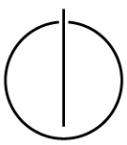
DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics: Games Engineering

TricycleGAN: A Temporally Consistent CycleGAN for Unpaired Video-to-Video Translation

Jonas Mayer



DEPARTMENT OF INFORMATICS

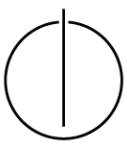
TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics: Games Engineering

TricycleGAN: A Temporally Consistent CycleGAN for Unpaired Video-to-Video Translation

TricycleGAN: Ein temporär konsistenter CycleGAN für ungepaarte Video-zu-Video-Übersetzung

Author:	Jonas Mayer
Supervisor:	Prof. Dr. Nils Thuerey
Advisor:	Mengyu Chu
Submission Date:	16.08.2019



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

Garching, 16.08.2019 Jonas Mayer

Acknowledgements

I'd like to thank Rachel aka Mengyu Chu and Prof. Dr. Nils Thuerey for going the extra mile to provide no less than excellent supervision throughout my time working on the thesis. Their constant support was what made all of this possible and motivated me to do my part in delivering what I hope to be more than a solid conclusion to my master's studies.

I'd also like to give a big shout out to the amazing people that i had the pleasure of working with at TNG Technology Consulting, above all Thomas Endres. Not only for keeping me fed over the last year, but also for providing an amazing learning environment that sparked the idea for this thesis.

Big ups to Dr. Nicholas Katzakis, my former bachelor thesis supervisor, for kicking my arse all the way to Japan where i had the honour of presenting my first publication. This thesis would probably be an even rougher read if it wasn't for what I learned from you.

Finally, I'd like to thank all the positive influences in my life for proof reading this thesis, improving my English skills, keeping up my motivation or even just distracting me for a change. Same goes for the bad influences for ultimately pushing me towards the positive ones.

Abstract

In this master's thesis, we propose the tricycleGAN, a neural network model for video-to-video translation using unpaired data. Based on a proven model for unpaired image-to-image translation, it uses frame-recurrent generators, an adversarial temporal loss as well as Ping-Pong loss to ensure temporal consistency in the output video. In addition, a novel discriminator style loss is proposed that significantly speeds up generator convergence. The theoretical fundamentals behind the final model are described and the main implementation considerations and challenges are documented. Finally, the strength of the tricycleGAN is demonstrated on different datasets and the importance of its individual components is highlighted in a ablation study.

Contents

1. Introduction

Image-to-image translation describes the process of transferring an input image from one visual domain to another. Using neural networks for this task, remarkable results have been achieved in many different fields like style transfer, image segmentation and object translation [11, 38, 39, 16, 15, 44]. While these techniques can also be applied to videos frame by frame, this approach completely ignores temporal relationships between the individual frames. A commonly observed consequence is temporal inconsistency in the form of flickering artefacts.

Traditional video-to-video solutions achieve temporal consistency by introducing hand-crafted temporal losses to the training objective. These often use optical flows to compensate for motion and aim to minimise a pixel-wise difference between neighbouring frames. However, these hand-crafted losses often suffer from being oversimplified approximations of temporal consistency and are notorious for sacrificing sharpness in the outputs for temporal consistency. With the rise of generative adversarial networks, a new line of research focuses on achieving temporal consistency with versatile adversarial losses instead.

One major problem in video-to-video translation is the scarcity of useful training data. Good video datasets are hard to come by and those that exist are mostly limited in data quality and quantity. This is especially true for paired video data: Obtaining matching input-output pairs often requires extensive human supervision or is simply impossible due to the nature of the problem. Thus, being able to learn a valid domain mapping from unpaired training data is a highly desirable property. While the cycleGAN [44] solves this problem using cycle consistency, it provides no mechanisms for temporal consistency.

In this thesis, we propose the tricycleGAN, a recurrently trained cycleGAN that learns short- and long-term temporal consistency from a third temporal discriminator and a Ping-Pong loss, as a solution for unpaired video-to-video translation.

Our central contributions can be summarised as:

- A temporally consistent cycleGAN with frame-recurrent generators and a shared temporal discriminator.
- A novel discriminator style loss to speed up training convergence.

2. Related Work

2.1. Image-to-Image Translation

2.1.1. Neural Style Transfer

Neural Style Transfer as originally proposed by Gatys et al. [11] translates real-world photographs into artistic paintings. Several papers [16, 38, 39] have dealt with replacing the tedious frame-by-frame optimization with a quicker feed-forward process in order to enable real-time applications. A convolutional neural network is trained to preserve the content of the input while applying the style of a fixed template. Both training objectives are implemented as handmade loss functions based on VGG-19 feature activations and their correlations. Successful as it is at quickly producing beautiful results, it is limited to one style input and struggles with context-aware stylisation tasks. Attempts to overcome these shortcomings [43, 32] reveal the limited scalability of hand-crafted losses for increasingly complex problems.

2.1.2. Adversarial Training

Having achieved impressive results in various fields, generative adversarial networks (or GANs for short) [12] replace increasingly complex hand-crafted loss functions with a versatile adversarial loss. This loss is particularly powerful since it is learned by a discriminator network that tries to distinguish between generated and real inputs.

Unconditional GANs, like the DCGAN [27], aim to reproduce a certain target distribution, using a random input as a source of entropy. In contrast, conditional GANs [24] provide the generator and discriminator networks with additional auxiliary information in order to control the generated output. Since the generated outputs can be directly coupled to meaningful inputs, conditional GANs are particularly useful for transformation tasks such as image-to-image transformation.

pix2pix [15] uses adversarial training to solve paired image-to-image tasks, meaning that matching input-output pairs are available in the training data. Using a unet [28] inspired generator and an additional L2-loss to further stabilize the generator training, pix2pix achieves impressive results in tasks like creating street scenes from semantic labels or photos from sketches. However, many real-word problems exist, where such paired inputs are not available (like synthetic to real smoke) or the mapping might not even be well defined (horse to zebra), making them unsuited for paired image-to-image translation.

In the absence of paired training data, the problem of learning a meaningful mapping between two image domains becomes highly under-constrained. Seemingly random mappings and mode collapse are commonly seen outcomes here. While some papers tackle these problems with weight sharing between generators [21], a more popular solution to this is the

introduction of cycle consistency [44, 17] to the training objective. The main idea behind this is that a forward followed by a backwards translation should result in the original input. Using this technique, various impressive results have been achieved in style transfer[44], object-to-object transfer [44, 17], face-to-face transfer [17] and caricature generation [5].

2.2. Temporal Consistency

While image-to-image translation techniques produce impressive results for single images, they struggle when applied to video data. The added complexity of time often causes visible temporal inconsistencies such as flickering artefacts in the output.

In the area of neural style transfer, several papers [29, 13, 10] have targeted the problem by introducing a number of hand-crafted temporal losses. The most basic idea is to enforce equal stylisation by compensating the motion using optical flows (Figure 5) and minimising a pixel-wise loss between neighbouring frames. More sophisticated solutions include consistency by initialisation, long term losses over multiple frames, multi-pass algorithms, luminance-warping constraints and feature-map-level losses.

For video generation using unconditional GANs [30, 37], a common approach is to treat content and motion separately and evaluate the final result with temporal discriminator that takes a 3D-video tensor as input.

tempoGAN [42] is a system for 3D fluid flow super resolution. It combines a conditional spatial discriminator with an unconditional temporal discriminator. Using ground truth velocities obtained from the simulation, the temporal discriminator inputs are advected to compensate for motion. Convergence is further improved by adding in a discriminator feature loss: It encourages the generation of outputs that produce similar discriminator activations as real data.

Compared to fluid simulations, obtaining ground truth optical flows is less trivial for most visual scenarios. vid2vid [40] approaches this problem by introducing a sequential generator that also outputs flow estimates. Combined with the conditional spatial and temporal discriminators the entire system is trained end-to-end. FRVSR [31] and TecoGAN [7] train FNet to estimate optical flows and directly provide the recurrent generator with motion-compensated previous outputs. In TecoGAN, a novel spatio-temporal discriminator automatically balances spatial and temporal optimisation and also cuts down on network size. Long-term consistency is further improved with a novel Ping-Pong loss.

2.2.1. CycleGANs for Videos

Unpaired video-to-video translation is commonly viewed as a translation task for unpaired but ordered frame sequences, combining the under-constrained nature of unpaired translation

task with the additional temporal constraints.

RecylceGAN [2] solves this problem by adding in domain specific, recurrent temporal predictors to a traditional CycleGAN. Using a novel recycle-loss that ensures a correct prediction of future frames after domain transfer, the otherwise unchanged (non-recurrent) generators will implicitly learn to produce temporally consistent results. Inference is then done by either applying the trained generator to an input video frame by frame or smoothly interpolating the latter with frame predictions.

Bashkirova et al. [3] follow the approach of treating video input as a 3D-tensor of consecutive frames. Using 3D convolutions in their spatio-temporal generators and discriminators, they manage to produce impressive, temporally consistent results. However, their solution suffers from well known memory related inefficiencies of 3D convolutional networks, ultimately sacrificing resolution and speed for temporal consistency.

3. A Temporally Consistent CycleGAN

We approach the problem of unpaired video-to-video translation by taking a well proven solution for unpaired image-to-image translation as a baseline, using it to translate a video frame by frame. In order to promote temporal consistency, we implement changes to the generators' and discriminators' architecture as well as their training objectives.

In this chapter, we will start by summarising the basics of adversarial training and the CycleGAN [44] before diving into our contributions for temporal consistency and accelerated convergence.

3.1. Adversarial Training

Generative adversarial networks consist of two competing neural networks, a generator and a discriminator. The general idea is for the two networks to contest in a zero-sum training game and gradually improve over time based on each other's feedback. In their game, the generator tries to fake samples of a certain target domain e.g. images of zebras and the discriminator will try to distinguish between real and fake samples.

More mathematically speaking, the objective of the generator G is to produce samples $G(z)$ that match a target data distribution X. In the simplest case, the generator input z serves as a source of entropy and is sampled from a random distribution Z. In that case, we call the generator network unconditional. The discriminator D acts as a binary classifier that is trained to distinguish between real data samples $D(X) = 1$ and generated ones $D(G(Z)) = 0$. By learning to fool the discriminator $D(G(Z)) = 1$, the generator will learn to match the target distribution $G(Z) = X$, without ever directly seeing the target data X.

The overall training objective can be summarised with the following minimax formulation, traditionally using the sigmoid cross entropy loss:

$$\min_D \max_G L_{GAN} = \mathrm{E}_{x \sim p(x)}[log(D(x))] + \mathrm{E}_{z \sim p(z)}[1 - log(D(G(z)))] \tag{3.1}$$

In their work, Goodfellow et al. [12] show that the optimal solution to the above problem also ensures $X = Z$ by minimising the Jensen–Shannon divergence.

As pointed out by Mao et al. [22] the standard sigmoid cross entropy can suffer from vanishing gradient problems during training. To combat this, they propose an alternative least-squares

loss that instead minimises the Pearson χ^2 divergence.

$$\min_D \max_G L_{LSGAN} = \mathrm{E}_{x\sim p(x)}[(D(x)-1)^2] + \mathrm{E}_{z\sim p(z)}[(D(G(z)))^2] \tag{3.2}$$

During initial tests, we found that the least squares loss generally produced better results for our visual use case. We will therefore only focus on results using the least squares loss in this work. In the following, we will also omit the $\mathrm{E}_{x\sim p(x)}$ and $\mathrm{E}_{z\sim p(z)}$ subscripts in favour of readability, since they only constitute a constant normalisation term in practice.

3.2. CycleGAN

In unpaired image-to-image translation, the objective is to learn a meaningful mapping between two image domains in the absence of matching input-output pairs. Since the cycleGAN has proven to be a powerful solution for this task, we will use it as a starting point for our work.

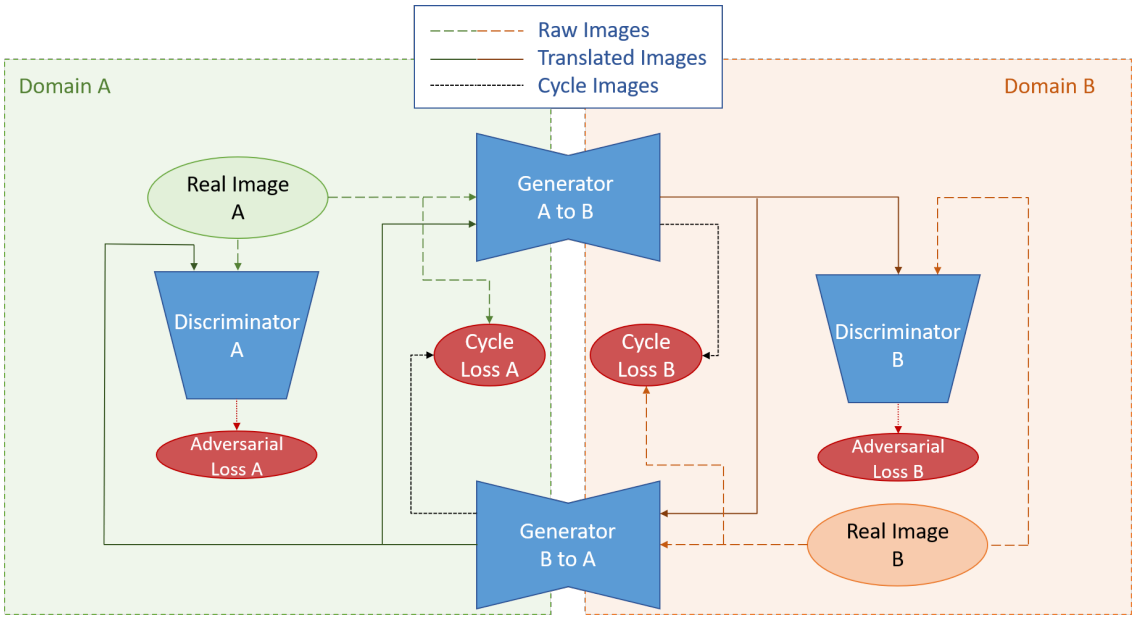


Figure 1 Architecture overview of the cycleGAN [44].

CycleGANs learn a mapping from a domain A to domain B in both forward and backward directions. To achieve this, two pairs of generators G_{A2B}, G_{B2A} and discriminators D_A, D_B are trained simultaneously. Since the goal is not just to generate any sample from the target domain, conditional generators [24] are used that take an input sample and generate a matching output. Due to the lack of 1-to-1 mappings of inputs and outputs, the discriminators remain unconditional.

Since we're dealing with pixel-image data, we use fully convolutional architectures for the gen-

erators and discriminators. We base our generators off the residual bottleneck architecture proposed by Johnson et al. [16] and combine them with markovial discriminators as used in pix2pix [15].

