CycleGAN implementation on MNIST and SVHM

Xing Guan xguan@kth.se
Aleksandar Balicevac avisnjic@kth.se
Velisarios Miloulis miloulis@kth.se
Oguzhan Ugur Oguzhanu@kth.se

Project in Deep Learning DD2424

Abstract. This paper reviews the literature about the CycleGAN [1] and GANs[2] in general, throughly examines the algorithm introduced by Jun-Yan Zhu,Taesung Park, Phillip Isola, Alexei A.Efros by reimplementing the algorithm, changing its initial architecture and evaluating it on different datasets such as MNIST and SVHM. The architecture of the network has been twisted both in a manner of the number of layers and number of different parameters and also different functions and techniques were tested in order to see who the learning is affected. The results show that the network is hard to control in a precise manner due to its instability but some of the parameters presented in the report do have a significant impact on the network performance.

Keywords: GAN, CycleGAN, image-to-image translation

1 Introduction

The lately explosive development of deep learning techniques has opened a new era in computer vision. New techniques for computer vision applications such as face recognition and photo stylization have been introduced and applied into our everyday lives[3]. A significant brakethrough within the field occurred with development of Generative Adversial Network in 2014 by Ian J.Goodfellow[2, 3]. The idea has been appointed by the Yan LeCunn, one of the fathers of deep learning, as the most important idea in the field of Machine Learning in the last 20 years[5].

The Generative Adversarial Network, or as abbreviated GAN, is primarily applied to natural images and consists of two networks called generator and discriminator[4]. The generative model captures the data distribution and the discriminative model estimates the probability that the input came from a training sample rather than the generator[2]. The generator gets trained by trying to fool the discriminator, in other words it gets trained by maximizing the probability of the discriminator making a mistake. This is similar to a zero-sum/minimax game where the generator is getting better at fooling the discriminator and the discriminator is getting better at distinguishing generated data from real data.

GANs have been used in several areas when it comes to visualization, one of these areas is image-to-image translation[1]. The purpose of the Image-to-image translation is to translate an image, say a horse to another image of a zebra. The unpaired image-to image translation model is named CycleGAN[1], mostly because it consists of two GANs that cycles in between them. The most compelling feature of a CycleGAN compared to a Vanilla GAN is that the CycleGAN has two generators and two discriminators that take two inputs into account. The Vanilla GAN on the other hand is as the original GAN described on the previous paragraph and consists of, one discriminator, one generator and one input. The output of the vanilla GAN corresponds to the output of the trained generator which in this case will be a generated image of the real input[2, 4]. The CycleGAN will have two outputs which corresponds to the outputs from the two trained generators. These two outputs will have some characteristic contours from the two real inputs depending on which input the generator is trained on.

1.1 Problem Formulation

The aim of this project is to investigate/create a CycleGAN network similar to the network described in the paper *"Unpaired image-to-image Translation using Cycle-Consistent Adversarial Networks"*[1] and produce some results on image-to-image translation between colored and uncolored MNIST dataset, uncolored and SVHM dataset Furthermore, the difficulty of the training procedure and initialization as well as how different network parameters affect the performance is also investigated.

2 Background

2.1 Generative Adversarial Networks

With the development of Deep Convolutional Networks computers ability to see and understand images has experienced a significant improvement . The General Adversarial Networks (GANs) have taken this development one step further creating a significant breakthrough when it comes to generation and manipulation of images.

