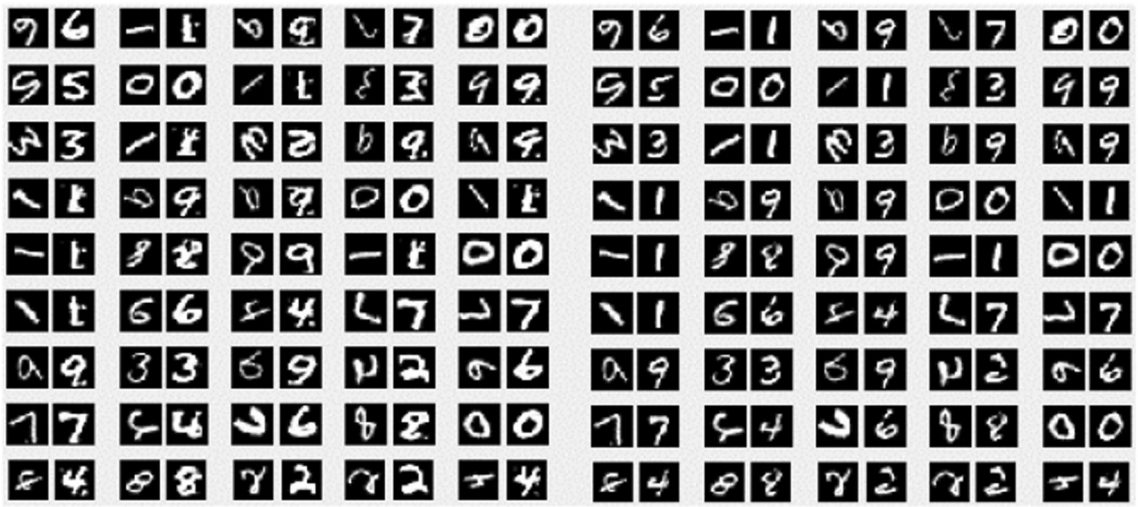
We explore two experimental pathways: domain translation with cross-domain pattern correspondence and domain translation without cross-domain pattern correspondence. We study both pathways for the MNIST variants translation to the MNIST domain, while the translation of the SVHN domain to the MNIST domain is performed only according to the latter pathway. According to the proposed methodology, we train the MNIST auto-encoder making use of the whole 60,000 available patterns in the dataset. Next, we express the MNIST dataset by the latent representation of the auto-encoder and all constant-valued variables (1 or 0) are removed by making the necessary bias weights' adjustments for the decoder stage where necessary. Furthermore, we calculate the mean latent representations per class for the case of the SVHN-to-MNIST translation. Finally, we place the individual stages defined by the proposed model in a unified architecture which is trained with a tiny learning rate for the final stages.

3.2 The target-domain auto-encoder

For the construction of the MNIST auto-encoder, we use a binary restricted Boltzmann machine with 1000 neurons in the hidden layer, providing a latent representation of 1000 variables. Since the MNIST image size is 28×28, the



(a)



(b)

(c)

(d)

Fig. 7 continued

auto-encoder has an architecture of 784-1000. During the training, we applied a weight decay of $\lambda = 1e^{-5}$ and used the full MNIST dataset comprising of 60,000 samples. We next unfolded the trained auto-encoder according to its encoding and decoding parts in an architecture of 784-1000-784 with the weights of the second layer (decoder) being the transposed weights of the first layer (encoder).

3.3 The target-domain classifier

Training on the full 60,000 patterns, MNIST dataset (50,000 patterns for training and 10,000 for evaluation) provides classification results that exceed 99% for the 10,000 patterns test set. Hinton et al. [12] reported a successful classification rate of 99.21% by using dropout and pre-training of the network layers with restricted Boltzmann machines. Rasmus et al. [13] used the ladder network (semi-supervised learning) and obtained results that were slightly better, namely 99.43%. We trained a 784-800-800-10 MNIST classifier with dropout and used it in all experiments. The classifier's hidden layers contain rectified linear units (ReLUs), and the final layer is a soft-max function. We accomplished a classification success rate of 99.17% on the 10,000 patterns of the test set which is comparable to the state-of-the-art results for the MNIST dataset.

