Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Conditional Adversarial Networks for Colorization and Stylization of Hand-drawn Sketches

Barbora Dědková

Supervisor: Georgios Tolias, Ph.D.
Field of study: Open Informatics
Subfield: Computer Science
May 2019

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Dědková Barbora**	Personal ID number:	**466307**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Branch of study:	**Computer and Information Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Conditional Adversarial Networks for Colorization and Stylization of Hand-drawn Sketches

Bachelor’s thesis title in Czech:

Použití Conditional Adversarial Networks pro kolorování a stylizaci skic

Guidelines:

The objective of the project is to develop a method for automatic colourization and stylization of hand-drawn sketches. The methodology will be based on Generative Adversarial (GAN) networks [2] and in particular their recently proposed conditional variant, i.e. Conditional Generative Adversarial Networks (CGAN) [1,4]. CGANs are used for image-to-image translation [1] in a variety of different domains. This project applies CGANs to the domain of sketch colorization with the goal of analyzing the impact of different design factors to the performance and injecting prior knowledge of the particular domain into the training.
1. Design and implement CGANs (based on pubicly available implementation in PyTorch).
2. Train CGANS with clipart (cartoon-like) images. The input is the edge-map [3] of a clipart image, while the desired output is the clipart itself.
3. Test how the CGAN generizes to hand-drawn sketches.
4. Performance analysis for factors such as model size, training set size, number of training classes/objects.
5. Exploit the characteristics of the output domain (i.e. smooth/uniform regions) to improve the results.

Bibliography / sources:

[1] Isola, Zhu, Zhou, and Efros, Image-to-Image Translation with Conditional Adversarial Networks, In CVPR17.
[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In NIPS, 2014.
[3] P Dollár, CL Zitnick, Structured forests for fast edge detection, In ICCV 2013.
[4] M. Mirza, S. Osindero, Conditional generative adversarial nets, In arxiv 2014.

Name and workplace of bachelor’s thesis supervisor:

Georgios Tolias, Ph.D., Visual Recognition Group, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **14.02.2019** Deadline for bachelor thesis submission: **24.05.2019**

Assignment valid until: **30.09.2020**

Georgios Tolias, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to thank the supervisor of the thesis, Georgios Tolias, Ph.D. Thank you for your patience, guidance and advice through out the whole year.

Let me also thank ČVUT, my *alma mater*, for supporting me during my studies.

Declaration

I declare that that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 24. May 2019

Abstract

Conditional generative adversarial networks learn a mapping from one domain to another. They are ideal for image transformation tasks such as colorization. The initial domain in colorization is an edge map. The output would be the colorized image. This thesis explores said setting in a domain of hand-drawn sketches. The networks train to colorize the drawings in a vector art or comic style. The main goal is to examine various approaches to the network design and the data set attributes to obtain information about the model's capabilities.

Keywords: Conditional Adversarial Networks, Generative Adversarial Networks, Colorization, Image translation

Abstrakt

Conditional generative adversarial networks se učí mapování z jednoho prostoru do druhého. Jsou ideální k digitálnímu zpracování obrazu, jakým je například kolorování. Počáteční prostor pro mapování je v tomto případě mapa hran či obrysů. Cílový prostor je pak obarvený obraz původních obrysů daného předmětu. Tato závěrečná práce se věnuje mapování ručně kreslených skic na jejich obarvené verze. Konkrétně se soustředí na obarvení skic ve stylu klipart a komiksových obrázků. Hlavním cílem práce je prozkoumat různý design použitých sítí nebo vlastnosti datasetů a získat tak informaci o možnostech této metody.

Klíčová slova: Conditional Adversarial Networks, Generative Adversarial Networks, kolorování, zpracování obrazu

Contents

Figures

Chapter 1

Introduction

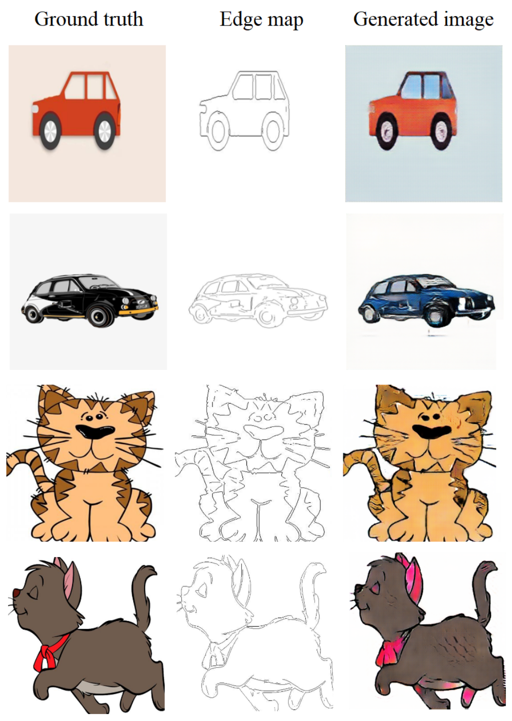
Recent advances in deep learning made tasks such as object or image recognition possible. There are plenty of implementations and approaches that are very good at it. Especially in domains such as face recognition [Wang and Deng, 2018] or vehicle registration plate recognition [Laroca et al., 2018] that are useful in security and surveillance systems. Recognizing an object is one thing. Generating the object and the whole image is another thing. Even that is starting to become feasible nowadays.

Image-to-image translation (or image transformation) is active and challenging research in computer vision and graphics problems. The goal is to learn a mapping between an input domain and an output domain. That is, transform an input image into an output image. This work focuses on the colorization of hand-drawn sketches. The input image is a black and white sketch; the output image is a colored version of the sketch.

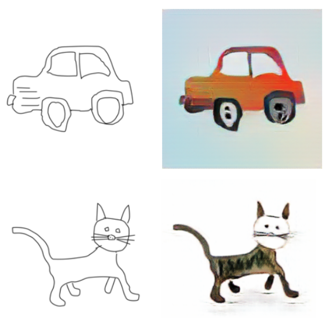
Until recently, we had to hand-design a lot of algorithms and cost function to generate believable, accurate, and perfect results. In addition, the hand-designed method usually serve only for some very specific task. Some of the examples might be transformation of seasons in outdoor images [Laffont et al., 2014], transformation of human faces from young to old [Upchurch et al., 2016], changing colors [Zhang et al., 2016], or materials.

Not long ago introduced Generative Adversarial Networks (GANs) tackle this problem trying to generalize the approach. Two networks are trained against each other, pushing each other to do better. One of the networks generates the images and the other tries to determine if the images are generated or come from the training data.

We use Conditional Adversarial Networks for image transformation (or translation). The condition of the network is the input image. In the change of seasons example the condition (input image) would be a summer landscape, and the output the same landscape but in winter. In our case, it will be the sketch as the initial condition and a colorized image as the output.



(a) : Training phase.



(b) : Testing phase.

Figure 1.1: The training data set 1.1a uses vector art and clipart images, denoted as ground truth, and their edge maps. The network generates colored versions of the edge maps. After the training, we test the model by feeding it hand-drawn sketches from the same domain it trained with. Figure 1.1b shows the output colored sketches.

1.1 Motivation

The progress of conditional GANs development is enormous. If we want to measure it somehow, we can look at the number of papers getting published in this area. It is growing exponentially. Or maybe the fact that GANs can produce high-resolution images indistinguishable from reality can serve as a proper measurement. [Bau et al., 2018] offers the research of GAN capabilities and makes a step towards understanding the network's internal mechanisms. The work also provides an application for painting with GANs. It has several interactive tools for painting a scene in real time by labeling areas. The number of similar applications is growing, and soon they could be used commercially.

