DHBW
Duale Hochschule
Baden-Württemberg

Developing a Diffusion Pipeline for Optimization of Video Game Asset Generation

from the course of studies Mobile Computer Science

at the Cooperative State University Baden-Württemberg
Ravensburg Campus Friedrichshafen

by

Valentino Pecchinenda

Danny Seidel

17.07.2024

Student ID, Course: 5906762, TIM21
7024923, TIM21

Supervisor at DHBW: Eric Balogh

Declaration of Authorship

Gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden- Württemberg vom 29.09.2017. Wir versichern hiermit, dass wir unsere Arbeit mit dem Thema:

Developing a Diffusion Pipeline for Optimization of Video Game Asset Generation

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Friedrichshafen, 17.07.2024

Valentino Pecchinenda

Danny Seidel

List of Figures

Code Snippets

Table of Contents

List of Acronyms

AI	Artificial Intelligence
AN	Artificial Neuron
ANN	Artificial Neural Network
CLIP	Contrastive Language–Image Pre-training
CNN	Convolutional Neural Network
DCGAN	Deep Convolutional Generative Adversarial Network
DM	Diffusion Model
GAN	Generative Adversarial Network
GPU	Graphical Processing Unit
ITI	image-to-image
LDM	Latent Diffusion Model
LoRA	Low Rank Adaptation
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
MSE	mean squared error
NFT	Non-fungible token
PAXL	Pixel Art XL
PCG	Procedural Content Generation
RPG	Role-Playing Game
ReLU	Rectified Linear Unit
SD	Stable Diffusion
SDE	Stochastic Differential Equation
SDXL	Stable Diffusion XL
TTI	text-to-image
VAE	Variational Autoencoder

VQ-VAE	Vector Quantized-Variational Autoencoder
VQGAN	Vector-Quantized Generative Adversarial Network

Abstract

Creating assets such as sprites and tiles poses a significant challenge in video game development, especially for developers with limited artistic skills or time. This thesis investigates the development of a diffusion pipeline to optimize the generation of low-resolution video game assets, leveraging state-of-the-art machine learning techniques. The proposed pipeline utilizes modern diffusion models to generate photorealistic and stylistically consistent images from textual descriptions. By integrating models such as Stable Diffusion and its enhanced version, Stable Diffusion XL (SDXL), along with fine-tuning techniques like Low Rank Adaptation (LoRA), we demonstrate the capability to automate and streamline the creation of video game graphics. The methodology includes initialization of diffusion pipelines, structured prompt creation, controlled seed generation, and detailed post-processing to yield high-quality, pixel-perfect backgrounds, characters, and animations. This work significantly reduces the manual effort and expertise required for asset creation, accelerating the development process and enhancing accessibility for a broader range of developers. The findings highlight the potential of AI-assisted tools in fostering creativity and efficiency in game design.

Die Erstellung von Assets wie Sprites und Tiles stellt in der Videospielentwicklung eine große Herausforderung dar, insbesondere für Entwickler mit begrenzten künstlerischen Fähigkeiten oder Zeit. In dieser Arbeit wird die Entwicklung einer Diffusionspipeline zur Optimierung der Erzeugung von Videospiel-Assets mit niedriger Auflösung untersucht, wobei modernste Techniken des maschinellen Lernens eingesetzt werden. Die vorgeschlagene Pipeline nutzt moderne Diffusionsmodelle, um fotorealistische und stilistisch konsistente Bilder aus textuellen Beschreibungen zu generieren. Durch die Integration von Modellen wie Stable Diffusion und seiner verbesserten Version, Stable Diffusion XL (SDXL), zusammen mit Feinabstimmungstechniken wie Low Rank Adaptation (LoRA), demonstrieren wir die Fähigkeit, die Erstellung von Videospielgrafiken zu automatisieren und zu rationalisieren. Die Methodik umfasst die Initialisierung von Diffusionspipelines, die strukturierte Erstellung von Prompts, die kontrollierte Erzeugung von Seeds und eine detaillierte Nachbearbeitung, um qualitativ hochwertige, pixelgenaue Hintergründe, Charaktere und Animationen zu erzeugen. Diese Arbeit reduziert den manuellen Aufwand und das erforderliche Fachwissen für die Erstellung von Assets erheblich, beschleunigt den Entwicklungsprozess und verbessert die Zugänglichkeit für ein breiteres Spektrum von Entwicklern. Die Ergebnisse unterstreichen das Potenzial von AI-gestützten Tools zur Förderung von Kreativität und Effizienz bei der Spieleentwicklung.

1. Introduction

In video game development, the artistry and design of sprites and tiles – critical components in numerous popular Role-Playing Games (RPGs) – present a mesmerizing challenge, particularly for developers lacking traditional artistic understanding or possessing limited patience. Despite the considerable progress in computer game graphics over the past decades, we have sustained this challenge.

Although modern RPGs contain advanced game engines, they often draw upon low-pixel, top-view graphics. They use tiles to construct varying landscapes and sprite sheets to animate characters within these worlds. These components' granular and manual creation consumes significant time and resources, thus posing a considerable challenge for video game developers.

Motivated by these challenges, this project explores an innovative solution anchored in modern diffusion models known for their capabilities to generate photorealistic images from textual descriptions. However, a severe problem persists within commonly utilized models that fail to create low-resolution pictures – a fundamental requirement for our context – due to considerable variance from their training data.

This thesis, therefore, advances an experimental approach to developing a diffusion pipeline capable of generating low-resolution tiles and spirits. The task's feasibility and performance remain uncertain, necessitating experimentation, optimization, and validation processes. This thesis underlines our journey towards achieving faster and more accessible video game development.

1.1. Problem Statement

Currently, low-resolution video game development involves designing tiles and sprite sheets. These create a video game world containing diverse scenery and characters. As the creation often requires creativity and artistic skill, many developers struggle with designing tiles and sprite sheets.

1.2. Research Objectives

The project aims to speed up the development of the mentioned graphical elements while making the generation more accessible. Thereby, video game developers do not require a designer during development. Should a team still include a designer, the generated graphics can inspire or accelerate the progress.

1.3. Thesis Scope

This thesis aims to create a diffusion pipeline that automatically generates video game tiles and sprite sheets to achieve the previously addressed goals. We focus on exploring different approaches to produce usable graphics with consistent results. Furthermore, we aim to achieve simple animation of game characters.

We discuss the reasons for choosing a pipeline approach over training a new diffusion model later. For reference, check Section 4.2.1.

2. Related Work

This chapter aims to present similar projects that either focus on analogous goals or demonstrate results usable in our development and optimization. Additionally, the chapter focuses on already existing solutions.

2.1. Literature Review

The thesis "Towards machine-learning assisted asset generation for games: a study on pixel art sprite sheets" by Y. R. Serpa and M. A. F. Rodrigues [18] discusses the challenges in game development related to asset creation, focusing on pixel art sprite sheets. It delves into deep-learning techniques using Generative Adversarial Networks (GANs) to generate pixel art sprites from sketches, aiming to reduce costs and streamline the asset-making pipeline for game development teams. The study showcases successful results in creating sprites resembling those made by artists, emphasizing the potential of deep-learning asset generation in the gaming industry. The paper highlights the importance of balancing technical and artistic aspects in game development. It addresses the increasing demand for high-quality visuals and efficient tools to support artists in realizing their creative visions.

"On the Challenges of Generating Pixel Art Character Sprites Using GANs" by F. Coutinho and L. Chaimowicz [19] discusses the challenges of generating pixel art character sprites using GANs, focusing on image-to-image translation tasks with Pix2Pix architecture modifications. It explores color palettes and histogram loss to enhance results, highlighting issues like overfitting and quality improvements. The paper emphasizes the significance of improving generated content quality for game asset creation, especially in pixel art, where constraints like limited color palettes and pixel information pose unique challenges compared to photorealistic image generation techniques. The research aims to advance Procedural Content Generation (PCG) techniques for efficiently creating artistic game assets.

The article “Pixel VQ-VAEs for Improved Pixel Art Representation” by A. Saravanan and M. Guzdial [20] discusses the Pixel Vector Quantized-Variational Autoencoders (VQ-VAEs) model tailored for pixel art representation, addressing the limitations of traditional models in capturing individual pixel importance. It highlights the significance of pixel art in popular media and proposes a specialized approach to enhance pixel art processing in machine learning models. By leveraging Variational Autoencoders (VAEs) for learning embeddings, the model demonstrates improved quality and performance on pixel art-related tasks, potentially impacting various domains involving pixel art. The work emphasizes exploring diverse art styles beyond photorealism in machine-learning applications.

The document “Gorgeous Pixel Artwork Generation with VQGAN-CLIP” by T. Yuan, X. Chen, and S. Wang [21] discusses the significance of deep learning in generating pixel artwork, focusing on VQGAN-CLIP as a powerful tool for creating images from text prompts. It highlights the impact of deep learning technologies in art, mentioning applications like Non-fungible tokens (NFTs) and virtual influencers. The collaboration between Vector-Quantized Generative Adversarial Networks (VQGANs) and Contrastive Language–Image Pre-training (CLIP) is explained, where VQGANs generate images based on text prompts while CLIP evaluates the correspondence between the image and text. The paper emphasizes the potential of these technologies in various fields beyond just machine learning, showcasing their cultural impact and practical applications.

The paper “Web-based Sprite Sheet Animator for Sharing Programmatically Usable Animation Metadata” by X. Lin [22] presents a prototype application for creating and sharing sprite sheets online, emphasizing programmatically usable raster graphics for 2D game programmers. It introduces the concept of a Meta-pixel Enhanced Sprite Sheet to ensure animation metadata is linked to Sprite sheet images. The document covers topics like 2D game graphics, animation metadata, and web application development to streamline obtaining 2D game animations. The significance lies in addressing the challenges faced by game developers in managing sprite animations efficiently and providing a solution through a web-based tool that simplifies this process.

The work "General Video Game Level Generation" by A. Khalifa, D. Perez-Liebana, S. M. Lucas, and J. Togelius [23] explores the challenge of creating a universal framework for generating video game levels across diverse games. Emphasizing the necessity of Procedural Content Generation (PCG) due to the varied preferences of players and the demand for high-quality content, the authors propose a general approach that leverages the General Video Game AI framework and the Video Game Description Language. This framework supports the creation of level generators capable of producing levels for multiple games without requiring game-specific engineering. It is presented as part of a new competition track to stimulate innovation in the field. The paper introduces three level-generating algorithms: Random, Constructive, and Search-based, with the Search-based generator showing the best results in a pilot study where human participants indicated a preference for its levels. This work aims to benefit small developers and the academic community by providing a benchmark for future research in generalized PCG.

When we look at the models used in related projects, we see that all employ GANs. Therefore, the results from these papers can only be partially applied, as we aim to utilize a diffusion model. Still, they provide valuable insights and anchor points for our work.

2.2. Existing Solutions

Two notable solutions have emerged in pixel art and retro-style game asset generation. Each leverages diffusion models to produce high-quality results. These are the open-source Pixel Art XL and the Retro Diffusion models. Both models offer unique features and capabilities that address the challenges of generating pixel art assets for video games.

2.2.1. Open-Source Pixel Art XL Model

The Pixel Art XL (PAXL) model, available on Civitai, is a specialized Low Rank Adaptation (LoRA) model built on the Stable Diffusion XL (SDXL) framework. It is designed to generate highly detailed pixel art images, seamlessly downscaling to classic pixel art resolutions without losing detail or coherence [1].

Key Features:

1. **Foundation on SDXL**: Utilizing the advanced capabilities of Stable Diffusion XL, the Pixel Art XL model inherits the flexibility and performance of SDXL, allowing it to handle various aspect ratios and deliver high-resolution outputs.

2. **Pixel Art Specialization**: Fine-tuned specifically for pixel art through LoRA, this model can produce images that match the nostalgic aesthetic of retro video games. The outputs maintain pixel-perfect detail, making them suitable for direct game use.

3. **High Customizability**: Users can input descriptive prompts to generate specific pixel art scenes, characters, or backgrounds. The model accommodates a range of styles within the pixel art domain, making it a versatile tool for game developers.

4. **Efficiency**: LoRA significantly reduces the computational cost of fine-tuning, enabling quicker adaptations and lower memory usage than full model retraining. This efficiency does not compromise the quality of generated assets.

