Sebastian Vildalen Ekpete

Pixel-Perfect

Style Transfer for Pixel Art Synthesis using Cycle-Consistent Generative Adversarial Networks

Master's Thesis in Computer Science, July 2023
Supervisor: Björn Gambäck

Data and Artificial Intelligence Group
Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

Abstract

This thesis explores the synthesis of pixel art by image-to-image translation using cycle-consistent generative adversarial networks (CycleGANs). This is achieved by translating cartoon illustrations into the pixel art style.

As constrained to the technical limitations of their time, early computer graphics programmers conveyed art using a limited set of pixels and colors, deliberately placing each pixel. Even though the technical limitations are no longer present, pixel art has survived as a popular art form. Despite the simplistic nature of pixel art, artists must adhere to specific stylistic rules and guidelines, making the process time-consuming and meticulous. This motivates the automatic generation of pixel art using creative computer systems. Such systems often employ artificial intelligence and deep learning algorithms to generate images but are rarely optimized for the generation of pixel art.

By fine-tuning and training state-of-the-art deep learning models and leveraging a custom, high-quantity, and quality dataset, this thesis succeeds in generating pixel art on the level of human artists. This is done through a series of experiments, where the importance of the dataset is examined, and different model architectures are iterated and improved upon. All the experiments have the CycleGAN architecture as a foundation, and three significant architectural changes are examined, each optimized in response to the quality of the generated images.

The generated images are evaluated using visual inspection and statistical metrics. The visual inspection and statistical metrics compare the results of the models against each other and show that the models are able to translate to the pixel art style effectively, with certain models performing better. Furthermore, a new color consistency evaluation metric for image-to-image translation models is introduced, which compares color palettes of original and translated images. The results of this metric show that the models ensure sufficient color consistency between the images.

The results from the best-performing model are further evaluated through a domain expert survey. The responses to the survey reveal that the model can generate artwork on the level of a beginner human pixel artist but that the model cannot incorporate advanced pixel art techniques. Despite this, the domain experts expressed that the results were impressive compared to other AI-generated pixel art. The generated images show adherence to the basic rules and guidelines of the pixel art style, and the model shows applicability as an assistive system to pixel artists, designers, and video game developers alike.

Sammendrag

Denne masteroppgaven utforsker generering av pikselkunst gjennom bilde-til-bilde oversettelse ved bruk av sykliske generative nevrale nettverk (CycleGANs). Dette blir gjort gjennom å oversette digitale illustrasjoner til pikselkunst.

Som følge av de tekniske begrensingene på 70- og 80-tallet, formidlet grafiske designere og programmerere kunst ved hjelp av et begrenset sett med piksler og farger, ved å bevisst plassere hver piksel individuelt. Til tross for at disse tekniske begrensingene ikke lenger finnes, så har pikselkunst overlevd som en populær kunstform. Selv om pikselkunst kan virke enkelt, så må pikselkunstnere følge spesifikke regler og retningslinjer i henhold til kunststilen, som gjør prosessen tidkrevende og vanskelig. Dette motiverer til automatisk generering av pikselkunst ved bruk av kreative datasystemer. Slike systemer benytter ofte kunstig intelligens og dyp læring for å generere bilder, men de er sjelden optimalisert for generering av pikselkunst.

Ved å utvikle og trene generative maskinlæringsmodeller og bruke et høy-kvalitets datasett, klarer denne oppgaven å genere pikselkunst på nivå med menneskelige kunstnere. Dette oppnås gjennom en rekke eksperimenter, der betydningen av datasettene testes, og flere utvidelser til modell-arkitekturen blir undersøkt. Alle eksperimentene bruker den originale CycleGAN-arkitekturen som grunnlag, og tre store arkitekturendringer blir undersøkt, der hver arkitektur blir optimalisert med hensyn til kvaliteten på de genererte bildene.

De genererte bildene blir evaluert ved hjelp av visuell inspeksjon og statistiske metrikker. Den visuelle inspeksjonen og de statistiske målingene sammenligner resultatene fra modellene med hverandre og viser at modellene effektivt klarer å oversette bilder til pikselkunst-stilen. Videre blir det introdusert en ny metrikk for evaluering av fargebevaring for bilde-til-bilde oversettelsesmodeller, som sammenligner fargepalettene til originale og oversatte bilder. Resultatene fra denne metrikken viser at modellene oppnår tilstrekkelig fargebevaring mellom bildene.

Videre blir resultatene fra den beste modellen vurdert via en undersøkelse med eksperter innenfor pikselkunst-domenet. Svarene på undersøkelsen viste at modellene er i stand til å generere kunstverk på nivå med en nybegynner innen pikselkunst, men at bildene ikke viser noe form for avanserte pikselkunstteknikker. Til tross for dette, uttrykte ekspertene at resultatene var imponerende sammenlignet med annen AI-generert pikselkunst. De genererte bildene viser bruk av de grunnleggende reglene og retningslinjene som finnes for pikselkunst-stilen, og den endelige modellen viser anvendelighet som et støttesystem for pikselkunstnere, designere og videospillutviklere.

Preface

This thesis is the final part of a Master's degree in Computer Science with a specialization in Artificial Intelligence at the Norwegian University of Science and Technology (NTNU) in Trondheim. The work was conducted at the Department of Computer Science under the supervision of Professor Björn Gambäck.

Parts of the work in this thesis were inspired by work in my specialization project, *Computational Creativity with Generative Adversarial Networks*, written during the fall semester of 2022. Literature, explanations, and equations from the specialization project are adapted in some sections and will be indicated in the respective section introductions.

The experiments and training of the models in this thesis would not be feasible without the computing resources from the IDUN cluster (Själander et al., 2019). A thanks goes out to the High-Performance Computing group at NTNU.

Furthermore, I would like to thank artists Italivy Cortes and Nicola Di Concilio for evaluating the model results and providing thorough feedback.

I also want to thank my supervisor, Björn Gambäck, for guidance and valuable feedback during the past two semesters.

Finally, a special thanks to my parents for always supporting me in any way possible and inspiring me to pursue a Master's degree.

Sebastian Vildalen Ekpete
Trondheim, 3rd July 2023

Contents

List of Figures

List of Tables

1. Introduction

Artificial intelligence (AI) applications in creative tasks have seen great attention and innovation in recent years. The intersection of AI and creativity is in the field of computational creativity, where new research and advancements are continuously being made. Computational creativity can be defined as computational systems exhibiting behaviors that unbiased observers deem creative (Colton and Wiggins, 2012). These systems often use machine learning models and neural networks to learn from examples and generate new works. With the ascent of available computing resources and open-source machine learning frameworks and models, creating, leveraging, and using AI systems is highly accessible. As a result of recent advancements and a surge in the public interest, such systems can now create complex art in multiple domains and exhibit creative behavior on the level of professional artists. This is highly evident in digital art creation, where AI systems can generate artwork from text prompts, transfer the style of one image to another, and even create new art from seemingly nothing. One popular class of models used in these systems is Generative Adversarial Networks (GANs, Section 2.7.1). GANs can effectively create artworks of many styles but are highly limited to the art styles they learn from. Publicly available systems today lack the means to create quality images of the pixel art style, which motivates the need for a specifically designed AI model for generating pixel art.

This thesis explores the generation of pixel art by image-to-image translation using cycle-consistent generative adversarial networks (CycleGAN, Section 2.7.7). This is done through experiments where three architecture variants are tested and optimized according to the generated images' quality. The results of the models are evaluated using visual inspection, statistical metrics, and a domain expert survey. Furthermore, a new color consistency evaluation metric for image-to-image translation models is introduced, which compares color palettes of original and translated images. This metric is introduced since the colors of the input images should not be changed during the style transfer.

The models in this thesis are trained to translate images from a cartoon domain to the style of a pixel art domain. For the models to effectively translate images, the training data must be diverse, of sufficient size, and accurately represent the distributions and styles of the two domains. No large datasets of quality pixel art are currently available, suggesting the need for collecting a custom one. In addition, many image generation models are trained on data found online, despite the possible intellectual property rights of the images. Because of this, the intersection of art and AI is a highly debated subject, as many artists fear that AI systems "steal" and copy from artworks. This motivates the need for collecting a dataset from sources under creative commons licenses, such as CC BY-SA 4.0[1] and CC0 (public domain)[2]. This thesis provides a custom-collected open-source dataset with a total of 3,600 images consisting of cartoon illustrations and pixel art.

1.1. Background and Motivation

Images displayed on screens are composed of individual pixels, and early computer graphics were programmed by controlling the colors of these pixels one at a time (Silber, 2015). This was due to technical restraints, and low-resolution graphics were the only choice. With the advent of the video game industry, many techniques and best practices for creating aesthetically pleasing low-resolution artwork were introduced. Despite the technical limitations having disappeared, pixel art's unique style, nostalgia, and retro feel have contributed to its survival and popularity today. It can be found in video games produced today, in addition to becoming an independent art form outside of video games, and is popular among digital artists.

Conceptually, the art form is not limited to digital images, as it has many similarities to other art forms and techniques, such as pointillism, mosaic art, and cross-stitching. The common factor here is that the artworks comprise discernable small dots or squares to form illustrations. In pixel art, the placement and color of each pixel must be chosen intentionally since they encompass much more information in low-resolution images than in high-resolution images. This demands careful consideration and skill when creating pixel art. The process of creating pixel art involves more complexity than solely reducing the resolution of existing images. It demands adherence to established stylistic rules and guidelines, including how lines and outlines are created, shading techniques, and the use of color. The most basic rule is that each pixel must be discernible and have a uniform square shape with one color. The total use of colors in the artwork should be limited, and the transitions between colors should either be sharp or empose shading techniques such as dithering. Curved lines should be formed in a zig-zag manner, and there should be a clear distinction between objects and image elements. An example of pixel art is shown in Figure 1.1.

Learning the basic rules and placing pixels is of a low threshold, but creating advanced and aesthetically pleasing pixel art takes creativity, ingenuity, and advanced techniques. As with all art creation, it takes practice to excel, and pixel art can be challenging due to the stylistic rules, even for experienced artists. In addition, creating large quantities of pixel art can be time-consuming. These factors motivate the automatic generation of pixel art as a tool for beginners, artists, designers, and video game developers. This field has garnered some research attention, but most algorithms employ clustering and downsampling techniques. Using deep learning to synthesize pixel art requires more research, and this thesis explores this field.

[1]https://creativecommons.org/licenses/by-sa/4.0/
[2]https://creativecommons.org/share-your-work/public-domain/cc0/

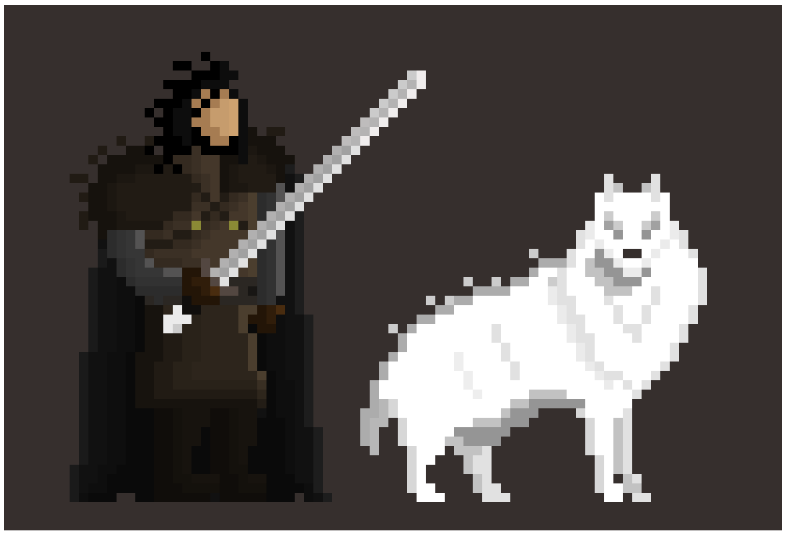


Figure 1.1.: A pixel art illustration with discernible pixels, shading techniques, and a minimal color palette. Image by author.

1.2. Goals and Research Questions

Goal *Create a generative adversarial network model capable of synthesizing high-quality pixel art.*

Pixel art is an art form with a distinct style. In order to be considered high quality, the artwork must conform to a specific set of rules and principles. Current generative models can struggle with certain stylistic demands, and generative adversarial networks (GANs, Section 2.7.1) do not perform well in the synthesis of pixel art. To achieve the goal of this thesis, image-to-image translation techniques will be explored using a cycle-consistent adversarial network, and suitable datasets will be collected and assembled. Experiments will be conducted to document how the techniques and the datasets affect the generated pixel art. The thesis will try to answer the following research questions.

Research question 1 *How can GAN architectures be designed to be able to create pixel art effectively?*

Several experiments will be carried out to test different architecture extensions, techniques, and configurations. These will build upon the official implementation of the state-of-the-art CycleGAN model (Section 2.7.7). The effects of hyperparameter tuning and optimization techniques will also be researched.

Research question 2 *What impacts do the datasets' quality and quantity have on the generated artwork?*

A major issue in the synthesis of pixel art by generative models is the lack of suitable available datasets. Both the quality and the quantity of the datasets may have significant impacts on how well the models can perform. The experiments will be conducted using a variety of datasets to review these impacts.

Research question 3 *How can the performance of the models be measured and evaluated?*

The generated samples must be compared to the real images to evaluate the model. The quality and the variety of the generated samples need to be captured. This can be done with commonly used metrics such as Fréchet Inception Distance or Kernel Inception Distance (Section 2.7.8). Human evaluation may also be needed to further evaluate the quality of the generated samples. This may be done through conducting surveys and consulting with domain experts. Custom evaluation metrics to explicitly evaluate pixel art will be explored.

1.3. Research Method

Several research methods will be applied to answer the three research questions and ultimately try to achieve the research goal. First, a study on necessary background theory will be conducted to gain a thorough understanding of the technologies applied. Furthermore, literature relevant to the synthesis of pixel art will be explored. Relevant literature from the specialization project mentioned in the Preface will be included, and the Google Scholar search engine[3] will be used to find new literature.

Several experiments will be conducted to answer Research Questions 1 and 2. Different model architectures will be tested and optimized during the experiments according to an experimental plan. To answer Research Question 3, the collected dataset will be compared to a subset of itself and to an existing lower-quality dataset.

The generated images during training will be visually inspected to evaluate the individual architectural changes during the experiments. After the experiments, the results from the best-performing models will be further evaluated and compared using a combination of visual inspection and statistical metrics. Finally, a domain expert survey will assess the resulting pixel art images.

1.4. Contributions

The main contributions of this thesis are as follows:

1. *An investigation of architectural changes to the base CycleGAN model for effectively generating pixel art.*

2. *An introduction of a new color palette evaluation metric. This metric extracts dominant colors from images and scores how consistent the colors are with each other. This metric applies to all style transfer tasks where the input and the output images should have the same dominant colors.*

[3]https://scholar.google.com

3. *A collection of a high-quality dataset with 1,800 cartoon illustrations and 1,800 pixel art images, all under a creative commons license.*

4. *An approach for implementing a ResNet generator with modified upsampling layers and spectral normalization in the discriminator. The implementation generates results with no artifacts and sufficient color consistency between input and generated images.*

5. *An approach for implementing a U-NET generator with added SSIM and perceptual loss functions. This implementation was shown to have improved color consistency.*

1.5. Thesis Structure

- Chapter 2 introduces the theory, tools and methods necessary to understand the work.
- Chapter 3 gives an overview of related work in the field of pixel art synthesis.
- Chapter 4 introduces the collected custom dataset used in the experiments.
- Chapter 5 presents the network architectures used in the experiments.
- Chapter 6 provides the experimental plan, the experiments, and the experimental results.
- Chapter 7 evaluates, compares, and discusses the experimental results in the context of each other and the context of real pixel art.
- Chapter 8 concludes the thesis and provides the main contributions and suggestions for future work.

2. Background Theory

This chapter provides the necessary background theory for understanding the research conducted in this thesis. First, Section 2.1 presents an introduction to machine learning. Section 2.2 introduces neural networks and deep learning, including network elements, parameters, and optimizations. Section 2.3 presents how images are represented for use in neural networks. Sections 2.4, 2.5, and 2.6 present various deep learning models, respectively CNN, ResNet, and U-NET. Section 2.7 introduces GAN, along with several variants, such as CycleGAN. Lastly, a selection of GAN evaluation metrics is presented.

2.1. Machine Learning

Machine learning algorithms learn from a set of input data. The most common form of machine learning, supervised learning, requires labeled data. This data is a set of real-world examples from a particular domain. The data is made up of features (inputs), denoted by X, along with corresponding labels (outputs), denoted by y. The algorithms work by making output estimations based on the input X and computing errors (or losses) by comparing the estimations to the correct output y. The comparison is done by an objective function (often called loss function) and is used to gradually improve the accuracy of the model by updating its parameters θ. This process of evaluating and optimizing is done repeatedly until the accuracy of the model reaches a certain threshold (IBM, 2022b). In short, supervised algorithms want to model the probability distribution $p(y|x, \theta)$. For the classification example, this would mean the probability of example x belonging to class y, based on θ. Supervised machine learning models which predict class labels are called discriminative models.

In contrast to supervised learning, unsupervised learning uses datasets with no labels. These algorithms learn the properties of the structure of the input data. Unsupervised models want to learn the probability distribution of the dataset $p(X)$, without any labels y (Goodfellow et al., 2016, p. 105). This means that instead of comparing the models' output to a label y, the output is compared to the input x by some objective/loss function. The model can then use the generated probability distribution to fabricate new data instances that are similar to or even indistinguishable from the input data. Machine learning models that generate these sorts of distributions are called generative models.

The input data to machine learning models is often divided into training data and test data. The training data is used during the learning process, where the model is trained and optimized. Therefore, the training data should be of a large enough size to represent the dataset as a whole. The test set is used to evaluate if the model can generalize to new, unseen data. The challenge for the algorithm is to make the model fit the training data by reducing the training error, while also reducing the test/generalization error (Goodfellow et al., 2016, p. 110).

2.2. Neural Networks

Artificial neural networks (ANNs) are machine learning algorithms that are good at recognizing patterns in large amounts of data. ANNs are often used in discriminative tasks, but can also be used with unlabelled data in generative tasks. Relevant tasks can be found in many fields, with some being object recognition (Zhao et al., 2019), speech recognition (Graves et al., 2013), and image creation (Goodfellow et al., 2014). ANNs are modeled on a simplified version of the human brain, in that they consist of large numbers of interconnected neurons (or nodes). Neurons in a neural net mimic how biological neurons send and receive signals between each other. Together, neurons in a neural network form layers, and neural nets with several layers are called deep neural networks. The resulting learning algorithms are under the term *deep learning*. The most prominent types of deep learning models are deep feedforward networks, also called feedforward neural networks or multilayer perceptrons (MLPs).

A feedforward neural net wants to approximate some function f, and for the classification example, the network creates a mapping $y = f(x;\theta)$, analogous to finding $p(y|x,\theta)$ (Goodfellow et al., 2016, p. 167). A network consists of several layers, namely the input layer, the hidden layer(s), and the output layer. The input layer is just a representation of the input data in the form of nodes. The hidden layers are the intermediate layers in the network where computation and training take place. The output layer produces the results after the computations in the hidden layers (Goodfellow et al., 2016, p. 168). Together, the layers form the function f. In each node in each layer, computation takes place during forward propagation, and when the input data has propagated through every layer, one forward pass has been completed. After a forward pass, the loss is computed, and the network parameters are updated through backpropagation. This cycle repeats until the model converges. After the model has converged, one can feed the network with new examples. The new example is propagated forward through the trained network until an output prediction is computed. In short, neural network models are trained using an optimization algorithm that incrementally adjusts weights to minimize a loss function. These weights are then used to make predictions, with the use of a forward pass through the network.

2.2.1. Forward Pass

The neural network in figure Figure 2.1 has two input variables, which can be represented as a vector X. The weights going from the first layer can be represented as a matrix W_1. The weights indicate the importance of the inputs to each node. A weighted sum of the inputs is computed during the forward pass. This sum is added with a bias matrix B_1, which has the same dimensions as W_1. The bias matrix are a set of constants to adjust the outputs of the nodes. The resulting matrix Z is then fed into a ReLU activation

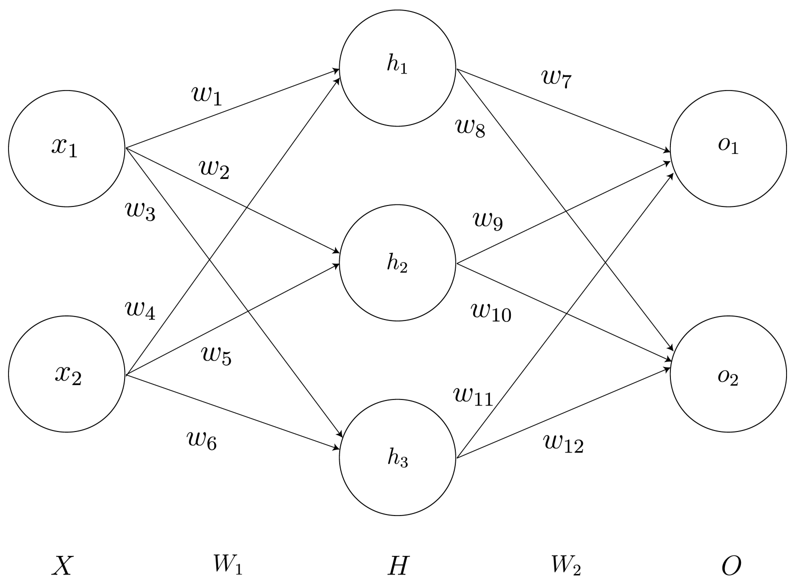


Figure 2.1.: Model of a fully connected neural network (FCNN) with one hidden layer. The input layer $X = \begin{pmatrix} x_1 & x_2 \end{pmatrix}$ has two input variables. The hidden layer H is composed of three hidden nodes: $H = \begin{pmatrix} h_1 & h_2 & h_3 \end{pmatrix}$. The output layer O has two output nodes: $O = \begin{pmatrix} o_1 & o_2 \end{pmatrix}$. The arrows represent the forward pass through the network, and each arrow has a corresponding weight w_i. The weights w_i between each layer form the weight matrices $W_1 = \begin{pmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \end{pmatrix}$ and $W_2 = \begin{pmatrix} w_7 & w_8 \\ w_9 & w_{10} \\ w_{11} & w_{12} \end{pmatrix}$. Deep neural nets are scalable to have many hidden layers, and thousands or even millions of nodes in each layer. The number of nodes in the input layer is dependent on the number of features in the input data, and the number of output nodes is dependent on what is being predicted. The number of hidden layers and the number of nodes in them are dependent on the complexity of the task at hand and are up to the designer of the network to decide.

function (Figure 2.2, page 11) in the hidden layer. The output vector H of the hidden layer becomes the input to the final layer, and the same calculation is done again. This time with the weights matrix W_2 and the bias matrix B_2. Finally, the output layer's activation function σ produces the output O. The complete computation of one forward pass is shown in Equation 2.1.

(2.1) Computations of a single forward pass in an FCNN. " $\cdot$ " is the dot product.

$$\begin{aligned} Z_1 &= X \cdot W_1 + B_1 \\ H &= ReLU(Z_1) \\ Z_2 &= H \cdot W_2 + B_2 \\ O &= \sigma(Z_2) \end{aligned} \tag{2.1}$$

2.2.2. Activation Functions

Activation functions are mathematical functions that are applied to the output of a neuron, or a group of neurons, to introduce non-linearity into the model. The standard functions that are used are often $Tanh(x)$ or $\sigma(x)$. However, these saturating nonlinearities are much slower than non-saturating nonlinearity functions (Krizhevsky et al., 2012). In this case, saturating functions refer to functions that squeeze numbers into a range. The ReLU activation function (Figure 2.2 (a)) for neurons provides much faster training for gradient descent, especially with large networks and datasets. Because of this computational efficiency, ReLU is one of the most successful and popular choices of activation functions, especially in convolutional neural networks (Krizhevsky et al., 2012). One issue of ReLU is the "dying ReLU" problem when ReLU neurons become inactive and only output 0 for any input, which prevents learning (Lu et al., 2019). An alternative is to use LeakyReLU (Figure 2.2 (c)), which has a small slope for negative values instead of ReLU's flat slope.

For networks that want to predict a probability as an output, the sigmoid activation functions are good choices, because they produce an S-curved shape in the range [0,1]. The standard sigmoid function is the logistic function (Figure 2.2 (b)). Sigmoid pairs well with the binary cross-entropy loss function (2.4) in the context of networks made for binary classification problems. Sigmoid and BCE are used in GANs (2.7.1) to decide if images are real or fake. In some GAN variants, the output is fed through a Tanh (Figure 2.2 (b)) activation function to map the output to the range [-1,1].

2.2.3. Loss Functions

After the forward pass, the predicted output vector O is compared with the ground truth $Y = \begin{pmatrix} y_1 & y_2 \end{pmatrix}$ to get a measure of the degree of discrepancy between the two. This measure is performed through an objective/loss function. The loss function can be any function, but two popular ones, especially in image processing, are the $L1$ and $L2$ loss functions (Zhao et al., 2017). $L1$ loss calculates the absolute difference between the predicted output o and the true value y. $L2$ loss calculates the squared difference. In practice, $L1$ is analogous to the Mean Absolute Error function (MAE, Equation 2.2), and $L2$ is analogous to Mean Squared Error (MSE, Equation 2.3). The difference is that MAE and MSE calculate the average of the differences across the whole training set, and are called cost functions, while $L1$ and $L2$ calculate the loss of one training example (IBM, 2022a).

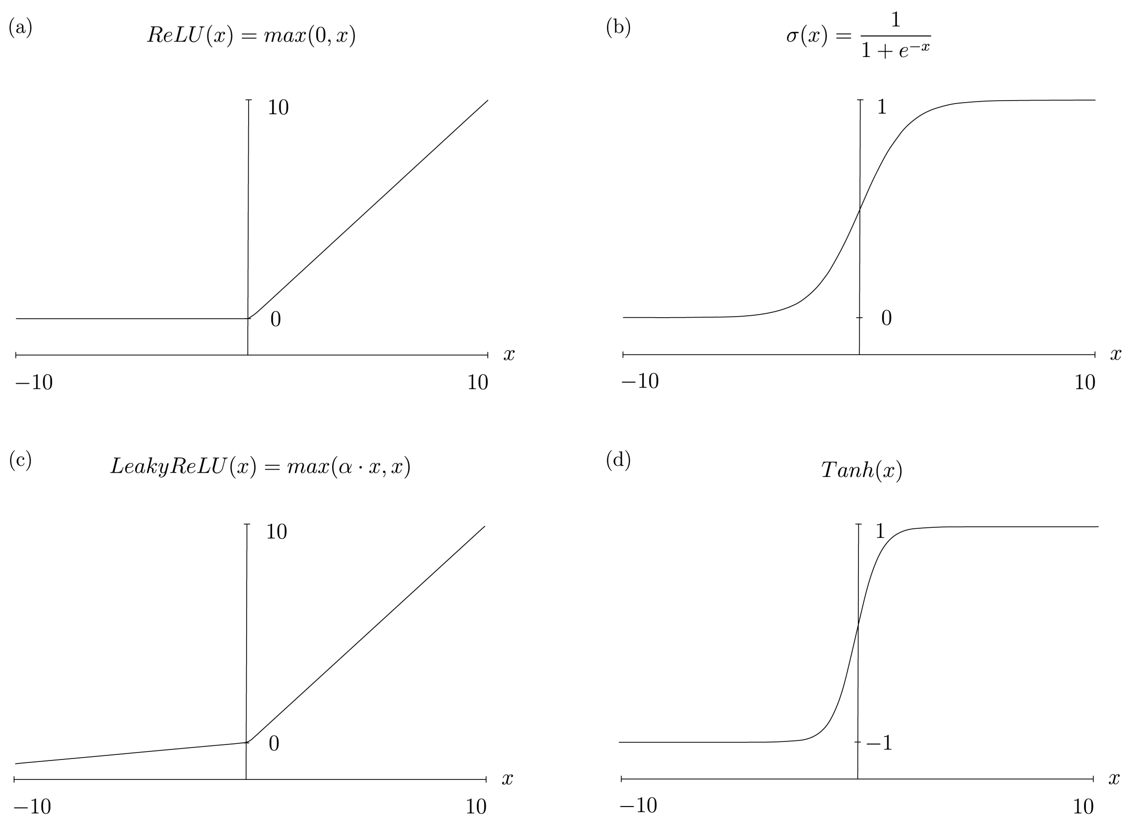


Figure 2.2.: **(a)**: *ReLU* (Rectified Linear Unit) activation function. Returns the input x if x is positive, returns 0 otherwise. **(b)**: Sigmoid (σ) activation function. Maps the input x to the range $[0, 1]$. **(c)**: LeakyReLU with slope $\alpha = 0.2$. **(d)**: Hyperbolic tangent *Tanh*.

$$L_{MAE} = \frac{1}{N}\sum_{i=1}^{N}|Y_i - O_i| \tag{2.2}$$

$$L_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(Y_i - O_i)^2 \tag{2.3}$$

The choice of loss function is not arbitrary and is dependent on the task at hand. E.g. $L1$ and $L2$ loss are good choices for many prediction tasks. If the network is a classifier where the classification task only has two class labels, it is often efficient to use a binary cross-entropy loss function (Mao et al., 2016), also known as log or BCE loss (Equation 2.4). In this case, the network only has one output node o. Here, o is the probability of the prediction belonging to the first class, while $1 - o$ is the probability of the prediction belonging to the second class.

$$L_{BCE} = \frac{1}{N}\sum_{i=1}^{N} -y_i log(o_i) - (1-y_i)log(1-o_i) \tag{2.4}$$

2.2.4. Backpropagation and Gradient Descent

The learning part of a neural network takes place during backpropagation, and it is during this process the network's parameters are updated. After each forward pass, a backward pass follows, and this cycle is repeated iteratively to improve the network's performance until a predefined error threshold is reached. The mathematics behind the backpropagation involves partial derivatives and the chain rule to compute gradients. Gradients are vectors that point in the direction of the steepest ascent on a curve, and in the case of neural networks, the curve is the loss function. After computing the total gradient of the loss function starting from an arbitrary point and subsequently traversing in the opposite direction, the minimum of the loss function can be identified after a sufficient number of iterations. The point where the loss function has its lowest value is known as the point of convergence (IBM, 2022a). This optimization technique is called stochastic gradient descent (SGD).

