Stefan Ainetter, BSc

Evaluation of Spatiotemporal GANs

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisors

Pinz, Axel, Ao.Univ.-Prof. Dipl.-Ing. Dr.techn.
Feichtenhofer, Christoph, Dipl.-Ing. Dr.techn. BSc

Institute for Electrical Measurement and Measurement Signal Processing

Graz, October 2018

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date	Signature

Abstract

Generating high-resolution, photo-realistic images has been a long-standing goal in machine learning and is a popular field of research nowadays. Using Generative Adversarial Networks (GANs) for video generation is less explored, and therefore needs further attention.
We present a spatiotemporal GAN for action recognition which makes it possible to generate videos for 101 action classes, at a spatial resolution of $227x227px$. With this method, we are able to outperform other state-of-the-art spatiotemporal GANs in terms of spatial resolution and image quality. We provide a quantitative evaluation of our results, which underlines the improvement compared to other methods.
Furthermore, using a different approach, we tackle the problem of generating videos at increased spatial resolution. We use a Progressively growing GAN, an approach originally developed for image generation, to generate videos for action recognition at a spatial resolution of $512x512px$.
Our approach is able to generate videos with high image quality, which means that these videos could be used as augmented data in the domain of action recognition, to boost the performance of state-of-the-art action recognition classifiers in the future.

Kurzfassung

Die Erzeugung hochauflösender, fotorealistischer Bilder ist seit langem ein Ziel des maschinellen Lernens und ein beliebtes Forschungsgebiet. Der Bereich der Videoerzeugung ist weniger erforscht und bedarf deshalb zusätzlicher Aufmerksamkeit.
Wir präsentieren ein raum-zeitliches "Generative Adversarial Network" (GAN) im Bereich der Bewegungsmustererkennung, welches es ermöglicht, Videos für 101 Bewegnungsarten mit einer räumlichen Auflösung von $227x227px$ zu erzeugen. Mit dieser Methode sind wir in der Lage, andere moderne raum-zeitliche GANs bezüglich räumlicher Auflösung und Bildqualität zu übertreffen. Wir bieten eine quantitative Evaluierung unserer Ergebnisse, welche die Verbesserung im Vergleich zu anderen Methoden unterstreicht.
Darüber hinaus stellen wir eine Methode vor, die das Problem der Videoerzeugung mit erhöhter räumlicher Auflösung behebt. Wir verwenden ein "Progressively growing GAN", ein ursprünglich für die Bilderzeugung entwickeltes neuronales Netzwerk, um Videos mit einer räumlichen Auflösung von $512x512px$ zu generieren.
Unsere Methode ist in der Lage, Videos mit hoher Bildqualität zu erzeugen, welche als zusätzliche Trainingsdaten verwendet werden können, um Methoden der Bewegungsmustererkennung in Zukunft zu verbessern.

Contents

1. Introduction

Since their first introduction, Generative Adversarial Networks (GANs) [17] have been a popular field of research due to their ability to learn perceptual representations of images in an unsupervised manner. Especially in computer vision, this method can be beneficial for a variety of tasks like clustering, classification, and sample generation. Recently, using GANs for image generation made a big step forward, resulting in methods which are able to synthesize images up to a spatial resolution of 2 megapixel [6, 58]. However, using GANs in the field of video generation has not been tackled with the same intensity, and therefore needs further attention.

This work addresses the generation of videos in the domain of action recognition. Training a model to generate realistic-looking videos for action recognition could provide further understanding of how classifiers in this domain work, and therefore could boost the results of current state-of-the-art action recognition classifiers. For this purpose, we provide a spatiotemporal GAN, which is able to generate videos in the domain of action recognition at a spatial resolution up to $227x227px$. Although state-of-the-art GANs are able to generate images at spatial resolution $1024x1024px$, for video generation in the domain of action recognition, our output resolution is state-of-the-art. We quantitatively evaluate our results using the Inception score [45]. We compare our model to state-of-the-art approaches for action video generation, and are able to outperform them in terms of Inception score and spatial resolution. Furthermore, we demonstrate that it is possible to fool the Inception score metric by generating samples that are optimized to achieve a high score, as claimed by [45]. We show how to use the Inception score to obtain valid results, by strictly separating the video generation process and the evaluation.
Additionally, we present a method to further increase the spatial resolution of generated videos. Using the framework provided by [26], we are able to

generate action videos at a resolution of $512x512px$, which is the highest spatial resolution for action videos at the time of writing, to the best of our knowledge. Figure 1.1 provides an example of a video in the domain of action recognition, which is generated with our approach.

Problem Statement

To summarize, this thesis addresses the following issues:

- Implementing a spatiotemporal GAN for action recognition, which aims to improve the quality of the generated videos, as well as the spatial resolution, according to state-of-the-art approaches.
- Using quantitative evaluation metrics for generative models, to gain further insights about our approach and be able to compare it with other state-of-the-art methods.
- Generating high resolution videos in the domain of action recognition. To the best of our knowledge, a spatial resolution of $227x227px$ is state-of-the-art, which we want to increase.

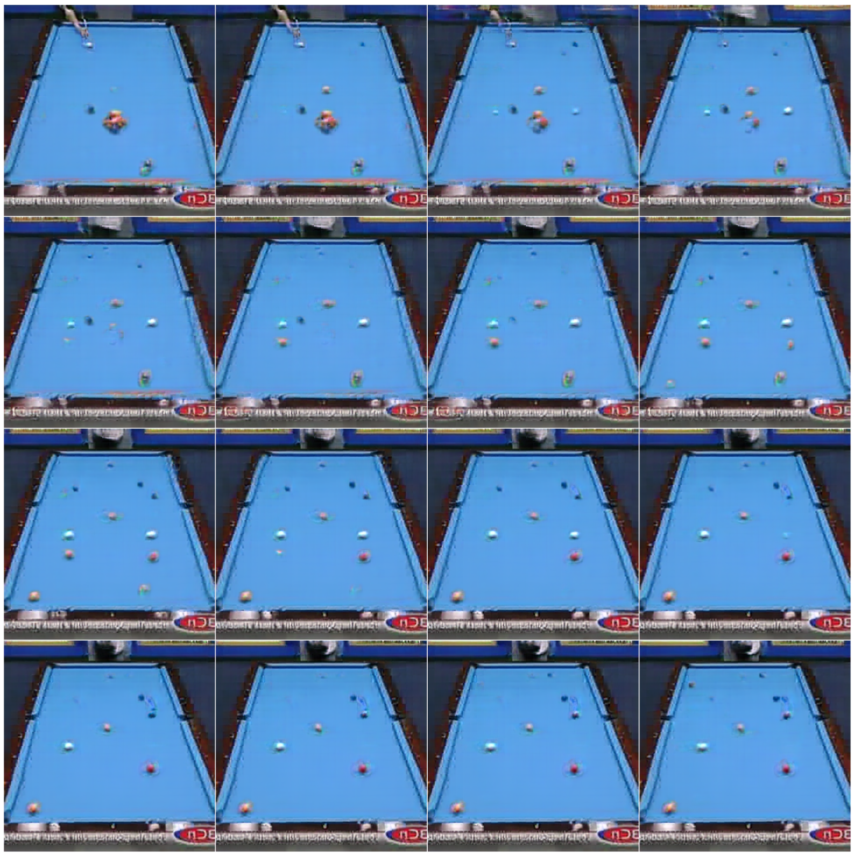


Figure 1.1.: An example of our generated action videos with a spatial resolution of $512x512px$. The sequence shows 16 consecutive frames of playing billiard, all frames with a spatial resolution of $512x512px$.

2. Related Work

In this chapter, we briefly discuss the usage of GANs in computer vision and provide an overview about related work on GANs in image and video domain. We also give a short overview about popular datasets for action recognition and discuss methods to quantitatively evaluate generative models, including the most popular GAN evaluation metrics. Finally, we discuss state-of-the-art approaches for high resolution image/video generation.

Generative models for image generation. The idea of GANs was first proposed in [17], outperforming other generative models like autoencoders [29, 56] and Boltzmann machines [35, 44] according to image quality. Since then, GANs have been a popular field of research, and are used in different domains like image in-painting [22], image-to-image translation [36, 59] and super resolution [34]. GAN training using the loss function proposed in [17] is considered as unstable. Therefore, Dosovitskiy and Brox [9] combined the GAN loss with additional losses in image and feature space, to make the training more stable. Their loss function proved to be stable during training and generated realistic-looking images, although their model is more like an auto-encoder combined with the GAN training, than a GAN itself. Several attempts [38, 14, 39] used this loss function in combination with activation maximization [65, 11, 66, 46] to generate class specific images.
Another popular approach is the Wasserstein-GAN [3], which uses the Earth-Mover or Wasserstein-1 distance as basis for their loss function. Several GANs [41, 18, 45, 5, 2] adapted this loss function and were able to improve quality, stability and variation of the generated images.

Video Generation using GANs. Video recognition and classification received a lot of attention in recent past. Therefore, generating videos via generative models is also drawing much attention. TGAN [43] generates videos for action recognition using two generators, where one generator learns to generate images, and the other one models temporal coherence. It produces frames with the spatial resolution of $64x64px$. Vondrick et al. [57] proposed a model to generate videos with a spatiotemporal convolutional architecture that separates foreground and background. Spampinato et al. [50] use a similar approach for video generation, which also untangles the foreground from the background. Fuchs [14] proposed a model to generate videos of arbitrary length for autonomous driving.

Action recognition datasets. Using a popular dataset enables opportunities to compare our performance to other approaches. Therefore, one of the main points of this work is to evaluate spatiotemporal GANs in the context of action recognition. The UCF-101 dataset [49] is a widely used dataset for action recognition, and using it enables us to compare the performance of our approach to state-of-the-art work. It contains video data of 101 different sports activities and each video comes at a resolution of $320x240px$, and runs at 25 frames per second.
There are several datasets for action recognition available:

- UCF-101: 13,320 clips from 101 actions
- Kinetics Human Action Video Dataset [27]: at least 400 video clips for each of the 400 human action classes
- Sports-1M [25]: 1.1M videos from 487 sports classes
- Youtube-8M [1]: more than 8M videos from 4800 classes

Evaluation methods for generative models. Directly comparing the images of two different generative models, in terms of image quality, is hard to achieve. Salimans et al. [45] proposed an evaluation metric called the Inception score. To achieve a high Inception score, images should contain meaningful objects and the GAN should generate varied images. This metric is widely used to evaluate the quality of generated images [50, 43, 18]. Another way of GAN evaluation is to measure the variance of the generated data. This can be achieved using structural similarity [62], and was used as evaluation metric in [40, 26].
Another popular method is using the Fréchet Inception distance [19], which is based on the Fréchet distance [10].

GANs for high resolution output. There are several papers [48, 34] which use GANs in the context of super-resolution to create high resolution output from low resolution input. Denton et al. [7] proposed a framework that generates images in a coarse-to-fine fashion, combining a conditional GAN model with a Laplacian pyramid representation. Karras et al. [26] were able to generate images at a resolution of $1024x1024px$ using GANs and a coarse-to-fine structure. For this approach, both the generator and discriminator grow progressively during training. GANs are also often used for synthesizing videos. Koltun and Chen [6] and Wang et al. [58] are able to synthesize videos with a spatial resolution of $2048x1024px$, also using progressively growing network architectures. However, they use semantic layouts instead of noise as input, which simplifies the process of generating samples.

3. Preliminaries: Machine Learning

This chapter covers the most important preliminaries of machine learning as well as specific concepts that are important for this work. The focus is on deep learning, which is a sub category of machine learning. A more detailed description about deep learning is provided by [16].

3.1. Deep Learning

Deep learning is a specific research area in the field of Artificial Intelligence (AI) using a hierarchy of concepts, with each concept defined through the relation to simpler concepts. Therefore, this hierarchy enables computers to learn complicated concepts from simpler ones. Nowadays more computational power is available to build deep models, and GPU usage enables us to highly parallelize operations, so that deep learning can operate in complex real-world environments. Here, we focus on deep neural networks in the context of computer vision. These deep neural networks learn the representation of image and video data using a mathematical function mapping of raw data to some output values.

3.2. Convolutional Neural Networks

A key component of computer vision with deep neural networks is the usage of a Convolutional Neural Network (CNN) [16, 33, 32]. CNNs are specially used to process input data with a grid like topology. This kind of network is often used for time-series data or image data. One of the main concepts of CNNs is to use convolution instead of matrix multiplication in some layers. In general, the convolutional operation is written as

$$s(t) = (x * w)(t), \tag{3.1}$$

where x refers to the input sequence and w is the kernel function depending on time t. The new function $s(t)$ is often called feature map. The convolution operator $*$ can also be replaced by using an integral

$$s(t) = \int x(\tau)w(t-\tau)\,\mathrm{d}\tau, \tag{3.2}$$

where τ describes the age of sequential data.

In computer vision, the input is usually a two-dimensional array of input data, and the kernel is a two-dimensional array of parameters to be learned. In this case, convolution S can be written as

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n), \tag{3.3}$$

where I is the input image, K is the kernel, and m, n, j, i are spatial indices. In practice, most machine learning libraries use cross-correlation, which is the same as convolution but without flipping the kernel K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n). \tag{3.4}$$

The main difference between convolution and correlation is that the associative law only holds for convolution. However, we are not interested in combining several kernels, but rather want to learn the kernel values during training. Therefore, using correlation is perfectly fine in the context of CNNs. Further information about convolution and correlation is provided by [23].

The motivation behind using CNNs is that they support three important concepts that help to improve machine learning systems:

Sparse weights mean that not every input unit interacts with each output unit. This can be achieved by making the kernel w smaller than the input, which improves both the amount of memory needed to store parameters and also computing the output requires fewer operations.

Parameter sharing: By shifting the kernel along the whole input, each member of the kernel is used at every position. This means that only one set of parameters has to be learned rather than specific weights for each input. Parameter sharing therefore reduces the memory requirements.
Equivariant representations: Sharing the weights across one layer has the additional benefit of equivariance in translation. In images, convolution creates a 2D feature map as output. When the objects in the image move, this translation is also observable in the feature map.

3.2.1. Structure of CNNs

CNNs typically contain several **convolutional layers** which perform convolutions on the input to produce linear activations. The **detection layer** uses non-linear activation functions for extracting features. Rectified Linear Units (ReLUs) are often used as non-linear activation function in this layer. The ReLU function is described as follows:

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0. \end{cases} \tag{3.5}$$

Two major benefits of ReLUs are sparsity and a reduced likelihood of vanishing gradient. Sparsity of parameters can speed up training. Vanishing gradients often occur in deep networks, where many back-propagation steps are performed for training. This means that after some steps the gradient vanishes which is a major problem for learning. However, because the output of the ReLU has no upper boundary this problem occurs less often.
The **pooling layer** helps to reduce the complexity by reducing the number of outputs of the network. A popular pooling function is **max pooling**, where the maximum of multiple output values is taken to represent this area. Other pooling functions use a weighted average or L^2 Norm to calculate the output.

More detailed information about CNNs can be found in [16, 33, 32].

3.2.2. Upconvolution

Upconvolution or deconvolution [68] was initially used for visualizing layers of convolutional neural networks. It describes the process of transposing the convolution. This leads to a higher resolution in output space after each layer. Figure 3.1 shows a simple example for upconvolution. Each input value is elementwise multiplied by a filter, and overlapping output values are summed up. Furthermore, upconvolution uses the convolutional layer properties (padding, stride) in the opposite way, which means that padding is removed from the output rather than added to the input, and stride results in upsampling rather than downsampling. For generative models, upconvolutional layers are used to transform a random code vector to an image. These transposed convolutions help to expand the resolution of data after each layer, until one arrives at a suitable resolution for image data. Further information about upconvolution is provided by [67, 68].

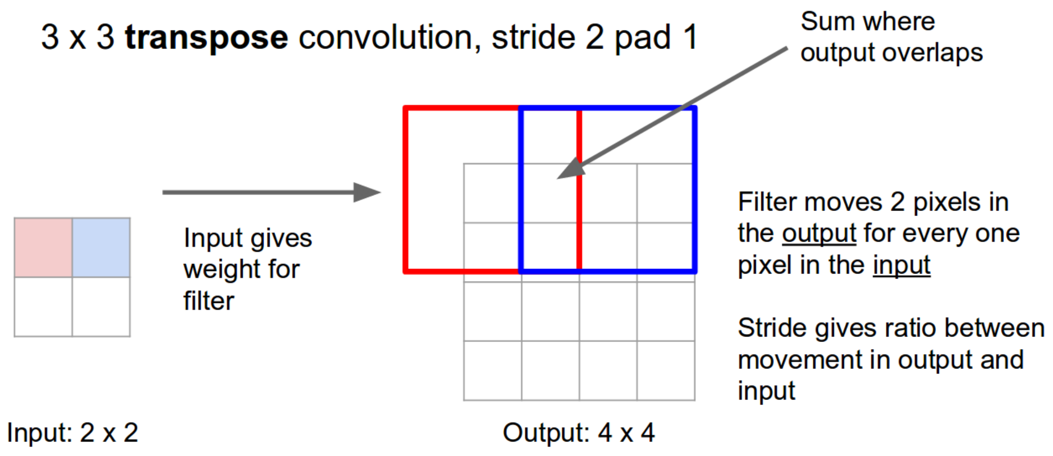

3 x 3 **transpose** convolution, stride 2 pad 1
Sum where output overlaps
Input gives weight for filter
Filter moves 2 pixels in the output for every one pixel in the input
Stride gives ratio between movement in output and input
Input: 2 x 2
Output: 4 x 4

Figure 3.1.: Example for upconvolution. Upconvolution is performed on a $2x2$ input, using a $3x3$ kernel with $stride = 2$ and $padding = 1$. One can see that the overlapping output values are summed up. The result is a $4x4$ output matrix. (Figure from [13])

3.3. Learning Algorithms, Back-Propagation, Optimizer

Deep neural networks learn to approximate specific functions during training. The aim of training is to minimize the loss function J so that the predictions are as close as possible to the ground truth values. In standard back-propagation [42], the parameters $\boldsymbol{\theta}$ of a neural network are updated in the negative direction of the gradient of the loss function J to minimize the error. The update function for a specific network parameter θ_{ij} looks as follows:

$$\begin{aligned} \theta_{ij}^{new} &= \theta_{ij}^{old} + \Delta\theta_{ij}, \\ \Delta\theta_{ij} &= -\eta \frac{\partial J}{\partial \theta_{ij}}. \end{aligned} \tag{3.6}$$

The parameter η describes the learning rate and θ_{ij} could either be a weight or a bias of a neuron. The term back-propagation refers to the concept of information flowing backward through the network to compute the gradient with respect to the network parameters. This gradient is afterwards used by a specific optimization algorithm for parameter update. Implementing an optimization algorithm in deep neural networks is computationally expensive, as the number of propagation steps increases with the number of network layers. Therefore, we decided to use already implemented optimization methods to calculate the gradient and minimize the loss function.

Adam Optimizer. The Adam Optimizer [28] uses adaptive moments to overcome local minima in the error function. Furthermore, gradient clipping is used to avoid instabilities and large jumps in the parameter space. Initialization of the moments m and v, as well as the time-step t, looks as follows:

$$\begin{aligned} m_0 &\leftarrow 0, \\ v_0 &\leftarrow 0, \\ t &\leftarrow 0. \end{aligned} \tag{3.7}$$

The update rule for the network parameters θ, with the gradient g and the adaptive learning rate lr_t at time-step t, is described in [28] as:

$$\begin{aligned} t &\leftarrow t+1, \\ lr_t &\leftarrow \alpha \frac{\sqrt{1-\beta_2^t}}{(1-\beta_1^t)}, \\ m_t &\leftarrow \beta_1 m_{t-1} + (1-\beta_1) g, \\ v_t &\leftarrow \beta_2 v_{t-1} + (1-\beta_2) g^2, \\ \theta_t &\leftarrow \theta_{t-1} - lr_t \frac{m_t}{(\sqrt{v_t}+\epsilon)}. \end{aligned} \tag{3.8}$$

The default values are as follows: the constant learning rate $\alpha = 0.001$, the exponential decay for the first momentum $\beta_1 = 0.9$ and the exponential decay for the second momentum $\beta_2 = 0.999$. The constant parameter $\epsilon = 10^{-8}$ is used for numerical stability. More details about the implementation are available online[1].

3.4. Generative Models

The general idea of generative models is to take a collection of training data and form a representation of the probability distribution. This can be achieved by analyzing a density estimation of the input data and then fitting a specific known probability density function (e.g. Gaussian distribution). Another way is to show training data to a model and train it to generate samples with similar image statistics, rather than learning a specific distribution function. This section contains an overview about the most important concepts of generative models. The focus is on Generative Adversarial Networks (GANs), because they are used for the experiments in this thesis. More detailed information about generative models is provided by [15, 16, 17, 45].