The basic principle of GAN is based on unsupervised learning and in its simplest form the network consists of two models: *Discrimative model* D and *Generative model* G. The model D learns to map the input data x into a desired class label $\mathbf{y}$ which in probabilistic terms means that it learns the conditional distribution $P(y|x)$. The model G that competes against the *Discriminative model* is trained to learn the joint probability of the input data x and labels y - $P(x, y)$ as illustrated below.See Fig.1

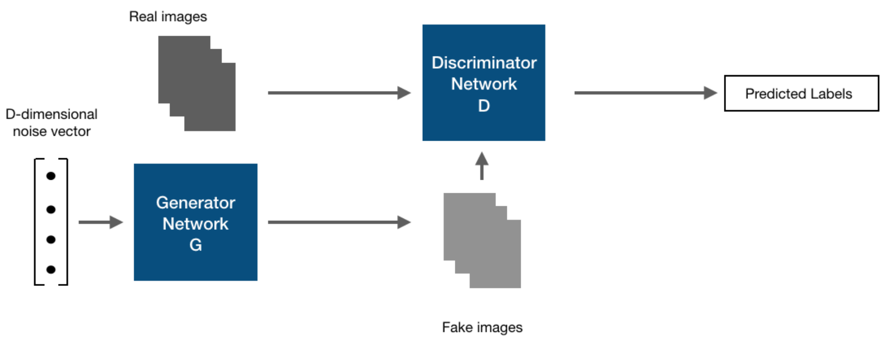


Fig. 1.

Loss Minimization

The minimax game is the process of loss minimization for both networks where the D is trained to maximize the probability of assigning the correct label both to the training data x and the samples z generated by G, while G is trained to generate samples z as similar as possible to those from x. The minimax game is represented with value function $V(G, D)$ as following [2]:

$$\min(G) max(D) V(D, G) = \mathbb{E}_{xpdata(x)}[logD(x)] + (\mathbb{E})_{zpz(z)}[log(1D(G(z)))] \tag{1}$$

2.2 Cycle-Consistent Adversarial Networks

Attracting many machine learning experts the Vanilla GAN was twisted and further developed resulting in one of the latest architecture versions that is investigated in the course of this project - the Cycle-Consistent Adversarial Network (CycleGAN). The model is built upon the PIX2PIX architecture but it translates characteristics of one domain to another in the absence of any paired training samples [1]. The problem can broadly be described as image-to-image translation of for example sunny beach photos from domain X into overcast beach photos from domain Y without any previous X-Y- pairing.

The Model

The model consists of two generators and two discriminators. Mathematically describing the model, the two generators can be denoted as two mapping functions: G: $X \rightarrow Y$ and F: $Y \rightarrow X$ while the two discriminators can be denoted as D_y and D_x. Based on the principle of the Vanilla GAN described above, D_y encourages G to translate the domain X into into outputs similar to the domain Y and vice versa for the discriminator D_x and generator F followed by the *adversarial losses* to express the outcome of the process. To improve the translation and achieve *cycle consistency*, the cyclical steps are added through *forward mapping* of the generated images by the generator G back to their original domain X: $x \to G(x) \to F(G(x))x$, and *backward mapping* of the generated images by the generator F to their original domain Y: $y \to F(y) \to G(F(y))y$. The network is updated by two critical loss functions - *adversarial* and *cycle consistency loss*.See Fig.2

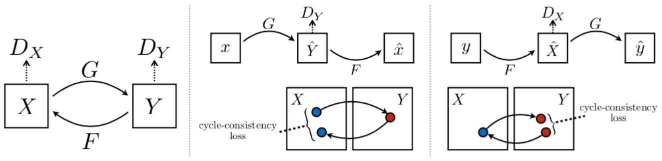


Fig. 2.

Adversarial Loss

Adversarial loss [2] is the basic loss calculated for both mapping functions on the same principle as if network was a simple Vanilla GAN. For the mapping function G: $X \to Y$ and the corresponding discriminator D_y, the loss function is mathematically represented below by the equation (2) and its aim is to solve the same problem of the minimax game as described in the section 2.1.1 above[1]. The same loss is used for regulation of the mapping function F: $Y \to X$ with its corresponding values.

$$L_{GAN}(G, D_Y, X, Y) = E_{y \sim p_{data}(y)}[log D_Y(y)] + E_{x \sim p_{data}(x)}[log(1 - D_Y(G(x)))] \tag{1}$$

