NTNU
Norwegian University of
Science and Technology

Guiding the training of Generative Adversarial Networks

Simen Selseng

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Björn Gambäck, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Image generation with Artificial Neural Networks (ANNs) has been popular in the last few years by using the knowledge about objects and structures to generate dreamlike images, applying smart filters or making images appear scary. Although vivid and impressive, the generated images do not appear natural to a human viewer. Training ANNs to generate better images is a hard problem to solve with conventional machine learning techniques.

Generative Adversarial Networks (GAN) is a new approach that utilizes two ANNs in one system. The first ANN is called the Discriminator and evaluates the quality of generated images from the other neural network called the Generator. These two networks partake in an adversarial battle to improve their own performance. The results of this battle is a model capable of generating new natural looking images based on unsupervised learning of an image dataset. GAN have evolved considerably since its inception, but there are still open problems.

One such problem is how to objectively evaluate a GAN model. Image generation has no fixed solution, and evaluating it is a difficult as evaluating other creative works. One method allows for two GAN models to be compared by evaluating each other. This method leverage the adversarial min-max aspect of GANs, by having the Discriminator from one model classify generated samples from the other, and vice versa. This thesis will propose a custom implementation of this method that allows for comparison between two GAN models both overall and throughout the learning process.

GANs have a reputation for being hard to train. One concrete problem is maintaining the balance between the Generator and Discriminator. As for humans, it is easier to rate the quality of images then it is to actually create them. A good evaluator is necessary, but it must not out power the generative model. Two main approaches will be explored to achieve a better balanced GAN model. The first method makes guided alterations to the usually random input of the Generator. The second method adds an additional Discriminator to the model. Techniques based on both of these methods were shown to effectively guide the training process and creating strong models that outperformed regular GANs when compared with the previously mentioned evaluation metric.

Sammendrag

Bruk av Kunstige Nevrale Nettverk (ANN) til kreative oppgaver som bildegenerering har vært populært de siste årene. Nevrale nettverk kan bli trent på bildedata til å lære seg repeterende mønstre og former som igjen kan brukes til å generere nye bilder. Selv om disse bildene kan være både fargerike og slående, er det sjeldent noe tvil om at de er genererte eller i det minste behandlet av en datamaskin. Nevrale nettverk har hovedsakelig blitt trent med data som inneholder et korrekt svar for hvert datapunkt. Dette fungerer bra for oppgaver som gjenkjenning av ansikter eller objekter i et bilde, men er svært vanskelig for kreative oppgaver uten en klar fasit.

En ny teknikk innen nevrale nettverk kalt Generative Motstående Nettverk (GAN) bruker et separat nettverk til å sammenligne genererte bilder med ekte bilder og avgjør hvor naturlige de genererte bildene ser ut. Dette krever altså to nettverk: Generatoren, som generere bilder, og Dommeren, som vurderer de genererte bildene. Generatoren prøver å lure Dommeren til å tro de genererte bildene er ekte. Dommeren prøver å ikke bli lurt. Hvert nettverk prøver å forbedre sine egne egenskaper på bekostning av den andre. Denne kampen frem og tilbake kan trene Generatoren til å kan skape nye, realistiske eksempler basert på et eksisterende sett med bilder.

GAN er en relativt ny teknikk og det er fortsatt åpne problemer. Et av disse problemene er hvordan man objektivt kan bedømme kvaliteten til de genererte bildene. Bildegenerering har ingen fasit og kan sammenlignes med bedømmelse av kunst og andre kreative verk. Mange metoder er foreslått, men det fortsatt ingen klar løsning. En metode går ut på å sammenligne to trente GMN-modeller ved å la de bedømme hverandre. Denne avhandlingen vil implementere og videreutvikle denne teknikken for objektiv evaluering av det endelige resultatet sågar delresultater underveis i læringsprosessen.

GAN har pådratt seg et rykte for å være vanskelig å trene. Et konkret problem er at det blir en ubalanse i egenskapene til et av nettverkene, oftest Dommeren. I likhet med mennesker, er det oftest lettere å bedømme noe, enn å faktisk produsere det selv. Likevel er en god Dommer nødvendig for å kunne gjengi treningsbildene, så lenge den ikke overkjører Generatoren. Denne avhandlingen vil se nærmere på to metoder som gjør jobben lettere for Generatoren uten å begrense Dommeren. Generatoren baserer de genererte bildene på en liste med tilfeldige tall. Den første metoden vil gjøre endringer av disse tallene for å styre hvordan modellen trenes. Den andre teknikken vil utvide GAN med en ny Dommer og dermed hindre at en blir for sterk. Konkrete teknikker basert på disse metodene ble testet gjennom evalueringsmetoden beskrevet tidligere og begge resulterte i bedre modeller som krevde mindre treningstid og var bedre balansert.

Preface

This Master's Thesis is part of a Master's Degree in Computer Science at the Norwegian University of Technology (NTNU) in Trondheim.

This report is written for the Artificial Intelligence group and conducted with supervision from Professor Björn Gambäck at the Department of Computer Science.

The original task description was given by Professor Björn Gambäck and was an open-ended project within Computational Creativity and Art. A pre-study, *TDT4506 Computer Science specialization Course*, was conducted in the Fall 2016 with the goal researching state-of-the-art of a subject related to Artificial Intelligence. This project, *TDT4900 Computer Science Master's Thesis*, is a continuation of the pre-study.

Simen Selseng
Trondheim, 11th June 2017

Acknowledgment

I would like to thank my supervisor Professor Björn Gambäck the helpful guidance and assistance since our very first meeting and allowing me work on such an open project.

I thank Magnus S. Gundersen and Simon B. Johasen for every joyful break throughout this semester. Likewise, I thank Andreas Molund for keeping me companied at the study hall and for maintaining our mutual coffee-brewing deal.

I would also express gratitude to Ian J. Goodfellow and his team as well as Alec Radford, Luke Metz, Soumith Chintala for allowing the use of their images in this thesis.

Contents

List of Figures

List of Tables

1. Introduction

This chapter gives an introduction to the thesis. The motivation is the first part of this chapter. The overall research goal of the project is then defined with three research questions proposed based on it. The contributions and thesis structure is covered next.

1.1. Motivation

Generative Adversarial networks (GANs) is a new technique within Deep Learning better suited for machine learning tasks without a fixed solution. GAN research has rapidly increased in the last year as it has many appealing aspects that are hard or not possible with existing techniques, such unsupervised learning and image generation. Yann LeCun, Director of AI Research at Facebook and Professor, stated in 2016 that GAN was the advancement within Deep Learning that he was the most excited about (LeCun and Bennett, 2016). A prominent use case, and the focus of this thesis, is the way GANs can be trained to generate new examples of real life images. New techniques will be explored to further specialize and improve GANs for the task of image generation.

Using computers to generate images, such as a GAN system, falls under the subject of *Computational creativity*. Defining what is creative is difficult as it is highly subjective. For something to be classified as creative, it should be surprising, novel and give some form of value to the observer (Boden, 1998). Humans may have no trouble calling the drawing of a child creative, but have a stronger bias against a creative computer system.

A GAN model consists of multiple Artificial Neural Networks (ANNs), a machine learning technique inspired by the human brain. Using ANNs for computational creative tasks has been prominent within the last few years with Mordvintsev et al. (2015) demonstrating how Google's Inception network, normally used to classify objects in images, could use its deep knowledge of shapes and patterns to generate vivid imagery (Figure 1.1), to interactive web based tools, games and demos including *Nightmare Machine*[1] which transforms uploaded images into

[1]http://nightmare.mit.edu/

dark and scary versions, and to *Quick, Draw*[2] that tasks the users with drawing certain objects while the artificial intelligence tries to guess which objects are drawn. Many of these go viral and reach mainstream media.

The common approach to ANNs is to use a single network and train it with labelled training data. This is called supervised learning as the model is explicitly told what output is correct for a given input. A drawback to supervised learning is that it requires data with labeled output, often created by human labour. Neural networks require thousands or even millions of elements for training, and supervised learning requires human labelling of each element. These labels may not always be available or even feasible. Unsupervised learning, on the other hand, is training without labels, which forces the learner to make its own generalizations about the data.

Generating images can be done using tools like Google's *Deep Dream Generator*[3] and the previously mentioned *Nightmare Machine*. The generated images are impressive; however, they look quite unnatural and artificial. This is not surprising as the networks are usually trained to maximise object recognition. Using a statistical model to learn shapes and patterns in images lets the network get a deep structural understanding. An example of how Google's Inception network (Mordvintsev et al., 2015) sees objects in images is depicted in Figure 1.2.
Training a network to generate more natural looking images is certainly possible, but would be a hard problem to solve with conventional supervised learning, requiring either a human to evaluate every generated image or creating a large dataset with examples of good and bad images. *Nightmare Machine* uses a version of the former, allowing online users to rate generated samples as either *scary* or *not scary*.

The innovative approach with GANs is to replace the human evaluator with an additional neural network. The concept of GAN was introduced by Goodfellow et al. (2014) and utilizes two ANNs in a min-max game. The first network, the generative model, is tasked with generating new samples, often images, based on an existing set of samples. The second network, the discriminative model, evaluates and judges whether the sample is from the dataset or synthesized by the other model. The two networks partake in an *adversarial zero sum min-max battle* to improve their own performance. In other words, the Discriminators ability to separate real samples from the ones generated by the Generator. Meanwhile, the Generator will generate images of improved quality to deceive the Discriminator. Goodfellow et al. (2014) have the following causal description of GAN:

"The generative model can be thought of as analogous to a team of counterfeiters,

[2] Google Quick Draw https://quickdraw.withgoogle.com/
[3]https://deepdreamGenerator.com/

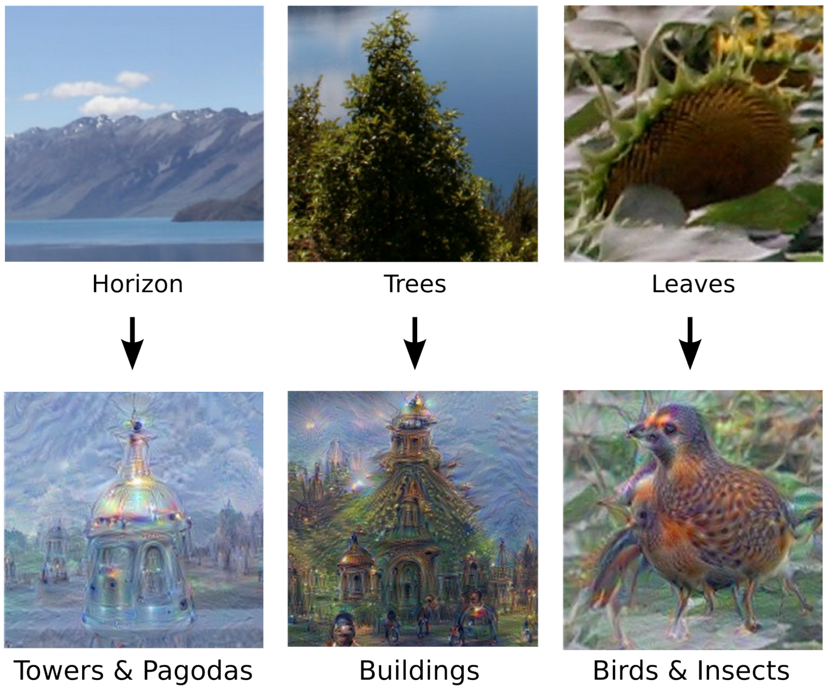

Figure 1.1.: Mordvintsev et al. (2015) demonstrated how the Inception ANN can be used to detect features in everyday images. The result is vivid and dream-like imagery. (Source: Mordvintsev et al., 2015. Used under Creative Commons Attribution 4.0 International License.)

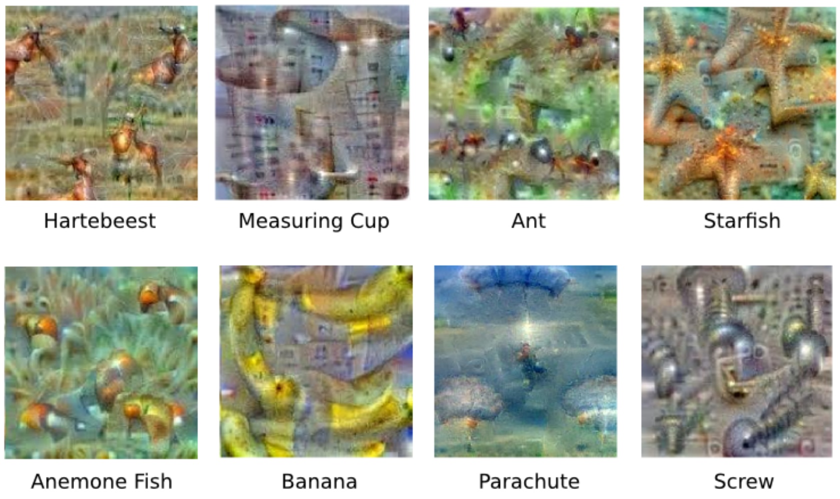

Figure 1.2.: Random noise can be used to visualize how a neural network recognizes objects. Examples generated by Mordvintsev et al. of various objects using Google's Inception network. (Source: Mordvintsev et al., 2015. Used under Creative Commons Attribution 4.0 International License.)

trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles." (Goodfellow et al., 2014, p. 1)

GANs have evolved considerably since their inception. Architectural changes, new techniques and more suitable dataset have specialised GANs towards image generation and greatly increased the quality of the generated samples. Figure 1.3 displays examples of randomly generated faces generated by a modern GAN (Radford et al., 2015). While many of the images are hard to distinguish from real images, some far from perfect with unnatural features and shapes.

GANs have a reputation as being hard to train. The performance of the Discriminator model should be kept in balance of the Generator; otherwise image generation may not improve. Also, as with all computational creativity, it is hard to quantitatively evaluate the quality of a generated sample. GAN is a relatively new framework so problems are expected. This is what makes GAN such an interesting area of research as new improvements are made continuously. This thesis will research state-of-the-art and explore new techniques to help addressing the mentioned problems with the overall goal of improving image generation.

1.2. Goal and Research Questions

G1 *Explore and develop new techniques for Generative Adversarial Networks specialized for image generation*

The GAN framework is suitable for several kinds of machine learning problems. The largest area of research has so far been with using GANs to generate images based on unsupervised learning. This thesis will focus solely on using GANs for image generation and further specializes GANs towards this task by developing new techniques. Experiments will be conducted to document how these new techniques influence and affect the quality of the generated images.

RQ1 *How is performance affected by altering the input to be more favorable towards the Generator model in a Generative Adversarial Network?*

A problem with GAN is the problem of synchronizing the Generator and Discriminator during training. The generative model is generally more difficult to train then a Discriminator model. As a result of this, the Discriminators performance is improving at a faster pace than the Generator which makes the Generator too skilled at classifying the Generators generated samples. A suggested way of altering this balance is modifying the otherwise random input to be more favorable towards the Generator.

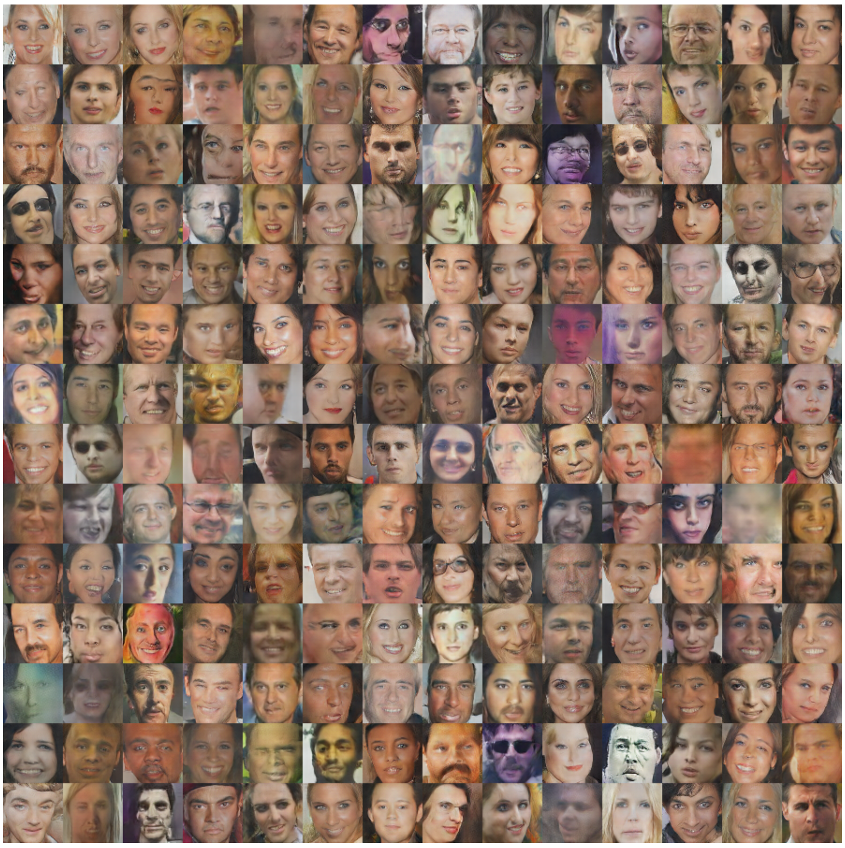

Figure 1.3.: Randomly selected samples generated with the DCGAN architecture which specialized GAN towards image generation. The networks are trained a dataset of celebrity faces. (Source: Radford et al., 2015. With permission.)

RQ2 *In what way is performance of a Generative Adversarial Network model impacted by adding additional, asymmetrically trained Discriminator models?*

The fundamental difference of GAN compared to usual ANN is the use of two networks in the same model. Adding additional neural networks is therefore a logical next step and a growing area of research. Although there are many ways of expanding the number of networks in a GAN, this thesis will focus on adding additional Discriminators that are trained on different data .

RQ3 *How can the performance of the suggested techniques related to RQ1 and RQ2 be accurately measured and objectively evaluated?*

Another problem with GANs is the difficulty of objectively evaluating the quality of a trained model. This problem makes it hard to accurately measure the effect of new techniques. The most obvious way, when dealing with image generation, is to manually inspect the quality of the generated samples. This is possible to some degree if the comparison is done between systems of widely different quality, but is often too subjective if comparing similar models. A major area of research with GANs is finding a metric that is able to objectively and universally evaluate the quality of a model. This thesis will explore methods to evaluate the techniques derived from RQ1 and RQ2.

1.3. Contributions

C1 Literary review of the current state-of-the-art of for GANs specialized for image generation.

C2 Evaluation and discussion of various public and custom image datasets in relation to unsupervised learning with GANs.

C3 SGAM: Custom memory efficient implementation of GAM for comparison of two GAN models throughout the training process.

C4 New techniques for altering the Generative models' input to give it at an edge during training: SRN, IBNG and ABNS.

C5 GMAN-HD: Extends GAN with additional Discriminator past output from the Generator.

1.4. Thesis structure

- **Chapter 1** introduces the thesis and the project conducted as a part of it.
- **Chapter 2** gives a brief overview of various smaller subjects and theory that will be discussed throughout this thesis. The main part of the background chapter is introduction to the main subject of this thesis, Generative Adversarial Networks.
- **Chapter 3** offers a literary review of the state-of-the-art for GANs specialized for image generation.
- **Chapter 4** details the system implementation and the hardware it runs on.
- **Chapter 5** describes the dataset and sources for data that may be used for the experiments.
- **Chapter 6** discuss the research question in regards to the related work. From this discussion, custom techniques proposed to answer the research questions.
- **Chapter 7** presents the results of experiments conducted to the techniques proposed in *Chapter 6*.
- **Chapter 8**, concludes the thesis and discusses the project in relation to the project goal and research questions defined in *Chapter 1*.

2. Background

This chapter will give a introduction to the necessary theory, technologies and datasets that will be used or discussed throughout this thesis. Section 2.6 describes Generative Adversarial Networks (GANs), the main subject of this thesis.

2.1. Artificial neural networks

Artificial Neural Networks (ANNs) is a field within Artificial Intelligence (AI) that takes inspiration from the human brain. A network consists of at least two nodes, input and output, with weights between them. These weights are adjusted to give the desired output when given an input. This is called training the network.

A Convolutional Neural Network (CNN) is form of ANN that utilizes at least one convolutional layer in its architecture. Convolution being a kind of linear operation replacing general matrix multiplication. CNNs are mostly used in conjunction with images as regular ANNs usually become too large to efficiently train. CNNs commonly implements features such as pooling layers for image down sampling and sparse connectivity to reduce the overall size of the network, decreasing training time.

2.2. Parallelization and GPU-based calculations

Parallel computing is the concept of dividing a computational task into smaller subtasks, using multiple processors to calculate each subtask simultaneously. This is normally done on the central processing unit (CPU). Certain problems can utilize the parallel nature of the graphics processing unit (GPU). This can boost performance considerably as modern GPUs have thousands of cores compared to CPUs having 2 to 8 cores. Neural networks are highly susceptible to parallel programming as the training phase consists of updating thousands or millions of weights independently of each other.

2.3. Frameworks

The software framework used or considered used for this thesis are described in this section.

2.3.1. CUDA

CUDA[1] is a platform for parallel computations using a modern GPU. The framework enables developers to send code directly to the GPU. CUDA is a product of NVIDIA and requires a GPU from them with specialized hardware known a CUDA-cores. CUDA-cores are highly parallelized processors and the number of cores ranges from about 500 on the lower end cards to several thousand on the high-end ones. Machine learning frameworks can utilize CUDA through a framework called cuDNN[2], a library of low-level operations, implementing many common features of ANN that help boost performance of deep neural networks using GPU-based calculations. The framework is freely available, but a developer account at NVIDIA is required.

2.3.2. Deep learning libraries

There exists many frameworks and libraries for deep learning. The most used are all quite similar in that they are open-source, usually use Python to run underlying C/C++ implementations for efficient calculations in parallel using either CPU or GPU (through CUDA).

- **TensorFlow[3]** is a popular open source library for deep learning. It is developed by Google Brain, a research project in deep learning at Google. TensorFlow runs on 64-bit versions of MacOS and Linux, servers and mobile devices. TensorFlow includes native support for the most common features of ANNs including activations functions and loss functions. Other core features include tools for visualization and ability to utilize multiple GPUs and CPUs automatically for parallel computations. GPU-calculations utilize CUDA with cuDNN. TensorFlow can be programmed using Python, C and C++, with Python being the best documented and most used of the three.

- **Theano[4]** is another popular Python library for deep learning. Developed at Université de Montréal to efficiently calculate large mathematical expressions like multi-dimensional arrays. It supports Linux, MacOS and

[1]https://www.nvidia.com/object/cuda_home_new.html
[2]https://developer.nvidia.com/cudnn
[3]https://www.tensorflow.org/
[4]http://deeplearning.net/software/theano/index.html

Name	Size	Annotations	Resolution
CelebA	202,599	Name, description	Variable
CelebA Aligned	202,599	Name, description	178x218
CAT	10,000	Facial features	Variable
CIFAR-10	60,000	Objects	32x32
Flickr30k	31,783	Objects	Variable
MNIST	60,000	Digit	28x28

Table 2.1.: Dataset summary

Windows. As with TensorFlow, Theano is programmed using Python and supports CUDA.

- **Torch**[5] is deep learning framework with focus on GPU-calculations. Torch utilises CUDA and is programmed with the language LuaJIT. Torch is embeddable to mobile and maintained by scientists and engineers at Facebook, Twitter and Google DeepMind.
- **Keras**[6] is a high-level library that runs on top of either TensorFlow or Theano. Keras makes modelling neural networks easier and more accessible.

2.4. Data

Machine learning is dependent on the dataset used for training. Assembling datasets of high quality is expensive and time consuming. Using pre-existing datasets created by others is therefore often preferred, but may not always be possible. Section 2.4.1 introduced pre-existing datasets while Section 2.4.2 covers potential sources for image data.

2.4.1. Public datasets

Public datasets are pre-existing datasets created by others. Figure 2.1 summarizes the mentioned datasets and its content. Chapter 5 will discuss the datasets introduces in this section in further detail.

- **CelebA** (Large-scale Celeb Faces Attributes) (Liu et al., 2015) Dataset is a collection of 202,599 facial images of 11,117 celebrities. The size of the datasets results in multiple poses as well a large set of backgrounds.

[5]http://torch.ch/
[6]https://keras.io/

Each image is annotated with relevant attributes as well as the name of the depicted person. CelebA is available in two editions; one with images cropped and aligned and the other with original, *in-the-wild* images. Both versions are relevant for this project. CelebA has become quite popular for the use with GAN after Radford et al. used it with its DCGAN architecture (Section: 3.1).

- **MNIST** (Mixed National Institute of Standards and Technology database) (LeCun et al., 1998) has become a classic dataset for machine learning. MNIST is a database of 70.000 labelled examples of handwritten digits. Each image is of size 28x28 pixels and contains a single digit in the range of 0 to 9. Being a popular dataset, MNIST is well suited for comparison between papers.
- **CAT** (Zhang et al., 2008) The CAT dataset contains 10.000 images of cats. Every images contains the face of a cat visible in frame with annotations of where various facial features are located. Framing and depictions varies between each image
- **CIFAR**-10 (Krizhevsky and Hinton, 2009) has become a popular dataset for use in machine learning. The dataset is a labelled subset of the Tiny Images Dataset (Torralba et al., 2008) and contains 60,000 images at a resolution of 32x32. The resolution is quite low and the content is often hard to make out.
- **Flicrk30k** (Young et al., 2014) is a dataset of 30,000 randomly collected images from the social media site Flickr. All images are captioned with multiple textual descriptions about the content of the image as well as markings for various objects in the image. The images vary widely in depiction as well as physical properties like aspect ratio and resolution.

2.4.2. Data sources

The sources described here can potentially be used to create unique and custom datasets that can be fine-tuned for each task-specific problem.

- **Instagram**[7] is social network owned by Facebook. It lets users upload either images or video to their account. The media content can be tagged with information including location, user specified tags, known as a hashtag, and a description. Users can also like an image and post comments. Instagram is primarily viewed through their mobile application, but all public content is also available through their website. Publishing is only available

[7]https://www.instagram.com/

through the mobile application. Additionally, Instagram offers an Application Program Interface (API) to let developers interact with Instagram's content through their own applications. Initially open for all developers, the API now requires approval for full access to Instagram's content. Accessing the API without approval is allowed, but the content is restricted to a small subset of the developer's own content.

- **Flickr[8]** is web service for hosting images and video, owned by Yahoo!. Like Instagram, images on Flickr are also tagged with relevant information. Flickr offers a public API that enables developers to access most of the features available through Flickr's website[9].

- **Web crawlers** are automated programs that roam the internet and collect data as they crawl along. Web crawlers are therefore not single datasets but rather an overall method to collect data. They can be used to collect data not accessible through official sources such as a public API.

2.5. Supervised and unsupervised learning

Supervised and unsupervised learning are two different approaches to training ANNs. With supervised learning, the datasets are explicitly labelled with the correct output for that particular element. For instance, with MNIST, every handwritten digit is labelled with the correct number. This is applicable to data that can have the output directly mapped to the input. Unsupervised learning, on the other hand, is where the training dataset is unlabelled, meaning the learner is not told the correct output for a given input. This can be any type of data and is used where the task of the computer is to find patterns in the data. ANNs have for the most parts been trained using supervised learning, for instance, learning to recognize elements in images.

