DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Video Synthesis from a Single Image

Patrick Radner

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Video Synthesis from a Single Image

Videosynthese von einem einzelnen Bild

Author:	Patrick Radner
Supervisor:	Prof. Dr. Matthias Nießner
Advisor:	Dr. Justus Thies
Submission Date:	15.12.2020

I confirm that this master's thesis in informatics: games engineering is my own work and I have documented all sources and material used.

Munich, 15.12.2020 Patrick Radner

Acknowledgments

I´d like to thank my advisor Dr. Justus Thies for putting up with me throughout the 3D scanning lecture, the spatial learning practical, my guided research under him and now this master´s thesis. He was a big part of my entire master's program. Our *weekly* meetings during this project were inspiring and his guidance was invaluable. This document would also look a lot more chaotic without his feedback.

Thanks also go out to all of the excellent professors at TUM, whose lectures I attended and my fellow students who I got to work with during various practical. We built some pretty cool stuff, gained lots experience and had some great.

Abstract

In recent years the deep learning community has seen giant leaps when looking at generative models. Generative Adversarial Networks, or GANs, have gained immense popularity and with the introduction of works like StyleGAN or BigGAN it is now possible to generate realistic looking images at resolutions large enough for practical use. As the available memory on graphics processing units increases rapidly the size of deep neural networks also increases. This not only results in better performance on existing problems, but also allows the exploration of new applications for deep learning. One of these areas includes generative modeling for videos, which has gained a lot of attention in the last few years. In this work a novel generator architecture is developed for the task of video synthesis from a single input image. This is probably the most complex task in generative modeling for videos, as no flow or segmentation information is available and the model has to learn which image elements to animate. Furthermore the conditional video generation task is widely considered to be more difficult than the unconditional one. The proposed architecture combines recurrent feature pyramid architectures from previous works such as DVD-GAN with styled convolutions introduced by StyleGAN. The resulting network is able to generate sharper images and more realistic motion using roughly the same number of network parameters. The proposed model is evaluated against state-of-the-art baselines and is shown to work on standard video prediction benchmarks.

The task of "bringing landscape images to live" is also investigated. The general motion model used by this work allows to model complex, detailed changes, such as flowing water. Most modern works focusing specifically on this task tend to struggle with such effects due to their simplified motion models. A new dataset for video synthesis from landscape images is created. This dataset focuses on fine detailed motion rather than global changes. The architecture is also shown to work on traditional landscape video generation by being trained on time lapse videos of moving clouds.

Kurzfassung

In den letzten Jahren konnte die Deep Learning Gemeinschaft gewaltige Fortschritte im Bereich der generativen Modellierung verzeichnen. Besonders Generative Adversarial Networks, GANs, wurde sehr viel Aufmerksamkeit zugetragen. Dank Architekturen wie StyleGAN oder BigGAN ist es nun möglich realistische Bilder mit hoher Auflösung zu erzeugen. Durch die Verbesserung von Graphikprozessoren können größere und komplexere neuronale Netze trainiert werden, was wiederum das Erforschen neuer Anwendungsgebiete für Deep Learning ermöglicht. In diese Kategorie fallen unter anderem Generative Modelle für Videos. Diese Masterarbeit präsentiert eine neuartige Deep Learning Architektur für Videosynthese von einem einzigen Bild, was laut aktuellem Stand die wohl komplizierteste Aufgabe in diesem Feld darstellt, da keine Bewegungsinformation aus dem Eingabebild gewonnen werden kann und das neuronale Netzwerk lernen muss, welche Regionen animiert werden müssen. Für diesen Zweck wird die bekannte DVD-GAN Architektur modifiziert und mit Styled Convolutions erweitert. Diese sind bereits sehr gut bekannt im Feld der Bildsynthese und führten dort zu einem großen Qualitätssprung gegenüber vorherigen Modellen. Die fertige Architektur wird auf Basis aktueller Evaluierungsmethoden mit existierenden Arbeiten verglichen. Auch das Animieren von Landschaftsbildern mittels der neuartigen Architektur wird in dieser Arbeit erforscht. Dank des generellen Bewegungsmodells, welches hier verwendet wird, können komplexe und chaotische Bewegungsvorgänge, wie zum Beispiel fließendes Wasser, realistisch erzeugt werden. Hierzu wird ein neuer Datensatz erstellt, welcher sich auf Wasserkörper spezialisiert. Das Modell wird auch auf einem existierenden Datensatz von Zeitraffervideos von Wolken trainiert und erzeugt plausible Ergebnisse.

Contents

1 Introduction

Deep learning has drastically revolutionized the fields of computer vision and graphics in the last decade. Starting with huge successes on ImageNet[1] classification, the community branched out into more complex applications, such as detection and segmentation, deep reinforcement learning, etc. The introduction of generative models like Variational Auto Encoders (VAEs)[2] and the more famous Generative Adversarial Networks (GANs) [3] has enabled the deep learning researchers to take on an even larger variety of tasks. Recently papers like StyleGAN [4] and BigGAN [5] demonstrated that GANs can be used to generate photo realistic at high resolution (1024×1024). Recent works [6, 7, 8, 9] have shown that generating videos on modern hardware is possible. These models were trained on different datasets ranging from the small BAIR robot pushing dataset [10] to the large Kinetics 600 [11], which contains 600 different human actions. While they can generate short realistic looking sequences, the models use hundreds of millions of parameters and require several enterprise-level GPUs or tensor processing units (TPUs)[12] for training. This is however not feasible for most applications. In terms of resolution the generated videos are also not yet viable for most practical use-cases. Additionally, a lack of objective metrics makes is difficult to compare results of competing works.

This master's thesis explores the task of video synthesis using generative adversarial networks. As input a single image is required. A new architecture based on a combination of state of the art image and video generation architectures is introduced and evaluated against existing methods. The architecture is trained on common datasets and evaluated against existing methods. An additional focus lies on the task of "Bringing landscape images to life" and custom dataset is created for this task. In this task a video is generated from a single landscape image. The video shows realistic moving scenery, such as flowing water, waves, moving leaves, etc. Since only a single frame is given, the algorithm has to learn, which areas should be animated and which ones should remain static. The algorithm should be able to work with high resolutions, as motion in landscape images, such as flowing rivers is usually very fine-scale. Sample sequences of such videos are shown in Figure 1.1. As a secondary objective this work tries to show, how video prediction could be applied to landscape images created with GauGAN [13]. GauGAN allows users to create fairly realistic landscape images by drawing semantic segmentation maps. Combined with a single image landscape video prediction algorithm this might show the way towards a new way of automatic video content creation. Lastly, the architecture is constrained by the available compute resources, namely *Nvidia GTX 1080ti* graphic cards.

When discussing possible applications for generative models for videos it is important to differentiate between video generation and video prediction. Video generation describes the classical GAN task of generating samples from a latent feature code, which is usually

Figure 1.1: Sample videos generated by the proposed architecture. The videos also show the new dataset created for this work. Videos are generated at 128×128 resolution. For full video please visit the supplementary website: `mirjang.github.io/mt_videoprediction`

drawn from a probability distribution such as the Normal distribution. Video prediction on the other hand takes as input one or more frames and predicts a possible future from those frames. Since the future is inherently multi-modal, some form of latent code might still be required to generate multiple possible videos. This work focuses on the specific video prediction case where only a single input image is available. Video prediction in general finds its use in a variety of fields. The ability to predict likely events in the very near future can be very valuable in control tasks, such as autonomous driving. Being able to predict possible outcomes given past states is also a key component in model based reinforcement learning.

When it comes to the task of "Bringing landscape images to life", the applications are mostly artistic and can be used to support creators of video content. In 2019 Park et al. released a paper called “Semantic Image Synthesis with Spatially-Adaptive Normalization” [13] in which they introduced the image-to-image translation network GauGAN. This architecture allows artists to create fairly realistic landscape images by drawing simple segmentation maps. The method presented here or in similar papers such as [14] further allows artists to quickly create realistic landscape videos, which might for example be used as background in movies or video games. This could drastically increase production speed and reduce costs, as GauGAN images can be created within minutes and inference of video prediction methods such as the one presented here can be computed within seconds on commodity GPUs. Some sample landscape images created with GauGAN are shown in Figure 1.2. Once one learns how to use the GauGAN interface and its intricacies, such as how the placement of the horizon affects perspective, realistic looking images can be created by drawing simple semantic segmentation masks. State-of-the-art video generation and prediction is limited to only a few seconds at fairly low resolution. As research in this area progresses and more powerful hardware becomes available the length, resolution and quality of generated videos will only increase. Future methods might eventually be able to generate entire movies. This would allow for virtually infinite amounts of watchable content. While this might seem like science-fiction, today’s large scale natural language processing models are already capable of generating long and consistent articles and scripts [15] and could be used as basis for video generation.

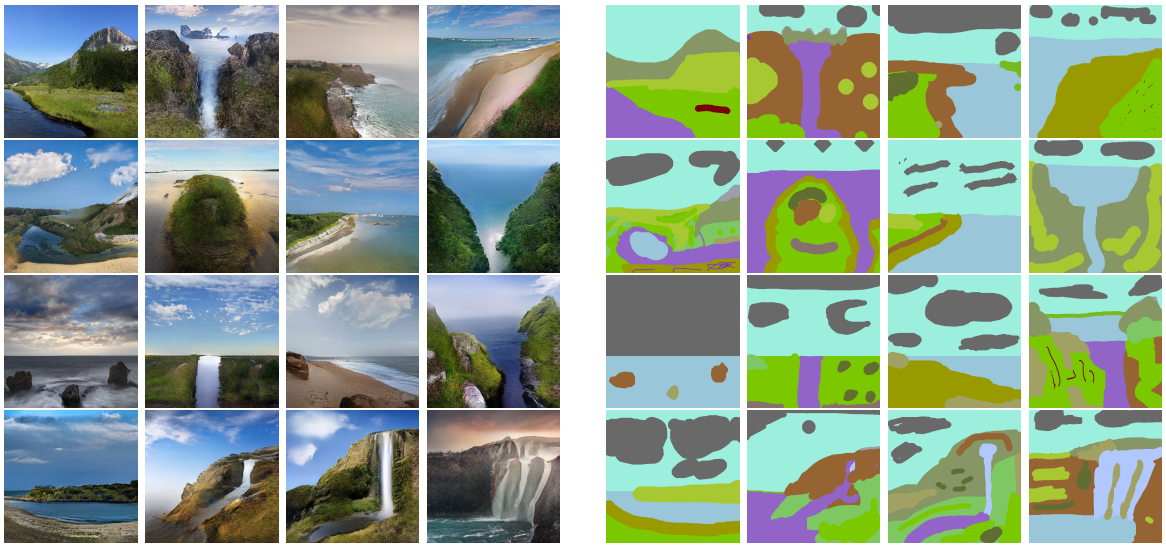


Figure 1.2: Landscape images created with GauGAN [13]. On the left side are the generated images and on the right side the corresponding, hand-drawn segmentation masks using the online tool: `www.nvidia.com/en-us/research/ai-playground/`

2 Related Work

In this chapter previous works and algorithms are explored, which either serve as basis for this work or help in comparing it against the state of the art. At first the basic literature on generative adversarial networks is reviewed in section 2.1. Next common image generation algorithms are mentioned in section 2.2. Then section 2.3 goes over the state of the art in video generation and video prediction. In section 2.4 existing art for video synthesis from landscape imagery is explored. Finally section 2.5 shows common datasets used as benchmarks to measure video generation and video prediction performance.

2.1 Generative Adversarial Networks

GANs were first introduced in 2014 by Goodfellow et al. [3]. Since then they have become the most dominant architecture for generative modeling in deep learning. Their basic principle involves two networks: the generator and the discriminator. The generators job is to generate samples close to the distribution of a given dataset, while the discriminator has to determine whether a given sample was drawn from the original dataset or generated by the generator. Goodfellow et al. [3] shows the existence of an equilibrium, when the generator matches the dataset distribution and the discriminator can only guess if a sample is real or fake. In practice, however, vanilla GANs often suffer from stability issues during training. An additional concern is (partial) mode collapse, where the generator fails to match the complexity of the original distribution and only produces a small variety of samples. "Improved Techniques for Training GANs" [16] provides a set of instructions and best practices on how to successfully train vanilla GANs. Another problem of vanilla GANs is that it is impossible to define when to stop training based on the computed loss values. Convergence is usually determined by either generating samples from the same latent code at different times in the training process and manually inspecting them or by computing external scores such as Inception Scores [16] or Fréchet Inception Distance (FID) [17]. To alleviate this problem Wasserstein GANs [18], or WGANs for short, propose a different algorithm, in which the discriminator learns to approximate the earth movers distance between a given sample and the dataset distribution. During training this results in the discriminator loss converging to zero, which indicates that training is complete. Wasserstein GANs have the requirement, that the function represented by the discriminator needs to be 1-Lipschitz continuous. Some ways to fulfill this constraint include crudely clipping network weights, applying a gradient penalty during training [19] and using spectral normalization [20]. A number of best practices for training Wasserstein GANs for image synthesis are provided in "Large Scale GAN Training for High Fidelity Natural Image Synthesis" [5]. For a detailed description of GAN and

Wasserstein GAN loss functions see section 3.1. "Conditional Generative Adversarial Nets" [21] introduced the idea of conditioning generators on a given input, such as class labels or segmentation masks. The conditioning inputs are also added to the input of the discriminator. Instead of generating samples from scratch, CGANs are given some constraints on what the output should look like. This gives the user more control over the generator output. For image synthesis this lead to the idea of image to image translation networks such as Pix2Pix [22] or GauGAN [13].

2.2 Image Generation

Often form the basis for video generation algorithms is found in current image generation techniques. As such, this section explores state of the art GANs for image generation. Most of the advancements in image synthesis using GANs has come form trying to stabilize the training process [16, 18, 19, 23]. State-of-the-art image GANs are also mostly trained using Wasserstein loss.

StyleGAN [4] and the follow-up work "Analyzing and Improving the Image Quality of StyleGAN" [24] have demonstrated GANs ability to generate realistic, high quality images of single objects at a high resolution of 1024×1024. StyleGAN owes its success largely to its unique generator architecture built from modulated convolutions, in which individual feature channels are scaled based on a latent style vector. The StyleGAN architecture has been shown to provide an incredible amount of control over the image synthesis process. Lower layers usually control global attributes including object pose, whereas higher ones change details like glasses and hair color for faces. StyleGAN also introduced style-mixing, which refers to using different style vectors at different network layers, and showed how it can be used to smoothly interpolate between samples. It is also used as a regularization technique during training. A detailed explanation of the building blocks used in the StyleGAN and StyleGAN2 architectures is given in section 3.3.

While StyleGAN has been mostly applied to a single domain, such as faces or cars, BigGAN [5] is able to generate samples from a large number of different classes and is trained using the entirety of ImageNet [1]. Its main building blocks closely resemble residual blocks [25] and it uses spectral normalization [20] to ensure 1-Lipschitz property. The huge number of classes is dealt with by providing a class embedding as input to the generator and by using conditional batch normalization [26]. BigGAN is also aptly named, as it has a large number of parameters, a large batch size of up to 2048 samples and requires an enormous amount of compute resources to be trained.

