Bachelor thesis
Computer Science

Radboud University

On the replication of CycleGAN

Author:
Robin Elbers
r.j.elbers@student.ru.nl
s4225678

First supervisor/assessor:
MSc. Jacopo Acquarelli
j.acquarelli@cs.ru.nl

Second assessor:
Prof. Tom Heskes
t.heskes@science.ru.nl

August 10, 2018

Abstract

The cycle-consistent generative adversarial network (CycleGAN) is a type of artificial neural network that is capable of performing image-to-image translations. It is capable of doing this translation when paired data is not available for training. In this thesis we will explain the cycle-consistent generative adversarial network in detail as well as replicate the results. We make several modifications to the original version and evaluate how it impacts the performance by means of the fully convolutional network score. We start by getting a baseline score of the network in the original version. Then we will replace the residual generator with a U-Net generator. Afterwards, we will train the network using the last generated image, instead of using a history of generated images. Finally, we will change the least squares adversarial loss for the Wasserstein loss with Lipschitz penalty.

Contents

Chapter 1

Introduction

The way we communicate, pay and travel has evolved over the last several years partly due to advancements in machine learning. In fact, credit card companies use fraud detection algorithms to label transactions as fraudulent or not, Facebook uses facial recognition to detect faces in photos and Tesla uses computer vision to recognize traffic signs.

These tasks can be solved using by discriminative models that can separate different samples into different classes. However, this approach requires datasets to be annotated which in some cases can be expensive. Another approach which does not necessarily require annotations uses a generator in combination with a discriminator. The generator is trained to generate data according to a data distribution. In the field of artificial neural networks (ANN), there are several different models which are capable of such tasks, e.g. restricted Boltzmann machines, variational autoencoders or generative adversarial networks (GAN). This thesis will focus on the cycle-consistent generative adversarial network (CycleGAN) which is a type of GAN.

CycleGAN is designed to perform unpaired image-to-image translations. This means that the input and the output of the artificial neural network is an image. In the world of image-to-image translation, there are two variants of training data: the data can be paired or unpaired. In the former case, every input sample is paired with exactly one output sample. In the latter case, there is no such mapping. Colourizing black and white images is an example of a paired image-to-image translation. However, paired data is not always available. Style transfer or image stylification is a problem for which there are, usually, no paired data available. For example, training a model to translate pictures to look like Van Gogh paintings requires paired data which are typically not available.

An important constraint to such translations is that the semantics of the input must be preserved. For example, a photo of a bridge, when stylized in the style of Van Gogh, should look like a painting of the same bridge. CycleGAN is capable of preserving semantics on unpaired training data. In

this thesis, we will replicate the results of CycleGAN in its original version and with some modifications like changing the generator and discriminator architecture or changing the loss function.

Chapter 2

Related Work

2.1 Generative models

The subject of this thesis is CycleGAN, which is a kind of GAN. However, there are other generative models besides GANs. In this section we will briefly explain how restricted Boltzmann machines and variational autoencoders work.

2.1.1 Restricted Boltzmann machine

The Restricted Boltzmann machine[21] is an adaption to the Boltzmann machine such that it can be more easily trained. Instead of forming a complete graph, the hidden and visible nodes in a RBM form a bipartite graph. RBMs are trained by using Gibbs sampling instead of backpropagation, which consists of updating the hidden units by sampling the visible units and updating the visible units by sampling the hidden units. New samples can be generated by randomly initializing the hidden units, performing Gibbs sampling and then sampling the visible units.

2.1.2 Variational autoencoders

Autoencoders[5] are a type of ANN consisting of an encoder, which embeds the input into a lower dimensional latent space, and a decoder, which maps the embedding in the latent space back to the input. The autoencoder has a reconstruction loss which ensures that the reconstructed output is close to the input. Variational autoencoders[11] (VAE) further require that the embedding of samples in the latent space follows a normal distribution. New samples can be generated by initializing the latent space from a random normal distribution and using the decoder to reconstruct an image.

2.2 GANs for image-to-image translations

Since their appearance, many GANs besides CycleGAN have been proposed for image-to-image translation. We will briefly discuss the ones most relevant to CycleGAN.

2.2.1 pix2pix

With regards to CycleGAN, pix2pix[8] is the most relevant GAN. In fact, the authors of CycleGAN state that CycleGAN builds on the pix2pix framework. Pix2pix is designed for paired image-to-image translation. Let (x, y) be a paired sample from the dataset. The generator of pix2pix attempts to map x to y, but it does not observe y directly. The discriminator receives both x and the generated y' and predicts if y' is real or fake given x. Pix2pix uses U-Net as generator and PatchGAN as discriminator. Both PatchGAN and U-Net will be explained later in this thesis. The pix2pix authors also describe a metric for quantitatively evaluating the performance of pix2pix called the FCN score. This metric is also used by the CycleGAN authors for evaluating their network, and we will use it as well. The FCN score will be explained later in this thesis.

2.2.2 Domain transfer network

The domain transfer network[22] (DTN) is a type of GAN which uses a single autoencoder as the generator, which (during training) takes input from both the source and target domain and maps it to the target domain. The reconstruction loss is only applied if the input image is from the target domain. If, on the other hand, the input is from the source domain there is a loss that minimizes the distance between the encoded input and the encoded output.

2.2.3 DiscoGAN

DiscoGAN[10] is similar to CycleGAN: both of them use a reconstruction loss to enforce cycle-consistency at a pixel level. However, for the adversarial loss DiscoGAN uses cross-entropy instead of the mean squared error that CycleGAN uses. Furthermore, DiscoGAN does not use a PatchGAN discriminator or residual generator.

2.2.4 UNIT

The UNIT framework[13] uses a combination of variational autoencoders and generative adversarial networks, called VAE-GAN, to perform image-to-image translation. UNIT uses a pair of VAE-GANs, one for each domain. The weights of the last few layers of the encoders and the first few layers of

the decoders are shared between the domains. This shared lower dimensional embedding can then be decoded to one of the two domains by their respective decoders.

2.2.5 Cross-GAN

Cross-GAN[18] (XGAN) uses a similar method to UNIT with a network that consists of 2 autoencoders with a shared embedding. However, they use regular autoencoders instead their variational counterpart. And in contrast to CycleGAN, which enforce cycle-consistency at a pixel level, they propose to enforce cycle-consistency at a feature level. Such that the embedding of the input is close to the embedding of the output when encoded again. They argue that this allows for more flexible transformations.

Chapter 3

Preliminaries

This chapter will contain the background information needed to understand the rest of the thesis.

3.1 Artificial Neural Networks

An ANN is a directed graph of nodes or neurons. The nodes are grouped into layers. The output of a node is the weighted sum of its inputs applied to an activation function ϕ. The output of a node is also called the activation of a neuron. The bias node has a constant output of 1 and is connected to every output node in a layer. A typical deep neural network has one input layer, one output layer and several hidden layers. The input layer is fed with samples from a dataset while the output layer provides predictions to corresponding input samples.