[8]https://www.flickr.com/
[9]https://www.flickr.com/services/developer/

2.6. Generative Adversarial Network

Generative Adversarial Networks (GANs) were introduced in Chapter 1 and will be more thoroughly detailed in this section. Section 2.6.1 describes some common problems with GANs. One such problem is how to evaluate the output of GAN which is covered in Section 2.6.2.

Goodfellow et al. (2014) introduced the concept of GAN which consists of two feed-forward neural networks that train on a given dataset. The two networks partake in a *min-max, zero-sum two player game* that enables the networks to learn the data distribution of the training dataset.

- The task of the generative model *G*, often called the *Generator*, is to generate realistic data samples that approximate the training set. The goal is to generate samples that the Discriminator cannot distinguish from real samples from the dataset.
- The of the discriminative model *D*, often called the *Discriminator*, is to rate whether an image is from the training dataset or a generated sample by *G*. The goal is to correctly classify the samples and not get fooled by *G*.

Training of these networks occur simultaneously and can be described as a minimax game. The Discriminator is trying to maximize its own performance of distinguishing between real and generated samples, while the Generator G is maximizing its ability to generate samples that manage to fool the Discriminator. A more formal definition of GAN's *min-max game* can be described with the following function V:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data(x)}}[logD(x)] + \mathbb{E}_{z \sim p_{noise(z)}}[log(1 - D(G(\boldsymbol{z})))] \quad (2.1)$$

- x is a sample from the training dataset which has the probability distribution p_{data}
- z is a randomly generated noise sample from the distribution p_{noise}. This is often a list of random numbers in range -1 to 1. See Figure 2.2 for examples.
- E_x and E_z indicates which data distribution the sample belongs to. The Discriminator must classify that the sample belong to the expected distribution. In other words, E_x denotes the sample is a real image and E_z denotes that the sample is generated.
- *G(z)* represents a sample generated by the *Generator* model *G* with noise z as input.

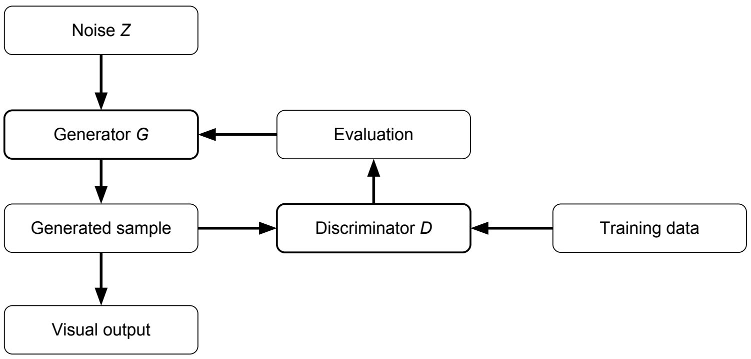

Figure 2.1.: Overview of the GAN framework. GAN consist of two models, Generator *G* and Discriminator *D*. Both of these are CNNs, See Section 2.1. *G* receives uniformly generated noise samples*Z* as input. Examples of such noise samples are shown in Figure 2.2. *Z* propagates through the ANN and results in an image. Model *D* is trained with images from the training set and generated images from *G*. It is taught to rate real images 1 and generated images 0. *G* is trained through feedback from *D*. *G* generates samples that *D* evaluates. The results of this evaluation, which images were able to fool *D*, is used to improve *G*. *G* is thereby never shown images from the training set directly, only through evaluation from *D*.

- *D(x)* is the probability that the sample x is a real sample and not generated by *G*

The strength of GAN lies in its ability to learn an arbitrary data distribution with just unsupervised learning. This is a desirable feature as it does not require costly labelled datasets. Conventional ANNs do generally not perform well with unsupervised learning and GANs are therefore a popular area of research. One use case for unsupervised learning is image generation.

The Discriminator *D*'s task is to rate images on how natural they appear and output a scalar value. Ideally, an image from the training set, a real image, should get a good rating while an image that is generated from *G* should get a poor rating. During training, the Discriminator network is shown different images from both the training set and samples from *G* and taught how it should rate them. This is then used to adjust the weights and improve the Discriminator by

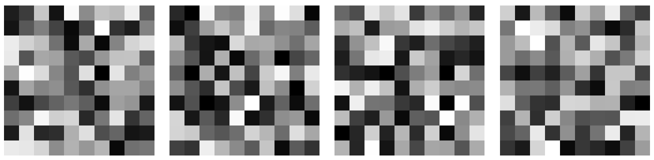
Figure 2.2.: Example of uniformly generated noise used as input to the generative model in a GAN

slowly modifying what features are used to spot fake images. These adjustments are supplied back to the Generator and in turn used to further improve image generation. The Generator G generates its samples given an input Z, which is generally an image consisting of randomly generated noise. The network will try to recognize patterns in the noise and apply elements of previously seen images, resulting in an image that should look natural. This process is shown in Figure 2.1.

Goodfellow et al. (2014) highlight the potential for GAN as an image Generator with samples generated after training on various datasets. Figures 2.3, 2.4 and 2.5 show randomly selected samples completely generated by unsupervised learning with the MNIST dataset (Section 2.4.1), Toronto Face Dataset (TFD) (Susskind et al.) and CIFAR-10 (Section 2.4.1), respectably. The rightmost column, yellow boxes, shows the most similar sample from the training dataset. Comparing the real images to the generated ones shows that GAN is not simply overfitting the Generator to a single image, but rather matching overall features in the training data. This is clearly visible by comparing the two last columns in Figure 2.4 and somewhat apparent in Figure 2.3 in the slight variations in the shape of the digits.

2.6.1. General problems with GAN

Some of the more general unsolved problems with GANs will be presented in this section and used throughout the rest of this thesis. With GANs being a relatively new framework, there are still open problems to be solved.

Model balancing and synchronization

A problem with GANs is keeping the performance of the Discriminator and Generator models balanced during training. The Generator is generally harder to efficiently train than the Discriminator. This can result in a situation where the Discriminator eventually outpace the Generator. The Discriminator will in such cases become too skilled at classifying the generated samples, which halts the

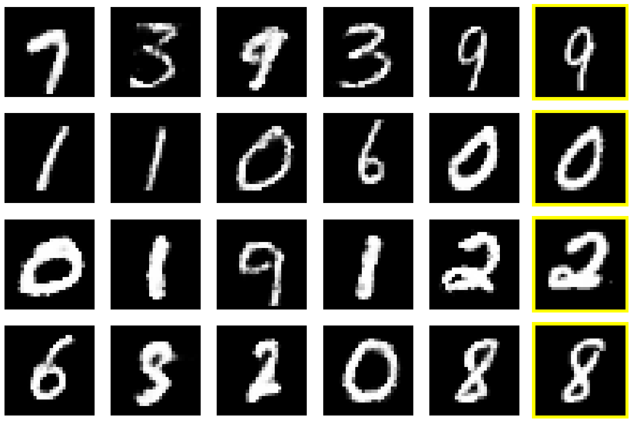

Figure 2.3.: Randomly selected samples generated by Goodfellow et al. using the GAN framework. GAN trained with unsupervised learning on the MNIST dataset (Section 2.4.1). The rightmost column, yellow boxes, shows the most similar sample from the training dataset. (Source: Goodfellow et al., 2014. With permission.)

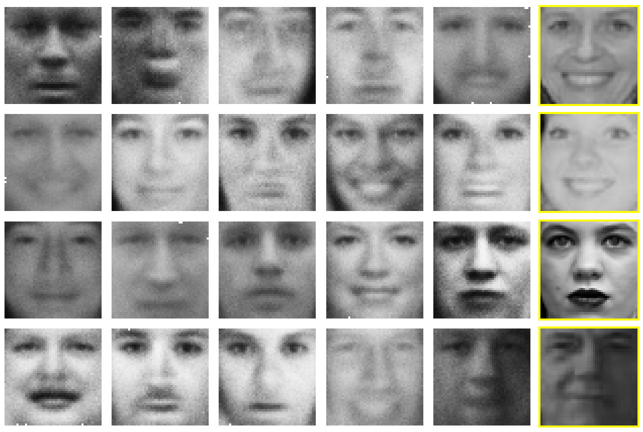

Figure 2.4.: Randomly selected samples generated by GAN framework after unsupervised training on the Toronto Face Dataset (Goodfellow et al., 2014). GAN is able to learn the general concept of a human face and generate new examples. The rightmost column, yellow boxes, shows the most similar sample from the training dataset. (Source: Goodfellow et al., 2014. With permission.)

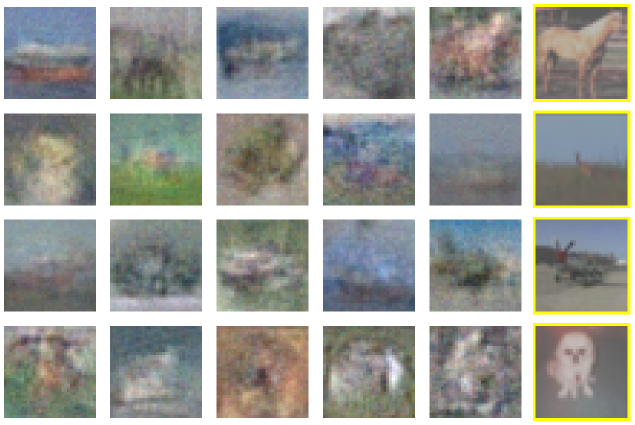

Figure 2.5.: Random samples generated from the CIFAR-10 dataset with GAN (Goodfellow et al., 2014). The rightmost column, yellow boxes, shows the most similar sample from the training dataset. (Source: Goodfellow et al., 2014. With permission.)

progression of the Generator. The main method of addressing this problem has been to artificially limit the performance of the Discriminator. Goodfellow (2017) is critical to this approach as a well-trained Discriminator is required to accurately describe the probability density of the training data. Goodfellow (2017) still admits this as a problem, but suggests other solutions should be applied instead.

Non-convergence and Mode collapse

GANs will seldom reach a point of convergence in the learning process. The Generator and Discriminator will continue to increase their respective performance until the learning is terminated. This causes the generated samples to drift after most of major learning has ceased. As this point, the quality of the images will not improve, but drift through various alterations such as changing colour or reintroducing past mistakes. This is called *non-convergence*, as defined in Goodfellow (2017). *Mode collapse* is a problem caused by non-converge that occurs when the Generator maps multiple inputs to the same output. The result of this is generated images which share many of the same features and thereby look similar.

Mode coverage

GAN trained through unsupervised learning has the potential of not capturing the whole distribution of the training data. Im et al. (2016b) defines *mode coverage* as this problem of not fully assigning probabilities to all distributions in the training data. In other words, the Discriminator learns only a few of the many depictions of an image dataset and thereby not correctly capturing the whole dataset. A well performing Generator will not to alter its image generation as long as it successfully fools the. The result of this is generated images that are quite similar and only samples from a smaller part of the dataset.

Evaluation

GAN have no obvious method of quantitatively evaluating the quality of the output. Assessing the quality of a generated image is hard, as there is neither a fixed answer nor a Boolean score. A generated image will seldom be completely unnatural nor natural, but have areas of both. Assessing the quality is also highly subjective. Section 2.6.2 will discuss the problem further by detailing some suggested evaluation metrics.

2.6.2. Evaluation metrics

Section 2.6.1 briefly introduced the problem of evaluating generative models such as GANs. Evaluating GANs is still considered as an open problem, but there are some problem-specific solutions described in this section. Theis et al. (2016) discussed some the more popular evaluation metrics and concluded that both the training and evaluation of a GAN should be modelled after the target application.

Log-likelihood

Using probabilistic based likelihood was originally proposed by Goodfellow et al. (2014) to compare the learned probability function of the training data. Goodfellow et al. did warn, however, that it was not suitable for data of higher dimensions, such as images. A GAN model can generate samples of high quality with poor likelihood and vice versa.

Nearest Neighbours and Image retrieval

The term Nearest Neighbour(NN) are a common definition for methods of calculating the distance between two samples. Examples of NN algorithms include Euclidean and Manhattan distance. NN can be used to evaluate GANs by comparing the pixel data of generated samples to the training datasets.

Durugkar et al. (2016a) creates a larger evaluation system where a separate CNN finds the generated images that are most similar to images in the training dataset. Two variations of NN are then used as an evaluation metric between the real image and the retrieved images. Ledig et al. (2016) use GAN to upscale the resolution of images and use nearest neighbour as one of the measures of comparing the result to the training data. Theis et al. (2016) argues against using NN as an evaluation metric as even a small shift in the images can drastically change the distance.

Visual Turing Test

The Visual Turing Test is an adaptation of the famous Turing test for human computer interactions. The regular Turing test is passed if a human is unaware he is communicating with a computer. The visual Turing test is similarly passed if a computer generated image is believed to be a real life image.

Salimans et al. (2016) used human annotators to assess the quality of their GAN. Users were shown images either generated or from the training set, and tasked with evaluating the images as either real or fake. Salimans et al. found this approach of evaluation less than desirable. Result were depended on the annotator and their motivation. Feedback on their own performance, such as which images were correctly classified or certain patterns to look for, made the annotators more successful. Experimentation with image generation using the CIFAR-10 dataset saw annotators correctly categorizing 78.7% of the images correctly as either real or fake. However, Salimans et al. themselves were able to get over 95% accuracy as they were more familiar with CIFAR-10 and GAN.

Object classification

Regular ANNs, and specifically CNNs, have become quite good at classifying objects in an image. These networks are trained on a large set of labelled images. When shown an image, the network will output which objects it believes is in the picture and a percentage on the degree of confidence. These networks can be used to evaluate the quality of the generated images from GAN.

Salimans et al. (2016) create an evaluation system using the pre-trained Inception (Szegedy et al., 2016) model to classify objects in every generated sample. Inception is a large CNN trained to recognize certain objects and output what degree of confidence it has about the object. Salimans et al. (2016) exploits this by assuming generated images with clear objects are better. Secondly, the model as a whole is evaluated by the number of different objects recognized. Variation in the generated samples are preferred. Salimans et al. (2016) claim this approach

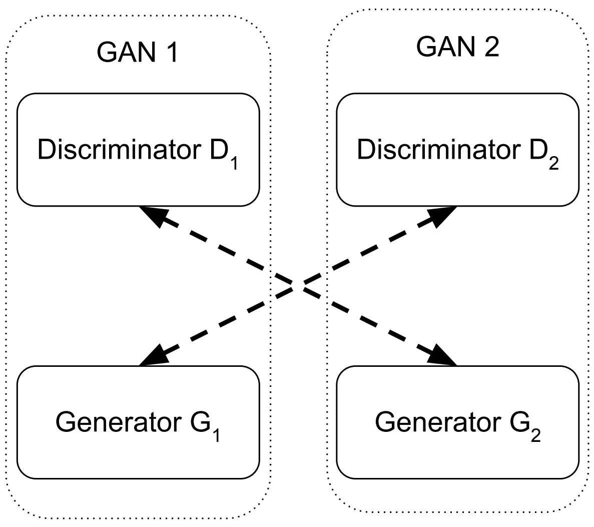

Figure 2.6.: Generative Adversarial Metric (GAM) method for comparing two GANs (Im et al., 2016a). The Generator/discirmantor pair are exchanged between the two GANs. The better GAN is able to both generated samples that decive the other while not being deceived itself.

correlates well with their experimentation on human annotator as evaluations. This approach does require a labelled dataset to train the evaluator network.

Generative Adversarial Metric

Generative Adversarial Metric (GAM) (Im et al., 2016a) is a method for comparing GANs. GAM leverages the adversarial min-max aspect of GANs, by having the Discriminator from one model classify generated samples from the other, and vice versa. , Figure 2.6. The GAN with the least classification error may be the best model. Im et al. (2016a) argue that the two Discriminator should also be verified against actual training data to ensure that one model is not overfitted more than the other. In other words, the two Discriminator must perform about equal on the test dataset for GAM to elect the better GAN. Equation 2.2 and Equation 2.3 show, respectivly, how the ratio of classification error for generated samples and test data are calculated. The value of these are then used in Equation 2.4 to either select a winning GAN or a tie, depending on Equation 2.3.

Im et al. (2016b) further expands GAM to work with their GAP system of multiple Generator/Discriminator pairs, Section 3.5. Improvements in generalization resulted in the test ratio of Equation 2.3 failing when compared to other GANs. To solve this, Im et al. (2016b) use either average error rate or select the worst one across all the Discriminator.

Durugkar et al. (2016b) also created a variation of GAM to evaluate their GMAN system consisting of multiple Discriminator, see Section 3.4.

$$r_{sample} = \frac{\epsilon(D_1(G_2(z)))}{\epsilon(D_2(G_1(z)))} \tag{2.2}$$

$$r_{test} = \frac{\epsilon(D_1(x))}{\epsilon(D_2(x))} \tag{2.3}$$

$$winner = \begin{cases} \text{GAN 1 if } r_{sample} < 1 \text{ and } r_{test} \simeq 1 \\ \text{GAN 2 if } r_{sample} > 1 \text{ and } r_{test} \simeq 1 \\ \text{Tie otherwise} \end{cases} \tag{2.4}$$

- x is a sample from the training dataset which has the probability distribution p_{data}
- z is a randomly generated noise sample from the distribution p_{noise}.
- *G_y(z)* represents a sample generated with noise z by *Generator* in GAN model *y*. $y = \{1, 2\}$
- *D_y(x)* is the probability that the sample x is a real sample and not generated
- ϵ is the error rate of the Discriminator in the classification of images as real or generated

3. Related work

A literary study of the current state-of-the-art for Generative Adversarial Networks (GAN) related to image generation was conducted for this thesis and is covered in this chapter.

3.1. Deep Convolutional Generative Adversarial Networks

Deep Convolutional Generative Adversarial Networks (DCGAN) (Radford et al., 2015) is an architecture that specializes GANs for image generation by applying advances from Convolutional Neural Networks.

DCGAN also employs various other techniques that greatly improve the overall quality of the generated images. Figure 3.1 shows a comparison of the generated samples from GAN and DCGAN compared to the original dataset. The samples generated with DCGAN are both crisper and smoother than then the ones generated by regular GAN.

DCGAN allows for unsupervised learning on images of higher resolution than regular GAN. Radford et al. (2015) trained their network on a dataset containing over three million images of bedrooms. The network was able to learn the general patterns and from that generate new images that were not in the original dataset. The result after five epochs, five full iterations through the training data, is shown in Figure 3.2. Although low resolution, the images are distinctly looking like bedrooms.

The third dataset Radford et al. used to train DCGAN was the CelebA dataset (Section 2.4.1). Compared to the Toronto Face Dataset (Susskind et al.) used with GAN, CelebA contains high resolution images in colour of celebrity faces. The sharp and colourful samples generated with DCGAN (Figure 1.3, page 6) are in stark contrast to the blurry, noisy and black and white images generated with GAN (Figure 2.3, page 17). The images are far less noisy and much sharper with accurate colour representation. The images in Figure 1.3 (page 6) are randomly selected and as a result the quality of the images varies greatly.

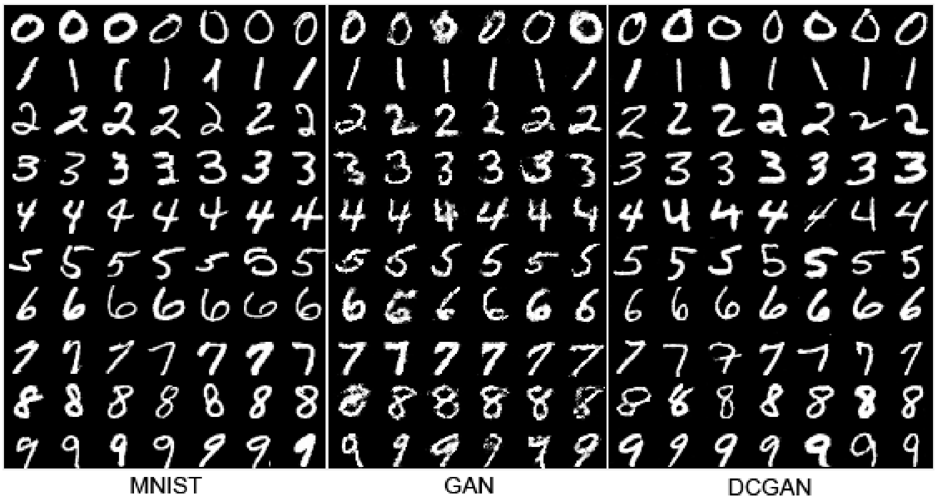

Figure 3.1.: Comparison between original MNIST dataset and images generated with GAN (Goodfellow et al., 2014) and DCGAN (Radford et al., 2015). Samples generated with GAN retain the overall shape of each number, but the lines are quite fuzzy and blurry. DCGAN's samples on the other hand are arguably indistinguishable from the original MNIST dataset. (Source: Radford et al., 2015. With permission.)

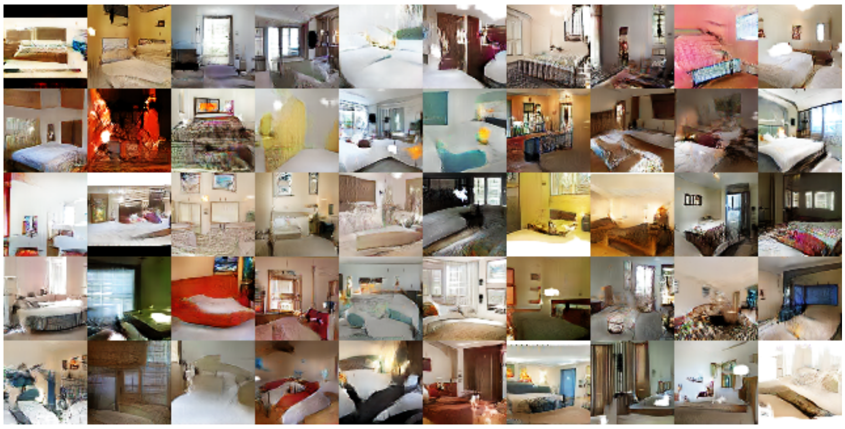

Figure 3.2.: Generated samples from DCGAN (Radford et al., 2015) after 5 epochs of training on bedroom dataset. (Source: Radford et al., 2015. With permission.)

DCGAN has become a popular architecture for GAN. Architectural features make the network flexible and able to dynamically adjust to most kinds of input. The same model that generated faces can also be trained to generate handwritten digits and images of bedrooms without changing any parameters. The original code used by Radford et al. is open source and there exist ports to most popular ANN frameworks.Radford et al. (2015) list three main techniques that DCGAN employs and that help improve the image generation: Batch normalization, the all convolutional network, and sparse connectivity.

- **Batch normalization** (Ioffe and Szegedy, 2015) changes normalization from a task that is executed once during pre-training and on the whole dataset, to something that occurs at each iteration and only on a small batch at a time. Normalization is a step in training of ANNs that usually only occurs during the pre-processing of input data, before the training phase. This technique makes the network less susceptible to unoptimized parameters while also allowing for the use of much higher learning rates, which allows for fewer iterations as well as higher degree of accuracy than normal normalization. Regular normalization can still be applied during pre-processing. Salimans et al. (2016) introduce virtual batch normalization which normalizes the images during pre-processing to be better suited for batch normalization later on.
- **The all convolutional network** (Springenberg et al., 2014) is an approach to convolutional neural networks that replaces the pooling layers, used to down sample input images, with a pure convolutional layer. Regular pooling layers are predefined to behave in a certain way, for instance, max-pooling that is used to down sample an input by choosing its greatest value. An all convolutional network is able to learn its own kind of down sampling without being restricted by the initial architecture of the network. This approach can be used by both the Generator and Discriminator network.
- **Sparse connectivity** (Goodfellow et al., 2016) is the concept of reducing the number of connections within the network, eliminating fully connected layers. This is also known as sparse connections, sparse weights and sparse interactions. The benefits to this approach are fewer number of operations as well as reduced memory consumption. The idea behind sparse connectivity is that not every pixel of an image is important on its own, but rather the embodied shapes and patterns.

3.2. Improved Techniques for Training GANs

Salimans et al. (2016) improve GANs further by presenting new architectural features that they apply to unsupervised and semi-supervised learning. Salimans et al. (2016) also introduced two of the evaluation metrics presented in Section 2.6.2, *object classification* and *visual Turing test.* Three of the presented techniques are detailed below:

- **Feature matching** (Salimans et al., 2016) alters how the Generator learns new features. In regular GAN, the Generator trains by purely using the outputs from the Discriminator. Feature matching changes this by making the Generator learn features directly from the training data, and not the Discriminator. The Discriminator is tasked with selecting the features that are most important. Feature matching is way of addressing *mode collapse* as the Generator must learn from the interior dataset. Experiments conducted with feature matching showed that it makes the networks more stable during training and creates a more robust classifier, but not necessarily more natural images (Salimans et al., 2016).
- **Minibatch discrimination** helps stabilize the Discriminator by letting it handle images in batches rather than a single sample at a time. This reduces the effect of mode collapse as the Generator is penalized for generating similar images. This forces the Generator to improve its samples in a broader spectrum and results in better image generation. Salimans et al. (2016) compare minibatch discrimination with feature matching and conclude that minibatch discrimination converges more quickly resulting in better images, but the resulting classifier is not as strong as one with feature matching. It should be noted that these methods are not compatible with each other because minibatch discrimination requires a regular Discriminator.
- **One-sided label smoothing** softens the outputs from the Discriminator to help balance the performance of the two models. When the Discriminator is trained, it is explicitly told real images should be rated 1 and generated samples should be rated 0. Original label smoothing softens these output to values such as 0.9 and 0.1 respectively. Salimans et al. (2016) use a variation called one-sided label smoothing which only softens the output of real images as values other than 0 for generated samples cause problems.

3.3. Associative Adversarial Networks

Arici and Celikyilmaz (2016) argue that many of the problems related to the training GAN, such as keeping the Discriminator and Generator synchronized are partly caused by the difficulty of mapping flat randomly generated noise, Generator input Z, and the training data. Arici and Celikyilmaz (2016) add an intermediate third neural network as link between the Discriminator and Generator. This additional network, a Restricted Boltzmann machine (RBM), samples data from one of the intermediate hidden layers of the Discriminator to generate inputs to the Generator. This replaces the uniformly generated noise with samples more similar to training data. Arici and Celikyilmaz (2016) trained their system on CelebA and the third network was indeed able to learn the probabilistic model and generate samples based on it. The authors did not go into further detail about how it affected image generation, although they mentioned it did little to improve the learning gap between the Discriminator and Generator.

3.4. Generative Multi-Adversarial Networks

Durugkar et al. (2016b) created a GAN system known as Generative multi-adversarial network (GMAN) with multiple, symmetrical Discriminator models and a single Generator model. The Discriminators are instantiated with slightly varying parameters, but are otherwise architecturally the same and trained in similar fashion to regular GANs.

