Autolume-Live: An Interface for Live Visual Performances using GANs

by

Jonas Frederik Kraasch

B.Sc., University of Osnabrueck, 2020

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Interactive Arts and Technology
Faculty of Communication, Art and Technology

Declaration of Committee

Name:	**Jonas Frederik Kraasch**
Degree:	**Master of Science**
Thesis title:	**Autolume-Live: An Interface for Live Visual Performances using GANs**
Committee:	**Chair:** Wolfgang Stuerzlinger Professor, Interactive Arts and Technology
	Philippe Pasquier Supervisor Professor, Interactive Arts and Technology
	Yagiz Aksoy Committee Member Assistant Professor, Computing Science
	Steve DiPaola Committee Member Professor, Interactive Arts and Technology
	Kate Hennessy Examiner Associate Professor, Interactive Arts and Technology

Abstract

As of writing this, Deep Learning models are able to generate images with high-fidelity. They are becoming an integral tool for creative expression and democratizing creativity. With the introduction of prompt-based generative systems we have seen that reducing the skill floor to accessing leads to an explosion of possibilities for creative practices. But, most creative AI systems are still inaccessible to users that are not trained in coding or the respective research domain. We present *Autolume-Live*, an interface created to make a deep learning model a malleable material for live performances. We aggregate and link together creative practices with current state-of-the-art frameworks from model compression, generative AI and interpretable AI, resulting an interactive tool that opens up a new paradigm for audio-visual performances and visual performance in general. We show that by shifting the focus on creating a tool for a valid user scenario (VJing), it is possible to use current technologies to run StyleGAN2 at a framerate of 30FPS (1024px) to 50FPS (512px) without compressing the model and 40FPS and 60FPS after compression in a tool for live performances.
Autolume-Live can be used on its own to train and compress models, find salient features and create audio-visual mappings. Furthermore, we include networking protocols that allow users to control the tool via Open-Sound-Control. Hence, they can interface *Autolume-Live's* parameters with their preferred software and hardware, e.g. TouchDesigner, PureData or MaxMSP. Lastly, the resulting video stream is sent out as an NDI-stream allowing for post-processing in software such as OBS or TouchDesigner. We have showcased the work in two installations during the COVID19 pandemic showcasing the generative power (offline) and we showcase setups for live performances in this work. Beyond this thesis, there is continued development on new features and work on both interactive installations and audio visual pieces.

Keywords: Deep Learning; XAI; human-centered AI; AI Interface; GAN; Audio-Visual performace

Dedication

I dedicate this thesis to my family that supports me even when separated by the ocean. My dog Pelle. And my partner Andrew, for keeping me going and feeding me through an intense month of work.

Acknowledgements

As a settler I want to acknowledge my studies have taken place on the unceded traditional territories of the xʷməθkʷəy̓əm (Musqueam), Sḵwx̱wú7mesh Úxwumixw (Squamish), səl̓ilw̓ətaʔɬ (Tsleil-Waututh), q̓íc̓əy̓ (Katzie), kʷikʷəƛ̓əm (Kwikwetlem), Qayqayt, Kwantlen, Semiahmoo and Tsawwassen peoples.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Often music and audio performances are coupled with visuals. Examples are music videos, and sets from Visual Jockeys (VJs) during live performances in clubs or at festivals. These visuals create a multidimensional experience for the audience, which can be used to tell a story or as a reference guiding people through the performance, enhancing the perception of certain musical features. Creating the respective visuals can be a taxing task, due to its multi-modal nature. We want to address this problem using Deep Generative Models.

Deep Generative Models (DGM) are Deep Learning (DL) systems that generate new samples from an underlying distribution. Researchers and artists use these systems to automate and augment creative processes. For example, in visual practices, artists use Generative Adversarial Networks (GAN) for static [9] and interactive installations [10].
Furthermore, multimodal approaches have been developed. Images can be generated based on text prompts [11]. Videos can be generated given a music track [4, 5, 6]. These systems both arise from academic research and artistic practices. While newer work in prompt based generation and work from Huggingface [12] improved accessibility by allowing the deployment of systems for users through interfaces or basic text prompts [11, 13], most DGM either lack documentation or a user interface, which makes the systems inaccessible for new users. Hence, artists need expertise in coding and Deep Learning, which gate-keeps artists from exploring computational creativity. The domain of explainable AI mostly focuses on expert-to-expert communication [14]. Furthermore, the systems that are accessible to users do not allow for live performances [13].

Reviewing these works a few research gaps arise:

- **Computational Complexity**: DGMs for video generation require multiple high-end graphics cards during training and rendering. Furthermore, their inference speeds are rather slow, which can create a detached creative process, where the user changes parameters and needs to wait for their results, which can makes it more difficult to iterate during the creative process.

- **Controllability**: Most of these systems either have no means of directly controlling the dynamics found in the videos.
- **Accessibility**: Both the academic and artistic work often lack documentation or can only be accessed with coding experience.
- **Data Complexity**: When using models that generate videos, they require a large database of videos with similar content, which is often not feasible to collect.

Offering up the question, if we can develop a co-creative tool for visual synthesis that closes those gaps.

We propose *Autolume-Live*, to our knowledge, the first live Music Visualizer and VJing software based on GANs [1], specifically StyleGAN2 [2]. Users can either load pretrained models or train new models. Our interface opens up the parameters of these models and adds further means of manipulating them based on GANSpace [8] and Network Bending [3].

Our current version of *Autolume-Live* has been iteratively developed based on the needs of Visual Jockeys [1] (VJ), as described by Hook et al [15], and thus we mean to push towards more usable AI [16]. Originally, users were only able to control the visuals with the incoming audio signal and predefined controls from a MIDI-Controller [7]. We have expanded that version to allow users to control *Autolume-Live* with any software that can process Open Sound Control (OSC) messages [17], which is commonly used in VJing and audio processing. This allows users to create pieces with any control signal they want, and not only audio.

We show that with modern graphic cards and developments in generative models, interpretable AI and compression techniques, it is possible to create an interface that allows artists to create live (interactive) visual performances.

We developed our tool through an iterative research through design process, where we reviewed the literature of Deep Generative Models and controllable image synthesis. Following we mapped out which features we can address with the which technological approach and implemented *Autolume-Live 1.0.* To review the system we created installations and explored and tested its functionality internally. Following the review we extracted changes we deemed important for its usability and usefulness and implemented those resulting in *Autolume-Live 2.0.* While iterating over the features for the second version we met up and rehearsed creating visuals for a live music feed and refined the system based on our experiences in the rehearsal setting.

[1]Visual Jockeys practice realtime visual performances. While these performances can happen on their own, they are most often linked to audio at concerts, clubs or festivals

The thesis is structured as follows. First, in our Background chapter, we lay out the domain of audio-reactive visuals, defining different approaches to audio-driven visuals and technologies used by VJs (Section 2.1). Following, we discuss concepts of human-centered AI, laying out the paradigm our work is residing in, i.e. creating an AI solution that is usable and useful AI for humans (Section 2.2). We then discuss different generative systems that could be used for our task and reason why we choose styleGAN2 (Section 2.3). Following the background, we discuss previous works that allow users to control Deep Generative Models (Section 3.1), these include live interfaces and low latency techniques for meaningful model manipulations. Furthermore, we lay out offline systems that generate audio-reactive visuals (Section 3.2). These systems are the closest to our work when it comes to the generative task, but they do not tackle real-time generation and require coding experience to be used. We lay out *Autolume-Live 1.0* (Section 4.1) followed by installations we have made with the system and reflections that inspired further iterations of our system (Figure 4.8). Afterwards, we discuss the revised version *Autolume-Live 2.0* (Section 4.3), followed by use-cases to showcase its capabilities. With all our work laid out we discuss our contributions, possible extensions and considerations regarding dataset aggregation (Chapter 5).

Chapter 2

Background

2.1 Audio-reactive Visual Systems

Before discussing our framework, we first have to understand the different types of real time audio-driven visual software.
First, there are Music Visualizers, which map musical features to parameters of a visual system. While Taylor et al. synthesize a complex scene with virtual characters and spaces that respond to audio features [18], commonly music is visualized by showing the live spectrogram in different formats.
Secondly, Video Synthesizers relate to the mapping of any signal to control visual generation. Most Music Visualizers and Video Synthesizers can be used offline and online. A specific application of these systems for live performance is VJing software. These generally combine the audio-reactive components of Music Visualizers and Video Synthesizers and add an interactive component that allows an artist to manipulate media of any kind live. By applying visual transformations or adjusting mappings, every performance can become a unique experience based on the performer, aka Visual Jockey (VJ).

In their study, Hook et al. explore the ways VJs adapt and appropriate technology to create visual performances through dialogue and the creation of a documentary film [15]. They focused on multiple artists and performances and extracted the essence of the practice for those artists. When it comes to VJing tools, there is a need for a physical, tangible, interface that responds with immediacy, and allows the user to accompany music live and improvise. The tool should be re-configurable and allow the artist to control a variety of parameters, while keeping usability in mind. The benefit of a physical interface, such as a MIDI controller, is that it can be seen as another instrument that can be incorporated into a performance. Lastly, the visuals should be responsive to its context and surrounding, e.g. the music, and every performance should have the possibility to be unique.

Touch Designer and Resolume are popular VJing software [19, 20]. By looking at current VJing practices, we see a trend to use video loops, shaders and generative algorithms. These software incorporate multiple levels of music analysis. Both low level features such as amplitude, onset strength, timbre etc., and high level features, for example affective features computed through sentiment analysis, can be used to modify parameters, such as color, shape and position. When it comes to VJing, response time is important, requiring a latency smaller then 50ms which comes to a framerate of 20FPS minimum. Hence, Touch Designer and Resolume focus on systems that are efficient and can process information with little latency. For that reason these applications have not used DL based generative models out of the box.

2.2 Human-Centered AI

With AI solutions becoming more sophisticated, they can address valid user scenarios. But, current DL-based systems are black-boxes. Users can not infer the process or ”reasoning” behind a system’s output. The field of explainable AI (XAI) addresses this problem [14]. They develop algorithms and interfaces that give users insights into DL models. In general, XAI research addresses other experts as their audience, either in the representation of the explanation or the accessibility of the interface [21, 22]. This leaves behind novices in AI or programming. To address this gap, Wei proposes a human and ethics-centered AI research paradigm, called HAI [16]. HAI research is based on three main components ”1) ethically aligned design, which creates AI solutions that avoid discrimination, maintain fairness and justice, and do not replace humans; 2) technology that fully reflects human intelligence, which further enhances AI technology to reflect the depth characterized by human intelligence; and 3) human factors design to ensure that AI solutions are explainable, comprehensible, useful, and usable”.
Researchers should inspect valid user scenarios and implement DL solutions that would be *useful* for said scenarios. Furthermore, their should be a shallow learning curve to accessing the functionality of the DL system and a user interface should be implemented based on HCI principles to make it *usable*. The research following this paradigm should emphasize the augmentation or enhancement of human capabilities, rather than their complete automation or replacement. The resulting interactions with these AI solutions are called human-machine teaming. The underlying machine and the human user fulfill their own roles and work towards a goal in collaboration.

2.3 Deep Learning

Given the background on audio visual systems and Human-Centered AI, we are looking for a generative system that is both usable and useful for audio visual performances. This demands, low latency in rendering, the possibility to control the generative process, a low

barrier of entry and accessibility to general users that do not have access to large datasets. We end up choosing StyleGAN2 [2] as our visual backbone, generating visuals frame-by-frame. By reviewing different approaches to generating visuals, we can discern the benefits and drawbacks behind the techniques. This will allow us to explain our choice of the generative framework in later chapters.
As such, this section provides an introduction to Deep Generative models. We will briefly explain Variational Autoencoders [23] and Diffusion models [24] and then outline the development of GAN research culminating in StyleGAN2-ada. We finish off the section with a short review and explanation on which Deep Generative Model fits our use case the best.

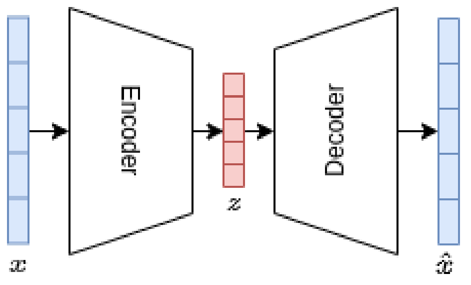
2.3.1 Overview

Deep Generative Models [25] are a family of Artificial Neural Networks (ANNs) [26]. They use multiple hidden layers to learn the underlying high-dimensional probability distribution of a set of samples, called training data.
What differentiates Deep Generative Models from other ANNs is that most ANNs learn a mapping from a high-dimensional intractable distribution to a generally lower-dimensional tractable distribution [1], e.g. classification tasks. On the other hand, Deep Generative Models map from a tractable distribution, also called latent space, to a generally higher-dimensional intractable distribution.
Deep Generative Models have rapidly grown in their performance and are used, for example, to generate music, voices, images or videos. For Autolume-Live, we want to harness the generative powers of this growing technology for live visual performances. In particular, with a focus on users being able to control the model's output, with the primary application of VJing. We originally looked at frameworks for video generation as we wanted to incorporate temporal information, but as of now the quality of the generated samples and the inference speeds are subpar for our goal. Hence, in our analysis of Deep Generative Models, we looked at systems used for image, which can mostly be separated into Variational Autoencoders (VAEs), GANs and Deep Diffusion models.

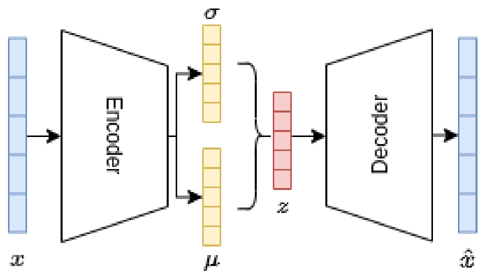
2.3.2 Variational Autoencoder

Variational Autoencoders (VAE) [23] are an extension of Autoencoders [27]. Autoencoders are neural networks that follow an hourglass-shaped architecture. They compress data with an Encoder into a lower-dimensional latent vector and reconstruct the input from said latent vector with a Decoder. When looking at the architecture, it seems like the model would lend itself to generative tasks. The idea being that we can decode randomly sampled latent vectors into the input space. But, during training Autoencoders learn to encode each input

[1]A distribution is **tractable** if any probability induced by it can be computed in linear time



(a) Autoencoder



(b) Variational-Autoencoder

Figure 2.1: A side-by-side of the AE and VAE frameworks. Both map a sample x to a latent vector z. In the case of the VAE the input is not directly encoded to z, but rather to a mean and standard-deviation vector, μ and σ. The decoder maps z to $\hat{x}$, the reconstruction of the input x

sample independently without enforcing any limitations on the compressed distribution. Hence we have no information on the latent space and how we could sample from it. If we randomly sample vectors, we would most likely retrieve a latent vector that the Decoder maps to an invalid output.

To circumvent this problem, VAEs enforce the latent space to follow a predefined distribution. As can be seen in Figure 2.1, instead of compressing the input into a single vector, it maps it to two vectors μ and σ. These two vectors are used to represent a Normal distribution, where the elements of μ represent the mean and the elements of σ the standard deviation for every dimension. To reconstruct the input, we sample a latent vector from the Normal distribution. Then, we follow the same procedure as in the original Autoencoder and map the latent vector to the higher-dimensional space.

With this change to the architecture we enforce the latent space to follow a Normal distribution. But, we are still unable to generate new samples, because each input is still encoded independently returning a unique mean and standard deviation vector for every sample. To enforce the model to encode samples into a global distribution we use the Kullback-Leibler (KL) Divergence. The KL Divergence measures the similarity between two distributions. We want the latent space to follow a distribution we can randomly sample from. Hence, we can use KL Divergence as a regularizer that forces the latent distribution from the batch of encoded samples to follow the unit Gaussian.

While VAEs have a comparably fast inference speed (Table 2.1), the model struggles to generate sharp images. The loss function assumes that the latent space follows a uni-modal Gaussian distribution. Consequently, if the data follows a multi-modal distribution, it incorrectly forces the model to find a singular mean.