GANs and cGANs have considerable applicability in real life. They can be used in the entertainment industry such as film production, animation and game development. An example from this branch of industry is [Jin et al., 2017] with their auto-generation and colorizatin of Anime characters. Another use could be the colorization of old black and white movies or photos. There is also [Ma et al., 2017] with Pose Guided Person Image Generation. Their work proposes a method for generating images of people in various poses. It could be utilized in the fashion or advertisement industry.

The technology of image generation could potentially save a lot of work and resources. However, there is still space for improvement, and the research is at the beginning. The motivation for further investigation of this topic is clear.

1.2 Related Work

There are many successful approaches to sketch colorization and image generation. User-guided Colorization [Sangkloy et al., 2016] proposes a deep generative network for colorizing sketches. The network enables the user to control the generated output by using sparse color strokes across the sketch. In contrast with our network whose only input is the gray-scale sketch. Our network has to put the color together on its own. Sometimes, that might result in blended or dull color instead of a brighter one. Furthermore, our approach focuses on hand-drawn sketches made by non-artists. Training on a set of comic and vector art images. Sangkloy et al. uses more sophisticated and elaborate form of sketches. The different characteristics of sketches can have large impact on the results or even the training itself.

Another approach to colorization focuses on generation of Anime facial images [Jin et al., 2017]. It incorporates label information as a condition of the network, the same way we use edge maps, to control the generation of the output image.

In the unpaired setting, we have Unpaired Image-to-Image Translation [Zhu et al., 2017]. Convenient for cases where the training data is not available in a paired form. In our case, we have each training gray-scale sketch paired with its colored opposite. The unpaired setting investigates the relationship between two data domains: X and Y. This approach is often called Style Transfer. An advantage of the unpaired setting is a more straightforward collection of training data. It can be done without much extra effort, in contrast with the paired setting. We will discuss data collecting and preprocessing further in Chapter 4.

1.3 Thesis Outline

Let us present a short overview of the structure of the work.

- Chapter 2 introduces the background of Adversarial networks and generative models. It explains the concept of adversarial game and its value for the generative models. It also briefly looks into the conditional setting of the networks.
- Chapter 3 considers more technical details of the networks, explaining the architectures of both adversaries. It also covers the training algorithm of the networks and analyses individual terms of the objective function.
- A network is not complete without a data set to train on. Chapter 4 describes the characteristics of the sets and the data itself.
- Chapter 5 introduces some of the results of this work in a quantitative way. It presents the setups and approaches we investigated in order to overcome the difficulties presented by this work.
- The final Chapter 6 summarizes and evaluates the work.

Chapter 2

Generative Models and GANs

The second chapter of this work introduces the background of all Generative models. What generative models are and why we might want to use them or what does the objective function derive from.

The first section describes the fundamental understanding and taxonomy of generative models. The second section explains how GANs work and tries to explain their objective function comparing them to a minimax game. The last section looks at the conditioned setting of GANs.

2.1 Generative Models

A generative model is a model that draws samples from a training set. The training set is identified by a distribution p_{data}. This distribution is not know and the model tries to estimate it only by analyzing the samples. We will call its estimate p_{model}.

A common method for training generative models is using the *maximum likelihood* (ML) method. The likelihood is the probability that the model assigns to the training data. The goal is to maximize this likelihood by fitting the parameters θ of the generative model.

Some generative models do not use ML, however, they will not be discussed as GANs use it. The models that use ML divide into two groups, depending on if they try to explicitly estimate the density function $p_{model}(\mathbf{x}, \theta)$ ($\mathbf{x}$ being the set of n training examples and θ the set of parameters) or only draw samples from p_{model}. The likelihood $\mathcal{L}$ is the joint probability of all samples x_i from $\mathbf{x}$:

$$\mathcal{L}(\mathbf{x}, \theta) = \prod_{i=1}^{n} p_{model}(x_i, \theta) \tag{2.1}$$

If the model defines probability function p_{model}, we can apply it in the likelihood expression (2.1) and follow the gradient uphill.

The challenging part is designing a model that captures all of the complexity of the data and that is computationally manageable. To maximize the likelihood, we have to design the model very carefully or make approximations. These models are represented by *Variational Autoencoder* [Kingma and Welling, 2013] or *Fully Visible Belief Networks*.

If the probability distribution p_{model} is implicit, we can at least draw samples from it. Some of these models drawing samples use *Markov chains* to stochastically transform a sample into another sample from the same distribution. It must be run several times to generate the sample.

Other models, such as GANs, can generate a sample in a single step. They do not need Markov chains, and the generator has fewer restrictions relative to *Boltzmann machines* [E. Fahlman et al., 1983] or *generative stochastic networks*, GSN [Bengio et al., 2013], which both use Markov chains. Some models (including GANs) can do both drawing samples from p_{model} and explicitly estimating the distribution. Even though GANs focus primarily on generating samples.

2.2 Generative Adversarial Networks

We can understand the adversarial modeling framework as a zero-sum game between two players. The first one is called *generator*. It creates samples drawn from the distribution p_{model}. The second one, *discriminator*, returns a number in the interval [0,1] which determines the probability of the sample being fake (created by the generator) or real (sampled from the training data). They set up a competition, trying to beat each other by improving their models.

Let us imagine the generator as an art forger and the discriminator as an art analyst. The analyst is trying to recognize an original piece of art from a forgery, and the forger is creating fake pieces. They are both improving their methods and techniques of how to do so over time. The process is illustrated in Figure 2.1

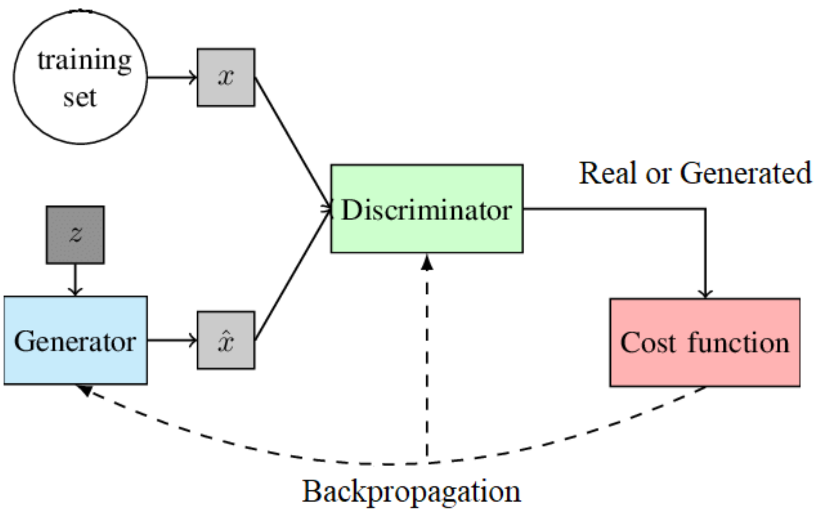


Figure 2.1: Generative Adversarial Networks Framework [Ponti et al., 2017], illustrating the training procedure. The only input into the generator is a latent variable z. A drawn sample from the training set, x, along with $\hat{x}$, the generated sample, are the inputs of the discriminator. The discriminator decides if the received sample is generated or real. The result is processed by the cost function (in detail in Chapter 3) and backpropagated to the networks.

The generator (forger) tries to maximize the probability of the discriminator (analyst) making a mistake. The ideal solution would be G recovering the distribution of the data and D returning 0.5 all the time. That is, the generated samples would be indistinguishable from the samples drawn from the training set.