The training of the Generator on the other hand is somewhat different. Each Discriminator evaluates and outputs its scores on the current generated sample. The scores are processed through a selection metric before being used to train the Generator. This process is shown in Figure 3.3. The selection metrics are divided into two main categories, depending on the strictness of the Discriminators:

- **A formidable adversary** sets the Discriminators to maximize their own performance by providing strict feedback to the lone Generator. A generated sample must please all the Discriminators to get a good rating.
- **A forgiving teacher** limits the Discriminator models to be more favourable towards the Generator model, addressing the GAN balance problem. The feedback from the Discriminators are aggregated and averaged before used with the Generator. In addition, the Generator is allowed to limit the performance of the Discriminators if they become too strong, but is incentivized to challenge itself.

Experiments were conducted with a GAN using the DCGAN architecture and assessed with a variation of GAM (Section 2.6.2) adapted for multiple Discriminator. Using the CIFAR-10 dataset, the GMAN trained using the Discriminator as *a forgiving teacher* without Discriminator performance limitation was declared the best performing model. Experiments using CIFAR-10 were verified with evaluation by object classification using Inception (Section 2.6.2). Using the MNIST dataset saw GMAN with automatic Discriminator limitation outperform all other variations.

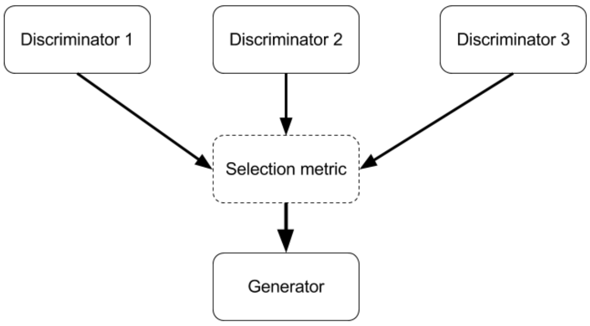

Figure 3.3.: Illustration on the training of the Generator in Generative multi-adversarial network (GMAN) (Durugkar et al., 2016b). Each Discriminator evaluates the samples generated by the Generator. A selection metric will then assess and process the ratings before they are used to train the Generator.

Overall, Durugkar et al. (2016b) found that all variations of GMAN required fewer iterations of training to reach a point of high quality samples compared to a regular single Discriminator GAN. Further, Durugkar et al. (2016b) claim GMAN to be robust against *mode collapse* since the Generator must appease multiple Discriminators.

3.5. Generative Adversarial Parallelization

Im et al. (2016b) argue that a single Generator/Discriminator pair is susceptible to not learning the whole distribution of the training data and suggest using multiple pairs as a way to address this problem.

Generative Adversarial Parallelization (GAP) is the resulting system of this assumption, illustrated in Figure 3.4. GAP removes the usual tight connection between Discriminator and Generator by randomly connecting pairs during training. Experiments show GAP to be robust against both *mode coverage* and *mode collapse* considering the Generators must generate images that fool multiple Discriminators. A single Discriminator can no longer overfit itself towards a single Generator, but must evolve to learn the characteristic of all Generators, which improves balance between models.

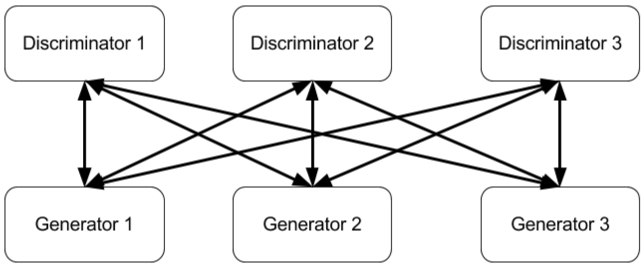

Figure 3.4.: Illustration on the how GAP consist of multiple Discriminators and Generators (Im et al., 2016b).

As the names implies, all models are trained in parallel using multiple GPUs. GAP was implemented with either the DCGAN architecture or an alternate architecture using recurrent neural networks. Evaluation was done with the adaptation of GAM (Im et al., 2016a) mentioned in Section 2.6.2.

Im et al. (2016b) state that GAP strongly outperform regular implementations and suggest images generated with GAP have slightly higher quality then those generated using individual GANs.

3.6. Ensembles of Generative Adversarial Networks

Durugkar et al. (2016a) experiments with using ensembles of GAN, multiple Discriminator and Generator pairs. Durugkar et al. (2016a) express two main observations with GAN they want to address with ensembles. The first observation can be connected to *non-convergence* and how the generated samples can change appearance considerably from epoch to epoch while having the same quality. The second observation is the concern that GAN will not accurately learn the whole data distribution, the problem of *mode coverage*. Durugkar et al. (2016a) define three variations:

- **Standard Ensemble of GANs (eGANs)** Train multiple GANs models, Discriminator and Generator pairs, with varying initial parameters. One of the GAN models are chosen at random when outputting a generated image.
- **Self-ensemble of GANs (seGANs)** Ensembles of GANs with equal initial parameters, but different number of training iterations. Exploits the min-max aspect of GAN in that the networks never converge, but evolve

and shift during training. Less computational expensive than eGAN as only one model is trained from scratch.

- **Cascade of GANs (cGANs)** (Durugkar et al., 2016a) wanted to address the problem of GAN not representing the whole data by creating a cascade of GANs, separated by gates. Generated images that manage to fool the Discriminator were passed down to the next GAN. The goal behind cGAN was to address the second observation, of not representing the whole data distribution.

The experiments were conducted on the CIFAR-10 (Section 2.4) with two ensembles in each variation. Durugkar et al. (2016a) evaluated the systems by using an image retrieval system, described in Section 2.6.2. A second ANN was used for the retrieval process with Euclidian distance used as the evaluation metric. Each of the ensembles outperform regular GAN with eGAN and seGAN being the best performers. Interestingly, seGAN performed slightly better than eGAN while being less computationally expensive. The Euclidian distance was 40% lower with the use seGAN compared to regular GAN.

3.7. Learning from Simulated and Unsupervised Images through Adversarial

Shrivastava et al. (2016) use GAN as a part of a larger machine learning system. The task Shrivastava et al. wanted to solve was improving the realism while maintaining annotated information of generated images. The overall goal was to increase the size of training data for human computer interaction such as eye tracking and gesture recognition of human hands.

The system consists of two parts, a simulator that generates synthetic images with annotations, such as eye tracking data, and a refiner network that uses GAN to improve the quality and maintaining the annotations. The simulator creates images based on a labelled subset. The problem being that these images are not realistic enough and contains artefacts. The refiner network uses GAN and unsupervised learning to turn syntactic images into realistic training data for a larger system.

Shrivastava et al. (2016) change the Discriminator to addresses the instability of GAN and remove artefacts that are common in generated imagery. Firstly, *local adversarial loss* which divides the Discriminator field of operations into smaller local areas instead of the entire image. As with Im et al. (2016b), Shrivastava et al. (2016) believe regular, single Discriminators may put too much emphasis on only a few, strong features in the training images. This may lead to unwanted artefacts and similar looking images, *mode collapse*. *Local adversarial loss* addresses these issues by modifying the Discriminator to output multiple scores for each image compared to the regular single score.

A history of previous images is used when updating the Discriminator. This is done to stabilize the Discriminator and to not reintroduce previous artefacts, addressing the problems commonly related with *non-convergence*. During training, the Discriminator is shown a mix of previous and current generated images from the Generator. Shrivastava et al. (2016) replace half the current batch with previous images and randomly update half the buffer for each iteration of training.

Experiment show that using GAN to generate additional training data is a promising use case for GANs. Shrivastava et al. (2016) use the GAN system to create a new dataset that is of a much larger size than the original. Using this dataset to train a regular CNN on gaze estimation results reduce the error rate from 13.9% to 7.8%.

4. System

The overall system that will be used to conduct the experiments is described in this chapter.

4.1. Machine learning framework

TensorFlow was chosen as the machine learning framework to use during the pre-study. A preference towards Python as the main programming language eliminated some of the possible choices of machine learning framework mentioned in Chapter 2. At the time, both Theano and TensorFlow were comparable features-wise. Previous experience with Theano left more to be desired. Theano was not created with machine learning in mind and thus the documentation and built in functions were unintuitive for this purpose. TensorFlow on the other hand, was created specifically for this purpose, having built in support for most common features related to ANNs. The documentation is also thorough and offers simple and intuitive tutorials. The quality of the publicly available implementations of GAN was eventually the biggest factor in selection TensorFlow as the framework of choice.

4.2. Initial implementation

As mentioned in the previous section, the pre-study was initially open to different choices of machine learning framework. Multiple open implementations of GAN were researched before settling on a TensorFlow port of the Radford et al. (2015) original implementation of DCGAN[1]. This implementation is continually updated and used by many as a framework for their projects. This is understandable as the code is well structured, highly configurable, has automatic tools for visualization and built in support for running on custom datasets. This was the reasoning for choosing it as a starting point for this project as well. At the start of this thesis, other implementations were also researched such as the ones used by Salimans et al. (2016), but neither one was as robust and flexible as TensorFlow port used.

[1] Radford et al. (2015) implemented DCGAN in both Theano (*https : //github.com/Newmu/dcgan_code*) and Torch (*https : //github.com/soumith/dcgan.torch*

The DCGAN architecture is used by most image related GANs including most of the related work mentioned in 2.6 and is thus a good point of comparison. The framework has been continually updated since the pre-study and support the latest versions of TensorFlow.

4.3. Hardware

The systems at disposal at the beginning of the project were a MacBook Pro running MacOS or a Windows desktop computer. The desktop computer was considerably more powerful with more CPU cores, 3 times as much RAM and most importantly a GPU from NVIDIA. However, as TensorFlow was not available on Windows at that time, the MacBook Pro was tested out first. Installing TensorFlow and getting DCGAN to run was a trivial process. However, the performance and training time was quite slow with only CPU calculations being available. Using the MacBook Pro as the main system was an option, but not preferred with the promises of GPU based parallelization. The next step was to get TensorFlow running on the desktop computer. Some attempts were made at running it under Microsoft Windows with a virtual machine. This was quickly abandoned as it was uncovered that GPU calculation with this configuration was impossible. The next step was then to create a partition and install Linux. It took a few days with some missteps along the way, but was well worth it in the end. Running TensorFlow and DCGAN with CPU was only slightly faster than on the MacBook Pro. However, using the GPU saw an exponential increase in computational power and the training time per epoch was greatly reduced. It was clear that this was the system configuration of choice. The computers specifications is summarized in Table 4.1. Note that GPU-based calculations in TensorFlow is limited by the 4GB of memory in the GPU. CPU-based calculations can use all the system RAM.

4.4. GAN model architecture

The DCGAN architecture has arguably become the standard architecture for GANs, especially concerned with image generation. This means that regular a DCGAN is used as the baseline for comparison between the proposed techniques. This will be the case for this thesis as well. The default configuration and architecture of Kim's TensorFlow implementation are therefor left unchanged. Figure 4.1 from Radford et al. (2015) shows the architecture of the DCGAN Generator. Table 4.2 summarizes the most relevant settings. The proposed techniques will alter the training process, but a kept as consistent as possible.

	System
OS	Ubuntu 16.06
CPU	2x Intel Xeon E5640 8 physical core 16 with Intel's hyperthreading
GPU	GeForce GTX 960 4GB RAM, 1024 CUDA cores
RAM	23,5 GB
Software	TensorFlow version 1.0 Python 3.5

Table 4.1.: Research machine specifications

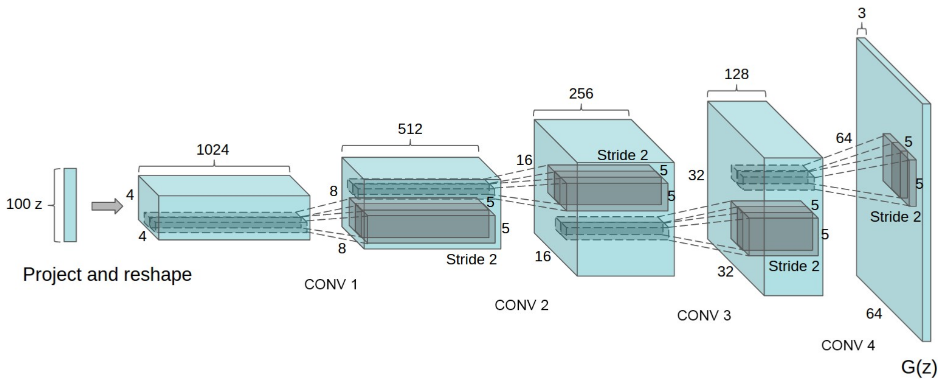

Figure 4.1.: Overview of the generative model in a DCGAN architecture. (Source: Radford et al., 2015. With permission.)

Learning rate	0.0002
Momentum term of AdamOptimizer	0.5
Image input dimensions	64x64
Image output dimensions	64x64
Image color channels	3
Generator Z-noise input size	100

Table 4.2.: DCGAN model specifications

Layer	**Dimension Discrimantor**	**Dimension Generator**
Input layer	64x64x3	100
Hidden layer 1	32x32x64	4x4x512
Hidden layer 2	16x16x128	8x8x256
Hidden layer 3	8x8x256	16x16x128
Hidden layer 4	4x4x512	32x32x64
Output layer	1	64x64x3

Table 4.3.: Discriminator and Generator model architecture

4.5. Minor framework changes

The implementation, as previously mentioned, is heavily based on Kim's TensorFlow port of DCGAN. There have been numerous changes to implementation that will be described in the coming sections. The minor changes done to framework will be listed in this section.

- Naming and organization — A problem from pre-study was that the original framework overwrites file every time the framework is run. Output is saved the same folder and with the same name. This was improved for this project by automatically organize the generated samples in folder named after the settings used. The generated samples also reflect these settings.
- Image files — The framework outputs a grid of generated samples at a fixed interval. This showed how the quality of the images were progressive, but were otherwise not that useful for analysis. To get a better understanding and overview on the learning task, the generated samples are evaluated with Discriminator to show their rating. These ratings are then printed on the output file. Further, images from the training set are also added in to better compare the generated images to the real one as well as showing the evaluation of the real images.
- Analysis tools — Tools evaluate the performance of the Discriminator model at fixed intervals throughout training. The Discriminator is evaluated by classifying images from the training set and images generated by the Generator. The results are aggregated to better visualize the learning curve.
- Checkpoints — TensorFlow has a robust method for saving and restoring past checkpoints of a model. The original framework would save checkpoints based on number of iterations, overwrite past checkpoints and delete older checkpoints if kept more than five. To consistently evaluate the models throughout the learning process, the framework was tweaked to save

checkpoint after every epoch of training and to never remove older checkpoints. The checkpoints names are also named according to the settings used.

5. Data

Chapter 2 introduced the publicly available datasets researched for this project. This chapter will discuss potential sources of image for the creation of custom datasets. Furthermore, all datasets public and custom datasets will be discussed in context of using them to train a GAN system for image generation. Unsupervised learning is highly dependent on the training dataset as this is the source of all attained knowledge. The overall goal of this chapter is to highlight and discuss the characteristics of each dataset and how they may affect image generation with GAN. To better help summarize, each dataset is custom rated according to the difficulty of using it to train a GAN system. This is further explained in Section 5.1. Table 5.2 sums up the selected dataset with details about the data as well as the difficulty rating.

5.1. Difficulty Rating

Each dataset discussed in this chapter will have a custom rating of difficulty. This rating reflects how well suited the images are for unsupervised learning with GAN and are based on knowledge gathered through experimentation in the pre-study. Generally, datasets with higher degree of similarity between images result in higher quality. The number of images in the dataset is also important and larger datasets are preferred. The rating has three classification: *easy*, *normal* and *hard*. Table 5.1 summarizes each rating. The ratings are included to better contextualize the results in Chapter 7, and clarify why results may differ between various datasets. A dataset with a lower difficulty rating will probably result in images of higher quality, but a more difficult dataset may better show the subtle variations between techniques. The difficulty rating is hence also used to get an even collection of datasets and not just the ones most suited for GAN.

5.2. Personal photos datasets

One of the first dataset tested during the pre-study was a set of about one thousand personal images containing my face somewhere in the image. About 10% of the dataset were taken in the same photoshoot. These images have my face aligned centrally and with the same background. The dataset was used to train

Difficulty	Description
Easy	Images are similar. Common framing, structure and features
Medium	Images share many similarities, but framing and rotation varies
Hard	The dataset contains images of varying subjects, objects and angels.

Table 5.1.: Description for difficulty rating of datasets

Dataset	Size	Annotations	Resolution	Rating
CelebA	202,599	Name,description	Variable	Easy
CelebA Aligned	202,599	Name, description	178x218	Easy
MNIST	60,000	Digit	28x28	Easy
CAT	10,000	Facial features	Variable	Medium
IKEA Original	7,383	None	250x250	Medium
IKEA (Augmented)	191,828	None	250x250	Medium
CIFAR-10	60,000	Objects	32x32	Hard
Flickr30k	31,783	Description	Variable	Hard

Table 5.2.: Summary of the datasets selected for use in this project.

a DCGAN model. The output is shown in Figure 5.1. The problems of mode collapse and overfitting is clearly visible. The image in the lower right corner is a copy of one of the training images, variations of this image is visible throughout. The model is able to pick up similar features such as general features of the human face. It was clear from this experiments that this dataset was not suited for training as it was both too small and the images were too different.

5.3. Instagram datasets

A great deal of effort was spent during the pre-study to research the potential for using images from Instagram as a source for training images. Instagram is a large social image sharing network with billions of images tagged with relevant metadata. Instagram does not have a public API where images can be downloaded through official channels.

The first attempt at assembling a dataset was done early in the pre-study. A tool[1] was used to download the uploaded images for a given profile. The tool was used on my personal profile to create a small dataset of around 300 images. These

[1]https://github.com/rarcega/instagramscraper

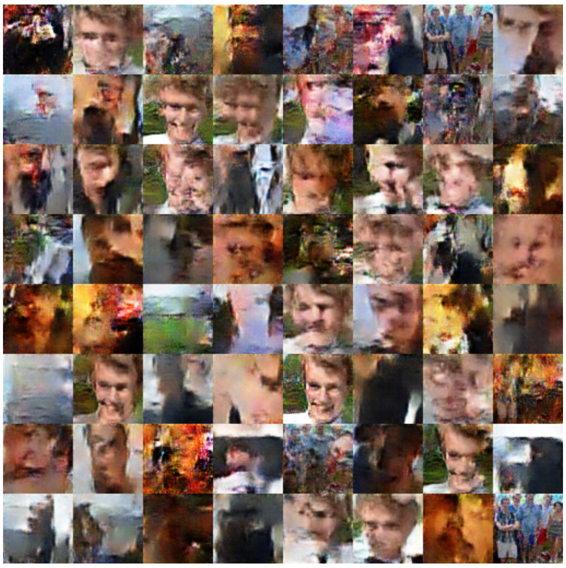
Figure 5.1.: Samples generated by DCGAN after training on personal images.

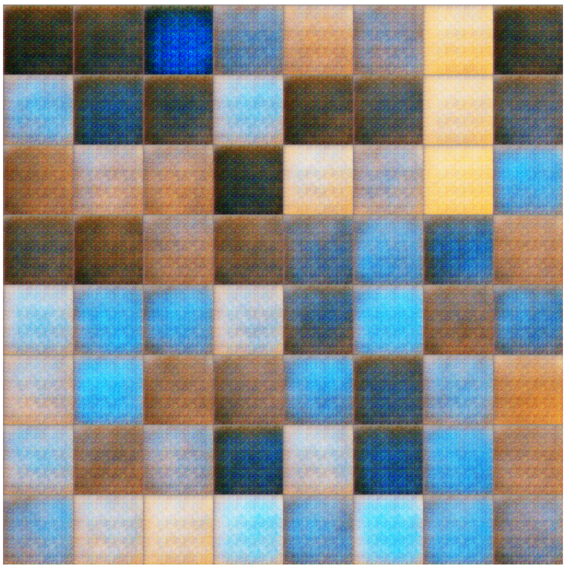
Figure 5.2.: Samples generated by DCGAN after training on images from personal Instagram account. DCGAN is not able to find any similarities between the images and the models learning collapses.

images vary widely in size, what they depict and the colours used. Results from running this dataset on DCGAN were not good, shown in Figure 5.2. DCGAN was not able to generalize and the model completely collapsed.

A larger dataset with more similar images was needed. The second attempt at using Instagram as a source, used an unofficial downloader. A set of about 60,000 images were collected. The images were all tagged with the hashtag sunset This dataset was never used during the pre-study as it was too hard to get running, caused by various problems with the images. Even with better error-handling, a large manual curation would most likely still be required. In addition, a manual inspection of the images showed that many images were low resolution, poor quality and hard to make out. Many were also randomly tagged and not relevant to the search query. Instagram will for all of these reasons not be used as an image source for this project. Other pre-existing datasets are more viable and require less pre-processing and manual labour before use.

5.4. IKEA dataset

The IKEA dataset is a custom collection of product images for furniture from IKEA's website[2]. The problems of the previous datasets led to idea of using the product images from the furniture retailer IKEA. IKEA offers thousands of products with each product having images following the same guidelines. Every product image shares the same composition with the product aligned centrally and a white background. The images were collected by recursively web-crawling IKEA's website and downloading every available images. The data collection was time consuming and required much manual effort. The crawler was prone to random crashes and had to be restarted multiple times. This caused the dataset to contain a great deal of duplicates as the crawler were unaware of previously download images. The initial set contained some hundred thousand images with various depictions and dimensions. This set was reduced to about 60.000 images by removing the images of resolution other than 250x250.

A second manual curation effort using an automatic tool to remove all duplicates and reduced the number of images from 60.000 to about 10.000. Testing this dataset with DCGAN produced decent results, but ambiguous images showing whole rooms or product images for textile and other fabrics severely degraded the generated images. Some of images quite clearly depict furniture. The rest of the images are harder to make out and consists mostly of noisy patterns. It is believed that this was result of the training data, not the algorithm. Many of the images in the set are for textiles which have mostly rectangular shaped boxes

[2]http://www.ikea.com/no/no/

as product images. Figure 5.3 shows a generated sample by this dataset. Notice the repeating images without the white background, around top left and bottom right. These images are the results of a the previously mentioned ambiguous images and images showing interior rooms not following the usual guidelines with white background.

Another manual curation effort removed most of the ambiguous images with mostly furniture remaining. The final dataset contains about 7.000 images. The previous attempts at dataset collection proved that the size is important. It is a known fact that machine learning prefers as much training training as possible. The IKEA dataset share a common framing which makes is susceptible to augmentations, artificially increase the size of the dataset by making slightly adjusted copies of an original image. The white background greatly expands possible alterations as the object can be moved around the image. This can compensate for the dataset relatively small size. The size of the dataset was artificially increased by implementing pre-processing techniques that transforms the images slightly to produce a larger dataset. These slight transformations are random and include moving the products around, rotating, stretching and flipping the images. The results from using the dataset with DCGAN are shown in Figure 5.3. The images that were much sharper and cleaner than the previous iteration. Many of the images resemble furniture, but the images are still simple to classify as generated. The IKEA dataset is therefore rated at *Medium* difficulty.

5.5. Custom Flickr datasets

Flickr is social network for sharing images online. Like Instagram, Flickr has billions of images uploaded by its users and has arguably a higher level of quality. Unlike Instagram, Flickr offers a public API with full access to search tools and metadata. Initially, Flickr was planned as source for training images for DCGAN. Full access to search tools and metadata create datasets of various depictions. The plans for creating such a tool were eventually dropped in favour of already existing datasets, with the reasoning being that it would be too time consuming as well as shifting the focus of attention away from the research goals.

5.6. Public datasets

The public datasets were introduced in Section 2.4.1. This section will briefly discuss the difficulty of each dataset for use with GAN.

- **CelebA** — Preliminary result from the pre-study showed that the large sample size in conjunction with the general similarity of human faces pro-

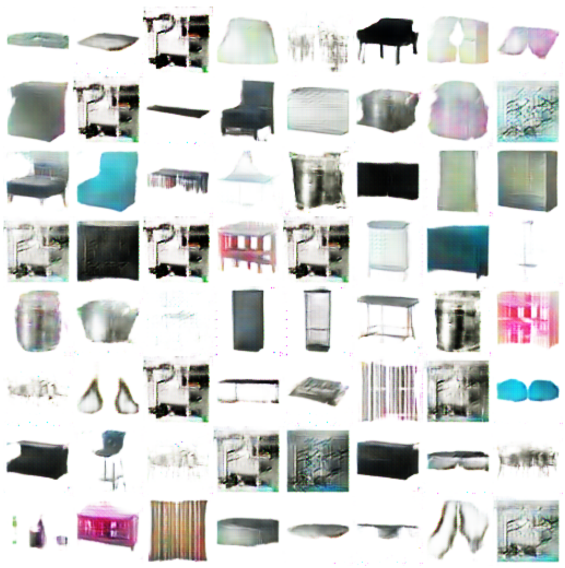

Figure 5.3.: Samples generated with DCGAN on the first iteration of the custom IKEA dataset of size 7.000.

duced decent looking images although the variety of poses occasionally resulted in some less then desirable images. CelebA is for this reason rated at *easy* difficulty.

- **MNIST** is a simple for modern machine learning algorithms and is therefore rated as *easy*.
- **CAT** — Framing and depictions varies between each image which may result in some difficulties for DCGAN. This dataset is thereby rated at *medium* difficulty.
- **CIFAR**-10 is consequently one of the harder ones to generate natural looking images from, and earning a difficulty rating of *hard*. Still, it should be helpful in showing the variations from the various techniques.
- **Flicrk30k** — The images vary widely in depiction as well as physical properties like aspect ratio and resolution. These properties may make the dataset hard to generalize and is thus rated as a *hard* dataset.

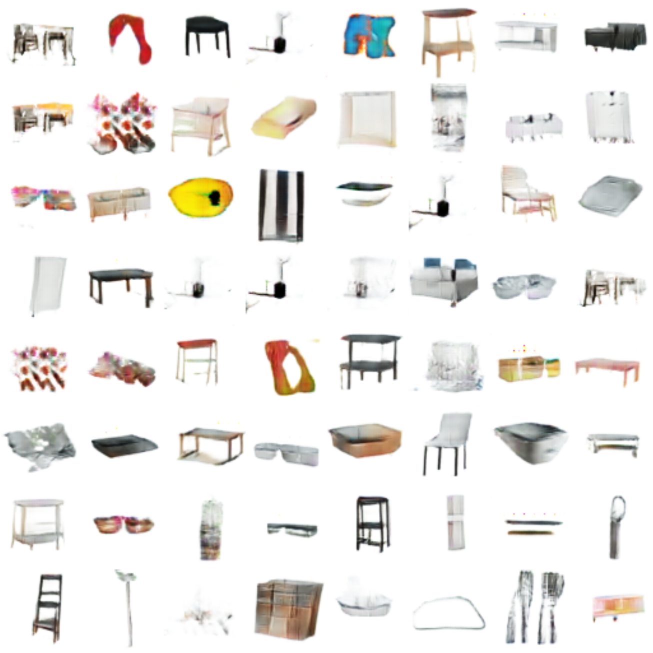
Figure 5.4.: Random samples from curated IKEA dataset with pre-processed

6. Methodology

This chapter will discus the research question in regards to the state-of-the-art presented in Chapter 3, the system described in Chapter 4 and the training data detail in Chapter 5. One ore more solutions is then proposed to answer the research question. The proposed techniques will be tested and the results presented in Chapter 7.