An ANN can also be seen as a function from a tensor of inputs to a tensor of outputs. For example, the ANN in figure 3.1 is a function $N : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ defined by the equation

$$N(x, y) = \phi(0.6 \cdot \phi(0.4 \cdot \phi(1x) + 0.2 \cdot \phi(0.5y)) + 0.5 \cdot \phi(0.3 \cdot \phi(1x) + 0.1 \cdot \phi(0.5y))),$$

with

$$\phi(x) = x.$$

Since the activation function ϕ is linear, N can be simplified. The simplified equation shows that we can express N using a network with only 2 weights, meaning we don't need the hidden layers. In fact, every ANN with only a linear activation function has an equivalent ANN with only a single layer.

$$\begin{aligned} N(x, y) &= \phi(0.6 \cdot \phi(0.4 \cdot \phi(1x) + 0.2 \cdot \phi(0.5y)) + 0.5 \cdot \phi(0.3 \cdot \phi(1x) + 0.1 \cdot \phi(0.5y))) \\ &= 0.6 \cdot (0.4 \cdot x + 0.2 \cdot 0.5y) + 0.5 \cdot (0.3 \cdot x + 0.1 \cdot 0.5y) \\ &= 0.39x + 0.085y \end{aligned}$$

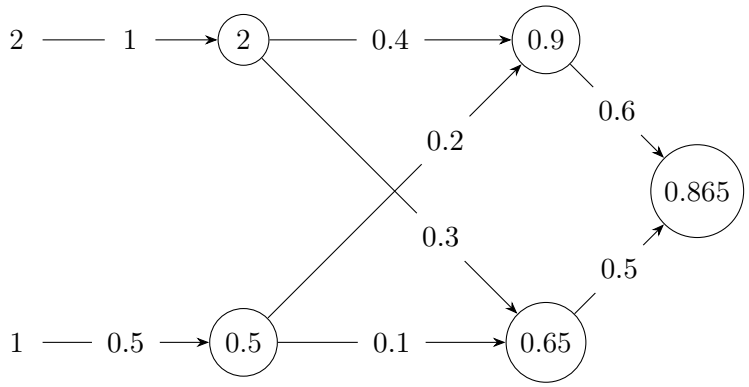

Figure 3.1: Example of a neural network without bias nodes, with $f(x) = x$.

An ANN with at least one hidden layer and a non-linear activation function can approximate any continuous function[6]. To do this the weights of the network needs to be adjusted to minimize the error between the target values and the actual output of the network. The feedforward algorithm is used to calculate the output, and the backpropagation algorithm is used to learn the weights.

The following activation functions are used in the ANNs described in this thesis:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

$$\text{ReLU}(x) = \begin{cases} 0, & \text{if } n < 0 \\ x, & \text{otherwise} \end{cases},$$

$$\text{LeakyReLU}(x) = \begin{cases} \alpha x, & \text{if } n < 0 \\ x, & \text{otherwise} \end{cases}.$$

Feed forward Input samples are given to the ANN and modified by the hidden layers. The last layer of the network produces the predictions. The output for every n-th layer is calculated as follows:

$$X_n = \begin{cases} X, & \text{if } n = 0 \\ A_{n-1}, & \text{otherwise} \end{cases}$$
$$A_n = f(Z_n)$$
$$Z_n = W_n X_n,$$

where X are the input samples and W_n are the weights of the n-th layer.

Backpropagation Gradient descent is used to find the optimal weights for the ANN. Gradient descent needs a function to optimize in order to reduce the prediction error. This function is called the loss function and it quantitatively describes the prediction error. Gradient descent is used to change the weights in the direction of the minimum of the loss function, which is essentially a distance measure between two tensors. The sum of squares error and cross entropy are examples of loss functions used by ANNs and their definitions are shown below, with Y as the output of the network, T as the target and $\odot$ as the Hadamard product.

$$E : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

Sum of squares:

$$\mathrm{E}(Y, T) = \sum_{i=0}^{|T|} (Y_i - T_i)^2.$$

Cross entropy:

$$\mathrm{E}(Y, T) = -(T \odot \ln(Y) + (1 - T) \odot \ln(1 - Y)).$$

To apply gradient descent we need the gradient of the error with respect to the weights of the ANN. The backpropagation algorithm uses the chain rule to calculate $\frac{dE}{dW_n}$ for the weights of every layer W_n in the ANN. In this way, we can change the weights of each layer in the negative direction of the error.

Normalization One way to speed up the training process is to apply normalization to the inputs of the layers. While training the ANN, the distribution of the input of each layer changes as the weights in the network changes. This shift is called internal covariate shift and normalization reduces this to speed up the training. However, a recent paper claims it is not effective because it reduces internal covariate shift, but because it makes the error function more smooth[19].

Batch normalization[7] works by normalizing the inputs of a layer such that the mean and variance of the entire minibatch is 0 and 1 respectively.

The definition is shown below. Where $x \in \mathbb{R}^{T \times C \times W \times H}$ is a minibatch of T images with channels C, width W and height H.

$$
\begin{aligned}
y_{tijk} &= \frac{x_{tijk} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}, \\
\mu_i &= \frac{1}{HWT} \sum_{t=1}^{T} \sum_{l=1}^{W} \sum_{m=1}^{H} x_{tilm}, \\
\sigma_i^2 &= \frac{1}{HWT} \sum_{t=1}^{T} \sum_{l=1}^{W} \sum_{m=1}^{H} (x_{tilm - m\mu_i})^2.
\end{aligned}
$$

Instance normalization[23] works similar to batch normalization, but instead it normalizes each sample in the minibatch to a mean and variance of 0 and 1.

$$
\begin{aligned}
y_{tijk} &= \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}, \\
\mu_{ti} &= \frac{1}{HW} \sum_{l=1}^{W} \sum_{m=1}^{H} x_{tilm}, \\
\sigma_{ti}^2 &= \frac{1}{HW} \sum_{l=1}^{W} \sum_{m=1}^{H} (x_{tilm - m\mu_{ti}})^2.
\end{aligned}
$$

3.2 Convolutional Neural Networks

Let's consider the task of detecting if an image contains a car or not. It does not matter if the car is in the top left or bottom right of the image, the ANN should still detect the car. This property is called translational invariance. Convolutional layers have a related property called equivalence, which means that a translation in the input corresponds to an equivalent translation in the output. Convolutional layers can be combined with pooling layers to provide translational invariance. An ANN using such convolutional layers is called a convolutional neural network (CNN).

A convolutional layer uses a smaller filter or kernel which is convolved with the image. In each convolutional step, the dot product of the kernel with the overlapping part of the image is computed. The resulting values are captured in the feature map of the filter. The weights of the kernel are

parameters which are learned while training the CNN. Consider the input matrix:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix},$$

with the filter:

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}.$$

The output of the convolution (the feature map) is:

$$\begin{pmatrix} \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} & \begin{pmatrix} 2 & 3 \\ 5 & 6 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \\ \begin{pmatrix} 4 & 5 \\ 7 & 8 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} & \begin{pmatrix} 5 & 6 \\ 8 & 9 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 18 & 30 \\ 36 & 42 \end{pmatrix}.$$

Padding Convolution makes the output to be smaller than the input. In fact, every convolutional step corresponds to one value in the output. We can pad the input matrix such that the output will have the same size as the input. Typically we either pad with zeroes or pad in such a way that the padding 'reflects' the values at the border as shown below.

No padding:

$$\begin{bmatrix} 7 & 3 & 6 & 5 & 5 \end{bmatrix}.$$

Zero padding:

$$\begin{bmatrix} 0 & 0 & 7 & 3 & 6 & 5 & 5 & 0 & 0 \end{bmatrix}.$$

Reflect padding:

$$\begin{bmatrix} 6 & 3 & 7 & 3 & 6 & 5 & 5 & 5 & 6 \end{bmatrix}.$$

Number of filters More than one filter can be used in order to increase the dimensionality of the output. We applied a single filter in our example, so the output of the convolutional layer was of size 2×2. If the task of the CNN is to generate an RGB image, we can apply 3 filters to get an output of size $2 \times 2 \times 3$. Where the 3 generated feature maps correspond to the three colours.

