Tanja Linkermann, BSc

Motion Prediction with the Improved Wasserstein GAN

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisors

Univ.-Prof. Dipl-Ing. Dr.techn. Thomas Pock,
Dipl.-Inform. Dr.sc.ETH Christoph Vogel,
Dipl.-Ing. Dipl.-Ing.(FH) Markus Hofinger

Institute of Computer Graphics and Vision

Graz, April 2019

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

Acknowledgements

Being part of the Master program of Computer Science at TU Graz was a great opportunity to expand my knowledge in the fields of Computer Vision and Computational Intelligence. My education finally coming to a successful end means I found a lot of new friends, discovered different perspectives, and overcame some challenges – I chose to not always take the easiest route.

First of all I would like to express my appreciation for my supervisors Markus Hofinger, Christoph Vogel and Thomas Pock. Your support, guidance, and suggestions in the various meetings contributed a lot to this thesis, and I could always count on your help.

Secondly, I am very grateful to my parents who always supported me with the idea of studying abroad and continuing my education. Without your support it would not have been possible!
On that score, a thank you also goes to the professors at TU Delft that made me write various reports and papers, so I was not afraid of writing down this thesis at all.

Thirdly, a special thank you to Ceca for all the support and the hours spend at Inffeld together, I do not know how I could have survived this year without you.
Also thanks to my friends Michaela, Kristina and Tanja who were always there for me when I needed them. Thank you Michi for spending your vacation with reading this thesis!

A last thanks to all the people from BEST Graz for keeping me busy and thus keeping my motivation up.
Finally, a word of thanks to the ones not mentioned in this part who indirectly also supported me.

Tanja Linkermann
Graz, 8th April 2019

Abstract

Frame prediction is a well studied problem in Computer Vision, but no satisfactory results are given so far, and the road to unsupervised video representation learning is still far. Problems are, for instance, the generation of unrealistic scenes, the computational excess of computing pixel trajectories and the poor visual quality of resulting images.

With recently emerged models like the Variational Autoencoder (VAE) and the Generative Adversarial Network (GAN) the generated images look promising, and training them became more stable since the Wasserstein GAN (WGAN) was introduced. We present a new VAE-GAN mixture model, named EncGAN, that is able to predict a future frame of a video sequence when given a sequence of past frames. To take the temporal component of the past frames into account, a convTime network layer is used. The improved WGAN loss function is used to learn the networks in an adversarial manner. The model extends a normal WGAN to a larger network that comprises an Encoder, which in turn includes the aforementioned time convolutions.

Experiments were conducted on the MNIST and moving MNIST datasets. The network is applicable to different datasets when adapted to the image size. The results show that the mixture model outperforms the improved WGAN and is able to learn to generate sharp images.

Code is available at `https://github.com/tanlinc/Motionganvae`.

Contents

List of Figures

1 Introduction

Imagine you have a robot arm and would like to program it to grasp an arbitrarily placed item. Or you are working with autonomous cars and should decide which direction to steer. What is common between the two is that they rely on Computer Vision. Having one or more cameras mounted on them, the robot and the car both produce image sequences that can be used for the tasks. But for the decision which action should come next, we do not only need to know the past action, but also the possible future motions. Thus, knowing what happens next in a scene, based on a sequence of images, is the fundamental question. However, the quality of a predicted future frame is crucial. If it is blurry or unrealistic, its reliabilty and usage for applications are limited. Being able to predict a sharp and realistic future frame for a given video would enable us to synthesize a short movie from a few initial frames, e.g., to show ideal movements of a sport action, restore videos with missing frames, or compute super slow motion of an existing video. A problem – but also a great chance – is that there are multiple plausible future frames given some past frames, so judging the realism may be difficult. On the other hand, models that are able to generate different possible future frames may reveal interesting information about how ambiguous a detected motion is.

By *predicting motion* it is possible to decide on future actions. But what exactly is meant by *motion*, and how can it be described? Different ways of describing a motion pattern are common in computer vision, depending on the discipline and the use context. For example, it could be represented as an optical flow field (given an image pair), as a direction vector (e.g., of the camera movement), as a label of an action that is performed in a given image or video, or as a sequence of (future) frames.
Thus, specifying a motion can mean to classify into labels, e.g., Skiing, Swimming or Flying. In this way it is called *action classification*. A motion

can also be represented by an optical flow field. An example flow field is depicted in figure 1.1. Usually the flow field between two consecutive frames is estimated (*flow estimation*), but it is also possible to predict future flow fields [23]. Predicting motion can also mean the generation of future frames. In this thesis this *frame prediction* will be investigated. The camera motion is not separately modeled but rather implicitly included in the objects moving in the scene by the future frame predictor.

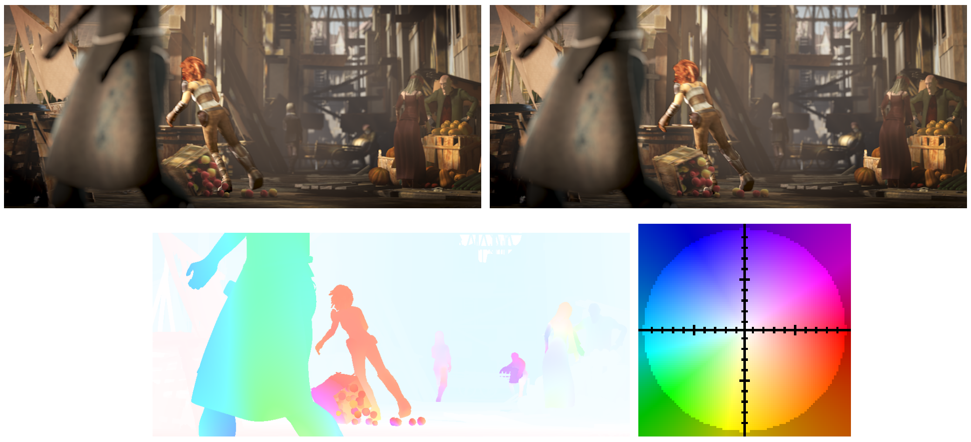


Figure 1.1: An example of Optical Flow on a scene from the MPI Sintel dataset, category *Market* 2. The two frames in the first row, the Optical Flow between them in the bottom left corner, visualized by the Middlebury Flow Color Code, which is representing different motion directions by different colors, code key in bottom right corner.

Predicting the future is however not unimodal. Usually more than one possible outcome is plausible. As you can see in figure 1.2 a frame can have multiple plausible future frames. The same is valid for a sequence of frames – more frames may restrict the possibilities of plausible future frames, but there is never only one realistic future frame.

This motion ambiguity can be tackled, e.g., by not predicting future frames directly, but instead by sampling the frames from an estimated probability distribution of the future frames. This approach is used by recently emerged models like Variational Autoencoder (VAE) and Generative Adversarial

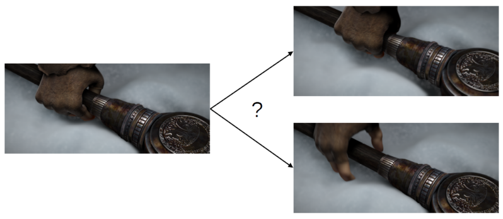


Figure 1.2: Future frame prediction is not unimodal. Given one frame (left), there are multiple plausible future frames and movements (right). Images are taken from MPI Sintel dataset, category *Ambush 7*

Network (GAN), which will be explained in more detail in section 2.2.

Current research in Computer Vision already uses these models for motion prediction, e.g., [38] uses a conditional VAE to predict future frames up to one second with only one past frame as input. One frame can of course not reveal the direction of the past movement, so sampling preconditioned on a given input frame can lead to movement in different directions, which again shows the multimodality. This thesis will experiment with more than one frame as input in order to make the model infer a future frame matching to a sequence of past frames. Early GANs had problems with mode collapse, meaning that they would only output samples from few modes of the probability distribution and disregard other modes, as can be seen in figure 1.3.

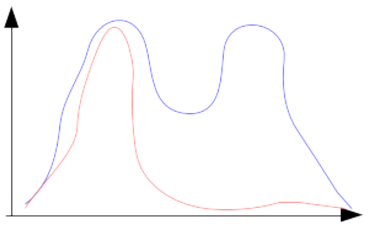


Figure 1.3: Mode collapse problem in 2D. The desired distribution p_{data} in blue has two modes, but the learned distribution p_{model} in red is only covering one mode, disregarding the other peak.

This mode collapse is claimed to be solved with the improved Wasserstein

GANs [2, 12], which makes GANs the better choice for image generation, since they showed better image quality in terms of sharpness from the start. Therefore the GAN model was chosen to be the basis to build on for this thesis, with the goal to develop a more realistic and good quality future frame prediction. In their paper *Dual motion GAN for future-flow embedded video prediction*, Liang, Lee, Dai and Xing [23] are describing how they predict frames and flow, using warped flow to improve the frame predicition as well as estimating the flow from the predicted frame to enhance the flow prediction, thus calling it a *dual* GAN. Their architecture is trained end-to-end on large images (256x256 pixels), which is hard to achieve in a GAN setting. At the time of writing no source code was publically available.

In this thesis we will implement a novel model combining elements of VAEs and GANs to get the benefits of both concepts. Through adding a temporal layer it will be possible to process several consecutive input frames simultaneously. We will investigate if these steps help to generate plausible future frames. For the experiments our developed model was trained on certain datasets, namely MNIST and moving MNIST, but it can be trained on any dataset with a sufficient number of images. Therefore any domain of movements, also real-world scenes, become predictable. With the help of experiments VAE and GAN will be compared, and it will be investigated if by combining both models better images can be generated. Moreover, a frame prediction will be built by adding conditional input, and the benefit of a sequence of frames as input will be researched.

2 Basics

A digital image consists of pixels, where each pixel captures the light intensities observed from a point in a scence. The common RGB format uses three channels, i.e., three intensity values, one value for red, one for green and one for blue, whereas grayscale images only have one intensity channel. To analyze videos which consist of several frames, the changes from frame to frame in every pixel can be evaluated. However, not only the intensity of single pixels is important, but it also helps to look for features. These can be edges or other regions having a big contrast in at least one direction to their neighboring pixels and being rotation invariant. Applications like gesture recognition, object tracking and motion estimation were made possible with the help of manually created features. The so called *Neural networks* define features implicitly, since the architecture does not restrict which feature should be searched in the images. The networks thus learn their own filters, which can be visualized by plotting the learned weights.

In the following part, neural networks are presented. In addition, generative networks – special architectures of neural networks that can generate new images – are introduced. Especially the used concepts VAE and GAN are explained in detail, since they are the basis for the newly developed model. Afterwards, different datasets for machine learning on images are presented shortly, to explain why the certain datasets were chosen for our experiments. Furthermore, the evaluation of the experiments is explained and assessed for trustworthiness. For that, we take a closer look at image metrics and their advantages and drawbacks. In the end, a short overview of how to process a sequence of past frames is given, where the used time convolution method is presented.

2.1 Neural networks

Neural networks are a model trying to imitate how a human brain works. However, the complex biological model is only adapted in a very simplified way. In the end, neural networks are a model able to approximate functions. Starting from a single neuron, we will proceed to see how to build powerful *deep* networks with the help of simple mathematics. We will take a look at the training of these networks, the so called *backpropagation*, and the algorithms to train them.

2.1.1 General structure

The sections 2.1.1 and 2.1.3 are based on the book [31]. The nervous system in the brain is composed of neurons that receive signals and produce a response. To build artificial neurons, only a few parts are adopted from the biological model: input channels (*dendrites*), a cell body, and an output channel (*axon*).

Signals coming from different input channels are collected in a cell. Each signal is an all-or-none event but the number of input channels that need to be triggered for an output signal to happen is different. Sometimes a single input channel can push a cell to fire, but other input channels can achieve this only by simultaneously exciting the cell.
To imitate this, we associate a weight w_i with each input channel i $(1 \leq i \leq n)$.
In the brain, if all n input channels are activated at the same time, the signal which will be received is an additive one: $w_1 + w_2 + \ldots + w_n$. If this value is greater than the cell's threshold, the cell will fire a pulse.

The general structure of an artificial neuron is shown in figure 2.1. It can be seen that each neuron collects the information from n input channels with an *integration function* $g : \mathbb{R}^n \mapsto \mathbb{R}$. Usually the integration function is the sum of the inputs, i.e., the incoming real-valued x_i is multiplied by the corresponding real-valued weight w_i and everything is summed up. The output value of that unit is then determined using an *activation function* $f : \mathbb{R} \mapsto \mathbb{R}$. The activation function compares the sum with a threshold –

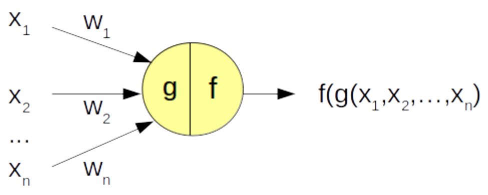


Figure 2.1: Abstract neuron with n inputs. Integration function g is usually addition, activation function f determines the output value.

usually it is chosen such that it can produce all values between 0 and 1, e.g., the sigmoid function $\sigma(x)$.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

If we conceive each node in an artificial neural network (NN) as a primitive function capable of transforming its input to an output, then NNs are nothing but networks of primitive functions. In fact, they represent a chain of function compositions which transform an input to an output vector – a *network function* Φ which is evaluated at the point $(x_1, x_2, \ldots, x_n)$.
Assuming g is the usual summation, the network function for the simple network consisting of one neuron in figure 2.1 can be computed as follows:

$$\Phi(x_1, x_2, \ldots, x_n) = f(g(x_1, x_2, \ldots, x_n)) = f(w_1x_1 + w_2x_2 + \ldots + w_nx_n) \tag{2.2}$$

The learning problem consists of finding the optimal combination of weights so that Φ approximates the target function ϕ as closely as possible. However, the target function ϕ may be complex and is not provided explicitly to the network, but only implicitly through some known datapoints. Thus, the network must learn to interpolate the input-output mapping pairs that can be found in the training set in order to be capable of evaluating new points that are not in the training set.

The models of artificial neural networks differ mainly in the primitive functions used, the topology of the graph and the learning algorithm that is used.

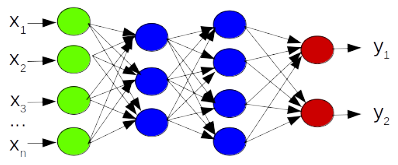


Figure 2.2: Layered architecture with n inputs. The green neurons are the input layer, the blue ones build the hidden layers, and the output layer consists of the red units.

The simplest type of connecting neurons is to build a directed graph, called a *feed-forward network*. It comprises successive *layers* of neurons and is not allowed to contain cycles.

Layered architectures have a set N of neurons which can be divided into subsets building the layers. So $N_1, N_2, \ldots, N_k$ are the layers, and neurons in N_i only connect to neurons in N_{i+1} for $1 \leq i \leq k-1$. The input is only connected to the neurons in the *input layer* N_1, and the neurons of the *output layer* N_k are the only ones connected to the output. Any other layers with no direct connections from or to the input or output are called *hidden layers*. Such a general layered network is depicted in figure 2.2. As also visible in the figure, in layered architectures usually all neurons from one layer are connected to all neurons in the following layer, which is called a *fully connected layer*.