Cycle Consistency Loss

The draw-back of the *adversarial loss* is that if the network is large enough, the domain X could be mapped into any sample in the domain Y[2]. That is where the magic part of the CycleGAN comes in - the *cycle consistency loss*. The aim of cycle consistency loss is to reduce the space of possible mapping functions and increase the possibility of unique
$x_i \to y_i$ mappings by pushing the mapping functions to be *cycle consistent*[1]. Cycle consistency means that network should be able to bring back each sample x to its original form after is has been translated to the domain Y (the *forward cycle consistency*), i.e. X: $x \to G(x) \to F(G(x))x$. The same concerns the samples y translated to domain X that should be able to be brought back to their original form (the *backward cycle consistency*). Mathematically, the cycle loss is expressed as following:

$$L_{cyc}(G,F) = \mathbb{E}_{x\sim p_{data}(x)}[||F(G(x))-x||_1] + \mathbb{E}_{y\sim p_{data}(y)}[||G(F(y))-y||_1] \tag{2}$$

The Full Network Objective
The full network objective is to minimize the loss of the whole network combining the two loss types explained above which is mathematically expressed as:

$$L(G,F,D_X,D_Y) = L_{GAN}(G,D_Y,X,Y) + L_{GAN}(F,D_X,Y,X) + \lambda L_{cyc}(G,F) \tag{3}$$

λ - represents the relative importance of the cycle loss.

3 Approach

3.1 Dataset

Datasets used for image-to-image translation were MNIST, Colored MNIST and SVHM. The MNIST datasets were used due to their simplicity. The implemented translations were MNIST to colored MNIST and MNIST to SVHM.

MNIST $\leftrightarrow$ Colored MNIST
This translation was anticipated to be less difficult than the translation MNIST to SVHM translation since the samples from both domains have the similar digit features. The expectations of this translation was that the standard MNIST dataset images should be colorful and the colored MNIST should become black and white.

MNIST $\leftrightarrow$ SVHM
The SVHM dataset was expected to be more complex both in terms of preprocessing of the data and training of the network. The version of the SVHM dataset used in our experiments is the cropped one [6] because of the image size that is closer to the one of the MNIST samples. The dataset consists of street numbers that are not clearly visualized and therefore more difficult to anticipate. The expectations of this transformation was that the MNIST dataset should to a certain degree look like a street number and the SVHM like a MNIST digit. The different datasets are depicted on Fig. 3,4,5

3.2 Preprocessing of Data

MNIST $\leftrightarrow$ Colored MNIST
The MNIST dataset was generated from Tensorflow which uses Yann LeeCun's MNIST database. This dataset consists of 60000 training images with image size 28 x 28. The number of images used during training was only 100 for image-to-image translation. The images were normalized to [-1,1]. The colored MNIST

dataset was generated by taking a random crop of Lenna image [8] which was put in the background. To make the digit visual, every pixel of the digit color was inverted [7]. The obtained result is depicted on Fig. 4. Those images were also normalized to [-1,1].

MNIST $\leftrightarrow$ SVHM
The SVHM dataset was downloaded from the official Stanford webpage[6]. Images in this dataset were in different sizes and were therefore resized to 32x32. Some of the images became blurred during this process, the resolution of the samples were degraded. The MNIST Dataset on the other hand was resized to 32x32 by using zero padding. The images of these datasets were normalized to [-1,1].

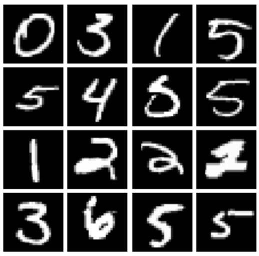


Fig. 3. Standard MNIST

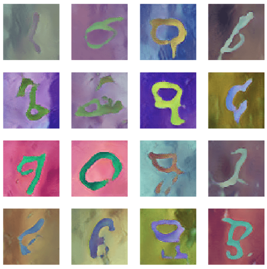


Fig. 4. Colored MNIST[7]

Fig. 5. SVHM [6]