Stride Instead of sliding the filter one pixel at a time, we can change the number of pixels the filter is moved each convolutional step. The number of pixels the filter is moved is called the stride. A stride higher then one downscales images proportional to the stride. To upscale images transposed convolutions (also known as fractionally strided convolution) can be used instead of regular convolutions. A padded convolution over an image of size 32×32 with a stride of 2 will have a feature map of size 16×16. With a stride of 4 this will be 8×8.

Receptive field Input regions corresponding to the values in the output are called the receptive fields. For example, the receptive field of 30 in the feature map from the example above corresponds to the following submatrix of the input:

$$\begin{pmatrix} 2 & 3 \\ 5 & 6 \end{pmatrix}.$$

In this case, the dimensions of the receptive field are equal to the dimensions of the filter, but if multiple convolutional layers are stacked after each other this receptive field can grow larger. The receptive field can be calculated using the formula below, where r_i is the receptive field, s_i is the stride and k_i is the kernel size of the $(n-i)$-th layer. The receptive field is calculated starting from the output layer and working backwards to the input:

$$r_i = (r_{i-1} - 1) \cdot s_i + k_i.$$

3.3 Generative models

Traditionally ANNs have been used for feature extraction and discriminatory tasks. Generative models instead, aim to approximate a data distribution as closely as possible. In order to define a cost function for such generative models, it is useful to look at datasets as probability distributions. As an example, we will describe how to train a model that generates handwritten digits like in the MNIST dataset[12]. The dataset contains 28x28 pixels and a probability can be assigned to every combination of pixel values. The combination of pixel values that represent an image in the dataset will have a high probability and those that do not represent an image in the dataset have a low probability. Equivalently, the images that a generative model can generate can be seen as a probability distribution as well. Thus, if we have a function that can measures the difference (or similarity) between probability distributions, we can use it as a loss function and use it to train an ANN.

Generative Adversarial Networks A Generative Adversarial Network[3] (GAN) is a type of ANN introduced by Goodfellow et al. in 2014 and is such a generative model. It consists of two smaller ANNs with two distinct roles. One part of the GAN is responsible for generating data from a noise vector, this part is also known as the generator. The other part of the GAN, called the discriminator, is responsible for predicting whether an input image is real or fake. The discriminator tries to minimize the cross-entropy loss of those labels, while the generator tries to maximize that same loss. In other words, the generator tries to fool the discriminator by mapping the input z to an image that the discriminator will predict to be real. The optimal parameters for the generator is defined as:

$$\arg\min_G \max_D \big[\mathbb{E}_{x\sim p_r}\big[\log(D(x))\big] + \mathbb{E}_{z\sim p_z}\big[\log(1-D(G(z)))\big]\big],$$

with p_r the distribution of real data and p_z the distribution of noise. From this, we can derive a loss function for the discriminator and the generator. The loss for the generator is expressed as $-L_D$, however when taking the derivative with respect to the generator's parameters the term with $D(x)$ is a constant, so it can be removed. Goodfellow et al. states this loss function represents the Jensen–Shannon divergence between the model's distribution and the target distribution. We will refer to this loss as the adversarial loss.

$$\begin{aligned} L(D) =& -\mathbb{E}_{x\sim p_r}\big[log(D(x))\big] - \mathbb{E}_{x\sim p_g}\big[log(1-D(x))\big], \\ L(G) =& \mathbb{E}_{z\sim p_z}\big[log(1-D(G(z)))\big]. \end{aligned}$$

Mode collapse Generative models should be capable of capturing all modes of the target data distribution. Mode collapse is when the GAN fails to do this. In the context of image generation, we want the GAN to generate a diverse set of images. However, if there is an image that minimizes the loss for the generator, then the generator may learn to map every input to only that point. A straightforward way of detecting mode collapse is to look at the generated images: if the generator only creates similar looking images there is mode collapse.

Chapter 4

Methods

In this section we will describe the methods. First we will give a high-level description of CycleGAN. Then we will describe the loss function, discriminator and generator of the CycleGAN in more detail. Finally we will show an alternative generator architecture in the U-Net and an alternative loss function in the Wasserstein loss with Lipschitz penalty.

4.1 CycleGAN

The CycleGAN[24] is a GAN designed for unpaired image-to-image translation, where the task is to translate images from a source domain $\mathcal{A}$ to a target domain $\mathcal{B}$. It consists of two GANs, one for translating from domain $\mathcal{A}$ to $\mathcal{B}$ and one from $\mathcal{B}$ to $\mathcal{A}$. The two discriminators represent the functions:

$$D_{\mathcal{A}} : \mathcal{A} \to \mathbb{R}$$
$$D_{\mathcal{B}} : \mathcal{B} \to \mathbb{R}.$$

The two generators represent the functions:

$$G_{\mathcal{A}} : \mathcal{A} \to \mathcal{B}$$
$$G_{\mathcal{B}} : \mathcal{B} \to \mathcal{A}.$$

$G_{\mathcal{A}}$ maps images from $\mathcal{A}$ to $\mathcal{B}$. $D_{\mathcal{B}}$ receives real images sampled from $\mathcal{B}$ and fake images generated by $G_{\mathcal{A}}$ and predicts whether an image is real or fake.

The images generated by $G_{\mathcal{A}}$ are mapped back to the original domain by $G_{\mathcal{B}}$. The image generated by $G_{\mathcal{B}}$ should accurately represent the original image that was the input of $G_{\mathcal{A}}$. The cycle-consistency loss minimizes the distance between the actual input and this reconstructed input.

$G_{\mathcal{A}}$ maps images to $\mathcal{B}$. If the input itself is sampled from $\mathcal{B}$, the generator should map the input to itself. The identity loss minimizes the distance

between the input and the image that is generated by mapping it to the same domain.

The same explanation holds when the domains $\mathcal{A}$ and $\mathcal{B}$ are swapped. Figure 4.1 is a graphical overview of what is explained above.

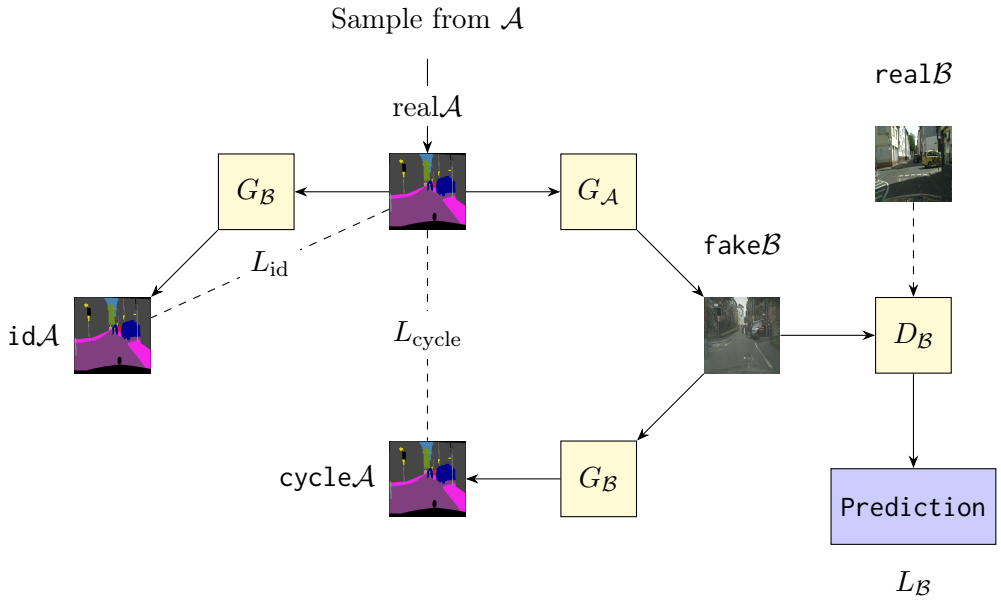

Figure 4.1: Overview of CycleGAN. real$\mathcal{A}$ is an image sampled from $\mathcal{A}$, real$\mathcal{B}$ is an image sampled from $\mathcal{B}$, fake$\mathcal{B}$ $= G_{\mathcal{A}}$(real$\mathcal{A}$), Prediction $= D_{\mathcal{B}}$(fake$\mathcal{B}$), cycle$\mathcal{A}$ $= G_{\mathcal{B}}$(fake$\mathcal{B}$), id$\mathcal{A}$ $= G_{\mathcal{B}}$(real$\mathcal{A}$), L_{id} represents the identity loss, L_{cycle} represents the cycle-consistency loss and $L_{\mathcal{B}}$ represents the adversarial loss.