So to build a layered neural network, we can decide on the structure in regard to the number of layers, the number of neurons in these layers and the activation functions of these neurons.

2.1.2 Convolutional neural networks

Convolutional Neural Networks (CNNs), a specific kind of neural networks, are one of the wide-spread architectures nowadays. They were already used for recognition of handwriting in 1996 by LeCun [21]. CNNs consist of *convolutional layers*, working similar to the convolution done in image processing. In these layers a small filter kernel is moved over the whole input and for the current window patch the filter value is computed - one neuron is needed for each patch. The computation is still efficient, because the weights are shared – the whole layer has the same weights for the filter kernel. Therefore, the weights that need to be learned are not scaling up with the resolution of the input.

There can be several convolutional layers after each other, which is then often called a *deep CNN*. By stacking the CNNs, in each layer larger and more descriptive features can be extracted, e.g., first only edges, but later whole faces. This happens because each layer is already getting a more descriptive input than the previous one, allowing it to extract even more descriptive information. Furthermore, the deeper we go into the network, the more of the input image will be reached by the neuron, since the neurons are connected locally. Neighboring image patches are neighboring neurons which will be both used for the next layer input. This creates a local connectivity, similar to receptive fields in the human eye.

The most common activation function in CNNs is a rectified linear unit (*ReLU*), which maps all negative input to zero, but keeps positive input as it is.

$$ReLU(x) = max\{0, x\} \tag{2.3}$$

Another version of it is the *leaky ReLU*, multiplying the input by a small real constant α if the neuron outputs a value smaller than 0. That is how it ensures to have a small positive gradient even when the neuron is not active.

$$leaky\,ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{2.4}$$

2.1.3 Training of neural networks

The topology of a defined network is not modified during learning, so only an optimal combination of weights is sought. Learning takes place by adapting the weights of the network with a numerical algorithm. A learning algorithm is an adaptive method by which a neural network is tuned to implement the desired behavior. This is for example done by presenting some examples of the desired input-output mapping from the training set to the network. A correction step of the network parameters is executed iteratively until the network learns to produce the desired response.

Let us consider a feed-forward network with n input units, m output units and any number of hidden units. Let the primitive functions at each node of the network be continuous and differentiable, and the weights of the network be real numbers that are randomly initialized. Lastly, let the training set $(x_1, t_1), \dots, (x_p, t_p)$ consist of p ordered pairs of n- and m-dimensional vectors, where t_i is the target vector of x_i. So, when feeding x_i to the network, we expect the output to be t_i.
When the input x_i from the training set is presented to the network, it produces an output vector o_i different from the target vector t_i. Our aim is to make o_i and t_i identical for $i = 1, \dots, p$ by using a learning algorithm. More precisely, we want to minimize the loss function $\mathcal{L}_{NN}$ of the network, which can be, for instance, defined with the l_2 loss:

$$\mathcal{L}_{NN} = \frac{1}{2} \sum_{i=1}^{p} \| o_i - t_i \|_2^2 \, . \tag{2.5}$$

When unknown inputs are presented to the network, it is expected to interpolate. It should recognize whether a new input is similar to a learned one and if so produce a similar output.
In order to find a local minimum of the loss function – or if possible even the global minimum $\nabla \mathcal{L}_{NN} = 0$ – the *backpropagation algorithm* can be used. The gradient of the quadratic loss function, $\nabla \mathcal{L}_{NN} = (\frac{\partial \mathcal{L}_{NN}}{\partial w_1}, \frac{\partial \mathcal{L}_{NN}}{\partial w_2}, \dots, \frac{\partial \mathcal{L}_{NN}}{\partial w_j})$, is a continuous and differentiable function of the weights $w_1, w_2, \dots, w_j$. It can be computed recursively with the help of the chain rule, since the network is equivalent to a chain of function compositions. We can thus minimize $\mathcal{L}_{NN}$ by using an iterative process of *gradient descent*, every time calculating the gradient and using it to correct the weights.

$$w_i^{k+1} = w_i^k - \gamma \frac{\partial \mathcal{L}_{NN}}{\partial w_i^k} \text{ for } i = 1, \dots, j \tag{2.6}$$

Each weight is adjusted by adding the partial gradient to it. γ represents the *learning rate*, i.e., a parameter which defines the step length of each iteration in the negative gradient direction.

We will now take a closer look at gradient based optimization algorithms.

Standard gradient descent (batch learning) is an iterative method to find a minimum. It works with objective functions that are composed of a sum of differentiable functions. It sums up all gradient updates of the training set for the specific weight before executing the update of the weight.

Stochastic gradient descent (SGD, online learning) works similarly, but the true gradient for the specific weight is approximated by the gradient calculated at a single example. The algorithm thus performs a weight update after every randomly selected example of the training set. To ensure that the algorithm converges, the training set can be passed several times, or an adaptive learning rate can be used.

A compromise between the two approaches is to compute the gradient with the help of more than one training example for each step (on a *mini-batch*). Working on mini-batches usually leads to smoother convergence, in contrast to the zig-zagging often seen with SGD.

Root Mean Square Propagation (RMSProp)[37] is a modified version of SGD, and can even work with mini-batches. It adopts the learning rate to each parameter, by using a running average of the squared gradients.

Adaptive Moment Estimation (Adam)[17] is an updated version of RMSProp. It keeps running averages of both the gradients and the second moments of the gradients.

2.2 Generative models

Let us assume we have a dataset and a distribution p_{data} that represents that dataset. Moreover, let us assume that the distribution p_{data} is unknown to us, and we can only use samples to find out more about it.
The generative models that are investigated within this work are models that learn to produce new image samples that appear to come from the data distribution p_{data}. In an unsupervised learning phase the input for the model consist of samples from the dataset. During the training the model learns to match the distribution p_{model} to p_{data}. The output (when testing)

are image samples from the learned p_{model} distribution.
It is obvious that generative models are of great value when the probability distribution p_{data} is a very complicated distribution and very hard or even intractable to infer. So, having a generative machine that generates samples from p_{model} that appear as if they come from p_{data}, without having to deal with the complex probability distribution itself is useful. This is especially true if we have another process that requires samples from p_{data}, as we can get samples relatively cheaply using the trained model.

As mentioned before and shown in figures 1.2 and 1.3, the desired distribution p_{data} may be multimodal, meaning there can be many different plausible outcomes. The samples drawn from the learned distribution p_{model} should reflect that. For instance, when the task is to produce similar images as in the MNIST dataset (handwritten digits from 0 to 9), all digits should be present if enough samples are generated and the style of the digits in the output samples should differ, e.g., partly being cursive or bold similar to the original dataset.

Generative models can also be used for tasks like single image super resolution (input: low resolution, output: synthesized higher resolution) [22], text to image [42] and image to image translation (e.g., aerial photo to map) [16].

2.2.1 Variational Autoencoder (VAE)

To understand what a *Variational Autoencoder* is, let us first see what an Autoencoder is, or even simpler, an Encoder.

Encoding means to translate from one domain to another, and if desired, with the possibility of getting the original input back, which is *decoding*. This procedure is known from Cryptography, e.g., applying Caesar chiffre, or in Image compression, e.g., run length encoding images. So having an Encoder (and Decoder) means having a function mapping from a certain domain to another one (and back respectively).

In image applications, encoding can be realized via convolutions, and decoding via transpose convolutions, so the Encoder and Decoder are

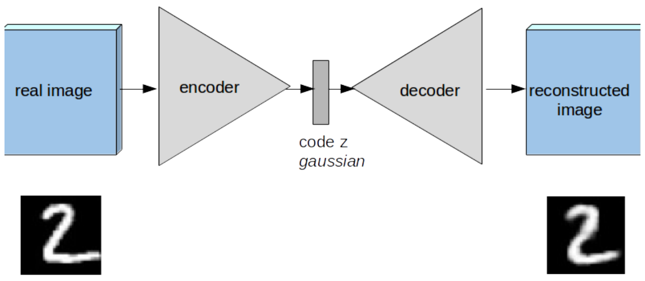


Figure 2.3: Autoencoder structure

usually neural networks. The *Encoder* maps an input image to a smaller representation, namely the *code vector*, through a convolution or a series of convolutions, and the *Decoder* can transform that code vector back to the original image by a convolution or a series of deconvolutions. That leads to the certain shape of an *Autoencoder* as can be seen in figure 2.3.

Autoencoders in general thus try to represent the data of an image in the compact representation of a code vector. An application is to store this code vector instead of the original image to only reconstruct the original image later, when needed. How well they can reconstruct the input depends on the size of the code vector, i.e., how many features can be stored, as well as on the size and complexity of the Encoder and the Decoder networks. The training loss for an Autoencoder is thus a reconstruction error, i.e., the error comparing the reconstructed image to the original image with a metric like Mean Squared Error (MSE).

However, what we would like to have is a generative model that reconstructs an image similar to the input image, but not an Autoencoder that memorizes the input image completely. This can be achieved by adding a regularization term (for instance a sparsity penalty) to the training loss or by only keeping the strongest hidden neuron activations [24].

A similar method is the *Variational Autoencoder (VAE)* which is useful for not only reconstructing original images from their codes, but also generating new similar, but not identical images by sampling from a prior distribution. A VAE is still formally an Autoencoder, but with the additional constraint on the Encoder that it must generate a code which is roughly like being sampled from a desired prior distribution. Mostly a simple distribution like Gaussian is chosen for the prior [6]. As the goal of VAE is to be able to generate new samples according to a given prior distribution, it is still seen as a generative model. The VAE was invented at the end of 2013 by Kingma and Welling [18] as well as Rezende et al. [30]. It is an unsupervised method that can be trained with standard gradient descent-based methods.

A VAE is able to capture dependencies between pixels in an image, for instance, pixels of the same color, identical objects, or movements in the same direction. Because of that, the low-dimensional representation of the data that is learned by it is not called *code vector* but rather *latent space* or *latent variable z*. The *latent* refers to the information represented in the code vector, which cannot be influenced directly, but if we look at the images generated when interpolating one dimension of it, we might see what changes in the generated images are influenced by it. However, since the representation is intended to be very compact, it is very unlikely that a single dimension just represents one single property. The latent variable therefore stores the truly relevant data from the input image.
Since these latent variables are usually not designed manually, it is mostly unknown what information exactly is stored there, and we can only get hints about what they contain through interpolation. The latent variable is usually multi-dimensional, but for visualization purposes we can choose a two dimensional vector. In figure 2.4 we see that for a VAE trained to reproduce the MNIST dataset of handwritten digits, the horizontal dimension is the style of the handwriting – rather straight turning to rather cursive.

In the paper [29] Radford et al. train a generative network on a newly assembled dataset of faces, and show that it is possible to do easy arithmetic calculations in latent space with associated property labels to get a desired output image.
[smiling woman] - [neutral woman] + [neutral man] = [smiling man].

The parameters that can be chosen for a VAE are partly architecture-related, e.g., the number of hidden layers, the number of neurons, the activation functions and the initialization of the network weights. On the other hand we can choose the learning rate for the optimization algorithm, the batch

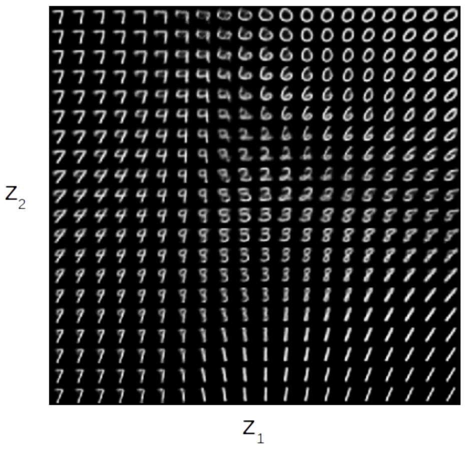


Figure 2.4: 2D MNIST learned latent space: reconstructions interpolating in z-space seem to get more cursive in horizontal direction

size for the mini-batches that are used for the optimization step and the number of training epochs.

The VAE has to handle the trade-off between reconstruction accuracy of the network versus the closeness of the latent variable z to the desired prior distribution $p(z)$. The loss $\mathcal{L}_{VAE}$ that is optimized therefore consists of two parts [1]– the *reconstruction loss* $\mathcal{L}_{rec}$ and the *latent loss* $\mathcal{L}_{latent}$.

$$\mathcal{L}_{VAE} := \mathcal{L}_{rec} + \mathcal{L}_{latent} \tag{2.7}$$

The reconstruction loss $\mathcal{L}_{rec}$ (also called *generative loss*) is defined as the negative log probability of the input image x under the reconstructed output distribution of the Decoder $Dec(x \mid z)$. It represents the measurement of accuracy of the reconstruction.

$$\mathcal{L}_{rec} := -\mathbb{E}_{z \sim Enc(z|x)}[\log Dec(x \mid z)] \tag{2.8}$$

[1]The loss can be derived from a lower bound on the data likelihood of the network parameters [18]

The latent loss $\mathcal{L}_{latent}$ is the *Kullback-Leibler divergence (KL)* between the distribution in latent space induced by the Encoder $Enc(z \mid x)$ and the prior distribution $p(z)$. The KL divergence is a measure of difference between two distributions. It thus serves as a regularizer for the optimization, forcing the latent space to be as close as possible to the chosen prior, e.g., a Gaussian distribution.

$$\mathcal{L}_{latent} := \mathbb{KL}[Enc(z \mid x) \parallel p(z)] \tag{2.9}$$

If both distributions, $Enc(z \mid x)$ and $p(z)$, are assumed multivariate Gaussians, the Kullback-Leibler divergence $\mathbb{KL}$ can be calculated in closed form [6]. The final latent loss is then given as:

$$\mathcal{L}_{latent} = \frac{1}{2}(1 + \log(\sigma^2) - \mu^2 - \sigma^2) \tag{2.10}$$

Usually the Adam optimizer [17] is chosen to minimize the cost function. The optimization with such a standard gradient descent algorithm is only possible due to the *reparametrization trick*. Without this reparametrization, the Kullback-Leibler divergence would not be viable for backpropagation, and it would therefore not be possible to optimize the loss with a SGD algorithm. The reparametrization, shown in figure 2.5, reformulates the latent loss containing the Kullback-Leibler divergence. It tells the Encoder to produce a vector of means μ and a vector of standard deviations σ instead of producing a code z. A sampled z for the Decoder is obtained by summing the mean and the product of standard deviation and a sample from a prior distribution, changing its location and scale.

$$z_{sampled} = \mu + (\sigma \cdot samples) \tag{2.11}$$

Hence the intractable sampling operation is shifted into an input layer and made usable for backpropagation.

This reparametrization trick in fact also generalizes the network and makes the Encoder more efficient in generating new examples, since a random code (not from the training set) will lead to a new decoded image.