3.3 Network architecture

High Level Structure

The network consists of two Generators - $Generator_{A->B}$ and $Generator_{B->A}$, and two discriminators - $Discriminator_A$ and $Discriminator_B$. Generators role is to take image samples from their corresponding domains ($input_A$ and $input_B$) and learn how to translate these images into some of the samples in the opposite domain providing generated images $Generated_B$ and $Generated_A$. Generated images are evaluated by the discriminators "defending" their original domains who's intention is to reject the generated images and evaluate as correct the original images. Generator images are also sent back through the opposite generator in order to try to recreate the original image that it originates from. The $cycle - samples$ are then compared to their original correspondents. See Fig.6

Generator

The generators consist of 3 main parts who's sizes in manner of number of layers and number of extracted features were changed during our experiments *(see Fig. 7)*:

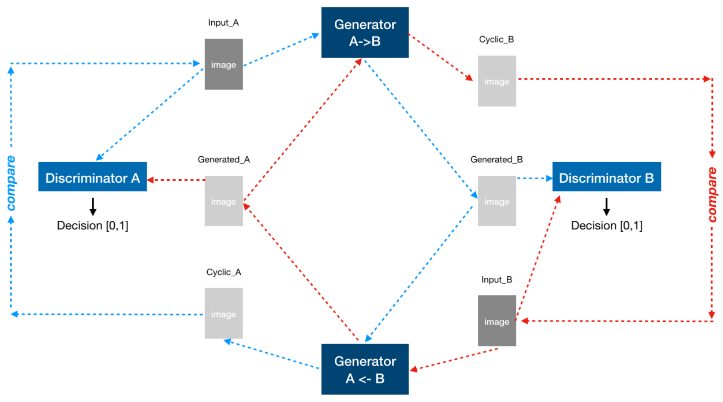


Fig. 6.

1. **Encoder** that is used to extract image features on different levels. It consists of several convolutional layers with progressively increasing number of features for each layer, and the same kernel and stride size for all the layers except for the first one.
2. **Transformer** that withholds the characteristics which on it decides how the input vector that comes from decoder should be translated to the opposite domain. Transformer consists of several *ResNet Blocks*
with same number of features.
3. **Decoder** Decoder takes the the translated high level vector from the *Transformer* and builds back low level features from it in a reversed manner compared to *Encoder*
. It consists of several *deconvolutional layers* (transposed convolution) with progressively decreasing number of features and one convolutional layer at the end to convert the low level features to a vector that should represent an image in the opposite domain.

Discriminator

The discriminator takes a vector representation of an image as input and produces a scalar output that predicts if an image comes from the original domain or if it is a generated one. It consists of a several convolutional layers who's number was changed during our experiments, with progressively increasing number of features and constant window and stride size (layers 1 - $n-1$) and one last layer (layer n) with one feature *(see Fig. 7)*.

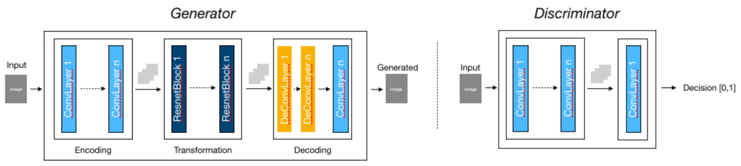
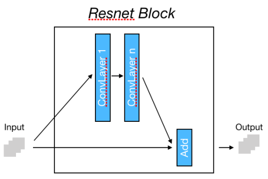


Fig. 7.

ResNet Block

ResNet block consists of two convolutional layers with same number of features, kernel and stride sizes. In ResNet blocks, the residue of input is added to the output in order to ensure that some of the properties of original image are retained.



Convolutional (conv2D) Deconvolutional (deconv2d) Layers These two layer types as explained above are the main building blocks for the whole network. **conv2D** computes a 2-D convolution with a possibility to perform normalization for which we used a couple different functions in our experiments and activation for which Leaky ReLU [9] was used. **deconv2d** performs transposed convolution with the same possibilities to perform normalisation and activation as the **conv2d**.