3.4 The input representation network

The latent representations for the MNIST target domain consisted of 1000 variables. After exempting the constant-valued latent representations only 922 remained. We used the latent variables provided by the target-domain auto-encoder to train a 2000-2000-922 network, with the output stage corresponding to the latent representations $L \in \Re^{922}$. For the distorted MNIST datasets, the input images are of size 28 × 28 so the network input has a size of 784. For the SVHN case, the input image is 32 × 32, so we used a network input layer of size 1024. The hidden layers neurons use rectified linear activations (RLUs), and the output layer units are sigmoidal. We trained the network with batch normalization and mini-batches of 1000 patterns, a learning rate of $\lambda_w = 0.5$, and a tiny weight decay since the targets' sampling occurring on every mini-batch creation adds a lot of gradient noise. We used two metrics to evaluate the performance of the network during training: the cross-entropy loss based on the targets and the actual outputs and the precision score of both states (1 or 0) of the binary outputs. The second metric is rather necessary in order to detect misleading performance scores obtained by matching many of the outputs on one binary state while heavily neglecting the other state.

3.5 Constructing and training the unified deep architecture

After the unified model resulted from the embedding of the various stages, we trained it with mini-batch gradient descent. According to the networks used to form the model, the final architecture for the MNIST variants is 784-2000-2000-922-784-800-800-10 and for the SVHN is 1024-2000-2000-922-784-800-800-10. As expected, every layer size reflects the underlying individual network embedded in the architecture, for example, the fourth hidden layer of both models is of size 784 because it lies at the position where the decoder part of the auto-encoder is placed and thus it must have an output equal to the MNIST image size. The third hidden layer of the networks reflects the final number of the latent variables of the target-domain auto-encoder after the constant-valued variables are removed.

Finally, the last two hidden layers correspond to the architecture of the target-domain classifier. We trained the first three hidden layers with a learning rate of $\lambda = 0.05$, and for the next layers, we used a learning rate of $\lambda = 1e^{-8}$. The final results are shown in Fig. 7 for the MNIST_Noisy, MNIST_Rot, MNIST_RotBck and SVHN datasets, respectively. For the MNIST variants, the left side of the figures shows the results of a model trained with the mean latent class representations. For the SVHN domain, the 1:1 pattern relation is not applicable.

According to the results, the input-domain patterns are translated to a pattern in the target domain that belongs to the same class and has an image quality that generally reflects the quality of the input image. All models take advantage of the existence of prior information on how to feature map the input domain to the target domain and seem to build a concept of what defines a MNIST sample. This prior knowledge is embedded to the model through the pre-trained input-domain representation network, the embedded target-domain decoder stage and the target-domain classifier lying in the last stage of the model. This information guides the training of the initial stages of the network toward learning how to translate an image coming from a different domain to the specific format. For the specific implementation which uses the MNIST domain as the target domain, this means disposing the noise in the image as efficiently as possible, reconstructing or rearranging the digit areas that are lost or modified and maintaining a uniform dark background. None of this information was given to the model explicitly, but it is implied by the embedded stages that were trained based on the target domain. The results are particularly interesting for the SVHN-to-MNIST translation setup since the model learns implicitly an efficient way to reconstruct the MNIST format of the number in the middle of the image ignoring any numbers appearing at its sides. Additionally, this model trains itself on dealing with distorted digit images and many variations of the original patterns. During experimentation, it was noted that the quality of the target-domain auto-encoder and the training of the input-domain representation network were more crucial for the performance of the final model than extreme tuning of the final stage classifier. Another, rather unexpected, observation extracted from the results is the fact that cross-domain pattern correspondence is not as advantageous as expected. The resulted quality of both experimental pathways (1:1 correspondence and 1:*M* correspondence) is not that different which suggests that during training of the input-domain representation model, an adequate approximation of the target-domain manifold is sufficient in terms of enabling the unified architecture to learn how to perform the domain translation.

4 Discussion

We have introduced a method for performing cross-domain pattern translation which transfers a pattern from one domain to a similar domain that spans the same classification categories. Figure 8 compares our model's results for the translation of the MNIST_Rot dataset to the MNIST dataset with the results of a CoGAN [7] trained to output MNIST images rotated by 90° specifically. Our model produces at least comparable results while performing a significantly harder task since the rotation of each pattern is not fixed but varies in the range of 0 to 2π.