To compute the total gradient, the gradient of the loss function is computed separately in each layer with respect to the current layer's parameters, conducted in a backward manner through the network. The simplified steps of the backpropagation algorithm for the example network in Figure 2.1 are shown in equations (2.5)-(2.10). Note that only the calculations for the weights are included. In practice, biases are updated in the manner as the weights, but with the derivative of the loss with respect to the biases.

(2.5)-(2.10) (Simplified) computations of a single backward pass in an FCNN. " $\cdot$ " is the dot product.

$$Error_{output} = L'(y,o) \cdot \sigma_2'(Z_2) \tag{2.5}$$
$$Error_{hidden} = Error_{output} \cdot W_2 \cdot ReLU'(Z_1) \tag{2.6}$$

$$dW_2 = Error_{output} \cdot H \tag{2.7}$$
$$dW_1 = Error_{hidden} \cdot X \tag{2.8}$$

$$W_2 = W_2 - (lr \cdot dW_2) \tag{2.9}$$
$$W_1 = W_1 - (lr \cdot dW_1) \tag{2.10}$$

Equations 2.5-2.6 show the computation of the error term for the output layer O and the hidden layer H. L' is the derivative of the cost function. σ' is the derivation of the activation function and Z_2 is the input to the output layer. Equations 2.7-2.8 show the computation of the cost derivatives for the weights, where H is the output of the hidden layer, and X is the output of the input layer. Equations 2.9-2.10 show the updating of the weights by subtracting them with their respective cost derivatives multiplied by the learning rate lr. lr is a predefined hyperparameter. After the gradients are computed, the weights and biases are updated, and another forward/backward pass with new training examples can be conducted. The mechanics of forward/backward passes and gradient descent mostly stay the same for both supervised and unsupervised training. However, other network features such as how loss functions are handled differ between them.

2.2.5. Weight Initialisation

The weights W in a network should be initialized to small random values prior to training. When working with networks that employ ReLU activations, He Initialization (He et al., 2015) is a good choice. The weights for each node derived from He Initialization are random numbers from a zero-centered Gaussian probability distribution ($\mathcal{N}$) with a standard deviation of $\sqrt{2/n}$. He initialization for weight w_l is show in Equation 2.11. Here, n_l is the number of inputs to the node. Initializing weights can prevent the network from reducing or magnifying the magnitudes of the input signal exponentially (He et al., 2015). He Initialization sets biases to 0.

$$w_l = \mathcal{N}(0, \frac{2}{n_l}) \tag{2.11}$$

2.2.6. Hyperparameters

Hyperparameters are parameters that need to be chosen before network training begins. Hyperparameters are not the weights and biases, but rather a set of values that control the learning process of the model. Correct configurations of these values are key to the learning process, but the choice of these values often needs to be revised through empirical testing of the model. Hyperparameters include but are not restricted to the structure of the network, the learning rate, the division of the input data (batches and batch size), and the number of iterations.

Hidden Layers and Neurons

The network structure refers to the number of hidden layers and the number of neurons in each layer. The number of hidden layers is the number of layers between the input and output layers. Increasing the number of hidden layers and the number of hidden neurons can increase the network's capacity to learn complex features, but it can also increase the risk of overfitting. Overfitting is discussed in Section 2.2.9.

Learning Rate

Learning rate controls the step size used during gradient descent to update the weights of the neural network, i.e. how fast the network should learn from its gradients with each iteration. A larger learning rate can result in faster convergence, but it may also cause

[1]https://creativecommons.org/licenses/by/3.0/

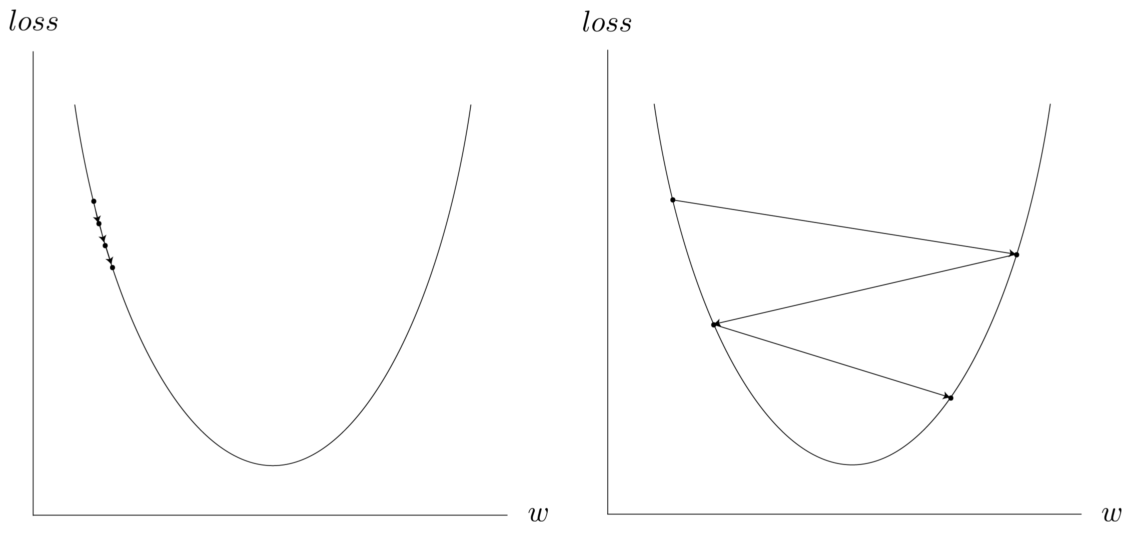


Figure 2.3.: The size of the learning rate and its effect on convergence. **Left:** Small step size can lead to slow convergence. **Right:** Large step size can lead to overshooting. Figure adapted from Cui (2018), under licence CC BY 3.0[1].

the model to overshoot the optimal weights and lead to instability and even divergence. A smaller learning rate can cause slow convergence and cause the optimization process to be stuck in a local minimum of the cost function. The effects of the step size are shown in Figure 2.3. Typically, the learning rate is set to a small positive value, such as 0.1, 0.01, or 0.001. However, the optimal learning rate can vary depending on several factors, including the architecture of the network, the complexity of the problem, and the size of the dataset. Finding the optimal learning rate is important for the convergence of the network, and can be done automatically by gradient descent optimization techniques.

Batches and Epochs

The dataset to be used in training neural networks can range from only a few to a million or more different samples. A neural network normally can not handle entire large datasets at once, so the dataset is often divided into equally sized mini-batches. The size of each mini-batch is usually defaulted to a power of two, with normal values being 32, 64, or 128. Every time the network has passed through every sample in the dataset, one epoch is completed. The number of epochs defines how many times the network will use gradient descent on the entire dataset. For each epoch, the network completes a number of n iterations, where the gradients are computed and updated every iteration. The number of iterations depends on the variant of gradient descent.

2.2.7. Gradient Descent Variants

The gradient descent algorithm has several variants and optimizations. The three major variants are SGD, batch gradient descent, and mini-batch gradient descent. SGD updates the network's parameters for each training example x_i individually. Here, the number of iterations is the same as the number of N training samples and corresponds to one epoch. Batch gradient descent, on the other hand, updates the parameters by computing the gradient of the loss function for the entire training set. Here, one epoch corresponds to one single iteration, since the batch size is 1. This can lead to faster convergence but can be computationally expensive for large datasets compared to SGD. Mini-batch gradient descent is a compromise between the previous two, as it computes the gradients for smaller batches. The parameters are then updated based on the average of the gradients computed for each batch. Here, one epoch has $\frac{N}{b}$ iterations, where N is the number of samples and b is the batch size. Mini-batch gradient descent can provide a good balance between computational efficiency and accuracy (Cotter et al., 2011).

2.2.8. Gradient Descent Optimisations

The three gradient descent variants do not guarantee convergence, and the choice of learning rate can heavily influence this, as seen in Figure 2.3. To combat this issue, several gradient descent optimization algorithms have been introduced. One of these optimizers is Adam (Adaptive Moment Estimation, Kingma and Ba (2015)), and its job is to help with hyperparameter initialization and updating. Adam adapts the learning rate to different parameters automatically, based on the statistics of gradients during training (Zhang, 2018). Specifically, Adam stores an exponentially decaying average of the gradients and their squared values. It then uses these estimates to update the learning rate for each parameter. Adam has three main configuration parameters: α, β_1, and β_2. α is the learning rate, and $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of the moving averages of the gradient and the squared gradient. Kingma and Ba recommend $\alpha = 0.001$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ as starting values. Adam is well-suited for dealing with noisy data since it can continuously adjust the learning rate in response to fluctuating gradients.

2.2.9. Overfitting

When the gap between the model's training error and test error is too large, the model is overfitting. In other words, the model is memorizing properties of the training set that do not apply to the test set (Goodfellow et al., 2016, p. 111-112). This defeats the purpose of the model since it cannot perform accurately on new data. Overfitting can occur when the model is trained for too long or if the model is too complex compared to the complexity of the input data. However, if the training is stopped too early or the model is not complex enough, the model may become underfitted and not able to establish a connection between the input and the output data.

To achieve the optimal balance of the model, where it can both find the underlying structure of the training data while also generalizing well to new data, regularization approaches are often used. Regularization approaches are modifications to the networks which are intended to reduce their generalization error but not their training error (Goodfellow et al., 2016, p. 120). The Adam optimizer has an option for regularization, where it can use weight decay. Weight decay is when a penalty term is added to the cost function, based on the parameter λ.

Another form of regularization is the dropout technique. Dropout randomly drops nodes and their connections from the network during training. Every node has a probability p of being dropped. This simple, yet effective technique, prevents the network from adapting too much and can improve performance in image-related tasks when combined with other regularization methods (Srivastava et al., 2014).

2.2.10. Normalization

Training deep neural networks can be difficult due to the distribution of the inputs to the layers in the network. During training, the parameters of the layers change, which in turn will change the distribution of their outputs. These outputs are the inputs to the subsequent layers. The change in input distribution is defined as an "internal covariate shift", which can be a problem when more layers are added to the network. This problem increases when the number of layers increases because every layer needs to adapt to new distributions for every training iteration. To combat the problem of internal covariate shift, a normalization step is introduced. In general, normalization fixes the mean and variance of an input x to the network layers by subtracting the mean μ and dividing by the standard deviation σ. x has dimensions $C \times W \times H$. The operation is shown in equation (2.12).

$$\bar{x} = \frac{x - \mu}{\sigma} \tag{2.12}$$

Batch normalization (Ioffe and Szegedy, 2015) allows for easier initialization of the network and a higher learning rate and is also beneficial for the gradient flow through the network. These advantages allow a network to be trained efficiently in fewer iterations, provide stabilization, and allow deeper networks to converge. The normalization process is done individually for each channel, where each sample will be normalized by the mean and variance of the whole batch.

An alternative to batch normalization is instance normalization (Ulyanov et al., 2016), where the mean and variance are calculated for each channel in each sample and not for whole batches. Here, the mean and variance of the rest of the samples do not affect the normalization of each sample. This method proves effective in style transfer tasks and prevents instance-specific mean and covariate shift. Instance normalization can replace batch normalization in the generator network in GANs. This removes instance-specific contrast information (range of brightness) from the image that needs to be transformed, which speeds up the generation and can provide better results.

To specifically stabilize the training of the discriminator network in GANs (Section 2.7.1), Miyato et al. (2018) introduced spectral normalization. Spectral normalization controls the Lipschitz constant of the discriminator function f by constraining the spectral norm of each weight layer. A Lipschitz constant is how much the output of a function can change with respect to changes in the input (Avant and Morgansen, 2021). The spectral norm is simply the largest singular value of a matrix which is also called the $L2$ matrix norm. The spectral norm of a weight matrix W is denoted $\sigma(W)$, and its full mathematical computation can be found in Miyato et al. (2018). By this, the spectral norm of W is normalized to satisfy the Lipschitz constraint $\sigma(W) = 1$. For a neural network, each weight layer can be normalized by equation (2.13).

$$\bar{W}_{SN}(W) := \frac{W}{\sigma(W)} \tag{2.13}$$

2.3. Representing Images

Neural networks receive vectors of numerical values as input. Together, these numerical values are representations of real-world data that the network will process. This data should be something that has a learnable pattern, and in the field of computational creativity, this data can for example be music or images. A digital image is composed of pixels in a grid, which is by default suitable for input to a neural network after the pixel values are fed into vectors to form a matrix. A small grayscale image with a resolution of 16×16 pixels can be fed into a 16×16 matrix, where each entry in the matrix is the value of a pixel. If the image has colors, it is necessary to represent the individual color values of each pixel. Each pixel often has three color values, referred to as RGB (Red Green Blue), which mixed can produce any color. The advantage of this representation is that any color can be decomposed into three discrete values only ranging from 0 to 255 (8 bits). To represent a $W \times H$ pixel RGB color image, three matrices of size $W \times H$ are needed, one for each color channel C, as seen in Figure 2.4.

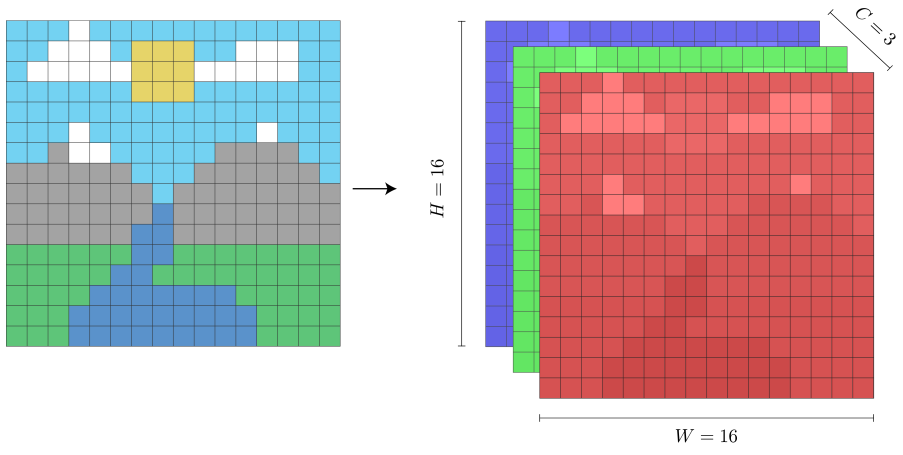


Figure 2.4.: 16×16 pixel color image converted to RGB representation for input to a neural network. The pixel grid is included for illustration.

2.4. Convolutional Neural Networks

Convolutional neural networks (CNNs) are built upon the same concepts as ANNs, in that they are comprised of neurons and that they try to learn through forward passes and back propagation. CNNs were initially designed for pattern recognition in images, and they allow for image-specific features to be encoded into the architecture (O'Shea and Nash, 2015). CNNs make strong and accurate assumptions about the contents and statistics of images and make good approximations for the relations between pixels. Compared to standard feed-forward neural networks of equal size, CNNs have fewer parameters and connections and are thus easier to train (Krizhevsky et al., 2012). Standard ANNs struggle with images because of the vast amount of weights needed for each neuron. If an image of size 64x64 pixels with 3 color channels is sent as input to a standard ANN, each neuron in the first layer would have 12,288 weights. This complexity can also lead to overfitting (O'Shea and Nash, 2015). CNNs are constructed to handle exactly this challenge, and neurons within a CNN layer will only connect to a small region of the previous layer, not the whole layer, as in standard ANNs. The neurons in the layers of a CNN are organised into three dimensions, namely the width (W) and the height (H) of the image, and the depth (D). When considering color images, the depth is referring to the number of color channels, and this extra dimension preserves the spatial integrity of the input image during computations. The main trick convolutional neural networks perform is extracting important features from the input image through convolution, such as edges of objects, and then lowering the dimensionality of the image through pooling (O'Shea and Nash, 2015).

A CNN is comprised of four major parts. The input layer, the convolutional layer, the pooling later and lastly fully-connected layers. Similar to standard ANNs, the input layers simply holds the numeric values of the input (in this case, the pixel values). The pooling layer gradually reduces the dimensionality of the image, and by this, reduces the number of parameters that need to be computed. The fully connected layers are on the same form as the ones described in Section 2.2. The convolutional layer is the core of the CNN architecture. The convolutional layer contains a kernel (also called a filter), which is typically a 3×3 matrix of numbers that sweeps across the input image and performs a two-dimensional cross-correlation operation. This computation is a series of dot products between the kernel and the pixel values of the image (Figure 2.5). The kernel matrix contains weights that will be adjusted trough learning, analogous to the weights in a standard ANN. When the kernel traverses the image, the step size is defined by the *stride*, which can be set to 1 or higher, depending on the wanted amount of overlapping. If stride > 1, it is often called a strided convolution. It is also normal to add padding of zero-pixels around the edges of the input image to not lose pixel information during the convolution process. The convolution happens for each layer in the depth dimension, which is then fed into a ReLU activation function (O'Shea and Nash, 2015).

In some cases, it can be useful to inverse the reduction process and reconstruct the spatial resolution of the original image. This is done by a fractionally strided convolution (or deconvolution/transpose convolution), which performs a regular convolution and a

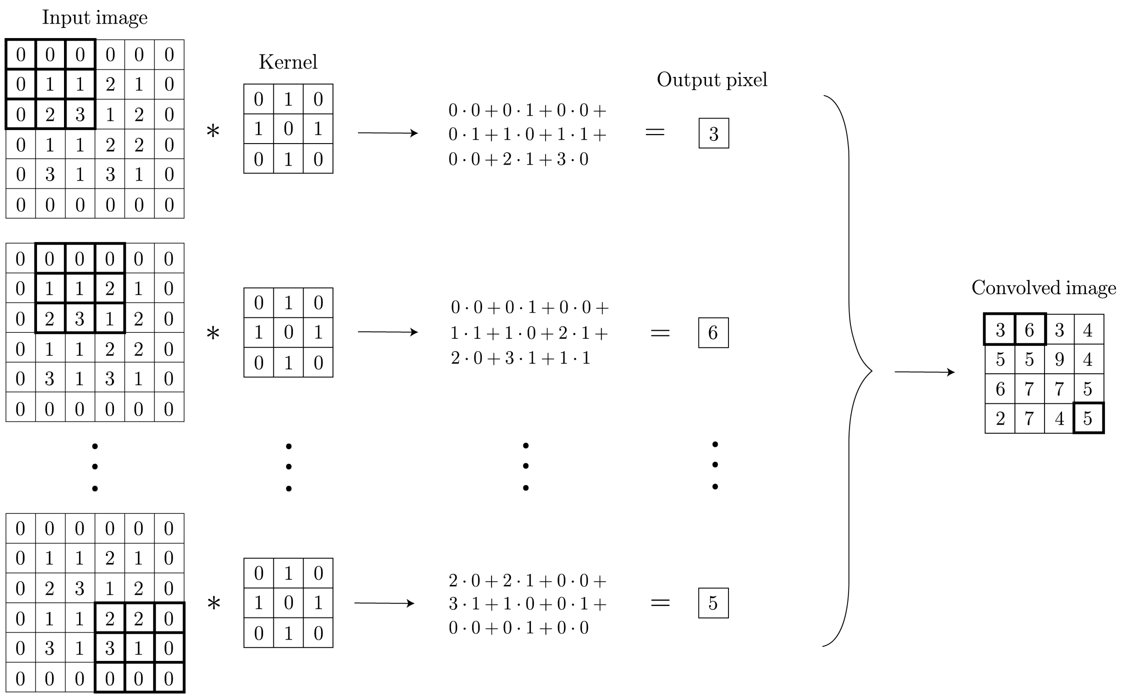

Input image
Kernel
Output pixel
$0\cdot0+0\cdot1+0\cdot0+$
$0\cdot1+1\cdot0+1\cdot1+$
$0\cdot0+2\cdot1+3\cdot0$
$= 3$
$0\cdot0+0\cdot1+0\cdot0+$
$1\cdot1+1\cdot0+2\cdot1+$
$2\cdot0+3\cdot1+1\cdot1$
$= 6$
$2\cdot0+2\cdot1+0\cdot0+$
$3\cdot1+1\cdot0+0\cdot1+$
$0\cdot0+0\cdot1+0\cdot0$
$= 5$
Convolved image

Figure 2.5.: Convolution operation on a 4x4 grayscale image with zero-padding, stride=1 and one kernel. The convolved image has a depth dimension equal to the number of kernels. The width and height is 4, since the step size is 1. For RGB images, the kernel would have a depth dimension of 3 (equal to the input image), the convolution would happen for each color channel and the convolved image would have a depth dimension of 3.

reconstruction in the same process, by adding padding between the pixels before the convolution operation.

2.5. Residual Networks

For deep convolutional neural networks, network depth (number of layers) is important in achieving good results. Deep models with a depth of up to thirty layers can perform better than smaller networks on difficult image classification tasks. However, adding more layers to a network invites several issues. One issue is vanishing/exploding gradients, which is largely addressed by normalization techniques. A second issue is the degradation problem. When the network depth increases, the accuracy of the network can degrade which leads to higher training errors. To alleviate degradation, Residual Networks, or ResNets, were introduced by He et al. (2016). The core of the ResNet is the residual block in Figure 2.6.

Compared to a regular block in a CNN, the residual block adds a skip connection. In addition to the normal forward pass of the input x through the layers, x is added to the

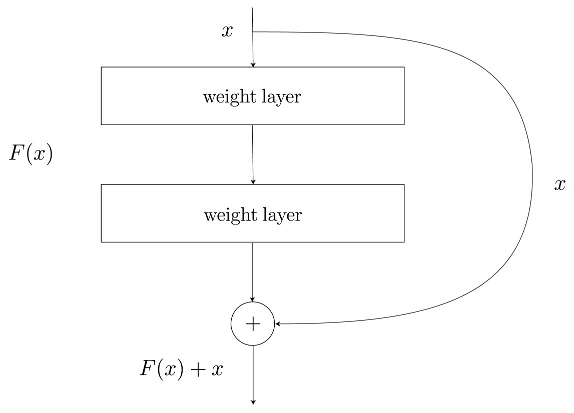


Figure 2.6.: Residual block with a skip connection. The input x of the block is added to the output $F(x)$. Adapted with permission from He et al. (2016), Copyright © 2016 IEEE.

layers' output. The output of the block is then: $F(x)+x$. This ensures that information is not lost and that it is safe to add more layers to the network. He et al. (2016) empirically shows that the addition of skip connections in a deep convolutional neural network can increase performance, without adding computational load or complexity. Instead of letting the layers learn only the underlying mapping of the data, ResNets lets the layers fit a residual mapping based on residual functions that reference the input x.

2.6. U-NET

U-NET (Ronneberger et al., 2015) is a convolutional network architecture for fast and precise segmentation of images. U-NET incorporates skip connections between its downsampling and upsampling layers to facilitate low-level information "skipping" across the network. These skip connections enable the network to keep features that otherwise could have been lost during downsampling and upsampling. The n layers form a U-shape, where layer number i will have a connection to layer $n - i$. The downsampling (or encoding) path is in the form of a typical convolutional network. The path has repeating blocks, where each block consists of two 3×3 unpadded convolutions, each followed by a ReLU and a 2×2 max pooling operation with stride 2 for downsampling. Each block doubles the number of feature channels and halves the spatial resolution. Downsampling is followed by a path of upsampling (decoding) blocks, where each block consists of a 2×2 transpose convolution, concatenation with the corresponding skip connection channels, and two 3×3 convolutions each followed by a ReLU activation. The upsampling block doubles the spatial resolution and halves the number of feature channels. U-NET can be used as the generator network in GANs (Section 2.7.1). This application, along with the skip connections and the U-shape of the network is shown in Figure 5.3 (page 46) in Chapter 5.

2.7. Generative Adversarial Networks

This section introduces Generative Adversarial Networks (GANs), and present several GAN variations, including DCGAN, LSGAN, SAGAN, PatchGAN and CycleGAN. This section contains paragraphs and figures that are inspirations and revisions of work from the specialization project. This applies to some of the explanations of GAN (Section 2.7.1) and CycleGAN (Section 2.7.7). Specifically, Equations 2.14, 2.17, 2.18, 2.20 and Algorithm 1 are copied directly, along with some of their explanations.

2.7.1. GAN

As mentioned in Section 2.2, generative models are machine learning models that try to learn the underlying probability distributions $p(X)$ for unlabeled datasets. Historically, these probability distributions have been challenging to approximate by generative models. Goodfellow et al. (2014) introduced adversarial neural networks, where a generative model competes with a discriminative model. The two models combine into the Generative Adversarial Network (GAN), which avoids many of the difficulties of learning $p(X)$. In GANs, the generative and discriminate models are called the generator and the discriminator, respectively. The generator creates new, previously unseen samples derived from the learned distribution $p(X)$. The discriminator learns to determine if the samples are from the real dataset or if they are generated. The generator tries to fool the discriminator by creating samples that are more and more indistinguishable from the real samples. Both models are neural networks trained using backpropagation algorithms.

Goodfellow et al. provided a suitable analogy to GANs: "The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles".

G (the generator network in Figure 2.7) takes a noise vector z from a standard normal distribution p_z as input. G then generates a fake sample $G(z)$, which in turn becomes an input to the discriminator network D. D takes either a real (x) or a generated $(G(z))$ image as input and produces a single scalar $D(x)$ or $D(G(z))$, respectively. x derives from the real distribution $p_{data}(x)$. The output scalar is the probability of the input being real or fake. D tries to guess correctly by maximizing the value of $D(x)$ being 1 (real) and $D(G(z))$ being 0 (fake). Meanwhile, G wants to maximize $D(G(z))$ being 1. In other words, the discriminator is trained to maximize the probability of assigning the correct label to training examples from the real world and samples created by the generator. The generator is trained to minimize the probability of the generator assigning the correct label. This results in the minimax objective function (Equation 2.14), which is used during backpropagation for both networks. Equation 2.14 is the average log probability of $D(x)$ plus the average log probability of $D(z)$. This loss function is analogous to the BCE loss (Equation 2.4). The output of the discriminator goes through a sigmoid activation function which works well as input to the BCE loss function.

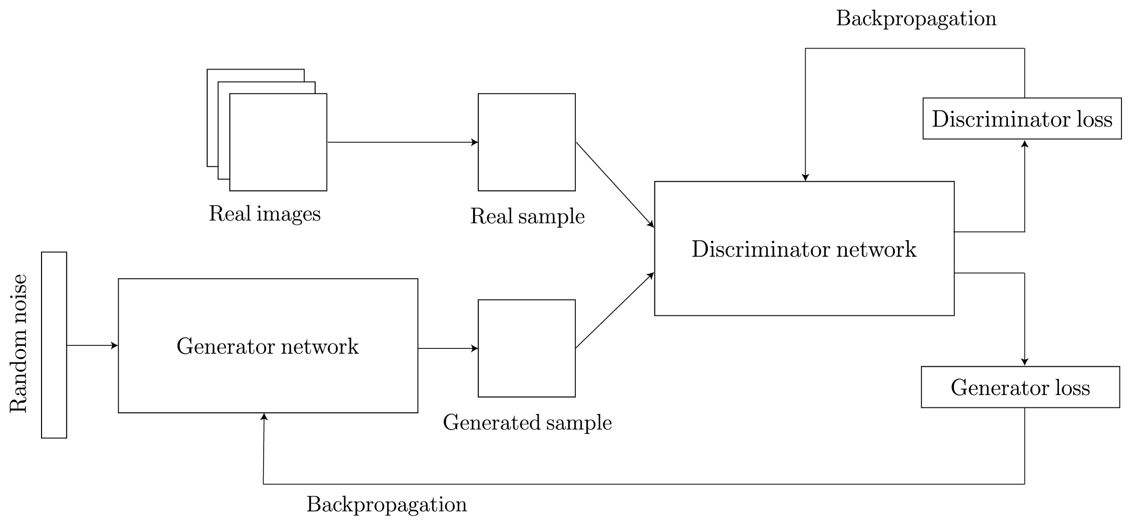


Figure 2.7.: GAN architecture. The generator inputs a random noise vector and produces a fake image. The discriminator takes an image as input and decides whether it is real or fake. The generator learns by maximizing the discriminator's loss, and the discriminator learns by minimizing its loss, both through back-propagation. The generator wants to learn $p(X)$ without having direct access to X but implicitly through gradients flowing through the discriminator. In this case, GAN is used to generate images. GAN figure inspired by Google (2022a), under license CC BY 4.0.

.

(2.14) Minimax objective function for GANs.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x))}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z)))] \tag{2.14}$$

To train G and D, the GAN model uses Algorithm 1. In the inner for loop, the discriminator's parameters (θ_d) are updated by calculating the gradient (∇) of the loss function with respect to θ_d. For the discriminator, the gradient is maximized, so gradient ascent is used. In the outer loop, the gradient of the loss function with respect to the generator's parameters (θ_g) is calculated. Here, the gradient is minimized, so gradient descent is used to update θ_g. Note that during the training of the generator, the discriminator remains constant, and during the training of the discriminator, the generator remains constant. The objective function (Equation 2.14) is used in both loops in Algorithm 1, and for each training iteration, m samples are sampled from the real and the noise distributions. The discriminator penalizes the generator if it labels the generated sample as fake.

Algorithm 1 GAN training algorithm with stochastic gradient descent.

```
for number of training iterations do
    for $k$ steps do
        - Sample $m$ noise samples $z^{(1)}, ..., z^{(m)}$ from $p_g(z)$
        - Sample $m$ real samples $x^{(1)}, ..., x^{(m)}$ from $p_{data}(x)$
        - Update the discriminator by ascending its stochastic gradient:
            $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [log D(x^{(i)}) + log(1 - D(G(z^{(i)})))]$
    end for
    - Sample $m$ noise samples $z^{(1)}, ..., z^{(m)}$ from $p_g(z)$
    - Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} log(1 - D(G(z^{(i)})))$
end for
```

2.7.2. Failure Modes

The GAN minimax game should, in theory, result in the discriminator being unable to differentiate between the real samples and the generated ones because the generator has successfully learned $p(x)$. In practice, the model does not always converge to this point. The convergence of GANs is still being actively researched, and it is generally known that GANs can be hard to train and that gradient descent-based optimization often does not lead to convergence (Mescheder et al., 2018).

An example of non-convergence is if the loss function does not provide sufficient gradient for the generator to learn fast enough, and the discriminator "wins" (Goodfellow et al., 2014). This is known as vanishing gradients (or the saturation problem) and is caused by the discriminator being too good with respect to the generator. Another example is if the training continues past the point of convergence and the discriminator does not give any valuable feedback to the generator. Then, the generator might adjust its parameters based on bad feedback, and the quality of the generated images may decrease.