2.3.3 Deep Diffusion Models

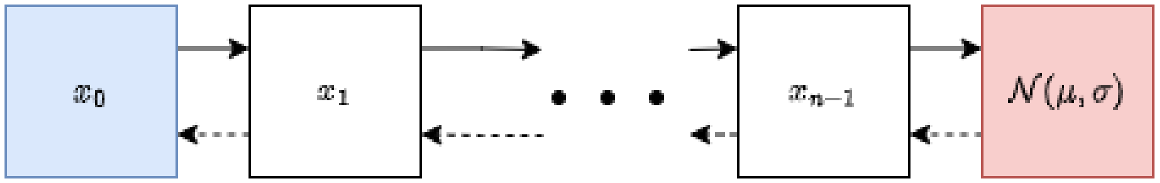


Figure 2.2: The forward and backward pass of a Deep Diffusion model. Where x_0 is the starting image and the upper path applies Gaussian noise in every step ending with complete noise.

Instead of following an hourglass-shape with the latent space being the latent space, Deep Diffusion Models learn to perform transitions of a Markov-Chain that maximize the likelihood of the training data [24]. In its original definition, the model learns to perform Gaussian transitions (forward pass) and their inverse transition (backwards pass), i.e. iteratively applying or removing Gaussian Noise (Figure 2.2). We train the model by minimizing the KL-Divergence between the Gaussian transitions in the forward and backwards pass. To generate a new sample, we give the model noise in the shape of the desired output. Then, the model repeatability performs the learned inverse Gaussian transition.

These types of models are able to generate high fidelity images with a better diversity than GANs, but at the moment they are still lacking in inference speed, as they need to apply multiple backwards passes to synthesize an image. With faster sampling algorithms the speed has been and will continue to improve, but at the time of writing they are not usable for real-time image generation.

2.3.4 Generative Adversarial Networks

The goal of DGMs is to synthesize new samples that could reasonably be considered to come from the training data. The problem consists of both finding an architecture, and to find a loss function that generalizes past reconstruction. Generative Adversarial Networks (GAN) introduce a novel way to measure the distance between the synthesized ”fake” data and the training ”real” data [1]. The model consists of a Generator and a Discriminator, where the Discriminator evaluates the origin of its input, see Figure 2.3.
The framework is called adversarial as in its training, the Generator and Discriminators compete in a mini-max game. The Generator is learning to generate samples close to the training distribution, becoming better at ”tricking” the Discriminator, whereas the Discriminator trains to become more accurate in its evaluations.

Originally, Goodfellow et. al defined the Generator as a network that maps a latent vector drawn from a standard Gaussian to an image and the Discriminator as a network that performs binary classification, labeling it either as real or fake or as a continuous

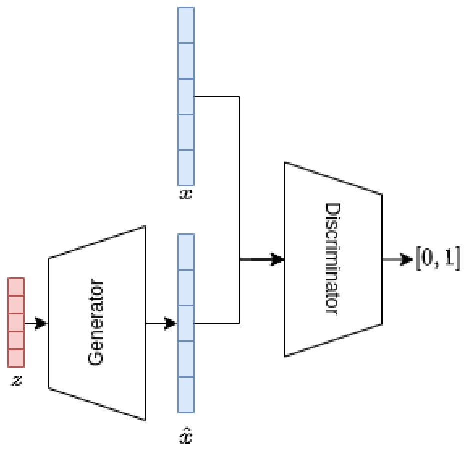


Figure 2.3: The GAN framework as defined by Goodfellow et. al [1]. The latent vector z is mapped to a new sample $\hat{x}$ and the Discriminator classifies both samples x from the training set and the generated samples $\hat{x}$.

score [1]. The two networks are trained in unison using the cross-entropy loss function V (Function 2.1). During training, the Generator minimizes the value function V, and the Discriminator maximizes it. The optimum of this loss function is an equilibrium between the two networks.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(z)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]. \tag{2.1}$$

The problem of finding an equilibrium is unstable, which complicates the training of GANs. If either model is sub-optimal, the loss function might be non-informative for the other. Hence, it is common for GANs to have difficulties converging to realistic samples, or they fall into the problem of mode collapse, where the Generator learns to reproduce a small subset of the training data.
Since its conception, there have been a variety of adaptations to improve the original framework [28, 29, 30, 31]. The training process has become more stable, the overall quality of the generative power has improved, and the amount of data necessary to train a model decreased.

StyleGAN2

StyleGAN2 is a result of iterative improvements to the GAN framework [32, 33]. It uses a changed generator architecture that introduces an intermediate Mapping Network. Furthermore, it changes the training procedure by alternating between different training phases, performing data augmentation and applying regularization.

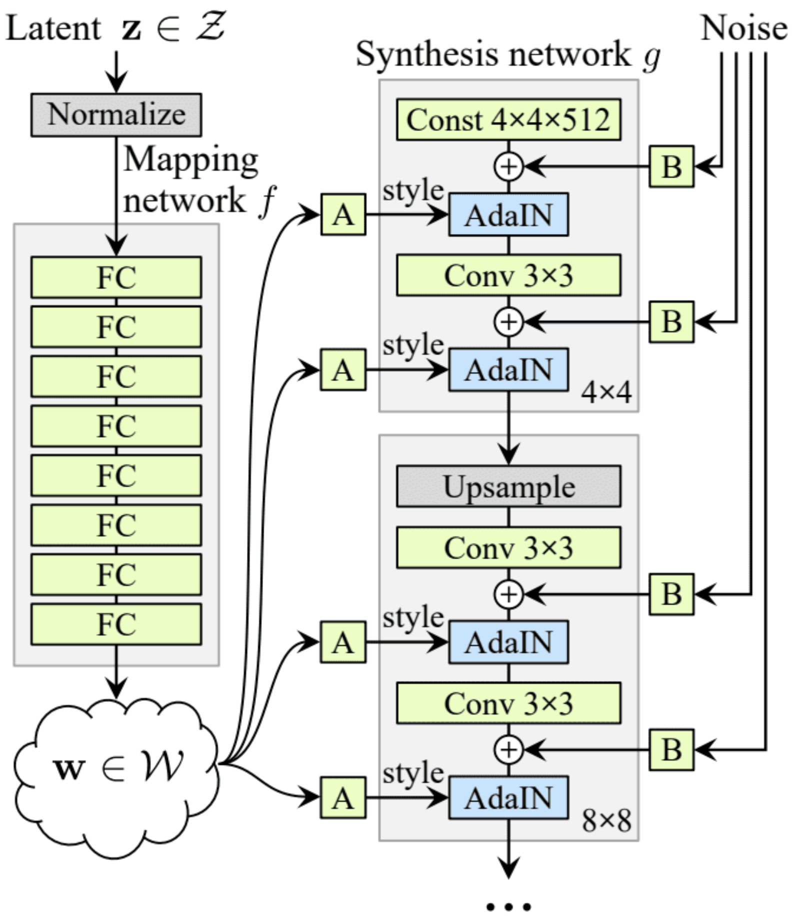


Figure 2.4: Style-based image generation [2]

The *Mapping Network* consists of fully-connected layers and map a randomly sampled vector into an intermediate space W. The resulting vector is called a style vector. While, the original GAN Generator used a vector as its input, style vectors are used to modulate the weights in the Generator at every layer (Figure 2.4).
The idea behind mapping the latent vector into an intermediate style vector is, that the *Mapping Network* disentangles features and creates a better behaving latent space for image sampling. Additionally, by modulating the weights with the style vector instead of using it as the input to the Generator, it is possible to mix together images. The different layers in the network affect different detail levels of the resulting images. Early layers define the coarse features and later layers fine details. Hence, it is possible to use one style vector for the coarse features and one for the finer details.

In addition to the changes made to how latent vectors are sampled and used in the synthesis process, the Generator, in this work also called *Synthesis Network*, includes further changes from the original GAN architecture. As the content of the image is mainly defined by the style vector, it was found that replacing the input, that is normally a random vector, with a learnable constant, simplifies the architecture without losing any image quality [2]. Lastly, noise is added to every layer. This noise introduces stochastic features to images, e.g. freckle or hair positions. This architecture has been iteratively adapted [32, 33, 2] and a newer framework [34] has been released to further improve image quality, but its inference speed is slower then StyleGAN2.

In addition to changes to the architecture, a new training loop was proposed. Instead of updating the weights of the Generator and Discriminator at the same time, they are trained in alternation. Furthermore, every n-steps a regularization step is performed. The regularization includes two process, i.e. L1-regularization and Perceptual Path Length (PPL) regularization. The L1-regularizer computes the norms of the gradients, representing the change between training steps. It punishes large changes to the networks' weights.
The Path Length (PL) regularizer encourages that a step-size of a step in the style space W results in a respectively sized change in the generated output. This is done by applying random noise to a generated image and observing the gradients of the style vectors that are induced by applying the noise. During training these gradients' lengths of the current training step are compared to a decaying mean of previous training steps. The regularizer punishes if the gradient lengths of the current training step is different from the mean.

StyleGAN2-ADA

To reduce the amount of data necessary to train GANs dataset augmentations, Adaptive Discriminator Augmentations (ADA), and further changes to the training loop were made [35]. The augmentations are applied to the Discriminators input, real or fake, based on a set probability. This probability changes during training based on how much the Discriminator is overfitting. To quantify overfitting, two heuristics are used.
First, by using an additional validation set, we can compare the Discriminator's performance on the real, fake and validation data. We can see the Discriminator overfits, if the performance on the validation set is closer to its performance on the fake data, as it should actually be closer to the real data. This heuristic is only useful if enough data is available as the validation set would require an additional set of "real" images.
The second heuristic, and also the one they end up using, is the computed average over the sign of the Discriminator output on the real data. This estimates the portion of the training set that gets positive Discriminator outputs, i.e. correct assignments. It is likely that the

[2]When discussing image quality we define it as a combination of subjective review, discerning whether the generated images realistically could come from the original dataset, and objective qualities, e.g. the resolution at which the model can produce realistic samples.

Discriminator is overfitting if the entire training set is correctly assessed, especially when augmentations are performed on them.
When applying augmentations to generative model training, we need to watch out that the augmentations do not leak into the generated samples. It was found though, that applying the augmentations randomly given a likelihood of them occurring, the augmentations will not leak through if the likelihood stays below 0.8.

Differentiable Augmentations

A similar approach is applying Differentiable Augmentations, translation, rotation and random cutoffs to the images during training [36]. While the augmentations in StyleGAN2-ADA were performed to preprocess the data to train only the Discriminator, these augmentations are differentiable and the gradients can be back propagated to the Generator and the Discriminator. With these augmentations, it is possible to train StyleGAN2 with a dataset with 100 images.

Projected GANs Converge Faster

In addition to further data augmentation, further changes to the discriminator are proposed to speed up training even more. By projecting the real and generated images into a latent space using a pretrained classification model, and using the discriminator on the latent space level, they speed up training up to 40 times [37].

Top-k Training of GANs

An additional improvement to GAN training, reducing mode collapse and improving image quality, is called top-k updating [38]. When training the GAN, normally all samples are considered when updating the model. Following this technique, we discard samples with the lowest score given by the Discriminator, updating the Generators parameters only on the best samples.

2.3.5 Video Generation

In addition to models that can generate static images, we reviewed video generative models, which are extend on the previously described GANs, VAEs and Diffusion models by introducing temporal information into the generation process. We generally can separate these models into either sequence continuation models, that generate the next frame given a history of frames, or full sequence models, that generate an entire clip at once. Over time the line between the both is becoming blurry as most full sequence models allow to be conditioned by a prior allowing the user to perform sequence continuation with them.

We generally found, that the state of video generation models are not at a state that they are usable for our task. Most models are only able to generate short few second clips at a resolution of 256x256 or lower [39, 40, 41, 42]. Furthermore, due to the higher dimensionality of the task, temporal and spatial, the models require more computing power and longer training times. Additionally, these models generally could not be trained on diverse datasets, but rather needed a large collection of video content with a homogeneous subject.

Although as of writing, video generation models are not yet usable for end-users, we are seeing developments that point to it being possible in the near future. Models such as StyleGAN-V, TATs and LongVideoGAN show that it is possible to generate longer form content [43, 44, 45]. Furthermore, StyleGAN-V and the LongVideoGAN are built on the StyleGAN2 and StyleGAN3 architecture and generate one image at a time with little overhead for temporal dependencies, this could allow for close to real time image generation with further improvements in hardware or model optimizations. Lastly, most video generation models lack control mechanisms, but with models like GODIVA, GameGAN and Make-A-Video, we see a push towards adding further modalities to video generation [46, 47, 48]. These systems allow users to generate short clips through text description or play complete video games with controller inputs.

2.3.6 Review

	Image Quality	Inference Speed	Ease of Training	Dataset Size	User-Base
VAE	×	✓	✓	∼	×
DDM	✓	×	✓	×	✓
GAN	✓	✓	∼	∼	✓

Table 2.1: Evaluation of Deep Generative Models

Knowing about these different approaches we reviewed how well they fit our use cases. We evaluate them based on their image quality, inference speed, ease of training, dataset size and user-base(Table 2.1).
For our system it is important that the generative model adds little latency to our system, and can capture the image quality found in the dataset. Furthermore, the model should me trainable by non-experts and the required dataset should be in the magnitude of hundreds of images. Lastly, we also look at whether the framework is used frequently and has additional support available. This is important to us, as we want the barrier of entry to be low. By having an already existing larger user-base, there is generally more support and resources for the technique [49]. Furthermore, it is more likely that users can transfer knowledge and tools from their previous work if they have already seen work or worked themselves with

the framework before.

Looking at the models, we decide to go with GANs. GANs can achieve high quality image generation, which VAEs can struggle with. Although, VAEs technically need even less computing power, GANs are comparatively fast. DDMs in particular are currently not usable for real-time image generation due to their inference speed. Hence for our usecase VAEs and GANS would be the only option. When it comes to training, GANs are usually more difficult to train than VAEs, as described in Section 2.3.4. But with newer research, training GANs has become more stable. Both VAEs and GANs originally needed large datasets, but progress was made for both models to reduce dataset size. Lastly, GANs have been used more frequently than VAEs in installations, and there are collections of pretrained models for GANs that makes it easier for new users to make use of their generative power. Specifically for StyleGAN2, there are online resources to train models [50] and a plethora of online classes that teach beginners how to train them on your own dataset and create art with them [49].

2.4 Model Distillation and Compression

While Machine-Learning solutions are becoming more accurate, their inference speed can be lacking. One reason is that the number of features and layers have been increased to reach better performance. Depending on the layer type, this can have an exponential effect on the runtime. One way to improve the inference speed of a model for deployment is model compression.
Although, StyleGAN2 can run at 30FPS at 1024x1024 and 45FPS at 512x512, which would suffice for real-time rendering, we explore theses techniques to allow our system to run on older hardware and create even smoother rendering.

There are a variety of approaches to compress models. Most of them boil down to three strategies, pruning, knowledge distillation and quantization.
To reduce the number of parameters, we can use **Pruning**, which can be performed on different levels, e.g. layers, features, or kernels [51, 52]. The idea is to take out parts of the network weighing the improved speed with the loss in accuracy.
The process of using original model as a teacher model for a smaller network is called **Distillation** [53]. It can be linked with pruning, where we would first prune the teacher network and then train the pruned model using the original network as its teacher [54].
Quantization does not effect the number of parameters in the network, but rather effects inference speed, by quantizing the weights, e.g. by changing the representation to a lower bit representation (32-bit float to 16-bit float).
All these ideas can be combined as needed, and the trade off on the models performance

assessed on a case by case basis. But, most frameworks that were proposed for classification tasks do not apply to GANs, resulting in sub-optimal compression. To address this problem GAN specific variations of pruning and distillation were proposed [55]. By introducing content-aware pruning and distillation Y. Liu et al. have shown that it is possible to compress StyleGAN2(-ada) and reduce the complexity of the network counted in the number of floating point operations (FLOPS) by a factor of 11. The idea of content-aware compression is that a content-parsing network is trained to find the features in the model which have the biggest impact on the visuals. During the pruning and distillation process this model masks the generated image of both the pruned and teacher model. Then the distillation loss is applied on the masked image generated by the teacher and the pruned model. This enforces that the salient objects in an image are distilled into the pruned model.