4 Experiments

The experiments are carried out by implementing the original CycleGAN architecture that are suited for the images of size 256x256 and 128x128. The initial implementation and work was inspired by [10] as the starting point and the implementation was modified by changing the hyper- parameters and numbers of layers of both generator and discriminator. Since the MNIST and SVHN datasets are comparably small that is why we are trying with smaller network but kept the overall architecture of the network. The idea is to remove the unnecessary layers but still retain good style transfer of the images. In the following subsections, we will explain what we adjusted in the network to suit to the MNIST and SVHN dataset. Other results that are worthmentioning are presented under section 5, $Hyper-parametersandotherfindings$.

4.1 Generator VS Discriminator

The tricky part with training GAN is that it is hard to find the perfect balance between learning the generator and discriminator. If the discriminator is not improving then the generated image will passed easily and not getting improved even though the generated images are corrupted. In a practical manner it means that we need a good discriminator as a judge to identify whether the generated images are good or bad in order to increase the loss for the generator. The generator should be complex or large enough to learn the feature of the images.

By changing the numbers of layer of generators or discriminators we could either increase or decrease the complexity of the network. In the original paper the ResNet size varied depending on the size of input images. Since the MNIST and SVHN datasets are much smaller, we choose to experiment by having 2, 6, and 9 ResNet layer and see if 2 ResNet layers are equally good as 6 and 9 ResNet layers.

4.2 Patch Discriminator VS Vanilla Discriminator

Our special Patch discriminator differs from the conventional Patch discriminators by only taking one cropped image. The size of the patch was chosen to be 8x8 for the MNIST dataset and 9x9 for the SVHN. The results are presented under the subsection Effects of Patch Discriminators.

5 Results and Conclusion

The results are divided into the MNIST $\leftrightarrow$ colorful MNIST part and MNIST $\leftrightarrow$ SVHM part. In the following subsections, the results are attached and some techniques to improve the network will be discussed.

5.1 Result of MNIST $\leftrightarrow$ colorful MNIST

The results obtained of this translation are depicted on Fig.8, as it is possible to see the results are pretty good. There are still some constraints while translating the images since the two datasets are different in terms of complexity. The colorful MNIST has more different colors and due to its complexity it is a bit harder to generate. The network used while generating these images were 2 conv layers, 2 ResNETs and 2 deConvs on the generator and the discriminator had 3 conv layers and a reflection padding.

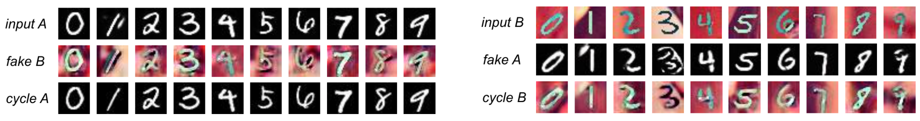


Fig. 8.

5.2 Corrupted images

Some of the images were corrupted during the training. It is hard to tell why some the images were not successfully generated. The network can be sensitive to the color of the inputs and therefore leads to undesireble result. See figure 9

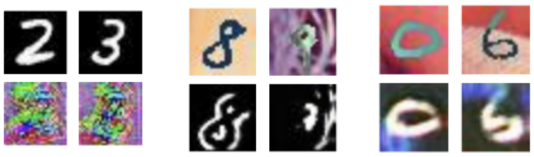


Fig. 9.

5.3 Effects of ResNet

By looking at the quality of the generated images of different numbers of the ResNet we could see that the loss of discriminators did not change much when using the 6 ResNet instead of 9 ResNet layers. It implies that the network with 2 ResNet can captured the features of the input images as good as 6 and 9. The following plots are the loss of discriminators of 2, 6, and 9 ResNet layers from left to right. The losses did not change much even with decreasing ResNet layers. The network with 2 ResNet layers seem to be enough to generate good quality images. See Fig.10