The training objective of the cycleGAN's generators contains adversarial and cycle consistency terms. The adversarial term, learned by the discriminators, assures that the output of the generators matches the desired target domains. Instead of maximising the discriminator loss, the generators' objective is often reformulated as a separate minimisation problem:

$$\begin{aligned} L_{spatial} = \lambda_{spatial}[&\|1 - D_A(G_{B2A}(b))\|_2 \\ &+\|1 - D_B(G_{A2B}(a))\|_2] \end{aligned} \tag{3.3}$$

The cycle consistency term ensures that the generators learn non-contradicting domain mappings, by punishing differences between the output of a full domain transfer cycle, e.g. $A \rightarrow B \rightarrow A$ and the original input. As Zhu et al. [44] showed, a simple pixel-wise L_1 loss is sufficient for this purpose.

$$\begin{aligned} L_{cycle} = \lambda_{cycle}[&\|G_{B2A}(G_{A2B}(a)) - a\|_1 \\ &+\|G_{A2B}(G_{B2A}(b)) - b\|_1] \end{aligned} \tag{3.4}$$

Even with the cycle consistency term in the objective, the cycleGAN just learns any cycle-consistent mapping between the two domains, allowing for random permutations of domain features. In visual applications, this is often noticeable as a colour shift during translation. While this would be totally acceptable for applications like style transfer to some degree, it can lead to a complete colour inversion (Figure 2). To avoid this, an identity loss term [36] is added to the generators' training objective. The identity loss makes sure that the colours of the generator inputs match those of the output and is implemented using a pixel-wise L_2-loss.

$$\begin{aligned} L_{identity} = \lambda_{identity}[&\|G_{A2B}(a) - a\|_2 \\ &+\|G_{B2A}(b) - b\|_2] \end{aligned} \tag{3.5}$$

In contrast to the original cycleGAN paper, we only use the identity loss to ensure correct initialisation and fade out $\lambda_{identity}$ later on. This allows the network to quickly learn trivial feature mappings (e.g. grass $\rightarrow$ grass) and still learn feature mappings that don't conserve colour (e.g. brown fur $\rightarrow$ zebra stripes) later on.

One potential problem for adversarial training is high model fluctuation due to strongly coupled generator-discriminator optimisation. This problem becomes even worse in our scenario with

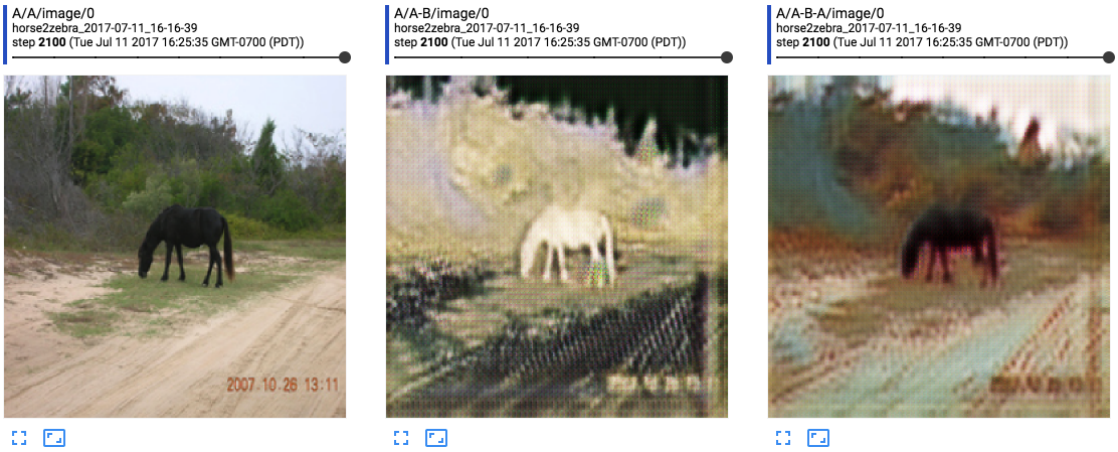


Figure 2 Undesirable colour flipping artefact as discovered in TensorBoard: input (left), generated output (middle), cycle reconstruction (right). Taken from [19].

four adversarial entities coupled in a cyclic relationship. To reduce the coupling and thus improve training stability, we follow the approach of Shrivastava et al. [33] and provide the discriminators with historic samples rather than direct generator outputs.

3.3. Temporal Consistency

In order to achieve temporally consistent results with a cycleGAN, we combine frame-recurrent generators with a shared temporal discriminator. To further improve long-term consistency, we extend the generators' training objective by another temporal term, the Ping-Pong loss.

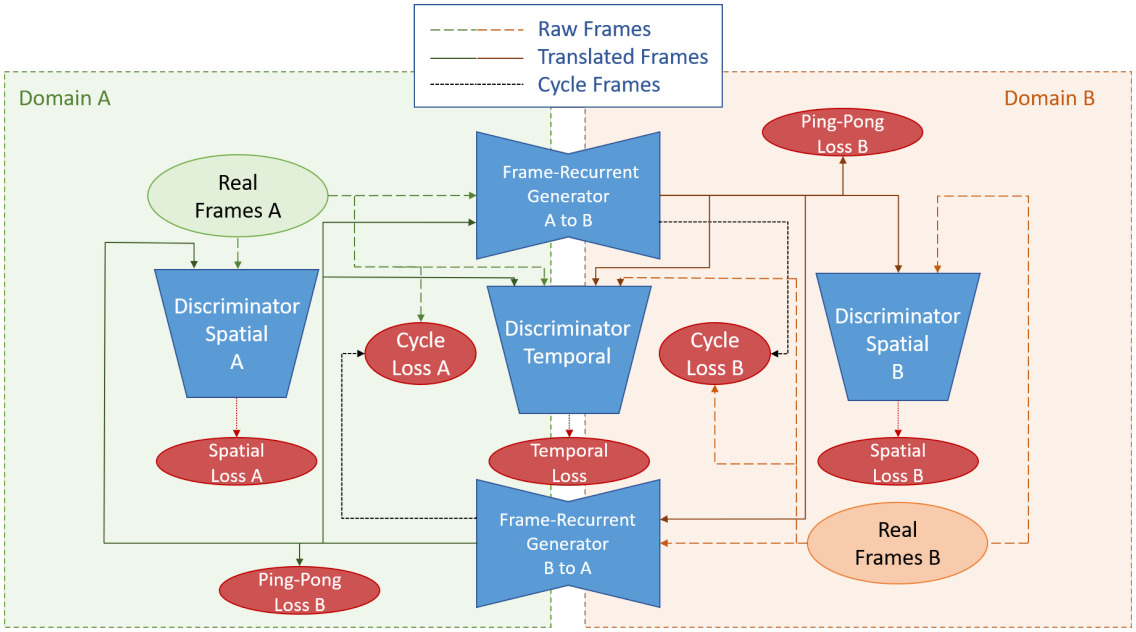


Figure 3 Architecture overview of the tricycleGAN.

Figure 3 shows a high level overview of our model. It is noteworthy that while the training process is rather complex, inference can be done frame by frame using only the trained

Figure 4 While the classical bottleneck generator struggles with preserving details in mostly unaltered image areas (left), the image information can skip the bottleneck in our unet generator, which helps preserving details.

generators. The following section will highlight the main design considerations and concepts of the above mentioned components.

3.3.1. Frame-Recurrent Generator

As a first step towards temporal consistency, we allow our generator to "see" back in time in order to facilitate temporally coherent predictions. While temporally consistent predictions can be achieved without temporal context [13], it often comes at the expense of added details. In these cases the generator will learn to favour temporal consistency and not boldly add details to temporally uncertain areas of the output.

One approach to promote temporal consistency is to provide the generator with an entire sequence of frames instead of just one input frame. However, the need for multiple frames at once to produce one output makes this approach unintuitive for processing real-time video streams and the need to redundantly process the same frames multiple times has shown to also contribute to poor performance. In the end, the output frames are still effectively generated independent of each other.

Since the ultimate goal is to have temporal consistency in the output sequence, we instead follow the idea of providing the generator with previous generator outputs. To keep it simple, we refrain from using an actual RNN architecture [4] and instead manually feed the generator with its last output frame g_{t-1} as a second input, thus recursively defining $g_t = G(z_t, g_{t-1})$. Despite artificially constraining the generator's temporal information to just the previous frame, we found this solution to work sufficiently well. For completeness, we define $g_0 = -1.0$ corresponding to a constant black frame. The intuition behind this is to not provide the generator with any temporal information and thus allow it to freely generate the first frame.

For some problem domains (e.g. Horse to Zebra) we found that in less crucial and thus mostly unaltered image areas (such as patches of grass) suffered from quite significant loss of detail (Figure 4). We suspect the cause for this to lie in the bottleneck structure of our generators. Since the entire image information needs to pass the central bottleneck, less important, low-level information will be lost along the way. Similar to pix2pix [15], we approach the problem by enhancing our generators with skip connections [28] and thus allow information to skip the bottleneck. While slightly slowing down inference times, we found this to be quite beneficial in these cases. (Figure 4)

3.3.2. FNet

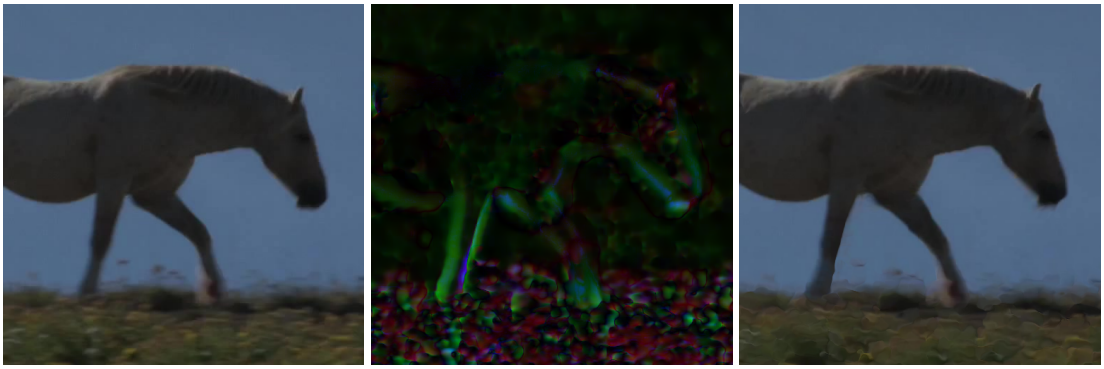


Figure 5 An example of motion compensation based on optical flow: Given an input frame (left) and the optical flow to the next frame (middle), we can compensate for motion (right) by warping the input.

In order to generate a new frame based on a previous output, the generator network would need to understand the motion between the two frames. While the generators could learn to do this themselves, a common approach is to use a dedicated motion estimation network to estimate optical flows (Figure 5) and compensate for motion in a preprocessing step. In contrast to vid2vid [40], who train a motion estimator end-to-end, we use a pre-trained general-purpose optical flow estimation network F based on the FNet architecture proposed by Sajjadi et al. [31]. Compared to more sophisticated solutions like FlowNet 2.0 [14], we chose FNet for its architectural simplicity while still providing sufficient results for frame preprocessing.

Figure 6 shows the full motion compensated generator pipeline. We first use FNet to estimate the optical flow $f_{t-1,t} = F(z_{t-1}, z_t)$ and then warp the previous output $g'_{t-1} = w(g_{t-1}, f_{t-1,t})$ to compensate for motion. Finally, we feed the current input and our warped output into the generator and receive the new frame $g_t = G(z_t, g'_{t-1})$.

To achieve even better results in specific problem domains, it would be possible to train the FNet even further using something like a reprojection L1-loss:

$$L_F = \|w(z_{t-1}, F(z_{t-1}, z_t)) - z_t\|_1 \tag{3.6}$$

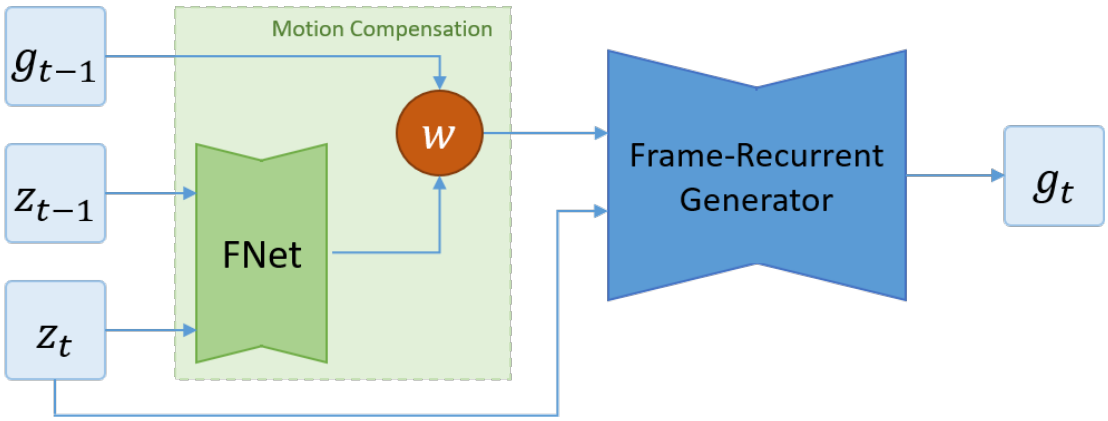


Figure 6 The full frame-recurrent generator pipeline with motion compensation.

where w is the warping function. However, we found that the pretrained FNet produces sufficient results out of the box. Only contributing to a diminishing improvement in overall quality, we decided to not train the FNet in favour of speeding up the overall training process.

3.3.3. Temporal Discriminators

Previous Work

To ensure that the generator actually learns to produce temporally consistent results, a temporal consistency term needs to be added to the generator's training objective. A traditional approach for this would be an L2-loss between neighbouring frames, compensating for motion using optical flow.

$$L_{tempL2} = \|g'_{t-1} - g_t\|_2 \tag{3.7}$$

where $g'_{t-1} = w(g_{t-1}, F(z_{t-1}, z_t))$ and w is the warping function. However, these kinds of pixel-wise temporal losses are known to sacrifice fine details for temporal consistency, due to the averaging nature of the L2-loss.

Using adversarial losses for temporal consistency avoids these problems and has proven to produce convincing results in multiple areas of application [15, 7, 42]. For this purpose, we experimented with different concepts of temporal discriminators which will be discussed in the following. While the purposes of the temporal and spatial discriminators differ greatly, we will follow the proven approach to base them on the same architecture [7, 42, 40].

Shared Temporal Discriminator

The most straight-forward approach is to introduce a dedicated unconditional temporal discriminator that takes a sequence of frames as input. Since its only task is to evaluate temporal consistency in general, it can be shared between domains. This ultimately leads to a cycleGAN with two spatial D_A, D_B and one shared temporal discriminator D_T (Figure 7).

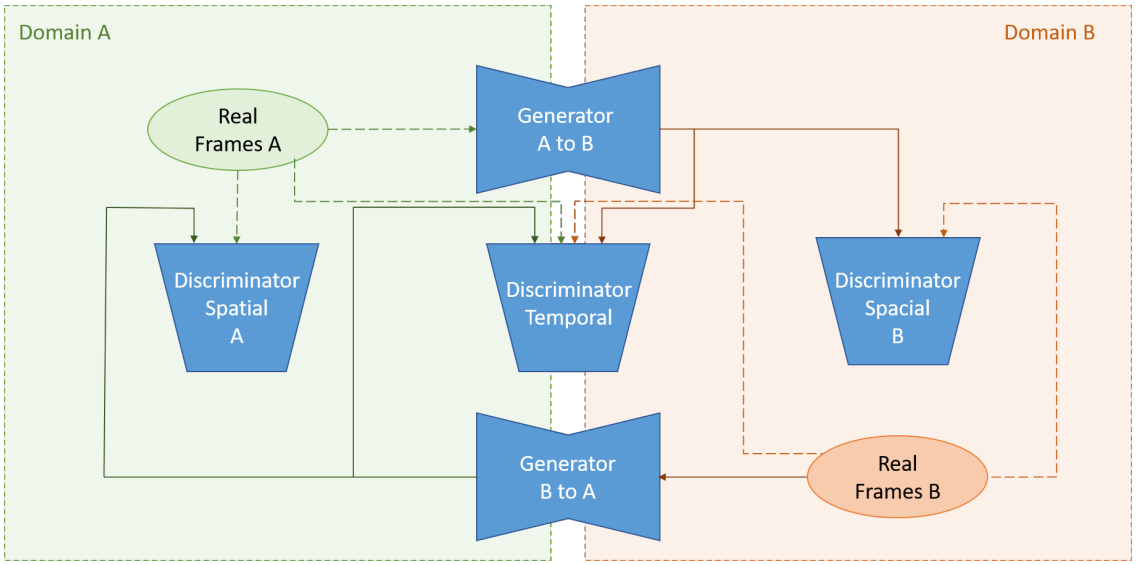


Figure 7 A cycleGAN with two spatial discriminators and a shared temporal discriminator.