Both networks are multilayer perceptrons. They are represented by two differentiable functions, both with respect to its inputs and its parameters. The generator is a function $G(z, \theta_G)$, that takes z, a noise variable from a prior noise distribution $p_z(z)$, as input and uses θ_G as parameter. The discriminator is a differentiable function $D(x, \theta_D)$ that takes x as input and θ_D as its parameter. The input x can be either the output of G or a sample from the training set, as we can see in Figure 2.1. Function D returns a scalar, representing the probability that x is real rather than generated.

2.2.1 Discriminator's loss

The discriminator's loss function is the cross entropy loss function. It is defined as

$$H(p,q) := -\sum_i p_i \log q_i$$

where p is real data distribution and q is the estimated distribution. We can sum over the individual components, denoted with i, because they are discrete. If they were continuous, we would use integral.

We can apply the cross entropy on a binary classification task which we want the discriminator to perform. We already have the function $D : \mathbb{R}^n \to [0,1]$ which is the probability of a data point x_i being in the first class. Then the probability of this point being in the second class is $1 - D(x_i)$. For one individual data point x_1 we have:

$$H((x_1, y_1), D) = -y_1 \log D(x_1) - (1 - y_1) \log(1 - D(x_1))$$

where y_1 is the label of x_1. For the data distribution defined as $P[y_i = 0] = 1$ if $y_i = 0$, $P[y_i = 1] = 1$ if $y_i = 1$, we know that one term in the formula above is zero. We can generalize this formula for N elements (x_i, y_i):

$$H((x_i, y_i)_{i=1}^N, D) = -\sum_{i=1}^N y_i \log D(x_i) - \sum_{i=1}^N (1 - y_i) \log(1 - D(x_i)).$$

In GANs the elements x_i come either from the real data distribution p_{data} or from the generator as $x_i = G(z)$ where z comes from p_{model}. We restrict this proportion to be exactly 1/2. Then we can afford to substitute the labels y_i with 1/2. We replace the sum with expectation, $\log(1 - D(x_i))$ becomes $\log(1 - D(G(z)))$. We get the discriminator loss

$$\mathcal{L}_D = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}}[\log D(x)] - \frac{1}{2}\mathbb{E}_z[\log(1 - D(G(z)))]. \qquad (2.2)$$

The discriminator tries to minimize this loss and the generator wants to maximize it.

2.2.2 Minimax game

In the concept of a minimax zero-sum game, we need to get a zero when all player's costs are summed. In this game we have $\mathcal{L}_D = -\mathcal{L}_G$. The value function is:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_z[\log(1 - D(G(z)))]. \qquad (2.3)$$

The value function is equal to the negative discriminator loss which is equal to the generator loss.

In the early stage of learning, D can reject samples with high confidence because the difference between generated and real samples is obvious. Then $log(1-D(G(z)))$ saturates, the gradient of $\mathcal{L}_D$ approaches zero along with the gradient of $\mathcal{L}_G$, and the generator stops updating. One solution [Goodfellow, 2017] to this can be changing the generator's loss. Instead of minimizing $log(1 - D(G(z)))$ it can maximize $log(D(G(z)))$. This objective function provides much stronger gradient while keeping the main idea of the loss function.

2.3 Conditional Generative Adversarial Networks

We do not have any control over the mode of generating the samples in an unconditioned generative model. It is possible to steer the data generation by conditioning the model on additional information. The information can be a discrete label, text, images, and others.

We already know from Section 2.2 that G is a mapping function $G(z, \theta_G)$ from a noise variable z and parameters θ_G to data space. The discriminator function $D(x, \theta_D)$ outputs a value that gives the probability of x coming from the data distribution rather than from G.

We condition both G and D with y, the additional information. In the generator the prior noise z and the information y combine. The discriminator adds y to its input. Our value function 2.4 changes to:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x,y)] + \mathbb{E}_z[\log(1 - D(y, G(z,y)))]. \quad (2.4)$$

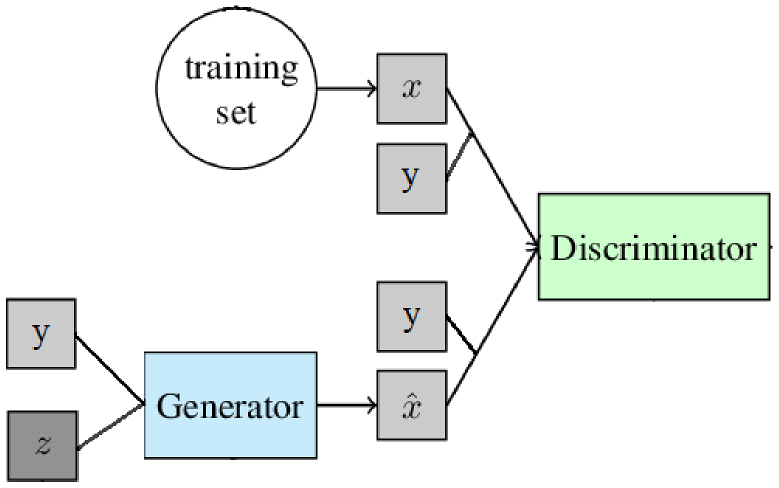


Figure 2.2: Conditional Generative Adversarial Network scheme. The generator takes a noise variable z and additional information y as input. Outputs a sample $\hat{x}$ which forms the input of the discriminator along with y. The discriminator can also get the same form of input but with the sample x from the training set. The rest of the conditional network is the same as in GANs.

The conditional GAN (cGAN) objective is $G^* = arg\min_G \max_D V(D, G)$. It is beneficial to mix this objective with a more traditional loss. Some approaches work with L2 distance, for example. The discriminator's objective remains the same. The generator, on the other hand, tries to be close to the ground truth in an L2 sense, in addition to trying to fool the discriminator. The additional loss can look like:

$$\mathcal{L}_{L2}(G) = \mathbb{E}_{x,y,z}[\|x - G(z, y)\|_2]. \tag{2.5}$$

The final objective would be:

$$G^* = arg\min_G \max_D V(D, G) + \lambda \mathcal{L}_{L2}(G) \tag{2.6}$$

where λ is a weight of the additional adversarial loss.

Chapter 3

Method

This thesis builds on the PyTorch implementation of Image-to-Image Translation [Isola et al., 2017]. The following chapter describes the architecture of the network and the process of training.

The first section focuses on the architectures of the generator and the discriminator, which are the components of an adversarial network. The following section describes the method of training the network, presenting a learning algorithm. Finally, we can find more details about the additional terms of the objective function in the last section of this chapter.

3.1 Network Architecture

The architecture of both generator and discriminator is adapted from [Radford et al., 2015]. They introduced several modifications to commonly used CNN (Convolutional Neural Network) architectures. Both networks use standard modules of the form convolution-BatchNorm-ReLu.

The new approach replaces any pooling layers (maxpooling) with strided convolutions. Both in the generator and the discriminator, it allows the networks to learn their own spatial upsampling or downsampling.

The second trend is to replace fully connected hidden layers for deeper architectures. The first layer of the generator is fully connected, but then reshaped into a 4-dimensional tensor and used as the start of the convolution stack. The last convolution layer of the discriminator is flattened and then processed by a single sigmoid output.

The third change is introducing *Batch Normalization* [Ioffe and Szegedy, 2015]. It can be applied to any set of activations in the network. The input of each unit is normalized to have zero mean and unit variance. The batchnorm layer is applied everywhere except generator's last (output) layer and discriminator's first (input) layer. This normalization prevents the generator from collapsing all samples to a single point and helps the gradient flow in deeper models. It enables higher learning rates, and it prevents small changes in activations in gradients to become large suboptimal changes. Thanks to Batch Normalization, the backpropagation through the layers is unaffected by the scale of parameters.