[1]https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer, last visited: Aug. 2018

3.4.1. Maximum Likelihood Estimation

The basic idea of maximum likelihood is to define a model that provides an estimate of a probability distribution, parameterized by $\mathbf{\Theta}$. The likelihood can be described as

$$\prod_{i=1}^{m} \log p_{model}(\mathbf{x}^{(\mathbf{i})}; \mathbf{\Theta}), \tag{3.9}$$

which refers to the probability that the model assigns to the training data containing m training examples $\mathbf{x}^{(i)}$. In logarithmic space, the maximum likelihood estimator $\mathbf{\Theta_{ML}}$ looks as follows

$$\mathbf{\Theta_{ML}} = \arg\max_{\Theta} \sum_{i=1}^{m} \log p_{model}(\mathbf{x}; \mathbf{\Theta}), \tag{3.10}$$

where $p_{model}(\mathbf{x}; \mathbf{\Theta})$ is the probability distribution that maps any $\mathbf{x}$ to a real number estimating the true probability distribution of the data.

Not every generative model uses the concept of maximum likelihood, however many of them, like GANs can be made to do so.

3.4.2. Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a specific type of generative models which only focus on the ability to generate samples, and do not learn an explicit density $p_{model}(\mathbf{x}; \mathbf{\Theta})$. GANs are using an implicit approach that samples directly from the distribution represented by the model. Typically, both generator and discriminator are represented by deep neural networks. This subsection briefly explains the basic GAN structure and describes the training of GANs. Furthermore, we describe popular evaluation techniques that are used nowadays to measure the performance of generative models.

Network Structure

The basic idea of GANs is to set up a game between two networks. One of them is the generator which creates samples that are intended to come from

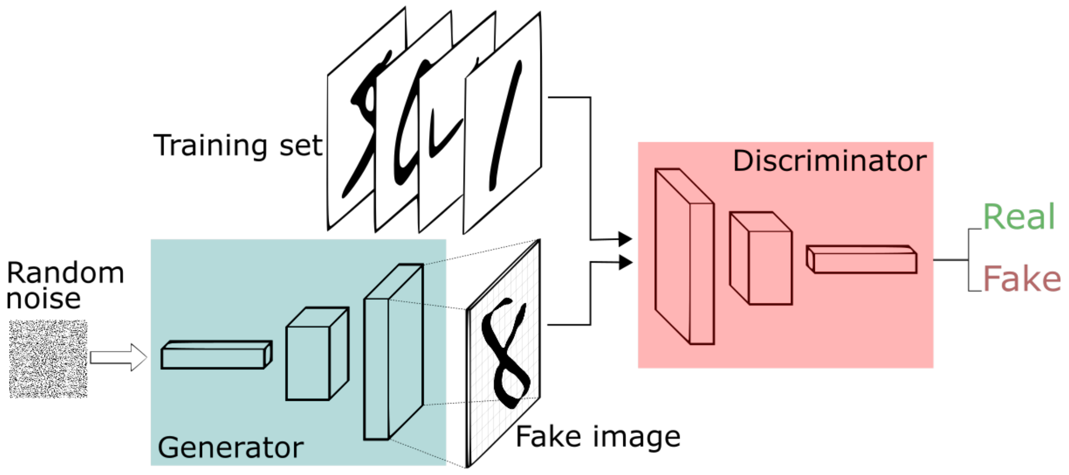


Figure 3.2.: Basic GAN structure shown in the context of learning to generate handwritten digits. The generator receives random noise as input, and tries to generate realistic-looking samples. During training, the discriminator receives fake images from the generator and real images, and classifies them as real or fake. The generator uses an upconvolutional structure to transform random code vectors to images. The discriminator uses convolutional layers and fully-connected layers to classify the images as real or fake. (Figure from [47])

the same distribution as the training data. The other network is called the discriminator. The discriminator is trained to classifying data as real or fake. To achieve this goal, the discriminator uses traditional supervised learning techniques, classifying inputs into real or fake. The generator is trained to fool the discriminator. This means that the generator tries to produce fake samples, which are indistinguishable from real ones. To succeed in this game, the generator must learn to create samples that are drawn from a distribution which is as similar as possible to the distribution of the training data. Figure 3.2 shows an example of the basic GAN structure.
The discriminator uses convolutional layers for feature extraction. These features serve as input for fully-connected layers, which learn to classify the input. Fully-connected layers are defined as specific layers in neural networks, where each input neuron is connected with each output neuron. The last layer of a discriminator is typically a softmax layer, which uses the softmax function to represent the probability distribution over the possible classes (real or fake) (see [16] for more details).

GAN Training

During GAN training, the generator and the discriminator play a so called minimax game. The objective is to balance both cost functions, which can be written as

$$J^{(G)} = -J^{(D)}, \tag{3.11}$$

where $J^{(G)}$ and $J^{(D)}$ are the cost functions of the generator and discriminator respectively. Although there are many different variations of the cost functions, the basic discriminator cost function, as used in [17], is the standard cross-entropy error written as

$$J^{(D)}(\boldsymbol{\Theta}^{(D)}, \boldsymbol{\Theta}^{(G)}) = -\frac{1}{2}\,\mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] - \frac{1}{2}\,\mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]. \tag{3.12}$$

In general, $\mathbb{E}$ defines the expectation of a function with respect to a specific probability distribution. $D(\mathbf{x})$ defines the discriminator function that gets an input vector $\mathbf{x}$ and is represented with parameters $\boldsymbol{\Theta}^{(D)}$. The generator G implicitly defines the probability distribution p_{model} and is able to generate samples from a random variable $\mathbf{z}$. The parameters $\boldsymbol{\Theta}^{(G)}$ are used to define G.
The generator loss function is defined as

$$J^{(G)} = -\frac{1}{2}\,\mathbb{E}_{\mathbf{z}}[\log(D(G(\mathbf{z})))]. \tag{3.13}$$

The objective of the generator is to maximize the log-probability of the discriminator being mistaken.

To conclude, the goal of training GANs is to find a unique solution, where G recovers the distribution of the training data, and D is equal to $\frac{1}{2}$, which means that the discriminator is not able to differentiate between real and fake data.

Advantages of GANs

Sample quality. GANs are considered to be the type of generative models that produce the best samples. However, it is very hard to quantify this, as there is no way to directly measure the quality.

Generator training not directly dependent on real data. During training, $J^{(G)}$ does not directly receive any real data. This means that all information about the training data comes only through the back-propagated gradient from the discriminator. Therefore, GANs are more resistant to overfitting, because the generator has no opportunity to directly copy training examples [63, 17].

No difficulties when sampling data. After training, sample generation with GANs only requires one forward pass through the generator. Other generative models, for example Boltzmann machines, need an unknown number of iterations via a Markov chain to sample data.

Disadvantages of GANs

Non-convergence. GANs require finding the equilibrium to a target, depending on the generator and the discriminator. Therefore, it can be very hard to balance the training. Even if one network successfully minimizes its own cost function in one update, it might increase the cost for the other one. Therefore, it is very important to find the right balance between G and D to guarantee training progress and to avoid mode collapse.

Mode collapse. Mode collapse is a problem that occurs when the generator learns to map several different input vectors to the same output point. It is common that trained GANs are not capturing all dataset statistics, which means that partial mode collapse occurs. Partial mode collapse refers to scenarios in which the generator produces multiple images that contain the same color or texture themes, or multiple images containing different views of the same object. Full mode collapse results in a generator producing samples from only one mode, and is therefore a clear failure case.

Results are hard to evaluate. Another disadvantage of GANs is that quantitative evaluation of samples is hard, because no explicit distribution function is learned. This means that only the generated output can be evaluated. Further difficulties regarding GAN evaluation are described by [54].

3.4.3. Evaluation Techniques for Generative Models

Evaluation of generative models is non trivial, because one cannot directly compare the quality of two generated samples in terms of photorealism. In this subsection, we look at state-of-the-art metrics that are used to evaluate the performance of generative models in terms of image quality and sample variance. Further information about evaluation metrics for generative models is provided by [45, 63, 4, 54].

Inception Score

The Inception Score (IS) was proposed in [45] to directly assess the quality of generated images. Before the IS was proposed, the quality of generated images was mainly assessed by human annotators. The task of annotators is to distinguish between real and fake data. However, these evaluation results vary depending on the data domain, as well as the annotators themselves.

The IS aims to achieve the following two objectives:

1. The images should contain meaningful data, which means that classification of the generated images should be possible. In other words, the conditional label distribution $p(y|\mathbf{x})$ of an image belonging to a specific class should have low entropy.
2. The generative model should be able to generate images with high diversity. Formally, this means that the marginal $\int p(y|\mathbf{x} = G(z))dz)$ should have a high entropy. In the best case, the generator learned a uniform distribution of all classes, which means that no mode dropping or mode collapse [17] appears.

By combining these objectives, the Inception score is defined as

$$\ln(IS) = \mathbb{E}_{\mathbf{x}\sim p_{model}}[D_{KL}(p(y|\mathbf{x})||p(y))], \tag{3.14}$$

where $D_{KL}(p(y|\mathbf{x})||p(y))$ describes the Kullback-Leibler divergence [31] of the two probability distributions $p(y|\mathbf{x})$ and $p(y)$. $\mathbb{E}_{\mathbf{x}\sim p_{model}}$ is the expectation of $D_{KL}(p(y|\mathbf{x})||p(y))$ with respect to $p_{model}(\mathbf{x})$. Simplifications of Equation 3.14 are provided in Appendix A.1.

Salimans et al. [45] were the first who proposed this evaluation metric, using the Inception model [52] to classify the generated samples.

Structural Similarity

Structural Similarity (SSIM) [62, 61] is a measure for image quality which aims to measure perceived image quality. This method uses the assumption that structural information in images is highly important for humans to perceive vision-based information.
SSIM uses a weighted product of comparison measures for luminance l, contrast c and structure s which are defined as

$$l(\mathbf{x}, \mathbf{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, \tag{3.15}$$

$$c(\mathbf{x}, \mathbf{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \tag{3.16}$$

$$s(\mathbf{x}, \mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}. \tag{3.17}$$

For two images $\mathbf{x}$ and $\mathbf{y}$ the parameters μ_x and σ_x^2 define the mean and variance of $\mathbf{x}$, and $\mathbf{y}$ respectively. Furthermore, σ_{xy} defines the covariance between $\mathbf{x}$ and $\mathbf{y}$. The parameters C_1, C_2 and C_3 are small constants, helping to avoid numerical instability.

SSIM is then defined as

$$SSIM(\mathbf{x}, \mathbf{y}) = [l(\mathbf{x}, \mathbf{y})]^{\alpha}[c(\mathbf{x}, \mathbf{y})]^{\beta}[s(\mathbf{x}, \mathbf{y})]^{\gamma}. \tag{3.18}$$

The parameters α, β and γ are used as weighting to define the importance of the different components.
Multi-Scale Structural Similarity (MS-SSIM) uses the same calculations to measure the similarity of images, but uses different scales of the images $\mathbf{x}$ and $\mathbf{y}$ for calculation. MS-SSIM can be applied on full spatial image resolution or on patches of images. SSIM and MS-SSIM are used to evaluate GAN performances [40, 34], showing that this metric can be used to measure the variance of generated samples, and therefore gives information about possible GAN faults like mode collapse.

4. Generative Adversarial Networks for Video Generation

This chapter describes our approach to generate images and videos in the domain of action recognition using a GAN architecture. First, we describe the network architecture of our GAN, which is based on [38]. We explain our composed loss function, which is originally provided by [9] for images, and extended to work in the spatiotemporal domain by [14]. We explain how we transferred the network parameters of the spatial GAN to our spatiotemporal GAN. Then, we focus on the video generation process itself, and explain how we were able to generate videos for action recognition using pre-trained action recognition classifier networks. In the end, we evaluate the generated videos both quantitatively and qualitatively, and compare the results to other state-of-the-art approaches.

4.1. Spatial Network Architecture

The spatial network structure used in this chapter is provided by [9] and is used for image generation. This generative model consists of a generator and a discriminator, with an additional encoder network used for feature extraction. During training, the extracted features serve as input code vector $\mathbf{z}$ for the generator and the discriminator. All code vectors $\mathbf{z}$ belong to a low-dimensional input space, which we denote as latent space.

Generator. The generator network consists of fully-connected layers, followed by convolutional and upconvolutional layers to map a code vector $\mathbf{z}$ to an image. Figure 4.1 shows the network architecture. Upconvolutional layers make it possible to up-sample the respective layer input so that the

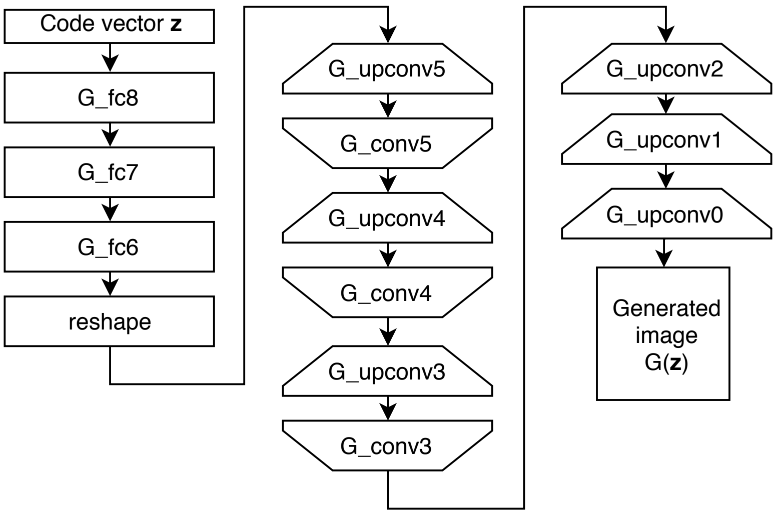


Figure 4.1.: Network architecture of generator network. The generator consists of three fully-connected layers, followed by several convolutional and upconvolutional layers. After G_fc6, the output vector is reshaped to match the input dimensionality of the upconvolutional layer G_upconv5. The network performs a mapping from latent space to image space.

code vector **z** with dimension $1x4096$ maps to an image of $256x256px$. RELUs are used as nonlinear activation functions in all layers.
Discriminator. The architecture of the discriminator network consists of five convolutional layers, followed by a pooling layer to extract features of the input images. These features are concatenated with a code vector **z**, which is extracted from real images by the encoder network. Dropout [51] is performed on the resulting feature vector before it is further processed by two fully-connected layers and then classified as real or fake image. Figure 4.2 shows the network architecture of the discriminator.
Encoder network. During training, the encoder network is used as auxiliary network to extract features from real and generated images. The network architecture is AlexNet [30] pre-trained on ImageNet. The network parameters are frozen during GAN training. Figure 4.3 shows the network architecture of the encoder.

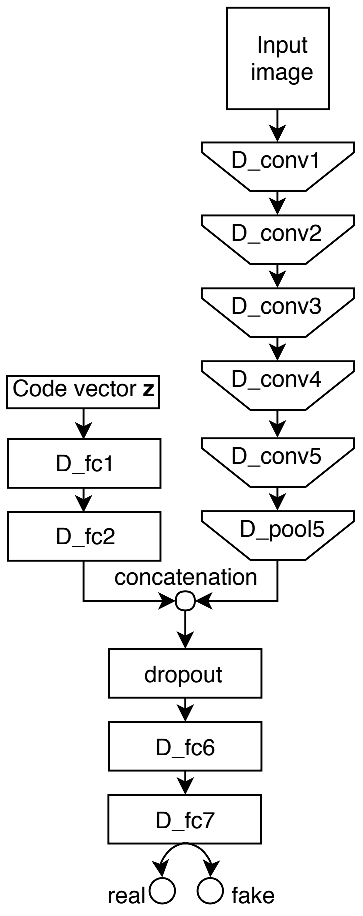


Figure 4.2.: Network architecture of the discriminator. The layers D_conv1 to D_pool5 perform feature extraction from input images. These features are then concatenated with a code vector **z**, which is extracted from real images using the encoder network, and further processed by the fully-connected layers D_fc1 and D_fc2. Afterwards, a dropout layer and two fully-connected layers are used to classify the image as real or fake.

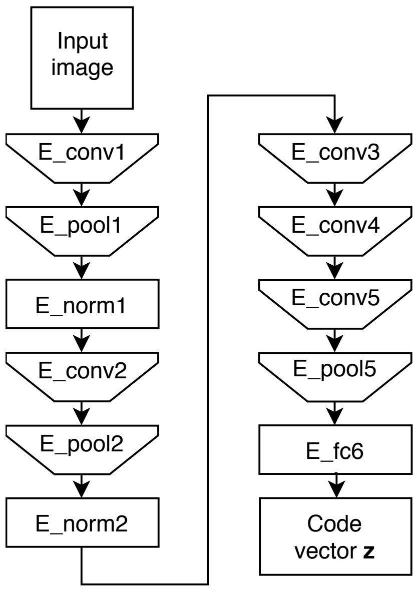


Figure 4.3.: Network architecture of the encoder, which is AlexNet truncated at the fully-connected layer E_fc6. The first and second convolutional layers are followed by pooling and normalization layers. Afterwards, three additional convolutional layers E_conv3 to E_conv5 and one pooling layer E_pool5 are used to perform feature extraction. The last layer E_fc6 is fully-connected, and provides the extracted code vector **z**.

4.2. Spatiotemporal Network Structure and Parameter Transfer

We extended the network structures described in Section 4.1 to generate videos instead of images. The main idea is to convert the filters from spatial to spatiotemporal kernels using the approach proposed in [12]. Then, the parameters of the pre-trained networks provided by [9] are transferred to the spatiotemporal domain. The new spatiotemporal weights $\mathbf{w}_{3D}$ of all convolutional and upconvolutional layers were initialized layerwise, by following these steps:

1. The temporal weights for each layer are defined as $\mathbf{w}_{2D}$ with the dimensions $[WxHxC]$, where W and H define the spatial filter size, and C defines the number of channels. In the first step, the dimensions of the spatiotemporal weights $\mathbf{w}_{3D}$ were defined as $[WxHxT'xC]$, where T' describes the temporal filter depth.
2. All spatial weights $\mathbf{w}_{2D}$ were then copied to each temporal position t of the weights $\hat{\mathbf{w}}_{3D}$. This step can be written as

$$\hat{\mathbf{w}}_{3D}(t) = \mathbf{w}_{2D}, \forall t \in [1, T']. \tag{4.1}$$

3. The last step is to divide the spatiotemporal weights $\hat{\mathbf{w}}_{3D}$ by the temporal filter depth T':

$$\mathbf{w}_{3D} = \frac{\hat{\mathbf{w}}_{3D}}{T'}. \tag{4.2}$$

   This step is used to average the weights across time, and therefore serves as temporal smoothing.

We decided to use a temporal filter depth of $T' = 3$ for all our layers. All biases, and weights from the fully-connected layers are directly copied from the $2D$ to the $3D$ architecture. The only exceptions are the parameters of the last discriminator layer which are initialized randomly. This is necessary because in $3D$, the last discriminator layer (which classifies videos as real or fake) accumulates over the full sequence length.

The temporal stride is kept dense throughout all network layers, to ensure that the full duration of the sequence is preserved through the whole

Layer	Kernel	Padding	Stride	Output Size
G_fc8	/	/	/	8, 4096
G_fc7	/	/	/	8, 4096
G_fc6	/	/	/	8, 4096
reshape layer	/	/	/	4x4x8, 256
G_upconv5	4x4x3	1x1x1	2x2x1	8x8x8, 256
G_conv5	3x3x3	1x1x1	1x1x1	8x8x8, 512
G_upconv4	4x4x3	1x1x1	2x2x1	16x16x8, 256
G_conv4	3x3x3	1x1x1	1x1x1	16x16x8, 256
G_upconv3	4x4x3	1x1x1	2x2x1	32x32x8, 128
G_conv3	3x3x3	1x1x1	1x1x1	32x32x8, 128
G_upconv2	4x4x3	1x1x1	2x2x1	64x64x8, 64
G_upconv1	4x4x3	1x1x1	2x2x1	128x128x8, 32
G_upconv0	4x4x3	1x1x1	2x2x1	256x256x8, 3

Table 4.1.: Network structure of 3D generator. The spatiotemporal dimensions for kernel, padding and stride are shown in the order [width x height x time]. The output size is written as [width x height x time, # channels]. For training, the spatial duration is set to $T = 8$ frames.

network. Tables 4.1 and 4.2 show a detailed overview of our network architectures after parameter transfer for generator and discriminator.

