SVHN-to-MNIST translation using CycleGANs

Adriano Cardace
TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY
a.cardace@student.tue.nl

Abstract

Image-to-image translation is a computer vision problem in which we want to convert one image that belongs to a particular domain to another domain. CycleGANs have been used and investigated for this task, and amazing results have been achieved with this approach. In this project, we would like to test the effectiveness of this strategy by performing image-to-image translation between two popular datasets: SVHN and MNIST. We start with two simple methods that share the same architecture but with different loss functions. Then we propose a third architecture that extends the baseline with residual blocks. We show that this simple addition helps to achieve a better score. We provide both qualitative and quantitative results.

1. Introduction

Image-to-image translation is a computer vision problem in which the goal is to learn a mapping between an input domain and an output domain. A typical example is to transform a horse into a zebra. Ideally, we would like to perform this transformation preserving other features, namely everything that is not directly related to the input domain. Again in the horse-zebra example, we want to maintain untouched information such as the background of the image (see 1 from [9]). In this paper we show the effectiveness of CycleGANs [9] for this problem. The real power of this method is the ability of learning a bidirectional mapping without the needing of a paired dataset. By exploiting this technique, we want to solve the Image-to-image translation between MNIST and SVHN. The real challenge of this task lies in the diversity of the two datasets, since one is composed by simple black-and-white images with only one digit per image, while the other contains colored images with multiple digits per image.

2. Related work

In the last years, Generative Adversarial Networks [1] have been proved to be a successful method in image generation [6]. The reason of this success, is due to the adversarial loss, that makes possible to generate images that are indistinguishable from real looking pictures. A popular task in computer vision is image translation, in which GANs have been deployed with remarkable results. In [3] for example, they used conditional adversarial networks as a general-purpose solution to image-to-image translation. The proposed method, is able to solve complex task such as reconstructing objects from edge maps and colorizing images. However, most of these works are limited in the sense that they work in a fully supervised setting. This means that given two domains, for each image of the first dataset, we need the corresponding counterpart in the second domain. Obtaining such a paired data, is not easy in practice, indeed quite often the situation is the opposite: we have two unpaired datasets and we would like to learn a mapping between the two domains. CycleGANs were introduced to solve the same problem but without the need of paired training data.

Figure 1. Example of a successful translation from horse to zebra

3. Methodology

3.1. CycleGANs

As explained before, we are in the scenario in which we have an unpaired dataset, hence we will use a CycleGAN. This model is formed by two generators and two discriminators. The first generator (G) converts images from the X domain to the Y domain. The other generator (F) does the opposite. Each generator has a corresponding discriminator, which tries to distinguish synthesized images from real ones. There are two components in the CycleGAN objective function, an adversarial loss (for both mappings) and a cy-

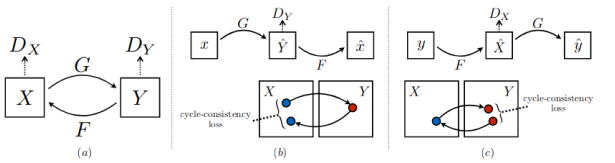

Figure 2. CycleGANs losses [9]. On the left the two adversarial losses are depicted. The two images on the right represent the cycle consistency losses

cle consistency loss. Both are essential to get good results. The former is defined as follows (considering the mapping $G : X \rightarrow Y$):

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

In particular, the discriminator for Y tries to maximize 1, while the generator $G : X \rightarrow Y$ tries to minimize it. However, the adversarial losses are not enough to produce good images, as it leaves the model under-constrained. It enforces the generated output to belong to the right distribution, but does not enforce the output to preserve the characteristic of the input image. For this reason, we add a second loss called cycle consistency loss:

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

The idea is that if we convert an image to the other domain and back again, by successively feeding it through both generators, we should get back something similar to the original input. In [9] they performed ablation studies to demonstrate the importance of this component. 2 shows an illustration of the adversarial losses and the cycle consistency loss. In conclusion, the full loss can be expressed as

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

We propose thereby three different models to solve the proposed task. The first two networks can be considered as our baselines, since they consist of a UNET[7] generator and a regular CNN discriminator that outputs one single number. The differences between these two implementation is the loss function. The first method uses NS GAN, while the second one LS GAN. The third and more advanced method, tries to improve the baseline by building a more complex architecture for the generator, that incorporates residual blocks as suggested in [9].