Another common GAN failure is mode collapse. This happens when the current generator optimizes its output according to the current discriminator. In other words, if the generator starts to produce only a small set of outputs, the discriminator learns to always reject those outputs. This limited strategy can cause the discriminator to get stuck in a local minimum (Google, 2022b). Then the next iteration of the generator can simply create a different set of outputs that always trick the discriminator, with the limited set of generated samples not necessarily being close to the real images. One way to identify mode collapse is to visually check the generated images for repeated patterns and objects between them.

Because of the different failure modes, the original GAN algorithm has seen several improvements. One improvement is using other loss functions for G and D. An example is in WGAN (Arjovsky et al., 2017), where the discriminator model is changed to a critic that scores the realness or fakeness of a given sample instead of only a probability. One of the goals of WGANs is to prevent vanishing gradients which is a common problem for GANs.

2.7.3. DCGAN

One of the most significant GAN improvements is the Deep Convolutional GAN (DCGAN) (Radford et al., 2015), where CNNs are used for more stable training in the networks. DCGAN provides good representations of images for supervised learning and generative modeling while reducing the complexity of the networks. DCGAN makes three significant changes from standard CNNs. The first change is to replace pooling layers with strided convolutions for the discriminator and fractionally-strided convolutions for the generator. The second is the removal of fully connected hidden layers for deeper architectures, and the third is batch normalization. Batch normalization helps the generators to begin learning by preventing them from collapsing all samples to a single point (mode collapse). Batch normalization is applied to all layers in both networks, except for the generator's output and the discriminator's input, since this can result in more model instability. In addition, the ReLU activation function is used in the generator for all layers except for the output, which uses Tanh (Figure 2.2 (d), page 11). Tanh is used to make the model quickly learn the color space of the training distribution. LeakyReLU (Figure 2.2 (c), page 11) is used for all layers in the discriminator, which works well for higher-resolution images (Radford et al., 2015).

2.7.4. Least Squares GAN

The discriminator in regular GANs is a binary classifier, which places images on either side of a decision boundary, depending on whether they are deemed real or fake. When updating the generator in regular GANs, the sigmoid cross entropy loss function will cause the problem of vanishing gradients for the samples that are on the correct side of the decision boundary but are still far from the real data (Mao et al., 2017). This loss can be replaced by a least squares loss function, which may lead to more stable training and higher-quality generated images. This least squares loss will penalize generated samples that lie far from the decision boundary, leading to the generator generating samples closer to the real data. Since samples that are a long way from the decision boundary are penalized, more gradients can be generated, which can alleviate the issue of vanishing gradients. For the discriminator and the generator, the new least squares loss functions are defined in Equations 2.15 and 2.16, where the discriminator wants $D(x)$ to be as close to 1 (real label) and $D(G(z)$ to be as close to 0 (fake label) as possible. The generator wants $(D(G(z)))$ to be as close to 1 as possible.

$$\min_D V_{LSGAN}(D) = \frac{1}{2}\mathbb{E}_{x\sim p_{data}(x))}[(D(x)-1)^2] + \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[(D(G(z)))^2] \tag{2.15}$$

$$\min_G V_{LSGAN}(G) = \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[(D(G(z))-1)^2] \tag{2.16}$$

2.7.5. Self-Attention GAN

Traditional convolutional GANs generate higher-resolution details as a function of only spatially local points in lower-resolution feature maps. Zhang et al. (2019) introduced Self-Attention GAN (SAGAN), which uses an approach that enables the generation of details by incorporating information from all feature locations. Furthermore, the discriminator in SAGAN can verify the consistency of highly detailed features across distant parts of the image. This entails that, for example, a feature in the top left of an image can be ensured to be consistent with a feature in the bottom right of an image. SAGAN works by adding a self-attention module that calculates attention weights for each feature map by looking at many spatial positions, which signify the importance of the positions. This is done by transforming the self-attention layer's input features and generating three feature matrices which are multiplied together. The Self-Attention module is further explained in Section 5.5.

2.7.6. PatchGAN

In the original GAN architecture, the discriminator predicts an image as real or fake with a single scalar by considering the entire image simultaneously. Isola et al. (2017) introduced PatchGAN, a CNN-based discriminator. This architecture was adopted from Li and Wand, which originally wanted to capture local style statistics. PatchGAN maps the input image to an $N \times N$ array of outputs X, where X_{ij} indicates whether patch ij is real or fake. This enables the architecture to penalize the structure of smaller image patches at a time. The discriminator convolution operation is run across all patches, averaging all the X_{ij}s and producing a final scalar D, which accounts for the entire image. Mathematically, the convolutions over all patches are identical to dividing the image into patches, manually feeding the patches into a regular discriminator one at a time, and then averaging the outputs. However, PatchGAN is advantageous to this manual operation since it requires fewer parameters, runs faster, and is scalable to large images. Intuitively, PatchGAN can be understood as a type of texture/style loss (Isola et al., 2017).

2.7.7. CycleGAN

After the DCGAN was introduced, the use of convolutions in GANs became standard, as they performed well in generating samples. The traditional GAN models synthesize new samples from random noise based on the distribution of the training data. Another application of GANs is to translate an image from one domain to another, also called image-to-image translation or style transfer. A domain is a set of images with characteristics in common, e.g. high-resolution photos, Monet paintings, hand-drawn sketches or cartoons. An image-to-image translation model takes an image from a source domain and transfers the contents of the image into the style of the target domain. This technique can for example convert sketches into photo-realistic images, and vice versa.

Isola et al. (2017) introduced the pix2pix model for image-to-image translation, which requires a training set of aligned image pairs from the two domains. The pix2pix model has one generator and one discriminator, where the networks are conditioned on class labels. The generator tries to transform the image, and the networks are trained with an additional loss function, pixel-wise loss, in combination with the min-max loss function (Equation 2.20). This loss function compares the generated image with its actual labeled counterpart from the target domain.

For many domains, especially in visual art, it can be difficult to find pairs of images. If the style transfer task were to be from pictures of people to paintings of people, a training pair should preferably be a painting and a picture of the same person with the same pose in the same environment. A painting of a landscape and a picture of a landscape should preferably be in the same location at the same time of day. In short, the layout of both images in the pair should closely resemble each other, despite being of different styles.

CycleGAN, introduced by Zhu et al. (2017), was designed specifically to achieve image-to-image translation for unpaired, unlabelled sets of images. This alleviates the need for collecting and labeling image pairs for the training data and only requires sets of images from the two different domains. Where regular GAN architectures normally consist of one generator and one discriminator, CycleGAN uses two generators and two discriminators, to generate and discriminate samples for both domains. The first generator translates images from the source domain to the target domain, while the second generator translates the generated images back to the source domain. The recreated image is compared to its original, to ensure that the generator does not disregard important features and still encompasses the same structure of the objects in the source image. The corresponding discriminators classify the generated images as being real or fake for the two domains. The generators and discriminators and their connections are shown in Figure 2.8.

CycleGAN firstly wants to create a mapping $G : A \rightarrow B$ where G is the generator, A is the source domain and B is the target domain. a is a sample image from domain A and b from domain B. The goal is to make $G(a)$ indistinguishable from b. To achieve this, CycleGAN uses the principle of cycle consistency. Cycle consistency can be thought of as if you translate a sentence from one language to another and translate it back, you should arrive back to the original sentence. Applied to GAN, this means that another mapping is created: $F : B \rightarrow A$, such that generators G and F are inverses of each other. CycleGAN trains G and F simultaneously, and adds a loss function that encourages forward ($a \rightarrow G(a) \rightarrow F(G(a)) \approx a$) and backward ($b \rightarrow F(b) \rightarrow G(F(b)) \approx b$) cycle consistency. The cycle consistency loss is defined as:

$$\mathcal{L}_{cyc}(G,F) = \mathbb{E}_{a\sim p_{data}(a)}[||F(G(a)) - a||_1] + \mathbb{E}_{b\sim p_{data}(b)}[||G(F(b)) - b||_1] \tag{2.17}$$

In the two terms on the right side of Equation 2.17, the images a and b are being translated back and forth through G and F, and then compared with themselves, to measure the forward and backward cycle consistency. Cycle consistency loss further reduces the space of possible mapping functions, such that the generators become more restricted and are able to learn the correct mapping. CycleGAN combines the cycle

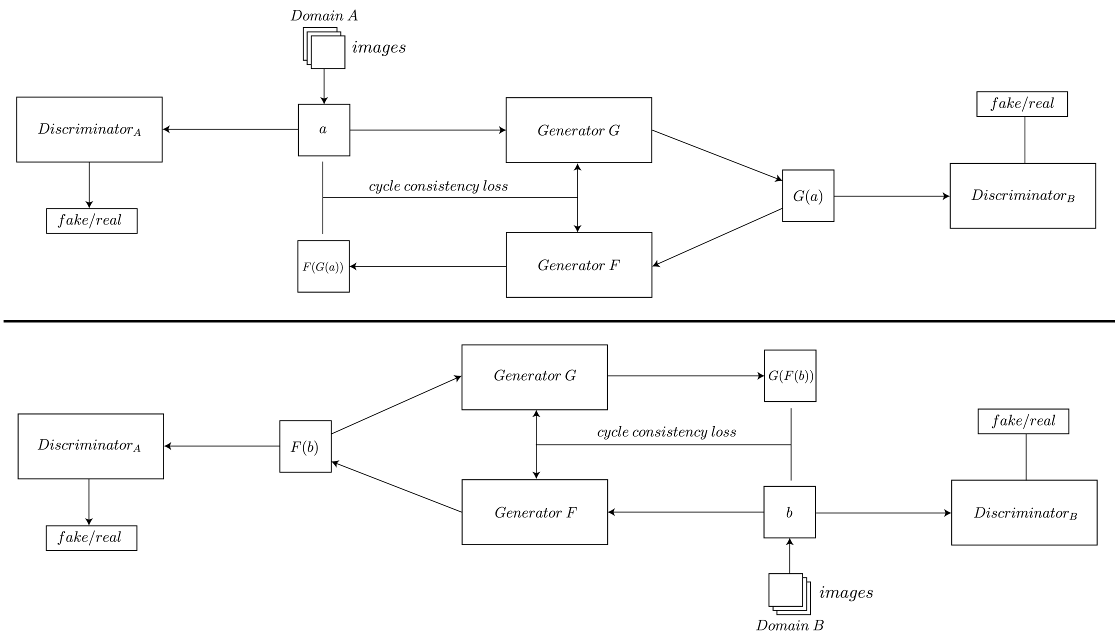


Figure 2.8.: Simplified CycleGAN model structure. Adapted from Bansal and Rathore (2017) with permission from author.

consistency loss with the adversarial loss from GAN (Section 2.7.1). Applying the adversarial loss function to the mapping function of generator G and discriminator D_B results in:

$$\min_{G}\max_{D_B}\mathcal{L}_{GAN}(G, D_B, A, B) = \mathbb{E}_{b\sim p_{data}(b))}[logD_B(b)] + \mathbb{E}_{a\sim p_{data}(a))}[log(1-D_B(G(a)))] \tag{2.18}$$

where $G(A)$ generated images resembling images from domain B, and D_B tries to discriminate between the generated images and real images from B. The same principle is applied to generator F and discriminator D_A, resulting in:

$$\min_{F}\max_{D_A}\mathcal{L}_{GAN}(F, D_A, B, A) = \mathbb{E}_{a\sim p_{data}(a))}[logD_A(a)] + \mathbb{E}_{b\sim p_{data}(b))}[log(1-D_A(G(b)))] \tag{2.19}$$

Combining cycle consistency loss and the adversarial losses, the full objective function for CycleGAN is achieved:

$$\mathcal{L}(G, F, D_A, D_B) = \mathcal{L}_{GAN}(G, D_B, A, B) + \mathcal{L}_{GAN}(F, D_A, B, A) + \lambda\mathcal{L}_{cyc}(G, F) \tag{2.20}$$

where λ controls the relative importance of the two objectives. Zhu et al. (2017) recommend $\lambda = 10$. CycleGAN wants to minimize the two generators G and F, and maximize the two discriminators D_A and D_B, shown by Equation 2.21, where G^* and F^* are the learned mappings.

$$G^*, F^* = \arg\min_{G,F} \max_{D_A, D_B} \mathcal{L}(G, F, D_A, D_B) \tag{2.21}$$

In short, the adversarial losses help with matching the distribution of generated images to the data in the target domain, while the cycle consistency losses help to prevent the learned mappings G and F from contradicting each other.

In the actual implementation by Zhu et al. (2017), the generator architecture in CycleGAN consists of three convolutions, two fractionally-strided convolutions with stride set to $\frac{1}{2}$, and one convolution that maps features to RGB colors. The generator has several residual blocks with the number depending on the resolution of the training images. In addition, instance normalization is used. The discriminator networks are PatchGANs (Section 2.7.1) of size 70×70, which try to classify overlapping 70×70 image patches individually. Mathematically, this results in the output of the generator being a 30×30 matrix. The receptive field of 70×70 is determined by the depth of the PatchGAN, and the authors experienced the best results for 256x256 images with this choice. For the final layer of the discriminator, the authors remove the sigmoid activation function which often is present in GANs. During the training of the model, the authors alter the adversarial loss function (Equation 2.14) by replacing the negative log-likelihood objective (sigmoid cross entropy) with a least squares loss, which results in the generators trying to minimize Equation 2.22 and the discriminators minimizing Equation 2.23.

$$\mathbb{E}_{a \sim p_{data}(a))}[(D(G(a)) - 1)^2] \tag{2.22}$$

$$\mathbb{E}_{b \sim p_{data}(b))}[(D(b) - 1)^2] + \mathbb{E}_{a \sim p_{data}(a))}[D(G(a))^2] \tag{2.23}$$

In addition, the discriminator is trained by looking at a history of previously generated images, and not only the images generated by the latest generator, by having an image buffer that stores the 50 previously created images. This technique was first introduced by Shrivastava et al. (2017), where the authors discuss that it improves training stability.

CycleGAN can be used for many different image-to-image translation tasks. Figure 2.9 shows examples of translating horses to zebras and apples to oranges.

2.7.8. GAN Evaluation Metrics

For regular regression and classification tasks, the quantitative performance of machine learning models can be evaluated in several ways. The accuracy is often measured by how many samples in the test set the model correctly predicted divided by the total number of predictions. This, however, requires data with corresponding labels where the ground truth is known. This method of measuring accuracy could apply to the discriminator network in GANs, but it is often not of interest to measure the discriminator after training. The interesting part is measuring the performance of the generated images by the generator. This, however, is not a trivial task since the visual quality of a generated

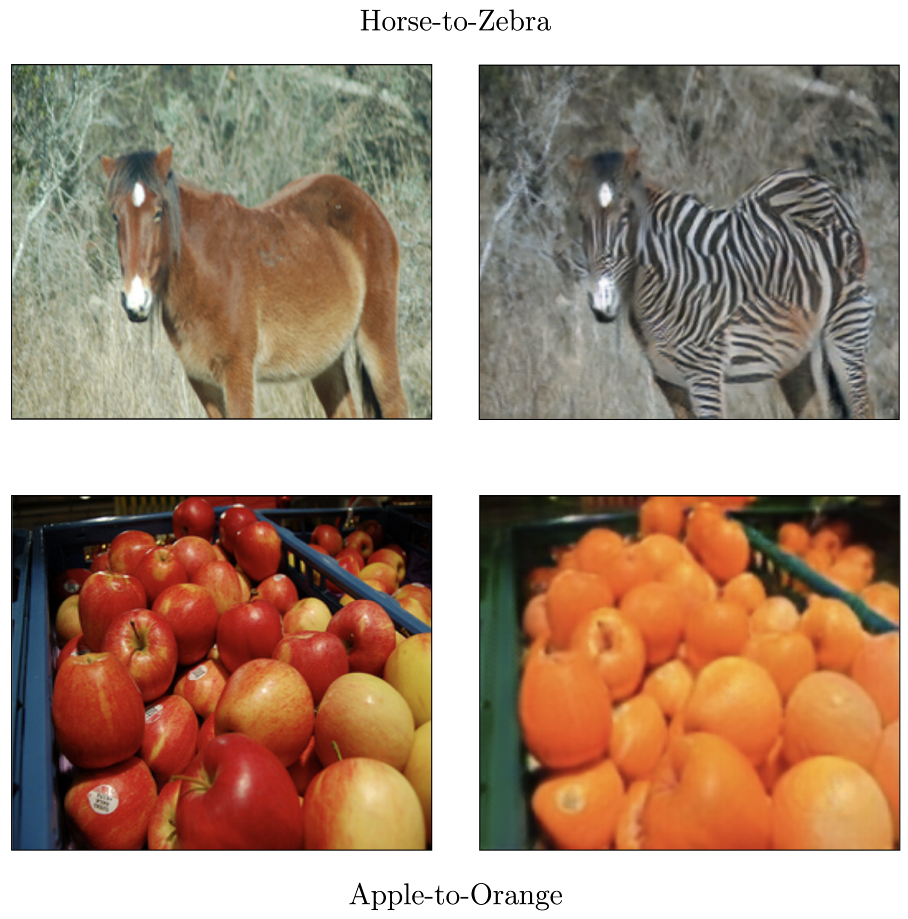


Figure 2.9.: Some applications of the CycleGAN model. Reprinted with permission from Zhu et al. (2017), Copyright © 2017, IEEE.

image is highly subjective and difficult to capture mathematically. Metrics such as per-pixel mean-squared error do not assess joint statistics of the generated images and therefore do not measure the overall structure (Isola et al., 2017). Because of this, several quantitative evaluation metrics have been developed, mainly to assess the quality and the diversity of generated images in a perceptual manner without human subjectivity. Many such metrics exist, but for image-to-image translation models, examples of applicable ones are IS, FID, and SSIM.

IS

Salimans et al. (2016) introduced the Inception Score (IS), which uses a pre-trained network (Inception model, Szegedy et al. (2016)) to get the conditional label distribution $p(y|x)$ for every generated image x. This label distribution is used to compute the probability of correct class assignments to quantify quality. The integral of the marginal distribution of the generated samples is also captured to measure diversity. The authors show that IS correlates well with human judgment but does not capture how the generated images compare to real images of the same domain.

FID

Heusel et al. (2017) introduced Fréchet Inception Distance (FID) which is an improvement to IS. FID was introduced to quantify how the distribution of generated images compares to the distribution of real images of the same domain. Similarly to IS, FID uses a pre-trained Inception model (Szegedy et al., 2016). FID compares the mean and covariance of the last pooling layer of the Inception network. In other words, the output of the network of the final classification layer is removed. This is done to extract high-dimensional features from images to capture characteristics. The extracted feature distributions from the two sets of images are compared by measuring the Fréchet distance between them. This distance is computed into a single scalar score, and a lower FID score implies a higher-quality image. Mathematically, FID is given by: $\text{FID} = |\mu_r - \mu_g| + tr(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{\frac{1}{2}})$, where μ_r and μ_g are the means and Σ_r and Σ_g are the covariances of the extracted features of the real and generated images, respectively. tr is the resulting matrix's trace (sum of the diagonal elements).

The authors show that FID also correlates well with human judgment and is more consistent in dealing with image disturbances than IS (i.e., the more they disturb or distort an image, the higher the FID score). FID can also capture small changes, such as slight blurring or small artifacts (Borji, 2022). FID is popularly used to evaluate many GAN models, such as the state-of-the-art StyleGAN models (Karras et al., 2020), and is, together with IS, the most popular GAN evaluation metrics (Borji, 2022). A minimum sample size of 10,000 is often used to compute FID (Chong and Forsyth, 2020).

KID

FID assumes that the features extracted from the image sets have a normal distribution. Bińkowski et al. (2018) proposed Kernel Inception Distance (KID) which aims to improve FID by relaxing the normal distribution assumption. KID uses a polynomial kernel to measure the squared Maximum Mean Discrepancy (MMD) between the extracted features (Betzalel et al., 2022). MMD is a distance measure between probability distributions of features, and a polynomial kernel is a function to transform data (Yang et al., 2019). KID requires fewer data samples than FID since it does not need to compute the quadratic covariance matrix.

SSIM

Wang et al. (2004) developed the Structural Similarity index (SSIM) to objectively assess the difference in perceptual image quality between two images. SSIM measures visible structures in images by three factors; luminance, contrast, and structure. Luminance is a measure of the average brightness, quantified to be the average value of all the pixels in an image. Contrast is the difference between the light and dark parts of an image, which is measured by the standard deviation of the pixels. The structure is intuitively the shapes in an image and is quantified by measuring the correlation between local structures in images. These three measures are combined into a single scalar SSIM score, with each measure having its own weight. The complete mathematical calculations can be found in Wang et al. (2004). SSIM serves as a popular method of measuring perceptual structure in images.

Human Evaluation

When evaluating the performance of GANs, more than one metric is often used since each metric has its applications and limitations. To further evaluate the generated images, researchers may create user studies. Denton et al. (2015) asked 15 people to distinguish real samples from fake samples. In their experiment, the users only had limited viewing time for each sample. They collected information about a total of 10k generated and real samples from various GAN models and calculated the percent of generated samples that fooled the participants as a function of time spent looking at the photos.

Salimans et al. (2016) performed a similar task of having humans distinguish between real and fake samples. They automated the process using Amazon Mechanical Turk (MTurk[2]), a crowdsourcing marketplace for human insight. Isola et al. (2017) and Zhu et al. (2017) also used MTurk for “real vs. fake” perceptual human studies. For each model, Isola et al. (2017) showed each image for 1 second, and after 10 images, the 50 participants were given feedback. For the 40 rest of the images, no feedback was given. Zhu et al. (2017) employed the same study with only 25 participants per model.

Elgammal et al. (2017) created a GAN model for generating art that they deemed “creative” compared to human creativity. To validate their model, they conducted three experiments in the form of questionnaires, which they sent out through MTurk. This was to compare the responses of humans to the generated art with the responses to art created by artists. Their first experiment asked if the sample was real or fake and further asked to rate the realness on a scale from 1 to 5. The following experiments asked several domain-related questions about the images regarding aesthetics, complexity, emotions, and more. Finally, they conducted a survey with a pool of art history students to gather responses from domain experts.

[2]https://www.mturk.com

3. Related Work

In this chapter, related work in the field of pixel art synthesis is introduced. To motivate the need for deep learning methods, insights from natural computational methods will be presented. In the remainder of the chapter, efforts toward image-to-image pixelization using GANs will be introduced, along with the datasets used. The presentation of image-to-image pixelization efforts and their datasets contain revisions of work from the specialization project.

3.1. Natural Computation

Many efforts have been made toward the automatic generation of pixel art, most of which are natural computation methods (Shang and Wong, 2021). These methods include image downscaling (Öztireli and Gross, 2015; Kopf et al., 2013) and other optimization techniques (Gerstner et al., 2012; Inglis and Kaplan, 2012). Downscaling methods do well in converting lines and curves to pixelated lines and curves and ensuring color retention, but they struggle with intricate shapes. The optimization techniques further assess the stylistic demands of pixel art by bettering the mapping of features, reducing color palettes, and focusing on sharp edges. The results from these techniques correlate well with manual pixel art, but improvements are needed to deal with stylistic demands like dithering and abstraction. Furthermore, Inglis and Kaplan do not provide any results where the methods are tested on domains other than vector line art, which may indicate limitations. Gerstner et al. and Shang and Wong further developed optimization techniques mainly for the pixelization of photographic portraits. Both of these algorithms focused on a reduced color palette and the preservation of facial features. They provide good results for portraits and other photographic images but with limited adaptability to other domains, as well as not having precise edges around objects. In addition, Shang and Wong experienced issues with dominant colors in the source image, affecting the color distribution of the output.

The common factor of the methods mentioned above is that they do not learn directly from the artistic style of pixel art but rely on hand-crafted computational rules. They may not be able to consider all the stylistic conventions of pixel art, which can result in restricted adaptability to different input domains and a lack of aesthetics, creativity, and artistry. These findings motivate using deep learning and image-to-image translation models to synthesize pixel art since they are shown to translate artistic style between domains effectively.

3.2. Paired Image-to-Image Pixelization

With the pix2pix architecture as a basis, Coutinho and Chaimowicz (2022) generated pixel art character sprites. The goal was to make assets for video games by generating different poses (e.g., left, right, or back facing) from a source pose (e.g., front-facing). This translation was not from cartoon to pixel art but from pixel art to pixel art. From previous results, the authors show that the base pix2pix model is not capable of translating pixel art sprite poses because of the characteristics of pixel art (such as limited colors) and the lack of large datasets. To further develop the model, the authors introduced a histogram loss term, which they experienced led to slightly better results. This histogram loss term penalizes generated images with different color palettes than their source inputs. This is done by extracting the input and output color histograms and measuring the difference between them. The authors evaluated their model by visual inspection, FID score (Section 2.7.8), and the L1 distance (Section 2.2.3) between the generated images and their labeled targets.

Even though the results were only marginally better, the histogram loss can be applied to domain transfer tasks where color preservation is important, such as cartoon-to-pixel art. Another relevant part of this work is their overview of characteristics that need to be considered when creating pixel art regarding color palette, shading, and texturing. This overview coincides with the stylistic rules presented in Section 1.1.

Coutinho and Chaimowicz (2022) used a small dataset of 294 pixel art character sprites, where each character had four poses with a uniform art style. The authors state that no large datasets of pixel art are openly available, which presents a challenge for style-transferring pixel art. They trained their model with a train/test dataset split of 85/15. During training, data augmentations of hue rotation and character offset were applied to the images.

3.3. Unpaired Image-to-Image Pixelization

A reoccurring challenge with synthesizing pixel art is the limited availability of suitable datasets, particularly paired ones. To circumvent this issue, unpaired image-to-image translation models can be applied. Kuang et al. (2021) and Yang et al. (2022) apply cycle-consistent models with GANs to synthesize pixel art. Their results vary in quality but show that such models can learn and transfer the pixel art style with limited training data, which motivates further research.

With CycleGAN (Section 2.7.7) as a basis, Kuang et al. (2021) generated pixel art from cartoon images. The authors' goal was to retain the characteristics of pixel art, such as clear contours and jagged edges in their generated images. They proposed a new generator for CycleGAN and adopted a nested U-NET structure called U-Net++ (Zhou et al., 2018). The switch from ResNet (Section 2.5) to U-NET (Section 2.6) was motivated by the mainstream use of U-Net in medical image segmentation, where the connection structure provides more precise segmentation. The authors believe that the multi-scale feature fusion in U-NET can restore contour information when generating pixel art. They state that U-Net++ can effectively improve the sensitivity to the edge information of large objects and the perception of small objects, which can help with the edges of the pixel objects. They further changed the network architecture by replacing ReLU functions with LeakyRelu (Section 2.2.2) during downsampling. Another addition was a topology-aware loss, where they extracted features of some layers from a pre-trained VGG16 network (Simonyan and Zisserman, 2014). This was applied to cartoon images before and after the network loop, but they do not mention whether they applied it to the pixel images. Despite this, they stated that the added loss further preserved the linear topology of pixel images while ensuring that the input and output were connected in a meaningful way. The discriminator remained unchanged from the original CycleGAN, along with the use of losses (cycle consistency, adversarial, and identity). To evaluate their model, they compared their results to the base CycleGAN model, shown in Figure 3.1.

The suggested architecture was able to transfer the pixel art style to cartoons, with results shown in Figure 3.1. The model showed improvements over the original CycleGAN model, but it still had some artifacts and limitations. In image (a) of Figure 3.1, yellow and gray artifacts can be seen. In addition, clear black lines in the original image are not translated to clear black pixels. In all three images, the individual "pixels" are of different shapes, and there are many occurrences of each pixel not being one uniform color. On closer inspection, none of the images have clear and sharp edges, and frequent blurring occurs. Despite the results, U-NET and topology-aware losses are reasonable model extensions to be tested in this thesis. The reasoning is that the techniques and extensions presented were explicitly applied to work with cartoon-to-pixel art style transfer and may work well with another dataset and in combination with other model extensions.

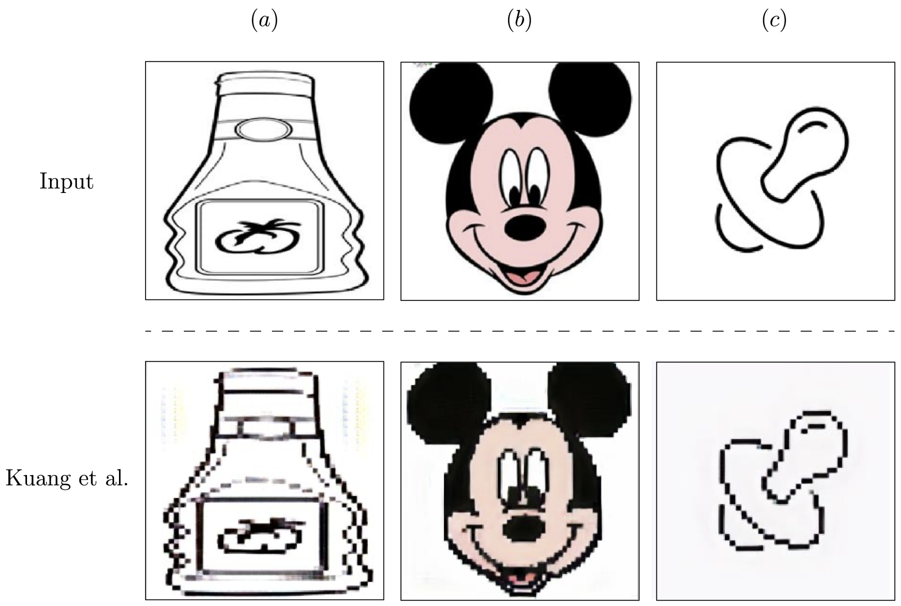


Figure 3.1.: Results from Kuang et al. (2021). Reprinted with permission, Copyright © 2021, IEEE.

Similar to the work in this thesis, Kuang et al. (2021) used unpaired cartoon and pixel art images as training data. The authors used a dataset consisting of 900 cartoon images and 900 pixel art images collected from the internet. They do not state their data sources, nor if the images are under any creative commons license or in the public domain. The authors show examples from the dataset (Figure 3.2), and it is apparent that many of the images from the pixel domain are of poor quality. One example of inadequate quality is that the images inhibit different pixel sizes. That is, one "pixel" in one image is not the same size as a "pixel" in another, despite the images being of the same spatial resolution. Furthermore, many images have blurred edges and few discernable pixels. These images do have a "blocky" art style but do not conform to the pixel art style. Another flaw is that many of the images have a visible pixel grid. A pixel grid is often used as a guide when manually creating pixel art but should not be included in the final product. All of these artifacts and irregularities will act as noise, and it can be difficult for a deep learning model to learn from a dataset that does not represent the true distribution of the target domain.