Layer	Kernel	Padding	Stride	Output Size
D_conv1	7x7x7	0x0x3	4x4x1	56x56x8, 32
D_conv2	5x5x5	0x0x2	1x1x1	52x52x8, 64
D_conv3	3x3x3	0x0x2	2x2x1	25x25x8, 128
D_conv4	3x3x3	0x0x1	1x1x1	23x23x8, 256
D_conv5	3x3x3	0x0x1	2x2x1	11x11x8, 256
D_pool5	11x11x1	/	/	1x1x8, 256
D_fc1	/	/	/	8, 1024
D_fc2	/	/	/	8, 512
D_fc6	/	/	/	8,512
D_fc7	/	/	/	1,2

Table 4.2.: Network structure of 3D discriminator. The spatiotemporal dimensions for kernel, padding and stride are shown in the order [width x height x time]. The output size is written as [width x height x time, # channels]. For training, the spatial duration is set to $T = 8$ frames.

4.3. Loss Function

Dosovitskiy and Brox [9] proposed a generator loss function which minimizes distances in image and feature space, additionally to the adversarial loss. This boosts the invariance with respect to irrelevant transformations, and the sensitivity to local image statistics, according to [9]. The composition of these three losses leads to better results in image quality, because the additional feature loss reflects the perceptual similarity of images. The loss in image space helps to stabilize the training process, as adversarial training is known to be unstable and sensitive to hyperparameter selection [17].

Fuchs [14] adapted this composite loss to work in the video domain, which is defined as:

$$\mathcal{L} = \lambda_{feat}\mathcal{L}_{feat} + \lambda_{adv}\mathcal{L}_{adv} + \lambda_{vid}\mathcal{L}_{vid}, \tag{4.3}$$

with the Euclidean loss in feature space $\mathcal{L}_{feat}$, the adversarial loss $\mathcal{L}_{adv}$ and the Euclidean loss in video space $\mathcal{L}_{vid}$. All three parts are weighted with a specific parameter λ. We used this combined loss function $\mathcal{L}$ in our approach, as we want to train our GAN on video data instead of images.

The loss in video space $\mathcal{L}_{vid}$ is written as

$$\mathcal{L}_{vid} = \sum_{h,w,t} \|G(\mathbf{z}) - \mathbf{x}\|_2^2, \tag{4.4}$$

where $G(\mathbf{z})$ and $\mathbf{x}$ are the generated and real videos, respectively, with the dimensionality $[HxWxTx3]$, where H and W define the spatial resolution, and T defines the duration of the RGB video. The indices h, w, t are therefore the spatiotemporal indices of the video. The loss in video space helps to stabilize the training process.
The feature loss $\mathcal{L}_{feat}$ is calculated using the pre-trained encoder network E, and is defined as

$$\mathcal{L}_{feat} = \sum_{h,w,t} \|E(G(\mathbf{z})) - E(\mathbf{x})\|_2^2. \tag{4.5}$$

The encoder network E extracts features form real and generated videos. During GAN training, the network parameters of the encoder E are frozen. The adversarial loss $\mathcal{L}_{adv}$ is defined as

$$\mathcal{L}_{adv} = -\log D(G(\mathbf{z})). \tag{4.6}$$

Furthermore, it is also necessary to optimize the discriminator, to successfully classify real and fake videos. The generator and discriminator are trained concurrently, using

$$\mathcal{L}_{discr} = -[\log(D(\mathbf{x})) + \log(1 - D(G(\mathbf{z})))], \tag{4.7}$$

as the loss function for the discriminator .

4.4. Generating Output using Activation Maximization

To generate videos for a specific action, we use our spatiotemporal generator with a technique called Activation Maximization (AM) [11, 66, 46] to maximize the probability that a generated video belongs to a specific class.

For this purpose, we use a task specific condition network Φ. This condition network is a pre-trained classifier, and delivers output probabilities for each class in the dataset. The goal of this technique is to find a specific code vector $\hat{\mathbf{z}}$ that maximizes the activation of a neuron h. Formally, the objective function can be written as

$$\hat{\mathbf{z}} = \arg\max_{\mathbf{z}}(\Phi_h(G(\mathbf{z})) - \lambda|\mathbf{z}|), \tag{4.8}$$

with the parameter λ used for the regularization term. The experimental setup using our generator in combination with a condition network is shown in Figure 4.4.
The generator serves as a learned natural image prior, which makes it possible to synthesize realistic, interpretable videos. Without using the generator as prior, it would not be possible to generate realistic-looking samples, because the set of all possible outputs is too vast.

Finding the optimal input $\hat{\mathbf{z}}$ is an iterative process, where the gradient is back-propagated through both the condition network Φ and the generator G to the input. In general, the aim is to perform AM until the softmax probability of the target neuron h is maximized, which means that the softmax output for this neuron is equal to 1. Figure 4.5 shows the iterative process of AM, transforming random code vectors to a specific action video. Other approaches using activation maximization in combination with generative models can be found in [39, 38, 14].

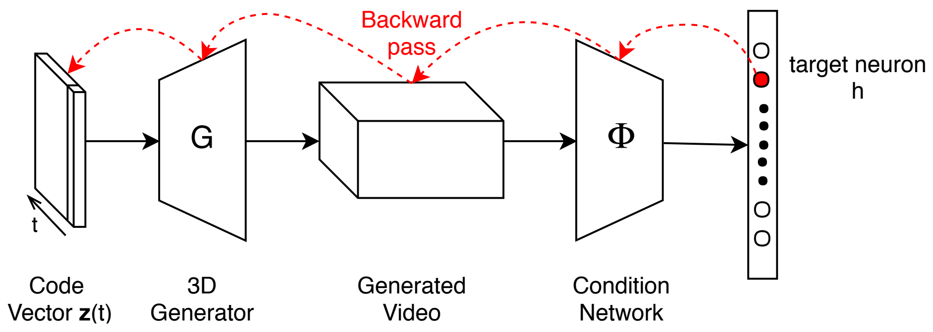


Figure 4.4.: Video generation using activation maximization. The 3D generator G uses a random code vector $\mathbf{z}(t)$, which consists of multiple 1x4096 vectors, and each vector corresponds to one frame of the generate video. This video serves as input for a pre-trained condition network Φ. The goal is to maximize the activation for a specific target neuron h, and to back-propagate the gradient through both networks to adjust the code vector $\mathbf{z}(t)$.

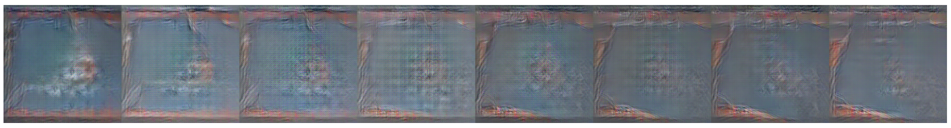
4.5. Condition Networks

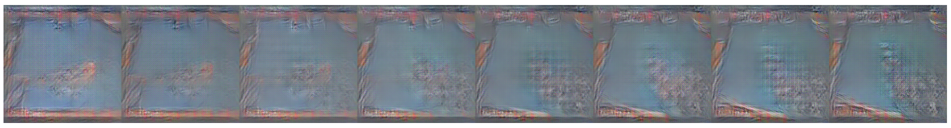
A condition network is a trained classifier, which helps to perform activation maximization. Therefore, the generator and the condition network should operate in a similar domain, using similar datasets for training. As we want to generate videos for action recognition we were looking for classifier networks trained on datasets like UCF-101 [49] and Sports-1M [25]. This section gives a brief overview about the structure of the condition networks that we used in our experiments.

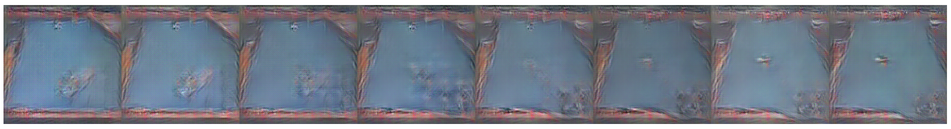
4.5.1. LRCN - Long-term Recurrent Convolutional Network

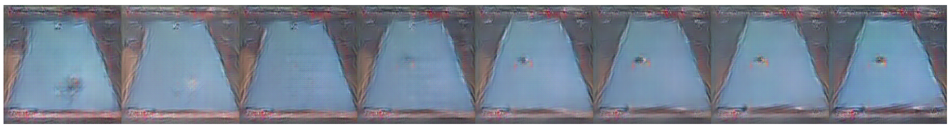
The LRCN [8] is a deep convolutional network with additional recurrent neural networks for visual recognition. The advantage of this network is that it is able to handle a sequence of images with variable length. This is possible with the usage of Long Short-Term Memory (LSTM) [20]. LSTM cells are a special kind of neural networks, designed for processing sequences of data. The main idea is to share parameters across different parts of the model. This can be achieved by applying the same parameter update rule to all

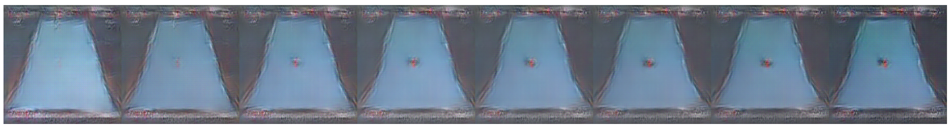
(a) Number of iterations: 0


(b) Number of iterations: 10


(c) Number of iterations: 20


(d) Number of iterations: 30


(e) Number of iterations: 40


(f) Number of iterations: 50

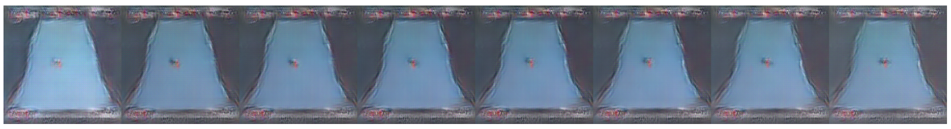

(g) Number of iterations: 60

Figure 4.5.: Visualization of activation maximization for class **Billiard** using the LRCN condition network. The Figures 4.5a to 4.5g show the process of activation maximization applied to 8 frames at different iteration steps. The sequence of images changes from random patterns at Figure 4.5a to billiard at Figure 4.5g. The videos were generated with our spatiotemporal GAN and the LRCN condition network.

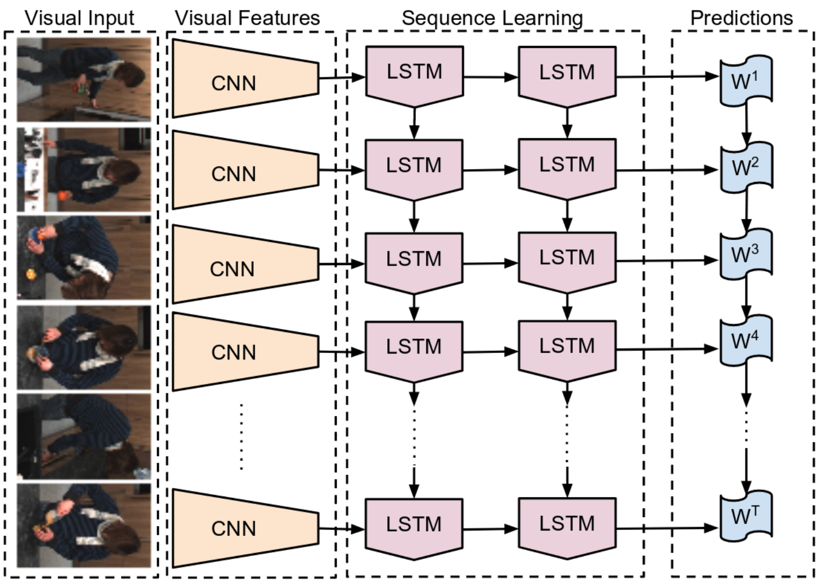


Figure 4.6.: Structure of LRCN. In the first stage, CNNs are used to extract features from the input. Then, LSTM cells are used for sequence learning, and afterwards the output for a given visual input is predicted. (Figure from [8])

network layers recurrently [16]. LSTM cells have the additional advantage that self-loops are created in the network where the gradient can flow for a long duration. This helps remembering information that was gained some time ago.
This property makes LSTM interesting for processing videos. Figure 4.6 shows the network architecture of LRCN. It takes a sequence of images as input, where each image is cropped to a spatial resolution of $227x227px$.

4.5.2. C3D - 3D Convolutional Networks

The C3D network [55] is another neural network used for classification of video data. It uses 3D kernels in all convolutional layers with size $3x3x3$ in space and time. These kernels make it possible to extract features not only spatially, but across frames in time. Therefore, it is a simple and effective way to achieve video classification for action recognition. C3D takes a sequence of 16 images as input, with a spatial resolution of $112x112px$. Figure 4.7 shows the network architecture of the C3D network.

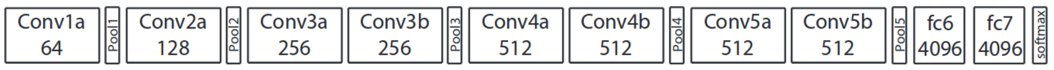


Figure 4.7.: Network structure of C3D. The network consists of eight convolutional,five max-pooling and two fully-connected layers, followed by a softmax output layer. All 3D convolution kernels are $3x3x3$. The number of filters is denoted in each box. The 3D pooling layers are denoted from $pool1$ to $pool5$. All pooling kernels are $2x2x2$, except $pool1$ which is $1x2x2$. Each fully-connected layer has 4096 output units. (Figure from [55])

4.6. Experimental Results

This section summarizes our experiments using our 3D generator combined with different condition networks to generate videos for action recognition. First, we explain our approach to fine-tune the spatiotemporal GAN for action recognition. After training, we combine the 3D generator with the LRCN condition network, building an end-to-end network to perform activation maximization. Then, we look at the quantitative results of our generated videos for action recognition. At last, we quantitatively evaluate our results using the Inception score.

4.6.1. Fine-tuning our Spatiotemporal GAN using Video Data

As described in Section 4.2, the network parameters for the spatiotemporal GAN were transferred from a 2D generator pre-trained on ImageNet. In this section, we describe our attempt to fine-tune the ST-GAN for action recognition, using the UCF-101 dataset.

Data Preprocessing

During fine-tuning on video data, the spatiotemporal GAN takes batches of 8 consecutive images as input, each of them representing a video sequence. To achieve this, we extract images from each video of the UCF-101 dataset, at a frame rate of 5fps. The resulting frames are then center cropped to $227x227px$ and grouped to sequences of 8 frames. Figure 4.8 shows examples of 8 frame video sequences after preprocessing.

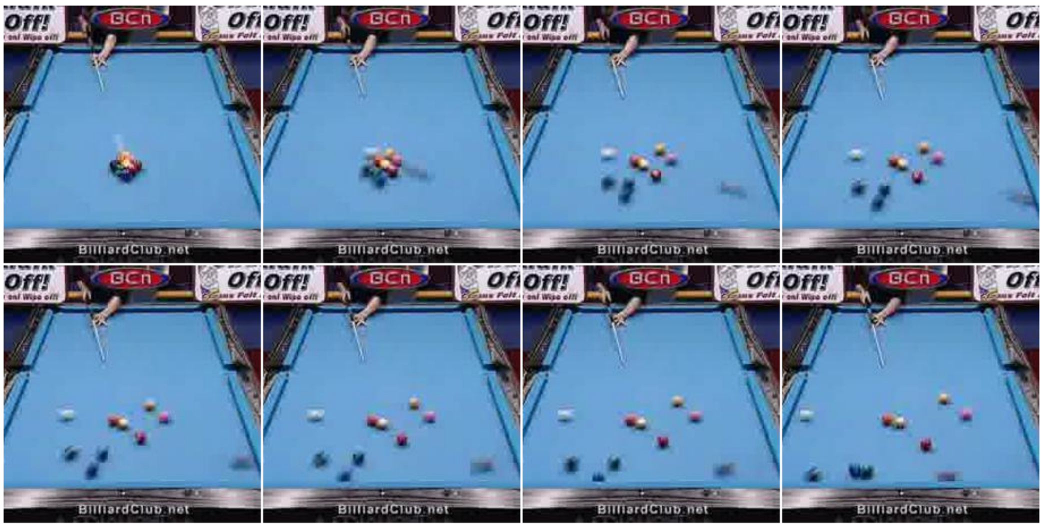
Training Details

During training, we use a mini-batch size of 8, with every mini-batch containing 8 consecutive frames. For optimization, we use the Adam Optimizer (see Section 3.3) provided by the Caffe framework [24], with the standard parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\alpha = 0.001$. The original learning rate is multiplied by the factor $\frac{1}{100}$, to make sure that the pre-trained weights will not be destroyed. During training, we follow the example of [9] by decreasing the learning rate further by the factor $\frac{1}{2}$ after the iterations $[10k, 25k, 50k, 70, 90k]$. Weight decay is used as regularization, with a regularization parameter $\lambda = 4 \cdot 10^{-4}$.
For training the generator, our aim is to minimize the composite loss function $\mathcal{L}$ defined in Equation 4.3. Concurrently, we want to minimize the discriminator loss $\mathcal{L}_{discr}$, which is defined in Equation 4.7.

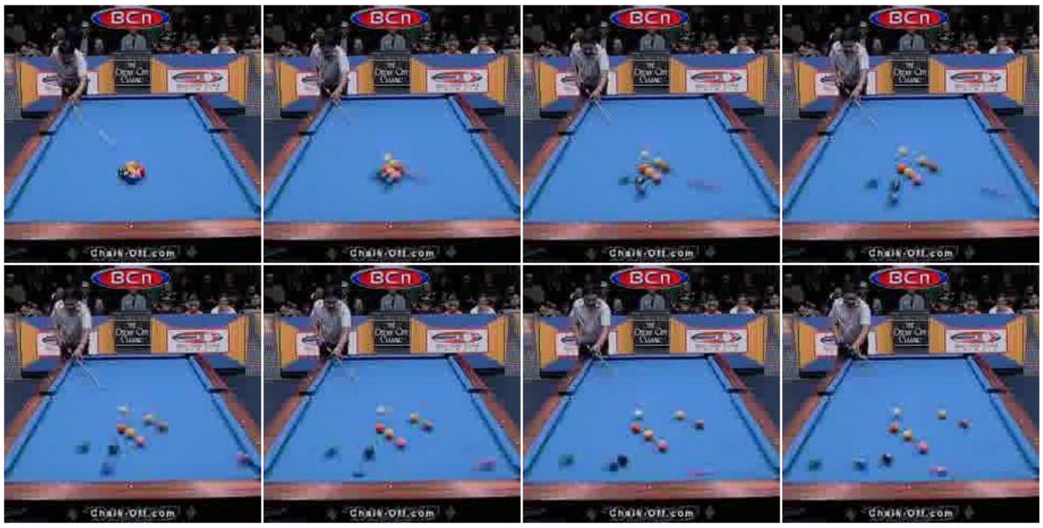
To make sure that the training for the generator and discriminator is balanced, we follow [9] and monitor the ratio r. This ratio is defined as $r = \frac{\mathcal{L}_{discr}}{\mathcal{L}_{adv}}$, with $\mathcal{L}_{discr}$ and $\mathcal{L}_{adv}$ defined in Equation 4.7 and 4.6, respectively. Depending on the ratio r, we define three different training modes:

$$mode = \begin{cases} \text{train G and D} & \text{for } 0.1 \leq r \leq 10 \\ \text{train G only} & \text{for } r < 0.1 \\ \text{train D only} & \text{for } r > 10. \end{cases} \tag{4.9}$$

The training is performed on the GPU, to make the computation feasible.



(a) First video.



(b) Second video.

Figure 4.8.: Preprocessed UCF-101 training videos. The original data for both videos belong to the class **Billiard**. A training video consists of 8 frames, all center cropped to a spatial resolution of $227x227px$.

From now on, we refer to the fine-tuned version of our spatiotemporal GAN as $STGAN_{ucf}$. The pre-trained ImageNet version is named $STGAN_{ImNet}$. We make this distinction because we use both versions for qualitative and quantitative evaluation.

4.6.2. Video Generation using LRCN as Condition Network

This subsection describes our approach to generate videos using our generator combined with the LRCN condition network using AM. First, we explain how to combine the generator and the condition network to one end-to-end network. Then, we evaluate the video quality, and calculate the Inception score as quantitative evaluation measure.