6.1. Generator input improvements

One of issues with GANs presented in Section 2.6.1 is the problem of synchronising the performance of the generative model and discriminative model during training. More specifically that the Discriminator may become too strong compared to the Generator.

RQ1 was defined to explore the possibility of using guided alterations to the input of the Generator, in turn making a strong model and hopefully create a more balanced system.

RQ1 *How is performance affected by altering the input to be more favorable towards the Generator model in a Generative Adversarial Network?*

6.1.1. Discussion

The main approach to counter an over powered Discriminator has been to use artificial limitations, such as pausing the Discriminators training while the Generator continues. Although Goodfellow acknowledge this as a problem in Goodfellow (2017), he urged not to not limit the Discriminator performance as a strong Discriminator is required to accurately represent the data.

Durugkar et al. (2016b) proved Goodfellow suspensions, Section 3.4. The best performing, at least for high resolution image data, was the $forgivingteacher$ model that aggregated the result of multiple Discriminators and without restricting their output.

The Associative Adversarial Network proposed by Arici and Celikyilmaz (2016) also attempted to balance the performance by making the learning task easier for the Generator. Their assumption was that the issue was partly caused by the difficulty of mapping flat noise samples to realistic output image. Their proposed solution was by using a third neural network as a link between the Generator and Discriminator. This approach is based log-likelihood which multiple times has been discouraged with use on image data. Their approach also seems somewhat excessive and unnecessary since this associative network essentially acts as a second Generator. Still, their initial assumption is interesting and worth exploring further.

6.1.2. Proposed solution

The three techniques proposed in this section is based on a similar assumption as the one proposed in Arici and Celikyilmaz (2016), the large distribution of the Z noise is limiting the learning process of the Generator.

Static Reusable Noise

The discriminative model in a GAN allowed to focus its learning on training dataset. The Generator is given the more difficult task of mapping a large, almost infinite set of random noise samples to natural generated images. The Discriminator will learn on the same images every iteration while the Generator will seldom, if ever see the same input during training.

Static Reusable Noise (SRN) is a proposed technique reuse the noise samples every epoch. The noise samples will therefore be familiar to Generator which may better focus its effort at generating better images. The generated images will be quite similar each epoch and the feedback from the Discriminator will still be relevant during the next epoch.

The nosie samples are generated before training commences and is sampled through seed. This enables the same noise to be reused during evaluation. Figure 6.1 illustrates the proposed technique.

Image Based Noise Generation

Image Based Noise Generation (IBNG) is a proposed techniques that generates noise samples for the Generator based on images from the training dataset. Samples from the training set are first scaled down to the smaller size of the noise data and then converted to black and white values in the range $[-1, 1]$. Figure 6.2 illustrates the three steps of the algorithm. Figure 6.3 and Figure 6.4 shows samples generated from the IKEA and CelebB datasets respectively.

IBNG is based on a similar assumption as the one proposed in Arici and Celikyilmaz (2016). The overall solutions to problem are similar in that both replaces the

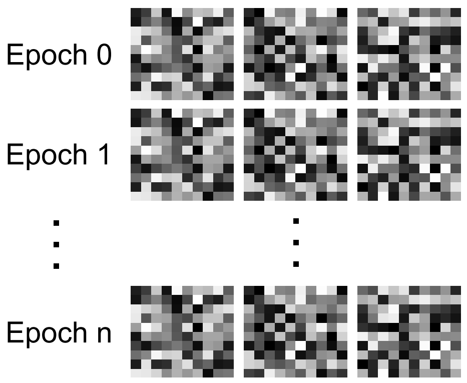

Figure 6.1.: Static Reusable Noise (SRN) is a proposed technique reuses the otherwise random input of the generative model in a GAN. The noise samples are created once and reused every epoch.

regular noise with input more akin to the training data. IBNG is a simpler and less computationally expensive compared to the Associative Network proposed by Arici and Celikyilmaz (2016).

Audition Based Noise Selection

Audition Based Noise Selections (ABNS) use the Discriminator to select representative generated samples to guide the training. ABNS is a pre-processing technique that selects the noise to be used for training. Audition based noise selections is based on an assumption that a strong Discriminator is able to accurately select generated samples that provide the most feedback to Generator.

The techniques is called *audition based* as the technique auditions a large set of noise samples where only a select few are chosen. The number of auditioned noise samples is set by the variable *audition size* which is multiplied with the batch size. For this thesis, *audition size* is set to 3 and 6.

A selection metric picks images based on their score and assembles a regularly sized batch. This batch is used to train the system normally without any modifications to the GAN model. The selection metric has two modes:

- **Best** will only pick the images that attained the highest rating with the Discriminator.
- **Mixed** is used to give the Generator balanced feedback on its image generation. Mixed mode divides the batch in three and fills it with equal amounts

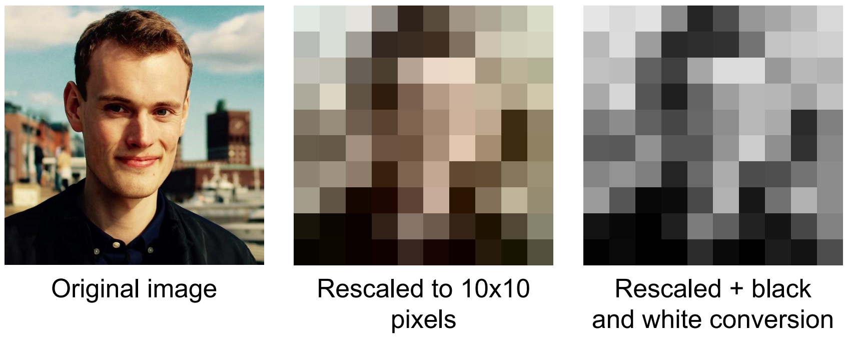

Figure 6.2.: Illustration on the process behind image based noise generation. Images from the training dataset are scaled down and converted to black and white. The images are then used as input to the Generator.

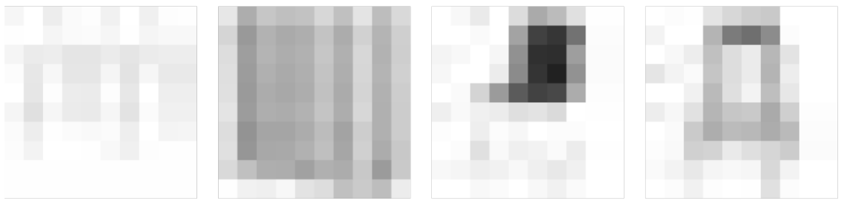

Figure 6.3.: Examples of image based noise generation used on the IKEA dataset to generated input to the Generator.

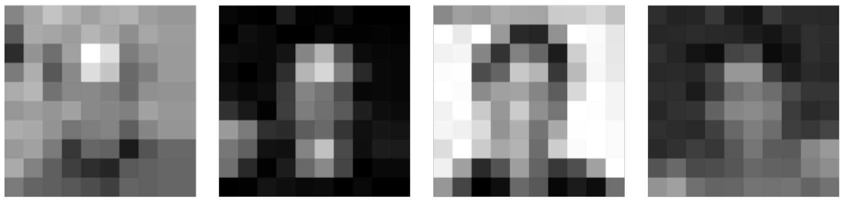

Figure 6.4.: Examples of image based noise generation used on the celebrity faces from the CelebA dataset. The structure of the human face is clearly visible on all four images.

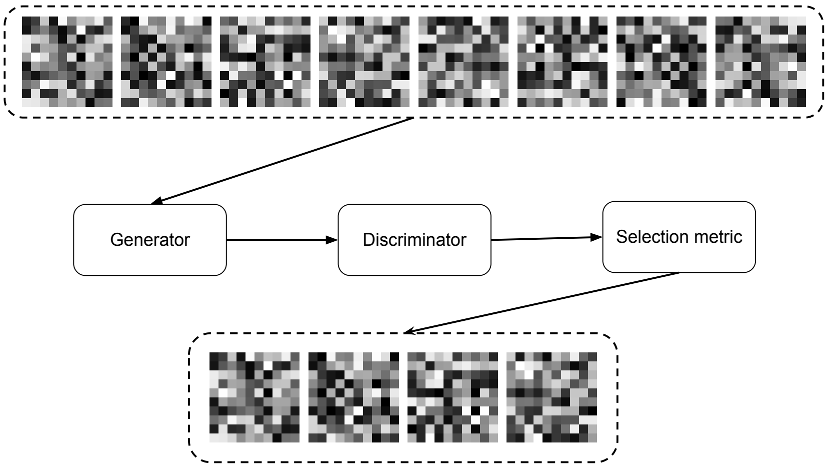

Figure 6.5.: Audition Based Noise Selection creates a large set of Z noise samples and generated images based on them. The generated images are evaluated by the Discriminator. A metric selects the most representative image such as only the best or mix between qualities. The noise used to generated the selected images are then used to train the system normally.

of images with the highest and lowest rating. The rest are randomly generated.

6.2. Generative Multi-Adversarial Networks

Chapter 3 presented several proposed GAN models using additional ANNs. RQ2 was proposed to explore the use of GANs with additional discriminative models, Generative Multi-Adversarial Networks (GMAN).

RQ2 *In what way is performance of a Generative Adversarial Network model impacted by adding additional, asymmetrically trained Discriminator models?*

6.2.1. Discussion

Durugkar et al. (2016b) used multiple Discriminators against a single Generator in their system. The evaluation from the Discriminators would either be aggregated or selected through a metric. Durugkar et al. (2016b) experimented with the Discriminators being either strict or kind in their evaluation. They found that the kinder model performed best which again indicates that the generative and discriminant model should be balanced.

Im et al. (2016b) created a system of multiple GAN pairs that interchanged during training. Im et al. (2016b) argued that their GAP system removed the tight connection between a single Generator/Discriminator pair and reduces the effect of problems such as mode coverage. Durugkar et al. (2016a) used ensembles of GAN models for image retrieval. Tests on the CIFAR-10 dataset showed that ensembles consisting of the same model at different epochs were able to outperform regular GAN by 40%.

These papers showed that GANs with additional models is a promising evolution for GANs and should be further explored. The discussed papers still maintain most of tropes of regular GAN which leaves certain areas unexplored. Firstly, all models are trained in the same way on the same data. The Discriminator is trained with images from the training set and current images from the Generator. Varying the training data may make the models focus on different aspects with helps the model overall.

Models in GAP are interchanged at an interval of multiple iterations. This differs from GMAN where the Generator is trained by multiple Discriminators each iteration. Although Im et al. (2016b) argues this removes the tight connection between models, it is easy to believe the models may quickly forget knowledge from previous models in favour of the current opponent. This may halt the learning process compared to multiple fixed opponents. The Generator in GMAN must evolve to please multiple opponents which may create a much stronger model overall.

6.2.2. Proposed solution

Generative Multi-Adversarial Network with Historic Discriminator (GMAN-HD) is a proposed GAN architecture with a second discriminative model. The additional discriminative network is trained on a record consisting of previously generated samples. GMAN-HD is inspired by the success of using additional Discriminator networks as seen in Durugkar et al. (2016b), Im et al. (2016b). The historical archive is inspired by the archive used by the Discriminator in Shrivastava et al. (2016).

The historic archive is the same size as a regular batch. At each iteration, a minimum of a quarter and up to one half of the archive is randomly replaced with the best images selected through ABNS with audition size equal a regular batch. The archive was originally much larger with images spanning the last epochs. This did not perform well and the archive was changed to be more up to date. The auditions size was kept low to increase performance. The Generator is trained against both Discriminators each iteration. The regular Discriminator is trained in usual fashion, but the Historic Discriminator is trained against the training data and the images in the archive. The Historic Discriminator is never shown the current output from the Generator.

6.3. Evaluation metric

Section 2.6.2 introduced the problem of objectively evaluating GAN and described some evaluation metrics used. Section 6.3.1 discusses the proposed metric from Section 2.6.2 with respect to this project. Finally, Section 6.3.2 will detail the selected evaluation metric and its custom implementation.

This thesis has to far described the system, the hardware it runs on and the training data. This thesis is focused on GAN specialised with image generation with two research question related to this topic. The final evaluation metric must fit within the specification defined while at the same time help reach the project goal and answer RQ3:

RQ3 *How can the performance of the suggested techniques related to RQ1 and RQ2 be accurately measured and objectively evaluated?*

6.3.1. Discussion

Theis et al. (2016) concluded that both the training and evaluation of a GAN model should be modelled after the target application which for this thesis is image generation.

Using log-likelihoods as an evaluation metric for GANs were first introduced by Goodfellow Goodfellow et al. (2014). However, as stated in both Goodfellow et al. (2014) and Theis et al. (2016), log-likelihood is not suitable for evaluation of higher dimensional data, such as image, and has largely been phased out as GANs have evolved.

Similarly, using nearest neighbour algorithm to compare pixel data in images was used by Durugkar et al. (2016a) and Ledig et al. (2016). Theis et al. (2016) argued is was not sufficient and will not be used for this project.

Salimans et al. (2016) used human annotator to evaluate the quality of their GAN. Conducting a visual Turing test would certainly be possible for this thesis. It is however, expensive and time consuming to conduct human evaluation. The tests conducted in Salimans et al. (2016) rated images as either *fake* or *real*. This may not be accurate enough to pick up the slights variations of quality between techniques. It is doubtful that the techniques proposed in this thesis will impact the quality of image generation in such a way it is able to deceive a human evaluator. Evaluation with human annotators will therefore not be the main evaluation metric for this thesis, but some degree of human inspections may still be necessary to detail some of the differences in the generated images.

Salimans et al. (2016) also used object classification to evaluate a GAN model. This is a good approach for certain problems. It provides a consistent and objective score on the image quality of images. There are however two main concerns related to this thesis. First of this approach requires a labelled dataset for comparison. Inception (Szegedy et al., 2016) is trained on CIFAR-10 so using this dataset would work, but will otherwise limit the number of usable dataset considerably. Secondly, using the Inception network is computation expensive and it on the modest hardware. If used on other datasets, Inception requires the last layers to be retrained.

Generative Adversarial Metric (GAM) (Im et al., 2016a) is a general method for comparing two GANs. GAM does not directly evaluate the quality of image generation, but compares two trained GAN models by evaluating each other. It is however not perfect since it does not evaluate image quality compared to the training dataset. As Salimans et al. (2016) encountered when comparing *Feature matching* and *Minibatch discrimination*, certain techniques may create a better classifier(Discriminators) without improving image quality.

Still, GAM is the best suited evaluation metric for this thesis and general framework means that GAM can be customized to the desired task. The background chapter mentions two implementations for GAM. Both of these are specialized for GAM models with multiple Discriminators and will not be used.
Proposed solution

6.3.2. Proposed solution

Generative Adversarial Metric (GAM) (Im et al., 2016a) was selected as the main evaluation metric for this thesis. GAM is a general framework for comparing two GAN models.

The custom implementation of GAM is named Serial Generative Adversarial Metric (SGAM). A high-level overview of SGAM is represented in Algorithm 1. Serial refers to the fact that checkpoints of the GAN models are never loaded at the same time, but one at time in series. The main reason behind this is memory consumption. Keeping two models in memory at the same time requires double the GPU memory. The generated samples are stored in the systems main memory. All experiments and evaluations are run on the same computer with relatively low amount of GPU memory for such as task. Running the two models in series is therefore a simple trick to guarantee there is always enough available GPU memory. TensorFlow has built in support for saving and restoring checkpoints of a previously trained model. Loading a model is done is generally quite efficient and a fast process.

The simplicity of TensorFlow checkpoints is exploited further by evaluating models throughout the learning process. This requires the models to have checkpoints saved after each epoch of training. SGAM load these checkpoints one epoch at a

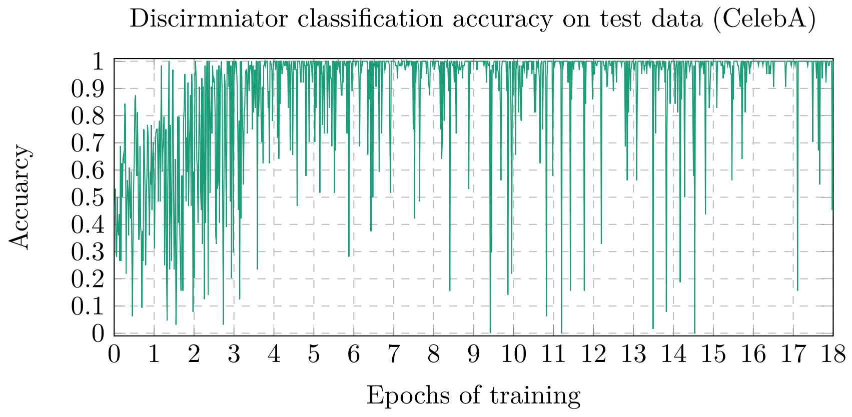

Figure 6.6.: This plot of the classification accuracy of the Discriminator on test data shows some of the problem of evaluating GANs.

time to find the GAM result at that the time during training. The classification accuracy is also averaged for each checkpoint which is used at the end to calculate a global winner. Both the local and global results are saved to file for comparison later on.

Implementing GAM through the equations in (Im et al., 2016a) resulted in an unstable evaluation when tested on techniques proposed for this project. The original formulas states that the classification error should be used. It is natural to assume this means the amount of wrongly classified samples, for instance an error rate of 0.1 means 10% on the images were classified wrong. For the test to qualify, the ratio between the two GAN must be roughly equal to 1. Testing GAM with this assumption however proved difficult as the error rate were generally so low that the test fail. For instance, comparing two GANs achieving an error rate of 0.01 and 0.001 will fail the test.

Attempts to stabilize GAM includes making the accuracy measure stricter and drastically increasing the amount of images tested. It was also believed the global averaging across multiple epochs would help. None of these methods helped stabilized the metric. Even testing the exact same model against itself resulted in GAM selecting one model as better one with a sample ratio of 0.8885.

The min max battle of GANs separates them from conventional ANNs. The models are constantly evolving caused by the min-max battle. An example plot of the classification accuracy on test data is shown in Figure 6.6 and is an argument against a too strong test.

The solution to this problem was simply to modify the equations to use classification accuracy and not the error. This helps stabilize the metric when error rates are low, but still fails if the accuracies are too different. This change may not be ideal for all projects and GAN models, but was necessary for having a stable evaluation metric that worked with the models and data used in this project.

Using the classification accuracy requires the equations from Im et al. (2016a) shown in Section 2.6.2 to be altered. The updated equations are shown in Equation 6.2, 6.3 and 6.4. Testing a trained GAN model against itself, using the updated equations, resulted in a tie with a sample ratio of 1.0004.

$$\begin{cases} \textit{correct} \text{ if } x = \textit{generated} \text{ and } D(z) < 0.5 \\ \textit{correct} \text{ if } x = \textit{real} \text{ and } D(x) > 0.5 \\ \textit{wrong} \text{ otherwize} \end{cases} \tag{6.1}$$

$$r_{sample} = \frac{1 - \epsilon(D_1(G_2(z)))}{1 - \epsilon(D_2(G_1(z)))} \tag{6.2}$$

$$r_{test} = \frac{1 - \epsilon(D_1(x))}{1 - \epsilon(D_2(x))} \tag{6.3}$$

$$winner = \begin{cases} \text{GAN 1 if } r_{sample} > 1 \text{ and } r_{test} \simeq 1 \\ \text{GAN 2 if } r_{sample} < 1 \text{ and } r_{test} \simeq 1 \\ \text{Tie otherwise} \end{cases} \tag{6.4}$$

```
Define a test dataset, $samples_{test}$;
foreach epoch of training do
    begin Load the corresponding checkpoint for $GAN_1$
        Find classification accuracy of $D_1$ on test dataset;
        Generate dataset $samples_1$;
    end
    begin Load the corresponding checkpoint for $GAN_2$
        Find classification accuracy of $D_2$ on test dataset;
        Find classification accuracy of $D_2$ on $samples_1$;
        Generate dataset $samples_2$;
    end
    begin Load the corresponding checkpoint for $GAN_1$ again
        Find classification accuracy of $D_1$ on $samples_2$;
    end
    Calculate $r_{test}$ (Eq 6.2) and $r_{sample}$ (Eq 6.3);
    Use Eq 6.4 to decide local result;
end
Average the local classification accuracies;
Calculate $r_{test}$ (Eq 6.2) and $r_{sample}$ (Eq 6.3);
Use Equation 2.4 decide global result;
```

Algorithm 1: Serial Generative Adversarial Metric (SGAM) algorithm

7. Results and Discussion

The previous chapter introduced four main techniques proposed by this thesis: *Static Reusable Noise* (SRN), *Image Based Noise Generation* (IBNG), *Audition Based Noise Selection* (ABNS) and Generative Multi-Adversarial Network with Historic Discriminator (GMAN-HD). The results of experimentation with these techniques on three different will be presented, Section 7.2 and discussed, Section . Section 7.3.

7.1. Experimental Setup

Experimental setup briefly describes the dataset used and the evaluation metric.

7.1.1. Data

The techniques will be trained on three datasets: CelebA and CIFAR-10, Section 2.4, and the custom IKEA dataset, Section 5.4. The three remaining datasets, MNIST, Flickr30k and CAT, will not be used. MNIST is a simple dataset. DCGAN is allready more than capable of using it. The Flickr30k and CATA datasets were omitted because the variable image size and formats was too inconsistent to train. ANNs require images to be of a fixed size. Automatic cropping of the images from was inconsistent. The remaining three dataset represent the three difficulty ratings and should be good enough to highlight the variations between techniques.

7.1.2. Evaluation

Serial Generative adversarial metric (SGAM), see Section 6.3, is the main evaluation metric. Each technique will be compared to against a default implementation of DCGAN. The final output score will not the sole determining factor. SGAM calculates local result after each epoch and the results of these will also be a point of discussion. A technique may for instance be better during the early epochs, but worse overall. Some manual inspection of the generated samples will also be conducted to highlight some of the variations between techniques.

To maintain a balance between performance and accuracy, the test accuracy and sample accuracy are calculated by classifying 12,88 batches, equal to 200 batches. The test dataset consists of image from the training data.

SGAM results

Figure 7.1 shows an example plot of a SGAM comparison. The test score is solely based on the Discriminator ability to classify a set of real images as real. The example plotted in Figure 7.1 shows that the test scores are comparable at around 80% accuracy.

The sample score is the real point of comparisons, as it evaluates the actual image generation of the two models. The sample score is the accuracy of a model classifying the generated samples from the other model correctly as fake. The sample score for GAN_1 shows that it is able to correctly classify most of the samples generated by GAN_2. At the same time, GAN_2 is only able to correctly classify about half of the samples from GAN_1. The rest is believed to be real images. The better model is in this example GAN_1 since it is able to both fool the other model while not getting fooled itself. The Discriminator is taught that images from the training set should be rated high and samples generated from the Generator to be rated low. This means that all knowledge about the characteristics of a generated images comes from the models Generator and the sample score is therefore an indication on the quality of the models Generator as well. A model with a lower samples score will have a worse Generator than the other.

7.2. Results

The results of the experiments with of the four techniques proposed in Chapter 6 are presented in this section.

7.2.1. Static reusable noise

The SGAM results for comparing models trained with SRN against regular DCGAN are shown in Table 7.1 and Table 7.2. The effect of SRN varied depending on the dataset, but did not have a large effect on performance. Notice that the model's performance was not affected by whether random noise or the noise used during training was used to generat samples.

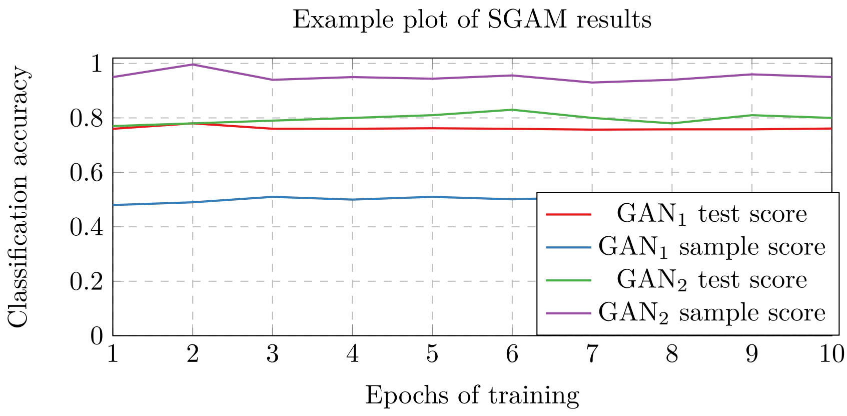

Figure 7.1.: The plots shows an example result from SGAM comparison between GAN_1 and GAN_2. The classification accuracy on test data is comparable at around 80% for both models. GAN_1 is however outperforming GAN_2.

CelebA

When trained on CelebA dataset, the model using SRN won both globally and most local victorious. Figure 7.2 shows how SRN is performing slightly better than DCGAN thoughts most of the epochs. Comparing the plotted test scores shows that SRN is making the learning process more difficult for the Discriminator. This may indicate a stronger Generator.

CIFAR-10

CIFAR-10 trained with SRN performed worse overall, but won more local victories. Models trained using the CIFAR-10 dataset usually varies immensely and any technique should make more an impact on the SGAM ratios. The local scores calculated through SGAM are plotted in Figure 7.3 and shows the difficulty DCGAN faces when training on the CIFAR-10 dataset. Neither models are consistently able to classify samples from the other model during the first 20 epochs. The large increase in accuracy occurs between epoch 15 and 25. The sample scores do vary quite a bit, but are overall comparable. Interestingly, SRN appears to have created a less overfitted Discriminator. Comparing the test scores shows that SRN has overall worse accuracy and less renown peaks in performance. This might indicate a better balance between the Discriminator and Generator.

Table 7.1.: Global result from SGAM comparison between regular DCGAN and DCGAN trained with Static Reusable Noise (SRN) on the three datasets. The parenthesis indicate whether the models were tested by generating samples with random generated or the static noise used to train the model.

Dataset	Epochs	GAN 1	GAN 2	r_{test}	r_{sample}	Winner
CelebA	16	DCGAN	$\text{SRN}_{Z=random}$	0.9867	0.9981	$\text{SRN}_{Z=random}$
	16	DCGAN	$\text{SRN}_{Z=training}$	0.9875	0.9984	$\text{SRN}_{Z=training}$
IKEA	22	DCGAN	$\text{SRN}_{Z=random}$	0.9981	4.5108	DCGAN
	22	DCGAN	$\text{SRN}_{Z=training}$	0.9980	4.5106	DCGAN
CIFAR-10	50	DCGAN	$\text{SRN}_{Z=random}$	1.0026	1.0571	DCGAN
	50	DCGAN	$\text{SRN}_{Z=training}$	1.0019	1.0556	DCGAN

IKEA

SRN did not perform well when trained on the IKEA dataset. Figure 7.4 shows the scores plotted. The Discriminator appears to be too powerful and and is quickly unable to correctly classify the samples generated from regular DCGAN. This is noticeable by its sample score collapsing at epoch three and five.