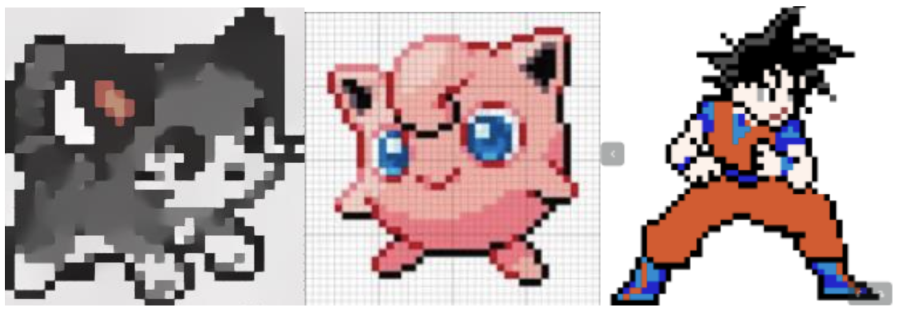


Figure 3.2.: Examples of the dataset from Kuang et al. (2021). Reprinted with permission, Copyright © 2021, IEEE.

Yang et al. (2022) used CycleGAN as a base model for synthesizing pixel art from photographic images. The authors extended the model by adding a shared latent space assumption, as presented in UNIT (Liu et al., 2017). Shared latent space can be thought of as two domains having points similar to each other in an encoded representation. This assumption is needed to be able to learn the joint distribution between the two domains (Liu et al., 2017). Yang et al. further added a cycle consistency loss between the latent space blocks of the generators and argued that it helped stabilize the model. In addition, they added Gaussian noise to the generators by concatenating noise vectors to each residual block. This method was used in StyleGAN (Karras et al., 2019) and was added to help the model generate random patterns. They evaluated their model only by visual inspection.

The resulting images do not conform well with the artistic style of pixel art. It is not easy to discern individual pixels, and there are few sharp edges. In addition, the results show white spot artifacts, which are not present in the input image. Despite the varying results, the idea of shared latent space could be beneficial to learn the one-to-one mapping between the cartoon domain and the pixel art domain. It can be applied to the model if the experiments show difficulties in learning mapping. Noise concatenation might not be applicable since random patterns might interfere with the goal of keeping all objects and colors of the original image.

Yang et al. (2022) collected a dataset of 500 pixel art images from game asset websites and pixel art forums and 4300 photographic images from the Kaggle online database[1]. During data collection, they only kept images of landscapes and nature scenes and filtered out very bright and very dark images. As a preprocessing step, they resized their image to the same aspect ratio. To not distort images, they scaled their images while preserving the aspect ratio and randomly cropped a square region during training, which resulted in a final resolution of 256×256. The authors present some examples of their dataset. Prior to scaling, their pixel images are of different resolutions, which dictates several artistic styles. This may cause the model not to produce images with "pixels" of varying sizes, as some pixel art styles contain more pixels than others.

[1]https://www.kaggle.com/datasets

4. Datasets

This chapter introduces the acquisition and processing of a custom dataset. The dataset is composed of images from the pixel art domain and the cartoon domain. The resources used for obtaining pixel art were first introduced in the specialization project.

No large datasets of high-quality pixel art are currently publicly available. The majority of pixel art present online is copyrighted, many of them being trademarked game characters and other licensed works of pixel artists. It would take great effort to collect from all the various sources and obtain permission to use the artwork. This motivates to collect artwork from resources where large quantities of pixel art are available and are open-source under a creative commons license.

4.1. Data Sources

One of the main motivations behind CycleGAN was to translate between domains without paired images since obtaining paired data can be difficult and expensive. Since Zhu et al. (2017) significantly reduced the effort required to collect a dataset, assembling a sizable high-quality pixel art dataset becomes feasible. To assemble a dataset, one can either manually download each image or use web-scarping techniques on open-source online image databases. eBoy.com[1] provides thousands of pixel artworks of people, vehicles, buildings, and other objects, with several distinctive styles. OpenGameArt.org[2] provides video game pixel art sprites for characters, items, and levels, among other things. These two resources provide artwork under the creative commons licenses CC BY-NC-ND 4.0[3] and CC0[4]. For the cartoon domain, publicdomainvectors.com[5] provide thousands of vector images under the public domain (CC0). These images are diverse in the use of colors and their art style. In addition, Google Research provides a collection of 100k 2D cartoon avatar images with different colors and proportions. The dataset is called *Cartoon Set*[6] and is available under CC BY-SA 4.0[7].

[1]https://hello.eboy.com/
[2]https://opengameart.org
[3]https://creativecommons.org/licenses/by-nc-nd/4.0/
[4]https://creativecommons.org/share-your-work/public-domain/cc0/
[5]https://publicdomainvectors.org
[6]https://research.google/resources/datasets/cartoon-set/
[7]https://creativecommons.org/licenses/by-sa/4.0/

4.2. Dataset Size

An important factor when collecting a dataset is to determine how many images are needed. Typically, datasets must represent the proper distribution of the domains for deep learning models to learn from them accurately. If a dataset has too few images, the model may overfit the data. Zhu et al. (2017) trained CycleGAN on several datasets of varying sizes during experiments and showed adequate results with datasets with a size of around only a thousand images per domain. For their horse-to-zebra and apple-to-orange experiments (Figure 2.9, page 29), the training set size of each class was 939 (horse), 1,177 (zebra), 996 (apple), and 1,020 (orange), all resized to 256×256. For their photo-to-art style transfer tasks, their datasets ranged from 401 to 1433 images for the different art styles, all scaled to 256×256. Their experiments suggest that datasets with around 1,000 images for each domain are suitable for style transfer with CycleGAN. In addition, one of the authors explicitly stated that reasonable results can be achieved with relatively small datasets (e.g., 1,000, 2000 images) but that it depends on the complexity of the style transfer and the diversity of the dataset. These findings serve as guidelines for the size of the dataset that will be assembled for experiments in this thesis. However, more data acquisition can be conducted if the models become prone to overfitting during experiments.

4.3. Dataset Acquisition

To acquire images from publicdomainvectors.com, a Python web scraping script was created using the Requests[8] and BeautifulSoup[9] Python packages. Requests was used to fetch HTML data from URLs, and BeautifulSoup to parse the data and fetch all image elements. To ensure that the quality of the image was acceptable, the top 1,800 most downloaded images from the website were scraped. To acquire images from eBoy.com, the same script could not be used since the website is dynamic. BeautifulSoup works mainly for static websites, but the Selenium package[10] handles dynamic websites well. Using Selenium, every image from eBoy.com was scraped. Scraping was unnecessary for the remainder of the dataset, as the images were bundled and available for download. OpenGameArt.com user and pixel artist Henrique Lanzini (username: 7soul) made their collection of 496 pixel art video game icons available under the public domain. The 10k version of Google Research's Cartoon Set was acquired from their web pages.

4.4. Dataset Processing

After the acquisition, manual filtering was needed. From the pixel art from eBoy.com, only images with the .png extension were kept. In addition, only images with a max resolution of 64×64 were selected since larger-resolution images often dictate different art

[8]https://pypi.org/project/requests/
[9]https://pypi.org/project/beautifulsoup4/
[10]https://pypi.org/project/selenium/

styles. Another reason for this choice is that the larger the images are, the harder it is for a deep learning model to learn the importance of each pixel, as evident in 3. Lastly, logos and vulgar depictions were removed. From the images from publicdomainvectors.com, images with many elements and non-cartoon artwork were removed. Manual filtering resulted in 1304 images from eBoy.com and 1562 images from publicdomainvectors.com. The 496 images from 7soul were added to the pixel domain, while the cartoon domain was supplemented with 238 images from the Cartoon Set. This resulted in a total number of 1800 images for each domain.

All the images from both domains were RGBA images. RGB are the color channels (Section 2.3). The A is the alpha channel, which allows for a transparent background. If the alpha channel were to be included, a deep learning model would need to learn from an additional channel, which adds unnecessary complexity since no information about the actual art is stored there. Because of this, it is better to have a uniform white background for all the images. By replacing the transparent pixels with white pixels, artifacts around the edges of objects occur. A solution to this was to add the images onto a white background and then remove the alpha channel. The pixel art images kept their resolution of 64×64 and were added onto a white 64x64 image using the PIL Python package[11]. All cartoon images were downscaled with PIL using bicubic interpolation. They were downscaled such that the longest side had a length of 256 to keep their aspect ratio. This resulted in varying image sizes but was fixed after pasting them onto a white 256×256 background. After pasting, the images were converted from RGBA to RGB using PIL and saved as PNGs.

Statistics about the collected dataset is given in Table 4.1, and examples from the cartoon and pixel art domains are shown in Figure 4.1.

Domain	**HxW**	**Color space**	**Total**	**Resource**	**Images**
Pixel art	64x64	RGB	1,800	eBoy	1,304
				7Soul/OpenGameArt	496
Cartoon	256x256	RGB	1,800	Publicdomainvectors	1,562
				Google Research Cartoon Set	238

Table 4.1.: The collected dataset, divided into the pixel art and cartoon domains.

[11] https://pypi.org/project/Pillow/

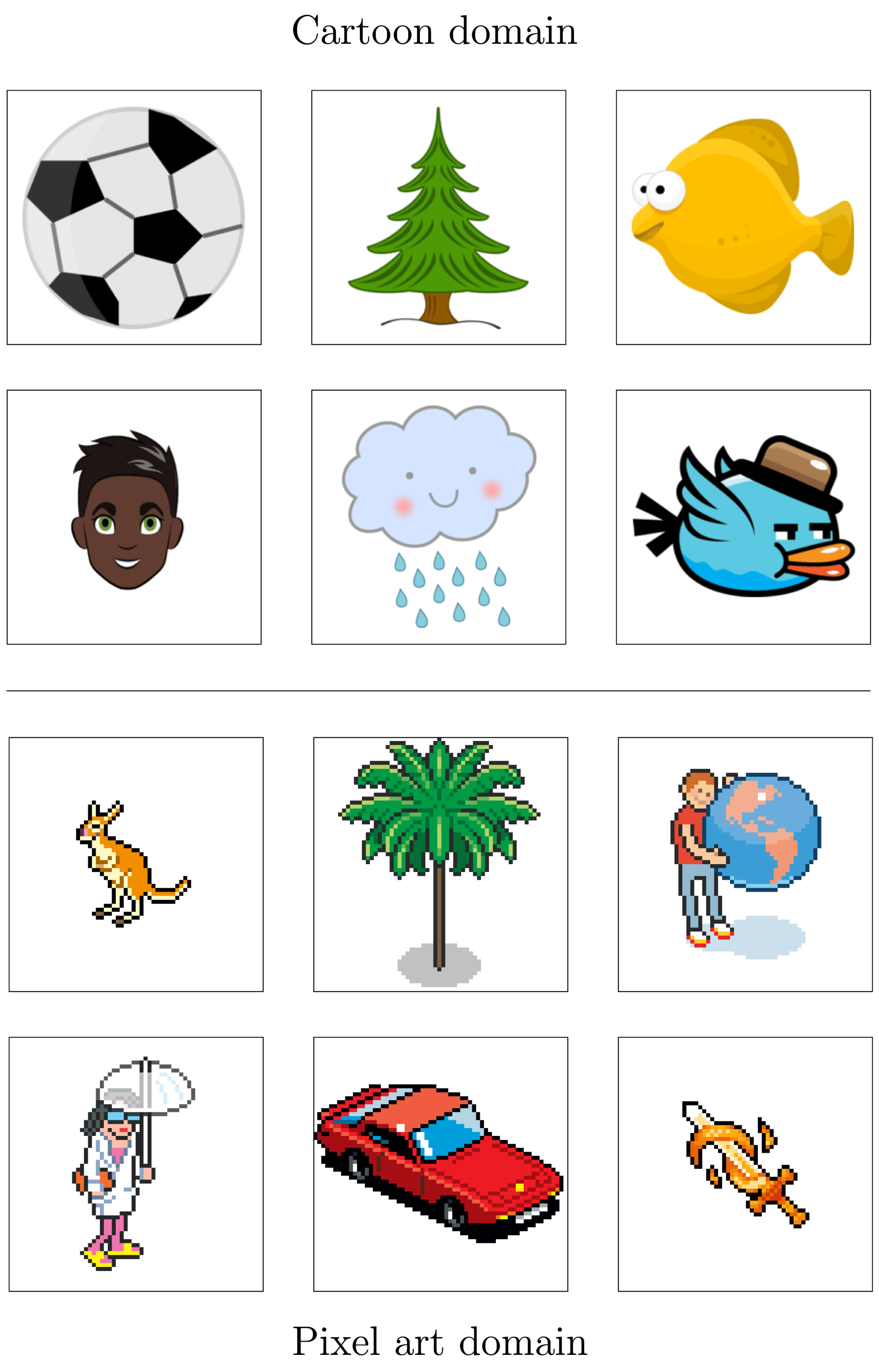


Figure 4.1.: Examples from the dataset with images of the cartoon and pixel art domains.

5. Architecture

This chapter introduces the architectures used in the four experiments in Chapter 6. The CycleGAN architecture introduced in Section 2.7.7 is presented as the base architecture for the experiments. The base CycleGAN architecture contains a ResNet generator and a PatchGAN discriminator. This is the recommended architecture from the official implementation by Zhu et al. (2017). Both networks are built upon the concept of DCGAN (Section 2.7.3), where fully connected layers are replaced with convolutional layers.

Several extensions and modifications are made to the base architecture, resulting in four different architectures for the four experiments. Each architecture comprises two equal generator networks and two equal discriminator networks. Each network is an instance of the same architecture, and a network pair will thus be referred to as one generator and one discriminator in subsequent chapters. The model structure shown in Figure 2.8 (page 27) applies to all architectures presented.

5.1. ResNet Generator

The ResNet generator, shown in Figure 5.1, performs downsampling, transformation, and upsampling. The input to the generator is a 256×256 pixel image with a depth of three color channels.

The downsampling step performs three convolutions (Section 2.4), where each convolution is followed by an instance normalization (Section 2.2.10) of the weights and a ReLU activation function (Section 2.2.2). The dimensions of each convolution block represent the output of the convolution operation. For example, the left-most convolution block in Figure 5.1, transforms the input image to dimensions $256 \times 256 \times 64$. The spatial resolution of the image is not changed because the stride is set to 1, and 64 filters with kernel size 7 are applied to extract features. The second left-most convolution block takes the $256 \times 256 \times 64$ image as input and outputs a $128 \times 128 \times 128$ image. Here, 128 filters with kernel size 3 are applied. The spatial resolution is halved since the stride is set to 2. This is repeated for the third convolution, resulting in an image size of $64 \times 64 \times 256$. The number of filters is doubled to extract more features from the images, and the spatial resolution is halved to reduce the complexity of the network while still maintaining the information in the features. Each convolution layer adds padding around the image to not lose information around the edges. This is done by reflection padding, where values are padded outside borders by mirroring the neighboring interior pixels of the edge pixels. Zhu et al. states that the reflection padding reduces boundary artifacts during training.

The transformation of features follows the downsampling. Here, the transformation of

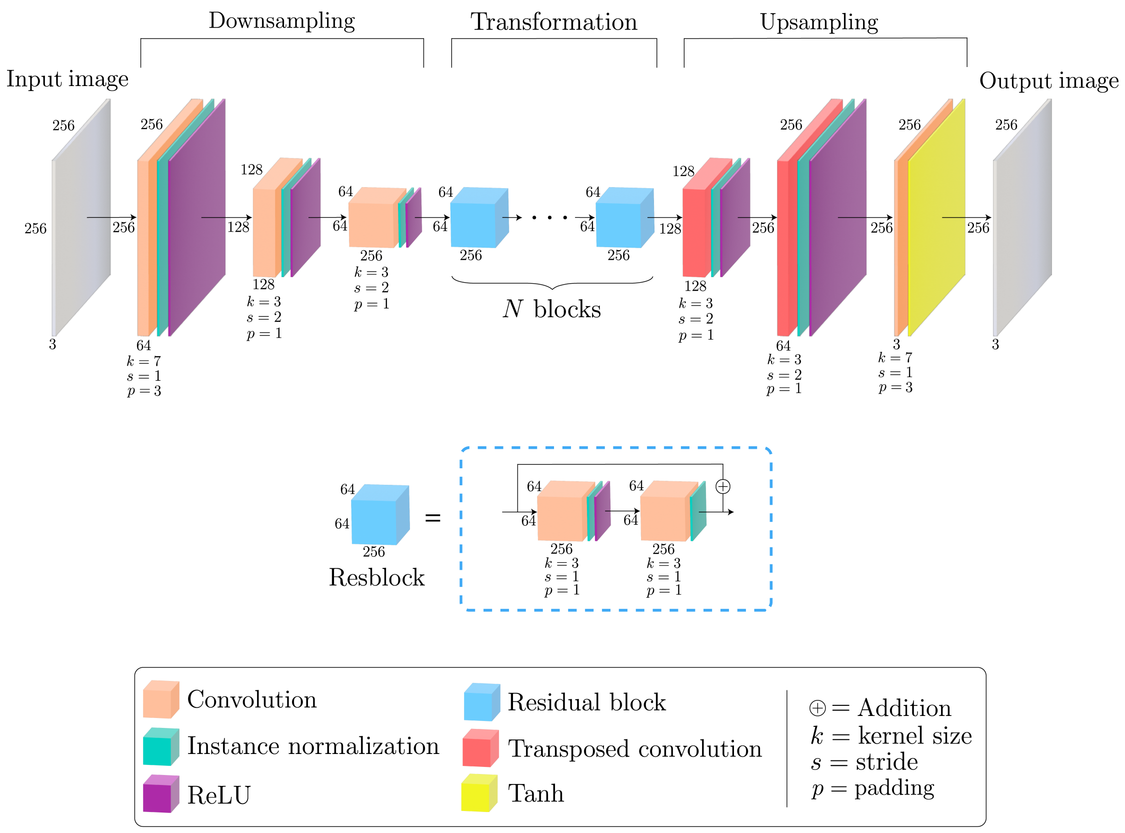


Figure 5.1.: ResNet generator architecture

the image from the input to the output domain occurs. The transformation comprises N residual blocks (Section 2.5), each containing two convolutions, followed by instance normalization, ReLU, and a skip connection. The skip connection adds the input and the output of the block together. The value of N blocks is often chosen based on the complexity of the style transfer task and the size of the input image and is often chosen to be 6 or 9. These values of N will be tested throughout the experiments.

After transforming the features, the image's resolution is upsampled, and the number of features is reduced. This is done by a series of transposed convolutions (Section 2.4), where 0-pixels are inserted between every row and every column in the image, followed by a convolution with kernel size 3. This results in doubling the resolution caused by the stride with value 2. Each transposed convolution is followed by instance normalization and ReLU activations. After two upsampling layers, the image is fed to a final convolution to reduce the number of filters from 64 to 3. This is followed by a Tanh activation function (Section 2.2.2) to map the pixel values to the range [-1,1], resulting in the output image.

5.2. PatchGAN Discriminator

After an image has gone through the style transfer of the generator, it is fed into a PatchGAN discriminator to determine if it can be classified as real or fake. The architecture for PatchGAN is shown in Figure 5.2. Overlapping patches of size 70×70 in the image are evaluated, each receiving a score between 0 and 1. The resulting scores are stored in a 30×30 matrix to be compared to the ground truth. Like the generator, the convolutional layers downsample the resolution and increase the filters. The first four convolutions extract features, while the last convolution determines whether the patches' features are real or fake. The first four convolutions in the discriminator use LeakyReLU activations (Section 2.2.2) with a slope of 0.2. The middle three convolutions are all followed by instance normalization. A further explanation of PatchGAN is given in Section 2.7.6.

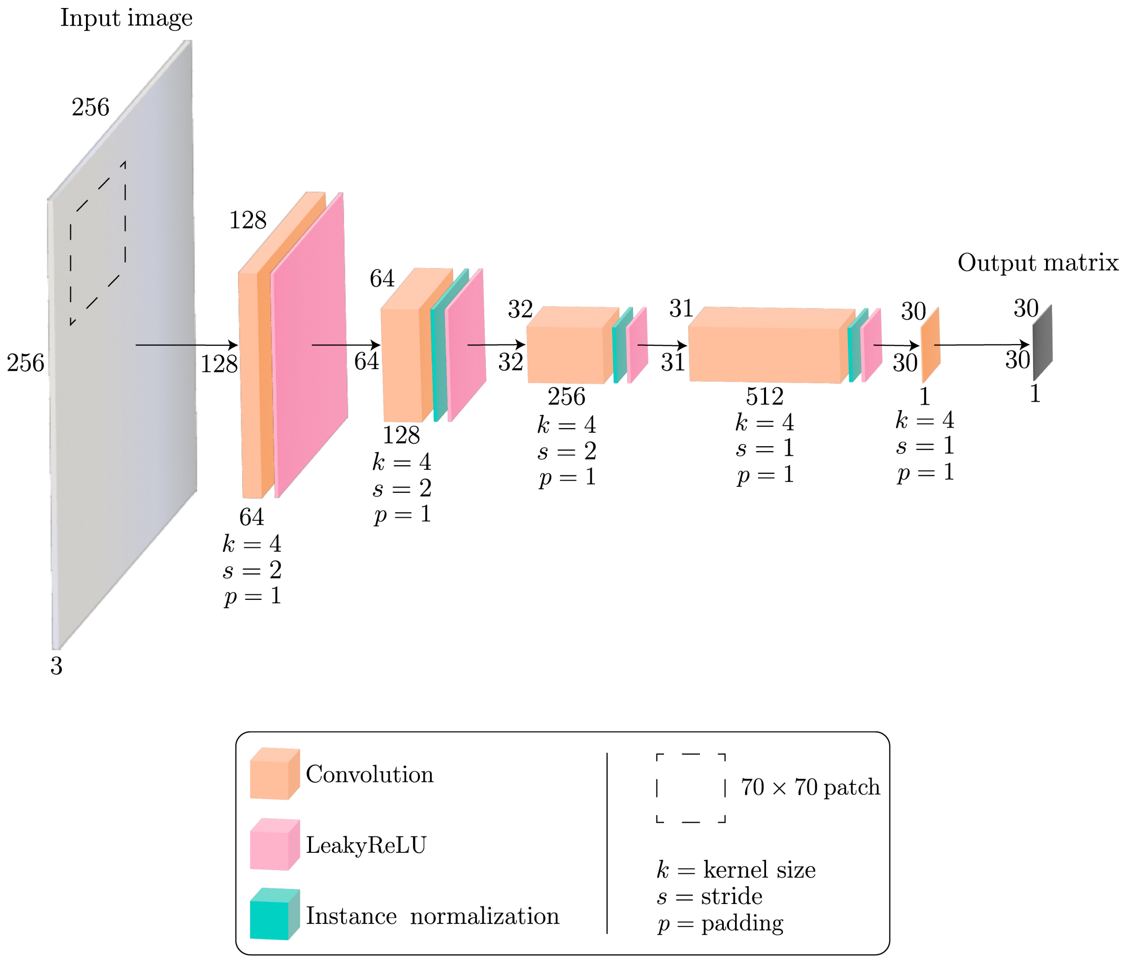


Figure 5.2.: PatchGAN discriminator architecture.

5.3. U-NET Generator

For the image-to-image translation model using paired images, pix2pix (Section 2.7.7), Isola et al. (2017) suggested a modified U-NET architecture as the generator network. The authors motivate the choice of the U-NET generator by considering that even though the input and image should differ in surface appearance, both are renderings of the same underlying structure. Because of this, information from the input image should be able to flow to the output image. Similar to pix2pix, using U-NET as the generator architecture can be directly applied to CycleGAN with some modifications. Modifying the original U-NET (Section 2.6) replaces the max-pooling layers with convolutions and transposed convolutions with stride = 2. These changes make the U-NET operate similarly to ResNet in that the convolutions halve the image's resolution and increase the number of filters by setting the stride to 2. The reverse occurs for the upsampling layers. As shown in Figure 5.3, the first four convolutions double the number of filters. All convolutions halve the spatial resolution until the bottleneck layer where the image is of size 1x1 with 512 filters. A LeakyReLU activation with a slope of 0.2 follows the first convolution. ReLU follows the last convolution. The remaining convolutions are followed by instance normalization and LeakyReLU.

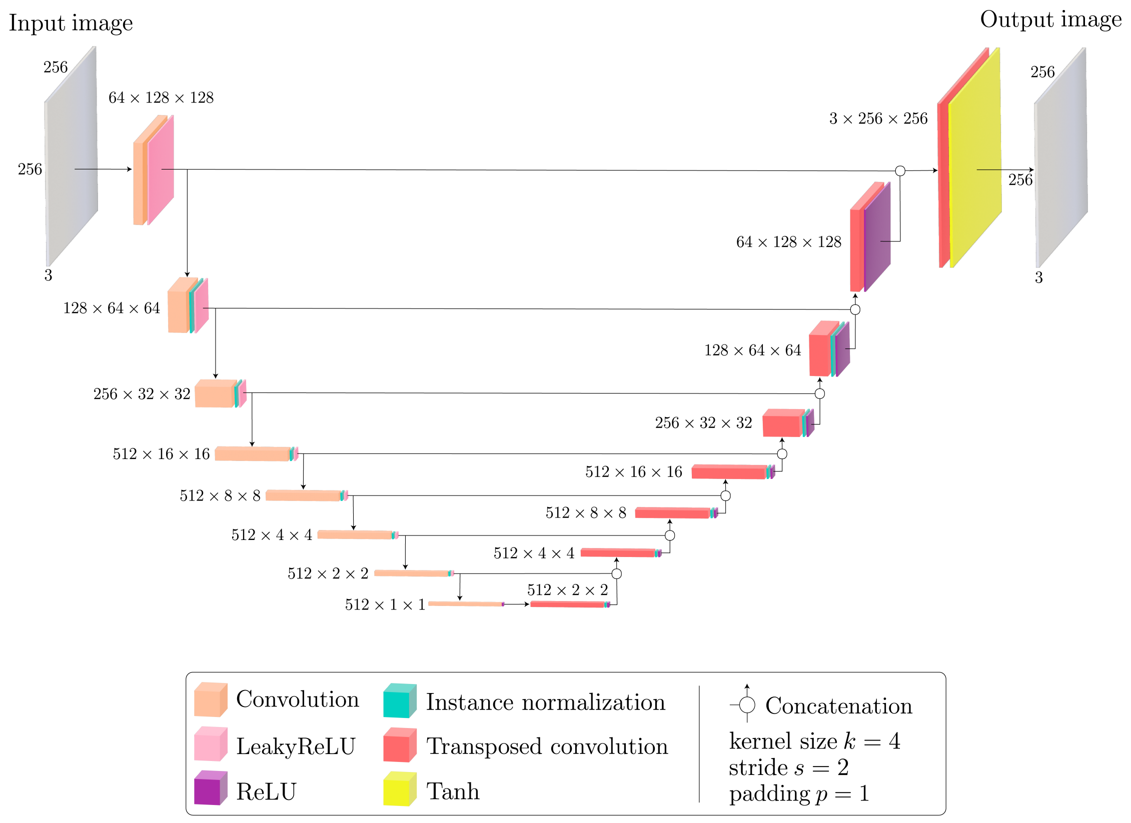


Figure 5.3.: U-NET generator architecture

After downsampling, the resolution is doubled through transposed convolution layers, with the final four layers also reducing the number of filters. Except for the final two, each transposed convolution is followed by instance normalization and ReLU activations. The last layer is followed by a Tanh activation to map the output to the range [-1,1]. Each downsampling and upsampling layer is responsible for changing the resolution and number of filters and learning transformation features. The output of equal-sized downsample and upsample layers are concatenated and fed into their next upsampling layer.

5.4. Nearest-Neighbor Upsampling Layers

For several experiments using the ResNet generator, the transposed convolution layers are replaced with nearest-neighbor upsampling, followed by a regular convolution. Nearest-neighbor upsampling scales the spatial resolution of the image by a factor f and is further explained in Section 6.2. The upsampling part of the ResNet generator in Figure 5.1 (page 44) is replaced by the upsampling in Figure 5.4 (page 48). Here, the spatial resolution is doubled (by a factor of 2). In contrast to the transposed convolution layers, the nearest-neighbor upsampling layers do not change the number of filters or have any learnable weights. Because of this, the upsampling is followed by a regular convolution to reduce the number of filters. To not further change the spatial resolution, the stride of these convolutions is set to 1. The normalization and activation layers remain unchanged.

For the experiments with the U-NET generator and the modified ResNet generator, the PatchGAN discriminator will be improved by adding spectral normalization (Section 2.2.10). This will be done by replacing the three instance normalization layers (Figure 5.2, page 45) with spectral normalization. Spectral normalization helps stabilize the training of the discriminator.

5.5. Self-Attention Layers

The last experiment in Chapter 6 adds Self-Attention layers (Section 2.7.5) to the ResNet generator and the PatchGAN discriminator. This complete architecture is shown in Figure 5.5 (page 49), where the Self-Attention layers comprise several convolutions. Dimensions are omitted for simplicity, but all layers in Figure 5.5 have the same dimensions as those in Figures 5.1 (ResNet generator) and 5.2 (PatchGAN discriminator).

First, the input to the layer is fed through three separate convolutions with a kernel size of 1. The two top convolutions in Figure 5.5 (c) outputs 1/8 of the number of input filters. E.g., if the number of input filters is 256, the output is 32. These output matrices are multiplied and fed into a softmax activation function. Softmax is a generalized version of the sigmoid function (Section 2.2.2). The output matrix of the softmax function is called the attention map, which is then multiplied by the output of the bottom convolution. The result of the multiplication is then fed into a final convolution, resulting in the Self-Attention feature maps. These feature maps are added together with the input to the Self-Attention layer.