Combining 3D Generator and LRCN

To be able to use activation maximization to generate videos, we have to combine our generator and a condition network to one end-to-end network (see Figure 4.4). We point out all steps which are necessary to combine these two networks:

Adaptation of layer dimensions. To be able to forward the generated video to the condition network, the video has to match the spatial and temporal input format of the LRCN. For the spatial case, we center crop each image of our generated video to $227x227px$. To match the temporal format, we reshape the LRCN input layer, to be able to dynamically process input videos up to at most 160 frames.
Calculation of the class probability using LRCN. The videos are then pushed through the LRCN, which outputs a probability of a video belonging to a specific UCF-101 class.
Back-propagate the gradients through the LRCN. The gradients are calculated for the last LRCN layer, and then back-propagated through the condition network to the video domain. Back-propagating the gradients in deep networks can lead to several failure cases, for example exploding and vanishing gradients [16]. Therefore, it is important to carefully monitor the gradients, and if necessary to apply gradient clipping or other normalization

techniques to avoid these problems.
Expand gradients to match output layer dimension of generator. In the video domain, it is again necessary to match the dimensions of the last output layer for the backward pass. Therefore, the dimension of the gradients are expanded with zero padding, to match the spatial output format of the generator, which is $256x256$.
Back-propagation through generator and updating the code vector. The last step is to use the gradient information to update code vector $\mathbf{z}$. To achieve this, we adapt the update rule from [38] to the spatiotemporal domain. One important detail is that Gaussian noise $\mathcal{N}(0, 10^{-17})$ is added to the gradients, which is important to boost the variance of the generated samples. We perform up to 100 iterations of AM, where the learning rate for the update function starts at 1.0 and linearly decays to 0.1. Other AM specific parameters are taken from [38].

Note that we also experimented with other condition networks. We used the C3D as condition network, following the above mentioned steps to construct an end-to-end network with our generator. The qualitative results using C3D are provided in Appendix A.4.
Another attempt was to use [60] as condition network. However, it was not possible to back-propagate a meaningful gradient, and therefore AM with this network was not possible.

Evaluating Qualitative Results

The input for the generator is a sequence of random code vectors, each of them drawn from a Gaussian distribution $\mathcal{N}(0, 1)$. We perform AM for each class in the UCF-101 dataset. We decided to sample videos with a length of 16 frames in all main experiments, to be consistent in our evaluations. The qualitative results are shown for the classes **Apply Lipstick**, **Baseball Pitch**, **Billiard** and **Clean and Jerk** as running examples for all our experiments in this chapter.

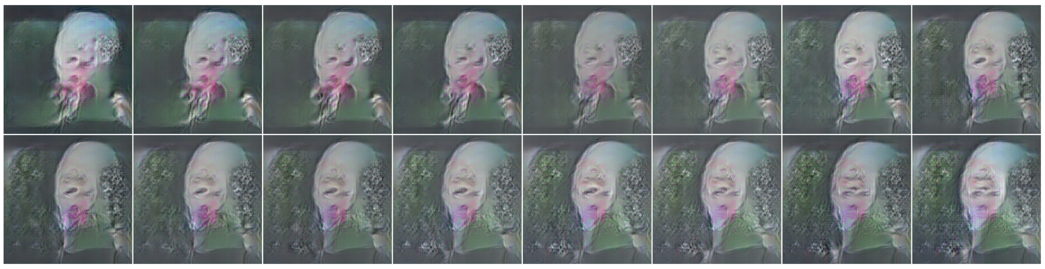
Figure 4.9 shows the results using $STGAN_{ucf}$ with LRCN as condition network. Figure 4.10 shows results using the pre-trained $STGAN_{ImNet}$. Experiments with longer sequences can be found in Appendix A.3.

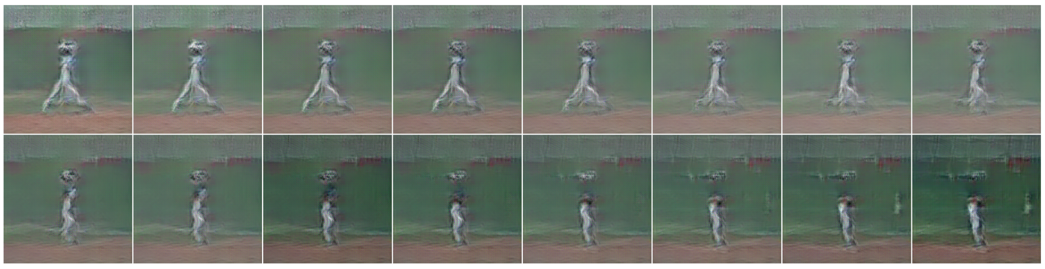
By comparing the results of Figure 4.9 and Figure 4.10, several things are striking. Figure 4.9a and Figure 4.10a show results for the class **Apply Lipstick**. One can see that both GANs do have problems to generate faces correctly. This is because our GANs are not able to learn the correct amount of face parts, like eyes and nose. Similar problems occur for all classes which focus on human faces. The reason for this problem could be the usage of max-pooling at some stage of the convolutional neural network, which makes the representation invariant to small translations of the input. This means, that the network only focuses on a feature being absent or present, but not on how many times it occurs (see [16] for more information).
Videos generated for the classes **Baseball Pitch** (Figure 4.9b and 4.10b) and **Billiard** (Figure 4.9c and 4.10c) show results with high image quality. This means, that the quality for each individual frame is high, and the whole video shows a reasonable sequence of consecutive frames which represent a specific action. However, it is not obvious if the fine-tuning of the generator provides a significant advantage for generating frames with temporal coherence. One can also argue that the combination of weight transfer to 3D and a condition network pre-trained on video action recognition are enough to generate videos. This is the case for the $STGAN_{ImNet}$, which was never trained on video data, and therefore the information of temporal coherence has to be provided by the condition network during AM.
Figure 4.9d and Figure 4.10d show samples for the class **Clean and Jerk**. Because we use activation maximization, the generated videos indicate what is most important for the condition network to discriminate different classes. In this case, the network clearly focuses on the weights. It seems that the human body is not important for classifying this action, and therefore the frames contain no human body. This insight could possibly be used to further understand how convolutional neural networks work, and could therefore help to improve the quality of a classifier.

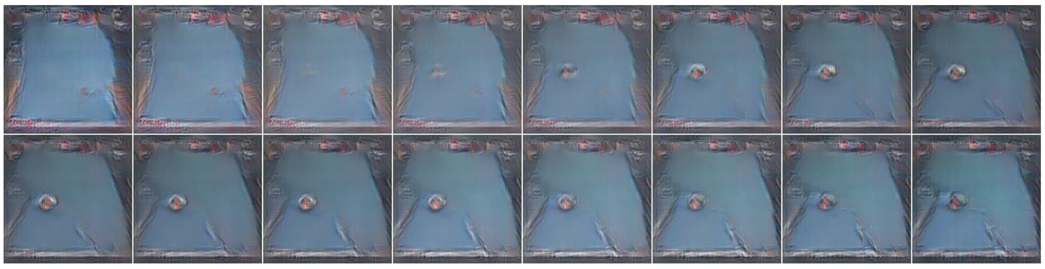
It also appears that the video quality is higher if the pre-trained $STGAN_{ImNet}$ is used for sampling. This leads to the conclusion that our task specific fine-tuning does not increase the GAN performance. There are several issues that possibly lead to the decrease of qualitative performance:

- **The UCF-101 dataset is too small.** The size of the dataset is crucial for training neural networks. A careful selection of hyperparameters and a suitable optimizer can boost learning, also for small datasets. However, if the dataset is too small it is not possible to learn anything meaningful.

- **Wrong choice of hyperparameters.** Training is highly dependable on the hyperparameter setup. Especially, the learning rate is crucial for fine-tuning. Although we decreased the learning rate significantly, as it is common during fine-tuning, it is still possible that our choice of hyperparameters is not optimal.
- **The difference between datasets for pre-training and fine-tuning is too big.** During training, neural networks typically learn to operate in a very specific domain, depending on the task and the dataset. When we try to fine-tune this network afterwards on a different dataset, we try to learn to operate on a different task. This forces the network to extrapolate to another manifold (function space), rather than interpolate in the learned space. It seems that our GAN is not able to do that, which in other words means that the network is not able to generalize well to other tasks. Yosinski et al. [64] already stated that the transferability of features decreases as the distance between the base task and the target task increases.


(a) Apply Lipstick


(b) Baseball Pitch


(c) Billiard

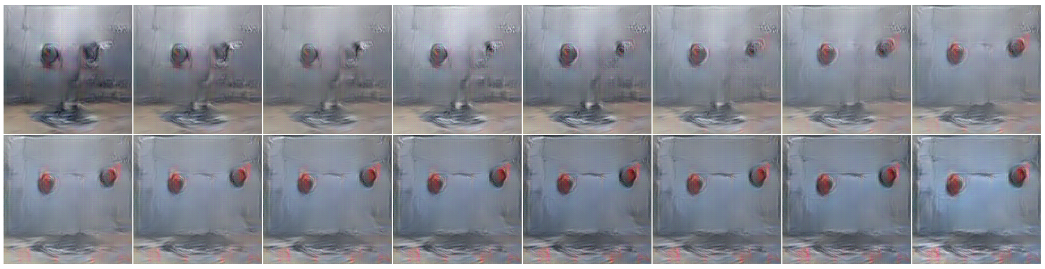
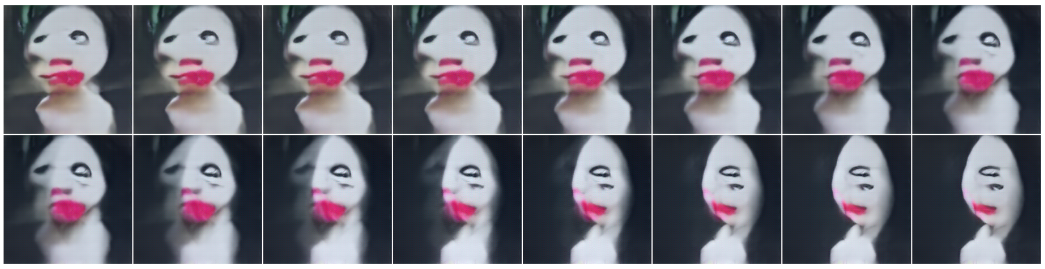
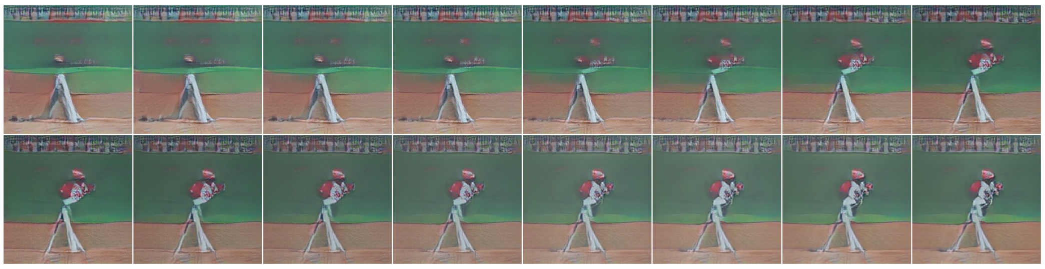
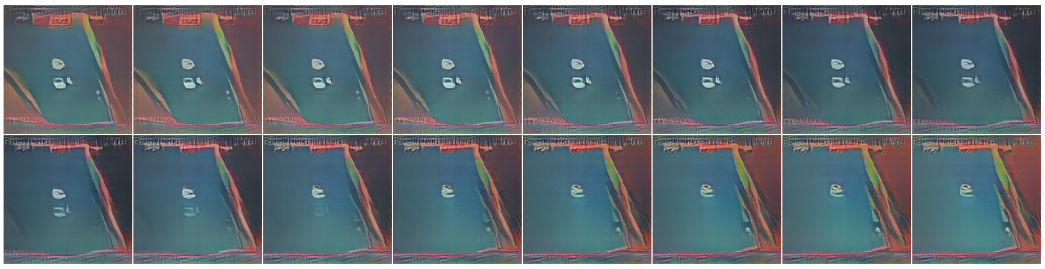

(d) Clean and Jerk

Figure 4.9.: Visualization of generated videos using the $STGAN_{ucf}$ and the LRCN as condition network. Each video consists of 16 consecutive frames. Results are shown for the classes **Apply Lipstick**, **Baseball Pitch**, **Billiard** and **Clean and Jerk**.


(a) Apply Lipstick


(b) Baseball Pitch


(c) Billiard

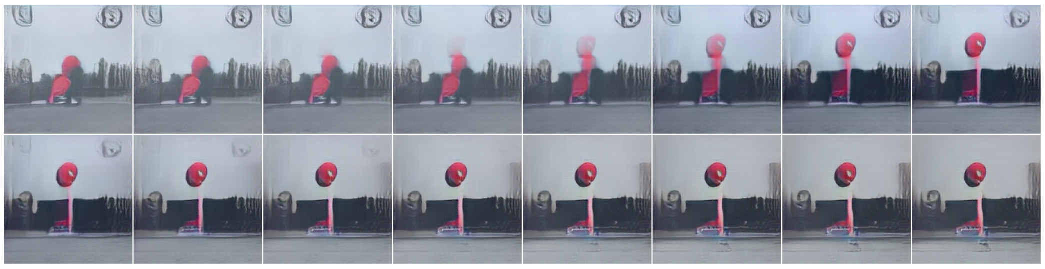

(d) Clean and Jerk

Figure 4.10.: Visualization of generated videos using the $STGAN_{ImNet}$ and the LRCN as condition network. Each video consists of 16 consecutive frames. Results are shown for the classes **Apply Lipstick**, **Baseball Pitch**, **Billiard** and **Clean and Jerk**.

4.6.3. Quantitative Evaluation using the Inception Score

As mentioned in Subsection 3.4.3, a main drawback of GANs is that it is hard to quantitatively evaluate the results. The dataset distribution is learned implicitly, which means that only the generated samples can be evaluated. This subsection describes our approach to quantitatively evaluate our GAN results using the Inception Score (IS). We compare our results to other state-of-the-art GANs, which also operate in the field of video generation for action recognition. We computed the IS following Equation A.2, see Appendix A.1.

Upper bound for the IS. The upper bound of the IS depends on the number of classes in the dataset [4]. The UCF-101 dataset contains 101 classes of different actions, and therefore the maximum possible IS using this dataset is 101.
Evaluation procedure. In order to compute the Inception score, we generate 10000 videos in each experiment, where all videos are normally distributed over the whole UCF-101 classes. Afterwards, we use one of the networks described in Section 4.5, which are pre-trained to classify UCF-101 videos. The idea is, that the classifier should be able to correctly classify the generated videos. The higher the number of correctly classified videos, the higher is the IS. Another criterion for the IS is the variance in data. In the optimal case, the generated videos are uniformly distributed over all UCF-101 classes. An uneven distribution would be an indication for mode dropping.

Additionally, we calculate the IS on the whole UCF-101 dataset. This serves as a key score, which indicates a real dataset. Table 4.3 shows the IS of our approaches compared to other state-of-the-art GANs for action recognition.

By analyzing the results in Table 4.3, we gain the following insights:
Fooling the Inception Score is possible. The results show that some of our methods outperform the recent state-of-the-art video generation GAN approaches (Video GAN and TGAN), according to the IS. However, a close look at the scores shows that some results are too good to be true, and therefore point to a drawback in the IS metric. Especially, if the same network is used as condition network for AM and as classifier, the IS score can be fooled, because the generated samples are in fact generated to have

a high probability with a certain condition network. This means that our experiments prove that it is easily possible to fool the IS.

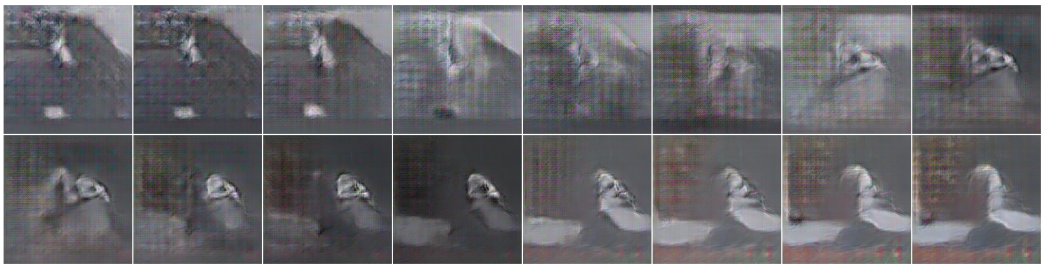
Generated samples are better than real? Our record IS, using the $STGAN_{ImNet}$ with LRCN as condition network **and** as classifier, is even higher than the IS calculated for the UCF-101 dataset. As mentioned above, activation maximization directly optimizes the activation of a certain neuron, which is one of the properties the IS aims for. Furthermore, as we sample our videos uniformly distributed over all UCF-101 classes, also the second criterion for a high IS is satisfied. This results in a very high IS, which is even higher than the score for real data. However, by comparing the generated video in Figure 4.10c with real image sequences shown in Figure 4.8, it is obvious that the image quality of the generated samples is lower. The main reason why the UCF-101 dataset does not achieve a higher IS is because the classes are not uniformly distributed in the dataset.

Our GAN outperforms other state-of-the-art approaches. We performed experiments with different networks for conditioning and classification. This is, in our point of view, the only valid way to use AM in combination with the IS as evaluation metric. Using the LRCN as condition network and the C3D network for evaluation, we achieve an IS of 23.44, which clearly outperforms the other approaches.

Fine-tuning the GAN does not improve the performance. In all experiments, the fine-tuned $STGAN_{ucf}$ performs worse than the pre-trained $STGAN_{ImNet}$, according to the IS. This confirms our suspicion from Subsubsection 4.6.2, that fine-tuning does not improve the performance of our generator.

Not every classifier network is equally suitable as condition network. We also evaluated the IS using the C3D as condition network. The qualitative results are provided in Appendix A.4. Table 4.4 shows the IS results of our generators combined with the C3D as condition net. The results verify the findings from Table 4.3, that the IS can easily be fooled with activation maximization.

Furthermore, by directly comparing the IS in Table 4.3 and Table 4.4, one can see that the achieved results with the C3D as condition network are considerably lower than with the LRCN for all experiments. Although we were able to back-propagate the gradient with C3D, and succeeded in maximizing the activation for each neuron during sampling, the results are worse than with LRCN. This leads to the assumption that not every classifier is


(a) BaseballPitch

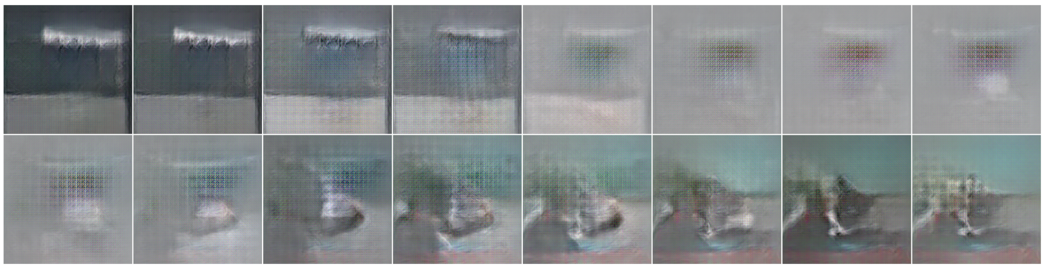

(b) Clean and Jerk

Figure 4.11.: Adversarial examples generated with the $STGAN_{ucf}$ and the C3D. The videos are maximized for the classes **BaseballPitch** and **Clean and Jerk**. The C3D classifier network is able to correctly classify them, however the samples do not look like realistic actions.

equally suitable as condition network, even if activation maximization can be achieved. It seems that the network architecture of the condition network is highly important for activation maximization.

Directly optimizing the IS leads to adversarial examples. Adversarial examples [16, 53, 45] are input data for neural networks that cause unexpected output. In our case, the IS in Table 4.4 with the $STGAN_{ImNet}$ and the C3D as condition network and classifier is 66.33. This would suggest that the generated samples are near photorealism. However, Figure 4.11 shows generated videos for this method. By analyzing the samples, one can see that the videos do not contain any real objects, which indicates that these are adversarial examples. Salimans et al. [45] stated that directly optimizing the IS would lead to adversarial examples, but did not provide results to prove this claim.

Method	**Condition Network**	**Inception Score - Classifier Network**	**Inception Score**
Video GAN [57]	/	C3D	8.18
Conditional TGAN [43]	/	C3D	15.83
$STGAN_{ucf}$	LRCN	C3D	**17.42**
$STGAN_{ucf}$	LRCN	LRCN	32.15
$STGAN_{ImNet}$	LRCN	C3D	**23.44**
$STGAN_{ImNet}$	LRCN	LRCN	82.87
UCF-101 dataset	/	C3D	70.07

Table 4.3.: Comparison of the IS for different models. State-of-the-art approaches (Video GAN and TGAN) are compared to our approaches using the STGAN with the LRCN as condition network. Also, the IS for the UCF-101 dataset is stated. The **bold** IS scores are achieved by using different networks for conditioning and classifying, which is in our view the only valid way to calculate a meaningful IS. By comparing the IS, one can see that our approaches outperform the other methods according to the IS.

Method	**Condition Network**	**Inception Score - Classifier Network**	**Inception Score**
$STGAN_{ucf}$	C3D	C3D	16.23
$STGAN_{ucf}$	C3D	LRCN	**1.74**
$STGAN_{ImNet}$	C3D	C3D	66.33
$STGAN_{ImNet}$	C3D	LRCN	**3.97**

Table 4.4.: Additional IS results using the C3D as condition network. The IS for all methods is significantly lower using the C3D as condition network, instead of the LRCN (see Table 4.3). The **bold** scores are meaningful (using different networks for conditioning and classifying), which indicates that the C3D is not suitable as condition network.