In our work, we will focus on temporal discriminators $D_{temp}(x'_{t-1}, x_t, x'_{t+1})$ with three input frames: the current frame x_t as well as one predecessor x_{t-1} and successor x_{t+1}. To simplify the task for the discriminator, we first compensate for motion in the inputs using the previously mentioned motion estimator network F (Figure 8), where $x'_{t-1} = w(x_{t-1}, F(x_{t-1}, x_t))$, $x'_{t+1} = w(x_{t+1}, F(x_t, x_{t+1}))$ and w is the warping function.

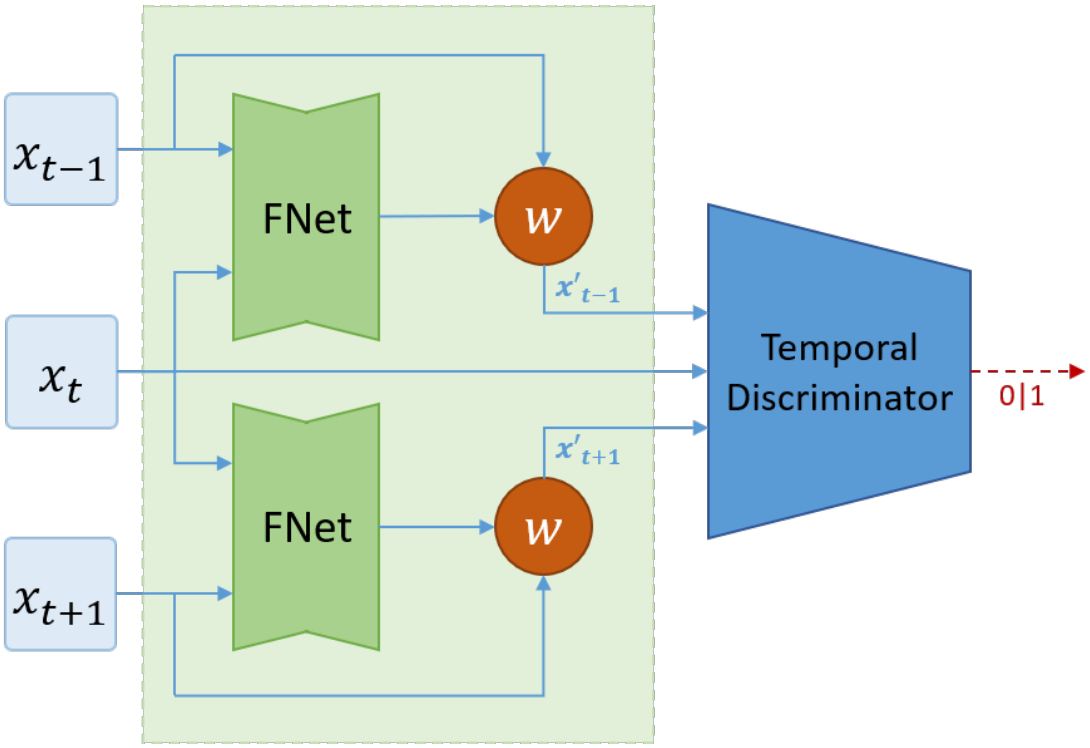


Figure 8 The temporal discriminator pipeline with motion compensation.

Finally, we add the temporal adversarial loss term to the generators' training objective:

$$L_{temporal} = \lambda_{temporal}[\|1 - D_T(ga'_{t-1}, ga_t, ga'_{t+1})\|_2 + \|1 - D_T(gb'_{t-1}, gb_t, gb'_{t+1})\|_2] \tag{3.8}$$

where $ga = G_{B2A}(a_t, ga_{t-1})$ and $gb = G_{A2B}(b_t, gb_{t-1})$.

Spatio-Temporal Discriminator

Instead of dealing with spatial and temporal optimisation in separate, the idea of spatio-temporal discriminators is to directly learn the domain-specific spatio-temporal properties. For this, both spatial discriminators are modified to directly accept motion compensated frame triplets $(x'_{t-1}, x_t, x'_{t+1})$ as inputs (Figure 9), making them architecturally identical to the shared temporal discriminator.

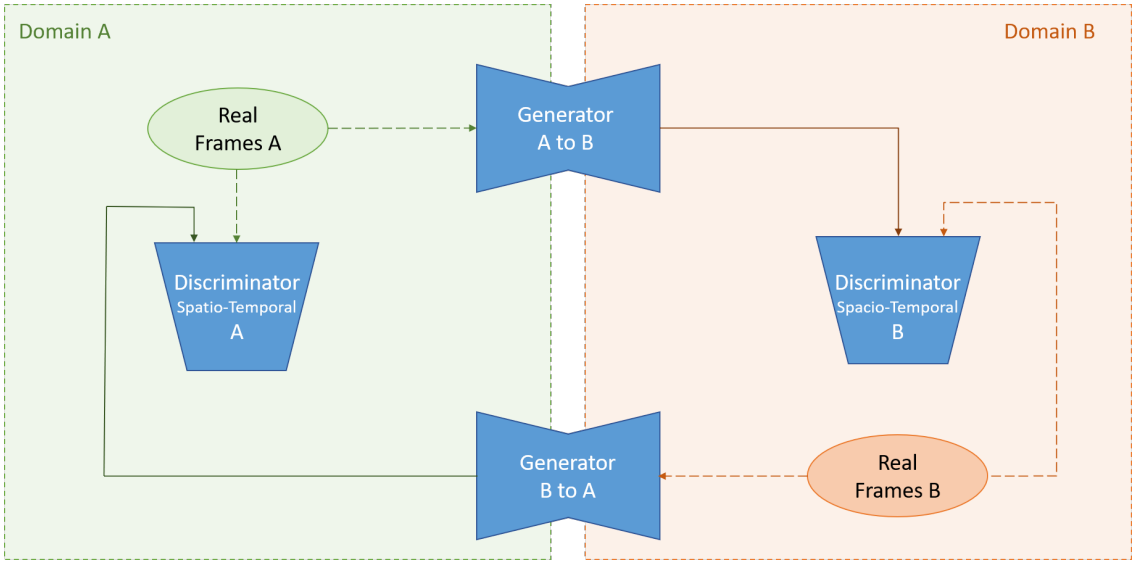


Figure 9 A cycleGAN with two spatio-temporal discriminators.

Spatio-temporal discriminators have the benefit of automatically balancing and mixing spatial and temporal features in the spatio-temporal training objective. We also expect reduced memory consumption due to the reduced total number of trainable network weights. Finally, only having to train one discriminator per domain should further improve training time and memory consumption.

Spatio-Temporal Multi-Domain-Discriminator

Taking the idea of merging discriminators even further, we also experimented with a novel spatio-temporal multi-domain-discriminator. Merging spatio-temporal discriminators for both domains, this ultimately leaves just one discriminator for the entire cycleGAN (Figure 10). In order to not lose expressiveness, the discriminator output changes from a single probability value to a one-hot-encoded vector representing the different classes of images, in our exam-

ple $(Real_A, Fake_A, Real_B, Fake_B)$. Where a sigmoid activation was used in the last layer previously, we now employ its multidimensional generalisation, the softmax.

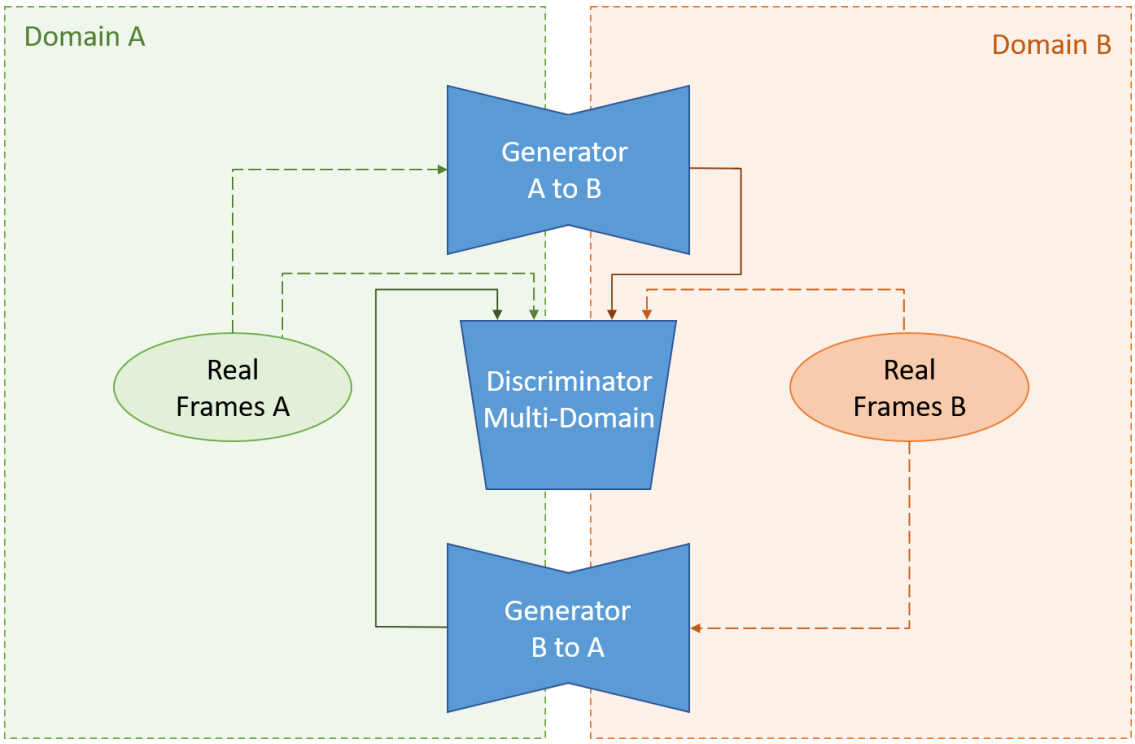


Figure 10 A cycleGAN with one shared spatio-temporal multi-domain-discriminator.

This approach is motivated by the assumption that the spatio-temporal discriminators would learn similar features in both domains. By merging them into one, we can get rid of redundant structures, effectively reducing the total number of trainable weights and therefore reducing memory consumption even further.

Final Discriminator Choice

Figure 11 shows a comparison of results for the three different architectures after 50k training iterations and otherwise identical settings. For the shared temporal discriminator loss, we empirically found a temporal weight $\lambda_{temporal} = 0.5$ to produce best results.

Figure 11 Zoomed-in results for different temporal discriminators: shared temporal discriminator (left), spatio-Temporal discriminators (middle), spatio-temporal multi-domain-discriminator (right).

In our tests we found the shared temporal discriminator generally converged the fastest and thus produced the best results. While the training process for the spatio-temporal discriminator was slightly faster, convergence slowed down significantly. The same story goes for the multi-domain-discriminator, where it is hard to make out any useful convergence. We suspect this to be the result of the increasingly complex training objective of the discriminators.

While some of the issues could potentially have been solved with further architecture and parameter tweaking, the performance benefits of the more involved discriminator concepts fell short of our expectations. GPU memory being the most precious resource with a network of this size, we found that reducing the number of trainable parameters by merging discriminator functionality had a negligible effect on overall memory consumption. Our reasoning for this behaviour and further details on memory management in TensorFlow can be found in Appendix A.1.

Due to its best overall results, the negligible real-world overhead and also better user controllability thanks to separated spatial and temporal optimisation, we will only focus on the shared temporal discriminator architecture in our results.

3.3.4. Ping-Pong Loss

While the introduction of a temporal discriminator effectively ensures temporal consistency from one frame to the next, long term consistency still remains an open problem. Gradual vanishing or reinforcement of spatial details are commonly observed artefacts (Figure 25, middle).

Extending the frame window for the temporal discriminator would potentially solve this problem, but at the cost of unreasonably slowing down training and increasing memory consumption. We therefore employ a Ping-Pong loss [7] to take care of the long-term drift. The Ping-Pong loss is based on the observations that a sequence of frames played in reverse still represents a valid video and that the output of video-to-video translation should be independent of the frame ordering.

For the Ping-Pong loss, we chain a forward frame sequence e.g. $ping^A = (a_1, ..., a_t, ..., a_n)$ with its reversed counterpart $pong^A = (a_n, ..., a_t, ..., a_1)$ and use this extended sequence to train the network. In the generated output, we have one corresponding frame in $pong^{A2B}$ for each frame in the $ping^{A2B}$ that, based on our assumptions, should still be identical (Figure 12). We thus formulate the Ping-Pong loss as an L_2 norm:

$$\begin{aligned} L_{PP} = \lambda_{PP}[&\sum_{i=2}^{n-1} \|ping_i^{A2B} - pong_i^{A2B}\|_2 \\ &+ \sum_{i=2}^{n-1} \|ping_i^{B2A} - pong_i^{B2A}\|_2] \end{aligned} \tag{3.9}$$

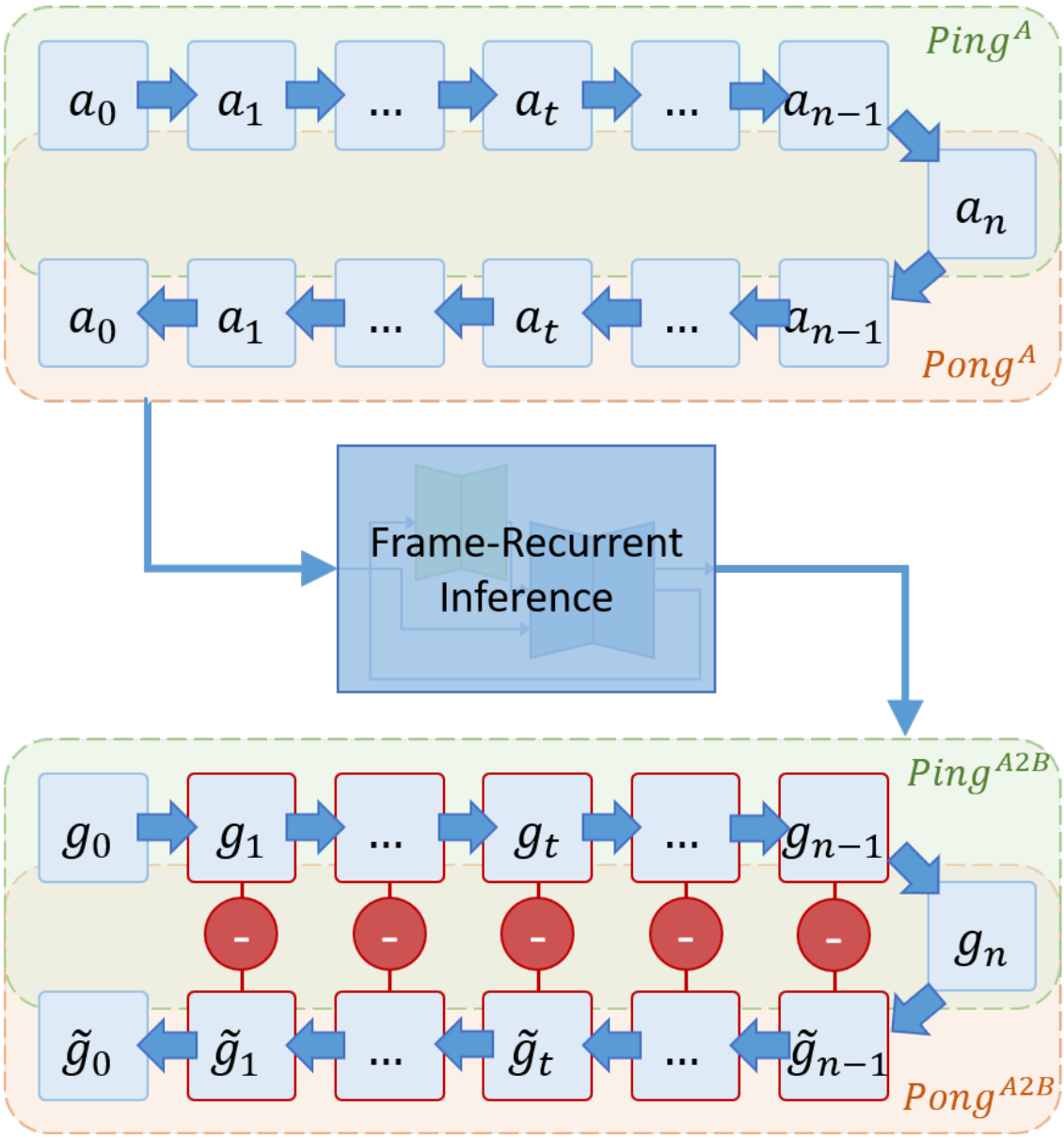


Figure 12 An overview over the Ping-Pong loss computation.