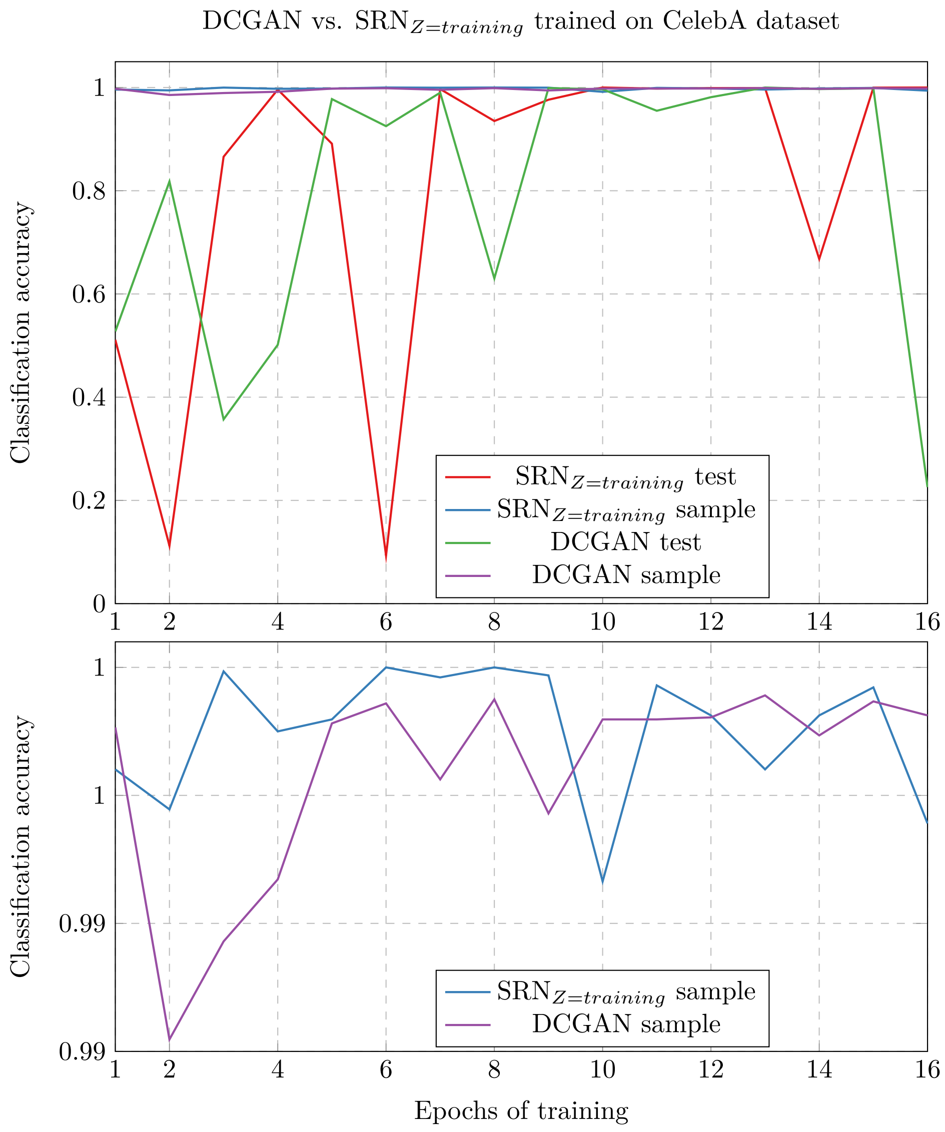

Figure 7.2.: SGAM scores for DCGAN versus SRN trained on the CelebA dataset. Looking at the bottom plot, SRN is performing slightly better than DCGAN thoughts most of the epochs. Comparing the test accuracy in the second plot show that the scores do vary quite a bit. SRN has more fluctuations while DCGAN is more stable.

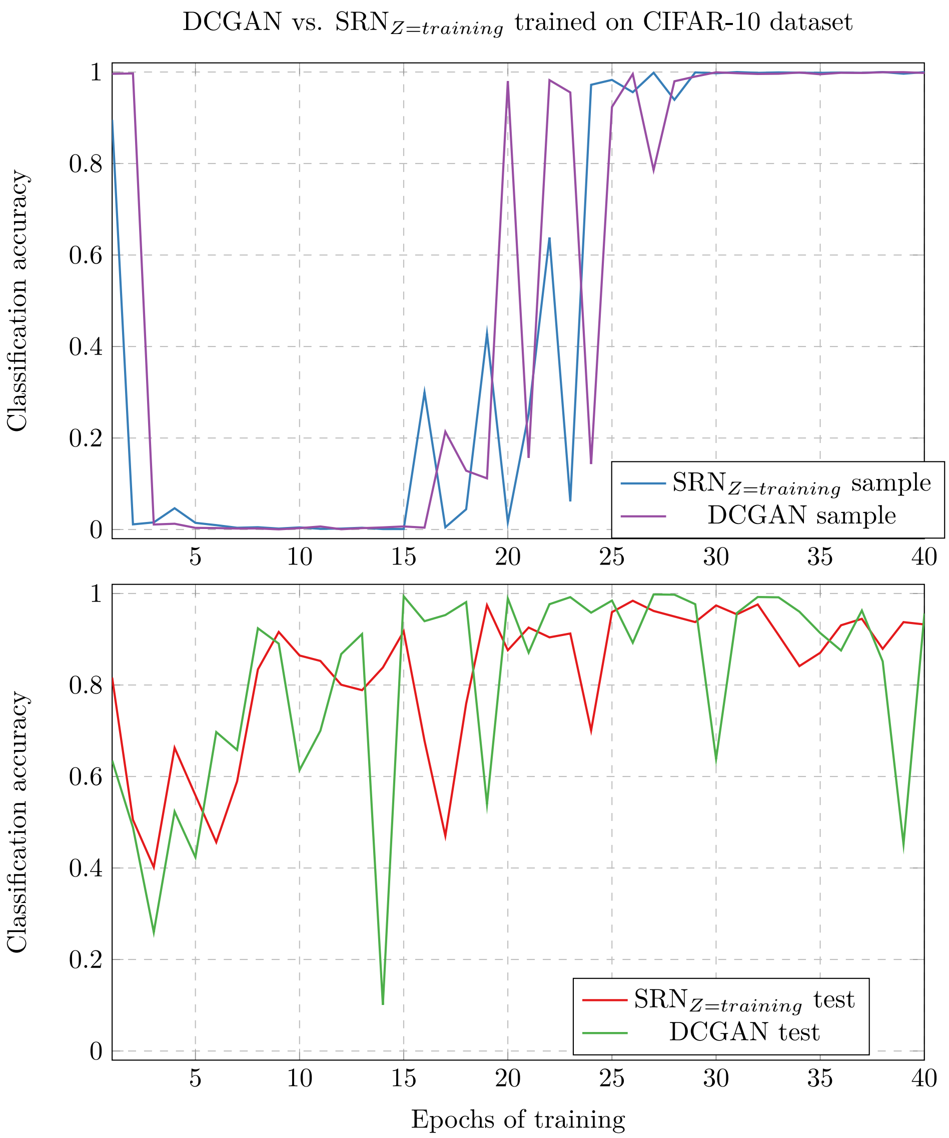

Figure 7.3.: The plots shows the performance of regular DCGAN and DCGAN with SRN trained on the CIFAR-10 dataset. Performance is overall comparable. Comparison of the sample score show SRN is limiting the model during epochs 15 to 25.

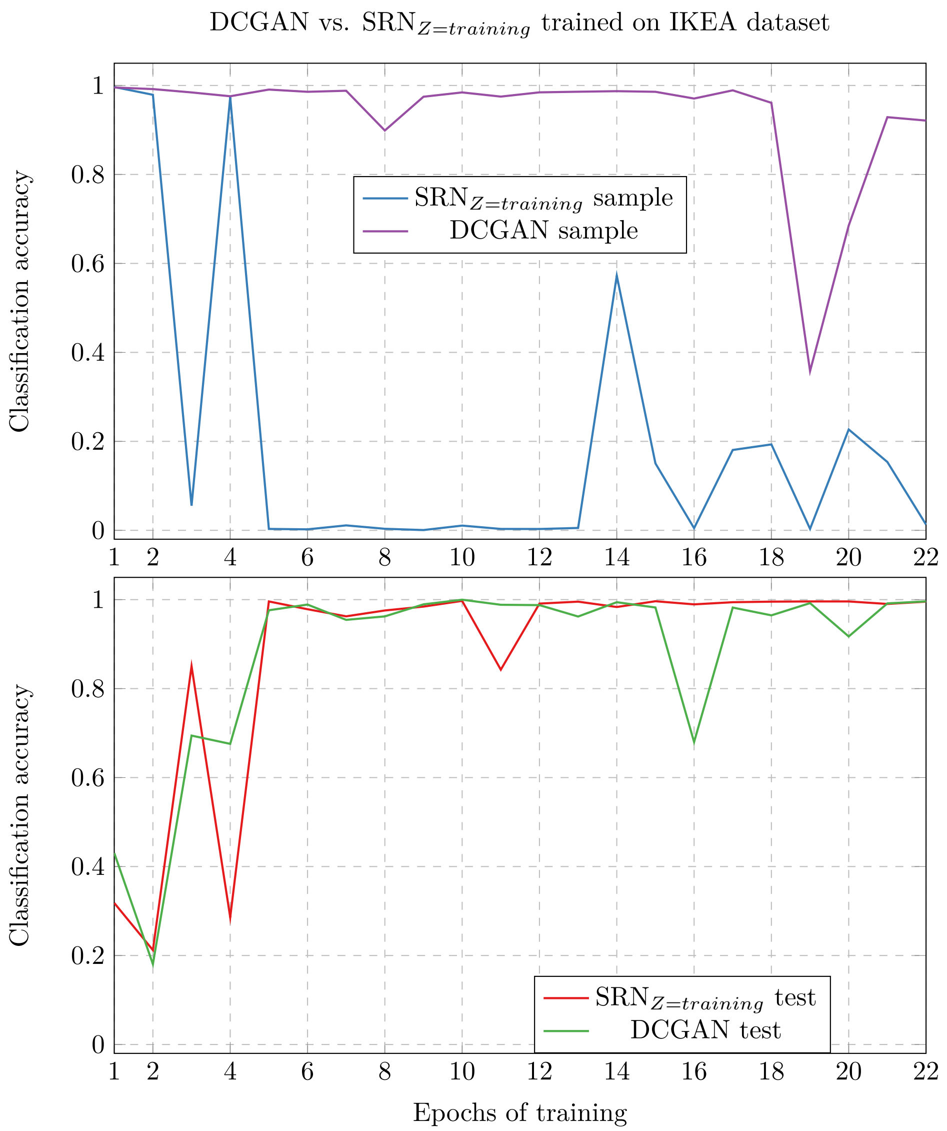

Figure 7.4.: SGAM scores for DCGAN versus DCGAN with Static Reusable Noise (SRN) trained on the IKEA dataset. Model is clearly impacted negatively by SRN as its sample scores collapses after 5 epochs. SRNs ability to classify test data have more variation during the first epochs, but are overall comparable to DCGAN.

Table 7.2.: Local results from SGAM comparison between regular DCGAN and DCGAN trained with Static Reusable Noise (SRN) on the three datasets. The variable Z denotes which input noise was used by the generative model to generate samples for the other model to judge.

Dataset	Epochs	GAN 1	GAN 2	Ties	Wins$_{GAN_1}$	Wins$_{GAN_2}$
CelebA	16	DCGAN	SRN$_{Z=random}$	6	3	7
	16	DCGAN	SRN$_{Z=training}$	6	3	7
IKEA	22	DCGAN	SRN$_{Z=random}$	1	20	1
	22	DCGAN	SRN$_{Z=training}$	1	20	1
CIFAR-10	50	DCGAN	SRN$_{Z=random}$	11	14	25
	50	DCGAN	SRN$_{Z=training}$	14	11	25

7.2.2. Image based noise generation

The results of comparing models trained with IBNG with regular DCGAN are shown in Table 7.1 and Table 7.4. The results show that IBNG created a worse model that was more overfitted to the training data. Each model was compared with both random noise and noise generated with IBNG. Of the two, the familiar noise of IBNG performed slightly better, but still worse than regular DCGAN.

CelebA

Using the CelebA dataset with IBNG resulted in a worse model. Comparison to DCGAN is shown in Figure 7.5. IBNG is clearly limiting performance. The test scores do fluctuate quite a bit, but are comparable.

CIFAR-10

CIFAR-10 has widely different images and as a result the noise samples are mostly different as well. Using IBNG is essentially the same as SRN. Figure 7.6 shows that both models perform about equal.

IKEA

The most interesting result was with models trained on the IKEA dataset, Figure 7.7. As with SRN, the sample score does eventually collapse. Images in IKEA dataset have much common with each other. Using IBNG, this aspect is helpful during the early epochs of training, but a curse later. During the first six epochs,

Table 7.3.: Global result from SGAM comparison between regular DCGAN and DCGAN trained with Image Based Noise Generation (IBNG) on the three datasets. IBNG did not improve performance on any dataset. This is indicated by every r_{sample} being greater than one.

Dataset	Epochs	GAN 1	GAN 2	r_{test}	r_{sample}	Winner
CelebA	18	DCGAN	IBNG$_{Z=random}$	0.9762	1.0101	DCGAN
	18	DCGAN	IBNG$_{Z=training}$	0.9742	1.010	DCGAN
IKEA	20	DCGAN	IBNG$_{Z=random}$	0.9656	1.2667	DCGAN
	20	DCGAN	IBNG$_{Z=training}$	0.9597	1.2203	DCGAN
CIFAR-10	25	DCGAN	IBNG$_{Z=training}$	0.9437	1.0473	DCGAN
	50	DCGAN	IBNG$_{Z=random}$	0.9888	1.0239	DCGAN
	50	DCGAN	IBNG$_{Z=training}$	0.9872	1.0083	DCGAN

the samples generated from the model trained using IBNG is superior and able to correctly classify every sample from the regular DCGAN. The performance of both models begins to align after about four epochs of training. Regular DCGAN is able to stay at this performance level, but after ten epochs, IBNG begins to fail at classifying the images from regular DCGAN. The model begins to collapse as the Discriminator become too strong for the Generator. At the same time, regular DCGAN has constantly improved and performing. The peak of both sample scores at epoch 19 is interesting. The likely cause is that the Generator trained IBNG are generating images of so poor quality regular DCGAN fails to accurately classify them.

7.2.3. Audition Based Noise Selection

The results of the models trained using Audition Based Noise Selection (ABNS) are shown in Table 7.5 and Table 7.6. The usual problems observed with IKEA dataset occurred with ABNS as well. Otherwise, ABNS improved perfomance somewhat for CelebA and considerably for CIFAR-10.

ABNS made the already computationally expensive training process even worse. This limited the number of experiments conducted for CelebA and IKEA datasets. CIFAR-10 requires considerably less time and using this dataset was therefore prioritized.

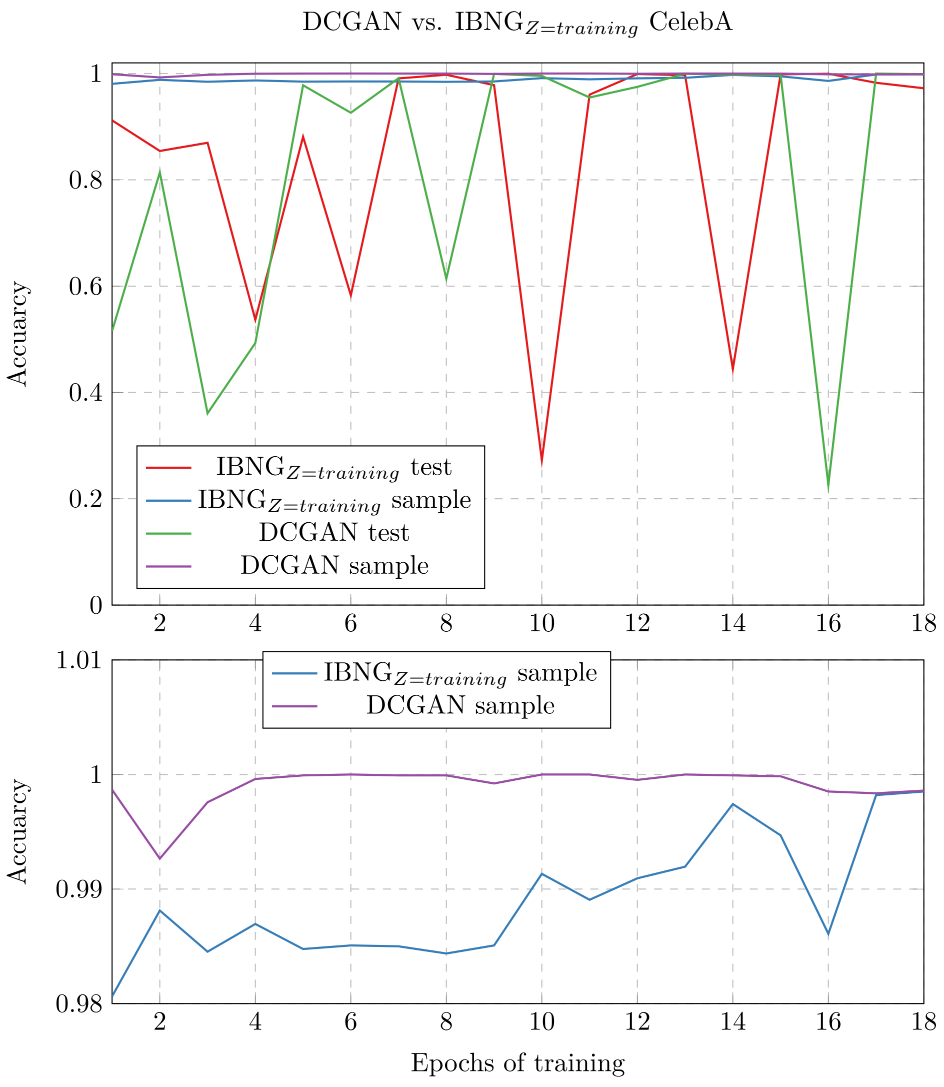

Figure 7.5.: Performance of regular DCGAN and DCGAN with IBNG trained on the CelebA dataset. Comparing the samples scores show DCGAN being the best model. IBNG isclearly limiting perfomance. The test scores do fluctuate quite a bit, but are comparable.

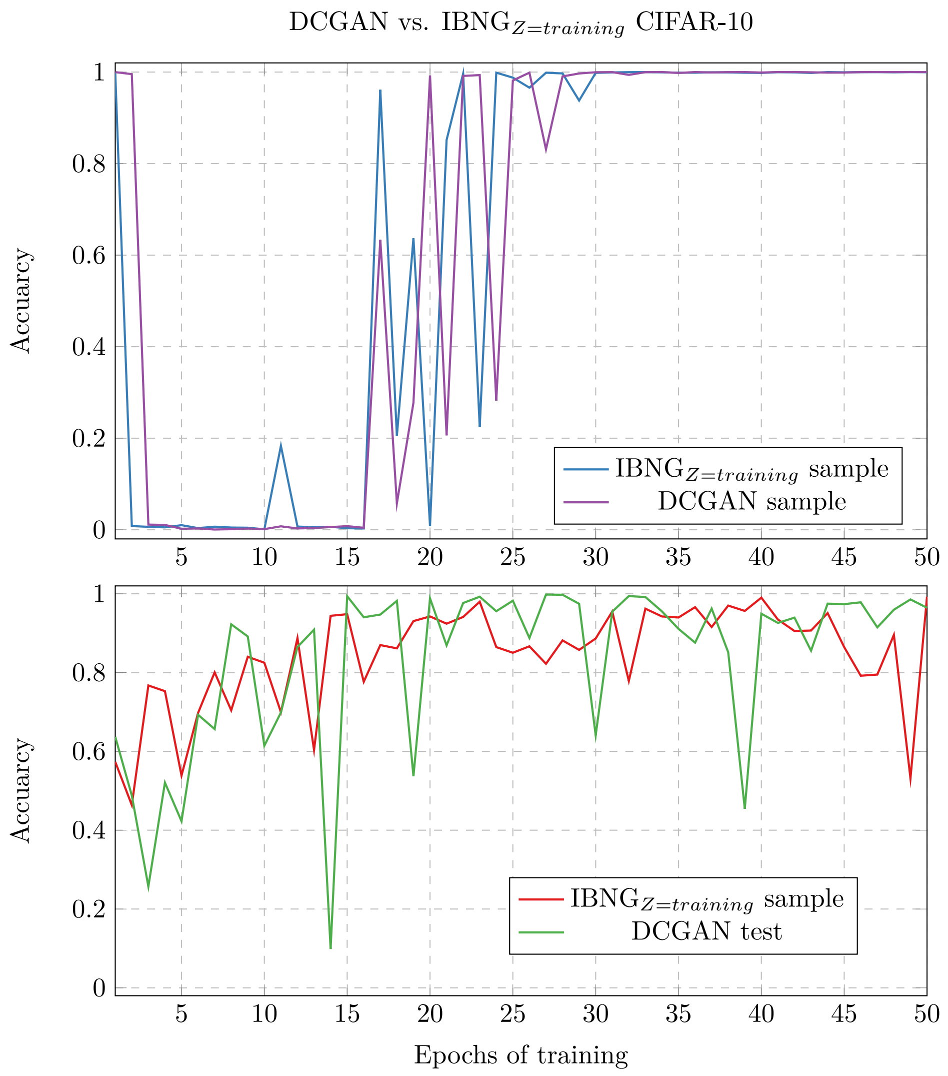

Figure 7.6.: Performance of regular DCGAN and DCGAN with IBNG trained on the CIFAR-10 dataset. Comapring the samples score show that the models are comparable. The model trained with IBNG do have more distinct peaks. The test scores show IBNG creates more stable and less overfitted Discirmantor compared to DCGAN.

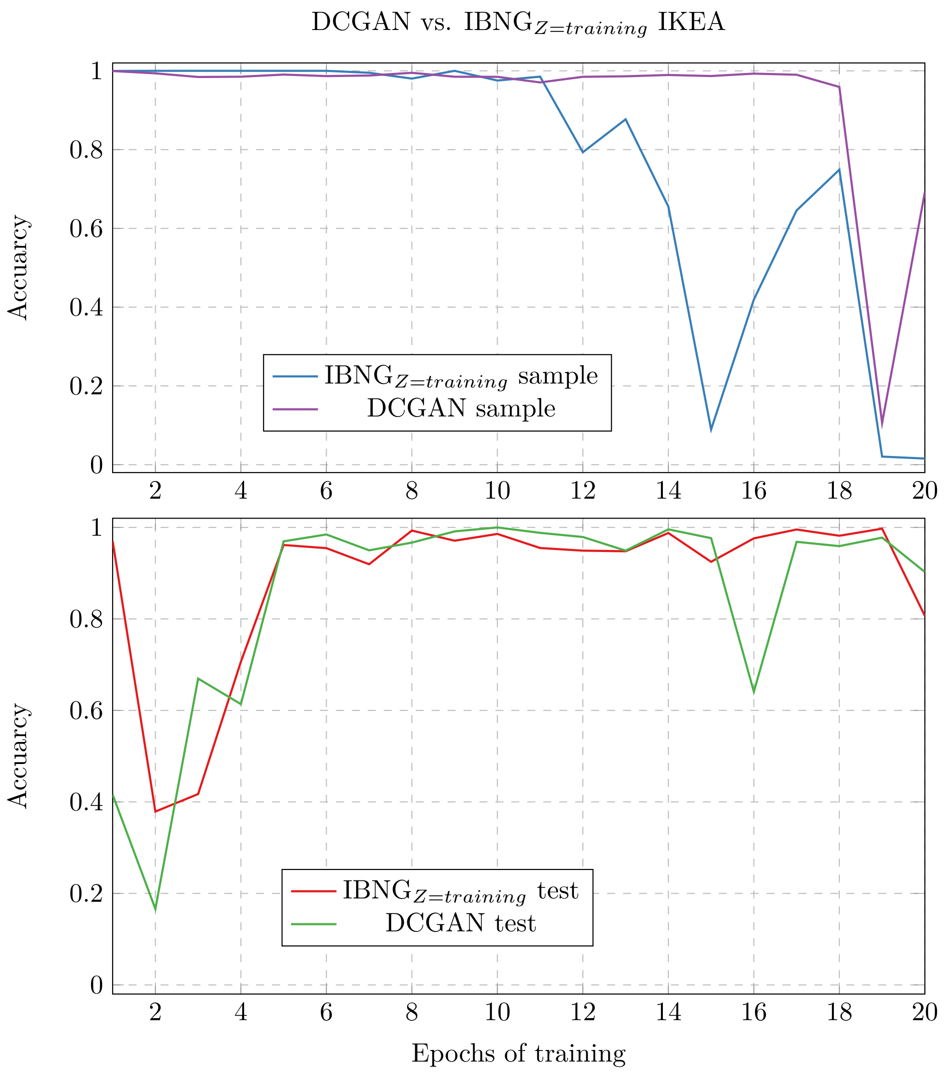

Figure 7.7.: Performance of regular DCGAN and DCGAN with IBNG trained on the IKEA dataset. Model trained with IBNG is comparable DCGAN during the first 11 epochs, but eventually fails to classify generated samples. Such as collapse indicates an overpowered Discriminator.

Table 7.4.: Local results from SGAM comparison between regular DCGAN and DCGAN trained with Image Based Noise Generation (IBNG) on the three datasets. The variable Z denotes which input noise was used by the generative model to generate samples for the other model to judge. Z is either generated at *random*, thereby most likely not observed during training, or identical to the Z nose used during *training*

Dataset	Epochs	GAN 1	GAN 2	Ties	Wins_{GAN_1}	Wins_{GAN_2}
CelebA	18	DCGAN	$\text{IBNG}_{Z=random}$	6	11	1
	18	DCGAN	$\text{IBNG}_{Z=training}$	7	11	0
IKEA	20	DCGAN	$\text{IBNG}_{Z=random}$	3	10	7
	20	DCGAN	$\text{IBNG}_{Z=training}$	3	11	6
CIFAR-10	25	DCGAN	$\text{IBNG}_{Z=training}$	3	8	0
	50	DCGAN	$\text{IBNG}_{Z=random}$	15	15	20
	50	DCGAN	$\text{IBNG}_{Z=training}$	12	18	20

CelebA

ABNS was able to improve performance slightly when trained on the CelebA dataset. Both mixed and best mode won the global victory, but the local victories were more divided. Figure 7.8 plots the results of the SGAM comparison between ABNS in best mode and Figure 7.9 shows the result of mixed mode. Both were trained with auditions size of 3 times the batch size of 64. ABNS-mixed$_3$ is more stable overall but ABNS-best$_3$) is noticeable better during the first six epochs. ABNS-mixed$_3$ is still performing better than DCGAN during the early epochs. The results after this point are more varied with regular DCGAN often being the best performing model.