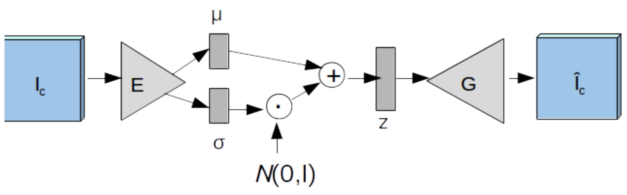


Figure 2.5: Reparametrization trick in the VAE: mean and standard deviation vectors are used to sample z for the Decoder.

With the trained network, apart from generating new similar images, the partial step of transforming images to latent space or the broader step of reconstructing input images is possible. In figure 5.2(a) in the evaluation section we can see an example reconstruction of images of the MNIST dataset. It is visible that the reconstructed images are close to the input images, but a bit blurry and some digit parts are missing or incomplete.

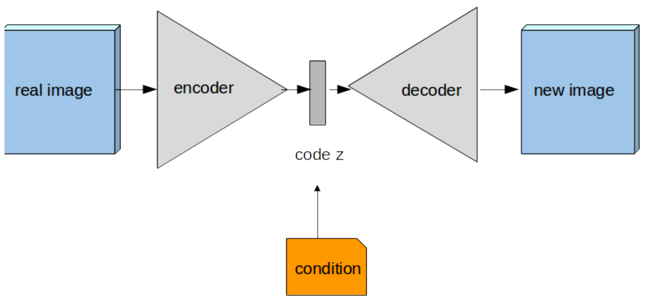


Figure 2.6: Conditional VAE: some dimensions of the latent code vector are manually determined to represent conditional information like labels.

Conditional VAEs (*cVAEs*) are an expansion of VAEs - as the name suggests, they condition the network on something, e.g., a label. It is not standardized in what form and when the conditional information is provided, e.g., it can be concatenated with the code vector at the input of the Decoder or at one or more convolutions. A rough structure is shown in figure 2.6. Sticking to the MNIST example, we could provide the label of the digit that is shown in the image as additional information to both the Encoder and the Decoder, and thus make sure that when giving the Decoder a certain latent code and a digit label, that only that digit is produced as output. We can condition the network on any information, not only labels, but also images or text.

To sum it up, due to the Encoder-Decoder architecture, the VAE enables comparison of generated images to input images. For the comparison of images, different image metrics are used. With VAEs we can also do arithmetic operations in latent space to create an image with certain features. However, VAEs tend to give blurry outputs, since they work with the *Mean Square Error* directly.

2.2.2 Generative Adversarial Network (GAN)

Another generative model is the so called Generative Adversarial Network. It was first introduced in a NIPS 2014 paper by Ian Goodfellow et al. [10] and rose a lot of interest from the community. In the paper Goodfellow gives a short overview of the GAN model and its applications. Later [11] he presents the GAN in more detail including: (1) Why generative modeling is a topic worth studying, (2) how generative models work, and how GANs compare to other generative models, (3) the details of how GANs work, (4) research frontiers in GANs, and (5) state-of-the-art image models that combine GANs with other methods. Soon also variants of the standard GAN like Deep Convolutional GAN (DCGAN) [29] or Sequence-GAN [41] were developed and GANs got used for many applications, as presented in section 3.2.

As seen in figure 2.7, the structure of a GAN is like a 2-player game, with the two players called *Generator G* and *Discriminator D*. The name adversarial stems from there, since it can be analyzed with the tools of game theory, where two adversaries play against each other. The Generator creates samples, and is trained to make them look as if they were coming from the training data (distribution). Its goal is to make its samples indistinguishable from the training distribution samples. The Generator gets noise as input so he will not produce the same image once he is trained but gives another image variation for every little change in the noise input.

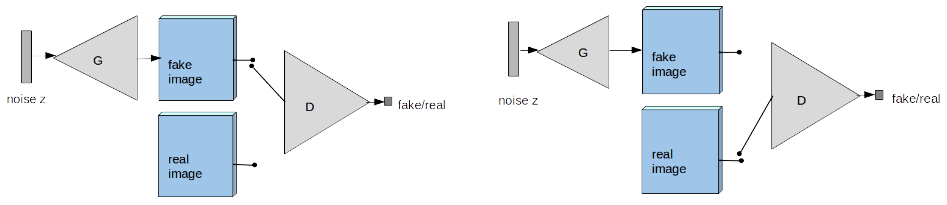


Figure 2.7: Generator G creates fake images from noise. They are given to the Discriminator D for evaluation. Discriminator D gets fake and real images. It tries to classify all real ones as 1s and all fake ones as 0s. Learning is converging if neither Discriminator nor Generator can improve its performance.

The Discriminator gets in total 50% real samples from training data and 50% fake samples from the Generator, examines one by one and has to decide if the sample is real or fake. *D*s goal is to have a low classification error. This procedure corresponds to traditional supervised learning.

The solution to this 2-player game is a so called *Nash equilibrium* [13]. In this case, we want the Generator to learn to generate samples so well that they are drawn from the same distribution like the training data and the Discriminator to have a 50% chance to guess right. As seen in figure 2.8, at first the distribution of the generated samples and the original data distribution do not match. However, the Discriminator will soon learn features to distinguish real from fake samples. Through the adversarial learning, information is backpropagated to the Generator, and the Generator can learn to reproduce the important features. In the end, if a perfect equilibrium is reached, the Discriminator can only guess, and the distribution represented by the Generator matches the original data distribution from the given dataset.

Usually both Generator *G* and Discriminator *D* are deep neural networks, so to enable learning with backpropagation for both *G* and *D*, they must be fully differentiable. The training consists of updates on both the Generator and the Discriminator, either done after each other or simultaneously. Some works recommend doing more steps for one of the players, e.g., the improved Wasserstein GAN [12], whereas Ian Goodfellow suggests simultaneous

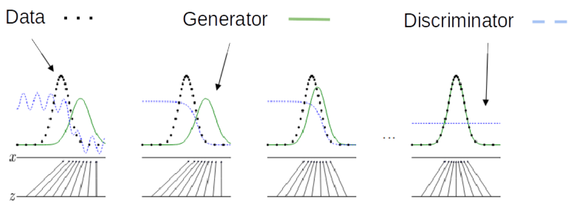


Figure 2.8: Learning of a GAN. The Discriminator learns first and will incentivise the Generator to do so as well. If a perfect equilibrium is reached, the Discriminator can only guess, and the distribution represented by the Generator matches the original data distribution. Figure adapted from [10].

updates to be most successful in his GAN tutorial from 2016 [11]. The updates are usually done as a step of Stochastic Gradient Descent, e.g., with the Adam optimizer.

The cost function for D is chosen as a cross-entropy loss. The original formulation of the cost from Goodfellows paper [10] is as follows:

$$\mathcal{L}_{GAN} = \min_G \max_D \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.12}$$

For a Generator update, only the last part of the term has to be evaluated, since the Generator is not involved in assessing the real samples. The Generator tries to minimize the equation 2.12. If the Discriminator outputs 1 for a fake sample, the logarithm will be negative. The maximum possible – if the Discriminator classifies all fake samples as 0s correctly – is the Generator cost to be zero since $\log(1) = 0$.
For a Discriminator update, the whole term is taken, and it is maximized, trying to reach zero for both parts by classifying all real examples as 1s (first part) and all fake examples as 0s (second part).

An improvement to the cost function was suggested by Arjovsky et al. [2] in 2017, taking the so called *Earth-Mover distance (EM)* or *Wasserstein-1 distance* as the distance between the probability distributions. This distance gives an estimate about the cost of transforming p_{data} into p_{model}. The Wasserstein distance defined the name of the new GAN variation, namely *Wasserstein GAN (WGAN)*. The authors also introduced *weight clamping* to operate on compact space and have k-Lipschitz functions which are differentiable. Because of that we are able to train the Discriminator till optimality and thus their model shows promising results to get rid of the problem of mode collapse that the original GAN formulation had. They call the Discriminator part *critic*, since it is not discriminating any more. The best critic is found by optimizing this equation:

$$\min_G \max_{D \in \mathcal{D}} W(D, G) \tag{2.13}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions and

$$W(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[D(x)] + \mathbb{E}_{z \sim p_z(z)}[D(G(z))] \tag{2.14}$$

The authors admit that the training may become unstable if a momentum based optimizer like Adam is used, or if too high learning rates are applied.

It was shown in [12] by Guljarani et al. that Adam can actually be used and accelerates the learning, if weight clipping is not used but instead of it another method called *gradient penalty* is applied to ensure Lipschitz continuity of the Discriminator. Guljarani et al. claim that weight clipping leads to undesired behaviour, e.g., deep WGAN critics not converging and failing to capture higher moments of the data distribution. They clarify that it occurs because weight clipping makes the critic turn toward simpler functions, only learning approximations of the optimal functions. They also point out that there can be interactions between the weight constraint and the cost function causing exploding or vanishing gradients. Moreover, they mention that batch normalization is changing the form of the Discriminator problem and therefore should be left out, they propose to use layer normalization instead.

The gradient penalty, which enforces a gradient norm of 1 almost everywhere for the optimal WGAN Discriminator function puts a penalty on the gradient norm for random samples p_{itpl} taken form the straight lines between points from the data distribution and the Generator distribution.

The Discriminator will thus be 1-Lipschitz if it has a unit gradient norm almost everywhere. This gradient penalty term is added to the Discriminator loss of the original WGAN and is empirically given a weight of $\lambda = 10$. The authors suggest a two-sided penalty, only forcing the gradient to go towards 1 and not having to stay below 1 at all times. The loss for the improved WGAN training is thus looking like this:

$$\mathcal{L}_{wgan-gp} = \min_G \max_D W(D,G) + \lambda \mathbb{E}_{\hat{x} \sim p_{itpl}}[(\| \nabla_{\hat{x}} D(\hat{x}) \|_2 - 1)^2] \quad (2.15)$$

As mentioned earlier, the paper also suggests five Discriminator updates for every Generator update, both networks being updated after each other and not simultaneously. Guljarani et al. claim that their algorithm is stable for different sizes and designs of network architecture, converging for most models and showing a better performance than the original WGAN formulation. The implementation of this thesis is based on the code of the improved WGAN training.

There are so called *conditional GANs*, similarly structured like the conditional VAEs that were explained earlier. The structure can be seen in figure 2.9. The network also gets a condition as additional input, e.g., the label of the image it is learning. Thus the GAN can for instance learn to produce a specific digit when trained with the MNIST database and being conditioned on labels. It means there is an additional input for the Generator next to the noise and also the same additional input for the Discriminator next to the real or fake image it has to classify. The Generator can and should use the additional information to generate an image corresponding to that label, since the Discriminator will use the information to learn to discard any images that are not matching the condition it gets as input. When the Generator is finally trained and noise as well as a label as condition are given to it, it will output a corresponding handwritten digit – matching the label – similar to those images of the digit that were in the database.

The cost function does not have to be changed, only the input for both Generator and Discriminator. Analogously to the conditional VAE, it must be considered how the conditional information is handled in the Generator

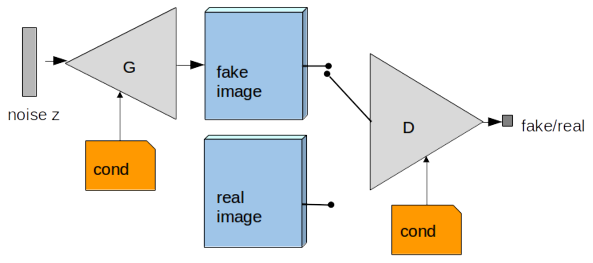
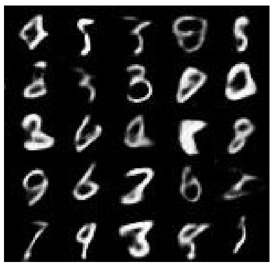


Figure 2.9: Conditional GAN. Conditional information, for instance labels, are provided to both Generator and Discriminator. The conditional information gives both networks the opportunity to learn to represent or discriminate the image with that specific information.

and the Discriminator respectively. It can be appended to the rest of the input or somehow processed with it.

All in all, GANs are widely used, despite some drawbacks they have compared to VAEs. Comparing generated to original images is not possible for standard GANs – the images produced by GANs are generated out of arbitrary noise, meaning we cannot generate an image with specific features. If the GAN gets a dataset with three different classes, for any noise input it may give an output resembling any of the three classes. However, as just mentioned, generating an image with specific features is possible with conditional GANs. But apart from brute force search over the entire distribution there is no way of determining which initial noise values would produce a certain image from the dataset. Another problem is that GAN only discriminates between *real* and *fake* images. There are no constraints that an image of for instance a cat has to look like a cat. This may lead to images where there is no actual object in the generated image, but the style looks like the training images. On the other side, GANs tend to give sharp output images, since the adversarial gameplay is leading them towards good image quality, and with the improved WGAN formulation they get more stable in training as well.


(a) VAE

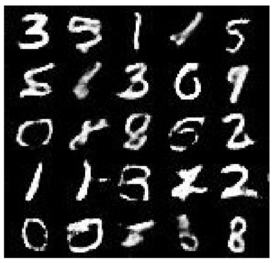

(b) GAN

Figure 2.10: The Variational Autoencoder generates rather blurry images, the Generative Adversarial Network rather sharp digits. The models are trained on the MNIST dataset.

2.3 Datasets

There are numerous Vision datasets available online, e.g., for object recognition, classification, or Optical Flow computation. We will now take a closer look at two of them.

2.3.1 MNIST

To accelerate the development of our novel model, the well-known MNIST dataset that is easy and fast to learn was taken to test first implementations.

MNIST The MNIST Database, developed in 1998, shows handwritten digits which are sorted in 10 classes representing the digits from 0 to 9. The image size is 32x32x1 (grayscale). It consists of 60, 000 Images in total and is used for Image Classification.
`http://yann.lecun.com/exdb/mnist/`

MNIST is often used in Machine Learning to develop and test new algorithms. It allows proofing of concepts with reasonable effort before switching to more difficult datasets. Thus it can be used a benchmark to demonstrate that a new method is able to distinguish the numbers or is able to generate similar numbers in case of generative models. The dataset provides labels that can be used as condition for the networks to produce a certain type of digit.

2.3.2 Moving MNIST

Since it is not possible to do motion prediction with MNIST, a similar dataset to MNIST was needed, but consisting of video data.

moving MNIST The moving MNIST dataset, developed in 2015, consists of videos generated from the MNIST database. It shows 2 handwritten digits in motion. In total there are $10,000$ sequences with 20 frames each, and the dataset is used for Frame or Sequence Prediction. The image size is 64x64x1 (grayscale).
`http://www.cs.toronto.edu/~nitish/unsupervised_video/`

Moving MNIST is building on MNIST, showing two digits moving in different directions in a 64x64 frame over the course of 20 frames. The digits are sometimes overlapping, but they will never disappear, as they always bounce from the borders. As visible in figure 2.11 the movement in moving MNIST is not complicated, so the results can be easily checked for plausibility.