In contrast to [7], we don't take the very first frame into account, since it's generated without any temporal information. As a consequence, we found the first frames to be highly volatile and dominate the Ping-Pong loss.

In combination with the temporal discriminator, the Ping-Pong loss manages to improve both short and long-term consistency (Figure 25).

3.4. Convergence Improvements

With all the mechanisms for temporal coherence in place, convergence is slowed down by conflicting training objectives while the individual training iterations themselves also take sub-

stantially longer. As a result, a full training run would commonly take days even on a high end single-GPU setup (30h on an Nvidia RTX Titan). Since not much can be done to the iteration speed without sacrificing quality, we make an effort to speed up convergence. As adversarial training is notorious for its instability, we aim to improve convergence by further constraining the training objective with auxiliary non-adversarial losses.

3.4.1. Latent Space Consistency Loss

The main objective of the cycleGAN is to learn mappings between two domains A and B. This is done by the generators that individually learn $A \rightarrow B$ and $B \rightarrow A$ respectively. Intuitively, $A \rightarrow B$ should be the inverse mapping of $B \rightarrow A$ and vice versa which is also enforced by the cycle consistency term.

When looking at the encoder-decoder structure of our generators $G_{AB}(a) = Dec_{AB}(Enc_{AB}(a))$, it would be intuitive to assume that the encoder part translates the input into a universal latent space representation $Z = Enc_{AB}(A)$ first which is then translated to the other domain by the decoder $B = Dec_{AB}(Z)$. By explicitly forcing both generators to learn the same latent space Z, we can constrain the training objective even further, therefore expecting faster convergence.

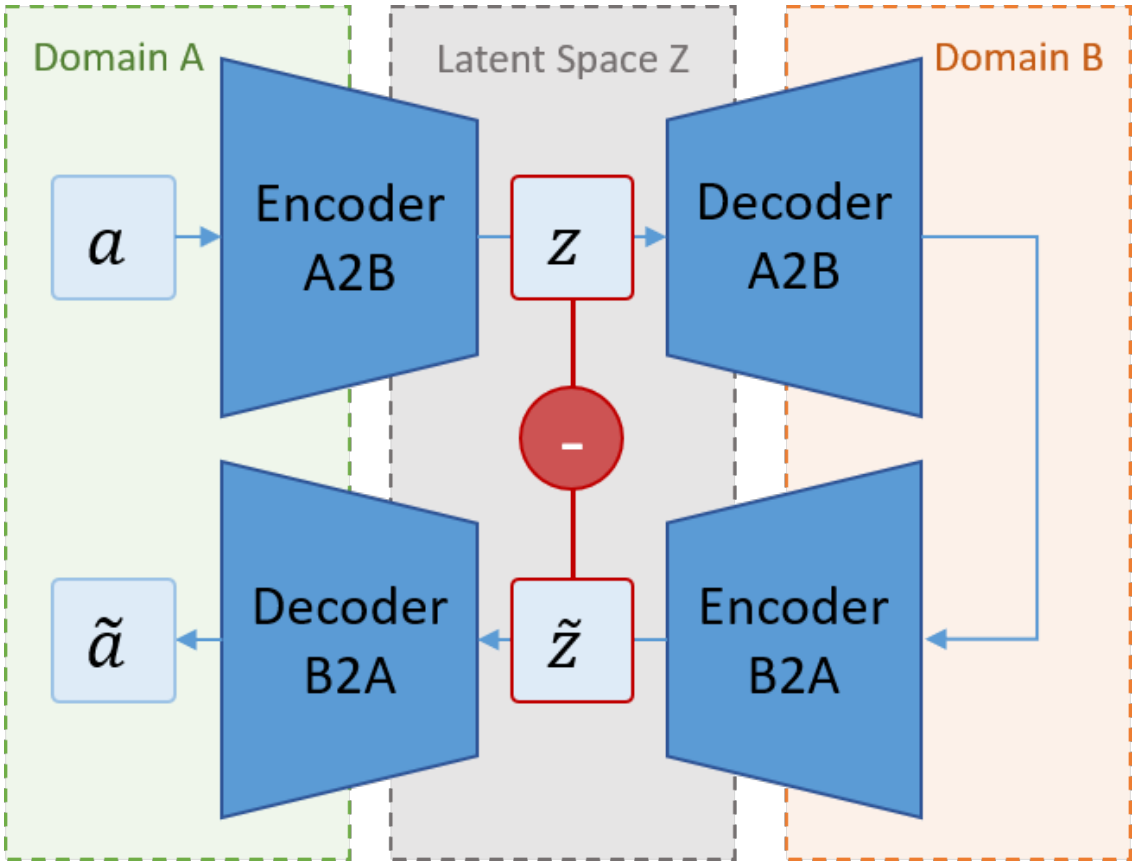


Figure 13 By forcing the latent spaces to be identical for both generators, we intend to speed up convergence

We propose a novel latent space consistency loss that ensures a uniform latent space representation Z in both generators. It follows the general principle of the standard cycle consis-

tency loss but focuses on the generators' latent space activations instead of their final outputs. More specifically, we perform a full domain transfer cycle e.g. $A(\rightarrow Z) \rightarrow B(\rightarrow Z) \rightarrow A$ and compute an L_1 norm over the latent space activations.

$$\begin{aligned} L_{LSCL} = \lambda_{LSCL}[&\|Enc_{AB}(a) - Enc_{BA}(G_{AB}(a))\|_1 \\ &+\|Enc_{BA}(b) - Enc_{BA}(G_{BA}(b))\|_1] \end{aligned} \tag{3.10}$$

As it can be seen in Figure 14, the latent space consistency loss slightly improves convergence for a fixed number of training iterations. However, we also found it to slow down training by 70 ms per iteration accounting for 5-7% longer training time, depending on the configuration. Ultimately neutralising its benefits on total convergence time, we decided to leave the latent space consistency loss out of the training objective for our final results and focus on more promising approaches to stabilise and speed up convergence.

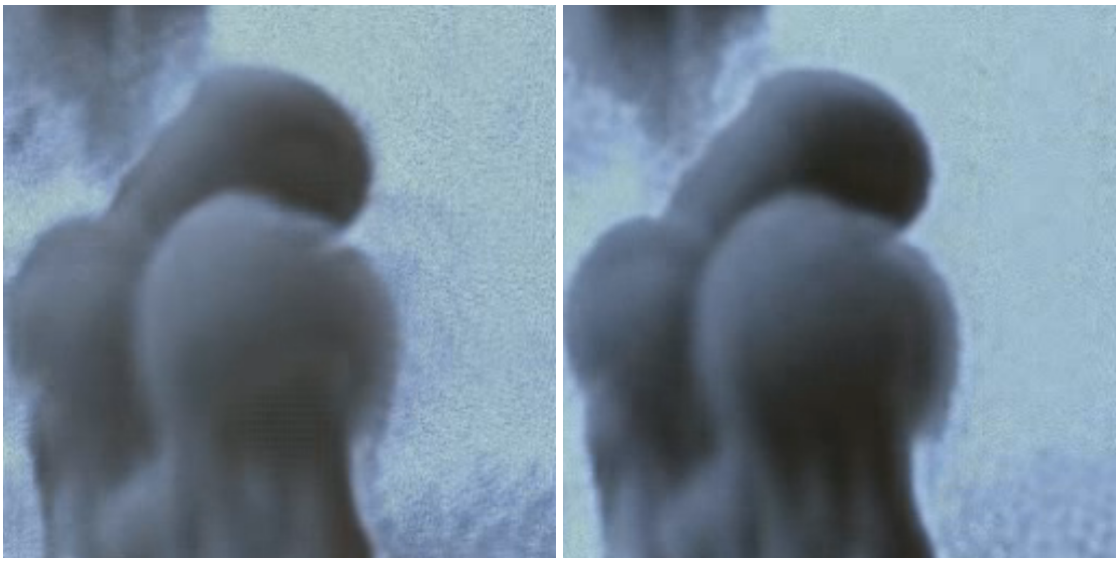


Figure 14 The latent space consistency loss slightly improves convergence: Results after 50k iterations without (left) and with latent space consistency loss (right).

3.4.2. VGG Style Loss

Perceptual losses have been proven quite effective in producing results of improved visual quality. TecoGAN [7] uses feature extracted by the VGG-19 [35] image classification network, to further constrain the training objective by encouraging outputs that are perceptually closer to ground truth data.

Since we don't have paired ground truth data, it would not make sense to apply a spatially aware feature loss to our training. Instead, we focus on producing outputs with similar occurrences of features or, mathematically speaking, equal feature correlations. This is equivalent to the concept of a style loss as it is used for neural style transfer [11] and computed as the gram matrix of feature vectors.

Summing over VGG-19 layers, the feature loss is computed as an L_2-norm of generated and ground truth feature gram matrices $M^l_{VGG}(x) = F^l_{VGG}(x)^T F^l_{VGG}(x)$ where $F^l(x)$ are the VGG-19 feature activations in layer l for a fixed input x.

$$\begin{aligned} L_{VGG} = \lambda_{VGG}[& \sum_{l \in layers} \|M^l_{VGG}(a) - M^l_{VGG}(G_{B2A}(b))\|_2 \\ & + \sum_{l \in layers} \|M^l_{VGG}(b) - M^l_{VGG}(G_{A2B}(a))\|_2] \end{aligned} \tag{3.11}$$

For our setting, we found that extracting features from layers `conv2_1`, `conv3_1` and `conv4_1` empirically gave us the best trade-off between performance and quality. To approximate the ground truth style, we use a sample from the ground truth domain as a stochastic style template. Even with a small sample size such as 1, we found the VGG style loss to be an effective tool to speed up convergence. Nonetheless, pre-computing average gram matrices for both domains remains an interesting item for future work.

3.4.3. Discriminator Style Loss

While it's surely desirable to produce outputs that match ground truth data perceptually, the generator's main objective in adversarial training remains to "fool" the discriminator. To account for this, Xie et al. [42] apply the concept of feature losses directly to the discriminator. The idea is to encourage the generator to produce features similar to ground truth data.

In similar fashion to the VGG style loss, we focus on the gram matrices M^l_A and M^l_B instead of directly comparing feature activations and compute a sum of $L2$-norms.

$$\begin{aligned} L_{DSL} = \lambda_{DSL}[& \sum_{l \in layers} \|M^l_A(a) - M^l_A(G_{B2A}(b))\|_2 \\ & + \sum_{l \in layers} \|M^l_B(b) - M^l_B(G_{A2B}(a))\|_2] \end{aligned} \tag{3.12}$$

For the computation, we use activations from all inner discriminator layers. It is worth noting, that unlike the VGG-19, the discriminators are constantly updating during training, thus making precomputation of average gram matrices across the entire data set unreasonable.

In addition to significantly speeding up convergence (Figure 15), we also found the discriminator style loss to help with balancing the training, by closing down the gap between generators and discriminators.

While the discriminator style loss is a valuable tool for quickly pushing the generator towards producing outputs similar to the target domain, we found it to be detrimental in later training stages. It can be observed that the generators would learn to add in domain specific features

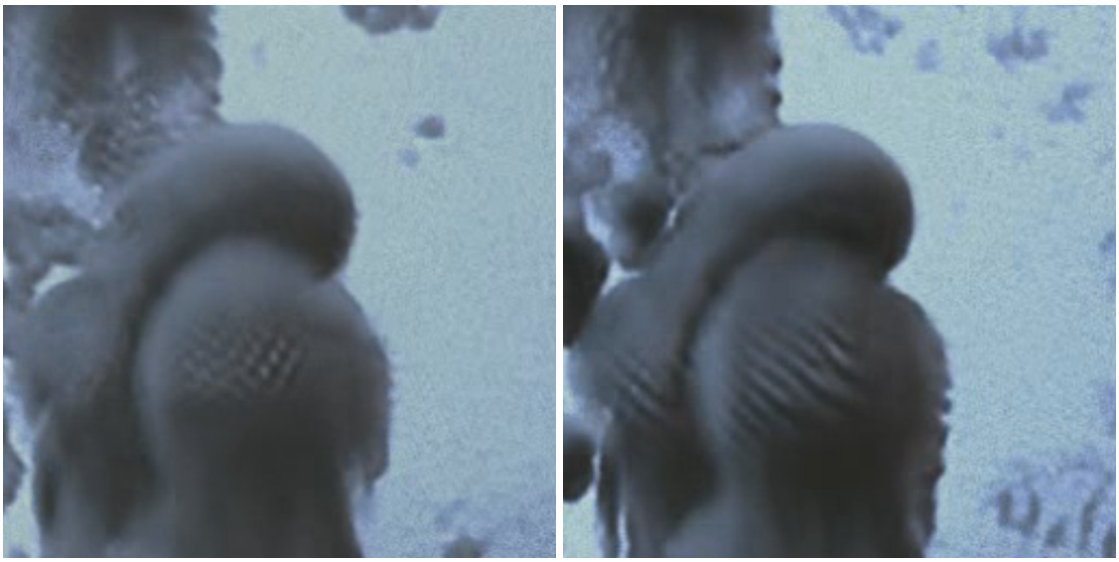


Figure 15 The discriminator style loss improves convergence significantly: Results after 50k iterations without (left) and with discriminator style loss (right).

out of context (Figure 16). We explain this behaviour as a consequence of shifting the training objective from learning a valid domain mapping towards simply creating any output with a certain style. To prevent this from happening while still getting the convergence benefit, we slowly fade out the discriminator style loss after around 50k iterations.