Other suggestion might be to use *ReLu activations* [Arora et al., 2016] everywhere. It allows faster saturation and covering the color space of the training distribution. Only in the last layer of the generator we use Tanh function. We can use leaky ReLu activation in the discriminator in all layers.

3.1.1 Generator

In image-to-image translation, we map a high-resolution input grid to a high-resolution output grid. Both input and output look different but have the same underlying structure. In our case, those are the edges of the objects. It is desirable to keep the information about structure throughout the colorization process. The generator architecture is designed to fulfill these requirements.

Previous solutions use a encoder-decoder network. The information flows through a network, at first downsampling through the layers until a bottleneck layer. Then subsequently upsampling back again. The information has to pass through all the layers, including the bottleneck layer. The network uses a *U-Net* with skip connections [Ronneberger et al., 2015] to pass some additional information between corresponding layers. We are exploring various sizes of the U-Net in the thesis.

We have seen in Section 2.2, that the generator takes a noise variable and various parameters as input. In this work, we are not going to use the noise variable, as suggested by the Pix2Pix implementation [Isola et al., 2017], because they do not offer significant changes and variations in the results. The rest of the input in cGANs (edge maps in our case) is sufficiently complex to compensate for the noise. The mapping learned by the model is deterministic.

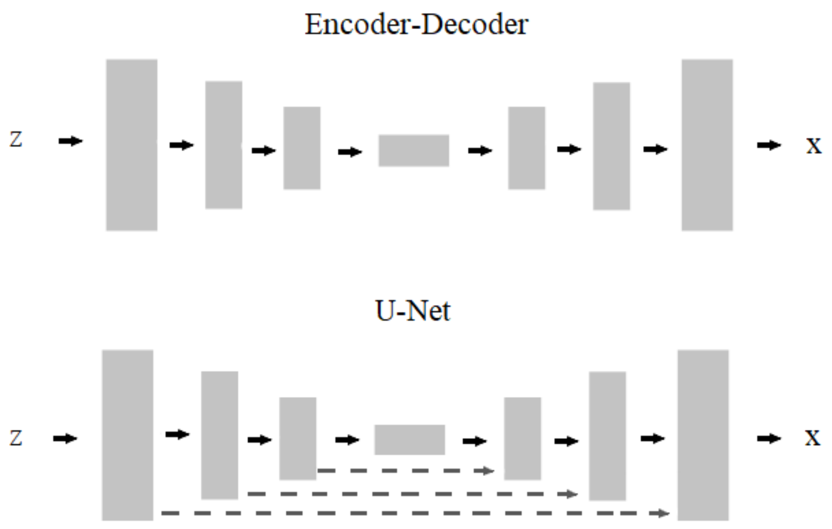


Figure 3.1: The figure shows the architecture of an Encoder-Decoder network. Follows a U-Net architecture with skip connections between layer i in the encoder part and $n - i$ in decoder, where n is the total number of layers.

3.1.2 Discriminator

The architecture of the discriminator is a *PatchGAN*, a network that penalizes structures only at the scale of patches. It runs convolutionally across the image deciding for each patch if it is real or fake. In the end, the discriminator takes the average of all the responses as a final response for the whole image. It secures the modeling of high-frequency structures. The correctness of low-frequencies is left to the additional loss term we saw in Equation 2.5.

We explore how the size of the patches affects the results in Chapter 5, more in Figure 5.2.

3.2 GAN training

For training the networks, we use *minibatch SGD* (Stochastic Gradient Descent) shown in Algorithm 1. We use $k = 1$ as a hyperparameter in the algorithm. That means one gradient descent step on the discriminator and one on the generator. Some approaches use rather more steps for the generator to slow down the learning of the discriminator. In this work, before the update of the discriminator, we instead divide the objective by 2. Which also slows down the rate of learning relative to the generator. For the gradient optimization we apply *Adam optimizer* [Kingma and Ba, 2014] instead of momentum to accelerate the training.

Algorithm 1: GAN training with minibatch stochastic gradient descent. Hyperparameter k is the number of times we want to update the discriminator.

```
1 for number of training iterations do
2     for k steps do
3         • Sample minibatch of $m$ paired examples $\{(x_1, y_1), ..., (x_m, y_m)\}$
            from data generating distribution $p_{data}$.
4         • Update the discriminator by ascending its stochastic gradient:
              $$\nabla_{\theta_D} \frac{1}{m} \sum_{i=1}^{m} \log(D(x_i, y_i)) + \log(1 - D(G(x_i), y_i)).$$
5     end
6     • Sample minibatch of $m$ paired examples $\{(x_1, y_1), ..., (x_m, y_m)\}$
        from data generating distribution $p_{data}$.
7     • Update the generator by descending its stochastic gradient:
          $$\nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(x_i), y_i)).$$
8 end
```

3.3 Cost Functions

We saw the final objective of the generator in Equation 2.6. The additional loss term uses L2 distance, but we can use any other loss, and we can even combine more than one. Each additional term always has a weight factor, which is another parameter for us to optimize.

In this work, we explore L1 distance, L2 distance and total variation (TV) loss. L1 distance might encourage less blurring. For that reason we prefer L1 over L2 distance, nevertheless there are experiments with both, Figure 5.3. Finally, we also add total variation loss to encourage smoothness in the output. We find it to work very well with the term of L1 loss.

L1 or Euclidean distance between two images x and y measures the distance between each corresponding pixel of the images using Euclidean norm. It is the mean absolute error between the pixels.

$$\sum_{i,j} |x_{i,j} - y_{i,j}| \tag{3.1}$$

where $x_{i,j}$ is a pixel in location (i, j) of the image x.

L2 is another way to measure distance between pixels, similar to L1.

$$\sqrt{\sum_{i,j}(x_{i,j} - y_{i,j})^2} \tag{3.2}$$

We use both L1 and L2 terms of the loss to make the generated samples closer to the ground truth (in color, sharpness, edges). In comparison, the objective function alone only makes the samples real. Real in the sense of fooling the discriminator, although they do not necessarily look close to the ground truth.

Total variation loss encourages spatial smoothness in the generated samples. It removes noise from the image by diminishing differences between neighboring pixels. It can be defined as this:

$$TV(x) = \sum_{i,j}((x_{i,j+1} - x_{i,j})^2 + (x_{i+1,j} - x_{i,j})^2)^{\frac{\beta}{2}} \tag{3.3}$$

where β is a hyperparameter. We are using $\beta = 2$ in our work. subfig [demo]graphicx

Chapter 4

Data set

The selection of a training data set and test data is very important in deep learning. It affects quality and capabilities of the networks. It is one of the considerable challenges of this work. This thesis explores the capabilities of the network and examines a lot of aspects of the data. We are going to consider the amount, size, or cleanness of the image content.

We investigate the paired setting of the data. Our training set is composed of aligned image pairs. The first image is a black-and-white edge map of the second image, a comic or vector art styled image. We will look at the edge detection closer in Section 4.1.2. The following sections discuss the importance of individual characteristics of the data set and the data itself.

Regarding the sketches we are going to use, there are a lot of styles of hand-drawn sketches. Such as pen-and-ink illustrations or elaborate pencil-like drawings with shading. The characteristics might be very diverse. What works for doodle-like drawings might not work for synthetic sketches we generated algorithmically. We may overfit to a particular style of sketch even with the various augmentations, random cropping, random brightness adjustment and random cut-off.

We are aiming to colorize very simple sketches of non-artists. The difference in style and the art level gap between the sketches in the training data and the test data is a challenge to overcome.