CIFAR-10

Training on CIFAR-10 was greatly improved with ABNS. The table shows that increasing auditions size does incrase performance. All variations of ABNS severely outperforms regular DCGAN. The results from SGAM when comparing best performing model, ABNS-mixed$_6$ against regular DCGAN are plotted in Figure 7.10. Comparing samples scores reveal ABNS-mixed$_6$ to severely outperform DCGAN up until epoch 30. At this point DCGAN is slightly outperforming ABNS-mixed$_6$. Figure 7.11 shows randomly sampled output from both models at certain time during training in correlation with Figure 7.10. The samples generated by ABNS-mixed$_6$ do appear to be more varied and have better details.

Table 7.5.: Global result from SGAM comparison between regular DCGAN and DCGAN trained with Audition Based Noise Selection (ABNS) on the three datasets. The modes of ABNS are *best* which only selected the best rated images and *mixed* which return a selection of various qualities. The number indicates the *audition size*, how many samples were analyzed.

Dataset	Epochs	GAN 1	GAN 2	r_{test}	r_{sample}	Winner
CelebA	18	DCGAN	ABNS-best$_3$	1.0815	0.9989	ABNS-best$_3$
	15	DCGAN	ABNS-mixed$_3$	1.0690	0.9987	ABNS-mixed$_3$
IKEA	8	DCGAN	ABNS-mixed$_3$	0.8715	1.7889	DCGAN
	14	DCGAN	ABNS-mixed$_6$	0.9630	2.0341	DCGAN
CIFAR-10	40	DCGAN	ABNS-best$_3$	1.0053	0.9129	ABNS-best$_3$
	40	DCGAN	ABNS-best$_6$	1.0517	0.8508	ABNS-best$_6$
	50	DCGAN	ABNS-mixed$_3$	1.0054	0.7805	ABNS-mixed$_3$
	40	DCGAN	ABNS-mixed$_6$	0.9683	0.743	ABNS-mixed$_6$
	40	ABNS-mixed$_6$	ABNS-mixed$_3$	0.9215	0.9059	ABNS-mixed$_3$
	40	ABNS-best$_6$	ABNS-mixed$_6$	0.9215	0.9059	ABNS-mixed$_6$

IKEA

Using ABNS with the IKEA dataset once again caused the Discriminator to become overpowered. Neither of the two modes nor increasing the auditions size was able to prevent the collapse. Figure 7.12 plots the results of ABNS in mixed mode with auditions size of 3. The collapse is clearly visible at epoch four. . The Generator does make some strong attempts a recuperating between epoch 6 and 10, but the strong Discriminator eventually regains total control. Inspecting both the off balanced test ratio in Figure 7.5 and the plotted test score in Figure 7.12 does indicate the ABNS-mixed$_6$ made the Discriminator more overfitted towards the training data.

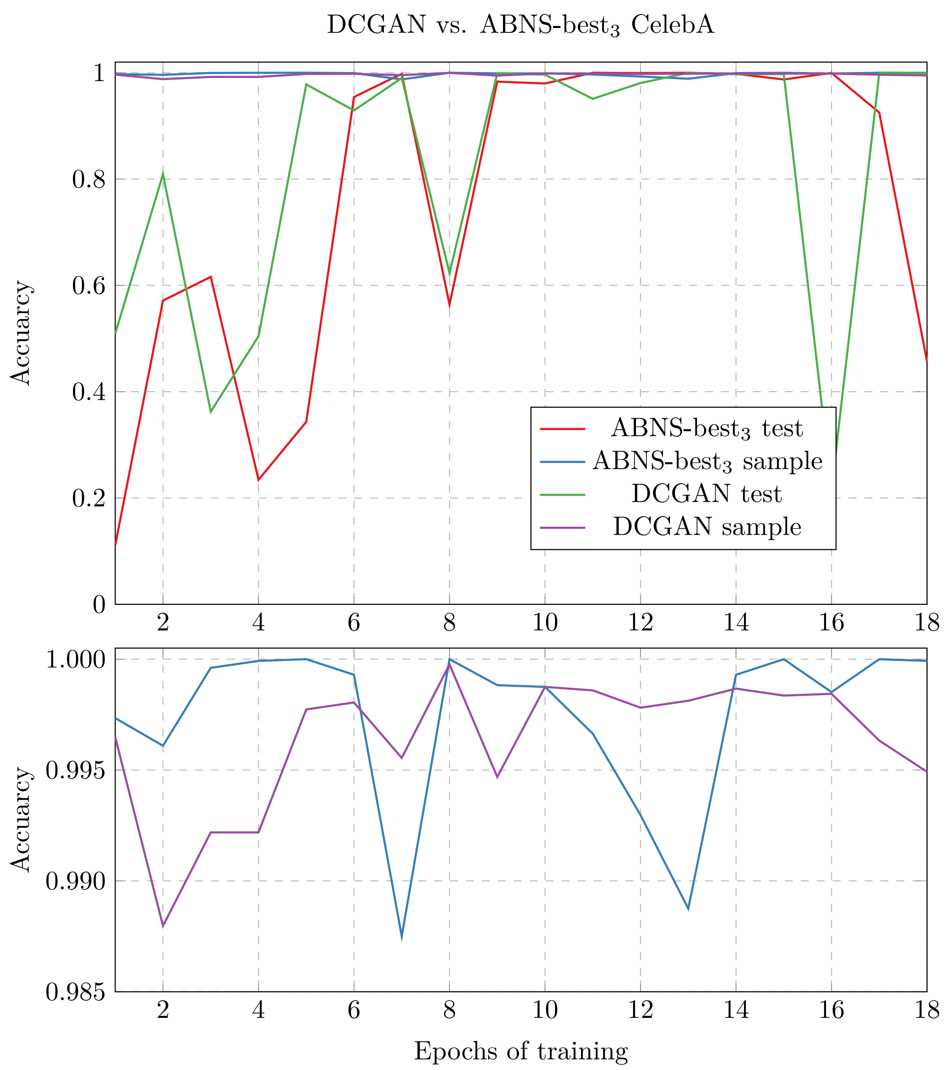

Figure 7.8.: Comparison between DCGAN and ABNS-best$_3$ trained on the CelebA dataset. ABNS-best$_3$ is successfully guiding the model during the first six epochs of training but are otherwise more unstable the regular DCGAN. Performance on test data is however more stable.

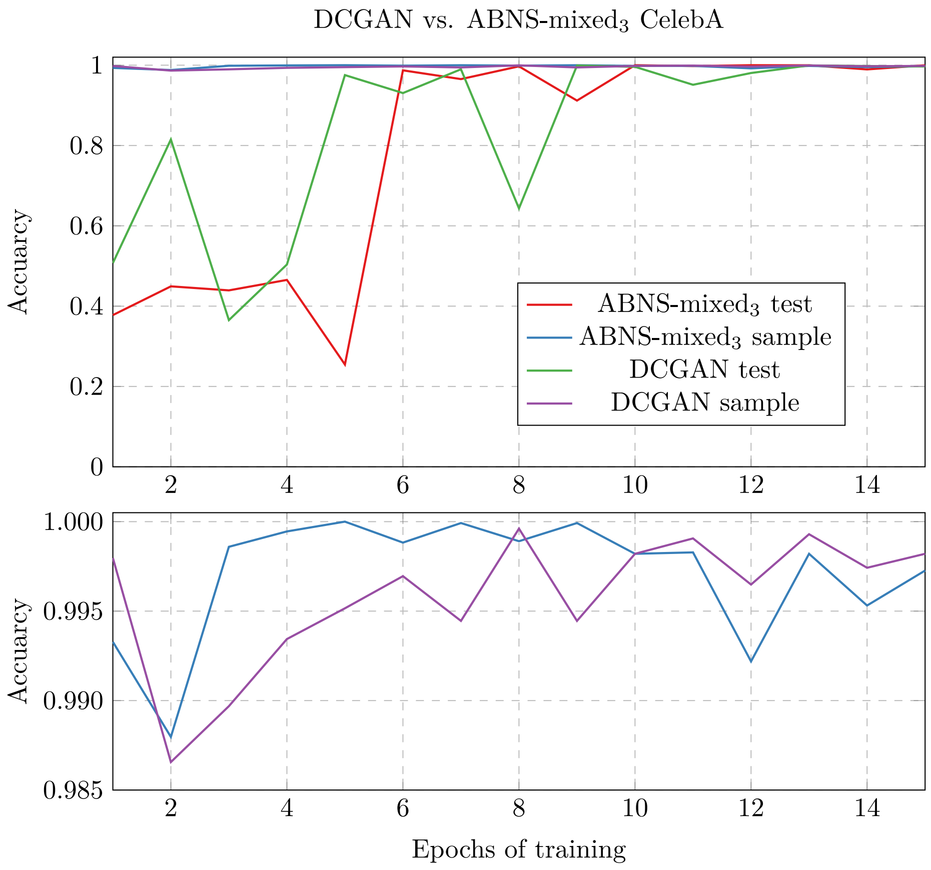

Figure 7.9.: Comparison between DCGAN and ABNS-mixed$_3$ trained on the CelebA dataset. ABNS-mixed$_3$ is better during the first 10 epochs as seen by its samples score being higher than DCGANs'. Another interesting observation is that the Discriminator trained with ABNS-mixed$_3$ is performing worse than DCGAN at classifying samples from the training set. This is an indication that ABNS-mixed$_3$ creates a stronger Generator that puts up more a challenge for the Discriminator. The tenth epoch is when the two models performance begins to align and coincidentally is when the tests score of model ABNS-mixed$_3$ begins to converge. The Discriminator of DCGAN has at this point not converged and is now perfoing better then ABNS-mixed$_3$.

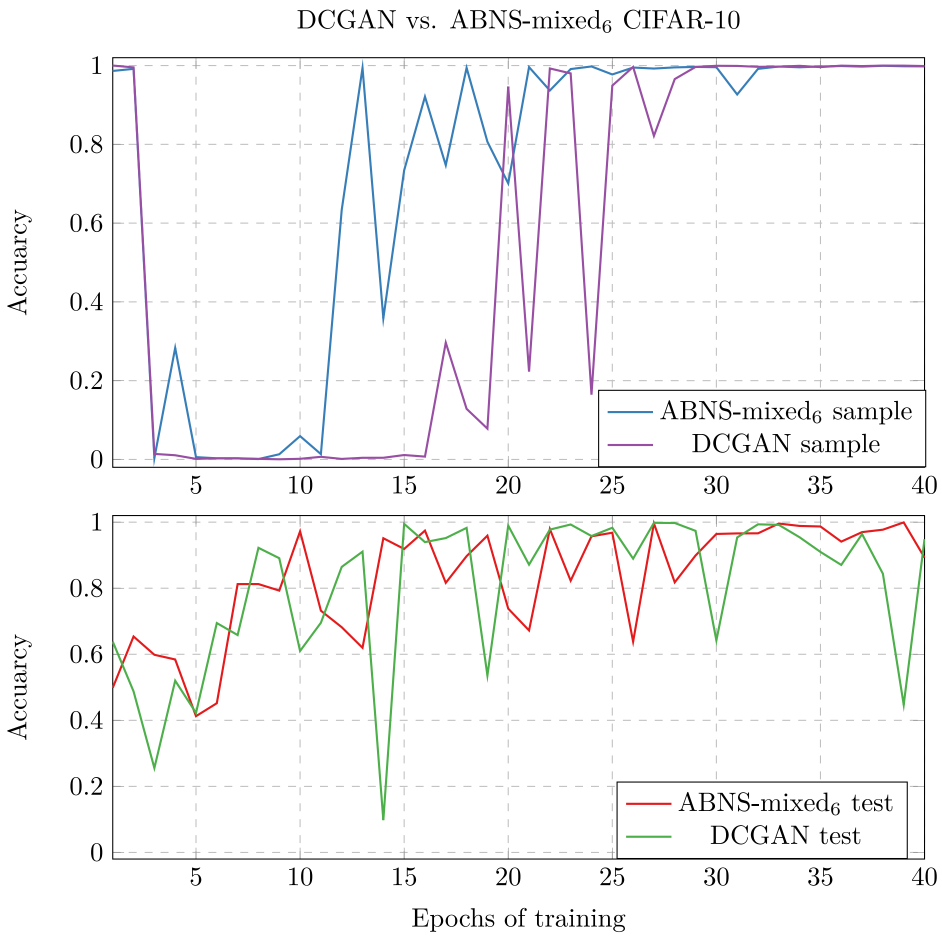

Figure 7.10.: The plots shows the comparison between DCGAN and ABNS-mixed$_6$ trained on CIFAR-10 datasets. The model trained with ABNS-mixed$_6$ is greatly outperforming DCGAN until epoch 29.At this point, the models are comparable with regular DCGAN being slightly better.The test scores are comparable although DCGAN has larger drops inaccuracy. Figure 7.11 displays examples of generated samples at certain parts of the training.

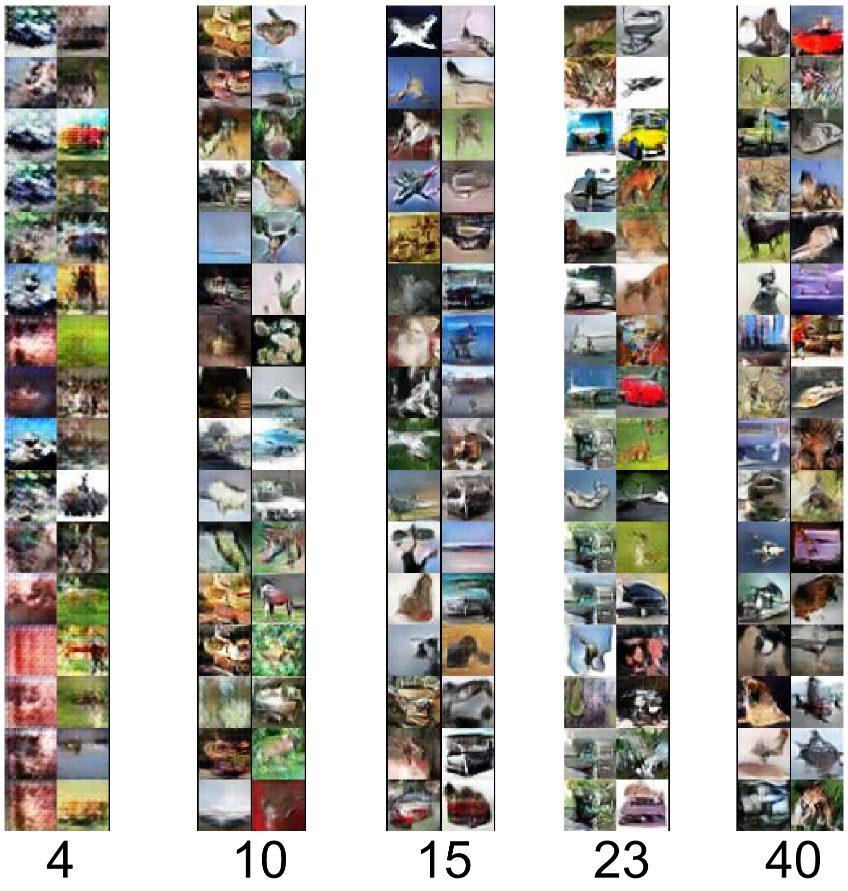

Figure 7.11.: Randomly selected outputs from models from DCGAN(Left columns) and DCGAn trained with ABNG-mixed$_6$ (Right columns) on the CIFAR-10 dataset.Reference. The numbers indicate the number of epochs trained and correltases with events in Figure 7.10

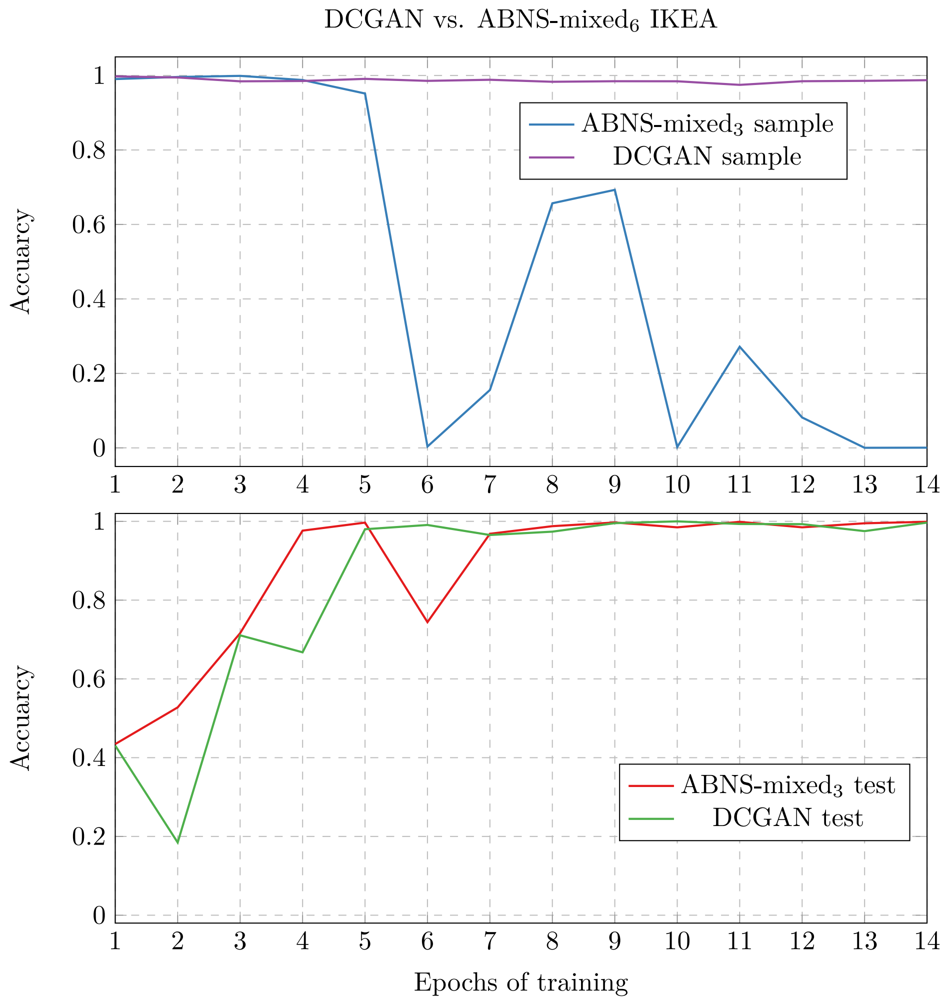

Figure 7.12.: The plots shows the performance of regular DCGAN versus ABNS-mixed$_3$ trained on the IKEA dataset. ABNS-mixed$_3$ performs well during the first 4 epochs. ABNS-mixed$_3$ appears to create a stronger Discriminator as seen by the the test score being noticable higher during he first four epochs comapred to regulr DCGAN. During this periode ABNS-mixed$_3$ does perform better however this does not last. After five epochs, the usual collapse of the Generator occurs. It is believed that the Discriminator becomes to strong and outpower the Generator.

Table 7.6.: Global result from SGAM comparison between regular DCGAN and DCGAN trained with Audition Based Noise Selection (ABNS) on the three datasets. The modes of ABNS are *best* which only selected the best rated images and *mixed* which return a selection of various qualities. The number indicates the *audition size*, how many samples were analyzed.

Dataset	Epochs	GAN 1	GAN 2	Ties	Wins_{GAN_1}	Wins_{GAN_2}
CelebA	18	DCGAN	ABNS-best_3	7	4	7
	15	DCGAN	ABNS-mixed_3	3	7	5
IKEA	8	DCGAN	ABNS-mixed_3	2	4	2
	14	DCGAN	ABNS-mixed_6	1	11	2
CIFAR-10	40	DCGAN	ABNS-best_3	7	13	20
	40	DCGAN	ABNS-best_6	11	7	22
	50	DCGAN	ABNS-mixed_3	13	4	33
	40	DCGAN	ABNS-mixed_6	7	14	19
	40	ABNS-mixed_6	ABNS-mixed_3	4	8	28
	40	ABNS-best_6	ABNS-mixed_6	6	17	17

7.2.4. Generative Multi-Adversarial Network with Historic Discriminator

Generative Multi-Adversarial Network with Historic Discriminator (GMAN-HD) performed well against regular DCGAN when train on CelebA and CIFAR-10. GMAN-HD was also tested against models trained using ABNG. The results are shown in Table 7.7 and Table 7.8. Note that only the main Discriminator was used for SGAM comparisons.

IKEA dataset

Historic Discriminator did not perform well on IKEA dataset. The Discriminator collapses after just two epochs of training. Still, the Generator and second Discriminator does make some attempts at increasing performance. The sample accuracy begins to rise at the eleventh epoch. The tests scores are similar. DCGANs test score have a peak at epoch 16 while GMAN-HDs accuracy is stays close to 100% from epoch 10.

Table 7.7.: Global results from SGAM comparison between regular DCGAN and DCGAN trained with Multi-Adversarial Network with Historic Discriminator on the three datasets.

Dataset	Epochs	GAN 1	GAN 2	r_{test}	r_{sample}	Winner
CelebA	14	DCGAN	GMAN-HD	1.0728	0.9980	GMAN-HD
	14	ABNS-mixed$_3$	GMAN-HD	0.9983	0.9989	GMAN-HD
IKEA	19	DCGAN	GMAN-HD	0.9734	2.9231	DCGAN
	8	ABNS-mixed$_3$	GMAN-HD	1.207	1.7083	ABNS-mixed$_3$
	14	ABNS-mixed$_6$	GMAN-HD	1.0289	1.5507	ABNS-mixed$_6$
CIFAR-10	46	DCGAN	GMAN-HD	1.0058	0.8194	GMAN-HD
	40	GMAN-HD	ABNS-best$_3$	0.9964	1.2095	Histric
	46	GMAN-HD	ABNS-mixed$_3$	0.9984	0.9222	ABNS-mixed$_3$
	40	GMAN-HD	ABNS-mixed$_6$	0.9610	0.9576	ABNS-mixed$_6$

CelebA dataset

The results of training historic Discriminator on CelebA dataset resembles a mix between ABNG in best mode and mixed mode. The sample score is constantly above the score of DCGAN until a small peak at epoch 13.

CIFAR-10 dataset

Results from CIFAR-10 was interesting. The samples scores are almost identical until epoch 10 when HDs score suddenly begins to rise. The score fluctuates for some time before stabilizing at around 99.9% accuracy after 18 epochs. DCGAN is not able to catch up until 10 epochs later and even then, HDs sample accuracy is still higher. The test scores are similar throughout, but HD follows the same patterns as with ABNS, creating weaker but more stable Discriminator.

7.3. Discussion

The results of the experiments have been now presented and will be discussed in this section. It is clear from the experiments that the dataset used for training had a large impact on the results. This discussion will therefore discuss the results overall, Section 7.3.1, and then in relation to each dataset, Sections 7.3.2, 7.3.3 and 7.3.4.

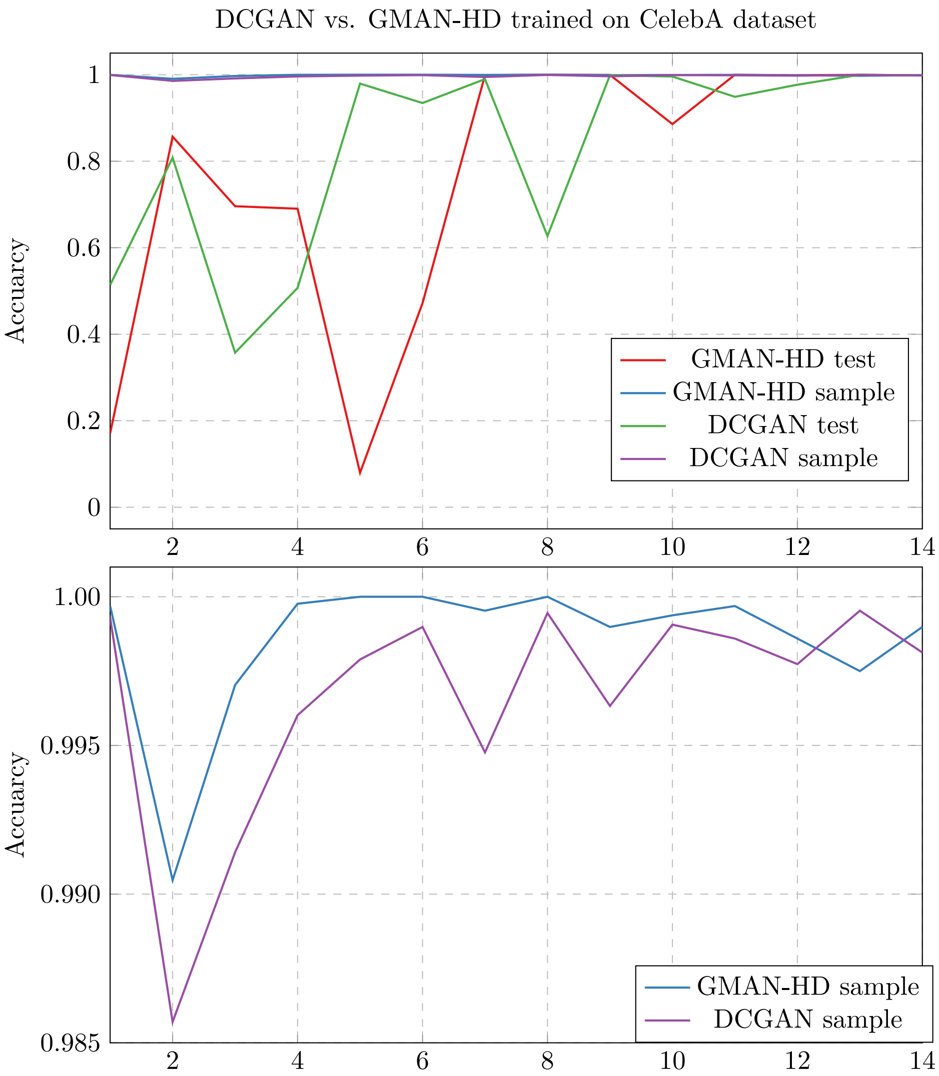

Figure 7.13.: GMAN-HD achieves higher classification accuracy throughout, with the exception during epoch 13. The test score is clearly affected during the first epochs.