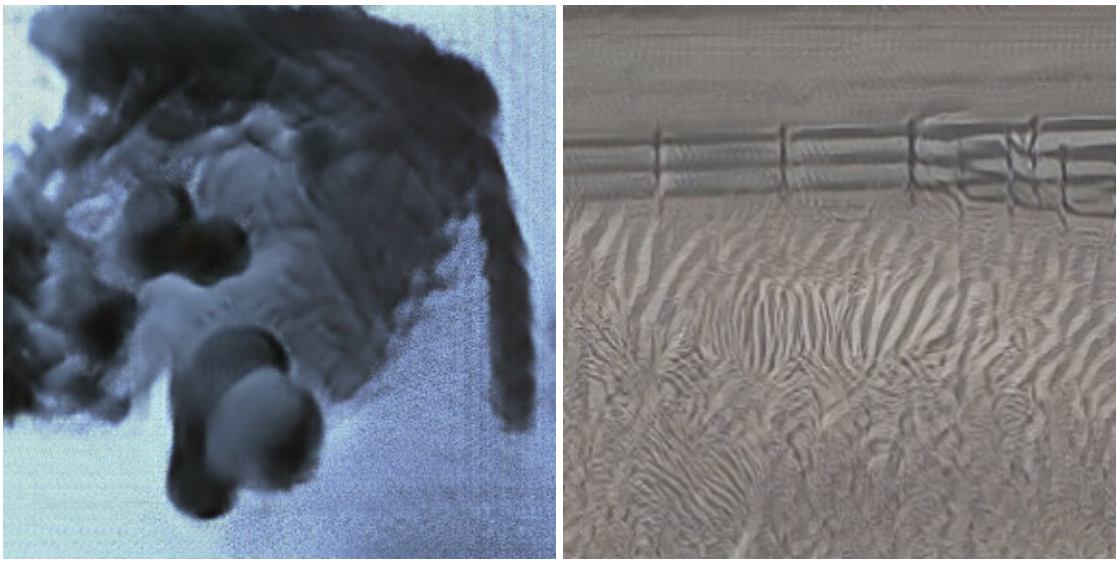


Figure 16 In later training stages the discriminator style loss becomes detremental: The generators learn to fill up empty space with smoke (left) or create phantom zebras (right).

Compared to the VGG-19 style loss, the discriminator style loss poses a more goal oriented approach to speeding up convergence, since it directly supports the generator in the adversarial competition. While both losses ultimately show similar effects on training, we will only focus on the discriminator style loss for the generation of our results.

3.4.4. Training Balancing

Our final approach on speeding up training is based on a common observation when training adversarial nets in practice. Since the discriminator ultimately has the easier objective of only rating the generated images, it will most often dominate the adversarial game by confidently identifying real and fake images. As a consequence, we assume that the discriminator should still have a fair chance even if it skips training every now and then.

As a measure to close the gap between generator and discriminator and also speed up training, we stop training the discriminator once it becomes ”too good“. As a metric for this, we use the mean difference between predictions for real and generated inputs. Since discriminator outputs are highly volatile, we use an exponential moving average to smooth our training balancer values.

In practice we found that using this technique, we managed to arbitrarily restrain the strength of the discriminator. However, we found the performance benefit from this to be insignificant. We discuss potential reasons for this in Appendix A.2. Since we expect a stronger discriminator to produce more meaningful gradients for the generator, we decided to not use training balancing for our results.

3.5. The Big Picture

After shedding light onto the individual ideas and components needed for temporally consistent unpaired video-to-video translation, we will now focus on their interplay in the full model. Due to the presence of three discriminators during training, we will refer to the full model as tricycleGAN.

Since we don’t train the FNet, the total optimisation problem breaks down to an adversarial team game. While the generators need to work together in order to tackle cycle consistency, each one still has to fight his personal spatial discriminator, as well as the common enemy, the temporal discriminator. To regulate the outcome of the game, the generators are furthermore constrained with identity and Ping-Pong losses as well as a Discriminator Style Loss, to give them a slight advantage. It is noteworthy that even though in our formulation (Equation 3.13) both generators share the exact same loss function, back propagation will automatically take care of correctly splitting up the gradients for optimisation.

$$L_{G_{A2B}} = L_{G_{B2A}} = L_{spatial} + L_{cycle} + L_{identity} + L_{temporal} + L_{PP} + L_{DSL} \tag{3.13}$$

The discriminators on the other hand have the much easier objective of only distinguishing real from fake images. While the spatial discriminators will only consider inputs from their specific target domain (Equations 3.14, 3.15), the temporal will check temporal consistency

for both domains (Equation 3.16).

$$L_{D_A} = \frac{1}{2}[(D_A(a_t) - 1)^2 + (D_A(ga_t))^2] \tag{3.14}$$

$$L_{D_B} = \frac{1}{2}[(D_B(b_t) - 1)^2 + (D_B(gb_t))^2] \tag{3.15}$$

$$\begin{aligned} L_{D_T} = \frac{1}{4}[&(D_T(a'_{t-1}, a_t, a'_{t+1}) - 1)^2 + (D_T(ga'_{t-1}, ga_t, ga'_{t+1}))^2 \\ &+(D_T(b'_{t-1}, b_t, b'_{t+1}) - 1)^2 + (D_T(gb'_{t-1}, gb_t, gb'_{t+1}))^2] \end{aligned} \tag{3.16}$$

4. Implementation Details

The following chapter will give an overview of the main tasks and pitfalls when implementing the tricycleGAN in a Python/TensorFlow environment. We will first briefly summarise the features of the baseline cycleGAN implementation, before touching on some highlights with regard to implementing the model's compute graph, the training and inference pipelines as well as other quality of life improvements. The code and documentation on how to use it can be found in our GitHub Repository [23].

4.1. Baseline Implementation

We base our implementation on Youngwoon Lee's CycleGAN-Tensorflow repository [19], an open-source GitHub project. While a multitude of functional cycleGAN implementations can be found freely available, we chose this specific one for its comparatively clear and up-to-date design principles, structured architecture and clean code.

The repository implements most features and data sets of the original cycleGAN paper [44]. It provides a basic implementation of the cycleGAN architecture (Chapter 3.2) using the LSGAN-loss [22]. The self-contained training pipeline automatically downloads training data from a pre-defined selection of datasets, preprocesses it to .h5 files, runs a full training run and subsequently performs a test run on the freshly trained model. The command line interface offers a handful of useful customisation options and the entire training process can be monitored with TensorBoard.

However, training with a custom dataset is not supported by default, requiring changes in the source code or tempering with existing dataset directories. Additionally, the size of datasets is also limited by the fact that they are completely loaded into memory for training. Another shortcoming is the lack of an identity loss implementation. As a consequence of that, the implementation suffers from frequent colour inversion problems.

4.2. Compute Graph

Figure 17 shows an overview of the compute graph for one generator of the tricycleGAN. The following section will walk through the different stages of the pipeline and highlight some implementation details.

To enable the processing of frame sequences instead of single images, the input `tf.placeholder` is extended from 2D to 3D data, ultimately generating `tf.tensors` of shape (batch size, #frames, width, height, #colour channels). For augmenting our data, we use the workaround of embedding the frames into the colour channels first, since most random data augmenta-

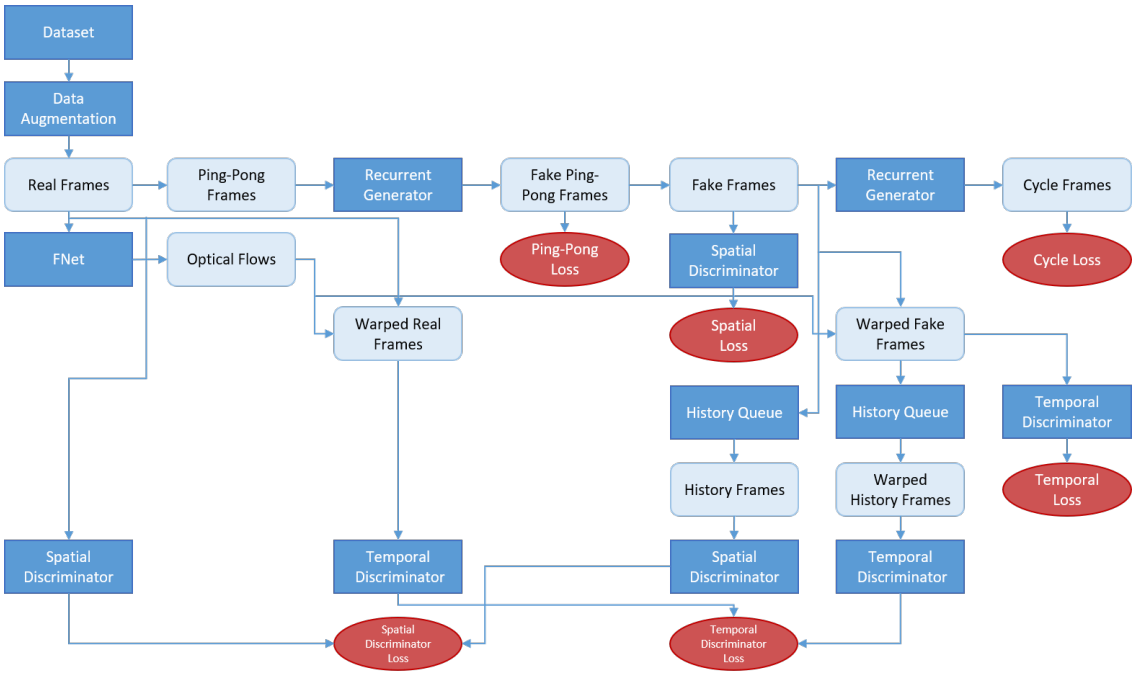


Figure 17 A simplified overview of the compute graph for one half of the tricycleGAN.

tion mechanisms do not support 3D-data in TensorFlow. This also ensures that all frames in a sequence are augmented in the same way.

As a first step in the pipeline, optical flow are estimated for the frame triplet that will later be passed to the temporal discriminator. Since arbitrary lengths of frame sequences are supported, we choose the last three frames for this. For motion estimation, we use a pre-trained FNet model. Having a simple encoder-decoder architecture, the FNet takes adjacent frames z_{t-1} and z_t as inputs and estimates the optical flow motion in between. The FNet was pretrained for TecoGAN [7] using an L2 warping loss, $L_{\text{warp}} = \|z_t - w(z_{t-1}, F(z_{t-1}, z_t))\|_2$ on 32x32 cropped patches from 250 short videos, each with 120 frames and varying resolutions of 1280x720 and upwards.

Instead of actually treating the frames as two separate input tensors, they are again stacked in the colour channels and fed as one tensor. We will generally use this approach for all our networks with multiple inputs. Thanks to automatic shape inference in TensorFlow, this allowed us to implement the recurrent generators and temporal discriminator without any changes to the network architecture code.

Using the estimated flow, the ground truth frames are warped by applying `tf.contrib.image.dense_image_warp` to the outer frames. These warped frames serve as a real input for the temporal discriminator. The spatial discriminator is supplied with the middle frame only.

For domain translation, the recurrent generator with motion compensation is used. Initialising the zeroth input and result as `tf.constant(-1.0, ... )`, it iterates over the frames: com-

pute the optical flow to the last input frame using FNet, warp the last result frame and finally feed both the current frame and the warped last result to the generator network.

The last three generated frames are again warped using the precomputed optical flow. The motion compensated fake frame triplet is then directly passed to the discriminators to compute the adversarial generator loss and also pushed into a history queue. A random sample from the history queue is then used to compute the adversarial discriminator loss. In order to increase training stability during early stages, we found it effective to start training without temporal losses and with the identity loss in place. During training, the respective weights are gradually faded in or out based on the global training step.

With the translated frame sequence, we compute the identity loss as a sum of absolute errors compared to the raw frames. We do the back translation of the full frame sequence in the same manner as the forward translation and compute the cycle consistency loss as yet another sum of absolute errors.

In order to compute the Ping-Pong loss, the full frame sequence is first concatenated with its reversed copy to create the Ping-Pong sequence. In the process, the duplicate first frame of the pong sequence is removed, effectively leaving its counterpart in ping as a pivot frame in the Ping-Pong sequence. Afterwards, the entire Ping-Pong sequence is translated recurrently and the translated Ping and Pong sequences are recovered. To reduce redundant computation, we reuse the processed ping sequence instead of domain translating the full frame sequence by itself.

We base our VGG-19 off the pre-trained implementation that ships with `tf.slim.nets.vgg` and modify it to support variable reuse. This was necessary in order to support feeding multiple tensors in one run. The `tf.slim` implementation automatically returns a dictionary with intermediate activations in addition to the final output. For the selected VGG layers, the gram matrices are computed by multiplying the activation vectors with their transpose. The style loss is then computed as a sum of squared errors between corresponding real-fake gram matrix sets.

For the computation of the discriminator style loss, we added the option to return a list of all inner layer activations in the discriminator. Similar to the VGG style loss, the style losses are then computed for both spatial and the temporal discriminator.

4.3. Training

To facilitate training with the new architecture, a handful of changes were made to the training process in TensorFlow. The following section will give a brief overview over some of the highlights.

4.3.1. Video Data Handling

While training neural networks with images is a well studied problem with decent API support, using videos for training is much less common. In order to maintain compatibility with image-to-image translation and also use well proven image processing pipelines, we decided to pre-process the video data to individual numbered frame images prior to training. It is noteworthy that while it is becoming common practice to use `TFRecords` to handle video data in TensorFlow, our requirement of accessing random batches of frames throughout the dataset made TFRecords unprofitable.

Since preloading all training data into memory becomes unfeasible for larger datasets, we follow the approach of loading data from disk at runtime. To do this efficiently, we take advantage of the `tf.data` pipeline. It provides mechanisms to prefetch and preprocess random batches of data asynchronously and feed them into the training process in a highly performant manner.

In the beginning of a training run, the `data_loader` will check in the specified dataset directory whether image or video data is available for each domain. In the case of video data, the `video_preprocessor` will extract the frames from all videos into a frame directory if not done in a previous run. The `data_loader` then prepares a list of all frame image paths, groups them into a tensor of frame sequences and generates a `tf.data.Dataset` with it. At runtime, the `tf.data` pipeline will then take care of selecting random batches and loading the actual image data. We found this solution to be just as fast as preloading the entire data set while saving a significant amount of system memory.

4.3.2. Training Balancing

While computing the averaged training balancer values (Chapter 3.4.4) is pretty straight forward, deactivating training for discriminators turned out to be a trickier problem. Simply excluding the corresponding discriminator optimisers from the `Session.run(...)` turned out to trigger the TensorFlow runtime to rebuild and re-optimise the entire compute graph. This ultimately caused minute-long hick-ups every time training was turned on or off for any discriminator.

Ultimately we use `tf.cond` which allows to conditionally branch between two input tensors at runtime. Since both branches will be evaluated regardless of which one will be passed on, we need another workaround to prevent the optimiser from being called. The solution is to move the optimiser into a `tf.control_dependencies` that nests the output of one branch. This way, the optimiser is only evaluated if the corresponding branch value is used.

4.3.3. TensorBoard Summary

To better monitor and analyse the training process, we made extensive use of TensorBoard. We track all full and partial losses for both generators and discriminators, model parameters such as the current identity loss and temporal loss weights and training balancer val-

ues, as well as all raw discriminator outputs as `tf.summary.scalar`s. Additionally, we log `tf.summary.image`s for raw, translated and full cycle images. Since temporal consistency is a main concern of our training, we also include motion compensated versions of the outputs as well pixel-wise difference images. For better tracking and reproducability of results, we also include information about the training run as a `tf.summary.text`, including the current git branch and commit as well as the user-defined training configuration.

4.4. Inference

In order to facilitate testing our trained model on different sets of test data, we wrote an entire inference pipeline from scratch. Just like the training pipeline, it works for both image and video inputs and outputs.

In the original implementation, the training process was orchestrated by a `tf.train.Supervisor`. While this automatically takes care of many things for the user, it is limited to only one `tf.train.Saver` that saves the entire model. For inference however, we only need the pre-trained generator weights, and loading the entire model thus posed a quite significant overhead. To solve this problem, we got rid of the `tf.train.Supervisor` and equipped every generator, discriminator and FNet with it's own custom `Saver`. The `Saver`s provide functionality to automatically load and store network weights from and to a save directory. For training, the `Saver`s will automatically create correct directory structures within the save directory and can also initialise from external checkpoints if needed.