2.3 Video Generation and Prediction

Early video generation models tried to apply well studied architectures for image generation and switch the 2D convolutions for 3D convolutions [27]. The overall quality of these approaches was, however, not satisfying. Most state of the art architectures for video generation and prediction follow the paradigm of using recurrent neural networks to predict

temporal changes in a latent space and them use the latent codes to predict each frame independently [6, 7, 8]. This section lists some of the most seminal works on video generation and prediction:

The first time generative adversarial networks were used for video generation was in the paper "Generating Videos with Scene Dynamics" [27], which was published in 2016. There the authors propose a two stream approach, where foreground and background are generated separately. The static background is generated using a 2D convolutional neural network and is the same for all frames. The foreground is generated with a 3D ConvNet to allow for dynamic changes over time. The foreground network also computes a mask for each pixel in each frame. This mask is used to combine foreground and background stream to form the final video. The videos generated by VGAN are usually blurry and unrealistic. The model does, however, show good results for its time when looking at its video prediction performance. This is interesting as video prediction is currently considered to be more difficult for neural networks than video generation [7, 8].

Temporal Generative Adversarial Net or TGAN is introduced by "Temporal Generative Adversarial Nets with Singular Value Clipping" [28]. TGAN is the first model to propose splitting a spatio-temporal generator into a temporal and an image generator. The algorithm first generates a set of latent codes using a 1D convolutional neural network. These codes are then individually transformed into the final frames by the image generator. Compared to most modern approaches, TGAN uses a single discriminator consisting of 3D convolutions. "MoCoGAN: Decomposing Motion and Content for Video Generation" [6] generates videos by splitting their framework into a motion and a content part. Content specifies the objects present in a video, while motion deals with dynamic elements. Both are represented by a random vector which is then used in an image synthesis network to generate a sequence of frames. While the content part stays fixed throughout the entire sequence, the motion part is updated for each frame using a recurrent neural network. MoCoGAN was also the first paper to propose splitting the discriminator part of the GAN into an image and a video discriminator. While the video discriminator alone should be enough to train the generator, the paper found that adding the additional image discriminator helps significantly with training, as its task is comparably simpler to solve. This results in a more stable learning signal for the generator. In "Train Sparsely, Generate Densely: Memory-efficient Unsupervised Training of High-resolution Temporal GAN" [9] the authors introduce TGANv2, a successor to TGAN that is capable of state-of-the-art video generation. It still uses a temporal and an image generator. The temporal generator is now replaced by a Convolutional Long Short Term Memory network. The image generator is changed to output an image after each up-sampling operation. The images at each level are concatenated to a video and given to an individual discriminator based on a 3D ResNet architecture. The TGANv2 architecture is shown in Figure 2.1.

Following the lessons learned from MoCoGAN, the paper "Adversarial Video Generation on Complex Datasets" [7] introduces the DVD-GAN architecture. It improves upon the

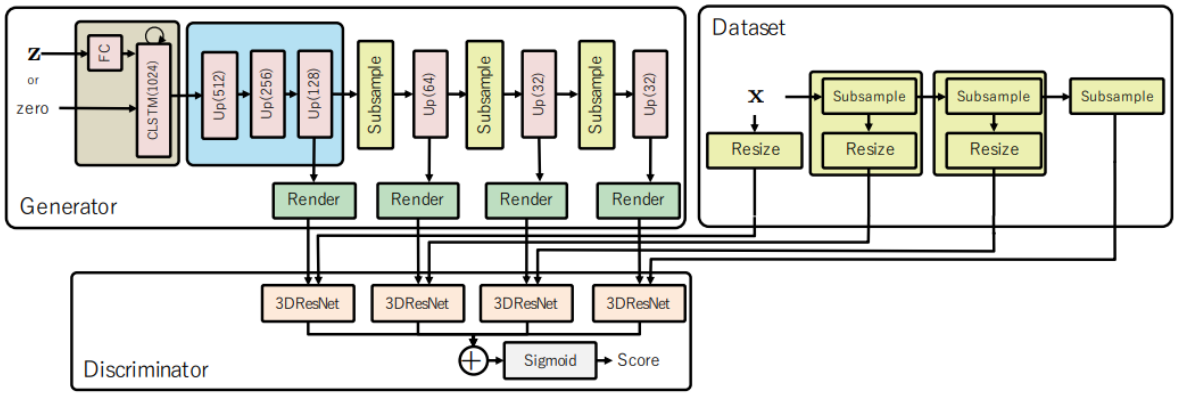


Figure 2.1: Overview of the TGANv2 architecture. CLSTM is a convolutional Long Short Term Memory network [29]. Up blocks are equivalent to GResNet blocks used in DVD-GAN[7], which are explained in section 3.2. Render is a simple block consisting of normalization and a single 3×3 convolutional layer. 3DResNet is a simple ResNet [25] architecture using 3D convolutions. The first layer of the discriminator uses spatial average pooling to reduce input dimensions. Source: "Train Sparsely, Generate Densely: Memory-efficient Unsupervised Training of High-resolution Temporal GAN" [9]

discriminator decomposition by spatially down sampling the input of the video discriminator, reducing memory requirements and thus allowing the generation of longer and more high resolution videos. Additionally it extends the image generation architecture introduced in BigGAN [5] to be used for video generation. The resulting architecture is in spirit similar to the one introduced by TGAN [28], as it also decomposes the generation process into a temporal and an image generator. Together with a large amount of TPUs [12], this allows DVD-GAN to synthesize high quality videos from the Kinetics 600 dataset [11] and sets a good baseline for large scale video generation. A coarse representation of the DVD-GAN architecture is shown in Figure 2.2.

As a follow-up work to [7], "Transformation-based Adversarial Video Prediction on Large-Scale Data" [8] introduces the TrIVD-GAN architecture. The generator in this model replaces the gated recurrent unit used in DVD-GAN with a kernelized version of TrajGRU [31]. The paper proposes various ways to rewire such transformation-based recurrent units, but the common theme is that all of them predict and apply an implicit flow based warping to the latent images. The paper studies different discriminator decomposition and demonstrates a faster model in which both image and video discriminator only look at down sampled frames. The video discriminator input is also only a short sub-sequence of the final video. As this might miss global consistency issues an additional strong version is proposed, which consists of the fast version and an additional video discriminator looking at the entire, down sampled sequence.

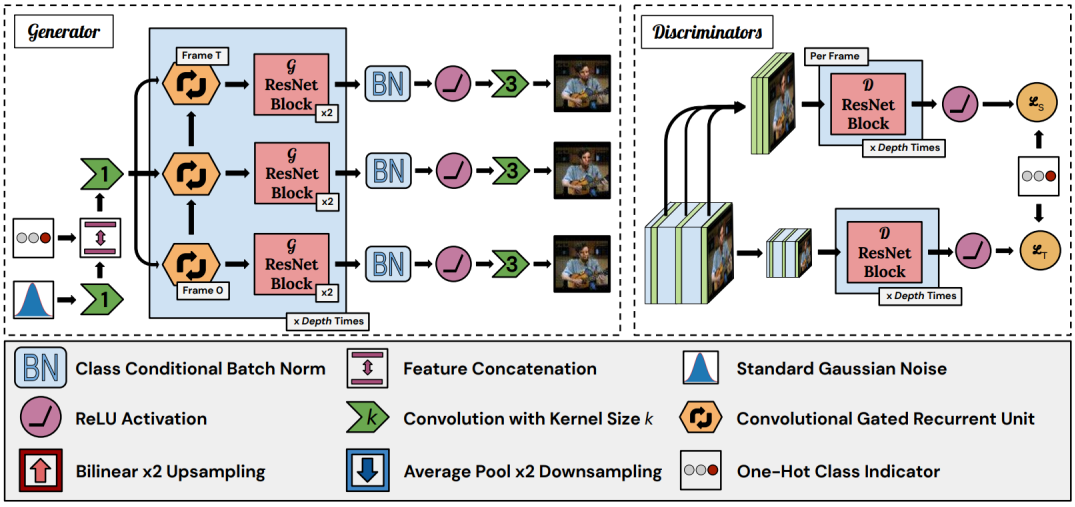


Figure 2.2: Simplified DVD-GAN architecture. For the generator a latent vector is created from Gaussian noise and a class embedding. Afterwards a ConvGRU [30] is unrolled to generate the video and the frames are up sampled using GResNet blocks introduced by[5]. Note the $\times DepthTimes$ notation implies a feature pyramid architecture. The discriminators are split into a spatial and a temporal discriminator. Source: "Adversarial Video Generation on Complex Datasets" [7].

2.4 Landscape Videos

The task of "Bringing Landscape Images to Life" is a special variation of video prediction, which focuses on landscape images and allows for only a single input image. "Animating Landscape: Self-Supervised Learning of Decoupled Motion and Appearance for Single-Image Video Synthesis" [32] tackles this by firstly warping the input image based on a predicted backwards flow and then applying color transfer based on latent codes stored during training to synthesize a looping landscape video at high resolutions of up to $1024x512$. The videos generated by this method are up to 10 seconds long, which is a huge leap compared to the state of the art, however, the authors found, that their model failed when it came to handling non uniform motion, such as splashing water. DeepLandscape [14] proposes using a StyleGAN based architecture to generate its images. Input images are first processed through a neural network to predict StyleGAN latent codes, which are then further optimized via reconstruction loss on the predicted image. Latent codes are divided into static and dynamic vectors and matrices. While the latent vectors encode global, the matrices encode local scene information. Dynamics are generated by applying a simple transformation to the dynamic elements of the latent vectors. This transformation is not learnable, which means, that no gradients have to be propagated back through time. This simplifies the training process tremendously. The model is trained using a StyleGAN like image discriminator as well as a pairwise discriminator, which ensures that any motion generated by the transformation model leads to realistic outputs.

Figure 2.3: Sample sequence of the BAIR robot pushing dataset [10]. As the name suggests, a robotic arm is recorded pushing around various objects. Everything else in the scene, as well as the camera position is static throughout the video.

2.5 Video Datasets

There are actually very few datasets dedicated to video generation or prediction. Datasets such as UCF101 [33] or Kinetics [11] are often used to demonstrate general video generation or prediction performance. They contain videos of certain action paired with ground truth labels of which action is performed and were originally meant for action recognition or similar tasks. For the task of video synthesis from single landscape images a custom dataset was created.

BAIR Robot Pushing Dataset "Self-Supervised Visual Planning with Temporal Skip Connections" [10] introduces the BAIR robot pushing dataset. The dataset contains 43264 training and 256 test sequences. Each sequence shows a robotic arm pushing various objects. The sequences are 30 frames long and have a resolution of 64×64. Due to the limited domain and small resolution with clear motion this dataset is great for developing video generation and prediction models. A sample sequence of the dataset is shown in Figure 2.3. The same camera is used throughout the dataset and it is completely stationary during each clip.

UCF101 UCF101 is a dataset of 101 human actions consisting of over 13,000 video clips [33] gathered from `youtube.com` [34]. The dataset was originally designed for human action recognition, but has found wide use as a video generation and prediction benchmark. The videos have a resolution of 320×240 at various lengths (usually several seconds at 30 frames

per second). A sample can be seen in Figure 2.4. The videos were shot on different cameras and lighting configurations.

Kinetics Kinetics is a series of incremental datasets by DeepMind [35]. The datasets contain 650,000 URLs of video clips, each labeled with one of 400/600/700 classes. Each clip is about 10 seconds long. State-of-the-art video generation and prediction methods like mostly work with the Kinetics-600 [11] version [7, 8]. Same as UCF, the videos can contain a significant amount of camera shake, as well as moving cameras.

FaceForensics++ FaceForensics++ [36] is an extension of the FaceForensics dataset [37]. The FaceForensics datasets contain real videos of people speaking, as well as fake ones generated by various facial reenactment techniques. The dataset is intended to be used to train models to detect such DeepFakes, as they are commonly referred to. The real videos can also be used to train video generation and prediction models [9]. In fact, human faces are very often used to train generative adversarial networks [4, 24].

Sky Time-Lapse: "Learning to Generate Time-Lapse Videos Using Multi-Stage Dynamic Generative Adversarial Networks" [38] introduces the sky time-lapse dataset. As the name suggests, this dataset consists of time-lapse videos of clouds and landscape images showing a large portion of sky, which were gathered from YouTube [34]. There are over 1000 long videos, which were manually cut into 2392 short videos for training and 225 short videos for testing. Each video has a spatial resolution of 640×360. Some sample sequences are shown in Figure 2.5.

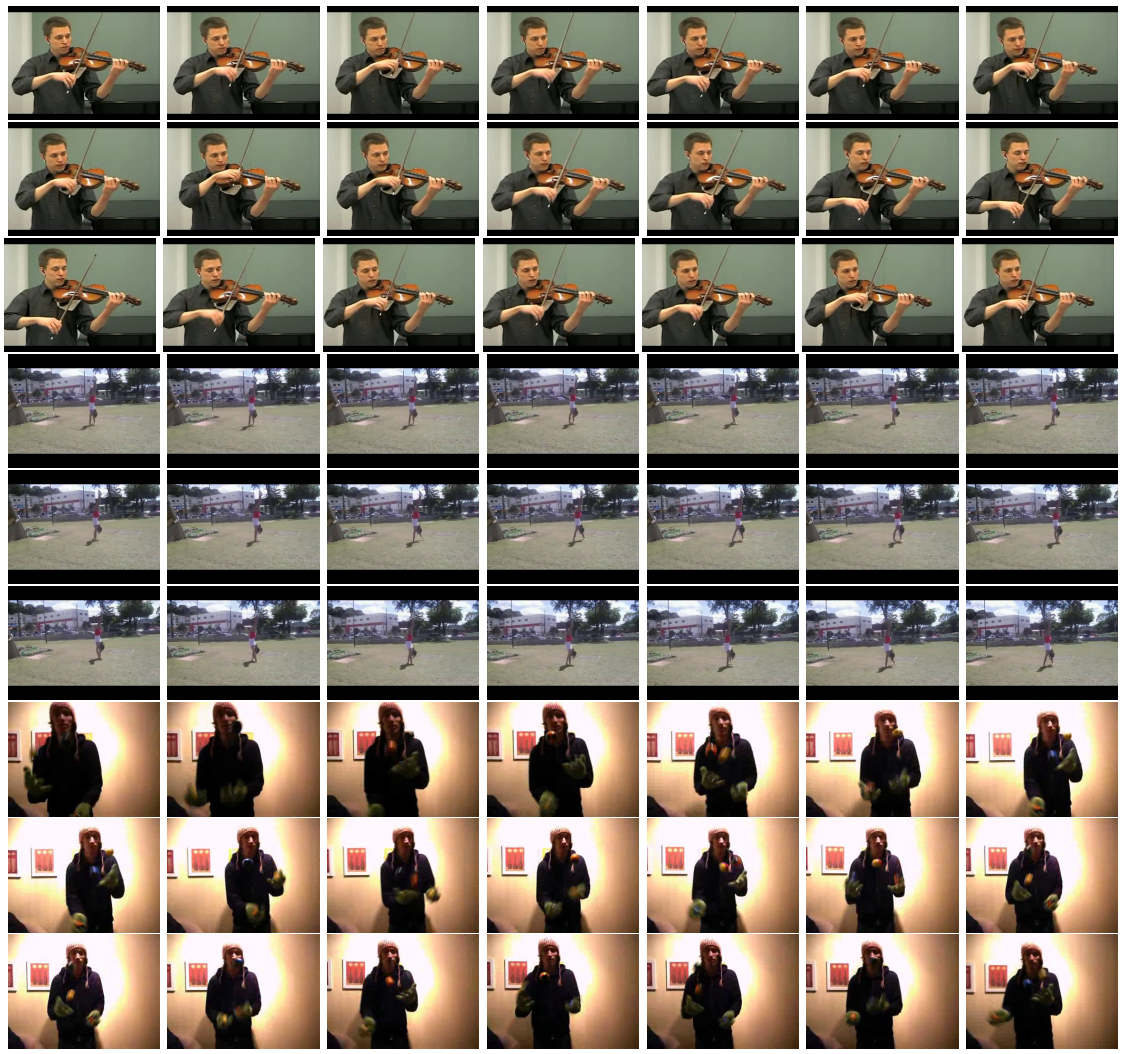


Figure 2.4: Sample sequences of the UCF101 dataset [33]. Every third frame is shown. Classes from top to bottom are: playing violin, handstand walking and juggling balls.