4.7. Conclusion

This chapter described our approach to generate videos for action recognition, at a spatial resolution of $227x227px$. We were able to extend the generative model proposed by [9] from spatial to spatiotemporal domain, and used it for video generation. Our approach outperforms other state-of-the-art methods in terms of resolution and image quality. To back up our claim of higher image quality, we used the Inception score as quantitative measure and compared our approach to other methods.
Our evaluation also pointed out that the Inception score can be fooled by directly optimizing the properties of it, which is the main weakness of this metric.
According to the network architecture, we can conclude that using our approach enabled us to generate videos at a fairly high resolution, for a large number of classes. We achieved this by using a condition network with activation maximization, where our generator is used as image prior. We found out, that not every classifier is equally suitable as condition network. Therefore, further experiments with other condition networks could lead to information about what makes a classifier a good choice for a condition network.
Another finding of this chapter is, that fine-tuning our spatiotemporal GAN for action recognition did not increase the image quality. We believe that the small size of the UCF-101 dataset is one main reason. Another one could be that the domains for pre-training and fine-tuning are too different. Therefore, an interesting experiment would be to train the generator from scratch for action recognition, using a dataset with a higher number of training data (for example the Sports-1M dataset [25]).
To conclude, our spatiotemporal GAN enables us to generate videos for a high number of different action recognition classes. However, there is a still a long way to go to generate realistic-looking action videos.

5. Generating High Resolution Action Videos

This chapter describes our approach to generate videos with increased spatial resolution. We use the network structure proposed by [26] as GAN. We describe how we trained this GAN for action recognition in the image domain and then explain our approach to generate videos by interpolating in the latent space of the generator. At last, we quantitatively evaluate the results by measuring the diversity of the generated images using structural similarity.

5.1. Progressively Growing GAN

To generate videos with high image resolution, we decided to use the framework provided by [26], which we refer to as Progressively growing GAN (PGAN). We briefly discuss the most important aspects of the network architecture, the loss function and the techniques that are used to make the training more stable. More details are provided by [26].

5.1.1. Network Architecture

The main advantage of this network architecture is, that both the generator and discriminator are growing progressively during training. The key idea is, that the mapping from latent space to high-resolution images is easier to learn stepwise. Therefore, the network starts with convolutional layers operating on a spatial resolution of $4x4px$, and grows while training progresses. Figure 5.1 shows the progressively growing network architecture

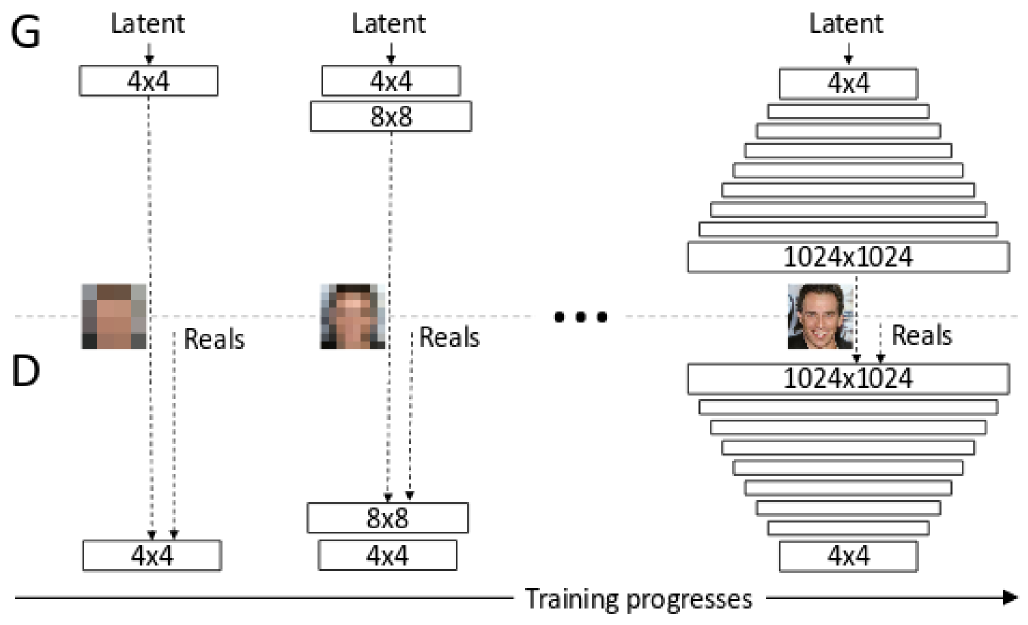


Figure 5.1.: Progressively growing GAN network architecture. Karras et al. [26] used this structure to generate images of human faces, at a resolution of $1024x1024px$. As training progresses, the structure of the generator and discriminator grow progressively. We use the same architecture for our experiments, with the only difference that we train PGAN to generate images for action recognition at $512x512px$. (Figure from [26])

during training.

The benefits of this structure are that early layers converge quickly, and that there are only a few layers to train from scratch at the same time. This leads to a significant reduction of training time.

5.1.2. Loss Function

PGAN uses a loss function called WGAN-GP [18]. The objective of this loss function is to minimize the Earth-Mover (also called Wasserstein-1) distance. The Wasserstein-1 distance is defined by the minimal cost of transporting mass in order to transform the distribution p into another distribution q.

"The Wasserstein-1 distance is continuous everywhere and differentiable almost everywhere, under mild assumptions (that the 1-Lipschitz constraint holds for the discriminator). A differentiable function is 1-Lipschitz if and only if it has a gradient with norm at most 1 everywhere." [18]

WGAN-GP achieves this constraint by using an additional gradient penalty, which constrains the gradient norm of the output with respect to the input, at the discriminator. The advantage of the loss function is, that mode collapse occurs less often, and that balancing the discriminator and the generator is less important. Further information about this loss is provided by [18].

5.1.3. Regularization Techniques

PGAN uses several techniques to improve variation and quality of the output, as well as training stability:
Increasing variation using minibatch standard deviation. At the end of the discriminator, a special layer is added which calculates the standard deviation of a minibatch of images, and adds this information to the feature vector of each image, which is used for classification. Therefore, the discriminator does not only look at an individual image, but also obtains information about the distribution of a set of images. This additional information forces the generator to learn a distribution which is similar to the training data.
Equalized learning rate. This method normalizes the gradient update by its estimated standard deviation dynamically during training, which ensures fair competition between generator and discriminator.
Pixelwise feature vector normalization. This normalization technique is used in the generator. After each convolutional layer, the resulting feature vectors are normalized to unit length for each pixel. Karras et al. [26] claimed that this normalization avoids the scenario where the signal magnitudes in the networks "spiral out of control" due to the competition of both networks.

5.2. Experimental Results

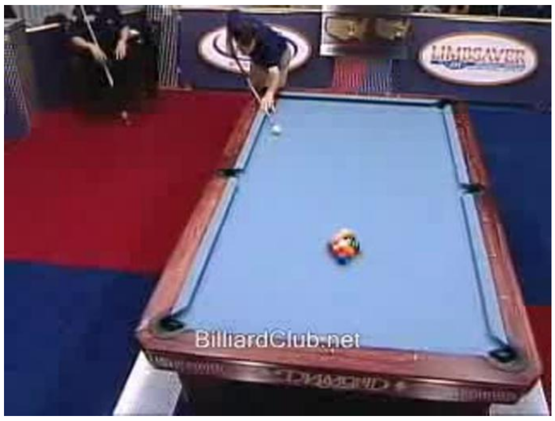
This section describes our experiments to generate high resolution videos for action recognition, using the PGAN framework. First, we talk about the pre-processing of the dataset. Then, we show some results of the trained generator in the image domain. At last, we explain how we generate videos for action recognition, by interpolating in the latent space of the trained generator, and evaluate our results using structural similarity.

5.2.1. Dataset Preprocessing

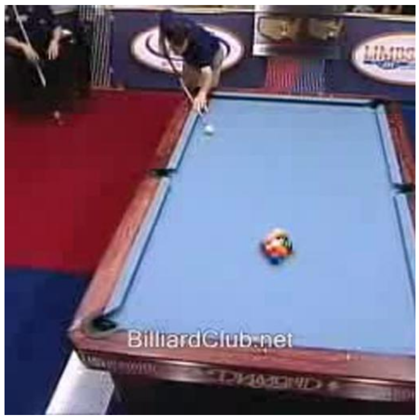
The PGAN framework grows progressively, starting at a resolution of $4x4px$ and doubles until the final resolution is reached, which in our experiments is $512x512px$. This means that we also need training data at different sizes, to train the network at all resolutions. We refer to the different sizes of the images as levels of detail. Preprocessing of the dataset is done in two steps:
Increasing the resolution of the training data. To be able to generate frames at a resolution of $512x512px$, our training data have to have at least the same resolution, or higher. The problem is that there is no action recognition dataset that contains enough high resolution videos to train a generator with. Therefore, we decided to use the UCF-101 dataset to be consistent throughout this thesis, and produce results that are comparable to all our other experiments. All frames in the UCF-101 dataset have a spatial resolution of $320x240px$. We increased the spatial resolution of all input images to $682x512px$ using bicubic interpolation. After resizing to $682x512px$, we center cropped the image to a resolution of $512x512px$, where the aspect ratio is kept the same as in the original image. Figure 5.2 shows the process of resizing one image. We are aware that bicubic interpolation may not be the best technique for super resolution and a better method could lead to better image quality at the end. However, we wanted to focus on video generation and therefore used a method with low computational costs for resizing. We leave the improvement of using a better resizing technique for the future.
Sampling each image at different scales. Each image that is used for training was then downscaled to each of the levels of detail at which PGAN

operates. The size of each image is reduced by $\frac{1}{4}$, starting from $512x512px$ until the smallest resolution of $4x4px$ is reached. The reduction of resolution is achieved by using $2x2$ average pooling on the full image. Figure 5.3 shows an example of all levels of detail for one image.

(a) Original Image ($320x240px$).



(b) Resized Image ($682x512px$).

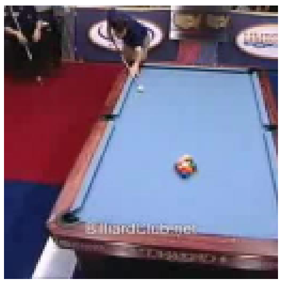


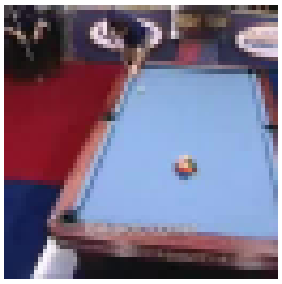
(c) Cropped Image ($512x512px$).

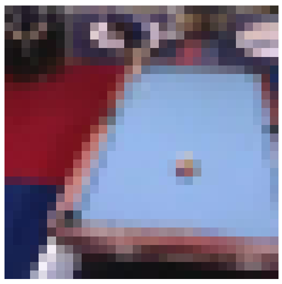
Figure 5.2.: Preprocessing of an input image for PGAN training. The original image is first resized to $682x512px$, and then center cropped to $512x512px$. The sample belongs to the class **Billiard**.

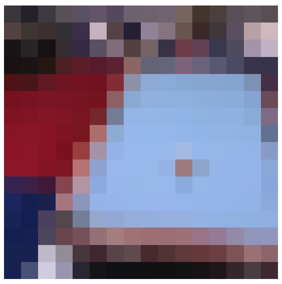
(a) Resolution: 512x512px.

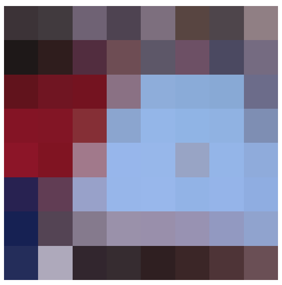
(b) Resolution: 256x256px.


(c) Resolution: 128x128px.


(d) Resolution: 64x64px.


(e) Resolution: 32x32px.


(f) Resolution: 16x16px.


(g) Resolution: 8x8px.

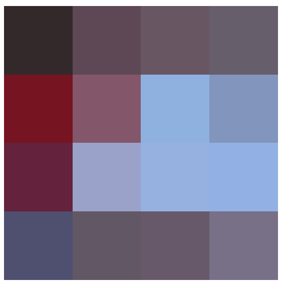

(h) Resolution: 4x4px.

Figure 5.3.: Training sample at all levels of detail. Average pooling is used at each level of detail to reduce the size of the image by $\frac{1}{4}$, until the resolution of 4x4px is reached, which is the smallest resolution PGAN operates on. The sample belongs to the class **Billiard**.

Level of Detail	# Real Images in Thousands	Accumulated Training Duration
4x4px	300	2m 52s
8x8px	600	14m 04s
16x16px	600	42m 19s
32x32px	600	2h 02m 46s
64x64px	600	5h 06m 10s
128x128px	600	10h 50m 23s
256x256px	600	18h 20m 41s
512x512px	4000	3d 20h 04m

Table 5.1.: PGAN training details for the class **Billiard**. The accumulated training duration shows that training at low resolution is fast, whereas training at high resolution is more expensive.

5.2.2. Training Details

Due to the high computational cost of training the PGAN structure, we decided to train our generator for one class of the UCF-101 dataset which is **Billiard**, to ensure a high image quality and acceptable training duration. Further experiments with other classes and multi class training are provided in Appendix A.5.
Before training, we decided to mirror all training images, to double the amount of data from 42938 to 85876 images, and then shuffled them. During training, both networks were trained evenly, which means that there was no further balancing between generator and discriminator. We used the Adam Optimizer for parameter update, with the parameters $\beta_1 = 0.0$, $\beta_2 = 0.99$ and the learning rate $\alpha = 0.001$.
Table 5.1 shows an overview about how many real images were used for training at each level of detail, combined with the accumulated training duration. One can see that training at low resolution is fast, and slows down at higher levels of detail. We parallelized the training using four GPUs (NVIDIA GeForce Titan X Pascal) to speed up training. Note that the number of training images at each level of detail is a multiple of the training dataset size, which means that the discriminator saw every real image multiple times at each level of detail.

5.2.3. One Class Image Generation

After training, we generated images using random code vectors as input for the generator. Each code vector is drawn from a Gaussian distribution $\mathcal{N}(0, 1)$. Figure 5.4 shows generated samples for the class **Billiard** at a resolution of $512x512px$. The samples show a high diversity, which suggests that the variation of the training data is captured by the generator. It is striking that the images contain very specific patterns, like readable text. Goodfellow et al. [17] stated that GANs are not prone to overfitting, because the generator update does not directly depend on the training data. However, by looking at the results, one can at least say that the network architecture has the capabilities to memorize such patterns and generate them after training.
Furthermore, compared to our first approach in Section 4.6, it is obvious that the image quality using PGAN is higher, and also the resolution increased to $512x512px$. However, due to the high computational costs, we only trained PGAN for one class at $512x512px$, which is a clear disadvantage to our first approach.

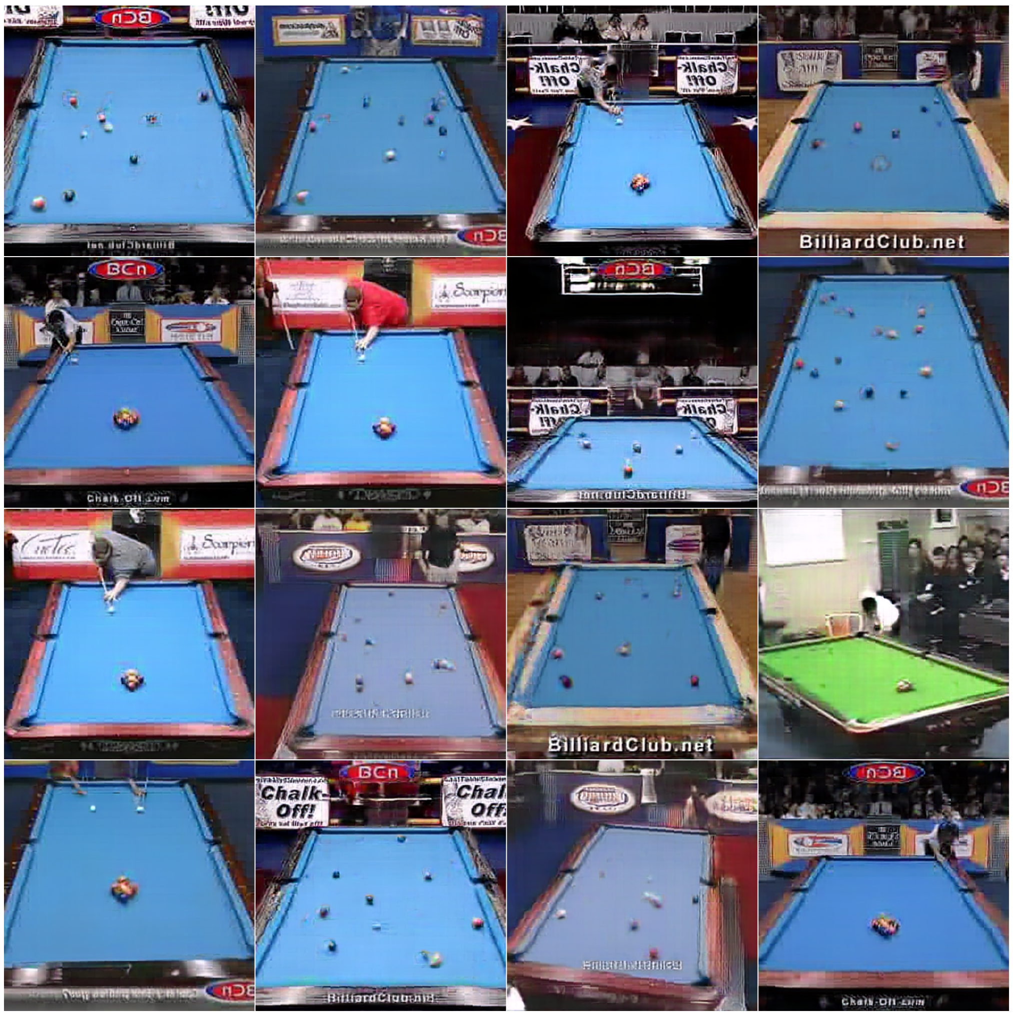


Figure 5.4.: Generated images for the class **Billiard** with a spatial resolution of $512x512px$. One can see that the generator is able to generate images from different scenes, which means that the generator learned to capture the diversity of the dataset.

5.2.4. Video Generation using Latent Space Interpolation

Up to this point, the generator is only able to generate samples in the image domain. For video generation, we use the fact that the whole latent space is continuous, which means that every code vector produces a reasonable output. For video generation, we randomly generated 100 code vectors, and used these code vectors to generate the corresponding images by pushing them through the generator. Afterwards, we manually selected two of these images as start and end frame for the image sequence. Note that our aim is to select a pair of images which already shows the same scene, because this leads to a generated video with reasonable temporal coherence. Figure 5.5 shows an example of such frames.
We denote the code vectors for the start and the end frame as $\mathbf{z}^0$ and $\mathbf{z}^{N-1}$ respectively, where the index N defines the video length. Each code vector has the dimensionality $[1x512]$. We then linearly interpolate between each feature of the two code vectors by following these steps:

1. First, we calculate the stepsize η_i between each i-th feature of $\mathbf{z}^0$ and $\mathbf{z}^{N-1}$:
$$\eta_i = \frac{z_i^{N-1} - z_i^0}{N-1}, \forall i \in [1, 512]. \tag{5.1}$$
2. We then use $\boldsymbol{\eta}$ to calculate the code vector $\mathbf{z}^n$ for all intermediate frames with
$$\mathbf{z}^n = \mathbf{z}^0 + n \cdot \boldsymbol{\eta}, \forall n \in [1, N-2]. \tag{5.2}$$

This method allows us to generate an arbitrary number of frames between the start frame and the end frame. Figure 5.6 shows a generated video with 32 frames, all with a spatial resolution of $512x512px$. By analyzing the image sequence, the temporal coherence is clearly visible.
Further experiments on video generation for other classes are described in Appendix A.6.