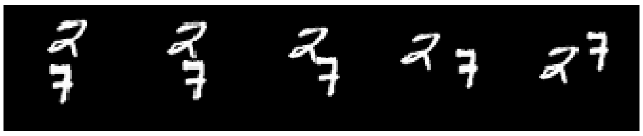


Figure 2.11: Moving MNIST dataset. The motion is shown with frames t, t+1, t+2, t+4, t+9.

2.4 Image Quality metrics

For the task of frame prediction, the result, i.e., the predicted frame, can be compared to the ground truth frame to see how plausible and similar the prediction is. Since it is impossible to have a human comparing all the amount of data, we need to find a metric that is as close to human perception as possible. Up to date, no such metric that is perfectly aligned with the human vision is developed, but there are different metrics that can be used to compare two images with each other, or rather to measure the similarity between them.

MSE Mean Squared Error
PSNR Peak-Signal-to-Noise Ratio
SSIM Structural Similarity Index Measure

All are full-reference metrics, i.e., they take the original (noise-free and uncompressed) image I as reference and compare the approximated (noisy or compressed) $\tilde{I}$ to it. The three metrics will be presented in more detail in the following sections.

2.4.1 Mean Squared Error (MSE)

MSE is highly used for machine learning tasks and uses the l_2 *norm*. MSE is very simple, but does not reflect human perception.
The Mean Squared Error is a summarized error, taking the sum of the squared errors computed at every pixel. The squared error is taken because otherwise the error can be positive at one pixel and negative at another one and they would cancel out each other when summing up. Explicitly it is expressed as:

$$MSE = \| I - \tilde{I} \|_2^2 = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (I(i,j) - \tilde{I}(i,j))^2 \qquad (2.16)$$

where m and n are the image width and height.

2.4.2 Peak-Signal-to-Noise Ratio (PSNR)

PSNR is a metric for comparing signals, i.e., audio, images or video data, especially compressed signals to the original ones. It is thus used to see how well a lossy compression can be reconstructed.
The Peak-Signal-to-Noise Ratio is most easily defined via the MSE [3], and the unit is decibel (dB).

$$PSNR = 10\,\log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20\,\log_{10}\left(MAX_I\right) - 10\,\log_{10}\left(MSE\right) \quad (2.17)$$

MAX_I is the maximum possible pixel value of the image, e.g., when the pixels are represented using 8 bits per sample, it is $2^8 - 1 = 255$. If the two images I and $\tilde{I}$ are identical, the MSE is zero, and the $PSNR$ is infinite. PSNR can also be computed on RGB color images, for which the MSE is summed over the channels and divided by three.

Typical values for $PSNR$ for example in lossy image and video compressions are between 30 and 50 dB (for $MAX_I = 255$). Generally, the higher the PSNR value, the better is the compression method, or rather the quality of its reconstruction.

2.4.3 Structural Similarity Index Measure (SSIM)

SSIM was developed to improve on image and video comparison by taking texture into account. The SSIM was designed by Wang et al. [39] in order to indicate perceived changes in structural information of the image, trying to be more consistent with human perception of differences. Added noise on top of the image, lost structure or harsh smoothing will result in a lower similarity score, which is not always the case with simpler methods like MSE as you can see in figure 2.12.

The SSIM computation works with an index map created by sliding windows of a fixed size. The index map shows local image quality over space – a higher SSIM value of a certain window is depicted as a brighter pixel in the

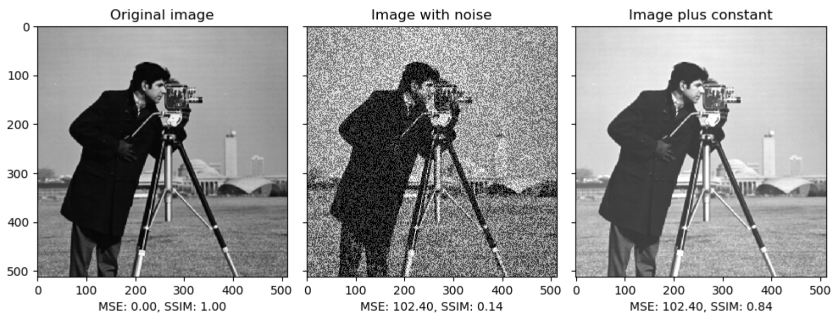


Figure 2.12: Comparison of MSE and SSIM values for different noise. In the right image the same absolute value of noise is added as positive constants. Images are generated with the script from `http://scikit-image.org/docs/dev/auto_examples/transform/plot_ssim.html`

index map. The typical window size is 8x8. By not calculating the full pixel-by-pixel displacement throughout the image, the computation complexity can be decreased. The window size determines the scale, the final value is obtained by averaging over all patches and is thus depending on image resolution. Extensions of SSIM include a multi-scale version (MS-SSIM) that is weighting the information content of different scales. The SSIM is usually applied on grayscale images, although a multi-channel version exists.

The formula for the index between two windows x from I and y from $\tilde{I}$ is:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2.18}$$

where μ_x, μ_y is the mean of x or y respectively, σ_x^2, σ_y^2 is the variance of x or y respectively, σ_{xy} is the covariance of x and y, and $c_1 = (0.01 \cdot MAX_I)^2, c_2 = (0.03 \cdot MAX_I)^2$ are two variables to stabilize the division.

The resultant SSIM index is a bounded decimal value between -1 and 1, where the unique maximum 1 represents two identical images. The closer to 1 the value is, the more similar are the images.

MSE, PSNR and SSIM are to this day the most commonly used metrics for image comparison.

2.5 GAN evaluation metrics

Apart from the evaluation of the generated images with the help of the aforementioned image comarison metrics, it is hard to judge on a quantifiable basis if a GAN is performing well.

2.5.1 Inception Score

In [33] the authors introduce a metric called *Inception Score (IS)*, as an attempt to quantify realism in GANs. The IS is a measure of *on average, how different is the score distribution for a generated image from the overall class balance*. It can be used to measure the quality of images generated from GAN models, correlating with human judgment, and is thus used to quantify the performance of the GAN in general.

The formula is

$$IS(x,y) = e^{(\mathbb{E}_x[\mathbb{KL}[p(y|x)\|p(y)]])} \tag{2.19}$$

There are two criteria:

- $p(y \mid x)$ represents *Saliency*
  how easy it is to determine what class an individual image belongs to
- $p(x)$ represents *Diversity*
  how well-balanced the training set is, not only showing the same object

In order to compute this score for a set of generated images a good image classifier is required – the authors used a pretrained *Inception Network* for calculating the distributions, which is why the metric is called Inception Score.

A disadvantage of the Inception Score is that the statistics of real world samples and synthetic samples are not compared.

2.5.2 Fréchet Inception Distance

To improve the Inception Score, the *Fréchet Inception Distance* (FID) was introduced in [13]. It is proven to be more consistent with a rising noise level than the Inception Score.

The authors use the first two moments, namely mean and covariance, of the feature space generated from the image. Since Gaussians represent the maximum entropy distribution in terms of these moments, they set up the FID to measure the difference between the synthetic images and real-world images Gaussian distribution. They claim that the FID is consistent with increasing disturbances and human judgment, e.g., when adding Gaussian noise, Gaussian blur or salt and pepper noise, the disturbance level of FID rises correspondingly.

A drawback of both Inception Score and FID is that they need a large sample size to give good results, suggested are 50000 samples. When using fewer samples, the FID will be overestimated, giving better results than it would give with more samples, and the variance of the estimates increases with less samples.

2.5.3 Limitations

In general, there is no standard evaluation method of GANs, but a variety of measures to classify the performance, realism and diversity reached by a GAN. There are also measures comparing two or more GANs, or tests done with humans to evaluate image quality and realism. [3] enumerates these measurements that were used in different papers and classifies them into quantitative and qualitative measures. The author also states seven desiderata of a GAN evaluation: high fidelity, diverse samples, disentangled latent space and space continuity, well-defined bounds, sensitivity to image distortions and transformations, agreement with human perceptual judgements, as well as low sample and computational complexity. From the analysis in the paper the measures fulfilling the most of these criteria are the FID and Image quality measures like SSIM, but the author points out that there is no measure that fulfills all desiderata up to this point. Therefore,

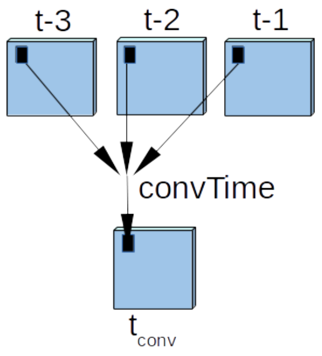


Figure 2.13: A convTime layer for three past input frames. The output t_{conv} is computed by convolving the same pixel locations.

using more than one measure to evaluate the performance of a GAN is appropriate.

2.6 Time convolution

The goal of the newly developed generative model is to have more than one past frame as input. This information over time has to be processed. A simple way would be to concatenate all past frames and give the result as conditional input to the network, but the conditional input is supposed to be small in comparison to the regular input, since usually the networks learn better features by themselves than using manually determined ones. There already exist different layers to deal with time information in neural networks, of which two will be presented now.

In [7] and [8] Feichtenhofer et al. propose a *convTime* layer and apply it to existing networks. Their overall task is action recognition, so they do convolutions in time in order to recognize motions in the scenes. ConvTime is a regular convolutional network layer, but the dimensions of the input are transposed in a way that not spatial neighboring pixels in the image are convolved like in CNNs, but different time points of the same pixel, as can be seen in figure 2.13.

So the layer gets an input of $[batchsize, x \cdot y, channel, timesteps]$ and will output the convolved information in the format $[batchsize, x, y, channel]$. The filter determines how the different time information of the same pixel is combined. It is learned like in any other network layer, by backpropagation. There are different variants of convTime, but only V3 includes feature information of the pixel, i.e., the values of *channel* for the given pixel are also used for the convolution.

Three versions are tested in the scope of this work:

V1 convTime without using feature information.
V2 convTime without using feature information, but having an own time filter for each feature.
V3 convTime including feature information.

The results of the test can be found in section 5.3.

Another layer that can be used to handle time series input is the so called convLSTM, implementing Long Short-Term Memory cells. The concept of convLSTM for 2D images was first used in CNNs in 2015, proposed in [34], although the principle of the Long Short-Term Memory as a way to process time information was demonstrated already in 1997 by [14]. A LSTM cell has gates with thresholds that decide if information is stored, passed on, or forgotten. It always receives the last internal state as input again and uses it for the current computation. Thus the model can be unrolled over time to compute the latest state. In [23] a time analysis with convLSTM is done in their so called Motion Encoder. With the help of the convLSTM layer they calculate a mean and standard deviation map from multiple input frames, which is then given to the Frame and Flow Generator to sample z from mean and standard deviation. The convLSTM layer in the Motion Encoder is located after the spatial convolutional layers, so it is the last layer before the mean and the standard deviation are returned.

In this thesis the implementation of time processing layers was restricted to a convTime layer due to time limitations.

3 Related Work

As introduced in section 2.2, VAEs and GANs can generate samples from an unknown data distribution. These generative models can be trained to match any data distribution. Since the developed model is based on the idea and implementations of VAE and GAN, this chapter is splitted into works applying VAE, GAN or a mixture of both. However, this is just a brief overview of selected papers due to the sheer amount of VAE and GAN papers that were published in the last five years.

3.1 Applications of VAE

Walker et al. [38] use VAEs to predict the future from a single image. They predict the dense trajectory of pixels in a scene with the help of a conditional VAE. Their only input is one static image. They exploit the fact that a VAE can make multiple different predictions to account for ambiguous action movements – the model can output several possible future frames showing different motions.

In 2016, Pu et al. [28] use a VAE to learn images and associated labels using deep convolutional networks. Their model can be used for semi-supervised (with labels/captions) or even unsupervised learning (only images).

An extension of VAEs are *Importance Weighted Autoencoders* (IWAEs) introduced in [4]. They show that the VAE objective can lead to overly simplified representations which fail to use the network's entire modeling capacity. Thus they propose using a strictly tighter log-likelihood lower bound derived from importance weighting with the VAE architecture. Their recognition network uses multiple samples to approximate the posterior, which leads to increased flexibility to model complex posteriors which do not fit the VAE modeling assumptions.

Another extension are Ladder VAEs, proposed by [35]. This model recursively corrects the generative distribution by a data dependent approximate likelihood, resembling the Ladder Network.

3.2 Applications of GAN

Denton, Chintala and Fergus [5] use the original GAN formulation to generate higher resolution images than in the original paper by Goodfellow [10]. They propose to build a laplacian pyramid model to upscale to higher resolutions in several steps and learn these GANs separately. Thus the pyramid starts with an 8x8 image, via 16x16 and 32x32, and outputs a 64x64 image.

An expansion of GANs were the so called deep convolutional GANs (DC-GANs) presented by Radford et al. in 2015 in [29]. The authors propose to use the success of supervised convolutional networks (CNNs) also on unsupervised tasks by creating an architecture with certain constraints - the convolutional networks are learned with the adversarial GAN loss. This is claimed to make the training more stable. To gain insight into the learning, they also visualize the filters learned by the discriminator, and do semantic vector arithmetic to make the generators generate certain properties in the images. With the help of this, they show that both Generator and Discriminator learn a hierarchy of representations as features, going from parts of objects to the whole scene, and point out that these features are generalizing well, because they can even be used for new tasks.

On the basis of the first GAN formulation, Mathieu, Courprie and LeCun [26] developed a deep architecture to predict future frames giving a sequence of frames as input. They use a conditional GAN approach and provide four input frames to the Generator to predict the next frame and also providing the last frame to the Discriminator to decide if the latest frame is from the dataset or generated. They also recursively feed the predicted images back to the network to predict yet another future frame.

They use a pyramidic upscaling to obtain 64x64 images. In their appendix they also show experiments to predict eight future frames simultaneously from eight last input frames, but the results are worse than the four input for one future frame. They show that compared to the standard l_1 and l_2 loss the GAN gives better results, since it does not average over possible futures like l_2. They also use an additional sharpening loss term, named Image Gradient Difference Loss $\mathcal{L}_{gdl}$, next to a ground truth and a similarity term in order to remove the blurriness on edges of moving objects.

$$\mathcal{L}_{gdl}(\hat{I}, I) = \sum_{i,j} \Big| |I_{i,j} - I_{i-1,j}| - |\hat{I}_{i,j} - \hat{I}_{i-1,j}| \Big|^{\alpha} + \Big| |I_{i,j-1} - I_{i,j}| - |\hat{I}_{i,j-1} - \hat{I}_{i,j}| \Big|^{\alpha} \tag{3.1}$$

where I and $\hat{I}$ are the two images to be compared and $\alpha \geq 1$ is an integer.