Figure 2.5: The sky time-lapse dataset containing about 2600 short video clips of moving clouds and changing skies. Source: "Learning to Generate Time-Lapse Videos Using Multi-Stage Dynamic Generative Adversarial Networks" [38].

3 Method

This master's thesis presents a deep learning architecture for video synthesis from a single image. The main contribution is the novel generator architecture described in section 3.5. It combines the recurrent architecture introduced by TGAN [28] with style blocks introduced by StyleGAN [4, 24]. ConvGRUs [30] are unrolled to generate the image sequence. Additionally a feature pyramid like approach is taken similar to DVD-GAN [7]. The input is passed through an encoder similar to the one proposed by DVD-GAN [7]. The intermediate features of this encoder serve as input to the feature pyramid network at their respective resolutions. The resulting architecture is capable of generating high quality videos, while requiring smaller batch sizes and fewer compute resources overall. The dual discriminator decomposition is the same as in DVD-GAN [7]. It is reviewed in section 3.6. Before going into detail about the proposed architecture the basic workings of Generative Adversarial Networks are explained in section 3.1. Then key building blocks from the two most prominent architectures in image generation are explored in section 3.2 and section 3.3. Both of them are used to build the generator shown in this work. Next Recurrent Neural Networks and some common variations are explained in section 3.4. The network and training process is thoroughly defined in section 3.7 and section 3.8. For video synthesis from landscape images a custom dataset is created. How this dataset was constructed is described in section 3.9. Finally, in section 3.10 the process down-scaling existing work to allow comparison given the available hardware is explained.

3.1 Generative Adversarial Networks

Generative adversarial networks were first introduced by Goodfellow et al. [3] in 2014. Since then, they have gained huge popularity in the deep learning community. The principal idea behind GANs is to have two competing networks, the generator and the discriminator. The generator constantly tries to produce samples that "fool" the discriminator into thinking, they are from a real distribution represented by the training set. The discriminator on the other hand has to determine, whether a sample stems from the dataset or the generator. This leads to a min-max game with a theoretical equilibrium where the discriminators best option is to randomly guess where a given sample came from [3]. The min-max game is described by Equation 3.1. The classical GAN architecture is shown in Figure 3.1.

$$\max_{\mathcal{G}} \min_{\mathcal{D}} \mathcal{L}_{vanilla}(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{x \sim p_{real}(x)}[log(\mathcal{D}(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - \mathcal{D}(\mathcal{G}(z))] \tag{3.1}$$

Discriminators are the characterizing feature of every generative adversarial network. They provide the only learning signal for the generator. Prior to StyleGAN [4] most GAN papers

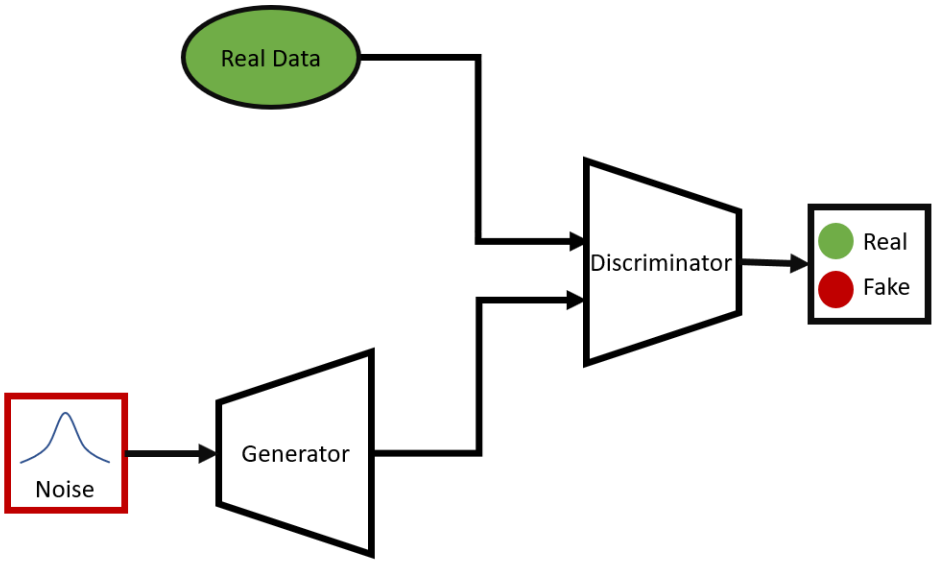


Figure 3.1: Basic GAN architecture. The generator receives noise input and generates a fake sample. The discriminator receives either a sample from the real dataset or a fake sample and has to determine which source it came from.

focused on improving discriminators and their loss functions to achieve better results [39, 40]. A discriminator takes as input either a sample from the dataset or the output of the generator and then learns a function to determinate where the input came from. Dataset samples are usually referred to as real samples, whereas ones drawn from the generator are called fake samples. The output of the discriminator is then a score describing how "real" the image is. This score can either be a single value describing the entire image or a tensor in the case of DCGANs introduced in “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks” [41]. There each entry only describes an area equivalent to the receptive field of a fully convolutional network. The goal of the generator is to learn how to produce images that achieve a high realness in the eyes of the discriminator. The discriminator can thus be seen as a learnable loss function for the generator. In the vanilla GAN setting the realness is seen as a binary label and the discriminator is trained by minimizing the Jensen–Shannon divergence [42]. This leads to the min-max fame described in Equation 3.1. The resulting loss function for the discriminator alone is shown in Equation 3.2. Here $\mathcal{D}$ represents the discriminator and p_{real}, p_{fake} describe the real distribution of the dataset and the fake distribution modelled by the generator. The output of $\mathcal{D}$ has to be in $[0, 1]$, which is usually ensured by applying a sigmoid activation at the end. The loss function in Equation 3.2 is thus simply derived by using Binary Cross Entropy and labeling a real sample with 1 and a fake sample with 0.

$$\mathcal{L}_{vanilla}(\mathcal{D}) = \mathbb{E}_{x\sim p_{real}(x)}[log(\mathcal{D}(x)] + \mathbb{E}_{x\sim p_{fake}(x)}[log(1-\mathcal{D}(x)] \tag{3.2}$$

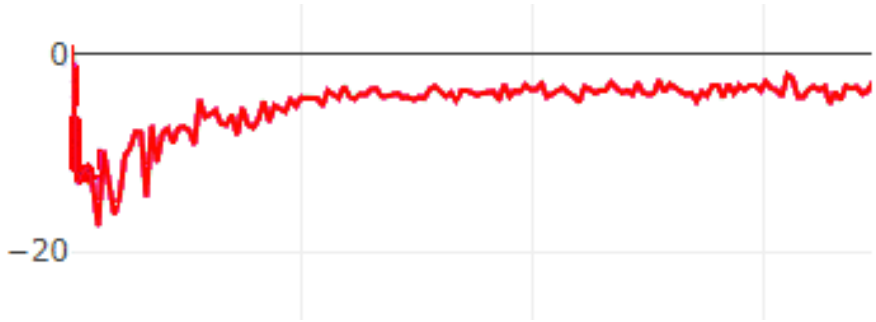


Figure 3.2: Wasserstein GAN training loss a discriminator $\mathcal{D}_S$. X-axis shows time. Y-axis shows the Wasserstein loss given by Equation 3.3. Note that the loss value itself has no direct meaning.

More recently the Wasserstein loss [18] has been introduced for GAN training. It proposes replacing the Jensen-Shannon divergence with the Wasserstein or Earth Movers distance, as it has a much smoother value space, meaning it provides meaningful distance measure even if two distributions are far apart. Directly computing the Wasserstein distance is intractable , which is why it is instead learned by the discriminator. This leads to the loss function shown in Equation 3.3 for the discriminator. For a full deviation of the Wasserstein loss function please see “Wasserstein GAN” [18].

$$\mathcal{L}_{WGAN}(\mathcal{D}) = \mathbb{E}_{x\sim p_{real}(x)}[\mathcal{D}(x)] - \mathbb{E}_{x\sim p_{fake}(x)}[\mathcal{D}(x)] \tag{3.3}$$

Next to a smoother training process, using Wasserstein GANs or WGANs also allows for an easy measure of convergence by looking at the discriminator loss. This was not possible with vanilla GANs. Figure 3.2 shows a classic discriminator loss over time when training Wasserstein GANs. As the loss curves go towards zero, the network converges, demonstrating one of the advantages of Wasserstein GANs. On the downside, however, Wasserstein GANs require the function learned by the Discriminator to fulfill the 1-Lipschitz property. A function is K-Lipschitz continuous if it fulfills Equation 3.4.

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2| \tag{3.4}$$

In order to fulfill this property the authors propose clipping the weights of the entire network to a small range such as $[-0.01, 0.01]$ after each gradient update.

“Improved Training of Wasserstein GANs” [19] shows a more sophisticated approach of enforcing the 1-Lipschitz property by introduction a gradient penalty to each discriminator update. This penalty directly enforces the constraint that "a differentiable function is 1-Lipschitz if and only if it has gradients with at most norm 1 everywhere" [19]. The proposed penalty is given by Equation 3.5 where $\hat{p}(x)$ is created by interpolating along a line between a sample from the real and one from the fake distribution. The resulting network is called WGAN-GP and its discriminator loss is given by combining WGAN loss with the gradient

penalty scaled by a factor λ_{GP} as shown Equation 3.6 .All training in this work was done using the Wasserstein distance with gradient penalty.

$$\mathcal{L}_{\mathcal{GP}}(\mathcal{D}) = \mathbb{E}_{x\sim\hat{p}(x)}[(\|\nabla_x \mathcal{D}(x)\|_2 - 1)^2] \tag{3.5}$$

$$\mathcal{L}_{\mathcal{WGAN-GP}}(\mathcal{D}) = \mathcal{L}_{\mathcal{WGAN}}(\mathcal{D}) + \lambda_{GP}\mathcal{L}_{\mathcal{GP}}(\mathcal{D}) \tag{3.6}$$

Another way of ensuring the 1-Lipschitz constraint is to utilize spectral normalization [20], as is done in BigGAN, DVD-GAN and TriVD-GAN [5, 7, 8]. Spectral normalization ensures, that each weight tensors eigenvalues are less than or equal to one. This is done by computing the highest eigenvalue for each weight matrix during back-propagation by using power iteration. The weight matrices are then divided by their highest eigenvalue. This works for networks with ReLU activation functions, as the Lipschitz constant of ReLU is 1 and the Lipschitz constant of a matrix multiplication is the highest eigenvalue of that matrix.

When training a generative adversarial network, the aim is usually to get the generator to learn a function that represents the distribution behind a given dataset. While during training both the generator and discriminator part of a GAN are trained, discriminators are usually discarded afterwards, as only generators are needed for inference. Generators take some form of latent feature vector and transform it into a sample that tries to fool the discriminator into thinking it came from the training dataset. Classically, the latent vector z is drawn from a Uniform or Gaussian random distribution $p_z(z)$. "Conditional Generative Adversarial Nets" [21] allowed for it to also contain encoded information describing what the generated sample should look like. The generator part of a GAN is trained using the signal coming from the discriminator that has been given a generated, fake image and had its loss function applied as if it were a real image. Note that this requires a separate pass through the discriminator in which its weights must not be optimized. The resulting loss function for the generator is then simply the opposite of the discriminator loss and described by Equation 3.7 for vanilla GANs and Equation 3.8 for Wasserstein GANs.

$$\mathcal{L}_{vanilla}(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{z\sim p_z(z)}[log(\mathcal{D}(\mathcal{G}(z)))] \tag{3.7}$$

$$\mathcal{L}_{WGAN}(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{z\sim p_z(z)}[\mathcal{D}(\mathcal{G}(z))] \tag{3.8}$$

3.2 BigGAN Residual Blocks

This section presents the basic building blocks of the BigGAN [5] generator and discriminator. These blocks are also used by DVD-GAN [8] and TriVD-GAN [8]. The GResNet and DResNet blocks are shown in Figure 3.3. As their names suggest, GResNet blocks are used for the generator part and DResNet blocks serve in the discriminator. They are also inspired by the classical residual blocks first introduced with ResNet [25]. Residual blocks are famous for

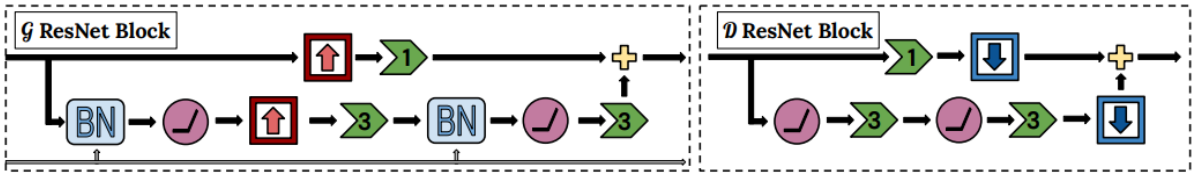


Figure 3.3: Basic building blocks of the BigGAN generator and discriminator. Green arrows are convolutional layer – the number indicates the kernel size. BN stands for Batch Normalization [43]. Red and blue arrows show bilinear up- and down-sampling. The activation function is ReLU [44]. Source: "Adversarial Video Generation on Complex Datasets" [7]

providing a "highway" for gradients, by using skip connections from the input to the output of each individual block. This allows the training of deeper neural networks and reduces the vanishing gradient problem.

3.3 Style Blocks

Changes to the convolutions of the generator are the key feature of the StyleGAN [4] architecture. This section reviews the concept behind the AdaIN operator and modulated convolutions, as well as other key aspects of StyleGAN [4] and StyleGAN2 [24] and describes the style blocks used in this work. An overview of both architectures is shown in Figure 3.4. The original StyleGAN paper introduces the adaptive instance normalization, or AdaIN, operator shown in Equation 3.9.

$$AdaIN(x_i, y) := y_{s,i}\frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i} \tag{3.9}$$

This operator is applied after each convolution in the generator. It normalizes each feature channel x_i individually and then scales them by the scale and bias provided by the style vector $y = (y_s, y_b)$. One important regularization method used in training a StyleGAN network is style mixing. Style mixing is done by using different style vectors at different layers. This decouples the networks layers and allows for different level of detail layers to focus on different aspects of the image. For example a 4×4 resolution layer close to the networks input will likely focus on pose, wheras a layer close to the output might focus on color. As shown in Figure 3.4 (a) and (b), after each convolution operation Gaussian noise is added to each pixel. This provides a direct means to generate stochastic noise details [4]. The changes proposed by the StyleGAN architecture allowed for unprecedented control over the images generated by the network, while still increasing image quality and resolution compared to existing methods.