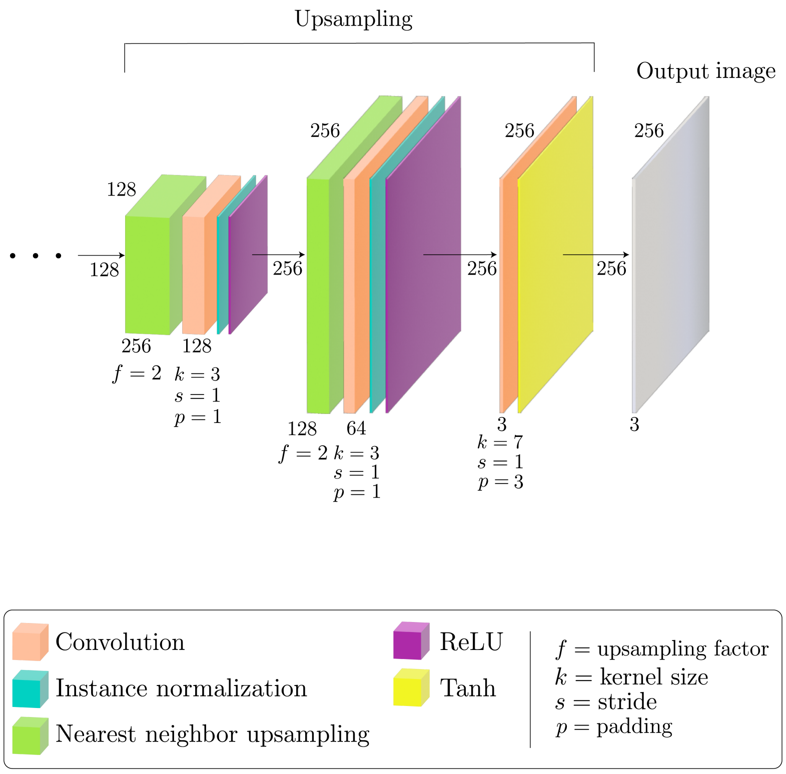


Figure 5.4.: Nearest-neighbor upsampling layers

The relevant aspect of the Self-Attention layer in relation to the experiments is that it enables the networks to view and compare local features in distant parts of the image. This may ensure consistency in the generation and the discrimination of features across the image. The complete calculations can be found in Zhang et al. (2019).

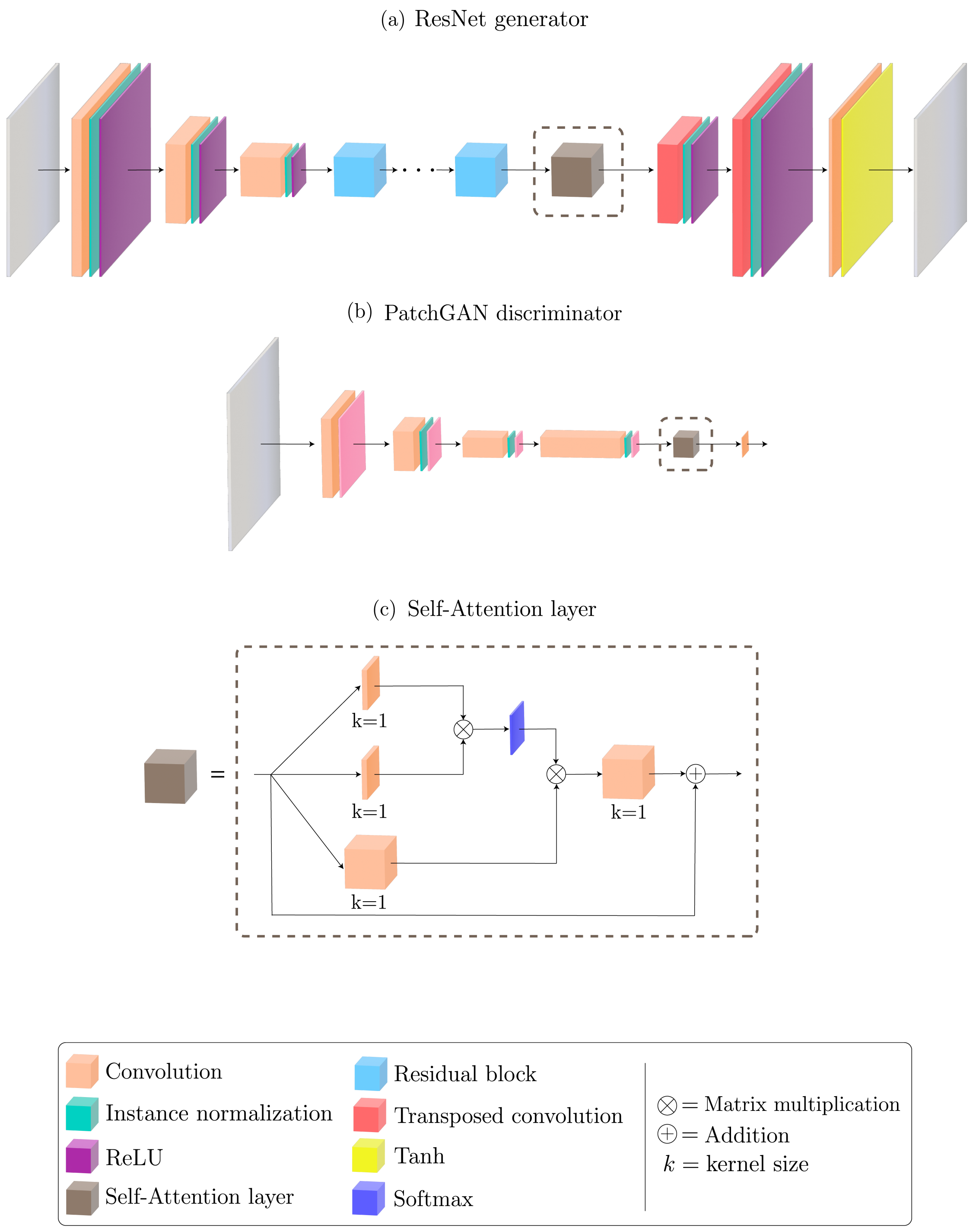


Figure 5.5.: Self-Attention layers

6. Experiments and Results

This Chapter presents a series of experiments that employ the various network architectures presented in Chapter 5. Section 6.1 presents the experimental plan, Section 6.2 presents the experimental setup, and Section 6.3 presents each conducted experiment, along with corresponding motivations, implementations, and results. A broader discussion of the results will be given in Chapter 7.

6.1. Experimental Plan

The plan for the experiments is to test different datasets and different architectural extensions and improvements to answer the research questions:

RQ1 *How can GAN architectures be designed to be able to create pixel art effectively?*

RQ2 *What impacts do the datasets' quality and quantity have on the generated artwork?*

The experiments will use the base CycleGAN model as a starting point. The base CycleGAN model is the official implementation presented in Section 2.7.7, with the ResNet generator (Figure 5.1, page 44) and the PatchGAN discriminator (Figure 5.2, page 45), using the parameters in Table 6.1.

The overall experimental plan is divided into four experiments. Experiment 0 will use the base CycleGAN model to compare the dataset collected for this thesis (Chapter 4) to the dataset by Kuang et al. (2021) presented in Chapter 3, and thus answer RQ2. Experiment 1 will optimize the ResNet generator, Experiment 2 will optimize the U-NET generator (Figure 5.3, p. 46), and Experiment 3 will use Self-Attention layers(Figure 5.5, p. 49). All three experiments will begin with the PatchGAN discriminator. Optimizations will be done by iterating the architectures through several sub-experiments, where the results of each sub-experiment will dictate the next. Each sub-experiment is termed "Experiment 1.1", "Experiment 1.2", and so forth. Motivation, implementation details, and results will be presented for each sub-experiment. The goal of the experiments is to discover which architecture is the most suitable for synthesizing pixel art and thus answer RQ1, so the results for each sub-experiment will be discussed in their corresponding sections. The results from the most successful sub-experiments will be further discussed and evaluated in Chapter 7.

6.2. Experimental Setup

This section first presents how the architectures from Chapter 5 are implemented, followed by the environments and resources the experiments will run on. Then, the choice of parameters is presented, followed by how the dataset will be processed.

6.2.1. Implementations

The experiments will be carried out utilizing the implemented network architectures, dataset loader, and training loop from the codebase specifically developed for this thesis, which is available on Github[1]. To implement the network architectures presented in Chapter 5, PyTorch[2] was used. Code for the ResNet generator, U-NET generator, and PatchGAN discriminator were copied from the official CycleGAN implementation[3] from Zhu et al. (2017). The official implementation also provides code for replacing transposed convolutions with custom upscaling layers for the generators. Spectral normalization (Section 2.2.10) is included with PyTorch utils[4] and was directly applied to the discriminator. More details about the architecture implementations are discussed in their respective experiments. Elements of the code for loading datasets and training the networks were adapted from the official CycleGAN implementation and from a collection of PyTorch GAN implementations[5]. This collection is a good resource for training GANs on custom datasets, including CycleGAN. The code in the collection is available under the MIT license[6].

6.2.2. Environment and Resources

All the experiments will run on the NTNU HPC Idun Cluster (Själander et al., 2019), which is a computing resource with high-performance CPUs and GPUs. The experiments will be conducted in Jupyter Notebooks[7] in an Anaconda[8] environment (version 3/2022.05). The notebooks will run on an Nvidia Tesla A100 GPU with 40 GB memory, using the PyTorch CUDA interface[9]. The codebase uses Python version 3.9.12 and PyTorch version 2.0.0. Using the A100 GPU, each sub-experiment takes approximately 3 hours, except for Experiment 3, which takes 14 hours because of network complexity.

6.2.3. Parameters

Hyperparameters, variables, and other architectural choices are presented in Table 6.1. These are the default parameter values for all the experiments and remain the same

[1]https://github.com/ekpete/masterthesis
[2]https://pytorch.org
[3]https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix
[4]https://pytorch.org/docs/stable/generated/torch.nn.utils.parametrizations.spectral_norm
[5]https://github.com/eriklindernoren/PyTorch-GAN/tree/master
[6]https://github.com/eriklindernoren/PyTorch-GAN/blob/master/LICENSE
[7]https://jupyter.org
[8]https://www.anaconda.com
[9]https://pytorch.org/docs/stable/cuda.html

unless otherwise stated. Changes to parameters are stated prior to each sub-experiment.

Parameter	Value	Description
n epochs	125	Number of epochs.
batch size	1	Batch size.
H	256	Height of the images.
W	256	Width of the images.
C	3	Number of color channels in the images.
lr	0.0002	ADAM learning rate.
β_1	0.5	ADAM decay of first-order momentum of gradient.
β_2	0.999	ADAM decay of second-order momentum of gradient.
λ_{cyc}	10	Weight of the cycle-consistency loss.
λ_{id}	5	Weight of the identity loss.
ngf	64	The number of filters in the first layer of the generator.
ndf	64	The number of filters in the first layer of the discriminator.
n blocks	6	The number of ResNet blocks.
n downs	8	The number of U-NET downsampling layers (depth).

Table 6.1.: The default values of parameters used in the experiments.

6.2.4. Data Preprocessing

For each sub-experiment, the models will train for 125 epochs. This was chosen since Zhu et al. recommend training CycleGAN models for 200k iterations. The dataset will be split with a ratio of 90/10, with a training set size of 1,620 and a test set size of 180. Setting the batch size of 1 results in $125 \times 1{,}620 = 202{,}500$ iterations. The remainder of the hyperparameters has the default values from the official CycleGAN implementation.

As shown in Chapter 4, the images from the cartoon domain have a resolution of 256x256, while images from the pixel art domain have a resolution of 64×64. The base CycleGAN architecture assumes an equal size for both domains since the loss functions need tensors of identical dimensions for comparison. It is possible to change the architecture such that the cartoon-to-pixel art generator takes in 256×256 images and outputs 64×64 images by removing upsampling layers. By adding upsampling layers, the pixel art-to-cartoon generator can take 64×64 images as input and output 256×256 images. However, this may hinder first generator from learning essential features since layers with weights are removed.

To circumvent the need to remove layers, either the cartoon images can be downsampled or the pixel images can be upsampled prior to training. Downsampling and upsampling of images are usually done with interpolation techniques such as bilinear interpolation or nearest-neighbor interpolation. Nearest-neighbor interpolation identifies an unknown data point on the basis of its nearest neighbor whose value is already known (Bhatia and Vandana, 2010). Nearest-neighbor scales pixel art without information loss, due to

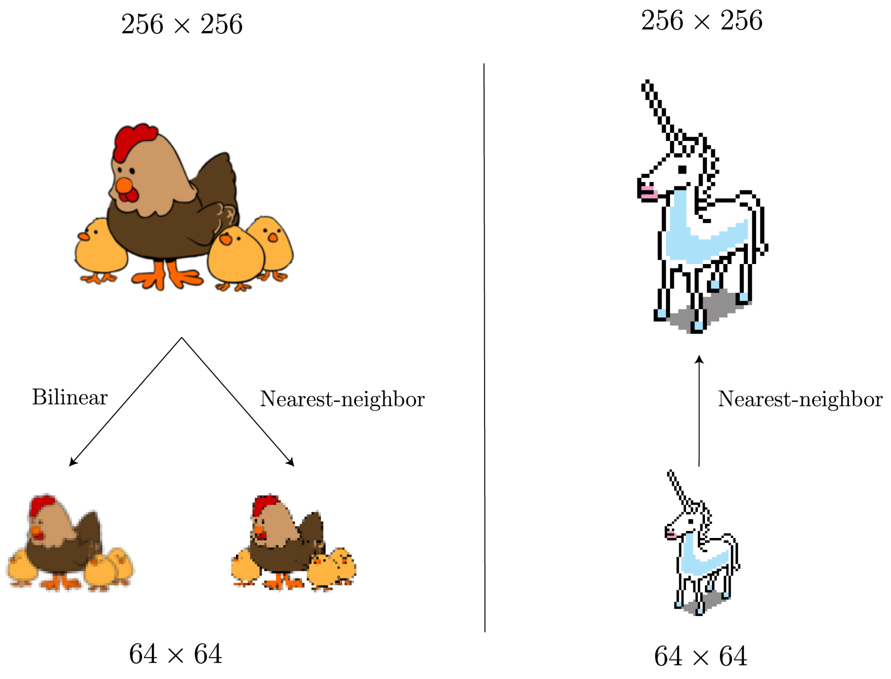


Figure 6.1.: Left: Downsampling from 256 × 256 to 64 × 64 using nearest-neighbor and bilinear interpolation. Right: Upsampling from 64 × 64 to 256 × 256 using nearest-neighbor interpolation.

the organization of the pixels, but applying it to a continuous-tone image may result in blocky artifacts.

Bilinear interpolation is an extension of linear interpolation to two dimensions, where unkown data points are fitted to the linear curve between existing points. Bilinear interpolation can result in blurry edges and loss of texture (Suwendi and Allebach, 2008). The effects of downsampling cartoon images and upsampling pixel art images are shown in Figure 6.1, where the cartoon image loses essential features, but the pixel image is perfectly scaled. Because of these results, all images from the pixel art domain will be upscaled to 256 × 256 prior to training using nearest-neighbor interpolation.

Furthermore, several preprocessing steps will be done for all images of both domains. By default, PyTorch scales RGB color values from $[0, 255]$ to $[0, 1]$. In addition to this, a normalization step will be added to normalize the images to the range $[-1, 1]$ to make the images coincide with the output of the Tanh activation function (Section 2.2.2). This can be done by subtracting each color value by 0.5 and then dividing by 0.5. Finally, the order of the images will be randomized each epoch, and images will be randomly flipped horizontally. This has the effect of creating a more diverse dataset, which may prevent overfitting.

6.3. Experimental Results

This section presents the results from the four experiments. Each experiment has several sub-experiments, where the goal is to optimize the architecture used in the experiment.

6.3.1. Experiment 0: Datasets

This experiment tested the importance of the quality and quantity of the dataset while also providing a baseline for the remainder of the experiments. Firstly, the base CycleGAN model was trained on the dataset from Kuang et al. (2021) presented in Chapter 3. Secondly, the model was trained on the collected dataset for this thesis, presented in Chapter 4. Lastly, the model was trained on a subset of the collected dataset to test the importance of the dataset size. The first dataset had 810 images per domain, the second had 1620, and the third had 500. Because of the varying dataset sizes, the model trained for 250 epochs for the first dataset, 125 epochs for the second, and 400 epochs for the third. This resulted in approximately 200k iterations for all three datasets.

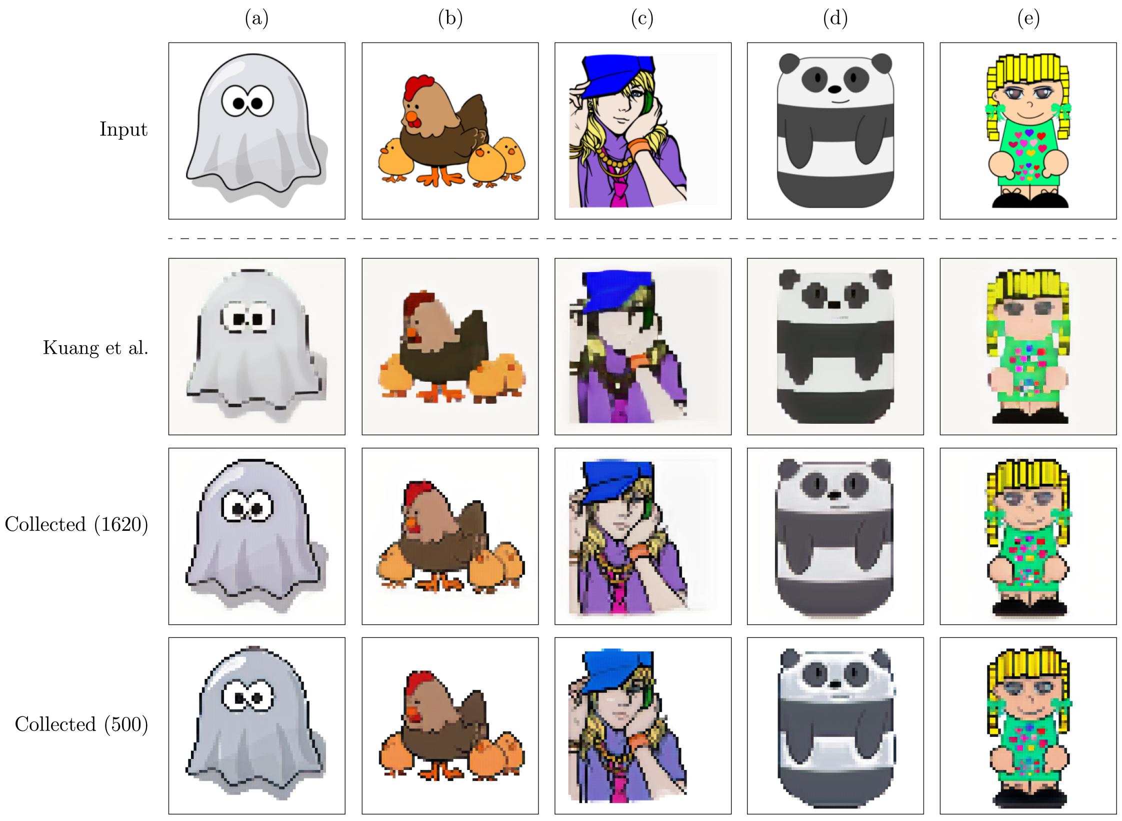


Figure 6.2.: Results from Experiment 0.

Results from training the base model on the three datasets are shown in Figure 6.2. When trained on the dataset from Kuang et al., blocky images were generated, but with no apparent regard for the stylistic rules of pixel art. This comes as a result of the training data having images of many different pixel sizes, despite having equal resolution. In addition, this training data has many occurrences of background noise, which can contribute to the generated images having a non-white background, as seen in Figure 6.2. Furthermore, many important details are lost, such as edges.

The model trained on the full collected dataset significantly improves the model trained on the dataset from Kuang et al.. Individual pixels are discernible, they are of equal size and details are preserved. In addition, no style transfer is being done to the background. The partial dataset exhibits slightly worse results in some cases, compared to the full dataset. The main issue is the failure to produce uniform colors, as seen in Figure 6.2 (c) and (d).

The results show the importance of a large, uniform dataset with images that all conform to the pixel art style. The complete collected dataset will be used for the remainder of the experiments. Despite this, training the base model on the complete dataset presents certain challenges, primarily in relation to the preservation of colors and other details. This will be further discussed in subsequent experiments.

6.3.2. Experiment 1: ResNet

In Experiment 0, the base model performed well on the full collected dataset and created images composed of equal-sized pixels while retaining the structure and essential features of the input images. One apparent issue, however, is color retention, where light colors were darkened in several images. The next sub-experiments will explore architectural improvements to the ResNet generator and the PatchGAN discriminator.

Experiment 1.1: Increasing λ_{id}

Zhu et al. (2017) added an additional loss function to the model to ensure color consistency between the input and output images. This loss function is called identity loss, and it regularizes the generators to be close to an identity mapping when real samples of the target domain are taken as input. In other words, the identity loss ensures that if, for example, the cartoon-to-pixel art generator receives pixel art as input, the pixel image should remain the pixel domain after the style transfer. This also entails that the generators will retain the colors of the input image, apart from the colors that are supposed to change during the style transfer. The effect of the identity loss is controlled by a weight λ_{id}. By this, increasing the weight may further ensure overall color consistency. This experiment increased the weight λ_{id} by a factor of 2.

Increasing λ_{id} to 10 resulted in improved color retention for images with light colors, as shown in Figure 6.3. Further increasing the weight may result in the other losses becoming overlooked, so $\lambda_{id} = 10$ is used for the remainder of the experiments. Even though the color was improved, another issue remained. On closer inspection, many of the images developed repeating patterns instead of having solid colors. Figure 6.4 shows examples of this, where the repeating checkerboard patterns can be seen. These artifacts are subtle but give the appearance of non-uniform colors.

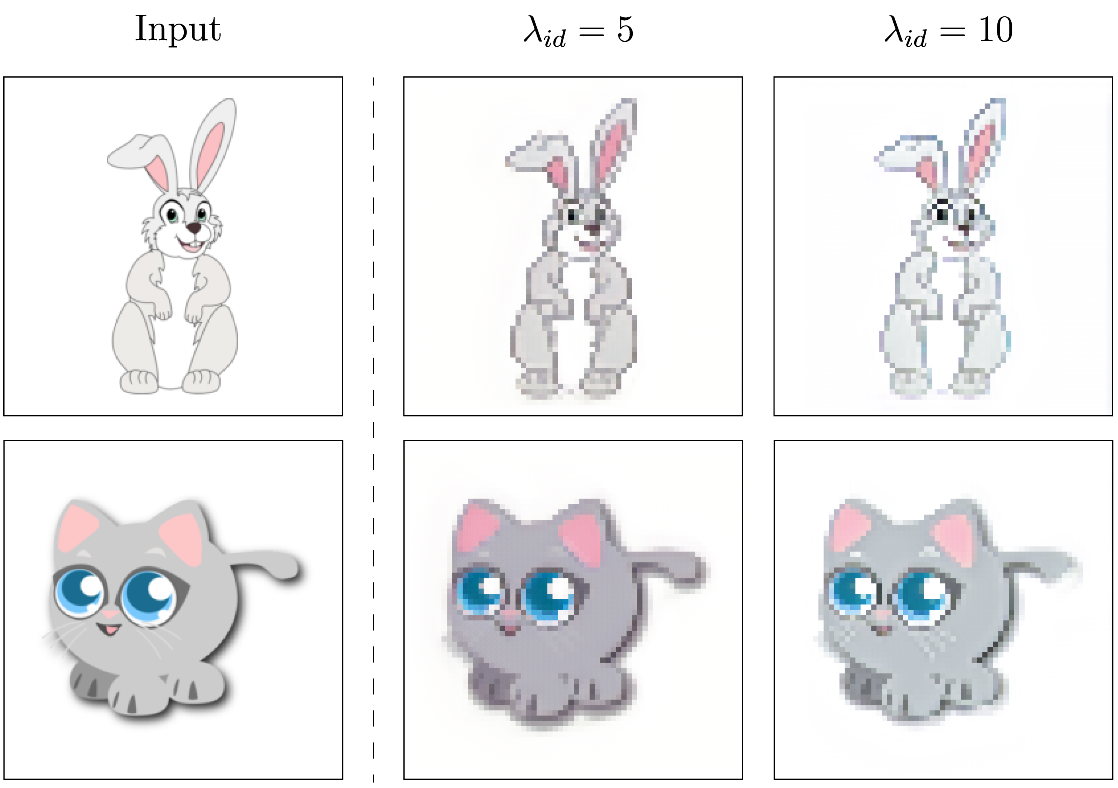


Figure 6.3.: Results from Experiment 1.1

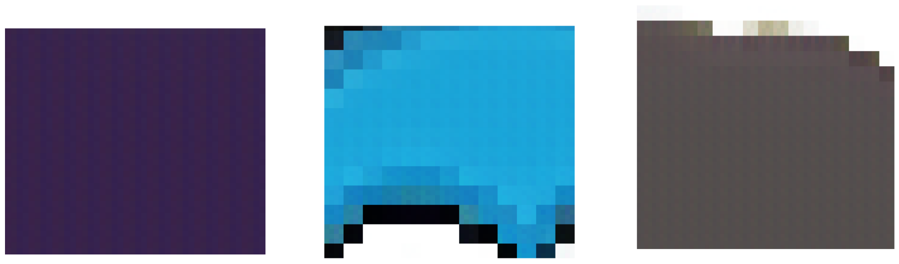


Figure 6.4.: Checkerboard artifacts

Experiment 1.2: Upsampling

Checkerboard artifacts are a common problem for GANs. Odena et al. (2016) showed that these artifacts may stem from transposed convolutions in the upsampling layer. Networks may be able to adjust their weights to avoid this, but they may not be able to do so completely. This was evident in Experiment 1.1, where the artifacts were more severe in earlier parts of training. This may be because transposed convolutions can cause uneven overlapping of pixels when upsampling. The authors suggest replacing all transposed convolutions with nearest-neighbor upsampling layers followed by regular convolutions.

These upsampling layers do not have any learnable weights, but normal convolutions are added to ensure that the upsampling still can learn features. This experiment replaced the two transposed convolution layers in the ResNet generator with two nearest-neighbor upsampling layers. The replaced transposed convolutions had kernel size = 3, stride = 2, and padding = 1. The stride with value 2 was responsible for the double upsampling of the spatial resolution. After the change, each nearest-neighbor upsampling layer is responsible for the doubling. Each upsampling was followed by reflection padding and a convolution layer with kernel size = 3 and stride = 1. The nearest-neighbor replacement in the generators is denoted NN_G.

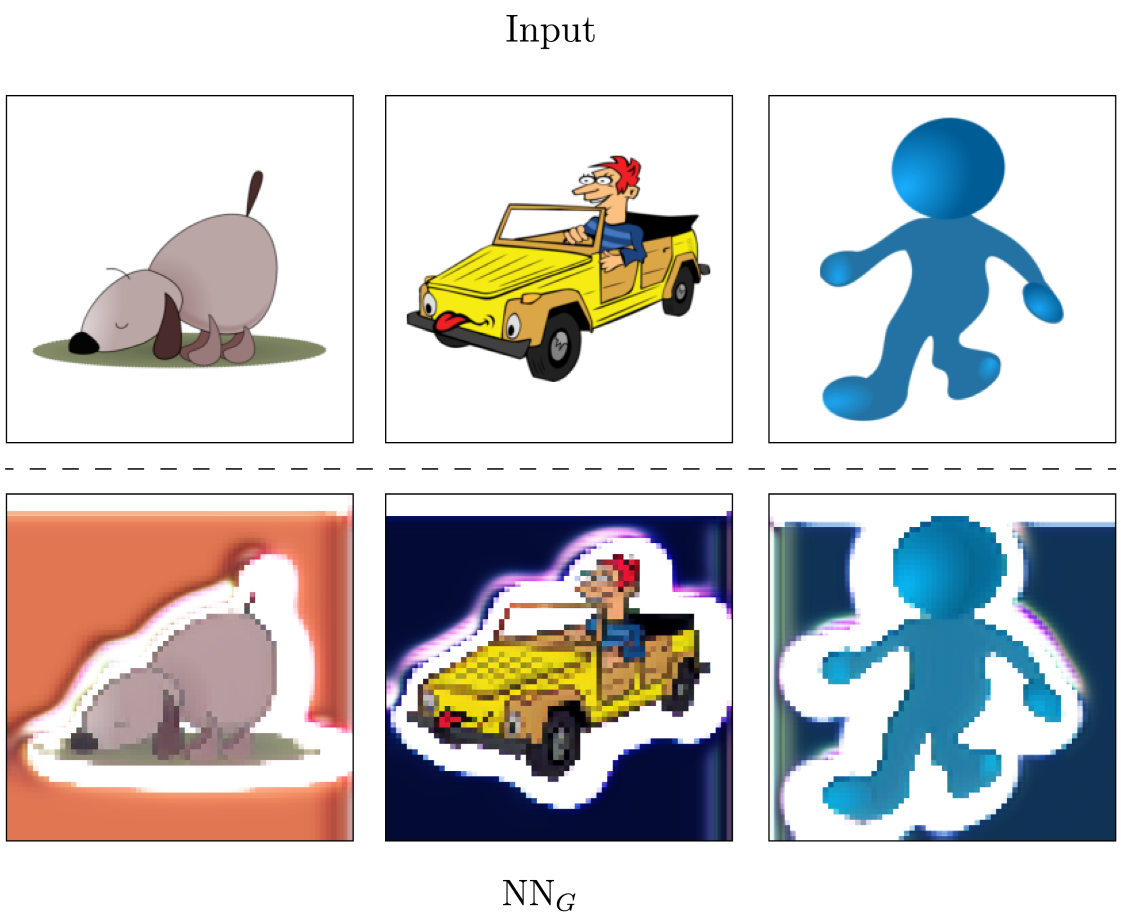


Figure 6.5.: Results from Experiment 1.2

As shown in Figure 6.5, replacing the transposed convolutions resulted in unstable training and solid color artifacts around the generated images. It seems like the generator only learned the mapping of the object and ignored the background. This issue was persistent for most of the training epochs for the generated pixel art, but not for the generated cartoons. A reason for this can be instability in the pixel art discriminator which can result in non-convergence, as mentioned in Section 2.7.1. When machine learning models fail to converge, the graphs of the loss functions usually indicate training instabilities. However, the loss functions of CycleGAN usually do not unveil a lot of information. Despite this, Figure 6.6 shows that the losses in this experiment acted differently than the losses in the base model. In addition, the pixel art discriminator (D_B) had more variance than the cartoon discriminator (D_A), which may suggest that stabilization of D_B is needed.