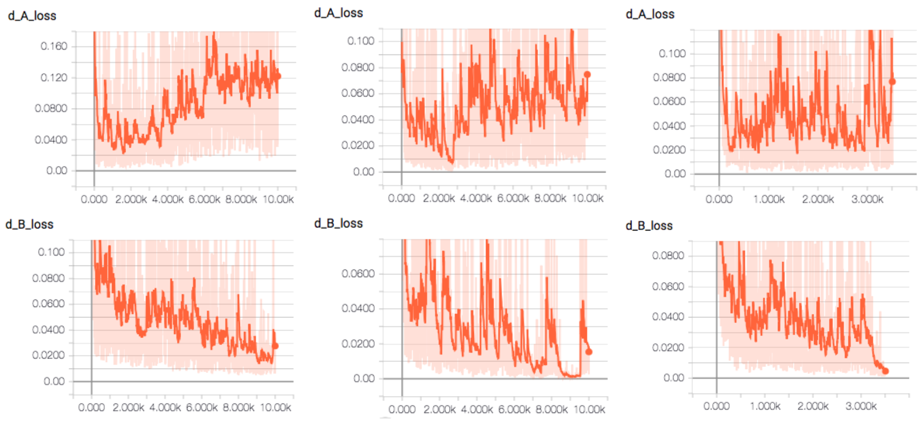


Fig. 10.

5.4 Effects of Patch discriminators

By comparing the Patch Discriminators and Vanilla Discriminators, the Patch Discriminator A has more stable loss ranging between 0.215 and 0.265 compare to the Vanilla discriminator loss ranging from 0.05 to 0.25. The Patch discriminator B is not as stable as A. The input data B, colorful MNIST are more complex than the black white MNIST, it could be a reasonable explanation of the behavior of the discriminator B. See Fig.11

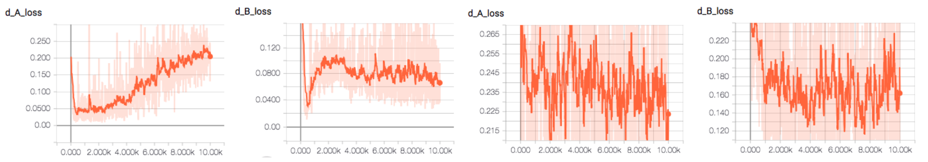


Fig. 11. The left plots discriminator A and B are from the Vanilla Discriminator and right plots are from the Patch discriminators

5.5 Hyper-parameters and other findings

Some hyper-parameters such as stride, kernel size and output size were adjusted to stabilize the network. The strides were set to 2 for the generators and discriminators except the last layer. Changing the stride to 1 did not affect the loss of network. We also attempted to change the kernel size of the generators by setting the first layer kernel size to 5 instead of 7 and rest kernel size to 2 instead of 3. The losses did not change much either here. Since the GAN by nature is hard to stabilize, even though the results improve a bit, we can not draw the conclusion of the effects of those changes. Some obvious improvements were done by increasing the size of the dataset from 100 images to 500 images for the MNIST to colorful MNIST and vice versa. To start with, we set the number of generator filters to 8 and discriminator to 16. By increasing the numbers to 16 and 32 and 32 and 64, the results improved. Since there are more features captured by filters, it could be the explanation to the improvement. By observing the losses of discriminators and generators, we saw that the discriminator losses drops quite fast compare to the generator losses. We chose to set the learning rate to 0.0001 and 0.0002 as the original setting for the generators. The losses of the discriminators became more "spiky" because of that and it was hard to tell whether the generated images were "better" or not.

5.6 Result of MNIST $\leftrightarrow$ SVHN

The transformation MNIST $\leftrightarrow$ SVHN is more complex than the transformations we attempted before. Due to the complexity it was expected that the behavior of the network would become more clear during our experiments with this task. Initial experiments were conducted with a reduced capacity network and decisions were made based on both the quality of the generated images and the behavior of the networks losses.

Out of all the normalization methods, the best results were achieved with normalizing in the [-1, 1]. The G losses were decreasing steadily but the D losses were unstable and the quality of the generated images was not satisfactory.