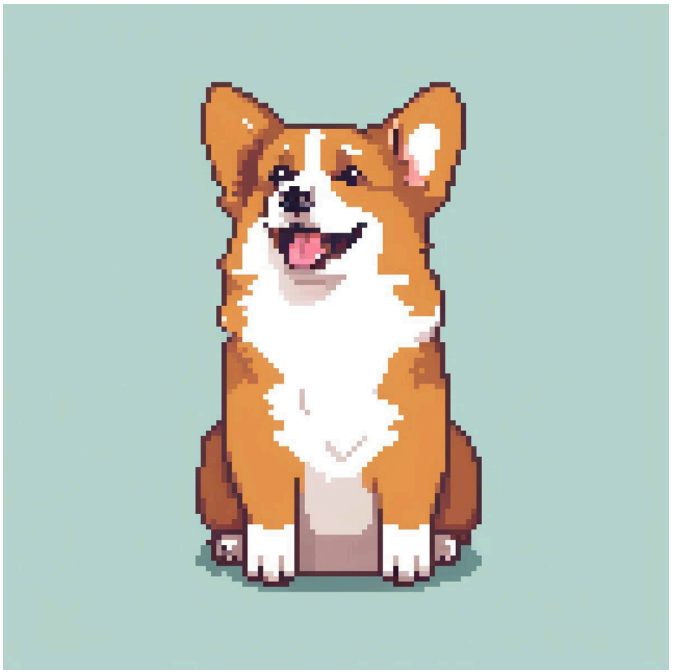

Figure 1 — Image generated using PAXL with the prompt: `a cute happy corgi` [1]

Applications:

- **Game Development**: Ideal for indie game developers looking to create classic pixel art assets without extensive artistic skills quickly.
- **Prototyping**: This is useful for rapid prototyping of game ideas, enabling developers to iterate quickly on visual concepts.
- **Educational Purposes**: This is a learning tool for those new to machine learning and game development.

2.2.2. Retro Diffusion Model

The Retro Diffusion model, available on Astropulse, takes a commercial approach to pixel art generation. It offers a diffusion-based model that targets the creation of retro-styled graphics [2].

Key Features:

1. **Diffusion-Based Generation**: Leveraging the power of diffusion models, Retro Diffusion ensures high-quality, noise-free outputs that capture the essence of classic video game art styles.

2. **Focused on Retro Aesthetics**: Tailored specifically for the retro aesthetic, the model produces assets that evoke the look and feel of 8-bit and 16-bit games, with attention to the limited color palettes and simple geometries characteristic of the era.

3. **Ease of Use**: Retro Diffusion is designed with user-friendliness. Even those with minimal machine learning expertise can generate visually appealing game assets. The purchase includes user-friendly tools and detailed documentation.

4. **Commercial Licensing**: Besides the one-time purchase model, Retro Diffusion includes licensing options suitable for commercial game development, ensuring that generated assets can be legally used in professional projects.

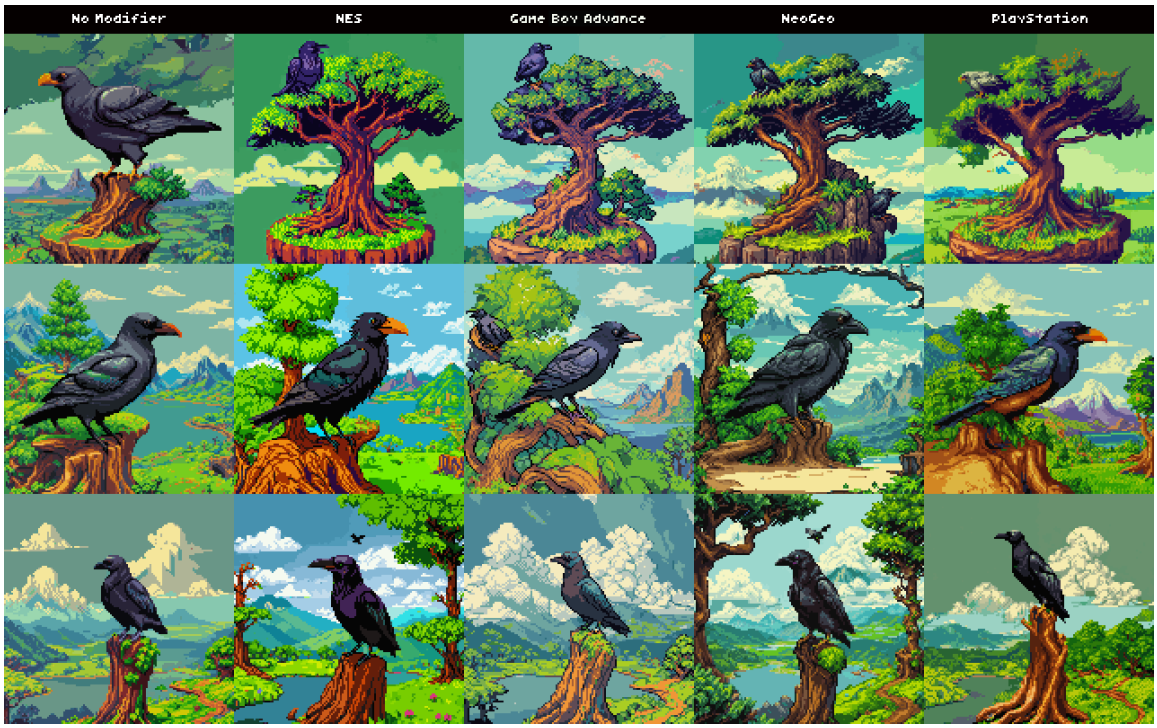

Figure 2 — Different styles that are possible with the Retro Diffusion model [2]

Applications:

- **Commercial Game Development**: This program is tailored for developers who want to release retro-style games commercially. It provides a legal framework for asset use.
- **Artistic Exploration**: Allows artists to explore and experiment with retro styles, potentially integrating modern twists or creating entirely new visual experiences.
- **Game Jams**: Perfect for game jams and hackathons, where quick asset creation can make or break the success of a project.

2.2.3. Comparison and Practical Implications

Both models represent significant advancements in the automation of pixel art creation, harnessing the power of diffusion models to streamline asset production. The open-source Pixel Art XL offers a versatile and efficient solution for developers looking for a highly customizable tool that is free of licensing costs but requires some technical knowledge to implement effectively. In contrast, Retro Diffusion provides a commercial, user-friendly package ideal for those seeking a straightforward, legally compliant solution with additional support and resources.

Ultimately, the choice between these models depends on specific project needs, budget constraints, and the desired level of customization and control. Both models pave the way for democratizing game asset creation, making high-quality pixel art accessible to a broader audience of developers and artists.

3. Theoretical Background

This chapter elaborates on **essential technologies and concepts** vital to understanding the work and our reasoning for the following development.

3.1. Machine Learning Models

Machine Learning (ML) is one of the many subfields of computer science. It can be explained by its capability to **extract patterns from raw data** to acquire knowledge about it. When computers are introduced to ML, they can work on problems in the real world and make their own decisions. cf. 1, 3 [24]

3.1.1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) belong to the general class of Artificial Neural Networks (ANNs) and are machine learning models. To better understand the architecture of ANNs, we use Figure 3.

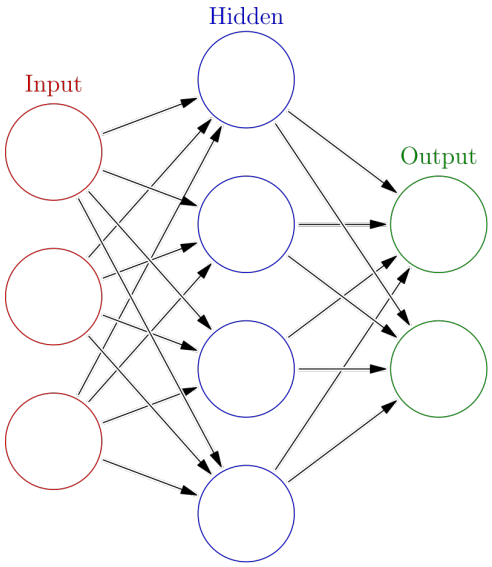

Figure 3 — Architecture of a Artificial Neural Networks (ANNs), [3] CC BY 3.0

As shown in Figure 3, the architecture of any ANN consists of **three major layers**, the first layer being the input layer. For instance, in Computer Vision, the input

layer can be all the pixels of an image to detect objects. An example is the Modified National Institute of Standards and Technology (MNIST) database [11]. It has about 70000 handwritten 28 by 28-pixel sized images of numbers to train an ANN to predict which number has been written. If the image has the dimensions 28 by 28, there would be 784 input neurons[1].

The next layer in the image is the hidden layer. This layer is particularly important because it consists of **as many hidden layers as required** for any ANN. In general, the more hidden layers it has, the more abstract it becomes and, therefore, has more trainable parameters.

Finally, the output layer is the end of the layer chain. It is where the data has the designated format and can finally be used for evaluation. In the case of the MNIST database [11], the output layer would consist of 10 nodes, one for each number from 0 to 9. Cf. p. 164-165 [24].

Now, onto CNNs, they are machine learning models explicitly used in **image pattern recognition**. The critical hidden layer architecture consists of three layers. The interaction of these three key layers can be viewed in Figure 4.

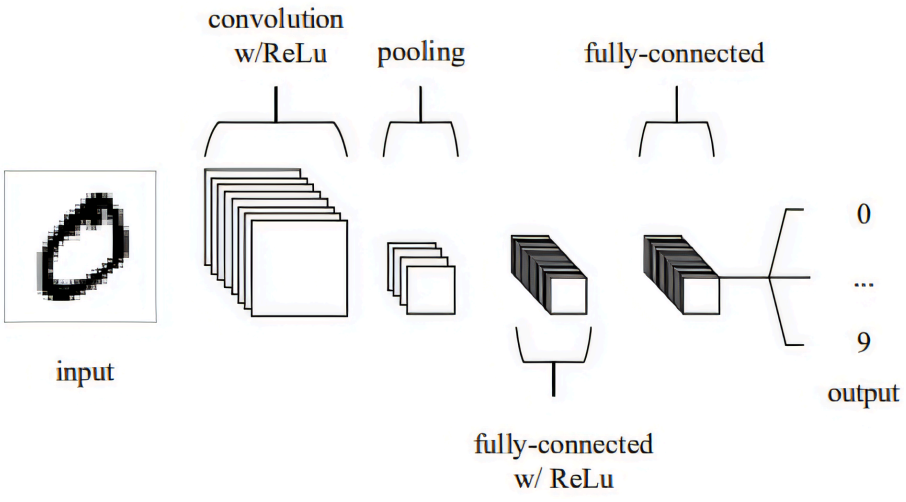

Figure 4 — Architecture of CNNs [4]

[1]In this case, the neuron is an Artificial Neuron (AN), used to sum the strength of the connections, also called weights, of the previous ANs up, and use a specific function called activation function to determine the own output cf. p. 195 [25].

As seen in Figure 4, the basic hidden layer architecture of a CNN is a **convolutional layer**, a **pooling** layer, and a **fully-connected** layer. A few of these layers utilize an activation function to achieve their results. The activation function is used within an AN, transforming a vector into a scalar. It uses the weights of the previous ANs, which are vectors, to call the function on a per element basis, to calculate the own AN's output, a scalar cf. p. 167, 170 [24]. To implement the activation function within an AN, the following pseudo-code algorithm shows the necessary computational steps:

Algorithm 1: Pseudo Code for an Activation Function in an Artificial Neuron

input: Weights vector w, Input vector x
output: Scalar output y
activation_function(w, x) $\rightarrow$ **y**

```
y ← 0
for i in 0 to n − 1 do
    y ← y + w_i c · x_i
end
y ← activation(y)
return y
```

The first one is the convolutional layer to return to the key hidden layers. It is a **crucial layer** for the CNNs because the layer **builds on** matrices, or in this case, **kernels** as learnable parameters. A kernel is used within the mathematical technique called convolution. This technique can be explained using Figure 5.

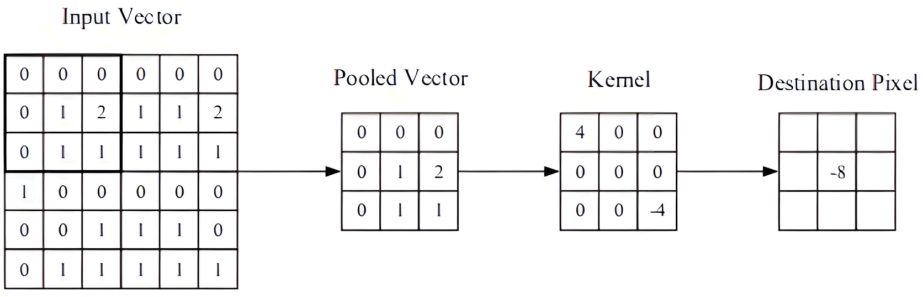

Figure 5 — A visual representation of convolution p. 6 [4]

The input vector can be seen as an image, from this image a pooled vector is going to be extracted which has the same size as the kernel. This pooled vector is extracted by placing the kernel on the input vector, using these matrices a **scalar** can be calculated and **placed** on the new image. To continue the convolution in the layer, the kernel would now be put one space to the right, and another value would be calculated. The kernel has the **advantage of being learnable**, meaning that its values are optimized during the training phase to detect specific features. In the case of the MNIST database [11], a feature of a 5 could be an edge. Thus, the convolutional layer is crucial because it enables a CNN to automatically learn the most relevant features for the task cf. p. 2, 4-6 [4]. This layer utilizes the activation function called ReLU with the equation $f(x) = \max\{0, x\}$ and Figure 6 to output a so-called feature map that represents each detected feature of an image.