Figure 9 shows the results of our model, CycleGAN [10] and one-sided unsupervised domain mapping [5] when performing translation from the SVHN domain to the MNIST domain.

The results are also interesting from another point of view: The stages of the model that perform the domain translation and the classifier at the final stage of the unified network resemble the mechanisms of a conditional GAN. The former group of stages represents the GAN generator and the latter represents the GAN discriminator that outputs the K problem classes. In respect to Odena's semi-supervised GAN [14] which implements $K + 1$ discriminator outputs, the proposed model omits the fake image output class. This output is rather safely omitted because the generator is pre-trained to produce images in the vicinity of the domain information format and is prevented from deviating away from it since the training gradients come from a well-performing classifier acting on that domain. The concept served by the feature mapping property added to the objective function of Tim Salivan et al. method is also served by the pre-training of the input-domain representation network proposed by our approach: Instead of enforcing a similarity on how an intermediate layer of the discriminator is mapping real and generated information, our model consolidates this representation through the pre-training process, the embedding of the decoder stage of the target-domain decoder and the tiny learning rate applied on this stage. Extending this rational, the proposed model is acting similarly to a conditional GAN with $K + 1$ outputs, starting from an advanced training point, after the generator has started performing "reasonably" by producing images that should mostly be classified as belonging to one of the K problem classes. Hypothetically, the discriminator is constantly fooled by the generator in identifying one of the available K classes for every fake image it examines. Consequently, the fake output is omitted and not considered as an output option. Jost Tobias Springenber [15] also uses K output classes for training a GAN in an unsupervised or semi-supervised manner by omitting the fake image output. His strategy is

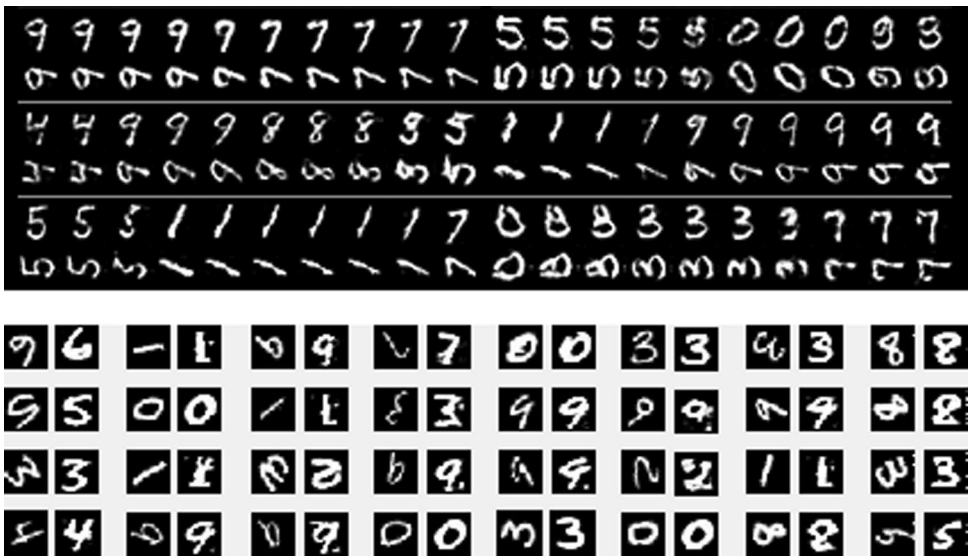


Fig. 8 Comparison of the proposed model to the CoGAN for the rotation operation. The top figure shows the results obtained by CoGAN with the top row of each group containing unrotated MNIST patterns while the second row of each group shows their counterpart rotated by 90°. The bottom image shows the results of our model when trained on the rotation-inversion task with the MNIST_Rot dataset. Our model performs a more complicated task because the rotation applied is not of a constant value like in CoGAN but significantly varies in the range $[0, 2\pi]$

Fig. 9 The results of translating SVHN domain patterns to the MNIST domain for the CycleGAN (top left), the one-sided unsupervised domain mapping (top right) and our model (bottom figure). The results of our model are at least comparable to the other methods' results

to train an artificial image generator which produces fake images that seem real and uses a uniform distribution of samples in terms of their associating label. At the same time, the discriminator must be trained to perform well on real data, to raise classification uncertainty when dealing with fake data and to choose the output class uniformly. From the requirements, it is obviously assumed that the model deals with uniform class priors. Our model complies with all the requirements stated above both for the generator and for the discriminator. Uniform distribution of

sample generation is satisfied by the generator due to equal class priors. The discriminator also satisfies this requirement and the one for performing well on real data because of its pre-training on the target-domain data. The raised uncertainty condition for the discriminator is naturally satisfied due to the noise injected during sampling of the latent probabilities. The proposed model also shares some theoretic principles with the semi-supervised GAN described by Odena [14] which uses $K + 1$ discriminator outputs.