Chapter 3

Related Work

3.1 Controlling GANs

As discussed in Section 2.2, with AI systems becoming increasingly more applicable for every day tasks, the demographic of end users of these systems is shifting. While explainable AI has been focusing on expert to expert explainability [56], there have been developments in creating user interfaces that allow novices that have no experience in either AI or coding. Although there is a rapid influx in user interfaces for text-to-image systems such as Stable-Diffusion [11], those systems can not be used for live performances as their inference speed is too low [13]. Furthermore, their controls are mostly not applicable to StyleGAN2 due to the different nature of the different algorithms. Hence, we will focus on two interfaces that were implemented for either StyleGAN2 and the newer StyleGAN3, which have influenced the development process of Autolume-Live.
In addition to interfaces, researchers have also explored manipulations in the latent space or on the model to add further controls to the generative process. These techniques range from performing affine transformations on the activations [3], finding meaningful directions in the latent space [57, 58, 8]. We will only address works that do not require an additional model or other overhead, to reduce the amount of additional latency that is introduced.

3.1.1 Interfaces

First, Live-GAN is a project that implemented an adhoc mapping between StyleGAN2 features and a MIDI-controller [59]. This system gives user control over the generation process through:

- **Random latent sampling**: Sample a random point in the latent space, resulting in random images.
- **Per-scale truncation**: Change the truncation value per layer or globally, which restricts the generation space, if set to 0 generates the average image from the train dataset

- **Global truncation**: This value is normally set between 0.8 and 1, where increasing the value increases the variety of images, but also allows for images that are less like those the model was trained on.
- **Global noise scaling**: Scale the noise used in StyleGAN2, which can increase or decrease the effect it has on the final image. When set to 0 the resulting image generally loses most of its texture.
- **Latent direction exploration**: Move into a specific direction in the latent space.
- **Latent interpolation animation**: Interpolate between randomly sampled key frames.
- **Random latent jittering**: Perform a step into a random direction in the latent space

Its implementation is not able to in real time for an extended period of time, as the latency accumulated when changing parameters in real time. It had the benefit that it allows users to interact with ML with a tactile controller that is already commonly known to live performers.

Second, NVIDIA has released a visualizer tool for its StyleGAN3 framework. It allows users to explore features of the model by changing numbers and moving sliders to adjust parameters. In its implementation, it uses a Python binding of **imgui** [60], and handles both image generation and rendering on the GPU with little transfer between CPU and GPU. This allows for efficient rendering. The interface has multiple drawbacks, first it does not run live when using the new StyleGAN3 frameworks at resolutions bigger than 256px on a commercial gaming GPU.
Furthermore, the interface is meant as an exploration tool of StyleGAN3 for other experts. Users are unable to interact with the model with controls other than mouse or keyboard. This makes it not feasible to use for performances, especially when wanting to map audio features to visual features.

3.1.2 Meaningful Model Manipulations

In this section, we will showcase two techniques for interpretable editing of GAN outpu. First finding meaningful directions in the latent space of a GAN. Second, Network Bending, applies affine transformations on the activations of the model.

Interpretable Latent Space

While StyleGAN2 already disentangles the latent space, moving into random directions does not result in any interpretable changes to the output. One way of finding directional vectors in the latent space that have interpretable meaning is GANSpace. GANSpace identifies interpretable directions in the latent space using PCA. This method is lightweight and

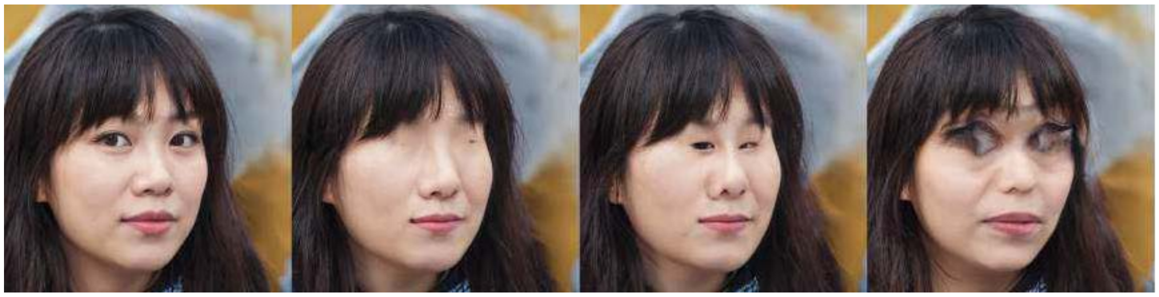


Figure 3.1: Modulating eye size using feature specific Network Bending [3]

does not need any supervision [58]. It finds the basis of the latent space in which the generated images have a high variance. To do so we sample N amount of latent vectors and project them into the style space. Then we perform PCA, or variations of it, to extract the Eigenvectors of the style space.

Network Bending

We call applying affine transformations on the activations of a model *Network Bending* [3]. It allows us to manipulate the synthesized images in ways not possible by latent space traversal. E.g. by applying image transformations (rotation, translation scaling) on entire layers these transformations propagate to the resulting image.
Furthermore, by targeting a subset of features per layer it is possible to perform feature specific changes. This can be seen in Figure 3.1, where a secondary algorithm finds clusters in the networks activation to find salient features. In that example features that represent eyes in a portrait were resized, which allows us to scale the eye size. In addition to the targeted Network Bending, it is also possible to manually define which feature maps in a layer the network bending should be performed on.

3.2 Responsive Visual Generation

Our goal is to implement a tool for live audio-visual performances using a DGM, with a Graphical User Interface that does not require any coding experience. Although this has not been down before, there are a variety of offline systems that map audio signals to parameters in DGMs to create audio-reactive videos. In this section, we will give an overview to some of these approaches, that inspired our development of Autolume-Live.

First we are going to distinguish between two different techniques of audio reactive content generation. One approach is to create collages, stitching together videos from a collected corpus based on musical features [61]. The other is, to generate every frame from scratch. We want to harness the complete generative power of DGMs and move away from

Figure 3.2: Example generated with Deep Music Visualizer [4].
`https://www.youtube.com/watch?v=L7R-yBZ5QYc`

the common practice of VJs of repurposing video clips during their performances and create a completely new paradigm, closer to music visualizers. Hence, our Autolume-Live is based on the latter approach. For the latter approach we have seen three different ways to generate audio-reactive videos using GANs: Latent Space Traversal, Latent Space Interpolation, Chroma-Based Interpolation.

As discussed in Section 2.3 GANs use vectors sampled from a latent space to generate images. Vectors that are close together in said space are mapped to visually similar images. By moving around in the latent space it is possible to generate smooth videos interpolating between different key-frames. Researchers and artists have used this quality of GANs to create videos showcasing images morphing from one into another. The three mentioned offline audio-reactive systems use the topology of the latent space in different ways.

One way of mapping audio to video through movement in the latent space is based on a random walk. We call this approach Latent Space Traversal (Figure 3.2) and a version of it was used in Deep Music Visualizer (DMV) [4]. In this case, the amplitude of the spectrogram and the shift in amplitudes between time steps are used to vary the step size of the random walk. This approach of audio-reactive image generation has the benefit that it can respond to music without any harmonic, percussive separation, but it is unable to capture repetition which is a common quality of music. As the model moves into random directions at every time-step it is unable to return previous points based on the audio. Furthermore, this approach lacks a variety of parameters, since only the audio features and a multiplier to the step-size are used.

Figure 3.3: Example generated with Lucid Sonic Dreams [5]. `https://www.youtube.com/watch?v=l-nGC-ve7s`

Another way of navigating the latent space is Latent Space Interpolation, i.e. interpolating between preset positions in the latent space. While the previous Latent Space Traversal was unable to capture repetition, Latent Space Interpolations allow creating loops. Lucid Sonic Dream (LSD) ([5] is a system that uses this approach, similar to DMV the amplitude and shift in amplitudes are mapped to the speed of the interpolation. While looping is one of the strengths of Latent Space Interpolations this does create a repetitive experience. To circumvent this problem, LSD uses a combination of both Latent Space Interpolations and random walks. The harmonic and percussive tracks are separated. While the harmonic track is used to modulate the speed of the loops, the percussive track is used to introduce a momentary “pulse”. This “pulse” is created by performing a random step as seen in the Deep Music Visualizer for one time step and then returning to interpolating to the next latent vector. An example of a video generated by LSD is shown in Figure 3.3.

Lastly, there is Chroma-Based Interpolation, showcased in Figure 3.4. Instead of using the amplitude of the audio signal, the pitches are calculated through a chromagram and mapped to the visuals. A chromagram is computed by clustering a spectrogram into frequency bins for every pitch [62, 63]. In western music theory, normally 12 bins are used. In LSD and DMV, conditional GANs can be used, where a class vector is appended to the input to define the content of the synthesized image [64, 65]. It is then possible to use a chromagram to influence the content of the video, i.e. every pitch is linked to a different content class. A different approach to incorporate pitch is to map every pitch to a key-frame [6]. The resulting video consists of interpolations between key-frames based on the pitches present in the music. For example, if the note C is mapped to a blue frame and the note D to a red frame. At every point in the music where the note C is played on its own the video would show the blue frame, but if multiple notes are played the two key-frames are interpolated,

Figure 3.4: Example Chroma-Based Interpolation as implemented by Brouwer [6]. `https://wavefunk.xyz/assets/audio-reactive-stylegan/rhodes.mp4`

i.e. if we play C and D together a purple frame will be shown. The mixing ratio between key-frames is the amplitude of the responding pitches. While this approach implicitly captures repetition through the pitch, it is not able to represent atonal or monotonous musical. Without any pitch changes, the mapping will create a static video. In addition to using short-term features, amplitude and pitch, Brouwer incorporates long-term musical features to visualize long-term musical structures in the music. By applying Laplacian-Segmentation [66] on the audio track, the song is clustered into sections and different sets of latent vectors are used for the distinct parts in the music.

Outside of making the visuals audio-responsive based on movements in the latent space, Brouwer also proposed using Network Bending from Section 3.1 to add further changes to the visuals based on the input audio signal.

In addition to the aforementioned GAN based audio-reactive video systems, Deep Diffusion Models were also used for the task. As discussed in Section 2.3, DDMs do not have the adequate inference speed to be used in a real-time setting, hence we will not go into detail for those approaches. Nevertheless, we want to shine a light on some techniques for the case that improvements on the models' generation speed are made allowing them to be used for real-time applications. They are generally combined with text-to-image systems, either using the lyrics as the prompts, or using audio features to drive transformations on the diffusion's input, e.g. amplitude to zoom speed, or translation to a specific frequency band [67].

Chapter 4

Autolume-Live

Our work has focused on creating a tool that incorporates and adapts current technologies in generative modeling, inference speed ups and interpretable AI. Our system has gone through multiple iterations from a minimal program that was implemented using Processing, mapping audio amplitude to the speed of a random walk through the latent space. We then iterated on incorporating a hardware controller common for VJing. And now the newest version Autolume-Live that extended the interface to no longer be adhoc bound to a specific controller. We went back to a purely digital system, but we allow users to map all the parameters with OSC-addresses [17]. This allows the user to create their own mappings, either with other software, or with their preferred hardware controller.

Autolume-Live was conceived from the idea of accompanying live music performances with an AI-based visual performance. The system should allow for responsive visual generation with further adjustments being possible with control signals defined by users. Our final version of Autolume-Live has the following features:

- **Presets**: Allowing users to save and load mappings to prepare and structure performances
- **Multiple Latent Representation**: We allow users to interact with the latent space either through a seed or directly with a latent vector.
- **Looping**: The user can define and perform loops between a set of keyframes.
- **Animation Speed**: Users can modulate the speed of the animation.
- **Network Specific Features**: Users can modulate parameters that are part of the StyleGAN2 architecture, e.g. the injection noise and global truncation
- **Network Bending**: Perform affine transformations on any layer.
- **Output Scaling**: The width and height of every layer can be adjusted, opening up the capabilities of rendering at higher or lower resolutions and different aspect ratios.

- **Audio Responsive Mappings**: Users can define audio mappings within *Autolume-Live*
- **Communications**: Users can send signals to the parameters from other software using the OSC protocol. Additionally, *Autolume-Live* sends out a stream of the rendered visuals using NDI.
- **Salient Feature Extraction**: Using GANSpace extract salient features through an interface.
- **Model Training**: Train a model using an interface, with data-augmentation [35, 36] and transfer learning.
- **Model Compression**: Perform model compression [55] through an interface, with additional data-augmentations.

Although originally thought out for audio-visual performances, we implemented Autolume-Live in a manner where we opened up parameters of the generative systems, extended them with more manipulations and allow users to adhust them within Autolume-Live and also through other software. This means that the system can be used in other contexts depending on the artists vision.

In this section, we will give an overview of the version we published in early 2022, which we call *Autolume-Live 1.0* [7]. Then we establish all the features of the newest version. Furthermore, we are going to discuss the technical implementation, including challenges we faced. Lastly, we showcase the capabilities of our system both with artwork that we have created throughout its development process, which also inspired further changes. And we are going to showcase current possible use cases with common VJing software, Touch Designer [20] and Pure Data [68].

4.1 Autolume-Live 1.0

Our original version of *Autolume-Live* is a live visual engine that can be controlled by discrete triggers and continuous controls. In particular, it allows users to control visuals both through audio and a physical controller. We based our development of the system on [15] culminating in the architecture laid out in Figure 4.1. An Audio Module extracts onset strength, amplitude and pitch and maps these either to the latent space W or the noise term that is added to every layer. The Controller Module processes the interactions with the MIDI-Controller, applying affine transformations to the Synthesis Modules layers or manipulating the latent space.

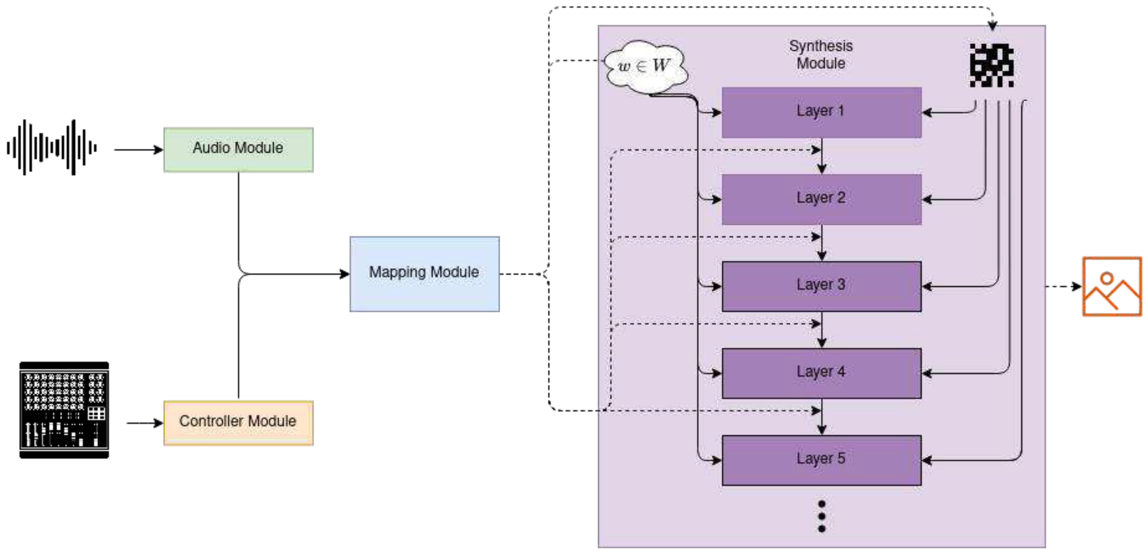


Figure 4.1: *Autolume-Live 1.0* framework showing how we mapped audio and controller signals into manipulations for the Synthesis Module [7].

4.1.1 Visual Module

We decided to use StyleGAN2-ada as the generative module to render he visuals. We were interested in a model that is already widely used in artistic works, allows users to train it without a lot of coding experience and can run with little latency, while still achieving high-quality renders and being controllable.
StyleGAN2 has been used for a variety of artworks, and there are collections of pretrained models. Furthermore, there is RunwayML [50] that allows users to train there own model, simply by uploading their collection of images. Lastly we have seen projects implementing systems that allows users to edit images and frameworks to compress models to reduce latency that allow StyleGAN2 to run at up to 60FPS at 512x512. Hence we chose StyleGAN2 [2] as the rendering engine. (ADA interchangeable generator). For our installations we used the newer ADA extension which does not change the generator, hence it is interchangeable, but reduces the training speed and data necessary [2].