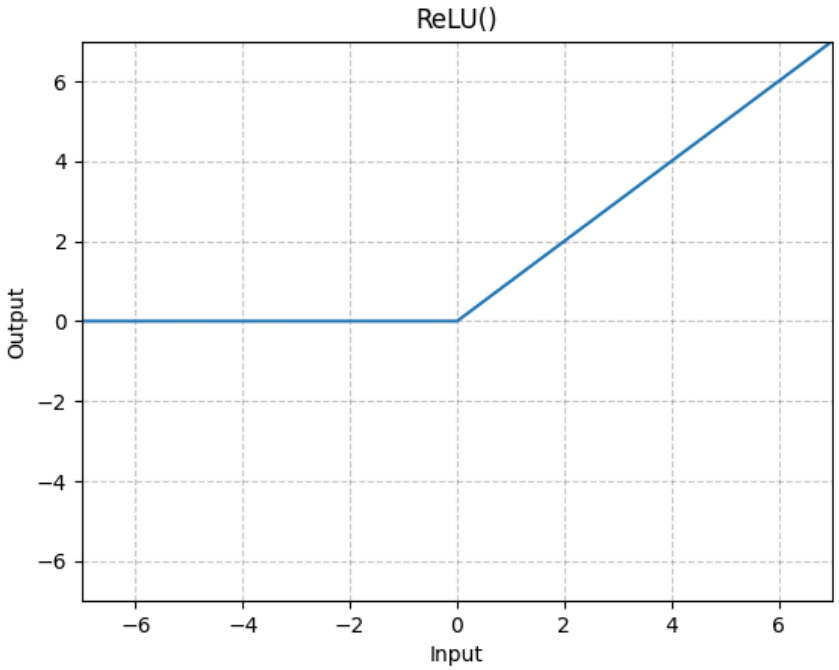

Figure 6 — A visual representation of the ReLU activation function [5]

It returns the **maximum between 0 and the calculated value**. Thus, negative values will return 0 after going through the ReLU function. Since the activation function remains adjacent to a linear function, it maintains multiple properties of

linear models, which are easy to optimize using different methods, for instance, gradient-based methods. cf. p. 167, 170 [24]

The next layer is the pooling layer. Its task is to **perform downsampling** without complex computations along the width and height of the images without touching the color channels. Using the pooling layer, the amount of data that needs to be processed is further decreased. cf. p. 5 [4]

This dimensionality reduction is achieved using the pooling layer to operate over each feature map from the input to scale its dimensionality using the “max” function. In this case, this is typically done by 2 by 2 kernels. They are applied with a stride of 2 along the width and height of the feature map, further downscaling it by 25%. cf. p. 8 [4]

The operation can be visually explained by the following image:

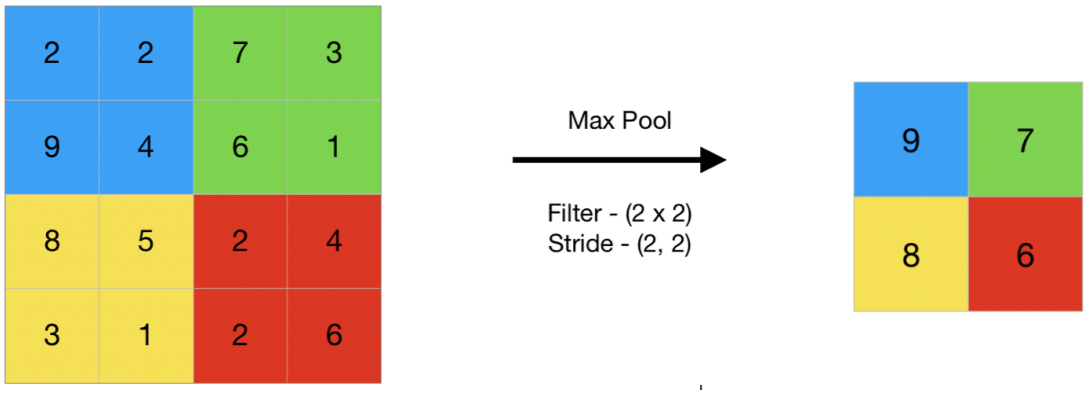

Figure 7 — A visual representation of the max-pooling layer [6]

Using the above image, the pooling layer utilizes a dimensionality reduction by selecting the maximum value from each 2x2 field of the input matrix (left), using a stride of 2. This way, the 4x4 matrix is reduced to a 2x2 matrix. So, the **output** of the pooling layer is a **downscaled version of a feature map**, with the width and height reduced. This reduction helps decrease the number of parameters and computations in the CNN since the important features are still preserved. cf. p. 8 [4]

The last key layer, the fully-connected layer, is **fully connected with all the ANs from the previous layer** and the next layer in the layer chain and will attempt to calculate class scores from the activations used for the classification task. If we think back to the MNIST database [11] example from earlier, the output layer has 10 ANs, to know which number of the 10 available is being drawn in the input, the fully-connected layer tries to have a class score for each of the tens classes. In the case of the earlier shown Figure 4, the AN for class 0 should have the highest class score. It is important to note that the fully-connected layer does not have connections between his ANs. cf. p. 5, 8 [4].

In contrast to the convolutional layer, the fully-connected layer works by applying the dot product (weighted sum) of each connection between the ANs. Mathematically, the layer **calculates the weighted sum of all input activations** (feature maps) x, AN weights w, and the bias b which each AN has. This can be mathematically expressed as the following cf. [26]:

$$y_j = \sum_i (w_{ij} * x_i) + b_j$$

Overall, CNNs transform input data layer by layer using convolutional and pooling operations to produce high-level features used by fully-connected layers for final classification, making them powerful tools for **image recognition tasks**.

3.1.2. Diffusion Models

DMs are probabilistic models **designed to learn a data distribution by gradual denoising** of a normally distributed variable. The key idea behind it is to slowly and systematically destroy the structure in a data distribution, for instance, image data, using an iterative forward diffusion process. This process must be iterative, since then a reverse diffusion process, which restores the structure in data, can be learned. This way, the problem of modeling complex data sets can be eased, considering that the DM can now learn to **generate high-quality data by reversing the diffusion steps**, allowing for more accurate and efficient generative modeling. Using the below example, the diffusion process can be visually represented, with

t being the current iteration time step and T being the final time step cf. p. 1, 3, 4 [7].

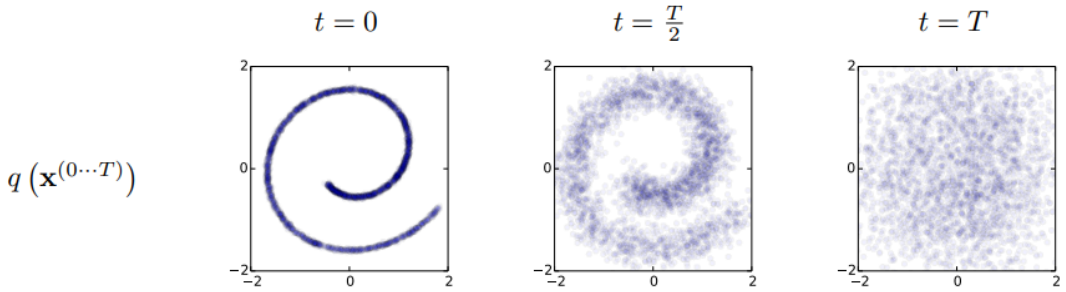

Figure 8 — A visual example of DMs p. 3 [7]

Regarding image generation, DMs can be utilized by injecting or ejecting noise. In this case, the noise has a Gaussian kernel cf. p. 5 [9]. The Gaussian kernel can be visualized using Figure 9.

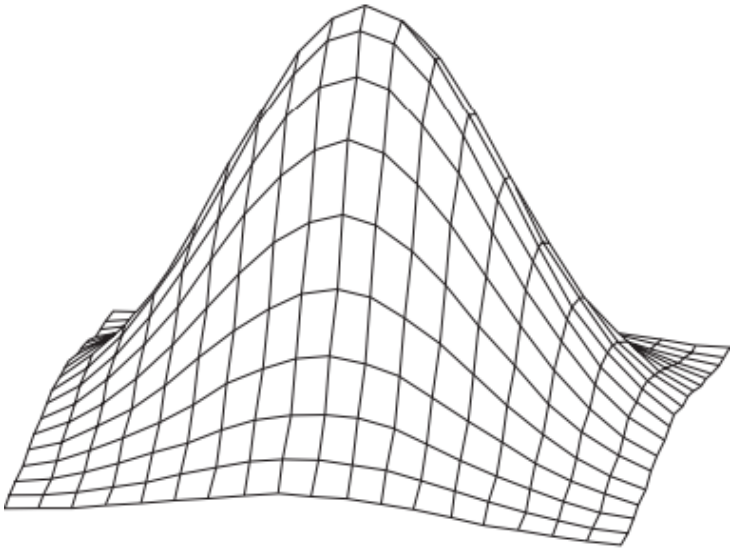

Figure 9 — A visual example of the Gaussian kernel p. 86 [8]

As explained earlier, the noise from Figure 9 can be inserted iteratively to train a model to predict a denoised input variant. This data injection is made possible by the so called Forward Stochastic Differential Equation (SDE), as seen in Figure 10, this function utilizes the Gaussian noise to be injected iteratively starting by the point $x(0)$, by making the **deterministic and stochastic parts of the image independent**. The deterministic part, which is, for example, the randomness while

taking a picture, is $f(x, t)$ in the image, with t being the time step for the strength of the Gaussian noise, and $x(t)$ representing the state of the data at time t. The stochastic part, the Gaussian noise, is captured by $g(t)$, which determines how much noise is introduced at each step, thus controlling the diffusion process. The closer t is to T, the more unrecognizable the image gets.

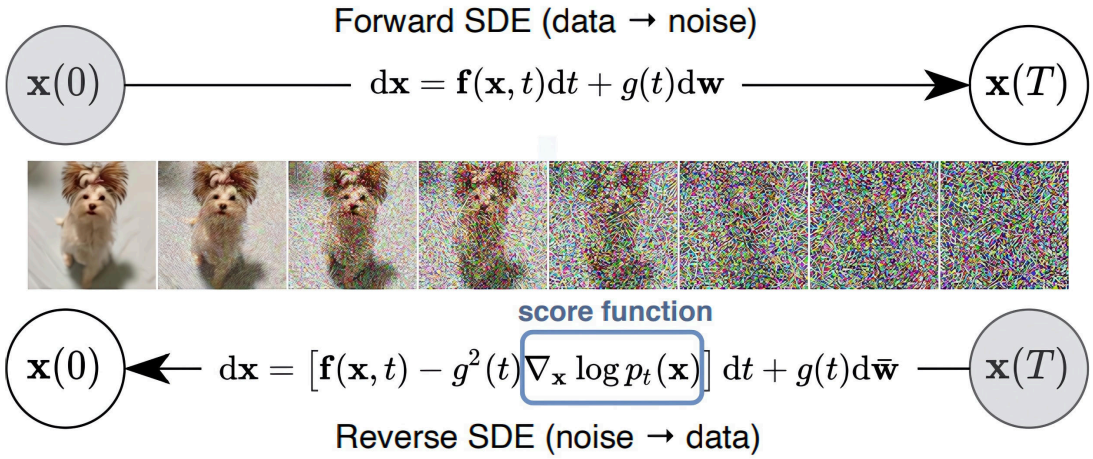

Figure 10 — Forward and Reverse SDE [9]

On the other hand, the Reverse SDE aims to reconstruct the original data from noisy input. It essentially tries to reverse the Forward SDE process, as seen in Figure 10. It starts with $x(T)$ by adding new parts to the equation. The score function[2]: $\nabla_x \log(p_t(x))$ helps to guide the process by indicating how to adjust the data at each step to restore the structure that was lost during the Forward SDE. And the so-called Brownian motion $d\tilde{w}$ represents the stochastic influence. Using these new components, the Reverse SDE **gradually removes the noise**, changing the x value step by step and reconstructing the image to its original state. cf. p. 1-2, 3-4 [9]. So, these models are trained to predict a denoised variant of the input cf. p. 4 [27].

3.1.3. Variational Autoencoders

Variational Autoencoders (VAEs), which have been developed by D. P. Kingma and M. Welling [28] in 2013, are neural network architectures designed for generative tasks. They aim to **learn a representation of the input data while also being**

[2]Mathematically the gradient (∇) of the $\log$ probability density function at the specific time t

capable of generating new data, which is similar to the training data cf. p. 1, 5, 8 [28].