"Analyzing and Improving the Image Quality of StyleGAN" [24] introduced the StyleGAN2 architecture, an update of StyleGAN based on an extensive empirical study. One of the main changes was the removal of the AdaIN operator and the switch to modulated convolutions. Instead of normalizing and then scaling the feature maps, modulated convolutions scale the

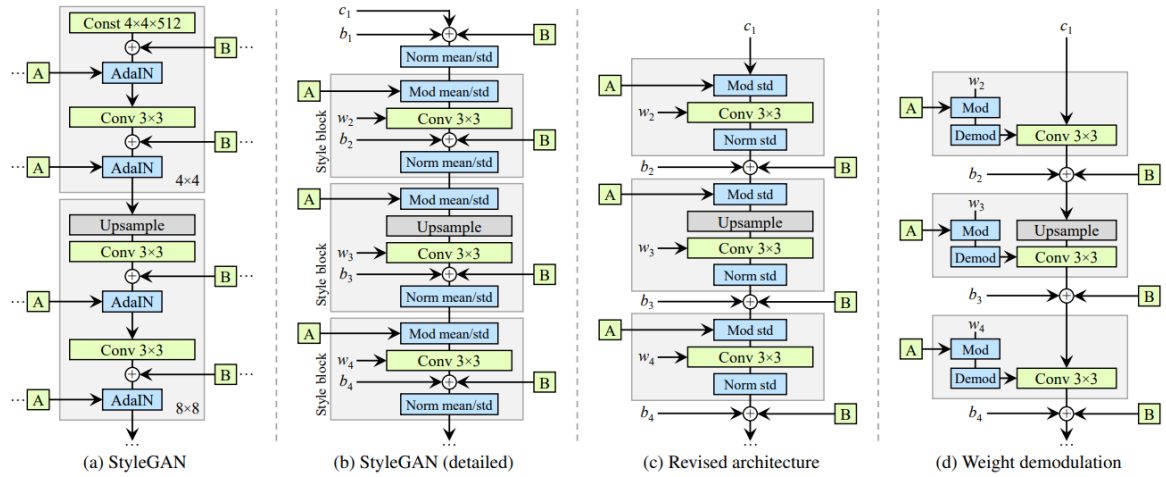


Figure 3.4: Comparison of the building blocks for StyleGAN and StyleGAN2. A refers to a learned, affine transformation of the latent vector and B indicates learned scaling for added Gaussian noise. (a,b) show the original architecture of StyleGAN using the AdaIN operator. (b) Deconstructs the operator into its components (c,d) Source: "Analyzing and Improving the Image Quality of StyleGAN" [24].

weights of the convolution, which is of course equivalent to scaling the feature maps, and then normalize them back to unit variance. The scaling, or weight modulation is described by Equation 3.10 and the normalization or "demodulation" is shown in Equation 3.11.

$$w'_{ijk} = s_i \cdot w_{ijk} \tag{3.10}$$

$$w''_{ijk} = w'_{ijk} \Big/ \sqrt{\sum_{i,k} {w'_{ijk}}^2 + \epsilon} \tag{3.11}$$

Here s_i corresponds to the ith feature map, w refers to the convolution weights and ϵ is a small constant to provide numerical stability.

Replacing the AdaIN operator with modulated convolutions removed the characteristic "water droplet" artifacts that could be found in images generated by StyleGAN. Additionally [24] analyzes the use of progressive growing [23] for training and opts to instead use residual blocks [25] for both generators and discriminators. Following the findings from StyleGAN2 this work also uses style blocks with modulated convolutions to generate individual images in the video sequence.

3.4 Recurrent Neural Networks

Recurrent neural networks (RNNs) were first introduced in 1985 by "Learning Internal Representations by Error Propagation" [45]. Their purpose is to work on sequences of

variable length by applying the same operations and storing data in a so-called hidden state $h(t)$. Their basic update rule is shown in Equation 3.12. Here W and U are learnable weight matrices, b is the bias vector and $x(t)$ is the current input at time step t. In order to speed up computation both weight matrices are usually concatenated into a single matrix and the hidden state, input and bias are concatenated into one large vector. The output of a classic RNN can thus be computed with a single matrix-vector multiplication followed by a non-linearity such as $tanh()$.

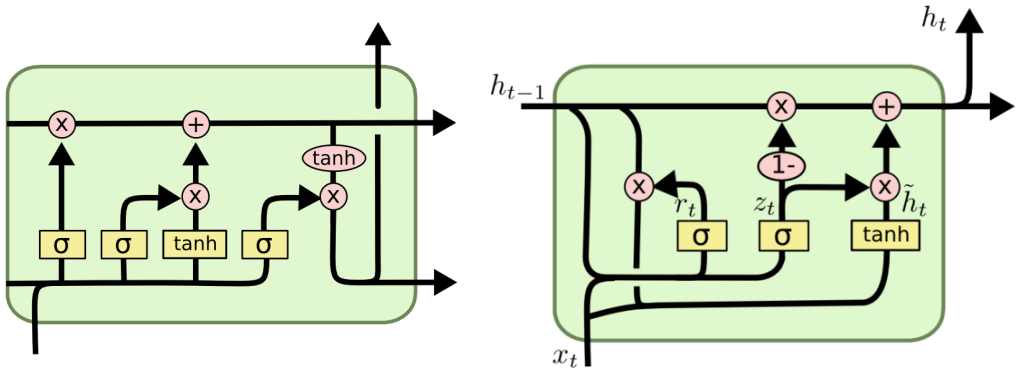
$$h(t) = tanh(b + W \cdot h(t-1) + U \cdot x(t)) \tag{3.12}$$

Classical RNNs often tend to suffer from vanishing or exploding gradient problems when unrolled over long sequences. Two common extensions of RNNs, that help in alleviating this problem, are LSTMs and GRUs. They are explained in the following sections. LSTMs and GRUs operate on input and hidden state via a series of so-called gates, which combine each input with a learnable linear layer before adding them together and passing them through an activation function. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling” [46] provides a detailed explanation and comparison of both architectures.

LSTM [47] stands for "Long Short-Term Memory". It follows the idea of using skip-connections as in ResNet [25] to provide better gradient flow. This is achieved by having two states per cell: a hidden and a cell state. Before applying an update to the hidden state, the input is first combined with a the cell state. The hidden state is then updated using a forget and an input gate. Finally the new cell state is computed from the updated hidden state and the input via the so-called output gate. Note the use of sigmoid activation functions here can be interpreted as choosing whether to remember certain information or whether to replace it with new input information. A graphical representation of LSTMs is shown in Figure 3.5a.

GRUs [48] or "Gated Recurrent Units" were developed after LSTMs. They follow the same principle of skip-connections, but simplify the operations in each cell. The key idea behind gated recurrent units is to compute the current hidden state $h(t)$ by interpolating between the previous hidden state $h(t-1)$ and a candidate hidden state computed from $h(t)$ and the input $x(t)$. Their diagram is shown in Figure 3.5b. Due to their reduced complexity, GRUs are usually preferred in tasks with two or more spatial dimensions. The use of linear layers in the gates of LSTMs and GRUs becomes intractable with larger image resolutions. This leads to the concept of convolutional RNNs, which replace the linear layers with convolutional ones. ConvLSTM [29] and ConvGRU [30] are the two most famous examples of such networks. They are both widely used when processing image sequences. Due to its lower complexity ConvGRU is usually preferred for simple changes over time.

A newer form of recurrent units were designed specifically for modeling motion on image sequences. They are collectively referred to as transformation-based recurrent units and work by introducing motion between frames via a differentiable warping function. “SDCNet: Video Prediction Using Spatially-Displaced Convolution” [49] categorize these functions into *vector-based* and *kernel-based* operations: vector-based and kernel-based. Vector-based



(a) LSTM

(b) GRU

Figure 3.5: Wiring diagrams for LSTMs [47] and GRUs [48]. Sources: "colah.github.io/posts/2015-08-Understanding-LSTMs" and "primo.ai/index.php?title=Gated_Recurrent_Unit_(GRU)"

operations predict the coordinates for a bilinear sampling operation applied to the input of the recurrent unit [50]. These approaches have also been used for optical flow prediction, as the coordinates correspond to a 2D optical flow field in latent space [51, 52]. Kernel-based methods, on the other hand, predict a local sampling kernel [53, 54, 55, 49]. The output pixel is then computed by the weighted sum of the inputs. This kernel can be shared across spatial locations, which means it applies the same motion at all locations. To model varying motion either multiple kernels have to be computed and weight based on their location [53]. Alternatively a separate kernel can be predicted for each location. "Unsupervised Learning for Physical Interaction through Video Prediction" [54] compares these approaches. The paper finds them to perform similarly on the BAIR robot pushing dataset [10]. When comparing the two approaches, the authors of SDCNet [49] find, that kernel-based approaches can only model small motion, limited by the kernel size. Vector-based approaches do not fall to this limitation and can produce large motion, however they often suffer from noise artifacts. The paper thus combines the two approaches by computing a motion vector and a kernel and applying the kernel at the displaced location given by the vector. The resulting model is able to predict large motion while still maintaining sharp images. "Deep Learning for Precipitation Nowcasting: A Benchmark and A New Model" [31] introduce TrajGRU, which warps the hidden state based on an implicitly predicted flow field. The warped hidden state is then presented as input to the gates of a standard ConvGRU. "Transformation-based Adversarial Video Prediction on Large-Scale Data" [8] extends the idea of TrajGRU to a kernelized version called K-TrajGRU and formulates several new transformation-based spatial recurrent units called TSRU. The diagrams for those are shown in Figure 3.6. In their experiments show, that $TSRU_p$ slightly outperforms the other TSRU variants, all of which in turn outperform the K-TrajGRU baseline [8].

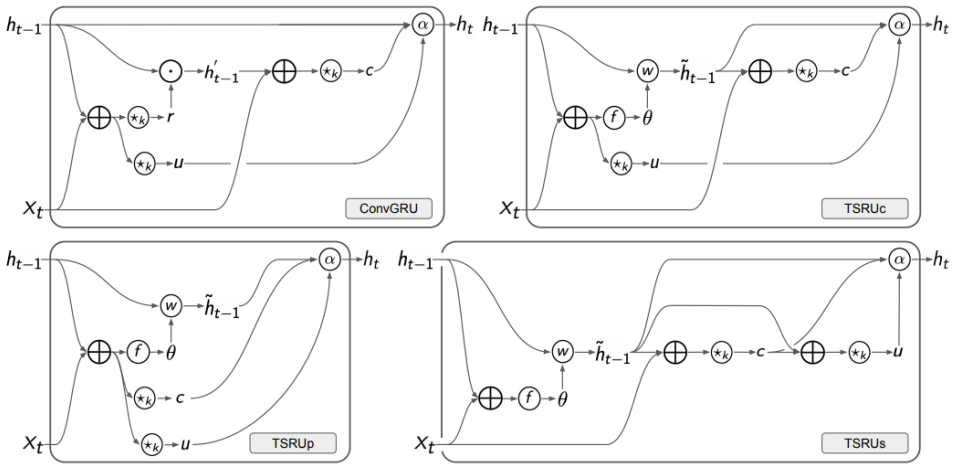


Figure 3.6: Transformation-based spatial recurrent units introduced for TrIVD-GAN. Top-left shows the classical ConvGRU architecture. Source: "Transformation-based Adversarial Video Prediction on Large-Scale Data" [8]

3.5 Generator Architecture

State-of-the-art methods follow the principle of splitting the spatio-temporal generator into a temporal generator and an image generator. This concept was introduced by TGAN [28]. While TGAN uses a 1D convolutional neural network to generate the temporal latent codes, modern methods generate them by unrolling some form or recurrent neural network for temporal effects [6, 9, 7, 8]. The image generator is then built using standard 2D convolutional blocks such as residual blocks [25] to generate each frame individually or building blocks from well studied image generation architectures[7, 8, 56]. As mentioned in section 3.3 the basic building block of the generator architecture introduced in this work is the updated style block introduced by StyleGAN2 [24].

Both DVD-GAN [7] and TriVD-GAN[8] utilize a feature pyramid architecture, meaning that a recurrent network is not only unrolled at the lowest resolution and then developed into a frame, but additionally RNNs are used at each intermediate resolution and added to the output of the up-sampling image generation blocks. This is can be seen in Figure 2.2 for the DVD-GAN architecture of [7]. The generator architecture presented in this thesis follows this trend and also utilizes a feature pyramid like architecture as it allows to control temporal features at different resolutions.

Figure 3.7 depicts the generator architecture presented in this work. A given input image is first passed through an encoder style network using GResNet Blocks [5]. After each block the resolution is halved using bilinear interpolation. Note, a similar encoder using two GResNet blocks per depth level is proposed in [7] and used by [8]. At the end of the encoder 2 additional blocks followed by a multi layer perceptron (MLP) are used to predict the 256-dimensional style vector which is input into each style block according to Equation 3.10. Intermediate features of the encoder are stored and serve as initial hidden states of the gated

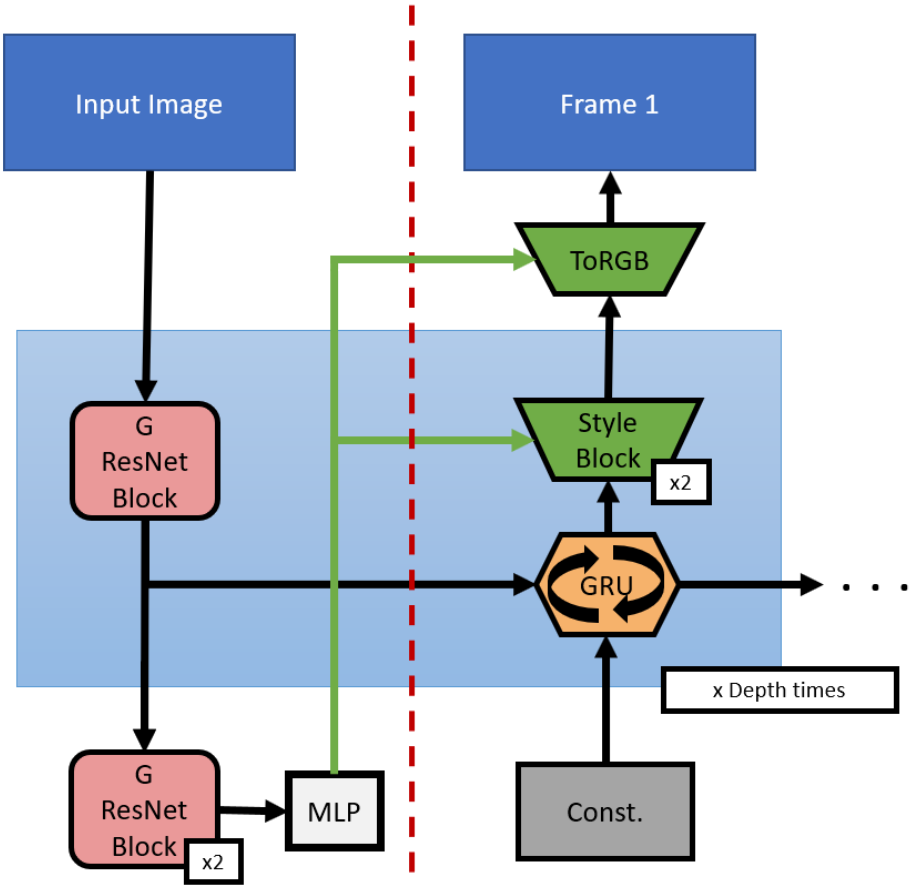


Figure 3.7: Proposed style based architecture for video prediction. The green arrow indicates the predicted style vector. The dashed red line logically separates the generator into an encoder and decoder part. Individual components are explained in the main text.