An application for motion estimation is presented by [19], learning Optical Flow with GANs. They propose a semi-supervised algorithm to predict motion. As opposed to former attempts of estimating Optical Flow with CNNs, no large dataset with labels or ground truth flow fields is needed by exploiting the structure of GANs. They show that the adversarial loss is able to capture patterns of flow warping errors without making assumptions like *Brightness Constancy* and *Spatial Smoothness*, which are needed by unsupervised Optical Flow methods. Thus, the developed GAN algorithm makes it possible to learn Optical Flow with the help of non-synthetic, realistic datasets without Optical Flow ground truth, outperforming supervised and other semi-supervised methods in experiments.

In March 2018, Wei et al. [40] proposed to improve the current WGAN formulation with gradient penalty [12] by adding another term that they called *consistency term*, to enforce the Lipschitz continuity on the real data manifold.

$$\mathcal{L}_{WGAN+} = W(D, G) + \lambda_1 \mathcal{L}_{gp} + \lambda_2 \mathcal{L}_{ct} \tag{3.2}$$

where $W(D, G)$ is the standard WGAN loss, $\mathcal{L}_{gp}$ is the gradient penalty term with its weight $\lambda_1 = 10$, and

$$\mathcal{L}_{ct} = \mathbb{E}_{x_1,x_2}\left[\max\left(0, \frac{d(D(x1), D(x_2))}{d(x_1, x_2)} - M'\right)\right]. \tag{3.3}$$

$M' \geq 0$ is a small real constant, set to 0 for the experiments in the paper. $d(\cdot, \cdot)$ denotes any metric, e.g., the l_2 norm. The consistency term is added with the weight λ_2 set to 2, serving as regularizer. Wei et al. still suggest five Discriminator updates per every Generator update, executed after each other. They also added dropout layers and report an improved performance on MNIST among others and claim that their method ensures a better prevention of overfitting, especially if not much data is present. They support that claim by reporting very good results of learning semi-supervised with only 4000 instead of 10000 labels. The consistency term addition might also be interesting for our approach, but it was too recent and not yet acknowledged in the research world and is therefore left for future work.

3.3 Combinations of VAE and GAN

An interesting combination of the VAE and the GAN models was presented by Larsen et al. [20]. The authors use the features of the GAN discriminator as the metric for the VAE. Thus they learn their own similarity metric, which outperforms the pixelwise error used by standard VAEs. The unsupervised model proposed by them can encode, generate and compare samples from the dataset due to its structure where parts are taken from VAEs and parts from GANs. The Decoder of the VAE and the Generator of the GAN are the same network, the Encoder is in front of it and the Discriminator after it, which was adapted for this thesis. The authors suggest to replace the VAE reconstruction loss (expected loglikelihood) with a reconstruction error learned in the GAN discriminator. For this, they introduce a Gaussian observation model for the lth layer of the Discriminator $D_l(x)$.

$$p(D_l(x) \mid z) = \mathcal{N}(D_l(x) | D_l(\tilde{x}), I) \tag{3.4}$$

where $\tilde{x} \sim G(z)$ is sampled from the Generator.

With this, the VAE reconstruction error is rewritten as

$$\mathcal{L}_{ll}^{D_l} = -\mathbb{E}_{q(z|x)}[\log p(D_l(x) \mid z)]. \tag{3.5}$$

The whole loss function $\mathcal{L}_{combi}$ is consisting of three terms: the standard GAN loss $\mathcal{L}_{gan}$, the latent loss term from the VAE loss $\mathcal{L}_{latent}$, and the replaced VAE error term $\mathcal{L}_{ll}^{D_l}$. Adding them gives the loss

$$\mathcal{L}_{combi} = \mathcal{L}_{latent} + \mathcal{L}_{ll}^{D_l} + \mathcal{L}_{gan}. \tag{3.6}$$

They also show that semantically meaningful arithmetic options can be applied in latent space to get an image with certain desired attributes or missing certain attributes – e.g., adding glasses, a smile or a beard to a face.

A mixture of both models is also presented in [23] where the goal is to predict the pixel-wise flow in videos with a dual motion GAN. The primal future-frame prediction and dual future-flow prediction enhance each other by generating informative feedback signals. The future-flow prediction is able to help infer realistic future-frames, while the future-frame prediction in turn leads to realistic optical flows, working on making both synthesized future frames and flows indistinguishable from reality. To handle natural motion uncertainty in different pixel locations, a new probabilistic motion encoder, based on VAEs, is used.

Some parts of that paper are reimplemented in this work, since no code was made available by the authors even when directly requested. Analogously to their paper we set up an Encoder and a time convolution before giving the output of those to a GAN.

Another mixture, called *adversarial autoencoder (AAE)*, is introduced in [25]. The authors propose a probabilistic autoencoder that uses the GAN model to perform variational inference by matching the aggregated posterior of the hidden code vector of the autoencoder with an arbitrary prior distribution. Their applications include semi-supervised classification, disentangling style and content of images, unsupervised clustering, dimensionality reduction and data visualization.

In 2018, Hou and Qiu [15] propose to improve VAE performance by applying deep feature consistency (DFC-VAE). They state that the VAE output and input images should have similar features. They also add adversarial training by feeding the output of the VAE to a discriminator, which according to them helps to generate more natural and realistic images. They show this with face generation and latent space manipulation with facial attribute recognition features.

A network having similar goals and structure as this thesis is the FutureGAN introduced by Aigner and Körner in 2018 [1]. They developed a multi-scale GAN that is able to predict a sequence of future frames given a sequence of past frames. By using progressively growing GANs the output frame size is increased step by step. Their generator consists of an Encoder and a Decoder part, and the input to the Encoder is a sequence of past frames. 3D convolutions are used to deal with spatial and temporal dependences at once. The training is done with a WGAN gradient penalty loss, but on top of that Aigner and Körner add an epsilon penalty term to prevent the overall loss from drifting. Part of the experiments are conducted on the moving MNIST dataset that was also used for our experiments on frame prediction. They also use the same evaluation methods like us, namely MSE, PSNR and SSIM.

Another paper describing a mixture of VAE and GAN is [32]. Their optimization objective is a combination of the standard Discriminator equation and the reconstruction loss from VAEs, preventing mode collapse for the learned model. The authors replace the intractable likelihood of the variational interference by a synthetic one and also substitute an implicit distribution for the unknown posterior distribution. The synthetic likelihood and the implicit distribution are learned by Discriminators.

4 Implementation

We know that generative models like GANs can generate images similar to the ones provided in a dataset, that VAEs can encode an image to a vector with fewer dimensions, and that a GAN Discriminator can distinguish generated images from real images. In the following chapter it will be demonstrated how all these advantages of the different models can be used in a combined model.

The first section will deal with the available hardware and the choice of the framework. It will be explained why the WGAN was chosen as basis model. In parallel with implementing a combined model, the evaluation was built to be able to judge the performance both qualitatively and quantitatively. That is why the next section deals with testing.
The novel combined model will be built in steps – adding conditional information as input first, then adding an Encoder, and finally adding a time convolution layer. One attempt that did not bring the desired results is to combine the losses of VAE and GAN in a way to learn both sharp images and have a distribution close to the Gaussian one. All this is explained in more detail in the last section, which is about the novel model named EncGAN.
Finally we will point out how the developed model can be adapted to different datasets.

4.1 Hardware and Framework

The hardware on which the computation was done is a computer running Ubuntu 16.04 which has two NVIDIA supported GPUs: GeForce GTX 1070 Ti and GeForce GT 1030. There is 13 GB RAM available for computing.

It was decided to use TensorFlow [36] as it is one of the most used frameworks in Neural Networks research. TensorFlow has the advantage of being widely in use since it is open source and actively developed, thus providing many functions. Additionally it offers CUDA support, so the programs can run fast on GPUs. There are existing GAN and VAE implementations in TensorFlow, one of which was used as a base for our model.
TensorFlow is especially used by researchers applying Machine Learning like Neural Networks. It supports defining CNNs in various languages, including python. After creating a session and defining the graph, actual values can be fed to the network. Many functions like crossentropy, SSIM and even whole layers already exist, which helps developing learning models from scratch.

4.2 Choice of model basis

The code of this work is based on the improved WGAN code from `https://github.com/igul222/improved_wgan_training`, published as [12]. This was chosen as basis as it is one of the state-of-the-art models and the loss formulation is claimed to be stable.

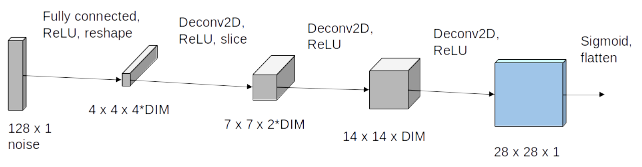
A GAN (not an VAE) was chosen to build upon because GANs are claimed to have a better output quality. Samples are supposed to look more realistic and much less blurred in comparison to the output of VAEs. The WGAN is one of the wider spread GAN networks and so it is expected to be more tested and reliable than, for example, the original GAN or the DC-GAN.
WGANs are stable in training, and they do not show the undesired mode collapse. We chose to work with the extension, the *improved WGAN*, which uses a gradient penalty. The equation 2.15 was used as can be seen in listing 4.1. The improved WGAN is claimed to be more stable and easier to train than the original WGAN.

Listing 4.1: Loss formulation of Wasserstein GAN with gradient penalty

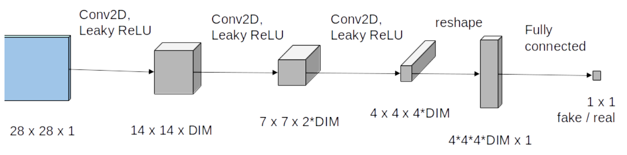
```
import tensorflow as tf

fake_data = Generator(BATCH_SIZE)
disc_real = Discriminator(real_data)
```

```
disc_fake = Discriminator(fake_data)

## Standard WGAN loss
gen_loss = -tf.reduce_mean(disc_fake)
disc_loss = tf.reduce_mean(disc_fake) - tf.reduce_mean(disc_real)

## Gradient penalty on Discriminator
alpha = tf.random_uniform(shape=[BATCH_SIZE,1], minval=0., maxval=1.)
interpolates = real_data + (alpha*(fake_data - real_data))
gradients = tf.gradients(Discriminator(interpolates),
    [interpolates])[0]
slopes = tf.sqrt(tf.reduce_sum(tf.square(gradients), 1))
gradient_penalty = tf.reduce_mean((slopes-1.)**2)
disc_loss += LAMBDA*gradient_penalty
```

The network structure of the plain GAN can be seen in figure 4.1.



(a) Generator G



(b) Discriminator D

Figure 4.1: Implemented GAN network. G consists of a fully connected layer in the beginning, followed by three convolutional layers. D consists of three convolutional layers and a fully connected layer at the end.

However, the improved WGAN code was completely refactored to only include the code parts needed (*wgan-gp*) and to use TensorFlows *contrib.layers*

instead of creating all weights manually, thus making the code shorter and more understandable. The way how to reuse the network weights was reimplemented with the TensorFlow *reuse* Flag in the corresponding layers, so no parameter dictionary was needed any longer.
The code was ported to python 3 from python 2 as well, because python 3 provides all recent standard library improvements, including NumPy improvements.

Tensorboard summary functions like (partial) losses and image outputs were added so the progress can be followed with Tensorboard as well. An automatic saving functionality was implemented and thus the possibility to stop learning and continue with the same weights later was enabled. Additionally to the existing MNIST loading and usage, the loading of moving MNIST was implemented. It includes six consecutive frames at a time to be able to give up to five past frames as conditional input to the model.

4.3 Testing

The Discriminator training loss and validation loss are calculated every 100 iterations and are plotted over the iterations. The Generator and Discriminator training losses are also written in a file so that it is possible to keep track of them. MSE, PSNR and SSIM are available in TensorFlow, so these functions were used for the evaluation in the scope of this work. The batch average value for all three is computed every 100 iterations, so the current image quality can be assessed along with the performance. This also allows to check if the network is learning what it is expected to learn. At the end of the training all PSNR, MSE and SSIM values of the last batch are saved so outliers can be identified.

In addition to that, the existing samples visualization was adapted to being able to show the condition – one or more past frames – and ground truth – the future frame from the dataset – next to the generated sample. Furthermore, an alternating gray background was added to distinguish the different samples more easily, of which a example can be seen in figure 5.1.

The code base also has an inception score implementation. However, it is not working with the newer TensorFlow version that was used for this thesis. As mentioned in section 2.5.2, the FID score computation is the most recent measurement to compare the quality of different GANs. The FID should only be used with more than 10000 training images, which are not available for all datasets, thus no FID evaluation was used.

To complete the evaluation measures, a classfier is trained to evaluate the accuracy of the generated images on MNIST. A simple neural network consisting of two layers with 800 hidden units is used, since it does not take long to train and already provides sufficiently good results on MNIST. According to the ranking of Yann LeCun on `http://yann.lecun.com/exdb/mnist` such a network should have about 2% error rate on the MNIST test set, so the classifier is trained until it reaches at least 98% accuracy. The classification of the generated digits is carried out at the end of the learning phase and the accuracy is saved.

4.4 Building up EncGAN

The goal is to stepwise build a bigger model that can predict future frames when past frames are provided as input, and that has elements of both the GAN and the VAE model. We will present the development in three steps, namely adding conditional information, adding an Encoder, and adding time convolution. In addition, we attempted to combine losses of a standard VAE and the improved WGAN to reach a more powerful loss, which did not give the expected results. This will be explained in more detail in the last section.

4.4.1 Adding conditional information

The first step was to provide additional conditional input to the WGAN. The aim was to provide one or more past frames in their original size as additional input to the networks, to make the model produce a future frame of exactly that scene. This is most easily done with a conditional GAN (cGAN), since according to cGAN, one can input any additional information to both Generator and Discriminator. So it was decided to input the last frame as conditional information to both networks. To keep it as simple as possible, concatenation was used for this conditional input. The noise is concatenated with the conditional frame as an input for the Generator; the generated or real frame respectively is concatenated with the conditional frame to create the input for the Discriminator.

All images, both regular input and conditional frames, are handed to the networks as flattened images. Some concatenations require to work on a $[batchsize, xdim, ydim, channel]$ shape, so a reshape can be done first.
$[batchsize, xdim \cdot ydim \cdot channel] \mapsto [batchsize, xdim, ydim, channel]$

If the conditional information are labels, they are concatenated with the noise at the Generator.
$[batchsize, noisedim] \mapsto [batchsize, noisedim + labeldim]$.

To add frames at the Generator in the cGAN, the flattened conditional images of size $(xdim \cdot ydim \cdot 1)$ are concatenated with the noise.
$[batchsize, noisedim] \mapsto [batchsize, noisedim + (xdim \cdot ydim \cdot 1)]$.

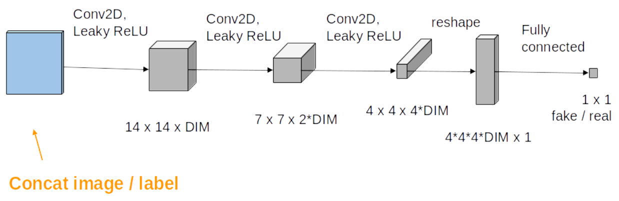
To add a frame as conditional information in the Discriminator, it is concatenated with the (real or fake) regular input image. Both frames are brought into the shape $[batchsize, xdim, ydim, 1]$. Therefore, for MNIST and moving MNIST, concatenation means to add a second dimension to the binary input image.
$[batchsize, xdim, ydim, 1] \mapsto [batchsize, xdim, ydim, 2]$.