4.1.1 Objective function

The objective function of CycleGAN has many differences compared to that of a regular GAN. First of all, instead of using cross-entropy for the adversarial loss, CycleGAN minimizes the mean squared error. A GAN using this loss is described by Mao et. al.[15]. They claim that minimizing such a loss corresponds to minimizing the Pearson χ^2 divergence. Furthermore, they claim the training is more stable and that it generates higher quality images. They also remove the sigmoid activation that is usually present in the final layer of the discriminator, so the codomain for the discriminators is $\mathbb{R}$ instead of $(0, 1)$. The gradient of the sigmoid function approaches zero when the input increases in magnitude. This means that if the discriminator is confident in its output, the generator will receive less feedback in the form of a gradient. By removing the sigmoid activation that problem is no longer present.

The objective for the CycleGAN has an additional term for the so-called the cycle-consistency loss. This tries to enforce the requirement that $G_{\mathcal{A}}$ and $G_{\mathcal{B}}$ are each others inverses. The cycle-consistency loss prevents mode collapse by ensuring that not too much information is lost in the translation between domains. Hence, the following should hold:

$$
\begin{aligned}
G_{\mathcal{A}} \circ G_{\mathcal{B}} &= \mathrm{id}, \\
G_{\mathcal{B}} \circ G_{\mathcal{A}} &= \mathrm{id}.
\end{aligned}
$$

CycleGAN also uses loss term called the identity loss. Let's assume the task is to translate paintings to photos. The ANN may learn to map paintings of daytime to photos of sunsets. The cycle-consistency and adversarial loss does not prevent this mapping, while the identity loss does. An image sampled from domain $\mathcal{A}$, when mapped to the same domain, should give the same image. This loss is optional and the CycleGAN authors suggest to use it when the network seems to produce images that have a different tint or colour than expected. It enforces the following property:

$$
\begin{aligned}
\mathbb{E}_{x\sim\mathcal{B}}\big[G_{\mathcal{A}}(x)\big] &= \mathbb{E}_{x\sim\mathcal{B}}\big[\mathrm{id}(x)\big] \\
\mathbb{E}_{x\sim\mathcal{A}}\big[G_{\mathcal{B}}(x)\big] &= \mathbb{E}_{x\sim\mathcal{A}}\big[\mathrm{id}(x)\big]
\end{aligned}
$$

This gives us the loss shown below with all terms combined, where the cycle-consistency and identity losses are weighted by λ_{c} and λ_{i} respectively.

$$
\begin{aligned}
L_{\mathcal{A}}(D_{\mathcal{A}}) =& \mathbb{E}_{x\sim\mathcal{A}}\big[(D_{\mathcal{A}}(x)-1)^2\big] + \mathbb{E}_{x\sim\mathcal{B}}\big[(D_{\mathcal{A}}(G_{\mathcal{B}}(x)))^2\big] \\
L_{\mathcal{B}}(D_{\mathcal{B}}) =& \mathbb{E}_{x\sim\mathcal{B}}\big[(D_{\mathcal{B}}(x)-1)^2\big] + \mathbb{E}_{x\sim\mathcal{A}}\big[(D_{\mathcal{B}}(G_{\mathcal{A}}(x)))^2\big] \\
L_{\text{cycle}}(G_{\mathcal{A}}, G_{\mathcal{B}}) =& \mathbb{E}_{x\sim\mathcal{A}}\big[\big\|G_{\mathcal{B}}(G_{\mathcal{A}}(x)) - x\big\|_1\big] + \mathbb{E}_{x\sim\mathcal{B}}\big[\big\|G_{\mathcal{A}}(G_{\mathcal{B}}(x)) - x\big\|_1\big] \\
L_{\text{id}}(G_{\mathcal{A}}, G_{\mathcal{B}}) =& \mathbb{E}_{x\sim\mathcal{A}}\big[\big\|G_{\mathcal{B}}(x) - x\big\|_1\big] + \mathbb{E}_{x\sim\mathcal{B}}\big[\big\|G_{\mathcal{A}}(x) - x\big\|_1\big] \\
L(D_{\mathcal{A}}, D_{\mathcal{B}}, G_{\mathcal{A}}, G_{\mathcal{B}}) =& L_{\mathcal{A}}(D_{\mathcal{A}}) + L_{\mathcal{B}}(D_{\mathcal{B}}) + \lambda_{\text{c}} L_{\text{cycle}}(G_{\mathcal{A}}, G_{\mathcal{B}}) + \lambda_{\text{i}} L_{\text{id}}(G_{\mathcal{A}}, G_{\mathcal{B}})
\end{aligned}
$$

4.1.2 Discriminator

For the discriminator, CycleGAN uses the PatchGAN classifier as described in the pix2pix paper[8]. Instead of classifying the entire image, PatchGAN instead determines if $N \times N$ patches of the image are real or not. This is done by stacking convolutional layers after each other, resulting in a $U \times V$ matrix, such that every value of the output has a receptive field of $N \times N$. The pix2pix authors determined that 70×70 patches produce the best result. Because it classifies patches instead of the entire image, it is both faster and it uses less memory. Figure 4.2 shows the discriminator in more detail.

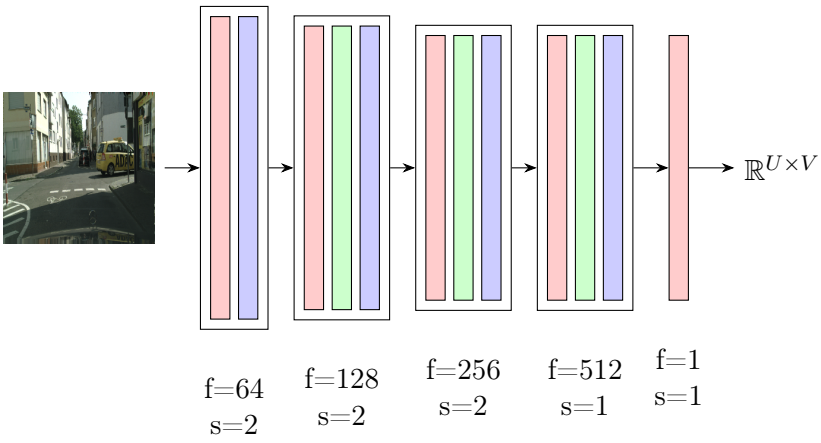

Figure 4.2: PatchGAN discriminator. A red layer is a convolutional layer with a kernel of size 4×4, f filters and a stride of s. A blue layer is a LeakyReLU activation layer with a slope of 0.2. A green layer is an instance normalization node. The receptive field for every entry in the output is a 70×70 patch of the input.