4.1 Training data

The majority of our training data comes from the BAM data set (The Behance Artistic Media Dataset) [Wilber et al., 2017] from their category of vector art. We also add some images from web crawling, but not a significant amount. The BAM data set offers data from various categories such as people, animals, cars, nature, etc. A careful selection process must take place in order to choose a suitable data set in size and content.

4.1.1 Selection

It is a quite challenging task, so, we start by simple cases of a single object category with many clean examples. Categories such as *bicycles, flowers* or *trees* were disregarded either for small coloring area or the total amount of data available. The category of *people* involves a lot of inter-image variance, which is a challenging aspect to work with, although it is a category with the highest number of samples. A lot of available data sets, in general, suffer from noise and high variance in content even when categorized. Cluttered images might be useful for other work but in our case, not exactly. We choose two reasonable categories - *cats* and *cars*.

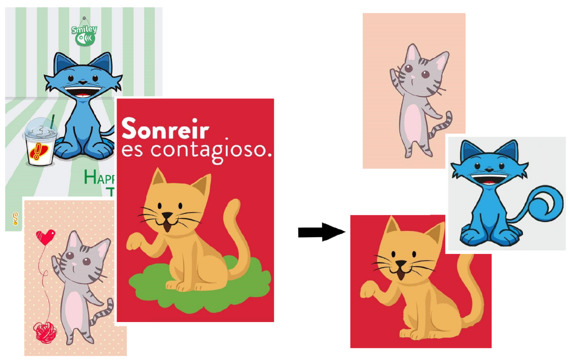


Figure 4.1: It is essential for the training to have a clean set of samples. We find that a network trained on the right set of images produces better results. (See Figure 5.6)

We manually select plausible images from the two categories to create a more clean and well-suited data set. The clean data set has a uniform background without redundant objects or text. In each category, we get around 400 samples from the initial amount of over a thousand.

4.1.2 Edge Detection

After careful selection of the training data, we need to create a pair image for every training sample. We use P. Dollar's edge detector [Dollár and Zitnick, 2014] for the extraction of edges. The goal is to make the edges as similar in style as possible to hand-drawn sketches of non-artists.

After the edge detection, we need to adjust the edges further to make them appear more like a sketch. One of the techniques is to use a boundary for turning the edges from grey-scale to black and white. Each pixel is compared to this boundary, and depending on if it is higher or smaller, it is colored black or white. After exploring different values of the boundary, we learn that value from 0.75 to 0.8 works the best. We get rid of a lot of minor insignificant edges in the image, that a sketch would not have. Another idea is to thin out the edges. The thinner they are, the more they resemble the irregular lines of a hand-drawn sketch. After that, the result is comparable with our test data, which was our goal (see Figure 5.7).

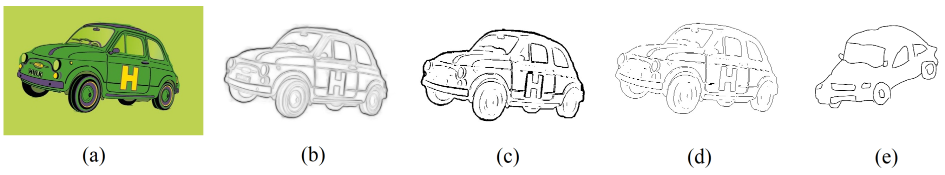


Figure 4.2: The process of edge detection. The first image, 4.2a, is the ground truth from which we detect the edges. 4.2b is a grey-scale edge map from the detector. Subsequently, we turn the grey-scale into black and white (4.2c) and thin the edges out (4.2d), so that they all have the same thickness. The final form of the edge map still has more details than the sketch (4.2e), in this case.

4.1.3 Amount and size of samples

We mentioned in Section 4.1.1 that the clean data set has around 400 images. How do we know that is enough for the model to train and produce reasonable results? Generally, both over-constrained and under-constrained models result in poor performance.

We discovered that having 400 samples is sufficient to produce decent results. However, it depends on the quality of the samples. We found out that in the category of *cats*, there is more inter-image variance (the cats can be sitting, standing, facing different directions), than in the category of *cars*. A quick hand-drawn sketch of a car is usually in 2-D, occasionally in 3-D.

Most of the time, it is unclear where the front of the car is. Therefore the sketches look more similar to each other than the drawings of cats. That is one of the reasons we get overall better results for cars than cats when training with a 400-sample set.

We experiment with augmenting the data set using standard techniques [Perez and Wang, 2017] of augmentation. In all experiments, we use flipped images in addition to the initial 400, unless told otherwise. Other than that, we apply random cropping, color jitter, or rotation.

The size of individual samples varies as a result of the initial selection followed by the augmentation. The network scales and crops the images at the beginning of the training according to the size it needs.

4.2 Test data

The network is training with clipart (cartoon-like) images where the input is the edge map of a clipart image, and the output is its colorized version. We already explained in Section 4.1.2 how we exploit the characteristics of the input. We attempted to make the edge map resemble the hand-drawn sketch as much as possible.

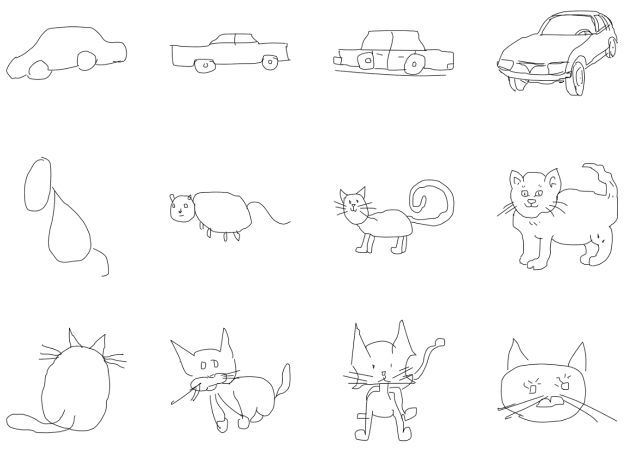


Figure 4.3: Examples of hand-drawn sketches. The first two rows show the progress of the artistic level of the drawings, getting more sophisticated and detailed from left to right. The last row shows the variance in the *cats* class of sketches.

Our test data consists of hand-drawn sketches from non-artists. The sketches often lack details and the model has to manage the coloring of large areas without any guidance of the edges. That might sometimes result in visual artifacts such as small patches of the same pattern in place of a smooth colored area. We are going to investigate this further in Chapter 5. Figure 5.4 in that chapter shows the impact of the sketch style on the results. We are also going to test the model on edge maps of clipart images, that it did not see during training, for comparison.

Chapter 5

Experiments

This chapter presents some of the experiments analyzing the characteristics of both discriminator and generator as well as of the training and test set. We try to exploit the domains we are working with as much as possible. We want to use the knowledge of the data to our benefit and play with different additional loss terms or various patch sizes and other components of the network to put that knowledge to use.

Earlier results of this work showed some unwanted visual artifacts such as tiling effects, blurred lines, and small color spots. In the following section, we are going to look at these problems both from the architecture and model aspect and from the data point of view.

In all experiments we are going to use 70×70 PatchGAN, U-Net of size 256, batch size 1, cGAN+L1 loss, flipping and random cropping of images unless specified otherwise. Batch size set to 1 is called *instance normalization* [Ulyanov et al., 2016]. It is applied on all layers of the U-Net. It skips over the bottleneck layer and it zeros out the activations there. We find that it produces more stable and smoother results. However, we investigate larger batch sizes and find that it encourages colorfulness in the results, although at the cost of other unwanted aspects.