With the help of labels from the MNIST dataset different forms and positions providing conditional information to the networks were tested. It was tested if providing the label accelerates the learning and if the network is able to produce the desired digit instead of a random one.
Providing the labels as digits or as one-hot vector did not make any difference in speed or accuracy. [9] suggests to provide the condition at a later layer. However, when trying to provide the corresponding label to the Discriminator before the fully connected layer, as seen in figure 4.2(b), it led to the model not learning to output the corresponding digit, but any digit similar to MNIST. Therefore giving the conditional information to

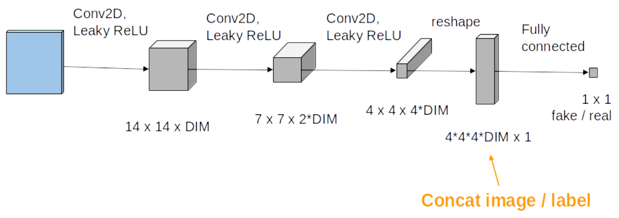
the Discriminator in the beginning, as depicted in [27], was chosen for the model of this thesis.

The Discriminator was found to learn faster when the conditional information is provided more often. Providing only a one-hot vector (10 entries) did not work well, but repeating the label information, e.g., as a second dimension of the input image or as a padding to create a bigger image (from 28x28 to 32x32 = 240 entries), accelerates the learning.

So, at the Discriminator, the 10-dimensional label vector that represents the digit is repeated 24 times. It thus forms a 240-dimensional vector that is concatenated to the flattened 28x28 MNIST input image of the Discriminator, resulting in a 32x32x1 matrix after reshaping.

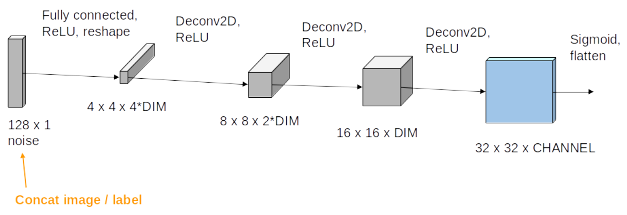


(a) Condition in the beginning

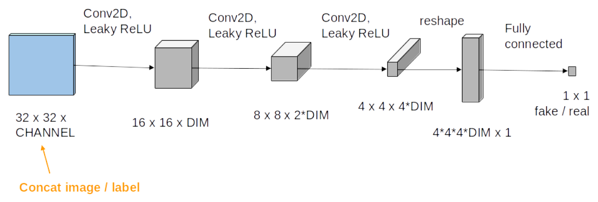


(b) Condition at the end

Figure 4.2: cGAN structure. Discriminator *D* still consists of three convolutional layers and a fully connected layer. It can get conditional information in the input layer (a). It was tried to give conditional information later, before the fully connected layer (b), but this lead to the GAN not learning the correct labels anymore.



(a) Generator G



(b) Discriminator D

Figure 4.3: The cGAN model for images of size 32x32. G consists of a fully connected layer followed by three convolutional layers. D consists of three convolutional layers and a fully connected layer at the end. The conditional information is concatenated at the input layer of both G and D.

$$[batchsize, 28 \cdot 28 \cdot 1] \mapsto [batchsize, 28 \cdot 28 \cdot 1 + 240] \mapsto [batchsize, 32, 32, 1].$$

The amount of conditional information provided at the Generator – a 10-dimensional one-hot vector vs. a 60-dimensional repeated label vector – did not make any difference, which causes us to claim that it is most important that the Discriminator gets enough information at the right position. If the Discriminator learns fast to distinguish the generated wrong digits from real ones, it will lead to the Generator incorporating that information as well.
To test the conditional GAN, a variant was developed that shows five images of every MNIST digit to have a balance over all digits and an overview of how far the learning has proceeded. The complete cGAN model with all its layers is depicted in figure 4.3.

4.4.2 Adding an Encoder

The second step is to include an *Encoder*, which will also be a network consisting of convolutional layers. The Encoder will be implemented with a layered structure similar to the Discriminator, mirroring the Generator. The input is for now one image from the dataset, with the output of the Encoder being provided to the Generator.
As explained in section 2.2.1, in the VAE the Decoder does an additional sampling operation, leading to equation 2.11. The Encoders output is a vector of means μ and a vector of standard deviations σ and therefore the (noisy) z input for the Generator can be sampled from these Encoder outputs. To ensure that the standard deviation σ that the Encoder provides is positive we take $e^{\frac{\sigma}{2}}$ instead.
The whole model *Encoder - Generator - Discriminator* still needs conditional input for the Discriminator. When providing the code computed by the Encoder or the sampled z to the Discriminator, the model was not learning properly. Providing the repeated label information to the Discriminator like in the conditional GAN made it work. The developed model will be called EncGAN from now on. Its rough structure can be seen in figure 4.4, and the detailed layout of the layers in figure 4.5.

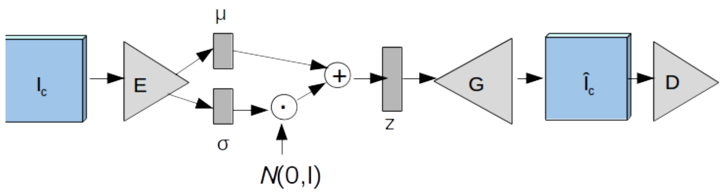
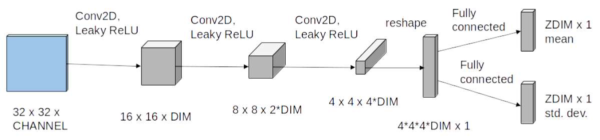
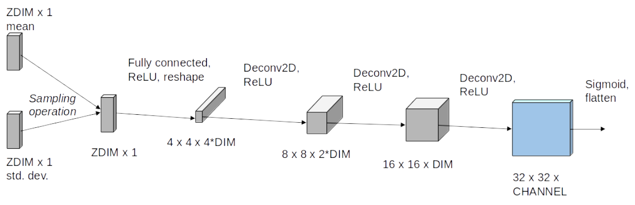


Figure 4.4: Structure of developed EncGAN. The model is a combination of an Encoder, the sampling of VAE and the 2-player game of GAN. It uses the gradient penalty GAN loss. For the Generator loss the weights of the Encoder and the Generator are trained together end-to-end.

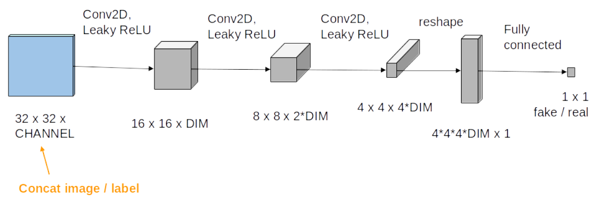
With the Encoder in place, the GAN model can be tested against an analogous VAE. The VAE is the same model without the Discriminator but using the VAE loss from equations 2.7, 2.8 and 2.10 instead. It thus compares the



(a) Encoder *Enc*



(b) Generator *G*



(c) Discriminator *D*

Figure 4.5: The EncGAN model for images of size 32x32. *Enc* consists of three convolutional layers and two fully connected layers at the end, producing μ and σ vectors. G samples its input from μ and σ and consists of a fully connected layer followed by three convolutional layers. D comprises three convolutional layers and a fully connected layer at the end. D gets conditional information concatenated at its input layer.

input frame provided to the Encoder with the output given by the Generator. The corresponding code in TensorFlow can be seen in listing 4.2.

Listing 4.2: Loss formulation of VAE

```
import tensorflow as tf

mean_data, sd_data = Encoder(real_data)
fake_data = Generator(BATCH_SIZE, mean=mean_data, sd=sd_data)

## Standard VAE loss
# reconstruction loss: pixel-wise L2 loss # MSE(fake - real)
img_loss = tf.reduce_sum(tf.squared_difference(fake_data,
   real_data), 1)

# latent loss: KL(latent code, unit Gaussian) # closed form
latent_loss = -0.5 * tf.reduce_mean(1. + sd_data -
   tf.square(mean_data) - tf.exp(sd_data), 1)

vae_loss = tf.reduce_mean(img_loss + latent_loss)
```

Moreover, a VAE+ model can be tested, predicting a future frame given one past frame. The frame prediction can be set up by calculating the VAE loss between the predicted frame and the ground truth next frame instead of between the predicted frame and the provided input frame.

4.4.3 Adding time convolution

The last step is to add more frames as input. Different methods were considered of how to bundle the information, using convTime without or with feature convolutions. It was chosen to implement one convTime layer, not multiple layers after each other. The place to put the time convolution layer was chosen to be the same as in [23]. Figure 4.6 shows the placement of the convTime layer in the model. Three different convTime layers V1, V2 and V3 were built into the EncGAN one by one to see if and how much the results can be improved by any of them. The number of frames given to the model was kept variable. Experiments were done with two, three and five past frames.

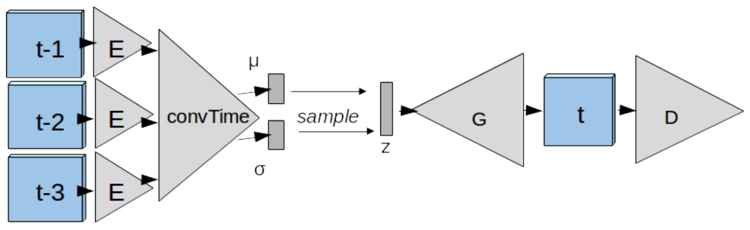


Figure 4.6: Structure of EncGAN with convTime. The Encoder is split in two parts, first encoding each input frame separately, but with shared weights. Here it is shown with three past input frame as an example. The second part is the convTime layer, plus the fully connected layers to compute mean and standard deviation.

4.4.4 Attempt to combine losses

It was tried not only to combine the structures of VAE and GAN, but also to have a combined loss or at least parts of both losses to get the benefits of both approaches. Different combinations were tried, e.g., to use the Discriminator loss with the VAE loss $\mathcal{L}_{VAE}$, to use the Generator loss with the $\mathcal{L}_{VAE}$, or to use the Generator loss with the MSE part of the VAE loss ($\mathcal{L}_{rec}$).

$$\mathcal{L}_{combi1} = c_1 \cdot \text{Discriminator loss} + c_2 \cdot \mathcal{L}_{VAE}$$

$$\mathcal{L}_{combi2} = c_1 \cdot \text{Generator loss} + c_2 \cdot \mathcal{L}_{VAE}$$

$$\mathcal{L}_{combi3} = c_1 \cdot \text{Generator loss} + c_2 \cdot \mathcal{L}_{rec}$$

The constants c_1 and c_2 were chosen to amplify the Discriminator or Generator loss respectively, because these losses are much smaller compared to the VAE loss or the reconstruction loss. It was also tried to gradually increase the influence of the Generator loss over the iterations, since it would be plausible that it is easier to first learn to be close to the original image with the help of the VAE loss and only later use the adversarial learning to improve the image quality. Finally it was tried to optimize the combined loss either with both Encoder and Generator parameters at the same time or to optimize the parameters of Encoder and Generator after each other.
None of these attempts led to a recognizable digit output. The samples show blurry blobs that average over all digits. That is why the idea to combine the

losses in such a way was discarded and only the WGAN loss with gradient penalty was used for the EncGAN.

4.5 Challenges with different datasets

Since MNIST is quite easy to learn with such a large architecture, another dataset was tested as a continuation – *moving MNIST*. As presented in section 2.3, the images of moving MNIST are 64x64 pixels, which meant an additional challenge to generate bigger images. Thus, the first idea was to insert an additional layer in Encoder, Generator and Discriminator. This bigger architecture, however, proved to only work with the GAN model, namely with the standard WGAN and the conditional version of it.
Those two models need much longer training to produce acceptable results than on MNIST, and the finally obtained images do not look as good. The results could not be improved by more training iterations. For the VAE and the EncGAN the input images were downsampled to 32x32 pixels, so they work as before, without additional layers.

To ensure the model is working properly with a new dataset, first GAN and VAE should be tested to produce similar images to the dataset. After that, the frame prediction can be tested. One past frame can be given to the cGAN and the EncGAN as condition and they predict the future frame from this single input image. It is evaluated by giving the Discriminator the next frame of the dataset to compare the plausibility.
Since the classifier only works on MNIST – it can only classify one digit – and in moving MNIST the two digits often overlap, it does not make sense anymore to calculate the digit accuracy. However, PSNR, MSE and SSIM can be evaluated on any dataset.

5 Evaluation

Having the conditional models cGAN and EncGAN and the baseline networks GAN and VAE, the question is how much better the new ones perform. Since GAN and VAE share network parts, and are also evaluated on the same datasets, direct comparison is possible. This is done in the form of samples as well as the quantitative results. The quantitative results are shown in the tables 5.1, 5.2 and 5.3.

In the tables the different evaluation metrics are listed. SSIM, MSE and PSNR are measured in comparison to the input image taken from the test set or to the next frame taken from the dataset respectively. To revise, SSIM goes from -1 to 1, the value 1 indicating two identical images. MSE is positive but the lower the better, and PSNR is positive but the higher the better.
The MNIST table also includes inference time which means the time it takes to generate 50 samples. It is measured in seconds.
For GANs the loss column in the tables show the Discriminator validation loss which indicates how well the Discriminator has learned to distinguish between real and fake images from the validation set. For VAE the loss column represents the VAE loss as defined in section 2.2.1.
The classification score in the MNIST table represents how many digits are correctly identified as corresponding to their input label by the MNIST classifier.
For moving MNIST the subjective visual appearance of the samples was added to the tables to have a better overview of the performance without having to compare to the figures all the time.
The results printed in bold indicate the best value for that metric.

To be able to compare the results to other papers, some more parameter, which stayed fixed for the experiments, are listed here. As suggested in the improved WGAN paper [12], the gradient penalty parameter λ is set to 10 and in every iteration one Generator and five Discriminator updates are done. The learning rate for the Adam optimizers was 0.0002 for all presented results. Other learning rates were tried but did not give better results.

MNIST is split in three parts: 50000 images training dataset, 10000 images validation dataset and 10000 images test dataset. The batch size was chosen to be 50. This means that for MNIST one epoch comprises 1000 iterations. The dimension of the feature channel, which is called DIM in the figures in section 4, is set to 10. The dimension of noise or sampled z vector respectively was also chosen to be 10 for MNIST. This is sufficient to learn the digits features.

The Moving MNIST dataset of 10000 images was also split in three parts: 7000 images training dataset, 2000 images validation dataset and 1000 images test dataset. The batch size was again chosen to be 50. This means that for moving MNIST one epoch comprises 140 iterations. The dimension of the feature channel DIM is empirically set to 64. The dimension of noise or sampled z vector respectively was chosen to be 60 for moving MNIST.