4.1.2 Audio Module

We expand the offline techniques described in Section 3.2 to run live. To do so, we use a monophonic audio stream with a chunk size of 1024, a sampling rate of 44100 and compute the mel-spectrogram, chromagram [63], onset detection and strength locally. For both the spectrogram and the chromagram, we use a FFT window size of 2048. We compute the mel-spectrogram instead of a normal spectrogram, as it better represents the perceived amplitude of the different frequencies by human ears. For efficiency we detect onsets and compute their strengths by approximating a threshold to track onsets with a moving Root

Mean Square over the last second.
We originally computed the harmonic percussive decomposition [69], but due to the small active window size and efficiency of the algorithm the quality of the decomposition did not justify the drop in framerate. As *Autolume-Live 1.0* takes a live signal as its input, the analysis is noisier than the previous offline approaches, which leads to flickering in the images. To reduce the noisiness and make the visuals respond smoother we compute the moving average over the last 3 steps of the audio signal before extracting audio features.

Mappings

(a) Example of Chroma-Based Interpolation given a digital piano input to our system. (`https://vimeo.com/670850243`).

(b) Example Latent Space Traversal using *Autolume-Live 1.0* (`https://vimeo.com/670850199`).

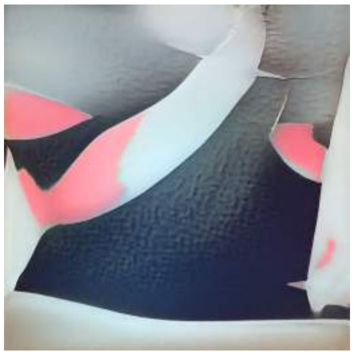

(c) Example of a Latent Space Interpolation using *Autolume-Live 1.0* (`https://vimeo.com/670858023`).

Figure 4.2: Audio-reactive mappings available in *Autolume-Live 1.0*

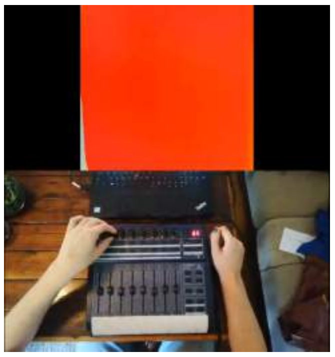
Autolume-Live 1.0 is implementing **Chroma-Based Interpolations**, **Latent Space Traversal** and **Latent Space Interpolation** (Figure 4.2), which can be switched between on the fly. But, the system is written to be modular, where future iterations could allow the user to reconfigure the mappings to their liking. Additionally, we use the strength of the onset, i.e. the amplitude of the spectrogram when an onset was detected as a multiplier to the standard deviation of the noise that is injected in StyleGAN2(-ada). This creates a perceivable change in the image.

For both the chroma-agnostic **Latent Space Traversal** and **Interpolation** approaches from Section 3.2, we use the same audio mapping, only adjusting the directional vector to either be random or pointing to a specific seed. We normalize the directional vector, so that the length before applying audio mappings is consistent. The vector is then multiplied by both the average amplitude of the spectrogram at the current time step and the difference between the current average amplitude and the previous. This leads to bigger steps when the amplitude is high or the amplitude shifts drastically, hence the biggest step would occur when a high amplitude, loud sound, follows a low amplitude, e.g. silence.

4.1.3 Controller Module

Autolume-Live 1.0 is meant as an expressive interactive tool to accompany live audio performances. Jonathan Hook et al. explore the needs of VJs through an in depth qualitative study [15]. They show the need for a physical interface that can be used for performances, i.e. an interactive system that is visible to the audience and a tool that gives haptic feedback. This is why we chose to integrate a MIDI-Controller, which is a common tool for VJing, into our system. Through its buttons the controller allows the user to toggle binary interactions, while its faders are a useful tool for continuous manipulations. For our working system we use the Behringer BCF2000 controller which has eight faders, 16 buttons and 8 knobs. An additional benefit of using a physical controller in comparison to a digital interface is the possibility of parallel interaction. Which allows the user to adjust multiple parameters at the same time. Where a mouse would limit the intractability, with the controller the user can use their hands and fingers.

(a) Moving through the latent space using the faders of the MIDI-Controller. Every fader's values range between -1 and 1 weighting the first 8 directions found by GANSpace [8] (`https://vimeo.com/670850116`).


(b) Using the 8 knobs at the top of the MIDI-controller it is possible to apply Network Bending on the visuals generated by *Autolume-Live 1.0* (`https://vimeo.com/670849859`) .

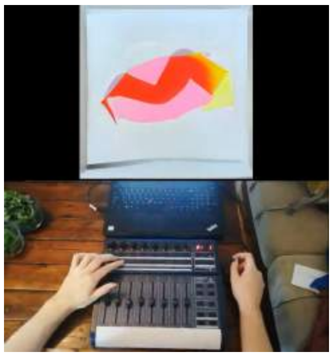

(c) Accessing keyframes using the Midi-Controller (`https://vimeo.com/670850154`).

Figure 4.3: Controller mappings available in *Autolume-Live 1.0*

We decided to link the manipulations described in Section 3.1 to said MIDI-Controller (Figure 4.1.3). The user can edit the current image according to salient directions, using the eight directional vectors we extracted with GANSpace [8] to every fader. Moving them up or down will set the coefficient for the weighted sum off all directional vectors that are added to the current latent vector.

The knobs toggle further manipulations, i.e. Network Bending [3] (rotation, translation, scaling) and truncation value changes. By pressing the knobs the transformation is applied and the intensity can then be adjusted by turning the knob. In this version of *Autolume-Live 1.0* the user is not able to set the features/layers the Network Bending should be performed on, but rather is predefined to effect an entire early layer in the *Synthesis Network*.
We also linked the different audio-visual mappings to the controller. It is possible to switch between mappings pressing on one of the knobs. For both Latent Space Interpolation and Latent Space Traversal the same knob allows the user to add an additional multiplier to the step size, i.e. allows them to manipulate the sensitivity of the visuals reactions to the audio.

It is normal for VJs to pre-record and save visuals that they want to use one or more times in a performance. Although we do not allow users to pre-record and save video clips, we allow the user to preset certain parameters of *Autolume-Live 1.0*. First of all, it is possible to define a set of latent vectors that are used by the Latent Space Interpolation and can be accessed during performances.
As *Autolume-Live 1.0* uses a StyleGAN2(-ada) Generator, the visuals that are generated are dependent on the data the model was trained on. Mostly these Generators synthesize images with only a few varying contents. Hence, to increase the visual variety, a set of models can be given to the software, all with their own predefined set of key-frames.
To access the different presets we use the array of buttons on the controller. On startup, every button is linked to a latent vector that can be predefined by the user or overwritten during the performance. These latent vectors are the positions the Chroma-Based Interpolation passes through. By pressing a button a position is loaded and the visualization process proceeds from that point. Furthermore, when multiple buttons are pressed at a time the average of all these seeds is used to create the visuals.
Lastly, it is possible to iterate through the list of models by clicking the space bar of the computer that is running *Autolume-Live 1.0*. This allows users to switch between the visual content of the performance with a single button press.

4.2 Artworks

Due to the 2020-2022 situation surrounding COVID-19, we were unable to use our system to accompany live performances. We have used different iterations of Autolume-Live to create two installations. We recorded some curated sessions and displayed them at the Distopya sound art festival in Istanbul 2021 [70] and Light-Up Kelowna 2022 [71].
In both installations, we let the audio mapping automatically generate the video without using any of the additional image manipulations. These installations show that the system

is able to generate interesting and responsive visuals for a musical piece.

4.2.1 Autolume-Mzton

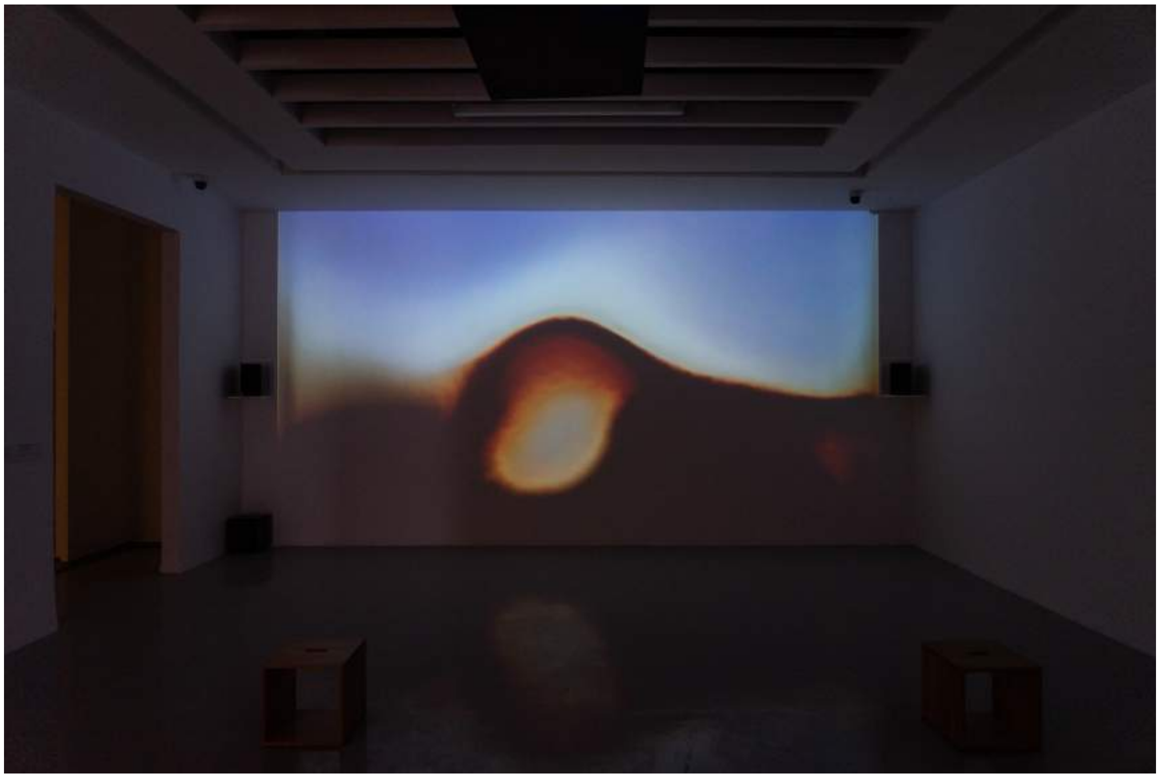


Figure 4.4: Installation showcased at the Dystopia Festival 2021. `https://vimeo.com/527564204`

We displayed 'Autolume Mzton' at the Distopya sound-art festival [70]. It is using an audio track from French analog improvisation collective Robonom called 'Mzton' generated in 2003 and *Autolume-Live 1.0* for the video generation. Robonom uses a collection of analog modular synthesizers. Band members interconnect these modules in a rhizome-like network. Rich electronic textures emerge from these complex interactions through improvisation sessions that are then edited and mastered into final works. For 'Autolume Mzton', we use a re-mastered version of the track and deployed both a 5.1 and a stereo version of the piece (Figure 4.4).
Due to a limitation in time available before the installation we explored pre-existing image collections that fall under the Creative-Commons license or are in the public domain. Due to the experimental and abstract nature of the music we ended up deciding to render abstract imagery using our system. For that, we used a subset from the WikiArt dataset [72] that consists of abstract paintings that are in the public domain. After choosing the dataset we trained a StyleGAN2-ada model on it, saving checkpoints during the training process, stop-

ping training once we are satisfied with the visual qualities of the samples from the model. With the trained model we start an iterative process rendering visuals using *Autolume-Live 1.0*. The real-time nature of the system allows us to change parameters live, and co-creative interactions akin to audio-synthesizers, where the user iteratively adjust the parameters and builds up to the final version. With an offline system, the user would have to render a video with the changed parameters, review the video and adjust the parameters again. But, the drawback of using our system for this scenario, compared to an offline system, is the restrictions set on the resolution. For this particular piece, since we wanted a 4k video, we used *Autolume-Live* to render the video and then ran the video2x super-resolution model on it as a post-processing step [73]. The final mapping we chose was the Latent Interpolation, sampling a new keyframe when finishing an interpolation, hence having no inherit loops.

4.2.2 Autolume-Acedia

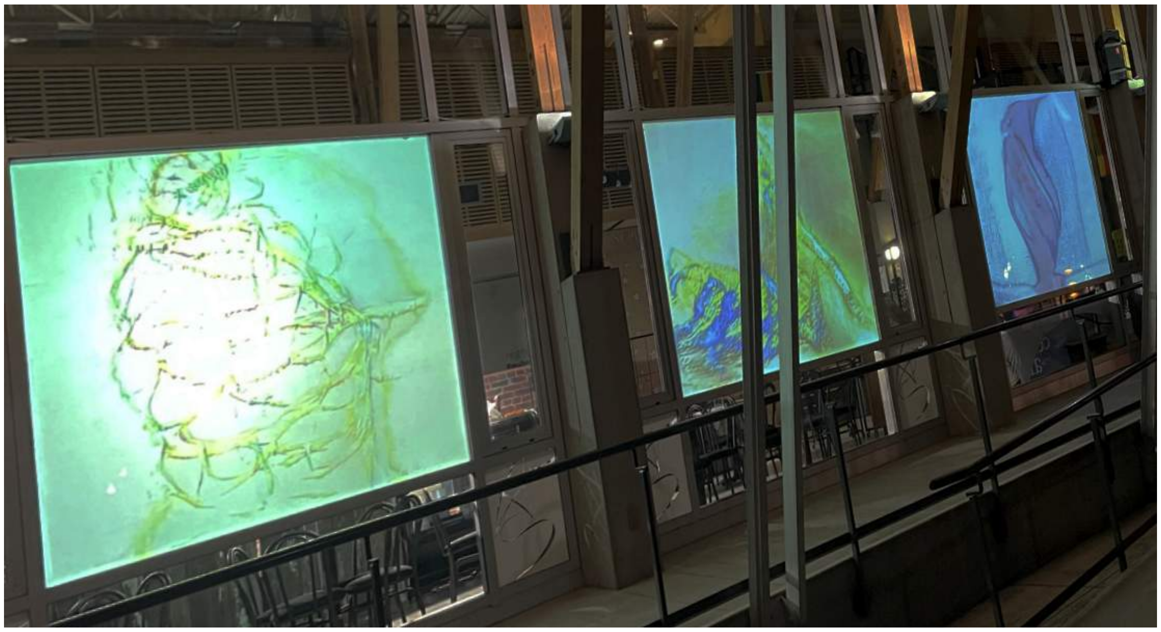


Figure 4.5: Installation displayed during Light Up Kelowna 2022. `https://vimeo.com/696819050/4448ef1fc0?embedded=false&source=video_title&owner=16495219`

Another installation we created using *Autolume-Live 1.0* was shown at Light-Up Kelowna in February of 2022 [71]. The installation was made up of three urban screens, which we incorporated into a video triptych. The audio is a 2020 production by Monobor (a.k.a Philippe Pasquier), and one of few manually composed pieces by the Canadian musician. The piece revolves around a Jazzy bassline and an ethereal chord progression on a background of winter soundscapes (Figure 4.5).

For our piece at Light-Up Kelowna we ran *Autolume-Live* with the Latent Space Interpo-

lation mapping. In this case we chose multiple sets of keyframes that we switched between sections. We did so as we wanted to introduce repetition into the visual performance, but using one set of keyframes made the visuals too repetitive.
The display included three urban screens, which allowed us to showcase three renders at the same time. After experimenting with a variety of different datasets (Figure 4.7) we ended up choosing a dataset of figure drawings, and a dataset of medical sketches.