Since the G losses remained quite high, and it was unable to generate images of good quality, we tried to slow down the rate of learning of D. We attempted two different methods, division of the D objective or using another optimizer. The authors of the original paper decreased the loss objective dividing it by 2 which in our experiment was increased to 4, with no apparent effect on the losses or the quality of the images.

The attempt to slow the learning rate of G by using a worse optimizer for it, namely Vanilla Stochastic Gradient Descent, had similarly poor effects.

We turned back to using Adam for all the optimizers, and we used format 2 of the SVHN dataset, which consists of single SVHN digits cropped to 32x32. This allowed us to also increase the size of the MNIST digits by zero padding.

This produced really unstable Discriminators as it can be seen in Fig. 12.

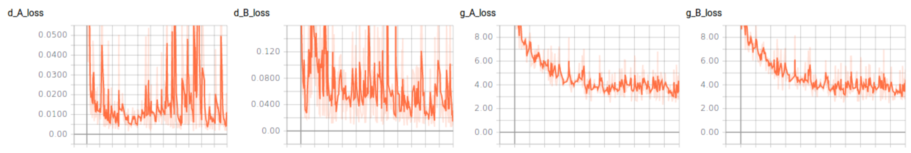


Fig. 12. Losses after while using ADAM and format 2 of the SVHN

In an attempt to generate better looking images, while being able to preserve the overall structure of the digit, the number of features extracted from each convolution in G was drastically increased. The reflection padding was removed before the input and output layers of the discriminator, to account for the changed expected dimensionality of the inputs.

The fake MNIST images produced by that configuration look quite convincing, even if the original digit is transformed into a different one. Some examples of that behavior can be seen in Fig: 13 The fake SVHN images are very bad, and seemingly noisy. This led us to tinker further with the architecture, and reintroduce reflection padding, in the hopes that it would reduce the artifacts, as claimed in the original paper.

The produced losses by that configuration can be seen in Fig. 14. The discriminator is very unstable and with very high variance. Taking a cue from the original paper, we changed the input to D, from the full image to a random 9x9 crop of it. This seemingly small change had drastic effect on our results.

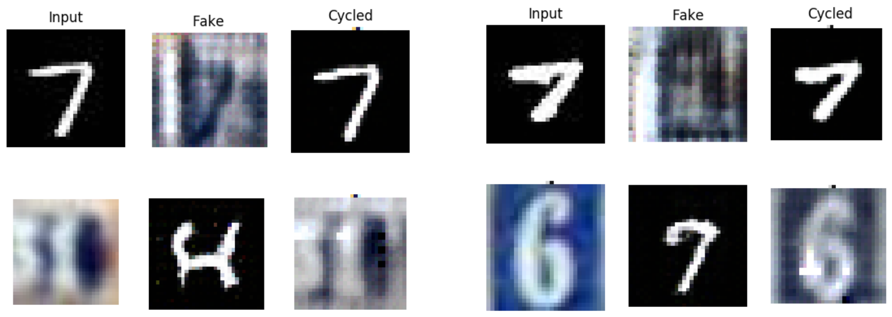


Fig. 13. Image transformations after drastic increase of the extracted features in G.

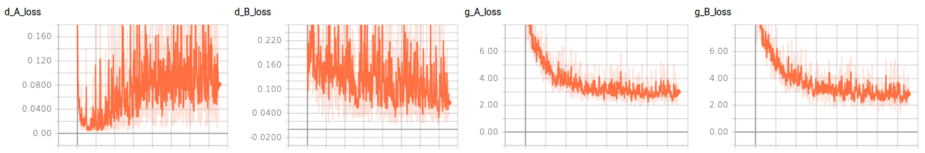


Fig. 14. Losses after reintroducing reflection padding and increasing the number of extracted features

As can be seen from Fig. 15, the discriminator loss stabilized to a large extend, with its variance reducing and its actual value high enough for the generators to keep receiving error signals and continue falling.