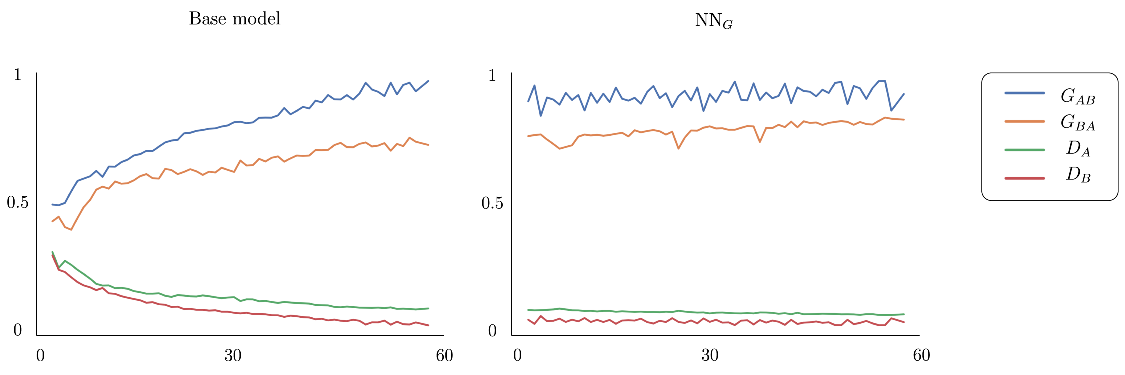


Figure 6.6.: Loss functions from Experiment 1.2

Experiment 1.3: Spectral Normalization

To stabilize the discriminators, and thus stabilize the generators, spectral normalization (Section 2.2.10) can be applied to the discriminators. This experiment replaced all occurrences of instance normalization in the discriminators with spectral normalization, which is denoted SN_D.

As shown in Figure 6.7 (page 60), spectral normalization greatly stabilized the training. The major background artifacts were gone, except for some contours around the objects. Because of the remaining artifacts, another experiment was conducted by spectral normalizing the generators to further stabilize the model. This resulted in model failure during early training, where all generated images became completely black. The reason for this could be related to the normalization process. The documentation for the PyTorch spectral normalization implementation[10] states that the method is made for discriminators. This could suggest that the implementation is not optimized for generators, which could explain the failure.

Experiment 1.4: 9 ResNet Blocks

The artifacts in Experiment 1.3 may be caused by the model not being complex enough to learn all features since the transposed convolution layers were replaced. One way to make the model more complex is to add more ResNet blocks. Even though the official CycleGAN implementation defaults to 6 ResNet blocks, Zhu et al. recommend 9 blocks for 256x256 images. This experiment increased the number of ResNet blocks from 6 to 9.

[10]https://pytorch.org/docs/stable/generated/torch.nn.utils.parametrizations.spectral_norm

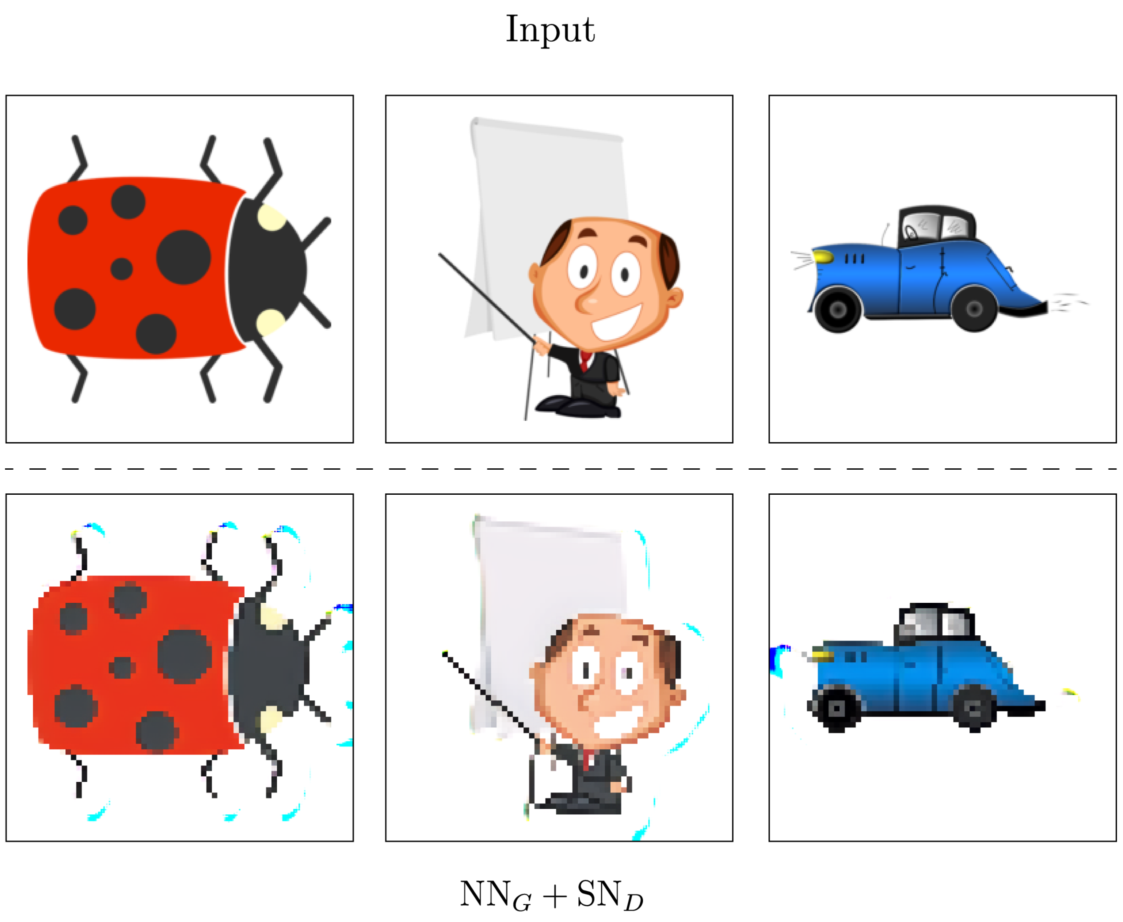


Figure 6.7.: Results from Experiment 1.3

As shown in Figure 6.8 (page 61), increasing the number of ResNet blocks completely removed the artifacts present in Experiment 1.3. This may be because the increased model complexity allowed the networks to learn not to produce artifacts. Furthermore, 9 ResNet blocks and spectral normalization allowed the model to use nearest-neighbor upsampling instead of transposed convolutions. This resulted in zero artifacts and zero checkerboard patterns. The overall use of color is good, but the color changes slightly from the input in some generated samples. A further discussion on color retention is done in Section 7.2.

6.3.3. Experiment 2 - U-NET

This experiment will test the U-NET generator to examine if it improves the ResNet generator in the synthesis of pixel art.

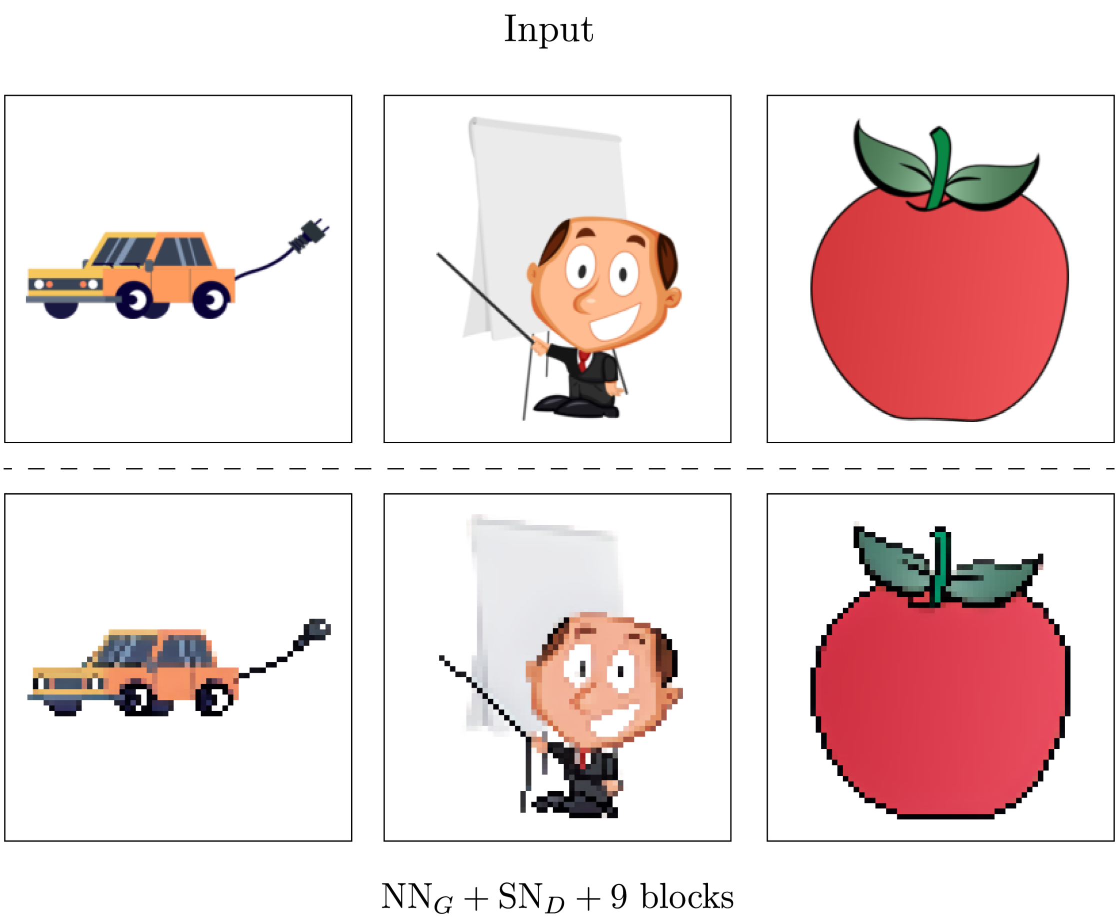


Figure 6.8.: Results from Experiment 1.4

Experiment 2.1: U-NET Generator

Experiment 2.1 replaced the ResNet generators with U-NET generators. This was done because the skip connections in U-NET may further ensure the integrity of the input image and allow for better color retention and sharp edges. The discriminators and all other parameters in Table 6.1 remained the same. The generator change resulted in severe mode collapse, where all generated images exhibited the same noisy filter, as shown in Figure 6.9 (page 62). This may be caused by unstable discriminators, as experienced in Experiment 1.2.

Experiment 2.2: Spectral Normalization

Similar to Experiment 1.3, this experiment replaced all occurrences of instance normalization in the discriminators with spectral normalization, denoted SN_D. This significantly stabilized the training and enabled the model to produce images without disturbing filters. Despite this, cases of checkerboard artifacts occurred, as shown in Figure 6.10 (page 63).

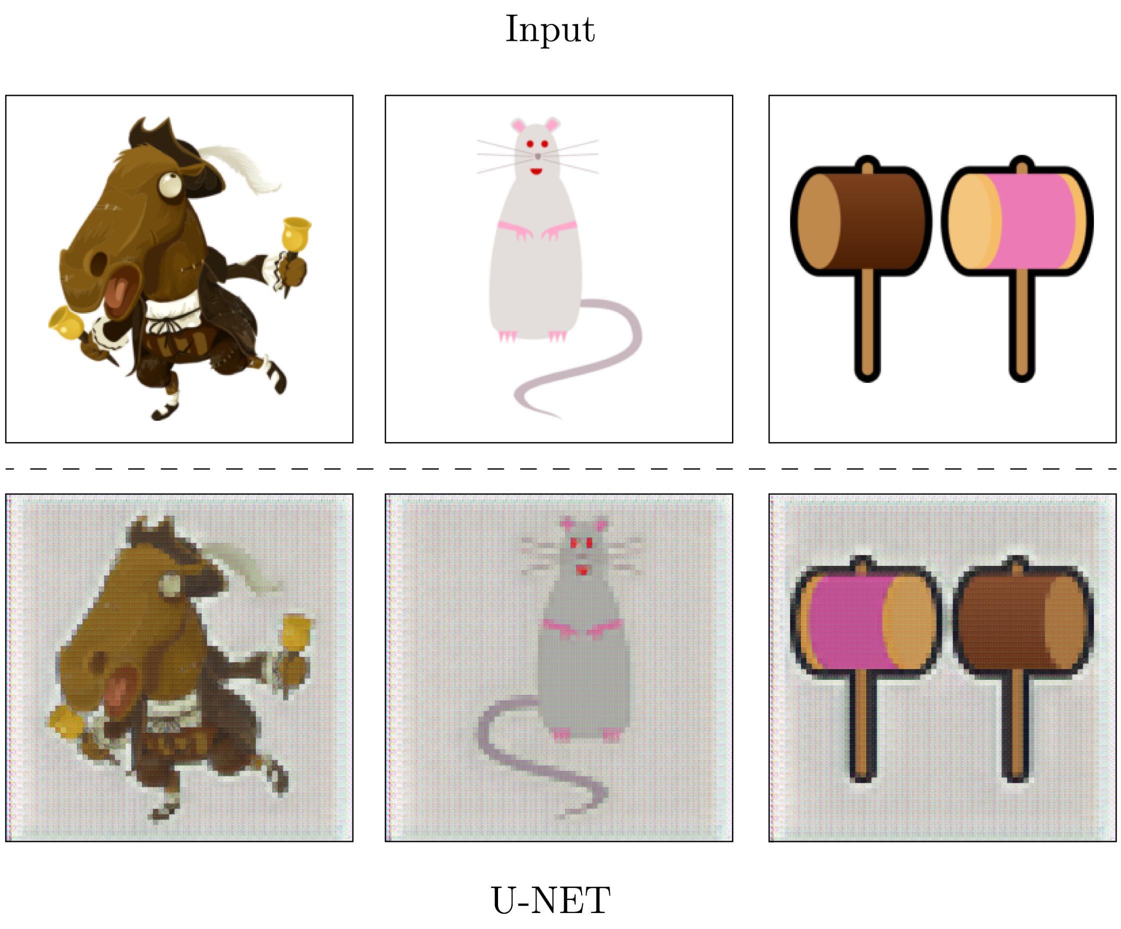


Figure 6.9.: Results from Experiment 2.1

These artifacts were more extensive and prominent than in the ResNet generator and may again be caused by the transposed convolution layers.

Experiment 2.3: Upsampling

Similar to Experiment 1.2, this experiment replaced the transposed convolution layers in the generator with nearest-neighbor upsampling and regular convolutions. The U-NET generator has 8 transposed convolutions for upsampling. All of these have a kernel size = 4, stride = 2, and padding = 1. Replacing these with upsampling layers and convolutions with stride = 1 (as done in Experiment 1.2) will not work because of a difference in resolutions. This is because of the skip-connections, where U-NET concatenates outputs from the downsampling layers to the inputs of the upsampling layers, which demands the two instances be of equal resolution. Changing the kernel size to 3 will result in corresponding resolutions. However, this change resulted in the generated images being only one color, in most cases completely yellow, entirely white, or completely black. This may be caused by the difference in kernel size between the downsampling and

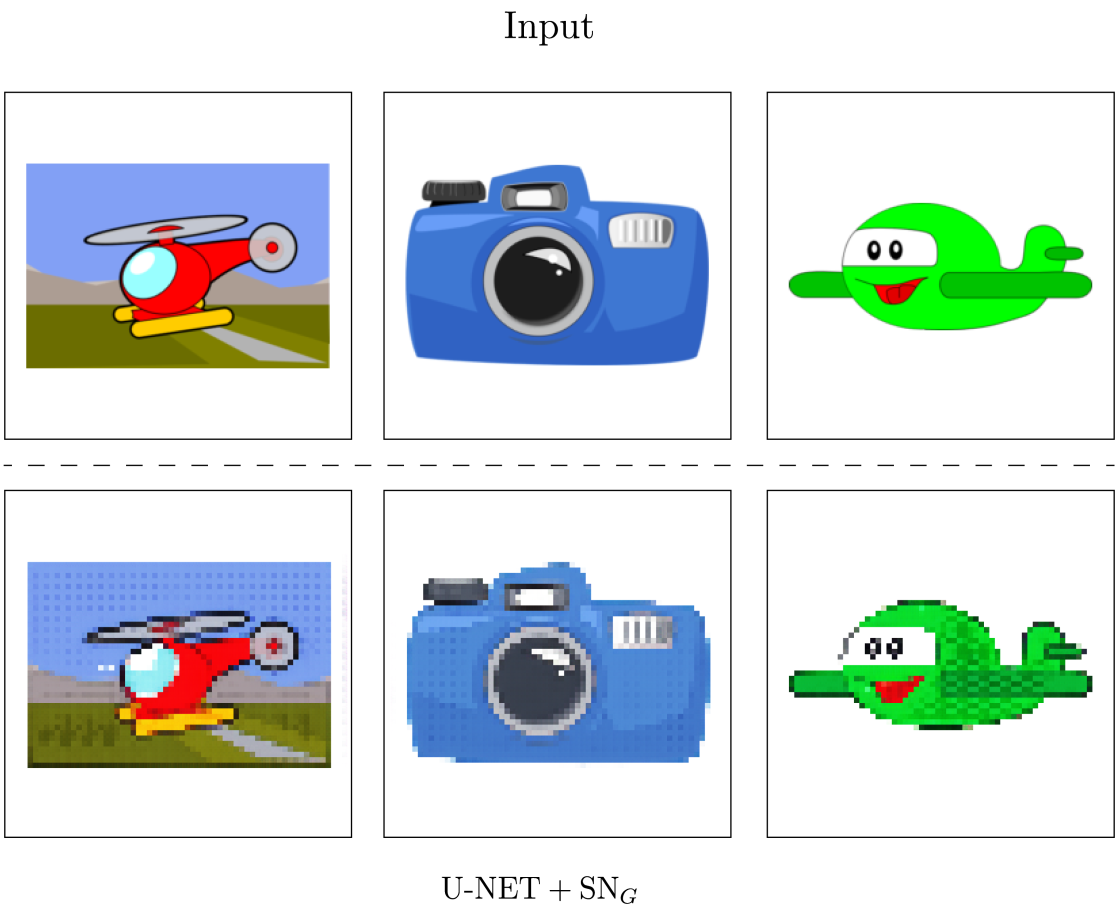


Figure 6.10.: Results from Experiment 2.2

the upsampling, such that the learned features do not correspond. The issue persisted after extensive testing with different values for kernel, stride, and padding. As a result of this, alternatives are explored in the next experiments while still using transposed convolutions.

Experiment 2.4: SSIM Losses

Since the artifacts in Figure 6.10 are only slight variations of the input colors, the cycle consistency and identity losses may not register the changes since the average pixel-wise distance is not that large, thus not penalizing them. This experiment added another loss function to the training to try to remove the artifacts while circumventing replacing transposed convolutions. This loss function is a structural similarity loss based on the SSIM evaluation metric (Section 2.7.8). The SSIM should, in theory, penalize perceptual differences, such as structure, luminance, and contrast. SSIM loss has been used in several deep learning applications where structural similarity is important (Huang et al., 2021; Zhao et al., 2017).

To implement SSIM loss, the PIQA PyTorch library was used. PIQA (François Rozet, 2020) is a collection of PyTorch metrics for image quality assessment in various image processing tasks such as generation. To use SSIM from PIQA, the color values of the images must be in the range $[0, 1]$, so images were renormalized to this by multiplying with 0.5 and adding 0.5. After renormalization, the SSIM loss was computed between input images and cyclic (reconstructed) images, such as in cycle consistency loss. SSIM outputs metrics in the range $[0, 1]$, where a higher value indicates more similarity. Since the loss function should be minimized, it was defined as $1 - SSIM(x, y)$, where x is a real sample, and y is a reconstructed sample. The loss was added to the total generator loss.

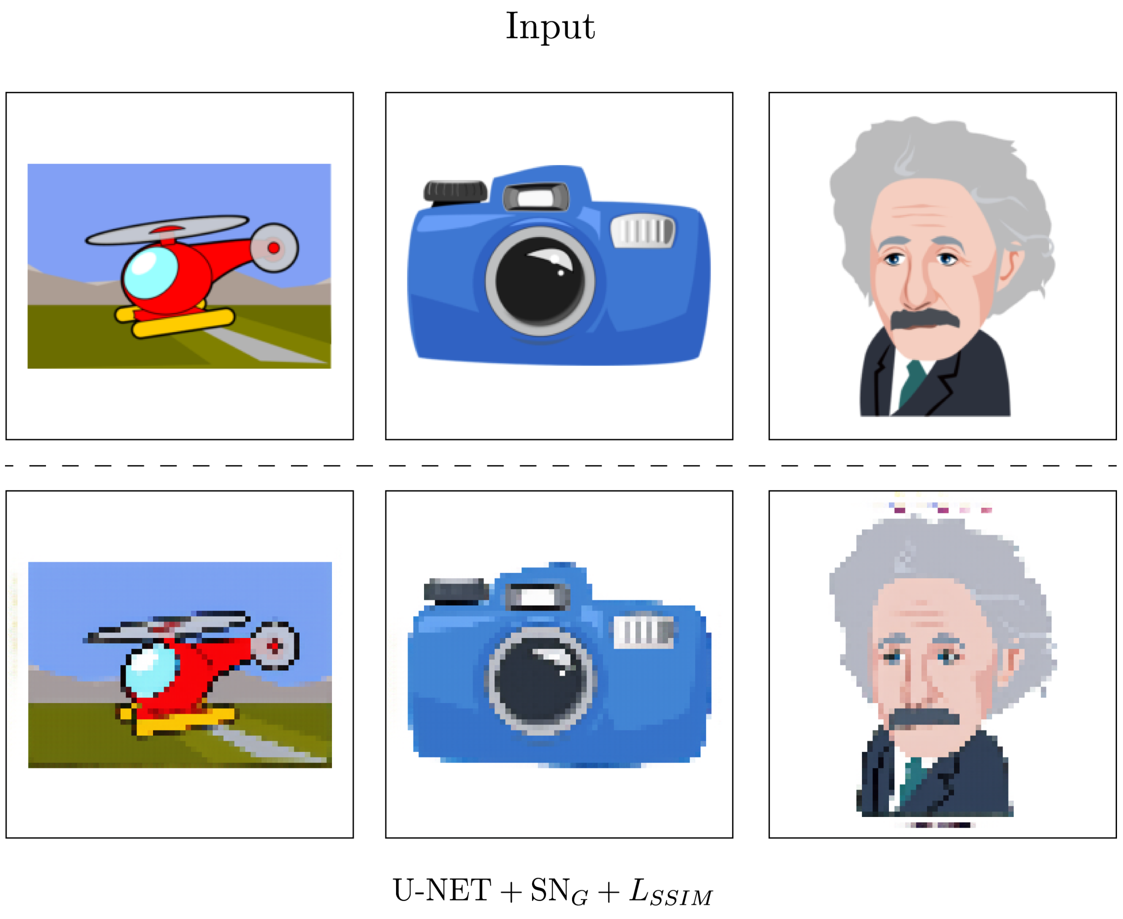


Figure 6.11.: Results from Experiment 2.4

As shown in Figure 6.11, the SSIM loss removed the artifacts present in Figure 6.10. Despite this, artifacts in the form of black lines were added to some of the images. These new artifacts around the objects can be caused by faults in the SSIM loss. Fei et al. (2012) argue that a drawback of SSIM is that it struggles with distinguishing edges and texture information. In this case, SSIM might not be able to discern the edges of the objects and thus add artifacts around them. Zhao et al. state that since SSIM was created for grayscale images, using it in the context of color images introduces an approximation. This may further explain the added artifacts and that the generators were steered in the wrong direction.

Experiment 2.5: Perceptual Losses

Engin et al. (2018) proposed the use of perceptual losses in addition to the cycle-consistency loss in CycleGAN. Perceptual losses were added to preserve the image structure, clarity, and sharpness after going through both generators. This is done by comparing high and low-level features of real images and reconstructed images. These features were extracted from the 2nd and 5th pooling layers of a pre-trained convolutional neural network VGG16 (Simonyan and Zisserman, 2014). VGG16 is trained for image recognition on the ImageNet dataset and is effectively able to extract features from any image. As mentioned in Section 3.3, Kuang et al. (2021) combined cycle-consistency loss with a topology-aware loss, which essentially is the same as the perceptual loss from Engin et al. (2018), in that they extract features from some layers from VGG16. By this, a perceptual loss function seems promising to supplement SSIM and cycle-consistency loss to further ensure sharpness, precise edges, and overall structure of the input images, and may remove unwanted artifacts.

This experiment extracted features from the 2nd and 5th pooling layers of a pre-trained VGG19 (Simonyan and Zisserman, 2014). The main difference between VGG19 and VGG16 is that VGG19 has more layers. The pre-trained VGG19 model was imported from the Torchvision library[11], and the real and reconstructed images were passed through the model to extract the features. Examples of feature maps can be seen in Figure 6.12.

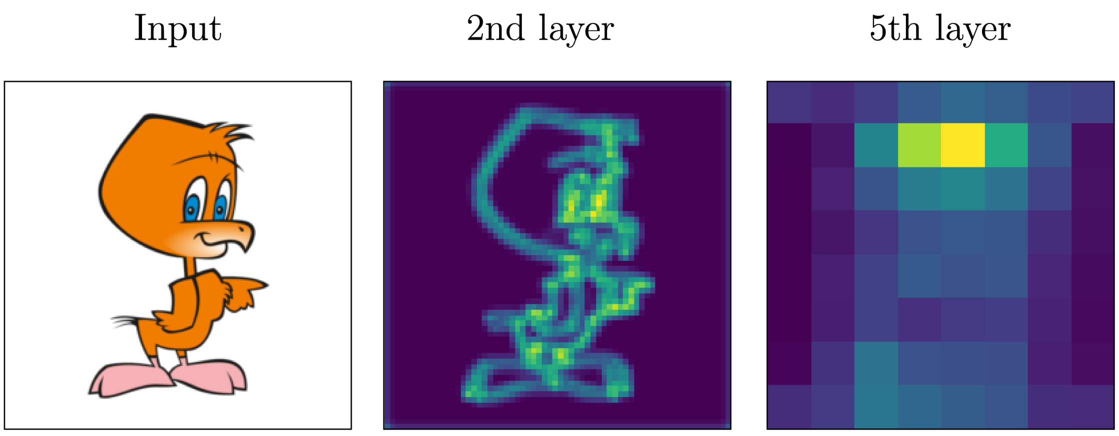
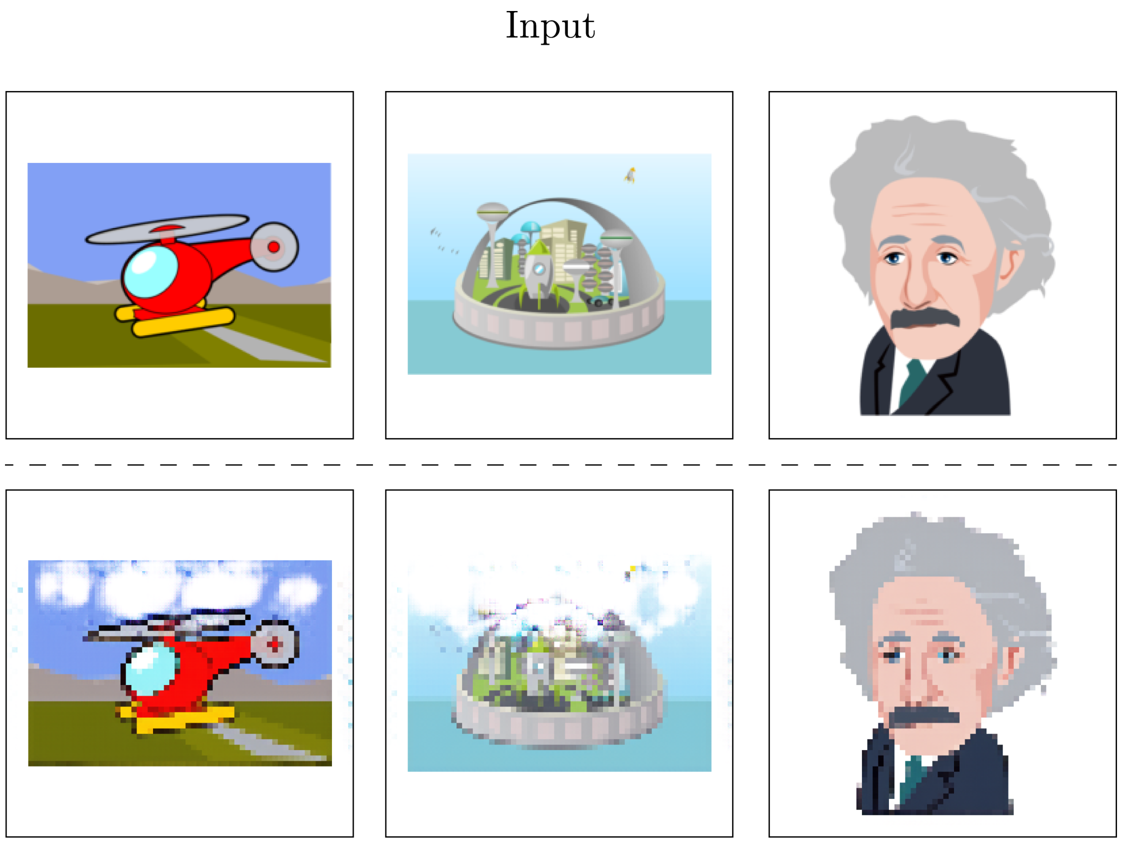


Figure 6.12.: Extracted features from VGG19

Engin et al. (2018) experienced that a high weight of perceptual loss caused a loss of color information. As a result, they weighted the perceptual loss with a factor of 0.0001.

[11]https://pypi.org/project/torchvision/

For the current experiment, a factor of 0.0001 did not have any effect on the generated images. Because of this, the weight was changed to 0.01. Adding perceptual losses with this weight effectively removed the artifacts around the images, as seen in Figure 6.13. The reason for this may be the penalization of change in low-level features. However, new artifacts occurred in the shape of white clouds. This may indicate that simply adding more loss functions can be difficult for the model to interpret since conflicting information can affect the weights. Another experiment using perceptual losses without SSIM losses was conducted, but the artifacts were still present.