3.2. UNET generator with LS GAN

For our baseline, the generator consists of a UNET. More precisely, we start with an image of size 32x32. The image is then shrinked by means of convolutional layers with stride 2 until we get an activation map that is 8x8. The number of channel increases while we reduce the size of the activation maps. The input is then upsampled twice with transposed convolutional layers in order to get an image of the same size of the input. The activation used in each layers are *LeakyReLU* with $\alpha = 0.2$, with the exception of the last layer that is a *sigmoid* since we normalized the data in the range [0,1]. The discriminator is instead a simple CNN that reduces the image size until we get a single number. An important thing to notice is that the activation of the last layer of the discriminator is based on the loss function that we are using. In particular, as explained in [5], when we use LS GAN, the output of the discriminator is unbounded, hence the very last activation is linear. LS GAN loss is defined as follows (assuming gradient ascent for mapping $G : X \rightarrow Y$):

$$\mathcal{L}_D = \mathbb{E}_{y \sim p_{data}(y)}[(D(y) - 1)^2] + \mathbb{E}_{x \sim p_{data}(x)}[D(G(x))^2] \tag{4}$$

$$\mathcal{L}_G = \mathbb{E}_{x \sim p_{data}(x)}[(D(G(x)) - 1)^2] \tag{5}$$

3.3. UNET generator with NS GAN

NS GAN is an updated version of loss function originally introduced in [1]. In particular, instead of minimizing the probability of the discriminator being correct, we maximize the probability of the discriminator being wrong:

$$\begin{aligned}\mathcal{L}_D = \mathbb{E}_{y \sim p_{data}(y)}[logD(y)] \\ + \mathbb{E}_{x \sim p_{data}(x)}[log(1 - D(G(x)))]\end{aligned} \tag{6}$$

$$\mathcal{L}_G = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] \tag{7}$$

The discriminator maximize 6, while the goal of the generator is to maximize 7. The architecture of both the generator and discriminator remain the same. The only difference is in the activation of the last layer for the discriminator. In fact, when using NS GAN, this number is considered as the probability of the generated image to belong to the target distribution. Hence the output should be in the range [0,1], so a *sigmoid* activation must be used.

3.4. Extended generator with NS GAN

The generator in our third architecture is based on the one used in [9]. It consists of three sections: an encoder, a transformer, and a decoder. The input image is fed directly into the encoder, which shrinks the representation size to 8x8 while increasing the number of channels. The encoder

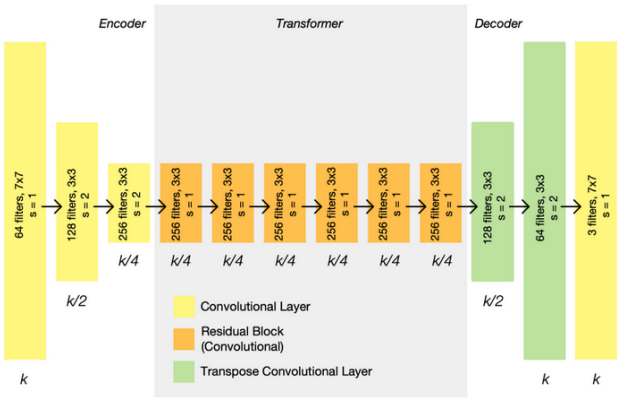

Figure 3. Generator, [8]