5.2.5. Evaluation

We decided to use the Multi-Scale Structural Similarity (MS-SSIM) (see Subsection 3.4.3) for quantitative evaluation. We chose this metric, because

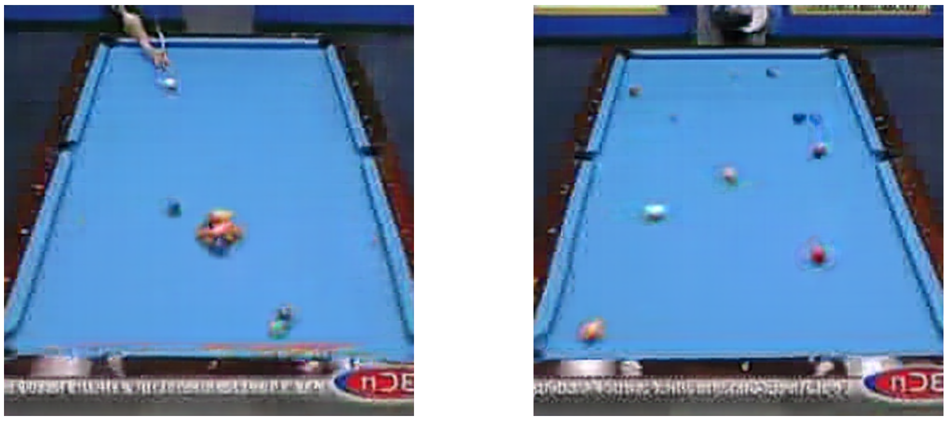

(a) Start frame. (b) End frame.

Figure 5.5.: Start frame and end frame for video generation. These two frames were generated with our PGAN, and their latent vectors serve as starting and ending point for latent space interpolation.

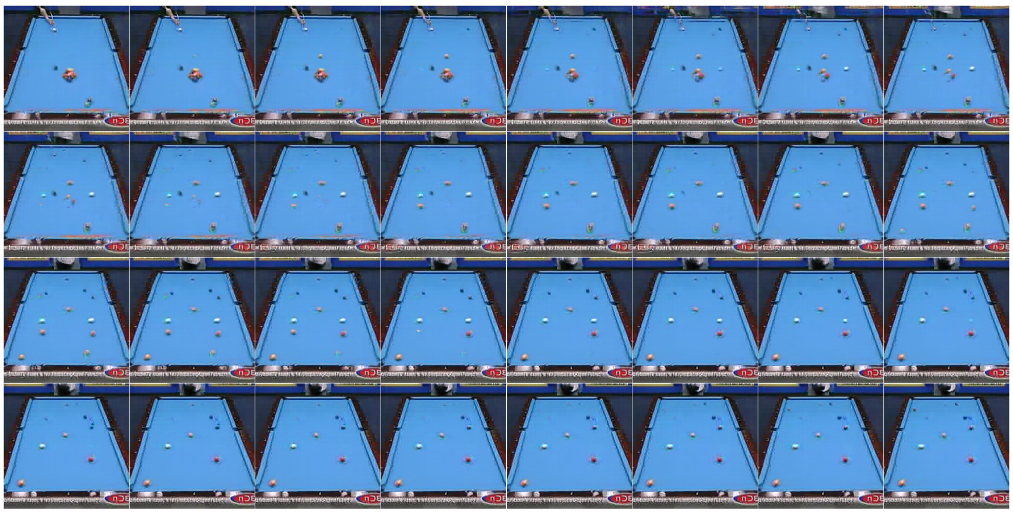


Figure 5.6.: An example of our generated action videos with a spatial resolution of $512x512px$. The sequence shows 32 consecutive frames of playing billiard, all frames with a spatial resolution of $512x512px$. Each frame is generated using latent space interpolation between the first and the last frame of the sequence.

MS-SSIM for UCF-101 class Billiard	
Method	MS-SSIM
PGAN $64x64px$	0.35
PGAN $128x128px$	0.33
PGAN $256x256px$	0.34
PGAN $512x512px$	0.31
Training data	0.31

Table 5.2.: MS-SSIM results for the class **Billiard**. The results of our trained generator at different levels of detail are similar to the result for the training data. This indicates that the diversity of generated samples at different scales is similar to the real data.

we trained our generator for only one class of the UCF-101 dataset, which means that we are not able to use the IS.
The MS-SSIM has a range between 0 and 1, where higher scores refer to perceptually more similar images. Therefore, a high number of similar images would lead to a high MS-SSIM score, which would indicate mode collapse.

We calculated the MS-SSIM between 10000 image pairs generated from our trained PGAN at different resolutions, and also calculated the score for the training data to be able to compare them. Table 5.2 shows the results for the class **Billiard**. The scores for our PGAN at different scales are similar to the score for the training data, which indicates that the diversity of our generated samples is similar to the training data. Therefore, we can rule out mode collapse.

5.3. Conclusion

Chapter 5 described our approach to further increase the spatial resolution of our action videos to $512x512px$. We were able to generate meaningful videos with increased image quality and spatial resolution, compared to our first approach. One key component of this GAN is the WGAN-GP [18] loss function, which has the benefit that the generator is less prone to mode collapse. By calculating the structural similarity, we additionally added

evidence that no mode collapse appeared.
The image quality of our generated images is very high, also containing readable text in the images. This leads to the question, if the network really learns the distribution of the training data, or only memorizes patterns. For a small dataset, memorizing would certainly be feasible due to the high number of parameters in the network. However experiments in Appendix A.5 show that the generator is also capable of generating images for multiple classes, but with decreased image quality. In my opinion, the high level of detail in the generated images is a result of the high number of parameters in the networks, which makes it possible to memorize patterns. However, if GANs are supposed generate huge amounts of reasonable data to augment datasets, it is necessary that the image quality is high, also if certain patterns repeat themselves in the images.
Furthermore, we want to point out that our video generation is only semi-automatic, because we have to generate a number of images and then manually select start and end frame, before we are able to interpolate in the latent space. Making this approach fully automatic is work which we aim to do in the future. An attempt to solve this problem would probably focus on conditioning the PGAN during training to force the generation of start and end frames.
PGAN is able to generate images for action recognition without the usage of a condition network, which is a clear advantage to our STGAN approach. One significant drawback of the PGAN is, that the computational costs are significantly higher compared to the STGAN.

Our experiments also show that a high resolution action recognition dataset would be required. This could significantly increase image and video quality of the generated samples. One possibility is to use state-of-the-art super resolution methods to upscale already existing action recognition datasets.

6. Discussion and Future Work

This thesis describes our approaches to generate videos for action recognition, using GANs. The work focused on two main points:
Our spatiotemporal GAN, proposed in Chapter 4, is able to generate videos in the domain of action recognition at a spatial resolution up to $227x227px$. These generated videos give additional insights about what is important for the condition network. By further analyzing this property, it may be possible find out if state-of-the-art models are over-parameterized, and if a simpler model could be used for classification.
However, if one wants to use this generator for data augmentation to improve the training for action recognition classifiers, our spatiotemporal GAN may not suit this task, because the overall video quality seems to be too low.

With the PGAN proposed in Chapter 5, on the other hand, we were able to generate meaningful high resolution videos. These generated videos may provide the quality to further improve state-of-the-art classifiers.
According to the video generation process using latent space interpolation, the most interesting work for the future would certainly be to investigate how to control the latent space. The fact that simple latent space interpolation led to action videos with temporal coherence shows that analyzing the latent space could lead to improved sample generation. It would be interesting to see if one can find explicit features in the latent space, which would enable us to condition the video generation.

Further information about the latent space would also impact our aim to make the video generation fully-automatic, without the need to manually select the first and the last frame for the video.

Another important point is the need for a high resolution action recognition dataset, because the generator is highly dependent on the training set.

For this, it may be possible to use state-of-the-art super resolution techniques to improve the quality of low resolution dataset like UCF-101.

Appendix A.

Additional Material

A.1. Simplification of the Inception Score

Simplification of the IS definition in Equation 3.14:

$$\ln(IS) = \mathbb{E}_{\mathbf{x}\sim p_{model}}[D_{KL}(p(y|\mathbf{x})||p(y))] \tag{A.1a}$$

$$= \sum_{\mathbf{x}} p_{model}(\mathbf{x})[D_{KL}(p(y|\mathbf{x})||p(y))] \tag{A.1b}$$

$$= \sum_{\mathbf{x}} p_{model}(\mathbf{x}) \sum_{i} p(y=i|\mathbf{x}) \ln \frac{p(y=i|\mathbf{x})}{p(y=i)} \tag{A.1c}$$

$$= \sum_{\mathbf{x}} \sum_{i} p(\mathbf{x}, y=i) \ln \frac{p(\mathbf{x}, y=i)}{p(\mathbf{x})p(y=i))} \tag{A.1d}$$

$$= \sum_{\mathbf{x}} \sum_{i} [p(\mathbf{x}, y=i) \ln(p(\mathbf{x}, y=i)) - p(\mathbf{x}, y=i) \ln(p(\mathbf{x})p(y=i))]. \tag{A.1e}$$

Equation A.1a is the definition of the IS. In Equation A.1b, we replaced the expectation $\mathbb{E}_{\mathbf{x}\sim p_{model}}$ with a summation over all samples, with $\mathbf{x}$ defining one sample. The definition of the KL-divergence is used in Equation A.1c with the summation index i defining one specific class of the dataset . In Equation A.1d, we use the definition of the conditional probability, which is defined as $P(y|x) = \frac{P(y,x)}{P(x)}$. The last step shown in Equation A.1e is to apply the rules of logarithmic calculation, which provides us with a computationally feasible way to calculate the IS.

We decided to take the exponential of the IS in our experiments, which makes it easier to interpret the score:

$$IS = \exp(\sum_{\mathbf{x}} \sum_{i} [p(\mathbf{x}, y = i) \ln(p(\mathbf{x}, y = i)) - p(\mathbf{x}, y = i) \ln(p(\mathbf{x}) p(y = i))]). \tag{A.2}$$

A.2. UCF-101 Image Generation

This section shows the experiments of generating images for action recognition using the 2D generator provided by [9] combined with our approach of using the LRCN as condition network. Figure A.1 and A.2 provide the qualitative results for all classes of the UCF-101 dataset.

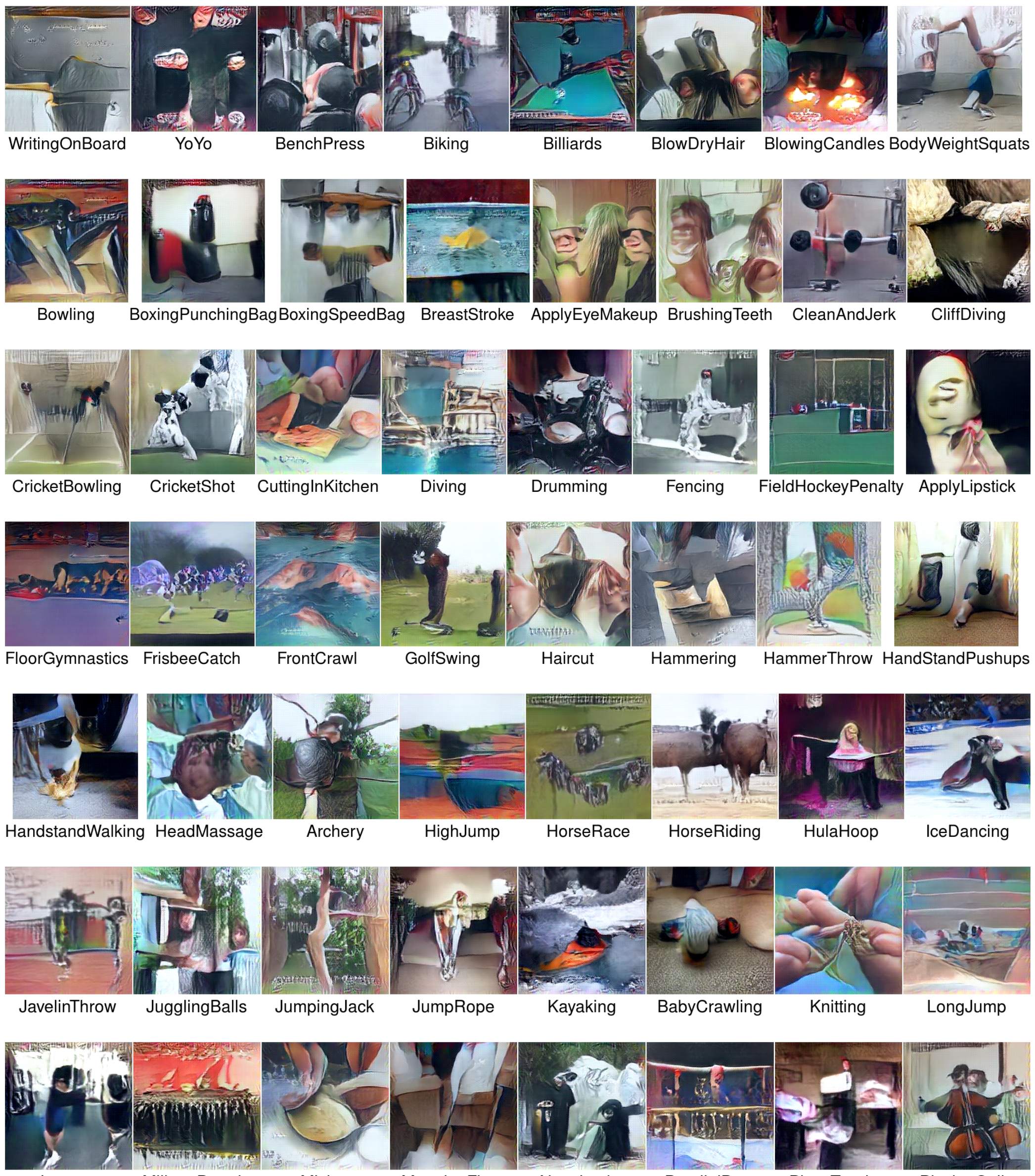


Figure A.1.: Image generation for action recognition. One can see generated images for 56 UCF-101 classes. The results are achieved by using a 2D GAN architecture pre-trained on ImageNet. We used the LRCN condition network to generate images in the context of action recognition.

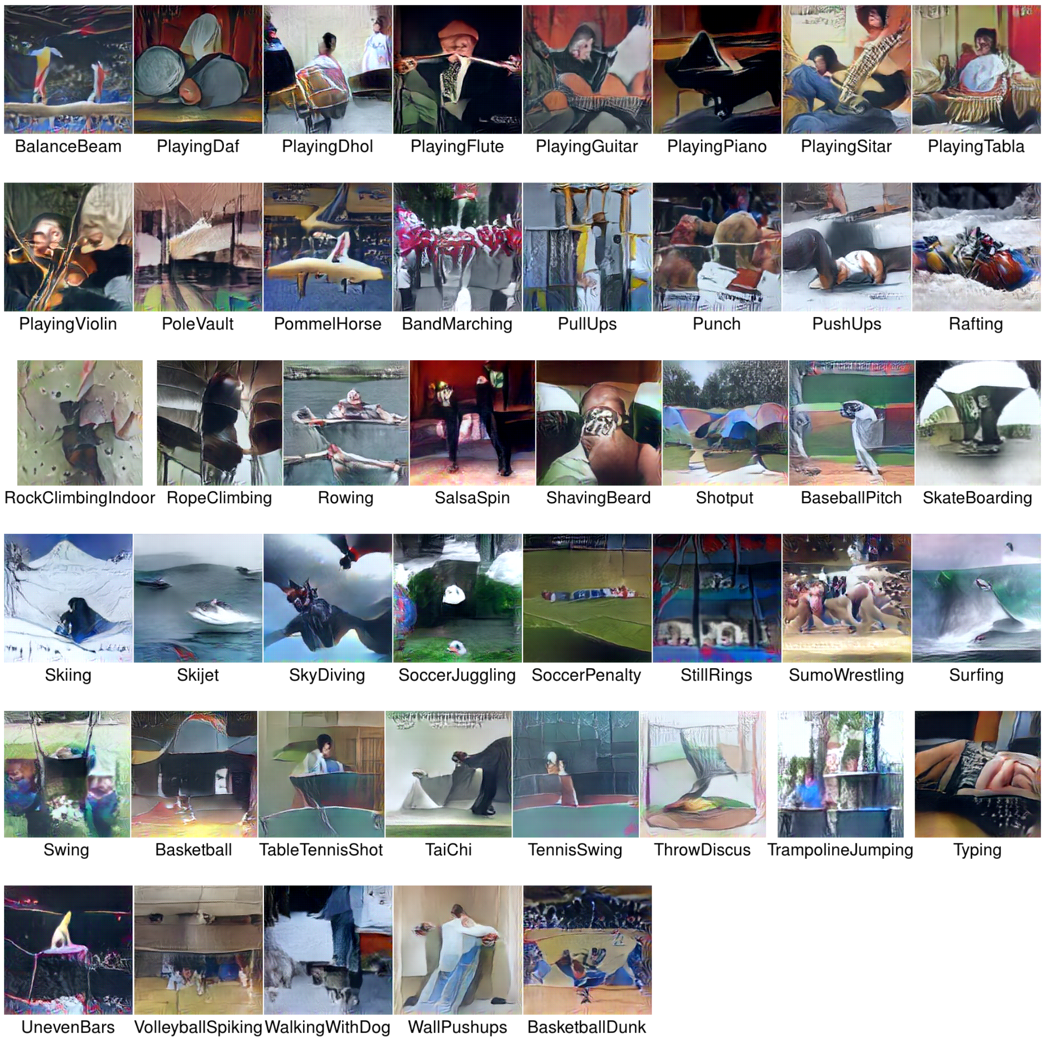


Figure A.2.: Image generation for action recognition cont'd. One can see generated images for 45 UCF-101 classes. The results are achieved by using a 2D GAN architecture pre-trained on ImageNet. We used the LRCN condition network to generate images in the context of action recognition.

A.3. Long-Sequence Video Generation

This section shows generated videos with a length of 48 frames. Using the LRCN as condition network enables us to generate videos up to 160 frames, but the computational costs also increase with the length of the sequence. Therefore, we were restricted to generate videos with a maximum length of 48 frames.

Figures A.3 and A.4 show results using our $STGAN_{ImNet}$ as generator. Figures A.5 and A.6 show results using $STGAN_{ucf}$, which is fine-tuned on the UCF-101 dataset.

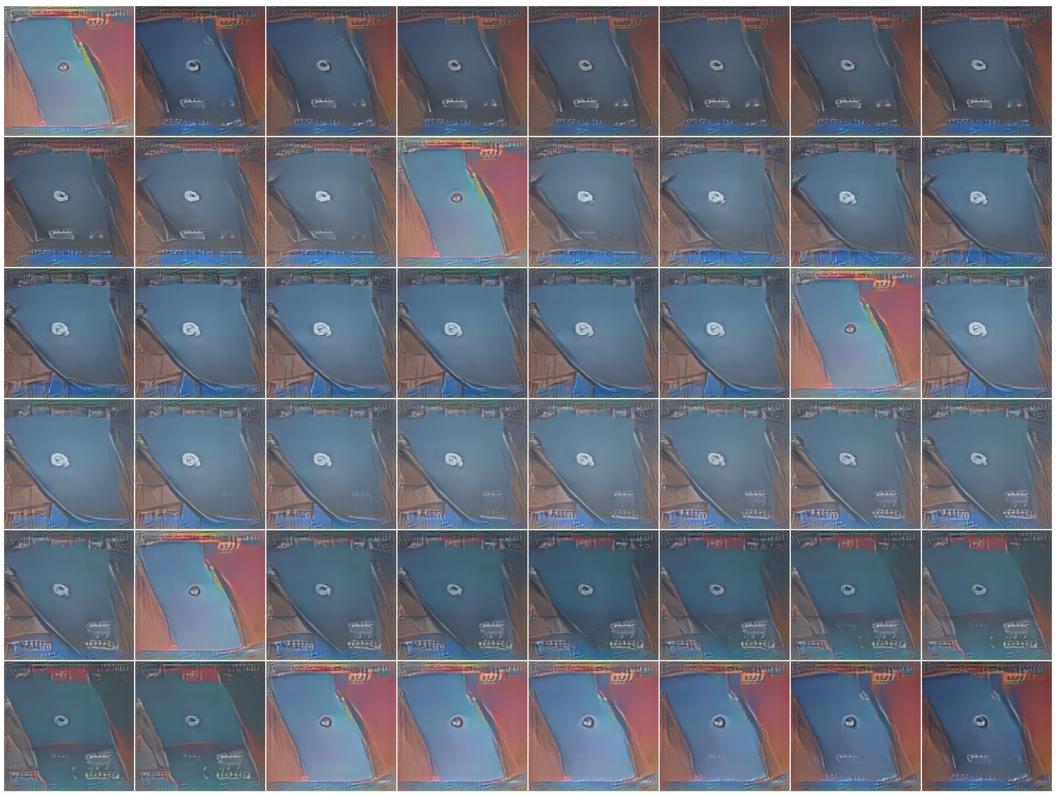


Figure A.3.: Video generation using the $STGAN_{ImNet}$ with the LRCN, using activation maximization for the class Billiards. The sequence contains 48 frames.