5.1 Tiling effect

In some results we observe undesirable patch-like artifacts. In some cases a longer time of training is sufficient to eliminate them, Figure 5.1. When that is not an option, we try to investigate the patch size of the discriminator. We would like it to penalize the sample in a bigger scale to smooth out the result. See effects of patch size in Figure 5.2. Another approach to solve this problem is using a modified cost function to encourage smoothness of the output. In Figure 5.3 we compare different additional terms in the loss function.

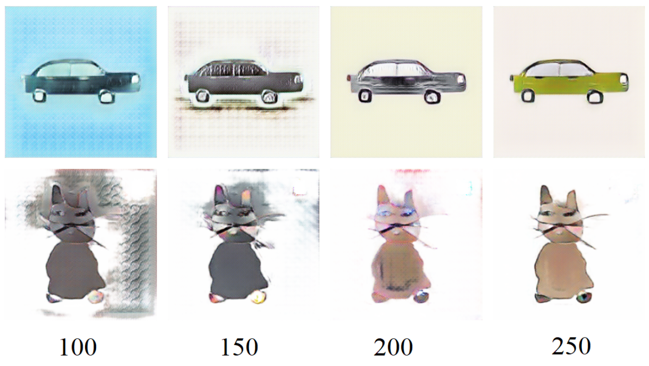


Figure 5.1: Results generated by a model from various epochs of training. Beneath the generated samples are the epoch numbers. The training was run on a 256 U-Net with cGAN loss without any additional loss terms. Notice the tiling effect going away with increasing number of epoch.

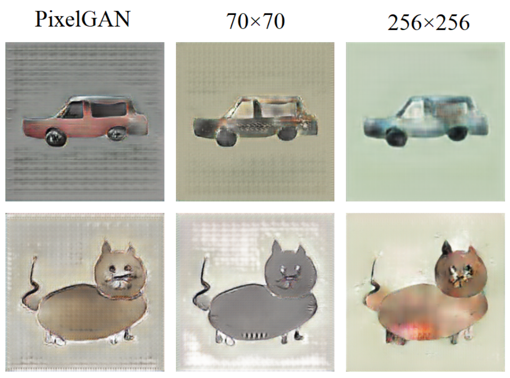


Figure 5.2: Patch size. PixelGAN penalizes every pixel of the image individually. The result is sharp but has too much noise and we would like it smoother. Patches of size 70×70 fulfill this need, however they still cause some tiling effects. A patch of the same size as the sample creates smooth samples without sharp details.

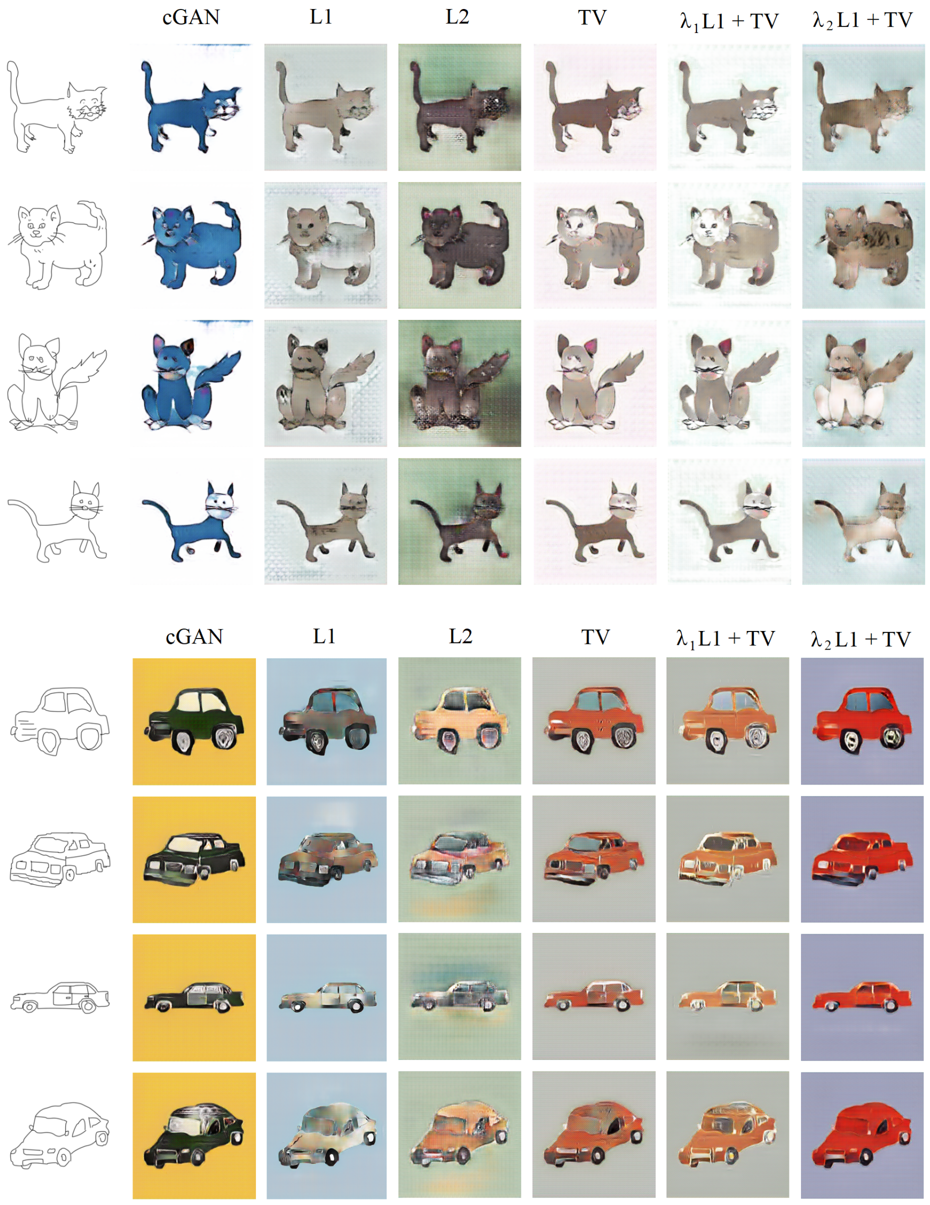


Figure 5.3: The first column presents the sketches to be colorized. The first colorized column of the results is trained with a cGAN loss and no additional loss terms. The rest of the columns add one or more terms to the cGAN loss. We also explore different weight factors of the terms. We used the following parameters here: 100 for the L1 and L2 columns, 1e$-$5 for the TV loss, $\lambda_1 = 10$, and $\lambda_2 = 50$. We can see, that L1 and TV encourage more smoothness in the result that L2. Notice that cGAN's goal is to generate samples that are real to the discriminator. This alone does not mention anything about how similar the samples are to the ground truth. The results can end up in a local minimum close to the input and not improve further. In comparison, the additional terms such as L1 and L2 force the results to be closer to the ground truth. However, if it is unclear what color or structure to use, L1 and L2 average the color and blur the edges to ensure closeness to everything. CGAN, on the other hand, does not enforce the closeness to the truth and therefore chooses sharper colors and edges which might result in unwanted artifacts.

5.2 Training data amount and style

We saw in Figure 4.3 the variance in content of the *cats* category. There are many poses the cats can be in and compared to the *cars* category we need more training data to generate results of the same quality. We investigate how the amount of the training data affects the results in Figure 5.5. The style of the sketches is also crucial to obtain reasonable results. See more in Figure 5.4.