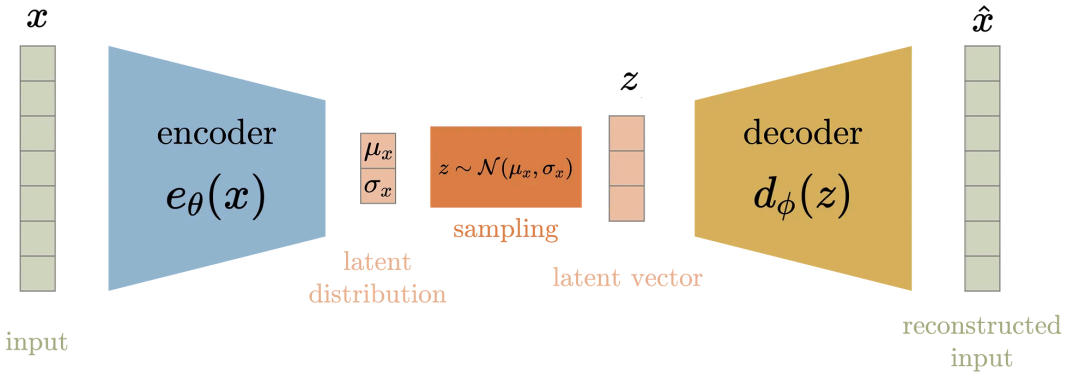

Figure 11 — Architecture of a VAE [10]

The architecture of VAEs consists of elements from recognition and generative models. It has a probabilistic encoder and decoder. The encoder, also called a probabilistic encoder, uses the input data to generate a **latent representation** of the input data to capture essential features.

This encoding works by **using the input data x and encoding it into a latent representation** z. This process involves producing a specific distribution over the possible values of z, typically a Gaussian distribution as earlier seen in Figure 9. To make this process more tangible, for instance, x can represent an image of a handwritten digit, just like from the MNIST database [11], the encoders representation of x could be a distribution in the latent space where certain regions correspond to specific features, such as the **sharp edges from the number five**. So, by encoding the input data into the latent space, the encoder captures the essential features and variations within the data by abstracting the image, thus allowing the model to learn a compact and meaningful representation of the input data. cf. p. 3 [28].

The latent representation can also be viewed as a **vector space of a reduced dimensionality** whose size is relative to the input size. This representation is then used to create more general features to help characterize the input. It is important to use latent representation in two main aspects. At first, it can provide deep insights into the data while revealing hidden relationships. Second, they can serve

as feature spaces for machine learning applications because of their lower dimensional size cf. p. 1,2 [29]. Another thing to note is how the hidden relationships get revealed. This is due to the technique of encoding similar features close to each other. So, in this context, similarities in the vector space mean that data points with **similar characteristics are represented by vectors close to each other**.

Figure 12 — An example of the different writing styles in the MNIST database [11]

This concept can be explained using the following Figure 12 from the MNIST database [11], images of handwritten digits that look similar, for example, multiple images of the number five written in a slightly different way **will be mapped to points close to each other in the latent space**. This shows that the model has learned to recognize essential features of the number five while ignoring the writing style of the number. cf. p. 6-7 [28].

After creating a latent representation smaller than the input, the decoder uses the encoder's representation to reproduce the input data's potential corresponding values to reconstruct the original data cf. p. 3 [28].

The training objective of a VAE is to **minimize the Kullback-Leibler divergence[3] and the reconstruction error[4]** at the same time.

3.1.4. U-Nets

U-Nets are an extended variant of the CNN architecture, which reduces the number of training images required to **produce more precise image segmentations**. This is achieved through the following architecture:

[3] How close the latent variables are following a probability distribution
[4] How different the decoded example is from the input

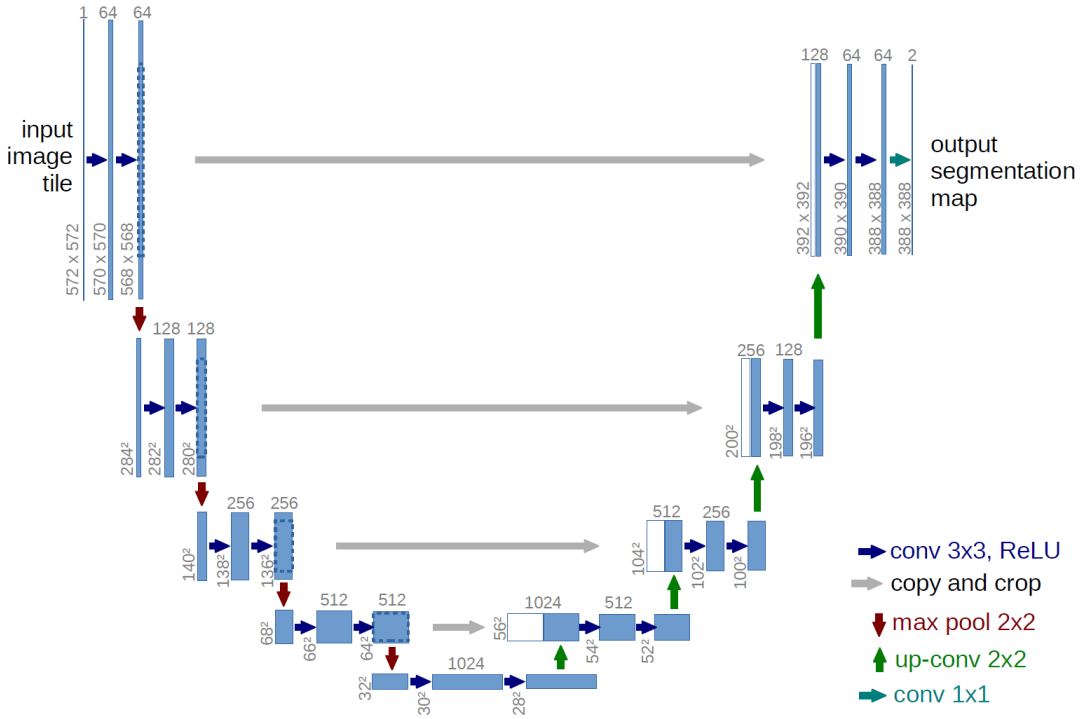

Figure 13 — Architecture of a U-Net [12]

A CNN architecture can be recognized in the architecture of a U-Net. The architecture can be split into two major parts: the **contracting part (left side) and the expansive part (right side)**. The CNN architecture can be recognized in the contracting part. Its architecture has already been explained in Section 3.1.1, though it is important to note that the number of feature maps doubles at every step.

In the expansive part, in each step, the **feature map (blue boxes) will be upsampled and then convoluted** by a 2x2 convolution (green arrow). This specific action halves the number of available feature maps. After this, at each level of the architecture, a concatenation with the corresponding feature map on the opposite side is carried out because otherwise, the edge pixels would be lost. As in the contracting part, there are two 3x3 convolutional layers, each followed by a ReLU activation function. A 1x1 convolution is required to map each feature vector to the number of available classes.

Because the architecture resembles a contracting and expansive part, the network is also called a fully convolutional network. cf. p. 4 [30].

This architecture helps **reduce training images through extensive data augmentation**. By applying deformations to training images, the U-Net can learn the invariance of such deformations without needing many annotated images. In this case, the invariance could mean a distortion in the image. This method allows the network to be trained effectively with fewer images cf. p. 3, 6-7 [31].

It also helps with more precise image segmentations. This is crucial because the high-resolution features from the contracting part are combined with the upsampled features from the expansive part. This concatenation **preserves the image's width and height information**, also called spatial information, which is required for localization tasks. These localization tasks are essential for accurately outlining structures within the images cf. p. 3 [31].

To make predictions, the convolutional layer can learn to assemble output using the combined high-resolution and upsampled features from both parts. The **output will be refined with consecutive convolutional layers** at each step, leading to more precise segmented outputs cf. pp. 2-3 [31].

The U-Net Architecture has also been used in DMs to denoise images iteratively. cf. p. 2 [30].

3.2. Image Generation with Stable Diffusion

This section explains the advancements in image synthesis through Stable Diffusion (SD) and its extensions. It is divided into three sections, each focusing on a specific aspect of these advancements.

Section 3.2.1 will provide a look at the original SD model using the earlier explained key components. These components are the mandatory VAE, the U-Net backbone, and the optional text encoder for image generation conditioning. This section will give some insights into how high-quality images are generated.

In Section 3.2.2, an enhanced version of the original model called Stable Diffusion XL (SDXL) is covered. It features significant improvements to enable multiple aspect ratios and superior performance.

Lastly, in Section 3.2.3, the fine-tuning technique LoRA will be explained, which outstandingly reduces the computational requirements for training models. Additionally, a discussion of the application of LoRA in creating specialized models like Pixel Art XL (PAXL) is featured.

These chapters will provide a comprehensive understanding of the evolution and capabilities of SD and its variants.

3.2.1. Stable Diffusion

The CompVis group at LMU Munich developed Stable Diffusion (SD) to reduce the computational demands of training diffusion models for high-resolution image synthesis. It uses a Latent Diffusion Model (LDM), which consists of three parts: a **VAE**, a **U-Net**, and an **optional text encoder**. Cf. p. 3-5 [27].

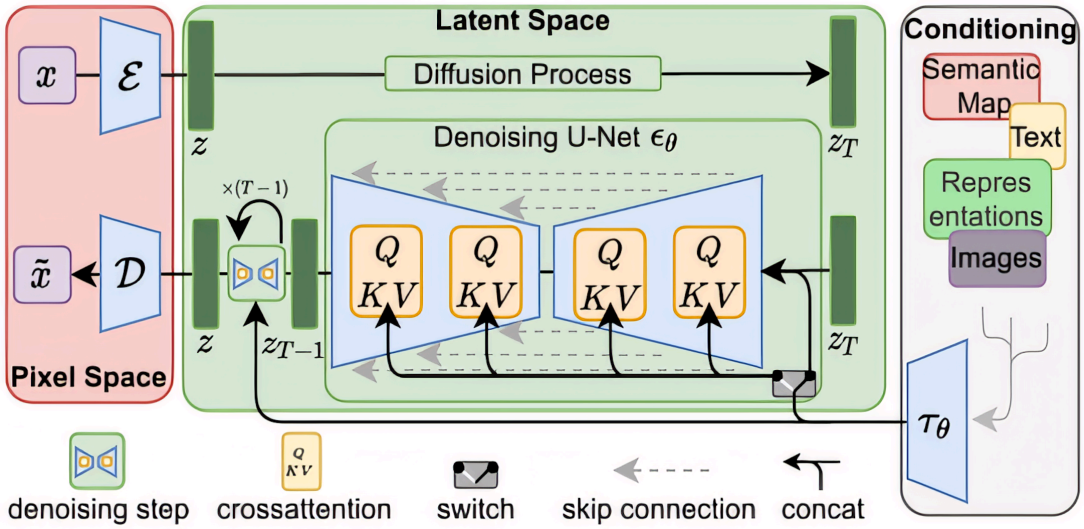

Figure 14 — Architecture of Stable Diffusion [13]

The core motivation behind using LDMs is to overcome the limitations of operating directly in high-dimensional pixel space, which requires immense computational resources. By shifting the diffusion process into a latent space, SD leverages the advantages of lower-dimensional data representation, enabling efficient training and inference without compromising visual fidelity.

The first component of SD is the VAE, which **encodes the input image into a compact latent representation**. This latent space is critical because it captures the

essential features of the image while discarding high-frequency noise that is not perceptible to humans. The VAE consists of an encoder and a decoder, where the encoder compresses the image into the latent space, and the decoder reconstructs it back to the image. This process is guided by pixel space and perceptual losses, ensuring the generated images stay realistic and sharp [27].

The VAE's training process is structured to balance compression and detail preservation. Unlike traditional methods that rely heavily on pixel-wise loss calculations, SD's **VAE incorporates perceptual losses that focus on what humans perceive**. This reduces the computational load while maintaining high-quality reconstruction capabilities [27].

Once the image is transformed into the latent space by the VAE, the diffusion model operates to generate new images. The central part of SD is the U-Net, which forms the backbone of the latent diffusion model. The U-Net architecture is particularly effective due to its convolutional nature, which is well-suited to handling the spatial dependencies inherent in image data. The model operates by **iterative denoising the latent space representation**, gradually refining the image from a noisy version to a clear, high-quality output. The U-Net structure allows for efficient training and high-quality image synthesis by leveraging its downsampling and upsampling pathways to capture and reconstruct finer details at multiple scales [27].

The training objective for the U-Net in the latent diffusion model is to **learn the reverse diffusion process**. Given a noisy latent representation, the U-Net is trained to predict the noise added in the forward process, enabling it to refine a noisy image into a high-quality output iteratively. The loss function typically used is the mean squared error (MSE) between the predicted and actual noise components [27].

A significant extension in Stable Diffusion includes an optional text encoder, which conditions image generation on textual descriptions. The text encoder processes textual descriptions into a form that can influence image generation. This allows text-to-image (TTI) capabilities, where the model generates images that align with

a given textual prompt. The **text is encoded into embeddings** integrated at various points within the U-Net. This mechanism ensures that the generated images follow the textual description closely, enabling controlled and precise image synthesis [27].