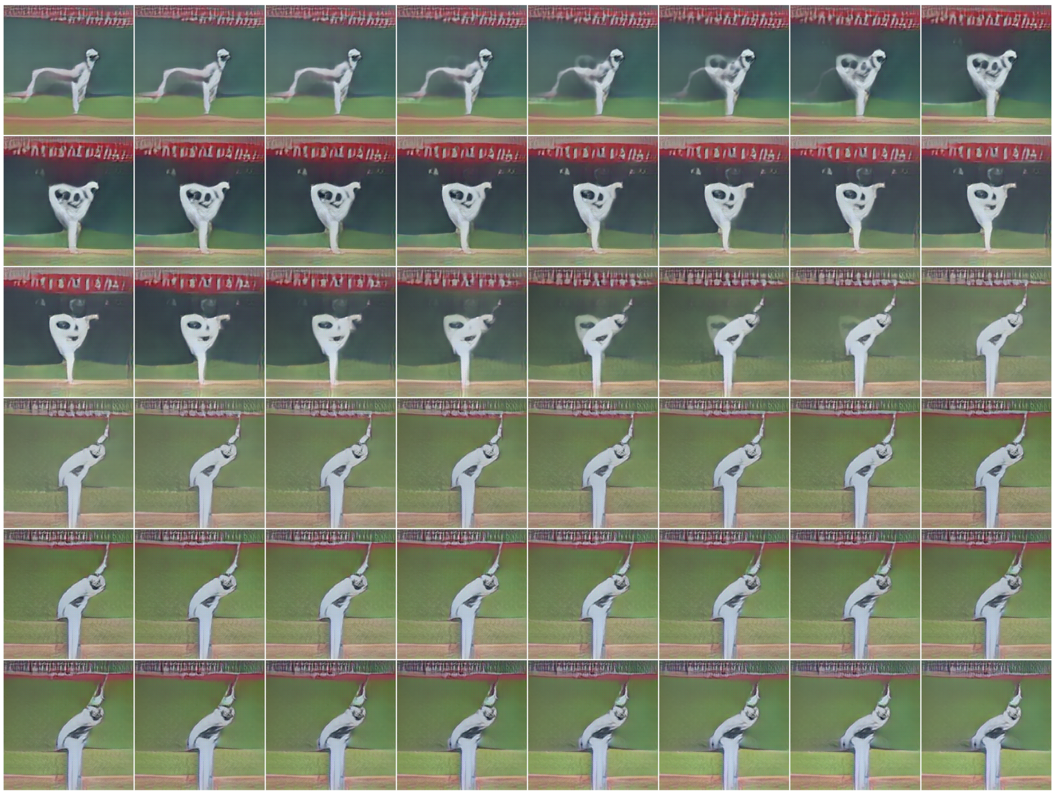


Figure A.4.: Video generation using the $STGAN_{ImNet}$ with the LRCN, using activation maximization for the class Baseball Pitch. The sequence contains 48 frames.

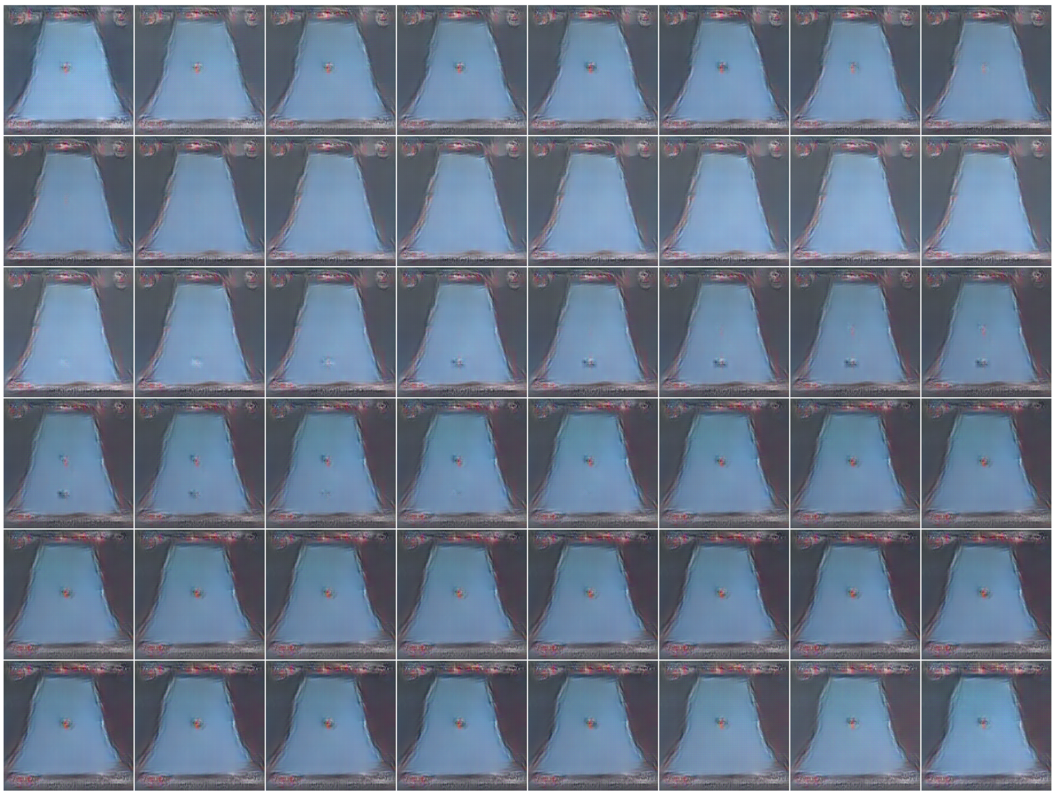


Figure A.5.: Video generation using the $STGAN_{ucf}$ with the LRCN, using activation maximization for the class Billiards. The sequence contains 48 frames.

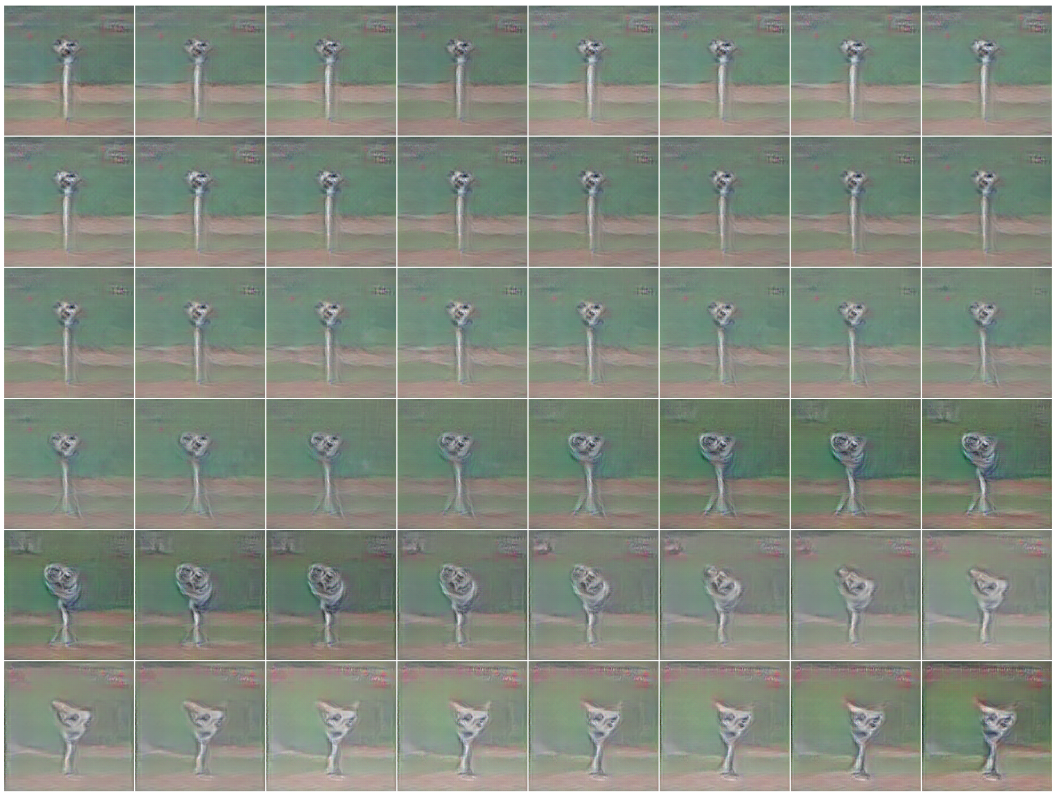


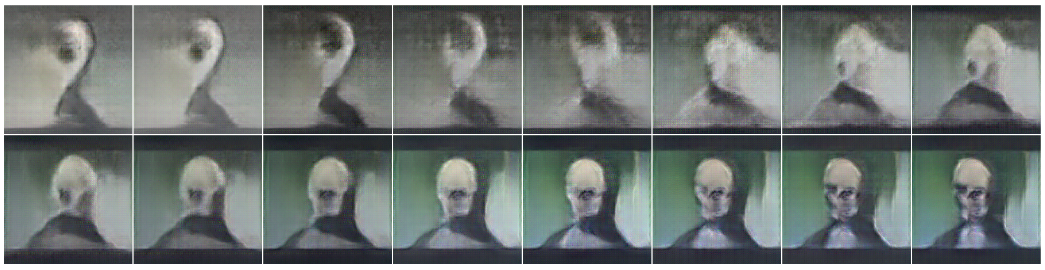
Figure A.6.: Video generation using the $STGAN_{ucf}$ with the LRCN, using activation maximization for the class Baseball Pitch. The sequence contains 48 frames.

A.4. Video Generation using C3D as Condition Network

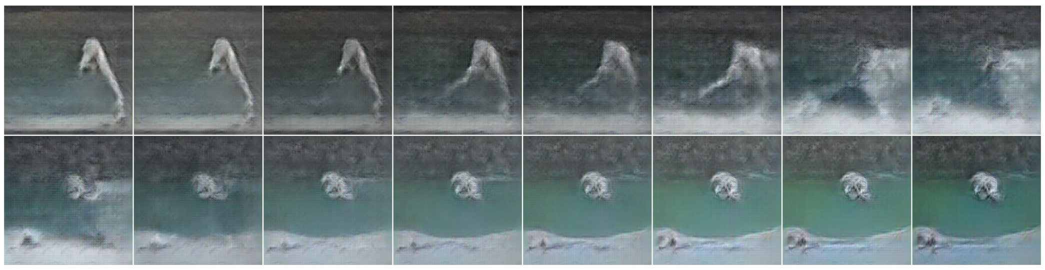
This section provides additional results using the C3D as condition network. We followed the same steps as described in Subsection 4.6.2 to build an end-to-end network consisting of our 3D generator and the C3D. We used the tensorflow [37] implementation of the C3D [21], which is pre-trained on Sports1M and UCF-101. However, we made one specific adaptation to use C3D in our experiments. We doubled the spatial stride in the first layer of the C3D, which enabled us to feed videos with a spatial resolution of $224x224px$ to the network. This simplifies the matching of the layer dimensions between the two networks.

The qualitative results are shown for the classes **Apply Lipstick**, **Baseball Pitch**, **Billiard** and **Clean and Jerk**. Figure A.7 shows results using $STGAN_{ucf}$ with the C3D as condition network. For comparison, Figure A.8 shows results using the pre-trained $STGAN_{ImNet}$.

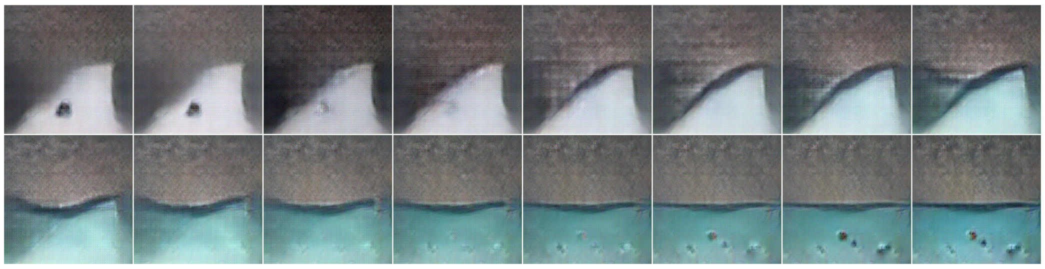
Note that the image quality using C3D as condition network is considerably lower than with LRCN. As discussed in Subsection 4.6.3, we believe that the image quality highly depends on the network structure of the condition network, and how this condition network is trained. Therefore, not every classifier is equally suitable as condition network.



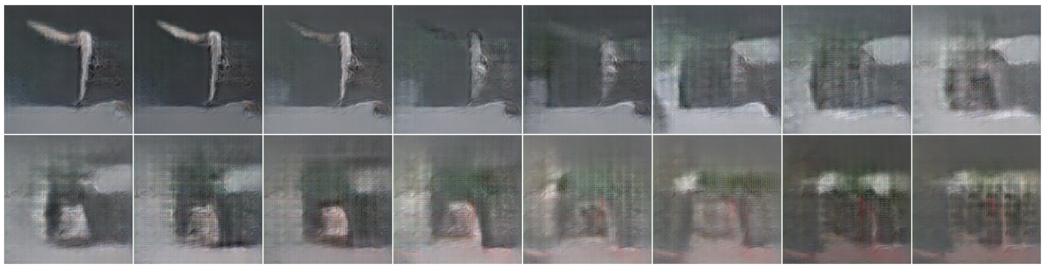
(a) Apply Lipstick



(b) Baseball Pitch

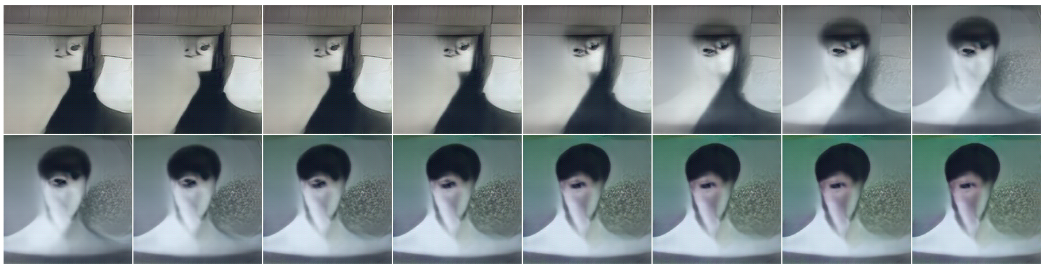


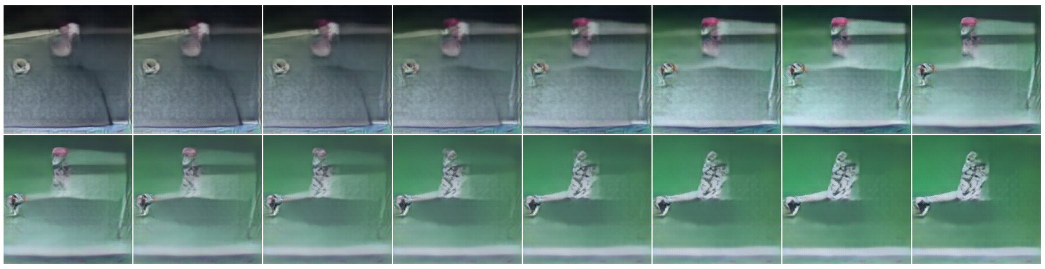
(c) Billiard

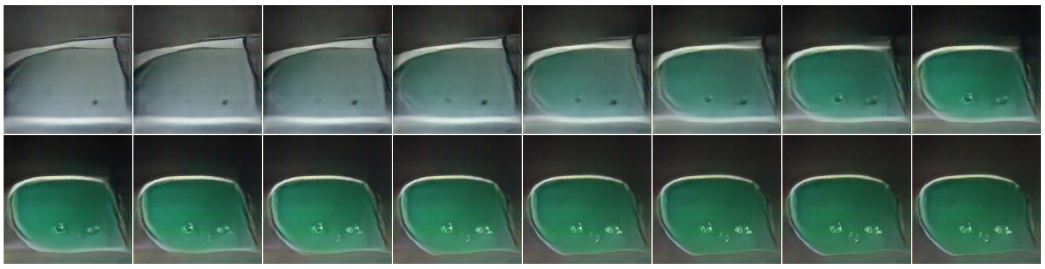


(d) Clean and Jerk

Figure A.7.: Visualization of generated videos using $STGAN_{ucf}$ and the C3D as condition network. Each video consists of 16 consecutive frames. Results are shown for the classes **Apply Lipstick**, **Baseball Pitch**, **Billiard** and **Clean and Jerk**.


(a) Apply Lipstick


(b) Baseball Pitch


(c) Billiard

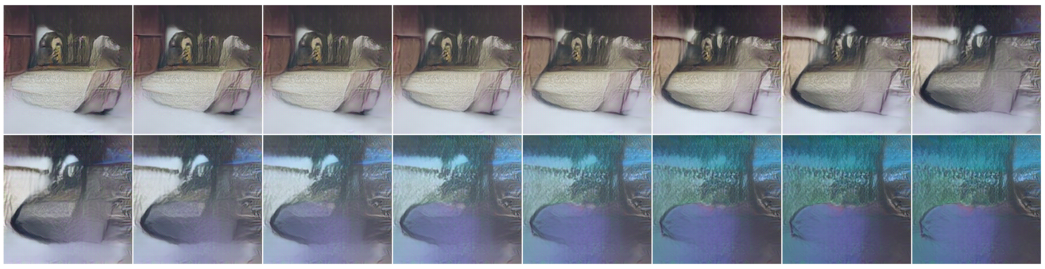

(d) Clean and Jerk

Figure A.8.: Visualization of generated videos using $STGAN_{ImNet}$ and the C3D as condition network. Each video consists of 16 consecutive frames. Results are shown for the classes **Apply Lipstick**, **Baseball Pitch**, **Billiard** and **Clean and Jerk**.

A.5. Multi Class Image Generation using Progressively Growing GAN

This section shows results of our approach to train the PGAN framework for multiple UCF-101 classes concurrently, compared to results for one-class training. Samples for training the framework only for the class **Billiard** are shown in Figure A.9. Figure A.10 shows examples of five different classes. Figure A.11 shows 16 classes, each of them from the subcategory **Body-Motion Only**. More details about the UCF-101 subcategories are available online[1]. Note that the generators for these experiments are trained up to a spatial resolution of $256x256px$ to speed up training. One can see that training for more than one class is possible, but the training time increases significantly (see Table A.1). This encourages the assumption that training PGAN for all UCF-101 classes at high resolution is possible, if enough computational power is available.

Classes / Category	# Classes	Training Duration
Billiard	1	23h 03m 52s
[Apply Eye Makeup, Typing, Skiing, Band Marching, Hammering]	5	2d 21h 58m
Body-Motion Only	16	5d 07h 50m

Table A.1.: Comparison of training duration for single and multi class training. One can see that a higher number of training classes leads to an increased training duration. Note that we did not use a specific stopping criterion for training, but rather trained until a reasonable image quality was achieved.

[1]http://crcv.ucf.edu/data/UCF101.php, last visited: Oct. 2018

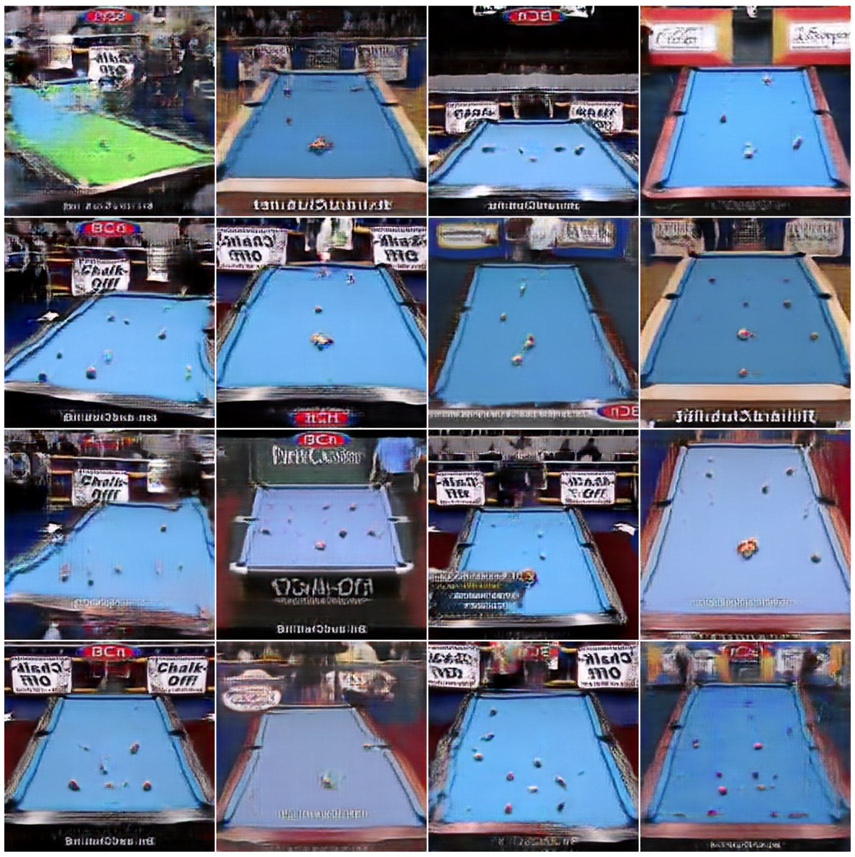


Figure A.9.: Results from PGAN trained for **Billiard**. The generator was trained to sample images at a spatial resolution of $256x256px$.