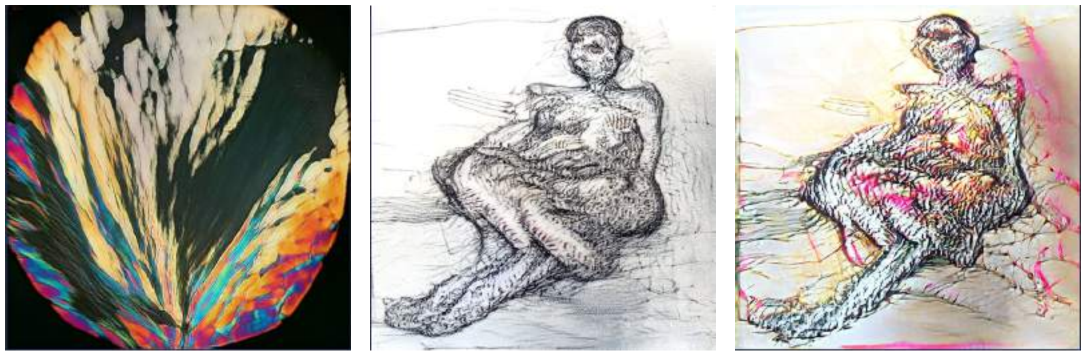


Figure 4.6: By mixing the models generating the first two images, we get a model that generates images like the one on the right, with the content of the middle image, and the colors of the image on the left.

As the installation was in the form of a triptych, we wanted to tie the visuals together. We did so in two ways. First, we created a third dataset that contained both the figure drawings and medical sketches, training a model that was able to generate both types of images and morph between them.
Second, we made all three models generate images with a similar color profile. To do so, we combined networks by swapping layers. By replacing the later layers of the models with layers from a model trained on colorful microscopic scans, we transfer the textures and colors from the second model onto the other models while keeping the content the same (Figure 4.6). This allowed us to add colors to the figure drawing and in regards to the triptych create visual consistency between the videos.

4.2.3 Reflections

Working on these two pieces we have been able to showcase and reflect on the usefulness and usability of *Autolume-Live 1.0*. We perceived that the immediacy of the rendering allowed us to more quickly iterate through mappings and created a sketching like experience when developing the installations.

In this version of *Autolume Live*, we only allowed the users to scale, rotate and translate the activations at the third layer in the network. Furthermore, all the mappings with the

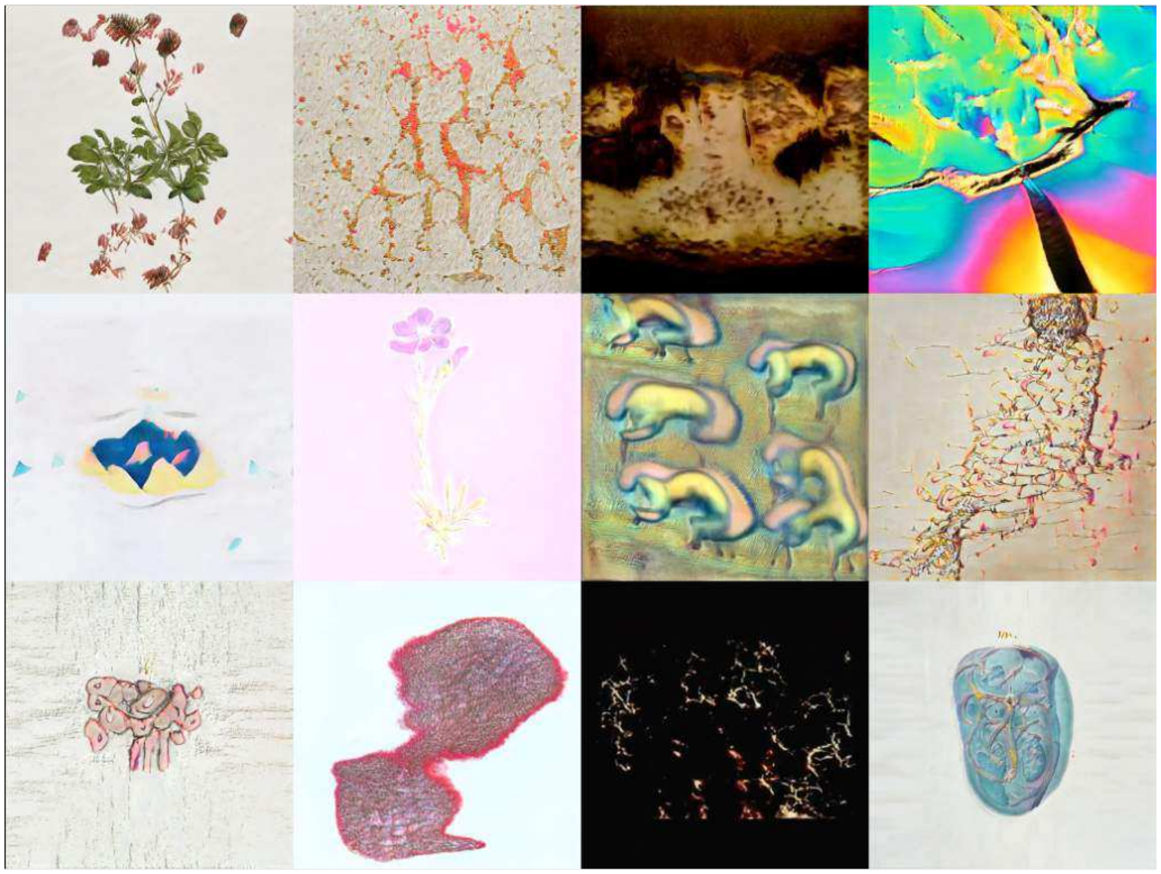


Figure 4.7: Screenshots from a selection of models we have trained for the Autolume-Acedia installation.

controller were hardcoded. This made us feel restricted in what we were capable of creating and depending on the mapping, it made parts of the interface feel useless. In the case of the installations we reconfigured things when needed inside the code to work around those restrictions. For example, adjusting the looping for *Autolume-Acedia*. As we want our tool to be accessible to coding novices, we realize the importance of making the interface reconfigurable without editing the code. This is also something reflected by the VJs in the documentary by Hook et al [15].
In addition to the importance of creating responsive mappings, we acknowledge the impact the visual language, based on the model we are using has. We realize, that although there are services, such as Runway-ML that allow users to train their own StyleGAN2 models [50], it would be beneficial to include both a training and a compression module to allow users to perform these tasks inside of *Autolume-Live*.

Lastly, in *Autolume-Live 1.0* we allow users to adjust the rendered image by pushing the latent vector around in the latent space through moving the sliders. With out further changes, the directions the sliders push the latent vector are randomly sampled, but as already discussed, the intended use is to extract salient features using GANSpace and assigning those to the sliders. This again would require coding experience, and would preferably become part of the interface. We use these reflections to create an updated version that we call *Autolume-Live 2.0*.

4.3 Autolume-Live 2.0

Although the first version of *Autolume-Live*, described in Section 4.1, is already usable for live performances, it is implemented adhoc for a specific controller. The mappings between controller, music and the generative model were predefined and the user would have to change the code to bring out new mappings. As we want users to be able to work with our system without being experts in coding, we redesigned *Autolume-Live* as a digital interface. This interface is more flexible and allows custom mappings both in the standalone tool or in combination with other software that the user feels comfortable with. This allows them to continue to use a physical controller by setting it up with an external software.

Although we discuss our system in the use-case of audio-reactive visual creation, it can be used with other control signals from OSC complient software. As an example, the ”Wekinator” is a tool that can be used for live interactive installations [74]. This section describes all the different widgets we developed for the users to create live visuals with. Afterwards, we describe use cases that show how the new version of *Autolume-Live* can be used on its own or with TouchDesigner and PureData for audio-reactive visuals.

The interface is based on the StyleGAN3 visualizer code [34]. It is based on ”PyImgui” [60] and includes a rendering engine for StyleGAN3 and StyleGAN2 with little transfer between CPU and GPU. We originally implemented our own interface, but switching to their interface code increased our framerate. The original visualizer is a tool to explore the latent space and activations of the model and was rather limited for creative practices, especially since it was unable to create mappings with the interface. Except for parts of the ”Loading widget”, the ”Latent widget” and ”Truncation Widget” all the features are completely original to our interface.

4.3.1 Performance Interface

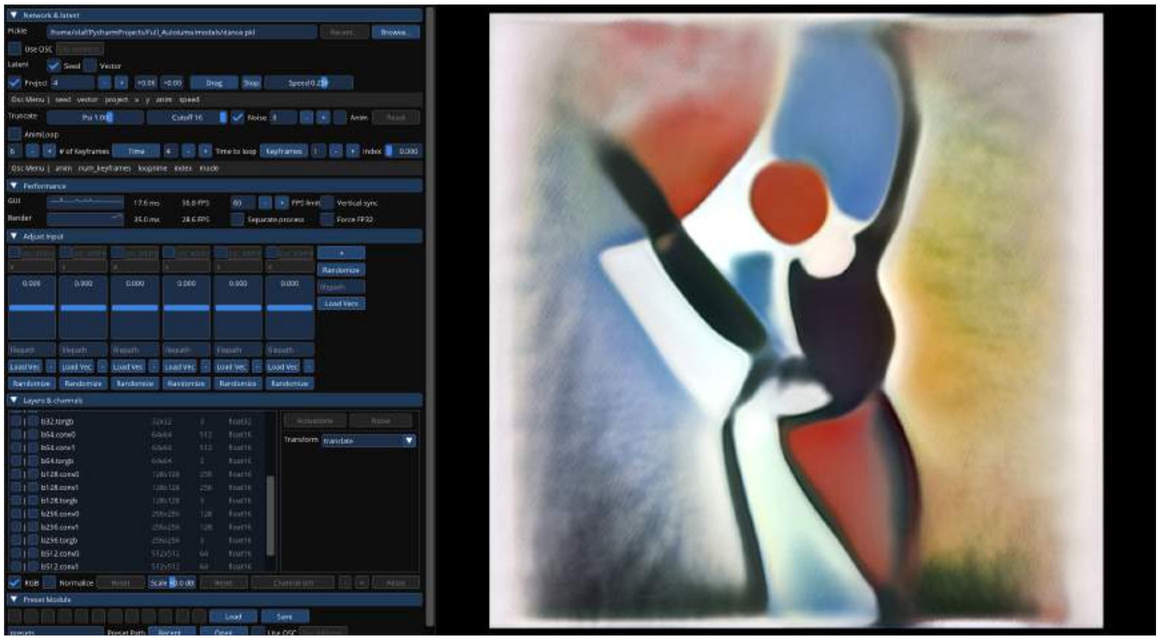


Figure 4.8: The Interface of *Autolume-Live* that the user sees during performances.

Communication

In this section we will dissect *Autolume-Live 2.0* by going through the different widgets of the interface. But, first we will discuss the ways it is possible to communicate with it.
We allow users to send control-signals to the interface through Open Sound Control (OSC) messages [17]. OSC is a protocol designed for networking sound synthesizers, computers and other multimedia devices for performances. It is a common messaging protocol found in live audio or visual software, either locally or through the internet using UDP or Ethernet. Although *Autolume-Live 2.0* is a digital interface and no longer incorporates a physical controller, users can now choose their own means of controlling through the adequate software, including setting up their own physical controller. This makes Autolume-Live accessible to

a broader audience and allows for experimentation with the control-mechanisms.

Most parameters in our system include a way to map an OSC-channel to them, either displayed as a menu bar for a collection of parameters or in the case of layer transformations, as two text prompts. The user can input the OSC-channel for the parameter in the first text prompt.
Furthermore, we want to give users the ability to adjust the control-signal inside our system, to quickly change the magnitude of the control-signal. That is why the user can write a mapping function inside the second text prompt. This mapping function has to be written in Python syntax and users can use NumPy and PyTorch functions in it. This is the only point that users need to understand the math syntax of Python. But this feature is completely optional and users can apply mappings to the control-signal in their preferred environment and send the processed signal to *Autolume-Live*.

The OSC protocol is unable to handle images, hence if the user wanted to postprocessing of the output outside of Autolume-Live, we are using the Network Device Interface protocol. Network Device Interface (NDI) is a royalty-free software specification developed by NewTek to enable video-compatible products to communicate, deliver, and receive high-definition video over a computer network in a high-quality, low-latency manner that is frame accurate and suitable for switching in a live production environment [75]. This allows users to send the resulting images to other visual software, e.g. commonly used streaming software OBS or VJing software such as TouchDesigner or Resolume [20, 19].

Loading Widget

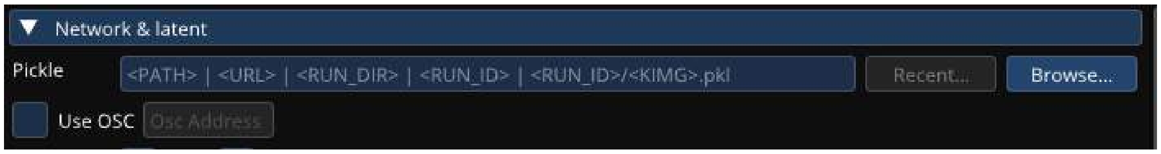


Figure 4.9: Loading-Widget available in *Autolume-Live 2.0*

The user can load models either by inputting the path to the checkpoint or by selecting a model from the browsing list. The browsing list consists of all the models that are found in the ‘models‘ directory found in the *Autolume-Live 2.0* project directory. So the user can simply put all their checkpoints into said folder to more easily access their pretrained models. It is possible to load models by sending in the absolute path to a model through an OSC-message containing a string. This is done by checking the ”Use OSC” checkbox and typing in the OSC-channel that should be used.

Latent Widget

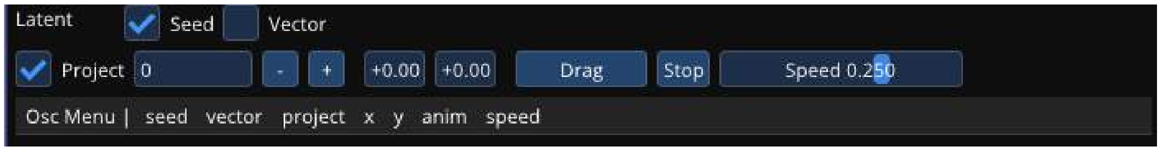


Figure 4.10: Latent-Widget available in *Autolume-Live 2.0*

While the previous version of *Autolume-Live* had preset Latent space traversals, this version allows the user define their own mappings. The Latent widget allows the user to adjust the parameters of the input to the generator, i.e. the latent vector. By clicking the checkbox labeled ”project” we allow the user to choose whether the input vector should be projected to the intermediate W [2] space. Furthermore, there are two representation types for the latent vector. Users can interact with them either as seeds or directly as vectors.
If the user wants to interact with the Generator based on a vector representation, the user can randomly sample vectors. As the dimensionality of the vector is too high this representation does not allow the user to traverse the latent space manually. Nevertheless, it is possible to do latent space traversals, through the animation part of the widget. The user sets a speed and then can start an animation. This animation creates an interpolation from the current vector to another randomly sampled vector. Once the next vector is reached a new destination is sampled. The animation has three modes stop, animate or step. If the animation is set to step, it will only move toward the new vector when the speed parameter is changed. But, when the animation mode is set to animate, the interpolation continues until the user stops the animation. The user can set both the current vector, the destination vector, speed and animation setting with OSC messages.

The seed representation, represents the vectors by seeds of the random sampling algorithm, hence it is possible to move around in this representation as it is one dimensional. For that reason we allow the user to explore the latent space through a dragging action. We interpolate between seeds n and n+1 by sampling the vectors from both seed n and seed n+1 and taking the weighted average. This representation also allows for latent space traversal with the same interface as the vector widget, where the interpolation moves between seeds. The interface for the seed is mainly taken from the visualization interface for StyleGAN3 [34]

The OSC Menu receives the following message types:

- The **seed** is a floating point number which is used as the seed in the latent space traversal when using the ’seed representation’.

- The **vec** field receives a list of floating point numbers with the same length as the latent vector for the StyleGAN2 model which normally is 512. It is the input for the model to generate images when using the 'vector representation'.
- The **project** menu item receives a boolean value, toggling whether the latent vector should be protected into the style (W) space.
- To remotely toggle the animation mode an integer can be send to the **anim** field, where 0 means stop, 1 means animate and 2 means step.
- **speed** receives a floating point number, and is one of the main parameters that can be used to make the animation react to an external signal. Where the slider in the interface is capped between 5 and -5, using OSC there is no limit to the value.