The discriminator is trained using a history of generated images, instead of just the latest generated image. This technique is first described by Shrivastava et al. and they claim that this improves the stability of training[20]. Let $D_{\texttt{Buffer}}$ be a buffer of size m and n be the batch size. After each training iteration, we add the n generated images to the buffer. If the buffer is full we randomly replace n images in $D_{\texttt{Buffer}}$ with the newly generated ones. The

discriminator gets the training samples by sampling from $D_{\texttt{Buffer}}$ instead of directly sampling the generator.

4.1.3 Generator

The architecture for the generators is adapted from Johnson et al.[9] and uses residual blocks. In these blocks the input is added to the output, so the block represents a function $g(x) = f(x) + x$. The intuition behind this is that a block will not perform worse than an identity mapping since the input is always available. Furthermore, it helps alleviate the gradient vanishing problem for deep networks. Figure 4.3 shows a single residual block and figure 4.4 shows the complete generator architecture.

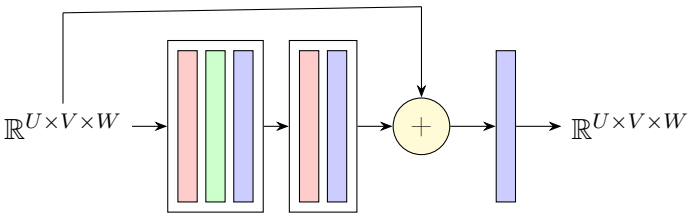

Figure 4.3: A single residual block. The red layers are convolutional layers. The green layer is an instance normalization layer. The blue layers are ReLU activation layers. The yellow plus adds the inputs up.

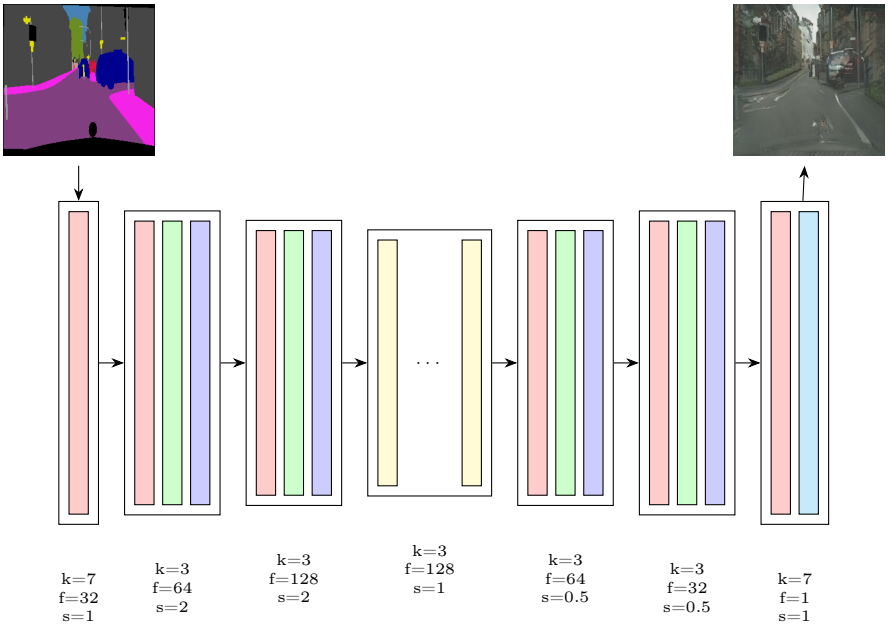

Figure 4.4: Generator with residual blocks. A red layer is a convolutional layer. A green layer is an instance normalization node. A blue layer is a ReLU activation layer. A yellow layer is a residual block. A cyan layer is a tahn activation layer. Convolutional layers have a kernel of size $k \times k$, f filters and a stride of s. For images of size 256×256 or higher, the number of residual blocks is 9. For smaller images 6 residual blocks are used.

4.2 U-Net

The pix2pix network uses a different type of generator. The U-Net was designed for medical imaging[17] and can be described as a convolutional autoencoder with skip connections. The encoder downscales the image to a $1 \times 1 \times 512$ latent space using convolutional layers, starting with 64 filters and doubling the number of filters every layer up to a maximum of 512. The decoder equivalently upscales the latent space back to the original dimensions. Every transposed convolutional layer in the decoder has a so-called skip connection to a layer of the encoder. Meaning that the output of the i-th encoder layer is concatenated with the output of the $(n - i)$-th decoder layer. Figure 4.5 shows the generator in more detail.

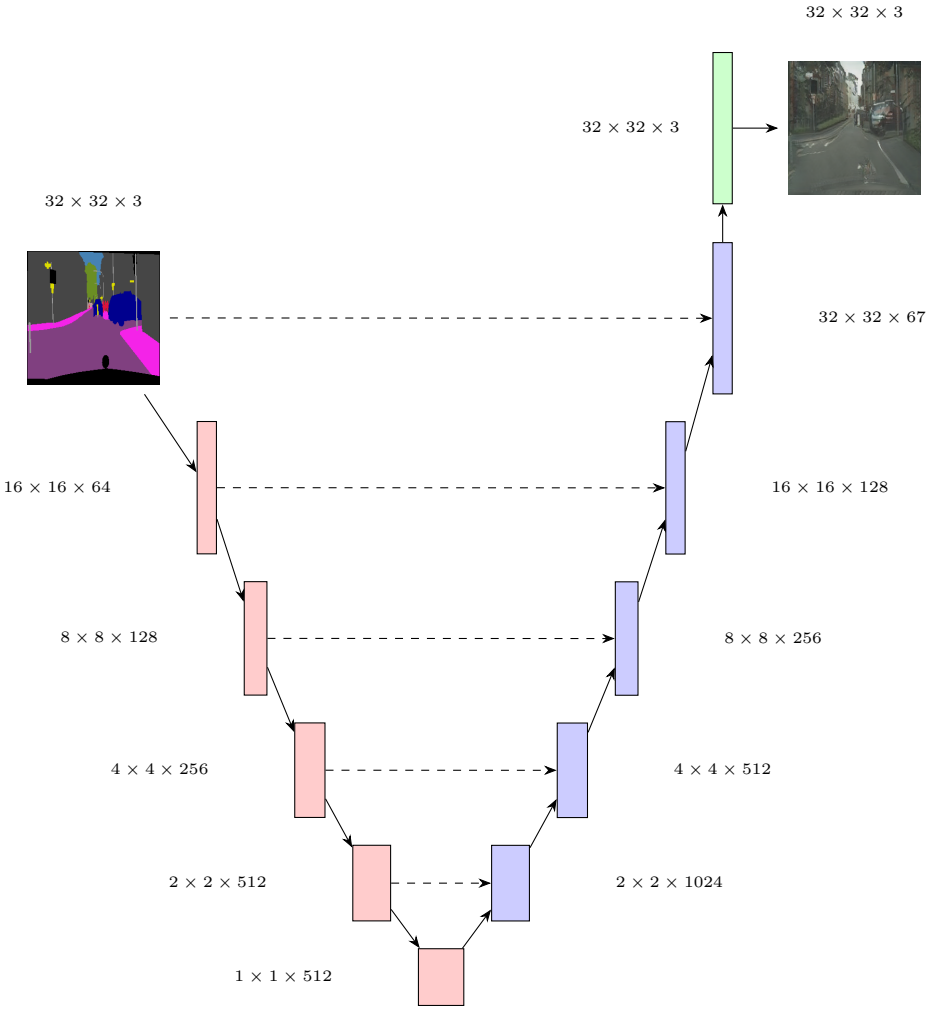

Figure 4.5: U-Net generator for 32×32 images. The red layers are blocks with first a convolutional layer, then an instance normalization layer and finally a LeakyReLU activation with slope 0.2. The blue layers are blocks with a transposed convolutional layer, then an instance normalization layer and finally a ReLU activation. The output of the ReLU is concatenated with the output of the convolutional block that is connected with a dashed line. The stride for these is 2 and the kernel size is 4×4. The green layer is a convolutional layer with 3 filters, a stride of 1 and with a tanh activation. The dimensions of the output of a block of layers is shown next to it. The number of filters for the convolutional layers starts from 64 and is doubled every next layer up to a maximum of 512 filters.