The main advantage of the latent diffusion model approach in SD is its computational efficiency. By handling the generative process in the latent space, **SD significantly reduces the number of parameters and operations** required compared to pixel-based methods. This makes high-resolution image synthesis feasible even on limited hardware.

Stable Diffusion combines the powerful features of VAEs and U-Nets within a latent space framework to create a highly efficient and scalable solution for high-resolution image synthesis. The incorporation of text conditioning further enhances its capabilities, making SD a versatile tool for various image-generation tasks.

3.2.2. Stable Diffusion XL

Stable Diffusion XL (SDXL) represents a significant milestone in the evolution of latent diffusion models for high-resolution image synthesis. Introduced in 2023 by D. Podell *et al.* [14], SDXL is designed to address several limitations observed in earlier versions of SD by enhancing the model's architectural complexity and training methodologies.

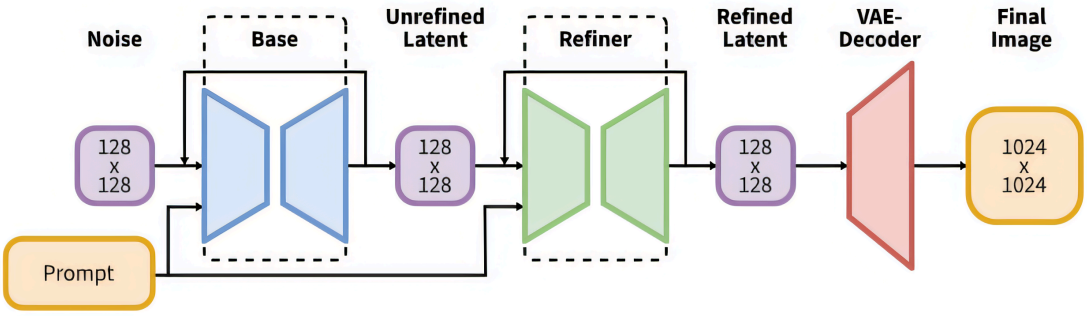

Figure 15 — Architecture of SDXL [14]

SDXL utilizes a **U-Net backbone three times larger than its predecessors**. This expansion is primarily attributed to an increase in the number of attention blocks

and the incorporation of a more extensive cross-attention context. The large backbone enhances the model's ability to capture intricate details and improves the fidelity of generated images [14].

Unlike earlier models, which typically used a single text encoder, SDXL integrates **two text encoders** (Figure 15). This dual encoder setup allows for richer and more contextually aware textual conditioning, enabling better alignment between the generated images and their textual descriptions [14].

SDXL introduces innovative conditioning techniques that do not require additional supervision. These include conditioning the model on the original image resolution and size, which helps maintain the fidelity of various image features across different scales. During training, the original image height and width are provided as an additional conditioning input to the model, which is embedded using Fourier feature encoding, aiding in the retention of resolution-dependent features during inference [14].

SDXL employs multi-aspect training to tackle the issue of varying image aspect ratios. The model is **trained on images of a fixed aspect ratio and then finetuned on multiple aspect ratios**. This is accomplished by partitioning the dataset into buckets of different aspect ratios and training the model on these varied datasets, which enhances its capability to generate images in diverse aspect ratios without compromising quality [14].

A **separate refinement model** significantly adds to SDXL (Figure 15). This model uses a noising-denoising approach to improve the visual fidelity of samples produced by the primary SDXL model. During inference, this refinement model further processes initial latent samples generated by SDXL, enhancing the details and overall quality of the images. This step is useful for refining intricate details such as human faces and complex backgrounds [14].

Extensive user studies have shown that SDXL significantly outperforms earlier versions of Stable Diffusion. Users consistently preferred images generated by SDXL across various prompts and scenarios. The model also demonstrates superior per-

formance in generating higher-resolution images, making it competitive with other state-of-the-art image generators such as DALL-E and MidJourney [14].

SDXL includes 2.6 billion parameters in the U-Net, significantly higher than the 865 million parameters in Stable Diffusion. This parameter increase is primarily due to the added attention layers and the dual text encoder setup. The SDXL training dataset encompasses various aspect ratios and resolutions, ensuring the model can generalize well across different images and prompts [14].

In line with the principles of open research, the SDXL model's **code and weights have been made publicly available** on platforms such as GitHub and Hugging Face. This transparency promotes further innovation and allows researchers and developers to build on SDXL's advancements.

The introduction of SDXL marks a significant advancement in latent diffusion models. Its larger architecture, innovative conditioning techniques, and the inclusion of a refinement model collectively contribute to its superior performance. By addressing the limitations of earlier models and incorporating cutting-edge improvements, SDXL sets a new benchmark for high-resolution image synthesis, making it a versatile tool for applications ranging from artistic image generation to photorealistic visualizations.

3.2.3. LoRA

Low Rank Adaptation (LoRA) is a fine-tuning technique introduced to optimize the adaptation of large pre-trained language models to specific downstream tasks. The primary driving force behind LoRA is **addressing the computational and storage challenges of fine-tuning large models**. Traditional full fine-tuning methods, such as training all model parameters, become infeasible for such large models due to the massive resource requirements [15].

LoRA innovatively tackles these challenges by **freezing the pre-trained model weights** and **injecting trainable low-rank decomposition matrices** into the dense layers of the model. This method significantly reduces the trainable parameters and the computational resources needed for fine-tuning [15].

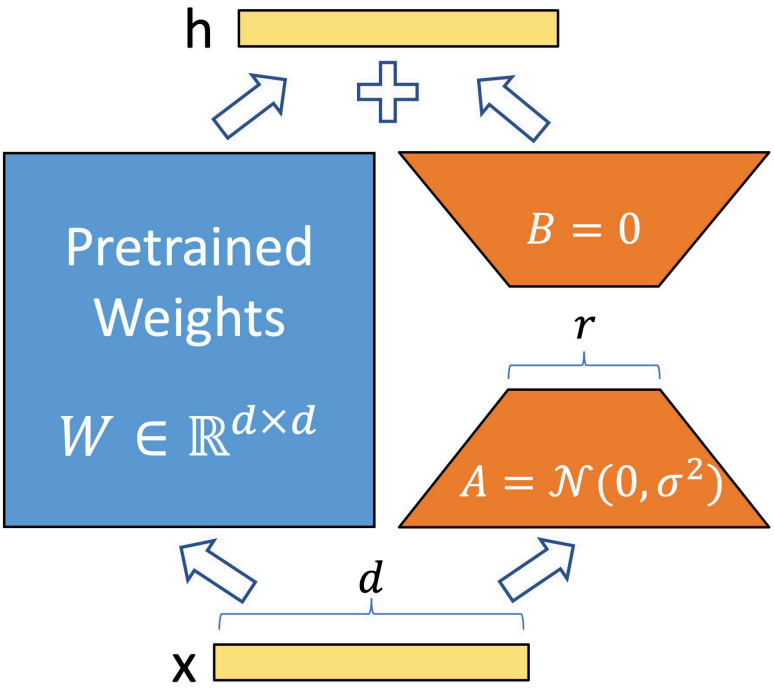

Figure 16 — LoRA output h produced with matrices A, B, and the pretrained weight matrix W from input x [15]

The weights of the pre-trained model are kept static during the fine-tuning process, so the original weights are not updated. Instead of updating the full weight matrix, LoRA introduces two smaller matrices, A and B, so their product BA approximates the necessary weight change ΔW (Figure 16). The rank r of these matrices is much smaller than the original, making the adaptation process more efficient. A transformer layer's original weight matrix W is modified as $W_0 + \Delta W$, where $\Delta W = BA$. Here, A and B are initialized with random Gaussian distribution and zeros, respectively, and are the ones being trained. Compared to full fine-tuning, **LoRA reduces the trainable parameters by around 10,000 times** and the **GPU memory requirement by about three times** [15].

To implement LoRA, the following steps are typically followed [15]:

1. **Initialization:**
   - Pre-trained weights W_0 are kept intact.
   - Low-rank matrices A and B are initialized. A is usually initialized with small random values, while B is initialized with zeros.

2. **Training:**
   - During the forward pass, the output of a dense layer is modified by adding BAx to the original output W_0x.
   - Only matrices A and B are updated during the backpropagation, significantly reducing the computational load.
3. **Deployment:**
   - The modified weight matrix $W = W_0 + BA$ is used at inference time, ensuring no additional inference latency compared to a fully fine-tuned model.
   - LoRA allows easy swapping of the task-specific components A and B, making it highly efficient for models deployed as services.

An example of a LoRA is PAXL, it is used with SDXL version 1.0. It was released on August 7, 2023, by the user NeriJS [1]. It can generate pixel art-like images with a resolution of 1024x1024. Downscaling the image to the resolution 128x128 fits pixel-perfect[5]. So, in this way, PAXL can be used to generate pixel-perfect images using the knowledge and advantages of SDXL without training [1].

3.3. Advances and Techniques in Video Game Graphics

Video game graphics have evolved significantly over the past few decades, migrating from simple 2D sprite-based visuals to highly detailed 3D environments. The desire to create more immersive and realistic gaming experiences drives continuous improvement in graphics technology. This chapter discusses various techniques and advancements in video game graphics, focusing on methods such as texture sprites, automatic sprite generation, image translation in-game character sprite drawing, and tiles in-game graphics while referencing key studies and technologies that have contributed to these developments.

3.3.1. Texture Sprites and Their Applications

Texture sprites are small texture elements mapped onto surfaces to create complex textures. S. Lefebvre, S. Hornus, and F. Neyret [16] introduced a method to texture complex geometries at very high resolutions using texture sprites that rely

[5]Meaning one pixel corresponds to one pixel in the image

on small texture elements splatted onto the surface. This approach supports real-time rendering and allows for interactive editing and animation of textures.

Texture sprites can be blended to produce sophisticated surface appearances. These sprites are described by attributes such as position, size, and texture ID, which are dynamically updated, enabling seamless and intuitive texturing. The sprites are stored compactly in Graphical Processing Unit (GPU) memory, reducing the overall memory cost while achieving high texturing resolution [16].

The effectiveness of texture sprites can be seen in various applications, including:

- **Animated Sprites**: Allowing for dynamic and responsive surface textures that can adapt to changes in geometry or user interactions.
- **Blobby Painting and Voronoi Blending**: Techniques that facilitate unique visual effects by blending multiple sprites to create fluid or cellular patterns.

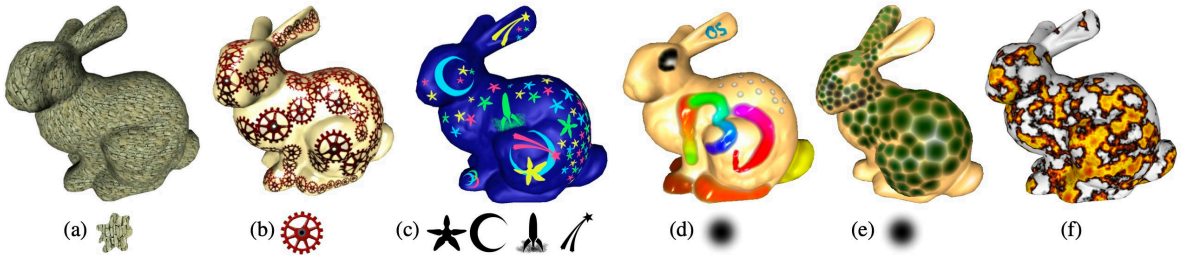

Figure 17 — Different sprite rendering techniques applied to a 3D object [16]

3.3.2. Automatic Sprite Generation Using Deep Learning

Deep Convolutional Generative Adversarial Networks (DCGANs) have shown promising results in automatically generating game sprites. L. Horsley and D. Perez-Liebana [17] demonstrated that DCGANs could generate human-like characters, faces, and creatures for games, even with relatively small datasets.

The DCGAN framework consists of two neural networks, a generator and a discriminator, which are trained simultaneously. The generator creates images based on random noise inputs, while the discriminator evaluates the authenticity of these images. The generator learns to produce increasingly realistic sprites [17] through this adversarial process.

L. Horsley and D. Perez-Liebana [17] showed that DCGANs could generate high-quality sprites with diverse characteristics from limited training data. This method can significantly streamline the sprite creation process in game development, reducing the need for manual artwork and speeding up production timelines.

3.3.3. Image Translation for Game Character Sprite Drawing

J.-I. Choi, S.-K. Kim, and S.-J. Kang [32] introduced an image translation method using GANs for game character sprite drawing, which automates and simplifies the creation of 2D character animations.

J.-I. Choi, S.-K. Kim, and S.-J. Kang [32] proposed a methodology that uses a mixed input of a small number of segmentations and skeletal bone paintings to generate sprite images. This method allows users to produce sprites by selecting a few colors and drawing simple bone structures, which the system then translates into fully textured character animations.