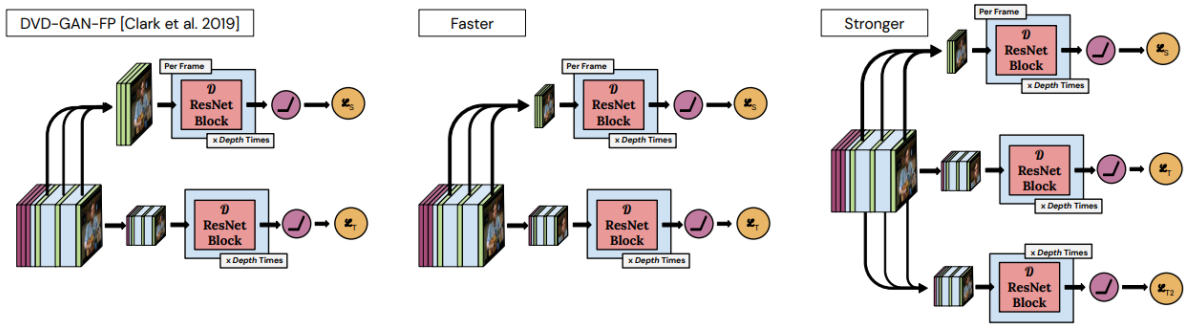


Figure 3.8: Discriminator decomposition proposed by TriVD-GAN. Left: Original DVD-GAN decomposition. Middle: Fast variant using down sampled image and down-sampled and temporally cropped video discriminator. Right: Strong variant using fast variant plus a video discriminator operating on the entire, down-sampled video. Source: "Transformation-based Adversarial Video Prediction on Large-Scale Data" [8].

recurrent units at their respective resolutions. For each further frame the GRUs are unrolled once. Note that GRU here are actually realized using convolutional gated recurrent units (ConvGRUs). The modular architecture allows these units to be easily exchanged for more sophisticated units such as LSTMs or transformation based recurrent units. For each frame the output of the GRUs are up-sampled. Two style blocks (see section 3.3) are then applied per depth level. The output is then either sent into the next level GRU or decoded to RGB using another modulated convolutional layer to produce the final frame. As initial input for the GRU at the lowest resolution serves a learnable constant vector this is similar to the constant input of StyleGAN [4]. As was done by DVD-GAN [7], the depth of the feature pyramid is chosen so that the lowest level outputs a 4×4 resolution latent image. This means, for example, a depth of 5 for a resolution of 128^2. The simple formula to compute network depth given resolution is shown in Equation 3.13. For the sake of simplicity only videos where the resolution is a power of 2 are used. This is very common in image and video research.

$$Depth(resolution) = \lfloor log_2(resolution) \rfloor - \lfloor log_2(lowest\_resolution) \rfloor \tag{3.13}$$

3.6 Dual Discriminators

The discriminators used to train the model presented in this thesis are the same ones used for DVD-GAN [7]. This section explains the basic encoder style behind them and explains the benefits of discriminator decomposition for video generation and prediction. The decomposition's introduced by TriVD-GAN [8] are also discussed.

Both StyleGAN [4, 24] and BigGAN[5] largely agree on the discriminator architecture for image generation. In both papers the architecture is built using a series residual blocks. While StyleGAN sticks with classic ResNet [25] blocks, BigGANs discriminators use DResNet blocks, which are explained in section 3.2. As DVD-GAN and TriVD-GAN are based on the results of BigGAN, they also utilize these blocks. The discriminator used in this work can be seen in Figure 3.8 (Left). It is the same as the one proposed in DVD-GAN [7]. As the figure shows, instead of having a single discriminator working on the entire video, there are two discriminators $\mathcal{D}_S$ and $\mathcal{D}_T$. $\mathcal{D}_S$ is called the spatial discriminator. It operates on a single, full-resolution frame. $\mathcal{D}_T$ is the temporal discriminator and its input is a down sampled version of the entire video. The down sampling operation is usually implemented using an average pooling layer. Such a decomposition was first proposed by MoCoGAN [6]. The advantage of this decomposition is, that instead of having to work on tensor of size $H \times W \times T$, the number of pixels is decreased according to Equation 3.14. Where H, W describe the spatial resolution, T is the number of frames and K is a hyper-parameter that determines how many single frames are evaluated by the per-frame discriminator during each iteration. ϕ_h and ϕ_W describe the down sampling factor used for the video discriminator. In practice K is usually set to 8 and 2×2 average pooling is used to down sample the video for the video discriminator. Additionally the first two convolutional layers in the video discriminator are GResNet blocks using 3D convolutions instead of 2D ones.

$$\#pixels = K \times H \times W + T \times \frac{H}{\phi_H} \times \frac{W}{\phi_W} \tag{3.14}$$

Depending on the dataset and the types of dynamic effects the network is expected to generate one might be forced to not use the initial down sampling. This is in particular necessary when generating landscape videos with flowing water, as the motion in these videos is very fine-scale. Applying average pooling to such videos results in most of the high-frequency effects being lost and the network never gets an incentive to generate those effects. Even in this case, the spatial discriminator is still useful, as it solely focuses on image quality. This is easier to learn and provides a good learning signal for the generator. "Transformation-based Adversarial Video Prediction on Large-Scale Data" [8] proposes additional discriminator decompositions shown in Figure 3.8(Middle) and (Right). The *faster* decomposition also down-samples the input for $\mathcal{D}_S$ and the input of $\mathcal{D}_T$ is only a cropped part of the generated video. This means, that the discriminator cannot detect global inconsistencies. The *stronger* decomposition fixes this by adding a third discriminator operating on a down-sampled version of the entire video, same as $\mathcal{D}_T$ introduced by DVD-GAN[7].

3.7 Network Details

The following paragraphs fully specify the proposed architecture by explaining how to compute the size of each layer in terms of feature channels, as is usually done for convolutional neural networks. To determine the number of feature channels in each layer the notation introduced by [5] is used. This specifies a constant for each layer and then uses a global

channel multiplier *ch* to compute the actual number of features as product of the constant and *ch*. For example: given a constant list $[1, 2, 4, ...]$ and $ch = 8$ the resulting channels per layer would be $[8, 16, 32, ...$ Each layer here refers to a gated recurrent unit followed by two style blocks as shown in Figure 3.7.

Generator: For a 64×64 the proposed constants are $[8, 4, 2, 1]$ with a channel multiplier of 16 and for 128×128 they are $[8, 8, 4, 2, 1]$ with channel multiplier of 32. The constants are given from lowest to highest resolution level and are used for both encoder and decoder. The final 2 GResNet blocks of the encoder have a width of $[8, 4]$ and the MLP consist only of a single layer going from the flattened output of the GResNet blocks to the dimension of the style vector, which was chosen to be 256. The noise input for the style blocks is drawn from a unit Gaussian $\mathcal{N}(0, I)$ and scaled by a learnable weight as shown in Figure 3.4 (d). Each GRU is built up of a single ConvGRU [30] cell, with the number of input and hidden features being equal to the number of features in the following style block. Unless specified otherwise all convolutional layers are same convolutions with a kernel size of 3×3.

Discriminator: The channel lists for $\mathcal{D}_S$ and $\mathcal{D}_T$ are [2, 4, 8, 16, 16] for 64×64 resolution and [1, 2, 4, 8, 16, 16] for 128×128 resolution, same as in DVD-GAN [7]. The channel multiplier *ch* for $\mathcal{D}_S$ is 16 and *ch* for $\mathcal{D}_T$ is 64 when using 128×128 resolution. For 64×64 resolution both discriminators use a *ch* multiplier of 32. The reduced *ch* multiplier at higher resolution is chosen mainly to reduce memory usage. Experiments show, that the fine detail quality does not suffer too much. Since predicting landscape videos requires some very fine motion, no down sampling was used for $\mathcal{D}_T$.

3.8 Training

Training neural networks for video generation is very compute and memory intensive. As a result the hyper parameters were not found via an exhaustive search, but instead either well known or fairly conservative values were used. Next to traditional parameters, the important question of how many frames to generate in a training sequence is discussed in detail. A technique to train with larger batch sizes than the given hardware allows is explained in the final paragraph.

The following settings are used to train the proposed model on 25 frame long clips from the custom landscape dataset at a resolution of 128×128: A common learning rate of 0.0003 with a batch size of 8 is used for all networks. The channel modifiers as explained in section 3.7 for $\mathcal{G}$, $\mathcal{D}_S$ and $\mathcal{D}_T$ are $[32, 16, 64]$, putting a large focus on the video discriminator. The spatial discriminator $\mathcal{D}_s$ is conditioned on the input frame, as this was found to speed up and stabilize the training process significantly. No down-sampling is used for $\mathcal{D}_T$ when working with landscape images, as these videos contain mostly fine scale motion. A 5-level feature pyramid is used with a single Gated Recurrent Unit at each layer. The network is trained by a Wasserstein loss with gradient penalty $\mathcal{L}_{WGAN-GP}$ as defined in Equation 3.8. The losses for the generator are weighted by $\lambda_S = 1$ and $\lambda_5 = 5$. Two discriminator training passes are

run for each generator training pass, as is common practice [19, 17, 5]. Gradients are clipped to $[-2.5, 2.5]$ in order to avoid exploding gradient problems which can sometimes recurrent architectures. This is mainly a safety precaution so training is not ruined by a bad outlier. A cosine annealing learning rate scheduler [57] is used after the 10th epoch. All models are trained until the Wasserstein loss indicated convergence, which resulted in roughly 1.000.000 iterations when training at a resolution of 128×128. A single *Nvidia GTX 1080ti* is used to train each model.

$$\mathcal{L}(\mathcal{G}, \mathcal{D}_S, \mathcal{D}_T) = \lambda_S \cdot \mathcal{L}_{WGAN-GP}(\mathcal{G}, \mathcal{D}_S) + \lambda_T \cdot \mathcal{L}_{WGAN-GP}(\mathcal{G}, \mathcal{D}_T) \tag{3.15}$$

Sequence Length: For each input sample a sequence of 25 frames was predicted. The length of the predicted training sequences has to be considered very carefully, as each unroll of the recurrent architecture consumes additional memory. On the other hand the temporal Discriminator $\mathcal{D}_T$ is found to provide better gradients and the network trains faster with larger sequence length. During development sequences shorter than 15 frames resulted in $\mathcal{D}_T$ not providing any useful gradient information. This in turn causes the generator to get stuck in a local minimum where it only generates constant videos, since this is in theory enough to satisfy the requirements of the spatial discriminator $\mathcal{D}_S$. This means, that the number of frames in each training sequence has to be large enough to allow the temporal discriminator to learn a meaningful criterion, while not overstepping the available memory budget. When training this or similar models on commodity hardware with limited memory capacity a number of frames between 20 and 30 is recommended.

Gradient Accumulation: Memory is often the limiting factor when training deep neural networks. The largest amount of memory is used by the intermediate values which are used for back-propagation. Each sample in a mini-batch requires its own set of intermediate values to be stored, making larger batch sizes very memory consuming. Large batch sizes, however, are often required to reduce the noise of mini-batch gradients and allow for stable training. Additionally, algorithms such as Batch Normalization [43] rely on larger batch sizes to work at all. This leads to specialty hardware such as Tensor Processing Units (TPUs) [12], which offer large amounts of high bandwidth memory, or concepts such as distributed training. When using more constrained training hardware, such as commodity GPUs, large batch sizes can still be achieved by accumulating and averaging gradients over multiple backward passes. Gradient accumulation requires at least enough memory to perform a forward and backward pass on a single sample to work. Let $\mathcal{L}$ be a loss function, NN an arbitrary function, such as a deep neural network and let $\{x, y\}^N$ be a mini-batch of size $N = J \cdot K$ and K be the maximum number of samples the given hardware can process in parallel. Then Equation 3.16 shows, how gradient accumulation can be used to compute the mini-batch gradient $\nabla\mathcal{L}(\{x, y\}^N)$. The constant factor $\frac{K}{N}$ is specifically drawn into the gradient operator to indicate a re-scaling of the loss function before back-propagation. Otherwise the gradients would have to be scaled

explicitly.

$$\nabla \mathcal{L}(\{x,y\}^N, NN) = \sum_j^J \nabla(\frac{K}{N}\mathcal{L}(\{x,y\}^{j\cdot K,\dots,j\cdot K+K-1}, NN)) \tag{3.16}$$

Equation 3.16 only holds true, if the network NN processes each sample individually. Common normalization techniques, such as Batch Normalization [43] violate this constraint, as they compute the mean and variance across the entire mini-batch during the forward pass. To alleviate this problem it is recommended to use a lower momentum m when computing batch statistics, effectively giving more weight to the previously accumulated statistics than the current ones. A common momentum value for BatchNorm in a standard setting is $m = 0.1$, when using accumulated gradients this value should be reduced to $m = 0.01$. Note, that the definition of momentum in Batch Normalization is different from the conventional notion, as well as the one used for optimizers. The update rule for BatchNorm with momentum is given in Equation 3.17, where x_t are the running statistics and $\hat{x}$ describes the statistics computed in the current forward pass.

$$x_{t+1} = (1-m)j \cdot kx_t + mj \cdot k\hat{x} \tag{3.17}$$

3.9 Custom Dataset Gathering

Gathering datasets is often the most tedious and time-consuming part of developing new machine learning algorithms. While there are some general video datasets available, such as the ones listed in section 2.5, datasets for artistic works are often kept private and only available to the authors of the corresponding paper. Most of the work on landscape images also focuses on large scale changes and often fails, when trying to generate fine detail [32]. As such, a custom dataset was created for this work, which focuses on fine scale motion; mainly flowing water such as rivers, waterfalls and beaches. About 500 long videos with a resolution of 640×360 were gathered from YouTube [34] via a search using the *YouTube Data API* [58]. The videos were automatically cut into short clips and further processed to detect bad sequences, such as static videos before being manually inspected. This process is described in the following paragraphs.