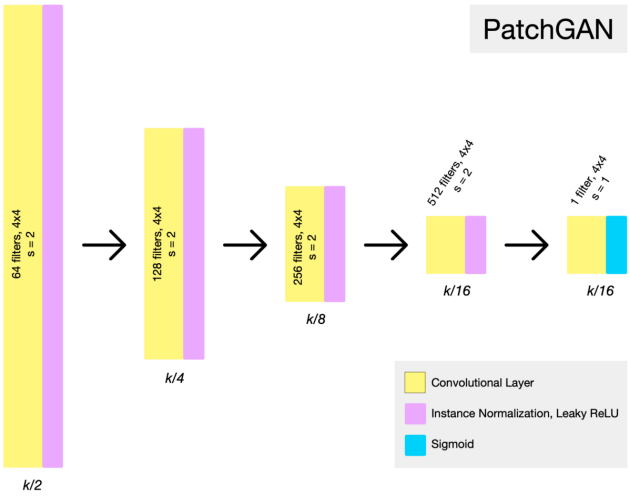

Figure 4. Discriminator [8]

is composed of three convolution layers. The resulting activation is then passed to the transformer, a series of six residual blocks. It is then expanded again by the decoder, which uses two transpose convolutions to enlarge the representation size back to 32x32, and one output layer to produce the final image. Each layer is followed by an instance normalization layer. We can consider this architecture as an extension of the previous one since the econder and the decoder are essentially the same, but with the addition of the residual blocks in the middle. 3 illustrates the full architecture of the generator. In our implementation, we only used 3 residuals blocks rather 6, for the sake of computation time.

The discriminators described by the authors of [9] are PatchGANs, fully convolutional neural networks that look at a patch of the input image, and output the probability of the patch being real. This is both more computationally efficient than trying to look at the entire input image, and is also more effective. However, since in our case the images to be discriminated are very small (i.e. 32x32) (unlike the images used in [9]), the two discriminators are just regular CNN that predict one single number, instead of a number for each patch.

3.5. Data preprocessing and training details

The learned mapping is between MNIST and SVHN. One key difference between these two datasets is the shape of the images. In the former case, the image size is 28x28x1, while for the latter is 32x32x3. Hence, a first important preprocessing step is to reshape all the images coming from MNIST to 32x32x1, by padding all the images with the constant 0 (black pixels). A common trick applied when using GANs in general, is to normalize images between 0 and 1. Another possibility would be to normalize between -1 and 1, but for this project, only the first option have been tested. It would be interesting to try the second alternative in the future. In order to monitor the learning process, a validation set is required. Both SVHN and MNIST come with only a training and a test set. For this reason, 10000 digits haven been taken from both the training set to build a validation set. To evaluate the models during learning we used the following strategy: every 50 epochs we used the MNIST validation set to synthesize the corresponding 10000 SVHN digits. We then compute the FID score using the generated digits and statistics for the SVHN validation set (the 10000 digits taken from the training set). In order to do this efficiently, we precomputed the statistics for the SVHN validation set. It is important to note that we did not use the FID score on fake MNIST digits to monitor the learning process, since after a few epochs it was already good, and recording this value wasn't really helpful. This is probably due to the fact that MNIST is much simpler than SVHN. Figure 5 shows the behaviour of the three models during training. LS GAN is clearly the worst. The FID score for NS GAN decreases gradually until the optimal value of 23 is reached, against the 25 achieved by the extended generator. Moreover, because of the differences between the two datasets, it was really difficult to find the right parameters for the two optimizer (one for the discriminators and one for the generators). In particular, with the extended model that uses residual blocks, the network was able to learn the mapping in the complex direction (i.e. from MNIST to SVHN), but in the other direction, we experienced model collapse. We hypothesize that the cause of this behavior was a too powerful architecture. This is the main reason why we reduced the number of residual blocks in our implementation in addition to the computational time. As optimizer, we used *RMSprop* for both the discriminators and the generators with learning rate $2e^{-4}$ and e^{-4} respectively. Another important parameter to tweak is λ in equation 3, namely the importance to give to the reconstruction loss. For our task, we found that the best value is 5. When we tried 10, as in [9], the digits were reconstructed perfectly, although the generated images were clearly worse.