U-NET + $\text{SN}_G + L_{SSIM} + L_{perc}$

Figure 6.13.: Results from Experiment 2.5

6.3.4. Experiment 3: Self-Attention

This experiment added self-attention layers to both the generator and the discriminator, as presented in Section 2.7.5. This was done because self-attention can ensure that generated images are globally consistent. Global consistency is vital in pixel art generation since every pixel should be the same size, and the use of colors should be limited. Code for the Self-Attention layers was adapted from a PyTorch implementation[12] of SAGAN (Section 2.7.5). The self-attention layers were added after the ResNet blocks, before upsampling layers in the generator, and before the second to last layer in the discriminator. Because of the added complexity of self-attention, training took approximately five times longer than in experiments 1 and 2. Furthermore, the self-attention layers significantly increased the memory requirement of the model, which may inhibit the model to train on GPUs with less than 40 GB of memory.

As shown in Figure 6.14, adding self-attention layers did not improve the results in experiments 1 and 2. This implementation led to darkened colors and frequent occurrences of minor artifacts. Experiments with adding self-attention in earlier parts of the discriminator were conducted but with worse results. Perceptual losses were tested to remove artifacts, but this resulted in unstable training. Adding spectral normalization did not stabilize the training, and artifacts were still present. No further experiments were conducted due to the long training time and no apparent improvements over experiments 1 and 2.

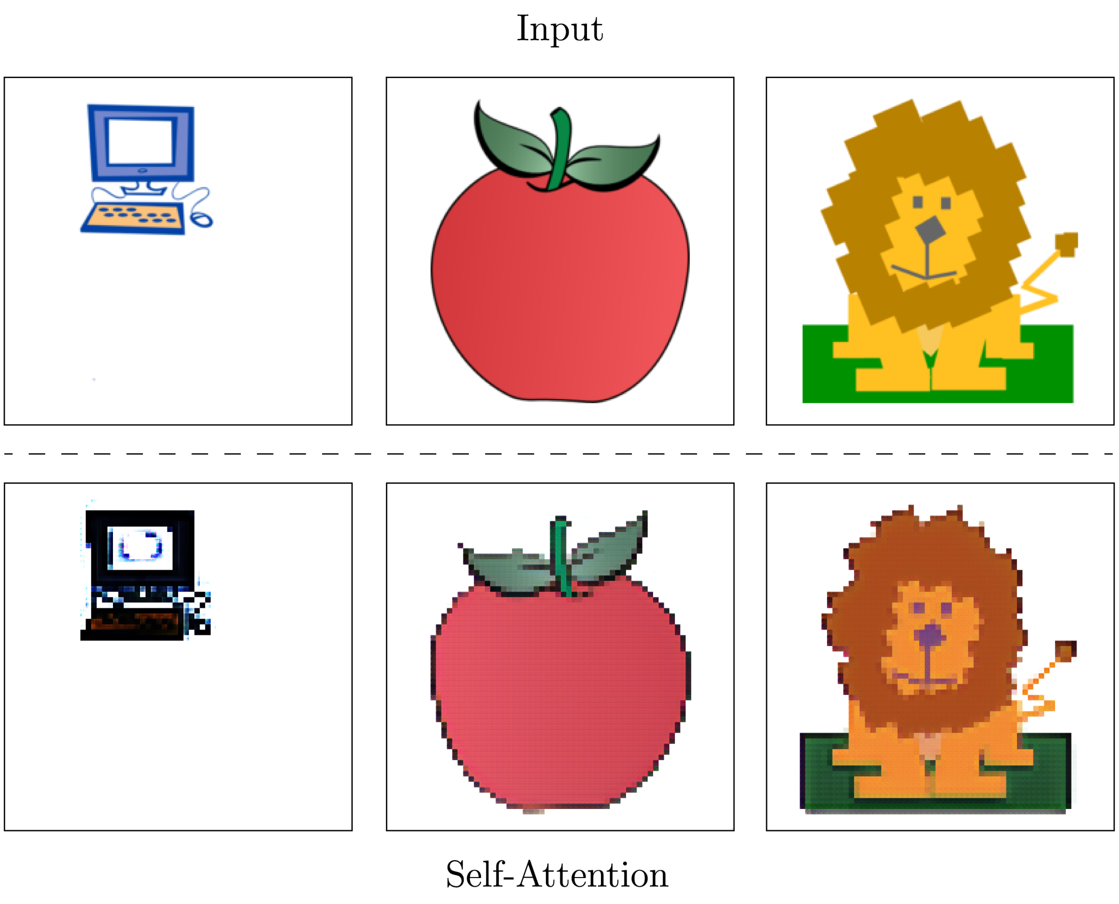


Figure 6.14.: Results from Experiment 3

[12]https://github.com/heykeetae/Self-Attention-GAN

7. Evaluation and Discussion

This chapter will evaluate, compare and discuss the results of Experiments 1, 2 and 3 from Chapter 6. First, the evaluation methods will be presented, followed by the evaluations and comparisons of the results, with discussions in view of the evaluations. Lastly, a broader discussion of the model with the best results will be given regarding advantages, use cases, and limitations.

7.1. Evaluation Method

Several evaluation metrics will be used to evaluate the results. IS, FID, and KID from Section 2.7.8 will be used to evaluate the statistics of the generated images. To supplement the statistical metrics, a new color evaluation metric for CycleGAN is introduced. Furthermore, a setup for domain expert evaluation is presented as an alternative to large-scale user-studies. These evaluation methods are introduced to answer the research question:

RQ3 *How can the performance of the models be measured and evaluated?*

7.1.1. Minimum Color Palette Distance

As discussed in Section 2.7.8, there is no perfect evaluation metric for GANs. This is partly because the terms of quality of generated images greatly vary depending on the domain. IS, KID, and FID measure the statistics of the generated images and are applicable to all types of images but do not specifically evaluate the stylistic demands of pixel art. Quantifying stylistic rules such as aliasing, dithering, and shading is challenging, and human visual inspection is required instead. The use of colors in an image, however, can be measured since the pixel color values are quantities. An essential part of style-transferring from cartoons to pixel art is preserving the input images' colors. Therefore, the colors of the input and output images should be compared to measure and ensure color consistency.

One way to compare the colors of two images is to pair-wise compare the color values of all pixels. This, however, would not account for the change in texture in the output image since some color values are supposed to change throughout the image when converting from cartoon to pixel art. A better solution would be to extract the images' color palettes by extracting the top most frequent or most representative colors. These color palettes can then be compared to measure how much they differ. As a result, this thesis developed a method for comparing the color palettes of input and generated images.

One way to make a color palette for an image is to extract the N most frequent colors. This can be problematic if the image contains large parts with many colors and shading. As seen in Figure 7.1, the resulting color palette is not representative of the whole image when the image has many different colors. A better solution is to use K-means clustering to extract representative dominant colors from the image (Lertrusdachakul et al., 2019).

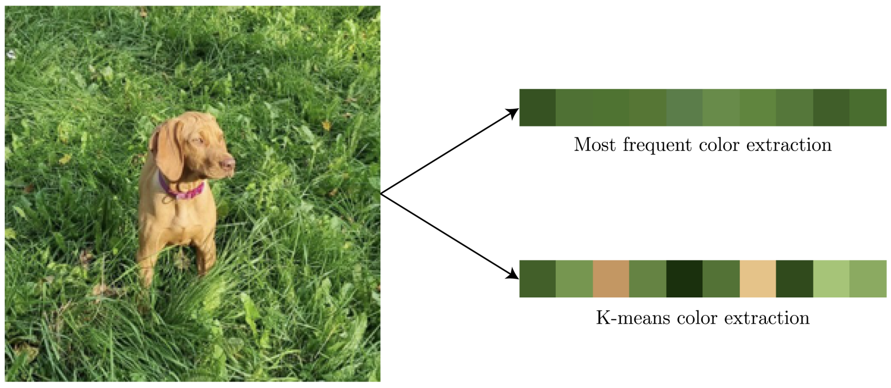


Figure 7.1.: K-means vs frequent color extraction.

K-means clustering is an unsupervised machine-learning algorithm for grouping data points (Na et al., 2010). K-means clustering can group similar color values into K clusters to extract a palette from an image. First, the algorithm creates K randomly chosen centers in the color space, with K being predetermined. Secondly, each data point (color value) is assigned to its nearest center, measured by Euclidian distance (distance between the coordinates of data points). Then, the averages of each of the clusters are computed, new center points are chosen, and all data points are assigned to a new center. This is done iteratively until a stopping criterion is met. After enough iterations, all color values are assigned to fitting clusters, and the centers of the clusters are chosen as representative color values (Na et al., 2010). This results in a color palette of K colors. This color palette is more representative than extracting the most frequent colors, as shown in Figure 7.1.

To compare two color palettes, the distances from each color in the source palette to each color in the target pallet must be measured. The Euclidian distance of each RGB channel can be used to compare two colors C_1: (R_1, G_1, B_1) and C_2: (R_2, B_2, G_2), and can be defined as $distance(C1, C2) = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$. However, Euclidian distance does not conform well with the human perception of colors (Lv and Fang, 2018), and may result in high distances for perceptually similar colors and vice versa.

CIEDE2000 is a color difference formula that measures the human perceptual difference between two colors and is a commonly used metric in the field of color science. Instead of using the RGB color space, CIEDE2000 uses the CIELAB (or L*a*b*) color space. Here, colors have three values, L*, a*, and b*, where L* represents perceptual lightness and a* and b* represent combinations of the four colors red, green, blue, and yellow (Johnson and Fairchild, 2003). These values are designed to represent human perception better.

CIEDE2000 uses this color representation to calculate the distance, and the complete CIEDE2000 algorithm can be found in Sharma et al. (2005). Since CIEDE2000 measures distance, a lower score indicates more similar colors.

After extracting the color palettes, each color in the source palette can be compared to each in the target palette. The distance between each color can be measured, and the minimum distances can be recorded and averaged. This will result in similar palettes having a smaller average distance than contrasting palettes since each color in the source palette can find its closest match in the second palette. An issue with this approach is that the operation is performed in only one direction, e.g., comparing the source to the target palette, but not vice versa. Pan and Westland (2018) use the same approach to measure color palette distance but alleviates this issue by also performing the operation from target to source and averaging the two results. The total average difference between the two palettes is then obtained.

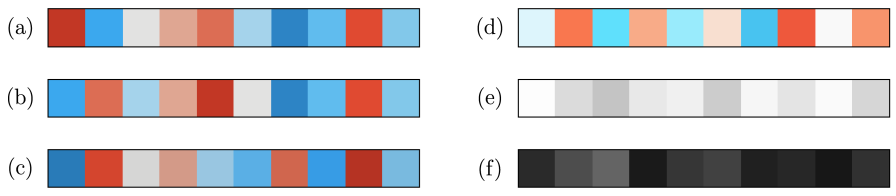


Figure 7.2.: Palettes

	(a) and (b)	(a) and (c)	(a) and (d)	(e) and (f)
Distance	0.0	3.228	7.717	47.389

Table 7.1.: CIEDE2000 distance between palettes.

Figure 7.2 and Table 7.1 show the CIEDE2000 color difference between palettes. The scale is $[0, 100]$, where 0 indicates equal colors, and 100 indicates completely different colors. Figure 7.2 (a) and (b) are the same palette but have different ordering. Despite the order of the colors, the distance is still 0. Figure 7.2 (c) is a slight variation of (a), with a distance from (a) close to 0. Figure 7.2 (d) is a more distinct variation, with a higher distance from (a) than (c) has. Figure 7.2 (e) and (f) differ significantly and thus have a high distance between them.

Extracting color palettes from images using K-means, converting from RGB to $L^*a^*b^*$, measuring the distance between the colors using CIEDE2000, and then averaging over all colors in the palettes is combined into an evaluation metric. The algorithm for the constructed color palette evaluation metric is shown in Algorithm 2. Here, the algorithm compares the color palette difference between the input image from the source domain to the generated image in the target domain. This is done for all images in the test set, with the mean (average) palette difference being the total score. A score closer to zero indicates better color consistency. The number of K-means clusters k determines the number of colors in the extracted color palettes. The implementation of the algorithm can be found in the codebase for this thesis[1]. In the implementation, palette extraction with K-means clustering uses the Scikit-learn library[2], conversion between RGB and LAB uses the Scikit-image library[3], and the CIEDE2000 distance uses an implementation[4] of the computations from Sharma et al. (2005).

Algorithm 2 Minimum color palette distance (MCPD) algorithm.

```
$k \leftarrow$ number of K-means clusters
for $n$ number of images in test set do
    - Extract palette of $k$ colors from source image using K-means clustering
    - Extract palette of $k$ colors from generated image using K-means clustering
    - Convert palettes from RGB to $L^*a^*b^*$ color space
    for $k$ colors in source palette do
        - Get distance to each color in generated palette using CIEDE2000
        - Save minimum distance
    end for
    for $k$ colors in generated palette do
        - Get distance to each color in source palette using CIEDE2000
        - Save minimum distance
    end for
    $\textit{image score} \leftarrow$ mean of saved distances for current image
end for
$\textit{score} \leftarrow$ mean of $n$ image scores
```

7.1.2. Domain Expert Survey

As presented in Section 2.7.8, conducting user studies to measure the realness of generated images is popular. This is often done by having a (preferably large) number of people guessing whether the generated images are real or fake. This works well for cases where the study participants have adequate knowledge of the target domain, such as real-life photos or paintings. Despite pixel art being a well-known art form, the assumption that the study participants inherently know how to distinguish between real and fake pixel art can not be made. This is because the stylistic rules of pixel art are not well known, and it takes practice to judge an image by them.

The most important rule of pixel art is that the image only comprises distinguishable pixels, where every pixel is of each size, and each pixel is of one color. Determining this can be done through visual inspection by only one person without needing a user survey. More detailed rules, such as consistency in the line art, anti-aliasing, shading, highlights, and shadows, should preferably be evaluated by a pixel art domain expert. As with all

[1]https://github.com/ekpete/masterthesis
[2]https://pypi.org/project/scikit-learn/
[3]https://pypi.org/project/scikit-image/
[4]https://github.com/lovro-i/CIEDE2000/tree/master

art forms, pixel art quality is subjective, and more than one expert should be consulted to ensure unbiased evaluations.

Professional pixel artists can be considered experts in the pixel art domain. To gather domain experts for evaluating the results in this thesis, the online marketplace for freelance services, Fiverr[5], was used. Several top-rated pixel artists on the site were contacted, and two agreed to evaluate the results. The first was Italivy Cortes, a pixel artist with five years of experience and an average rating of 4.9/5 stars from 533 reviewers on Fiverr. Some of their artwork is shown in Figure 7.3. The second was Nicola Di Concilio, a pixel artist with four years of experience and an average rating of 5/5 stars from 923 reviewers. Some of their artwork is shown in Figure 7.4.

Figure 7.3.: Pixel art created by artist Italivy Cortes. Printed with permission from the artist.

Figure 7.4.: Pixel art created by artist Nicola Di Concilio. Printed with permission from the artist.

The expert evaluation survey aims to determine the overall quality of the generated artwork concerning the stylistic rules of pixel art. For evaluation, the 180 generated images from the test set will be presented, and the following questions will be asked:

1. Overall, how well do these artworks follow the rules of the pixel art style (regarding line art, color use, shading, dithering, aliasing, etc.)?

[5]https://www.fiverr.com

2. Overall, are these artworks believable to be created by a human artist? Why, why not?

3. Do any of the images stand out as high quality, and do any stand out as low quality?

4. Do you have any additional comments about the pixel art presented?

The experts will be told that the artwork was computer generated and instructed to focus on the overall impression and quality of the pixel art. The answers to the questions will be presented in Section 7.2.4.

7.2. Evaluation

This section presents the evaluations of the sub-experiments with the best results from each of the four experiments in Chapter 6. The best sub-experiments are selected based on their respective sections' discussions and visual outcomes. For Experiment 0 (Section 6.3.1), the base CycleGAN model with the full collected dataset is selected and is termed "Base CycleGAN" in the comparisons in this section. For Experiment 1 (Section 6.3.2), Experiment 1.4 with 9 residual blocks and nearest-neighbor upsampling in the generator, and spectral normalization in the discriminator is selected and is termed "ResNet9 + NN_G + SN_D". For Experiment 2 (Section 6.3.3), Experiment 2.4 with the U-NET generator, spectral normalized discriminator and SSIM loss function is selected and is termed "U-NET + SN_D + L_{SSIM}". For Experiment 3 (Section 6.3.4), Experiment 3.1 with added Self-Attention layers is selected and is termed "Self-Attention". The results from the chosen sub-experiments will be compared using visual inspection, statistical evaluation, color evaluation, and domain expert evaluation. The presented generated images are the model outputs with no modifications or corrections.

7.2.1. Visual Inspection

Randomly chosen samples from the results are shown in Figure 7.5. To show how downsampling techniques are insufficient to create pixel art, nearest-neighbor interpolation is included. From all samples, it is evident that interpolation does not result in consistent line art, and details are often lost. Other downsampling techniques, such as bilinear interpolation (not included in the comparison), result in blurry images with no form of line art.

The base CycleGAN model has more consistent line art than nearest-neighbor interpolation, as evident in sample (b) in Figure 7.5. However, black lines usually fade with their surrounding colors, as seen in samples (b) and (c). Furthermore, checkerboard artifacts can be seen on closer inspection, as discussed in Section 6.3.2.

ResNet9 + NN_G + SN_D is a significant improvement over the base model in that the black line art is clear and consistent, with a uniform black color. This is evident in the contour lines of the objects in samples (b), (c), and (d) in Figure 7.5. Furthermore, the colors are better retained, no visible artifacts are present and essential details are

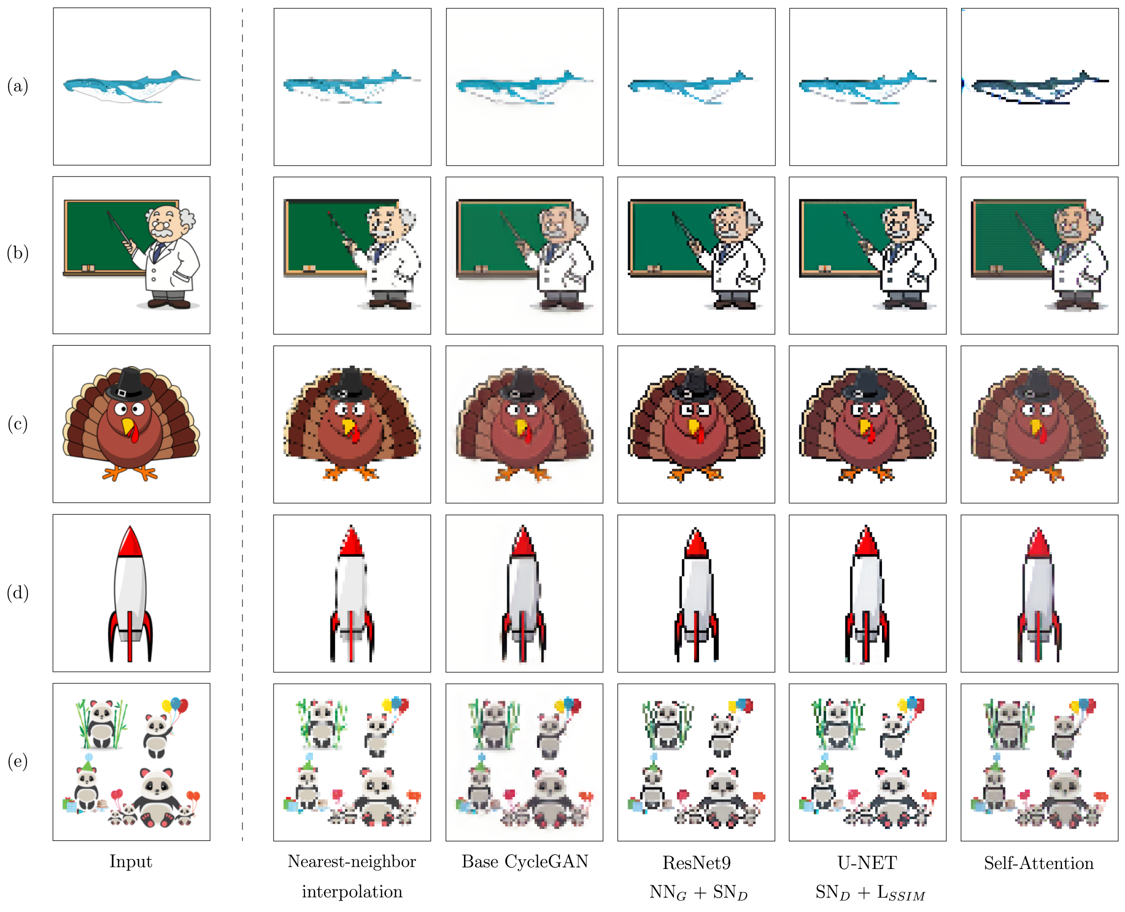


Figure 7.5.: Results from the four models from Experiments 0, 1, 2 and 3. The results of downsampling the input image with nearest-neighbor interpolation is included.

translated. Equal objects in the input image have an equivalent translation, evident in the eyes in sample (c) and the curved edges in sample (d). One issue can be seen in sample (a), where the thin black line at the bottom of the object is not kept. This can also be seen in the top-right balloons of sample (e). However, this issue only occurs in a few instances throughout the test set, where some indistinct narrow lines are affected.

U-NET + SN_D + L_{SSIM} is also an improvement over the base model because of more consistent black lines and better color retention. In addition, this model includes narrow lines, as seen in samples (a) and (e) in Figure 7.5. This may be because of the skip connections in the U-NET generator, which can transfer more features from the input image than the ResNet generator. Despite this, the translation of equal objects from the input image is inconsistent. The eyes in sample (c) are of different shapes, while the vertical edges in sample (d) have different curves. Despite not being evident in the samples presented in Figure 7.5, several generated images from the test set have unwanted artifacts around the objects. An example of this was shown in Figure 6.11 in Section 6.3.3.

Adding Self-Attention layers did not improve the results of Experiments 2 and 3. As seen in sample (a) in Figure 7.5, colors are distorted, and artifacts are added. Objects and details are translated similarly to the base model, but samples (b) and (c) show checkerboard artifacts and inconsistent line art. In addition, for many images in the remainder of the test set, the self-attention mechanism often caused light colors in images to be translated darker.

7.2.2. Statistical Metrics

To further compare the selected models, the statistics of the results from the experiments will be compared to the statistics of the real test samples from the pixel domain using FID and KID (Section 2.7.8). Since the dataset is unpaired, the statistics of the images as a whole will be compared, not individually. The results from the two metrics are shown in Table 7.2.

Model	IS	FID	KID
Base CycleGAN	**3.958±0.569**	162.447	0.048 ± 0.010
ResNet9 + NN$_G$ + SN$_D$	3.679 ± 0.600	**132.492**	**0.019±0.006**
U-NET + SN$_D$ + L$_{SSIM}$	3.726 ± 0.649	132.783	0.020 ± 0.006
Self-Attention	3.682 ± 0.276	162.329	0.056 ± 0.010

Table 7.2.: Inception Score (IS), Fréchet Inception Distance (FID) and Kernel Inception Distance (KID) of the generated test set images from four models. IS computes statistics of only the generated images. KID and FID compute and compare the statistics of generated images to images from the real domain. FID show the total score, while IS and KID show the mean scores with standard deviation (variability around the mean).

Inception Score (Section 2.7.8) was included as a supplement to FID and KID but has been considered unreliable for evaluating GANs (Barratt and Sharma, 2018). Since IS stems from the Inception network trained in the ImageNet dataset, applying it to GANs trained on other datasets gives misleading results. As shown in Table 7.2, the base CycleGAN model achieves the highest Inception Score, which contradicts the discussion in the visual inspection. The score for the real (non-generated) images from the pixel art domain is 3.515 (lower than all the generated images), which suggests that IS is not a suitable metric for this task. Because of this, more focus should be given to the FID and KID scores.

Similar to IS, FID is not without limitations. As stated in Section 2.7.8, the minimum amount of test images is usually above 10,000 for evaluation with FID. The test set used in this thesis only contains 180 images. Because of this, the FID results may be misleading. KID does not have a recommended minimum but may suffer from high variance when evaluating a small number of images. However, it is still possible that FID and KID can indicate the level of similarity between the generated and the real images. As shown in Table 7.2, ResNet9 + NN$_G$ + SN$_D$ achieves the lowest KID and FID scores, while U-NET + SN$_D$ + L$_{SSIM}$ has only marginally higher scores for both metrics. The base and Self-Attention models have significantly higher scores, indicating a larger deviation from the real domain. These results agree with the results from the discussion in the visual inspection, and suggest that the ResNet9 model outperforms the other models in generating pixel art images that resemble real-life artwork.

7.2.3. Color Palette Evaluation

Table 7.3 shows the MCPD score of the models for different values of k. k is the number of colors in the palettes being compared, and more than one value is shown to test whether the scores vary depending on k. As shown in the table, the scores are consistent. As shown in Section 7.1.1, a score close to 0 indicates good color retention. A score between 0 and 2 indicates that the generated images have colors perceptually identical to the source images. Images with scores around 2-3 begin to have slightly perceivable color deviations, while images with scores around 5 have visible color deviations.

Model	MCPD $k = 5$	MCPD $k = 10$	MCPD $k = 20$
Base CycleGAN	5.650±1.909	5.799±1.244	5.3218±1.031
ResNet9 + NN$_G$ + SN$_D$	5.435±2.181	5.546±1.273	5.137±1.012
U-NET + SN$_D$ + L$_{SSIM}$	**4.665±2.069**	**5.025±1.142**	**4.828±1.026**
Self-Attention	7.254±2.408	7.239±1.961	6.912±1.764

Table 7.3.: Minimum Color Palette Distance (MCDP) for all four models with three different palette lengths. Standard deviation (variability around the mean) is included.

As a baseline, the MCPD score (k=5) for nearest-neighbor interpolation is 1.21. No colors are changed during the interpolation, apart from the ones lost in the down-sampling. Ideally, the style transfer models should achieve an equal score. However, the results in Table 7.3 show that even though the overall color retention is adequate, it is still perceptually visible. U-NET + SN$_D$ + L$_{SSIM}$ achieves the lowest scores for all palette lengths. This may be because of the skip connections in the U-NET, as colors are more consistently transferred from the input to the output. ResNet9 + NN$_G$ + SN$_D$ scores moderately worse than the U-NET model, while the base CycleGAN model scores slightly worse than the ResNet9 model. The scores of around 4-5 indicate visible color differences between the input and the generated images. The Self-Attention model has the worst scores of around 7 across the values of k, which indicate obvious color variations between the input and the output.

7.2.4. Domain Expert Evaluation

The U-Net and ResNet9 models perform best out of the four models by the valuations in the previous sections. Furthermore, color retention was better in the results from the U-NET model than the ResNet9 model. However, the ResNet9 model performed slightly better than the U-NET model by visual inspection and statistics. In addition, the ResNet9 model did not have any visible artifacts in any generated images from the test set. Artifacts occurred in some generated samples from the U-NET model. As a result, ResNet9 + NN_G + SN_D is selected as the best-performing model from the experiments, and the generated images will be further evaluated using domain experts. Only one model is chosen because the goal of the expert evaluation is to compare the generated images to real pixel art and not to compare the models against each other, which was done in the previous sections. The pixel art experts were given the 180 generated images from the test set and answered the survey presented in Section 7.1.2. A summary of the responses is presented in this section, and the complete survey answers are given in Appendix A.

The two experts provided their observations and opinions regarding the generated pixel art. In response to the first question about how well the artworks followed the rules of the pixel art style (such as line art color use, shading, dithering, anti-aliasing, etc.), they pointed out several issues. The first expert stated that the line art was uneven for the most part, similar to basic outlines when starting a rough draft of an illustration. Anti-aliasing was sometimes used to smooth out the line art, but it was misapplied in many examples. Sometimes it was applied to the outside objects, which gave an unfortunate "glow" effect. The color palette was kept to a minimum for most examples, which was positive, but shading techniques were not applied correctly, making the images look flat. In the examples where shading techniques were applied, many only had a gradient effect, usually known as pillow shading and banding. Beginner-level pixel artists often use these shading techniques, and more advanced techniques, such as dithering, were not applied to any of the presented images.

The second expert gave ratings for the techniques used. They gave an overall adherence to the rules as 5 out of 10, with shading receiving a rating of 3, using colors a rating of 5, and line art a rating of 8. They also commented on shading, similar to the first expert. They stated that using gradients is against the pixel art guidelines, especially when applied to characters and smaller objects. Gradients should instead be applied to backgrounds. They commented that the usage of colors was excessive and wanted smaller color palettes. Regarding anti-aliasing, they stated that a darker shade of the line should be used rather than only gray colors.

For the second question, which addressed whether the generated images were believable to be created by a human artist, the first expert noted that while the artworks resemble pixel art, they appear to be the work of a beginner-level artist due to incorrect application of techniques. The second expert stated that the images looked like they were taken from the internet and made by beginner level pixel-artists. Furthermore, they had a "retro" feel and looked good.

When asked to point out examples of high and low-quality images, the first expert did not regard any of the images as high-quality pixel art. However, six of the images, amongst others, were classified as illustrations of an improving beginner pixel artist. The second expert observed that 44 of the images looked like they were created by a human and classified six as high quality. The images that the experts rated highest are shown in Figure 7.6.