Truncation & Noise Widget

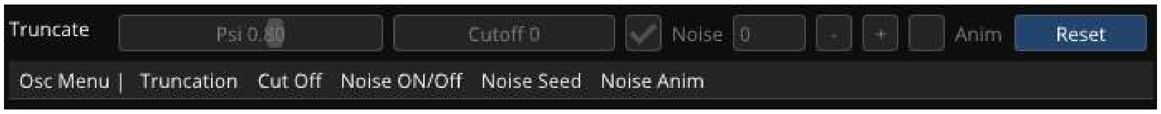


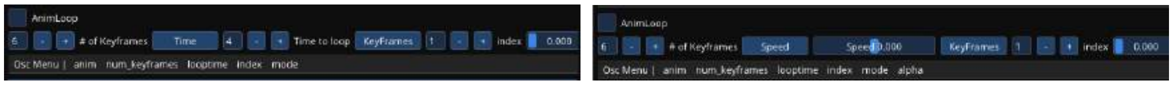
Figure 4.11: Truncation & Noise-Widget available in *Autolume-Live 2.0*

In addition to the latent vector StyleGAN2 has the parameters of truncation and the noise [2, 35]. We allow the users to adjust those to their needs using sliders for the truncation level and the truncation cutoff and for the noise we allow the users toggle on or off the stochastic noise and also, animate it, by sampling new noise every frame.
Generally speaking the truncation defines the diversity of the images that are generated. Keeping the truncation value close to 0 results in less variety but higher realism in the visuals. Setting it to 0 will render the same image with any seed, which corresponds to the average image the model extracted during training. Using the truncation cut off we can set how deep into the network the diversity should be configured This could allow us to have a high value on the lower levels creating less realistic general structures, but using the default values for the later layers meaning that the rendered texture is still realistic.
The noise parameter sets the seed for the noise that is injected during rendering. It can be toggled off, creating smoother images, or animated, meaning the noise seed is changed every rendering pass, resulting in small changes in the images every frame.
The OSC Menu receives the following message types:

- The **Truncation** tab receives a floating point number. In the interface the truncation slider is capped at -2 and 2, but with the OSC signal you can send values outside that range.
- **Cut Off** receives an integer between 0 and the number of layers in the current model, the program will automatically clip values below 0 or above the layer count.

- **Noise On/Off** receives a boolean, toggling the use of noise injections throughout the network.
- The **Noise Seed** parameter receives an integer that is used as the seed to sample the noise for the noise injection.
- The **Noise Anim** receives a boolean, toggling whether the noise should be animated, i.e. resampled every frame.

Looping Widget



(a) Time Based (b) Speed Based

Figure 4.12: Looping-Widget available in *Autolume-Live 2.0*

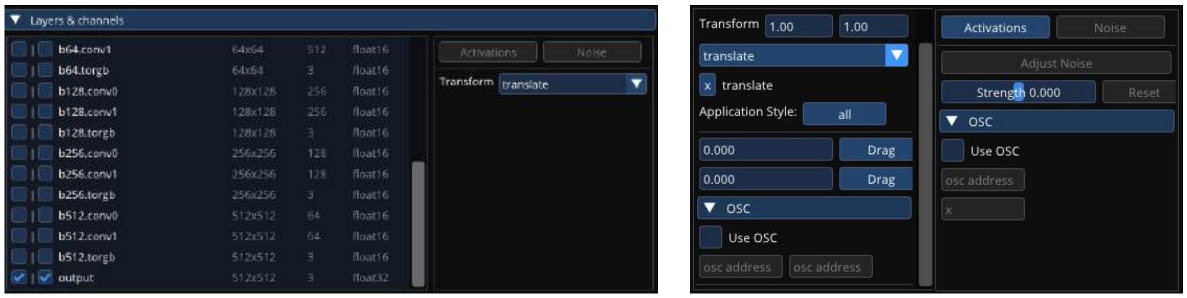
Although most mapping will probably be done outside of *Autolume-Live*, we want users to be able to program simple visual performances in the system itself. One common approach of VJs is to create visual loops and mapping their speed to the audio. This widget allows users to do so in the latent space. Users can define loops, with a number of keyframes. These keyframes can either be defined by loading vectors from a file, snapping the current vector, or using a toned down version of the latent widget, without the animation components. The loop can be played either in a set amount of time or given a speed parameter. When switching between the two looping styles the widget will either allow the user to set the time a loop takes in seconds or the speed, which modulates the step size in the interpolation (Figure 4.12).
The OSC Menu receives the following message types:

- The **anim** parameter receives a boolean and toggles if the rendered visuals should come from the loop defined by the keyframes from the looping widget.
- **num_keyframes** receives an integer and defines how many keyframes are used during the loop.
- The **looptime** parameter receives a floating point number and defines the amount of time a loop takes when using the time-based mode.
- The **speed** parameter receives a floating point and defines the speed of the loop when using the speed-based mode.
- With the **index** parameter, which receives integers clipped to be between 0 and the number of keyframes, users can jump to a specific keyframe in the loop.

- The **mode** parameter toggles between the time-based and speed-based loop, and receives a boolean.

- With the **alpha** parameter the user can specify the position when interpolating between two keyframes, with 0 being at the beginning of the interpolation and 1 being the end arriving at the new keyframe. It receives a floating point number that is being clipped between 0 and 1.

Layer Widget



☐	☐	b64.conv1	64x64	512	float16
☐	☐	b64.torgb	64x64	3	float16
☐	☐	b128.conv0	128x128	256	float16
☐	☐	b128.conv1	128x128	256	float16
☐	☐	b128.torgb	128x128	3	float16
☐	☐	b256.conv0	256x256	128	float16
☐	☐	b256.conv1	256x256	128	float16
☐	☐	b256.torgb	256x256	3	float16
☐	☐	b512.conv0	512x512	64	float16
☐	☐	b512.conv1	512x512	64	float16
☐	☐	b512.torgb	512x512	3	float16
✓	✓	output	512x512	3	float32

Transform translate

Transform 1.00 1.00

translate

x translate

Application Style: all

0.000

0.000

☐ Use OSC

Strength 0.000

☐ Use OSC

Figure 4.13: Layer-Widget available in *Autolume-Live 2.0*

The Latent and truncation widget already allow the users to manipulate the parameters that StyleGAN2 makes available to its users. But with the techniques described in Section 3.1 we open up further ways of manipulating the model on a layer by layer basis. As described in Section 2.3 every layer impacts the resulting output differently. By manipulating the layers output with Network Bending the user can create new visuals that would not be capable with StyleGAN2 alone. To do so, the user can select a layer and add transformations. Each transformation has a set amount of parameters. The parameters can be adjusted with OSC.
In addition to the activations, every layer also has stochastic noise. This noise effects features randomly on every representation level, e.g. in the last layers the position of birth marks. Normally the magnitude of the noise is set, but we allow users to change it. The right of the two checkboxes is used select which layer the user adds manipulations to or adjusts the noise of.
Furthermore every layer also has a visual representation, although only the final layer creates the realistic output, the intermediate layers could also be interesting for visual performances, hence we allow users to choose which layer should be displayed by clicking the left most check box for the corresponding layer.

Lastly, we allow users to change the output dimensions of every layer. This allows users to resize the output, making it possible to generate non-square images and even upscale or downscale the output. We also invite users to change the dimensions between layers and see what effect the different layers have. In our experiments we saw that first downscaling the output in lower layers and then upscaling it back to its original size in later layers creates more abstract versions of the content.

Adjustment Widget

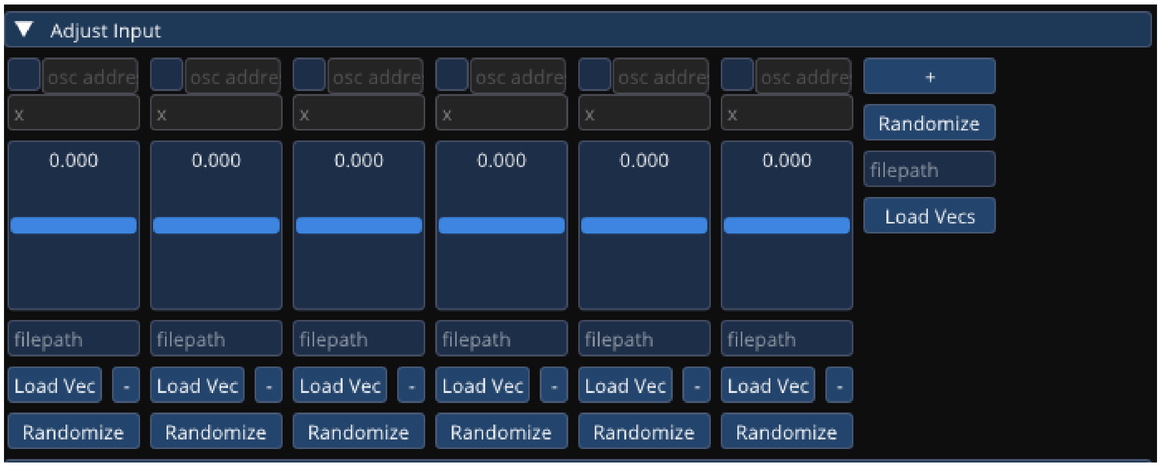


Figure 4.14: Adjustment-Widget available in *Autolume-Live 2.0*

We want users to be able to adjust the content of the synthesized images. We already mapped random directions and GANSpace [8] directions to sliders in *Autolume-Live 1.0*. Now, we allow users to add a variable amount of sliders for the direction vectors. Although the directions are random on initialization, it is possible to load either singular vectors to every slider or load a list of vectors at once. When loading a list of vectors, the interface automatically adds the number of vectors found in the file. This interface in combination with GANSpace, which can be accessed in our pre-performance interface (Section 4.3.2), allows users to edit images in an interpretable fashion. Every slider can be mapped to given their own OSC-channel, receiving a list of floats with the same size as the dimensionality of the latent space.

Preset Widget

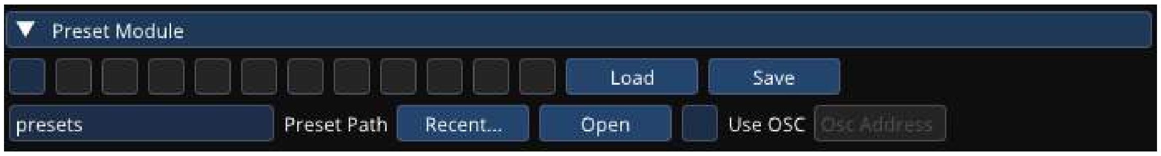


Figure 4.15: Preset-Widget available in *Autolume-Live 2.0*

While the original version of StyleGAN2 only saved the current latent vector when saving presets, the new version saves the state of the interface and the user can freely save and load states during their practice or performance. This allows for scene changes which are a crucial part for performances, as we can change multiple parameters at once, switching between completely different visual mappings.

Audio Widget

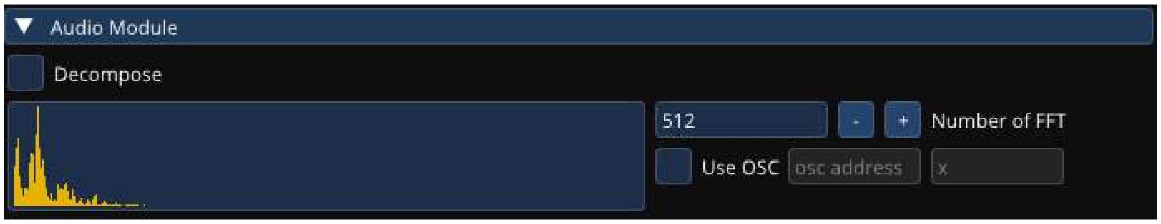


Figure 4.16: Audio-Widget available in *Autolume-Live 2.0*

The audio widget displays the frequency of the input audio signal. It uses the computers default audio in device, which without any changes is the microphone. We compute the FFT on the audio signal and the user can choose how many frequency bands it should compute. Furthermore, we found an efficient algorithm for harmonic percussive decomposition, that allows users to divide the frequency space into two. While it does not effect the performance drastically, the harmonic-percussive decomposition's quality is noise compared to performing it on a complete audio file and sometimes struggles distinguishing between a short percussive hit or input noise.
The resulting signal(s) can then be send out through an OSC-Message to other features in *Autolume-Live*.

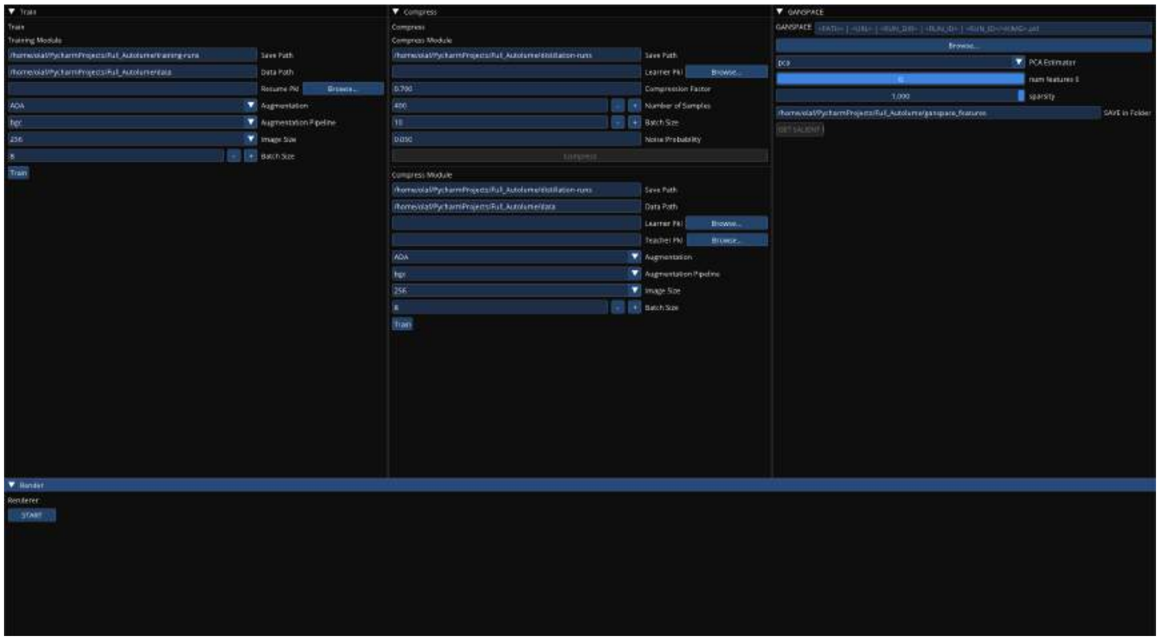
We originally included a widget to record the audio and visuals live, but it reduced the performance speed by 5-10 FPS. Recording it through a third-party software such as OBS or TouchDesigner, allows the same functionality without the drop in performance.

4.3.2 Pre-Performance Interface

While our main focus with *Autolume-Live* was to create an interface for live visual creation, offloading model training and other tasks to existing services, we introduced a minimal interface to train models, compress them to achieve faster rendering speeds, and extract meaningful directions using GANSpace.