The GAN-based image-to-image translation network generates high-resolution sprite images from user input by utilizing skeletal loss to ensure the consistency of body structures and poses. This improves the overall quality and accuracy of the generated sprites, making the process less labor-intensive [32].

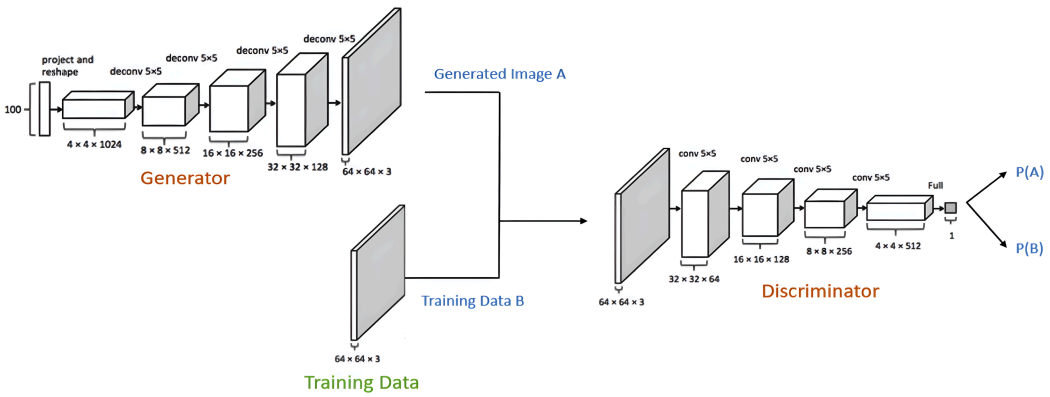

Figure 18 — DCGAN training workflow [17]

3.3.4. Tile-Based Modeling and Rendering

Tiles or small pre-rendered images, often used repetitively to create a larger image, have been a staple in video game graphics since the early days of digital gaming. Tiles allow developers to create large, detailed environments efficiently [33], [34].

Tile-based rendering involves dividing the game world into smaller, manageable pieces or "tiles," which can be reused and combined to create complex scenes. This method not only conserves memory but also simplifies level design. M. Terai, J. Fujiki, R. Tsuruno, and K. Tomimatsu [33] state that tile-based techniques can efficiently convert 2D overhead views into 3D models, allowing seamless transitions between 2D and 3D representations in games. M. Terai, J. Fujiki, R. Tsuruno, and K. Tomimatsu [33] demonstrated a system where users could create 3D geometry from 2D tiles on a canvas. This system used the height and depth information encoded in the tiles to generate 3D shapes that could be viewed from arbitrary angles, facilitating intuitive and versatile game design.

W. Burgers [34] explored the benefits of tile-based rendering in reducing off-chip memory traffic in 3D graphics pipelines, particularly for mobile devices. Tile-based rendering can enhance performance and reduce power consumption by independently rendering smaller sections of a scene (tiles) and minimizing the need to access off-chip memory.

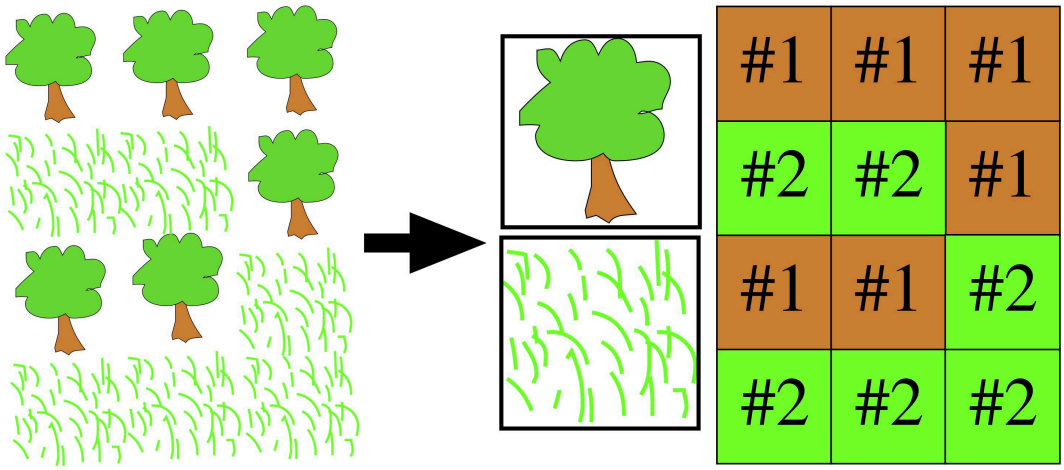

Figure 19 — Tile-based rendering [16]

Key Benefits of Tile-Based Rendering:

- **Memory Efficiency**: By reusing tiles, developers can save significant amounts of memory compared to using unique textures for each part of the game world.
- **Ease of Level Design**: Level designers can rapidly create varied environments by mixing and matching different tile sets.
- **Optimized Performance**: Tile-based rendering reduces off-chip memory accesses, which is crucial for performance, especially in devices with limited memory bandwidth.

Figure 20 — Example pixel-style sprites [17]

3.3.5. Broader Applications and Future Directions

The techniques discussed in this chapter advance the field of game graphics and have broader applications in other digital media domains. They can be used in animation studios, virtual reality environments, and AI-driven art creation.

Integrating PCG in gaming highlights the potential of automated systems to create endless variations of game levels and environments. These advancements complement graphical improvements by providing variety and depth to game worlds [23].

Future research should focus on enhancing the efficiency and capabilities of these graphical techniques. For instance, improving the neural network architectures used in sprite generation to handle more complex animations and developing more sophisticated algorithms for real-time texture manipulation will further elevate the visual quality of games.

Advances in video game graphics, such as texture sprites, deep learning-based sprite generation, image-to-image translation, and tile-based rendering, have revolutionized how visuals are created and rendered in games. The studies and methodologies discussed in this chapter demonstrate the potential of these techniques to enhance the aesthetic and immersive qualities of video games. As technology evolves, so will the tools and methods for creating compelling and dynamic game graphics.

4. Methodology

This chapter will describe the methodology used to create the project. First, we will provide an overview of the general concept used for the generation in Section 4.1. Thereupon, Section 4.2 elaborates on our pipeline approach and its components. Subsequently, the first part of the general concept will be explained in detail in Section 4.4. Ultimately, the second part of the general concept will be enlightened in Section 4.5.

4.1. General Concept

The project's general concept has two primary stages. The first stage is **image generation**. A prompt can generate a video game character or background in this stage. This prompt will be used to create an image with Stable Diffusion XL (SDXL) and the Low Rank Adaptation (LoRA) Pixel Art XL (PAXL). If the image to be generated is supposed to be a character, this character will be cropped so as not to distort the image. Afterward, the image will be downscaled to the specific requirement size.

In the second stage, we aim to achieve **animated characters**. This will be accomplished by applying the previously described workflow to generate two images with the same seed but a slightly different prompt.

4.2. Pipeline Architecture

This section addresses why we chose the pipeline approach over other approaches, such as training a new model. Furthermore, we explain the components of the pipeline in detail.

4.2.1. Approach

For this project, we chose a pipeline approach for several reasons. First, creating a pipeline takes far less time than training a diffusion model. Additionally, achieving consistent results with a newly trained model is challenging. We need a large

data set with pixel sprites and tiles to train the model to accomplish consistent results. Moreover, the created pipeline brings more flexibility to extend the workflow or change desired parts. Therefore, creating a pipeline allows easier integration into existing projects to speed up development. Lastly, training a new model limits creativity based on the trained data. Thus, an even larger data set is required to accomplish high creativity.

4.2.2. Components of the pipeline

Our pipeline uses the SDXL model with PAXL applied for fine-tuning. As described in Section 3.2.2, SDXL is an advanced version of the SD model, providing better performance across various aspect ratios and more detailed results. This is crucial to creating high-quality game assets.

The PAXL is utilized to fine-tune the SDXL model and achieve pixel-style outputs. It employs the LoRA technique (Section 3.2.3) to apply weights to the SDXL model. Thus, it reduces the required parameters and lowers the used GPU memory.

4.3. Generating backgrounds

The most straightforward workflow our project incorporates is background generation. To generate a background, we solely run it and apply minimal post-processing. The following section will give you an overview of how backgrounds are created.

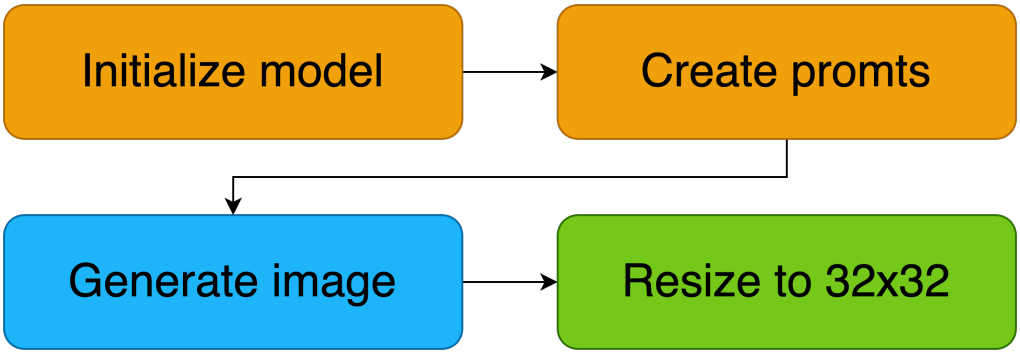

Figure 21 — Pipeline used for background generation

4.3.1. Initialization of a diffusion pipeline

The first step in generating an image is to create a diffusion pipeline to run specific models. This pipeline is created by using a Python library and loading the pretrained versions of both SDXL and PAXL. It will then be sent to the GPU to increase the inference speed. In Listing 1, you can find the code that is used for the initialization of the diffusion pipeline.

```
pipe = DiffusionPipeline.from_pretrained("stabilityai/stable-diffusion-xl-base-1.0")
pipe.load_lora_weights("nerijs/pixel-art-xl")
pipe.to(device="cuda", dtype=torch.float16)
```

Listing 1 — Initialization of the diffusion pipeline

4.3.2. Image generation

A few parameters (Listing 2) are required to run the previously created diffusion pipeline. First, we need the **prompt** and **negative prompt**. In this case, both the prompt and the negative prompt can be altered to achieve different results. The selected prompt and negative prompt obey the following structure. The prompt contains the `user input` plus the appended string "`simple, flat colors, pixel art, video game, tile`". Hence, the background's negative prompt is "`blurry, photorealistic, 3d render, realistic, character`". With these constraints, we achieve our desired pixel art look and ensure consistent results.

The next value is the **inference step value**, which is the number of denoising steps applied to the initial Gaussian noise. As the number of steps increases, the image's detail increases logarithmically. On the contrary, the processing time grows linearly. Through testing, we chose `20` as this value provides enough detail for our use case while keeping the processing time as low as possible.

The last parameter is the **guidance scale**. The value controls how closely the generated result follows a given prompt. Higher values indicate more accuracy but lower the model's creativity. We chose a relatively low value of `1.5` to maximize our mode's creativity.

```
image = pipe(
    prompt=prompt,
    negative_prompt=negative_prompt,
    num_inference_steps=20,
    guidance_scale=1.5,
).images[0]
```

Listing 2 — Running the pipeline with the given parameters

4.3.3. Post-processing

After finishing the generation, the background creation includes minimal post-processing. Since the model returns an image with a resolution of 1024x1024 pixels, the result is downscaled to our desired resolution of 32x32 pixels.

4.4. Generating characters

The generation of characters builds upon the workflow used to create a background. The largest differences can be found in the prompt structures and the post-processing. The specific changes will be outlined in succeeding sections.

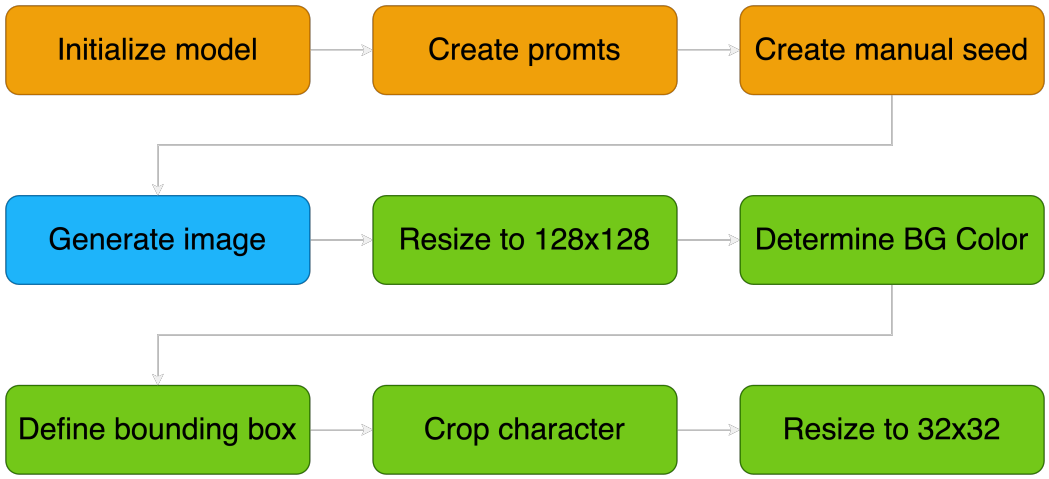

Figure 22 — Pipeline used for character generation

4.4.1. Image generation

The pipeline creation, the inference steps value, and the guidance scale are inferred from the background workflow. On the other hand, the employed prompt and the negative prompt are different.