4.3 Wasserstein GAN

The Wasserstein GAN[1] (WGAN) is a GAN with a loss function that minimizes the Wasserstein or Earth Mover distance. The Wasserstein distance is defined as:

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma}\big[\|x - y\|\big].$$

The authors of the WGAN paper state that the infinitum is intractable. They propose to use the dual form of the Wasserstein distance using the Kantorovich-Rubinstein duality:

$$W(P_r, P_g) = \sup_{\text{Lip}(f)\leq 1} \mathbb{E}_{x\sim P_r}\big[f(x)\big] - \mathbb{E}_{x\sim P_g}\big[f(x)\big],$$

with P_r the distribution of real images, P_g the distribution of fake images and the supremum taken over all 1 Lipschitz functions. Lipschitzness is defined as follows. A function f is K Lipschitz iff

$$\forall_{x,y} : |f(x) - f(y)| \leq K \cdot |x - y|.$$

Intuitively it means that the absolute slope of f at any point is no bigger than K. The function f will be our discriminator, which in the context of WGANs is also called the critic. Hence, the loss function for a WGAN becomes:

$$L(D) = \mathbb{E}_{x\sim P_r}\big[D(x)\big] - \mathbb{E}_{x\sim P_g}\big[D(x)\big].$$

As the discriminator is trained, it will map real images to smaller values and fake images to larger values. In fact, it is possible the WGAN will have a negative loss. The authors state that ideally we want to train the critic until optimality, such that the generator get a more reliable gradient. It is not always practical in terms of time to train till optimality, so how many times we update the critic per generator update is another hyperparameter we need to choose.

In order to enforce the 1 Lipschitz property, the authors propose to clamp the weights after each update. They admit that it is a terrible way to enforce this. Gulrajani et al. instead propose to add a term which they call the gradient penalty to the loss to enforce 1 Lipschitzness of the critic[4]. The resulting network is called a WGAN-GP or improved WGAN. This term is defined as:

$$GP(D) = \lambda_{gp}\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\big[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2\big].$$

where $P_{\hat{x}}$ is the uniform distribution along straight lines between pairs of points sampled from P_r and P_g. In simpler terms: we sample P_r and P_g and for every pair of points we uniformly at random choose a point

in between. The gradient penalty term adds a penalty when the gradient of the discriminator for that point deviates from 1. Petzka et al. instead propose to only penalize when the gradient is strictly greater than 1[16]. The regularization term they propose, called the Lipschitz penalty, is defined as:

$$LP(D) = \lambda_{lp}\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\left[(\max(0, \|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1))^2\right].$$

They show that this Lipschitz penalty results in a WGAN which has more stable learning, converges faster and which is less sensitive to the initialization of the λ weight for the regularization term. This Wasserstein GAN with the Lipschitz penalty term is called WGAN-LP.

4.4 Dataset

We will train CycleGAN to perform semantic segmentation on the Cityscapes dataset[2]. The dataset contains photos of urban street views. Every photo is paired with the label map for that photo, which gives a label to each pixel of the photo (e.g. car or vegetation). The dataset is already split in a train, test and validation set. We will use the train set for training and the validation set for evaluating our network. The images in the dataset have a resolution of 2048×1024. Since we do not have the resources to train the CycleGAN at that resolution, we instead train on images resized to 128×128. The dataset is first converted to 256×256 and locally saved as `.jpg` images.

Chapter 5

Experiments

In this chapter we will start by explaining how we trained CycleGAN. Then we will explain the metric we will use to evaluate the performance of the CycleGAN. Finally, we will describe the experiments we performed.

5.1 Training details

The network is trained for 200 epochs, where one epoch is one training pass over the entire dataset. The Adam optimizer is used with the parameters $\beta_1 = 0.5$, $\beta_2 = 0.9$ and $\eta = 0.0002$. The learning rate is constant for the first 100 epochs, for the final 100 epochs the learning rate is linearly decayed to 0. λ_c is 10 and λ_i is 0. When training the discriminator, the loss is divided by 2. The weights are initialized with a Gaussian distribution with a mean 0 and a standard deviation of 0.02. As explained before, the discriminator is trained using a history of generated images with a buffer size of 50. The `batch_size` is 1. For training the train set of cityscapes is used. Every epoch the training set is shuffled and partitioned into subsets the size of the minibatch. The subsets are the minibatches used for training. Every image in the minibatch is then resized to 158×158 and randomly cropped to 128×128 to increase the variety of the training data.

5.2 Evaluation metric

For evaluating the networks we use the same method as described in the CycleGAN paper called by the pix2pix authors as the FCN score. The performance of the CycleGAN is not measured by how well it generates this label map, instead the task is to generate a photo given a label map. Then another network that is pretrained on the Cityscapes dataset will generate a label map for that generated photo. This final generated label map is then compared to the original label map to get the per-pixel accuracy, per-class accuracy and class intersection over union[14]. The definitions of these scores

are:

$$\text{pixel accuracy} = \frac{\sum_i n_{ii}}{\sum_i t_i},$$

$$\text{class accuracy} = \frac{1}{n_{cl}} \sum_i \frac{n_{ii}}{t_i},$$

$$\text{intersection over union} = \frac{1}{n_{cl}} \sum_i \frac{n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}},$$

with n_{cl} as the number of classes in the ground truth label map, n_{ij} the number of pixels of class i that belong to class j, and t_i the total number of pixels with class i in the ground truth label map. These values are calculated for every image and averaged to get the final scores. For clarity, figure 5.1 shows the pseudo-code of the evaluation process. For evaluating the performance the validation set of Cityscapes is used. We will use the evaluation code[1] and pretrained network[2] provided in the pix2pix repository.

5.2.1 Experiment descriptions

Baseline We have trained the implementation of CycleGAN that Zhu et al. provide in their repository[3] using the same hyperparameters as described before. We will compare the FCN scores from their implementation to ours.

U-Net generator The residual generator is replaced with the U-Net generator.

No image buffer The discriminators are trained with the last images that were generated, instead of using a buffer containing a history of generated images.

Wasserstein loss The adversarial loss is replaced with the Wasserstein loss with Lipschitz penalty. The critic is updated 3 times for every generator update. λ_{LP} is 5. The RMSprop optimizer is used for training with $\eta = 0.0001$.