The videos for this dataset are gathered from a single YouTube Data API [58] search of the phrase *"river relaxing"*. This phrase in turn has to be found by manually trying some search phrases and looking at the first few results. Searching for simply "river" for example will result in mostly music videos, which will cause more manual work for the final pass. So it is recommended to try and add modifiers to the query that minimize unwanted results while still finding a sufficient amount of matches. Once the videos are downloaded, they are split into short video clips similar to [38]. This drastically reduces data size, as some of the downloaded videos were several hours long. Each video is split into up to 20 short 10 second clips. Since some videos show the same scene for hours, this means that there are some very

similar, but not completely identical clips in the final dataset. Once all videos are cut, the per pixel temporal variance is computed for each clip according to Equation 3.19. Here $\odot$ denotes the pair-wise product and T describes the number of frames, so that clip(t) refers to the frame at time t. The mean is computed according to Equation 3.18. The resulting per-pixel values are averaged and thresholded using a value of 0.001, with the original images being in range $[0, 1]$. This detects and discards clips with very little to no movement in them. This could also be used to automatically detect hard cuts, however a threshold for this is much harder to determine and by inspecting some of the processed clips, it was found that most transitions use some sort of fade effect, which cannot realistically be detected by such simple methods. Even though there are more sophisticated methods to detect for example transition effects, ultimately a manual review will likely still be required. These clips are thus left to be detected and removed during the manual selection process as described in the following paragraph.

$$\mu^{temp}(clip) = \frac{1}{T}\sum_{t}^{T} clip(t) \tag{3.18}$$

$$\sigma^{temp}(clip) = \frac{1}{T}\sum_{T}^{t}(clip(t) - \mu^{temp}(clip)) \odot (clip(t) - \mu^{temp}(clip)) \tag{3.19}$$

Once the simple automatic pipeline is finished, a manual pass is still necessary, as the dataset might still contain videos clips that have cuts or transition effects in them. Clips can also simply contain the wrong content. To find these unwanted clips, a human has to view the video and determine whether it should be kept or discarded. This is a very tedious and time consuming process. In order to speed it up, instead of showing one video after the other, the simple annotation system developed for this work shows 25 videos at a time in a 5-by-5 matrix. Since most clips turn out to be viable the annotator only has to select the ones that are unwanted with the mouse cursor and once the selection is complete hit the enter key to continue to the next batch. A picture of this process is shown in Figure 3.9. The Visdom [59] server used to display the videos does not record user feedback on video windows, so the selection window only shows the first frame of every video, while the videos themselves are played in a separate window. It is recommended to use two physical computer screens when processing the clips in order to view the video matrix in full screen while still having easy access to the selection window. After the a final manual filtering pass over the entire dataset, a total of 7480 training and 206 test sequences were kept. Compared to other datasets, this is a fairly small sample size. The dataset also shows a huge diversity compared to previous datasets used for landscape video prediction. This means, that a model trained on this dataset would have to be quite efficient or be equipped with strong inductive biases in order to avoid over fitting and show proper results on the test set. Additionally the gathered dataset contains some outliers, for example under-water videos. These tend to cause larger-than-normal error values during training, which could potentially cause exploding gradients during back propagation. A discriminator might, for example, predict an outlier to be a fake sample. The dataset was initially planned to be expanded by several such searches, which was however

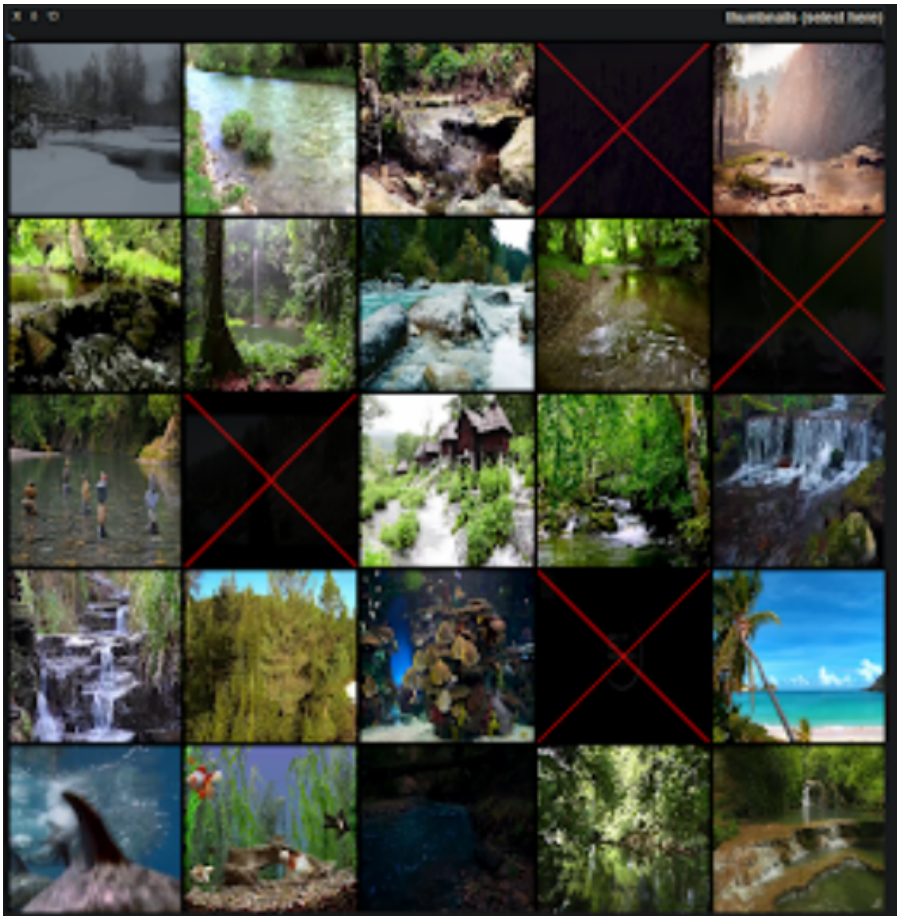


Figure 3.9: Manual video selection for dataset creation. 25 videos are displayed at once in a 5-by-5 matrix and bad samples are dismissed via mouse click. Dismissed samples are greyed out and overlaid with a red 'X'. A Visdom [59] server is used to display the images and videos and capture annotator input.

scratched after an update to YouTube´s JavaScript code ended up breaking the third party video download library, which is no longer actively maintained.

3.10 Down scaling DVD-GAN

Some of the architectures considered in this work were trained on a large cluster of tensor processing units (TPUs) [7, 8, 12] in their original publications. Their representatives in this work were trained on a single *Nvidia GTX 1080ti*, same as the architecture presented here. This section describes the challenges and decisions when adapting models designed to run on large-scale hardware. The following quote from "Adversarial Video Generation on Complex Datasets" [7] shows the amount of hardware resources used to train a single model:

> "Each DVD-GAN was trained on TPU pods [12] using between 32 and 512 replicas

> [...] Video Synthesis models are trained for around 300,000 learning steps, whilst Video Prediction models are trained for up to 1,000,000 steps. Most models took between 12 and 96 hours to train."

While the paper made great strides in video generation, the cost of training, let alone developing similar models is not affordable to everyone. All models presented in this work were trained on a single *Nvidia GTX 1080ti* with 11GB of video memory. In order to compare the proposed architecture to a meaningful baseline, the DVD-GAN model was down scaled to fit onto the available hardware. This smaller version of DVD-GAN will from here on be referred to as DVD-GAN-S. The number of parameters used in the DVD-GAN architecture is easily manipulated via the channel multiplier described in section 3.7. section 3.7 also mentions, that the channel multipliers chosen for this work are $[32, 16, 64]$ for $\mathcal{G}$, $\mathcal{D}_S$ and $\mathcal{D}_T$. The same channel multipliers were used to produce the DVD-GAN-S baseline model. The classical DVD-GAN [7] model on the other hand uses a multiplier of 96 for all networks when generating videos at 128×128 resolution. Note that the basic building blocks of this work are mainly Style Blocks (see section 3.3, which only have one convolutional layer, whereas DVD-GAN uses GResNet Blocks section 3.2 which contain two 3×3 convolutional layers and an additional 1×1 convolutional layer. While creating similar-size models is easy thanks to the channel multiplier notation, it is also important to consider batch size. The batch size for training deep neural networks is usually chosen to maximize memory utilization. Memory, however, was the main bottle neck for this work and often only 1 to 4 samples could be processed in parallel by the given hardware. section 3.8 shows how gradient accumulation can be used to process larger batch sizes. The model introduced in this work is capable of being trained with a fairly low batch size of 8. Since DVD-GAN relies heavily on batch normalization [43] a larger batch size is required. The original paper suggests using a batch size of 512, for the down-scaled version this work found a batch-size of 128 to be the minimum required size to properly train the network without collapsing. This still imposes a slow-down, as the network weights are only updated every $128/4 = 32$ backward passes when using gradient accumulation and processing 4 samples in parallel.

4 Results

The metrics used to quantitatively evaluate the proposed architecture are explained in section 4.1. An exhaustive comparison against existing generative models for video is difficult, as most papers do not share a common metric or even a common dataset. The proposed architecture was thus mainly compared to a down-scaled version of DVD-GAN explained in section 3.10. In section 4.2 some rolled-out predicted video sequences are shown frame-by-frame and compared to state-of-the-art baselines. For videos please see the provided supplementary material provided on `mirjang.github.io/mt_videoprediction`.

4.1 Metrics

Comparing the performance of generative models is still an active area of research [60, 61]. To date there is no single metric that can objectively quantify GAN performance. Most of the available metrics are designed to capture the two main values for generative models: sample quality and diversity. Sample quality measures the similarity between a fake sample generated by the model and the real samples in the dataset. This alone is not enough for measuring the quality of the model. GANs often suffer from (partial) mode collapse, where the generator is only able to generate a small set of samples. In the worst case the generator completely ignores the input and only a single sample is generated. It is thus important to measure the diversity of the samples generated by the generator.

Inception Score: Inception Score (IS) [16] computes quality by looking at the ability of a InceptionNet [62] to classify a set of generated samples. This is measured against the overall distribution of classes, which should ideally be uniform. Inception Score does, however, not measure the diversity within a given class and can thus be fooled by providing a single high quality sample per class. This implies that inception score cannot identify a mode collapsed generator. "A Note on the Inception Score" [61] raises additional concerns on the use of Inception Score for generators working on datasets other than ImageNet.

Fréchet Inception Distance: FID [17] uses the intermediate features of a pretrained InceptionNet for a set of real and fake samples and fits a multivariate Gaussian distribution onto them. The distance between real and generated samples is the computed according to Equation 4.1. This means, that FID also accounts for inter class diversity and can thus factor in mode collapse. The FID scores for this work were computed using the PyTorch implementation provided by [63]. An extensions to FID for videos, called Fréchet Video Distance (FVD) was introduced in "Towards Accurate Generative Models of Video: A New

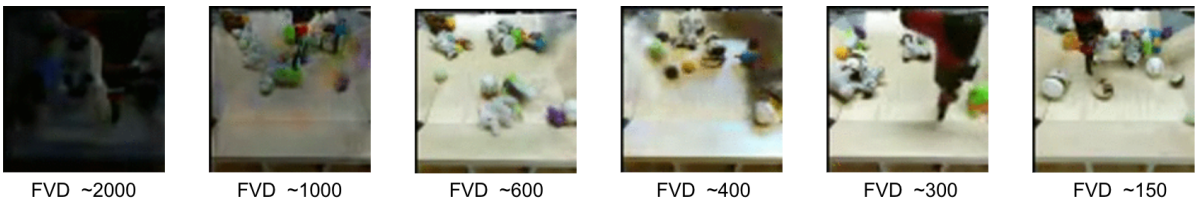


Figure 4.1: Fréchet Video Distance (FVD) for various models on the BAIR robot pushing dataset [10]. Source: "Towards Accurate Generative Models of Video: A New Metric & Challenges" [64].

Metric & Challenges" [64]. It is based on an Inflated 3D ConvNet [65] trained for action recognition on the Kinetics[11] dataset. Other than that it follows the same formula as the FID score. This work uses the TensorFlow implementation to compute the FVD score introduced with the original paper [64]. Figure 4.1 shows video sequences of varying quality ranked by FVD on the BAIR robot pushing dataset [10].

$$FID(real, fake) = \left\|\mu_{real} - \mu_{fake}\right\|_2^2 + Tr(\Sigma_{real} + \Sigma_{fake} - 2(\Sigma_{real}\Sigma_{fake})^{\frac{1}{2}}) \tag{4.1}$$

The models are compared on several datasets. The BAIR robot pushing dataset [10] is used for to evaluate general video prediction performance, as is a very common dataset in video generation and prediction and its lower resolution allows for faster training. The sky time-lapse dataset [38] is one of the few datasets publicly available that contains landscape videos. It is thus used to measure performance on the more classical landscape video prediction task. Lastly, methods are trained on the custom dataset created in this master's thesis. This dataset is comparably small and contains a lot of diverse videos with difficult scene dynamics, which makes it the most difficult to train on. For additional information on common video datasets see section 2.5.

4.2 Baseline Comparison

Figure 4.2 shows sequences generated by the introduced algorithm (top row for each pair) and DVD-GAN-S (bottom rows). Both approaches manage to reconstruct the static background, however, DVD-GAN-S clearly struggles when generating the moving robot arm. The arm in the DVD-GAN-S sequences is often distorted and sometimes even large artefacts appear, where one might expect to find the arm. Note, that the motion in the dataset sequences can be fairly large. Table 4.1 shows a quantitative evaluation of the generated sequences. For this comparison sequences of 25 frames were generated from a single input frame. The table shows a significant performance boost in terms of FID. This can also be seen when comparing the results visually by looking at Figure 4.2. The comparably bad inception score when evaluating on the custom dataset is likely due to problems with the metric itself [61].

	BAIR		Custom Dataset	
	IS (↑)	FID (↓)	IS (↑)	FID (↓)
DVD-GAN-S	10.68	81.02	**29.27**	194.30
Style-Based	**14.68**	**41.47**	13.07	**108.96**

Table 4.1: Comparison between the proposed style-based architecture and the down-scaled version DVD-GAN-S. Metrics used are Inception Score (IS) and Fréchet Inception Distance (FID). Videos on the BAIR dataset have a resolution of 64×64. Custom dataset videos have a resolution of 128×128.

Comparison for video prediction methods is often difficult, as there us no one dataset or metric that is used by a meaningful amount of papers.

4.3 Image Quality Over Time

As the recurrent neural network is unrolled, a decline in image quality is expected for each successive frame. The decaying quality is measured by computing the FID score for each time step. In this setup the architecture was trained on the BAIR robot pushing dataset [10]. The model was trained trained to generate 30 frame sequences and at test time 150 subsequent frames are predicted for each sample in the validation set. The result is plotted in Figure 4.3 and clearly shows the expected quality decay over time. The plot also shows periodic dips in the FID score. Visual investigation of the predicted videos reveals, that these dips occur when the robotic arm returns to its initial position, at which point the network just has to reconstruct the input frame. After about 80 frames the predicted sequence becomes also very repetitive and the network seems to get stuck in a loop of predicting the same sequence over and over. This could also be an artifact of the dataset, where the robot arm usually only moves within a small area. In Figure 4.4 the proposed model is trained on the sky timelapse dataset [38]. The resulting videos do achieve realistic looking motion of clouds in the first few frame. As the video progresses, small but visible GAN artifacts start to manifest at around 15 to 20 frames in some of the samples. Most sequences also start to become quite blurry.