An obvious modeling deviation of the proposed model from GANs is the absence of a noise generator at the input. Noise is important for the unconditional GAN for supplying the necessary variance at the input of the model. For conditional GANs, to which the proposed model is parallelized, this noise is not critical or even necessary since there is enough input variation due to the input-domain images applied to the network and are acting like a condition for the generative process. This is also supported by Isola et al. [13] and Mathieu et al. [16].

Besides the conditional GAN parallelization, there is another notable characteristic of the proposed model. It suggests that deep networks perform well when the layers obey an hierarchy of functionality. Of course, this is not a new idea since modern deep networks are built upon a function-specific layer architecture with various types of layers (convolutional, pooling and fully connected layers). However, a stricter sectional embedding paradigm in the form proposed might worth further investigation and experimentation. Embedding domain information to a model in the form of whole network blocks may produce networks that learn in a more efficient way. It could also provide the structural foundation of combining information from many similar domains to construct high-level concepts that are transferable between these domains and are used to build more sophisticated models.

5 Conclusions

This paper presents a novel approach for performing domain translation without incorporating a GAN in the model architecture like the great majority of the previously proposed models do. Our approach uses an hierarchical architecture comprised from individually trained modules. The architecture is trained as a whole (fine-tuned) and is able to achieve results that are at least comparable to the ones provided by the state-of-the-art models that use one or more GANs in their architecture. By not using GANs, the proposed architecture avoids deficiencies related to the training of GANs such as high instability and mode collapse.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng AY (2011) Reading digits in natural images with unsupervised feature learning. In: NIPS workshop on deep learning and unsupervised feature learning 2011
2. LeCun Y, Corinna C, Burges C (1998) MNIST handwritten digit database. The MNIST Database. http://yann.lecun.com/exdb/mnist/. Accessed 16 Feb 2019
3. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. Adv Neural Inf Process Syst 27:2672–2680
4. Kim T, Cha M, Kim H, Lee JK, Kim J (2017) Learning to discover cross-domain relations with generative adversarial networks. In: Proceedings of the international conference on machine learning (ICML)
5. Benaim S, Wolf L (2017) One-sided unsupervised domain mapping. In: Proceedings of the 31st international conference on neural information processing systems, USA
6. Benaim S, Wolf L (2018) One-shot unsupervised cross domain translation. In: Proceedings of the 32nd international conference on neural information processing systems, USA
7. Liu M-Y, Tuzel O (2016) Coupled generative adversarial networks. In: Advances in neural information processing systems, vol 29, pp 469–477
8. Isola P, Zhu J-Y, Zhou T, Efros A (2017) Image-to-image translation with conditional adversarial networks. In: IEEE conference on computer vision and pattern recognition (CVPR)
9. Ronneberger O, Fischer P, Brox T (2016) U-Net: Convolutional networks for biomedical image segmentation. CoRR, vol abs/1505.0, 2015
10. Zhu J-Y, Park T, Isola P, Efros AA (2017) Unpaired image-to-image translation using cycle-consistent adversarial networks. In: 2017 IEEE international conference on computer vision (ICCV)
11. Taigman Y, Polyak A, Wolf L (2016) Unsupervised cross-domain image generation. CoRR, vol abs/1611.0
12. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15:1929–1958
13. Rasmus A, Berglund M, Honkala M, Valpola H, Raiko T (2015) Semi-supervised learning with ladder networks. In: Advances in neural information processing systems, pp 3532–3540
14. Odena A (2016) Semi-supervised learning with generative adversarial networks. In: ICML
15. Springenberg JT (2016) Unsupervised and semi-supervised learning with categorical generative adversarial networks. ICLR 19:11
16. Mathieu M, Couprie C, LeCun Y (2015) Deep multi-scale video prediction beyond mean square error. ICLR 2015:1–14