Training Widget

Before entering the performance tool, the user can train a model locally with the interface seen in Figure 4.17. While services like RunwayML [50] made training StyleGAN2 accessible to the general public, we want to users to be able to do everything locally and contained in our interface. Furthermore, in addition to the basic StyleGAN2-ada [35] training loop we



▼ Train
Train
Training Module
/home/olaf/PycharmProjects/Full_Autolume/training-runs Save Path
/home/olaf/PycharmProjects/Full_Autolume/data Data Path
Resume Pkl Browse...
ADA Augmentation
bgc Augmentation Pipeline
256 Image Size
8 Batch Size
Train

▼ Compress
Compress
Compress Module
/home/olaf/PycharmProjects/Full_Autolume/distillation-runs Save Path
Learner Pkl Browse...
0.700 Compression Factor
400 Number of Samples
10 Batch Size
0.050 Noise Probability
Compress
Compress Module
/home/olaf/PycharmProjects/Full_Autolume/distillation-runs Save Path
/home/olaf/PycharmProjects/Full_Autolume/data Data Path
Learner Pkl Browse...
Teacher Pkl Browse...
ADA Augmentation
bgc Augmentation Pipeline
256 Image Size
8 Batch Size
Train

▼ GANSPACE
GANSPACE <PATH> | <URL> | <RUN_DIR> | <RUN_ID> | <RUN_ID>/<KIMG>.pkl
Browse...
pca PCA Estimator
0 num features 0
1.000 sparsity
/home/olaf/PycharmProjects/Full_Autolume/ganspace_features SAVE in Folder
GET SALIENT

▼ Render
Renderer
START

Figure 4.17: We allow users to train and compress models, and apply GANSpace [8] to models in a start-up menu, before entering the Performance Interface.

allow the user to pick the augmentations that should be applied during training and also switch to the *Differential Augmentations* [36] we described in Section 2.3. Lastly, we allow the user to enable the adapted training procedures that we described in Section 2.3.4 and the top-k update rule [38].
With these training frameworks, users will be able to generate high-fidelity images with a low amount of data. Users can reasonably train models on large datasets as well as on their own works, as the data necessary is reduced to the hundreds, compared to the over 1k images originally needed to train a GAN.

Compression Widget

Although StyleGAN2 can already run in realtime without further compression, we allow users to perform pruning and distillation to their models following the content aware GAN compression framework. While, it was originally implemented for vanilla StyleGAN2, we reimplemented it to incorporate NVIDIA's StyleGAN2-ada repository as its inference speed is faster than the implementation they used. Furthermore, we allow users to apply the same augmentation techniques, as in the training widget, in the distillation process, which can speed up the training process. If the teacher was trained with the projected GANs framework [37], the corresponding Discriminator is also used in the distillation process.

GANSpace Widget

Users can perform fine tuned adjustments on the latent vectors in *Autolume-Live*. Without further configuration random vectors are used, but we want users to be able to perform meaningful adjustments to the output. With the previous version of Autolume-Live, users had to run the GANSpace algorithm [8] on their own which needs them to run command line commands and perform changes to the code to perform them on their own models. Hence, we allow them to do so in our interface before starting a performance. The user can decide how many directions they want to extract and they can load them in the adjustment widget.

4.4 Applications

We iteratively expanded *Autolume-Live's* features, moving from the 1.0 version that contained adhoc audio and controller mappings to a more generic neural video generation engine that includes its own interface, but also can be used in combination with other software. This allows users to do audio analysis and image post-processing in their preferred existing environments, e.g. TouchDesigner or PureData. In this section, we are showcasing some examples using *Autolume-Live* without any other software, with TouchDesigner and with PureData.

4.4.1 Standalone

On its own, the easiest way to create an audio-reactive performance is linking the audio widget's output to parameters. An example can be seen at `https://youtu.be/FOxX5MLET2E`. In it, we show a few mappings that can be used to generate live audio-reactive visuals using our system. We map the mean amplitude of the FFT from the audio-widget to the speed of the latent space traversal, we explore a couple layer transformations. For the vertical flip transformation or transformations like it that can be turned either on or off, the signal is considered as a boolean expression. In the example, we showcase it by activating the flip when the amplitude exceeds a threshold.

4.4.2 Pure Data Integration

Pure data is an open source visual programming language for multimedia. It works with patches that can be linked together to create a data and transformation stream. In the example shown in Figure 4.18 we created a PureData patch, in which we can either read the audio stream from the microphone or an audio file. We extract the Root-Mean-Square (RMS) of the signal's amplitude. We further process the signal sending out two control-signals to *Autolume-Live*. The first simply being a cumulative average over a small time window, to smoothen the RMS. For the second signal, we perform smoothing on the ramp-

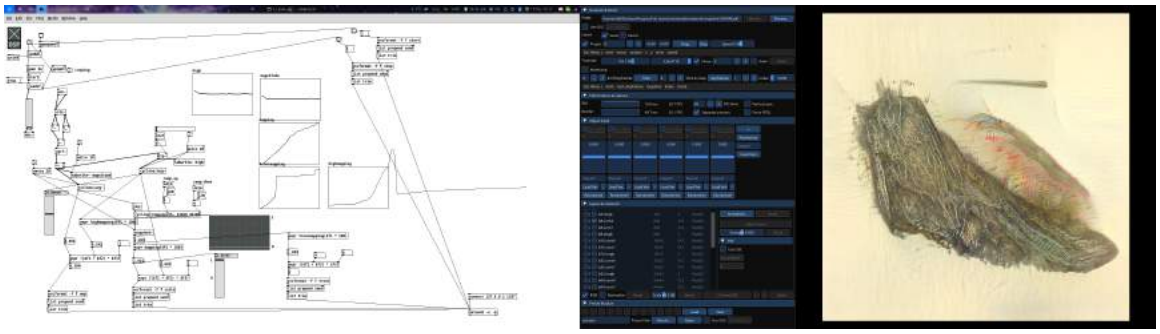


Figure 4.18: Audio-Visual sample created with *Autolume-live* and PureData. `https://youtu.be/qq9LhOmgCds`.

up and ramp-down of the signal, which can create a lingering effect. Another benefit using PureData for the audio-analysis is, that we can perform more complex mappings, by drawing mapping lines. For example, in this case, we want the RMS signal, which is normally between 0 and 1, to be in a different value range, and also add an ease in and ease out effect. This mapping is visible in the mapping array displayed in the PureData window.

We then assign these control-signals two a variety of parameters in *Autolume-Live*. We assign the first signal to the speed of the animation and the noise strength of one of the lower layers. We realized in the example that we would like the speed to be effected more strongly by the signal, hence we multiply it inside *Autolume-Live* with a factor of 3.
The second signal we map to a rotation on the activations. Again, as our system is live, we can do further adjustments. In this case, the rotation distracted to much, hence we first applied it only to a subset of activations and then completely turned it off.
Lastly, at the end of the example, we switch between models. The first time loading a model creates a short delay, but once loaded it stays in the cache allowing us to switch between models seamlessly. Hence, we recommend users to open the models that are used during a performance before, so the models can be loaded instantaneously.

4.4.3 TouchDesigner Integration

We take advantage of the efficient and accurate audio analysis tools in TouchDesigner, to perform more complex audio-visual mappings [20]. TouchDesigner is a node-based development environment that is used for live audio-visual performances.

Audio Processing

While our standalone use-case, only used the magnitude of the audio, in the example shown in Figure 4.19 we extract low, mid and high frequencies, and we detect the percussion separated in kicks, snares and rhythm. An example patch for the audio analysis can be seen in Figure 4.20

Figure 4.19: Audio-Visual sample created with *Autolume-live* and TouchDesigner. `https://vimeo.com/772213895`

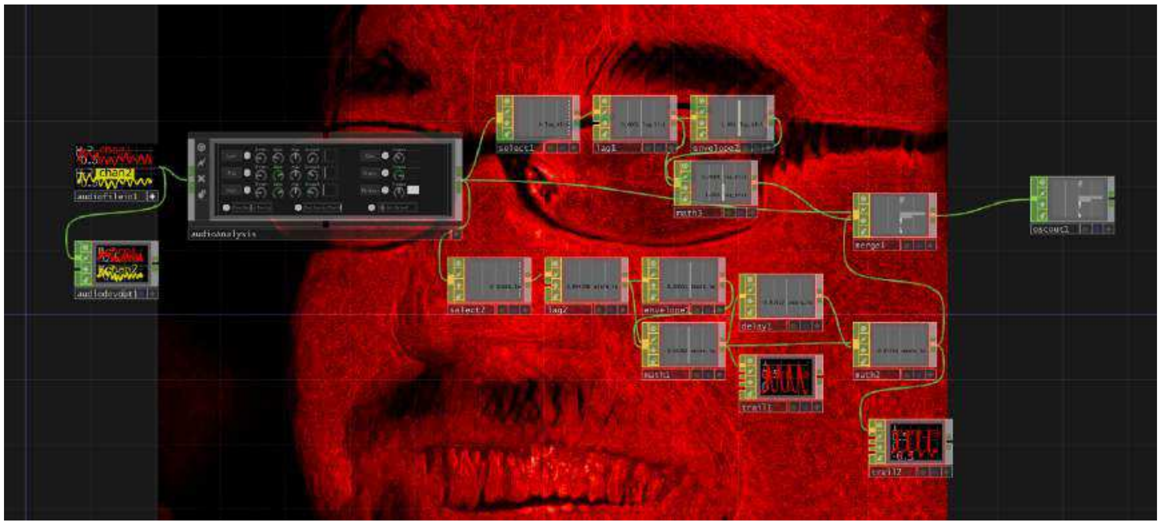


Figure 4.20: Example of an audio analysis pipeline in TouchDesigner.

We then send the extracted values to *Autolume-Live* through OSC, linking the amplitude of the low frequencies to the speed of the latent interpolation, the kicks to a zoom in the activations and the kicks to a thresholding of the activations, resulting in more abstracted versions of the images.

Video PostProcessing

Autolume-Live 2.0 sends the rendered images through an NDI stream. This allows us to perform post-processing on them in TouchDesigner. For this example, we perceived the audio to be dark and found a red color scheme to fit better. With all the algorithms available in

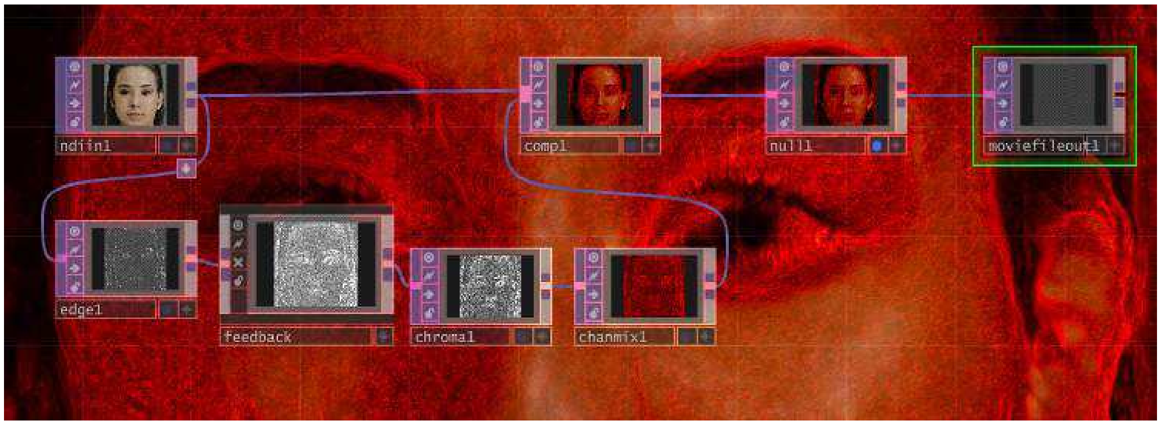


Figure 4.21: Example of an image postprocessing pipeline in TouchDesigner.

TouchDesigner, users will be able to enhance the visual expressiveness of *Autolume-Live*. The patch we used for our example can be seen in Figure 4.21.

4.4.4 TouchDesigner & Ableton-Live Integration

Figure 4.22: A clip from a live rehearsal mapping MIDI to visuals in *Autolume-Live* using TouchDesigner and Ableton-Live. `https://youtu.be/1yRP7lNS5qs`

We also experimented performing mappings on a live set, directly using the MIDI signals from Ableton Live instead of using the audio. This can open up further controls and reduce the latency and noise in the signal (Figure 4.22). In this example the MIDI signal from 8 Tracks are sent to TouchDesigner using the TDAbleton plugin. In our case the different tracks represent different parts in the music: Bass Drum, Snare, Hi-Hat, Percussion, Bass, Melody, Arpegiator, Chords.

Following we can process the signal and send it to *Autolume-Live* via OSC. In the example, we showcase a rehearsal we had, where we adjusted mappings, switched between models and presets, to create visuals for the audio that was produced live. We recorded the video feed inside TouchDesigner by accessing the NDI stream send by *Autolume-Live.*
We start off by a fade from black, and decided throughout the song to move from more abstract images to realistic images and back. First we use a model that is trained on dancers, originally used for a piece called 'Longing and Forgetting' [76]. To make the visuals more abstract we do not show the final layer, but render a feature map of the last convolutional layer. This results in vibrant colors where the dancer would be. Once more of the tracks come in we switch to the output of the model resulting in the dancers to show with their actual texture. Following, we tried out a model, trained on the experimental film 'Tarantella' [77], but decided to move to a model trained on the FFHQ dataset [33].
Normally, the model generates realistic portraits, but for this rehearsal we decided to apply Network Bending to both corrupt the image, resulting in morphed faces, and to perform scaling, rotation and translation, for audio-reactive movements. Close to the end the image turns into noise for a few instances. This occurred as we were modulating the scaling factor, and if it is set too low the generated image turn into noise, with colors representing those found in the original dataset.

Chapter 5

Conclusion

5.1 Discussion

We explored the space of real-time audio-visual generation using Deep Generative Models, and created a co-creative tool incorporating contemporary AI research, called *Autolume-Live*. As discussed in Section 2.2, for a tool to support artists it is necessary that their needs to understand the tool and can control it to its will. By using it internally for both live and pre-recorded performances. Furthermore, we are in the process of collaborating with VJs and other visual artists to explore additional usecases and features that will improve usability. So far the research of interpretable AI has been separate from creating user oriented tools and *Autolume-Live* is a first step to creating new types of DL-based interfaces without the need for any coding experience.

Our system was iteratively developed, where *Autolume-Live 1.0* used predefined mappings between an Audio- and Controller module to a neural rendering engine based on StyleGAN2. The Controller-module utilizes a hardware MIDI-controller allowing users to navigate the latent space and perform Network Bending with a physical interface. The Audio-mapping was hard-coded to allow two audio-reactive mappings, i.e. performing a random walk based on the amplitude of the signal, or displaying the weighted average of a variety of keyframes based on the notes present in the signal. The current version, *Autolume-Live 2.0*, switched from an adhoc mapping to a GUI that allows users to create their own mappings with additional parameters that can be controlled, for more details review Section 4.3. Furthermore, we include a ”pre-performance” interface, to train and compress models, and find salient directions in the latent space which can be loaded during the performance.

Our work has culminated in a functional interface that we have used to create installations and will be continuously updated with features based on either findings through internal usage and through future user studies including expert workshops. Currently, we see that some of the parameters in the interface have a high learning curve. Which we recognize originates from us using their technical names, e.g. truncation, instead of a more general term, e.g. variety. Furthermore, the inclusion of tool tips would further improve the accessibility of our system. Additionally, the system is currently released through gitlab: `https://gitlab.com/jkraasch/autolumelive_colab`, which allows users to help with improving our system. The repository does not include an installation file. So the local setup process, still requires usage of the terminal, which we plan on simplifying in the near future.

Lastly, with the rise of larger generative systems, research in this domain raises ethical questions on ownership and environmental impact. As our tool is used to augment the creative process and not automate it, we believe that in our case the question of ownership is easier to answer. Without a user choosing models and defining mappings the system does not create any content. Hence, it gives the agency to the artist and hence it can be compared to using other VJing tools, where the ownership is given to the user. But, what can blur the lines of ownership is the origin of the models. Recently DALL-E 2 [78] and Stable Diffusion [11] were criticized for obscuring the origin of their data and training their models on images that might have a copyright and without acknowledgment. In our case users will train their own models or use pretrained models. Hence questions regarding copyright and ownership falls back on the user. Similar to re-purposing other peoples creative works, the user has to be aware of licensing.
Some of the code our system is based on falls under the NVIDIA Source Code License-NC. Which restricts any derivative work to be non-commercial, which does not constrain our work in researching a live VJing tool. But, since the copyright and ownership of the rendered results is still a gray area, we need to follow the legal developments around this topic, when it comes to commercializing the usage or results of using our tool. For now, there have been no actions restricting artists to distribute and monetize their work when using code under the same NVIDIA Source Code License-NC.