[1] `https://github.com/phillipi/pix2pix/tree/master/scripts/eval_cityscapes`. Accessed at 2018-07-20

[2] `http://people.eecs.berkeley.edu/~tinghuiz/projects/pix2pix/fcn-8s-cityscapes/fcn-8s-cityscapes.caffemodel`. Accessed at 2018-07-10

[3] `https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix`. Accessed at 2018-07-20.

```
# Generate fake photos
fake_photos = []
for label_map in Cityscapes_labels:
    fake_photos += gen_b(label_map)

# Recreate labels given the fake photos
fake_labels = []
for fake_photo in fake_photos:
    resized_photo = resize(fake_photo, (2048, 1024))
    fake_labels += pretrained_network(resized_photo)

# Calculate scores
pa, ca, iou = [], [], []
for (real, fake) in zip(Cityscapes_labels, fake_labels):
    pa += calculate_pixel_accuracy(real, fake)
    ca += calculate_class_accuracy(real, fake)
    iou += calculate_class_intersection_over_union(real, fake)

pixel_accuracy = mean(pa)
class_accuracy = mean(ca)
class_intersection_over_union = mean(iou)
```

Figure 5.1: Evaluation pseudocode. Let `Cityscapes_labels` be the set of label maps from the Cityscapes dataset. Let `gen_b` be $G_{\mathcal{B}}$ trained to generate photos from label maps of the Cityscapes dataset. Let `pretrained_network` be a network pretrained on the Cityscapes dataset.

Chapter 6

Discussion

In this chapter we will discuss the results. The FCN scores are shown in table 7. Figure 7.1 contains generated images of the different experiments at several different epochs. Figure 7.2, 7.3, 7.4 and 7.5 show additional results for the experiments with the Wasserstein loss.

6.1 Baseline

The FCN scores show that our implementation achieves a better score than the official implementation.

6.2 U-Net generator

The FCN scores with the U-Net generator are lower than for the residual generator. This is an interesting result since the U-Net generator, with the number of filters that the pix2pix authors suggest, has significantly more trainable parameters than the residual generator; around 40 million for the U-Net generator vs. around 8 million for the residual generator. This result does support the CycleGAN authors in their decision of changing the generator architecture.

6.3 No image buffer

Training the discriminators using just the last generated images results in a slightly higher FCN score. We do not have sufficient evidence to address the claim by the CycleGAN authors that using the image buffer results in a more stable training.

6.4 Wasserstein loss

The output samples, as well as the FCN scores, show that CycleGAN with Wasserstein loss fails to produce compelling images. As explained before, the discriminator should give smaller values to real images and higher values to fake ones. However, figure 7.2 shows that when we split the loss into separate terms for real and fake samples, the loss of $D_{\mathcal{A}}$ does the opposite of this. The fake images are mapped to negative values and real images to positive values. When we look at the samples in figure 7.4 we can see that although the translation between domains itself was unsuccessful, the reconstructed images look almost perfect. Our hypothesis is that the cycle-consistency constraint is too strong when combined with the Wasserstein loss with Lipschitz penalty. To test this, we have another experiment where we lower the cycle-consistency weight to $\lambda_c = 0.5$ and change the mean absolute error to mean squared error for the cycle-consistency loss, which penalizes the ANN more as the reconstructed input deviates more from the actual input. Due to time constraints, we choose to train with an increased batch size of 32.

Figure 7.3 is a plot of the loss with this alternate cycle-consistency loss. It shows that the loss now behaves as we expected. The fake images are mapped to higher values and the real images to lower values as the network is trained more. The samples in figure 7.1 show that the output is at least visually more convincing than the output with the original Wasserstein loss. However, it still looks less realistic than the baseline implementation. In order to find out if CycleGAN can be combined with the Wasserstein loss and still produce as realistic images as the original implementation, more research is needed to find optimal hyperparameters, including finding out how to best express the cycle-consistency loss.

Chapter 7

Results

	Per-pixel accuracy	Per-class accuracy	Class IOU
Zhu et al.	0.559	0.187	0.132
Baseline (ours)	0.704	0.234	0.171
U-Net generator	0.688	0.222	0.169
No image buffer	0.720	0.237	0.179
Wasserstein loss	0.492	0.112	0.070

Table 7.1: FCN scores. Zhu et al. represents the FCN-scores from the network provided by the CycleGAN authors trained on Cityscapes.

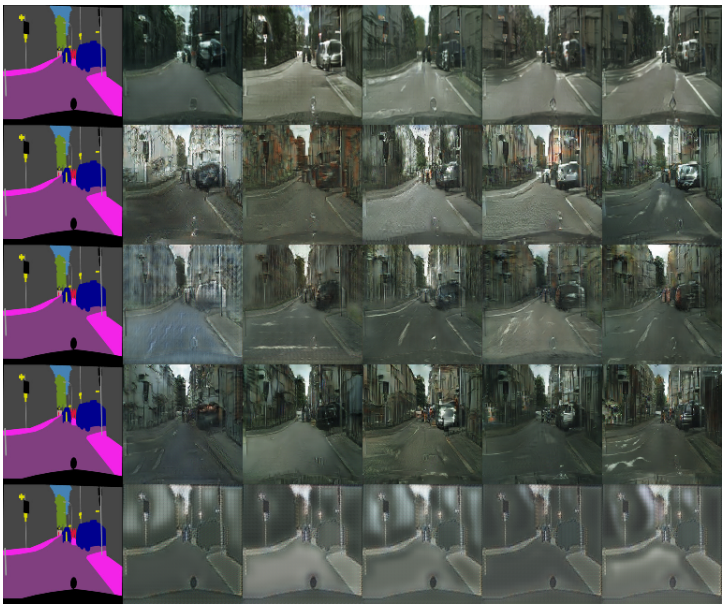

Figure 7.1: Generated images after n epochs. From left to right: input image, 40, 80, 120, 160, 200 epochs. From top to bottom: Zhu et al., Baseline (ours), U-Net generator, no image buffer, Wasserstein loss.

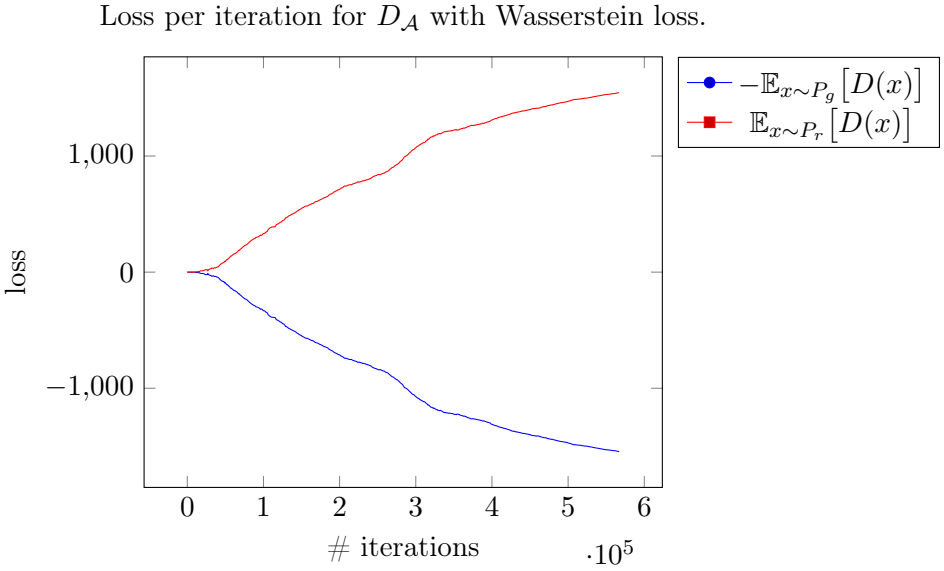

Figure 7.2: CycleGAN with Wasserstein loss and Lipschitz penalty.

Loss per iteration for $D_{\mathcal{A}}$ with Wasserstein loss.