We built a custom `InferenceMachine` class that contains all tricycleGAN components needed for inference, namely the two generators and the FNet. It automatically takes care of loading the network weights and also keeps a state of the last generated result. This allows the user to easily test different models on different data both frame by frame or as a full batch of frames.

5. Training

To validate our model, we tested its performance with different datasets in diverse settings. The following chapter will describe the tested datasets, the training parameters that were used as well as our training hardware.

5.1. Datasets

Compared to image-to-image translation, large, high-quality labeled datasets are hard to come by in video-to-video translation. We therefore test our model with four datasets: horse to zebra, shaded low-res to high-res smoke, unshaded simulation to real smoke and Obama to Trump.

5.1.1. Horse to Zebra

Paying homage to one of cycleGAN's [44] most impressive use cases, we try to bring the "Horse to Zebra" dataset into the realm of videos. While sourcing labelled images of wild horses and zebras is pretty straight forward with publicly available datasets as Image-Net [8] and COCO [20], only a few, rudimentary datasets are available for labelled video data.

The Calvin Research YouTube-Objects dataset [26] contains 720000 video frames (around 7 hours) from 10 different categories, including horses. Google also created their own youtube dataset: The YouTube-8M dataset [1] contains 6.1 million videos from 3862 classes, including horses and zebras. While both datasets mentioned above provide a good starting point, their labels are rather imprecise in many cases. As a consequence, we end up with many frames or even entire videos that don't contain horses or zebras. This is especially problematic for the tricycleGAN, since we found it to be quite sensitive to poor data. Mode collapse is a commonly seen consequence in these cases. By hand-selecting relevant video for both domains and manually discarding irrelevant frames, we managed to gather around 55000 relevant frames per domain, exclusively sourced from YouTube.

5.1.2. Shaded Low-Res to High-Res Smoke Simulation

For visual effects, creating high-detail fluid simulations traditionally comes at a high computational cost. One possible approach to creating high fidelity renderings in reasonable time is to start with a low-resolution rendering and add details in a post processing step. This is possible since the ultimate goal is to create believable 2D renderings rather than a fully realistic 3D simulation.

We will use the tricycleGAN to learn this post-processing function. In order to do so, we generated two sets of randomised 3D smoke simulations using MantaFlow. The low-resolution set is simulated with a grid resolution of 64^3, the high-resolution one at 256^3. For each of

Figure 18 Sample frames from the horse (left) to zebra (right) dataset.

them, we generate around 20 simulations. Every simulation contains around 200 frames using static time-steps. We randomly initialize a certain number of smoke inflow regions, another set of velocity inflows, and a randomized buoyancy force. After simulation, we render our simulations using PBRT to generate 512x512 sequences containing 120 frames per domain. Frames in early stage are skipped because latter frames are more natural and contain more interesting fluid motion.

5.1.3. Unshaded Simulation to Real Smoke

Since we work with only visual information, one tempting idea is to directly transform a preview rendering into something photo-realistic using real world recordings of smoke. We focus on unshaded smoke plumes in front of a plain background. The real smoke sequences were captured using a smoke machine and a Raspberry Pi camera setup as described by Eckert et al. [9]. Using a similar approach as in subsection 5.1.2, we also carefully created a set of low-resolution smoke simulations to roughly match the complexity and colour schemes of the real data. Ultimately, we again end up with 20 sequences with 120 frames each for both domains. This scenario is very challenging because the real smoke captures contain vivid fluid motions with many vortices and high-frequency details, while the low-resolution simulation contains strong numerical viscosity and details are limited by the simulation resolution.

5.1.4. Obama to Trump

We finally demonstrate the versatility of our model by taking on face-to-face translation. Since the human vision is extremely sensitive to faces, producing believable spatial and temporal features is even harder.

For this task, we use the Donald Trump to Barack Obama dataset that is publicly available on the recycleGAN [2] project page. It contains 256x256 frames for Obama (3.7k) and Trump (4.7k) sourced from "publicly available videos of various public figures " [2]. The faces were detected and aligned using the OpenPose Library [6].

5.2. Training Parameters and Hardware

To optimise our generators and discriminators, we use the Adam Optimiser [18] with a learning rate of 0.0001 and $(\beta_1, \beta_2) = (0.5, 0.999)$. We feed the training images in batches of 1 and in a resolution of 256x256 in order to pick up as many small scale details as possible.

If not explicitly specified otherwise, the results in Chapter 6 were achieved with traditional bottleneck generators trained for 100k iterations with a decaying learning rate in the last 10k iterations. Identity loss was faded out after the first 1000 iterations over the course of 1500 iterations, the temporal losses were faded in after 2k iterations over the course of 4k iterations and the style loss was faded out after 50k iterations over the course of 10k iterations. Table 1 lists the used loss weights.

Figure 19 Sample frames from the low-res (left) to high-res (right) smoke simultion dataset.

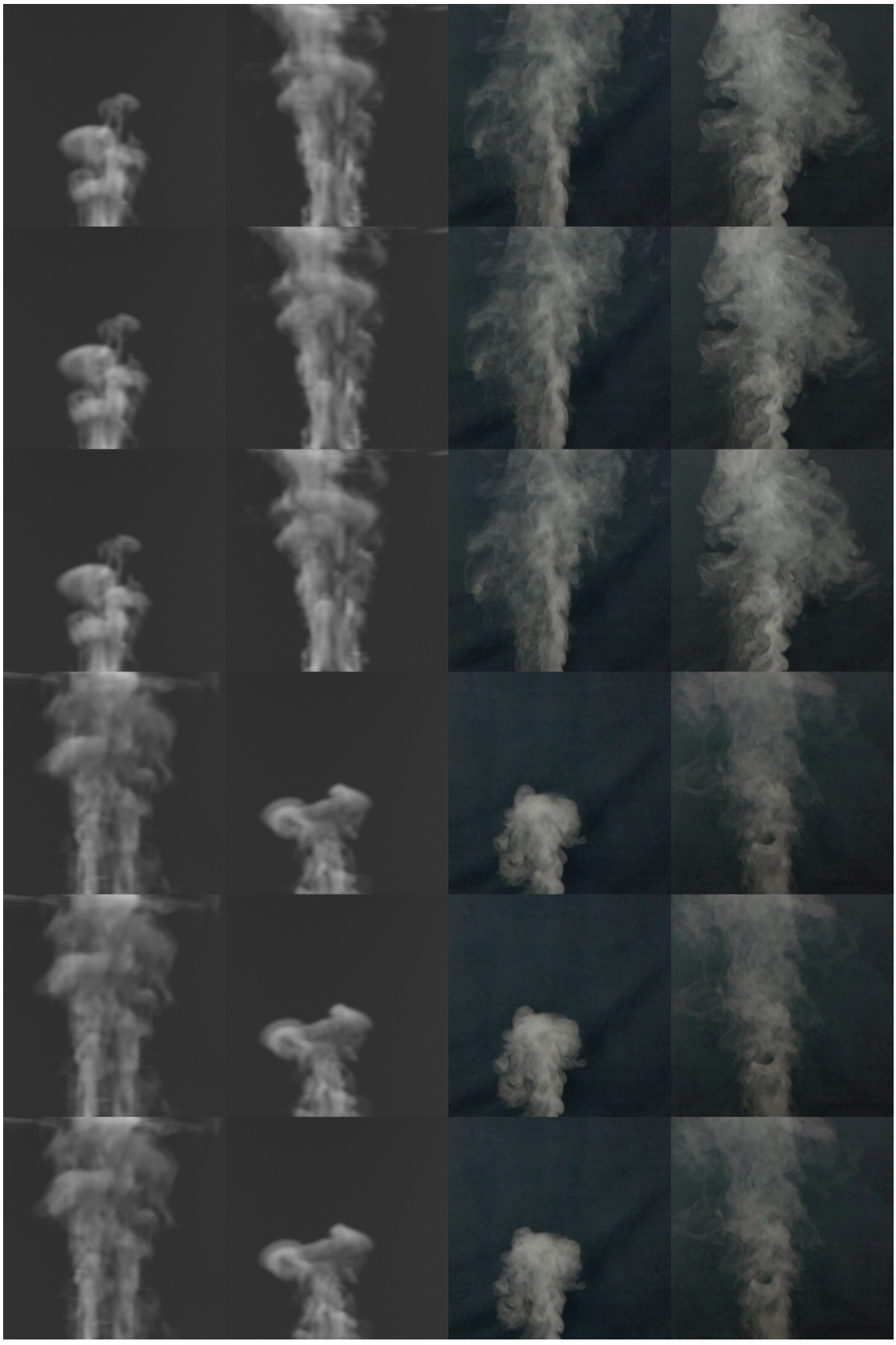


Figure 20 Sample frames from the simulation (left) to real smoke (right) dataset.

Figure 21 Sample frames from the Obama (left) to Trump (right) dataset.

Loss	Weight
$\lambda_{temporal}$	0.5
λ_{cycle}	10
$\lambda_{identity}$	10
λ_{style}	100000
$\lambda_{PingPong}$	100

Table 1 Loss weights for training

We run our training on an Ubuntu 16.4 LTS machine, featuring an Nvidia Titan RTX GPU (24GB VRAM), an Intel Core i9 9900k CPU and 128GB of system memory. We run our model in Python 3.6.4 environment, with TensorFlow 1.13.1 running on CUDA 10.

6. Results and Discussion

In the following, we will compare the cycleGAN to different versions of the tricycleGAN in an ablation study and highlight results achieved with the various fields of application introduced in section 5.1. We will display frame sequences and difference maps to showcase temporal features, however we recommend viewing the supplementary video material that can be found on the project page [23] for the best temporal effect.

6.1. Ablation Study

We first compare our full model to the original cycleGAN [44] as well as an intermediate version of the tricycleGAN: The tricycleGAN Light is trained with a simplified objective, lacking the Ping-Pong loss and the discriminator style loss.

Figure 22 shows a zoomed-in time sequence of from the shaded low-res to high-res smoke simulation dataset. We use this problem domain to highlight the model differences since temporal inconsistencies become particularly noticeable in comparison to seemingly simple spatial features. To better visualise temporal inconsistencies, we also provide absolute difference maps to previous frames in Figure 23.

While the results from the original cycleGAN don't seem too inconsistent at first glance, looking at the difference map (Figure 23) reveals high-frequency inconsistencies that don't correspond to the direction of motion. Thanks to the frame-recurrent generators and the addition of the temporal discriminators, the tricycleGAN Light manages to achieve far greater temporal consistency in its outputs. However, since the generators now have to trade off spatial against temporal features, the added details become far less spectacular and also smoother in comparison.

The addition of the discriminator style loss encourages the generation of more interesting spatial and temporal features (e.g. the swirl in Figure 22) while also sharpening the results even more. At the same time, the Ping-Pong loss ensures long-term consistency, thus reducing drifting of features over multiple frames. Compared to the original cycleGAN, the tricycleGAN produces comparably interesting spatial features while also reproducing the temporal features of the target domain. The severity of the added temporal consistency is especially visible in high-resolution data, as shown in Figure 24. While tricycleGAN generates difference maps mostly corresponding to the flow of the input, cycleGAN causes high-frequency temporal inconsistencies throughout the entire image.

To highlight the importance of the Ping-Pong loss, we also compare results with and without the Ping-Pong loss. For this we use the simulation-to-real smoke dataset due to its char-

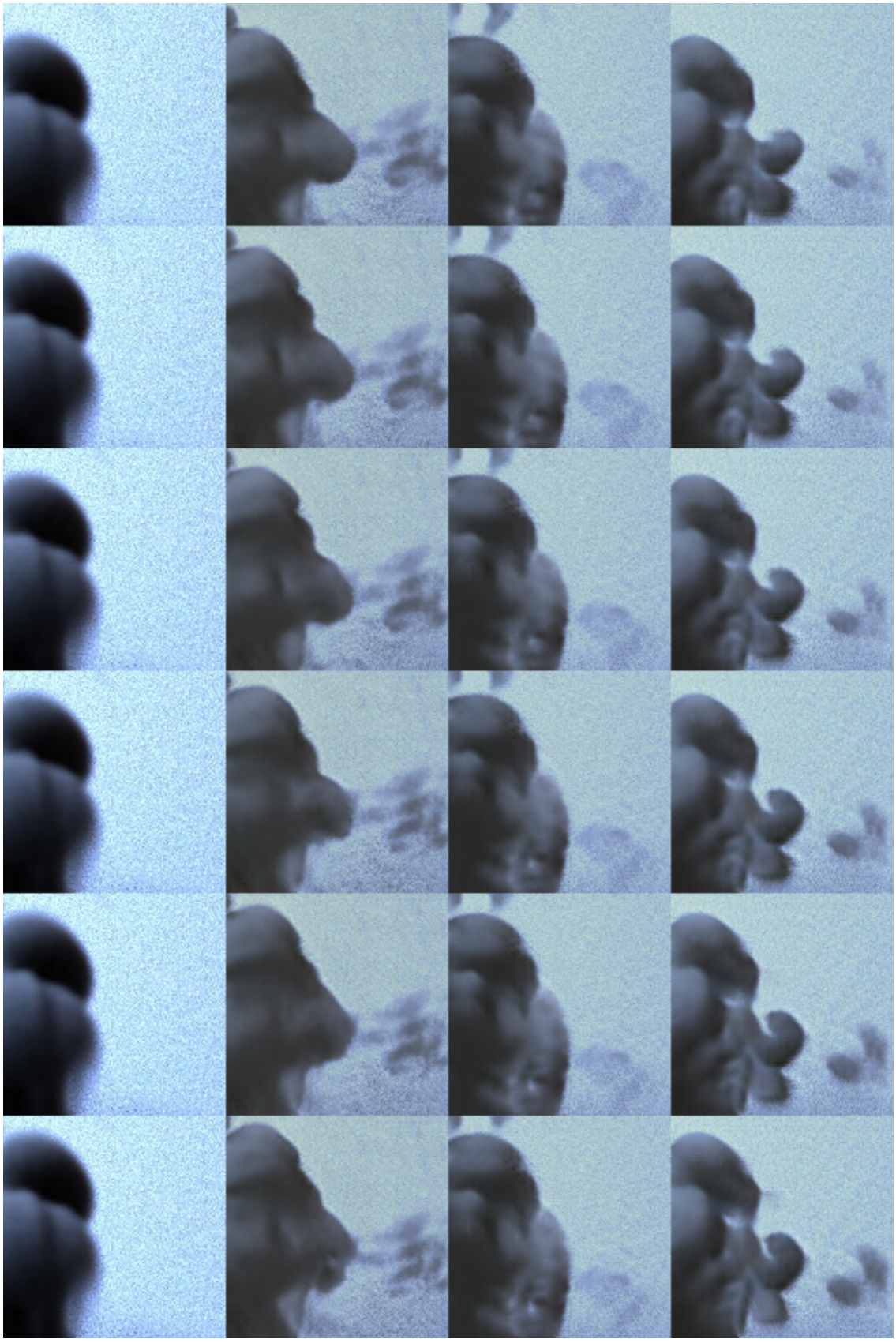


Figure 22 Result zoom-ins from the low-res to high-res datasets. From left to right: input, cycleGAN, tricycleGAN light, full tricycleGAN.