5.1 Comparison VAE and GAN on MNIST

We will now compare the VAE against the improved WGAN, and also assess the developed models cGAN and EncGAN. First, let us see the performance of the standard models on the MNIST dataset. It can be seen in figure 5.1 how good the VAE and the WGAN are at generating samples from noise. The VAE samples are blurrier, whereas the GAN generates sharper but partly incomplete digits. The cGAN learns to produce the asked digits quite well. The image quality difference between VAE and GAN is claimed to be a consequence of using an MSE loss for the VAE loss directly. Incomplete digits are claimed to be a consequence of the latent space not being covered tightly.

How good the model is at reconstructing the input image (VAE), using conditional information like labels (cGAN) or input images and labels (EncGAN) can be seen in figure 5.2. It shows that the VAE is able to reconstruct MNIST images very well. The cGAN is able to generate sharp digits corresponding to the label it received. Since its Generator input is a label and not more information on the original image is provided, the output image does not correlate with the image it is compared to at test time. The left image is thus just there to show which digit the label belonged to. On the contrary, the EncGAN learns to produce the shown digit, and does reconstruct the input image, but not as much as the VAE does. The EncGAN generates the same digit, but generalizes more to also remove or include features seen at other images of that digit, e.g., it leaves out the additional line of a seven.

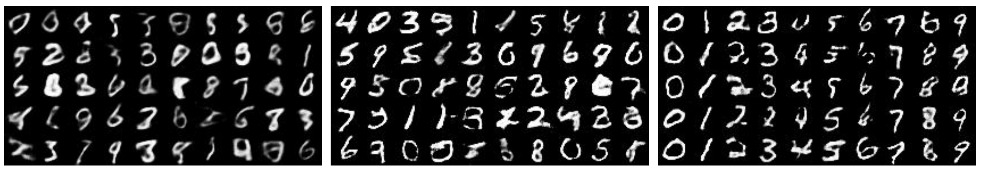

(a) VAE (b) GAN (c) cGAN (ordered labels)

Figure 5.1: Samples produced by noise input for VAE, noise input for GAN, and ordered label information for cGAN. The networks were trained on the MNIST dataset for 5000 iterations.

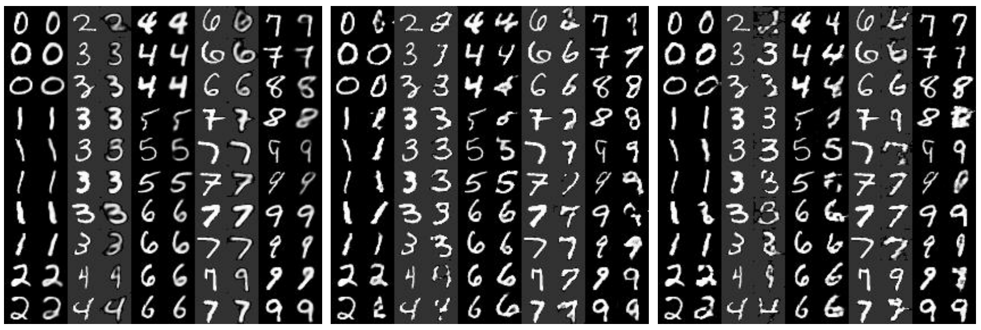

(a) VAE recon. (b) cGAN (c) EncGAN

Figure 5.2: Samples of the VAE reconstructing input images, the cGAN generating images corresponding to the provided labels, and the EncGAN inferring the label information from the input image. Trained on the MNIST test dataset for 5000 iterations.

What can be read from the table 5.1 is that all architectures have similar inference times on MNIST. As a side note, the training time is different, since for instance for the VAE only one update step is performed per iteration, whereas for all GAN variants there are one Generator and five Discriminator updates per iteration. It is clear that VAE gives the best image similarity scores in contrast to the GANs, because it tries to reconstruct the given image and thus aims to produce the nearest approximation. That also explains why the MNIST classifier shows the highest accuracy for the VAE, since the output image is a reconstruction of a real MNIST image and not completely newly generated. In contrast to that, the cGAN, which does not even see the image, just the label, has the lowest image similarity scores, but a higher classification accuracy than EncGAN, showing that the labels given as condition are learned and used well. Finally, EncGAN, making use of the image input, gives better similarity (SSIM) values and also better MSE and PSNR values than cGAN. However, the values are still far from the results of the VAE. This is because the EncGAN loss function is not aiming at reconstructing the input, but at fooling the Discriminator to believe the generated samples are taken from the dataset.

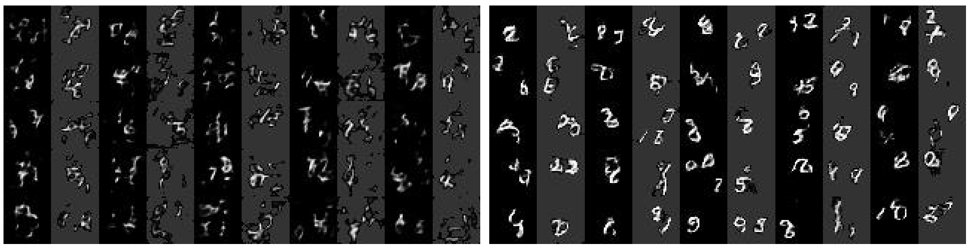
	inf. time	loss	SSIM	MSE	PSNR	classif.
GAN	**0.0004 sec**	-0.5818	n.a.	n.a.	n.a.	n.a.
cGAN	0.0005 sec	-0.1567	0.2082	0.1048	9.9233	84%
EncGAN	0.0007 sec	-0.3862	0.3718	0.0777	11.5734	82%
VAE recon.	0.0008 sec	15.4637	**0.8053**	**0.0153**	**19.0292**	**92%**

Table 5.1: Results on MNIST

5.2 Comparison VAE and GAN on moving MNIST

We are now moving on to see the performance of the different models on the moving MNIST dataset.

It is visible in figure 5.3 that the GAN performs much better in image generation from noise. The VAE samples are very blurry, the GAN generates sharper but partly incomplete digits. As mentioned above, the bad



(a) VAE (b) GAN

Figure 5.3: Sampling from the VAE and the GAN is done by providing Gaussian noise as input. Trained on the moving MNIST dataset for 10000 iterations.

performance of the VAE in comparison to the GAN is believed to stem from the VAE latent space not being covered tightly.

However, as demonstrated in figure 5.4, the VAE can reconstruct given input images well. The VAE samples are still a bit blurry, but the reconstruction looks much better than the image generation from noise.

The performance of using a past image as conditional information to predict the future frame (VAE+, cGAN, EncGAN) can be seen in figure 5.5. Left of the predicted frame (*pred*) the conditional past frame (*t*) is shown, and to the right of the predicted frame the ground truth future frame (*t+1*) from the dataset is displayed. It is visible that the VAE+ does not lead to the desired result, since it is not able to produce sharp digits. The VAE+ just generates white blobs at the location of the digits from the previous frame. With these, it still manages to obtain the best MSE and PSNR values, as can be seen in table 5.2. However, it is obvious that the VAE is good at reconstructing images, but not good at predicting the future. It is thus not used for further experiments and not recommended to use for frame prediction in any case.

None of the networks in figure 5.5 is able to produce completely correct digits for all inputs. Comparing cGAN and EncGAN, the EncGAN produces slightly sharper images. In table 5.2 we see that the SSIM value of cGAN is higher, whereas MSE and PSNR of EncGAN are better.

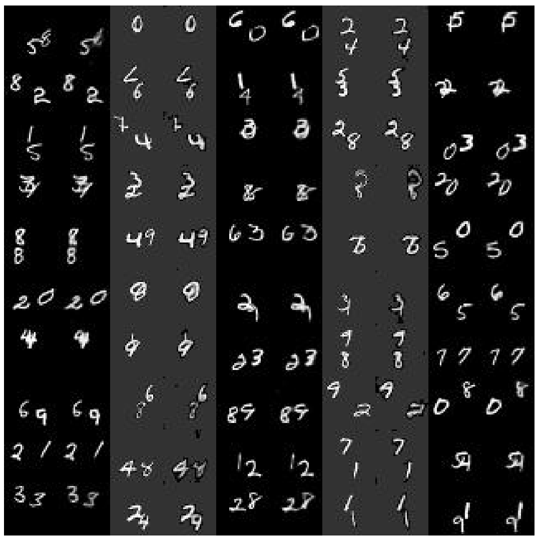


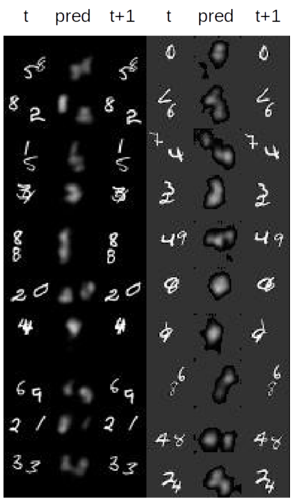
Figure 5.4: An example output of VAE with the goal to *reconstruct* the input images from the 32x32 moving MNIST test dataset. The model was trained for 50000 iterations.

	loss	SSIM	MSE	PSNR	visual appearance
VAE recon.	6.2946	0.5179	0.0316	15.2287	+
VAE+	21.1785	0.5041	**0.0199**	**17.1428**	–
cGAN	-1.2369	0.5127	0.0338	14.8944	+
EncGAN	-1.5284	**0.5329**	0.0312	15.2344	**++**

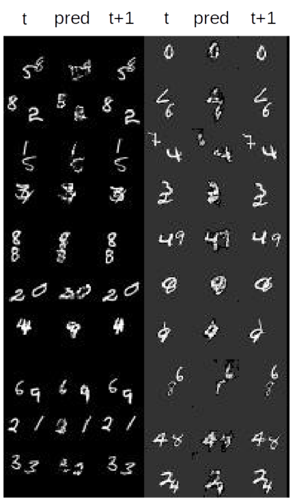
Table 5.2: Results on moving MNIST

The EncGAN plots in figure 5.6 show that the different metrics improve a lot in the beginning, and are quite stable in the later iterations, indicating convergence of the learning. It is shown that SSIM, MSE and PSNR improve in a similar manner, so they correlate.

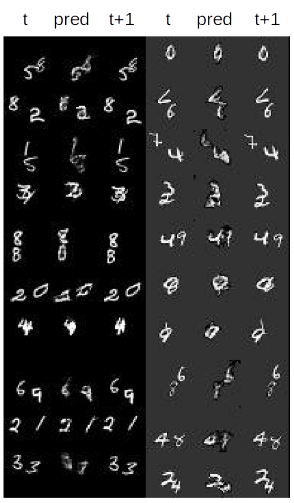
In figure 5.7 the training and validation loss for the EncGAN are shown. Both losses are from the Discriminator, but the training loss is evaluated on the training dataset and the validation loss is computed every 100 iterations on a validation dataset. The losses show a huge improvement in the first thousand iterations, whereas the values oscillate around a stable value later.



(a) VAE+

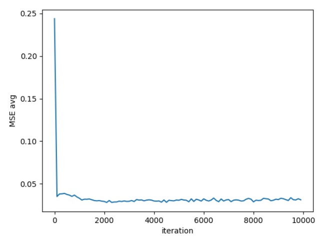


(b) cGAN

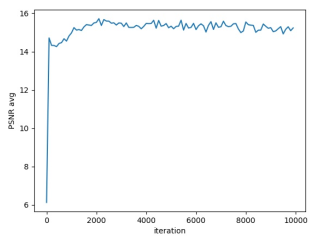


(c) EncGAN

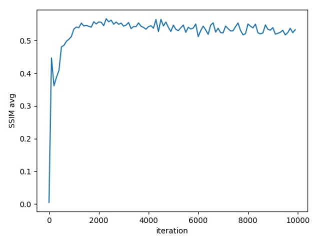
Figure 5.5: An example output of VAE+, cGAN and EncGAN trained on the 32x32 moving MNIST test dataset for 10000 iterations.



(a) MSE



(b) PSNR

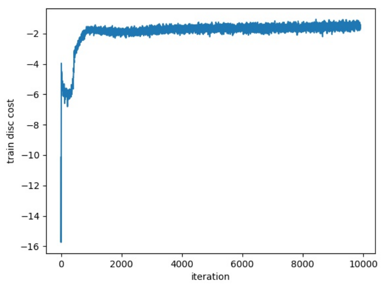


(c) SSIM

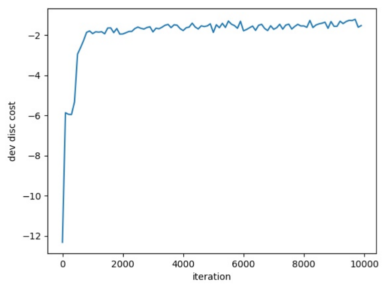
Figure 5.6: Tracking the learning of the EncGAN with one past input frame on the moving MNIST test dataset over 10000 iterations. The batch averages of MSE, PSNR and SSIM are computed every 100 iterations.

This demonstrates that the network has learned the critical features in the beginning and can then only achieve minor improvements.

Furthermore, looking at the produced samples of cGAN and EncGAN over



(a) D training loss



(b) D validation loss

Figure 5.7: Tracking the loss of the EncGAN with one past input frame on the moving MNIST test dataset over 10000 iterations.

the course of hundreds of iterations, it is visible that the EncGAN varies the direction of the movement of the digits more than the cGAN. This demonstrates the multimodality of the problem and the EncGANs ability to generate diverse images.

It is also worth to mention that the GAN and cGAN do work with the full resolution of 64x64. For that, an additional convolutional layer is added in both Generator and Discriminator. The results can be seen in the appendix. In contrast to that, the VAE and the EncGAN were both not able to generate digits in the 64x64 setting for moving MNIST. It is assumed that the cause is the Encoder, and that if a certain capacity is reached, the learning is not working properly anymore. This is backed up by the VAE not being able to learn to reconstruct RGB color images of size 32x32x3 from another dataset, also exceeding the capacity with the color channels.

5.3 EncGAN with more time frames as input

To include more past time frames, the EncGAN was chosen as the baseline. When giving more than one past frame as conditional input, there are two different questions that need to be answered:

1. How many past frames should be given as conditional input?
2. Which time processing layer should be used?

To find a solution to these problems, two comparisons are shown in this section. The first one compares the output and metrics of a different number of input frames, namely 2, 3 or 5 past frames as condition. The results can be observed in figure 5.8 and in table 5.3.

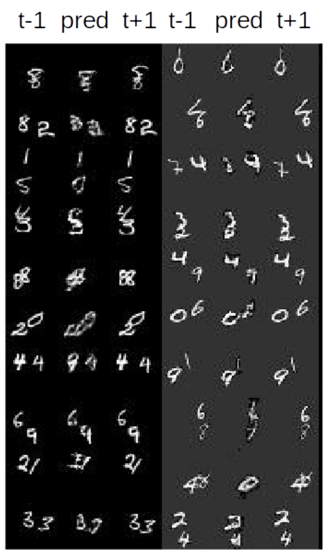
In the figure, the two most recent past frames (*t-1, t*) are shown left of the predicted image (*pred*) for all experiments with more than one past frame as conditional input, whereas for the single frame as input only the single past frame (*t*) is displayed.