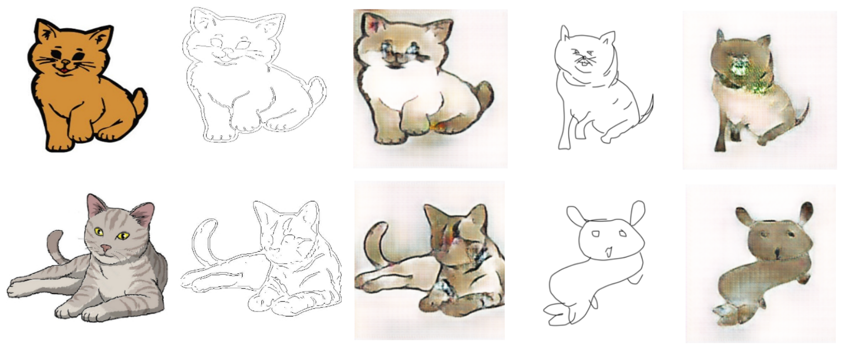


Figure 5.4: Difference in the artistic level of the sketches. The leftmost column is the ground truth, followed by its edge map and the colored version of the edge map generated by the trained model. Follows a badly hand-drawn sketch of a cat in a similar pose as the previous one. The colored version on its right is very different from the colored edge map, even though it is colorized by the same trained model of the generator. We can see that the input data have a great impact on the quality of the results.

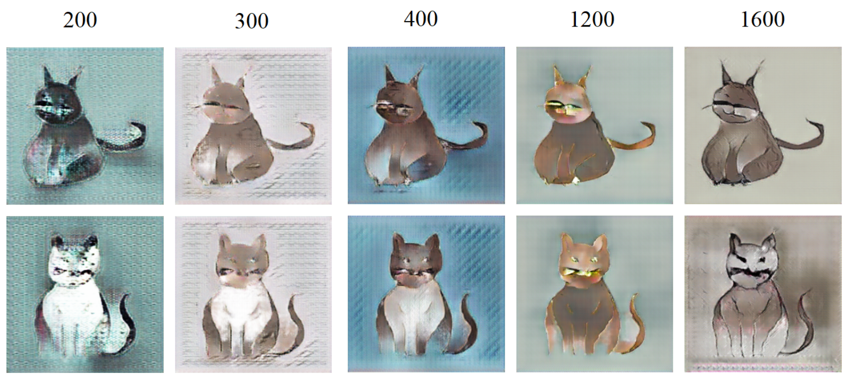


Figure 5.5: Amount of data. The starting data set without augmentation has around 400 samples. We train with 200 and 300 samples too. We augment initial 400 to 1200 samples by flipping the images and cropping them. Adding random color jitter and rotation achieves the last data set of 1600 samples. The setting of this experiment is otherwise the same for all the data sets.

Notice, in Figure 5.5, that more training samples help to reduce the tiling effect. Training with less than 400 samples does not offer convincing results. We find that exaggerated augmentation leads to unstable results. A few samples are generated well (first row of the 1600 set), but the majority completely fails. The experiment uses cGAN + L1 loss. The cGAN loss has a quality of creating a structure, that does not exist in the edge map input when there is not enough information. It also visualizes colors the same way. It is one of the causes of visual artifacts in the images. Notice this phenomenon in the previous Figure 5.3 in the first cGAN column. The L1 term has an opposite approach when uncertain. By trying to be close to the ground truth it produces average color and edges. This is both beneficial and unhelpful. It repairs the tiling effect and partially eliminates some other visual artifacts, but it can also blur the image and produce grayish outputs.

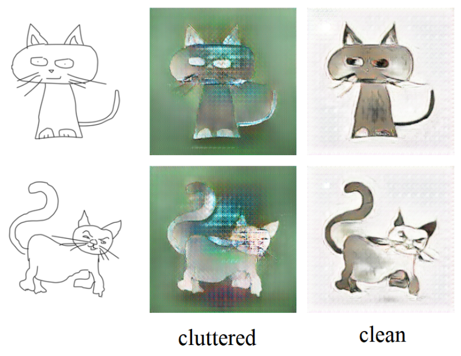


Figure 5.6: Cleanness of data set. The experiment used data of the same size and amount with difference in the cleanness of the samples. A representation of both data sets can be found in Figure 4.1. Training with the clean samples displayed better results in all experiments.

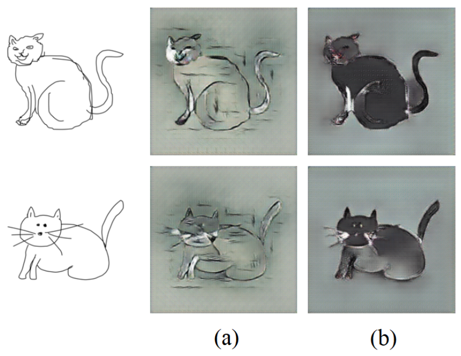


Figure 5.7: This experiment investigates the behavior of the model when trained with edge maps that have been thinned out (as described in Section 4.1.2) and that have not. 5.7a is without thinning; 5.7b is a result from the thinned data set.

An easy way to obtain much more data from our domain (clipart, vector art), besides augmentation, is from cartoons. We test the capabilities of our network on a short video from the series The Simpsons [Brooks, 2016]. The experiment takes the first part of the episode (around 4800 images) as the training set, and we test it on the rest of the episode. The images are much more complex than the vector art we were experimenting with before. Our goal in this experiment is to train the network to recognize the characters. The results in Figure 5.8 partially reach the goal. Although when tested on hand-drawn sketches of the characters, the method fails to produce reasonable results.

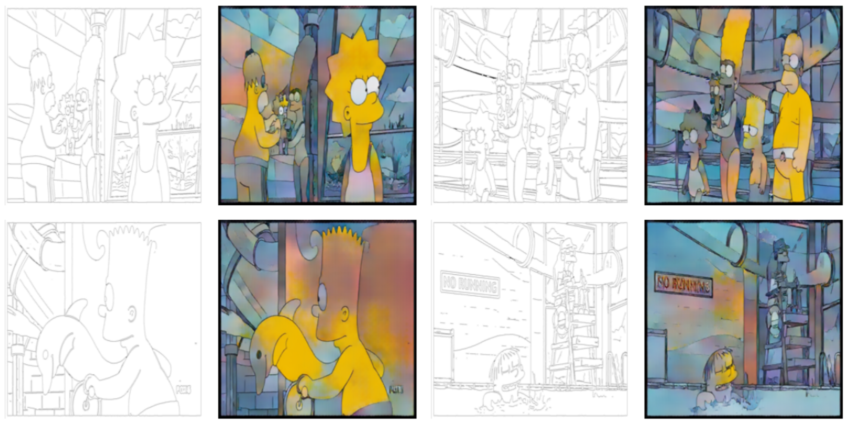


Figure 5.8: Cartoon data set [Brooks, 2016]. The network trained on bigger data set learns to recognize some of the characters. The background and other objects in the scene are colored very poorly.

We conducted another experiment with the cartoon data set. It takes the whole episode (7000 images) and selects every 20th image resulting in a total of 350 training images. The amount is smaller than in most of our experiments with clipart images. We test the model on the rest of the data. See Figure 5.9. The results are very good in images similar to the training data (from the same scene of the episode). Considering images from scenes that do not appear int the training, the results are less satisfactory. However, the network still recognizes the characters.