The prompt used in the character generation contains the `user input` followed by the string "`displayed from the front showing the whole body in the middle of the screen with a clear grey background`". The appended string is essential to separate the character from its background later. Furthermore, the negative prompt looks like this: "`blurry, photorealistic, 3d render, realistic, multiple characters, shadow, detailed`". Like the background generation workflow, the negative prompt guarantees our desired pixel art look and supports consistent results.

To further improve consistency, we limited the range of the utilized **seed** in the character creation. Check Section 4.6 for additional information.

4.4.2. Post-processing

After the generation is finished, we start with the post-processing, which, in the case of the character, includes more steps. When the diffusion pipeline completes, the output can be viewed as a 1024x1024 image. We then take this output and downscale it to 128x128. Therefore, we use Python library functions to create a pixel-perfect image. This is achieved using the resize algorithm with `NEAREST` and a filter function.

After the result has been modified into a perfect pixel image. The character will be cropped so that the proportions stay as there would be no background. The following algorithm is used and also proposed as a pseudo-code in Algorithm 2:

1. At first, the specific background color, which should be a shade of grey, will be chosen by extracting the pixel color of the first point `(0,0)` in the top left corner.
2. Now, the bounding box coordinates are getting initialized by setting the values based on the picture dimensions; in the normal case, they should be `128x128`.
3. As Figure 23 visually explains, the next step is to iterate through every row and get the pixel color value using the Python image library function `Image.getpixel()`.
4. The pixel color value will then be used to check if the current pixel color is not within the color tolerance of the background color by using a custom tolerance function explained in Section 4.7.

5. If this condition is the case, the location of the bounding box coordinates will be saved.
6. The last step is to crop the image to the top left point and bottom right point by using the Python image library function: `Image.crop()`

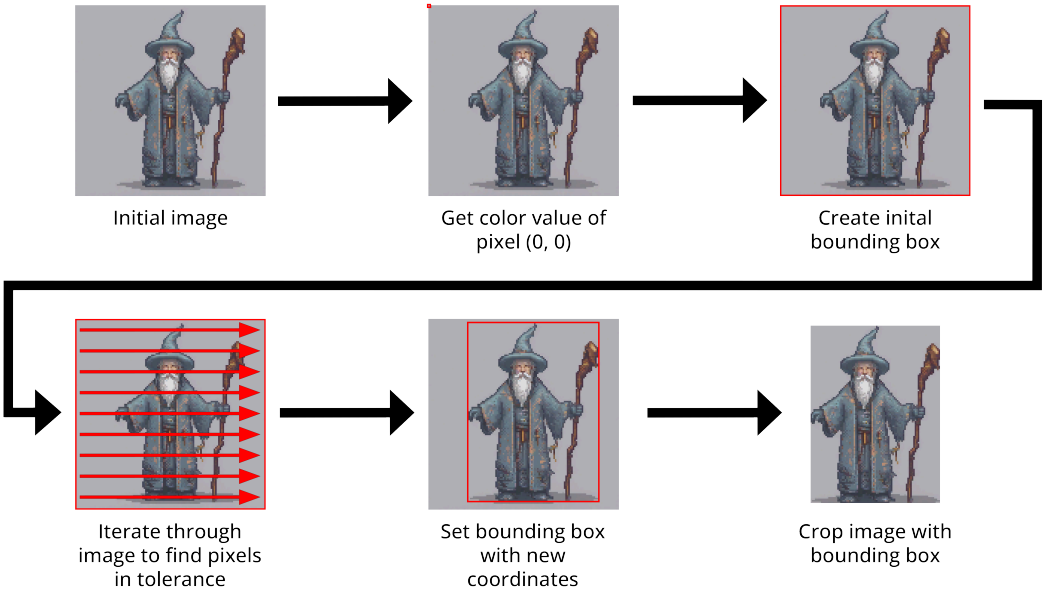

Figure 23 — Algorithm used to crop a character

Finally, the image will be downscaled to the required size, in this case 32x32, using the resize and filter function like before.

4.5. Generating animations

To achieve an animation, two images are generated that can later be alternated to create the animation effect. Generally, the pipeline for accomplishing animations represents an extended version of the character generation pipeline.

4.5.1. Image generation

Like the character workflow, the previously created diffusion pipeline can produce animations. The pipeline's initialization is the same as the background or character pipeline.

When it comes to running the pipeline, there are a few differences. First, instead of using only one prompt, we employ two prompts as we want to create two images. The two prompts should be similar as we want to achieve two images with the same character but a different pose.

As earlier stated, we aim to generate the same image with an equivalent design. Therefore, we must use the same seed to generate both images (Section 4.6). To achieve this, we randomize the seed before running the pipeline and pass the same seed as a parameter to both image pipelines.

4.5.2. Post-processing

The workflow for the post-processing steps is the same as for standard character generation. The only difference is that generated images are stored in a list. Moreover, this implies that all post-processing steps are applied to all list elements.

4.6. Ensuring consistent Results

One of our main goals was to achieve consistently good results. During the experimentation to create animations, we took on the challenge of accomplishing more consistent results, which was crucial for generating two similar images.

The most important parameter for the similarity of resulting images is the used **seed**. The seed is a random integer value used to produce different results even if all parameters are unchanged. Normally, the seed is randomly generated before the image generation is executed.

We had to use a manual seed for our animation creation approach, so we explored various ideas for fixating the seed at a specific value. In our testing, the best solution was to utilize a base seed and generate a random integer offset that is added to it. The simple code for this can be found in Listing 3.

```
base_seed = 4382339547
seed_offset = random.randint(0, 20000)

seed = base_seed + seed_offset
```

Listing 3 — Generating a random seed in a fixed range

In our solution, we define a base seed of `4382339547` and randomize an integer value between `0` and `20000`, giving us `20001` different possible seed values. This number of different seeds makes it highly unlikely that someone who gives the same prompt receives an identical result. Although this particular seed range worked well in our testing, there is no guarantee that all resulting images will be usable.

4.7. Custom tolerance function

Because there is randomness in the image generation and the same background color can't be guaranteed, we decided to create a tolerance function within the earlier explained cropping algorithm. This function uses the `color1`, which is the background color, and the `color2`, which is the current pixel color, and returns `True` or `False` based on the information if the pixel color is insight the tolerance of the background color, meaning the pixel color belongs to the background.

```
def color_within_tolerance(color1, color2, tolerance):
  for c1, c2 in zip(color1[:3], color2[:3]):
      if abs(c1 - c2) > tolerance:
          return False
  return True
```

Listing 4 — Custom tolerance function

4.8. Pseudo-code algorithm for cropping

The following pseudo-code algorithm is proposed to handle the cropping:

Algorithm 2: Custom Cropping Algorithm

input: Image I, tolerance t, Image Height $I.h$, Image Width $I.w$
output: Cropped character image C
get_bounding_box(I, t) $\rightarrow$ **(top_left, bottom_right)**
background_color $\leftarrow$ I.getpixel((0, 0))
leftmost $\leftarrow I.w - 1$
rightmost $\leftarrow$ 0
topmost $\leftarrow I.h - 1$
bottommost $\leftarrow$ 0
for y in 0 to $I.h - 1$ **do**
  for x in 0 to $I.w - 1$ **do**
    pixel $\leftarrow$ I.getpixel((x, y))
    if not color_within_tolerance(pixel, background_color, t) **then**
      leftmost $\leftarrow$ min(leftmost, x)
      rightmost $\leftarrow$ max(rightmost, x)
      topmost $\leftarrow$ min(topmost, y)
      bottommost $\leftarrow$ max(bottommost, y)
    end
  end
end
return (leftmost, topmost), (rightmost, bottommost)
crop_character(I, t) $\rightarrow$ **C**
(top_left, bottom_right) $\leftarrow$ get_bounding_box(I, t)
C $\leftarrow$ I.crop((**top_left,** bottom_right))
return C

5. Results

This chapter presents the result we achieved by utilizing the previously described techniques. We will focus on the earlier outlined categories: `background`, `character`, and `animated character`.

5.1. Background

Figure 24 shows a simple grass background generated by our pipeline with the prompt "`green grass background`". Figure 25 shows a stone background created by our pipeline with the prompt "`cobblestone background, grey textured`"

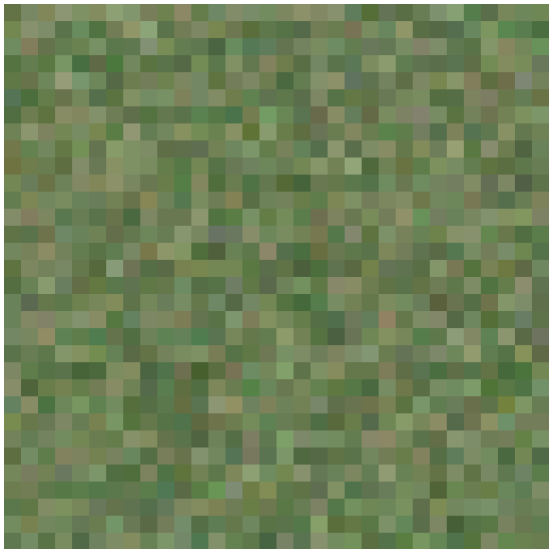

Figure 24 — Grass background

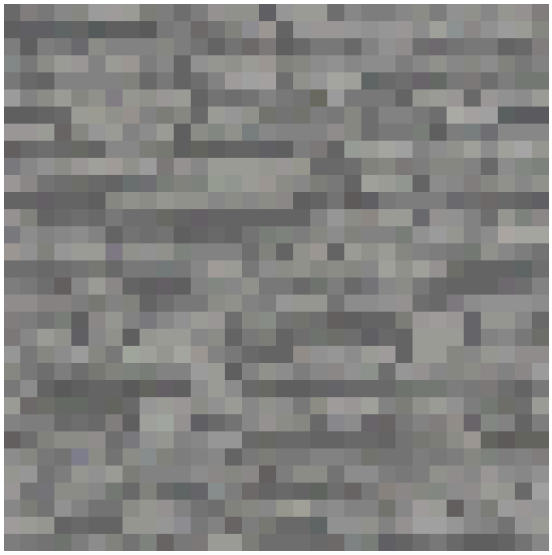

Figure 25 — Stone background

5.2. Character

Figure 26 displays a wizard generated with the prompt "`One wizard with a blue robe and blue hat and wooden staff and white beard`". Figure 27 shows a swordsman created with the prompt "`One swordsman with a green tunic and a green hat carrying a sword`".

Figure 26 — Wizard

Figure 27 — Swordsman

5.3. Animated Character

Figure 28 and Figure 29 display the two different states of the animation when creating an animated character. The prompt "`One woman as a mage and a red robe`" was used in these two images. Furthermore, the prompt of the first state was appended with "`hands next to the body`" and the second state was appended with "`hands in the air`".

Figure 28 — Animation State 1

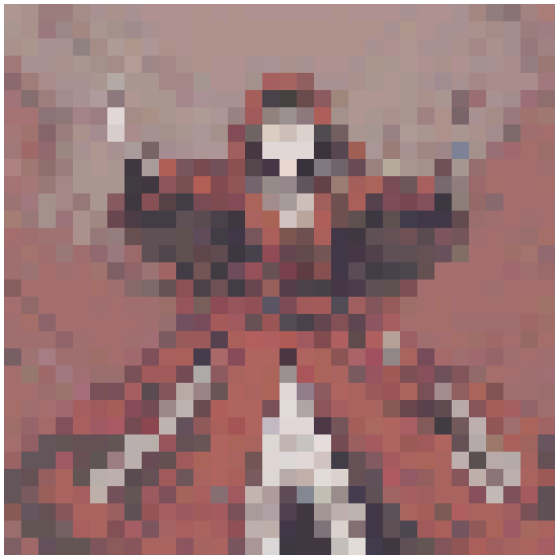

Figure 29 — Animation State 2

6. Conclusion

This thesis explored the development of a diffusion pipeline to optimize the generation of video game assets, focusing on low-resolution tiles and sprites. By combining modern machine learning techniques, advancements in image processing, and state-of-the-art diffusion models, we successfully created a methodology that bridges a significant gap in game asset generation.