4.4 Generalization to GauGAN

One motivation for this work was to animate landscape images created with GauGAN [13]. GauGAN is an image-to-image translation network, that turns manually drawn semantic segmentation masks into photo realistic landscape images. After a short learning period it takes only a few minutes to create good looking images with GauGAN. Some landscape images drawn with GauGAN are shown in Figure 1.2. In Figure 4.5 some sequences generated by training the proposed algorithm on the custom dataset created for the task of "bringing landscape images to life". The dataset spans a large space of landscape videos containing water bodies. When using it for training, the resulting model manages to achieve decently

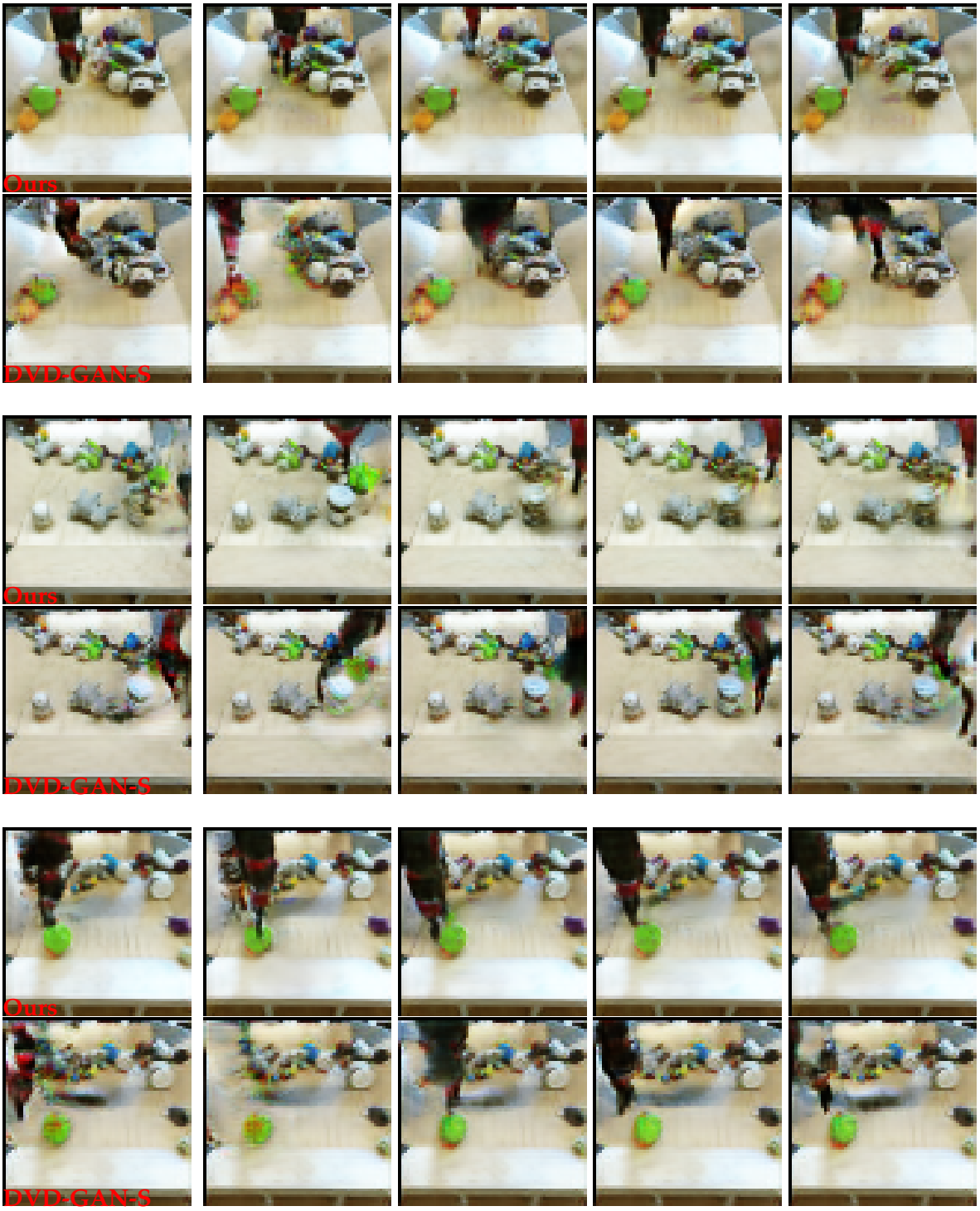


Figure 4.2: Comparison between the proposed architecture and DVD-GAN on the BAIR robot pushing dataset [10]. Top rows were generated by this work, bottom rows by DVD-GAN.

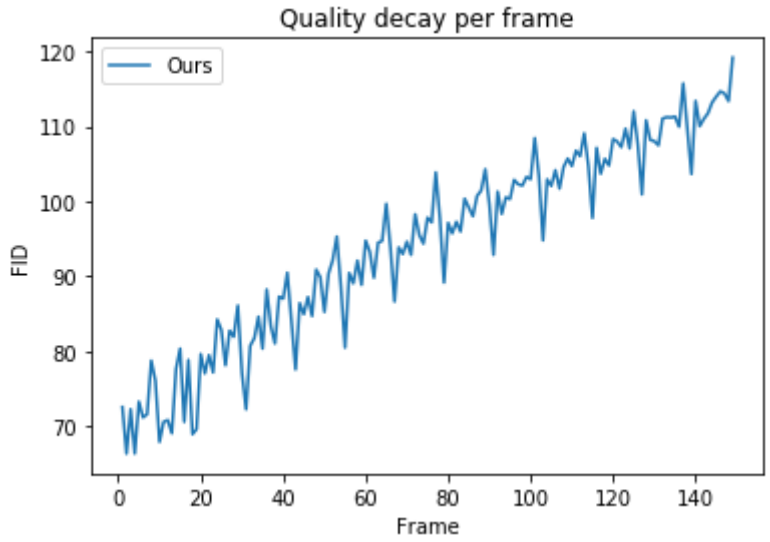


Figure 4.3: Per frame FID scores for sequences predicted from the BAIR [10] validation set using the proposed architecture.

realistic results on the validation set. The same model is applied to landscape images created with GauGAN, that contain some form of water body. Compared to the predictions on validation set images such as the ones shown in Figure 1.1, the sequences generated from GauGAN images in Figure 4.5 show a significant decrease in quality. Individual images sometimes show artifacts and the full videos often show some unrealistic motion upon closer look. In most sequences the model still manages to identify the bodies of water and add dynamic effects. In the worst failure cases these effects are also added to other scene elements, such as sky or grass. None of the generated videos show a constant image, which would seem like a common failure mode. This is likely to the strong weighting of the temporal discriminator explained in section 3.8.

4.5 Limitations

Most of the limitations of the novel architecture come from the restrictions posed by the available hardware and could likely be overcome with larger amounts of compute resources. Nonetheless they are listed here for completeness.

Resolution: The most notable limitation, of course, is the spatial resolution of the generated videos, which is at most 128×128. This is a fairly comparable resolution for state-of-the-art video generation and prediction models. DVD-GAN [7] presents videos generated at a maximum resolution of 256×256. A model based on the proposed architecture was trained to generate 256×256 resolution videos using a significantly reduced number of parameters. The channel multipliers (see section 3.7) for $\mathcal{G}$, $\mathcal{D}_S$ and $\mathcal{D}_T$ used for this model were 8, 8, and 32. A down-sampling factor of 2 was used for the temporal discriminator. Some resulting

Figure 4.4: Several sequences generated from a model trained on the sky timelapse dataset [38]. Each row was generated from a single input frame of the validation set. Frames 1, 5, 10, 15 and 20 of the generated sequence are shown.

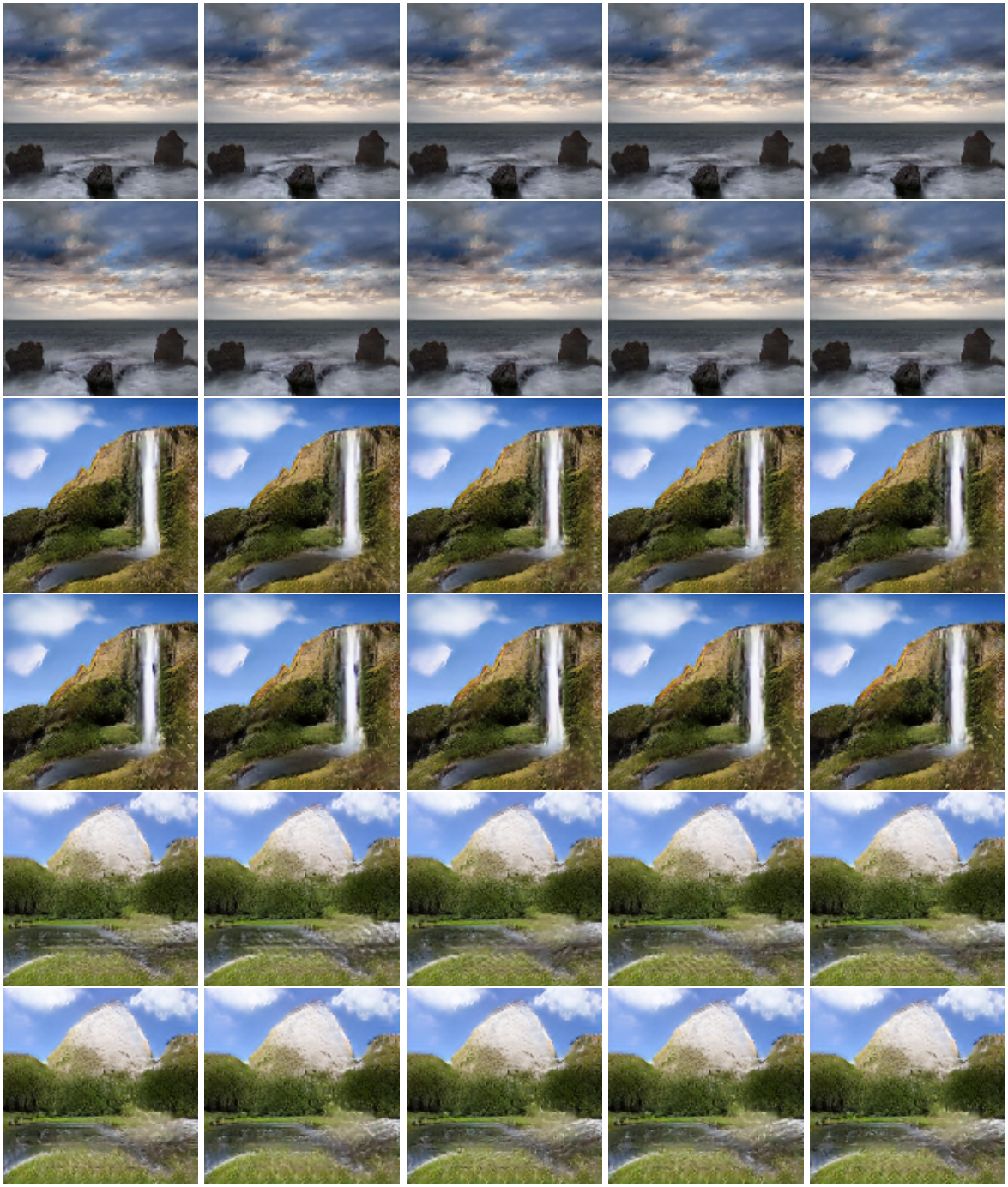


Figure 4.5: Landscape videos predicted from GauGAN [13] images. The first frame in each sequence is the input landscape image. There is still a significant decrease in video quality compared to the validation set images of the custom dataset. The full sequences for all 25 landscape images that were drawn for this work is shown in the supplementary website `mirjang.github.io/mt_videoprediction/`.

Figure 4.6: Attempt to generate videos at a resolution of 256×256. The number of parameters for the generator had to be reduced drastically, resulting in visible artifacts in the water (first sequence) and around the tree leaves (second sequence).

video sequences are shown in Figure 4.6. One can clearly see artifacts in the shown sequences. The severely reduced generator in this work had only about 1 million parameters to work with, which is nothing in terms of modern deep learning models and certainly not enough for working on videos, however, the total amount of memory required during training was still at about 9GB. For this experiment a sequence length of 15 frames was used. I most cases the network still learns to separate static from dynamic video regions. Static regions are reconstructed almost perfectly. This is likely due to the feature-pyramid style architecture in which skip connections allow for static high-resolution features to "bypass" deeper parts of the network. Dynamic regions on the other hand usually show blurry and unrealistic motion.

Resolution for Landscape Videos: The resolutions shown in this work are fairly common for general video generation and prediction works. When generating videos from landscape images, however, the resolution achieved by this work is far from state-of-the-art works, such as the ones mentioned in section 2.4. The algorithms presented in these papers can generate videos of up to 1024×512 [32, 14]. This is thanks to their simplified motion models, which are usually not represented by a recurrent neural network and can even go as far as not having learnable components at all [14], meaning frames only have to be judged individually or in pairs by the discriminator. As a consequence lots of memory is saved and can be put to use for generating higher resolution images. The down side of these simple motion models, of course, is that they only work on certain video prediction tasks. “Animating Landscape: Self-Supervised Learning of Decoupled Motion and Appearance for Single-Image Video Synthesis” [32] for example demonstrates how such an algorithm can fail when trying to predict chaotic, non-uniform motion in the form of a crashing wave.

Artifacts: Some generated samples show large, curvy artifacts. These artifacts propagate in a wave-like fashion across the image over time or move back and forth across a small area. They are likely a result of low model capacity. Other times a small, blurry region is generated, which quickly extends across a large portion of the image. Such a failure case is shown in Figure 4.7. These severe failures do, however, mostly occur during training and tend to indicate that the model has not converged fully. They can thus be avoided by simply training the model for longer periods of time. When working with landscape images, in a few cases, the model confuses static and dynamic regions and tries to add motion to, for example, blue sky instead of water when trained on the custom dataset. This is likely a result of a too sparse dataset. Additional data would make it easier for the network to learn to distinguish between static and dynamic areas in an image. Alternatively, segmentation masks could be used as additional inputs to the network or as hard constraints only allowing changes in certain regions. Semantic segmentation techniques could be used to compute such a mask. This was for example done to create the training data for GauGAN [13]. The constraints imposed by such a mask would, however, disallow or at least discourage some forms of motion, such as the one in the BAIR dataset, where a robot arm moves across the image. All of these artifacts are especially prominent when looking at sequences generated form GauGAN [13] images as shown in Figure 4.5. Comparing the GauGAN images to the custom dataset, it is easy to see, that GauGAN landscapes usually depict a larger scene. The custom dataset videos are usually shot close up. It is thus easy to conclude that the artefacts on GauGAN images are caused by domain mismatch, as the GauGAN images are too far away from the training set distribution. Their quality could thus be improved by applying stronger regularization. This might however lead to further problems, as the previous paragraph already determined, that the models likely suffer from to low capacity and regularization will further worsen this problem.

Motion Complexity: While Figure 4.2 clearly outperforms the DVD-GAN-S baseline, it still shows some distortion and color changes in the robot arm. This indicates, that the model does not perfectly capture the concept of moving rigid objects. This is not really a surprise, as the underlying motion model is a simple convolutional gated recurrent unit, that does not enforce any kind of object motion. Other works try to solve this problem by more complex motion models that either solely rely on object motion or combine object motion and other dynamic effects [31, 6, 8].

Training Time: As mentioned in section 3.8, training the proposed algorithm takes about 1,000,000 iterations with a batch size of 8 samples. For each mini-batch the samples are computed in sequence and the gradients are accumulated. When training on a single *Nvidia GTX 1080ti* (on a busy student server) this usually results in training times of up to 14 days for models trained at a resolution of 128×128. Training the proposed architecture on the BAIR robot pushing dataset [10], which is limited to a 64×64 resolution, usually takes about 2 days. Long training times can of course be *easily* remedied by adding more compute resources.