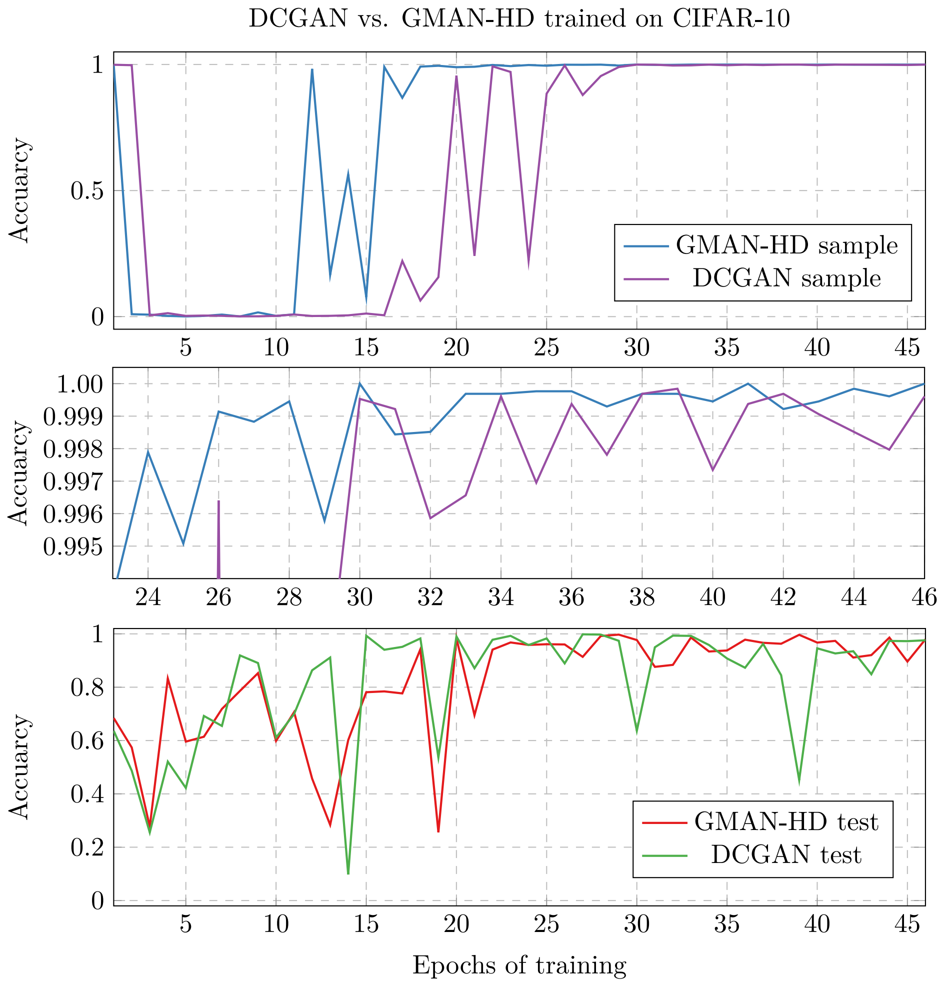

Figure 7.14.: GMAN-HD clearly improves the learning process. Comapring the sample scores show GMAN-HD begins to converge on the dataset after 11 epochs. Regular DCGAN does evuallty reach simialr perfomance, but requires more training. The sampels score after convergence, midle plot, show GMAN-HD to still be perfoming better and also beeing more stable. The test scores are mostly similar througout.

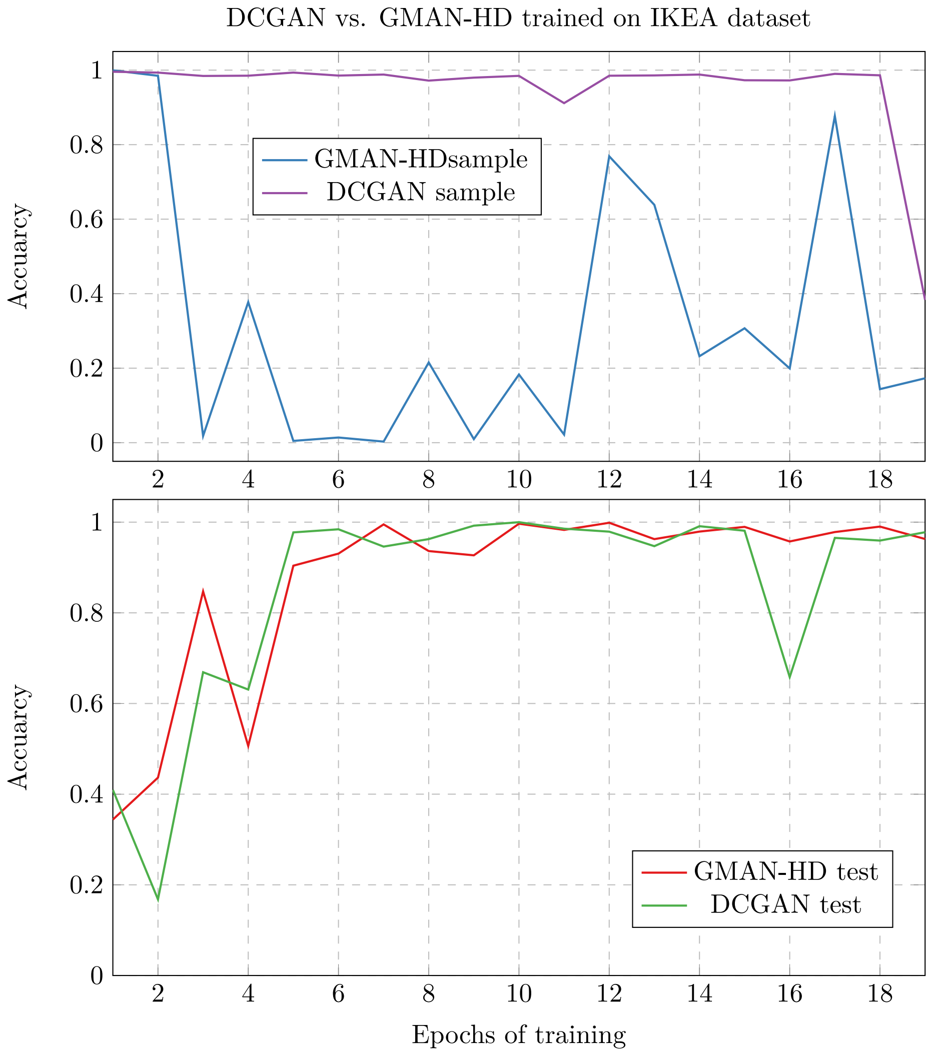

Figure 7.15.: GMAN-HD appears to create a stronger Discirmantor even faster. The sample score of GMAN-HD collapses after only two epochs compared to the usual five. The Generator does make some strong attempts at improving performance as seen by the peaks at epoch 12 and 17.

Table 7.8.: Local results from SGAM comparison between regular DCGAN and DCGAN trained with Multi-Adversarial Network with Historic Discriminator on the three datasets.

Dataset	Epochs	GAN 1	GAN 2	Ties	Wins$_{GAN_1}$	Wins$_{GAN_2}$
CelebA	15	DCGAN	GMAN-HD	4	1	8
	14	ABNS-mixed$_3$	GMAN-HD	7	2	5
IKEA	19	DCGAN	GMAN-HD	1	17	1
	8	ABNS-mixed$_3$	GMAN-HD	4	2	2
	14	ABNS-mixed$_6$	GMAN-HD	1	7	6
CIFAR-10	46	DCGAN	GMAN-HD	12	6	28
	40	GMAN-HD	ABNS-best$_3$	13	18	9
	46	GMAN-HD	ABNS-mixed$_3$	18	16	12
	40	GMAN-HD	ABNS-mixed$_6$	6	25	9

7.3.1. Overall

SRN was a simple technique reused the same noise very epoch. SRN was able to increase accuracy slightly on CelebA and CIFAR-10 during the earlier epochs when most the learning occurs. SRN in its current form is not a recommended technique but does highlight the potential for reducing the range of Generator input to stabilize the learning process.

IBNG was a technique similar to SRN that generated the input noise from images in the training set. The goal behind the technique was to make the learning task easier for the Generator. IBNG had the opposite effect and created a too strong Discriminator that was more overfitted towards the training data and performed slightly worse than regular DCGAN. Although IBNG still cause the model to collapse when trained on the IKEA dataset, the collapse occurred much later in the training process than any of the other techniques.

ABNS was a more advanced and computationally expensive technique that task the Discriminator to pick the most representable images to guide training. Using ABNS when trained on CelebA performed similar to SRN. Experimenting with larger auditions sizes may improve performance further. This is interesting as SRN is a much more simple technique. ABNS was more successful on CIFAR-10, a more difficult dataset. All variations of ABNS were able to drastically increase performance. Testing on CIFAR-10 saw an increase in performance with an increase in auditions size.

The final proposed technique added an additional Discriminator to the DCGAN architecture. This GMAN-HD Discriminator was trained on past, well rated images from the Generator. While requiring less training time, GMAN-HD performed better than the other techniques on the CelebA dataset, winning both globally and almost all local victories. HD did also perform well on CIFAR-10 dataset, but was beat by some variations of ABNS.

While SRN and IBNG did perform well on under certain conditions, overall, the techniques are too simple to warrant further use. Both of these technique were proposed to help the Generator during training by putting limitations on its input. This did to a lesser degree stabilized training during the early epochs, but usually ended up limiting the model overall. ABNS had better success at creating a stronger Generator as it does not limit the available range of the input. ABNS showed that the learning process could be improved by maximize learning each iteration. GMAN-HD achieved similar results on CIFAR-10 and CelebA. The models trained with GMAN-HD was not as powerful as ABNS with larger auditions sizes, but proved to be more stable against unaltered DCGAN.

7.3.2. CelebA dataset

The CelebA dataset rated as an easy dataset for use with GAN, Section refsec:publicdatasetsDisucsusion. The dataset has a large number of images with common features of a face centrally aligned in frame. Regular DCGAN is more than able to learn these features through unsupervised learning. Figure 7.16 plots the classification accuracy for the Discriminator model in regular DCGAN when training on CelebA dataset. The accuracy converges after about four epochs of training. Being an easy dataset made the proposed techniques have less of impact on performance. Although small, three of techniques were able to consistently improve performance on CelebA. The largest improvements were seen during the early of epochs of training.

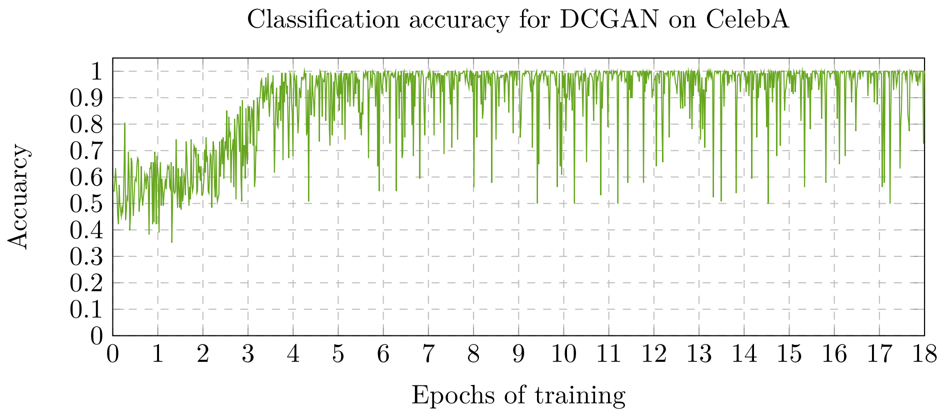

Figure 7.16.: Classification accuracy for training data and samples from the generative model for regular DCGAN when trained on the CelebA dataset. The accuracy converges after 5 epochs of training.

7.3.3. CIFAR-10 dataset

CIFAR-10 was the most convenient dataset used for the experiments. The dataset has fewer images and lower resolution which requires less training time then the other two datasets. CIFAR-10 was rated as a hard dataset in the previous chapter and this proved accurate. Figure 7.17 shows how DCGAN struggles during the early training phase and requires several dozen epochs of training before be able to accurately classifying generated and real images correctly. With CIFAR-10 being a harder dataset, some of the suggested techniques were better able to increase performance when using it as training data. The simpler techniques of SRN and IBNG did limit the performance somewhat. The more advanced techniques of ABNG and MGAN with historic Discriminator were more successful.

7.3.4. IKEA dataset

The IKEA dataset was created during the pre-study. The dataset was rated at medium difficult in the previous chapter, but using it for this experiments seems to indicate it is a harder dataset than first anticipated. Regular DCGAN is able to capture the white background and overall create quite sharp images. The images do however never reach a point where they appear natural. This may be caused the images having few consistent features among them except for the white background. The dataset is also being artificially augmented to increase its size. For this dataset, each original image was copied and slightly adjust 100

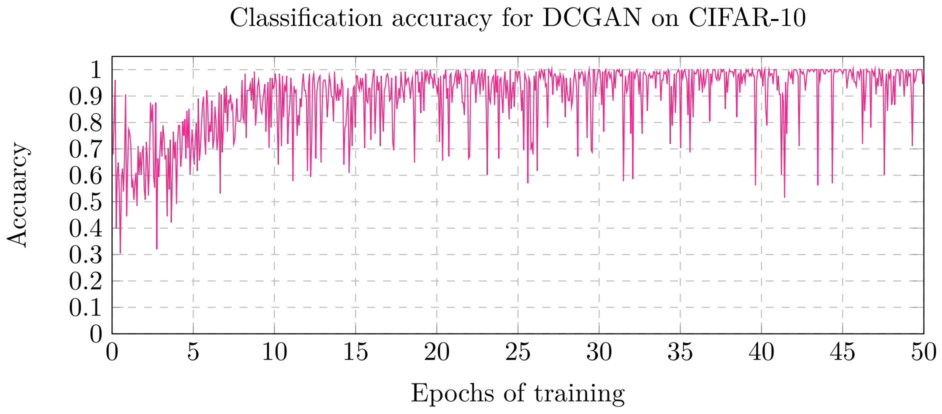

Figure 7.17.: Classification accuracy for training data and samples from the generative model for regular DCGAN when trained on the CIFAR-10 dataset. CIFAR-10 is a hard dataset which is noticeable by the accuracy requiring about 20 epochs of training before the peaks consistently reach close to 100% accuracy.

times which may impact learning in some way. Figure 7.19 plots the classification accuracy of the Discriminator when regular DCGAN is trained on the dataset. Notice that the DCGAN requires about five to seven epochs of training before reaching 100% accuracy.

None of the proposed techniques were able to consistently improve the models trained on the IKEA dataset. Every attempt ended with the models sample score eventually collapsing. The collapse occurs after about five to seven epochs which correlates with Discriminator performance of Figure 7.19. This collapse was surprising and conflicting with the results of the other datasets. Interestingly, it was only the sample score of the afflicted model that collapse, the test score was usually high. The sample score of the other model, evaluating the generated images from the afflicted model, was usually above 90%.

Figure 7.18 shows samples generated with ABNS and shows samples generated before and after the collapse. The number next to each sample represent the rating of the models Discriminator. Rating above 50 is believed to be real images and rating under is believed to be generated. At epoch 4 the performance of the Discriminator and generates is well balanced. The Generator is able generated decent samples that often fool the Discriminator. These images provide valuable feedback on how to improve further. At epoch six, the Discriminator has become too strong as seen by all rating being below 50 and most even being below 30.

It should be noted that this sort of collapse does occur on occasion with other datasets, but that is only between a single iteration. A strong Discriminator halts the learning of the Generator as none of its improvements is able to fool the Generator. It is reasonable to believe a strong Discriminator should result in high sample score. In fact, the opposite is true. The Discriminator is only taught how generated samples should look from its own Generator. A strong Discriminator leads to a poor Generator which produces low quality samples. The Discriminator is taught that these low quality samples represents all generated samples. This is of course not the case when shown samples from another model with better balanced models. These samples will have much higher quality than those of its own Generator and thus believe they to be real. This causes the collapse.

The analysis of the results seems to indicate an unbalanced and overpowered Discriminator to be the root of the problem. Training on the IKEA dataset is a delicate process. The images in the IKEA dataset appears to be easier to learn for the Discriminator then the Generator. All the suggested techniques ended up make the task easier for the Discriminator. After about five epochs of training, the Discriminator has become so skilled at its task of separating training data from generated samples the Generator is unable to progress.

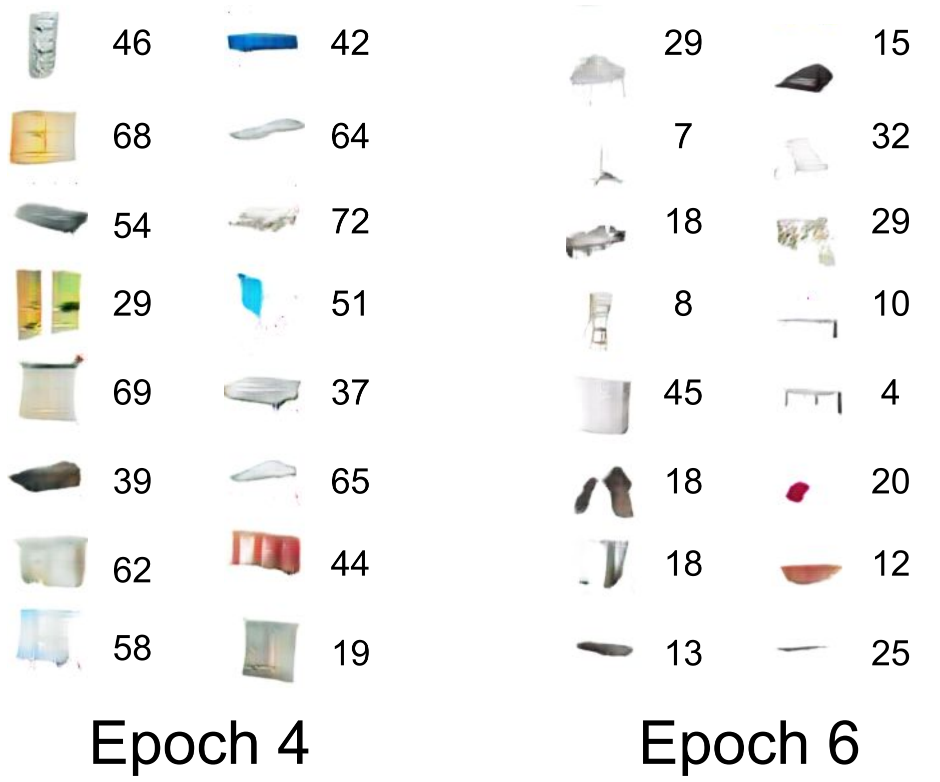

Figure 7.18.: The figure shows the problem of unbalanced GAN model with where the discriminative model is outperforming the generative model. The samples are generated by DCGAN with ABNS-mixed$_3$ on the IKEA dataset during epoch 4 and 6. The number next to each sample represent the rating of the models Discriminator. Rating above 50 is believed to real images and rating under is believed to be generated. This is the same model compared in Figure 7.12 and showcases the model at its peak, epoch 4, and bottom, epoch 5. During epoch 4, the performance of the Discriminator and generates is well balanced. The Generator is able generated decent samples that often fool the Discriminator. These images provide valuable feedback on how to improve further. At epoch 6 however, the Discriminator has become too strong for the Generator to progress.

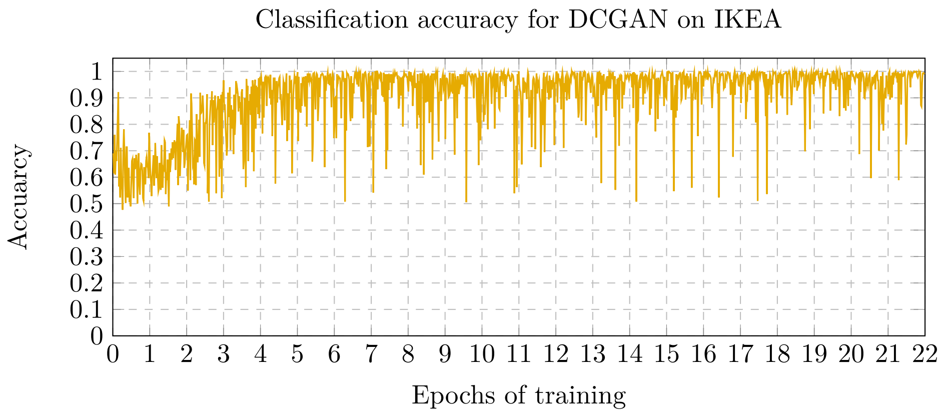

Figure 7.19.: Classification accuracy for training data and samples from the generative model for regular DCGAN when trained on the IKEA dataset. The model requires 5 epochs of training before the peaks begin to reach 100% accuracy.

8. Conclusion and Future Work

Section 1.2 defined one Project Goal and three Research Questions (RQ). This chapter will answer the RQs and discuss to what degree the Project Goal have been reached, Section 8.1. Section 8.2 will define future work.

8.1. Conclusion

This section will make attempts at answering the research questions through the work conducted for this thesis. The last section discuss the project in relation to the Project Goal.

8.1.1. Research Question 1

RQ1 *How is performance affected by altering the input to be more favorable towards the Generator model in a Generative Adversarial Network?*

Three main techniques were proposed based on RQ1. Each make guided modifications the otherwise random input of the generative model to make the learning task easier.

Reusing the input of the Generators through Static Reusable Noise was a simple technique which did affect the performance during training in a positive, albeit minor, way. The noise was only repeated once every epoch of training which may be too infrequent. Still, tests using Image Based Noise Generation showed that making the input too similar only limits the model.

The most promising of the three suggested was Audition Based Noise Selection (ABNS). This technique allows the Discriminator to select representable noise samples that help guides training. Experiments conducted with difficult dataset showed that ABNS was able to guide the training through the most difficult parts of learning. ABNS had less effect on simpler dataset. Comparison between regular models showed that ABNS makes for a Discriminator that is overall weaker but more stable.

The three techniques show that guided alterations to the input of the Generator has the potential of increasing performance throughout the learning process, but not overall. Regularly trained GAN models were eventually able to reach similar performance on all datasets.

8.1.2. Research Question 2

RQ2 *In what way is performance of a Generative Adversarial Network model impacted by adding additional, asymmetrically trained Discriminator models?*

Expanding GAN with additional ANNs is a popular area of research. The common approach so far has been to add identical additional networks with slightly varying parameters. RQ2 was proposed to examine the effect of GAN models were the models have varying training data. The suggested Generative Multi-Adversarial Network with Historic Discriminator (GMAN-HD) was the resulting model of this question. The two Discriminators in GMAN-HD are trained separately and on different data, thereby asymmetric. The Historic Discriminator is trained on previous well rated output from the Generator.

GMAN-HD creates a model that outperform regular GAN models on multiple datasets. The Discriminators are more stable and less overfitted towards the training data. The largest gap in performance was seen early when most of learning occurs.

8.1.3. Research Question 3

RQ3 *How can the performance of the suggested techniques related to RQ1 and RQ2 be accurately measured and objectively evaluated?*

Properly evaluating the quality of a GAN model is difficult s there is no fixed solution to unsupervised learning. Multiples methods were researched and discussed. Generative Adversarial Metric (GAM) (Im et al., 2016a) was chosen as the main evaluation metric. GAM leverages the adversarial min-max aspect of GANs. The Discriminator from one model classify generated samples from the other, and vice versa.

Serial Generative Adversarial Metric (SGAM) is a concrete implementation of GAM that addresses some of the problems uncovered when using it for this thesis. SGAM compares the two models both overall and after epoch of training.

SGAM, and GAM, is not a perfect evaluation metric suitable for all projects. SGAM is not capable of directly evaluate image quality compared to the training data set. A model may produce superior images compared to another model, but the images may look unrealistic to a human evaluator. Likewise, SGAM evaluates every image individually and is therefore unable to evaluate the mode coverage, range of generated samples. To evaluate the local results, SGAM requires checkpoints of past models to be saved and that they are easily retrieved. This was especially designed for use with TensorFlow, but not available for all Deep Learning frameworks. This makes SGAM less universal. With TensorFlow, the checkpoints do require some deal of available storage, especially with additional networks. The storage limitation might not always be ideal.

Nevertheless, SGAM was used to accurately and consistently compare the techniques developed for answering RQ1 and RQ2. The local result was easily plotted which was a helpful tool when discussing the results.

8.1.4. Project Goal

The research goal of this thesis was presented in Chapter 1:

G1 *Explore and develop new techniques for Generative Adversarial Networks specialized for image generation*

The three research questions were defined to reach this goal. Each question led to the exploration and development of at least one concrete technique.

RQ1 asks how performance is altered by making alterations to the input of the generative model. Three techniques were suggested to answer this question. Two of these were too simple and should not be researched further. The third technique, ABNS, was able to reduce training time on more difficult training sets. The technique did not improve performance overall and was quite computationally expensive.

GMAN-HD was proposed to answer RQ2 and explores the field of Multi-GANs, models with multiple Generator and/or Discriminators. GMAN-HD adds a second Discriminator that is trained on differently compared to a conventional Discriminator. GMAN-HD creates a strong model that was shown to outperform conventional GANs. Performance was increased overall and throughout the learning process. GMAN-HD is comparable to ABNS while requiring less computationally power.

RQ3 asks how to properly evaluate techniques related to RQ1 and RQ2. SGAM is a custom implementation of the GAM (Im et al., 2016a) for comparing the quality of two models. Makes it possible to evaluate the effect of concrete techniques, both overall and during training.

A total of five techniques were explored and tested for this thesis. While two of them should not be used further, the rest are promising additions to the field of Generative Adversarial Networks. The conclusion is that the Project Goal has been met.

8.2. Future Work

Covered in this Section is ideas for further exploring the work conducted during this thesis.

8.2.1. Verify results on different datasets

The experiments showed how much the training set affected the results. ABNS and GMAN-HD saw the largest increase in performance on the most difficult dataset tested, CIFAR-10. The results should be verified on other datasets as well. ABNS and GMAN-HD helps guide the learning process which may enable even more difficult datasets be used with GANs.

8.2.2. Experiment with larger auditions sizes for ABNS

ABNS is used every iteration to select the noise for further training. Any increase in auditions size will therefore results in longer training time. The experiment showed a positive correlation between auditions size and performance. Increasing the auditions size on more powerful hardware is a logical next step.

8.2.3. Expanding GMAN-HD

Using two asymmetrically trained Discriminators was shown to increase performance and create a more stable model. GMAN-HD was just one concrete implementation of this method. Further experimentation should be conducted with an increasing the number of models. Using additional models requires more memory to store the network's weights, but the computationally expensive ABNS could be exploited to output images of varying quality during the same run, thereby keeping training time reasonable.

Bibliography

Tarik Arici and Asli Celikyilmaz. Associative adversarial networks. *arXiv preprint arXiv:1611.06953*, 2016.

Margaret A. Boden. Creativity and artificial intelligence. *Artificial Intelligence*, Volume 103:347–356, 1998.

Ishan Durugkar, Ian Gemp, and Sridhar Mahadevan. Generative multi-adversarial networks. *arXiv preprint arXiv:1611.01673*, 2016a.

Ishan Durugkar, Ian Gemp, and Sridhar Mahadevan. Generative multi-adversarial networks. *arXiv preprint arXiv:1611.01673*, 2016b.

Ian Goodfellow. Tutorial: Generative adversarial networks. *Advances in Neural Information Processing Systems 29, Barcelona, Spain*, April 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 1 edition, 2016.

Daniel Jiwoong Im, Chris Dongjoo Kim, Hui Jiang, and Roland Memisevic. Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*, December 2016a.

Daniel Jiwoong Im, He Ma, Chris Dongjoo Kim, and Graham Taylor. Generative adversarial parallelization. *arXiv preprint arXiv:1612.04021*, 2016b.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *The 32nd International Conference on Machine Learning*, pages 448–456, 2015.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 2009.