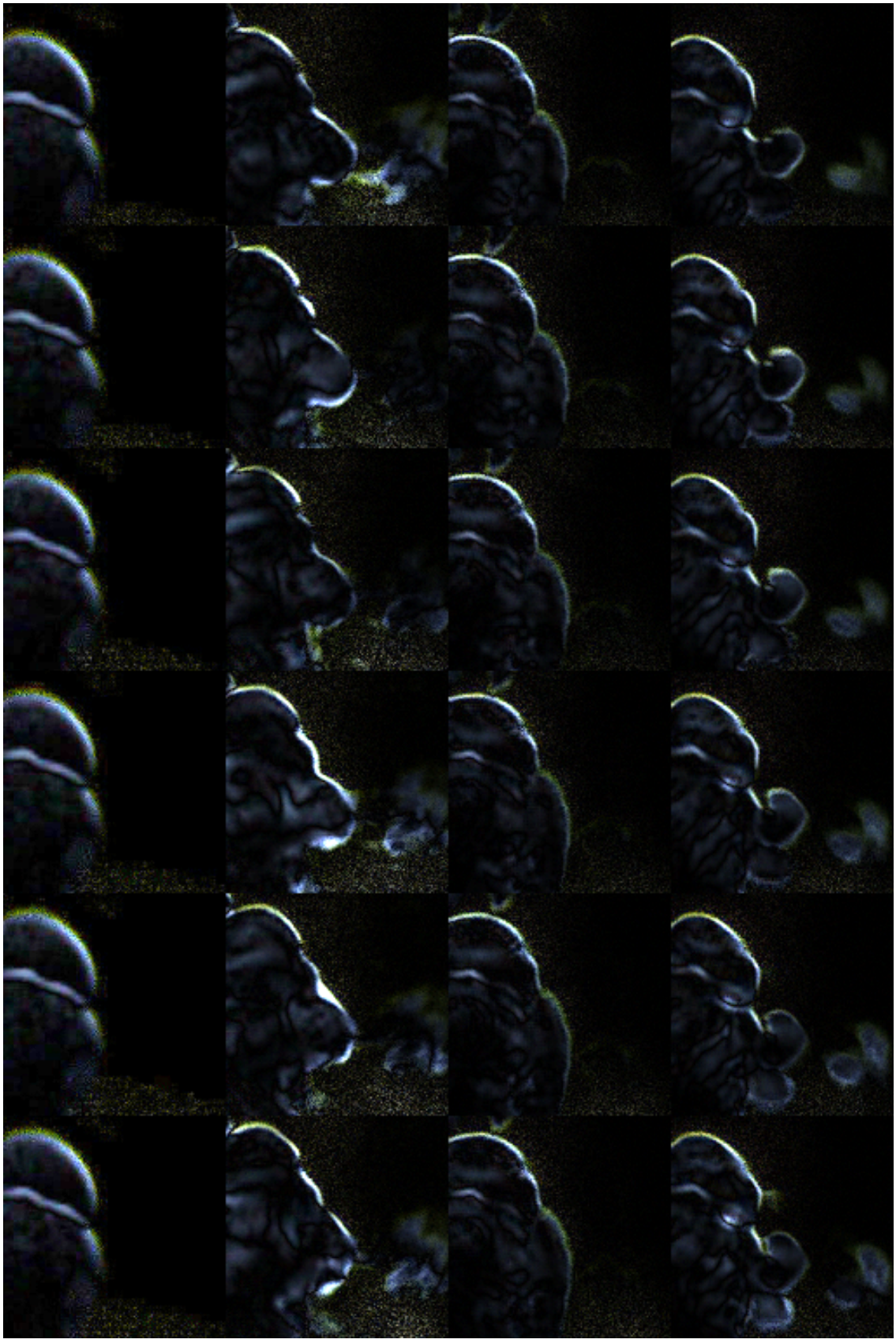


Figure 23 Pixel-wise absolute differences to the previous frame of our results in Figure 22. The differences were amplified by a factor of 4 to improve visibility.

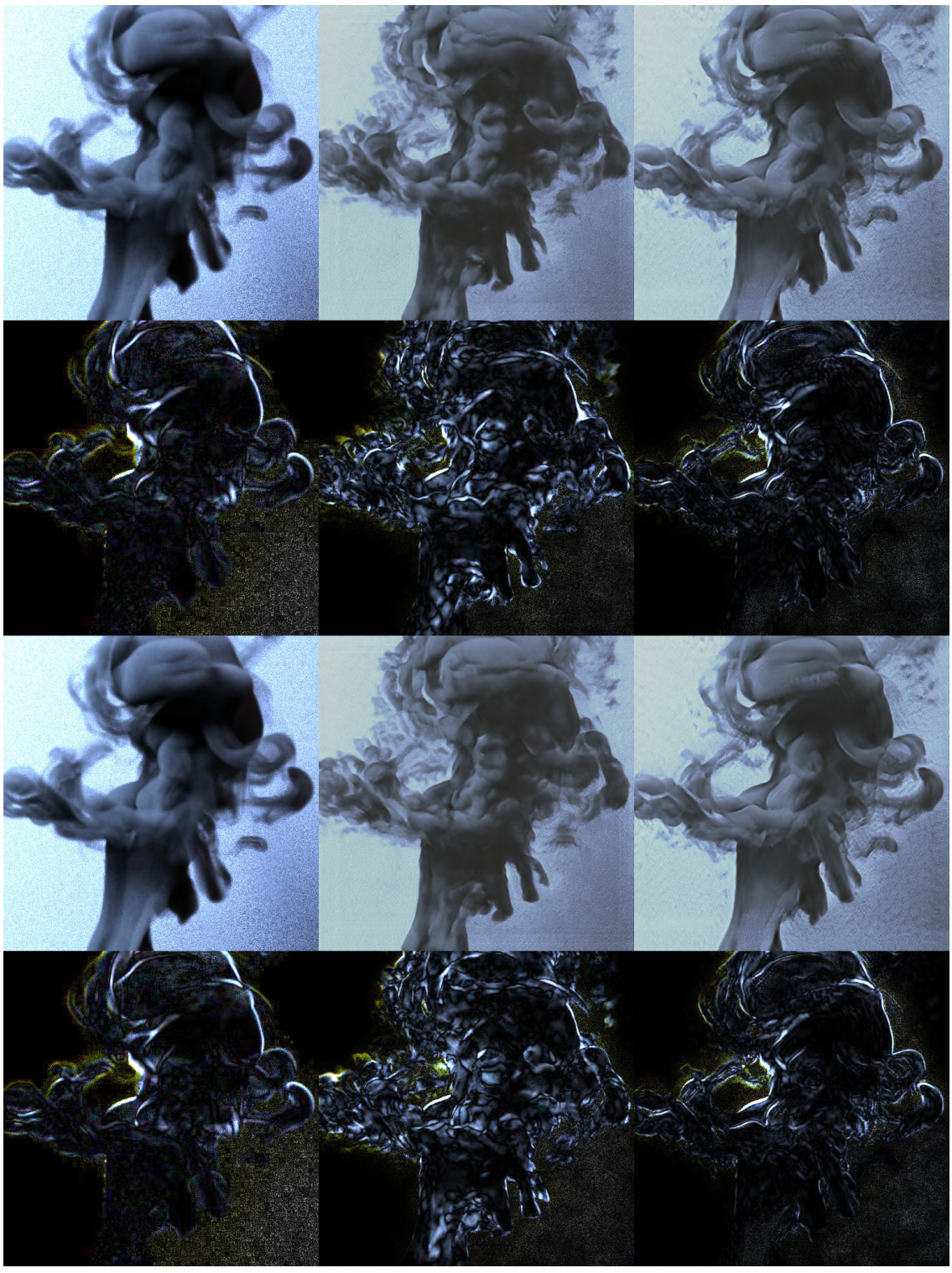


Figure 24 Results from the low-res to high-res datasets and pixelwise differences to the next frame. From left to right: input, cycleGAN, tricycleGAN.

Figure 25 Results from the simulation to real smoke datasets with a frame step size of 5. From left to right: input, tricycleGAN w/o Ping-Pong loss, tricycleGAN with Ping-Pong loss and a frame sequence length of 6.

acteristic and easily trackable smoke swirls. Figure 25 shows results without (middle) and with (right) Ping-Pong loss with a frame step size of 5. While the tricycleGAN produces more interesting details without the Ping-Pong loss, they fade away over multiple frames and are replaced by new ones. With the Ping-Pong loss in place, the generator produces less but more persistent features. An interesting side effect is that the generator implicitly learns a mapping closer to the actual physical properties due to the added constraint of long-term consistency.

6.2. Fields of Application

6.2.1. Horse to Zebra

Figure 26 shows our results for the horse to zebra dataset. While the results definitely benefit from the unet generator, the background still seems rather flat and the generator struggles to cleanly segment the horses from the background. As a consequence, the zebra stripes often bleed into the background, leading to less satisfactory results.

We suspect these results to be a combination of poor data and and weak generators. While the dataset contains around an hour of footage, the actual number of unique scenes is most likely still too low to create a sufficient amount of entropy for the problem domain. Additionally, it seems that the quality of selected clips still leaves some room for improvement. Finally, it is questionable whether the rather complex task of transforming horses to zebras in a temporally coherent fashion can be performed by a simple fully connected unet architecture.

6.2.2. Shaded Low-Res to High-Res Smoke Simulation

Given a low-resolution sample sequence, the tricycleGAN manages to generate convincing high-resolution outputs in a temporally consistent fashion (Figure 27). It does so by sharpening the volume edges and also adding in interesting new small scale details like smoke swirls (Figure 22).

One interesting observation here is the perceived lack of contrast in the outputs. We found this effect to be particularly strong in early training stages and diminish later on. As a reason for this, we suspect the L_2-norm in the Ping-Pong Loss to encourage the outputs to drift towards the trivial optimum of constant 0 (grey frames). While the generators find a more desirable and less trivial solution to long-term consistency during the training, more advanced loss terms for image similarity could be beneficial here.

6.2.3. Unshaded Simulation to Real Smoke

For the task of simulation to real smoke, we found tricycleGAN to produce deceptively real results (Figure 28). Not only does it generally sharpen the contours of the smoke plumes, but it also manages to add in temporally consistent details that create a sense of detail and depth.

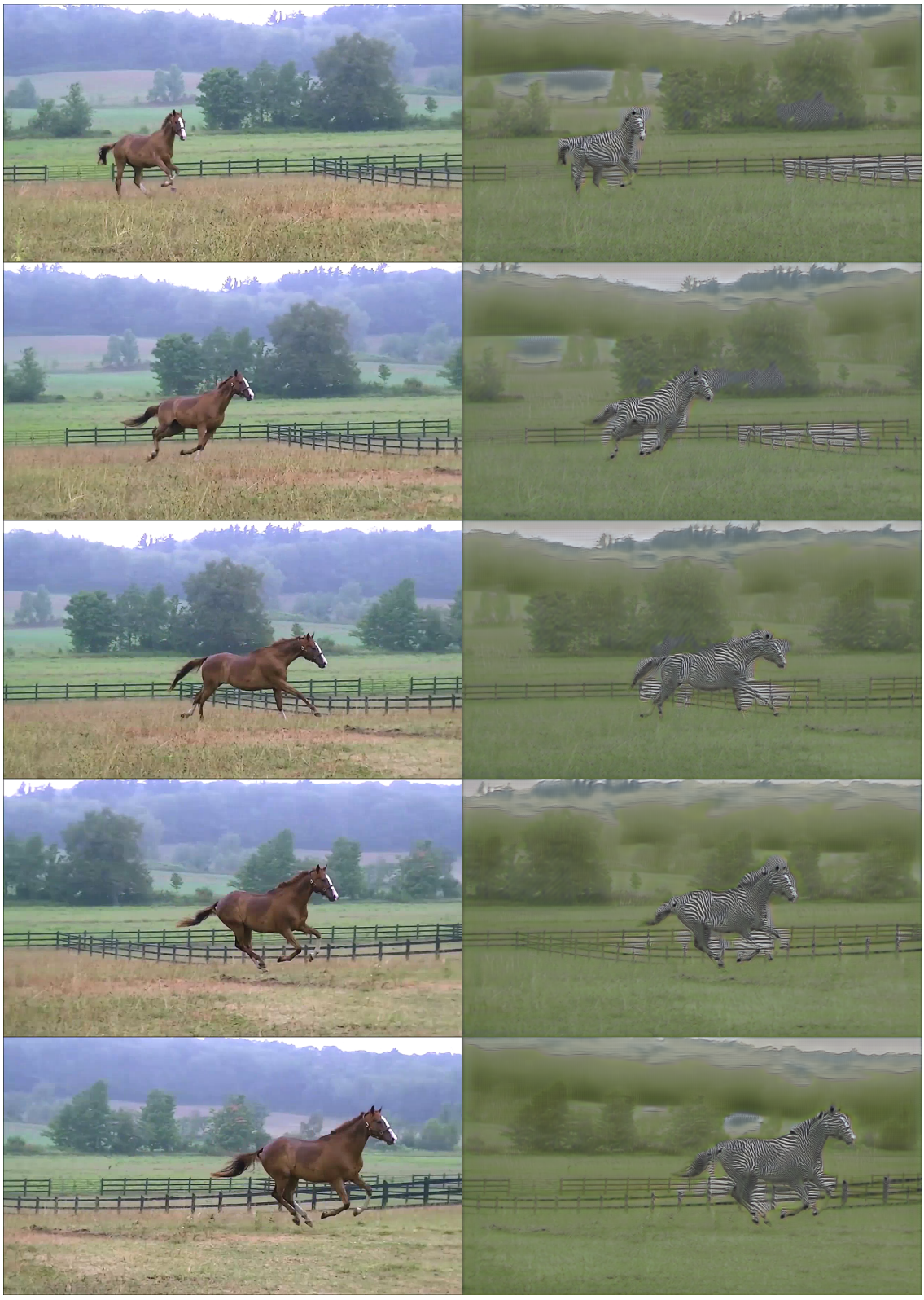


Figure 26 Results from the Horse to Zebra dataset: Input (left) and Output (right).