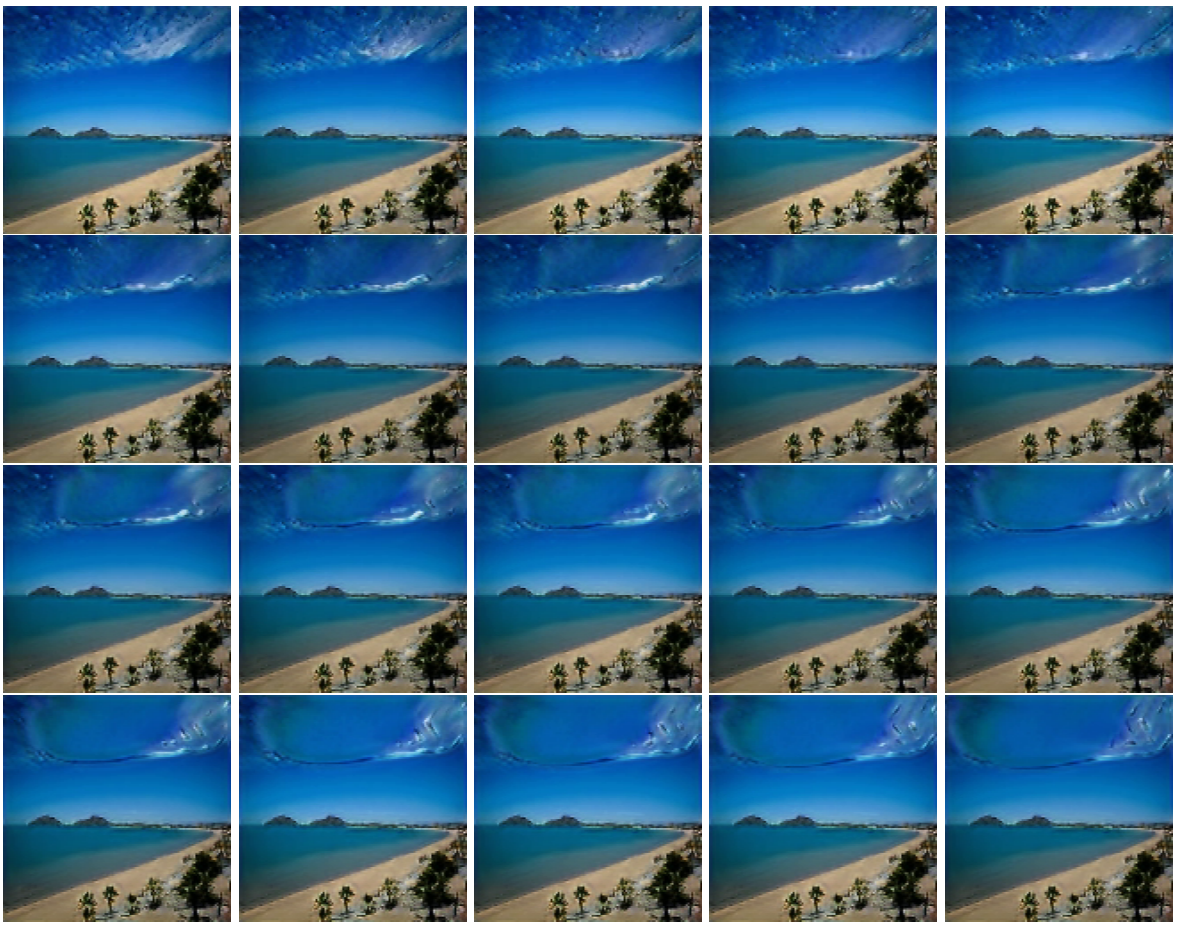


Figure 4.7: Failure mode where a small, blurry region is created. This region then extends rapidly across a large portion of the image.

5 Conclusion

A novel generator architecture for GAN-based video synthesis from a single input frame is introduced based on a combination of StyleGAN and DVD-GAN. The proposed architecture can be trained using commodity graphics cards and still achieves high quality results, albeit at slightly lower resolution. When down-scaling current models to train them on the same hardware, the proposed algorithm is able to generate significantly higher quality videos. This can be accredited to the fact, that most big models often heavily rely on Batch Normalization [43], which works significantly better when training with very large batch sizes and large compute resources. While video generation is still in its infancy the field is developing rapidly and the insights gained in this work can very well be useful for future style-based algorithms.

A new dataset is created for the task of "bringing landscape images to life". This dataset consists of several hundreds of videos of rivers and beaches. The proposed model performs reasonably well for this task. It is however out-shined by models, which were specifically created for video prediction on landscape images. These models usually allow for larger spatial resolutions by using a simplified and often even static motion model. This on the other hand, limits their ability to generate more complex motion. While other algorithms outperform this work in terms of resolution, this algorithms general motion model allows for more complex dynamic changes to be generated in a video. The landscape videos predicted by this and similar works could be used as background videos in games and movies in the future. A detailed comparison with other works for landscape video synthesis was not possible, as they are usually trained on proprietary datasets or their code was not made public at the time this work was created.

5.1 Future Work

For any type of future, practical application of this work the limitations of this work mentioned in section 4.5 would presumably have to be fixed. A first step to address them would likely be adding more compute resources and increasing model size. As such this section mainly considers how the proposed algorithm might be extended to explore similar video synthesis tasks.

Dataset: Due to the data collection pipeline breaking half way through this work, the gathered dataset was left wanting in terms of size and diversity. Only videos of rivers and some oceans were gathered. Future work could pick up here and gather a larger and, more importantly, denser dataset. This would allow for easier training and better generalization,

especially when considering the case of predicting videos from GauGAN images, as was proposed in this work.

Predicting the Future: The presented algorithm can be extended to allow for multiple input frames. Solving this problem is likely going to be easier, as the model can already infer motion form a given sequence instead of having to implicitly differentiate between static and dynamic areas and then generating motion based on the content in these areas. This also limits the modality of possible futures that can be predicted. The resulting model would be useful for a different set of tasks including autonomous driving and general search tasks based on a learned environment model. The predicted sequences in this setting would probably only have to be a few frames up to maybe one or two seconds long. Anything beyond that would probably result in too much uncertainty.

Unconditional Generation: An unconditional generator similar to DVD-GAN [7] could be developed by removing the encoder part of the architecture and drawing hidden states and the style-vector form a random distribution. Current research on video generation and prediction seems to agree, that unconditional video generation is an easier problem than video prediction [7]. Such models tend to generate interesting art and are already capable of generating short, video-realistic sequences.

List of Figures

List of Tables

Bibliography

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. *ImageNet Large Scale Visual Recognition Challenge*. 2015. arXiv: 1409.0575 [cs.CV].

[2] D. P. Kingma and M. Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[4] T. Karras, S. Laine, and T. Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: 1812.04948 [cs.NE].

[5] A. Brock, J. Donahue, and K. Simonyan. *Large Scale GAN Training for High Fidelity Natural Image Synthesis*. 2019. arXiv: 1809.11096 [cs.LG].

[6] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz. *MoCoGAN: Decomposing Motion and Content for Video Generation*. 2017. arXiv: 1707.04993 [cs.CV].

[7] A. Clark, J. Donahue, and K. Simonyan. *Adversarial Video Generation on Complex Datasets*. 2019. arXiv: 1907.06571 [cs.CV].

[8] P. Luc, A. Clark, S. Dieleman, D. de Las Casas, Y. Doron, A. Cassirer, and K. Simonyan. *Transformation-based Adversarial Video Prediction on Large-Scale Data*. 2020. arXiv: 2003.04035 [cs.CV].

[9] M. Saito, S. Saito, M. Koyama, and S. Kobayashi. "Train Sparsely, Generate Densely: Memory-Efficient Unsupervised Training of High-Resolution Temporal GAN". In: *International Journal of Computer Vision* 128.10-11 (May 2020), pp. 2586–2606. ISSN: 1573-1405. DOI: 10.1007/s11263-020-01333-y. URL: http://dx.doi.org/10.1007/s11263-020-01333-y.

[10] F. Ebert, C. Finn, A. X. Lee, and S. Levine. *Self-Supervised Visual Planning with Temporal Skip Connections*. 2017. arXiv: 1710.05268 [cs.RO].

[11] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, M. Suleyman, and A. Zisserman. *The Kinetics Human Action Video Dataset*. 2017. arXiv: 1705.06950 [cs.CV].

[12] Google. *Visdom*. Google. 2018. URL: https://cloud.google.com/tpu/.

[13] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu. *Semantic Image Synthesis with Spatially-Adaptive Normalization*. 2019. arXiv: 1903.07291 [cs.CV].

[14] E. Logacheva, R. Suvorov, O. Khomenko, A. Mashikhin, and V. Lempitsky. *DeepLandscape: Adversarial Modeling of Landscape Video*. 2020. arXiv: 2008.09655 [cs.CV].

[15] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. "Language Models are Unsupervised Multitask Learners". In: (2019).

[16] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG].

[17] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2018. arXiv: 1706.08500 [cs.LG].

[18] M. Arjovsky, S. Chintala, and L. Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[19] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. *Improved Training of Wasserstein GANs*. 2017. arXiv: 1704.00028 [cs.LG].

[20] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. *Spectral Normalization for Generative Adversarial Networks*. 2018. arXiv: 1802.05957 [cs.LG].

[21] M. Mirza and S. Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[22] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. arXiv: 1611.07004 [cs.CV].

[23] T. Karras, T. Aila, S. Laine, and J. Lehtinen. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2018. arXiv: 1710.10196 [cs.NE].

[24] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. *Analyzing and Improving the Image Quality of StyleGAN*. 2020. arXiv: 1912.04958 [cs.CV].

[25] K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[26] H. de Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A. Courville. *Modulating early visual processing by language*. 2017. arXiv: 1707.00683 [cs.CV].

[27] C. Vondrick, H. Pirsiavash, and A. Torralba. *Generating Videos with Scene Dynamics*. 2016. arXiv: 1609.02612 [cs.CV].

[28] M. Saito, E. Matsumoto, and S. Saito. *Temporal Generative Adversarial Nets with Singular Value Clipping*. 2017. arXiv: 1611.06624 [cs.LG].

[29] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo. *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting*. 2015. arXiv: 1506.04214 [cs.CV].

[30] M. Siam, S. Valipour, M. Jagersand, and N. Ray. *Convolutional Gated Recurrent Networks for Video Segmentation*. 2016. arXiv: 1611.05435 [cs.CV].

[31] X. Shi, Z. Gao, L. Lausen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo. *Deep Learning for Precipitation Nowcasting: A Benchmark and A New Model*. 2017. arXiv: 1706.03458 [cs.CV].

[32] Y. Endo, Y. Kanamori, and S. Kuriyama. *Animating Landscape: Self-Supervised Learning of Decoupled Motion and Appearance for Single-Image Video Synthesis*. 2019. arXiv: 1910.07192 [cs.GR].

[33] K. Soomro, A. R. Zamir, and M. Shah. *UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild*. 2012. arXiv: 1212.0402 [cs.CV].

[34] Youtube. *Youtube*. Youtube. 2020. URL: https://www.youtube.com.

[35] DeepMind. *DeepMind*. DeepMind. 2020. URL: https://deepmind.com/research/open-source/kinetics.

[36] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner. *FaceForensics++: Learning to Detect Manipulated Facial Images*. 2019. arXiv: 1901.08971 [cs.CV].

[37] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner. *FaceForensics: A Large-scale Video Dataset for Forgery Detection in Human Faces*. 2018. arXiv: 1803.09179 [cs.CV].

[38] W. Xiong, W. Luo, L. Ma, W. Liu, and J. Luo. *Learning to Generate Time-Lapse Videos Using Multi-Stage Dynamic Generative Adversarial Networks*. 2018. arXiv: 1709.07592 [cs.CV].

[39] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG].

[40] M. Arjovsky and L. Bottou. *Towards Principled Methods for Training Generative Adversarial Networks*. 2017. arXiv: 1701.04862 [stat.ML].

[41] A. Radford, L. Metz, and S. Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[42] B. Fuglede and F. Topsoe. "Jensen-Shannon Divergence and Hilbert Space Embedding". In: Jan. 2004, p. 31. ISBN: 0-7803-8280-3. DOI: 10.1109/ISIT.2004.1365067.

[43] S. Ioffe and C. Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[44] A. Krizhevsky, I. Sutskever, and G. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362. ISBN: 026268053X.

[46] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE].

[47] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[48] Y. Wang, W. Liao, and Y. Chang. "Gated Recurrent Unit Network-Based Short-Term Photovoltaic Forecasting". In: *Energies* 11 (Aug. 2018), p. 2163. DOI: 10.3390/en11082163.

[49] F. A. Reda, G. Liu, K. J. Shih, R. Kirby, J. Barker, D. Tarjan, A. Tao, and B. Catanzaro. *SDCNet: Video Prediction Using Spatially-Displaced Convolution*. 2018. arXiv: 1811.00684 [cs.CV].

[50] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. *Spatial Transformer Networks*. 2016. arXiv: 1506.02025 [cs.CV].

[51] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra. *Video (language) modeling: a baseline for generative models of natural videos*. 2016. arXiv: 1412.6604 [cs.LG].

[52] V. Patraucean, A. Handa, and R. Cipolla. *Spatio-temporal video autoencoder with differentiable memory*. 2016. arXiv: 1511.06309 [cs.LG].

[53] T. Xue, J. Wu, K. L. Bouman, and W. T. Freeman. *Visual Dynamics: Probabilistic Future Frame Synthesis via Cross Convolutional Networks*. 2016. arXiv: 1607.02586 [cs.CV].

[54] C. Finn, I. Goodfellow, and S. Levine. *Unsupervised Learning for Physical Interaction through Video Prediction*. 2016. arXiv: 1605.07157 [cs.LG].

[55] C. Vondrick and A. Torralba. "Generating the Future with Adversarial Transformers". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2992–3000. DOI: 10.1109/CVPR.2017.319.

[56] L. Zhang. *Video Synthesis from the StyleGAN Latent Space*. 2020. (Visited on 10/20/2020).

[57] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. *Snapshot Ensembles: Train 1, get M for free*. 2017. arXiv: 1704.00109 [cs.LG].

[58] Y. D. API. *Youtube Data API*. Youtube. 2020. URL: https://developers.google.com/youtube/v3.

[59] facebookresearch. *Visdom*. Facebook. 2020. URL: https://github.com/facebookresearch/visdom.

[60] M. S. M. Sajjadi, O. Bachem, M. Lucic, O. Bousquet, and S. Gelly. *Assessing Generative Models via Precision and Recall*. 2018. arXiv: 1806.00035 [stat.ML].

[61] S. Barratt and R. Sharma. *A Note on the Inception Score*. 2018. arXiv: 1801.01973 [stat.ML].

[62] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[63] M. Seitzer. *pytorch-fid: FID Score for PyTorch*. https://github.com/mseitzer/pytorch-fid. Version 0.1.1. 2020.

[64] T. Unterthiner, S. van Steenkiste, K. Kurach, R. Marinier, M. Michalski, and S. Gelly. *Towards Accurate Generative Models of Video: A New Metric & Challenges*. 2019. arXiv: 1812.01717 [cs.CV].

[65] J. Carreira and A. Zisserman. *Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset*. 2018. arXiv: 1705.07750 [cs.CV].