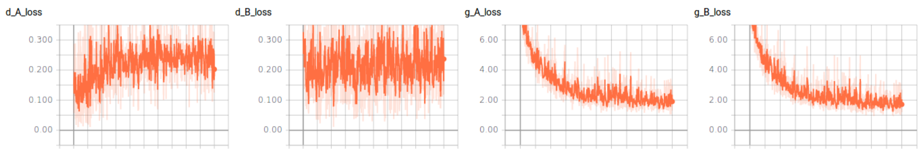


Fig. 15. Losses after changing the input to the discriminator to a random 9x9 crop.

The results of the better behaved losses can be seen in the quality of the images produced too, Fig: 16.

6 Discussion

After extensive experimentation with the architecture, we can draw some conclusions about its behavior. First of all, training the network is a game of balancing between the losses of D and those of G. The urge to minimize the losses of the network can lead to bad training iterations, since the overall behavior of the

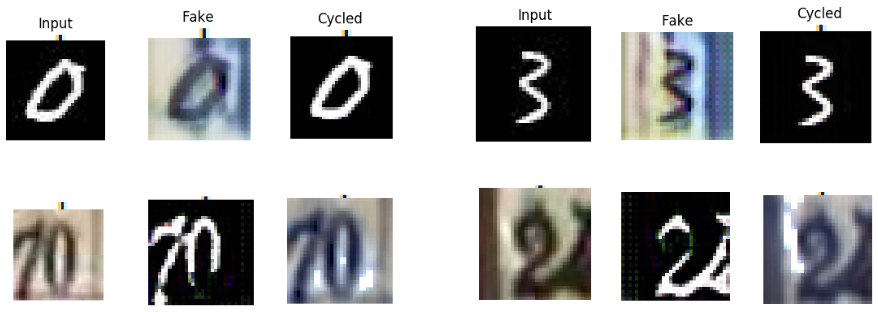


Fig. 16. Image transformations after changing the input to the discriminator to a random crop.

loss curve and its relation to the loss curve of the other network is much more important than the actual loss value. The smoothness and stability of the losses are much better indicators of a good training iteration.

In general, it seems that the network is quite robust in incremental changes to its architecture. Changing parameters such as the kernel size, the stride, or to some extent the number of extracted features does not seem to affect the final output much. Bigger changes, such as the addition or removal of layers in either of the networks, or the image preprocessing can have some immediately noticeable effects.

In our experiments, we defined the Generators and Discriminators for both domains as the same networks. This was done in accordance with the original paper, where both domains were of similar complexity. Since in our case one domain was simpler than the other, MNIST to SVHN, there were some hints during training that the simpler domain could do with a simpler architecture. Again, that can be seen from the general curves of the losses - if a discriminator loss is unstable for one domain, only its D may need changing.

References

1. Jun-Yan Zhu, Taesung Park, Philip Isola, Alexei A.Efros *Unpaired image-to-image Translation using Cycle-Consistent Adversarial Networks*
2. Ian J.Goodfellow *Generative Adversarial Nets*
3. National Research University Higher School of Economics, *Deep Learning in Computer vision*
   https://www.coursera.org/learn/deep-learning-in-computer vision
4. John Glover, *An introduction to Generative Adversarial Networks*
   http://blog.aylien.com/introduction-generative adversarial networks-code-tensorflow/
5. *Yan Lecunn, This, and the variations that are now being proposed is the most interesting idea in the last 10 years in ML, in my opinion.*
   https://www.kdnuggets.com/2016/08/yann-lecun-quora session.html
6. *SVHM*
   http://ufldl.stanford.edu/housenumbers
7. Wouter Bulten, *Colorful MNIST*
   https://www.wouterbulten.nl/blog/tech/getting-started-withgans-2-colorful-mnist/
8. *Lenna* https://en.wikipedia.org/wiki/Lenna
9. *Rectifier*
   http://cs231n.github.io/neural-networks-1/
10. *Hardik Bansal CycleGAN*
    https://hardikbansal.github.io/CycleGANBlog/