Visibly, three frames give the best output image, whereas the quantitive results point out that two past input frames are the best choice for the frame prediction. Two input frames have a higher SSIM, a lower MSE score and a higher PSNR value than three or five input frames, see table 5.3. All experiments except 2 input frames result in worse image quality values than the baseline that gets only a single frame. The measurements concerning 1 frame are slightly different than the ones shown in section 5.2 since the evaluation was done on other test frames than before. In general, the model outputs are quite blurry images of the bouncing numbers, and none of the models predict the future frame as well as expected.

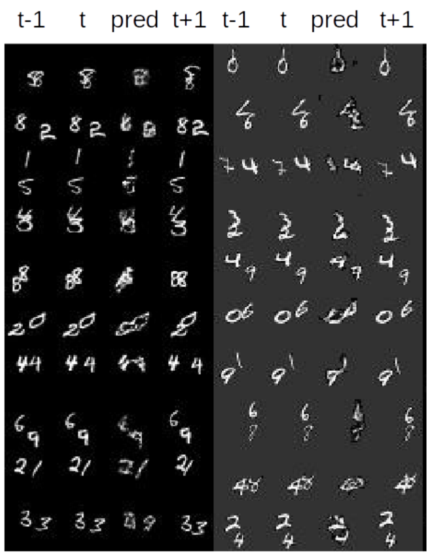
	loss	SSIM	MSE	PSNR	visual appearance
1 frame	-1.5007	0.5470	0.0324	15.1242	∼
2 frames V1	-1.9290	**0.5867**	**0.02779**	**15.8534**	+
3 frames V1	-1.0653	0.5243	0.0333	15.0003	++
5 frames V1	-2.9740	0.4522	0.0373	14.5094	∼
3 frames V1	-1.0653	0.5243	0.0333	**15.0003**	++
3 frames V2	-1.0577	**0.5246**	**0.0332**	14.9874	++
3 frames V3	-1.1062	0.5210	0.0337	14.9106	++

Table 5.3: Results on moving MNIST with time series input

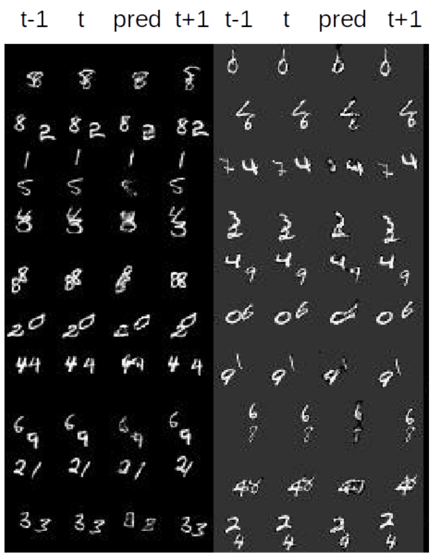
To see the effect of the different convTime versions, for the following experiments the number of past frames was fixed. Three past frames are each passed through the Encoder independently and thus provided to the



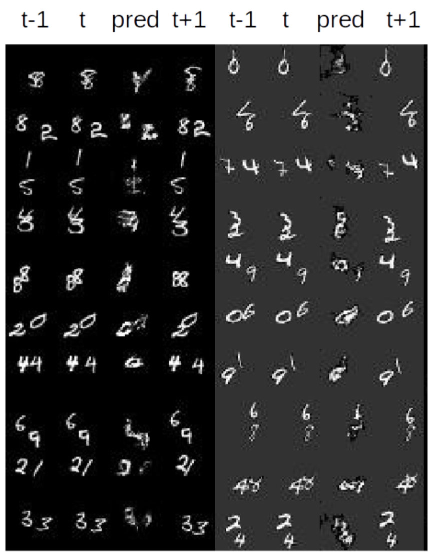
(a) 1 frame



(b) 2 frames



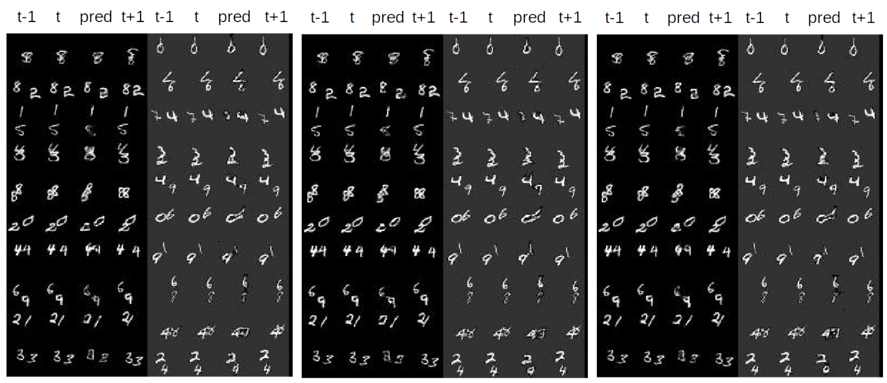
(c) 3 frames



(d) 5 frames

Figure 5.8: The EncGAN architecture is now equipped with an additional convTime layer, getting 2, 3, or 5 past frames as input. In the top left corner is the baseline that receives only a single input frame. Trained on moving MNIST for 10000 iterations.

convTime layer. The Discriminator gets only one past frame as conditional information to know which digits are involved in the sequence.

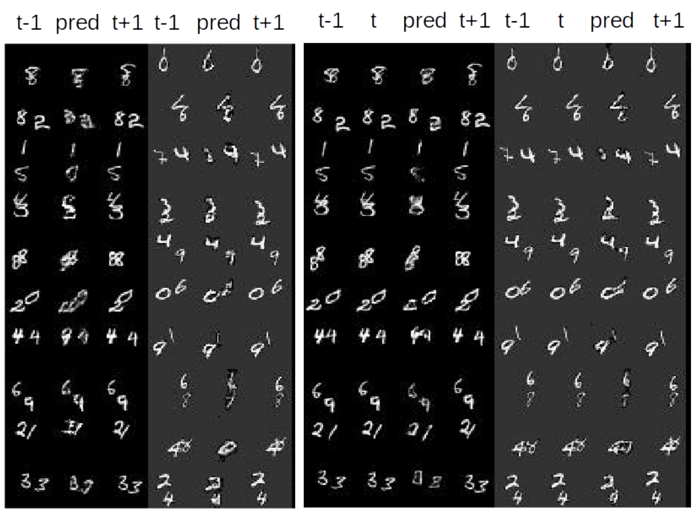

t-1 t pred t+1 t-1 t pred t+1
t-1 t pred t+1 t-1 t pred t+1
t-1 t pred t+1 t-1 t pred t+1

(a) V1 (b) V2 (c) V3

Figure 5.9: Each of the three convTime layers is added to the EncGAN. Trained on moving MNIST for 10000 iterations with 3 past frames.

In figure 5.9 you can see the results of the different convTime versions, and in table 5.3 there is the information on the convTime versions for three past input frames (3 frames V1/V2/V3). It is striking that the feature space convolution in connection with the time convolution (V3) does not help, the samples from V3 do not look better than from the first two versions in figure 5.9. Table 5.3 shows that all the tested convTime versions have very similar image similarity values, namely a SSIM score of 0.52 and a MSE value of 0.03. V1 reaches a better PSNR value of 15.0 compared to v3 with 14.9. This leads us to the conclusion that the computational overhead of including the feature space in the convolutions is not needed.

To see the difference between using more past frames and using only a single last frame, a direct comparison can be found in figure 5.10. The network with 3 past frames as conditional input and V1 convTime layer is used as the best result and compared to the baseline EncGAN with a single conditional input image. The convTime layer makes the network output sharper and produces more complete digits, so it can be seen as an



(a) EncGAN (b) with convTime

Figure 5.10: The left model got 1 past frame, the right model got 3 past frames and is equipped with the V1 convTime layer. Both networks are trained for 10000 iterations.

improvement. It indicates that convTime does give a benefit to the EncGAN and should be used to get better results.

6 Conclusion

This thesis is dealing with how to use a combination of an VAE and a GAN model for frame prediction. Firstly, the results will be summarized and after that possible future work will be pointed out.

6.1 Summary

A comparison between VAE and GAN was made, confirming literature that VAEs are best at reconstructing the input, whereas GANs can generate new sharp images with a similar style like the dataset provided to them. The EncGAN model was developed by mixing the two generative models, or more precisely by expanding a GAN by an Encoder before the Generator. It was shown that this model gives sharp results as well, but its output images are also more similar to the dataset than with the regular GAN. It was pointed out that a generative model can be made to learn to produce specific images – for instance including a certain digit or continuing the motion of the last frames – by adding conditional input to the networks. Using the VAE for frame prediction (VAE+) did not give satisfactory results. The VAE model alone is not able to predict a sharp future frame. The Encoder as a part of the VAE and the EncGAN also seems to be limiting the image size, since using 32x32x3 or 64x64x1 images let the models stop learning without having produced corresponding images. The benefit of time series input, i.e., including more than one past frame, was investigated and it turns out that it helps to generate a future frame in terms of visual image quality.

Different datasets were applied to see how image generation works, how conditional information can be made usable for the networks and how much the condition helps the learning. An architecture similar to the paper [23] was reimplemented. Our model has a lower frame resolution and adds conditional input to the Discriminator. It is unclear why the authors of the dual model decided to only provide the produced frame or flow to the Discriminator without any conditional information, making it impossible for the Frame and Flow Discriminator to differentiate between a plausible *future* frame/flow or just any plausible frame/flow similar to the dataset.

6.2 Future work

For future research it would be interesting to test the model on a real-world video dataset, since the chosen moving MNIST dataset is synthesized. It is expected that the frame prediction works similarly as in the presented experiments. Additionally, the image quality for moving MNIST is not as good as expected, thus it would be interesting to tune the networks more to the used datasets to achieve better quality.

The dual architecture as described in paper [23] can be investigated further, for instance how the authors manage to use a high frame resolution and if a convLSTM layer gives better results for the EncGAN than our convTime layer. It would be interesting to see if Optical Flow prediction is superior to Frame prediction and if the dual model using both predictions to improve on each other gives the best results as is claimed in the paper.

In general, a higher image resolution would be desirable, since it was not reachable in the experiments done in the scope of this thesis. As mentioned in the related works, there are some methods to gradually get to a higher end resolution, e.g., progressively growing layers [1] or pyramidic upscaling [5].

All in all, this work showed promising results with a conditional GAN and an EncGAN model and it also demonstrates that there are still many variants left to explore.

Bibliography

[1] Sandra Aigner and Marco Körner. Futuregan: Anticipating the future frames of video sequences using spatio-temporal 3d convolutions in progressively growing gans. 10 2018.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[3] Ali Borji. Pros and cons of gan evaluation measures. *arXiv preprint arXiv:1802.03446*, 2018.

[4] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

[5] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[6] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[7] Christoph Feichtenhofer, Axel Pinz, and Richard Wildes. Spatiotemporal residual networks for video action recognition. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3468–3476. Curran Associates, Inc., 2016.

[8] Christoph Feichtenhofer, Axel Pinz, and Richard P. Wildes. Temporal residual networks for dynamic scene recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[9] Jon Gauthier. Conditional generative adversarial nets for convolutional face generation. 2015.

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

[11] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[12] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5767–5777. Curran Associates, Inc., 2017.

[13] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[15] Xianxu Hou and Guoping Qiu. Improving variational autoencoder with deep feature consistent and generative adversarial training. 2018.

[16] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[18] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[19] Wei-Sheng Lai, Jia-Bin Huang, and Ming-Hsuan Yang. Semi-supervised learning for optical flow with generative adversarial networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 354–364. Curran Associates, Inc., 2017.

[20] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

[21] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[22] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[23] Xiaodan Liang, Lisa Lee, Wei Dai, and Eric P Xing. Dual motion gan for future-flow embedded video prediction. *arXiv preprint*, 2017.

[24] Alireza Makhzani and Brendan J. Frey. k-sparse autoencoders. *CoRR*, abs/1312.5663, 2013.

[25] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[26] Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *CoRR*, abs/1511.05440, 2015.

[27] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[28] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. In *Advances in neural information processing systems*, pages 2352–2360, 2016.

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

[30] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1278–II–1286. JMLR.org, 2014.

[31] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[32] Mihaela Rosca, Balaji Lakshminarayanan, David Warde-Farley, and Shakir Mohamed. Variational approaches for auto-encoding generative adversarial networks. *arXiv preprint arXiv:1706.04987*, 2017.

[33] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[34] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015.

[35] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pages 3738–3746, 2016.

[36] TensorFlow. TensorFlow description on homepage, 2018.

[37] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[38] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from variational autoencoders. In *European Conference on Computer Vision*, 2016.

[39] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[40] Xiang Wei, Zixia Liu, Liqiang Wang, and Boqing Gong. Improving the improved training of wasserstein GANs. In *International Conference on Learning Representations*, 2018.

[41] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016.

[42] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, and Dimitris N. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *CoRR*, abs/1612.03242, 2016.

Appendix

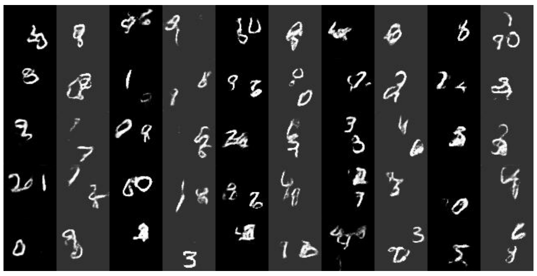


Figure 6.1: Samples of a GAN trained on the full resolution (64x64) moving MNIST test dataset for 10000 iterations.

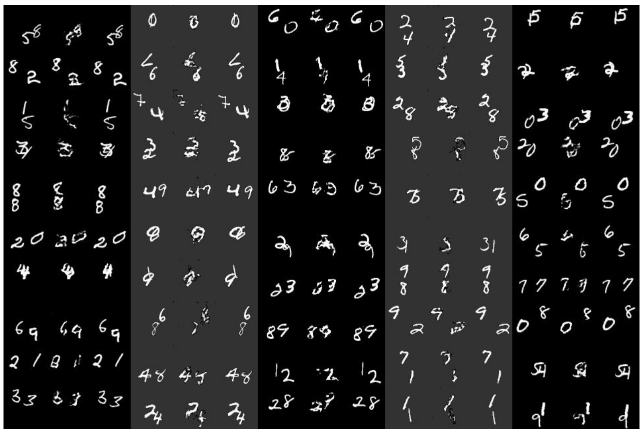


Figure 6.2: Samples of a cGAN trained on the full resolution (64x64) moving MNIST test dataset for 50000 iterations. The cGAN output (middle) is trained to be a prediction of the next frame (right) given the past frame (left).