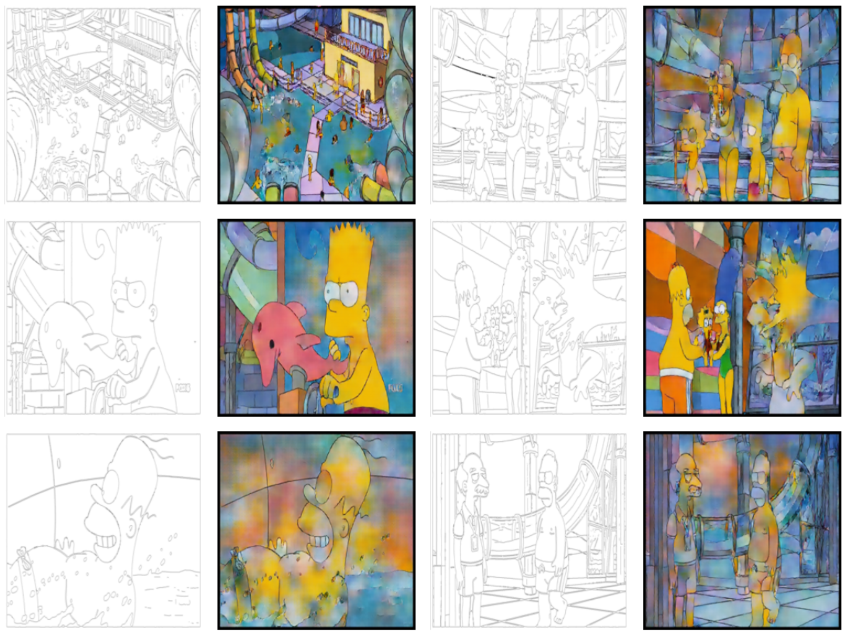


Figure 5.9: Cartoon data set [Brooks, 2016]. Network trained on selected images from the whole episode. The top four colored samples are from the scenes similar to the training ones. Bottom two are more distant in content, therefore the result is blurry and uncertain. The network blends the colors, but they are comparable with the previous experiment.

Chapter 6

Conclusion

Working with Generative adversarial networks can be a challenging task. They are sensitive to parameters, and finding the right combination is not easy. The networks have to learn roughly at the same speed. Otherwise one is going to overpower the other. Even if we have the right parameters, there is no guarantee that the training will converge. Nevertheless, we can still obtain satisfactory results.

The evaluation of GANs models is not simple either. It can be very difficult to estimate the likelihood. Excellent results can be generated by models with poor likelihood, and poor results by models with very good likelihood. A reliable method is to inspect the results and decide if they are acceptable.

The goal of the thesis was to explore and test the abilities of the cGAN model in the domain of sketches and their colorization. Investigation of various data sets and design factors of the networks led to interesting results.

In Chapter 4, we discussed the characteristics of the data. Incorporating the knowledge about the test data (elementary hand-drawn sketches), we found that working with a single object category with clean training data set works the best. We can obtain reasonable results even with a small set in size. Confronted with more cluttered data when the difference between training and test data is more significant, the method fails to produce good results. We also tested a cartoon data set with many objects in the images. The model was able to recognize the repetitive characters but nothing more. Finally, all the results can be further improved by playing with the network design and parameters.

Appendix A

Acronyms

GAN	Generative Adversarial Network
cGAN	Conditional GAN
ML	Maximum Likelihood
CNN	Convolutional Neural Network
SGD	Stochastic Gradient Descent
TV	Total Variation

Appendix B

Contents of the DVD

```
/
├── src
│   ├── edge .................................. code for edge detection
│   └── gan
│       ├── data ............................ code for creating datasets
│       ├── datasets
│       ├── checkpoints ......... setups for experiments, trained models
│       ├── models
│       ├── options
│       ├── results ...................... results used in experiments
│       ├── scripts
│       ├── util
│       ├── test.py
│       └── train.py
└── thesis ......................... pdf and LaTeXcodes of this thesis
    ├── fig
    └── chapters
```

Appendix C

Bibliography

[Arora et al., 2016] Arora, R., Basu, A., Mianjy, P., and Mukherjee, A. (2016). Understanding deep neural networks with rectified linear units. *CoRR*, abs/1611.01491.

[Bau et al., 2018] Bau, D., Zhu, J., Strobelt, H., Zhou, B., Tenenbaum, J. B., Freeman, W. T., and Torralba, A. (2018). GAN dissection: Visualizing and understanding generative adversarial networks. *CoRR*, abs/1811.10597.

[Bengio et al., 2013] Bengio, Y., Thibodeau-Laufer, É., Alain, G., and Yosinski, J. (2013). Deep generative stochastic networks trainable by backprop. *arXiv:1306.1091*.

[Brooks, 2016] Brooks, J. L., G. M. . S. S. E. P. (2016). "lisa the veterinarian." the simpsons. FOX, WRAZ, Raleigh. March. 6, 2016. Television.

[Dollár and Zitnick, 2014] Dollár, P. and Zitnick, C. L. (2014). Fast edge detection using structured forests. *ArXiv*.

[E. Fahlman et al., 1983] E. Fahlman, S., E. Hinton, G., and Sejnowski, T. (1983). Massively parallel architectures for ai: Netl, thistle, and boltzmann machines. pages 109–113.

[Goodfellow, 2017] Goodfellow, I. J. (2017). NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

[Isola et al., 2017] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In

Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on.

[Jin et al., 2017] Jin, Y., Zhang, J., Li, M., Tian, Y., Zhu, H., and Fang, Z. (2017). Towards the automatic anime characters creation with generative adversarial networks. *CoRR*, abs/1708.05509.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *arXiv:1312.6114*.

[Laffont et al., 2014] Laffont, P.-Y., Ren, Z., Tao, X., Qian, C., and Hays, J. (2014). Transient attributes for high-level understanding and editing of outdoor scenes. *ACM Transactions on Graphics (proceedings of SIGGRAPH)*, 33(4).

[Laroca et al., 2018] Laroca, R., Severo, E., Zanlorensi, L. A., Oliveira, L. S., Gonçalves, G. R., Schwartz, W. R., and Menotti, D. (2018). A robust real-time automatic license plate recognition based on the YOLO detector. *CoRR*, abs/1802.09567.

[Ma et al., 2017] Ma, L., Jia, X., Sun, Q., Schiele, B., Tuytelaars, T., and Gool, L. V. (2017). Pose guided person image generation. *CoRR*, abs/1705.09368.

[Perez and Wang, 2017] Perez, L. and Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621.

[Ponti et al., 2017] Ponti, M., Ribeiro, L., Santana Nazare, T., Bui, T., and Collomosse, J. (2017). Everything you wanted to know about deep learning for computer vision but were afraid to ask. pages 17–41.

[Radford et al., 2015] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1511.06434.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597.

[Sangkloy et al., 2016] Sangkloy, P., Lu, J., Fang, C., Yu, F., and Hays, J. (2016). Scribbler: Controlling deep image synthesis with sketch and color. *CoRR*, abs/1612.00835.

[Ulyanov et al., 2016] Ulyanov, D., Vedaldi, A., and Lempitsky, V. S. (2016). Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022.

[Upchurch et al., 2016] Upchurch, P., Gardner, J. R., Bala, K., Pless, R., Snavely, N., and Weinberger, K. Q. (2016). Deep feature interpolation for image content changes. *CoRR*, abs/1611.05507.

[Wang and Deng, 2018] Wang, M. and Deng, W. (2018). Deep face recognition: A survey. *CoRR*, abs/1804.06655.

[Wilber et al., 2017] Wilber, M. J., Fang, C., Jin, H., Hertzmann, A., Collomosse, J., and Belongie, S. J. (2017). Bam! the behance artistic media dataset for recognition beyond photography. *CoRR*, abs/1704.08614.

[Zhang et al., 2016] Zhang, R., Isola, P., and Efros, A. A. (2016). Colorful image colorization. *CoRR*, abs/1603.08511.

[Zhu et al., 2017] Zhu, J., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593.