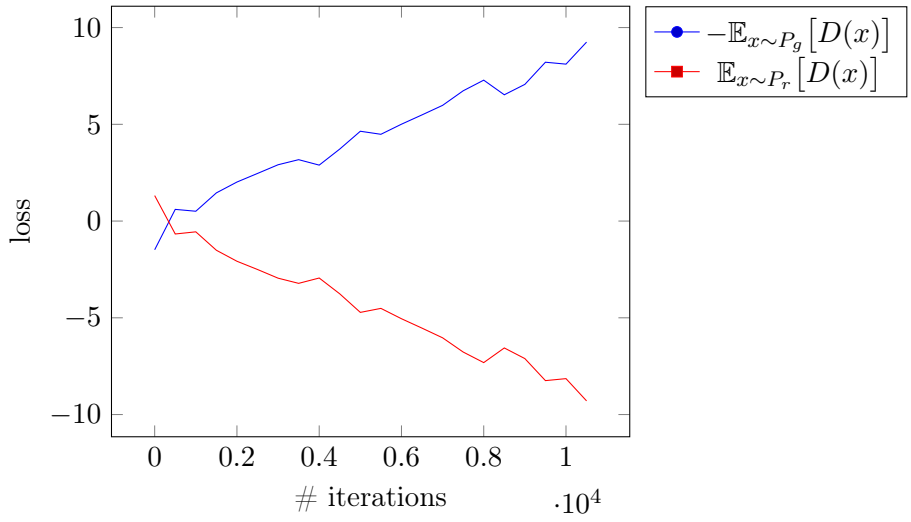

Figure 7.3: CycleGAN with Wasserstein loss and Lipschitz penalty with mean squared cycle-consistency loss and $\lambda_c = 0.5$.

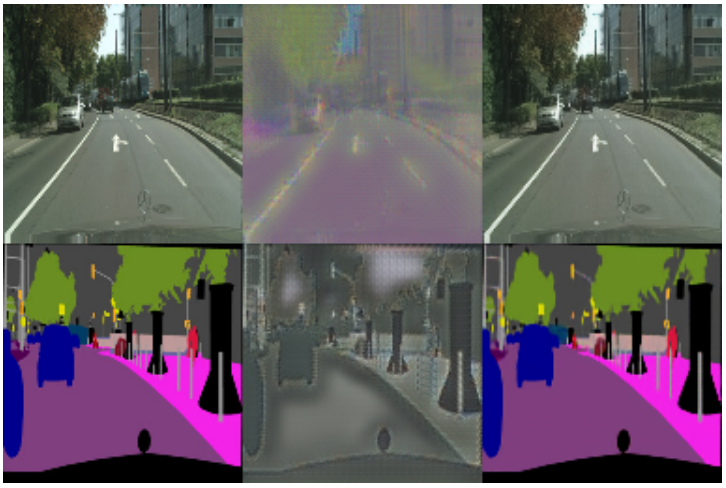

Figure 7.4: Generated images with reconstructed input for CycleGAN with Wasserstein loss. Top row: $\mathbb{E}_{x\sim\mathcal{A}}$, $G_{\mathcal{A}}(x)$, $G_{\mathcal{B}}(G_{\mathcal{A}}(x))$. Bottom row: $\mathbb{E}_{y\sim\mathcal{B}}$, $G_{\mathcal{B}}(y)$, $G_{\mathcal{A}}(G_{\mathcal{B}}(y))$.

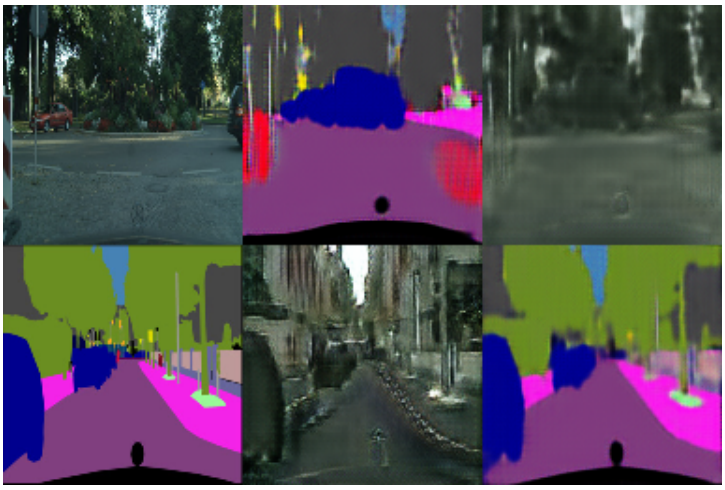

Figure 7.5: Generated images with reconstructed input for Wasserstein CycleGAN and Lipschitz penalty with mean squared cycle-consistency loss and $\lambda_c = 0.5$. Top row: $\mathbb{E}_{x\sim\mathcal{A}}$, $G_{\mathcal{A}}(x)$, $G_{\mathcal{B}}(G_{\mathcal{A}}(x))$. Bottom row: $\mathbb{E}_{y\sim\mathcal{B}}$, $G_{\mathcal{B}}(y)$, $G_{\mathcal{A}}(G_{\mathcal{B}}(y))$.

Chapter 8

Conclusions

In this thesis we have replicated the results of the CycleGAN in its original version and we have evaluated several modifications. We have shown that changing the generator to a U-Net architecture results in a lower FCN score. As such, we conclude that a residual network is a better choice of generator then a U-Net. Changing the training process such that the discriminators are trained with the last generated image instead of a history of generated images has resulted in a slightly better FCN score. Furthermore, we have shown that changing the least squares loss to the Wasserstein loss with Lipschitz penalty makes the FCN score much worse. Finally, we have shown that the Wasserstein loss works better when the cycle-consistency constraint is relaxed by lowering the weight and changing the absolute error for the sum squared error for the cycle-consistency term.

Bibliography

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *ArXiv e-prints*, January 2017.

[2] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. *ArXiv e-prints*, April 2016.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

[4] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved Training of Wasserstein GANs. *ArXiv e-prints*, March 2017.

[5] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. 313:504–7, 08 2006.

[6] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[7] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv e-prints*, February 2015.

[8] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. *ArXiv e-prints*, November 2016.

[9] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. *ArXiv e-prints*, March 2016.

[10] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim. Learning to Discover Cross-Domain Relations with Generative Adversarial Networks. *ArXiv e-prints*, March 2017.

[11] D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, December 2013.

[12] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[13] M.-Y. Liu, T. Breuel, and J. Kautz. Unsupervised Image-to-Image Translation Networks. *ArXiv e-prints*, March 2017.

[14] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. *ArXiv e-prints*, November 2014.

[15] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, and Zhen Wang. Multi-class generative adversarial networks with the L2 loss function. *CoRR*, abs/1611.04076, 2016.

[16] H. Petzka, A. Fischer, and D. Lukovnicov. On the regularization of Wasserstein GANs. *ArXiv e-prints*, September 2017.

[17] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *ArXiv e-prints*, May 2015.

[18] A. Royer, K. Bousmalis, S. Gouws, F. Bertsch, I. Mosseri, F. Cole, and K. Murphy. XGAN: Unsupervised Image-to-Image Translation for Many-to-Many Mappings. *ArXiv e-prints*, November 2017.

[19] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How Does Batch Normalization Help Optimization? (No, It Is Not About Internal Covariate Shift). *ArXiv e-prints*, May 2018.

[20] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from Simulated and Unsupervised Images through Adversarial Training. *ArXiv e-prints*, December 2016.

[21] P Smolensky. Information processing in dynamical systems: Foundations of harmony theory. 1:194–291, 1986.

[22] Y. Taigman, A. Polyak, and L. Wolf. Unsupervised Cross-Domain Image Generation. *ArXiv e-prints*, November 2016.

[23] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. *ArXiv e-prints*, July 2016.

[24] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networkss. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.