To situate our solution in the broader context of existing methodologies, we compare our approach with those discussed in the related work chapter, particularly focusing on using GANs and other diffusion models for similar tasks.

GANs such as DCGANs used by L. Horsley and D. Perez-Liebana [17] and Y. R. Serpa and M. A. F. Rodrigues [18] have shown promising results in generating high-quality sprites and pixel art from sketches. However, GANs often struggle with stability during training and require extensive computational resources and large datasets to produce consistent and high-quality outputs. In contrast, our diffusion-based pipeline offers a more stable approach, efficiently generating diverse and high-quality outputs from textual descriptions without needing large training datasets.

The Pixel Art XL (PAXL) model, which we utilized for fine-tuning Stable Diffusion XL (SDXL), focuses on generating pixel art, providing detailed and stylistically consistent outputs. PAXL inherits the flexibility and performance of SDXL, making it suitable for pixel art styles while maintaining a high resolution that can be downscaled appropriately for game assets.

Similar to PAXL, the Retro Diffusion model targets the generation of retro-styled graphics but caters more to commercial needs and ease of use for those without in-depth ML knowledge [2].

Our methodology employs SDXL fine-tuned with PAXL, leveraging robust post-processing steps and ensuring the generation of static assets and animated sprites.

This makes our approach more versatile and suited for a broader range of applications in game development. Moreover, the combination of controlled seed generation and custom tolerance functions enhances the consistency and reliability of our results.

The primary findings of this thesis contain the following:

1. **Innovation in Diffusion Models**: By leveraging Stable Diffusion and its enhanced version, SDXL, we generated high-quality, low-resolution images suitable for video game contexts. When fine-tuned with techniques such as Low Rank Adaptation (LoRA), these models provided flexibility and effectiveness in creating diverse game assets from textual descriptions.

2. **Efficient Pipeline Creation**: We developed a robust pipeline that generates static backgrounds and characters and extends to creating animated sprites. This pipeline leverages the strengths of SDXL and Pixel Art XL (PAXL) to ensure that the generated assets maintain a pixel-perfect appearance appropriate for game design.

3. **Consistent Results**: Ensuring consistent and repeatable results was critical to our approach. By implementing controlled seed generation and a custom color tolerance function, we produced stable and reliable outputs crucial for game development.

4. **Methodological precision**: The project detailed specific methodologies for generating and post-processing different types of assets. The steps included initialization of diffusion pipelines, prompt structuring, seed manipulation, and detailed post-processing techniques to achieve the desired output quality.

This thesis makes several key contributions to the field of video game asset generation:

- **Automation and Efficiency**: It addresses the challenge of manually creating game assets, offering an automated solution significantly reducing the time and artistic expertise required.

- **Integration of Modern Techniques**: By integrating advanced machine learning models and fine-tuning techniques, this work pushes the boundaries of what is possible in procedural content generation.
- **Enhanced Creativity**: The low guidance scale used in our pipeline promotes creativity in the generated assets, allowing for various outputs while maintaining essential stylistic consistency.

While the project achieved notable success, several limitations were observed:

- **Randomness and Variability**: Despite the controlled seed and structured prompts, some variability in the generated images' quality and style persisted.
- **Complex Animations**: The current approach to generating animations is limited to simple state changes. More complex animation sequences would require further refinement and more sophisticated methods.

In conclusion, the diffusion pipeline developed in this thesis represents a significant step forward in optimizing video game asset generation. It promises to simplify and accelerate the creation of engaging and visually appealing games. The methodologies and findings articulated here lay a strong foundation for ongoing research and development in this exciting interdisciplinary field.

7. Outlook

Creating a diffusion pipeline for optimizing video game asset generation marks a major milestone. However, there are many ideas for improvement and expansion. This chapter outlines key ideas for further development and improvement, focusing on creating a more comprehensive, reliable, and efficient system.

One promising direction is the refinement of the animation generation process. Currently, two slightly different prompts are required to create animation frames. A future goal is to develop a method for generating animations based on a single prompt. This approach could simplify the workflow and produce smoother, more coherent animations by leveraging advanced text-to-image models and incorporating temporal consistency within diffusion models. Automating prompt variations could significantly streamline the creation of animated assets, making the process more user-friendly and efficient.

Improving the reliability and consistency of generated assets is essential. Future enhancements could focus on refining seed generation algorithms to reduce variability and increase predictability. Implementing advanced validation techniques and adaptive feedback loops could ensure that the generated assets consistently meet quality standards. Additionally, integrating more robust error-checking mechanisms and incorporating user feedback during the generation process would help achieve more reliable results.

Efforts to increase the detail and intricacy of pixel art are also critical. This can be achieved by fine-tuning models on high-quality pixel art datasets and integrating super-resolution algorithms to enhance details. User-guided refinement tools could allow developers to adjust details at preliminary stages, ensuring the final output aligns with their artistic vision. This enhancement would make the generated assets suitable and desirable for game development.

User-friendly interfaces and tools are essential for enhancing the diffusion pipeline's usability. Creating plugins for popular game development engines like Unity or Unreal Engine or standalone applications with a graphical user interface would make the technology accessible to a broader range of creators, including those with limited technical expertise. This democratization of game asset creation enables more developers to benefit from advanced AI technologies.

Real-time asset generation could revolutionize Procedural Content Generation (PCG) by allowing on-the-fly creation of game content. This capability would enable games to offer unique, dynamic experiences tailored to each player, enhancing replayability and engagement. Real-time generation requires optimizing the pipeline for speed and efficiency, ensuring it can produce high-quality assets on demand.

Fostering collaboration between AI-generated content and traditional artistic techniques could lead to hybrid creations that combine the best of both worlds. Tools that allow artists to integrate AI-generated assets into their workflows smoothly would enhance productivity and creativity. Providing capabilities for seamless adjustments and refinements could see more widespread adoption of AI-assisted tools in professional environments.

Expanding the pipeline to accept various input modalities, such as sketches, photographs, and audio cues, could dramatically enhance its versatility. This capability would allow for a broader range of creative inputs and outputs, making the diffusion pipeline a more powerful tool for game developers. Multi-modal input handling can drive innovation in game design and asset creation, providing new ways to generate and customize game content.

The outlook for further developing the diffusion pipeline for video game asset generation is bright and expansive. The field can make significant strides in streamlining game development processes, empowering creators, and enriching the gaming experience by pursuing these innovations. Enhancements in reliability, detail, real-time processing, and user accessibility will ensure that AI technology continues to transform game development in meaningful and impactful ways.

References

[1] NeriJS, “Pixel Art XL.” Accessed: Apr. 25, 2024. [Online]. Available: https://civitai.com/models/120096/pixel-art-xl

[2] C. Claus, “Retro Diffusion Extension for Aseprite.” Accessed: Jul. 04, 2024. [Online]. Available: https://astropulse.gumroad.com/l/RetroDiffusion

[3] Glosser.ca, “Artificial neural network with layer coloring.” Accessed: May 07, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network#/media/File:Colored_neural_network.svg

[4] K. O'Shea and R. Nash, “An Introduction to Convolutional Neural Networks.” Accessed: Apr. 22, 2024. [Online]. Available: https://arxiv.org/abs/1511.08458

[5] T. L. Foundation, “ReLU.” Accessed: Jul. 05, 2024. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html

[6] G. for Geeks, “CNN | Introduction to Pooling Layer.” Accessed: Jul. 05, 2024. [Online]. Available: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

[7] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, “Deep Unsupervised Learning using Nonequilibrium Thermodynamics.” [Online]. Available: https://arxiv.org/abs/1503.03585

[8] M. Nixon and A. Aguado, *Feature Extraction and Image Processing*. Academic, 2008. Accessed: Jul. 05, 2024. [Online]. Available: https://books.google.de/books?id=jXmJqzQgdY8C

[9] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, “Score-Based Generative Modeling through Stochastic Differential Equations.” Accessed: Apr. 20, 2024. [Online]. Available: https://arxiv.org/abs/2011.13456

[10] A. Anwar, "Difference between AutoEncoder (AE) and Variational AutoEncoder (VAE)," 2021, Accessed: Jun. 04, 2024. [Online]. Available: https://towardsdatascience.com/difference-between-autoencoder-ae-and-variational-autoencoder-vae-ed7be1c038f2

[11] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010, Accessed: May 07, 2024. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[12] U. of Freiburg, "U-Net: Convolutional Networks for Biomedical Image Segmentation." Accessed: Jun. 30, 2024. [Online]. Available: https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

[13] CompVis - Computer Vision and Learning LMU Munich, "Diagram of the architecture of Stable Diffusion." Accessed: May 30, 2024. [Online]. Available: https://github.com/CompVis/latent-diffusion/blob/main/assets/modelfigure.png

[14] D. Podell *et al.*, "SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis." Accessed: Apr. 25, 2024. [Online]. Available: https://arxiv.org/abs/2307.01952

[15] E. J. Hu *et al.*, "LoRA: Low-Rank Adaptation of Large Language Models." Accessed: Apr. 25, 2024. [Online]. Available: https://arxiv.org/abs/2106.09685

[16] S. Lefebvre, S. Hornus, and F. Neyret, "Texture sprites: Texture elements splatted on surfaces," in *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, 2005, pp. 163–170. Accessed: Jul. 03, 2024. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/1053427.1053454

[17] L. Horsley and D. Perez-Liebana, "Building an automatic sprite generator with deep convolutional generative adversarial networks," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 134–141. Accessed: Jul. 03, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8080426

[18] Y. R. Serpa and M. A. F. Rodrigues, "Towards machine-learning assisted asset generation for games: a study on pixel art sprite sheets," in *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2019, pp. 182–191. Accessed: Jul. 08, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8924853

[19] F. Coutinho and L. Chaimowicz, "On the challenges of generating pixel art character sprites using GANs," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2022, pp. 87–94. Accessed: Jul. 08, 2024. [Online]. Available: https://ojs.aaai.org/index.php/AIIDE/article/view/21951

[20] A. Saravanan and M. Guzdial, "Pixel VQ-VAEs for improved pixel art representation," *arXiv preprint arXiv:2203.12130*, 2022, Accessed: Jul. 08, 2024. [Online]. Available: https://arxiv.org/abs/2203.12130

[21] T. Yuan, X. Chen, and S. Wang, "Gorgeous Pixel Artwork Generation with VQ-GAN-CLIP," 2022.

[22] X. Lin, "Web-based Sprite Sheet Animator for Sharing Programmatically Usable Animation Metadata." Accessed: Jul. 08, 2024. [Online]. Available: https://www.diva-portal.org/smash/record.jsf?dswid=3924&pid=diva2:612988

[23] A. Khalifa, D. Perez-Liebana, S. M. Lucas, and J. Togelius, "General video game level generation," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 2016, pp. 253–259. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2908812.2908920

[24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. Accessed: Apr. 22, 2024. [Online]. Available: http://www.deeplearningbook.org/

[25] A. Krogh, "What are artificial neural networks?," *Nature Biotechnology*, vol. 26, no. 2, pp. 195–197, 2008, doi: 10.1038/nbt1386.

[26] builit, "Fully Connected Layer vs. Convolutional Layer: Explained." Accessed: Jul. 05, 2024. [Online]. Available: https://builtin.com/machine-learning/fully-connected-layer

[27] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-Resolution Image Synthesis with Latent Diffusion Models." [Online]. Available: https://arxiv.org/abs/2112.10752

[28] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes." Accessed: Apr. 22, 2024. [Online]. Available: https://arxiv.org/abs/1312.6114

[29] Y. Liu, E. Jun, Q. Li, and J. Heer, "Latent Space Cartography: Visual Analysis of Vector Space Embeddings," *Computer Graphics Forum*, vol. 38, no. 3, pp. 67–78, 2019, doi: https://doi.org/10.1111/cgf.13672.

[30] J. Ho, A. Jain, and P. Abbeel, "Denoising Diffusion Probabilistic Models." Accessed: Apr. 22, 2024. [Online]. Available: https://arxiv.org/abs/2006.11239

[31] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation." Accessed: Apr. 20, 2024. [Online]. Available: https://arxiv.org/abs/1505.04597

[32] J.-I. Choi, S.-K. Kim, and S.-J. Kang, "Image Translation Method for Game Character Sprite Drawing," *CMES-Computer Modeling in Engineering & Sciences*, 2022, Accessed: Jul. 08, 2024. [Online]. Available: https://cdn.techscience.cn/uploads/attached/file/20220314/20220314063648_98835.pdf

[33] M. Terai, J. Fujiki, R. Tsuruno, and K. Tomimatsu, "Tile-based modeling and rendering," in *Smart Graphics: 8th International Symposium, SG 2007, Kyoto, Japan, June 25-27, 2007. Proceedings 8*, 2007, pp. 158–163. Accessed: Jul. 08, 2024. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-73214-3_14

[34] W. Burgers, "Tile-Based Rendering," vol. 600. p. 3–4, 2005.