Yann LeCun and Jordan Bennett. What are some recent and potentially upcoming breakthroughs in deep learning?, 2016. URL `https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning`. [Online; accessed 15-October-2016].

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*, 2016.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. *Google Research Blog*, 2015. URL `https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`. [Online; accessed 9-November-2016].

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016.

Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. Learning from simulated and unsupervised images through adversarial training. *arXiv preprint arXiv:1612.07828*, 2016.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Joshua M. Susskind, Adam K. Anderson, and Geoffrey E. Hinton. The Toronto face database. *Technical Report, University of Toronto.*

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, April 2016.

Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 2008*, 30(11):1958–1970, 2008.

Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2014.

Weiwei Zhang, Jian Sun, and Xiaoou Tang. Cat head detection - how to effectively exploit shape and texture features. In *Proceedings of European Conference on Computer Vision 2008*, volume 4, pages 802–816, October 2008.

Appendices

A. Example of images generated with GAN

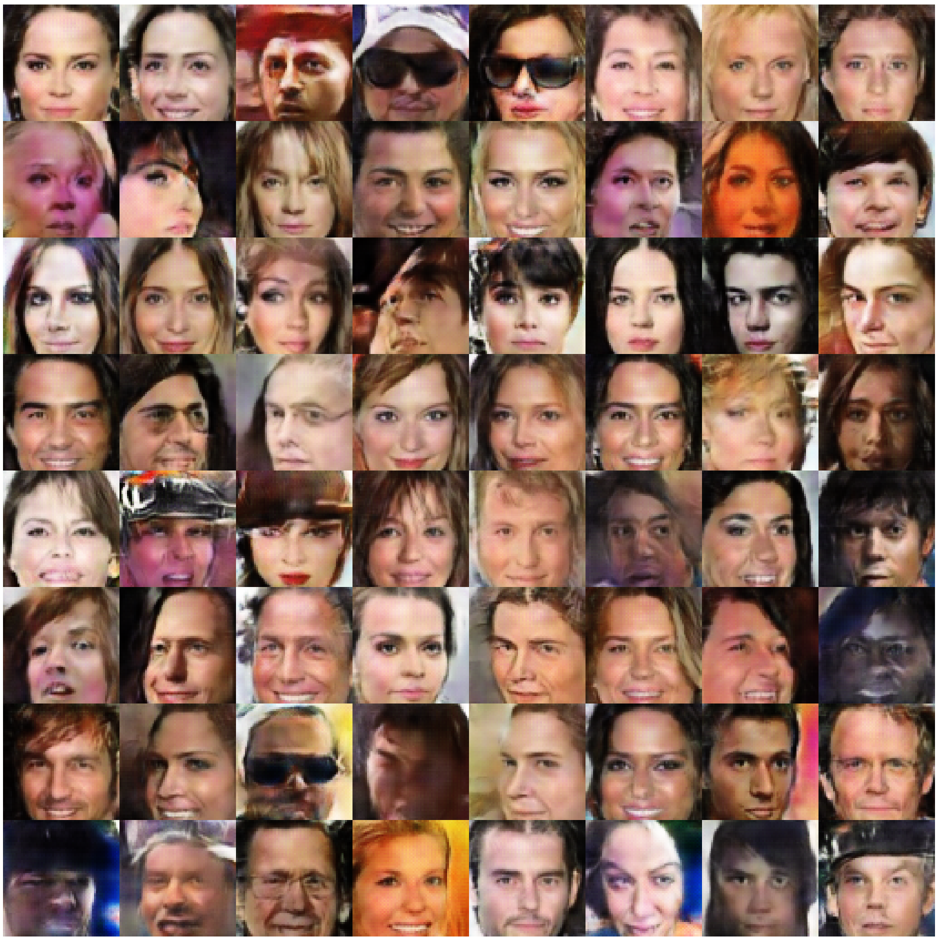

Figure A.1.: Randomly selected output from unaltered DCGAN trained on the CelebA dataset)

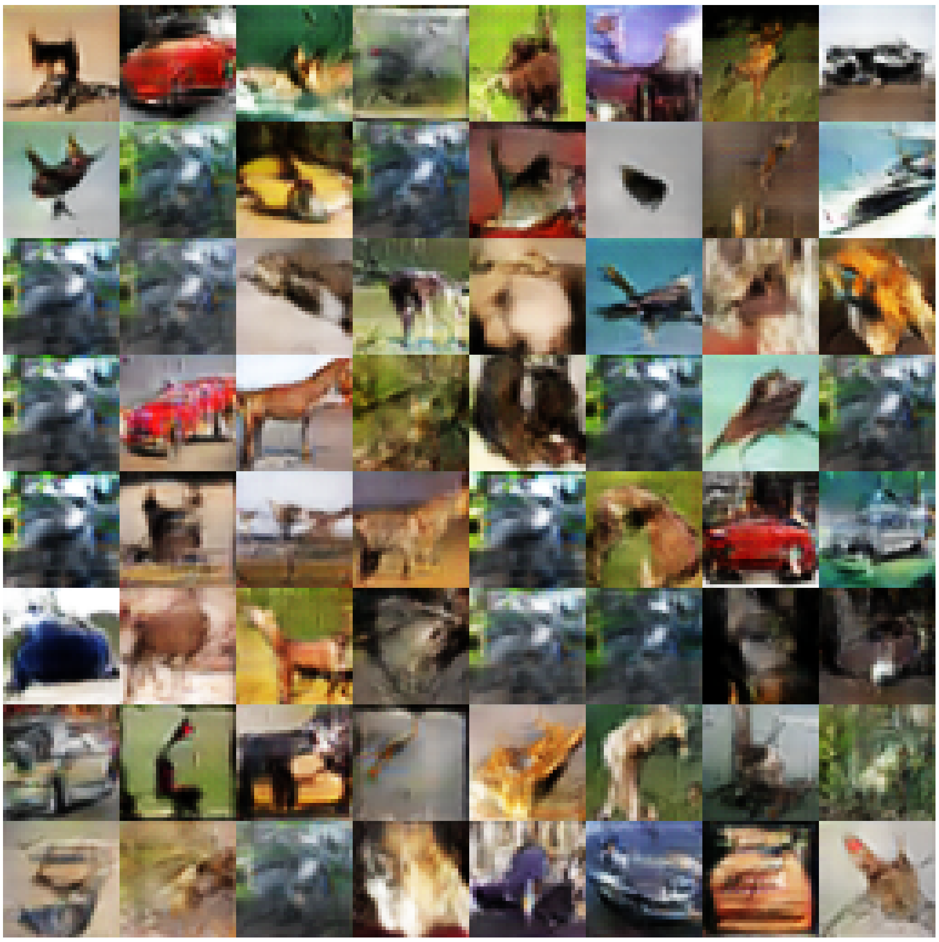
Figure A.2.: Randomly selected output from unaltered DCGAN trained on the CIFAR-10 dataset)

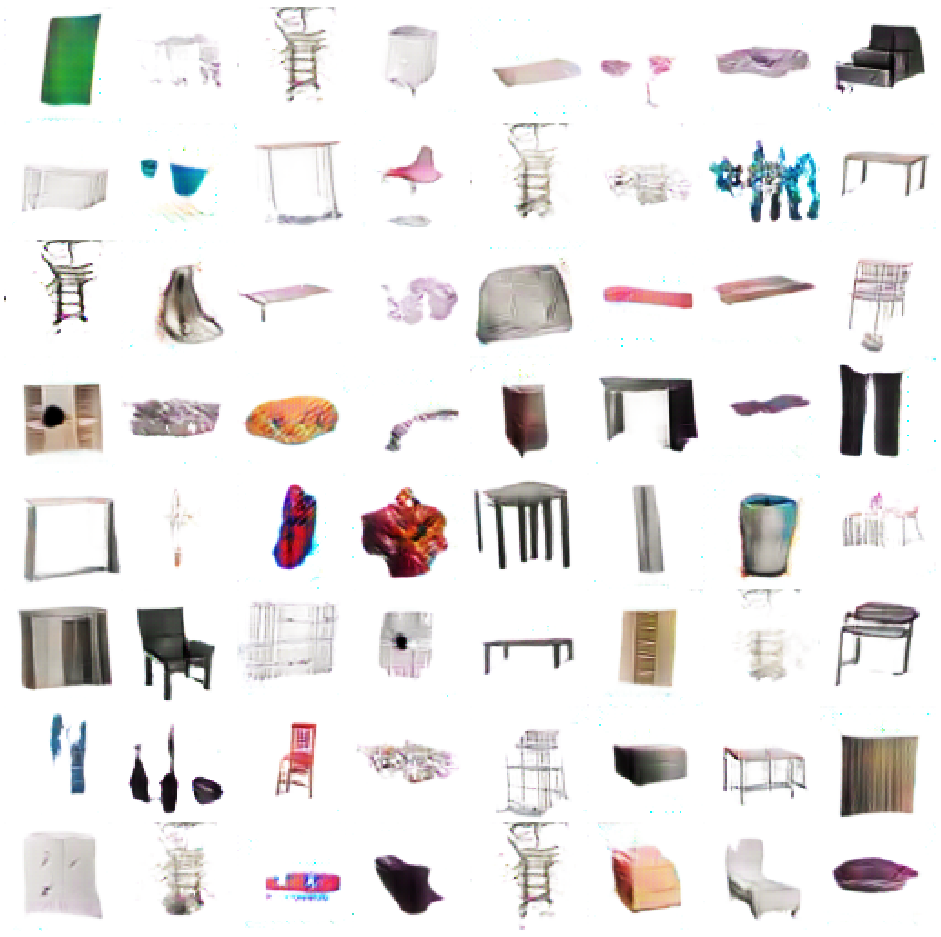

Figure A.3.: Randomly selected output from unaltered DCGAN trained on the IKEA dataset)

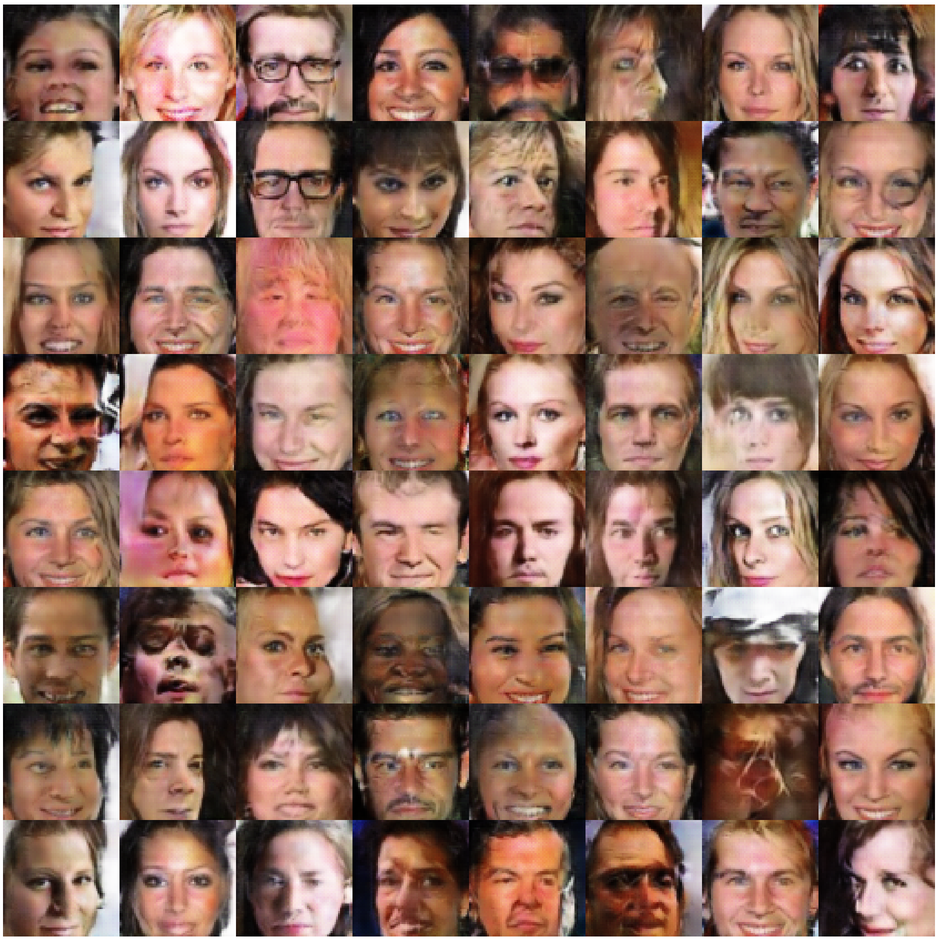

Figure A.4.: Randomly selected output from ABNS-mixed$_3$ trained on the CelebA dataset)

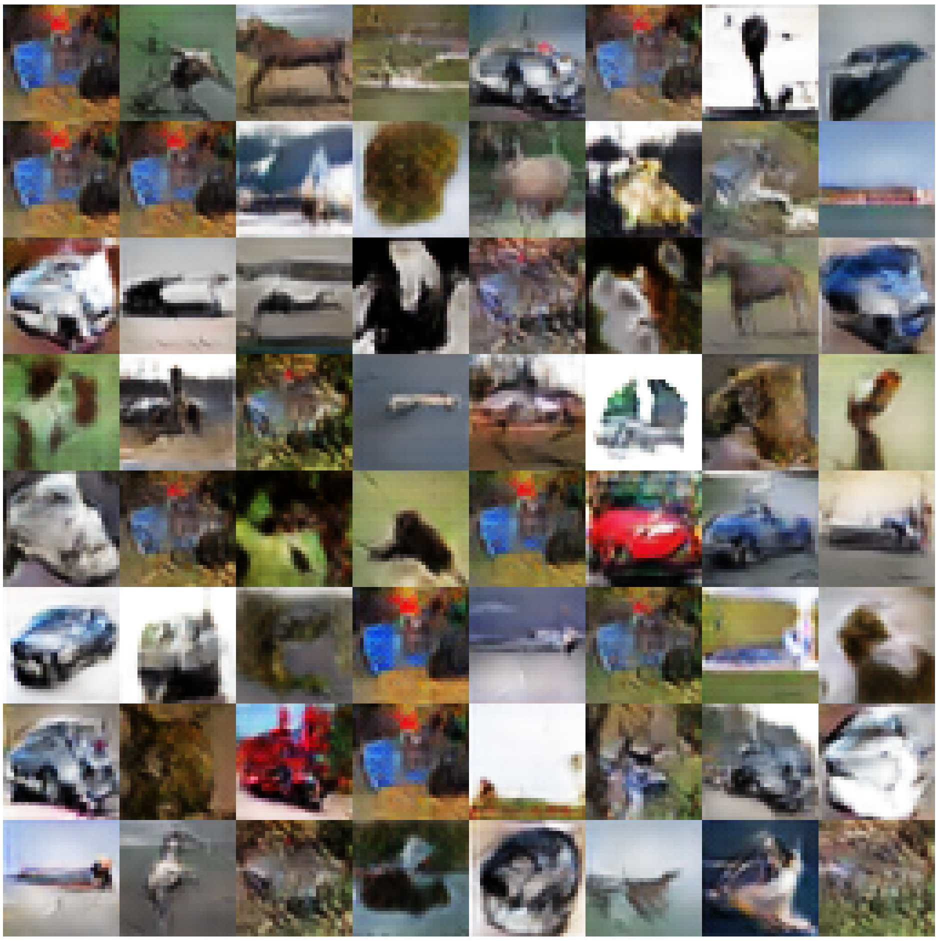

Figure A.5.: Randomly selected output from ABNS-mixed$_6$ trained on the CIFAR-10 dataset)

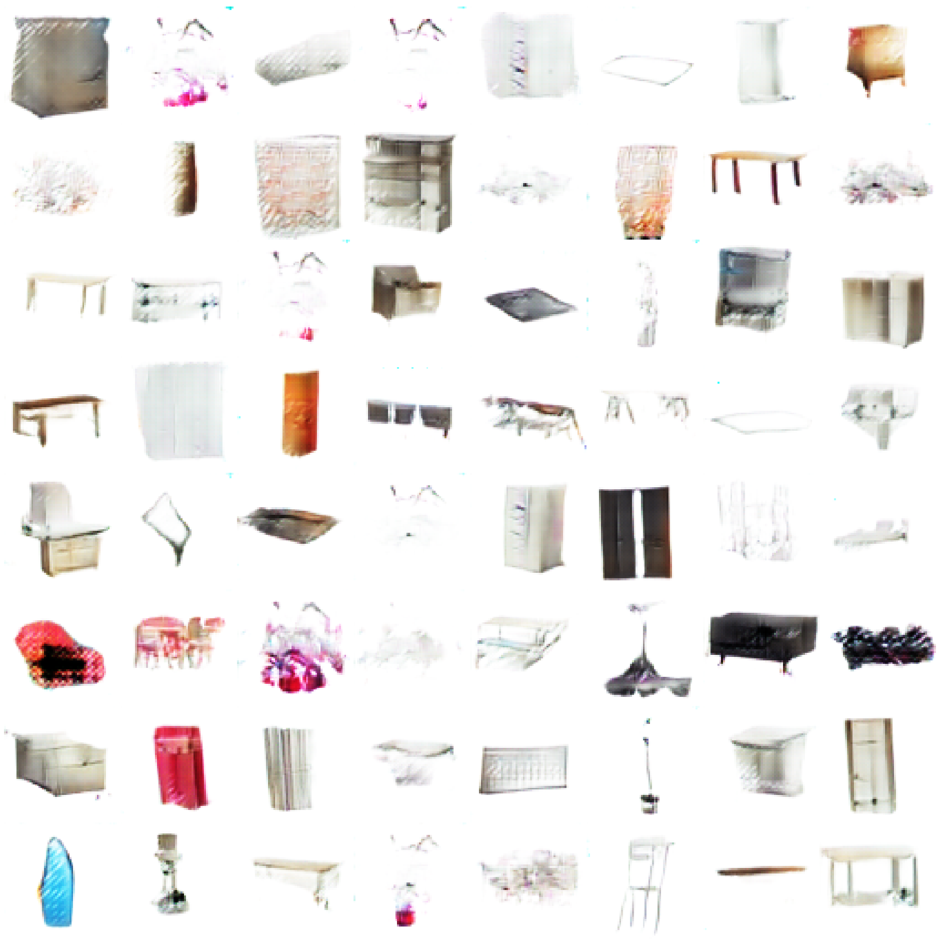

Figure A.6.: Randomly selected output from ABNS-mixed$_6$ trained on the IKEA dataset)

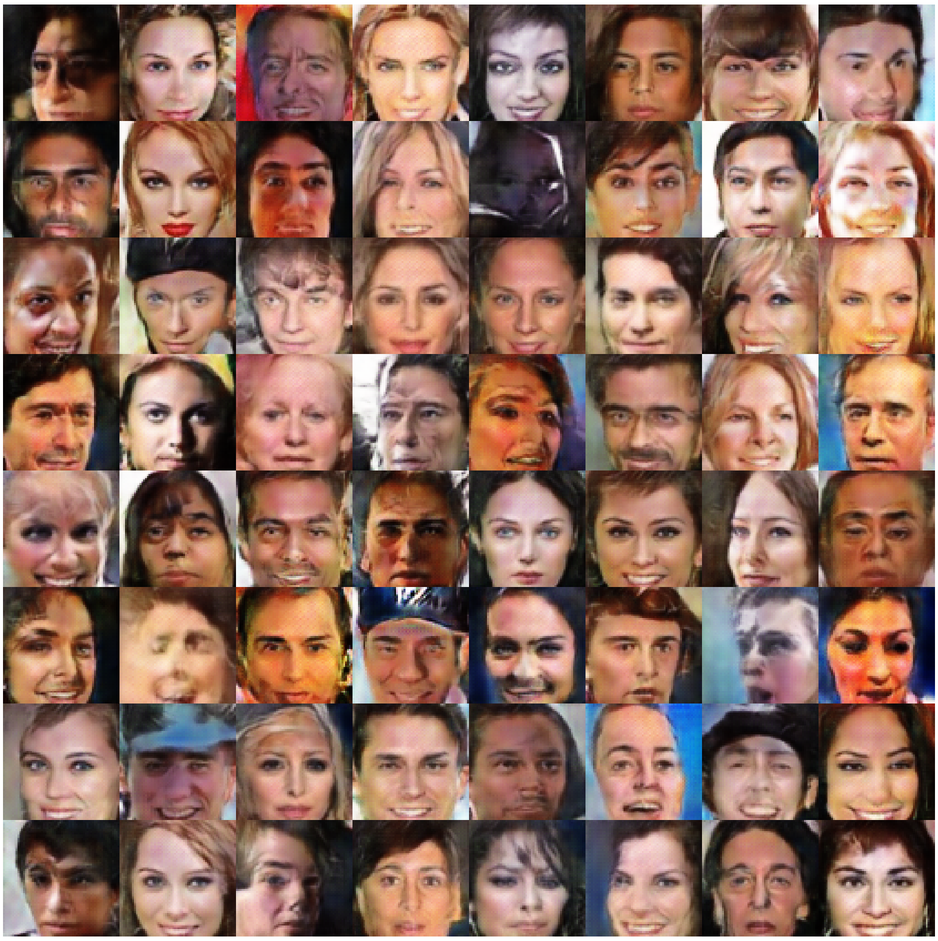

Figure A.7.: Randomly selected output from GMAN-HD trained on the CelebA dataset)

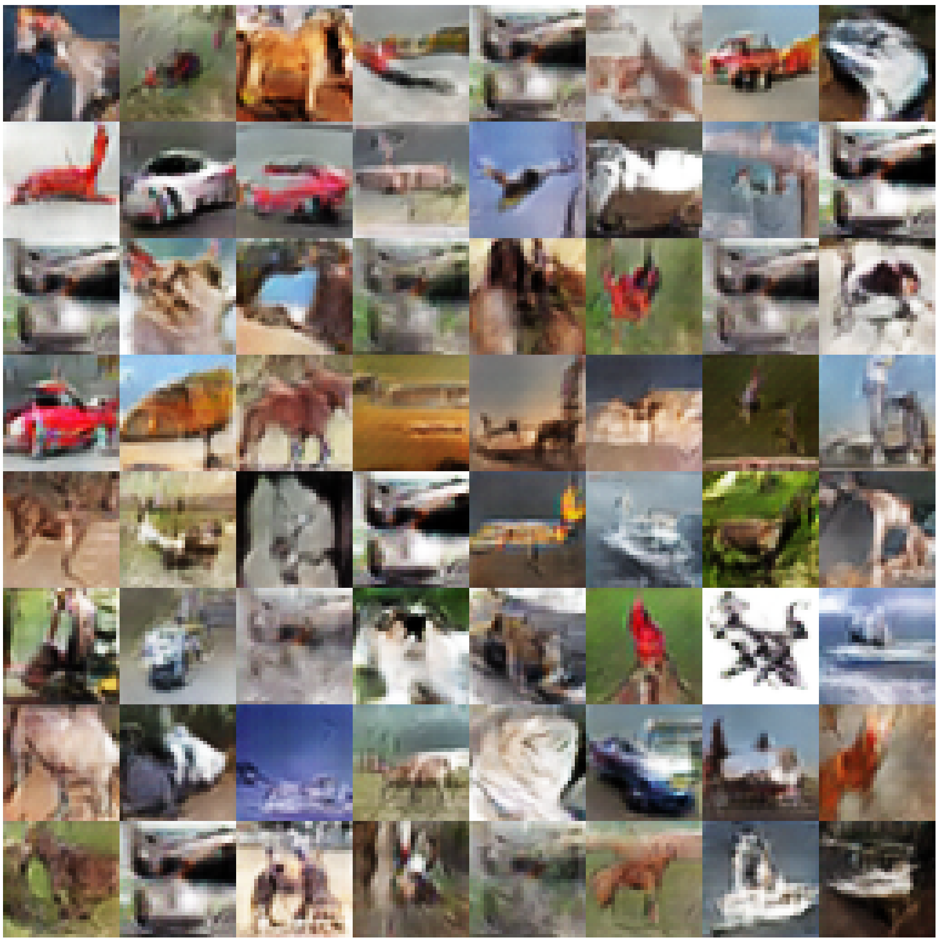

Figure A.8.: Randomly selected output from GMAN-HD trained on the CIFAR-10 dataset)

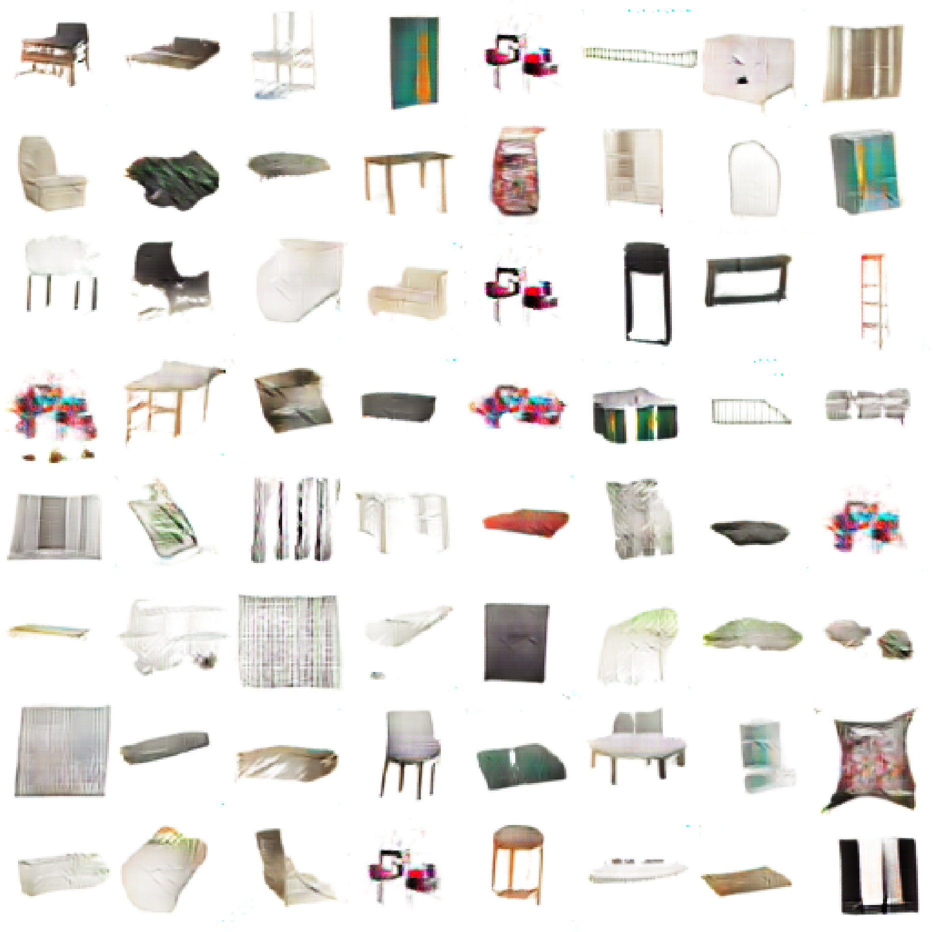

Figure A.9.: Randomly selected output from GMAN-HD trained on the IKEA dataset)