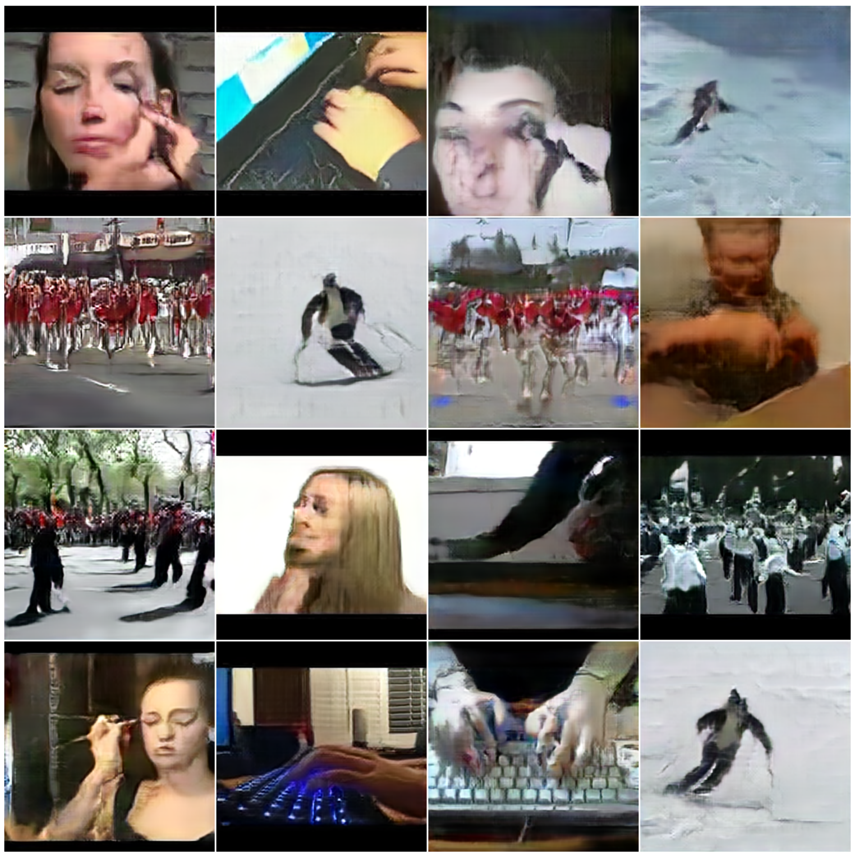


Figure A.10.: Results from PGAN trained for five classes. This generator was trained for the classes **Apply Eye Makeup**, **Typing**, **Skiing**, **Band Marching** and **Hammering**. The generator was trained to sample images at a spatial resolution of $256x256px$.

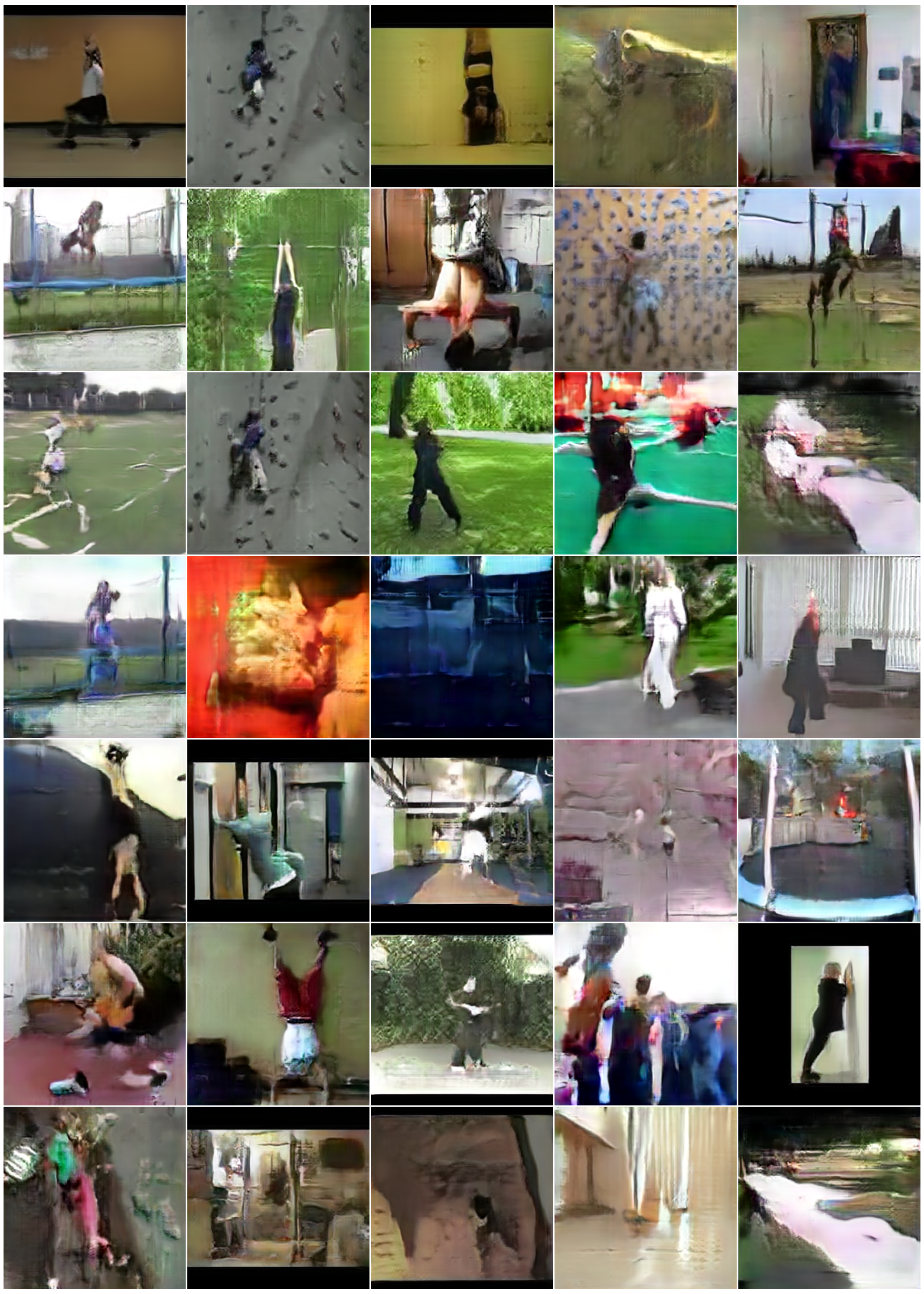


Figure A.11.: Results from PGAN trained for 16 classes. This generator was trained for the classes from the subcategory **Body-Motion Only** of the UCF-101 dataset. The generator was trained to sample images at a spatial resolution of $256x256px$.

A.6. Further Examples of Video Generation using Progressively Growing GAN

We trained our PGAN for several classes in the UCF-101 dataset (but only for one class at a time), and tried to generate videos using latent space interpolation. Figure A.12 shows a generated video for the class **Clean and Jerk**, Figure A.13 shows a video for the class **Archery**. Both examples show image sequences with clearly visible temporal coherence. Note that the networks in this section were trained to generate images at a resolution of $256x256px$, to speed up training.

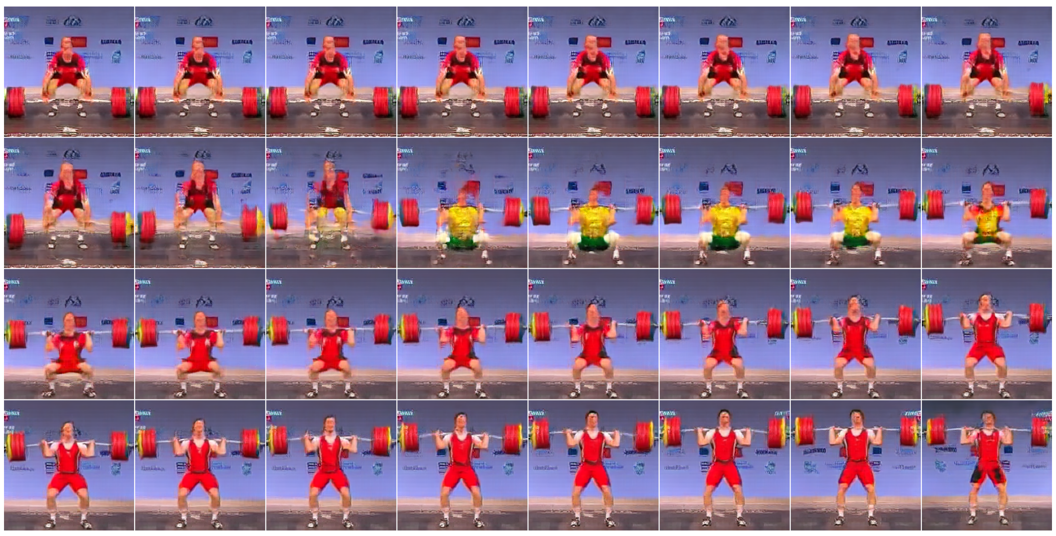


Figure A.12.: Generated video for the class **Clean and Jerk** using PGAN. Each frame has a spatial resolution of $256x256px$. By analyzing the image sequence, it is clear to see that the clothes of the person change. However, the action itself, which is lifting the weights, is clearly showing a temporal coherence.

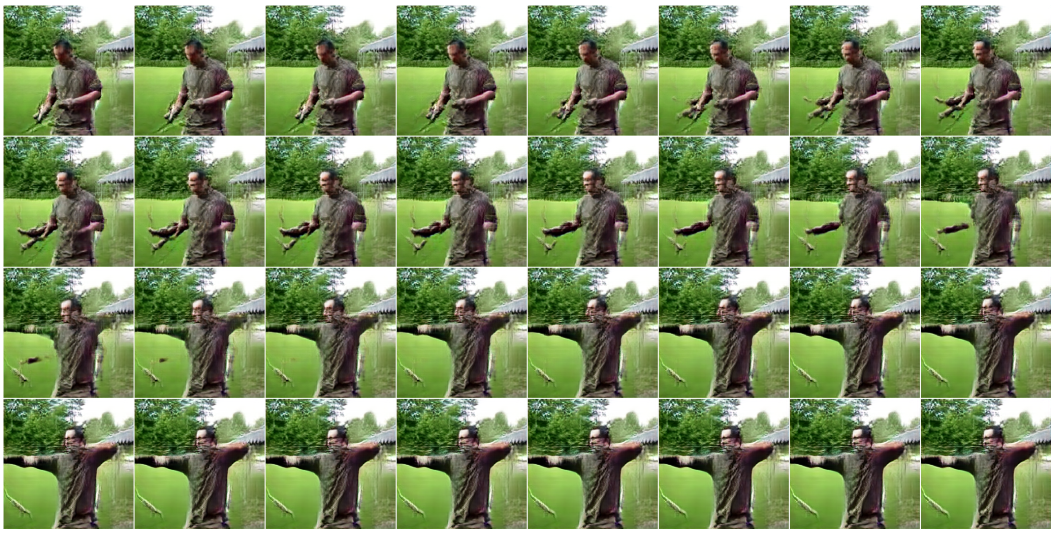


Figure A.13.: Generated video for the class **Archery** using PGAN. Each frame has a spatial resolution of $256x256px$.

Bibliography

[1] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. *Youtube-8m: A large-scale video classification benchmark*. Tech. rep. Google Research, 2016. [arXiv preprint arXiv:1609.08675].

[2] Martin Arjovsky and Léon Bottou. "Towards principled methods for training generative adversarial networks." In: *International Conference on Learning Representations (ICLR)*. 2017.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein Generative Adversarial Networks." In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. 2017, pp. 214–223.

[4] Shane Barratt and Rishi Sharma. "A Note on the Inception Score." In: *arXiv preprint arXiv:1801.01973* (2018).

[5] David Berthelot, Thomas Schumm, and Luke Metz. "BEGAN: boundary equilibrium generative adversarial networks." In: *arXiv preprint arXiv:1703.10717* (2017).

[6] Qifeng Chen and Vladlen Koltun. "Photographic image synthesis with cascaded refinement networks." In: *The IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 1520–1529.

[7] Emily L Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. "Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks." In: *Advances in Neural Information Processing Systems (NIPS)*. 2015, pp. 1486–1494.

[8] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. "Long-term recurrent convolutional networks for visual recognition and description." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 2625–2634.

[9] Alexey Dosovitskiy and Thomas Brox. "Generating images with perceptual similarity metrics based on deep networks." In: *Advances in Neural Information Processing Systems (NIPS)*. 2016, pp. 658–666.

[10] DC Dowson and BV Landau. "The Fréchet distance between multivariate normal distributions." In: *Journal of Multivariate Analysis* 12.3 (1982), pp. 450–455.

[11] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. *Visualizing higher-layer features of a deep network*. Tech. rep. University of Montreal, 2009.

[12] Christoph Feichtenhofer, Axel Pinz, and Richard Wildes. "Spatiotemporal residual networks for video action recognition." In: *Advances in Neural Information Processing Systems (NIPS)*. 2016, pp. 3468–3476.

[13] Li Fei-Fei, Justin Johnson, and Serena Yeung. *Lecture 11: Detection and Segmentation*. `http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf`. last visited: Oct 2018. URL: `http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf`.

[14] Horst Fuchs. "Visualizing and Understanding Deep Driving Models." Master's Thesis at Institute of Electrical Measurement and Measurement Signal Processing, Graz University of Technology. 2017.

[15] Ian Goodfellow. "NIPS 2016 tutorial: Generative adversarial networks." In: *arXiv preprint arXiv:1701.00160* (2016).

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." In: *Advances in Neural Information Processing Systems (NIPS)*. 2014, pp. 2672–2680.

[18] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. "Improved training of Wasserstein GANs." In: *Advances in Neural Information Processing Systems (NIPS)*. 2017, pp. 5767–5777.

[19] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. “GANs trained by a two time-scale update rule converge to a local nash equilibrium.” In: *Advances in Neural Information Processing Systems (NIPS)*. 2017, pp. 6626–6637.

[20] Sepp Hochreiter and Juergen Schmidhuber. “Long short-term memory.” In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[21] HouXin. *C3D-tensorflow*. `https://github.com/hx173149/C3D-tensorflow`. last visited: Sep 2018. 2018.

[22] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. “Globally and Locally Consistent Image Completion.” In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), 107:1–107:14.

[23] David Jacobs. “Correlation and convolution.” In: *Class Notes for CMSC 426* (2005). last visited: Oct. 2018. URL: `http://www.cs.umd.edu/~djacobs/CMSC426/Convolution.pdf`.

[24] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. “Caffe: Convolutional Architecture for Fast Feature Embedding.” In: *arXiv:1408.5093* (2014).

[25] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. “Large-scale video classification with convolutional neural networks.” In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 1725–1732.

[26] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. “Progressive Growing of GANs for Improved Quality, Stability, and Variation.” In: *International Conference on Learning Representations (ICLR)*. 2018.

[27] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. “The kinetics human action video dataset.” In: *arXiv preprint arXiv:1705.06950* (2017).

[28] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: *International Conference on Learning Representations (ICLR)*. 2015.

[29] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes.” In: *International Conference on Learning Representations (ICLR)*. 2014.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. “Imagenet classification with deep convolutional neural networks.” In: *Advances in Neural Information Processing Systems (NIPS)*. 2012, pp. 1097–1105.

[31] Solomon Kullback and Richard A Leibler. “On information and sufficiency.” In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning.” In: *Nature* 521.7553 (2015), pp. 436–444.

[33] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition.” In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[34] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew P Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network.” In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 105–114.

[35] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations.” In: *Proceedings of the 26th International Conference on Machine Learning (ICML)*. 2009, pp. 609–616.

[36] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. “Unsupervised image-to-image translation networks.” In: *Advances in Neural Information Processing Systems (NIPS)*. 2017, pp. 700–708.

[37] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[38] Anh Nguyen, Jeff Clune, Yoshua Bengio, Alexey Dosovitskiy, and Jason Yosinski. “Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space.” In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 3510–3520.

[39] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks." In: *Advances in Neural Information Processing Systems (NIPS)*. 2016, pp. 3387–3395.

[40] Augustus Odena, Christopher Olah, and Jonathon Shlens. "Conditional Image Synthesis with Auxiliary Classifier GANs." In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. 2017, pp. 2642–2651.

[41] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." In: *International Conference on Learning Representations (ICLR)*. 2016.

[42] David E Rumelhart, Geoffrey Hinton, and Ronald J Williams. "Learning representations by back-propagating errors." In: *Nature* 323.6088 (1986), pp. 533–536.

[43] Masaki Saito, Eiichi Matsumoto, and Shunta Saito. "Temporal generative adversarial nets with singular value clipping." In: *The IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2849–2858.

[44] Ruslan Salakhutdinov and Hugo Larochelle. "Efficient learning of deep Boltzmann machines." In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2010, pp. 693–700.

[45] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training GANs." In: *Advances in Neural Information Processing Systems (NIPS)*. 2016, pp. 2234–2242.

[46] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." In: *International Conference on Learning Representations (ICLR)* (2014).

[47] Skymind.ai. *A Beginner's Guide to Generative Adversarial Networks (GANs)*. `https://skymind.ai/wiki/generative-adversarial-network-gan/`. last visited: Sep 2018. URL: `https://skymind.ai/wiki/generative-adversarial-network-gan`.

[48] Casper Kaae Soenderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. “Amortised map inference for image super-resolution.” In: *International Conference on Learning Representations (ICLR)*. 2017.

[49] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. *UCF101: A dataset of 101 human actions classes from videos in the wild*. Tech. rep. Center for Research in Computer Vision (CRCV), Nov. 2012. [arXiv preprint arXiv:1212.0402].

[50] Concetto Spampinato, Sergio Palazzo, P D’Oro, Francesca Murabito, Daniela Giordano, and M Shah. “VOS-GAN: Adversarial Learning of Visual-Temporal Dynamics for Unsupervised Dense Prediction in Videos.” In: *arXiv preprint arXiv:1803.09092* (2018).

[51] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting.” In: *The Journal of Machine Learning Research (JMLR)* 15 (2014), pp. 1929–1958.

[52] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. “Rethinking the inception architecture for computer vision.” In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2818–2826.

[53] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. “Intriguing properties of neural networks.” In: *International Conference on Learning Representations (ICLR)*. 2014.

[54] Lucas Theis, Aäron van den Oord, and Matthias Bethge. “A note on the evaluation of generative models.” In: *International Conference on Learning Representations (ICLR)*. 2015.

[55] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. “Learning spatiotemporal features with 3D convolutional networks.” In: *The IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 4489–4497.

[56] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. “Extracting and composing robust features with denoising autoencoders.” In: *Proceedings of the 25th International Conference on Machine Learning (ICML)*. 2008, pp. 1096–1103.

[57] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. “Generating videos with scene dynamics.” In: *Advances in Neural Information Processing Systems (NIPS)*. 2016, pp. 613–621.

[58] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. “Video-to-Video Synthesis.” In: *Advances in Neural Information Processing Systems (NIPS)*. 2018.

[59] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. “High-Resolution Image Synthesis and Semantic Manipulation With Conditional GANs.” In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[60] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. “Non-local Neural Networks.” In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).

[61] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. “Image quality assessment: from error visibility to structural similarity.” In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612.

[62] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. “Multiscale structural similarity for image quality assessment.” In: *Asilomar Conference on Signals, Systems & Computers*. Vol. 2. 2003, pp. 1398–1402.

[63] Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger Grosse. “On the quantitative analysis of decoder-based generative models.” In: *International Conference on Learning Representations (ICLR)*. 2016.

[64] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. “How transferable are features in deep neural networks?” In: *Advances in Neural Information Processing Systems (NIPS)*. 2014, pp. 3320–3328.

[65] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. “Understanding Neural Networks Through Deep Visualization.” In: *Deep Learning Workshop, International Conference on Machine Learning (ICML)*. 2015.

[66] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks." In: *European Conference on Computer Vision (ECCV)*. 2014, pp. 818–833.

[67] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. "Deconvolutional networks." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2010, pp. 2528–2535.

[68] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. "Adaptive deconvolutional networks for mid and high level feature learning." In: *The IEEE International Conference on Computer Vision (ICCV)*. 2011, pp. 2018–2025.