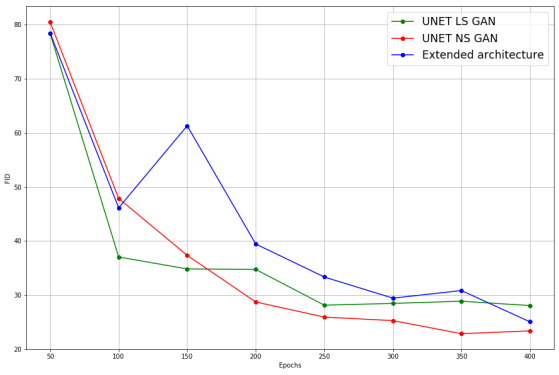

Figure 5. FID score computed every 50 epochs using generated SVHN digits and the corresponding validation set.

Figure 6. Random generated MNIST digits (left) and random generated SVHN digits (right).

	MNIST Fid	*SVHN Fid*
UNET with LS GAN	10.91	37.47
UNET with NS GAN	**9.28**	32.63
Extended model with NS GAN	14.33	**30.59**

Table 1. FID scores for the three models.

4. Results

To compare the quality of the different versions objectively we used the Frchet Inception Distance [2]. To compute the FID score, we use the Inception network to extract features from an intermediate layer. Then we model the data distribution for these features using a multivariate Gaussian distribution with mean μ and covariance Σ. Lower FID values mean better image quality and diversity.

4.1. Qualitative results

It is really hard to evaluate the quality of the generated images for humans. This is the reason we used the FID score. Indeed, by simply looking at the synthesized images we are not probably able to judge which perform better. For this reason, we only report some images produced with our best model. In figure 6 we report 100 images generated for both MNIST and SVHN. In the first case the digits are sharp and they really seem to belong to the corresponding distribution. Also in the second case the quality is acceptable, moreover we can appreciate the variability of the colors: some images are white, others are blue or orange. This was not obvious since all the images are generated from black and white digits.

4.2. Quantitative results

In order to provide a more robust evaluation of the different networks, we used the corresponding test sets to generate fake images and computed the FID score. Again we precomputed the statistics for both MNIST and SVHN for the test sets to be able to get this score. Table 1 summarizes the results. We can see that the extended architecture achieves the best score for SVHN. Interestingly enough, it is also the worst model for MNIST, albeit this difference in the score is not really noticeable simply by looking at the images because of the simplicity of this dataset. On the other hand, this is not true for the second domain, because even with a smaller improvement in the FID score for SVHN, we see that the model is actually able to exploit more colors and to create sharper images.

5. Conclusions

The goal of this project was to learn a bidirectional mapping between MNIST and SVHN. We tried firstly a simple architecture and then we performed some experiments to understand the best loss function to use for this specific task. Afterwards, we extended the architecture of the generator inspired by [9]. All the proposed methods were able to learn successfully a bidirectional mapping, although the last network performs better. It would be interesting in the future to run some other experiments. For example, we could perform some additional ablation studies trying other loss functions or by simply normalizing the input image between [-1, 1] instead of [0, 1] as we did. Another import aspect of this project is that all the proposed methods learn a one-to-many mapping. For example, given a 6 in the MNIST space, we can end up in many digits in the SVHN space. One could try to force the network to generate the same digit by adding a perceptual loss as done in [4].

References

[1] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[2] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017.

[3] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[4] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[5] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? A large-scale study. In *NeurIPS*, pages 698–707, 2018.

[6] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015. cite arxiv:1511.06434Comment: Under review as a conference paper at ICLR 2016.

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[8] Sarah Wolf. Cyclegan: Learning to translate images (without paired training data), 2018.

[9] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.