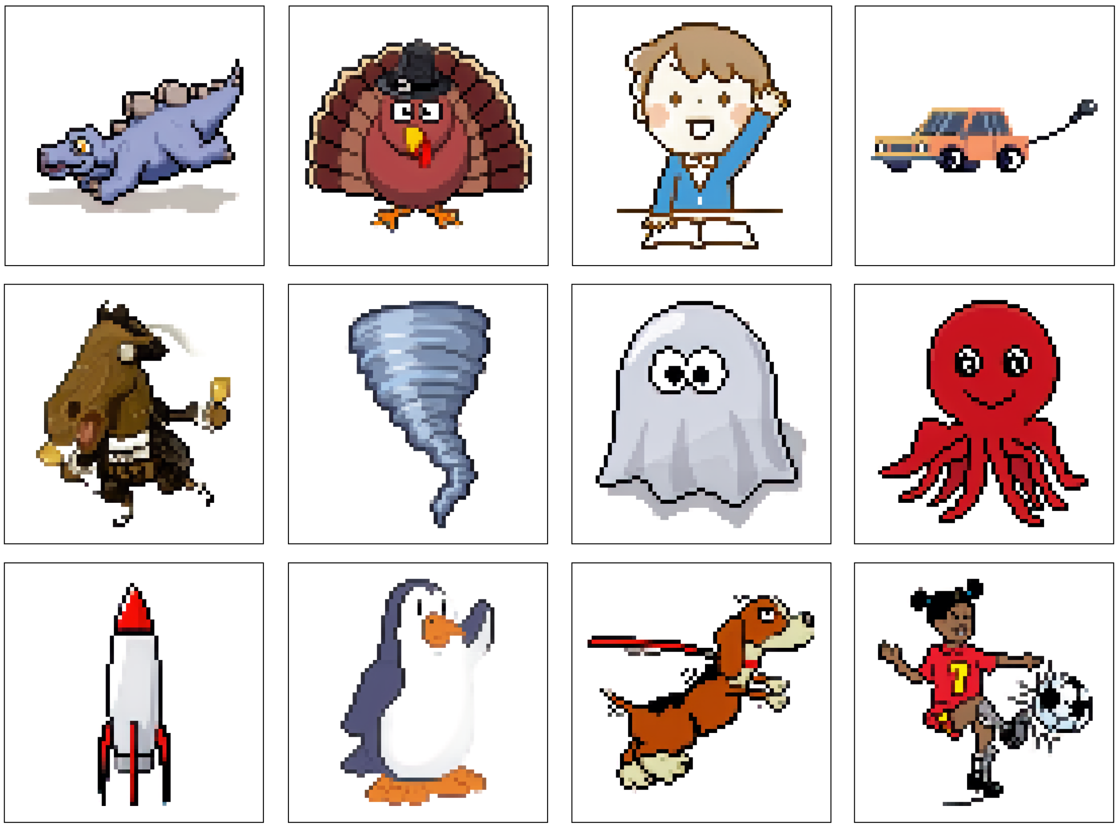


Figure 7.6.: Domain experts' selection of highly rated generated pixel art images.

Lastly, the first expert was impressed with the level of pixel art the model was able to produce. They stated pixel art is often hard for beginners and even intermediate-level pixel artists. Furthermore, they mentioned that other AI-generated pixel art they have seen often looks computer generated. They expressed that the presented images were perceived as art created by a beginner artist rather than a computer. The second expert gave some final warnings on the choice of datasets since most pixel art available online does not follow pixel art rules. They advised that more high-quality pixel art should be located and used to get more professional-looking results.

7.3. Discussion

The expert evaluators noted several shortcomings of the generated images regarding the stylistic rules of pixel art. This was due to the absence of proper anti-aliasing, lack of professional shading techniques, and inconsistent line art. They explained that the use of gradients between colors in the generated images is rarely seen in professional pixel art. The shortcomings can partly be explained by the fact that the model was trained on relatively simple pixel art, with few occurrences of techniques such as anti-aliasing and advanced shading. It is not possible for a model to learn techniques it has not been exposed to. A training dataset with more details and advanced artwork may generate higher-quality images. Still, a too complex dataset might inhibit the model from learning the basic rules, such as equal-sized and distinguishable pixels. This was shown by Yang et al. (2022), as described in Chapter 3, where the model was trained on large landscape pixel art and could not learn the basic rules. Additionally, collecting a data set with more images that use advanced techniques may be a solution, but locating high-quality publicly available artwork may be difficult.

Furthermore, it is worth noting that the quality of a generated image (after training) depends on the features learned by the model and the quality of the input image. A poor input illustration might result in poor pixel art since the model only changes the style, not the actual objects and other image content. One of the experts stated that the generated pixel art appeared as if they were taken from the internet. This is partly correct since the input images are simple illustrations scraped from an open-domain source. This may suggest that the input's simplicity dictates the output's simplicity. Even though the model was trained on simple illustrations, it can generate more advanced pixel art by translating more advanced illustrations. An example of this is shown in Figure 7.7, where a detailed painting is translated to pixel art.

In Figure 7.7, even though the model does not directly apply advanced techniques, the generated image can be perceived as more complex pixel art caused by the complexity of the input illustration. Furthermore, compared to downsampling, lines are more precise and uniform in the generated image, as seen on the floor and stairs in the bottom left. Colors are more uniform, as seen in the sky, and small objects are easier to discern, as seen in the sailboats. Another thing to note in this example is that since the model is fully convolutional, it can take images of any size as input. The image in Figure 7.7 is of size 764x600, which is almost three times the size of the training images. Despite the image's increased resolution, the model can still produce square pixels of uniform size.

Despite the shortcomings in the use of advanced techniques, both expert evaluators expressed that the model generated pixel art at a human beginner level. This shows that the model can provide a starting point for an artwork, where only a sketch, a simple cartoon, or a more complex illustration can be used as input. Drawing a simple cartoon has a lower threshold than creating pixel art for inexperienced artists. The generated pixel art can be further edited and cleaned up by, for example, adding anti-aliasing and applying more advanced shading techniques. This can save time and effort in generating artwork, which can be valuable when creating large quantities. This can, for example,

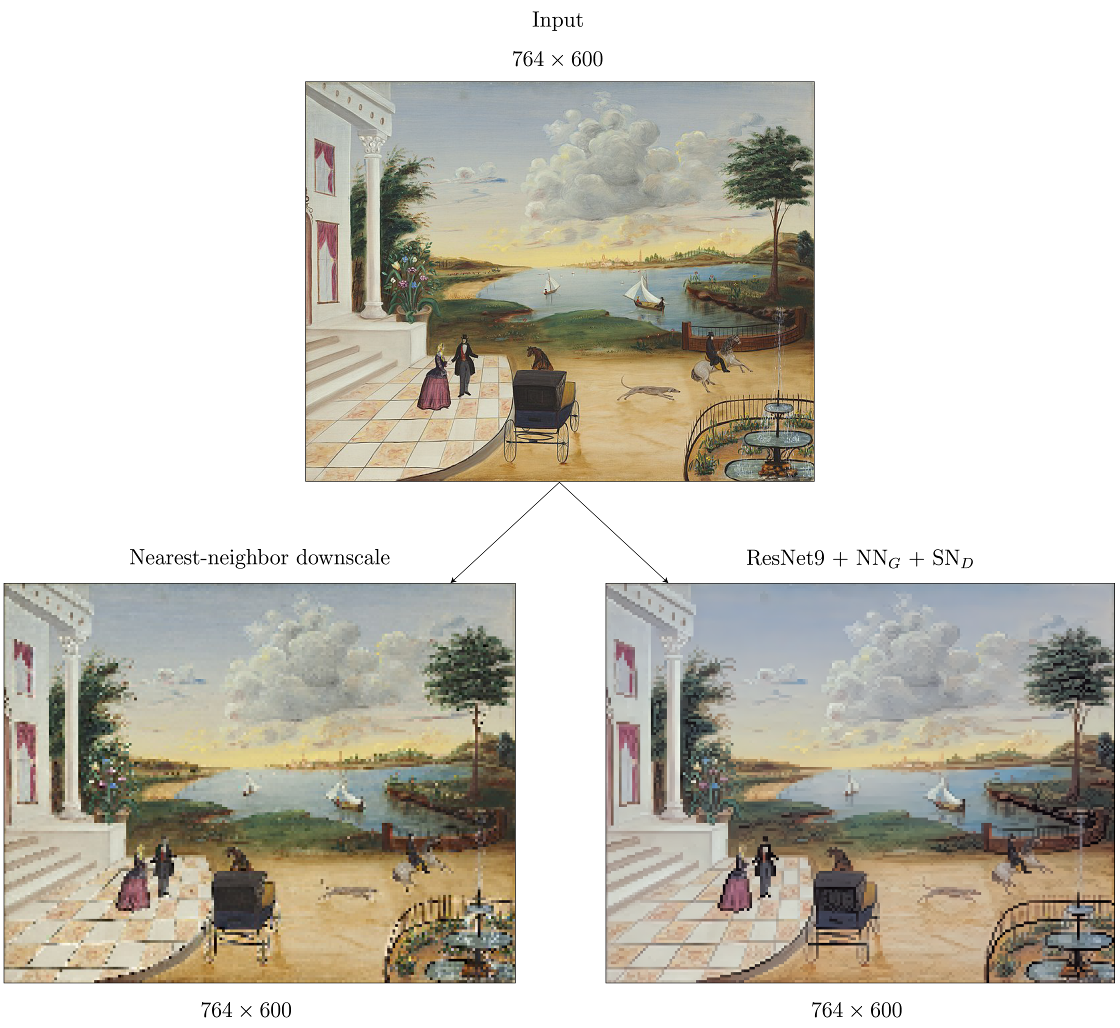


Figure 7.7.: A comparison of the ResNet model and nearest-neighbor interpolation on a input painting of size 764 × 600.

be useful for video game developers as they often need to make many artworks when creating assets for their games.

When manually editing or using the generated images for game assets, they should be downscaled to their proper resolution. As explained in Section 6.2, when the model was trained, the pixel art images were upscaled by a factor of 4 using nearest-neighbor interpolation. This entails that one pixel is composed of four actual pixels. To get the proper resolution of the generated image without the loss of information, such that each pixel is composed of only one pixel, one can apply a nearest-neighbor downscaling with a factor of 1/4. The images can also be upscaled to larger resolutions as long as the factor is a multiple of 4. This ensures that all pixels are squares of equal sides. Another advantage of the model is that it can generate pixel art of different detail levels depending on the size of the input image. By downscaling the input to a smaller size, fewer pixels will represent the content. This is shown in Figure 7.8.

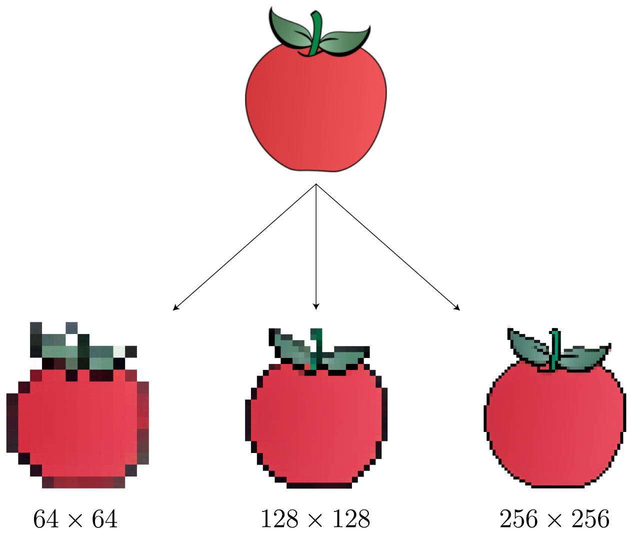


Figure 7.8.: Varying the resolution of the input image results in different pixel art styles.

Because of the cycle consistency concept, the model can translate pixel art into cartoons. This feature is vital during training, but it does not have any practical applications because of the low quality of the generated cartoons. Some examples are shown in Figure 7.9.

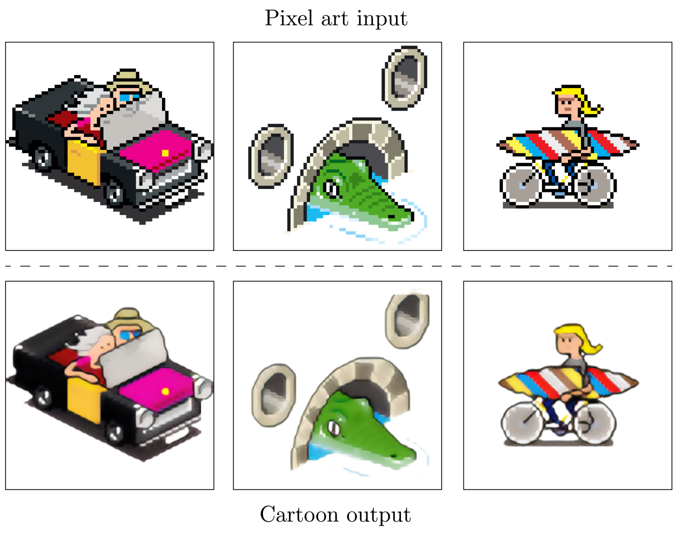


Figure 7.9.: Pixel art translated to the cartoon domain. This is possible because of cycle consistency.

As previously discussed, a limitation of the model is the inability to produce pixel art that uses advanced techniques. The model is trained on relatively simple pixel art, thus outputting simple pixel art. Simple pixel art, however, is an entirely valid form of pixel art. Pixel art can have different substyles, and the artwork in the training data only has a few. The CycleGAN model may have issues learning more than one style, but an alternative is to create several models trained on different datasets with different pixel art styles. More alternatives are discussed in Chapter 8.

Another limitation is that the model cannot ensure complete color retention. As previously shown, the colors do not majorly deviate, but there are still visible differences. Collecting a larger dataset with more diverse colors may further ensure color retention, as discovered in the dataset experiment (Section 6.3.1). Another possible solution is to extract the color palette from the input and generated images during training, compare the two, and penalize large distances. This can be done using K-means clustering and CIEDE2000 distance, effectively turning the MCPD algorithm (Section 7.1.1) into a loss function. To do this, all the operations in the algorithm would need to be converted into mathematically differentiable functions. This would, however, be a computationally expensive operation, as palette extraction would need to be done for every iteration during training. Even though the 1,620 input images' palettes would only need to be computed once and then saved, the palettes of the generated images would need to be recomputed every epoch. This would amount to over 200,000 K-means clustering runs, which may result in drastically longer training time. An alternative to this could be to compare color histograms, as done by Coutinho and Chaimowicz (2022), presented in Chapter 3.

8. Conclusion and Future Work

This chapter presents the main contributions of this thesis in relation to the field of pixel art synthesis and to the research questions presented in Chapter 1. Lastly, suggestions for extensions, improvements, and future work are presented.

8.1. Contributions

As discussed in Chapter 7, this thesis succeeds in synthesizing quality pixel art from cartoons. CycleGAN with a ResNet generator with nine residual blocks, nearest-neighbor upsampling, and a spectral normalized discriminator outperformed the other model architectures from the experiments, as evidenced by the visual inspection and statistical metrics. Pixel art experts further evaluated this model and concluded it could generate pixel art on the same level as a beginner human pixel artist.

The work in this thesis contributes to the field of pixel synthesis by presenting suggestions for viable architectures options for CycleGAN to generate pixel art. The results from the experiments and sub-experiments answered the research question: "How can GAN architectures be designed to be able to create pixel art effectively?". The ResNet model showed results with no artifacts, adherence to the basic stylistic rules, and adequate color retention. The U-NET model was shown to generate quality pixel art with good color retention, albeit with more artifacts than the ResNet model. This thesis also reveals that adding additional loss functions, such as SSIM and perceptual losses, may sometimes help with artifacts. More fine-tuning of the models with additional loss functions is needed to determine if the functions steer the generation in the correct direction and produce better results. Lastly, the experiments showed that adding Self-Attention layers to the model resulted in worse results.

A common issue in the field of pixel art synthesis is the lack of large, high-quality datasets. This thesis contributes to this issue by suggesting relevant data sources and facilitating means to collect, parse, and process the data. This resulted in a dataset with 1,800 cartoon illustrations and 1,800 pixel art images, all under a creative commons license. In addition, this thesis showed that the collected dataset significantly improves the CycleGAN model's performance compared to the model trained on lower quantity and quality datasets. This answers the research question: "What impacts do the datasets' quality and quantity have on the generated artwork?".

The last significant contribution to the field is a more comprehensive way of evaluating the generated images. Most of the research in this field only evaluates results by visual inspection. This thesis shows that metrics such as KID and FID can be relevant to the quality of the generated images. In addition, consultation with domain experts can uncover more insights about the generated images than only statistical metrics and visual inspection.

This thesis also introduces a new color palette evaluation metric. This metric extracts dominant colors from images and scores how consistent the colors are with each other. The approach of comparing color palettes is not new. Still, to the author's knowledge, the customization and use of this metric in image-to-image translation tasks have not been done before. This metric applies to all style transfer tasks where the input and the output images should have the same dominant colors. The combination of visual inspection, statistical metrics, domain expert survey, and color palette evaluation sufficiently answers the research question: "How can the performance of the models be measured and evaluated?".

8.2. Future Work

This thesis tried to accomplish the main goal: "Create a generative adversarial network model capable of synthesizing high-quality pixel art.". Even though the results were not considered high quality, they were evaluated as of the quality of a beginner-level pixel artist. Further work should be done to improve the results and ultimately create high-quality pixel art.

More experiments with different combinations of loss functions and hyperparameters may be conducted to improve the models further. Most of the hyperparameters used for the experiments in this thesis were the default recommended parameters by Zhu et al. (2017), but other combinations may have a positive effect on training. In addition, complete color retention should be strived towards, and adding a color-related loss function may improve this. This may include directly comparing colors of input and output images or comparing color histograms or palettes.

Even though the model can create different pixel art variations of the same input image, as shown in Figure 7.8 (page 82), details may be lost when downscaling images before translation. Separate models can be trained on different datasets to alleviate this issue and make the model effectively translate images to several pixel art styles. These can be several datasets, each with a distinct pixel art style. One dataset may have low-resolution pixel art with only a few pixels, while another can be highly detailed high-resolution pixel art. Considerable effort must be taken to assemble such datasets, and many images must be hand-picked. The availability of open-source pixel art online is scarce, and it may be challenging to create several datasets, each consisting of thousands of images. If such datasets are collected, separate CycleGAN models can be used. This also opens the possibility of multimodal image-to-image translation using a single model, such as MUNIT (Huang et al., 2018). MUNIT can train on more than one dataset, generating more than one style for the input image. A suggestion for future work is to train the MUNIT model on several datasets consisting of more than one pixel art style.

Bibliography

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, pages 214–223. PMLR, 2017.

Trevor Avant and Kristi A Morgansen. Analytical bounds on the local Lipschitz constants of ReLU networks. *arXiv preprint arXiv:2104.14672*, 2021.

Hardik Bansal and Archit Rathore. Understanding and implementing cyclegan in tensorflow, 2017. URL `https://hardikbansal.github.io/CycleGANBlog/`.

Shane Barratt and Rishi Sharma. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018.

Eyal Betzalel, Coby Penso, Aviv Navon, and Ethan Fetaya. A study on the evaluation of generative models. *arXiv preprint arXiv:2206.10935*, 2022.

Nitin Bhatia and Vandana. Survey of nearest neighbor techniques. *arXiv preprint arXiv:1007.0085*, 2010.

Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. *arXiv preprint arXiv:1801.01401*, 2018.

Ali Borji. Pros and cons of GAN evaluation measures: New developments. *Computer Vision and Image Understanding*, 215:103329, 2022.

Min Jin Chong and David Forsyth. Effectively unbiased FID and Inception Score and where to find them. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6070–6079, 2020.

S. Colton and Geraint Wiggins. Computational creativity: The final frontier? *Frontiers in Artificial Intelligence and Applications*, 242:21–26, 01 2012. doi:10.3233/978-1-61499-098-7-21.

Andrew Cotter, Ohad Shamir, Nati Srebro, and Karthik Sridharan. Better mini-batch algorithms via accelerated gradient methods. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL `https://proceedings.neurips.cc/paper_files/paper/2011/file/b55ec28c52d5f6205684a473a2193564-Paper.pdf`.

Flávio Coutinho and Luiz Chaimowicz. On the challenges of generating pixel art character sprites using GANs. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 18(1):87–94, 2022. doi:10.1609/aiide.v18i1.21951.

Nan Cui. Applying gradient descent in convolutional neural networks. *Journal of Physics: Conference Series*, 1004:012027, 04 2018. doi:10.1088/1742-6596/1004/1/012027.

Emily L Denton, Soumith Chintala, Rob Fergus, and Arthur Szlam. Deep generative image models using a Laplacian pyramid of adversarial networks. *Advances in neural information processing systems*, 28, 2015.

Ahmed Elgammal, Bingchen Liu, Mohamed Elhoseiny, and Marian Mazzone. CAN: Creative adversarial networks, generating art by learning about styles and deviating from style norms. *arXiv preprint arXiv:1706.07068*, 2017.

Deniz Engin, Anil Genç, and Hazim Kemal Ekenel. Cycle-Dehaze: Enhanced CycleGAN for single image dehazing. In *Proceedings of the IEEE CVPR*, pages 825–833, 2018.

Xuan Fei, Liang Xiao, Yubao Sun, and Zhihui Wei. Perceptual image quality assessment based on structural similarity and visual masking. *Signal Processing: Image Communication*, 27(7):772–783, 2012. ISSN 0923-5965. doi:https://doi.org/10.1016/j.image.2012.04.005.

François Rozet. PIQA: PyTorch Image Quality Assessement, 2020. URL `https://pypi.org/project/piqa`.

Timothy Gerstner, Doug DeCarlo, Marc Alexa, Adam Finkelstein, Yotam Gingold, and Andrew Nealen. Pixelated image abstraction. In *NPAR 2012, Proceedings of the 10th International Symposium on Non-photorealistic Animation and Rendering*, June 2012.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL `https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Google. GAN structure, 2022a. URL `https://developers.google.com/machine-learning/gan/gan_structure`.

Google. GAN problems, 2022b. URL `https://developers.google.com/machine-learning/gan/problems`.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013. doi:10.1109/ICASSP.2013.6638947.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015. doi:10.1109/ICCV.2015.123.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 06 2016. doi:10.1109/CVPR.2016.90.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local Nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 172–189, 2018. doi:10.1007/978-3-030-01219-9_11.

Youyou Huang, Rencheng Song, Kuiwen Xu, Xiuzhu Ye, Chang Li, and Xun Chen. Deep learning-based inverse scattering with structural similarity loss functions. *IEEE Sensors Journal*, 21(4):4900–4907, 2021. doi:10.1109/JSEN.2020.3030321.

IBM. What is gradient descent?, 2022a. URL `https://www.ibm.com/topics/gradient-descent`.

IBM. Machine learning, 2022b. URL `https://www.ibm.com/topics/machine-learning`.

Tiffany C. Inglis and Craig S. Kaplan. Pixelating vector line art. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, page 21–28. Eurographics Association, 2012. ISBN 9783905673906.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on Machine Learning, PMLR 37*, pages 448–456, 02 2015. doi:10.48550/arXiv.1502.03167.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

Garrett M. Johnson and Mark D. Fairchild. A top down description of S-CIELAB and CIEDE2000. *Color Research & Application*, 28(6):425–435, 2003. doi:10.1002/col.10195.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF CVPR*, pages 4401–4410, 2019.

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 CVPR*, pages 8110–8119, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

Johannes Kopf, Ariel Shamir, and Pieter Peers. Content-adaptive image downscaling. *ACM Transactions on Graphics*, 32(6), 2013. ISSN 0730-0301. doi:10.1145/2508363.2508370.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

Hailan Kuang, Nan Huang, Shuchang Xu, and Shunpeng Du. A pixel image generation algorithm based on CycleGAN. In *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, volume 4, pages 476–480, 2021. doi:10.1109/IMCEC51613.2021.9482118.

Thitiporn Lertrusdachakul, Kanakarn Ruxpaitoon, and Kasem Thiptarajan. Color palette extraction by using modified k-means clustering. In *2019 7th International Electrical Engineering Congress (iEECON)*, pages 1–4, 2019. doi:10.1109/iEECON45304.2019.8938867.

Chuan Li and Michael Wand. Precomputed real-time texture synthesis with Markovian generative adversarial networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*, pages 702–716. Springer, 2016.

Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. *Advances in neural information processing systems*, 30, 2017.

Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying ReLU and initialization: Theory and numerical examples. *ArXiv*, 1903.06733, 2019.

Jingqin Lv and Jiangxiong Fang. A color distance model based on visual recognition. *Mathematical Problems in Engineering*, 2018, 2018. doi:10.1155/2018/4652526.

Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, and Zhen Wang. Multi-class generative adversarial networks with the l2 loss function. *ArXiv*, 1611.04076, 2016.

Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE ICCV*, pages 2794–2802, 2017.

Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *ICML*, pages 3481–3490. PMLR, 2018.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Shi Na, Liu Xumin, and Guan Yong. Research on k-means clustering algorithm: An improved k-means clustering algorithm. In *2010 Third IITSI*, pages 63–67. IEEE, 2010. doi:10.1109/IITSI.2010.74.

Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. doi:10.23915/distill.00003. URL `http://distill.pub/2016/deconv-checkerboard`.

A. Cengiz Öztireli and Markus Gross. Perceptually based downscaling of images. *ACM Transactions on Graphics*, 34(4), 2015. ISSN 0730-0301. doi:10.1145/2766891.

Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *ArXiv*, 1511.08458, 2015.

Qianqian Pan and Stephen Westland. Comparative evaluation of color differences between color palettes. In *Color and Imaging Conference*, volume 2018, pages 110–115. Society for Imaging Science and Technology, 2018.

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

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-NET: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

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.

Yunyi Shang and Hon-Cheng Wong. Automatic portrait image pixelization. *Computers Graphics*, 95:47–59, 2021. ISSN 0097-8493. doi:https://doi.org/10.1016/j.cag.2021.01.008. URL `https://www.sciencedirect.com/science/article/pii/S009784932100008X`.

Gaurav Sharma, Wencheng Wu, and Edul N Dalal. The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application*, 30(1):21–30, 2005. doi:10.1002/col.20070.

Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE CVPR*, pages 2107–2116, 2017.

Daniel Silber. *Pixel Art for Game Developers*. Taylor & Francis, 2015. ISBN 9781482252309. URL `https://books.google.no/books?id=n0zRrQEACAAJ`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014. doi:10.48550/arXiv.1409.1556.

Magnus Själander, Magnus Jahre, Gunnar Tufte, and Nico Reissmann. EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Ariawan Suwendi and Jan P Allebach. Nearest-neighbor and bilinear resampling factor estimation to detect blockiness or blurriness of an image. *Journal of Electronic Imaging*, 17(2):023005–023005, 2008.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. 07 2016. doi:10.48550/arXiv.1607.08022.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi:10.1109/TIP.2003.819861.

Bin Yang, Yaguo Lei, Feng Jia, Naipeng Li, and Zhaojun Du. A polynomial kernel induced distance metric to improve deep transfer learning for fault diagnosis of machines. *IEEE Transactions on Industrial Electronics*, 67(11):9747–9757, 2019.

Runtian Yang, Yudi Wang, Yiran Wang, and Michelle Xu. Application of neural network in pixel art creation: Bi-directional conversion between photo and pixel art with GAN base model. In *2022 2nd International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, pages 855–858, 2022. doi:10.1109/ICCECE54139.2022.9712735.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-Attention generative adversarial networks. In *ICML*, pages 7354–7363. PMLR, 2019.

Zijun Zhang. Improved ADAM optimizer for deep neural networks. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–2, 2018. doi:10.1109/IWQoS.2018.8624183.

Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57, 2017. doi:10.1109/TCI.2016.2644865.

Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212–3232, 2019. doi:10.1109/TNNLS.2018.2876865.

Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. U-NET++: A nested U-NET architecture for medical image segmentation. In Danail Stoyanov, Zeike Taylor, Gustavo Carneiro, Tanveer Syeda-Mahmood, Anne Martel, Lena Maier-Hein, João Manuel R.S. Tavares, Andrew Bradley, João Paulo Papa, Vasileios Belagiannis, Jacinto C. Nascimento, Zhi Lu, Sailesh Conjeti, Mehdi Moradi, Hayit Greenspan, and Anant Madabhushi, editors, *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 3–11, Cham, 2018. Springer International Publishing. ISBN 978-3-030-00889-5.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

Appendix

A. Domain Expert Survey Answers

Overall, how well do these artworks follow the rules of the pixel art style (regarding line art, color use, shading, dithering, aliasing, etc.)?

Expert 1: "The line art is uneven for the most part. Similar to basic outlines when starting a rough draft of an illustration. Anti-aliasing is used to smooth out the line art, however, there are many examples where its being applied incorrectly, for instance there are many examples where it is applied outside the outline of the illustration, giving a sort of glow effect. This would be understandable for a lightbulb but not for the majority of the images. The color pallette is kept to a minimum for most images, however theres not much shading going on in those examples, making them look flat. The examples where shading is being applied, a lot of them have a gradient effect applied, usually known as pillow shading and banding. This type of shading is common when beginner pixel artists. Dithering is not used in the vast majority of these examples, if at all."

Expert 2: "Starting with gradients which are not very welcome in the pixel art guidelines, especially when it comes to characters and smaller sprites (could be ok for a big background, for example). The shapes are still weird, but I believe it's getting there. The lines aren't great but not so bad for an AI. The artworks follow the rules mentioned with a rating of 5/10 overall. They're not bad, but far from good. They look mostly amateur-made. Shading 3/10, colors 5/10, and lines 8/10. The shadings are not being applied to every image. I understand there are different styles, but it could be way better overall. The colors are being too overused. You can see many different tones everywhere, there's no need for it. Anti-aliasing is kind of the same thing with the colors. It seems like it's using a black semi-transparent dot instead of a darker shade of that color."

Overall, are these artworks believable to be created by a human artist? Why, why not?

Expert 1: "The artworks do resemble human created pixel art, however, a very beginner level pixel artist. Even though pixel art techniques are visible, they are mostly incorrectly applied."

Expert 2: "It has a feel that the images were taken from the internet. The anti-aliasing and other stray pixels are inconsistent. It often looks like images that are shrunk down resemble pixel art. Besides that, they look good. They have a retro feel, and look like they are made by an entry-level pixel artist, but not bad."

Do any of the images stand out as high quality, and do any stand out as low quality?

Expert 1: "None of the images stand out as high quality. The vast majority are low quality pixel art. Examples 124, 128, 144, 160, 115, 73, among others can be classified as illustrations of an improving beginner pixel artist."

Expert 2: "Images 4, 7, and 16 look like a human's first attempts at pixel art. Images 18 and 20 are really good, and I love images 6, 82, 160, and 165. 35 of the remaining images look like a human being made them, styles considered."

Do you have any additional comments about the presented pixel art?

Expert 1: "Even though most images resemble beginner level pixel art, its impressive that even AI can create these types of artworks that show some sort of technique being applied. Pixel art is very tricky for beginners and can even be tricky for intermediate level pixel artists. Very impressed with the AI. I've seen other AI generated pixel art and you can tell right away they were computer generated. Your examples look more beginner level pixel artist."

Expert 2: "In order to get something more professional, I wouldn't recommend using any database since you have to dig down deep on the internet to find good stuff. Most of the pixel art posted online is absolutely amateur, not knowing any rules."