5.2 Future & Ongoing Work

We see multiple ways *Autolume-Live* can be expanded and are currently working on new installations, features and collaborations. It is currently used to revive collaborative work between Matt Gingold, Philippe Pasquier called "Longing + Forgetting" [79]. In the process of developing this installation, we again saw the incredible potential of being able to sketch out performances in real-time, while also finding new features that could improve the usefulness of the system. For example, we are working on introducing a module that allows users to project images and find the closest match in the latent space, so they can more intuitively define keyframes.

As of writing this, we are working together with local Vancouver artists on aggregating data collaboratively. In that work we want to showcase an ethical approach on training a Deep Generative Model on artists work and the new capabilities it opens up. In this collaborative work we are working on making the artists more AI literate and make them feel empowered using our system rather than taken advantage of, which was a shared experience from a variety of artists during the release of Stable Diffusion [11].
Specifically, we collaboratively aggregate a corpus of both the artists' archives and newer works. Then we take it upon ourselves to train models with their data. In this preparation period, we regularly check in with the artists, showcasing the quality of the models we have trained so far and discuss further steps, e.g. if more pictures of their work could improve the model, further edits we could perform on the models such as described in Figure 4.6. Once the artists have confirmed their support we would then generate visual performances using *Autolume-Live*.

Furthermore, we are starting to collaborate with VJs and are in the process of developing performances and other art works with them. Through the work with VJs we expect to receive further feedback, especially towards the usability of the UI. As we discussed in Section 5.1, our research intersects with HCI and changes to the interface could be made to improve usability.
In addition to our empirical work reflecting on the current state of *Autolume-Live*, we are planning structured workshops and user studies with expert groups. We intend to use the workshops to create awareness around the possibilities co-creative Deep Learning systems can open up, teach them how to use our system, and get feedback on both the interface and features to do further adjustments.
We believe that it might be interesting to see if a change in the interface paradigm could improve usability. MaxMSP, Touch Designer and Pure Data are all node-based interfaces. This seems to be a trend in VJing software,hence it might make our tool more accessible to follow a similar design principle.

Parallel to the work we already initiated we could see further improvements by introducing further Deep Learning models. For example, the Audio Module could be replaced with a Neural Network to harness more complex audio feature extraction. More high-level characteristics of the music could be extracted, such as the emotions and themes, which could then influence the content of the video. Furthermore, with the rapid development in text based image generation, we could see users explore the latent space to define keyframe through prompts.
We could see further improvement in the speed of our system, either through hardware developments or developments in the DGM research. As of writing we are also experimenting with real-time super resolution models which would allow us to use lower resolution

StyleGAN2 models, which could lead to improved performance.

Lastly, our work is still in development and is hosted only as a git repository (`https://gitlab.com/jkraasch/autolumelive_colab`). We plan on releasing the system with its own installer and are eager to see how the community will use our tool.

5.3 Conclusion

Music is often accompanied with visuals, be it in the form of music videos, or performances by VJs. We set out to develop a system that tackles the task of AI based audio-reactive visual generation. Prior to our work, algorithms for video generation are only capable of generating short clips, and were unable to create long form content as needed in this task. After reviewing prior artistic approaches for audio reactive visuals, which were capable of longer form content but required coding experience, we came to the conclusion that there is a gap between the generative power of Deep Generative Models and their usability for artistic practices. Hence, we have focused our work on VJing and creating a co-creative tool using Deep Generative Models for their practice. Following the findings of Hook et al [15], our system had to fulfill the following requirements for the possible interactions:
The output should be **immediate**, there is a preference for **haptic feedback**, there should be a set of powerful and varied **manipulation of the medium**, interactions should be **parallel**, and the interface should be **reconfigurable** according to the needs of the artist.

With *Autolume-Live* we believe, we have addressed these needs. Users can improvise and manipulate the medium live, either in the interface itself or using other software through an OSC-messaging protocol. This allows users to control features in a suitable environment and also use physical controllers for their live performances. By deploying Network Bending and GANSpace the user has control over the visual performance in a responsive and understandable manner. Furthermore the user can store and call upon checkpoints and switch between generative models on the fly allowing them to practice before performances and fine-tune the visual content to fit the current musical content.
We have showcased both the usability and usefulness of our tool through the creation of multiple installations and through the examples described in this thesis (Section 4.2, Section 4.4). Additionally, further collaboration both with VJs and visual artists are in process. We hope that the work with the VJs will help us to further improve our system. The goal of collaborating with visual artists is both to create new installations, while also exploring the space of ethical data use in art using co-creative AI solutions, i.e. finding ways to make artists feel empowered using models trained on their archives, instead of feeling like their art is being stolen.

Lastly, as the work on *Autolume-Live* still continues, we are eager to see further development in the space of co-creative systems, especially with a additional focus given on the human interaction with AI solutions [16]. With the release of DALL-E 2 and Stable Diffusion [78, 11], we have seen a variety of interfaces and software being published that allow users to generate images based on text prompts. We believe that this showcases that we have reached a turning point, where more focus will be put on Human-Centered AI (HAI), and creating usable and useful AI solutions. We also see this outside of the creative field with the release of ChatGPT [80] and the growing interest in Large Language Models as search engines.

Bibliography

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[2] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119, 2020.

[3] Terence Broad, Frederic Fol Leymarie, and Mick Grierson. Network bending: Expressive manipulation of deep generative models. In *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*, pages 20–36. Springer, 2021.

[4] Matt Siegelman. Deep music visualizer. `https://github.com/msieg/deep-music-visualizer`. Accessed: 2022-2-8.

[5] Mikael Alafriz. Lucid sonic dreams. `https://github.com/mikaelalafriz/lucid-sonic-dreams`, March 2021. Accessed: 2022-2-6.

[6] Hans Brouwer. Audio-reactive latent interpolations with StyleGAN. In *Proceedings of the 4th Workshop on Machine Learning for Creativity and Design at NeurIPS 2020*, December 2020.

[7] Jonas Kraasch and Philippe Pasquier. Autolume-live: Turning GANs into a live VJing tool. *Proceedings of the 10th Conference on Computation, Communication, Aesthetics & X*, pages 168–185, 2022.

[8] Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. GANSpace: Discovering interpretable GAN controls. In *Proc. NeurIPS*, 2020.

[9] Obvious. La famille de belamy. `https://obvious-art.com/la-famille-belamy/`.

[10] Mario Klingemann. Uncanny mirror. `https://artsandculture.google.com/asset/mario-klingemann-uncanny-mirror-mario-klingemann/AQGCavNzwcOlEg?hl=en`, 2019.

[11] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[12] Hugging Face. Hugging face – the ai community building the future. `https://huggingface.co/`. Accessed: 2022-8-2.

[13] AUTOMATIC1111. Stable diffusion web ui. `https://github.com/AUTOMATIC1111/stable-diffusion-webui/`. Accessed: 2022-8-2.

[14] Prashant Gohel, Priyanka Singh, and Manoranjan Mohanty. Explainable AI: current status and future directions. *CoRR*, abs/2107.07045, 2021.

[15] Jonathan Hook, David Green, John McCarthy, Stuart Taylor, Peter Wright, and Patrick Olivier. A VJ centered exploration of expressive interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1265–1274, New York, NY, USA, May 2011. Association for Computing Machinery.

[16] Wei Xu. Toward human-centered ai: A perspective from human-computer interaction. *Interactions*, 26(4):42–46, jun 2019.

[17] Matthew James Wright and Adrian Freed. Open SoundControl: A new protocol for communicating with sound synthesizers. In *ICMC*, 1997.

[18] Robyn Taylor, Pierre Boulanger, and Daniel Torres. Real-time music visualization using responsive imagery. In *8th International Conference on Virtual Reality*, pages 26–30, 2006.

[19] Resolume. Resolume. `https://resolume.com/`. Accessed: 2021-9-25.

[20] Derivative. Touch designer. `https://derivative.ca/`. Accessed: 2021-9-25.

[21] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *CoRR*, abs/1706.07979, 2017.

[22] Zhuwei Qin, Fuxun Yu, Chenchen Liu, and Xiang Chen. How convolutional neural network see the world - A survey of convolutional neural network visualization methods. *CoRR*, abs/1804.11191, 2018.

[23] Jiawei He, Andreas M. Lehrmann, Joseph Marino, Greg Mori, and Leonid Sigal. Probabilistic video generation using holistic attribute control. *CoRR*, abs/1803.08085, 2018.

[24] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *CoRR*, abs/2105.05233, 2021.

[25] Lars Ruthotto and Eldad Haber. An introduction to deep generative modeling. *GAMM-Mitteilungen*, 44(2):e202100008, 2021.

[26] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[27] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *CoRR*, abs/2003.05991, 2020.

[28] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv e-prints*, page arXiv:1701.07875, Jan 2017.

[29] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs. *arXiv e-prints*, page arXiv:1704.00028, Mar 2017.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[31] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

[32] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *arXiv e-prints*, page arXiv:1710.10196, Oct 2017.

[33] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1812.04948, Dec 2018.

[34] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *CoRR*, abs/2106.12423, 2021.

[35] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. *Advances in Neural Information Processing Systems*, 33:12104–12114, 2020.

[36] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient GAN training. *CoRR*, abs/2006.10738, 2020.

[37] Axel Sauer, Kashyap Chitta, Jens Müller, and Andreas Geiger. Projected gans converge faster. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[38] Samarth Sinha, Zhengli Zhao, Anirudh Goyal ALIAS PARTH GOYAL, Colin A Raffel, and Augustus Odena. Top-k training of gans: Improving gan performance by throwing away bad samples. *Advances in Neural Information Processing Systems*, 33:14638–14649, 2020.

[39] Dinesh Acharya, Zhiwu Huang, Danda Pani Paudel, and Luc Van Gool. Towards high resolution video generation with progressive growing of sliced wasserstein gans. *CoRR*, abs/1810.02419, 2018.

[40] Ruslan Rakhimov, Denis Volkhonskiy, Alexey Artemov, Denis Zorin, and Evgeny Burnaev. Latent video transformer. *CoRR*, abs/2006.10704, 2020.

[41] Ruslan Rakhimov, Denis Volkhonskiy, Alexey Artemov, Denis Zorin, and Evgeny Burnaev. Latent video transformer. *CoRR*, abs/2006.10704, 2020.

[42] Aidan Clark, Jeff Donahue, and Karen Simonyan. Efficient video generation on complex datasets. *CoRR*, abs/1907.06571, 2019.

[43] Ivan Skorokhodov, Sergey Tulyakov, and Mohamed Elhoseiny. StyleGAN-V: A Continuous Video Generator with the Price, Image Quality and Perks of StyleGAN2. *arXiv e-prints*, page arXiv:2112.14683, December 2021.

[44] Tim Brooks, Janne Hellsten, Miika Aittala, Ting-Chun Wang, Timo Aila, Jaakko Lehtinen, Ming-Yu Liu, Alexei A. Efros, and Tero Karras. Generating long videos of dynamic scenes, 2022.

[45] Songwei Ge, Thomas Hayes, Harry Yang, Xi Yin, Guan Pang, David Jacobs, Jia-Bin Huang, and Devi Parikh. Long video generation with time-agnostic vqgan and time-sensitive transformer, 2022.

[46] Chenfei Wu, Lun Huang, Qianxi Zhang, Binyang Li, Lei Ji, Fan Yang, Guillermo Sapiro, and Nan Duan. GODIVA: generating open-domain videos from natural descriptions. *CoRR*, abs/2104.14806, 2021.

[47] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to simulate dynamic environments with gamegan. *CoRR*, abs/2005.12126, 2020.

[48] Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, Devi Parikh, Sonal Gupta, and Yaniv Taigman. Make-A-Video: Text-to-Video Generation without Text-Video Data. *arXiv e-prints*, page arXiv:2209.14792, September 2022.

[49] Derrick Schultz. Artifical images (channel). `https://www.youtube.com/c/ArtificialImages`, 2022. Accessed: 2022-11-10.

[50] Runway AI, Inc. Ruway. `https://runwayml.com/`, 2022. Accessed: 2022-2-10.

[51] Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 709–718. IEEE, 2018.

[52] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

[53] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[54] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[55] Yuchen Liu, Zhixin Shu, Yijun Li, Zhe Lin, Federico Perazzi, and S Y Kung. Content-Aware GAN compression. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2021.

[56] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou, Joshua B Tenenbaum, William T Freeman, and Antonio Torralba. Gan dissection: Visualizing and understanding generative adversarial networks. *arXiv preprint arXiv:1811.10597*, 2018.

[57] Daniel Roich, Ron Mokady, Amit H Bermano, and Daniel Cohen-Or. Pivotal tuning for latent-based editing of real images. *ACM Transactions on Graphics (TOG)*, 42(1):1–13, 2022.

[58] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9243–9252, 2020.

[59] Cameron Smith. Livegan: Programming a hardware midi controller for interacting with generative models. `https://www.dropbox.com/s/9jy99rxzse1lqze/neurips_cw_2020.pdf?dl=0`. Accessed: 2022-8-2.

[60] Michał Jaworski. pyimgui. `https://github.com/pyimgui/pyimgui`, 2013.

[61] Jianyu Fan, William Li, Jim Bizzocchi, and Philippe Pasquier. DJ-MVP an automatic music video producer. In *13th International Conference on Advanced in Computer Entertainment Technology*. unknown, October 2016.

[62] Hassan Ezzaidi, Mohammed Bahoura, and Glenn Eric Hall. Towards a characterization of musical timbre based on chroma contours. In *Advanced Machine Learning Technologies and Applications*, pages 162–171. Springer Berlin Heidelberg, 2012.

[63] Dan Ellis. Chroma feature analysis and synthesis “. `https://www.ee.columbia.edu/~dpwe/resources/matlab/chroma-ansyn/`, April 2007. Accessed: 2022-1-26.

[64] Cedric Oeldorf and Gerasimos Spanakis. LoGANv2: Conditional Style-Based logo generation with generative adversarial networks. September 2019.

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

[66] Brian McFee and Dan Ellis. Analyzing song structure with spectral clustering. In *ISMIR*, pages 405–410, 2014.

[67] DoodleChaos. I asked AI to make a Music Video... the results are trippy — youtube.com. `https://www.youtube.com/watch?v=0fDJXmqdN-A`, May 2022. [Accessed 11-Nov-2022].

[68] Pure Data; Pd Community Site. `https://puredata.info/`. [Accessed 11-Nov-2022].

[69] Jonathan Driedger, Meinard Müller, and Sascha Disch. Extending harmonic-percussive separation of audio signals. In *ISMIR*, pages 611–616, 2014.

[70] Dystopia sound and art festival. `https://www.dystopie-festival.net/2021/l`, 2021. Accessed: 2023-1-16.

[71] Light Up Kelowna. Light up kelowna. `https://fccs.ok.ubc.ca/about/events-workshops/light-up-kelowna/`, February 2022.

[72] WikiArt. WikiArt.org - Visual Art Encyclopedia — wikiart.org. `https://www.wikiart.org/`. [Accessed 23-Feb-2023].

[73] k4yt3x. Video2x. `https://github.com/k4yt3x/video2x/`. Accessed: 2023-1-16.

[74] Margaret Schedel, Phoenix Perry, and Rebecca Fiebrink. Wekinating 000000swan: Using machine learning to create and control complex artistic systems. In *11th International Conference on New Interfaces for Musical Expression, NIME 2011, Oslo, Norway, May 30 - June 1, 2011*, pages 453–456. nime.org, 2011.

[75] NewTek. Network device interface. `https://ndi.tv/`. Accessed: 2022-8-2.

[76] Mathew Gingold, Thecla Schiphorst, and Philippe Pasquier. Never alone. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, May 2017.

[77] Mary Ellen Bute and Ted Nemeth. *Tarantella : a Swift Moving Dance*. 1940.

[78] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[79] Matt Gingold and Philippe Pasquier. Longing + forgetting. `https://abbotsfordconvent.com.au/event/longing-forgetting/`. Accessed: 2023-2-13.

[80] OpenAI. Chatgpt: Optimizing language models for dialogue. `https://openai.com/blog/chatgpt/`. Accessed: 2023-2-13.

Appendix A

Code

The code for the newest *Autolume-Live* version can be found here: `https://gitlab.com/jkraasch/autolumelive_colab`