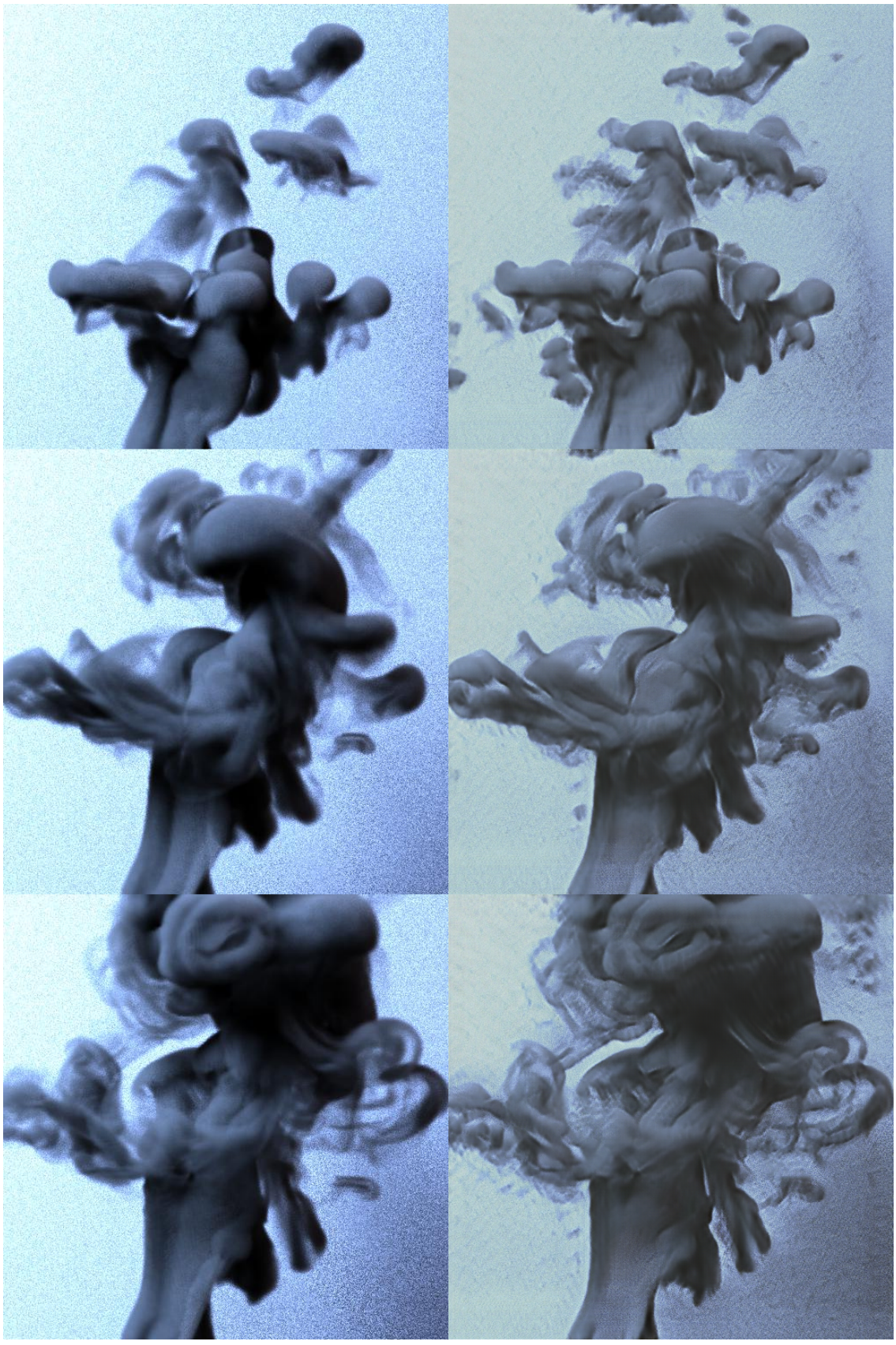


Figure 27 Results from the Low-Res to High-Res Smoke Simulation dataset: Input (left) and Output (right).

We found this problem domain to particularly benefit from the Ping-Pong loss as it ensures the long-term persistence of smoke swirls that is fundamental for believable results in full motion.

6.2.4. Obama to Trump

The probably most surprising results were achieved in our face-to-face translation task. The generated results for both Obama to Trump (Figure 29) and Trump to Obama (Figure 30) are practically indistinguishable from the training data and still manage to represent a meaningful mapping from inputs to outputs.

This is especially remarkable given the fully convolutional generator architecture which is rather uncommon in face-to-face applications such as deepfakes. A potential reason for these results is the limited amount of entropy in the training data, ultimately allowing the generators to overfit to a very limited set of features.

6.3. Performance

We found that training tricycleGAN for 100k iterations posed the best quality/performance trade-off. For the full model, training runs typically take 30h on the Titan RTX machine (section 5.2). Compared to 3.5h that our non-temporal cycleGAN implementation takes, it is a seemingly high price to pay for temporal consistency. We found the slowdown to be mostly dominated by the motion-compensation and frame-recurrent iterations for the full frame sequence. As a consequence training time correlates almost linearly with the length of the frame sequence length.

Comparing the speed of tricycleGAN to vid2vid [40] that typically takes 10 days on a quad-GPU machine for a full training run, the speed advantage seems obvious. However, the difference in resolution (256x256 vs. 2048x1024) and fidelity make a fair comparison impossible.

While training surely requires both powerful hardware and a significant amount of time, it poses a one-time cost. Inference on the other hand is comparatively quick, also thanks to the fact that only one frame-recurrent generator step is required for each frame. Averaged over 500 frames, we measure a frame time of 60 ms per 512x512 frame on a Nvidia GTX 1080ti consumer grade GPU.

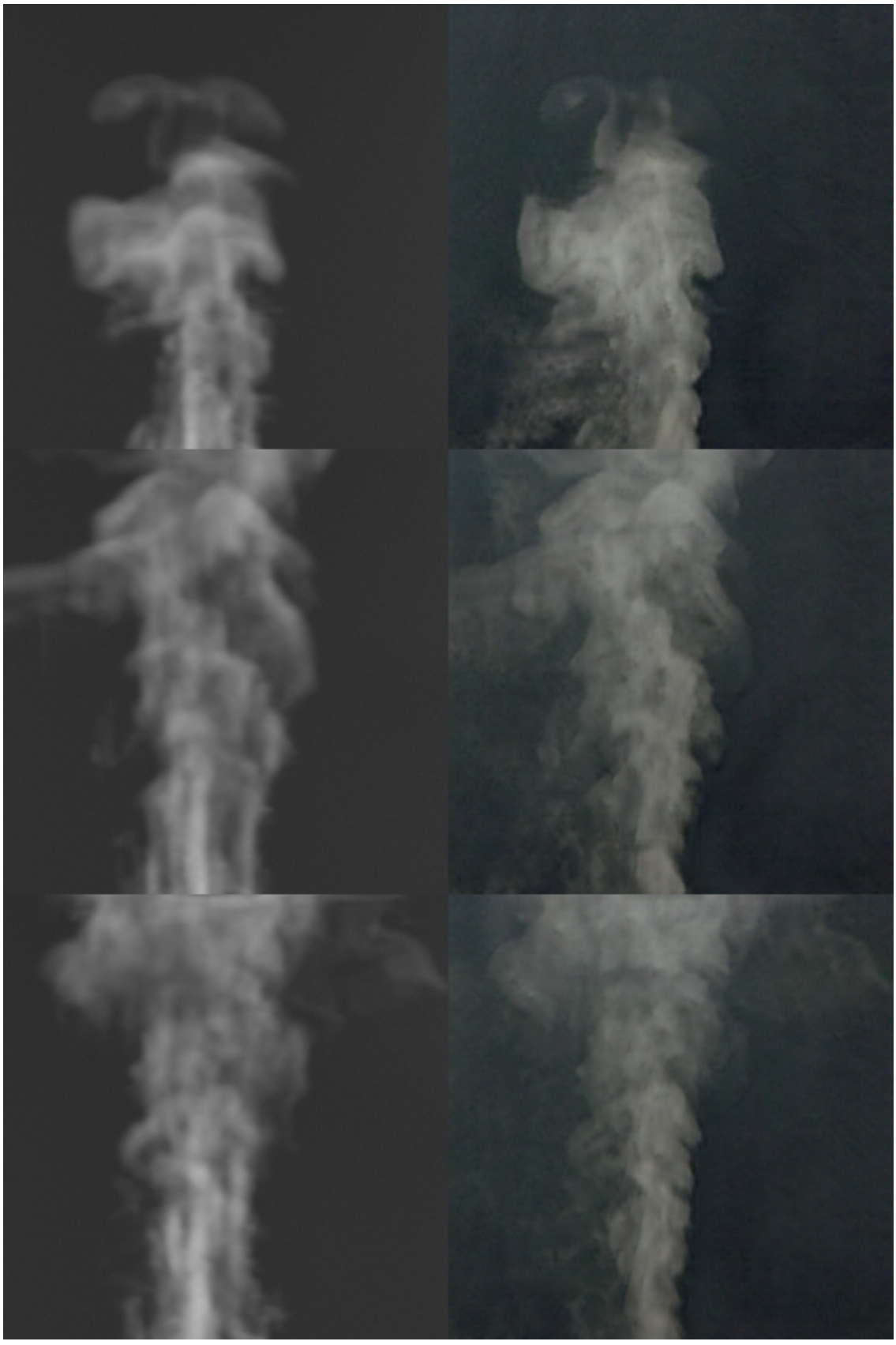


Figure 28 Results from the Simulation to Real Smoke dataset: Input (left) and Output (right).

Figure 29 Results from the Obama-to-Trump dataset: Translation from Barack Obama (left) to Donald Trump (right).

Figure 30 Results from the Obama-to-Trump dataset: Translation from Donald Trump (left) to Barack Obama (right).

7. Conclusion and Future Work

In this thesis we present the tricycleGAN as a solution for unpaired video-to-video translation. For this, we extend the cycleGAN with a temporal discriminator and frame-recurrent generators to ensure temporal consistency. We strengthen our model with an auxiliary Ping-Pong loss for long-term consistency and speed up convergence with a novel discriminator style loss. We show that our model generates convincing, temporally consistent results for a multitude of datasets. While we focus on unpaired video-to-video translation, our process of adding temporal consistency to a previously non-temporal model is generally applicable thanks to its adversarial nature.

While the tricycleGAN represents a valuable temporal extension of the cycleGAN, it is still far from perfect. The generators with their simple bottleneck architecture struggle with generalising more complex tasks such as translating horses to zebras. An interesting next step would be to experiment with more complex but yet lightweight generator architectures. Something like a pretrained mobilenet-unet segmentation network [34] could significantly speed up training and increase the quality of the results while maintaining comparable inference speeds. On the other side of the adversarial game, the training would also benefit from more powerful discriminators and transfer learning.

Our shared temporal discriminator architecture is based on the assumption that both domains share the same temporal characteristics. As claimed by Bansal et al. [2], this is not always necessarily the case. While we still manage to produce convincing results, capturing the specific spatio-temporal characteristics for each domain would surely be an interesting direction for further research.

As discussed in subsection 6.2.2, some results suffer from reduced contrast which we suspect to be a consequence of the L_2 formulation of the Ping-Pong Loss. One direction of future work could be to experiment with different variations of the Ping-Pong Loss, e.g. based on structural similarity [41] or adversarial loss terms.

Being a data driven solution, the main limitation remains the quality of the training data. A common result are poorly generalised models that struggle with new inference data as a consequence of overfitting to the features and scale of the training data. To this day, high-quality labelled video data sets are hard to come by. While investing more time in compiling larger and better datasets is surely inevitable in the long run, we could still increase the robustness of our model by increasing the extend of augmentation to our data.

List of Figures

List of Tables

Bibliography

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

[2] Aayush Bansal, Shugao Ma, Deva Ramanan, and Yaser Sheikh. Recycle-gan: Unsupervised video retargeting. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 119–135, 2018.

[3] Dina Bashkirova, Ben Usman, and Kate Saenko. Unsupervised video-to-video translation. *arXiv preprint arXiv:1806.03698*, 2018.

[4] Mikael Boden. A guide to recurrent neural networks and backpropagation. *the Dallas project*, 2002.

[5] Kaidi Cao, Jing Liao, and Lu Yuan. Carigans: Unpaired photo-to-caricature translation, 2018.

[6] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7291–7299, 2017.

[7] Mengyu Chu, You Xie, Laura Leal-Taixé, and Nils Thuerey. Temporally coherent gans for video super-resolution (tecogan). *arXiv preprint arXiv:1811.09393*, 2018.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[9] M-L Eckert, Wolfgang Heidrich, and Nils Thuerey. Coupled fluid density and motion from single views. In *Computer Graphics Forum*, volume 37, pages 47–58. Wiley Online Library, 2018.

[10] Chang Gao, Derun Gu, Fangjun Zhang, and Yizhou Yu. Reconet: Real-time coherent video style transfer network. *arXiv preprint arXiv:1807.01197*, 2018.

[11] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil

Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[13] Haozhi Huang, Hao Wang, Wenhan Luo, Lin Ma, Wenhao Jiang, Xiaolong Zhu, Zhifeng Li, and Wei Liu. Real-time neural style transfer for videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 783–791, 2017.

[14] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2462–2470, 2017.

[15] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016.

[16] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[17] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1857–1865. JMLR. org, 2017.

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Youngwoon Lee and Honghua Dong. Cyclegan implementation in tensorflow. `https://github.com/gitlimlab/CycleGAN-Tensorflow`, 2017.

[20] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[21] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. In *Advances in neural information processing systems*, pages 469–477, 2016.

[22] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[23] Jonas Mayer. A temporally coherent cyclegan. `https://github.com/mayerjRRR/CycleGAN-Tensorflow`, 2019.

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

[25] OpenAI. Saving memory using gradient-checkpointing. `https://github.com/cybertronai/gradient-checkpointing`, 2017.

[26] Alessandro Prest, Vicky Kalogeiton, Christian Leistner, Javier Civera, Cordelia Schmid, and Vittorio Ferrari. Youtube-objects dataset v2.0. *ac. uk/datasets/youtube-objects-dataset. University of Edinburgh (CALVIN), INRIA Grenoble (LEAR), ETH Zurich (CALVIN)*, 2014.

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

[28] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[29] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos. In *German Conference on Pattern Recognition*, pages 26–36. Springer, 2016.

[30] Masaki Saito, Eiichi Matsumoto, and Shunta Saito. Temporal generative adversarial nets with singular value clipping. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2830–2839, 2017.

[31] Mehdi SM Sajjadi, Raviteja Vemulapalli, and Matthew Brown. Frame-recurrent video super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6626–6634, 2018.

[32] Artsiom Sanakoyeu, Dmytro Kotovenko, Sabine Lang, and Bjorn Ommer. A style-aware content loss for real-time hd style transfer. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 698–714, 2018.

[33] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2107–2116, 2017.

[34] Mennatullah Siam, Mostafa Gamal, Moemen Abdel-Razek, Senthil Yogamani, and Martin Jagersand. Rtseg: Real-time semantic segmentation comparative study. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 1603–1607. IEEE, 2018.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[36] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200*, 2016.

[37] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1526–1535, 2018.

[38] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, volume 1, page 4, 2016.

[39] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[40] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *CoRR*, abs/1808.06601, 2018.

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

[42] You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. *ACM Transactions on Graphics (TOG)*, 37(4):95, 2018.

[43] Rujie Yin. Content aware neural style transfer. *arXiv preprint arXiv:1601.04568*, 2016.

[44] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

A. Appendix

A.1. Memory Management in TensorFlow

For training a neural network using gradient descent based optimisation, the gradients of the weights with respect to the total loss need to be calculated. This is done with back-propagation.

In TensorFlow each training step consists of a (feed) forward pass to calculate the activations of the individual layers, followed by backwards pass where gradients corresponding to the activations are calculated. The calculations of the gradients in the backwards pass require the intermediate activations of the forward pass. TensorFlow's standard strategy is to thus keep all the intermediate values in memory. This comes at a great advantage of only needing to compute said values once, but on the flip side consumes a significant amount of memory. While other backpropagation strategies exist, they are only available through third-party tools in TensorFlow [25].

We calculated that the trainable weights of our discriminators amount to around 10.6 MB depending on the number of input channels. Even for a consumer grade GPU (11GB Memory), this would account for less then one percent of memory for all three discriminators. In comparison, for a typical input of size 256x256, the size of the intermediate activations in the forward pass amount to 7.5 MB. Since we do multiple discriminator and even more generator runs per training step, it becomes obvious that the effect of only reducing trainable weights becomes insignificant in the big picture.

A.2. Training Balancing

To determine the performance benefit of the training balancing, we compared two versions of the tricycleGAN: One version without training balancing and one versison with training balancing cranked up to the maximum, where the discriminators are never trained. Even in this best case, we only gained approximately 2 seconds per hour of training.

We suspect the poor performance benefit to be a result of multiple factors:

1. The training is already fairly balanced due to the generators' diverse auxiliary losses that speed up generator convergence. In practice, the discriminators would still train most of the time.
2. Compared to the complex generator training objective, training the discriminator is rather simple. The one loss term only requires two forward passes of the discriminator

which is comparatively quick. Thus the time for training the discriminators becomes negligible in the big picture.

3. The implementation-specific overhead of switching on and off training using `tf.cond` statements neutralises the already slim performance benefit even further.

As a conclusion, it seems unreasonable in our case to weaken the discriminators for close to no performance benefit.